NAMA : SYAFIEQ MAULANA KHAIRSYAH SANUDIN

NIM : 2300018220

Pola Arsitektur Android

Pola arsitektur Android adalah struktur organisasi kode yang menentukan bagaimana komponen-komponen aplikasi (UI, data, logika) berinteraksi dan terhubung satu sama lain. Pola ini berfungsi sebagai blueprint yang memberikan pedoman dalam mengorganisir kode, memisahkan tanggung jawab, dan mengatur alur data dalam aplikasi. Dengan menerapkan pola arsitektur yang tepat, developer dapat membangun aplikasi yang lebih terstruktur, mudah dikembangkan, dan dapat diuji secara efektif.

1. MVC (Model-View-Controller)

Konsep: Pola tradisional yang memisahkan aplikasi menjadi 3 komponen utama. Cocok untuk aplikasi Android sederhana.

Komponen:

- **Model**: Data, logika bisnis, database
- **View**: XML layout, UI components
- **Controller**: Activity/Fragment, navigasi

Cara Kerja:

1. User berinteraksi dengan View (tampilan UI)
2. Controller (Activity/Fragment) menerima event dari user
3. Controller memproses input dan memanggil Model
4. Model melakukan operasi data/logika bisnis
5. Model mengembalikan hasil ke Controller
6. Controller memperbarui View dengan data baru

Kelebihan:

- Sederhana dan mudah dipahami
- Cocok aplikasi kecil
- Development paralel mungkin

Kekurangan:

- Activity jadi terlalu besar (Massive View Controller)
- Sulit di-test karena Android dependencies
- Tight coupling antara komponen

2. MVP (Model-View-Presenter)

Konsep: Perbaikan dari MVC dengan Presenter sebagai perantara. View menjadi pasif, semua logika ada di Presenter.

Komponen:

- **Model**: Data, logika bisnis
- **View**: Activity + Interface
- **Presenter**: Kelas terpisah, logika presentasi

Cara Kerja:

1. User berinteraksi dengan View (tampilan UI)
2. View meneruskan event ke Presenter melalui interface
3. Presenter memproses input dan memanggil Model
4. Model melakukan operasi data/logika bisnis
5. Model mengembalikan hasil ke Presenter
6. Presenter memformat data dan memperbarui View melalui callback

Kelebihan:

- Presenter mudah di-test (non-Android)
- Separation of concern lebih baik
- View lebih sederhana

Kekurangan:

- Banyak boilerplate code untuk interface
- Manual lifecycle handling
- Overkill untuk proyek kecil

3. MVVM (Model-View-ViewModel)

Konsep: Pola modern dengan pendekatan reactive. Menggunakan observables untuk komunikasi otomatis.

Komponen:

- **Model**: Data, repository pattern
- **View**: Activity/Fragment + Observers
- **ViewModel**: State management, lifecycle-aware

Cara Kerja:

1. User berinteraksi dengan View (tampilan UI)
2. View memanggil method di ViewModel
3. ViewModel memproses input dan memanggil Model
4. Model melakukan operasi data/logika bisnis
5. Model mengembalikan hasil ke ViewModel
6. ViewModel mengupdate LiveData/StateFlow
7. View secara otomatis terupdate melalui observers

Kelebihan:

- UI updates otomatis
- ViewModel bertahan pada configuration changes
- Official Google support (Jetpack)
- Kurang boilerplate dengan data binding

Kekurangan:

- Potensi memory leaks
- Learning curve untuk LiveData/Flow
- ViewModel bisa jadi terlalu besar

4. MVI (Model-View-Intent)

Konsep: Pendekatan unidirectional dengan state management ketat. Semua UI berasal dari state immutable.

Komponen:

- **Model**: Immutable state
- **View**: Render state → UI
- **Intent**: User actions/events
- **State**: Representasi UI

Cara Kerja:

1. User berinteraksi dengan View (tampilan UI)
2. View mengirim Intent (user action) ke ViewModel
3. ViewModel memproses Intent dan menghasilkan State baru
4. State baru dikirim ke View melalui observables
5. View mere-render UI berdasarkan State baru

Kelebihan:

- State yang predictable dan immutable
- Unidirectional data flow
- Thread-safe operations
- Debugging mudah dengan state changes

Kekurangan:

- Banyak boilerplate code
- Overkill untuk aplikasi sederhana
- Learning curve tinggi

5. Clean Architecture

Konsep: Arsitektur berlapis dengan aturan ketat dependency. Fokus pada pemisahan concerns dan independensi dari framework.

Komponen:

- **Domain Layer**: Entities, Use Cases
- **Data Layer**: Repositories, Data Sources
- **Presentation Layer**: ViewModels, UI Components

Cara Kerja:

1. User berinteraksi dengan Presentation Layer (UI)
2. Presentation Layer memanggil Use Cases di Domain Layer
3. Use Cases memanggil Repository interfaces
4. Data Layer mengimplementasi operasi data
5. Data mengalir kembali ke Presentation Layer
6. UI diperbarui dengan data yang sudah diproses

Kelebihan:

- Highly testable setiap layer
- Independent dari framework
- Maintainable untuk aplikasi besar

Kekurangan:

- Over-engineering untuk aplikasi kecil
- Banyak boilerplate code
- Learning curve sangat tinggi

Kesimpulan

Pemilihan pola arsitektur Android yang tepat sangat penting untuk membangun aplikasi yang terstruktur, mudah dikembangkan, dan dapat diuji dengan baik, di mana setiap pola memiliki kelebihan dan kekurangan tersendiri yang perlu disesuaikan dengan kompleksitas proyek, kebutuhan tim, dan skalabilitas aplikasi yang diinginkan.