iOS

FUNDAMENTOS

Objetivos

1. 📱 **Comprender los fundamentos del desarrollo en iOS:** Conocer la historia y evolución de iOS, su impacto global y sus ventajas en seguridad. Familiarizarse con las herramientas de desarrollo como Xcode y Playground.

2. 💻 **Dominar los conceptos básicos de programación en Swift:** Aprender la sintaxis y principales características de Swift. Comprender su uso en el desarrollo de aplicaciones iOS.

3. 🎨 **Construir interfaces de usuario con UIKit y SwiftUI:** Diseñar pantallas con UIKit, utilizando ViewControllers y componentes nativos. Implementar interfaces declarativas con SwiftUI y gestionar estados dinámicos.

4. 🚀 **Publicar una aplicación en la App Store:** Gestionar dependencias con CocoaPods y Swift Package Manager. Configurar certificados, perfiles de desarrollo y firmar una aplicación para su distribución.

Hitos

1. Introducción a desarrollo con iOS
2. Entornos de desarrollo
3. Programación con Swift
4. Componentes aplicación iOS

Hitos

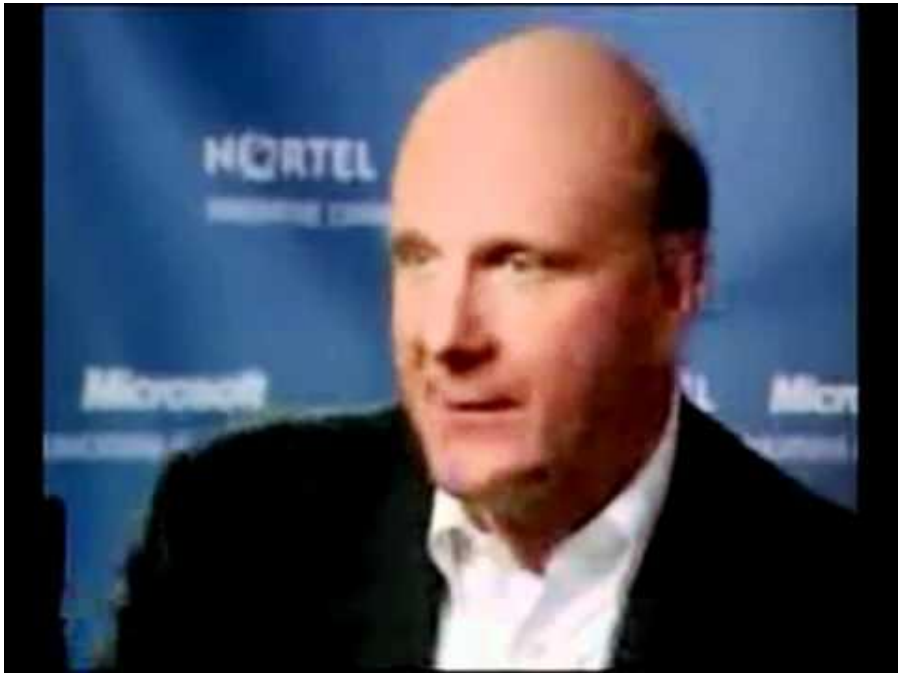
5. Diseño de interfaces en UIKit
6. Diseño de interfaces en SwiftUI
7. Dependencias con CocoaPods y Swift Package Manager
8. Gestión de Certificados, Profile y publicación

iOS

Introducción al desarrollo

¿Por qué iOS?

👀 *"¡500 dólares! ¡**El teléfono más caro del mundo**! Y no está enfocado a los empresarios porque no tiene teclado" Steve Ballmer ex-Microsoft CEO*



¿Por qué iOS?

🔒 **Ecosistema seguro y optimizado**

- Apple controla tanto hardware como software, permitiendo una integración eficiente.
- Mejor rendimiento y estabilidad en comparación con sistemas abiertos como Android.
- Apple revisa manualmente todas las apps antes de publicarlas en la App Store.
- Reduce la posibilidad de malware y aplicaciones maliciosas.

¿Por qué iOS?

🔒 **Ecosistema seguro y optimizado**

- iOS tiene una alta tasa de adopción de nuevas versiones, lo que reduce vulnerabilidades.
- Las apps solo pueden acceder a datos específicos si el usuario da permiso explícito.
- Procesador independiente que protege datos sensibles como Face ID y Touch ID.
- Almacén seguro de claves Keychain

Conocimientos previos

📌 **Glosario**

- ***Xcode:*** *Entorno de desarrollo integrado (IDE) oficial de Apple para crear aplicaciones en iOS, macOS, watchOS y tvOS. Incluye herramientas de diseño de interfaz, simuladores y depuración.*

- ***Playground:*** *Herramienta dentro de Xcode o uso online que permite probar código Swift en tiempo real sin necesidad de compilar una app completa. Ideal para aprender, experimentar y hacer pruebas rápidas.*

- ***Objective-C:*** *Lenguaje de programación utilizado históricamente en el desarrollo de aplicaciones iOS y macOS. Aunque ha sido reemplazado en gran parte por Swift, sigue presente en muchas aplicaciones y frameworks.*

Conocimientos previos

📌 **Glosario**

- ***Swift:*** *Lenguaje de programación moderno, seguro y optimizado para el desarrollo en el ecosistema Apple. Ofrece mejor rendimiento y seguridad que Objective-C.*

- ***UIKit:*** *Framework principal para construir interfaces de usuario en aplicaciones iOS. Basado en vistas y controladores, ha sido el estándar en desarrollo iOS durante muchos años.*

- ***SwiftUI:*** *Framework más reciente para construir interfaces declarativas en iOS, macOS, watchOS y tvOS. Facilita el desarrollo con menos código y una vista previa en tiempo real.*

Xcode

Entorno de desarrollo

Xcode

🛠️ **El IDE oficial de Apple.**

¿Qué es?

- Xcode es el entorno de desarrollo integrado (IDE) oficial de Apple para crear aplicaciones en iOS, macOS, watchOS, tvOS y visionOS.

- Incluye herramientas para escribir código, diseñar interfaces, depurar y probar aplicaciones.

https://developer.apple.com/xcode/

Xcode

⚙️ **Principales características:**

- **Editor de código avanzado:** Soporte para Swift y Objective-C, con sugerencias inteligentes y detección de errores en tiempo real.
- **Interface Builder:** Diseño de interfaces visuales sin código, usando UIKit o SwiftUI.
- **Simulador:** Emula iPhones, iPads y otros dispositivos Apple sin necesidad de hardware físico.
- **Depuración y profiling:** Herramientas como el Debugger y el Profiler para analizar el rendimiento de las apps.
- **Integración con Git:** Permite gestionar el control de versiones desde el IDE.

Swift Playground

🛠️ **Un Entorno de Pruebas para Swift.**

¿Qué es?

- Swift Playground es una herramienta dentro de Xcode y una app para iPad que permite ejecutar código Swift en tiempo real sin necesidad de compilar un proyecto completo.
- Ideal para aprender Swift, probar código rápidamente y experimentar con lógica de programación.

https://www.apple.com/es/swift/playgrounds/

Swift Playground

⚙️ **Principales características:**

- **Ejecución en tiempo real:** Permite ver los resultados del código de inmediato.
- **Soporte para gráficos e interacción:** Se pueden crear animaciones y pequeños juegos con Swift y SwiftUI.
- **Interfaz intuitiva:** Muy útil para principiantes y para prototipado rápido de código.
- **Disponible en iPad:** Se puede programar en Swift sin necesidad de un Mac.

Disponible online: https://swiftfiddle.com/

Desarrollo

Swift

Swift

📌 **El Lenguaje de Programación de Apple.**

- Lenguaje de programación moderno creado por **Apple** en **2014.**
- **Seguro y optimizado** para el desarrollo en iOS, macOS, watchOS, tvOS y visionOS.
- **Seguro,** manejo de memoria automático y evita errores comunes.
- De **código abierto,** Disponible en GitHub y compatible con otras plataformas.

https://www.swift.org/

Swift

⚙️ **Características principales:**

- 🚀 **Mayor productividad:** Código más conciso y legible
- 🛡️ **Menos errores:** Tipado seguro y manejo de memoria automático
- ⚡ **Mejor rendimiento:** Más rápido que Objective-C en la mayoría de los casos.
- 🌍 **Compatible con Objective-C:** Se puede integrar en proyectos existentes.
- 💻 **Multiplataforma:** Soporta desarrollo para Linux y servidores.

Componentes

UIKit

Componentes Aplicación iOS

- **AppDelegate.swift**
  - Punto de entrada de la aplicación.
  - Gestiona eventos globales, como el inicio y la terminación de la app.
  - Maneja notificaciones push y la configuración inicial.

UIKit

- **SceneDelegate.swift (a partir de iOS 13+)**
  - Gestiona múltiples escenas o ventanas de la aplicación.
  - Configura la vista inicial de la aplicación.
  - Se usa en apps que permiten multitarea, como en iPad.

UIKit

- **UIViewController.swift**

- Controlador de vista encargado de gestionar la UI y la lógica de una pantalla.
- Su ciclo de vida se resumen en 5 funciones principales:

Método	Descripción
viewDidLoad()	Se llama una vez cuando la vista se carga en memoria. Ideal para inicializar datos.
viewWillAppear()	Se llama antes de que la vista aparezca en pantalla.
viewDidAppear()	Se llama cuando la vista ya está en pantalla.
viewWillDisappear()	Se llama antes de que la vista desaparezca.
viewDidDisappear()	Se llama cuando la vista ya no es visible.

UIKit

- **Storyboard**

  - Archivo visual que define la interfaz de usuario con pantallas y transiciones.

  - **XIBs** (Interface Builder Files): Archivos individuales para diseñar interfaces reutilizables.

UIKit

- **Navigation Controller y Tab Bar Controller**

  - **UINavigationController:** Permite gestionar la navegación entre pantallas con una pila de vistas.

  - **UITabBarController:** Controlador de pestañas para cambiar entre diferentes secciones de la app.

UIKit

- **TableView y CollectionView**

  - **UITableView:** Se usa para mostrar listas con celdas reutilizables.

  - **UICollectionView:** Similar, pero con una estructura más flexible (grid, carrusel, etc.).

Componentes

SwiftUI

SwiftUI

- **SwiftUI** es el framework de interfaz de usuario declarado y moderno creado por Apple, lanzado en **2019** para facilitar el desarrollo de aplicaciones.
- **Declarativo:** La interfaz se describe por lo que debe hacer, simplifica comprensión del código.
- **Reactivo:** Los cambios en el estado de la aplicación se reflejan automáticamente en la interfaz.
- **Compatibilidad multiplataforma:** Se puede usar en macOS, watchOS y tvOS.
- **Menos código:** Más fácil de aprender que UIKit, menos líneas de código para lograr lo mismo.
- **Integración con Swift:** Se integra de forma nativa con las características de Swift, como las funciones y propiedades @State y @Binding.

SwiftUI

- **Views**

Componentes:

- **Text:** Muestra texto en la pantalla.
- **Image:** Muestra una imagen en la interfaz de usuario.
- **Button:** Un componente de botón interactivo.

Containers:

- **VStack:** Un contenedor vertical que apila sus vistas hijo en una columna.
- **HStack:** Similar a VStack, pero apila las vistas de manera horizontal.
- **ZStack:** Apila las vistas sobre otras, como capas, lo que es útil para superponer elementos.

SwiftUI

- **Views**

Modificadores:

- **Padding:** Añade espacio alrededor de una vista.
- **CornerRadius:** Redondea las esquinas de una vista.
- **Background:** Establece un fondo para una vista.

State y Binding:

- **@State** Usado para declarar variables que controlan el estado de una vista y que pueden cambiar dinámicamente.
- **@Binding** Permite que las vistas compartan datos y se actualicen automáticamente.

Gestión dependencias

CocoaPods

Gestión de dependencias CocoaPods

📂 **¿Qué es CocoaPods?**

- A menudo necesitamos usar bibliotecas y frameworks de terceros para añadir funcionalidades específicas.

- Para gestionar estas dependencias, existen varias herramientas, pero las dos más utilizadas son:
  - **CocoaPods** https://cocoapods.org/
  - Swift Package Manager (**SPM**) https://www.swift.org/documentation/package-manager/

Gestión de dependencias CocoaPods

🔧 Configuraciones principales

- Cuando se inicia CocoaPods con ***pod init*** se crea un archivo ***Podfile*** en el directorio raíz de tu proyecto, que es donde se declararán las dependencias.
- Dentro del fichero ***Podfile*** se añaden las bibliotecas que se quieren usar.
- Con ***pod install*** se descargarán e instalarán las dependencias.
- Esto generará un archivo ***.xcworkspace*** que será el que se debe abrir para trabajar con el proyecto.

Gestión de dependencias CocoaPods

🚀 **Ventajas de CocoaPods:**

- **Amplia comunidad y soporte**, ya que lleva años siendo la herramienta estándar para gestionar dependencias en proyectos iOS.
- Fácil de usar y configurar.
- Permite trabajar con **dependencias** escritas en **Objective-C y Swift**.

👀 **Desventajas de CocoaPods:**

- Puede ser un poco más lento, ya que necesita integrar y modificar el proyecto Xcode.
- No está tan integrado con el sistema de Xcode como Swift Package Manager.
- Es necesario utilizar el fichero .xcworkspace
- Requiere instalación por separado.

Gestión dependencias

Swift Package Manager

Swift Package Manager

📂 **¿Qué es Swift Package Manager?**

- Swift Package Manager (SPM) es una herramienta de gestión de dependencias desarrollada por Apple y ya integrada en Xcode.
- SPM es la alternativa nativa a CocoaPods y es cada vez más popular debido a su integración directa con el ecosistema de Swift.
- No requiere instalación por separado, ya que SPM viene integrado directamente con Xcode desde la versión 11.

Swift Package Manager

🔧 Configuraciones principales

- En Xcode en la barra de menú, se encuentra en ***File > Swift Packages > Add Package Dependency***.
- Se introduce la URL del repositorio del paquete que se desea agregar.
- Xcode pedirá se seleccione una versión del paquete (versión exacta o un rango de versiones).
- Después de añadir el paquete, se descargará e integrará automáticamente en el proyecto.
- Para actualizar las dependencias gestionadas por SPM, se puede hacer desde Xcode seleccionando ***File > Swift Packages > Update to Latest Package Versions***.

Swift Package Manager

🚀 **Ventajas de Swift Package Manager**

- **Integración nativa:** SPM está completamente integrado en Xcode y no requiere configuración adicional.
- **Más rápido y ligero:** No modifica el proyecto de Xcode de la misma manera que CocoaPods, lo que hace que la configuración sea más rápida y sencilla.
- **Compatible con Swift:** Esto es ideal si todas las dependencias del proyecto son escritas en Swift.

👀 **Desventajas de Swift Package Manager**

- Menos soporte para dependencias escritas en Objective-C.
- La comunidad de paquetes es más pequeña en comparación con CocoaPods.

AppStore

Gestión de publicación

Cuentas de Apple

🌐 **Apple Developer Account**

- Para poder distribuir aplicaciones en la App Store, es necesaria una cuenta de **Apple Developer.**

- Esta cuenta tiene un coste anual y proporciona acceso a herramientas como:
  - Certificados, Identifiers & Profiles.
  - App Store Connect para gestionar la app.
  - TestFlight para distribuir versiones beta de la aplicación.

https://developer.apple.com/

Cuentas de Apple

🌐 App Store Connect

Plataforma web que permite gestionar todos los aspectos de la distribución de la aplicación, como:

- Envío y gestión de apps.
- Análisis de ventas y descargas.
- Gestión de actualizaciones de la app.
- Acceso a TestFlight para distribuir pruebas beta a usuarios seleccionados.
- Monitoreo de revisiones de aplicaciones y manejo de feedback de los usuarios.

https://developer.apple.com/app-store-connect/

Gestión de dependencias

📊 La publicación de una aplicación en la App Store implica varios pasos clave en los que es esencial gestionar:

- **Certificados**
- Perfiles de aprovisionamiento (**Provisioning Profiles**)
- Configuraciones de distribución.

Además, se debe disponer de una cuenta de ***App Store Connect*** para gestionar la aplicación, sus metadatos, y su lanzamiento.

Certificados y Provisioning Profiles

🔑 Certificados (Certificates)

Apple requiere certificados para poder firmar la aplicación y garantizar que solo las aplicaciones autorizadas sean instaladas en dispositivos, debes crearlos en la cuenta de ***Apple Developer***.

Los certificados son esenciales para:

- **Desarrollo:** Permiten ejecutar la aplicación en dispositivos físicos mientras se desarrolla.
- **Distribución:** Permiten que la aplicación se firme de forma que pueda ser publicada en la App Store o distribuida a través de Enterprise Program.

Certificados y Provisioning Profiles

🔑 **Tipos de Certificados:**

- **Development**
  - **Certificado de Desarrollo:** Necesario para ejecutar la aplicación en un dispositivo propio o en otros dispositivos durante la fase de pruebas.

- **Distribution**
  - **Certificado de Distribución:** Necesario para publicar la aplicación en la App Store.
  - **Certificado de Distribución Ad-Hoc:** Usado para distribuir aplicaciones a un grupo de dispositivos específicos (para pruebas).
  - **Certificado de Distribución en la Empresa (Enterprise):** Usado por empresas para distribuir aplicaciones internamente en la organización.

Certificados y Provisioning Profiles

🎨 **Provisioning Profiles**

Los perfiles de aprovisionamiento son necesarios para especificar qué dispositivos pueden ejecutar la aplicación y qué certificados deben usarse para firmarla. Un perfil de aprovisionamiento vincula el certificado con un conjunto de dispositivos y un ID de aplicación, se crea en ***Apple Developer***.

Tipos de Perfiles:

- **Development:** Permite ejecutar la aplicación en dispositivos físicos durante el desarrollo.
- **Ad-Hoc:** Permite distribuir la aplicación a un número limitado de dispositivos para pruebas.
- **App Store:** Usado para distribuir la aplicación públicamente a través de la App Store.

https://developer.apple.com/

Publicación App Store

🌐 App Store Connect

Para publicar una aplicación en la App Store, se debe crear un registro de la app en ***App Store Connect***.

Se han de completar los detalles de la aplicación, incluyendo el nombre, ID de la aplicación (que debe coincidir con el configurado en el App ID en el portal de desarrolladores), categoría, información de la privacidad y países/regiones donde estará disponible.

Publicación App Store

🚶 **Subir la Aplicación a App Store Connect desde Xcode:**

- En Xcode, selecciona el **archivo del proyecto**.
- Seleccionar el **perfil de distribución** adecuado.
- En el menú de Xcode, en ***Product > Archive*** para generar un archivo listo para ser enviado a la App Store.
- Una vez completada la compilación, seleccionar el archivo y haz clic en **Upload** o ***Distribute App***.
- Seguir los pasos en Xcode para enviar la app a App Store Connect.

Existen aplicaciones que permiten la gestión de distribuir y publicar las aplicaciones como ***Firebase*** o ***Fastlane***.

Publicación App Store

🛠️ Proceso de Revisión de Apple y Aspectos Importantes

- Apple revisa cada aplicación enviada para asegurarse de que cumpla con las Directrices de la App Store, así como con los estándares de seguridad, privacidad y rendimiento.
- El proceso de revisión garantiza que las aplicaciones ofrecidas a los usuarios sean de alta calidad y no pongan en riesgo la seguridad de los dispositivos ni la privacidad de los usuarios.
- Asegura que la aplicación esté completamente funcional antes de enviarla para revisión.
- Realizar pruebas exhaustivas en dispositivos físicos y usar herramientas de análisis para verificar el rendimiento.

Publicación App Store

⏳ Cómo Funciona el Proceso de Revisión

Envío de la Aplicación:
Una vez se haya subido la aplicación a App Store Connect y completado los metadatos (como descripción, capturas de pantalla, etc.), se puede enviar la aplicación para su revisión.
Este proceso se inicia haciendo clic en el botón "Submit for Review".

Revisión de la Aplicación por Parte de Apple:
Apple realiza una revisión manual y automatizada de la aplicación. El equipo de revisión comprueba que la aplicación cumpla con todas las directrices de la App Store, como el cumplimiento de las políticas de privacidad, la usabilidad, el rendimiento y la seguridad.

Publicación App Store

⏳ Cómo Funciona el Proceso de Revisión

Resultado de la Revisión:

- **Aprobación:** Si la aplicación cumple con todos los requisitos, será aprobada y estará disponible en la App Store.
- **Rechazo:** Si no cumple con las directrices, se recibirá un informe detallado de las razones del rechazo. Apple proporcionará sugerencias sobre cómo corregir los problemas.
- **Retraso en la Revisión:** En ocasiones, la revisión puede retrasarse debido a una alta carga de trabajo o la necesidad de una revisión más exhaustiva.

Publicación App Store

📈 **Consejos publicación**

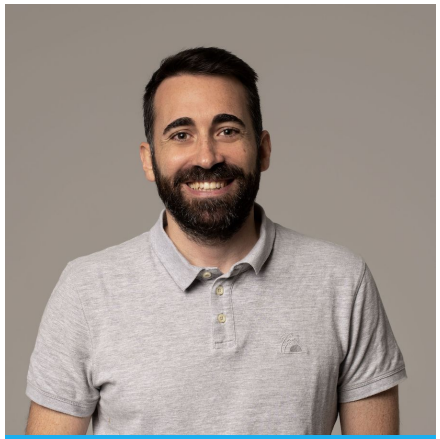
- **La aplicación debe ser completamente funcional:** Apple revisa que la aplicación no tenga errores ni funciones rotas. Las aplicaciones que se envían en un estado incompleto o que no cumplen con su propósito serán rechazadas**.**

- **Interfaz de usuario clara y atractiva:** Apple valora mucho la experiencia de usuario. La aplicación debe ser fácil de usar, tener un diseño intuitivo y respetar los estándares de diseño.

- **Funcionalidades engañosas:** Las aplicaciones no deben tener funcionalidad oculta, como el uso de publicidad intrusiva o funcionalidades que engañen a los usuarios.

https://developer.apple.com/app-store/review/guidelines/

¡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.