Trabajo de Fin de Grado

GRADO DE INGENIERÍA INFORMÁTICA

Facultat de Matemàtiques
Universitat de Barcelona

APLICACIÓN MÓVIL PARA CREAR UNA RED SOCIAL DIRIGIDA A MOTORISTAS

Rocío Tovar Pérez

Directora: Inmaculada Rodríguez Santiago
Realizado en: Departament de Matemàtica Aplicada i Anàlisi. UB
Barcelona, 20 de junio de 2014

Agradecimientos

Para empezar me gustaría antes de todo ofrecer mi más sincero agradecimiento a todas las personas que han hecho posible que el desarrollo de este proyecto.

En primer lugar a mi familia y amigos por apoyarme y ayudarme a salir adelante en todos los momentos de este largo camino que ha sido mi formación en la Universitat de Barcelona.

A los profesores de la facultad de informática que me han transmitido sus conocimientos durante cada curso de la carrera en cada una de las asignaturas.

A mis compañeros de trabajo tanto de la Universitat de Barcelona donde estuve trabajando como becaria por primera vez en el campo de la informática, como de Tempos 21 donde realicé mis prácticas y donde han decidido confiar en mí para formar parte de su equipo profesional.

Finalmente me gustaría agradecer a mi tutora Inmaculada Rodríguez Santiago por su paciencia y apoyo en la realización de este proyecto.

A todos ellos, muchas gracias.

Resumen

Es muy normal en el ámbito del motor, de las motos concretamente, encontrar grupos de motoristas que planean salidas con sus motocicletas recorriendo ciertas rutas estudiadas y escogidas con anterioridad.

Actualmente la red social más usada por los motoristas para organizar salidas juntos con distintas rutas no es una red social focalizada en el ámbito motorista, cosa que hace que los usuarios experimenten carencias a la hora de interactuar dentro de esta herramienta. Existe también una aplicación focalizada en el mundo del motor pero la experiencia de usuario obtenida no es muy buena, debido entre otros factores a la lentitud que sufre esta herramienta.

Con este proyecto se pretende poner fin a algunas de esas carencias y dar así una herramienta útil y focalizada en su uso específico por parte de motoristas interesados en compartir rutas y organizar salidas, además de conocer a otras personas o grupos de sus alrededores.

Así pues se desea diseñar e implementar una aplicación en Objective-C para poder ofrecer a los motoristas una buena herramienta con la que poder comunicarse, compartir rutas y organizar sus salidas.

Abstract

It is a very common thing in the motorcycle world to find groups of riders who plan trips with their bikes traveling certain routes previously studied and chosen.

Nowadays the social network bikers use the most to organize trips together with different routes is actually not a social network focused in the motorcycle world, which makes the users to experience deficiencies when interacting within this tool. It also exists one app focused in the motorcycle ambit but the user experience obtained is not very good, due to, among other factors, the slow fluency that this tool is suffering from.

This project aims to put an end to some of these gaps and provide the users an useful tool, targeted specifically to riders interested in sharing routes and organize trips, in addition to meet other people or groups around.

So I want to design and implement an application in Objective-C to offer motorists a good tool to communicate, share and organize their trips and routes with.

Thus this project focuses on designing and implementing an application in Objective-C to offer motorists a good tool to communicate, share their routes and organize their trips.

Índice de contenidos

Tabla de figuras

1. Introducción

En el mundo del motor, es muy normal encontrar grupos de motoristas que planean salidas con sus motocicletas recorriendo ciertas rutas estudiadas y escogidas con anterioridad.

En la actualidad, gracias al avance de la tecnología y el mundo de las redes sociales la mayoría de estos encuentros se organizan mediante aplicaciones móviles. Se usan los teléfonos móviles y sus aplicaciones como medio para formar un grupo en alguna red social e intercambiar información sobre rutas, además de organizar eventos para salir en grupo con la moto.

Después convivir durante algún tiempo con un grupo de motoristas, se puede observar que actualmente la red social más usada para realizar estos eventos o quedadas no es una red social focalizada en el ámbito motorista, cosa que hace que los usuarios se vean limitados por el carácter general de la misma a la hora de interactuar dentro de esta aplicación. Existe también una aplicación focalizada en el mundo del motor llamada WeRide pero la experiencia de usuario obtenida no es muy buena, debido entre otros factores a la lentitud que sufre esta herramienta.

Con este proyecto se pretende poner fin a algunas de esas carencias y dar así una herramienta útil y focalizada en su uso específico por parte de motoristas interesados en compartir rutas y organizar salidas, además de conocer a otras personas o grupos de sus alrededores.

Para ello se implementará una aplicación móvil para iOS. El desarrollo de la aplicación se hará en Objective-C, el lenguaje nativo del sistema para poder aportar velocidad y una experiencia de usuario fluida.

2.Objetivos

El objetivo general de este proyecto es el diseño e implementación de una red social para motoristas en la que puedan compartir rutas con el resto de usuarios y organizar eventos, principalmente de salidas en grupo con otros motoristas.

Para el correcto desarrollo del objetivo general y poder conseguir así el resultado final deseado se han marcado una serie de subobjetivos:

- Análisis previo del estado actual del mercado. Esto es, buscar y estudiar las aplicaciones y redes sociales disponibles actualmente en el sector. Además de investigar y explorar el uso que los motoristas hacen de estas herramientas. De esta manera se podrá detectar con exactitud las limitaciones existentes en los productos actuales que hacen que el objetivo final del usuario no se cumpla de la manera más sencilla y eficiente posible.

- El siguiente objetivo es hacer un análisis de las necesidades y los propósitos de los motoristas. Con esto se quiere poder identificar funcionalidades inexistentes en los productos actuales y estudiar su incorporación en la aplicación móvil que se quiere presentar.

- Un buen diseño debe tener un carácter iterativo en el que siempre se llega al final de un ciclo cuando se da a conocer el producto al usuario esperando una feedback o retroalimentación a partir de la cual se empieza la siguiente iteración en el desarrollo. Es por esto que, finalmente el último objetivo es documentar de una manera clara y eficiente los pasos a seguir en el futuro para modificar o añadir utilidades y funcionalidades en la aplicación a medida que se detecten nuevas necesidades en los usuarios finales.

3. Antecedentes

En este apartado de la memoria estudiaremos el estado del arte actual en relación con el proyecto que se presenta. Además introduciremos brevemente las tecnologías utilizadas en el desarrollo de la aplicación.

3.1. Herramientas similares

A continuación se exploran aquellas herramientas existentes previamente al desarrollo de este proyecto.

Para llevar a cabo éste proceso de análisis se han consultado diversas publicaciones del mundo del motor además de haberse realizado pequeñas encuestas a motoristas sobre el sistema que utilizan normalmente para organizarse.

Existen actualmente en el mundo de las aplicaciones móviles varias redes sociales con distintos públicos, pero la más popular de todas es Facebook. Esta red social cuenta entre otros servicios con los siguientes:

- Lista de amigos: En ella el usuario puede agregar a cualquier persona que esté registrada en el sistema mediante un sistema de invitación de amistad. Para ello Facebook posee una herramienta de búsqueda de personas así como de sugerencia de posibles personas que conozcas.
- Grupos: Se trata de una utilidad que permite reunir a personas con intereses comunes y publicar en ese ámbito videos, fotos, mensajes, etc.
- Eventos: Son un tipo de publicación disponible para los usuarios del sistema en el que se puede invitar a otros usuarios registrados a participar en un evento específico. En este tipo de publicación se puede definir además de los invitados las fechas de inicio y de fin del evento.

Las funcionalidades detalladas anteriormente hacen que esta red social sea la herramienta más utilizada por los motoristas para organizar salidas con sus motos y compartir fotografías de estos viajes. Para ello, normalmente se crea un grupo en el que se van publicando eventos con sus respectivas fechas y posteriormente se van añadiendo fotografías.

El principal inconveniente de este sistema es la ausencia de una publicación específica para las rutas. De manera que los usuarios deben recurrir a herramientas

externas al sistema para planear las rutas que van a tomar en las diferentes salidas. Ésta es una de las carencias que se pretende solventar durante el desarrollo de este proyecto.

En el momento en el que se realiza este análisis acaba de salir al mercado una aplicación específica para motoristas llamada WeRide. Esta aplicación acaba con la principal deficiencia de Facebook que es la falta de herramientas para publicar y compartir rutas, pero en general es una aplicación que no ofrece una buena experiencia de usuario. Las principales quejas de los usuarios son la lentitud general de la aplicación, la complejidad de uso de la misma y los fallos en la implementación que hacen que la aplicación se cierre inesperada y continuamente.

Figura 3.1. Captura de pantalla de WeRide

Después de una búsqueda y análisis de la aplicación se observa que no se ha desarrollado una aplicación nativa para cada plataforma y se llega a la conclusión de que este puede ser el principal motivo del desencanto del usuario con la aplicación ya que no sigue patrones de diseño específicos que hacen que la interfície de usuario sea inconsistente con aplicaciones de la misma plataforma. Esto es, que utiliza elementos interactivos que un usuario de Android o de iOS esté acostumbrado a ver en el resto de las aplicaciones del sistema.

Además el no utilizar tecnologías nativas de cada plataforma también puede plantear una serie de deficiencias técnicas, como por ejemplo la respuesta lenta del sistema de la que se quejan la mayoría de los usuarios.

Existen además algunos errores en el diseño de las funcionalidades de la aplicación. Un claro ejemplo de ésto es la opción de buscar rutas. Pongámonos en situación: un usuario entra en la sección de rutas y aprieta el botón de rutas cercanas. En este momento el sistema hace una petición a su servidor para obtener TODAS las rutas cercanas al usuario, lo que significa que si existen 600 rutas alrededor, se mostrarán todas de golpe. Esto conlleva un considerable tiempo de espera que muchas veces el usuario no está dispuesto a asumir, en el mejor de los casos. En el peor de los casos, el usuario puede tener una mala conexión a internet (debido a interferencias o lentitud en la red, por ejemplo) y quedarse esperando eternamente porque aunque se ha recibido una respuesta del servidor no se han recibido todas las rutas requeridas y se vuelve a lanzar la llamada al servidor continuamente intentando recibir un volumen de datos considerable.

Una solución sencilla e implementada en la mayoría de las aplicaciones móviles que tienen un listado largo de ítems que cargar consiste en pedir solo una parte de la lista, los primeros 20 ítems por ejemplo, y cuando el usuario llegue al ítem 19 empezar a pedir los 20 siguientes.

Con estrategias como esta se pretende mejorar la experiencia y satisfacción general del usuario en el uso de una aplicación móvil para organizar salidas y consultar rutas.

3.2. Tecnologías utilizadas

En este apartado de la memoria pasaremos a detallar las principales características técnicas de las diferentes librerías y tecnologías usadas en el desarrollo del proyecto, además de argumentar su elección para la implementación de esta aplicación.

3.2.1 Lenguaje y base de datos

Objective - C

Debido al carácter nativo de la implementación de la aplicación en el momento de desarrollo de este proyecto Objective-C es la opción recomendada desde Apple para realizar una aplicación iOS.

Este lenguaje es el principal usado para el desarrollo de software para OS X e iOS. Es un superconjunto del lenguaje de programación C con la adición de capacidades orientadas a objetos y un tiempo de ejecución dinámico. También añade soporte para la gestión de objetos gráficos.

Gestión de datos en iOS

Existen 4 opciones distintas para almacenar información en disco cuando desarrollamos una aplicación iOS:

- **NSUserDefaults**: Con esta clase provista por el sistema podemos almacenar cosas sencillas y simples que necesiten de un tipo de dato básico como por ejemplo Strings. Se podría usar por ejemplo para guardar preferencias de usuario.
- **Ficheros .plist**: Son ficheros de tipo XML donde se pueden almacenar medianas cantidades de información como por ejemplo algunos objetos serializados.
- **SQL Lite**: es un motor de base de datos que ofrecen los dispositivos móviles iOS y sobre el cual se pueden construir modelos completos de datos.
- **Core Data**: es un framework facilitado por Apple que a bajo nivel utiliza SQL Lite. Core Data es una solución completa de modelo de datos que permite diseñar virtualmente una base de datos relacional, codificar la creación y consulta de objetos de la mencionada base de datos y programar de forma muy sencilla la persistencia de los objetos en disco.

En este caso hemos optado por utilizar la clase NSUserDefaults simplemente para almacenar el identificador del usuario una vez se ha conectado a la aplicación. Para el modelo de datos de la aplicación se ha utilizado Core Data ya que nos facilita el manejo de objetos a alto nivel.

3.2.2 Librerías

EGOImage

Esta librería ofrece una subclase de la clase de Objective-C *UIImageView* llamada *EGOImageView*. La clase *UIImageView* nos permite establecerle una imagen a una vista para mostrarla por pantalla. La clase *EGOImageView* puede establecerse una URL la que se almacena una imagen en vez de la imagen directamente de manera que la librería se encarga de gestionar una conexión para conseguir la imagen alojada en la URL establecida y cargarla en la vista así como cachearla para sus próximos usos.

CHSlideController

Para poder obtener la navegación de menú lateral con diferentes opciones y que cada una de éstas te muestre un nuevo *UINavigationController* se ha utilizado la librería *CHSlideController*. Esta librería nos permite conseguir el comportamiento deseado de una manera muy sencilla ya que mediante reconocedores de gestos (*UIGestureRecognizers)* responde a los gestos de deslizar y anima las diferentes vistas (menú lateral y pantalla central de la aplicación) cuando los detecta.

FacebookSDK

En esta aplicación se quiere dar la opción al usuario de registrarse utilizando su cuenta de Facebook de manera que no tenga que volver a introducir sus datos personales. Con este propósito se ha utilizado la librería proporcionada por Facebook.

3.2.3 Otras tecnologías

Photoshop

Para crear algunos de los elementos de la interfaz gráfica de la aplicación se ha utilizado el editor de gráficos Photoshop. Se ha escogido este programa, en vez de otras soluciones gratuitas como por ejemplo GIMP, porque el conocimiento previo de la herramienta permitiría ahorrar el tiempo de aprendizaje de un editor nuevo pudiendo emplear así ese tiempo en el diseño y desarrollo de la aplicación.

OmniGraffle

Para crear el diagrama de clases se ha utilizado un producto de la compañía The Omni Group llamado OmniGraffle. Esta herramienta es un software muy potente para crear diagramas.

Xcode

Para poder desarrollar una aplicación para iOS de manera nativa en Objective-C, se ha utilizado el entorno de desarrollo integrado Xcode proporcionado por Apple. Este IDE trabaja cuenta con Interface Builder que es una herramienta gráfica para la creación de interfaces de usuario y que se ha utilizado en este proyecto para la creación y edición de los archivos .xib incluidos en la aplicación. Aunque no todas las vistas se han creado con la ayuda de esta herramienta ya que debido a la complejidad de alguna de ellas ha sido mucho más eficiente crearlas mediante programación.

4. Análisis

4.1 Diagrama de casos de uso

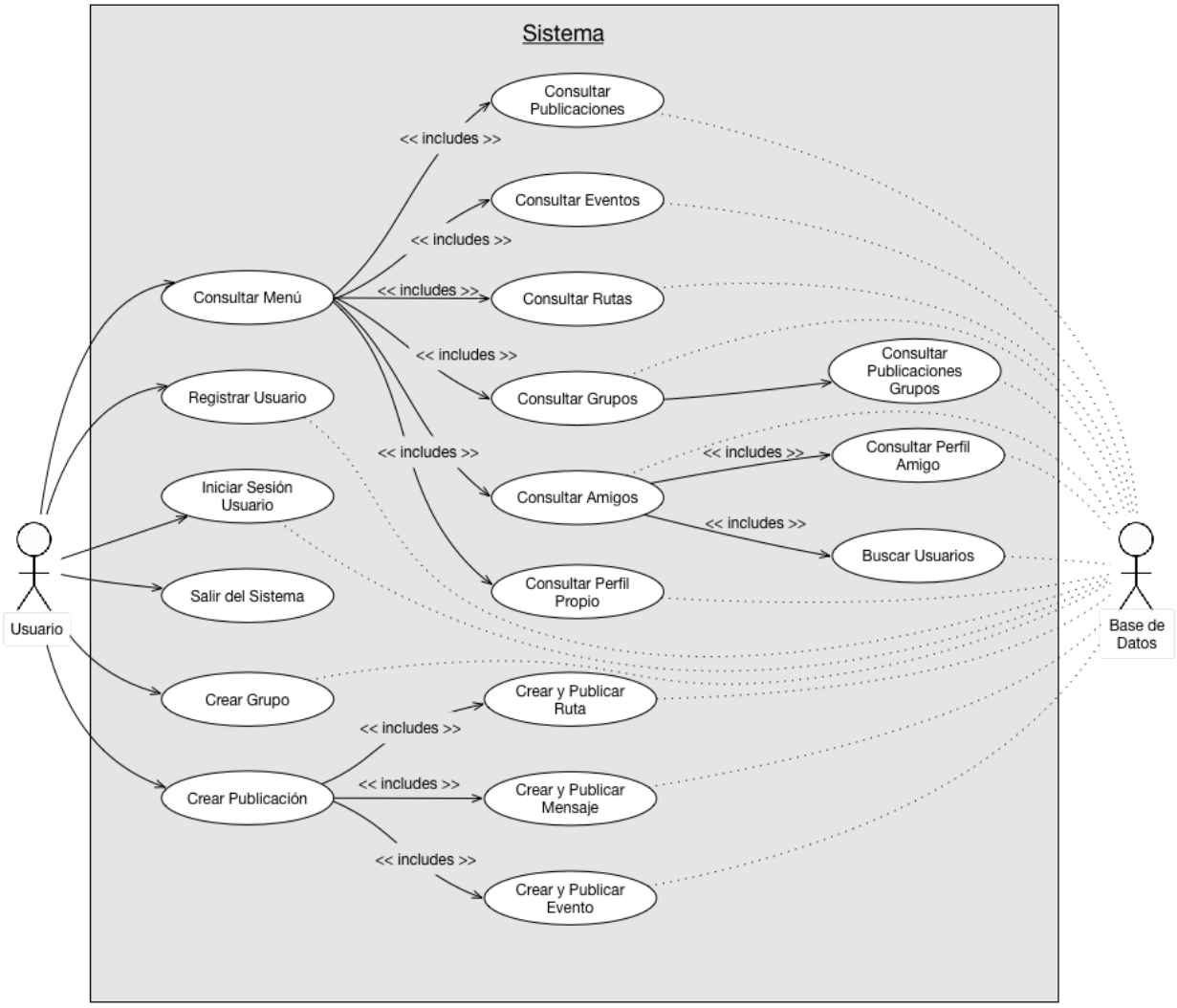

Figura 4.1. Diagrama de casos de uso

4.2 Descripción de los casos de uso

En este punto se describirán cada uno de los casos de uso que aparecen en el diagrama anterior (figura 4.1):

Nombre:	UC1 - Registrar Usuario
Actor:	Usuario
Descripción:	Proceso mediante el cual se registra un nuevo usuario en la base de datos.
Precondiciones:	El usuario se encuentra en la pantalla inicial de la aplicación.

Flujo Básico:

1. El usuario selecciona la opción de registrarse en la pantalla inicial de la aplicación.
2. El sistema presenta al usuario la pantalla de registro de usuario con su correspondiente formulario y una opción para registrarse con su cuenta de Facebook.
3. El usuario introduce su nombre y apellidos, su dirección de correo electrónico y la contraseña elegida.
4. El usuario confirma el registro.
5. El sistema comprueba los datos introducidos con la base de datos, registra al usuario e inicia su sesión.

Flujo Alternativo:

3a1. El usuario selecciona la opción de "Facebook" para registrarse.
3a2. El sistema muestra una pantalla controlada por Facebook para que el usuario se registre con su cuenta.
3a3. El usuario introduce sus credenciales de Facebook.

5a1. El sistema detecta que los datos introducidos son incorrectos, no registra al usuario y le muestra un mensaje de error.

Postcondiciones:	El usuario se encuentra en la pantalla de inicio de de su sesión registrado y con la sesión iniciada en el sistema.

Nombre:	UC2 - Iniciar Sesión Usuario
Actor:	Usuario
Descripción:	Proceso mediante el cual se autentifican los datos de un usuario en la base de datos.
Precondiciones:	El usuario se encuentra en la pantalla inicial de la aplicación.

Flujo Básico:

1. El usuario selecciona la opción "Inicia sesión" de la pantalla inicial de la aplicación.
2. El sistema presenta al usuario la pantalla de inicio de sesión con su correspondiente formulario y una opción para iniciar sesión con su cuenta de Facebook.
3. El usuario introduce su dirección de correo electrónico y su contraseña.
4. El usuario confirma el inicio de sesión.
5. El sistema comprueba los datos introducidos con la base de datos e inicia su sesión.

Flujo Alternativo:

3a1. El usuario selecciona iniciar sesión con "Facebook".
3a2. El sistema muestra una pantalla controlada por Facebook para que el usuario inicie sesión con su cuenta.
3a3. El usuario introduce sus credenciales de Facebook.

5a1. El sistema detecta que los datos introducidos son incorrectos, no inicia sesión y le muestra un mensaje de error al usuario.

Postcondiciones:	Se ha iniciado la sesión del usuario y éste se encuentra en la pantalla de inicio de su sesión.

Nombre:	UC3
Actor:	Usuario
Descripción:	Proceso mediante se le muestra al usuario el menú de la aplicación.
Precondiciones:	El usuario ha iniciado sesión en el sistema.

Flujo Básico:

1. El usuario selecciona la opción de menú.
2. El sistema muestra al usuario el menú.

Flujo Alternativo:

Postcondiciones:	El usuario se encuentra en la pantalla del menú.

Nombre:	UC4 - Consultar Publicaciones
Actor:	Usuario
Descripción:	Proceso mediante el cual se le muestran al usuario las últimas publicaciones que han hecho en su red de amistades y grupos así como las publicadas por él mismo.
Precondiciones:	El usuario se encuentra en la pantalla del menú.
Flujo Básico: 1. El usuario selecciona la opción "Inicio". 2. El sistema esconde el menú, pide a la base de datos las últimas publicaciones y se las muestra al usuario en la pantalla de inicio.	
Flujo Alternativo:	
4a1.El sistema esconde el menú, pide a la base de datos las últimas publicaciones y ésta no le devuelve ninguna. 4a2.El sistema muestra al usuario un mensaje de que no hay ninguna publicación disponible.	
Postcondiciones:	El usuario se encuentra en la opción de "Inicio" de la aplicación donde se muestran las últimas publicaciones.

Nombre:	UC5 - Consultar Eventos
Actor:	Usuario
Descripción:	Proceso mediante el cual se le muestran al usuario los últimos eventos que han hecho en su red de amistades y grupos así como los publicadas por él mismo.
Precondiciones:	El usuario se encuentra en la pantalla del menú.
Flujo Básico: 1. El usuario selecciona la opción "Eventos". 2. El sistema esconde el menú, pide a la base de datos los últimos eventos y se los muestra al usuario en la pantalla de eventos.	
Flujo Alternativo:	
4a1.El sistema esconde el menú, pide a la base de datos los últimos eventos y ésta no le devuelve ninguno. 4a2.El sistema muestra al usuario un mensaje de que no hay ningún evento publicado.	
Postcondiciones:	El usuario se encuentra en la opción de "Eventos" de la aplicación donde se muestran los últimas eventos.

Nombre:	UC6 - Consultar Rutas
Actor:	Usuario
Descripción:	Proceso mediante el cual se le muestran al usuario las últimos rutas que se han publicado. Las rutas se mostrarán por orden de proximidad al usuario.
Precondiciones:	El usuario se encuentra en la pantalla del menú.
Flujo Básico:	
1. El usuario selecciona la opción “Rutas”. 2. El sistema esconde el menú, pide a la base de datos las últimas rutas ordenadas por proximidad y se las muestra al usuario en la pantalla de rutas.	
Flujo Alternativo:	
4a1.El sistema esconde el menú, pide a la base de datos las últimas rutas y ésta no le devuelve ninguna. 4a2.El sistema muestra al usuario un mensaje de que no hay ninguna ruta publicada.	
Postcondiciones:	El usuario se encuentra en la opción de “Rutas” de la aplicación donde se muestran las últimas rutas ordenadas por proximidad.

Nombre:	UC7 - Consultar Grupos
Actor:	Usuario
Descripción:	Proceso mediante el cual se le muestran al usuario los grupos a los que pertenece.
Precondiciones:	El usuario se encuentra en la pantalla del menú.
Flujo Básico:	
1. El usuario selecciona la opción “Grupos”. 2. El sistema esconde el menú, pide a la base de datos los grupos a los que pertenece el usuario y se los muestra en la pantalla de grupos.	
Flujo Alternativo:	
4a1.El sistema esconde el menú, pide a la base de datos los grupos a los que pertenece el usuario y ésta no le devuelve ninguno. 4a2.El sistema muestra al usuario un mensaje de que aún no pertenece a ningún grupo.	
Postcondiciones:	El usuario se encuentra en la opción de “Grupos” de la aplicación donde se muestran los grupos a los que pertenece el usuario.

Nombre:	UC8 - Consultar Publicaciones Grupo
Actor:	Usuario
Descripción:	Proceso mediante el cual se le muestran al usuario las publicaciones de un grupo seleccionado.
Precondiciones:	El usuario se encuentra en la pantalla de grupos que no está vacía
Flujo Básico:	
1. El usuario selecciona un grupo. 2. El sistema muestra una pantalla nueva con la información del grupo seleccionado y pide a base de datos las últimas publicaciones del grupo y las muestra en la pantalla.	
Flujo Alternativo:	
2a1.El sistema muestra una pantalla nueva con la información del grupo seleccionado y pide a base de datos las últimas publicaciones del grupo, ésta no le devuelve nada y el sistema muestra un mensaje de que no hay publicaciones disponibles en ese grupo.	
Postcondiciones:	El usuario se encuentra en la opción de “Grupos” de la aplicación y se le muestra por pantalla la información y las publicaciones de un grupo concreto.

Nombre:	UC9 - Crear Grupo
Actor:	Usuario
Descripción:	Proceso mediante el cual el usuario crea un grupo nuevo.
Precondiciones:	El usuario ha iniciado sesión en el sistema.

Flujo Básico:

1. El usuario selecciona la opción de crear grupo.
2. El sistema muestra una pantalla con un formulario para crear un nuevo grupo.
3. El usuario selecciona una foto de su galería para usar como avatar del grupo.
4. El usuario introduce un nombre para el grupo.
5. El usuario selecciona la opción de añadir miembros al grupo.
6. El sistema consulta los usuarios disponibles en base de datos y los muestra por pantalla.
7. El usuario selecciona a todos los usuarios que quiere añadir al grupo y vuelve atrás.
8. El sistema actualiza el número de usuarios añadidos al grupo.
9. El usuario confirma la creación del grupo.
10. El sistema comprueba con la base de datos que los datos introducidos son correctos, crea el grupo en la base de datos y vuelve a mostrar la pantalla del listado de grupos con el grupo recién creado añadido.

Flujo Alternativo:

3a1.El usuario realiza una foto con la cámara de su dispositivo para usar como avatar del grupo.

6a1.El sistema consulta los usuarios disponibles en base de datos y ésta no le devuelve ninguno.
6a2.El sistema muestra un mensaje de que no hay usuarios disponibles.

10a1.El sistema detecta un error en los datos al consultarlo con la base de datos y le muestra al usuario un mensaje de error.

Postcondiciones:	El usuario se encuentra en la pantalla de grupos con el grupo que acaba crear en la lista.

Nombre:	UC10 - Consultar Amigos
Actor:	Usuario
Descripción:	Proceso mediante el cual se le muestran al usuario los amigos que tiene.
Precondiciones:	El usuario se encuentra en la pantalla del menú.
Flujo Básico:	
1. El usuario selecciona la opción “Amigos”. 2. El sistema esconde el menú, pide a la base de datos los amigos del usuario y se los muestra en la pantalla de amigos.	
Flujo Alternativo:	
4a1.El sistema esconde el menú, pide a la base de datos los amigos del usuario y ésta no le devuelve ninguno. 4a2.El sistema muestra al usuario un mensaje de que aún no tiene amigos.	
Postcondiciones:	El usuario se encuentra en la opción de “Amigos” de la aplicación donde se muestran los amigos que tiene el usuario.

Nombre:	UC11 - Consultar Perfil Amigo
Actor:	Usuario
Descripción:	Proceso mediante el cual se le muestra al usuario el perfil de un amigo seleccionado.
Precondiciones:	El usuario se encuentra en la pantalla de amigos que no está vacía.
Flujo Básico:	
1. El usuario selecciona un amigo. 2. El sistema consulta la información del amigo seleccionado con la base de datos y la muestra en una pantalla nueva.	
Flujo Alternativo:	
2a1.El sistema consulta la información del amigo seleccionado con la base de datos y ésta no le devuelve nada.	
Postcondiciones:	El usuario se encuentra en la opción de “Amigos” de la aplicación y se le muestra por pantalla la información de un amigo concreto.

Nombre:	UC12 - Buscar Usuarios
Actor:	Usuario
Descripción:	Proceso mediante el cual se le muestra al usuario un listado de usuarios que contengan el texto buscado en su nombre, apellidos o dirección de correo electrónico.
Precondiciones:	El usuario se encuentra en la pantalla de amigos.
Flujo Básico:	
1. El usuario selecciona la opción de búsqueda, e introduce un texto. 2. El sistema pide a la base de datos los usuarios que contengan el texto introducido por el usuario y se los muestra al usuario en la pantalla de amigos.	
Flujo Alternativo:	
2a1. El sistema pide a la base de datos los usuarios que contengan el text introducido por el usuario y ésta no le devuelve ninguno. 2a2. El sistema muestra al usuario un mensaje de que no ha encontrado ningún usuario con el texto introducido.	
Postcondiciones:	El usuario se encuentra en la opción de "Amigos" de la aplicación donde se muestra una lista de usuarios que contienen el texto buscado.

Nombre:	UC13 - Consultar Perfil Propio
Actor:	Usuario
Descripción:	Proceso mediante el cual se le muestra al usuario su perfil.
Precondiciones:	El usuario se encuentra en la pantalla del menú.
Flujo Básico:	
1. El usuario selecciona la opción de ver su perfil. 2. El sistema consulta con la base de datos la información del usuario y la muestra en una pantalla nueva.	
Flujo Alternativo:	
Postcondiciones:	El usuario se encuentra en la pantalla de su perfil.

Nombre:	UC14 - Escoger Tipo Publicación
Actor:	Usuario
Descripción:	Proceso mediante el cual el usuario escoge un tipo de publicación.
Precondiciones:	El usuario ha iniciado sesión en el sistema.
Flujo Básico:	
1. El usuario selecciona la opción de crear publicación. 2. El sistema muestra un popup con la lista de publicaciones disponibles. 3. El usuario selecciona el tipo de publicación deseada. 4. El sistema muestra una pantalla nueva con el formulario de la publicación escogida.	
Flujo Alternativo:	
Postcondiciones:	El usuario se encuentra en la pantalla de publicación del tipo seleccionado.

Nombre:	UC15 - Publicar Mensaje
Actor:	Usuario
Descripción:	Proceso mediante el cual el usuario publica un mensaje nuevo en el sistema.
Precondiciones:	El usuario se encuentra en la pantalla de publicación de mensaje.
Flujo Básico:	
1. El usuario introduce el mensaje deseado. 2. El sistema habilita la opción de "Publicar". 3. El usuario selecciona la opción de "Publicar". 4. El sistema crea el mensaje introducido en la base de datos, cierra la pantalla con el formulario y vuelve a la pantalla de inicio con el mensaje añadido en la lista de publicaciones.	
Flujo Alternativo:	
1a1. El usuario selecciona la opción de añadir imagen y añade una imagen al mensaje. 4a1. El sistema intenta crear el mensaje introducido en la base de datos pero no lo consigue y le muestra al usuario un mensaje de error.	
Postcondiciones:	El usuario se encuentra en la pantalla de inicio con el mensaje publicado añadido en la lista.

Nombre:	UC16 - Publicar Evento
Actor:	Usuario
Descripción:	Proceso mediante el cual el usuario publica un evento nuevo en el sistema.
Precondiciones:	El usuario se encuentra en la pantalla de publicación de evento.
Flujo Básico:	
1. El usuario introduce el título del evento. 2. El usuario selecciona la opción de escoger fecha del formulario. 3. El sistema muestra una nueva pantalla con un calendario para que el usuario escoja la fecha del evento. 4. El usuario escoge la fecha del evento y vuelve atrás. 5. El sistema actualiza el formulario con la fecha introducida por el usuario. 6. El usuario selecciona la opción de añadir invitados del formulario. 7. El sistema muestra una nueva pantalla con una lista de los grupos a los que pertenece el usuario y sus amigos. 8. El usuario selecciona a los invitados deseados y vuelve atrás. 9. El sistema actualiza el formulario con el número de invitados seleccionados. 10. El sistema habilita la opción de “Publicar”. 11. El usuario selecciona la opción de "Publicar”. 12. El sistema crea el evento introducido en la base de datos, cierra la pantalla con el formulario y vuelve a la pantalla de inicio con el evento añadido en la lista de publicaciones.	
Flujo Alternativo:	
11a1.El usuario introduce información sobre el evento. 11b1.El usuario selecciona la opción de escoger rutas del formulario. 11b1.El sistema muestra una nueva pantalla con la lista de rutas disponibles. 11b1.El usuario selecciona las rutas deseadas y vuelve atrás. 11b1.El sistema actualiza el formulario con el número de rutas seleccionadas. 12a1.El sistema intenta crear el evento introducido en la base de datos pero no lo consigue y le muestra al usuario un mensaje de error.	
Postcondiciones:	El usuario se encuentra en la pantalla de inicio con el evento publicado añadido en la lista.

Nombre:	UC17 - Publicar Ruta
Actor:	Usuario
Descripción:	Proceso mediante el cual el usuario publica una ruta nueva en el sistema.
Precondiciones:	El usuario se encuentra en la pantalla de publicación de ruta.
Flujo Básico:	
1. El usuario introduce el punto de inicio de la ruta. 2. El usuario introduce el punto final de la ruta. 3. El usuario introduce el número de kilómetros de la ruta. 4. El usuario introduce el enlace al mapa de la ruta. 5. El sistema habilita la opción de "Publicar". 6. El usuario selecciona la opción de "Publicar". 7. El sistema crea la ruta introducida en la base de datos, cierra la pantalla con el formulario y vuelve a la pantalla de inicio con la ruta añadida en la lista de publicaciones.	
Flujo Alternativo:	
4a1. El usuario introduce información sobre el evento. 7a1. El sistema intenta crear la ruta introducida en la base de datos pero no lo consigue y le muestra al usuario un mensaje de error.	
Postcondiciones:	El usuario se encuentra en la pantalla de inicio con la ruta publicada añadida en la lista.

Nombre:	UC18 - Salir del Sistema
Actor:	Usuario
Descripción:	Proceso mediante el cual se cierra la sesión del usuario.
Precondiciones:	El usuario se encuentra en la pantalla del menú.
Flujo Básico:	
1. El usuario selecciona la opción de "Cerrar Sesión". 2. El sistema cierra la sesión del usuario y le muestra la pantalla inicial de la aplicación.	
Flujo Alternativo:	
Postcondiciones:	El usuario se encuentra en la pantalla inicial de la aplicación sin sesión iniciada.

4.3 Modelo de dominio

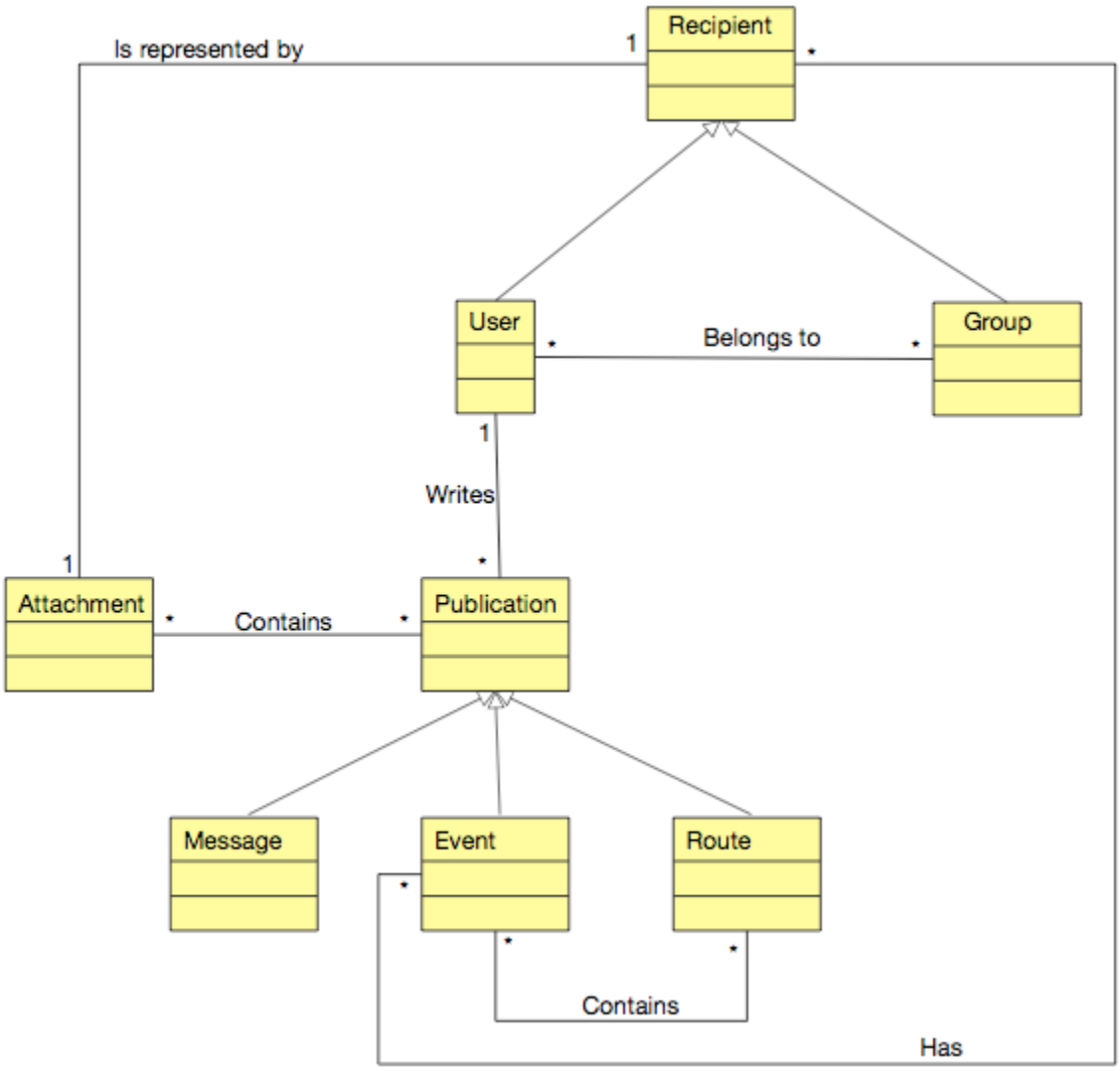

Figura 4.2. Modelo de Dominio

5. Diseño

5.1 Patrones de diseño usados

En esta sección describiremos los patrones de diseño utilizados más relevantes utilizados para desarrollar el proyecto.

5.1.1. MVC

En primer lugar tal y como hemos aprendido a lo largo de la carrera el más básico de todos los patrones es el patrón Model View Controller. Se ha dividido el proyecto en tres grandes bloques:

- **Model**: Contiene todas las clases (beans) referentes al modelo de datos de la aplicación, así como el archivo con la estructura e interrelaciones del modelo y una clase *ContentProvider* que se encarga de manejar los datos y el contenido necesario dentro de la aplicación. Incluimos aquí también la clase *UserDefaults* encargada de guardar qué usuario que ha iniciado sesión en el dispositivo actual.
- **View**: Contiene todas las clases referentes a la interfaz de usuario, que en nuestro caso serán archivos de tipo .xib
- **Controller**: Contiene todas las clases encargadas de la lógica de la aplicación y del control de la interfaz de usuario.

5.1.2. Singleton

El patrón de diseño Singleton se asegura de que existe tan solo una instancia de una clase dada y de que haya un punto de acceso global a esa instancia desde el resto de clases.

Hay algunos casos en los que tiene sentido y es más que recomendable tener una sola instancia de una clase concreta. Un buen ejemplo sería una clase de configuración del sistema. Teniendo sólo una instancia de esta clase es mucho más

sencillo y fiable hacer su implementación thread-safe a la hora de sobreescribir un archivo de configuración compartido.

En este proyecto se ha utilizado este patrón de diseño con la clase *FacebookHelper* por ejemplo, la cual cosa permite que podamos acceder a todas las funciones de manejo de la sesión de Facebook desde cualquier punto de la aplicación además de asegurarnos de un acceso sincronizado a sus funciones.

5.1.3. Facade

El patrón de diseño Facade (Fachada) consiste en proporcionar una sola interfaz al usuario (programador) de un complejo subsistema. De esta manera en vez de exponer al usuario a un conjunto de clases y sus respectivas APIs, le exponemos a una sola API sencilla y unificada.

El usuario de esta API tiene que ser completamente ajeno a la complejidad que pueda haber debajo, de manera que si tomamos como ejemplo una API encargada de proporcionar los datos al sistema, la clase *ContentProvider* en este proyecto, tendremos la ventaja de que si en algún punto nuestro backend o bien nuestro sistema de base de datos cambia, el sistema será totalmente ajeno a este cambio y podrá seguir utilizando los métodos de la API *ContentProvider*, mientras su implementación interna se habrá transformado adaptándose a los cambios.

En el caso de la clase *ContentProvider* se ha utilizado además del patrón Facade, el patrón Singleton para poder tener acceso a los datos desde cualquier punto de la aplicación.

5.1.4. Decorator

El patrón de diseño Decorator solventa la necesidad de añadir dinámicamente una nueva funcionalidad a un objeto, de manera que podemos evitar la creación de subclases que heredan de la primera solo para incorporar una nueva funcionalidad.

En Objective-C existen dos maneras diferentes, muy comunes, de implementar este patrón de diseño, que son las *Categories* y los *Delegates*. Hablaremos de ellas a continuación.

Categories

Una *Category* de una clase es un mecanismo muy poderoso que te permite añadir métodos y funciones nuevas a clases ya existentes sin la necesidad de crear nuevas subclases.

Los nuevos métodos creados se añaden en tiempo de compilación y pueden ser ejecutados como cualquier otro método de la clase extendida.

En este proyecto se han usado varias *Categories* para clases como *NSString* en la que se han incorporado dos métodos, uno para comprobar que el string tiene el formato correcto de un correo electrónico y otro método para encriptar el string por ejemplo.

Además usando Core Data resulta muy útil crear *Categories* de clases del modelo que necesiten algún método adicional, de manera que si tenemos que cambiar la clase del modelo no tenemos que volver a implementar el método de la categoría cada vez.

Delegates

La otra implementación del patrón de diseño *Decorator* establecida en Objective-C consiste en el patrón *Protocolo - Delegado*. Éste es un mecanismo mediante el cual un objeto actúa en nombre de, o en coordinación con otro objeto.

Un ejemplo muy aclaratorio y explicativo de este patrón es el utilizado en la clase *UITableView* de Objective-C. Esta clase tiene un conjunto de métodos, llamado protocolo, que implementa el delegado de la clase. Por ejemplo la clase *UITableView* no puede saber cuantas filas tendrá la tabla ya que este dato es un valor específico de la aplicación, así pues el delegado de la clase será el encargado de calcular este valor y tendrá que implementar el método `tableView:numberOfRowsInSection:` del protocolo de la clase UITableView para que esta sepa el número de filas que tiene que insertar en cada sección de la tabla. Esta estrategia permite que la clase *UITableView* sea totalmente independiente de los datos que muestra.

En este proyecto se ha utilizado este patrón de diseño en varias ocasiones, una de ellas ha sido con la clase *DatePicker* utilizada para que el usuario pueda escoger la fecha de un evento. En esta clase hemos creado un protocolo formado por el método `datePickerDidEndSelectionWithStartDate:endDate:allDayEvent:` que avisa al delegado de la clase (*PublishViewEvent*) de que el usuario ha seleccionado la fecha concreta. De esta manera el delegado puede actualizar su vista con los datos obtenidos.

5.1.5. Observer

El patrón de diseño Observer (Observador en español) define una dependencia del tipo “uno a n” entre objetos, de manera que cuando uno de los objetos cambia su estado, el resto de objetos dependientes es notificado de este cambio. La implementación usual requiere que un observador registre su interés en el estado de otro objeto, así cuando el estado cambia todos los objetos observadores son notificados del cambio.

En este proyecto se han usado las *Notifications* de Objective-C para implementar este patrón de diseño. Como ejemplo ilustrativo explicaremos que en la clase *StartupVC* de la aplicación se añadido como observador de la notificación “userLoggedIn” lanzada por la clase *SignInVC* cuando el usuario ha iniciado sesión en el sistema correctamente. De manera que una vez el usuario inicia sesión la clase *SignInVC* desaparece y es la clase *StartupVC* la encargada de presentar al usuario la pantalla principal de la aplicación.

5.2. Diagramas de Secuencia

A continuación se muestran aquellos diagramas de secuencia de los casos de uso explicados en el apartado 4.2 que conviene detallar gráficamente:

UC1 - Registrar Usuario

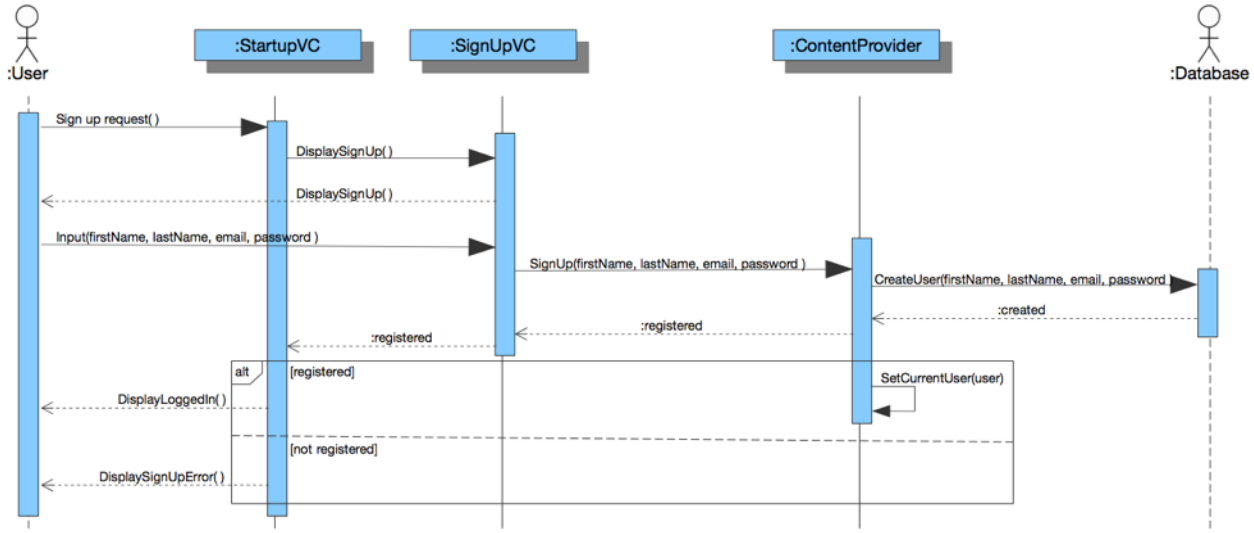

Figura 5.1. Diagrama de secuencia UC1 - Registrar Usuario

UC2 - Iniciar Sesión de Usuario

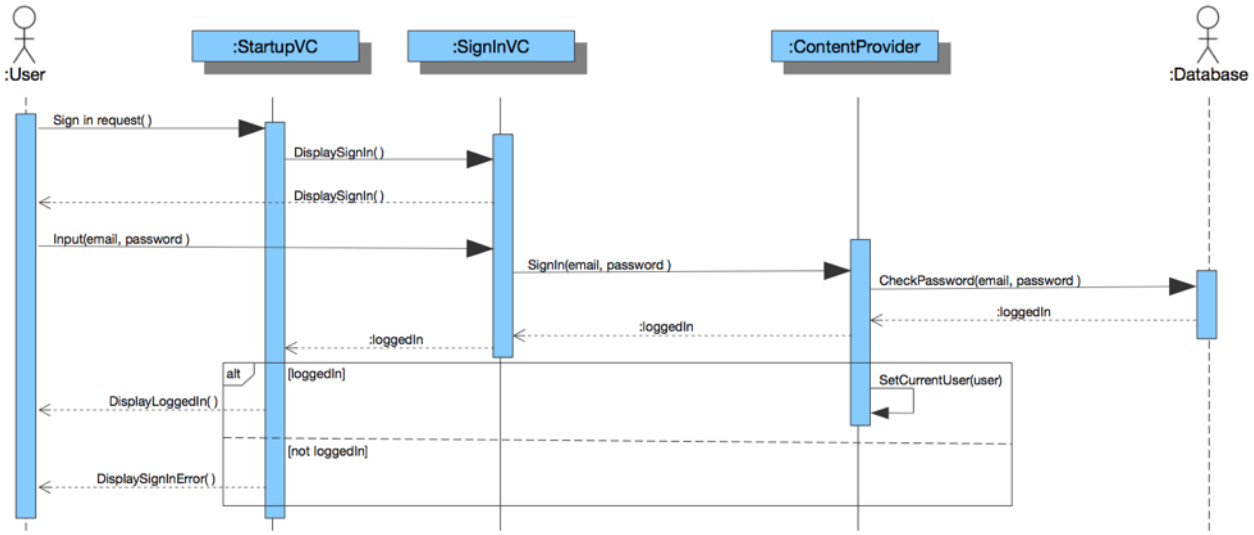

Figura 5.2. Diagrama de secuencia UC2 - Iniciar Sesión de Usuario

UC4 - Consultar Publicaciones

Este diagrama es muy similar en todos los casos de consulta de datos como son los siguientes casos de uso:

- UC5 - Consultar Eventos
- UC6 - Consultar Rutas
- UC7 - Consultar Grupos
- UC8 - Consultar Publicaciones Grupo
- UC9 - Consultar Amigos
- UC12 - Buscar Usuarios

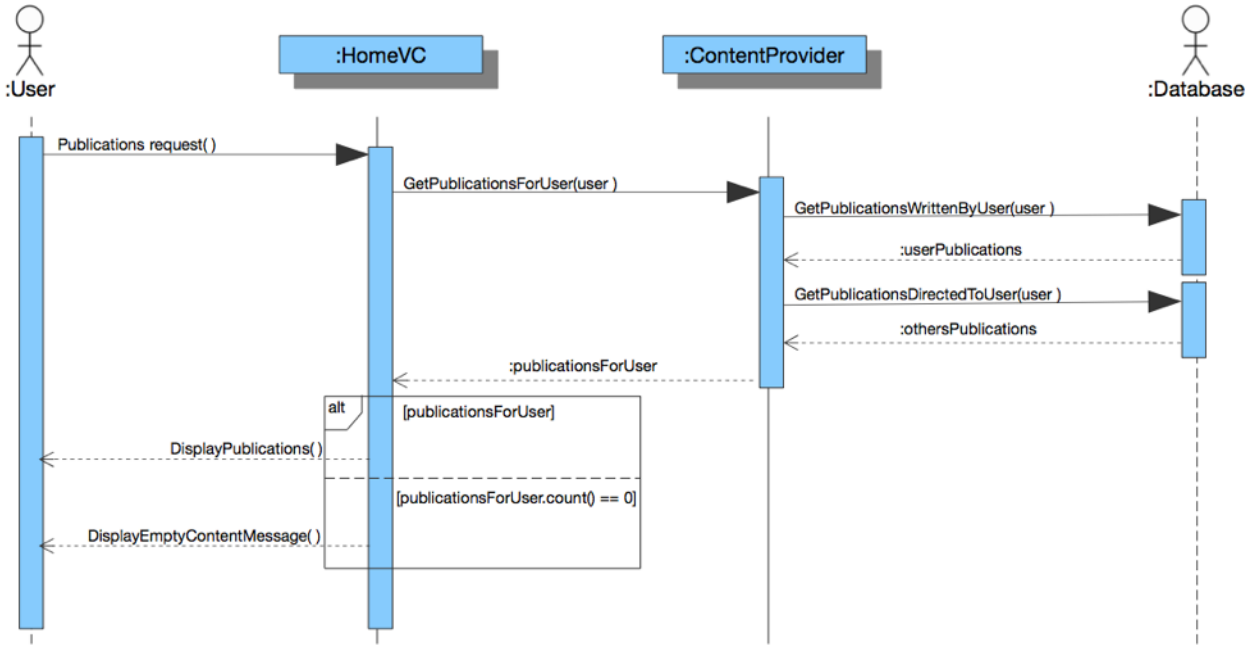

Figura 5.3. Diagrama de secuencia UC4 - Consultar Publicaciones

UC9 - Crear Grupo

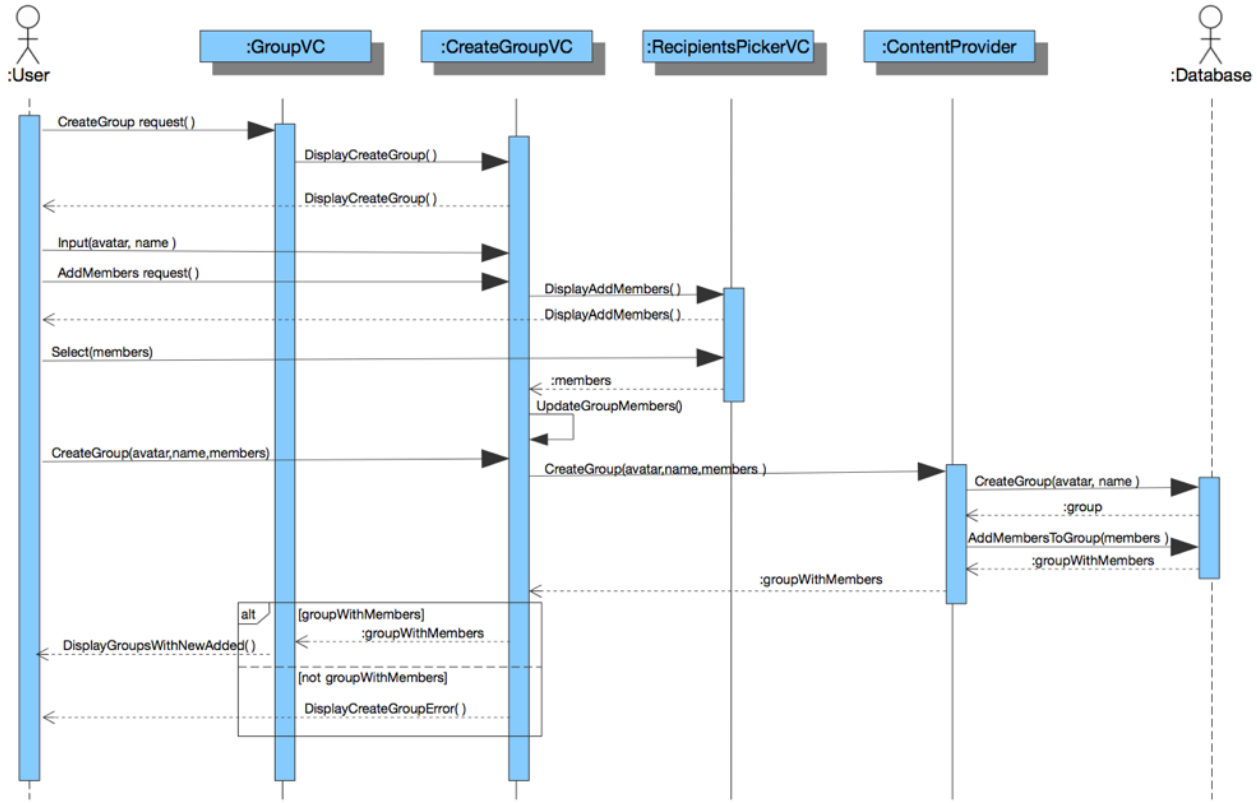

Figura 5.4. Diagrama de secuencia UC9 - Crear Grupo

UC11 - Consultar Perfil Amigo

Este caso de uso es muy similar al UC13 - Consultar Perfil Propio.

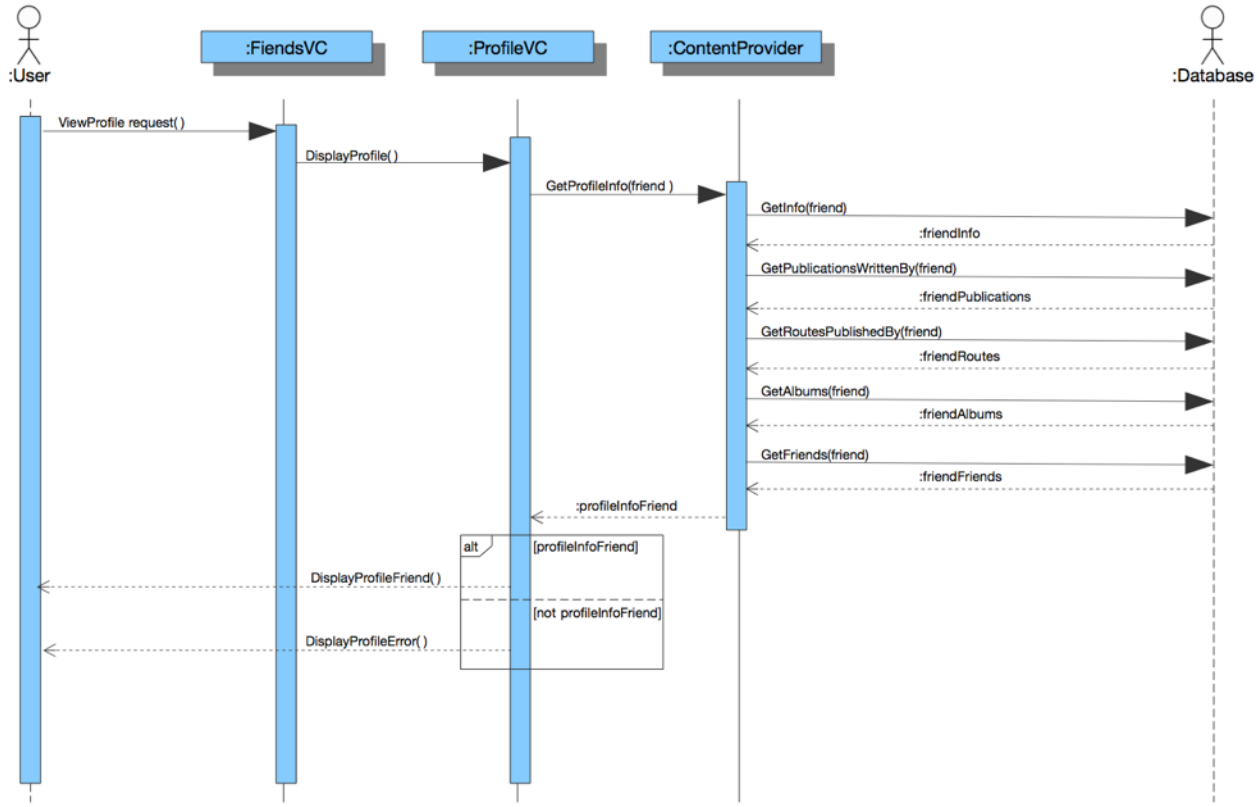

Figura 5.5. Diagrama de secuencia UC11 - Consultar Perfil Amigo

UC16 - Publicar Evento

Los casos de uso UC14 - Publicar Mensaje y UC16 - Publicar Ruta son muy similares a este.

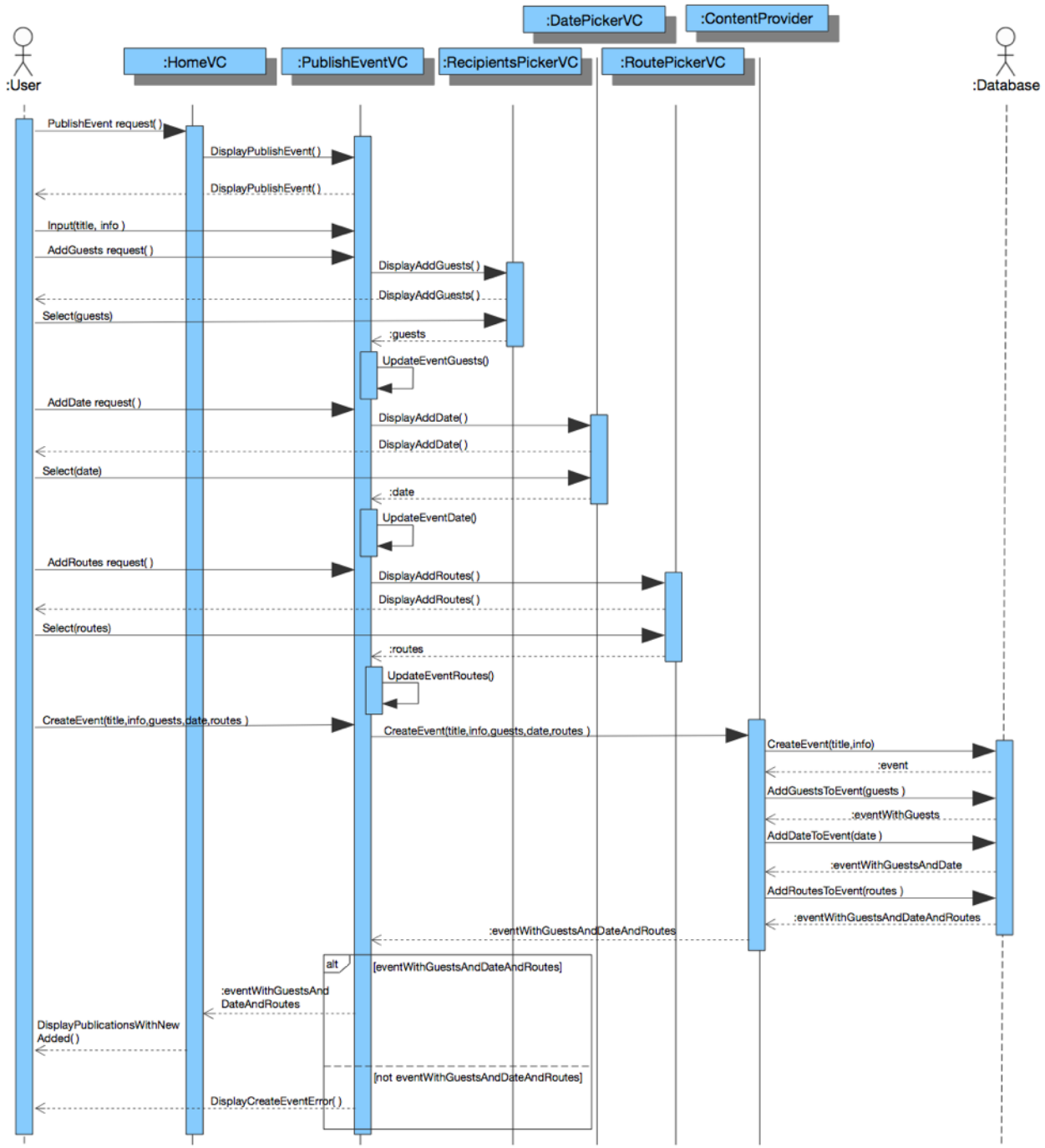

Figura 5.6. Diagrama de secuencia UC16 - Publicar Evento

5.3. Diagrama de Clases

He aquí una representación de las clases con sus variables y métodos más distintivos:

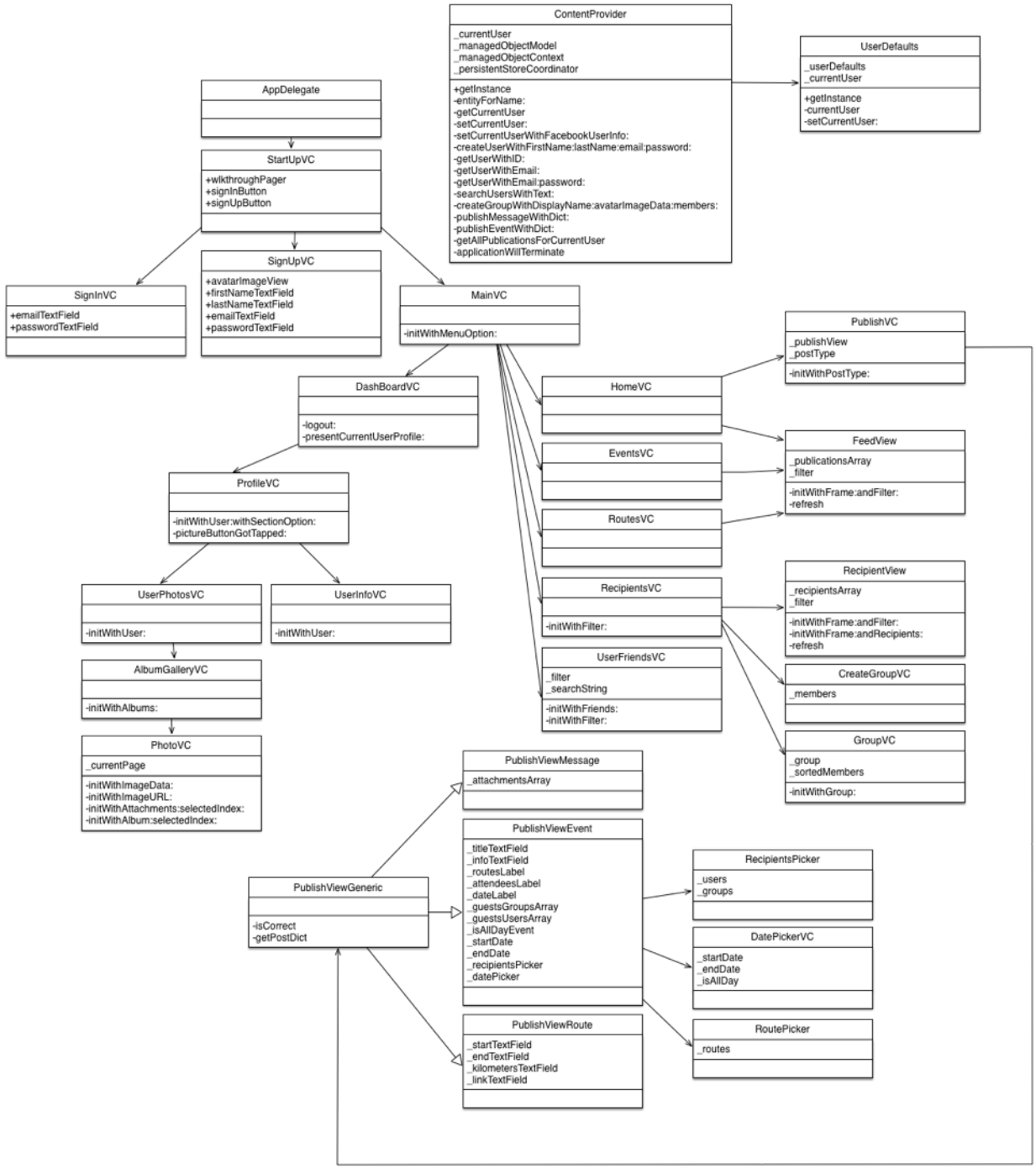

Figura 5.7. Diagrama de Clases

6. Implementación y resultados

Dentro de esta sección de la memoria se tratan diversos aspectos importantes relacionados con el desarrollo y la implementación de la aplicación móvil llamada BuenaRuta.

6.1. Arquitectura de la navegación dentro de la aplicación

Debido a la navegación de la aplicación la arquitectura de las clases ha sido una parte de la implementación muy importante en la que se ha pensado muy a fondo. Antes de entrar a hablar sobre este tema se definirá el concepto de *ViewController* para facilitar la explicación del resto de conceptos.

La clase *ViewController* es una clase especializada en la gestión de vistas. Por lo general, se debe crear un controlador de vista (*UIViewController*) para cada vista de una aplicación. El controlador contiene todos los atributos de la vista así como las acciones asociadas a éstos. Después de esta aclaración continuaremos con la explicación de la navegación de la arquitectura.

En primer lugar tenemos que para navegar entre los distintos *ViewControllers* de la aplicación se usan una serie de funciones que se repiten en todos ellos. Además se detectan una serie de patrones que se repiten a lo largo de la aplicación para personalizar la barra de navegación, ya sea añadir un título o botones a la izquierda o derecha de la barra.

Por todo esto se decide crear una clase *BaseVC* de la que heredaran el resto de *ViewControllers* de la aplicación. Esta clase tiene una serie de métodos para realizar las tareas descritas que se repetían una y otra vez en múltiples clases.

En segundo lugar se ha decidido crear una aplicación con un menú lateral que está escondido debajo de la pantalla principal de la aplicación y que el usuario verá haciendo un gesto de swipe hacia la derecha o bien apretando al botón del menú situado en la barra de navegación, de manera que el *ViewController* actual se moverá hacia la derecha dejando a la vista el menú existente debajo.

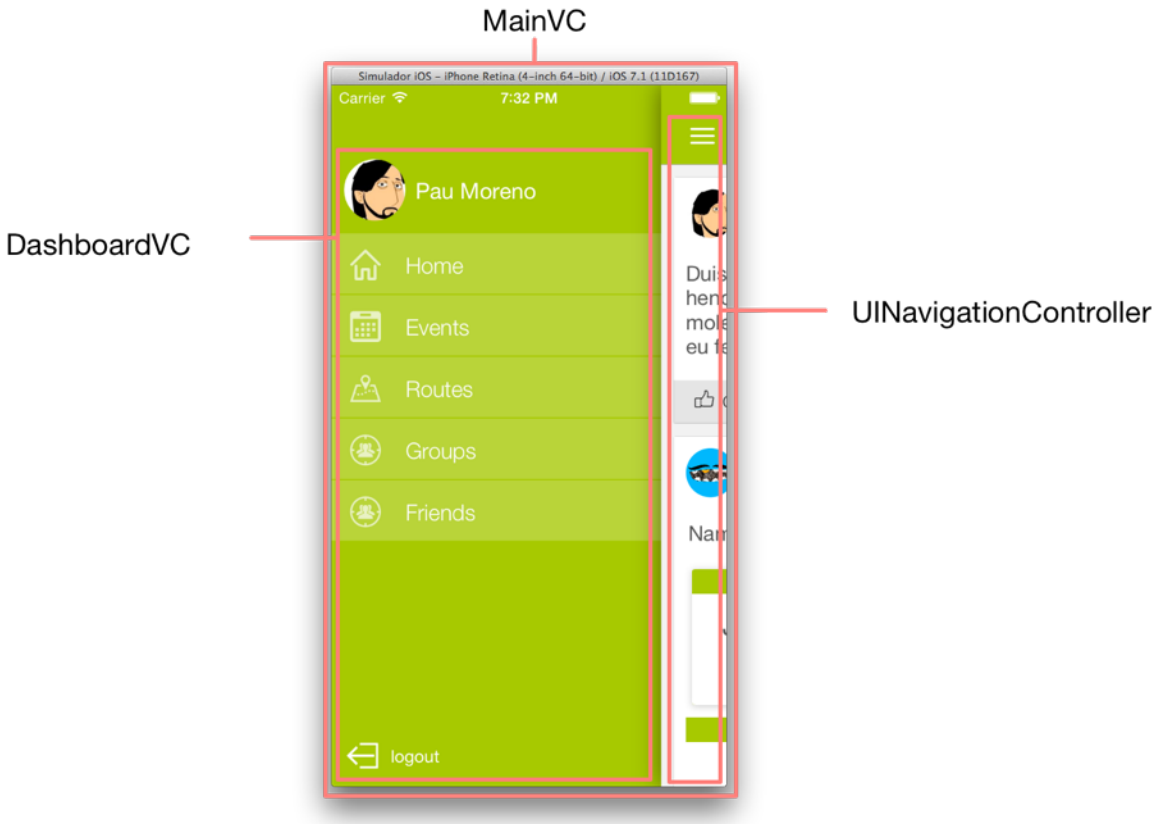

Figura 6.1. Navigation Diagram

Para conseguir este comportamiento se ha utilizado la librería *CHSlideController* mencionada en el punto 3.2.2. Lo que se ha hecho ha sido crear una clase *MainVC* que hereda de la clase *CHSlideController* comentada [Figura 6.1].

Esta clase tiene un objeto *DashBoardVC* [Figura 6.1] que será el menú de la aplicación o leftViewController. Además tiene cinco *UINavigationController* [Figura 6.1] con sus respectivos *ViewControllers*, uno para cada sección de la aplicación.

El view controller *DashBoardVC* está dividido en tres partes. La parte superior muestra la información del usuario actual, una botón con el avatar del usuario desde el que se puede acceder a su perfil, y una etiqueta con su nombre. La parte central es una tabla desde la que podemos navegar a las diferentes secciones de la aplicación. Y finalmente la parte superior tiene un botón mediante el cual el usuario podrá cerrar su sesión.

Pasaremos a explicar ahora cada una de las secciones que contiene el *MainVC*, para ver sus propiedades y navegación interna.

HomeNavC

Es la pantalla de inicio de la aplicación una vez el usuario ha iniciado sesión. Desde esta sección el usuario puede consultar las publicaciones más recientes de sus amigos y grupos, aquí están incluidos los tres tipos de publicaciones: mensajes, eventos y rutas.

Esto es posible gracias a que el view controller de esta sección cuenta con una vista llamada *FeedView* que hereda de la clase *UITableView* de Objective-C y que representa una tabla con diferentes celdas, una para cada publicación. Se han creado tres tipos de celdas diferentes:

- ***FeedCellGeneric***: es la clase base de todas las celdas correspondientes a la clase *FeedView* y se utiliza para mostrar las publicaciones de tipo mensaje.
- ***FeedCellEvent***: hereda de la clase *FeedCellGeneric* y se utiliza para mostrar las publicaciones de tipo evento.
- ***FeedCellRoute***: hereda de la clase *FeedCellGeneric* y se utiliza para mostrar las publicaciones de tipo ruta.

Para hacer más fácil el manejo de los diferentes tipos de celda se ha creado una clase *FeedCellFactory* que se encarga de crear una celda del tipo adecuado dada una publicación concreta.

Esta sección dispone también de un botón en la parte derecha de la barra de navegación para que el usuario pueda hacer sus publicaciones. Este botón presenta un pop up con los diferentes tipos de publicaciones. Al seleccionar un tipo de publicación, el *UINavigationController* presenta una nueva pantalla con un *PublishVC* que se encarga de la publicación correspondiente al tipo seleccionado. Una vez hecha la publicación si todo ha ido bien se desecha el *ViewController* de la publicación y se vuelve a mostrar la pantalla de inicio con la nueva publicación agregada a la lista de publicaciones recientes.

EventsNavC

Esta sección es la encargada de mostrar al usuario las publicaciones filtradas por el tipo evento. Su funcionamiento es muy similar al del *homeNavC*, con la diferencia de que al apretar el botón de crear una nueva publicación el *UINavigationController*

directamente presenta el *PublishVC* correspondiente al tipo evento, eliminando el pop up existente en el *homeNavC*.

RoutesNavC

En esta sección tenemos el view controller encargado de mostrar al usuario las rutas más cercanas. Una vez más su funcionamiento es muy similar al del *homeNavC*, eliminando el pop up para escoger el tipo de publicación existente en éste.

GroupsNavC

Esta sección muestra al usuario los grupos a los que pertenece gracias a su vista de tipo *RecipientView*. Esta vista igual que la *FeedView* hereda de la clase *UITableView*, pero en este caso cada celda representa un grupo.

Al seleccionar una de las celdas el *UINavigationController* crea un nuevo *ViewController* de tipo *GroupVC* que se contiene dos secciones en la pantalla, una encargada de mostrar la información del grupo (avatar, nombre y miembros) y la otra del tipo *FeedView* encargada de mostrar las publicaciones hechas en el grupo.

Además de la opción de entrar al detalle de un grupo el *groupsNavC* tiene un botón en la parte derecha de la barra de navegación que nos muestra un nuevo view controller de tipo *CreateGroupVC* diseñado para la creación de nuevos grupos. Su funcionamiento es muy similar al de las publicaciones ya que una vez que el usuario ha introducido todos los datos pertinentes y ha apretado el botón de crear, si todo va bien se desecha el *CreateGroupVC* y se vuelve a mostrar la lista de grupos con el nuevo grupo añadido.

FriendsNavC

En esta sección el usuario puede consultar su listado de amigos gracias a su view controller de tipo *UserFriendsVC*. Este view controller contiene una vista de búsqueda en la parte superior y una *UICollectionView* en la parte inferior.

La vista superior de búsqueda es una instancia de la clase *SearchView* y permite al usuario buscar a otros usuarios aunque aún no sean amigos dentro de la aplicación.

La *UICollectionView* es muy parecida a la *UITableView* con la diferencia de que los datos se pueden mostrar en una matriz, mientras que en la otra vista solo podemos mostrar una celda por fila. Esta vista se utiliza para mostrar los amigos del usuario actual. Hay que añadir que igual que pasaba con la tabla de grupos aquí cuando el usuario selecciona la celda de un usuario concreto se presenta un nuevo view controller con el perfil del usuario seleccionado.

ProfileVC

Por último el *ViewController* encargado de mostrar el detalle de un usuario. Este *ViewController* consta de una parte fija superior donde se muestra la foto de portada del usuario, su avatar y el nombre de su moto. Mientras que en la parte inferior se ha implementado una vista con cinco pestañas diferentes.

Para ello esta clase consta de cinco componentes distintos que procederemos a explicar a continuación.

UserInfoVC

Este view controller es el encargado de mostrar la información básica del usuario que es el nombre, una descripción y una pequeña *MapView* con la localización del usuario.

FeedView - General

En este componente se muestran las publicaciones realizadas por el usuario. Funciona igual que en el *homeNavC* explicado anteriormente.

FeedView - Rutas

En este componente se muestran las publicaciones de tipo ruta realizadas por el usuario. Una vez su funcionalidad es identica a la detallada en el *homeNavC*.

UserPhotosVC

Este *ViewController* se encarga de mostrar los álbumes del usuario. Aquí se nos presentan los diferentes albumes de fotos y al seleccionar alguno de ellos se abre un nuevo controlador para navegar entre todas las fotos del álbum.

UserFriendsVC

Por último tenemos este controlador que ya se ha explicado anteriormente ya que es el mismo elemento que aparece en el *MainVC* para controlar la sección de amigos del usuario actual.

6.2. Base de datos

En lo que respecta a los datos de la aplicación se ha decidido utilizar dos patrones de diseño de los mencionados en el apartado 5.1. En primer lugar se ha usado el patrón de diseño Facade creando una clase *ContentProvider* que manejará todos los datos de la aplicación, tanto en memoria (con el usuario actual por ejemplo), como en Core Data y en los *UserDefaults*. De esta manera siempre que se necesiten datos de la aplicación podemos recurrir a esta única clase. Además se ha utilizado también el patrón de diseño Singleton de manera que esta clase estará siempre disponible desde cualquier punto de la aplicación y se creará una sola instancia de ella para conseguir que la gestión de los datos sea más robusta en cuanto a los diferentes threads que quieran modificar los datos de una manera concurrente.

Pasaremos a explicar ahora el funcionamiento de Core Data. En primer lugar tenemos que recalcar que Core Data no es un motor de base de datos en sí, sinó que es un framework que nos ayuda a gestionar una base de datos de tipo SQLite.

Vamos a ver en primer lugar vamos definir cuales son los diferentes componentes de Core Data y para qué sirve cada uno de ellos. Para ello nos vamos a ayudar del siguiente esquema (figura 6.2) con los cuatro componentes:

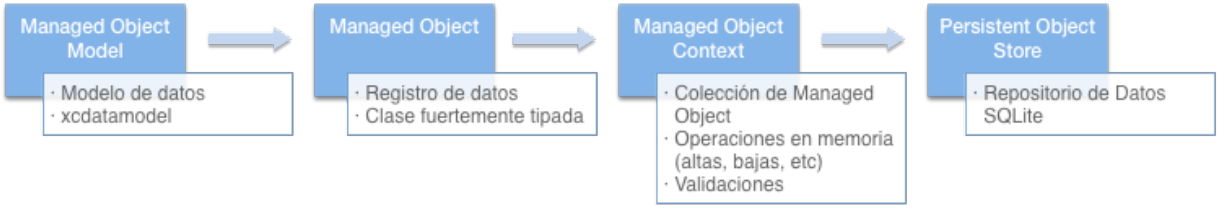

Figura 6.2. Esquema de componentes de Core Data

En primer lugar es imprescindible hablar del **Managed Object Model**, este objeto es una instancia de la clase *NSManagedObjectModel* y describe el esquema del modelo de datos a utilizar dentro de la aplicación. Este objeto se carga desde el archivo del modelo que contiene la descripción de todas las entidades con sus respectivos atributos y relaciones.

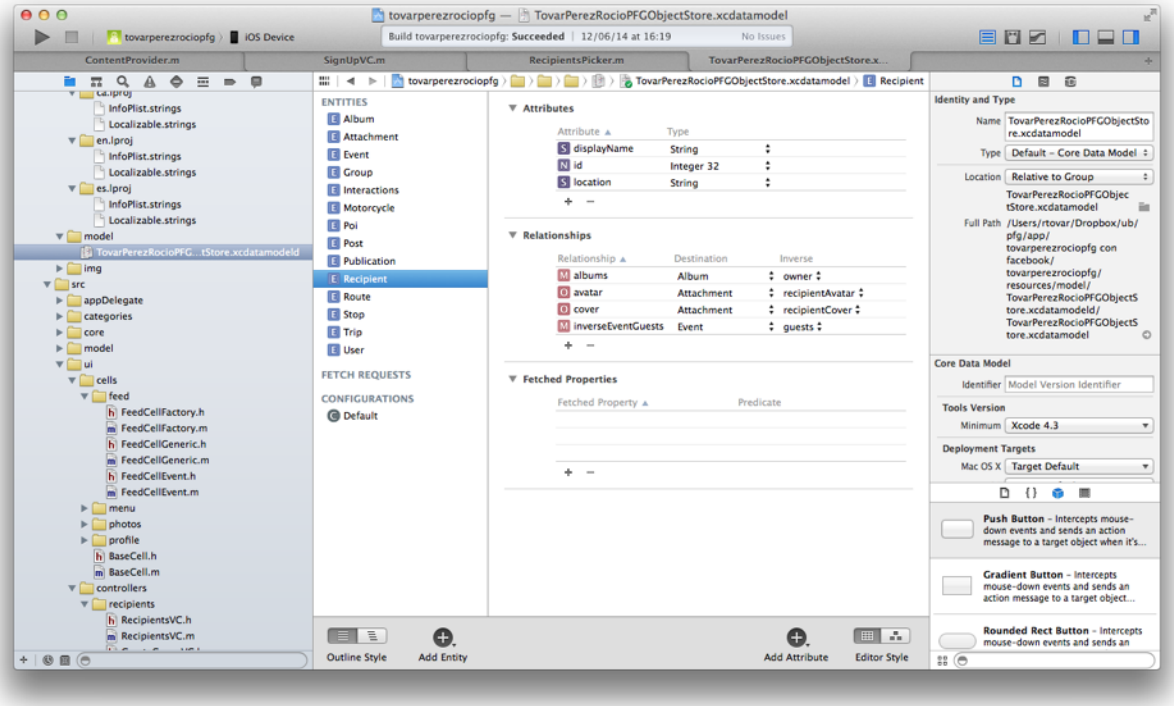

Figura 6.3. Captura de pantalla del modelo de datos en Core Data

Xcode (ver sección de antecedentes) tiene un editor que nos permite crear nuestro modelo de datos de una manera muy sencilla. En la figura 6.3. tenemos una captura de pantalla de la herramienta que nos ofrece el IDE de Apple para la gestión de nuestro modelo de datos.

Otra solución que nos ofrece Xcode es crear todas las clases resultantes de nuestro modelo de datos a partir del archivo .xcdatamodeld que genera la Managed Object Model.

En segundo lugar tenemos los **Managed Objects** que son subclases de la clase *NSManagedObject*. Estos objetos representan un registro de la base de datos,

llamados entidades en Core Data. En algunos aspectos un *NSManagedObject* actúa igual que un diccionario ya que es un contenedor genérico que proporciona almacenamiento para sus atributos definidos. Estos objetos soportan un rango de tipos comunes limitado que son strings, números (int, double, float), fechas y booleanos. Se pueden además almacenar objetos de tipo *NSData* si ninguno de los tipos proporcionados nos sirve para nuestra implementación.

Es conveniente explicar que con este framework las relaciones entre los distintos objetos de nuestro modelo de datos no se realizan mediante tablas que relacionan los ids de una tabla con los ids de otra sinó que existen las llamadas Relationships (relaciones en castellano) que nos facilitan esta tarea. Por ejemplo, supongamos que tenemos un objeto *User* que tiene como atributos un nombre, un correo electrónico y una contraseña. Aparte tenemos un objeto *Attachment* que tiene como atributos un nombre y una url. Bien, ahora queremos que el objeto *User* tenga un campo avatar que sea de tipo *Attachment*, pues lo que tenemos que hacer es crear una relación directa de *User* a *Attachment* y nombrarla "avatar". Además tenemos que crear la relación inversa de *Attachment* a *User* con el nombre *inversaUsuarioAvatar* por ejemplo. De esta manera nos evitamos tener que darle un campo id a cada uno de nuestros objetos si luego no va a ser necesario para la lógica de nuestra aplicación.

El siguiente paso en el ciclo de Core Data es trabajar con los Managed Objects de la aplicación, para eso este framework nos proporciona el **Managed Object Context** que es algo así como una memoria temporal donde podemos cambiar los valores de nuestros objetos sin escribir nada en disco. Así pues, siempre que modifiquemos los valores de los objetos que hayamos creados, estos cambios se llevarán a cabo en el contexto.

Para poder persistir las modificaciones realizadas en el contexto existe la **Persistent Object Store**. Este objeto es el que contiene la base de datos SQLite donde los datos se escriben a disco. Cada vez que queramos persistir los datos creados en el managed object context tenemos que guardar el contexto en la persisten object store, si no lo hacemos los cambios que hayamos hecho se perderán una vez cerremos la aplicación y no estarán disponibles cuando la volvamos a abrir.

6.3. Interfaz de usuario

Se hablará ahora de la interfaz de usuario creada para esta aplicación y de cómo cumple con las 10 heurísticas de usabilidad de Jakob Nielsen.

H1: Visibilidad del estado del sistema

Para cumplir con esta heurística que nos dice que el sistema debe mantener siempre informado al usuario de lo que está ocurriendo hemos utilizado el activity indicator de los dispositivos iOS que se encuentra en la barra de estado del teléfono. De esta manera podemos informar al usuario de que el sistema está buscando o procesando información.

H2: Relación entre el sistema y el mundo real.

Esta heurística nos explica que la aplicación debe utilizar el lenguaje del usuario con expresiones y palabras que le resulten familiares. Además es aconsejable que la información aparezca en un orden lógico y natural. Bien, pues en relación a esta heurística podemos comentar el sistema de visualización de los invitados a un evento. Sabemos que en el mundo real, en las culturas occidentales, normalmente el color rojo detiene mientras que el verde valida, es por eso que para mostrar el número de invitados que atenderán a un evento se ha decidido usar el color verde, mientras que usamos el color rojo para mostrar a los miembros que no asistirán. Otro ejemplo del cumplimiento de esta heurística se puede encontrar en los iconos gráficos usados que intentan representar el mismo concepto del mundo real.

H3: Libertad y control por parte del usuario.

Hablamos ahora de los casos en que el usuario navega a un sitio de la aplicación por error, en este momento el sistema debe proporcionar una “salida de emergencia” para abandonar de una manera fácil e intuitiva el estado no deseado en el que se encuentra. Es por eso que se ha escogido el sistema del menú lateral disponible desde cualquier sitio de la aplicación con solo deslizar un dedo hacia la derecha. De esta manera da igual en qué punto del sistema se encuentre el usuario, siempre tendrá acceso al menú principal.

H4: Consistencia y estándares

Esta heurística nos dice que los usuarios no deberían tener que preguntarse si diferentes palabras, situaciones o acciones significan lo mismo. Además nos indica que se deben seguir las convenciones de la plataforma. Es por esto que se ha

decidido hacer una aplicación nativa con un diseño específico para el sistema operativo iOS. Otro sitio donde se ha tenido en cuenta esta heurística ha sido en la barra de navegación de la aplicación que siempre nos muestra en la parte izquierda un botón para salir de la pantalla actual, en el centro el título del sitio en el que estamos ahora y en la derecha un botón de opciones de creación de contenido (con el símbolo "+").

H5: Prevención de errores

Incluso mejor que buenos mensajes de error es un diseño cuidadoso aquél que evite que un error se produzca. Esto es lo que nos dice la heurística 5. Por ello en todas las pantallas de publicación el botón de "Publicar" se muestra deshabilitado hasta que el usuario ha rellenado todos los campos obligatorios del formulario.

H6: Reconocer en lugar de recordar

La heurística 6 nos dice que es necesario hacer visibles acciones y opciones para que el usuario no tenga que recordar información entre distintas secciones o partes de la aplicación. Para ello, en este proyecto se reutilizan varias vistas, como es el caso de la clase FeedView que se utiliza siempre que tenemos que mostrar un listado de publicaciones al usuario. Otro ejemplo de la aplicación de esta heurística está en los botones de publicar o crear un nuevo grupo. Se utiliza siempre el mismo botón con el símbolo "+" de manera que el usuario puede reconocer que ese botón es para crear contenido.

H7: Flexibilidad y eficiencia de uso

En este caso se nos recomienda acelerar la interacción para el usuario de tal manera que el sistema pueda servir tanto a los usuarios sin experiencia como a los más avanzados. Es por eso que se además de poner el conjunto de todos los tipos de publicaciones en la pantalla principal tenemos una sección para cada tipo de publicación proporcionando así un atajo al usuario. También podemos ver la aplicación de esta heurística en el diseño de la navegación para acceder al menú lateral. Cuando se ha ido navegando entre pantallas el usuario puede ir volviendo atrás hasta llegar a la primera pantalla donde tendrá el botón de la barra de navegación para acceder al menú, pero como atajo siempre puede deslizar el dedo hacia la derecha y acceder así al menú en cualquier punto de la navegación.

H8: Diseño estético y minimalista

Los diálogos no deben contener información que es irrelevante o raramente necesaria. Cada unidad adicional de información en un diálogo compite con las unidades relevantes de información y disminuye su visibilidad relativa. Esto es lo que nos dice la heurística 8. Para conseguir este aspecto del diseño en la pantalla del perfil de usuario por ejemplo, mostramos siempre el nombre y avatar del usuario seleccionado y por defecto su información básica. Si el usuario quiere acceder a más información, como por ejemplo las rutas o amigos del usuario que está viendo, tiene una serie de pestañas que le servirán para ampliar la información deseada.

H9: Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de los errores

Esta heurística nos señala que los mensajes de error deben ser expresados en un lenguaje sencillo (sin códigos), indicar con precisión el problema y sugerir una solución constructiva. Por ello en las pantallas de inicio de sesión y de registro se utiliza una vista especial de color rojo que aparece cuando el usuario ha cometido algún error rellenando en el formulario, de esta manera es fácil reconocer que se ha producido un error. Para ayudar a diagnosticar y recuparse del error al usuario utilizamos los mensajes apropiados para cada error así como la manera de solucionarlos.

H10: Ayuda y documentación

Finalmente la heurística 10 nos dice que aunque es mejor que la aplicación pueda ser usada sin ayuda, en ocasiones puede ser necesario proveer cierto tipo de ayuda. En este caso, la ayuda debe ser fácil de localizar, especificar los pasos necesarios y no ser muy extensa. Para ello la primera pantalla mostrada al usuario antes de registrarse o iniciar sesión tiene un pequeño y conciso tutorial de uso de la aplicación.

6.4. Localización

Una característica que no se ha mencionado hasta ahora de la aplicación es que se ha desarrollado de manera que los textos que aparecen en ella se adapten al idioma que el usuario haya seleccionado en su teléfono. De manera que siempre que la traducción al idioma del usuario esté disponible los textos de la aplicación se

mostrarán en este. Si el idioma no se encuentra disponible, los textos se mostrarán en el lenguaje por defecto que en este caso es el inglés.

Para conseguir esto se han tenido que configurar las propiedades del proyecto para indicar que existen varios idiomas disponibles. Además de esto se ha creado un archivo .strings diferente para cada idioma disponible donde tenemos una lista de cadenas con el formato “llave_de_la_cadena” = “cadena traducida”. Así si queremos localizar el texto “atrás” por ejemplo tenemos que añadir la entrada “back” = “Back” en el archivo de textos en inglés, la entrada “back” = “Enrere” en el archivo del idioma catalán y finalmente la entrada “back” = “Atrás” en el archivo .strings correspondiente al castellano.

Una vez tenemos configurada la aplicación y hemos creado los diferentes strings para cada idioma, podemos utilizar la función *NSLocalizedString* de Objective-C que dada una clave nos devuelve su traducción para el idioma actual.

6.5. Resultados

Una vez especificados todos los detalles de la implementación, pasaremos a ver los resultados obtenidos en el desarrollo de esta aplicación.

La primera vez que entramos a la aplicación se muestra la pantalla principal (ver figura 6.4 con un pequeño tutorial y las opciones de iniciar sesión y registrarse.

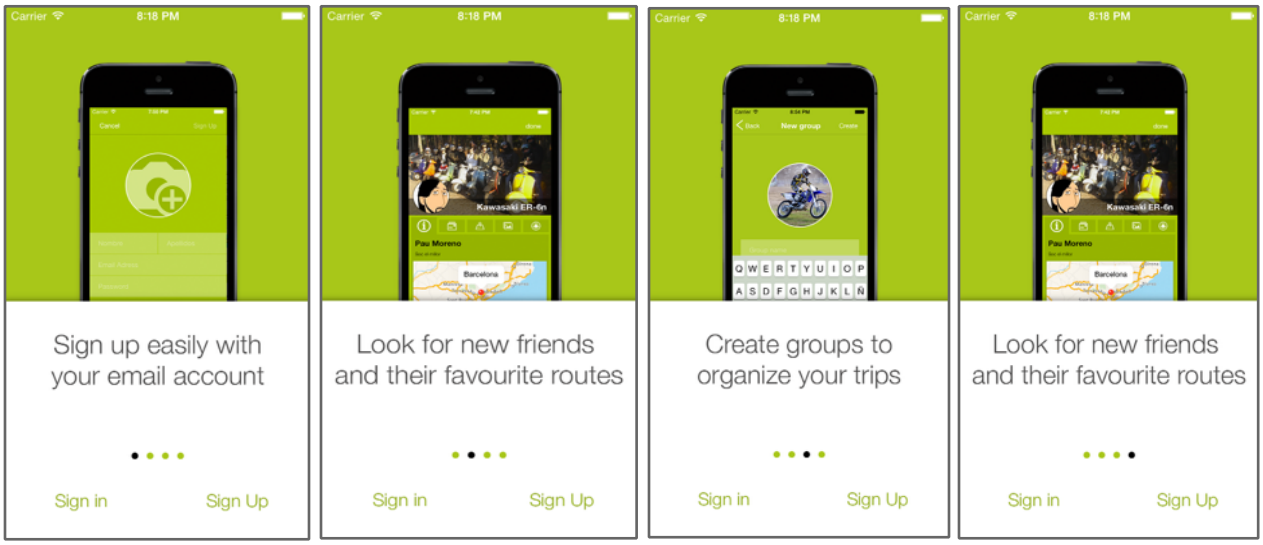

Figura 6.4. Pantalla inicial

Una vez hemos iniciado sesión en la aplicación nos aparece la pantalla principal con las últimas publicaciones y la barra de navegación desde donde podemos acceder al menú o crear una publicación nueva:

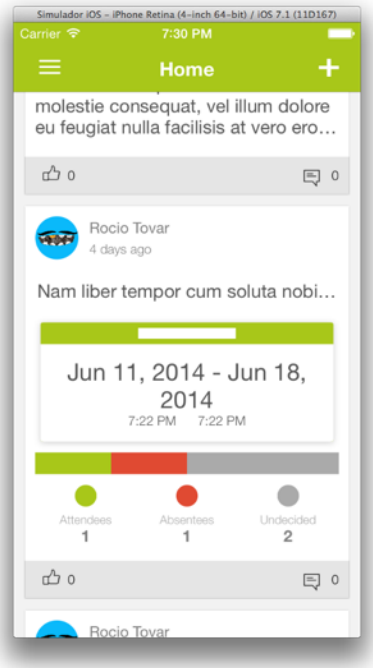

Figura 6.5. Pantalla principal

Cuando desplegamos el menú, ya sea apretando en el botón de la barra de navegación o utilizando el gesto de deslizar un dedo hacia la derecha podemos ver las diferentes secciones de la aplicación así como el avatar y nombre del usuario actual y un botón para cerrar la sesión:

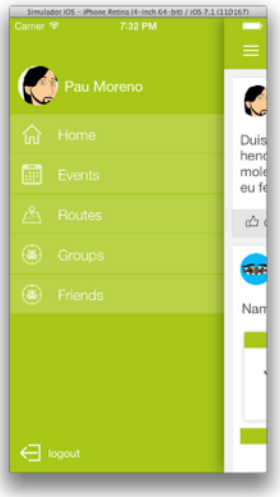

Figura 6.6. Menú lateral

Finalmente tenemos la pantalla de perfil de usuario a la que podemos acceder a través del menú lateral si queremos consultar nuestra información de perfil o bien desde la pantalla de amigos si lo que queremos es consultar la información de otra persona:

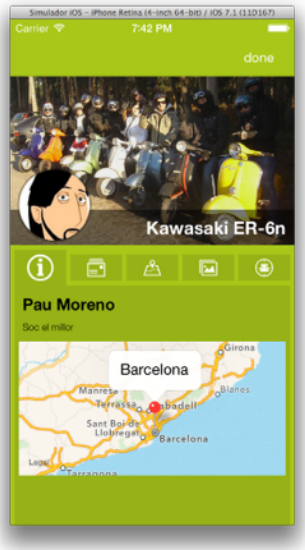

Figura 6.6. Perfil de usuario

7. Conclusiones y trabajo futuro

7.1. Conclusiones

Analizando el trabajo realizado en este proyecto y considerando los objetivos iniciales se pueden establecer las siguientes conclusiones:

El primer objetivo marcado era analizar el estado del arte previo al desarrollo de este proyecto. Se han conseguido identificar las dos herramientas más utilizadas por los motoristas hoy en día para organizar sus salidas además, después de observar el uso que se hacen de estas aplicaciones y las necesidades que tienen los motoristas se han detectado también las principales carencias de las herramientas existentes.

Después de este análisis se ha llegado a enumerar una serie de nuevas funcionalidades a incorporar en la aplicación diseñada para satisfacer las necesidades del usuario.

Finalmente, se ha diseñado una aplicación móvil con todas las funcionalidades deseadas teniendo en cuenta los principios de usabilidad estudiados en la carrera que hacen que un producto tenga calidad y sea intuitivo y fácil de usar. Además para implementar la aplicación se ha utilizado un diseño modular usando patrones de diseño que facilita la reutilización y modificación del mismo.

Por todos estos motivos pensamos que se han cumplido los objetivos establecidos al iniciar este proyecto. Aún así queda mucho trabajo por hacer ya que el límite de tiempo hace que no se hayan podido incorporar al proyecto nuevas ideas que han ido surgiendo a lo largo del desarrollo de la aplicación. Estas ideas se detallan a continuación.

7.2. Trabajo futuro

En primer lugar puesto que se ha apostado por crear una aplicación nativa para iOS es evidente que una parte del mercado queda fuera de nuestros objetivos, es por

esto que el primer paso a dar para ampliar el trabajo hecho aquí es utilizar las ideas desarrolladas en este proyecto para crear una aplicación nativa para Android.

Una vez se haya conseguido llegar a la mayoría de los dispositivos existentes en el mercado habría que plantear una mejora de la aplicación haciendo posible que el usuario pueda crear una ruta dentro de un mapa incorporado en la misma aplicación. Este sería un buen proyecto para empezar, ya que supondría mucho trabajo por parte del equipo de desarrolladores.

Otro punto interesante sería poder añadir dentro de la aplicación una sección de puntos de interés o puntos de parada que puedan ser agregados a los mapas de las rutas de un evento concreto, para que todos los asistentes a ese evento tengan claras las paradas del viaje.

Además, para motivar al usuario el uso de la aplicación y favorecer que participe de forma activa en la red social, sería una buena opción añadir algunos elementos de gamificación como puntos, scoreboards o badges.

Finalmente, como siempre en el ciclo del desarrollo de software es muy importante que escuchar la opinión que tienen los usuarios de nuestra aplicación para incorporar sus ideas en el desarrollo de una nueva versión de la aplicación lo que conllevaría a un análisis de la usabilidad de la aplicación.

8. Referencias bibliográficas

[1] **Mac Developer Library.** Repositorio con documentación sobre la programación para dispositivos iOS.
[2] **Ray Wenderlich.** Tutoriales de programación para dispositivos iOS.
[3] **iPhone Development 101.** Tutoriales de programación para dispositivos iOS.
[4] **Miguel Díaz Rubio.** Blog con tutoriales y consejos para programar aplicaciones para dispositivos iOS.
[5] **Facebook Developers.** Documentación sobre el SDK de Facebook.
[6] **EGOImageLoading.** Repositorio en Github con la librería EGOImage e instrucciones para su uso.
[7] **CHSlideController.** Repositorio en Github con la librería CHSlideController y las instrucciones para su uso.
[8] **pttrns**. Galería con ejemplos de diseños de aplicaciones iOS para diferentes elementos.
[9] **Cocoa Controls.** Repositorio con componentes de código abierto para dispositivos iOS.
[10] **Nielsen Norman Group.** Artículo sobre las heurísticas de Nielsen.
[11] **Teehanlax**. Plantilla de Photoshop de elementos de la interfaz gráfica de iOS 7.

Apéndice A. Manual técnico

Esta sección de la memoria está diseñada para ayudar a futuros desarrolladores o estudiantes que quieran llevar a cabo alguna de las mejoras propuestas en las conclusiones del punto 7.

El primer requisito será tener un ordenador con sistema operativo Mac OS con el IDE Xcode instalado. A continuación explicaremos como instalar el IDE necesario. Primero necesitamos descargarlo desde la App Store de nuestro mac (figura A.1) una vez descargado el proceso de instalación es muy sencillo y rápido.

Figura A.1. Descarga del IDE Xcode

Una vez descargado el IDE necesitaremos instalar al menos un simulador de dispositivo iOS para realizar pruebas durante el desarrollo de la aplicación. Para ello tenemos que dirigirnos al menú de la aplicación y abrir la opción *Xcode → Preferences → Downloads*. Una vez allí seleccionamos el simulador que necesitamos para descargarlo e instalarlo:

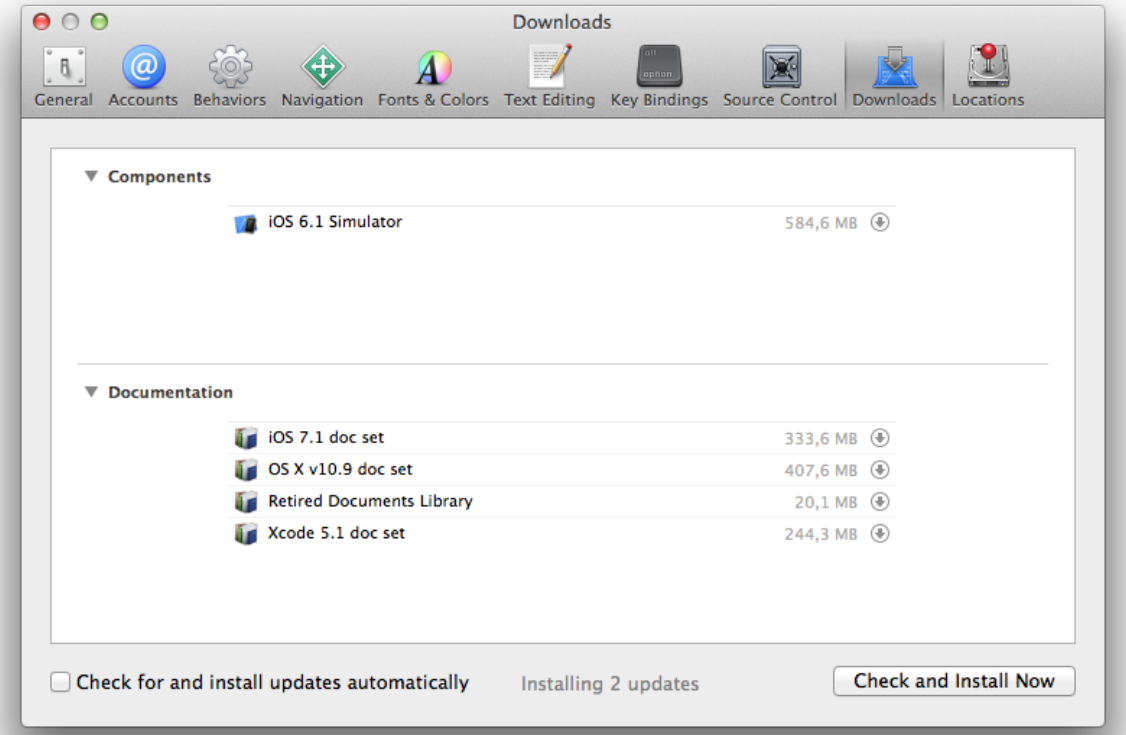

Figura A.2. Instalación de simuladores

A partir de aquí, lo único que tenemos que hacer es abrir el archivo .xcodeproj de nuestro proyecto con esto se nos mostrará una ventana del Xcode donde podremos navegar por el código fuente de la aplicación.

Una vez hecho las modificaciones pertinentes si queremos testear nuestros cambios en el simulador solo tenemos que seleccionar el dispositivo en que queremos llevar a cabo nuestra simulación de la lista disponible:

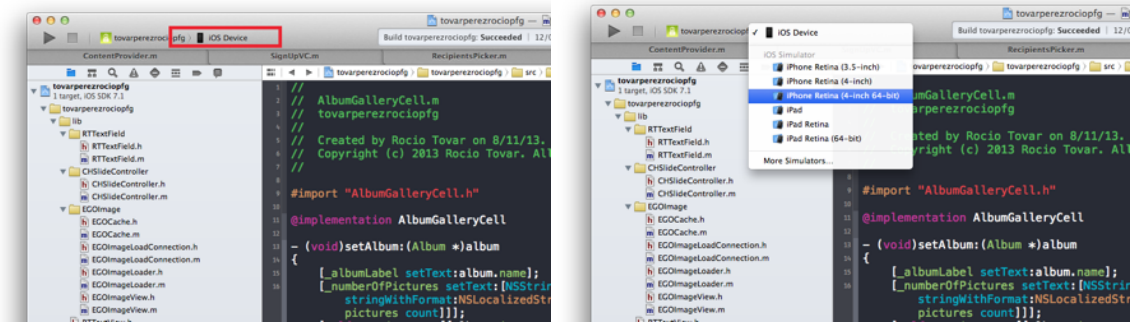

Figura A.3. Ejecutar la aplicación en un simulador

Explicaremos ahora la organización y estructura de los archivos que forman el proyecto. A continuación se presenta un listado detallado con las diferentes carpetas y la descripción de su contenido.

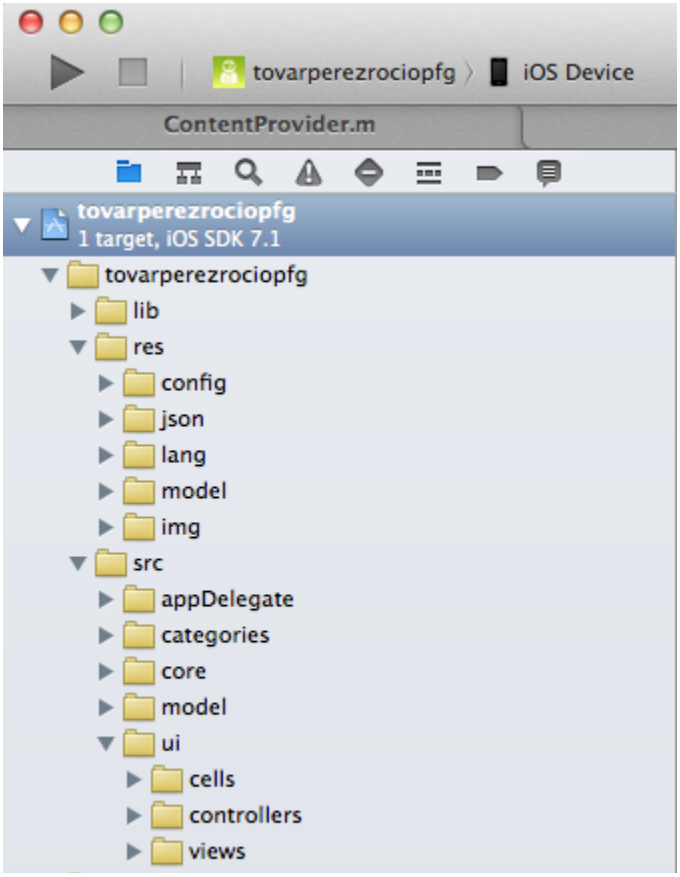

Figura A.4. Estructura del proyecto

- **lib**: librerías añadidas directamente al proyecto. Cada librería está incluida en una subcarpeta con su nombre.
- **res**: todos los archivos que no sean pertenezcan al código fuente de la aplicación.
    - **config**: archivos de configuración como .plist.
    - **json**: archivos con datos .json.
    - **lang**: archivos relacionados con la localización de la aplicación (en.strings, es.strings, ca.strings, …).
    - **model**: modelo de datos de Core Data (archivo .xcdatamodeld.
    - **img**: imágenes de la aplicación separadas en dos carpetas, una carpeta llamada normal con las imágenes en la resolución estándar y otra llamada @2x con las imágenes a doble resolución para dispositivos con pantalla retina.
- **src**: carpeta contenedora de los archivos con el código de la aplicación (.h, .m y .xib).
    - **appDelegate**: archivo con la clase desde la que se lanza la aplicación.
    - **categories**: contiene las categorías creadas para la aplicación, cada categoría está en una subcarpeta que lleva el nombre de la clase original de la categoría, por ejemplo si hemos creado una clase *NSString+Utils*, sus archivos .h y .m deberán estar dentro de la subcarpeta llamada *NSString*.
    - **core**: contiene las clases encargadas de la lógica de datos de la aplicación, como por ejemplo la clase *ContentProvider*.
    - **model**: clases con los managed objects creados a partir del modelo de datos de la aplicación.
    - **ui**: contenedor de las vistas y los *ViewControllers* de la aplicación.
        - **cells**: subclases de celdas específicas.
        - **controllers**: *ViewControllers* (agrupar los archivos .h, .m y .xib de cada *ViewController*).
        - **views**: subclases de vistas específicas.

Apéndice B. Manual de usuario de la aplicación

Esta sección de la memoria explica, sin entrar en detalles de la implementación el manual de usuario para poder ejecutar la aplicación y aprovechar todos sus contenidos. Se supone que el usuario ya tiene instalada la aplicación.

La primera pantalla que se nos muestra al entrar en la aplicación es la pantalla de inicio donde podemos navegar sobre un pequeño tutorial (figura 6.4).

El siguiente paso para empezar a usar la aplicación es registrarse, para eso solo tenemos que apretar el botón de “Registrar” y rellenar el formulario.

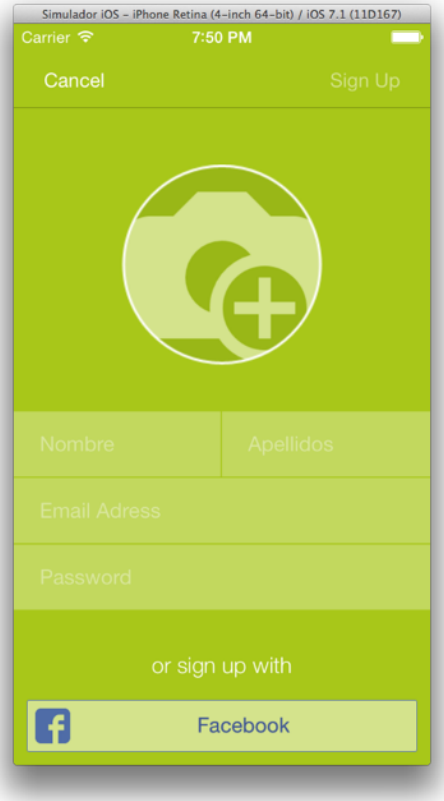

Figura B.1. Registro de un nuevo usuario

Se nos mostrará ahora la pantalla de inicio de la aplicación donde podemos ver la lista de las publicaciones más recientes y publicar nuevo contenido. Por ejemplo, para publicar un evento por ejemplo, apretamos en el botón de con el símbolo “+” situado a la derecha en la barra de navegación y seleccionamos la opción “Evento”. Nos aparecerá la siguiente pantalla:

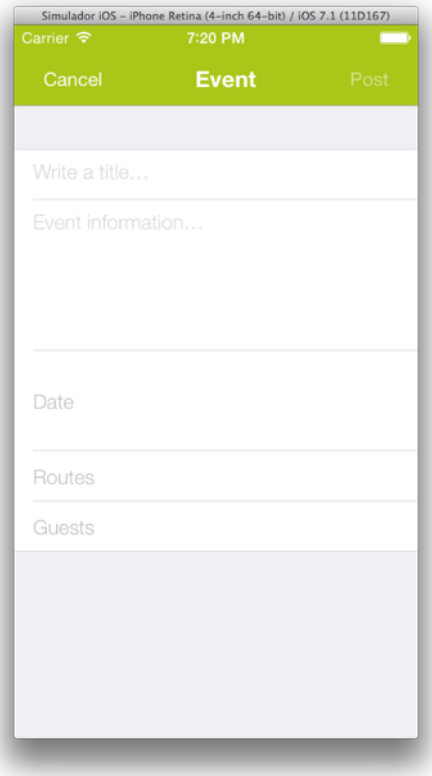

Figura B.2. Creación de un evento

Solo tenemos que rellenar los campos requeridos y escoger la fecha, rutas e invitados deseados, una vez listos pulsamos el botón de publicar que tenemos a la derecha del título en la barra de navegación.

Para crear un grupo nuevo debemos seguir el mismo proceso pero esta vez desde la pantalla de grupos a la que habremos accedido desde el menú lateral (figura B4).

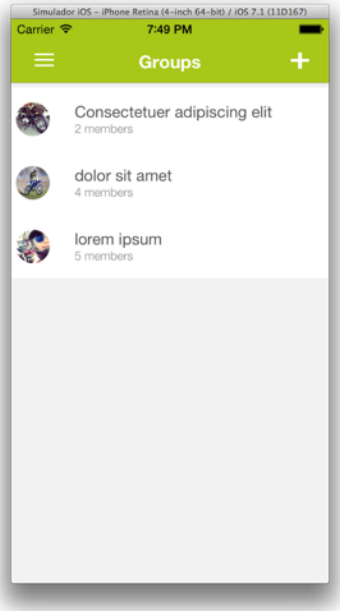

Figura B.3. Vista de grupos

Finalmente para poder ver nuestro perfil de usuario, desplegamos el menú lateral y clicamos en nuestro avatar esto nos llevará a nuestro propio perfil donde podremos navegar entre nuestra información principal, nuestras publicaciones, nuestras rutas, nuestros albums y nuestros amigos gracias a las pestañas de la parte inferior de la pantalla (figura B.5).

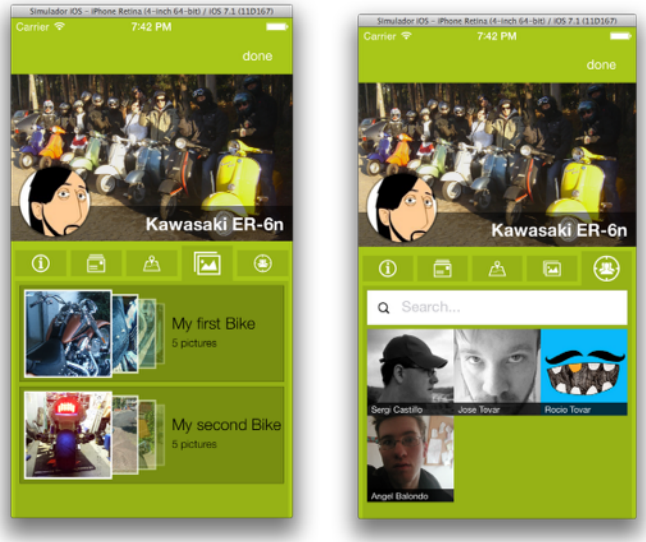

Figura B.4. Perfil de usuario