# Tema 2.2 - Entendiendo la estructura básica

Descargar estos apuntes pdf o html

## Índice

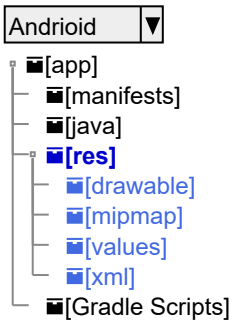
# Introducción

En este tema vamos a ver cómo se estructura una aplicación Android y cómo se relacionan estos componentes entre sí. Pero antes de empezar con la capa más laboriosa de UI, vamos a ver cómo se estructura una aplicación Android a nivel de ficheros y a profundizar en algunos elementos que ya hemos visto al renombrar un proyectos básico, además de otros componentes esenciales para el funcionamiento de una aplicación Android.

# Recursos de la aplicación

Los recursos son archivos o datos externos que soportan el comportamiento de nuestra aplicación Android, por ejemplo **imágenes**, **strings**, **colores**, **temas**, etc.

Formalmente en un proyecto, estos elementos se encuentran en la carpeta `res`. Allí se encuentran subdirectorios que agrupan los diferentes tipos de recursos.




La idea del uso de recursos es dividir el código de tu app para mantener independencia. Todo con el fin de agregar variaciones de los archivos para adaptar la aplicación a diferentes tipos de pantallas, idiomas, versiones, dimensiones, configuraciones de orientación, etc. *Por ejemplo: no es lo mismo el espacio de un teléfono móvil que el de una tableta, sin embargo se pueden crear variaciones de los recursos para que la aplicación se adapte a la densidad de pantalla de cada uno.*

Los grupos de recursos se dividen en subdirectorios. Cada uno de ellos contiene archivos que cumplen una función específica dentro de la aplicación y en ocasiones estos directorios de recursos dependerá del tipo de desarrollo que estemos haciendo. Por ejemplo, si estamos usando XML para crear la interfaz de usuario, tendremos los directorios `layout` y `menu` donde se guardarán los archivos XML que contienen la interfaz de usuario y los menús de la aplicación respectivamente.

Debes respetar esta estructura de documentos para no tener problemas en la ejecución. En la siguiente **Tabla de Recursos**, podemos ver un resumen de los más usados.

En nuestra plantilla de aplicación de compose tenemos los siguientes recursos ...

## Recursos alternativos

Un recurso alternativo es una variación de un recurso, que se ajusta a una característica de configuración en el dispositivo móvil donde se ejecuta la aplicación. Esto se logra a través de una **tabla** de calificadores creada por Google que estandariza todas las configuraciones posibles que se puedan presentar.

Un calificador es un mecanismo gramatical que especifica el propósito de un recurso. Su sintaxis es la siguiente: `<nombre_recurso>-<calificador>`

> **Ejemplo:** `mipmap-hdpi/` contendrá la variación del recurso `ic_launcher.png` para dispositivos con densidad de pantalla de `240 dpi` .

Todos los archivos que correspondan al mismo recursos deberán de tener el mismo nombre, se identificarán gracias al calificador añadido al identificador de carpeta.

Si hay que indicar **más de un calificador**, se separan por guiones y el orden de preferencia es el que se sigue en la tabla, el primer calificador será el de más preferencia.

> **Ejemplo:** Dispositivos en **inglés de Estados Unidos** y en **orientación horizontal**, tendrán un nombre de recurso alternativo en el `drawable/` , de la siguiente manera `drawable-en-rUS-land/` .

Una vez que guardes los recursos alternativos en directorios denominados con estos calificadores, Android aplicará automáticamente los recursos en tu aplicación de acuerdo con la configuración del dispositivo actual.

## Drawables ( `drawable/` )

Contiene los **elementos de diseño** ('*dibujables*') en nuestra aplicación. Iremos usándolos a lo largo del curso.

- Archivos de mapas de bits o vectoriales
- Archivos nine-patch (mapas de bits reajustables)
- Listas de estados
- Formas
- Elementos de diseño de animaciones
- Otros elementos de diseño

## Mipmaps ( `mipmap/` )

Son archivos de diseño para **diferentes densidades de los íconos de selectores**.

Android Studio crea un **icono adaptativo** para su aplicación que solo está disponible en SDK 26 y posteriores. Estos iconos adaptativos se incluyen en la carpeta `mipmap-anydpi-v26`, son archivos `.xml` que se adaptan a la tecnología del dispositivo.

> **Ejemplo**: Un ícono de selector adaptable se puede mostrar con una forma circular en un dispositivo OEM y con un cuadrado con esquinas redondeadas en otro.

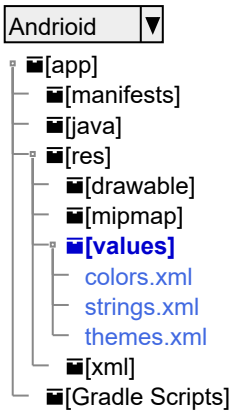
La diferencia con respecto a los contenidos de `drawable\` es que esta última sirve para colocar todas las imágenes que vaya a utilizar nuestra aplicación, siguiendo el mismo esquema de sufijos que mipmap. Mientras que en la carpeta `mipmap\` únicamente colocaremos el icono de la app.

Como podemos observar, cuando creamos un proyecto en Android Studio, **por defecto**, ya nos coloca ahí el icono con el nombre de `ic_launcher`.

Lo más común es encontrar recursos alternativos dependientes de la densidad y la versión del SDK.

| Densidad | Descripción | DPI | Dispositivos | Ejemplo |
|---|---|---|---|---|
| **ldpi** | Baja densidad | 120 | 240x320 | Android watch |
| **mdpi** | Densidad media | 160 | 320x480 | Teléfonos básicos |
| **hdpi** | Alta densidad | 240 | 480x800 | Smartphones antiguos |
| **xhdpi** | Extra alta densidad | 320 | 720x1280 | Smartphones modernos |
| **xxhdpi** | Extra extra alta densidad | 480 | 1080x1920 | Smartphones premium |
| **xxxhdpi** | Extra extra extra alta densidad | 640 | 1440x2560 | Televisión 4K |
| **nodpi** | Sin densidad | 0 | | |
| **anydpi** | Cualquier densidad | 0 | | |
| **anydpi-v26** | Cualquier densidad para SDK 26 y posteriores | 0 | | |

## Values ( `values/` )




Son archivos en formato XML que contienen valores simples, como cadenas, valores enteros y colores.

Los archivos de recursos XML en otros subdirectorios `res/` definen un único recurso basado en el nombre del archivo en formato XML, mientras que los archivos del directorio `values/` describen **varios recursos**.

Dado que cada recurso se define con su propio elemento XML, puedes asignar el nombre que desees al archivo y colocar diferentes tipos de recursos en un archivo. Sin embargo, para mayor claridad, es recomendable que coloques tipos de recursos únicos en diferentes archivos. Por ejemplo, a continuación, se incluyen los archivos de recurso que se crean en la **platilla de compose**:

## Colores ( `colors.xml` )

Definirá nombres de **colores** a usar en nuestra aplicación o nuestros temas. Por ejemplo ...

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    ...
</resources>
```

## Textos ( `strings.xml` )

En lugar de '*hardcodear*' los **textos** en nuestra aplicación, los definiremos en este archivo. Por ejemplo ...

```
<!-- string.xml dentro de res → values -->
<resources>
    <string name="app_name">HolaMundo</string>
</resources>
```

Si quisiéramos tener diferentes traducciones pulsaríamos el botón derecho sobre el archivo `strings.xml` y seleccionaríamos `Open Translations Editor` en el menú contextual. En la ventana que se abre podemos añadir diferentes traducciones para los textos de nuestra aplicación.

El editor básicamente me creará un **recurso alternativo** `strings.xml` por cada idioma que añada. Por ejemplo, si añado el idioma inglés de UK, me creará un fichero `strings.xml` en el directorio `values-en-rGB\` con el siguiente contenido...

```
<resources>
<!-- string.xml dentro de res → values-en-rGB -->
    <string name="app_name">HelloWorld</string>
</resources>
```

> 📌 **Nota**: Si quieres sabes más de cómo localizar tu aplicación usando el ***'Editor de Traducciones`*** de android estudio. Puedes consultar la **documentación oficial**.

A nivel organizativo creará una estructura de directorios `values` con el sufijo de la localización según la ISO 639-1 y la ISO 3166-1. Por eso es interesante usar el ***'Editor de Traducciones`*** en lugar de hacerlo manualmente. Una vez, creado ya lo podríamos modificar a mano. Aunque no es recomendable porque el editor de traducciones me gestiona aquellas que están pendientes de traducir.

En el siguiente esquema puedes ver la estructura de directorios que se crea en la vista `Proyecto` y la estructura organizativa equivalente en la vista `Android`.

### Temas ( `themes.xml` )

Los **estilos** XML son este caso una herencia de estilos que se aplican a la aplicación o a un componente concreto. En el tema de **Capa UI** daremos alguna pincelada más sobre los temas en Compose.

## Acceder a los recursos de la aplicación

Ahora que se ha entendido que son los recursos, vamos a entender como pueden ser accedidos en el código Kotlin o en el código XML de otros recursos. Para obtener la referencia de un recurso **es necesario que se le asigne un identificador que lo diferencie** y le indique al programador donde se encuentra.

Cada identificador se ubica en una clase llamada `R` a través de una constante entera. Esta clase es **generada automáticamente** por una herramienta del SDK llamada appt, por lo que **no es recomendable editarlo manualmente**.

Cada tipo de **recurso existente** dentro de la carpeta `res` **es una clase anidada estática** del archivo `R.class`.

> **Ejemplo**: El subdirectorio `drawable/` puede ser accedido como `R.drawable.`

### Accediendo desde Kotlin

Ahora, si deseas obtener el identificador de un recurso desde código Kotlin, entonces usas el operador punto para acceder a los miembros de cada clase interna `Paquete.R.tipoRecurso.nombreRecurso`.

> **Ejemplo**: Si tenemos una imagen cuyo identificador es `ic_launcher_background` . Para acceder a su contenido debes navegas de la siguiente forma `R.drawable.ic_launcher_background` .
>
> Por ejemplo, aunque hablaremos de '*Compose*' cuando lleguemos a la capa de UI, aquí tienes un fragmento de código para crear un componente con una imagen a partir del recurso del ejemplo.

```
@Composable
fun MiApp() {
    Image(
        // Reemplaza "R.drawable.ic_launcher_background" con el ID de tu recurso drawable
        painter = painterResource(id = R.drawable.ic_launcher_background),
        contentDescription = null,
        modifier = Modifier
            .fillMaxSize()
            .background(Color.White)
    )
}
```

## Accediendo desde otro recurso XML

Esta forma de referencia se da cuando dentro de un recurso definido en un archivo XML necesitamos usar el valor de otro recurso. Para ello usamos la sintaxis `@[Paquete:]tipoRecurso/nombreRecurso`

**Ejemplo**: Si tenemos una imagen cuyo identificador es `ic_launcher_background`. Para acceder a su contenido solo navegas de la siguiente forma `@drawable/ic_launcher_background`.

Por ejemplo, **aunque este curso no vamos a abordar la interfaces de usuario con XML**, aquí tienes un fragmento de código para crear una vista con una imagen a partir del recurso del ejemplo.

```
<ImageView android:background="@drawable/ic_launcher_background" />
```

# Activities (Actividad)

- **Documentación oficial**.

Las actividades son uno de los **componentes fundamentales** de las apps en la plataforma de Android. Sirven como **punto de entrada para la interacción del usuario** con una app.

Una actividad es una **ventana** que contiene la **interfaz de usuario** de una app. Por lo general, cada actividad tiene una **pantalla completa** que ocupa toda la superficie de la pantalla.

Tradicionalmente con los interfaces de usuario basadas en XML podíamos tener una o varias actividades entre las cuales se podía navegar. Posteriormente apareció el concepto de **fragmento** que es una parte de una actividad que se puede reutilizar en otras actividades e incluso en otras aplicaciones. De esta forma se creaba **una única actividad** que contenía varios fragmentos y se podía navegar entre ellos.

Con JetPack Compose y la posibilidad de crear interfaces de usuario de forma declarativa, se ha vuelto a la idea de tener una única actividad que contiene la interfaz de usuario completa de la aplicación. En este caso la navegación entre pantallas se realiza mediante la **composición de vistas**. A este tipo de aplicaciones se les llama **Single Activity Apps**. Profundizaremos más adelante en la navegación basada en composición de vistas.

## El concepto de actividades

- **Documentación oficial**
- Recursos Adicionales:
  - **Castellano**: **DevExperto**
  - **Inglés**: **Philipp Lackner**

La experiencia con la app para dispositivos móviles difiere de la versión de escritorio, ya que la interacción del usuario con la app no siempre comienza en el mismo lugar. En este caso, no hay un lugar específico desde donde el usuario comienza su actividad. Por ejemplo, si abres una app de correo electrónico desde la pantalla principal, es posible que veas una lista de correos electrónicos. Por el contrario, si usas una app de redes sociales que luego inicia tu app de correo electrónico, es posible que accedas directamente a la pantalla de la app de correo electrónico para redactar uno.

La clase Activity está diseñada para facilitar este paradigma. Cuando una app invoca a otra, la app que realiza la llamada invoca una actividad en la otra, en lugar de a la app en sí. De esta manera, la actividad sirve como el punto de entrada para la interacción de una app con el usuario. Implementas una actividad como una subclase de la clase Activity.

Una actividad proporciona la ventana en la que la app dibuja su IU. Por lo general, esta ventana llena la pantalla, pero puede ser más pequeña y flotar sobre otras ventanas. Generalmente, una actividad implementa una pantalla en una app. Por ejemplo, una actividad de una app puede implementar una pantalla Preferencias mientras otra implementa una pantalla Seleccionar foto.

La mayoría de las apps contienen varias pantallas, lo cual significa que incluyen varias actividades. Por lo general, una actividad en una app se especifica como la actividad principal, que es la primera pantalla que aparece cuando el usuario inicia la app. Luego, cada actividad puede iniciar otra actividad a fin de realizar diferentes acciones. Por ejemplo, la actividad principal de una app de correo electrónico simple podría proporcionar una pantalla en la que se muestra una casilla de correo electrónico. A partir de aquí, la actividad principal podría iniciar otras actividades que proporcionan pantallas para tareas como redactar correos y abrir correos electrónicos individuales.

Si bien las actividades trabajan en conjunto a fin de crear una experiencia del usuario coherente en una app, cada actividad se relaciona vagamente con otras actividades; por lo general, hay una pequeña cantidad de dependencias entre las actividades de una app. De hecho, estas suelen iniciar actividades que pertenecen a otras apps. Por ejemplo, una app de navegador podría iniciar la acción de "Compartir actividad" de una app de redes sociales.

> ✋ **Importante**: Si quieres usar actividades en tu app, debes registrar información sobre estas en el **manifiesto** de la app y administrar los **ciclos de vida** de las actividades de manera apropiada.

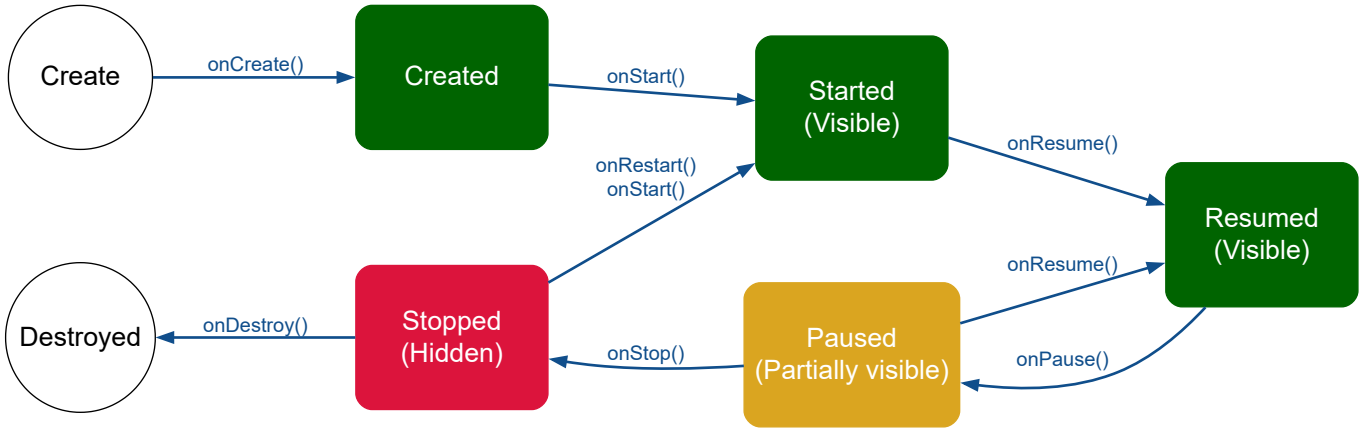
## Ciclo de vida de una Activity

- **Documentación oficial**
- Recursos Adicionales:
  - **Inglés**: **Udacity**

Como hemos comentado en el punto anterior. La activity es uno de los componentes esenciales de una aplicación Android, además de ser el componente que tiene asociada la interfaz de usuario. Una misma aplicación puede tener una o varias actividades y Android permite controlar por completo el **ciclo de vida** de cada una de estas.

Los estados por los cuales puede pasar una Activity son los siguientes: **Creación, Ejecución, Reanudación, Pausa, Parada y Destrucción**.

A la relación entre ellos se le llama **Ciclo de vida de una Actividad**.




El ciclo de vida de una Activity nos describe los estados y las transiciones entre estados de una determinada Activity, que como se ve en la imagen son:

- **Creación**: una actividad se ha creado cuando su estructura se encuentra en memoria, pero esta no es visible aun. Cuando el usuario presiona sobre el icono de la aplicación en su dispositivo, el método `onCreate()` es ejecutado inmediatamente para cargar el layout de la actividad principal en memoria.
- **Ejecución-Reanudación**: después de haber sido cargada la actividad se ejecutan en secuencia el método `onStart() y onResume()`. Aunque `onStart()` hace visible la actividad, es `onResume()` quien le transfiere el foco para que interactúe con el usuario.
- **Pausa**: una actividad está en pausa cuando se encuentra en la pantalla parcialmente visible.

  > **Ejemplo**: Cuando se abren diálogos que toman el foco superponiéndose a la actividad. El método llamado para la transición hacia la pausa es `onPause()` .

- **Detención**: Una actividad está detenida cuando no es visible en la pantalla, pero aún se encuentra en memoria y en cualquier momento puede ser reanudada. Cuando una aplicación es enviada a segundo plano se ejecuta el método `onStop()`. Al reanudar la actividad, se pasa por el método `onRestart()` hasta llegar a el estado de ejecución y luego al de reanudación.
- **Destrucción**: cuando la actividad ya no existe en memoria se encuentra en estado de destrucción. Antes de pasar a destruir la aplicación se ejecuta el método `onDestroy()`. Es común que la mayoría de actividades no implementen este método, a menos que deban destruir procesos como servicios en segundo plano.

> 📌 **Nota**: Aunque hay muchos más conceptos básicos que debemos conocer sobre las actividades y la comunicación entre ellas. Por ahora, con lo que hemos visto es suficiente para empezar a crear nuestras primeras aplicaciones.

# El contexto en Android

El **contexto** de una aplicación es una **interfaz entre la aplicación y el sistema operativo**, la cual describe la información que representa tu aplicación dentro del ámbito del sistema operativo.

También permite acceder a los **recursos** de la aplicación y coordinar el funcionamiento de los bloques de la aplicación.

El contexto representa toda la meta-información sobre las relaciones que tiene la aplicación con otras aplicaciones o el sistema y podemos implementarlo a través de la **Context**.

En Android nos encontramos con los siguientes tipos de contextos:

- **Aplicación**: Este contexto engloba a todos los demás, y cubre todo el ciclo de vida de la aplicación desde que la arrancamos hasta que muere. Por lo que cada aplicación tiene *'un único contexto de aplicación'*. El `Context` de la aplicación vive hasta que se termina por completo la aplicación (hasta que muere, no hasta que se pausa)

  > ✋ **Importante**: Se puede acceder desde una `Activity` o un `Service` con `application` o desde cualquiera que herede de `Context` con `applicationContext` .

- **Activity o Service**: Como hemos dicho, un `Context` vive tanto como el elemento al que pertenece, por lo que depende del ciclo de vida. Así, un Context de una `Activity` vivirá el mismo tiempo que viva la activity y siempre será un tiempo menor que el de la Aplicación.
- Recursos Adicionales:
  - **Inglés**: **Philipp Lackner**

# La clase Application

- **Documentación oficial**

La clase `Application` es la clase base para las aplicaciones de Android y es el punto de partida para iniciar las actividades de la aplicación. Cuando se inicia una aplicación, el sistema crea una única instancia de `Application` para la misma, y la asigna a un proceso de ejecución de la aplicación. La clase `Application` permanece en el proceso de ejecución de la aplicación **durante toda la duración de la vida de la aplicación**.

Esta clase `Application` es una subclase de `Context` y por tanto tiene acceso a todos los métodos y propiedades de esta clase. Además, es un **objeto singleton** que se crea cuando se inicia la aplicación y se destruye cuando se cierra la aplicación.

> 📌 **Nota**: Aunque **no es obligatorio**, es una buena práctica crear una subclase de `Application` para mantener el estado global de la aplicación. En el siguiente apartado veremos como crearla.

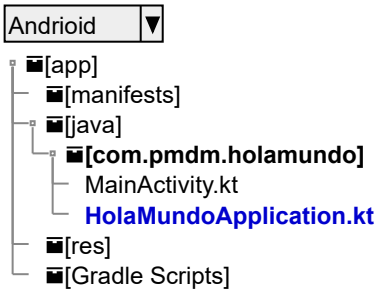
## Crear una subclase de Application

Por ejemplo, si quisiéramos añadir a nuestro '*Hola Mundo*' nuestra propia clase `Application`. Lo primero que tenemos que hacer es crear una clase que herede de `Application`. Por ejemplo, `MiHolaMundo`. Para ello, seguiremos estos **dos** pasos:

1. Creamos una clase `HolaMundoApplication` que herede de `Application` en el paquete principal `com.pmdm.holamundo`.

```
package com.pmdm.holamundo
import android.app.Application

// No tiene porque tener cuerpo si no tiene nada que inicializar
class HolaMundoApplication : Application()
```



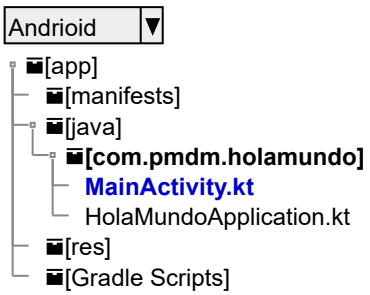

2. Modificamos el archivo `AndroidManifest.xml` para que la aplicación use nuestra clase `MiHolaMundo` en lugar de la clase `Application` por defecto.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    <application
        android:name="com.pmdm.holamundo.HolaMundoApplication"
        ...>
    </application>
</manifest>
```

# La clase Activity

- **Documentación oficial**

Anteriormente ya hemos explicado el concepto de Activity y su ciclo de vida. Ahora vamos a ver sobre la clase que la representa y la actividad principal que se crea en la plantilla de aplicación de compose.




En el programa por defecto tendremos ya definida una clase `MainActivity` que hereda de `ComponentActivity`. Esta clase es una subclase de `Activity` que nos proporciona la librería de Jetpack Compose y nos permitirá crear la interfaz de usuario de nuestra aplicación usando Compose en lugar de XML tradicional.

El código de la clase `MainActivity` es el siguiente:

```
class MainActivity : ComponentActivity() {
    // Método que se ejecuta cuando se crea la actividad.
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // Código donde se establece la UI de la Activity
        // y que comentaremos más adelante
        setContent {
            HolaMundoTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Greeting("Hola Mundo")
                }
            }
        }
    }
}
```

y básicamente no hace nada más que establecer la interfaz de usuario de la aplicación. Aunque como hemos comentado, desde ella podremos acceder a la aplicación y a sus recursos....

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

    val aplicacion = this.application as HolaMundoApplication
}
```

# Android Manifest

- **Documentación oficial**

El manifiesto de Android es un archivo XML que **contiene información esencial sobre la aplicación para el sistema operativo Android**. Esta información sirve para que el sistema operativo pueda identificar la aplicación y sus **componentes**, así como para declarar los **permisos** que necesita la aplicación para acceder a partes del sistema operativo o a otros **servicios**.


Si examinamos el contenido del archivo `AndroidManifest.xml` de nuestra plantilla de aplicación de compose, podemos ver que tiene la siguiente estructura...

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

    <application
        android:name="com.pmdm.holamundo.HolaMundoApplication"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/HolaMundo"
        android:usesCleartextTraffic="true"
        tools:targetApi="31">
        <activity
            android:name="com.pmdm.holamundo.MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/HolaMundo">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Vamos a ver los elementos más importantes de este archivo. El cual iremos modificando a lo largo del curso, añadiendo permisos, servicios, etc.

- **<manifest>** contiene como atributos los dos espacios de nombres con la definición del esquema XML. Su inclusión permitirá al editor de XML de Android Studio validar el contenido del archivo y ofrecernos ayuda mientras lo rellenamos.

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

- **<application>** es el elemento raíz del manifiesto. Contiene información sobre la aplicación y cada uno de sus componentes. En este caso, solo tenemos un componente, la actividad principal.

```
<application
    android:name="com.pmdm.holamundo.HolaMundoApplication"
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/HolaMundo"
    android:usesCleartextTraffic="true"
    tools:targetApi="31">
    ...
</application>
```

  - `android:name` es el nombre de la clase que implementa la clase `Application`. Es **opcional y por defecto no se define**. En el apartado anterior le asignamos el valor `com.pmdm.holamundo.HolaMundoApplication`.
  - `android:allowBackup` determina si se permite a la aplicación participar en la infraestructura de copia de seguridad y restablecimiento.
  - `android:dataExtractionRules` las aplicaciones pueden establecer este atributo en un recurso XML en el que se especifican las reglas que determinan qué archivos y directorios puedes copiar del dispositivo como parte de las operaciones de copia de seguridad o transferencia. Por defecto, el valor está definido en el recurso `@xml/data_extraction_rules`.
  - `android:fullBackupContent` es un archivo XML que contiene reglas de exclusión de datos para la copia de seguridad de la aplicación. Por defecto, el valor está definido en el recurso `@xml/backup_rules` y puedes ver su sintaxis **aquí**.

- `android:icon` es el icono de la aplicación. Por defecto, el valor es el recurso `@mipmap/ic_launcher`.
- `android:label` es el nombre de la aplicación. Por defecto, el valor está en el recurso `@string/app_name`.
- `android:roundIcon` es el icono redondo de la aplicación. Por defecto, el valor está en el recurso `@mipmap/ic_launcher_round`.
- `android:supportsRtl` indica si la aplicación soporta la escritura de derecha a izquierda. Por defecto, el valor es `true`.
- `android:theme` es el tema de la aplicación. Por defecto, el valor está en el recurso `@style/HolaMundo`.
- `android:usesCleartextTraffic` indica si la aplicación usa tráfico **HTTP de texto simple sin formato** implica que cualquier persona que supervise el tráfico de red pueda ver y manipular los datos que se transmiten. Esta es una vulnerabilidad si los datos transmitidos incluyen información sensible, como contraseñas, números de tarjetas de crédito o cualquier otro tipo de información personal. Por razones obvias, el valor por defecto `false` pero **nosotros lo hemos cambiado a `true` para poder hacer peticiones a una API de prueba**.

- **<activity>** es el **componente** que representa una pantalla con una **interfaz de usuario**. En este caso, solo tenemos una actividad, la principal.

```
<application ...>
    <activity
    android:name="com.pmdm.holamundo.MainActivity"
    android:exported="true"
    android:label="@string/app_name"
    android:theme="@style/HolaMundo">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    </activity>
</application>
```

  - `android:name` es el nombre de la clase que implementa la actividad. En nuestro ejemplo, el valor es `com.pmdm.holamundo.MainActivity`.
  - `android:exported` indica si la actividad puede ser iniciada por componentes de otras aplicaciones. Esto es, si es accesible desde fuera de la aplicación.
  - `android:label` la etiqueta que se muestra en la pantalla cuando la actividad se representa al usuario. Por lo general, se muestra junto al ícono de la actividad. Si no se establece este atributo, en su lugar se usa la etiqueta configurada para toda la aplicación (consulta el

atributo label del elemento `<application>`). Por defecto, el valor está en el recurso `@string/app_name`.

- `android:theme` Es la referencia a un recurso de estilo que define un tema general para toda la actividad. Establece automáticamente el contexto de la actividad para el uso de este theme y también puede generar animaciones de "inicio" antes del lanzamiento de la actividad (para una mejor adecuación a la apariencia real de la actividad). Por defecto, el valor está en el recurso `@style/HolaMundo`.
- **<intent-filter>** en Android no existe un único punto de entrada para nuestra aplicación. Podemos iniciarla a través de múltiples activities o services que pueden ser iniciados a partir de intents específicos que puede enviar el sistema u otra aplicación. Para decir a Android ante qué intent debe reaccionar nuestra aplicación y cómo, existe esta etiqueta.
  - `<action>` es un elemento que contiene un nombre de acción que describe la capacidad general que la actividad puede realizar. Con `android.intent.action.MAIN` es una acción que indica que la actividad puede ser el punto de entrada principal de la aplicación (es decir, la actividad que se inicia cuando el usuario selecciona el ícono de la aplicación). Sólo puede haber **una activity que reaccione** a este intent.
  - `<category>` es un elemento que contiene un nombre de categoría que describe la capacidad general que la actividad puede realizar. Con `android.intent.category.LAUNCHER` le decimos a Android que queremos que esta activity sea añadida al lanzador de la aplicación. Pueden haber varias activities que reaccionen a este intent.

> 📌 **Nota**: Aunque hay muchos más conceptos básicos que debemos conocer sobre el manifiesto de la aplicación. Por ahora, con lo que hemos visto es suficiente para empezar a crear nuestras primeras aplicaciones. Más adelante iremos viendo más conceptos. Por ejemplo, como añadir permisos, etc.