Departamento de Informática

PROYECTO FIN DE CARRERA

Aplicación de visualización de archivos Passbook para Android

Alumno: Alicia Rivas Pérez

Tutor: José María Álvarez Rodríguez

Leganés, Diciembre 2015

Agradecimientos

A mis padres, José Antonio y Elu, por darme una educación excepcional dentro de sus posibilidades y por ser pacientes, porque sé lo importante que es para ellos el que "ponga la guinda final" a la carrera.

A mis hermanas, Emi y Silvia, porque sé que en ellas tengo a unas grandes compañeras de viaje y por su confianza en mí haga lo que haga.

A Alicia, por ser un grandísimo apoyo, por las risas, por la rutina,…

A Alis y Laura, por ser mis pepitos grillos y ayudarme en todo lo que han podido y más, por ponerme en mi sitio cuando tocaba y decirme las cosas como son. ¡Gracias infinitas!

A mí tutor José María, por responderme rápidamente al email en el que le pedía ser mi tutor y añadir raudamente el proyecto al tablón para que pudiera empezar cuanto antes.

Y por último, gracias a todos los que durante estos años han estado ahí pacientemente y que han seguido creyendo en mí.

Resumen

En un mundo en el que la presencia de la tecnología en la vida cotidiana está cada vez más extendido, sustituir los billetes de transporte, entradas de cine, eventos, etc. por su homólogo en el universo digital parece un paso lógico. El formato **Passbook**, diseñado por Apple, surge precisamente para llenar ese vacío y es una cartera virtual donde poder guardar toda esa información y mucho más.

La finalidad de la aplicación que se describe en este documento, es la creación de visualizador de ese tipo de archivos de una forma nativa y con una apariencia que resulte intuitiva a los usuarios de la plataforma móvil Android ya que las aplicaciones que se encuentran actualmente en Google Play (tienda de aplicaciones de estos dispositivos), tienen una apariencia más propia de móviles con sistema operativo iOS o son aplicaciones híbridas.

La tecnología empleada se basa en Java, Android SDK y Gradle más otras librerías de soporte proporcionadas por Google para poder portar ciertas funcionalidades a versiones más antiguas de su plataforma, la arquitectura se realizará siguiendo los principios de Clean Code y también se utilizarán librerías de generación de imágenes QR, carga de las mismas, etc.

Abstract

In a world in which the presence of technology in everyday life is increasing, replacing transport cards, movie and theatre tickets, etc. by it's counterpart in the digital world looks like a reasonable step. The Passbook format, designed by Apple, was created precisely to fill that void and it is a digital wallet where one can store all that information and more.

The purpose of the application described in this document, is the creation of a reader and renderer of this kind of files in a native way and with a look and feel that feels intuitive to the users of the Android mobile platform since the applications that can be currently found in Google Play (app store for these devices), have an iOS look and feel or are hybrid applications.

The technology that will be used is based in Java, Android SDK and Gradle, plus other support libraries provided by Google to be able to port certain functionalities to older versions of their platform, the architecture will follow the principles of Clean Code and libraries for QR generation, image loading, etc will also be used.

Índice general

1. Introducción y objetivos 1
1.1 Introducción 1
1.2 Objetivos 2
1.3 Estructura de la memoria 3

2. Estado del arte 6
2.1. Sistemas operativos móviles 6
2.2. Android 11
2.3. Passbook 12
2.3.1. Estructura de un pase 13
2.3.2. Tipos de pases 15
2.4. Lectores de archivos Passbook 22
2.4.1. Wallet (iOS) 23
2.4.2. PassWallet 24
2.4.3. Pass2U 25
2.4.4. PassAndroid 26

3. Análisis 28
3.1. Requisitos de usuario 28
3.1.1. Requisitos de capacidad 29
3.1.2. Requisitos de restricción 33
3.2. Casos de uso 34
3.3. Requisitos software 41
3.3.1. Requisitos funcionales 41
3.3.2. Requisitos no funcionales 48
3.3.3. Requisitos de entorno 48
3.4. Matrices de trazabilidad 49

4. Diseño 52
4.1. Arquitectura del sistema 52
4.2. Aplicando Clean Architecture 56
5. Implementación 64
5.1. Ficha técnica de recursos 64
5.1.1. Gson 64
5.1.2. Butter Knife 66
5.1.3. Glide 67
5.1.4. Dagger 2 67
5.1.5. ZXing 70
5.2. Implementación 70
5.2.1. Librería PkReader 70
5.2.2. Domain 72
5.2.3. Data 73
5.2.4. App 76
5.3. Pruebas 78
5.4. Interfaz 80
6. Gestión del proyecto 82
6.1. Planificación 82
6.2. Presupuesto 83
6.2.1. Recursos humanos 83
6.2.2. Material 84
6.2.3. Gastos indirectos 85
6.2.4. Coste total 85
7. Entorno de desarrollo 87
7.1. Android Studio 88
7.2. JDK 88
7.3. Gradle 89
7.4. JUnit 4 89
7.5. Espresso 89
7.6. Robolectric 90
7.7. Git 90
8. Conclusiones 93

9. Trabajos futuros 96

10. Definiciones, acrónimos y abreviaturas 99

11. Referencias 102

Índice de figuras

Figura 1: Distribución sistema operativo Android 8
Figura 2: Distribución sistema operativo iOS 9
Figura 3: Número de aplicaciones en las principales tiendas móviles 9
Figura 4: Cuota uso sistemas operativos en supercomputadoras 11
Figura 5: Componentes tecnología Passbook 13
Figura 6: Estructura de directorios de un pase de ejemplo 14
Figura 7: Disposición de una tarjeta de embarque 17
Figura 8: Disposición de un cupón 17
Figura 9: Disposición de una entrada a evento 18
Figura 10: Disposición de un pase genérico 18
Figura 11: Disposición de una tarjeta de fidelización 19
Figura 12: Pase tipo boardingPass con campos sencillos 20
Figura 13: Ejemplo de campo con formato de fecha 21
Figura 14: Contenido parcial de un pase con barcode 21
Figura 15: QR 21
Figura 16: PDF417 21
Figura 17: Aztec 21
Figura 18: Code 128 21
Figura 19: Campo de un pase de ejemplo en el que el campo label está localizado 22
Figura 20: Fichero localización en español dentro de directorio es.lproj 22
Figura 21: Fichero localización en inglés dentro de directorio en.lproj 22
Figura 22: Icono aplicación Wallet 23
Figura 23: Wallet - Listado de pases y detalle 23
Figura 24: Aplicación PassWallet 24
Figura 25: Capturas aplicación PassWallet 24
Figura 26: Aplicación Pass2U 25

Figura 27: Capturas aplicación Pass2U 25
Figura 28: Aplicación PassAndroid 26
Figura 29: Capturas aplicación PassAndroid 26
Figura 30: Regla de dependencia en Clean Code 53
Figura 31: Capas en Clean Architecture 54
Figura 32: Capa de Presentación 54
Figura 33: Capa de Dominio 55
Figura 34: Capa de Datos 56
Figura 35: Dependencia entre módulos del proyecto 57
Figura 36: Modelo de datos librería PkReader 58
Figura 37: Modelo de datos Domain 59
Figura 38: Estructura de clases relacionadas con la inyección de dependencias 60
Figura 39: Clases dentro del paquete "component" relacionadas con la inyección de dependencias 60
Figura 40: Clases dentro del paquete "module" relacionadas con la inyección de dependencias 61
Figura 41: Modelo de datos App 62
Figura 42: Diseño de ejemplo para Butter Knife 66
Figura 43: View binding sin Butter Knife 66
Figura 44: View binding con Butter Knife 66
Figura 45: Dagger - Componente de ejemplo 69
Figura 46: Dagger - Módulo de ejemplo 69
Figura 47: Dagger – Ejemplo de creación de componente 69
Figura 48: Estructura librería PkReader 72
Figura 49: Estructura capa Domain 73
Figura 50: Extracto implementación LocalPassDataSource 74
Figura 51: Extracto implementación PassRepository 75
Figura 52: Estructura capa Data 75
Figura 53: Inyección dependencias PassKitApplication 76
Figura 54: Inyección dependencias MainActivity 77
Figura 55: MainActivityComponent 77
Figura 56: Diagrama secuencia de carga de un pase 78
Figura 57: Capturas aplicación lector de ficheros Passbook 80
Figura 57: Planificación del proyecto 83

Índice de tablas

Tabla 1: Previsión de Smartphones por sistema operativo, envíos, cuota de mercado, crecimiento en 2015 (unidades en millones) 7
Tabla 2: Previsión de Smartphones por sistema operativo, envíos, cuota de mercado, crecimiento y TCAC (unidades en millones) 7
Tabla 3: Fragmentación en Android 8
Tabla 4: Resumen comparativa plataformas 10
Tabla 5: Archivos disponibles en la raíz del paquete del pase 15
Tabla 6: Tipo de pases, imágenes y disposición 16
Tabla 7: CAP-1 29
Tabla 8: CAP-2 29
Tabla 9: CAP-3 30
Tabla 10: CAP-4 30
Tabla 11: CAP-5 30
Tabla 12: CAP-6 30
Tabla 13: CAP-7 31
Tabla 14: CAP-8 31
Tabla 15: CAP-9 31
Tabla 16: CAP-10 31
Tabla 17: CAP-11 32
Tabla 18: CAP-12 32
Tabla 19: CAP-13 32
Tabla 20: CAP-14 32
Tabla 21: CAP-15 33
Tabla 22: CAP-16 33
Tabla 23: RES-1 33
Tabla 24: RES-2 34

Tabla 25: CU-1 34
Tabla 26: CU-2 35
Tabla 27: CU-3 35
Tabla 28: CU-4 35
Tabla 29: CU-5 36
Tabla 30: CU-6 36
Tabla 31: CU-7 37
Tabla 32: CU-8 37
Tabla 33: CU-9 38
Tabla 34: CU-10 38
Tabla 35: CU-11 39
Tabla 36: CU-12 39
Tabla 37: CU-13 40
Tabla 38: CU-14 40
Tabla 39: CU-15 41
Tabla 40: RF-1 42
Tabla 41: RF-2 42
Tabla 42: RF-3 43
Tabla 43: RF-4 43
Tabla 44: RF-5 43
Tabla 45: RF-6 44
Tabla 46: RF-7 44
Tabla 47: RF-8 44
Tabla 48: RF-9 45
Tabla 49: RF-10 45
Tabla 50: RF-11 45
Tabla 51: RF-12 46
Tabla 52: RF-13 46
Tabla 53: RF-14 47
Tabla 54: RF-15 47
Tabla 55: RF-16 47
Tabla 56: RNF-1 48
Tabla 57: RE-1 48
Tabla 58: RE-2 48

Tabla 59: Matriz de trazabilidad requisitos de usuario y requisitos software 49
Tabla 60: Matriz de trazabilidad requisitos de usuario y casos de uso 50
Tabla 61: Matriz de trazabilidad casos de uso y requisitos software 50
Tabla 62: Ejemplo serialización Gson 65
Tabla 63: Ejemplo deserialización Gson 65
Tabla 64: Carga de imagen usando Glide 67
Tabla 65: Formatos soportados por librería ZXing 70
Tabla 66: Validación requisitos en diferentes terminales 79
Tabla 67: Costes personal 84
Tabla 68: Coste de material 84
Tabla 69: Gastos indirectos 85
Tabla 70: Coste total 85
Tabla 71: Definición de términos del documento 99
Tabla 72: Acrónimos del documento 100

Capítulo 1
Introducción y objetivos

1.1 Introducción

Hoy en día, la idea del teléfono móvil como mero instrumento para realizar llamadas o enviar mensajes cortos ha quedado obsoleta. Cada vez hay más modelos diferentes de móviles inteligentes y surgen nuevas formas de realizar tareas, para las que antes necesitarías un ordenador, que puedes hacer desde prácticamente cualquier sitio disponiendo de uno de estos dispositivos.

Un teléfono inteligente o Smartphone es un dispositivo capaz de ejecutar aplicaciones con diversas funcionalidades y con capacidad de conectarse a la red manteniendo a los usuarios conectados permanentemente si así lo desean. Cuentan con una pantalla táctil que va creciendo en tamaño con los últimos modelos y con diferentes sensores, entre ellos GPS, acelerómetro y giroscopio.

Diferentes estudios, como el realizado por la empresa IDC (International Data Corporation) demuestran que el crecimiento en ventas de estos dispositivos parece imparable ya que sólo en el año 2015 la venta de teléfonos inteligentes tiene un crecimiento del 10.4% frente al año anterior con un volumen de ventas previsto a fin de año de unos 1.400 millones de unidades que, aunque menor de lo esperado, sigue siendo en positivo.

Una de las cosas que ha evolucionado con el tiempo es la gestión de tarjetas de embarque, entradas a eventos, tarjetas de fidelización, etc. gracias a Passbook, un formato definido y creado por Apple, que permite a los usuarios de teléfonos inteligentes disponer de una versión

digital de dichas tarjetas, entradas, etc. con validez legal, facilitando así el proceso. De hecho, más de 400[1] aplicaciones disponibles la App Store, son compatibles con Passbook a día de hoy.

Debido a que Passbook fue creado por Apple, las aplicaciones que surgieron alrededor referentes a teléfonos móviles eran de iOS. Sin embargo, nos encontramos que en el mercado más del 81% de los terminales tienen el sistema Android. Entonces, ¿qué ocurre con Passbook para estos terminales?

Para Android, existen aplicaciones, pero no implementan el formato de forma nativa. En su gran mayoría son aplicaciones que tienen una estética más propia de iOS. Por tanto, parece interesante el estudio de viabilidad de cómo realizar una aplicación nativa Android que gestione billetes y tickets con formato Passbook, y a partir de los resultados del estudio, tratar de diseñar e implementar una aplicación.

1.2 Objetivos

El principal objetivo de la aplicación de visualización de archivos Passbook, es la creación de una utilidad para Android que lea y muestre ese tipo de archivos de una forma nativa y con una apariencia que resulte intuitiva a los usuarios de la plataforma móvil ya que las aplicaciones que se encuentran actualmente en Google Play (tienda de aplicaciones de estos dispositivos), tienen una apariencia más propia de móviles con sistema operativo iOS o son aplicaciones híbridas. Este objetivo se divide en los siguientes aspectos.

- Estudio del formato Passbook.
- Estudio del estado del arte de aplicaciones tanto en iOS como en Android que implementen o usen dicho formato.
- Diseño de una aplicación que permita:
    - Correcta visualización de archivos Passbook siguiendo los patrones establecidos por Apple para su lectura.
    - Persistir dichos archivos en la memoria del teléfono para que el usuario pueda acceder a ellos siempre que desee evitando así que tenga que añadirlos una y otra vez siempre que quiera visualizarlos.
    - Gestión de dichos archivos permitiendo al usuario eliminarlos o compartirlos con otra persona si así lo deseara.

[1] Valor estimado obtenido de la búsqueda de la palabra “Passbook” en www.appshopper.com.

    - Proporcionar una interfaz de usuario intuitiva, siguiendo los patrones de diseño de Android.
- Implementación de la aplicación, usando las herramientas proporcionadas por la plataforma (Android, Gradle, Git, …).
- Usar los principios de Clean Code como base para la arquitectura del proyecto.
- Banco exhaustivo de pruebas, tanto a nivel software, como a nivel usuario.

1.3 Estructura de la memoria

El contenido de la memoria está organizado según el orden cronológico de las distintas fases que componen un proyecto software. A continuación se muestra el objetivo de cada uno de los apartados de lo que consta el presente documento.

1. **Introducción y objetivos:** es el apartado actual y, además de describir la estructura de la memoria, es donde se explican las motivaciones que han derivado en la realización de este proyecto. También se muestran los objetivos que se pretenden alcanzar con el producto una vez finalizado.

2. **Estado del arte:** realiza una comparativa de las distintas opciones que existen en el mercado con una funcionalidad equivalente o similar a la aplicación que se pretende desarrollar. También se evalúan las distintas plataformas y se ahonda en lo que consiste el formato Passbook.

3. **Análisis:** recoge las funcionalidades que deberá tener la aplicación final a un alto nivel y desde el punto de vista del usuario así como los distintos casos de uso de los que se compone.

4. **Diseño:** en él se recoge el diseño arquitectónico en el que se basará la posterior implementación.

5. **Implementación:** describe el proceso de implementación y aplicación de lo descrito en el apartado de Diseño.

6. **Gestión del proyecto:** este apartado tiene como objetivo recoger todos los aspectos del proyecto relacionados con su administración, lo que incluye es establecimiento y descripción

de las diferentes fases de las que constará el proyecto, así como la planificación de las mismas, los recursos que serán utilizados y una estimación de presupuesto.

7. **Entorno de desarrollo:** será aquí donde se describirán las diferentes herramientas necesarias para realizar la aplicación.

8. **Conclusiones:** descripción de las conclusiones extraídas tras la finalización del proyecto.

9. **Trabajos futuros:** propuestas de ampliaciones y mejoras en la aplicación.

10. **Definición de acrónimos y abreviaturas:** descripción del significado de los diferentes acrónimos y abreviaturas utilizados a lo largo del proyecto.

11. **Referencias:** recoge las distintas referencias utilizadas a lo largo del proyecto.

Capítulo 2
Estado del arte

En este capítulo se introducirán los conceptos de los que se ha hablado en la introducción, para proporcionar el contexto actual sobre el formato Passbook, los sistemas operativos para terminales móviles existentes en el mercado, y las aplicaciones móviles concretas para terminales Android que usan Passbook.

2.1. Sistemas operativos móviles

Dentro del mercado de teléfonos inteligentes se distinguen tres sistemas operativos que destacan por encima de los demás, éstos son: Android, iOS y Windows Phone. Existen otras alternativas como Symbian, Bada, BlackBerry, etc. pero sus números de ventas son muy inferiores a los anteriores por lo que se han englobado en el grupo *Otros*.

A la hora de determinar la plataforma sobre la cual realizar el proyecto, hay que tener en cuenta varios aspectos. Estos factores son: la cuota de mercado con respecto a sus competidores; su crecimiento actual y estimado; dificultades a la hora de desarrollar aplicaciones software para esos terminales así como las características propias de la plataforma.

En lo que respecta al crecimiento actual de las diferentes plataformas, tal y como indica el estudio anteriormente mencionado y realizado por la empresa IDC. En el año 2015 aunque el crecimiento en ventas de un 10,4% ha disminuido considerablemente frente al año anterior (27,5%), la tendencia sigue siendo positiva.

Sector	Volumen envíos 2015*	Cuota mercado 2015*	Crecimiento Año a Año 2015*
Android	1.164,3	81,1%	9,9%
iOS	223,7	15,6%	16,1%
Windows Phone	36,9	2,6%	5,8%
Otras	11,5	0,8%	-15,5%
TOTAL	**1.436,5**	**100%**	**10,4%**

Tabla 1: Previsión de Smartphones por sistema operativo, envíos, cuota de mercado, crecimiento en 2015 (unidades en millones) [2]
** Datos estimados*

En cuanto a sistemas operativos, se puede observar la predominancia de Android sobre sus competidores con un 81,1%, seguido de iOS con un 15,6% y Windows Phone con un 2,6%. El informe indica también que dichos valores no van a variar mucho en los próximos años por lo que Android seguirá dominando como plataforma móvil más extendida, tal y como se puede ver en la siguiente tabla.

Sector	Volumen envíos 2019*	Cuota mercado 2019*	Crecimiento Año a Año 2019*	Tasa crecimiento anual compuesto 5 años
Android	1.541,9	81,1%	5.0%	7,8%
iOS	269,6	14,2%	3.3%	7,0%
Windows Phone	67,8	3,6%	12,8%	14,2%
Otras	23,0	0.8%	8,6%	11.0%
TOTAL	**1.436,5**	**100%**	**10,4%**	**7,9%**

Tabla 2: Previsión de Smartphones por sistema operativo, envíos, cuota de mercado, crecimiento y TCAC (unidades en millones) [2]
** Datos estimados*

Comprobamos que, aunque Microsoft/Windows Phone seguirá manteniendo una cuota de mercado marginal con respecto a Android e iOS, es el que mayor crecimiento estimado presenta para el año 2019.

En lo que se refiere a la dificultad de realizar aplicaciones software para las diferentes plataformas, cabe destacar que Android es un sistema abierto que se puede beneficiar de los aportes de la comunidad de desarrolladores mientras que tanto iOS como Windows Phone son

sistemas operativos privativos cuyo código fuente no está disponible para ser auditado por terceros.

Los lenguajes de programación utilizados para las diferentes plataformas son Java en el caso de Android; Objective-C será requerido para realizar aplicaciones para iOS mientras que Windows Phone soporta los lenguajes de programación C# y Visual Basic .NET.

Uno de los principales motivos por los que Android ocupa ese primer puesto como plataforma móvil más extendida es el hecho de que, al contrario que su principal competidor iOS, hay mucha variedad de terminales con características dispares que funcionan con este sistema operativo. Este factor combinado con el hecho de que, en la mayoría de los casos, muchos de esos dispositivos son distribuidos por las operadoras de telefonía en lugar de por Google directamente hace que exista mucha fragmentación en Android ya que el control sobre las actualizaciones no recae sobre el creador del software sino sobre los fabricantes y operadoras de telecomunicaciones.

Versión	**Codename**	**API**	**Distribución**
2.2	Froyo	8	0,2%
2.3.3 – 2.3.7	Gingerbread	10	3,8%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	3,3%
4.1.x	Jelly Bean	16	11,0%
4.2.x		17	13,9%
4.3		18	4,1%
4.4	KitKat	19	37,8%
5.0	Lollipop	21	15,5%
5.1		22	10,1%
6.0	Marshmallow	23	0,3%

Tabla 3: Fragmentación en Android [1]

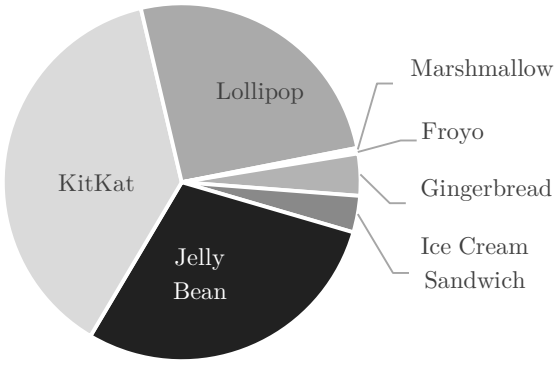

Figura 1: Distribución sistema operativo Android [1]

En esta gráfica se hace más que evidente la gran fragmentación que existe en Android. El ultimo sistema operativo lanzado en octubre del año 2015 llamado Marshmallow, ha tenido un nivel de penetración de sólo el 0,3% y muy probablemente un alto porcentaje de esos dispositivos sean los pertenecientes a la gama Nexus, unos terminales que el gigante americano ha fabricado con la intención de minimizar la fragmentación, ofreciendo a los usuarios móviles que serán los primeros en recibir las actualizaciones ya que será Google el que se encargará de distribuirlas y no habrá ningún otro intermediario.

Esta fragmentación contrasta con la rápida penetración de las diferentes versiones de iOS, tal y como se puede observar en la gráfica de la derecha, debido a que su sistema de distribución es mucho más directo y al hecho de que tienen que dar soporte a menos terminales, llamados iPhone, lo que implica que la adaptación a las diferentes características hardware de los dispositivos es mucho más veloz.

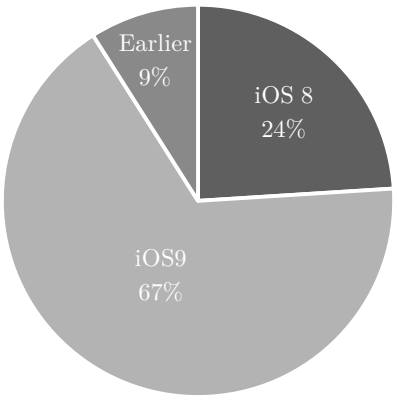

Figura 2: Distribución sistema operativo iOS [3]

En lo que respecta al número de aplicaciones móviles en las diferentes tiendas disponibles en cada plataforma, Android con 1.6 millones ha conseguido superar a su gran competidor iOS que tiene la nada despreciable cifra de 1.5 millones de aplicaciones. Cabe destacar también que algunos dispositivos vendidos por la tienda Amazon App Store están basados en el sistema operativo Android.

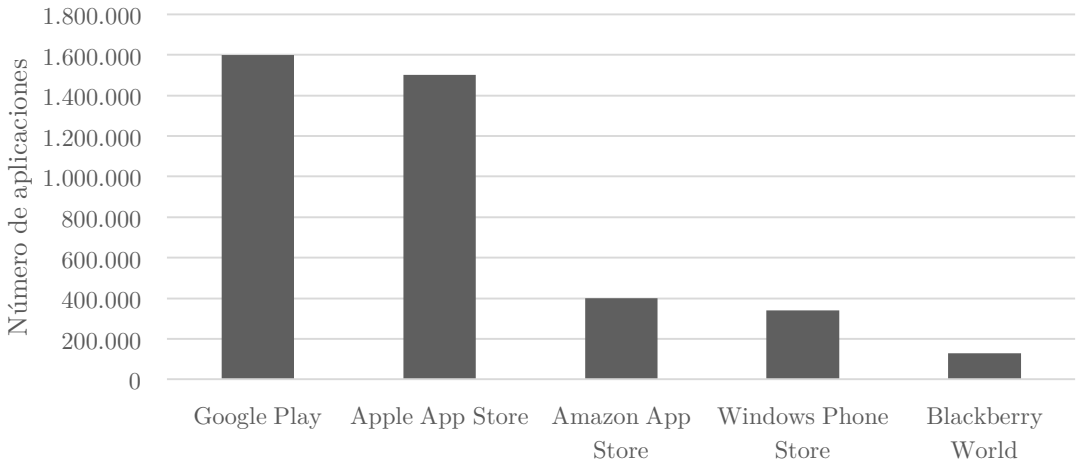

Figura 3: Número de aplicaciones en las principales tiendas móviles [4]

Otro detalle que merece la pena destacar es el coste para poder publicar una aplicación en las diferentes tiendas. Los desarrolladores iOS, tienen que pagar 99 dólares americanos (aprox. 93€) para poder hacerlo; en Windows Phone hay dos tipos de cuentas, una cuenta individual, con funcionalidades restringidas, que cuesta unos 19 dólares americanos (aprox. 17€), y una cuenta de empresa que, al igual que la cuenta de desarrollador de Apple, supone $99. Los precios de ambas cuentas varían en función del país o región; y Android, cuyo coste es de $25 (aprox. 23,5€).

En el siguiente cuadro se resumen las valoraciones de las diferentes características tenidas en cuenta para seleccionar una plataforma en lugar de otra.

Característica	**Android**	**iOS**	**Windows Phone**
Uso	Alto	Medio	Bajo
Crecimiento	Medio	Medio	Medio
Número aplicaciones	Alto	Alto	Bajo
Facilidad desarrollo	Alto	Medio	Medio
Coste desarrollo	Bajo	Alto	Bajo/Alto
Compatibilidad	Alto	Alto	Bajo
Fragmentación	Alto	Bajo	Medio

Tabla 4: Resumen comparativa plataformas

Teniendo en cuenta todos los factores mencionados anteriormente y dando especial importancia al nivel de uso y crecimiento de la plataforma y la alta compatibilidad que posee, se puede observar que hay un claro vencedor, Android.

Android se posiciona por delante de sus competidores con una amplia ventaja en la cuota de mercado, además, gracias a que está presente en diferentes terminales con características hardware muy diferentes hace que pueda estar disponible tanto en teléfonos inteligentes de alta gama como de gama media y baja por lo que la probabilidad de que una aplicación desarrollada para esta plataforma pueda impactar en un gran número de usuarios es alta.

Por otro lado, hay que tener en cuenta que trabajamos con una plataforma con un gran nivel de fragmentación, aunque desde Google están tomando medidas para intentar solventarla además de proporcionar diferentes librerías de soporte que ayudan a los desarrolladores a portar funcionalidades presentes en las últimas versiones de su sistema operativo a versiones más antiguas, para que todos las personas poseedoras de uno de estos terminales disfruten de una experiencia de usuario única.

2.2. Android

Android es un sistema operativo basado en Linux, diseñado especialmente para dispositivos móviles con pantalla táctil y desarrollado inicialmente por Android Inc. Dicha empresa fue comprada en 2005 por Google, subsidiaria de Alphabet, una multinacional estadounidense especializada en productos y servicios relacionados con Internet, software, dispositivos electrónicos y otras tecnologías.

Linux ofrece robustez y fiabilidad al núcleo de este sistema operativo, de hecho, tal como se puede ver en la gráfica a continuación, esas características llevaron a Linux en junio del año 2014 a estar presente en el 97% de las supercomputadoras más potentes del mundo según datos ofrecidos por Top 500, una página que recoge una lista de dichos procesadores.

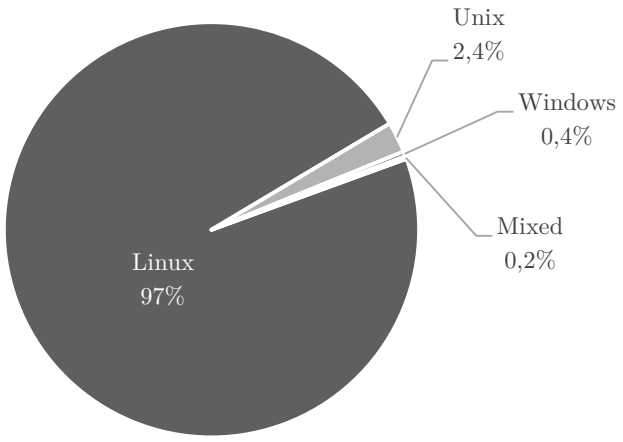

Figura 4: Cuota uso sistemas operativos en supercomputadoras [5]

En lo que respecta a las características propias de Android, el lenguaje utilizado para realizar aplicaciones es Java y C, este último a través de la JNI (Java Native Interface). El hecho de que se está trabajando sobre un núcleo Linux está totalmente abstraído de tal manera que cada aplicación dispone de una carpeta privada dedicada en memoria interna y al desarrollador se le proporcionan métodos para acceder a ella, pero nunca se accede directamente.

Los archivos ejecutables tienen la extensión .apk. Este fichero es un mero contenedor y en su interior se encuentran las clases compiladas, firmas y recursos necesarios para el correcto funcionamiento de la aplicación. Para instalar un archivo .apk, basta con copiar el archivo a la memoria del teléfono y abrirlo, el sistema se encargará del resto.

En el párrafo anterior se ha mencionado la presencia de firmas ya que un ejecutable de Android siempre va firmado. Existen dos tipos de firmas, de desarrollo y de producción. Por defecto, los binarios que se generan mientras se está desarrollando van firmados con una clave de desarrollo autogenerada, por lo que es transparente para el programador, pero si se quisiera publicar la aplicación en Google Play, habría que generar una firma de producción y firmar la aplicación con ella, sino, cuando se intente subir la aplicación al mercado de aplicaciones, el proceso fallará.

Otro aspecto a destacar son los permisos, debido a que un teléfono inteligente dispone de diferentes sensores, para definir cuáles de ellos se van a utilizar, hay que pedir los permisos adecuados. Existen permisos de todo tipo, para escribir y/o leer en memoria interna, para usar la cámara, tener acceso a Internet, acceder al GPS, etc. El usuario antes de instalar una aplicación podrá ver los permisos que ésta solicita y decidir si quiere seguir adelante o no. Además, en la última versión del sistema operativo, Marshmallow, el usuario podrá desactivar permisos uno a uno, dándole más control los mismos.

Por último, para desarrollar aplicaciones para esta plataforma basta con descargar el SDK de Android. Actualmente dicho SDK se integra perfectamente con Eclipse y con AndroidStudio, basado en IntelliJ, siendo este último el que más protagonismo está teniendo hace unos años ya que está siendo desarrollado y mejorado intensamente por Google.

2.3. Passbook

Passbook es una librería creada y diseñada por Apple que transforma las tarjetas de fidelización, de embarque, cupones, entradas a eventos, etc. en componentes llamados *pases* que son una representación digital de información que en otro caso sería impresa en pequeños trozos de papel o plástico. Los pases pueden contener imágenes y un código de barras, y se pueden actualizar mediante notificaciones *push*.

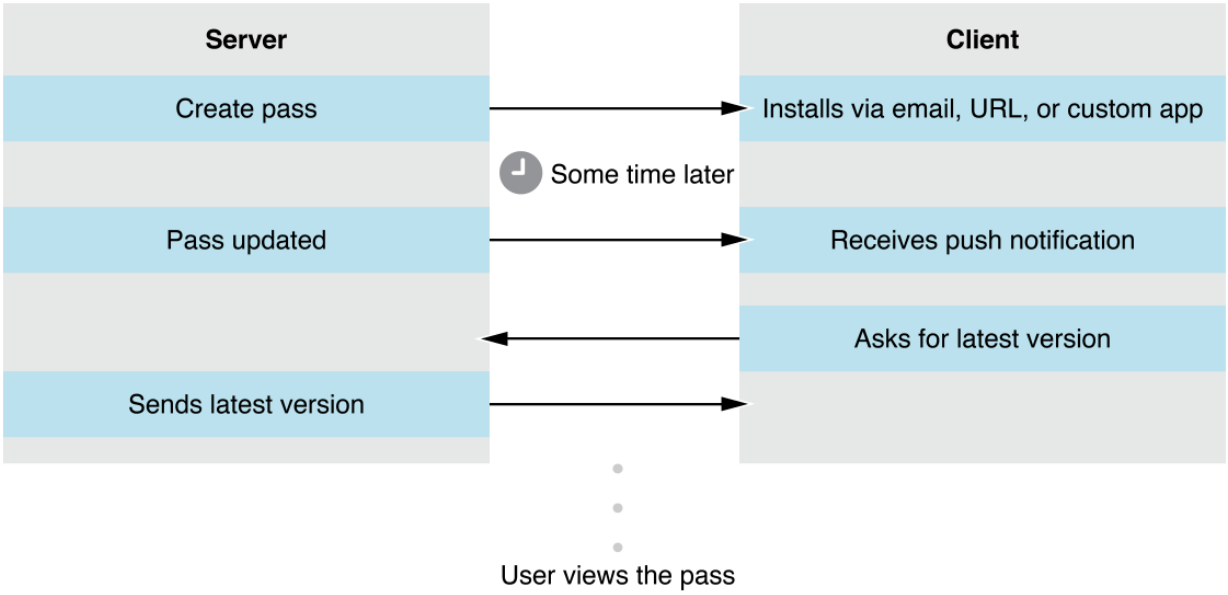

Figura 5: Componentes tecnología Passbook [6]

Esta tecnología consiste en tres componentes:

- Una estructura de paquete para crear los pases.
- Un API de servicios para actualizar los pases, implementado en un servidor propio.
- Un API escrita en Objective-C que usarán las aplicaciones MacOS e iOS para interactuar con la librería de pases del usuario.

Hay tres partes principales en el ciclo de vida de un pase: creación, gestión y canjeo. La aplicación nativa de iOS (Wallet), utilizando el API mencionado anteriormente, se encarga de manejar la parte central permitiendo a los usuarios ver y gestionar sus pases integrándose con la pantalla de bloqueo para mostrar diversas notificaciones relacionadas con ellos.

2.3.1. Estructura de un pase

Los archivos que componen un pase están contenidos dentro de un paquete. Un archivo clave dentro del mismo es el llamado *pass.json*, que es el que define el pase. El fichero JSON contiene información que identifica el pase además del texto que aparece en él junto con otra información. Adicionalmente, el paquete puede contener imágenes e información de localización.

A continuación, se muestra en detalle la estructura de directorios de un pase de ejemplo que contiene un icono, un logo y un thumbnail. El pase está localizado en inglés y en chino, contiene imágenes de alta (identificadas con el *@x2*) y baja calidad y el logo está localizado.

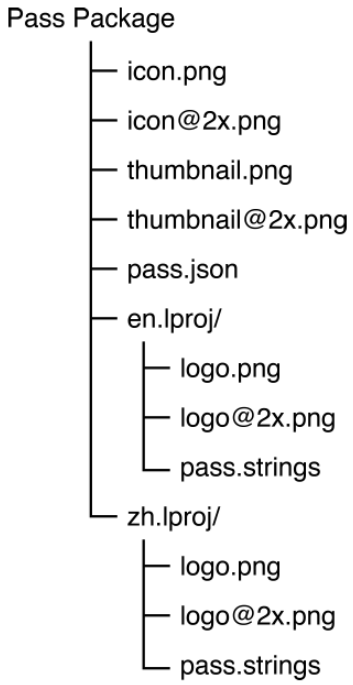

Figura 6: Estructura de directorios de un pase de ejemplo [7]

Por lo tanto, crear un pase consiste básicamente en escribir el archivo *pass.json* y diseñar las imágenes que se incluirán en él.

En la siguiente tabla se describen los diferentes tipos de ficheros que se pueden encontrar en la raíz del paquete:

`background.png`	Imagen que se mostrará como fondo en la parte frontal del pase.
`footer.png`	Imagen que se mostrará en la parte frontal del pase cerca del código de barras.
`icon.png`	El icono del pase. Este icono se mostrará en las notificaciones y en la pantalla de bloqueo.
`logo.png`	Imagen que se mostrará en la parte frontal del pase en la parte superior izquierda.
`manifest.json`	Diccionario JSON. Cada clave es la ruta a un archivo (relativo al nivel superior del paquete) y el valor de dicha clave es el hash SHA-1 para ese archivo. Todos los archivos en el paquete

	aparecen en este fichero, excepto el propio `manifest` y la firma (`signature`).
`pass.json`	Diccionario JSON que define el pase.
`signature`	Firma PKCS #7
`strip.png`	Imagen que se mostrará detrás de los campos principales en la parte frontal del pase.
`thumbnail.png`	Imagen adicional que se muestra en la parte frontal del pase.

Tabla 5: Archivos disponibles en la raíz del paquete del pase [7]

Un pase se identifica inequívocamente de otros por el valor de su identificador de tipo de pase (`passTypeIdentifier)` y su número de serie (`serialNumber`). El primero es una cadena de texto que define una clase o categoría de pases, que siempre comienza con `pass.` y que usa el estilo DNS inverso, por ejemplo, `pass.com.ejemplo.boarding-pass`. El valor de esta clave tiene que coincidir con el certificado utilizado para firmar el pase.

Además del identificador del tipo de pase y del número de serie, hay otras claves que todos los pases contienen:

- La versión del formato del fichero es especificada por la clave numérica `formatVersion`.
- El identificador de grupo de la organización que firmó el pase. Una serie de números y letras que Apple proporciona. Esta clave se llama `teamIdentifier`.
- El nombre de la organización (`organizationName`) que se mostrará en la pantalla de bloqueo y por las aplicaciones que así lo requieran.
- La clave `description` será usada por los servicios talk-to-speech de los dispositivos para que aquellas personas con discapacidades puedan saber el contenido del pase.

2.3.2. Tipos de pases

Dentro del diccionario *pass.json*, se encuentran también diferentes claves que ayudarán a determinar el tipo o estilo del pase. Estos estilos son fijos y forman parte del API y no se pueden ni añadir ni modificar.

El tipo de pase vendrá definido por el valor de la clave correspondiente en la raíz del archivo *pass.json*:

1. Las tarjetas de embarque usan la clave `boardingPass`. Este pase será apropiado para sistemas de transporte como pueden ser billetes de tren, avión, autobús y otros.
2. Los cupones usan la clave `coupon`. Se usará este pase para cupones, ofertas especiales y otros descuentos.
3. Las entradas a eventos usan la clave `eventTicket`. Este tipo de será el utilizado para pases que den acceso a algún evento como puede ser un concierto, una película de cine, una obra de teatro o evento deportivo.
4. Las tarjetas de fidelización usan la clave `storeCard`.
5. Los pases genéricos usan la clave `generic`. Se generará un pase de este tipo en el caso de que no encaje en ninguna de las otras categorías.

El estilo del pase controla cómo se disponen los campos y qué imágenes serán utilizadas.

La siguiente tabla muestra las diferentes imágenes que soporta cada tipo de pase y cuál es su disposición:

Tipo de pase	**Imágenes soportadas**	**Disposición**
Tarjeta de embarque	logo, icon, footer	Ver Figura 7
Cupón	logo, icon, strip	Ver Figura 8
Entrada a evento	logo, icon, strip, background, thumbnail. Si se proporciona un strip no se deberá añadir ni background ni thumbnail.	Ver Figura 9
Genérico	logo, icon, thumbnail	Ver Figura 10
Tarjeta de fidelización	logo, icon, strip	Ver Figura 11

Tabla 6: Tipo de pases, imágenes y disposición [7]

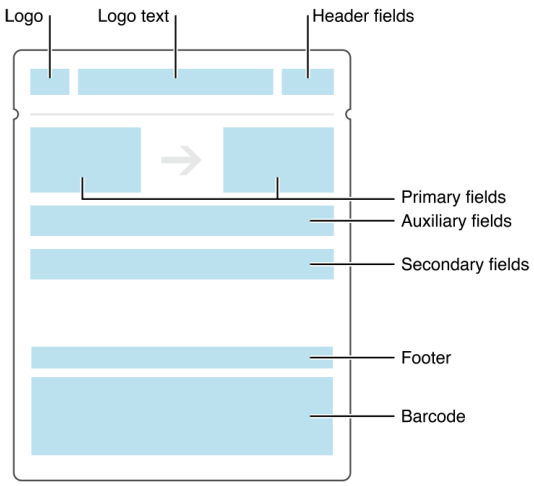

Figura 7: Disposición de una tarjeta de embarque [7]

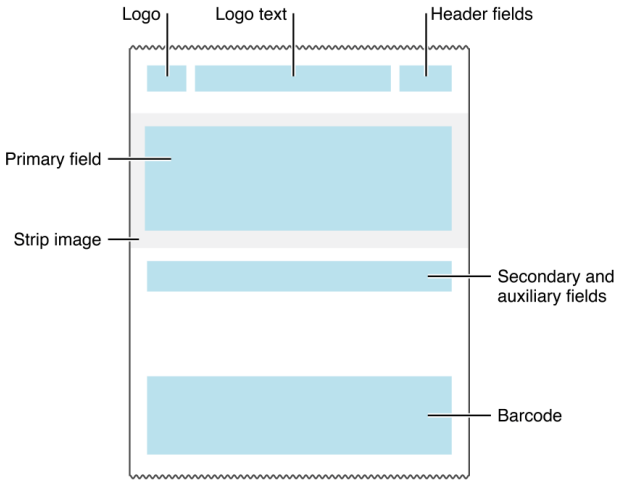

Figura 8: Disposición de un cupón [7]

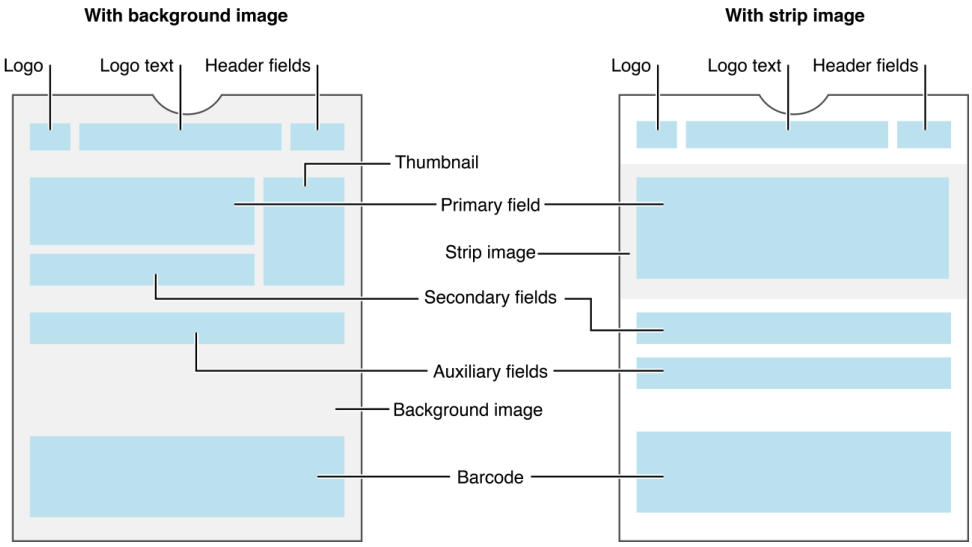

Figura 9: Disposición de una entrada a evento [7]

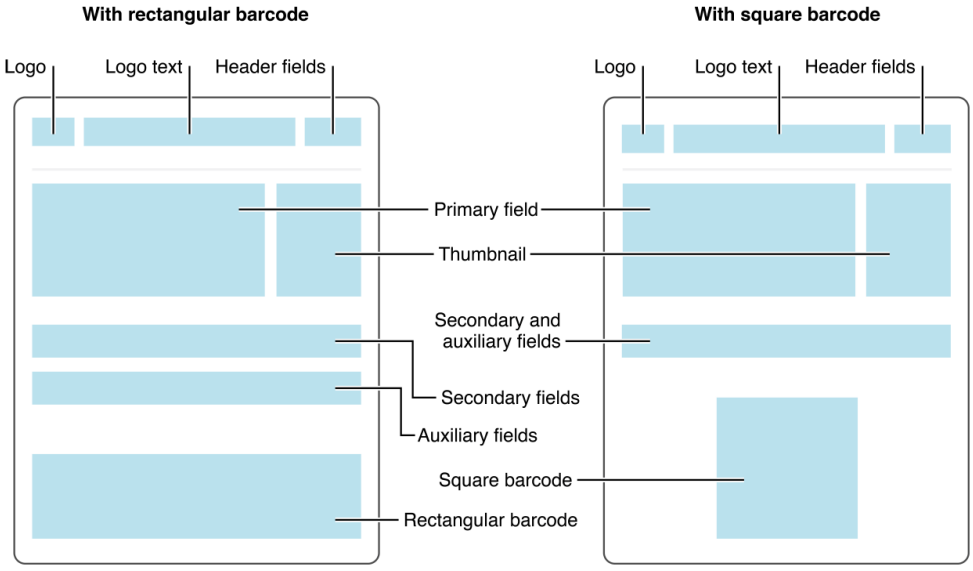

Figura 10: Disposición de un pase genérico [7]

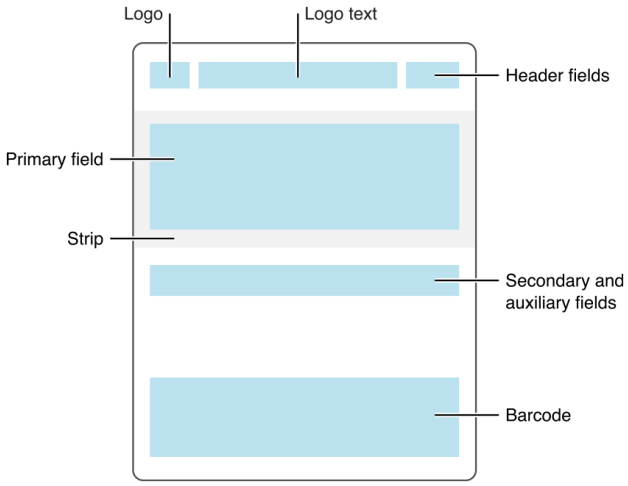

Figura 11: Disposición de una tarjeta de fidelización [7]

El número de campos que pueden aparecer en la parte frontal del pase viene determinado por el tipo de pase:

- En general, un pase puede tener como mucho tres campos en la cabecera (header fields), un solo campo principal (primary field), hasta cuatro campos secundarios (secondary fields) y cuatro campos auxiliares (auxiliary fields).
- Las tarjetas de embarque tendrán como máximo dos campos principales y hasta cinco campos auxiliares.
- Los cupones, tarjetas de fidelización y los pases genéricos con código de barras cuadrado tendrán un total de cuatro campos secundarios y auxiliares, combinados.

La información que se muestra en el pase está dividida en campos. Cada campo está definido por un diccionario o mapa, que proporciona una valor y una etiqueta (que se mostrarán al usuario), una clave única e información sobre cómo debería formatearse el valor.

Los campos principales contienen la información más importante y se muestran de una forma destacada en el pase. Los campos secundarios son menos importantes y menos prominentes, y los campos auxiliares lo son aún menos. Los campos presentes en la cabecera contienen información altamente destacada, y son los únicos campos visibles cuando los pases no están desplegados.

```
{
  "description" : "Boarding pass for October 4, San Francisco to London",
  "formatVersion" : 1,
  "passTypeIdentifier" : "pass.com.example.boarding-pass",
  "serialNumber" : "123456",
  "boardingPass" : {
    "primaryFields" : [
      {
        "key" : "origin",
        "label" : "San Francisco",
        "value" : "SFO"
      },
      {
        "key" : "destination",
        "label" : "London",
        "value" : "LHR"
      }
    ],
    "secondaryFields" : [
      {
        "key" : "boarding-gate",
        "label" : "Gate",
        "value" : "F12"
      }
    ],
    "auxiliaryFields" : [
      {
        "key" : "seat",
        "label" : "Seat",
        "value" : "7A"
      },
      {
        "key" : "passenger-name",
        "label" : "Passenger",
        "value" : "John Appleseed"
      }
    ],
    "transitType" : "PKTransitTypeAir"
  }
}
```

Figura 12: Pase tipo boardingPass con campos sencillos [7]

Los campos mencionados anteriormente soportan tres tipos diferentes de formato: Alineación, formato de fecha y de número:

- Para establecer la alineación, se especificará un valor para la clave `alignment` en el diccionario de campos.
- Para formatear una fecha, se proporcionarán los campos `dateStyle` y `timeStyle`.
- Para formatear un valor numérico o tipo de moneda, se añadirán los campos `numberStyle` o `currencyCode`.

En la siguiente figura se puede ver un ejemplo en el que se está dando formato a una fecha:

```
{
  "dateStyle": "PKDateStyleMedium",
  "isRelative": true,
  "key": "doors-open",
  "label": "Doors open",
  "timeStyle": "PKDateStyleShort",
  "value": "2013-08-10T19:30-06:00"
}
```

Figura 13: Ejemplo de campo con formato de fecha [7]

El espacio en la parte frontal del pase es limitado y el contenido de los campos debe ser breve. Para añadir más campos con una longitud mucho mayor está la parte trasera del pase que mostrará un listado con todos esos campos. Un ejemplo de información que podría ir en la parte trasera del pase son los términos y condiciones, sinopsis del evento, etc.

Otro elemento importante de los pases es el código de barras que proporcionan información para el posterior canjeo del mismo. Para añadir un código de barras al fichero *pass.json* hay que añadir una entrada a la lista de `barcodes`. Cada entrada define el formato del código de barras y el mensaje que se deberá mostrar y la codificación. Se puede proporcionar un texto alternativo por si se produjera un error al leer el código.

```
{
  ...

  "barcodes": [
    {
      "message": "ABCD 123 EFGH 456 IJKL 789 MNOP",
      "format": "PKBarcodeFormatPDF417",
      "messageEncoding": "iso-8859-1"
    }
  ]
}
```

Figura 14: Contenido parcial de un pase con barcode [7]

Hay cuatro tipos de códigos de barras:

QR

Figura 15: QR

PDF417

Figura 16: PDF417

Aztec

Figura 17: Aztec

Code 128

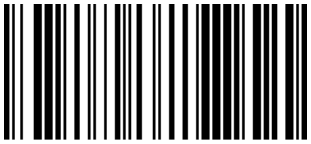

Figura 18: Code 128

Al igual que se puede aplicar formato al tipo de dato, se puede personalizar los colores del texto con el formato `rgb(0, 255, 0)`. Se pueden personalizar tres colores diferentes:

- El color de fondo (`backgroundColor`), se utiliza para determinar el color de la parte frontal y trasera del pase. Si se proporciona una imagen de fondo este valor es ignorado.
- El color destacado (`foregroundColor`), se utiliza para el valor de los campos que se muestran en la parte frontal del pase.
- Color para las etiquetas (`labelColor`), utilizado para las etiquetas de los campos que se muestran en la parte frontal del pase.

Por último, tal y como se ha comentado al principio, los pases pueden estar localizados, para localizar un pase, se debe crear un fichero llamado *pass.strings* dentro de una carpeta con el código ISO de dos dígitos del país con la extensión `.lproj`. A continuación se muestra un ejemplo de pase localizado en español y en inglés.

```
{
  "primaryFields": [
    {
      "key": "origin",
      "label": "origin_SVQ",
      "value": "SVQ"
    },
    {
      "key": "destination",
      "label": "destination_LHR",
      "value": "LHR"
    }
  ]
}
```

Figura 19: Campo de un pase de ejemplo en el que el campo label está localizado [7]

```
"origin_SVQ" = "Sevilla";
"destination_LHR" = "Londres";
```

Figura 20: Fichero localización en español dentro de directorio es.lproj [7]

```
"origin_SVQ" = "Seville";
"destination_LHR" = "London";
```

Figura 21: Fichero localización en inglés dentro de directorio en.lproj [7]

2.4. Lectores de archivos Passbook

Existen diversas aplicaciones para la lectura de archivos Passbook y a continuación se procederá a mostrar las más destacadas. Primero se hablará de *Wallet*, aplicación oficial de Apple que viene preinstalada en todos sus terminales tanto en sus dispositivos móviles iPhone

como en los ordenadores Mac, para posteriormente discutir las distintas soluciones que se ofrecen en Android, hablaremos sobre las tres aplicaciones más descargadas en Google Play para leer pases.

2.4.1. Wallet (iOS)

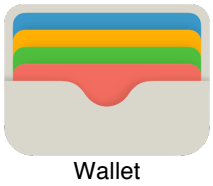

Figura 22: Icono aplicación Wallet

Wallet es la aplicación oficial de Apple y la referencia para el resto de aplicaciones, hace uso de la librería PassKit para realizar todas las funciones que se han descrito en las secciones anteriores: Permite a los usuarios gestionar sus pases, actualizarlos mediante notificaciones push, notificar al usuario en la pantalla de bloqueo si se produce algún evento importante relacionado con alguno de ellos, etc.

Figura 23: Wallet - Listado de pases y detalle

En esta imagen, se puede apreciar a la izquierda la pantalla principal de la aplicación que muestra todos los pases apilados uno encima de otro como si de una cartera física se tratara. La parte que queda visible son los campos de la cabecera, que es la información que siempre estará presente.

En la parte de la derecha, se muestra el detalle de un pase genérico, dicho pase mantiene un balance de saldo e identifica al usuario. También se puede apreciar el barcode tipo PDF417 con un texto alternativo por si acaso la lectura del pase fallara.

2.4.2. PassWallet

Figura 24: Aplicación PassWallet [8]

Esta aplicación es la más descargada en la tienda de aplicaciones de Google con de un millón a cinco millones de descargas. El diseño de la aplicación es muy similar a la original de iOS como se puede apreciar en las siguientes capturas.

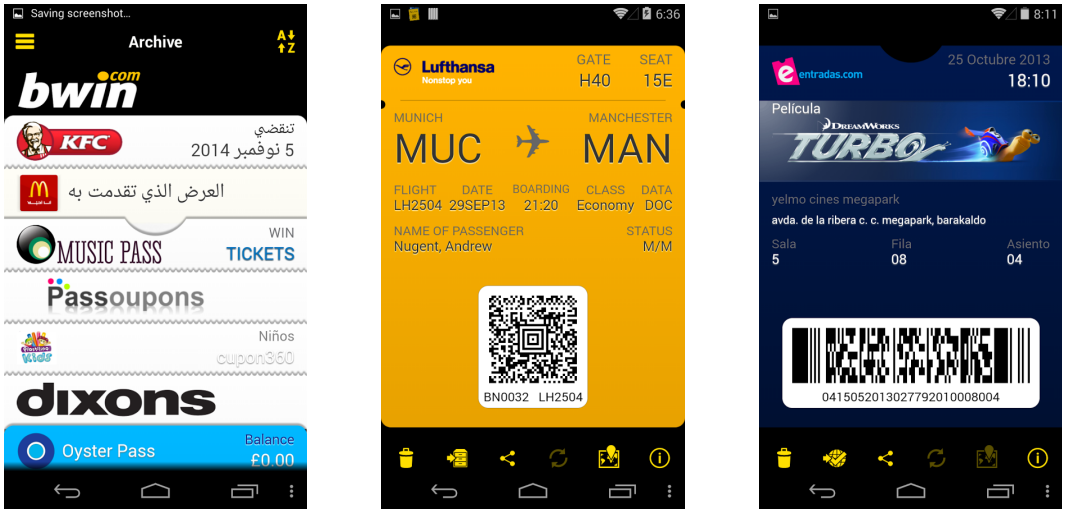

Figura 25: Capturas aplicación PassWallet [8]

El hecho de que el diseño sea exactamente igual que la aplicación de la plataforma iOS hace que para los usuarios de Android no resulte familiar y que no se integre con el sistema operativo ya que no sigue los guías de diseño establecidas por Google.

2.4.3. Pass2U

Figura 26: Aplicación Pass2U [9]

Esta aplicación también cuenta con una gran aceptación dentro de los usuarios de la plataforma con un índice de descargas de 100.000 a 500.000. Es una aplicación bastante completa, pero tiene el mismo fallo que la anterior, el diseño es prácticamente una copia de la versión oficial y no resulta intuitiva para los usuarios de Android, de hecho utiliza hasta la misma iconografía dentro de los pases.

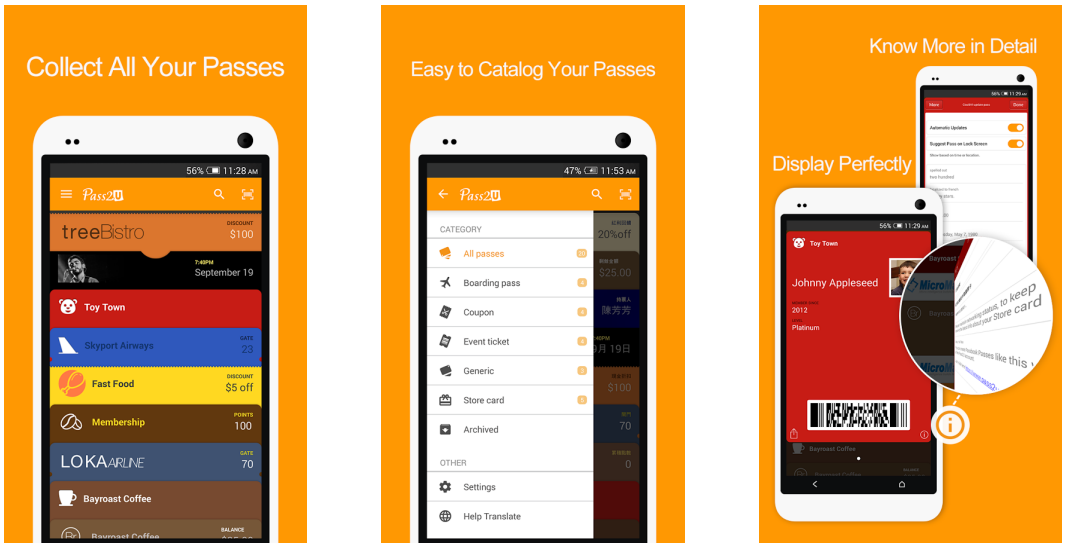

Figura 27: Capturas aplicación Pass2U [9]

2.4.4. PassAndroid

Figura 28: Aplicación PassAndroid [10]

PassAndroid está dentro del mismo rango de descargas que la aplicación anterior, ésta ha optado por realizar un diseño alternativo que no sigue ninguno de los patrones descritos en las secciones anteriores como se puede apreciar en las siguientes capturas.

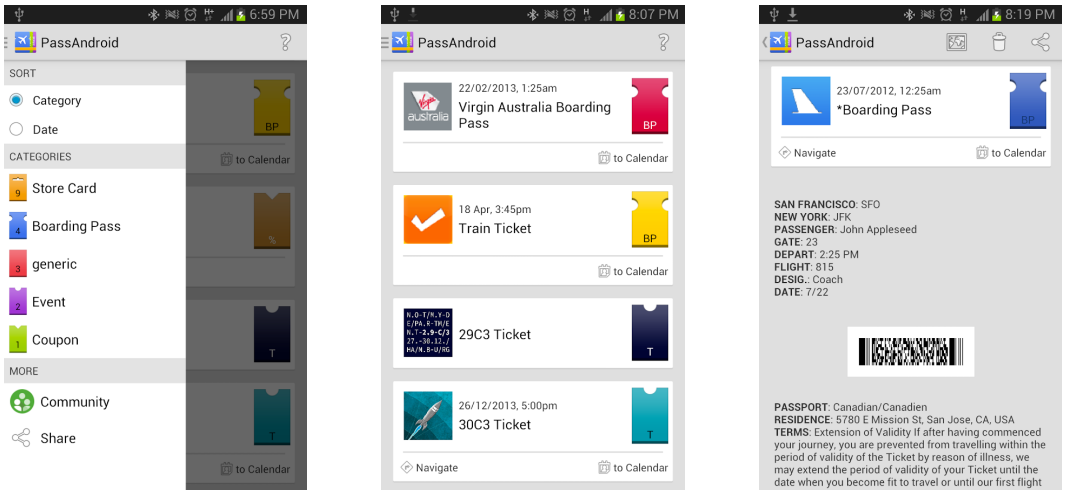

Figura 29: Capturas aplicación PassAndroid [10]

Como se puede observar en el detalle del pase, los datos se muestran en forma de listado y no quedan legibles. Por ejemplo, en la tercera imagen, se trata de una tarjeta de embarque y no queda claro, de un simple vistazo, cuál es el punto de partida y cuál el de llegada.

Capítulo 3
Análisis

Concluido el estudio del estado del arte, se dispone de la información necesaria para abordar el siguiente paso, el análisis del sistema. En esta sección se recopilarán los requisitos del usuario y requisitos software.

Una vez obtenidos los requisitos del sistema, se procederá detallar los casos de uso, que servirán como guía tanto para el desarrollo como para el testeo de la aplicación.

3.1. Requisitos de usuario

En este apartado se van a describir los distintos requisitos de usuario a partir de las indicaciones dadas por el cliente, clasificándolos como requisitos de capacidad o de restricción. Cada requisito está especificado siguiendo un formato tabular conteniendo la siguiente información para cada uno de ellos:

ID:	**Tipo:**	**Necesidad:**
Título:		
Fuente: Cliente		
Descripción:		
Estabilidad:		
Verificabilidad:		

La tabla cuenta con los siguientes campos:

- **ID:** identifica de forma univoca un requisito, siguiendo las reglas de nombrado CAP-X, para los requisitos de capacidad, y RES-X, para los de restricción. El valor de X es numérico y empieza en uno.
- **Tipo:** tipo del requisito.
- **Necesidad:** obligatorio, conveniente u opcional.
- **Título:** breve descripción del requisito.
- **Fuente:** se refiere a quién ha propuesto el requisito.
- **Descripción:** información detallada del requisito.
- **Estabilidad:** probabilidad de que el requisito cambie
- **Verificabilidad:** cómo de verificable o comprobable debe ser el requisito del sistema.

3.1.1. Requisitos de capacidad

Los requisitos de usuario de capacidad, que indican qué es capaz de realizar el sistema. Se listan a continuación.

ID: CAP-1	**Tipo:** Capacidad	**Necesidad:** Obligatorio
Título: Añadir fichero Passbook		
Fuente: Cliente		
Descripción: El usuario será capaz de añadir ficheros Passbook presentes en su tarjeta de memoria.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 7: CAP-1

ID: CAP-2	**Tipo:** Capacidad	**Necesidad:** Conveniente
Título: Añadir fichero al ser descargado		
Fuente: Cliente		
Descripción: El usuario será capaz de añadir ficheros Passbook tras haberlos descargado. Se mostrará un diálogo en el que se mostrará la opción de abrir el fichero usando la aplicación.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 8: CAP-2

ID: CAP-3	**Tipo:** Capacidad	**Necesidad:** Conveniente
Título: Añadir fichero desde navegador de archivos		
Fuente: Cliente		
Descripción: El usuario será capaz de añadir ficheros Passbook desde el navegador de archivos. Se mostrará un diálogo en el que se mostrará la opción de abrir el fichero usando la aplicación.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 9: CAP-3

ID: CAP-4	**Tipo:** Capacidad	**Necesidad:** Obligatorio
Título: Listar ficheros Passbook		
Fuente: Cliente		
Descripción: El usuario será capaz de visualizar todos los pases añadidos hasta el momento.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 10: CAP-4

ID: CAP-5	**Tipo:** Capacidad	**Necesidad:** Obligatorio
Título: Visualizar detalle pase *boardingPass*		
Fuente: Cliente		
Descripción: El usuario será capaz de visualizar el detalle de un pase tipo tarjeta de embarque.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 11: CAP-5

ID: CAP-6	**Tipo:** Capacidad	**Necesidad:** Obligatorio
Título: Visualizar detalle pase *coupon*		
Fuente: Cliente		
Descripción: El usuario será capaz de visualizar el detalle de un pase tipo cupón.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 12: CAP-6

ID: CAP-7	**Tipo:** Capacidad	**Necesidad:** Obligatorio
Título: Visualizar detalle pase *eventTicket*		
Fuente: Cliente		
Descripción: El usuario será capaz de visualizar el detalle de un pase tipo entrada a evento.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 13: CAP-7

ID: CAP-8	**Tipo:** Capacidad	**Necesidad:** Obligatorio
Título: Visualizar detalle pase *storeCard*		
Fuente: Cliente		
Descripción: El usuario será capaz de visualizar el detalle de un pase tipo tarjeta de fidelización.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 14: CAP-8

ID: CAP-9	**Tipo:** Capacidad	**Necesidad:** Obligatorio
Título: Visualizar detalle pase *generic*		
Fuente: Cliente		
Descripción: El usuario será capaz de visualizar el detalle de un pase tipo genérico.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 15: CAP-9

ID: CAP-10	**Tipo:** Capacidad	**Necesidad:** Obligatorio
Título: Visualizar parte trasera del pase		
Fuente: Cliente		
Descripción: El usuario será capaz visualizar la parte trasera del pase.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 16: CAP-10

ID: CAP-11	**Tipo:** Capacidad	**Necesidad:** Obligatorio
Título: Guardar pase		
Fuente: Cliente		
Descripción: El usuario será capaz de guardar un pase		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 17: CAP-11

ID: CAP-12	**Tipo:** Capacidad	**Necesidad:** Obligatorio
Título: Eliminar pase		
Fuente: Cliente		
Descripción: El usuario será capaz de eliminar un pase.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 18: CAP-12

ID: CAP-13	**Tipo:** Capacidad	**Necesidad:** Conveniente
Título: Compartir pase		
Fuente: Cliente		
Descripción: El usuario será capaz de compartir el pase con otra persona a través de email, servicios de mensajería, etc.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 19: CAP-13

ID: CAP-14	**Tipo:** Capacidad	**Necesidad:** Conveniente
Título: Añadir evento al calendario		
Fuente: Cliente		
Descripción: El usuario será capaz de crear un evento en el calendario con la fecha relevante del evento, fecha de vuelo, etc.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 20: CAP-14

ID: CAP-15	**Tipo:** Capacidad	**Necesidad:** Obligatorio
Título: Visualizar código de barras en pantalla completa		
Fuente: Cliente		
Descripción: El usuario será capaz de visualizar el código de barras en pantalla completa para facilitar la correcta visualización por los dispositivos de escaneado.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 21: CAP-15

ID: CAP-16	**Tipo:** Capacidad	**Necesidad:** Obligatorio
Título: Localización de archivos		
Fuente: Cliente		
Descripción: El usuario será capaz de visualizar los archivos localizados según su *Locale* si dicho idioma está disponible.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 22: CAP-16

3.1.2. Requisitos de restricción

En este apartado se describen los requisitos de usuario de restricción, que indican cómo se deben realizar las funcionalidades.

ID: RES-1	**Tipo:** Restricción	**Necesidad:** Obligatorio
Título: Plataforma Android		
Fuente: Cliente		
Descripción: El dispositivo en el que se instale la aplicación debe contar con una versión de Android igual a superior a la 4.1 (API 16).		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 23: RES-1

ID: RES-2	**Tipo:** Restricción	**Necesidad:** Obligatorio
Título: Permisos		
Fuente: Cliente		
Descripción: Al instalar la aplicación, el usuario debe otorgar los permisos necesarios a la aplicación para su correcto funcionamiento. Estos permisos son: • Montar la unidad de almacenamiento externo. • Escribir en la unidad de almacenamiento externo.		
Estabilidad: Alta		
Verificabilidad: Alta		

Tabla 24: RES-2

3.2. Casos de uso

Un caso de uso permite describir las interacciones entre los usuarios del sistema con el propio sistema para alcanzar un objetivo.

Cada caso de uso va a ser especificado con un formato tabular en el que se encuentran los siguientes campos:

- **ID:** identificador del caso de uso.
- **Título:** breve descripción del caso de uso
- **Descripción:** pasos seguidos por el usuario para llegar a la situación planteada.
- **Alternativas posibles:** diferentes escenarios en los que se puede encontrar el usuario.
- **Requisitos relacionados:** requisito de usuario relacionado con este caso de uso.

ID: CU-1	**Título:** Importar fichero Passbook
Descripción 1. El usuario pulsa sobre el botón para añadir pase 2. La aplicación mostrará el navegador de ficheros nativo. 3. El usuario selecciona un fichero. 4. La aplicación añadirá el fichero al listado de pases y mostrará el detalle del mismo. **Alternativas posibles** 4.1. La aplicación mostrará un error si el archivo no pude ser abierto.	
Requisitos relacionados: CAP-1, CAP-11	

Tabla 25: CU-1

ID: CU-2	**Título:** Importar fichero Passbook al ser descargado
Descripción 1. El usuario descarga un pase en su navegador 2. La aplicación debe estar escuchando ese evento, se abrirá al recibirlo, añadirá el fichero al listado de pases y mostrará el detalle del mismo. **Alternativas posibles** 2.1. Si hay más de una aplicación instalada escuchando este tipo de eventos, la aplicación aparecerá en el listado de selección con las demás aplicaciones.	
Requisitos relacionados: CAP-2, CAP-11	

Tabla 26: CU-2

ID: CU-3	**Título:** Importar pase al ser abierto desde el navegador de archivos
Descripción 1. El usuario abre un fichero Passbook desde el navegador de archivos. 2. La aplicación debe estar en la lista de aplicaciones compatibles con esos ficheros y se abrirá al recibirlo, añadiendo el fichero al listado de pases y mostrando el detalle del mismo. **Alternativas posibles** 2.1. Si hay más de una aplicación instalada compatible con este tipo de archivos, la aplicación aparecerá en el listado de selección con las demás aplicaciones.	
Requisitos relacionados: CAP-3, CAP-11	

Tabla 27: CU-3

ID: CU-4	**Título:** Mostrar lista de pases
Descripción 1. El usuario abre la aplicación. 2. La aplicación mostrará las cabeceras de los diferentes pases importados por el usuario hasta el momento. Los diferentes campos aparecerán en el idioma correspondiente si dentro del pase se proporcionan archivos de localización. **Alternativas posibles** 2.1. Si no hay ningún pase mostrará un texto indicándolo e informará al usuario sobre qué debe hacer para añadir uno.	
Requisitos relacionados: CAP-4, CAP-16	

Tabla 28: CU-4

ID: CU-5	**Título:** Mostrar cabecera de pase en listado
Descripción 1. El usuario abre la aplicación. 2. La aplicación mostrará las cabeceras de los diferentes pases importados por el usuario hasta el momento teniendo en cuenta que una cabecera puede tener un logo, un título y hasta tres campos destacados en la misma. Los diferentes campos aparecerán en el idioma correspondiente si dentro del pase se proporcionan archivos de localización. **Alternativas posibles** 2.1. Si no hay ningún pase mostrará un texto indicándolo e informará al usuario sobre qué debe hacer para añadir uno.	
Requisitos relacionados: CAP-4, CAP-16	

Tabla 29: CU-5

ID: CU-6	**Título:** Mostrar cabecera pase en el detalle
Descripción 1. El usuario abre la aplicación. 2. La aplicación mostrará los diferentes pases importados por el usuario hasta el momento. 3. El usuario selecciona un pase 4. La aplicación mostrará el detalle de un pase y la cabecera irá en la parte superior teniendo en cuenta que una cabecera puede tener un logo, un título y hasta tres campos destacados en la misma. Los diferentes campos aparecerán en el idioma correspondiente si dentro del pase se proporcionan archivos de localización. **Alternativas posibles** 2.1. Si no hay ningún pase mostrará un texto indicándolo e informará al usuario sobre qué debe hacer para añadir uno.	
Requisitos relacionados: CAP-5, CAP-6, CAP-7, CAP-8, CAP-9, CAP-16	

Tabla 30: CU-6

ID: CU-7	**Título:** Visualizar detalle de un pase tipo tarjeta de embarque
Descripción 1. El usuario abre la aplicación. 2. La aplicación mostrará los diferentes pases importados por el usuario hasta el momento. 3. El usuario selecciona un pase tipo `boardingPass`. 4. La aplicación mostrará el detalle del pase teniendo en cuenta que tendrá una cabecera, dos campos principales (origen y destino), que tipo de transporte puede ser `PKTransitTypeAir`, `PKTransitTypeBoat`, `PKTransitTypeBus`, `PKTransitTypeGeneric` o `PKTransitTypeTrain`, que puede tener hasta cinco campos auxiliares y cuatro campos secundarios, además, mostrará el código de barras y el footer si lo hubiera. Los diferentes campos aparecerán en el idioma correspondiente si dentro del pase se proporcionan archivos de localización. **Alternativas posibles** 2.1. Si no hay ningún pase mostrará un texto indicándolo e informará al usuario sobre qué debe hacer para añadir uno.	
Requisitos relacionados: CAP-5, CAP-15, CAP-16	

Tabla 31: CU-7

ID: CU-8	**Título:** Visualizar detalle de un pase tipo cupón
Descripción 1. El usuario abre la aplicación. 2. La aplicación mostrará los diferentes pases importados por el usuario hasta el momento. 3. El usuario selecciona un pase tipo `coupon`. 4. La aplicación mostrará el detalle del pase teniendo en cuenta que tendrá una cabecera, un solo campo principal, que podrá tener como máximo cinco campos auxiliares y secundarios combinados si el código de barras es cuadrado o hasta cuatro campos auxiliares y cuatro campos secundarios si el código de barras es rectangular. Además, mostrará la imagen *strip* detrás del campo principal si la hubiera. Los diferentes campos aparecerán en el idioma correspondiente si dentro del pase se proporcionan archivos de localización. **Alternativas posibles** 2.1. Si no hay ningún pase mostrará un texto indicándolo e informará al usuario sobre qué debe hacer para añadir uno.	
Requisitos relacionados: CAP-6, CAP-15, CAP-16	

Tabla 32: CU-8

ID: CU-9	**Título:** Visualizar detalle de un pase tipo entrada a evento
Descripción 1. El usuario abre la aplicación. 2. La aplicación mostrará los diferentes pases importados por el usuario hasta el momento. 3. El usuario selecciona un pase tipo `eventTicket`. 4. La aplicación mostrará el detalle del pase teniendo en cuenta que tendrá una cabecera, un solo campo principal, que podrá tener hasta cuatro campos auxiliares y cuatro campos secundarios, un *thumbnail* e imagen de fondo (si hubiera imagen de fondo se ignoraría el color de fondo). Además, mostrará la imagen *strip* detrás del campo principal y *thumbnail* si la hubiera. Los diferentes campos aparecerán en el idioma correspondiente si dentro del pase se proporcionan archivos de localización. **Alternativas posibles** 2.1. Si no hay ningún pase mostrará un texto indicándolo e informará al usuario sobre qué debe hacer para añadir uno.	
Requisitos relacionados: CAP-7, CAP-16	

Tabla 33: CU-9

ID: CU-10	**Título:** Visualizar detalle de un pase tipo genérico
Descripción 1. El usuario abre la aplicación. 2. La aplicación mostrará los diferentes pases importados por el usuario hasta el momento. 3. El usuario selecciona un pase tipo `generic`. 4. La aplicación mostrará el detalle del pase teniendo en cuenta que tendrá una cabecera, un solo campo principal, y que podrá tener como máximo cinco campos auxiliares y secundarios combinados si el código de barras es cuadrado o hasta cuatro campos auxiliares y cuatro campos secundarios si el código de barras es rectangular. Los diferentes campos aparecerán en el idioma correspondiente si dentro del pase se proporcionan archivos de localización. **Alternativas posibles** 2.1. Si no hay ningún pase mostrará un texto indicándolo e informará al usuario sobre qué debe hacer para añadir uno.	
Requisitos relacionados: CAP-9, CAP-15, CAP-16	

Tabla 34: CU-10

ID: CU-11	**Título:** Visualizar detalle de un pase tipo tarjeta de fidelización
Descripción 1. El usuario abre la aplicación. 2. La aplicación mostrará los diferentes pases importados por el usuario hasta el momento. 3. El usuario selecciona un pase tipo `storeCard`. 4. La aplicación mostrará el detalle del pase teniendo en cuenta que tendrá una cabecera, un solo campo principal, y que podrá tener como máximo cinco campos auxiliares y secundarios combinados si el código de barras es cuadrado o hasta cuatro campos auxiliares y cuatro campos secundarios si el código de barras es rectangular. Además, mostrará la imagen *strip* detrás del campo principal si la hubiera. Los diferentes campos aparecerán en el idioma correspondiente si dentro del pase se proporcionan archivos de localización. **Alternativas posibles** 2.1. Si no hay ningún pase mostrará un texto indicándolo e informará al usuario sobre qué debe hacer para añadir uno.	
Requisitos relacionados: CAP-8, CAP-15, CAP-16	

Tabla 35: CU-11

ID: CU-12	**Título:** Visualizar parte trasera de un pase
Descripción 1. El usuario abre la aplicación. 2. La aplicación mostrará los diferentes pases importados por el usuario hasta el momento. 3. El usuario selecciona un pase. 4. La aplicación mostrará el detalle del pase. 5. El usuario desliza el dedo hacia la izquierda sobre la tarjeta. 6. La aplicación mostrará la parte trasera del pase en la que irán todos los campos dentro del objeto `backFields`. Los diferentes campos aparecerán en el idioma correspondiente si dentro del pase se proporcionan archivos de localización. **Alternativas posibles** 2.1. Si no hay ningún pase mostrará un texto indicándolo e informará al usuario sobre qué debe hacer para añadir uno.	
Requisitos relacionados: CAP-10, CAP-16	

Tabla 36: CU-12

ID: CU-13	**Título:** Eliminar un pase
Descripción 1. El usuario abre la aplicación. 2. La aplicación mostrará los diferentes pases importados por el usuario hasta el momento. 3. El usuario selecciona un pase. 4. La aplicación mostrará el detalle del pase. 5. El usuario pulsa sobre el icono de una papelera. 6. La aplicación eliminará el pase. **Alternativas posibles** 2.1. Si no hay ningún pase mostrará un texto indicándolo e informará al usuario sobre qué debe hacer para añadir uno.	
Requisitos relacionados: CAP-12	

Tabla 37: CU-13

ID: CU-14	**Título:** Compartir un pase
Descripción 1. El usuario abre la aplicación. 2. La aplicación mostrará los diferentes pases importados por el usuario hasta el momento. 3. El usuario selecciona un pase. 4. La aplicación mostrará el detalle del pase. 5. El usuario pulsa sobre el icono de tres puntos formando un triángulo invertido con la parte superior abierta. 6. La aplicación mostrará un diálogo al usuario con las diferentes aplicaciones que tiene instaladas en su dispositivo capaces de enviar un archivo. 7. El usuario selecciona una aplicación 8. El pase aparecerá como adjunto en la aplicación seleccionada. **Alternativas posibles** 2.1. Si no hay ningún pase mostrará un texto indicándolo e informará al usuario sobre qué debe hacer para añadir uno.	
Requisitos relacionados: CAP-13	

Tabla 38: CU-14

ID: CU-15	**Título:** Crear evento en calendario
Descripción 1. El usuario abre la aplicación. 2. La aplicación mostrará los diferentes pases importados por el usuario hasta el momento. 3. El usuario selecciona un pase. 4. La aplicación mostrará el detalle del pase. 5. El usuario pulsa sobre el icono del calendario que se encuentra dentro de un círculo. 6. La aplicación mostrará un diálogo al usuario con las diferentes aplicaciones que tiene instaladas en su dispositivo que permiten añadir un pase. 7. El usuario selecciona una aplicación 8. El usuario verá el resumen del evento y podrá editarlo. **Alternativas posibles** 2.1. Si no hay ningún pase mostrará un texto indicándolo e informará al usuario sobre qué debe hacer para añadir uno.	
Requisitos relacionados: CAP-14	

Tabla 39: CU-15

3.3. Requisitos software

Los requisitos software especifican en detalle las funcionalidades del sistema. La información de los requisitos de software proviene de los requisitos del usuario.

Se pueden distinguir dos tipos diferentes de requisitos de software, los requisitos de funcionales y no funcionales. Los primeros definen qué tiene que hacer el sistema, y los segundos ponen restricciones indicando cómo se deben hacer las funcionalidades.

3.3.1. Requisitos funcionales

Este tipo de requisitos definen el propósito y funcionalidad de la aplicación. Se indicará qué requisitos de usuario están relacionados con cada uno de estos requisitos.

ID: RF-1	**Tipo:** Funcional	**Necesidad:** Obligatorio
Título: Abrir fichero Passbook		
Fuente: Cliente		
Descripción: La aplicación debe mostrar un botón en la pantalla principal que permita al usuario importar un nuevo pase. Al pulsar dicho botón, se abrirá el sistema de navegación de archivos nativo y el usuario podrá seleccionar el pase que necesite. El único formato soportado será *.pkpass*. Si el usuario intenta abrir otro tipo de fichero la aplicación mostrará un error indicando que ese fichero no puede ser abierto.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-1		

Tabla 40: RF-1

ID: RF-2	**Tipo:** Funcional	**Necesidad:** Obligatorio
Título: Abrir fichero Passbook tras ser descargado		
Fuente: Cliente		
Descripción: La aplicación deberá estar escuchando el evento del sistema que indica que se ha descargado un archivo. Si dicho archivo es de tipo *.pkpass*, la aplicación se ofrecerá para abrirlo por lo que: • Si es la única aplicación instalada, esta se abrirá automáticamente. • Si hay más de una aplicación suscrita para abrir ese tipo de archivos, la aplicación aparecerá en la lista que el sistema mostrará al usuario.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-2		

Tabla 41: RF-2

ID: RF-3	**Tipo:** Funcional	**Necesidad:** Obligatorio
Título: Abrir fichero Passbook tras ser seleccionado en el navegador de archivos		
Fuente: Cliente		
Descripción: La aplicación deberá estar escuchando el evento del sistema que indica que se ha seleccionado un archivo dentro del navegador de ficheros. Al igual que en el requisito RF-3, si dicho archivo es de tipo *.pkpass*, la aplicación se ofrecerá para abrirlo por lo que: • Si es la única aplicación instalada, esta se abrirá automáticamente.		

• Si hay más de una aplicación suscrita para abrir ese tipo de archivos, la aplicación aparecerá en la lista que el sistema mostrará al usuario.
Estabilidad: Alta
Verificabilidad: Alta
Requisito relacionado: CAP-3

Tabla 42: RF-3

ID: RF-4	**Tipo:** Funcional	**Necesidad:** Obligatorio
Título: Listar ficheros Passbook		
Fuente: Cliente		
Descripción: La aplicación mostrará al arrancar todos los pases que el usuario ha abierto hasta la fecha y que han sido guardados. Los campos del pase visibles en cada elemento del listado serán: • Logo • Logo text • Header fields		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-4		

Tabla 43: RF-4

ID: RF-5	**Tipo:** Funcional	**Necesidad:** Obligatorio
Visualizar detalle pase *boardingPass*		
Fuente: Cliente		
Descripción: La aplicación mostrará el detalle de un pase tipo *boardingPass* tras la selección del mismo en el listado del requisito RF-5 o tras su apertura como es el caso de los requisitos RF-1, RF-2 y RF-3. El detalle del pase será concordante con la Figura 7.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-5		

Tabla 44: RF-5

ID: RF-6	**Tipo:** Funcional	**Necesidad:** Obligatorio
Visualizar detalle pase *coupon*		
Fuente: Cliente		
Descripción: La aplicación mostrará el detalle de un pase tipo *coupon* tras la selección del mismo en el listado del requisito RF-5 o tras su apertura como es el caso de los requisitos RF-1, RF-2 y RF-3. El detalle del pase será concordante con la Figura 8.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-6		

Tabla 45: RF-6

ID: RF-7	**Tipo:** Funcional	**Necesidad:** Obligatorio
Visualizar detalle pase *eventTicket*		
Fuente: Cliente		
Descripción: La aplicación mostrará el detalle de un pase tipo *eventTicket* tras la selección del mismo en el listado del requisito RF-5 o tras su apertura como es el caso de los requisitos RF-1, RF-2 y RF-3. El detalle del pase será concordante con la Figura 9.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-7		

Tabla 46: RF-7

ID: RF-8	**Tipo:** Funcional	**Necesidad:** Obligatorio
Visualizar detalle pase *storeCard*		
Fuente: Cliente		
Descripción: La aplicación mostrará el detalle de un pase tipo *storeCard* tras la selección del mismo en el listado del requisito RF-5 o tras su apertura como es el caso de los requisitos RF-1, RF-2 y RF-3. El detalle del pase será concordante con la Figura 11.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-8		

Tabla 47: RF-8

ID: RF-9	**Tipo:** Funcional	**Necesidad:** Obligatorio
Visualizar detalle pase *generic*		
Fuente: Cliente		
Descripción: La aplicación mostrará el detalle de un pase tipo *generic* tras la selección del mismo en el listado del requisito RF-5 o tras su apertura como es el caso de los requisitos RF-1, RF-2 y RF-3. El detalle del pase será concordante con la Figura 10.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-9		

Tabla 48: RF-9

ID: RF-10	**Tipo:** Funcional	**Necesidad:** Obligatorio
Visualizar parte trasera del pase		
Fuente: Cliente		
Descripción: La aplicación mostrará la parte trasera de un pase que consiste en listar los campos contenidos en el objeto `backFields` dentro del JSON, `pass.json`.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-10		

Tabla 49: RF-10

ID: RF-11	**Tipo:** Funcional	**Necesidad:** Obligatorio
Guardar pase		
Fuente: Cliente		
Descripción: La aplicación persistirá la información de un pase tras la apertura del mismo (RF-1, RF-2 y RF-3), el proceso será automático y no requerirá ninguna interacción por parte del usuario.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-11		

Tabla 50: RF-11

ID: RF-12	**Tipo:** Funcional	**Necesidad:** Obligatorio
Eliminar pase		
Fuente: Cliente		
Descripción: En la pantalla de detalle de un pase (RF-5, RF-6, RF-7, RF-8 y RF-9), se mostrará el icono  en la parte superior. Si el usuario pulsa sobre ese icono, la aplicación mostrará un diálogo con el texto "¿Está seguro de que desea eliminar este pase?" con las opciones "Aceptar" y "Cancelar". Si el usuario pulsa sobre "Aceptar", la aplicación eliminará el pase persistido y el usuario no podrá volver a acceder a él; si pulsa sobre "Cancelar", el usuario se quedará en la pantalla en la que estaba.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-12		

Tabla 51: RF-12

ID: RF-13	**Tipo:** Funcional	**Necesidad:** Obligatorio
Compartir pase		
Fuente: Cliente		
Descripción: En la pantalla de detalle de un pase (RF-5, RF-6, RF-7, RF-8 y RF-9), se mostrará el icono  en la parte superior. Si el usuario pulsa sobre ese icono, la aplicación mostrará un diálogo con las diferentes aplicaciones del sistema que permiten adjuntar archivos. Cuando el usuario pulsa sobre una de dichas aplicaciones, el pase aparecerá como adjunto.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-13		

Tabla 52: RF-13

ID: RF-14	**Tipo:** Funcional	**Necesidad:** Obligatorio
Añadir evento al calendario		
Fuente: Cliente		
Descripción: En la pantalla de detalle de un pase (RF-5, RF-6, RF-7, RF-8 y RF-9), se mostrará el icono  . Si el usuario pulsa sobre ese icono, la aplicación mostrará un diálogo con las diferentes aplicaciones del sistema que permiten añadir eventos al calendario. Cuando el usuario pulsa sobre una de dichas aplicaciones, cierta información de dicho evento estará rellena, si se dispone de ella, como puede ser: • Nombre del evento.		

• Fecha. • Hora. • Localización.
Estabilidad: Alta
Verificabilidad: Alta
Requisito relacionado: CAP-14

Tabla 53: RF-14

ID: RF-15	**Tipo:** Funcional	**Necesidad:** Obligatorio
Visualizar código de barras en pantalla completa		
Fuente: Cliente		
Descripción: En la pantalla de detalle de un pase (RF-5, RF-6, RF-7, RF-8 y RF-9), se mostrará el código de barras siempre que el pase lo contenga. Si el usuario pulsa sobre esa imagen, la aplicación abrirá una nueva pantalla en la que se podrá ver el código de barras en pantalla completa y en un mayor tamaño para facilitar la lectura del mismo por los diferentes lectores.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-15		

Tabla 54: RF-15

ID: RF-16	**Tipo:** Funcional	**Necesidad:** Obligatorio
Localización de archivos		
Fuente: Cliente		
Descripción: La aplicación mostrará todos la información de los pases en el listado (RF-4), detalle (RF-5, RF-6, RF-7, RF-8 y RF-9) y parte trasera (RF-10) localizados en los idiomas que se encuentren presentes en el pase dentro de las carpetas llamadas `<código_idioma>.lproj`. Si el usuario pulsa sobre esa imagen, la aplicación abrirá una nueva pantalla en la que se podrá ver el código de barras en pantalla completa y en un mayor tamaño para facilitar la lectura del mismo por los diferentes lectores.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-16		

Tabla 55: RF-16

3.3.2. Requisitos no funcionales

ID: RNF-1	**Tipo:** No funcional	**Necesidad:** Obligatorio
Guardar pase en memoria externa		
Fuente: Cliente		
Descripción: Los diferentes pases de la información residirán en una carpeta en la memoria externa del teléfono.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: CAP-11		

Tabla 56: RNF-1

3.3.3. Requisitos de entorno

ID: RE-1	**Tipo:** Entorno	**Necesidad:** Obligatorio
Entorno aplicación móvil: Software		
Fuente: Cliente		
Descripción: La aplicación debe funcionar correctamente en móviles con versión de Android igual a superior a la 4.1 (API 16).		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: RES-1		

Tabla 57: RE-1

ID: RE-2	**Tipo:** Entorno	**Necesidad:** Obligatorio
Entorno aplicación móvil: Pantalla		
Fuente: Cliente		
Descripción: La aplicación debe funcionar correctamente en un Smartphone con pantalla táctil de una diagonal de hasta 5 pulgadas.		
Estabilidad: Alta		
Verificabilidad: Alta		
Requisito relacionado: RES-1		

Tabla 58: RE-2

3.4. Matrices de trazabilidad

Matriz de trazabilidad entre los requisitos de usuario y requisitos software.

RF\CAP	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	X															
2		X														
3			X													
4				X												
5					X											
6						X										
7							X									
8								X								
9									X							
10										X						
11											X					
12												X				
13													X			
14														X		
15															X	
16																X

Tabla 59: Matriz de trazabilidad requisitos de usuario y requisitos software

Matriz de trazabilidad entre los requisitos de usuario y casos de uso.

CU\CAP	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	X										X					
2		X									X					
3			X								X					
4				X												X
5				X												X
6					X	X	X	X	X							X
7					X										X	X
8						X									X	X
9							X									X
10									X						X	X

CU\CAP	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
11								X							X	X
12										X						X
13												X				
14													X			
15														X		

Tabla 60: Matriz de trazabilidad requisitos de usuario y casos de uso

Matriz de trazabilidad entre los casos de uso y requisitos software.

CU\RF	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	X										X					
2		X									X					
3			X								X					
4				X												X
5				X												X
6					X	X	X	X	X							X
7					X										X	X
8						X									X	X
9							X									X
10									X						X	X
11								X							X	X
12										X						X
13												X				
14													X			
15														X		

Tabla 61: Matriz de trazabilidad casos de uso y requisitos software

Capítulo 4

Diseño

Tras recopilar todos los requisitos en el análisis, se va a abordar el diseño del sistema. Se justificarán todas las decisiones tomadas en torno al diseño a lo largo del capítulo.

En primero lugar se hablará de la arquitectura de la aplicación en muy alto nivel para luego identificar las clases que forman parte de la aplicación más en detalle.

4.1. Arquitectura del sistema

La arquitectura de la aplicación se ha desarrollado siguiendo los principios de Clean Code [11], con la intención de generar un código robusto, fácil de mantener, probar y lo suficientemente flexible para adaptarse al crecimiento y los cambios.

La idea principal es simple, *Clean Architecture* viene a ser un conjunto de buenas prácticas que producen sistemas que son:

- Independientes del entorno.
- Testeable.
- Independientes de la interfaz de usuario.
- Independientes de la base de datos.
- Independientes de cualquier agente externo.

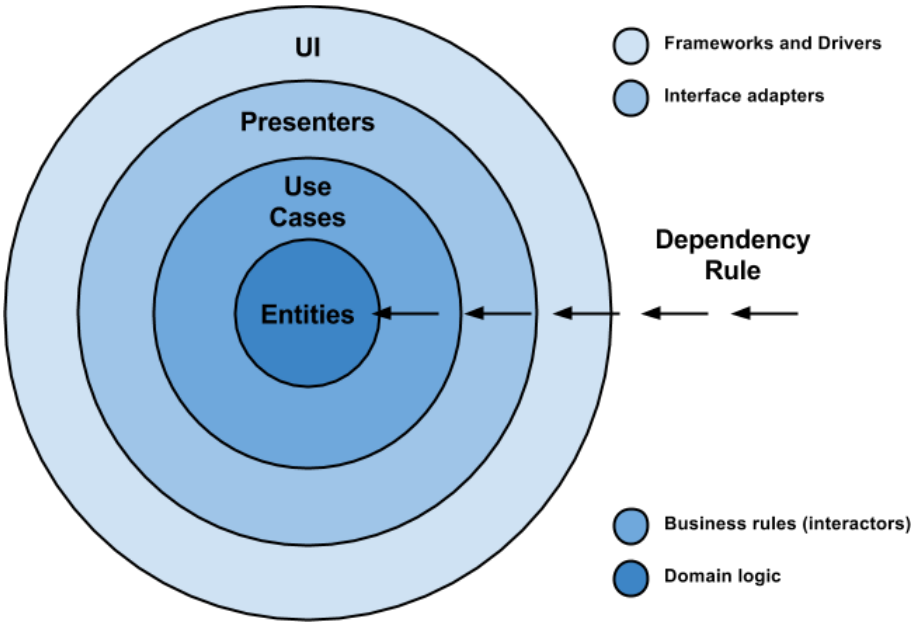

Figura 30: Regla de dependencia en Clean Code [11]

No es obligatorio usar siempre cuatro niveles, pueden usarse más, pero lo que sí hay que cumplir es la regla de dependencia: Las dependencias sólo pueden ir de fuera hacia adentro y ningún círculo interno puede saber de lo que se encuentra en los círculos superiores a él. En particular, el nombre de algo declarado en un círculo exterior, no debe ser mencionado en el código de un círculo interno. Esto incluye, funciones, clases, variables, o cualquier otra entidad.

Por el mismo motivo, los tipos de datos utilizados en un círculo externo, no deben ser utilizados por un círculo interno, ya que se intenta evitar que cualquier cambio en alguno de los círculos exteriores tengan algún impacto en los internos.

Existen diferentes componentes que se utilizarán para intentar llevar acabo este sistema:

- **Entidades:** son los objetos de negocio de la aplicación.
- **Casos de Uso:** se encargarán de enviar a y de obtener los datos de las entidades. También pueden ser llamados *Interactors*.
- **Adaptadores de interfaz:** su misión será convertir la información al formato más conveniente para los casos de uso y entidades.
- **Entorno y controladores:** aquí es donde se encuentran los detalles: UI, herramientas, frameworks, etc.

La intención es la separación de responsabilidades manteniendo a las reglas de negocio desconectadas del mundo exterior, y de esta manera, pueden ser testeadas sin ninguna dependencia con componentes externos.

Para conseguir esto, el proyecto se ha dividido en tres capas y cada una tiene un único propósito y funciona independientemente de las otras. Cada capa usará su propio modelo de datos para poder conseguir dicha independencia por lo que serán necesarios *mappers* para transformar los objetos de una capa a los de otra. A continuación se muestra una imagen del esquema:

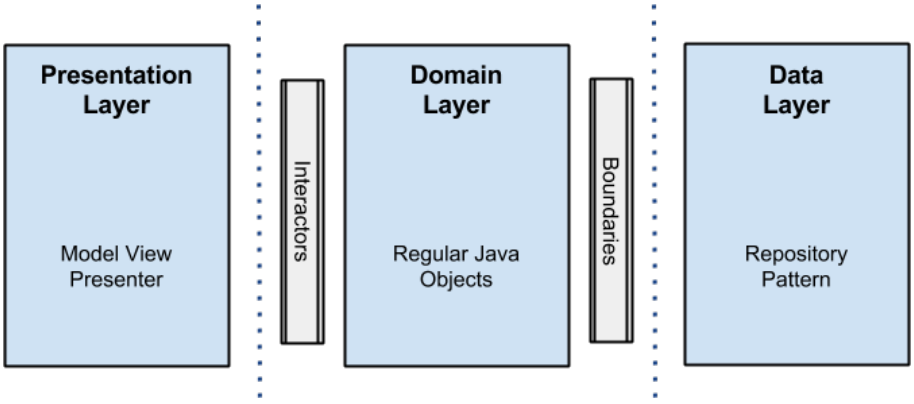

Figura 31: Capas en Clean Architecture [11]

Capa de Presentación

En esta capa es donde irá la lógica relacionada con las vistas y animaciones y se usará el patrón *Model View Presenter* (MVP). Aquí los *Fragments* y Actividades de Android son simples vistas, no contendrán ninguna lógica que no esté relacionada con la interfaz de usuario.

Los *Presenters* estarán compuestos por *UseCases* que realizarán el trabajo en un hilo fuera del hilo principal, y devolverán la información que será mostrada en la vista a través de un *callback*.

Figura 32: Capa de Presentación [11]

Capa de Dominio

Aquí es donde estarán las reglas de negocio, toda la lógica ocurre en esta capa. Además, será donde se encuentren los diferentes casos de uso.

Esta capa es un módulo puro de java y, por lo tanto, no tendrá ninguna dependencia del framework de Android. El resto de componentes externos usarán interfaces para comunicarse con los objetos de negocio.

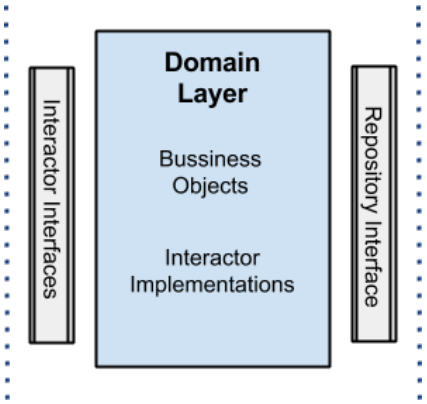

Figura 33: Capa de Dominio [11]

Capa de Datos

Toda la información que necesita la aplicación es proporcionada por esta capa a través de repositorios (cuya interfaz se sitúa dentro de la capa de dominio) y que utiliza el Patrón Repositorio, de tal manera que elegirá una fuente de datos u otra según necesite.

La idea detrás de utilizar ese patrón, es la de hacer que la procedencia de los datos sea transparente para el usuario, de tal manera que no sabrá si la información viene de memoria, disco o la nube, sólo que llegará.

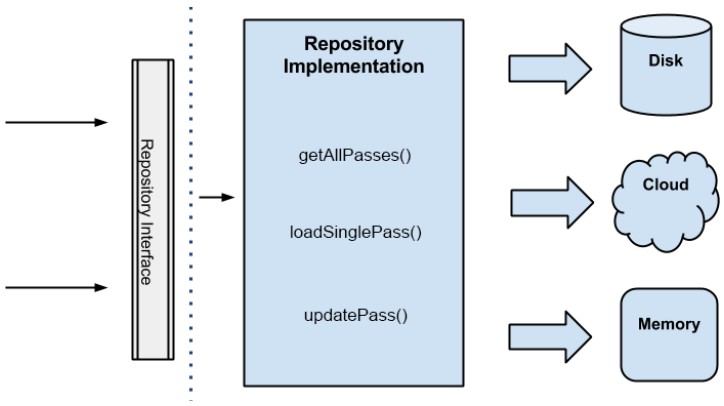

Figura 34: Capa de Datos [11]

Gestión de errores

La estrategia que se ha utilizado es la de usar *callbacks*. Cada *callback* tendrá dos métodos: uno en caso de éxito, que retornará el tipo de datos correspondiente, y otro en caso de error que devolverá un objeto `ErrorBundle`, que encapsulará las excepciones que se han podido producir en las otras capas.

Tests

En lo que respecta a tests, se han utilizado distintas herramientas dependiendo de la capa:

- **Capa de presentación:** se han utilizado los test de instrumentación de Android y *Espresso* para los tests de integración y funcionales.
- **Capa de dominio:** *JUnit* y *mockito* para tests unitarios se utilizan en esta capa.
- **Capa de datos:** *Robolectric*, ya que esta capa tiene dependencias de Android, junto con *mockito* para los tests de integración y unitarios.

4.2. Aplicando Clean Architecture

Siguiendo lo descrito en la sección anterior, en la aplicación se han creado cuatro módulos diferentes:

- **app (Android App):** en este nivel se encontrarán las diferentes vistas, *Presenters* y lógica de inyección de dependencias. El nombre del paquete principal será: `com.rigo.passkit.presentation`.
- **data (Android Library):** aquí se encuentran las diferentes implementaciones de los repositorios y *DataSources*. El nombre del paquete principal será: `com.rigo.passkit.data`.
- **domain (Java):** en este módulo se encuentra la lógica de negocio. Se usarán *UseCases* que utilizarán tantos *DataSources* como sea necesario para obtener la información. El nombre del paquete principal será: `com.rigo.passkit.domain`.
- **pkreader (Java):** librería que se ha desarrollado para este proyecto, que se encargará de la lectura del fichero Passbook y devolverá una representación del mismo con diferentes estructuras de datos. El nombre del paquete principal será: `com.rigo.passkit.pkreader`.

El principal motivo por el que se han nombrado así los diferentes paquetes es porque simplifican la lectura del código y es más rápido ver si se están usando clases que no se deben en una capa en concreto.

La dependencia entre proyectos queda de la siguiente forma:

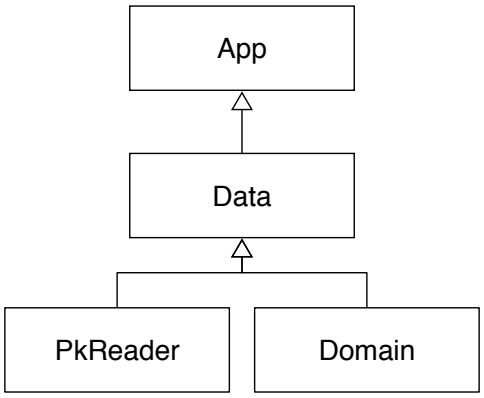

Figura 35: Dependencia entre módulos del proyecto

Cada módulo, excepto **Data**, cuenta a su vez con su propia estructura de objetos por lo tanto, la estructura de objetos del módulo **PkReader** queda como describe la figura a continuación:

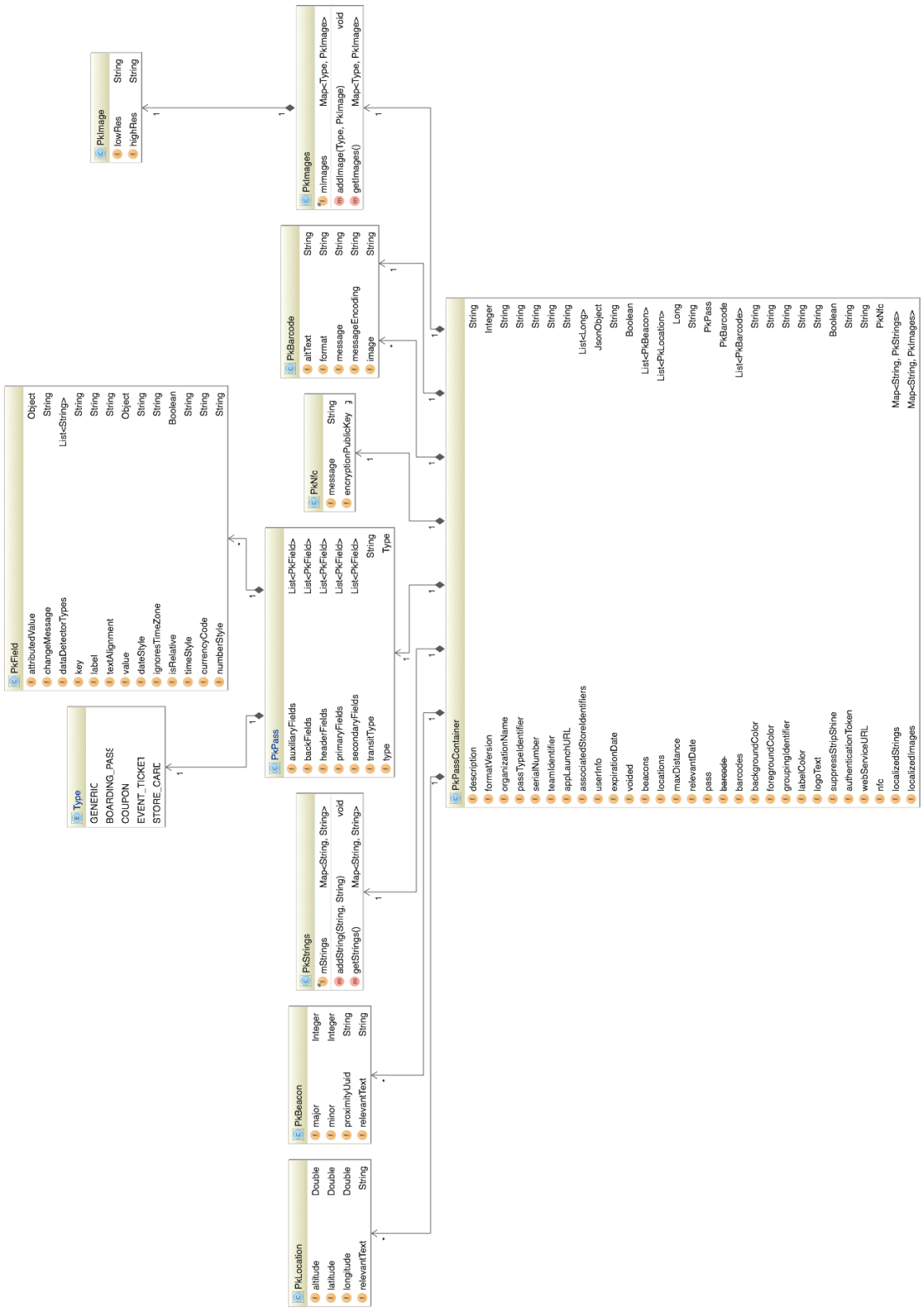

Figura 36: Modelo de datos librería PkReader

La estructura de datos en la capa de **Dominio** se ajusta más a lo descrito anteriormente sobre el formato **Passbook**, como se puede apreciar en la siguiente imagen:

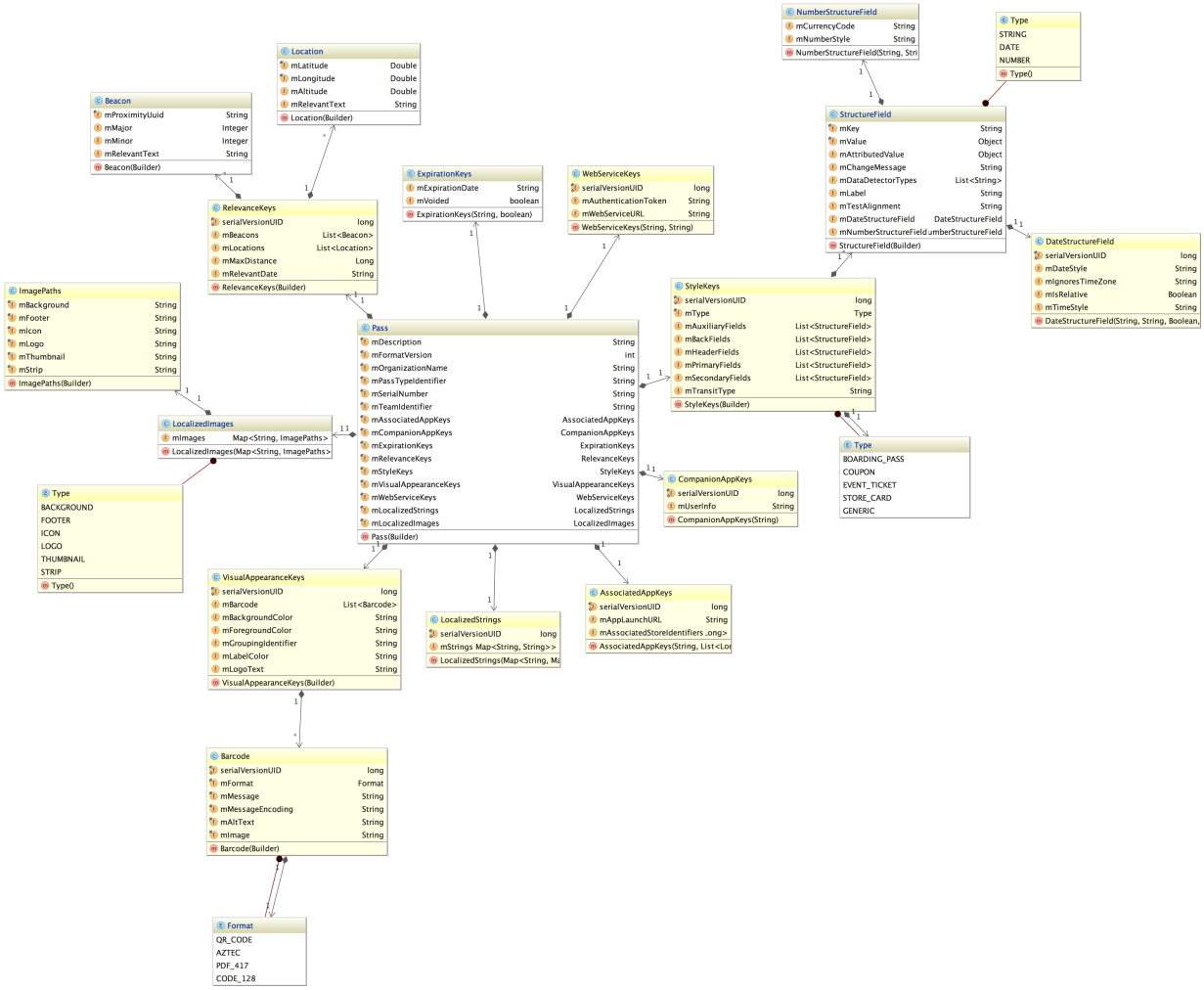

Figura 37: Modelo de datos Domain

Por último tenemos el módulo App. Se va a dividir su estructura de objetos en varias figuras, las relacionadas con la inyección de dependencias y el modelo de vista.

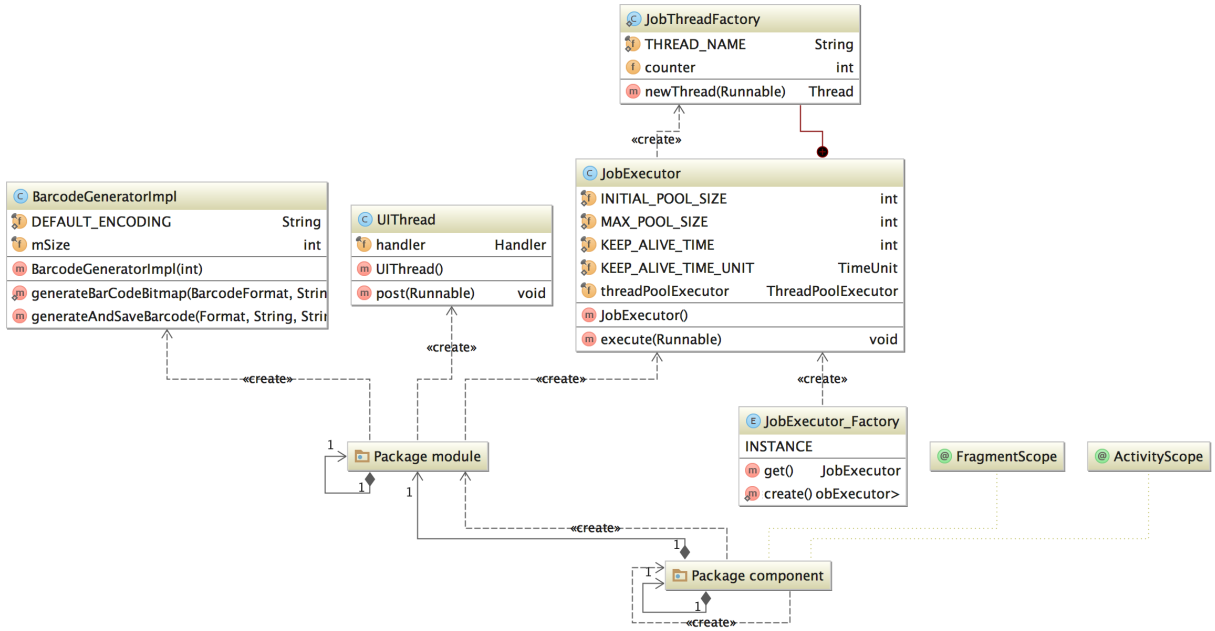

Figura 38: Estructura de clases relacionadas con la inyección de dependencias

A continuación se muestran las diferentes clases contenidas dentro del paquete *component*.

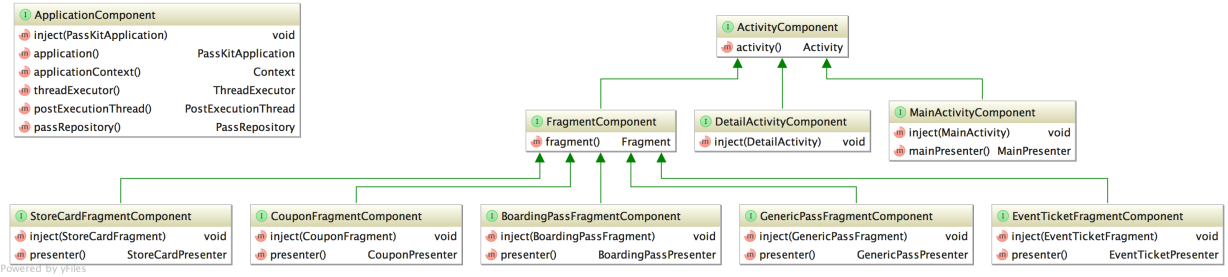

Figura 39: Clases dentro del paquete "component" relacionadas con la inyección de dependencias

Al igual que se ha mostrado el contenido del paquete *component* se hará lo propio con el paquete que contiene todos los módulos.

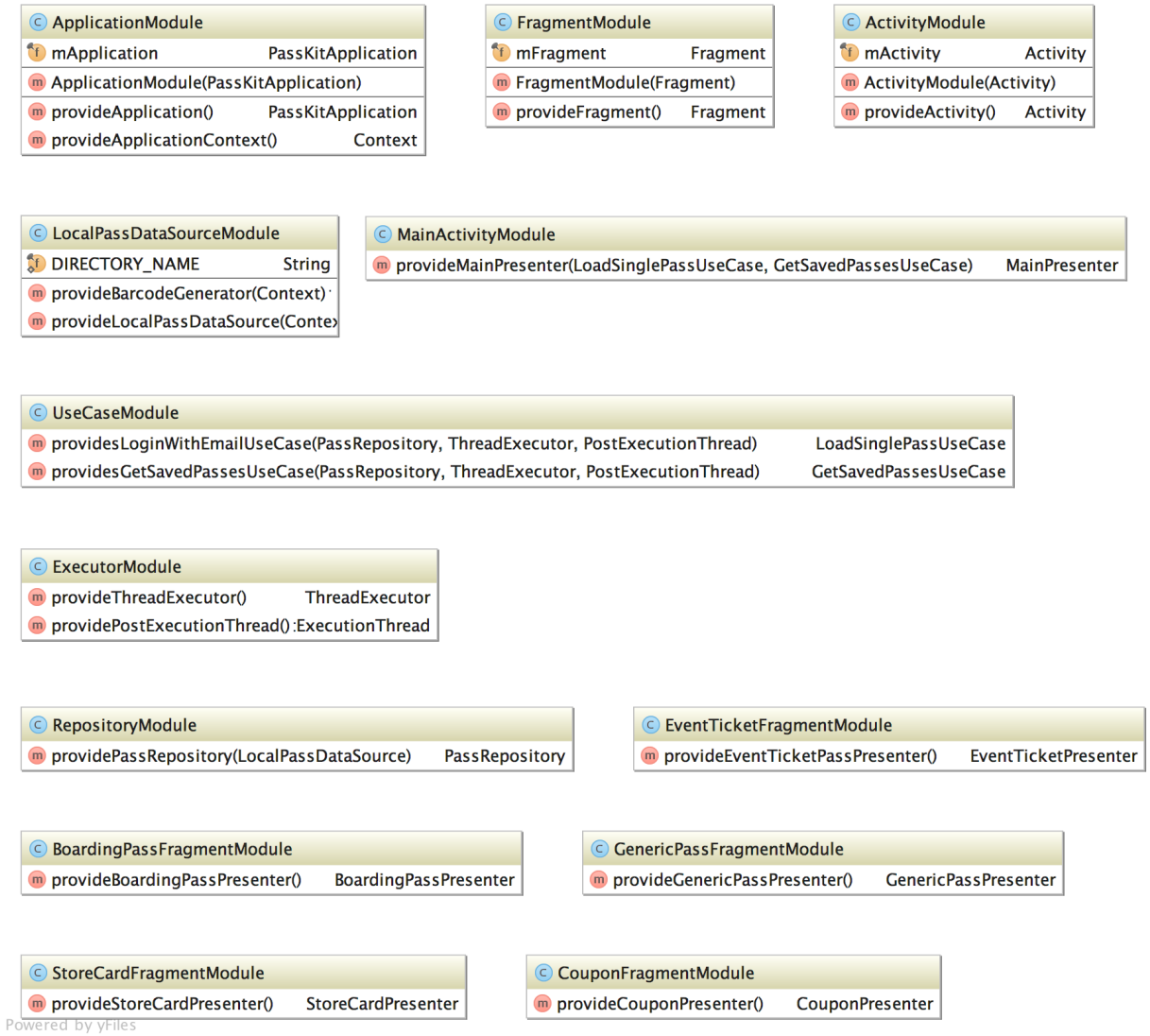

Figura 40: Clases dentro del paquete "module" relacionadas con la inyección de dependencias

El modelo de datos en esta capa está ligado a la vista, por lo tanto los objetos son mucho más sencillos y contienen sólo la información necesaria para mostrar el pase al usuario.

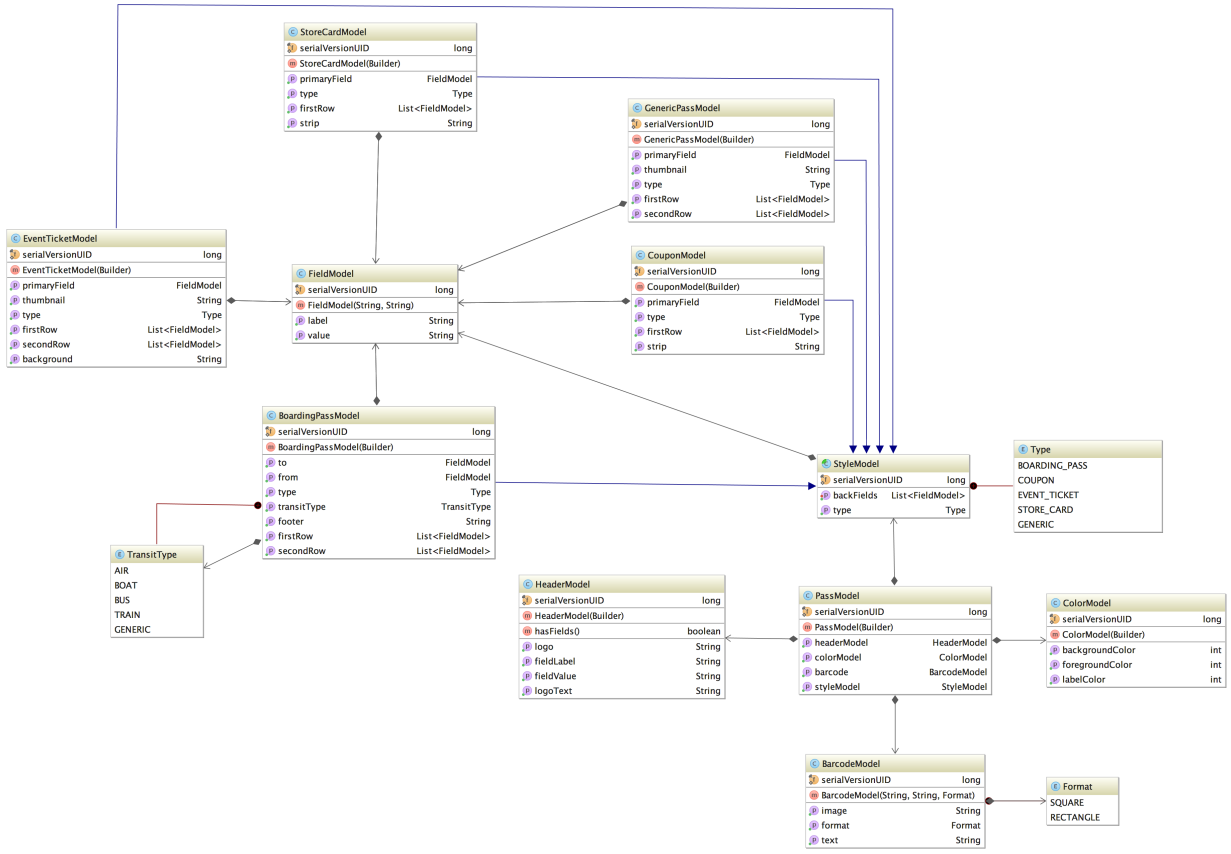

Figura 41: Modelo de datos App

Capítulo 5

Implementación

En este capítulo se procederá a detallar cómo se ha plasmado lo descrito en el capítulo anterior en el desarrollo de la aplicación.

5.1. Ficha técnica de recursos

Antes de explicar cómo se ha llevado a cabo la arquitectura que se acaba de describir, se van a detallar los recursos técnicos que se han empleado para ello.

5.1.1. Gson

También conocida como Google Gson [12], es una biblioteca de código abierto para el lenguaje de programación Java que permite la serialización y deserialización entre objetos Java y su representación en notación JSON.

Entre sus principales características cabe destacar:

- Permite la conversión entre objetos Java y Json de una manera sencilla, simplemente invocando los métodos `toJson()` o `fromJson()`.
- Permite la conversión de objetos inmutables ya existentes.
- Soporte para tipos genéricos de Java.
- Permite la representación personalizada de objetos.
- Soporte para "Objetos arbitrariamente complejos".

A continuación se muestran unos sencillos ejemplos de cómo funciona la librería:

De objeto Java a notación JSON

Objeto Java	Representación JSON

`public class Persona implements Serializable {`
`  private String nombre;`
`  private int edad;`
 	
`  public String getNombre() {`
`  return nombre;`
`  }`
 	
`  public void setNombre(String nombre) {`
`  this.nombre = nombre;`
`  }`
 	
`  public int getEdad() {`
`  return edad;`
`  }`
 	
`  public void setEdad(int edad) {`
`  this.edad = edad;`
`  }`
`}`
	`Persona persona = new Persona();`
	`persona.setNombre("María");`
	`persona.setEdad(28);`
	
	`Gson gson = new Gson();`
	`System.out.println(gson.toJson(persona));`
	
	Resultado:
	
	`{"nombre":"María","edad":28}`

Tabla 62: Ejemplo serialización Gson

De notación JSON a objeto Java

Notación JSON	Objeto Java

`{"nombre":"María","edad":28}`	`Gson gson = new Gson();`
	
	`Persona persona = (Persona) gson.fromJson(json, Persona.class);`
	
	`System.out.println(persona.getNombre());`
	`System.out.println(persona.getEdad());`
	
	Resultado:
	
	`María`
	`28`

Tabla 63: Ejemplo deserialización Gson

5.1.2. Butter Knife

En Android, los diseños se realizan en un archivo XML y después, para poder acceder a los objetos que se han declarado en él, hay que llamar al método `findViewById()` para cada una de las vistas. Cuanto más complejos se vuelven diseños, llamar a este método se vuelve repetitivo y ahí es donde entra en escena Butter Knife [14].

Esta librería tiene anotaciones que ayudan a los desarrolladores a instanciar las vistas definidas en los ficheros XML así como otras otras anotaciones para gestionar ciertos eventos como por ejemplo, un clic sobre la vista (`@OnClick`).

Partiendo del siguiente diseño:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

  <ImageView
      android:id="@+id/my_image_view"
      android:layout_width="match_parent"
      android:layout_height="match_parent"/>
</RelativeLayout>
```

Figura 42: Diseño de ejemplo para Butter Knife

Para poder utilizar ese `ImageView` desde la aplicación habría que hacer lo siguiente:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  ...
  ImageView imageView = (ImageView) findViewById(R.id.my_image_view);
  imageView.setVisibility(View.GONE);
}
```

Figura 43: View binding sin Butter Knife

Lo que propone esta librería es no tener que realizar esa llamada y ese casting por cada vista que se encuentra en el diseño sustituyéndolo por:

```
@Bind(R.id.my_image_view) ImageView mImageView;

@Override
public void onCreate(Bundle savedInstanceState) {
  ...
  mImageView.setVisibility(View.GONE);
}
```

Figura 44: View binding con Butter Knife

5.1.3. Glide

Glide [14] es una librería rápida y eficiente de código abierto para la gestión y descarga de imágenes para Android que engloba la decodificación, cacheo en memoria y disco, y gestión de recursos en una interfaz simple y fácil de usar.

```
@Override
public void onCreate(Bundle savedInstanceState) {
  ...
  ImageView imageView = (ImageView) findViewById(R.id.my_image_view);

  Glide.with(this).load("http://test/image/url").into(imageView);
}
```

Tabla 64: Carga de imagen usando Glide

`ImageView` es un tipo de componente en Android que puede alojar una imagen, por lo tanto en este ejemplo se está indicando que se cargue la imagen `http://test/image/url` en él.

5.1.4. Dagger 2

El concepto de inyección de dependencias fue acuñado por Martin Fowler y consiste en suministrar a una clase los objetos que necesita para su funcionamiento en lugar de que ésta la que cree dichos objetos. Este proceso permite cambiar los objetos suministrados sin que esto implique ningún otro cambio en la clase.

Las librerías de inyección de dependencias llevan años existiendo con una gran variedad de APIs para su configuración e inyección. Dagger 2 [15] es la primera de ellas que lo realiza 100% con código autogenerado. La intención principal es la de generar código que imite el código que habría sido escrito sino para asegurarse de que la inyección de dependencias es simple, trazable y tan rápida como sea posible.

En una aplicación nos encontramos con diferentes tipos de clases, están las clases que realizan funcionalidades y las que ayudan a las anteriores a llevarlas a cabo. Las primeras tienen como dependencia a las segundas por lo que, para crear una instancia de ellas, hay que generar código de inicialización de las diferentes dependencias algo que puede llegar a ser repetitivo y comprometer la legibilidad del código.

Dagger es un remplazo para estas clases factoría que implementa el patrón de inyección de dependencias sin tener que escribir todo ese código repetitivo. Permite al desarrollador centrarse en las clases importantes. Por lo tanto, esta librería nos aporta las siguientes ventajas:

- Dado a que las dependencias pueden ser inyectadas y configuradas externamente, se pueden reutilizar componentes.
- Cuando se inyectan abstracciones como colaboradores, se puede cambiar la implementación de cualquier objeto sin tener que hacer muchos cambios en el código, ya que la instanciación de ese objeto reside en un lugar aislado y desacoplado.
- Las dependencias pueden ser inyectadas en un componente: Es posible inyectar dependencias de prueba o *mocks* que simplifican la realización de tests.

A continuación de definen los diferentes elementos que componen la librería:

- `@Inject`: con esta anotación se solicitan las dependencias. En otras palabras, esta anotación es utilizada por Dagger para saber que esa clase o campo quiere participar en la inyección de dependencias. Por lo tanto, Dagger construirá instancias de estas clases anotadas para satisfacer sus dependencias.
- `@Module`: los módulos con clases cuyos métodos proveen las dependencias, así si se define una clase y se anota con `@Module`, Dagger sabrá dónde encontrar las dependencias para poder satisfacerlas a la hora de construir las instancias de clases.
- `@Provide`: dentro de los módulos, se definen métodos que contienen esta anotación que indican cómo queremos construir y proveer las dependencias.
- `@Component`: los componentes son básicamente los inyectores. Son un puente entre `@Inject` y `@Module`.
- `@Scope`: mecanismo que se encarga de mantener una sola instancia de una clase mientras ese *scope* exista. En la práctica, lo que significa es que las instancias marcadas con `@ApplicationScope` vivirán tanto como lo haga el objeto Aplicación. En resumen, los *scopes* proporcionan "singletons locales".
- `@Qualifier`: se usa esta anotación cuando el tipo de una clase es insuficiente para identificar una dependencia. Por ejemplo, en el caso de Android, muchas veces necesitamos diferentes tipos de contexto, así que podemos crear la anotación `@ForApplication` y `@ForActivity` para que cuando Dagger vaya a inyectarlas sepa si tiene que ser el contexto de una aplicación o de una actividad.

Dado a que es una librería compleja y que tiene una alta curva de aprendizaje, se va a mostrar un pequeño ejemplo para que se pueda ver todo lo descrito anteriormente en funcionamiento.

ApplicationComponent.java

```
@Singleton
@Component(modules = ApplicationModule.class)
public interface ApplicationComponent {
  void inject(MyApplication application);

  // Objetos que podrán ser inyectados
  Context applicationContext();
}
```

Figura 45: Dagger - Componente de ejemplo

ApplicationModule.java

```
@Module
public class ApplicationModule {
  private final MyApplication mApplication;

  public ApplicationModule(MyApplication application) {
    mApplication = application;
  }

  @Provides
  @Singleton
  public MyApplication provideApplication() {
    return mApplication;
  }

  @Provides
  @Singleton
  public Context provideApplicationContext() {
    return mApplication.getApplicationContext();
  }
}
```

Figura 46: Dagger - Módulo de ejemplo

MyApplication.java

```
public class MyApplication extends Application {
  private ApplicationComponent mComponent;

  @Override public void onCreate() {
    super.onCreate();
    mComponent =
        DaggerApplicationComponent.builder().applicationModule(new
        ApplicationModule(this)).build();
  }
}
```

Figura 47: Dagger – Ejemplo de creación de componente

5.1.5. ZXing

ZXing [17] es una librería de código abierto, que procesa múltiples formatos de códigos de barras 1D y 2D e implementada en Java.

Esta será la librería que se utilizará para generar los diferentes códigos de barras de tipo Code 128, PDF 417, QR Code y Aztec, pero estos no son los únicos formatos que soporta.

1D product	1D industrial	2D
UPC-A	Code 39	QR Code
UPC-E	Code 93	Data Matrix
EAN-8	Code 128	Aztec (beta)
EAN-13	Codabar	PDF 417 (beta)
	ITF	
	RSS-14	
	RSS-Expanded	

Tabla 65: Formatos soportados por librería ZXing

5.2. Implementación

La implementación de la aplicación se ha realizado siguiendo los principios de *Clean Architecture*, que ya se explicó en el Capítulo 4, por lo tanto se procederá a describir cómo se han plasmado en la aplicación que se ha desarrollado para la lectura de Passbooks.

5.2.1. Librería PkReader

El nombre viene de "Pass Kit Reader" y será la librería que, pasado un flujo de datos, directorio o fichero, se encargará de transformarlo y convertirlo a una estructura de objetos para facilitar su manejo más adelante.

La clase principal de la librería es la llamada `PkPassLoader`, que será la que se encargue de todo el proceso de cargar un pase. Dicha clase recibe dos parámetros en su constructor, una

cadena de texto que será la ruta al directorio de destino y un objeto `BarcodeGenerator`, que es una interfaz que la librería expone y que utilizará para poder generar los códigos de barras.

El motivo por el que se tiene que exponer la interfaz `BarcodeGenerator` y por la que no se pueden generar los códigos de barras directamente en este módulo es porque, para hacerlo, se necesitan clases Java que no se encuentran presentes en el SDK de Android, por lo que se tiene que delegar esa tarea a quien quiera que vaya a instanciar este objeto.

El método para leer un fichero Passbook recibe un `InputStream` y el proceso es el siguiente:

1. Crear directorio temporal llamado `temp_<current_time_in_millis>`.
2. Descomprimir el archivo `.pkpass` en ese directorio.
3. Crear archivo `.nomedia` para evitar que las imágenes contenidas en el directorio sean indexadas por el escáner de medios de Android.
4. Parsear el archivo `pass.json` usando la librería *Gson*.
5. Al haber parseado el archivo, ya se tiene acceso a los valores de `passTypeIdentifier` y `serialNumber` que, como se describió en anteriores capítulos, son los atributos que conjuntamente identifican unívocamente un pase. Por lo tanto, se renombra la carpeta a `passTypeIdentifier-serialNumber`.
6. Generar y guardar el código *barcode*. Al persistir el código de barras, se evita que este se esté generando cada vez que se quiere mostrar el pase, ya que es un proceso costoso.
7. Obtener las imágenes asociadas a un pase, teniendo en cuenta también aquellas que estén localizadas.
8. Obtener los textos del pase teniendo en cuenta, al igual que en el punto anterior, aquellos que estén localizados.

Al terminar el proceso, toda la información estará contenida en el objeto `PkPassContainer`.

La librería expone los siguientes métodos:

```
public List<PkPassContainer> loadAndParseAllPasses() throws
PkPassLoaderException

public PkPassContainer loadAndParseFile(InputStream inputStream) throws
PkPassLoaderException

public void deletePass(String passTypeIdentifier, String serialNumber)

public String getPassUri(String passTypeIdentifier, String serialNumber, Pass)
```

Por lo tanto, este módulo queda definido de la siguiente forma:

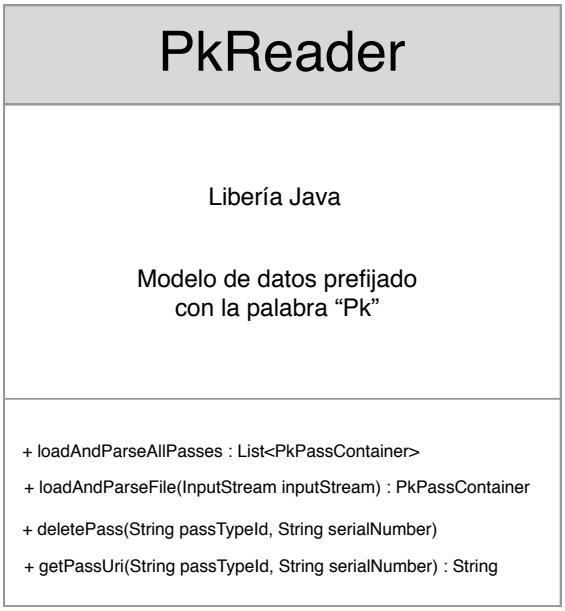

Figura 48: Estructura librería PkReader

El motivo por el que se ha creado una librería de Java pura para la lectura de pases es su posible reutilización en un futuro en cualquier otra aplicación Android o incluso aplicación Java.

5.2.2. Domain

Tal y como se ha explicado con anterioridad, en esta capa se encuentran las reglas de negocio y los diferentes *UseCases* que permitirán obtener la información necesaria. En concreto hay dos:

- `GetSavedPassesUseCase` – Devolverá todos los pases que se encuentran guardados en la memoria del teléfono.
- `LoadSinglePassUseCase` – Cargará y persistirá un nuevo pase en la memoria del teléfono.
- `DeletePassUseCase` – Eliminará un pase de la memoria interna del teléfono.
- `GetPassUriUseCase` – Devolverá la ruta donde se aloja el archivo *.pkpass* en la memoria del teléfono.

Además, en esta capa se expondrán las interfaces de los repositorios que serán implementados en la capa de **Data**. En este proyecto sólo hay un repositorio llamado `PassRepository` que expone los siguientes métodos:

- `loadPass(InputStream inputStream, LoadPassCallback loadPassCallback)`
- `getSavedPasses(GetSavedPassesCallback getSavedPassesCallback)`
- `void deletePass(String passTypeId, String serialNumber, PassDeletedCallback callback)`
- `void getPassUri(String passTypeId, String serialNumber, GetPassUriCallback callback)`

Los *UseCases* se ejecutan siempre en segundo plano y hacen uso de los repositorios que son los que se encargarán de proveer la información, dentro del *UseCase* se tratará dicha información si hiciera falta, y se devolverá en un *callback*.

El hecho de que se produzca el cambio de hilo dentro de los *UseCase* facilita enormemente la gestión de la sincronización en Android, ya que de otra forma se estaría cambiando de contexto continuamente, además, permite realizar todas las tareas de forma serializada sin riesgo de estar bloqueando el hilo principal. Para ello, se mantiene una referencia al hilo principal que será sobre el cual se realizará la llamada al *callback*, y una referencia a un `ThreadPoolExecutor` que será donde se ejecutarán los *UseCase* en segundo plano.

Los los objetos de esta capa se transforman a objetos de la capa de dominio antes de ser devueltos de tal forma que no se expone con qué tipo de objetos se está trabajando en este nivel.

Por lo tanto, al igual que se ha hecho con el módulo anterior, este queda definido de la siguiente forma:

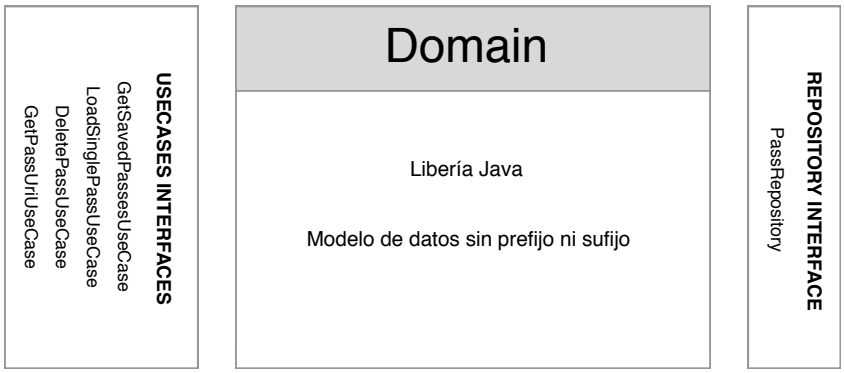

Figura 49: Estructura capa Domain

5.2.3. Data

Aquí es donde se implementará la interfaz del repositorio expuesto por la capa **Domain** llamada `PassRepository`. Esta capa hace de "pegamento" entre las diferentes fuentes de datos o *DataSources* para, como se ha mencionado en párrafos anteriores, evitar que en las capas

superiores se tenga constancia de la procedencia de los datos, pasando a ser esto un mero detalle de implementación.

Este módulo es de tipo librería de Android y también contiene los diferentes convertidores o *mappers* que se encargarán de transformar los objetos retornados por los diferentes módulos a los esperados por otros.

Se define una fuente de datos llamada `LocalPassDataSource` que será un *wrapper* sobre la librería **PkReader**, de esta forma, se podría sustituir la librería sin afectar las demás implementaciones. Todo queda completamente desacoplado.

La implementación de dicho *DataSource* es la siguiente (sólo se muestra el método `loadPass`):

```
public class LocalPassDataSourceImpl implements LocalPassDataSource {

  private final PkPassLoader mPassLoader;

  public LocalPassDataSourceImpl(PkPassLoader passLoader) {
    mPassLoader = passLoader;
  }

  @Override
  public PkPassContainer loadPass(InputStream inputStream)
      throws LocalPassDataSourceException {
    try {
      return mPassLoader.loadAndParseFile(inputStream);
    } catch (PkPassLoaderException e) {
      throw new LocalPassDataSourceException(e);
    }
  }

  . . .
}
```

Figura 50: Extracto implementación LocalPassDataSource

Al final queda una clase bastante sencilla en la que es muy fácil saber qué está haciendo de un solo vistazo.

El constructor de la clase que implementa la interfaz `PassRepository`, recibirá por constructor una instancia del *DataSource* descrito anteriormente y queda como se muestra a continuación:

```java
public class PassRepositoryImpl implements PassRepository {

  private final LocalPassDataSource mLocalPassDataSource;

  public PassRepositoryImpl(LocalPassDataSource localPassDataSource) {
    mLocalPassDataSource = localPassDataSource;
  }

  @Override
  public void loadPass(InputStream inputStream, LoadPassCallback callback) {
    try {
      PkPassContainer pkPassContainer = mLocalPassDataSource.loadPass(inputStream);
      if (callback != null) {
        callback.onPassLoaded(PassMapper.map(pkPassContainer));
      }
    } catch (LocalPassDataSourceException e) {
      if (callback != null) {
        callback.onLoadingPassError(ErrorMapper.map(e));
      }
    }
  }

  . . .
}
```

Figura 51: Extracto implementación PassRepository

Esta implementación hace uso de `LocalPassDataSource` para obtener los datos y posteriormente utiliza un *mapper* llamado `PassMapper` para transformar el objeto `PkPassContainer` devuelto por la librería **PkReader** al objeto de la capa de dominio `Pass`.

También envolverá los errores lanzados en un objeto llamado `ErrorBundle`, que también cuenta con un *mapper* para realizar esta función.

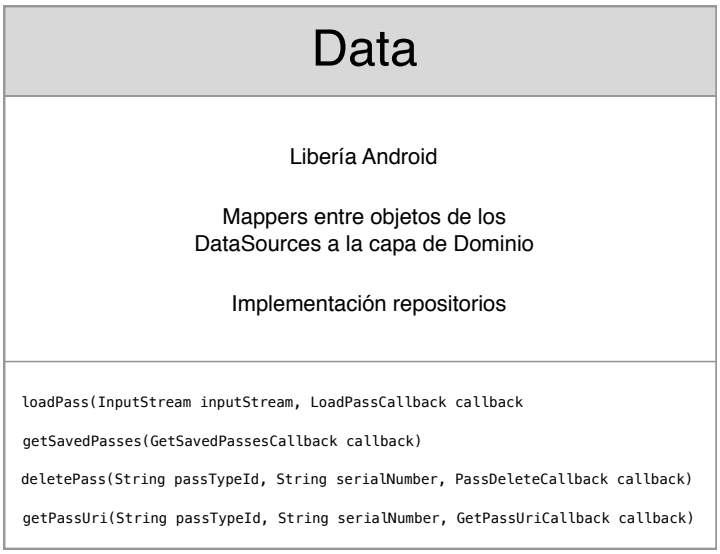

Figura 52: Estructura capa Data

5.2.4. App

Esta capa también se conoce como capa de presentación y en ella se encontrarán todos los objetos y clases relacionados con la vista, además de las inicializaciones necesarias de la librería de inyección de dependencias que se explican a continuación.

La librería que se ha utilizado para realizar la inyección de dependencias es Dagger 2, de código abierto, apoyada por Google y sobre la que se están realizando mejoras de forma continua.

Dentro del paquete `internal.di` se encuentra toda la lógica relacionada con ello, se ha separado por funcionalidad y los componentes y módulos se encuentran en diferentes paquetes (Figura 38, Figura 39 y Figura 40). También se encuentran aquí las implementaciones de las diferentes interfaces que requerirán en sus constructores los diferentes objetos que serán inyectados.

Una vez que se dispone de toda la configuración de Dagger, ya se puede utilizar la anotación `@Inject` para que sea la librería la que se encargue de satisfacer las dependencias, este proceso se ve descrito en los siguientes extractos de código.

```
public class PassKitApplication extends Application {
  private ApplicationComponent mComponent;

  @Override
  public void onCreate() {
    super.onCreate();
    mComponent =
        DaggerApplicationComponent.builder().applicationModule(new
        ApplicationModule(this)).build();
  }

  public ApplicationComponent getComponent() {
    return mComponent;
  }
}
```

Figura 53: Inyección dependencias PassKitApplication

En el clase `PassKitApplication`, se creará el componente autogenerado por la librería llamado `DaggerApplicationComponent` que recibe en su creación una instancia de `ApplicationModule` que es la clase que contienen los métodos que indican cómo se satisfarán esas dependencias. También se expone el método `getComponent` que será utilizado por las diferentes actividades para crear sus componentes ya que tendrán como dependencia dicho al `ApplicationComponent` como se puede observar a continuación.

```
public class MainActivity extends AppCompatActivity
    implements MainView, NavigationView.OnNavigationItemSelectedListener {
  @Inject
  MainPresenter mPresenter;

  . . .

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    . . .

    injectComponent();
  }

  . . .

  private void injectComponent() {
    MainActivityComponent component = DaggerMainActivityComponent.builder()
        .applicationComponent(((PassKitApplication) getApplication()).getComponent())
        .activityModule(new ActivityModule(this))
        .build();
    component.inject(this);
  }
}
```

Figura 54: Inyección dependencias MainActivity

El `MainActivityComponent` expone dos métodos: `inject()`, que será el encargado de inicializar todas las variables anotadas con `@Inject;` y `mainPresenter()` que indica que ese componente es inyectable, tal y como se puede ver en el extracto de código con `@Inject MainPresenter`. A partir de este momento, siempre que queramos utilizar esa variable, esta estará inicializada correctamente.

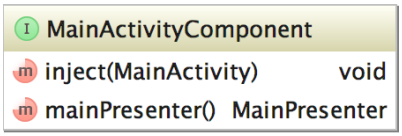

Figura 55: MainActivityComponent

En este módulo, se ha utilizado el patrón *Model View Presenter* (MPV), toda la lógica de negocio se encontrará en el *Presenter* y éste se comunicará con la vista a través de una interfaz.

A continuación se muestra un diagrama de secuencia que muestra cómo es el proceso para cargar un pase y la comunicación entre las diferentes capas. El caso que se muestra a

continuación es de la carga de un solo pase, pero dada la arquitectura de la aplicación, las diferentes operaciones que tengan que ver con la creación, modificación o borrado de pases siguen el mismo patrón.

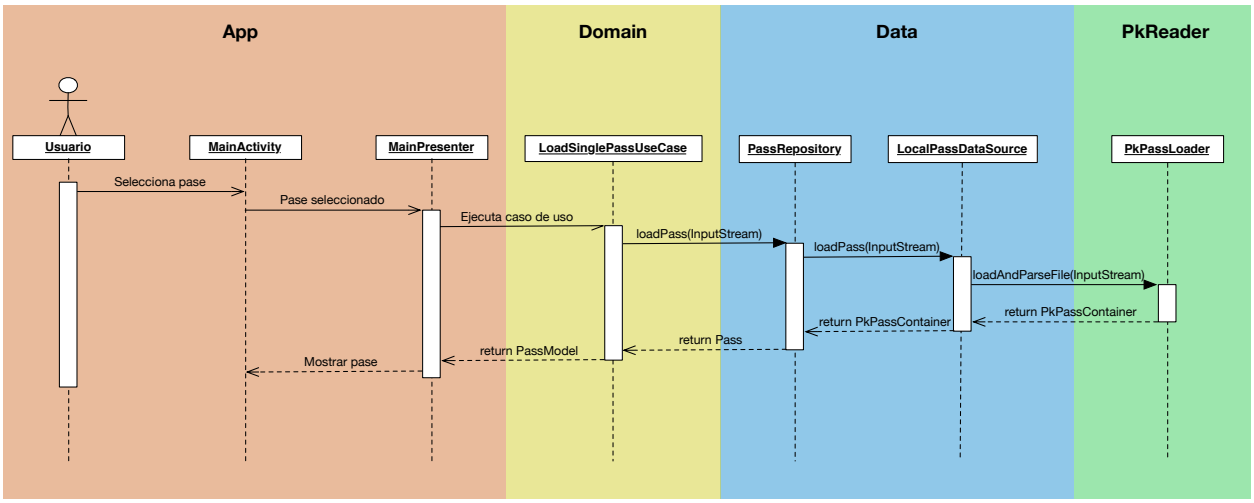

Figura 56: Diagrama secuencia de carga de un pase

5.3. Pruebas

Durante la fase final del proyecto se ha ofrecido la aplicación a diferentes personas a las cuales se les ha pedido su opinión sobre la misma. La aplicación ha sido utilizada satisfactoriamente para realizar viajes en avión, ir al teatro y al cine.

Así mismo la aplicación ha sido probada en los siguientes terminales:

- **Samsung Galaxy S6** con Android 5.1 y 5.1.1
- **Samsung Galaxy Nexus** con Android 4.4
- **HTC One** con Android 5.0
- **Motorola Moto G** (1^a y 2^a generación) ambos con Android 5.1.1
- **LG G3** con Android 5.0
- **Huawei P8** con Android 5.0.1
- **Emuladores** con diferentes configuraciones y versiones.

Requisito	Samsung Galaxy S6	Samsung Galaxy Nexus	HTC One	Motorola Moto G (1st gen)	Motorola Moto G (2nd gen)	LG G3	Huawei P8
RF-1	✓	✓	✓	✓	✓	✓	✓
RF-2	✓	✓	✓	✓	✓	✓	✓
RF-3	✓	✓	✓	✓	✓	✓	✓
RF-4	✓	✓	✓	✓	✓	✓	✓
RF-5	✓	✓	✓	✓	✓	✓	✓
RF-6	✓	✓	✓	✓	✓	✓	✓
RF-7	✓	✓	✓	✓	✓	✓	✓
RF-8	✓	✓	✓	✓	✓	✓	✓
RF-9	✓	✓	✓	✓	✓	✓	✓
RF-10	✓	✓	✓	✓	✓	✓	✓
RF-11	✓	✓	✓	✓	✓	✓	✓
RF-12	✓	✓	✓	✓	✓	✓	✓
RF-13	✓	✓	✓	✓	✓	✓	✓
RF-14	✓	✓	✓	✓	✓	✓	✓
RF-15	✓	✓	✓	✓	✓	✓	✓
RF-16	✓	✓	✓	✓	✓	✓	✓

Tabla 66: Validación requisitos en diferentes terminales

Como se puede observar, la aplicación ha funcionado con éxito en terminales de los principales fabricantes móviles.

5.4. Interfaz

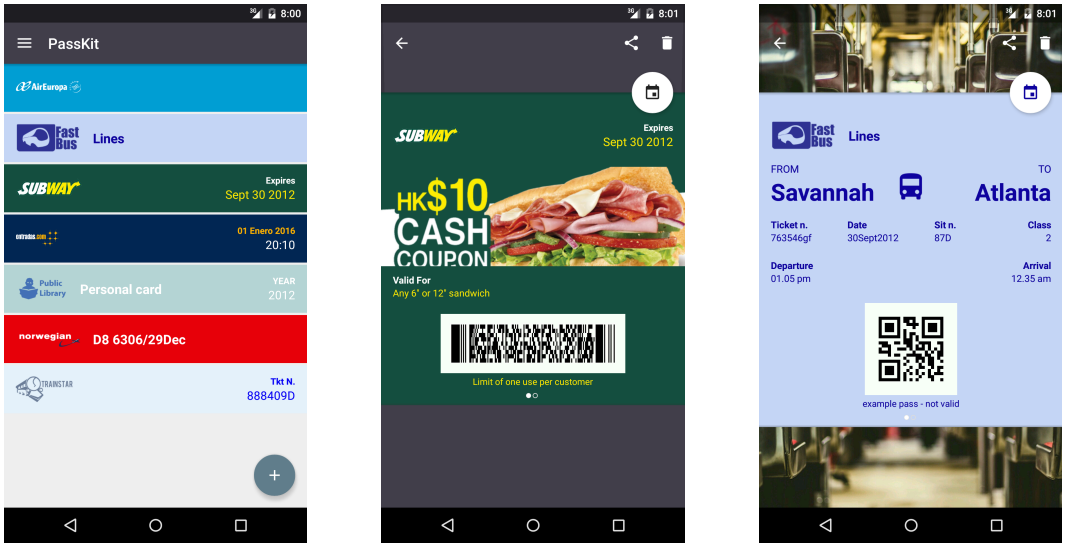

Figura 57: Capturas aplicación lector de ficheros Passbook

La interfaz respeta las reglas de diseño actuales definidas por Google con el nombre de Material Design.

En la primera imagen se muestra el listado de pases que el usuario ha añadido y que han sido persistidos en la memoria del teléfono, además, desde esta pantalla el usuario podrá añadir un nuevo pase pulsando sobre el botón de la parte inferior derecha.

Al pulsar sobre uno de esos pases, o tras añadir un pase nuevo, se mostrará al usuario la pantalla de detalle del pase. La imagen central se trata de un cupón mientras que la imagen de la derecha es una tarjeta de embarque.

El usuario dispone de varias opciones en esta pantalla, en la barra superior se encuentran las opciones de compartir y eliminar un pase mientras que si el usuario pulsa sobre el botón con forma circular, se añadirá un evento a su calendario. Asimismo, al pulsar sobre el código de barras, este podrá ser visto a pantalla completa y si el usuario desliza hacia la izquierda, podrá acceder a los campos secundarios.

Capítulo 6

Gestión del proyecto

En este apartado se describirá la planificación seguida para la elaboración del proyecto. Se ha de tener en cuenta que la dedicación al proyecto no ha sido exclusiva, ya que de forma simultánea ya que se han desarrollado otras actividades que no han permitido dedicar todos los esfuerzos al mismo.

6.1. Planificación

El sistema se ha desarrollado siguiendo la metodología de desarrollo en cascada y se ha dividido en las siguientes tareas:

- Requisitos
- Análisis
    - Análisis de requisitos
    - Casos de uso
- Diseño
    - Diseño de arquitectura
    - Diseño de clases
- Implementación
    - Codificación
    - Documentación
    - Pruebas

El proyecto se ha comenzado el 20 de Julio y está planificado para terminarlo a finales de Diciembre. En total son algo más de 5 meses de trabajo con una dedicación media de 3 horas al día.

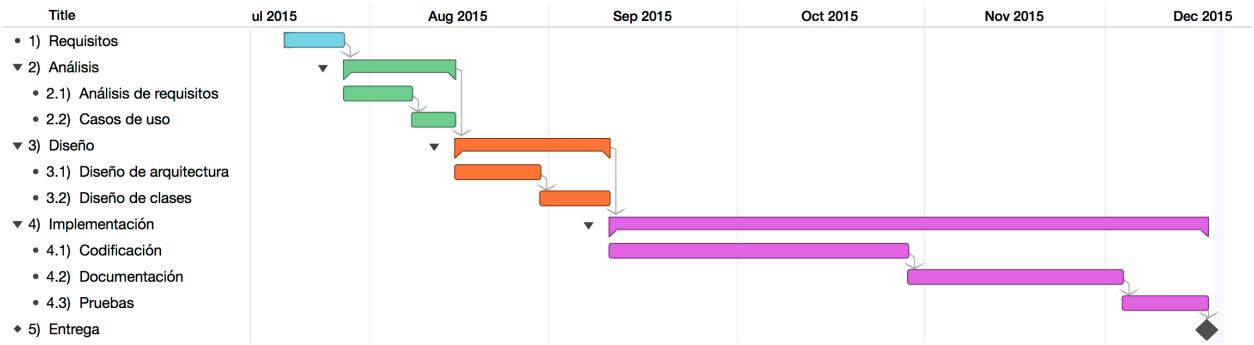

Figura 58: Planificación del proyecto

Con esta planificación se dispone de los datos necesarios para realizar el presupuesto del proyecto.

6.2. Presupuesto

En este apartado se encuentra detallado el presupuesto para el desarrollo de la aplicación. Se han desglosado los costes de la siguiente forma:

- Recursos humanos
- Material
- Gastos indirectos

La unidad que se ha utilizado es el Euro (€) y en todos los cálculos se utilizará el redondeo con dos decimales.

6.2.1. Recursos humanos

En este apartado se presupuesta la carga económica a soportar en personal dedicado al proyecto.

Para realizar este cálculo, hay que contabilizar las horas que se han dedicado a la realización de la aplicación. Teniendo en cuenta que el proyecto se inició el 20 de Julio y acabó el 16 de Diciembre (130 días) y que se dedicaron, de media, tres horas cada día, el número de horas totales asciende a 390.

Sabiendo las horas totales dedicadas, se procede a calcular los costes asociados a los recursos humanos. En la siguiente tabla se muestran los costes asociados al personal teniendo en cuenta las horas totales dedicadas y asumiendo un coste de 40€/hora de un Ingeniero Senior.

Personal	**Categoría**	**Coste persona (€/hora)**	**Horas dedicadas**	**Coste (€)**
Alicia Rivas Pérez	Ingeniero Senior	40,00	390	15.600,00

Tabla 67: Costes personal

El coste total en cuanto a personal dedicado al proyecto, sin incluir impuestos, asciende a QUINCEMIL SEISCIENTOS EUROS.

6.2.2. Material

La realización del proyecto ha requerido el uso de diferentes materiales. En este apartado se desglosará el coste por cada herramienta utilizada teniendo en cuenta el tiempo de uso del mismo así como su precio y periodo de amortización.

Una vez desglosados todos los materiales utilizados, se realizará el cálculo del coste total utilizando la siguiente fórmula.

$$Coste = Coste\ equipo\ x\ Porcentaje\ uso\ x\ \frac{Número\ meses\ dedicación}{Periodo\ amortización}$$

En la siguiente tabla se muestran las diferentes herramientas utilizadas distinguiendo entre materiales hardware y software:

Material	**Tipo**	**Precio (€)**	**Porcentaje Uso**	**Dedicación (Meses)**	**Amortización (Meses)**	**Coste (€)**
MacBook Pro Retina 15" i7 2,5GHz con 16GB de RAM	HW	2.799,00	100	4,3	60	200,60
Samsung Galaxy S6	HW	799,00	80	3,44	50	43,98
Microsoft Office	SW	149,00	40	1,72	48	2,14
TOTAL						**252,72**

Tabla 68: Coste de material

De esta forma y tal y como se puede observar en la tabla, los costes de material ascienden a DOSCIENTOS CINCUENTA Y DOS CON SETENTA Y DOS CÉNTIMOS.

6.2.3. Gastos indirectos

En esta sección se detallarán los costes indirectos del proyecto que incluyen los gastos de Internet y luz.

Concepto	**Coste/mes (€)**	**Meses dedicación**	**Coste (€)**
Internet	45	4,3	193,50
Luz	40	4,3	172,00
TOTAL			365,50

Tabla 69: Gastos indirectos

En este caso los gastos suponen TRESCIENTOS SESENTA Y CINCO CON CINCUENTA CÉNTIMOS.

6.2.4. Coste total

Una vez calculados los diferentes gastos imputables a la realización de la aplicación, se procede a computar el coste total del proyecto.

Concepto	**Importe (€)**
Coste personal	15.600,00
Coste material	252,72
Gastos Indirectos	365,50
Total (Sin IVA)	16.218,22
TOTAL (IVA 21%)	**19.624,05**

Tabla 70: Coste total

Por lo tanto, tal y como se puede observar en la tabla, el coste total es de DIECINUEVE MIL SEISCIENTOS VEINTICUATRO CON CINCO CÉNTIMOS.

Capítulo 7

Entorno de desarrollo

El entorno de desarrollo está formado por un entorno de desarrollo integrado IDE y por otras herramientas extra como pueden ser aquellas de control de versiones, máquinas virtuales para pruebas, librerías para realizar tests, etc.

Las herramientas que se han empleado para el desarrollo de la aplicación han sido:

- Android Studio
- JDK
- Gradle

Para las pruebas:

- JUnit 4
- Espresso
- Robolectric

Para el control de versiones:

- Git

En las siguientes secciones se procederá a describir cada una de las herramientas.

7.1. Android Studio

Android Studio [20] es el IDE oficial para el desarrollo de aplicaciones Android, basado en IntelliJ IDEA [19]. Además de las funcionalidades que ofrece IntelliJ, Android Studio ofrece, entre otras cosas:

- Sistema de construcción flexible basado en Gradle.
- Diferentes configuraciones de construcción y generación de múltiples archivos *apk*.
- Plantillas de código que facilitan la creación de tareas comunes.
- Potente editor de diseño con soporte para la creación de temas arrastrando y soltando elementos.
- Herramientas *lint* [22] que miden el rendimiento, usabilidad, compatibilidad de versiones, y otros problemas.
- *ProGuard* [21] y opciones para firmar la aplicación.
- Soporte integrado para *Google Cloud Platform* [24].

La utilización de Android Studio supuso una gran mejora frente al previo IDE oficial, basado en Eclipse [25].

Las versiones que se han utilizado para este proyecto son la 1.5 y 2.0.

7.2. JDK

El Java Development Kit [24] de Oracle [25] es una serie de herramientas para la realización de aplicaciones para Java. La versión que se ha utilizado es Java 7.

De todas las utilidades que contiene, cabe destacar:

- **appletviewer:** visor de applets para generar sus vistas previas, ya que un *applet* carece de método *main* y no se puede ejecutar con el programa Java.
- **javac:** compilador de Java.
- **java:** intérprete de Java.
- **javadoc:** genera la documentación de las clases de un programa.

7.3. Gradle

Gradle [28] es una herramienta para automatizar el proceso de construcción del proyecto (compilación, testing, dependencias, empaquetado,…). Está basado en Groovy [27], un lenguaje de programación orientado a objetos con una sintaxis muy parecida a la de Java.

Gradle fue diseñado para proyectos multi-módulo que pueden llegar bastante grandes, y soporta construcciones incrementales al ser capaz de determinar qué partes de la misma están actualizadas y cuales no.

Otra funcionalidad muy interesante es la de gestión de dependencias ya que facilita el proceso al no tener que estar trabajando con archivos *.jar* directamente, sino que se indicará cuál es el id del artefacto, id de grupo y número de versión y será Gradle quien se ocupe de descargarse la dependencia si no se encuentra en nuestro repositorio local.

7.4. JUnit 4

JUnit [28] es un conjunto de clases que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

JUnit es también un medio de controlar las pruebas de regresión necesarias cuando una parte del código ha sido modificada y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación.

7.5. Espresso

Entorno de pruebas incluido en la librería de soporte de testing y que proporciona un API para escribir test para comprobar que la interfaz de usuario funciona correctamente

permitiendo similar interacciones de usuario dentro de una aplicación. Los tests de Espresso [28] pueden ejecutarse en dispositivos con Android 2.2 (nivel API 8) y superiores.

Un beneficio clave de usar Espresso es que proporciona sincronización automática de las acciones de los tests con la interfaz de usuario mientras el hilo principal está desocupado, siendo capaz de ejecutar los comandos de los tests en el momento apropiado, mejorando la fiabilidad de los mismos.

7.6. Robolectric

Ejecutar los tests en el emulador de Android es lento. Construir, desplegar, y lanzar la aplicación en ocasiones tarda varios minutos. Robolectric [30] nace con la intención de ejecutar esos tests directamente en la máquina virtual de Java, reduciendo así el tiempo de minutos a segundos.

7.7. Git

Git [30] es un software libre de control de versiones distribuido diseñado para manejar tanto grandes como pequeños proyectos con velocidad y eficiencia. Sus principales características son:

- **Crear ramas y mezclarlas:** Git permite la creación de múltiples ramas locales que pueden ser completamente independientes. La creación, mezcla y eliminación de esas líneas de código lleva segundos por lo que se pueden hacer cosas como cambios de contexto, desarrollo basado en funcionalidades, realizar código experimiental, etc. sin afectar el código principal.
- **Pequeño y rápido:** con Git, prácticamente todas las operaciones se hacen en local, lo que proporciona una gran ventaja frente a sistemas centralizados que tienen que estar comunicándose continuamente con un servidor.
- **Distribuido:** al descargar el código por primera vez, en lugar de descargarse sólo la última copia, se descarga todo el repositorio, lo que significa que, aunque se estuviera usando un flujo centralizado, cada usuario tendría una copia completa del servidor principal.

- **Seguridad:** el modelo de datos que usa Git asegura la integridad criptográfica de todo el proyecto ya que se realiza un *checksum* de cada archivo y cada subida o *commit* que se realiza.

Para este proyecto se ha utilizado el servidor en remoto BitBucket [31] para alojar el repositorio en la nube.

Capítulo 8

Conclusiones

El estudio realizado a lo largo de este trabajo ha mostrado cómo Passbook se está posicionando cada vez más como el reemplazo perfecto para todos esos billetes y tarjetas que utilizamos en el día a día. No hay más que ver cómo cada vez más empresas lo ofrecen como sustituto al papel (Iberia, AirEuropa, Norwegian, Yelmo, etc). Sin embargo, al tratarse de un tipo de archivos definido por Apple, el formato es bastante estricto que hace difícil innovar y mostrar la información de una forma muy diferente a la establecida. Por ello, todas las aplicaciones Android con funcionalidad para la gestión de Passbook existentes hasta el momento mostraban una estética y funcionamiento que seguía claramente el modelo del sistema operativo competidor, iOS, rompiendo así las directrices establecidas para aplicaciones Android.

Con el propósito de cambiar este hecho, se propuso una aplicación de lectura de archivos Passbook para dispositivos Android que, siguiendo las directrices y funcionamiento esperado de las aplicaciones de dicho sistema operativo, cubre las necesidades y expectativas planteadas inicialmente, dando a sus usuarios la capacidad de visualizar y persistir este tipo de archivos de una forma nativa, intuitiva y con un diseño adaptado a los patrones establecidos por Android.

El sistema completo incluye el diseño e implementación de una aplicación intuitiva, cumpliendo las directrices de diseño de Android, y una librería propia, además del uso e integración de numerosas librerías de terceros. Por consiguiente, y tras comprobar que las pruebas realizadas en los diferentes terminales han demostrado que la aplicación funciona correctamente en todos ellos y ha podido ser utilizada con éxito como cartera virtual, se puede concluir que todos los objetivos planteados en la introducción se han cumplido satisfactoriamente.

La utilización de Clean Architecture para realizar la aplicación ha servido para afianzar los conocimientos que ya tenía sobre esta forma de estructurar los proyectos y ver su gran potencial. Cuando empiezas en una tecnología tiendes a imitar lo que su creador comparte y, desde los recursos existentes en las páginas de desarrolladores de Android, se presentaban clases muy acopladas y en las que claramente no se aplicaban los principios S.O.L.I.D, por lo que el uso cada vez más extendido de este tipo de arquitecturas demuestra lo mucho que está avanzando la comunidad de Android. Asimismo, ha resultado muy interesante saber cómo funciona en profundidad un formato tan usado como Passbook y la realización de una librería genérica para aplicaciones Java ya que, aunque existe una librería para iOS y MacOS, no es así para Java y por lo tanto Android.

Capítulo 9

Trabajos futuros

Se han identificado las siguientes líneas de mejora en la aplicación:

Agrupar pases relacionados

El archivo `pass.json`, que describe los pases, contiene un parámetro que permite agrupar pases que están relacionados, como pueden ser diferentes entradas al cine para la misma sesión, vuelos con escala, etc. Al mantener los pases agrupados, al usuario le resultará mucho más fácil localizarlos y, en el caso de ser diferentes entradas, agilizará su lectura por el lector de código de barras.

Los pases que están relacionados entre sí siempre aparecerán juntos y una vez en el detalle, podrá desplazarse horizontalmente entre ellos.

Actualizar pases

Los pases también cuentan con información para poder actualizarse, por ejemplo, en el caso de un vuelo, los pases pueden ser actualizados para reflejar cambios en la puerta de embarque, retrasos, etc. Dentro del archivo pass.json, viene toda la información necesaria para realizar estas actualizaciones.

También se le podría dar al usuario la opción de si desea o no actualizar un pase, si sólo desea realizarlo cuando disponga de conexión WiFi, si desea recibir notificaciones push cuando se produzcan cambios, etc.

Mostrar localización del pase

Algunos pases vienen con las coordenadas de longitud y latitud que resultan relevantes para el pase en cuestión como podrían ser la dirección del aeropuerto, cine, teatro, gimnasio, centro comercial, etc.

Una vez mostrado el mapa, el usuario podría tener la opción de pedir las indicaciones para llegar a dicho destino, en cuyo caso se abriría la aplicación de Google Maps enviando como destino las coordenadas anteriormente mencionadas. A partir de ahí, el usuario podrá elegir si quiere indicaciones para ir en coche, transporte público o andando.

Agrupar pases por tipo

En el menú lateral de la aplicación se podrían mostrar las secciones "Tarjeta de embarque", "Cupón", "Tarjeta de fidelización", "Entrada a evento" o "Genérica" que permitirían al usuario filtrar rápidamente los pases de los que dispone.

Archivar pases

Sería interesante poder darle al usuario la opción de archivar pases en lugar de eliminarlos. Esto le permitiría mantener su listado principal libre de pases que pueden haber expirado pudiendo al mismo tiempo mantener un registro de todos los pases que ha utilizado.

Añadir pases existentes en el dispositivo

Al añadir la opción de explorar la memoria interna del teléfono y añadir los pases existentes, se le facilitaría al usuario la opción de migrar toda su información a la aplicación. Mejorando así la experiencia de usuario.

Definiciones, acrónimos y abreviaturas

Aquí se reflejan los distintos términos junto con su definición, acrónimos y abreviaturas que serán utilizadas a lo largo del documento.

Término	Definición
Sistema Operativo	Un sistema operativo (SO) es el programa o conjunto de programas que efectúan la gestión de los procesos básicos de un sistema informático y permite la normal ejecución del resto de las operaciones.
Android	Sistema operativo con núcleo Linux desarrollado por Google.
iOS	Sistema operativo móvil de la multinacional Apple Inc.
Java	Lenguaje de programación multiplataforma orientado a objetos
MacOS	Sistema operativo de ordenadores Apple

Tabla 71: Definición de términos del documento

Acrónimo	Definición
XML	**Ex**tensibe **M**arkup **L**anguage (Lenguaje de Marcas Extensible)
JSON	**J**ava**S**cript **O**bject **N**otation

	(Notación de Objetos JavaScript)
JAR	**J**ava **Ar**chive (Archivo java)
APK	**A**pplication **P**ac**k**age (Archivo ejecutable de Android)
SDK	**S**oftware **D**evelopment **K**it (Herramientas de desarrollo software)
GPS	**G**lobal **P**ositioning **S**ystem (Sistema de posicionamiento global)
JNI	**J**ava **N**ative **I**nterface (Interfaz nativa de java)
API	**A**pplication **P**rogramming **I**nterface (Interfaz de programación de aplicaciones)
SHA-1	**S**ecure **H**ash **A**lgorithm-1 (Algoritmo de hash seguro)
PKCS	**P**ublic **K**ey **C**ryptography **S**tandard (Estándar de criptografía de clave pública)
DNS	**D**omain **N**ame **S**ystem (Sistema de nombres de dominio)
ISO	**I**nternational **S**tandards **O**rganization (Organización de estándares internacionales)
MPV	**M**odel **V**iew **P**resenter (Modelo Vista Presentador)

Tabla 72: Acrónimos del documento

Referencias

[1] Android Developers, «Dashboards» [En línea]. Disponible: http://developer.android.com/intl/es/about/dashboards/index.html. [Último acceso: 4 Noviembre 2015].

[2] International Data Corporation, «Worldwide Smartphone Growth Expected to Slow to 10.4% in 2015, Down From 27.5% Growth in 2014, According to IDC» [En línea]. Disponible: http://www.idc.com/getdoc.jsp?containerId=prUS25860315. [Último acceso: 1 Noviembre 2015].

[3] Apple Developer Support, «App Store - Distribution» [En línea]. Disponible: https://developer.apple.com/support/app-store/. [Último acceso: 21 Agosto 2015].

[4] Statista, «Number of apps Disponible in leading app stores as of July 2015» [En línea]. Disponible: http://www.statista.com/statistics/276623/number-of-apps-Disponible-in-leading-app-stores/. [Último acceso: 26 Octubre 2015].

[5] Top 500. The List, «List Statistics per Operating System Family» [En línea]. Disponible: http://www.top500.org/statistics/list/. [Último acceso: 26 Octubre 2015].

[6] Wallet Developer Guide, «Introducing Wallet» [En línea]. Disponible: https://developer.apple.com/library/watchos/documentation/UserExperience/Conceptual/PassKit_PG/index.html#//apple_ref/doc/uid/TP40012195-CH1-SW1. [Último acceso: 21 Noviembre 2015].

[7] Apple Developer, «Wallet Developer Guide - Pass Design and Creation» [En línea]. Disponible: https://developer.apple.com/library/watchos/documentation/UserExperience/Conceptual/Pas

sKit_PG/Creating.html#//apple_ref/doc/uid/TP40012195-CH4-SW1. [Último acceso: 1 Noviembre 2015].

[8] Google Play, «PassWallet» [En línea]. Disponible: https://play.google.com/store/apps/details?id=com.attidomobile.passwallet. [Último acceso: 27 Noviembre 2015].

[9] Google Play, «Pass2U» [En línea]. Disponible: https://play.google.com/store/apps/details?id=com.passesalliance.wallet. [Último acceso: 27 Noviembre 2015].

[10] Google Play, «PassAndroid» [En línea]. Disponible: https://play.google.com/store/apps/details?id=org.ligi.passandroid&hl=es. [Último acceso: 27 Noviembre 2015].

[11] R. C. Martin, Clean code: A Handbook of Agile Software Craftsmanship, PRENTICE HALL, 2008.

[12] F. Cejas, «Architecting Android the Clean way» [En línea]. Disponible: http://fernandocejas.com/2014/09/03/architecting-android-the-clean-way/. [Último acceso: 28 Octubre 2015].

[13] Google, «GitHub - Gson: A Java serialization library that can convert Java Objects into JSON and back» [En línea]. Disponible: https://github.com/google/gson. [Último acceso: 20 Noviembre 2015].

[14] J. Wharton, «Butter Knife - Field and method binding for Android views.» [En línea]. Disponible: http://jakewharton.github.io/butterknife/. [Último acceso: 2 Diciembre 2015].

[15] Glide, «GitHub - Glide: An image loading and chaching library for Android focused on smooth scrolling.» [En línea]. Disponible: https://github.com/bumptech/glide. [Último acceso: 19 Noviembre 2015].

[16] Google, «Dagger 2 - Dependency injection library» [En línea]. Disponible: http://google.github.io/dagger/. [Último acceso: 20 Noviembre 2015].

[17] ZXing Team, «GitHub - Zxing: Barcode processing library» [En línea]. Disponible: https://github.com/zxing/zxing. [Último acceso: 18 Noviembre 2015].

[18] Google, «Android Studio Overview» [En línea]. Disponible: http://developer.android.com/intl/es/tools/studio/index.html. [Último acceso: 15 Noviembre 2015].

[19] JetBrains, «IntelliJ IDEA» [En línea]. Disponible: https://www.jetbrains.com/idea/. [Último acceso: 20 Noviembre 2015].

[20] Android Developers, «Lint» [En línea]. Disponible: http://developer.android.com/intl/es/tools/help/lint.html. [Último acceso: 8 Diciembre 2015].

[21] ProGuard, «ProGuard» [En línea]. Disponible: http://proguard.sourceforge.net/. [Último acceso: 12 Diciembre 2015].

[22] Google, «Google Cloud Platform» [En línea]. Disponible: https://cloud.google.com/tools/android-studio/app_engine/add_module. [Último acceso: 15 Noviembre 2015].

[23] Eclipse Foundation, «Eclipse» [En línea]. Disponible: https://eclipse.org/. [Último acceso: 19 Noviembre 2015].

[24] Oracle, «Java SE Overview» [En línea]. Disponible: http://www.oracle.com/technetwork/java/javase/overview/index.html. [Último acceso: 12 Diciembre 2015].

[25] Oracle, «Oracle» [En línea]. Disponible: http://www.oracle.com/index.html. [Último acceso: 12 Diciembre 2015].

[26] Gradle Inc, «Gradle» [En línea]. Disponible: http://gradle.org/. [Último acceso: 5 Diciembre 2015].

[27] Groovy project, «Groovy» [En línea]. Disponible: http://www.groovy-lang.org/. [Último acceso: 5 Diciembre 2015].

[28] JUnit, «JUnit: simple framework to write repeatable tests» [En línea]. Disponible: http://junit.org/. [Último acceso: 20 Noviembre 2015].

[29] Android Developers, «Testing UI for a Single App» [En línea]. Disponible: http://developer.android.com/intl/es/training/testing/ui-testing/espresso-testing.html. [Último acceso: 10 Noviembre 2015].

[30] Robolectric, «Robolectric: Test-drive your Android code» [En línea]. Disponible: http://robolectric.org/. [Último acceso: 10 Noviembre 2015].

[31] Git, «Git» [En línea]. Disponible: https://git-scm.com/. [Último acceso: 19 Noviembre 2015].

[32] Atlassian, «BitBucket» [En línea]. Disponible: https://bitbucket.org/. [Último acceso: 20 Diciembre 2015].