Blazebooks

Mounir Zbayr Samadi

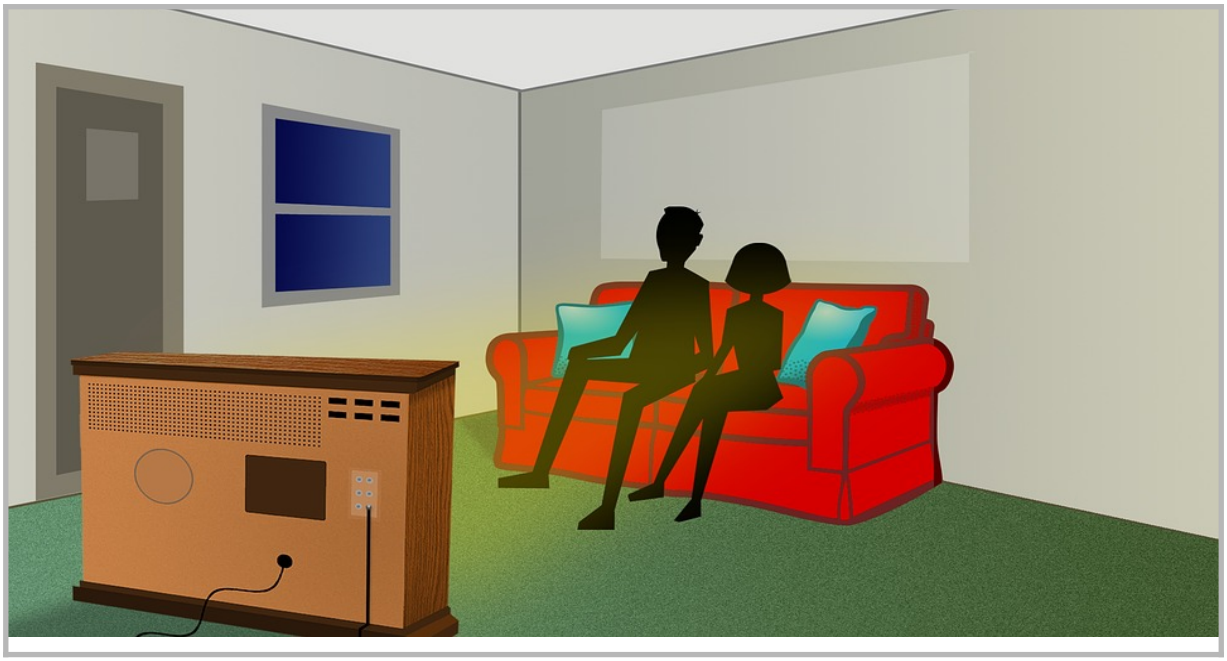
Víctor González Gaitero

D.A.M. 2º

Índice

Introducción

En los últimos tiempos, plataformas como HBO, Netflix, Spotify o Amazon Prime han vivido un apogeo que no para de crecer. Todas ellas enfocan su contenido en películas, músicas o series. Nosotros seremos partícipes de este apogeo dando un pequeño giro, ofreciendo una **plataforma cuyo contenido se basará en libros con un formato de lectura rápida.**



Blacebooks permite llenar esos tiempos muertos con letras y cultura. Por ello, ofrecemos diferentes géneros literarios, lecturas técnicas, e incluso libros interactivos. Además, acompañamos las lecturas con música e incorporamos un sistema de recomendaciones personalizadas para el usuario.

Muchos de los contenidos son originales de la plataforma, como ocurre con plataformas ya citadas anteriormente. Y es aquí donde se diferencia nuestra idea ya que, a diferencia de servicios como Kindle (Amazon) que ofrecen libros por una mensualidad, nosotros ofrecemos libros exclusivos de la plataforma. La plataforma cuenta con contenido gratuito muy amplio y variado. Sin embargo, implementa también una **modalidad premium**, la cual da acceso al usuario al contenido exclusivo.

Como ocurre con todo en la vida, este proyecto no tiene una única motivación. Evidentemente, el objetivo primero es el de **concluir con éxito** nuestro periodo de aprendizaje en el IES Barajas. Por otro lado, buscamos **fomentar y modernizar la lectura**, a la par que adquirir nuevos conocimientos y experiencia.

Huelga decir que la aplicación tiene como fin quedar lista para su **distribución y posterior monetización**, comprobando si es o no una propuesta viable para iniciar **nuestra entrada en el mercado**.

Metodología & Desarrollo del Proyecto

Metodología de Desarrollo: Kanban + GIT

Para este proyecto hemos escogido utilizar la metodología **Kanban**. Esta forma de trabajar es muy popular en muchas empresas y corporaciones, combinándola con otras metodologías como SCRUM.

Kanban funciona mediante una tabla, en la cual tendremos tres columnas: tareas **para hacer**, tareas **en proceso** y tareas **hechas**. Cada vez que una tarea cambie de estado de una tarea hay que moverla a su fase correspondiente. Como ampliación, hemos incluido dos subestados para las tareas en proceso: **trabajándose** y **a la espera**. Esto es debido a que a veces tenemos que dejar de trabajar en una tarea por algún motivo, pero esta tarea aún no se ha terminado.

Para implementarla, hemos creado un documento en Google Drive al cual añadimos las tareas con las que estábamos trabajando. A partir de ahí, hemos actualizado el documento dinámicamente, incluyendo la inicial de la persona encargada de cada tarea entre paréntesis. Simultáneamente hemos ido actualizando el repositorio de **GIT**.

Método Kanban			
Para hacer	En proceso		Hecho
	Trabajándose	A la espera	
	Incluir content-description en las vistas (v	**Arreglar setText Username y Email (MainActivity)**	Diseño y flujo de la app (v)
	Lector de Libros (M)	**Implementar Settings (V)**	Implementación de la búsqueda (v)
Añadir Filtros a la Búsqueda			Icono de la App (v)
Arreglar el flujo de navegación	Implementar DAO(m)		Implementación de Firebase (m)
Implementar luz de lectura			Implementación de Coil, spin-kit (v)
Mejorar plantilla correo recuperacion de contraseña			Autenticación usuarios (m)
Añadir lista de Favoritos			Método recuperación contraseña(m,v)
Añadir lista de descargados			Descarga archivos Firebase (m)
Añadir avatares de usuario			Implementar Idiomas (v)
Añadir content-description a la vista			
Acceder con Google (?)			

Escogimos esta metodología porque no requiere de una fase de **planificación** de tareas muy compleja. Además, su **implementación** es sencilla, rápida, y no quita mucho **tiempo** actualizar/revisar el documento.

Utilizar esta metodología nos ha permitido tener una mejor **perspectiva** del estado del desarrollo del proyecto, así como saber cuáles eran las tareas que cada miembro del equipo estaba realizando. Además, ha evitado que el proyecto se estanque.

Arquitectura: MVVM

Tenemos que admitir que la fase inicial del desarrollo se implementó con **MVC**. Lo escogimos porque teníamos mucha familiaridad con este sistema y porque no conocíamos otro. Sin embargo, durante el

desarrollo nos dimos cuenta de que una arquitectura **MVVM** proporcionaría grandes beneficios a nuestra aplicación.

Esta arquitectura permite separar la vista de la parte lógica, dejando un código más limpio y fácil de mantener. Al combinar la arquitectura MVVM con la **inyección de dependencias** y el patrón **factory** consigue una gran independencia entre las clases y el código resulta incluso más fácil de mantener y/o modificar.

Para migrar la aplicación a MVVM realizamos **dos fases**. Realmente podría haberse realizado en una, aunque la implementación fue mutando según fuimos aprendiendo más acerca de la arquitectura MVVM.

Primera Fase de la Implementación

La primera fase consistió en implementar el **viewmodel** de todas las vistas. Además, se implementó la **inyección de dependencias**. Separamos la parte lógica de la vista y conectamos la vista con el activity host mediante una **interfaz**.

A su vez, el activity host se conecta con la vista utilizando **Data Binding**. La etiqueta **<data>** permite conectar la vista con el viewmodel y acceder directamente a sus atributos y métodos desde el xml. La mayoría de **eventos onClick** fueron sustituidos por métodos del objeto viewmodel.

Segunda Fase de la Implementación

En la segunda fase , pretendíamos hacer aún más independientes la parte lógica y la vista. Para ello, se **eliminó la interfaz** de conectaba el viewmodel y el activity host: ahora el activity host contenía un **objeto viewmodel** que le permite llamar a los métodos de la parte lógica sin necesidad de interfaces.

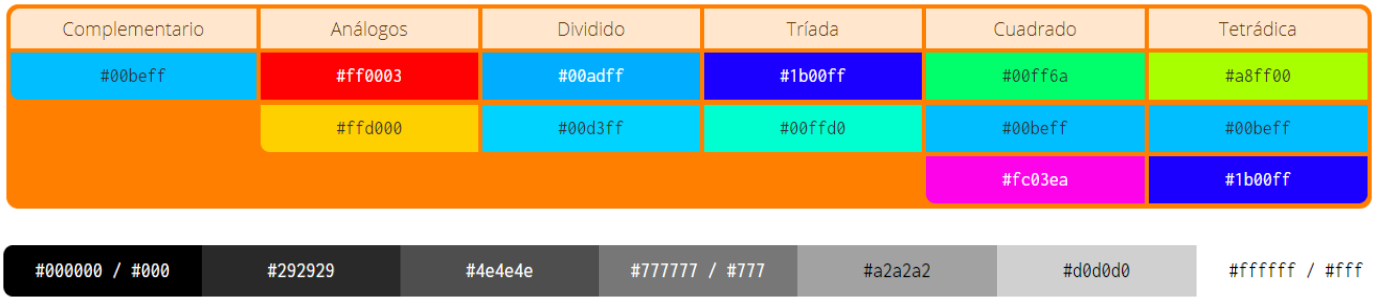
El **Data Binding** del xml se simplificó. En muchas de las vistas **no era necesario** incluir la etiqueta <data> con el viewmodel para acceder a los atributos y métodos desde la vista, ya que no requieren de actualización constante en tiempo de ejecución. Muchos de los **eventos onClick fueron "re-implementados".**

Diseño de la Interfaz de Usuario

La interfaz de usuario es una parte fundamental de las aplicaciones Android, ya que permite al usuario interaccionar, a parte de volverla más atractiva y amigable. Es por ello que se debe mostrar el contenido con el **patrón y los controles propios del UI de Android**, facilitando su uso al usuario y haciendo la aplicación más intuitiva.

Colores

Los colores de la aplicación no se han escogido al azar, sino siguiendo un esquema que toma como color principal el **naranja** (#ff80000). Por ello, los otros dos colores principales de la aplicación son el **rojo y el ámbar**. Para darle más juego a esto se han utilizado también diferentes colores de la **escala de grises, azules, morados, etc.**



Animaciones

Las animaciones son importantes, ya que ayudan a mantener al usuario orientado al UI. En nuestra aplicación, hemos usado tres maneras diferentes para animar el UI : Spin-Kit, Lottie y Android Animations.

Android-Spin-Kit

La primera implementación de animaciones vino de la mano de **Android-Spin-Kit**, una librería de animaciones de carga. Su implementación es muy sencilla: solo hay que añadir a Gradle la dependencia correspondiente.

Para **mostrar un elemento**, simplemente se debe hacer uso de la etiqueta SpinKitView dentro de un XML. Los diferentes **estilos** que podemos usar se encuentran en la documentación oficial de la librería, pudiendo hacer uso de una amplia variedad. El **bucle** se produce de manera automática. Tanto el **color** como el **diseño** del spinner se configura con atributos desde el XML, por lo que no es necesario codificar nada.

En un principio, usamos esta librería para **animar todas las cargas**, y así dar información al usuario. Con el avance del desarrollo, las cargas más importantes están animadas con **Lottie**, mientras que las menos importantes siguen estando animadas con esta librería.

Lottie

Lottie es una gran librería, usada en múltiples tecnologías y en desarrollo web, iOS y Android. Algunas tecnologías que hacen uso de esta librería son Flutter, React, Xamarin, Windows, Angular y After Effects.

La **implementación** es muy sencilla. Primero, se añaden las dependencias a Gradle. Una vez hecho esto, se debe añadir una etiqueta LottieAnimationView a la vista XML para incluir la animación. Aquí también incluiremos **atributos sobre la ejecución** de la misma: velocidad, en bucle, etc.

Las animaciones podemos descargarlas de la página oficial, en **formato Json**. No todas son libres de derechos, pero existe una gran variedad de animaciones hechas por la comunidad que sí lo son. Una vez descargadas, las animaciones se guardan en la carpeta **res/raw**. Desde el **código**, las animaciones pueden reproducirse una sola vez o en bucle, de forma inversa, con una velocidad determinada, o parándose en algún momento concreto de la misma. Tiene muchas opciones.

En nuestra aplicación, se hace uso de Lottie en la **carga principal**, poniendo en bucle la animación desde el XML. Como puede apreciarse, el atributo **lottie_autoPlay** es el que indica hace que la animación se ejecute de manera automática al mostrar la vista; el atributo **lottie_loop** indica que la animación está en bucle; **lottie_rawRes** indica qué recurso se está utilizando para la animación.

```
<com.airbnb.lottie.LottieAnimationView
    android:id="@+id/iv_splash"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:lottie_autoPlay="true"
    app:lottie_loop="true"
    app:lottie_rawRes="@raw/loading" />
```

También hacemos uso de la librería de una manera distinta en el **botón de añadir a favoritos** de la aplicación. En este caso, la animación no se inicia automáticamente, sino que espera a que se accione su **click listener** para hacerlo.

```
<com.airbnb.lottie.LottieAnimationView
    android:id="@+id/showBookBtnFav"
    android:layout_width="50sp"
    android:layout_height="50sp"
    android:layout_marginStart="20sp"
    android:layout_marginTop="5sp"
    android:layout_marginBottom="5sp"
    android:onClick="addFav"
    android:scaleType="center"
    app:lottie_rawRes="@raw/like" />
```

Dependiendo de si se está añadiendo o removiendo un libro de favoritos, la animación se reproducirá normal o de forma inversa. Para cambiar el sentido de la ejecución de la animación, su velocidad (speed) se pone a 1f (sentido normal) o -1f (sentido inverso). Luego se indica al botón que debe estar listo para que al refrescar la vista, se ejecute la animación una vez. Por último, se refresca la vista.

```
fun addFav(view: View) {
    liked = when (liked) {
        true -> {
            //TODO -> remove from favs
            showBookBtnFav.speed = -1f
            showBookBtnFav.playAnimation()
            Toast.makeText( context: this, getString(R.string.rm_favs), Toast.LENGTH_SHORT).show()
            false
        }
        false -> {
            //TODO -> add to favs
            showBookBtnFav.speed = 1f
            showBookBtnFav.playAnimation()
            Toast.makeText( context: this, getString(R.string.add_favs), Toast.LENGTH_SHORT).show()
            true
        }
    }
    view.refreshDrawableState()
}
```

Android Animations

Android incluye el trabajo con transiciones para animar fácilmente los cambios entre vistas. Para animar las vistas en tiempo de ejecución, se cambian algunos de sus valores de propiedad a lo largo del tiempo. Además, se incluyen animaciones predefinidas, aunque nosotros hemos creado las nuestras.

Todas las animaciones se guardan dentro de la carpeta **res/anim** en formato XML. A la hora de editar el XML para la animación, existen varias formas, dependiendo del uso que se le vaya a dar.

- Se puede utilizar **una única animación** para establecer la posición inicial de la vista y la posición final.
- Se pueden crear **dos animaciones**, que indiquen la posición desde donde parte la vista y a la cual se dirige.

```xml
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="500">
    <translate
        android:fromYDelta="50%p"
        android:interpolator="@android:anim/accelerate_decelerate_interpolator"
        android:toYDelta="0" />
    <alpha
        android:fromAlpha="0"
        android:interpolator="@android:anim/accelerate_decelerate_interpolator"
        android:toAlpha="1" />

</set>
```

En este caso, la animación mueve la vista **desde abajo hasta arriba**. A su vez, cambia la **transparencia** de la vista de 0 a 1 progresivamente, y dura medio segundo. En caso de querer hacer esto mismo, pero con dos animaciones, el resultado serían los dos ficheros XML siguientes:

```xml
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="300">
    <translate
        android:fromYDelta="0%"
        android:toYDelta="-100%" />
</set>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="300">
    <translate
        android:fromYDelta="100%"
        android:toYDelta="0%" />
</set>
```

La imagen de la izquierda indica el estado inicial del que se parte, mientras que la de la derecha indica el estado final. Según la manera de adjuntar una animación a una vista, se utilizará una de estas maneras. Existe una forma más, pero hablaremos de ella en el siguiente apartado.

Estas animaciones **se han usado de dos formas diferentes:** directamente en la vista XML o en tiempo de ejecución. De la primera forma se crea una animación especial y se la vincula a una vista desde el XML, de manera que al aparecer la vista, ejecute la animación.

Para implementar animaciones sobre el código hay varias maneras. En este caso, se anima un cambio de activity con el método overridePendingTransition. Como dijimos anteriormente, la animación hace uso de un estado inicial (primer parámetro) y un estado final (segundo parámetro) para ejecutarse. De modo que, cuando se lance el nuevo activity, este aparecerá por la derecha.

```
private fun goToPreferenceActivity() {
    startActivity(Intent( packageContext: this, SettingsActivity::class.java))
    overridePendingTransition(R.anim.slide_from_right, R.anim.slide_to_left)
}
```

Aquí se anima la aparición de un cuadro de diálogo de la misma manera. La implementación es ligeramente diferente.

```
fun setProfileImage(view: View) {
    mainActivityProfileImgFragment.visibility = View.VISIBLE
    supportFragmentManager.beginTransaction()
        .setCustomAnimations(R.anim.slide_from_right, R.anim.slide_to_left)
        .replace(R.id.mainActivityProfileImgFragment, ProfileImageDialog())
        .commit()
}
```

Layout Animations

Los layouts se pueden animar directamente en el XML. Lo primero es crear las animaciones utilizando la etiqueta LayoutAnimation. En este caso, la animación mueve el layout desde abajo a arriba, utilizando otro recurso de animación.

```
<?xml version="1.0" encoding="utf-8"?>
<layoutAnimation xmlns:android="http://schemas.android.com/apk/res/android"
    android:animation="@anim/down_to_up"
    android:animationOrder="normal"
    android:delay="15%" />
```

Para animar un layout, se incluye en la etiqueta XML el atributo layoutAnimation. En este caso, se le aplica la animación de desplazamiento lateral a un layout que será cargado en un fragment. Con tan solo incluir el atributo layoutAnimation, se logra el efecto deseado.

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layoutAnimation="@anim/layout_animation_right_to_left"
    android:layout_height="match_parent"
    android:background="@drawable/dark_background">
```

Custom RecyclerGridLayout Animado

Para animar un RecyclerGridLayout, no se puede añadir la animación como en el resto de casos. Android no reconoce la animación correctamente y da un error. Para resolver este problema hemos creado nuestra propia clase custom.

Implementación: Clase & Animación

Esta clase hereda de RecyclerView, pero sobreescribe dos de sus métodos: **setLayoutManager** y **attachLayoutAnimationParameters**. El primero se encarga de asegurar que el layout manager usado sea un GridLayoutManager. El segundo método es el encargado de establecer los parámetros del ViewGroup para que la animación funcione.

Por último, se debe crear la animación de una forma similar a la expuesta en la sección anterior, pero utilizando la etiqueta gridLayoutAnimation. Se le aplica un pequeño delay para que los items no apliquen la animación simultáneamente, sino que exista un pequeño desfase entre los elementos.

```
<?xml version="1.0" encoding="utf-8"?>
<gridLayoutAnimation xmlns:android="http://schemas.android.com/apk/res/android"
    android:animation="@anim/down_to_up"
    android:animationOrder="normal"
    android:columnDelay="15%"
    android:direction="top_to_bottom|left_to_right"
    android:rowDelay="15%"
    android:startOffset="700" />
```

Uso

Para usar este elemento en una vista, hay que usar la clase custom en el XML. Desde el activity, debemos instanciar el layout y su adapter de la forma normal de hacerlo. Además de esto, hay que incluir un método que ejecute la animación cada vez que queramos, y notifique al adapter los cambios.

```
<!-- Custom RecyclerView ... Animated -->
<com.blazebooks.model.CustomGridRecyclerView
    android:id="@+id/recyclerView_search"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_below="@+id/search_toolbar" />
```

Cuadros de Diálogo Custom y Cuadros de Diálogo Dinámicos

Para dar un aspecto más característico y aumentar las posibilidades de un cuadro de diálogo normal, hemos creado nuestros propios cuadros de diálogo. Todos ellos heredan de la clase DialogFragment de Android, y tienen utilidades diferentes.

Implementación: Mostrar los Cuadros de Diálogos Custom

El DialogFragment() de Android sobre el que hemos construido nuestros cuadros de diálogos personalizados requiere de un **FrameLayout** para poder inflar la vista. Por ello, el primer paso para mostrarlo es incluir dicho elemento.

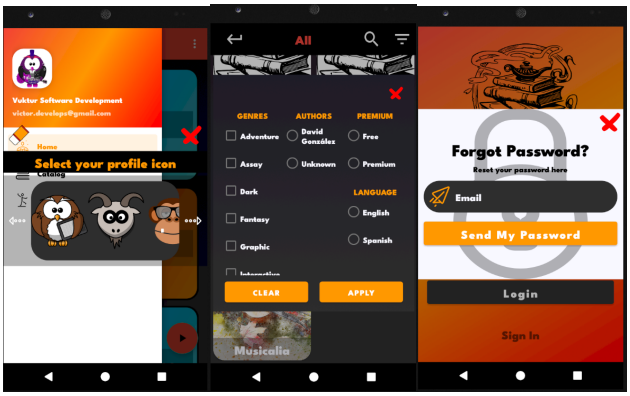
En en la etiqueta debemos poner los atributos **clickable y focusable a true**, para bloquear la interacción con el resto de la vista que no pertenezca al cuadro de diálogo. También se debe poner el atributo **visibility a gone** para que el elemento se vaya fuera de la vista hasta que se vaya a hacer uso de él. En algunos casos hemos agregado un fondo de color gris transparente, indicando que el fondo se ha bloqueado.

```
<FrameLayout
    android:id="@+id/mainActivityProfileImgFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_centerInParent="true"
    android:clickable="true"
    android:focusable="true"
    android:visibility="gone" />
```

Una vez hecho todo esto, procedemos a llamar al cuadro desde su host activity. Al accionar el evento que va a mostrar el cuadro, lo primero que hemos de hacer es **cambiar la visibilidad** del FrameLayout a visible. Realizamos una **transacción** con Support Fragment Manager que cargará el cuadro de diálogo dentro del layout. Antes de realizar el replace se configuran las **animaciones** de aparición del cuadro, de derecha a izquierda en este ejemplo.

```kotlin
fun setProfileImage(view: View) {
    mainActivityProfileImgFragment.visibility = View.VISIBLE
    supportFragmentManager.beginTransaction()
        .setCustomAnimations(R.anim.slide_from_right, R.anim.slide_to_left)
        .replace(R.id.mainActivityProfileImgFragment, ProfileImageDialog())
        .commit()
}
```

Algunos **ejemplos** de cuadros de diálogos personalizados serían los siguientes:



Implementación: Comunicación Entre Cuadros De Diálogo Custom & Su Activity Host

Haciendo uso de una **interfaz** en el cuadro de diálogo, se consigue **comunicar** con su actividad host. En la propia clase del custom dialog se declara la interfaz y una **instancia** de la misma sin inicializar. Esta se inicializará en el método **onAttach**. Además, debemos incluir los métodos de la interfaz dentro de los click listeners que queramos que realicen alguna acción dentro del host.

```kotlin
interface ForgotPasswdDialogListener {
    fun onForgotPasswdSent(dialog: ForgotPasswdDialog)
    fun onCloseForgotPasswdDialog(dialog: ForgotPasswdDialog)
}

private lateinit var listener: ForgotPasswdDialogListener

override fun onAttach(context: Context) {
    super.onAttach(context)
    try {
        listener = context as ForgotPasswdDialogListener
    } catch (e: ClassCastException) {
        throw ClassCastException("$context must implement ForgotPasswdDialogListener")
    }
}
```

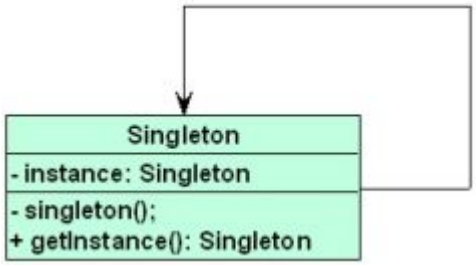
El activity host debe implementar la interfaz declarada en el custom dialog para establecer la comunicación correctamente. De lo contrario aparecerá un error en consola.

Diseño de la Aplicación: Patrones de Diseño

Al ser un proyecto con una relativa amplitud, se han utilizado diferentes patrones de diseño en la aplicación.

Singleton

Este patrón se utiliza para que solo exista una instancia de una determinada clase. Así evitamos tener más instancias de la cuenta "pululando" por la aplicación, evitamos accesos simultáneos a las bases de datos o ficheros, favorecemos la inyección de dependencias, etc.



En nuestro caso, lo hemos utilizado conjuntamente con inyección de dependencias, para así evitar instancias repetitivas e innecesarias que implican un mayor gasto de memoria. **La clase App.kt** contiene todas las instancias de bases de datos, repositorios y factories que se requieren en la aplicación, que inicializa utilizando Singleton cuando la aplicación empieza a funcionar.

Inyección de Dependencias

Patrón de diseño en el que se suministran objetos de una clase en lugar de ser la propia clase la que cree dichos objetos. Por tanto, las clases no crean los objetos que necesitan, sino que se suministran a través de otra clase contenedora que inyectará la implementación deseada. Permite delegar la responsabilidad de la creación de instancias de un componente en otro. En nuestra aplicación se utiliza conjuntamente con singleton para permitir así una mayor optimización y limpieza en el código.

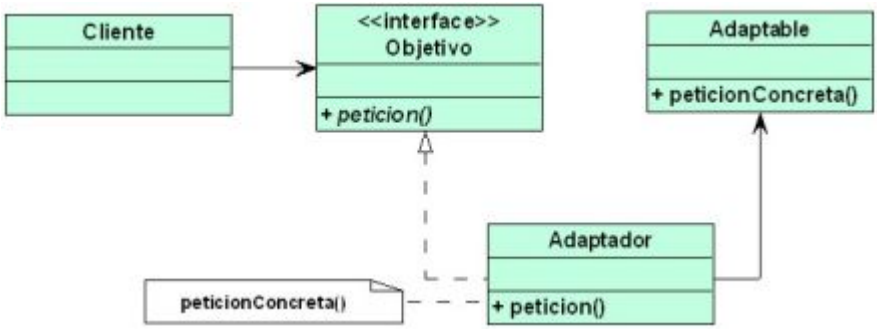
El patrón está implementado utilizando la librería Kodein. Se implementa creando una clase (App.kt) que contendrá todas las instancias que vayamos a inyectar en la app. Esta clase se debe configurar para que se lance al abrir la aplicación en el manifest.xml. Una vez hecho esto, se puede acceder a cualquier instancia creada implementando la interfaz kodeinAware en las activities y sobreescribiendo la variable kodein con la instancia que se requiera.

Es una implementación muy sencilla con la que, en apenas una línea de código, podemos tener una instancia del objeto que necesitemos sin tener que instanciarlo.

Adapter

Este patrón permite que trabajen juntas clases con interfaces incompatibles. Para ello, un objeto adaptador reenvía al otro objeto datos que recibe tras manipularlos en caso necesario. En nuestra aplicación, este

patrón se utiliza para adaptar listas de libros en vistas con las que el usuario pueda interactuar. También se utiliza para crear una vista en detalle de un único libro.



Concretamente, utilizamos dos tipos de adaptadores diferentes. El que más aparece es un adaptador normal para crear vistas partiendo de listas de objetos; la vista de búsqueda es un ejemplo de esto. El segundo es un adaptador de paginación, que nos permite utilizar pestañas; este se utiliza en la vista detalle de los libros.

Event Emitter

Este es un patrón relativamente moderno, el cual deriva del patrón observer. Escucha a un determinado evento que realiza una respuesta. Justo en ese momento emite ese evento con un valor.

En nuestro proyecto se ha utilizado para interpretar las comunicaciones con las bases de datos. Esto nos permite mostrar una u otra cosa al usuario en función de si las operaciones han sido exitosas o ha habido algún tipo de error.

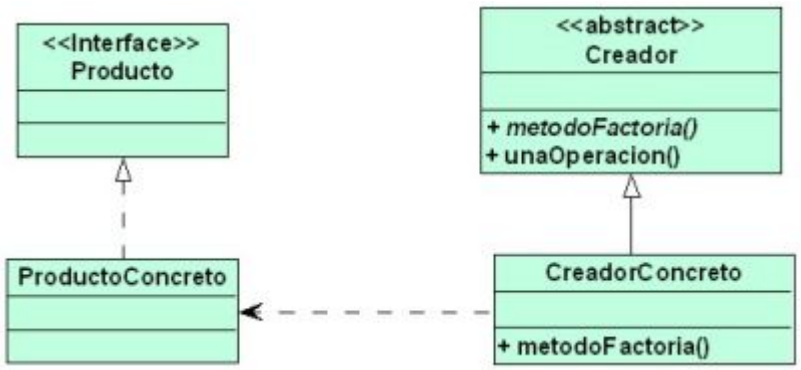
Se ha implementado en la clase FirebaseSource.kt mediante observables y completables de la librería RxJava. Lo que ocurre es que estos métodos devuelven un emitter con los datos obtenidos, y con información acerca de si se pudo realizar correctamente.

Posterior a esto, se debe implementar en el view model de las Activities. Para ello, se crea un método suspend (similar al synchronized de Java). Dentro del mismo hay que establecer el contexto en el que se va a ejecutar el método. Normalmente Dispatchers.IO.

Por último, en el host activity hay que correr las consultas mediante coroutines (similar a threads de Java). Estas corrutinas llamarán al método del viewmodel y se suscribirán a él para obtener una respuesta.

Factory

Podemos utilizar este patrón cuando definamos una clase a partir de la que se crearán objetos pero sin saber de qué tipo son, siendo otras subclases las encargadas de decirlo.



El patrón factory sirve para instanciar los viewModel en nuestra app. Para ello, se crea una clase Factory.kt que devuelva un viewModel construido. Por otro lado, en el host activity hay que implementar la interfaz KodeinAware e instanciar el objeto factory mediante ella. Luego simplemente hay que instanciar el viewModel de la clase mediante ViewModelProvider, al cual pasaremos el contexto y el objeto factory.

Todo esto tiene la finalidad de no tener instancias de los repositorios corriendo innecesariamente por la aplicación.

Comunicación FrontEnd/BackEnd

En el FrontEnd, la clase FirebaseSource.kt es la que contiene todas las consultas y conexiones con la base de datos externa (Firebase). Después, las consultas se organizan en una serie de clases XRepository.kt. Estos repositorios son incluidos en los viewModel de las activities para, finalmente, poder utilizar las consultas, normalmente empleando suspend y coroutines (similar a synchronized y thread en Java).

Un emitter permite a la aplicación saber si las consultas se han realizado exitosamente o ha ocurrido algún tipo de error. Así, podemos gestionar los fallos de conexión, consultas fallidas o cualquier tipo de error que pudiera surgir.

Modelo y base de datos

Para la creación de la base de datos hemos optado por usar Firebase. Decidimos elegir el Cloud Firestore, el cual nos permite una mayor facilidad y recursos para acceder a los datos que su otra opción, el RealTime Database. Esta opción se maneja de una manera similar a la ya aprendida en Mongo DB, basándose en colecciones donde vamos insertando un documento por cada dato.
Además del Cloud Firestore hemos utilizado otras herramientas que proporciona Firebase como pueden ser la **Autenticación** de usuarios para el manejo del login y el **Storage** para el almacenamiento de recursos. Entre estos recursos se encuentran los libros disponibles para el lector, las imágenes de portada de cada uno de ellos y la música que se usará en la aplicación.

En primer lugar, fue necesario crear un proyecto en la página de Firebase con una cuenta de google que previamente creamos y con ello Firebase proporciona una ayuda para su implementación en Android Studio. Con ello ya realizado y en el caso de nuestra app tenemos 3 apartados en la web en los cuales debemos entrar.

En el apartado **Database** creamos las colecciones que usa la app para almacenar los datos:

En el apartado **Storage** separados por carpetas insertamos los recursos a los que accederá la web usando el link que se genera.

En el apartado **Autenhication** tenemos los usuarios registrados en la app y los diferentes métodos que estos tienen para registrarse. En el caso de nuestra app usamos autenticación por correo electrónico y por cuenta de Google.

Metodología de autenticación de usuarios: Para añadir la funcionalidad del login de usuarios seguimos la página oficial de Firebase para implementar los métodos que manipulan los datos de los usuarios. Para la implementación de estos usamos el Patrón DAO, lo que simplifica el código y lo deja más entendible.

Descarga de libros

Funcionalidad de botón de descarga: Para acceder a la lectura del libro es necesario la previa descarga de este. Para ello hemos implementado la librería PWDownloader. En primera instancia usamos un método interno de android llamado DownloadManager pero por problemas con el tiempo de ejecución decidimos cambiar.
Esta librería incluye el método download al que se le pasan una URL (en nuestro caso la dirección del libro seleccionado almacenado en Firebase), la dirección de la carpeta de destino y el nombre del archivo. La carpeta de destino se crea justo antes de realizar el método download y se borra si este da error.

Carpeta de destino y archivos

La carpeta de destino se crea en Android/data/com.blazebooks/files/books y tiene el nombre del libro descargado. Dentro de ella en se encuentran 2 carpetas (Images y Style) y el libro en formato epub. Las 2 carpetas se crean al descargar y en ellas se introducen los archivos almacenados dentro del archivo epub, usando el metodo resource de la librería siegmann.epublib y separándolos por tipo de archivo. En la carpeta Image se almacenan todas las imágenes incluidas en el libro y se acceden a ellas desde el lector para poder mostrarlas en la aplicación. Por otro lado, la carpeta Style tiene dentro el archivo css que da el estilo al libro, el cual se accede y se modifica desde las opciones introducidas dentro de los ajustes del lector.

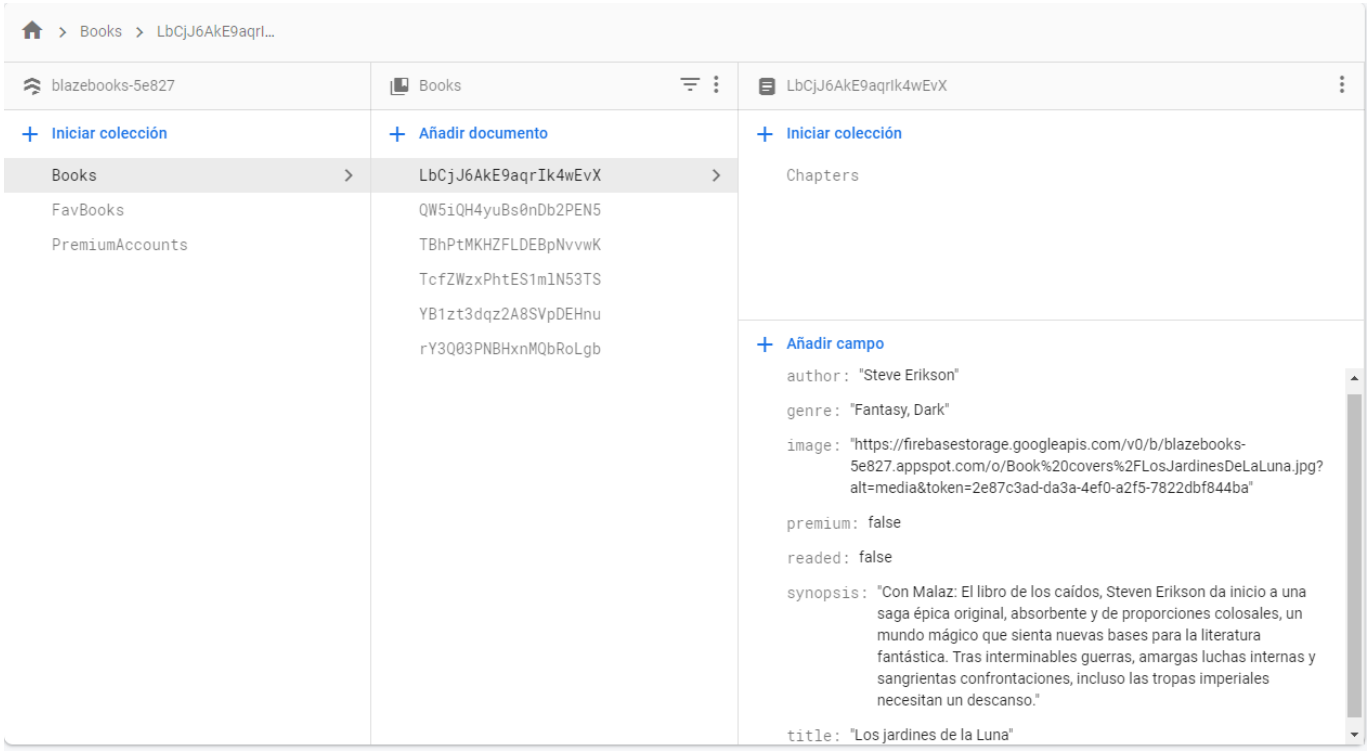
Borrado del libro

Una vez el libro ha sido descargado el botón de descarga cambia al botón de borrado. Al pulsarlo la aplicación manda un mensaje de confirmación al usuario y si se acepta se borrará la carpeta previamente creada con todos los archivos en su interior, como también se borra de la base de datos local el libro borrado.

Estructura de la base de datos

La base de datos creada en Firebase se compone de 3 colecciones, Books, Favbooks y PremiumAccounts.

Books contiene todos los datos de los libros accesibles en la aplicación desde el apartado de All. En ella se almacenan los datos principales del libro, la URL de la imagen de portada, 2 booleanos para comprobar si es un libro Premium y si el libro ha sido leído y por ultimo una subcolección con los capítulos del libro (Esta última con su número, título y comprobación de leído).


Books > LbCjJ6AkE9aqrI...
blazebooks-5e827
Iniciar colección
Books
FavBooks
PremiumAccounts
Books
Añadir documento
LbCjJ6AkE9aqrIk4wEvX
QW5iQH4yuBs0nDb2PEN5
TBhPtMKHZFLDEBpNvvwK
TcfZWzxPhtES1mlN53TS
YB1zt3dqz2A8SVpDEHnu
rY3Q03PNBHxnMQbRoLgb
LbCjJ6AkE9aqrIk4wEvX
Iniciar colección
Chapters
Añadir campo
author: "Steve Erikson"
genre: "Fantasy, Dark"
image: "https://firebasestorage.googleapis.com/v0/b/blazebooks-5e827.appspot.com/o/Book%20covers%2FLosJardinesDeLaLuna.jpg?alt=media&token=2e87c3ad-da3a-4ef0-a2f5-7822dbf844ba"
premium: false
readed: false
synopsis: "Con Malaz: El libro de los caídos, Steven Erikson da inicio a una saga épica original, absorbente y de proporciones colosales, un mundo mágico que sienta nuevas bases para la literatura fantástica. Tras interminables guerras, amargas luchas internas y sangrientas confrontaciones, incluso las tropas imperiales necesitan un descanso."
title: "Los jardines de la Luna"

Accesibilidad

En android, existe un lector que va reproduciendo todos los textos en pantalla e indicando si existen botones, menús u otros elementos en la pantalla. El problema viene cuando se encuentra con una imágen. Para solventar este problema, se ha establecido un texto de ayuda, que se mostrará en pantalla en caso de no poder cargar la imagen o en caso de tener configurado el reproductor de ayuda de android.

Usando el atributo <content-description>

Como es lógico, esto no se ha aplicado a todas las imágenes, sino que solamente aplica a imágenes relevantes en la navegación por la aplicación. Si se hubiese incluido en todas las imágenes, la velocidad de lectura de la aplicación podría verse comprometida innecesariamente, haciendo muy poco usable la aplicación.

Referencias

Bibliografía

- https://academiaandroid.com/elementos-y-diseno-de-interfaz-de-usuario-en-apps-android/ [18-04-2020]
- https://software.intel.com/es-es/android/articles/building-dynamic-ui-for-android-devices [18-04-2020]
- https://www.ecured.cu/Diseño_de_Interfaces_de_Usuario [18-04-2020]
- https://encycolorpedia.es/ff8000 [18-04-2020]
- https://es.wikipedia.org/wiki/Decorator_(patrón_de_diseño) [18-04-2020]
- https://es.wikipedia.org/wiki/Patrón_de_diseño [18-04-2020]
- https://www.deque.com/blog/android-imageviews-accessible-content-descriptions/ [04-05-2020]
- https://developer.android.com/guide/topics/text/autofill-optimize [04-05-2020]
- https://developer.android.com/guide/topics/graphics/drawable-animation [05-05-2020]
- https://www.youtube.com/watch?v=dpgUYoy-Ilk [05-05-2020]
- https://kanbantool.com/es/metodologia-kanban [08-05-2020]
- https://developer.android.com/reference/kotlin/androidx/fragment/app/FragmentManager [18-05-2020]
- https://developer.android.com/reference/kotlin/androidx/fragment/app/FragmentManager [18-05-2020]
- https://developer.android.com/training/animation [20-05-2020]
- https://medium.com/@requeridosblog/requerimientos-funcionales-y-no-funcionales-ejemplos-y-tips-aa31cb59b22a [10-08-2020]
- https://diagramasuml.com/casos-de-uso/ [10-08-2020]
- https://informaticapc.com/patrones-de-diseno/adapter.php [29-09-2020]
- https://informaticapc.com/patrones-de-diseno/singleton.php [29-09-2020]
- https://informaticapc.com/patrones-de-diseno/factory-method.php [29-09-2020]
- https://es.wikipedia.org/wiki/Inyecci%C3%B3n_de_dependencias [29-09-2020]
- https://css-tricks.com/understanding-event-emitters/ [29-09-2020]
- https://www.bbvaapimarket.com/es/mundo-api/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos/ [05-10-2020]

Tecnologías

- ★ Kotlin
- ★ Firebase & SQLite
- ★ Android Studio
- ★ Git

Librerías Implementadas

- ★ Android-SpinKit
- ★ Coil
- ★ Epublib
- ★ Google Cloud Firestorage
- ★ Airbnb Lottie Android
- ★ RxJava
- ★ Kodein-DI

Recursos

- ➢ Iconos diseñados por Freepik y Prettycons
- ➢ Fuente League Spartan
- ➢ Imágenes tomadas de Pixabay

Cursos & Formación Extra

- ● Android MVVM Architecture (Simplified Coding)
- ● Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin) (edición 2012)

Aplicaciones de Referencia

- ❖ Calisteniapp

Anexos/ Otros

¿Por qué Blazebooks?

Blazebooks es un nombre que habla por sí solo. Los libros aportan conocimiento, siendo una representación de este la llama y la iluminación. Blaze es una palabra que aúna esas dos acepciones.

El portador de luz