

Navegación y Bases de Datos

Projecte Integrat de Software (PIS)

Universitat de Barcelona

victor.campello@ub.edu

c.izquierdo@ub.edu

carlos.martinisla@ub.edu

18-20 de Febrero de 2020

1 Intents

- Definición
- Ciclo de transmisión de los Intents
- Selector de apps
- Filtros de Intents
- Cómo crear un Intent

2 Ejercicios

3 Datos de la web

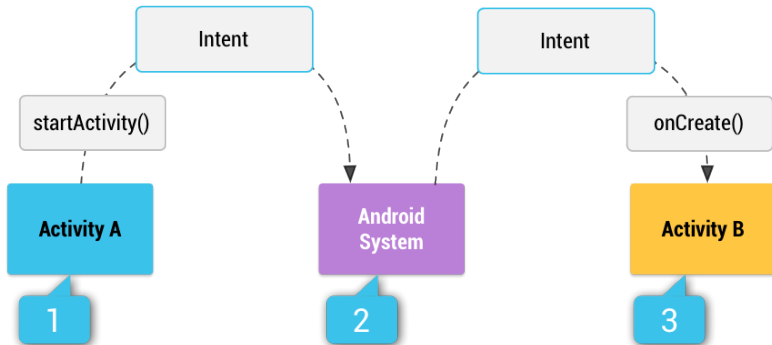
Una **Intent** es un objeto de mensajería que puedes usar para solicitar una acción de otro componente de una app.

Dos tipos de *Intents*:

- **Intents explícitas:** Especifican que aplicación administra (recibe) el intent. Se usan normalmente para iniciar actividades dentro de una misma app, por el simple hecho que en el propio intent se puede determinar con el nombre(conocido) de la clase o actividad que llama.
- **Intents implícitas:** No nombran ningún componente o clase en específico, así puede usarse (interpretarse) por otras apps del sistema. Ejemplo: cuando de una app necesitas abrir el mapa, el intent se traslada a otra aplicación que permita abrir mapas. En este caso, al ser implícita, Google Maps puede leer la información y ejecutar la acción.

Intent específico vs Intent genérico

Intent - Ciclo de transmisión



Cómo crear un Intent

Un objeto **Intent** tiene información que el sistema Android usa para determinar qué componente debe iniciar (como el nombre exacto del componente o la categoría que debe recibir la intent), además de información que el componente receptor usa para realizar la acción correctamente (por ejemplo, la acción que debe efectuar y los datos en los que debe actuar).

Los campos a crear para definir una intent són los siguientes:

- ➊ **Nombre del componente.** Componente que se debe iniciar.
Opcional. Lo que define una intent como explícita.
- ➋ **Acción** String que define la acción genérica a realizar. (View, Send, etc)
- ➌ **Datos** Tipo de datos a manejar por la aplicación que genera y/o recibe
- ➍ **Categoría** String con información sobre los datos a manejar.

Existe información extra que puede ser asociada a un Intent. Para más información, podéis consultar este enlace: <https://developer.android.com/guide/components/intents-filters>

Ejemplos de Intents

1 Intent explícita

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
val downloadIntent = Intent(this, DownloadService::class.java).apply {
    data = Uri.parse(fileUrl)
}
startService(downloadIntent)
```

2 Intent implícita

```
// Create the text message with a string
val sendIntent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, textMessage)
    type = "text/plain"
}

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(packageManager) != null) {
    startActivity(sendIntent)
}
```

Selector de apps

Cuando creas un intent implícito, necesariamente pueden existir varias aplicaciones que puedan ejecutar el Intent. Para ello, a veces requerimos de diseñar un selector que nos permita escoger la aplicación con la que queremos interactuar.

```
val sendIntent = Intent(Intent.ACTION_SEND)
...

// Always use string resources for UI text.
// This says something like "Share this photo with"
val title: String = resources.getString(R.string.chooser_title)
// Create intent to show the chooser dialog
val chooser: Intent = Intent.createChooser(sendIntent, title)

// Verify the original intent will resolve to at least one activity
if (sendIntent.resolveActivity(packageManager) != null) {
    startActivity(chooser)
}
```

Para mostrar el diálogo de selección, crea una Intent usando `createChooser()` y transfírela a `startActivity()`, tal como se muestra en el siguiente ejemplo. Aquí se muestra un diálogo con una lista de apps que responden a la intent transferida al método `createChooser()`, con el texto proporcionado como título del diálogo.

En el proceso de recibir una Intent, debes definir que parámetros ha de tener para poder recibir en una actividad determinada. Para esto se diseñan filtros, con un elemento `<intent-filter>` declarados en el `<manifest.xml>`. Un componente de aplicación debe declarar filtros independientes para cada tarea única que puede hacer. Se deben especificar estos elementos:

- `<action>`
- `<data>`
- `<category>`

Ejemplo de filtros

```
<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

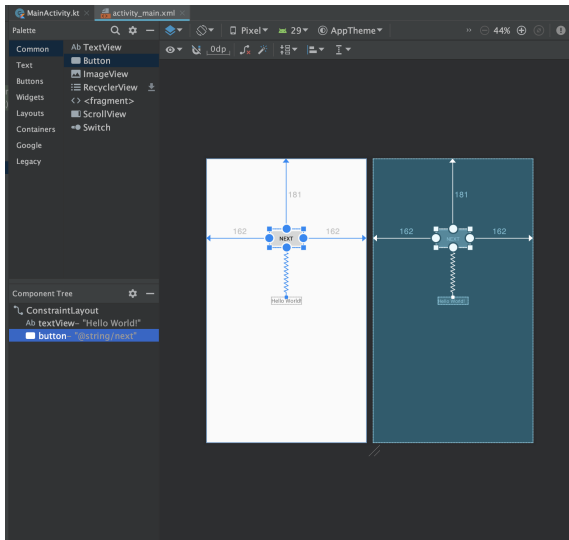
<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <action android:name="android.intent.action.SEND_MULTIPLE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>
```

Cómo crear un Intent: Ejemplo

- 1 Crea un nuevo proyecto llamado Lab3PIS2020
- 2 Disenya un layout con un Boton
- 3 Crea otro layout asociado a otra actividad
- 4 Asocia la creación del intent al click del boton
- 5 Evalua en el emulador

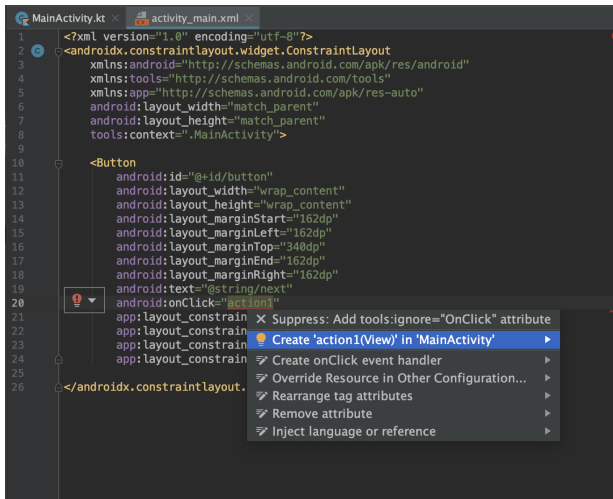
Paso 1

Genera un layout con un Boton en la posición deseada.



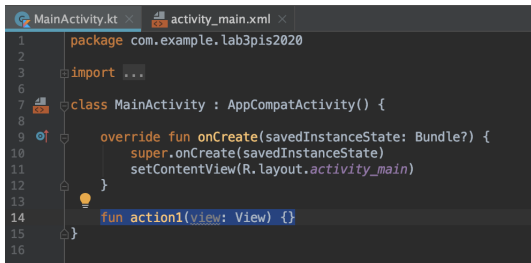
Paso 2

En el editor Text de layout, crea la instancia **android:onClick** y ponle un nombre.



Paso 3

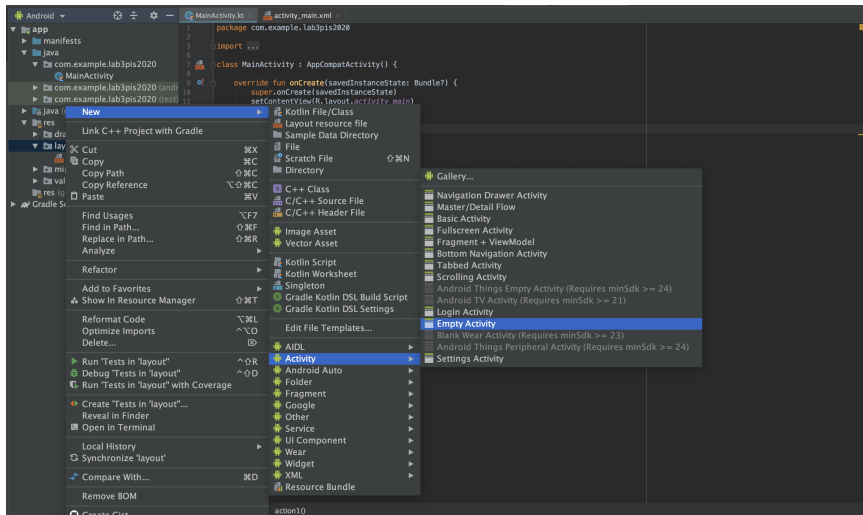
Si observais el archivo MainActivity.kt veréis cómo se ha generado la función onClick creada en el paso anterior.

A screenshot of an IDE showing the MainActivity.kt file. The code is in Kotlin and defines a MainActivity class that inherits from AppCompatActivity. It overrides the onCreate method and includes an action1 function. The line number indicator on the left shows lines 1 through 16. The package is com.example.lab3pis2020. The onCreate method calls super.onCreate and setContentView. The action1 function is currently selected with a blue highlight.

```
1 package com.example.lab3pis2020
2
3 import ...
4
5
6
7 class MainActivity : AppCompatActivity() {
8
9     override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11         setContentView(R.layout.activity_main)
12     }
13
14     fun action1(view: View) {}
15 }
16
```

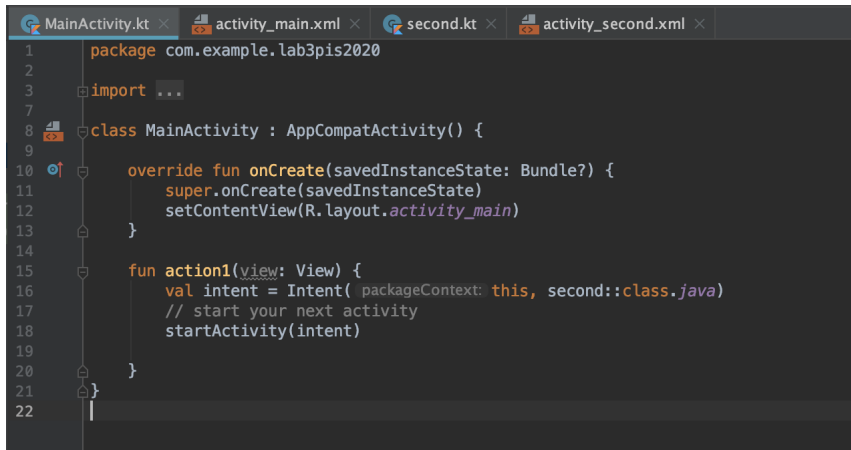
Paso 4

Una vez creada la función, hay que crear otra actividad con su respectivo layout. Se ha de crear a través de Android menu.



Paso 5

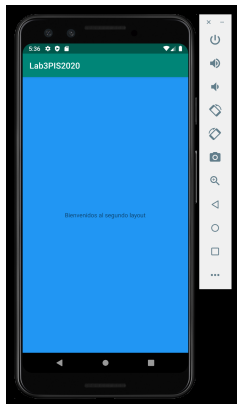
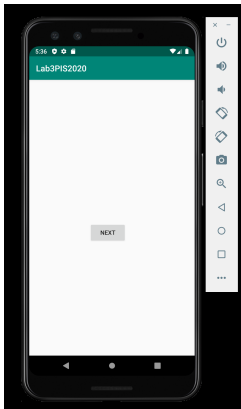
Crear el intent que hace **startActivity()** en MainActivity.kt



```
1 package com.example.lab3pis2020
2
3 import ...
4
5
6
7
8 class MainActivity : AppCompatActivity() {
9
10     override fun onCreate(savedInstanceState: Bundle?) {
11         super.onCreate(savedInstanceState)
12         setContentView(R.layout.activity_main)
13     }
14
15     fun action1(view: View) {
16         val intent = Intent(packageContext: this, second::class.java)
17         // start your next activity
18         startActivity(intent)
19     }
20
21 }
22
```


Paso 6

Prueba en el emulador si el boton te traslada a la siguiente actividad/layout.



Mostrar un mensaje 1

Primero crear los siguientes mensajes dentro de la función.

```
const val EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE"

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

    /** Called when the user taps the Send button */
    fun sendMessage(view: View) {
        val editText = findViewById<EditText>(R.id.editText)
        val message = editText.text.toString()
        val intent = Intent(this, DisplayMessageActivity::class.java).apply {
            putExtra(EXTRA_MESSAGE, message)
        }
        startActivity(intent)
    }
}
```

Mostrar un mensaje 2

En la segunda actividad, ubicar el TextView y asignarle valor. Crear variable message que contiene el texto del intent de la actividad anterior.

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_display_message)  
  
    // Get the Intent that started this activity and extract the string  
    val message = intent.getStringExtra(EXTRA_MESSAGE)  
  
    // Capture the layout's TextView and set the string as its text  
    val textView = findViewById<TextView>(R.id.textview).apply {  
        text = message  
    }  
}
```

Crear barra ascendiente

Crear una barra que permita deshacer el paso y volver atrás.

```
<activity android:name=".DisplayMessageActivity"
    android:parentActivityName=".MainActivity">
    <!-- The meta-data tag is required if you support API level 15 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

Generar intent implícita

Ahora generaremos una intent implícita que nos abra el buscador google por defecto del sistema. En este caso, 3 importantes detalles:

- ❶ Al ser implícita, el parámetro *packageContext* queda vacío.
- ❷ No declaramos la Actividad que llama.
- ❸ Estamos usando *Uri: Uniform Resource Identifier*. Nos sirve para llamar datos externos, en este caso, parsear una URL.
- ❹ Estamos declarando una acción. En este caso, mostrar la app que por defecto su filtro permita abrir la actividad.

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

fun showWebPage(view: View){
    val intent2 = Intent(Intent.ACTION_VIEW, Uri.parse( urlString: "https://www.google.com"))
    startActivity(intent2)
}
```

Compartir imagen

Código para compartir una imagen. Intent implícita.

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        val shrButton = findViewById<Button>(R.id.bt1)  
  
        shrButton.setOnClickListener { it: View!  
            val imagesample1 = findViewById<ImageView>(R.id.image1)  
            val imagesample = imagesample1.drawable  
  
            val bitmap = (imagesample as BitmapDrawable).bitmap  
  
            val file = File(externalCacheDir, child: "shaq.jpg")  
            val fOut = FileOutputStream(file)  
            val intent = Intent(Intent.ACTION_SEND)  
  
            intent.putExtra(Intent.EXTRA_STREAM, Uri.fromFile(file))  
            intent.type = "image/jpeg"  
            intent.putExtra(Intent.EXTRA_SUBJECT, value: "Subject here")  
  
            startActivity(Intent.createChooser(intent, title: "share image"))  
        }  
    }  
}
```

- 1 Cread un intent simple que traspase entre 3 pantallas, con su respectiva barra de acción y botón *Atrás*
- 2 Añadid al penúltimo ejemplo de intent implicito la opción para navegar directamente en la URL introducida por vosotros una vez pulsado el botón.
- 3 Trabajad en grupo y empezad a diseñar la interfície de usuario de vuestro proyecto grupal.

API

Vamos a ver un ejemplo de cómo obtener datos externos via una API. Una API es una *Application Programming Interface*, una interfície entre programas. En la práctica, es una “web” a la que podemos hacer “llamadas” (*requests*) para obtener información.

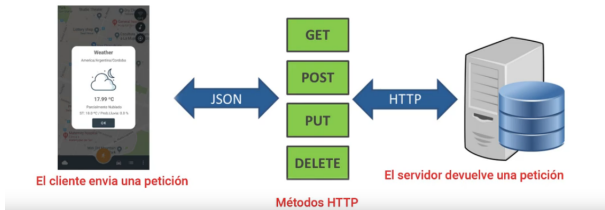


Figure: Esquema de una API

Hay diferentes métodos para llamar a la API según lo que necesitemos. El nombre de cada método es obvio de su nombre en inglés.

Datos de la web

Por ejemplo, Twitter tiene una API para obtener *Tweets* de forma programática, en formato JSON:

```
1 # Request Tweets from last 7 days
2 # Search query: from:Nasa OR #nasa
3
4 twurl "/1.1/search/tweets.json?q=from%3ANasa%20OR%20%23nasa"
5
6 # Response, an array of Tweet JSON:
7
8 {
9   "statuses": [
10     {
11       "created_at": "Wed Apr 12 04:53:25 +0000 2017",
12       "id": 852021818290352129,
13       "id_str": "852021818290352129",
14       "text": "Watch NASA's first 4K broadcast from space on April 26th - E",
15       "truncated": false,
16       "entities": {
17         "hashtags": [
18
19         ],
20         "symbols": [
21
22         ],
23         "user_mentions": [
24
25         ],
```

Figure: Resultado de una consulta a la API de Twitter.

Datos de la web

En el presente ejemplo vamos a utilizar una API de datos sobre calidad del aire: *OpenAQ* (<https://openaq.org>).

Lo primero que debemos hacer es añadir una librería para hacer las llamadas HTTP, *Volley*, en el *build.gradle* de nuestra app y sincronizar de nuevo el proyecto.

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation androidx.appcompat.appcompat:1.0.2
    implementation androidx.core.core-ktx:1.0.2
    implementation androidx.constraintlayout.constraintlayout:1.1.3
    implementation "com.android.volley:volley:1.1.1"
    testImplementation "junit:junit:4.12"
```

En la documentación de Android hay ejemplos sobre cómo implementar este tipo de llamadas [docs]. Nosotros usaremos lo siguiente:

```
val textView: TextView = findViewById(R.id.contentTV)
// Instantiate the RequestQueue.
val queue = Volley.newRequestQueue(context: this)
val url = "http://www.google.com"

// Request a string response from the provided URL.
val stringRequest = StringRequest(
    Request.Method.GET, url,
    Response.Listener<String> { response ->
        // Display the first 500 characters of the response string.
        textView.text = "Response is: ${response.substring(0, 500)}"
    },
    Response.ErrorListener { volleyError ->
        textView.text = "That didn't work! Response: ${volleyError}" })

// Add the request to the RequestQueue.
queue.add(stringRequest)
```

No obstante, esta llamada aún no funcionará, ya que no hemos pedido el permiso para conectarnos a internet desde la App.

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

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

*Quizá tengáis que desinstalar la App del emulador y volverla a instalar. Después de estos cambios, deberías ser capaz de ver los primeros 500 caracteres de la respuesta en el TextView.

Datos de la web

Si cambiamos ahora la URL por la de nuestra API, podremos obtener contenido sobre polución. En la documentación de OpenAQ podéis encontrar información sobre las diferentes llamadas disponibles [web]

GETTING STARTED

CITIES

GET

COUNTRIES

GET

FETCHES

GET

LATEST

GET

LOCATIONS

GET

PUT

MEASUREMENTS

GET

PARAMETERS

GET

SOURCES

GET

Measurements - GET

Provides data about individual measurements

GET

```
https://api.openaq.org/v1/measurements
```

Parameter

Field	Type	Description
country	optional string	Limit results by a certain country.
city	optional string	Limit results by a certain city.
location	optional string	Limit results by a certain location.
parameter	optional string	Limit to certain one or more parameters (ex. <code>parameter=pm25</code> or <code>parameter[]=co&parameter[]=pm25</code>) Allowed values: <code>pm25</code> , <code>pm10</code> , <code>so2</code> , <code>no2</code> , <code>o3</code> , <code>co</code> , <code>bc</code>
has_geo	optional boolean	Filter out items that have or do not have geographic information. Allowed values: <code>true</code> , <code>false</code>
coordinates	optional string	Center point (<code>lat</code> , <code>lon</code>) used to get measurements within a certain area. (ex. <code>coordinates=40.23,34.17</code>)
radius	optional number	Radius (in meters) used to get measurements within a certain area, must be used with <code>coordinates</code> . Default value: <code>2500</code>
value_from	optional number	Show results above value threshold, useful in combination with <code>parameter</code> .
value_to	optional number	Show results below value threshold, useful in combination with <code>parameter</code> .

Nosotros usaremos la llamada *measurements*.

Una vez procesada la llamada y organizado el text, el resultado debería ser algo similar a la siguiente imagen:



End