Introducción a la programación en dispositivos móviles.

1. Ficha pedagógica.

Temporalización: 10 horas.	
RESULTADOS DE APRENDIZAJE	RA1
CONTENIDOS	
– Tecnologías disponibles. – Entornos integrados de trabajo. – Módulos para el desarrollo de aplicaciones móviles. – Emuladores. - Jerarquía de clases según configuración y perfil. – Perfiles. Características. Arquitectura y requerimientos. Dispositivos soportados.	
CRITERIO DE EVALUACIÓN	1a, 1b, 1c, 1d, 1e, 1f, 1g, 1h

2. Contenidos.

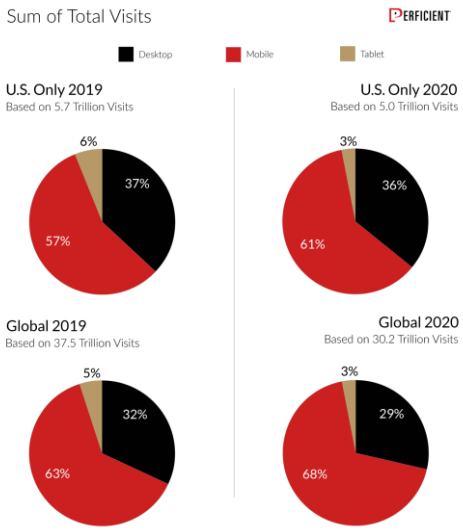
2.1. Introducción.

En los últimos 20 años la informática de consumo ha experimentado un cambio significativo, lejos quedan los tiempos de los años 60 y 70 donde existían grandes servidores centrales “Mainframe” a los cuales los equipos terminales se conectaban, o los años 80 y 90 donde se pasó a disponer de equipos sobremesa y portátiles en los hogares (la mayoria de veces solo se disponía de un equipo en el hogar compartido por todos los miembros de la familia). A partir de inicio del siglo los sistemas operativos empiezan a extenderse más allá de los ordenadores clásico, en primer lugar, a los móviles, pasando a continuación a las tabletas los televisores y en la actualidad encontrar en una gran cantidad de dispositivos: teléfonos, tabletas, televisores, relojes, coches, gps...

Esta evolución ha provocado que un alto porcentaje de usuarios finales ya no utilicen en su día a día los clásicos ordenadores, sino dispositivos móviles o que si bien no son móviles utilizan las tecnologías propias de dichos dispositivos. Estos dispositivos poseen características diferentes a los ordenadores clásicos:

- La capacidad de cómputo y memoria principal suele ser menor que los ordenadores clásicos (aunque esto cada vez es menos frecuente).
- Gran variedad de dispositivos en cuanto a procesador, memoria, almacenamiento y capacidad de comunicación

- El consumo de energía es un factor determinante, usando en muchas ocasiones baterías, limitando la arquitectura de los procesadores.
- Los periféricos de entrada integrados y variados, aunque en la mayor parte de los dispositivos son pantallas táctiles o mandos remotos, y de forma excepcional otros como sensores de movimiento.
- Sistemas operativos con objetivos y funcionalidad distintas a los clásicos de escritorio.



Los sistemas operativos y las tecnologías para el desarrollo de aplicaciones en este tipo de dispositivos se han de adaptar a las características anteriores, siendo diferentes a las de las aplicaciones de escritorio clásicas, por ejemplo, para introducir datos no se dispone del teclado, sino se ha de establecer otros mecanismos para esto, de igual forma el tamaño de las pantallas en caso de existir son en su mayoría de pocas pulgadas.

2.2. Evolución histórica.

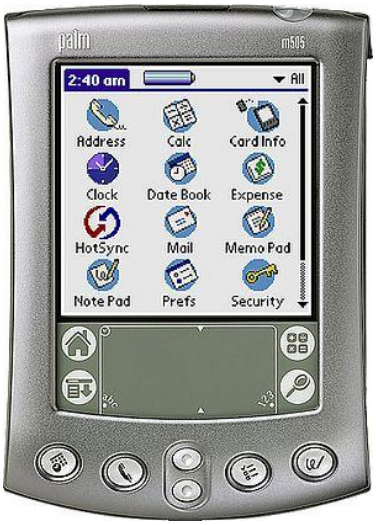
Junto con el aumento del uso de los dispositivos móviles, su incremento de potencia y el aumento de las velocidades de transmisión de datos, se produjo una evolución del software que corrían sobre los mismos, siendo similar a la progresión que se experimentó en los ordenadores clásicos. Este punto se centra en los teléfonos móviles, ya que son el tipo de dispositivos más característicos de los dispositivos móviles.

El primer teléfono móvil (usando tecnologías de antenas agrupadas en celdas o células) como tal (aunque existen dispositivos previos basados en radio, pero no con todas las características) se creó en 1983 por la empresa motorola, estando implementada la lógica del dispositivo se realizaba con hardware (agenda, conexión a red o gestión de llamada).

Desde el primer móvil hasta la aparición de los primeros sistemas operativos especiales para móviles, la funcionalidad de estos se realizaba por hardware o con firmware muy sencillo, evolucionando en cuanto a la tecnología de red 1G, 2G, 2.5G, 3G, y si bien era posible desarrollar aplicaciones para estos dispositivos se encontraba muy ligados a estos, si se quería portar una aplicación a otra plataforma, era necesario rehacer el código para adaptarlo.

A mediados de los años 90 comienzan a aparecer sistemas operativos para dispositivos móviles, los más conocidos son:

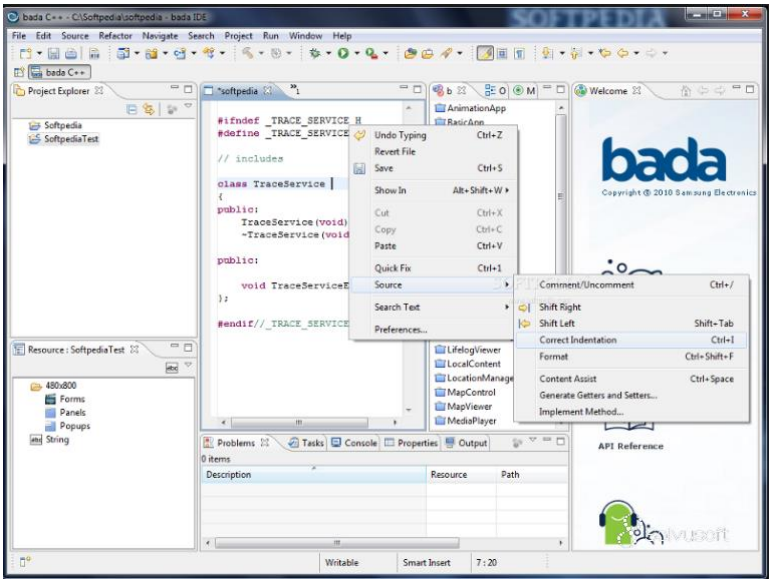
Palm Os. Sistema operativo diseñado especialmente para dispositivos móviles, precursor de los actuales sistemas, especial para dispositivos de la misma empresa. Escrito en C y C++. Lanzamiento en 1996.



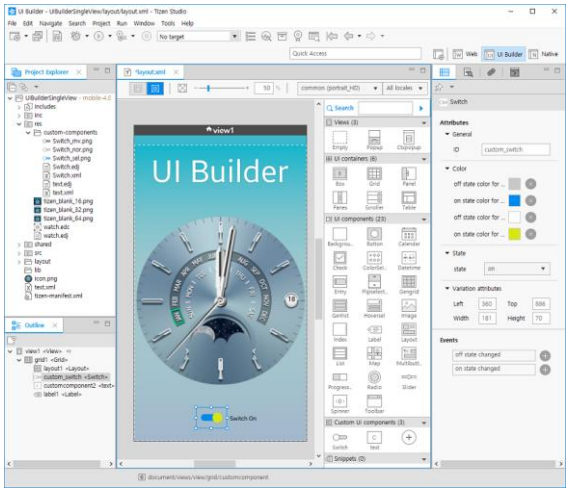
Symbian. El primero en llegar al gran público de la mano de Nokia, lanzado en 1997, se podían desarrollar aplicaciones para cualquier dispositivo que tuviera este sistema operativo, la última versión es del 2012, desarrollado en C++. Se desarrollaron SDK para facilitar el desarrollo.

Backberry OS. Sistema operativo de la empresa Backberry desarrollado para sus teléfonos móviles, apareció en el año 1999, alcanzando gran popularidad gracias a su seguridad, la inclusión de mensajería gratuita para dispositivos móviles de esa empresa, una excelente gestión de correo electrónico y su fama de durabilidad. Los lenguajes que se utilizaron fueron Java y C++.

Bada. Sistema operativo desarrollado por Samsung para uso en sus dispositivos, tanto móviles, televisiones u otros, se lanzó en el año 2010 utilizando los lenguajes C y C++, tomando como base el núcleo de QNX (sistema operativo en tiempo real). No triunfo, algunas fuentes indican que la causa principal fue la limitación en el desarrollo de aplicaciones por terceros.

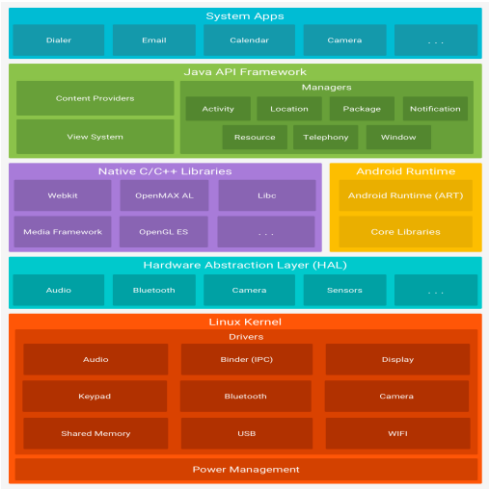


Tizen: Sistema operativo basado en GNU/Linux y la plataforma Linux de Sambsumg, lanzamiento en el 2012, con núcleo Linux y desarrollado en HTML y C++. Las aplicaciones se gestionan con RPM. En principo las aplicaicones que funcionan en Linux debían funcionar en Tizen. La última versión en de finales del 2022. Posee un SDK llamado Tizen Studio.

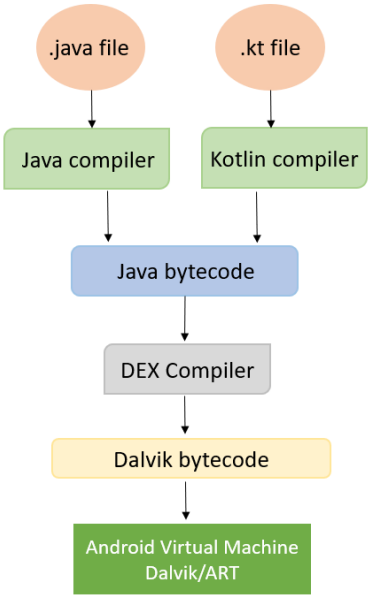


Estos sistemas operativos en la actualidad han quedado en desuso, se encuentran abandonados excepto Tizen, pero con una cuota de mercado prácticamente inexistente, estando el mercado dominado por dos sistemas operativos:

Android: Núcleo basado en Linux, se puede encontrar prácticamente en cualquier tipo de dispositivo actual. La primera versión se lanzó al mercado en 2008, experimentando un crecimiento en su uso exponencial, desbancando en poco tiempo a Symbian que hasta ese momento era el sistema operativo para móviles más popular.



En los inicios del sistema la ejecución de aplicaciones sobre una máquina virtual denominada Dalvik, similar a la JVM, encontrándose optimizada para dispositivos con poca memoria y poder ejecutar varias máquinas virtuales al mismo tiempo, realizando una compilación “Just In Time” para mejorar el rendimiento.



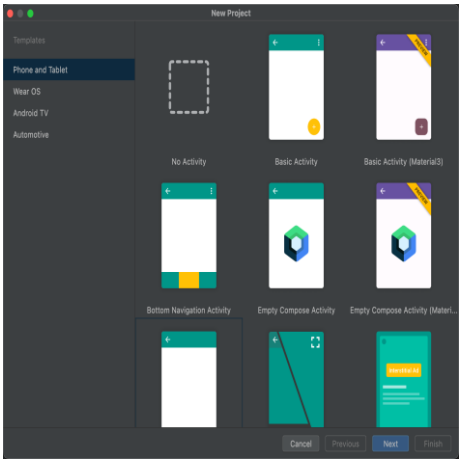
Actualmente Android utiliza un nuevo entorno de ejecución denominado Android Runtime (ART), cambiando la filosofía de JIT por AOT (compilación anticipada) en la cual se crea un archivo compilado al instalar la aplicación no siendo necesaria la "recompilación" en cada ejecución. Otras mejoras son:

- Recolector de basura más optimizado.
- Reducción de la fragmentación de memoria.
- Mejoras en la depuración.
- Mejor diagnóstico de excepciones.

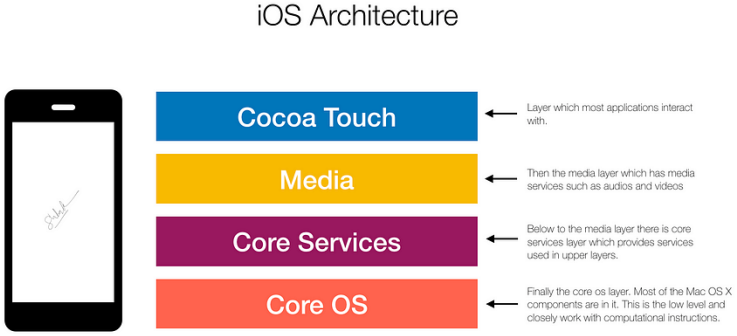
Otra de las características de Android (y en general de los sistemas operativos actuales) es la existencia de repositorios o tiendas en las que se encuentran las aplicaciones, estas tiendas pueden ser oficiales o de terceros.

En cuanto al desarrollo nativo, si bien inicialmente el desarrollo se realizaba en Java, en la actualidad es posible desarrollar prácticamente en cualquier lenguaje que se pueda traducir al "ByteCodes", siendo los más utilizados Java y Kotlin, aunque se puede programar en C++, Python, tecnologías web como Javascript, Dart...

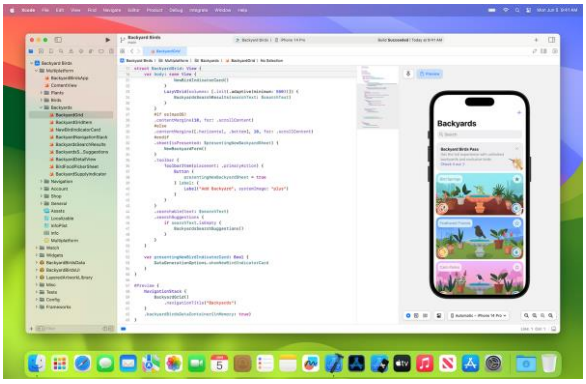
De forma oficial cuenta con un entorno de desarrollo llamado Android Studio, aunque es posible usar otros IDES y tecnologías.



IOS: Sistema operativo de la empresa Apple Inc, usado tanto en los teléfonos móviles como en el reproductor de música y la tableta (desde 2019 con variaciones) de la misma empresa. Lanzado en el 2007 con el primer "iPhone". Es un sistema cerrado y propietario. Posee una tienda de aplicaciones y las aplicaciones se encuentran compiladas y preparadas para un único tipo de hardware, al contrario que Android que puede correr en ARM, AMD64, MIPS..., con lo que no necesita máquina virtual.



Los lenguajes de desarrollo para iOs son Object-C y Swift, el primero está en desuso. Cuenta con un entorno de desarrollo denominado Xcode.



2.3. Tecnologías actuales.

En el mundo del desarrollo de aplicaciones para dispositivos móviles o multimedia actual existen multitud de tecnologías y filosofías, desde el clásico desarrollo nativo

hasta el desarrollo en web con el uso de librerías externas para dar el aspecto de aplicaciones nativas.

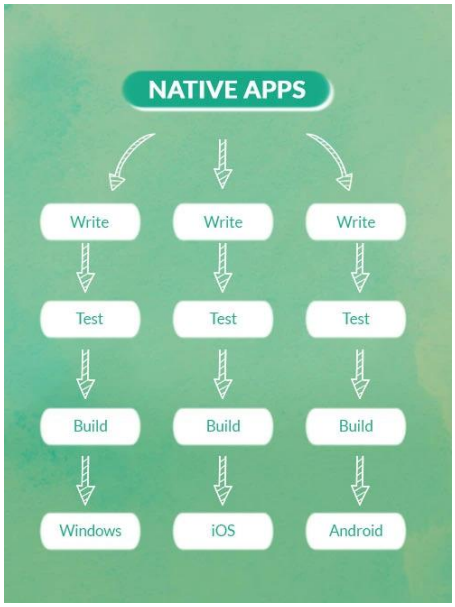
2.3.1. Nativas.

Las clásicas, se desarrollar para una plataforma concreta, como puede ser Android, Windows, iOS o Linux, en el caso de los móviles se suelen usar los lenguajes y herramientas proporcionados por el fabricante, como puede ser Kotlin y Android Studio, las ventajas de crear aplicaciones nativas son:

- Mayor rendimiento.
- Acceso a todas las características del dispositivo.
- Mayor seguridad.

En cuanto a las desventajas:

- Mayor coste de desarrollo para diferentes plataformas.
- Mayor tiempo de desarrollo.
- Problemas de actualización.
- Menos personal cualificado con conocimientos avanzados.

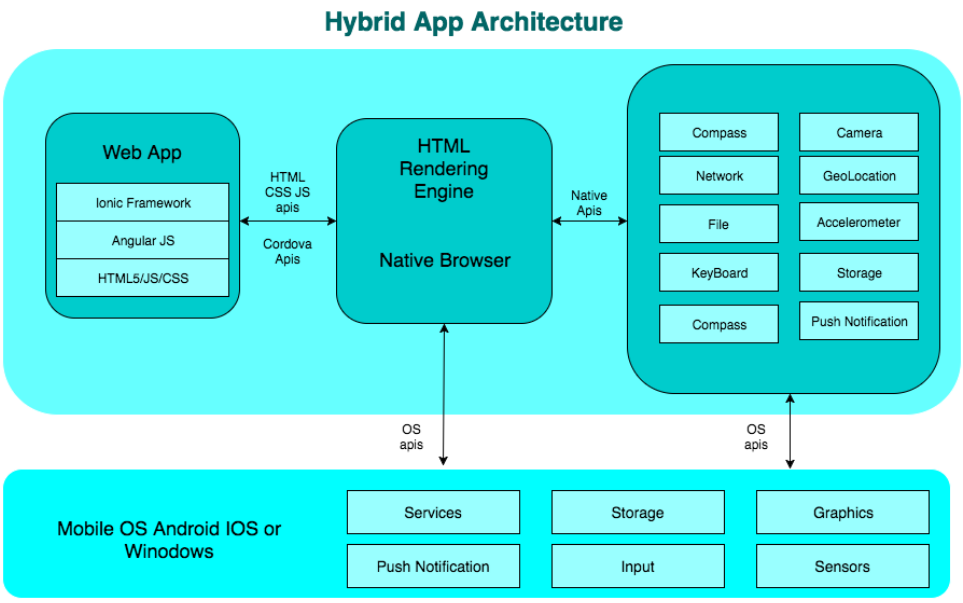


2.3.2. Híbridas.

Las tecnologías web son ampliamente utilizadas en la actualidad, saliendo de su ámbito, el navegador para extenderse prácticamente a cualquier dispositivo. Se basan en el uso estas 3 tecnologías:

- HTML para dar la estructura a la aplicación.
- CSS para el aspecto y algunos efectos visuales.
- Javascript con la lógica de la aplicación e interactuar con HTML y CSS.

A partir de estas tecnologías básicas se dispone de “frameworks” especializados como pueden ser Vue, React, Angular, Boostrap...



En el caso de que la aplicación móvil no necesite hacer uso de las características propias de dispositivo (GPS, cámara...), con estas 3 es suficiente, en caso de necesitar acceso a estos periféricos, existen librerías que permiten desde Javascript acceder a estos, por ejemplo, el SDK Ionic.

```
import { Geolocation } from '@capacitor/geolocation';
const printCurrentPosition = async () => {
  const coordinates = await Geolocation.getCurrentPosition();
  console.log('Current position:', coordinates);
};
```

Las ventajas de este tipo de aplicaciones son:

- Tecnologías universales.
- Una base de código.
- Desarrollo más rápido

En cuanto a las desventajas:

- Menor rendimiento.
- Acceso limitado al hardware.

El principio es abrir una instancia del navegador modificado para poder acceder a las características propias del dispositivo con librerías externas usando las tecnologías clásicas de las aplicaciones web. El código ejecutado es Javascript con lo que se puede ejecutar en cualquier dispositivo que disponga de un navegador.

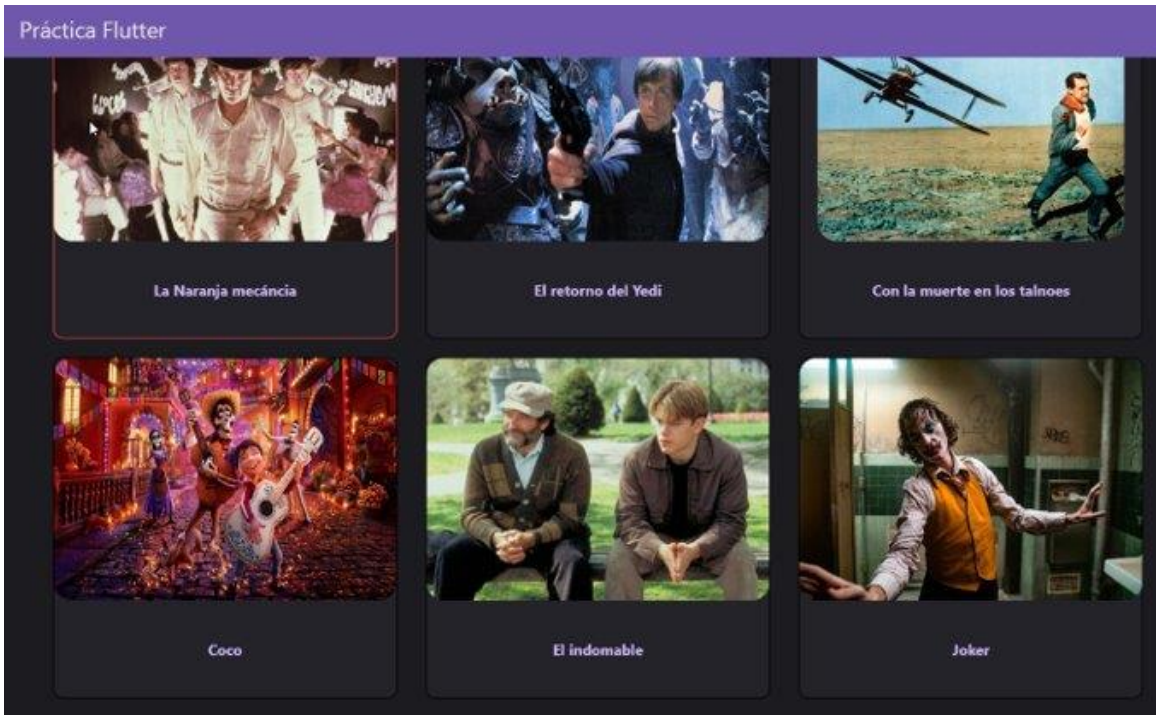
2.3.3. Multiplataforma.

Se tiene en primer lugar las aplicaciones nativas, que, si bien son las más optimizadas, su principal desventaja es el tiempo de desarrollo para múltiples plataformas y su mantenimiento, por otro lado las aplicaciones híbridas cuya principal ventaja es la facilidad de desarrollo, pero el rendimiento es menor.

Las aplicaciones multiplataforma intentan solucionar los problemas de los tipos anteriores manteniendo sus ventajas. Este tipo de aplicaciones se desarrollan una única vez, generando a partir del código versiones concretas y optimizadas para cada plataforma.

Los “frameworks” más usados en aplicaciones multiplataform son:

Flutter (Casi abandonado): SDK que usa el lenguaje Dart, creado por Google, pudiendo generar aplicaciones para Android, iOS, Fuchsia, Web, macOS, Windows y Linux.



ReactNative: Basado en el famoso “framework” web React (más utilizado en la actualidad), utiliza tecnologías web para el desarrollo, pero se produce una compilación de a código nativo (en el caso de Ionic, se ejecuta Javascript sobre el navegador).

Kotlin Multiplataform(KMP). Similar a Flutter, al cual, en teoría, ha desbancado, ya que Google apuesta por KMP. Permite escribir código compartido para múltiples plataformas, como Android, iOS, escritorio, web, e incluso servidores. Es especialmente útil en proyectos que necesitan compartir lógica entre una app de Android y una app de iOS, sin necesidad de escribir dos códigos completamente diferentes para ambas.

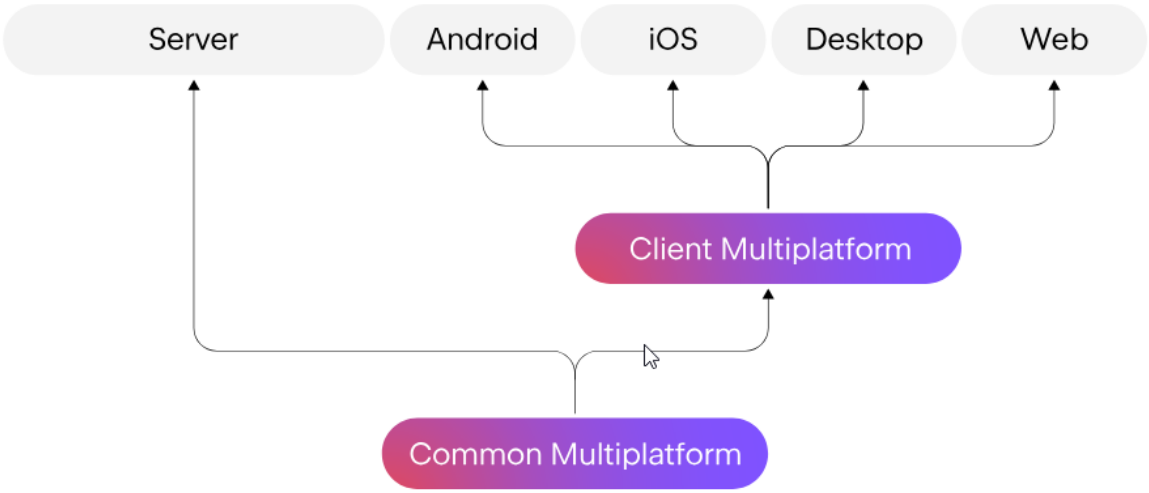


Imagen de arquitectura. Fuente: https://kotlinlang.org

Otros menos usados: Xamarin y Unity .

2.4. Android.

Junto con iOS, el sistema operativo más usado en la actualidad, de hecho, el tándem posee prácticamente el 100% del mercado. Su núcleo se basa en Linux y es posible crear aplicaciones, estás aplicaciones hacen uso del sistema operativo y del hardware de este.

Para el desarrollo existe un ecosistema compuesto de entre otros los siguientes elementos:

Lenguajes de Programación: Los dos lenguajes de programación principales para el desarrollo de aplicaciones Android son **Java y Kotlin**. Aunque Java ha sido el lenguaje predominante durante años, Kotlin se ha convertido en una opción popular debido a su modernidad y concisión.

SDK de Android: El SDK de Android (kit de desarrollo de software) es un conjunto de herramientas y bibliotecas que permiten a los desarrolladores crear aplicaciones Android. Incluye APIs para acceder a características del sistema, servicios y hardware del dispositivo. Cada SDK puede poseer APIs propias que no se encuentren en otras, por ejemplo.

Emulador de Android: Android Studio incluye un emulador de dispositivos Android que permite a los desarrolladores probar sus aplicaciones en una variedad de configuraciones de dispositivos, desde teléfonos hasta tablets y wearables.

Bibliotecas y Frameworks: Android ofrece una amplia gama de bibliotecas y frameworks que facilitan el desarrollo de aplicaciones, como la biblioteca de interfaz de usuario de Android (Android UI), Firebase (para servicios en la nube), y muchas otras.

Herramientas de Pruebas y Depuración: Android Studio incluye herramientas de depuración y pruebas que facilitan la identificación y solución de problemas en las aplicaciones, como el depurador y el analizador de rendimiento.

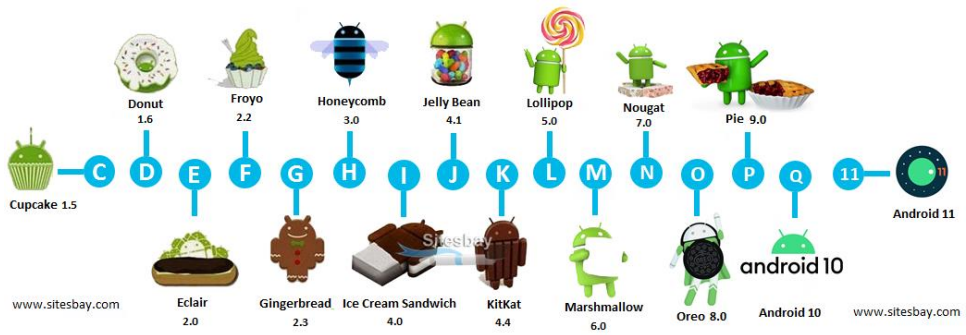
2.4.1. Versiones.

Android como cualquier sistema operativo evolucionan con el tiempo, de igual forma las clases y paquetes incluidas en el SDK, de ahí la razón de que ciertas aplicaciones dejen de funcionar en caso de no actualizarse a las últimas versiones de Android, ya que parte de las clases que utiliza ya no se encuentran disponibles. Un listado de las diferentes versiones con sus principales características:

1. **Android 1.0 (API nivel 1):** Esta fue la primera versión oficial de Android lanzada en septiembre de 2008. El SDK original incluía herramientas para el desarrollo de aplicaciones básicas de Android, como el manejo de interfaces de usuario y la gestión de eventos.
2. **Android 1.5 (Cupcake - API nivel 3):** Introdujo mejoras en la interfaz de usuario, como widgets y carpetas personalizables, además de mejoras en la cámara y la grabación de video.

3. **Android 2.0/2.1 (Eclair - API niveles 5 y 7):** Estas versiones añadieron soporte para pantallas de diferentes tamaños y resoluciones, así como mejoras en la navegación y las capacidades de sincronización.
4. **Android 2.2 (Froyo - API nivel 8):** Introdujo el soporte para Adobe Flash Player, mejoras en el rendimiento y capacidades de punto de acceso Wi-Fi.
5. **Android 2.3 (Gingerbread - API niveles 9 y 10):** Gingerbread incluyó mejoras en la velocidad y la eficiencia de la batería, así como mejoras en el teclado virtual.
6. **Android 3.0/3.1/3.2 (Honeycomb - API niveles 11, 12 y 13):** Esta versión fue diseñada específicamente para tablets y presentó una interfaz de usuario optimizada para pantallas más grandes.
7. **Android 4.0 (Ice Cream Sandwich - API nivel 14):** Unificó la experiencia de Android en dispositivos móviles y tabletas, además de agregar funciones de accesibilidad y NFC (Near Field Communication).
8. **Android 4.1/4.2/4.3 (Jelly Bean - API niveles 16, 17 y 18):** Introdujo el Proyecto Butter para mejorar el rendimiento y Project Svelte para mejorar la eficiencia en dispositivos con recursos limitados.
9. **Android 4.4 (KitKat - API nivel 19):** Optimizó el sistema operativo para funcionar mejor en dispositivos con especificaciones más bajas, además de introducir la compatibilidad con impresoras y el modo de pantalla completa.
10. **Android 5.0/5.1 (Lollipop - API niveles 21 y 22):** Lollipop presentó el diseño de Material Design, notificaciones mejoradas y cambios en la administración de la energía.
11. **Android 6.0 (Marshmallow - API nivel 23):** Marshmallow introdujo permisos de aplicaciones, Doze Mode para mejorar la duración de la batería y el sistema de control de volumen.
12. **Android 7.0/7.1 (Nougat - API niveles 24 y 25):** Nougat incluyó la capacidad de ejecutar dos aplicaciones al mismo tiempo (modo de pantalla dividida) y mejoras en la seguridad.
13. **Android 8.0/8.1 (Oreo - API niveles 26 y 27):** Oreo introdujo Project Treble para facilitar las actualizaciones de Android, así como mejoras en la duración de la batería y notificaciones.
14. **Android 9 (Pie - API nivel 28):** Pie presentó el control de gestos, bienestar digital y mejoras en la inteligencia artificial.

15. **Android 10 (Q - API nivel 29):** Android 10 trajo el modo oscuro, controles de privacidad más avanzados y notificaciones más inteligentes.
16. **Android 11 (API nivel 30):** Esta versión incluyó mejoras en la gestión de conversaciones, burbujas de chat y un mejor control sobre los permisos.
17. **Android 12 (API nivel 31 y 32):** Android 12 se centra en la personalización y la privacidad del usuario, además de introducir nuevas funciones de diseño y un mejor rendimiento.
18. **Android 13 (API nivel 33).** Mejora la granularidad de los permisos para multimedia y las notificaciones. Avanza en el sistema de diseño de Android 12 o define un nuevo sistema para el portapapeles entre otras novedades.
19. **Android 14 (API nivel 34).** Su principal novedad es introducir funciones relacionadas con la IA, mejora la seguridad y la gestión de aplicaciones en segundo plano.



2.4.2. Herramientas de desarrollo.

La principal herramienta de desarrollo para el sistema Android es Android Studio que integra muchas otras como emuladores o compiladores. Se basa en el IDE IntelliJ, y entre sus características destacan:

1. **Interfaz de Usuario Intuitiva:** Android Studio ofrece una interfaz de usuario intuitiva y organizada que facilita la navegación y el desarrollo de aplicaciones.
2. **Emulador de Dispositivos:** Incluye un emulador de dispositivos Android que permite probar aplicaciones en una variedad de configuraciones de dispositivos, incluyendo diferentes versiones de Android y tamaños de pantalla.
3. **Editor de Código Avanzado:** Android Studio cuenta con un potente editor de código que ofrece funciones de autocompletado, resaltado de sintaxis, análisis estático y refactorización de código.

4. **Integración de Kotlin:** Es compatible de forma nativa con el lenguaje de programación Kotlin, que es el lenguaje recomendado por Google para el desarrollo de aplicaciones Android.
5. **Administración de Dependencias:** Android Studio facilita la gestión de dependencias de terceros a través de **Gradle**, un sistema de construcción que permite agregar y actualizar bibliotecas y paquetes de forma sencilla.
6. **Diseñador de Interfaces Gráficas:** Incluye un diseñador de interfaces gráficas (Layout Editor) que permite crear y editar interfaces de usuario de forma visual, lo que facilita la creación de diseños atractivos y funcionales.
7. **Conversión Automática de Recursos:** Android Studio puede convertir automáticamente recursos gráficos, como imágenes, para adaptarlos a diferentes resoluciones y tamaños de pantalla.
8. **Inspección de Recursos:** Permite inspeccionar y editar recursos de la aplicación, como layouts, drawables y valores, directamente desde el IDE.
9. **Herramientas de Depuración Avanzadas:** Android Studio incluye un depurador con funciones avanzadas, como puntos de interrupción, inspección de variables y seguimiento de la ejecución del código.
10. **Perfiles y Análisis:** Ofrece herramientas de análisis de rendimiento y perfiles que permiten identificar cuellos de botella y optimizar el rendimiento de la aplicación.
11. **Integración con Google Cloud y Firebase:** Android Studio se integra con servicios de Google Cloud y Firebase para facilitar el desarrollo de aplicaciones en la nube y la implementación de características como autenticación, notificaciones y análisis.
12. **Plantillas de Proyectos:** Proporciona plantillas de proyectos predefinidas que ayudan a comenzar rápidamente con diferentes tipos de aplicaciones, como aplicaciones en blanco, aplicaciones con pestañas, aplicaciones de navegación y más.
13. **Soporte Multilingüe:** Facilita la creación de aplicaciones multilingües al proporcionar herramientas para traducir y localizar la interfaz de usuario.
14. **Integración con Google Play:** Permite cargar y publicar aplicaciones directamente en Google Play Store.