

# Android

*FUNDAMENTOS*

# Formador



**David Jardón**  
Formador



Tengo más de **13 años de experiencia** en el desarrollo de aplicaciones móviles nativas en **Android** e **iOS**.

He colaborado en múltiples proyectos con: **BBVA**, **Santander**, **Movistar**, **Openbank**, **Bankinter**, **Mutua Madrileña**, **Ilunion**, **Adif**, **Inditex** y **Didomi**.

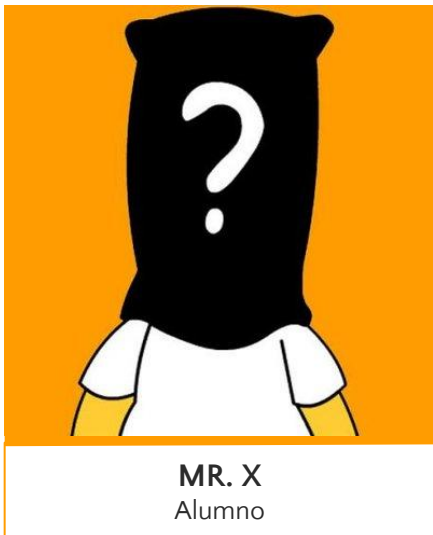
Co-Founder y CEO de Dust Summit, gestiono y desarrollo proyectos de software especializado en tecnologías mobile e imparto formaciones.

# Introducción

- **25 horas** de formación en **Android e iOS**.
- Horario de **09:30 a 15:00** con (30 minutos de descansos).
- Participar durante toda la formación es muy recomendado.
- Usaremos **Android Studio y Xcode** para programar y realizaremos pruebas con el simulador
- En cualquier momento durante la formación se podrán preguntar dudas, pidiendo antes **turno de palabra**

¡Muy importante participar activamente durante toda la formación!

# Es tu oportunidad



Cuando sea tu turno:

1. Dí tu **nombre**
2. Cuenta tu **experiencia** previa en desarrollo de aplicaciones móviles
3. Detalla tus **conocimientos** previos en programación

# Somos Developers







*“¡Developers! ¡Developers! ¡Developers!...”*



*Steve Ballmer ex-Microsoft CEO*



# Objetivos

1.  **Comprender los fundamentos del desarrollo en Android:** Conocer la historia y evolución de Android, su impacto global y sus ventajas en seguridad. Familiarizarse con las herramientas de desarrollo como Android Studio e IntelliJ.
2.  **Dominar los conceptos básicos de programación en Kotlin:** Aprender la sintaxis y principales características de Kotlin. Comprender uso en desarrollo de aplicaciones Android.
3.  **Construir interfaces de usuario con XML y Compose:** Diseñar pantallas con XML, utilizando Activities y Fragments. Implementar interfaces declarativas con Compose.
4.  **Publicar una aplicación en la PlayStore:** Gestionar dependencias con Gradle y Manifest. Configurar firma de una aplicación para su distribución.



# Hitos

1. Introducción a desarrollo con Android
2. Entornos de desarrollo
3. Programación con Kotlin
4. Componentes aplicación Android



# Hitos

5. Diseño de interfaces en XML
6. Diseño de interfaces en Compose
7. Dependencias con Gradle
8. Gestión de Certificados y publicación



Android

# Introducción al desarrollo

# ¿Por qué Android?

## ⚙️ Principales características:

- A diferencia de iOS (limitado a los dispositivos de Apple), Android es utilizado por Samsung, Xiaomi, OnePlus, Motorola, Oppo, Vivo, entre otros.
- Esto ha permitido que haya dispositivos Android en todas las gamas de precio.
- Con Android Open Source Project (AOSP), cualquier fabricante puede modificar y personalizar Android, lo que ha permitido una gran diversidad de dispositivos en el mercado.
- Android es más flexible y personalizable, pero sufre de fragmentación.

# Conocimientos previos

## Glosario

- **Android Studio:** El IDE oficial para desarrollar aplicaciones Android. Incluye herramientas para diseño, depuración, simulación y pruebas de apps.
- **Gradle:** Sistema de automatización de compilación utilizado en Android. Permite gestionar dependencias y configurar variantes de compilación.
- **ADB:** Herramienta de línea de comandos para interactuar con dispositivos Android conectados. Se usa para instalar apps, depurar y acceder al shell del sistema.

# Conocimientos previos

## Glosario

- **Java:** Lenguaje tradicional para el desarrollo de Android, usado desde el lanzamiento de la plataforma en 2008. Basado en JVM (Java Virtual Machine), permitiendo ejecución en múltiples dispositivos.
- **Kotlin:** Lenguaje moderno y oficial para Android desde 2017. Más conciso, seguro y fácil de leer en comparación con Java. Compatible con código Java, lo que permite migraciones progresivas.
- **Kotlin Multiplatform (KMP):** Tecnología que permite escribir código en Kotlin y compartirlo entre diferentes plataformas como Android, iOS, desktop, web y backend.

# Conocimientos previos

## Glosario

- **Activity:** *Componente de una app que representa una pantalla con una interfaz de usuario.*
- **Fragment:** *Una parte reutilizable de la UI dentro de una Activity. Permite crear interfaces modulares y flexibles.*
- **Intent:** *Mecanismo para iniciar actividades o comunicarse entre aplicaciones. Puede ser explícito (entre componentes de la misma app) o implícito (para abrir apps externas).*
- **ViewModel:** *Componente de Jetpack que almacena y gestiona datos relacionados con la UI de forma segura frente a cambios de configuración.*

# Conocimientos previos

## Glosario

- ***XML Layouts:*** Archivos XML donde se define la estructura visual de las pantallas de la app.
- ***Jetpack Compose:*** Nuevo framework declarativo para diseñar interfaces en Android de forma más intuitiva y moderna. Similar a SwiftUI en iOS.
- ***Material Design:*** Conjunto de directrices de Google para el diseño visual de apps Android.
- ***Firebase:*** Plataforma de Google para agregar backend en tiempo real, autenticación y notificaciones push.

# Conocimientos previos

## Glosario

- **Manifest:** Archivo de configuración donde se declaran componentes, permisos y configuraciones de la app.
- **APK / AAB:** Los **APK** son el formato tradicional de paquete de aplicaciones Android. Los **AAB (Android App Bundle)** son el nuevo formato optimizado para distribución en Google Play.
- **Google Play Console:** Plataforma para subir y gestionar apps en Google Play.

Android Studio

# Entorno de desarrollo



# Android Studio

🔧 El IDE oficial de Android.

¿Qué es?

- Android Studio es el entorno de desarrollo integrado (IDE) oficial de Android para crear aplicaciones en Android y otros dispositivos.
- Incluye herramientas para escribir código, diseñar interfaces, depurar y probar aplicaciones.



<https://developer.android.com/studio>

# Android Studio

## ⚙️ Principales características:

- **Editor de código avanzado:** Soporte para Kotlin y Java, con autocompletado inteligente.
- **Emulador:** Simulación de dispositivos con diferentes versiones de Android. Pruebas en múltiples tamaños de pantalla y configuraciones de hardware.
- **Layout Editor (Diseñador de Interfaces Visuales):** Permite diseñar interfaces en XML de forma visual. Soporte para ConstraintLayout, Material Design y Jetpack Compose.
- **Integración con Git:** Permite gestionar el control de versiones desde el IDE.

# Android Studio

## ⚙️ Principales características:

- **Depuración y Logcat:** Depurador (Debugger) para inspeccionar variables y analizar la ejecución. Logcat para ver registros del sistema y detectar errores.
- **Gestión de Dependencias con Gradle:** Facilita la instalación y actualización de librerías externas. Soporte para Maven y JCenter.
- **Compatibilidad con Jetpack y Firebase:** Facilita la integración de Jetpack (ViewModel, Room, WorkManager). Configuración rápida para Firebase (Auth, Firestore, Analytics).

Desarrollo

# Kotlin

# Kotlin

## El Lenguaje de Programación de Android.

- Lenguaje de programación moderno creado por **Jetbrains** en **2011**. Declarado lenguaje oficial para **Android** por **Google** en **2017**.
- Diseñado para ser **seguro**, **conciso** y **compatible** con **Java**.
- **Seguro**, manejo de memoria automático y evita errores comunes.
- **Multiplataforma**, se puede usar en Android, Backend (Ktor), iOS (KMP) y Web (Kotlin/JS).



<https://www.kotlinlang.org>

# Kotlin

## ⚙️ Características principales:

- 🚀 **Sintaxis concisa** → Menos código en comparación con Java.
- 🛡️ **Seguridad contra `NullPointerException`** → Soporte nativo para valores nulos.
- ↺️ **Compatibilidad con Java** → Kotlin y Java pueden coexistir en el mismo proyecto.
- 🔗 **Funciones de extensión** → Permite añadir funcionalidades a clases sin modificarlas.
- ⚡ **Programación funcional** → Soporta lambdas, inmutabilidad y funciones de orden superior.
- 📦 **Multiplataforma (KMP)** → Compartir código entre Android, iOS, web y backend.

# Kotlin

## Kotlin en Diferentes Plataformas

- **Kotlin para Android** → Desarrollo nativo con Jetpack y Compose.
- **Kotlin Multiplatform (KMP)** → Código compartido entre Android, iOS, Desktop y Web.
- **Kotlin para Servidores** → Desarrollo backend con Ktor y Spring Boot.
- **Kotlin/JS** → Desarrollo frontend con Kotlin y React.

***Conclusión:** Kotlin es un lenguaje versátil que se adapta a múltiples plataformas, permitiendo escribir código reutilizable y eficiente.*

Componentes

# XML Layout



# Componentes XML Layout

## ¿Qué es XML en Android?

- XML (Extensible Markup Language) se usa en Android para definir interfaces de usuario de forma estructurada y legible.
- Se utiliza para diseñar layouts, definir estilos, animaciones y configurar recursos.
- Se encuentra en la carpeta res/layout/ dentro de un proyecto Android.

# Componentes XML Layout

- ◆ Principales layouts de la interfaz de usuario.

- **LinearLayout:** Organiza elementos en fila (horizontal) o columna (vertical).
- **RelativeLayout:** Posiciona elementos relativos entre sí. (*Obsoleto, reemplazado por ConstraintLayout*)
- **ConstraintLayout:** Permite crear diseños flexibles con restricciones entre elementos. Reduce la anidación de vistas, mejorando el rendimiento.
- **FrameLayout:** Útil para superponer elementos (por ejemplo, mostrar una imagen sobre un botón).



# Componentes XML Layout

- ◆ Principales layouts de la interfaz de usuario.
  - **TextView:** Muestra texto en la pantalla.
  - **EditText:** Permite ingresar texto desde el teclado.
  - **Button:** Componente interactivo para acciones del usuario.
  - **ImageView:** Muestra imágenes desde recursos o URLs.

# Componentes XML Layout

## Propiedades de vistas:

- `match_parent` → Ocupa todo el espacio disponible del padre.
- `wrap_content` → Se ajusta al contenido del elemento.
- `dp (density-independent pixels)` → Medida recomendada para tamaños.
- `sp (scale-independent pixels)` → Se usa para tamaños de texto.

Componentes

# Jetpack Compose

# Componentes Jetpack Compose

## 📌 ¿Qué es Jetpack Compose?

- Framework **declarativo**, **flexible** y **eficiente** de UI creado por Google para Android.
- Alternativa a XML, permitiendo escribir **interfaces en Kotlin** puro.
- **Reactivo**, la UI se actualiza automáticamente cuando cambian los datos.
- **Multiplataforma**, compatible con Android, iOS, Desktop y Web (Compose Multiplatform).



<https://developer.android.com/compose>

# Componentes Jetpack Compose

## ⚙ Características principales

- 🚀 **Sintaxis más simple y menos código** → Diseñar UI sin necesidad de XML.
- ↺ **Enfoque declarativo** → Define "qué" debe mostrarse, en lugar de "cómo".
- 📦 **Composables reutilizables** → Los componentes pueden anidarse y reutilizarse fácilmente.
- ⚡ **Integración con ViewModel y Flow** → Soporte nativo para estados dinámicos.
- 🌐 **Compatibilidad con Material Design 3** → Implementación moderna y adaptable.
- 🎨 **Soporte para animaciones avanzadas** → Transiciones fluidas y personalizadas.



# Componentes Jetpack Compose

- ◆ Principales componentes de la interfaz de usuario.
  - **Column:** Organiza elementos en columna (vertical).
  - **Row:** Organiza elementos en fila (horizontal).
  - **Box:** Útil para superponer elementos (por ejemplo, mostrar una imagen sobre un botón).
  - **LazyColumn:** Lista horizontal optimizando su rendimiento para el procesamiento de datos.
  - **LazyRow:** Lista horizontal optimizando su rendimiento para el procesamiento de datos.





# Componentes Jetpack Compose

- ◆ Principales componentes de la interfaz de usuario.
  - **Text:** Muestra texto en pantalla con personalización de fuente y color.
  - **Button:** Elemento interactivo para recibir acciones del usuario.
  - **Image:** Carga imágenes desde recursos o URLs.

Gestión dependencias

# Gradle

# Gestión de dependencias Gradle

## ¿Qué es Gradle?

- Gradle es el sistema de compilación utilizado en Android para gestionar dependencias y automatizar tareas de construcción.
- Se basa en un sistema declarativo con archivos de configuración en Kotlin DSL o Groovy.

## Repositorios de dependencias:

- Maven Central
- Google Maven Repository (para Android Jetpack, Play Services, etc.)
- JitPack (para bibliotecas de código abierto en GitHub)

# Gestión de dependencias Gradle

## Archivos de configuración principales

- **settings.gradle(.kts):** Define los módulos del proyecto y los repositorios globales.
- **build.gradle(.kts) (*nivel de proyecto*):** Configura plugins, repositorios y configuración general del proyecto.
- **build.gradle(.kts) (*nivel de módulo*):** Define las dependencias específicas del módulo y las configuraciones de compilación.

# Gestión de dependencias Gradle



## Agregar dependencias a un módulo

- Las dependencias se añaden en el archivo **build.gradle(.kts)** a nivel de módulo dentro de la sección dependencies.



## Dependencias en versiones centralizadas

- Para evitar redundancias, se recomienda centralizar versiones en *gradle/libs.versions.toml*



## Sincronización y actualización de dependencias

- Sincronizar Gradle en ***File > Sync Project with Gradle Files***

Gestión permisos

# Android Manifest

# Gestión de permisos AndroidManifest

## ¿Qué se define en el AndroidManifest?

- Nombre del paquete de la aplicación.
- Componentes principales: Activities, Services, Broadcast Receivers y Content Providers.
- Permisos necesarios para el funcionamiento de la app.
- Configuraciones de hardware y software requeridas.

# Gestión de permisos AndroidManifest

## Permisos comunes:

- **Internet:** Para acceder a la red *android.permission.INTERNET*
- **Ubicación:** Para la ubicación del usuario *android.permission.ACCESS\_FINE\_LOCATION*
- **Cámara:** Para tomar fotos o grabar videos. *android.permission.CAMERA*
- **Almacenamiento:** Para leer/escribir en el almacenamiento del dispositivo *android.permission.READ\_EXTERNAL\_STORAGE*



 Desde Android 6.0 (API 23), los permisos sensibles requieren solicitud en tiempo de ejecución con ***requestPermissions()*** en el código 



# Gestión de permisos AndroidManifest

## Importancia del AndroidManifest.xml

- Define la estructura de la aplicación, asegurando que Android pueda ejecutarla.
- Gestiona permisos y seguridad, evitando accesos no autorizados a funciones críticas.
- Configuraciones de compatibilidad, versiones mínimas de SDK y requerimientos de hardware.
- Permite interacción entre aplicaciones con intent-filters.

 *El **AndroidManifest.xml** es clave para el **funcionamiento** y **seguridad** de cualquier aplicación Android. ¡Comprender su estructura y configuraciones es esencial para un desarrollo eficiente!* 

PlayStore

# Gestión de publicación

# Gestión publicación

## Publicación en Google Play Store

- Crear una cuenta de desarrollador en Google Play Console.
- **Configurar la aplicación:**
  - Ingresar información básica (nombre, descripción, categoría, íconos y capturas de pantalla).
  - Definir políticas de contenido y privacidad



<https://play.google.com/console/signup>

# Gestión publicación

## Firmado de la aplicación

- Google exige que todas las aplicaciones estén firmadas digitalmente antes de su distribución.
- **Métodos de firma:**
  - Firma con una **clave propia**: Se genera un keystore y se configura en `build.gradle.kts`.
  - Firma con **Google Play App Signing**: Google gestiona las claves de firma por seguridad.

# Gestión publicación

## 17 Generación del APK/AAB

- Google Play recomienda distribuir aplicaciones en **formato AAB** en lugar de APK.

## Pruebas y revisión

- Subir la aplicación a la Play Console y completar la revisión.
- Pruebas internas y cerradas antes del lanzamiento público.
- Aprobación de Google tras revisar la aplicación según sus políticas.

## Lanzamiento en producción

- Una vez aprobada, se puede lanzar la aplicación y configurar actualizaciones futuras.

# Gestión publicación



## Cumplimiento de Políticas

- Revisar las Políticas de Google Play antes de enviar la aplicación.
- No incluir contenido prohibido (malware, contenido violento, engañoso o ilegal).
- Asegurar que la app cumpla con las regulaciones de privacidad y uso de datos.

<https://play.google/intl/es/developer-content-policy/>

# Gestión publicación



## Permisos y Seguridad

- Evitar solicitar permisos innecesarios, ya que pueden generar rechazo.
- Proporcionar un aviso claro y justificado del uso de datos personales.
- Cumplir con los requisitos de la Sección de Seguridad de Datos en Google Play Console.

<https://developer.android.com/training/permissions/requesting-special>

# Gestión publicación

## Calidad y Experiencia de Usuario

- Asegurarse de que la app esté completamente funcional antes del envío.
- Optimizar el rendimiento y la estabilidad de la aplicación.
- Probar la app en diferentes dispositivos y resoluciones.



# Gestión publicación

## **Transparencia y Metadatos**

- Proveer una descripción precisa y clara de la aplicación.
- Usar capturas de pantalla y videos representativos de la experiencia real de la app.
- No incluir referencias engañosas o irreales en la descripción y gráficos promocionales.

## **Tiempos de Revisión**

- La revisión puede tardar desde unas horas hasta varios días.
- Google puede requerir información adicional o cambios antes de la aprobación.

# ¡Muchas gracias!



**David Jardón**  
Formador



Hemos concluido este módulo de la formación, muchas gracias por tu participación.

- Aprovecha para preguntar cualquier duda
- Recuerda guardar mi contacto en **LinkedIn** para futuras consultas.