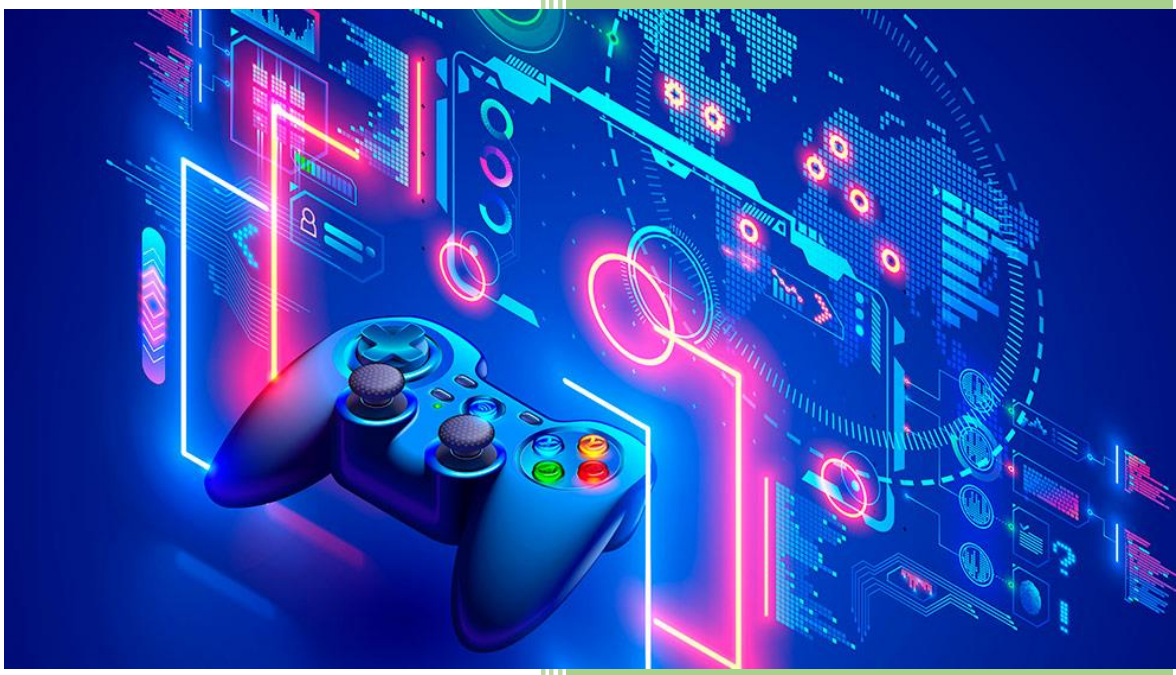
Francisco Nache

NeoGameDB



2ºDAM

IES AL-ANDALUS

2024

Índice

1. Resumen del proyecto

NeoGameDB es una aplicación de registro, gestión y expansión de una biblioteca personal de videojuegos, además, incluye una sección de noticias recientes, provenientes de los medios más afines al mundo del videojuego, y junto a esto, posee también una pestaña dónde los usuarios podrán encontrar contenido de sus obras favoritas

2. Descripción del proyecto

La aplicación está enfocada al público de los videojuegos, usada para que los usuarios que juegan a diario y, por ende, completan un catálogo considerable, puedan llevar un registro ordenado y calificado de sus obras completadas. Gracias al uso de una API, se obtienen juegos tanto actuales cómo antiguos, teniendo en cuenta la preservación de los juegos antiguos.

También se piensa en el propio proceso de juego, para lo cual, se cuenta con un apartado de Descubrimiento, en el que encontrar contenido de los juegos que se están jugando, así cómo de aquellos en los que se tiene interés, siendo también que la aplicación cuenta con un apartado de Noticias, con las últimas novedades del mundo de los videojuegos y sus adyacentes en tecnología.

3. Herramientas necesarias

Hardware:

- Ordenador de escritorio
- Smartphone

Software:

- Sistema Operativo de escritorio Windows 10 Pro
- Sistema Operativo de smartphone Android
- Android Studio
- Paquete Microsoft Office
- Herramientas de Google
- Navegador Web

Herramientas:

- Conocimiento de desarrollo obtenido durante el curso

4. Elementos relacionados con el contenido lectivo del CFGS de Desarrollo Aplicaciones Multiplataforma.

Primer curso:

- Sistemas informáticos
  - Instalación y configuración de Windows 10 Pro
  - Instalación y configuración de Softwares previamente mencionados

- Bases de datos
  - Creación de Bases de Datos SQL
  - Uso de lenguajes SQL Related

- Programación
  - Programación en Java junto a Kotlin

- Lenguaje de marcas
  - Uso y gestión de clases y archivos XML

- Entornos de desarrollo
  - Buenas prácticas de programación
  - Uso de GIT

Segundo curso:

- Acceso a datos
  - Creación de Bases de Datos SQL
  - Uso de lenguajes SQL Related

- Desarrollo de interfaces
  - Diseño y creación de interfaces en la aplicación
  - Conceptos de Usabilidad aplicados

- Programación multimedia y dispositivos móviles
  - Uso de Android Studio
  - Programación Kotlin

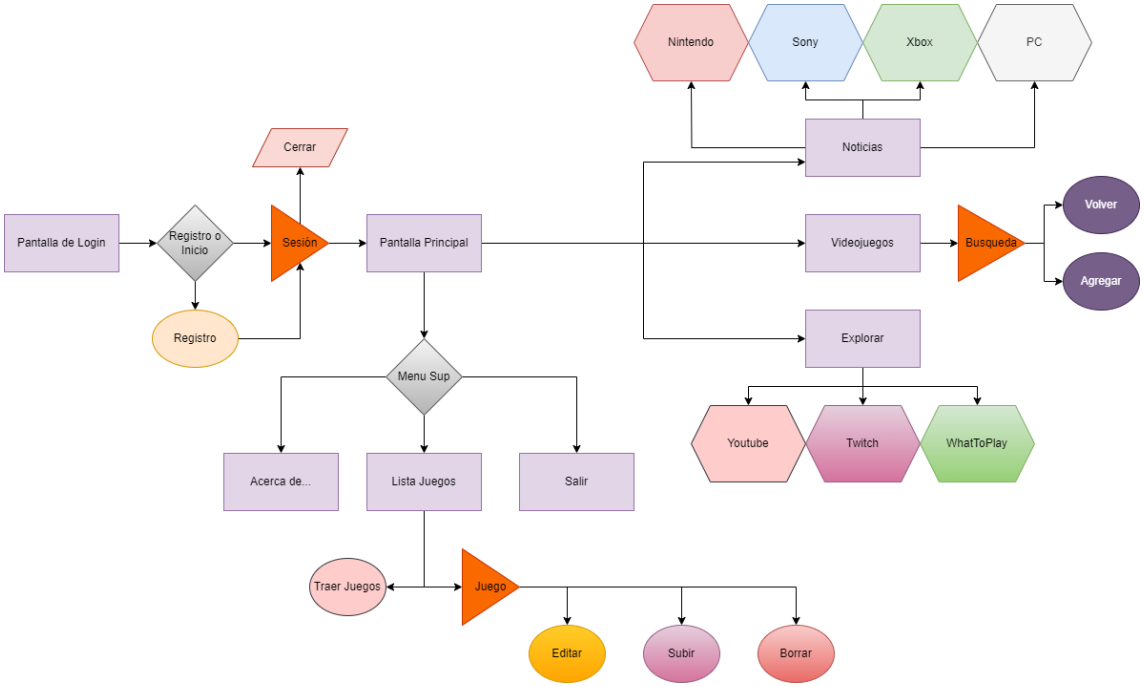
- Programación de servicios y procesos
  - Crear hilos y procesos asíncronos

5. Elementos destacables del proyecto en cuanto a investigación e innovación.

- Investigación sobre bases de datos remotas
- Investigación sobre sincronización de contenido Offline/Online

6. Diagrama de navegación

En el diagrama siguiente, apreciaremos cómo se estima el flujo de uso de la aplicación:

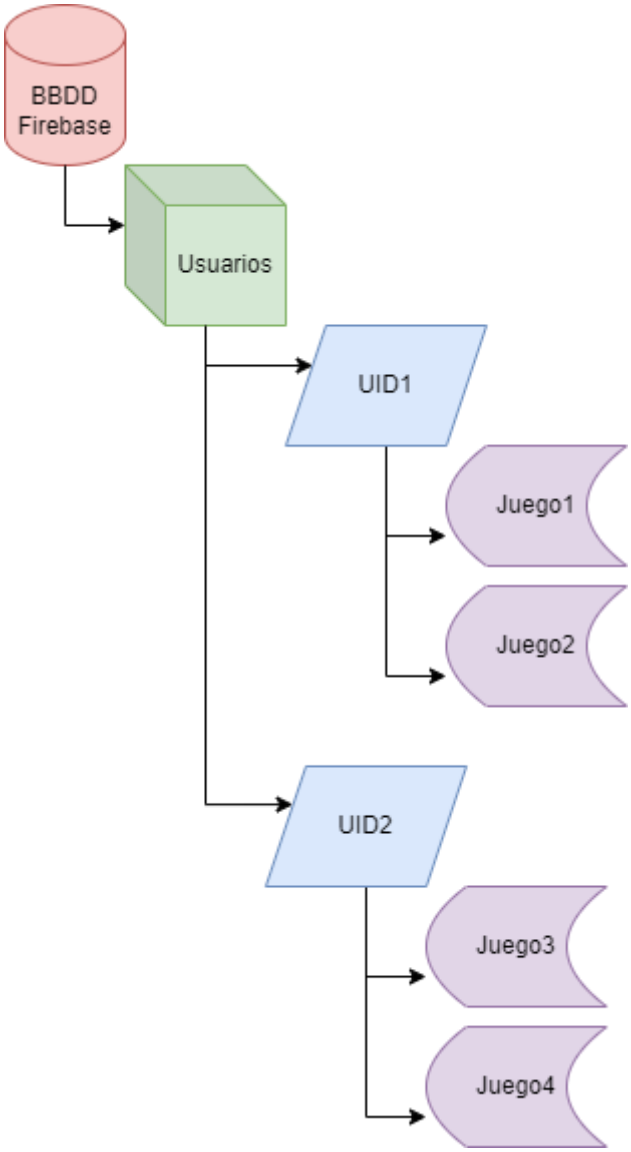


7. Base de datos

Para este proyecto, he utilizado los servicios Firebase de Google:

- Firebase Aunthentication: Será la api encargada de gestionar la seguridad del login de la aplicación, gestionando las credenciales de las cuentas y su protección.

- Firebase Realtime Database: Se usará este servicio de Base de Datos en la nube de Google para almacenar los usuarios y los juegos asociados a estos, tal y como se muestra en el siguiente diagrama.



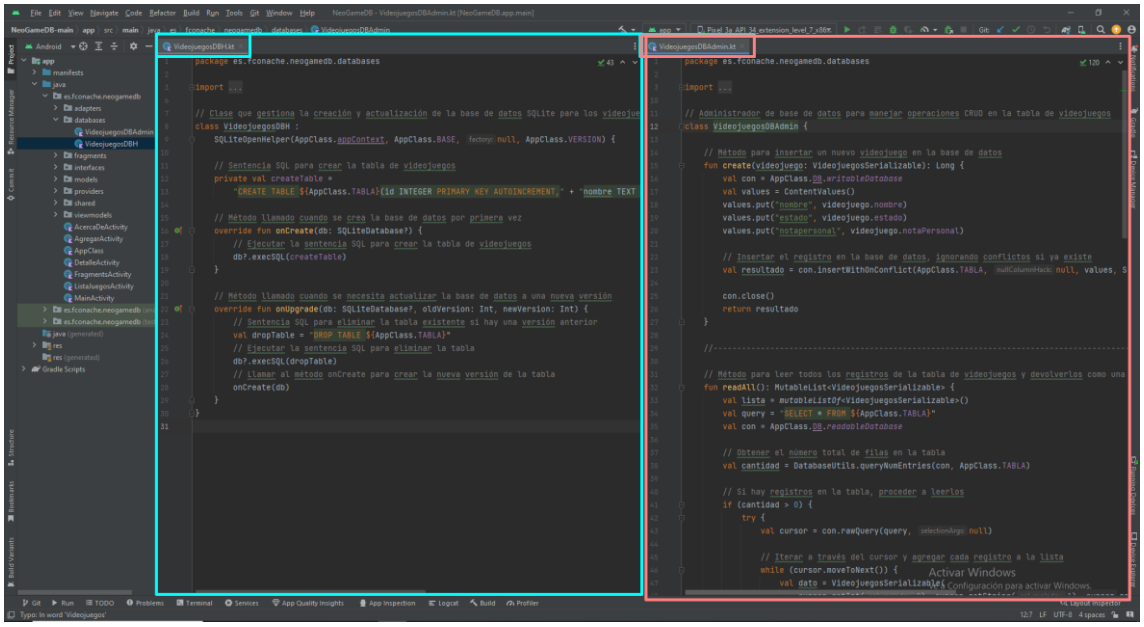
8. Desarrollo del código fuente

Todo el código fuente comentado y detallado puede encontrarse en el repositorio de GitHub del proyecto, a continuación, se explican resumidas las distintas clases usadas y separadas por apartados en la App:

8.1. Clases sobre Bases de Datos

Para este proyecto, se utilizan dos bases de datos distintas; Una SQLite local en nuestro propio dispositivo, y otra en la nube proporcionada por los servicios Google Firebase.

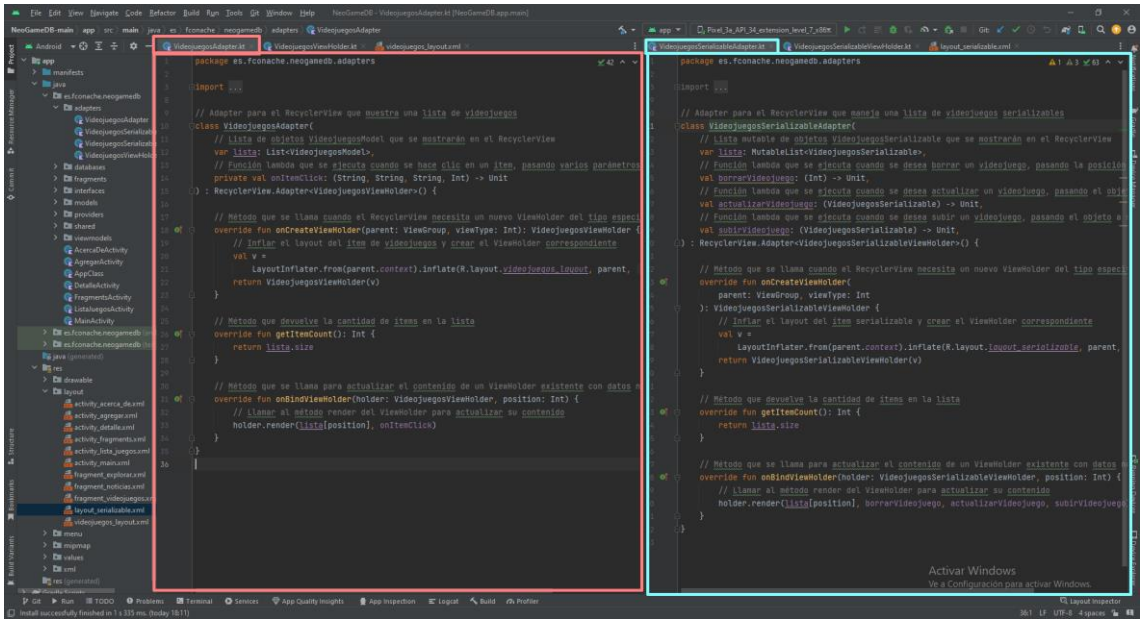
Para gestionar la base de datos local, se utilizan dos clases, VideojuegosDBAdmin, usada para la creación y actualización de la base de datos, y, VideojuegosDBH, una clase ayudante que se usa para la fácil implementación de las Querys necesarias para el uso de la BBDD.

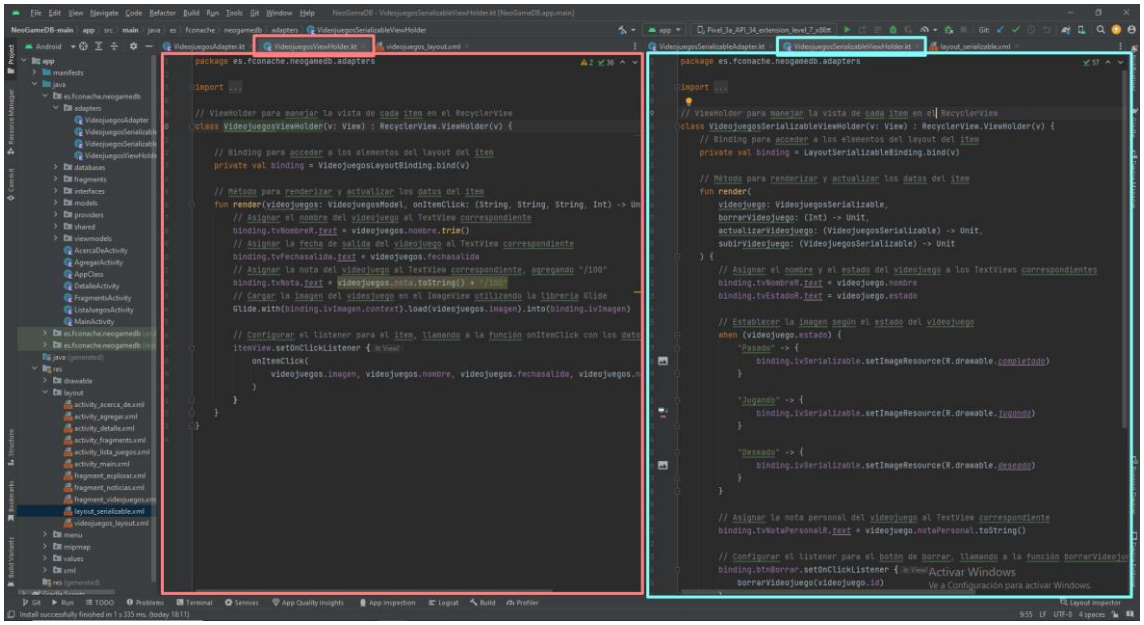


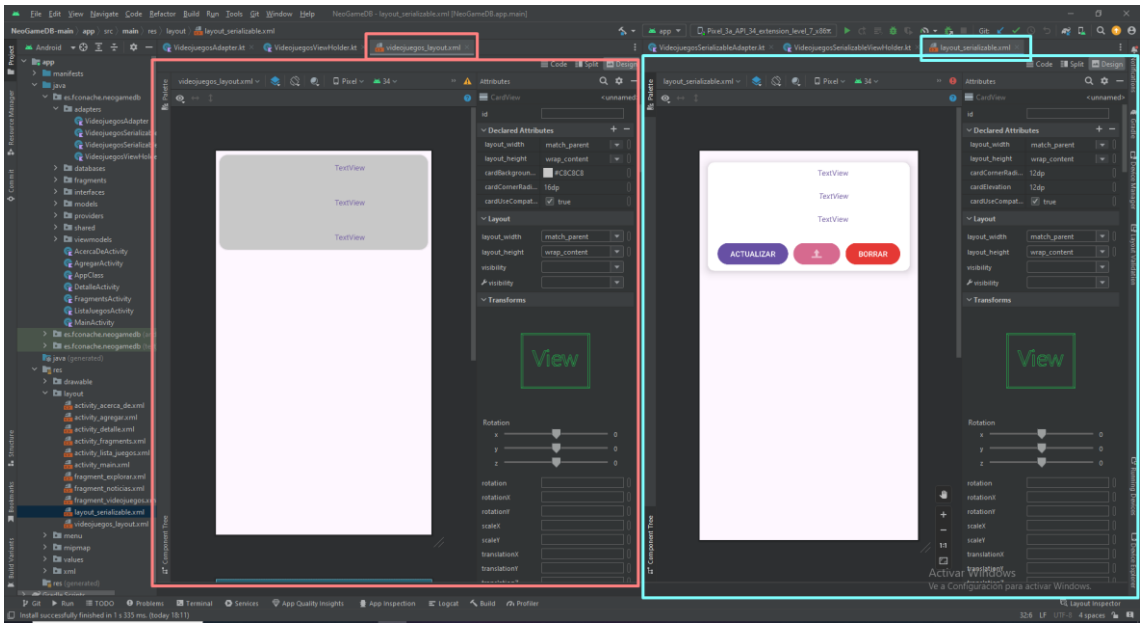
8.2. Clases Adaptador

El proyecto contempla 2 clases distintas de “videojuego”, la primera, la utilizable por la BBDD local SQLite, y la segunda, una versión serializable que pueda ser manejada en red de forma sencilla.

Cada clase de videojuego tiene de forma esperable y adyacente su propia clase ViewHolder para manejar la impresión visual de la clase junto a su layout en .xml correspondiente

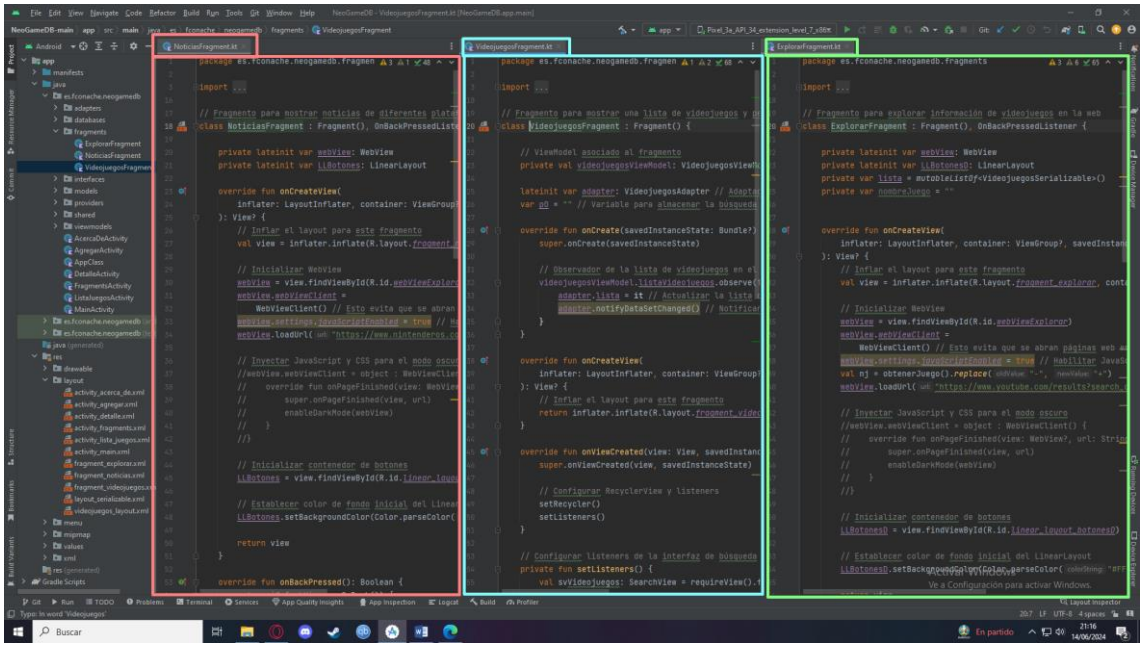






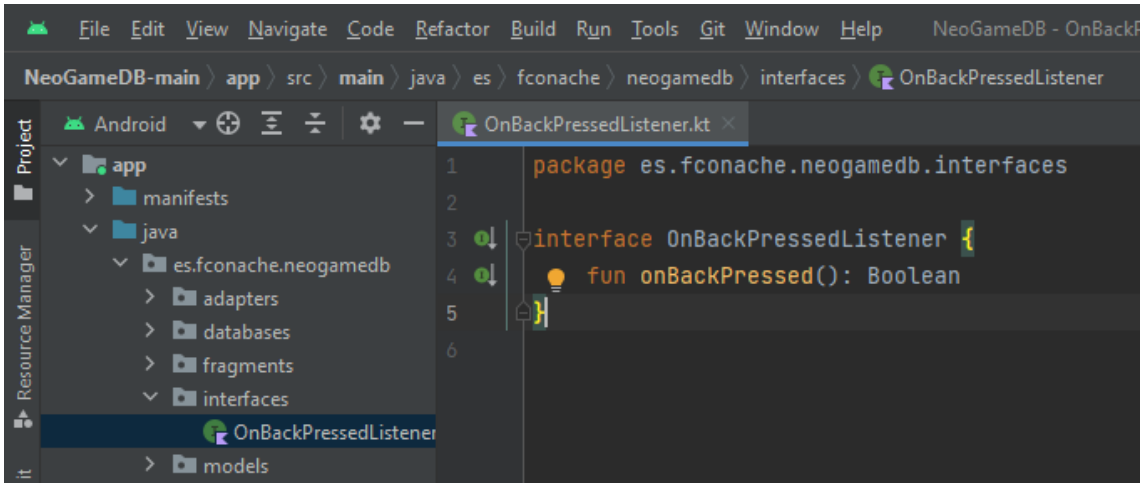
8.3. Fragments

La aplicación utiliza un contenedor de fragments principal, estimado así para un diseño más eficiente de la misma. En este contenedor, se irán alternando por voluntad del usuario los 3 Fragments utilizados en la aplicación: Noticias, Videojuegos y Explorar.



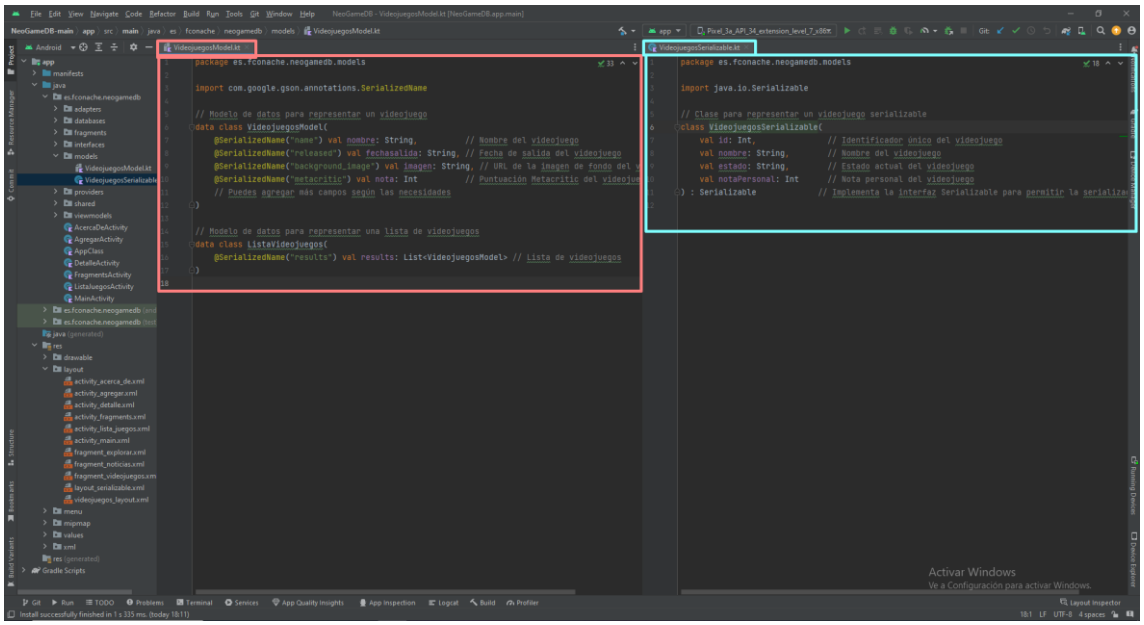
8.3.1. Interface

Un detalle de estos fragments, es que usan funcionalidades de WebView, las cuales pueden generar la necesidad instintiva de utilizar el botón de retroceso del dispositivo, y, para que ese uso sea el esperado, es necesario crear e implementar una interfaz, debido a que el uso de sobreescritura de los botones del dispositivo está obsoleto.



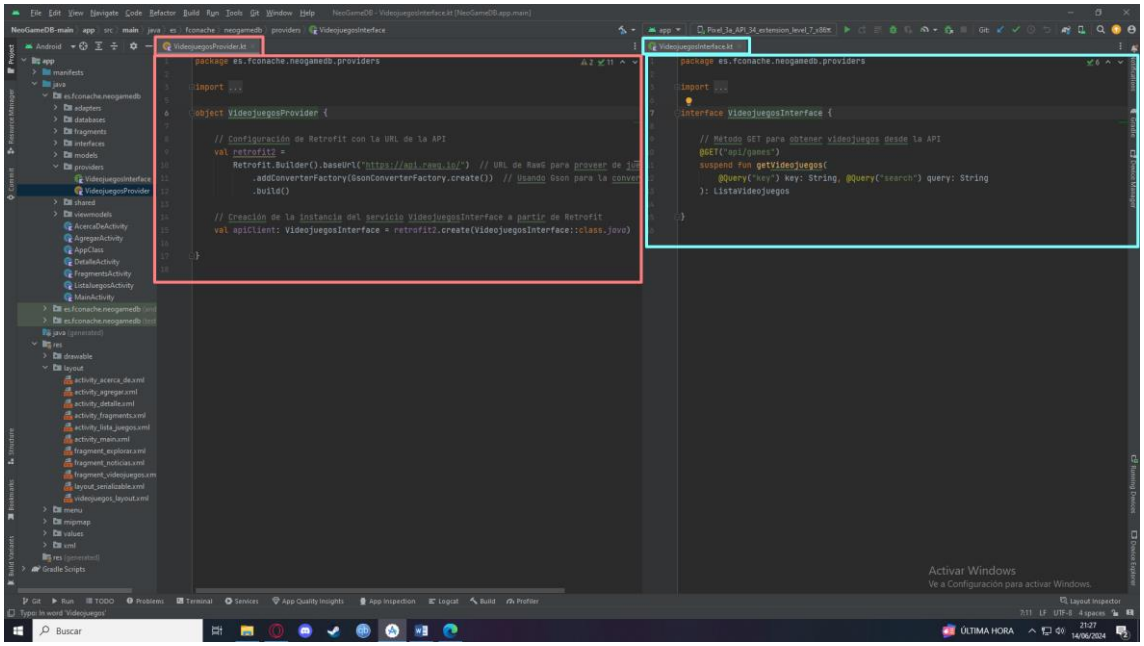
8.4. Clases Model

Como era de esperar, necesitamos clases modelo para cada tipo de videojuego, tanto SQLite como serializable, y esas clases se muestran a continuación:



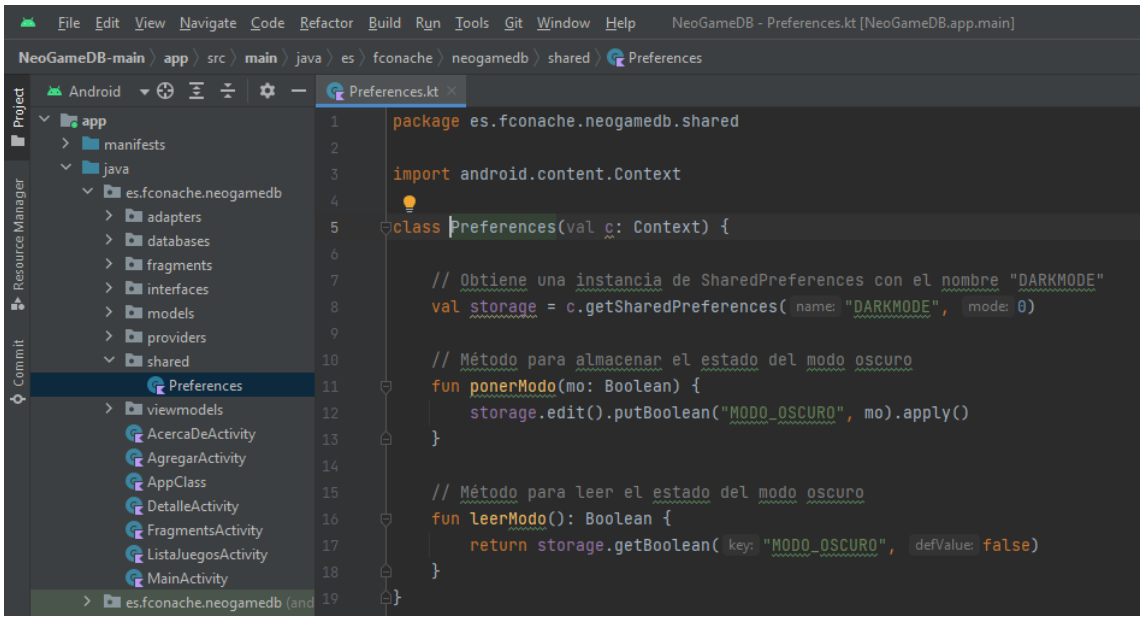
8.5. Clases Provider

Para traer a nuestra aplicación los datos obtenidos de la API de rawg.io que nos proporcionan la información sobre los videojuegos, se necesitan una clase proveedora de estos datos que se comunique con dicha Api, y una interface que la acompañe para poder ejecutar correctamente las Querys.



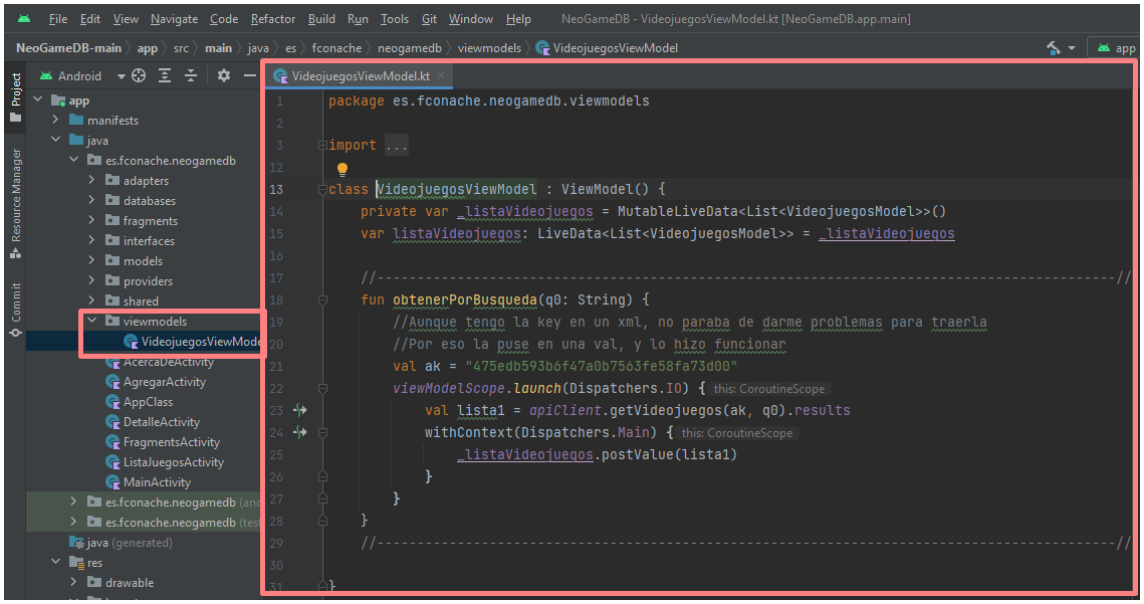
8.6. Shared Preferences

Para el proyecto, he utilizado la característica de SharedPreferences de Android que permite mantener un número ligero de datos de forma no-volátil, para mantener la decisión de si se desea usar la aplicación en Modo Oscuro o Claro.



8.7. ViewModel

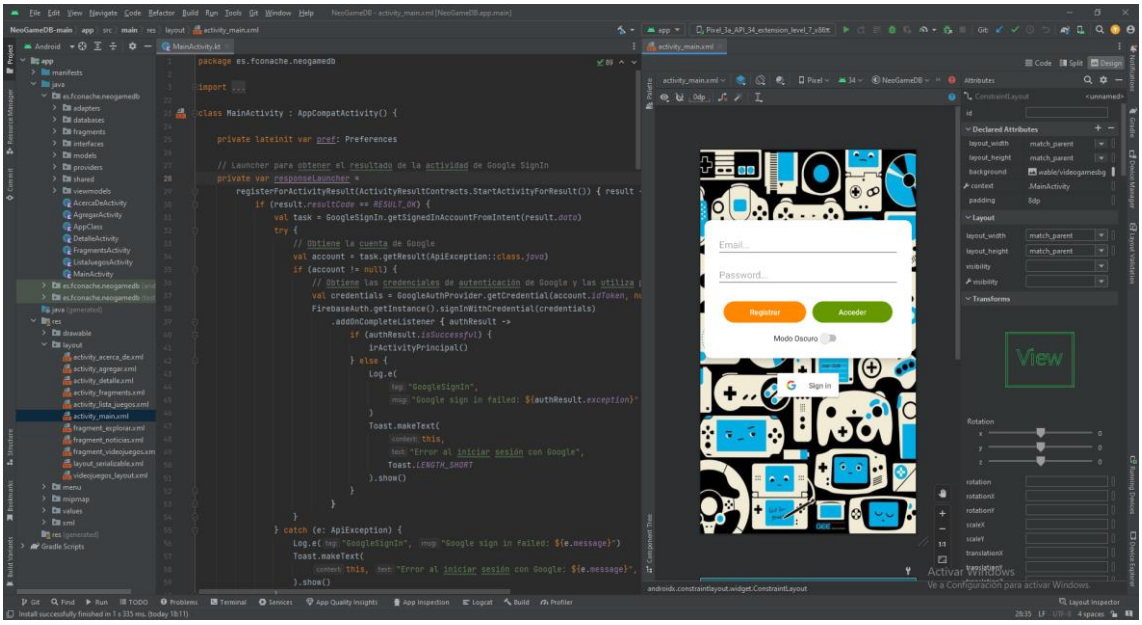
Decidí usar el modelo “Model-View-View-Model” en la aplicación para un desarrollo más dinámico, sencillo e implementable, para esto, utilizo la clase VideojuegoViewModel, que trabaja en conjunto con el Fragment de videojuegos y sus clases anidadas para hacer funcionar la sección principal.



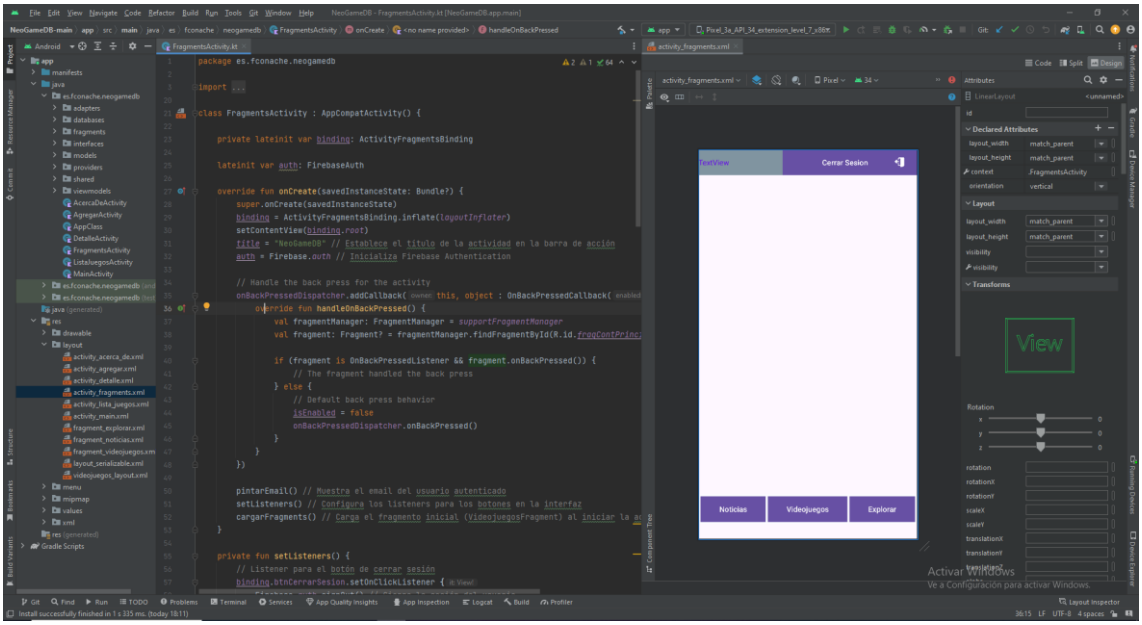
8.8. Clases Activity

Toda aplicación Android utiliza Activitys para la implementación de las funcionalidades definidas a lo largo de la misma, y aquí llegamos a las nuestras:

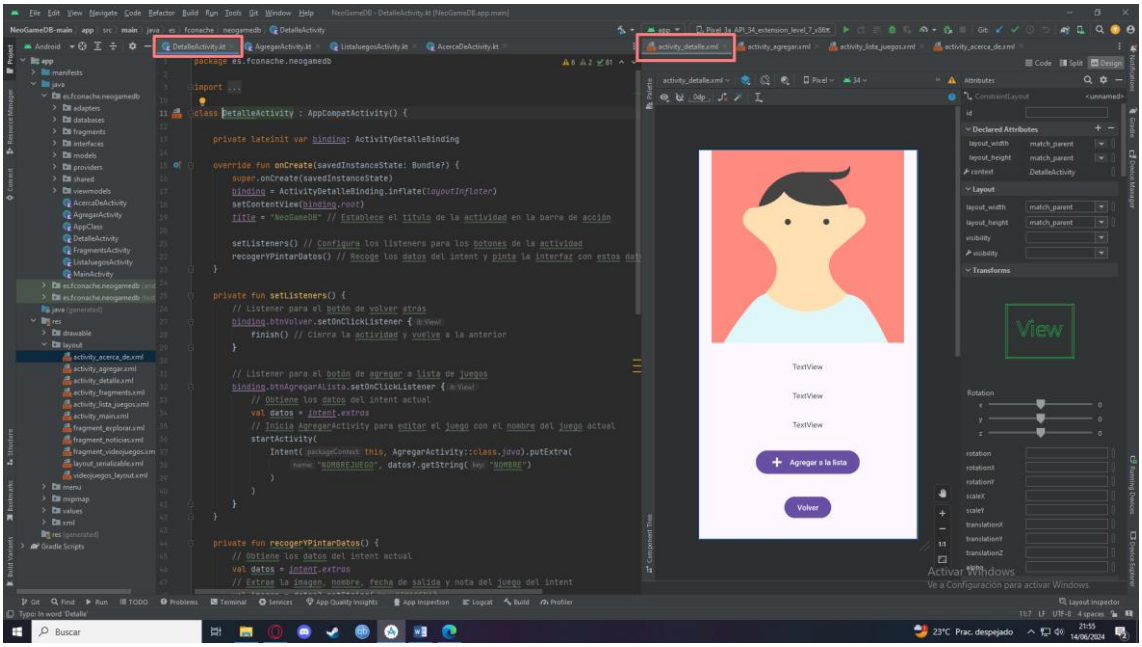
- MainActivity: La actividad principal por la que empieza la aplicación, cómo es de esperar, es la pantalla de Login usando los servicios de FireBase



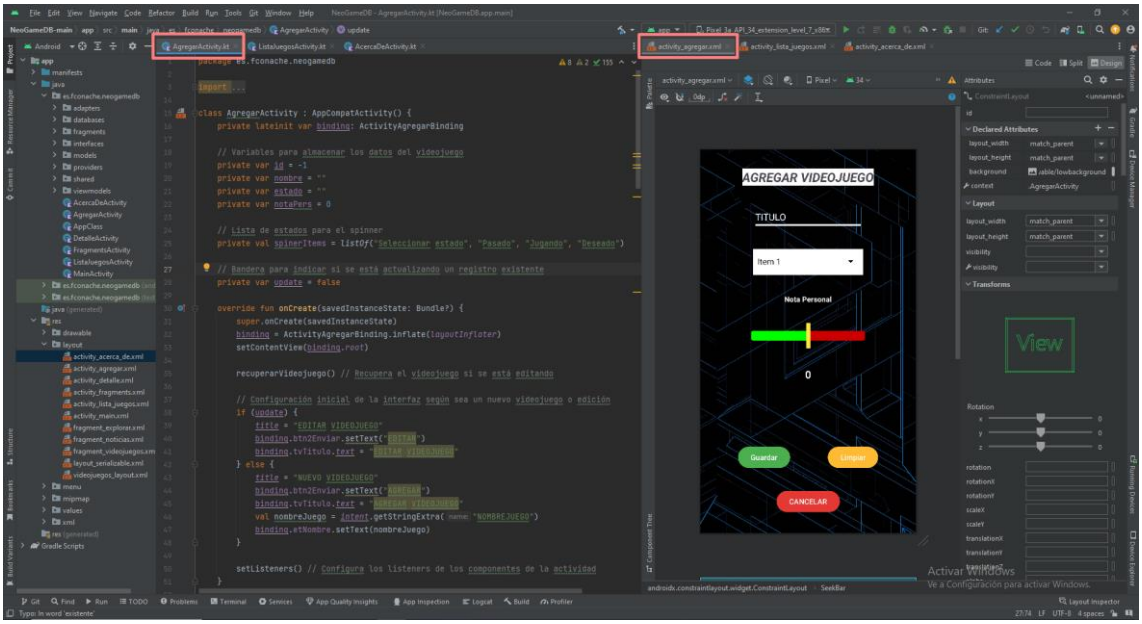
- FragmentsActivity: Es la actividad encargada de gestionar el uso de los distintos Fragments dentro de la aplicación:



- DetalleActivity: Esta es la actividad a la que se nos dirige al seleccionar un videojuego provisto tras una búsqueda en el apartado Videojuegos

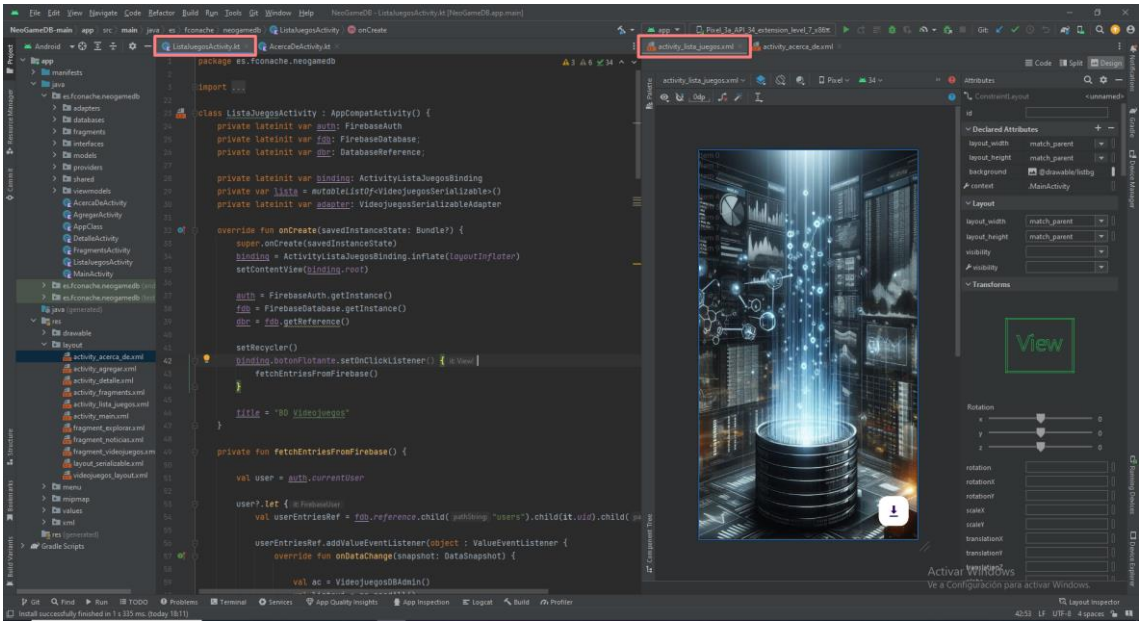


- AgregarActivity: Esta será la actividad a la que se accede tras pulsar el botón “Agregar a la lista” en la anterior. Ahora, una vez aquí, deberemos rellenar los datos para crear nuestra instancia de Videojuego:

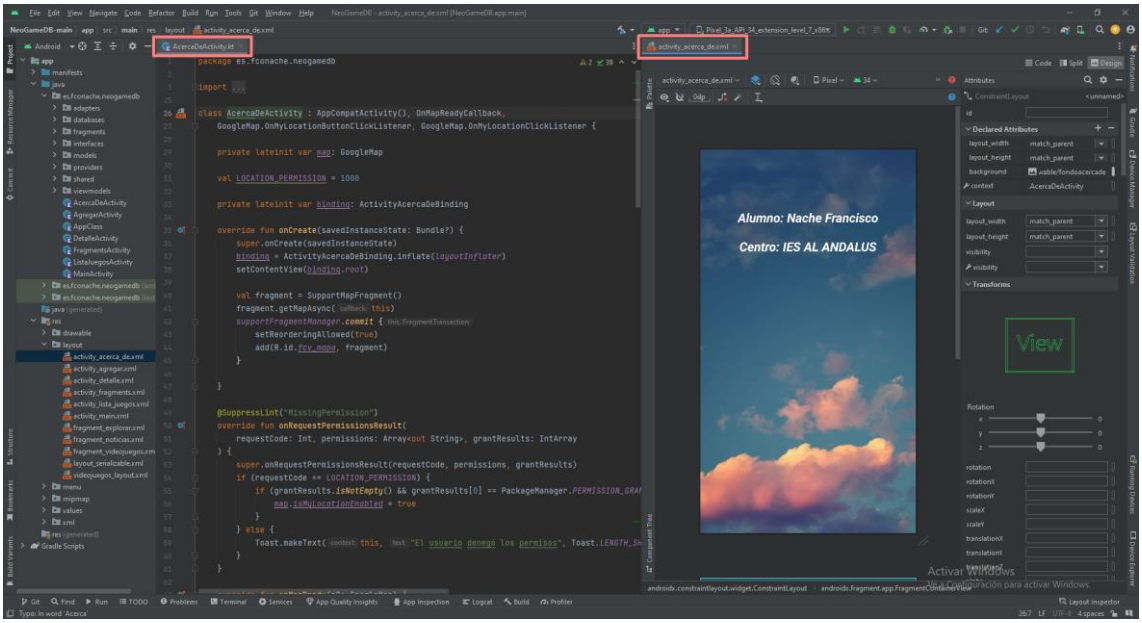


- ListaJuegosActivity: Aquí podremos ver y gestionar nuestra lista de videojuegos almacenada en la BBDD SQLite local de la aplicación en el dispositivo, será dónde editemos, subamos o borremos los juegos de nuestra lista personal, además de poder

traer de vuelta los juegos almacenados online:

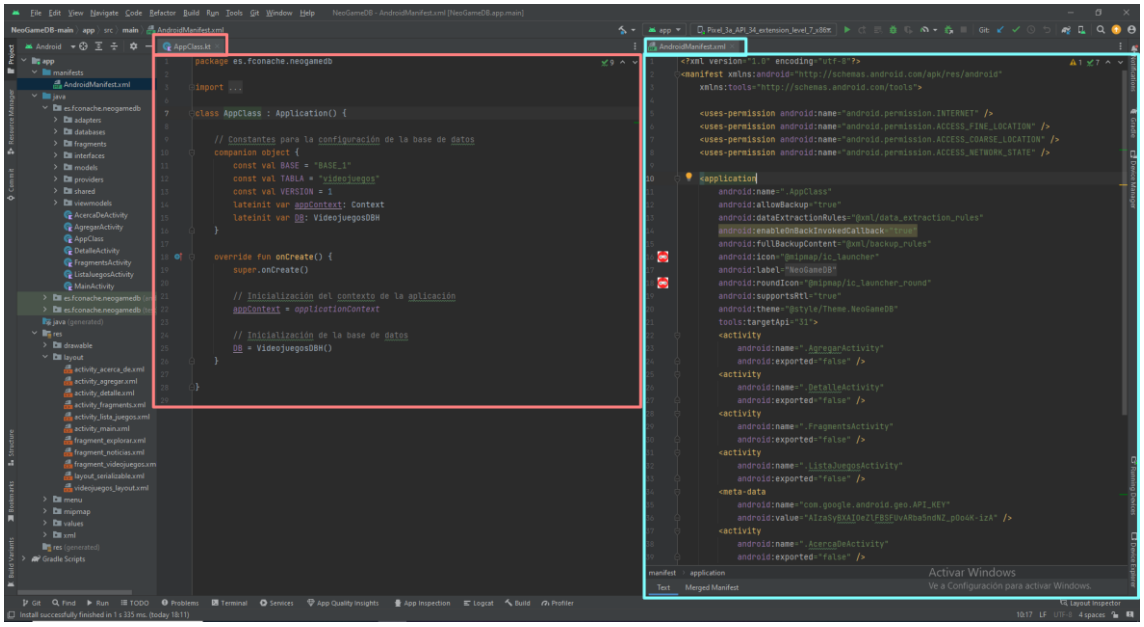


- AcercaDeActivity: Esta será la actividad de créditos, dónde figuran mi nombre y centro, para este último, se implementó un mapa de Google, sin emabrgo, la empresa limitó esta característica a usuarios de pago y actualmente no funciona correctamente, se deja como ejemplo de implementación de permisos de mapa:



8.8.1. Manifest y AppClass

Para finalizar con el apartado de código, mostraré el Manifest de la aplicación, dónde se configuran los permisos y servicios de la misma, y la clase AppClass, que se establece como Clase de ejecución, previa incluso a AppMain, para poder implementar e inicializar correctamente la BBDD local:



9. Errores y Soluciones

En este apartado, voy a enumerar los principales errores encontrados a lo largo del desarrollo y sus soluciones:

ERROR: La aplicación no se abre en mi dispositivo físico.

SOLUCIÓN: Se establece en el Manifest y en Gradle la versión y paquetes de Android necesarios y actuales.

ERROR: No es posible logearse ni registrarse.

SOLUCIÓN: Actualizar las credenciales y fechas límite de los servicios de Firebase.

ERROR: El botón “retroceder” no retrocede dentro de los WebViews.

SOLUCIÓN: Implementar la interface OnBackPressedListener.

ERROR: La base de datos almacena los juegos arbitrariamente.

SOLUCIÓN: Implementar una variable User ID para ordenar los juegos por usuarios.

ERROR: El botón “traer” devuelve los juegos cómo uno solo, en lugar de traerlos uno por uno.

SOLUCIÓN: Reescribir el método fetch para que diferencie y separe correctamente los juegos.

ERROR: Las distintas pestañas de la sección “Explorar” devuelven URLs fallidas.

SOLUCIÓN: Programar un formateo de cadenas específico para cada URL.

ERROR: La sección “Acerca De…” cierra la aplicación al intentar acceder a ella.

SOLUCIÓN: Eliminar los métodos corruptos del mapa de Google.

ERROR: El modo oscuro no permite ver el texto de los anidados en el RecyclerView de Videojuegos.

SOLUCIÓN: Formatear el color del texto en las distintas CardView de la app.

10. Conclusión

El proyecto ha resultado moderadamente ambicioso y aún hay ideas planteadas que no se han podido implementar por una razón u otra, y siendo esto así, se aplica de forma concisa todo lo aprendido a lo largo del curso, demostrando su utilidad real y la capacidad tanto de aprendizaje del alumnado, cómo de explicación del profesorado.