**1. UI Layout: ConstraintLayout – LinearLayout – RelativeLayout **

**1.1 ConstraintLayout **

Khái niệm

ConstraintLayout là layout hiện đại nhất trong Android, cho phép xây dựng giao diện bằng cách **ràng buộc (constraint)** các view theo vị trí của:

- Parent (khung chứa)
- View khác
- Guideline, Barrier
- Chain (nhóm view liên kết)

Điểm mạnh nhất: → Nó giúp tạo UI phức tạp mà vẫn **ít nested layout**, tối ưu hiệu năng.

📌 Đặc điểm quan trọng

✔ Chỉ cần 1 layout chứa toàn bộ UI phức tạp

Bạn có thể làm giao diện login, dashboard, profile… chỉ bằng 1 ConstraintLayout.

✔ Cơ chế "Constraint"

Mỗi view phải có **ít nhất 2 constraint**: theo trục ngang + dọc.

Ví dụ constraint trái – phải:

```
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"
```

✔ Bias

Cho phép căn chỉnh không cân đối (ví dụ 30% – 70%)

```
app:layout_constraintHorizontal_bias="0.3"
```

✔ Chain

Liên kết một nhóm view → canh đều / pack / spread…

✔ **Guideline**

Tạo đường tham chiếu giúp căn UI cực đẹp.

✔ **Barrier**

Tự động dời view theo vị trí view khác (rất mạnh khi UI động).

📌 Khi nào dùng ConstraintLayout?

- Khi UI phức tạp
- Khi muốn hạn chế nested layout
- Khi cần UI responsive trên nhiều kích thước màn hình
- Khi cần animation với MotionLayout

> "ConstraintLayout là layout mạnh nhất hiện nay của Android. Nó cho phép sử dụng constraint để định vị view mà không cần lồng nhiều layout, nhờ đó hiệu năng tốt hơn. Các tính năng như chain, guideline, barrier giúp tối ưu UI phức tạp."

**1.2 LinearLayout **

Khái niệm

LinearLayout là layout đơn giản xếp các view theo **một chiều duy nhất**:

- Chiều dọc (vertical)
- Chiều ngang (horizontal)

Nó giống như 1 "stack" UI.

📌 Feature chính

✔ **orientation:**

```
android:orientation="vertical"
```

✔ **weight – chia đều chiều rộng/chiều cao**

```
android:layout_weight="1"
```

✔ **Dễ dùng – phù hợp UI đơn giản**

Nếu bạn chỉ cần xếp 2–3 item theo hàng/dòng → LinearLayout là tốt nhất.

📌 Nhược điểm

- Nếu nest nhiều lớp → hierarchy sâu → UI render chậm
- Không phù hợp UI phức tạp
- Thiếu sự linh hoạt về vị trí view (khó căn giữa, khó canh theo view khác)

> "LinearLayout dễ dùng nhất nhưng nếu lồng nhiều layout sẽ dẫn đến deep view hierarchy, ảnh hưởng hiệu năng."

1.3 RelativeLayout (chi tiết)

Khái niệm

RelativeLayout cho phép đặt view **tương đối** theo:

- Parent
- Một view khác

VD: "TextView A nằm dưới Button B"

📌 Ví dụ: đặt view dưới 1 view khác

```
android:layout_below="@id/btnLogin"
```

📌 Ưu điểm

- Linh hoạt hơn LinearLayout
- Dễ tạo UI trung bình (không quá phức tạp)

📌 Nhược điểm

- Từ khi có ConstraintLayout → RelativeLayout **bị thay thế**
- UI phức tạp sẽ rất khó maintain
- Không tối ưu bằng ConstraintLayout

✔ Cách nói phỏng vấn

> "RelativeLayout từng rất phổ biến nhưng hiện tại đã được thay thế bởi ConstraintLayout vì thiếu các công cụ mạnh như chain, guideline và performance chưa tối ưu bằng ConstraintLayout."

---

2. RecyclerView – Adapter – ViewHolder – DiffUtil – ListAdapter

2.1 RecyclerView

Khái niệm

RecyclerView là component dùng hiển thị danh sách lớn. Điểm mạnh nhất:

- **Tái sử dụng ViewHolder** → tránh tạo view mới liên tục
- **Tách riêng logic hiển thị (Adapter)**
- Dễ mở rộng với:
    - Divider
    - ItemDecoration
    - LayoutManager
    - DiffUtil
    - Pagination

> "RecyclerView tối ưu nhất trong việc hiển thị list lớn vì nó tái sử dụng ViewHolder, giảm tải cho hệ thống thay vì tạo mới view."

2.2 ViewHolder

Khái niệm

ViewHolder là class chứa các view của 1 item để tái sử dụng. Giúp tránh gọi `findViewById` nhiều lần.

2.3 Adapter (nơi bind data)

Khái niệm

Adapter là lớp quản lý:

- Tạo ViewHolder
- Bind data vào ViewHolder
- Quản lý số lượng item

Ví dụ Adapter (Kotlin)

```
class UserAdapter : RecyclerView.Adapter<UserAdapter.UserVH>() {

    private val items = mutableListOf<String>()

    inner class UserVH(val binding: ItemUserBinding) :
        RecyclerView.ViewHolder(binding.root)

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): UserVH {
        val binding = ItemUserBinding.inflate(
            LayoutInflater.from(parent.context),
            parent,
            false
        )
        return UserVH(binding)
    }

    override fun onBindViewHolder(holder: UserVH, position: Int) {
        holder.binding.tvName.text = items[position]
    }

    override fun getItemCount() = items.size
}
```

2.4 DiffUtil

Khái niệm

DiffUtil tính toán sự khác nhau giữa 2 list để update:

- insert
- remove
- update

→ Không refresh toàn bộ list (good performance).

2.5 ListAdapter (Modern)

Khái niệm

ListAdapter = RecyclerView.Adapter + DiffUtil tự động.

➡️ Đây là cách dùng hiện đại nhất.

> "Em ưu tiên dùng ListAdapter vì DiffUtil tự động giúp update list mượt hơn, đúng chuẩn Android hiện đại."

**3. ViewBinding vs DataBinding **

**3.1 ViewBinding **

Khái niệm

ViewBinding tạo ra 1 class tương ứng với mỗi XML → giúp truy cập view **type-safe**.

Ưu điểm:

- Nhanh
- Không lỗi null
- Nhẹ
- Không cần @{} như DataBinding

Ví dụ:

```
val binding = ActivityMainBinding.inflate(layoutInflater)
binding.tvTitle.text = "Hello"
```

3.2 DataBinding

Khái niệm

DataBinding cho phép **bind logic vào XML** bằng expression.

VD bind text trực tiếp:

```
android:text="@{user.name}"
```

Ưu điểm:

- Hỗ trợ MVVM
- Observable fields
- Two-way binding

Nhược điểm:

- Build chậm
- XML khó debug

> "ViewBinding dùng để findView an toàn và nhanh, còn DataBinding dùng khi cần binding logic theo MVVM trong XML."

4. Custom View (onMeasure & onDraw)

4.1 Custom View là gì?

Custom View = bạn tự tạo ra một loại View mới bằng cách:

- vẽ
- tính kích thước
- xử lý touch event
- tạo UI đặc biệt

4.2 onMeasure()

Chức năng:

Xác định kích thước view (width, height).

Khi override:

- View cần hình dạng đặc biệt (hình vuông/tròn)
- Tính size theo dữ liệu (progress bar)

Ví dụ View vuông

```
override fun onMeasure(w: Int, h: Int) {
    val size = MeasureSpec.getSize(w)
    setMeasuredDimension(size, size)
}
```

**4.3 onDraw() **

Chức năng:

Dùng để **vẽ** lên canvas:

- hình tròn

- biểu đồ
- progress bar custom
- hiệu ứng đồ họa

Ví dụ vẽ hình tròn

```
override fun onDraw(canvas: Canvas) {
    val paint = Paint().apply {
        color = Color.BLUE
    }
    canvas.drawCircle(width / 2f, height / 2f, 100f, paint)
}
```

> “CustomView được sử dụng khi UI cần được vẽ thủ công trên Canvas hoặc cần hiệu năng cao. onMeasure xử lý kích thước, còn onDraw xử lý việc vẽ.”
