Programación Multimedia y Dispositivos Móviles

Tema 1. Análisis de tecnologías para aplicaciones en dispositivos móviles

Índice


ANÁLISIS DE TECNOLOGÍAS PARA APLICACIONES EN DISPOSITIVOS MÓVILES
ECOSISTEMAS DE DESARROLLO Y SISTEMAS OPERATIVOS
Android
iOS
Otros
LIMITACIONES
Fragmentación de dispositivos
Hardware heterogéneo
Software (diferentes versiones del sistema operativo [SO])
HERRAMIENTAS DE DESARROLLO
Aplicaciones nativas
Aplicaciones híbridas
Android.
iOS.
Xamarin.
Ionic.
PhoneGap.
React Native.

1.1. Introducción y objetivos

En el transcurso de este tema se realizará una **introducción** al mundo del **desarrollo** de **aplicaciones móviles,** donde se analizarán sus **características** y **componentes principales.** En particular, nos centraremos en el **ecosistema Android;** abordaremos tanto los entornos de desarrollo como los emuladores, además del desarrollo y ciclo de vida de las aplicaciones Android.

Durante el desarrollo de este tema se deben conseguir los siguientes objetivos:

- Comprender los conceptos básicos relacionados con las aplicaciones en dispositivos móviles.
- Aprender las limitaciones de estas aplicaciones.
- Conocer las posibilidades tecnológicas que tenemos a nuestra disposición.
- Conocer los diferentes entornos de desarrollo integrados (IDE) para la creación de aplicaciones en dispositivos móviles.
- Conocer los emuladores, que nos van a servir para verificar nuestra aplicación.
- Aprender a instalar a la par que configurar Android Studio como entorno de desarrollo para aplicaciones Android.

1.2. Aplicaciones en dispositivos móviles: conceptos, limitaciones y tecnologías disponibles

Actualmente, hay a nuestra disposición **multitud** de **dispositivos móviles,** desde teléfonos hasta tabletas, pasando por reproductores multimedia, etc. Estas sus **características comunes:**

- Tamaño pequeño. Son aparatos que se pueden transportar fácilmente y utilizar tanto con una mano como con las dos.
- Movilidad. Pueden usarse sin necesidad de estar conectados físicamente a una fuente de energía. También poseen movilidad en sus datos y la capacidad de modificarse mientras se encuentran en distintos lugares.
- Conexión. Tienen la posibilidad de conectarse a Internet o a distintas redes de manera continua o intermitente, así como entre sí y a otros dispositivos auxiliares con distintos protocolos (p. ej., Bluetooth).
- Capacidades. Poseen capacidad de procesamiento mediante una unidad central de procesamiento (CPU) multinúcleo y de memoria (de acceso aleatorio [RAM], tarjetas micro-SD, *flash,* etc.) Además, pueden tener cámaras, sistema de posicionamiento global (GPS), acelerómetros, así como otros sensores u otras capacidades.
- Interacción. Cuentan con una alta capacidad de interacción, habitualmente, a través de una pantalla táctil o de algún otro método (p. ej., el teclado).
- Uso. Lo habitual es que sean de uso individual y, en muchos casos, de una única persona.

Limitaciones

La totalidad de estos nuevos dispositivos y tecnologías novedosas también tienen sus **limitaciones,** que vamos a enumerar a continuación:

- Fragmentación de dispositivos. Es uno de los problemas que nos encontramos a la hora de desarrollar una aplicación para un dispositivo móvil. Existen infinidad de tipos, marcas y, dentro de estas, también hay modelos variados. Cada uno de ellos tendrá distintas características; el tamaño y la resolución de la pantalla son las que más quebraderos de cabeza suelen dar.

- *Hardware*. Derivada de esta fragmentación de dispositivos anterior, nos encontramos con las características técnicas, tales como el procesador o las posibilidades de conexión, y, sobre todo, la memoria. Todas estas harán que una aplicación pueda ir muy lenta en el dispositivo o que, directamente, no sea capaz de ejecutarse.

- *Software*. La versión del sistema operativo (SO) que tenga instalado el dispositivo móvil influirá en las posibilidades que podrá tener la aplicación y, en el caso más extremo, incluso en el hecho de que se pueda o no instalar.

Tecnologías disponibles

A lo largo de estos años hemos podido conocer **diferentes SO** para **distintos dispositivos móviles.** Unos han desaparecido prácticamente y otros o bien están a punto de hacerlo, o bien tienen un uso muy residual. Veamos algunos de los más importantes.

Android

Es un SO diseñado por **Google,** basado en el *kernel* (núcleo) de **Linux.** Se implementó principalmente para dispositivos móviles, tabletas, y, actualmente, también para relojes, televisores y automóviles.

Si deseamos crear una aplicación para nuestro teléfono inteligente o Google Play Store, el **kit** de **desarrollo** de ***software*** (SDK) de Android nos ofrece un conjunto de **herramientas.** Cuenta con los siguientes **elementos:**

- Depurador de código.
- Biblioteca.
- Simulador (emulador) de teléfono, basado en QEMU.
- Documentación.
- Muchos ejemplos de código.

La web oficial Android Developer nos ofrece toda la información a fin de poder empezar a desarrollar aplicaciones para dispositivos con este SO; mediante el pago de una única cuota podemos registrarnos y obtener una cuenta de desarrollador de Play Store.

iOS

Este SO cuenta con la **siguientes características:**

- Se trata de un SO móvil desarrollado por Apple, Inc. para sus dispositivos, que incluyen el iPhone, iPad y iPod Touch. Se lanzó por primera vez en 2007, junto con el primer iPhone, y, desde entonces, es uno de los más populares del mundo.
- Permite a quienes lo usan descargar y comprar aplicaciones a través de la App Store, administrada por la empresa. Esta tienda ofrece millones de ellas, desde juegos hasta herramientas de alta productividad.
- Está diseñado para integrarse a la perfección con otros productos de la marca, como Apple Watch, Apple TV y Mac, lo que permite una experiencia de usuario (UX) fluida a la par que sincronizada entre dispositivos.

Apple controla estrictamente el desarrollo de aplicaciones para iOS, utilizando su entorno de desarrollo Xcode y el lenguaje de programación Swift, lo que garantiza un alto nivel de calidad y rendimiento en estas.

La información completa para convertirnos en personas desarrolladoras Apple la podemos encontrar en Apple Developer, donde se puede conseguir el SDK gratis. Con la misión de publicar nuestras aplicaciones en la App Store necesitamos una **cuenta** de **desarrollador;** hay dos opciones, a saber, la **estándar,** por 99 $ (al año), y la de **empresas,** por 299 $.

Otros

Aquí encontramos los siguientes:

- Windows Phone. Este es el SO que Microsoft utiliza en los teléfonos inteligentes y en otros dispositivos móviles. En 2010 la empresa inauguró la nueva plataforma para teléfonos inteligentes, conocida como Windows Phone 7, y la última actualización es el SO Windows 10 Mobile. Aunque actualmente se mantiene en algunos dispositivos y con actualizaciones de seguridad, muchas aplicaciones están desapareciendo de su ecosistema.

- Amazon Fire OS. Se basa en el SO de Android, Amazon es su productor y está diseñado de forma específica tanto para el Fire Phone de Amazon como la gama de tabletas de Kindle Fire. Tiene estas **características:**

  - Se enfoca principalmente en el consumo de contenidos.
  - Viene con una interfaz de usuario (UI) fortificada.
  - Está hecho a medida para hacer disponible el contenido de las tiendas y los servicios de Amazon.

También podemos nombrar otros que o bien o han **desaparecido,** han pasado a ser **mantenidos** por **comunidades** de **usuarios,** o bien son **intentos** de **fabricantes** por **crear** sus **propios SO** y **desvincularse** de **Google** (véase la Figura 1).

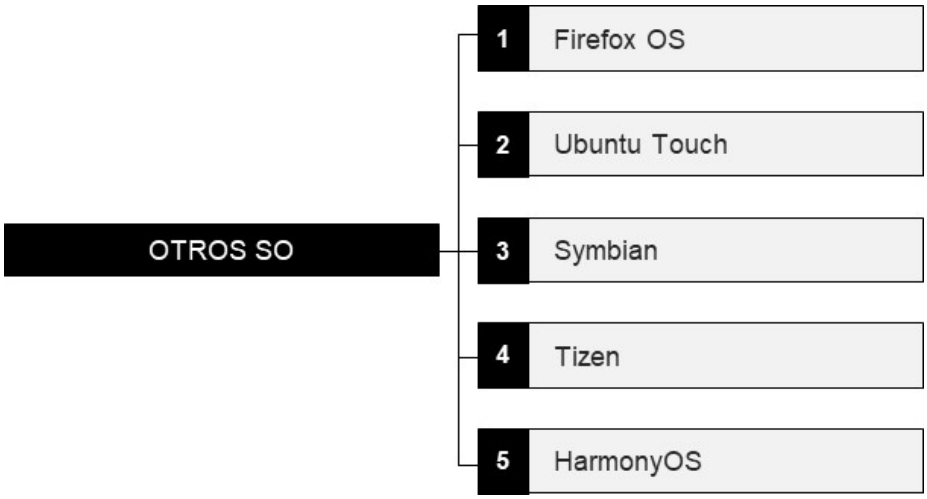


Figura 1. Otros ejemplos de SO. Fuente: elaboración propia.

1.3. Introducción al desarrollo de aplicaciones móviles

El desarrollo de aplicaciones móviles es el **proceso** en el que se **desarrolla** un ***software*** para **dispositivos móviles** a fin de **realizar** una **determinada tarea.** Estas pueden venir **preinstaladas** o ser **descargadas** por las **personas usuarias** desde las distintas plataformas móviles (Android, iOS o Windows Phone).

¿Qué es una aplicación móvil?

Una aplicación móvil o *app* es un **tipo** de **software** diseñado específicamente para **ejecutarse** en **dispositivos móviles,** como teléfonos inteligentes o tabletas. Aunque por lo general son **programas pequeños** con funciones específicas, logran **ofrecer** a quienes las usan **servicios** y **experiencias satisfactorias** de **alta calidad.**

En contraposición con las aplicaciones de escritorio, las **aplicaciones móviles no** forman **parte** de los **sistemas** de ***software*** **integrados** y, en su lugar, **ofrecen** una **funcionalidad específica** (p. ej., podría ser un juego, un bloc de notas o un navegador web adaptado para dispositivos móviles).

En los **primeros dispositivos móviles** las **aplicaciones evitaban** la **multifuncionalidad** debido a las **limitaciones** de ***hardware.*** Sin embargo, a pesar de que los actuales son mucho más avanzados y potentes, las **aplicaciones móviles** continúan siendo **especializadas** en sus funciones, algo que permite a quienes las usan **elegir** exactamente qué **funciones** desean en sus dispositivos.

Aplicaciones nativas

El desarrollo de aplicaciones nativas supone **utilizar** el **lenguaje nativo** de la propia **plataforma** para la construcción:

- Android. Kotlin (hasta 2019 era Java; se puede seguir usando, pero Google recomienda el uso de Kotlin para proyectos nuevos).
- iOS. Swift.

Sus **ventajas** se presentan a continuación:

- Estamos trabajando directamente con las funciones y el SDK oficial del SO, por lo que no tendremos límite alguno en esa plataforma.
- La UX y el entorno visual ofrecen un rendimiento del 100 %, ya que se hace uso directo de las funciones visuales y los elementos gráficos que ofrece cada sistema.
- Podemos acceder a todo tipo de sensores y *hardware* del dispositivo, así como conectarnos a *hardware* externo.
- Tenemos la posibilidad de acceder a cualquier funcionalidad del dispositivo.

Aplicaciones híbridas

El **desarrollo** de una **aplicación sencilla** o que use **elementos comunes** en las distintas plataformas puede realizarse a través de un ***framework*** de **manera híbrida.** De esta manera, se hace **un solo desarrollo** que se exportará a distintas plataformas, algo que **abarata** su **creación** debido a la necesidad de menos personas y tiempo.

Existe una **gran variedad** de **librerías** *(frameworks)* que nos permiten realizar este **desarrollo híbrido.** Algunas de las más utilizadas son estas:

- Xamarin. Es una herramienta que utilizan quienes programan para crear aplicaciones móviles. Permite crear código en el lenguaje C# de Microsoft, el mismo que hace posible traducir el código del lenguaje para diferentes dispositivos, tanto Android como iOS y Windows Phone.

- Ionic. Es el *framework* con el que puedes utilizar hojas de estilo en cascada (CSS) para crear increíbles diseños que parezcan nativos; no obstante, su verdadero potencial es lo sincronizado que está con AngularJS. Otra gran ventaja es la interfaz de línea de comandos, que está llena de interesantes características, como emuladores integrados y un *app packager* basado en Apache Cordova.

- PhoneGap. Es la herramienta por excelencia de Adobe para desarrollar aplicaciones móviles y una distribución de código abierto de Apache Cordova. Puede reutilizar las habilidades de desarrollo web existentes a fin de crear rápidamente aplicaciones híbridas, creadas con lenguaje de etiquetas de hipertexto (HTML), CSS y JavaScript, para múltiples plataformas con una única base de código con el objetivo de que pueda llegar a su audiencia sin importar su dispositivo.

- React Native. Nació en 2015 y fue creado por Meta Platforms. Basado en React, utiliza JSX para desarrollar y renderizar las aplicaciones. No permite utilizar el mismo código para todas las plataformas, sino que ofrece componentes para cada una de ellas y deja a las personas desarrolladoras la responsabilidad de ver qué componentes se asemejan más al comportamiento nativo que se esperaría en cada caso.

Las aplicaciones híbridas son, esencialmente, páginas web incrustadas en una móvil a través de un WebView.

Presentan estas **características:**

- Pueden ejecutar el mismo código con independencia de cuál sea la plataforma objetivo, aunque también cabe la posibilidad de usar un código específico para cada una.

- Tienen la facultad de acceder a alguna capa nativa de los dispositivos (cámara, GPS, etc.) usando *plugins,* aunque, en algunos casos, no es posible o requiere desarrollos independientes para cada sistema.

1.4. Entornos de desarrollo y emuladores en Android

Android Studio es el **IDE oficial** de **Google** en el desarrollo de aplicaciones para Android; actualmente **soporta** los lenguajes de **Java** y **Kotlin** (este último es el estándar actual y el recomendado por Google a la hora de desarrollar los nuevos proyectos). Para poder ejecutarlo en nuestro ordenador necesitamos cumplir con unos **requisitos mínimos:**

- La cantidad de memoria RAM recomendada es de ocho *gigabytes* (GB).
- Nuestro procesador debe tener capacidades para virtualizar (emulador).

Una de las **funciones** que nos ofrece Android Studio es la **posibilidad** de **montar** una **máquina virtual (VM)** de Android con la que poder hacer **uso** del **SO** de Google y de sus **aplicaciones** desde nuestro ordenador.

Esta versión ha supuesto para Google un gran desafío, ya que la compañía ha trabajado duro para implementar una serie de novedades, mejoras y cambios que las personas usuarias llevaban tiempo pidiendo y que, por motivos técnicos, no se habían podido incorporar antes.

En la sección de A fondo localizarás la documentación oficial de Gradle y una web para comenzar a programar en Android.

Android Emulator ejecuta una **pila completa** de sistema de Android, hasta el **nivel** del ***kernel,*** que incluye un conjunto de **aplicaciones preinstaladas** (como el teléfono) a las que puedes acceder desde tus aplicaciones. Además, cabe la posibilidad de **elegir** la **versión** del sistema que deseas ejecutar en el emulador al crear un **dispositivo virtual** de **Android** (AVD).

Las **imágenes** de sistema de Android disponibles a través del Administrador de AVD contienen los siguientes **elementos:**

- Código para el *kernel* de Linux de Android.
- Bibliotecas nativas.
- VM.
- Diferentes paquetes de Android (p. ej., las aplicaciones preinstaladas y el *framework* de Android).

Hay que tener en cuenta que desde el emulador **no** se puede **acceder** a **todas** las **características** del **dispositivo** y que **no incluye *hardware* virtual** para lo siguiente:

- Wifi.
- Bluetooth.
- Comunicación de campo cercano (NFC).
- Inserción/expulsión de tarjetas SD.
- Auriculares conectados a dispositivos.
- Bus de serie universal (USB).

Además de Android Studio, podemos utilizar otros IDE para crear aplicaciones en Android. El más famoso y valorado es **IntelliJ IDEA,** de la compañía **JetBrains,** la misma que desarrolla actualmente Android Studio. A diferencia de este último, es un **IDE** de **propósito general** que **soporta** los lenguajes de **Java** y **Kotlin,** al que podemos **añadir *plugins*** o **extensiones** para **administrar** a la par que **desarrollar** proyectos Android.

Material de estudio

En la sección de A fondo encontrarás la página web de JetBrains.

1.5. Desarrollo de aplicaciones móviles en Android

Un **elemento imprescindible** para el desarrollo de Android es el **uso** del **SDK.** Se trata de una serie de **interfaces** de **programación** de **aplicaciones** (API) que facilita el **desarrollo** de las aplicaciones móviles.

Algunas de las **características** que posee son estas:

- Licencias, distribución y desarrollo gratuitos; tampoco hay procesos de aprobación del *software*.
- Acceso al *hardware* de wifi, GPS, Bluetooth y telefonía, lo que permite realizar y recibir tanto llamadas como mensajes cortos (SMS).
- Control completo de multimedia, lo que incluye la cámara y el micrófono.
- API para los sensores (acelerómetros y brújula).
- Mensajes entre procesos (IPC).
- Almacenes de datos compartidos, SQLite y acceso a tarjetas SD.
- Aplicaciones y procesos en segundo plano.
- *Widgets* para la pantalla de Inicio (Escritorio).
- Integración de los resultados de búsqueda de la aplicación con los del sistema.
- Uso de mapas y sus controles desde las aplicaciones.
- Aceleración gráfica por *hardware,* lo que incluye OpenGL ES 2.0 para los 3D.
- Controles de Google Maps en nuestras aplicaciones.
- Procesos y servicios en segundo plano.

- Proveedores de contenidos compartidos y comunicación entre procesos.
- No diferenciación entre aplicaciones nativas y de terceros. Todas se crean igual, con el mismo aspecto e iguales posibilidades de usar el *hardware* y las API.
- *Widgets* de escritorio.

Android Studio dispone de una **herramienta** de **gestión, SDK Manager,** que nos permitirá **instalar** y **administrar diferentes versiones** del SDK dependiendo de la versión de Android que queramos utilizar. Debemos tener en cuenta que las más modernas poseen características que quizá no encontremos en las anteriores, pero, a su vez, habrá menos dispositivos donde podamos instalar nuestra aplicación.

Este punto es muy importante cuando generamos un proyecto. Hemos de tener un balance entre no coger un SDK mínimo muy antiguo, que nos limite la funcionalidad, ni uno muy moderno que nos restrinja el número de dispositivos compatibles con la aplicación.

En la sección de A fondo localizarás un resumen de todas las versiones de Android API.

Capas en el SO

Los SO implementan **multitud** de **servicios** y **funciones,** como, entre otros, la gestión de entrada y salida, el control de los programas y la gestión de la memoria. Estos procesos se encuentran en **capas,** construyendo una **jerarquía** de **niveles** de **abstracción,** de modo que cada uno proporciona un **conjunto específico** de **funciones primitivas** que podrán usar las funciones de la capa superior.

- Aplicaciones. Se compone de las nativas y las de terceros, así como las de desarrollo.
- *Framework* de aplicaciones. Formado por las clases que se utilizan para crear aplicaciones Android (actividades, servicios, *views,* proveedores de contenidos, etc.).
- Android Runtime. Es lo que hace a Android diferente de una distribución de Linux embebido. Se compone de las librerías *core* (núcleo) y de Dalvik, la VM de Java basada en registros y que cuenta con el núcleo de Linux para la gestión tanto de hilos como el manejo de memoria a bajo nivel.
- Librerías. Compuestas por las librerías generales.
- Núcleo de Linux. Es la capa encargada de los controladores *(drivers)* del *hardware,* los procesos, la memoria, seguridad, red y gestión de energía. Es la que abstrae el resto de las capas del *hardware.*

Las **clases más importantes** para el desarrollo de aplicaciones en Android son estas:

- ActivityManager. Controla el ciclo de vida de las actividades.
- View. Se usa para construir interfaces en las actividades.
- NotificationManager. Mecanismo para mostrar avisos a la persona usuaria.
- ContentProvider. Permite intercambiar datos de una manera estandarizada.
- ResourceManager. Hace posible el uso en la aplicación de recursos que no forman parte del código, como el lenguaje de marcado extensible (XML), recursos gráficos, audio, vídeo, etc.

AndroidManifest

El fichero AndroidManifest.xml se encuentra en **todos** los **proyectos** de **Android** y su **función** es **declarar** una serie de **metadatos** de la **aplicación** que el dispositivo debe **conocer antes** de **instalarla.** Describe información esencial sobre tu aplicación para las herramientas de compilación de Android, su SO y Google Play Store.

En este fichero se indican **aspectos** tales como los siguientes:

- El nombre del paquete Java para la aplicación, que sirve como un identificador único.
- Los componentes de la aplicación, como las actividades, los servicios, los receptores de mensajes y los proveedores de contenido que la integran. También nombra las clases que implementa cada uno de los componentes y publica sus capacidades (p. ej., los mensajes Intent con los que pueden funcionar). Estas declaraciones notifican al sistema Android los componentes y las condiciones para el lanzamiento.
- Los procesos que alojan los componentes de la aplicación.
- Los permisos que debe tener la aplicación para acceder a las partes protegidas de una API e interactuar con otras aplicaciones. Este archivo también declara los permisos que otros han de tener para interactuar con los componentes de la aplicación.
- Las clases Instrumentation, que proporcionan un perfil y otra información mientras la aplicación se ejecuta. Estas declaraciones, enumeradas en el archivo, están en el manifiesto solo mientras la aplicación se desarrolla y se eliminan antes de su publicación.
- El nivel mínimo de Android API que requiere la aplicación.
- Las bibliotecas con las que debe vincularse la aplicación.

Material de estudio

En la sección de A fondo encontrarás una página web con información sobre AndroidManifest.

1.6. Ciclo de vida de una aplicación en Android

Cuando se empieza a programar en un lenguaje como Java lo primero que se enseña es el **método** main() , que es el **punto** al que **llamará** el SO cuando vayamos a **arrancar** nuestra **aplicación.**

En Android no existe como tal, pero sí hay varios métodos de nuestra actividad a los que el SO llamará cuando ocurran eventos importantes.

La **clase Activity** es un **componente crucial** de una aplicación Android y la **forma** en la que se **inician,** así como se **organizan,** las actividades es una **parte fundamental** del **modelo** de **aplicación** de la plataforma.

El **ciclo** de **vida** de una aplicación Android se refiere a los **estados** y las **transiciones** que atraviesa desde que se **inicia** hasta que se **cierra.** Su comprensión es **crucial** para **desarrollar aplicaciones eficientes** y **evitar problemas** de **rendimiento** o **pérdida** de **datos.** A continuación, podemos ver los diferentes estados y métodos clave que forman parte del ciclo de vida de una Activity en una aplicación Android.

Estados del ciclo de vida de una Activity

Una Activity es una **sola pantalla** con **una UI** en una aplicación Android y a lo largo de su ciclo de vida puede estar en **varios estados:**

- *Running* (en ejecución). Está en primer plano y es interactiva.
- *Paused* (en pausa). Se encuentra parcialmente oculta, pero aún visible (p. ej., por una ventana emergente o un diálogo), y no es interactiva.

- *Stopped* (detenida). Está completamente oculta y no es visible, aunque sigue en memoria.
- *Destroyed* (destruida). Se ha eliminado de la memoria.

Métodos clave del ciclo de vida

Para gestionar estos estados Android proporciona **varios métodos** en la clase Activity que se **llaman automáticamente** al **cambiar** el **estado** de la actividad:

- onCreate() . Este método se llama cuando la actividad se crea por primera vez. Aquí es donde inicializas los componentes básicos de la actividad, como la UI y los recursos necesarios. Este es el primer método que se ejecuta y es esencial para configurar la actividad.
- onStart() . Llamado en el momento en el que la Activity se vuelve visible para la persona usuaria, pero aún no es interactiva. Es el lugar adecuado a fin de realizar las configuraciones finales antes de que sea visible.
- onResume() . Se llama cuando la Activity comienza a interactuar con quien la usa. Está en primer plano y es el estado en el que permanece la mayor parte del tiempo mientras está en uso. Aquí es donde se reanuda cualquier tarea que estaba pausada, a saber, la reproducción de un vídeo.
- onPause() . Este método se llama cuando la Activity pierde el enfoque, pero aún es visible. Aquí es donde debes pausar las operaciones que no deberían continuar cuando no está en primer plano, como detener animaciones o liberar recursos que no son necesarios hasta que vuelva a estar en primer plano.
- onStop() . Llamado en el momento en el que la Activity ya no es visible para la persona usuaria. Puedes usarlo para realizar operaciones más pesadas de guardado de datos o liberar recursos que no se necesitan mientras está detenida.

- onDestroy() . Se llama antes de que la Activity se destruya por completo, ya sea porque el sistema está recuperando memoria o porque se está cerrando intencionalmente. Aquí puedes hacer la limpieza final de recursos.
- onRestart() . Este método se llama cuando la Activity se está reiniciando después de haberse detenido. Le sigue onStart() y es útil para realizar cualquier inicialización que se necesite cuando vuelve a ser visible.

Ejemplo del ciclo de vida en acción

Imagina que la persona usuaria lanza una aplicación, la navega y, luego, la minimiza:

- Lanzar la aplicación. Se llama a onCreate() , seguido de onStart() y onResume() . La aplicación ahora está en el estado *running*.
- Minimizarla. Se llama a onPause() , y luego, a onStop() . La Activity ahora está en el estado *stopped*.
- Volver a la aplicación. Se llama a onRestart() , seguido de onStart() y onResume() . La Activity vuelve al estado *running*.
- Cerrarla. Se llama a onPause() , onStop() , y finalmente, onDestroy() .

Gestión de la configuración y persistencia de datos

onSaveInstanceState(Bundle outState) y onRestoreInstanceState(Bundle savedInstanceState) son **métodos cruciales** para **manejar** la **persistencia** del **estado** de la actividad durante **cambios** en la **configuración,** como la rotación de la pantalla:

- onSaveInstanceState se utiliza para guardar los datos importantes antes de que la actividad se destruya.
- onRestoreInstanceState restaura estos datos cuando la actividad se recrea.

Consideraciones de diseño

Al diseñar y desarrollar una aplicación Android se debe tener presente la **gestión** de **recursos** del **dispositivo** y la **optimización** de **memoria:**

- Recursos limitados. Como el SO puede destruir y recrear las actividades en cualquier momento para liberar recursos, es importante diseñar la aplicación para manejar correctamente estas transiciones.
- Optimización de la memoria. Has de tener cuidado con la gestión de recursos, especialmente en los métodos `onPause()`, `onStop()` y `onDestroy()`, a fin de evitar fugas de memoria y otros problemas relacionados con la gestión ineficiente de los recursos.

Comprender y manejar correctamente el ciclo de vida de las actividades es fundamental para desarrollar aplicaciones Android robustas y eficientes. Permite asegurar su correcto comportamiento en diferentes situaciones (p. ej., cuando hay cambios en la configuración o el sistema recupera recursos).

1.7. Referencias bibliográficas

Android Developers. (s. f.). *Cómo descargar Android Studio y App Tools*. https://developer.android.com/studio?hl=es-419

Anuar-UNIR. (2024a). *PMDM_2024-2025/Tema 1*. https://github.com/Anuar-UNIR/PMDM_2024-2025/tree/main/Tema%201

Anuar-UNIR. (2024b). *PMDM_2024-2025/Tema 1/Entrenamiento 5*. https://github.com/Anuar-UNIR/PMDM_2024-2025/tree/main/Tema%201/Entrenamiento%205

ElPotroST. (2024, enero 29). *Hola mundo en Android Studio 2023–fácil y rápido–tu primera aplicación* [Vídeo]. YouTube. https://www.youtube.com/watch?v=HtPzUo8vQE0

Android API Levels

Página de Android API levels

Este es un resumen de todas sus versiones y sus correspondientes identificadores para las personas desarrolladoras de Android, así como del nivel de uso acumulativo.

Documentación oficial de Gradle

Gradle. (s. f.). *Gradle quick start.* https://docs.gradle.org/current/userguide/quick_start.html

Gradle es una herramienta de automatización de la construcción de nuestro código que bebe de las aportaciones que han realizado herramientas, como Apache Ant y Apache Maven, pero intenta llevarlo todo un paso más allá. En esta documentación encontrarás toda la información necesaria. Los proyectos de Android Studio trabajan con Gradle.

AndroidManifest

Google for Developers. (2025, abril 3). *Descripción general del manifiesto de la app*. https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419

Todos los proyectos de aplicaciones deben tener un archivo AndroidManifest.xml, con ese nombre preciso, en la raíz del conjunto de orígenes del proyecto. Este describe información esencial sobre tu aplicación para las herramientas de compilación de Android, su SO y Google Play Store. En esta documentación encontraras toda la información, los elementos y la configuración de este importante fichero.

Android developers

Google for Developers. (s. f.). *Cómo descargar Android Studio y App Tools*. https://developer.android.com/studio?hl=es-419

Web oficial para quienes desarrollan en Android, donde podemos encontrar toda la información para comenzar a programar, así como descargar nuestro IDE.

JetBrains

Página de JetBrains

Portal web de JetBrains, donde se pueden encontrar todos sus productos y proyectos.

Entrenamiento 1

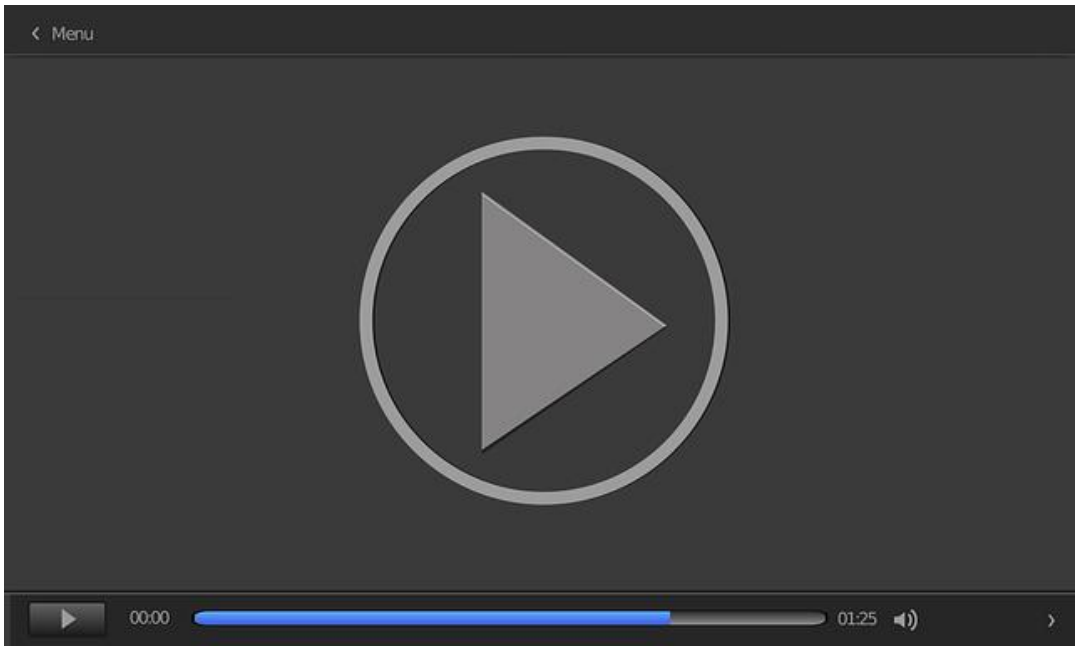
- Desarrolla un **cronograma** con los **SO móviles** más relevantes a lo largo de la historia, junto con sus **fechas** de **lanzamiento** y algunas de sus **características clave**. Debe abarcar desde los primeros (años noventa) hasta los más recientes.
- El desarrollo paso a paso es este:
  - Realiza una búsqueda por Internet.
  - Plasma los resultados encontrados.
- La solución se encuentra en Anuar-UNIR (2024a).

Entrenamiento 2

- Elabora un **cronograma** de las **diferentes versiones** de **Android,** desde su lanzamiento hasta la actualizad.
- El desarrollo paso a paso es el siguiente:
  - Realiza una búsqueda por Internet.
  - Plasma los resultados encontrados.
- La solución se encuentra en Anuar-UNIR (2024a).

Entrenamiento 3

- **InstalaAndroid Studio** y su **SDK.**
- El desarrollo paso a paso es este:
  - Descarga la aplicación desde Android Developers (s. f.).
  - Sigue los pasos de instalación.
- En el vídeo *Instalación Android Studio* se explica la solución.

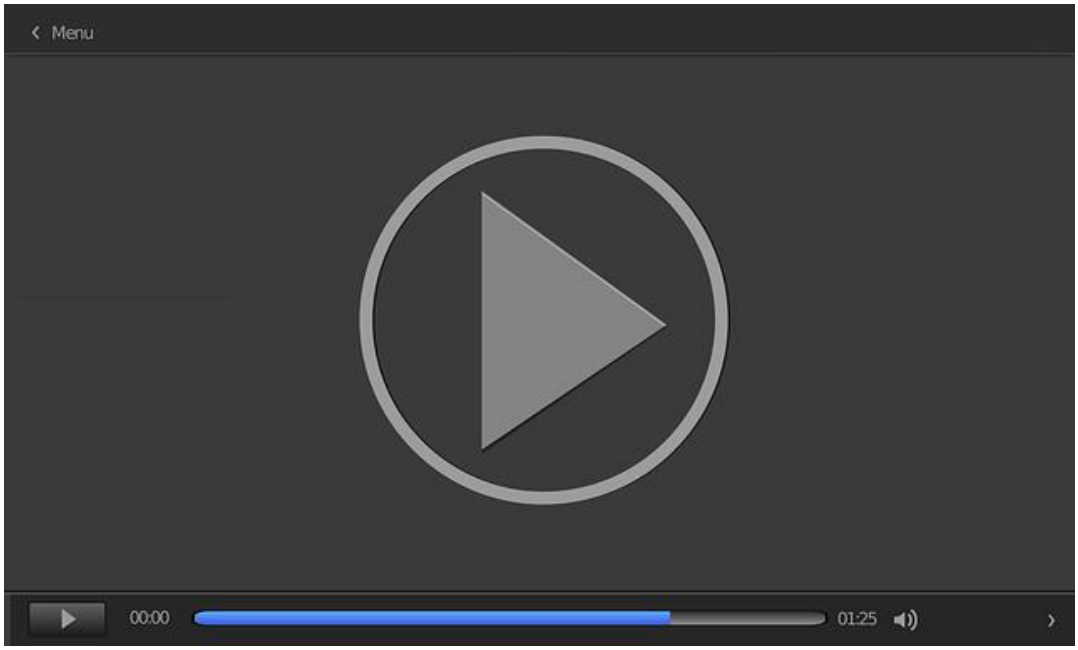


Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=1ef92cbb-9b3b-4b94-820b-b1e700de11eb

Entrenamiento 4

- **Configurados dispositivos** en el **emulador** de **Android Studio.**
- El desarrollo paso a paso es el siguiente:
  - Abre el Configurador de Dispositivos de Android.
  - Configura un dispositivo.
  - Repite el proceso para el segundo.
- En el vídeo *Instalación Android Studio* se explica la solución.



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=1ef92cbb-9b3b-4b94-820b-b1e700de11eb

Entrenamiento 5

- Genera un **primer proyecto** en **Android Studio** en **Java** y **mostrar** el mensaje «**Hola mundo**» por la **Activity principal.**
- El desarrollo paso a paso se encuentra en ElPotroST (2024).
- La solución se encuentra en Anuar-UNIR (2024b).