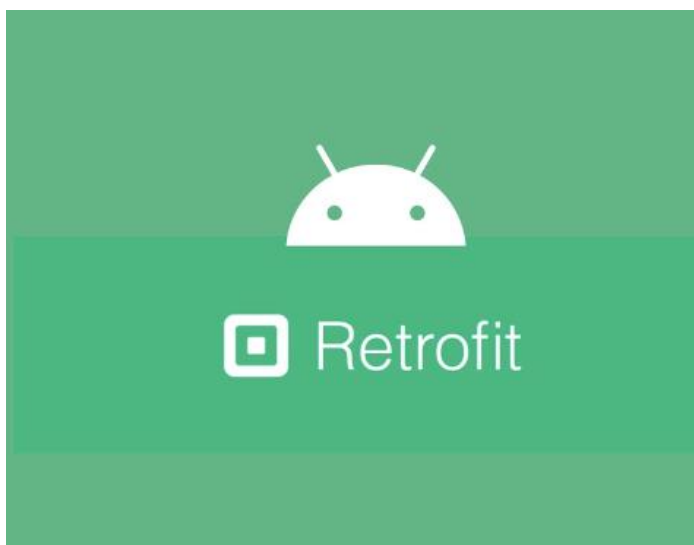


Programación multimedia y dispositivos móviles

Guía de un proyecto usando Retrofit.

2º de Desarrollo de Aplicaciones Multiplataforma



Por Agustín Rodríguez Márquez



IES Rafael Alberti

Este ha sido un proyecto basado en Retrofit, para hacer el proyecto me he guiado por un codelab de Google, disponible para mostrar un primer contacto con esta tecnología y que me ha hecho entender de forma sencilla como funciona Retrofit, y cuál es su aplicación.

El proyecto se enfoca principalmente en la capa de datos de la app y en como conectarse a internet para poder descargar datos desde una app Android, lo cual me ha parecido interesante.

El proyecto está basado en una arquitectura MVVM, como buena práctica de programación Android:

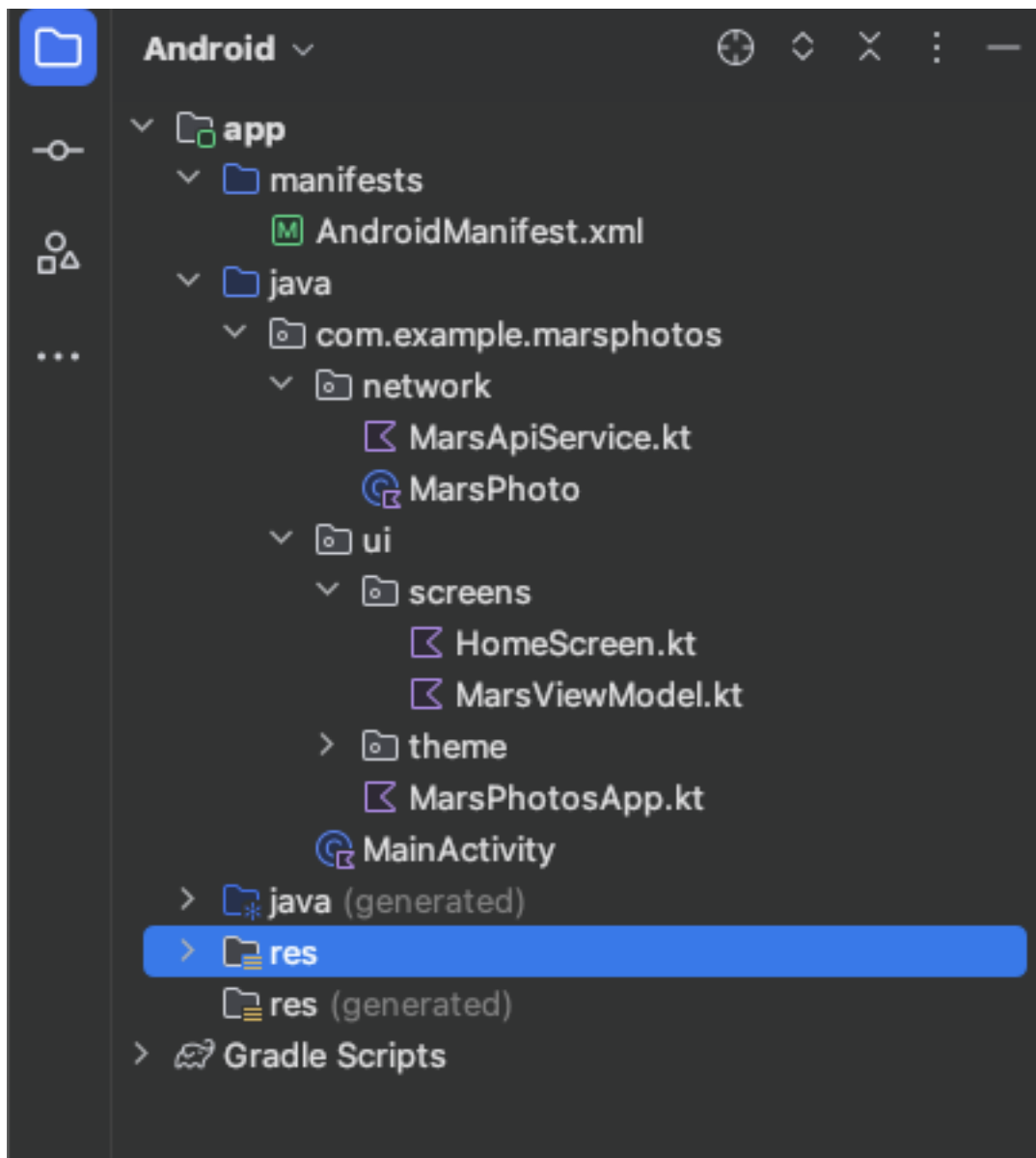


Imagen 1: Arquitectura del proyecto.

Sobre Retrofit



El uso de Retrofit

- Retrofit es una biblioteca destinada a facilitar la recuperación y el envío de datos a través de RESTful webservice.
- Retrofit es simple de usar, de ahí su uso extendido en Android, con poco código, es fácil crear una interfaz que envía solicitudes GET a un servidor, y además permite manejar las respuestas.
- Retrofit impone la verificación de tipo en tiempo de compilación, es decir, es tipo-safe, lo que ayuda a evitar posibles errores en tiempo de ejecución.
- Retrofit es además tremendamente eficiente, las solicitudes y respuestas, incluyendo la serialización y deserialización de cuerpos de respuesta, lo que al final se traduce en mejor rendimiento y menos tiempos de espera.
- Por último, el punto fuerte de Retrofit, su buena integración con otras librerías Android, como Gson, para la serialización y deserialización de archivos JSON.
- Todo esto convierte a Retrofit en una herramienta muy poderosa para nuestras aplicaciones Android, su simpleza, así como seguridad de tipo y eficiencia, lo convierte en una excelente opción para el desarrollo móvil.

Durante este trabajo he creado una capa para el servicio de red que se comunica con un servidor que recupera los datos que le he solicitado gracias al codelab. La herramienta que me ha permitido realizar el proyecto ha sido **Retrofit**. El ViewModel es el encargado de comunicarse con la capa de datos. En MarsViewModel, se ha llevado a cabo la llamada a la red para obtener los datos del proyecto, el MutableState, como mencioné antes, se encarga de que cada vez que cambie, se vuelva a dibujar la vista.

Durante el codelab he podido aprender que la mayoría de los servidores web se ejecutan con una arquitectura conocida como REST, Representational State Transfer (transferencia de estado representacional) a los servicios web que ofrecen esta arquitectura se les conoce como servicios RESTful.

Las solicitudes a los servicios RESTful, se hacen a través de URI, identificadores uniformes de recursos. EL URI se dedica a identificar recursos en el servidor por su nombre. Durante la realización del proyecto, he recuperado las URLs de las imágenes con el URI del servidor android-kotlin-fun-mars-server.appspot.com

La URL no deja de ser un subconjunto de URI que especifica la ubicación de un recurso y el mecanismo a través del que se recupera.

Solicitar, solicitar, solicitar..

Cada solicitud de servicio web contiene un URI y se transferirá a un servidor mediante el mismo protocolo HTTP.

Entre las solicitudes comunes de HTTP se incluyen:

- **GET:** para recuperar datos de un servidor
- **POST:** para crear datos nuevos en el servidor
- **PUT:** para actualizar datos en el servidor
- **DELETE:** para borrar los datos del servidor.

A través del GET al servidor, se obtiene la información sobre las fotos, el servidor luego muestra una respuesta, que son las URL. La respuesta del servidor se obtiene mediante un fichero JSON, el cual tiene un id para cada foto, y un valor, entrecomillado, que es la URL de la foto.

Durante el proyecto, Retrofit se comunica con el backend de REST, que se dedica a generar el código, mientras que yo solo proporciono los URIs para el servicio web.

Las dependencias:

Para la realización del proyecto he tenido que insertar las siguientes dependencias con el fin de poder usar Retrofit sin problemas. Además, también ha sido necesario implementar una librería de kotlin pensada para deserializar archivos JSON.

```
// Retrofit
implementation("com.squareup.retrofit2:retrofit:2.9.0")

// Retrofit with Kotlin serialization Converter
implementation("com.jakewharton.retrofit:retrofit2-kotlinx-serialization-converter:1.0.0")
implementation("com.squareup.okhttp3:okhttp:4.11.0")

// Kotlin serialization
implementation("org.jetbrains.kotlin:kotlinx-serialization-json:1.5.1")

debugImplementation("androidx.compose.ui:ui-test-manifest")
debugImplementation("androidx.compose.ui:ui-tooling")
```

Imagen 2. Dependencias en Gradle del proyecto.

El proyecto explicado

```
8
9 private const val BASE_URL =
10     "https://android-kotlin-fun-mars-server.appspot.com"
11
12
13
14 private val retrofit = Retrofit.Builder() Retrofit.Builder
15     .addConverterFactory(Json.asConverterFactory("application/json".toMediaType())) Retrofit.Builder
16     .baseUrl(BASE_URL)
17     .build()
18
19 ± Agustrodmar
20 interface MarsApiService {
21     ± Agustrodmar
22     @GET("photos")
23     suspend fun getPhotos(): List<MarsPhoto>
24 }
25
26 ± Agustrodmar
27 object MarsApi {
28     val retrofitService : MarsApiService by lazy {
29         val retrofit = Retrofit.Builder() Retrofit.Builder
30             .addConverterFactory(Json.asConverterFactory("application/json".toMediaType())) Retrofit.Builder
31             .baseUrl(BASE_URL)
32             .build()
33         retrofit.create(MarsApiService::class.java) ^lazy
34     }
35 }
36 }
```

Imagen 3. MarsApiService.kt.

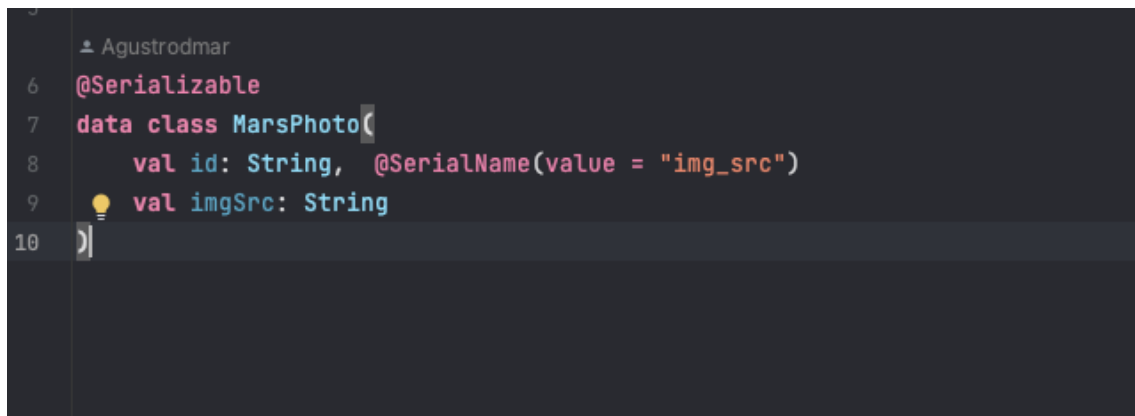
En primer lugar, podemos encontrar dentro del paquete network, el archivo kotlin “MarsApiService”, aquí declaro una constante que contiene la URL de la web a la que me voy a conectar.

Seguidamente declaro un objeto Retrofit, al que llamo retrofit. Este se configura con la URL base y un ConverterFactory, que sirve para convertir JSON, con esto se consigue que Retrofit convierta automáticamente las respuestas JSON en objetos de Kotlin.

La interfaz MarsApiService, una interfaz donde se encuentra la función getPhotos(). Le añado la solicitud @GET(“photos”), que hace que cuando se llame a la función, Retrofit haga una solicitud GET a la ruta “photos” de la web.

MarsApi, es un object. Este tiene la propiedad retrofitService, una instancia Lazy de MarsApiService, gracias a que es lazy, no se crea hasta que no se accede por primera vez.

MarsPhoto

A screenshot of a code editor showing the definition of the MarsPhoto data class. The code is in Kotlin and includes the @Serializable annotation. The class has two properties: id and imgSrc. The id property is annotated with @SerializedName(value = "img_src"). The code is as follows:

```
5  @Serializable
6
7  data class MarsPhoto(
8      val id: String, @SerializedName(value = "img_src")
9      val imgSrc: String
10 )
```

Imagen 4. Data class MarsPhoto.

El siguiente paso es la data class MarsPhoto, serializable, que es la anotación que sirve para indicar que los objetos de la clase pueden ser serializados y deserializados desde y hacia formatos de datos como JSON.

Por lo demás MarsPhoto es simplemente una clase de datos con propiedades como “id” e “imageSrc” para las representaciones de las imágenes a obtener, correspondiendo con el esquema clave:“valor” de los archivos JSON.

HomeScreen

Dejando la carpeta network, accedo al paquete en el que se incluye la interfaz, ui, dentro de la carpeta screens, se encuentra HomeScreen:

```
@Composable
fun HomeScreen(
    marsUiState: MarsUiState,
    modifier: Modifier = Modifier
) {
    when (marsUiState) {
        is MarsUiState.Loading -> LoadingScreen(modifier = modifier.fillMaxSize())
        is MarsUiState.Success -> ResultScreen(
            marsUiState.photos, modifier = modifier.fillMaxWidth()
        )

        is MarsUiState.Error -> ErrorScreen(modifier = modifier.fillMaxSize())
    }
}

+ Agustrodmar
@Composable
fun LoadingScreen(modifier: Modifier = Modifier) {
    Image(
        modifier = modifier.size(200.dp),
        painter = painterResource(R.drawable.loading_img),
        contentDescription = stringResource("Loading")
    )
}

+ Agustrodmar
@Composable
fun ErrorScreen(modifier: Modifier = Modifier) {
    Column(
        modifier = modifier,
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) { this: ColumnScope
        Image(
            painter = painterResource(id = R.drawable.ic_connection_error), contentDescription = "
        )
        Text(text = stringResource("Failed to load"), modifier = Modifier.padding(16.dp))
    }
}

/**
 * ResultScreen displaying number of photos retrieved.
 */
+ Agustrodmar
@Composable
fun ResultScreen(photos: String, modifier: Modifier = Modifier) {
    Box(
        contentAlignment = Alignment.Center,
        modifier = modifier
    ) { this: BoxScope
        Text(text = photos)
    }
}
```

Imagen 5. HomeScreen.

Incluye las principales composables del proyecto, HomeScreen, LoadingScreen, ErrorScreen y ResultsScreen. Durante el proyecto he trabajado principalmente en HomeScreen, así como LoadingScreen y ErrorScreen

HomeScreen toma una instancia MarsUiState y un Modifier como parámetros, dependiendo del estado de MarsUiState, mostrará las diferentes pantallas, que corresponden con las siguientes composables.

MarsUiState

```
/**
 * A sealed interface representing the state of the Mars UI.
 */
@Agustrodmar
sealed interface MarsUiState {
    /**
     * Represents a successful state with the retrieved Mars photos.
     *
     * @property photos The retrieved Mars photos.
     */
    @Agustrodmar
    data class Success(val photos: String) : MarsUiState

    /**
     * Represents an error state.
     */
    @Agustrodmar
    object Error : MarsUiState

    /**
     * Represents a loading state.
     */
    @Agustrodmar
    object Loading : MarsUiState
}
```

Imagen 6. Interfaz sellada MarsUiState.

MarsUiState no deja de ser una interfaz sellada que define los tres estados posibles de la interfaz del usuario, éxito al conectar, error al conectar (un modo avión activado, por ejemplo), y un cargando, para el tiempo en el que se tarde en conectar. Representan a cada estado diferente en la solicitud.

MarsViewModel

```
32
33 class MarsViewModel : ViewModel() {
34     /** The mutable State that stores the status of the most recent request */
35     var marsUiState: MarsUiState by mutableStateOf(MarsUiState.Loading)
36     private set
37
38
39     /**
40      * Call getMarsPhotos() on init so we can display status immediately!.
41      */
42     init {
43         getMarsPhotos()
44     }
45
46     /**
47      * Gets Mars photos information from the Mars API Retrofit service and updates the
48      * [MarsPhoto] [List] [MutableList].
49      */
50     private fun getMarsPhotos() {
51         viewModelScope.launch { this: CoroutineScope
52             try {
53                 val listResult = MarsApi.retrofitService.getPhotos()
54                 marsUiState = MarsUiState.Success(
55                     photos: "Success: ${listResult.size} Mars photos retrieved"
56                 )
57             } catch (e: IOException) {
58                 "Error"
59             }
60         }
61     }
62 }
63
```

Imagen 7. MarsViewModel.

El ViewModel no tiene mucha complejidad en comparación al de proyectos anteriores. Como siempre es una clase, que contiene un estado mutable marsUiState, y almacena el estado de la solicitud más reciente. Al comenzar, llama a getMarsPhotos.

MarsPhotoApp

```
48 @Composable
49 fun MarsPhotosApp() {
50     val scrollBehavior = TopAppBarDefaults.enterAlwaysScrollBehavior()
51     Scaffold(
52         modifier = Modifier.nestedScroll(scrollBehavior.nestedScrollConnection),
53         topBar = { MarsTopAppBar(scrollBehavior = scrollBehavior) }
54     ) { it: PaddingValues
55         Surface(
56             modifier = Modifier
57                 .fillMaxSize()
58                 .padding(it)
59         ) {
60             val marsViewModel: MarsViewModel = viewModel()
61             HomeScreen(
62                 marsUiState = marsViewModel.marsUiState
63             )
64         }
65     }
66 }
67
68 ± Agustrodmar
69 @Composable
70 fun MarsTopAppBar(scrollBehavior: TopAppBarScrollBehavior, modifier: Modifier = Modifier) {
71     CenterAlignedTopAppBar(
72         scrollBehavior = scrollBehavior,
73         title = {
74             Text(
75                 text = stringResource("Mars Photos"),
76                 style = MaterialTheme.typography.headlineSmall,
77             )
78         },
79         modifier = modifier
80     )
81 }
```

Imagen 8. MarsPhotoApp

En MarsPhotoApp.kt incluyo las composables que muestran el contenido visualizable de la app. No he tenido que trabajar sobre este archivo, ya que por defecto viene así en el codelab.

La composable MarsPhotoApp crea la UI principal de la app.

MarsTopAppBar es la composable que muestra una barra superior personalizada de la aplicación, y alinea el título en el centro.

AndroidManifest

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Mars Photos"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/Theme.MarsPhotos"
        tools:targetApi="33">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="Mars Photos"
            android:theme="@style/Theme.MarsPhotos">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

En el AndroidManifest he tenido que añadir la siguiente sentencia:

```
<uses-permission android:name="android.permission.INTERNET" />
```

El haber hecho esto, le da capacidad a la app de solicitar permiso al sistema Android para conectarse a internet. Es un requisito indispensable. Hasta ahora no había desarrollado ninguna app con esta funcionalidad, por lo que me ha parecido interesante aprender a como conectar una app a internet.

Un vistazo final

El haberme guiado en el uso de Retrofit a través del codelab me ha parecido una decisión acertada. No hay muchos recursos en la web sobre cómo usar la librería, y esta me ha parecido una forma genial de implementarla. Es diseño sencillo, siguiendo el patrón MVVM, con el propósito de obtener datos de un JSON, el cual se serializa y deserializa.

Bibliografía

- Programación y más – *Android: Cómo consumir una API y procesar respuestas JSON. Retrofit*. Última consulta 27/01/2023.
- Android para desarrolladores (21 de septiembre de 2023) – *Cómo obtener datos de Internet*. Última consulta 27/01/2023.
- Retrofit – *Retrofit*. Última consulta 27/01/2023.