



Project Based Internship

Android

UI & Design Pattern

Daftar Isi

A. XML pada Android	3
B. View dan ViewGroup	3
C. Struktur XML untuk Layouting di Android	5
D. UI Design Pattern Android	6
E. Golden Rules of Interface Design	6
F. Architecture Design Pattern pada Android	9
G. Deep Dive MVVM pada Android	14
References	19

A. XML pada Android

Extended Markup Language atau yang biasa dikenal dengan XML merupakan sebuah bahasa markup yang diprakarsai oleh World Wide Web (W3C) dan merupakan versi lanjutan dari HTML yang memproses dan mengolah sebuah informasi atau element. Pada Android Studio, penulisan atau coding XML berfungsi sebagai salah satu cara untuk melakukan penempatan tata letak yang nantinya akan digunakan sebagai User Interface untuk para user aplikasi Android.

Semua elemen yang terdapat pada layout dibuat dengan menggunakan hierarki objek View dan View Group. View biasanya berisikan tentang elemen yang terlihat dan user dapat melakukan interaksi dengan elemen tersebut. Sedangkan, View Group merupakan container tak terlihat yang menentukan struktur tata letak bagi View dan objek View Group.

Struktur dalam XML terbilang cukup sederhana, karena hanya memuat tiga segmen saja, yaitu :

1. **Deklarasi**, bagian yang menunjukkan versi XML yang digunakan.
2. **Atribut**, bagian yang berisi keterangan objek seperti nama, judul dan sejenisnya.
3. **Elemen**, bagian yang berisi tag yang mendeskripsikan objek.

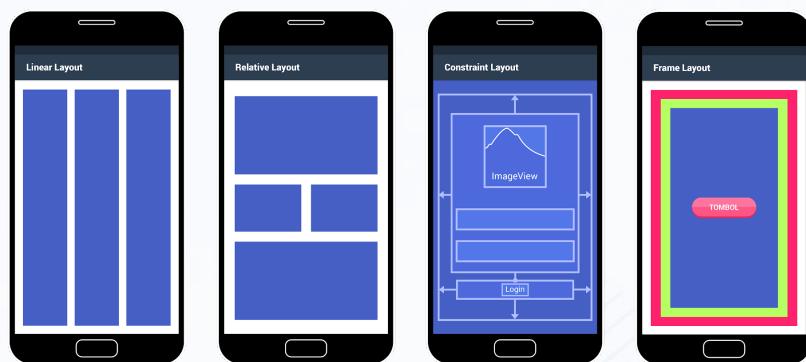
B. View dan ViewGroup

Objek View yang biasanya disebut “widget” dapat berupa salah satu dari banyak subclass seperti Button, atau TextView. Objek ViewGroup yang

biasanya disebut “tata letak” dapat berupa salah satu dari banyak jenis yang menyediakan berbagai struktur tata letak.

Berikut ini merupakan beberapa View yang ada di Android :

- **TextView**: menampilkan text
- **ImageView**: menampilkan gambar
- **Button**: menampilkan tombol
- **RadioButton** : Menampilkan pilihan yang harus dipilih salah satu
- **EditText**: menampilkan text yang bisa diubah (input)



Sementara itu gambar di atas merupakan beberapa contoh view group atau kita dapat menyebutnya layout, dan inilah pengertiannya :

- **Linear Layout** : Layout yang menyajarkan semua child view-nya dalam satu arah, secara vertikal atau horizontal.
- **Relative Layout** : Layout yang penataan nya ini adalah penataan yang menempatkan widget-widget di dalamnya seperti layer, sehingga sebuah widget dapat berada di atas/di bawah widget lainnya atau dengan kata lain Relative merupakan layout yang penataannya lebih bebas (Relative) sehingga bisa di tata di mana saja.
- **Constraint Layout** : Sebenarnya Constraint Layout mirip dengan Relative Layout, karena letak View bergantung pada View lain dalam satu Layout

ataupun dengan Parent Layoutnya. Contohnya di Relative Layout kita bisa meletakkan sebuah View di atas, bawah, atau samping View lain. Kita juga dapat mengatur posisinya berdasarkan Parent Layout menggunakan tag seperti centerVertical, alignParentBottom, dll. Akan tetapi, Constraint Layout jauh lebih fleksibel dan lebih mudah digunakan di Layout Editor.

- **Frame Layout** : Layout yang biasanya digunakan untuk membuat objek yang saling bertindihan contohnya yaitu kita membuat button di atas image.

C. Struktur XML untuk Layouting di Android

```
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="Hello World!" />  
  
</LinearLayout>
```

Gambar di atas merupakan struktur layout UI Android sederhana yang terdapat sebuah TextView yang berfungsi menampilkan teks dan TextView tersebut berapa di dalam sebuah layout yang bernama LinearLayout. Untuk menunjukkan di mana suatu elemen dimulai dan diakhiri, kita menulis tag. Tag mudah dikenali karena selalu diawali dan diakhiri dengan karakter < dan >. Tag juga berisi nama elemen (yaitu LinearLayout) yang awal dan akhirnya ditandai. Elemen sering kali terdiri dari sepasang tag, ditambah semua konten di antaranya.

Dalam hal ini, tag kedua dari pasangan dimulai dengan karakter </ dan kita mengatakan bahwa tag kedua menutup yang pertama. Elemen yang tidak

perlu menyertakan konten apa pun dapat terdiri dari satu tag. Dalam hal ini, tag diakhiri dengan karakter /> dan kita katakan bahwa tag tersebut adalah tag penutup sendiri.

D. UI Design Pattern Android

User Interface (UI) adalah segala bentuk visual yang memungkinkan user (pengguna) dapat berinteraksi dengan suatu produk atau layanan. UI berfokus pada keindahan dari sebuah tampilan, pemilihan warna yang baik dan hal-hal lain yang membuat tampilan menjadi lebih menarik dipandang dan membuat user nyaman sehingga dapat sering menggunakan aplikasi tersebut. Tujuan UI yaitu membuat interaksi sesederhana dan seefisien mungkin dalam hal mencapai tujuan user.

E. Golden Rules of Interface Design

8 Golden Rules of Interface Design merupakan sebuah “aturan” yang perlu diperhatikan dalam membuat desain interface. Aturan Emas ini di propose oleh Ben Schneiderman pada 21 Agustus 1947 dalam bukunya yang berjudul *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Berikut ini merupakan 8 Golden Rules of Interface Design :

Rule 1 : Strive for Consistency

Konsistensi dalam desain baik itu dari sisi layouting (tata letak) maupun tema warna dan tulisan dalam aplikasi merupakan salah satu aspek yang penting dalam perancangan desain aplikasi. Hal ini bertujuan agar user dapat dengan mudah menemukan informasi serta mengetahui navigasi halaman dalam

aplikasi. Pentingnya guideline style / theme aplikasi seperti untuk warna dasar aplikasi (Livin : Biru & Kuning), style tulisan pada aplikasi (ada perbedaan antara judul fitur dan deskripsi), dll. Agar user dapat segera menemukan dan memahami informasi yang ada di halaman tersebut.

Rule 2 : Enable Frequent Users to Use Shortcuts

Adanya shortcut untuk sebuah task atau fitur penting dalam aplikasi sangat membantu user terutama bagi yang rutin menggunakan fitur tersebut karena dapat memangkas waktu penggunaan. Contohnya yaitu shortcut untuk top up e-money pada halaman depan aplikasi Livin. User dapat langsung segera mengisi saldo kartu e-money mereka dari halaman depan aplikasi Livin tanpa login dan membuka banyak halaman terlebih dahulu pada aplikasi.

Rule 3 : Offer Informative Feedback

Pemberian informasi mengenai apa yang sedang diproses atau terjadi pada aplikasi dapat membuat user dapat langsung memahami bahwa aksi yang telah dilakukan sebelumnya telah direspon oleh aplikasi. Untuk memberikan informasi mengenai apa yang telah atau sedang dilakukan oleh user, diperlukan sebuah feedback yang dapat terlihat jelas secara visual dan memiliki makna yang baik. Contoh : Saat aplikasi sedang memproses task (load data), muncul loading atau shimmer yang menandakan bahwa proses sedang berlangsung sehingga user tidak kebingungan alih-alih hanya menampilkan halaman kosong saat load data.

Rule 4 : Design Dialog to Yield Feedback

Desain aplikasi yang baik adalah yang tidak ambigu dengan memberikan informasi seperti dialog yang menyimpulkan akhir dari proses yang dilakukan oleh user di aplikasi. Hal ini sangat diperlukan untuk menjelaskan flow kepada user sehingga user dapat menyadari bahwa proses sudah selesai dan tidak ada proses lanjutan lagi. Contoh : setelah sukses melakukan transaksi transfer uang, ada halaman yang memberitahu bahwa proses transaksi berhasil berserta rincian data transaksinya.

Rule 5 : Offer Simple Error Handling

Bila ada error dalam aplikasi, diperlukan pemberian informasi mengenai error apa yang terjadi beserta informasi mengenai cara penyelesaiannya. Contohnya yaitu adanya pesan yang menginformasikan bahwa aplikasi gagal mengambil data karena tidak ada koneksi internet pada device user.

Rule 6 : Permit Easy Reversal of Actions

Dalam aplikasi, user bisa saja melakukan kesalahan. Maka user dapat melakukan pembatalan ataupun perbaikan kesalahan tersebut yang mudah diakses dan dilakukan oleh user. Contohnya yaitu saat melakukan pembelian barang di e-commerce, user ingin membatalkan pemesanan maka dapat segera dilakukan pada aplikasi. Sehingga user tidak perlu menghubungi customer service. Hal ini tentu akan membuat user semakin nyaman dengan aplikasi.

Rule 7 : Support Internal Locus of Control

Perlunya penyesuaian preferensi user di aplikasi sehingga user dapat mengatur tampilan pada fitur atau aplikasi sesuai keinginannya. Hal ini bertujuan agar kepuasan user dalam menggunakan aplikasi dapat meningkat. Contoh : Pada aplikasi Livin, terdapat fitur quick pin (fitur yang dapat langsung melakukan transaksi tanpa repot isi detail transaksinya lagi) pada halaman utama aplikasi. Fitur ini dapat diatur urutan transaksinya sesuai keinginan user.

Rule 8 : Reduce Short-term Memory of Load

Dalam interface yang baik, informasi visual sangatlah penting karena user dapat dengan nyaman menggunakan aplikasi tanpa harus mengingat-ingat flow yang perlu ia lakukan dengan cara menghafal bagian-bagian layar. Di dalam aplikasi harus terdapat bantuan informasi visual berupa informasi dalam teks dan juga dengan bantuan ikon unik & mudah diingat untuk aplikasi yang membantu user untuk lebih memahami aplikasi.

F. Architecture Design Pattern pada Android

Dalam konteks pengembangan aplikasi, jika kita mengembangkan sebuah aplikasi dalam struktur yang terorganisir, mengikuti seperangkat aturan, mendeskripsikan fungsi yang jelas serta mengimplementasikannya dengan protokol yang sesuai, maka itulah sebuah arsitektur. Untuk memberikan alur data yang jelas dan menghasilkan aplikasi yang andal, minim bug, mudah dipahami, mudah dimodifikasi serta dapat meningkatkan produktivitas

programmer, kita perlu memilih arsitektur yang sesuai dan cocok untuk dipakai di sebuah tim.

Software Architectural Design pattern mempromosikan pemrograman yang terorganisir dengan baik. Memisahkan antar fungsi aplikasi membuatnya mudah untuk diuji dan membuat pemeliharaan menjadi lebih sederhana. MVC, MVP, MVVM merupakan sebagian pola arsitektur yang paling populer untuk programmer Android.

1. MVC

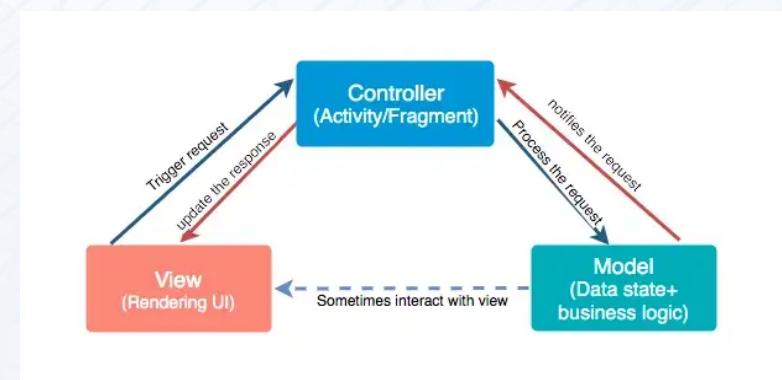
Singkatan dari **Model-View-Controller**, merupakan arsitektur paling umum. Berikut komponen-komponen yang dipakai di MVC :

Model- Menentukan jenis data dan business logic. Bagaimana mengambil dan manipulasi data, berkomunikasi dengan controller, berinteraksi dengan database, dan sesekali meng-update view.

View- Merupakan hasil yang dilihat. Dalam pemrograman Android View ini adalah XML-nya. Ia berkomunikasi dengan controller dan sesekali dengan model.

Controller- Adalah Activity/Fragment. Ia yang berkomunikasi dengan View atau Model (bisa dalam bentuk database lokal seperti SQLite/Room maupun sistem dari luar melalui REST API).

Cara kerjanya seperti pada gambar:



Kelebihan MVC :

- Business logic terpisah dari model
- Mendukung teknik asinkron
- Modifikasi yang dilakukan pada salah satu komponen tidak mempengaruhi komponen lain.
- Proses pengembangan lebih cepat

Kekurangan MVC :

- Controller dapat berisi proses yang kelewat besar sehingga sulit dikelola.
- Unit testing relatif lebih merepotkan.
- Kompleksitas aplikasi akan terus meningkat.

2. MVP

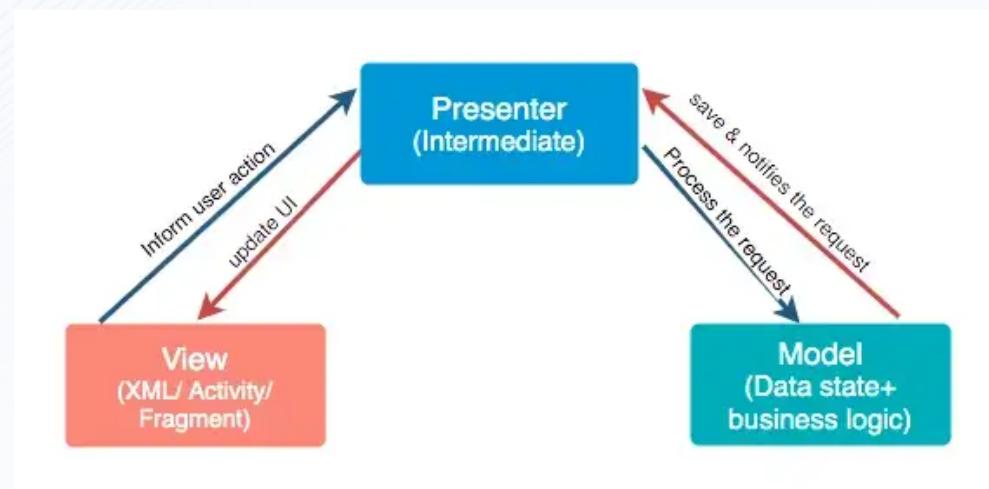
Singkatan dari **Model-View-Presenter**. Dalam proses pengembangan aplikasi, developer harus memanfaatkan waktu yang ada untuk memilah arsitektur ke dalam lapisan-lapisan. Lapisan tersebut akan membagi dependensi yang biasanya terdapat di view. Berikut komponen-komponen yang dipakai di MVP :

Model — Merupakan business logic dan pengelola state suatu data. Mengambil dan memanipulasi data, komunikasi dengan presenter, serta berinteraksi dengan database. Model tidak terhubung langsung dengan view.

View — Terdiri dari komponen UI seperti Activity dan Fragment. Ia hanya berkomunikasi dengan presenter.

Presenter — Sebagai penghubung antara model dengan view. Ia mengatur apa saja yang akan ditampilkan ke view, ia yang memberitahu view apa yang mesti muncul. Ia juga yang akan mengirimkan data yang perlu disimpan lewat model.

Cara kerjanya seperti pada gambar:



Kelebihan MVP :

- View tidak tahu apa-apa soal apa yang ia tampilkan sehingga layer view bisa diganti dengan lebih mudah.
- Komponen View dan Presenter yang bisa dipakai ulang (reuse)
- Kode lebih mudah dipahami dan dikelola
- Pengujian lebih mudah karena business logic dipisah dari UI

Kekurangan MVP :

- View dan Presenter memiliki hubungan terlalu erat
- Banyak interface dibutuhkan untuk komunikasi antar layer
- Ukuran kode bisa menjadi besar

3. MVVM

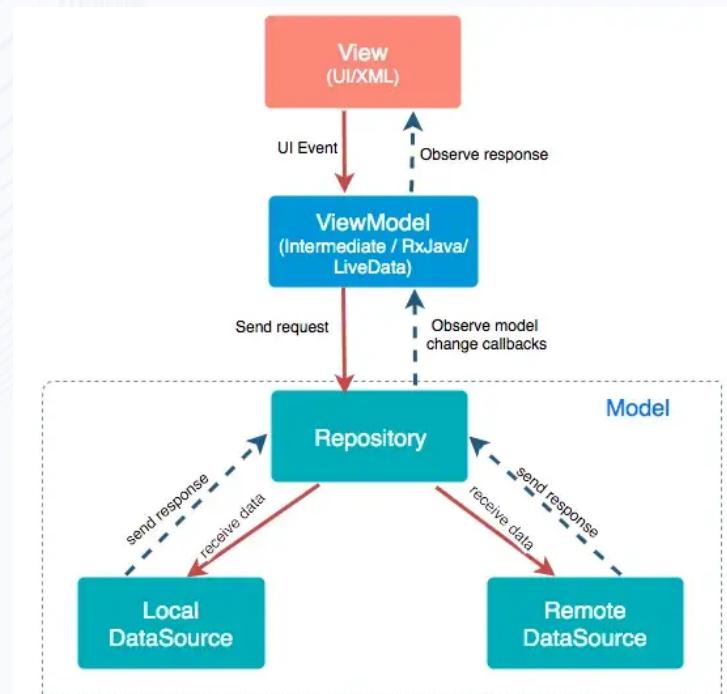
Singkatan dari **Model-View-ViewModel**. Ia melonggarkan hubungan antar komponen. Ia juga bekerja dengan konsep observable. Dari satu komponen ke komponen lain tidak terhubung secara langsung melainkan lewat reference yang terdapat di-observable. Berikut komponen-komponen yang dipakai di MVVM :

Model — Memiliki kode untuk business logic serta repositori. Tugas repositori untuk menjembatani request dari ViewModel ke sumber data lokal atau remote.

View — Tidak ada business logic, hanya tampilan yang terhubung dengan user (XML). Aksi user dikirim ke view model tapi tidak akan langsung mendapatkan respons. Untuk mendapatkan responsnya, View akan terima dari ViewModel.

ViewModel — Mayoritas proses yang ada di UI terjadi di komponen ini. Ia tidak perlu tahu view mana yang memanggilnya. Ia hanya memberikan data yang dibutuhkan kepada siapapun yang menghubunginya melalui observable.

Cara kerjanya seperti pada gambar:



Kelebihan MVVM :

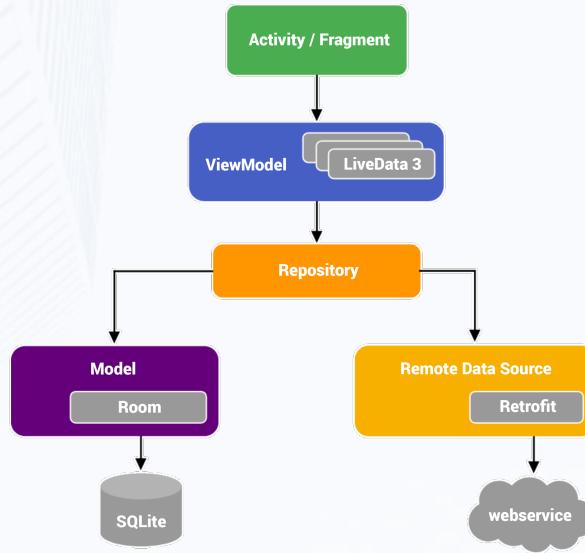
- Tidak ada hubungan erat antar view dan view model
- Tidak ada interface antara view dan model
- Mudah untuk menulis unit testing dan kodenya pun event-driven.

Kekurangan MVVM :

- Harus membuat observable untuk setiap komponen UI
- Kode yang ditulis bisa banyak

G. Deep Dive MVVM pada Android

Berikut ini merupakan gambaran lebih detail mengenai arsitektur pattern MVVM yang banyak diimplementasikan oleh para developer Android :



- **View** (Activity/Fragment) : UI controller atau Fragment dan Activity yang berfungsi menampilkan data atau melakukan aksi di UI.
- **ViewModel**: Meneruskan data ke UI. Bertindak sebagai pusat komunikasi antara Repotori dan UI. Fungsi ViewModel menyimpan data untuk perubahan konfigurasi.
- **LiveData**: Kelas yang dapat diamati data holder. Selalu menyimpan data versi terbaru. Memberitahu observer ketika data telah berubah. Komponen UI hanya mengamati atau mengamati data yang relevan dan tidak akan menghentikan atau melanjutkan pengamatan.
- **Repositori**: Kita menggunakan Repotori untuk mengelola beberapa sumber data, seperti network, lokal, atau cache.
- **Room** (atau DB lokal lainnya) : Sumber data dari database lokal. Room adalah library yang menyediakan lapisan abstraksi di atas SQLite untuk memungkinkan akses database yang lebih ketat dengan memanfaatkan kekuatan SQLite.
- **Rest Client**: Sumber data yang berasal dari jaringan, seperti Rest API, Firebase, atau data klien lainnya.

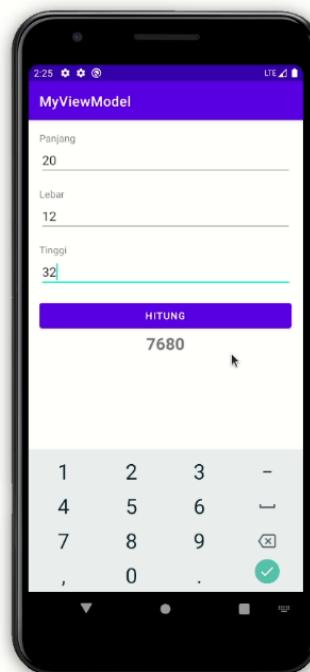
ViewModel adalah komponen lifecycle-aware Android. Mereka secara otomatis dihapus ketika Lifecycle yang mereka amati dihancurkan secara permanen.

Dirancang untuk menyimpan dan mengelola data terkait UI dengan cara sadar siklus hidup. ViewModel biasanya menyimpan status data tampilan dan berkomunikasi dengan komponen lain, seperti repositori data atau lapisan domain yang menangani logika bisnis. ViewModel bertahan dari perubahan konfigurasi seperti rotasi layar.

Untuk mengimplementasikan MVVM pada project Android, kita dapat menambahkan dependensi berikut pada file gradle project kita :

```
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:x.x.x"
```

Berikut ini merupakan contoh implementasi MVVM untuk aplikasi kalkulator sederhana seperti pada tampilan aplikasi di bawah ini :



Pada tampilan aplikasi di atas, terdapat 3 buah edit text yang berfungsi untuk input data panjang, lebar, dan tinggi. Setelah user mengklik button hitung, maka hasil perhitungan dari ketiga data tersebut akan ditampilkan di bawah button hitung. Jika menggunakan MVVM, maka pertama kita harus membuat class model terlebih dahulu sebagai berikut :

MainModel.kt

```
class MainModel {  
    fun getVolume(length:Double, width:Double, height:Double): Double {  
        return length * width * height  
    }  
}
```

Kode di atas merupakan class model yang bernama MainModel. Terdapat satu buah method yang bernama getVolume yang berfungsi untuk melakukan perhitungan volume dan mengembalikan nilai volume tersebut ke method yang memanggilnya (nantinya dipanggil oleh ViewModel). Setelah itu, kita membuat ViewModel sebagai berikut :

MainViewModel.kt

```
class MainViewModel : ViewModel() {  
    var result = 0.0  
    var mainModel = MainModel()  
  
    fun calculate(width: Double, height: Double, length: Double) {  
        result = mainModel.getVolume(width, height, length)  
    }  
}
```

Kode di atas merupakan ViewModel yang berfungsi untuk menghandle data volume yang didapat dari model dan akan ditampilkan ke UI. Maka, kode pada view atau activity akan seperti berikut :

MainActivity.kt

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        ...  
        val viewModel=ViewModelProvider(this).get(MainViewModel::class.java)  
        tvResult.text = viewModel.result.toString()  
        btnCalculate.setOnClickListener {  
            val length = edtLength.text.toString().toDouble()  
            val width = edtWidth.text.toString().toDouble()  
            val height = edtHeight.text.toString().toDouble()  
            viewModel.calculate(width, height, length)  
            //no business logic in Activity (DO!)  
        }  
    }  
}
```

Dapat terlihat pada kode dalam class MainActivity di atas bahwa di class tersebut hanya mengambil nilai dari edit text kemudian memanggil ViewModel untuk melakukan proses perhitungan volume sehingga kode pada activity atau

UI menjadi terpisah dan tidak tergantung dengan business logic (perhitungan volume).

References

[https://sis.binus.ac.id/2022/02/22/xml-pada-pengembangan-aplikasi-andr
oid/](https://sis.binus.ac.id/2022/02/22/xml-pada-pengembangan-aplikasi-android/)

[https://medium.com/temancatat/the-8-golden-rules-of-interface-design-2
d40f639e3e2](https://medium.com/temancatat/the-8-golden-rules-of-interface-design-2d40f639e3e2)

[https://lobothijau.medium.com/arsitektur-mvc-vs-mvp-vs-mvvm-di-pemro
graman-android-387d9c99e893](https://lobothijau.medium.com/arsitektur-mvc-vs-mvp-vs-mvvm-di-pemrograman-android-387d9c99e893)

<https://developer.android.com/topic/libraries/architecture/viewmodel>