KISI-KISI MATERI UNTUK MENJAWAB UJIAN TENGAH SEMESTER

Layouting eXtensible Markup Language (XML) adalah suatu fondasi utama untuk membangun antarmuka pengguna/*User Interface* (UI) yang efektif dan responsif. XML pada Android digunakan sebagai **metode deklaratif** untuk mendefinisikan UI. Prinsip utama dalam XML akan memisahkan bagian tampilan (presentasi) dengan logika (prilaku).

a. XML Layout (.xml) menentukan tampilan (*apa saja tombolnya atau dimana teksnya*)
b. Kode Kotlin/Java (.kt/.java) menentukan prilaku (*apa yang terjadi ketika tombol di-klik*)

1. Komponen Fundamental
   a. View: komponen UI dasar yang dilihat dan diperintahkan untuk berinteraksi dengan pengguna
      Contoh:
      - TextView: Menampilkan teks
      - ImageView: Menampilkan Gambar
      - Button: Tombol yang dapat di-klik
      - EditText: Kotak/Wadah untuk teks
   b. ViewGroup: kontainer yang tidak terlihat dan bertugas menampung serta mengatur view lain. ViewGroup adalah View Khusus yang berprilaku sebagai induk (parent) untuk View (children).
      Contoh:
      - ContraintLayout: Mengatur View anak menggunakan contraints yang fleksibel (sangat direkomendasikan)
      - LinearLayout: Mengatur View anak dalam satu baris (horizontal atau vertikal)
      - FrameLayout: Menumpuk View anak di atas satu sama lain
2. Prinsip Fundamental (Aturan Main) adalah atribut-atribut XML yang digunakan untuk mengontrol komponen di atas.
   a. Prinsip Ukuran: layout_width & layout_height
      Setiap View dan ViewGroup **wajib** memiliki atribut ini.
      - match_parent: View akan meregang untuk mengisi semua ruang yang tersedia di dalam induknya (ViewGroup).
      - wrap_content: View akan menyusut seukuran konten di dalamnya (misalnya, pas seukuran teks atau gambar).
      - Nilai Tetap (misal 100dp): View akan memiliki ukuran pasti, terlepas dari konten atau induknya.

b. Prinsip Spasi: padding vs margin: sangat penting untuk mengatur jarak.
   - padding (Bantalan Internal): Ruang di dalam batas View, antara batas dan kontennya.
   - margin (Jarak Eksternal): Ruang di luar batas View, menciptakan jarak ke elemen lain di sekitarnya.

c. Prinsip Identifikasi: android:id, adalah "nama" unik untuk sebuah View.
   - Prinsip: id adalah "jembatan" yang memungkinkan kode Kotlin/Java Anda menemukan (findViewById atau View Binding) dan memanipulasi View tersebut.
   - Sintaks: android:id="@+id/my_button" (tanda + berarti "buat ID baru ini")

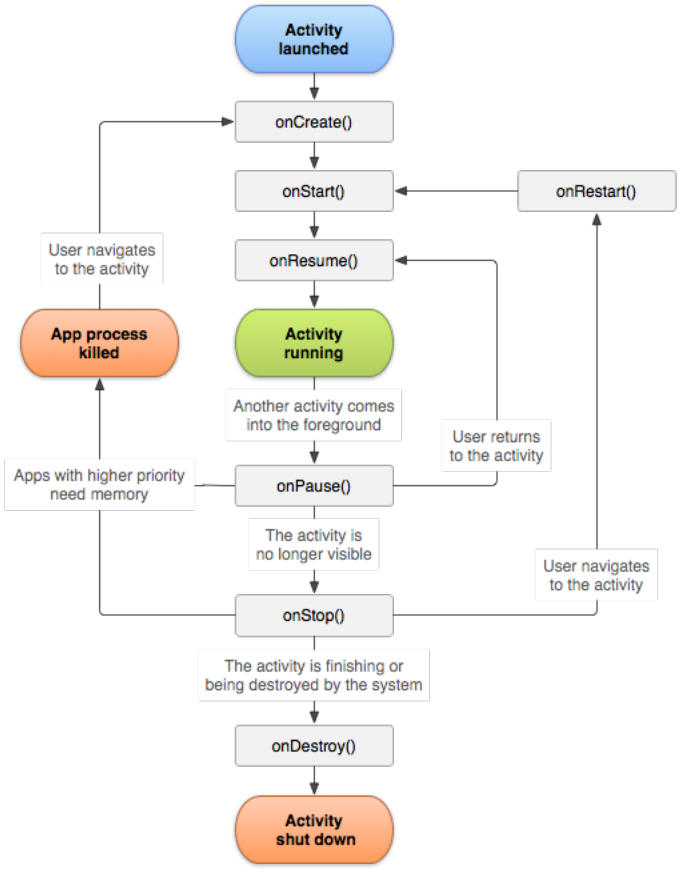
d. Prinsip Unit Responsif: dp & sp, adalah kunci untuk membuat aplikasi yang terlihat bagus di semua ukuran layar.
   - dp (Density-independent Pixel): Gunakan ini untuk semua ukuran dan spasi (layout_width, margin, padding). Ini adalah unit fleksibel yang diskalakan berdasarkan kepadatan piksel layar.
   - sp (Scale-independent Pixel): Gunakan ini hanya untuk ukuran font (textSize). Ini mirip dp, tetapi juga menghormati pengaturan ukuran font yang dipilih pengguna di sistem mereka (penting untuk aksesibilitas).

e. Prinsip Abstraksi Resource adalah *best practice* untuk kode yang bersih dan mudah dikelola.
   - Prinsip: Jangan pernah menulis teks, warna, atau ukuran secara langsung di file layout (hardcoding). Selalu simpan di file resource terpisah.
   - Contoh:

     Buruk: android:text="Login"

     Baik: android:text="@string/login_button_text"

Konsep siklus proses aktivitas atau Activity Lifecycle adalah konsep paling fundamental dalam pengembangan Android. *Activity Lifecycle* adalah serangkaian status yang dilewati oleh sebuah Activity (satu layar aplikasi) ketika pertama kali dibuat, menjadi aktif, dijeda, hingga akhirnya dihancurkan. Ilustrasi dari activity lifecycle sebagai berikut:



a. **onCreate()**: Dipanggil saat layar *pertama kali dibuat*. Tempat Anda menyiapkan layout (XML) dan inisialisasi data.
b. **onStart()**: Dipanggil saat layar akan *terlihat* oleh pengguna.
c. **onResume()**: Dipanggil saat layar berada di *latar depan* dan pengguna dapat berinteraksi dengannya.
d. **onPause()**: Dipanggil saat layar *kehilangan fokus* (misal: ada pop-up atau panggilan telepon) tapi mungkin masih terlihat.
e. **onStop()**: Dipanggil saat layar *tidak lagi terlihat* oleh pengguna (misal: pengguna menekan tombol Home atau pindah ke layar lain).
f. **onDestroy()**: Dipanggil tepat sebelum layar *dihancurkan* (misal: pengguna menutup aplikasi atau menekan tombol kembali).

g. **onRestart()**: Dipanggil saat Activity yang tadinya onStop() (di background) ingin kembali ditampilkan ke pengguna. Setelah onRestart(), metode onStart() akan dipanggil.

h. **onSaveInstanceState(Bundle outState)**: Dipanggil oleh sistem sebelum onStop() jika Activity berpotensi dihancurkan (misalnya saat rotasi layar). Ini adalah tempat Anda menyimpan data UI sementara (seperti teks yang diketik pengguna) ke dalam Bundle. Data ini bisa diambil kembali nanti di onCreate().

Memahami *lifecycle* adalah wajib karena tiga alasan utama:

a. **Menghindari Boros Baterai & Resource:** Anda tidak ingin mengambil data lokasi GPS (onResume) saat aplikasi Anda sedang tidak dilihat pengguna (onStop). Ini memboroskan baterai.

b. **Menghindari *Crash* dan *Memory Leaks*:** Jika Anda memulai proses di onCreate tapi lupa menghentikannya di onDestroy, aplikasi Anda bisa mengalami *memory leak* (kebocoran memori) dan akhirnya *crash*.

c. **Pengalaman Pengguna (UX) yang Baik:** Pengguna berharap aplikasi Anda:
   - Tidak kehilangan data (menyimpan *draft* di onPause).
   - Berhenti memutar musik saat ada panggilan telepon (onPause).
   - Melanjutkan proses *download* di *background* (onStop).

Contoh kode program:

```
override fun onCreate(savedInstanceState: Bundle?) {                // (a)
    super.onCreate(savedInstanceState)                              // (b)
    setContentView(R.layout.activity_main)
    Log.d(TAG, "onCreate: Activity Dibuat. Siap untuk inisialisasi.")
}

override fun onStart() {                                            // (c)
    super.onStart()                                                 // (d)
    Log.d(TAG, "onStart: Media DISIAPKAN (Loaded).")
}

override fun onResume() {                                           // (e)
    super.onResume()                                                // (f)
    if (!isMediaPlaying) {
        Log.i(TAG, "onResume: Activity Aktif. Media DILANJUTKAN.")
    }
}

override fun onPause() {                                            // (g)
    super.onPause()                                                 // (h)
    if (isMediaPlaying) {
        isMediaPlaying = false
        Log.w(TAG, "onPause: Activity Dijeda. Media DIJEDA (Paused).")
    }
}

override fun onStop() {                                             // (i)
    super.onStop()                                                  // (j)
    Log.d(TAG, "onStop: Media di-release atau sumber daya berat DIBERSIHKAN.")
}
```

Terminologi Konsep M–V–C pada Pengembangan Aplikasi Android

MVC adalah singkatan dari **Model – View – Controller**, yaitu **pola arsitektur**

(software architecture pattern) yang memisahkan aplikasi menjadi tiga bagian utama:

Komponen	Singkatan	Fungsi Utama	Tujuan
Model	*Data & Logic Layer*	Menyimpan data, mengelola logika bisnis, dan berinteraksi dengan database atau API.	Agar data dan logika bisnis terpisah dari tampilan.
View	*UI Layer*	Menampilkan data kepada pengguna dalam bentuk antarmuka (UI).	Agar tampilan bisa diubah tanpa mengubah logika program.
Controller	*Interaction Layer*	Menjadi penghubung antara Model dan View — menerima input dari user dan menentukan tindakan yang harus dilakukan.	Mengatur aliran data dan interaksi antara UI dan data.

Dalam konteks **Android**, penerapan MVC bertujuan agar aplikasi:

a. Lebih **terstruktur dan mudah dirawat**.

b. Lebih **mudah diuji (testable)**.

c. Dapat **memisahkan tanggung jawab (separation of concerns)** antar bagian.

Peran Komponen MVC dalam Pengembangan Aplikasi Android

Berikut penjelasan rinci peran tiap komponen dalam konteks Android:

1. Model (M) — Logika Bisnis & Data

Peran:

a. Bertanggung jawab atas **pengelolaan data** aplikasi.

b. Menyediakan **fungsi logika bisnis**, seperti perhitungan, validasi, atau interaksi dengan sumber data (database, API, repository).

Dalam Android, Model dapat berupa:

a. **Class Repository** (misalnya UserRepository, ProductRepository).

b. **Room Database / SQLite / Realm** (penyimpanan data lokal).

c. **Data Model Class** (misalnya data class User(val id: Int, val name: String)).

d. **Network Layer (Retrofit, Volley, dsb.)** untuk komunikasi API.

Contoh sederhana:

```
data class User(val id: Int, val name: String)

class UserRepository {
    fun getUser(): User {
        return User(1, "Dinan")
    }
}
```

2. View (V) — Tampilan Antarmuka Pengguna

Peran:

a. Menampilkan data dari Model kepada pengguna.

b. Menerima input dari pengguna (misalnya klik tombol, mengetik teks).

c. Tidak memiliki logika bisnis — hanya bertanggung jawab untuk **UI**.

Dalam Android, View dapat berupa:

a. **XML Layout Files** (activity_main.xml, fragment_home.xml).

b. **Widgets/UI Components** seperti TextView, Button, RecyclerView.

c. Bisa juga **View Binding** atau **Jetpack Compose UI**.

Contoh:

```
<!-- activity_main.xml -->
<LinearLayout ... >
    <TextView
        android:id="@+id/tvUsername"
        android:text="Nama Pengguna"/>
    <Button
        android:id="@+id/btnLoadUser"
        android:text="Tampilkan User"/>
</LinearLayout>
```

3. Controller (C) — Pengendali & Penghubung

Peran:

a. Menjadi **penghubung antara View dan Model**.

b. Mengatur aliran data: menerima input dari pengguna, memprosesnya dengan bantuan Model, lalu mengupdate View.

c. Mengatur **lifecycle**, event handling, dan navigasi antar layar.

Dalam Android, Controller biasanya berupa:

a. **Activity** (misalnya MainActivity.kt).

b. **Fragment** (misalnya HomeFragment.kt).

c. Kadang juga **ViewModel** (pada arsitektur modern seperti MVVM, tapi masih serupa konsepnya).

Contoh:

```
class MainActivity : AppCompatActivity() {
    private lateinit var repository: UserRepository

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

        repository = UserRepository()
        val user = repository.getUser()

        val tvUsername = findViewById<TextView>(R.id.tvUsername)
        tvUsername.text = user.name
    }
}
```

Ringkasan

a. **MVC** membantu memisahkan tanggung jawab:
   - *Model* fokus pada **data dan logika bisnis**,
   - *View* fokus pada **tampilan**,
   - *Controller* fokus pada **pengendalian alur dan interaksi**.

b. Dalam **Android**, pemetaan sederhananya:
   - Model → Repository, Database, Logic Class
   - View → XML Layout
   - Controller → Activity / Fragment

c. Dengan penerapan MVC, aplikasi Android menjadi:
   - Lebih **modular**,
   - **Mudah diuji dan dikembangkan**,
   - Serta **mudah dipelihara** karena perubahan UI tidak memengaruhi logika data.