By - Yash Mali (Android)

📄 RecyclerView in Android – (Basic to Advanced)

1. What is RecyclerView?

RecyclerView is a flexible and efficient Android UI widget used to display large sets of data in a list or grid format.
It is an advanced version of `ListView` and `GridView` with improved performance, animations, and customization support.

---

2. Why We Use RecyclerView

- **Efficient Memory Usage** – Only visible items are created, and views are recycled for better performance.
- **Flexible Layouts** – Supports Linear, Grid, and Staggered Grid layouts.
- **Built-in Animations** – Smooth add/remove animations without extra coding.
- **Supports Large Data** – Ideal for data fetched from API or local database.
- **Highly Customizable** – Full control over how each item looks and behaves.

---

3. RecyclerView vs ListView

Feature	RecyclerView	ListView
View Recycling	✅ Efficient	⚠ Less efficient
Layout Managers	✅ Multiple (Linear, Grid, Staggered)	❌ Only vertical
Animations	✅ Built-in	❌ Manual
ViewHolder Pattern	✅ Mandatory	⚠ Optional
Item Decoration	✅ Easy	❌ Limited

4. When to Use RecyclerView

- When you need **dynamic lists or grids**.
- When performance matters (especially for large datasets).
- When you want custom item designs.
- When you need animations and flexibility.

---

5. Important Components

(a) Adapter

The **Adapter** is the bridge between your data and the RecyclerView.
It **creates ViewHolders** and binds data to them.

(b) ViewHolder

The **ViewHolder** holds the references to the views inside each list item.
It prevents calling `findViewById()` repeatedly, improving performance.

(c) LayoutManager

Defines how items are arranged in RecyclerView:

- `LinearLayoutManager` → Vertical/Horizontal list
- `GridLayoutManager` → Grid display
- `StaggeredGridLayoutManager` → Pinterest-style layout

(d) Binding

Binding is the process of taking data from your dataset and assigning it to the views inside the ViewHolder.

6. How RecyclerView Works – Step by Step

1. RecyclerView asks the Adapter to **create a ViewHolder** for the first visible items.
2. The Adapter **binds data** to that ViewHolder.
3. When an item scrolls off-screen, its ViewHolder is **recycled** (reused for a new item).
4. This process repeats for smooth scrolling and performance.

7. Basic Example – News App

item_news.xml (Layout for single item)

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="8dp">

    <ImageView
        android:id="@+id/newsImage"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:scaleType="centerCrop" />

    <TextView
        android:id="@+id/newsTitle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/newsDescription"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="14sp" />
```

```
    <TextView
        android:id="@+id/newsAuthor"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="12sp"
        android:textStyle="italic" />

    <TextView
        android:id="@+id/newsDate"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="12sp" />
</LinearLayout>
```

NewsAdapter.kt

```
class NewsAdapter(private val newsList: List<News>) :
    RecyclerView.Adapter<NewsAdapter.NewsViewHolder>() {

    class NewsViewHolder(itemView: View) :
RecyclerView.ViewHolder(itemView) {
        val newsImage: ImageView =
itemView.findViewById(R.id.newsImage)
        val newsTitle: TextView =
itemView.findViewById(R.id.newsTitle)
        val newsDescription: TextView =
itemView.findViewById(R.id.newsDescription)
        val newsAuthor: TextView =
itemView.findViewById(R.id.newsAuthor)
        val newsDate: TextView = itemView.findViewById(R.id.newsDate)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
NewsViewHolder {
        val view =
LayoutInflater.from(parent.context).inflate(R.layout.item_news,
parent, false)
```

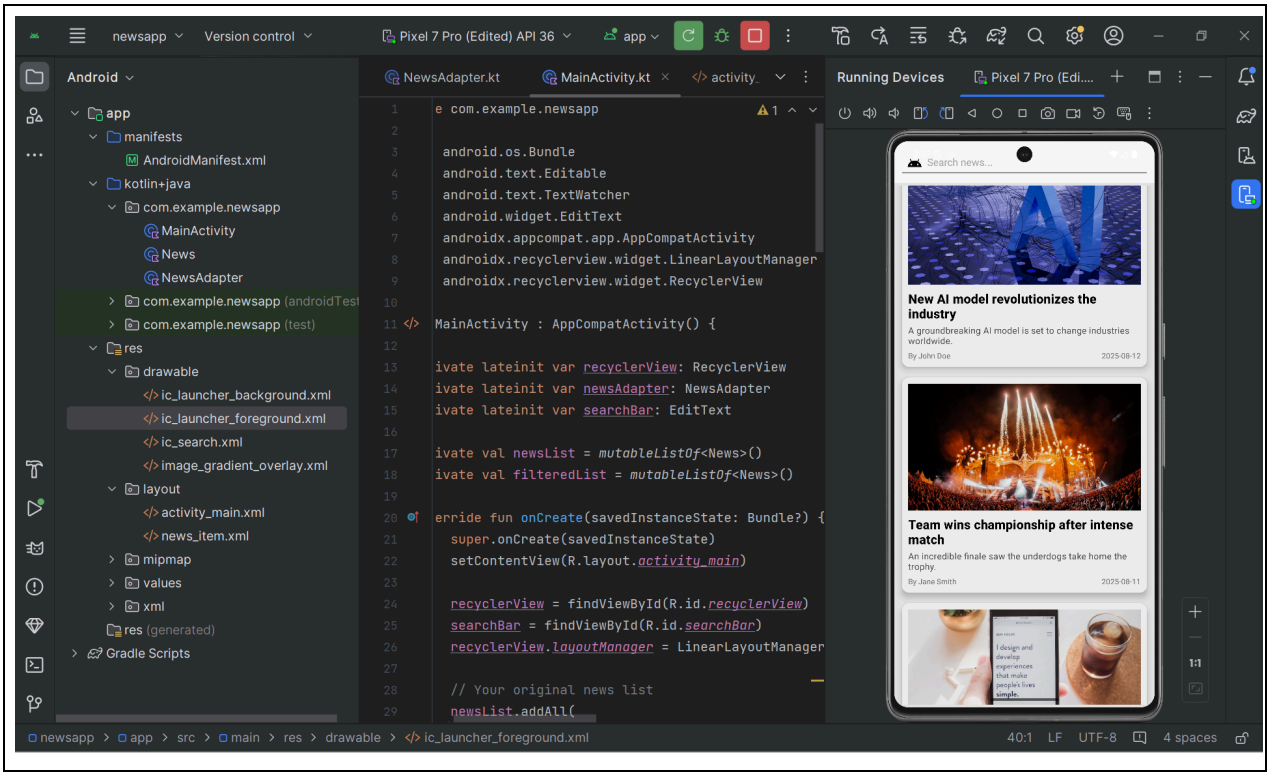
```
        return NewsViewHolder(view)
    }

    override fun onBindViewHolder(holder: NewsViewHolder, position:
Int) {
        val news = newsList[position]
        holder.newsTitle.text = news.title
        holder.newsDescription.text = news.description
        holder.newsAuthor.text = news.author
        holder.newsDate.text = news.date

Glide.with(holder.itemView.context).load(news.imageUrl).into(holder.ne
wsImage)
    }

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

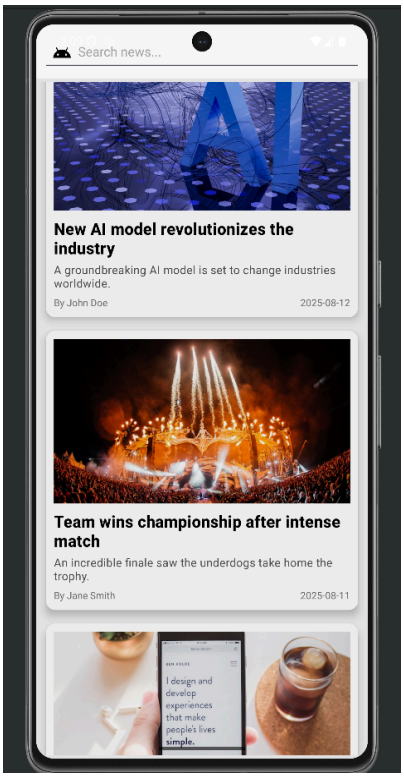
MainActivity.kt

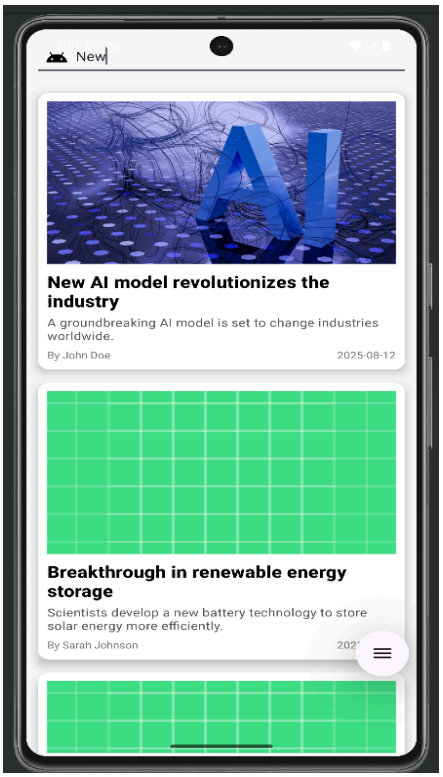
```
class MainActivity : AppCompatActivity() {
    private lateinit var recyclerView: RecyclerView
    private lateinit var newsAdapter: NewsAdapter

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

        recyclerView = findViewById(R.id.recyclerView)
        recyclerView.layoutManager = LinearLayoutManager(this)

        val newsList = listOf(
            News("Leaders gather for climate summit",
                 "World leaders meet to discuss urgent climate
action.",
                 "Michael Lee",
                 "2025-08-10",
```

```
"https://images.unsplash.com/photo-1509395176047-4a66953fd231?w=800"),
            // Add more news items here...
        )

        newsAdapter = NewsAdapter(newsList)
        recyclerView.adapter = newsAdapter
    }
}
```

News App - OutputApp







8. Pros of RecyclerView

- ✅ Efficient for large data
- ✅ Smooth scrolling
- ✅ Flexible layouts
- ✅ Easy animations
- ✅ Customizable item design

9. Cons of RecyclerView

- ⚠ More complex than ListView
- ⚠ Requires more code to implement
- ⚠ Needs Adapter, ViewHolder, and LayoutManager setup

10. Best Practices

- Always use `ViewHolder` to avoid performance issues.
- Use `DiffUtil` for updating lists efficiently.
- Avoid creating new objects inside `onBindViewHolder()` unnecessarily.
- Use `Glide` or `Picasso` for image loading.