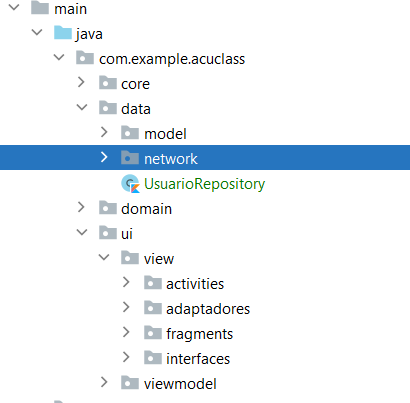
**Reflexión**

**Unidad 3:** Retrofit con Corrutinas

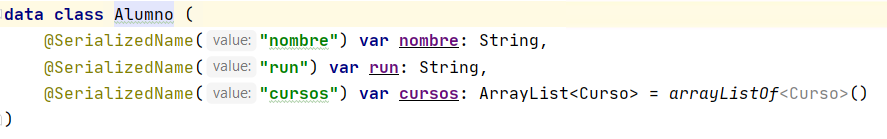
16 de agosto de 2021

* **Aprendizaje:** Implementación de corrutinas y retrofit
* **Reflexión:**  Daily meeting
  + **¿Qué he hecho desde la última reunión de sincronización para ayudar al equipo a cumplir su objetivo?**
    - Implementar retrofit con corrutinas en la aplicación
    - Cambio de la estructura de los package
  + **¿Qué voy a hacer a partir de este momento para ayudar al equipo a cumplir su objetivo?**
    - Mantener el usuario guardado en viewModel (Pendiente)
    - Implementar retrofit con corrutinas en la aplicación (continuación)
  + **¿Qué impedimentos tengo o voy a tener que nos impidan conseguir nuestro objetivo?**
    - No hay impedimentos de momento
* **Link:** 
  + <https://www.youtube.com/watch?v=hhhSMXi0R3E&t=616s&ab_channel=AristiDevs>
  + <https://cursokotlin.com/mvvm-en-android-con-kotlin-implementando-retrofit-corrutinas-y-clean-architecture/>
* **Imágenes:**

**Nueva estructura de la aplicación**



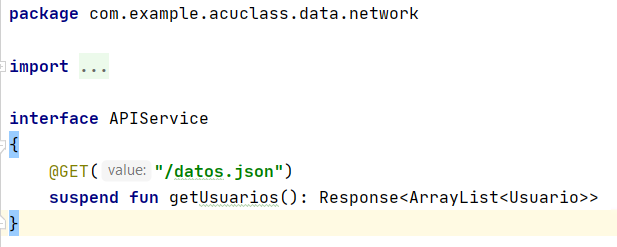
**Serialización de las variables que provienen desde la base de datos para guardarlos en la clase**



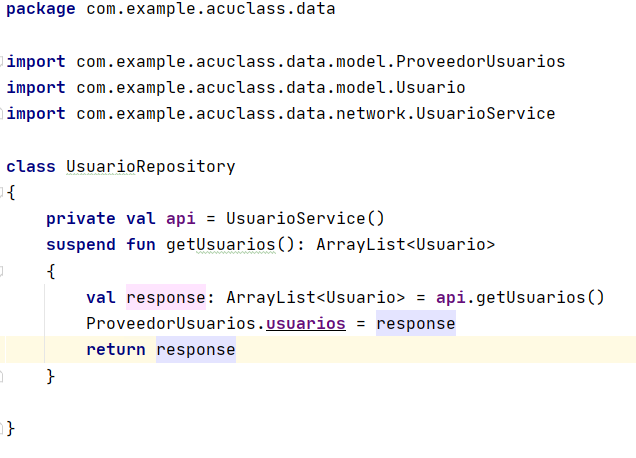
**Uso de retrofit**



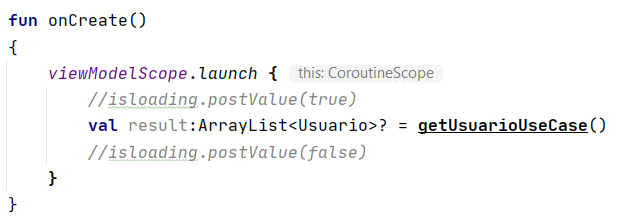
**Llamando a json en donde se encuentran los datos**



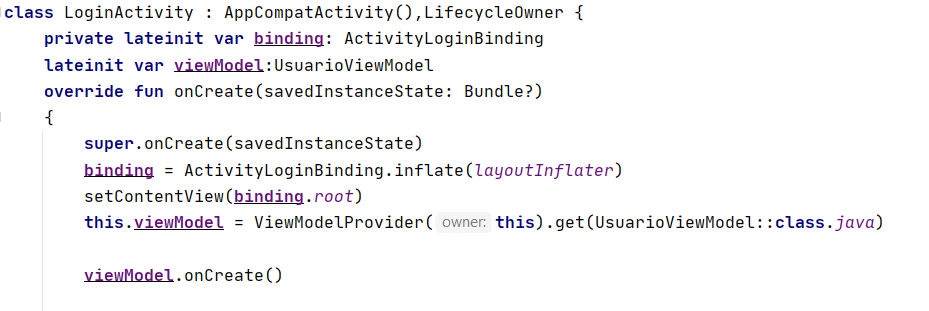
**Llamada a la API y guardar los datos de momento en una clase de proveedor de usuarios**



**Llamando desde el viewmodel al repositorio para que baje los datos al momento de inicializar la vista**



**Llamada al método onCreate() del viewmodel desde el activity login**



**Documentación de la pagina en donde se extrajo la información**

**Flujo completo**

Para el primero de los flujos completos que será el de consumir todas las citas de internet tendremos que ir a nuestro **QuoteViewModel**y veremos varias cosas, la primera que la función **randomQuote()**está fallando ya que modificamos nuestro provider, pero borraremos todo el contenido de esta.

El siguiente paso será crear una función llamada **onCreate()**en nuestro **ViewModel**que lo llamaremos al crear la *activity*.

Obviamente para poder llamar a nuestro caso de uso debemos añadir una instancia de este.

**var** getQuotesUseCase = GetQuotesUseCase**()**

Ahora si podríamos llamarlo en nuestra función **onCreate()** pero si recuerdas, nuestro caso de uso es **suspend** ya que se trata de una corrutina así que para poder lanzarlo desde aquí tendremos que usar **ViewModelScope**, si quieres saber más a fondo sobre su uso te recomiendo que mires el vídeo de arriba.

**fun** onCreate**()** **{**

viewModelScope.launch **{**

**val** result = getQuotesUseCase**()**

**if(**!result.isNullOrEmpty**()){**

quoteModel.postValue**(**result**[**0**])**

**}**

**}**

**}**

Nuestra función **onCreate()** de nuestro **ViewModel** quedaría así. Para darte feedback en este ejemplo simplemente compruebo si el resultado no es vacío ni null y pinto en la pantalla la primera posición (ya que anteriormente no hacía nada hasta hacer el primer clic en la pantalla).

Ahora desde el **MainActivity** simplemente llamamos a esta función desde su **onCreate()**.

override **fun** onCreate**(**savedInstanceState: Bundle?**)** **{**

super.onCreate**(**savedInstanceState**)**

binding = ActivityMainBinding.inflate**(**layoutInflater**)**

setContentView**(**binding.root**)**

quoteViewModel.onCreate**()**

quoteViewModel.quoteModel.observe**(**this, Observer **{**

binding.tvQuote.text = **it**.quote

binding.tvAuthor.text = **it**.author

**})**

binding.viewContainer.setOnClickListener **{** quoteViewModel.randomQuote**()** **}**

**}**

Si ejecutas verás que funciona pero que durante 1 o 2 segundos no se ve nada ya que está cargando la información. El problema es que no le estamos dando ningún *feedback*al usuario así que vamos a ir la **activity\_main.xml** y meteremos un progress bar que no es más que un circulo con animación que hace que el usuario vea que está cargando la app.

**<ProgressBar**

android:id="@+id/loading"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

app:layout\_constraintTop\_toTopOf="parent"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"**/>**

Ahora tenemos que jugar con la visibilidad del componente y para ello lo haremos desde el **ViewModel**. Crearemos un **LiveData** de tipo **Boolean**y con él controlaremos el estado de nuestro *loading*.

**val** isLoading = MutableLiveData**<**Boolean**>()**

Y controlaremos el estado para ponerlo visible antes de la llamada al servidor y para ocultarlo al terminar.

**fun** onCreate**()** **{**

viewModelScope.launch **{**

isLoading.postValue**(true)**

**val** result = getQuotesUseCase**()**

**if(**!result.isNullOrEmpty**()){**

quoteModel.postValue**(**result**[**0**])**

isLoading.postValue**(false)**

**}**

**}**

**}**

Pero quedaría que nuestra **MainActivity** se suscriba a dichos cambios para poder cambiar la visibilidad.

override **fun** onCreate**(**savedInstanceState: Bundle?**)** **{**

super.onCreate**(**savedInstanceState**)**

binding = ActivityMainBinding.inflate**(**layoutInflater**)**

setContentView**(**binding.root**)**

quoteViewModel.onCreate**()**

quoteViewModel.quoteModel.observe**(**this, Observer **{**

binding.tvQuote.text = **it**.quote

binding.tvAuthor.text = **it**.author

**})**

quoteViewModel.isLoading.observe**(**this, Observer **{**

binding.loading.isVisible = **it**

**})**

binding.viewContainer.setOnClickListener **{** quoteViewModel.randomQuote**()** **}**

**}**

Entonces cada vez que el **ViewModel** haga un cambio en ese*boolean*, automáticamente aparecerá o desaparecerá el *progress bar.* Si ejecutas la app verás que ahora si va perfecto.

Ahora solo queda solucionar la parte de la cita aleatoria ya que si ahora mismo haces clic en la pantalla ya no se actualizan las citas como en la primera parte. Para ello tendremos que crear otro caso de uso que se encargue de devolvernos una cita aleatoria.

Dentro del directorio **domain** crearemos un nuevo caso de uso y lo llamaremos **GetRandomQuoteUseCase**.

class GetRandomQuoteUseCase **{**

operator fun invoke**()**:QuoteModel?**{**

val quotes = QuoteProvider.quotes

**if(**!quotes.isNullOrEmpty**()){**

val randomNumber = **(**quotes.indices**)**.random**()**

**return** quotes**[**randomNumber**]**

**}**

**return** **null**

**}**

**}**

El comportamiento es muy sencillo (y poco correcto) ya que lo primero que estamos haciendo es acceder directamente a la memoria en vez de pasar por nuestro repositorio, pero como es una corrutina no quiero complicar más esta segunda parte. Luego comprobamos si no es null ni vacío y entonces generamos un número *random* entre cero (la primera posición del listado) y la última posición del listado para que devuelva una cita aleatoria o sino hay devolverá un null.

Volvemos a nuestro **QuoteViewModel**para completar la función **randomQuote()**.

var getRandomQuoteUseCase = GetRandomQuoteUseCase**()**

fun randomQuote**()** **{**

isLoading.postValue**(true)**

val quote = getRandomQuoteUseCase**()**

**if(**quote!=**null){**

quoteModel.postValue**(**quote**)**

**}**

isLoading.postValue**(false)**

**}**

Ahora ya tenemos nuestra app completa (al menos para esta segunda parte) y si ejecutas la app verás que todo funciona de maravilla.