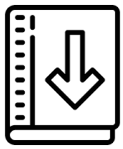
MODUL 11
Mengambil Data dan menampilkan gambar dari Internet

CAPAIAN PEMBELAJARAN

1. Mahasiswa dapat membuat aplikasi Koneksi dengan layanan backend server.

KEBUTUHAN ALAT/BAHAN/SOFTWARE

1. Android Studio 3.4.
2. Handphone Android versi 7.0 (Nougat)
3. Kabel data USB.
4. Driver ADB.



DASAR TEORI

1. Retrofit (digunakan untuk koneksi ke API web service)

Apa itu Retrofit?

https://code.tutsplus.com/id/tutorials/sending-data-with-retrofit-2-http-client-for-android--cms-27845

Retrofit adalah client HTTP type-safe untuk Android dan Java. Retrofit memudahkan untuk terhubung ke layanan web REST dengan menerjemahkan API ke dalam antarmuka Java. Kita akan melihat salah satu library HTTP yang paling populer dan sering direkomendasikan untuk Android. Library yang kuat ini mempermudah penggunaan data JSON atau XML, yang kemudian diurai menjadi Plain Old Java Objects (POJOs).

Permintaan GET, POST, PUT, PATCH, dan DELETE semua bisa dieksekusi. Seperti kebanyakan perangkat lunak open-source, Retrofit dibangun di atas beberapa library dan alat bantu lainnya. Di balik layar, Retrofit memanfaatkan OkHttp (dari pengembang yang sama) untuk menangani permintaan jaringan. Selain itu, Retrofit tidak memiliki konverter JSON built-in untuk mengurai dari objek JSON ke Java. Sebagai gantinya, ini mendukung library konverter JSON untuk menangani hal itu:

Gson: `com.squareup.retrofit:converter-gson`
Jackson: `com.squareup.retrofit:converter-jackson`
Moshi: `com.squareup.retrofit:converter-moshi`

Untuk Protocol buffers, Retrofit mendukung:

Protobuf: `com.squareup.retrofit2:converter-protobuf`
Wire: `com.squareup.retrofit2:converter-wire`

dan untuk XML Retrofit, mendukung:

Simple Framework: `com.squareup.retrofit2:converter-simpleframework`

Jadi Mengapa Menggunakan Retrofit?

Mengembangkan library HTTP type-safe sendiri untuk berinteraksi dengan API REST bisa menjadi kesulitan yang nyata: kita harus menangani banyak aspek, seperti membuat koneksi, caching, mencoba kembali permintaan yang gagal, threading, parsing respons, penanganan kesalahan, dan banyak lagi. Retrofit, di sisi lain, adalah library yang terencana, terdokumentasi dan teruji yang akan menghemat banyak waktu dan tenaga.

Kita akan pelajari cara menggunakan Retrofit 2 untuk menangani permintaan jaringan dengan Retrofit 2 untuk menangani permintaan jaringan dengan membangun aplikasi sederhana yang akan menjalankan permintaan POST, permintaan PUT (untuk memperbarui entitas), dan permintaan DELETE. Kita juga akan pelajari cara mengintegrasikannya dengan RxJava dan cara membatalkan permintaan. Kita akan menggunakan API yang disediakan oleh JSONPlaceholder - ini adalah API REST online palsu untuk pengujian dan pembuatan prototipe.

Glide (untuk download gambar (caching))

Apa itu Glide?

https://code.tutsplus.com/id/tutorials/code-an-image-gallery-android-app-with-glide--cms-28207

Glide adalah sumber terbuka library Android yang populer untuk memuat gambar, video, dan GIF animasi. Dengan Glide, anda dapat memuat dan menampilkan media dari berbagai sumber, seperti server jarak jauh atau sistem file lokal. Secara default, Glide menggunakan penerapan khusus **HttpURLConnection** untuk memuat gambar melalui internet. Namun, Glide juga menyediakan plugin ke pustaka jaringan populer lainnya seperti **Volley** atau **OkHttp**.

Jadi mengapa menggunakan glide?

Mengembangkan fungsi pemuatan dan tampilan media kita sendiri di Java bisa menjadi keribetan yang nyata: kita harus mengurus cache, decoding, mengelola koneksi jaringan,

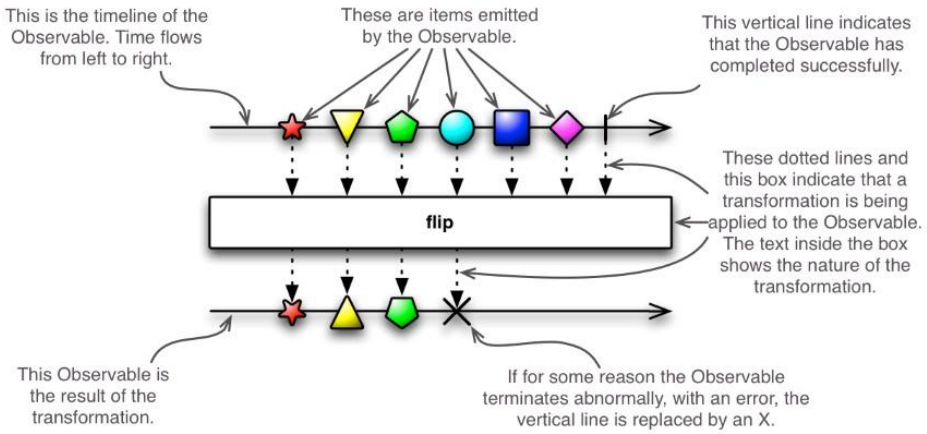
threading, exception, dan banyak lagi. Glide adalah library yang mudah digunakan, terencana dengan baik, terdokumentasi dengan baik, dan benar-benar teruji yang dapat menghemat banyak waktu berharga.

Reactivex (untuk mendukung mvvm)

Apa itu RxJava?

https://code.tutsplus.com/id/tutorials/getting-started-with-rxjava-20-for-android--cms-28345

RxJava adalah library yang memungkinkan kita membuat aplikasi dalam gaya pemrograman reaktif. Pada intinya, pemrograman reaktif menyediakan cara pemrosesan dan reaksi yang bersih dan efisien terhadap arus data real-time, termasuk data dengan nilai dinamis. Aliran data ini belum tentu harus mengambil bentuk tipe data tradisional, karena RxJava cukup banyak memperlakukan semuanya sebagai arus data - mulai dari variabel hingga properti, cache, dan bahkan events masukan pengguna seperti klik dan gesekan. Data yang dipancarkan oleh setiap aliran dapat berupa nilai, kesalahan, atau sinyal "komplit", walaupun kita tidak perlu menerapkan dua hal terakhir. Setelah membuat aliran pemancar data, kita menggabungkannya dengan objek reaktif yang mengkonsumsi dan kemudian bertindak berdasarkan data ini, melakukan tindakan yang berbeda bergantung pada apa yang telah dipancarkan oleh arus. RxJava meliputi sejumlah operator yang berguna untuk bekerja dengan aliran, sehingga mudah untuk melakukan hal-hal seperti filtering, pemetaan, menunda, penghitungan, dan banyak lagi.



Untuk membuat alur data stream dan objek yang bereaksi terhadapnya, RxJava memperluas pola perancangan perangkat lunak Observer. Intinya, di RxJava, kita miliki objek Observable yang memancarkan aliran data dan kemudian berhenti, dan objek Observer yang berlangganan menjadi Observable. Observer menerima pemberitahuan setiap kali mereka menugaskan Observable memancarkan nilai, kesalahan, atau sinyal yang telah komplit.

Jadi pada tingkat yang sangat tinggi, RxJava adalah tentang:

- Membuat Observable.

- Memberikan beberapa data Observable untuk dipancarkan.
- Membuat Observer.
- Menetapkan Observer menjadi Observable.
- Memberi tugas Observer untuk melakukan setiap kali menerima emisi dari Observable yang ditugaskan.

Mengapa RxJava?

Belajar teknologi baru membutuhkan waktu dan usaha, dan karena library berorientasi data, RxJava tidak selalu menjadi API yang paling mudah untuk diatasi. Untuk membantu kita memutuskan apakah belajar RxJava layak untuk investasi awal, mari jelajahi beberapa manfaat utama untuk menambahkan library RxJava ke proyek Android kita.

Lebih ringkas, kode yang dapat dibaca. Kode yang rumit, verbose dan sulit dibaca adalah *selalu* kabar buruk. Kode Messy lebih rentan terhadap bug dan inefisiensi lainnya, dan jika ada kesalahan yang terjadi maka kita akan memiliki waktu yang jauh lebih sulit untuk melacak sumber kesalahan ini jika kode kita berantakan. Bahkan jika proyek kita tidak dibangun tanpa ada kesalahan, kode yang rumit masih bisa kembali menghantui kita biasanya saat kita memutuskan untuk merilis pembaruan. RxJava menyederhanakan kode yang diperlukan untuk menangani data dan events dengan memungkinkan kita mendeskripsikan apa yang ingin kita capai, daripada menulis daftar instruksi untuk aplikasi kita. RxJava juga menyediakan alur kerja sekitar yang dapat kita gunakan untuk menangani semua data dan events di seluruh aplikasi kita-buatlah Observable, buat Observer, menetapkan observable ke observer, rinsen dan ulangi. Pendekatan formulatik ini membuat kode yang mudah dibaca dan mudah dibaca manusia.

Multithreading Made Easy. Aplikasi android modern harus bisa *multitask*. Paling tidak, pengguna kita akan berharap untuk dapat terus berinteraksi dengan UI saat aplikasi kita melakukan beberapa pekerjaan di latar belakang, seperti mengelola koneksi jaringan, mendownload file, atau bermain musik. Masalahnya adalah bahwa Android adalah single-threaded secara default, jadi jika aplikasi kita pernah berhasil multi-task maka kita perlu membuat beberapa thread tambahan. Android memang menyediakan sejumlah cara untuk membuat thread tambahan, seperti layanan dan IntentServices, namun tidak satupun dari solusi ini sangat mudah diterapkan, dan mereka dapat dengan cepat menghasilkan kompleks, kode verbose yang rentan terhadap kesalahan. RxJava bertujuan untuk menghilangkan kesulitan dari pembuatan aplikasi Android multi-threaded, dengan menyediakan penjadwal dan operator khusus. RxJava memberi kita cara mudah untuk menentukan thread dimana pekerjaan harus dilakukan dan thread di mana hasil dari pekerjaan ini harus diposting. RxJava 2.0 mencakup sejumlah penjadwal secara default, termasuk Schedulers.newThread, yang *terutama* berguna karena menciptakan thread baru. Untuk mengubah thread tempat di mana pekerjaan dilakukan, kita hanya perlu mengubah tempat observer berlangganan ke observable,

dengan menggunakan operator subscribeOn. Sebagai contoh, di sini kita membuat thread baru dan menentukan bahwa pekerjaan harus dilakukan pada thread baru ini:

```
observable.subscribeOn(Schedulers.newThread())
```

Masalah lama lainnya dengan multithreading di Android adalah kita hanya dapat memperbarui UI aplikasi kita dari untaian utama. Biasanya, kapan pun kita perlu mengeposkan hasil beberapa karya latar ke UI aplikasi kita, kita harus membuat Handler yang berdedikasi. Sekali lagi, RxJava memiliki solusi yang jauh lebih mudah. Kita bisa menggunakan operator observeOn untuk menentukan bahwa Observable harus mengirimkan notifikasinya menggunakan scheduler yang berbeda, yang pada dasarnya memungkinkan kita mengirim data Observable kita ke thread pilihan kita, termasuk thread UI utama. Ini berarti bahwa dengan hanya dua baris kode, kita dapat membuat thread baru dan mengirim hasil kerja yang dilakukan di thread ini ke thread utama Android:

```
.subscribeOn(Schedulers.newThread())
.observeOn(AndroidSchedulers.mainThread())
```

Peningkatan Fleksibilitas. Observables memancarkan data mereka dengan cara yang sepenuhnya menyembunyikan cara data dibuat. Karena observers kita bahkan tidak dapat melihat bagaimana data dibuat, kita bebas untuk menerapkan Observables kita dengan cara apapun yang kita inginkan. Setelah kita menerapkan Observables kita, RxJava menyediakan operator dalam jumlah besar yang dapat kita gunakan untuk menyaring, menggabungkan dan mengubah data yang sedang dipancarkan oleh Observables. Kita bahkan bisa menjerumuskan lebih dan lebih banyak operator bersamaan sampai kita menciptakan aliran data yang tepa*t* dibutuhkan aplikasi kita. Misalnya, kita dapat menggabungkan data dari beberapa aliran, memfilter arus yang baru digabungkan, lalu menggunakan data yang dihasilkan sebagai masukan untuk arus data berikutnya. Dan ingat bahwa di RxJava, hampir semua hal diperlakukan sebagai aliran data, jadi kita bahkan dapat menerapkan operator ini ke "data" non-tradisional, seperti events klik.

Membuat Aplikasi yang Lebih Responsif. Gone adalah hari-hari ketika sebuah aplikasi bisa lolos dengan memuat halaman konten dan kemudian menunggu pengguna untuk mengetuk tombol **Berikutnya**. Saat ini, aplikasi seluler khas kita harus dapat bereaksi terhadap berbagai peristiwa dan data yang terus berkembang, idealnya secara real time. Misalnya, aplikasi jejaring sosial khas kita harus terus mendengarkan keinginan masuk, komentar, dan permintaan pertemanan, sambil mengelola koneksi jaringan di latar belakang dan merespon segera kapan pun pengguna mengetuk atau menggesek layar. Library RxJava dirancang untuk dapat mengelola berbagai data dan events secara bersamaan dan secara real time, menjadikannya alat yang ampuh untuk menciptakan jenis aplikasi yang sangat responsif yang diharapkan oleh pengguna ponsel modern.

PRAKTIK

1. Kita akan membuat aplikasi untuk mengambil data dari Internet dan menampilkan gambar hasil ke view.
2. Atur gradle Module.app. Tambahkan

```
apply plugin: 'kotlin-kapt'
```

```
def lifeCycleExtensionVersion = '1.1.1'
def supportVersion = '28.0.0'
def retrofitVersion = '2.3.0'
def glideVersion = '4.8.0'
def rxJavaVersion = '2.0.1'

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.2.1'

    //retrofit
    implementation "com.squareup.retrofit2:retrofit:$retrofitVersion"
    implementation "com.squareup.retrofit2:converter-gson:$retrofitVersion"
    implementation "com.squareup.retrofit2:adapter-rxjava2:$retrofitVersion"

    //rxJava
    implementation "io.reactivex.rxjava2:rxjava:$rxJavaVersion"
    implementation "io.reactivex.rxjava2:rxandroid:$rxJavaVersion"

    //Glide
    implementation "com.github.bumptech.glide:glide:$glideVersion"

    //ui
    implementation "com.android.support:appcompat-v7:$supportVersion"
    implementation
"android.arch.lifecycle:extensions:$lifeCycleExtensionVersion"
    implementation 'com.android.support:cardview-v7:28.0.0'
    implementation "com.android.support:recyclerview-v7:$supportVersion"
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
}
```

3. Buat sebuah project, kemudian buat 4 buah package : **api, model, view, viewmodel**
4. Pindahkan MainActivity.kt ke dalam package **view**.
5. Buat interface data **PhotosApi** di dalam package **api**

```
interface PhotosApi {
    @GET("photos")
    fun getPhotos(): Single<List<Photo>>
}
```

6. Buat kelas **PhotosService** untuk membuat instance dari retrofit, masih di package **api**

```
class PhotosService {
    private val BASE_URL = "https://jsonplaceholder.typicode.com/"
    private val api: PhotosApi

    init {
        api = Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
            .build()
            .create(PhotosApi::class.java)
    }

    fun getPhotos(): Single<List<Photo>> {
        return api.getPhotos()
    }
}
```

7. Buat kelas data **Photo** untuk menghubungkan model ke android, di package model

```
data class Photo(
    @SerializedName("id")
    val id: Int?,
    @SerializedName("title")
    val title: String?,
    @SerializedName("thumbnailUrl")
    val thumbnail: String?
)
```

8. Buat kelas PhotoListAdapter di package view

```
class PhotoListAdapter(var photos: ArrayList<Photo>) :
RecyclerView.Adapter<PhotoListAdapter.ViewHolder>() {

    fun updatePhotos(newPhotos: List<Photo>) {
        photos.clear()
        photos.addAll(newPhotos)
        notifyDataSetChanged()
    }

    override fun onCreateViewHolder(parent: ViewGroup, p1: Int) = ViewHolder(
        LayoutInflater.from(parent.context).inflate(R.layout.item_list,
parent, false)
    )

    override fun getItemCount() = photos.size

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        holder.bind(photos[position])
    }

    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        fun bind(photos: Photo) {
            itemView.tvTitle.text = photos.title
            itemView.setOnClickListener { view ->

Toast.makeText(itemView.context,"Hello",Toast.LENGTH_LONG).show()
            }

Glide.with(itemView.context).load(photos.thumbnail).into(itemView.imageView)
        }
```

```
    }
}
```

9. Buat kelas ListViewMode di package viewmodel

```
class ListViewModel : ViewModel() {
    private val photosService = PhotosService()
    private val disposable = CompositeDisposable()
    val photos = MutableLiveData<List<Photo>>()

    fun fetchData() {
        disposable.add(
            photosService.getPhotos()
                .subscribeOn(Schedulers.newThread())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribeWith(object :
DisposableSingleObserver<List<Photo>>() {
                    override fun onSuccess(value: List<Photo>?) {
                        photos.value = value
                    }
                    override fun onError(e: Throwable?) {
                        Log.e("ERRORFETCHDATA", "error$e")
                    }
                })
        )
    }

    override fun onCleared() {
        super.onCleared()
        disposable.clear()
    }
}
```

10. Tambahkan komponen **RecyclerView** pada layout **activity_main.xml**

```
<android.support.v7.widget.RecyclerView
        android:id="@+id/rv_list"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginEnd="8dp"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toBottomOf="parent"/>
```

11. Buat file layout dengan nama **item_list.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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:layout_width="match_parent"
              android:orientation="horizontal"
              android:layout_marginBottom="8dp"
              android:layout_height="wrap_content">

    <android.support.v7.widget.CardView
            android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            tools:layout_editor_absoluteX="8dp"
            app:layout_constraintTop_toTopOf="parent">
        <android.support.constraint.ConstraintLayout
                android:layout_width="match_parent"
                android:padding="8dp"
                android:layout_height="wrap_content">
            <TextView
                    android:text="@string/imagename"
                    android:layout_width="0dp"
                    android:layout_height="wrap_content"
                    android:id="@+id/tvTitle"
                    app:layout_constraintBottom_toBottomOf="@+id/imageView"

android:textAppearance="@style/TextAppearance.AppCompat.Small"
                    app:layout_constraintTop_toTopOf="@+id/imageView"
                    app:layout_constraintStart_toEndOf="@+id/imageView"
                  android:layout_marginStart="8dp"
            />
            <ImageView
                    android:layout_width="80dp"
                    android:scaleType="fitXY"
                    android:layout_height="80dp"
                  tools:srcCompat="@tools:sample/avatars"
                    android:id="@+id/imageView"
                    app:layout_constraintStart_toStartOf="parent"
                    app:layout_constraintTop_toTopOf="parent"
                    app:layout_constraintBottom_toBottomOf="parent"/>

        </android.support.constraint.ConstraintLayout>
    </android.support.v7.widget.CardView>
</LinearLayout>
```

12. Tambahkan resource string berikut

```
<string name="imagename">imagename</string>
```

13. Ubah MainActivity sehingga menjadi sebagai berikut

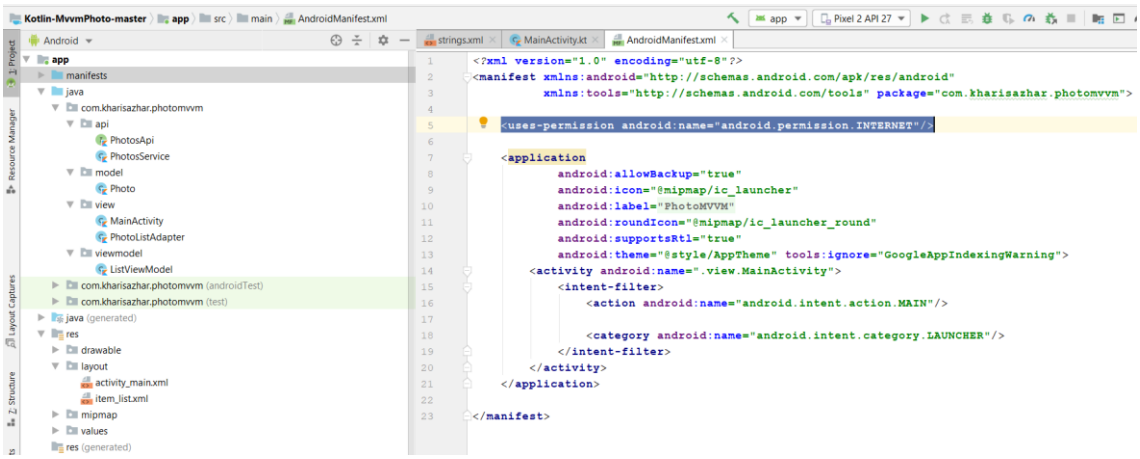
```
class MainActivity : AppCompatActivity() {

    lateinit var viewModel: ListViewModel
    private val photoListAdapter = PhotoListAdapter(arrayListOf())

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        viewModel = ViewModelProviders.of(this).get(ListViewModel::class.java)
        viewModel.fetchData()

        rv_list.apply {
            layoutManager = LinearLayoutManager(context)
            adapter = photoListAdapter
        }
        observeViewModel()
    }

    fun observeViewModel() {
        viewModel.photos.observe(this, Observer { photos ->
            photos?.let {
                photoListAdapter.updatePhotos(it)
```

```
                }
            })
        }
}
```

14. Berikan permission untuk internet access pada AndroidManifest

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

15. Susunan akhir file menjadi sebagai berikut. Perhatikan di **AndroidManifest**. Untuk activity dari **MainActivity** berada pada package **view**.



16. Jalankan dan amati hasilnya.
17. Analisislah hasil tampilan tersebut alurnya bagaimana.

LATIHAN

1. Modifikasilah aplikasi dengan menambahkan detil data yang lain. Akses api dari alamat web nya

TUGAS

1. Buat aplikasi baru dengan mengembangkan project di atas

REFERENSI

1. https://kotlinlang.org/docs/reference/
2. https://developer.android.com/kotlin
3. https://developer.android.com/courses/kotlin-android-fundamentals/toc
4. https://codelabs.developers.google.com/android-kotlin-fundamentals/
5. https://developer.android.com/kotlin/learn
6. https://developer.android.com/kotlin/resources