





Kotlin Multiplatform

KMP

Objetivos

1.  **Comprender qué es Kotlin Multiplatform y sus beneficios:** Comparar KMP con otras alternativas y explicar cuándo es más adecuado usarlo. Conocer las herramientas y plugins necesarios para facilitar el desarrollo.
2.  **Conocer los componentes clave de Kotlin Multiplatform:** Explicar los elementos fundamentales que permiten compartir código, como las funciones *expect* y *actual*.
3.  **Desarrollar aplicaciones móviles para Android e iOS utilizando KMP:** Implementar funcionalidades en ambas plataformas utilizando código compartido.
4.  **Utilizar recursos compartidos y bibliotecas multiplataforma:** Conocer las bibliotecas más comunes y cómo integrarlas en un proyecto KMP.



Hitos

1. ¿Qué es Kotlin Multiplatform?
2. Alternativas a Kotlin Multiplatform
3. Entorno de desarrollo: Android Studio
4. Estructura del proyecto
5. Componentes de Kotlin Multiplatform

KMP

¿Qué es Kotlin Multiplatform?

¿Qué es Kotlin Multiplatform?

⚙️ Principales características:

- También conocido como **KMP**, es un enfoque para compartir lógica de negocio común en aplicaciones móviles, web y de escritorio utilizando **Kotlin**.
- Permite escribir código compartido (como la lógica de negocio, los modelos de datos y la capa de red) en un solo lugar, evitando la duplicación de código.
- Se puede desarrollar interfaz común con **Compose** en cualquier plataforma móvil o web
- Genera **código nativo** para Android y iOS, aprovechando lo mejor de ambas plataformas.

<https://kotlinlang.org/docs/multiplatform.html>

¿Qué es Kotlin Multiplatform?

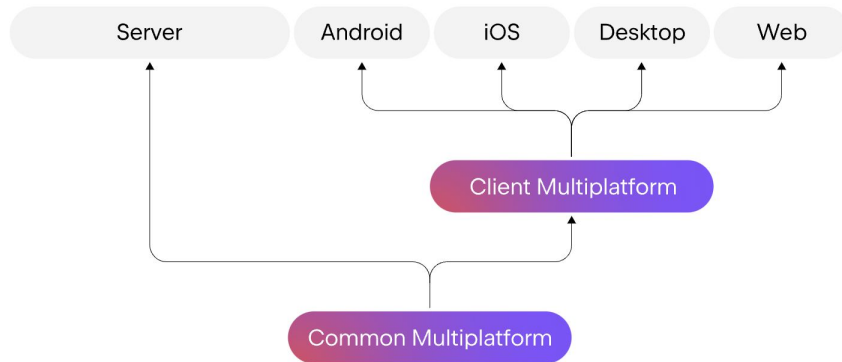
Ventajas de KMP

- ***Reutilización del código***: Compartir código entre plataformas reduce el esfuerzo de desarrollo y mantenimiento.
- ***Interoperabilidad con el código nativo***: El código de la plataforma específica sigue escribiéndose en Swift para iOS y en Kotlin para Android.
- ***Menos duplicación de lógica de negocio***: La lógica de negocio, las validaciones, gestión de recursos y el manejo de errores pueden centralizarse en una capa común.

¿Qué es Kotlin Multiplatform?

🚀 Ventajas de KMP

- **Unificar Tests:** Permite unificar todo el proceso de testing de la lógica de negocio en un único punto siendo más eficiente, rápido y fácil de mantener para todas las plataformas.

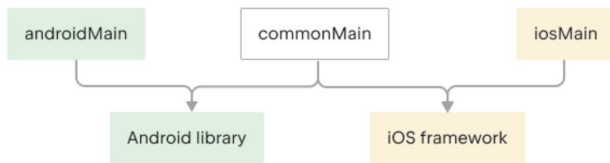


¿Qué es Kotlin Multiplatform?

🔧 Cómo funciona KMP

Los proyectos de KMP se dividen en dos bloques principales:

- **Módulo común:** Donde se ubica el código que se comparte.
- **Módulos específicos de la plataforma:** Donde se implementan las funciones específicas de cada plataforma.



¿Qué es Kotlin Multiplatform?

Ecosistema y librerías soportadas

Algunas de las librerías más importantes de Kotlin son también soportadas en KMP:

- *Kotlinx.serialization* para la serialización de datos.
- *Ktor* para hacer solicitudes de red en ambas plataformas.
- *ROOM*, *SQLDelight* o *Realm* para el manejo de bases de datos.
- *Flow* y *Coroutines* para programación asíncrona.
- Inyección de dependencias con *Koin*.

KMP

Alternativas




Alternativas a KMP

⚡ Existen varias alternativas a Kotlin Multiplatform Mobile (*KMP*) para el desarrollo de aplicaciones móviles multiplataforma.

- Ninguna utiliza un lenguaje común al desarrollo nativo como lo es *Kotlin*.
- *KMP* permite compartir la lógica de negocio e interfaz, pero las interfaces de usuario se pueden desarrollar de forma nativa en cada plataforma.
- Normalmente, no necesita desarrollos nativos para implementar las funcionalidades comunes.

Alternativas a KMP




KMP

-  **Lenguaje:** Kotlin.
-  **Ventajas:**
 - Permite compartir la lógica de negocio y código común entre iOS y Android.
 - Aprovecha la interoperabilidad total con el código nativo de cada plataforma.
 - La adopción de Kotlin en Android es alta, lo que facilita su integración en proyectos.
-  **Desventajas:**
 - El ecosistema es más nuevo en comparación con otras.
 - Algunas integraciones pueden requerir ajustes adicionales.



Alternativas a KMP




Flutter

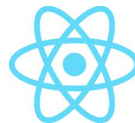
-  **Lenguaje:** Dart.
-  **Ventajas:**
 - Usa un único lenguaje para la interfaz de usuario y la lógica de negocio.
 - Amplia comunidad y ecosistema de plugins.
 - Hot reload para facilitar el desarrollo.
-  **Desventajas:**
 - El rendimiento puede no ser comparable a aplicaciones nativas.
 - La curva de aprendizaje de Dart puede ser un obstáculo.



Alternativas a KMP




React Native

-  **Lenguaje:** JavaScript o TypeScript.
-  **Ventajas:**
 - Facilita la integración con aplicaciones existentes.
 - Gran soporte de la comunidad y bibliotecas populares.
-  **Desventajas:**
 - El rendimiento depende del uso del puente JavaScript-Nativo.
 - La gestión de dependencias nativas puede ser compleja.
 - Necesarios desarrollos nativos.



Alternativas a KMP




Xamarin

-  **Lenguaje:** C#.
-  **Ventajas:**
 - Permite compartir hasta el 90% del código entre plataformas.
 - Gran integración con el ecosistema de Microsoft.
-  **Desventajas:**
 - El tamaño de la aplicación es más grande en comparación con otras soluciones.
 - La comunidad no es tan activa como la de Flutter o React Native.



Alternativas a KMP




Ionic

-  **Lenguaje:** JavaScript/TypeScript (utilizando Angular, React, o Vue).
-  **Ventajas:**
 - Desarrollo rápido y fácil para desarrolladores web.
 - Buen soporte para Progressive Web Apps (PWAs).
-  **Desventajas:**
 - La interfaz de usuario no se siente nativa.
 - El rendimiento puede ser inferior al de aplicaciones nativas o compiladas.



Alternativas a KMP

Unity

-  **Lenguaje:** C#.
-  **Ventajas:**
 - Popular para el desarrollo de juegos móviles.
 - Buena compatibilidad multiplataforma (iOS, Android, Windows).
-  **Desventajas:**
 - No es ideal para aplicaciones que no sean juegos.
 - Interfaz y lógica de negocio no son flexibles para aplicaciones tradicionales.



Android Studio

Entorno de desarrollo

Entorno de desarrollo

Android Studio

Configurar el entorno de desarrollo en IntelliJ para trabajar con Kotlin Multiplatform Mobile (KMM):

- La instalación de los **plugins** necesarios
- La **configuración** del proyecto
- La comprensión de las **herramientas** que facilitan el desarrollo

Entorno de desarrollo

IntelliJ IDEA

- Se recomienda la versión **Ultimate**, ya que tiene soporte completo para desarrollo móvil y multiplataforma.
- La versión **Android Studio** suele ser la más utilizada, pero algunas características avanzadas podrían no estar disponibles.

<https://www.jetbrains.com/es-es/idea/download/?section=mac>

Entorno de desarrollo

Instalación de plugins necesarios:

- **Kotlin:** Viene preinstalado en la mayoría de las versiones de IntelliJ, pero asegúrate de que esté actualizado.
- **Kotlin Multiplatform Mobile (KMM):** Android Studio facilita la configuración inicial.
- **Android Studio:** Necesario configurar el SDK y el emulador de Android.
- **Xcode (solo macOS):** Para desarrollo en iOS, se necesita tener Xcode instalado.

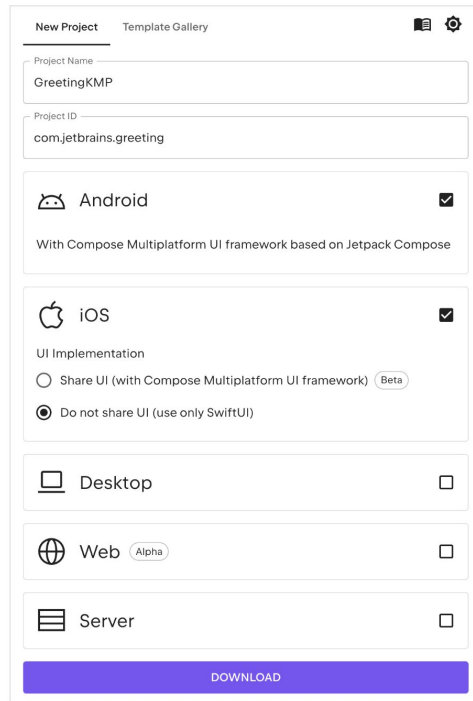
<https://www.jetbrains.com/help/kotlin-multiplatform-dev/multiplatform-setup.html>

Entorno de desarrollo

Configuración del proyecto KMM

- Seleccionar la plantilla de proyecto KMP que ofrece el IDE.
- Esto configura un proyecto con módulos comunes, Android e iOS.

<https://kmp.jetbrains.com/>





The screenshot shows the 'New Project' dialog with the 'Template Gallery' tab selected. The 'Project Name' is 'GreetingKMP' and the 'Project ID' is 'com.jetbrains.greeting'. Under 'Platform', 'Android' and 'iOS' are selected with checkboxes. The 'UI Implementation' section shows 'Do not share UI (use only SwiftUI)' selected. Other options like 'Desktop', 'Web', and 'Server' are unchecked. A 'DOWNLOAD' button is at the bottom.


New Project Template Gallery


Project Name
GreetingKMP


Project ID
com.jetbrains.greeting

 Android ☒
With Compose Multiplatform UI framework based on Jetpack Compose

 iOS ☒
UI Implementation
☐ Share UI (with Compose Multiplatform UI framework) Beta
☒ Do not share UI (use only SwiftUI)

 Desktop ☐

 Web Alpha ☐

 Server ☐

DOWNLOAD

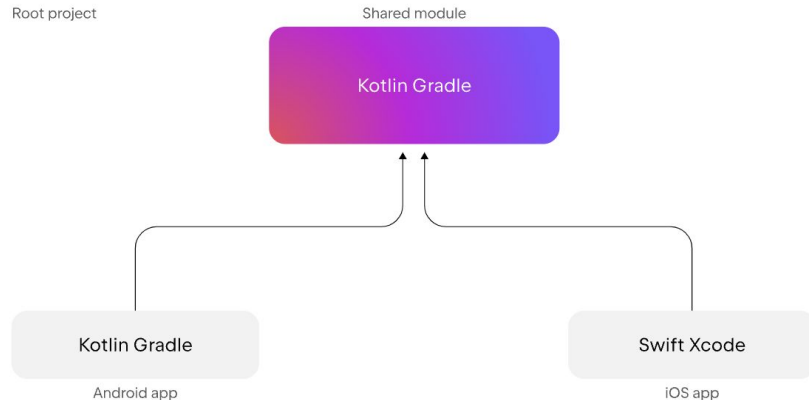
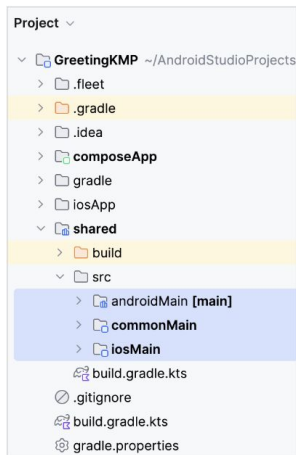
Entorno de desarrollo

Estructura del proyecto inicial:

- Módulo compartido (***shared***): Aquí es donde se colocará el código común. Este módulo será un módulo de Kotlin con el código compartido entre Android e iOS.
- Módulo de ***Android***: Un módulo específico de Android que puede incluir código Java/Kotlin nativo.
- Módulo de ***iOS***: Generalmente configurado para integrarse con un proyecto Xcode existente.

Entorno de desarrollo

📁 Estructura del proyecto inicial:



KMP

Conceptos básicos

Conceptos básicos KMP

👁️ Funcionamiento de *actual* y *expect* en KMP:

- ***expect***: Define una interfaz o contrato en el código compartido. Por ejemplo, una clase o función que las plataformas específicas deben implementar.
- ***actual***: Proporciona la implementación específica de una plataforma para la declaración *expect*.

```
// Shared Code
expect fun getPlatformName(): String

// Android Implementation
actual fun getPlatformName(): String = "Android"

// iOS Implementation
actual fun getPlatformName(): String = "iOS"
```

Conceptos básicos KMP

Uso de Jetpack Compose y SwiftUI con KMP

- Jetpack Compose es el framework de UI declarativo de Android basado en Kotlin.
- SwiftUI es el equivalente en iOS, ofreciendo una forma declarativa y moderna de construir interfaces.
- Con KMP, se puede compartir la lógica de presentación mientras se mantienen interfaces nativas en cada plataforma o compartir también la interfaz.

Conceptos básicos KMP

⚡ Optimización del rendimiento

- Evita llamadas innecesarias entre plataformas: Reduce el uso de expect/actual cuando no sea estrictamente necesario.
- Usa Flow para manejar eventos y datos de manera eficiente.
- Minimiza el tamaño del binario: Evita dependencias pesadas en la capa compartida y optimiza el código nativo cuando sea posible.

Conceptos básicos KMP

¿Por qué usar Compose Multiplatform?

- **Reducción de código duplicado:** Compartir el código de la interfaz entre plataformas, ahorra tiempo y esfuerzo en el desarrollo de aplicaciones.
- **Consistencia** de UI: Al compartir la misma lógica de UI, se garantiza una experiencia consistente entre plataformas.
- **Productividad:** Compose **permite crear interfaces con menos código, lo que acelera el desarrollo.**

Conceptos básicos KMP

Consideraciones y buenas prácticas

- **Modularidad:** Compartir la mayor cantidad posible de lógica entre las plataformas, pero las características específicas de cada plataforma, como acceso a hardware o APIs nativas, deben ser implementadas en módulos separados.
- **Composición de interfaces complejas:** Compose permite una interfaz declarativa, cuando comparten componentes complejos entre plataformas, es importante realizar pruebas exhaustivas en cada plataforma.
- **Estrategias de compatibilidad:** Algunos controles de UI específicos pueden no estar completamente disponibles en todas las plataformas actualmente.

Conceptos básicos KMP

Ventajas de Compose Multiplatform con KMP

- **Desarrollo simultáneo:** Se pueden desarrollar interfaces de usuario para múltiples plataformas en paralelo, compartiendo la mayoría del código.
- **Consistencia visual:** Al usar el mismo código de UI en todas las plataformas, los usuarios tendrán una experiencia coherente y customizable.
- **Futuro prometedor:** Con el crecimiento de Compose, su soporte multiplataforma promete convertirse en una de las soluciones más robustas y escalables para el desarrollo de apps.

¡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.