



Programación multimedia y dispositivos móviles

UT-4. Servicios API Rest. Recycler

<https://developer.android.com/guide/>
<https://github.com/stars/JulioHornos/lists/móviles-publico>



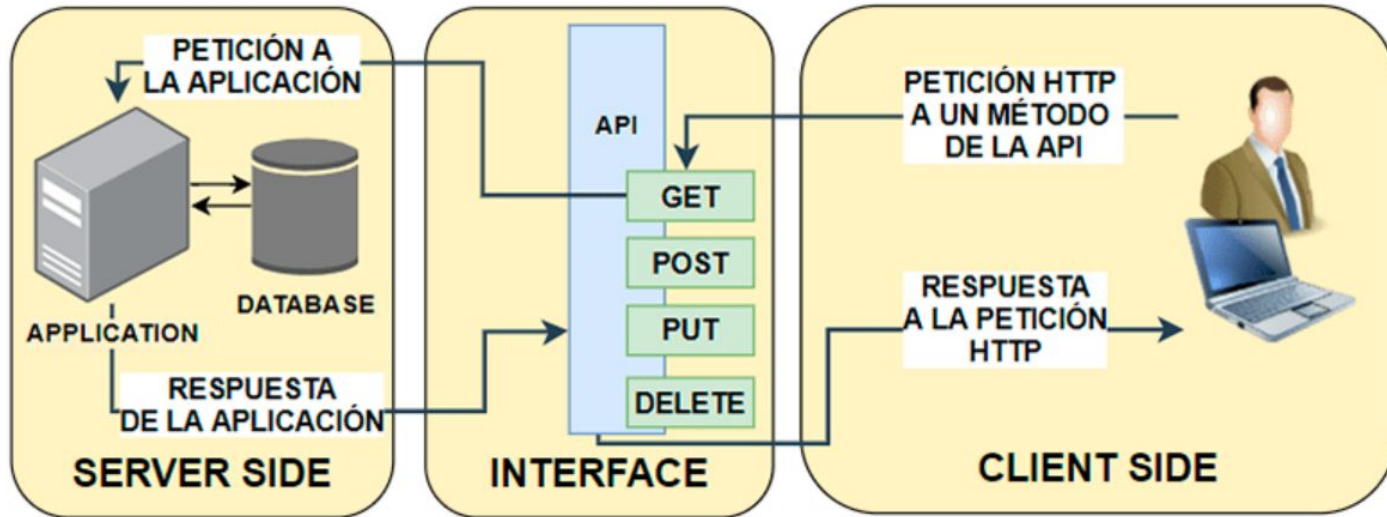
Acceso a Servicios Web REST en Android

Llegados a este punto, tenemos claro que las aplicaciones modernas se organizan en varias capa, independizando la presentación, con su propia lógica, de la capa con lógica de la negocio y la capa de acceso a datos.

Es muy común que debajo de los diferentes canales por los que los usuarios acceden a una aplicación (web apps y aplicaciones móviles sobre todo) exista una sola base de datos y servicios de backend compartidos.

Esto generalmente se hace exponiendo los servicios de negocio o acceso a datos en un API. Una API no es más que un intermediario entre una base de datos y una aplicación móvil (sea Android, iOS u otra tecnología).

Acceso a Servicios Web REST en Android



Ejemplo de petición a una API REST.



Acceso a Servicios Web REST en Android

Las APIs REST son interfaces para compartir recursos entre clientes y servidor. Mediante solicitudes HTTP podemos lograr que todos hablen el mismo idioma, sin importar el sistema operativo, plataforma o el lenguaje de programación de cada uno.

Eso sí, normalmente incorporan medidas de seguridad adicionales tales como autenticación OAuth o similares.

En nuestro caso vamos a hacer una aplicación muy sencilla que busca imágenes de perros. Tendremos un buscador y un RecyclerView. El usuario buscará una raza de perro y en ese momento haremos una petición REST que nos devolverá un array de imágenes de dicha raza y actualizaremos el RecyclerView.



Acceso a Servicios Web REST en Android

Utilizaremos un API pública llamada Dog API que podemos encontrar aquí: <https://dog.ceo/dog-api/>

Las APIS públicas no suelen incorporar medidas de autenticación, aunque muchas de ellas sólo incorporan métodos GET para recuperar información, como es este caso.

Vamos a consumir un servicio GET, al cual le pasaremos el nombre de la raza y nos devolverá un Json similar a este.

```
{  
  "status": "success",  
  "message": [  
    "https://images.dog.ceo/breeds/akita/512px-Ainu-Dog.jpg",  
    "https://images.dog.ceo/breeds/akita/512px-Akita_inu.jpeg",  
    "https://images.dog.ceo/breeds/akita/Akina_Inu_in_Riga_1.JPG",  
    "https://images.dog.ceo/breeds/akita/Akita_Dog.jpg",  
    "https://images.dog.ceo/breeds/akita/Akita_Inu_dog.jpg",  
    "https://images.dog.ceo/breeds/akita/Akita_inu_blanc.jpg",  
    "https://images.dog.ceo/breeds/akita/An_Akita_Inu_resting.jpg",  
    "https://images.dog.ceo/breeds/akita/Japaneseakita.jpg"  
  ]  
}
```



Acceso a Servicios Web REST en Android - Internet

Este mensaje de vuelta contiene dos atributos:

- Status: Se trata de una String que nos devolverá success o error dependiendo de si encuentra la raza o no.
- Message: Una array de String que contiene todas las imágenes que mostraremos.

Hacer uso de una REST API, obviamente, implica usar Internet. Las apps en Android que quieren acceder a Internet necesitar tener permiso.

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

Debido a que android.permission.INTERNET no es considerado un permiso peligroso, no tenemos que solicitarlo durante el tiempo de ejecución, pero si añadirlo en el manifiesto (justo antes del nodo de aplicación, hija directa del nodo manifest)



Acceso a Servicios Web REST en Android - Hilos

En versiones posteriores a la 3.0 (a nuestros efectos todas), Android no permite realizar operaciones de larga duración directamente en el hilo principal de la aplicación.

Para solucionar esto será necesario utilizar funciones de suspensión y corrutinas para el consumo del API.

REST se asienta sobre el protocolo HTTP como mecanismo de transporte entre cliente y servidor, y Android ofrece clases para trabajar con las comunicaciones en este protocolo. Sin embargo en “casi” toda la documentación que he encontrado, incluida la documentación oficial, he visto que utilizaban librerías de terceros. Vamos a ver qué librerías vamos a necesitar.

<https://developer.android.com/codelabs/basic-android-kotlin-compose-getting-data-internet?hl=es-419#0>



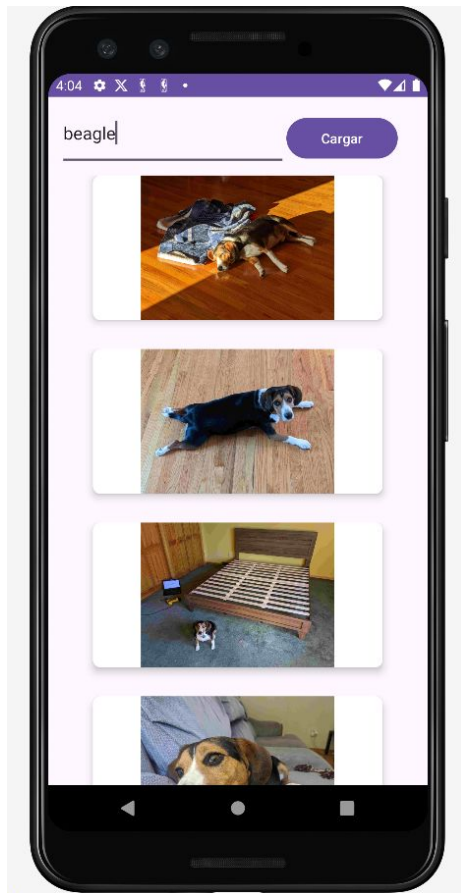
Acceso a Servicios Web REST en Android - librerías

- **Retrofit:** es un cliente REST simple de configurar. Nos permitirá tratar las llamadas a la API como funciones, así definiremos solamente las URLs que queremos llamar y los tipos de petición y respuesta que esperamos.
- **Gson:** es una librería de Google muy popular para serializar y de-serializar data entre objetos JSON y objetos Java. Así podremos tratar a los datos JSON como clases de Java.
- **Glide:** es una popular librería Android de código abierto para cargar imágenes, videos y GIFs animados. Con Glide puedes cargar y mostrar medios de muchas fuentes diferentes, tales como servidores remotos o el sistema local de archivos.

MyAPIDOG - Paso 1: Configuración inicial

El primer paso es añadir los permisos en el manifiesto y añadir las librerías en el fichero build.gradle, en la parte de dependencias (justo después de añadirlas, debemos sincronizarlas):

```
implementation ("com.squareup.retrofit2:retrofit:2.9.0" )  
  
implementation ("com.squareup.retrofit2:converter-gson:2.9.0" )  
  
implementation ("com.github.bumptech.glide:glide:4.16.0" )  
  
implementation  
("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.9" )  
  
implementation ("androidx.lifecycle:lifecycle-runtime-ktx:2.8.7" )
```



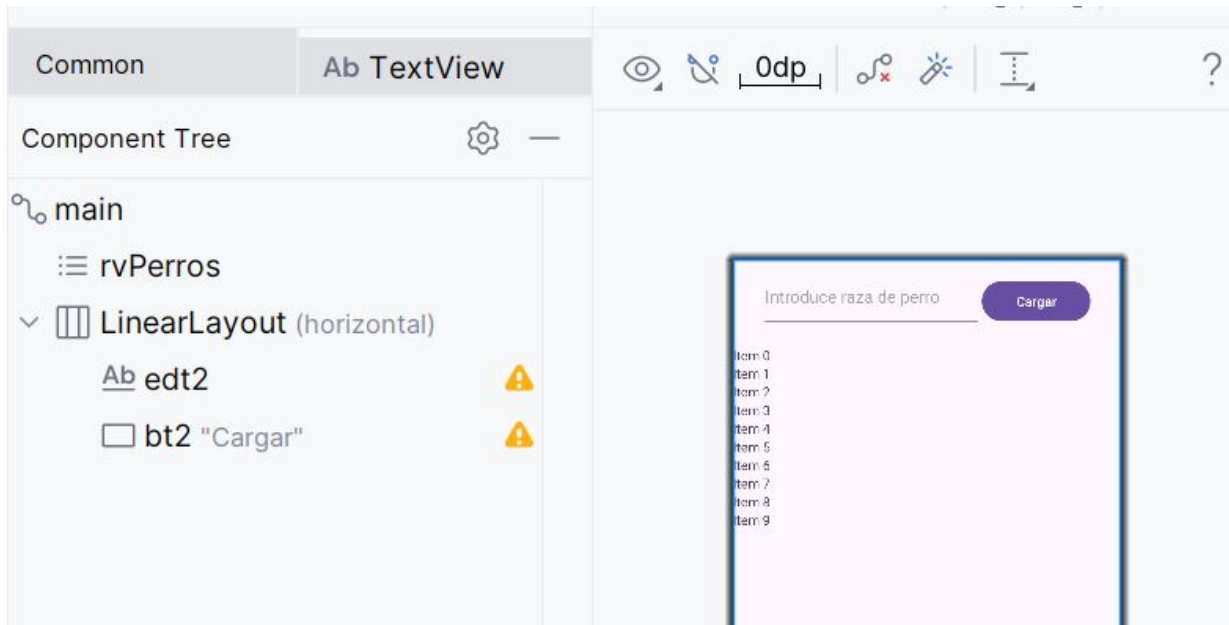


MyAPIDOG - Paso 2: Diseño de Layout

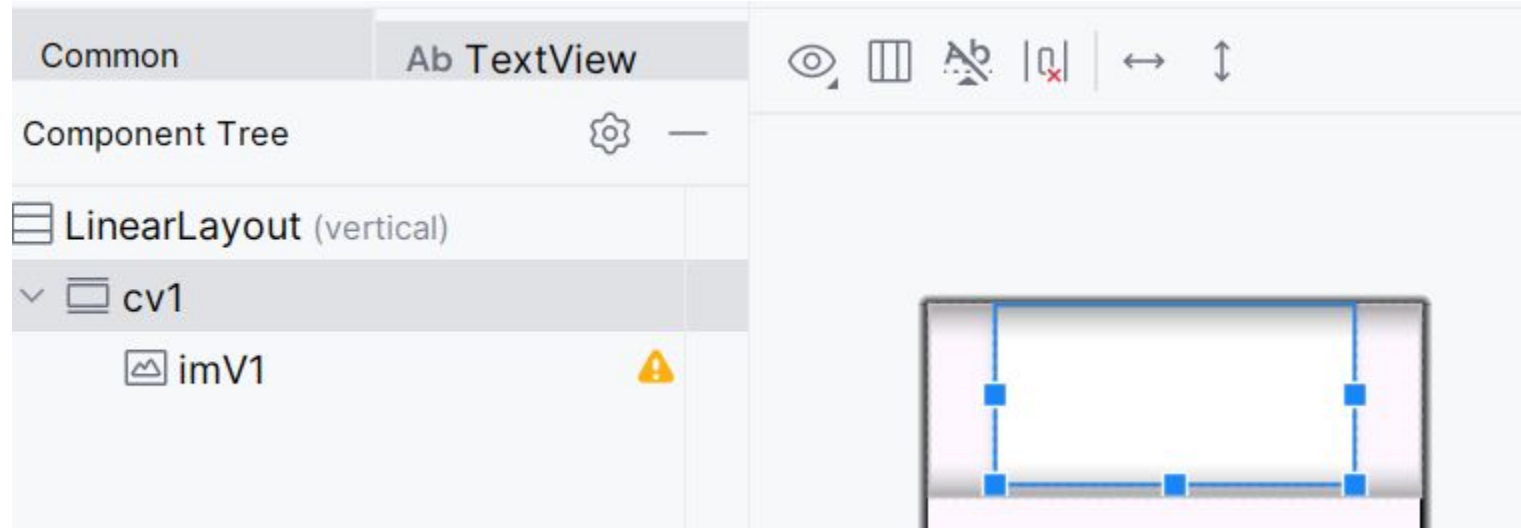
Vamos a pintar la interfaz de usuario. Yo he hecho esto, pero sentiros libres si queréis de hacer otra cosa:

- Un `LinearLayout` horizontal de 350dp de ancho x 80dp de alto. Dentro tengo
 - Un `EditText` que admita caracteres
 - Un botón
- Un `RecyclerView` match parent de ancho x de 640 de alto.
 - En el `RecyclerView` diseñamos la fila (row) con un `LinearLayout` de 180 de alto y match parent de ancho.
 - Dentro habrá un `CardView` de 300dp de ancho x 150dp de alto y dentro del `CardView` ponemos únicamente un `ImageView` con match parent x match parent.

MyAPIDOG - Paso 2: Diseño de Layout



MyAPIDOG - Paso 2: Diseño de Layout





MyAPIDOG - Paso 3: Definir endpoints y modelos

Como siguiente paso debemos crear una clase de datos que va representar cada uno de los datos que vamos a estar recibiendo por medio de la API y vamos a estar consumiendo. Es decir una clase que permita a Gson extraer los valores del Json y crear un objeto de la clase con dichos valores.

Se tratará de una data class llamada DogRespuesta y contendrá dos campos, los mismos que nos devolvía la respuesta de la petición Y CON EL MISMO NOMBRE.

Os dejo una página web en la que tu metes el Json de respuesta y te dice los atributos que debes definir en tu clase para poder tratar el json: <https://www.jsonschema2pojo.org/>

MyAPIDOG - Paso 3: Definir endpoints y modelos

 DogRespuesta.kt ×

 MainState.kt

 MainActivity.kt

 MainViewModel.kt

```
1 package com.example.myapidogkotlin.model
2
3 class DogRespuesta(val status: String, val message: List<String>?) {
4 }
```



MyAPIDOG - Paso 3: Definir endpoints y modelos

Crea una nueva Interfaz llamada **DogsAPIService**. En esta interfaz vamos a definir métodos abstractos. Cada método abstracto va a representar una ruta específica de nuestra API.

Por ejemplo, podemos tener un método para realizar un inicio de sesión. Le pasamos un usuario y una contraseña y obtenemos un token como respuesta.

Podríamos tener métodos post, put o delete para actualizar información.

En este caso sólo tenemos el método de buscar perros cuya respuesta se almacena en el objeto DogRespuesta creado (lo habitual será que las respuestas se capturen en un objeto o en un array de objetos, depende de cómo sea el la respuesta del API).

Cada método que implique una futura salida a internet para recuperar datos debe ser una función suspendida.

MyAPIDOG - Paso 3: Definir endpoints y modelos



The screenshot shows an IDE with four tabs: DogRespuesta.kt, DogAPIService.kt (active), MainState.kt, and MainAc. The active file, DogAPIService.kt, contains the following Kotlin code:

```
1 package com.example.myapidogkotlin.model
2
3 import retrofit2.Response
4 import retrofit2.http.GET
5
6 interface DogAPIService {
7
8     @GET("images")
9     suspend fun getFotosPerros(): Response<DogRespuesta>
10 }
```




MyAPIDOG - Paso 4: Consumiendo el API Rest

Como ya hemos comentado, Android no te deja que los proceso pesados, este en realidad no lo es, pero cuando invocas un servicio web nunca estás seguro, se deben ejecutar fuera del hilo principal. Android NO te deja ejecutarlo dentro del hilo principal.

Vamos a implementar la llamada siguiendo el patrón MVVM. La llamada la incluiremos en el MainState.

La llamada a APIs con Retrofit está “industrializada” y “simplemente” necesitamos saber la URL que debemos montar para hacer la llama. El resto de pasos, sota, caballo y rey.

En nuestro caso, y viendo la documentación del API, en la url debemos incluir la raza a buscar y luego “images”.

MyAPIDOG - Paso 4: Consumiendo el API Rest

```
✓ class MainState {  
    var cadena = "https://dog.ceo/api/breed/"  
  
    ✓ suspend fun recuperaFotos(raza: String): DogRespuesta {  
        val cadenaFinal = cadena +raza+ "/"  
        val retrofit = Retrofit.Builder()  
            .baseUrl(cadenaFinal)  
            .addConverterFactory(GsonConverterFactory.create())  
            .build()  
  
        val call = retrofit.create(DogAPIService::class.java).getFotosPerros()  
        val fotosPerros = call.body()  
        ✓ if(fotosPerros!=null){  
            return fotosPerros  
        }else{  
            return DogRespuesta( status: "no success", message: null )  
        }  
    }  
}
```

Montamos la cadena con la url de llamada

Montamos la llamada con Retrofit

Hacemos la llamada y recuperamos los datos. Se infiere el objeto de respuesta



MyAPIDOG - Paso 4: Consumiendo el API Rest

El ViewModel es , como de costumbre, un “puente” donde lo más importante es que se utilizan corrutinas para desacoplar la llamada

```
class MainViewModel : ViewModel() {

    val myEstado = MainState()
    private val _datos = MutableLiveData<DogRespuesta>(DogRespuesta(null.toString(), ArrayList()))
    val datos: LiveData<DogRespuesta> get() = _datos

    fun devuelveFotos(raza: String){
        viewModelScope.launch {
            _datos.value = myEstado.recuperaFotos(raza)
        }
    }
}
```



MyAPIDOG - Paso 5: Objetos View y Adaptador

En el View como siempre, casteamos aquellos controles sobre los que necesitemos trabajar. En este caso sólo necesitamos el imageView para poner las fotos de perros

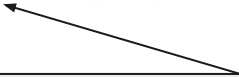
```
class MyView(itemView: View) : RecyclerView.ViewHolder(itemView) {  
    val imV1 = itemView.findViewById<View>(R.id.imV1) as ImageView  
}
```



MyAPIDOG - Paso 5: Objetos View y Adaptador

A estas alturas ya debemos saber hacer un Recycler y los objetos necesarios. Simplemente apuntar que, como hemos visto antes, hemos dejado el objeto `dogRespuesta` preparado para ser utilizado en el adaptador por lo que el constructor debe recibirlo:

```
class MyAdapter (private val dataSet: DogRespuesta) : RecyclerView.Adapter<MyView>() {  
  
    lateinit var myContexto : Context
```



Glide necesita el contexto así que me defino un atributo de clase para tenerlo

MyAPIDOG - Paso 5: Objetos View y Adaptador

```
// Create new views (invoked by the layout manager)
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyView {
    // Create a new view, which defines the UI of the list item
    myContexto = parent.context
    val view = LayoutInflater.from(parent.context)
        .inflate(R.layout.row, parent, attachToRoot: false)

    return MyView(view)
}
```

Asignamos el contexto para poder utilizarlo luego

```
// Return the size of your dataset (invoked by the layout manager)
override fun getItemCount() = dataSet.message!!.size
```

Recordemos que message es en realidad una lista de strings con las urls de las fotos

MyAPIDOG - Paso 5: Objetos View y Adaptador

```
// Replace the contents of a view (invoked by the layout manager)
override fun onBindViewHolder(holder: MyView, position: Int) {
    // Get element from your dataset at this position and replace the
    // contents of the view with that element
    val url : String = dataSet.message!![position]
    Glide.with(myContexto)
        .load(url)
        .into(holder.imV1);
}
```

Usamos Glide para pintar la imagen de la url que nos han dado dentro del imageview



MyAPIDOG - Paso 6: La Actividad

Pues como último punto de “montaje” en la actividad debemos llamar al Viewmodel, pasando la raza escrita, y nos suscribimos a los cambios en el objeto para “pintar” resultados (todo ello después de haber vinculado vistas)

```
// Asignamos la forma en que queremos ver el Recycler
val mLayout = LinearLayoutManager(context: this@MainActivity)
rvPerros.layoutManager = mLayout

bt2.setOnClickListener{
    val raza = edt2.getText().toString();
    if (raza.isEmpty()){
        Toast.makeText(applicationContext, text: "Debe seleccionar una raza", Toast.LENGTH_SHORT)
            .show();
        return@setOnClickListener;
    }
    myViewModel.devuelveFotos(raza)
}
```




MyAPIDOG - Paso 6: La Actividad

```
myViewModel.datos.observe( owner: this@MainActivity){  
    if(it.status=="success"){  
        myAdapter = MyAdapter (it)  
        //Y el adaptador se lo asignamos al recycler.  
        rvPerros.adapter = myAdapter  
    } else {  
        Toast.makeText(applicationContext, text: "No hay fotos de esa raza", Toast.LENGTH_SHORT)  
            .show()  
    }  
}
```



Otras cosas

Cómo registrar un icono de los que vienen prediseñados en la app para poder utilizarlo:

<https://stackoverflow.com/questions/47338053/using-android-studio-default-search-icon-for-search-view>

Un listado de 100 APIS públicas que ofrecen contenido diverso. Seguro que hay muuuuchas más

<https://ichi.pro/es/una-lista-seleccionada-de-100-api-publicas-geniales-y-divertidas-para-inspirar-su-proximo-proyecto-8109114517939>



Otras cosas

O esta con toooooodo el mundo pokemon:

https://pokeapi.co/?utm_source=Direct%20-%2018831&source=Blog

Mundo Marvel: <https://developer.marvel.com/>

Rick&Morty: <https://rickandmortyapi.com/>

Un API de la NASA: <https://api.nasa.gov/>

NBA: <https://www.balldontlie.io/#get-all-players>

Datos meteorológicos, cambios de divisa, CryptoMonedas, Datos de Covid, memes de Chuck Norris, ..



Documentación

<https://code.tutsplus.com/es/tutorials/android-from-scratch-using-rest-apis--cms-27117>

<https://coderslink.com/talento/blog/como-consumir-una-api-desde-una-aplicacion-android/>

<https://programmerclick.com/article/74311415781/>

<https://programacionymas.com/blog/consumir-una-api-usando-retrofit>

<https://ed.team/blog/como-consumir-una-api-rest-desde-android>

<https://keepcoding.io/blog/retrofit-en-kotlin-para-consumir-apis/>

