android

Grupo1:

Guillermo Blanco Andrés.
Raúl Moreno Martínez.
Carlos Mahía Rachero.
Borja Argibay Vázquez.

Sumario

1.OBJETIVO.

El objetivo de esta presentación es la explicación de estos componentes y tecnologías de manera que sepamos identificarlos y saber sus implementaciones:

- RecyclerView.
- Retrofit.
- Gson.
- Glide.

2. RecyclerView.

Definición:

Es una versión más avanzada del tradicional ListView y lo que hace es mostrar datos cuyos elementos se van reciclando cuando ya no son visibles por el scroll de la lista, lo que mejora la rendimiento en gran medida.

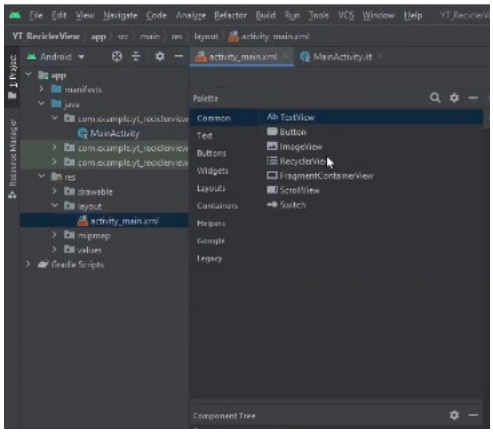
Función:

- Generar listas mucho más optimizadas.
- Se encargará de crear tantos view holder como sea necesario para mostrar los elementos de la lista que se ven en pantalla.
- «Reciclar» aquellos elementos que ya no sirven por estar asociados a otros elementos de la lista que ya han salido de la pantalla.

Pasos para crear:

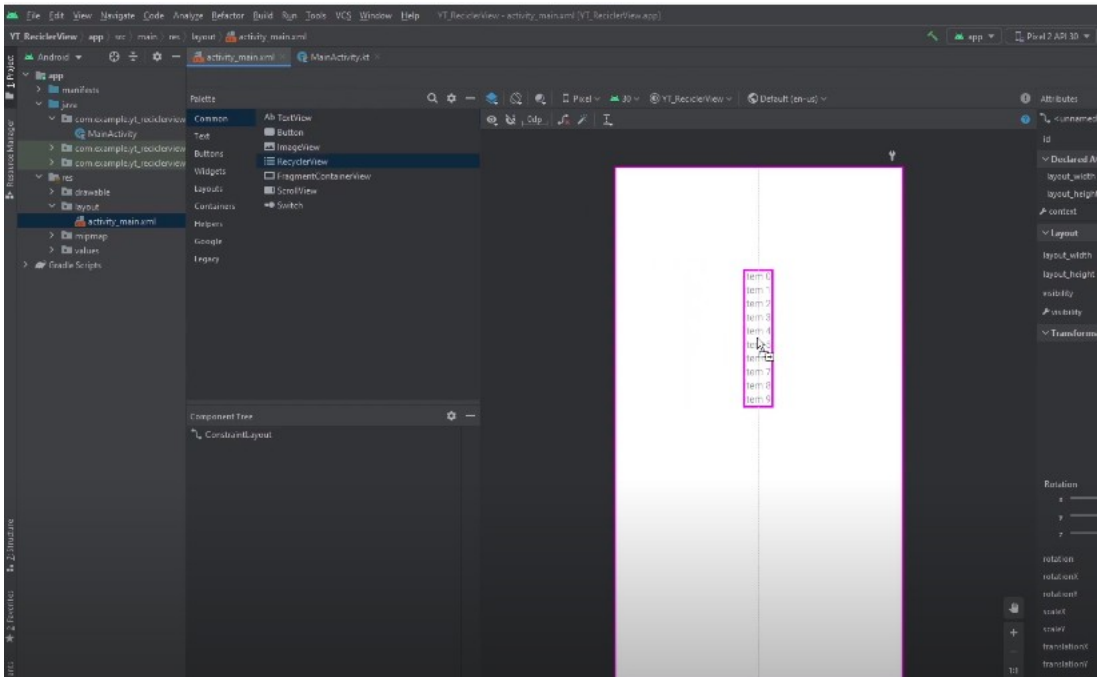
1ºPaso:

Abrir nuestro fichero ativity_main.



2ºPaso:

Introducir el elemento recyclerview dentro de nuestro menú.



3.Retrofit

Definición:

Es un cliente de servidores REST para Android y Java desarrollado por Square, permite hacer peticiones al servidor tipo: GET, POST, PUT, PATCH, DELETE y HEAD, y gestionar diferentes tipos de parámetros, paseando automáticamente la respuesta a un tipo de datos.

Función:

.Realizar Peticiones HTTP En Android

.@GET: Realiza una petición GET

.@POST: Realiza una petición POST

.@PUT: Realiza una petición PUT

.@DELETE: Realiza un petición DELETE

Como Añadirlo a nuestro proyecto:

Añadiremos retrofit a nuestro proyecto mediante Gradle.

1ºPaso:

Abrir nuestro gradle e introducir las siguientes implementaciones.

implementation 'com.squareup.retrofit2:retrofit:2.3.0'

implementation 'com.squareup.retrofit2:converter-gson:2.3.0'

implementation 'com.squareup.okhttp3:logging-interceptor:3.9.1'

```
21
22  dependencies {
23      compile fileTree(dir: 'libs', include: ['*.jar'])
24      androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
25          exclude group: 'com.android.support', module: 'support-annotations'
26      })
27
28      compile 'com.android.support:appcompat-v7:25.3.0'
29      compile 'com.android.support.constraint:constraint-layout:1.0.2'
30
31      compile 'com.squareup.retrofit2:retrofit:2.2.0'
32      compile 'com.squareup.retrofit2:converter-gson:2.2.0'
33
34      testCompile 'junit:junit:4.12'
35  }
36
```

Se recomienda añadir la dependencia :"logging interceptor".

2ºPaso:

Crear una clase y una interfaz

El ApiAdapter es una clase que se encargará de instanciar un objeto Retrofit (aplicando el patrón de diseño Singleton), y este objeto hará posible las peticiones.Además, en esta clase se definirá la ruta base de la API que queremos consultar. El ApiService en cambio es una interfaz. Aquí vamos a definir métodos abstractos. Cada método abstracto va a representar una ruta específica de nuestra API.

3ºPaso:

Añadir cuatro metodos:

.Una petición Get.

```
@GET("diseases")
        Call<DiseasesResponse> getDiseases();
```

.Una petición Post.

```
@FormUrlEncoded
 @POST("upload/photo")
  Call<SimpleResponse> postPhoto(
    @Field("image") String base64,
          @Field("extension") String extension,
           @Field("user_id") String user_id
```

.Inicio de sesión de una aplicación.

```
@GET("login")
  Call<LoginResponse> getLogin(
    @Query("username") String username,
          @Query("password") String password
```

.Registrar un producto a través de una petición Post.

```
@FormUrlEncoded
  @POST("product")
        Call<SimpleResponse> postNewProduct(
            @Field("code") String code,
                @Field("name") String name,
                        @Field("description") String description
```

Ejemplo de la clase implementada:

```
public class MyApiAdapter {

    private static MyApiService API_SERVICE;

    public static MyApiService getApiService() {

        // Creamos un interceptor y le indicamos el log level a usar
        HttpLoggingInterceptor logging = new HttpLoggingInterceptor();
        logging.setLevel(HttpLoggingInterceptor.Level.BODY);

        // Asociamos el interceptor a las peticiones
        OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
        httpClient.addInterceptor(logging);

        String baseUrl = "https://mi-pagina.com/api/";

        if (API_SERVICE == null) {
            Retrofit retrofit = new Retrofit.Builder()
                    .baseUrl(baseUrl)
                    .addConverterFactory(GsonConverterFactory.create())
                    .client(httpClient.build()) // <-- usamos el log level
                    .build();
            API_SERVICE = retrofit.create(MyApiService.class);
        }

        return API_SERVICE;
    }
```

4.GSON.

Definición:

GSON es un API en Java, desarrollada por Google, que se utiliza para convertir objetos Java a JSON (serialización) y JSON a objetos Java (deserialización).

Características son:

.Permite la conversión entre objetos Java y JSON de una manera sencilla, simplemente invocando los métodos toJson() o fromJson().

.Permite la conversión de objetos inmutables ya existentes.

.Soporte para tipos genéricos de Java.

.Permite la representación personalizada de objetos.

.Soporte para "Objetos arbitrariamente complejos".

Como usar:

Antes de nada necesitaremos añadir la siguiente dependencia a nuestra aplicación:

```Shell
    <dependency>
        <groupid>com.google.code.gson</groupid>
        <artifactid>gson</artifactid>
        <version>2.2.2</version>
    </dependency>
```

crear un objeto Gson e invocar a su método fromJson. Como parámetros le pasaremos el objeto JSON como String y la clase del objeto en que se deserializará.

```Shell
@Test
public void debeDevolverJSONEnUnProperties() {
    final String json = "{\"id\":46,\"nombre\":\"Miguel\",\"empresa\":\"Autentia\"}";
    final Gson gson = new Gson();
    final Properties properties = gson.fromJson(json, Properties.class);
    assertEquals("46", properties.getProperty("id"));
    assertEquals("Miguel", properties.getProperty("nombre"));
    assertEquals("Autentia", properties.getProperty("empresa"));
    assertNull(properties.getProperty("propiedadInexistente"));
}
```

5.GLIDE.

Definición:

Glide es una popular librería Android de código abierto para cargar imágenes, videos y GIFs animados.

Función:

Con Glide puedes cargar y mostrar medios de muchas fuentes diferentes, tales como servidores remotos o el sistema local de archivos. Por defecto, Glide usa una implementación personalizada de HttpURLConnection para cargar imágenes en internet. Sin embargo, Glide también proporciona complementos para otras librerías populares de red como Volley o OkHttp.

Ejemplo de uso

Integración en el glide estas implementaciones.

```
implementation('com.github.bumptech.glide:glide:4.8.0')
annotationProcessor 'com.github.bumptech.glide:compiler:4.8.0'
 // Si usa la anotación Glide en Kotlin, debe introducir la dependencia kapt en lugar de la a
//kapt 'com.github.bumptech.glide:compiler:4.8.0'
```

Se usa de esta manera.

```
1.  Glide.with(context).load(url).into(imageView)
```