ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION)

UI/UX DESIGN IN ANDROID (WEEKS 4-6)

UNDERSTANDING ANDROID LAYOUTS: LINEARLAYOUT, RELATIVELAYOUT, AND CONSTRAINTLAYOUT

CHAPTER 1: INTRODUCTION TO ANDROID LAYOUTS

1.1 What Are Android Layouts?

- Android layouts define the **structure and arrangement** of UI elements in an app.
- They determine **how Views (buttons, text, images, etc.) are positioned and interact** within an Activity or Fragment.
- Android provides multiple layout types, with **LinearLayout, RelativeLayout, and ConstraintLayout** being the most commonly used.

- **Why Are Layouts Important?**

✅ **Define the UI Structure** – Organizes UI components efficiently.

✅ **Ensure Responsiveness** – Adjusts UI across different screen sizes.

✅ **Improve Performance** – Helps optimize rendering speed and memory usage.

✅ **Maintain Scalability** – Supports complex designs with flexible positioning.

🔹 **Types of Layouts in Android:**

✓ **LinearLayout** – Arranges elements in a single direction (vertical/horizontal).

✓ **RelativeLayout** – Positions elements relative to each other.

✓ **ConstraintLayout** – Offers flexible and efficient positioning using constraints.

📌 **Example:**

- **A messaging app's chat screen may use a LinearLayout for message bubbles, a RelativeLayout for the input bar, and a ConstraintLayout for profile sections.**

CHAPTER 2: UNDERSTANDING LINEARLAYOUT

2.1 What Is LinearLayout?

🔹 LinearLayout is a **simple layout manager** that arranges child views in a **single direction** (either **vertically** or **horizontally**).

✅ **When to Use LinearLayout?**

✓ When UI elements **need to be stacked in a single direction**.

✓ When a **simple and predictable layout** is required.

✓ When the UI **does not require complex positioning**.

📌 **Example: Basic Vertical LinearLayout**

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, Android!" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me" />
</LinearLayout>
```

✅ **Output:**

Hello, Android!

[Click Me]

(Elements are stacked vertically.)

2.2 Key Properties of LinearLayout

Property	Description	Example

android:orientation	Defines direction (**vertical/horizontal**).	android:orientation="horizontal"
android:gravity	Aligns child views within LinearLayout.	android:gravity="center"
android:layout_weight	Distributes extra space among views.	android:layout_weight="1"

📌 **Example: Horizontal LinearLayout with Weights**

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

  <Button
    android:layout_width="0dp"
    android:layout_weight="1"
    android:layout_height="wrap_content"
    android:text="Button 1"/>

  <Button
    android:layout_width="0dp"
    android:layout_weight="2"
```

```
    android:layout_height="wrap_content"
    android:text="Button 2"/>
</LinearLayout>
```

✅ **Output:**

- "Button 2" occupies **twice the space** of "Button 1".

📌 **Example Use Case:**

- **A toolbar with evenly spaced buttons in a chat app can use a horizontal LinearLayout.**

CHAPTER 3: UNDERSTANDING RELATIVELAYOUT

3.1 What Is RelativeLayout?

🔹 RelativeLayout allows **positioning views relative to each other** or to the **parent container**.

✅ **When to Use RelativeLayout?**

✓ When elements **depend on each other's position**.
✓ When a **more dynamic and flexible layout** is needed.
✓ When **avoiding nested LinearLayouts** for performance improvement.

📌 **Example: Positioning Views Using RelativeLayout**

```
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
```

```
<TextView
    android:id="@+id/textHello"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, Android!"
    android:layout_centerHorizontal="true"/>

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click Me"
    android:layout_below="@id/textHello"
    android:layout_centerHorizontal="true"/>
</RelativeLayout>
```

✅ **Output:**

Hello, Android!

[Click Me]

(The TextView is centered, and the button appears below it.)

3.2 Key Properties of RelativeLayout

Property	Description	Example

android:layout_alignParentTop	Aligns the view to the top of the parent.	android:layout_alignParentTop="true"
android:layout_below	Positions a view below another view.	android:layout_below="@id/textHello"
android:layout_toRightOf	Positions a view to the right of another.	android:layout_toRightOf="@id/button1"

📌 **Example Use Case:**

- **A profile screen in a social media app can use RelativeLayout to position the profile picture, name, and follow button dynamically.**

CHAPTER 4: UNDERSTANDING CONSTRAINTLAYOUT

4.1 What Is ConstraintLayout?

🔹 ConstraintLayout is a **powerful and flexible layout** that allows positioning views using **constraints** rather than nested layouts.

🔹 It **improves performance** by reducing the **number of nested views**.

✅ When to Use ConstraintLayout?

✓ When creating **complex layouts with multiple alignments**.

✓ When improving **UI rendering speed and efficiency**.

✓ When reducing **unnecessary nested layouts**.

📌 Example: Simple ConstraintLayout Usage

```
<androidx.constraintlayout.widget.ConstraintLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

  <TextView
    android:id="@+id/textHello"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, Android!"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"/>

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click Me"
```

```
    app:layout_constraintTop_toBottomOf="@id/textHello"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

✅ **Output:**

Hello, Android!

[Click Me]

(Both views are centered dynamically without nesting.)

4.2 Key Properties of ConstraintLayout

Property	Description	Example
app:layout_constraintTop_toTopOf	Aligns top of the view to another view or parent.	app:layout_constraintTop_toTopOf="parent"
app:layout_constraintLeft_toLeftOf	Aligns left side of the view.	app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintBottom_toBottomOf	Aligns bottom side of the view.	app:layout_constraintBottom_toBottomOf="parent"

📌 **Example Use Case:**

- A login screen can use ConstraintLayout to align the username, password, and login button efficiently.

Exercise: Test Your Understanding

- What is the key difference between LinearLayout and RelativeLayout?
- When should you use ConstraintLayout over LinearLayout?
- What does android:layout_weight do in LinearLayout?
- How does ConstraintLayout improve performance?

Conclusion

✅ LinearLayout is best for simple, one-directional layouts.
✅ RelativeLayout allows dynamic positioning relative to other views.
✅ ConstraintLayout is the most powerful, reducing nested layouts and improving performance.

BUTTONS, TEXTVIEWS, EDITTEXTS, AND IMAGEVIEWS IN ANDROID DEVELOPMENT

CHAPTER 1: UNDERSTANDING ANDROID UI COMPONENTS

1.1 What Are UI Components in Android?

🔹 **User Interface (UI) components** are the **building blocks** of an Android application. They help developers create **interactive, visually appealing, and user-friendly** applications.

🔹 Android UI components are organized within **layouts** and define how an application looks and behaves.

✅ **Key UI Components in Android:**

✓ **TextView** – Displays text.

✓ **EditText** – Allows users to enter text.

✓ **Button** – Triggers an action when clicked.

✓ **ImageView** – Displays images.

📌 **Example:**

- **A login screen with an EditText for entering a username, a Button to submit, and a TextView for error messages.**

CHAPTER 2: WORKING WITH BUTTONS IN ANDROID

2.1 What is a Button?

🔹 A **Button** in Android is an interactive UI element that **triggers an action when clicked**.

🔹 It is commonly used for **submitting forms, navigating screens, and performing actions**.

✅ **Types of Buttons in Android:**

✓ **Basic Button (Button)** – A standard button that performs an action.

✓ **Image Button (ImageButton)** – A button that contains an image instead of text.

✓ **Floating Action Button (FAB)** – A circular button used for primary actions.

2.2 Implementing a Button in XML

📌 **Example:**

```
<Button
  android:id="@+id/btnSubmit"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Submit"
  android:onClick="submitForm"/>
```

🔹 **Explanation:**

- android:id ⟶ Unique identifier for the button.
- android:text ⟶ Text displayed on the button.
- android:onClick ⟶ Calls a function (submitForm) when clicked.

2.3 Handling Button Clicks in Java/Kotlin

📌 **Java Code:**

```
Button btnSubmit = findViewById(R.id.btnSubmit);
```

```
btnSubmit.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      Toast.makeText(MainActivity.this, "Button Clicked!", Toast.LENGTH_SHORT).show();
   }
});
```

📌 **Kotlin Code:**

```
val btnSubmit: Button = findViewById(R.id.btnSubmit)
btnSubmit.setOnClickListener {
   Toast.makeText(this, "Button Clicked!", Toast.LENGTH_SHORT).show()
}
```

🔹 **Explanation:**

- setOnClickListener → Defines what happens when the button is clicked.
- Toast.makeText() → Displays a short pop-up message.

CHAPTER 3: WORKING WITH TEXTVIEW IN ANDROID

3.1 What is a TextView?

🔹 **TextView** is a UI component that **displays static or dynamic text** in an Android app.

🔹 It is commonly used for **headings, labels, messages, and instructions**.

3.2 Implementing TextView in XML

📌 Example:

```
<TextView
  android:id="@+id/tvMessage"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Welcome to Android!"
  android:textSize="18sp"
  android:textColor="#oooooo"/>
```

🔹 Explanation:

- android:textSize ⟶ Defines text size in sp (scale-independent pixels).
- android:textColor ⟶ Sets text color using a hex code.

3.3 Updating TextView Dynamically in Java/Kotlin

📌 Java Code:

```
TextView tvMessage = findViewById(R.id.tvMessage);
tvMessage.setText("Hello, User!");
```

📌 Kotlin Code:

```
val tvMessage: TextView = findViewById(R.id.tvMessage)
tvMessage.text = "Hello, User!"
```

📌 **Example:**

- **A messaging app updates a TextView with the latest chat message.**

CHAPTER 4: WORKING WITH EDITTEXT IN ANDROID

4.1 What is EditText?

🔹 **EditText** is an interactive UI component that **allows users to enter and edit text**.

🔹 It is used for **forms, search bars, and login inputs**.

✅ **Types of Input Fields:**

✓ **Plain Text** – For normal text input.

✓ **Password** – Hides input characters.

✓ **Number** – Accepts only numerical input.

4.2 Implementing EditText in XML

📌 **Example:**

```
<EditText
  android:id="@+id/etUsername"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:hint="Enter Username"
  android:inputType="text"/>
```

🔹 **Explanation:**

- android:hint ⟶ Displays placeholder text inside the field.

- android:inputType → Restricts input to a specific type (text, number, email, etc.).

4.3 Retrieving User Input from EditText

📌 **Java Code:**

```
EditText etUsername = findViewById(R.id.etUsername);
String username = etUsername.getText().toString();
```

📌 **Kotlin Code:**

```
val etUsername: EditText = findViewById(R.id.etUsername)
val username = etUsername.text.toString()
```

📌 **Example:**

- **A login screen retrieves the username entered by the user.**

CHAPTER 5: WORKING WITH IMAGEVIEW IN ANDROID

5.1 What is an ImageView?

🔹 **ImageView** is a UI component that **displays images from local storage, drawable resources, or the internet**.

✅ **Common Use Cases:**

✓ Displaying **logos and icons**.

✓ Showing **user profile pictures**.

✓ Loading images dynamically from a server.

5.2 Implementing ImageView in XML

📌 **Example:**

```
<ImageView
  android:id="@+id/imgProfile"
  android:layout_width="100dp"
  android:layout_height="100dp"
  android:src="@drawable/profile_pic"/>
```

🔹 **Explanation:**

- android:src ⟶ Loads an image from **drawable resources**.

5.3 Dynamically Changing Image in Java/Kotlin

📌 **Java Code:**

```
ImageView imgProfile = findViewById(R.id.imgProfile);
imgProfile.setImageResource(R.drawable.new_profile_pic);
```

📌 **Kotlin Code:**

```
val imgProfile: ImageView = findViewById(R.id.imgProfile)
imgProfile.setImageResource(R.drawable.new_profile_pic)
```

📌 **Example:**

- **A social media app updates the user's profile picture dynamically.**

Exercise: Test Your Understanding

- **What is the difference between TextView and EditText?**
- **How do you set an image dynamically in an ImageView?**

- What is the purpose of setOnClickListener in a Button?
- How do you retrieve text entered in an EditText?
- List three real-world applications of ImageView in Android apps.

Conclusion

✅ Buttons handle user actions, such as submitting forms or navigating screens.
✅ TextViews display static or dynamic text in an app.
✅ EditText allows user input, commonly used in forms and authentication screens.
✅ ImageViews display images, which can be loaded from local storage or the internet.

LISTVIEW VS RECYCLERVIEW IN ANDROID DEVELOPMENT

CHAPTER 1: INTRODUCTION TO LISTVIEW & RECYCLERVIEW

1.1 What is ListView?

🔹 **ListView** is an Android UI component used to display **scrollable lists of data** in a vertical format.

🔹 It is part of the original **Android SDK** and provides a simple way to create lists.

🔹 ListView requires an **Adapter** (like ArrayAdapter or CursorAdapter) to bind data to the UI.

✅ **Key Features of ListView:**

✓ Displays **vertical** scrollable lists.

✓ Requires an **adapter** to bind data.

✓ Supports **single/multiple item selection**.

✓ Comes with built-in **view recycling** but is less optimized.

📌 **Example:**

- **A contacts app displays a list of saved contacts using ListView.**

1.2 What is RecyclerView?

🔹 **RecyclerView** is an improved version of ListView, introduced in **Android 5.0 (Lollipop)**.

🔹 It is part of the **AndroidX library** and is designed for better **performance, flexibility, and efficiency**.

🔹 RecyclerView uses a **ViewHolder pattern** for improved memory usage.

✅ **Key Features of RecyclerView:**

✓ More **efficient view recycling** compared to ListView.

✓ Supports **horizontal, vertical, grid, and staggered layouts**.

✓ Uses **RecyclerView.Adapter and ViewHolder** for optimized rendering.

✓ Supports **animation effects, drag & drop, swipe gestures**.

📌 **Example:**

- **A shopping app displays a grid of products using RecyclerView.**

CHAPTER 2: KEY DIFFERENCES BETWEEN LISTVIEW & RECYCLERVIEW

2.1 ListView vs RecyclerView – A Comparison Table

Feature	ListView	RecyclerView
Performance	Less optimized	High performance with better memory usage
View Recycling	Uses convertView in Adapter	Uses ViewHolder pattern for efficient recycling
Flexibility	Limited layouts	Supports Linear, Grid, and Staggered layouts
Customization	Harder to customize	Highly customizable
Animations	No built-in animations	Supports item animations

Scrolling Efficiency	Slower for large data sets	Faster and more responsive
Event Handling	Built-in click listener	Requires manual event handling

📌 **Example:**

- **If an app needs a simple vertical list, ListView is fine. If complex interactions or large data sets are required, RecyclerView is better.**

CHAPTER 3: IMPLEMENTING LISTVIEW IN ANDROID

3.1 Steps to Implement a ListView

✅ **Step 1: Add ListView in XML Layout**

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

✅ **Step 2: Create an ArrayAdapter for Data Binding**

```
String[] fruits = {"Apple", "Banana", "Cherry", "Date"};


ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, fruits);

ListView listView = findViewById(R.id.listView);

listView.setAdapter(adapter);
```

✅ **Step 3: Set Click Listener for ListView Items**

```
listView.setOnItemClickListener((parent, view, position, id) -> {
   String selectedItem = (String) parent.getItemAtPosition(position);
   Toast.makeText(getApplicationContext(), "Clicked: " +
selectedItem, Toast.LENGTH_SHORT).show();
});
```

📌 **Example:**

- **A simple ListView displaying a list of fruit names.**

CHAPTER 4: IMPLEMENTING RECYCLERVIEW IN ANDROID

4.1 Steps to Implement a RecyclerView

✅ **Step 1: Add RecyclerView in XML Layout**

```
<androidx.recyclerview.widget.RecyclerView
   android:id="@+id/recyclerView"
   android:layout_width="match_parent"
   android:layout_height="match_parent"/>
```

✅ **Step 2: Create a Model Class for Data**

```
public class Fruit {
   String name;

   public Fruit(String name) {
      this.name = name;
   }
```

```
    public String getName() {
        return name;
    }
}
```

✅ Step 3: Create a RecyclerView Adapter

```
public class FruitAdapter extends
RecyclerView.Adapter<FruitAdapter.ViewHolder> {
    private List<Fruit> fruitList;

    public FruitAdapter(List<Fruit> fruitList) {
        this.fruitList = fruitList;
    }

    public static class ViewHolder extends RecyclerView.ViewHolder {
        TextView fruitName;

        public ViewHolder(View itemView) {
            super(itemView);
            fruitName = itemView.findViewById(R.id.fruitName);
        }
    }

    @Override
```

```
  public ViewHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
    View view =
LayoutInflater.from(parent.getContext()).inflate(R.layout.fruit_item
, parent, false);
    return new ViewHolder(view);
  }

  @Override
  public void onBindViewHolder(ViewHolder holder, int position) {
    holder.fruitName.setText(fruitList.get(position).getName());
  }

  @Override
  public int getItemCount() {
    return fruitList.size();
  }
}
```

✅ Step 4: Set Up RecyclerView in MainActivity

```
RecyclerView recyclerView = findViewById(R.id.recyclerView);
recyclerView.setLayoutManager(new LinearLayoutManager(this));

List<Fruit> fruits = Arrays.asList(new Fruit("Apple"), new
Fruit("Banana"), new Fruit("Cherry"));
```

```
FruitAdapter adapter = new FruitAdapter(fruits);

recyclerView.setAdapter(adapter);
```

📌 **Example:**

- **A RecyclerView displaying a list of fruits with improved performance.**

CHAPTER 5: WHEN TO USE LISTVIEW VS RECYCLERVIEW?

Use Case	Choose
Simple, static lists	ListView
Small datasets	ListView
Complex lists with animations	RecyclerView
Large datasets with view recycling	RecyclerView
Grid or staggered layouts	RecyclerView
Dynamic content requiring smooth scrolling	RecyclerView

📌 **Example:**

- **An e-commerce app should use RecyclerView for product listings, while a basic settings screen might use ListView.**

Exercise: Test Your Understanding

- **What is the main advantage of RecyclerView over ListView?**
- **Explain how the ViewHolder pattern improves RecyclerView performance.**
- **Write a basic ListView implementation displaying a list of cities.**

- **How does RecyclerView handle multiple layouts in a single list?**
- **Which component would you use for a chat app's message list? Why?**

Conclusion

✅ **ListView is a simple way to display vertical lists but lacks flexibility and performance optimization.**
✅ **RecyclerView is a modern alternative with better memory usage, animations, and customization.**
✅ **For complex UI requirements, RecyclerView is the preferred choice.**
✅ **Understanding both ListView and RecyclerView helps developers build efficient Android applications.**

IMPLEMENTING ADAPTERS AND VIEWHOLDERS IN ANDROID DEVELOPMENT

CHAPTER 1: INTRODUCTION TO ADAPTERS AND VIEWHOLDERS

1.1 What are Adapters and ViewHolders?

🔹 In Android development, **Adapters and ViewHolders** are used to efficiently manage lists and grids in **RecyclerView** and **ListView**.

🔹 These components improve **memory efficiency, performance, and user experience** when displaying large datasets.

✅ **Adapter** – Acts as a bridge between the data source and UI components, converting data into viewable items.

✅ **ViewHolder** – Holds references to item views in memory, preventing repeated calls to findViewById() and improving performance.

📌 **Example:**

- **A shopping app uses an Adapter to populate a RecyclerView with product images, names, and prices.**

CHAPTER 2: UNDERSTANDING THE ADAPTER PATTERN

2.1 What is the Adapter Pattern?

🔹 The **Adapter pattern** is a structural design pattern used in Android to **bind data sources (arrays, databases, APIs) to UI elements like RecyclerView and ListView**.

🔹 The Adapter ensures that data is displayed correctly and efficiently.

✅ **Types of Adapters in Android:**

Adapter	Used With	Description
ArrayAdapter	ListView, Spinner	Converts an array into a ListView format.
BaseAdapter	ListView, GridView	Provides more customization than ArrayAdapter.
RecyclerView.Adapter	RecyclerView	The most efficient adapter for large datasets.
CursorAdapter	ListView, GridView	Used to bind data from SQLite databases.

📌 **Example:**

- **A chat app uses RecyclerView.Adapter to load recent conversations dynamically.**

CHAPTER 3: IMPLEMENTING RECYCLERVIEW ADAPTER AND VIEWHOLDER

3.1 Steps to Implement an Adapter in RecyclerView

1. **Create a Data Model** – Define a class representing list items.
2. **Design an Item Layout** – Create an XML layout file for each row.
3. **Create a ViewHolder Class** – Holds references to item views.
4. **Create an Adapter Class** – Binds data to the ViewHolder.
5. **Attach Adapter to RecyclerView** – Connect RecyclerView to the adapter.

3.2 Implementing a RecyclerView Adapter (Kotlin)

📌 Step 1: Create a Data Model

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

📌 Step 2: Create a Layout for Each List Item (item_user.xml)

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:padding="10dp"
  android:orientation="horizontal">

  <TextView
    android:id="@+id/tvName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="18sp" />

  <TextView
    android:id="@+id/tvAge"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingLeft="10dp"
    android:textSize="16sp"/>
```

```
</LinearLayout>
```

📌 Step 3: Create a ViewHolder Class

```
class UserViewHolder(itemView: View) :
RecyclerView.ViewHolder(itemView) {

  val nameTextView: TextView =
itemView.findViewById(R.id.tvName)

  val ageTextView: TextView = itemView.findViewById(R.id.tvAge)

}
```

📌 Step 4: Create an Adapter Class

```
class UserAdapter(private val userList: List<User>) :

  RecyclerView.Adapter<UserViewHolder>() {


  override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): UserViewHolder {

    val itemView = LayoutInflater.from(parent.context)

      .inflate(R.layout.item_user, parent, false)

    return UserViewHolder(itemView)

  }


  override fun onBindViewHolder(holder: UserViewHolder, position:
Int) {

    val user = userList[position]

    holder.nameTextView.text = user.name

    holder.ageTextView.text = "Age: ${user.age}"
```

```
    }

    override fun getItemCount(): Int {
        return userList.size
    }
}
```

📌 Step 5: Attach Adapter to RecyclerView in MainActivity.kt

```
val recyclerView: RecyclerView = findViewById(R.id.recyclerView)
recyclerView.layoutManager = LinearLayoutManager(this)

val userList = listOf(
    User("Alice", 25),
    User("Bob", 30),
    User("Charlie", 28)
)

val adapter = UserAdapter(userList)
recyclerView.adapter = adapter
```

🔹 **Output:**

✅ Displays a scrollable list of user names and ages in a RecyclerView.

CHAPTER 4: VIEWHOLDER OPTIMIZATION AND PERFORMANCE IMPROVEMENTS

4.1 Why Use ViewHolder in RecyclerView?

🔹 The **ViewHolder pattern** prevents excessive calls to findViewById(), improving **scrolling performance** and reducing **memory usage**.

✅ **Without ViewHolder:**
❌ RecyclerView calls findViewById() for each item, slowing performance.

✅ **With ViewHolder:**
✔ Item views are cached in memory, eliminating redundant lookups.

📌 **Example:**

- **A music app using ViewHolder reduces lag when scrolling through a song list.**

4.2 ViewHolder Pattern in Java (For Comparison)

```
public class UserViewHolder extends RecyclerView.ViewHolder {
  TextView nameTextView, ageTextView;

  public UserViewHolder(View itemView) {
    super(itemView);
    nameTextView = itemView.findViewById(R.id.tvName);
    ageTextView = itemView.findViewById(R.id.tvAge);
  }
```

```
}
```

📌 **Example:**

- **A news app implements ViewHolder for smooth scrolling through headlines.**

Chapter 5: Handling Click Events in RecyclerView

5.1 Adding Click Listeners to RecyclerView Items

📌 **Modify ViewHolder to Handle Clicks:**

```
class UserViewHolder(itemView: View) :
RecyclerView.ViewHolder(itemView) {

  val nameTextView: TextView =
itemView.findViewById(R.id.tvName)

  val ageTextView: TextView = itemView.findViewById(R.id.tvAge)


  init {

    itemView.setOnClickListener {

      Toast.makeText(itemView.context, "Clicked on
${nameTextView.text}", Toast.LENGTH_SHORT).show()

    }

  }

}
```

📌 **Example:**

- **A contacts app shows a detailed profile when a user taps a contact name.**

Exercise: Test Your Understanding

- What is the role of an Adapter in Android development?
- How does a ViewHolder improve RecyclerView performance?
- List three types of Adapters used in Android.
- Why is onCreateViewHolder() used in RecyclerView Adapter?
- How do you implement a click listener inside a RecyclerView?

Conclusion

✅ Adapters are essential for displaying data dynamically in RecyclerView.
✅ The ViewHolder pattern enhances performance by reducing findViewById() calls.
✅ RecyclerView is more efficient than ListView, making it the preferred choice for modern Android applications.
✅ Click listeners can be added inside the ViewHolder to handle user interactions.

MATERIAL DESIGN GUIDELINES IN ANDROID DEVELOPMENT

CHAPTER 1: INTRODUCTION TO MATERIAL DESIGN

1.1 What Is Material Design?

🔹 **Material Design** is a **design language developed by Google** to create **consistent, visually appealing, and user-friendly** interfaces across Android and web applications.

🔹 It provides **guidelines, components, and principles** that ensure **intuitive, smooth, and accessible UI experiences**.

1.2 Why Use Material Design?

✅ **Consistency** – Ensures a uniform look and feel across Android apps.

✅ **Responsiveness** – Adapts well to different screen sizes and devices.

✅ **Usability & Accessibility** – Improves user interaction with clear visual hierarchy.

✅ **Animation & Interaction** – Enhances UI with fluid motion and transitions.

🔹 **Key Principles of Material Design:**

✓ **Material is Metaphor** – Inspired by real-world objects (shadows, layers, depth).

✓ **Bold, Graphic, and Intentional** – Uses vibrant colors and meaningful typography.

✓ **Motion Provides Meaning** – Animations guide users and improve engagement.

📌 **Example:**

- **Google Apps (Gmail, Google Drive, YouTube) use Material Design for a seamless and unified user experience.**

CHAPTER 2: MATERIAL DESIGN COMPONENTS

2.1 Material Components in Android

- **Android provides built-in Material Design components** via the **Material Components for Android (MDC-Android) library**.
- To use Material Design, **add the Material Components library** in build.gradle:

```
dependencies {
    implementation 'com.google.android.material:material:1.7.0'
}
```

- **Key Material Design Components:**

✓ **Material Toolbar** – Stylish app bars.
✓ **Material Buttons** – Raised, outlined, and floating action buttons (FAB).
✓ **Material Text Fields** – Input fields with clear visual guidance.
✓ **Material Cards** – Grouping content with elevation.
✓ **Material Navigation Drawer** – Side panel for navigation.
✓ **Material Bottom Navigation** – Persistent navigation at the bottom.

CHAPTER 3: APPLYING MATERIAL DESIGN IN ANDROID UI

3.1 Material Buttons

- **Types of Buttons in Material Design:**

✓ **Elevated Button** – Raised above the surface with shadows.

✓ **Outlined Button** – Transparent button with a border.
✓ **Text Button** – Minimalist button without borders.
✓ **Floating Action Button (FAB)** – Circular button for primary actions.

📌 **Example: Using Material Buttons in XML**

```
<com.google.android.material.button.MaterialButton
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Elevated Button"
  style="@style/Widget.MaterialComponents.Button"/>
```

📌 **Example: Floating Action Button (FAB)**

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  app:srcCompat="@android:drawable/ic_input_add"
  app:backgroundTint="@color/purple_500"/>
```

📌 **Example Use Case:**

- **A "Compose Email" FAB in a mail app lets users quickly create new emails.**

3.2 Material Text Fields

🔹 **Material text fields improve user input interactions** with **clear labels, icons, and helper texts**.

📌 **Example: Implementing Material Text Fields**

```
<com.google.android.material.textfield.TextInputLayout
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:hint="Enter your email">

  <com.google.android.material.textfield.TextInputEditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
</com.google.android.material.textfield.TextInputLayout>
```

📌 **Example Use Case:**

- **A login screen uses Material Text Fields for better clarity in email and password inputs.**

3.3 Material Cards

🔹 **Material Cards are used to group related content** with **rounded corners, shadows, and elevation effects**.

📌 **Example: Using Material Cards in XML**

```
<com.google.android.material.card.MaterialCardView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  app:cardCornerRadius="8dp"
  app:cardElevation="4dp"
```

```
android:padding="16dp">

<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Material Card Example"/>
</com.google.android.material.card.MaterialCardView>
```

📌 **Example Use Case:**

- **An e-commerce app displays products in Material Cards for a modern UI.**

CHAPTER 4: MATERIAL DESIGN NAVIGATION COMPONENTS

4.1 Material Navigation Drawer

🔹 **A side panel (hamburger menu) used for app navigation.**

📌 **Example: Implementing a Navigation Drawer**

```
<com.google.android.material.navigation.NavigationView
  android:layout_width="wrap_content"
  android:layout_height="match_parent"
  app:menu="@menu/navigation_menu"/>
```

📌 **Example Use Case:**

- **A news app uses a Navigation Drawer to switch between categories like Politics, Sports, and Technology.**

4.2 Bottom Navigation Bar

- **A persistent bottom bar used for primary app navigation.**

📌 **Example: Implementing Bottom Navigation**

```
<com.google.android.material.bottomnavigation.BottomNavigationView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  app:menu="@menu/bottom_nav_menu"/>
```

📌 **Example Use Case:**

- **A music app uses Bottom Navigation to switch between Home, Library, and Search.**

CHAPTER 5: MATERIAL DESIGN THEMES & COLORS

5.1 Applying Material Design Themes

- **Material themes define colors, typography, and styles globally.**
- Add Material Design theme to themes.xml:

```
<style name="Theme.MyApp" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
  <item name="colorPrimary">@color/blue_700</item>
  <item name="colorAccent">@color/blue_500</item>
</style>
```

📌 **Example Use Case:**

- **A banking app applies a consistent Material theme with brand colors across all screens.**

5.2 Material Color System

🔹 Material Design follows a **structured color system**:

✅ **Primary Color** – The main color of the app.

✅ **Secondary Color** – Used for accents and highlights.

✅ **Surface Color** – Background colors for UI elements.

✅ **Error Color** – Indicates warnings or validation errors.

📌 **Example: Defining Colors in colors.xml**

```
<color name="colorPrimary">#6200EE</color>

<color name="colorSecondary">#03DAC5</color>

<color name="colorSurface">#FFFFFF</color>
```

📌 **Example Use Case:**

- **A food delivery app uses a warm color palette for an inviting feel.**

Exercise: Test Your Understanding

- **What are the three key principles of Material Design?**
- **What is the role of Material Components in Android?**
- **How does a Material Button differ from a normal Button?**
- **Why is ConstraintLayout preferred with Material Design?**
- **How do Material themes improve UI consistency?**

Conclusion

✅ **Material Design ensures visually appealing and user-friendly Android interfaces.**

✅ **It provides reusable UI components like buttons, text fields, cards, and navigation drawers.**

✅ **Material Design themes and colors help maintain consistency across applications.**

✅ **Using Material Design improves accessibility, responsiveness, and app branding.**

CUSTOMIZING UI WITH XML & STYLES IN ANDROID DEVELOPMENT

CHAPTER 1: INTRODUCTION TO UI CUSTOMIZATION IN ANDROID

1.1 Why Customize UI in Android?

🔹 **User Interface (UI) customization** enhances the visual appeal and usability of an Android application.

🔹 Android allows UI customization using **XML styles, themes, and attributes**, making it easy to maintain consistency across the app.

✅ **Benefits of UI Customization:**

✓ **Improves User Experience (UX)** – Aesthetic and intuitive designs attract users.

✓ **Ensures Consistency** – Reusing styles helps maintain a unified look.

✓ **Reduces Code Duplication** – Styles and themes prevent repetitive XML code.

✓ **Supports Multiple Screen Sizes** – Ensures adaptability across devices.

📌 **Example:**

- **A banking app applies a professional blue color theme with consistent typography across all screens.**

CHAPTER 2: UNDERSTANDING XML STYLING IN ANDROID

2.1 What is XML Styling in Android?

🔹 **XML (Extensible Markup Language)** is used to define **layouts, styles, and themes** for Android UI components.

🔹 Instead of defining UI properties in each layout file, styles allow **reusable design patterns** across multiple views.

✅ **Common Customizable UI Properties in XML:**

✓ **Text Appearance** – textColor, textSize, fontFamily

✓ **Backgrounds & Borders** – background, cornerRadius, strokeWidth

✓ **Padding & Margins** – padding, layout_margin

✓ **Animations & Shadows** – elevation, animation

📌 **Example:**

- **A custom button style defines background color, text size, and corner radius in XML.**

CHAPTER 3: DEFINING STYLES IN ANDROID

3.1 What Are Styles in Android?

🔹 A **style** in Android is a reusable collection of **UI attributes** applied to multiple UI components.

🔹 Styles are defined in the **res/values/styles.xml** file.

✅ **Advantages of Using Styles:**

✓ Reduces redundancy by defining UI properties once.

✓ Simplifies maintenance and consistency across screens.

✓ Allows easy updates to UI without modifying multiple files.

3.2 Creating a Custom Style in XML

📌 **Example – Defining a Style in styles.xml:**

<resources>

```
    <style name="CustomButton">
        <item name="android:background">#6200EE</item>
        <item name="android:textColor">#FFFFFF</item>
        <item name="android:padding">12dp</item>
        <item name="android:textSize">16sp</item>
        <item name="android:cornerRadius">8dp</item>
    </style>
</resources>
```

🔹 **Explanation:**

- The CustomButton style defines **background color, text color, padding, text size, and rounded corners**.

3.3 Applying Styles to a UI Component

📌 **Example – Applying a Style to a Button in XML:**

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click Me"
    style="@style/CustomButton"/>
```

📌 **Example – Applying Style Programmatically (Java/Kotlin):**

```
button.setTextAppearance(R.style.CustomButton);
button.setTextAppearance(R.style.CustomButton)
```

📌 **Example:**

- **A shopping app applies a "CustomButton" style to all "Buy Now" buttons to maintain a consistent UI.**

CHAPTER 4: USING THEMES FOR GLOBAL UI CUSTOMIZATION

4.1 What Are Themes in Android?

🔹 A **theme** is a collection of styles applied to an **entire app or activity**.

🔹 Themes are defined in **styles.xml** and referenced in the **AndroidManifest.xml** file.

✅ **Benefits of Using Themes:**

✓ Ensures **consistent branding** across all screens.

✓ Allows **dark mode and light mode** customization.

✓ Makes UI updates easier by modifying a single theme file.

4.2 Defining a Custom Theme in XML

📌 **Example – Defining a Custom Theme in styles.xml:**

```
<resources>
  <style name="Theme.MyApp"
parent="Theme.MaterialComponents.Light.DarkActionBar">
    <item name="colorPrimary">#6200EE</item>
    <item name="colorPrimaryVariant">#3700B3</item>
    <item name="colorAccent">#03DAC5</item>
    <item name="android:windowBackground">#FFFFFF</item>
```

```
    <item
name="android:statusBarColor">?attr/colorPrimaryVariant</item>
  </style>
</resources>
```

- **Explanation:**
  - The custom theme **inherits from Material Design's Light Theme**.
  - It customizes **primary colors, accent color, and background color**.

4.3 Applying a Theme to the App

📌 **Example – Setting a Theme in AndroidManifest.xml:**

```
<application
  android:theme="@style/Theme.MyApp">
</application>
```

📌 **Example – Applying a Theme Programmatically:**

```
setTheme(R.style.Theme_MyApp);
```

📌 **Example:**

- **A productivity app offers Light and Dark themes, allowing users to switch dynamically.**

CHAPTER 5: CUSTOMIZING UI WITH DRAWABLE RESOURCES

5.1 Using Drawable Resources for UI Customization

🔹 **Drawable resources** define **custom backgrounds, gradients, and borders** for UI components.

🔹 They are stored in the **res/drawable/** directory and referenced in XML.

✅ **Types of Drawables:**

✓ **Shape Drawables** – Define backgrounds and borders.

✓ **State Drawables** – Change appearance based on UI state (pressed, focused).

✓ **Gradient Drawables** – Create gradient backgrounds.

5.2 Creating a Custom Background Using Shape Drawable

📌 **Example – Creating a Rounded Button Background (rounded_button.xml):**

```
<shape
xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle">
  <corners android:radius="16dp"/>
  <solid android:color="#6200EE"/>
  <stroke android:width="2dp" android:color="#3700B3"/>
</shape>
```

📌 **Example – Applying the Drawable to a Button:**

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Click Me"
```

android:background="@drawable/rounded_button"/>

📌 **Example:**

- **A messaging app applies rounded corners and gradient colors to buttons for a modern look.**

Exercise: Test Your Understanding

🔹 **What is the difference between a style and a theme in Android?**

🔹 **How do you apply a custom style to a TextView?**

🔹 **What is the purpose of styles.xml in Android?**

🔹 **Explain how to use drawable resources to customize UI components.**

🔹 **How would you implement dark mode using themes in an Android app?**

Conclusion

✅ **Styles allow developers to define reusable UI properties for individual components.**

✅ **Themes apply a consistent UI across the entire app, supporting dark and light modes.**

✅ **Drawable resources provide custom shapes, gradients, and states for UI components.**

✅ **By mastering XML styling, developers can create professional and visually appealing Android applications.**

Animations, MotionLayout, and Transitions in Android Development

Chapter 1: Introduction to Animations in Android

1.1 What Are Animations in Android?

🔹 **Animations in Android** enhance user experience by adding smooth, visually appealing transitions to UI components.

🔹 They help in creating **dynamic interactions, visual feedback, and fluid UI flows**.

🔹 Android provides multiple **animation frameworks** for different types of animations.

✅ **Why Use Animations?**

✓ **Improves user engagement** with smooth UI interactions.

✓ **Enhances visual aesthetics** of the app.

✓ **Provides feedback** to users, improving usability.

📌 **Example:**

- **Expanding a card view when clicked, sliding in a menu, or animating a button press.**

1.2 Types of Animations in Android

Animation Type	Description	Example Use Case
Property Animations	Animates properties like position, scale, alpha, rotation.	Animating a button fade-in effect.

View Animations	Applies predefined transformations like translate, scale, rotate.	Moving a view from left to right.
Drawable Animations	Frame-by-frame animations using drawable resources.	Animated loading indicators.
MotionLayout Animations	Advanced animations using MotionLayout.	Complex UI transitions.
Activity Transitions	Animations between different activities/fragments.	Shared element transitions.

📌 **Example:**

- **A floating action button (FAB) animating on click to reveal additional options.**

CHAPTER 2: PROPERTY ANIMATIONS (OBJECTANIMATOR)

2.1 What Are Property Animations?

🔹 **Property animations modify object properties (e.g., translation, rotation, scale, opacity) over time.**

🔹 The main class used for property animations is ObjectAnimator.

✅ **Key Features:**

✓ Supports **alpha (opacity), translation (movement), rotation, and scale animations**.

✓ Allows **custom interpolators** for smooth motion effects.

✓ Provides **chaining multiple animations** using AnimatorSet.

2.2 Implementing ObjectAnimator in Android

✅ Fade-in animation using ObjectAnimator

```
ObjectAnimator fadeIn = ObjectAnimator.ofFloat(view, "alpha", of, 1f);

fadeIn.setDuration(1000);

fadeIn.start();
```

✅ Moving a Button from Left to Right

```
ObjectAnimator moveRight = ObjectAnimator.ofFloat(button, "translationX", of, 300f);

moveRight.setDuration(1000);

moveRight.start();
```

✅ Scaling an ImageView

```
ObjectAnimator scaleX = ObjectAnimator.ofFloat(imageView, "scaleX", 1f, 1.5f);

ObjectAnimator scaleY = ObjectAnimator.ofFloat(imageView, "scaleY", 1f, 1.5f);

AnimatorSet set = new AnimatorSet();

set.playTogether(scaleX, scaleY);

set.setDuration(1000);

set.start();
```

📌 Example:

- **A profile picture enlarges when clicked using a scale animation.**

CHAPTER 3: VIEW ANIMATIONS (TWEEN ANIMATIONS)

3.1 What Are View Animations?

- **View animations apply transformations (move, rotate, fade, scale) on UI elements.**
- These animations use **XML-defined animation files** or Animation classes.

✅ **Types of View Animations:**

✓ **TranslateAnimation** – Moves a view from one position to another.

✓ **ScaleAnimation** – Enlarges or shrinks a view.

✓ **RotateAnimation** – Rotates a view.

✓ **AlphaAnimation** – Changes view transparency.

3.2 Implementing View Animations (XML-based)

✅ **Create a translate animation (slide effect) in XML**

📌 **res/anim/slide_right.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<translate
xmlns:android="http://schemas.android.com/apk/res/android"
  android:fromXDelta="-100%"
  android:toXDelta="0%"
  android:duration="500"/>
```

✅ **Apply the animation in Java**

```
Animation slideRight = AnimationUtils.loadAnimation(this, R.anim.slide_right);
```

view.startAnimation(slideRight);

📌 **Example:**

- **A menu sliding in from the left when opened.**

CHAPTER 4: MOTIONLAYOUT IN ANDROID

4.1 What is MotionLayout?

🔹 **MotionLayout** is an advanced animation system introduced in **ConstraintLayout 2.0**.

🔹 It allows **complex animations and transitions between UI states**.

✅ **Why Use MotionLayout?**

✓ Creates smooth transitions between layouts.

✓ Eliminates the need for multiple manual animations.

✓ Uses declarative XML for defining animations.

4.2 Implementing MotionLayout Transitions

✅ **Step 1: Add MotionLayout to XML**

```
<androidx.constraintlayout.motion.widget.MotionLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  android:id="@+id/motionLayout"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
```

```
  <ImageView
    android:id="@+id/imageView"
    android:src="@drawable/ic_launcher"
    android:layout_width="100dp"
    android:layout_height="100dp"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintStart_toStartOf="parent"/>
</androidx.constraintlayout.motion.widget.MotionLayout>
```

✅ **Step 2: Define MotionScene for Transition**

📌 **res/xml/motion_scene.xml**

```
<MotionScene
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto">

  <Transition
    app:constraintSetStart="@+id/start"
    app:constraintSetEnd="@+id/end"
    app:duration="1000">
    <OnSwipe app:touchAnchorId="@id/imageView"
app:touchAnchorSide="top"/>
  </Transition>

  <ConstraintSet android:id="@+id/start">
```

<Constraint android:id="@+id/imageView"
app:layout_constraintTop_toTopOf="parent"/>

</ConstraintSet>

<ConstraintSet android:id="@+id/end">

<Constraint android:id="@+id/imageView"
app:layout_constraintBottom_toBottomOf="parent"/>

</ConstraintSet>

</MotionScene>

📌 **Example:**

- **A search bar smoothly expands and collapses when clicked.**

CHAPTER 5: ACTIVITY & FRAGMENT TRANSITIONS

5.1 What Are Activity Transitions?

🔹 Android supports **smooth transitions between activities/fragments,** creating a natural app flow.

🔹 These transitions are part of the **Material Design guidelines**.

✅ **Types of Transitions:**

✓ **Explode** – Expands the activity from the center.

✓ **Slide** – Slides the activity from left/right.

✓ **Fade** – Gradually fades out/in.

✅ **Adding a Slide Transition Between Activities**

@Override

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   getWindow().setEnterTransition(new Slide(Gravity.RIGHT));
   getWindow().setExitTransition(new Slide(Gravity.LEFT));
}
```

📌 **Example:**

- **A details page smoothly sliding in from the right when an item is clicked.**

Exercise: Test Your Understanding

🔹 **What is the difference between ObjectAnimator and MotionLayout?**
🔹 **How does RecyclerView use animations?**
🔹 **Write a basic XML-based animation for a fade effect.**
🔹 **What are the advantages of using MotionLayout over manual animations?**
🔹 **Implement a sliding transition between two fragments.**

Conclusion

✅ **Animations improve user experience by making UI interactions more dynamic.**
✅ **Android provides multiple animation techniques – ObjectAnimator, View Animations, MotionLayout, and Transitions.**
✅ **MotionLayout is a powerful tool for creating complex UI**

animations with ease.

✅ **Activity transitions enhance the flow of an app and make navigation smooth.**

ASSIGNMENT:

CREATE A **LOGIN SCREEN UI** FOLLOWING MATERIAL DESIGN GUIDELINES

Assignment Solution: Create a Login Screen UI Following Material Design Guidelines

📌 Step-by-Step Guide to Building a Material Design Login Screen in Android

In this assignment, we will create a **Login Screen UI** using **Material Design components** in **Android Studio**. We will use **EditText, Button, TextInputLayout, and Material Components** to enhance the user experience.

📌 Step 1: Set Up a New Android Project

1️⃣ **Open Android Studio** and create a new project.
2️⃣ Select **Empty Activity** and click **Next**.
3️⃣ Set the project name as **MaterialLoginScreen**.
4️⃣ Choose **Kotlin** as the language and click **Finish**.

📌 Step 2: Add Material Design Dependencies

1️⃣ Open **build.gradle (Module: app)** and add the following dependencies inside dependencies:

```
dependencies {
    implementation 'com.google.android.material:material:1.9.0'
}
```

2️⃣ **Sync the project** to apply changes.

📌 Step 3: Modify activity_login.xml Layout

🔹 We will use **TextInputLayout** for username and password fields to provide a cleaner UI with Material Design styling.

📍 Code: activity_login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".LoginActivity"
    android:background="@color/white">

    <!-- App Logo -->
    <ImageView
        android:id="@+id/logo"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:src="@drawable/ic_login_logo"
        android:contentDescription="Login Logo"
        app:layout_constraintTop_toTopOf="parent"
```

```
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="60dp" />

<!-- Username Input -->
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/usernameInputLayout"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:boxStrokeColor="@color/black"
    android:hint="Username"
    app:layout_constraintTop_toBottomOf="@id/logo"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="20dp"
    android:layout_marginHorizontal="30dp">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/usernameEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textEmailAddress"/>
```

```
</com.google.android.material.textfield.TextInputLayout>


<!-- Password Input -->
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/passwordInputLayout"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:boxStrokeColor="@color/black"
    android:hint="Password"
    app:passwordToggleEnabled="true"

app:layout_constraintTop_toBottomOf="@id/usernameInputLayou
t"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="10dp"
    android:layout_marginHorizontal="30dp">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/passwordEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword"/>
```

```
</com.google.android.material.textfield.TextInputLayout>

<!-- Login Button -->
<com.google.android.material.button.MaterialButton
    android:id="@+id/loginButton"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Login"
    android:backgroundTint="@color/teal_700"
    app:cornerRadius="12dp"
    android:textColor="@color/white"

app:layout_constraintTop_toBottomOf="@id/passwordInputLayout
"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="20dp"
    android:layout_marginHorizontal="30dp"/>

<!-- Forgot Password Text -->
<TextView
    android:id="@+id/forgotPasswordText"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="Forgot Password?"
    android:textColor="@color/black"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/loginButton"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="10dp"
    android:layout_marginEnd="30dp"/>

  <!-- Sign Up Text -->
  <TextView
    android:id="@+id/signUpText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Don't have an account? Sign Up"
    android:textColor="@color/black"
    android:textSize="14sp"
    android:textStyle="bold"

app:layout_constraintTop_toBottomOf="@id/forgotPasswordText"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="10dp"/>
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

📌 Step 4: Implement Login Button Logic in LoginActivity.kt

1️⃣ Open **LoginActivity.kt** and modify the onCreate method.
2️⃣ Add event listeners to validate input fields.

📌 Code: LoginActivity.kt

```
package com.example.materialloginscreen


import android.os.Bundle
import android.text.TextUtils
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.google.android.material.button.MaterialButton
import com.google.android.material.textfield.TextInputEditText


class LoginActivity : AppCompatActivity() {

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


        val usernameEditText =
findViewById<TextInputEditText>(R.id.usernameEditText)
```

```
    val passwordEditText =
findViewById<TextInputEditText>(R.id.passwordEditText)

    val loginButton =
findViewById<MaterialButton>(R.id.loginButton)


    loginButton.setOnClickListener {

      val username = usernameEditText.text.toString()

      val password = passwordEditText.text.toString()


      if (TextUtils.isEmpty(username) ||
TextUtils.isEmpty(password)) {

        Toast.makeText(this, "Please enter both username and
password", Toast.LENGTH_SHORT).show()

      } else if (username == "admin" && password == "password123")
{

        Toast.makeText(this, "Login Successful",
Toast.LENGTH_SHORT).show()

      } else {

        Toast.makeText(this, "Invalid Credentials",
Toast.LENGTH_SHORT).show()

      }

    }

  }

}
```

📌 Step 5: Run and Test the Login Screen

1️⃣ **Run the application** in an emulator or real device.

2️⃣ **Enter credentials:**

- Username: admin
- Password: password123

3️⃣ **Check different cases:**

- Empty fields should show an error.
- Incorrect credentials should show "Invalid Credentials".
- Correct credentials should display "Login Successful".

📌 Additional Enhancements

- 🔹 **Add Google Sign-In Button** using Firebase Authentication.
- 🔹 **Animate UI elements** for a smoother experience.
- 🔹 **Implement Dark Mode Support** using MaterialTheme.
- 🔹 **Enhance Security** by integrating a hashed password authentication system.

📌 Exercise: Test Your Understanding

- 🔹 **What is the purpose of TextInputLayout in Material Design?**
- 🔹 **How does TextInputEditText differ from a normal EditText?**
- 🔹 **Why is TextUtils.isEmpty() used in validation?**
- 🔹 **How can you add animations to the login button?**
- 🔹 **What are some ways to enhance user experience in a login UI?**

📌 Conclusion

✅ **We successfully built a Material Design Login Screen using RecyclerView, TextInputLayout, and Material Buttons.**

✅ **We implemented input validation to improve user experience.**

✅ **This UI follows Material Design guidelines, making it visually appealing and accessible.**