Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

ESTUDIO Y DESARROLLO DE UN PROTOTIPO DE APLICACIÓN MÓVIL PARA CONSULTA DE INFORMACIÓN SOBRE SEMANA SANTA

Alumno: David Galisteo Cantero

Tutor: Prof. D. José Ramón Balsas Almagro
Dpto: Informática

Junio, 2015

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don JOSÉ RAMÓN BALSAS ALMAGRO , tutor del Proyecto Fin de Carrera titulado: ESTUDIO Y DESARROLLO DE UN PROTOTIPO DE APLICACIÓN MÓVIL PARA CONSULTA DE INFORMACIÓN SOBRE SEMANA SANTA, que presenta DAVID GALISTEO CANTERO, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, JUNIO de 2015

El alumno: Los tutores:

DAVID GALISTEO CANTERO JOSÉ RAMÓN BLASAS ALMAGRO

1.- Introducción

1.1.- Propósito

Actualmente, el desarrollo de software está muy enfocado a aplicaciones móviles, no hay más que ver el número de aplicaciones que hay en las diferentes 'stores'. Es la manera más fácil de llegar a los usuarios, ¿qué persona no tiene un teléfono móvil en su bolsillo? Estas personas demandan multitud de servicios, como consultar información del tiempo, hablar con sus conocidos, etc...

Así, el propósito este proyecto es la creación de una aplicación móvil para el sistema operativo iOS[1], que presente información sobre la Semana Santa, procesiones, cofradías, eventos, recorridos…

Se ha elegido este proyecto ya que el alumno tenía en mente la elaboración de un proyecto similar y el objetivo de aprender a desarrollar aplicaciones en iOS.

1.2.- Objetivos

- Estudio del lenguaje de programación SWIFT[2] y de su entorno de desarrollo para la creación de aplicaciones móviles para la plataforma iOS.
- Analizar y diseñar un sistema informático para la consulta de información sobre la Semana Santa, tratándose de una arquitectura Cliente (iOS SWIFT) - Servidor (REST[3] PHP[4]).
- Implementar un prototipo del sistema utilizando la tecnología SWIFT accediendo a un servidor REST a través de Internet.

1.3.- Metodología

Par llevar a cabo el proyecto usaremos una metodología, la cual se basa en:

- Estudio de los requerimientos del sistema a desarrollar: Se analizarán los diferentes requisitos funcionales y no funcionales a implementar en la aplicación.

[1] iOS es el sistema operativo para dispositivos móviles creado por Apple.
[2] Swift es el nuevo lenguaje de programación para iOS, pretende sustituir a Objective-C
[3] REST es un estilo de arquitectura de software que usa el protocolo http para intercambiar información
[4] PHP es un lenguaje de programación del lado del servidor

- Recopilación bibliográfica sobre la temática y tecnologías relacionadas: Se investigarán detenidamente las diferentes tecnologías a usar, así como la plataforma de desarrollo objetivo.
- Estimación temporal y de costes del desarrollo: Se creará una planificación temporal para estimar la duración del proyecto, así como un plan de costes para el estudio de su viabilidad. Este punto es vital para decidir la viabilidad del proyecto.
- Diseño del sistema utilizando una metodología de orientación a objetos utilizando el patrón de diseño Modelo-Vista-Controlador[5]: Este patrón será utilizado tanto en la aplicación cliente como en el servidor.
- Implementación del prototipo del sistema: Se implementará el prototipo usando las tecnologías propuestas.

1.4.- Estructura del documento

Para facilitar la lectura de este documento, se proporcionan al lector los distintos puntos en los que se dividirá este texto:

1. En primer lugar, se llevará a cabo una fase de análisis del problema, donde veremos las necesidades a cubrir, análisis de requisitos, análisis de tareas, de costes… etc
2. A continuación, una fase de diseño, en el que se irá dando forma al sistema que se tiene en mente, usando diagramas de clases, de secuencia…, además, tendremos un apartado para el diseño de la interfaz, donde veremos Wireframes y Storyboards.
3. Después tenemos la etapa de implementación, donde se especificará la arquitectura utilizada y se comentarán los aspectos más importantes en cuanto a programación que se han ido encontrando a lo largo de este proyecto.
4. Por último, tendremos un apartado de conclusiones donde el alumno expondrá sus valoraciones finales sobre el proyecto, consecución de objetivos, futuras mejoras… etc

2.- Análisis

[5] Modelo-Vista-Controlador es un patrón de arquitectura de Software

2.1.- Análisis preliminar

Para empezar, se parte de la necesidad de consultar información sobre la Semana Santa en la palma de la mano. Actualmente existen diferentes alternativas en cuanto a aplicaciones Android[6] se refiere, pero muchas menos para plataformas iOS, de ahí que se pretenda realizar un prototipo para este ecosistema.

Tradicionalmente, la información sobre procesiones, cofradías y demás entes involucrados en la Semana Santa, se ha plasmado en periódicos locales, folletos creados para la ocasión, etc. También se ha transmitido esta información, en la mayoría de los casos, por el 'boca a boca', y, en los últimos años, en alguna web creada para estos eventos.

Con estos métodos surgían algunos problemas, qué pasa si se retrasa una procesión, si se cancela por lluvia, si se cambia un recorrido… etc. Con este prototipo se pretende eliminar estos problemas teniendo la información en tiempo real en la palma de la mano.

De igual forma, estas aplicaciones se centran en una localidad o ciudad en concreto, si hacemos una búsqueda, en fecha del comienzo de esta memoria, no hay ninguna para mi 'localidad objetivo', en este caso será Baena, he decidido crear el prototipo centrándome en datos de ejemplo de esta ciudad. Sin embargo, se pretende que la aplicación pueda ofrecer datos de cualquier localidad en el futuro.

Nota: A fecha de fin de esta memoria, SI hay una aplicación disponible en la App Store que soluciona, **en parte**, la problemática expuesta.

Respecto a funcionalidad, casi todas estas aplicaciones cubren las mismas funcionalidades, consulta de horarios, visualización de recorridos… pero muchas no implementan algo que comenté anteriormente, la consulta en tiempo real, o los dos colores en el recorrido simbolizando la ida y vuelta del mismo, en este prototipo se pretende que la información plasmada sea lo más actualizada posible.

Por poner un ejemplo, en la app oficial de la Semana Santa de Baena, cuyo enlace es https://itunes.apple.com/es/app/semana-santa-baena-oficial/id971638358?mt=8 , podemos observar, por ejemplo, en el mapa de la Figura 1, que en un recorrido cíclico, no podemos distinguir por dónde se desfila primero:

[6] Sistema operativo de Google para dispositivos móviles

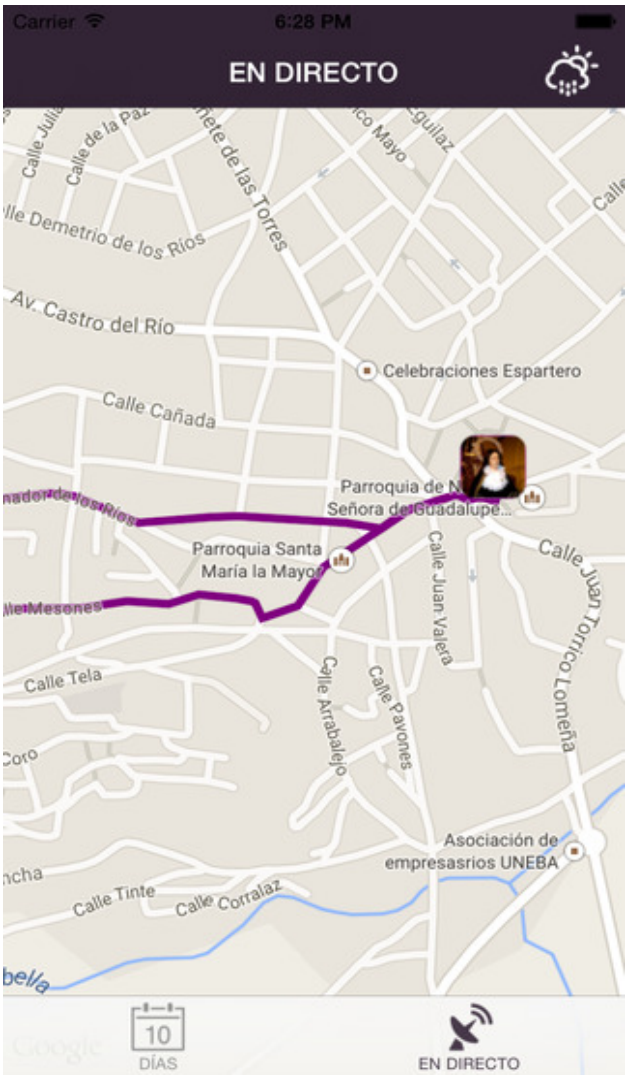

Figura 1

Este mismo problema ocurre en todas las apps similares. Sin embargo, uno de sus puntos fuertes es que proporciona información meteorológica.

Respecto al panorama en el desarrollo de aplicaciones para iOS, Apple ha lanzado recientemente el lenguaje de programación SWIFT, que busca mayor simplicidad y elegancia a la hora de programar, anteriormente, con Objective-C, se hacía compleja la programación de Apps, sobre todo para los más novatos, sin embargo, este nuevo lenguaje viene a intentar solventar ese problema.

Si hablamos del número de Apps, es cierto que para Android existe más cantidad que para iOS, debido al hecho de que todos podemos crear una app Android con cualquier arquitectura hardware. Sin embargo, una de las cosas que no me gusta de esta plataforma es su gran fragmentación, habiendo, al día de hoy, bastantes dispositivos con una versión 2.* de Android. En contraposición, Apple ha

solventado en gran medida ese problema, teniendo unos niveles de fragmentación muy por debajo de los de Android.

Respecto al futuro, creo que iOS crecerá en cuanto a número de desarrolladores, ya que, en el día que se escriben estas líneas, Apple acaba de lanzar la versión 2 de SWIFT, que será Open Source, esto provocará que en un futuro bastante cercano podamos desarrollar Apps con este lenguaje usando sistemas operativos como Linux o Windows.

2.3.- Propuesta de solución

A partir del análisis realizado anteriormente, se propone el desarrollo de una aplicación nativa para iOS que permita la consulta de horarios de procesiones, eventos, información de cofradías…, llegando un punto más allá que las soluciones actuales, como es, por ejemplo, permitir la visualización de recorrido de ida y vuelta de la procesión con colores distintos, o la realización de un diseño orientado a poder obtener información de más de una ciudad.

Se utilizará una arquitectura cliente servidor basada en servicios web para intercambio de información entre cliente y servidor en formato JSON.

Se utilizará el framework ‘Slim’ de PHP para crear el API REST, todo sobre un servidor web ‘Apache’, además, se usará ‘Eloquent’, el ORM del framework de PHP ‘Laravel’, para interactuar con el SGBD, que será ‘MySQL’.

Respecto al cliente, usaremos SWIFT para el desarrollo nativo en iOS.

2.2.- Requerimientos del sistema

Comencemos describiendo qué es un requisito. Como se dice en [1], un requisito es una condición o capacidad exigida por el usuario para solucionar un problema o alcanzar un objetivo. En esta fase se intentarán plasmar los requisitos funcionales y no funcionales que deberá satisfacer nuestro sistema para solucionar un problema.

En principio, los requisitos para nuestra aplicación serían:

- Funcionales
    - El usuario podrá consultar el horario de comienzo de una procesión, entendemos procesión como el “desfile” de varias hermandades con sus respectivas imágenes (Representaciones de Santos).

- El usuario podrá ver el recorrido que tendrá la procesión, que se visualizará a través de un mapa de Google, mostrará el recorrido con dos colores, uno simbolizando el recorrido de ida, y otro el de vuelta.
- El usuario verá la fecha y hora de los distintos eventos que se realizarán, un ejemplo de evento podría ser: “Besamanos de la hermandad Ntro. Padre Jesús Nazareno en la iglesia de San Francisco, el día X a las Y horas”, se visualizará el nombre, lugar (texto y mapa), y hora del mismo.
- El usuario dispondrá de información sobre cofradías para su consulta, se visualizará información sobre la cofradía y las distintas hermandades que la componen (una cofradía se compone de una o varias hermandades).
- El usuario podrá consultar información sobre las distintas hermandades, como el nombre, año de creación y número de hermanos.

● No funcionales

- El sistema permitirá la conexión de aplicaciones cliente de cualquier plataforma, Android, web...
- La información ha de estar actualizada (cancelación por lluvia, cambio de hora…)
- Funcionalidad en todos los contextos de uso, por lo que se hace necesaria una interfaz con colores legibles.
- Disponibilidad de datos sin conexión (previamente habrán sido descargados)

2.4.- Planificación de tareas

Una vez tenemos los requisitos, intentaremos descomponer el proyecto en diferentes tareas. Puesto que el TFG consta de 12 créditos, unas 300 horas de trabajo, vamos a ajustar todas las tareas a este lapso de tiempo. A continuación, en la Figura 2, se puede observar a simple vista la relación de tareas, en caso de no distinguirlo bien, se ruega al lector remitirse al apartado “Diagrama de tareas”, de los archivos del proyecto:

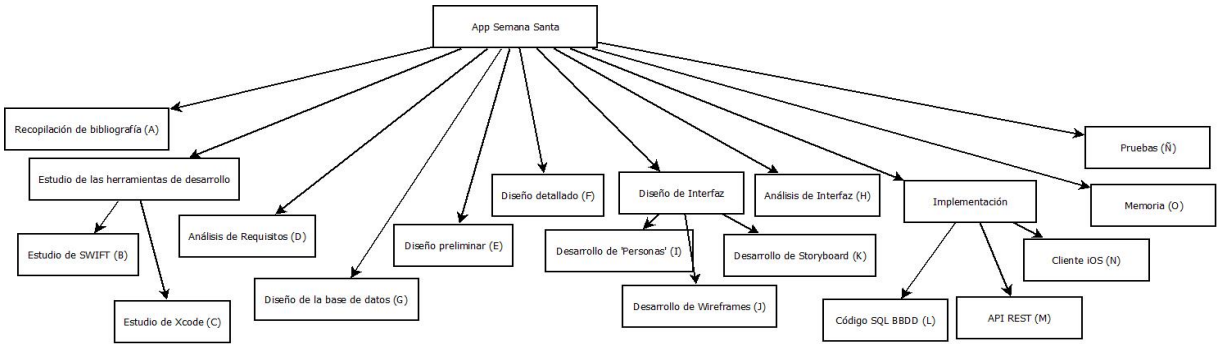

Figura 2

Tareas

1. Recopilación de bibliografía (A)
2. Estudio de las herramientas de desarrollo
    a. Estudio de los aspectos básicos de SWIFT (B)
    b. Estudio de Xcode (C)
3. Análisis de requisitos (D)
4. Diseño preliminar (E)
5. Diseño detallado (F)
6. Diseño de la base de datos (G)
7. Análisis de la interfaz. Estudio y propuesta de reglas UX para móviles (H)
8. Diseño de la interfaz
    a. Desarrollo de 'Personas' (I)
    b. Desarrollo de Wireframes (J)
    c. Desarrollo de Storyboard (K)
9. Implementación
    a. Código SQL para creación de base de datos (L)
    b. Implementación de API REST con PHP y SLIM (M)
    c. Implementación cliente iOS (N)
10. Pruebas (Ñ)
11. Elaboración de la memoria (O)

A continuación se plasman la duración y responsables de cada tarea:

Tabla Tareas - Encargado - Duración

Para empezar, se buscará bibliografía que trate sobre el tema que vamos a estudiar (libros, material en web…) , a continuación se pasará a estudiar las distintas

herramientas que usaremos para nuestro proyecto, como es el lenguaje SWIFT y el IDE de Apple Xcode.

A continuación comenzamos con el proceso de análisis y diseño del sistema, llevando a cabo un análisis de requisitos (recopilación de objetivos que deberá cumplir el sistema) , un diseño preliminar (elaboración de modelo de dominio) y otro detallado (elaboración de diagrama de clases y secuencia), para, posteriormente, pasar a describir el modelo de base de datos (diseño de tablas, atributos, relaciones…).

Posteriormente se llevará a cabo un análisis y diseño de la interfaz que tendrá nuestra aplicación, para ello se desarrollarán wireframes (dibujo esquemático de las vistas de la app) y un storyboard (guión gráfico de la app), además de llevar a cabo el método de 'Personas', consistente en representar a una o varias personas y su posible interacción con el sistema para encontrar nuevos requisitos para la interfaz.

Después se llevará a cabo la implementación, por un lado, la aplicación cliente en SWIFT, por otro, el API en el servidor con PHP.

Por último, se harán las pruebas, con las que se pretende ver el comportamiento del sistema enfrentándolo a diferentes situaciones y se llevará a cabo el desarrollo de la memoria.

A continuación se muestra, como resumen en la tabla 1, todas las tareas, con su duración, encargado, y si tienen o no tareas previas.

TAREA	DURACIÓN (horas)	PREVIAS	ENCARGADO
Recopilación de Bibliografía (A)	4	-	Analista
Estudio de aspectos básicos de SWIFT (B)	8	A	Analista
Estudio de Xcode (C)	8	A	Analista
Análisis de Requisitos (D)	8	A	Analista
Diseño Preliminar (E)	8	D	Analista
Diseño Detallado (F)	16	E	Analista

Diseño de la base de datos (G)	16	F	Analista
Análisis de Interfaz. Estudio y propuesta de reglas UX para móviles. (H)	8	D	Diseñador Experto UX
Desarrollo del método 'Personas' (I)	4	D	Diseñador Experto UX
Desarrollo de Wireframes (J)	8	I	Diseñador Experto UX
Desarrollo de Storyboards (K)	8	J	Diseñador Experto UX
Código SQL para BBDD (L)	4	G, J	Programador Senior
Implementación API REST (M)	40	L	Programador Senior
Implementación cliente iOS (N)	140	M	Programador Senior
Pruebas (Ñ)	8	N	Programador Senior
Elaboración de memoria (O)	16	N	Analista
TOTAL horas	304		

Tabla 1

En total serían unas 304 horas, que, traducido a meses, si se marcó la fecha de inicio el día 28/01/2015, se finalizaría el 13/03/2015. Aproximadamente 1 mes y medio. Suponiendo un total de 8 horas diarias, y teniendo en cuenta fines de semana y festivos.

Ahora se puede observar a simple vista el gráfico de Gantt del proyecto en la Figura 3, para verlo con más detalle, se ruega al lector que se remita al archivo adjunto 'TFG Gantt.mpp'.

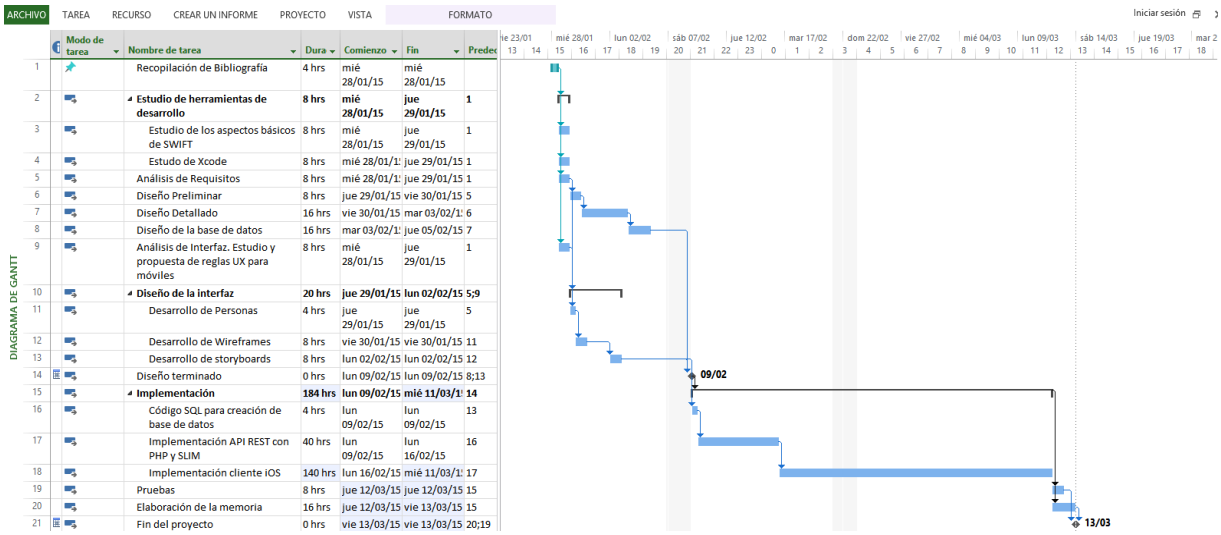

Figura 3

Como se puede comprobar, lo primero que hacemos es encargarnos de la recopilación de bibliografía, posteriormente, del estudio de herramientas de desarrollo. Una vez completadas estas tareas, comenzamos con las fases de análisis y diseño, así como del análisis y diseño de la interfaz. Marcamos un hito cuanto tenemos todo el diseño terminado, tanto de software como de interfaz. En ese momento, comenzamos con la implementación, por último, llevamos a cabo la fase de pruebas y elaboración de memoria, marcando, también, un hito al final del proyecto.

2.5.- Estudio de viabilidad

Una vez hemos hecho la planificación de tareas, vamos a realizar un estudio de viabilidad, con el propósito de descubrir si nuestro proyecto es interesante desde un punto de vista económico.

A continuación se desglosa el coste total que conlleva la realización del proyecto:

- Gastos de material:
  - Conexión a internet: Con la planificación del apartado anterior, se estima que las horas de trabajo se reparten aproximadamente en 2 meses: 2 mes x 42,23 €/mes = **84,46 €**
  - Equipo Mac para desarrollo, en este caso ya está comprado, por lo que vamos a amortizarlo. En principio se estimó la

vida del equipo en 3 años, tuvo un coste de 2.200 €, por lo que saldría a 2200 € / 36 mes = 61,11 €. Ahora: 2 mes x 61,11 € = **122,22 €**

- Equipo con sistema iOS para pruebas, en este caso se selecciona un iPad Mini 2, ya comprado, por 289 €, con una vida útil estimada de 2 años: 289 € / 24 mes = 12,04 €. 2 mes x 12,04 € = **24,08 €**
- Licencia desarrollador Apple Store: 99$ / año. Aprox **87 €** con cambio de divisa.
- Total gastos de material: **317,76 €**

Podemos apreciar un resumen en la tabla 2:

Gasto	**Total**
Conexión internet	84,46 €
Equipo Mac	122,22 €
Equipo iOS	24, 08 €
Licencia desarrollador Apple	87 €
Total	**317,76 €**

Tabla 2

- Gastos en nóminas:
  - El sueldo por hora del analista y analista programador los obtenemos de [2].
  - Analista: 2.555,29 € mensuales
    - Suponiendo: 8 horas x 5 dias x 4 semanas = 80 horas /semana
    - 2.555,29 / 80 = 31,94 € / h
    - Trabaja 84 h x 31,94 € = **2.682,96 €**
  - Programador senior: 2.166,79 € mensuales
    - Suponiendo: 2.166,79 / 80 = 27,08 / h
    - Trabaja 192 h x 27,08 € = **5.199,36 €**

- Diseñador UX: 14.952,54 € anuales / 14 pagas = 1068,03 €
  - El suelo para el diseñador lo obtenemos de [3].
  - 1068,03 / 80 = 13,35 € / h
  - Trabaja 28 h x 13,35 € = **373,8 €**
- También necesitamos un comercial/captador/visitante que visite a los posibles clientes:
  - El sueldo para el comercial lo obtenemos de [4].
  - 11010 € año / 14 pagas = 786,42 €
  - Este trabajador estará activo durante 10 meses (sin pagas extra)
  - 768,42 € x 10 pagas = **7684,20 €**
- Total nóminas: **15.940,32 €**

- **Gasto TOTAL = 16.258,08 €**

Podemos apreciar un resumen en la tabla 3:

Gasto en sueldos	**Total**
Analista	**2.682,96 €**
Programador Senior	**5.199,36 €**
Diseñador UX	**373,8 €**
Comercial	**7684,20 €**
Total	**15.940,32 €**

Tabla 3

A continuación podemos ver un resumen total de los diferentes gastos en la tabla 4:

Gasto	**Total**
Gasto en material	**317,76 €**
Gasto en sueldos	**15.940,32 €**

Total	**16.258,08 €**

Tabla 4

Plan de negocio

Se pretende vender esta aplicación a los ayuntamientos de los municipios, normalmente, el precio de una aplicación móvil nativa, hecha por encargo, puede rondar los varios miles de euros.

Sabiendo los posibles problemas que puede ocasionar la venta de una aplicación personalizada a pequeños o medianos consistorios debido a la situación económica actual, se ha decidido crear una aplicación general, de tal forma que podamos vender una suscripción anual a estas entidades (SaaS), y, simplemente, dándolas de “alta” en nuestro sistema, puedan ofrecer el servicio a los ciudadanos.

Así, podemos observar en la tabla 5 un rango de precios según número de habitantes:

Habitantes	**Precio**
20k - ∞ (Nivel 5)	300 €
15k - 20k (Nivel 4)	200 €
10k - 15k (Nivel 3)	150 €
5k - 10k (Nivel 2)	100 €
1k - 5k (Nivel 1)	50 €

Tabla 5

Como se puede observar, es un precio casi irrisorio para un consistorio, aún con la situación económica actual.

En cuanto a la obtención de clientes en este año, se pretende, teniendo en cuenta las fechas de la Semana Santa, obtener las cantidades especificadas en la tabla 6:

Meses	**Clientes**
Enero – Marzo	20
Abril – Septiembre	5

Septiembre – Diciembre	13

Tabla 6

En los meses previos a esta semana, se prevé una alta obtención de clientes, sin embargo, en los meses siguientes en los de verano, se espera una poca demanda. Finalmente, conforme vayamos acercándonos de nuevo a la semana clave, las ventas irían creciendo.

En total serían 38 clientes, supongamos que estos 38 se dividen según niveles tal y como se muestra en la tabla 7:

Nivel	**Cantidad**
5	3
4	8
3	10
2	15
1	2
Total	38

Tabla 7

Así, tendríamos unos ingresos de:

- 3 x 300 € = 900 €
- 8 x 200 € = 1600 €
- 10 x 150 € = 1500 €
- 15 x 100 € = 1500 €
- 2 x 50 € = 100 €
- **Total = 5600 €**

Como vemos, en el primer año NO cubrimos lo invertido, usamos la fórmula 1:

$$Ingresos - Gastos = Beneficio \quad (1)$$

5600 € - 12774,94 € = -7174,94 €

Si hablamos de los años posteriores, ya NO tendríamos los gastos de material, y supongamos que los gastos del desarrollo van destinados al mantenimiento, por lo que tendríamos unos gastos anuales de **12457,18 € (Solo nóminas).**

Así, si mantenemos el 90% de nuestros clientes, y volvemos a conseguir, como mínimo, el mismo número que el año anterior, el beneficio sería calculado según la fórmula 2:

$$Ingresos\ Nuevos\ clientes\ +\ (Ingresos\ Antiguos\ clientes\ -\ Ingreso\ Clientes\ perdidos)\ -\ Gastos\ =\ Beneficio \quad (2)$$

5600 € + (5600 € - 560 €) - 12457,18 € = -1.817,18 €

Y así sucesivamente los siguientes años, como se puede comprobar, hasta el tercer año, obteniendo una cantidad similar de clientes, no empezamos a obtener beneficios con este plan de negocio.

2.6.- Modelo de dominio

Una vez tenemos nuestro plan de tareas y análisis de costes, empezamos con el Modelo de dominio. En esta fase del análisis, identificaremos los conceptos del mundo real, relacionados con la Semana Santa, que intervendrán en nuestro sistema, además de crear un diagrama de modelo del dominio.

Listado de conceptos

- Cofradía
- Hermandad
- Evento
- Procesión
- Recorrido
- Día

Como aclaración, el concepto 'Día' se refiere a los distintos días de la semana: Lunes Santo, Martes Santo...

A continuación en la figura 4 se puede ver una representación del diagrama del modelo de dominio, para verlo más en detalle, remitirse al fichero 'modelo-dominio.pdf'.

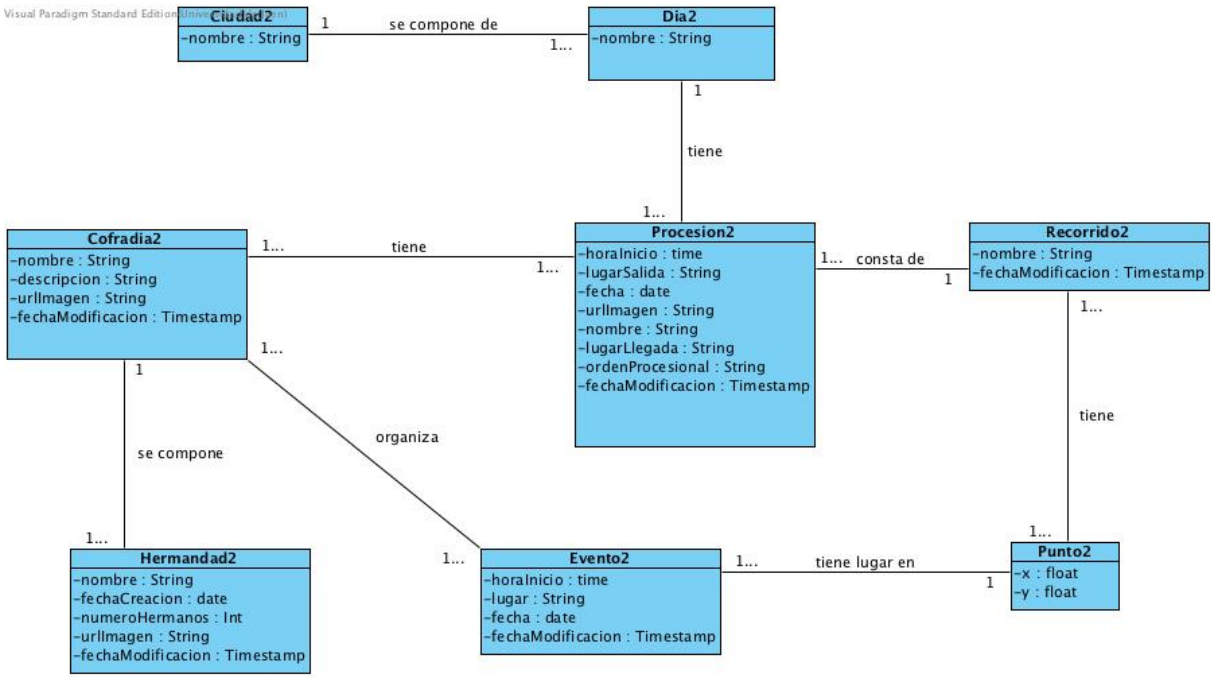

Figura 4

Pasemos a explicar los aspectos más relevantes del diagrama. El “eje central” del diseño serían las procesiones, contendrían la información más importante y a partir de esta entidad se construirían las demás.

Una ciudad tendría varios días (Lunes Santo, Martes Santo..), a su vez, en cada día, puede haber una o varias procesiones, en la que desfilan una o varias cofradías, toda procesión tiene un recorrido, que puede ser compartido por varias procesiones. Un recorrido se compone de “puntos”, representados por su latitud y longitud. Una cofradía organiza eventos, que se llevan a cabo en un lugar y hora determinados, el atributo lugar representaría una cadena, por ejemplo “Iglesia de San Francisco”, además, podemos observar que hay una relación a la entidad ‘Punto’, necesaria para dibujar un mapa y geolocalizar la lugar en cuestión. Por último, una cofradía se compone de 1 o varias hermandades.

Veamos ahora los casos de uso más representativos para los requisitos anteriormente obtenidos.

2.7.- Casos de Uso

Los casos de uso describen una serie de pasos que hay que realizar para llevar a cabo un proceso según se especifica en [5], ayudan a determinar la funcionalidad y características del software desde la perspectiva del usuario.

Los casos de uso que propongo para representar requisitos son:

- Consultar el recorrido de una procesión
- Consultar la información de una cofradía

A continuación se muestran en detalle:

Consultar recorrido procesión

El primer caso de uso se representa según la figura 5:

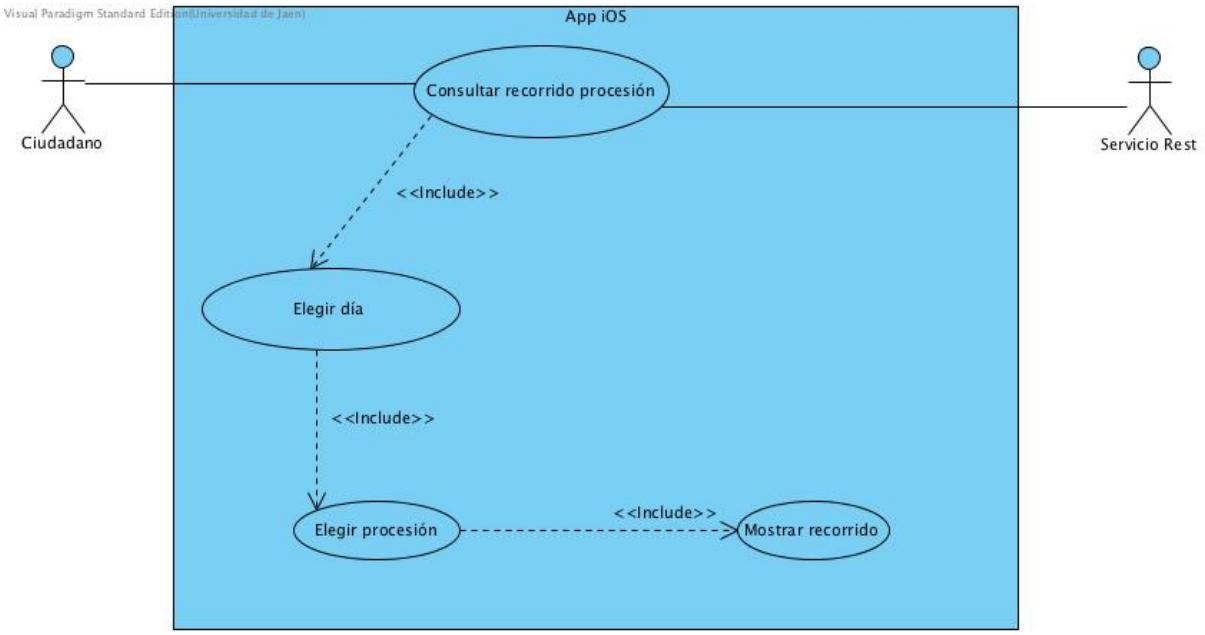

Figura 5

Para poder consultar el recorrido de una procesión, el ciudadano deberá elegir el día en el que se celebra la procesión, en caso de haber más de una el mismo día, deberá seleccionar la que desee, una vez hecho esto, podrá seleccionar la opción de mostrar recorrido.

Como se puede observar, hay un segundo actor que sería el servicio REST, encargado de proveer los datos para su consulta. La tabla 8 representa el caso de uso:

Caso de uso	Consultar recorrido procesión
Actor primario	Ciudadano
Sistema	Aplicación iOS
Participantes	Ciudadano
Nivel	Objetivo usuario
Condición previa	El dispositivo debe tener conexión a internet o, si no tiene, haber descargado los datos previamente
Operaciones básicas	
1	Seleccionar pestaña de procesiones
2	Seleccionar día de la procesión
3	Pulsar botón “Ver recorrido”
Alternativas	
2.A	¿Hay más de una procesión el mismo día?
2.A.1	Sí, Seleccionar procesión entre todas las de ese día y continuar
2.A.2	No, continuar proceso, se selecciona la única de ese dia

Tabla 8

Consultar la información de una cofradía

Podemos observar la representación gráfica del caso de uso en la Figura 6:

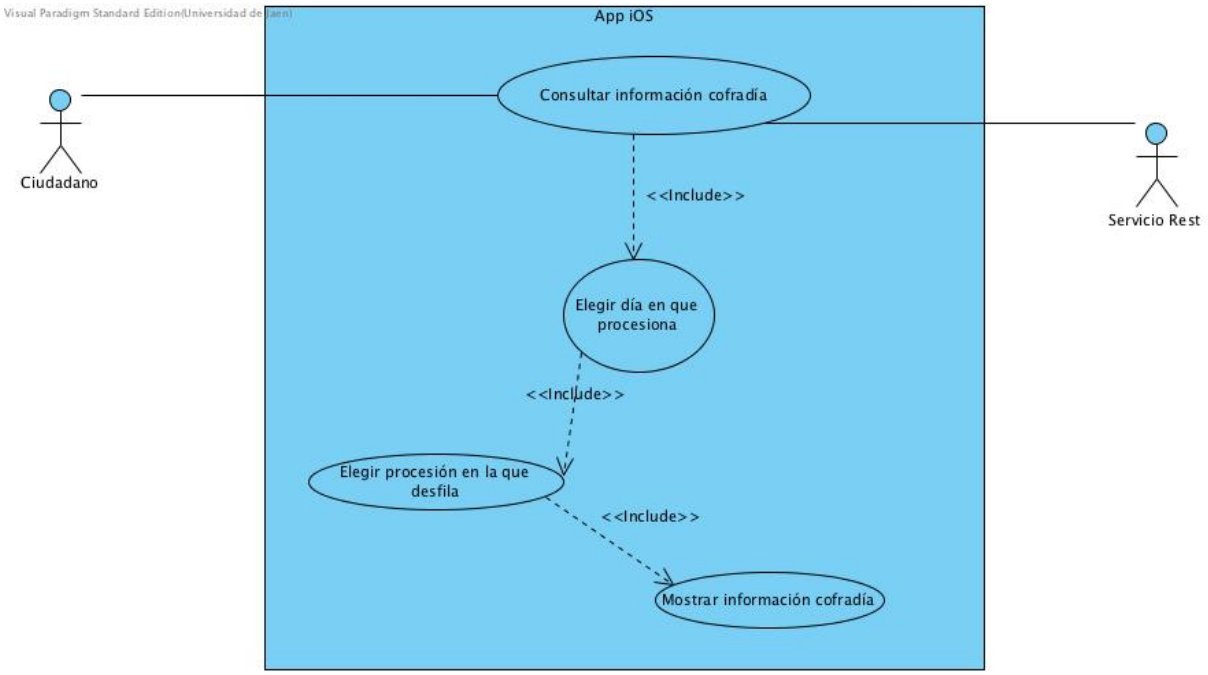

Figura 6

Para consultar la información de una cofradía, el ciudadano deberá elegir la procesión donde participa la cofradía en cuestión, a partir de ahí simplemente deberá seleccionar la cofradía deseada y se le mostrará la información.

También podemos observar la participación del servicio REST que proporciona los datos al cliente.

Se representa textualmente el caso de uso en la tabla 9

Caso de uso	Consultar información cofradía
Actor primario	Ciudadano
Sistema	Aplicación iOS
Participantes	Ciudadano
Nivel	Objetivo usuario
Condición previa	El dispositivo debe tener conexión a internet o, si no tiene, haber descargado los datos previamente

Operaciones básicas	
1	Seleccionar pestaña de procesiones
2	Seleccionar día de la procesión
3	Seleccionar cofradía
Alternativas	
2.A	¿Hay más de una procesión el mismo día?
2.A.1	Sí, Seleccionar procesión y continuar
2.A.2	No, continuar proceso, se selecciona la única de ese dia

Tabla 9

2.8.- Análisis de la interfaz

Una vez tenemos el modelo de dominio y algunos casos de uso, llega la hora de prestar un poco de atención a la interfaz, veamos algunos elementos interesantes al respecto.

Se pretende un diseño óptimo para la gran mayoría de personas, por lo que se tiende a una interfaz fácil y sencilla, el usuario debería “disfrutar” usando nuestra aplicación. El diseño no es simplemente hacer una interfaz bonita, sino que hay que estudiar distintos temas de experiencia de usuario para conseguir buenos resultados en este apartado.

En mi opinión, existen algunos principios básicos de diseño para móviles, vamos a ver cuales son y cómo encajan en nuestra aplicación:

- Apreciar el ingreso de datos → Se ha de requerir el mínimo esfuerzo por parte del usuario a la hora de insertar datos, así, en nuestra aplicación, solo tenemos que seleccionar objetos, evitando la entrada de datos.

- Nuestra App no es el centro del universo → Nuestra app no será, probablemente, la más importante dentro del teléfono del usuario, en este sentido, debería hacer "interesantes" esos pequeños momentos de uso por parte del mismo, ya que este no va a estar constantemente consultando información, así, también debemos evitar el continuo envío de notificaciones (a menos que sea importante), ya que podemos cansar al usuario. En definitiva, ha de resolver algo puntual en un momento puntual.
- Debe funcionar en todos los contextos → Hemos de tener en cuenta que no siempre tendremos buena conexión a internet, buena iluminación, etc… Pensar en los contextos es pensar en tecnología móvil, puesto que las apps han de ser útiles en todos ellos. En este sentido, nuestra app contará (en el futuro) con un sistema para consultar información sin necesidad de conexión a internet (guardando datos consultados previamente), respecto a la iluminación, se han seleccionado colores que hacen que se pueda "leer" la información con distintos niveles de iluminación.
- Los móviles son "Smart" → En este sentido me refiero a que debemos utilizar los sensores que se ponen a nuestra disposición, pongamos un ejemplo, si establezco una reunión en el calendario para mañana a las 10, ¿Por qué no debería nuestra app ser capaz de poner en silencio el teléfono automáticamente a esa hora, sabiendo que he llegado a la cita? Tiene toda la información para hacerlo. Respecto a nuestra aplicación, a la hora de visualizar el mapa, mostrará (en el futuro) un punto que simbolizará la situación actual del usuario y, además, una ruta para llegar al recorrido de la procesión.

Otra de las cosas muy interesantes respecto a UX, sería identificar el recorrido de ida y el de vuelta a la hora de mostrar el mapa, así, se identificarán con colores distintos, verde para ida y rojo para vuelta. Esto se hace necesario ya que, en localidades pequeñas, un recorrido puede ser cíclico, empezando y finalizando en el mismo punto. También se mostraría una pequeña leyenda que indicaría el significado de los colores.

Lo ideal a partir de aquí sería realizar 4 pasos, algunos de los cuales trataremos en este documento:

1. Identificación de ideas
2. Wireframes

3. MockUp
4. Prototipo

3.- Diseño

Ya hemos finalizado la fase del análisis, ahora nos centraremos en la del diseño, en la que empezaremos a dar forma al sistema que queremos construir. Empezaremos con el diagrama E-R, seguido del diagrama de clases y diagramas de secuencia. Cabe destacar que, la mayor parte del contenido de la parte de Diseño ha sido consultada en [6].

3.1.- Diagrama Entidad-Relación

En el diagrama de Entidad - Relación se muestran las distintas relaciones que tienen las entidades de nuestro proyecto, haciendo posible, a partir de aquí, crear nuestra base de datos.

En este apartado trataremos aspectos relacionados con el diseño de la base de datos, se ha optado por un SGBD tipo SQL, ya que las soluciones NO SQL no se ajustaban a nuestro propósito.

Para la realización de este diagrama hemos de identificar las entidades que intervienen en nuestro sistema, podemos crearnos una primera idea a partir del diagrama de modelo del dominio. Así, identificamos las siguientes entidades:

- Procesión
- Cofradía
- Hermandad
- Evento
- Punto
- Recorrido
- Día

A continuación se muestra con detalle el diagrama, en la Figura 7:

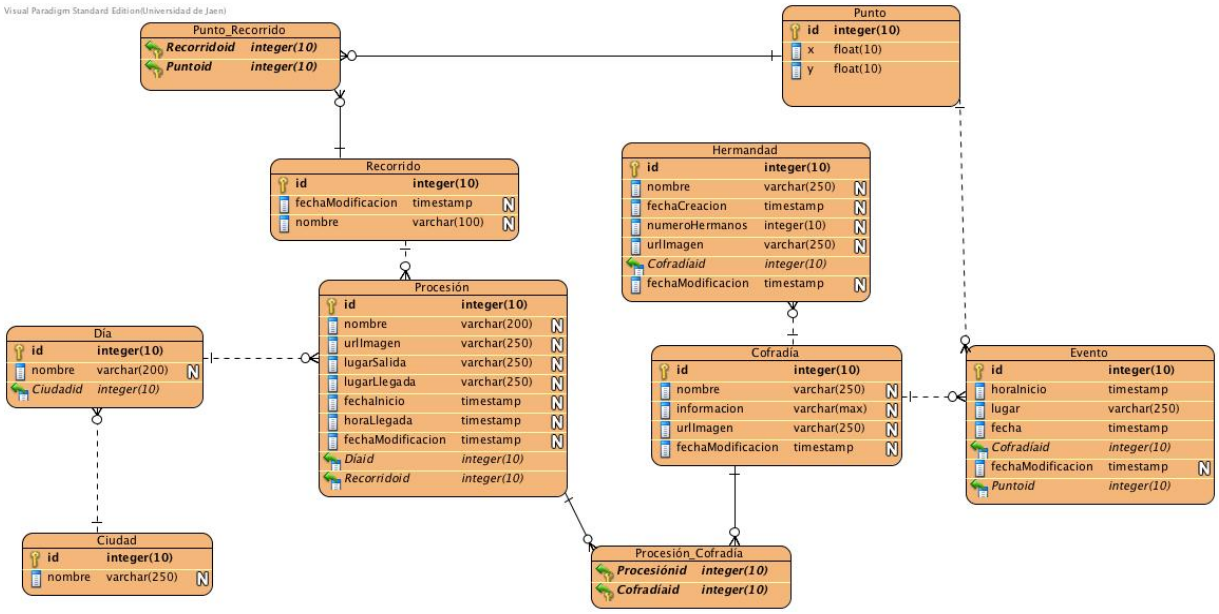

Figura 7

Como se puede observar en el diagrama, hay relaciones de N → N que se representan con una tabla adicional, es el caso de las relaciones entre 'Punto' y 'Recorrido', ya que un recorrido se compone de muchos puntos y un punto puede pertenecer a más de un recorrido, igual pasa con 'Procesion' y 'Cofradía', en una Procesión puede participar más de una cofradía, y una cofradía puede participar en más de una procesión, este último caso no es muy común, pero puede ocurrir.

Respecto a las relaciones de 1 → N , en la tabla "destino", se hace referencia con una clave foránea al id de una tupla de la tabla "origen".

Normalización

Como bien sabemos y según [7], la normalización es un proceso consistente en imponer a las tablas ciertas restricciones, aplicando una serie de transformaciones, para evitar problemas como información redundante, pérdida de información… Todo ello para conseguir una estructura óptima. En este apartado estudiaremos si podemos hacer alguna transformación a las diferentes tablas hasta llegar hasta la Forma Normal 3.

FN1

Una tabla está en FN1 si todos los atributos no clave, dependen funcionalmente de la clave, una dependencia funcional A → B quiere decir que, para cada valor de A, solo puede existir un valor de B.

Debido a que en todas nuestras tablas tenemos un valor ‘id’ como clave primaria, no es posible tener 2 tuplas con el mismo id y con distinto atributo no clave, por lo que todas nuestras tablas estarían en FN1.

FN2

Una tabla estará en FN2 si está en FN1 y además, todos los atributos que no pertenecen a la clave dependen funcionalmente de forma completa de ella y no de un subconjunto de la misma. De aquí concluimos que si una tabla está en FN1 y tiene solo un atributo como clave, también estará en FN2, por lo que se puede observar que todas nuestras tablas también están en FN2.

FN3

Una tabla estará en FN3 si está en FN2 y además no existen atributos que no pertenecen a la clave que dependen transitivamente de la clave. Es decir, la tabla se encuentra en FN2 y no existen atributos NO clave que tengan dependencias funcionales entre sí.

Si una tabla tiene más de 2 atributos, no incumple la condición, por lo que las tablas **Recorrido** y **Día** están en FN3.

Además, las siguientes tablas NO tienen dependencias transitivas, todos los atributos no clave dependen de la clave, no de otros atributos no clave, por lo que estarían en FN3.

3.2.- Diagrama de Clases

Ya teniendo perfiladas nuestras posibles tablas, vamos a entrar en detalle con este apartado, en el cual diferenciaremos los diagramas de clases para la aplicación cliente y para la aplicación servidor.

Aplicación servidor

Se puede observar el diagrama de clases de la aplicación servidor en la Figura 8:

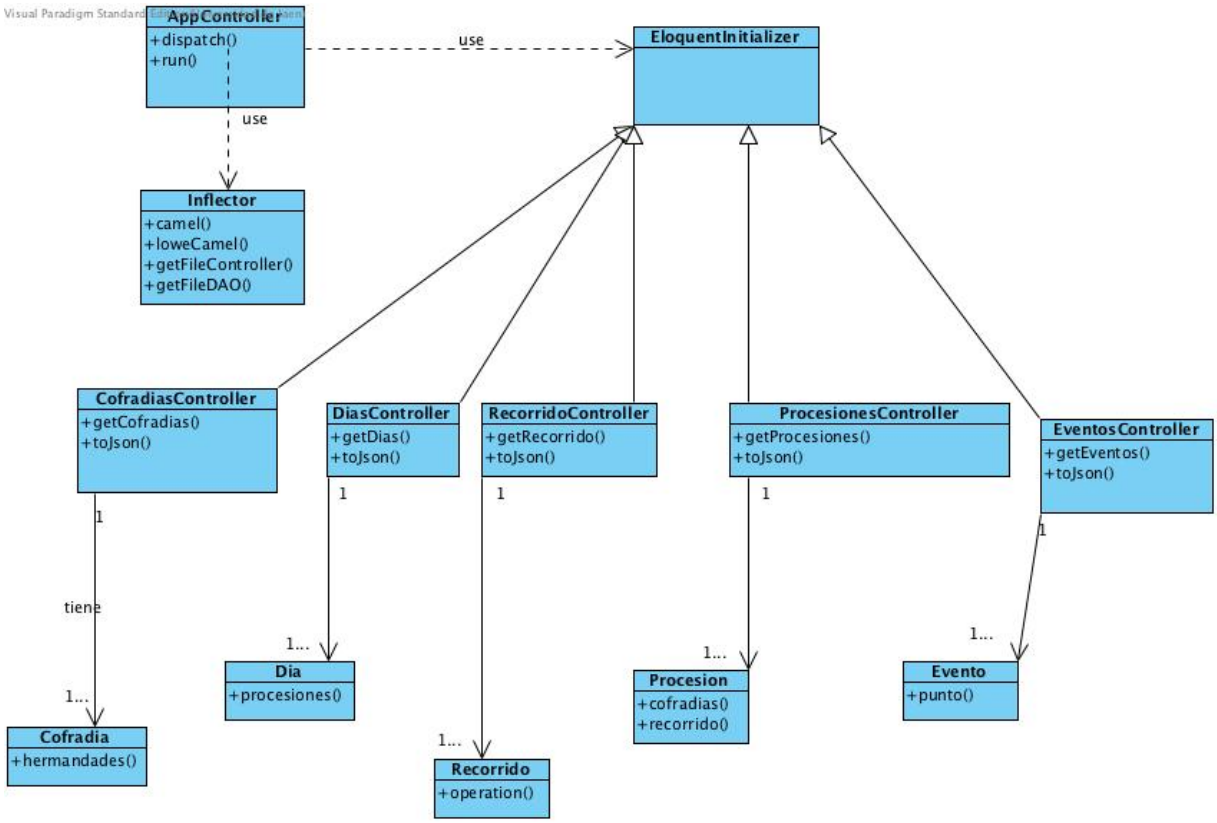

Figura 8

Podemos observar una clase 'AppController', que usa objetos de la clase 'EloquentInitializer', estos pueden ser controladores para Cofradías, Días, Recorrido, Procesiones o Eventos, usará uno u otro en función de la petición, la superclase solo se encarga de conectar con la base de datos e inicializar el ORM 'Eloquent'.

Se ha omitido el uso de clases DAO[7] ya que poseían solo 1 método, que podría entrar entre las responsabilidades del controlador, a decir verdad, teniendo el ORM, podemos abstraernos de los DAO.

Aplicación cliente

Se puede observar el diagrama de clases de la aplicación cliente en la Figura 9:

[7] Componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos

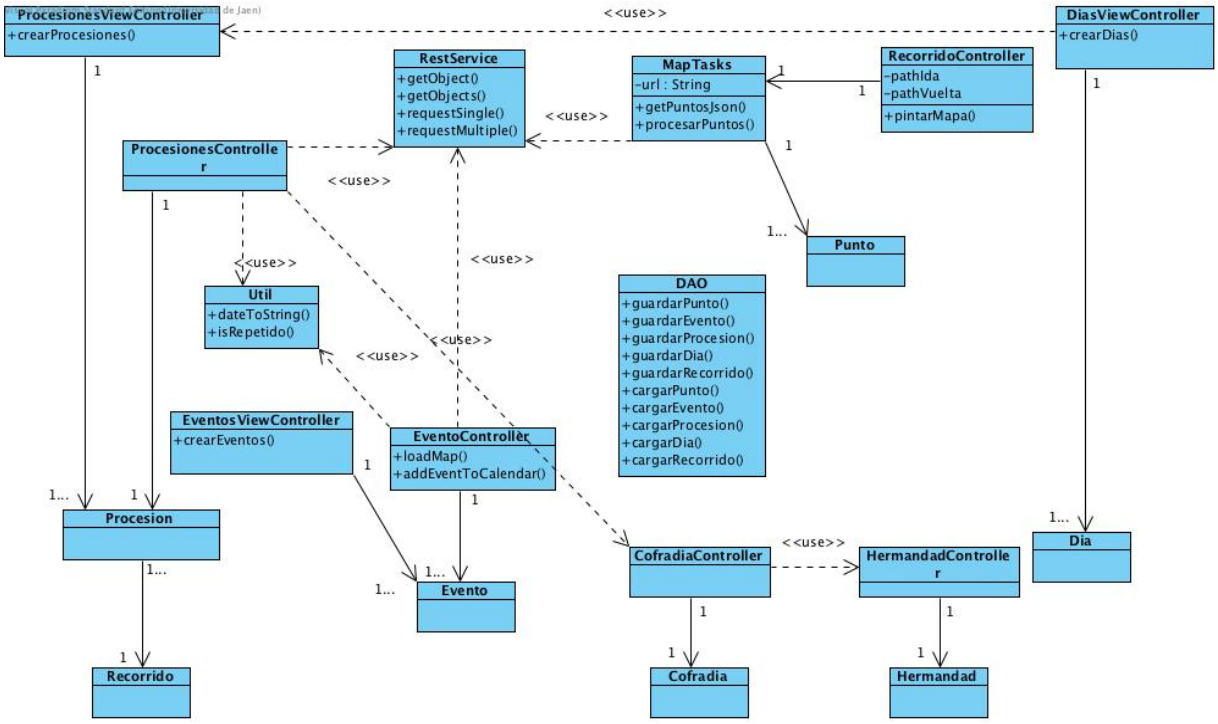

Figura 9

Para la aplicación Cliente, debido a la complejidad del mismo, se han tomado las siguientes decisiones SOLO a la hora de hacer el diagrama:

- Omisión de los atributos de las clases del modelo (Procesión, cofradia, Hermandad, Evento…)
- Omisión de las relaciones de todos los controladores con DAO, serían asociaciones de 1 a 1, debido al cruce de diferentes relaciones, no se han plasmado en el diagrama.

Como se puede observar, tenemos varias clases 'ViewController', son las encargadas de mostrar listas de objetos en nuestra aplicación, así, tendríamos para las entidades Dia, Procesion y Evento. Los 'Controller', serían los encargados de mostrar información para cada entidad, por ejemplo, 'EventoController' sería el encargado de esto y de la lógica de los eventos.

Aparte, contamos con otras clases como 'Util', usada para formatear fechas, por ejemplo, 'RestService', usada para hacer peticiones GET al API REST, y 'MapTask', encargada de realizar diferentes tareas de procesamiento a los puntos obtenidos del API de Google Maps para posteriormente mostrarlos en un mapa.

Como se puede observar, ambos diagramas tienen algunos elementos en común, como, por ejemplo, las clases del modelo (Procesión, Cofradia…).

Por si quedase alguna duda sobre el diagrama de clases para las clases del modelo, sería el siguiente, básicamente el diagrama del modelo de dominio más detallado, se muestra en la Figura 10:

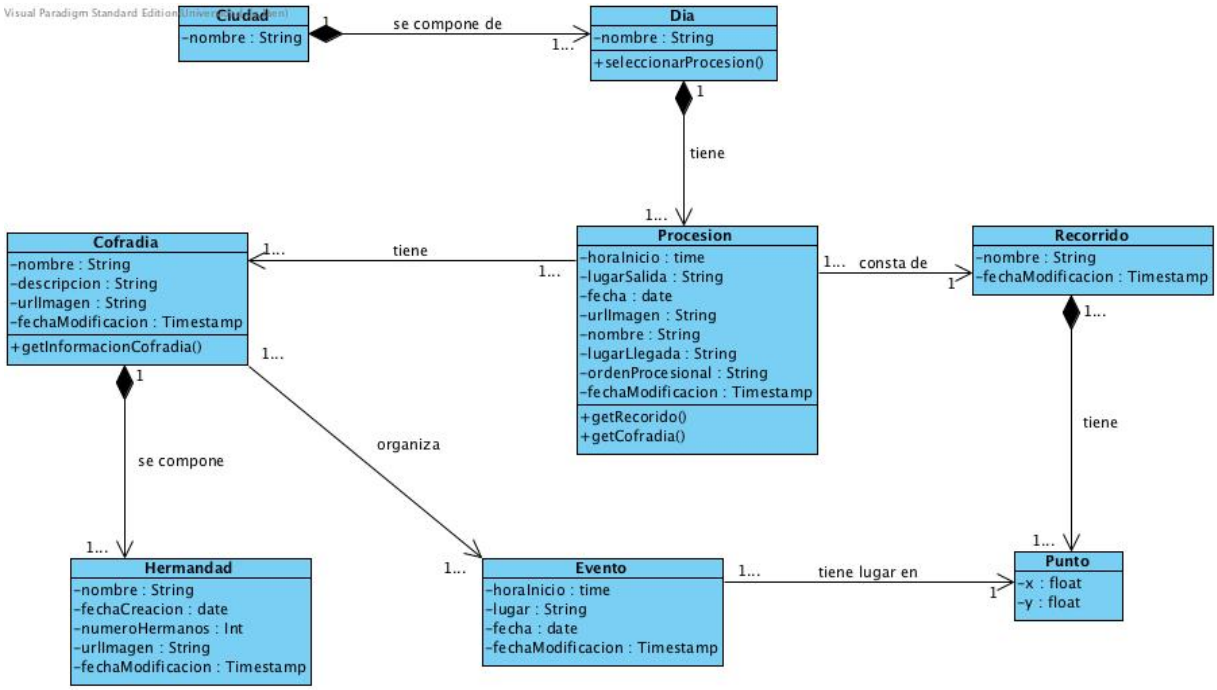

Figura 10

Si comentamos el uso de patrones, en el servidor se ha usado un M(V)C, tenemos controladores y clases del modelo, pero ya que los datos se exponen a través de un API REST, no se usaron vistas.

Respecto al cliente, se usa, además de MVC, con nuestras clases controladores y clases del modelo, y como representación de las vistas el storyboard de Xcode. Además, se usa un componente DAO, lo que nos proporciona una abstracción respecto al sistema de almacenamiento.

3.3.- Diagramas de secuencia

Una vez visto el diagrama de clases, en este apartado, vamos a plantear los diagramas de secuencia para los casos de uso que vimos anteriormente:

- *Consultar recorrido de una procesión*

Podemos observar el diagrama en la Figura 11:

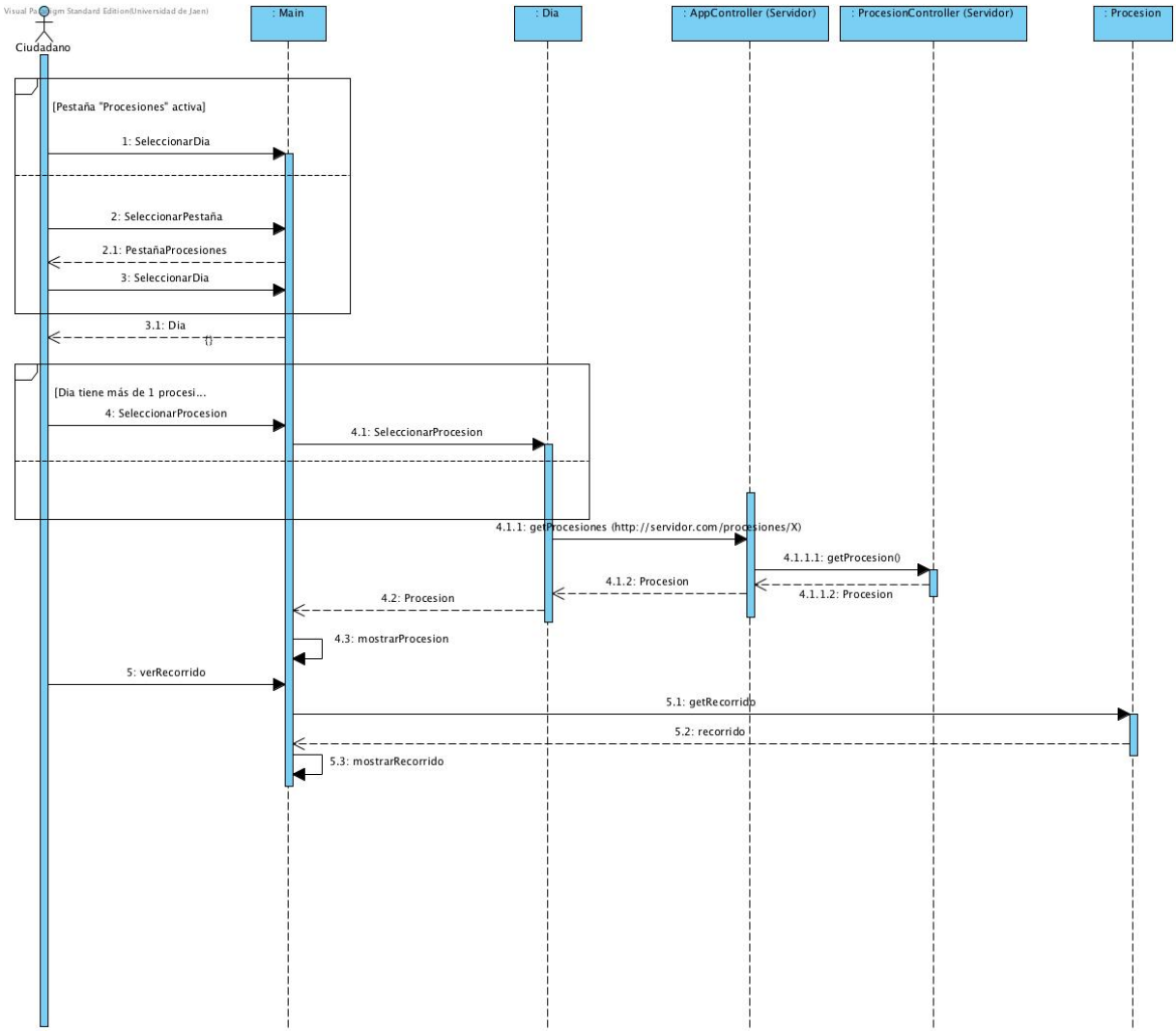

Figura 11

En el diagrama podemos observar que, antes de todo hay que seleccionar la pestaña 'Procesiones' (en el siguiente punto de diseño de la interfaz se mostrarán las distintas vistas de la aplicación), en el caso de no estar seleccionada, a continuación, se selecciona el Día, para posteriormente seleccionar la procesión (en caso de que haya más de una el mismo día), por último, se muestra el recorrido de la misma.

También podemos observar la comunicación efectuada entre el cliente y el servidor a través de http para obtener los datos de las procesiones.

- *Consultar información sobre una cofradía*

Podemos observar el diagrama en la Figura 12:

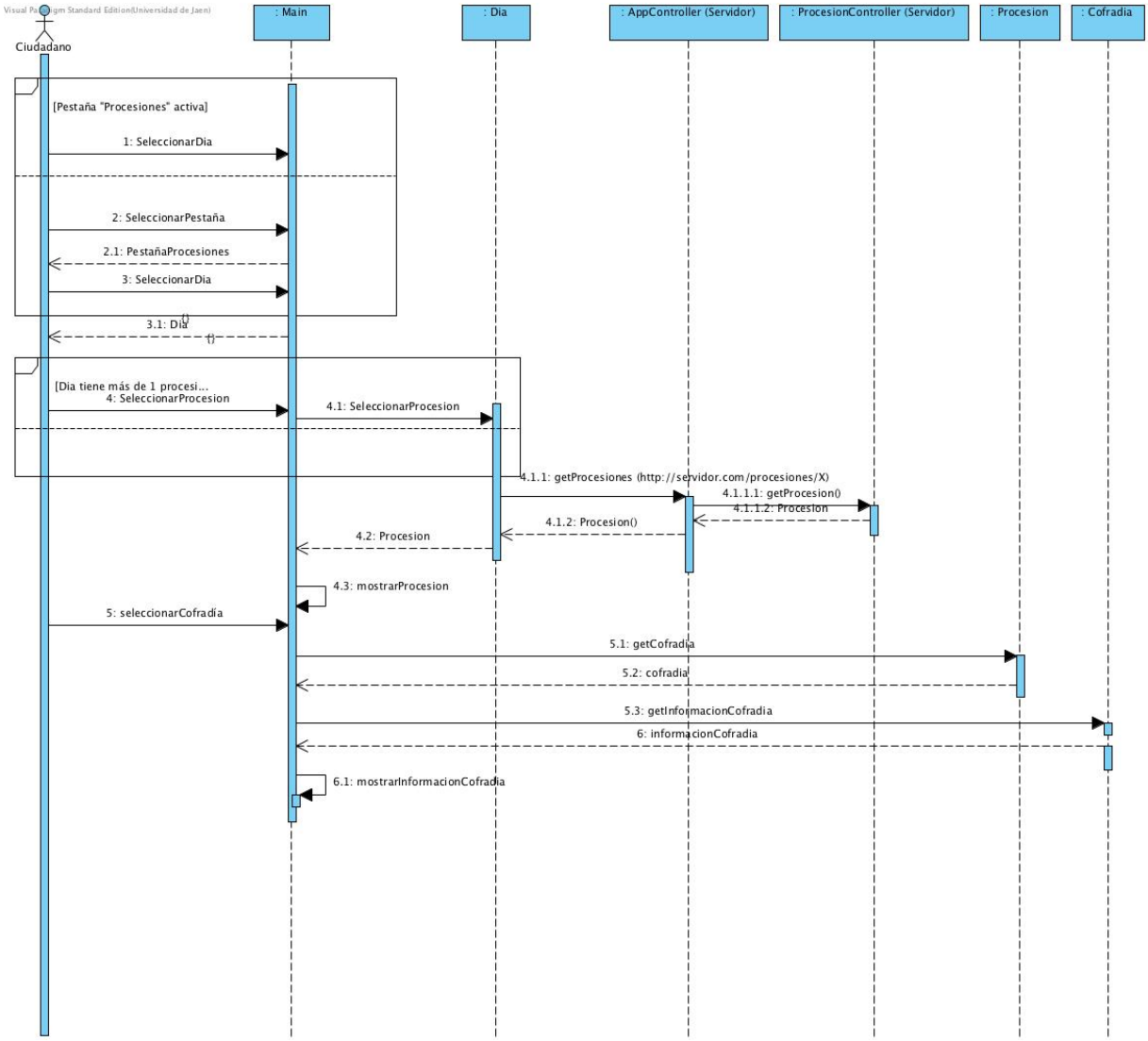

Figura 12

Como se puede comprobar, una vez seleccionada la procesión, como en el diagrama de secuencia anterior, ahora tenemos que seleccionar la cofradía que queramos, para posteriormente solicitar la información de la misma.

Aquí también podemos ver la comunicación que tiene lugar entre cliente y servidor a través de http para obtener información.

3.4.- Diseño de la Interfaz

Ya tenemos parte del diseño interno de nuestra aplicación, ahora vamos a centrarnos en el diseño exterior o de interfaz. A continuación, se muestran las vistas

más importantes que tendría la aplicación, en forma de wireframes, además de un storyboard y un pequeño prototipo funcional.

Wireframes

Un Wireframe es una representación simple de una interfaz, en el que se hace hincapié, sobre todo, en las aristas que componen los objetos de la interfaz.

Podemos observar en la Figura 13 y 14 la vista inicial de la aplicación, donde se muestra la lista de los distintos días en los que hay procesión y los distintos eventos que tendrán lugar:

Figura 13

Figura 14

Podemos cambiar entre una u otra en el menú de abajo, profundicemos en cada una de ellas, eventos primero, observemos la Figura 15:

Figura 15

Tendríamos diferente información del evento, como el lugar y la hora, además, abajo mostraría un mapa de Google con la visualización del sitio físico del evento.

Sigamos con las procesiones, visualicemos la Figura 16:

Figura 16

Aquí tenemos la vista detallada de la procesión, como se puede observar, con una pequeña imagen, diferente información de la misma, posibilidad de consultar el recorrido, además de poder seleccionar una cofradía.

Storyboard para el caso de uso 'Consultar recorrido de una procesión'

Habiendo visto los wireframes, podemos construir de forma sencilla algunos storyboards haciendo uso de ellos.

Un Storyboard, según [8], es un conjunto de representaciones gráficas, mostradas en secuencia, con el objetivo de servir de guía para entender una historia.

En la Figura 17, podemos ver el storyboard referido al caso de uso para mosrar un recorrido:

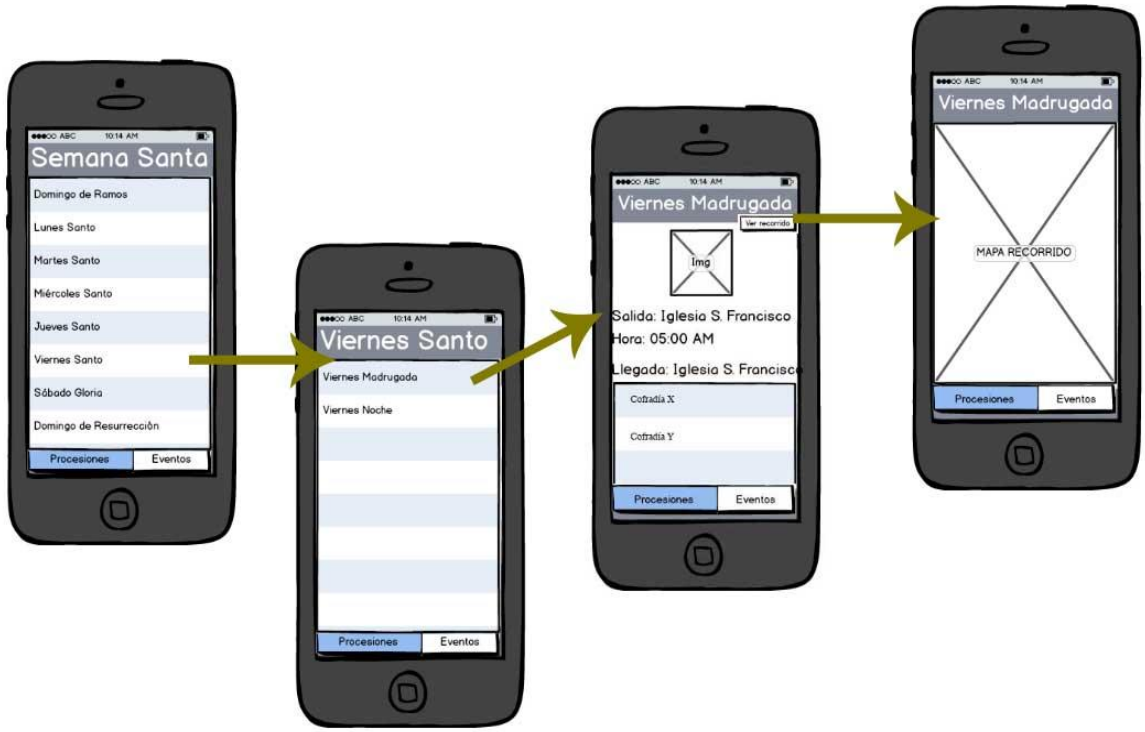

Figura 17

Como aclaración, en el storyboard se muestra la transición de la vista de selección de días a la vista de selección de procesiones, en el caso de haber solo una procesión ese día, se pasa directamente a la vista de información de la procesión.

Storyboard para el caso de uso 'Consultar información de una cofradía'

Podemos observar el storyboard para el caso de uso de obtener información de una cofradía en la Figura 18:

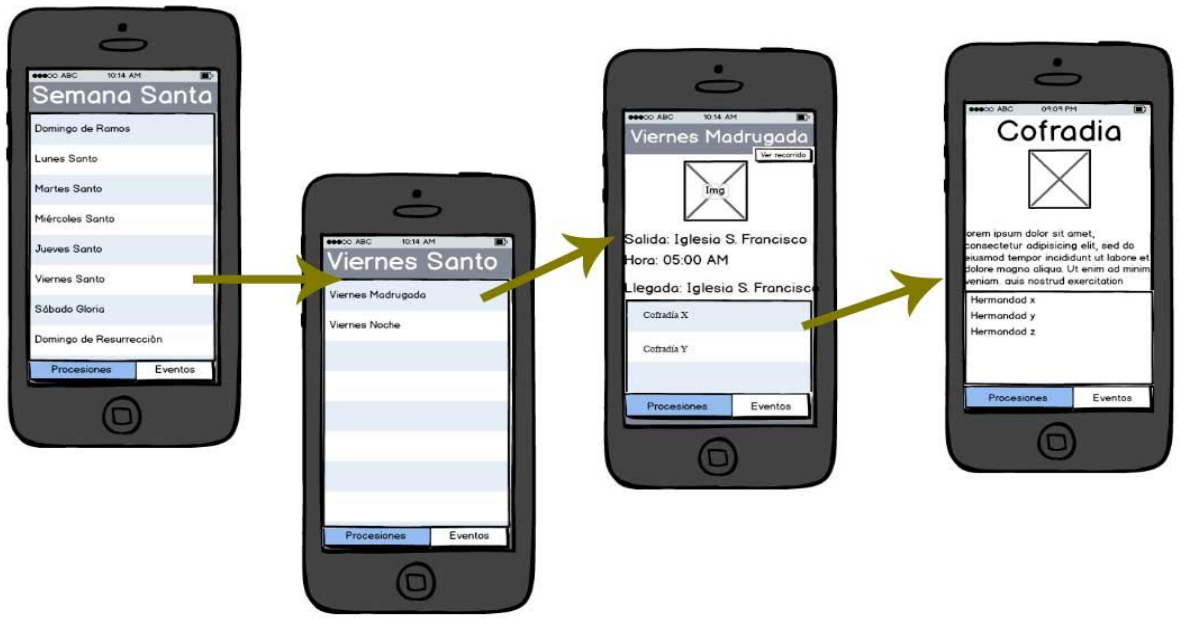

Figure 18

Como se puede observar, los dos storyboards son similares, con la salvedad de que en uno se muestra el recorrido de la procesión, y en otro información de una de las cofradías que participa en la misma.

Prototipo

Se ha creado un prototipo funcional con la herramienta Invision , en el apéndice “Material Suministrado”, se puede copiar la url del mismo para visualizarlo en un navegador web.

4.- Implementación

Ya tenemos lista la parte del diseño, estamos listos para pasar a la implementación, aquí veremos por encima la arquitectura de nuestro sistema, así como algunos detalles sobre implementación.

4.1.- Arquitectura

A continuación se muestra un diagrama arquitectónico, en la Figura 19, donde se puede observar el esqueleto del sistema cliente / servidor, podemos apreciar un elemento ‘Almacenamiento de datos’, será el SGBD encargado de almacenar todos nuestros datos, también apreciamos un ‘Servidor’, montado sobre

Apache Http Server, exponiendo a través de un API REST usando Slim framework, los diferentes datos en formato JSON, por último, se ven varios 'Software cliente', corriendo la aplicación cliente para iOS, programada en SWIFT, que se conecta con nuestro API, también podríamos tener otras aplicaciones cliente, por ejemplo, una aplicación Android o Web.

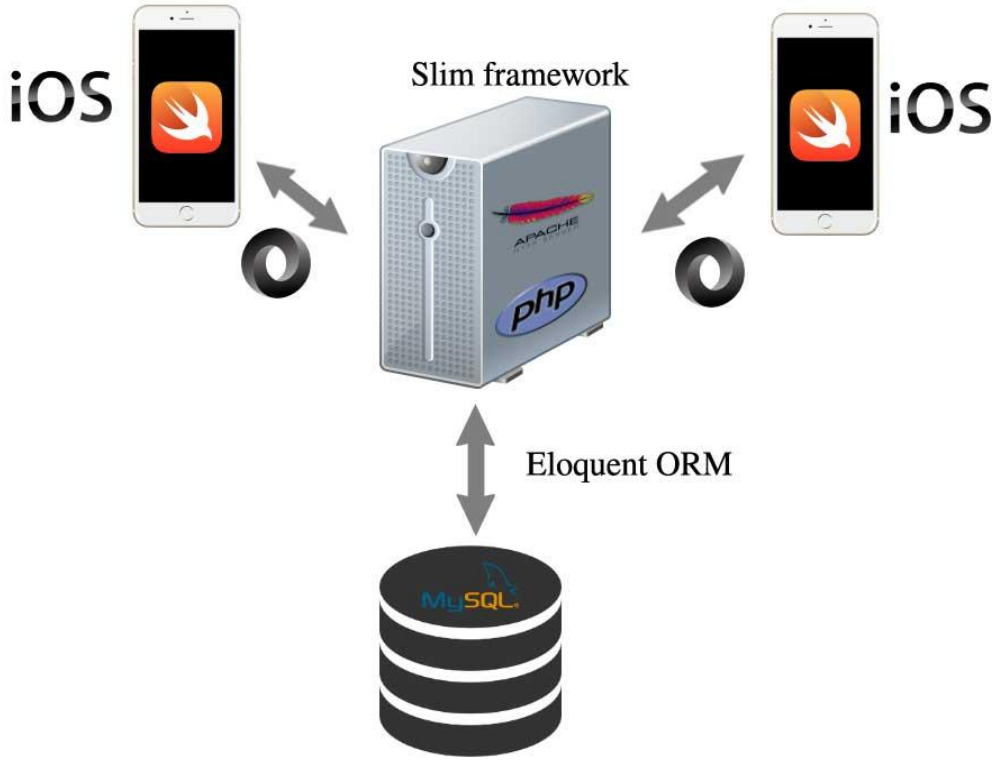

Figura 19

4.2.- Detalles sobre implementación

Si entramos en detalle en materia de implementación, para la programación en Swift, me he servido mucho del libro [9] y el capítulo [10], respecto a su ejecución, debido a que no tengo un dispositivo iOS con Jailbreak, ni tengo cuenta de desarrollador de Apple, me he visto obligado a utilizar el emulador de xCode que, todo hay que decirlo, ha superado mis expectativas, ya que vengo del mundo Android y, en algunos casos, los emuladores de esta plataforma dejan mucho que desear.

Para implementar la funcionalidad de mapas de Google, he seguido los pasos que se indican en [11].

En cuanto a la parte servidor en PHP, ya tenía algunos conocimientos del lenguaje, pero aprendí a estructurar bien el código y construir un pequeño “framework” para el propósito del proyecto, respecto al framework para el API REST, Slim, usé su documentación [12] para llevar a cabo las tareas necesarias. La versión usada es la 2, al haber instalado el framework con el gestor de paquetes ‘composer’, podemos actualizar la misma cuando haya una nueva versión.

Respecto al acceso a base de datos, se ha usado el ORM del framework Laravel, ‘Eloquent’, cuya documentación, [13], he tenido que usar en multitud de ocasiones para realizar algunas consultas algo complejas al SGBD. Para usarlo, simplemente tendríamos que extender nuestros objetos del modelo de la clase ‘Model’ de Eloquent, y a partir de ahí construir métodos donde se usan, a su vez, métodos de Eloquent para hacer consultas.

A la hora de subir la aplicación del servidor a producción, no tendríamos muchos problemas, ya que, a día de hoy, en multitud de sitios ofrecen soporte para aplicaciones PHP, simplemente tendríamos que modificar el fichero ‘.htaccess’ y configurar la ruta de nuestra aplicación. Así, para hacer las pruebas se ha usado el servidor Apache, y para la futura versión de desarrollo, también se usará.

4.3.- Pruebas

Para este proyecto se han realizado dos pruebas, relacionadas con interfaz y usabilidad, enumeradas y detalladas a continuación:

- Prueba en la visualización de los recorridos
    - En este caso, se daba a probar la aplicación, en el punto de visualizar el recorrido de una procesión, a diferentes personas, en un caso, aparecía la pequeña leyenda de qué significaban los dos colores con los que se pintaba el recorrido, en el otro no.
    - La gran mayoría de usuarios, al preguntarles cuál era el recorrido de salida y de llegada en la modalidad del mapa sin leyenda, no sabían especificarlo a menos que les dieras algunas pistas, es más, no tenían ni idea del por qué del uso de dos colores, así, se llegó a la conclusión de que había que especificar esta pequeña leyenda para que el usuario no se desorientara con los dos colores.
- Prueba en la visualización de la lista de procesiones

- En este caso, teníamos dos posibles maneras de representar las distintas procesiones que tenían lugar, la primera de ellas, era una lista con todas las procesiones, si en un día había tres procesiones, las 3 aparecían junto a las demás, en el segundo caso, aparecían los diferentes días en los que había procesiones y, en el caso de que un día tuviera más de una procesión, salía una nueva ventana donde seleccionar la deseada.
- El resultado fue que la gran mayoría de los usuarios encuestados preferían la segunda opción, en el primer caso la lista podía llegar a ser bastante amplia y no encontrabas el item deseado con facilidad, así, se optó por listar los diferentes días (como máximo siete), y si había más de una procesión el mismo día, aparecía una lista donde seleccionar la deseada.

5.- Conclusiones

Para finalizar con la memoria, vamos a ver el apartado de conclusiones, en el que se expondrán las opiniones del alumno, así como mejoras y trabajos futuros para el proyecto.

Para empezar con las conclusiones, he de decir que se han cumplido los objetivos inicialmente citados, se ha hecho un estudio sobre SWIFT y el desarrollo en iOS, se ha diseñado un sistema cliente / servidor para la consulta de información sobre Semana Santa y, por último, se ha desarrollado un prototipo usando SWIFT accediendo a un servidor REST.

Respecto a las impresiones sobre las tecnologías utilizadas, Swift me ha parecido un lenguaje bastante sencillo de usar, aunque, no puedo compararlo con Objective-C ya que no he tenido ocasión de usarlo. Respecto al servidor, me ha gustado montar un “mini framework” con php para este proyecto, además, el hecho de usar el framework para REST, Slim, me evitó el uso de otro marco de trabajo más grande y cuyas posibilidades no iba a aprovechar.

Si hablamos de xCode, en general me parece un IDE bastante bueno, que sobrepasa por mucho a la competencia, Android Studio, sin embargo, no me gusta que algunas funciones básicas, como refactorizar una clase, no estén disponibles por defecto.

Para concluir con las impresiones, el hecho de haber descompuesto la aplicación en cliente - servidor y exponer un API REST hace posible que en cualquier momento se puedan añadir otros clientes, sean de la plataforma que sea, lo que le da al proyecto un punto más en cuanto a escalabilidad.

Sin duda, personalmente, el aspecto más importante del proyecto para mí, era introducirme en el mundo iOS, cosa que creo que he logrado con creces, cuando llegue el momento de crear mi segunda aplicación, estoy seguro que saldrá mucho mejor que la actual, ya que he aprendido de los errores cometidos.

Por último, si hacemos referencia a lo aprendido en el grado, en este proyecto se han utilizado algunas de las cosas aprendidas en el mismo, por ejemplo:

- Utilización de arquitectura MVC para el servidor, aunque en este caso sin vistas, aprendido en Desarrollo de Aplicaciones Web
- Utilización de API REST, para exponer datos en JSON de manera sencilla, aprendido en Desarrollo de Aplicaciones Empresariales
- Conocimiento de POO en general, cuyo aprendizaje inicial fue en la asignatura Programación Orientada a Objetos y fue mejorando a lo largo de los distintos cursos.

Respecto a Swift, no tengo conocimiento de que se haya impartido en el grado una asignatura para desarrollo de dispositivos móviles con este lenguaje, puede que por su reciente aparición, aunque sí se que se ha impartido usando Objective-C. Sin embargo, no estuve matriculado de la asignatura.

Un hecho destacable, en mi opinión, es la intención de salir de mi ‘zona de confort’ para aprender y profundizar sobre un tema que no conocía, no era mi intención desarrollar un proyecto única y exclusivamente con tecnologías que ya había utilizado, así que, como todo en la vida, es bueno probar tecnologías distintas para, a la hora de desarrollar un nuevo proyecto, tener más alternativas donde elegir y poder llevarlo a cabo con la más indicada para el propósito.

5.1.- Mejoras y trabajos futuros

Como futuras mejoras para el prototipo que se ha llevado a cabo, se plantean los siguientes puntos:

1. Disminución de la cantidad de información que se transmite del servidor a la aplicación, podríamos conseguirlo comprobando la última fecha

de modificación de una entidad en el servidor, con la almacenada en el cliente.

2. Mostrar un botón de “Cómo llegar” en el mapa que muestra la ubicación de un evento.

3. Posibilidad de notificar al usuario sobre el retraso o cancelación de una procesión.

4. Señalar en el mapa del recorrido los puntos donde se encuentran las imágenes en tiempo real.

5. Hacer posible el funcionamiento sin conexión, en este momento no es posible, por lo que se pretende que, en un futuro cercano, si el usuario ya ha descargado datos previamente usando la conexión a internet, pueda acceder a los mismos sin necesidad de estar conectado.

6. Mostrar en el mapa del recorrido un ‘punto’ que simbolice dónde se encuentra el usuario, simplemente habría que leer el API de Google para saber cómo hacerlo. Además, mostrar una ruta óptima de llegada al recorrido de la procesión.

7. Ofrecer información personalizada según la localidad donde nos encontremos, habría que obtener la geolocalización y consultar al servidor en función de estos parámetros.

8. Construir un portal web para la administración de contenidos, creación de procesiones, modificación de datos...

Estos cambios podrían llevarse a cabo sin apenas modificaciones en la arquitectura y diseño de la aplicación. Por ejemplo:

- Para el punto 1, la arquitectura ya está preparada, simplemente habría que añadir la lógica en el cliente. El campo ‘fecha modificación’ ya presente en la mayoría de entidades, lo que haría posible esta mejora.
- Para el punto 2, ya que en nuestro diseño contamos con el punto (latitud y longitud) donde se lleva a cabo el evento, simplemente tendríamos que obtener la geolocalización del terminal y pedir al API de Google Maps que nos calcule la ruta de un punto a otro.
- Para la tercera mejora, , solo tendríamos que añadir un campo “cancelada” o “retrasada” a la tabla ‘Procesión’, y enviar una notificación PUSH al usuario cuando llegue el momento.

- Para la mejora 4, sí habría que añadir una tabla ‘Imágen’ con campos ‘latitud’ y ‘longitud’, y asociarla con las procesiones. Por imágen entendemos la representación del Santo en procesión.
- Para la mejora 5, no tenemos que modificar nada en la arquitectura, solo comprobar que nuestro dispositivo tenga conexión a internet o no, en cada intento de llamada a nuestro API.
- Respecto al punto 6, no tenemos que modificar nada nuestra arquitectura, solo interactuar con el API de Google para obtener la geolocalización.
- En lo que concierne al punto 7, ya tenemos una tabla ‘Ciudad’, relacionada con los diferentes días de procesiones, precisamente pensado para este punto, así, solo tenemos que obtener la geolocalización del usuario, o preguntarle de qué ciudad es, para darle información según su situación geográfica.
- Si hablamos sobre el punto 8, tampoco necesitamos modificar la arquitectura en exceso, ya que la plataforma web de administración se crearía en torno a la que ya hay, es más, solo habría que añadir las distintas tablas que puedan ser necesarias, por ejemplo, para identificación de usuarios del portal.

Como se puede comprobar, no es necesario hacer grandes cambios en el diseño o arquitectura para realizar los cambios.

6.- Bibliografía

[1] *Requisito*. Wikipedia. [Online] Disponible en: https://es.wikipedia.org/wiki/Requisito . Acceso: Jun, 2015

[2] *Boletín Oficial del Estado*. (Febrero 2013). Agencia Estatal Boletín Oficial del Estado. IX Convenio Colectivo de Telefónica Telecomunicaciones Públicas, S.A. [Online] Disponible en: http://www.boe.es/boe/dias/2013/02/26/pdfs/BOE-A-2013-2153.pdf

[3] *Boletín Oficial del Estado*. (Abril 2014). Agencia Estatal Boletín Oficial del Estado. Acta de la reunión de la comisión mixta del convenio colectivo nacional del ciclo de comercio del papel y artes gráficas. [Online] Disponible en: http://www.boe.es/boe/dias/2014/04/10/pdfs/BOE-A-2014-3855.pdf

[4] *Boletín Oficial del Estado*. (Mayo 2014). Agencia Estatal Boletín Oficial del Estado. Tablas Salariales 2014. [Online] Disponible en: http://www.boe.es/boe/dias/2014/05/27/pdfs/BOE-A-2014-5544.pdf

[5] Casos de uso. Wikipedia [Online] Disponible en: https://es.wikipedia.org/wiki/Caso_de_uso . Acceso: Jun, 2015

[6] Roger S. Pressman, *Ingeniería del software. Un enfoque práctico*. 7ª ed. Madrid: McGrawHill/Interamericana de España, 2010

[7] Normalización de Bases de Datos. Wikipedia [Online] Disponible en: https://es.wikipedia.org/wiki/Normalizaci%C3%B3n_de_bases_de_datos. Acceso: Jun, 2015.

[8] Storyboards. Wikipedia [Online] Disponible en: https://es.wikipedia.org/wiki/Storyboard. Acceso: Jun, 2015.

[9] Sean Liao , *Migrating to Swift From Android*. Apress (Kindle Edition), 2014. Disponible en Amazon Kindle

[10] Aaron Douglas, Saul Mora, Matthew Morey and Prieto Rea "Your first Core Data App", en Core Data By Tutorials. Ed. Razeware LLC, 2014, pp. 15-33.

[11] *Google Maps SDK for iOS*. [Online] Disponible en: https://developers.google.com/maps/documentation/ios/start . Acceso: Recuperado en Mayo. 25, 2015

[12] Slim Framework. Web oficial de Slim [Online] Disponible en: http://www.slimframework.com/ . Acceso: Jun, 2015.

[13] Eloquent ORM. Web oficial de Laravel [Online] Disponible en: http://laravel.com/docs/5.0/eloquent. Acceso: Jun, 2015.

[14] Composer. Getting Started [Online] Disponible en: https://getcomposer.org/doc/00-intro.md. Acceso: Jun, 2015.

Apéndice I. Manual de Instalación del sistema

La instalación del sistema se dividiría en dos partes:

Instalación del servidor

Si queremos instalarlo en un servidor en internet, podríamos comprar un hosting compartido, pero en este caso vamos a ver cómo podemos ejecutarlo en nuestra máquina local.

Lo más sencillo es usar un entorno como WAMP (Windows), XAMP(Linux), o MAMP(Mac). En este caso vamos a hacerlo con MAMP.

Simplemente vamos a la web de MAMP https://www.mamp.info/en/ , y descargamos el software. Una vez descargado, lo ejecutamos e iniciamos servidores, una vez hecho, veremos algo como lo mostrado en la Figura 20:

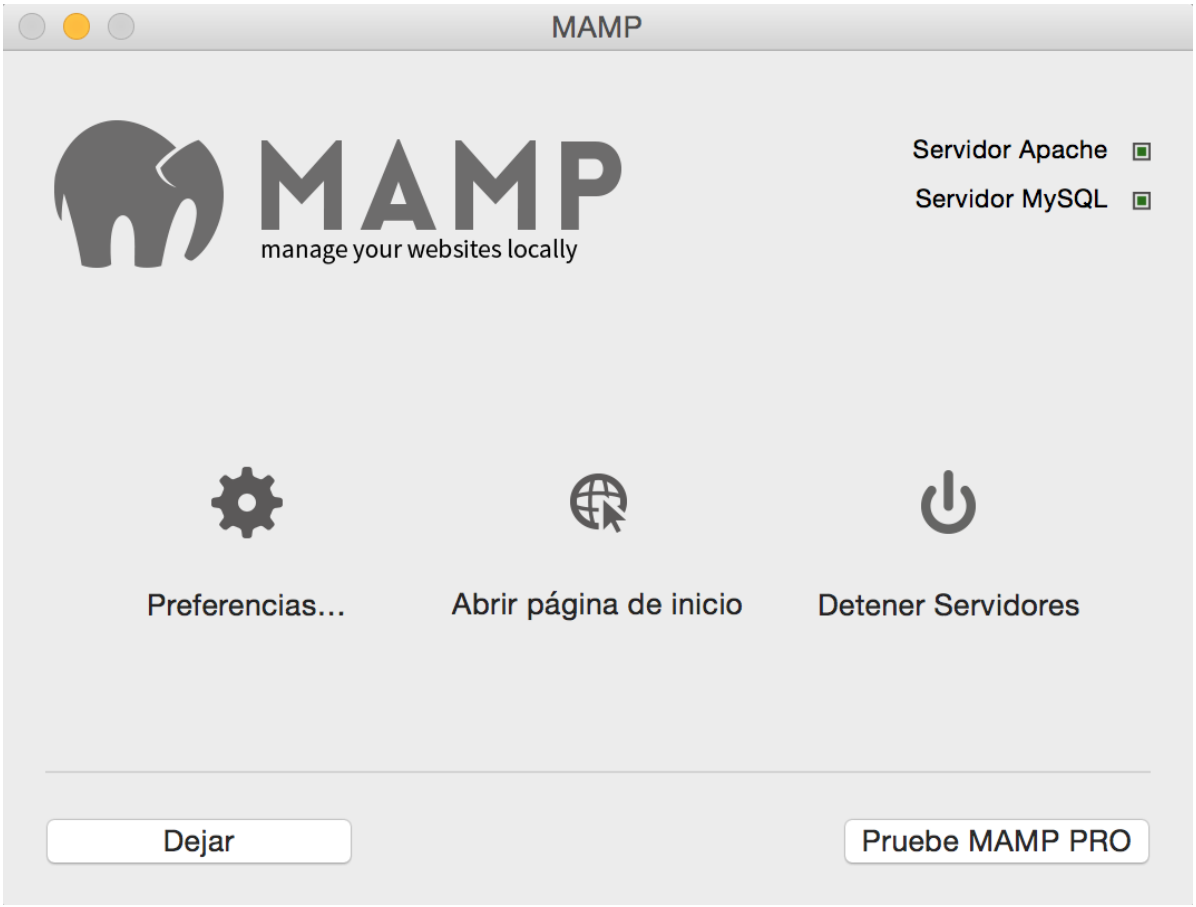

Figura 20

Ahora, si entramos a 'http://localhost' desde un navegador web, nos aparecerá una página de bienvenida.

Bien, para ejecutar nuestra aplicación servidor, necesitaremos SLIM y Eloquent, podemos descargarlo desde su web oficial, [12] y [13] o usando el gestor de paquetes composer, tal y como se muestra en [14].

Para ello, creamos en nuestra carpeta del proyecto para el servidor e insertamos los ficheros fuente, (ha de ser una carpeta que esté dentro de la raíz de documentos que tenga MAMP para Apache (Preferencias → Apache)), un fichero llamado 'composer.json', con el contenido que aparece en la Figura 21:

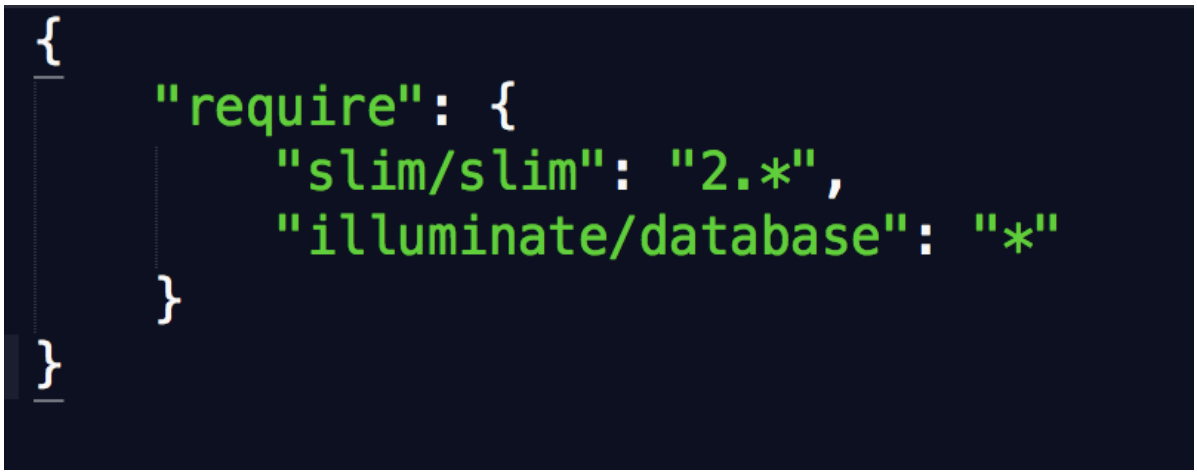

Figura 21

Una vez hecho esto, vamos al terminal, nos situamos en la carpeta del proyecto y ejecutamos 'composer install', automáticamente se instalarán las dependencias.

A partir de ahí, nuestra API REST debería funcionar.

Ejecución de aplicación cliente en simulador de xCode

Ya que en el momento de finalizar la memoria, la app no está subida a la AppStore, tenemos varias posibilidades:

- Hacer JailBreak a un dispositivo iOS, no se recomienda esta opción, debido a su ilegalidad y a posibles problemas técnicos.
- Obtener licencia desarrollador de Apple, cuyo precio es $99.
- Ejecutar el proyecto en un simulador de xCode, esta sería la opción recomendable, y la que se especifica a continuación.

Para empezar, necesitaremos un ordenador Mac, una vez lo tengamos, abrimos la aplicación "App Store" y descargamos "xCode".

Una vez listo, vamos a "File → Open" y buscamos el proyecto. A continuación pulsamos el botón 'Play', se puede observar en la Figura 22.

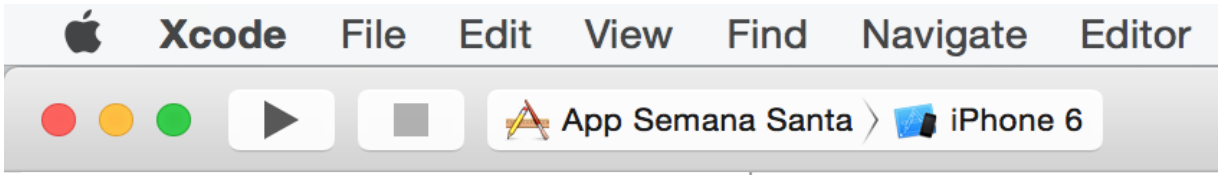

Figura 22

El proyecto de la aplicación cliente necesita una serie de librerías, que habrá que incluir para poder ejecutarlo, son las que se muestran en la Figura 23:

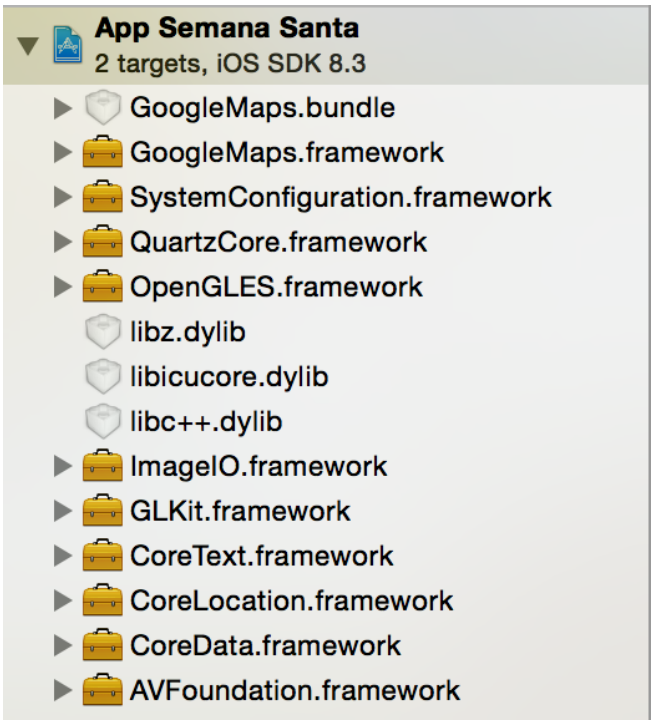

Figura 23

Para añadir una librería, pulsamos en nuestro proyecto, A continuación se nos despliega una ventana a la derecha, pulsamos en 'Targets → App Semana Santa' , y a continuación hacemos scroll hasta abajo y pulsamos en el botón '+', ver Figura 24:

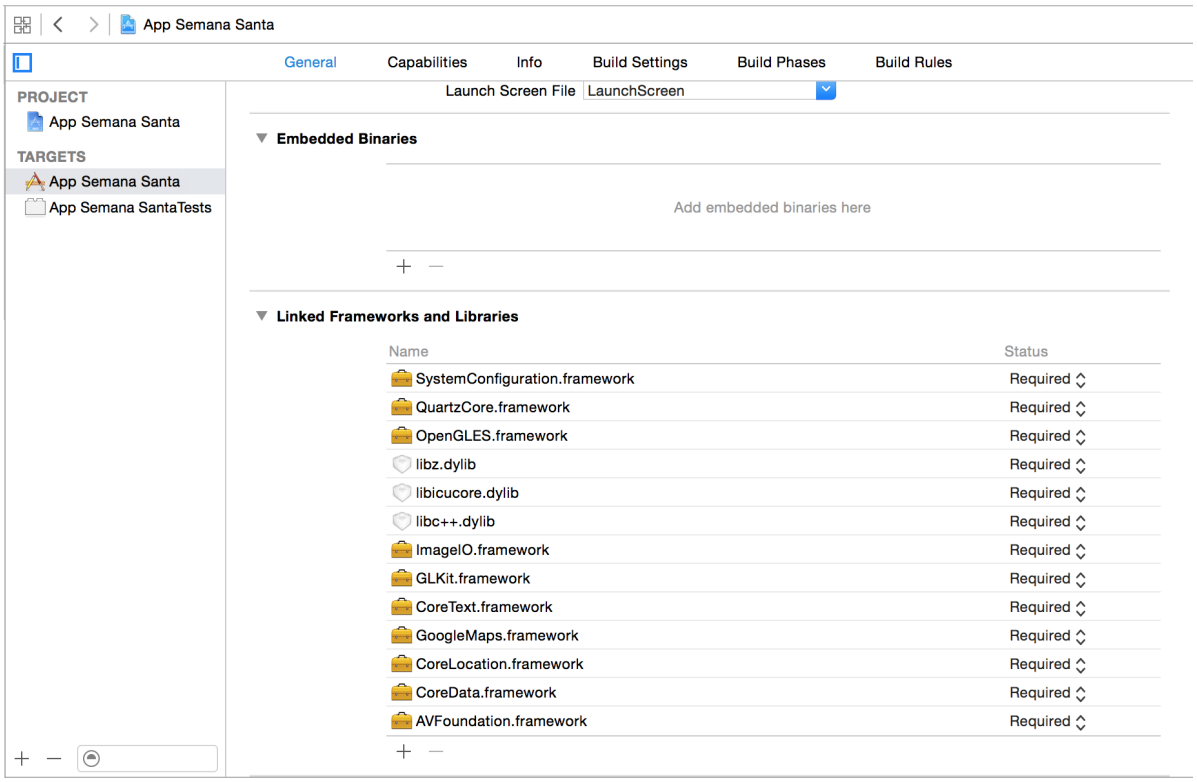

Figura 24

Para poder usar el API de Google Maps, necesitamos generar una clave para el API, tal y como se especifica en [11].

Ya deberíamos poder ejecutar la aplicación en el simulador.

Apéndice II. Manual de Usuario

A continuación pasan a detallarse el funcionamiento de la aplicación:

Para comenzar, tendríamos una lista con los diferentes días de la Semana Santa en los que hay como mínimo una procesión, tal y como aparece en la Figura 25:

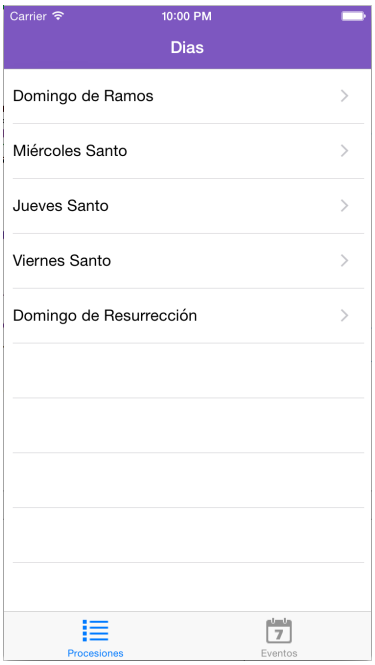

Figura 25

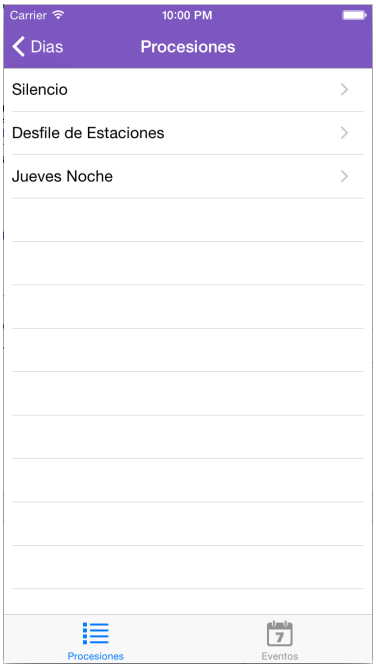

Figura 26

En la Figura 26 se aprecia que, en el caso de haber más de una procesión el mismo día, nos aparecerá una lista para seleccionar la deseada, si no, aparecerá directamente la procesión que tenga lugar ese día.

Una vez seleccionada, podremos ver información detallada de la misma, horario, lugar de salida, orden procesional… además de las cofradías que participan,

pudiendo consultar el recorrido de la procesión de información de las cofradías, tal y como podemos observar en las Figuras 27 y 28:

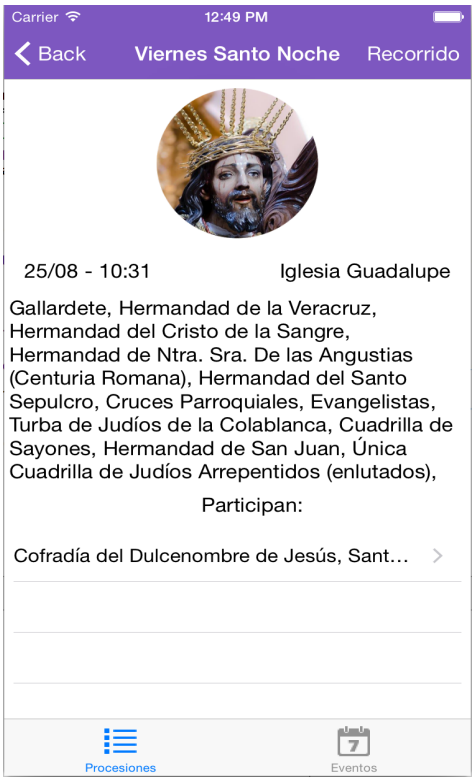

Figura 27

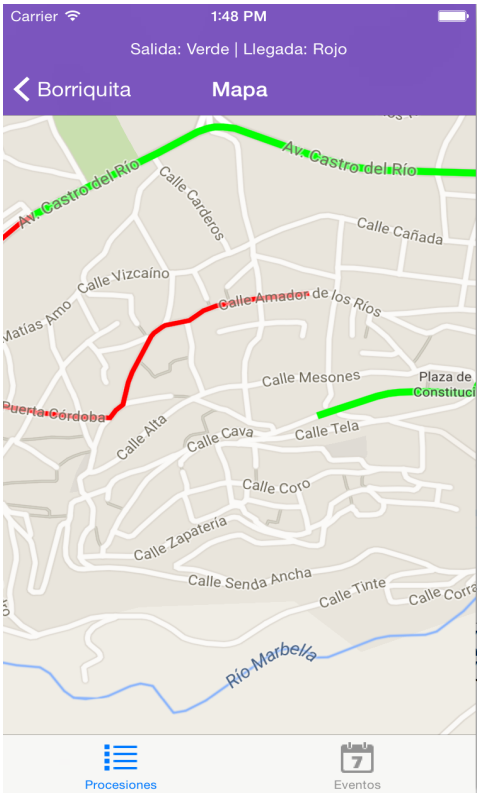

Figura 28

En la vista de la cofradía, también tenemos información sobre las distintas hermandades de la cofradía, pudiendo consultar información sobre las mismas, ver Figuras 29 y 30:

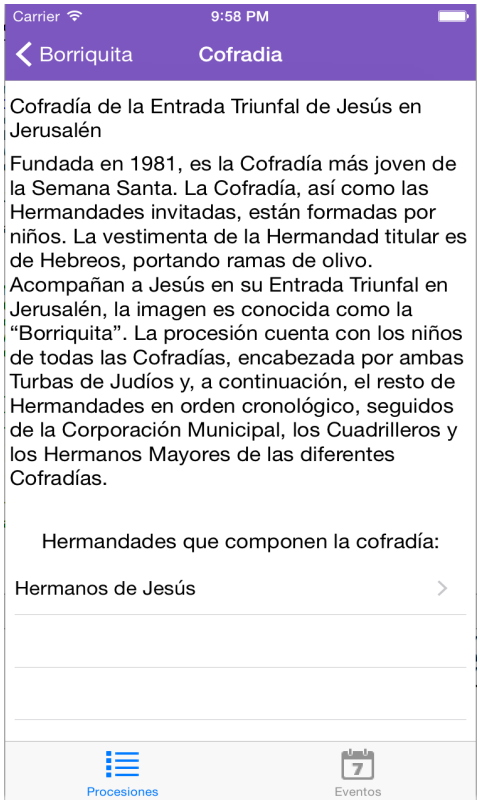

Figura 29

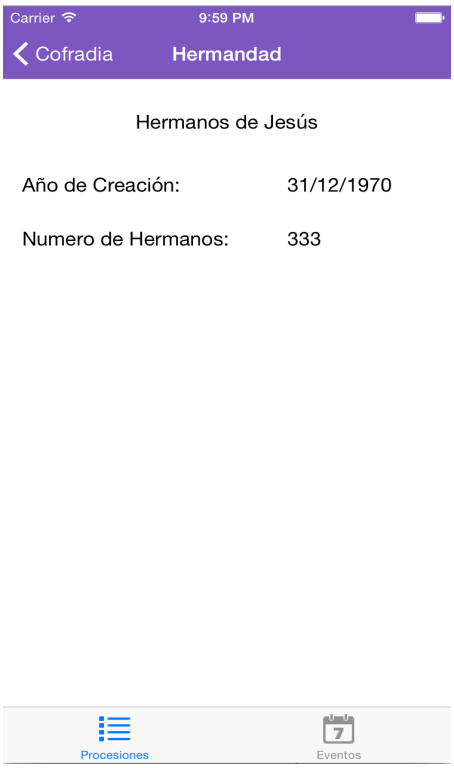

Figura 30

Si cambiamos de pestaña y vamos a ‘Eventos’, tendremos una lista con los diferentes eventos que se llevarán a cabo en la semana, si pulsamos en alguno de ellos tendremos información detallada sobre los mismos, ver Figuras 31 y 32:

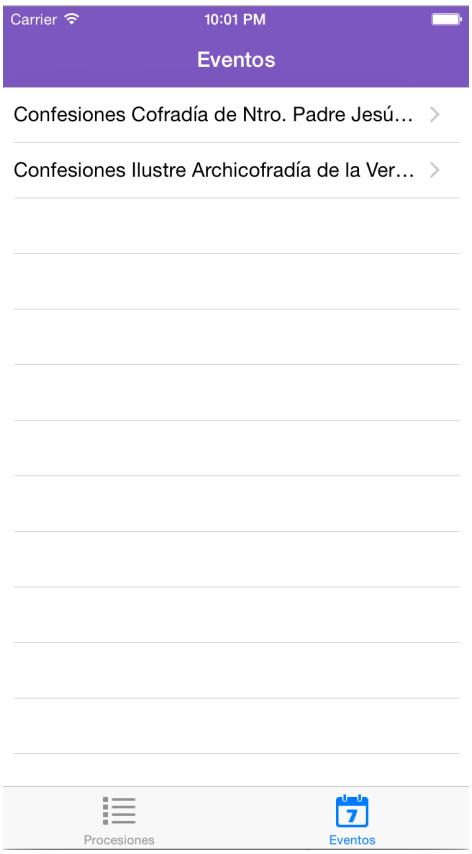

Figura 31

Figura 32

Como apunte final, se da la posibilidad al usuario de incluir el evento en su calendario, esto es posible solo desde un dispositivo iOS físico, no desde el simulador.

Apéndice III. Descripción de contenidos suministrados

En el DVD, se incluye toda la información relativa al proyecto, que se estructurará de la siguiente manera:

- Carpeta Documentación:
  - Memoria TFG en formato Word y PDF
- Carpeta Diagramas:
  - Aquí se incluyen los diferentes diagramas que se han desarrollado a lo largo del proyecto:
    - Arquitectura: formatos jpg y psd
    - Casos de uso: formato jpg
    - Diagrama Clases: formato jpg
    - Diagrama E-R: formato jpg
    - Model del dominio: formato jpg
    - Planificación de tareas: proyecto Microsoft Project, diagrama tareas jpg, proyecto Dia y jpg con diagrama jerárquico.
    - Secuenca: formato jpg
- Carpeta Diseño:
  - Aquí se incluye:
    - Personas: formato docx y pdf
    - Wireframes: Distintas vistas de la app en formato png
    - Storyboards: formato jpg
- Carpeta Implementación:
  - Aquí se incluye el código fuente de la aplicación servidor y el proyecto de xCode de la aplicación cliente, que contiene el código fuente de la app y el framework de Google Maps para iOS.
- Otros
  - Código SQL para creación de base de datos (bbdd.sql)

- Datos de ejemplo para la base de datos del servidor (datosejemplo.sql)
- Proyecto Visual Paradigm con todos los diagramas (TFG_David_Galisteo.vpp)
- Enlace a un prototipo funcional con la herramienta Invision, ya que no puede ser incluido en el documento de la memoria, se deja un enlace público al mismo.