

**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	2251061906	Nguyễn Quốc Trung	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

Tạo giao diện

Item 1
Sub Item 1

Item 2
Sub Item 2

Item 3
Sub Item 3

Item 4
Sub Item 4

Item 5
Sub Item 5

Item 6
Sub Item 6

Item 7
Sub Item 7

Item 8
Sub Item 8

Item 9
Sub Item 9

Item 10
Sub Item 10

Item 11
Sub Item 11

Thêm yêu cầu quyền trong Android Manifest

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools">
4          <uses-permission android:name="android.permission.READ_CONTACTS" />
5          <application
6              android:allowBackup="true"
7              android:dataExtractionRules="@xml/data_extraction_rules"
8              android:fullBackupContent="@xml/backup_rules"
9              android:icon="@mipmap/ic_launcher"
10             android:label="@string/app_name"
11             android:roundIcon="@mipmap/ic_launcher_round"
12             android:supportsRtl="true"
13             android:theme="@style/Theme.AppCompat.DayNight"
14             tools:targetApi="31">
15                 <activity
16                     android:name=".MainActivity"
17                     android:exported="true"
18                     android:label="@string/app_name"
19                     android:theme="@style/Theme.AppCompat.DayNight">
20                         <intent-filter>
21                             <action android:name="android.intent.action.MAIN" />
22
23                             <category android:name="android.intent.category.LAUNCHER" />
24                         </intent-filter>
25                     </activity>
26                 </application>
27
28             </manifest>
```

Viết hàm yêu cầu quyền truy cập vào danh bạ

```
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
            Toast.makeText(context, text: "Không có quyền truy cập vào danh bạ", Toast.LENGTH_SHORT).show()
        }
    }
}
```

Viết hàm lấy dữ liệu từ danh bạ sau khi đã có quyền

```

private fun loadContacts() {
    // Xóa dữ liệu cũ trước khi nạp dữ liệu mới
    contactsList.clear()

    try {
        // Lấy dữ liệu từ Content Provider
        val projection = arrayOf(
            ContactsContract.Contacts._ID,
            ContactsContract.Contacts.DISPLAY_NAME_PRIMARY
        )

        val cursor: Cursor? = contentResolver.query(
            ContactsContract.Contacts.CONTENT_URI,
            projection,
            selection: null,
            selectionArgs: null,
            sortOrder: ContactsContract.Contacts.DISPLAY_NAME_PRIMARY + " ASC" // Sắp xếp theo tên
        )

        // Sử dụng HashSet để loại bỏ tên trùng lặp
        val uniqueNames = HashSet<String>()

        cursor?.use {
            val nameColumnIndex = it.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME_PRIMARY)

            if (nameColumnIndex != -1) {
                while (it.moveToNext()) {
                    val name = it.getString(nameColumnIndex)

                    // Chỉ thêm tên nếu nó chưa tồn tại trong tập hợp
                    if (name != null && uniqueNames.add(name)) {
                        contactsList.add(name)
                    }
                }
            }
        }
    }

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

} catch (e: Exception) {
    Toast.makeText(context: this, text: "Error loading contacts: ${e.message}", Toast.LENGTH_SHORT).show()
    e.printStackTrace()
}

```

Gọi hàm vừa tạo ở trong onCreate để khi chạy sẽ thực hiện các logic vừa tạo

```
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(
            context: this,
            Manifest.permission.READ_CONTACTS
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        ActivityCompat.requestPermissions(
            activity: this,
            arrayOf(Manifest.permission.READ_CONTACTS),
            REQUEST_READ_CONTACTS_PERMISSION
        )
    } else {
        loadContacts()
    }
}
```

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:

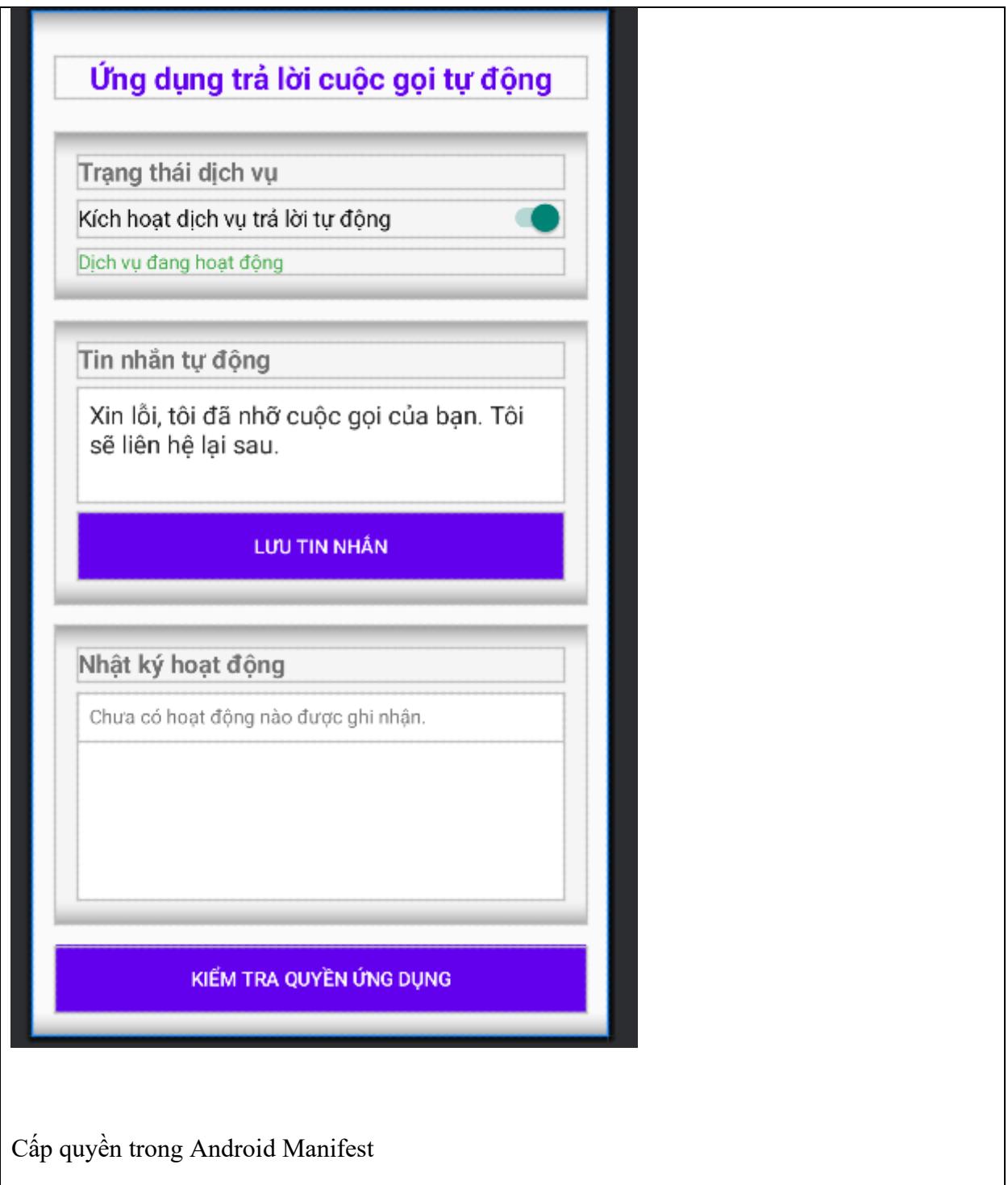
- Sử dụng `SmsManager` để gửi tin nhắn SMS.
- `SmsManager.getDefault().sendTextMessage()` để gửi tin nhắn.

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 được :D

Tạo giao diện



Cấp quyền trong Android Manifest

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools">
4
5     <uses-feature android:name="android.hardware.telephony" android:required="false" />
6
7     <uses-permission android:name="android.permission.READ_PHONE_STATE" />
8     <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS" />
9     <uses-permission android:name="android.permission.READ_CALL_LOG" />
10    <uses-permission android:name="android.permission.SEND_SMS" />
11    <uses-permission android:name="android.permission.READ_PHONE_NUMBERS" />
12    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
13    <uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
14
15    <application
16        android:allowBackup="true"
17        android:dataExtractionRules="@xml/data_extraction_rules"
18        android:fullBackupContent="@xml/backup_rules"
19        android:icon="@mipmap/ic_launcher"
20        android:label="bt2_th5"
21        android:roundIcon="@mipmap/ic_launcher_round"
22        android:supportsRtl="true"
23        android:theme="@style/Theme.AppCompat.DayNight"
24        tools:targetApi="31">
25
26        <receiver android:name=".PhoneStateReceiver"
27            android:enabled="true"
28            android:exported="true">
29            <intent-filter>
30                <action android:name="android.intent.action.PHONE_STATE" />
31                <action android:name="com.example.bt2_th5.SERVICE_STATUS_CHANGED" />
32            </intent-filter>

```

Text Merged Manifest

Viết hàm để tải các cài đặt và lưu cài đặt để trả lời tự động

```

65
66    private fun loadSettings() {
67        val message = sharedPreferences.getString(key: "message", defaultValue: "Xin lỗi, tôi đang bận. Tôi sẽ gọi lại cho bạn sau.")
68        val isEnabled = sharedPreferences.getBoolean(key: "isEnabled", defaultValue: false)
69        val log = sharedPreferences.getString(key: "log", defaultValue: "Chưa có hoạt động nào được ghi nhận.")
70
71        messageEditText.setText(message)
72        enableServiceSwitch.isChecked = isEnabled
73        logTextView.text = log
74    }
75
76    private fun saveSettings() {
77        val message = messageEditText.text.toString()
78        val isEnabled = enableServiceSwitch.isChecked
79
80        val editor = sharedPreferences.edit()
81        editor.putString("message", message)
82        editor.putBoolean("isEnabled", isEnabled)
83        editor.apply()
84    }
85

```

Kiểm tra quyền và hiển thị thông tin

```

private fun checkPermissions() {
    val permissionsToRequest = ArrayList<String>()

    for (permission in requiredPermissions) {
        if (ContextCompat.checkSelfPermission(context, permission) != PackageManager.PERMISSION_GRANTED) {
            permissionsToRequest.add(permission)
        }
    }

    if (permissionsToRequest.isNotEmpty()) {
        ActivityCompat.requestPermissions(
            activity, this,
            permissionsToRequest.toTypedArray(),
            PERMISSIONS_REQUEST_CODE
        )
        statusTextView.text = "Đang chờ cấp quyền..."
    } else {
        statusTextView.text = "Dịch vụ đã sẵn sàng"
    }
}

```

```

    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)

        if (requestCode == PERMISSIONS_REQUEST_CODE) {
            var allGranted = true

            for (result in grantResults) {
                if (result != PackageManager.PERMISSION_GRANTED) {
                    allGranted = false
                    break
                }
            }

            if (allGranted) {
                statusTextView.text = "Dịch vụ đã sẵn sàng"
                addLogEntry(entry: "Đã được cấp tất cả quyền cần thiết")
            } else {
                statusTextView.text = "Thiếu quyền - dịch vụ không thể hoạt động"
                addLogEntry(entry: "Thiếu quyền cần thiết")
                Toast.makeText(
                    context, this,
                    text: "Bạn cần cấp tất cả quyền để ứng dụng hoạt động",
                    Toast.LENGTH_LONG
                ).show()
            }
        }
    }

```

Thêm hàm để ghi lại log

```

fun addLogEntry(entry: String) {
    val dateFormat = SimpleDateFormat(pattern: "dd/MM/yyyy HH:mm:ss", Locale.getDefault())
    val timestamp = dateFormat.format(Date())

    val currentLog = logTextView.text.toString()
    val newLog = if (currentLog == "Chưa có hoạt động nào được ghi nhận.") {
        "[${timestamp}] $entry"
    } else {
        "[${timestamp}] $entry\n$currentLog"
    }

    logTextView.text = newLog

    // Lưu log vào SharedPreferences
    val editor = sharedpreferences.edit()
    editor.putString("log", newLog)
    editor.apply()
}

```

Gọi các hàm vừa tạo vào trong onCreate

```

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

    // Khởi tạo các thành phần UI
    statusTextView = findViewById(R.id.statusTextView)
    messageEditText = findViewById(R.id.messageEditText)
    enableServiceSwitch = findViewById(R.id.enableServiceSwitch)
    saveButton = findViewById(R.id.saveButton)
    logTextView = findViewById(R.id.logTextView)

    // Khởi tạo SharedPreferences để lưu cài đặt
    sharedpreferences = getSharedPreferences(name: "AutoCallResponderPrefs", MODE_PRIVATE)

    // Tải cài đặt đã lưu
    loadSettings()

    // Kiểm tra quyền
    checkPermissions()

    // Xử lý sự kiện nút Lưu
    saveButton.setOnClickListener {
        saveSettings()
        Toast.makeText(context: this, text: "Đã lưu cài đặt", Toast.LENGTH_SHORT).show()
        addLogEntry(entry: "Đã cập nhật cài đặt")
    }
}

```

Tạo PhoneStateReceiver để nhận sự kiện cuộc gọi

```

override fun onReceive(context: Context, intent: Intent?) {
    Log.d(TAG, msg: "BroadcastReceiver triggered with action: ${intent?.action}")

    // Log device info
    logDeviceInfo(context)

    // Show notification for debugging
    showDebugNotification(context, intent)

    if (intent?.action == "android.intent.action.PHONE_STATE") {
        val state = intent.getStringExtra(TelephonyManager.EXTRA_STATE)
        Log.d(TAG, msg: "Phone state changed: $state, previous state: $lastState")

        when (state) {
            TelephonyManager.EXTRA_STATE_RINGING -> {
                incomingNumber = intent.getStringExtra(TelephonyManager.EXTRA_INCOMING_NUMBER)
                lastState = "RINGING"
                ringStartTime = System.currentTimeMillis()
                Log.d(TAG, msg: "Incoming call from $incomingNumber at $ringStartTime")
            }
            TelephonyManager.EXTRA_STATE_IDLE -> {
                Log.d(TAG, msg: "Phone is now IDLE, previous state was: $lastState")
                if (lastState == "RINGING") {
                    val ringDuration = System.currentTimeMillis() - ringStartTime
                    Log.d(TAG, msg: "Ring duration was: $ringDuration ms, timeout is: $RINGING_TIMEOUT ms")
                    Log.d(TAG, msg: "Incoming number is: $incomingNumber")

                    if (ringDuration < RINGING_TIMEOUT && !incomingNumber.isNullOrEmpty()) {
                        Log.d(TAG, msg: "This appears to be a missed call, attempting to send SMS")
                        incomingNumber?.let {
                            sendSMS(context, it)
                        }
                    } else {
                        Log.d(TAG, msg: "Not a missed call or number is null. Duration: $ringDuration, Number: $incomingNumber")
                    }
                }
                lastState = "IDLE"
            }
            TelephonyManager.EXTRA_STATE_OFFHOOK -> {
                lastState = "OFFHOOK"
                Log.d(TAG, msg: "Call answered")
            }
        }
    }
}

private fun showDebugNotification(context: Context, intent: Intent?) {
    try {
        val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
        if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O) {
            val channel = NotificationChannel(id: "phone_state", name: "Phone State", NotificationManager.IMPORTANCE_DEFAULT)
            notificationManager.createNotificationChannel(channel)
        }

        val notification = NotificationCompat.Builder(context, channelId: "phone_state")
            .setSmallIcon(R.drawable.ic_launcher_foreground)
            .setContentTitle("Phone State Changed")
            .setContentText("Action: ${intent?.action}, State: ${intent?.getStringExtra(TelephonyManager.EXTRA_STATE)}")
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .build()

        notificationManager.notify(id: 1, notification)
    } catch (e: Exception) {
        Log.e(TAG, msg: "Failed to show notification: ${e.message}")
    }
}

```

```
private fun checkSmsPermission(context: Context): Boolean {
    val smsPermission = ContextCompat.checkSelfPermission(context, Manifest.permission.SEND_SMS)
    Log.d(TAG, msg: "SMS permission status: ${smsPermission == PackageManager.PERMISSION_GRANTED}")
    return smsPermission == PackageManager.PERMISSION_GRANTED
}

private fun logDeviceInfo(context: Context) {
    Log.d(TAG, msg: "Device Manufacturer: ${android.os.Build.MANUFACTURER}")
    Log.d(TAG, msg: "Device Model: ${android.os.Build.MODEL}")
    Log.d(TAG, msg: "Android Version: ${android.os.Build.VERSION.RELEASE}")
    Log.d(TAG, msg: "API Level: ${android.os.Build.VERSION.SDK_INT}")
}

private fun checkCarrierRestrictions(context: Context): Boolean {
    try {
        val telephonyManager = context.getSystemService(Context.TELEPHONY_SERVICE) as TelephonyManager
        val carrierName = telephonyManager.networkOperatorName
        Log.d(TAG, msg: "Current carrier: $carrierName")
    } catch (e: Exception) {
        Log.e(TAG, msg: "Failed to check carrier: ${e.message}")
    }
    return true
}

private fun sendSMS(context: Context, phoneNumber: String) {
    Log.d(TAG, msg: "Attempting to send SMS to: $phoneNumber")

    if (!checkSmsPermission(context)) {
        Log.e(TAG, msg: "SMS permission not granted, cannot send SMS")
        return
    }
}
```

```

checkCarrierRestrictions(context)

try {
    val sharedpreferences = context.getSharedPreferences(name: "AppPrefs", Context.MODE_PRIVATE)
    val isEnabled = sharedpreferences.getBoolean(key: "service_enabled", defaultValue: true)

    if (!isEnabled) {
        Log.d(TAG, msg: "Service disabled, not sending SMS")
        return
    }

    val message = sharedpreferences.getString(key: "auto_reply_message",
        defaultValue: "Xin lỗi, tôi đang bận. Tôi sẽ gọi lại cho bạn sau.")
        ?: "Xin lỗi, tôi đang bận. Tôi sẽ gọi lại cho bạn sau."

    Log.d(TAG, msg: "Sending message: \"$message\" to $phoneNumber")

    // Try all available methods to send SMS

    // Method 1: Direct SMS Manager
    try {
        val smsManager = SmsManager.getDefault()
        smsManager.sendTextMessage(phoneNumber, scAddress: null, message, sentIntent: null, deliveryIntent: null)
        Log.d(TAG, msg: "Method 1: SMS sent via SmsManager directly")

        // Show notification of success
        showResultNotification(context, title: "SMS Sent", message: "Message sent to $phoneNumber")
        return
    } catch (e: Exception) {
        Log.e(TAG, msg: "Method 1 failed: ${e.message}")
    }

    // Method 2: SMS Intent
    try {
        val smsIntent = Intent(Intent.ACTION_SENDTO)
        smsIntent.data = Uri.parse(uriString: "smsto:$phoneNumber")
        smsIntent.putExtra(name: "sms_body", message)
        smsIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
        context.startActivity(smsIntent)
        Log.d(TAG, msg: "Method 2: SMS intent launched")
        return
    } catch (e: Exception) {
        Log.e(TAG, msg: "Method 2 failed: ${e.message}")
    }

    // Method 3: Try with multipart message
    try {
        val smsManager = SmsManager.getDefault()
        val parts = smsManager.divideMessage(message)
        smsManager.sendMultipartTextMessage(phoneNumber, scAddress: null, parts, sentIntents: null, deliveryIntents: null)
        Log.d(TAG, msg: "Method 3: Multipart SMS sent")
        return
    } catch (e: Exception) {
        Log.e(TAG, msg: "Method 3 failed: ${e.message}")
    }

    // Show failure notification
    showResultNotification(context, title: "SMS Failed", message: "Could not send SMS to $phoneNumber")

} catch (e: Exception) {
    Log.e(TAG, msg: "Failed to send SMS: ${e.message}")
    e.printStackTrace()
    showResultNotification(context, title: "SMS Error", message: e.message ?: "Unknown error")
}

```

```
private fun showResultNotification(context: Context, title: String, message: String) {
    try {
        val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
        if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O) {
            val channel = NotificationChannel( id: "sms_result", name: "SMS Results", NotificationManager.IMPORTANCE_DEFAULT)
            notificationManager.createNotificationChannel(channel)
        }

        val notification = NotificationCompat.Builder(context, channelId: "sms_result")
            .setSmallIcon(R.drawable.ic_launcher_foreground)
            .setContentTitle(title)
            .setContentText(message)
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .build()

        notificationManager.notify( id: 2, notification)
    } catch (e: Exception) {
        Log.e(TAG, msg: "Failed to show result notification: ${e.message}")
    }
}
```

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:

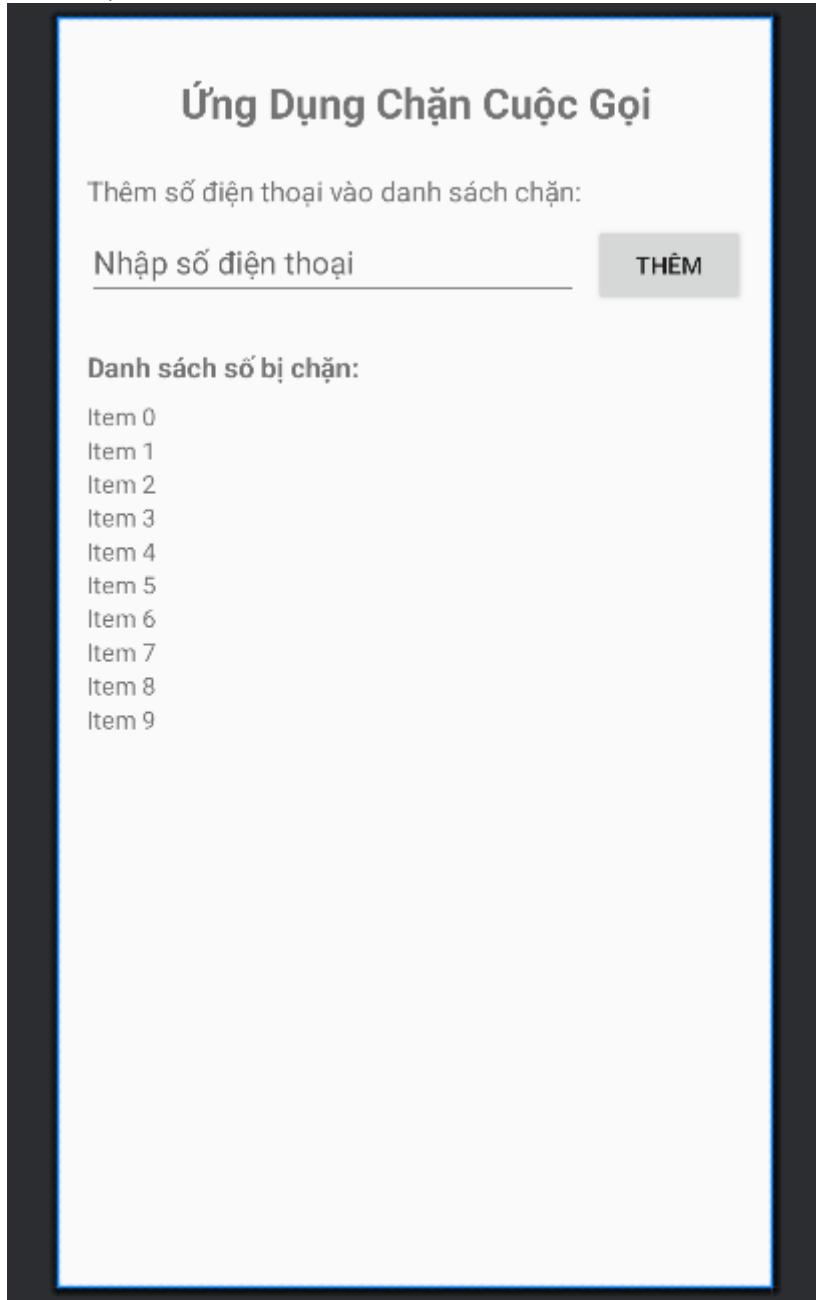
- Đăng ký receiver trong AndroidManifest.xml.

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.

Giao diện:



Thêm quyền trong AndroidManifest

```

<?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.telephony"
        android:required="false" />

    <uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.READ_CALL_LOG" />
    <uses-permission android:name="android.permission.ANSWER_PHONE_CALLS" />
    <uses-permission android:name="android.permission.CALL_PHONE" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"

```

Code hiển thị danh sách chặn

```

>MainActivity.kt    item_blocked_number.xml    BlockedNumbersAdapter.kt    activity_main.xml
1 package com.example.bai3_th5
2
3 import android.view.LayoutInflater
4 import android.view.View
5 import android.view.ViewGroup
6 import android.widget.ImageButton
7 import android.widget.TextView
8 import androidx.recyclerview.widget.RecyclerView
9
10 class BlockedNumbersAdapter(
11     private val blockedNumbers: List<String>,
12     private val onDeleteClick: (Int) -> Unit
13 ) : RecyclerView.Adapter<BlockedNumbersAdapter.ViewHolder>() {
14
15     class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
16         val textViewNumber: TextView = view.findViewById(R.id.textViewNumber)
17         val buttonDelete: ImageButton = view.findViewById(R.id.buttonDelete)
18     }
19
20     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
21         val view = LayoutInflater.from(parent.context)
22             .inflate(R.layout.item_blocked_number, parent, attachToRoot: false)
23         return ViewHolder(view)
24     }
25
26     override fun onBindViewHolder(holder: ViewHolder, position: Int) {
27         holder.textViewNumber.text = blockedNumbers[position]
28         holder.buttonDelete.setOnClickListener {
29             onDeleteClick(position)
30         }
31     }
32
33     override fun getItemCount() = blockedNumbers.size
34 }

```

Code file main

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

- File Structure:** The left sidebar shows the project structure under "Android". It includes the "app" module with "manifests", "kotlin/java" (containing "BlockedNumbersAdapter", "CallBlockService", "CallReceiver", and "MainActivity"), and "res" (containing "drawable" with "ic_block.xml", "ic_delete.xml", "ic_launcher_background.xml", and "ic_launcher_foreground.xml").
- Main Activity Code (MainActivity.kt):**

```

17 class MainActivity : AppCompatActivity() {
18     private lateinit var blockedNumbersAdapter: BlockedNumbersAdapter
19     private lateinit var blockedNumbersList: ArrayList<String>
20     private lateinit var inputNumber: EditText
21     private lateinit var addButton: Button
22
23     private val PERMISSIONS_REQUEST_CODE = 100
24     private val REQUIRED_PERMISSIONS = arrayOf(
25         Manifest.permission.READ_PHONE_STATE,
26         Manifest.permission.CALL_PHONE,
27         Manifest.permission.READ_CALL_LOG
28     )
29
30     // Thêm quyền ANSWER_PHONE_CALLS cho Android P trở lên
31     private val ADDITIONAL_PERMISSIONS = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
32         arrayOf(Manifest.permission.ANSWER_PHONE_CALLS)
33     } else {
34         emptyArray()
35     }
36
37     override fun onCreate(savedInstanceState: Bundle?) {
38         super.onCreate(savedInstanceState)
39         setContentView(R.layout.activity_main)
40
41         // Khởi tạo danh sách số bị chặn
42         blockedNumbersList = ArrayList()
43         loadBlockedNumbers()
44
45         // Khởi tạo RecyclerView
46         val recyclerView = findViewById<RecyclerView>(R.id.recyclerViewBlockedNumbers)
47         blockedNumbersAdapter = BlockedNumbersAdapter(blockedNumbersList) { position ->
48             removeBlockedNumber(position)
49         }
50     }

```
- Bottom Status Bar:** Shows the path "bal3_th5 > app > src > main > java > com > example > bal3_th5 > MainActivity > onCreate", the time "46:33 LF UTF-8", and the date "3/31/2025".
- Code Editor (MainActivity.kt):**

```

17 class MainActivity : AppCompatActivity() {
18     override fun onCreate(savedInstanceState: Bundle?) {
19         recyclerView.layoutManager = LinearLayoutManager(context, this)
20         recyclerView.adapter = blockedNumbersAdapter
21
22         // Khởi tạo các thành phần giao diện
23         inputNumber = findViewById(R.id.editTextPhoneNumber)
24         addButton = findViewById(R.id.buttonAddNumber)
25
26         addButton.setOnClickListener {
27             val number = inputNumber.text.toString().trim()
28             if (number.isNotEmpty()) {
29                 addBlockedNumber(number)
30                 inputNumber.text.clear()
31             } else {
32                 Toast.makeText(context, this, text: "Vui lòng nhập số điện thoại", Toast.LENGTH_SHORT).show()
33             }
34         }
35
36         // Kiểm tra và yêu cầu quyền
37         checkAndRequestPermissions()
38
39         // Khởi động dịch vụ
40         startCallBlockService()
41     }
42
43     private fun checkAndRequestPermissions() {
44         val permissionsToRequest = ArrayList<String>()
45
46         for (permission in REQUIRED_PERMISSIONS + ADDITIONAL_PERMISSIONS) {
47             if (ContextCompat.checkSelfPermission(context, this, permission) != PackageManager.PERMISSION_GRANTED) {
48                 permissionsToRequest.add(permission)
49             }
50         }
51     }

```

```

        if (permissionsToRequest.isNotEmpty()) {
            ActivityCompat.requestPermissions(
                activity: this,
                permissionsToRequest.toTypedArray(),
                PERMISSIONS_REQUEST_CODE
            )
        }
    }

    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        if (requestCode == PERMISSIONS_REQUEST_CODE) {
            var allGranted = true
            for (result in grantResults) {
                if (result != PackageManager.PERMISSION_GRANTED) {
                    allGranted = false
                    break
                }
            }

            if (allGranted) {
                Toast.makeText( context: this, text: "Tất cả quyền đã được cấp", Toast.LENGTH_SHORT).show()
                startCallBlockService()
            } else {
                Toast.makeText(
                    context: this,
                    text: "Một số quyền bị từ chối, ứng dụng có thể không hoạt động đúng",
                    Toast.LENGTH_LONG
                )
            }
        }
    }

    private fun startCallBlockService() {
        val serviceIntent = Intent( packageContext: this, CallBlockService::class.java)
        startService(serviceIntent)
    }

    private fun addBlockedNumber(number: String) {
        if (!blockedNumbersList.contains(number)) {
            blockedNumbersList.add(number)
            blockedNumbersAdapter.notifyItemInserted( position: blockedNumbersList.size - 1)
            saveBlockedNumbers()
            Toast.makeText( context: this, text: "Đã thêm số $number vào danh sách chặn", Toast.LENGTH_SHORT).show()
        } else {
            Toast.makeText( context: this, text: "Số này đã có trong danh sách chặn", Toast.LENGTH_SHORT).show()
        }
    }

    private fun removeBlockedNumber(position: Int) {
        if (position >= 0 && position < blockedNumbersList.size) {
            val number = blockedNumbersList[position]
            blockedNumbersList.removeAt(position)
            blockedNumbersAdapter.notifyItemRemoved(position)
            saveBlockedNumbers()
            Toast.makeText( context: this, text: "Đã xóa số $number khỏi danh sách chặn", Toast.LENGTH_SHORT).show()
        }
    }

    private fun saveBlockedNumbers() {
        val sharedPreferences = getSharedPreferences( name: "CallBlockerPrefs", MODE_PRIVATE)
        val editor = sharedPreferences.edit()
        val numbersSet = blockedNumbersList.toSet()
        editor.putStringSet( key: "blockedNumbers", numbersSet)
    }
}

```

```
        blockedNumbersAdapter.notifyItemRemoved(position)
        saveBlockedNumbers()
        Toast.makeText( context: this, text: "Đã xóa số $number khỏi danh sách chặn", Toast.LENGTH_SHORT).show()
    }
}

private fun saveBlockedNumbers() {
    val sharedpreferences = getSharedPreferences( name: "CallBlockerPrefs", MODE_PRIVATE)
    val editor = sharedpreferences.edit()
    val numbersSet = blockedNumbersList.toSet()
    editor.putStringSet( key: "blockedNumbers", numbersSet)
    editor.apply()
}

private fun loadBlockedNumbers() {
    val sharedpreferences = getSharedPreferences( name: "CallBlockerPrefs", MODE_PRIVATE)
    val numbersSet = sharedpreferences.getStringSet( key: "blockedNumbers", setOf())
    blockedNumbersList.clear()
    if (numbersSet != null) {
        blockedNumbersList.addAll(numbersSet)
    }
}
}
```

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.

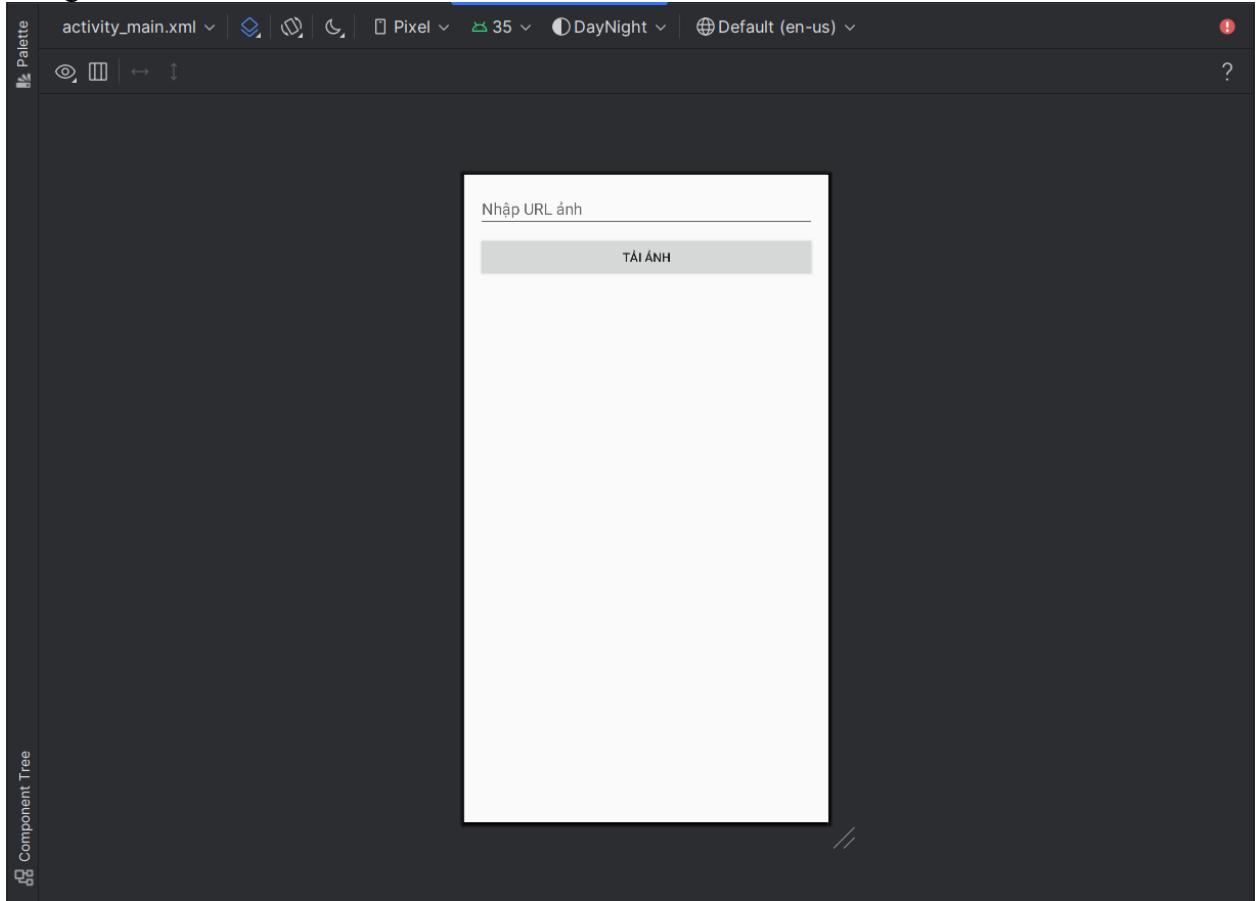
2. Tạo `AsyncTask`:

- 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)`.

Tạo giao diện



Code file main

```
class MainActivity : AppCompatActivity() {
    private lateinit var buttonLoad: Button
    private lateinit var imageView: ImageView
    private lateinit var progressBar: ProgressBar

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

        // Khởi tạo các thành phần UI
        editTextUrl = findViewById(R.id.editTextUrl)
        buttonLoad = findViewById(R.id.buttonLoad)
        imageView = findViewById(R.id.imageView)
        progressBar = findViewById(R.id.progressBar)

        // Xử lý sự kiện khi nhấn nút tải
        buttonLoad.setOnClickListener {
            val url = editTextUrl.text.toString().trim()
            if (url.isEmpty()) {
                Toast.makeText(context, text: "Vui lòng nhập URL ảnh", Toast.LENGTH_SHORT).show()
                return@setOnClickListener
            }

            // Khởi chạy AsyncTask để tải ảnh
            ImageDownloader().execute(url)
        }
    }
}
```

```

    // AsyncTask để tải ảnh từ internet
    private inner class ImageDownloader : AsyncTask<String, Int, Bitmap?>() {

        override fun onPreExecute() {
            // Hiển thị progress bar trước khi bắt đầu tải
            progressBar.visibility = View.VISIBLE
            progressBar.progress = 0
            imageView.setImageBitmap(null)
        }

        override fun doInBackground(vararg urls: String): Bitmap? {
            val imageUrl = urls[0]
            var bitmap: Bitmap? = null
            try {
                val url = URL(imageUrl)
                val connection = url.openConnection() as HttpURLConnection
                connection.doInput = true
                connection.connect()

                // Lấy kích thước file để tính toán tiến trình
                val contentLength = connection.contentLength
                val inputStream: InputStream = connection.inputStream
                val buffer = ByteArray( size: 1024 )
                var total = 0
                var count: Int

                // Tạo một ByteArrayOutputStream để lưu dữ liệu
                val data = java.io.ByteArrayOutputStream()

                // Đọc dữ liệu từ InputStream và cập nhật tiến trình
                while (inputStream.read(buffer).also { count = it } != -1) {
                    total += count
                }
                data.write(buffer, off: 0, count)
            }

            // Chuyển đổi dữ liệu đã tải thành Bitmap
            val imageData = data.toByteArray()
            bitmap = BitmapFactory.decodeByteArray(imageData, offset: 0, imageData.size)

            inputStream.close()
            data.close()
        } catch (e: Exception) {
            e.printStackTrace()
        }
        return bitmap
    }

        override fun onProgressUpdate(vararg values: Int?) {
            // Cập nhật tiến trình trên progress bar
            values[0]?.let {
                progressBar.progress = it
            }
        }

        override fun onPostExecute(result: Bitmap?) {
            //Ẩn progress bar và hiển thị ảnh khi tải xong
            progressBar.visibility = View.GONE

            if (result != null) {
                imageView.setImageBitmap(result)
                Toast.makeText(applicationContext, text: "Tải ảnh thành công", Toast.LENGTH_SHORT).show()
            }
        }
    }
}

```

```

    override fun onPostExecute(result: Bitmap?) {
        // Ẩn progress bar và hiển thị ảnh khi tải xong
        progressBar.visibility = View.GONE

        if (result != null) {
            imageView.setImageBitmap(result)
            Toast.makeText(applicationContext, text: "Tải ảnh thành công", Toast.LENGTH_SHORT).show()
        } else {
            Toast.makeText(applicationContext, text: "Không thể tải ảnh. Kiểm tra URL và kết nối mạng", Toast.LENGTH_SHORT)
        }
    }
}

```

Cấp quyền internet:

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools">
4          <uses-permission android:name="android.permission.INTERNET" />
5          <application>
6              android:allowBackup="true"
7              android:dataExtractionRules="@xml/data_extraction_rules"
8              android:fullBackupContent="@xml/backup_rules"
9              android:icon="@mipmap/ic_launcher"
10             android:label="@string/app_name"
11             android:roundIcon="@mipmap/ic_launcher_round"
12             android:supportsRtl="true"
13             android:theme="@style/Theme.AppCompat.DayNight"
14             tools:targetApi="31">
15                 <activity>
16                     android:name=".MainActivity"
17                     android:exported="true"
18                     android:label="@string/app_name"
19                     android:theme="@style/Theme.AppCompat.DayNight">
20                         <intent-filter>
21                             <action android:name="android.intent.action.MAIN" />
22
23                             <category android:name="android.intent.category.LAUNCHER" />
24                         </intent-filter>
25                     </activity>
26                 </application>
27
28 </manifest>

```

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.

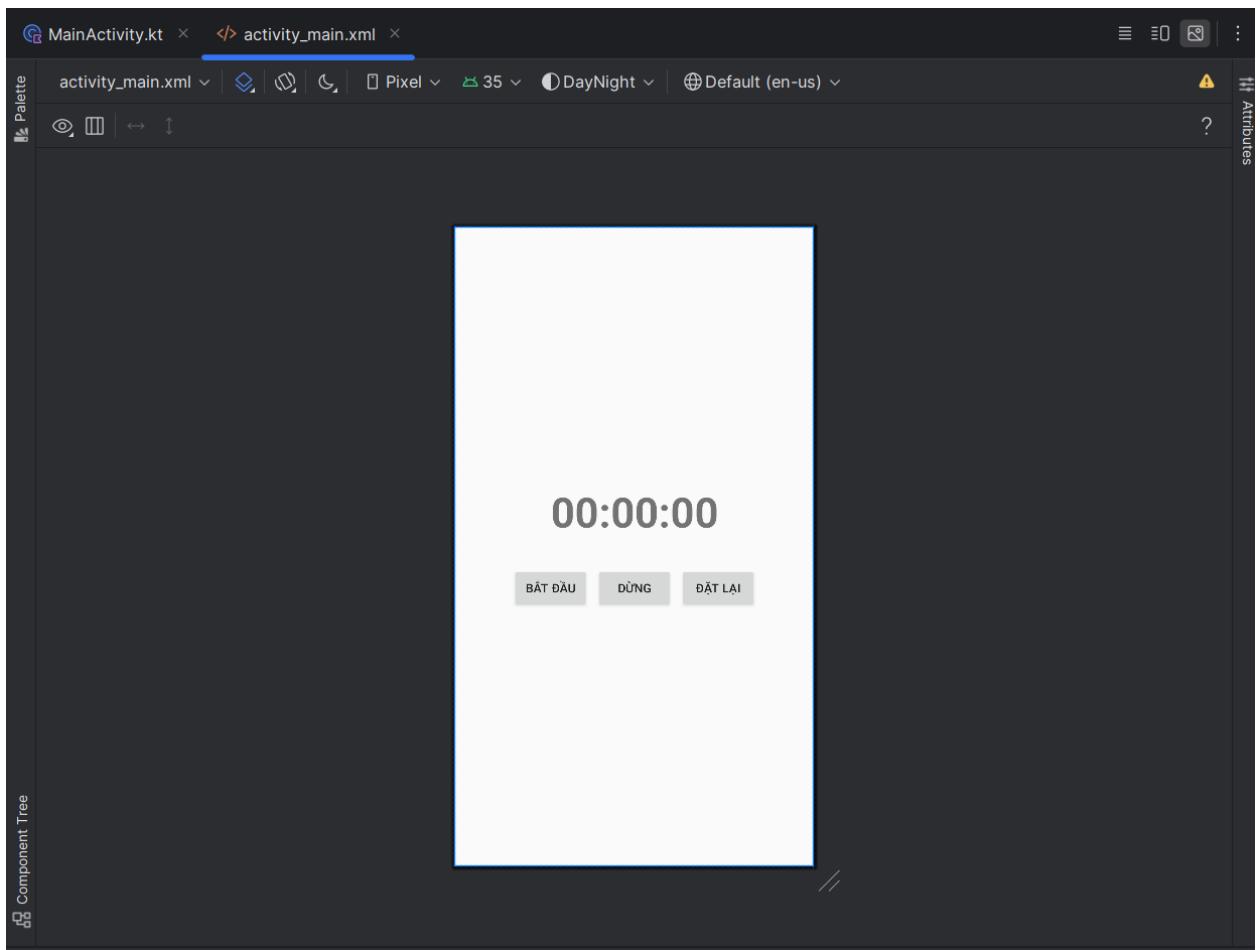
Các bước thực hiện:

1. **Thiết kế giao diện:**
 - o Một TextView để hiển thị thời gian.
2. **Tạo Handler:**
 - o Tạo một Handler trong Activity chính.
 - o Override phương thức `handleMessage()` (nếu dùng Message) hoặc tạo một Runnable.
3. **Tạo Thread:**
 - o Tạo một Thread mới.
 - o 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:**
 - o 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.

Tạo giao diện



Code MainActivity

```
class MainActivity : AppCompatActivity() {

    private lateinit var timeTextView: TextView
    private lateinit var startButton: Button
    private lateinit var stopButton: Button

    private val handler = Handler(Looper.getMainLooper())
    private var seconds = 0
    private var minutes = 0
    private var hours = 0
    private var isRunning = AtomicBoolean(initialValue: false)
    private var timerThread: Thread? = null

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

        // Ánh xạ các view
        timeTextView = findViewById(R.id.timeTextView)
        startButton = findViewById(R.id.startButton)
        stopButton = findViewById(R.id.stopButton)

        // Thiết lập sự kiện click cho nút Bắt đầu
        startButton.setOnClickListener {
            startTimer()
        }

        // Thiết lập sự kiện click cho nút Dừng
        stopButton.setOnClickListener {
            stopTimer()
        }
    }
}
```

```
    // Thiết lập sự kiện click cho nút Đặt lại
    resetButton.setOnClickListener {
        resetTimer()
    }

    private fun startTimer() {
        if (isRunning.get()) return

        isRunning.set(true)

        timerThread = Thread {
            try {
                while (isRunning.get()) {
                    // Cập nhật UI thông qua Handler
                    handler.post {
                        updateTimerText()
                    }

                    // Tăng thời gian
                    seconds++
                    if (seconds >= 60) {
                        seconds = 0
                        minutes++
                        if (minutes >= 60) {
                            minutes = 0
                            hours++
                        }
                    }
                }
            } catch (e: Exception) {
                e.printStackTrace()
            }
        }
        timerThread?.start()
    }

    private fun stopTimer() {
        isRunning.set(false)
        timerThread?.interrupt()
    }

    private fun resetTimer() {
        isRunning.set(false)
        timerThread?.interrupt()
        seconds = 0
        minutes = 0
        hours = 0
        updateTimerText()
    }

    private fun updateTimerText() {
        timeTextView.text = String.format("%02d:%02d:%02d", hours, minutes, seconds)
    }
}
```

```

        // Delay 1 giây
        Thread.sleep( millis: 1000 )
    }
} catch (e: InterruptedException) {
    // Thread bị interrupt, dừng timer
    isRunning.set(false)
}
}

timerThread?.start()
}

private fun stopTimer() {
    isRunning.set(false)
    timerThread?.interrupt()
    timerThread = null
}

private fun resetTimer() {
    // Dừng timer nếu đang chạy
    stopTimer()

    // Đặt lại các biến thời gian
    seconds = 0
    minutes = 0
    hours = 0

    // Cập nhật giao diện
    updateTimeText()
}

private fun updateTimeText() {
    val timeString = String.format("%02d:%02d:%02d", hours, minutes, seconds)
    timeTextView.text = timeString
}

override fun onDestroy() {
    super.onDestroy()
    stopTimer()
}

```

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:

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

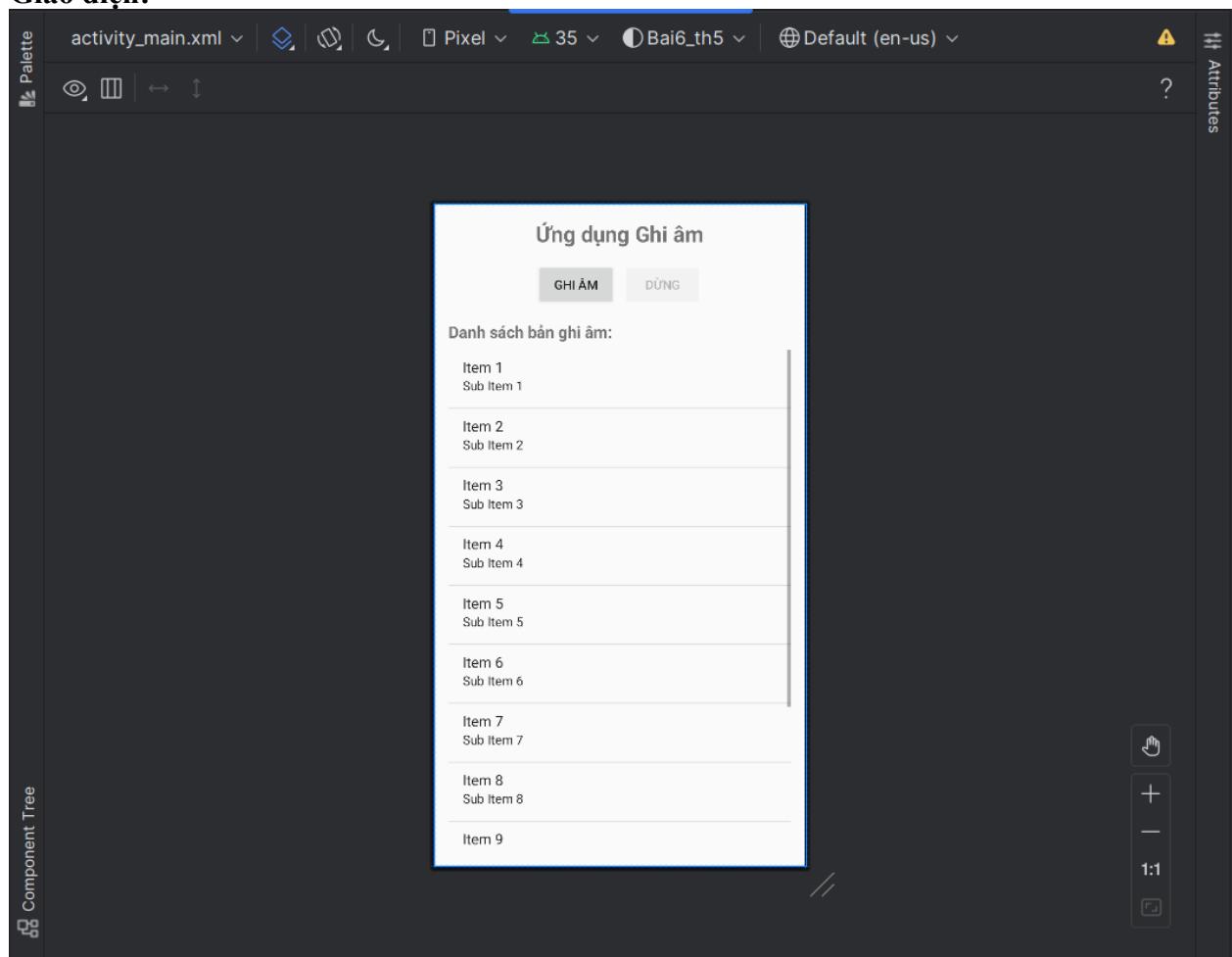
2. Phát lại:

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

3. Hiển thị danh sách file ghi âm:

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

Giao diện:



MainActivity:

```
class MainActivity : AppCompatActivity() {

    private val PERMISSION_REQUEST_CODE = 200

    // UI components
    private lateinit var recordButton: Button
    private lateinit var stopButton: Button
    private lateinit var playButton: Button
    private lateinit var pauseButton: Button
    private lateinit var recordingsListView: ListView
    private lateinit var nowPlayingText: TextView

    // Media components
    private var mediaRecorder: MediaRecorder? = null
    private var mediaPlayer: MediaPlayer? = null

    // State variables
    private var isRecording = false
    private var isPlaying = false
    private var recordingFile: File? = null
    private var recordingUri: Uri? = null
    private var selectedRecordingUri: Uri? = null
    private var selectedRecordingName: String? = null

    // Recordings list
    private val recordingsList = ArrayList<HashMap<String, String>>()
    private val dateFormat = SimpleDateFormat(pattern: "dd/MM/yyyy HH:mm:ss", Locale.getDefault())

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

```

        setContentView(R.layout.activity_main)

        // Initialize UI components
        recordButton = findViewById(R.id.recordButton)
        stopButton = findViewById(R.id.stopButton)
        playButton = findViewById(R.id.playButton)
        pauseButton = findViewById(R.id.pauseButton)
        recordingsListView = findViewById(R.id.recordingsListView)
        nowPlayingText = findViewById(R.id.nowPlayingText)

        // Check and request permissions
        if (!checkPermissions()) {
            requestPermissions()
        }

        // Set up recordings list adapter
        val adapter = SimpleAdapter(
            context: this,
            recordingsList,
            android.R.layout.simple_list_item_2,
            arrayOf("name", "date"),
            intArrayOf(android.R.id.text1, android.R.id.text2)
        )
        recordingsListView.adapter = adapter

        // Load existing recordings
        loadRecordings()
        recordButton.setOnClickListener {
            if (checkPermissions()) {
                startRecording()
            } else {
                requestPermissions()
            }
        }

        stopButton.setOnClickListener {
            stopRecording()
            loadRecordings() // Refresh the list
        }

        playButton.setOnClickListener {
            if (selectedRecordingUri != null) {
                playRecording()
            } else {
                Toast.makeText(context: this, text: "Chọn một bản ghi để phát", Toast.LENGTH_SHORT).show()
            }
        }

        pauseButton.setOnClickListener {
            pausePlayback()
        }

        // Set up list item click listener
        recordingsListView.setOnItemClickListener { _, _, position, _ -
            val item = recordingsList[position]

```

```

        selectedRecordingName = item["name"]
        selectedRecordingUri = Uri.parse(item["uri"])

        nowPlayingText.text = "Đang phát: ${selectedRecordingName}"
        nowPlayingText.visibility = View.VISIBLE
        playButton.isEnabled = true

        // Stop current playback if playing
        if (isPlaying) {
            stopPlayback()
        }
    }

private fun checkPermissions(): Boolean {
    val recordPermission = ContextCompat.checkSelfPermission(context, Manifest.permission.RECORD_AUDIO)
    val storagePermission = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
        PackageManager.PERMISSION_GRANTED // Android 10+ uses scoped storage
    } else {
        ContextCompat.checkSelfPermission(context, Manifest.permission.WRITE_EXTERNAL_STORAGE)
    }

    return recordPermission == PackageManager.PERMISSION_GRANTED &&
           storagePermission == PackageManager.PERMISSION_GRANTED
}

private fun requestPermissions() {
private fun requestPermissions() {
    val permissions = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
        arrayOf(Manifest.permission.RECORD_AUDIO)
    } else {
        arrayOf(Manifest.permission.RECORD_AUDIO, Manifest.permission.WRITE_EXTERNAL_STORAGE)
    }

    ActivityCompat.requestPermissions(activity, permissions, PERMISSION_REQUEST_CODE)
}

private fun startRecording() {
    if (isRecording) return

    try {
        // Initialize MediaRecorder
        mediaRecorder = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
            MediaRecorder(context)
        } else {
            MediaRecorder()
        }

        mediaRecorder?.apply {
            setAudioSource(MediaRecorder.AudioSource.MIC)
            setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP)
            setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB)

            // Create filename based on timestamp
            val timestamp = SimpleDateFormat(pattern: "yyyyMMdd_HHmmss", Locale.getDefault()).format(D

```

```

    val fileName = "Recording_$timestamp.3gp"

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
        // Use MediaStore on Android 10+
        val contentValues = ContentValues().apply {
            put(MediaStore.Audio.Media.DISPLAY_NAME, fileName)
            put(MediaStore.Audio.Media.MIME_TYPE, "audio/3gpp")
            put(MediaStore.Audio.Media.RELATIVE_PATH, Environment.DIRECTORY_MUSIC + "/Recordings")
        }

        val contentResolver = applicationContext.contentResolver
        recordingUri = contentResolver.insert(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, contentValues)

        recordingUri?.let {
            setOutputFile(contentResolver.openFileDescriptor(it, mode: "w")?.fileDescriptor)
        }
    } else {
        // Use direct file on Android 9 and below
        val musicDir = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_MUSIC)
        val recordingsDir = File(musicDir, child: "Recordings")

        if (!recordingsDir.exists()) {
            recordingsDir.mkdirs()
        }

        recordingFile = File(recordingsDir, fileName)
        setOutputFile(recordingFile?.absolutePath)
    }

    try {
        prepare()
        start()
        isRecording = true

        // Update UI
        recordButton.isEnabled = false
        stopButton.isEnabled = true

        Toast.makeText(context: this@MainActivity, text: "Đang ghi âm...", Toast.LENGTH_SHORT)
    } catch (e: IOException) {
        e.printStackTrace()
        Toast.makeText(context: this@MainActivity, text: "Lỗi khi ghi âm: ${e.message}", Toast.LENGTH_SHORT)
    }
}

} catch (e: Exception) {
    e.printStackTrace()
    Toast.makeText(context: this, text: "Lỗi khi khởi tạo ghi âm: ${e.message}", Toast.LENGTH_SHORT)
}
}

private fun stopRecording() {
    if (!isRecording) return

    try {
        mediaRecorder?.apply {
            stop()
        }
    }
}

```

```

        reset()
        release()
    }
    mediaRecorder = null
    isRecording = false

    // Update UI
    recordButton.isEnabled = true
    stopButton.isEnabled = false

    Toast.makeText( context: this, text: "Đã lưu bản ghi âm", Toast.LENGTH_SHORT).show()
} catch (e: Exception) {
    e.printStackTrace()
    Toast.makeText( context: this, text: "Lỗi khi dừng ghi âm: ${e.message}", Toast.LENGTH_SHORT).sh
}
}

private fun loadRecordings() {
    recordingsList.clear()

    val projection = arrayOf(
        MediaStore.Audio.Media._ID,
        MediaStore.Audio.Media.DISPLAY_NAME,
        MediaStore.Audio.Media.DATE_ADDED
    )

    // Only get .3gp files in the Recordings folder
    // Only get .3gp files in the Recordings folder
    val selection = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
        "${MediaStore.Audio.Media.RELATIVE_PATH} LIKE ? AND ${MediaStore.Audio.Media.DISPLAY_NAME} LI
    } else {
        "${MediaStore.Audio.Media.DATA} LIKE ? AND ${MediaStore.Audio.Media.DISPLAY_NAME} LIKE ?"
    }

    val selectionArgs = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
        arrayOf("%Recordings%", "%Recording_%.3gp")
    } else {
        arrayOf("%/Music/Recordings%", "%Recording_%.3gp")
    }

    val sortOrder = "${MediaStore.Audio.Media.DATE_ADDED} DESC"

    contentResolver.query(
        MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
        projection,
        selection,
        selectionArgs,
        sortOrder
    )?.use { cursor ->
        val idColumn = cursor.getColumnIndexOrThrow(MediaStore.Audio.Media._ID)
        val nameColumn = cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.DISPLAY_NAME)
        val dateColumn = cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.DATE_ADDED)

        while (cursor.moveToNext()) {
            val id = cursor.getLong(idColumn)

```

```

        val name = cursor.getString(nameColumn)
        val dateAdded = cursor.getLong(dateColumn)

        val contentUri = Uri.withAppendedPath(
            MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
            id.toString()
        )

        val recording = HashMap<String, String>()
        recording["name"] = name
        recording["date"] = "Ngày tạo: ${dateFormat.format(Date(date: dateAdded * 1000))}"
        recording["uri"] = contentUri.toString()

        recordingsList.add(recording)
    }
}

(recordingsListView.adapter as SimpleAdapter).notifyDataSetChanged()
}

private fun playRecording() {
    if (isPlaying) {
        stopPlayback()
    }

    try {
        mediaPlayer = MediaPlayer().apply {
            setDataSource(applicationContext, selectedRecordingUri!!)
            prepare()
            start()

            this@MainActivity.isPlaying = true
            playButton.isEnabled = false
            pauseButton.isEnabled = true

            // When playback completes
            setOnCompletionListener {
                stopPlayback()
            }
        }
    } catch (e: Exception) {
        e.printStackTrace()
        Toast.makeText(context: this, text: "Lỗi khi phát: ${e.message}", Toast.LENGTH_SHORT).show()
    }
}

private fun pausePlayback() {
    if (!isPlaying) return

    mediaPlayer?.pause()
    isPlaying = false
    playButton.isEnabled = true
    pauseButton.isEnabled = false
}

```

```
private fun stopPlayback() {
    mediaPlayer?.apply {
        stop()
        reset()
        release()
    }
    mediaPlayer = null
    isPlaying = false
    playButton.isEnabled = true
    pauseButton.isEnabled = false
}

override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)

    if (requestCode == PERMISSION_REQUEST_CODE) {
        if (grantResults.isNotEmpty() && grantResults.all { it == PackageManager.PERMISSION_GRANTED }) {
            Toast.makeText(context, text: "Đã cấp quyền", Toast.LENGTH_SHORT).show()
            loadRecordings()
        } else {
            Toast.makeText(context, text: "Cần cấp quyền để sử dụng ứng dụng", Toast.LENGTH_SHORT)
        }
    }
}

override fun onDestroy() {
    super.onDestroy()

    // Release resources
    mediaRecorder?.apply {
        if (isRecording) {
            stop()
        }
        release()
    }

    mediaPlayer?.apply {
        if (isPlaying) {
            stop()
        }
        release()
    }
}
}
```

Thêm quyền

```

<uses-tools="http://schemas.android.com/tools">
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    android:maxSdkVersion="29" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
    android:maxSdkVersion="32" />

<application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"

```

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.

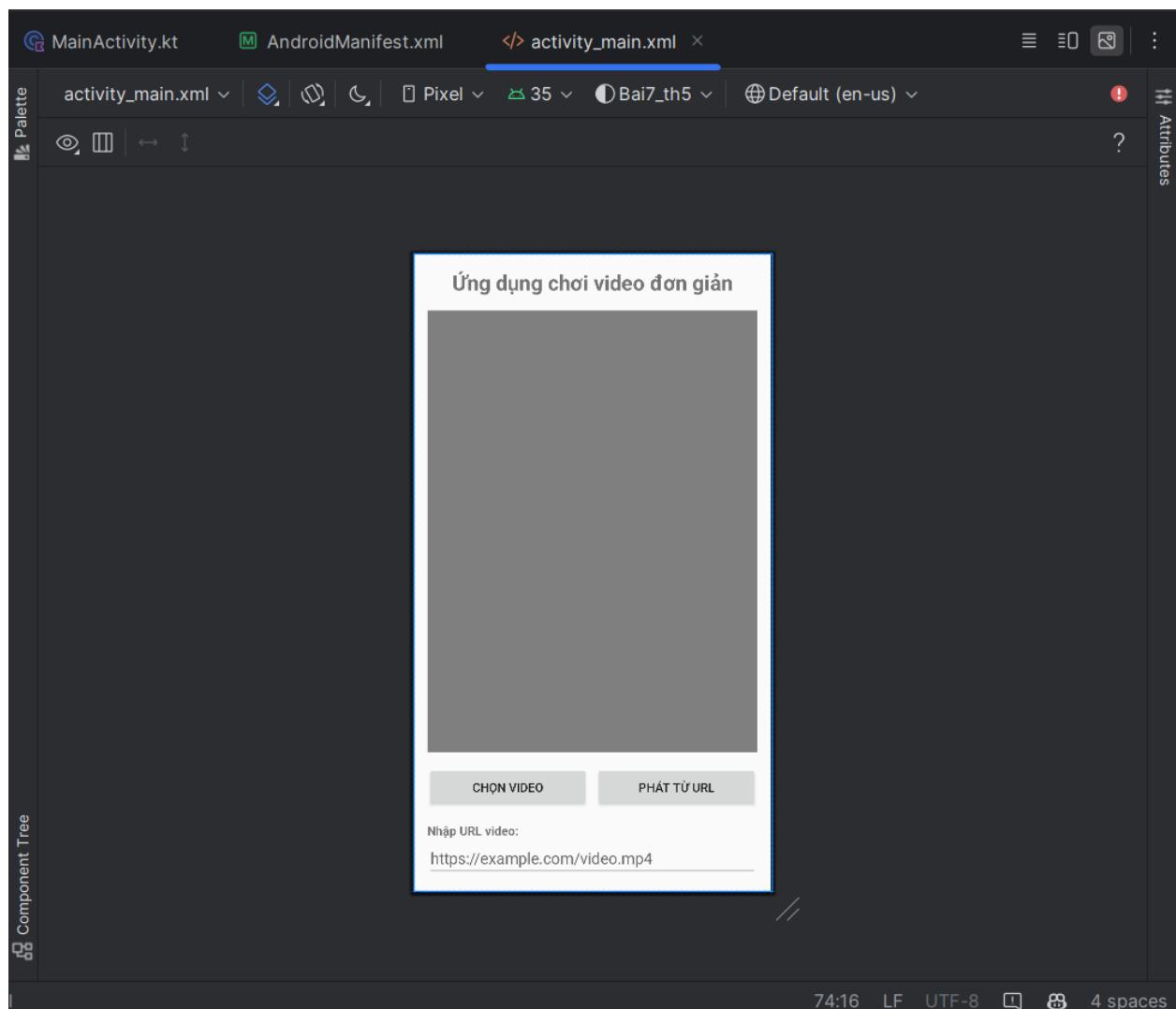
Các bước thực hiện:

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

Giao diện:



MainActivity:

```
class MainActivity : AppCompatActivity() {

    companion object {
        private const val PERMISSION_REQUEST_CODE = 100
        private const val VIDEO_PICK_CODE = 1000
    }

    private lateinit var videoView: VideoView
    private lateinit var mediaController: MediaController
    private lateinit var btnChooseVideo: Button
    private lateinit var btnPlayUrl: Button
    private lateinit var etVideoUrl: EditText

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

        // Ánh xạ các view
        videoView = findViewById(R.id.videoView)
        btnChooseVideo = findViewById(R.id.btnChooseVideo)
        btnPlayUrl = findViewById(R.id.btnPlayUrl)
        etVideoUrl = findViewById(R.id.etVideoUrl)

        // Thiết lập MediaController
        mediaController = MediaController(context)
        mediaController.setAnchorView(videoView)
        videoView.setMediaController(mediaController)
    }
}
```

```
// Thiết lập sự kiện hoàn thành
videoView.setOnCompletionListener {
    Toast.makeText(context: this, text: "Video đã phát xong", Toast.LENGTH_SHORT).show()
}

// Thiết lập sự kiện lỗi
videoView.setOnErrorListener { _, what, extra ->
    val errorMessage = when (what) {
        MediaPlayer.MEDIA_ERROR_UNKNOWN -> "Lỗi không xác định"
        MediaPlayer.MEDIA_ERROR_SERVER_DIED -> "Lỗi máy chủ"
        else -> "Lỗi không xác định: $what, $extra"
    }
    Toast.makeText(context: this, text: "Lỗi khi phát video: $errorMessage", Toast.LENGTH_LONG).show()
    true
}

// Nút chọn video từ thiết bị
btnChooseVideo.setOnClickListener {
    if (checkPermissions()) {
        pickVideoFromGallery()
    } else {
        requestPermissions()
    }
}

// Nút phát video từ URL
btnPlayUrl.setOnClickListener {
    val videoUrl = etVideoUrl.text.toString().trim()
```

```

        if (videoUrl.isEmpty()) {
            Toast.makeText(context: this, text: "Vui lòng nhập URL video", Toast.LENGTH_SHORT).show()
            return@setOnClickListener
        }

        playVideoFromUrl(videoUrl)
    }

    // Thêm một số URL video mẫu
    etVideoUrl.setText("https://commondatastorage.googleapis.com/gtv-videos-bucket/sample/BigBuckBunny.mp4")
}

private fun checkPermissions(): Boolean {
    return if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
        true // Android 10+ không cần quyền đọc bộ nhớ ngoài để chọn media
    } else {
        ContextCompat.checkSelfPermission(
            context: this,
            Manifest.permission.READ_EXTERNAL_STORAGE
        ) == PackageManager.PERMISSION_GRANTED
    }
}

private fun requestPermissions() {
    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.Q) {
        ActivityCompat.requestPermissions(
            activity: this,
            arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE),
            PERMISSION_REQUEST_CODE
        )
    }
}

private fun pickVideoFromGallery() {
    val intent = Intent(Intent.ACTION_PICK)
    intent.type = "video/*"
    startActivityForResult(intent, VIDEO_PICK_CODE)
}

private fun playVideoFromUrl(url: String) {
    try {
        // Đặt URI cho VideoView
        videoView.setVideoURI(Uri.parse(url))

        // Hiển thị dialog đang tải
        val loadingDialog = AlertDialog.Builder(context: this)
            .setMessage("Đang tải video...")
            .setCancelable(false)
            .create()
        loadingDialog.show()

        // Thiết lập sự kiện chuẩn bị
        videoView.setOnPreparedListener { mp -
            // Đóng dialog khi video đã sẵn sàng phát
        }
    }
}

```

```

        loadingDialog.dismiss()

        // Bắt đầu phát video
        videoView.start()

        // Thiết lập các thuộc tính cho MediaPlayer
        mp.setOnVideoSizeChangedListener { _, _, _ ->
            // Hiển thị MediaController khi video bắt đầu phát
            mediaController.show( timeout: 0 )
        }
    }

} catch (e: Exception) {
    e.printStackTrace()
    Toast.makeText( context: this, text: "Lỗi khi phát video: ${e.message}", Toast.LENGTH_LONG).show()
}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    if (requestCode == VIDEO_PICK_CODE && resultCode == Activity.RESULT_OK && data != null) {
        val selectedVideoUri = data.data
        selectedVideoUri?.let {
            // Lấy tên file video
            val videoName = getVideoName(it)
            Toast.makeText( context: this, text: "Đã chọn video: $videoName", Toast.LENGTH_SHORT).show()

            // videoView.setVideoURI(it)
            videoView.start()
        }
    }
}

private fun getVideoName(uri: Uri): String {
    var videoName = "Video không xác định"
    val projection = arrayOf(MediaStore.Video.Media.DISPLAY_NAME)

    contentResolver.query(uri, projection, selection: null, selectionArgs: null, sortOrder: null)? .use { cu
        val nameIndex = cursor.getColumnIndexOrThrow(MediaStore.Video.Media.DISPLAY_NAME)
        if (cursor.moveToFirst()) {
            videoName = cursor.getString(nameIndex)
        }
    }

    return videoName
}

override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
}

```

```
        if (requestCode == PERMISSION_REQUEST_CODE) {
            if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(context, text: "Đã cấp quyền truy cập", Toast.LENGTH_SHORT).show()
                pickVideoFromGallery()
            } else {
                Toast.makeText(context, text: "Cần quyền truy cập để chọn video", Toast.LENGTH_SHORT)
            }
        }
    }

    override fun onPause() {
        super.onPause()
        // Tạm dừng video khi Activity bị tạm dừng
        if (videoView.isPlaying) {
            videoView.pause()
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        // Giải phóng tài nguyên
        videoView.stopPlayback()
    }
}
```

Thêm quyền

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
    android:maxSdkVersion="32" />

<application
    android:allowBackup="true"
```

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.

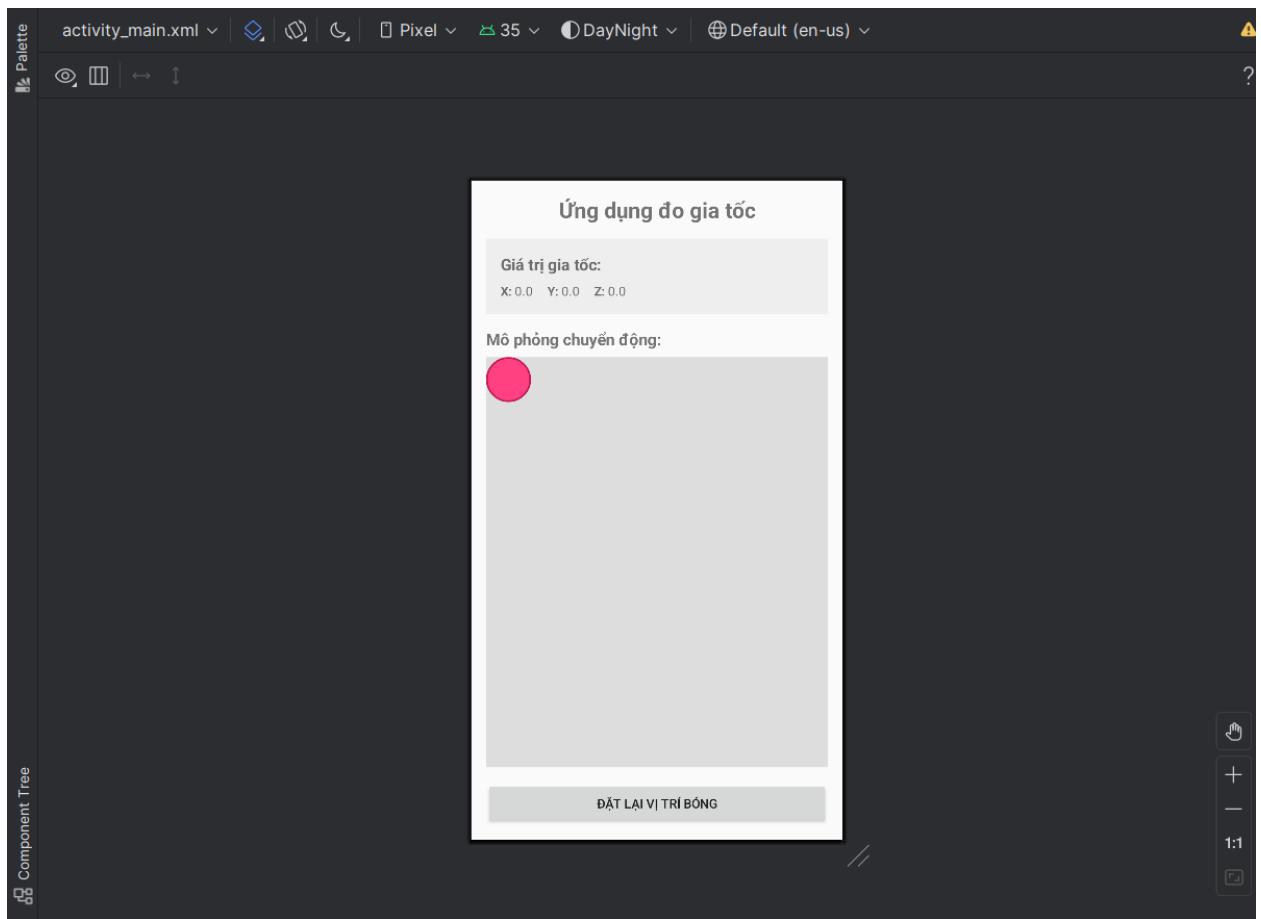
3. Hiển thị giá trị gia tốc:

- 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.

Giao diện:



MainActivity:

```
class MainActivity : AppCompatActivity(), SensorEventListener {

    // Khai báo các thành phần UI
    private lateinit var tvX: TextView
    private lateinit var tvY: TextView
    private lateinit var tvZ: TextView
    private lateinit var ballImage: ImageView
    private lateinit var ballContainer: FrameLayout
    private lateinit var btnReset: Button

    // Khai báo các thành phần liên quan đến cảm biến
    private lateinit var sensorManager: SensorManager
    private var accelerometer: Sensor? = null

    // Các biến cho mô phỏng chuyển động
    private var ballX = 0f
    private var ballY = 0f
    private var lastX = 0f
    private var lastY = 0f
    private var speedX = 0f
    private var speedy = 0f
    private val damping = 0.9f // Hệ số giảm tốc (ma sát)
    private val sensitivity = 1.0f // Độ nhạy của chuyển động

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

```

// Ánh xạ các view
_tvX = findViewById(R.id.tvX)
_tvY = findViewById(R.id.tvY)
_tvZ = findViewById(R.id.tvZ)
_ballImage = findViewById(R.id.ballImage)
_ballContainer = findViewById(R.id.ballContainer)
_btnReset = findViewById(R.id.btnReset)

// Khởi tạo SensorManager và lấy cảm biến gia tốc
_sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
_accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)

// Kiểm tra xem thiết bị có cảm biến gia tốc không
if (accelerometer == null) {
    tvX.text = "Thiết bị không có cảm biến gia tốc"
    tvY.text = ""
    tvZ.text = ""
}

// Đặt lại vị trí bóng khi nhấn nút Reset
btnReset.setOnClickListener {
    resetBallPosition()
}

// Thiết lập sự kiện layout cho container để lấy kích thước thực
ballContainer.post {
    resetBallPosition()
}
}

private fun resetBallPosition() {
    // Đặt bóng ở giữa container
    ballX = (ballContainer.width - ballImage.width) / 2f
    ballY = (ballContainer.height - ballImage.height) / 2f
    updateBallPosition()

    // Đặt lại vận tốc
    speedX = 0f
    speedY = 0f
}

private fun updateBallPosition() {
    // Cập nhật vị trí của bóng
    ballImage.x = ballX
    ballImage.y = ballY
}

override fun onResume() {
    super.onResume()
    // Đăng ký lắng nghe sự kiện cảm biến khi Activity được hiển thị
    accelerometer?.let {
        sensorManager.registerListener(listener: this, it, SensorManager.SENSOR_DELAY_GAME)
    }
}

override fun onPause() {
    super.onPause()
    // Hủy đăng ký lắng nghe sự kiện cảm biến khi Activity bị tạm dừng
    sensorManager.unregisterListener(listener: this)
}

```

```

override fun onSensorChanged(event: SensorEvent) {
    if (event.sensor.type == Sensor.TYPE_ACCELEROMETER) {
        // Lấy giá trị gia tốc từ sự kiện
        val x = event.values[0]
        val y = event.values[1]
        val z = event.values[2]

        // Hiển thị giá trị gia tốc lên TextView
        tvX.text = String.format("%.2f", x)
        tvY.text = String.format("%.2f", y)
        tvZ.text = String.format("%.2f", z)

        // Tính toán chuyển động của bóng
        // Lưu ý: trục Y của cảm biến tương ứng với trục X trên màn hình
        // và trục X của cảm biến tương ứng với trục Y trên màn hình (đảo ngược)

        // Cập nhật vận tốc dựa trên gia tốc
        speedX -= y * sensitivity
        speedY += x * sensitivity

        // Áp dụng ma sát để giảm dần vận tốc
        speedX *= damping
        speedY *= damping

        // Cập nhật vị trí dựa trên vận tốc
        ballX += speedX
        ballY += speedY

        // Kiểm tra va chạm với viền container
        val maxX = ballContainer.width - ballImage.width.toFloat()
        val maxY = ballContainer.height - ballImage.height.toFloat()

        // Xử lý va chạm với viền trái/phải
        if (ballX < 0) {
            ballX = 0f
            speedX = -speedX * 0.7f // Phản xạ và giảm năng lượng
        } else if (ballX > maxX) {
            ballX = maxX
            speedX = -speedX * 0.7f // Phản xạ và giảm năng lượng
        }

        // Xử lý va chạm với viền trên/dưới
        if (ballY < 0) {
            ballY = 0f
            speedY = -speedY * 0.7f // Phản xạ và giảm năng lượng
        } else if (ballY > maxY) {
            ballY = maxY
            speedY = -speedY * 0.7f // Phản xạ và giảm năng lượng
        }

        // Cập nhật vị trí bóng trên giao diện
        updateBallPosition()
    }
}

```

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.

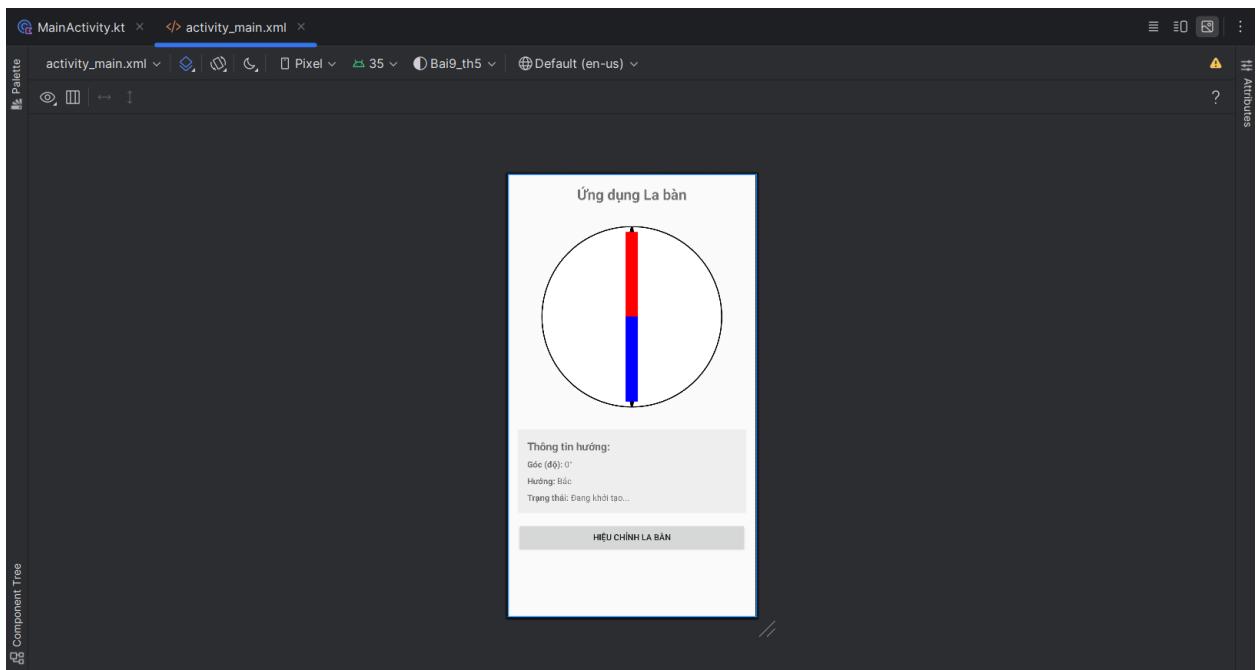
4. Hiển thị góc lệch:

- 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.

Giao diện:



MainActivity:

```
class MainActivity : AppCompatActivity(), SensorEventListener {  
  
    // Khai báo các thành phần UI  
    private lateinit var compassNeedle: ImageView  
    private lateinit var tvDegree: TextView  
    private lateinit var tvDirection: TextView  
    private lateinit var tvStatus: TextView  
    private lateinit var btnCalibrate: Button  
  
    // Khai báo các thành phần liên quan đến cảm biến  
    private lateinit var sensorManager: SensorManager  
    private var accelerometer: Sensor? = null  
    private var magnetometer: Sensor? = null  
  
    // Mảng lưu giá trị cảm biến  
    private val accelerometerReading = FloatArray(size: 3)  
    private val magnetometerReading = FloatArray(size: 3)  
  
    // Mảng ma trận để tính toán hướng  
    private val rotationMatrix = FloatArray(size: 9)  
    private val orientationAngles = FloatArray(size: 3)  
  
    // Biến lưu góc hiện tại và góc trước đó  
    private var currentDegree = 0f  
  
    // Biến kiểm tra trạng thái cảm biến  
    private var hasAccelerometerData = false  
    private var hasMagnetometerData = false
```

```

// Hàng số cho các hướng
private val directions = arrayOf("Bắc", "Đông Bắc", "Đông", "Đông Nam",
    "Nam", "Tây Nam", "Tây", "Tây Bắc", "Bắc")

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

    // Ánh xạ các view
    compassNeedle = findViewById(R.id.compassNeedle)
    tvDegree = findViewById(R.id.tvDegree)
    tvDirection = findViewById(R.id.tvDirection)
    tvStatus = findViewById(R.id.tvStatus)
    btnCalibrate = findViewById(R.id.btnCalibrate)

    // Khởi tạo SensorManager và lấy các cảm biến
    sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
    accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
    magnetometer = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)

    // Kiểm tra xem thiết bị có cảm biến cảm ứng không
    if (accelerometer == null || magnetometer == null) {
        tvStatus.text = "Thiết bị không hỗ trợ cảm biến cảm ứng"
        Toast.makeText(context, text: "Thiết bị không hỗ trợ la bàn", Toast.LENGTH_LONG).show()
    } else {
        tvStatus.text = "Sẵn sàng"
    }

    // Thiết lập sự kiện nút hiệu chỉnh
    btnCalibrate.setOnClickListener {
        calibrateCompass()
    }
}

override fun onResume() {
    super.onResume()

    // Đăng ký lắng nghe sự kiện cảm biến khi Activity được hiển thị
    accelerometer?.let {
        sensorManager.registerListener(listener: this, it, SensorManager.SENSOR_DELAY_NORMAL)
    }
    magnetometer?.let {
        sensorManager.registerListener(listener: this, it, SensorManager.SENSOR_DELAY_NORMAL)
    }
}

override fun onPause() {
    super.onPause()

    // Hủy đăng ký lắng nghe sự kiện cảm biến khi Activity bị tạm dừng
    sensorManager.unregisterListener(listener: this)
}

override fun onSensorChanged(event: SensorEvent) {
    // Xử lý dữ liệu từ cảm biến gia tốc
    if (event.sensor.type == Sensor.TYPE_ACCELEROMETER) {
        System.arraycopy(event.values, srcPos: 0, accelerometerReading, destPos: 0, accelerometerReading.size)
    }
}

```

```

        hasAccelerometerData = true
    }
    // Xử lý dữ liệu từ cảm biến từ trường
    else if (event.sensor.type == Sensor.TYPE_MAGNETIC_FIELD) {
        System.arraycopy(event.values, srcPos: 0, magnetometerReading, destPos: 0, magnetometerReading.size)
        hasMagnetometerData = true
    }

    // Nếu có dữ liệu từ cả hai cảm biến, tính toán hướng
    if (hasAccelerometerData && hasMagnetometerData) {
        updateOrientationAngles()
    }
}

override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
    // Xử lý khi độ chính xác của cảm biến thay đổi
    if (sensor.type == Sensor.TYPE_MAGNETIC_FIELD) {
        when (accuracy) {
            SensorManager.SENSOR_STATUS_UNRELIABLE ->
                tvStatus.text = "Không đáng tin cậy - Cần hiệu chỉnh"
            SensorManager.SENSOR_STATUS_ACCURACY_LOW ->
                tvStatus.text = "Độ chính xác thấp"
            SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM ->
                tvStatus.text = "Độ chính xác trung bình"
            SensorManager.SENSOR_STATUS_ACCURACY_HIGH ->
                tvStatus.text = "Độ chính xác cao"
        }
    }
}

private fun updateOrientationAngles() {
    // Tính toán ma trận xoay từ dữ liệu cảm biến
    SensorManager.getRotationMatrix(
        rotationMatrix,
        null,
        accelerometerReading,
        magnetometerReading
    )

    // Tính toán góc hướng từ ma trận xoay
    SensorManager.getOrientation(rotationMatrix, orientationAngles)

    // Chuyển đổi từ radian sang độ và lấy góc azimuth (hướng bắc)
    // Azimuth là góc giữa trục y của thiết bị và hướng Bắc từ tính
    val azimuthInRadians = orientationAngles[0]
    val azimuthInDegrees = Math.toDegrees(azimuthInRadians.toDouble()).toFloat()

    // Chuẩn hóa góc về 0-360 độ
    var azimuthFormatted = (azimuthInDegrees + 360) % 360

    // Cập nhật góc hiển thị
    val degreeFormatted = azimuthFormatted.toInt()
    tvDegree.text = "$degreeFormatted"

    // Xác định hướng dựa trên góc
    val directionIndex = ((azimuthFormatted / 45) + 0.5).toInt() % 8
    tvDirection.text = directions(directionIndex)

    // Tạo hiệu ứng xoay cho kim la bàn
    // Kim la bàn phải xoay ngược với góc azimuth để chỉ đúng hướng Bắc
}

```

```
    val rotateAnimation = RotateAnimation(
        currentDegree,
        -azimuthFormatted,
        Animation.RELATIVE_TO_SELF, pivotXValue: 0.5f,
        Animation.RELATIVE_TO_SELF, pivotYValue: 0.5f
    )

    // Thiết lập thời gian và chế độ nội suy cho hiệu ứng
    rotateAnimation.duration = 250
    rotateAnimation.fillAfter = true

    // Áp dụng hiệu ứng xoay cho kim la bàn
    compassNeedle.startAnimation(rotateAnimation)

    // Cập nhật góc hiện tại
    currentDegree = -azimuthFormatted
}

private fun calibrateCompass() {
    // Hiển thị hướng dẫn hiệu chỉnh la bàn
    Toast.makeText(context: this,
        text: "Xoay thiết bị theo hình số 8 để hiệu chỉnh la bàn",
        Toast.LENGTH_LONG).show()

    tvStatus.text = "Đang hiệu chỉnh..."

    // Đặt lại dữ liệu cảm biến
}

    // Đặt lại dữ liệu cảm biến
    hasAccelerometerData = false
    hasMagnetometerData = false
}
```

BÀI TẬP 10: ỨNG 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:

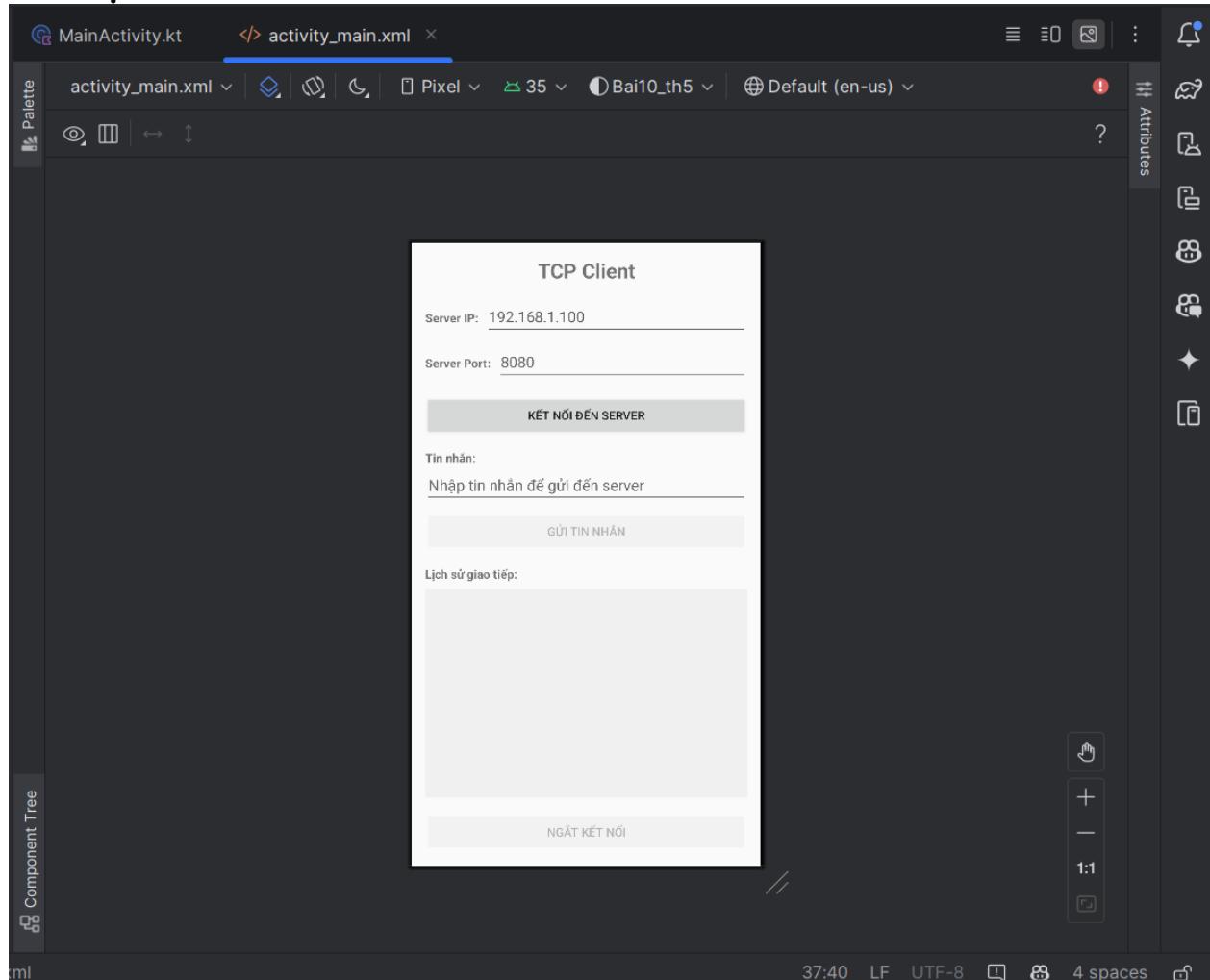
Phía Server:

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.

Giao diện:



MainActivity:

```
// Set default values
etServerIP.setText("10.0.2.2") // Default for Android emulator connecting to localhost
etServerPort.setText("8080")

// Set click listeners
btnConnect.setOnClickListener {
    if (!isConnected) {
        connectToServer()
    }
}

btnSend.setOnClickListener {
    sendMessage()
}

btnDisconnect.setOnClickListener {
    disconnectFromServer()
}
}

private fun connectToServer() {
    val serverIP = etServerIP.text.toString()
    val serverPortStr = etServerPort.text.toString()

    if (serverIP.isEmpty() || serverPortStr.isEmpty()) {
        Toast.makeText(context, text: "Vui lòng nhập IP và Port của Server", Toast.LENGTH_SHORT)
        return
    }
}
```

```
// Disable connect button and enable disconnect button
btnConnect.isEnabled() = false
appendToHistory("Đang kết nối đến $serverIP:$serverPort...")

// Connect in background thread
executorService.execute {
    try {
        // Create socket and connect to server
        clientSocket = Socket(serverIP, serverPort)
        reader = BufferedReader(InputStreamReader(clientSocket.getInputStream()))
        writer = PrintWriter(BufferedWriter(OutputStreamWriter(clientSocket.getOutputStream())))

        // Update UI on main thread
        handler.post {
            isConnected = true
            btnConnect.isEnabled() = false
            btnSend.isEnabled() = true
            btnDisconnect.isEnabled() = true
            appendToHistory("Đã kết nối thành công đến $serverIP:$serverPort")
        }
    }

    // Start listening for messages from server
    startListening()
} catch (e: Exception) {
    e.printStackTrace()
    // Update UI on main thread
    handler.post {
```

```

        btnConnect.isEnabled = true
        appendToHistory("Lỗi kết nối: ${e.message}")
        Toast.makeText(context: this@MainActivity, text: "Không thể kết nối đến server", T
    }
}

private fun startListening() {
    try {
        var message: String?
        // Keep reading messages until connection is closed
        while (reader?.readLine()?.also { message = it } != null) {
            val receivedMessage = message
            // Update UI on main thread
            handler.post {
                appendToHistory("Server: $receivedMessage")
            }
        }
    } catch (e: Exception) {
        e.printStackTrace()
        // Update UI on main thread if not already disconnected
        if (isConnected) {
            handler.post {
                appendToHistory("Kết nối bị ngắt: ${e.message}")
                disconnectFromServer()
            }
        }
    }
}

private fun sendMessage() {
    val message = etMessage.text.toString().trim()
    if (message.isEmpty()) {
        Toast.makeText(context: this, text: "Vui lòng nhập tin nhắn", Toast.LENGTH_SHORT).show()
        return
    }

    executorService.execute {
        try {
            writer?.println(message)
            // Update UI on main thread
            handler.post {
                appendToHistory("Bạn: $message")
                etMessage.text.clear()
            }
        } catch (e: Exception) {
            e.printStackTrace()
            // Update UI on main thread
            handler.post {
                appendToHistory("Lỗi gửi tin nhắn: ${e.message}")
                Toast.makeText(context: this@MainActivity, text: "Không thể gửi tin nhắn", Toast.LE
            }
        }
    }
}

```

```

private fun disconnectFromServer() {
    executorService.execute {
        try {
            // Close resources
            writer?.close()
            reader?.close()
            clientSocket?.close()

            // Update UI on main thread
            handler.post {
                isConnected = false
                btnConnect.isEnabled = true
                btnSend.isEnabled = false
                btnDisconnect.isEnabled = false
                appendToHistory("Đã ngắt kết nối từ server")
            }
        } catch (e: Exception) {
            e.printStackTrace()
            // Update UI on main thread
            handler.post {
                appendToHistory("Lỗi khi ngắt kết nối: ${e.message}")
            }
        } finally {
            writer = null
            reader = null
            clientSocket = null
        }
    }

    private fun appendToHistory(message: String) {
        val timeStamp = SimpleDateFormat(pattern: "HH:mm:ss", Locale.getDefault()).format(Date())
        val formattedMessage = "[${timeStamp}] $message\n"
        tvCommunication.append(formattedMessage)
    }
}

override fun onDestroy() {
    super.onDestroy()
    // Make sure to disconnect when activity is destroyed
    if (isConnected) {
        disconnectFromServer()
    }
    executorService.shutdown()
}

```

Thêm quyền

```

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

<application

```

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.

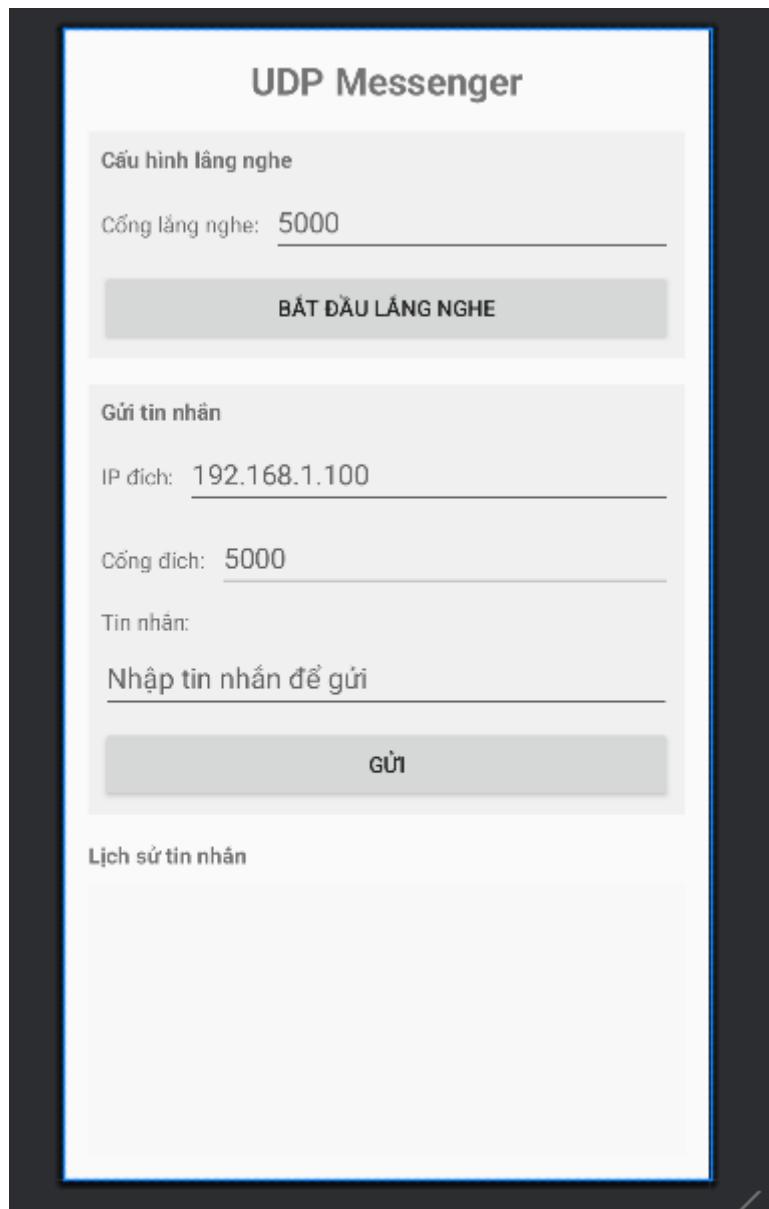
Nhận tin nhắn:

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).

Giao diện:



MainActivity:

```

    // Set click listeners
    btnStartListening.setOnClickListener {
        if (!isListening) {
            startListening()
        } else {
            stopListening()
        }
    }

    btnSend.setOnClickListener {
        sendMessage()
    }
}

private fun startListening() {
    val portStr = etListenPort.text.toString()

    if (portStr.isEmpty()) {
        Toast.makeText(context, "Vui lòng nhập cổng lắng nghe", Toast.LENGTH_SHORT).show()
        return
    }

    val port = portStr.toIntOrNull()
    if (port == null || port <= 0 || port > 65535) {
        Toast.makeText(context, "Cổng không hợp lệ. Phải là số từ 1-65535", Toast.LENGTH_SHORT).show()
    }

    // Start listening in background thread
    executorService.execute {
        try {
            // Create UDP socket
            udpSocket = DatagramSocket(port)

            // Update UI on main thread
            handler.post {
                isListening = true
                btnStartListening.text = getString(R.string.stop_listening)
                appendToHistory("Đang lắng nghe trên cổng $port...")
            }
        }

        // Start receiving packets
        while (isListening) {
            try {
                // Buffer for incoming data
                val buffer = ByteArray(size: 1024)
                val packet = DatagramPacket(buffer, buffer.size)

                // This will block until a packet is received
                udpSocket?.receive(packet)

                // Convert the received data to a string
                val message = String(packet.data, offset: 0, packet.length)
                val senderAddress = packet.address.hostAddress
                val senderPort = packet.port
            }
        }
    }
}

```

```
        // Update UI on main thread
        handler.post {
            appendToHistory("Nhận từ $senderAddress:$senderPort: $message")
        }
    } catch (e: Exception) {
        if (isListening) {
            handler.post {
                appendToHistory("Lỗi khi nhận gói tin: ${e.message}")
            }
        }
    }
} catch (e: Exception) {
    e.printStackTrace()
    // Update UI on main thread
    handler.post {
        isListening = false
        btnStartListening.text = getString(R.string.start_listening)
        appendToHistory("Lỗi khi lắng nghe: ${e.message}")
        Toast.makeText(context: this@MainActivity, text: "Không thể lắng nghe trên cổng $po
    }
}
}
```

```
private fun stopListening() {
    isListening = false

    executorService.execute {
        try {
            // Close the socket
            udpSocket?.close()
            udpSocket = null

            // Update UI on main thread
            handler.post {
                btnStartListening.text = getString(R.string.start_listening)
                appendToHistory("Đã dừng lắng nghe")
            }
        } catch (e: Exception) {
            e.printStackTrace()
            handler.post {
                appendToHistory("Lỗi khi dừng lắng nghe: ${e.message}")
            }
        }
    }
}

private fun sendMessage() {
    val targetIP = etTargetIP.text.toString()
    val targetPortStr = etTargetPort.text.toString()
    val message = etMessage.text.toString().trim()
```

```

        if (targetIP.isEmpty() || targetPortStr.isEmpty() || message.isEmpty()) {
            Toast.makeText( context: this, text: "Vui lòng nhập đầy đủ thông tin", Toast.LENGTH_SHORT).show()
            return
        }

        val targetPort = targetPortStr.toIntOrNull()
        if (targetPort == null || targetPort <= 0 || targetPort > 65535) {
            Toast.makeText( context: this, text: "Cổng đích không hợp lệ", Toast.LENGTH_SHORT).show()
            return
        }

        // Send message in background thread
        executorService.execute {
            var socket: DatagramSocket? = null
            try {
                // Create a new socket for sending
                socket = DatagramSocket()

                // Convert message to bytes
                val messageBytes = message.toByteArray()

                // Create the packet with target address and port
                val address = InetAddress.getByName(targetIP)
                val packet = DatagramPacket(messageBytes, messageBytes.size, address, targetPort)

                // Send the packet
                socket.send(packet)
            } catch (e: Exception) {
                e.printStackTrace()
                // Update UI on main thread
                handler.post {
                    appendToHistory("Lỗi khi gửi tin nhắn: ${e.message}")
                    etMessage.text.clear()
                }
            } finally {
                // Close the sending socket
                socket?.close()
            }
        }
    }

    private fun appendToHistory(message: String) {
        val timeStamp = SimpleDateFormat( pattern: "HH:mm:ss", Locale.getDefault()).format(Date())
        val formattedMessage = "[${timeStamp}] $message\n"
        tvMessageHistory.append(formattedMessage)
    }
}

```

```
override fun onDestroy() {
    super.onDestroy()
    // Make sure to stop listening when activity is destroyed
    if (isListening) {
        stopListening()
    }
    executorService.shutdown()
}
```

Thêm quyền

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

<application
```