



**Sistema para máquinas de vending en Conciertos**

**Proyecto de final de curso**

**Autor:** Sergio Martínez Aznar

**Grado Superior:** Desarrollo de Aplicaciones Multiplataforma

**Curso Académico 2020-2021**

**Contenido**

[1. RESUMEN 5](#_Toc72692480)

[2. ABSTRACT 6](#_Toc72692481)

[3. INTRODUCCIÓN 7](#_Toc72692482)

[3.1. Motivación 7](#_Toc72692483)

[3.2. Objetivos 7](#_Toc72692484)

[3.3. ¿Qué es el Internet de las Cosas? 8](#_Toc72692485)

[4. ANALISIS 9](#_Toc72692486)

[4.1. Objetivos Específicos 9](#_Toc72692487)

[4.1.1. Realizados 9](#_Toc72692488)

[4.1.2. Deseables (Futuras mejoras) 9](#_Toc72692489)

[4.2. Requisitos Generales: 10](#_Toc72692490)

[4.2.1. Funcionales 10](#_Toc72692491)

[4.2.2. No Funcionales 12](#_Toc72692492)

[4.3. Tecnologías y herramientas utilizadas 13](#_Toc72692493)

[4.4. Casos de uso 15](#_Toc72692494)

[5. PLANIFICACION TEMPORAL 16](#_Toc72692495)

[5.1. Planificación temporal del proyecto 16](#_Toc72692496)

[6. DISEÑO 17](#_Toc72692497)

[6.1. Prototipo 17](#_Toc72692498)

[6.1.1. Interacciones del usuario anónimo 17](#_Toc72692499)

[6.1.2. Interacciones del usuario administrador 18](#_Toc72692500)

[6.2. Arquitectura Hexagonal 19](#_Toc72692501)

[6.3. Modelo de datos 20](#_Toc72692502)

[6.4. Vistas 21](#_Toc72692503)

[6.4.1. Menú principal 21](#_Toc72692504)

[6.4.2. Venta de tickets 21](#_Toc72692505)

[6.4.3. Consulta de horarios 22](#_Toc72692506)

[6.4.4. Insertar monedas 22](#_Toc72692507)

[6.4.5. Recoger Ticket 23](#_Toc72692508)

[6.4.6. Inicio de sesión al panel de administrador 23](#_Toc72692509)

[6.4.7. Menú para mostrar los conciertos vigentes 24](#_Toc72692510)

[6.4.8. Menú para mostrar los anuncios vigentes 24](#_Toc72692511)

[6.4.9. Menú para editar el concierto seleccionado (Scrollable) 25](#_Toc72692512)

[6.4.10. Menú para editar el anuncio seleccionado (Scrollable) 25](#_Toc72692513)

[6.5. Diseños secundarios 26](#_Toc72692514)

[7. IMPLEMENTACIÓN 27](#_Toc72692515)

[7.1. Configuración del entorno de desarrollo 27](#_Toc72692516)

[7.2. Desarrollo 28](#_Toc72692517)

[7.2.1. Las interfaces implementadas para el cliente 28](#_Toc72692518)

[7.2.2. Las interfaces implementadas para el validador 29](#_Toc72692519)

[7.2.3. Desarrollo del caso de uso del usuario anónimo 30](#_Toc72692520)

[7.2.4. Desarrollo del caso de uso del usuario administrador 31](#_Toc72692521)

[7.2.5. Desarrollo del validador de tickets 32](#_Toc72692522)

[8. PRUEBAS 33](#_Toc72692523)

[8.1. End-to-End Testing 33](#_Toc72692524)

[8.2. Realización de las pruebas 34](#_Toc72692525)

[9. CONCLUSIONES 37](#_Toc72692526)

[9.1. Conclusiones 37](#_Toc72692527)

[9.2. Posibles mejoras 37](#_Toc72692528)

[10. Referencias 38](#_Toc72692529)

[11. ANEXOS 40](#_Toc72692530)

[11.1. Manual de usuario final 40](#_Toc72692531)

[11.2. Manual de usuario administrador 41](#_Toc72692532)

[11.3. Despliega tu propia máquina de vending 42](#_Toc72692533)

# RESUMEN

Cada día, las personas valoramos más lo momentáneo, lo fácil y lo funcional. También valoramos vivir nuevas experiencias, disfrutar de un concierto o un espectáculo rodeado de los nuestros.

Las empresas están viendo como este mercado puede ser un atractivo comercial que décadas anteriores ha demostrado su gran fortaleza y que se mueve en generación en generación.

Pero, ¿Las empresas están preparadas para ofrecer una experiencia digital adaptada a nuestro tiempo? ¿Están preparadas para traer innovaciones que no entorpezcan la experiencia de usuario y que tengan un valor añadido para él usuario?

Aquí es donde entra ExpoSeller, permitiendo con una inversión mínima poder crear sistemas altamente personalizables a las necesidades de cada empresa y/o entorno dedicada al mundo de espectáculo. Permitiendo con la adaptación de unas pocas líneas de código entregar experiencias de usuario totalmente distintas entre sí.

A lo largo del presente documento, presentaré en profundidad la funcionalidad del sistema ExpoSeller, donde veremos de forma conjunta tanto la parte del cliente, donde el usuario podrá hacer la mayoría de acciones del sistema, como la parte del validador, donde el usuario podrá validar su ticket y desatar las acciones posteriores tras dicha validación.

# ABSTRACT

Every day, people value the momentary, the easy and the functional more. We also value living new experiences, enjoying a concert or a show surrounded by our own.

Companies are seeing how this market can be a commercial attraction that previous decades has shown its great strength and that moves from generation to generation.

But, are companies prepared to offer a digital experience adapted to our time? Are they prepared to bring innovations that do not hinder the user experience and that have added value for the user?

This is where ExpoSeller comes in, allowing with a minimal investment to create highly customizable systems to the needs of each company and/or environment dedicated to the world of entertainment. Allowing with the adaptation of a few lines of code to deliver user experiences totally different from each other.

Throughout this document, I will present in depth the functionality of the ExpoSeller system, where we will jointly see both the client part, where the user can do most of the actions of the system, and the validator part, where the user can validate your ticket and trigger subsequent actions after such validation.

# INTRODUCCIÓN

## Motivación

La temática del proyecto ha estado influenciada en gran medida por el hecho de que actualmente no existen soluciones comerciales en el mercado que te permitan tener un sistema para crear y validar tickets de conciertos y/o espectáculos.

Así mismo este tipo de máquinas suelen ser pedidas por encargo a grandes consultoras de software que se encarga del ciclo de vida del producto hasta el final del contrato vigente.

Otro motivo es el hecho de querer exponer un proyecto donde vaya más allá de la interacción del usuario a través de simplemente una pantalla, usando como medio principal el teléfono inteligente del usuario.

Así el proyecto busca crear una solución para el mundo de los espectáculos y/o conciertos donde, a partir de una base de código existente, puedan adaptarlo de manera muy cómoda y sencilla a las necesidades de cada empresa.

## Objetivos

El proyecto tiene como meta principal crear un *sistema de vending automático* que genere Tickets de conciertos y/o espectáculos.

Para cumplir dicha meta utilizaremos un dispositivo basado en Android, donde expondrá toda la interfaz gráfica del sistema.

Por otro lado, tendremos un dispositivo que pueda ser capaz de ejecutar aplicaciones basadas en Java Standard Edition, donde expondremos toda lógica de validación del Ticket desembocando todos los eventos que correspondan tras la validación.

Para la persistencia de los datos, utilizaremos el servicio de base de datos de Google llamado Firebase, que este actuará como servidor para el sistema.

De esta forma, tendremos una visión de todo el sistema IoT para cada recurso prestado: cliente, servidor y validador. A lo largo de este proyecto también se presentarán implementaciones de ejemplo para que sea fácil probar y entender cada parte implicada del sistema, pudiéndose remplazar de una manera muy sencilla sin alterar la base de código existente.

## ¿Qué es el Internet de las Cosas?

A lo largo del proyecto se expondrá el concepto de Internet de las cosas (en inglés, *Internet of Things,* abreviado *IoT*). Aunque en el día a día es un concepto que ha ganado mucha popularidad e impacto social, no está de más explicar el concepto que fue propuesto en el año 1999 por Kevin Ashton en el Auto-ID Center del MIT.

Kevin Ashton propuso el concepto del Internet de las Cosas a partir del hecho de que las tecnologías de la información dependían en gran medida de las personas para obtener la información deseada, y los seres humanos disponemos de una atención, un tiempo y una precisión limitada, en cambio, si una maquina fuera capaz de reconocer el entorno donde se encuentran y consiguiera transformar esa información de forma automática, se conseguiría automatizar muchos de los elementos que antes dependían de una persona humana para recolectar y tratar esa información, con «*el potencial que este tendría para cambiar el mundo tal y como lo hizo la revolución digital hace unas décadas*».

Tal y como dijo Kevin Ashton para una entrevista para el diario RFID:

*Los ordenadores actuales —y, por tanto, internet— son prácticamente dependientes de los seres humanos para recabar información. (...) El problema es que las personas tienen un tiempo, una atención y una precisión limitados, y no se les da muy bien conseguir información sobre cosas en el mundo real. (…) Si tuviéramos ordenadores que supieran todo lo que tuvieran que saber sobre las “cosas”, mediante el uso de datos que ellos mismos pudieran recoger sin nuestra ayuda, nosotros podríamos monitorizar, contar y localizar todo a nuestro alrededor, de esta manera se reducirían increíblemente gastos, pérdidas y costes. (…)  La internet de las cosas tiene el potencial para cambiar el mundo tal y como hizo la revolución digital hace unas décadas. Tal vez incluso hasta más.* [1]

(Ashton, 2009)

A día de hoy, podemos ver este concepto aplicado a muchos elementos de nuestra vida cotidiana, como pueda ser en el metro, en nuestro hogar, en la industria petrolera, en la industria aeroespacial, etc… Así mismo, también podemos verlo en las máquinas de vending automáticas, donde en los modelos más nuevos disponen de conexión a internet para recibir actualizaciones y saber en qué estado están los productos, así como también gestionar pagos con tarjetas de crédito, etc.…

# ANALISIS

## Objetivos Específicos

En función del tiempo disponible, los objetivos se van a dividir en dos apartados diferentes:

## Realizados

* Diseño gráfico de la interfaz de usuario del sistema.
* Creación de las estructuras de la base de datos no relacional.
* Acople de un sistema que permita simular la entrada de dinero.
* Generación del diseño del ticket en formato Apple Passbook.
* Diseño del validador que comprobará si el ticket del evento entregado.
* Mostrar las consultas de la base de datos, y en caso de que proceda, ordenado por fecha.
* Permitir operaciones CRUD a través de una consola de administración que permita modificar la información de los eventos y los anuncios.
* Proteger mediante usuario y contraseña este panel de administración.

## Deseables (Futuras mejoras)

* Implantar un sistema real para los pagos con dinero en metálico.
* Permitir que cualquier usuario en producción pueda generar los tickets sin necesidad de instalar los certificados de desarrollo.
* Permitir imprimir tickets físicos con un código QR asociado al mismo.
* Permitir a los usuarios invidentes escuchar lo que se está mostrando en pantalla mediante mensajes personalizados.

## Requisitos Generales:

A continuación, vamos a definir los requisitos funcionales y no funcionales que tiene que cumplir el sistema.

### Funcionales

|  |  |
| --- | --- |
| RF0001 | |
| Nombre: | Realizar consultas contra la base de datos |
| Prioridad: | Alta |
| Descripción: | La aplicación permitirá obtener la información relativa a los conciertos desde la base de datos de Firebase |
| Modificaciones: | - |

|  |  |
| --- | --- |
| RF0002 | |
| Nombre: | Filtrar consultas realizadas contra la base de datos mediante fechas |
| Prioridad: | Alta |
| Descripción: | La aplicación permitirá mostrar aquellos conciertos en una fecha específica o a partir de determinada fecha |
| Modificaciones: | - |

|  |  |
| --- | --- |
| RF0003 | |
| Nombre: | Cargar imágenes procedentes del almacén de imágenes de Firestore |
| Prioridad: | Alta |
| Descripción: | La aplicación nos permitirá recuperar las imágenes de los conciertos asociados a cada registro o los banners de anuncios en el tiempo de inactividad |
| Modificaciones: | - |

|  |  |
| --- | --- |
| RF0004 | |
| Nombre: | Mostrar anuncios durante el tiempo de inactividad relativas a conciertos |
| Prioridad: | Media |
| Descripción: | La aplicación nos permitirá mostrar anuncios relativos a los conciertos futuros |
| Modificaciones: | - |

|  |  |
| --- | --- |
| RF0005 | |
| Nombre: | Subir al almacén de datos los tickets de los conciertos |
| Prioridad: | Alta |
| Descripción: | La aplicación subirá a Firestore el ticket generado a un directorio temporal donde el usuario podrá descargarlo mediante su dispositivo inteligente |
| Modificaciones: | - |

|  |  |
| --- | --- |
| RF0006 | |
| Nombre: | Generar un QR valido para descargar el ticket generado |
| Prioridad: | Alta |
| Descripción: | Para hacer un intercambio de datos entre la máquina de vending y el dispositivo del usuario, generaremos un QR que podrá descargar en su aplicación de tickets compatible con Passbook |
| Modificaciones: | - |

|  |  |
| --- | --- |
| RF0007 | |
| Nombre: | Crear un validador sencillo para comprobar si el usuario tiene autorización para acceder al concierto o no |
| Prioridad: | Alta |
| Descripción: | Nos permitirá comprobar que el usuario en cuestión tiene un ticket valido o no mediante la conexión con nuestra base de datos y la validación del identificador de ticket |
| Modificaciones: | - |

|  |  |
| --- | --- |
| RF0008 | |
| Nombre: | Crear una consola de administración que permita hacer operaciones CRUD sobre los conciertos y los anuncios |
| Prioridad: | Alta |
| Descripción: | Pese a que Firestore tiene una interfaz lista para operar con los documentos, debemos de tener una interfaz lista que sea agnóstica de la base de datos por debajo. |
| Modificaciones: | - |

|  |  |
| --- | --- |
| RF0009 | |
| Nombre: | Añadir una implementación funcional para insertar monedas emuladas |
| Prioridad: | Media |
| Descripción: | Para facilitar posteriores implementaciones, se debe de crear una implementación sobre hardware real que nos permita, mediante un evento físico, aumentar el contador de monedas insertadas en el sistema. |
| Modificaciones: | - |

### No Funcionales

|  |  |
| --- | --- |
| RNF0001 | |
| Nombre: | Seguridad |
| Prioridad: | Alta |
| Descripción: | Añadir mediante Firestore Rules una capa de seguridad a los datos guardados en la base de datos, para evitar que no se pueda acceder a la información no publica |
| Modificaciones: | - |

|  |  |
| --- | --- |
| RNF0002 | |
| Nombre: | Limpieza del ticket generado después de un tiempo de inactividad |
| Prioridad: | Alta |
| Descripción: | Limpiar localmente el ticket generado para evitar llenar la memoria del dispositivo encargado de la interfaz de usuario. |
| Modificaciones: | - |

|  |  |
| --- | --- |
| RNF0003 | |
| Nombre: | Rendimiento |
| Prioridad: | Alta |
| Descripción: | Los componentes principales del sistema, tanto la máquina de vending como el torno automático, deberán de tener un rendimiento aceptable a las condiciones impuestas en esta clase de entornos |
| Modificaciones: | - |

## Tecnologías y herramientas utilizadas

* Android SDK [2]

Android SDK y la versión específica para el Internet de las Cosas trae las herramientas necesarias para apoyar todo el ciclo de vida de la aplicación, pasando desde la fase de diseño de las interfaces hasta su implementación mediante su versión para el Internet de las Cosas.

* IntelliJ Community Edition [3]

Entorno de desarrollo que nos permitirá diseñar el sistema del torno automático y de la interfaz gráfica del usuario donde se gestionará toda la funcionalidad del sistema. Esta herramienta además trae soporte para crear interfaces gráficas, necesarias para esta segunda parte del proyecto.

* Firebase Authentication [4]

Base de datos de usuarios del conjunto de herramientas de Firebase donde podremos registrar a los usuarios autorizados a tocar el sistema en cuestión.

* Firebase Cloud Firestore [5]

Base de datos principal de todo el sistema, donde se guardará los conciertos y los anuncios disponibles y los números de tickets generados.

* Firebase Cloud Storage [6]

Almacén de datos en la nube donde podremos subir y exponer a la red los tickets generados por parte del sistema con carácter temporal.

* QRGenerator [7]

Librería que nos permitirá generar un QR con la dirección HTTPS del ticket generado y subido a Cloud Firestore.

* Passkit4j [8]

Librería que nos permitirá generar los tickets en formato Apple Passbook, con la consiguiente capacidad de acople a la base de datos de Firestore.

* OpenCV [9]

Librería de visión por computadora con capacidades de escanear códigos QR mediante una cámara web.

* Invision Studio [10]

Herramienta de diseño gráfico de interfaces de usuario, con capacidad de planificación de la interacción entre distintas interfaces.

* JavaFX [11]

Interfaz gráfica para el diseño del código de colores del torno, a modo de emulación de los efectos físicos del mismo y de los formularios para la creación de los datos en Cloud Firestore.

* OkHttp [12]

Librería HTTP que nos permitirá atacar a las APIs de Firebase desde los clientes de Java puro.

* GitHub [13]

Repositorio de proyectos de código abierto en la nube y compatible con el control de versiones de Git.

* Git [14]

Control de versiones compatible con IntelliJ Community Edition y Github.

* Github Projects [15]

Gestor de proyectos y tareas integrado en la plataforma de Github.

* Glide [16]

Librería para la visualización de las imágenes recomendada por Google.

* Visual Paradigm Online [17]

Herramienta online para la creación de los diagramas requeridos por el proyecto.

* GIMP [18]

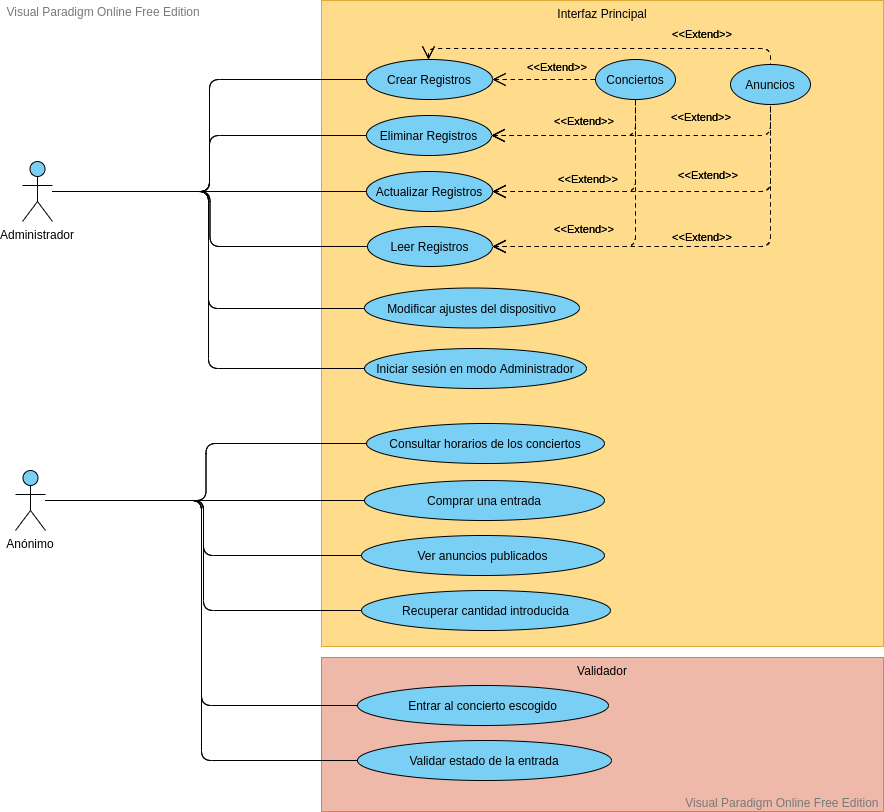
Herramienta grafica para editar imágenes rasterizadas y vectoriales.

* Adobe Colors [19]

Herramienta para obtener colores complementarios, análogos, paletas de colores, etc...

## Casos de uso

Los casos de uso del sistema nos permiten conocer el comportamiento y las funcionalidades del mismo. En este caso, se han identificado dos actores: El Usuario Administrador y el Usuario Anónimo.



En el caso del usuario administrador, como podemos comprobar, siempre estará identificado el usuario mediante email y contraseña. En el caso del usuario anónimo, no podremos nunca identificarlo, a excepción del código del ticket emitido, que nos servirá como referente para comprobar si puede pasar al evento en cuestión o no.

# PLANIFICACION TEMPORAL

A continuación, se describen las fases seguidas a lo largo de la realización del siguiente proyecto:

* **Estudio**: Investigación y/o valoración de las posibles tecnologías que pueden aplicar sobre el presente proyecto.
* **Análisis**: Definición de los requisitos y objetivos para definir en profundidad que se desea construir.
* **Diseño**: Tras la finalización de las fases de estudio y análisis, y con el listado de requisitos, se procede a diseñar las interfaces de usuarios y los casos de uso donde los propios usuarios podrán interaccionar con nuestro sistema, así mismo, también se procederá a diseñar la arquitectura de la base de datos NoSQL.
* **Implementación**: Desarrollo de los componentes propuestos en el proyecto mediante el lenguaje de programación común Java.
* **Pruebas**: Una vez finalizadas las fases anteriores, se realizan unas pruebas para comprobar que la calidad del presente sistema sea la deseada.
* **Documentación**: Descripción del proyecto mediante un documento de fácil lectura como con comentarios de código que permita su posterior mantenimiento

## Planificación temporal del proyecto

Tras la definición de las fases del proyecto, se ha realizado una planificación del tiempo para abordar el proyecto:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Estudio | Análisis | Diseño | Implementación | Pruebas | Documentación |
| Marzo | x | x | x |  |  |  |
| Abril |  |  | x | x |  |  |
| Mayo |  |  |  |  | x |  |
| Junio |  |  |  |  | x | x |

# DISEÑO

## Prototipo

El prototipo principal de la aplicación se ha realizado con la aplicación InVision Studio [10]. Se ha tenido en cuenta los detalles para optimizar la experiencia de usuario y la navegabilidad de la aplicación. A continuación, vamos a exponer todas las interacciones posibles para el usuario.

### Interacciones del usuario anónimo



Para el tema del generado del gradiente y edición de los iconos, se ha utilizado GIMP [18] y Adobe Colors [19]. Los iconos han sido extraídos del repositorio de imágenes Material Icons[20] y del paquete de iconos Vending Machine[21].

Al ser un sistema donde su destino principal es generar entradas y validar quien pasa puede pasar por el sistema, se ha tenido en cuenta principios de experiencia de usuario.

Elementos como el QR, se esperan que se genere en el momento con la URL del ticket subido a la nube para la posterior descarga por parte del usuario anónimo.

### Interacciones del usuario administrador



Manteniendo en la misma línea de diseño del caso de uso del usuario anónimo, se ha diseñado un panel de administración basándonos en el patrón de diseño gráfico Master-Detail.

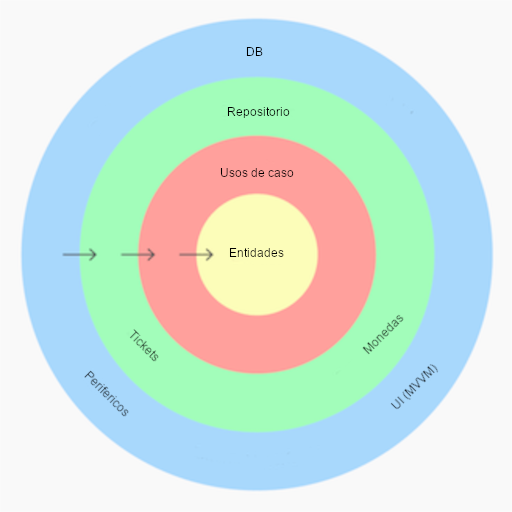
Desde el panel de administración se espera poder realizar las operaciones CRUD necesarias sobre las colecciones de Conciertos y Anuncios. Por defecto, en los listados aparecerán los conciertos vigentes, aunque siempre se podrá buscar a partir de una fecha estimada los elementos deseados.

También, al ser la interfaz de usuario una aplicación que no permitirá acceder a áreas no deseadas en Android, y, por consiguiente, siendo marcada por el sistema como una aplicación principal en el sistema, se ha colocado un botón que lanza directamente los ajustes del sistema, protegiendo así todas las configuraciones del dispositivo.

Para los campos de texto basados en fechas, se ha escrito una funcionalidad que permite directamente abrir el calendario y un reloj para elegir la fecha y la hora que se desea que represente ese campo.

## Arquitectura Hexagonal

El desarrollo del sistema ha seguido los patrones descritos en el libro *Clean Arquitecture* [22], donde se describe en particular un patrón denominado Arquitectura Hexagonal, en conjunto con el patrón de diseño Model View ViewModel para la interfaz de usuario:



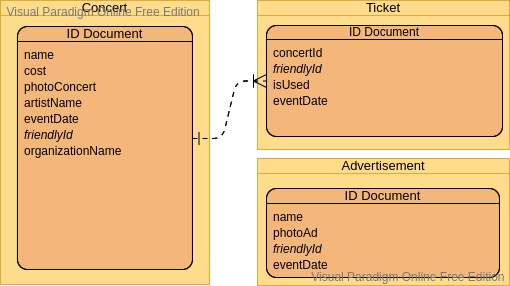
Este patrón de desarrollo nos propone dividir nuestro proyecto en distintas capas, donde cada capa solo deberá de depender de sus capas interiores, sin conocer nada de las capas superiores. Esto es posible gracias al uso de las interfaces que nos permite exponer los métodos que necesitamos de una implementación, ahorrando muchas líneas de código y problemas relacionados a la hora de implementar algo sin seguir un patrón de diseñar. Este proyecto, cabe mencionar, que se ha podido beneficiar de esta ventaja de manera significativa durante las fases de desarrollo y pruebas.

Por otra parte, tenemos el patrón de diseño aplicado a la interfaz de usuario denominado *Model View ViewModel* (Abreviado *MVVM*), este patrón que está siendo altamente usado en la nueva arquitectura denominada *Android Jetpack* nos invita a separar la capa de presentación, y su respectiva lógica, de la capa lógica de negocio, permitiendo que estos estén en tres bloques bien diferenciados y permitiendo un mayor reciclaje de código. Con la unión de estos dos patrones de diseño, obtenemos un código limpio y altamente reciclable.

## Modelo de datos

Cloud Firestore es una Base de Datos como Servicio (En inglés *Database as a Service*, Abreviado *DBaaS*) NoSQL proporcionado por Google basado en documentos y colecciones, la diferencia respecto a SQL radica en que no existe ningún mecanismo para relacionar los documentos y no tiene una estructura definida, permitiendo una flexibilidad mayor entre documentos y sus contenidos.

La imagen de a continuación representa el estado actual de la base de datos NoSQL:



Como podemos fijarnos, tenemos tres colecciones representando distintas entidades:

* Concert (O Concierto):

Aquí irán los documentos relacionados con los conciertos presentes en la base de datos, determinando así algunos datos útiles como coste y la fecha del evento.

* Ticket (O Entrada):

Aquí irán los documentos relacionados con los tickets emitidos para cada concierto en cuestión, a parte de los datos descritos para otros documentos, también nos encontramos con la particularidad de tener una clave para indicar si ha sido usado o no

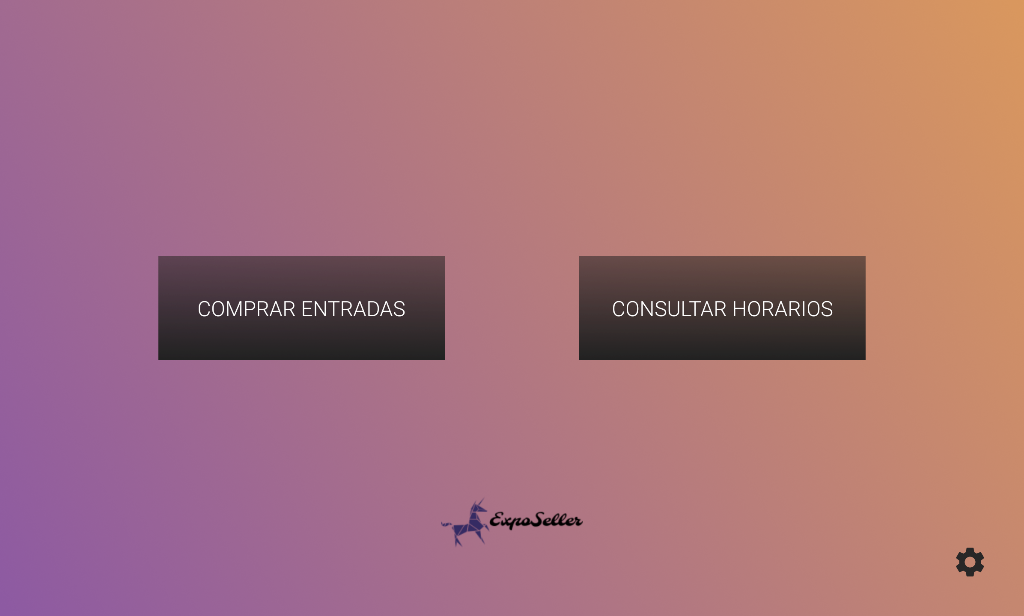
* Advertisement (O Anuncio):

Aquí irán los documentos relacionados con los anuncios listos para emitir en los tiempos de inactividad de la interfaz de usuario del sistema, que irá referenciado a una imagen cargada a un servidor de terceros.

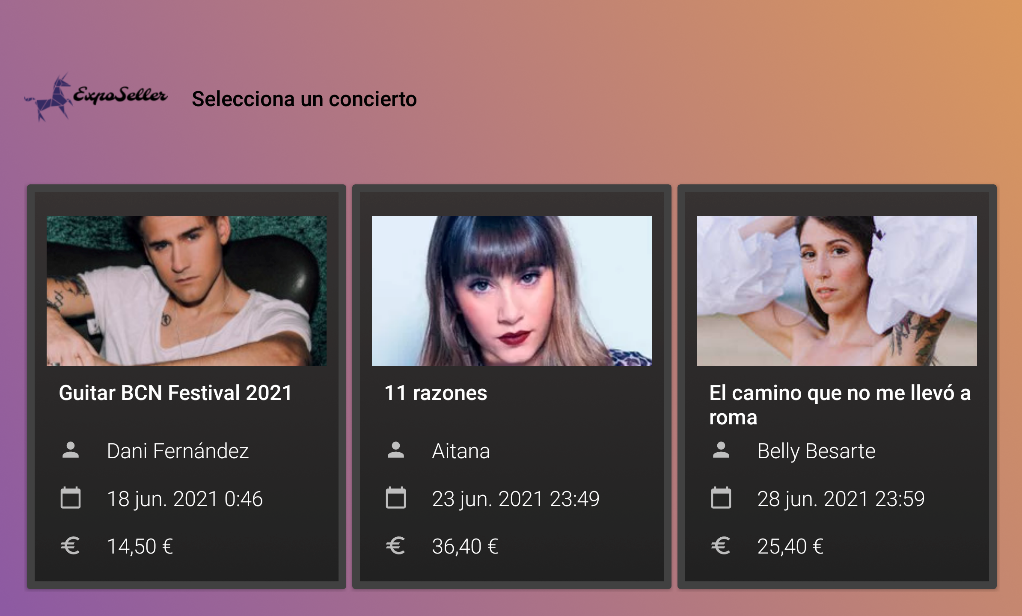
## Vistas

A continuación, mostraremos una serie de capturas relacionadas con la interfaz gráfica del usuario.

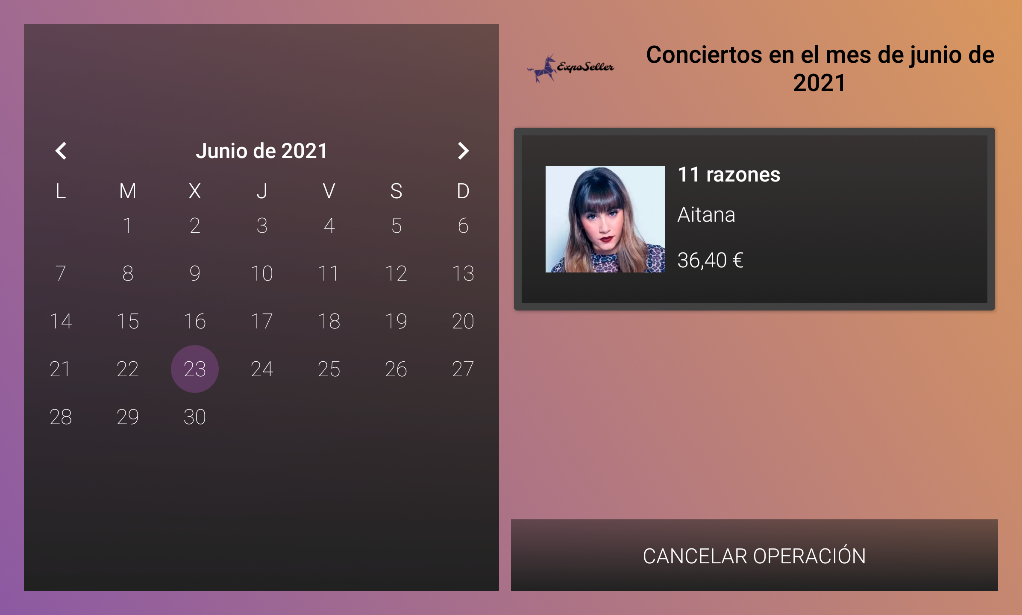
### Menú principal



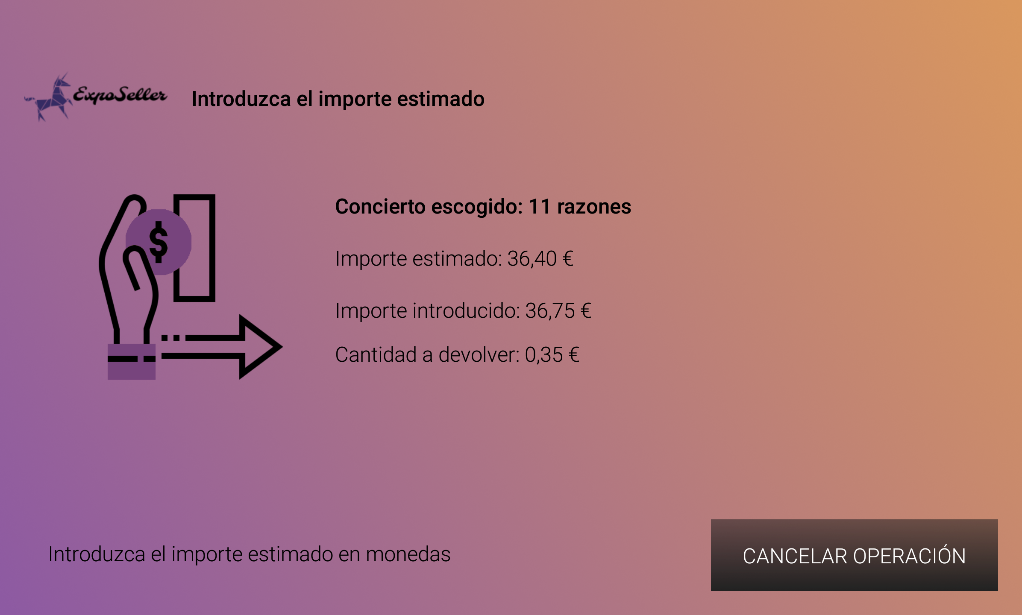
### Venta de tickets



### Consulta de horarios



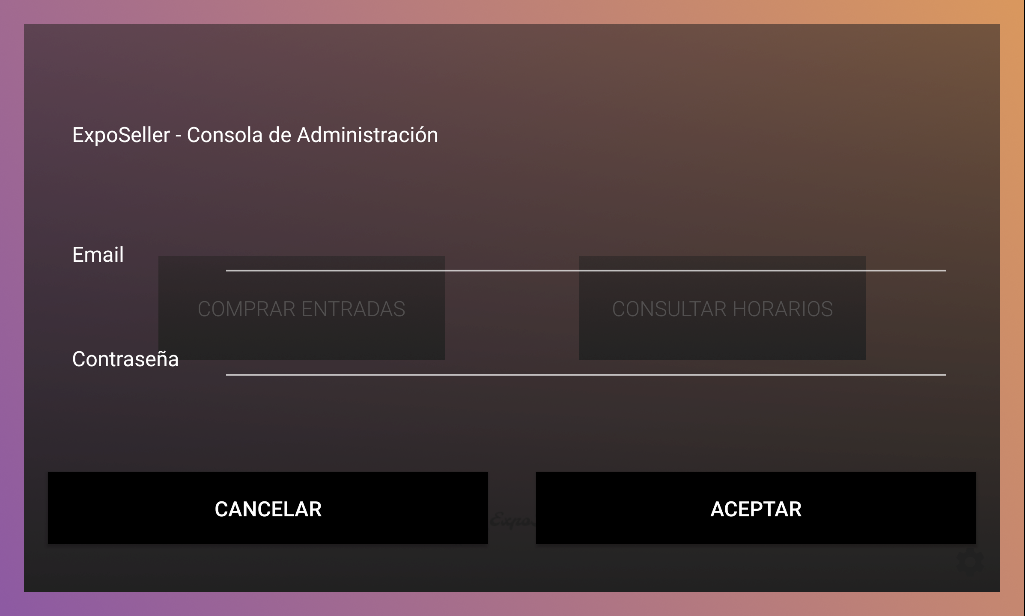
### Insertar monedas



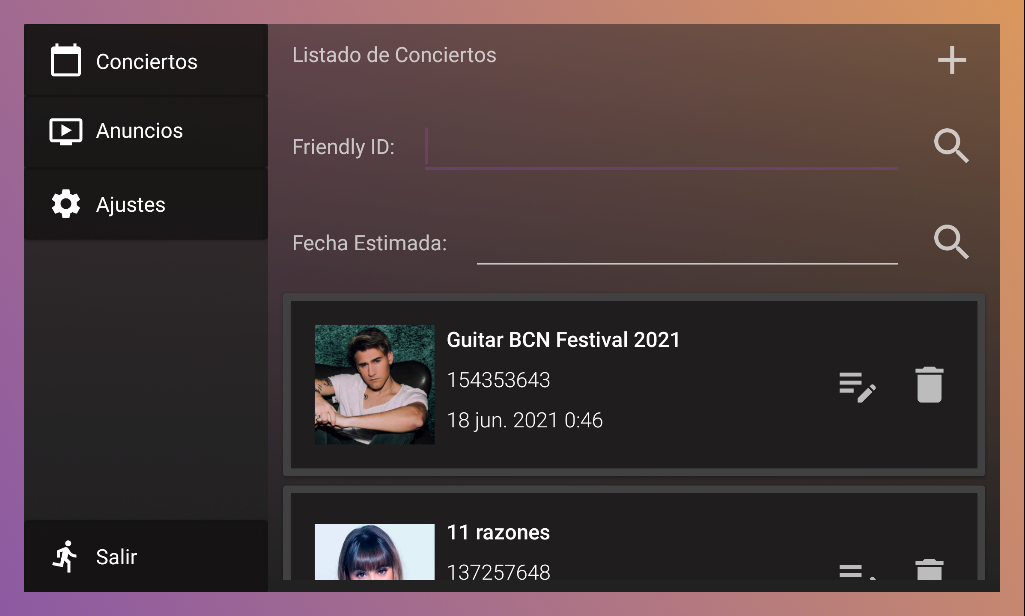
### Recoger Ticket



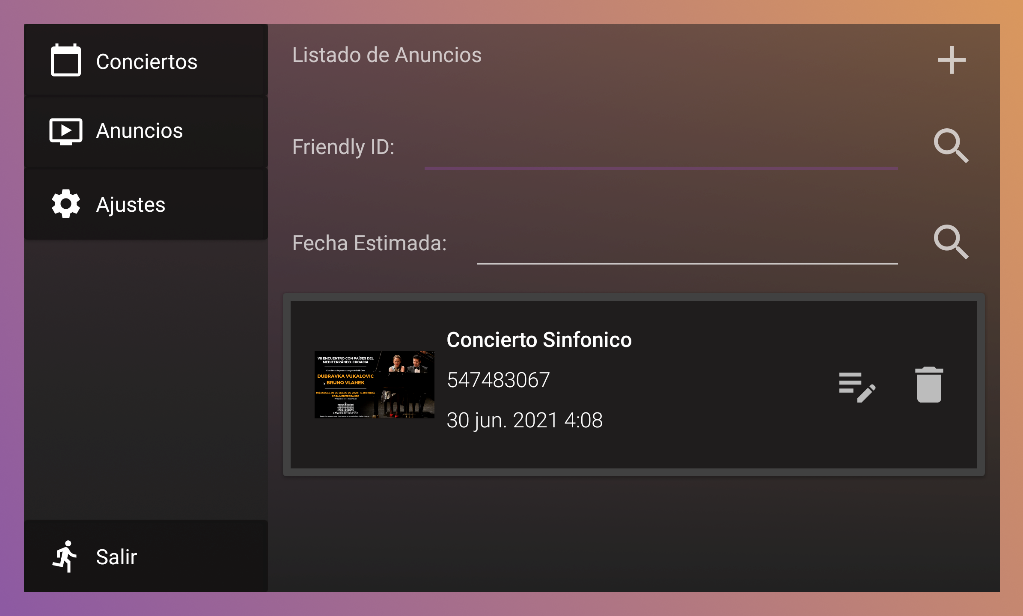
### Inicio de sesión al panel de administrador



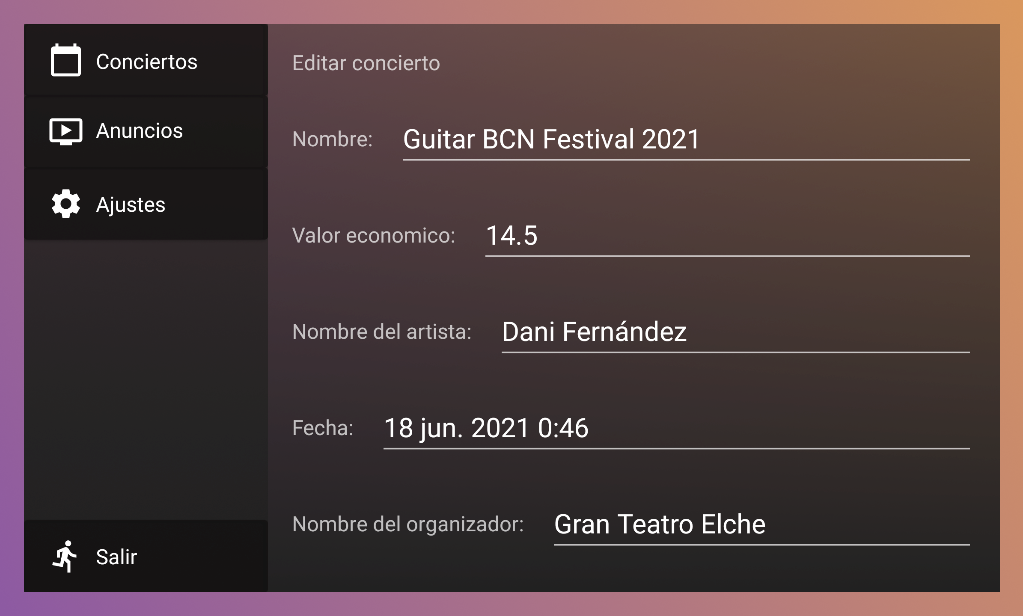
### Menú para mostrar los conciertos vigentes



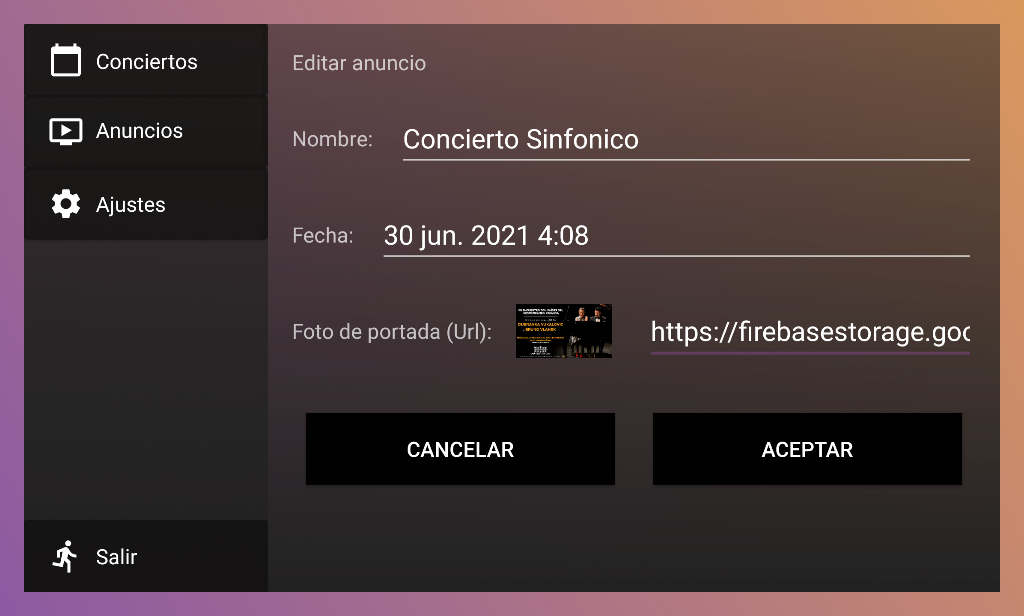
### Menú para mostrar los anuncios vigentes



### Menú para editar el concierto seleccionado (Scrollable)



### Menú para editar el anuncio seleccionado (Scrollable)

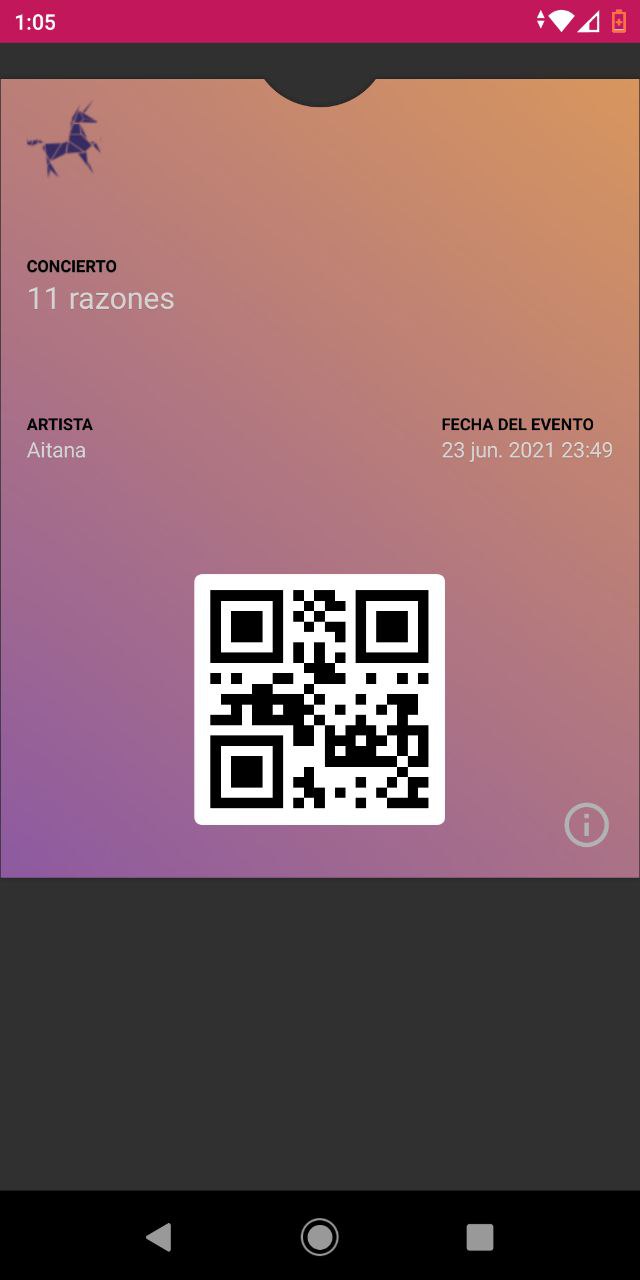


## Diseños secundarios

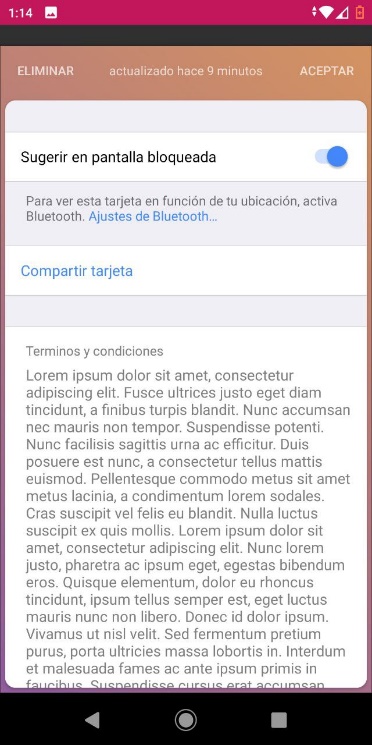
A continuación, mostraremos como se muestra el ticket virtual en formato *Apple Passbook* con la aplicación para Android de WalletPasses [23].

*Apple Passbook* es un formato digital para la emisión de entradas, tarjetas de fidelización, billetes de avión, etc.… usado como estándar *de facto* en la industria.

* + 1. Ticket de conciertos (Vista delantera)



* + 1. Ticket de conciertos (Vista trasera)



# IMPLEMENTACIÓN

## Configuración del entorno de desarrollo

Para la implementación del sistema se ha empleado un equipo:

* Portátil Dell Inspiron 7590 con 16 GB de RAM

En el equipo descrito anteriormente se ha utilizado el sistema operativo Fedora Linux 34, donde se ha instalado JavaFX [11], Git [14] y IntelliJ Community Edition [3] con las extensiones para Android SDK [2].

Para las pruebas sobre hardware real, se ha empleado los siguientes dispositivos:

* Raspberry Pi 3 Model B con 1GB de RAM

Sobre el dispositivo descrito anteriormente, se ha instalado una versión oficial de Android denominada Android Things, aunque se podría haber instalado otras versiones soportadas como EmteriaOS o LineageOS, también basados en Android.

La decisión de utilizar una Raspberry Pi en vez de utilizar una Tablet con una interfaz GPIO conectada por el puerto USB OTG es por la sencillez que supone acceder a esta interfaz incorporada directamente en la placa sobre el desarrollo, permitiendo hacer desarrollos agiles con componentes físicos.

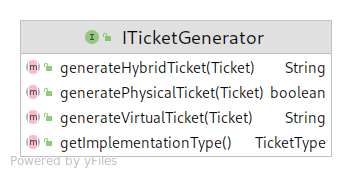
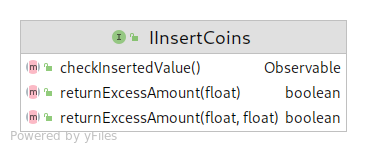
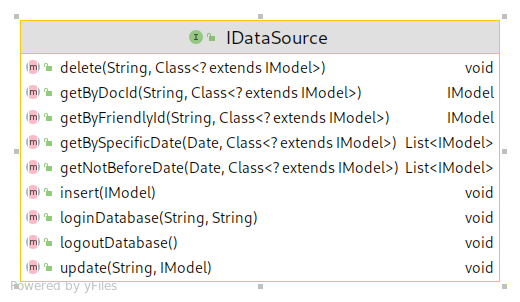
Para el validador, se ha empleado el mismo equipo usado para la implementación del sistema, dado que este es suficiente para iniciar JavaFX [11] donde mostrará el patrón de colores implementado para la demostración.

## Desarrollo

Una vez configurado el entorno de desarrollo, se inicializó los proyectos para el Validador, basado en el sistema de gestión de proyectos Maven, y la interfaz gráfica, proyecto basado en Gradle y destinado para Android 8.1 (API 27).

### Las interfaces implementadas para el cliente

Para el caso de la aplicación en el lado del cliente, siguiendo lo que está definido en el apartado *6.2 Arquitectura Hexagonal*, se ha definido 3 interfaces:



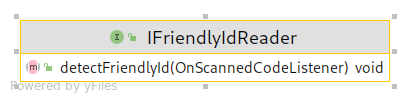
Esto nos permitirá desacoplar el código de las implementaciones permitiendo una alta reusabilidad de los componentes base del sistema. Como mención especial, tenemos la interfaz IDataSource, que dispondrá de una implementación para Cloud Firestore [5] adecuándolo a las necesidades de a la aplicación.

Para la interfaz IInsertCoins, dispondremos de una implementación basada en tiempo que nos irá sumando al contador un valor de 0,75€ hasta llegar a la cifra esperada, esto nos facilitará mucho la depuración de la aplicación, en caso de que queramos usar hardware real, simplemente tendremos que reimplementarlo.

Por último, para interfaz de ITicketGenerator, tendremos toda la lógica implementada para los tickets virtuales basados en Apple Passbook.

### Las interfaces implementadas para el validador

Para el caso de la aplicación del lado del Validador, siguiendo lo que está definido en el apartado 4.1.2 Deseables (Futuras mejoras), se han definido 2 interfaces:



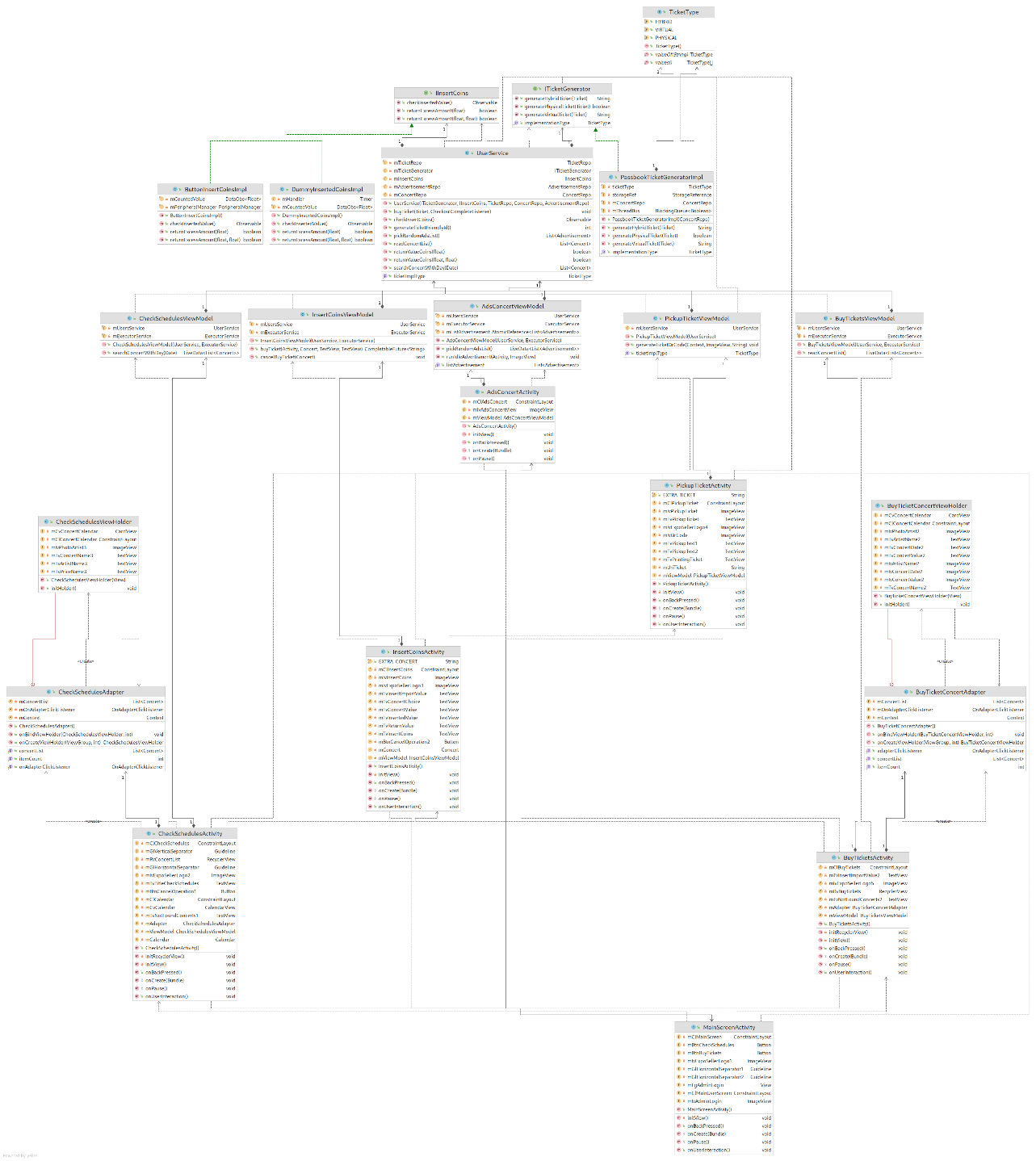


Esto nos permitirá desacoplar la lógica de las implementaciones respecto a la lógica de negocio, aparte de utilizar solo las características que necesitemos de las utilidades escogidas para esta parte del sistema.

En el caso de la interfaz IValidator, tenemos una implementación que lanza una query a Cloud Firestore [5] y nos indica, a partir de la valor de *isUsed* presente en el documento del ticket si está usado o no, y en caso de que no lo esté, indicará que este ha sido usado, actualizando este parámetro en la base de datos.

Para el caso de la interfaz IFriendlyIdReader, tenemos una implementación que estará en un hilo paralelo detectando el código QR con el código del ticket (O *friendlyId*) a través de la webcam del dispositivo. En caso de detectar el código QR, este llamará al evento pasado por parámetro del método.

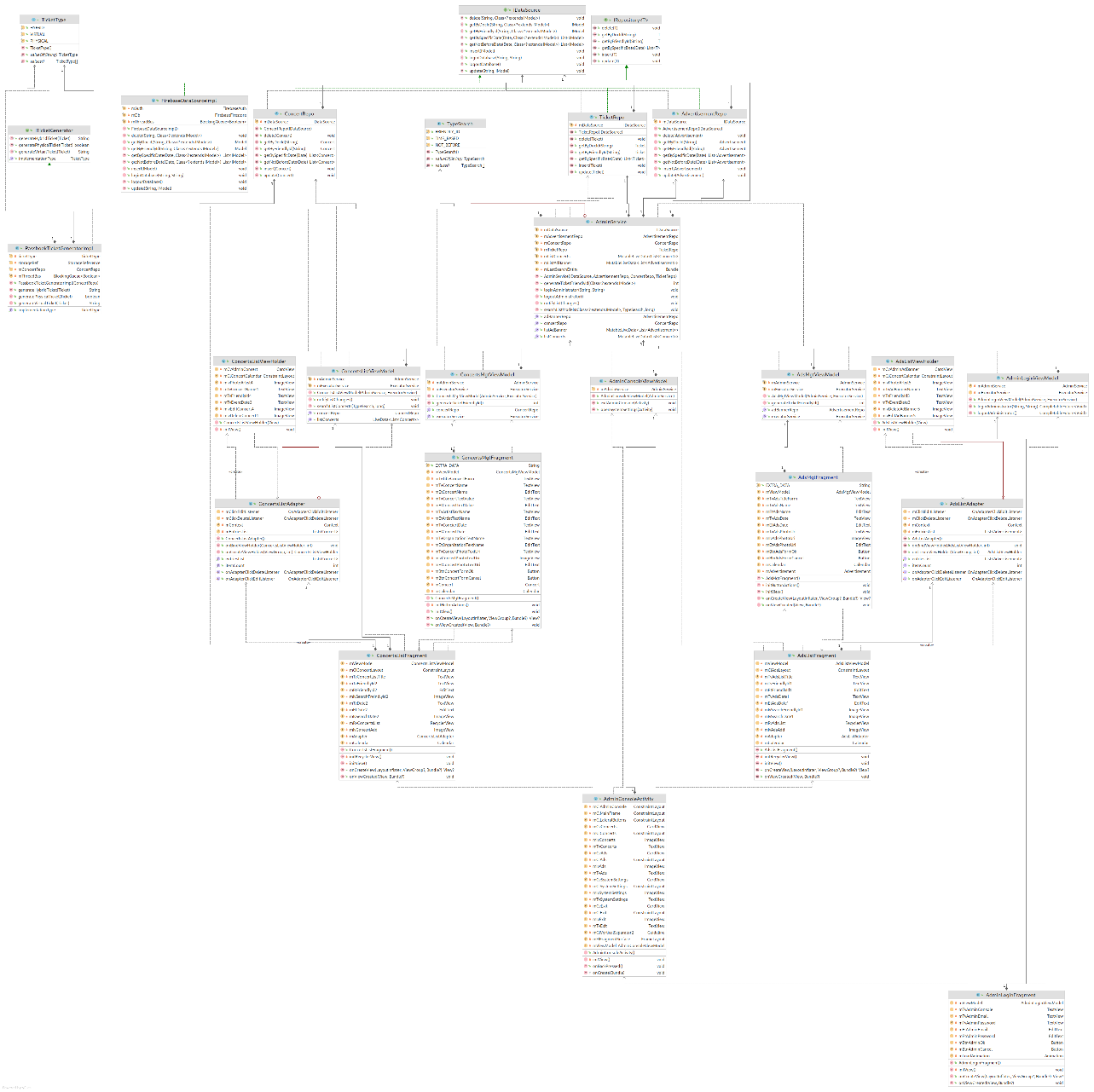
### Desarrollo del caso de uso del usuario anónimo



En el caso del desarrollo de la vista del usuario anónimo, tenemos un ViewModel por cada Actividad, los métodos disponibles en los ViewModel serán fundamentalmente un Wrapper a la clase de UserService, salvo en el caso de que necesitemos incorporar lógica de la capa de presentación al ViewModel.

Para hacer las operaciones de inyección de dependencias, utilizaremos la librería Dragger Hilt, que nos permitirá recuperar una instancia del UserService, del ExecutorManager, y otros sin tener que estar creando nuevas instancias adicionales.

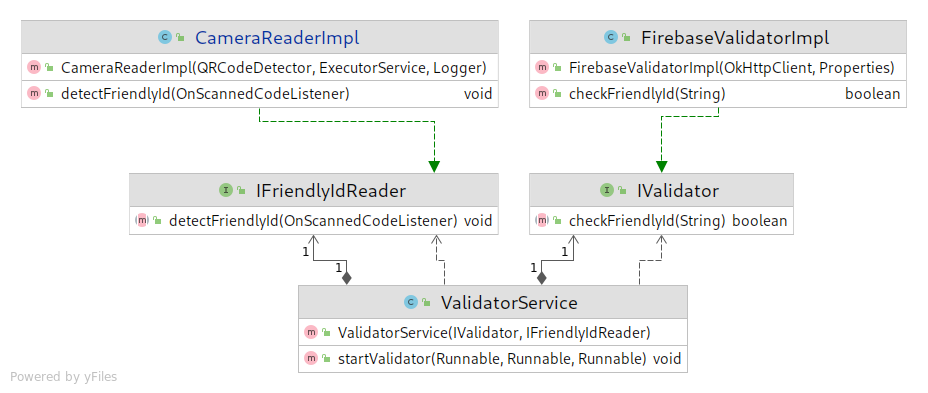
### Desarrollo del caso de uso del usuario administrador



En el caso del desarrollo de la vista del usuario administrador, se ha hecho uso de una actividad principal y de distintos fragmentos, emulando un patrón de diseño *Master-Detail*. Por cada fragmento, disponemos de un ViewModel, que fundamentalmente será un wrapper a la clase AdminService, salvo en el caso de que necesitemos incorporar lógica de la capa de presentación al ViewModel.

Para hacer las operaciones de inyección de dependencias, utilizaremos la librería Dragger Hilt, que nos permitirá recuperar una instancia del AdminService, del ExecutorManager, y otros sin tener que estar creando nuevas instancias adicionales.

### Desarrollo del validador de tickets



En el caso del desarrollo del validador, se ha optado por hacer una estructura lógica con componentes fácilmente remplazables, siguiendo los principios descritos en el apartado *7.2.2 Las interfaces implementadas para el validador*.

Aunque se ha desarrollado elementos gráficos con JavaFX [11] para esta parte, estos elementos cumplen un papel simplemente visual, mediante un código de colores (Rojo en caso de que no esté validado y Verde en el caso de que sí) nos informará del estado de la validación.

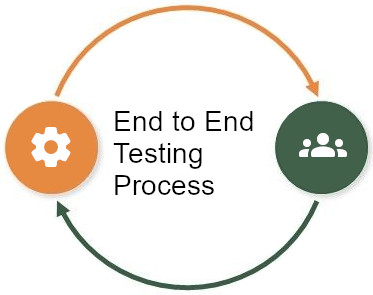
Así que dicho lo anterior, realmente la lógica de la validación reside tanto en la capa de servicio, siendo este el que tiene el mayor grado de abstracción, como en las implementaciones más externas el núcleo de la aplicación.

Para las operaciones de inyección de dependencias, utilizaremos la librería Weld, que nos permitirá recuperar una instancia del ValidationService, del ExecutorManager, y otros sin tener que estar creando nuevas instancias adicionales.

En caso de que se quisiese portar a hardware real para no depender de JavaFX [11], con esta estructura simplemente deberíamos de hacer los cambios oportunos en las clases implicadas en el proyecto con esta librería, en este caso ExpoSellerApplication, demostrando así el poder real que tiene este tipo de arquitecturas aplicadas a proyectos listos para producción.

# PRUEBAS

## End-to-End Testing



Las pruebas End-to-End es una técnica en la que consiste probar todo un producto de software desde el principio hasta el final para asegurarnos que el flujo de la aplicación y el comportamiento es el esperado. El propósito principal de este tipo de pruebas consiste en emular la experiencia final de usuario simulando un escenario real.

Los beneficios de las pruebas End-to-End consisten en la detección temprana de bugs, la contribución al funcionamiento correcto de todo el sistema, expandir la cobertura de test disponibles, reduce los tiempos para la comercialización del producto.

En este proyecto, existe más de un componente que conforma todo el sistema, entonces, este tipo de pruebas se vuelven ideales para detectar posibles fallos que pueda ocurrir entre la comunicación entre componentes y dentro de la lógica de los mismos. También nos servirá para comprobar el correcto funcionamiento con los servicios de Cloud Firestore [5], Firebase Authentication [4] y Cloud Storage [6] .

## Realización de las pruebas

Para la realización de las pruebas se ha tenido en cuenta la técnica descrita en el punto 8.1 End-to-End Testing. Además, se han hecho pruebas en las plataformas Android 8.1 (API 27) y Android 10 (API 29) para el caso del cliente, y en el caso del validador, se han hecho sobre Java SE 11 con las extensiones para JavaFX.

En los dispositivos donde se ha concentrado la mayoría de pruebas es principalmente en los siguientes:

* Google Pixel 3 (Android 8.1) (Android Emulator)
* Xaomi Mi A2 (Android 10)
* Raspberry Pi 3 Model B (Android Things 8.1)

Los primeros dos dispositivos me han permitido evaluar de forma rápida el funcionamiento del cliente en distintos factores de forma y la capacidad de adaptación del sistema, además de contar con la posibilidad de utilizar el emulador de Android SDK.

El uso de la Raspberry Pi 3 Model B, es debido a que realmente es la plataforma de destino, entonces, la mayoría de pruebas se han realizado sobre este dispositivo para el lado del cliente.

En la siguiente página, describiremos que tipos de pruebas se han realizado para este proyecto.

|  |  |
| --- | --- |
| Acciones | Resultado |
| El usuario anónimo puede comprar una entrada de un concierto | OK |
| El usuario anónimo no puede comprar una entrada de un concierto ya pasado | OK |
| El usuario anónimo puede acceder al calendario de eventos del sistema | OK |
| El usuario anónimo no puede acceder al panel de administrador | OK |
| El usuario anónimo no puede acceder a las configuraciones del dispositivo cliente | OK |
| El usuario administrador puede ejecutar operaciones CRUD sobre los conciertos y los anuncios | OK |
| Al salir del modo de administración, deja de disponer el terminal de los privilegios necesarios para modificar el valor de los conciertos | OK |
| El usuario administrador puede acceder a la configuración del dispositivo | OK |
| No existen fugas de memoria en el almacenamiento del dispositivo tras largas sesiones de uso | OK |
| El validador comprueba correctamente que el ticket a comprobar es válido o no | OK |
| El validador marca como usado el ticket validado previamente | OK |
| El validador, en caso de detectar un ticket no valido, no ejecuta operaciones adicionales | OK |

* 1. Evaluación

Durante la fase de pruebas, se tuvieron en cuenta las diversas opiniones de amigos, compañeros de trabajo, familiares. Se ha solicitado a estas mismas personas que realicen un proceso de comprar y validación del ticket, y otra de consulta de horarios, proceso de compra y validación del ticket sin intervención del desarrollador.

De esta forma se ha podido evaluar, mediante un examen visual de las acciones del usuario, la usabilidad y complejidad del uso del sistema, siendo este fácilmente entendible por las personas que han realizado las diferentes pruebas.

Una vez hecho este examen, han respondido a varias preguntas sobre cómo habían sentido la experiencia de usar el sistema y todos contestaron lo mismo: Funcional, Intuitivo, Directo, Rápido, Sencillo, Bonito (a nivel de interfaz de usuario) y Novedoso.

También, en esta fase, se han aceptado sugerencias de mejora por parte de los integrantes en estas pruebas, mejorando así las sensaciones de cara al usuario que transmite el sistema.

# CONCLUSIONES

## Conclusiones

Tras la realización de este proyecto, siento una gran satisfacción a nivel personal, dado la abstracción que poseía este proyecto, dejando así dentro de mí una gran experiencia que espero que pueda ser útil en un futuro.

Conceptos como la arquitectura hexagonal o la inyección de dependencias eran conceptos que eran bastante novedosos para mí, al menos desde el punto de vista donde lo he tomado. Pienso que este tipo de arquitectura es un acierto de cara a que este proyecto pueda ser integrado en un entorno real sin necesitar de mucho esfuerzo de desarrollo, pues esto ya simplemente trata de enlazar el hardware real con las interfaces expuestas para ello.

La planificación temporal del proyecto ha resultado ser algo mayor de las 40 horas esperadas, esto es debido a las pruebas que se les han hecho a todos los componentes del sistema, aunque tras la decisión de usar los principios de la arquitectura hexagonal se ha podido reutilizar un gran número de componentes (Clases, Implementaciones, Utilidades, etc…) del sistema, limitando el número de correcciones necesarias.

He de dar las gracias al equipo docente que me ha impartido todos los conocimientos que he precisado en este proyecto, destacando así la gran labor que han realizado en este curso con la mitad el tiempo disponible. “*Ser profesor en estos tiempos de pandemia es cosa de superhéroes.*”

## Posibles mejoras

Como he destacado en el punto *4.1.2 Deseables (Futuras mejoras)*, pienso que este proyecto puede mejorar abarcando áreas nuevas como la lectura en pantalla por parte de los usuarios invidentes. Por falta de tiempo y de la necesidad de una investigación profunda para no lastrar la experiencia de usuario, he optado por excluir esta característica, y otras, de esta versión de producto mínimo viable. Dejando así un producto funcional y que pueda ser implementado por una empresa dentro de sus instalaciones y requisitos de proyecto.

# Referencias

|  |  |
| --- | --- |
| [1] | K. Ashton, «Esa cosa del 'internet de las cosas',» 2009. [En línea]. Available: https://www.rfidjournal.com/that-internet-of-things-thing. |
| [2] | Google, Inc., «Android SDK,» [En línea]. Available: https://developer.android.com/studio#command-tools. |
| [3] | JetBrains s.r.o, «IntelliJ Community Edition,» [En línea]. Available: https://www.jetbrains.com/es-es/idea/. |
| [4] | Google, Inc., «Firebase Authentication,» [En línea]. Available: https://firebase.google.com/products/auth. |
| [5] | Google, Inc., «Firebase Cloud Firestore,» [En línea]. Available: https://firebase.google.com/products/firestore. |
| [6] | Google, Inc., «Firebase Cloud Storage,» [En línea]. Available: https://firebase.google.com/products/storage. |
| [7] | M. Mad, «QRGenerator,» [En línea]. Available: https://github.com/androidmads/QRGenerator. |
| [8] | R. Tenney, «Passkit4j,» [En línea]. Available: https://github.com/ryantenney/passkit4j. |
| [9] | Intel Corporation, «OpenCV,» [En línea]. Available: https://opencv.org/. |
| [10] | InVisionApp, Inc., «Invision Studio,» [En línea]. Available: https://invisionapp.com/studio. |
| [11] | Oracle Corporation, «JavaFX,» [En línea]. Available: https://openjfx.io/. |
| [12] | Square, Inc., «OkHttp,» [En línea]. Available: https://square.github.io/okhttp/. |
| [13] | GitHub, Inc., «GitHub,» [En línea]. Available: https://github.com. |
| [14] | Linus Torvalds, «Git,» [En línea]. Available: https://git-scm.com/. |
| [15] | GitHub, Inc., «Github Projects,» [En línea]. Available: https://github.com/features/project-management/. |
| [16] | Bumptech Inc., «Glide,» [En línea]. Available: https://github.com/bumptech/glide. |
| [17] | Visual Paradigm, «Visual Paradigm Online,» [En línea]. Available: https://visual-paradigm.com/. |
| [18] | P. Mattis, «GIMP,» [En línea]. Available: https://www.gimp.org/. |
| [19] | Adobe, Inc., «Adobe Colors,» [En línea]. Available: https://color.adobe.com/. |
| [20] | Google, Inc., «Material Icons,» [En línea]. Available: https://material.io/icons. |
| [21] | K. Abhiromsawat, «Vending Machine,» [En línea]. Available: https://www.flaticon.com/packs/vending-machine-5. |
| [22] | R. C. Martin, Clean Architecture, Addison-Wesley, 2017. |
| [23] | Wallet Passes Alliance, «WalletPasses,» [En línea]. Available: https://play.google.com/store/apps/details?id=io.walletpasses.android. |

# ANEXOS

## Manual de usuario final

## Manual de usuario administrador

## Despliega tu propia máquina de vending