



## **Sistema para máquinas de vending en conciertos**

**Proyecto final de curso**

**Autor:** Sergio Martínez Aznar

**Grado Superior:** Desarrollo de Aplicaciones Multiplataforma

**Curso Académico 2020-2021**

## CONTENIDO

1. RESUMEN .....	5
2. ABSTRACT .....	6
3. INTRODUCCIÓN .....	7
3.1. Motivación .....	7
3.2. Objetivos .....	7
3.3. ¿Qué es el Internet de las Cosas? .....	8
4. ANALISIS .....	9
4.1. Objetivos Específicos .....	9
4.1.1. Realizados .....	9
4.1.2. Deseables (Futuras mejoras) .....	9
4.2. Requisitos Funcionales y no Funcionales: .....	10
4.2.1. Funcionales .....	10
4.2.2. No Funcionales .....	12
4.3. Tecnologías y herramientas utilizadas .....	13
4.4. Casos de uso .....	16
5. PLANIFICACIÓN TEMPORAL .....	17
5.1. Planificación temporal del proyecto .....	17
6. DISEÑO .....	18
6.1. Prototipo .....	18
6.1.1. Interacciones del usuario anónimo .....	18
6.1.2. Interacciones del usuario administrador .....	19
6.2. Arquitectura Hexagonal .....	20
6.3. Modelo de datos .....	21
6.4. Vistas .....	22
6.4.1. Menú principal .....	22

6.4.2.	Venta de tickets .....	22
6.4.3.	Consulta de horarios .....	23
6.4.4.	Insertar monedas.....	23
6.4.5.	Recoger Ticket .....	24
6.4.6.	Inicio de sesión al panel de administrador.....	24
6.4.7.	Menú para mostrar los conciertos vigentes .....	25
6.4.8.	Menú para mostrar los anuncios vigentes .....	25
6.4.9.	Menú para editar el concierto seleccionado (Scrollable) .....	26
6.4.10.	Menú para editar el anuncio seleccionado (Scrollable).....	26
6.5.	Diseños secundarios.....	27
7.	IMPLEMENTACIÓN.....	28
7.1.	Configuración del entorno de desarrollo .....	28
7.2.	Desarrollo.....	29
7.2.1.	Las interfaces implementadas para el cliente.....	29
7.2.2.	Las interfaces implementadas para el validador .....	30
7.2.3.	Desarrollo del caso de uso del usuario anónimo .....	31
7.2.4.	Desarrollo del caso de uso del usuario administrador .....	32
7.2.5.	Desarrollo del validador de tickets.....	33
8.	PRUEBAS.....	34
8.1.	End-to-End Testing .....	34
8.2.	Realización de las pruebas .....	35
9.	CONCLUSIONES .....	38
9.1.	Conclusiones .....	38
9.2.	Posibles mejoras.....	38
10.	Referencias .....	39

11.	ANEXOS .....	42
11.1.	Manual de usuario administrador.....	42
11.1.1.	Inicio de sesión en el sistema como administrador .....	42
11.1.2.	La interfaz del listado de conciertos .....	43
11.1.3.	La interfaz del listado de anuncios .....	44
11.1.4.	Añadiendo o editando un concierto .....	45
11.1.5.	Añadiendo o editando un anuncio .....	46
11.1.6.	Añadiendo nuevos usuarios administradores.....	47
11.2.	Manual para desarrolladores .....	48
11.2.1.	Implementando la interfaz IInsertCoins .....	48
11.2.2.	Implementando la interfaz ITicketGenerator .....	49
11.2.3.	Implementando la interfaz IFriendlyIdReader.....	50
11.3.	Manual de instalación .....	51

## **1. RESUMEN**

El Internet de la Cosas nos permite explorar nuevas opciones de automatización obteniendo sistemas más eficientes y versátiles, hasta el punto de poder ser más fiables que la propia intervención humana.

La situación actual de pandemia ha incrementado la necesidad de disponer sistemas funcionales que eviten el contacto estrecho entre personas, necesidad que ha sido demandada tanto por parte de la ciudadanía como de empresas e instituciones.

A su vez, para las empresas esto se traduce en liberar parte de la carga de trabajo de sus empleados, así como reducir el riesgo de contagio y cualquier otro tipo de lesiones, limitando el campo de acción al mantenimiento del sistema.

Bajo estas premisas surgió ExpoSeller, un software libre pensado específicamente para eventos de espectáculo presenciales que permite automatizar la venta de entradas y su posterior validación. Para su uso se requiere un hardware especializado de vending y un segundo con detección QR que haga las funciones de validación. Los componentes del sistema tendrán que ser instalado en ambos dispositivos.

A lo largo del presente documento describiremos como el sistema cubre las necesidades expuestas y, además, el proceso que se ha llevado a cabo para el desarrollo.

## **2. ABSTRACT**

The Internet of Things allows us to explore new automation options, obtaining more efficient and versatile systems, to the point of being more reliable than human intervention itself.

The current pandemic situation has increased the need for functional systems that avoid close contact between people, a need that has been demanded by both citizens and companies and institutions.

In turn, for companies this translates into releasing part of the workload of their employees, as well as reducing the risk of contagion and any other type of injury, limiting the field of action to maintaining the system.

Under these premises, ExpoSeller emerged, a free software designed specifically for face-to-face show events that enables the automation of ticket sales and their subsequent validation. For its use, specialized vending hardware is required and a second with QR detection that performs the validation functions. System components will have to be installed on both devices.

Throughout this document, we will describe how the system meets the stated needs and, in addition, the process that has been carried out for the development.

### 3. INTRODUCCIÓN

#### 3.1. 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 espectáculos de manera autónoma y flexible para la empresa que desee utilizarlo con un hardware específico.

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

Otro motivo, es el hecho de querer exponer un proyecto que 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 de los espectáculos donde, a partir de una base de código existente, puedan adaptarlo de manera muy cómoda y sencilla a las necesidades de cada empresa.

#### 3.2. 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, que 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 la lógica de la validación del Ticket que generará una serie de eventos al finalizar la validación.

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

Con los elementos descritos en los párrafos anteriores, tendremos una visión de todo el sistema basado en el Internet de las cosas (en inglés, *Internet of Things*, abreviado *IoT*) para cada recurso: 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 reemplazar de una manera muy sencilla sin alterar la base de código existente.

### 3.3. ¿Qué es el Internet de las Cosas?

A lo largo del proyecto se expondrá el termino de Internet de las Cosas. Aunque en el día a día es un concepto que ha ganado mucha popularidad y trascendencia, no está de más explicar la idea, que fue propuesta 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 máquina fuera capaz de reconocer el entorno donde se encuentra y consiguiera transformar esa información de forma automática, se conseguiría automatizar mucho de los procesos evitando así una supervisión constante por parte de los usuarios, 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)

Hoy en día, podemos ver este concepto aplicado a muchos elementos de nuestra sociedad, como pueda ser en el metro, en nuestro hogar, en la industria petrolera, en la industria aeroespacial, ... 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, ...



## 4. ANALISIS

### 4.1. Objetivos Específicos

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

#### 4.1.1. Realizados

- Diseñar gráfico de la interfaz de usuario del sistema.
- Crear de las estructuras de la base de datos no relacional.
- Acoplar un sistema que permita simular la entrada de dinero.
- Generación del diseño del ticket en formato *Apple Passbook*.
- Implementar sistema de validación de tickets emitidos.
- Mostrar las consultas de la base de datos, y en caso de que proceda, ordenado por fecha.
- Implementar funciones CRUD a través de una consola de administración para modificar los eventos y los anuncios.
- Proteger mediante usuario y contraseña el panel de administración.

#### 4.1.2. Deseables (Futuras mejoras)

- Implementar un sistema real para los pagos.
- Permitir que cualquier usuario en producción pueda almacenar los tickets de eventos sin las limitaciones que traen el certificado auto firmado.
- Permitir imprimir tickets físicos con un código QR asociado al mismo.
- Adaptar el servicio para que sea accesible a personas discapacitadas.
- Permitir control de aforo de las entradas

#### 4.2. Requisitos Funcionales y no Funcionales:

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

##### 4.2.1. Funcionales

Los requisitos funcionales son aquellos que presenta un carácter concreto que define la inclusión de las características previstas en esta versión del proyecto.

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 <i>Firebase</i> .
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 a partir del campo de URI
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 del dispositivo de interfaz grafica
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 si 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 permite operar con los documentos, debemos tener una interfaz que sea agnóstica con respecto a la base de datos que haya por debajo de manera cómoda y sencilla.
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:	-

#### 4.2.2. No Funcionales

Los requisitos no funcionales son aquellos requisitos que involucran a todas, o a un gran número, las partes del sistema, estos requisitos pues presentan cierto carácter abstracto que debe de materializarse cuando se codifica el sistema.

RNF0001	
Nombre:	Seguridad
Prioridad:	Alta
Descripción:	Añadir mediante Firestore una capa de seguridad a los datos guardados en la base de datos, evitando que se pueda acceder a la información pública.
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 tener un rendimiento aceptable a las condiciones impuestas en esta clase de entornos.
Modificaciones:	-

RNF0004	
Nombre:	Sencillez para implementar las interfaces de hardware
Prioridad:	Alta
Descripción:	Se debe de permitir, a través de las interfaces expuestas, la documentación que deberá estar en el presente documento y a través de comentarios de código, implementar el hardware compatible con los casos de uso del sistema.
Modificaciones:	-

#### 4.3. 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.

- 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 administradores autorizados a cambiar la información disponible en el sistema.

- 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 en 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 almacenado en *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, ...

- Dragger Hilt [20]

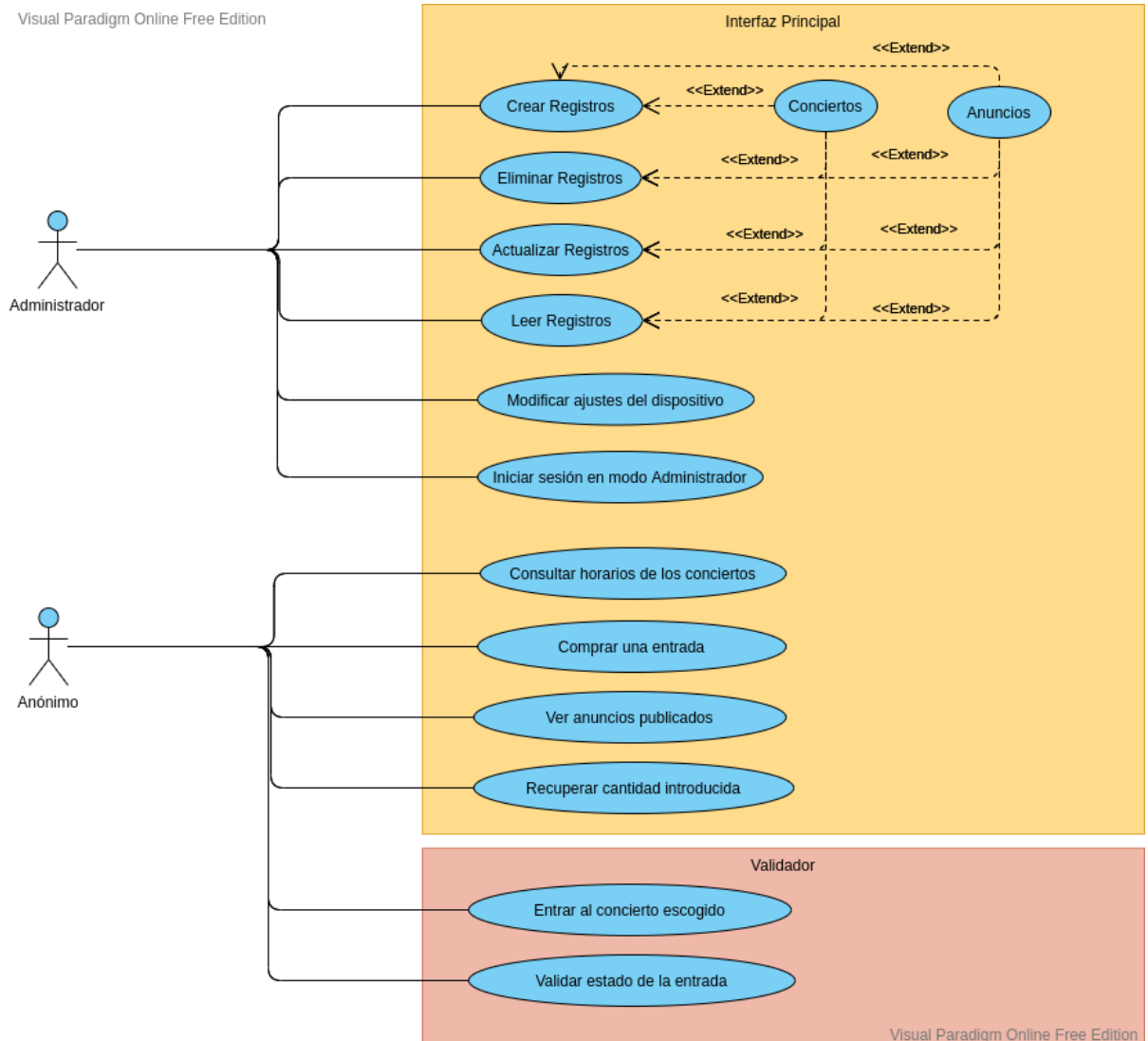
Tecnología para la inyección de dependencias recomendada por *Google*.

- Weld SE [21]

Tecnología de inyección de dependencias basado en el estándar *Java CDI* para la plataforma *Java Standard Edition*.

#### 4.4. Casos de uso

Los casos de uso del sistema nos permiten conocer el comportamiento y las funcionalidades de este. 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 mediante email y contraseña. Mientras que, 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. En ningún caso, ninguno de los dos actores deberá de acceder a los casos de uso del otro, respondiendo así al principio de división de responsabilidades.



## 5. PLANIFICACIÓN TEMPORAL

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

- **Estudio:** Investigación y/o valoración de las posibles tecnologías que se 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 implementa una batería de pruebas para garantizar la calidad del sistema.
- **Documentación:** Descripción del proyecto mediante un documento de fácil lectura con comentarios de código que permita su posterior mantenimiento.

### 5.1. Planificación temporal del proyecto

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

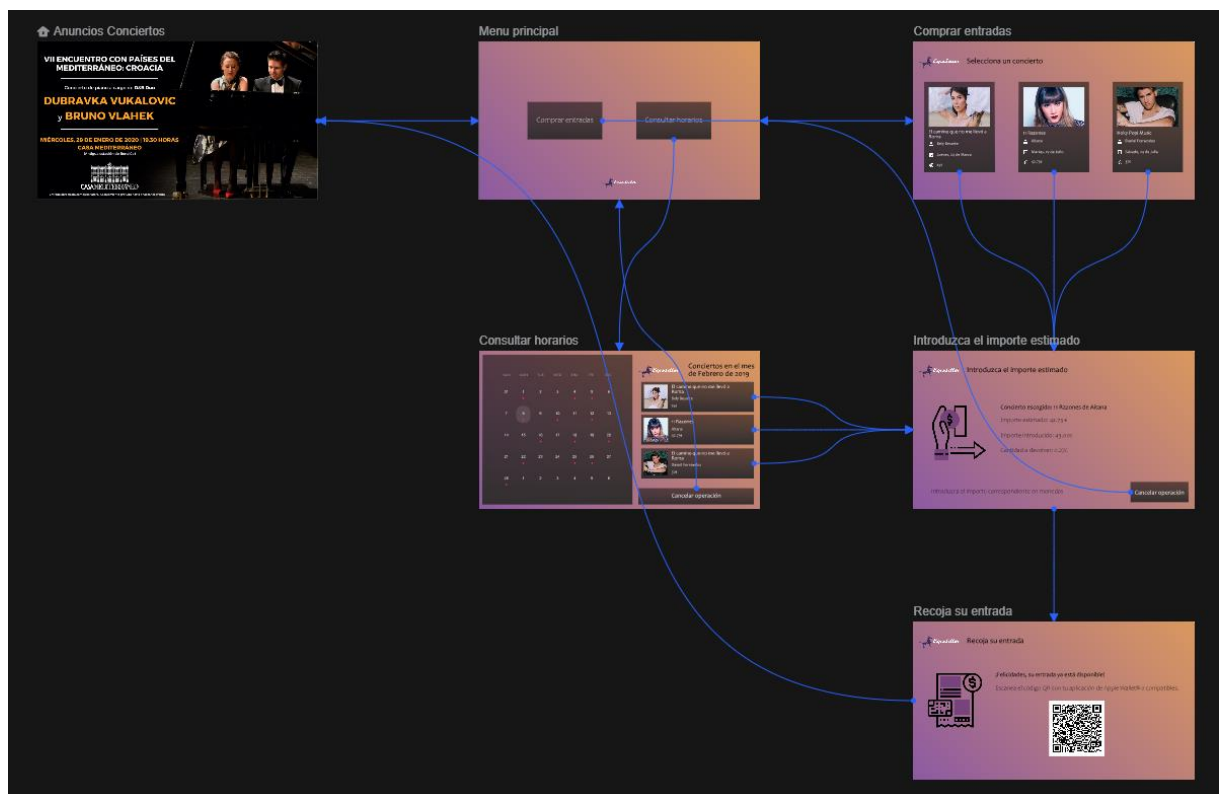
	Estudio	Análisis	Diseño	Implementación	Pruebas	Documentación
Marzo	x	x	x			
Abril			x	x		
Mayo					x	
Junio					x	x

## 6. DISEÑO

### 6.1. 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.

#### 6.1.1. Interacciones del usuario anónimo

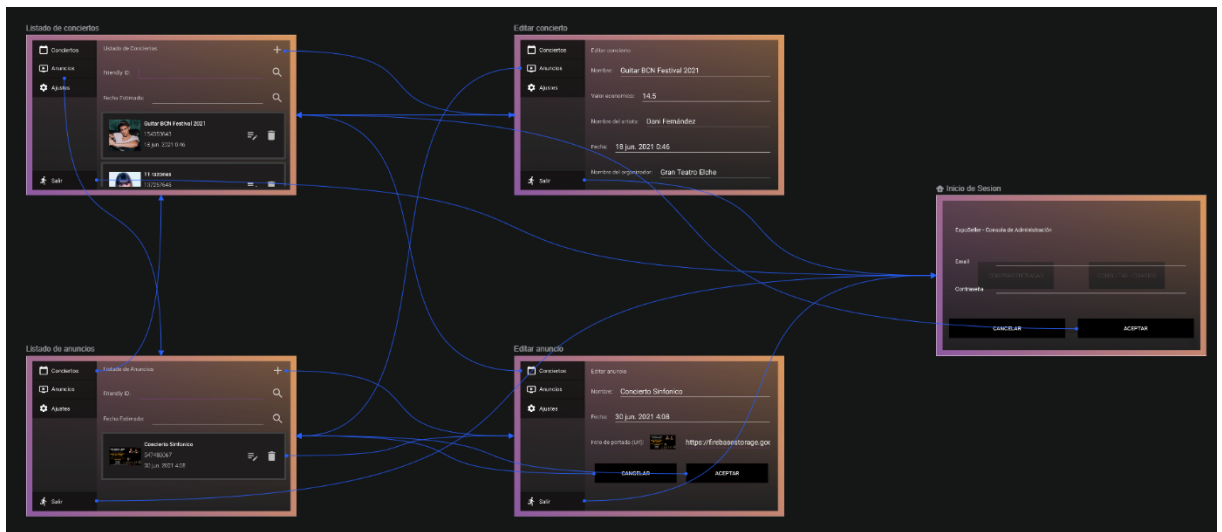


Para el tema del generado del gradiente y edición de los iconos, se ha utilizado *GIMP* [18] y *Adobe Colors* [19]. Los iconos han sido extraídos del repositorio de imágenes *Material Icons* [22] y del paquete de iconos *Vending Machine* [23].

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

El código QR se genera en el momento a partir de la URL del ticket generado a partir del proceso de compra del usuario.

### 6.1.2. Interacciones del usuario administrador



Manteniendo 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*.

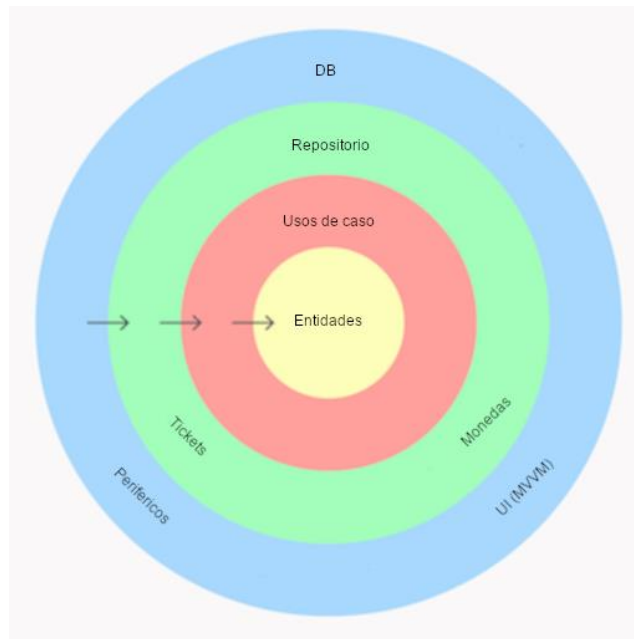
Desde el panel de administración se espera poder realizar las operaciones *CRUD* necesarias sobre las colecciones de Conciertos y Anuncios. Por defecto, en los listados aparecerán los conciertos vigentes, aunque siempre se podrán buscar a partir de una fecha los elementos deseados.

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, se ha colocado un botón que lanza directamente los ajustes del dispositivo, protegiendo así todas las configuraciones que no deben de ser expuestas.

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

## 6.2. Arquitectura Hexagonal

El desarrollo del sistema ha seguido los patrones descritos en el libro *Clean Architecture* [24], 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:



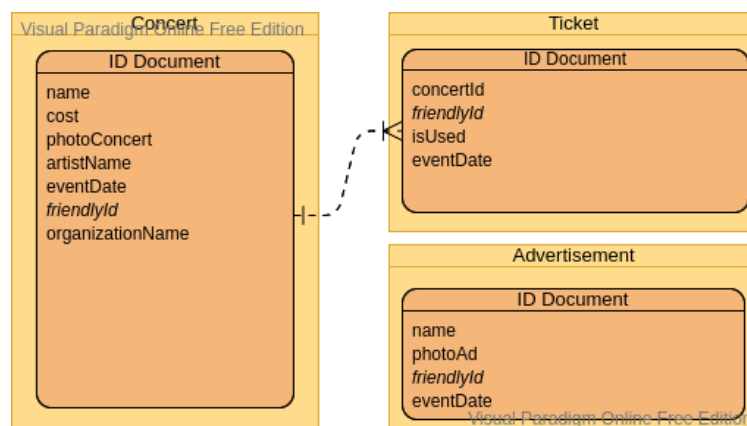
Este patrón de desarrollo nos propone dividir nuestro proyecto en distintas capas, donde cada una de ellas solo deberá de depender de sus capas interiores, sin conocer nada de las capas exteriores. Esto es posible gracias al uso de las interfaces que nos permite exponer los métodos que necesitamos de una implementación, ahorrando muchas líneas de código y problemas relacionados a la hora de implementar algo sin seguir un patrón de diseño. Cabe mencionar que este proyecto en particular se ha beneficiado de esta ventaja de manera significativa durante las fases de desarrollo y pruebas, ahorrando así tiempo y ganando en flexibilidad para futuros usos.

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* [25] 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 obteniendo un mayor reciclaje de código. Con la unión de estos dos patrones de diseño obtenemos un código limpio y altamente reciclable.

### 6.3. Modelo de datos

*Cloud Firestore* [5] 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 que se muestra a continuación representa el estado actual de la base de datos *NoSQL*:



Como puede verse, 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.

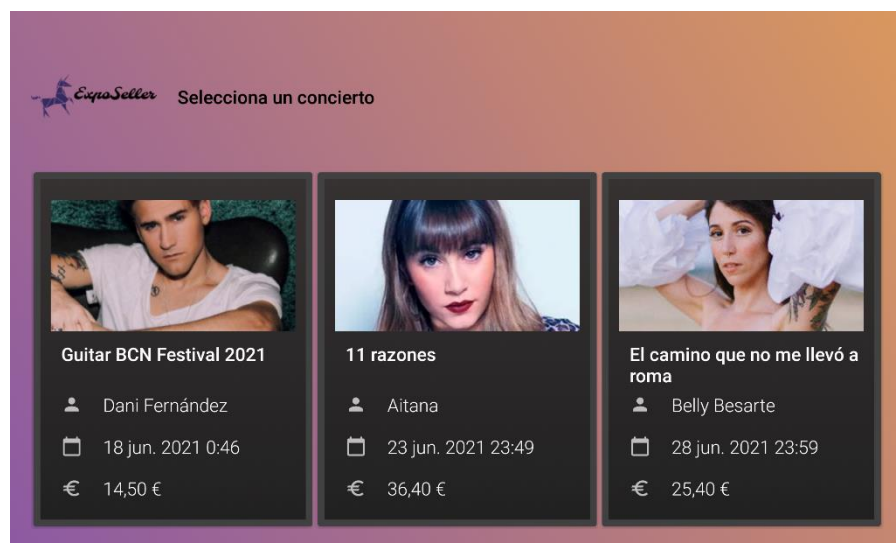
## 6.4. Vistas

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


### 6.4.1. Menú principal



### 6.4.2. Venta de tickets



### 6.4.3. Consulta de horarios

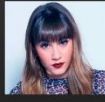
**Conciertos en el mes de junio de 2021**

<

Junio de 2021

>

L	M	X	J	V	S	D
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				



**11 razones**  
Aitana  
36,40 €

CANCELAR OPERACIÓN

### 6.4.4. Insertar monedas

**Introduzca el importe estimado**



**Concierto escogido: 11 razones**  
Importe estimado: 36,40 €  
Importe introducido: 36,75 €  
Cantidad a devolver: 0,35 €

Introduzca el importe estimado en monedas

CANCELAR OPERACIÓN

#### 6.4.5. Recoger Ticket

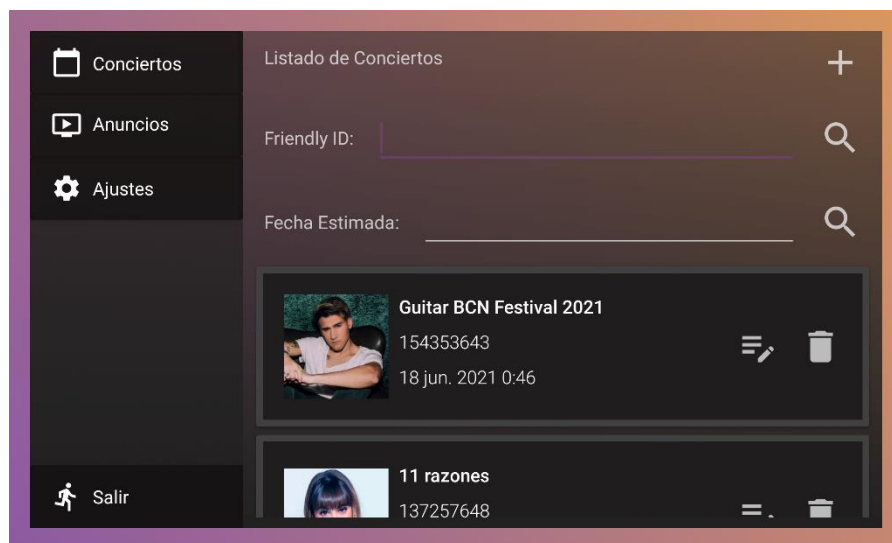


#### 6.4.6. Inicio de sesión al panel de administrador

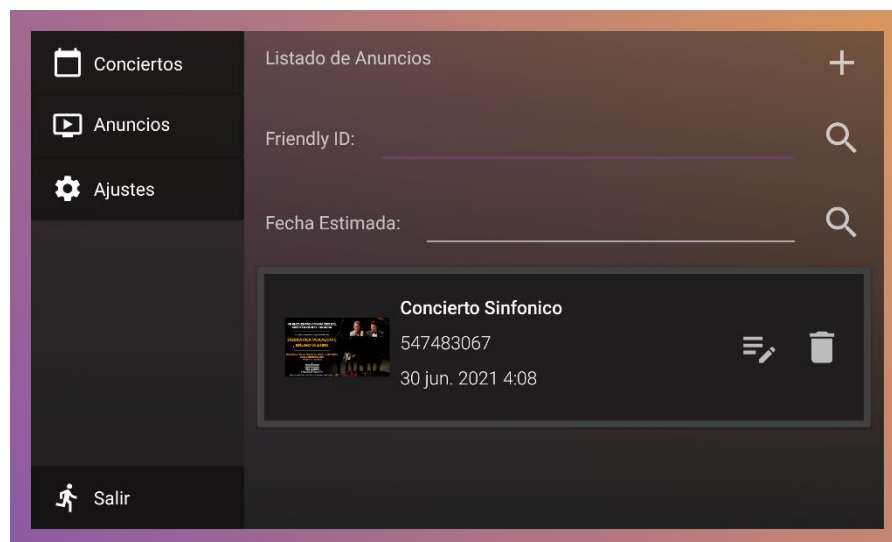




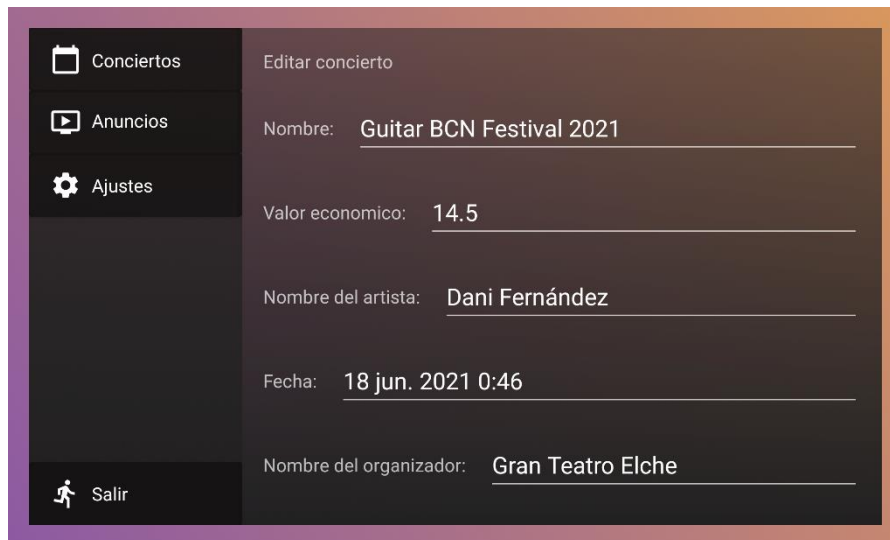
#### 6.4.7. Menú para mostrar los conciertos vigentes



#### 6.4.8. Menú para mostrar los anuncios vigentes



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



Editar concierto

Conciertos

Anuncios

Ajustes

Salir

Nombre: Guitar BCN Festival 2021

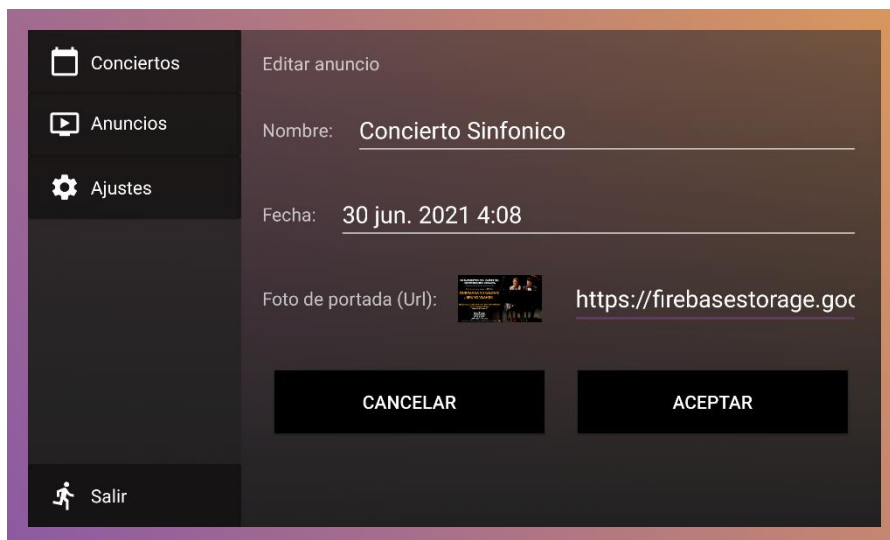
Valor economico: 14.5

Nombre del artista: Dani Fernández

Fecha: 18 jun. 2021 0:46

Nombre del organizador: Gran Teatro Elche

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



Editar anuncio

Conciertos


Anuncios

Ajustes

Salir

Nombre: Concierto Sinfonico

Fecha: 30 jun. 2021 4:08

Foto de portada (Url):  https://firebasestorage.goc

CANCELAR

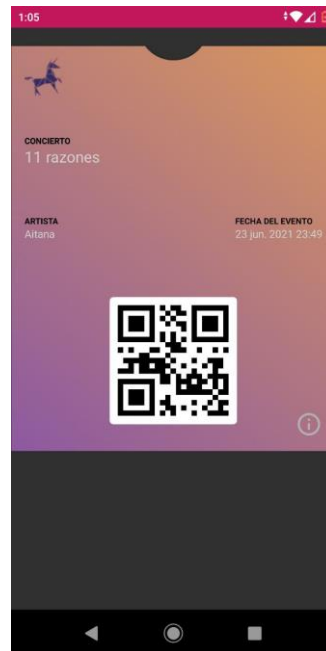
ACEPTAR

## 6.5. Diseños secundarios

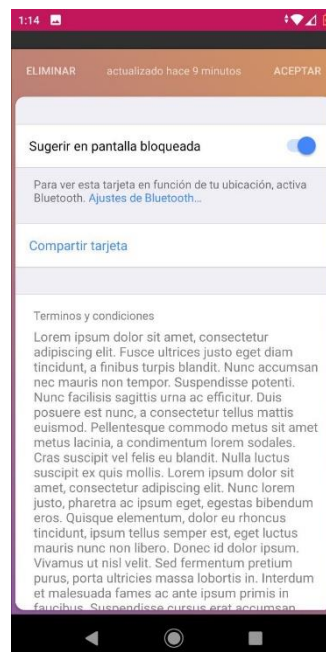
A continuación mostraremos como se muestra el ticket virtual en formato *Apple Passbook* con la aplicación para Android de *WalletPasses* [26].

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

### 6.5.1. Ticket de conciertos (Vista delantera)



### 6.5.2. Ticket de conciertos (Vista trasera)



## 7. IMPLEMENTACIÓN

### 7.1. 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 han 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 basadas 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 ágiles con componentes físicos que puedan ser utilizados durante una demo del proyecto. El integrador de nuestro proyecto no se verá limitado a tener que usar este hardware, pues podrá usar el que le convenga.

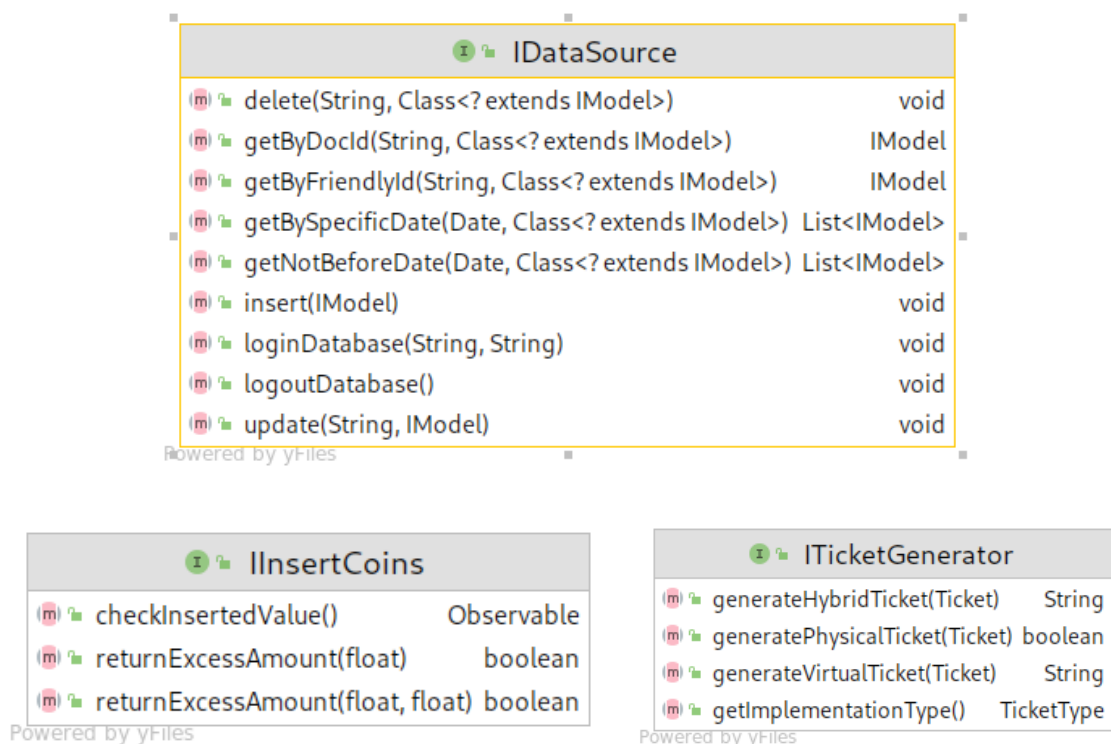
Para el validador se ha empleado el mismo equipo usado para la implementación del sistema, dado que este es suficiente para iniciar *JavaFX* [11] donde mostrará el patrón de colores implementado para la demostración.

## 7.2. Desarrollo

Una vez configurado el entorno de desarrollo se iniciaron los proyectos para el Validador, basado en el sistema de gestión de proyectos *Maven* [27], y la interfaz gráfica, proyecto basado en *Gradle* [28] y destinado para *Android 8.1* (API 27).

### 7.2.1. 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 han definido 3 interfaces:



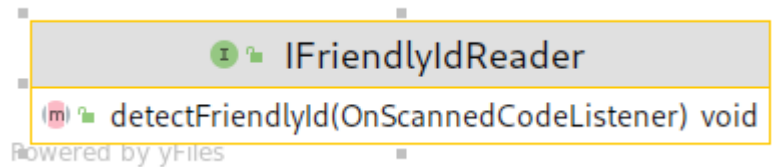
Esto nos permitirá desacoplar el código de las implementaciones permitiendo una alta reusabilidad de los componentes base del sistema. Como mención especial, tenemos la interfaz *IDataSource*, que dispondrá de una implementación para *Cloud Firestore* [5] adecuándolo a las necesidades de la aplicación.

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 y, en caso de que queramos usar hardware real, simplemente tendremos que implementarlo.

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

### 7.2.2. 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:

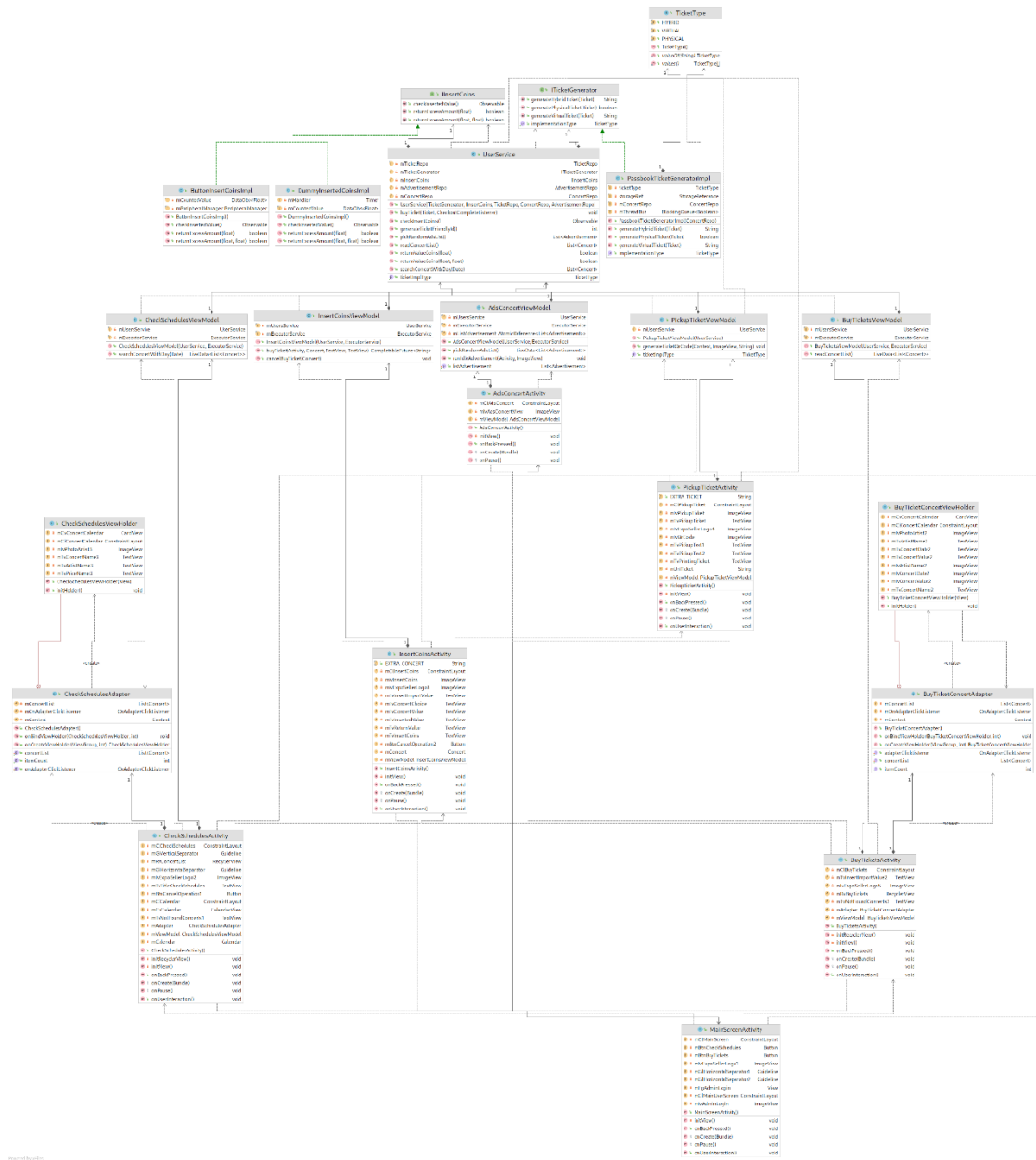


Esto nos permitirá a parte desacoplar la lógica de las implementaciones respecto a la lógica de negocio, aparte de utilizar solo las características que necesitemos de las utilidades escogidas para esta parte del sistema.

En el caso de la interfaz `IValidator`, tenemos una implementación que lanza una consulta a *Cloud Firestore* [5] y nos indica, a partir del 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 (*friendlyId*) a través de la webcam del dispositivo. En caso de detectar el código QR, este llamará al evento pasado por parámetro del método.

### 7.2.3. 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 la lógica de la clase de *UserService*, salvo en el caso de que necesitemos incorporar lógica de la capa de presentación al *ViewModel*.

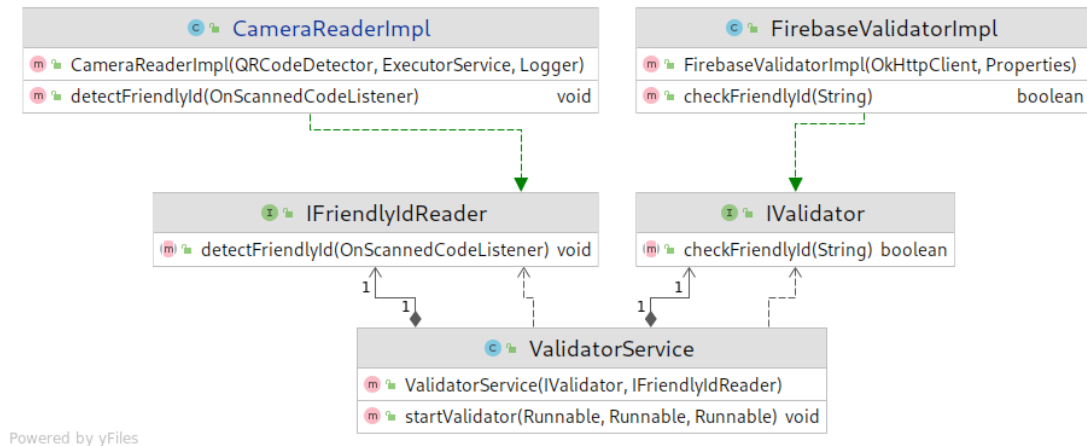
Para hacer las operaciones de inyección de dependencias utilizaremos la librería *Dragger Hilt* [20], que nos permitirá recuperar una instancia del *UserService*, del *ExecutorManager*, y otros, sin tener que estar creando nuevas instancias adicionales.

[illegible]

Para hacer las operaciones de inyección de dependencias utilizaremos la librería *Dragger Hilt* [20], que nos permitirá recuperar una instancia del *AdminService*, del *ExecutorManager*, y otros, sin tener que estar creando nuevas instancias adicionales.



### 7.2.5. Desarrollo del validador de tickets



En el caso del desarrollo del validador se ha optado por hacer una estructura lógica con componentes fácilmente reemplazables, 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. Sobre el hardware real, este tipo de señal puede implementarse como una alerta sonora o un movimiento físico, reduciéndose así la dependencia de *Java FX* [11] a puramente demostrativa de las capacidades del sistema.

Complementando lo anteriormente dicho, realmente la lógica de la validación reside tanto en la capa de servicio, siendo esta la que tiene el mayor grado de abstracción, como en las implementaciones más externas al núcleo de la aplicación, donde dispone de la lógica a bajo nivel para interactuar con los componentes.

Para las operaciones de inyección de dependencias utilizaremos la librería *Weld SE* [21], 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 que hacen uso de esta librería, en este caso *ExpoSellerApplication*, demostrando así el poder real que tiene este tipo de arquitecturas aplicadas a proyectos listos para producción.

## 8. PRUEBAS

### 8.1. End-to-End Testing



Las pruebas *End-to-End* es una técnica en 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 fallos, la contribución al funcionamiento correcto de todo el sistema, expandir la cobertura de pruebas disponibles y reducir 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 puedan ocurrir en la comunicación entre componentes y dentro de la lógica de estos. También nos servirá para comprobar el correcto funcionamiento con los servicios de *Cloud Firestore* [5], *Firebase Authentication* [4] y *Cloud Storage* [6].

## 8.2. 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*.

Los dispositivos sobre los que se han concentrado la mayoría de pruebas han sido los siguientes:

- Google Pixel XL (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.

La interfaz de usuario está pensada para ejecutarse sobre dispositivos tipo la *Raspberry Pi 3 Model B*, pues en este se han centrado la mayoría de pruebas que ha recibido esta parte del sistema.

En la siguiente página, describiremos qué 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 el terminal deja de disponer 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 válido, no ejecuta operaciones adicionales	OK

### 8.3. Evaluación

Durante la fase de pruebas se tuvieron en cuenta las diversas opiniones de amigos, compañeros de trabajo y 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 de estas pruebas, mejorando así las sensaciones de cara al usuario que transmite el sistema.

## 9. CONCLUSIONES

### 9.1. Conclusiones

Tras la realización de este proyecto, siento una gran satisfacción a nivel personal, dado la abstracción que poseía, 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 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...) 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 del tiempo disponible. *“Ser profesor en estos tiempos de pandemia es cosa de superhéroes.”*

### 9.2. 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 con discapacidad. 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 listo para exponer en una demostración del proyecto expuesto a lo largo de los anteriores puntos.

## 10. 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., «Dragger Hilt,» [En línea]. Available: <https://dagger.dev/hilt/>.
- [21] Red Hat, Inc., «Weld SE,» [En línea]. Available: <https://weld.cdi-spec.org/>.
- [22] Google, Inc., «Material Icons,» [En línea]. Available: <https://material.io/icons>.
- [23] K. Abhiromsawat, «Vending Machine,» [En línea]. Available: <https://www.flaticon.com/packs/vending-machine-5>.
- [24] R. C. Martin, Clean Architecture, Addison-Wesley, 2017.
- [25] Google, Inc., «Android Jetpak,» [En línea]. Available: <https://developer.android.com/jetpack>.
- [26] Wallet Passes Alliance, «WalletPasses,» [En línea]. Available: <https://play.google.com/store/apps/details?id=io.walletpasses.android>.
- [27] Apache Software Foundation, «Maven,» [En línea]. Available: <https://maven.apache.org/>.



[28] Gradle, Inc., «Gradle,» [En línea]. Available: <https://gradle.org/>.

## 11. ANEXOS

### 11.1. Manual de usuario administrador

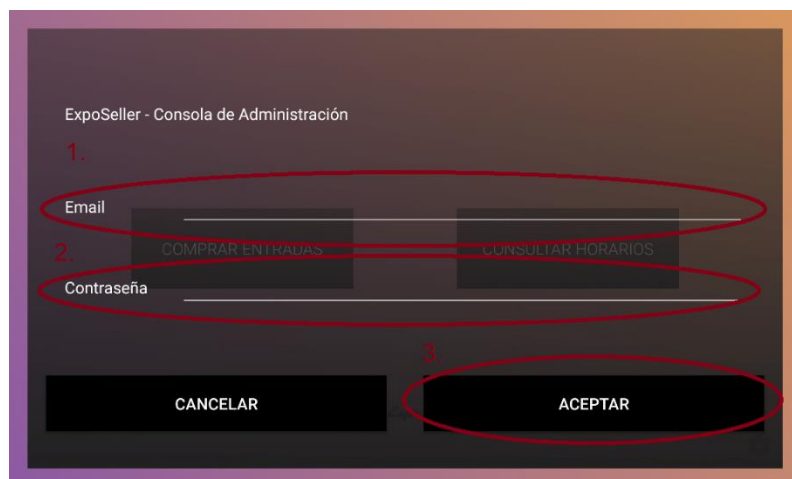
En este anexo explicaremos como usar correctamente el panel de administrador que está integrado con la interfaz gráfica del sistema:

#### 11.1.1. Inicio de sesión en el sistema como administrador

El inicio de sesión en el sistema nos acreditará para poder manipular los valores de las colecciones de conciertos y anuncios. Para poder acceder a dicho menú, deberemos de pulsar sobre el icono de engranaje señalado a continuación:



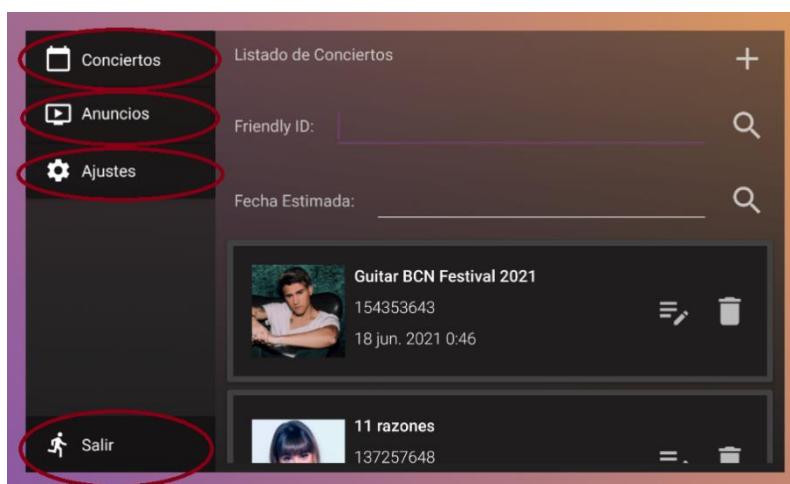
A continuación se nos abrirá un cuadro de diálogo donde podremos introducir las credenciales que nos hayan entregado previamente.



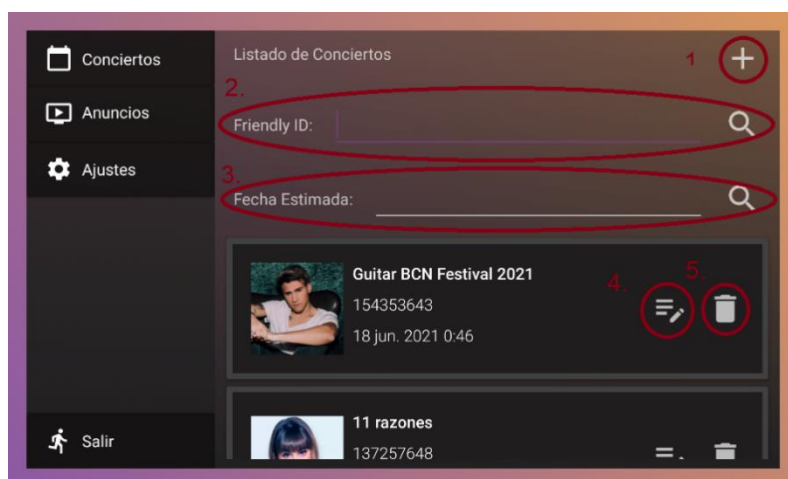
En el primer punto pondremos la dirección de correo electrónico del usuario, tras este primer paso deberemos de poner la contraseña del usuario, después debemos de presionar el botón de aceptar para entrar al panel.

### 11.1.2. La interfaz del listado de conciertos

Una vez se ha iniciado sesión en el sistema, podremos ver el listado de conciertos disponibles, podremos contemplar también que, en el lado izquierdo, disponemos de un listado de elementos que representan las diferentes opciones que disponemos en el menú, descritas con su respectiva etiqueta referenciando al tipo de acción que podemos esperar.



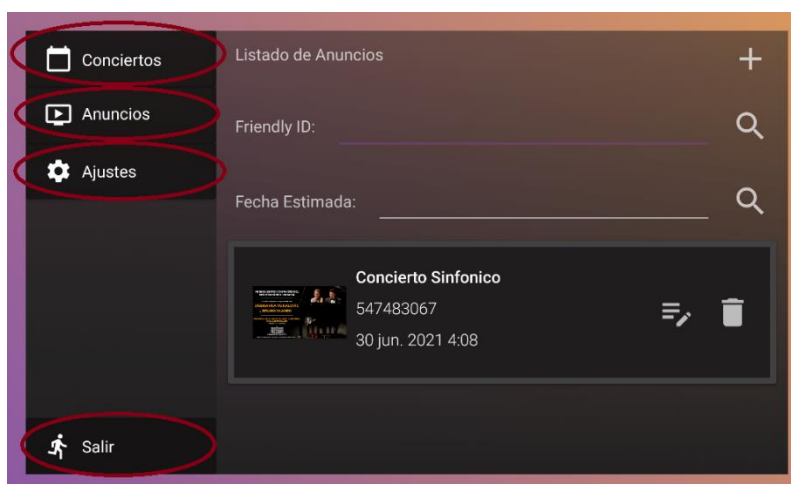
Volviendo con la descripción del listado podremos comprobar que disponemos de distintas opciones.



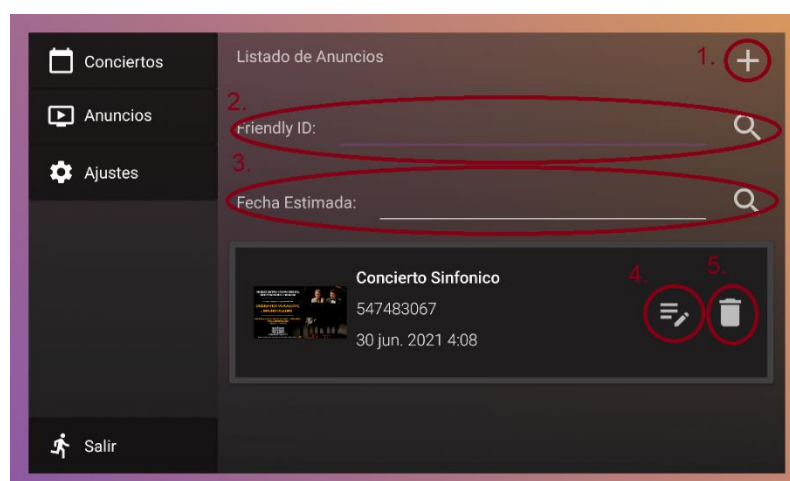
La primera nos servirá para añadir un nuevo concierto a la lista, cuyo formulario describiremos más adelante; la segunda opción nos permitirá buscar el concierto mediante el *friendlyId* que podemos encontrar en cada elemento de la lista, la tercera opción nos permitirá buscar los conciertos de una fecha específica, la cuarta y quinta opción nos permitirá interactuar con un elemento de la lista (como así sugieren los iconos, nos permitirá editar o eliminar el elemento).

### 11.1.3. La interfaz del listado de anuncios

Si hemos presionado sobre el botón de anuncios, podremos ver el listado de anuncios disponibles, podremos contemplar también que, en el lado izquierdo, disponemos de un listado de elementos que representan las diferentes opciones que disponemos en el menú, descritas con su respectiva etiqueta referenciando al tipo de acción que podemos esperar.



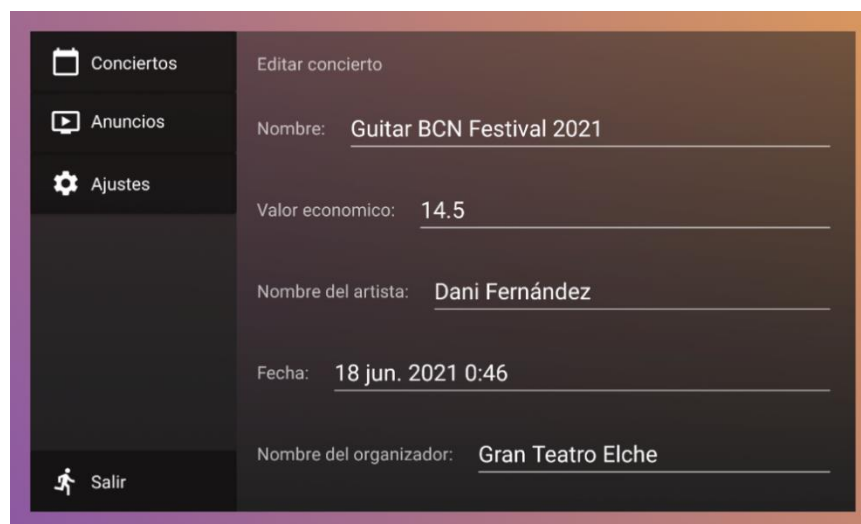
Volviendo con la descripción del listado, podremos comprobar que disponemos de distintas opciones.



La primera nos servirá para añadir un nuevo anuncio a la lista, cuyo formulario describiremos más adelante, la segunda opción nos permitirá buscar el concierto mediante el *friendlyId* que podemos encontrar en cada elemento de la lista, la tercera opción nos permitirá buscar los conciertos de una fecha específica, la cuarta y quinta opción nos permitirá interactuar con un elemento de la lista, como así sugieren los iconos, nos permitirá editar o eliminar el elemento.

#### 11.1.4. Añadiendo o editando un concierto

En caso de que nos hayamos decantado por editar o añadir un concierto, nos encontraremos con el mismo tipo de formulario, donde se nos pedirá insertar todos los valores propuestos para guardarlo en la base de datos.



Editar concierto

Nombre: Guitar BCN Festival 2021

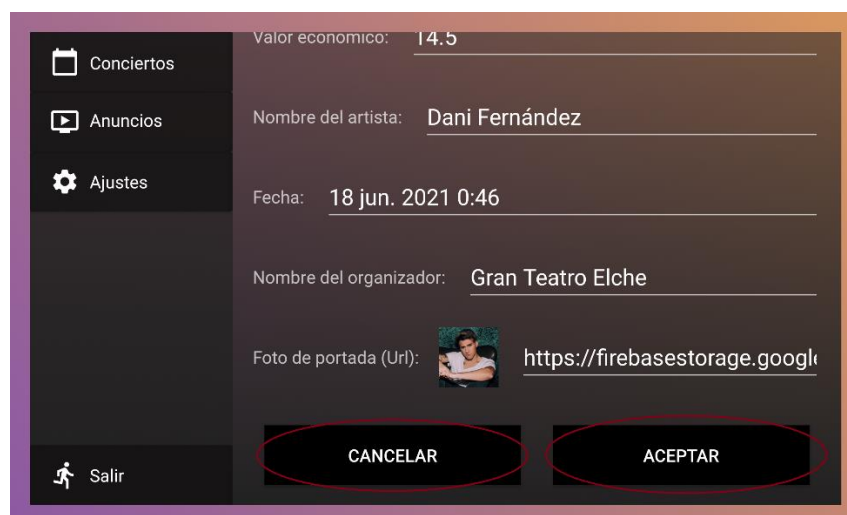
Valor economico: 14.5

Nombre del artista: Dani Fernández

Fecha: 18 jun. 2021 0:46

Nombre del organizador: Gran Teatro Elche

Una vez introducidos los valores propuestos, le podremos dar a aceptar para subir el nuevo documento (o en caso de estar actualizando, el concierto existente) a la base de datos, los cambios aparecerán en el listado tan pronto como se haya terminado de ejecutar la operación dentro de la base de datos.




Valor economico: 14.5

Nombre del artista: Dani Fernández

Fecha: 18 jun. 2021 0:46

Nombre del organizador: Gran Teatro Elche

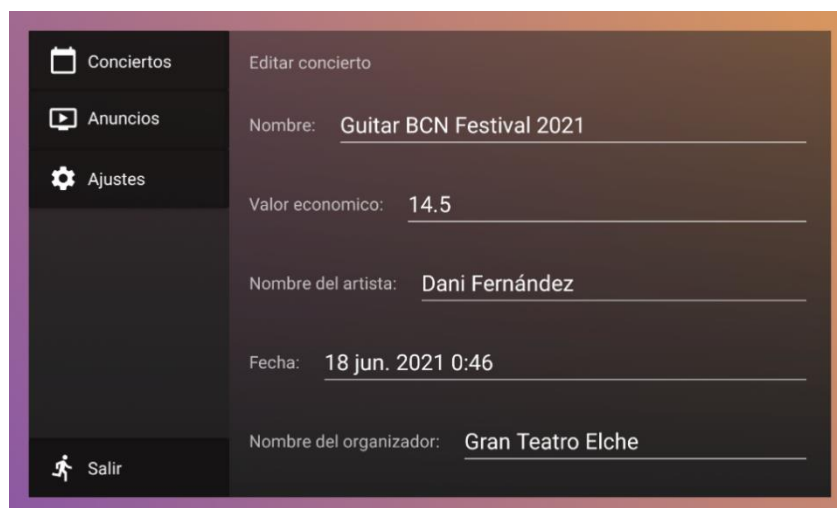
Foto de portada (Url):  <https://firebasestorage.google>

**CANCELAR** **ACEPTAR**

Así mismo, podremos presionar el botón cancelar para que nos devuelva al listado de conciertos sin producir ningún tipo de cambio. También podremos navegar con el menú izquierdo, constando como que se ha cancelado la operación.

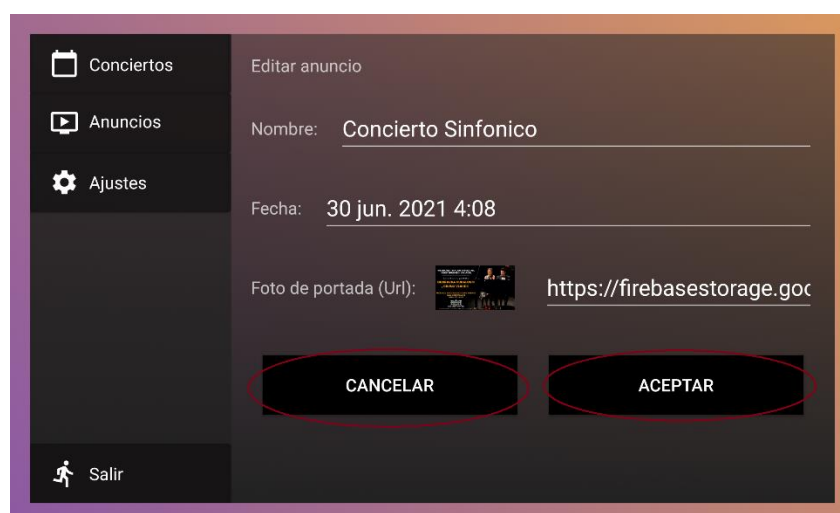
#### 11.1.5. Añadiendo o editando un anuncio

En caso de que nos hayamos decantado por editar o añadir un anuncio, nos encontraremos con el mismo tipo de formulario, donde se nos pedirá insertar todos los valores propuestos para guardarlo en la base de datos.



The screenshot shows a web application interface with a sidebar on the left containing icons and labels for 'Conciertos', 'Anuncios', 'Ajustes', and 'Salir'. The main content area is titled 'Editar concierto' and contains several input fields: 'Nombre:' with the value 'Guitar BCN Festival 2021', 'Valor economico:' with the value '14.5', 'Nombre del artista:' with the value 'Dani Fernández', 'Fecha:' with the value '18 jun. 2021 0:46', and 'Nombre del organizador:' with the value 'Gran Teatro Elche'.

Una vez introducidos los valores propuestos, le podremos dar a aceptar para subir el nuevo documento (o en caso de estar actualizando, el anuncio existente) a la base de datos, los cambios aparecerán en el listado tan pronto como se haya terminado de ejecutar la operación dentro de la base de datos.



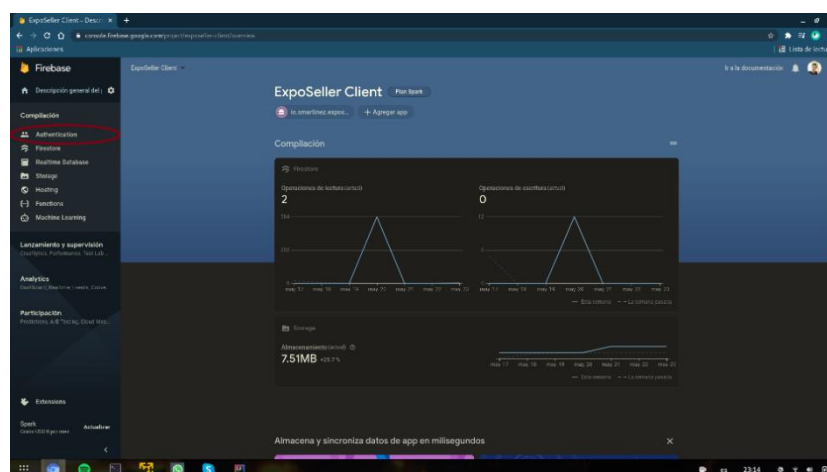
The screenshot shows a web application interface with a sidebar on the left containing icons and labels for 'Conciertos', 'Anuncios', 'Ajustes', and 'Salir'. The main content area is titled 'Editar anuncio' and contains several input fields: 'Nombre:' with the value 'Concierto Sinfonico', 'Fecha:' with the value '30 jun. 2021 4:08', and 'Foto de portada (Url):' with the value 'https://firebasestorage.goc'. Below the input fields, there are two buttons: 'CANCELAR' and 'ACEPTAR', both of which are circled in red.

Así mismo, podremos presionar el botón cancelar para que nos devuelva al listado de anuncio sin producir ningún tipo de cambio. También podremos navegar con el menú izquierdo, constando como que se ha cancelado la operación.

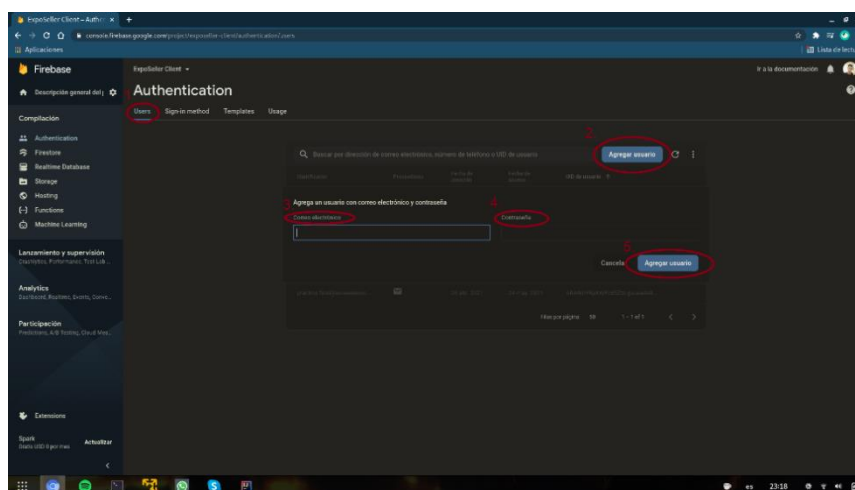
### 11.1.6. Añadiendo nuevos usuarios administradores

En este proyecto estamos usando como backend los servicios de *Firebase*, entre los que se destaca *Cloud Firestore* [5] y *Firebase Authentication* [4]. A modo de mantener la gestión de los usuarios centralizada, se ha optado por usar el panel de administración de *Firebase*.

Para ello debemos de ir a <https://console.firebase.google.com/> y seleccionar el proyecto que corresponda. Debemos de encontrarnos algo como lo siguiente:



Si nos fijamos en el lateral izquierdo podremos observar un campo llamado Autenticación, si presionamos sobre él nos enviará a la interfaz de Autenticación:

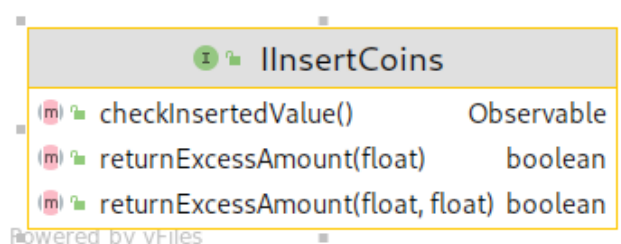


Si presionamos sobre la pestaña de usuarios y después sobre el botón “Agregar Usuario”, se nos abrirá el formulario para insertar el usuario y la contraseña del nuevo usuario, agregando este al registro tras pulsar el botón “Añadir usuario”.

## 11.2. Manual para desarrolladores

En este anexo explicaremos como implementar algunas de las interfaces del sistema para adaptarlo a un nuevo hardware, esto nos permitirá tener mayor flexibilidad a la hora de diseñar un producto comercial derivado de este proyecto.

### 11.2.1. Implementando la interfaz *IInsertCoins*



En el caso de la interfaz *IInsertCoins*, disponemos de una implementación basada en el tiempo, donde cada segundo va insertando 0,75€ al contador interno de la implementación, esto está muy bien en el caso de que deseemos hacer una demostración del proyecto, o simplemente para usarlo durante depuración, pero esta implementación no se ajustaría a los requisitos de un producto basado en este proyecto.

Para compensar este hecho, deberemos de crear una clase en el paquete *insertcoins*, que deberá implementar la interfaz *IInsertCoins* y deberá de tener el decorador `@Singleton` encima de la declaración de la clase, así también deberá de tener en cuenta el añadir la anotación `@Inject` encima del constructor por defecto.

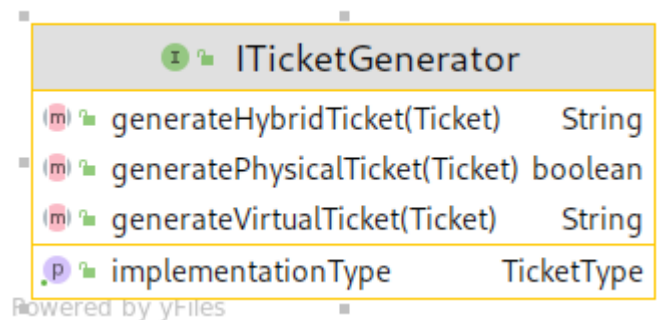
Dado que necesitamos trabajar con datos primitivos observables desde un hilo diferente al principal, se ha preparado la clase genérica *DataObs* para poder hacer uso de manera observable de estas clases primitivas, facilitando así el trabajo al desarrollador.

En este caso, no hace falta trabajar con hilos secundarios para evitar bloquear el hilo principal, pues ya se ha tenido en cuenta este hecho desde las capas interiores de la aplicación.

Por último, nos tocará cambiar la implementación por defecto en la clase *ExpoSellerBindings*, dado que *Dragger Hilt* [20] trabaja con un grafo de dependencias que debe ser declarado de forma explícita.



### 11.2.2. Implementando la interfaz ITicketGenerator



Aunque este proyecto ha tenido como objetivo el desarrollar un sistema adaptado a los tiempos de lo digital, puede darse el caso de que, por requerimientos del producto, debamos tener algún tipo de impresión de tickets, o incluso un modelo híbrido. Este proyecto ha tenido en cuenta la posibilidad de que se diera esta casuística.

Entonces, para adaptarnos a los nuevos requerimientos, deberemos de crear una nueva clase en el paquete `ticketgenerator` que extienda de la interfaz `ITicketGenerator` y que tenga el decorador `@Singleton` encima de la declaración de la clase. Así mismo, en el constructor por defecto, deberá de tener la anotación `@Inject`.

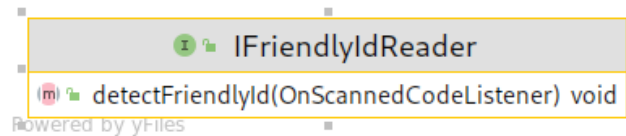
Sobre como implementar los métodos, deberemos de escoger el tipo de implementación que deseamos. En la enumeración `TicketType` tendremos los diferentes tipos que existen. Pues deberemos de definir uno como retorno a la propiedad `implementationType`.

Una vez definido el tipo de ticket que deseamos deberemos de implementar la lógica sobre el método destinado al tipo de ticket. El resto de los métodos deberán de lanzar la excepción de `UnsupportedOperationException`, indicando que el método al que se ha pretendido llamar no está implementado.

En este caso no hace falta trabajar con hilos secundarios para evitar bloquear el hilo principal, pues ya se ha tenido en cuenta este hecho desde las capas interiores de la aplicación.

Por último, nos tocará cambiar la implementación por defecto en la clase `ExpoSellerBindings`, dado que *Dragger Hilt* [20] trabaja con un grafo de dependencias que debe ser declarado de forma explícita.

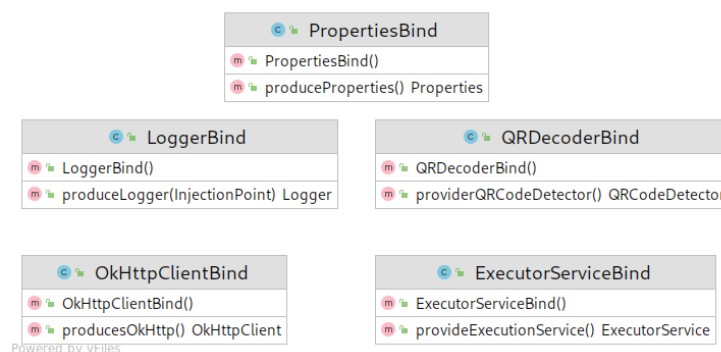
### 11.2.3. Implementando la interfaz IFriendlyIdReader



En el caso de la interfaz de *IFriendlyIdReader*, disponemos de una implementación por defecto basada en OpenCV [9]. En este proyecto tenemos de la flexibilidad de poder eliminar esta implementación (sin tener que refactorizar) y agregar la nuestra.

Tras eliminar la implementación por defecto, deberemos de, en el mismo paquete, crear una nueva clase que implemente la interfaz *IFriendlyIdReader*. Al mismo tiempo necesitaremos añadirle el decorador *@ApplicationScoped*. Que informará al inyector de dependencias de que existe una implementación para *IFriendlyIdReader*. También deberemos de añadir la anotación *@Inject* encima del constructor de la implementación. A partir de este punto podremos añadir nuestro código al método *detectFriendlyId()*.

Es aconsejable que todas las operaciones que realice este método sean sobre un hilo en el fondo, esto nos evitará los problemas que pueda suponer bloquear el hilo principal de la aplicación, complicando así el procesamiento de la información de las entradas.



Para poder acceder a instancias de las clases dentro de la aplicación, podemos hacer uso de las anotaciones propuestas por *Weld SE* [21], el cual es el quien gestiona todo lo relacionado con las dependencias de clases es el que pueda tener otra clase, de forma dinámica. Puedes consultar el paquete de *binding* para averiguar qué tipos de implementaciones se encuentran disponibles.

### 11.3. Manual de instalación

Para la instalación de la demo proporcionada para este proyecto, necesitaremos dos dispositivos.

Para el caso de la interfaz de usuario, nos servirá con instalarlo en un emulador, o en un dispositivo de 10" mínimo, con la versión de Android en la versión 8.1 y con las opciones de depuración por USB activadas previamente.

Para el caso del validador necesitaremos instalarlo en un dispositivo con Java SE 11 y que disponga de una cámara web, siendo así aconsejable ejecutarlo en un portátil, a nivel de sistema operativo es compatible con Windows 10, Linux y macOS pero no con el resto de variantes de UNIX.

Para descargar los dos ejecutables, procederemos a acceder a la URL [https://1drv.ms/u/s!AmY3X7cJi1yAiZsOO\\_u0L22Bs4qRcA?e=zu2clC](https://1drv.ms/u/s!AmY3X7cJi1yAiZsOO_u0L22Bs4qRcA?e=zu2clC), donde nos encontraremos los ficheros [ExpoSellerClient.apk](#) y [ExpoSellerValidator.jar](#)

Asumiendo de que los requisitos previos se han cumplido, nos descargaremos el archivo [ExpoSellerValidator.jar](#) y lo ejecutaremos mediante la línea de comando del siguiente modo:

```
PS C:\Users\Sergio Martinez> java -jar .\ExpoSellerValidator.jar
```

De esta manera deberemos de tener una ventana en blanco abierta esperando a que intentemos validar un ticket. Para desplegar la interfaz de usuario, deberemos de tener conectado el dispositivo a nuestro equipo y ejecutar los siguientes comandos procedentes del Sdk de Android.

```
PS C:\Users\Sergio Martinez> adb install .\ExpoSellerClient.apk  
PS C:\Users\Sergio Martinez> adb shell am start -n  
io.smartinez.exposeller.client/.ui.mainscreen.MainScreenActivity
```

De esta manera tendremos instalado la interfaz de usuario del sistema esperando a recibir peticiones de compra.

Para el dispositivo del usuario, se recomienda para esta demo que se utilice un smartphone basado en Android y se instale la aplicación WalletPasses [26], que nos servirá para gestionar los tickets. Esto es debido a que los certificados compatibles para los dispositivos basados en iOS no están generados debido a las limitaciones de Apple y su programa de desarrolladores, excluyéndolos así de esta demo técnica pero no del producto como tal.