Q1. Create an Android project with a minimum SDK of 21. Design a simple app that displays "Hello, World!" on the screen.

AIM

To understand the process of creating an Android project using Android Studio and implementing a basic mobile application.

Prerequisites

Before creating an Android project, ensure that the following prerequisites are met:

- Android Studio: Installed and configured on your system.
- Java Development Kit (JDK): Required for Java-based Android development.
- Android SDK: Includes essential libraries and tools.
- Emulator or Physical Device: For testing the application.

Steps to Create an Android Project

1. Open Android Studio
2. Create a New Project
    a. Click on 'Start a new Android Studio project' and press next
3. Configure Your Project
    a. Name: Provide a name for your application.
    b. Package Name: Define a unique package identifier (e.g., com.example.myapp).
    c. Save Location: Choose a directory where the project will be saved.
    d. Language: Select the programming language (Java/Kotlin).
    e. Minimum API Level: Choose the lowest Android version that your app will support.
    f. Click Finish to create the project.
4. Build and Run the Application

Code:

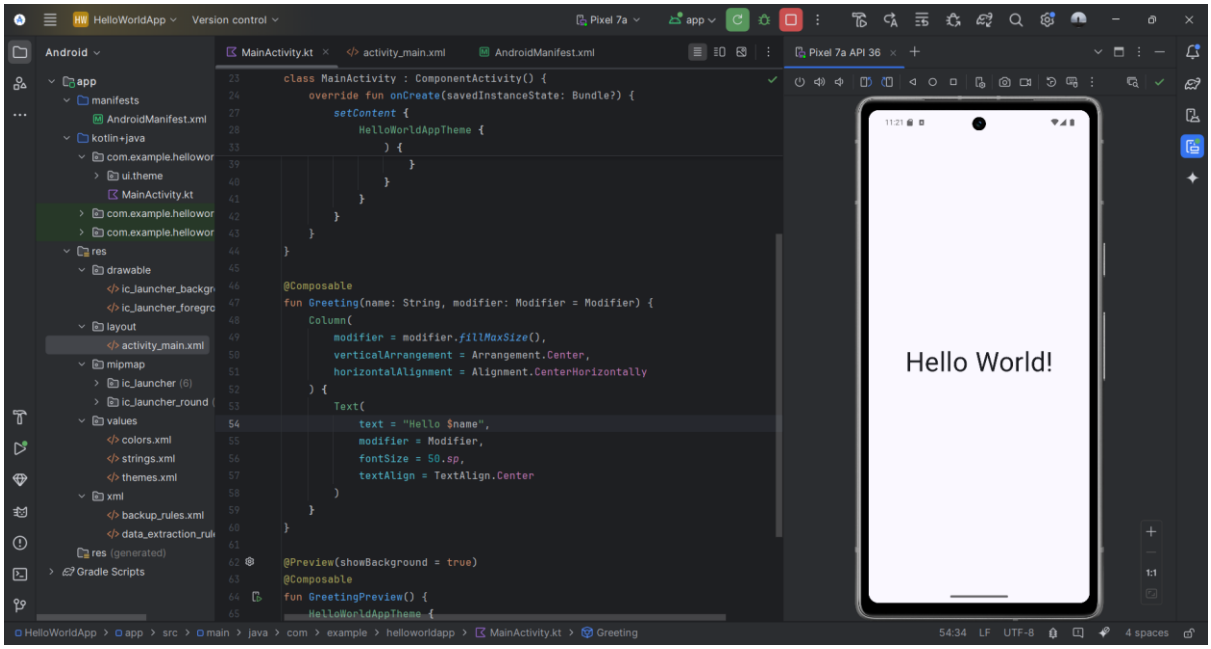
MainActivity.kt

```
class MainActivity :
ComponentActivity() {
  override fun
onCreate(savedInstanceState:
Bundle?) {

super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    setContent {
      HelloWorldAppTheme {
        // A surface container using
the 'background' color from the theme
        Surface(
          modifier =
Modifier.fillMaxSize(),
          color =
MaterialTheme.colorScheme.backgro
und
        ) {
          Scaffold(modifier =
```

```
Modifier.fillMaxSize()) { innerPadding ->
                Greeting(
                  name = "World!",
                  modifier =
Modifier.padding(innerPadding)
                )
              }
            }
          }
        }
    }
}

@Composable
fun Greeting(name: String, modifier:
Modifier = Modifier) {
    Column(
      modifier = modifier.fillMaxSize(),
      verticalArrangement =
```

```
Arrangement.Center,
      horizontalAlignment =
Alignment.CenterHorizontally
    ) {
      Text(
        text = "Hello $name",
        modifier = Modifier,
        fontSize = 50.sp,
        textAlign = TextAlign.Center
      )
    }
}

@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    HelloWorldAppTheme {
      Greeting("World!")
    }
}
```

Output:



Q2: Identify and explain the purpose of the AndroidManifest.xml file in your Kotlin project. Modify it to request INTERNET permission.

Purpose of AndroidManifest.xml

1. Acts as the **configuration file** for an Android application.

2. **Declares** essential information about the app to the Android system before any app code is run.
3. **Specifies** permissions, activities, services, content providers, and broadcast receivers.
4. **Defines** application-level properties like the app's name, icon, theme, and supported Android versions.

Without the AndroidManifest.xml, the app cannot be installed or executed properly on an Android device.

STEPS to Modify AndroidManifest.xml to Request INTERNET Permission

1. **Open** the AndroidManifest.xml file located inside the app/src/main directory of your project.
2. **Add the INTERNET permission** inside the <manifest> tag, but **outside** the <application> tag:

<uses-permission android:name="android.permission.INTERNET" />

3. **Ensure** the permission line is placed before the <application> tag starts.
4. **Save** the file and rebuild the project to apply the changes.
5. **Test** by running the app and verifying if the app can now access the internet (for example, by using an API or loading web content).

Code: AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.helloworldapp">

  <!-- Permission to access the internet -->
  <uses-permission android:name="android.permission.INTERNET" />

  <application
    android:allowBackup="true"

android:icon="@mipmap/ic_launcher"

android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"

android:theme="@style/Theme.HelloWorldApp">

    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

  </application>

</manifest>
```

Q3: Create an activity that logs lifecycle method calls (onCreate, onStart, etc.) in the Logcat.

STEPS

1. **Open** Android Studio and create or open your project.
2. **Create** or **modify** the MainActivity.kt file by overriding the lifecycle methods:
    - onCreate()
    - onStart()
    - onResume()
    - onPause()
    - onStop()
    - onDestroy()
3. **Inside each method**, add a log statement using Log.d(tag, "methodName() called"), where tag is a constant (e.g., "MainActivityLifecycle").
4. **Set up** a basic UI using setContent and a Greeting composable function to display "Hello World!".
5. **Build and run** the application on an emulator or physical device.
6. **Open** the **Logcat** panel at the bottom of Android Studio.
7. **Filter** the logs by the tag name (MainActivityLifecycle) to easily view your lifecycle method calls.
8. **Interact** with the app (open, minimize, rotate, close) and observe how the lifecycle method calls appear in the Logcat.

Code: MainActivity.kt

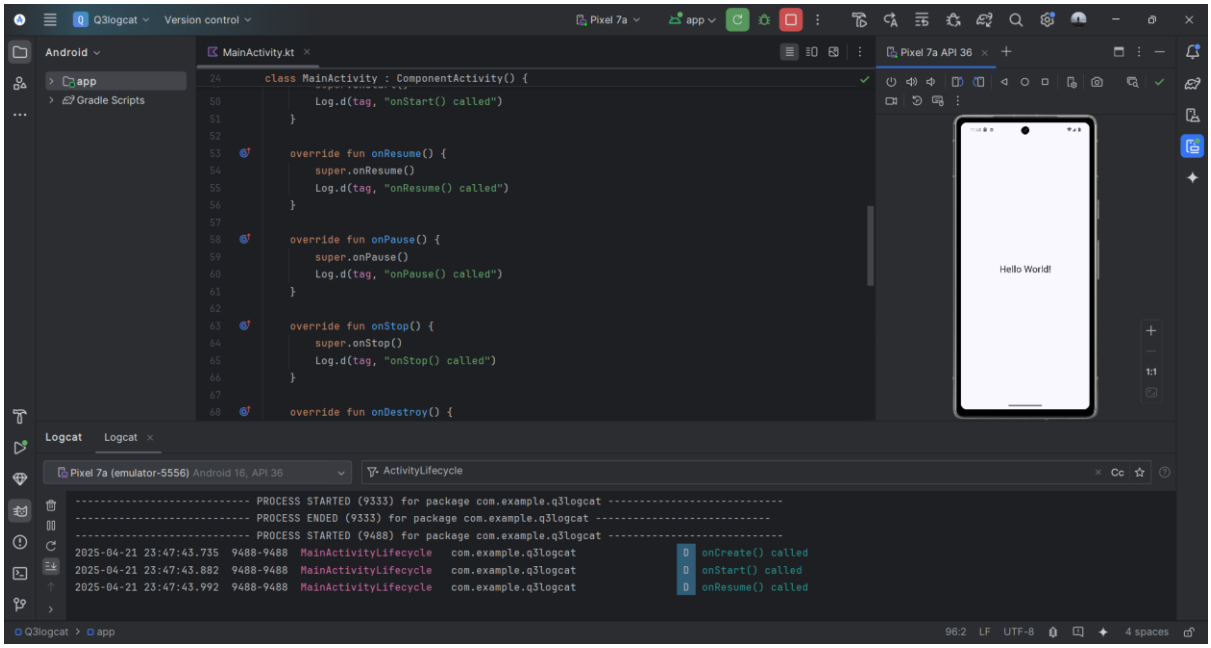
```
class MainActivity :
ComponentActivity() {
   private val tag =
"MainActivityLifecycle"

   override fun
onCreate(savedInstanceState:
Bundle?) {

super.onCreate(savedInstanceState)
      enableEdgeToEdge()
      Log.d(tag, "onCreate() called")
      setContent {
         Q3logcatTheme {
            Surface(
               modifier =
Modifier.fillMaxSize(),
               color =
MaterialTheme.colorScheme.backgro
und
            ) {
               Scaffold(modifier =
Modifier.fillMaxSize()) { innerPadding -
>
                  Greeting(
                     name = "World!",
                     modifier =
Modifier.padding(innerPadding)
                  )
               }
            }
         }
      }
   }

   override fun onStart() {
      super.onStart()
      Log.d(tag, "onStart() called")
   }
```

```
    override fun onResume() {
        super.onResume()
        Log.d(tag, "onResume() called")
    }
    override fun onPause() {
        super.onPause()
        Log.d(tag, "onPause() called")
    }
    override fun onStop() {
        super.onStop()
        Log.d(tag, "onStop() called")
    }
    override fun onDestroy() {
        super.onDestroy()
        Log.d(tag, "onDestroy() called")
    }
}
@Composable
fun Greeting(name: String, modifier:
Modifier = Modifier) {
    Column(
        modifier = modifier.fillMaxSize(),
        verticalArrangement =
Arrangement.Center,
        horizontalAlignment =
Alignment.CenterHorizontally
    ) {
        Text(
            text = "Hello $name",
            modifier = Modifier,
            fontSize = 30.sp,
            textAlign = TextAlign.Center
        )
    }
}
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    Q3logcatTheme {
        Greeting("World!")
    }
}
```

How to View Logs in Logcat:

1. Open Android Studio.
2. Go to Logcat at the bottom of the IDE.
3. Filter by tag: ActivityLifecycle.
4. Run the app and interact with it (minimize, reopen, rotate screen, etc.) to see the lifecycle logs.

Output:



Q4. Design an app with two activities and use lifecycle methods to save and restore a

STEPS

1. **Open** Android Studio and **create a new project** with an Empty Activity template.
2. **Create two activities**: MainActivity.kt and SecondActivity.kt.
3. **In MainActivity**:
    - Implement a **counter** using rememberSaveable to automatically save and restore the counter value across configuration changes (like screen rotation).
    - Use lifecycle methods (onCreate, onStart, onResume, onPause, onStop, onDestroy) and log their calls using Log.d() for monitoring the activity state.
    - Add a **Button** to **increment** the counter and another Button to **navigate** to SecondActivity using an **Intent**.
4. **In SecondActivity**:
    - Implement a simple screen that displays a greeting ("Hello Second Activity").
    - Override lifecycle methods (onCreate, onStart, etc.) and add corresponding Log.d() calls for each.
5. **Modify** the AndroidManifest.xml:
    - Declare both activities (MainActivity and SecondActivity).
    - Set MainActivity as the **launcher activity** using an <intent-filter>.
6. **Run the application**:
    - Observe the counter value persists when rotating the device or switching between activities.

- Monitor the lifecycle method logs in **Logcat** to understand the activity transitions.

Code: **MainActivity.kt**

```
class MainActivity :
ComponentActivity() {
  private val tag =
"MainActivityLifecycle"
  override fun
onCreate(savedInstanceState:
Bundle?) {
 super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    Log.d(tag, "onCreate() called")
    setContent {
      Q4twoactivitiesTheme {
        Surface(modifier =
Modifier.fillMaxSize()) {
          Scaffold(modifier =
Modifier.fillMaxSize()) { innerPadding -
>
MainScreen(Modifier.padding(innerPa
dding)) }  }} }}
  override fun onStart() { Log.d(tag,
"onStart() called") }
  override fun onResume() { Log.d(tag,
"onResume() called") }
  override fun onPause() { Log.d(tag,
"onPause() called") }
  override fun onStop() { Log.d(tag,
"onStop() called") }
  override fun onDestroy() { Log.d(tag,
"onDestroy() called") }
}
@Composable
fun MainScreen(modifier: Modifier =
Modifier) {
  var counter by rememberSaveable {
mutableIntStateOf(0) }
  val context = LocalContext.current
  Column(
    modifier = modifier.fillMaxSize(),
    verticalArrangement =
Arrangement.Center,
    horizontalAlignment =
Alignment.CenterHorizontally
  ) {
    Text(text = "Counter: $counter",
fontSize = 30.sp)
    Button(onClick = { counter++ }) {
Text("Increment Counter") }
    Spacer(modifier =
Modifier.height(16.dp))
    Button(onClick = {
      val intent = Intent(context,
SecondActivity::class.java)
      context.startActivity(intent)
    }) { Text("Go to Second Activity") }}}
```

SecondActivity.kt

```
class SecondActivity :
ComponentActivity() {
  private val tag =
"SecondActivityLifecycle"

  override fun
onCreate(savedInstanceState:
Bundle?) {

super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    Log.d(tag, "onCreate() called")
    setContent {
      Q4twoactivitiesTheme {
        Surface(modifier =
Modifier.fillMaxSize()) {
          Scaffold(modifier =
Modifier.fillMaxSize()) { innerPadding -
>Greeting(
            name = "Second
Activity",
```

```
                    modifier =
Modifier.padding(innerPadding)
  }}  } }}
   override fun onStart() { Log.d(tag,
"onStart() called") }
   override fun onResume() { Log.d(tag,
"onResume() called") }
   override fun onPause() { Log.d(tag,
"onPause() called") }
   override fun onStop() { Log.d(tag,
"onStop() called") }
   override fun onDestroy() { Log.d(tag,
"onDestroy() called") }
}
```

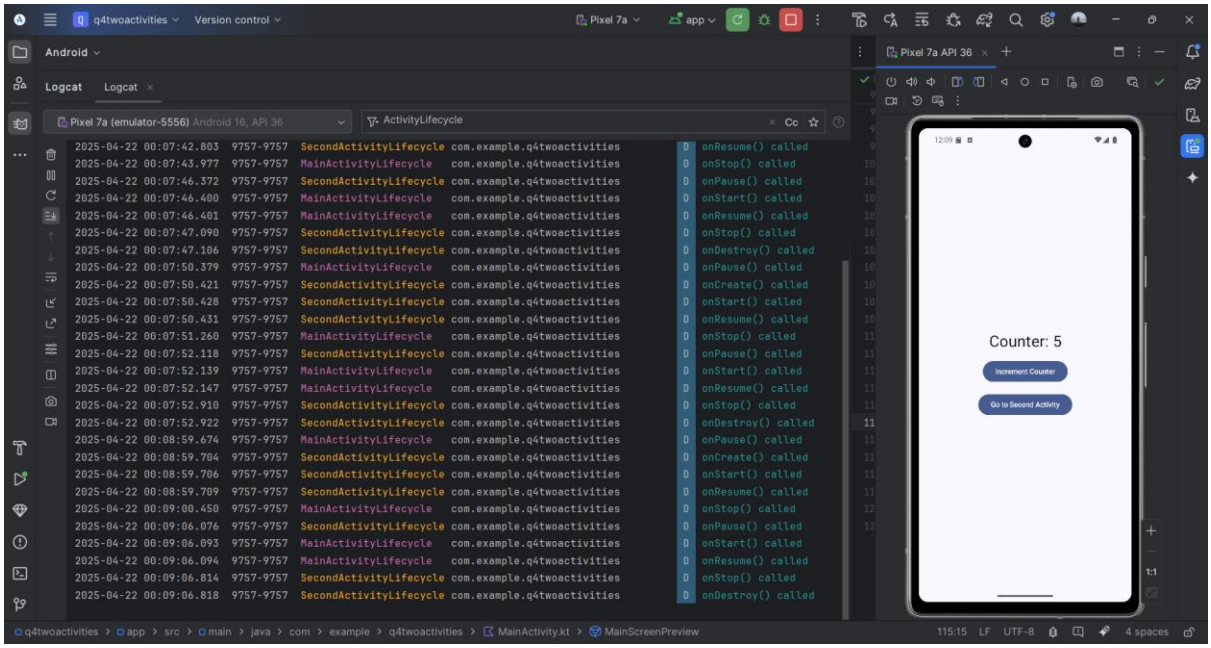
```
@Composable
fun Greeting(name: String, modifier:
Modifier = Modifier) {
   Column(
      modifier = modifier.fillMaxSize(),
      verticalArrangement =
Arrangement.Center,
      horizontalAlignment =
Alignment.CenterHorizontally
   ) {
      Text(text = "Hello $name", fontSize
= 30.sp, textAlign = TextAlign.Center)
   }}
```

AndroidManifest.xml

```
<manifest
xmlns:android="http://schemas.andr
oid.com/apk/res/android"

package="com.example.q4twoactiviti
es">

   <uses-sdk
android:minSdkVersion="24"
android:targetSdkVersion="35" />

   <application

android:label="@string/app_name"

android:icon="@mipmap/ic_launcher
"

android:theme="@style/Theme.Q4tw
oactivities">
```

```
      <activity
android:name="com.example.q4twoa
ctivities.SecondActivity" />
      <activity
android:name="com.example.q4twoa
ctivities.MainActivity">
         <intent-filter>
            <action
android:name="android.intent.action.
MAIN"/>
            <category
android:name="android.intent.catego
ry.LAUNCHER"/>
         </intent-filter>
      </activity>
   </application>
</manifest>
```

Output:



Q5. Create an activity with two fragments (e.g., Fragment A and Fragment B) that communicate with each other.

STEPS

1. **Open** Android Studio and create a new project with an Empty Activity.
2. **Design** the layout activity_main.xml:
    - Add two FrameLayout containers side by side (using a horizontal LinearLayout) to host the two fragments.
3. **Create FragmentA** (FragmentA.kt):
    - Define an interface FragmentAListener with a method onMessageSent(message: String).
    - In onAttach(), check if the host activity implements this interface.
    - Create a UI with an EditText and a Button in fragment_a.xml.
    - When the button is clicked, send the text from EditText to the activity using listener?.onMessageSent(message).
4. **Create FragmentB** (FragmentB.kt):
    - Design a simple layout (fragment_b.xml) with a TextView to display the message.
    - Create a method updateMessage(message: String) to update the TextView content.
5. **Modify MainActivity** (MainActivity.kt):
    - Implement the FragmentAListener interface.
    - In onMessageSent(), find FragmentB and call its updateMessage(message) function.
6. **Run the application**:
    - Enter a message in Fragment A, click the "Send" button, and see the message appear in Fragment B.

Code: **MainActivity.kt**

```
class MainActivity :
AppCompatActivity(),
FragmentA.FragmentAListener {
   override fun
onCreate(savedInstanceState:
Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity_mai
n)



supportFragmentManager.beginTrans
action()

.replace(R.id.fragment_container_a,
FragmentA())
```

```
.replace(R.id.fragment_container_b,
FragmentB())
        .commit()
   }
   override fun
onMessageSent(message: String) {
      val fragmentB =
supportFragmentManager.findFragme
ntById(R.id.fragment_container_b)
as? FragmentB

fragmentB?.updateMessage(message
)
   }
}
```

FragmentA.kt

```
class FragmentA : Fragment()
   private var listener:
FragmentAListener? = null
   interface FragmentAListener {
      fun onMessageSent(message:
String)
   }
   override fun onAttach(context:
Context) {
      super.onAttach(context)
      if (context is FragmentAListener) {
         listener = context
      } else {
         throw
RuntimeException("$context must
implement FragmentAListener")
      }
   }
   override fun onCreateView(
      inflater: LayoutInflater, container:
ViewGroup?,
      savedInstanceState: Bundle?
   ): View? {
      val view =
```

```
inflater.inflate(R.layout.fragment_a,
container, false)
      val editText =
view.findViewById<EditText>(R.id.edit
_text)
      val button =
view.findViewById<Button>(R.id.butto
n_send)

      button.setOnClickListener {
         val message =
editText.text.toString()

listener?.onMessageSent(message)
      }
      return view
   }
   override fun onDetach() {
      super.onDetach()
      listener = null
   }
}
```

FragementB.kt

```
class FragmentB : Fragment() {
   private lateinit var textView:
TextView
   override fun onCreateView(
      inflater: LayoutInflater, container:
ViewGroup?,
      savedInstanceState: Bundle?
   ): View? {
      val view =
inflater.inflate(R.layout.fragment_b,
container, false)
      textView =
view.findViewById(R.id.text_view)
      return view
   }
   fun updateMessage(message:
String) {
      textView.text = message
   }
}
```

Activity_main.xml

```
<LinearLayout
xmlns:android="http://schemas.andr
oid.com/apk/res/android"
android:layout_width="match_parent
"
android:layout_height="match_parent
"
   android:orientation="horizontal">
   <FrameLayout
android:id="@+id/fragment_container
_a"
      android:layout_width="0dp"
android:layout_height="match_parent
"
      android:layout_weight="1" />
   <FrameLayout
android:id="@+id/fragment_container
_b"
      android:layout_width="0dp"

android:layout_height="match_parent
"
      android:layout_weight="1" />
</LinearLayout>
```

Fragement_a.xml

```
<LinearLayout
xmlns:android="http://schemas.andr
oid.com/apk/res/android"
android:layout_width="match_parent
"
android:layout_height="match_parent
"
   android:orientation="vertical"
   android:padding="16dp">
   <EditText
      android:id="@+id/edit_text"

android:layout_width="match_parent
"

android:layout_height="wrap_content
"
      android:hint="Enter message" />
   <Button
      android:id="@+id/button_send"

android:layout_width="wrap_content"
android:layout_height="wrap_content
"
      android:text="Send" />
</LinearLayout>
```

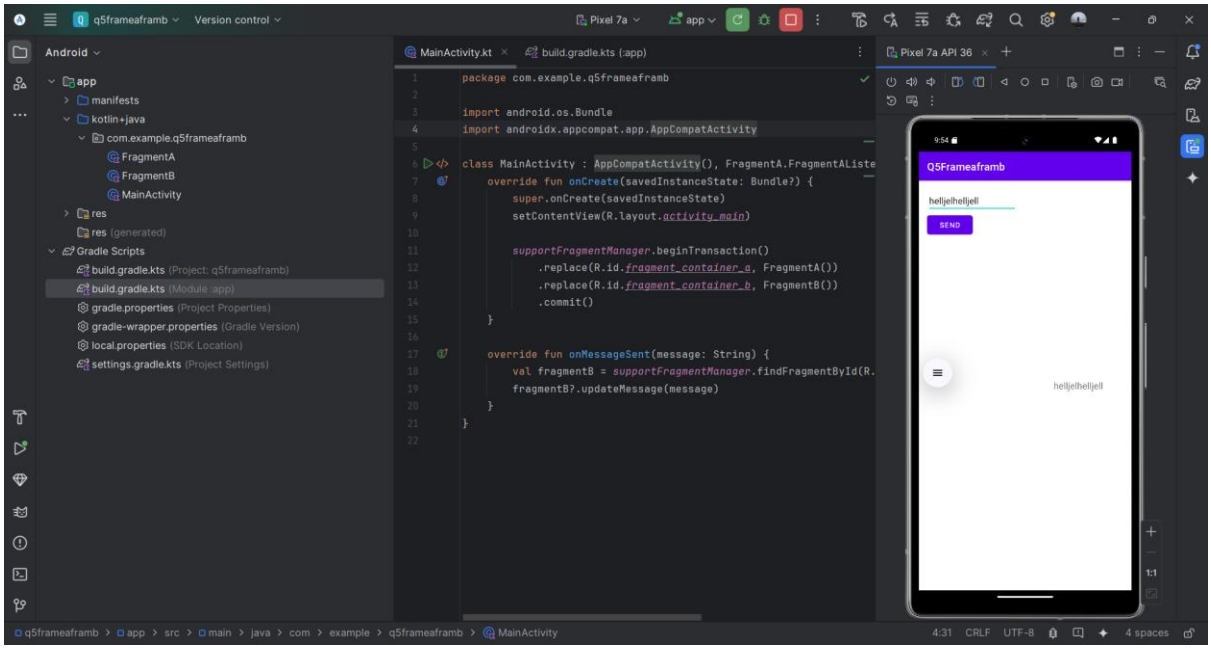
Fragment_b.xml

```
<LinearLayout
xmlns:android="http://schemas.andr
oid.com/apk/res/android"
```

```
android:layout_width="match_parent
"

android:layout_height="match_parent
"
   android:gravity="center"
   android:orientation="vertical">

   <TextView
      android:id="@+id/text_view"
```

```
android:layout_width="wrap_content"

android:layout_height="wrap_content
"
      android:text="Message will
appear here"
      android:textSize="18sp" />
</LinearLayout>
```

Output:



Q6: Implement lifecycle logging for a fragment and display the logs in a TextView inside the activity.

STEPS

1. **Create a new Android project** in Android Studio with an Empty Activity.
2. **Design the activity_main.xml layout**:
   - Add a TextView at the top to display lifecycle logs.
   - Add a FrameLayout below it to load the fragment dynamically.
3. **Create a fragment class MyFragment.java**:
   - Define an interface LifecycleLogger inside the fragment.
   - Call the onLogEvent() method of the activity from each lifecycle method (onAttach(), onCreate(), onCreateView(), etc.).
   - Also log lifecycle method calls using Log.d().
4. **Modify MainActivity.java**:
   - Implement the MyFragment.LifecycleLogger interface.

- In the onLogEvent() method, append each log message to the TextView.
- Load MyFragment into the fragmentContainer using a fragment transaction.

5. **Run the app**:
   - Watch the TextView update live with lifecycle method calls as you interact with the app (open, minimize, rotate, etc.).
6. **Test Lifecycle Events**:
   - Rotate the device, press home, reopen the app, etc., and observe how the fragment lifecycle methods are triggered and logged.

Code: MyFragment.java

```
public class MyFragment extends
Fragment {

  private LifecycleLogger logger;

  public interface LifecycleLogger {
    void onLogEvent(String message);
  }

  @Override
  public void onAttach(@NonNull
Context context) {
    super.onAttach(context);
    if (context instanceof
LifecycleLogger) {
      logger = (LifecycleLogger)
context;
    }
    log("onAttach");
  }

  @Override
  public void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);
    log("onCreate");
  }

  @Override
  public View
onCreateView(LayoutInflater inflater,
ViewGroup container,
                Bundle
savedInstanceState) {
    log("onCreateView");
    return
inflater.inflate(R.layout.fragment_my,
container, false);
  }

  @Override
  public void
onViewCreated(@NonNull View view,
@Nullable Bundle
savedInstanceState) {
    super.onViewCreated(view,
savedInstanceState);
    log("onViewCreated");
  }

  @Override
  public void onStart() {
    super.onStart();
    log("onStart");
  }

  @Override
  public void onResume() {
    super.onResume();
    log("onResume");
  }

  @Override
  public void onPause() {
    super.onPause();
    log("onPause");
  }

  @Override
```

```
    public void onStop() {
        super.onStop();
        log("onStop");
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        log("onDestroyView");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        log("onDestroy");
    }
```

```
    @Override
    public void onDetach() {
        super.onDetach();
        log("onDetach");
    }

    private void log(String
methodName) {
        if (logger != null) {
            logger.onLogEvent("Fragment: "
+ methodName);
        }
        Log.d("MyFragment",
methodName);
    }
}
```

MainActivity.java

```
public class MainActivity extends
AppCompatActivity implements
MyFragment.LifecycleLogger {

    private TextView logTextView;

    @Override
    protected void onCreate(Bundle
savedInstanceState) {
 super.onCreate(savedInstanceState);

setContentView(R.layout.activity_mai
n);
        logTextView =
findViewById(R.id.logTextView);

        // Load the fragment
```

```
        MyFragment fragment = new
MyFragment();

getSupportFragmentManager().beginT
ransaction()

.replace(R.id.fragmentContainer,
fragment)
                .commit();
    }
    @Override
    public void onLogEvent(String
message) {
        logTextView.append(message +
"\n");
    }
}
```

activity_main.xml

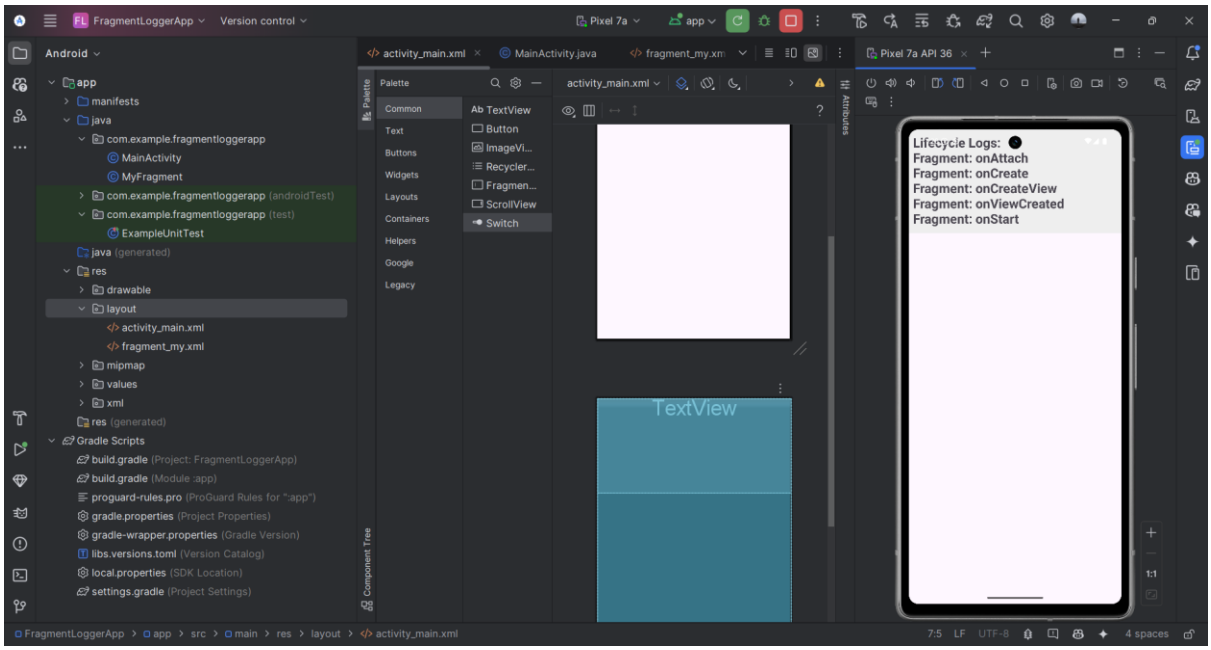
```
<?xml version="1.0" encoding="utf-
8"?>
<LinearLayout
xmlns:android="http://schemas.andr
oid.com/apk/res/android"
```

```
android:layout_width="match_parent
"

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

```
<TextView
    android:id="@+id/logTextView"

android:layout_width="match_parent
"
    android:layout_height="200dp"
    android:background="#EEE"
    android:padding="8dp"
    android:text="Lifecycle Logs:\n"
    android:scrollbars="vertical"
    android:textSize="14sp"/>
```

```
<FrameLayout

android:id="@+id/fragmentContainer
"

android:layout_width="match_parent
"
    android:layout_height="0dp"
    android:layout_weight="1"/>
</LinearLayout>
```

Output:



Q7: Design a layout using a LinearLayout to display a list of user profiles (name, email, and photo).

STEPS

1. **Create a new Android project** in Android Studio with an Empty Activity.
2. **Design the individual user profile layout**:
    - Create res/layout/user_profile_item.xml.
    - Use a horizontal LinearLayout containing an ImageView (for user photo) and a vertical LinearLayout (for name and email TextViews).
3. **Design the main activity layout**:
    - Create res/layout/activity_main.xml.
    - Add a vertical LinearLayout with id=profileContainer to dynamically add multiple user profiles inside it.
4. **Set up a simple User data class** in MainActivity.kt:

- Each User object holds a name, email, and a photo resource ID.

5. **In MainActivity**:
   - Inflate ActivityMainBinding using view binding.
   - Create a list of User objects.
   - For each user:
     - Inflate user_profile_item.xml.
     - Populate the ImageView and TextViews with user data.
     - Add the populated view into profileContainer.
6. **Add default resources**:
   - Use a default drawable (like default_user_photo.png) in res/drawable/.

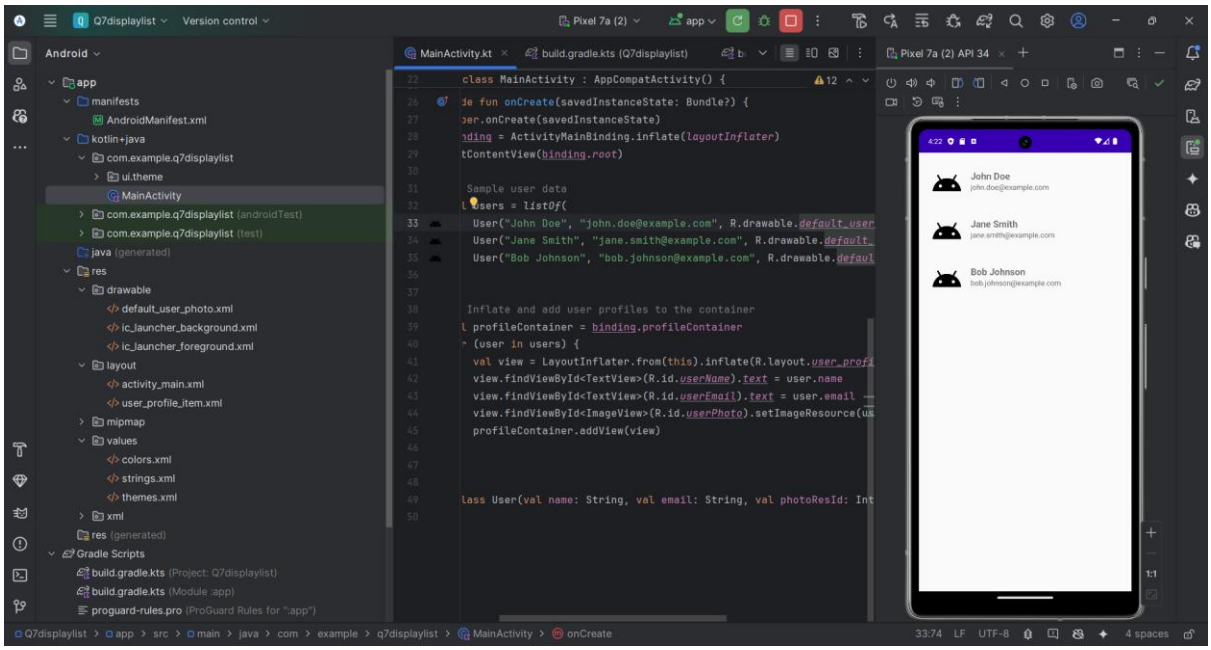
Code:

```
<!-- res/layout/user_profile_item.xml -->
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout_width="match_parent"

android:layout_height="wrap_content"
  android:orientation="horizontal"
  android:padding="16dp">

  <ImageView
    android:id="@+id/userPhoto"
    android:layout_width="60dp"
    android:layout_height="60dp"

android:contentDescription="User Profile Photo"

android:src="@drawable/default_user_photo"
    android:scaleType="centerCrop" />

  <LinearLayout

android:layout_width="match_parent"

android:layout_height="wrap_content"
    android:orientation="vertical"

android:layout_marginStart="16dp"

android:layout_gravity="center_vertical">

    <TextView
      android:id="@+id/userName"

android:layout_width="wrap_content"

android:layout_height="wrap_content"
      android:text="John Doe"
      android:textSize="18sp"
      android:textStyle="bold" />

    <TextView
      android:id="@+id/userEmail"

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:text="john.doe@example.com"
      android:textSize="14sp"
      android:textColor="#666666" />
  </LinearLayout>
```

```
</LinearLayout>

<!-- res/layout/activity_main.xml -->
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout_width="match_parent"

android:layout_height="match_parent"
   android:orientation="vertical"
   android:padding="8dp">

   <LinearLayout

android:id="@+id/profileContainer"

android:layout_width="match_parent"

android:layout_height="wrap_content"
      android:orientation="vertical" />
</LinearLayout>

<!-- res/values/themes.xml -->
<style name="Theme.Q7displaylist" parent="Theme.AppCompat.DayNight.NoActionBar">
   <item name="colorPrimary">@color/purple_500</item>
   <item name="colorPrimaryVariant">@color/purple_700</item>
   <item name="colorOnPrimary">@color/white</item>
   <item name="colorSecondary">@color/teal_200</item>
   <item name="colorSecondaryVariant">@color/teal_700</item>
   <item name="colorOnSecondary">@color/black</item>
   <item name="android:statusBarColor">?attr/colorPrimaryVariant</item>
</style>

<!-- res/values/colors.xml -->
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="purple_500">#FF6200EE</color>
   <color name="purple_700">#FF3700B3</color>
   <color name="teal_200">#FF03DAC5</color>
   <color name="teal_700">#FF018786</color>
   <color name="white">#FFFFFFFF</color>
   <color name="black">#FF000000</color>
</resources>

<!-- java/com/example/q7displaylist/MainActivity.kt -->
package com.example.q7displaylist

import android.os.Bundle
import android.view.LayoutInflater
import android.widget.LinearLayout
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import com.example.q7displaylist.databinding.ActivityMainBinding
```

```
class MainActivity :
AppCompatActivity() {

   private lateinit var binding:
ActivityMainBinding

   override fun
onCreate(savedInstanceState:
Bundle?) {

super.onCreate(savedInstanceState)
      binding =
ActivityMainBinding.inflate(layoutInfla
ter)
      setContentView(binding.root)

      val users = listOf(
         User("John Doe",
"john.doe@example.com",
R.drawable.default_user_photo),
         User("Jane Smith",
"jane.smith@example.com",
R.drawable.default_user_photo),
         User("Bob Johnson",
"bob.johnson@example.com",
R.drawable.default_user_photo)
      )

      val profileContainer =
binding.profileContainer
      for (user in users) {
         val view =
LayoutInflater.from(this).inflate(R.layo
ut.user_profile_item,
profileContainer, false)

view.findViewById<TextView>(R.id.use
rName).text = user.name

view.findViewById<TextView>(R.id.use
rEmail).text = user.email

view.findViewById<ImageView>(R.id.u
serPhoto).setImageResource(user.ph
otoResId)
         profileContainer.addView(view)
      }
   }

   data class User(val name: String, val
email: String, val photoResId: Int)
}
```

Output:



```
```

Q8: Use ConstraintLayout to create a responsive layout 10 that adapts to different screen sizes.

STEPS

1. **Create a new Android project** in Android Studio with an Empty Activity.
2. **Use ConstraintLayout** as the root layout in your activity_main.xml.
3. **Add an ImageView** for the app logo:
    - Set a fixed width and height (120dp x 120dp).
    - Center it horizontally (Start and End constraints to parent).
    - Vertically position it towards the top using layout_constraintVertical_bias="0.3".
4. **Add a TextView** for the title text:
    - Set the text to "Welcome!", make it bold with a larger font size (24sp).
    - Constrain its top to the bottom of the ImageView.
    - Center it horizontally (Start and End constraints to parent).
    - Adjust vertical bias slightly (layout_constraintVertical_bias="0.05") to give a nice gap.
5. **Add a Button** ("Get Started"):
    - Constrain its bottom to the parent's bottom.
    - Center it horizontally (Start and End constraints to parent).
    - Add some bottom margin (32dp) to keep distance from the screen edge.
6. **Make it responsive**:
    - Since ConstraintLayout uses relative positioning, the UI elements will automatically adapt across different screen sizes (small, normal, large, tablets, etc.).
    - Elements stay centered and spaced correctly without needing manual adjustments for different screen sizes.
7. **Set up functionality in MainActivity.java**:
    - In onCreate(), use findViewById() to reference the views (logoImage, titleText, startButton).
    - Set a click listener on the button to change the TextView text to "Button Clicked!" when the button is pressed.
8. **Run and test**:
    - Test on different emulators and physical devices.
    - Confirm that the layout adjusts nicely on various screen sizes.

Code:Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
```

```
xmlns:android="http://schemas.android.com/apk/res/android"
```

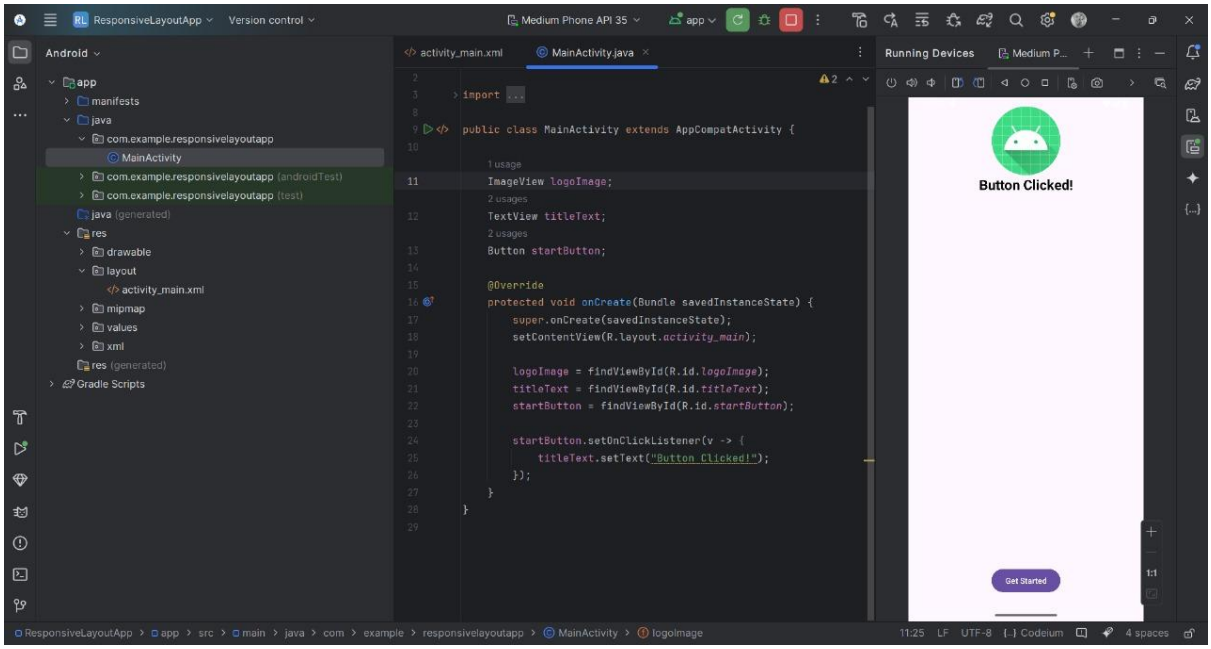
```
xmlns:app="http://schemas.android.
com/apk/res-auto"
  android:id="@+id/constraintLayout"

android:layout_width="match_parent
"

android:layout_height="match_parent
"
  android:padding="16dp">

  <ImageView
    android:id="@+id/logoImage"
    android:layout_width="120dp"
    android:layout_height="120dp"

android:src="@mipmap/ic_launcher"

app:layout_constraintTop_toTopOf="p
arent"

app:layout_constraintBottom_toTopO
f="@id/titleText"

app:layout_constraintStart_toStartOf
="parent"

app:layout_constraintEnd_toEndOf="
parent"

app:layout_constraintVertical_bias="
0.3"/>

  <TextView
    android:id="@+id/titleText"

android:layout_width="wrap_content"

android:layout_height="wrap_content
"
    android:text="Welcome!"
    android:textSize="24sp"
    android:textStyle="bold"
    android:textColor="#000000"

app:layout_constraintTop_toBottomO
f="@id/logoImage"

app:layout_constraintStart_toStartOf
="parent"

app:layout_constraintEnd_toEndOf="
parent"

app:layout_constraintVertical_bias="
0.05"/>

  <Button
    android:id="@+id/startButton"

android:layout_width="wrap_content"

android:layout_height="wrap_content
"
    android:text="Get Started"

app:layout_constraintBottom_toBotto
mOf="parent"

app:layout_constraintStart_toStartOf
="parent"

app:layout_constraintEnd_toEndOf="
parent"

android:layout_marginBottom="32dp"
/>
</androidx.constraintlayout.widget.C
onstraintLayout>
```

Responsivelayoutapp.java

```
package
com.example.responsivelayoutapp;
...

public class MainActivity extends
AppCompatActivity {

  ImageView logoImage;
```

```
    TextView titleText;
    Button startButton;

    @Override
    protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_mai
n);

        logoImage =
findViewById(R.id.logoImage);
        titleText =
findViewById(R.id.titleText);
        startButton =
findViewById(R.id.startButton);

        startButton.setOnClickListener(v -
> {
            titleText.setText("Button
Clicked!");
        });
    }
}
```

Output:



Q9: Create an app with an Explicit Intent to navigate from one activity to another and pass a message between them.

STEPS

1. **Create a new Android project** in Android Studio with an Empty Activity.
2. **Set up MainActivity**:
    - Open MainActivity.java (or MainActivity.kt if you're using Kotlin).
    - Define **UI components** (EditText and Button) to allow the user to input a message and send it.
    - Use findViewById() to reference the **EditText** (for message input) and the **Button** (for triggering the intent).
    - Set a **click listener** on the button:
        - Get the message from the **EditText** using getText().toString().

- Create an **Intent** to start the SecondActivity using new Intent(MainActivity.this, SecondActivity.class).
- Use **putExtra()** to attach the message to the Intent (putExtra("EXTRA_MESSAGE", message)).
- Start the SecondActivity using startActivity(intent).

3. **Set up SecondActivity**:
   - Open SecondActivity.java (or SecondActivity.kt if you're using Kotlin).
   - Define a **TextView** (textViewReceivedMessage) to display the message received from MainActivity.
   - Use getIntent().getStringExtra("EXTRA_MESSAGE") to retrieve the message from the Intent.
   - Set the retrieved message in the **TextView** using textViewReceivedMessage.setText("Message: " + message).
4. **Define the layout for MainActivity**:
   - Create an **EditText** where users can type a message.
   - Create a **Button** to send the message to SecondActivity.
   - Layout file (activity_main.xml): Arrange both views vertically with some padding.
5. **Define the layout for SecondActivity**:
   - Create a **TextView** to display the received message.
   - Layout file (activity_second.xml): Ensure it displays the message properly in a readable font size.
6. **Test the app**:
   - When the app starts, enter a message in MainActivity, press the "Send Message" button, and it will navigate to SecondActivity.
   - In SecondActivity, the message should be displayed in the TextView.

Code: Mainactivity.java

```
package com.example.q9explicit;
...
public class MainActivity extends
AppCompatActivity {

   EditText editTextMessage;
   Button buttonSend;

   @Override
   protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_mai
n);

     editTextMessage =
findViewById(R.id.editTextMessage);
     buttonSend =
findViewById(R.id.buttonSend);


buttonSend.setOnClickListener(new
View.OnClickListener() {
        @Override
        public void onClick(View view) {
          String message =
editTextMessage.getText().toString();
          Intent intent = new
```

```
Intent(MainActivity.this,
SecondActivity.class);

intent.putExtra("EXTRA_MESSAGE",
message);
            startActivity(intent);
        }
    });
  }
}
```

Secondactivity.java

```
package com.example.q9explicit;
...
public class SecondActivity extends
AppCompatActivity {

  TextView
textViewReceivedMessage;

  @Override
  protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_sec
ond);

    textViewReceivedMessage =
findViewById(R.id.textViewReceivedM
essage);

    String message =
getIntent().getStringExtra("EXTRA_ME
SSAGE");

textViewReceivedMessage.setText("M
essage: " + message);
  }
}
```

Activity_main.xml

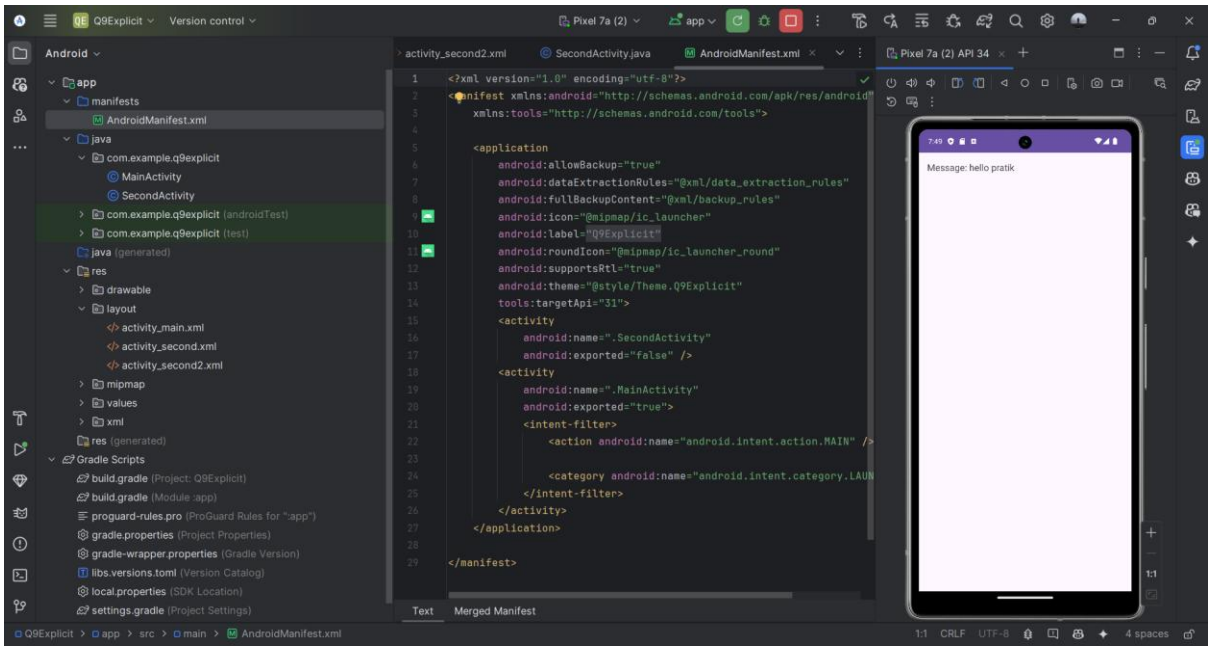
```
<?xml version="1.0" encoding="utf-
8"?>
<LinearLayout

xmlns:android="http://schemas.andr
oid.com/apk/res/android"

android:layout_width="match_parent
"

android:layout_height="match_parent
"
  android:orientation="vertical"
  android:padding="16dp">

  <EditText

android:id="@+id/editTextMessage"
android:layout_width="match_parent
"

android:layout_height="wrap_content
"
    android:hint="Enter a message" />

  <Button
    android:id="@+id/buttonSend"

android:layout_width="wrap_content"

android:layout_height="wrap_content
"
    android:text="Send Message" />
</LinearLayout>
```

Activity_second.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout_width="match_parent"


android:layout_height="match_parent"
   android:orientation="vertical"
   android:padding="16dp">
```

```
   <TextView

android:id="@+id/textViewReceivedMessage"

android:layout_width="wrap_content"

android:layout_height="wrap_content"
      android:text="Received message will appear here"
      android:textSize="18sp" />
</LinearLayout>
```

Output:



Q10: Implement an Implicit Intent to open a web page using the default browser.

Steps to implement an Implicit Intent to open a web page:

1. **Create a New Android Project**:
   - Start a new Android project in **Android Studio** with an **Empty Activity** template.
2. **Update MainActivity.java**:
   - Open the MainActivity.java (or MainActivity.kt if you're using Kotlin).

- Define a **Button** to allow the user to trigger the action of opening a web page.
- Set a **click listener** on the button:
  - Create an **Intent** with the action Intent.ACTION_VIEW to indicate that the app wants to view something (in this case, a web page).
  - Use setData(Uri.parse("https://www.google.com")) to specify the URL you want to open.
  - Call startActivity(intent) to open the URL in the default browser.

3. **Define the Layout in activity_main.xml**:
   - Use a **RelativeLayout** to place a **Button** at the center of the screen.
   - Set the **button's ID** as openWebButton so you can reference it in the activity.
   - The **button text** should say "Open Web Page."
4. **Ensure Permission (if necessary)**:
   - For this specific example (opening a URL), **no permissions** are required.
   - However, for more complex intents (e.g., opening a file), you may need permissions in AndroidManifest.xml.
5. **Testing the Intent**:
   - When you run the app, pressing the button will open **Google's homepage** in the default browser (or any other browser the user has set as the default).
   - The app uses **Implicit Intent** because it doesn't directly specify which activity (browser) should handle the request—it's up to Android to find a suitable app (browser) to handle the ACTION_VIEW intent.
6. **Update AndroidManifest.xml**:
   - You don't need to add any specific configuration in the manifest for implicit intents like this, as Android will automatically resolve the intent to the appropriate app (browser in this case).

Code:Mainactivity

```
package com.example.q10implicit;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import
androidx.appcompat.app.AppCompa
tActivity;

public class MainActivity extends
AppCompatActivity {
```

```
   @Override
   protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_mai
n);

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

```
openWebButton.setOnClickListener(
new View.OnClickListener() {
        @Override
        public void onClick(View v) {
           // URL to open
           String url =
"https://www.google.com";
           // Create the intent
           Intent intent = new
```

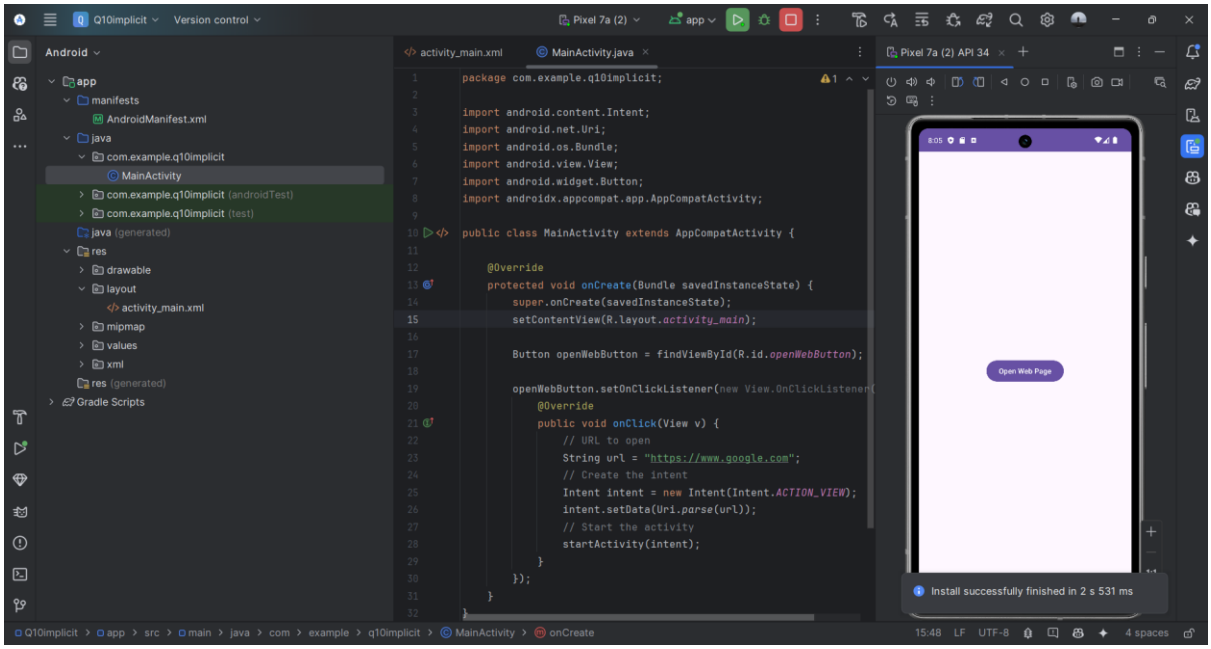
```
Intent(Intent.ACTION_VIEW);
           intent.setData(Uri.parse(url));
           // Start the activity
           startActivity(intent);
        }
     });
  }
}
```

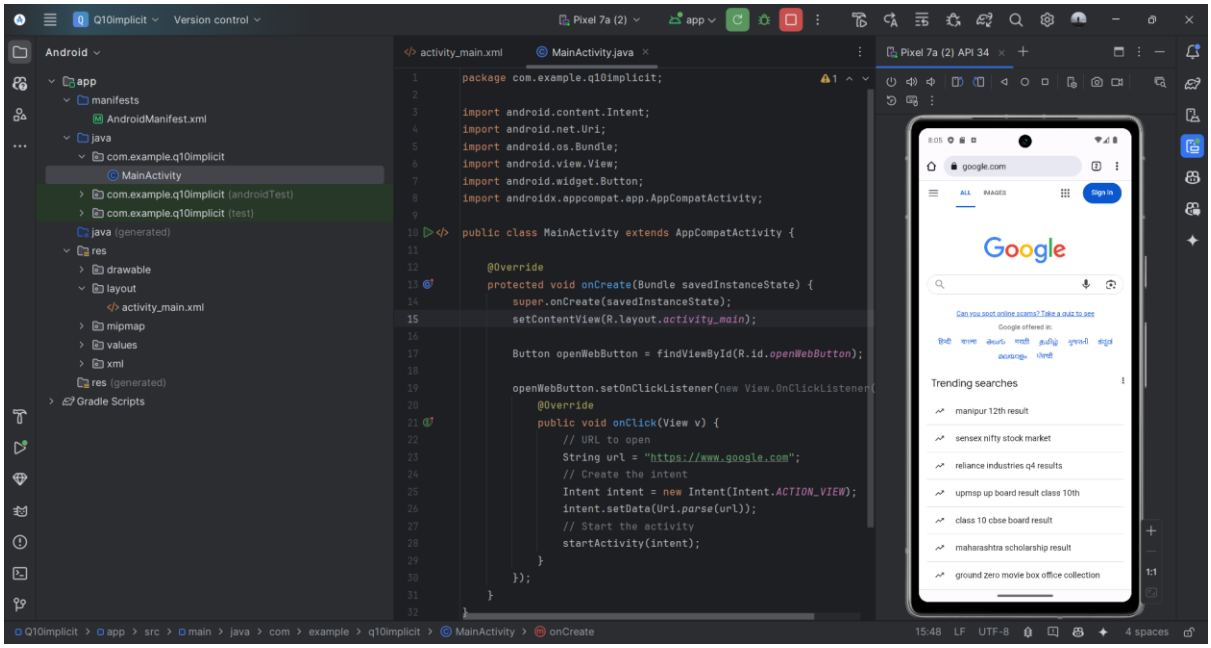
Activity_main.xml

```
<?xml version="1.0" encoding="utf-
8"?>
<RelativeLayout
xmlns:android="http://schemas.andr
oid.com/apk/res/android"

xmlns:tools="http://schemas.android
.com/tools"

android:layout_width="match_parent
"

android:layout_height="match_parent
"
   tools:context=".MainActivity">
```

```
   <Button

android:id="@+id/openWebButton"

android:layout_width="wrap_content"

android:layout_height="wrap_content
"
      android:text="Open Web Page"

android:layout_centerInParent="true"
/>
</RelativeLayout>
```

Output:





Q11: Create a RecyclerView to display a list of items (e.g., a to-do list). Use a custom adapter to bind the data.

Steps:

1. **Create Model Class**: Define a ToDoItem class with a task string.
2. **Item Layout**: Design a layout (todo_item.xml) for each to-do item using a CardView and TextView.
3. **Adapter**: Create a custom adapter (ToDoAdapter) to bind the data from a list of ToDoItem objects to the RecyclerView.
4. **Main Activity Layout**: Add a RecyclerView in the activity_main.xml.
5. **Main Activity**: Set up the RecyclerView in MainActivity.java, initialize the data, and attach the adapter.
6. **Run**: The RecyclerView will display the list of to-do items in a vertical list.

Code: activity_main.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"

android:layout_width="match_parent"

android:layout_height="match_parent">
```

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerView"
    android:layout_width="0dp"
    android:layout_height="0dp"

app:layout_constraintTop_toTopOf="parent"
```

```
app:layout_constraintBottom_toBottomOf="parent"

app:layout_constraintStart_toStartOf="parent"

app:layout_constraintEnd_toEndOf="parent"/>

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

ToDoItem.java

```
public class ToDoItem {
  private String task;

  public ToDoItem(String task) {
    this.task = task;
  }
```

```
  public String getTask() {
    return task;
  }
}
```

todo_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"

android:layout_width="match_parent"

android:layout_height="wrap_content"
  android:layout_margin="8dp"
  android:elevation="4dp">
```

```
  <TextView
    android:id="@+id/taskText"

android:layout_width="match_parent"

android:layout_height="wrap_content"
    android:padding="16dp"
    android:textSize="18sp" />
</androidx.cardview.widget.CardView>
```

ToDoAdapter.java

```
...
public class ToDoAdapter extends RecyclerView.Adapter<ToDoAdapter.ToDoViewHolder> {

  private List<ToDoItem> toDoList;
```

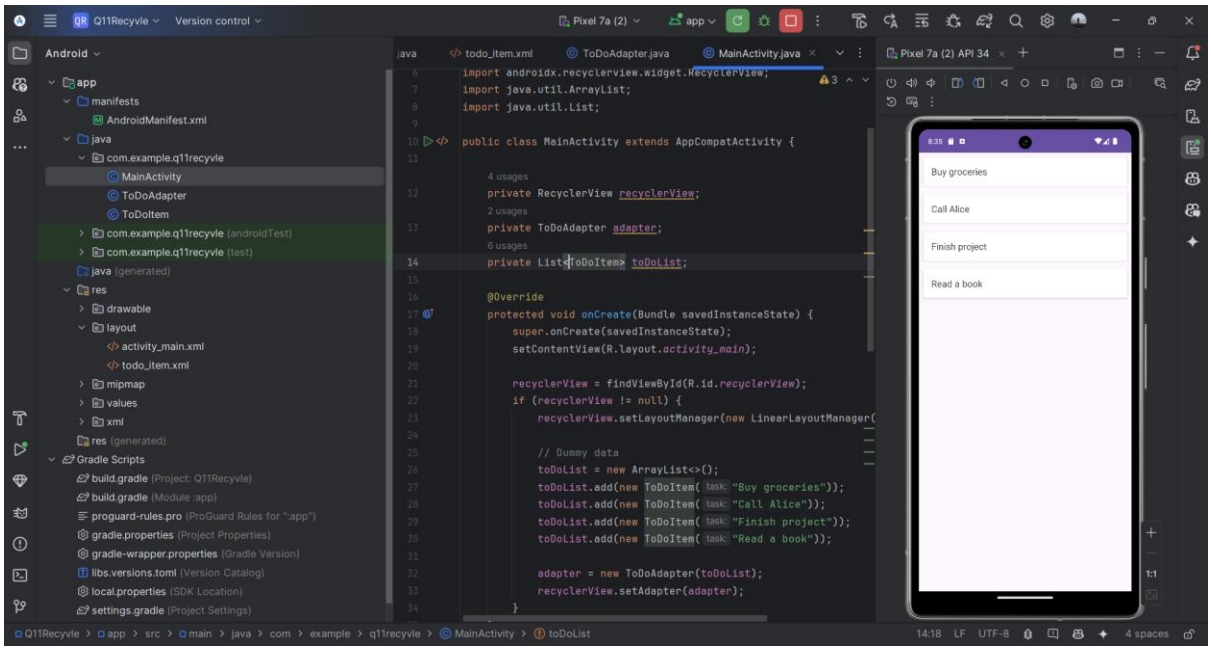
```
  public ToDoAdapter(List<ToDoItem> toDoList) {
    this.toDoList = toDoList;
  }

  @NonNull
  @Override
```

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

    @Override
    public void
onBindViewHolder(@NonNull
ToDoViewHolder holder, int position) {
        ToDoItem item =
toDoList.get(position);

holder.taskText.setText(item.getTask())
;
    }

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

    public static class ToDoViewHolder
extends RecyclerView.ViewHolder {
        TextView taskText;

        public
ToDoViewHolder(@NonNull View
itemView) {
            super(itemView);
            taskText =
itemView.findViewById(R.id.taskText);
        }
    }
}
```

MainActivity.java

```
...
public class MainActivity extends
AppCompatActivity {

    private RecyclerView recyclerView;
    private ToDoAdapter adapter;
    private List<ToDoItem> toDoList;

    @Override
    protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_mai
n);

        recyclerView =
findViewById(R.id.recyclerView);

recyclerView.setLayoutManager(new
LinearLayoutManager(this));

        // Dummy data
        toDoList = new ArrayList<>();
        toDoList.add(new ToDoItem("Buy
groceries"));
        toDoList.add(new ToDoItem("Call
Alice"));
        toDoList.add(new
ToDoItem("Finish project"));
        toDoList.add(new
ToDoItem("Read a book"));

        adapter = new
ToDoAdapter(toDoList);
        recyclerView.setAdapter(adapter);
    }
}
```

Output:



Q12: Implement a Spinner with an ArrayAdapter to display a list of countries.

Steps:

1. **Create Spinner in XML**: Add a Spinner element to your activity_main.xml.
2. **Define Data Source**: Create an array of countries in MainActivity.java.
3. **Create Adapter**: Use ArrayAdapter to bind the array data to the spinner.
4. **Set Drop-Down Layout**: Specify how the dropdown list should appear.
5. **Attach Adapter to Spinner**: Bind the ArrayAdapter to the spinner.
6. **Set Item Selected Listener**: Handle user selection and show a Toast message when an item is selected.

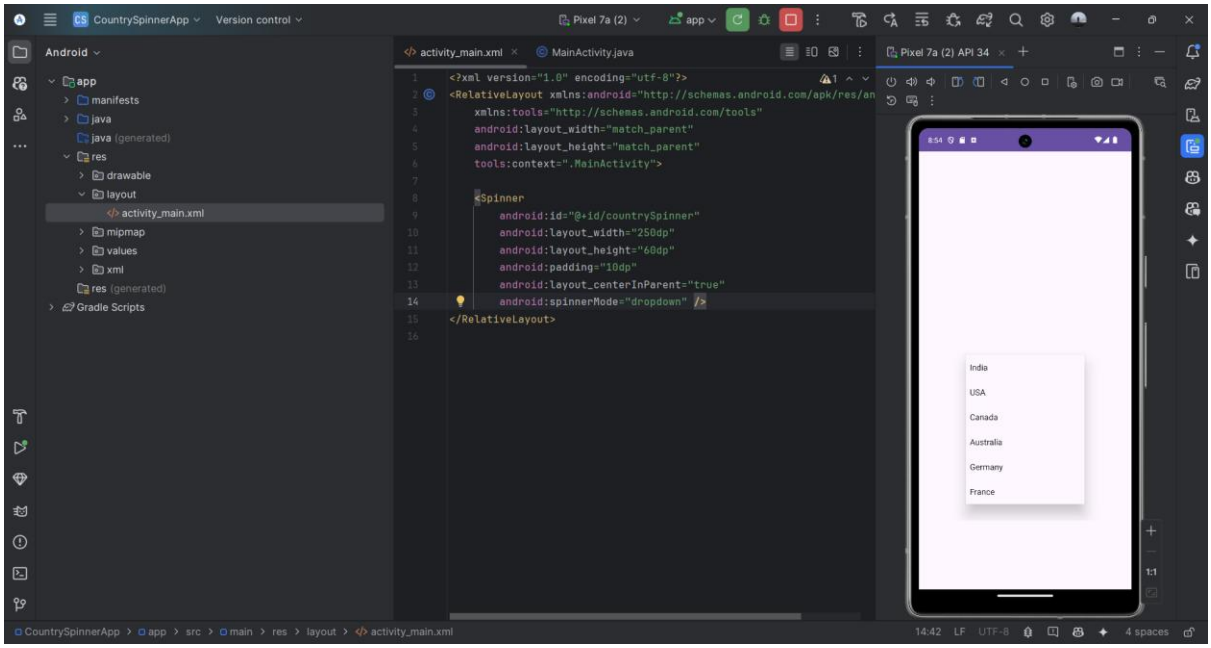
Code:activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout_width="match_parent"

android:layout_height="match_parent"
  tools:context=".MainActivity">

  <Spinner

android:id="@+id/countrySpinner"
    android:layout_width="250dp"
    android:layout_height="60dp"
    android:padding="10dp"

android:layout_centerInParent="true"

android:spinnerMode="dropdown" />
</RelativeLayout>
```

MainActivity.java

```
package
com.example.countryspinnerapp;
...
public class MainActivity extends
AppCompatActivity {

   Spinner countrySpinner;
   String[] countries = {"India", "USA",
"Canada", "Australia", "Germany",
"France"};

   @Override
   protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_mai
n);

      countrySpinner =
findViewById(R.id.countrySpinner);

      // Step 1: Create ArrayAdapter
      ArrayAdapter<String> adapter =
new ArrayAdapter<>(this,

android.R.layout.simple_spinner_ite
m, countries);

      // Step 2: Set drop-down layout
style

adapter.setDropDownViewResource(
android.R.layout.simple_spinner_dro
pdown_item);

      // Step 3: Attach adapter to
spinner

countrySpinner.setAdapter(adapter);

      // Step 4: Set listener

countrySpinner.setOnItemSelectedLi
stener(new
AdapterView.OnItemSelectedListener
() {
         @Override
         public void
onItemSelected(AdapterView<?>
parent, View view, int position, long id)
{
            String selectedCountry =
countries[position];

Toast.makeText(MainActivity.this,
"Selected: " + selectedCountry,
Toast.LENGTH_SHORT).show();
         }

         @Override
         public void
onNothingSelected(AdapterView<?>
parent) {
            // Optional: handle no
selection
         }
      });
   }
}
```

Output:



Q13: Create an AlertDialog with "Yes" and "No" buttons. Display a Toast message based on the user's choice.

Steps:

1. **Create a Button in XML**: Add a button (showDialogBtn) to your layout that will trigger the dialog.
2. **Set OnClickListener for Button**: In MainActivity.java, set an OnClickListener for the button to show the dialog.
3. **Build the AlertDialog**: Use AlertDialog.Builder to create an AlertDialog.
4. Set the title and message for the dialog using setTitle() and setMessage().
5. **Add "Yes" and "No" buttons**:
6. Set the "Yes" button using setPositiveButton() and display a Toast when clicked.
7. Set the "No" button using setNegativeButton() and display a Toast when clicked.
8. **Show the Dialog**: Call create() and show() to display the dialog.
9. **Handle Button Clicks**: In the button's click listener, display appropriate Toast messages for each button choice.

Code:MainActivity.java

```
// MainActivity.java
package
com.example.alertdialogexample;
...
public class MainActivity extends
AppCompatActivity {

   Button showDialogBtn;
```

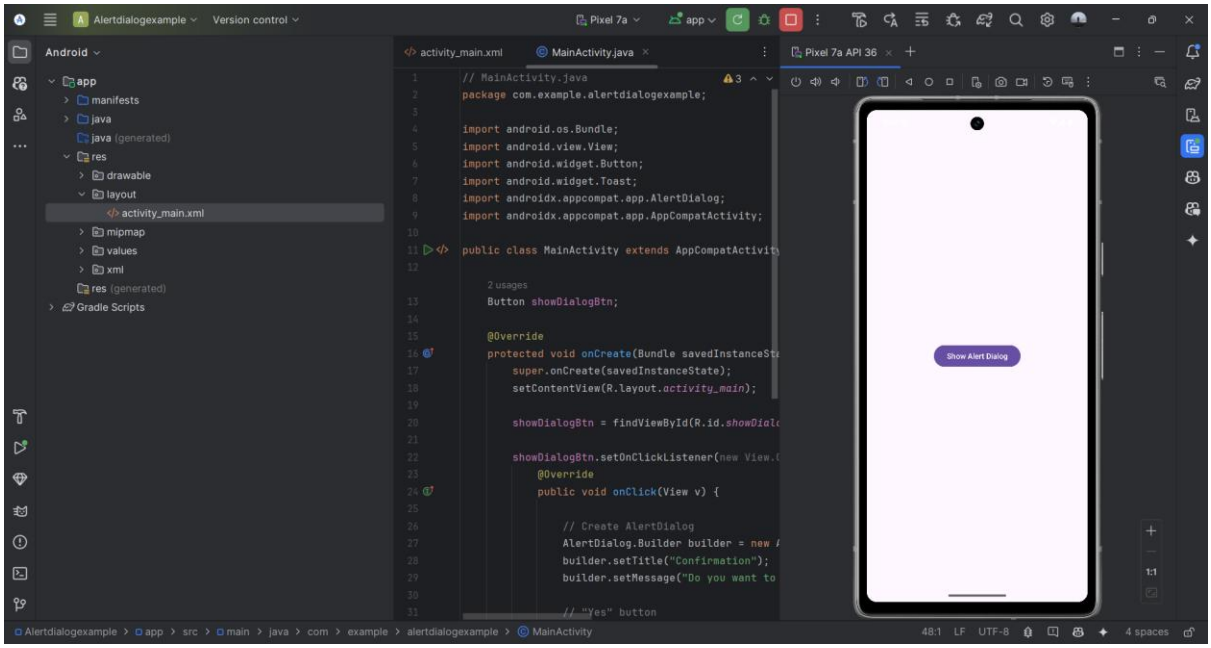
```
   @Override
   protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_mai
n);
```

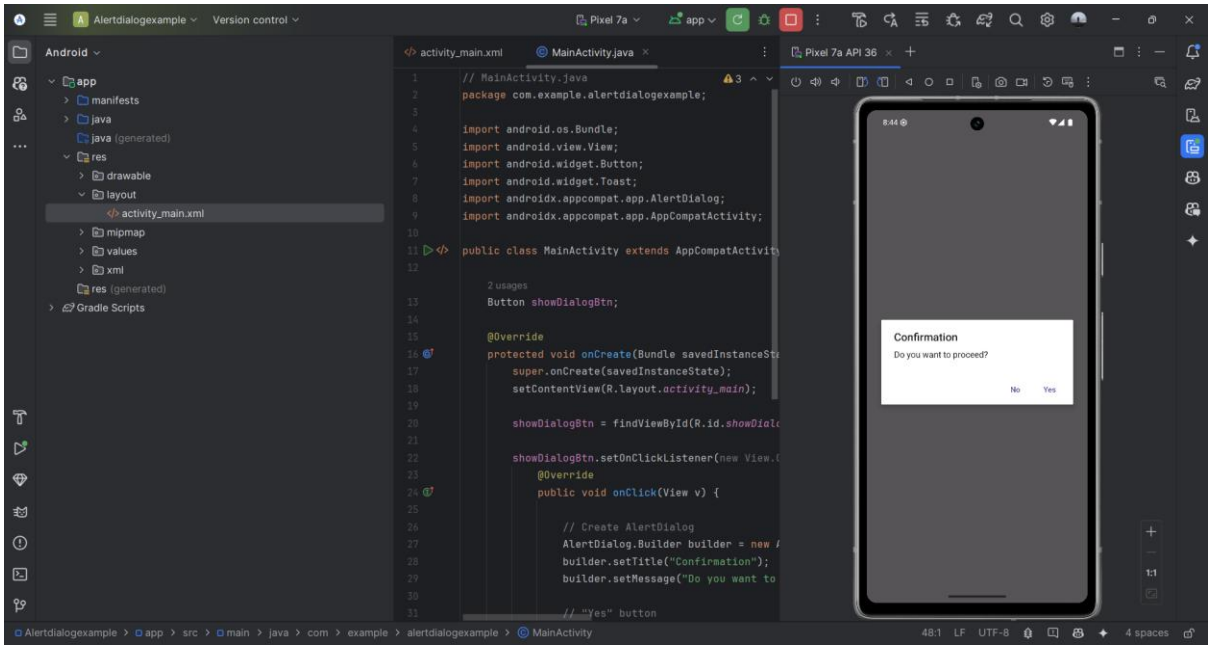
```
    showDialogBtn =
findViewById(R.id.showDialogBtn);


showDialogBtn.setOnClickListener(n
ew View.OnClickListener() {
        @Override
        public void onClick(View v) {

          // Create AlertDialog
          AlertDialog.Builder builder =
new
AlertDialog.Builder(MainActivity.this);

builder.setTitle("Confirmation");
          builder.setMessage("Do you
want to proceed?");

          // "Yes" button

builder.setPositiveButton("Yes",
(dialog, which) -> {

Toast.makeText(MainActivity.this, "You
clicked Yes",
Toast.LENGTH_SHORT).show();
          });

          // "No" button

builder.setNegativeButton("No",
(dialog, which) -> {

Toast.makeText(MainActivity.this, "You
clicked No",
Toast.LENGTH_SHORT).show();
          });

          // Show dialog
          AlertDialog dialog =
builder.create();
          dialog.show();
        }
    });
  }
}
```

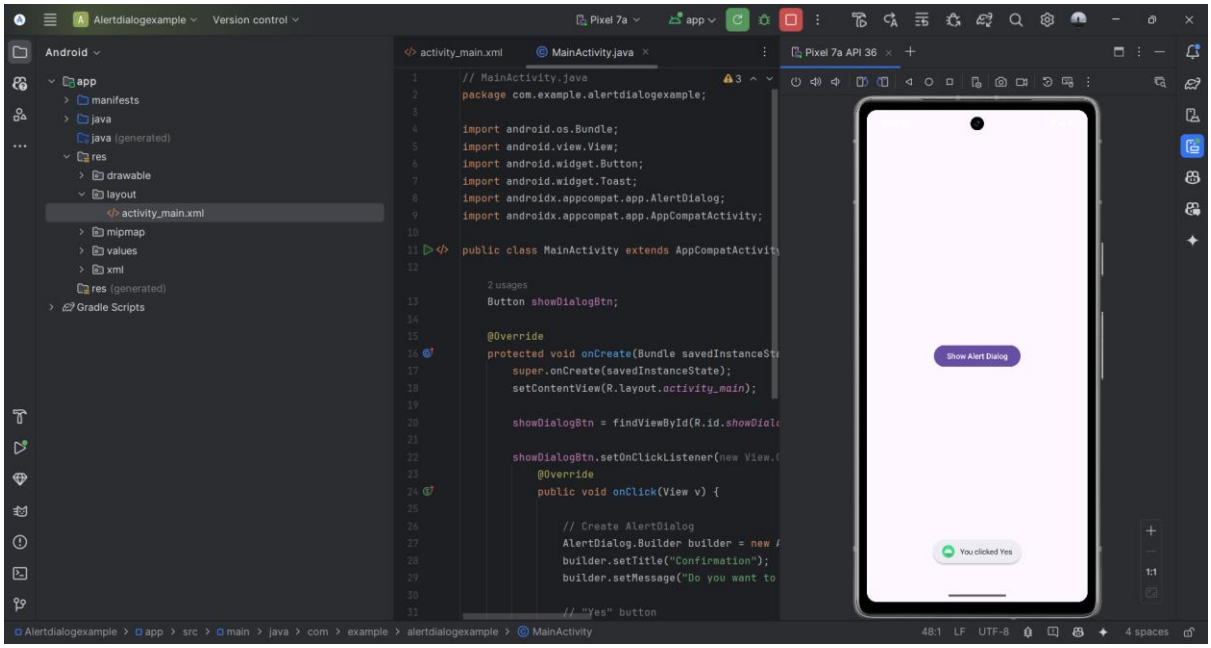
activity_main.xml

```
<!-- res/layout/activity_main.xml -->
<LinearLayout
xmlns:android="http://schemas.andr
oid.com/apk/res/android"

android:layout_width="match_parent
"

android:layout_height="match_parent
"
  android:gravity="center"
  android:orientation="vertical">


  <Button
    android:id="@+id/showDialogBtn"

android:layout_width="wrap_content"

android:layout_height="wrap_content
"
    android:text="Show Alert Dialog"
/>
</LinearLayout>
```

Output:


Alertdialogexample
Version control
Pixel 7a
app
Android
activity_main.xml
MainActivity.java
Pixel 7a API 36
app
manifests
java
java (generated)
res
drawable
layout
activity_main.xml
mipmap
values
xml
res (generated)
Gradle Scripts
// MainActivity.java
package com.example.alertdialogexample;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity
2 usages
Button showDialogBtn;
@Override
protected void onCreate(Bundle savedInstanceSta
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
showDialogBtn = findViewById(R.id.showDialo
showDialogBtn.setOnClickListener(new View.
@Override
public void onClick(View v) {
// Create AlertDialog
AlertDialog.Builder builder = new A
builder.setTitle("Confirmation");
builder.setMessage("Do you want to
// "Yes" button
Show Alert Dialog
Alertdialogexample
app
src
main
java
com
example
alertdialogexample
MainActivity
48:1
LF
UTF-8
4 spaces


Alertdialogexample
Version control
Pixel 7a
app
Android
activity_main.xml
MainActivity.java
Pixel 7a API 36
app
manifests
java
java (generated)
res
drawable
layout
activity_main.xml
mipmap
values
xml
res (generated)
Gradle Scripts
// MainActivity.java
package com.example.alertdialogexample;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity
2 usages
Button showDialogBtn;
@Override
protected void onCreate(Bundle savedInstanceSta
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
showDialogBtn = findViewById(R.id.showDialo
showDialogBtn.setOnClickListener(new View.
@Override
public void onClick(View v) {
// Create AlertDialog
AlertDialog.Builder builder = new A
builder.setTitle("Confirmation");
builder.setMessage("Do you want to
// "Yes" button
8:44
Confirmation
Do you want to proceed?
No
Yes
Alertdialogexample
app
src
main
java
com
example
alertdialogexample
MainActivity
48:1
LF
UTF-8
4 spaces



Q14: Implement a DatePickerDialog and display the selected date in a TextView.

Steps

1. **Create a TextView and Button**: Add a TextView to show the selected date.
2. Add a Button that will trigger the DatePickerDialog.
3. **Set OnClickListener for the Button**:
4. In MainActivity.java, set an OnClickListener for the button to open the DatePickerDialog when clicked.
5. **Initialize DatePickerDialog**: Get the current date (year, month, and day) using Calendar.getInstance().
6. Create a DatePickerDialog and set a listener to handle the selected date.
7. **Format the Date**: Inside the listener, format the selected date as a string (day/month/year).
8. **Display the Date**: Set the formatted date as the text of the TextView.

Code:MainActivity.java

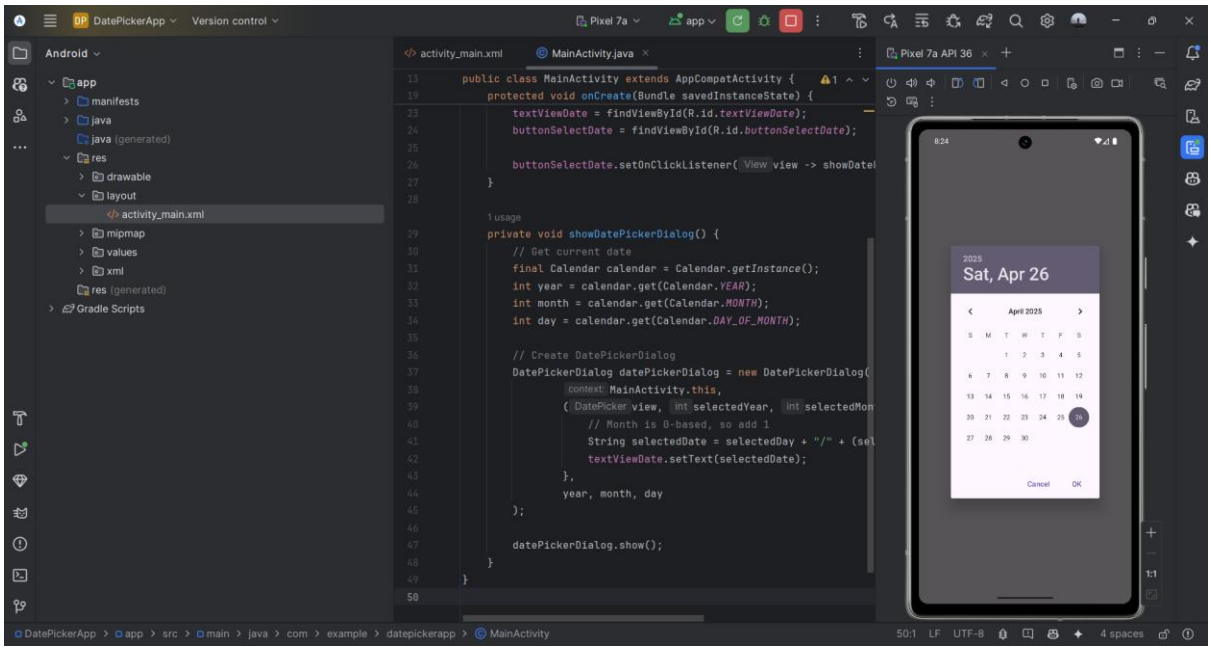
```
package
com.example.datepickerapp;
....
public class MainActivity extends
AppCompatActivity {

   TextView textViewDate;
   Button buttonSelectDate;

   @Override
   protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.activity_main);

    textViewDate = findViewById(R.id.textViewDate);
    buttonSelectDate = findViewById(R.id.buttonSelectDate);


buttonSelectDate.setOnClickListener(view -> showDatePickerDialog());
  }

  private void showDatePickerDialog() {
    // Get current date
    final Calendar calendar = Calendar.getInstance();
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);

    // Create DatePickerDialog
    DatePickerDialog datePickerDialog = new DatePickerDialog(
        MainActivity.this,
        (view, selectedYear, selectedMonth, selectedDay) -> {
          // Month is 0-based, so add 1
          String selectedDate = selectedDay + "/" + (selectedMonth + 1) + "/" + selectedYear;

textViewDate.setText(selectedDate);
        },
        year, month, day
    );

    datePickerDialog.show();
  }
}
```

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout_width="match_parent"

android:layout_height="match_parent"
  android:orientation="vertical"
  android:padding="24dp"
  android:gravity="center">

  <TextView
    android:id="@+id/textViewDate"

android:layout_width="wrap_content"

android:layout_height="wrap_content"
    android:text="Select a date"
    android:textSize="20sp"
    android:padding="16dp" />

  <Button

android:id="@+id/buttonSelectDate"

android:layout_width="wrap_content"

android:layout_height="wrap_content"
    android:text="Pick Date" />
</LinearLayout>
```

Output:



Q15: Add an Options Menu to an activity with actions like "Settings" and "Logout." Handle click events for these options.

Steps:

1. **Add a Toolbar**:
2. Place a Toolbar in activity_main.xml and set it as the app's ActionBar in MainActivity.java.
3. **Create Menu Resource**:
4. Create main_menu.xml inside the res/menu folder.
5. Add Settings and Logout items in the menu file.
6. **Inflate Menu**:
7. Override onCreateOptionsMenu() in MainActivity to inflate the menu.
8. **Handle Menu Clicks**:
9. Override onOptionsItemSelected() to detect clicks on "Settings" or "Logout" and show a Toast message accordingly.

Code:MainActivity.java

```
package com.example.yourapp15; // Change this to your actual package name
...
public class MainActivity extends AppCompatActivity {

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_main);
```

```
    // Set up the custom toolbar as the ActionBar
    Toolbar toolbar = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    // Force overflow menu to show (even on devices with hardware menu button)
    try {
      ViewConfiguration config = ViewConfiguration.get(this);
      Field menuKeyField = ViewConfiguration.class.getDeclaredField("sHasPermanentMenuKey");
      if (menuKeyField != null) {
        menuKeyField.setAccessible(true);
        menuKeyField.setBoolean(config, false);
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {

    getMenuInflater().inflate(R.menu.main_menu, menu);
    return true;
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();

    if (id == R.id.action_settings) {
      Toast.makeText(this, "Settings clicked", Toast.LENGTH_SHORT).show();
      return true;
    } else if (id == R.id.action_logout) {
      Toast.makeText(this, "Logout clicked", Toast.LENGTH_SHORT).show();
      return true;
    }

    return super.onOptionsItemSelected(item);
  }
}
```

main_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item
    android:id="@+id/action_settings"
    android:title="Settings"
    android:orderInCategory="100"
    android:showAsAction="never" />

  <item
    android:id="@+id/action_logout"
    android:title="Logout"
    android:orderInCategory="101"
    android:showAsAction="never" />
</menu>
```

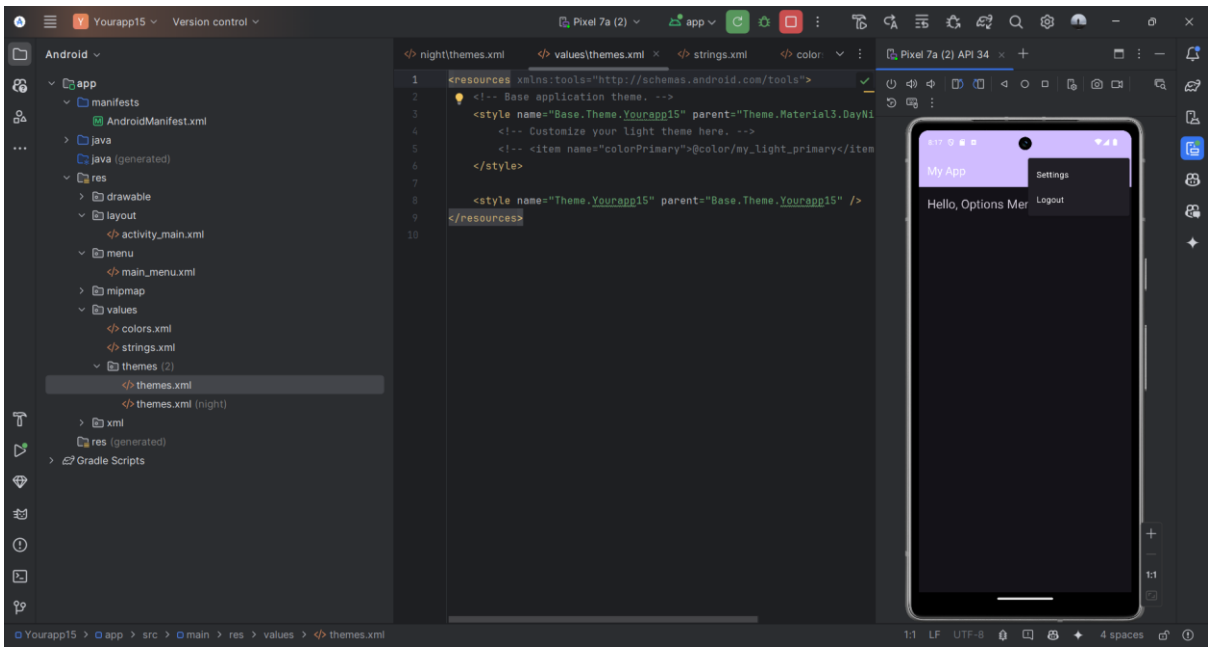
Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
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:orientation="vertical"

android:layout_width="match_parent"

android:layout_height="match_parent"
   tools:context=".MainActivity">

   <!-- Toolbar as ActionBar -->

<androidx.appcompat.widget.Toolbar
      android:id="@+id/toolbar"

android:layout_width="match_parent"

android:layout_height="?attr/actionBarSize"
```

```
android:background="?attr/colorPrimary"

android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
      app:title="My App"

app:titleTextColor="@android:color/white" />

   <!-- Main content here -->
   <TextView
      android:id="@+id/textView"
      android:text="Hello, Options Menu!"
      android:layout_gravity="center"

android:layout_width="match_parent"

android:layout_height="wrap_content"
      android:textSize="24sp"
      android:padding="16dp" />
</LinearLayout>
```

Output:



Q16: Implement a Popup Menu that appears when a button is clicked.

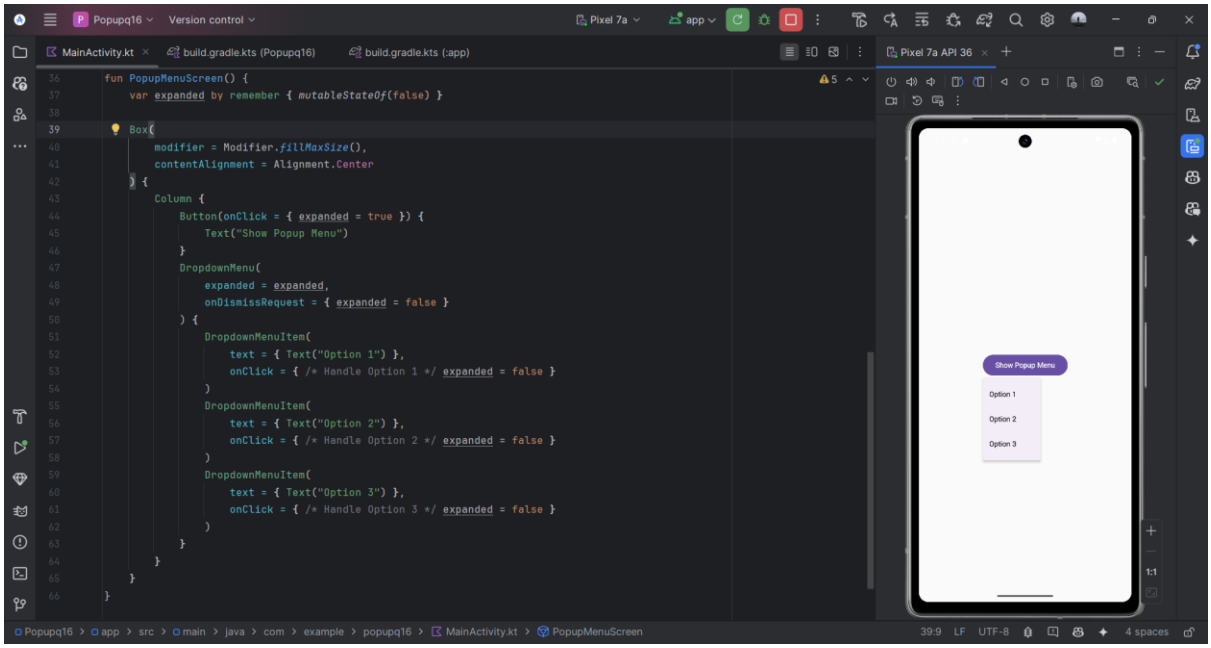
Steps:

1. **Create a Boolean state** (expanded) to control menu visibility.
2. **Add a Button** that, when clicked, sets expanded = true to show the menu.
3. **Use DropdownMenu**:
4. Bind it to expanded.
5. Provide onDismissRequest to hide the menu.
6. **Add DropdownMenuItems** for different options and handle their click events (and close menu).

Code : MainActivity

```
package com.example.popupq16
...
class MainActivity :
ComponentActivity() {
  override fun
onCreate(savedInstanceState:
Bundle?) {
super.onCreate(savedInstanceSta
te)
    setContent {
      PopupMenuScreen()   } }}
@Composable
fun PopupMenuScreen() {
  var expanded by remember {
mutableStateOf(false) }
  Box(
    modifier =
Modifier.fillMaxSize(),
    contentAlignment =
Alignment.Center
  ) {
    Column {
      Button(onClick = { expanded
= true }) {
        Text("Show Popup Menu")
      }
      DropdownMenu(
        expanded = expanded,
        onDismissRequest = {
expanded = false }
      ) {
        DropdownMenuItem(
          text = { Text("Option 1") },
          onClick = { /* Handle
Option 1 */ expanded = false }
        )
        DropdownMenuItem(
          text = { Text("Option 2") },
          onClick = { /* Handle
Option 2 */ expanded = false }
        )
        DropdownMenuItem(
          text = { Text("Option 3") },
          onClick = { /* Handle
Option 3 */ expanded = false }
        )
      }
    }
  }
}
```

Output:



Q17: Create a simple notification that displays when a button is clicked.

1. **Create a Notification Channel** (needed for Android 8+).
2. **Check and Request Notification Permission** (for Android 13+).
3. **Build UI** with a **Button** and a **Text** showing the count.
4. **Handle Button Click**:
    a. If permission is granted, call showNotification().
    b. Otherwise, request permission.
5. **Create and Show Notification** using NotificationCompat.Builder.

Code: MainActivity.kt

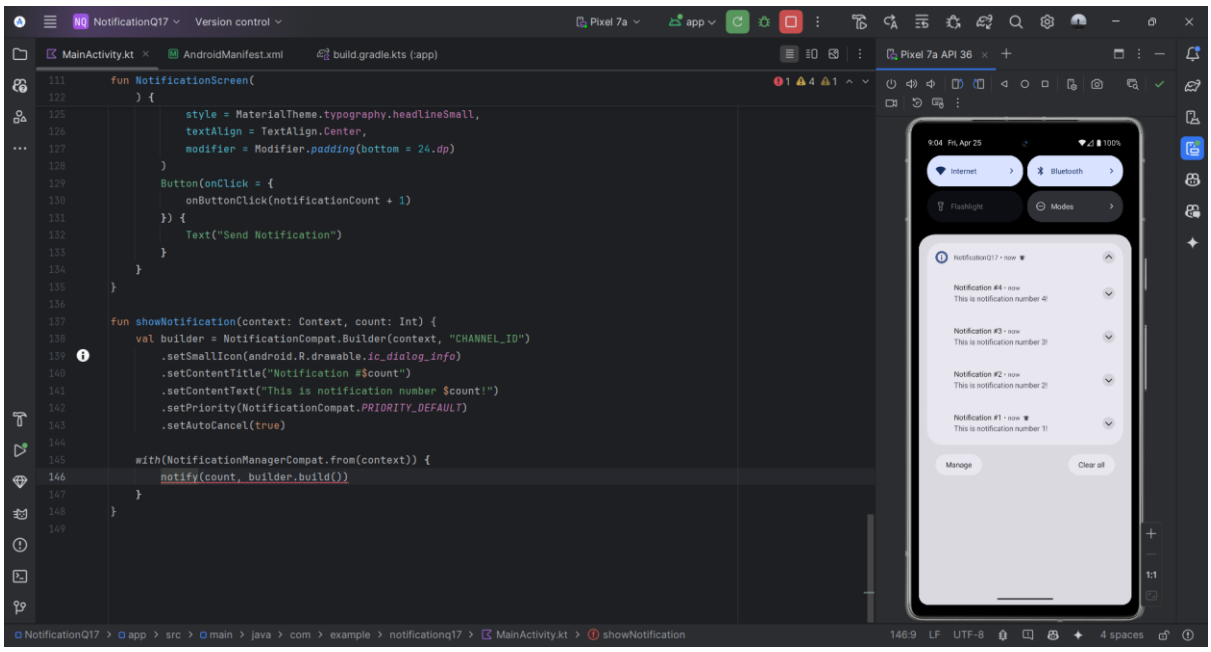
```
package
com.example.notificationq17
...

class MainActivity :
ComponentActivity() {
  private var
pendingNotificationCount: Int? = null
  private var notificationCount by
mutableStateOf(0)

  private val
requestPermissionLauncher =
registerForActivityResult(

ActivityResultContracts.RequestPerm
ission()
  ) { isGranted: Boolean ->
    if (isGranted) {
      pendingNotificationCount?.let {
count ->
        showNotification(this, count)
        notificationCount = count
        pendingNotificationCount =
null
      }
    } else {
      pendingNotificationCount = null
    }
  }

  override fun
onCreate(savedInstanceState:
Bundle?) {

super.onCreate(savedInstanceState)
```

```
    createNotificationChannel()
    setContent {
      NotificationQ17Theme {
        NotificationScreen(
          notificationCount =
notificationCount,
          onButtonClick = { newCount
->
            if
(hasNotificationPermission()) {
              showNotification(this,
newCount)
              notificationCount =
newCount
            } else {

pendingNotificationCount =
newCount

requestNotificationPermission()
            }
          }
        )
      }
    }
  }

  private fun
createNotificationChannel() {
    if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.O) {
      val name = "General
Notifications"
      val descriptionText = "Channel
for general notifications"
      val importance =
NotificationManager.IMPORTANCE_D
EFAULT
      val channel =
NotificationChannel("CHANNEL_ID",
name, importance).apply {
        description = descriptionText
      }
      val notificationManager:
NotificationManager =

getSystemService(Context.NOTIFICAT
ION_SERVICE) as
NotificationManager

notificationManager.createNotificatio
nChannel(channel)
    }
  }

  private fun
hasNotificationPermission(): Boolean
{
    return if (Build.VERSION.SDK_INT
>= Build.VERSION_CODES.TIRAMISU)
{

ContextCompat.checkSelfPermission
(
        this,

Manifest.permission.POST_NOTIFICA
TIONS
      ) ==
PackageManager.PERMISSION_GRAN
TED
    } else {
      true
    }
  }

  private fun
requestNotificationPermission() {
    if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.TIRAMISU) {

requestPermissionLauncher.launch(
Manifest.permission.POST_NOTIFICA
TIONS)
    }
  }
}

@Composable
fun NotificationScreen(
  notificationCount: Int,
  onButtonClick: (Int) -> Unit,
  modifier: Modifier = Modifier
) {
```

```
    Column(
        modifier = modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement =
Arrangement.Center,
        horizontalAlignment =
Alignment.CenterHorizontally
    ) {
        Text(
            text = "Notifications Sent:
$notificationCount",
            style =
MaterialTheme.typography.headlineS
mall,
            textAlign = TextAlign.Center,
            modifier =
Modifier.padding(bottom = 24.dp)
        )
        Button(onClick = {

onButtonClick(notificationCount + 1)
        }) {
            Text("Send Notification")
        }
    }
```

```
}

fun showNotification(context:
Context, count: Int) {
    val builder =
NotificationCompat.Builder(context,
"CHANNEL_ID")

.setSmallIcon(android.R.drawable.ic_
dialog_info)
        .setContentTitle("Notification
#$count")
        .setContentText("This is
notification number $count!")

.setPriority(NotificationCompat.PRIO
RITY_DEFAULT)
        .setAutoCancel(true)


with(NotificationManagerCompat.fro
m(context)) {
        notify(count, builder.build())
    }
}
```

Output:



Q18: Implement a notification with a pending intent that opens a new activity when clicked.

Steps:

1. **Request notification permission** (for Android 13+).
2. **Create a Notification Channel** (needed for Android 8+).
3. **Create an Intent** to launch the **SecondActivity**.
4. **Wrap the Intent in a PendingIntent** (use FLAG_IMMUTABLE for Android 12+).
5. **Build the Notification** and attach the PendingIntent using setContentIntent().
6. **Show the Notification** with NotificationManagerCompat.notify().

Code: MainActivity.java

```
package
com.example.q18notification;
...
public class MainActivity extends
AppCompatActivity {
   private static final String
CHANNEL_ID = "my_channel_id";
   private static final int
NOTIFICATION_ID = 1;
   private static final int
PERMISSION_REQUEST_CODE = 101;
   @Override
   protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_mai
n);
      // Step 1: Ask for notification
permission (API 33+)
      if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.TIRAMISU) {
         if
(ActivityCompat.checkSelfPermission
(this,
Manifest.permission.POST_NOTIFICA
TIONS) !=
PackageManager.PERMISSION_GRAN
TED) {

ActivityCompat.requestPermissions(t
his, new
String[]{Manifest.permission.POST_N
OTIFICATIONS},
PERMISSION_REQUEST_CODE);
            return;
         }
      }
      createNotificationChannel(); //
Step 2: Create the channel

      // Step 3: Intent for the new
activity
      Intent intent = new Intent(this,
SecondActivity.class);

intent.setFlags(Intent.FLAG_ACTIVITY
_NEW_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TASK);

      // Step 4: PendingIntent (use
FLAG_IMMUTABLE for API 31+)
      PendingIntent pendingIntent =
PendingIntent.getActivity(
            this, 0, intent,
PendingIntent.FLAG_IMMUTABLE);

      // Step 5: Build the notification
      NotificationCompat.Builder
builder = new
NotificationCompat.Builder(this,
CHANNEL_ID)
```

```
.setSmallIcon(android.R.drawable.ic_
dialog_info) // system icon
            .setContentTitle("My
Notification")
            .setContentText("Click to open
SecondActivity")
.setPriority(NotificationCompat.PRIO
RITY_DEFAULT)
.setContentIntent(pendingIntent)
            .setAutoCancel(true);
      // Step 6: Show the notification
      NotificationManagerCompat
notificationManager =
NotificationManagerCompat.from(thi
s);

notificationManager.notify(NOTIFICAT
ION_ID, builder.build());
   }
   // Step 2: Create a notification
channel (required for API 26+)
   private void
createNotificationChannel() {
      if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.O) {
         CharSequence name = "My
Channel";
         String description = "Channel for
activity notifications";
         int importance =
NotificationManager.IMPORTANCE_D
EFAULT;
         NotificationChannel channel =
new
NotificationChannel(CHANNEL_ID,
name, importance);

channel.setDescription(description);

         NotificationManager
notificationManager =
getSystemService(NotificationManag
er.class);
notificationManager.createNotificatio
nChannel(channel);
      }
   }
}
```

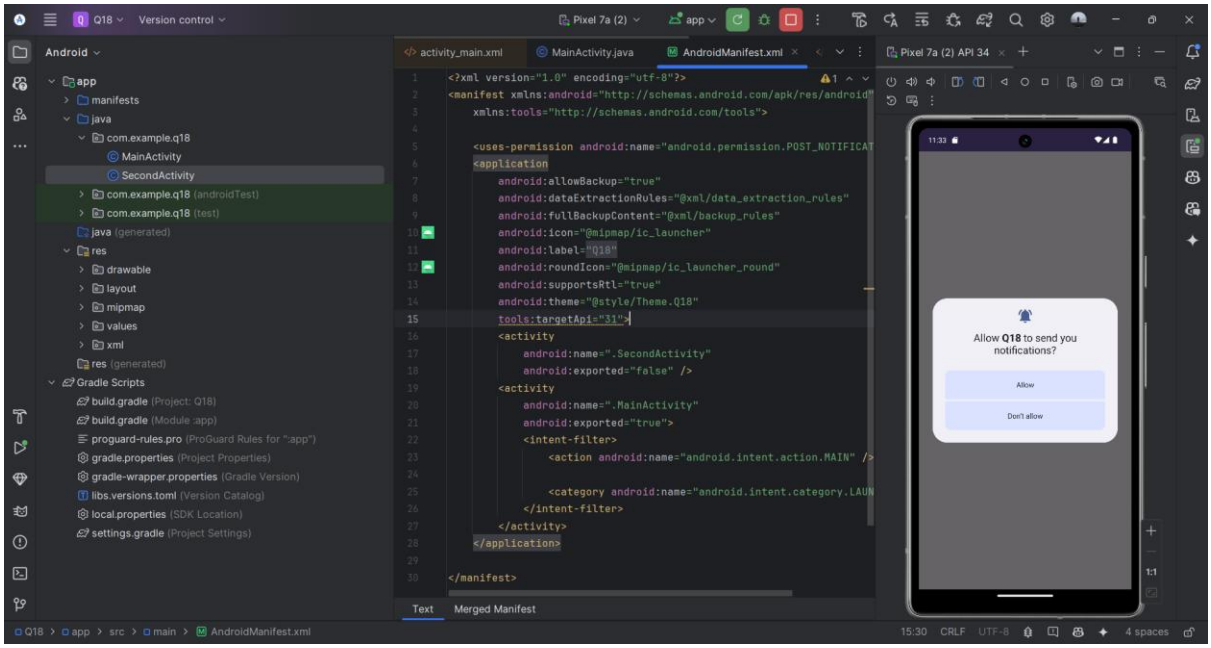
SecondActivity.java

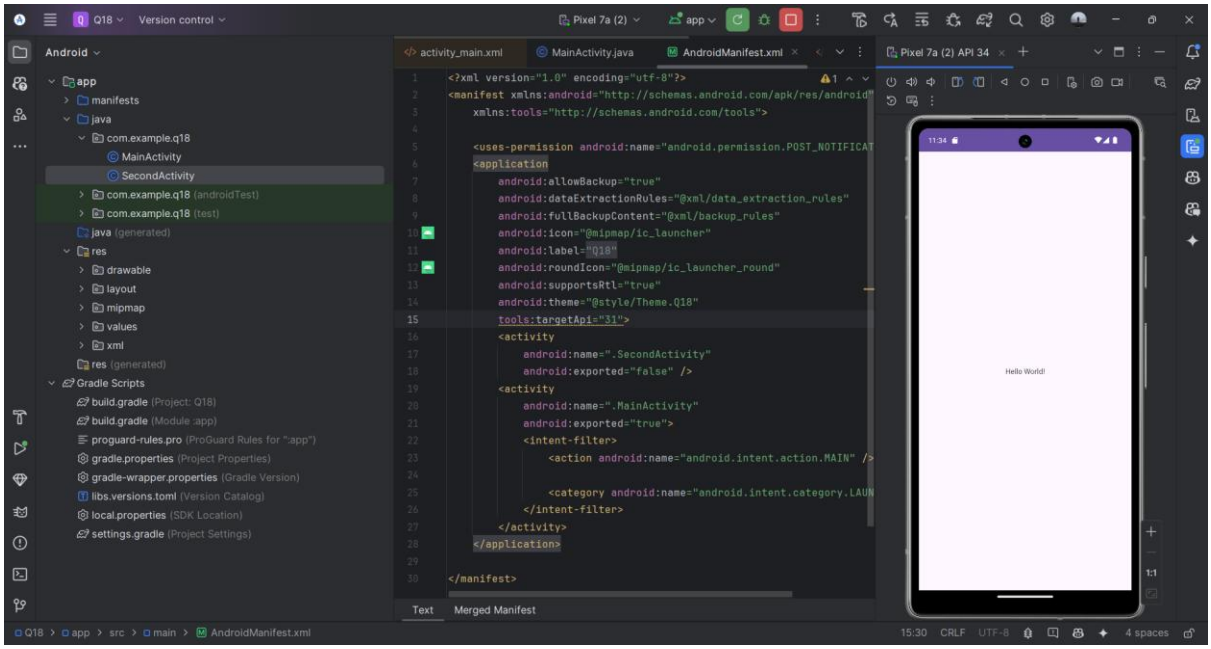
```
package
com.example.q18notification;
...
public class SecondActivity extends
AppCompatActivity {
   @Override
   protected void onCreate(Bundle
savedInstanceState) {
 super.onCreate(savedInstanceState);
setContentView(R.layout.activity_sec
ond);
   }
}
```

AndroidManifest.xml

```
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
```

Output:





Q19: Create an SQLite database to store user information (name, email). Create a form to add data and a ListView to display it.

Steps:

1. **Design the UI** with EditTexts for name and email, a Button to add users, and a ListView to display them.
2. **Create a SQLiteOpenHelper class** to manage the database (create table, add data, fetch all users).
3. **In MainActivity**, initialize views and database helper.
4. **Insert user data** into the database when the Add button is clicked.

5. **Fetch all users** from the database and **display them** in the ListView using an ArrayAdapter.

Code: MainActivity.java

```
package
com.example.userdatabaseapp;
...
public class MainActivity extends
AppCompatActivity {

   EditText editTextName,
editTextEmail;
   Button buttonAdd;
   ListView listViewUsers;
   UserDatabaseHelper dbHelper;
   ArrayAdapter<String> adapter;
   ArrayList<String> userList;

   @Override
   protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_mai
n);

      editTextName =
findViewById(R.id.editTextName);
      editTextEmail =
findViewById(R.id.editTextEmail);
      buttonAdd =
findViewById(R.id.buttonAdd);
      listViewUsers =
findViewById(R.id.listViewUsers);

      dbHelper = new
UserDatabaseHelper(this);
      loadUsers();


buttonAdd.setOnClickListener(new
View.OnClickListener() {
         @Override
         public void onClick(View v) {
            String name =
editTextName.getText().toString().trim(
);
            String email =
editTextEmail.getText().toString().trim(
);

            if (!name.isEmpty() &&
!email.isEmpty()) {
               dbHelper.addUser(name,
email);
               editTextName.setText("");
               editTextEmail.setText("");
               loadUsers();
            } else {

Toast.makeText(MainActivity.this,
"Enter all fields",
Toast.LENGTH_SHORT).show();
            }
         }
      });
   }

   private void loadUsers() {
      userList = dbHelper.getAllUsers();
      adapter = new
ArrayAdapter<>(this,
android.R.layout.simple_list_item_1,
userList);

listViewUsers.setAdapter(adapter);
   }
}
```

UserDatabaseHelper.java

```
package
com.example.userdatabaseapp;
...
public class MainActivity extends
AppCompatActivity {

   EditText editTextName,
editTextEmail;
   Button buttonAdd;
   ListView listViewUsers;
   UserDatabaseHelper dbHelper;
   ArrayAdapter<String> adapter;
   ArrayList<String> userList;

   @Override
   protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_mai
n);

      editTextName =
findViewById(R.id.editTextName);
      editTextEmail =
findViewById(R.id.editTextEmail);
      buttonAdd =
findViewById(R.id.buttonAdd);
      listViewUsers =
findViewById(R.id.listViewUsers);

      dbHelper = new
UserDatabaseHelper(this);
      loadUsers();


buttonAdd.setOnClickListener(new
```

```
View.OnClickListener() {
         @Override
         public void onClick(View v) {
            String name =
editTextName.getText().toString().trim(
);
            String email =
editTextEmail.getText().toString().trim(
);

            if (!name.isEmpty() &&
!email.isEmpty()) {
               dbHelper.addUser(name,
email);
               editTextName.setText("");
               editTextEmail.setText("");
               loadUsers();
            } else {

Toast.makeText(MainActivity.this,
"Enter all fields",
Toast.LENGTH_SHORT).show();
            }
         }
      });
   }

   private void loadUsers() {
      userList = dbHelper.getAllUsers();
      adapter = new
ArrayAdapter<>(this,
android.R.layout.simple_list_item_1,
userList);

listViewUsers.setAdapter(adapter);
   }
}
```

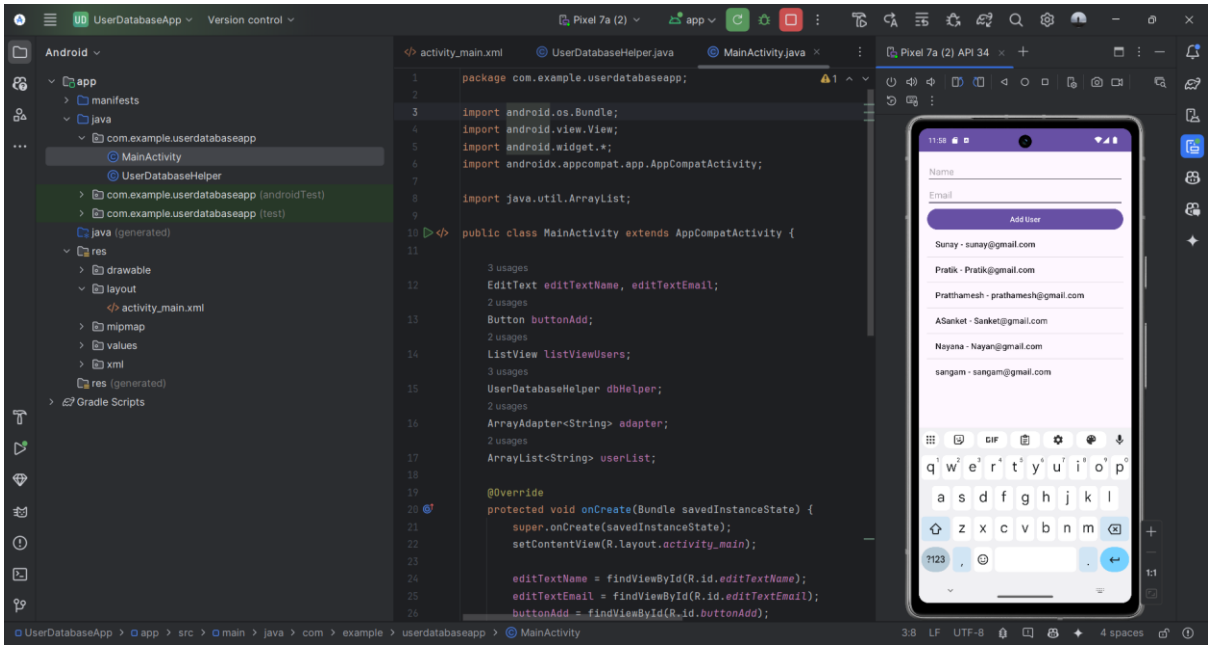
Activity_main.xml

```
<?xml version="1.0" encoding="utf-
8"?>
<LinearLayout

xmlns:android="http://schemas.andr
oid.com/apk/res/android"
```

```
android:layout_width="match_parent
"
   android:orientation="vertical"
   android:padding="16dp"
```

```
android:layout_height="match_parent
">
  <EditText
    android:id="@+id/editTextName"
android:layout_width="match_parent
"
android:layout_height="wrap_content
"
    android:hint="Name"/>

  <EditText
    android:id="@+id/editTextEmail"

android:layout_width="match_parent
"
android:layout_height="wrap_content
"
    android:hint="Email"/>
  <Button
    android:id="@+id/buttonAdd"
android:layout_width="match_parent
"
android:layout_height="wrap_content
"
    android:text="Add User"/>
  <ListView
    android:id="@+id/listViewUsers"

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

Output:



Q20: Use Firebase Realtime Database to create an app that allows users to post and view messages.

Steps:

1. **Design the UI** with an EditText (to type message), Button (to send), and ListView (to display messages).
2. **Connect Firebase** to your app (add Firebase SDK, configure google-services.json).

3. **Get a reference** to the "messages" node in Firebase Database.
4. **Send message**: On button click, **push** the new message to Firebase.
5. **Listen for updates**: Use addValueEventListener to **fetch and update** the ListView whenever the database changes.
6. **Update ListView**: Refresh the ListView with the latest messages automatically.

Code: MainActivity.java

```
package com.example.messageapp;
...
public class MainActivity extends
AppCompatActivity {

   private EditText messageEditText;
   private Button sendButton;
   private ListView messageListView;

   private ArrayList<String>
messageList;
   private ArrayAdapter<String>
adapter;

   private DatabaseReference
messagesRef;

   @Override
   protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_mai
n);

      messageEditText =
findViewById(R.id.messageEditText);
      sendButton =
findViewById(R.id.sendButton);
      messageListView =
findViewById(R.id.messageListView);

      messageList = new ArrayList<>();
      adapter = new
ArrayAdapter<>(this,
android.R.layout.simple_list_item_1,
messageList);
```

```
messageListView.setAdapter(adapter)
;

      // Reference to the "messages"
node in Firebase
      messagesRef =
FirebaseDatabase.getInstance().getR
eference("messages");


sendButton.setOnClickListener(new
View.OnClickListener() {
         @Override
         public void onClick(View v) {
            String message =
messageEditText.getText().toString().tr
im();
            if (!message.isEmpty()) {
               // Push the message to the
database

messagesRef.push().setValue(messa
ge);
               messageEditText.setText("");
            }
         }
      });

      // Listen for changes in the
database

messagesRef.addValueEventListener(
new ValueEventListener() {
         @Override
         public void
onDataChange(@NonNull
DataSnapshot snapshot) {
            messageList.clear();
```

```
            for (DataSnapshot
dataSnapshot :
snapshot.getChildren()) {
                String msg =
dataSnapshot.getValue(String.class);
                messageList.add(msg);
            }

adapter.notifyDataSetChanged();
        }
```

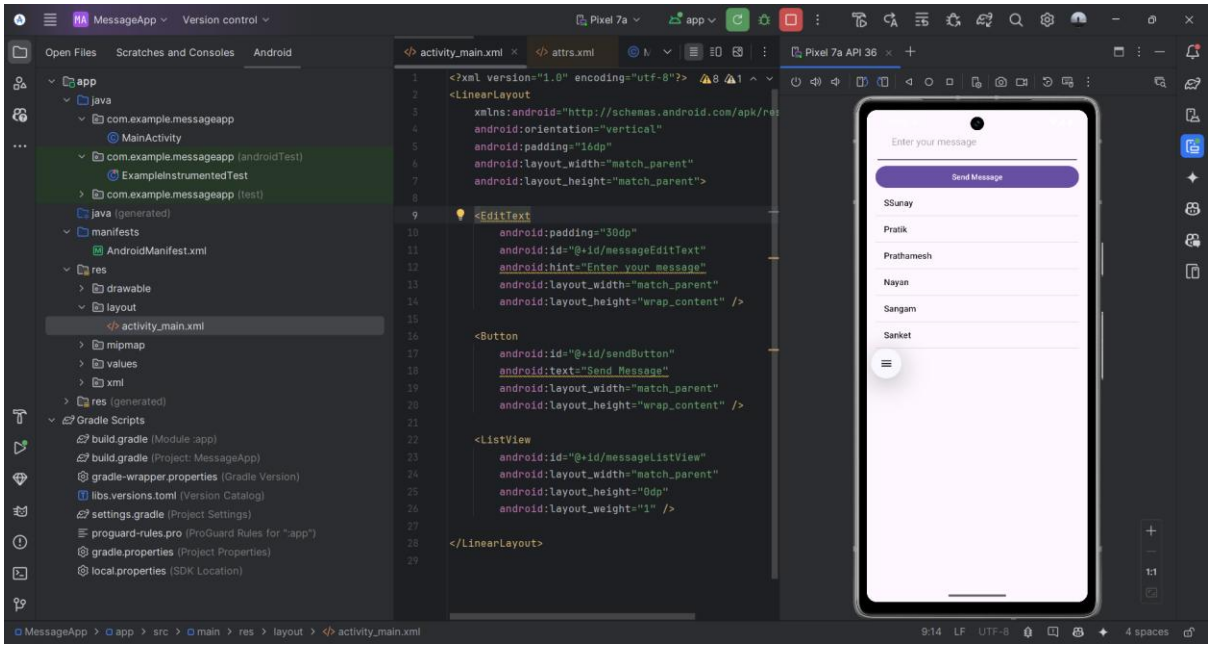
```
        @Override
        public void
onCancelled(@NonNull
DatabaseError error) {
            // Handle possible errors
        }
    });
  }
}
```

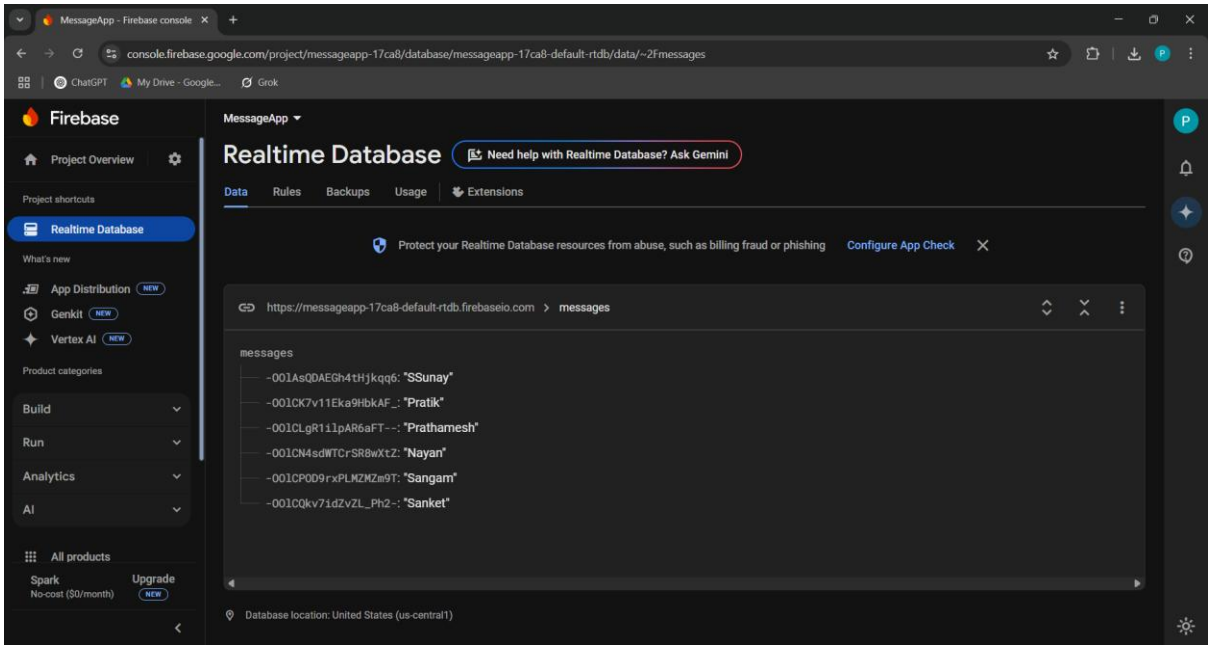
activity_main.xml

```
<?xml version="1.0" encoding="utf-
8"?>
<LinearLayout

xmlns:android="http://schemas.andr
oid.com/apk/res/android"
  android:orientation="vertical"
  android:padding="16dp"

android:layout_width="match_parent
"

android:layout_height="match_parent
">

  <EditText
    android:padding="30dp"

android:id="@+id/messageEditText"
    android:hint="Enter your
message"

android:layout_width="match_parent
"
```

```
android:layout_height="wrap_content
" />

  <Button
    android:id="@+id/sendButton"
    android:text="Send Message"

android:layout_width="match_parent
"

android:layout_height="wrap_content
" />

  <ListView

android:id="@+id/messageListView"

android:layout_width="match_parent
"
    android:layout_height="0dp"
    android:layout_weight="1" />

</LinearLayout>
```

Output:





Q21: Write a query to fetch all rows from an SQLite table and display them in a RecyclerView using a Cursor.

Steps:

1. **Create a DatabaseHelper** class with onCreate() to create the SQLite table.
2. **Open Database** in MainActivity using readableDatabase.
3. **Insert sample data** (optional) into the table.
4. **Query all rows** using rawQuery("SELECT * FROM table_name", null) to get a Cursor.
5. **Create a RecyclerView** in the layout (activity_main.xml).

6. **Implement a RecyclerView Adapter (UserAdapter)** that takes a Cursor and binds data to each ViewHolder.
7. **Set RecyclerView's adapter** with the Cursor data.

Code: MainActivity.kt

```
package com.example.q21fetch
...
class MainActivity :
AppCompatActivity() {

  private lateinit var database:
SQLiteDatabase
  private lateinit var adapter:
UserAdapter

  override fun
onCreate(savedInstanceState:
Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity_mai
n) // Ensure activity_main.xml exists

    val dbHelper =
DatabaseHelper(this)
    database =
dbHelper.readableDatabase

    // Insert sample data (if needed)
    database.execSQL("INSERT INTO
${DatabaseHelper.TABLE_NAME}
(name, age) VALUES ('John Doe', 25)")
    database.execSQL("INSERT INTO
${DatabaseHelper.TABLE_NAME}
(name, age) VALUES ('Jane Smith',
30)")

    // Fetch data
    val cursor =
database.rawQuery("SELECT * FROM
${DatabaseHelper.TABLE_NAME}",
null)

    // Set up RecyclerView
    val recyclerView: RecyclerView =
findViewById(R.id.recyclerView) //
Ensure ID matches in XML
    recyclerView.layoutManager =
LinearLayoutManager(this)
    adapter = UserAdapter(cursor) //
Ensure UserAdapter is implemented
    recyclerView.adapter = adapter
  }
}
```

DatabaseHelper.kt

```
package com.example.q21fetch
...
class DatabaseHelper(context:
Context) : SQLiteOpenHelper(context,
DATABASE_NAME, null,
DATABASE_VERSION) {

  override fun onCreate(db:
SQLiteDatabase) {
    db.execSQL("CREATE TABLE
$TABLE_NAME (id INTEGER PRIMARY
KEY, name TEXT, age INTEGER)")
  }

  override fun onUpgrade(db:
SQLiteDatabase, oldVersion: Int,
newVersion: Int) {
    db.execSQL("DROP TABLE IF
EXISTS $TABLE_NAME")
    onCreate(db)
  }

  companion object {
    const val DATABASE_NAME =
"example.db"
    const val DATABASE_VERSION = 1
```

```
    const val TABLE_NAME = "users"
  }
```

```
}
```

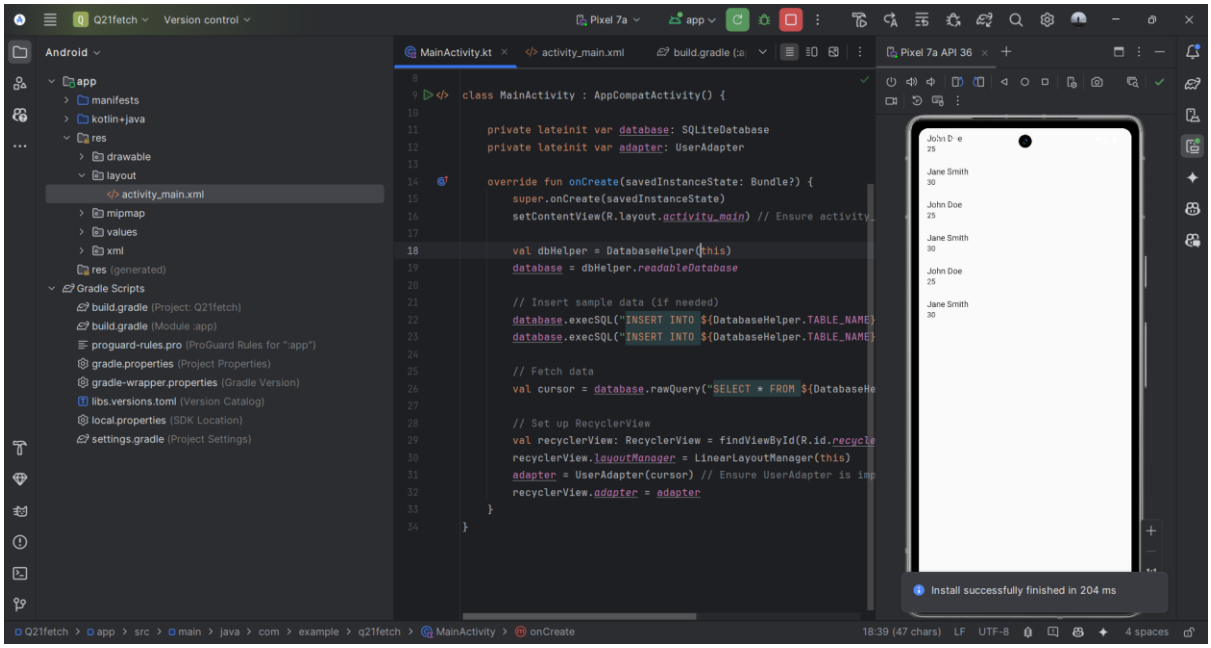
UserAdapter.kt

```
package com.example.q21fetch
...
class UserAdapter(private val cursor: Cursor) : RecyclerView.Adapter<UserAdapter.UserViewHolder>() {

  class UserViewHolder(view: View) : RecyclerView.ViewHolder(view) {
    val nameTextView: TextView = view.findViewById(android.R.id.text1)
    val ageTextView: TextView = view.findViewById(android.R.id.text2)
  }

  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): UserViewHolder {
    val view = LayoutInflater.from(parent.context)

.inflate(android.R.layout.simple_list_item_2, parent, false)
    return UserViewHolder(view)
  }

  override fun onBindViewHolder(holder: UserViewHolder, position: Int) {
    if (cursor.moveToPosition(position)) {
      holder.nameTextView.text = cursor.getString(cursor.getColumnIndexOrThrow("name"))
      holder.ageTextView.text = cursor.getInt(cursor.getColumnIndexOrThrow("age")).toString()
    }
  }

  override fun getItemCount(): Int {
    return cursor.count
  }
}
```

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">

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

Output:



Q22: Use ContentValues to insert a new record into an SQLite database.

Steps:

1. **Create a DatabaseHelper** class extending SQLiteOpenHelper.
2. **Inside onCreate()**, create a table using SQL (CREATE TABLE statement).
3. **Open writable database** using writableDatabase.
4. **Prepare a ContentValues object** and put() the data (e.g., name, email, etc.).
5. **Call insert() method** on the database, passing table name, null column hack, and ContentValues.
6. **Optionally, refresh UI** after insertion (e.g., update a list or RecyclerView).

Code: MainActivity.kt

```
package com.example.q23crud
...

class MainActivity :
ComponentActivity() {
   private lateinit var dbHelper:
DatabaseHelper

   override fun
onCreate(savedInstanceState:
Bundle?) {

super.onCreate(savedInstanceState)
      dbHelper = DatabaseHelper(this)
```

```
      setContent {
         Q23crudTheme {
            var records by remember {
mutableStateOf(dbHelper.getAllRecor
dsAsList()) }
            var newName by remember {
mutableStateOf("") }
            var showDialog by remember {
mutableStateOf(false) }
            var selectedRecord by
remember {
mutableStateOf<Record?>(null) }

            Scaffold(
               modifier =
```

```
Modifier.fillMaxSize(),
            content = { innerPadding ->
              Column(modifier =
Modifier.padding(innerPadding)) {
                LazyColumn(modifier =
Modifier.weight(1f)) {
                  items(records) { record
->
                    Row(
                      modifier = Modifier
                        .fillMaxWidth()
                        .padding(8.dp),

horizontalArrangement =
Arrangement.SpaceBetween
                    ) {
                      Text(text =
"${record.id}: ${record.name}")
                      Row {
                        Button(onClick =
{
dbHelper.deleteRecord(record.id)
                          records =
dbHelper.getAllRecordsAsList()

Toast.makeText(this@MainActivity,
"Deleted",
Toast.LENGTH_SHORT).show()
                        }) {
                          Text("Delete")
                        }
                        Spacer(modifier
= Modifier.width(8.dp))
                        Button(onClick =
{
selectedRecord = record
                          showDialog =
true
                        }) {
                          Text("Update")
                        }
                      }
                    }
                  }
                }
                if (showDialog &&
selectedRecord != null) {
                  UpdateRecordDialog(
                    record =
selectedRecord!!,
                    onDismiss = {
showDialog = false },
                    onUpdate = {
newName ->

dbHelper.updateRecord(selectedRec
ord!!.id, newName)
                      records =
dbHelper.getAllRecordsAsList()

Toast.makeText(this@MainActivity,
"Updated",
Toast.LENGTH_SHORT).show()
                    }
                  )
                }
                Row(
                  modifier = Modifier
                    .fillMaxWidth()
                    .padding(8.dp),

horizontalArrangement =
Arrangement.SpaceBetween
                ) {
                  TextField(
                    value = newName,
                    onValueChange = {
newName = it },
                    label = { Text("New
Record Name") },
                    modifier =
Modifier.weight(1f)
                  )
                  Spacer(modifier =
Modifier.width(8.dp))
                  Button(onClick = {
                    if
(newName.isNotBlank()) {

dbHelper.addRecord(newName)
                      records =
dbHelper.getAllRecordsAsList()
                      newName = ""
```

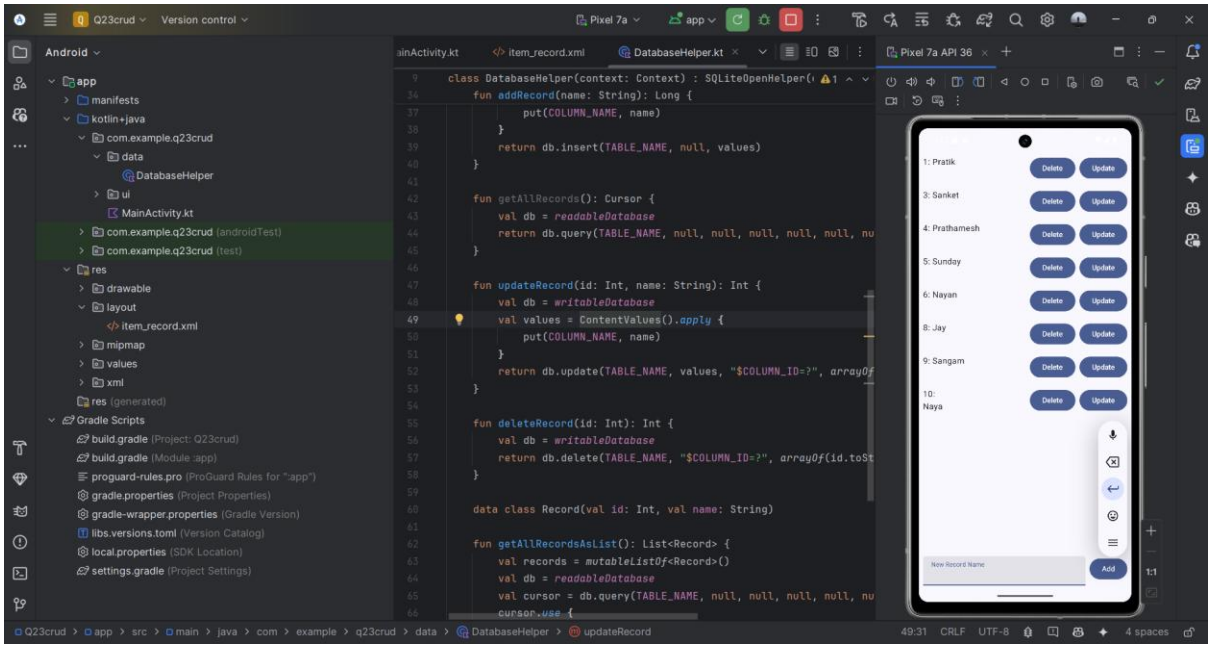
```
Toast.makeText(this@MainActivity,
"Added",
Toast.LENGTH_SHORT).show()
                        } else {

Toast.makeText(this@MainActivity,
"Name cannot be empty",
Toast.LENGTH_SHORT).show()
                        }
                    }) {
                        Text("Add")
                    }
                }
            }
        }
    )
  }
}

@Composable
fun UpdateRecordDialog(
  record: Record,
  onDismiss: () -> Unit,
  onUpdate: (String) -> Unit
) {
  var updatedName by remember { mutableStateOf(record.name) }

  AlertDialog(
    onDismissRequest = onDismiss,
    title = { Text("Update Record") },
    text = {
      TextField(
        value = updatedName,
        onValueChange = { updatedName = it },
        label = { Text("New Name") }
      )
    },
    confirmButton = {
      TextButton(onClick = {
        onUpdate(updatedName)
        onDismiss()
      }) {
        Text("Update")
      }
    },
    dismissButton = {
      TextButton(onClick = onDismiss) {
        Text("Cancel")
      }
    }
  )
}

@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
  Q23crudTheme {
    Text("Preview")
  }
}
```

DataBaseHelper.kt

```
package com.example.q23crud.data
...
class DatabaseHelper(context:
Context) : SQLiteOpenHelper(context,
DATABASE_NAME, null,
DATABASE_VERSION) {

  companion object {
    private const val
DATABASE_NAME = "Q23crud.db"
    private const val
DATABASE_VERSION = 1
    private const val TABLE_NAME =
"records"
    private const val COLUMN_ID =
"id"
    private const val COLUMN_NAME
= "name"
  }

  override fun onCreate(db:
SQLiteDatabase) {
```

```
    val createTableQuery = """
        CREATE TABLE $TABLE_NAME (
            $COLUMN_ID INTEGER
PRIMARY KEY AUTOINCREMENT,
            $COLUMN_NAME TEXT NOT
NULL
        )
    """
    db.execSQL(createTableQuery)
  }

  override fun onUpgrade(db:
SQLiteDatabase, oldVersion: Int,
newVersion: Int) {
    db.execSQL("DROP TABLE IF
EXISTS $TABLE_NAME")
    onCreate(db)
  }

  fun addRecord(name: String): Long {
    val db = writableDatabase
    val values =
ContentValues().apply {
        put(COLUMN_NAME, name)
    }
    return db.insert(TABLE_NAME,
null, values)
  }

  fun getAllRecords(): Cursor {
    val db = readableDatabase
    return db.query(TABLE_NAME,
null, null, null, null, null, null)
  }

  fun updateRecord(id: Int, name:
String): Int {
    val db = writableDatabase
    val values =
ContentValues().apply {
        put(COLUMN_NAME, name)
    }
    return db.update(TABLE_NAME,
values, "$COLUMN_ID=?",
arrayOf(id.toString()))
  }

  fun deleteRecord(id: Int): Int {
    val db = writableDatabase
    return db.delete(TABLE_NAME,
"$COLUMN_ID=?",
arrayOf(id.toString()))
  }

  data class Record(val id: Int, val
name: String)

  fun getAllRecordsAsList():
List<Record> {
    val records =
mutableListOf<Record>()
    val db = readableDatabase
    val cursor =
db.query(TABLE_NAME, null, null,
null, null, null, null)
    cursor.use {
      if (it.moveToFirst()) {
        do {
          val id =
it.getInt(it.getColumnIndexOrThrow(C
OLUMN_ID))
          val name =
it.getString(it.getColumnIndexOrThro
w(COLUMN_NAME))
          records.add(Record(id,
name))
        } while (it.moveToNext())
      }
    }
    return records
  }
}
```

Output:



Q23: Implement a complete SQLite CRUD operation: Add a new record. View all records in a RecyclerView. Update a record. Delete a record.

Steps:

1. **Create a DatabaseHelper** extending SQLiteOpenHelper with methods for addRecord(), getAllRecords(), updateRecord(), and deleteRecord().
2. **Design item layout** (item_record.xml) for RecyclerView items.
3. **Fetch all records** using a query (SELECT * FROM table) and show them in a **LazyColumn** (or RecyclerView).
4. **Add a new record** using a TextField + Button to call addRecord() and refresh the list.
5. **Delete a record** by clicking a "Delete" button, calling deleteRecord(), and refreshing.
6. **Update a record** (if implemented) by showing a dialog, modifying the record, and calling updateRecord().
7. **Refresh UI** after each operation to reflect changes.

Code: **MainActivity.kt**

```
package com.example.q23crud
...
class MainActivity :
ComponentActivity() {
   private lateinit var dbHelper:
DatabaseHelper

   override fun
onCreate(savedInstanceState:
Bundle?) {
```

```
super.onCreate(savedInstanceState)
      dbHelper = DatabaseHelper(this)

      setContent {
         Q23crudTheme {
            var records by remember {
mutableStateOf(dbHelper.getAllRecor
dsAsList()) }
            var newName by remember {
```

```
mutableStateOf("") }

        Scaffold(
            modifier =
Modifier.fillMaxSize(),
            content = { innerPadding ->
                Column(modifier =
Modifier.padding(innerPadding)) {
                    LazyColumn(modifier =
Modifier.weight(1f)) {
                        items(records) { record
->
                            Row(
                                modifier = Modifier
                                    .fillMaxWidth()
                                    .padding(8.dp),
horizontalArrangement =
Arrangement.SpaceBetween
                            ) {
                                Text(text =
"${record.id}: ${record.name}")
                                Button(onClick = {

dbHelper.deleteRecord(record.id)
                                    records =
dbHelper.getAllRecordsAsList()

Toast.makeText(this@MainActivity,
"Deleted",
Toast.LENGTH_SHORT).show()
                                }) {
                                    Text("Delete")
                                }
                            }
                        }
                    }
                    Row(
                        modifier = Modifier
                            .fillMaxWidth()
                            .padding(8.dp),

horizontalArrangement =
Arrangement.SpaceBetween
                    ) {
                        TextField(
                            value = newName,
                            onValueChange = {
newName = it },
                            label = { Text("New
Record Name") },
                            modifier =
Modifier.weight(1f)
                        )
                        Spacer(modifier =
Modifier.width(8.dp))
                        Button(onClick = {
                            if
(newName.isNotBlank()) {

dbHelper.addRecord(newName)
                                records =
dbHelper.getAllRecordsAsList()
                                newName = ""

Toast.makeText(this@MainActivity,
"Added",
Toast.LENGTH_SHORT).show()
                            } else {

Toast.makeText(this@MainActivity,
"Name cannot be empty",
Toast.LENGTH_SHORT).show()
                            }
                        }) {
                            Text("Add")
                        }
                    }
                }
            )
        }
    }
  }
}

@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
  Q23crudTheme {
    Text("Preview")
  }
}
```

DataBasehelper.kt

```
package com.example.q23crud.data
...
class DatabaseHelper(context:
Context) : SQLiteOpenHelper(context,
DATABASE_NAME, null,
DATABASE_VERSION) {

  companion object {
    private const val
DATABASE_NAME = "Q23crud.db"
    private const val
DATABASE_VERSION = 1
    private const val TABLE_NAME =
"records"
    private const val COLUMN_ID =
"id"
    private const val COLUMN_NAME
= "name"
  }

  override fun onCreate(db:
SQLiteDatabase) {
    val createTableQuery = """
      CREATE TABLE $TABLE_NAME (
        $COLUMN_ID INTEGER
PRIMARY KEY AUTOINCREMENT,
        $COLUMN_NAME TEXT NOT
NULL
      )
    """
    db.execSQL(createTableQuery)
  }

  override fun onUpgrade(db:
SQLiteDatabase, oldVersion: Int,
newVersion: Int) {
    db.execSQL("DROP TABLE IF
EXISTS $TABLE_NAME")
    onCreate(db)
  }

  fun addRecord(name: String): Long {
    val db = writableDatabase
    val values =
ContentValues().apply {
      put(COLUMN_NAME, name)
    }
    return db.insert(TABLE_NAME,
null, values)
  }

  fun getAllRecords(): Cursor {
    val db = readableDatabase
    return db.query(TABLE_NAME,
null, null, null, null, null, null)
  }

  fun updateRecord(id: Int, name:
String): Int {
    val db = writableDatabase
    val values =
ContentValues().apply {
      put(COLUMN_NAME, name)
    }
    return db.update(TABLE_NAME,
values, "$COLUMN_ID=?",
arrayOf(id.toString()))
  }

  fun deleteRecord(id: Int): Int {
    val db = writableDatabase
    return db.delete(TABLE_NAME,
"$COLUMN_ID=?",
arrayOf(id.toString()))
  }

  data class Record(val id: Int, val
name: String)

  fun getAllRecordsAsList():
List<Record> {
    val records =
mutableListOf<Record>()
    val db = readableDatabase
    val cursor =
db.query(TABLE_NAME, null, null,
null, null, null, null)
    cursor.use {
      if (it.moveToFirst()) {
        do {
```

```
            val id =
it.getInt(it.getColumnIndexOrThrow(C
OLUMN_ID))
            val name =
it.getString(it.getColumnIndexOrThro
w(COLUMN_NAME))
            records.add(Record(id,
```

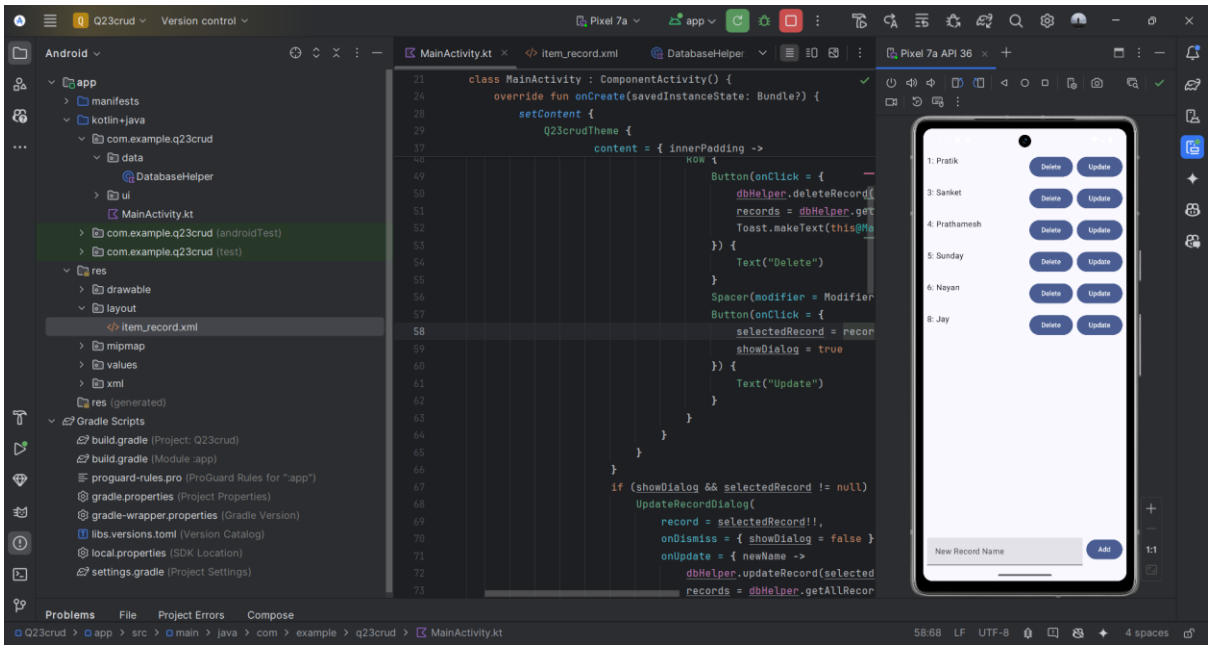
```
name))
            } while (it.moveToNext())
        }
    }
    return records
  }
}
```

Item_record.xml

```
<LinearLayout
xmlns:android="http://schemas.andr
oid.com/apk/res/android"
android:layout_width="match_parent
"
android:layout_height="wrap_content
"
  android:orientation="horizontal"
  android:padding="8dp">
  <TextView
    android:id="@+id/record_name"
    android:layout_width="0dp"

android:layout_height="wrap_content
```

```
"
    android:layout_weight="1"
    android:text="Record Name" />
  <Button
    android:id="@+id/delete_button"

android:layout_width="wrap_content"

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

Output:



24. Create an app that fetches data from a public API (e.g., OpenWeatherMap) and displays it in a TextView.

Steps:

General Steps for Fetching Weather Data:

1. Prepare the Environment

- **Install required dependencies** for networking, such as Retrofit, Gson, or any other HTTP client library for API communication.

2. Design the User Interface

- Create a layout that includes a TextView to display the fetched weather data.
- Optionally, add an EditText for the user to enter a city name, and a Button to trigger the API request.

3. Setup Network Client (e.g., Retrofit)

- **Configure the client** to communicate with the weather API.
- Provide **API endpoint details** (like OpenWeatherMap's URL) and any required parameters (e.g., city, API key).

4. Create Data Model (Weather Response)

- **Model the JSON response**: Create classes to match the data structure returned by the API (e.g., temperature, weather description).
- Map fields such as city name, temperature, humidity, etc.

5. Make the API Call

- **Send a network request** to the weather API using the HTTP client.
- Include necessary parameters like city name and API key.
- Handle the **response** asynchronously to prevent blocking the UI thread.

6. Handle the Response

- **Parse the response**: Convert the JSON data into your model objects.
- **Check for success or failure**: Handle the case where the API responds with data and where it fails (e.g., no internet, invalid API key, etc.).

7. Display the Data in UI

- Once the data is fetched and parsed, update the TextView with the relevant weather details (e.g., temperature, weather condition, city name).

8. Error Handling

- Handle any errors that occur during the network request (e.g., no internet, invalid city) and show appropriate messages to the user (using Toast, AlertDialog, etc.).

9. Test the App

- **Run the app** and test various cities to ensure that weather data is displayed correctly.
- Check edge cases (e.g., no internet connection, wrong city name).

Code: Activitymain.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent”
   android:orientation="horizontal"
   android:padding="16dp">
   <TextView
```

```
android:id="@+id/weatherTextView"

android:layout_width="match_parent"


android:layout_height="match_parent"
      android:text="Weather Info"

android:textAppearance="@style/TextAppearance.Material3.BodyLarge"
      android:textSize="18sp" />
</LinearLayout>
```

MainActivity.java

```
package com.example.weatherapp;
...
public class MainActivity extends AppCompatActivity {
   private TextView weatherTextView;
   private static final String API_KEY = "571934a18c733ecaac0fb4832d425bf7"; // Replace with your OpenWeatherMap API key
   private static final String TAG = "MainActivity";
   @Override
   protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
      weatherTextView = findViewById(R.id.weatherTextView);
      // Initialize Retrofit
      Retrofit retrofit = WeatherApiClient.getClient();
      WeatherApiService service = retrofit.create(WeatherApiService.class);
      // Make the network request to OpenWeatherMap
      Call<WeatherResponse> call =
```

```
service.getWeather("London", API_KEY);
      call.enqueue(new Callback<WeatherResponse>() {
         @Override
         public void onResponse(Call<WeatherResponse> call, Response<WeatherResponse> response) {
            if (response.isSuccessful() && response.body() != null) {
               WeatherResponse weatherResponse = response.body();
               String weatherInfo = "City: " + weatherResponse.getName() + "\n"
                     + "Temperature: " + (weatherResponse.getMain().getTemp() - 273.15) + "°C";

weatherTextView.setText(weatherInfo);
            } else {
               Log.e(TAG, "Error: " + response.code() + " - " + response.message());

Toast.makeText(MainActivity.this, "Error fetching data",
```

```
Toast.LENGTH_SHORT).show();
            }
        }
        @Override
        public void
onFailure(Call<WeatherResponse>
call, Throwable t) {
            Log.e(TAG, "Network failure: "
+ t.getMessage(), t);
```

```
Toast.makeText(MainActivity.this,
"Network failure",
Toast.LENGTH_SHORT).show();
        }
    });
  }
}
```

Weatherapiservice.java

```
package com.example.weatherapp;
import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Query;
public interface WeatherApiService {
```

```
  @GET("weather")
  Call<WeatherResponse>
getWeather(@Query("q") String city,
@Query("appid") String apiKey);
}
```

WeatherResponse.java

```
package com.example.weatherapp;
public class WeatherResponse {
  private Main main;
  private String name;
  public Main getMain() {
    return main;
  }
  public String getName() {
    return name;
```

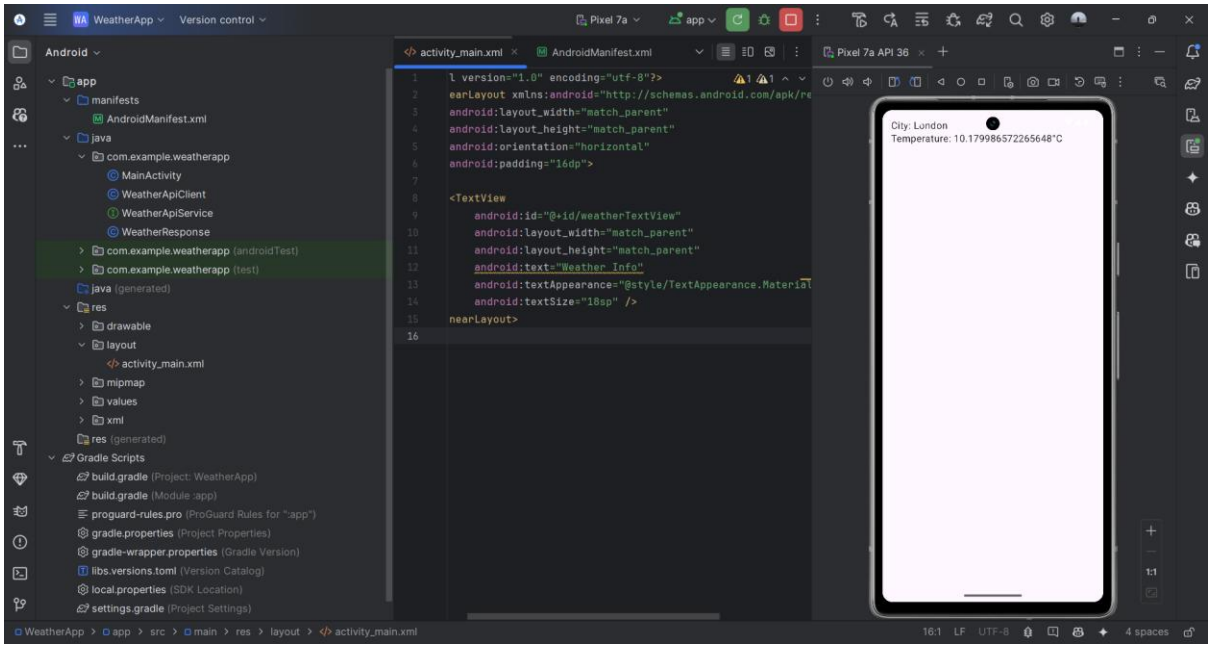
```
  }
  public class Main {
    private float temp;
    public float getTemp() {
      return temp;
    }
  }
}
```

WeatherApiclient.java

```
package com.example.weatherapp;
public class WeatherApiClient {
  private static final String BASE_URL
=
"https://api.openweathermap.org/dat
a/2.5/"; // Ensure trailing slash
  private static Retrofit retrofit = null;
  public static Retrofit getClient() {
    if (retrofit == null) {
      retrofit = new Retrofit.Builder()
```

```
            .baseUrl(BASE_URL)

.addConverterFactory(GsonConverter
Factory.create())
            .build();
    }
    return retrofit;
  }
}
```

Output:


WeatherApp
Version control
Pixel 7a
app
Android
activity_main.xml
AndroidManifest.xml
Pixel 7a API 36
City: London
Temperature: 10.179986572265648°C
WeatherApp › app › src › main › res › layout › activity_main.xml
16:1 LF UTF-8 4 spaces