Power Comic Reader
Lector de comics para iPad

Sebastián Suelves Sellés
Ingeniería Técnica en Informática de Sistemas

Consultor: Roberto Ramírez Vique

11 de Marzo de 2013 (PEC3-Memoria Preliminar)

INDICE

1. Introducción

1.1. Objetivo del proyecto

El presente Trabajo de Fin de Carrera tiene como objetivo el desarrollo de una aplicación iOS para dispositivos iPad (versiones mini y estándar) para la lectura de comics en formato .cbr y .cbz. Aplicando y ampliando los conocimientos adquiridos a lo largo de la carrera sobre un proyecto real enfocado a dispositivos móviles.

Existen tres tipos principales de comics en formato digital:

1) Formatos .cbr y .cbz, son los formatos mas extendidos de comics digitalizados o escaneados. No suelen tener soporte comercial por parte de las compañías editoras de comics, suelen provenir de usuarios los cuales se dedican a escanear hoja a hoja los diversos comics y ponerlos accesibles en la red internet. Básicamente consisten en una secuencia numerada de imágenes (una pagina por imagen y en algún caso dos si el modo de escaneado es apaisado), siendo las diferencias entre dichos formatos el modo de compresión usado. El formato .cbr corresponde a un fichero .rar renombrado como .cbr, mientras que el formato .cbz corresponde a un fichero .zip renombrado como .cbz.
2) Formato PDF, es prácticamente igual al anterior a excepción que el contenedor de la secuencia de imágenes se trata de un fichero PDF.
3) Formatos propietarios, en los últimos tiempos las editoras de comics han comprendido la importancia de mercado de comics en formato digital por lo que estas han lanzado aplicaciones con formatos propietarios validos únicamente esta y los comics de cada una de las editoriales, estos comics son adquiridos en sus portales específicos y visualizables únicamente por los lectores de comics específicos de la editora o consorcio de editoras, siendo estos visualizadores generalmente gratuitos y los comics de pago.

Nuestro software se centrara en los formatos .cbr y .cbz por ser de dominio publico, no descartándose como mejora en un futuro la ampliación al soporte .pdf.

Con el desarrollo del presente Trabajo Final de Carrera se persigue:

- Poner en practica los conocimientos adquiridos a lo largo de la carrera.
- Adquirir un nivel de conocimiento para el desarrollo de soluciones móviles en entorno iOS, tanto a nivel de aprendizaje del lenguaje Objective-C como de frameworks para la plataforma iOS.
- Dominar el desarrollo en entornos MAC, especialmente de XCODE así como diversos complementos como el control de versiones mediante GIT.
- Aprender una tecnología actual y con amplias posibilidades de desarrollo a nivel laboral.

1.2. Funcionalidades

El software tendrá como funcionalidad básica:

- Importación de comics a través de iTunes.
- Interfaz similar a la aplicación iBook (simulando un estante de libros) que muestre las portadas de los distintos comics importados.
- Tratamiento de la biblioteca de comics. Permitiendo la reorganización mediante el arrastre (drag and drop) de los libros y su eliminación (borrado) de los comics existentes en la biblioteca.

- Seguimiento del estado de la lectura de cada comic, de forma que la reanudación de un comic nos posicione en la portada de este (si es la primera vez que lo abrimos o si la ultima vez que se abrió se llego al final de este).
- Búsqueda y uso de frameworks para la descompresión de los formatos de compresión zip y rar (por ejemplo zlib).
- Diversos métodos de paso de pagina (soporte de varios gestos), mediante pulsación en los laterales de la pagina o deslizamiento de un dedo en sentido izquierda-derecha (pagina anterior) o derecha-izquierda (pagina siguiente).
- Zoom que facilite la lectura de los textos de los comics mediante gestos. (pellizco y doble pulsación sobre la pantalla). La doble pulsación sobre la pantalla conmutara entre el modo zoom y el tamaño a pantalla completa, siendo el factor de ampliación fijado a través de la aplicación del sistema **Ajustes** por el usuario en función de sus preferencias.
- Posibilidad de ordenar los comics/colecciones según las preferencias del usuario.
- Soporte para rotación del iPad.
- Opciones de ajuste (como la activación del zoom con uno o dos toques sobre la pantalla, así como alguna opción para personalizar el interfaz.
- Interfaz multi-idioma (Ingles y Español)

1.3. Futuras Mejoras

Las futuras mejoras del producto son:

- Soporte de comics en formato PDF.
- Zoom inteligente de texto mediante técnicas de Visión por Computador mediante el uso del framework OpenCV.
- Soporte de Dropbox, permitiendo la visualización de libros fuera de nuestro dispositivo (en la nube).
- Modo dos paginas en posición apaisada.
- Detección de viñetas, es decir, mediante técnicas de Visión por Computador detectar las distintas viñetas del comic permitiendo su visualización por separado y por tanto su cómoda visualización en dispositivos con pantalla pequeña permitiendo dar soporte a dispositivos tipo iPhone/iPod al descomponer cada pagina en sus sucesivas viñetas.
- Interfaz de la aplicación más personalizable con rediseño de botones, barras de herramientas, etc.
- Importación de ficheros .cbr y .cbz cuya extensión sea errónea.

1.4. Tecnología escogida

El proyecto se desarrollara para la plataforma de Apple iOS en su versión 6 y para ello se empleara un equipo iMac y XCODE como entorno de desarrollo, enfocándose el interfaz a dispositivos iPad dada la naturaleza de la aplicación dándose soporte a las resolución estándar del iPad 1, 2, y mini, así como a las resoluciones retina de los iPad 3, y 4. Haciéndose un especial hincapié en el desarrollo del patrón MVC (Modelo-Vista-Controlador) por adecuarse especialmente al proyecto. Para la depuración de la aplicación se dispondrá además del simulador de XCODE de un dispositivo iPad de tercera generación.

1.5. Estudio de Mercado

Se ha realizado un estudio de mercado mediante la búsqueda de análisis en diversas paginas web especializadas en entorno MAC buscando información sobre los programas para iPad recomendados para la lectura de comics.

En la categoría que nos ocupa (.cbr y .cbz) según las diversos análisis de paginas como www.applesfera.com, www.wired.com, y www.ipadizate.es, el ganador actual es ***Comic Zeal 5*** (4,49€), el cual destaca básicamente por su interfaz muy cuidado y personalizable así como una gestión de los comics sencilla que facilita su ordenación y clasificación por categorías de forma innovadora (en lugar de arrastrar se hacen deslizamientos hacia la derecha en los comics a mover pasando estos al portapapeles y desde aquí con un simple pegar se pegan todos en un solo paso), sin embargo según opiniones de varios usuarios y análisis, este tipo de gestión si bien resulta cómodo requiere una adaptación (pues se aleja bastante de lo que un usuario de iOS esta habituado) por lo que esta ventaja se convierte en ocasiones en una desventaja según el usuario, además permite la elección de diversas texturas y tonos por parte del usuario, el paso de paginas es muy rápido. Otras aplicaciones mas baratas no suelen tener un interfaz tan elaborado y a veces el paso de paginas es lento como serian las aplicaciones gratuitas ***iComix***, o ***Comic Viewer***. Otras han optado por un modelo ***in-app-purchase*** (muy criticado por los usuarios) como es el caso de ***ComicFlow*** que en principio era gratuito y ahora solo permite subir 50 comics vía web, si se desea más hay que desbloquear el modo de subida ilimitado previo pago de 3,59€ en ***in-app-purchase***. No obstante pese a parecer que el mercado pueda estar cubierto existe un vacío provocado por la desaparición de la App Store del lector ***Comic Reader Mobi***, el cual tuvo bastante éxito y buenas criticas cuando salió al mercado pero fue baneado de la *App Store* por intentar la subida de comics por USB sin necesidad de iTunes, este intento provoco el rechazo de la aplicación pero el desarrollador oculto esta característica para ofrecerla a espaldas de Apple y cuando fue descubierto por *Apple* fue baneado de la *App Store* (siendo únicamente posible su instalación en dispositivos jailbrekeados), esta aplicación tenia en realidad un único punto fuerte y es que a diferencia de las demás permitía hacer zoom únicamente a los textos (supuestamente mediante técnicas de visión por computador), siendo esta una prestación que creo factible de implementar en el futuro mediante el framework *OpenCV*. Actualmente el precio de esta aplicación comprada a través de la pagina del desarrollador tiene un precio en iPhone/iPad de 14,99$. Por lo que considero que un precio moderado entorno a los 2,5€-3,5€ seria ideal en vista del mercado actual.

1.6. Riesgos del proyecto

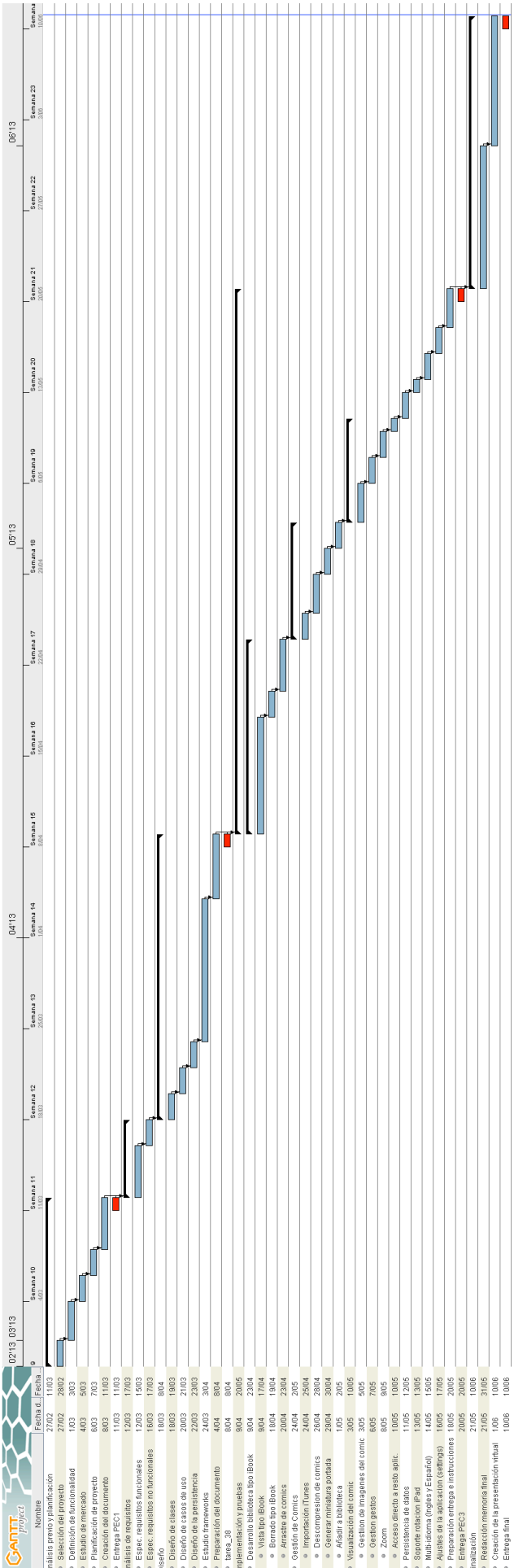
A nivel funcional el mayor problema radica en conseguir desarrollar un interfaz tipo iBook que no solo visualice la biblioteca sino que permita su gestión de una forma natural para el usuario (arrastre de comics para ordenarlos, mantenerlos pulsados para entrar en el modo de borrado, etc). Para minimizar estos riesgos se recurrirá en caso necesario a frameworks de terceros que permitan una solución a los problemas planteados.

Sin embargo desde un punto de vista comercial el mayor problema teórico seria que la propia aplicación de *Apple* ***iBook*** incorporase como mejora en el futuro el soporte de comics en estos formatos. Es por ello por lo que nos hemos centrado en los formatos .cbr y .cbz ya que estos al no disponer de DRM, no son probables que sean implementados por iBook en el futuro, pues no podrían incorporarse a su tienda y por tanto a su modelo de negocio. Así mismo podría suponer un problema la vuelta a la *App Store* de la aplicación ***Comic Reader Mobi***, aunque no es probable

(el baneo lleva activo desde Abril de 2010) y debido a su precio desproporcionado en principio no debería resultar un problema sino simplemente algo de competencia. Para minimizar el riesgo y teniendo en cuenta los precios de los productos de la competencia, se estima que un precio entre 1,79€ y 2,50€ minimizaría los riesgos, quedando aun algo de margen de bajada en caso de necesidad.

1.7. Calendario del proyecto

Análisis previo y planificación	13 días	27/02/2013	11/03/2013
Selección del proyecto	2 días	27/02/2013	28/02/2013
Definición de funcionalidad	3 días	01/03/2013	03/03/2013
Estudio de mercado	2 días	04/03/2013	05/03/2013
Planificación del proyecto	2 días	06/03/2013	07/03/2013
Creación del documento	4 días	08/03/2013	11/03/2013
Entrega PEC1 (11/03/2013)	Hito	11/03/2013	
Análisis de Requisitos	6 días	12/03/2013	17/03/2013
Especificación de requisitos funcionales	4 días	12/03/2013	15/03/2013
Especificación de requisitos no funcionales	2 días	16/03/2013	17/03/2013
Diseño	22 días	18/03/2013	08/04/2013
Diseño de clases	2 días	18/03/2013	19/03/2013
Diseño de caso de uso	2 días	20/03/2013	21/03/2013
Diseño de la persistencia	2 días	22/03/2013	23/03/2013
Estudio de frameworks disponibles para: Manejo de imágenes, persistencia, descompresión de ficheros (.zip y .rar), y demás necesidades tecnológicas.	11 días	24/03/2013	03/04/2013
Preparación del documento	5 días	04/04/2013	08/04/2013
Entrega PEC2	Hito	08/04/2013	
Implementación y pruebas	42 días	09/04/2013	20/05/2013
Desarrollo de la biblioteca tipo iBooks (visualización biblioteca, borrado de comics, arrastre de estos para su organización en la biblioteca)	15 días	09/04/2013	23/04/2013
Gestión de Comics (importación mediante iTunes, descompresión, añadir a biblioteca, generar miniatura portada comic)	9 días	24/04/2013	02/05/2013
Visualización del comic seleccionado (gestión de gestos avance y retroceso pagina, zoom, acceso a settings, biblioteca, etc)	8 días	03/05/2013	10/05/2013
Persistencia	2 días	11/05/2013	12/05/2013
Soporte de rotación	1 día	13/05/2013	13/05/2013
Multi-idioma (Ingles y Español)	2 días	14/05/2013	15/05/2013
Ajustes de la aplicación (Settings)	2 días	17/05/2013	17/05/2013
Preparación entrega e instrucciones	3 días	18/05/2013	20/05/2013
Entrega PEC3 (20/05/2013)	Hito	20/05/2013	
Finalización	21 días	21/05/2013	10/06/2013
Redacción memoria final	11 días	21/05/2013	31/06/2013
Creación de la presentación virtual	10 días	01/06/2013	10/06/2013
Entrega final (10/06/2013)	Hito	10/06/2013	


GANTT project
02/13
03/13
04/13
05/13
06/13
Nombre
Fecha d.
Fecha
Análisis previo y planificación 27/02 11/03
Selección del proyecto 27/02 28/02
Definición de funcionalidad 1/03 3/03
Estudio de mercado 4/03 5/03
Planificación de proyecto 6/03 7/03
Creación del documento 8/03 11/03
Entrega PEC1 11/03 11/03
Análisis de requisitos 12/03 17/03
Esec. requisitos funcionales 12/03 15/03
Esec. requisitos no funcionales 16/03 17/03
Diseño 18/03 8/04
Diseño de clases 18/03 19/03
Diseño de casos de uso 20/03 21/03
Diseño de la persistencia 22/03 23/03
Estudio frameworks 24/03 3/04
Preparación del documento 4/04 8/04
tarea_38 8/04 8/04
Implementación y pruebas 9/04 20/05
Desarrollo biblioteca tipo iBook 9/04 23/04
Vista tipo iBook 9/04 17/04
Borrado tipo iBook 18/04 19/04
Arrastre de comics 20/04 23/04
Gestión de Comics 24/04 2/05
Importación iTunes 24/04 25/04
Descompresion de comics 26/04 28/04
Generar miniatura portada 29/04 30/04
Añadir a biblioteca 1/05 2/05
Visualización del comic 3/05 10/05
Gestion de imagenes del comic 3/05 5/05
Gestion gestos 6/05 7/05
Zoom 8/05 9/05
Acceso directo a resto aplic. 10/05 10/05
Persistencia de datos 11/05 12/05
Soporte rotacion iPad 13/05 13/05
Multi-idioma (Ingles Y Español) 14/05 15/05
Ajustes de la aplicación (settings) 16/05 17/05
Preparación entrega e instrucciones 18/05 20/05
Entrega PEC3 20/05 20/05
Finalización 21/05 10/06
Redacción memoria final 21/05 31/05
Creacion de la presentación virtual 1/06 10/06
Entrega final 10/06 10/06

2. Estudio de requisitos

2.1 Requisitos Funcionales

El principal objetivo es la obtención de una aplicación para iPAD de lectura de comics, así pues dividiremos el sistema en tres bloques principales:

- Funcionalidades de la biblioteca de comics, es decir, la gestión de los comics importados en la aplicación como son el borrado, organización en colecciones, e importación de los mismos.
- Funcionalidad de visualización de los comics. Como son la apertura de un comic, el paso de pagina, zoom, movimiento por la pagina en modo zoom (panning), etc.
- Funcionalidad complementaria. Esta es aquella destinada a complementar la aplicación para que alcance sus objetivos y el mínimo de calidad exigido como son permitir la personalización de la aplicación del sistema **Ajustes** permitiéndole cambiar entre distintos estilos y habilitar o no los distintos tipos de zoom, e integración con iTunes para transferencia de ficheros de comics.

2.1.1 Funcionalidad de la biblioteca de comics

Las siguientes funcionalidades se encuentran dirigidas a gestionar los aspectos relacionados con la biblioteca de comics.

Importar comics
Permitirá al sistema detectar nuevos comics transferidos vía iTunes e incorporarlos a la biblioteca.

Añadir Colección
Permite crear una nueva colección en la que agrupar un conjunto de comics.

Eliminar Colección
Elimina una colección y los comics que contenga dicha colección de la biblioteca.

Seleccionar Colección
Seleccionando una colección dada se muestran únicamente los comics asignados a la colección seleccionada

Edición
Permite cambiar el modo en el que funciona la biblioteca, permitiendo modificarla trasladando o eliminando comics.

Fin Edición
Permite volver al modo estándar de trabajo de la aplicación en la que la pulsación sobre un comic se interpreta como solicitud de apertura del mismo.

Trasladar comic a otra colección
Mueve uno o varios comics de una colección a otra permitiendo selección múltiple.

Eliminar comic
Elimina un o varios comics de la biblioteca permitiendo selección múltiple.

Abrir comic
Lanza el modulo de visualización de comics al pulsar sobre un comic (fuera del modo edición).

Reordenar comic dentro de Colección
Permite cambiar el orden de visualización de los comics que posee una colección. Para ello se mantendrá pulsado el comic a reordenar y se arrastrara a su nueva posición.

2.1.2 Funcionalidad de visualización de comics

Menú oculto para volver a biblioteca
Existirá un menú tipo barra de herramientas que permita volver a la biblioteca, la ocultación/aparición se llevara a cabo mediante la detección de una pulsación en la zona central de la pantalla.

Zoom
El modo principal de zoom será mediante dos dedos (pellizco o pitch en ingles) de modo que separando los dedos se ampliara y juntándolos se reducirá. No podrá reducirse por debajo de la pantalla completa (escala 1:1). Como opción se permitirá hacer zoom mediante dos pulsación en un factor de ampliación ajustable, y reducirlo de nuevo con un segundo doble clic al tamaño original. Sin embargo por defecto este tipo de zoom estará deshabilitado pues puede ser bastante común hacer una doble pulsación por error en el manejo de la aplicación. No se podrá cambiar de pagina si estamos en modo pagina ampliada.

Deslizamiento por pagina ampliada (panning)
Si hemos aplicado zoom a la pagina, podremos movernos por esta mediante el movimiento de nuestro dedo.

Cambio de pagina
Podremos cambiar de pagina siempre que no estemos en una pagina ampliada mediante dos tipos de gestos: un desplazamiento con un dedo o pulsando la pantalla en un lateral de la pantalla. Si el desplazamiento es de izquierda a derecha nos moveremos a la pagina anterior, mientras que si es de derecha a izquierda nos moveremos a la pagina siguiente. Por otro lado en el caso de una única pulsación si pulsamos cerca del margen izquierdo nos movernos a la pagina anterior y si pulsamos cerda del derecho nos moveremos a la pagina siguiente.

2.1.3 Funcionalidad complementaria

Personalización (Ajustes)
Permite a través de la herramienta del sistema iOS ***Ajustes***, personalizar diversos aspectos de la aplicación como son el tema (color y textura de fondo), el factor de zoom empleado por la doble pulsación de la doble pulsación, y si esta estará o no activada.

Integración con iTunes
Permitirá que puedan transferirse comics desde el ordenador al dispositivo iPAD empleado.

2.2 Requisitos No Funcionales

La aplicación además deberá:

- Funcionar correctamente en los dispositivos iPAD de 1ª, 2ª, 3ª, y 4ª generación, además del iPAD Mini, por lo que deberá soportar las resoluciones tipo Retina y no Retina.
- Soporte multi-idioma (Español/Ingles).
- Soporte de rotación.
- Estilo visual similar a iBook.

2.3 Usuario al que va dirigida la aplicación

La aplicación Power Comic Reader esta dirigida al usuario amante de los comics permitiéndole llevar consigo su colección de comics con facilidad, con importante ahorro de espacio y facilidad de búsqueda, ayudándonos a mantener organizada nuestra colección de comics a la vez que su disponibilidad, permitiéndole aprovechar tiempos muertos (viajes en autobús, metro, descansos) para la lectura de comics y optimizar en gran medida su tiempo de ocio pues se facilita la localización de un comic concreto al estar todos organizados en colecciones y no existir la posibilidad de traspapelarlo por no mantener organizada la colección como puede ocurrir con una colección de comics física en lugar de electrónica.

Como se prevén picos de uso importantes ya que la aplicación esta enfocada especialmente a su uso en tiempo de ocio con duraciones que pueden oscilar de 20 minutos a varias horas, deberán tomarse en cuenta en consideración aspectos que maximicen su usabilidad, rendimiento, y duración de la batería del dispositivo usado, ofreciéndose un análisis de estos puntos en el siguiente punto de esta memoria.

2.4 Decisiones de diseño que afectaran al uso de la aplicación

Como se ha comentado anteriormente la aplicación hace uso de los formatos de comics .cbr y .cbz, dichos formatos son ficheros comprimidos en formato rar o zip que actúan como contenedores de las imágenes que componen las distintas paginas de un comic escaneado. Aquí podemos diseñar la aplicación bajo dos enfoques: mantener los ficheros comprimidos en la biblioteca de nuestra aplicación, o descomprimir estos ficheros y prepararlos para su uso.

El primer enfoque (mantenerlos comprimidos) únicamente tiene ventajas a nivel de espacio (y no muy relevantes como veremos a continuación), requiriendo bien procesar fichero a fichero dentro del fichero comprimido o su descompresión completa en un fichero temporal. Este tipo de procesamiento aumenta drásticamente el consumo de la batería, e introduce importantes retardos en la aplicación bien en el paso de la pagina (si descomprimimos pagina a pagina según la requiramos), o bien en la apertura del comic (al tener que descomprimirlo por completo en el directorio temporal). La única opción de optimización de estos tiempos esta en la descompresión pagina a pagina realizando dicha descompresión mediante técnicas de

multitarea y uso de caches. Sin embargo este tipo de procesamiento precisa de un uso de CPU importante y memoria, llevando a un nivel mas problemático la duración de la batería y memoria.

El segundo enfoque consiste en descomprimir completamente los comics en la biblioteca como secuencia de imágenes incorporando cada comic en un directorio en la biblioteca de nuestra aplicación ya descomprimido y accediendo directamente a cada imagen/pagina mediante el sistema de ficheros. Este enfoque puede parecer que puede redundar en un gasto excesivo de capacidad en nuestro dispositivo, pero en realidad no es cierto, hay que recordar que los formatos usados contienen una secuencia de imágenes generalmente en formatos .jpg o .png que poseen un alto grado de compresión, por lo que lo que la compresión aportada por los formatos suele rondar el 2% o menos, por lo que la perdida de espacio no es significativa en nuestro dispositivo, mientras que la duración de la batería se maximiza (solo es preciso descomprimir el comic una vez al importarlo) así como el uso de memoria pues el tamaño del fichero de imagen de una pagina suele estar en torno a los 200-250 Kbytes, tamaño fácilmente manejable por el sistema de fichero del iPad debido a la alta velocidad del almacenamiento de este. Siendo este el enfoque elegido en la aplicación **Power Comic Reader**, como ventaja añadida de este enfoque tenemos el hecho de facilidad de uso y catalogación, pues al mismo tiempo que se importan los comics es posible obtener los datos de interés (como son el numero de paginas) y añadirlos como objetos persistentes minimizándose las tareas en el arranque de la aplicación.

Así pues **Power Comic Reader** importara los distintos ficheros comprimidos descomprimiéndolos en su totalidad y categorizándolos para obtener un alto rendimiento y un uso de memoria y CPU bajos permitiéndole por tanto su uso por parte del usuario durante horas de uso continuo.

3. Análisis

En esta sección se recoge el análisis del sistema, presentándose en primer lugar el Diseño de Clases para a continuación continuar con el Diseño de Casos de Uso y posteriormente el Diseño de la Persistencia, para ello partiremos de la información general aportada por los diagramas de casos de uso, detallándose cada uno de ellos.

3.1 Análisis de clases

El siguiente diagrama representa las clases identificadas en la aplicación a solucionar:

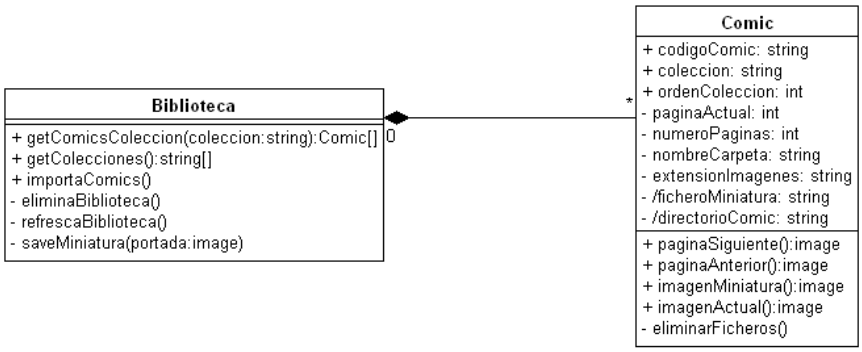


Figura 1. Modelo de dominio

Las entidades detectadas son:

- Biblioteca, que representa todo el conjunto de comics importados en la aplicación, un conjunto de comics, y provee los servicios básicos de la biblioteca como son el acceso a dichos comics, la obtención de la lista de colecciones de estos, e importa los nuevos comics extrayendo los ficheros .cbr y .cbz como una lista de imágenes en un directorio y generando la miniatura de la portada para su representación.
- Comic, la cual representa un comic en concreto y contiene la gestión de este (movimiento de paginas con devolución de la nueva pagina, devolución de imagen de miniatura, devolución de la imagen de la pagina actual). Posee dos atributos derivados que se forman a partir de la información obtenida mediante consulta al propio equipo (ruta al directorio ***Documents*** de la aplicación en iOS) en conjunción con el atributo ***nombreCarpeta*** y algunas reglas fijas (como que la miniatura tendrá nombre 0.png).

Un punto a considerar en el diseño es que se tratara de minimizar el uso de memoria de los objetos debido al enfoque de la aplicación a dispositivos móviles. Por ello puede observarse que la clase Comic no almacena en memoria las imágenes de cada una de las paginas que lo contienen ni la miniatura de la portada, sino que esta trabajara con el sistema de ficheros para la recuperación desde disco de la imagen deseada de modo que se minimice el almacenamiento en memoria de imágenes lo cual podría provocar problemas de memoria (especialmente en los dispositivos mas antiguos), es por ello que en las clases se trabaja principalmente con rutas a nivel de atributos y se implementan métodos que faciliten el acceso a las imágenes necesarias en cada momento.

3.2 Análisis de casos de uso

El siguiente diagrama recoge una vista global de los actores que describen los requisitos funcionales de la aplicación.

A continuación se incluye el diagrama y el detalle de cada uno de los casos de uso:

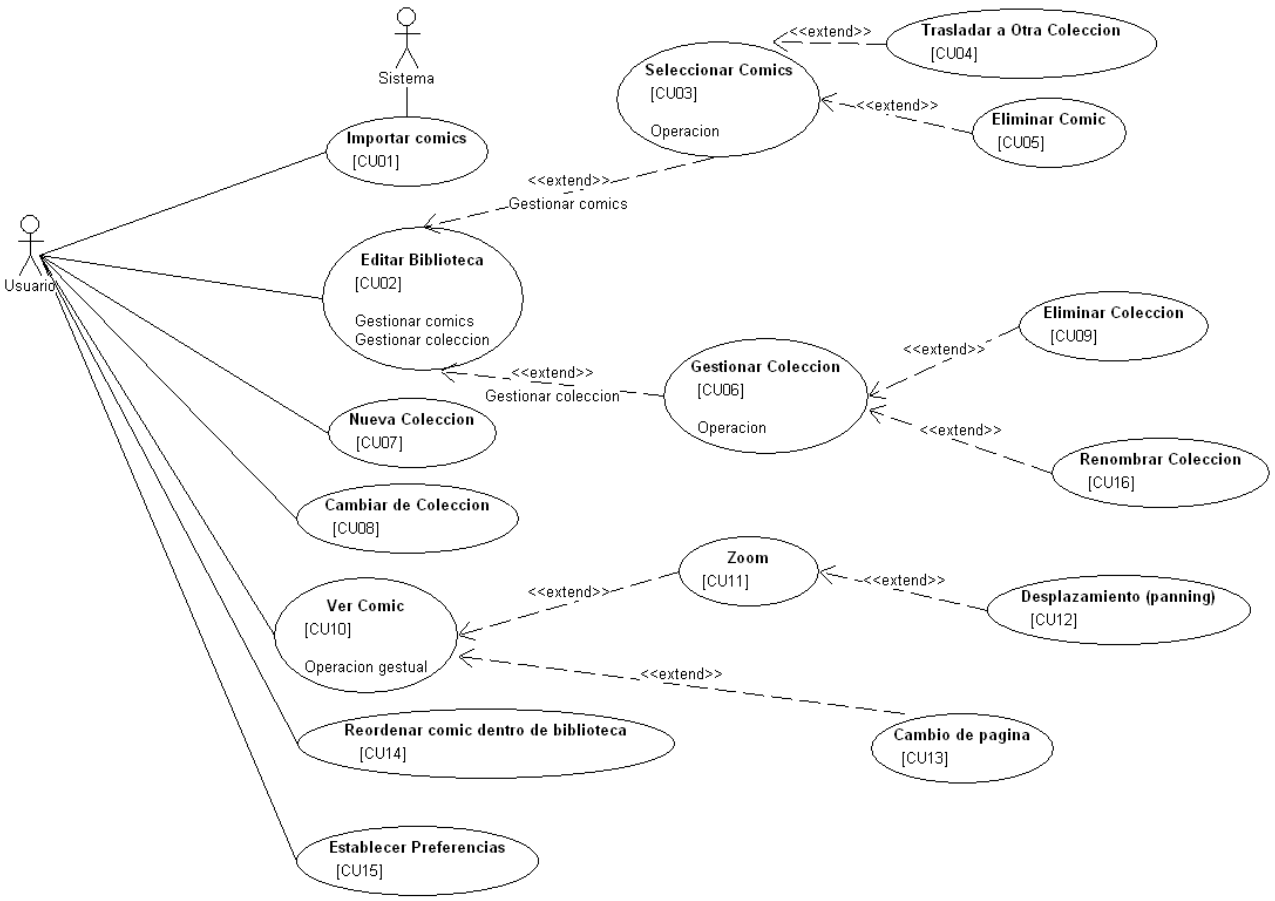


Figura 2. Casos de uso

Como se puede observar el sistema distinguirá dos tipos de actores diferentes:

- Usuario, es el actor general de la aplicación el cual puede hacer uso de toda la aplicación y se corresponde con el usuario físico de la aplicación.
- Sistema, actúa únicamente como actor en el inicio de la aplicación detectando automáticamente si existen nuevos comics por importar (producto de una importación vía iTunes) y en dicho caso lleva a cabo dicha importación de manera automática.

La siguiente tabla resume los casos de uso contemplados, que serán desarrollados a continuación.

Código	Descripción	Actor(es)
CU01	Importar comics	Usuario, Sistema
CU02	Editar Biblioteca	Usuario
CU03	Seleccionar Comics	Usuario
CU04	Trasladar a Otra Colección	Usuario
CU05	Eliminar Comic	Usuario
CU06	Gestionar Colección	Usuario
CU07	Nueva Colección	Usuario
CU08	Cambiar de Colección	Usuario
CU09	Eliminar Colección	Usuario
CU10	Ver Comic	Usuario
CU11	Zoom	Usuario
CU12	Desplazamiento (panning)	Usuario
CU13	Cambio de pagina	Usuario
CU14	Reordenar comic dentro de biblioteca	Usuario
CU15	Establecer Preferencias	Usuario
CU16	Renombrar Colección	Usuario

3.2.1 Descripción textual de casos de uso

3.2.1.1 CU01-Importar Comics

Identificador	CU01
Nombre	Importar Comics
Autor	Sebastián Suelves Sellés
Resumen	Detecta nuevos comics y los incorpora a la biblioteca
Actor(es)	Usuario, Sistema
Precondiciones	Existe uno o mas comics por importar
Postcondiciones	El fichero .cbr o .cbz importado se borrara
Flujo normal	1. El caso de uso se inicia en el arranque de la aplicación o por pulsación del botón refrescar. 2. Si existen comics por importar se informa y se indica mediante un indicador de procesando que se esta trabajando en la importación. 3. Se descomprime el fichero a importar. 4. Se genera la miniatura a partir de la portada. 5. Se crea el objeto de la clase Comic correspondiente y se almacena. 6. Se actualiza la biblioteca de comics de forma adecuada reflejando los efectos de la importación.
Flujos alternativos	Si la importación falla debido a que no se reconozca el formato a la hora de su descompresión se elimina el fichero comprimido de forma silenciosa intentándose importar el fichero comprimido siguiente en caso de existir, o finalizando la importación en caso contrario.
Inclusiones	
Extensiones	

3.2.1.2 CU02-Editar Biblioteca

Identificador	CU02
Nombre	Editar Biblioteca
Autor	Sebastián Suelves Sellés
Resumen	Permite modificar la biblioteca y gestionar las colecciones
Actor(es)	Usuario, Sistema
Precondiciones	Modo edición no activo
Postcondiciones	Modo edición
Flujo normal	1. El caso de uso se inicia cuando el usuario pulsa el botón ***Editar***. 2. Se habilitan las opciones de edición en el interfaz. 3. La salida del modo edición se produce pulsando en el botón ***Ok***.
Flujos alternativos	
Inclusiones	
Extensiones	CU03, CU06

3.2.1.3 CU03-Seleccionar Comics

Identificador	CU03
Nombre	Seleccionar Comics
Autor	Sebastián Suelves Sellés
Resumen	Selecciona uno o mas comics para su edición
Actor(es)	Usuario
Precondiciones	Modo edición
Postcondiciones	Comic añadido/eliminado de la lista de comics seleccionados
Flujo normal	1. El caso de uso se produce mediante la pulsación de un comic de la biblioteca, si el comic ya estaba seleccionado se deseleccionara.
Flujos alternativos	
Inclusiones	
Extensiones	CU04, CU05

3.2.1.4 CU04-Trasladar a Otra Colección

Identificador	CU04
Nombre	Trasladar a Otra Colección
Autor	Sebastián Suelves Sellés
Resumen	Mueve uno o varios comics de una colección a otra permitiendo
Actor(es)	Usuario
Precondiciones	Modo edición y al menos un comic seleccionado
Postcondiciones	Los comics implicados pasan a estar en la nueva categoría
Flujo normal	1. El caso de uso se inicia con la pulsación del botón ***Trasladar***. 2. Se solicita al usuario una categoría a la que mover los comics seleccionados. 3. Los comics son trasladados y se refresca la biblioteca.
Flujos alternativos	
Inclusiones	
Extensiones	

3.2.1.5 CU05-Eliminar Comic

Identificador	CU05
Nombre	Eliminar Comic
Autor	Sebastián Suelves Sellés
Resumen	Elimina uno o varios comics de la biblioteca permitiendo selección múltiple
Actor(es)	Usuario
Precondiciones	Modo edición y al menos un comic seleccionado
Postcondiciones	Los comics en la lista de comics seleccionados son eliminados y se refresca la interfaz
Flujo normal	1. El caso de uso se inicia con la pulsación del botón **Eliminar**. 2. Se pide confirmación al usuario. 3. Los comics seleccionados son eliminados del sistema y se refresca la biblioteca.
Flujos alternativos	
Inclusiones	
Extensiones	

3.2.1.6 CU06-Gestionar Colección

Identificador	CU06
Nombre	Gestionar Colección
Autor	Sebastián Suelves Sellés
Resumen	Permite editar las colecciones
Actor(es)	Usuario
Precondiciones	Modo edición colección no activo
Postcondiciones	Modo edición colección
Flujo normal	1. El caso de uso se inicia con la pulsación del botón ***Colección***, accediéndose a la vista de colecciones. 2. El usuario pulsa el botón ***Editar*** dentro de esta vista. 3. Se sale de este modo pulsando el botón ***Ok*** en dicha vista.
Flujos alternativos	
Inclusiones	
Extensiones	CU09, CU16

3.2.1.7 CU07-Nueva Colección

Identificador	CU07
Nombre	Nueva Colección
Autor	Sebastián Suelves Sellés
Resumen	Crea una nueva colección en la que agrupar un conjunto de comics
Actor(es)	Usuario
Precondiciones	Modo edición colección activo o inactivo
Postcondiciones	Añadida nueva colección
Flujo normal	1. El caso de uso se inicia con la pulsación del botón ***Nueva***, que aparece al pulsar sobre el botón ***Colecciones*** de la vista de la biblioteca. 2. Se solicita al usuario un titulo para la nueva colección, creándose esta en el sistema.
Flujos alternativos	
Inclusiones	
Extensiones	

3.2.1.8 CU08-Cambiar de Colección

Identificador	CU08
Nombre	Cambiar de Colección
Autor	Sebastián Suelves Sellés
Resumen	Selecciona una colección para su visualización
Actor(es)	Usuario
Precondiciones	Modo edición no activo
Postcondiciones	Se cambia la colección activa y se refresca el interfaz
Flujo normal	1. El caso de uso se inicia con la pulsación sobre el titulo de una colección que aparecerá tras la pulsación del botón ***Colecciones***, refrescándose la biblioteca y mostrándose los comics correspondientes a la colección seleccionada.
Flujos alternativos	
Inclusiones	
Extensiones	

3.2.1.9 CU09-Eliminar Colección

Identificador	CU09
Nombre	Eliminar Colección
Autor	Sebastián Suelves Sellés
Resumen	Elimina una colección y los comics que contenga dicha colección
Actor(es)	Usuario
Precondiciones	Modo edición colección
Postcondiciones	Se elimina la colección y los comics pertenecientes a esta
Flujo normal	1. El caso de uso se inicia marcando las colecciones a eliminar. 2. Se pulsa sobre el botón Eliminar, eliminándose la colección y los comics de esta, refrescándose la biblioteca.
Flujos alternativos	
Inclusiones	
Extensiones	

3.2.1.10CU10-Ver Comic

Identificador	CU10
Nombre	Ver Comic
Autor	Sebastián Suelves Sellés
Resumen	Visualiza un comic de la biblioteca
Actor(es)	Usuario
Precondiciones	Modo edición no activo
Postcondiciones	Se visualiza el comic
Flujo normal	1. El caso de uso se inicia pulsando de manera corta (tap) sobre un comic de la biblioteca. 2. Se sale de la visualización de comics pulsando sobre el botón ***Biblioteca*** de la vista de visualización de comics.
Flujos alternativos	
Inclusiones	
Extensiones	CU11, CU13

3.2.1.11CU11-Zoom

Identificador	CU11
Nombre	Zoom
Autor	Sebastián Suelves Sellés
Resumen	Aplica el zoom especificado en función del gesto
Actor(es)	Usuario
Precondiciones	Modo visualización de comics
Postcondiciones	Se aplica o restaura el nivel de zoom indicado por el usuario (gesto pellizco) o en las preferencias de la aplicación para el de doble pulsación
Flujo normal	1. El caso de uso se inicia mediante una doble pulsación sobre la pagina del comic o mediante un gesto tipo pellizco (pitch). 2. Se sale de este modo con otra doble pulsación (la cual restaura el factor de ampliación a escala 1:1) o mediante el gesto pellizco llegando a una escala inferior a 1:1.
Flujos alternativos	
Inclusiones	
Extensiones	CU12

3.2.1.12 CU12-Desplazamiento (panning)

Identificador	CU12
Nombre	Desplazamiento (panning)
Autor	Sebastián Suelves Sellés
Resumen	Desplazamiento a lo largo de una pagina ampliada (panning)
Actor(es)	Usuario
Precondiciones	Modo Zoom dentro de la visualización de comics
Postcondiciones	El área visible del comic con zoom cambia en función del gesto del usuario
Flujo normal	1. El caso de uso se inicia con el uso de gestos de arrastre sobre la pantalla (panning).
Flujos alternativos	
Inclusiones	
Extensiones	

3.2.1.13 CU13-Cambio de pagina

Identificador	CU13
Nombre	Cambio de pagina
Autor	Sebastián Suelves Sellés
Resumen	Cambia la pagina visible
Actor(es)	Usuario
Precondiciones	Si la pagina actual es la primera no podemos movernos a una pagina anterior. Si la pagina actual es la ultima no podemos movernos a una pagina posterior. No estamos en modo zoom (escala 1:1)
Postcondiciones	Cambio de pagina
Flujo normal	1. El caso de uso se inicia con la detección de dos tipos de gestos: desplazamientos horizontales (swipes) o pulsación simple cercana a los bordes de la pagina. 2. Si la pulsación es cercana al borde izquierdo o el gesto de desplazamiento es de izquierda a derecha se cambia a la pagina anterior, si la pulsación es cercana al borde derecho o el gesto de desplazamiento es de derecha a izquierda cambiamos a la pagina siguiente.
Flujos alternativos	
Inclusiones	
Extensiones	

3.2.1.14 CU14-Reordenar Comic dentro de Biblioteca

Identificador	CU14
Nombre	Reordenar Comic dentro de Biblioteca
Autor	Sebastián Suelves Sellés
Resumen	Permite arrastrar los diversos comics y ordenar las posiciones de estos en una colección
Actor(es)	Usuario
Precondiciones	Modo edición no activo
Postcondiciones	Comic en nueva posición
Flujo normal	1. El caso de uso se inicia pulsando, arrastrando, y soltando (drag and drop) el comic en la posición deseada.
Flujos alternativos	
Inclusiones	
Extensiones	

3.2.1.15 CU15-Establecer Preferencias

Identificador	CU15
Nombre	Establecer Preferencias
Autor	Sebastián Suelves Sellés
Resumen	Permite al usuario ajustar la configuración de la aplicación
Actor(es)	Usuario
Precondiciones	Abrimos la aplicación Ajustes de Apple, seleccionamos la aplicación para ajustar las preferencias
Postcondiciones	Los cambios tienen lugar en la aplicación cuando esta vuelve a ejecutarse o pulsamos el botón refrescar de esta
Flujo normal	1. El caso de uso se inicia abriendo la aplicación Ajustes de Apple y pulsando sobre ***Power Comic Reader***. Realizándose en esta los ajustes.
Flujos alternativos	
Inclusiones	
Extensiones	

3.2.1.16CU16-Renombrar Colección

Identificador	CU15
Nombre	Renombra una colección
Autor	Sebastián Suelves Sellés
Resumen	Permite al usuario renombrar una colección
Actor(es)	Usuario
Precondiciones	Modo edición colección
Postcondiciones	La colección tiene un nuevo nombre
Flujo normal	1. El caso de uso se inicia pulsando sobre el nombre de la colección a renombrar. 2. El usuario renombra la colección. 3. El usuario confirma con el botón ***OK***.
Flujos alternativos	
Inclusiones	
Extensiones	

4. Diseño

Debido a que la aplicación debe ser una aplicación nativa para iPad, esta debe diseñarse con la premisa del entorno de programación XCODE y como lenguaje de desarrollo se usaran Objective-C y C++ en algunos frameworks de terceros (UnRar/MiniZip), siendo nuestro objetivo la realización de una aplicación para iOS6 como versión mínima.

4.1 Diagrama estático de diseño

El siguiente diagrama estático recoge las entidades utilizadas por el sistema:

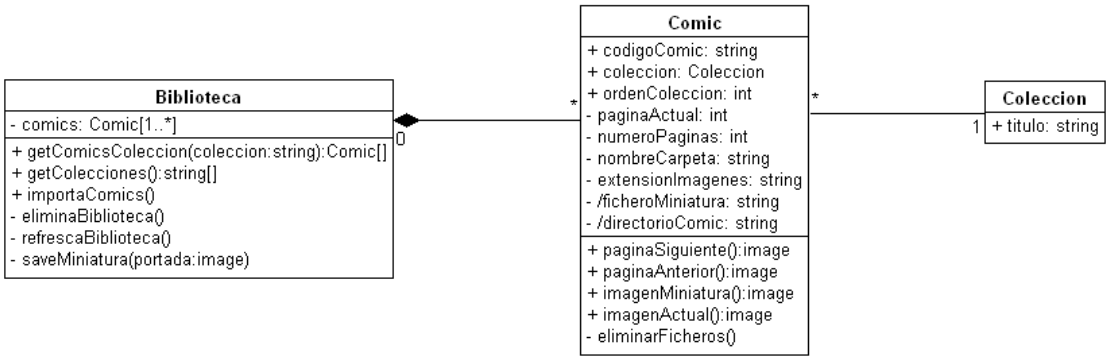


Figura 3. Diagrama estático de diseño

Como puede observarse ha aparecido una nueva clase ***Colección*** en la fase de diseño ya que se precisara posteriormente conocer las distintas colecciones existentes para operaciones como traslado de comics de una categoría a otra.

4.2 Diagrama de secuencia

El principal objetivo del diagrama de secuencia es mostrar las interacciones entre los objetos de un sistema. Dada la naturaleza de la aplicación, el entorno (iOS), la baja cantidad de clases del proyecto, y el uso del paradigma Modelo-Vista-Controlador la forma de interactuar es muy similar en todas las funcionalidades y el tipo de objetos utilizados es homogéneo, por lo que desarrollaremos exclusivamente el diagrama de secuencia correspondiente al caso de uso CU10 por ser este el único que implica interacción entre las clases de la aplicación con creación de objetos (de la clase comic) para ser visualizados tras su selección en la biblioteca.

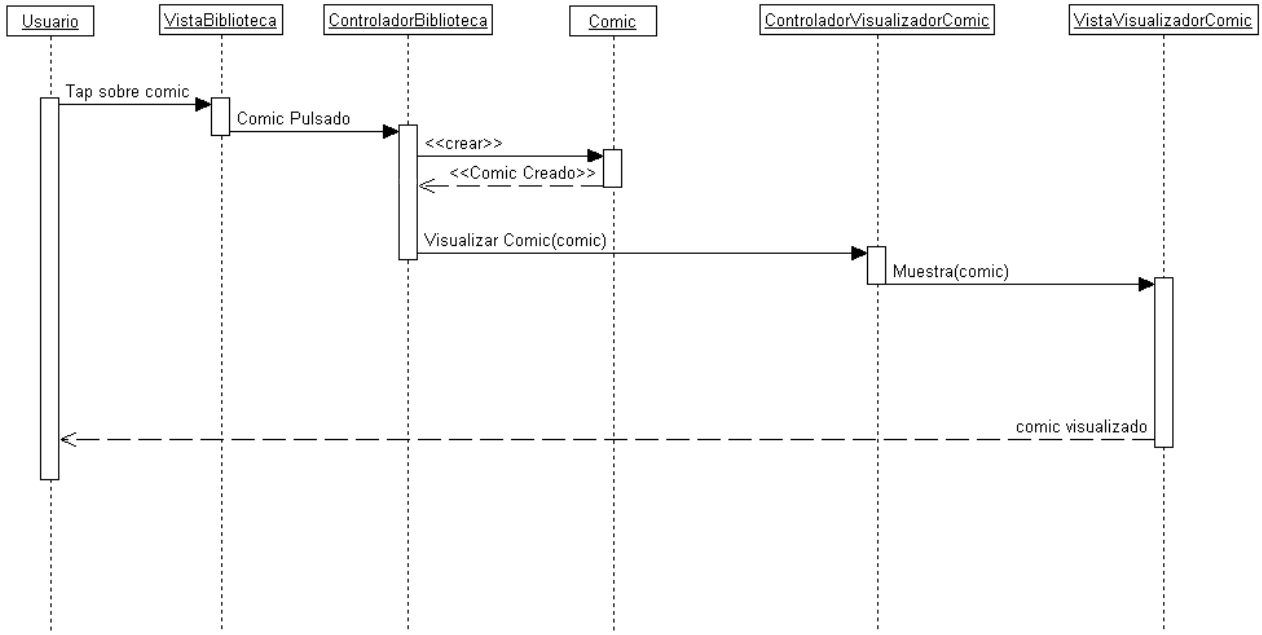


Figura 4. Diagrama de secuencia del caso de uso CU10

4.3 Diseño de la persistencia

El modelo relacional sobre el que se basara la aplicación será:

BIBLIOTECA (codigoComic, codigoColeccion)
codigoComic es clave extranjera hacia COMIC
codigoColeccion es clave extranjera hacia COLECCION

COMIC (codigoComic, codigoColeccion, ordenColeccion, paginaActual, numeroPaginas, nombreCarpeta, extensionImagenes)
codigoColeccion es clave extranjera hacia COLECCION

COLECCIÓN (codigoColeccion, titulo)

AJUSTES (estiloInterfaz, usarDobleTapParaZoomInOut, nivelZoomDobleTap)
estilo puede admitir los valores {'blanco', 'negro', 'papel', 'madera'}
AJUSTES únicamente constara de un registro para el almacenamiento de los ajustes de la aplicación, por lo que no requiere clave primaria.

A continuación se muestra el diagrama Entidad-Relación:

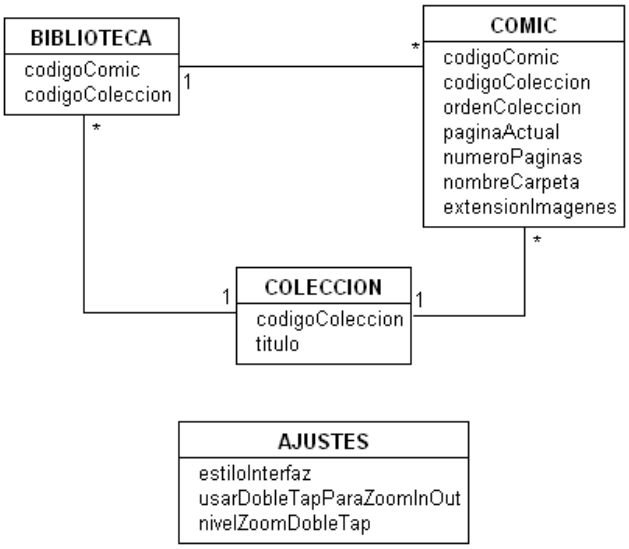


Figura 5. Diagrama del diseño de la persistencia

4.4 Frameworks disponibles en iOS6 de interés para la aplicación

iOS 6 ofrece mas de 1500 frameworks para el desarrollo de aplicaciones, la aplicación a desarrollar precisa diversa funcionalidad para la cual puede escogerse diversos caminos y opciones, por lo que para centrar el desarrollo se ha decidido elegir una serie de frameworks entre los existentes. Dicha selección se ha realizado atendiendo a la inexistencia de alternativa ofrecida por los frameworks estándares ofrecidos por iOS6 (como son el caso del framework Unrar), sencillez de uso, o estandarización. Los framewors seleccionados son:

Propósito	Elección	Proveedor
Descompresión zip (ficheros .cbz)	MiniZip	Cooliris (lic. APACHE)
Descompresión rar (ficheros .cbr)	UnRar	Cooliris (lic. APACHE)
Visualización de la biblioteca en estilo librería	UICollectionView	Apple
Visualización de imágenes	UIScrollView, UIImageView	Apple
Persistencia de ajustes de la aplicación	NSUserDefaults, settings.bundle	Apple
Persistencia de objetos	Core Data	Apple
Acceso a imágenes en el directorio Documents		Apple
Ventana de colecciones emergente	UIPopover, UITableView	Apple
Acceso al sistema de ficheros	NSSearchPathForDirectoriesInDomains, NSFileManager	Apple
Estructuras de memoria: Diccionarios, arrays, etc.	NSFoundation	Apple
Otros aspectos gráficos	UIButton, UILabel, UIToolBar, etc.	Apple
Reordenación de elementos en UIColectionView mediante técnica arrastrar y soltar	LXReorderableCollectionView	Github (lic. MIT)

4.5 Prototipos de la interfaz de usuario

Figura 6. Prototipo vista biblioteca mostrando los comics de la colección: Mortadelo y Filemon

Figura 7. Prototipo vista colecciones

Figura 8. Prototipo edición colección

Figura 9. Prototipo vista biblioteca en modo edición

Figura 10. Prototipo edición de comics (selección)

Figura 11. Prototipo traslado de comics a otra colección

Figura12. Prototipo visualizador de comics

Figura 13. Prototipo visualizador de comics en modo zoom

Figura 14. Prototipo visualizador de comics tras pulsación en zona central, apareciendo botón para retorno a la biblioteca

Figura 15. Prototipo ajustes de la aplicación en la aplicación Ajustes de Apple

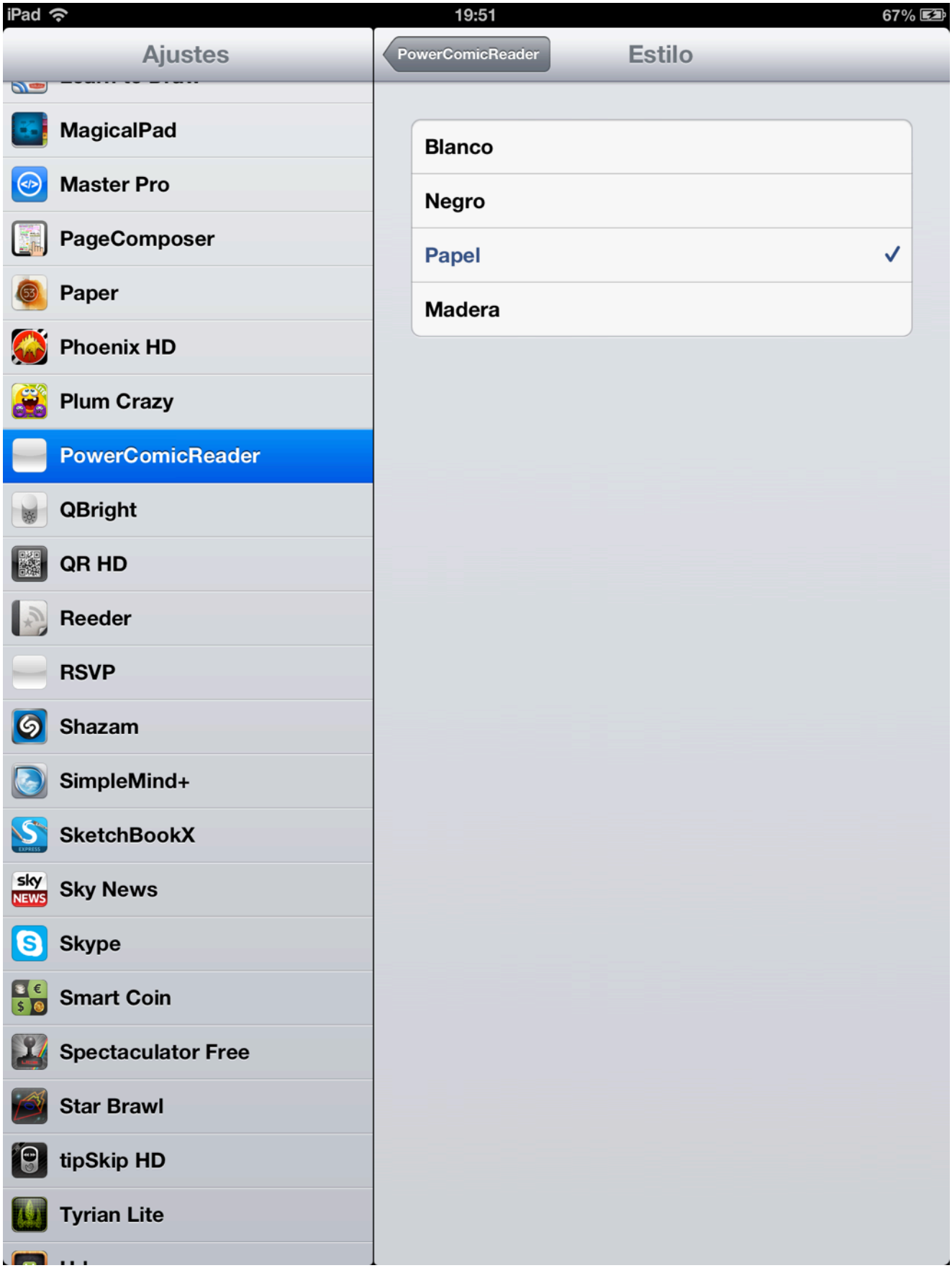


Figura 16. Prototipo ajustes de la aplicación en la aplicación Ajustes de Apple. Ajuste del estilo del interfaz

Figura 17. Prototipo ajustes de la aplicación en la aplicación Ajustes de Apple. Ajuste del nivel de zoom

5. Implementación

Los siguientes epígrafes describen el resultado de la implementación de la aplicación, sus principales características, particularidades, así como problemas específicos que han debido ser solventados para la consecución exitosa del proyecto.

5.1 Detalles generales de la Implementación

A continuación describiremos algunos detalles generales acerca de la implementación tales como el paradigma usado, persistencia, etc.

5.1.1 Paradigma Modelo-Vista-Controlador

Debido a que la aplicación tenia como objetivo la obtención de una aplicación iOS, se ha tratado de usar en todo momento un enfoque Modelo-Vista-Controlador por considerarse el mas adecuado en este entorno para una aplicación visual como se trata **Power Comic Reader**.

Para ello se han implementado un proyecto en XCode 4.5 para iOS 6, en la cual las vistas se han elaborado mediante el uso de **Storyboards**, los modelos obtenidos en las etapas de análisis y diseño mediante clases, y los controladores mediante **ViewControlers** tal como se recomienda en la documentación de Apple. Este enfoque permite independizar el modelo de la vista de modo que en un futuro sea posible reaprovechar los modelos en otras aplicaciones o modificar la vista sin que estos se vean afectados, consiguiéndose de este modo una mayor reutilización de código en el futuro.

Como ventaja añadida este enfoque posibilita el agrupamiento de los distintos tipos de elementos que componen los fuentes de la aplicación en carpetas en función de la categoría a la que pertenezcan. Así pues tenemos las carpetas: **CoreData**, **View**, **Controller**, **Model**, **Supporting Files**, **Iconos**, **Backgrounds**, **Frameworks Visuales**, y **Librerías de Compresión**.

Otra ventaja importante del paradigma Modelo-Vista-Controlador es que cada conjunto formado por un Modelo-Vista-Controlador es independiente del resto permitiendo que puedan invocarse/reutilizarse de modo que un Controlador puede hacer uso de Otro y así sucesivamente.

5.1.2 Persistencia con Core Data

Para la implementación de la persistencia se ha empleado **Core Data** sobre **sqllite3**, permitiendo el acceso a los datos con una orientación a objetos y evitando trabajar directamente con la interfaz C/C++ de sqllite3.

Para ello se han debido de implementar clases de apoyo que definen los datos de interés a almacenar por parte de **Core Data**.

Han presentado especial dificultad en este apartado la comprensión de las relaciones entre entidades de **Core Data**, en concreto como acceder desde una entidad a las entidades

relacionadas y sus datos, para lo cual fue preciso comprender como diseñar el modelo de base de datos **Core Data**:

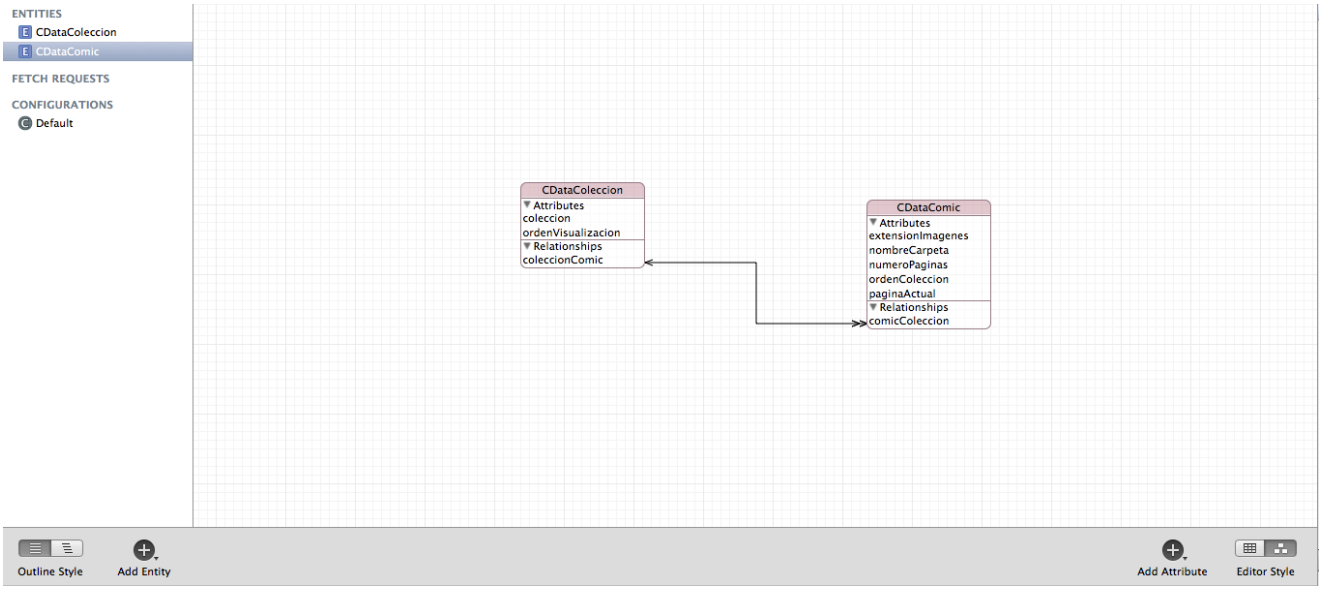


Figura 18. Modelo de datos en Core Data

Como puede observarse en la implementación **Core Data** ha desaparecido del modelo de persistencia la biblioteca pues su cometido era relacionar los comics vinculados a una colección de modo bidireccional, esto se consigue en **Core Data** simplemente con una relación en ambos sentidos no precisándose códigos que actúen como clave extranjera o primaria, estas cuestiones son responsabilidad de **Core Data** y son implementadas por este de modo transparente para el programador.

A partir de este modelo se generan las clases de Core Data, a continuación mostramos la clase generada para la entidad CDataComic:

```
//
//  CDataComic.h
//  PowerComicReader
//
//  Created by Sebastian Suelves Selles on 04/05/13.
//  Copyright (c) 2013 Sebastian Suelves Selles. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@class CDataColeccion;

@interface CDataComic : NSManagedObject

@property (nonatomic, retain) NSString * extensionImagenes;
@property (nonatomic, retain) NSString * nombreCarpeta;
@property (nonatomic, retain) NSNumber * numeroPaginas;
@property (nonatomic, retain) NSNumber * ordenColeccion;
@property (nonatomic, retain) NSNumber * paginaActual;
@property (nonatomic, retain) CDataColeccion *comicColeccion;

@end

//
//  CDataComic.m
//  PowerComicReader
//
//  Created by Sebastian Suelves Selles on 04/05/13.
//  Copyright (c) 2013 Sebastian Suelves Selles. All rights reserved.
//
```

```
#import "CDataComic.h"
#import "CDataColeccion.h"


@implementation CDataComic
@dynamic extensionImagenes;
@dynamic nombreCarpeta;
@dynamic numeroPaginas;
@dynamic ordenColeccion;
@dynamic paginaActual;
@dynamic comicColeccion;
@end
```

Core Data se basa en el manejo de Contexto de Objetos Manejados, de modo que los objetos anteriormente definidos son controlados por **Core Data** de un modo transaccional, es decir, si modificamos un objeto, este automáticamente será volcado persistentemente al almacén de datos siempre y cuando validemos la transacción o anulara los cambios en caso de anular la transacción. Un punto muy interesante y útil del uso de **Core Data** es que si diseñamos adecuadamente nuestro modelo de datos, podemos desde un objeto que represente a una colección seguir la relación y acceder a todos los objetos comics de esa colección y viceversa, lo cual facilita mucho el trabajo pues estas relaciones son mantenidas directamente por **Core Data** sin necesidad de definición del equivalente en SQL de claves primarias y extranjeras (de hay que no se requieran campos para cruzar las tablas como ocurriría en una base de datos SQL).

Así en siguiente código correspondiente a la inicialización del la biblioteca con una colección podemos observar la línea:

```
 NSArray *arrayComics = [[coleccionTemp.coleccionComic allObjects]
sortedArrayUsingDescriptors:sortDescriptors];
```

Mediante esta línea a partir de una colección podemos obtener todos los objetos del tipo Comic que están relacionados con la colección a través de la relación **coleccionComic** definida en el modelo **Core Data**.

```
- (id)initWithColeccion:(NSFetchedResultsController *)comicsColeccion ofColeccion:(CDataColeccion
*)coleccion
{
    self = [super init];

    if (self)
    {
        //inicializamos la coleccion
        self.coleccion = coleccion;

        self.comics=[[NSMutableArray alloc]init];
        self.cdComicsColeccion = comicsColeccion;

        NSArray *array = [comicsColeccion fetchedObjects];

        for (NSInteger indice=0; indice<[array count]; indice++)
        {
            //cargamos los datos desde core data y los cargamos en nuestro modelo
            CDataColeccion *coleccionTemp = (CDataColeccion *) [array objectAtIndex:indice];

            NSSortDescriptor *sortDescriptor;
            sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"ordenColeccion" ascending:YES];
            NSArray *sortDescriptors = [NSArray arrayWithObject:sortDescriptor];


            NSArray *arrayComics = [[coleccionTemp.coleccionComic allObjects]
sortedArrayUsingDescriptors:sortDescriptors];
            self.coleccionActiva = coleccion.coleccion;

            for (CDataComic *cdComic in arrayComics) {
                NSLog(@"%@", cdComic);
                Comic *comic = [[Comic alloc] initWithColeccion:coleccionTemp.coleccion
AndOrdenColeccion:cdComic.ordenColeccion AndPaginaActual:cdComic.paginaActual
```

```
                                                    AndNumPaginas:cdComic.numeroPaginas
AndDirectorio:cdComic.nombreCarpeta AndExtensionImagenes:cdComic.extensionImagenes];
                NSLog(@"%@ , %@", comic.directorio, comic.ordenColeccion);
                //añadimos el comic a nuestro modelo
                [self insertComicsObject:comic inComicsAtIndex:[self.comics count]
AndInsertInCoreData:NO];
            }
        }

        //importamos los comics y los reordenamos
        [self importaComics];
        [self reordenaComics];
    }
    return self;
}
```

5.1.3 Frameworks de terceros o propios

Algunas funcionalidades han requerido el uso de frameworks adicionales a los suministrados por Apple, este es el caso del Framework LXReorderableCollectionView que debido a limitaciones del componente UICollectionView en lo referente a permitir un tratamiento tipo arrastrar y soltar para reordenar los objetos del UICollectionView, pues esta funcionalidad no existe entre las funcionalidades ofrecidas por este framework. Lamentablemente esto se descubrió en una etapa adelantada del proyecto por lo que tuvo que buscarse una alternativa a ello.

Así mismo se han elaborado una clase de utilidad para el manejo de UIAlertView de modo modal, lo que facilitara en el futuro su uso en diversas aplicaciones al autocontener el manejo de delegados y evitando tener que implementarlo cada vez que deba usarse un UIAlertView únicamente para poder recibir el resultado de este.

5.2 Detalles de los Modelos-Vistas-Controladores que componen la aplicación

Como hemos comentado anteriormente la aplicación esta compuesta de diversos MVCs diseñados para encapsular la distinta funcionalidad requerida por la aplicación. En los siguiente epígrafes describiremos cada una de estas agrupaciones lógicas, haciendo referencia al controlador de esta.

5.2.1 Modelo-Vista-Controlador: BibliotecaViewController

Este MVC es el primero que se ejecuta en la aplicación, hace uso de los modelos de comics y biblioteca, e implementa gran parte de la funcionalidad de la aplicación. Este MVC se encarga de la visualización de la colección activa mediante el uso del modelo Biblioteca y de los comics que componen dicha colección.

Para ello implementa diversos delegados:

- UICollectionViewDataSource para la representación en forma de biblioteca de los comics que componen una colección. Para ello usa como fuente un objeto Biblioteca que suministra los datos.
- LXReorderableCollectionViewDelegateFlowLayout, para suplir las limitaciones de UICollectionView en lo referente a arrastrar y soltar objetos visualmente. Es parte del framework de terceros LXReordenableCollectionView.
- ColeccionViewDelegate, delegado desarrollado por nosotros que nos permite recibir la colección seleccionada en el popover Colección. Se apoya en el protocolo requerido por la clase Colección que obliga a la implementación del método **coleccionSeleccionada** recibiéndose a través de este la colección seleccionada.
- UIPopoverControllerDelegate, el cual nos permite controlar el popover sobre el que se visualizara la lista de colecciones tras la pulsación del botón **Colecciones**.

Así pues tenemos los siguientes elementos:

- Una barra de botones en la parte superior que nos permite establecer las acciones a realizar.
- Un fondo personalizable en función del estilo.
- Un objeto UICollectionView que nos permite visualizar, seleccionar, y deseleccionar los comics de la colección que trabajando en conjunción con el framework LXReorderableCollectionView nos permite además poder mover mediante una pulsación larga sobre un comic a una nueva posición dentro de la colección en modo edición. Si la pulsación es corta y fuera del modo de edición, entra en juego el segue abriéndose el comic en cuestión para su lectura.
- Acceso al MVC ColeccionViewController mediante un popover, desde el que podemos crear, borrar, y renombrar las colecciones. Si no editando las colecciones y pulsamos sobre una de ellas, esta desencadenara el envío al MVC BibliotecaViewController de la selección seleccionada.
- Reusabilidad del MVC ColeccionViewController para el traslado de comics, para ello se usa este mismo MVC pero llevando constancia mediante un booleano **trasladandoComic**

de modo que cuando recibimos la colección a la que se ha cambiado, en la implementación de la función de delegado **coleccionSeleccionada** sabiendo que estamos trasladando uno o varios comics se lleva a cabo el traslado y la selección de dicha colección.

Algunos aspectos que han resultado mas complejos en la implementación de este MVC son:

- Cambiar la forma de pensar a una orientación MVC.
- Comprender el uso de delegados y protocolos en Objective-C.
- Aprender las limitaciones de algunos framework y componentes como son el caso de la problemática de no poder ocultar botones en un UIToolBar lo que obligo a llevar una copia del array de objetos que componen el UIToolBar completo para eliminar los objetos a ocultar y poderlos recuperar con posterioridad en caso de precisar visualizarlos de nuevo, la imposibilidad de mostrar una etiqueta sobre el UIToolBar debiéndose emular mediante un botón sin bordes que asemejara a una etiqueta, o la imposibilidad de arrastrar un elemento (comic) mediante el UICollectionView requiriéndose buscar y estudiar un framework de apoyo como LXReorderableCollectionView.

5.2.2 Modelo-Vista-Controlador: ColeccionViewController

Este MVC tiene una doble funcionalidad, por un lado implementa el mantenimiento de las colecciones permitiendo su creación, modificación, y eliminación. Además permiten seleccionar una colección entre las existentes facilitando la selección de una colección para cambiar la colección activa o llevar a cabo el traslado de comics desde la colección activa a otra colección existente. Como puede observarse en el código este MVC no hace uso de un modelo implementado por una clase definida de modo separado, sino que directamente implementa una tabla de objetos String, no considerándose necesario crear una clase Colección para implementar un String, ya que la tabla de objetos String proporciona toda la funcionalidad requerida: la cadena almacena el titulo de la colección, y su posición en la tabla determina el orden de visualización.

El MVC implementa a nivel de delegados únicamente los necesarios para el funcionamiento del UITableView (UITableViewDataSource, UITableViewDelegate, UITextFieldDelegate) aunque este MVC implementa un protocolo que le permite comunicarse con el MVC que lo invoque. La implementación del protocolo se lleva a cabo mediante el siguiente fragmento de código:

```
//definicion del protocolo de comunicacion de la coleccion seleccionada a implementar en el delegado
@protocol ColeccionViewDelegate <NSObject>

@required
-(void)coleccionSeleccionada:(NSString *)nuevaColeccion;
@end
```

El uso de este protocolo se lleva a cabo en el MVC BibliotecaViewController por una parte indicando que la clase que lo implementa se acogerá al protocolo **ColeccionViewDelegate**:

```
@interface BibliotecaViewController () <UICollectionViewDataSource,
LXReorderableCollectionViewDelegateFlowLayout, ColeccionViewDelegate, UIPopoverControllerDelegate>
```

Lo que implica que deberá implementarse **coleccionSeleccionada** en el MVC **BibliotecaViewController**:

```
#pragma mark - ColeccionViewDelegate
-(void) coleccionSeleccionada:(NSString *)nuevaColeccion
{
    NSLog(@"Coleccion Seleccionada: %@", nuevaColeccion);

    //delegado que recibe la coleccion seleccionada en el popover

    //acualizamos la interfaz y la coleccionactiva
    [self actualizaBotonLabelColeccion:nuevaColeccion];
    self.coleccionActiva = nuevaColeccion;

    //Grabamos como activa para el reinicio de la aplicacion la coleccion seleccionada
    NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];

    // saving an NSString
    [standardUserDefaults setObject:nuevaColeccion forKey:@"coleccionActiva"];
    [standardUserDefaults synchronize];

    //si habia que trasladar los comics hacemos el traslado a la coleccion seleccionada
    if (self.trasladandoComic) {
        NSLog(@"Trasladando Comic parte 2");
        self.trasladandoComic = NO;

        //Trasladamos los comics seleccionados
        for (Comic *comic in [self.comicsSeleccionados copy]) {
            [self.biblioteca trasladaComic:comic toColeccion:nuevaColeccion];
            [self.comicsSeleccionados removeObject:comic];
        }
        //recargamos los datos
        dispatch_async(dispatch_get_main_queue(), ^{
            //recargar datos collectionview
            [self.collectionView reloadData];
        });
    }

    //actualizamos la biblioteca
    [self actualizaBiblioteca];
    if ([self.coleccionPopoverController isPopoverVisible])
    {
        //ocultamos popover
        [self.coleccionPopoverController dismissPopoverAnimated:YES];
        self.coleccionPopoverController = nil;
    }
}
```

Como puede observarse en función de si estamos trasladando un comic o simplemente seleccionándolo somos capaces de reutilizar el MVC **ColeccionViewController**.

En cuanto a las dificultades planteadas por este MVC cabe destacar la dificultad de edición del UITableView directamente sobre este, pues existen pocos ejemplos de cómo llevar esto cabo ya que generalmente suele emplearse un segue que permita la edición de una célula a nivel de inserción/modificación en un MVC diferente haciendo uso de un UINavigationController que permita una vez finalizada la edición volver al MVC anterior. Especialmente ha sido problemático la inserción de nuevas colecciones y la ocultación de teclado una vez finalizada la edición.

5.2.3 Modelo-Vista-Controlador: VisualizadorComicViewController

La funcionalidad de este MVC es la visualización de un comic concreto con opciones de zoom y desplazamiento, aceptando diversos gestos táctiles: pulsación en los bordes de las paginas permiten el acceso a la pagina siguiente y anterior así como los desplazamientos de izquierda a derecha (swipes) fuera del modo zoom, la doble pulsación sobre la imagen nos permite aplicar un nivel de zoom configurable y anulando dicho zoom mediante una segunda doble pulsación, además se posibilita mostrar una barra de utilidades oculta mediante una pulsación en la zona

central de la pantalla que nos permite volver al MVC principal, además se permite el ajuste manual del nivel de zoom mediante gesto multi táctil con dos dedos acercando y alejando. Dentro del modo zoom se permite el desplazamiento de la imagen mediante arrastre con un dedo.

En cuanto a delegados que implementa únicamente implementa UIScrollViewDelegate, para permitirnos llevar a cabo las opciones de zoom y de desplazamiento dentro de este. Esta clase recibe un objeto comic a visualizar y haciendo uso intensivo de la clase modelo **Comic** controla toda la visualización de este embebiendo un UIImageView en el UIScrollView.

En cuanto a dificultades planteadas por este MVC cabe destacar un error que se da en iOS6 al combinar la opción de Autolayout con UIScrollView y UIImageView que provoca que al alejar el zoom las constraints ejecutadas por el Autolayout provoquen que se pierda el origen del UIImageView con respecto al UIScrollView, tras una investigación en diversos foros (stackoverflow, developer.apple.com, etc.) se ha descubierto que es un problema provocado por el uso de Autolayout que aplica cambios para la adaptación automática de la vista en un orden distinto al que se aplicaría si no tuviese Autolayout, existen numerosas notificaciones del error a Apple, el cual aun no tiene una solución sencilla, por lo que se ha optado por desactivar la opción de Autolayout (de todo el proyecto de XCode, pues no se permite la desactivación para una única vista) y se ha debido ajustar las diversas vistas usando el método ofrecido por iOS 5. Algún otro punto de dificultad han sido coordinar los gestos táctiles para que no entraran en conflicto unos con otros. Así por ejemplo para evitar la aparición del menú oculto cuando se hacia doble pulsación (aparecía en la primera pulsación para desaparecer en la segunda) ha debido estudiarse el tema de los gestos multi táctiles en profundidad para encontrar la solución que consiste en indicarle al sistema que solo ejecute un gesto de simple pulsación si no ha tenido éxito uno de doble pulsación (lo cual provoca un pequeño retraso inapreciable para el usuario), mediante la línea:

```
[singleTap requireGestureRecognizerToFail:doubleTap];
```

5.2.4 Clase Comic

Esta es una de las clases correspondientes al modelo, la cual modela la entidad **Comic**, su implementación hace hincapié en el intento de minimizar el uso de memoria, por lo que los objetos de esta clase mantienen información de estado que les permite ofrecer las distintas paginas que componen el comic sin almacenar este en memoria, sino accediendo al sistema de ficheros cada vez que deba recuperarse una pagina en concreto y sirviendo esta imagen como salida para su procesamiento. Esta decisión no ha tenido repercusión en el rendimiento aparente de la aplicación manteniendo un nivel de uso de memoria mínimo. A continuación mostramos el método que devuelve la pagina actual del comic:

```
- (UIImage *)imagenActual
{
    //devolvemos la pagina actual
    NSString *ficheroImagen;

    ficheroImagen = [NSString stringWithFormat:@"%@%@%@", self.directorioComic, [self.paginaActual
stringValue], self.extensionImagenes];

    return [UIImage imageWithContentsOfFile:ficheroImagen];
}
```

5.2.5 Clase Biblioteca

Esta clase del modelo incluye la gestión de la biblioteca de la colección activa, es decir los comics que componen las colecciones, implementando además la gestión de estos a nivel de importación de comics (lo cual incluye la adición de estos a la biblioteca, y su descompresión), y los métodos relativos al mantenimiento de dicha biblioteca: borrar un comic de la biblioteca, traslado de comics, etc.

Unos de los puntos que han planteado mayor dificultad ha sido la inclusión de las librerías de descompresión en el proyecto (recordar que gran parte de estos frameworks están escritos en C/C++), pues han debido ajustarse opciones de compilación y aprender como incluir correctamente ficheros fuentes al proyecto desde una fuente externa, en el principio debido a la falta de experiencia en el entorno se cometió el error de al arrastrar los ficheros que componían dichos frameworks (UnRar.h, UnRar.m, MiniZip.h, y MiniZip.m) no marcar la casilla de **Agregar al proyecto PowerComicReader**, lo que ocasionaba que se recibieran errores de compilación al intentar usar dichos frameworks, resultando muy difícil dictaminar donde estaba el problema: opciones de compilación, versión del compilador, etc. Finalmente mediante prueba y error se descubrió que debía marcarse esta opción al arrastrar sobre un proyecto de XCode ficheros para que formen parte de este.

A continuación mostraremos un fragmento de código fuente, en el que se muestra el uso de estos frameworks para descomprimir un fichero (seleccionándose el sistema de descompresión en función de la extensión del objeto):

```
- (NSNumber *) descomprimirFichero:(NSString *)ficheroComprimido
enDirectorioDescompresion:(NSString*)directorioDescompresion yDetectarExtension:(NSString
**)extensionComic
{
    NSNumber* contadorPaginas;
    NSString* extension;

    contadorPaginas = @0;

    id archive = nil;

    extension = [ficheroComprimido pathExtension];

    //en funcion de la extension del fichero usamos un descompresor u otro
    if (![extension caseInsensitiveCompare:@"cbr"] || ![extension caseInsensitiveCompare:@"rar"])
    {
        archive = [[UnRAR alloc] initWithArchiveAtPath:ficheroComprimido];
    }
    else
    {
        archive = [[MiniZip alloc] initWithArchiveAtPath:ficheroComprimido];
    }
    [archive setSkipInvisibleFiles:YES];

    //obtenemos la lista de los ficheros dentro del fichero comprimido, ordenando por nombre
    NSMutableArray* listaFicheros = [NSArray arrayWithArray:[[archive retrieveFileList]
sortedArrayUsingComparator:^(NSString* a, NSString* b) {
        return [a compare:b options:NSNumericSearch];
    }]];

    for (NSString* file in listaFicheros)
    {
        //descomprimimos los ficheros de imagenes
        extension = [file pathExtension];
        if (![extension caseInsensitiveCompare:@"jpg"] || ![extension caseInsensitiveCompare:@"jpeg"]
||
            ![extension caseInsensitiveCompare:@"png"] || ![extension caseInsensitiveCompare:@"gif"])
{
            NSLog(@"fich: %@", file);
```

```
            //contamos las paginas descomprimidas
            unsigned int contador = [contadorPaginas unsignedIntValue] + 1;

            contadorPaginas = [NSNumber numberWithUnsignedInt:contador];

            //calculamos el destino del fichero descomprimido renombrando cada uno de los ficheros a
partir del 1.extension
            NSString* destino = [[directorioDescompresion
stringByAppendingPathComponent:[contadorPaginas stringValue]]
stringByAppendingFormat:@".%@",extension];
            NSLog(@"Destino: %@", destino);

            //descomprimimos el fichero
            if ([archive extractFile:file toPath:destino]) {
                NSLog(@"Fichero descomprimido cbr: %@", destino);
            }
        }
    }

    *extensionComic = [NSString stringWithFormat:@".%@",extension];

    //generamos la miniatura a partir de la portada
    [self saveMiniatura:directorioDescompresion desdeImagen:[UIImage
imageWithContentsOfFile:[[directorioDescompresion stringByAppendingPathComponent:@"1"]
stringByAppendingFormat:@".%@",extension]]];

    //devolvemos el numero de paginas descomprimidas
    return contadorPaginas;
}
```

5.2.6 Clases de apoyo

Además de los MVC que componen la aplicación han debido de desarrollarse/emplearse algunas clases de apoyo como son:

PreferenciasUsuario	Para la recuperación de los datos de ajuste
ModalAlertDelegate	Permite el uso de UIAlertViews modales sin necesidad de implementar delegados en nuestra aplicación, parando la ejecución en su invocación y retomando la ejecución junto con la respuesta del UIAlertView
UnRar y MiniZip	Para la descompresión de ficheros rar/cbr y zip/cbz respectivamente

Es especialmente interesante destacar el caso de ModalAlertDelegate por su simplicidad y reutilizabilidad, por lo general para usar un UIAlertView debemos por un lado lanzar la visualización del UIAlertView y preparar una función que implemente el delegado de este y que recoja el resultado del UIAlertView (pulsación de un botón u otro, un texto de respuesta, etc.), ya que la ejecución continua hasta finalizar la función desde donde se llama a un UIAlertView estándar, recibiéndose el resultado de la ejecución del UIAlertView de forma asíncrona a través del delegado de esta, esto provoca que el código de decisión en función del resultado deba desplazarse a dicha método que implemente el delegado, por lo que se pierde bastante legibilidad de código (empeorando si usásemos diversos UIAlertViews para distintos propósitos) por ello se ha implementado esta Clase gracias a la que no precisamos hacer uso de delegados en la clase que pretende usarla, deteniéndose la ejecución y retornando del método llamante y retornando directamente la respuesta del UIAlertView, lo que permite que acto seguido pueda decidirse la acción a llevar a cabo en función de la información suministrada por el UIAlertView. A continuación mostramos como quedaría el código llamante gracias a la implementación de ModalAlertDelegate:

```
- (IBAction)botonEliminarPulsado:(UIBarButtonItem *)sender
{
    NSLog(@"Pulsado Eliminar");

    //AlertView modal para pedir confirmacion de borrado
    NSString *mensajeAlertView = NSLocalizedString(@"deleteConfirmation", @"This action will delete %d
comics\nAre you sure?");
    mensajeAlertView = [NSString stringWithFormat:mensajeAlertView, [self.comicsSeleccionados count]];
    NSString *tituloAlertView = NSLocalizedString(@"Attention", @"Attention");
    UIAlertView* confirmaBorrado = [[UIAlertView alloc] initWithTitle:tituloAlertView
message:mensajeAlertView delegate:nil
                                                    cancelButtonTitle:NSLocalizedString(@"No", @"No")
                                                    otherButtonTitles:NSLocalizedString(@"Yes",
@"Yes"), nil];
    confirmaBorrado.alertViewStyle = UIAlertViewStyleDefault;
    ModalAlertDelegate *delegate = [ModalAlertDelegate delegateWithAlert:confirmaBorrado];
    NSString* respuesta;
    NSUInteger botonPulsado;
    botonPulsado = [delegate show];

    //si pulsamos "Si" borramos los comics seleccionados
    if (botonPulsado==1)
    {
        for (Comic *comic in [self.comicsSeleccionados copy])
        {
            //borramos los ficheros que conforman el comics
            [self.biblioteca removeComicsObject:comic];
            //borramos el comic del modelo de datos
            [self.comicsSeleccionados removeObject:comic];
        }
        dispatch_async(dispatch_get_main_queue(), ^{
            //recargar datos collectionview
            [self.collectionView reloadData];
        });
    }
}
```

5.3 Otros aspecto de la aplicación: múltiples idiomas, auto rotación, transferencia de ficheros vía iTunes, y ajustes de la aplicación

Además del trabajo visual habitual de los MVC han debido de trabajarse diversos aspectos de la aplicación como son el soporte de múltiples idiomas (en nuestro caso Ingles/Español), la auto rotación de las vistas en cualquier posición apaisada o vertical, la importación de los comic vía iTunes, y los ajustes de la aplicación desde la aplicación Ajustes estándar de iOS.

Algunas de estas tareas han resultado sencillas como el activar la transferencia de ficheros vía iTunes que consiste en activar el booleano **Application supports iTunes file sharing** en el fichero **PowerComicReader-info.plist**:

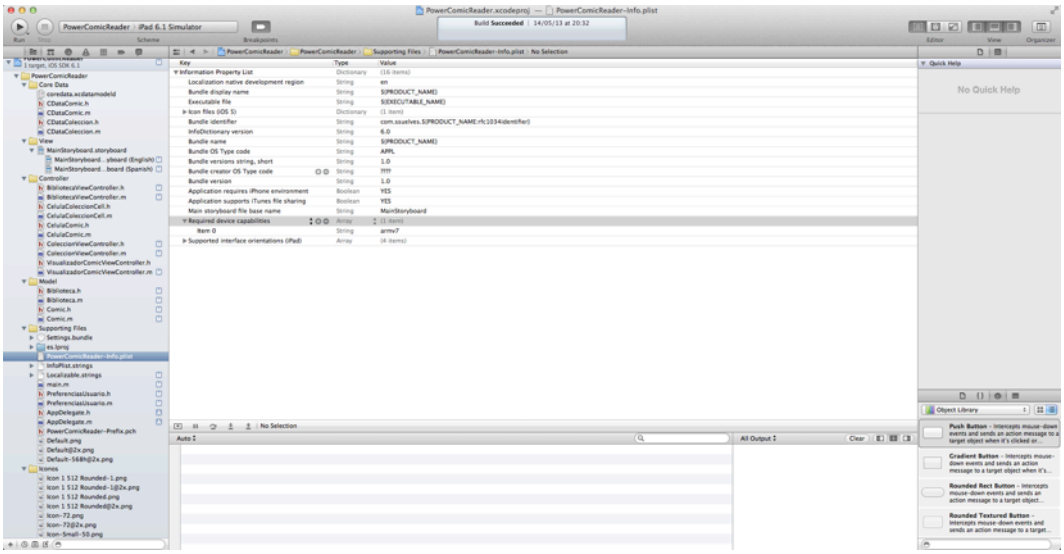


Figura 19. Fichero PowerComicReador-info.plist en XCode

Otras como al auto rotación han planteado mas dificultades en su comienzo al intentar usar la nueva característica de Autolayout de iOS 6, debiéndose implementar de modo similar a iOS 5, asignando los orígenes y si deben expandirse.

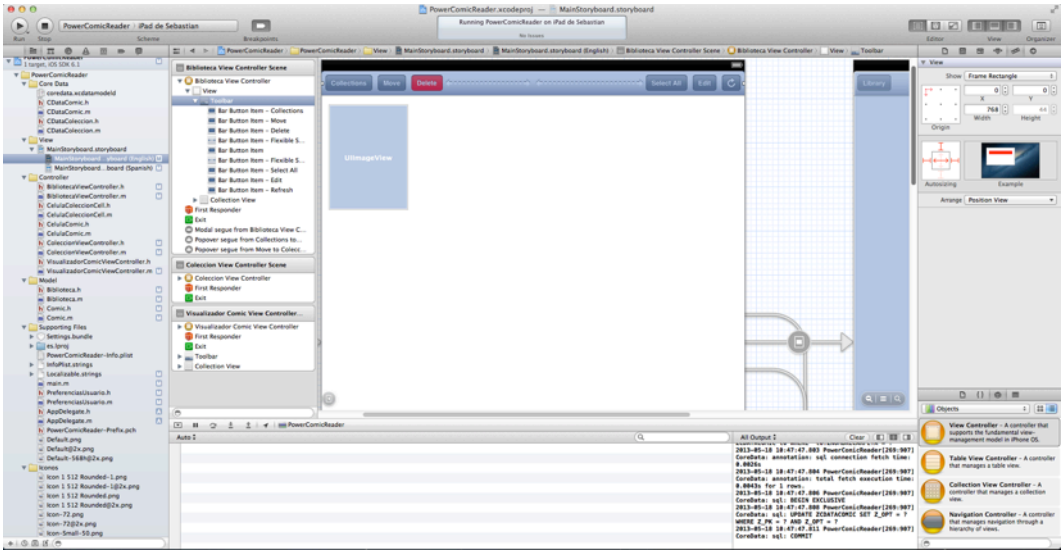


Figura 20. Preparación para auto rotación de UIToolBar, se define el origen desde arriba en la sección "Ruler" y que se expandirá horizontalmente dado soporte así a orientaciones verticales/apaisadas

En cuanto al soporte de ajustes de preferencias de usuario, esta ha resultado bastante sencilla pues únicamente se requiere añadir un fichero **Settings.bundle** al proyecto XCode definir las categorías de modo grafico, posibles valores, y valores por defecto. Para su uso por parte de la aplicación basta con hacer uso de NSUserDefaults, la aplicación además de leer las preferencias del usuario actualiza como preferencia la ultima colección usada, de modo que en caso de refresco o reapertura de la aplicación se muestre la ultima colección usada, para ello se usa el fragmento de código siguiente:

```
//Grabamos como activa para el reinicio de la aplicacion la coleccion seleccionada
    NSUserDefaults * standardUserDefaults = [NSUserDefaults standardUserDefaults];

    // saving an NSString
    [standardUserDefaults setObject:nuevaColeccion forKey:@"coleccionActiva"];
    [standardUserDefaults synchronize];
```

En cuanto a la carga de las opciones de usuario, estas se han separado en una nueva clase denominada **PreferenciasUsuario**, pues debido a que la opción estilo requiere que en función de la opción elegida se emplee una u otra textura debe hacerse una secuencias de instrucciones tipo **if** para ver que textura aplicar, además es muy probable la incorporación de nuevos estilos y opciones en el desarrollo futuro por lo que ha parecido una buena idea la separación de la carga de las preferencias en una clase aparte.

Por ultimo destacar de la implementación del soporte multi idioma este se ha implementado en dos vertientes, en primer lugar al establecer las opciones de soporte muti idioma las vistas se han separado en dos Storyboards uno en Ingles y otro en Español, resultando sencilla por tanto la traducción pues basta con modificar las propiedades de los componentes de las distintas vistas. Por otro lado para el soporte de los mensajes en códigos estos se han definido en dos ficheros **localizable.string** que contienen las parejas clave/valor necesarias para el uso de NSLocateString, permitiéndonos obtener la cadena correcta en función del lenguaje en que este configurado nuestro dispositivo:

```
NSString *deselecciona = NSLocalizedString(@"Select None", @"BibliotecaViewController Select None");
```

5.4 Soporte de resolución estándar y resoluciones retina

La aplicación soporta a nivel de interfaz tanto la resolución estándar del iPad 1, 2, y mini, así como las resoluciones retinas de los iPad 3, y 4. Sin embargo dado el carácter de la aplicación esto no puede aplicarse a los comics, pues los comics en formato .cbz/.cbr son al fin y al cabo un conjunto de ficheros de imágenes escaneadas a partir de un comic impreso, lo cual limita su resolución y puede que en función de si la resolución es o no retina, sea mas visible el pixelado u otros efectos provocado por el escaneado de las diversas paginas del comic en cuestión, siendo esta una limitación de la fuente de la que proviene el comic y no de la aplicación misma. No obstante todos los iconos, y fondos de la aplicación soportan ambas resoluciones para en la medida de lo posible ofrecer una calidad optima (al menos a nivel de interfaz).

5.5 Capturas de pantalla de la aplicación final

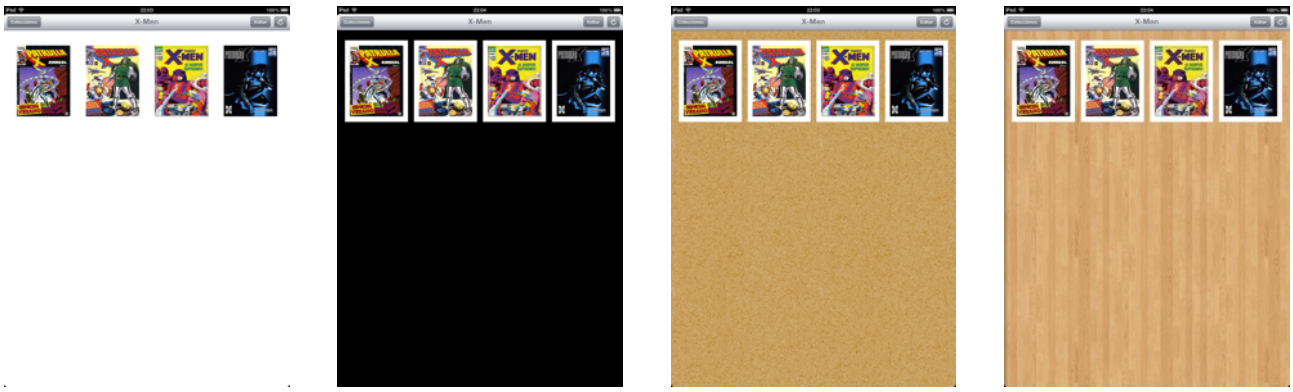


Figura 21. Estilos de la aplicación: Blanco, Negro, Papel, Madera

Figura 22. Selección de comics, Visualización, Zoom, Menú oculto

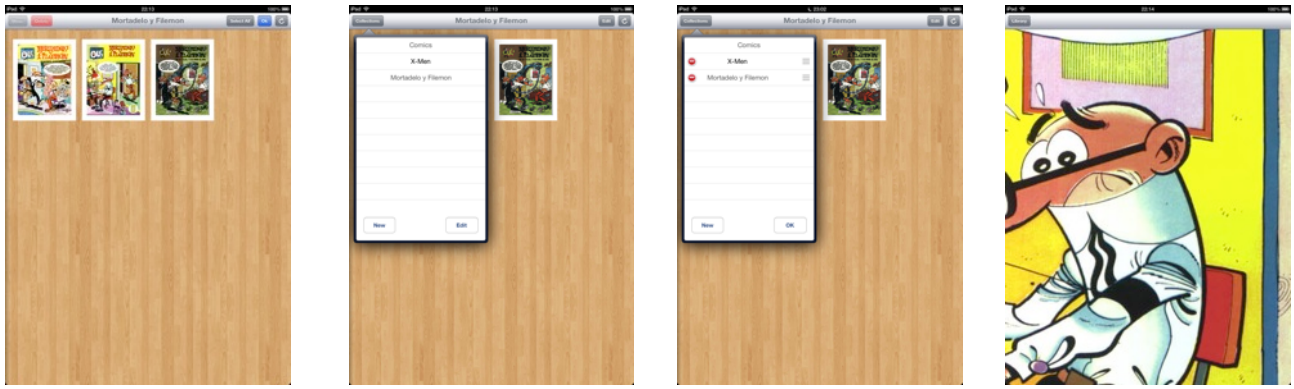


Figura 23. Interfaz en Ingles: Edición, Colecciones, Edición de colecciones, menú oculto en visualización

Figura 24. Detalle de petición de confirmación en el borrado de comics

5.6 Notas sobre como testear la aplicación en el simulador iOS de XCode

La aplicación precisa de comics en formato .cbr o .cbz para su funcionamiento, estos deben importarse en el directorio comics del iPad, esto es muy sencillo si se dispone de un dispositivo físico, sin embargo si solo disponemos del simulador de XCode, esto puede ser mas complejo pues no podremos usar iTunes, por ello a continuación doy unas indicaciones de cómo transferir al simulador los comics a probar.

Esta guía presume que se dispone de Mac OSX Mountain Lion (10.8.3) y XCode 4.6.2, en otro caso la ruta indicada podría variar.

El simulador de iPad de XCode debería estar en la ruta:
/Users/EL_USUARIO_DE_NUESTRA_MAQUINA/Library/Application\ Support/iPhone\ Simulator/6.1

Una vez situados en este directorio, aparecerán una serie de directorios: **Applications**, **Librería**, **Media**, **Root**, **tmp**. El directorio de nuestra aplicación estará dentro de **Applications**, sin embargo es posible que aquí tengamos dos tipos de incidencias:

1) El directorio **Applications** no existe, esto es debido a que estamos en un sistema recién instalado, o se ha usado la opción **Restablecer contenidos y ajustes** de la aplicación Simulador iOS. Para corregirlo basta con ejecutar dentro de XCode la aplicación con destino **iPad 6.1 Simulator**, creándose en este momento el directorio **Applications**.
2) El directorio **Applications** existe pero tiene mas de una carpeta y no sabemos cual es la de la aplicación. Aquí podemos usar el siguiente método: Salimos de XCode, borramos las carpetas existentes dentro de **Applications**, y volvemos a abrir XCode.

Una vez que tenemos localizada la ruta de la aplicación **PowerComicReader**, entramos en su carpeta y en el directorio **Documents**, siendo aquí donde deberíamos copiar los ficheros .cbr/.cbz que deseemos importar en la aplicación.

Una vez copiados los ficheros de comics en el directorio **Documents** de la aplicación basta con cerrar y volver a abrir la aplicación para que se importen o pulsar el botón de refrescar en la aplicación. Nota: no existe indicación de que la importación se están importando comics, habitualmente puede tardar 10-15 segundos en completar la importación para 7 u 8 comics. Una vez importados se actualiza el interfaz mostrándonos los comics importados.

Si deseamos cambiar el idioma del dispositivo en el simulador o en un iPad real podemos hacerlo desde la aplicación **Ajustes/Settings** (según el idioma de nuestro dispositivo) en:
Ajustes->General->Internacional->Idioma (en un iPad/Simulador en Español)
Settings->General->International->Language (en un iPad/Simulador en Ingles)

Para cambiar los ajustes de la aplicación, también debemos hacerlo desde la aplicación de **Ajustes/Settings**, para ello en la lista de aplicaciones buscaremos la aplicación **PowerComicReader**, desde donde podremos ajustar el estilo, nivel de zoom y si este se aplica con la doble pulsación o no. Para que los cambios tengan efecto podemos cerrar y reabrir la aplicación o pulsar en esta el botón refrescar.

5.7 Notas sobre como testear la aplicación en un dispositivo iPad físico

Para probar la aplicación en un dispositivo físico tipo iPad, en primer lugar deberemos obtener un certificado de desarrollo (lo que requiere una cuenta de desarrollador activa en developer.apple.com y la solicitud del certificado) que deberemos insertar en nuestro llavero de OSX y vincularlo a nuestro dispositivo de desarrollo, y modificar el nombre de la aplicación para que se corresponda con el solicitado, posteriormente seleccionaremos el iPad en XCode y ejecutando la aplicación para transferirla al dispositivo iPad.

Una vez que tengamos la aplicación instalada en el dispositivo iPad, podremos importar los comics mediante iTunes, para ello deberemos abrir en iTunes el dispositivo iPad conectado, y seleccionar el apartado **Aplic.**

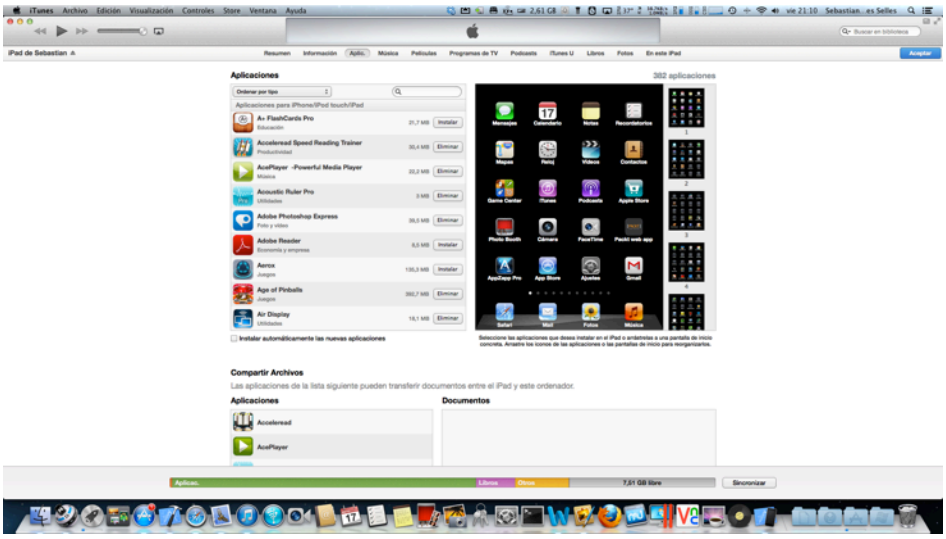


Figura 25. Búsqueda aplicación PowerComicReader para transferencia iTunes

Una vez seleccionado dicho apartado nos desplazaremos al apartado **Compartir Archivos** y **Aplicaciones**, buscando en la lista la aplicación **PowerComicReader**, una vez seleccionada la aplicación pulsando el botón **Añadir** podemos importar los comics en el directorio **Documents** de la aplicación. Una vez importados, estos se incorporaran a la aplicación bien en el arranque de esta o mediante la pulsación del botón refrescar de la aplicación.

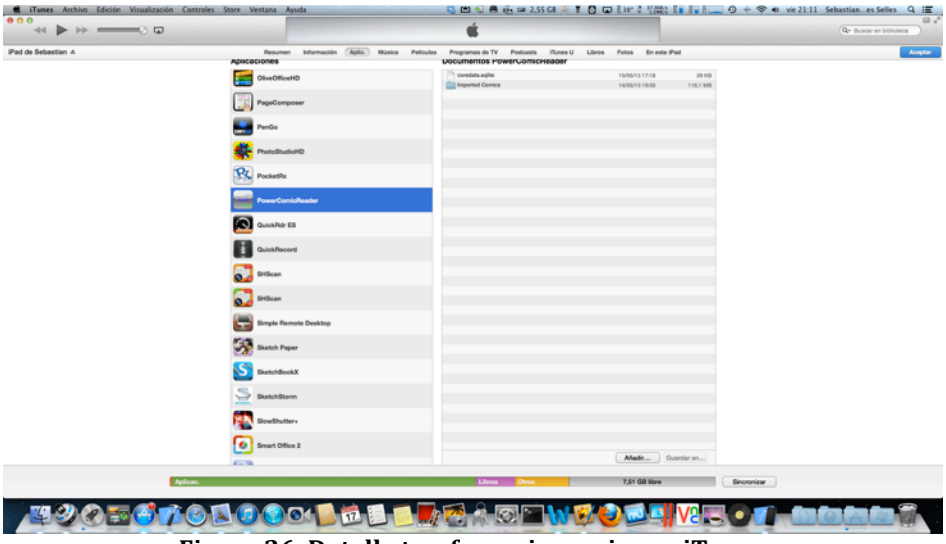


Figura 26. Detalle trasferencia comics en iTunes

5.8 Limitaciones de la aplicación en la importación de comics y tratamiento de errores en la importación de comics

En caso de errores en la importación debido a problemas en el formato interno del fichero, que este vacío, o simplemente la extensión del fichero importado sea errónea, el fichero a importar es ignorado de forma silenciosa, eliminándose el fichero causante del problema, sin notificación al usuario, pues es difícil informar al usuario de la causa del error, y continuándose la importación del siguiente fichero pendiente de importar. Así pues, si el comic tuviese una extensión errónea, es decir, si transferimos un comic mediante iTunes con extensión .cbr y el sistema de compresión es zip, no se importara, eliminándose de la carpeta **Documents** el comic erróneo. Esto mismo ocurre en el caso de intentar importar un comic en formato .cbz que en realidad debería ser un .cbr al estar comprimido mediante el compresor rar.

Se ha optado por esta solución, pues al tratarse con formatos libres como los .cbr y .cbz, la casuística de posibles problemas es muy alta y variada, por lo que debe estudiarse y ponderarse las distintas opciones y su tratamiento (como cambiar el descompresor de zip a rar si la descompresión zip fallase, etc.), por lo que actualmente se hace uso de las características de Objective-C que permiten obviar los mensajes a NULL sin provocar excepciones, y en caso de error se aborta la importación del comic en curso, y se continua con el siguiente.

Por lo que la mejora del tratamiento de errores de este tipo se deja pendiente para una futura versión.

6. Conclusiones

A lo largo del presente proyecto se ha implementado con éxito una aplicación para iPad cuya finalidad es la lectura de comics en formatos .cbr y .cbz.

El modelo de la arquitectura elegido para su implementación ha sido el paradigma MVC, que ha aportado grandes ventajas a nivel de organización, mantenimiento y reusabilidad.

Se han afrontado diversos problemas a lo largo de la implementación debiéndose integrar diversas librerías y frameworks de terceros para conseguir los objetivos marcados por el proyecto, estos problemas han provocado no solo el aprendizaje sino la búsqueda de recursos de calidad como paginas web, foros, y demás fuentes de información para resolverlos. Estos recursos son y serán una buena referencia en el futuro para acometer nuevos proyectos de forma exitosa, fijando un camino para la mejora de conocimientos y destrezas.

Desde el punto de vista personal la experiencia ha sido enriquecedora, permitiendo tomar contacto por primera vez tanto con el desarrollo en entornos de Apple como en el paradigma MVC, aportando unos conocimientos tecnológicos de amplia demanda actual en un campo en continuo desarrollo como son las aplicaciones móviles.

7. Glosario

La siguiente tabla contiene términos, abreviaturas y acrónimos necesarios para la correcta comprensión del documento.

Ajustes/Settings	Aplicación iOS que nos permite realizar los ajustes de las aplicaciones instaladas en nuestro dispositivo.
Arrastrar y Soltar	Gesto que consiste en presionar sobre un objeto en pantalla, permitiéndonos sin dejar de presionar arrastrar el objeto a una nueva posición, el gesto finaliza cuando dejamos de presionar sobre el objeto tomando dicho objeto la ultima posición donde finalizo el arrastre.
AutoLayout	Nueva característica de iOS 6 que permite definir las relaciones de distancias de los objetos en una vista facilitando la adaptación en caso de rotación y dando soporte sencillo a la nueva resolución del iPhone 5.
Control Transaccional	Agrupación de acciones sobre un manejador de gestor de persistencia que nos faculta para agrupar una o varias operaciones como un conjunto, de modo que podamos validar dicho conjunto o rechazarlo de forma simultanea. Una transacción únicamente tiene éxito (validación) si todas las operaciones que la componen lo tienen. Si una operación falla se aborta toda la transacción en su conjunto.
Contexto de Objetos Manejados	Objeto creado y manejado por Core Data, de modo que se aplican sobre este las reglas necesarias para hacerlo persistente con la mínima intervención por parte del programado.
Core Data	Framework de persistencia de iOS que nos permite acceder con una orientación a objetos a diversos almacenes de persistencia como son sqllite3, abstrayéndonos de tener que lidiar con la interfaz C/C++ de este desde nuestra aplicación Objective-C.
Delegado	Los delegados o delegación es un patrón donde un objeto actúa por cuenta de o en coordinación con otro objeto. El objeto delegante mantiene una referencia al otro objeto (el delegado) y en el momento apropiado le envía un mensaje. El mensaje informa al delegado de un evento que el objeto delegante va a manejar o ya manejó. El delegado debe responder al mensaje actualizando la apariencia, su estado u otros objetos de la aplicación, y en algunos casos puede regresar un valor que afecta como el próximo evento es manejado. El valor principal de la delegación es permitir la simple personalización del comportamiento de ciertos objetos en un objeto central.
Drag and Drop	Sinónimo de Arrastrar y Soltar.
Framework	Sinónimo de librería en nomenclatura iOS.
GIT	Software para el control de versiones.
iTunes	Aplicación de Apple que nos permite gestionar la biblioteca de documentos y aplicaciones de nuestros dispositivos iOS. Permitiendo la transferencia de documentos a nuestro dispositivo.
MVC	Modelo-Vista-Controlador
Panning	Gesto que se da en modo zoom o cuando estamos visualizando algún contenido cuyo tamaño es superior al tamaño de nuestra ventana y nos permite desplazarnos por esta pulsando sobre la pantalla y moviendo el dedo en la dirección en la que deseamos se produzca el desplazamiento.
Persistencia	Almacenamiento de objetos en una memoria persistente de modo que la información quede almacenada permanentemente para su uso futuro tras

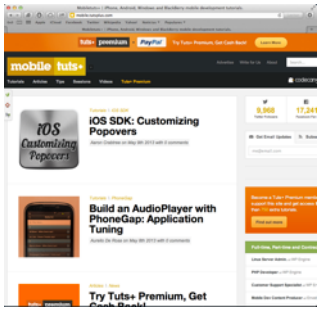
	el cierre de la aplicación o apagado de dispositivo.
Pitch	Gesto "pellizco", es un gesto multi táctil con dos dedos sobre la pantalla, si ambos dedos se acerca se aleja la imagen, mientras que si se alejan los dedos se aplica mayor nivel de zoom y se acerca la imagen.
Protocolo	Son métodos que pueden ser implementados por cualquier clase y permiten declarar métodos que se espera que otras clases implementen, permitiendo comunicar a la clase que implemente el protocolo eventos sucedidos en los objetos la clase que definió el protocolo.
Retina	Descripción comercial usada por Apple para sus dispositivos de alta resolución, duplicando la resolución correspondiente a los dispositivos en versión no retina, supuestamente la densidad por pulgada de dichas resoluciones es tan alta que la retina de ojo humano es incapaz de distinguir los puntos que componen la imagen (de aquí su nombre).
Segue	Vinculo que se crea entre dos vistas de un Storyboard que permite realizar una transacción desde una vista a otra mediante el encadenamiento de MVCs.
Storyboard	Organización de las Vistas de la aplicación en XCode 4, permitiéndose crear relaciones de modo grafico entre vistas (segues) y tener una visión mas general de las relaciones existentes entre estas de un vistazo.
Swipe	Gesto que consiste en pulsar la pantalla y realizar un desplazamiento con el dedo sin dejar de pulsar en una dirección, en función de dicha dirección podremos realizar una acción u otra. Por ejemplo un desplazamiento izquierda-derecha suele interpretarse en aplicaciones de lectura como pagina anterior.
Tap	Gesto de pulsación corta sobre un objeto/punto en la pantalla mediante un dedo.
XCode	Entorno de desarrollo de aplicaciones de Apple para MAC OSX/iOS, se requiere una maquina con MAC OSX para su ejecución.

8. Fuentes de información

Apple iOS Dev Center

Pagina oficial de Apple para desarrolladores iOS

developer.apple.com/devcenter/ios/index.action



Mobile Tutsplus
Pagina de tutoriales de desarrollo de aplicaciones móviles

mobile.tutsplus.com

RayWenderlich
Pagina de tutoriales iOS

www.raywenderlich.com

Curso de la universidad de Stanford: "Coding Together: Developing Apps for iPhone and iPad (Winter 2013)"

Impartido por Paul Hegarty a través de iTunes U

Curso de RayWenderlich: The iOS Apprentice Series
Curso introductorio y general de iOS adquirido en www.raywenderlich.com/store

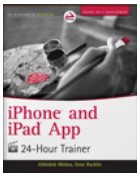
Cursos en forma de tutoriales de RayWenderlich: iOS 5 e iOS 6 By Tutorials
Cursos específicos de novedades con buenos ejemplos de iOS5 e iOS6 adquirido en www.raywenderlich.com/store

Stackoverflow
La mejor pagina de soporte para programación en entornos Apple

stackoverflow.com

Comic Book Plus
Pagina web donde descargar comics de dominio publico libres de copyright para pruebas

comicbookplus.com

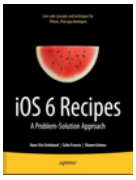


Libro: "iPhone and iPad App 24-Hour Trainer"

Varios autores

Libro: "Decoding the iOS 6 SDK", tutoriales de novedades de iOS 6 de mobile.tutsplus.com

Varios autores

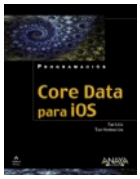


Libro: "iOS 6 Recipes"

Varios autores

Libro: "Programación con Objective-C"

Autor: STEPHEN G. KOCHAN



Libro: "Programación Core Data para iOS"

Autores: TIM ISTED, TOM HARRINGTON