

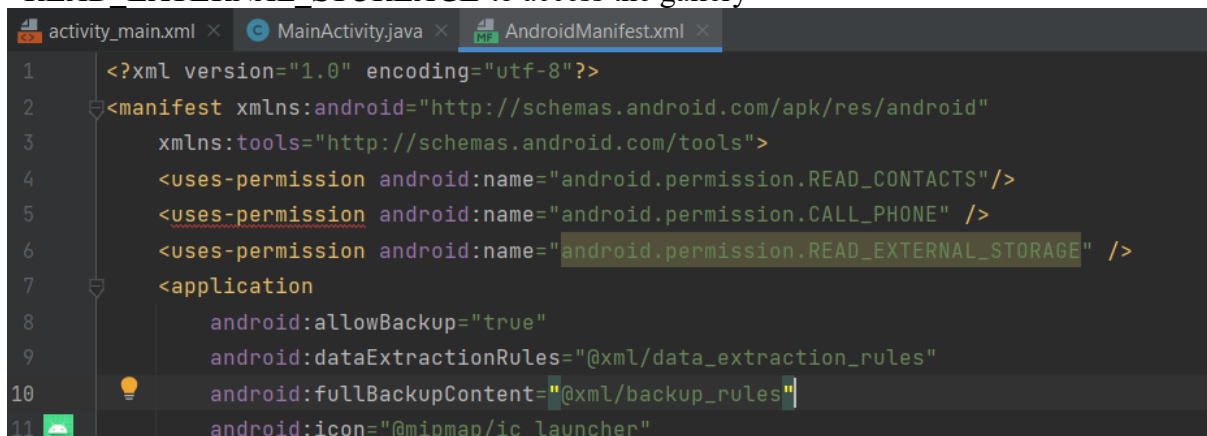
Common Intents in Android

Group members:

Student ID	Student name	Intents
21125080	Quan Phú Long	1, 2, 3
21125045	Hứa Trường Khả	4, 5, 6
21125053	Thi Hồng Nhựt	7, 8, 9
21125145	Vương Quang Việt Tùng	10, 11, 12

First put the permission to the AndroidManifest file, in my app there are three permissions that the app needs, so I've put:

- ***READ_CONTACTS*** for the selectContact Intent
- ***CALL_PHONE*** to call that contact and
- ***READ_EXTERNAL_STORAGE*** to access the gallery



```
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    <uses-permission android:name="android.permission.CALL_PHONE" />
6    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
7    <application
8        android:allowBackup="true"
9        android:dataExtractionRules="@xml/data_extraction_rules"
10       android:fullBackupContent="@xml/backup_rules"
11       android:icon="@mipmap/ic_launcher"
```

1. Select a Contact and Call:

```

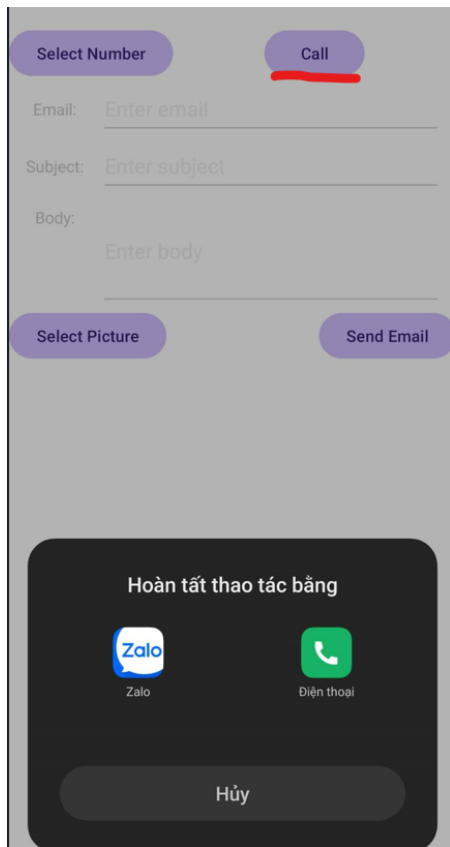
private void selectContact() {
    Intent intent = new Intent(Intent.ACTION_PICK, ContactsContract.Contacts.CONTENT_URI);
    startActivityForResult(intent, REQUEST_CONTACT);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == REQUEST_CONTACT && resultCode == RESULT_OK) {
        Uri contactUri = data.getData();
        String name = getContactName(contactUri);
        String phoneNumber = getContactPhoneNumber(contactUri);

        nameTextView.setText(name);
        phoneNumberTextView.setText(phoneNumber);
    }
    if (requestCode == REQUEST_IMAGE_PICK && resultCode == RESULT_OK && data != null) {
        Uri imageUri = data.getData();
        if (imageUri != null) {
            try {
                Bitmap bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(), imageUri);
                addImageToGridLayout(bitmap);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
}

```

The selectContact function will ask the user to select a contact. And make sure to override the onActivityResult function like on the common intent android guide. After picking the contact, it will call that person.



To be able to access contacts and then call phone, the following permissions must be granted and declared in *AndroidManifest.xml* beforehand:

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

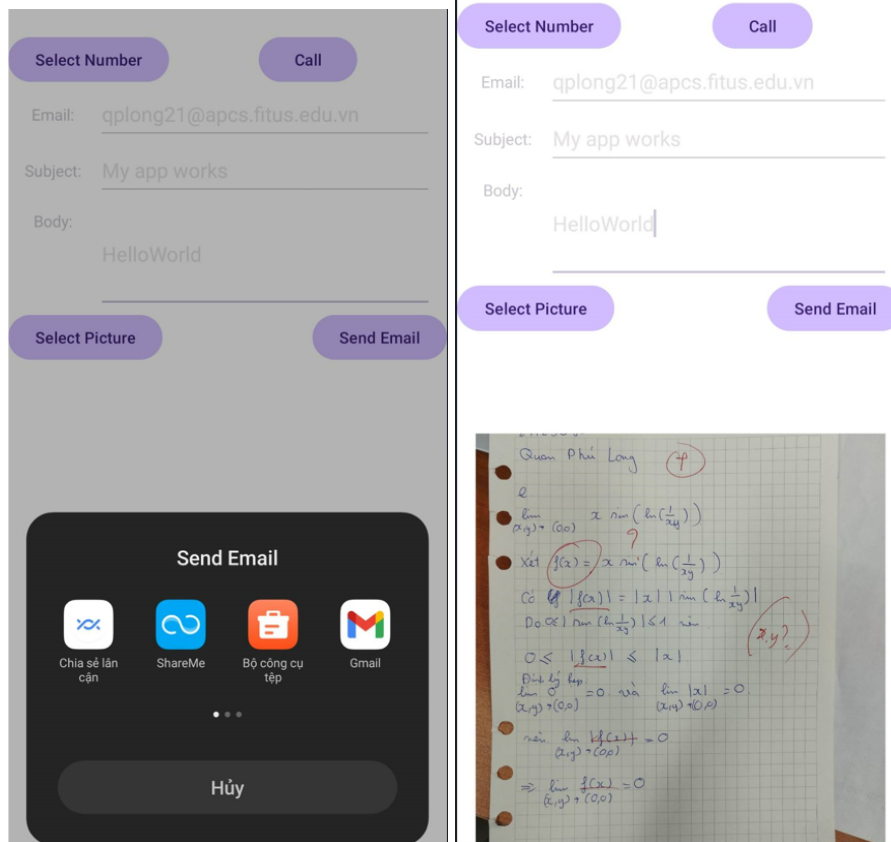
2. Send email:

```
private void composeEmail(String address, String subject, String body, Uri attachmentUri) {
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("message/rfc822");
    intent.putExtra(Intent.EXTRA_EMAIL, new String[]{address});
    intent.putExtra(Intent.EXTRA_SUBJECT, subject);
    intent.putExtra(Intent.EXTRA_TEXT, body);

    if (attachmentUri != null) {
        intent.putExtra(Intent.EXTRA_STREAM, attachmentUri);
    }

    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(Intent.createChooser(intent, "Send Email"));
    }
}
```

The application does support the ability to send email to a specified recipient. It will use the account from chosen Mail app (such as Gmail) to complete the task. The ability to send attachment is another feature of this intent, allowing the sender to add an attachment to an email.



3. Pick image from Gallery:

```
private void openGallery() {
    Intent galleryIntent = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
    startActivityForResult(galleryIntent, REQUEST_IMAGE_PICK);
}

1 usage
private void addImageToGridLayout(Bitmap bitmap) {
    // Remove the previous ImageView, if any

private static final int REQUEST_IMAGE_PICK = 2;
```

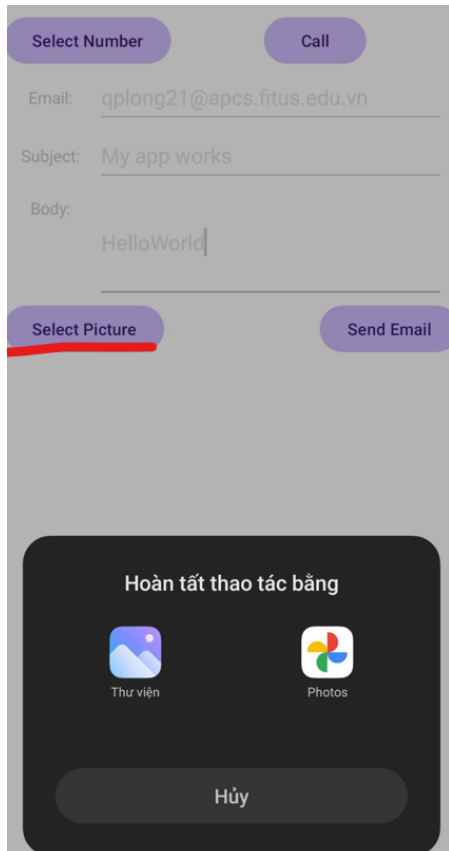
```

@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == REQUEST_CONTACT && resultCode == RESULT_OK) {
        Uri contactUri = data.getData();
        String name = getContactName(contactUri);
        String phoneNumber = getContactPhoneNumber(contactUri);

        nameTextView.setText(name);
        phoneNumberTextView.setText(phoneNumber);
    }
    if (requestCode == REQUEST_IMAGE_PICK && resultCode == RESULT_OK && data != null) {
        Uri imageUri = data.getData();
        if (imageUri != null) {
            try {
                Bitmap bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(), imageUri);
                addImageToGridLayout(bitmap);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
}
}
}
}

```

And these are some pictures about my app



The application must have the required permission of *reading external storage* in order to proceed with accessing photos and videos from the Gallery:

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

4. Create a timer

To create a timer, use the **ACTION_SET_TIMER** action.

To specify timer details, use the extras:

EXTRA_LENGTH: length of timer in second.

EXTRA_MESSAGE: timer name.

EXTRA_SKIP_UI: A boolean specifying whether the responding app must skip its UI when setting the timer. If true, the app must bypass any confirmation UI and start the specified timer.

The following permission must be declared in the *AndroidManifest.xml* file to be able to invoke the task of setting alarm:

```
<uses-permission android:name="com.android.alarm.permission.SET_ALARM"/>
```

Example:

```
public void startTimer(String message, int seconds) {
    Intent intent = new Intent(AlarmClock.ACTION_SET_TIMER)
        .putExtra(AlarmClock.EXTRA_MESSAGE, message)
        .putExtra(AlarmClock.EXTRA_LENGTH, seconds)
        .putExtra(AlarmClock.EXTRA_SKIP_UI, value: true);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```

5. Create a calendar event:

To add a new event to the user's calendar, use the **ACTION_INSERT** action.

To specify timer details, use the extras:

EXTRA_EVENT_ALL_DAY: boolean specifying whether this is an all-day event.

EXTRA_EVENT_BEGIN_TIME: start time of the event (milliseconds since epoch)

EXTRA_EVENT_END_TIME: end time of the event (milliseconds since epoch)

TITLE: event title

DESCRIPTION: event discription

EVENT_LOCATION: event location

EXTRA_EMAIL: a comma-separated list of email addresses that specify the invitees.

Example:

```

public void addEvent(String title, String location) {
    Intent intent = new Intent(Intent.ACTION_INSERT)
        .setData(CalendarContract.Events.CONTENT_URI)
        .putExtra(CalendarContract.Events.TITLE, title)
        .putExtra(CalendarContract.Events.EVENT_LOCATION, location);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
}

```

6. Open camera:

To open a camera app in *image* mode, use the **INTENT_ACTION_STILL_IMAGE_CAMERA** action.

Example:

```

Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
intent.putExtra(MediaStore.EXTRA_OUTPUT,
    Uri.withAppendedPath(locationForPhotos, targetFilename));

```

The application will start the camera app and allow user to capture using this intent. Another mode from the intent is **INTENT_ACTION_VIDEO_CAMERA**, in which it will initiate a video recording mode.

```

Intent intent = new Intent(MediaStore.INTENT_ACTION_VIDEO_CAMERA);

```

Android development also supports the developer to program how a program should function after taking a picture. It takes data from the intent in the below *Override* function:

```

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    if (requestCode == 1 && resultCode == RESULT_OK) {
        // Handle the result of the image capture
    }
}

```

7. Load a web URL

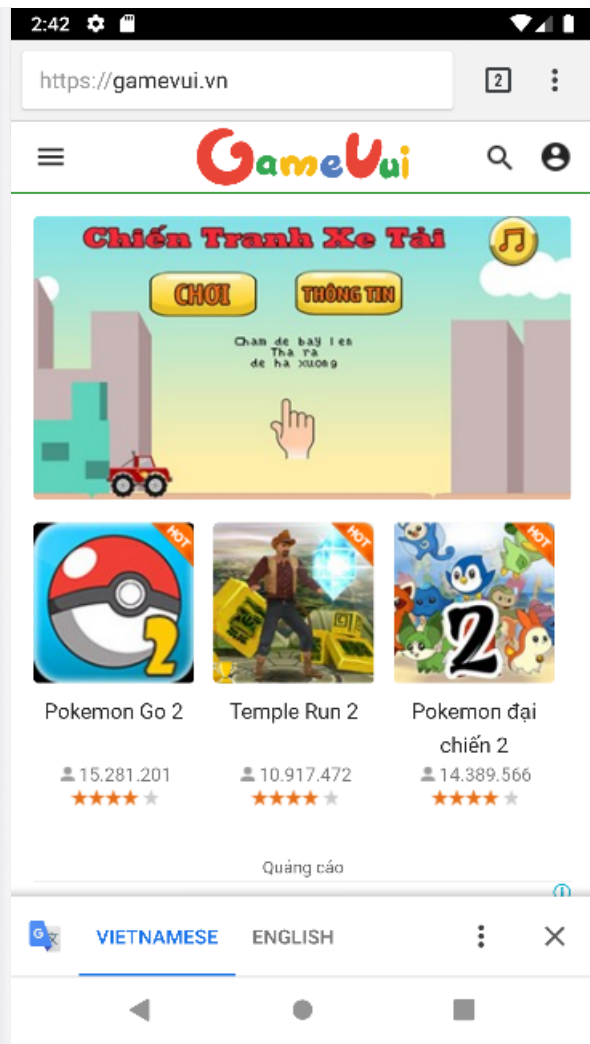
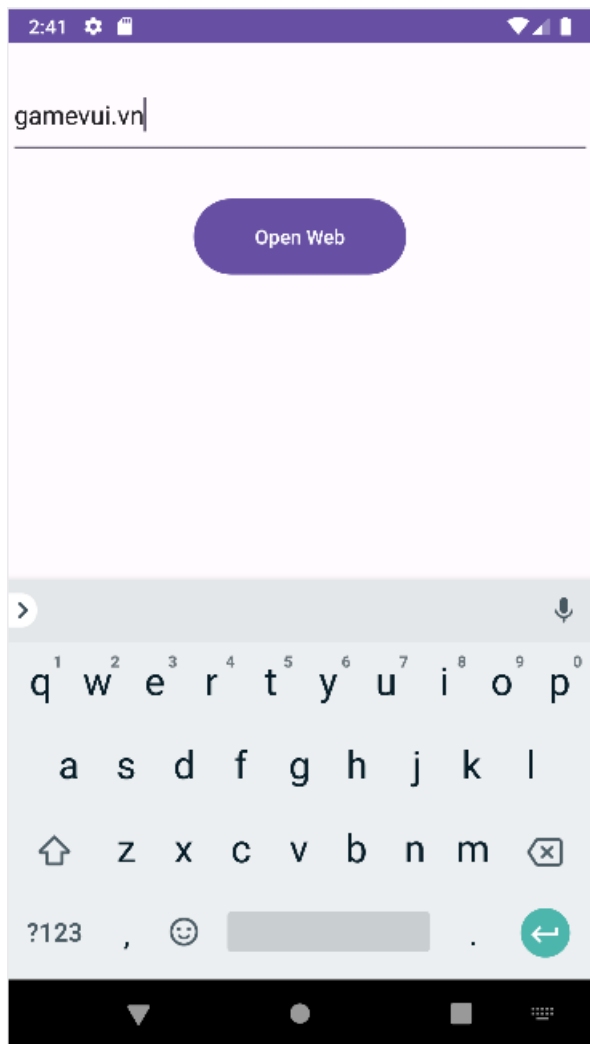
An Android application allows users to open a web browser with its URL. This is the syntax that we need to apply this intent.

```
public void openWebPage(String url) {  
    Uri webpage = Uri.parse(url);  
    Intent intent = new Intent(Intent.ACTION_VIEW, webpage);  
    if (intent.resolveActivity(getPackageManager()) != null) {  
        startActivity(intent);  
    }  
}
```

In the file AndroidManifest.xml, we also need to declare the following intent filter.

```
<intent-filter>  
    <action android:name="android.intent.action.VIEW" />  
    <!-- Include the host attribute if you want your app to respond  
        only to URLs with your app's domain. -->  
    <data android:scheme="http" android:host="www.example.com" />  
    <category android:name="android.intent.category.DEFAULT" />  
    <!-- The BROWSABLE category is required to get links from web pages. -->  
    <category android:name="android.intent.category.BROWSABLE" />  
</intent-filter>
```

To implement this activity as an application on Android Studio, we need to implement an edit text and a button. The edit text is used to get the URL of the web browser from users. The button is used to trigger the activity of getting data from the edit text and open the web page with that URL.



8. Maps: show a location on a map

An Android application allows users to input a location in the form of (latitude, longitude) or name. Then, the application will show that location on the map in the map application on the phone. This is the syntax that we need to apply to that activity.

```
public void showMap(String geoLocation) {  
    Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(geoLocation));  
    if (intent.resolveActivity(getPackageManager()) != null) {  
        startActivity(intent);  
    }  
}
```

In the file AndroidManifest.xml, we also need to declare the following intent filter.

```
<intent-filter>  
    <action android:name="android.intent.action.VIEW" />  
    <data android:scheme="geo" />  
    <category android:name="android.intent.category.DEFAULT" />  
</intent-filter>
```

To implement the activity as an application in Android Studio, we can choose whether user want to input the name of locations or the latitude and longitude of them. In both cases, we need to convert the information that users input into a form that is suitable for the intent.

```
String Slocation = "geo:" + latitude + "," + longitude + "?z=22&q=";  
String newLocationName = "geo:0,0?q="+formatName(locationName);
```

```

public String formatName(String name)
{
    char space = ' ';
    String comma = ",";
    name = name.replace(space, newChar: '+');
    name = name.replace(comma, replacement: "%2C");

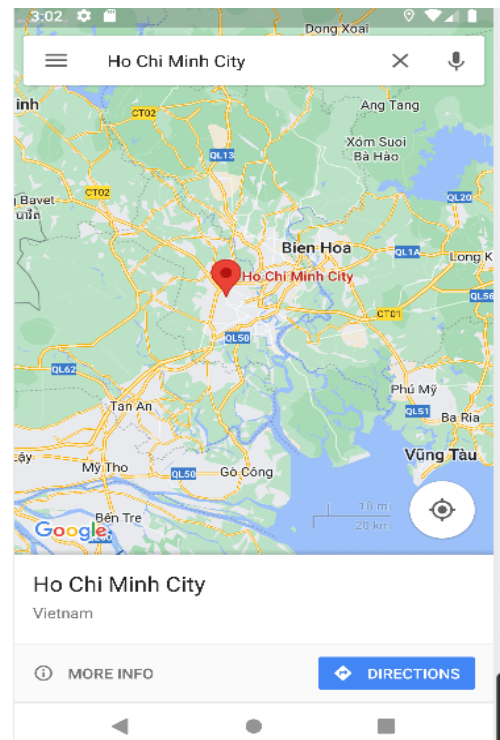
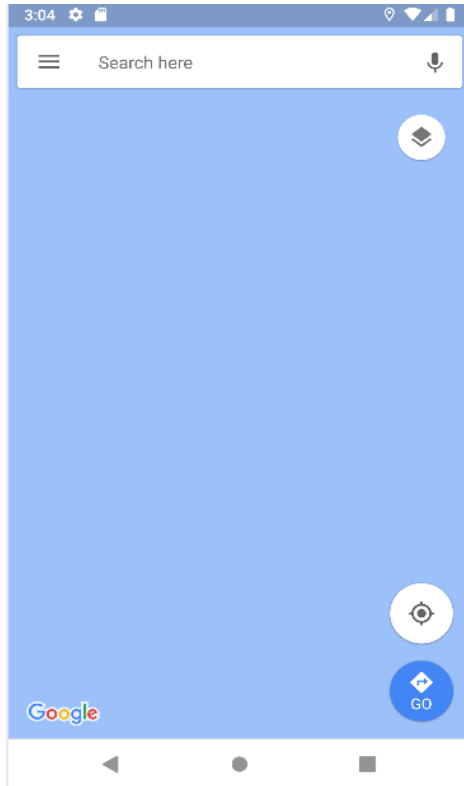
    return name;
};

```

We allow users to choose the mode of information that they want to input. Then, they can input the location information and press “Locate” button to find the location.

A screenshot of a mobile application interface. At the top, the status bar shows the time as 3:05. The app has a purple header bar. Below the header, there are two input fields: the first contains the text "100" and the second contains "-239". Below these fields are two purple buttons: "Locate" and "Locate with Name". The background is a light pink color.

A screenshot of a mobile application interface. At the top, the status bar shows the time as 3:01. The app has a purple header bar. Below the header, there is a text input field containing "Ho Chi Minh City". Below the input field are two purple buttons: "Locate" and "Locate with Coordinate". At the bottom, a keyboard is visible with the word "City" in the search bar. The background is a light pink color.



9. Play a media file

An Android application allows users to play a media file with the input artists, genres, titles, playlists, albums information of the song. This is the syntax that we need to run the application to play a song with all these attributes.

```

public void playPlaylist(String album, String artist, String genre, String playlist, String title)
{
    Intent intent = new Intent(MediaStore.INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH);
    intent.putExtra(MediaStore.EXTRA_MEDIA_FOCUS,
        MediaStore.Audio.Albums.ENTRY_CONTENT_TYPE);
    intent.putExtra(MediaStore.EXTRA_MEDIA_ALBUM, album);

    intent.putExtra(MediaStore.EXTRA_MEDIA_FOCUS,
        MediaStore.Audio.Artists.ENTRY_CONTENT_TYPE);
    intent.putExtra(MediaStore.EXTRA_MEDIA_ARTIST, artist);

    intent.putExtra(MediaStore.EXTRA_MEDIA_FOCUS,
        MediaStore.Audio.Genres.ENTRY_CONTENT_TYPE);
    intent.putExtra(MediaStore.EXTRA_MEDIA_GENRE, genre);

    intent.putExtra(MediaStore.EXTRA_MEDIA_FOCUS,
        MediaStore.Audio.Genres.ENTRY_CONTENT_TYPE);
    intent.putExtra(MediaStore.EXTRA_MEDIA_ALBUM, album);

    intent.putExtra(MediaStore.EXTRA_MEDIA_FOCUS,
        MediaStore.EXTRA_MEDIA_PLAYLIST);
    intent.putExtra(MediaStore.EXTRA_MEDIA_PLAYLIST, playlist);

    intent.putExtra(MediaStore.EXTRA_MEDIA_FOCUS,
        MediaStore.EXTRA_MEDIA_TITLE);
    intent.putExtra(MediaStore.EXTRA_MEDIA_TITLE, title);

    intent.putExtra(SearchManager.QUERY, playlist);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}

```

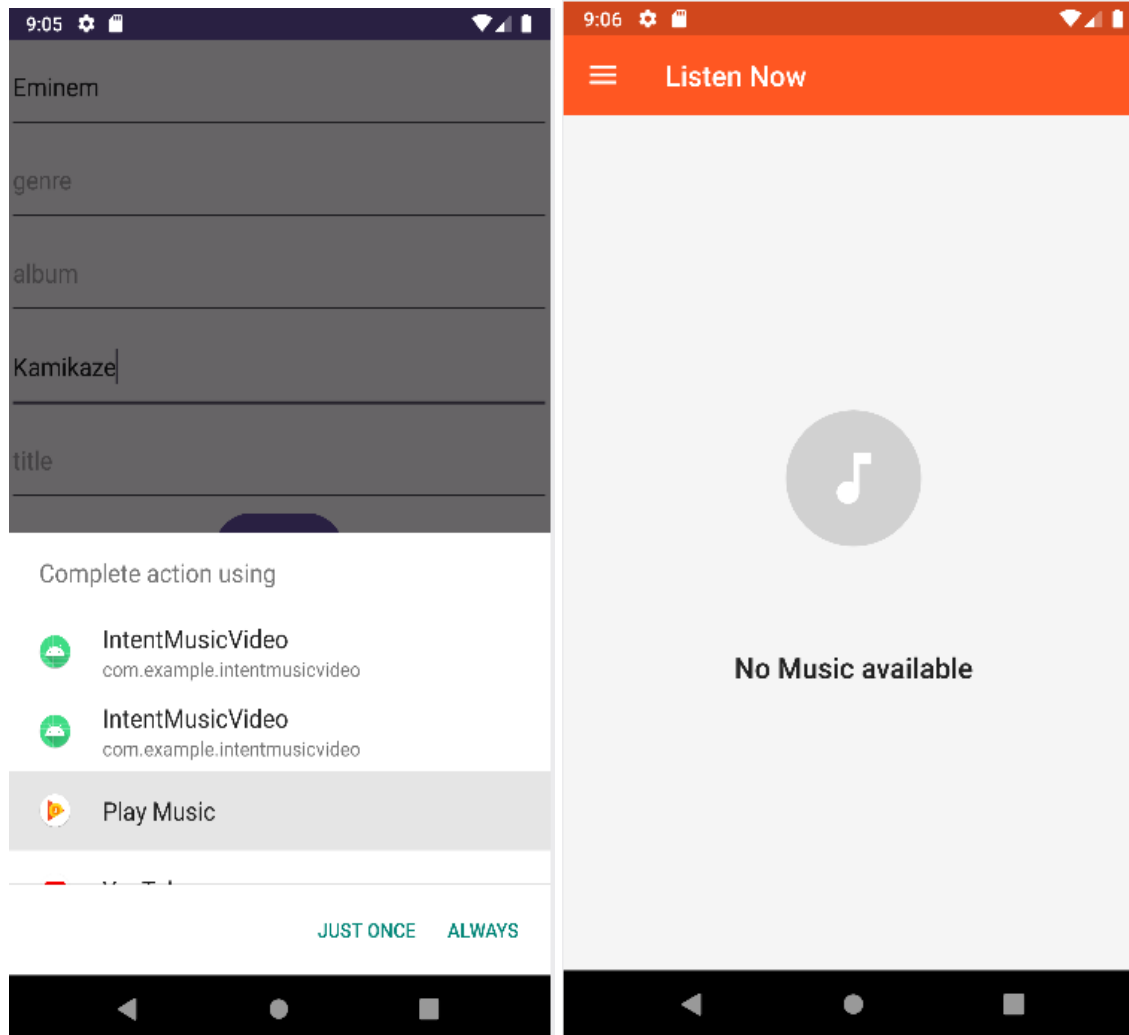
In the file AndroidManifest.xml, we also need to declare the following intent filter.

```

<intent-filter>
    <action android:name="android.media.action.MEDIA_PLAY_FROM_SEARCH" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>

```

To implement the activity, we need to have 5 edit texts to get the information from the user and a button to trigger the play activity. It will ask the user to pick a music app (for example: Spotify, Apple Music, YouTube Music) and play the corresponding music from the data of the input intent.



In this case, since there is no music media file available in the mobile app, the application just moves to the music playing application.

```
Intent intent = new Intent(MediaStore.INTENT_ACTION_MEDIA_PLAY_FROM_SEARCH);
intent.putExtra(MediaStore.EXTRA_MEDIA_FOCUS,
    MediaStore.Audio.Albums.ENTRY_CONTENT_TYPE);
intent.putExtra(MediaStore.EXTRA_MEDIA_ARTIST, "Eminem");
intent.putExtra(MediaStore.EXTRA_MEDIA_ALBUM, "Kamikaze");
intent.putExtra(SearchManager.QUERY, "Kamikaze");
}
```

The above example will play the album **Kamikaze** by American rapper **Eminem** in a chosen music app. The ***SearchManager.QUERY*** feature is **required** to proceed with the searching and the programmer should specify a search term closely related to the *SONG*, *ARTIST*, *ALBUM*, and *GENRE* features to give out the best result.



10. Send SMS message:

An Android application can direct user to choose which app to send SMS message (with attachment if needed). Here is a code snippet that can achieve this:

```

public static Intent sendMessage(String content) {
    Intent intent = new Intent(Intent.ACTION_SENDTO);
    intent.setData(Uri.parse("smsto:"));
    intent.putExtra("sms_body", content);
    return intent;
}

```

The ***setData*** configures the data scheme for SMS messaging (*smsto*). It will use the default messaging app on an Android phone (usually Google Messages on newer Android phones) or ask the user to pick one if there are multiple on the device. We use ***putExtra*** to add the content to the extra name of *sms_body* to declare the body of the message.

In *AndroidManifest.xml*, we declare the below `<queries>` tag to declare which outside application that the our application can work with:

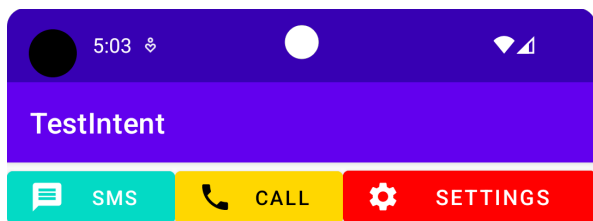
```

<queries>
<intent>
    <action android:name="android.intent.action.SENDTO" />
    <data android:scheme="smsto"/>
</intent>
</queries>
<uses-permission android:name="android.permission.SEND_SMS" />

```

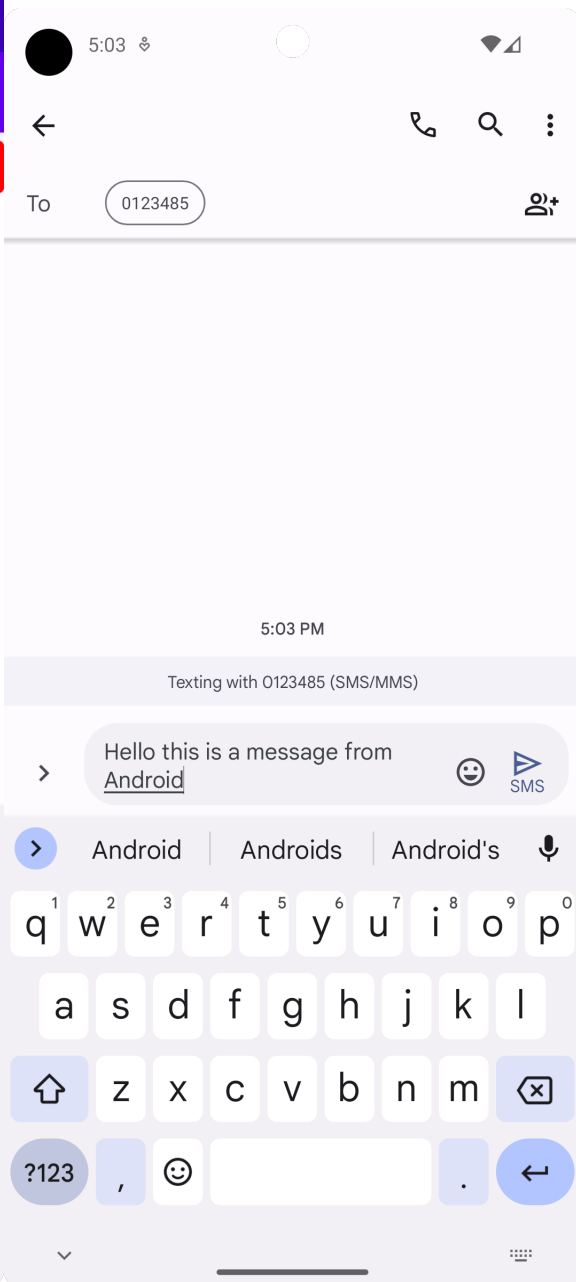
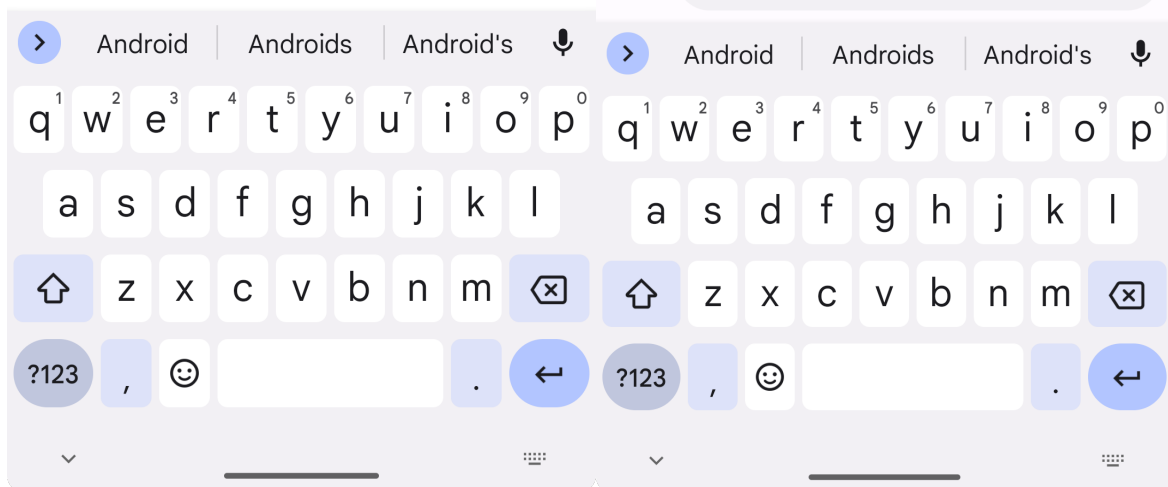
This also ensures the intent to work after Android 11.

After starting the activity from the intent, it will open the corresponding messaging app, ask the user to pick recipients, and the user has the last opportunity to modify the content of the message before hitting the Send button.



Hello this is a message from Android

SEND



11. Call to a specified phone number:

An Android application can initiate a call to a specified phone number. We can use the **setData** to specify the data scheme (*tel* for plain calling or *voicemail* for voicemailing).

```
public static Intent call(String phoneNumber) {
    Intent intent = new Intent(Intent.ACTION_DIAL);
    intent.setData(Uri.parse("tel:" + phoneNumber));
    return intent;
}
```

The data URI scheme has the syntax:

tel / voicemail: *<phone number>*

To execute the intent, the **CALL_PHONE** permission must be declared in the *AndroidManifest* file. Therefore we declare the *<queries>* tag and its permission:

```
<queries>
    <intent>
        <action android:name="android.intent.action.DIAL" />
        <data android:scheme="tel" />
    </intent>
</queries>

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

After starting the intent, it will open the Phone app of the operating system with the already-filled-in phone number in the dialpad interface. The user can change the phone number or click Call.

5:14

TestIntent

SMS

CALL

SETTINGS

999

CALL

1

2

3

-

4

5

6

⌋

7

8

9

✖

,

0

.

✓

5:15

Create new contact

Add to a contact

Send a message

999

1
ABC

2
DEF

3
MNO

4
PQRS

5
TUV

6
WXYZ

*

0
+

#

Call

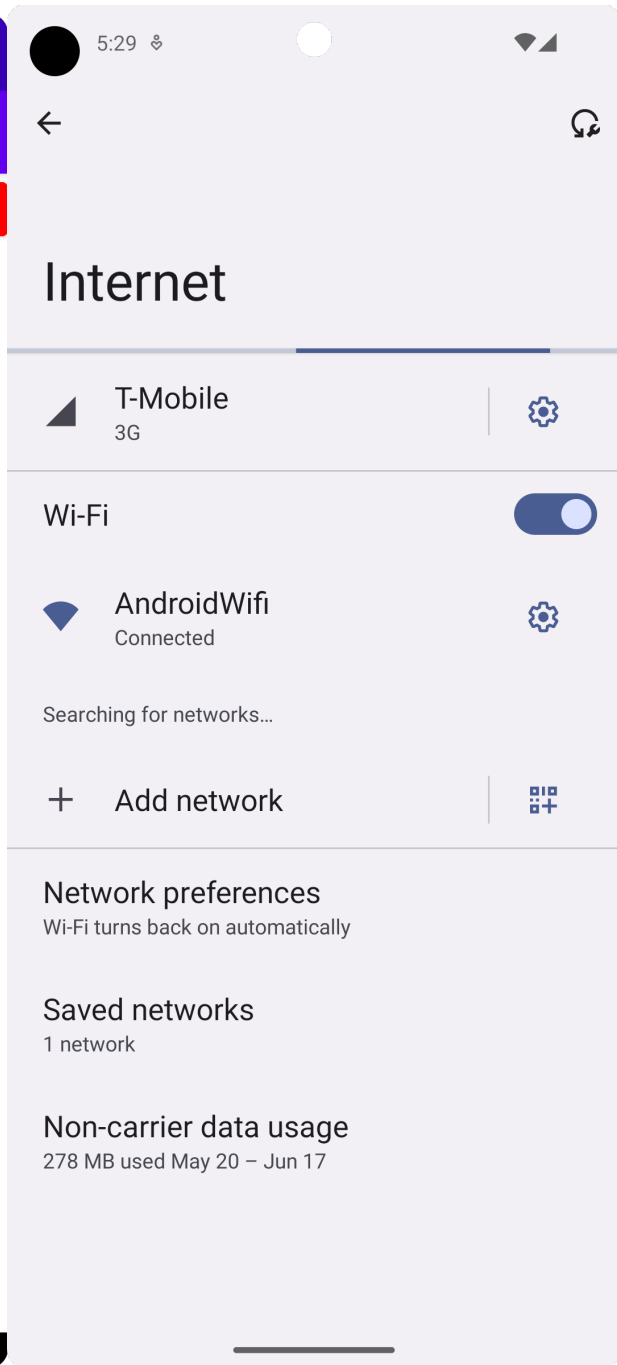
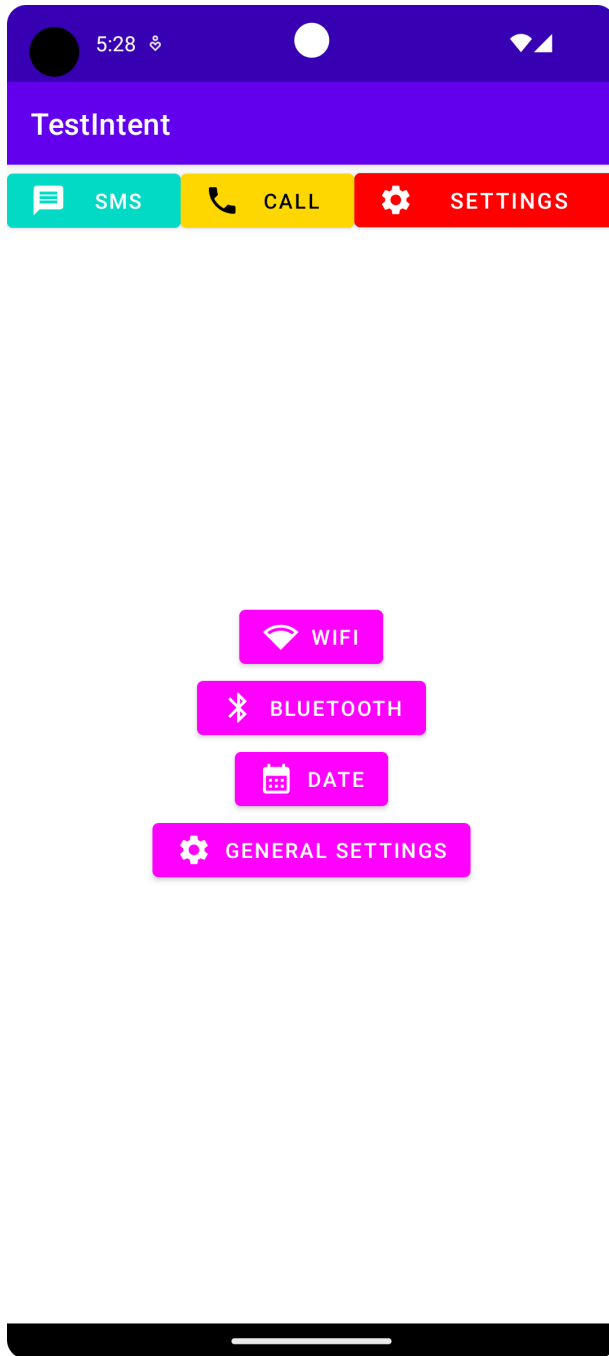
12. Settings

In practical use, the functionality of an Android application can only be executed after modifying a specific setting of the device (for example: Internet, Bluetooth, etc.). Therefore, Android developing supports a common intent for accessing the *Settings* of the device. Furthermore, the application can open to a specific section of the Settings. Here are some example *Settings ACTIONS*:

ACTION_SETTINGS	General Settings of Android
ACTION_WIRELESS_SETTINGS	Wireless connection (Wifi, Bluetooth, 5G, 4G)
ACTION_AIRPLANE_MODE_SETTINGS	Airplane mode
ACTION_WIFI_SETTINGS	Wifi connection
ACTION_BLUETOOTH_SETTINGS	Bluetooth connection
ACTION_DATE_SETTINGS	Datetime settings
ACTION_DISPLAY_SETTINGS	Display settings
ACTION_INPUT_METHOD_SETTINGS	Input method for keyboards settings

Here is the generalized syntax:

```
Intent(Settings.<Action name (from the above table)>);
```



Example of Wifi settings