# Prueba técnica

Desarrollador de aplicaciones móviles

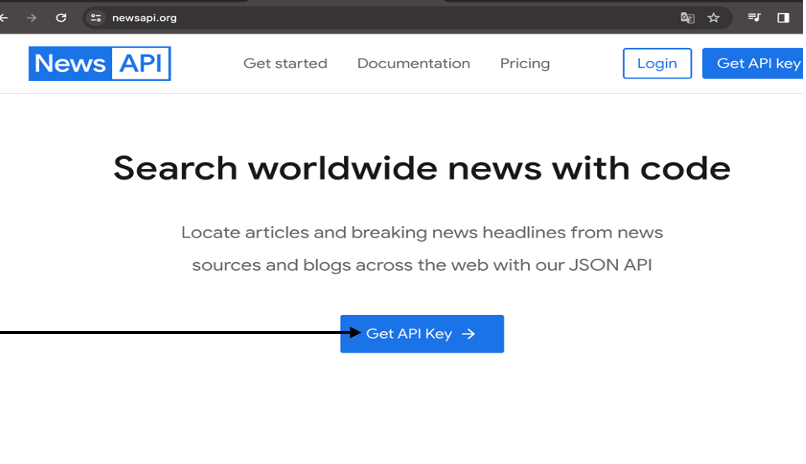
RECT NATIVE TRAINEE

SPS

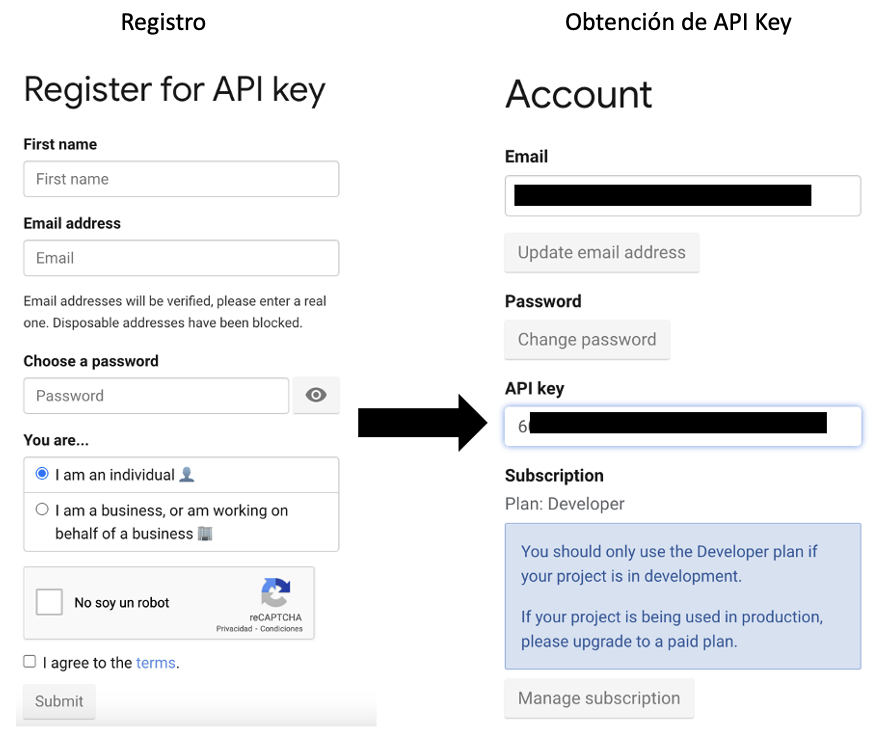
### TECH IS NOW

## **FECHA: 02/02/2024**

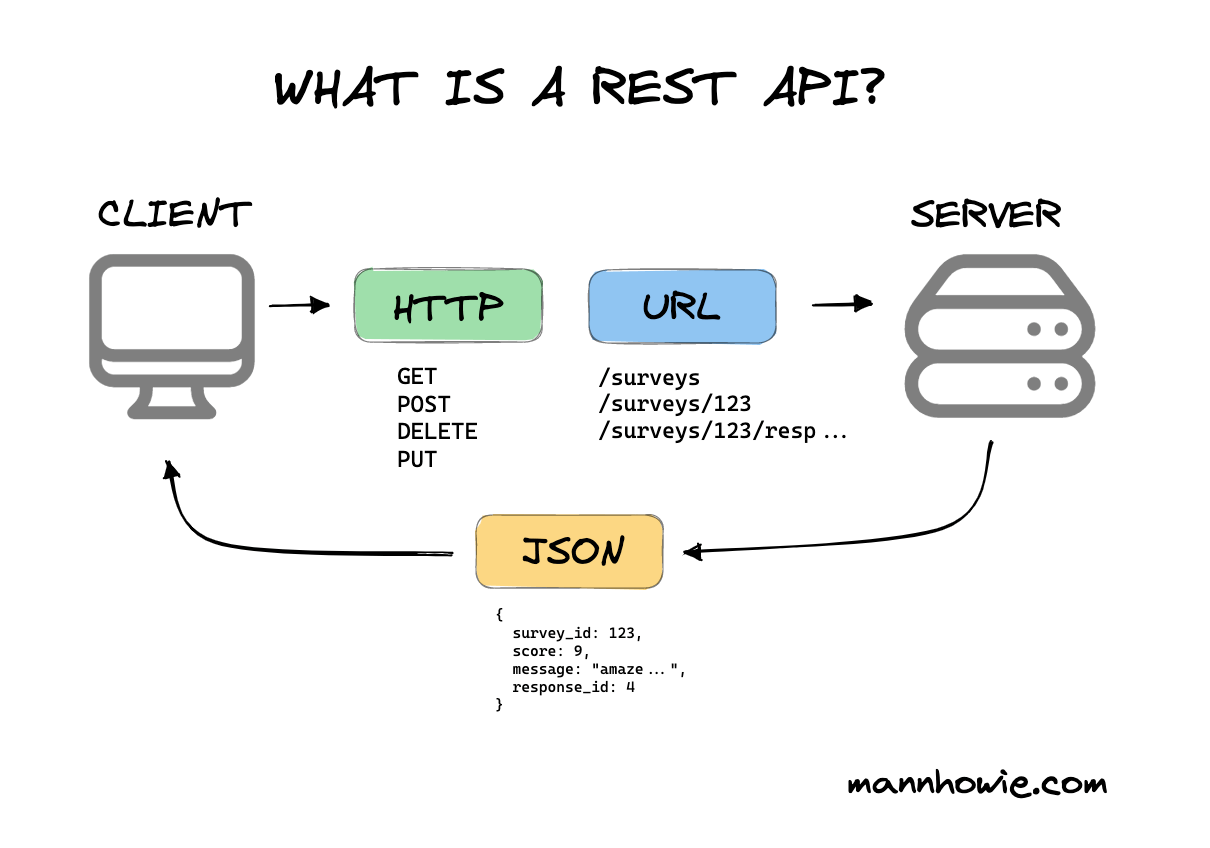
El primer paso es buscar (en su buscador de preferencia) la página newsapi.org, entrando al primer link, directamente debe dar clic izquierdo en “Get API Key”.



Lo llevará a un cuestionario donde debe registrarse, después de eso estará dado de alta en la página y obtendrá la “API Key” la cuál debe guardar ya que la va a utilizar más adelante.



Funcionamiento de una API



A continuación se usará una librería que se llama Retrofit , que va a permitir abstraer de tanto código y tanta complejidad para obtener la información de la que se requiere consultar a la API, normalmente esta información viene en un formato JSON y esto va a ser que sea tanto más fácil como sencillo poder consultar APIs y entender cómo parsear información. Parsear quiere decir que cuando se trae información desde la web como en esta API a algún dispositivo Android, se necesita de alguna forma desglosar esta información para poder ser interpretada en la aplicación.

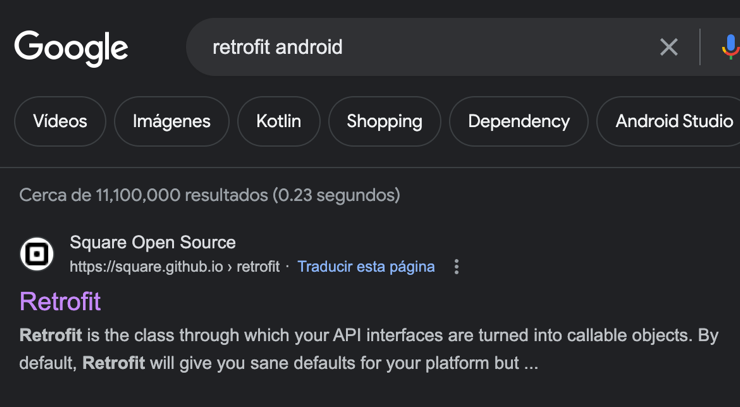
Con Retrofit y con un JSON Converter Factory se hace mucho más simple la conversión de JSON para poder trabajar en la App.

**Retrofit**

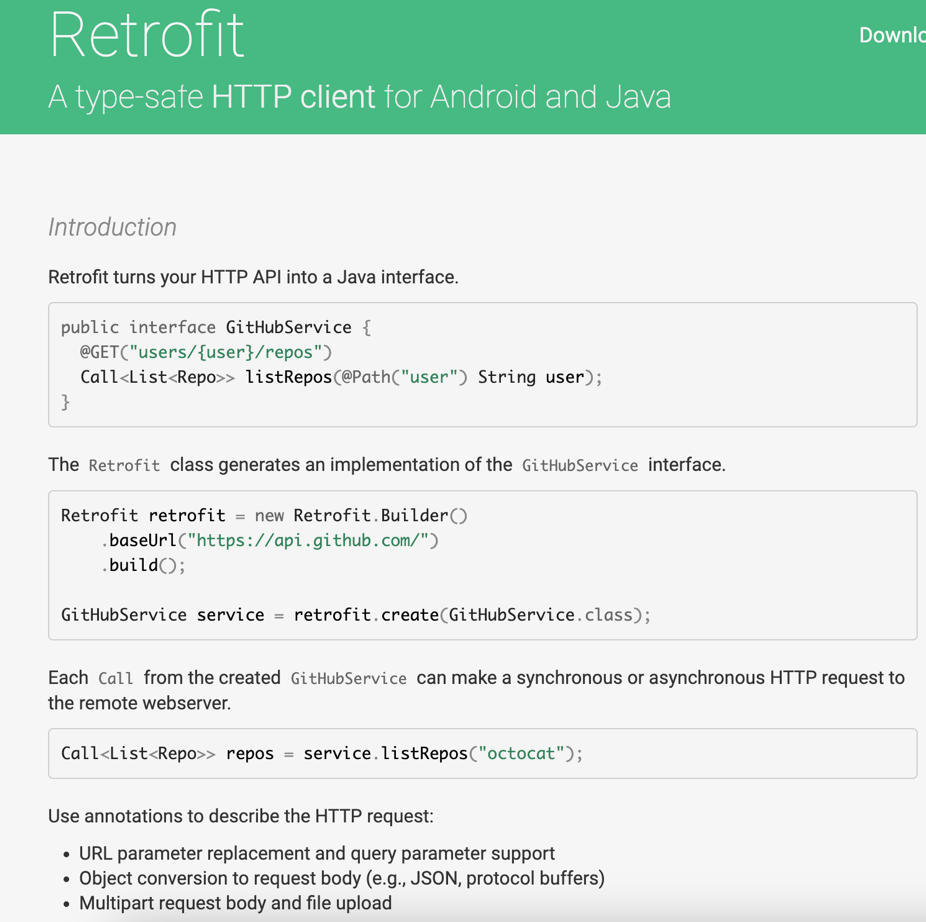
¿Cómo traer esa información del servidor a nuestra aplicación Android?

Para hacerlo se tiene que usar una librería que se llama retro fit, esa librería es para Android y sirve para poder hacer esa petición al servidor, obtener la información que viene en formato json, traerlo a la aplicación y mostrarle al usuario esa información.

Primero se debe ingresar en el buscador Retrofit android y dar clic derecho en el primer enlace.



donde desglosará lo siguiente:



Lo primero que se debe tener en cuenta viendo el ejemplo es lo siguiente:

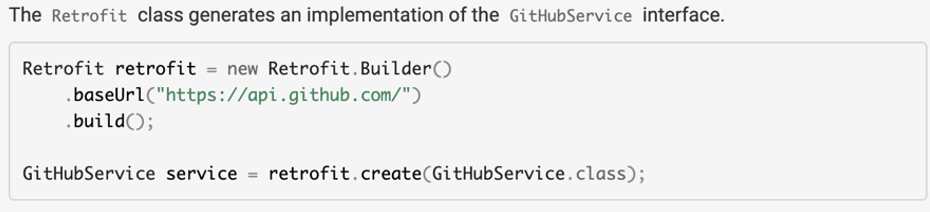


Se tiene que crear un servicio , el cual va a contener un método que va hacer la llamada al servidor por el desarrollador, para ese método se tiene que declarar primero un anotador , el anotador esta conformado del método HTTP en este caso , la dirección donde se va a consultar esa información al servidor  y luego se hace una llamada al servidor con este método: , el método tiene un call que es como una promesa que va a devolver en algún momento una lista de repositorios en este caso el ejemplo muestra que va a ir a GitHub, va a buscar un usuario y va a traer los repositorios de ese usuario.

Dentro de este método  se puede definir un @Path o un @Query, lo que uno quiera o requiera agregarle a la Url base y lo que se le está agregando es un usuario .

Entonces se esta llendo al URL  a la parte de usuarios y dentro de donde estas llaves  se esta mandando este  y va a buscar todos los repositorios  de ese usuario.

Luego se tiene que crear una instancia de Retrofit



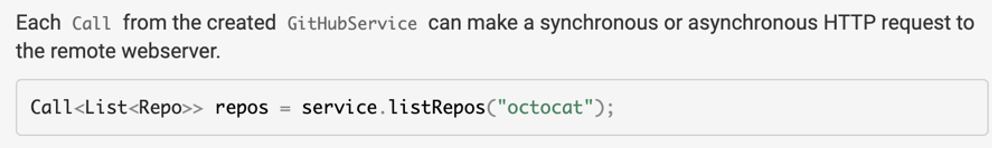
Instancia: como muestra el ejemplo; se le incorporar una baseUrl  que es donde se haran las consultas del @Get, se crea el servicio con Retrofit:



Con la instancia  y se crea ese servicio 

Que es lo que se vio con anterioridad en la interface

Y por ultimo se llama al servicio y se trae esa lista de repositorios



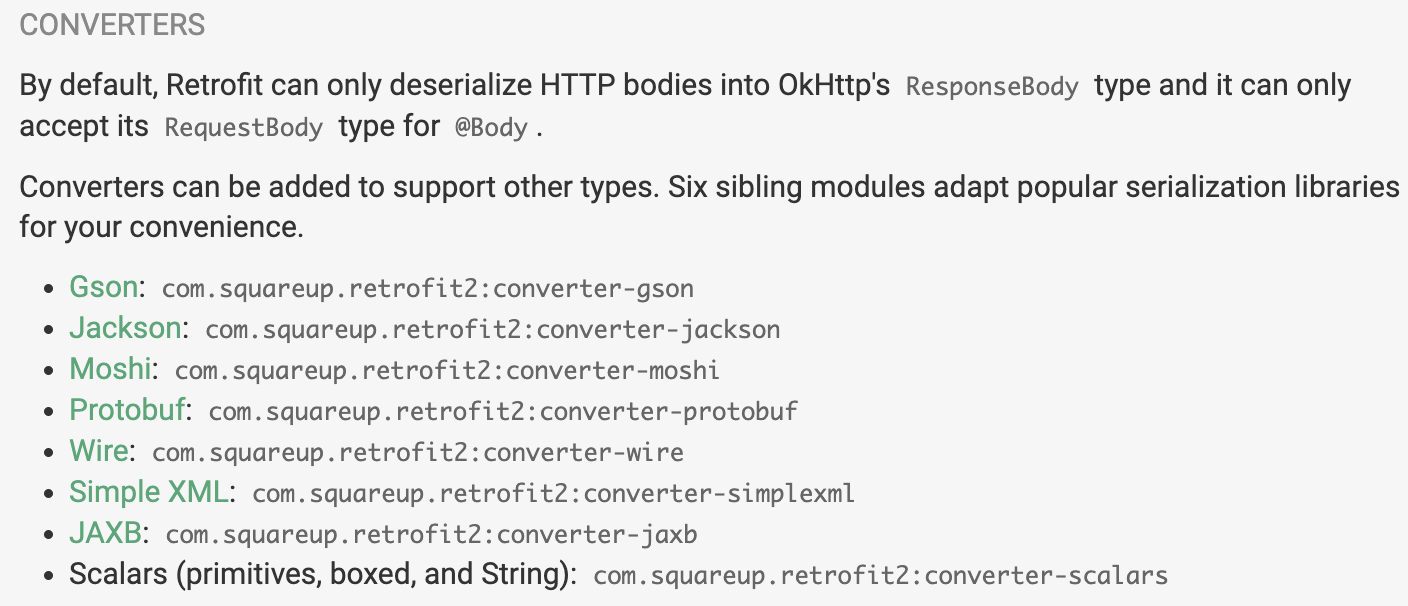
Del usuario con el nombre 

Viendo lo anterior lo que se hará para la aplicación de noticias se hará bastante similar pero un poco más “simple” porque este ejemplo está en Java y la App con la que se trabajará en Kotlin, se va a tener una Interface, se va a tener un método GET el cual va a consultar las noticias, se tendrá un método que va a ir a buscar las noticias y se le pasará un API Key aquí dentro:



Recordando, nosotros necesitamos conversores ya que cuando se esta trayendo información del servidor, se necesita deserializar esa información, la deserialización es el proceso inverso, tomando datos estructurados de algún formato y reconstruyéndolos en un objeto, esto porque esa información normalmente esta en formato JSON y se necesita transformar esa información a un modelo o un objeto en la app para que se le pueda mostrar esa información al usuario.

Bajando en la misma pagina de Retrofit la documentación muestra distintos conversores de los cuales se pueden usar los cuales son:



Se estará utlizando JSON que es una librería hecha por Google la cual se utiliza bastante.

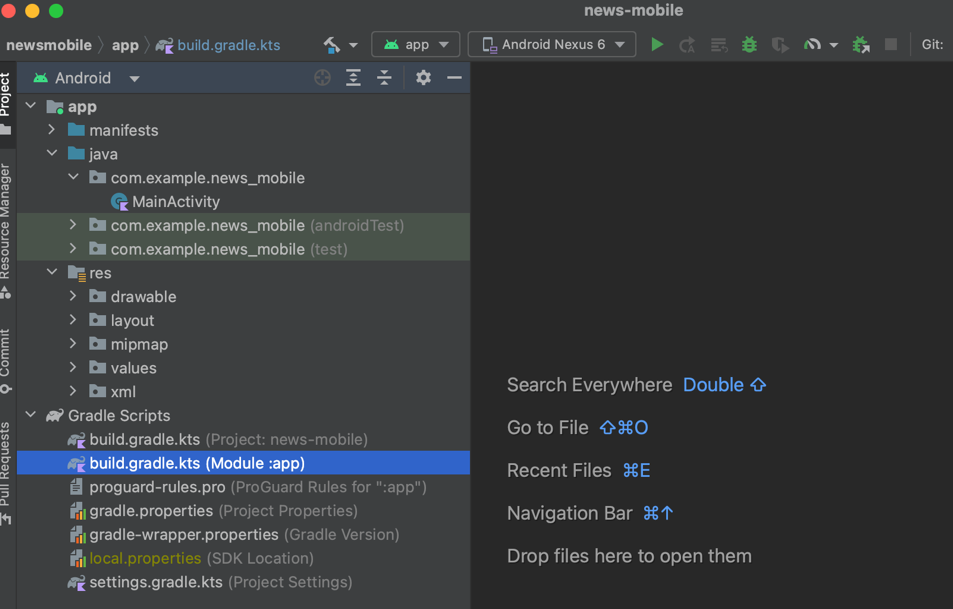
Si se le da clic en Gson los llevará a GitHub donde muestra la forma de implementar el conversor directamente.



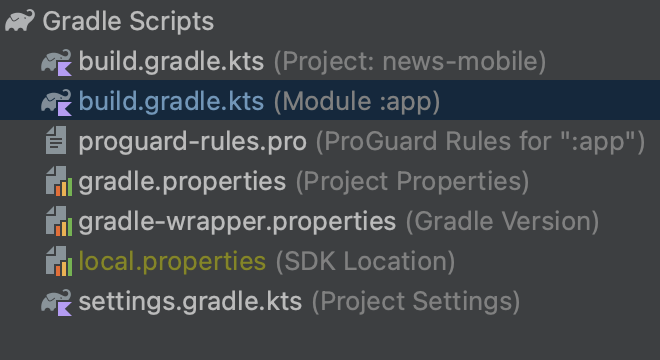
Ademas como para este proyecto se utilizara con Retrofit, también se debe agregar esta línea de código para implementarlo, se obtuvo donde con anterioridad se mostraron los distintos conversores.



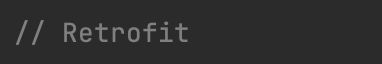
Sin embargo, antes de implementar el conversor y la librería de GSON se debe implementar Retrofit, viendo el proyecto en Android Studio en la sección de Gradle Scripts.



En el apartado del Modulo del build.gradle.



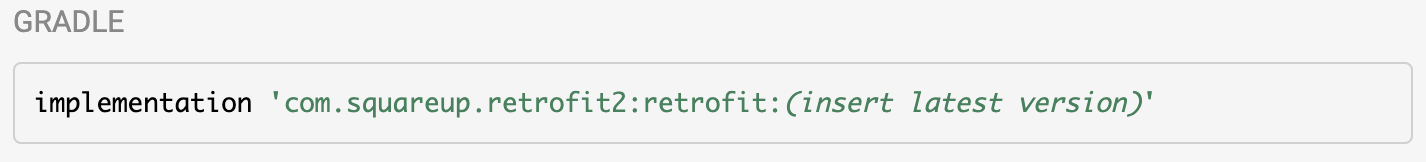
En las dependencias se comenta que esas lineas tendran lo que se va a implementar de Retrofit.

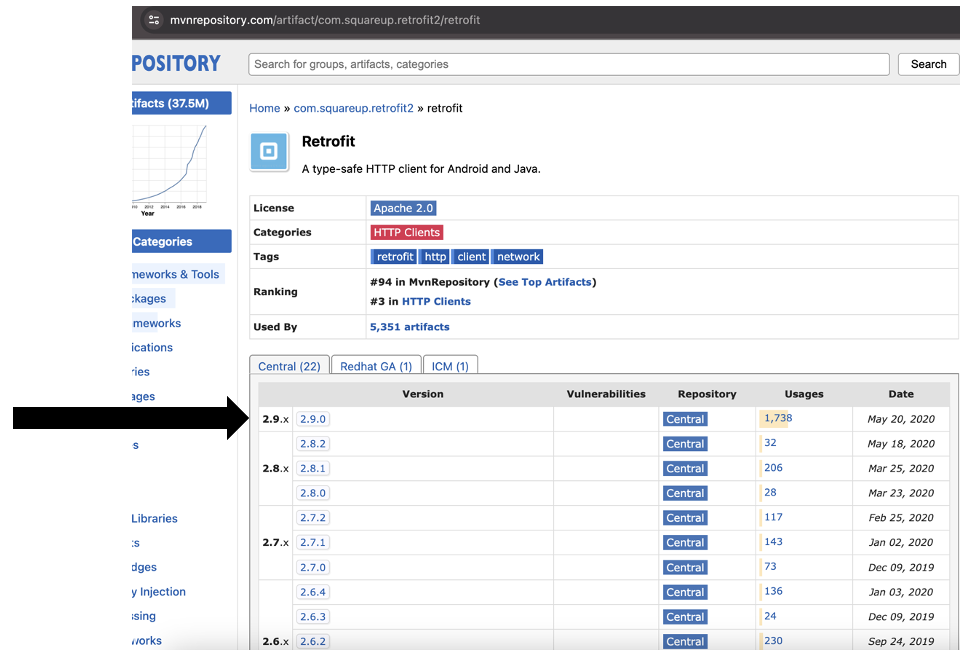


Y se pone la línea de código que nos muestra Retrofit en la parte de GRADLE:

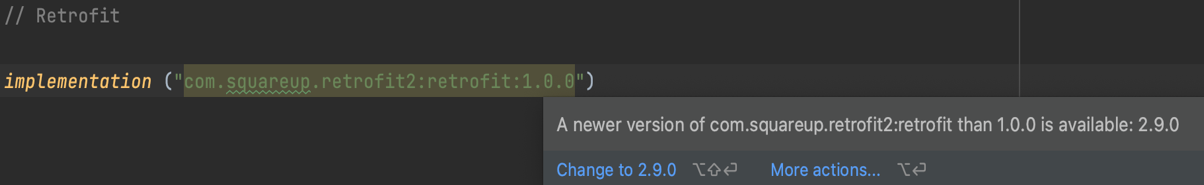


Lo cual nos dice “insert lates version” inserte la útima versión, no está dando la versión desde la página, por lo que tenemos dos formas de encontrar la versión:

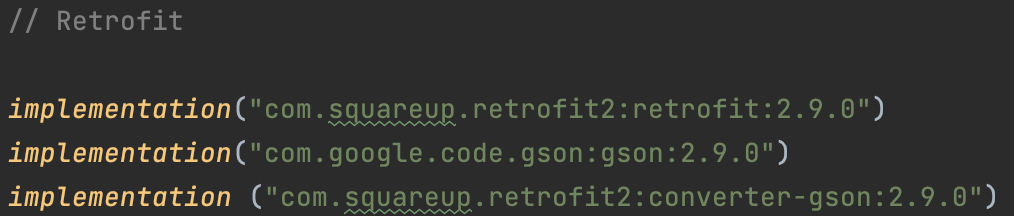
1. Se puede encontrar copiando la primer parte de la librería  como esta librería es de MAVEN, en el buscador de su preferencia puede pegar esa parte de la librería y en el primer link nos mostrará todas las versiones de Retrofit observando que la última versión es la 2.9.0.



1. Copiando y pegando esa línea de código en Android Studio el cual se subraya en amarillo, al colocar el puntero del mouse en la línea de código mostrará la última versión y al dar clic en “Change to 2.9.0 lo cambiará en automático.



Una vez implementado Retrofit en Android Studio se procede a implementar el conversor y la librería de GSON que se obtuvo el primero con anterioridad en GitHub y el segundo en la misma página de Retrofit obteniendo este resultado en las dependencias:

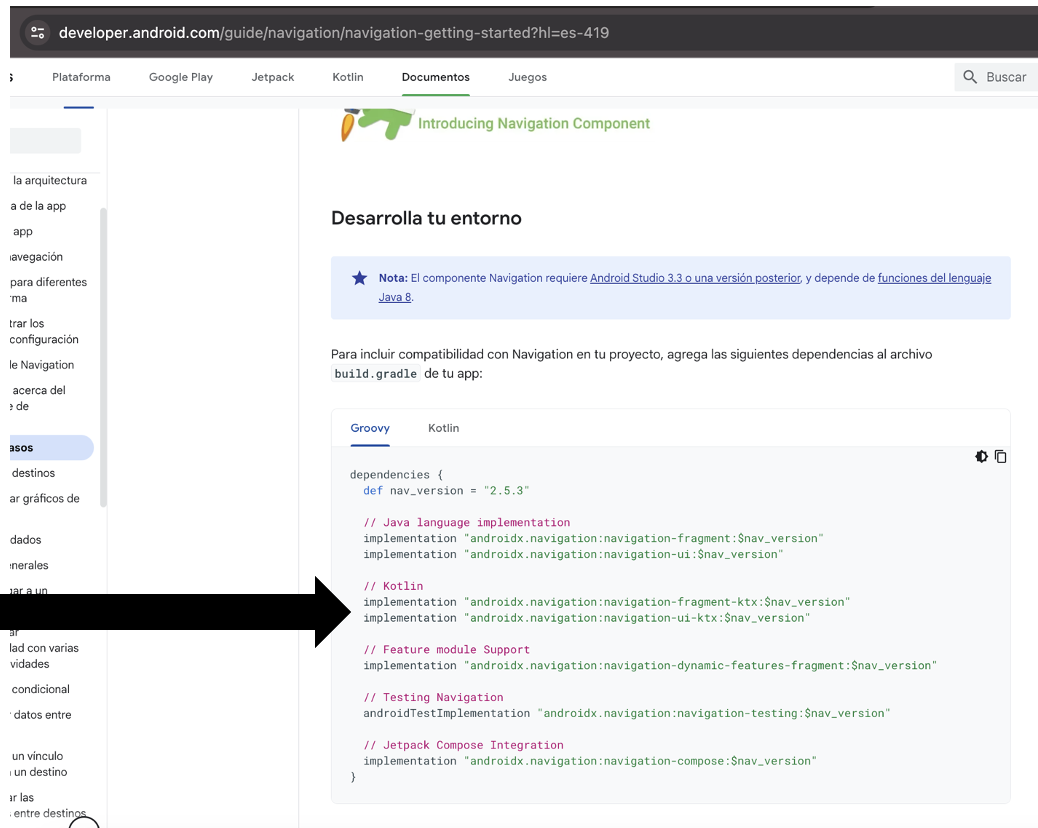


Con esto ya se puede utilizar Retrofit y ya se tiene todo para empezar a trabajar con la aplicación en Android.

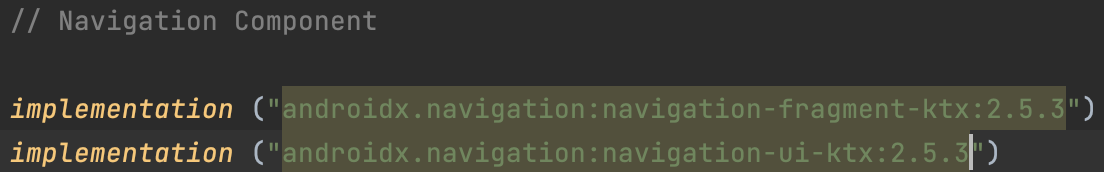
**Navegación**

Se van a incorporar las librerías de navegación ya que vamos a estar usando las librerías de Navigation Component, esto es para facilitar al desarrollador el poder navegar de un destino a otro, ya que cuando un usuario de clic en una noticia se querra que navegue hacía su detalle y tambien se querra una pequeña animación en la transición, ademas que para pasar los datos de una manera segura, también se utilizaran Safe Args por lo que se necesita incorporar Navigation Component y el plugin de Safe Args que se verá dentro del proyecto.

Para esto primero debe en su navegador de preferencia buscar navigation components android y dar clic en el primer o segundo link donde muestre la documentación para comenzar a usar Navigation component y al deslizar la página hacía arriba para ver el texto de abajo (mejor conocido como scroll) podrá ver la parte de dependencies donde tiene la librería que se debe implementar.

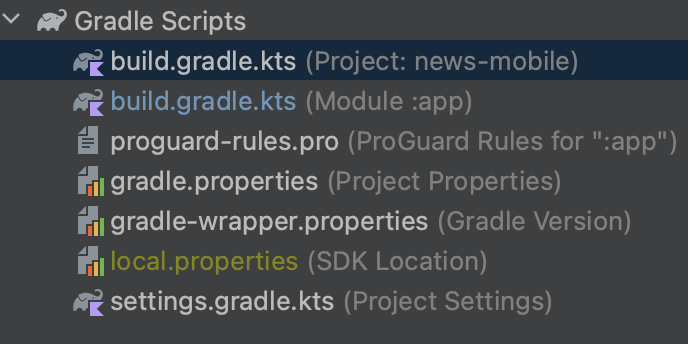


Se le debe poner la versión que indica (def nav\_version = “2.5.3”).



El IDE (entorno de desarrollo integrado) de Android ya nos muestra como se explico anteriormente que hay una versión más actual a la indicada, entonces sólo se hara el segundo paso dando clic en “Change” por la versión que indique el IDE.

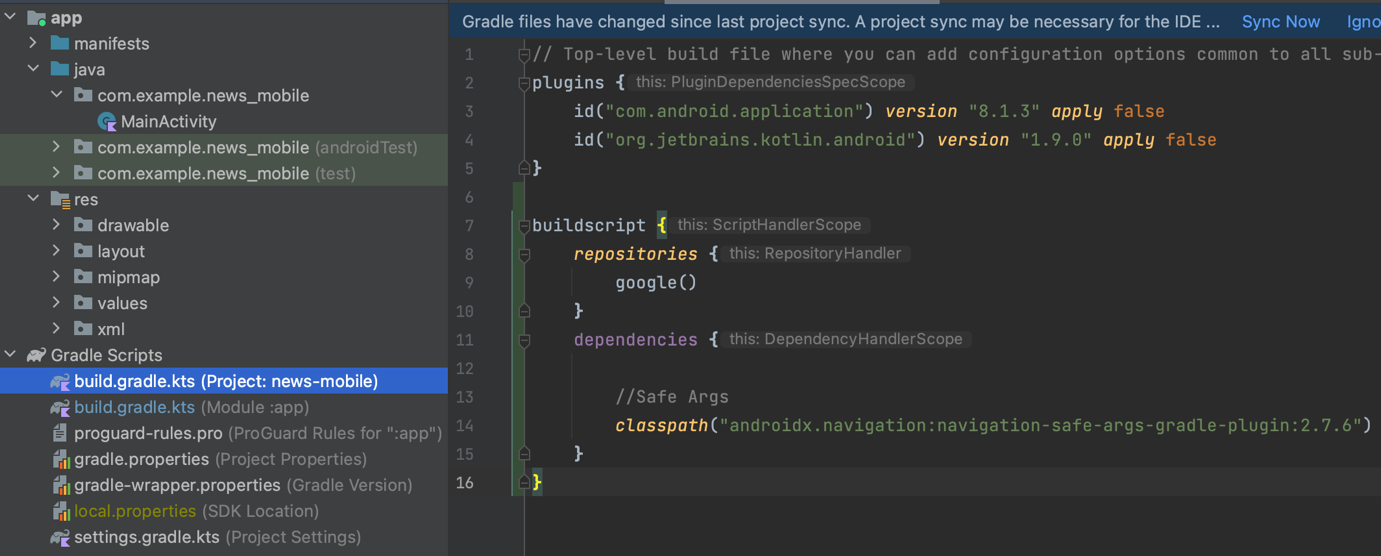
Ahora para implementar el plugin de Save Args se debe ir ahora al build.gradle del proyecto



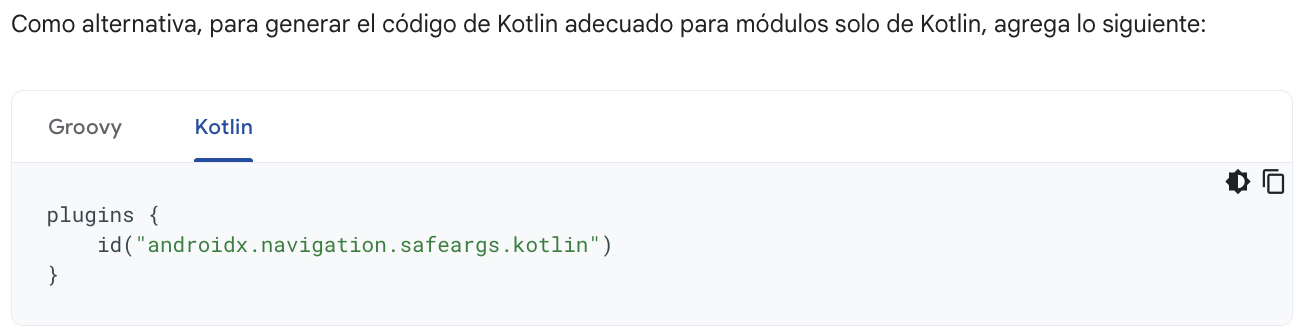
Ir al navegador, escribir Safe Args y en el primer link copiar las líneas de código que muestra:



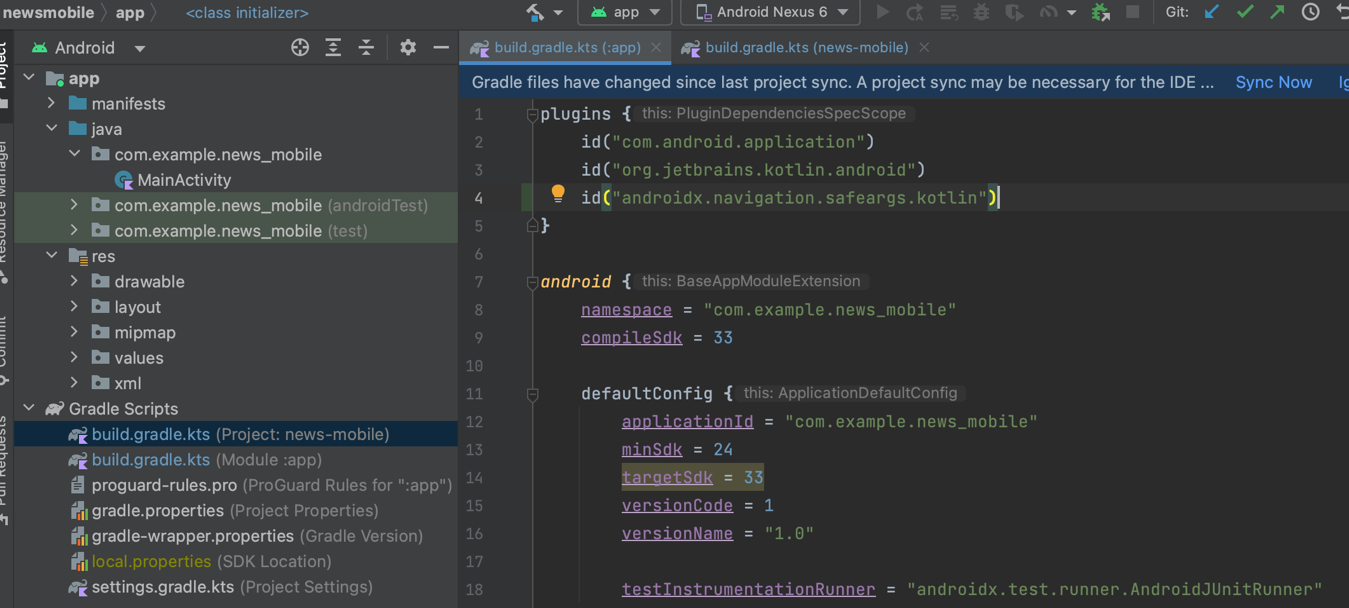
Omitiendo val nav\_version, ya que sólo nos indica la versión del classpath, dejandolo con esa versión o actualizando si el IDE muestra que hay una versión más actual, dejándolo así de esta manera:



Una vez incorporado Safe Args, se regresa a la página web, le hace scroll para poder visualizar más información donde debe copiar el siguiente código:



Una vez copiado lo que se debe hacer es implementar su plugin regresando al Module del build.gradle en la parte superior en el área de plugins y agregarlo:



Para finalizar dándole clic a Sync Now y ya se tendrá navegación en la aplicación con la librería de Navigation y también se tendrá Safe Args para pasar argumentos entre distintas pantallas.

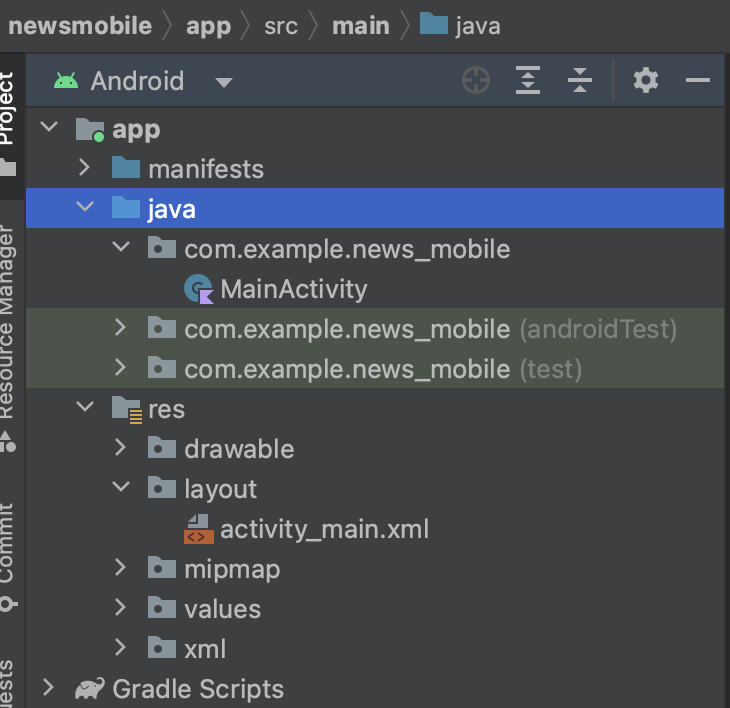
**Creando la interfaz**

Una vez con todo lo anterior incorporado, vamos a crear las pantallas principales, esto para que cuando sean llamadas desde la parte de la capa UI -que se explicará más adelante- ya esten listas.

Lo primero que se debe hacer es crear en la actividad principal llamada Activity Main el contenedor que tendra los fragmentos, pero ¿qué es un fragmento? En palabras de la pagina de Android developer:

“Un Fragment representa una parte reutilizable de la IU de tu app. Un fragmento define y administra su propio diseño, tiene su propio ciclo de vida y puede administrar sus propios eventos de entrada. Los fragmentos no pueden existir por sí solos. Deben estar alojados por una actividad u otro fragmento.”

Ya teniendo una mejor idea de lo que son los fragment, viendo el proyecto se detallaran partes escenciales para la creación de la aplicación lo cual son la parte de manifests, java y res.



En la carpeta java que tres com.example.news\_mobile siendo el primero donde estará el corazón de la aplicación En palabras de la pagina de Android developer:

“java: Contiene los archivos de código fuente de Kotlin y Java, incluido el código de prueba JUnit.”

En la carpeta manifests se tiene el archivo android manifest que detalla la implementación de la aplicación Android. Se especifica su nombre, ícono, diseños, etc. Se mencionan las funcionalidades del contenido asociados a la aplicación. Dentro de ese archivo también se obtienen los permisos que serán requeridos dependiendo de las funciones que se quieran implementar en la aplicación, en el caso de la aplicación a desarrollar se necesitará por ejemplo uso del permiso de internet ya que se obtendran datos de una API.

Y por último se tiene la carpeta res donde se tendra un conjunto de paquetes donde se desarrollará la parte de la vista de la aplicación con la que el usuario estará en constante contacto. Más adelante se le agregará un paquete llamado navigation para la parte navegación entre fragmentos. En palabras de la pagina de Android developer sobre que contiene res:

“res: Contiene todos los recursos sin código, como cadenas de IU y, además, imágenes de mapa de bits.”

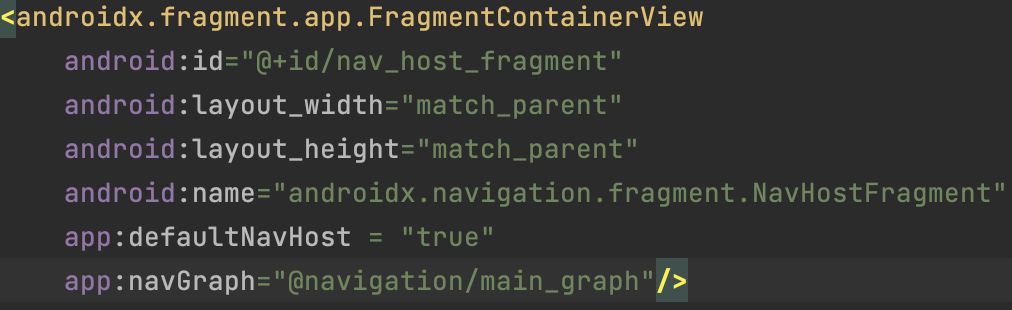
Regresando a la creación del Activity Main vamos a crear el contenedor de los fragmentos



Se debe crear un paquete de navegación dentro de res que contendra el archivo de navegación llamado main\_graph

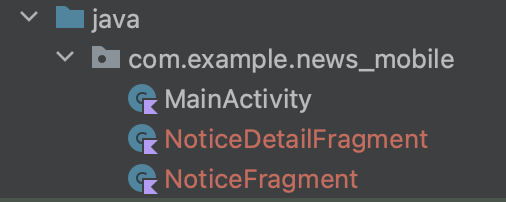


Una vez creado el recurso de navegación se debe incorporar en el contenedor de los fragmentos, ademas de agregar el name y su id al contenedor

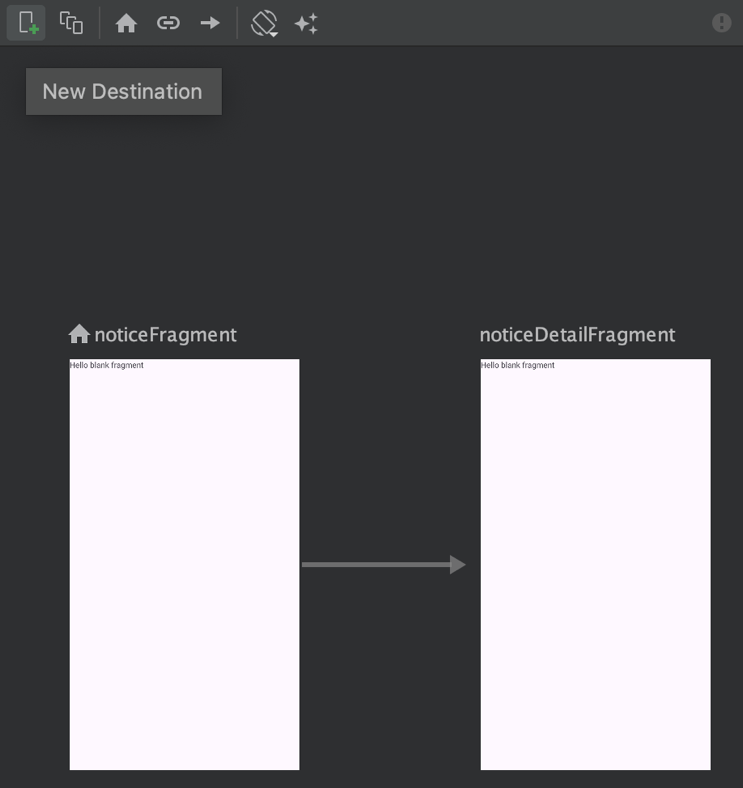


Ahora debe crear dos fragmentos, uno donde se visualizará todas las noticias y otra que mostrará sus detalles al momento de ingresar en esa noticia.

El primero se llamará NoticeFragment y el segundo NoticeDetailFragment.



Al crear estos fragmentos se deben agregar en la parte de navegación en el main\_graph y seleccióna haciendo clic en el icono del telefono con el más verde y agregando el primer fragment y despues el segundo arrastrando el punto de la derecha del primero al segundo para que se seleccione, teniendo así el grafico de navegación como se muestra a continuación:

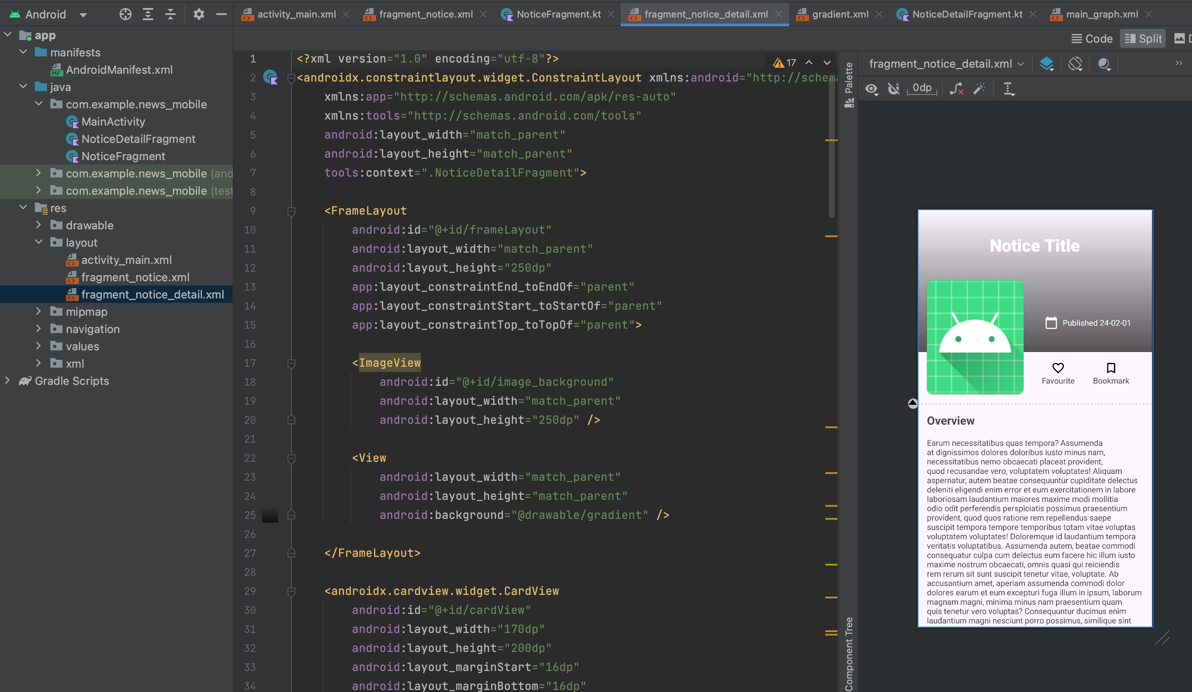


Ahora se trabajará en sus layout que es donde se le dara el formato de como se visualizarán estas pantallas, iniciando por el fragment\_notice ya que es el más sencillo,

Para poder mostrar el listado de noticias se ocupara algo que se llama RecycleView, esteRecycleView es una lista de información y utilizando Un layoutManager para darle el formato que en este caso será LinearLayoutManager.

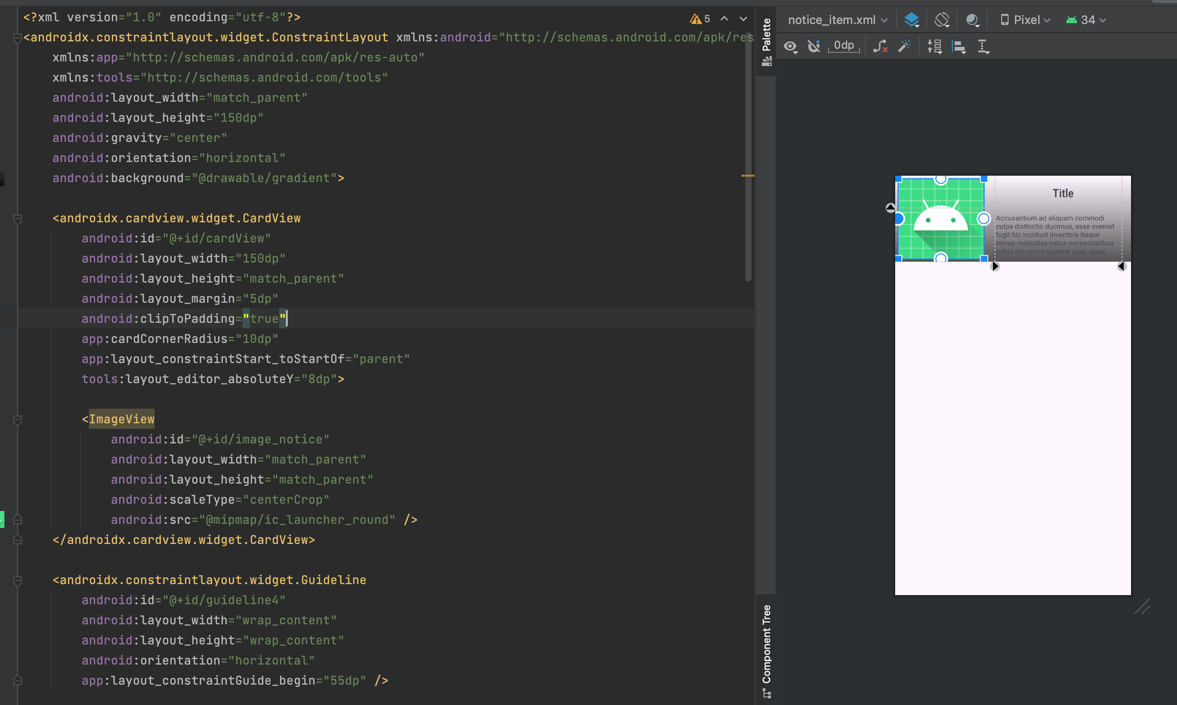


Continuando con el fragment\_notice\_detail donde lo primero que se debe hacer es cambiar el FrameLayout que viene por defecto con un constraintayout, depues se reemplaza el TextView que viene por default y se cambia por un FrameLayout, el cual sirve como un marco donde se tendrá una jerarquía de vistas puede ser una imagen o un texto y se puede poner esa jerarquía de vistas una detrás de otra, como por ejemplo podemos tener un imageview y arriba de eso podemos tener otra imagen u otro texto.

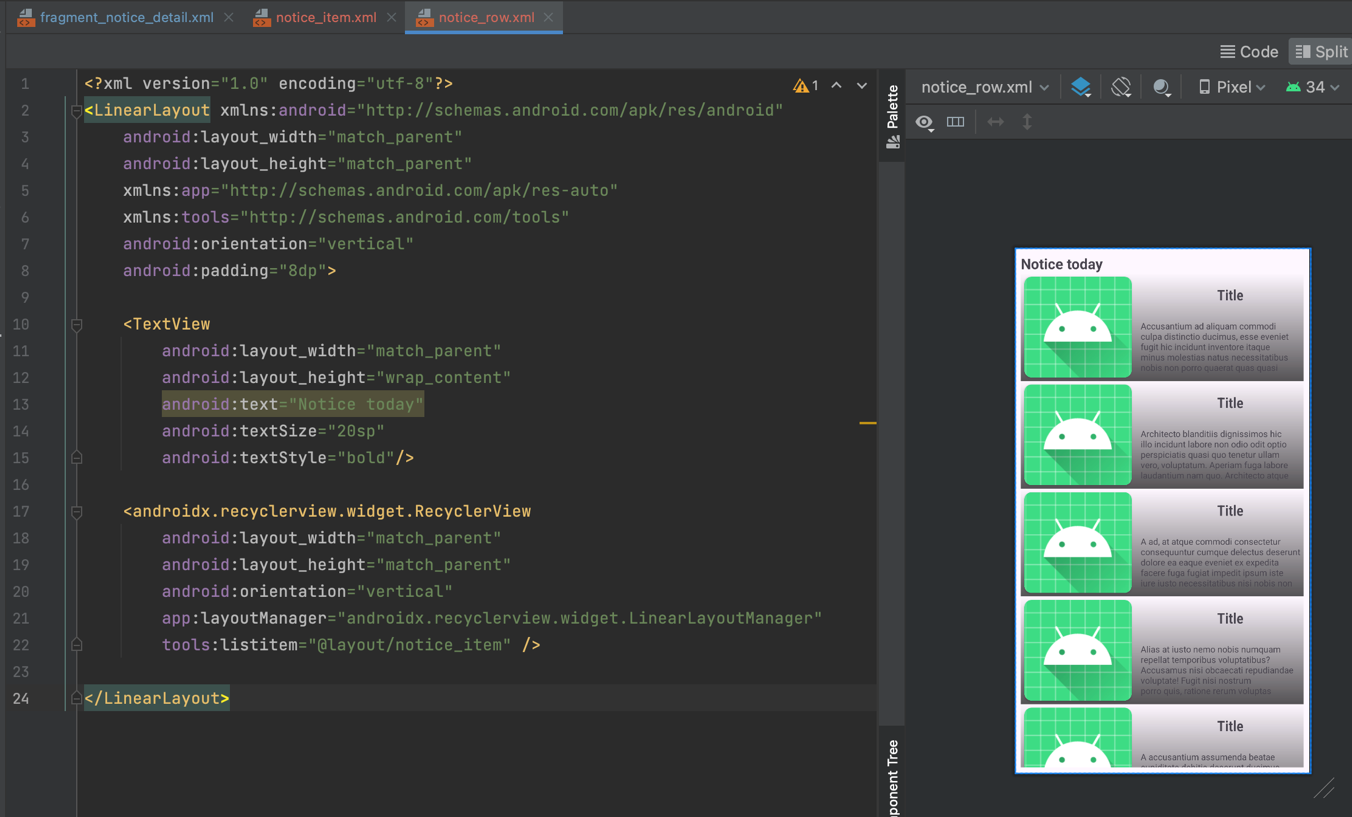


Obteniendo una visualización ya con más forma, para la parte de descripción se utilizó un tools para utlizar un lorem siendo así que no se verá en la versión final pero se pueda mostrar como es que se verá la descripción de la noticia.

Ahora se desarrollará unas nuevas layout que es cómo se verán las noticias, por lo que estás layout serán las que mostrarán como se verán cada elemento en particular.



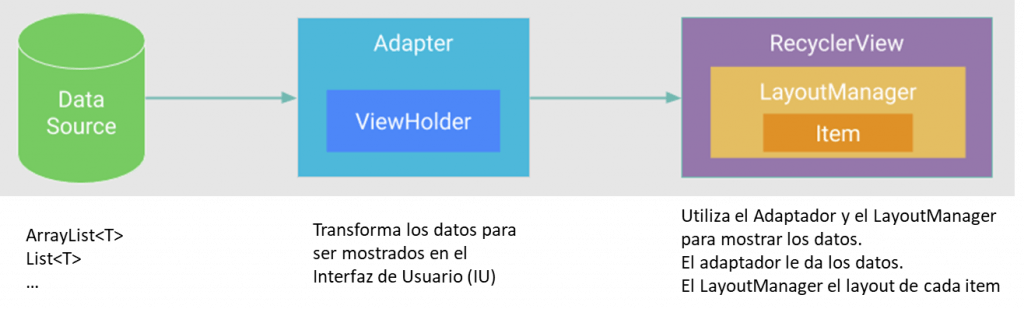
Se comienza diseñando el layout del item como se muestra en la imagen anterior, siguiendo por la creación de notice\_row, row -hace referencia a una columna- donde con el uso de RecyclerView, utilizando el layoutManager y dando un formato LinearLayoutManager, podremos hacer el formato de lista



Pero ¿cómo funciona RecyclerView? en palabras de la pagina de Android developer sobre RecyclerView:

* “RecyclerView facilita que se muestren de manera eficiente grandes conjuntos de datos. Tú proporcionas los datos y defines el aspecto de cada elemento, y la biblioteca RecyclerView creará los elementos de forma dinámica cuando se los necesite.Como su nombre lo indica, RecyclerView recicla esos elementos individuales. Cuando un elemento se desplaza fuera de la pantalla, RecyclerView no destruye su vista. En cambio, reutiliza la vista para los elementos nuevos que se desplazaron y ahora se muestran en pantalla. Esto mejora en gran medida el rendimiento y la capacidad de respuesta de tu app y reduce el consumo de energía.”

Aquí una imagen demostrativa de su arquitectura cómo funciona RecyclerView:

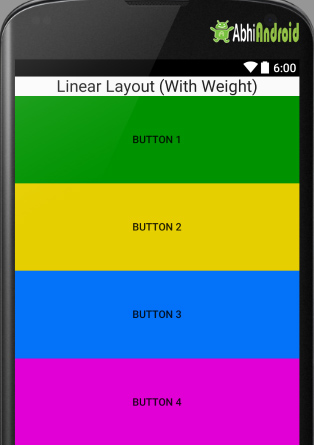
(obtenida de la página https://umhandroid.momrach.es/basicrecyclerview/)

Entonces podemos concluir que un RecyclerView que es donde se van almacenar cada una de las noticias, despues es importante definir el layoutManager que se puede implementar de dos formas:

1. Agregando el layoutManager directamente en el XML.
2. Agregando el layoutManager por código en la clase con la que se trabajará la vista.

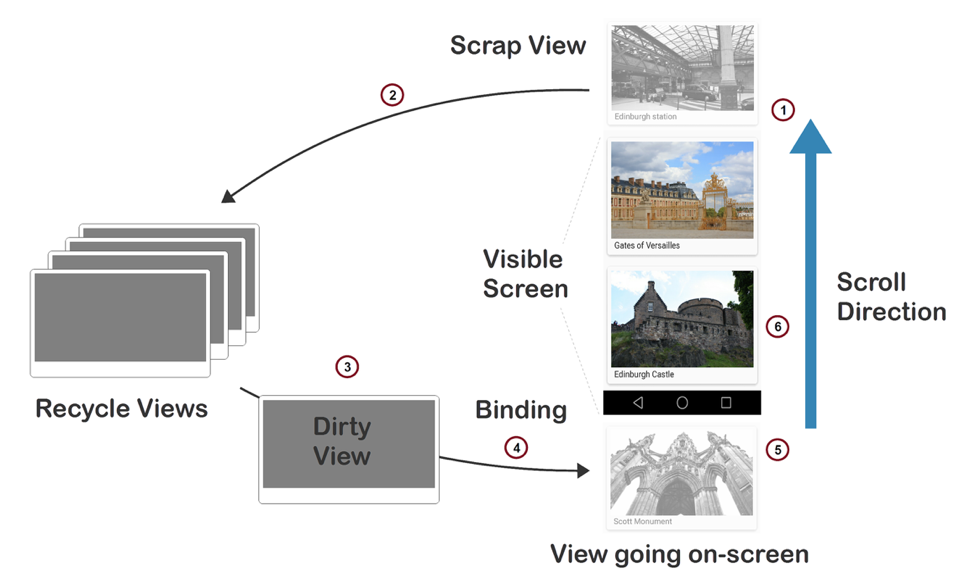
Sin embargo la opción A es la forma más sencilla ya que el desarrollador se ahorra el tener que escribir y describir ese código en la clase de la vista.

El layoutManager se define en el XML primero poniendo su orientación, en este caso es vertical y luego se define como tal el layoutManager que en este caso se utilizará de forma LinearLayoutManager (como se muestra en la imagen de referencia).



(obtenida de: https://abhiandroid.com/ui/linear-layout#gsc.tab=0)

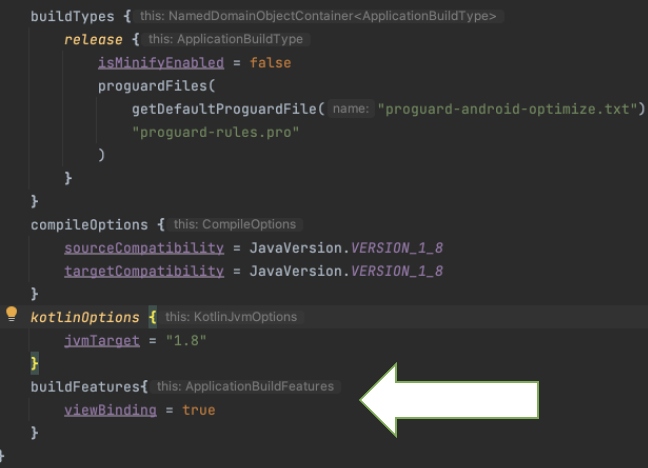
Despues se necesitará un adaptador, el adaptador es el encargado de tomar la información dentro del data set -porque el data set va a venir con información de noticia- por lo que vamos a tener una lista de noticias, esta lista de noticias se le va a pasar al adapter, el adapter va a tomar esa lista de noticias y va saber cómo ponerles información a cada uno de los elementos de la lista en el RecyclerView. Aquí otra imagen de muestra sobre este funionamiento pero más grafico:



(obtenida de https://learn.microsoft.com/es-es/xamarin/android/user-interface/layouts/recycler-view/parts-and-functionality).

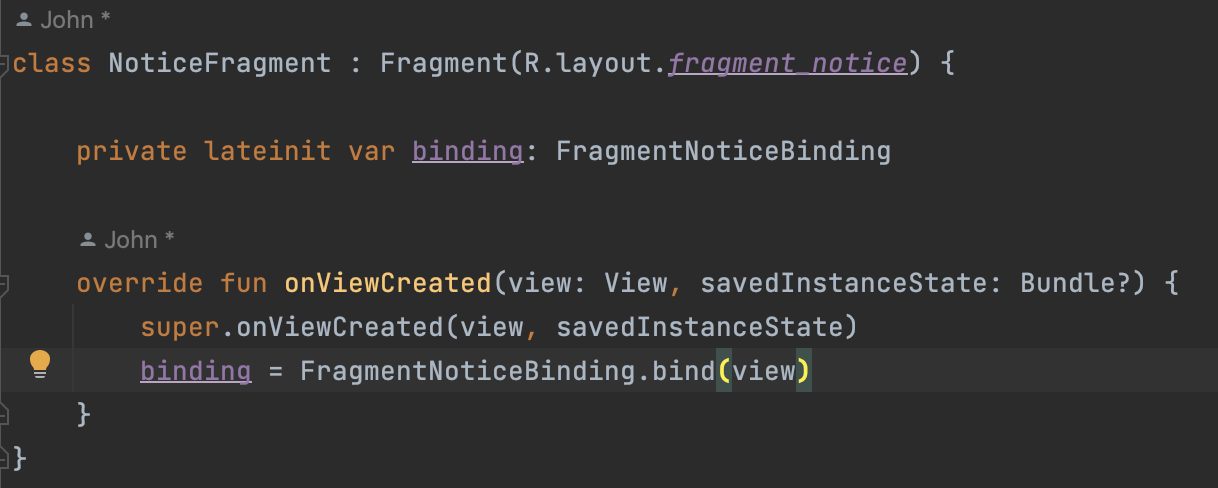
Veremos a continunación como usar View Binding el cual es utilizado para poder hacer referencia a las vistas que tenemos dentro de fragmentos o actividades.

Para poder hacer uso de viewbinding se debe implementar en el Modulo del proyecto agregando buildFeatures de esta forma:



Lo que hará esto es genera una clase binding para cada uno de las layout

Por lo que Binding es una clase que se genera para hacer referencia a todos los elementos que estan dentro de la layout ejemplo:



Si de hace control clic sobre  los llevará directamente a la layout, también se ocupo **private lateinit var** porque se va a inicializar el binding luego y lo más pronto posible.

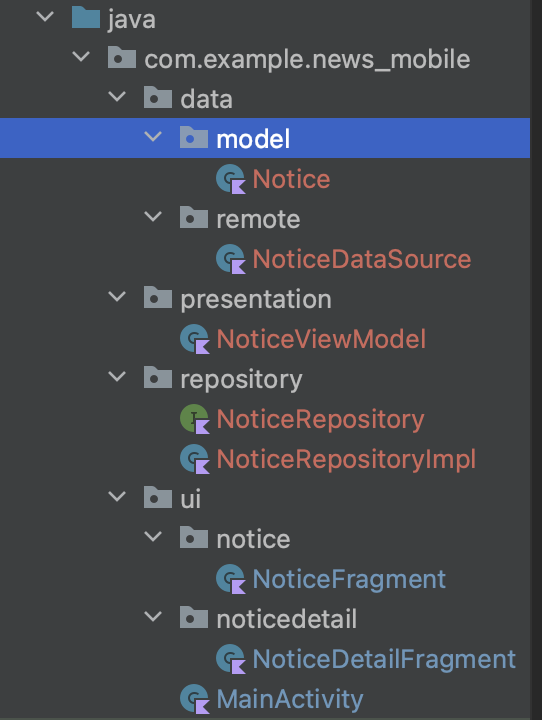
Aquí ya se inicializó ese binding:



View dentro de los parentesis es la vista del layout.

Se replica lo mismo en NoticeDetailFragment.

Una vez hecho todo lo anterior se pasará a darle orden al proyecto con la arquitectura MVVM. Quedando así la maqueta de nuestro proyecto:



**UI**

En este paquete se tendrá todas las interfaces que se vayan creando en el proyecto que son las que visualizará el usuario. O sea donde se mostrarán las noticias y donde se mostrarán sus detalles. Cada una de esas clases tiene sus subpaquetes para que se pueda organizar de una mejor manera.

**Presentation**

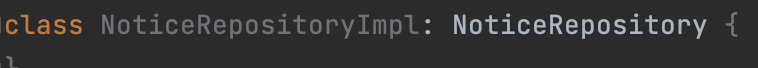
Este paquete tendrá dentro el ViewModel o los ViewModel creados (en el proyecto será el NoticeViewModel) -ya sea dependiendo el caso se puede tener un ViewModel por vista o un ViewModel compartido para más de una vista- en este proyecto sólo es un ViewModel ya que se tienen pocas pantallas, el cual hará referencia a las vistas del UI.

**Repository**

Dentro de repository vamos a tener la implementación de las clases que son :

* Las interfaces (donde se definiran los metodos y donde se tendrá la firma de estos mismos para buscar la información al servidor).
* El repositorio que implementará la interfaz.
* WebService.

Primero se crea la clase de la interfaz (NoticeRepository) y luego la de la implementación del repositorio(NoticeRepositoryImpl), ya creados se le debe implementar al segundo (NoticeRepositoryImpl) el primero (NoticeRepository) de está forma:



Y así ya se tiene el repositorio configurado para las operaciones que se van a requerir del servidor más adelante.

Por lo tanto lo que se necesita es una interfaz para definir esas acciones y se necesita un lugar que implemente esas aciones que despues van a buscar la información requerida.

**Data**

Es la encargada de buscar la información al servidor por lo que es donde se implementa la lógica del proyecto para traer la información buscada. Este paquete hace refencia a datos, entonces esos datos pueden ser objetos a los cuales podemos acceder en un futuro o utilizar en el proyecto.

Dentro de este paquete se tiene:

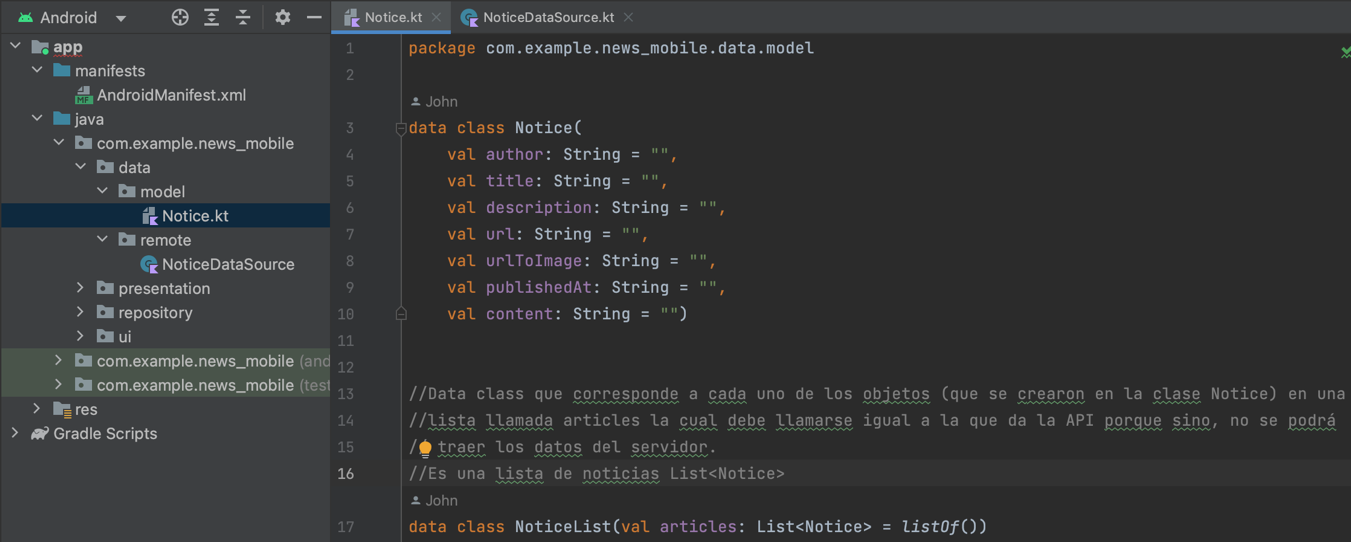
* Model (Aquí estará el objeto noticia).
* Remote (Sirve porque se buscará la información a una API y no se está buscando la información localmente, lo cual para mayor organización también se agregaria otro paquete dentro de data llamado local si se requiere buscar información de forma local).

En resumen, a través de el fragmentnotice se va a implementar el ViewModel que desde este fragmento se hará la llamada al ViewModel solicitando la información, despues el ViewModel lo que va hacer es ir al Repositorio y decirle que necesita cierta información, ¿de donde viene esa información? El ViewModel no tiene porque saber de donde se obtiene la información, no le interesa de donde venga sólo hace esa petición al repositorio.

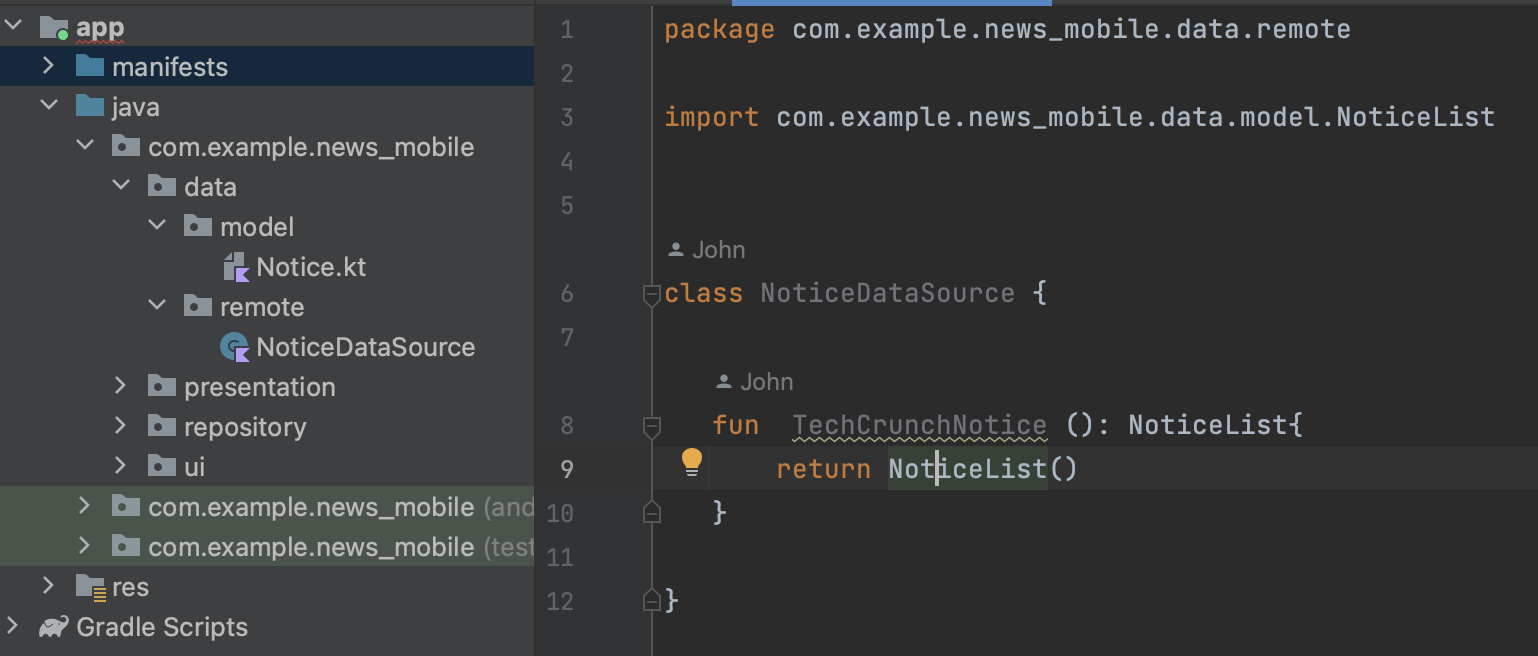
El repositorio implementador es quién decide si va a buscar la información en una base de datos local o la va a ir a buscar a un servidor, haciendo así una llamada a Data, que es donde viene la información, esa llamada se hace en este caso al DataSource que trae la información del servidor para finalizar el DataSource es el encargado de buscar la información al servidor con Retrofit, trae esa información y se la da al Repositorio implementador, estando ahí vuelve esa información al ViewModel, el ViewModel al tener la información necesita pasarle esa información a la vista. Para eso se utilizará LiveData, este emitirá ese resultado (valor) para que en la vista se pueda recibir, la vista lo que hará es observar ese valor del LiveData, y así es como se verá reflejado en pantalla ese valor o resuitado obtenido del servidor.

Ya entendiendo lo anterior, teniendo ya las vistas que se hicieron -reiterando- para tenerlas listas y preparadas para luego ser usadas en la capa de UI, por lo tanto ya se puede arrancar con el proyecto, empezando por la funcionalidad que es lo más importante en el proyecto, ¿cuál es esa funcionalidad? el paquete data, donde se tiene el modelo con el cual se empezará.

Model:



Remote:



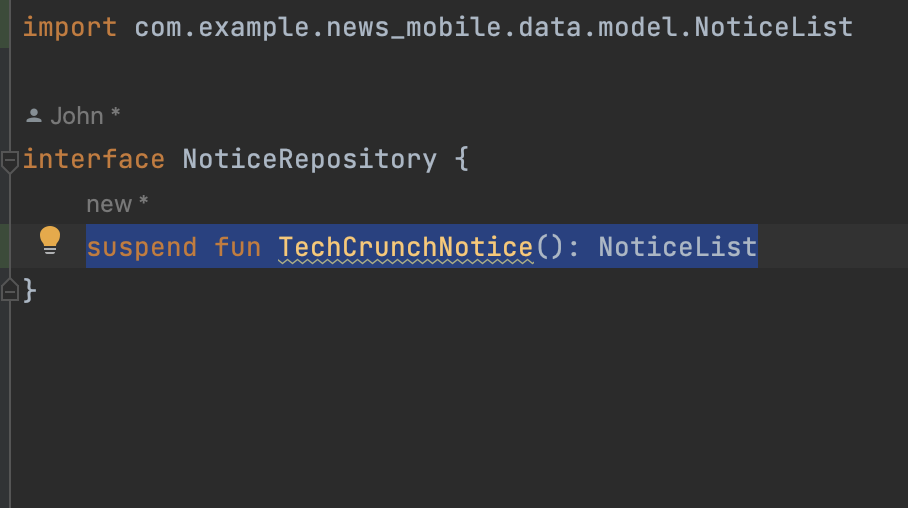
Repositorio

Interface (NoticeRepository):

Lo que hace la interfaz es simplemente definir los métodos que se van a usar luego en la implementanción y despues poder usar la interfaz para crear instancias de repositorio. Es decir la interfaz funciona para generar esa “firma” de esos métodos, solamente el método declarado, no su implentación.

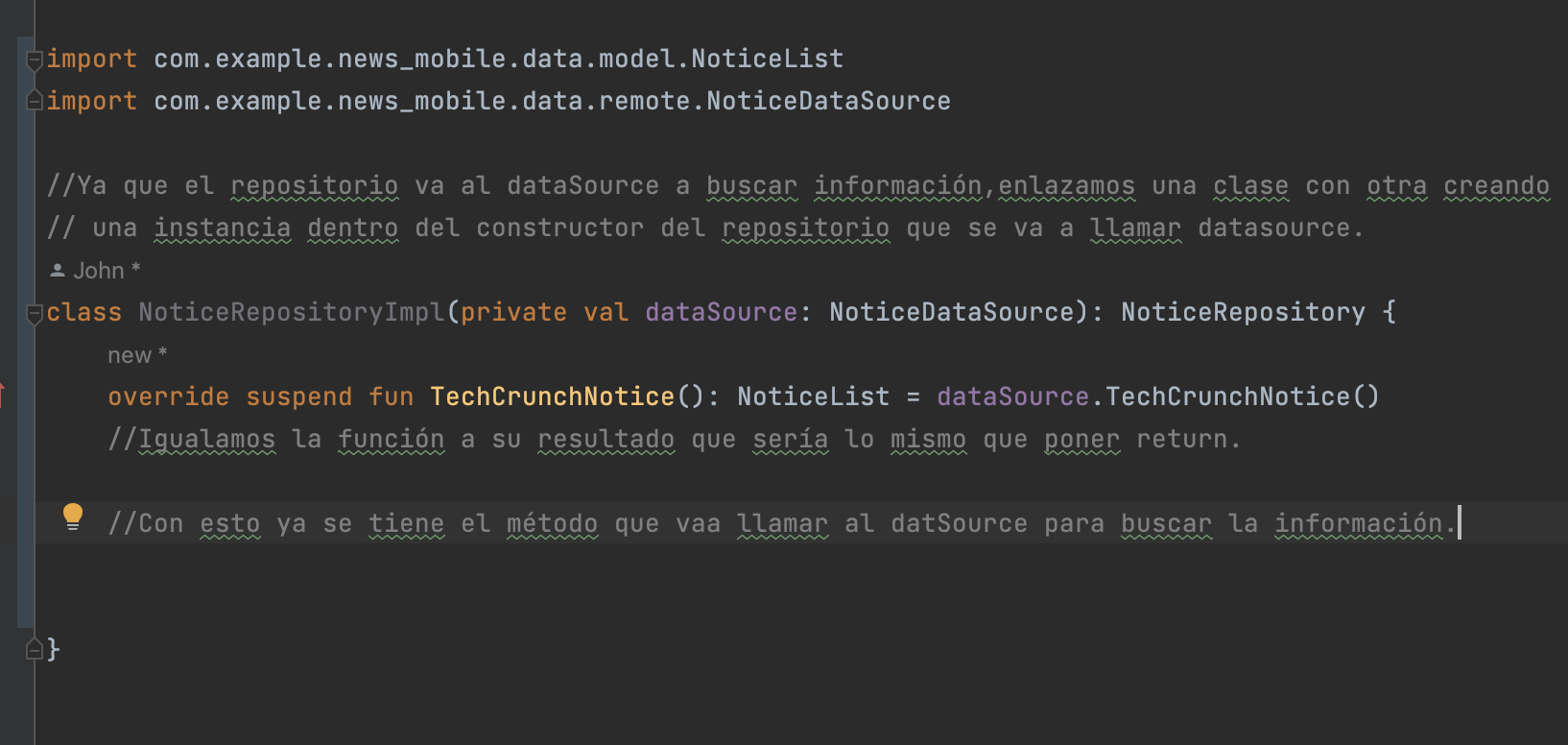
Para la creación de los métodos se ocuparán corutinas, las corutinas son funciones que tienen un característica, la cual las hace suspendables, se suspende esa función hasta que no realiza una acción como ir a buscar información al servidor, no termina su ejecución, una vez que vuelve el servidor de traer información(retornar data) ese suspend function, continúa, se hace un resume de esa función y se dan los datos (o se imprime un error). Entonces para usar estas corutinas se utiliza suspend antes de la función





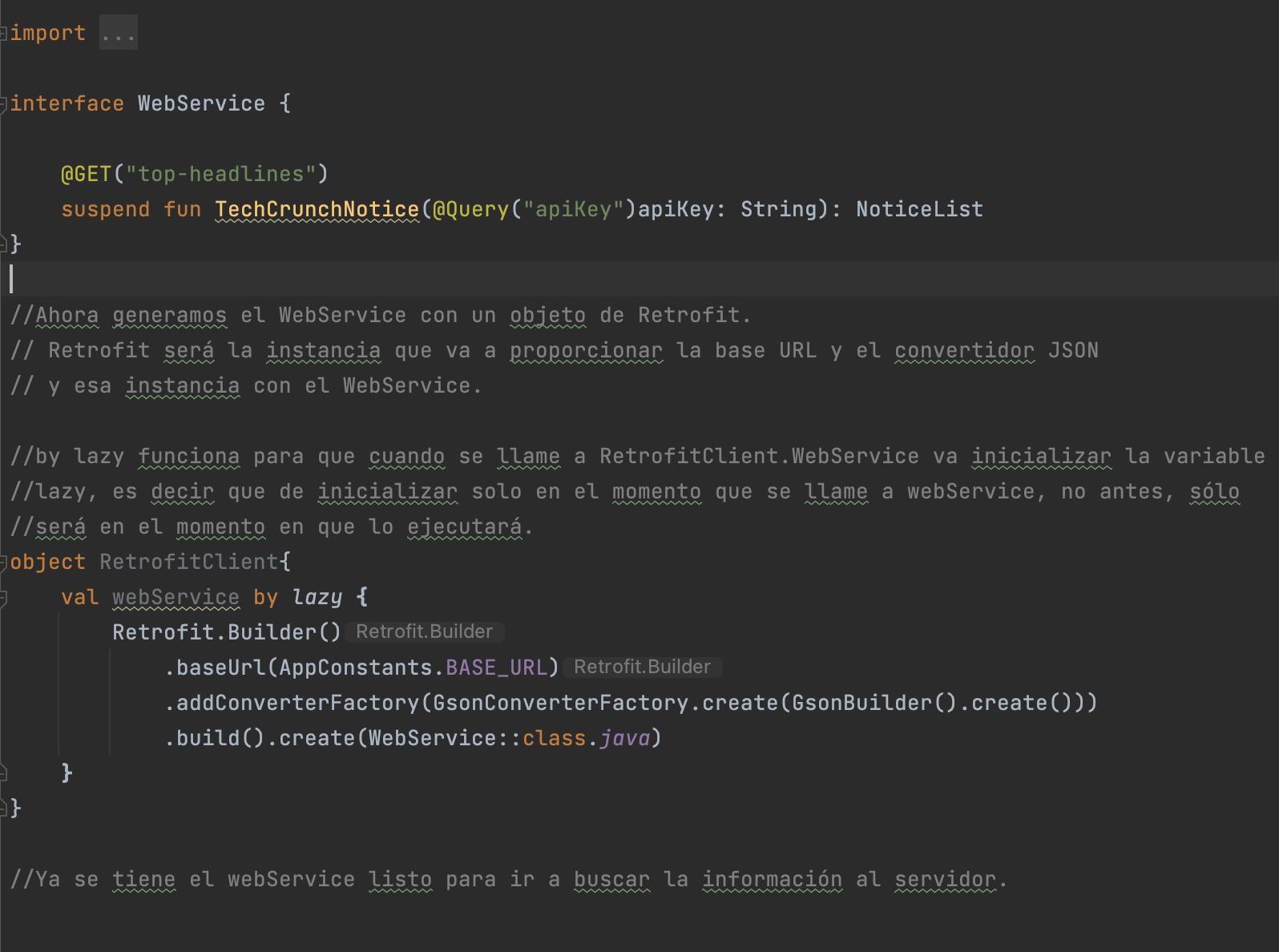
Clase (NoticeRepositoryImpl):

Aquí ya se debe tener los método con la “firma” para poder implementar lo que devuelva de forma más sencilla.



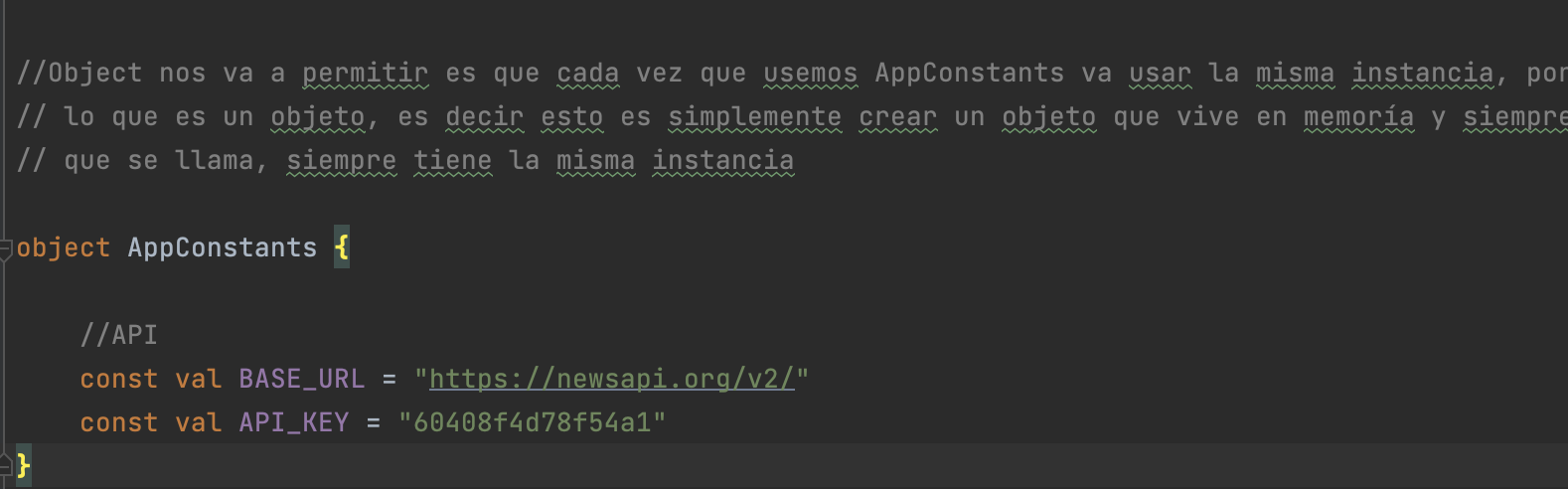
WebService:

Es el encargado de usar Retrofit para traer la información del servidor, lo que va hacer el servidor es usar los métodos GET para hacer la petición de la información, esos métodos se usan con un anotador que va arriba de la función



Entonces ya se puede directamente en el repositorio llamar a los métodos del DataSource y desde el DataSource se puede usar el webService para consultar datos al servidor.

Para hacer la sección anterior creamos un objeto donde adentro tiene la base Url y el Api Key necesario para hacer correctamente las llamadas al servidor.



ViewModel

Esta clase será para retornar tres estados para retornar estados y para poder trabajarlos en

kotlin hay algo llamado Sealed clases (clases selladas), que funcionan para este proposito en especial. Esto de las clases selladas y el uso de genericos siempre se ocupan según fuentes de internet cuando uno crea proyectos en kotlin -junto con el uso de genericos- y sólo se lo se haran una sola vez.

Se importa lifecycle ya que el ciclo de vida y el ViewModel están muy dados de la mano y trabajan en conjunto para poder garantizar a la interfaz que siempre tendrá los ultimos datos que ha solicitado en el servidor.

Para hacer la llamada que va a ir al servidor a buscar información, se necesita asignarle un hilo de ejecución, hay varios hilos de ejecución para correr una corutina pero el más usado

es Dispatchers.IO, los más usados son .IO, .Default, .Main, pero el más usado en los proces de servidores es, ir a buscar información es .IO, si se quiere hacer procesos con un mayor nivel de computo se puede usar Default y si se van a correr procesos en el main thread o en el mismo hilo principal donde corre la aplicación se puede ocupar main.

Usaremos un try catch para correr dentro la corutina



Paqueteria concat:

Creación de la clase NoticeAdapter

Dentro del adapter, como el RecyclerView.Adapter pide un ViewHolder, se le dará el BaseViewHolder y se le pondrá un asterisco dentro para representar cualquier tipo de ViewHolder, luego pide implementar los métodos.

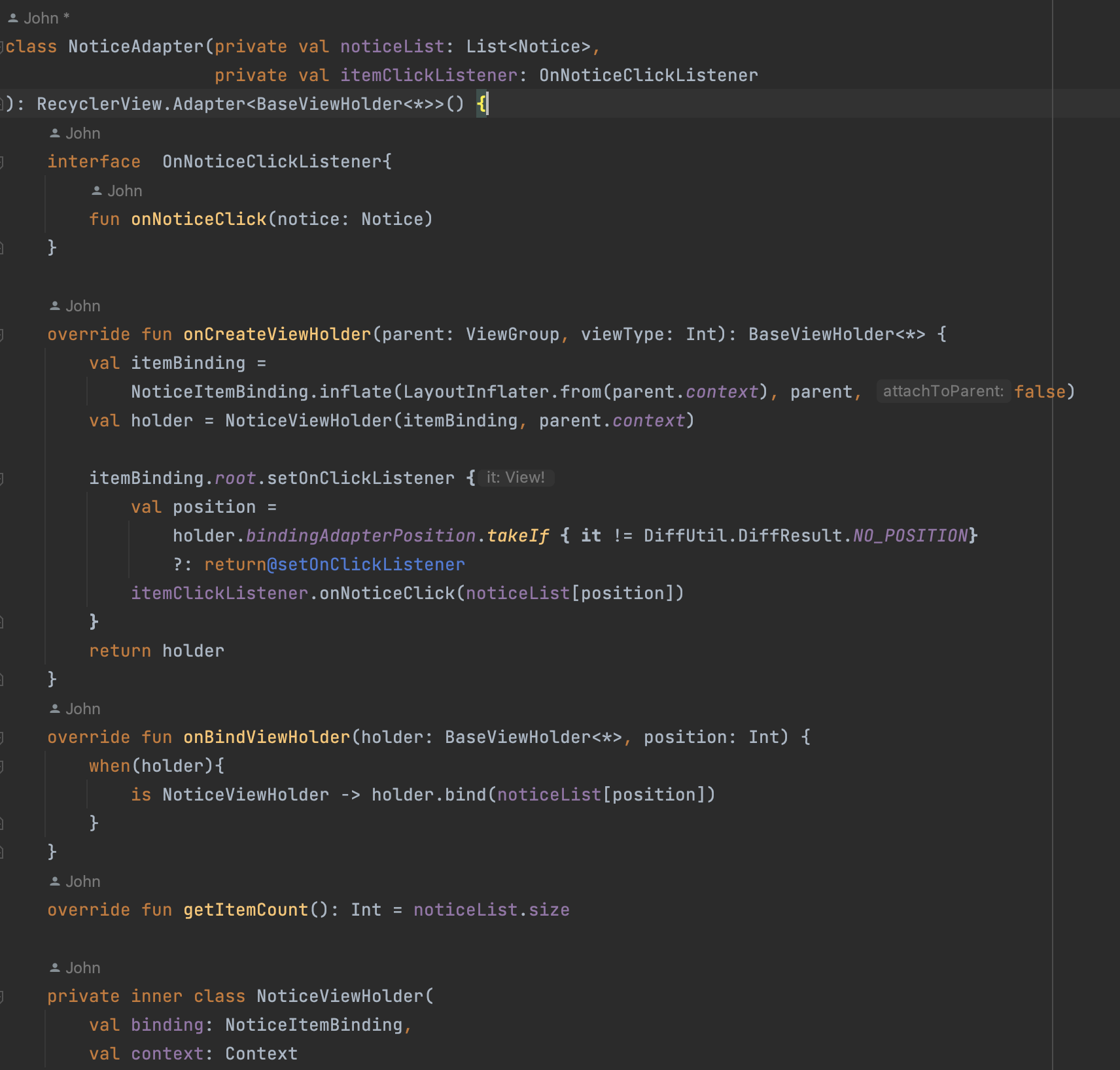
Ahora se debe crear, lo que va buscar la interfaz, que va a inflar cada elemento de la lista. el cuál será itemBinding que llevará a NoticeItemBinding, que cómo se mencionó con anterioridad binding facilitará acceder a la vista. Lo que se de hacer es llamar al layout que infla esa vista, es decir llamarla. En resumen lo que hace itemBinding es simplemente lo mismoq que cuando se hace el contenido en una activity para inflar una layout con R.layout o en un fragment cuando hacemos el inflate del onCreateView.

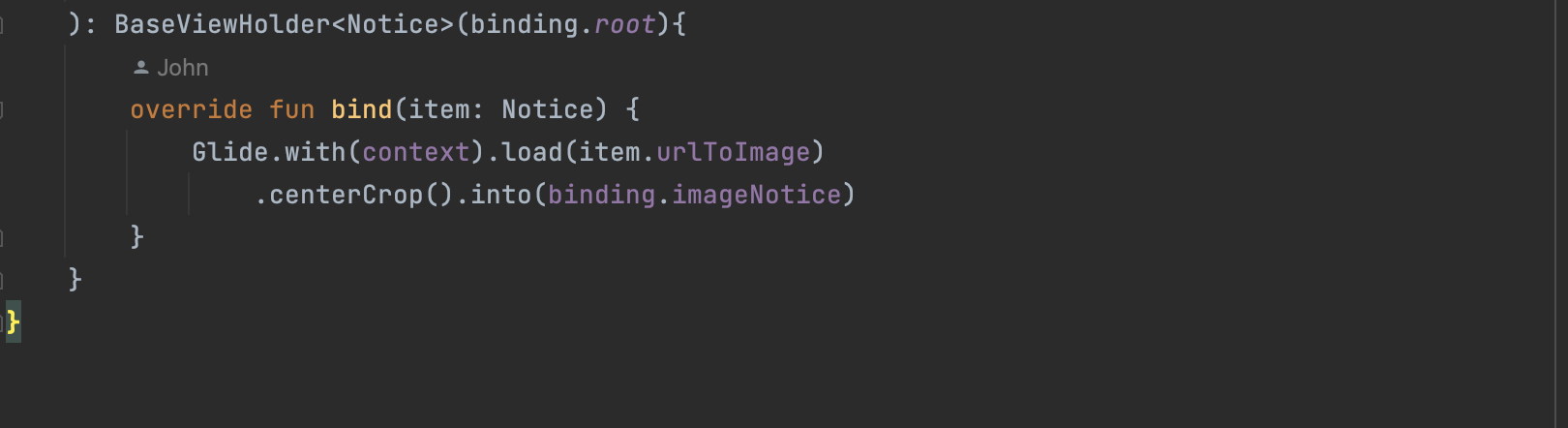
Luego se creará un Viewholder, con un inner class, da la posibilidad que al ser inner esta dentro de la clase padre que es NoticeAdapter, inner quiere decir que cuando la instancia NoticeAdapter muera, también la instancia de su inner class, si se ocupa sin el inner cuando NoticeAdapter puede que el el Holder del View Holder - que es la clase abajo-, puede que este objeto quede en memoria, ocasionando un alojamiento de memoria inecesario de un objeto que no se estará ocupando.

binding.root hace referencia a toda la layout completa podiendo acceder a ella.

Para poder cargar la imagen se ocupará la librería Glide la cuál va a permitir cargar

cualquier imagen de internet dentro del imageView.





Paqueteria core:

1.Creación de clase abstracta BaseConcatHolder.

2.Creación de clase abstracta BasViewHolder:

Esta clase se ocupará paratodo, se hará abstracto, es decir se podra rehusar en donde se quiera y va a facilitar la cración de adaptadores. BaseViewHolder tiene un génerico T que se le puede pasar cualquier tipo de ViewHolder, el cual tendra un itemView que será una vista, el cual va a retornar un RecyclerView.ViewHolder de la vista que se le está pasando, para finalizar lo unico que se requiere es agregar un método más

que es un abstract fun bind el cual se va a encargar de agarrar cada uno de los elementos de la

noticia y ponerle los datos que se requiere, como por ejemplo poner la imagen de la noticia, la descripción, etc.

paqueteria UI en NoticeFragment: se agregaron los "ultimos" pasos que es generar una instancia del adapter.

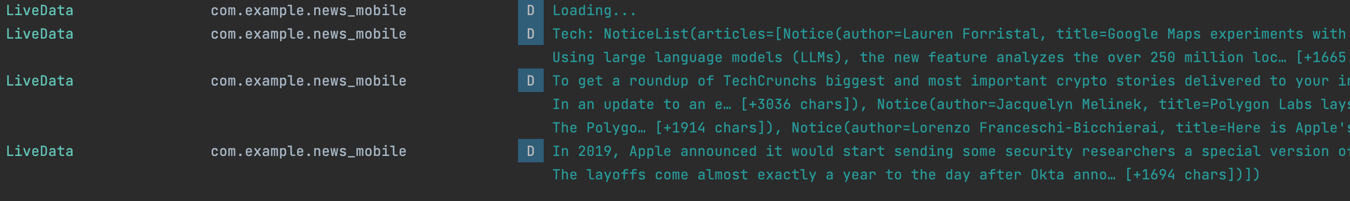
res:

Correción de layout fragment\_notice\_detail.xml

navigation:

se agregaron arguments a al fragment\_notice\_detail.xml.

Se logró la obtención sin problemas de la información del servidor.



Sin embargo se tuvo problemas para hacer posible la visualización de esa información.