BỘ NÔNG NGHIỆP VÀ MÔI TRƯỜNG
TRƯỜNG ĐẠI HỌC THỦY LỢI

BÀI TẬP THỰC HÀNH SỐ 5

PHÁT TRIỂN ỨNG DỤNG CHO THIẾT BỊ DI ĐỘNG

NỘI DUNG BỔ SUNG: ỨNG DỤNG VỚI CHỦ ĐỀ NÂNG CAO

STT	Mã sinh viên	Họ và tên	Lớp
1	2251061762	Vũ Xuân Duy	64CNTT2

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" />
   ```

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

    <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" />
   ```

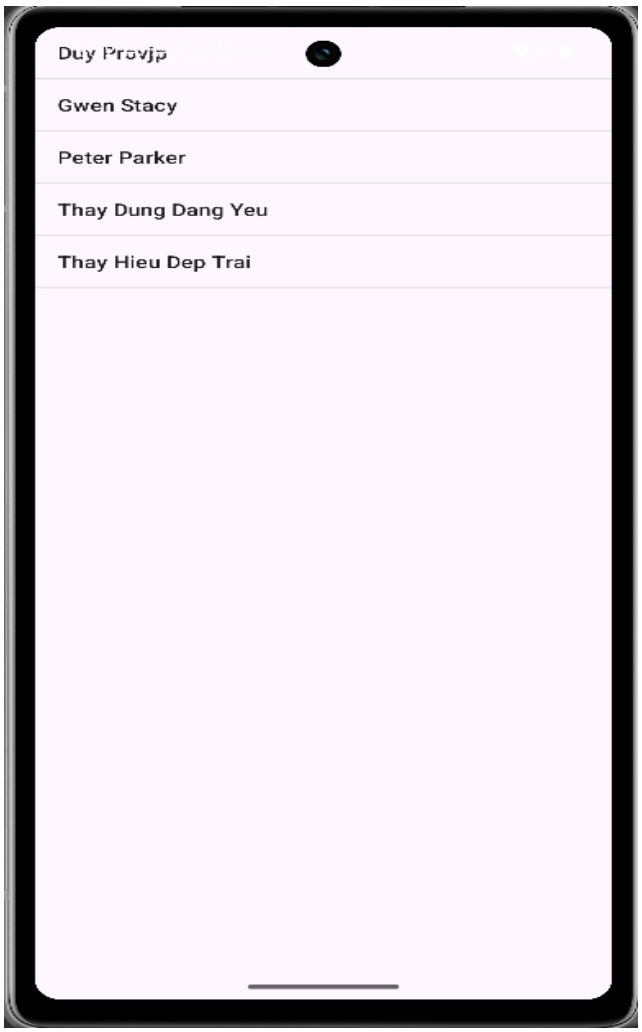
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

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

</LinearLayout>
```

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

4. **Hiển thị dữ liệu lên ListView:**
    - Tạo một `ArrayAdapter<String>` để đưa dữ liệu từ `ArrayList<String>` lên `ListView`.
    - Gán `ArrayAdapter<String>` cho `ListView`.

```
package vn.edu.tlu.contentproviders

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 android.widget.Toast
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 {
                // Hiển thị thông báo khi người dùng từ chối cấp quyền
```

```
                Toast.makeText(this, "Quyền truy cập danh bạ bị từ
chối", 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,
            ContactsContract.Contacts.DISPLAY_NAME + " ASC" // Sắp xếp
theo tên
        )

        cursor?.use {
            if (it.count > 0) {
                while (it.moveToNext()) {
                    val nameIndex =
it.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)
                    if (nameIndex >= 0) { // Kiểm tra chỉ số cột hợp lệ
                        val name = it.getString(nameIndex) ?: "Không có
tên"
                        contactsList.add(name)
                    }
                }
            } else {
                Toast.makeText(this, "Danh bạ trống",
Toast.LENGTH_SHORT).show()
            }
        }

        // 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.

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

Mục tiêu:

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

Mô tả:

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

Các bước thực hiện:

1. **Khai báo quyền:**
   - Thêm các quyền cần thiết vào `AndroidManifest.xml`:
     - `android.permission.READ_PHONE_STATE` (để theo dõi trạng thái cuộc gọi)
     - `android.permission.SEND_SMS` (để gửi tin nhắn SMS)
     - `android.permission.RECEIVE_SMS` (nếu muốn xử lý cả tin nhắn đến)
2. **Tạo Broadcast Receiver:**
   - Tạo một class kế thừa `BroadcastReceiver` để lắng nghe sự kiện `android.intent.action.PHONE_STATE`.
   - Trong phương thức `onReceive()`:
     - Kiểm tra trạng thái cuộc gọi (`TelephonyManager.EXTRA_STATE_RINGING`).
     - Nếu là cuộc gọi đến, lấy số điện thoại người gọi (`intent.getStringExtra(TelephonyManager.EXTRA_INCOMING_NUMBER)`).
     - Nếu cuộc gọi bị nhỡ (có thể theo dõi thêm sự kiện `TelephonyManager.CALL_STATE_IDLE` sau khi đổ chuông), gửi tin nhắn SMS đến số điện thoại đó.
3. **Gửi tin nhắn SMS:**
   - Sử dụng `SmsManager` để gửi tin nhắn SMS.
   - `SmsManager.getDefault().sendTextMessage()` để gửi tin nhắn.
4. **Đăng ký Broadcast Receiver:**
   - Đăng ký receiver trong `AndroidManifest.xml`.

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

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.

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:

Thêm quyền Internet

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


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

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.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/editTextUrl"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Nhập URL ảnh"
```

```
        android:inputType="textUri" />

    <Button
        android:id="@+id/buttonLoad"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tải ảnh" />

    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:visibility="gone" />

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_marginTop="16dp"
        android:scaleType="fitCenter"
        tools:src="@tools:sample/backgrounds/scenic" />

</LinearLayout>
```

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).

```
package vn.edu.tlu.ex4lab5

import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.os.AsyncTask
import android.view.View
import android.widget.ImageView
import android.widget.ProgressBar
import java.io.IOException
import java.net.HttpURLConnection
import java.net.URL
```

```
class DownloadImageTask(
    private val imageView: ImageView,
    private val progressBar: ProgressBar
) : AsyncTask<String, Integer, Bitmap>() {

    override fun onPreExecute() {
        super.onPreExecute()
        progressBar.visibility = View.VISIBLE
    }

    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()
            val inputStream = connection.inputStream
            bitmap = BitmapFactory.decodeStream(inputStream)
        } catch (e: IOException) {
            e.printStackTrace()
        }
        return bitmap
    }

    override fun onProgressUpdate(vararg values: Integer?) {
        super.onProgressUpdate(*values)
        // Bạn có thể cập nhật ProgressBar ở đây nếu muốn hiển thị tiến
trình chi tiết hơn
    }

    override fun onPostExecute(result: Bitmap?) {
        super.onPostExecute(result)
        progressBar.visibility = View.GONE
        if (result != null) {
            imageView.setImageBitmap(result)
        } else {
            // Xử lý lỗi khi tải ảnh không thành công (ví dụ: hiển thị ảnh
mặc định hoặc thông báo lỗi)
            imageView.setImageResource(R.drawable.baseline_error_24) //
Tạo một drawable ic_error
        }
    }
}
```

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)`.

```
package vn.edu.tlu.ex4lab5

import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.ImageView
import android.widget.ProgressBar
import androidx.appcompat.app.AppCompatActivity
```

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

        val editTextUrl = findViewById<EditText>(R.id.editTextUrl)
        val buttonLoad = findViewById<Button>(R.id.buttonLoad)
        val imageView = findViewById<ImageView>(R.id.imageView)
        val progressBar = findViewById<ProgressBar>(R.id.progressBar)

        buttonLoad.setOnClickListener {
            val url = editTextUrl.text.toString().trim()
            if (url.isNotEmpty()) {
                DownloadImageTask(imageView, progressBar).execute(url)
            } else {
                editTextUrl.error = "Vui lòng nhập URL ảnh"
            }
        }
    }
}
```

Giải thích code:

- **activity_main.xml:**
  - `EditText` cho phép người dùng nhập URL.
  - `Button` để kích hoạt việc tải ảnh.
  - `ProgressBar` hiển thị tiến trình tải, ban đầu được đặt `visibility="gone"`.
  - `ImageView` để hiển thị ảnh đã tải.
- **DownloadImageTask.kt:**
  - Kế thừa `AsyncTask<String, Integer, Bitmap>`.
  - **`onPreExecute()`:** Được gọi trước khi `doInBackground` chạy, hiển thị `ProgressBar`.
  - **`doInBackground(vararg urls: String?)`:**
    - Lấy URL từ tham số đầu vào.
    - Sử dụng `HttpURLConnection` để tải dữ liệu từ URL.
    - Giải mã dữ liệu tải về thành đối tượng `Bitmap`.
    - Trả về `Bitmap` đã tải.
    - **Lưu ý:** Bạn có thể thêm logic để cập nhật tiến trình tải bằng cách gọi `publishProgress(percentage)` trong quá trình tải, và xử lý nó trong `onProgressUpdate`. Tuy nhiên, với việc tải ảnh đơn giản, việc này có thể không cần thiết.
  - **`onProgressUpdate(vararg values: Integer?)`:** Được gọi khi `publishProgress()` được gọi từ `doInBackground`. Bạn có thể sử dụng nó để cập nhật giao diện người dùng về tiến trình tải (ví dụ: cập nhật giá trị của `ProgressBar`).
  - **`onPostExecute(result: Bitmap?)`:** Được gọi sau khi `doInBackground` hoàn thành.
    - Ẩn `ProgressBar`.
    - Nếu `result` không null (tức là tải ảnh thành công), hiển thị `Bitmap` lên `ImageView`.
    - Nếu `result` là null (tải ảnh thất bại), bạn có thể hiển thị một ảnh mặc định hoặc thông báo lỗi cho người dùng.
- **MainActivity.kt:**
  - Ánh xạ các view từ layout XML vào các biến trong code.
  - Đặt `OnClickListener` cho `Button` "Tải ảnh".
  - Khi nút được nhấn:
    - Lấy URL từ `EditText`.
    - Kiểm tra xem URL có rỗng không. Nếu không rỗng, tạo một instance của `DownloadImageTask` và gọi `execute(url)` để bắt đầu quá trình tải ảnh trong background.
    - Nếu URL rỗng, hiển thị thông báo lỗi trên `EditText`.

Các bước thực hiện chi tiết:

1. **Tạo Project Kotlin DSL:** Nếu bạn chưa tạo, hãy tạo một project Android mới và chọn "Empty Views Activity" và ngôn ngữ Kotlin DSL.
2. **Tạo Layout XML:** Tạo file layout `activity_main.xml` và thêm các view như đã mô tả ở trên.
3. **Tạo AsyncTask Class:** Tạo một file Kotlin mới tên là `DownloadImageTask.kt` và copy code của class `DownloadImageTask` vào đó.
4. **Sửa đổi MainActivity:** Mở file `MainActivity.kt` và copy code của `MainActivity` vào đó.
5. **Thêm quyền Internet:** Mở file `AndroidManifest.xml` và thêm quyền truy cập Internet:

XML

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

6. **Tạo Drawable lỗi (tùy chọn):** Nếu bạn muốn hiển thị một ảnh lỗi khi tải không thành công, hãy tạo một drawable vector hoặc bitmap và đặt tên là `ic_error`.
7. **Chạy ứng dụng:** Chạy ứng dụng trên thiết bị ảo hoặc thiết bị thật.

Lưu ý quan trọng:

- **Xử lý lỗi:** Trong `doInBackground`, bạn nên có cơ chế xử lý lỗi tốt hơn (ví dụ: bắt các ngoại lệ cụ thể và thông báo cho người dùng).
- **Thư viện bên thứ ba:** Đối với các ứng dụng thực tế, bạn có thể cân nhắc sử dụng các thư viện mạnh mẽ hơn để tải ảnh như Glide, Picasso hoặc Coil. Chúng cung cấp nhiều tính năng như caching, quản lý bộ nhớ và xử lý ảnh hiệu quả hơn.
- **Luồng chính (Main Thread):** Hãy nhớ rằng các thao tác liên quan đến UI (ví dụ: cập nhật `ImageView`, `ProgressBar`) phải được thực hiện trên UI Thread (Main Thread). `AsyncTask` giúp bạn thực hiện các tác vụ tốn thời gian trên background thread và tự động chuyển về UI Thread để cập nhật giao diện.

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

Mục tiêu:

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

Mô tả:

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

Các bước thực hiện:

1. **Thiết kế giao diện:**
    - Một TextView để hiển thị thời gian.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    android:padding="16dp"
    android:background="#000000" tools:context=".MainActivity">

    <TextView
        android:id="@+id/textViewTimer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="00:00:00"
```

```
        android:textSize="48sp"
        android:textColor="#FF0000" android:typeface="monospace"
android:textStyle="bold" />

</LinearLayout>
```

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

```
package vn.edu.tlu.ex5lab5

import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import java.util.concurrent.TimeUnit

class MainActivity : AppCompatActivity() {

    private lateinit var textViewTimer: TextView
    private var seconds = 0
    private val handler = Handler(Looper.getMainLooper())
    private lateinit var timerRunnable: Runnable

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

        textViewTimer = findViewById(R.id.textViewTimer)

        timerRunnable = object : Runnable {
            override fun run() {
                seconds++
                val timeString = formatTime(seconds)
                textViewTimer.text = timeString
                handler.postDelayed(this, 1000) // Cập nhật sau mỗi 1
giây
            }
        }

        startTimer()
    }

    override fun onStop() {
        super.onStop()
        stopTimer()
    }
```

```
    private fun startTimer() {
        handler.post(timerRunnable)
    }

    private fun stopTimer() {
        handler.removeCallbacks(timerRunnable)
    }

    private fun formatTime(totalSeconds: Int): String {
        val seconds = totalSeconds % 60
        val minutes = (totalSeconds / 60) % 60
        val hours = totalSeconds / 3600
        return String.format("%02d:%02d:%02d", hours, minutes, seconds)
    }
}
```

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.

Giải thích code:

- **activity_main.xml:**
  - Chỉ có một `TextView` để hiển thị thời gian đã trôi qua.
- **MainActivity.kt:**
  - **`textViewTimer`:** Biến để tham chiếu đến `TextView` trong layout.
  - **`seconds`:** Biến để lưu trữ tổng số giây đã trôi qua.
  - **`handler`:** Một instance của `Handler` được liên kết với `Looper.getMainLooper()`. Điều này đảm bảo rằng các tác vụ được gửi đến `Handler` sẽ được thực thi trên UI thread.
  - **`timerRunnable`:** Một instance của `Runnable` chứa logic để cập nhật giao diện.
    - **`run()`:** Phương thức này được gọi khi `Runnable` được thực thi.
      - Tăng giá trị `seconds`.
      - Định dạng thời gian thành chuỗi (ví dụ: "00:00:01").
      - Cập nhật `textViewTimer.text` trên UI thread.
      - Sử dụng `handler.postDelayed(this, 1000)` để lên lịch cho việc thực thi lại `Runnable` này sau 1 giây.
  - **`onCreate()`:**
    - Khởi tạo `textViewTimer`.
    - Tạo instance của `timerRunnable`.
    - Gọi `startTimer()` để bắt đầu đếm giờ.
  - **`onStop()`:**
    - Được gọi khi Activity không còn hiển thị.
    - Gọi `stopTimer()` để dừng việc cập nhật thời gian, tránh lãng phí tài nguyên khi Activity không hoạt động.
  - **`startTimer()`:**
    - Sử dụng `handler.post(timerRunnable)` để gửi `Runnable` lên UI thread để thực thi lần đầu tiên.
  - **`stopTimer()`:**
    - Sử dụng `handler.removeCallbacks(timerRunnable)` để loại bỏ bất kỳ lệnh gọi lại nào đang chờ xử lý của `timerRunnable`. Điều này dừng việc cập nhật định kỳ.
  - **`formatTime()`:**
    - Một hàm tiện ích để định dạng tổng số giây thành chuỗi "HH:MM:SS".

Các bước thực hiện chi tiết:

1. **Tạo Project Kotlin:** Tạo một project Android mới và chọn "Empty Views Activity" và ngôn ngữ Kotlin.
2. **Tạo Layout XML:** Tạo file layout `activity_main.xml` và thêm `TextView` như đã mô tả ở trên.
3. **Sửa đổi MainActivity:** Mở file `MainActivity.kt` và copy code của `MainActivity` vào đó.
4. **Chạy ứng dụng:** Chạy ứng dụng trên thiết bị ảo hoặc thiết bị thật. Bạn sẽ thấy `TextView` bắt đầu đếm giây. Khi bạn chuyển sang ứng dụng khác hoặc tắt màn hình, bộ đếm sẽ dừng lại và tiếp tục khi bạn quay lại ứng dụng.

Lưu ý quan trọng:

- **`Handler` và `Runnable`:** Đây là cách tiêu chuẩn để cập nhật UI từ một thread khác trong Android. `Handler` cho phép bạn gửi và xử lý các message hoặc `Runnable` trên một thread cụ thể (trong trường hợp này là UI thread).
- **`Looper.getMainLooper()`:** `Looper` quản lý một hàng đợi các message và `Runnable` cho một thread. `Looper.getMainLooper()` trả về `Looper` được liên kết với UI thread.
- **`postDelayed()`:** Phương thức này của `Handler` lên lịch cho việc thực thi `Runnable` sau một khoảng thời gianDelay (trong ví dụ này là 1000ms = 1 giây).
- **`removeCallbacks()`:** Rất quan trọng để loại bỏ các lệnh gọi lại đang chờ xử lý khi Activity không còn hoạt động. Nếu bạn không dừng `Runnable`, nó có thể tiếp tục chạy ngầm và cố gắng cập nhật UI khi Activity không còn hiển thị, gây ra lỗi.
- **Định dạng thời gian:** Hàm `formatTime()` giúp hiển thị thời gian một cách dễ đọc. Bạn có thể tùy chỉnh định dạng này theo ý muốn.

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:

Cấp quyền:

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

Giao diện

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

    <Button
        android:id="@+id/btnStartRecording"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bắt đầu ghi âm" />

    <Button
```

```
        android:id="@+id/btnStopRecording"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Dừng ghi âm"
        android:visibility="gone"/>

    <Button
        android:id="@+id/btnPlayRecording"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Phát lại"
        android:visibility="gone"/>
</LinearLayout>
```

1. **Ghi âm:**
    - Sử dụng `MediaRecorder` để ghi âm.
    - Thiết lập các thông số cần thiết như nguồn âm thanh, định dạng file, nơi lưu trữ.
    - Lưu file ghi âm vào một vị trí cụ thể.

```
mediaRecorder = MediaRecorder().apply {
    setAudioSource(MediaRecorder.AudioSource.MIC)  // Lấy âm thanh từ
micro
    setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP)  // Định dạng
file
    setOutputFile(audioFilePath)  // Lưu file tại đường dẫn
    setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB)  // Bộ mã hóa âm
thanh
    try {
        prepare()
        start()
    } catch (e: IOException) {
        e.printStackTrace()
    }
}
```

    - Sử dụng `ContentValues` để thêm thông tin về file ghi âm vào MediaStore.

```
private fun stopRecording() {
    mediaRecorder?.apply {
        stop()
        release()
    }
    mediaRecorder = null
}
```

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

```
private fun playRecording() {
    mediaPlayer = MediaPlayer().apply {
        try {
            setDataSource(audioFilePath)
```

```
            prepare()
            start()
        } catch (e: IOException) {
            e.printStackTrace()
        }
    }
}
```

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

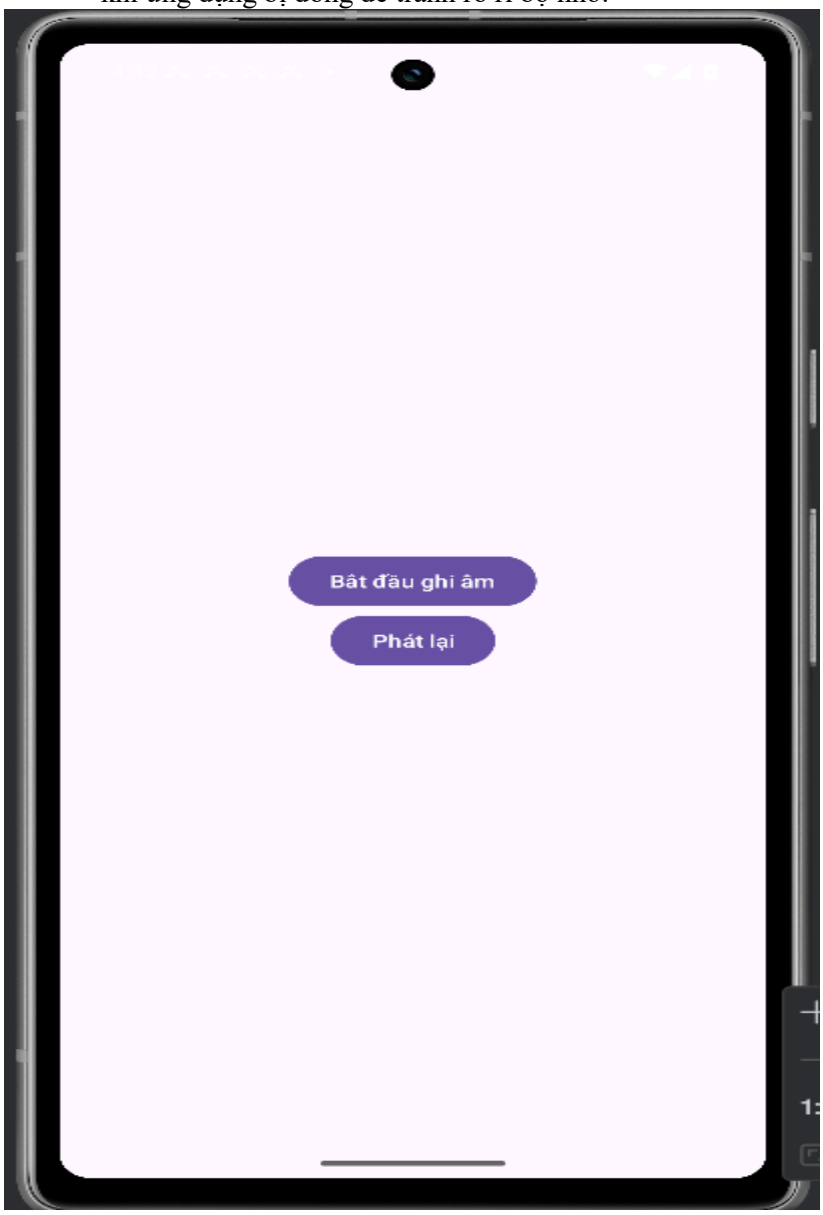
```
package android.example.com.audiorecorder

import android.Manifest
import android.content.pm.PackageManager
import android.media.MediaPlayer
import android.media.MediaRecorder
import android.os.Bundle
import android.widget.Button
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import java.io.IOException

class MainActivity : AppCompatActivity() {

    private var mediaRecorder: MediaRecorder? = null
    private var mediaPlayer: MediaPlayer? = null
    private var audioFilePath: String = ""

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

        val btnStartRecording = findViewById<Button>(R.id.btnStartRecording)
        val btnStopRecording = findViewById<Button>(R.id.btnStopRecording)
        val btnPlayRecording = findViewById<Button>(R.id.btnPlayRecording)

        // Kiểm tra quyền
        if (ContextCompat.checkSelfPermission(this,
Manifest.permission.RECORD_AUDIO) != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this,
arrayOf(Manifest.permission.RECORD_AUDIO), 0)
        }

        // Lưu đường dẫn file ghi âm
        audioFilePath = externalCacheDir?.absolutePath +
"/recorded_audio.3gp"

        // Bắt đầu ghi âm
        btnStartRecording.setOnClickListener {
            startRecording()
            btnStartRecording.visibility = android.view.View.GONE
            btnStopRecording.visibility = android.view.View.VISIBLE
        }

        // Dừng ghi âm
        btnStopRecording.setOnClickListener {
```

```
            stopRecording()
            btnStopRecording.visibility = android.view.View.GONE
            btnPlayRecording.visibility = android.view.View.VISIBLE
            btnStartRecording.visibility = android.view.View.VISIBLE
        }

        // Phát lại file ghi âm
        btnPlayRecording.setOnClickListener {
            playRecording()
        }
    }

    // Hàm bắt đầu ghi âm
    private fun startRecording() {
        mediaRecorder = MediaRecorder().apply {
            setAudioSource(MediaRecorder.AudioSource.MIC)  // Lấy âm thanh từ
micro
            setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP)  // Định
dạng file
            setOutputFile(audioFilePath)  // Lưu file tại đường dẫn
            setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB)  // Bộ mã hóa
âm thanh
            try {
                prepare()
                start()
            } catch (e: IOException) {
                e.printStackTrace()
            }
        }
    }

    // Hàm dừng ghi âm
    private fun stopRecording() {
        mediaRecorder?.apply {
            stop()
            release()
        }
        mediaRecorder = null
    }

    // Hàm phát lại file ghi âm
    private fun playRecording() {
        mediaPlayer = MediaPlayer().apply {
            try {
                setDataSource(audioFilePath)
                prepare()
                start()
            } catch (e: IOException) {
                e.printStackTrace()
            }
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        mediaRecorder?.release()
        mediaRecorder = null
        mediaPlayer?.release()
        mediaPlayer = null
    }
}
```

Giải thích:

- **mediaRecorder**: Đối tượng giúp ghi âm.

- **mediaPlayer**: Đối tượng giúp phát lại file ghi âm.
- **audioFilePath**: Đường dẫn file ghi âm.
- **findViewById<Button>(R.id.btnStartRecording)**: Kết nối các nút với giao diện.
- **btnStartRecording** → Bắt đầu ghi âm.
- **btnStopRecording** → Dừng ghi âm.
- **btnPlayRecording** → Phát lại file ghi âm.
- **Lưu file ghi âm trong thư mục externalCacheDir của ứng dụng**.
- Định dạng file là **.3gp** (nhẹ, tối ưu cho ghi âm).
- **setAudioSource(MediaRecorder.AudioSource.MIC)**: Ghi âm từ **microphone**.
- **setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP)**: Định dạng file .3gp.
- **setOutputFile(audioFilePath)**: Lưu file tại đường dẫn đã khai báo.
- **setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB)**: Bộ mã hóa âm thanh.
- **setDataSource(audioFilePath)** → Chọn file ghi âm vừa lưu.
- mediaRecorder?.release(), mediaPlayer?.release(): **Giải phóng MediaRecorder và MediaPlayer** khi ứng dụng bị đóng để tránh rò rỉ bộ nhớ.



BÀI TẬP 7: Ứng dụng chơi video đơn giản

Mục tiêu:

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

Mô tả:

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

Các bước thực hiện:

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

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

    <EditText
        android:id="@+id/edtVideoUrl"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Nhập URL video"
        android:inputType="textUri"/>

    <Button
        android:id="@+id/btnLoadUrl"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Phát từ URL"/>

    <Button
        android:id="@+id/btnPickVideo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Chọn Video từ thư viện"/>

    <VideoView
        android:id="@+id/videoView"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:layout_marginTop="10dp"/>
</LinearLayout>
```

2. **Chọn video:**
    - Cho phép người dùng chọn video từ MediaStore bằng cách sử dụng `Intent.ACTION_PICK` và `MediaStore.Video.Media.EXTERNAL_CONTENT_URI`.
    - Hoặc cho phép người dùng nhập URL của video.

```
btnPickVideo.setOnClickListener {
    val intent = Intent(Intent.ACTION_PICK)
    intent.type = "video/*"
```

```
    videoPickerLauncher.launch(intent)
}
```

3. **Phát video:**
    - Sử dụng `VideoView.setVideoURI()` để thiết lập nguồn video.
    - Sử dụng `MediaController` để cung cấp các điều khiển phát lại video (play, pause, stop,...).
    - `VideoView.start()` để bắt đầu phát video.

```
private fun playVideo(videoUri: Uri) {
    videoView.setVideoURI(videoUri)
    videoView.start()
}
```

MainActivity.kt

```
package android.example.com.videoplayer

import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.VideoView
import android.widget.MediaController
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    private lateinit var videoView: VideoView
    private lateinit var mediaController: MediaController

    // Sử dụng Activity Result API để thay thế startActivityForResult
    private val videoPickerLauncher =

registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
result ->
            if (result.resultCode == RESULT_OK) {
                val videoUri: Uri? = result.data?.data
                if (videoUri != null) {
                    playVideo(videoUri)
                }
            }
        }

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

        videoView = findViewById(R.id.videoView)
        val btnPickVideo = findViewById<Button>(R.id.btnPickVideo)
        val btnLoadUrl = findViewById<Button>(R.id.btnLoadUrl)
        val edtVideoUrl = findViewById<EditText>(R.id.edtVideoUrl)

        // Thêm MediaController để điều khiển phát video (play, pause, stop,
tua)
        mediaController = MediaController(this)
        mediaController.setAnchorView(videoView)
```

```
        videoView.setMediaController(mediaController)

        // Chọn video từ thư viện
        btnPickVideo.setOnClickListener {
            val intent = Intent(Intent.ACTION_PICK)
            intent.type = "video/*"
            videoPickerLauncher.launch(intent)
        }

        // Phát video từ URL nhập vào
        btnLoadUrl.setOnClickListener {
            val url = edtVideoUrl.text.toString()
            if (url.isNotEmpty()) {
                playVideo(Uri.parse(url))
            }
        }
    }

    // Phát video từ URI
    private fun playVideo(videoUri: Uri) {
        videoView.setVideoURI(videoUri)
        videoView.start()
    }
}
```

Giải thích:

registerForActivityResult:

- **resultCode == RESULT_OK**: Kiểm tra xem thao tác thành công không.
- **result.data?.data**: Lấy đường dẫn video đã chọn.
- **playVideo(videoUri)**: Gọi hàm phát video.

Hàm onCreate

- **videoView**: Hiển thị video.
- **btnPickVideo**: Nút chọn video từ thư viện.
- **btnLoadUrl**: Nút phát video từ URL.
- **edtVideoUrl**: Ô nhập URL video

- **MediaController(this)**: Tạo thanh điều khiển.
- **setAnchorView(videoView)**: Gắn điều khiển vào videoView.
- **videoView.setMediaController(mediaController)**: Liên kết thanh điều khiển với video

btnPickVideo.setOnClickListener

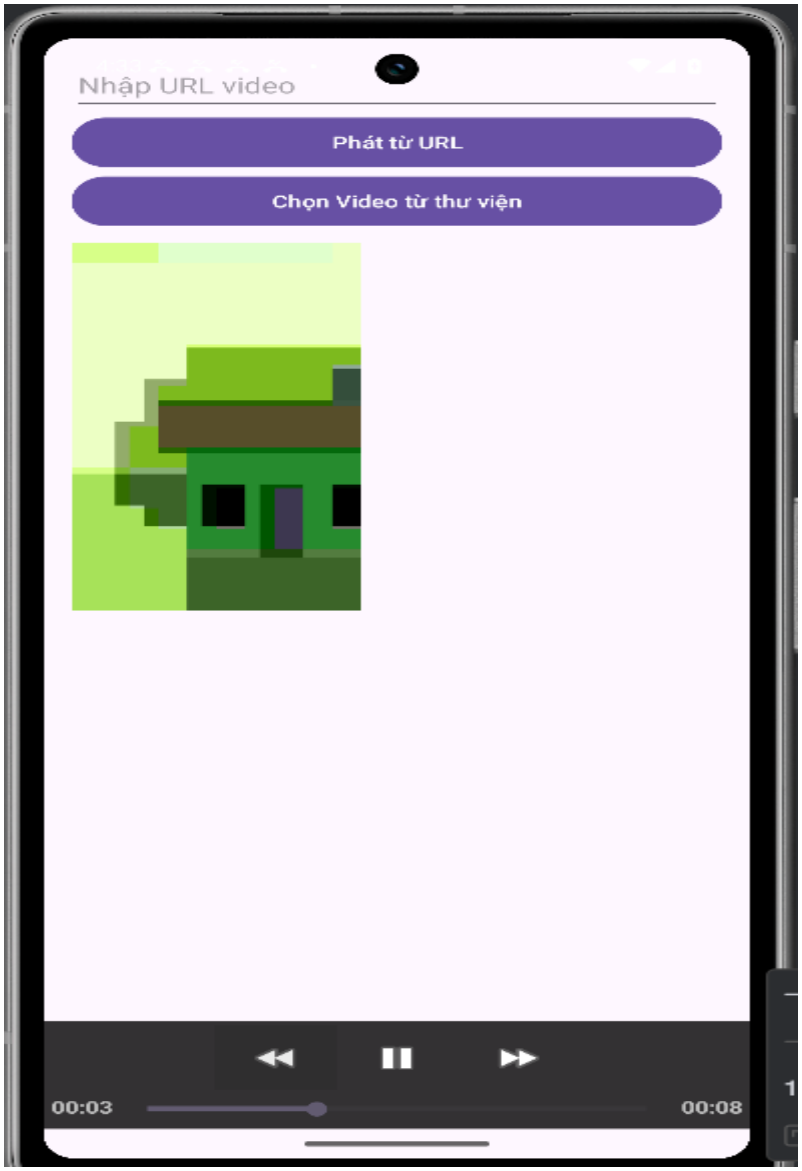
- Intent.ACTION_PICK: Cho phép người dùng chọn file video.
- intent.type = "video/*": Chỉ chọn file video.
- videoPickerLauncher.launch(intent): Gửi Intent và nhận kết quả thông qua videoPickerLauncher.

btnLoadUrl.setOnClickListener

- url = edtVideoUrl.text.toString() Lấy URL từ EditText
- if (url.isNotEmpty()) Kiểm tra nếu URL không rỗng.
- playVideo(Uri.parse(url)) Chuyển đổi URL thành Uri.Gọi playVideo(Uri.parse(url)) để phát video.

private fun playVideo

- setVideoURI(videoUri): Gán đường dẫn video cho VideoView.
- start(): 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.

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`.

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 nhiễu 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.

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.

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).