

Retrofit Kotlin-Android



Pada pengembangan aplikasi Android modern, bekerja dengan RESTful APIs adalah keterampilan dasar. RESTful APIs memungkinkan komunikasi antara aplikasi Android Anda dan server, memungkinkan Anda mengirim dan menerima data. Dalam artikel ini, kita akan menjelajahi tiga perpustakaan populer di ekosistem Android untuk melakukan panggilan API: Retrofit, Volley, dan OkHttp.

Retrofit adalah klien HTTP yang aman untuk Android dan Java yang dikembangkan oleh Square. Ini menyederhanakan proses melakukan panggilan API dengan mengonversi titik akhir API HTTP menjadi antarmuka Java. Dengan penanganan data serialisasi dan deserialisasi yang efisien menggunakan converter, Retrofit telah menjadi pilihan populer di kalangan pengembang Android.

Volley adalah perpustakaan jaringan yang disediakan oleh Google. Ini menawarkan API yang mudah digunakan dan fleksibel untuk menangani permintaan jaringan, pemuatan gambar, dan caching. Volley sangat cocok untuk proyek-proyek kecil atau ketika panggilan API yang cepat dan sederhana diperlukan.

OkHttp adalah perpustakaan klien HTTP yang kuat lainnya dari Square. Ini berfungsi sebagai perpustakaan tingkat rendah untuk melakukan panggilan API dan sebagai lapisan jaringan yang mendasari untuk Retrofit. Fokus OkHttp yang kuat pada kinerja, keamanan, dan penyesuaian membuatnya menjadi pilihan yang serbaguna untuk berbagai proyek Android.

Retrofit adalah sebuah klien HTTP yang kuat untuk Android dan Java yang dibangun oleh tim yang luar biasa di Square. Retrofit dikonfigurasi dengan converter yang memudahkan dalam melakukan serialisasi dan deserialisasi untuk kumpulan data terstruktur. Biasanya untuk JSON, kita menggunakan converter Gson untuk proses serialisasi dan deserialisasi. Retrofit menggunakan perpustakaan Okhttp untuk permintaan HTTP.

1. Menambahkan Dependensi: Anda perlu membuka proyek Anda dan menambahkan dependensi ke file build.gradle level aplikasi (Module: app). Di sini, kita menambahkan dependensi Gson, RecyclerView, CardView, Picasso, dan Retrofit.

2. Memastikan Izin INTERNET: Pastikan Anda telah menambahkan izin INTERNET di file AndroidManifest.xml. Izin ini diperlukan agar aplikasi dapat menggunakan koneksi internet.

3. Membuat Model Data: Anda perlu membuat model data untuk mem-parsing respon JSON. Dalam contoh ini, kita membuat kelas Kotlin bernama DataModel.kt di bawah paket model. Ini akan memudahkan Retrofit untuk mem-parsing respon JSON ke dalam objek Kotlin.

4. Membuat Instance Retrofit: Anda perlu membuat sebuah instance Retrofit untuk melakukan permintaan jaringan ke API REST dengan Retrofit. Di sini, kita membuat objek Kotlin bernama ApiClient.kt di bawah paket retrofit. BASE_URL adalah URL dasar dari API Anda di mana kita akan melakukan panggilan.

Dependensi yang diperlukan

Lib.versions.toml

```
[versions]
agp = "8.5.0-alpha07"
kotlin = "1.9.0"
coreKtx = "1.13.1"
junit = "4.13.2"
junitVersion = "1.1.5"
espressoCore = "3.5.1"
appcompat = "1.6.1"
material = "1.12.0"
activity = "1.9.0"
constraintlayout = "2.1.4"

gsonVersion = "2.10.1"
retrofit2Gson = "2.3.0"
okhttp3 = "3.8.0"
picasso = "2.71828"

[libraries]
androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx" }

junit = { group = "junit", name = "junit", version.ref = "junit" }
androidx-junit = { group = "androidx.test.ext", name = "junit", version.ref = "junitVersion" }
androidx-espresso-core = { group = "androidx.test.espresso", name = "espresso-core", version.ref = "espressoCore" }
androidx-appcompat = { group = "androidx.appcompat", name = "appcompat", version.ref = "appcompat" }
material = { group = "com.google.android.material", name = "material", version.ref = "material" }
androidx-activity = { group = "androidx.activity", name = "activity", version.ref = "activity" }
androidx-constraintlayout = { group = "androidx.constraintlayout", name = "constraintlayout", version.ref = "constraintlayout" }

gson-v2101 = { module = "com.google.code.gson:gson", version.ref = "gsonVersion" }
retrofit = { module = "com.squareup.retrofit2:retrofit", version.ref = "retrofit2Gson" }
retrofit2-gson = { group = "com.squareup.retrofit2", name = "converter-gson", version.ref = "retrofit2Gson" }
okhttp3 = { group = "com.squareup.okhttp3", name = "logging-interceptor", version.ref = "okhttp3" }
picasso = { group = "com.squareup.picasso", name = "picasso", version.ref = "picasso" }
```

```
[plugins]
android-application = { id = "com.android.application", version.ref = "agp" }
jetbrains-kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }
```

Build.gradle.kts (Module:app)

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.jetbrains.kotlin.android)
}

android {
    namespace = "com.example.retrofit1"
    compileSdk = 34

    defaultConfig {
        applicationId = "com.example.retrofit1"
        minSdk = 34
        targetSdk = 34
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = "1.8"
    }
}

dependencies {

    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.appcompat)
    implementation(libs.material)
    implementation(libs.androidx.activity)
    implementation(libs.androidx.constraintlayout)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)

    implementation(libs.gson.v2101)
    implementation(libs.retrofit)
    implementation(libs.retrofit2.gson)
    implementation(libs.okhttp3)
    implementation(libs.picasso)
}
```

- Tambahkan INTERNET permission pada **AndroidManifest.xml** seperti berikut :.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

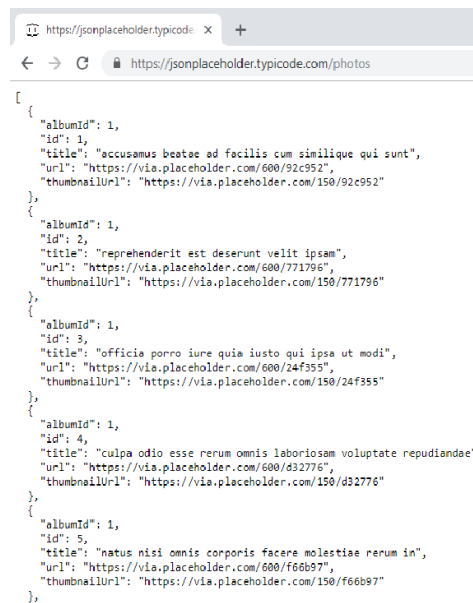
    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/Theme.Retrofit1"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

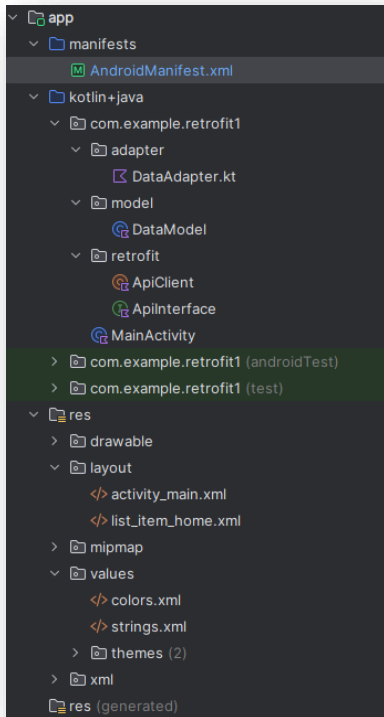
- Selanjutnya, kita perlu membuat model data untuk mem-parsing respons JSON contoh kita. Respons contoh diberikan di bawah ini.



```
[
  {
    "albumId": 1,
    "id": 1,
    "title": "accusamus beatae ad facilis cum similique qui sunt",
    "url": "https://via.placeholder.com/600/92c952",
    "thumbnailUrl": "https://via.placeholder.com/150/92c952"
  },
  {
    "albumId": 1,
    "id": 2,
    "title": "reprehenderit est deserunt velit ipsam",
    "url": "https://via.placeholder.com/600/771796",
    "thumbnailUrl": "https://via.placeholder.com/150/771796"
  },
  {
    "albumId": 1,
    "id": 3,
    "title": "officie porro iure quia iusto qui ipsa ut modi",
    "url": "https://via.placeholder.com/600/24f355",
    "thumbnailUrl": "https://via.placeholder.com/150/24f355"
  },
  {
    "albumId": 1,
    "id": 4,
    "title": "culpa odio esse rerum omnis laboriosam voluptate repudiandae",
    "url": "https://via.placeholder.com/600/d32776",
    "thumbnailUrl": "https://via.placeholder.com/150/d32776"
  },
  {
    "albumId": 1,
    "id": 5,
    "title": "natus nisi omnis corporis facere molestiae rerum in",
    "url": "https://via.placeholder.com/600/f66b97",
    "thumbnailUrl": "https://via.placeholder.com/150/f66b97"
  }
]
```

Sample response

Buat struktur proyek seperti berikut :



Sekarang Buatlah Kelas Kotlin bernama **DataModel.kt** didalam directory **model** seperti yang ditunjukkan di bawah ini.

DataModel.kt

```
package com.example.retrofit1.model

import com.google.gson.annotations.Expose
import com.google.gson.annotations.SerializedName

data class DataModel(

    @Expose
    @SerializedName("albumId")
    val albumid: Integer,
    @Expose
    @SerializedName("id")
    val id: Integer,
    @Expose
    @SerializedName("title")
    val title: String,
    @Expose
    @SerializedName("url")
    val url: String,
    @Expose
    @SerializedName("thumbnailUrl")
    val thumbnailurl: String
)
```

Mari kita bahas setiap baris kode dalam kelas DataModel:

1. `package com.example.retrofit1.model`: Ini adalah deklarasi paket di mana kelas `DataModel` berada. Ini menunjukkan bahwa kelas ini berada dalam paket `com.example.retrofit1.model`.
2. `import com.google.gson.annotations.Expose`: Ini adalah pernyataan impor yang mengimpor anotasi `Expose` dari pustaka Gson. Anotasi ini digunakan untuk menandai bidang yang harus di-ekspos selama proses serialisasi/deserialisasi Gson.
3. `import com.google.gson.annotations.SerializedName`: Ini adalah pernyataan impor yang mengimpor anotasi `SerializedName` dari pustaka Gson. Anotasi ini digunakan untuk menyesuaikan nama properti JSON dengan nama properti dalam kelas Kotlin.
4. `data class DataModel()`: Ini adalah deklarasi kelas Kotlin. Kata kunci `data` digunakan untuk mendeklarasikan kelas data, yang secara otomatis menyediakan metode `equals()`, `hashCode()`, `toString()`, dan metode lainnya secara otomatis berdasarkan properti-properti yang dideklarasikan di dalamnya.
5. `@Expose`: Ini adalah anotasi dari Gson yang menandai bahwa bidang tersebut harus di-ekspos selama proses serialisasi/deserialisasi. Biasanya digunakan bersamaan dengan anotasi `SerializedName`.
6. `@SerializedName("albumId")`: Ini adalah anotasi dari Gson yang menyesuaikan nama properti JSON dengan nama properti dalam kelas Kotlin. Dalam contoh ini, properti `albumId` dalam kelas Kotlin akan di-mapping dengan kunci `"albumId"` dalam JSON.
7. `val albumId: Integer`: Ini adalah deklarasi properti dalam kelas `DataModel`. Properti ini memiliki nama `albumId` dan tipe data `Integer`. Kata kunci `val` menunjukkan bahwa properti ini adalah properti hanya baca (immutable).
8. `val id: Integer`: Ini adalah deklarasi properti kedua dalam kelas `DataModel`. Properti ini memiliki nama `id` dan tipe data `Integer`.
9. `val title: String`: Ini adalah deklarasi properti ketiga dalam kelas `DataModel`. Properti ini memiliki nama `title` dan tipe data `String`.
10. `val url: String`: Ini adalah deklarasi properti keempat dalam kelas `DataModel`. Properti ini memiliki nama `url` dan tipe data `String`.
11. `val thumbnailurl: String`: Ini adalah deklarasi properti kelima dalam kelas `DataModel`. Properti ini memiliki nama `thumbnailurl` dan tipe data `String`.

Sekarang buatlah sebuah instance Retrofit untuk melakukan permintaan jaringan ke API REST dengan Retrofit.

- Di sini kita membuat sebuah objek Kotlin bernama `ApiClient.kt` di bawah paket `retrofit`.
- `BASE_URL` adalah URL dasar dari API Anda di mana kita akan melakukan panggilan.

`ApiClient.kt`

```
package com.example.retrofit1.retrofit

import com.google.gson.GsonBuilder
import okhttp3.OkHttpClient
import okhttp3.logging.HttpLoggingInterceptor
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

object ApiClient {

    var BASE_URL:String="https://jsonplaceholder.typicode.com/"
    val getClient: ApiInterface
        get() {
```

```

        val gson = GsonBuilder()
            .setLenient()
            .create()
        val interceptor = HttpLoggingInterceptor()
        interceptor.setLevel(HttpLoggingInterceptor.Level.BODY)
        val client = OkHttpClient.Builder().addInterceptor(interceptor).build()

        val retrofit = Retrofit.Builder()
            .baseUrl(BASE_URL)
            .client(client)
            .addConverterFactory(GsonConverterFactory.create(gson))
            .build()

        return retrofit.create(ApiInterface::class.java)
    }
}

```

Objek Kotlin bernama `ApiClient` di dalam paket `retrofit`. Objek ini bertanggung jawab untuk membuat instance `Retrofit` yang digunakan untuk melakukan permintaan jaringan ke API REST.

Penjelasan untuk setiap bagian dari kode:

1. `object ApiClient { ... }`: Ini adalah deklarasi objek Kotlin. Objek Kotlin adalah singleton, yang berarti bahwa hanya ada satu instance dari objek ini yang akan dibuat dan digunakan di seluruh aplikasi.
2. `var BASE_URL: String = "https://jsonplaceholder.typicode.com/"`: Ini adalah variabel yang menyimpan URL dasar dari API yang akan diakses.
3. `val getClient: ApiInterface`: Ini adalah properti yang memberikan akses ke instance `Retrofit` yang telah dikonfigurasi. Properti ini menggunakan pengambilan yang disetel (getter) yang kustom untuk membuat instance `Retrofit` ketika dipanggil.
4. Di dalam pengambilan yang disetel (getter), terdapat langkah-langkah berikut untuk membuat instance `Retrofit`:
 - o `GsonBuilder()` digunakan untuk membuat sebuah pembuat `Gson` yang dapat disesuaikan. Dalam contoh ini, metode `setLenient()` digunakan untuk mengizinkan pembaca `Gson` untuk menerima input yang tidak benar-benar valid, yang berguna saat bekerja dengan API yang tidak konsisten.
 - o `HttpLoggingInterceptor()` digunakan untuk mencatat log permintaan jaringan. Level logging yang ditetapkan ke `BODY` akan mencatat detail lengkap dari permintaan dan respons, termasuk payload data.
 - o `OkHttpClient.Builder()` digunakan untuk membuat sebuah klien HTTP menggunakan `OkHttp`. `Interceptor` logging yang telah dibuat ditambahkan ke klien.
 - o `Retrofit.Builder()` digunakan untuk membuat instance `Retrofit`. Di sini, kita menetapkan `BASE_URL`, klien `OkHttpClient` yang telah dikonfigurasi sebelumnya, dan converter `Gson` untuk melakukan konversi JSON ke objek Kotlin.
 - o `.build()` digunakan untuk menghasilkan instance `Retrofit` berdasarkan konfigurasi yang telah ditetapkan sebelumnya.
5. `return retrofit.create(ApiInterface::class.java)`: Akhirnya, metode `create()` dari instance `Retrofit` digunakan untuk membuat instance dari interface `ApiInterface`. Ini adalah interface yang digunakan untuk mendefinisikan endpoint API dan operasi permintaan yang diperlukan.

Definisikan endpoint yang akan didefinisikan pada **ApiInterface.kt** seperti berikut :

ApiInterface.kt

```
package com.example.retrofit1.retrofit

import com.example.retrofit1.model.DataModel
import retrofit2.Call
import retrofit2.http.GET

interface ApiInterface {

    @GET("photos")
    fun getPhotos(): Call<List<DataModel>>

}
```

Penjelasan dari kode program di dalam **ApiInterface.kt**:

1. `package com.example.retrofit1.retrofit`: Ini adalah deklarasi paket di mana interface **ApiInterface** berada. Interface ini digunakan untuk mendefinisikan titik akhir (endpoints) dari API yang akan diakses.
2. `import com.example.retrofit1.model.DataModel`: Ini adalah pernyataan impor yang mengimpor kelas **DataModel** yang telah kita buat sebelumnya. Interface **ApiInterface** akan menggunakan kelas ini untuk menentukan tipe data yang dikembalikan oleh endpoint API.
3. `import retrofit2.Call`: Ini adalah pernyataan impor yang mengimpor kelas **Call** dari Retrofit. Kelas **Call** digunakan untuk membuat permintaan jaringan asinkron dan menerima respons.
4. `import retrofit2.http.GET`: Ini adalah pernyataan impor yang mengimpor anotasi **GET** dari Retrofit. Anotasi ini digunakan untuk menandai metode HTTP yang digunakan untuk mengakses suatu resource pada server. Dalam konteks ini, metode **GET** digunakan untuk mengambil data dari server.
5. `interface ApiInterface { ... }`: Ini adalah deklarasi interface Kotlin yang akan mendefinisikan titik akhir API. Interface ini akan menyediakan definisi metode untuk setiap endpoint yang akan diakses.
6. `@GET("photos")`: Ini adalah anotasi Retrofit yang menandai metode `getPhotos()`. Anotasi ini menyatakan bahwa metode ini akan mengakses titik akhir API yang diberi nama "photos". Dalam hal ini, **photos** adalah bagian akhir dari URL lengkap yang akan dibuat saat menggunakan instance Retrofit.
7. `fun getPhotos(): Call<List<DataModel>>`: Ini adalah definisi metode `getPhotos()` yang akan diakses oleh Retrofit. Metode ini mengembalikan objek **Call** yang berisi daftar **DataModel**. Metode ini akan mengirimkan permintaan HTTP GET ke titik akhir API yang didefinisikan oleh anotasi **@GET**, dan kemudian akan menerima respons dalam bentuk daftar objek **DataModel**.

Sekarang kita sudah siap untuk melakukan panggilan API di **MainActivity.kt**.

- Metode `getData()` digunakan untuk melakukan permintaan HTTP menggunakan Retrofit.
- Di sini kita mengambil data dan mempopulasikannya ke dalam RecyclerView, adapter RecyclerView **DataAdapter** dan layout RecyclerView akan dijelaskan nanti.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
```



```

xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/main"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">

<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

MainActivity.kt

```

package com.example.retrofit1

import android.app.ProgressDialog
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.retrofit1.adapter.DataAdpter
import com.example.retrofit1.model.DataModel
import com.example.retrofit1.retrofit.ApiClient
import retrofit2.Call
import retrofit2.Callback
import retrofit2.Response

class MainActivity : AppCompatActivity() {

    lateinit var progerssProgressDialog: ProgressDialog
    var dataList = ArrayList<DataModel>()
    lateinit var recyclerView: RecyclerView
    lateinit var adapter: DataAdpter
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        recyclerView = findViewById(R.id.recycler_view)
        //setting up the adapter
        recyclerView.adapter= DataAdpter(dataList,this)
        recyclerView.layoutManager=
            LinearLayoutManager(this,LinearLayoutManager.VERTICAL,false)
        progerssProgressDialog=ProgressDialog(this)
        progerssProgressDialog.setTitle("Loading")
        progerssProgressDialog.setCancelable(false)
        progerssProgressDialog.show()
        getData()

    }

    private fun getData() {
        val call: Call<List<DataModel>> = ApiClient.getClient().getPhotos()
        call.enqueue(object : Callback<List<DataModel>> {

```

```

        override fun onResponse(call: Call<List<DataModel>>?, response:
            Response<List<DataModel>>?) {
                progerssProgressDialog.dismiss()
                dataList.addAll(response!!.body()!!)
                recyclerView.adapter?.notifyDataSetChanged()
            }

        override fun onFailure(call: Call<List<DataModel>>?, t: Throwable?) {
            progerssProgressDialog.dismiss()
        }
    })
}
}

```

Berikut adalah penjelasan kode program dari MainActivity.kt:

1. `import android.app.AlertDialog`: Ini adalah pernyataan impor yang mengimpor kelas `AlertDialog` yang digunakan untuk menampilkan dialog progres saat melakukan pengambilan data.
2. `import androidx.appcompat.app.AppCompatActivity`: Ini adalah pernyataan impor yang mengimpor kelas `AppCompatActivity` dari `AndroidX`, yang digunakan sebagai dasar aktivitas dalam aplikasi Android.
3. `import androidx.recyclerview.widget.LinearLayoutManager`: Ini adalah pernyataan impor yang mengimpor kelas `LinearLayoutManager` yang akan digunakan sebagai manajer layout untuk `RecyclerView`.
4. `import androidx.recyclerview.widget.RecyclerView`: Ini adalah pernyataan impor yang mengimpor kelas `RecyclerView` yang digunakan untuk menampilkan data dalam daftar gulir.
5. `import com.example.retrofit1.adapter.DataAdapter`: Ini adalah pernyataan impor yang mengimpor kelas `DataAdapter` yang akan digunakan sebagai adapter untuk `RecyclerView`.
6. `import com.example.retrofit1.model.DataModel`: Ini adalah pernyataan impor yang mengimpor kelas `DataModel` yang telah kita buat sebelumnya untuk merepresentasikan data yang diterima dari API.
7. `import com.example.retrofit1.retrofit.ApiClient`: Ini adalah pernyataan impor yang mengimpor objek `ApiClient` yang telah kita buat sebelumnya untuk melakukan panggilan ke API menggunakan `Retrofit`.
8. `class MainActivity : AppCompatActivity() { ... }`: Ini adalah deklarasi kelas Kotlin `MainActivity` yang merupakan turunan dari kelas `AppCompatActivity`. Aktivitas ini akan menjadi aktivitas utama dalam aplikasi.
9. Di dalam metode `onCreate(savedInstanceState: Bundle?)`, yang dipanggil ketika aktivitas dibuat, langkah-langkah berikut dilakukan:
 - o `setContentView(R.layout.activity_main)`: Ini mengatur tata letak tampilan aktivitas menggunakan file layout XML `activity_main.xml`.
 - o Inisialisasi `RecyclerView`, adapter, dan tampilkan data progres `ProgressDialog`.
10. `private fun getData() { ... }`: Ini adalah metode yang digunakan untuk melakukan panggilan ke API menggunakan `Retrofit`. Di dalam metode ini, kita membuat panggilan asinkron ke API menggunakan `ApiClient.getClient().getPhotos()` dan menangani respons dan kesalahan yang diterima menggunakan `enqueue()`.
11. Di dalam `enqueue()`, kita menangani respons menggunakan `onResponse()` dan kesalahan menggunakan `onFailure()`. Pada saat respons diterima, data yang diterima ditambahkan ke `dataList` dan adapter

RecyclerView diberitahu untuk memperbarui tampilan dengan memanggil `notifyDataSetChanged()`. Pada saat kesalahan, `ProgressDialog` akan dihilangkan.

Untuk menyiapkan RecyclerView, kita memerlukan tata letak daftar dan adapter RecyclerView yang disebut `DataAdapter.kt` seperti berikut.

- Tata letak ini hanya mencakup satu teks view untuk menampilkan judul. Buat layout baru pada directory `res` dan beri nama **`list_item_home`**

`list_item_home.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_margin="6dp"
    app:contentPadding="12dp"
    android:layout_height="wrap_content">

    <TextView
        android:maxLines="1"
        android:id="@+id/title"
        android:layout_gravity="center"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</androidx.cardview.widget.CardView>
```

`DataAdapter.kt`

```
package com.example.retrofit1.adapter

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.retrofit1.R
import com.example.retrofit1.model.DataModel

class DataAdapter(private var dataList: List<DataModel>, private val context: Context) :
    RecyclerView.Adapter<DataAdapter.ViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        return ViewHolder(LayoutInflater.from(context).inflate(R.layout.list_item_home,
parent, false))
    }

    override fun getItemCount(): Int {
        return dataList.size
    }
}
```

```

override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    val dataModel=dataList.get(position)

    holder.titleTextView.text=dataModel.title
}

class ViewHolder(itemLayoutView: View) : RecyclerView.ViewHolder(itemLayoutView) {
    lateinit var titleTextView: TextView
    init {
        titleTextView=itemLayoutView.findViewById(R.id.title)
    }
}
}

```

Penjelasan setiap fungsi dan kelas DataAdapter.kt:

1. `onCreateViewHolder(parent: ViewGroup, viewType: Int):` Fungsi ini digunakan untuk membuat instance `ViewHolder` yang akan digunakan untuk menampilkan item di dalam `RecyclerView`. Di dalamnya:
 - o `LayoutInflater.from(context).inflate(R.layout.list_item_home, parent, false):` Ini adalah proses inflasi (inflation) untuk mengubah tata letak XML yang telah ditentukan sebelumnya menjadi tampilan objek. Kami menggunakan `LayoutInflater` untuk melakukan inflasi layout `list_item_home` ke dalam tampilan objek.
 - o `ViewHolder(...):` Kami menginstansiasi `ViewHolder` dengan tampilan yang telah di-inflasi sebelumnya dan mengembalikannya.
2. `getItemCount():` Fungsi ini mengembalikan jumlah item dalam daftar data. Dalam hal ini, jumlahnya adalah panjang dari `dataList`, yaitu jumlah objek `DataModel` yang ada.
3. `onBindViewHolder(holder: ViewHolder, position: Int):` Fungsi ini digunakan untuk mengikat data dari posisi tertentu dalam daftar ke tampilan `ViewHolder`. Di dalamnya:
 - o `val dataModel = dataList[position]:` Kami mendapatkan objek `DataModel` pada posisi tertentu dalam `dataList`.
 - o `holder.titleTextView.text = dataModel.title:` Kami menetapkan teks dari judul `DataModel` ke `titleTextView` dalam `ViewHolder` yang sesuai. `titleTextView` adalah properti dalam `ViewHolder` yang digunakan untuk menampilkan judul dalam tampilan item daftar.
4. `class ViewHolder(itemLayoutView: View) : RecyclerView.ViewHolder(itemLayoutView) { ... }:` Ini adalah kelas inner `ViewHolder` yang bertanggung jawab untuk menyimpan referensi ke tampilan dalam item daftar. Di dalamnya:
 - o `lateinit var titleTextView: TextView:` Ini adalah properti `titleTextView` yang digunakan untuk menyimpan referensi ke `TextView` yang akan menampilkan judul item.
 - o `Blok init { ... }:` Di dalam blok inisialisasi, kami menginisialisasi properti `titleTextView` dengan mencari tampilan `TextView` yang sesuai dalam layout item daftar menggunakan ID `R.id.title`.

jalankan projek dan output program akan seperti berikut :

