Aspectos fundamentales de la aplicación

Para escribir aplicaciones de Android, es posible usar los lenguajes Kotlin, Java y C++.
Las herramientas de Android SDK compilan tu código, junto con los archivos de recursos y datos, en un APK:

- un *paquete de Android*, que es un archivo de almacenamiento con el sufijo **.apk**.
- Un archivo APK incluye todos los contenidos de una aplicación de Android y es el archivo que usan los dispositivos con tecnología Android para instalar la aplicación.

Cada aplicación de Android reside en su propia zona de pruebas de seguridad y está protegida por las siguientes características de seguridad de Android:

- El sistema operativo Android es un sistema Linux multiusuario en el que cada aplicación es un usuario diferente.
- De forma predeterminada, el sistema le asigna a cada aplicación un ID de usuario de Linux único (solo el sistema utiliza el ID y la aplicación lo desconoce).
    - El sistema establece permisos para todos los archivos en una aplicación de modo que solo el ID de usuario asignado a esa aplicación pueda acceder a ellos.
- Cada proceso tiene su propia máquina virtual (VM), por lo que el código de una aplicación se ejecuta de forma independiente de otras aplicaciones.
- De forma predeterminada, cada aplicación ejecuta su propio proceso de Linux.
    - El sistema Android inicia el proceso cuando se requiere la ejecución de alguno de los componentes de la aplicación y, luego, lo cierra cuando el proceso ya no es necesario o cuando el sistema debe recuperar memoria para otras aplicaciones.

De esta manera, el sistema Android implementa el *principio de mínimo privilegio*.

- ➔ Es decir, de forma predeterminada, cada aplicación tiene acceso solo a los componentes que necesita para llevar a cabo su trabajo y nada más.
- ➔ Esto crea un entorno muy seguro, en el que una aplicación no puede acceder a partes del sistema para las que no tiene permiso.
- ➔ Sin embargo, hay maneras en las que una aplicación puede compartir datos con otras aplicaciones y en las que una aplicación puede acceder a servicios del sistema:
- ➔ Es posible coordinar que dos aplicaciones compartan el mismo ID de usuario de Linux para que puedan acceder a los archivos de la otra.
- ➔ Para conservar recursos del sistema, las aplicaciones con el mismo ID de usuario también pueden coordinar la ejecución en el mismo proceso de Linux y compartir la misma VM. Las aplicaciones también deben estar firmadas con el mismo certificado.

- Una aplicación puede solicitar permiso para acceder a datos del dispositivo, como los contactos de un usuario, los mensajes de texto, el dispositivo de almacenamiento (tarjeta SD), la cámara y la conexión Bluetooth. El usuario debe conceder de manera explícita estos permisos.

Componentes de la aplicación

Los componentes de la aplicación son bloques de creación esenciales de una aplicación para Android.

1) Cada componente es un punto de entrada por el que el sistema o un usuario ingresan a tu aplicación.
2) Algunos componentes dependen de otros.

3) Las aplicaciones tienen cuatro tipos de componentes diferentes:
    - Actividades
    - Servicios
    - Receptores de emisiones
    - Proveedores de contenido

Cada tipo tiene un fin específico y un ciclo de vida característico que define cómo se crea y se destruye el componente.

Actividades

Una *actividad* es el punto de entrada de interacción con el usuario.
Representa una pantalla individual con una interfaz de usuario.
Por ejemplo, una aplicación de correo electrónico tiene:

1. una actividad que muestra una lista de los correos electrónicos nuevos
2. otra actividad para redactar el correo electrónico
3. y otra actividad para leerlo.

Si bien las actividades funcionan juntas para ofrecer una experiencia de usuario uniforme en la aplicación de correo electrónico, cada una es independiente de las demás.
De esta manera, una aplicación diferente puede iniciar cualquiera de estas actividades (si la aplicación de correo electrónico lo permite).
Por ejemplo,

> para que el usuario comparta una imagen, una aplicación de cámara puede iniciar la actividad correspondiente en la aplicación de correo electrónico que redacta el nuevo mensaje.

Una actividad posibilita las siguientes interacciones clave entre el sistema y la aplicación:

- Realizar un seguimiento de lo que realmente le interesa al usuario (lo que está en pantalla) para garantizar que el sistema siga ejecutando el proceso que aloja la actividad.
- Saber que los procesos usados con anterioridad contienen elementos a los que el usuario puede regresar (actividades detenidas) y, en consecuencia, priorizar más esos procesos que otros.
- Ayudar a la aplicación a controlar la finalización de su proceso para que el usuario pueda regresar a las actividades con el estado anterior restaurado.
- Permitir que las aplicaciones implementen flujos de usuarios entre sí y que el sistema los coordine (el ejemplo más común es compartir).

Las actividades se implementan como subclases de la clase Activity.

Servicios

Un *servicio* es un punto de entrada general que permite mantener la ejecución de una aplicación en segundo plano por diversos motivos.

- Es un componente que se ejecuta en segundo plano para realizar operaciones de ejecución prolongada o para realizar tareas de procesos remotos.
- Un servicio no proporciona una interfaz de usuario.
- Por ejemplo, un servicio podría:
    - reproducir música en segundo plano mientras el usuario se encuentra en otra aplicación
    - capturar datos en la red sin bloquear la interacción del usuario con una actividad.

Otro componente, como una actividad, puede iniciar el servicio y permitir que se ejecute o enlazarse a él para interactuar.

De hecho, existen dos semánticas muy diferentes que los servicios usan para indicarle al sistema cómo administrar una aplicación:

1) Los servicios iniciados le indican al sistema que los siga ejecutando hasta que finalicen su trabajo.
    - Dos ejemplos serían:
        - la sincronización de datos en segundo plano o la reproducción de música después de que el usuario sale de la aplicación.

La sincronización de datos en segundo plano o la reproducción de música también son dos tipos de servicios iniciados muy diferentes que modifican la manera en que el sistema los administra:

- La reproducción de música es algo de lo que el usuario es consciente, por lo que la aplicación se lo comunica al sistema indicándole que quiere ejecutarse en segundo plano y le envía una notificación al respecto al usuario. En este caso, el sistema sabe que debe hacer todo lo posible para mantener el proceso de ese servicio en funcionamiento porque el usuario no estará satisfecho si se interrumpe.
- Un servicio habitual en segundo plano no es algo de lo que el usuario sea consciente, por lo que el sistema tiene más libertad para administrarlo. Puede permitir que se interrumpa (y reiniciarlo posteriormente) si necesita memoria RAM en procesos que son más urgentes para el usuario.

Servicios enlazados

Los servicios enlazados se ejecutan porque otra aplicación (o el sistema) indicó que quiere usarlos. Básicamente, lo que sucede es que un servicio le brinda una API a otro proceso.

- Por lo tanto, el sistema sabe que hay una dependencia entre estos procesos.
- En consecuencia, si el proceso A está enlazado a un servicio en el proceso B, sabe que debe mantener funcionando el proceso B (y el servicio correspondiente) para el proceso A.
- Además, si el proceso A es de interés para el usuario, también sabe que debe tratar el proceso B teniendo esto en cuenta.

Gracias a su flexibilidad (o a pesar de ella), los servicios se han convertido en un componente muy útil para todos los tipos de conceptos generales del sistema.

- Los fondos de pantalla animados, los receptores de notificaciones, los protectores de pantalla, los métodos de entrada, los servicios de accesibilidad y muchas otras funciones básicas del sistema se compilan como servicios que las aplicaciones implementan y que el sistema vincula cuando deben ejecutarse.

Un servicio se implementa como una subclase de Service.

Receptores de emisiones

Un receptor de emisión es un componente que posibilita que el sistema entregue eventos a la aplicación fuera de un flujo de usuarios habitual, lo que permite que la aplicación responda a los anuncios de emisión de todo el sistema.
Dado que los receptores de emisión son otro punto bien definido de entrada a la aplicación, el sistema puede entregar emisiones incluso a las aplicaciones que no estén en ejecución.
Por ejemplo:

- una aplicación puede programar una alarma para publicar una notificación sobre un evento futuro destinada al usuario.
- Al entregar dicha alarma al receptor de emisión de la aplicación, no hace falta que dicha aplicación siga ejecutándose hasta que se active la alarma.

Muchas emisiones provienen del sistema (por ejemplo, las que anuncian que se apagó la pantalla, que el nivel de carga de la batería es bajo o que se capturó una imagen).
Las aplicaciones también pueden iniciar emisiones, por ejemplo, para avisarles a las demás aplicaciones que se descargaron datos al dispositivo y que están disponibles para el uso.

Si bien los receptores de emisión no exhiben una interfaz de usuario, pueden crear una notificación de la barra de estado para alertar al usuario cuando se produzca un evento de emisión.

En general, un receptor de emisión es simplemente una puerta de enlace a otros componentes y está destinado a realizar una cantidad mínima de trabajo.
Por ejemplo:

- podría programar un servicio **JobService** para que realice algunas tareas en función del evento con **JobScheduler**.

Un receptor de emisión se implementa como una subclase de BroadcastReceiver y cada receptor de emisión se entrega como un objeto Intent.

Proveedores de contenido

Un proveedor de contenido administra un conjunto compartido de datos de la aplicación que puedes almacenar en el sistema de archivos, en una base de datos SQLite, en la Web o en cualquier otra ubicación de almacenamiento persistente a la que tenga acceso tu aplicación.
A través del proveedor de contenido, otras aplicaciones pueden consultar o modificar los datos si el proveedor de contenido lo permite.
Por ejemplo:

- el sistema Android proporciona un proveedor de contenido que administra la información de contacto del usuario.
- De esta manera, cualquier aplicación con los permisos correspondientes puede consultar el proveedor de contenido (como **ContactsContract.Data**) para la lectura y la escritura de información sobre una persona específica.

En ocasiones, se interpreta que el proveedor de contenido es una abstracción en una base de datos, porque tiene un gran volumen de API y compatibilidad incorporado. Sin embargo, su finalidad principal es diferente desde una perspectiva de diseño del sistema.
Para el sistema:

- un proveedor de contenido es un punto de entrada a una aplicación para publicar elementos de datos con nombre y se identifica mediante un esquema de URI.
- Así, una aplicación puede decidir cómo quiere asignar los datos que contiene a un espacio de nombres de URI y entregar esos URI a otras entidades que, a su vez, pueden usarlos para acceder a los datos.
- Gracias a esto, el sistema puede realizar algunas tareas específicas cuando administra una aplicación:

1. Asignar un URI no exige que la aplicación permanezca ejecutándose, por lo que los URI pueden persistir incluso después de que se cierran las aplicaciones a las que pertenecen.
   El sistema solo necesita asegurarse de que la aplicación de un URI siga ejecutándose si debe recuperar los datos de la aplicación desde el URI en cuestión.
2. Estos URI también ofrecen un modelo de seguridad importante y detallado.
   Por ejemplo:

- una aplicación puede colocar el URI de una imagen que tiene en el portapapeles, pero bloquear al proveedor de contenido para que otras aplicaciones no puedan acceder a él libremente.
  Si otra aplicación intenta acceder a ese URI en el portapapeles, el sistema puede concederle acceso usando un permiso de URI temporal para que solo pueda acceder a los datos mediante ese URI, pero nada más.

Los proveedores de contenido también son útiles para leer y escribir datos privados de tu aplicación y que no se comparten.

Un proveedor de contenido se implementa como una subclase de ContentProvider y debe implementar un conjunto estándar de API que permitan a otras aplicaciones realizar transacciones.

Consideraciones:
Un aspecto exclusivo del diseño del sistema Android es que cualquier aplicación puede iniciar un componente de otra aplicación.
Por ejemplo:

- si quieres que el usuario tome una foto con la cámara del dispositivo, probablemente haya otra aplicación para eso y tu aplicación pueda usarla, en lugar de desarrollar una actividad para tomar una foto.
- No hace falta que incorpores ni vincules la aplicación de la cámara con el código.
- En cambio, basta con que inicies la actividad en la aplicación de la cámara que captura una foto.
- Cuando termines, la foto aparecerá en tu aplicación para que puedas usarla.
- Al usuario le parecerá que la cámara en realidad forma parte de tu aplicación.

Cuando el sistema inicia un componente:

1. inicia el proceso para esa aplicación (si es que no se está ejecutando)
2. crea una instancia de las clases necesarias para el componente.

Por ejemplo:

- si tu aplicación inicia la actividad en la aplicación de cámara que toma una foto, esa actividad se ejecuta en el proceso que pertenece a la aplicación de cámara, no en el proceso de tu aplicación.
- Por lo tanto, a diferencia de lo que sucede con las aplicaciones en la mayoría de los demás sistemas, las aplicaciones de Android no tienen un solo punto de entrada (no existe la función **main()**).

Como el sistema ejecuta cada aplicación en un proceso independiente con permisos de archivo que limitan el acceso a otras aplicaciones, tu aplicación no puede activar directamente un componente de otra.

- Sin embargo, sí puede hacerlo el sistema Android.
- Para activar un componente en otra aplicación, envía un mensaje al sistema que especifique tu intención de iniciar un componente específico.
- Luego, el sistema activa ese componente por ti.

Activación de componentes

De los cuatro tipos de componentes, tres (actividades, servicios y receptores de emisión) se activan mediante un mensaje asíncrono denominado intent.

Las intents vinculan componentes entre sí durante el tiempo de ejecución.

- Imagina que son los mensajeros que solicitan acciones de otros componentes, ya sea que estos pertenezcan a tu aplicación o a alguna otra.

Una intent se crea con un objeto **Intent**, que define:

- un mensaje para activar un componente específico (intent explícita)
- o un tipo específico de componente (intent implícita).

Para actividades y servicios, una intent define:

- la acción que se realizará (por ejemplo, ver o enviar algo) y puede especificar el URI de los datos en los que debe actuar, entre otras cosas que el componente que se está iniciando podría necesitar saber.
- Por ejemplo:
  - una intent podría transmitir una solicitud para que una actividad muestre una imagen o abra una página web.
  - en algunos casos, puedes iniciar una actividad para recibir un resultado. En ese caso, dicha actividad también devuelve el resultado en una **Intent**.
  - Por ejemplo:
    - puedes emitir una intent para permitir que el usuario elija un contacto personal y te lo devuelva. Esa intent devuelta incluye un URI dirigido al contacto elegido.

En el caso de los receptores de emisión, la intent tan solo define el anuncio que se emite.
Por ejemplo:

- una emisión para indicar que el nivel de batería del dispositivo es bajo incluye solo una cadena de acción conocida que indica batería baja.

A diferencia de las actividades, los servicios y los receptores de emisión, los proveedores de contenido no se activan con intents.

- En cambio, se activan mediante solicitudes de un **ContentResolver**.
- El solucionador de contenido aborda todas las transacciones directas con el proveedor de contenido, de modo que el componente que realiza las transacciones con el proveedor no deba hacerlo y, en su lugar, llame a los métodos del objeto **ContentResolver**.
- Esto deja una capa de abstracción entre el proveedor de contenido y el componente que solicita información (por motivos de seguridad).

Existen métodos independientes para activar cada tipo de componente:

1. Puedes iniciar una actividad o asignarle una tarea nueva al pasar un **Intent** a **startActivity()** o **startActivityForResult()** (cuando quieres que la actividad devuelva un resultado).
2. Con Android 5.0 (nivel de API 21) y las versiones posteriores, puedes usar la clase **JobScheduler** para programar acciones. En versiones anteriores de Android, puedes iniciar un servicio (o darle instrucciones nuevas a un servicio en curso) al pasar un **Intent** a **startService()**. Puedes establecer un enlace con el servicio al pasar un **Intent** a **bindService()**.
3. Puedes iniciar una emisión al pasar un **Intent** a métodos como **sendBroadcast()**, **sendOrderedBroadcast()** o **sendStickyBroadcast()**.
4. Puedes realizar una consulta a un proveedor de contenido si llamas a **query()** en un **ContentResolver**.

El archivo de manifiesto

Para que el sistema Android pueda iniciar un componente de la aplicación, debe reconocer la existencia de ese componente leyendo el archivo de manifiesto de la aplicación (**AndroidManifest.xml**).
Tu aplicación debe declarar todos sus componentes en este archivo, que debe encontrarse en la raíz del directorio de proyectos de la aplicación.
El manifiesto puede hacer ciertas cosas además de declarar los componentes de la aplicación, por ejemplo:

- Identificar los permisos de usuario que requiere la aplicación, como acceso a Internet o acceso de lectura para los contactos del usuario.
- Declarar el nivel de API mínimo que requiere la aplicación en función de las API que usa.
- Declarar características de hardware y software que la aplicación usa o exige, como una cámara, servicios de Bluetooth o una pantalla multitáctil.
- Declarar bibliotecas de la API a las que la aplicación necesita estar vinculada (además de las API del marco de trabajo de Android), como la biblioteca de Google Maps.

Declaración de componentes

La tarea principal del manifiesto es informarle al sistema acerca de los componentes de la aplicación.
Por ejemplo, un archivo de manifiesto puede declarar una actividad de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

a) En el elemento **<application>**, el atributo **android:icon** señala los recursos de un ícono que identifica la aplicación.
b) En el elemento **<activity>**, el atributo **android:name** especifica el nombre de clase plenamente calificado de la subclase **Activity**, y el atributo **android:label** especifica una cadena para usar como etiqueta de la actividad visible para el usuario.
c) Debes declarar todos los componentes de la aplicación mediante estos elementos:

- Elementos **<activity>** para las actividades.
- Elementos**<service>** para los servicios.
- Elementos **<receiver>** para los receptores de emisión.
- Elementos **<provider>** para los proveedores de contenido.

Consideraciones:
Si incluyes actividades, servicios y proveedores de contenido en tu archivo de origen, pero no los declaras en el manifiesto, no estarán visibles para el sistema y, por consiguiente, no se podrán ejecutar.
No obstante, los receptores de emisión pueden declararse en el manifiesto o crearse dinámicamente en forma de código como objetos **BroadcastReceiver** y registrarse en el sistema llamando a **registerReceiver()**.

Declaración de capacidades de los componentes

Como se comento antes, puedes usar un **Intent** para iniciar actividades, servicios y receptores de emisión.

- Puedes usar una **Intent** denominando explícitamente el componente objetivo (usando el nombre de clase del componente) en la intent.
- También puedes usar una intent implícita, que describe el tipo de acción que se realizará y, opcionalmente, los datos sobre los que se realizará la acción.
  - La intent implícita permite que el sistema busque un componente en el dispositivo que pueda realizar la acción e iniciarla.
  - Si hay varios componentes que pueden realizar la acción que describe la intent, el usuario selecciona la que quiere usar.

Advertencia: Si utilizas una intent para iniciar un **Service**, asegúrate de usar una intent explícita que garantice la seguridad de tu aplicación. El uso de una intent implícita para iniciar un servicio es un riesgo de seguridad porque no es posible estar seguro de qué servicio responderá a la intent, y el usuario no puede ver qué servicio se inicia. A partir de Android 5.0 (nivel de API 21), el sistema arroja una excepción si llamas a **bindService()** con una intent implícita. No declares filtros de intents de los servicios.

- El sistema identifica los componentes que pueden responder a una intent comparando la intent recibida con los filtros de intents que se proporcionan en el archivo de manifiesto de otras aplicaciones del dispositivo.
- Cuando declaras una actividad en el manifiesto de tu aplicación, tienes la posibilidad de incluir filtros de intents que declaran las capacidades de la actividad de modo que pueda responder a intents desde otras aplicaciones.
- Puedes declarar un filtro de intents de tu componente agregando un elemento **<intent-filter>** como campo secundario del elemento de declaración del componente.

Por ejemplo, si compilaste una aplicación de correo electrónico con una actividad para redactar un nuevo mensaje de correo electrónico, puedes declarar un filtro de intents para responder a las intents "enviar" (a fin de enviar un nuevo correo electrónico), tal como se muestra en el siguiente ejemplo:

```
<manifest ... >
    ...
    <application ... >
        <activity android:name="com.example.project.ComposeEmailActivity">
            <intent-filter>
                <action android:name="android.intent.action.SEND" />
                <data android:type="*/*" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- Luego, si otra aplicación crea una intent con la acción **ACTION_SEND** y la pasa a **startActivity()**, el sistema puede iniciar tu actividad para que el usuario pueda redactar y enviar un correo electrónico.

Declaración de requisitos de la aplicación

Existen muchos dispositivos con tecnología Android, pero no todos ofrecen las mismas funciones y capacidades.
A fin de evitar que se instale tu aplicación en dispositivos que no tienen las funciones que tu aplicación necesita, es importante que definas claramente un perfil para los tipos de dispositivos que la aplicación admite; para ello, declara los requisitos de dispositivos y software en tu archivo de manifiesto.
La mayoría de esas declaraciones son solo informativas y el sistema no las lee, pero servicios externos como Google Play sí lo hacen para ofrecerles a los usuarios opciones de filtrado cuando buscan aplicaciones desde sus dispositivos.
Por ejemplo:

- si tu aplicación requiere una cámara y usa API introducidas en Android 2.1 (nivel de API 7), debes declarar esto como requisito en tu archivo de manifiesto, tal como se muestra en el siguiente ejemplo:

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera.any"
                  android:required="true" />
    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />
    ...
</manifest>
```

- Con las declaraciones que figuran en el ejemplo, los dispositivos que no tengan cámara y tengan una versión de Android anterior a 2.1 no podrán instalar tu aplicación desde Google Play.
- No obstante, puedes declarar que tu aplicación usa la cámara, pero no la exige. En ese caso, tu aplicación debe fijar el atributo **required** en **false** y comprobar durante el tiempo de ejecución si el dispositivo tiene cámara e inhabilitar las funciones de cámara según corresponda.

Recursos de la aplicación

En Android, una aplicación está compuesta por más que simplemente código; requiere recursos independientes del código fuente, como imágenes, archivos de audio y otros elementos relacionados con la presentación de la aplicación.
Por ejemplo:

- Puedes definir animaciones, menús, estilos, colores y el diseño de las interfaces de usuario de la actividad con archivos XML.
    - Los recursos de la aplicación facilitan la actualización de numerosas características de la aplicación sin cambiar el código.
    - Además, proporcionar conjuntos de recursos alternativos te permite optimizar tu aplicación para una variedad de configuraciones de dispositivos, como diferentes idiomas y tamaños de pantalla.
- Por cada recurso que incluyes en tu proyecto de Android, las herramientas de compilación del SDK definen un ID con número entero único que puedes usar para hacer referencia al recurso desde el código de tu aplicación o desde otros recursos definidos en XML.
  Por ejemplo:
    - si tu aplicación contiene un archivo de imagen denominado **logo.png** (guardado en el directorio **res/drawable/**), las herramientas del SDK generan un ID de recurso llamado

R.drawable.logo. Este ID está asignado a un número entero específico de la aplicación que puedes usar para hacer referencia a la imagen e insertarla en tu interfaz de usuario.

Uno de los aspectos más importantes de proporcionar recursos independientes de tu código fuente es la capacidad de ofrecer recursos alternativos para diferentes configuraciones de dispositivos.
Por ejemplo:

- al definir cadenas de IU en XML, puedes traducir las cadenas a otros idiomas y guardarlas en archivos independientes.
- Luego, el sistema Android aplica las cadenas de idioma correspondientes a tu IU en función de un calificador de idioma que anexes al nombre del directorio de recursos (como **res/values-fr/** para los valores de la cadena de francés) y a la configuración de idioma del usuario.

Android admite muchos calificadores diferentes para tus recursos alternativos. El calificador es una cadena corta que incluyes en el nombre de tus directorios de recursos para definir la configuración del dispositivo en la que se deben usar esos recursos.
Por ejemplo:

- debes crear diferentes diseños para tus actividades según la orientación y el tamaño de la pantalla del dispositivo.
    - Cuando la orientación de la pantalla del dispositivo sea vertical (a lo alto), quizá quieras usar un diseño en el que los botones estén alineados de forma vertical
    - En cambio, cuando sea horizontal (a lo ancho), los botones podrían alinearse horizontalmente.
    - Para cambiar el diseño según la orientación, puedes definir dos diseños diferentes y aplicar el calificador adecuado al nombre de directorio de cada diseño.
    - Después, el sistema aplicará automáticamente el diseño correspondiente según la orientación actual del dispositivo.