Mobile Computing: Android layouts

Aniket Mishra

31st October 2024

1 LinearLayout (Linear Activity)

Purpose: Aligns child views in a single direction, either vertically or horizontally.

Typical Use: Ideal for simple layouts where you want to arrange views in a line, such as a form with labels and input fields.

Key Features:

- Can orient children either vertically or horizontally using `android:orientation`.
- Supports layout weights, allowing views to take proportional screen space.
- Efficient for stacking a small number of views but can become performance-heavy if deeply nested.
- Helps create intuitive, sequential layouts like settings lists or form inputs.

Example:

Listing 1: LinearLayout XML example

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Username"/>

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

2 WebView Activity

Purpose: Integrates a web browser into an app, allowing access to web pages within the app itself without redirecting the user to an external browser.

Typical use: Useful for apps that display articles, documentation, or web content.

Key features:

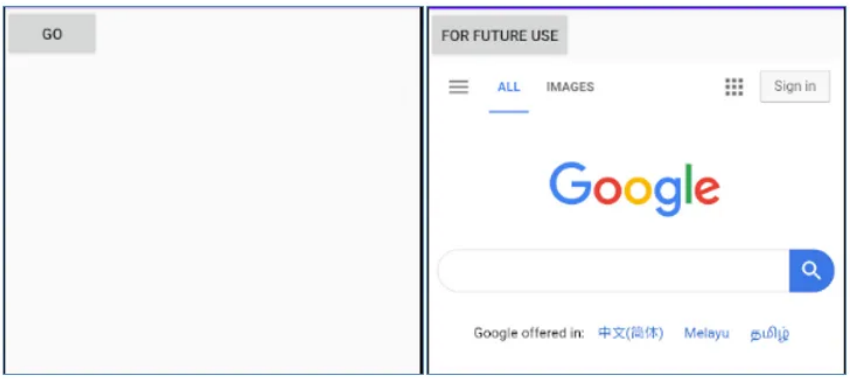
- Supports JavaScript and other web technologies (requires enabling JavaScript explicitly for security reasons).
- Can load both local content (assets) and remote URLs.
- Customizable: you can intercept URL loading, add custom HTML, or enable caching.
- Allows interaction between JavaScript and native Android code through WebView's addJavascriptInterface.

Example:

Listing 2: WebViewActivity example

```
class WebViewActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_webview)

        val webView: WebView = findViewById(R.id.webview)
        webView.settings.javaScriptEnabled = true
        webView.loadUrl("https://www.example.com")
    }
}
```



MainActivity WebViewActivity

3 Table Activity (TableLayout)

Purpose: Arranges child views into a grid structure with rows and columns, similar to a table in HTML.

Typical use: Ideal for displaying structured content, such as forms, data tables, and lists of paired information.

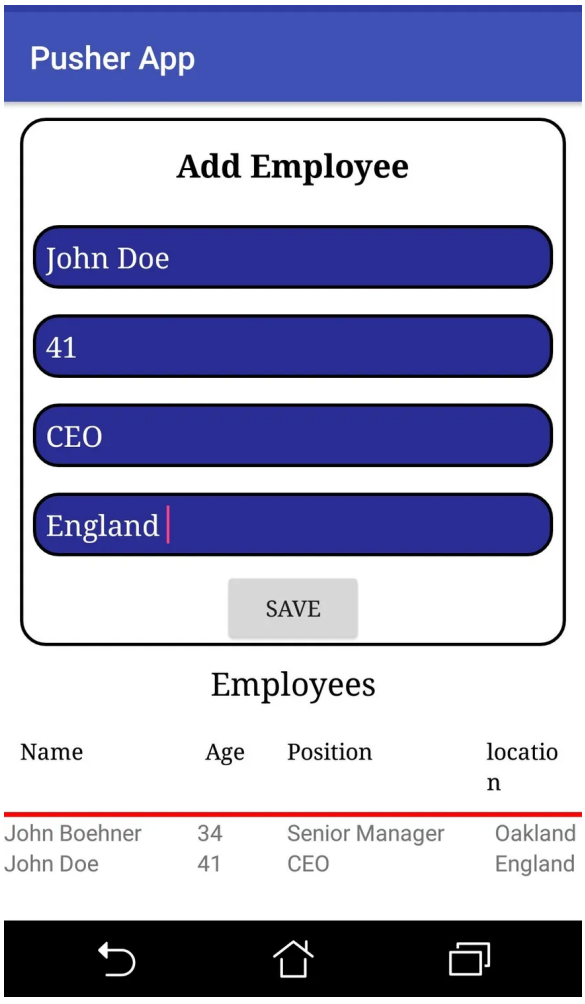
Key features:

- Rows can contain multiple columns, with each child view treated as a cell.
- Allows cells to span across multiple rows and columns.
- Simplifies complex layouts that need to organize content in a grid-like format.
- Respects layout weights to adjust the space each cell occupies within a row.

Example:

Listing 3: Table Activity XML example

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TableRow>
        <TextView android:text="Name"/>
        <EditText android:hint="Enter␣your␣name"/>
    </TableRow>
    <TableRow>
        <TextView android:text="Email"/>
        <EditText android:hint="Enter␣your␣email"/>
    </TableRow>
</TableLayout>
```



4 Scroll Activity (ScrollView)

Purpose: Allows for a vertically scrollable view, ideal for content that exceeds the screen height.

Typical Use: Used to wrap content that may exceed the screen size, such as long forms or articles.

Key Features:

- Can only support a single child view; for multiple views, wrap them in a layout (like `LinearLayout`).
- Smooth scrolling capabilities.
- Supports both vertical and horizontal scrolling.
- Useful with complex layouts to improve accessibility by allowing screen readers to scroll automatically.

Example:

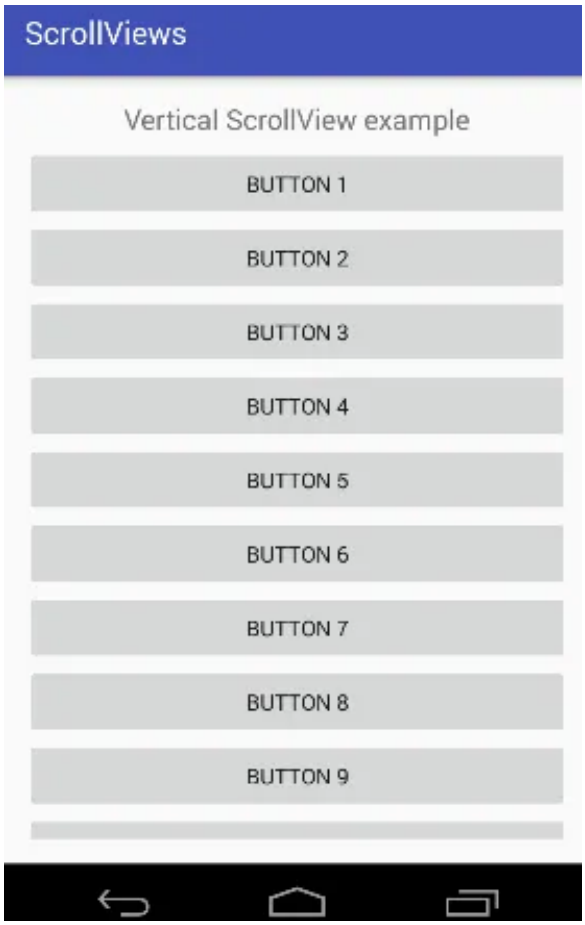
Listing 4: ScrollView XML example

```
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:orientation="vertical">

        <!-- Content goes here -->

    </LinearLayout>
</ScrollView>
```



5 Relative Activity (RelativeLayout)

Purpose: Positions UI elements relative to each other or to the parent container.

Typical Use: Good for creating dynamic, complex UI layouts that require views to be positioned in relation to each other.

Key Features:

- Offers positioning options like `layout_below`, `layout_above`, `layout_toLeftOf`, etc.
- Helps reduce nested layouts, improving performance and readability.
- Adaptive to different screen sizes.
- Can position elements based on the parent layout (e.g., centerHorizontal).

Example:

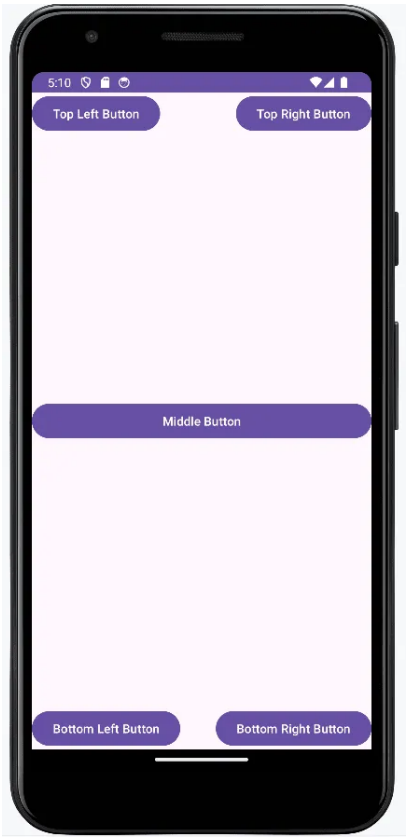
Listing 5: RelativeLayout XML example

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label"/>

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/label"
        android:hint="Enter␣text"/>

</RelativeLayout>
```



6 List Activity (ListView)

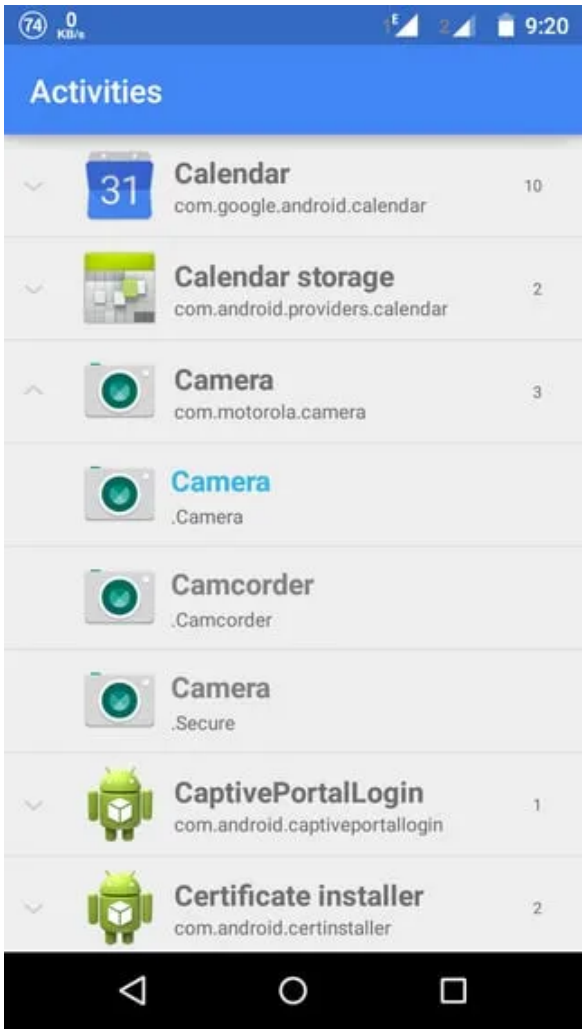
Purpose: Displays a scrollable list of items, such as a collection of data or menu options.
Typical Use: Suitable for data-driven UIs like contact lists, emails, or to-do lists.
Key Features:

- Supports adapters to bind data, making it flexible for dynamic content.
- Can handle large datasets efficiently.
- Offers customization options via custom item layouts.
- Includes built-in item click listeners.

Example:

Listing 6: ListView XML example

```
<ListView
    android:id="@+id/listView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```



7 Grid Activity (GridLayout)

Purpose: Arranges child views into a grid with rows and columns, making it easier to create visually structured layouts.

Typical Use: Useful for displaying elements like photo galleries, icon grids, or product catalogs.

Key Features:

- Flexible row and column count, allowing you to create complex grid-based layouts.
- Supports `layout_rowSpan` and `layout_columnSpan` for merging cells.
- Efficiently handles grid layouts without the need for nested layouts.
- Responsive to different screen sizes, ideal for arranging multiple elements.

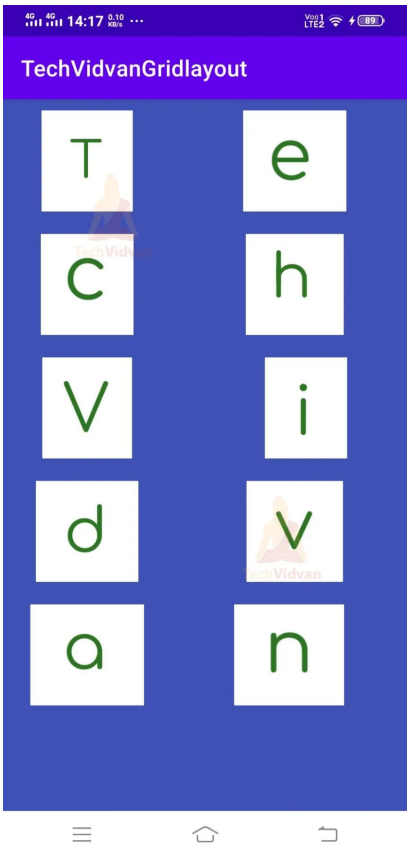
Example:

Listing 7: GridLayout XML example

```
<GridLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:rowCount="2"
    android:columnCount="2">

    <TextView android:text="Cell 1"/>
    <TextView android:text="Cell 2"/>
    <TextView android:text="Cell 3"/>
    <TextView android:text="Cell 4"/>

</GridLayout>
```



8 Frame Activity (FrameLayout)

Purpose: Designed to hold a single child view, although multiple views can overlap on top of each other.

Typical Use: Commonly used for creating UI overlays, such as floating buttons or notifications, or for loading screens.

Key Features:

- Child views are stacked, meaning they can overlay each other.
- Flexible for simple layouts where only one main child view is needed.
- Lightweight and less resource-intensive compared to more complex layouts.
- Great for handling fragments and for displaying dynamic, single-view elements.

Example:

Listing 8: FrameLayout XML example

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:src="@drawable/background"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Overlay␣Text"
        android:layout_gravity="center"/>

</FrameLayout>
```

9 Constraint Activity (ConstraintLayout)

Purpose: A powerful layout that allows for flexible, responsive positioning of UI elements, minimizing nested views.

Typical Use: Suitable for complex, responsive layouts where precise control over view alignment is required.

Key Features:

- Uses constraints to position views relative to each other and to the parent layout, allowing for flexible arrangements.
- Eliminates the need for deeply nested layouts, which improves performance.
- Great for building responsive designs with advanced features like chains, ratios, and bias.
- Supports tools like the Layout Editor in Android Studio for visual design.

Example:

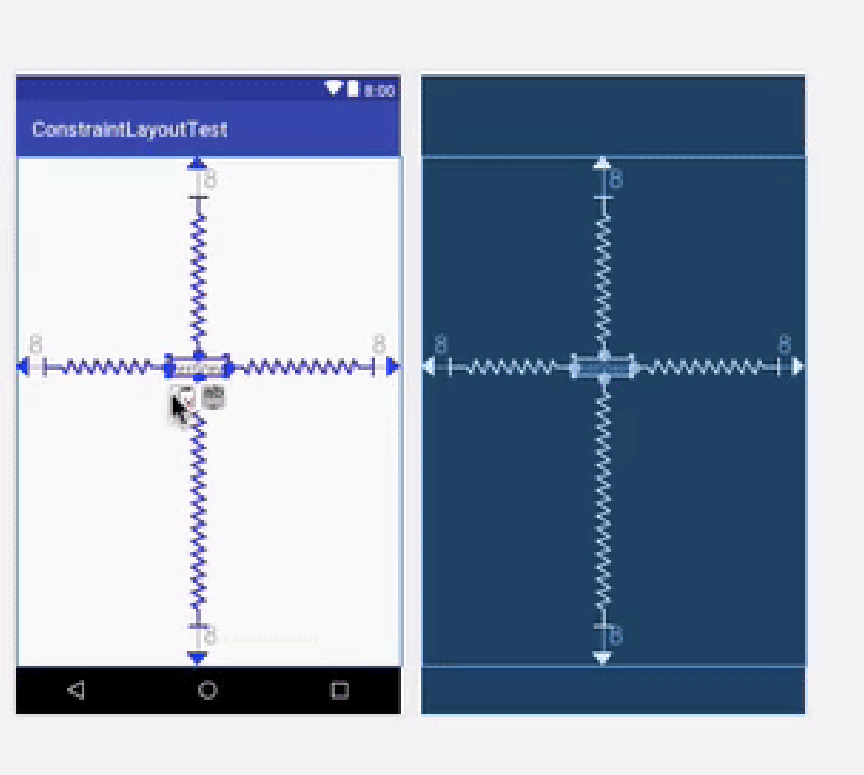
Listing 9: ConstraintLayout XML example

```
<ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"/>

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@id/label"
        app:layout_constraintStart_toStartOf="parent"/>

</ConstraintLayout>
```



10 Main Activity

Purpose: The primary entry point for an app, defined in the `AndroidManifest.xml`.

Typical Use: Serves as the main controller for user interactions, UI navigation, and app initialization.

Key Features:

- Configured with `intent-filter` tags to launch automatically when the app starts.
- Typically acts as a gateway to other activities.
- Manages lifecycle events such as `onCreate`, `onStart`, and `onResume`.
- Initializes essential components and data for the app.

Example:

Listing 10: MainActivity Java example

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```