

**BỘ NÔNG NGHIỆP VÀ MÔI TRƯỜNG
TRƯỜNG ĐẠI HỌC THỦY LỢI**



BÀI TẬP THỰC HÀNH SỐ 5

**PHÁT TRIỂN ÚNG DỤNG CHO THIẾT BỊ DI ĐỘNG
NỘI DUNG BỔ SUNG: ÚNG DỤNG VỚI CHỦ ĐỀ NÂNG CAO**

STT	Mã sinh viên	Họ và tên	Lớp
1	2251061863	Lê Hà Phương	64CNTT1

Hà Nội, năm 2025

BÀI TẬP 1: Content Providers

Mục tiêu:

- Tìm hiểu cách sử dụng Content Providers để truy cập dữ liệu từ ứng dụng khác (ứng dụng Danh bạ).
- Hiển thị danh sách tên các liên hệ trong danh bạ lên ứng dụng của mình.

Các bước thực hiện:

1. Thiết lập quyền truy cập:

- Mở file `AndroidManifest.xml` của ứng dụng.
- Thêm quyền `READ_CONTACTS` để xin phép ứng dụng được đọc dữ liệu danh bạ.

XML

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

2. Thiết kế giao diện (layout):

- Tạo một `ListView` trong file layout (ví dụ: `activity_main.xml`) để hiển thị danh sách tên liên hệ.

XML

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

3. Đọc dữ liệu từ Content Provider:

- Trong Activity chính (ví dụ: `MainActivity.kt`), sử dụng `ContentResolver` để truy vấn dữ liệu từ Content Provider của ứng dụng Danh bạ.
- URI của Content Provider Danh bạ là:
`ContactsContract.Contacts.CONTENT_URI`
- Sử dụng phương thức `query()` của `ContentResolver` để lấy dữ liệu. Phương thức này trả về một `Cursor` chưa kết quả truy vấn.
- Duyệt `Cursor` để lấy tên của từng liên hệ và lưu vào một `ArrayList<String>`.

4. Hiển thị dữ liệu lên ListView:

- Tạo một `ArrayAdapter<String>` để đưa dữ liệu từ `ArrayList<String>` lên `ListView`.
- Gán `ArrayAdapter<String>` cho `ListView`.

Code ví dụ (`MainActivity.kt`):

Kotlin

```
import android.Manifest  
import android.content.pm.PackageManager  
import android.database.Cursor  
import android.os.Bundle  
import android.provider.ContactsContract  
import android.widget.ArrayAdapter  
import android.widget.ListView  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.app.ActivityCompat  
import androidx.core.content.ContextCompat
```

```

class MainActivity : AppCompatActivity() {

    private lateinit var listViewContacts: ListView
    private lateinit var contactsList: ArrayList<String>
    private lateinit var contactsAdapter: ArrayAdapter<String>

    private val REQUEST_READ_CONTACTS_PERMISSION = 100

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

        listViewContacts = findViewById(R.id.listViewContacts)
        contactsList = ArrayList()

        // Kiểm tra và xin quyền READ_CONTACTS
        if (ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.READ_CONTACTS
            ) != PackageManager.PERMISSION_GRANTED
        ) {
            ActivityCompat.requestPermissions(
                this,
                arrayOf(Manifest.permission.READ_CONTACTS),
                REQUEST_READ_CONTACTS_PERMISSION
            )
        } else {
            loadContacts()
        }
    }

    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions,
        grantResults)
        if (requestCode == REQUEST_READ_CONTACTS_PERMISSION) {
            if (grantResults.isNotEmpty() && grantResults[0] ==
            PackageManager.PERMISSION_GRANTED) {
                loadContacts()
            } else {
                // Xử lý trường hợp người dùng từ chối cấp quyền
                // Ví dụ: Hiển thị thông báo cho người dùng
                // Giải thích lý do cần quyền truy cập danh bạ
                Toast.makeText(this, "Permission denied",
                Toast.LENGTH_SHORT).show()
            }
        }
    }

    private fun loadContacts() {
        // Lấy dữ liệu từ Content Provider
        val cursor: Cursor? = contentResolver.query(
            ContactsContract.Contacts.CONTENT_URI,
            null,
            null,
            null,
            null
        )

        cursor?.use {

```

```

        if (it.count > 0) {
            while (it.moveToNext()) {
                val nameIndex =
                    it.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)
                val name = it.getString(nameIndex)
                contactsList.add(name)
            }
        }
    }

    // Hiển thị dữ liệu lên ListView
    contactsAdapter = ArrayAdapter(this,
        android.R.layout.simple_list_item_1, contactsList)
    listViewContacts.adapter = contactsAdapter
}
}

```

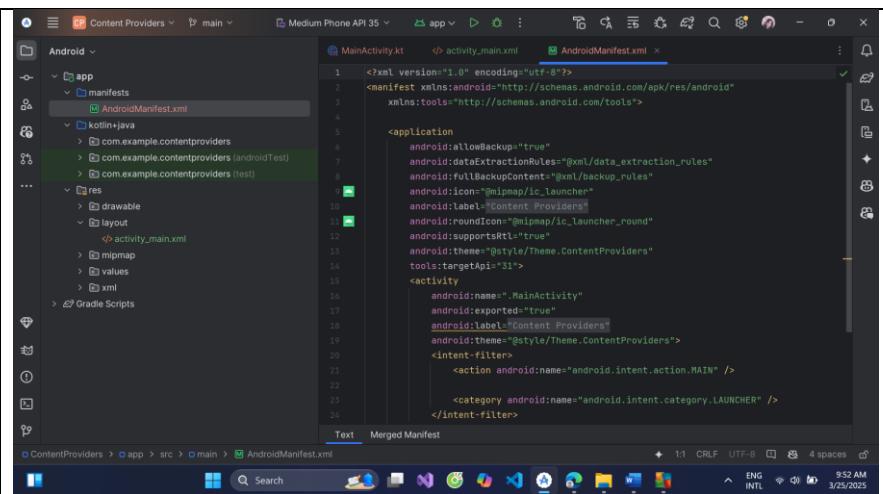
Giải thích chi tiết (Kotlin):

- `Manifest.permission.READ_CONTACTS`: Khai báo quyền truy cập danh bạ trong `AndroidManifest.xml`.
- `ContextCompat.checkSelfPermission()`: Kiểm tra xem ứng dụng đã được cấp quyền `READ_CONTACTS` hay chưa.
- `ActivityCompat.requestPermissions()`: Xin quyền `READ_CONTACTS` từ người dùng nếu chưa được cấp.
- `onRequestPermissionsResult()`: Xử lý kết quả trả về khi người dùng cấp hoặc từ chối quyền.
- `contentResolver`: Đối tượng `ContentResolver` cho phép ứng dụng tương tác với Content Providers.
- `ContactsContract.Contacts.CONTENT_URI`: URI của Content Provider chứa dữ liệu về các liên hệ.
- `Cursor`: Một interface đại diện cho tập kết quả của một truy vấn cơ sở dữ liệu. Trong trường hợp này, nó chứa dữ liệu từ Content Provider.
- `cursor?.use {}`: Sử dụng `use` block để tự động đóng `Cursor` sau khi sử dụng, tránh rò rỉ tài nguyên.
- `it.count`: Lấy số lượng hàng trong `Cursor`.
- `it.moveToNext()`: Di chuyển đến hàng tiếp theo trong `Cursor`.
- `it.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)`: Lấy chỉ số của cột chứa tên hiển thị của liên hệ.
- `it.getString(nameIndex)`: Lấy giá trị kiểu String từ cột tại chỉ số đã cho.
- `ArrayAdapter<String>`: Adapter để hiển thị danh sách các chuỗi (tên liên hệ) lên `ListView`.
- `listViewContacts.adapter`: Gán `ArrayAdapter` cho `ListView` để hiển thị dữ liệu.

Hướng dẫn Bài tập 01: Chụp lại mà hình từng bước thực hiện (tương tự cho các Bài tập bên dưới) để học sinh lớp 10 có thể thực hiện lại theo được :D

1. Thiết lập quyền truy cập:

- Mở file `AndroidManifest.xml` của ứng dụng.



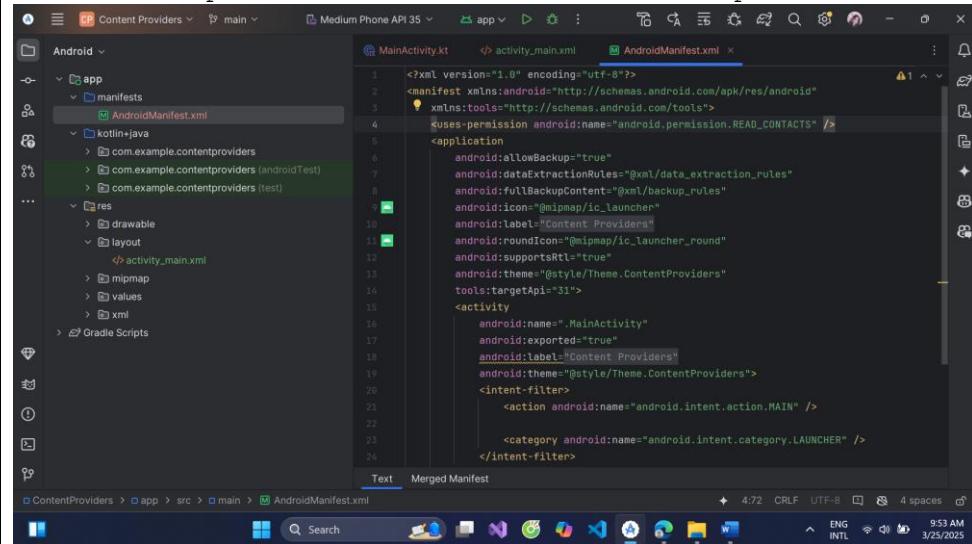
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Content Providers"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ContentProviders"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="Content Providers"
            android:theme="@style/Theme.ContentProviders">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- Thêm quyền READ_CONTACTS để xin phép ứng dụng được đọc dữ liệu danh bạ.

XML

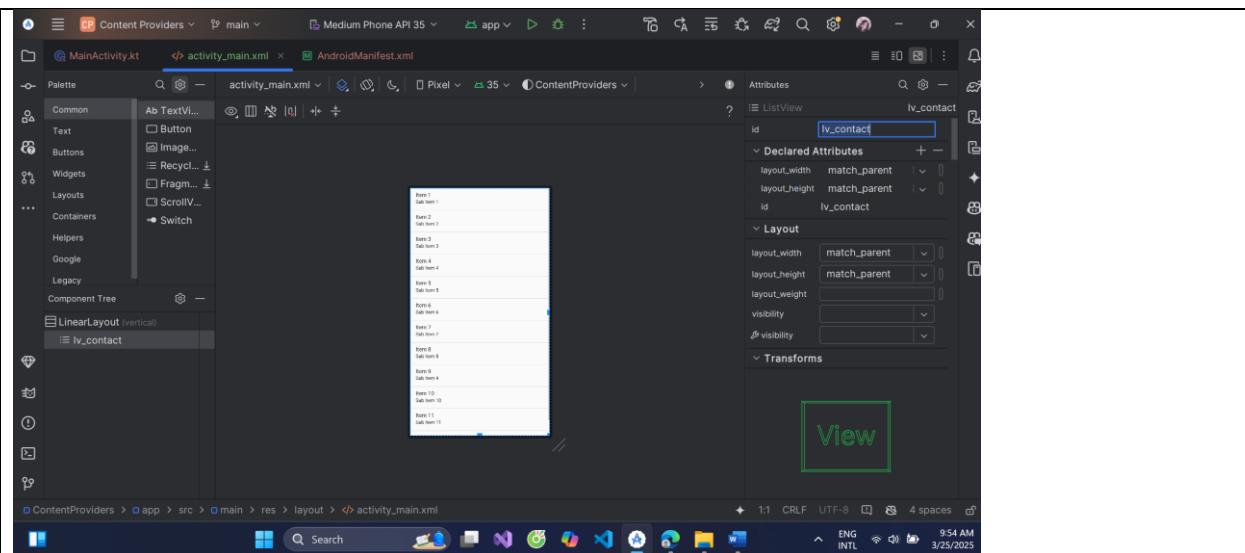
<uses-permission android:name="android.permission.READ_CONTACTS" />



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Content Providers"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ContentProviders"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="Content Providers"
            android:theme="@style/Theme.ContentProviders">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

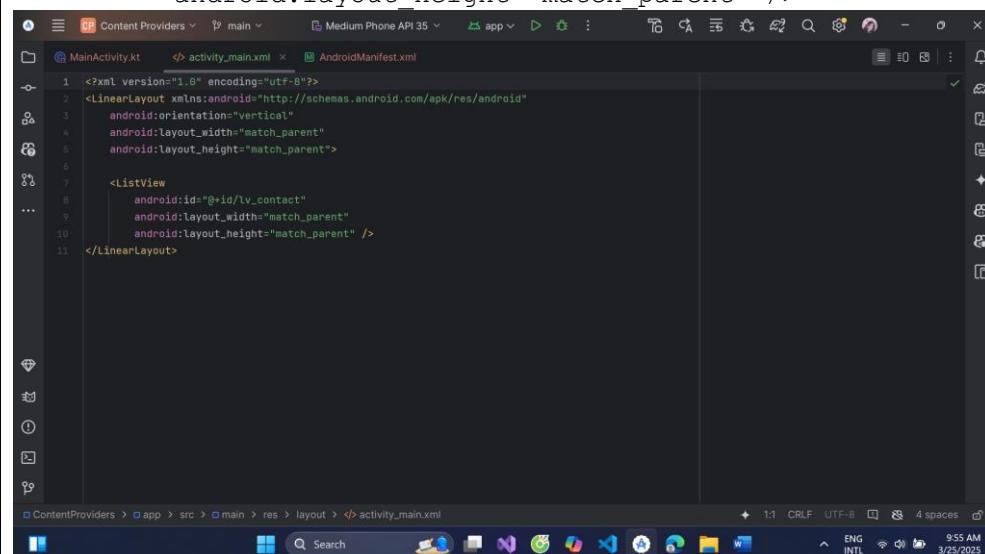
2. Thiết kế giao diện (layout):

- Tạo một ListView trong file layout (ví dụ: activity_main.xml) để hiển thị danh sách tên liên hệ.



XML

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



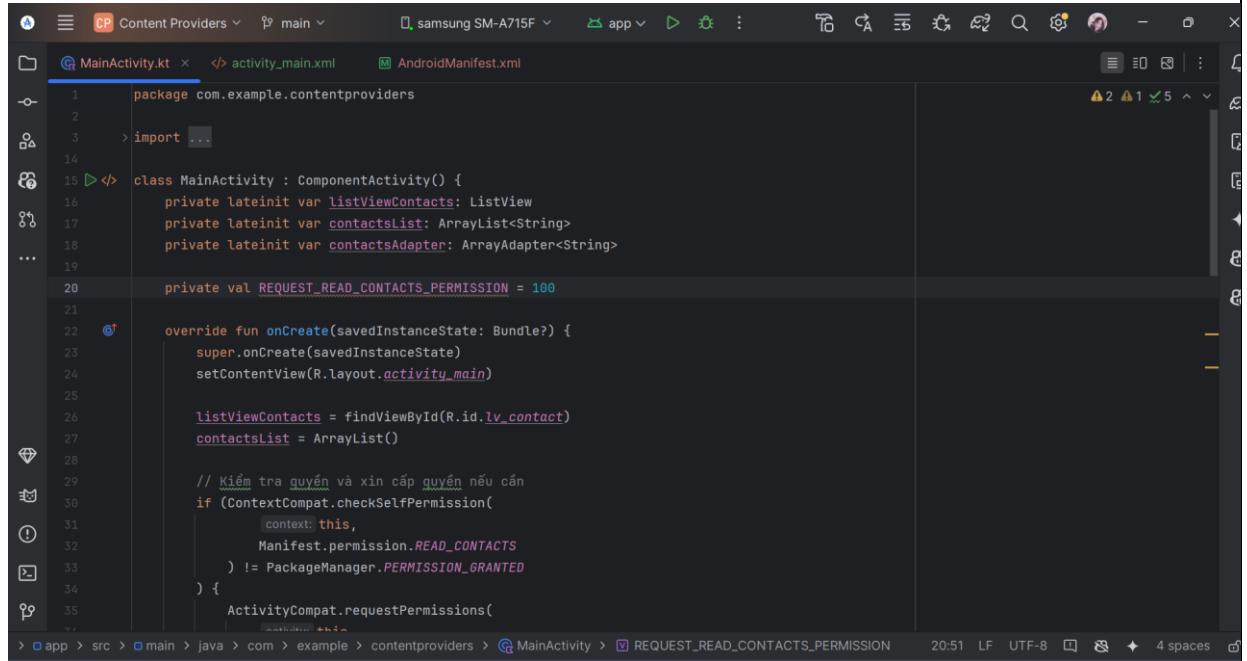
3. Đọc dữ liệu từ Content Provider:

- Trong Activity chính (ví dụ: MainActivity.kt), sử dụng ContentResolver để truy vấn dữ liệu từ Content Provider của ứng dụng Danh bạ.
- URI của Content Provider Danh bạ là:
ContactsContract.Contacts.CONTENT_URI
- Sử dụng phương thức query() của ContentResolver để lấy dữ liệu. Phương thức này trả về một Cursor chứa kết quả truy vấn.
- Duyệt Cursor để lấy tên của từng liên hệ và lưu vào một ArrayList<String>.

4. Hiển thị dữ liệu lên ListView:

- Tạo một ArrayAdapter<String> để đưa dữ liệu từ ArrayList<String> lên ListView.

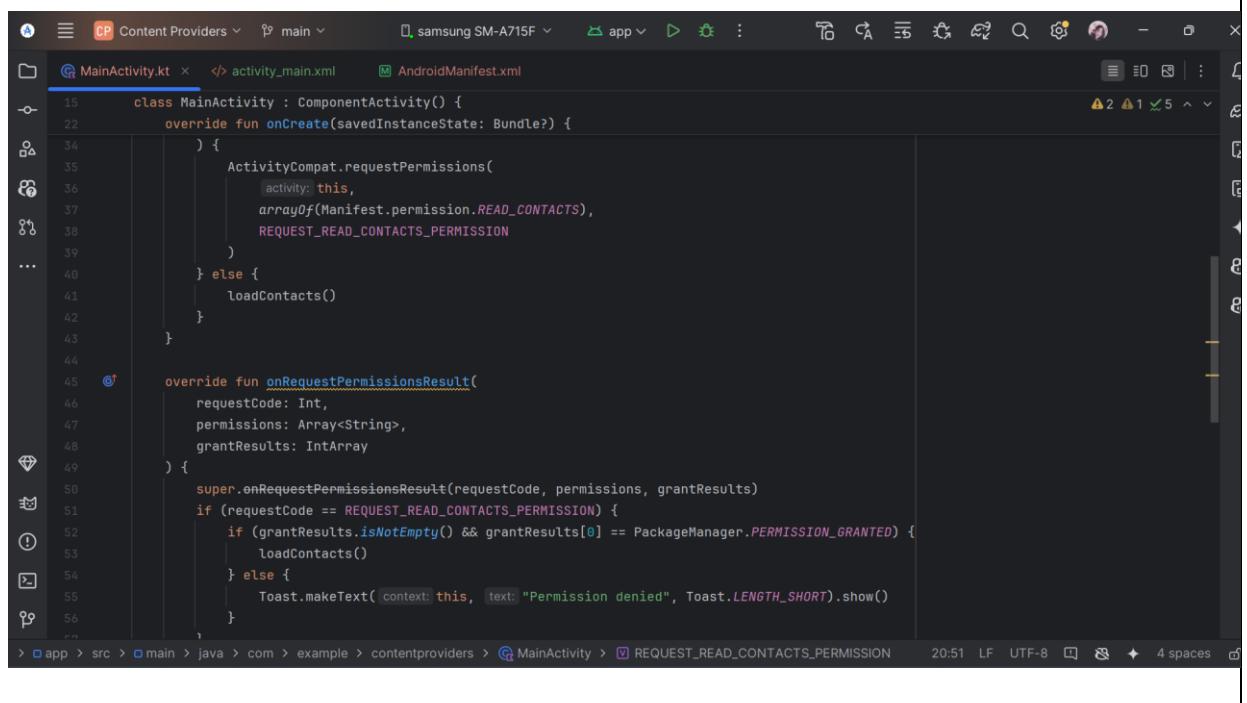
- o Gán ArrayAdapter<String> cho ListView.



```

1 package com.example.contentproviders
2
3 import ...
4
5 class MainActivity : ComponentActivity() {
6     private lateinit var listViewContacts: ListView
7     private lateinit var contactsList: ArrayList<String>
8     private lateinit var contactsAdapter: ArrayAdapter<String>
9
10    private val REQUEST_READ_CONTACTS_PERMISSION = 100
11
12    override fun onCreate(savedInstanceState: Bundle?) {
13        super.onCreate(savedInstanceState)
14        setContentView(R.layout.activity_main)
15
16        listViewContacts = findViewById(R.id.lv_contact)
17        contactsList = ArrayList()
18
19        // Kiểm tra quyền và xin cấp quyền nếu cần
20        if (ContextCompat.checkSelfPermission(
21            context, Manifest.permission.READ_CONTACTS
22        ) != PackageManager.PERMISSION_GRANTED) {
23            ActivityCompat.requestPermissions(
24                this, arrayOf(Manifest.permission.READ_CONTACTS),
25                REQUEST_READ_CONTACTS_PERMISSION
26            )
27        }
28    }
29
30    override fun onRequestPermissionsResult(
31        requestCode: Int,
32        permissions: Array<String>,
33        grantResults: IntArray
34    ) {
35        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
36        if (requestCode == REQUEST_READ_CONTACTS_PERMISSION) {
37            if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
38                loadContacts()
39            } else {
40                Toast.makeText(context, "Permission denied", Toast.LENGTH_SHORT).show()
41            }
42        }
43    }
44}

```



```

1 package com.example.contentproviders
2
3 import ...
4
5 class MainActivity : ComponentActivity() {
6     override fun onCreate(savedInstanceState: Bundle?) {
7         super.onCreate(savedInstanceState)
8         ActivityCompat.requestPermissions(
9             this,
10            arrayOf(Manifest.permission.READ_CONTACTS),
11            REQUEST_READ_CONTACTS_PERMISSION
12        )
13    }
14
15    override fun onRequestPermissionsResult(
16        requestCode: Int,
17        permissions: Array<String>,
18        grantResults: IntArray
19    ) {
20        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
21        if (requestCode == REQUEST_READ_CONTACTS_PERMISSION) {
22            if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
23                loadContacts()
24            } else {
25                Toast.makeText(context, "Permission denied", Toast.LENGTH_SHORT).show()
26            }
27        }
28    }
29}

```

```
15     class MainActivity : ComponentActivity() {
16         override fun onRequestPermissionsResult(
17             requestCode: Int, permissions: Array<String>, grantResults: IntArray) {
18             if (requestCode == REQUEST_READ_CONTACTS_PERMISSION) {
19                 if (grantResults.all { it == PackageManager.PERMISSION_GRANTED }) {
20                     loadContacts()
21                 } else {
22                     Toast.makeText(this, "Permission denied", Toast.LENGTH_SHORT).show()
23                 }
24             }
25         }
26     }
27
28     private fun loadContacts() {
29         val cursor: Cursor? = contentResolver.query(
30             ContactsContract.Contacts.CONTENT_URI,
31             arrayOf(ContactsContract.Contacts.DISPLAY_NAME), // Chỉ lấy cột tên
32             selection: null,
33             selectionArgs: null,
34             sortOrder: ContactsContract.Contacts.DISPLAY_NAME + " ASC" // Sắp xếp theo tên
35         )
36
37         cursor?.use {
38             if (it.count > 0) {
39                 while (it.moveToNext()) {
40                     val name = it.getString(columnIndex: 0) // Lấy tên từ cột đầu tiên
41                     contactsList.add(name)
42                 }
43             }
44         }
45     }
46
47     // Hiển thị danh sách tên lên ListView
48     contactsAdapter = ArrayAdapter(context: this, android.R.layout.simple_list_item_1, contactsList)
49     listViewContacts.adapter = contactsAdapter
50 }
```

```
51     }
52
53     // Hiển thị danh sách tên lên ListView
54     contactsAdapter = ArrayAdapter(context: this, android.R.layout.simple_list_item_1, contactsList)
55     listViewContacts.adapter = contactsAdapter
56 }
```

BÀI TẬP 2: Ứng dụng tự động trả lời tin nhắn cuộc gọi nhỡ

Mục tiêu:

- Sử dụng Broadcast Receiver để lắng nghe sự kiện cuộc gọi đến.
- Sử dụng Telephony API để lấy thông tin về cuộc gọi (số điện thoại).
- Sử dụng SMS API để gửi tin nhắn SMS.

Mô tả:

Ứng dụng sẽ tự động gửi một tin nhắn SMS đến số điện thoại của người gọi nhỡ, với nội dung thông báo rằng bạn đang bận và sẽ gọi lại sau.

Các bước thực hiện:

1. Khai báo quyền:

- Thêm các quyền cần thiết vào AndroidManifest.xml:
 - android.permission.READ_PHONE_STATE (để theo dõi trạng thái cuộc gọi)
 - android.permission.SEND_SMS (để gửi tin nhắn SMS)
 - android.permission.RECEIVE_SMS (nếu muốn xử lý cả tin nhắn đến)

2. Tạo Broadcast Receiver:

- Tạo một class kế thừa BroadcastReceiver để lắng nghe sự kiện android.intent.action.PHONE_STATE.
- Trong phương thức onReceive():
 - Kiểm tra trạng thái cuộc gọi (TelephonyManager.EXTRA_STATE_RINGING).
 - Nếu là cuộc gọi đến, lấy số điện thoại người gọi (intent.getStringExtra(TelephonyManager.EXTRA_INCOMING_NUMBER)).
 - Nếu cuộc gọi bị nhỡ (có thể theo dõi thêm sự kiện TelephonyManager.CALL_STATE_IDLE sau khi đổ chuông), gửi tin nhắn SMS đến số điện thoại đó.

3. Gửi tin nhắn SMS:

- Sử dụng SmsManager để gửi tin nhắn SMS.
- SmsManager.getDefault().sendTextMessage() để gửi tin nhắn.

4. Đăng ký Broadcast Receiver:

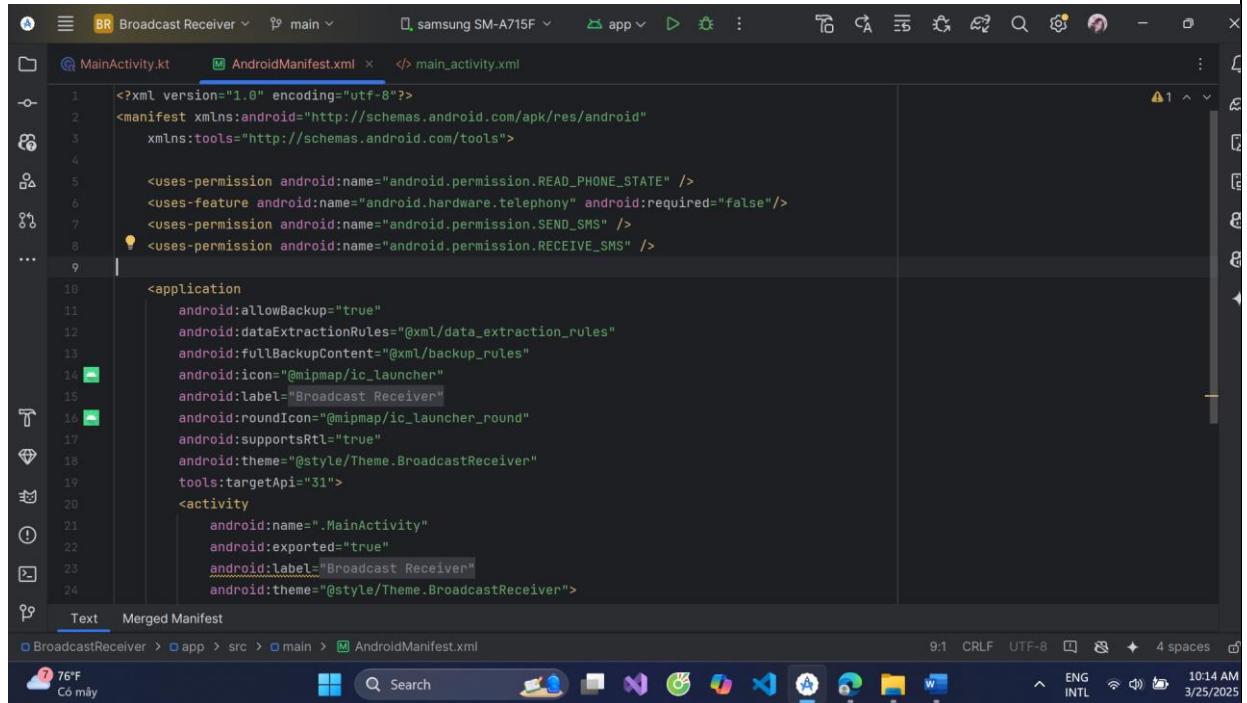
- Đăng ký receiver trong AndroidManifest.xml.

Hướng dẫn Bài tập 02: Chụp lại mà hình từng bước thực hiện (tương tự cho các Bài tập bên dưới) để học sinh lớp 10 có thể thực hiện lại theo được :D

1. Khai báo quyền:

- Thêm các quyền cần thiết vào AndroidManifest.xml:
 - android.permission.READ_PHONE_STATE (để theo dõi trạng thái cuộc gọi)
 - android.permission.SEND_SMS (để gửi tin nhắn SMS)
 - android.permission.RECEIVE_SMS (nếu muốn xử lý cả tin nhắn đến)

- Thêm dòng: <uses-feature android:name="android.hardware.telephony" android:required="false"/> để đảm bảo thiết bị không có phần cứng đó thì vẫn dùng được ứng dụng bởi không phải thiết bị nào cũng có loại phần cứng đó



```

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

    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-feature android:name="android.hardware.telephony" android:required="false"/>
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Broadcast Receiver"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.BroadcastReceiver"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="Broadcast Receiver"
            android:theme="@style/Theme.BroadcastReceiver">
    
```

1. Tạo Broadcast Receiver:

- Tạo một class kế thừa BroadcastReceiver để lắng nghe sự kiện android.intent.action.PHONE_STATE.

```
BR Broadcast Receiver ✓ main ✓ samsung SM-A715F ✓ app ✓ PhoneReceiver.kt ✘ main_activity.xml

>MainActivity.kt
AndroidManifest.xml
PhoneReceiver.kt
main_activity.xml

package com.example.broadcastreceiver

import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.telephony.TelephonyManager
import android.widget.Toast

class PhoneReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        if (intent?.action == TelephonyManager.ACTION_PHONE_STATE_CHANGED) {
            val state = intent.getStringExtra(TelephonyManager.EXTRA_STATE)

            when (state) {
                TelephonyManager.EXTRA_STATE_RINGING -> {
                    Toast.makeText(context, text: "来电", duration: Toast.LENGTH_SHORT).show()
                }
                TelephonyManager.EXTRA_STATE_OFFHOOK -> {
                    Toast.makeText(context, text: "通话中", duration: Toast.LENGTH_SHORT).show()
                }
                TelephonyManager.EXTRA_STATE_IDLE -> {
                    Toast.makeText(context, text: "挂断", duration: Toast.LENGTH_SHORT).show()
                }
            }
        }
    }
}

BroadcastReceiver > app > src > main > java > com > example > broadcastreceiver > PhoneReceiver > onReceive
19:39 CRLF UTF-8 4 spaces ENG INTEL 10:19 AM 3/25/2025
```

- Trong phương thức `onReceive()`:

- Kiểm tra trạng thái cuộc gọi (TelephonyManager.EXTRA_STATE_RINGING).
 - Nếu là cuộc gọi đến, lấy số điện thoại người gọi (intent.getStringExtra(TelephonyManager.EXTRA_INCOMING_NUMBER)).
 - Nếu cuộc gọi bị nhỡ (có thể theo dõi thêm sự kiện TelephonyManager.CALL_STATE_IDLE sau khi đổ chuông), gửi tin nhắn SMS đến số điện thoại đó.

The screenshot shows the Android Studio interface with the following details:

- Top Bar:** Shows "Broadcast Receiver" as the current project, "main" as the module, "samsung SM-A715F" as the device, and various icons for file operations.
- Side Navigation:** Displays the file structure with "MainActivity.kt", "AndroidManifest.xml", and "PhoneReceiver.kt" as the active file.
- Code Editor:** The "PhoneReceiver.kt" file contains Java code for a broadcast receiver. It imports necessary classes like BroadcastReceiver, Context, Intent, SmsManager, TelephonyManager, and Toast. The class "PhoneStateReceiver" extends BroadcastReceiver and overrides the onReceive method. Inside onReceive, it checks if the action is ACTION_PHONE_STATE_CHANGED. If true, it retrieves the state (EXTRA_STATE) and then uses a when block to handle two cases: EXTRA_STATE_RINGING (displaying the incoming number) and EXTRA_STATE_IDLE (displaying a message about the call ending).

```
1 package com.example.broadcastreceiver
2
3 import android.content.BroadcastReceiver
4 import android.content.Context
5 import android.content.Intent
6 import android.telephony.SmsManager
7 import android.telephony.TelephonyManager
8 import android.widget.Toast
9
10 class PhoneStateReceiver : BroadcastReceiver() {
11
12     private var incomingNumber: String? = null // Lưu số điện thoại gọi đến
13
14     override fun onReceive(context: Context?, intent: Intent?) {
15         if (intent?.action == TelephonyManager.ACTION_PHONE_STATE_CHANGED) {
16             val state = intent.getStringExtra(TelephonyManager.EXTRA_STATE)
17
18             when (state) {
19                 // Khi có cuộc gọi đến
20                 TelephonyManager.EXTRA_STATE_RINGING -> {
21                     incomingNumber = intent.getStringExtra(TelephonyManager.EXTRA_INCOMING_NUMBER)
22                     Toast.makeText(context, text = "来电号码: $incomingNumber", duration = Toast.LENGTH_SHORT).show()
23                 }
24
25                 // Khi cuộc gọi kết thúc
26                 TelephonyManager.EXTRA_STATE_IDLE -> {
27
28             }
29         }
30     }
31 }
```
- Bottom Status Bar:** Shows the current temperature (76°F), battery status (Có mây), system icons (Windows, Search, Task View, File Explorer, Paint, Photos, Snipping Tool, Task Manager, PowerShell, Visual Studio Code, Edge, File Explorer, Word), and system status (ENG INTL, 1022 AM, 3/25/2025).

```
BR Broadcast Receiver ✓ main ✓ Samsung SM-A715F ✓ app ✓ ... To CA Es E S Q G 12 9 ...
MainActivity.kt AndroidManifest.xml PhoneReceiver.kt main_activity.xml
10 class PhoneStateReceiver : BroadcastReceiver() {
11     override fun onReceive(context: Context?, intent: Intent?) {
12         ...
13         ...
14         ...
15         ...
16         ...
17         ...
18         ...
19         ...
20         ...
21         ...
22         ...
23         ...
24         ...
25         ...
26         ...
27         ...
28         ...
29         ...
30         ...
31         ...
32         ...
33         ...
34     }
35
36     // Gửi tin nhắn SMS đến số điện thoại gọi nhỡ
37     private fun sendMissedCallSMS(context: Context?, phoneNumber: String) {
38         try {
39             val smsManager = SmsManager.getDefault()
40             val message = "Xin chào, tôi đã bỏ lỡ cuộc gọi của bạn. Vui lòng gọi lại sau nhé!"
41             smsManager.sendTextMessage(phoneNumber, null, message, null, null)
42             Toast.makeText(context, text: "Đã gửi SMS đến: $phoneNumber", Toast.LENGTH_SHORT).show()
43         } catch (e: Exception) {
44             Toast.makeText(context, text: "Gửi SMS thất bại!", Toast.LENGTH_SHORT).show()
45         }
46     }
47 }
```

BroadcastReceiver > app > src > main > java > com > example > broadcastreceiver > PhoneReceiver.kt

6:36 CRLF UTF-8 ENG INTL 10:22 AM 3/25/2025

2. Gửi tin nhắn SMS:

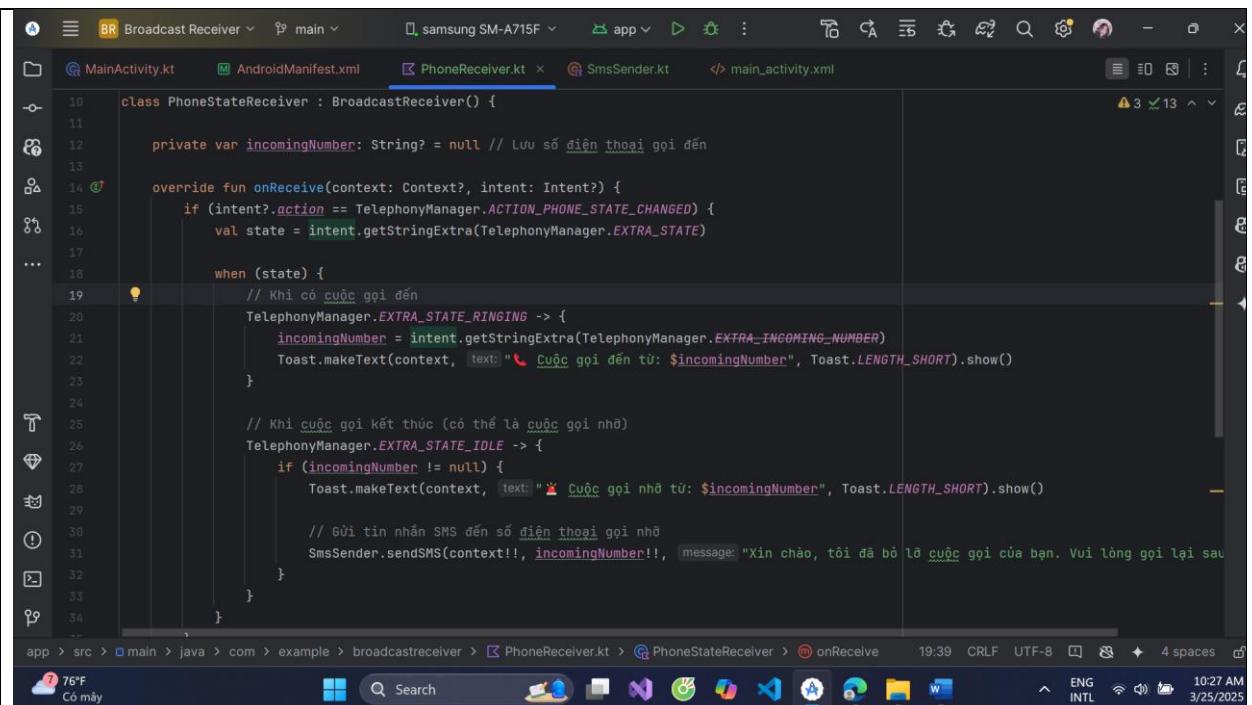
- o Sử dụng SmsManager để gửi tin nhắn SMS.
- o SmsManager.getDefault().sendTextMessage() để gửi tin nhắn.

```
BR Broadcast Receiver ✓ main ✓ Samsung SM-A715F ✓ app ✓ ... To CA Es E S Q G 12 3 ...
MainActivity.kt AndroidManifest.xml SmsSender.kt main_activity.xml
1 package com.example.broadcastreceiver
2
3 import android.content.Context
4 import android.telephony.SmsManager
5 import android.widget.Toast
6
7 object SmsSender {
8     fun sendsSMS(context: Context, phoneNumber: String, message: String) {
9         try {
10             val smsManager = SmsManager.getDefault()
11             smsManager.sendTextMessage(phoneNumber, null, message, null, null)
12             Toast.makeText(context, text: "Đã gửi SMS đến: $phoneNumber", Toast.LENGTH_SHORT).show()
13         } catch (e: Exception) {
14             Toast.makeText(context, text: "Gửi SMS thất bại!", Toast.LENGTH_SHORT).show()
15         }
16     }
17 }
```

BroadcastReceiver > app > src > main > java > com > example > broadcastreceiver > SmsSender.kt

6:1 CRLF UTF-8 ENG INTL 10:25 AM 3/25/2025

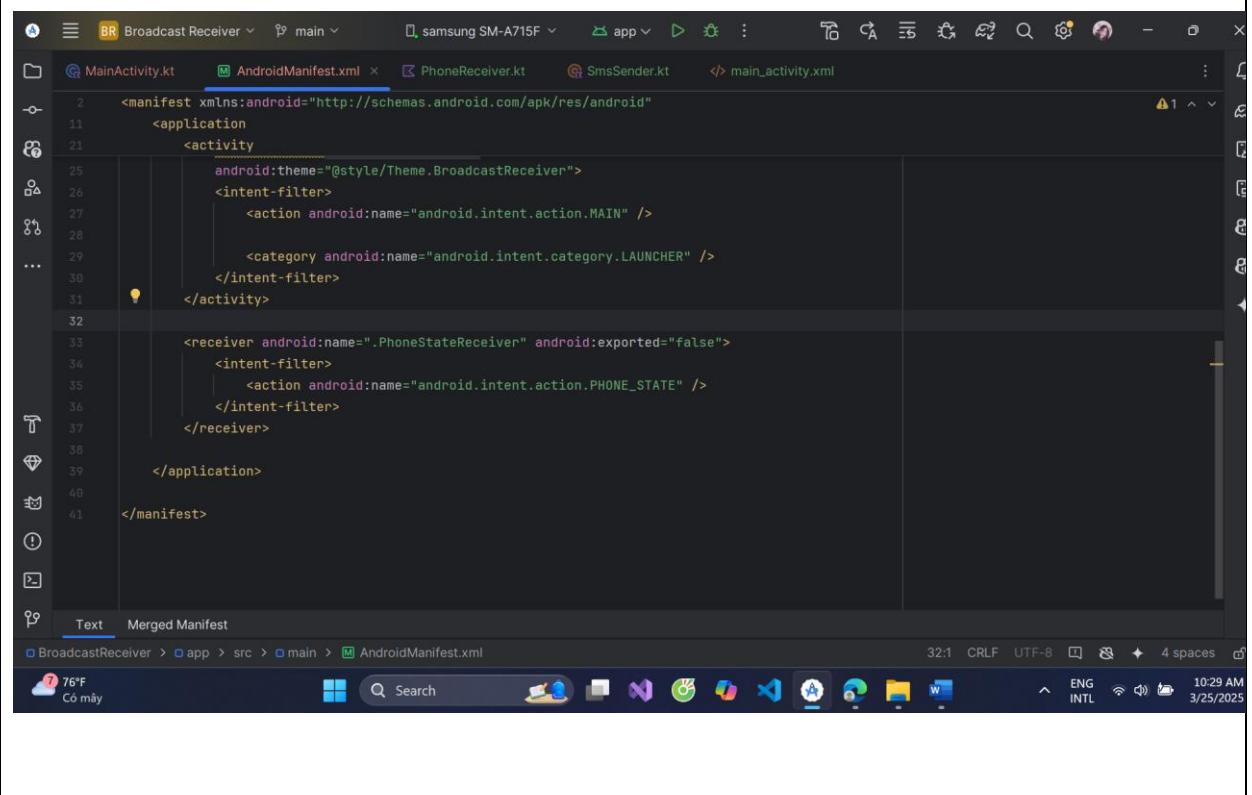
- File PhoneReceiver cập nhật để phát hiện cuộc gọi nhỡ sau khi có SmsSender



```
10 class PhoneStateReceiver : BroadcastReceiver() {
11
12     private var incomingNumber: String? = null // Lưu số điện thoại gọi đến
13
14     override fun onReceive(context: Context?, intent: Intent?) {
15         if (intent?.action == TelephonyManager.ACTION_PHONE_STATE_CHANGED) {
16             val state = intent.getStringExtra(TelephonyManager.EXTRA_STATE)
17
18             when (state) {
19                 // Khi có cuộc gọi đến
20                 TelephonyManager.EXTRA_STATE_RINGING -> {
21                     incomingNumber = intent.getStringExtra(TelephonyManager.EXTRA_INCOMING_NUMBER)
22                     Toast.makeText(context, "来电号码: $incomingNumber", Toast.LENGTH_SHORT).show()
23                 }
24
25                 // Khi cuộc gọi kết thúc (có thể là cuộc gọi nhỡ)
26                 TelephonyManager.EXTRA_STATE_IDLE -> {
27                     if (incomingNumber != null) {
28                         Toast.makeText(context, "已挂断: $incomingNumber", Toast.LENGTH_SHORT).show()
29
30                         // Gửi tin nhắn SMS đến số điện thoại gọi nhỡ
31                         SmsSender.sendSMS(context!!, incomingNumber!!, "Xin chào, tôi đã bỏ lỡ cuộc gọi của bạn. Vui lòng gọi lại sau")
32                     }
33                 }
34             }
35         }
36     }
37 }
38
39
40
41 }
```

3. Đăng ký Broadcast Receiver:

- Đăng ký receiver trong AndroidManifest.xml.



```
1 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2     <application
3         <activity
4             android:theme="@style/Theme.BroadcastReceiver"
5             <intent-filter>
6                 <action android:name="android.intent.action.MAIN" />
7
8                 <category android:name="android.intent.category.LAUNCHER" />
9             </intent-filter>
10            </activity>
11
12            <receiver android:name=".PhoneStateReceiver" android:exported="false">
13                <intent-filter>
14                    <action android:name="android.intent.action.PHONE_STATE" />
15                </intent-filter>
16            </receiver>
17
18        </application>
19
20    </manifest>
```

BÀI TẬP 3: Ứng dụng chặn cuộc gọi theo số điện thoại

Mục tiêu:

- Sử dụng Broadcast Receiver để lắng nghe sự kiện cuộc gọi đến.

- Sử dụng Telephony API để lấy thông tin về cuộc gọi (số điện thoại).
- (Nâng cao) Tìm hiểu cách chặn cuộc gọi (có thể cần các phương pháp không chính thức hoặc API riêng của nhà sản xuất điện thoại).

Mô tả:

Ứng dụng sẽ tự động từ chối hoặc ngắt kết nối các cuộc gọi đến từ một danh sách các số điện thoại bị chặn.

Các bước thực hiện:

1. Khai báo quyền:

- Thêm quyền android.permission.READ_PHONE_STATE vào AndroidManifest.xml.
- (Có thể cần thêm các quyền liên quan đến xử lý cuộc gọi tùy theo phương pháp chặn)

2. Tạo Broadcast Receiver:

- Tạo một class kế thừa BroadcastReceiver để lắng nghe sự kiện android.intent.action.PHONE_STATE.
- Trong phương thức onReceive():
 - Kiểm tra trạng thái cuộc gọi (TelephonyManager.EXTRA_STATE_RINGING).
 - Nếu là cuộc gọi đến, lấy số điện thoại người gọi.
 - Kiểm tra xem số điện thoại đó có nằm trong danh sách chặn hay không.
 - Nếu có trong danh sách chặn, thực hiện hành động chặn cuộc gọi.

3. Chặn cuộc gọi:

- (Phần này có thể phức tạp và phụ thuộc vào phiên bản Android và nhà sản xuất điện thoại)
- Có thể cần sử dụng TelephonyManager hoặc các API khác để thực hiện chặn cuộc gọi.
- Lưu ý rằng việc chặn cuộc gọi có thể bị hạn chế hoặc không được hỗ trợ trên một số thiết bị.

4. Đăng ký Broadcast Receiver:

- Đăng ký receiver trong AndroidManifest.xml.

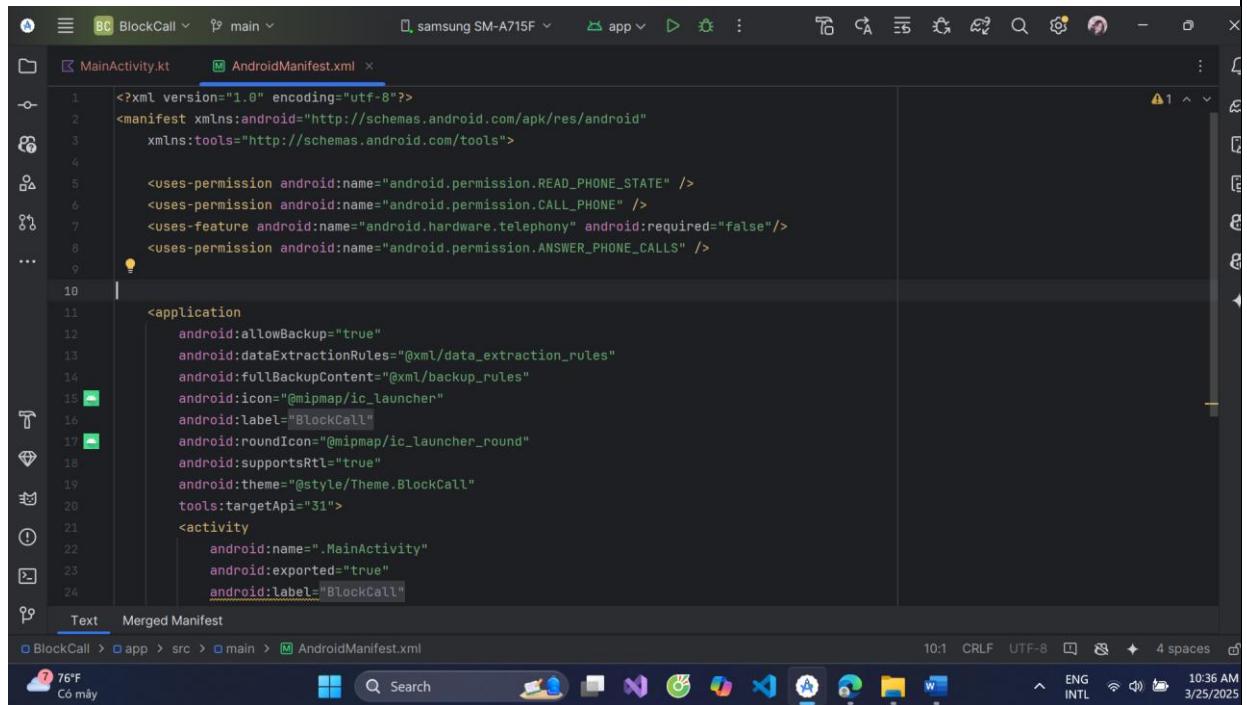
Lưu ý quan trọng:

- **Xử lý bất đồng bộ trong onReceive():** Tránh thực hiện các tác vụ tốn thời gian trong phương thức onReceive() của Broadcast Receiver. Nếu cần thực hiện các tác vụ dài, hãy sử dụng Service.
- **Quyền (Permissions):** Các thao tác liên quan đến Telephony và SMS đều yêu cầu các quyền đặc biệt. Đảm bảo bạn đã khai báo đầy đủ các quyền trong AndroidManifest.xml và xử lý việc xin quyền từ người dùng một cách thích hợp (đặc biệt là trên các phiên bản Android mới).
- **Hạn chế của Telephony API:** Một số chức năng liên quan đến Telephony có thể bị hạn chế hoặc không được hỗ trợ trên một số thiết bị hoặc phiên bản Android.
- **SMS PDU:** Khi nhận SMS, dữ liệu thường ở định dạng PDU. Cần xử lý để giải mã và đọc nội dung tin nhắn.

Hướng dẫn bài tập 3:

1. Khai báo quyền:

- Thêm quyền android.permission.READ_PHONE_STATE vào AndroidManifest.xml.



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

    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-feature android:name="android.hardware.telephony" android:required="false"/>
    <uses-permission android:name="android.permission.ANSWER_PHONE_CALLS" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="BlockCall"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.BlockCall"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="BlockCall">
    
```

- (Có thể cần thêm các quyền liên quan đến xử lý cuộc gọi tùy theo phương pháp chặn)

2. Tạo Broadcast Receiver:

- Tạo một class kế thừa BroadcastReceiver để lắng nghe sự kiện android.intent.action.PHONE_STATE.
- Trong phương thức onReceive():
 - Kiểm tra trạng thái cuộc gọi (TelephonyManager.EXTRA_STATE_RINGING).
 - Nếu là cuộc gọi đến, lấy số điện thoại người gọi.
 - Kiểm tra xem số điện thoại đó có nằm trong danh sách chặn hay không.
 - Nếu có trong danh sách chặn, thực hiện hành động chặn cuộc gọi.

```
1 package com.example.blockcall
2
3 import android.content.BroadcastReceiver
4 import android.content.Context
5 import android.content.Intent
6 import android.telephony.TelephonyManager
7 import android.util.Log
8
9 class PhoneStateReceiver : BroadcastReceiver() {
10     override fun onReceive(context: Context?, intent: Intent?) {
11         if (intent?.action == TelephonyManager.ACTION_PHONE_STATE_CHANGED) {
12             val state = intent.getStringExtra(TelephonyManager.EXTRA_STATE)
13             val incomingNumber = intent.getStringExtra(TelephonyManager.EXTRA_INCOMING_NUMBER)
14
15             if (state == TelephonyManager.EXTRA_STATE_RINGING && incomingNumber != null) {
16                 Log.d(tag: "PhoneStateReceiver", msg: "Cuộc gọi đến từ: $incomingNumber")
17
18                 // Kiểm tra số điện thoại có trong danh sách chặn không
19                 if (isBlockedNumber(incomingNumber, context!!)) {
20                     Log.d(tag: "PhoneStateReceiver", msg: "Chặn cuộc gọi từ: $incomingNumber")
21                     endCall(context)
22                 }
23             }
24         }
25     }
26 }
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
```

3. Chặn cuộc gọi:

- (Phần này có thể phức tạp và phụ thuộc vào phiên bản Android và nhà sản xuất điện thoại)
- Có thể cần sử dụng TelephonyManager hoặc các API khác để thực hiện chặn cuộc gọi.
- Lưu ý rằng việc chặn cuộc gọi có thể bị hạn chế hoặc không được hỗ trợ trên một số thiết bị.

```
1 package com.example.blockcall
2
3 import android.content.BroadcastReceiver
4 import android.content.Context
5 import android.content.Intent
6 import android.telephony.TelephonyManager
7 import android.util.Log
8
9 class PhoneStateReceiver : BroadcastReceiver() {
10     private fun isBlockedNumber(number: String, context: Context): Boolean {
11         // Giả lập danh sách chặn, bạn có thể thay bằng dữ liệu từ SharedPreferences hoặc Database
12         val blockedNumbers = listOf("+84901234567", "+84876543210")
13         return blockedNumbers.contains(number)
14     }
15
16     private fun endCall(context: Context) {
17         try {
18             val telephonyManager = context.getSystemService(Context.TELEPHONY_SERVICE) as TelephonyManager
19             val clazz = Class.forName(telephonyManager.javaClass.name)
20             val method = clazz.getDeclaredMethod(name: "getITelephony")
21             method.isAccessible = true
22             val telephonyInterface = method.invoke(telephonyManager)
23             val endCallMethod = telephonyInterface.javaClass.getMethod(name: "endCall")
24             endCallMethod.invoke(telephonyInterface)
25         } catch (e: Exception) {
26             Log.e(tag: "PhoneStateReceiver", msg: "Không thể chặn cuộc gọi: ${e.message}")
27         }
28     }
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
```

4. Đăng ký Broadcast Receiver:

- Đăng ký receiver trong AndroidManifest.xml.

The screenshot shows the Android Studio interface with the code editor open to the `AndroidManifest.xml` file. The manifest defines an application with a launcher icon, a round icon, and supports RTL. It includes a receiver for phone state changes and an activity for the main application. The code is written in XML and uses annotations like `tools:targetApi`.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="BlockCall"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.BlockCall"
        tools:targetApi="31">

        <receiver android:name=".PhoneStateReceiver" android:exported="false">
            <intent-filter>
                <action android:name="android.intent.action.PHONE_STATE" />
            </intent-filter>
        </receiver>

        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="BlockCall"
            android:theme="@style/Theme.BlockCall">
            <intent-filter>

```

BÀI TẬP 4: Ứng dụng tải và hiển thị ảnh từ Internet

Mục tiêu:

- Sử dụng `AsyncTask` để thực hiện tải ảnh từ một URL trên Internet trong background.
- Hiển thị ảnh đã tải xuống lên `ImageView` trong UI Thread.
- Hiển thị progress bar trong khi tải ảnh.

Mô tả:

Ứng dụng cho phép người dùng nhập một URL ảnh. Sau khi người dùng nhấn nút, ứng dụng sẽ hiển thị một progress bar và bắt đầu tải ảnh từ URL đó trong background. Khi tải xong, ứng dụng sẽ ẩn progress bar và hiển thị ảnh lên `ImageView`.

Các bước thực hiện:

1. Thiết kế giao diện:

- Một `EditText` để người dùng nhập URL.
- Một `ImageView` để hiển thị ảnh.
- Một `ProgressBar` để hiển thị tiến trình tải.
- Một `Button` để kích hoạt quá trình tải.

2. Tạo `AsyncTask`:

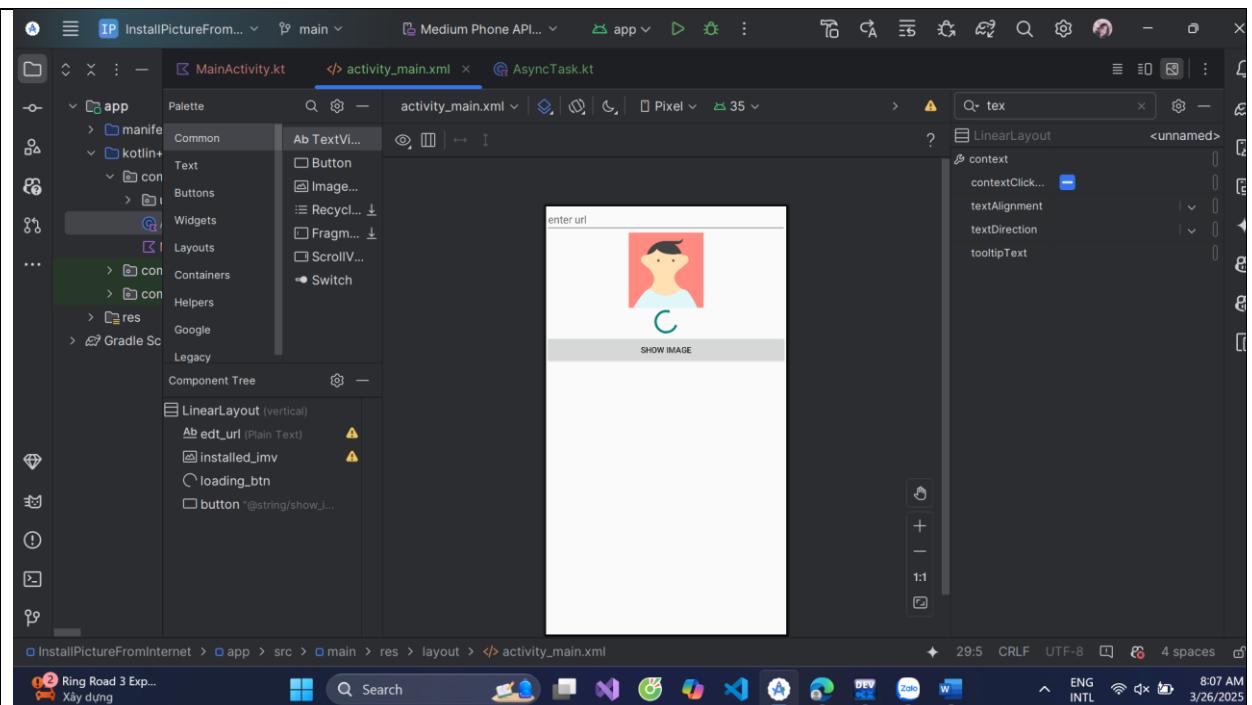
- Tạo một class kế thừa `AsyncTask<String, Integer, Bitmap>`.
 - `String`: URL của ảnh.
 - `Integer`: Tiến trình tải (ví dụ: phần trăm hoàn thành).
 - `Bitmap`: Ảnh đã tải.
- Implement các phương thức:
 - `onPreExecute()`: Hiển thị progress bar.
 - `doInBackground(String... urls)`:
 - Tải ảnh từ URL (sử dụng các thư viện như `HttpURLConnection` hoặc `OkHttp`).
 - Trong quá trình tải, gọi `publishProgress()` để cập nhật tiến trình (ví dụ: phần trăm tải).
 - Trả về `Bitmap` của ảnh đã tải.
 - `onProgressUpdate(Integer... values)`: Cập nhật progress bar trên UI thread.
 - `onPostExecute(Bitmap result)`:
 - Ẩn progress bar.
 - Hiển thị ảnh lên `ImageView` (nếu tải thành công).

3. Xử lý sự kiện Button click:

- Khi người dùng click nút, lấy URL từ `EditText`.
- Tạo một instance của `AsyncTask` và gọi `execute(url)`.

1. Thiết kế giao diện:

- Một `EditText` để người dùng nhập URL.
- Một `ImageView` để hiển thị ảnh.
- Một `ProgressBar` để hiển thị tiến trình tải.
- Một `Button` để kích hoạt quá trình tải.



2.Tạo AsyncTask:

- Tạo một class kế thừa AsyncTask<String, Integer, Bitmap>.
 - String: URL của ảnh.
 - Integer: Tiến trình tải (ví dụ: phần trăm hoàn thành).
 - Bitmap: Ảnh đã tải.

```

1 import android.graphics.BitmapFactory
2 import android.os.AsyncTask
3 import android.view.View
4 import android.widget.ImageView
5 import android.widget.ProgressBar
6 import java.io.InputStream
7 import java.net.HttpURLConnection
8 import java.net.URL
9
10
11 class DownloadImageTask(
12     private val imageView: ImageView,
13     private val progressBar: ProgressBar
14 ) : AsyncTask<String, Int, Bitmap?>() {
15
16     override fun onPreExecute() {
17         super.onPreExecute()
18         progressBar.visibility = View.VISIBLE // Hiển thị progress bar
19     }
20
21     override fun doInBackground(vararg params: String?): Bitmap? {
22         val urlString = params[0] ?: return null
23         var bitmap: Bitmap? = null
24         try {
25             val url = URL(urlString)
26             val connection = url.openConnection() as HttpURLConnection
27             val inputStream: InputStream = connection.inputStream
28             bitmap = BitmapFactory.decodeStream(inputStream)
29         } catch (e: Exception) {
30             e.printStackTrace()
31         }
32         return bitmap
33     }
34
35     override fun onPostExecute(result: Bitmap?) {
36         imageView.setImageBitmap(result)
37         progressBar.visibility = View.GONE
38     }
39 }

```

- Implement các phương thức:
 - onPreExecute(): Hiển thị progress bar.
 - doInBackground(String... urls):
 - Tải ảnh từ URL (sử dụng các thư viện như HttpURLConnection hoặc OkHttp).

- Trong quá trình tải, gọi `publishProgress()` để cập nhật tiến trình (ví dụ: phần trăm tải).
- Trả về `Bitmap` của ảnh đã tải.
- `onProgressUpdate(Integer... values)`: Cập nhật progress bar trên UI thread.
- `onPostExecute(Bitmap result)`:
 - Ẩn progress bar.
 - Hiển thị ảnh lên ImageView (nếu tải thành công).

```

11 class DownloadImageTask(
12
13     override fun doInBackground(vararg params: String?): Bitmap? {
14         val urlString = params[0] ?: return null
15         var bitmap: Bitmap? = null
16         try {
17             val url = URL(urlString)
18             val connection = url.openConnection() as HttpURLConnection
19             connection.doInput = true
20             connection.connect()
21
22             val fileLength = connection.contentLength // Lấy kích thước file
23             val inputStream: InputStream = connection.inputStream
24             val byteArray = ByteArray(fileLength)
25             var totalBytesRead = 0
26             var bytesRead: Int
27
28             while (inputStream.read(byteArray, totalBytesRead, byteArray.size - totalBytesRead)
29                   .also { bytesRead = it } != -1
30             ) {
31                 totalBytesRead += bytesRead
32                 publishProgress(...values: (totalBytesRead * 100) / fileLength) // Cập nhật tiến trình
33             }
34
35             bitmap = BitmapFactory.decodeByteArray(byteArray, 0, totalBytesRead)
36             inputStream.close()
37         } catch (e: Exception) {
38             e.printStackTrace()
39         }
40         return bitmap
41     }
42
43     override fun onProgressUpdate(vararg values: Int?) {
44         super.onProgressUpdate(*values)
45         val progress = values[0] ?: 0
46         progressBar.progress = progress // Cập nhật progress bar
47     }
48
49     override fun onPostExecute(result: Bitmap?) {
50         super.onPostExecute(result)
51         progressBar.visibility = View.GONE // Ẩn progress bar
52         if (result != null) {
53             imageView.setImageBitmap(result) // Hiển thị ảnh lên ImageView
54         }
55     }
56
57     companion object {
58         private const val TAG = "DownloadImageTask"
59     }
60 }

```

```

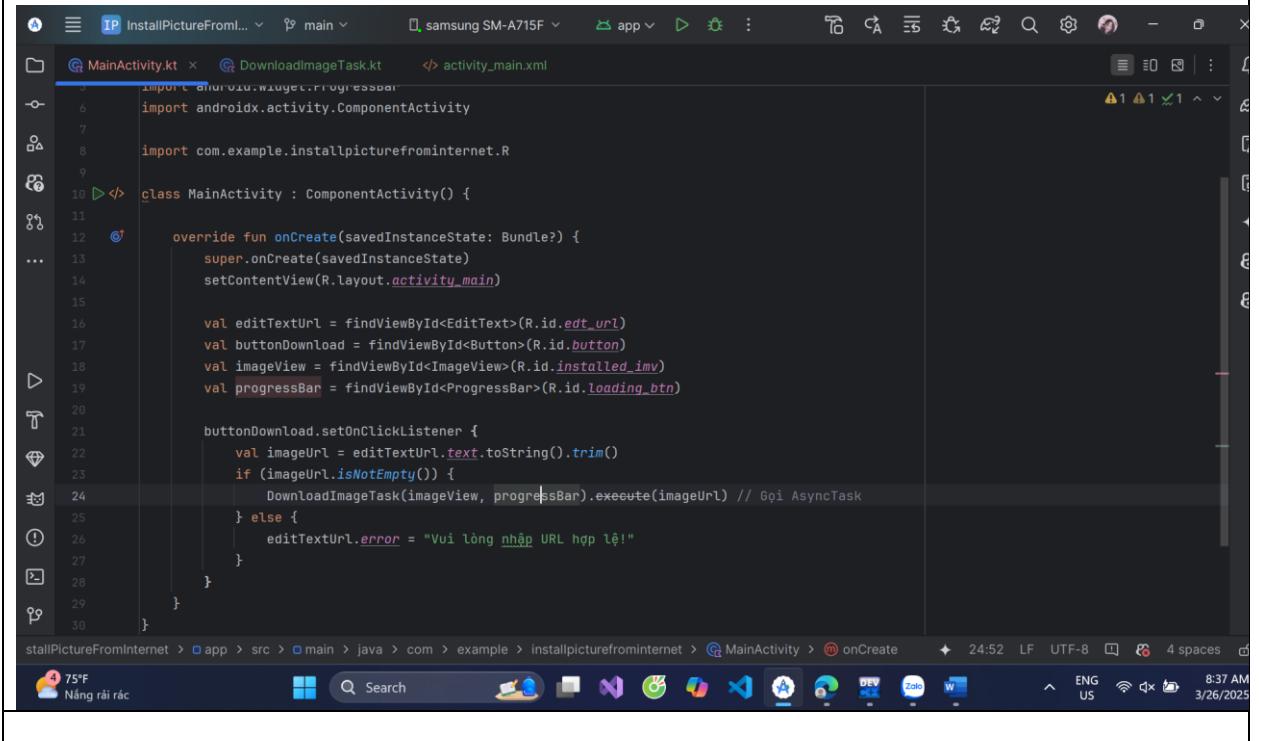
11 class DownloadImageTask(
12
13     override fun doInBackground(vararg params: String?): Bitmap? {
14
15         val url = URL(params[0] ?: return null)
16         val connection = url.openConnection() as HttpURLConnection
17         connection.doInput = true
18         connection.connect()
19
20         val fileLength = connection.contentLength // Lấy kích thước file
21         val inputStream: InputStream = connection.inputStream
22         val byteArray = ByteArray(fileLength)
23         var totalBytesRead = 0
24         var bytesRead: Int
25
26         while (inputStream.read(byteArray, totalBytesRead, byteArray.size - totalBytesRead)
27               .also { bytesRead = it } != -1
28         ) {
29             totalBytesRead += bytesRead
30             publishProgress(...values: (totalBytesRead * 100) / fileLength) // Cập nhật tiến trình
31         }
32
33         bitmap = BitmapFactory.decodeByteArray(byteArray, 0, totalBytesRead)
34         inputStream.close()
35     } catch (e: Exception) {
36         e.printStackTrace()
37     }
38
39     return bitmap
40 }
41
42
43     override fun onProgressUpdate(vararg values: Int?) {
44         super.onProgressUpdate(*values)
45         val progress = values[0] ?: 0
46         progressBar.progress = progress // Cập nhật progress bar
47     }
48
49     override fun onPostExecute(result: Bitmap?) {
50         super.onPostExecute(result)
51         progressBar.visibility = View.GONE // Ẩn progress bar
52         if (result != null) {
53             imageView.setImageBitmap(result) // Hiển thị ảnh lên ImageView
54         }
55     }
56
57     companion object {
58         private const val TAG = "DownloadImageTask"
59     }
60 }

```

3.Xử lý sự kiện Button click:

- Khi người dùng click nút, lấy URL từ EditText.

- Tạo một instance của AsyncTask và gọi execute(url).



The screenshot shows the Android Studio interface with the code editor open. The file is `MainActivity.kt`. The code implements an `onCreate` method that finds views by ID and sets up a click listener for a button. The button's click listener checks if an URL is entered in an edit text field. If the URL is not empty, it creates an instance of `DownloadImageTask` and calls its `execute` method with the URL. If the URL is empty, it sets an error message in the edit text field.

```
import android.widget.ProgressBar
import androidx.appcompat.app.AppCompatActivity
import com.example.installpicturefrominternet.R
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val editTextUrl = findViewById<EditText>(R.id.edt_url)
        val buttonDownload = findViewById<Button>(R.id.button)
        val imageView = findViewById<ImageView>(R.id.installed_imv)
        val progressBar = findViewById<ProgressBar>(R.id.loading_btn)

        buttonDownload.setOnClickListener {
            val imageUrl = editTextUrl.text.toString().trim()
            if (imageUrl.isNotEmpty()) {
                DownloadImageTask(imageView, progressBar).execute(imageUrl) // Gọi AsyncTask
            } else {
                editTextUrl.error = "Vui lòng nhập URL hợp lệ!"
            }
        }
    }
}
```

BÀI TẬP 5: Ứng dụng đếm giờ và cập nhật giao diện

Mục tiêu:

- Sử dụng Handler và Runnable để cập nhật UI định kỳ từ một thread khác.
- Hiển thị thời gian đã trôi qua trên TextView.

Mô tả:

Ứng dụng hiển thị một TextView để hiển thị thời gian đã trôi qua (ví dụ: số giây). Một thread nền sẽ tăng giá trị thời gian và sử dụng Handler để gửi thông tin cập nhật lên UI thread để hiển thị.

Các bước thực hiện:

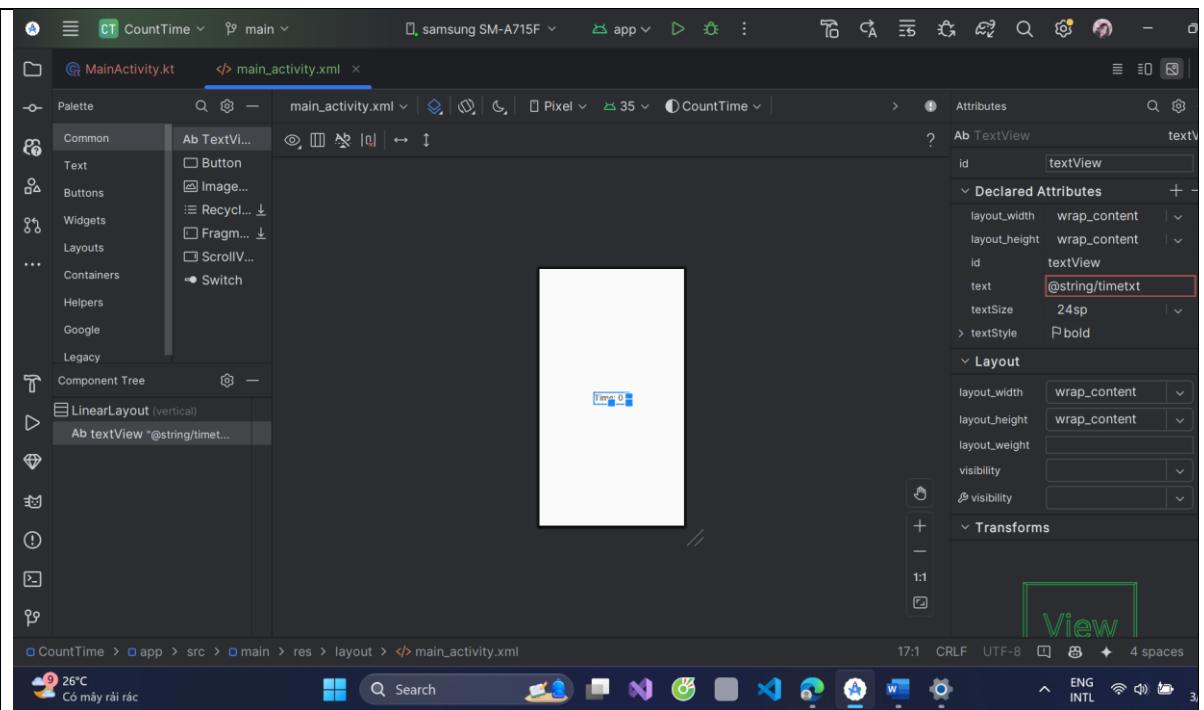
1. **Thiết kế giao diện:**
 - Một TextView để hiển thị thời gian.
2. **Tạo Handler:**
 - Tạo một Handler trong Activity chính.
 - Override phương thức `handleMessage()` (nếu dùng Message) hoặc tạo một Runnable.
3. **Tạo Thread:**
 - Tạo một Thread mới.
 - Trong `run()` method:
 - Lặp lại việc tăng giá trị thời gian.
 - Sử dụng `Handler.post(runnable)` để gửi một Runnable lên UI thread để cập nhật TextView.
4. **Cập nhật TextView:**
 - Trong Runnable, cập nhật `TextView.setText()` với giá trị thời gian hiện tại.

Lưu ý:

- **UI Thread:** Chỉ có UI thread mới được phép trực tiếp cập nhật các thành phần giao diện người dùng.
- **Tránh các tác vụ dài trên UI Thread:** Các tác vụ tốn thời gian (ví dụ: tải dữ liệu mạng, xử lý ảnh) nên được thực hiện trong background thread để tránh làm treo ứng dụng.
- **Sử dụng AsyncTask cho các tác vụ đơn giản:** AsyncTask là một cách dễ dàng để thực hiện các tác vụ nền đơn giản và tương tác với UI thread.
- **Sử dụng Handler và Thread cho các tác vụ phức tạp hơn:** Handler và Thread cung cấp sự linh hoạt cao hơn cho các tác vụ nền phức tạp hoặc cần kiểm soát thread chi tiết hơn.

1.Thiết kế giao diện:

- Một TextView để hiển thị thời gian.



2. Tạo Handler:

- Tạo một Handler trong Activity chính.
- Override phương thức handleMessage() (nếu dùng Message) hoặc tạo một Runnable.

```

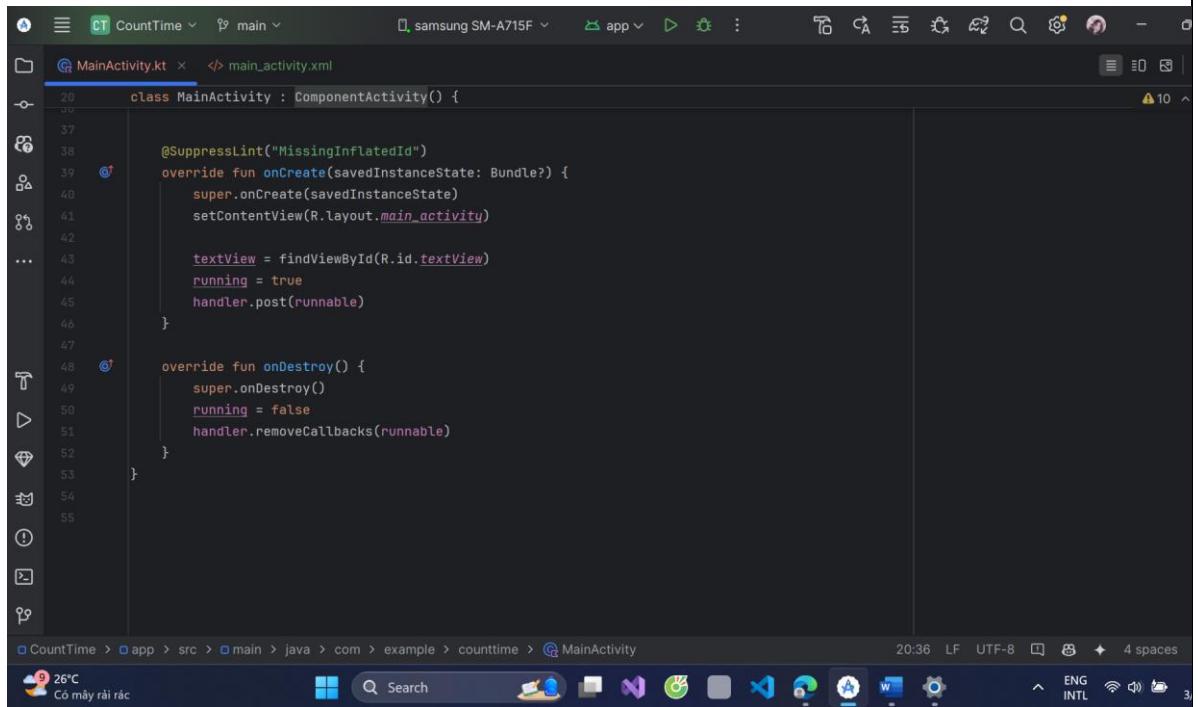
1 package com.example.counttime
2
3 import ...
4
5 class MainActivity : ComponentActivity() {
6     private lateinit var textView: TextView
7     private var seconds = 0
8     private var running = false
9     private val handler = Handler(Looper.getMainLooper())
10
11     private val runnable = object : Runnable {
12         @SuppressLint("SetTextI18n")
13         override fun run() {
14             if (running) {
15                 seconds++
16                 textView.text = "Time: $seconds s"
17                 handler.postDelayed(this, delayMillis: 1000)
18             }
19         }
20     }
21
22     @SuppressLint("MissingInflatedId")
23     override fun onCreate(savedInstanceState: Bundle?) {
24         super.onCreate(savedInstanceState)
25         setContentView(R.layout.main_activity)
26     }
27 }

```

3.Tạo Thread:

- Tạo một Thread mới.
- Trong run() method:
 - Lặp lại việc tăng giá trị thời gian.
 -

- Sử dụng Handler.post(runnable) để gửi một Runnable lên UI thread để cập nhật TextView.



```

class MainActivity : ComponentActivity() {

    @SuppressLint("MissingInflatedId")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main_activity)

        textView = findViewById(R.id.textView)
        running = true
        handler.post(runnable)
    }

    override fun onDestroy() {
        super.onDestroy()
        running = false
        handler.removeCallbacks(runnable)
    }
}

```

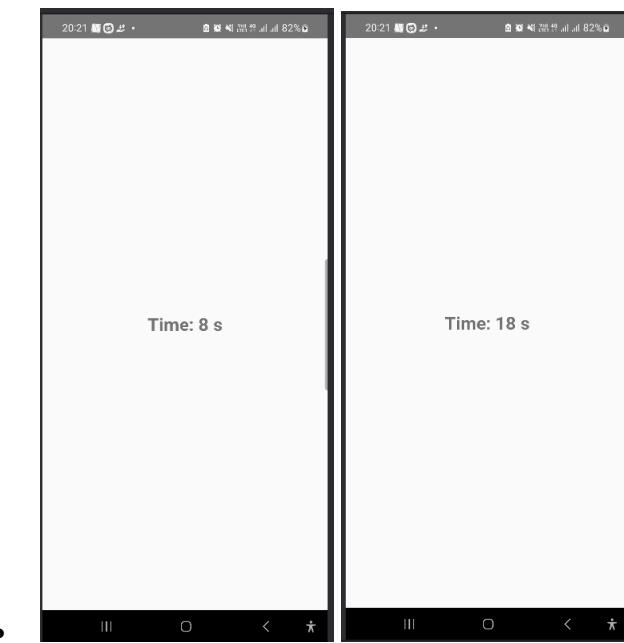
4.Cập nhật TextView:

- Trong Runnable, cập nhật TextView.setText() với giá trị thời gian hiện tại.

```

private val runnable = object : Runnable {
    @SuppressLint("SetTextI18n")
    override fun run() {
        if (running) {
            seconds++
            textView.text = "Time: $seconds s"
            handler.postDelayed(this, delayMillis: 1000)
        }
    }
}

```



BÀI TẬP 6: Ứng dụng ghi âm và phát lại

Mục tiêu:

- Sử dụng `MediaRecorder` để ghi âm từ microphone của thiết bị.
- Lưu file ghi âm vào `MediaStore` để các ứng dụng khác có thể truy cập.
- Sử dụng `MediaPlayer` để phát lại file ghi âm.
- Hiển thị danh sách các file ghi âm đã lưu trong `MediaStore`.

Mô tả:

Ứng dụng cho phép người dùng ghi âm, xem danh sách các bản ghi đã thực hiện và phát lại chúng.

Các bước thực hiện:

1. Ghi âm:

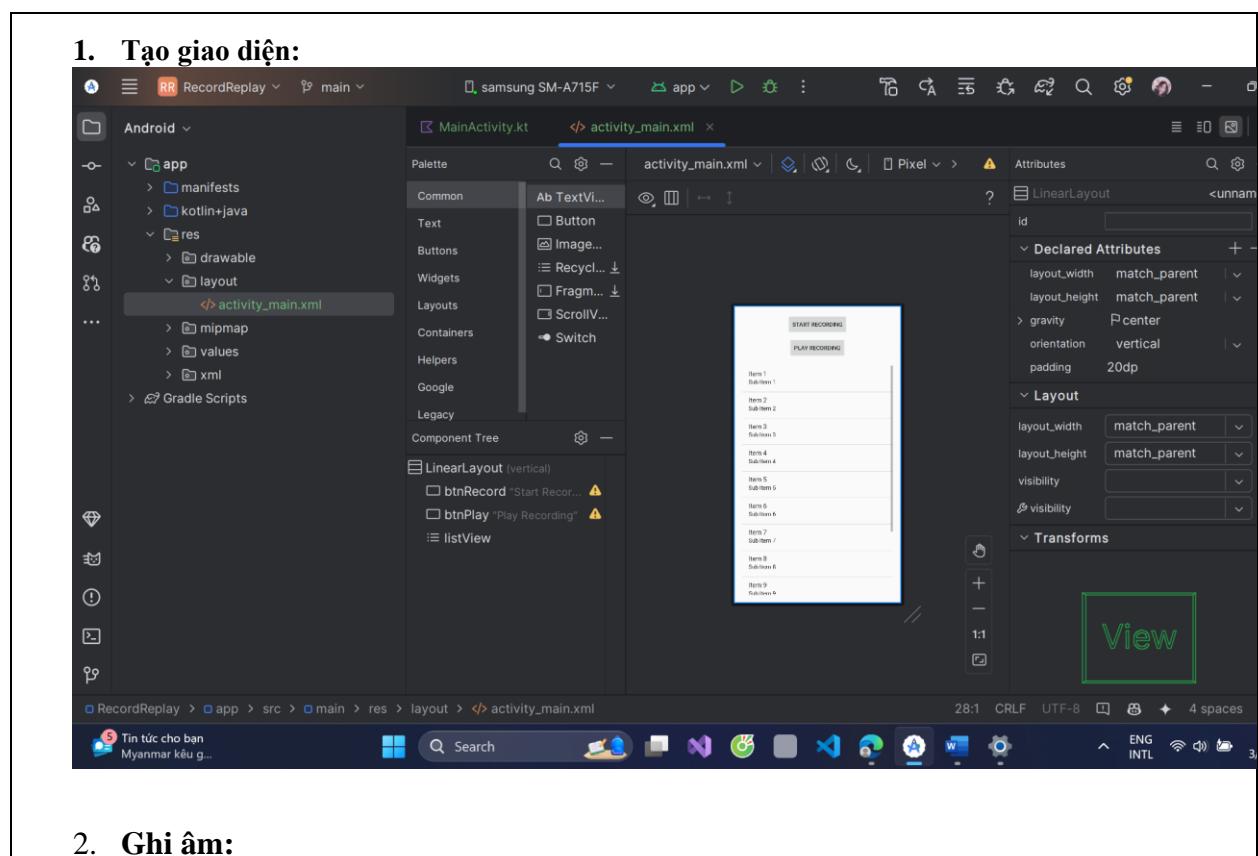
- Sử dụng `MediaRecorder` để ghi âm.
- Thiết lập các thông số cần thiết như nguồn âm thanh, định dạng file, nơi lưu trữ.
- Lưu file ghi âm vào một vị trí cụ thể.
- Sử dụng `ContentValues` để thêm thông tin về file ghi âm vào `MediaStore`.

2. Phát lại:

- Sử dụng `MediaPlayer` để phát lại file ghi âm.
- Có thể phát từ file hoặc từ một URI trong `MediaStore`.

3. Hiển thị danh sách file ghi âm:

- Sử dụng `ContentResolver` để truy vấn `MediaStore` và lấy danh sách các file âm thanh.
- Hiển thị danh sách này lên giao diện người dùng (ví dụ: `ListView`).



2. Ghi âm:

- a. Sử dụng MediaRecorder để ghi âm.
- b. Thiết lập các thông số cần thiết như nguồn âm thanh, định dạng file, nơi lưu trữ.
- c. Lưu file ghi âm vào một vị trí cụ thể.
- d. Sử dụng ContentValues để thêm thông tin về file ghi âm vào MediaStore.

```

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

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

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="RecordReplay"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.RecordReplay"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="RecordReplay"
            android:theme="@style/Theme.RecordReplay">
            <intent-filter>
        
```

- Thêm quyền vào AndroidManifest để xin quyền ghi âm và lưu file:

3. Phát lại:

- a. Sử dụng MediaPlayer để phát lại file ghi âm.
- b. Có thể phát từ file hoặc từ một URI trong MediaStore.

4. Hiển thị danh sách file ghi âm:

- a. Sử dụng ContentResolver để truy vấn MediaStore và lấy danh sách các file âm thanh.
- b. Hiển thị danh sách này lên giao diện người dùng (ví dụ: ListView).

Screenshot of Android Studio showing the MainActivity.kt file. The code initializes MediaRecorder and MediaPlayer, sets up button listeners, and handles permissions.

```
7 import android.media.MediaRecorder
8 import android.net.Uri
9 import android.os.Bundle
10 import android.provider.MediaStore
11 import android.widget.ArrayAdapter
12 import android.widget.Button
13 import android.widget.ListView
14 import android.widget.Toast
15 import androidx.activity.ComponentActivity
16 import java.io.IOException
17
18 class MainActivity : ComponentActivity() {
19     private var mediaRecorder: MediaRecorder? = null
20     private var mediaPlayer: MediaPlayer? = null
21     private var audioUri: Uri? = null
22     private var isRecording = false
23
24     override fun onCreate(savedInstanceState: Bundle?) {
25         super.onCreate(savedInstanceState)
26         setContentView(R.layout.activity_main)
27
28         val btnRecord = findViewById<Button>(R.id.btnRecord)
29         val btnPlay = findViewById<Button>(R.id.btnPlay)
30         val listView = findViewById<ListView>(R.id.listView)
31
32         btnRecord.setOnClickListener { toggleRecording() }
33         btnPlay.setOnClickListener { startPlaying() }
34
35         requestPermissions()
36     }
37
38     private fun requestPermissions() {
39         val permissions = arrayOf(
40             Manifest.permission.RECORD_AUDIO,
41             Manifest.permission.READ_MEDIA_AUDIO // Android 13+
42         )
43
44         if (!permissions.all { checkSelfPermission(it) == PackageManager.PERMISSION_GRANTED }) {
45             requestPermissions(permissions, requestCode: 0)
46         }
47     }
48
49     private fun toggleRecording() {
50         if (isRecording) {
51             stopRecording()
52         } else {
53             startRecording()
54         }
55     }
56
57     private fun startRecording() {
58         if (mediaRecorder == null) {
59             mediaRecorder = MediaRecorder()
60             mediaRecorder?.setAudioSource(MediaRecorder.AudioSource.MIC)
61             mediaRecorder?.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP)
62             mediaRecorder?.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB)
63             mediaRecorder?.setOutputFile(audioUri)
64             mediaRecorder?.start()
65         }
66     }
67
68     private fun stopRecording() {
69         if (mediaRecorder != null) {
70             mediaRecorder?.stop()
71             mediaRecorder?.release()
72             mediaRecorder = null
73         }
74     }
75
76     private fun startPlaying() {
77         if (mediaPlayer == null) {
78             mediaPlayer = MediaPlayer()
79             mediaPlayer?.setDataSource(audioUri)
80             mediaPlayer?.start()
81         }
82     }
83
84     private fun stopPlaying() {
85         if (mediaPlayer != null) {
86             mediaPlayer?.stop()
87             mediaPlayer?.release()
88             mediaPlayer = null
89         }
90     }
91
92     private fun releaseMediaRecorder() {
93         if (mediaRecorder != null) {
94             mediaRecorder?.release()
95             mediaRecorder = null
96         }
97     }
98
99     private fun releaseMediaPlayer() {
100        if (mediaPlayer != null) {
101            mediaPlayer?.release()
102            mediaPlayer = null
103        }
104    }
105
106    private fun releaseAll() {
107        releaseMediaRecorder()
108        releaseMediaPlayer()
109    }
110
111    private fun checkSelfPermission(permission: String): Int {
112        return ActivityCompat.checkSelfPermission(this, permission)
113    }
114
115    companion object {
116        const val REQUEST_CODE_PERMISSIONS = 0
117    }
118}
```

Screenshot of Android Studio showing the MainActivity.kt file with additional logic for recording and playing audio files.

```
18 class MainActivity : ComponentActivity() {
19     override fun onCreate(savedInstanceState: Bundle?) {
20         val listView = findViewById<ListView>(R.id.listView)
21
22         btnRecord.setOnClickListener { toggleRecording() }
23         btnPlay.setOnClickListener { startPlaying() }
24
25         requestPermissions()
26     }
27
28     private fun requestPermissions() {
29         val permissions = arrayOf(
30             Manifest.permission.RECORD_AUDIO,
31             Manifest.permission.READ_MEDIA_AUDIO // Android 13+
32         )
33
34         if (!permissions.all { checkSelfPermission(it) == PackageManager.PERMISSION_GRANTED }) {
35             requestPermissions(permissions, requestCode: 0)
36         }
37     }
38
39     private fun toggleRecording() {
40         if (isRecording) {
41             stopRecording()
42         } else {
43             startRecording()
44         }
45     }
46
47     private fun startRecording() {
48         if (mediaRecorder == null) {
49             mediaRecorder = MediaRecorder()
50             mediaRecorder?.setAudioSource(MediaRecorder.AudioSource.MIC)
51             mediaRecorder?.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP)
52             mediaRecorder?.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB)
53             mediaRecorder?.setOutputFile(audioUri)
54             mediaRecorder?.start()
55         }
56     }
57
58     private fun stopRecording() {
59         if (mediaRecorder != null) {
60             mediaRecorder?.stop()
61             mediaRecorder?.release()
62             mediaRecorder = null
63         }
64     }
65
66     private fun startPlaying() {
67         if (mediaPlayer == null) {
68             mediaPlayer = MediaPlayer()
69             mediaPlayer?.setDataSource(audioUri)
70             mediaPlayer?.start()
71         }
72     }
73
74     private fun stopPlaying() {
75         if (mediaPlayer != null) {
76             mediaPlayer?.stop()
77             mediaPlayer?.release()
78             mediaPlayer = null
79         }
80     }
81
82     private fun releaseMediaRecorder() {
83         if (mediaRecorder != null) {
84             mediaRecorder?.release()
85             mediaRecorder = null
86         }
87     }
88
89     private fun releaseMediaPlayer() {
90         if (mediaPlayer != null) {
91             mediaPlayer?.release()
92             mediaPlayer = null
93         }
94     }
95
96     private fun releaseAll() {
97         releaseMediaRecorder()
98         releaseMediaPlayer()
99     }
100
101    private fun checkSelfPermission(permission: String): Int {
102        return ActivityCompat.checkSelfPermission(this, permission)
103    }
104
105    companion object {
106        const val REQUEST_CODE_PERMISSIONS = 0
107    }
108}
```

The screenshot shows the Android Studio interface with the code editor open to the `MainActivity.kt` file. The code implements a `ComponentActivity` that handles recording audio from the microphone. It uses `ContentValues` to set the display name and MIME type for the recorded file, inserts it into the `MediaStore`, and then creates a `MediaRecorder` to start recording. The recorded audio is saved in `AMR_NB` format.

```
18 class MainActivity : ComponentActivity() {  
19     ...  
20     private fun toggleRecording() {  
21         if (isRecording) {  
22             stopRecording()  
23         } else {  
24             startRecording()  
25         }  
26     }  
27  
28     private fun startRecording() {  
29         val values = ContentValues().apply {  
30             put(MediaStore.Audio.Media.DISPLAY_NAME, "recording_${System.currentTimeMillis()}.3gp")  
31             put(MediaStore.Audio.Media.MIME_TYPE, "audio/3gpp")  
32         }  
33  
34         val resolver = contentResolver  
35         audioUri = resolver.insert(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, values)  
36  
37         try {  
38             val fileDescriptor = contentResolver.openFileDescriptor(audioUri!!, "w")?.fileDescriptor ?: return  
39             mediaRecorder = MediaRecorder().apply {  
40                 set AudioSource(MediaRecorder.AudioSource.MIC)  
41                 setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP)  
42                 setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB)  
43             }  
44         } catch (e: Exception) {  
45             e.printStackTrace()  
46         }  
47     }  
48  
49     private fun stopRecording() {  
50         mediaRecorder?.apply {  
51             stop()  
52             release()  
53         }  
54         mediaRecorder = null  
55     }  
56 }  
57  
58 }
```

The code editor shows several code completion suggestions and error markers. The bottom status bar indicates the device is a Samsung SM-A715F, the app is running, and the current time is 8:40 PM on March 28, 2025.

MainActivity.kt

```
18     class MainActivity : ComponentActivity() {
19         private fun startRecording() {
20             ...
21             private fun stopRecording() {
22                 mediaRecorder?.apply {
23                     stop()
24                     release()
25                 }
26                 mediaRecorder = null
27                 isRecording = false
28                 Toast.makeText(context, text: "Recording saved", Toast.LENGTH_SHORT).show()
29                 loadRecordings()
30             }
31
32             private fun startPlaying() {
33                 if (audioUri == null) {
34                     Toast.makeText(context, text: "No recording found", Toast.LENGTH_SHORT).show()
35                     return
36                 }
37
38                 mediaPlayer = MediaPlayer().apply {
39                     try {
40                         setDataSource(applicationContext, audioUri!!)
41                     ...
42                 }
43             }
44
45             private fun loadRecordings() {
46                 val recordings = mutableListOf<String>()
47                 val projection = arrayOf(MediaStore.Audio.Media._ID, MediaStore.Audio.Media.DISPLAY_NAME)
48                 val cursor = contentResolver.query(
49                     MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
50                     projection, selection: null, selectionArgs: null, sortOrder: MediaStore.Audio.Media.DATE_ADDED + " DESC"
51                 )
52             }
53
54         }
55     }
56 }
```

MainActivity.kt

```
18     class MainActivity : ComponentActivity() {
19         private fun startPlaying() {
20             ...
21             mediaPlayer = MediaPlayer().apply {
22                 try {
23                     setDataSource(applicationContext, audioUri!!)
24                     prepare()
25                     start()
26                     Toast.makeText(context: this@MainActivity, text: "Playing Audio", Toast.LENGTH_SHORT).show()
27                 } catch (e: IOException) {
28                     e.printStackTrace()
29                 }
30             }
31
32             private fun loadRecordings() {
33                 val recordings = mutableListOf<String>()
34                 val projection = arrayOf(MediaStore.Audio.Media._ID, MediaStore.Audio.Media.DISPLAY_NAME)
35                 val cursor = contentResolver.query(
36                     MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
37                     projection, selection: null, selectionArgs: null, sortOrder: MediaStore.Audio.Media.DATE_ADDED + " DESC"
38                 )
39             }
40
41         }
42     }
43 }
```

The screenshot shows the Android Studio IDE with the code editor open to `MainActivity.kt`. The code implements a list view for audio recordings, using `MediaStore` to query external content and an `ArrayAdapter` to display the results.

```

18     class MainActivity : ComponentActivity() {
19         private fun loadRecordings() {
20             val cursor = contentResolver.query(
21                 MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
22                 projection, selection: null, selectionArgs: null, sortOrder: MediaStore.Audio.Media.DATE_ADDED + " DESC"
23             )
24
25             cursor?.use {
26                 val nameIndex = it.getColumnIndex(MediaStore.Audio.Media.DISPLAY_NAME)
27                 while (it.moveToNext()) {
28                     recordings.add(it.getString(nameIndex))
29                 }
30             }
31         }
32
33         val adapter = ArrayAdapter(context: this, android.R.layout.simple_list_item_1, recordings)
34         findViewById<ListView>(R.id.listView).adapter = adapter
35     }

```

The bottom part of the screenshot displays three screenshots of the application running on an Android device. Each screenshot shows a list of audio files with two buttons: "START RECORDING" and "PLAY RECORDING". The files listed are:

- `recording_1743169124186.3gp.3ga`
- `ssstone.aac`
- `Ghi âm cuộc gọi 0996429462_241129_181634.m4a`
- `_samsung (3).mp3`
- `_samsung (2).mp3`
- `_samsung (1).mp3`
- `dung-lam-trai-tim-anh-dau-son-tung.mp3`
- `clickSound.mp3`
- `ui-click-43196.mp3`
- `_samsung.mp3`
- `Screen.Recording_20240416_173949_YouTube_16042024.mp3`
- `Screen.Recording_20240229_072459_YouTube_29022024.mp3`
- `Screen.Recording_20240229_072459_YouTube_29022024.mp3` (highlighted with a red box)
- `Screen.Recording_20240206_102553_Zing`

In the third screenshot, a notification bar message indicates "Recording saved" and "Playing Audio".

BÀI TẬP 7: Ứng dụng chơi video đơn giản

Mục tiêu:

- Sử dụng VideoView và MediaController để phát video.
- Cho phép người dùng chọn video từ MediaStore hoặc từ một URL.

Mô tả:

Ứng dụng cho phép người dùng xem video từ các nguồn khác nhau trên thiết bị hoặc từ Internet.

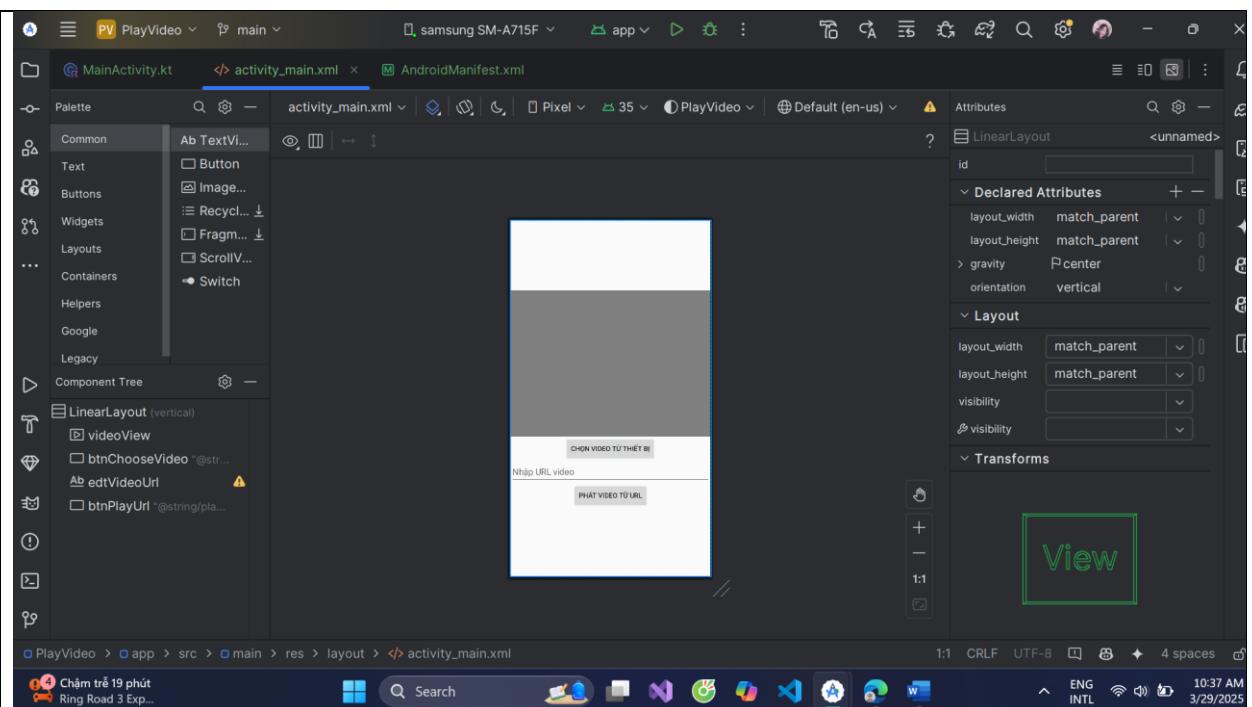
Các bước thực hiện:

1. **Thêm VideoView và MediaController vào layout.**
2. **Chọn video:**
 - Cho phép người dùng chọn video từ MediaStore bằng cách sử dụng Intent.ACTION_PICK và MediaStore.Video.Media.EXTERNAL_CONTENT_URI.
 - Hoặc cho phép người dùng nhập URL của video.
3. **Phát video:**
 - Sử dụng VideoView.setVideoURI() để thiết lập nguồn video.
 - Sử dụng MediaController để cung cấp các điều khiển phát lại video (play, pause, stop,...).
 - VideoView.start() để bắt đầu phát video.

Lưu ý:

- **Quyền (Permissions):** Đừng quên khai báo các quyền cần thiết trong AndroidManifest.xml, chẳng hạn như android.permission.RECORD_AUDIO, android.permission.READ_EXTERNAL_STORAGE, android.permission.WRITE_EXTERNAL_STORAGE, và android.permission.INTERNET.
- **Xử lý lỗi:** Cần xử lý các trường hợp lỗi có thể xảy ra, chẳng hạn như không tìm thấy file, lỗi khi ghi âm, lỗi khi phát lại, v.v.
- **Vòng đời (Lifecycle):** Quản lý các tài nguyên Media một cách chính xác trong các phương thức lifecycle của Activity/Fragment để tránh rò rỉ bộ nhớ và các vấn đề khác. Ví dụ, cần release() MediaPlayer và MediaRecorder khi không còn sử dụng.
- **MediaStore:** Làm quen với cách truy vấn và sử dụng MediaStore để lấy thông tin về các file media trên thiết bị.

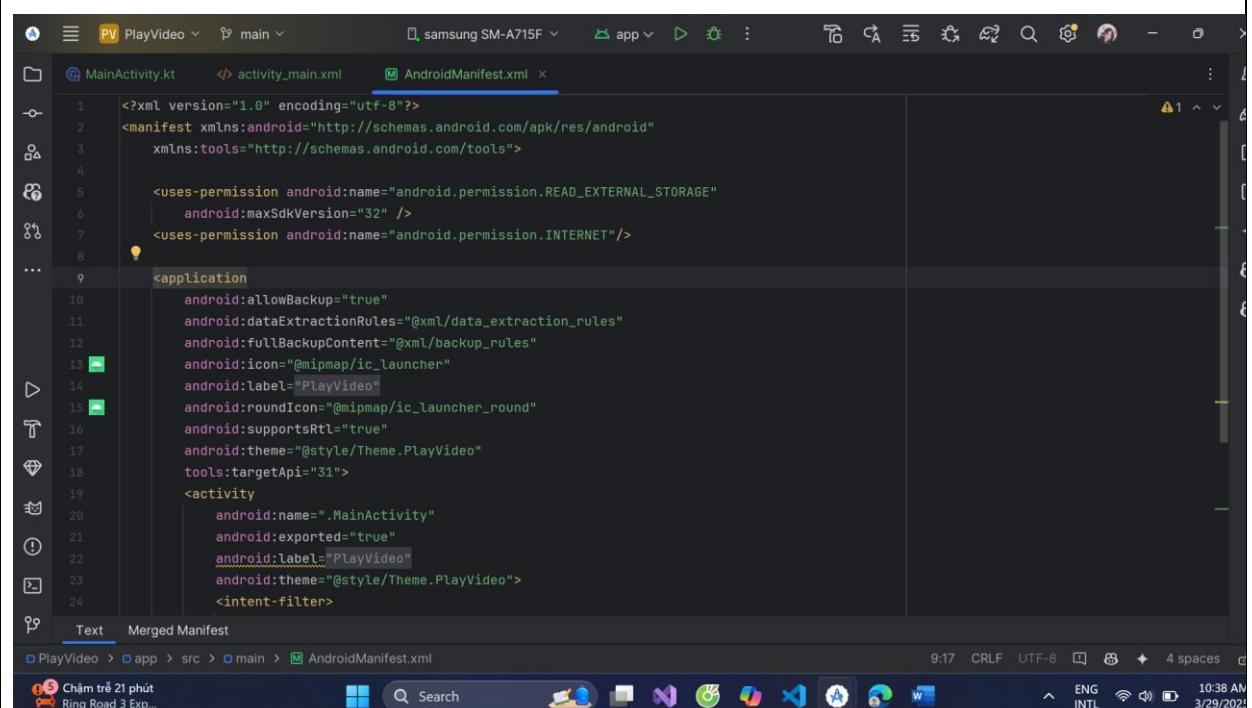
1. **Thêm VideoView và MediaController vào layout.**



2. Chọn video:

- o Cho phép người dùng chọn video từ MediaStore bằng cách sử dụng Intent.ACTION_PICK
MediaStore.Video.Media.EXTERNAL_CONTENT_URI.
- o Hoặc cho phép người dùng nhập URL của video.

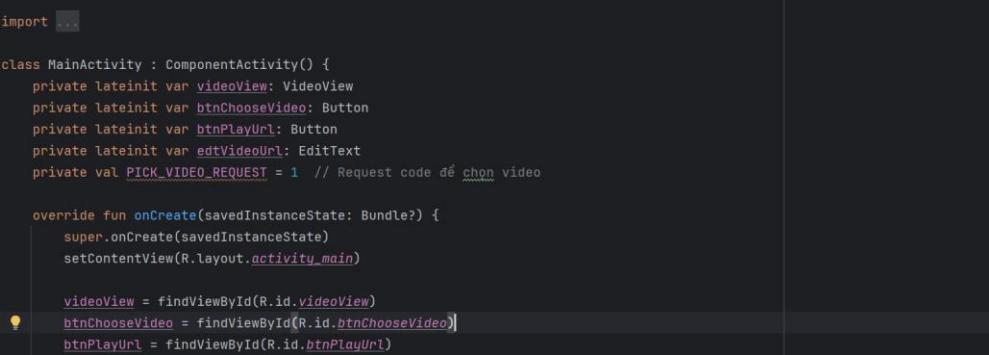
⇒ Vào androidManifest để thêm



3. Phát video:

- o Sử dụng VideoView.setVideoURI() để thiết lập nguồn video.

- Sử dụng `MediaController` để cung cấp các điều khiển phát lại video (play, pause, stop,...).
 - `VideoView.start()` để bắt đầu phát video.

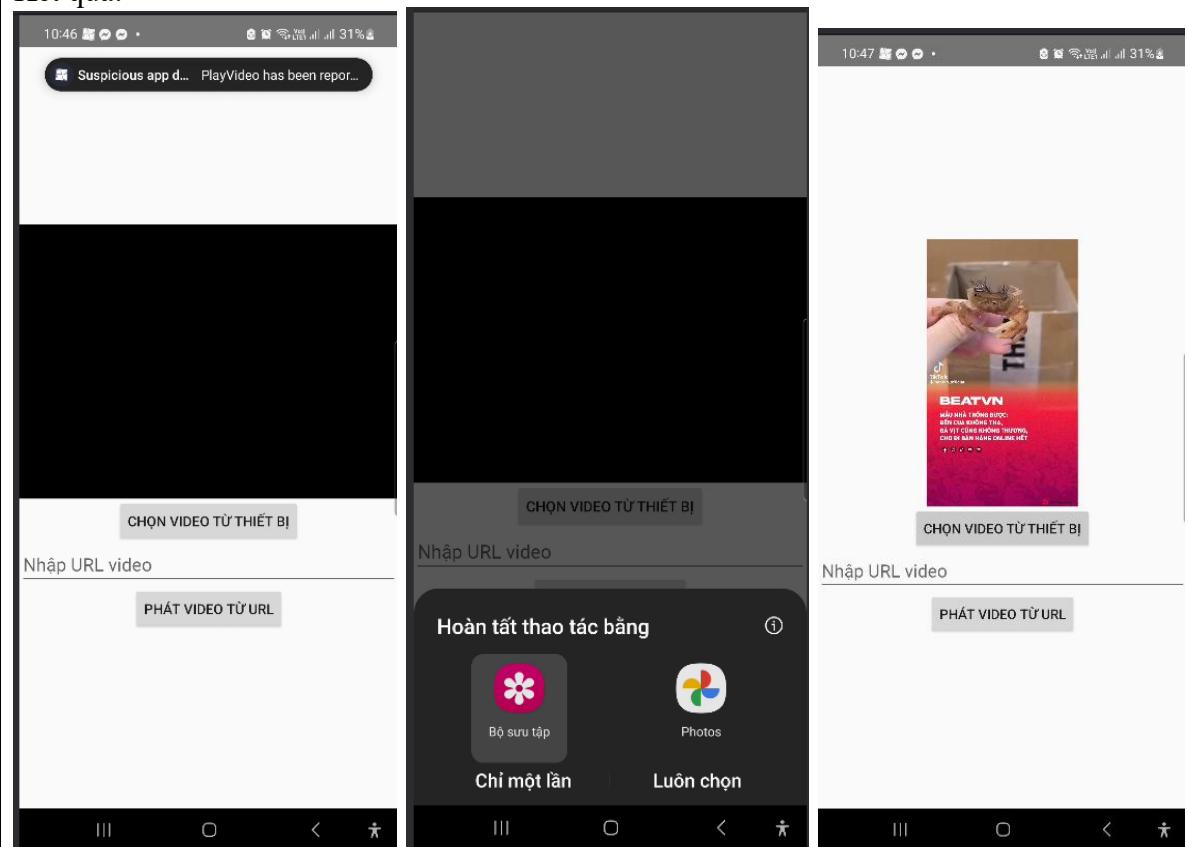


```
1 package com.example.playvideo
2
3 > import ...
4
5 <> class MainActivity : ComponentActivity() {
6     private lateinit var videoView: VideoView
7     private lateinit var btnChooseVideo: Button
8     private lateinit var btnPlayUrl: Button
9     private lateinit var edtVideoUrl: EditText
10    private val PICK_VIDEO_REQUEST = 1 // Request code để chọn video
11
12    @<> override fun onCreate(savedInstanceState: Bundle?) {
13        super.onCreate(savedInstanceState)
14        setContentView(R.layout.activity_main)
15
16        videoView = findViewById(R.id.videoView)
17        btnChooseVideo = findViewById(R.id.btnChooseVideo)
18        btnPlayUrl = findViewById(R.id.btnPlayUrl)
19        edtVideoUrl = findViewById(R.id.edtVideoUrl)
20
21        // Chọn video từ thiết bị
22        btnChooseVideo.setOnClickListener {
23            val intent = Intent(Intent.ACTION_PICK, MediaStore.Video.Media.EXTERNAL_CONTENT_URI)
24            startActivityForResult(intent, PICK_VIDEO_REQUEST)
25    }
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45 }
```

```

25     class MainActivity : ComponentActivity() {
26         override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
27             if (selectedVideoUri != null) {
28                 playVideo(selectedVideoUri)
29             }
30         }
31     }
32
33     private fun playVideo(uri: Uri) {
34         val mediaController = MediaController(context)
35         mediaController.setAnchorView(videoView)
36         videoView.setMediaController(mediaController)
37         videoView.setVideoURI(uri)
38         videoView.start()
39     }
40
41     override fun onPause() {
42         super.onPause()
43         if (videoView.isPlaying) {
44             videoView.pause()
45         }
46     }
47 }
48
49 
```

Kết quả:



BÀI TẬP 8: Ứng dụng đo gia tốc

Mục tiêu:

- Sử dụng cảm biến gia tốc (TYPE_ACCELEROMETER) để đo gia tốc của thiết bị theo ba trục x, y, z.
- Hiển thị giá trị gia tốc lên TextView.
- Hiển thị một hình ảnh động (ví dụ: một quả bóng) di chuyển theo hướng gia tốc.

Mô tả:

Ứng dụng hiển thị các giá trị gia tốc đo được từ cảm biến gia tốc và mô phỏng sự di chuyển của một vật thể dựa trên các giá trị này.

Các bước thực hiện:

- 1. Lấy SensorManager và Sensor:**
 - Lấy SensorManager từ hệ thống.
 - Sử dụng SensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) để lấy đối tượng Sensor.
- 2. Đăng ký SensorEventListener:**
 - Đăng ký một SensorEventListener để lắng nghe các sự kiện từ cảm biến gia tốc.
 - Implement hai phương thức của SensorEventListener:
 - onSensorChanged(SensorEvent event): Xử lý các sự kiện cảm biến, lấy giá trị gia tốc từ event.values.
 - onAccuracyChanged(Sensor sensor, int accuracy): Xử lý khi độ chính xác của cảm biến thay đổi.
- 3. Hiển thị giá trị gia tốc:**
 - Cập nhật các TextView để hiển thị giá trị gia tốc theo trục x, y, z.
- 4. Mô phỏng chuyển động:**
 - Sử dụng một ImageView để hiển thị hình ảnh động.
 - Trong phương thức onSensorChanged(), tính toán sự thay đổi vị trí của hình ảnh dựa trên giá trị gia tốc.
 - Cập nhật vị trí của ImageView.

1.Lấy SensorManager và Sensor:

- Lấy SensorManager từ hệ thống.
- Sử dụng SensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) để lấy đối tượng Sensor.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-feature android:name="android.hardware.sensor.accelerometer" android:required="true"/>
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Dogiatoc"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Dogiatoc"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="Dogiatoc"
            android:theme="@style/Theme.Dogiatoc">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
            <category android:name="android.intent.category.LAUNCHER" />
        
```

2. Đăng ký SensorEventListener:

- Đăng ký một SensorEventListener để lắng nghe các sự kiện từ cảm biến gia tốc.
- Implement hai phương thức của SensorEventListener:
 - onSensorChanged(SensorEvent event): Xử lý các sự kiện cảm biến, lấy giá trị gia tốc từ event.values.
 - onAccuracyChanged(Sensor sensor, int accuracy): Xử lý khi độ chính xác của cảm biến thay đổi.

3. Hiển thị giá trị gia tốc:

- Cập nhật các TextView để hiển thị giá trị gia tốc theo trục x, y, z.

4. Mô phỏng chuyển động:

- Sử dụng một ImageView để hiển thị hình ảnh động.
- Trong phương thức onSensorChanged(), tính toán sự thay đổi vị trí của hình ảnh dựa trên giá trị gia tốc.
- Cập nhật vị trí của ImageView.

```
package com.example.dogiatoc
import ...
class MainActivity : ComponentActivity(), SensorEventListener {
    private lateinit var sensorManager: SensorManager
    private var accelerometer: Sensor? = null
    private lateinit var tvX: TextView
    private lateinit var tvY: TextView
    private lateinit var tvZ: TextView
    private lateinit var ball: ImageView
    private var posX = 0f
    private var posY = 0f
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        // Ánh xạ view
        tvX = findViewById(R.id.tv_x)
        tvY = findViewById(R.id.tv_y)
        tvZ = findViewById(R.id.tv_z)
        ball = findViewById(R.id.ball)
    }
}
```

Dogiatoc > app > src > main > java > com > example > dogiatoc > MainActivity > onSensorChanged

```
class MainActivity : ComponentActivity(), SensorEventListener {
    override fun onCreate(savedInstanceState: Bundle?) {
        // Lấy SensorManager và cảm biến giá tốc
        sensorManager = getSystemService(SENSOR_SERVICE) as SensorManager
        accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
    }
    override fun onResume() {
        super.onResume()
        accelerometer?.let {
            sensorManager.registerListener(listener, it, SensorManager.SENSOR_DELAY_UI)
        }
    }
    override fun onPause() {
        super.onPause()
        sensorManager.unregisterListener(listener)
    }
    @SuppressLint("SetTextI18n")
    override fun onSensorChanged(event: SensorEvent) {
        if (event.sensor.type == Sensor.TYPE_ACCELEROMETER) {
            posX = event.values[0]
            posY = event.values[1]
            posZ = event.values[2]
            ball.x = posX
            ball.y = posY
        }
    }
}
```

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The left sidebar shows the project structure under "Android". The "app" module contains "manifests", "kotlin+java", "res", and "layout" directories.
- MainActivity.kt:** The code editor displays the `MainActivity` class. It includes methods for handling sensor changes and moving a ball. The ball's position is updated every 2 units, and it is bounded within a 1000x1000 area.
- AndroidManifest.xml:** The manifest file is shown at the bottom of the editor.
- activity_main.xml:** The XML layout file is shown at the bottom of the editor.
- Toolbars and Status Bar:** The top bar shows the device as "samsung SM-A715F", the app as "app", and the file as "MainActivity.kt". The status bar indicates the time as 11:05 AM and the date as 3/29/2025.

```
class MainActivity : ComponentActivity(), SensorEventListener {
    override fun onSensorChanged(event: SensorEvent?) {
        // Hiển thị giá trị trên TextView
        tvX.text = "X: $x"
        tvY.text = "Y: $y"
        tvZ.text = "Z: $z"

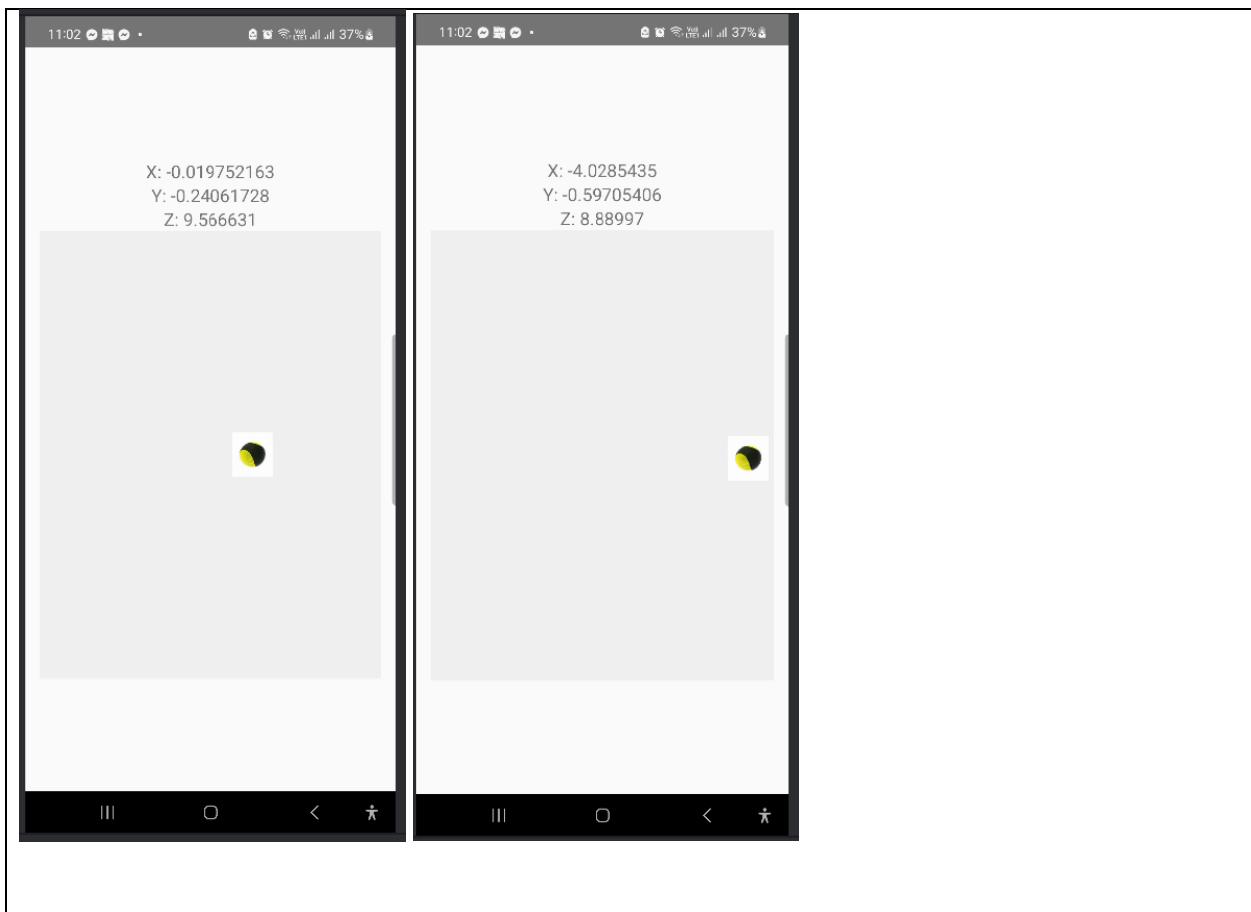
        // Cập nhật vị trí hình ảnh
        moveBall(x, y)
    }

    private fun moveBall(x: Float, y: Float) {
        posX -= x * 2
        posY += y * 2

        // Giới hạn biên
        if (posX < 0) posX = 0f
        if (posY < 0) posY = 0f
        if (posX > 1000) posX = 1000f
        if (posY > 1000) posY = 1000f

        ball.translationX = posX
        ball.translationY = posY
    }
}
```

Kết quả



BÀI TẬP 9: Ứng dụng la bàn

Mục tiêu:

- Sử dụng cảm biến từ trường (TYPE_MAGNETIC_FIELD) và cảm biến gia tốc (TYPE_ACCELEROMETER) để xác định hướng bắc.
- Hiển thị hướng bắc trên một ImageView (ví dụ: kim la bàn).
- Hiển thị góc lệch so với hướng bắc.

Mô tả:

Ứng dụng hiển thị la bàn và hướng bắc dựa trên dữ liệu từ cảm biến từ trường và cảm biến gia tốc.

Các bước thực hiện:

1. Lấy SensorManager và Sensor:

- Lấy SensorManager từ hệ thống.
- Sử dụng SensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) để lấy đối tượng Sensor từ trường.
- Sử dụng SensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) để lấy đối tượng Sensor gia tốc.

2. Đăng ký SensorEventListener:

- Đăng ký một SensorEventListener để lắng nghe các sự kiện từ cảm biến từ trường và gia tốc.
- Trong phương thức onSensorChanged():
 - Lấy giá trị từ cả hai cảm biến.
 - Sử dụng các hàm SensorManager.getRotationMatrix() và SensorManager.getOrientation() để tính toán hướng bắc và góc lệch.

3. Hiển thị la bàn:

- Sử dụng một ImageView để hiển thị hình ảnh la bàn.
- Sử dụng phương thức ImageView.setRotation() để xoay hình ảnh la bàn theo hướng bắc.

4. Hiển thị góc lệch:

- Hiển thị giá trị góc lệch so với hướng bắc lên một TextView.

Lưu ý:

- **Quyền (Permissions):** Khai báo các quyền cần thiết trong AndroidManifest.xml, bao gồm quyền sử dụng cảm biến.
- **Độ chính xác:** Độ chính xác của cảm biến có thể bị ảnh hưởng bởi nhiều từ môi trường. Cần có các biện pháp lọc dữ liệu hoặc hiệu chỉnh để cải thiện độ chính xác.
- **Tiết kiệm pin:** Hủy đăng ký SensorEventListener khi không sử dụng cảm biến để tiết kiệm pin.
- **Kiểm tra cảm biến:** Kiểm tra xem thiết bị có hỗ trợ các cảm biến cần thiết hay không trước khi sử dụng.

1. Lấy SensorManager và Sensor:

- Lấy SensorManager từ hệ thống.

- o Sử dụng `SensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)` để lấy đối tượng Sensor từ trường.
- o Sử dụng `SensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)` để lấy đối tượng Sensor gia tốc.

```

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

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

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="LABAN"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.LABAN"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="LABAN"
            android:theme="@style/Theme.LABAN">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```

2. Đăng ký SensorEventListener:

3. Đăng ký một SensorEventListener để lắng nghe các sự kiện từ cảm biến từ trường và gia tốc.

4. Trong phương thức `onSensorChanged()`:

- Lấy giá trị từ cả hai cảm biến.
- Sử dụng các hàm `SensorManager.getRotationMatrix()` và `SensorManager.getOrientation()` để tính toán hướng bắc và góc lệch.

- **Hiển thị la bàn:**

1. Sử dụng một `ImageView` để hiển thị hình ảnh la bàn.
2. Sử dụng phương thức `ImageView.setRotation()` để xoay hình ảnh la bàn theo hướng bắc.

- **Hiển thị góc lệch:**

1. Hiển thị giá trị góc lệch so với hướng bắc lên một `TextView`.

The image shows two identical screenshots of the Android Studio interface, each displaying the `MainActivity.kt` file. The code is written in Kotlin and handles sensor events.

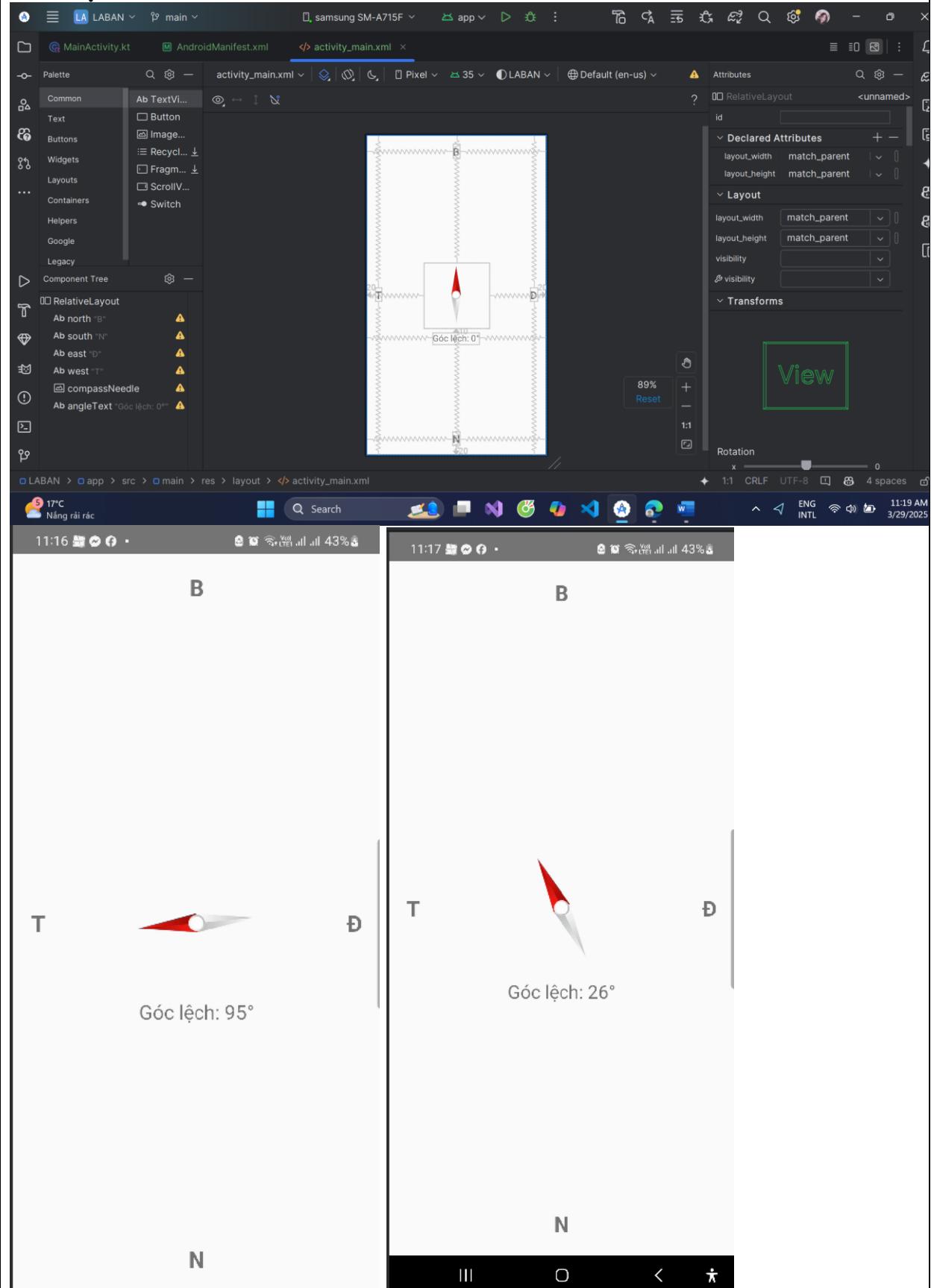
```
1 package com.example.laban
2
3 > import ...
4
5
6 <> class MainActivity : ComponentActivity() , SensorEventListener {
7     private lateinit var sensorManager: SensorManager
8     private var accelerometer: Sensor? = null
9     private var magnetometer: Sensor? = null
10
11     private val gravity = FloatArray( size: 3 )
12     private val geomagnetic = FloatArray( size: 3 )
13     private var azimuth = 0f // Góc lệch
14
15     private lateinit var compassNeedle: ImageView
16     private lateinit var angleText: TextView
17
18     override fun onCreate(savedInstanceState: Bundle?) {
19         super.onCreate(savedInstanceState)
20         setContentView(R.layout.activity_main)
21
22         compassNeedle = findViewById(R.id.compassNeedle)
23         angleText = findViewById(R.id.angleText)
24
25         // Lấy cảm biến
26         sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
27         accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
28         magnetometer = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)
29
30     }
31
32     override fun onResume() {
33         super.onResume()
34         accelerometer?.also { sensor ->
35             sensorManager.registerListener( listener: this, sensor, SensorManager.SENSOR_DELAY_UI )
36         }
37         magnetometer?.also { sensor ->
38             sensorManager.registerListener( listener: this, sensor, SensorManager.SENSOR_DELAY_UI )
39         }
40
41     }
42
43     override fun onPause() {
44         super.onPause()
45         sensorManager.unregisterListener( listener: this )
46     }
47
48     override fun onSensorChanged(event: SensorEvent?) {
49         ...
50     }
51
52 }
53
54
55
56
57
58
59
60
61
62
63
64
```

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** LABAN > app > src > main > java > com > example > laban > MainActivity
- MainActivity.kt:** This file contains Java code for a SensorEventListener. It overrides the onSensorChanged method to handle sensor events. The code uses System.arraycopy to copy sensor data into arrays for gravity and geomagnetic fields. It then calls getRotationMatrix and getOrientation to calculate the rotation matrix and orientation. Finally, it updates the azimuth value and sets the compass needle's rotation.
- AndroidManifest.xml:** Associated with the activity.
- activity_main.xml:** Associated with the activity.
- Toolbar:** Shows the device name (Samsung SM-A715F), battery level (12%), signal strength, and other system icons.
- Bottom Bar:** Shows the date (3/29/2025), time (11:18 AM), and system status (ENG INTL).

```
24     class MainActivity : ComponentActivity() , SensorEventListener {
62     }
63
64     override fun onSensorChanged(event: SensorEvent?) {
65         if (event == null) return
66
67         when (event.sensor.type) {
68             Sensor.TYPE_ACCELEROMETER -> {
69                 System.arraycopy(event.values, 0, gravity, 0, event.values.size)
70             }
71             Sensor.TYPE_MAGNETIC_FIELD -> {
72                 System.arraycopy(event.values, 0, geomagnetic, 0, event.values.size)
73             }
74         }
75
76         val rotationMatrix = FloatArray( size: 9)
77         val orientation = FloatArray( size: 3)
78
79         if (SensorManager.getRotationMatrix(rotationMatrix, null, gravity, geomagnetic)) {
80             SensorManager.getOrientation(rotationMatrix, orientation)
81             azimuth = Math.toDegrees(orientation[0].toDouble()).toFloat()
82             azimuth = (azimuth + 360) % 360
83
84             // Xoay kim la bàn
85             compassNeedle.rotation = -azimuth
86
87             // Cập nhật góc lệch
88             angleText.text = "Góc lệch: ${azimuth.toInt()}"
89         }
90     }
91
92     override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {}
93 }
```

Giao diện:



BÀI TẬP 10: ỦNG DỤNG CLIENT-SERVER ĐƠN GIẢN SỬ DỤNG TCP

Mục tiêu:

- Xây dựng một ứng dụng Android (Client) có thể gửi và nhận dữ liệu từ một ứng dụng Server (có thể chạy trên PC hoặc thiết bị Android khác) sử dụng giao thức TCP.
- Ứng dụng Client cho phép người dùng nhập tin nhắn và gửi đến Server.
- Ứng dụng Server nhận tin nhắn và hiển thị chúng.

Mô tả:

Ứng dụng này minh họa giao tiếp cơ bản giữa Client và Server sử dụng TCP, tập trung vào việc thiết lập kết nối, gửi và nhận dữ liệu.

Các bước thực hiện:

Phía Server:

1. **Tạo ServerSocket:** Sử dụng `ServerSocket` để lắng nghe kết nối đến trên một cổng cụ thể.
2. **Chấp nhận kết nối:** Sử dụng `ServerSocket.accept()` để chấp nhận kết nối từ Client.
3. **Tạo luồng (Thread):** Tạo một luồng mới để xử lý giao tiếp với mỗi Client.
4. **Giao tiếp:** Sử dụng `InputStream` và `OutputStream` của `Socket` để nhận và gửi dữ liệu.
5. **Đóng kết nối:** Đóng `Socket` và `ServerSocket` khi hoàn tất giao tiếp.

Phía Client (Android):

1. **Tạo Socket:** Sử dụng `Socket` để kết nối đến Server trên địa chỉ IP và cổng cụ thể.
2. **Giao tiếp:** Sử dụng `InputStream` và `OutputStream` của `Socket` để gửi và nhận dữ liệu.
3. **Gửi dữ liệu:** Lấy dữ liệu từ người dùng (ví dụ: một `EditText`) và gửi đến Server.
4. **Nhận dữ liệu:** Nhận phản hồi từ Server và hiển thị cho người dùng.
5. **Đóng kết nối:** Đóng `Socket` khi hoàn tất giao tiếp.

Phía Server:

1. **Tạo ServerSocket:** Sử dụng `ServerSocket` để lắng nghe kết nối đến trên một cổng cụ thể.
2. **Chấp nhận kết nối:** Sử dụng `ServerSocket.accept()` để chấp nhận kết nối từ Client.
3. **Tạo luồng (Thread):** Tạo một luồng mới để xử lý giao tiếp với mỗi Client.
4. **Giao tiếp:** Sử dụng `InputStream` và `OutputStream` của `Socket` để nhận và gửi dữ liệu.
5. **Đóng kết nối:** Đóng `Socket` và `ServerSocket` khi hoàn tất giao tiếp.

Phía Client (Android):

1. **Tạo Socket:** Sử dụng `Socket` để kết nối đến Server trên địa chỉ IP và cổng cụ thể.
2. **Giao tiếp:** Sử dụng `InputStream` và `OutputStream` của `Socket` để gửi và nhận dữ liệu.
3. **Gửi dữ liệu:** Lấy dữ liệu từ người dùng (ví dụ: một `EditText`) và gửi đến Server.
4. **Nhận dữ liệu:** Nhận phản hồi từ Server và hiển thị cho người dùng.
5. **Đóng kết nối:** Đóng `Socket` khi hoàn tất giao tiếp.

-Tạo class TCPServer (Cần tải JDK nếu thiếu): (**VÌ GIỚI HẠN FILE LÀ 20MB nên em sẽ copy code vào đây thay vì chụp ảnh**)

```
package com.example.tcp;

import java.io.*;
import java.net.*;

public class TCPServer {

    public static void main(String[] args) {
        int port = 5000; // Cổng server
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Server is running in port " + port);

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Client connected: " + clientSocket.getInetAddress());
                // Tạo luồng riêng để xử lý Client
                new Thread(() -> handleClient(clientSocket)).start();
            }
        }
    }

    private void handleClient(Socket clientSocket) {
        try {
            BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
            String message = reader.readLine();
            if (message != null) {
                System.out.println("Received message: " + message);
                PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true);
                writer.println("Hello, Client!");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
}

} catch (IOException e) {

    e.printStackTrace();

}

}

private static void handleClient(Socket clientSocket) {

    try {

        BufferedReader input = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

        PrintWriter output = new PrintWriter(clientSocket.getOutputStream(), true);

        BufferedReader consoleInput = new BufferedReader(new InputStreamReader(System.in));

        while (true) {

            String message = input.readLine(); // Nhận tin nhắn từ client

            if (message == null) break; // Nếu client ngắt kết nối thì dừng

            System.out.println("Client gửi: " + message); // Hiển thị tin nhắn trên terminal

            // Nhập phản hồi từ terminal

            System.out.print("Nhập phản hồi: ");

    
```

```
String response = consoleInput.readLine();

// Gửi phản hồi về Client

output.println(response);

System.out.println("Gửi phản hồi: " + response); // Hiển thị phản hồi trên terminal

}

clientSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}
```

Đăng kí trong AndroidManifest:): (VÌ GIỚI HẠN FILE LÀ 20MB nên em sẽ copy code vào đây thay vì chụp ảnh)

-

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

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

```
<application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.TCP"
    tools:targetApi="31">
    <activity
        android:name=".MainActivity"
        android:exported="true"
        android:label="@string/app_name"
        android:theme="@style/Theme.TCP">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

-MAIN ACTIVITY: (VÌ GIỚI HẠN FILE LÀ 20MB nên em sẽ copy code vào đây
thay vì chụp ảnh)

package com.example.tcp

```
import android.os.Bundle

import android.widget.Button

import android.widget.EditText

import android.widget.TextView

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.activity.enableEdgeToEdge

import androidx.compose.foundation.layout.fillMaxSize

import androidx.compose.foundation.layout.padding

import androidx.compose.material3.Scaffold

import androidx.compose.material3.Text

import androidx.compose.runtime.Composable

import androidx.compose.ui.Modifier

import androidx.compose.ui.tooling.preview.Preview

import com.example.tcp.ui.theme.TCPTheme

import java.io.BufferedReader

import java.io.IOException

import java.io.InputStreamReader

import java.io.PrintWriter

import java.net.Socket
```

```
import kotlin.concurrent.thread

class MainActivity : ComponentActivity() {

    private lateinit var etMessage: EditText

    private lateinit var btnSend: Button

    private lateinit var tvResponse: TextView

    private val serverIP = "10.0.107.140" // Thay bằng địa chỉ IP của Server

    private val serverPort = 5000

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        setContentView(R.layout.activity_main)

        etMessage = findViewById(R.id.etMessagee)

        btnSend = findViewById(R.id.btnSendd)

        tvResponse = findViewById(R.id.tvResponsee)

        btnSend.setOnClickListener {

            val message = etMessage.text.toString()

            thread {
                val response = sendRequest(message)
                tvResponse.text = response
            }
        }
    }
}
```

```
if (message.isNotEmpty()) {  
    sendMessage(message)  
}  
}  
}  
  
private fun sendMessage(message: String) {  
    thread {  
        try {  
            val socket = Socket(serverIP, serverPort)  
            val output = PrintWriter(socket.getOutputStream(), true)  
            val input = BufferedReader(InputStreamReader(socket.getInputStream()))  
  
            output.println(message) // Gửi tin nhắn đến Server  
  
            val response = input.readLine() // Nhận phản hồi từ Server  
  
            runOnUiThread { tvResponse.text = response }  
  
            socket.close()  
        } catch (e: Exception) {  
        }  
    }  
}
```

```
e.printStackTrace()
```

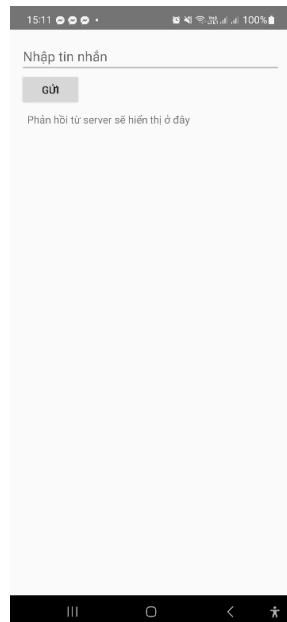
```
runOnUiThread { tvResponse.text = "Không thể kết nối đến server" }
```

```
}
```

```
}
```

```
}
```

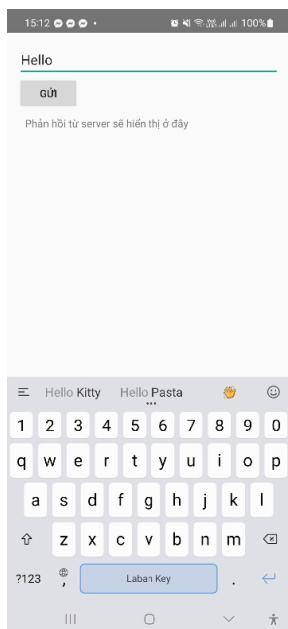
```
}
```



- **Chạy server:**

The screenshot shows the Android Studio interface. In the top navigation bar, the tabs are 'TCP' and 'main'. Below the tabs, the code editor displays Java code for a TCP server. The code includes a main method that creates a socket, prints a message when a client connects, and handles the client in a new thread. It also catches IOException and prints the stack trace. A private static void handleClient method is defined to handle each client socket. The terminal below shows the command line for running the server: 'cd C:\Users\minia\AndroidStudioProjects\TCP', 'javac -d . com.example.tcp.TCPServer.java', and 'java com.example.tcp.TCPServer'. The output indicates the server is running on port 5000. The status bar at the bottom right shows the time as 17:45, CRLF, UTF-8, and 4 spaces.

-Client viết “Hello” và nhấn gửi:



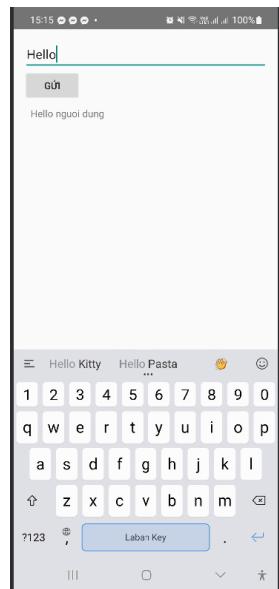
Sau khi Client ấn gửi thì trên Terminal sẽ hiển thị như sau:

The screenshot shows the terminal window from the previous screenshot. The server's response is displayed: 'Client connected: /10.0.107.144' followed by 'Client g?i: Hello'. The server then sends a response back to the client: 'Nh?p ph?n h?i:'.

Server gửi phản hồi đến client:

```
Terminal Local + ▾
PS C:\Users\minia\AndroidStudioProjects\TCP\app\src\main\java> javac -d . com/example/tcp/TCPServer.java
PS C:\Users\minia\AndroidStudioProjects\TCP\app\src\main\java> java com.example.tcp.TCPServer
Server is running in port 5000
Client connected: /10.0.107.144
Client g?i: Hello
Nh?p ph?n h?i: Hello nguoi dung
G?i ph?n h?i: Hello nguoi dung
```

Sau khi gửi đến người dùng thì màn hình người dùng sẽ hiển thị phản hồi “Hello nguoi dung”



BÀI TẬP 11: Ứng dụng gửi và nhận tin nhắn sử dụng UDP

Mục tiêu:

- Xây dựng một ứng dụng Android có thể gửi và nhận tin nhắn sử dụng giao thức UDP.
- Ứng dụng cho phép người dùng gửi tin nhắn đến một thiết bị khác trên mạng.
- Ứng dụng có thể nhận tin nhắn từ các thiết bị khác.

Mô tả:

Ứng dụng này minh họa giao tiếp sử dụng UDP, tập trung vào việc gửi và nhận các gói tin độc lập.

Các bước thực hiện:

Gửi tin nhắn:

1. **Lấy dữ liệu:** Lấy dữ liệu từ người dùng (ví dụ: một EditText).
2. **Tạo DatagramPacket:** Tạo một DatagramPacket chứa dữ liệu cần gửi, địa chỉ IP và cổng của người nhận.
3. **Tạo DatagramSocket:** Tạo một DatagramSocket để gửi gói tin.
4. **Gửi gói tin:** Sử dụng DatagramSocket.send() để gửi DatagramPacket.
5. **Đóng DatagramSocket:** Đóng DatagramSocket sau khi gửi.

Nhận tin nhắn:

1. **Tạo DatagramSocket:** Tạo một DatagramSocket và lắng nghe trên một cổng cụ thể.
2. **Tạo DatagramPacket:** Tạo một DatagramPacket để chứa dữ liệu nhận được.
3. **Nhận gói tin:** Sử dụng DatagramSocket.receive() để nhận DatagramPacket.
4. **Xử lý dữ liệu:** Trích xuất dữ liệu từ DatagramPacket và hiển thị.
5. **Đóng DatagramSocket:** Đóng DatagramSocket khi không còn cần nhận tin nhắn.

Lưu ý:

- **Quyền (Permissions):** Đảm bảo khai báo các quyền cần thiết trong AndroidManifest.xml, bao gồm android.permission.INTERNET.
- **Luồng (Threads):** Thực hiện các hoạt động mạng trong các luồng riêng để tránh làm treo UI thread.
- **Xử lý lỗi:** Xử lý các trường hợp lỗi có thể xảy ra, chẳng hạn như kết nối không thành công, mất kết nối, v.v.
- **Giao thức:** Xác định rõ giao thức giao tiếp giữa Client và Server (ví dụ: định dạng tin nhắn, các lệnh).

Gửi tin nhắn:

6. **Lấy dữ liệu:** Lấy dữ liệu từ người dùng (ví dụ: một EditText).
7. **Tạo DatagramPacket:** Tạo một DatagramPacket chứa dữ liệu cần gửi, địa chỉ IP và cổng của người nhận.
8. **Tạo DatagramSocket:** Tạo một DatagramSocket để gửi gói tin.

9. **Gửi gói tin:** Sử dụng `DatagramSocket.send()` để gửi `DatagramPacket`.
10. **Đóng DatagramSocket:** Đóng `DatagramSocket` sau khi gửi.

Nhận tin nhắn:

6. **Tạo DatagramSocket:** Tạo một `DatagramSocket` và lắng nghe trên một cổng cụ thể.
7. **Tạo DatagramPacket:** Tạo một `DatagramPacket` để chứa dữ liệu nhận được.
8. **Nhận gói tin:** Sử dụng `DatagramSocket.receive()` để nhận `DatagramPacket`.
9. **Xử lý dữ liệu:** Trích xuất dữ liệu từ `DatagramPacket` và hiển thị.
10. **Đóng DatagramSocket:** Đóng `DatagramSocket` khi không còn cần nhận tin nhắn.

MainActivity: (**VÌ GIỚI HẠN FILE LÀ 20MB nên em sẽ copy code vào đây thay vì chụp ảnh**)

```
package com.example.udp

import UDPReceiver
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import androidx.activity.ComponentActivity
import java.net.DatagramPacket
import java.net.DatagramSocket
import java.net.InetAddress
import kotlin.concurrent.thread

class MainActivity : ComponentActivity() {
    private lateinit var udpReceiver: UDPReceiver

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

        val tvReceivedMessages = findViewById<TextView>(R.id.tvReceivedMessages)
        val etMessage = findViewById<EditText>(R.id.etMessage)
        val etIP = findViewById<EditText>(R.id.etIpAddress)
        val btnSend = findViewById<Button>(R.id.btnSend)

        // Bắt đầu nhận tin nhắn
        udpReceiver = UDPReceiver(tvReceivedMessages)
        udpReceiver.startReceiving()

        // Xử lý khi nhấn nút Gửi
        btnSend.setOnClickListener {
            val message = etMessage.text.toString()
            val ip = etIP.text.toString()
            val port = 12345 // Cổng cố định

            if (message.isNotEmpty() && ip.isNotEmpty()) {
```

```

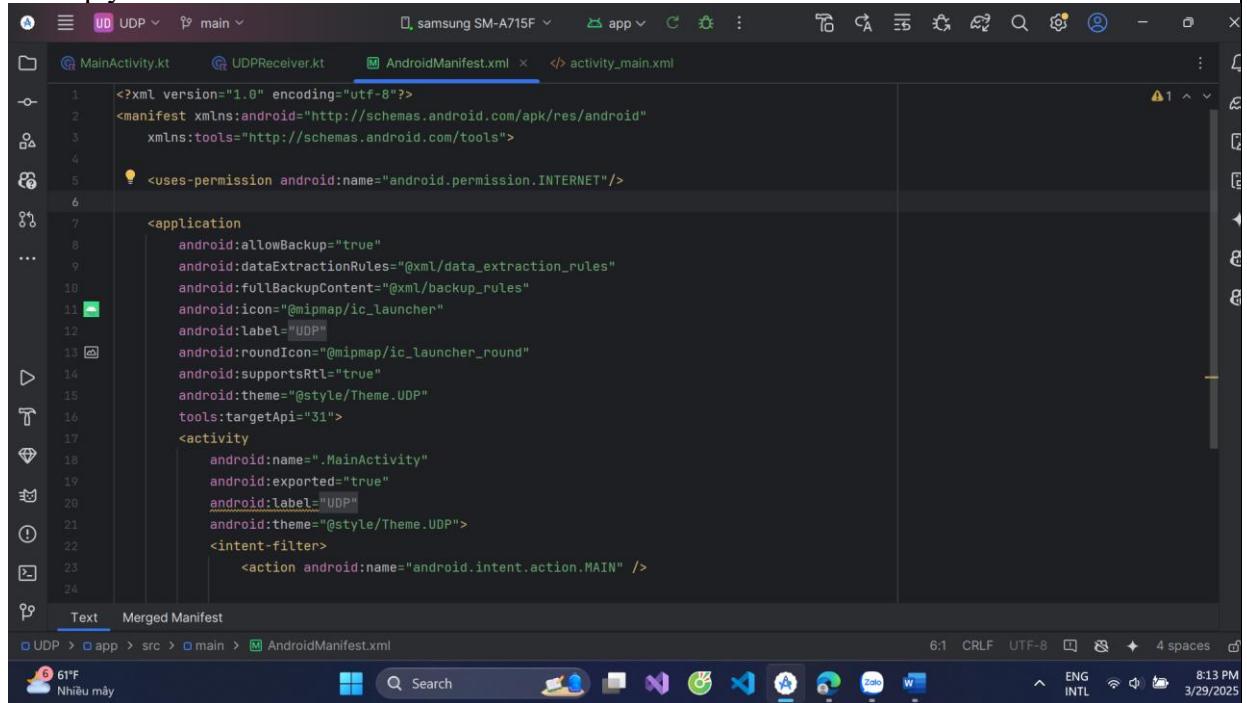
        sendUDPMessages(message, ip, port)
    }
}
}

private fun sendUDPMessages(message: String, ip: String, port: Int) {
    thread {
        try {
            val socket = DatagramSocket()
            val address = InetAddress.getByName(ip)
            val data = message.toByteArray()
            val packet = DatagramPacket(data, data.size, address, port)

            socket.send(packet)
            socket.close()
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
}
}

```

-Xin quyền:



-Tạo lớp UDPReceiver: **(VÌ GIỚI HẠN FILE LÀ 20MB nên em sẽ copy code vào đây thay vì chụp ảnh)**

```

import android.os.Handler
import android.os.Looper
import android.widget.TextView
import java.net.DatagramPacket
import java.net.DatagramSocket

```

```

import kotlin.concurrent.thread

class UDPReceiver(private val textView: TextView) {

    private val port = 12345 // Cổng UDP để lắng nghe tin nhắn
    private var isRunning = true
    private val handler = Handler(Looper.getMainLooper())
    private var socket: DatagramSocket? = null // Thêm socket để có thể đóng sau này

    fun startReceiving() {
        thread {
            try {
                socket = DatagramSocket(port)
                val buffer = ByteArray(1024)

                while (isRunning) {
                    val packet = DatagramPacket(buffer, buffer.size)
                    socket?.receive(packet)

                    val message = String(packet.data, 0, packet.length)
                    handler.post {
                        textView.append("\nNhận: $message")
                    }
                }
            } catch (e: Exception) {
                e.printStackTrace()
            } finally {
                socket?.close() // Đóng socket khi vòng lặp kết thúc
            }
        }
    }

    fun stopReceiving() {
        isRunning = false
        socket?.close() // Đóng socket để dừng receive ngay lập tức
    }
}

```

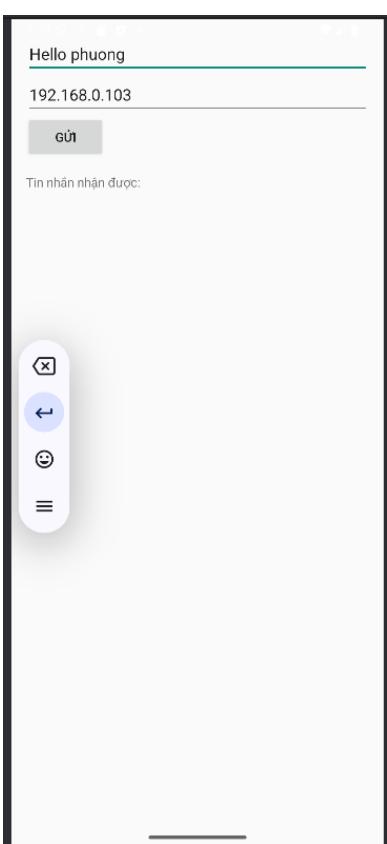
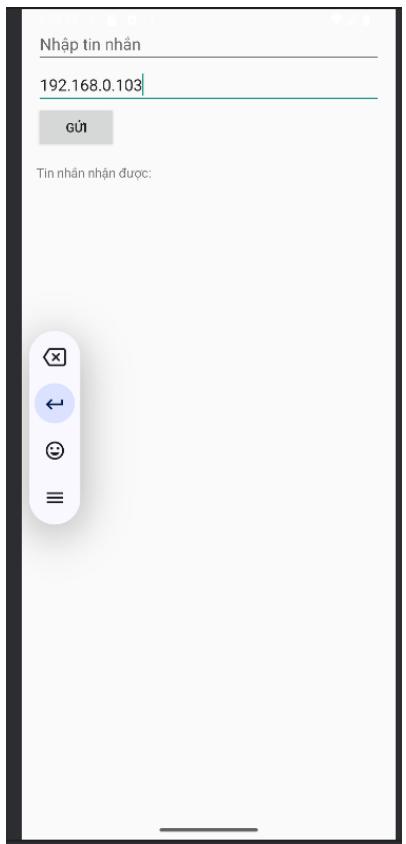
Thiết bị thật:



Thiết bị ảo



-Nhập ip wifi trên máy thật vào máy ảo (Cả 2 máy cùng 1 wifi)



-Sau khi nhấn 5 lần thì trên điện thoại thật đã nhận được 5 tin nhắn:

