Servicios REST en Android

Consumición imágenes Web

Picasso

Es una librería que a partir de una URL permite consumir una imagen alojada en un servidor. Las descargas de las imágenes las ejecuta en un Thread secundario, por lo que no provocará problemas.

Más información:

- https://square.github.io/picasso/
- https://github.com/square/picasso

¿Cómo agregar Picasso al proyecto?

Se debe acceder al archivo gradle de la aplicación, y agregar la siguiente línea dentro de las dependencias:

```
implementation 'com.squareup.picasso:picasso:X.X'
```

Nota: se debe reemplazar las "X" con la última versión obtenida a partir del siguiente link.

Luego se deberá agregar el permiso de Internet en el AndroidManifest si no se hizo:

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

¿Cómo utilizar Picasso?

Se debe obtener la instancia de Picasso, ejecutar el método *load()* e ingresar como parámetro la URL de la imagen. Por último ejecutar el método *into()* en donde se le pasará por parámetro el objeto de la ImageView donde se mostrará.

Ejemplo:

```
Picasso
    .get()
    .load( path: "http://i.imgur.com/DvpvklR.png")
    .into(imgWeb)
```

También se puede activar el *logging* para ver en el Logcat el detalle de lo que ocurre con Picasso:

```
Picasso.get().isLoggingEnabled = true
```

Nota: si se prueba en un dispositivo Android mayor o igual a 9, se debe agregar la siguiente línea dentro del AndroidManifest.xml:

```
android:usesCleartextTraffic="true"
```

Para más información:

- CleartextTrafficPermitted

Realizar laboratorio 3.

Conversor de JSON

¿Para qué sirve?

Un conversor de JSON permite crear objetos a partir de cadenas de texto con el formato JSON. A su vez permite revertir esta conversión, pudiendo así, enviar y recibir información mediante distintos tipos de Requests. En Android el conversor que se destaca y es el más rápido es Gson.

¿Cómo agregar Gson al proyecto?

Se debe acceder al archivo gradle de la aplicación, y agregar la siguientes líneas dentro de las dependencias:

```
implementation"com.google.code.gson:gson:2.8.8"
```

Nota: se debe reemplazar las "X" con la última versión obtenida a partir del siguiente repositorio.

¿Cómo utilizar Gson?

Una vez agregada las dependencia, se debe crear (o utilizar) una clase del tipo DTO (data transfer object), conocida por tener un constructor, getters y setters. Siguiendo el ejemplo de los JSON de personas, la clase a crear sería "Persona". Si los nombres de las propiedades de la clase no coinciden con el nombre clave del JSON, se le puede agregar otra annotation indicándole cómo vendrá. Ejemplo:

```
data class Persona(
    var nombre: String,
    @SerializedName( value: "apellido")
    var ape: String,
    var edad: Int
)
```

Consumición servicios RESTful

Retrofit

Es la librería más utilizada en Android para la consumición de servicios RESTful. Utiliza anotaciones para describir los llamados HTTP.

¿Cómo agregar Retrofit al proyecto?

Se debe acceder al archivo gradle de la aplicación, y agregar las siguientes líneas dentro de las dependencias:

```
implementation 'com.squareup.retrofit2:retrofit:X.X.X'
implementation "com.squareup.retrofit2:converter-gson:X.X.X"
```

Nota: se debe reemplazar las "X" con la última versión obtenida a partir del siguiente link.
Si se utiliza Corrutinas como solución para la ejecución de tareas asincrónicas se deberá agregar también la siguiente dependencia:

```
implementation "com.jakewharton.retrofit:retrofit2-kotlin-coroutines-adapter:Y.Y.Y"
```

La versión se puede obtener a partir de este link.

¿Cómo utilizar Retrofit?

Una vez agregadas las dependencias, se deben crear varias clases:

- Una clase que tendrá la instancia del cliente Retrofit apuntando hacia una URL.
- Una interfaz que va a contener los métodos HTTP a ejecutar con la utilización de annotation.

Luego se deberá agregar el permiso de Internet en el AndroidManifest si no se hizo:

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

Retrofit utiliza una clase llamada "Call", la cual es la encargada de enviar peticiones al servicio web, y devolver una respuesta. Para llamar al servicio se utiliza el método "execute" (de manera síncrona) ó "enqueue" (de manera asíncrona). También posee el método "cancel" para abortar la llamada.

Ejemplo de la instancia de Retrofit: se le agrega la URL base hacia la que apunta y el conversor interno que utilizará, que en este caso es Gson. Esta configuración permitirá la conversión automática del json obtenido o enviado al objeto definido. A su vez se definió un *CallAdapter* personalizado que es el de las Corrutinas. Esto permitirá que las llamadas a los servicios se puedan manejar con dicha solución.
Para el archivo del cliente de Retrofit se recomienda utilizar un object para que la instancia se maneje de manera estática y no se esté creando cada vez que se tenga que utilizar.

```
object RetrofitClient {

    private const val BASE_URL = "https://demo3691370.mockable.io/"

    private val retrofit: Retrofit = Retrofit.Builder()
        .baseUrl(BASE_URL)
        .addConverterFactory(GsonConverterFactory.create())
        .addCallAdapterFactory(CoroutineCallAdapterFactory())
        .build()

    val librosApi: LibrosApi = retrofit.create(LibrosApi::class.java)
}
```

Ejemplo de la interfaz que contiene los llamados a los servicios web.

```
interface LibrosApi {
    @GET( value: "librosFavoritos")
    suspend fun getLibrosFavoritos(): List<LibroResponse>
}
```

Para realizar la petición del servicio se accederá a la instancia *librosApi* creada en *RetrofitClient* y luego se llamará al método correspondiente, que en este caso es *getLibrosFavoritos*.
Luego, se debe llamar al método de la Api
Ejemplo GET:

```
lifecycleScope.launch(Dispatchers.IO) {  this: CoroutineScope
    val libros = RetrofitClient.librosApi.getLibrosFavoritos()
}
```

Recordar manejar los errores a través de la estrategia seleccionada y moverse entre hilos si se desea comunicar con la vista.

Nota: el laboratorio de este ejercicio se hará en el módulo siguiente.