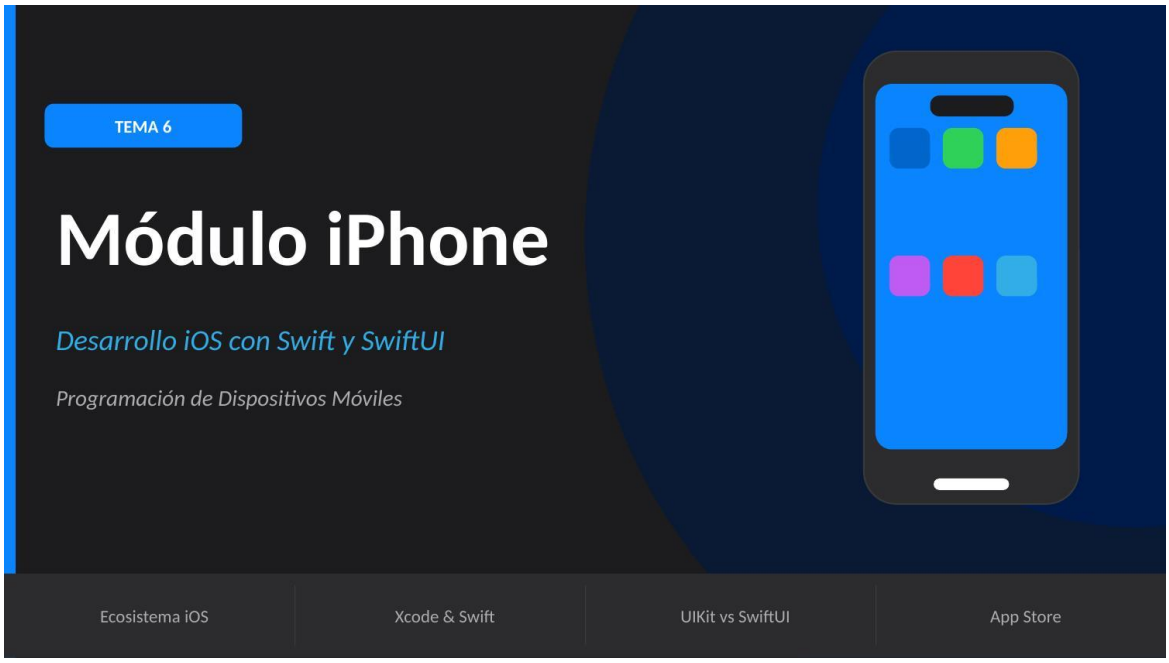
PROGRAMACIÓN DE DISPOSITIVOS MÓVILES

Tema 6

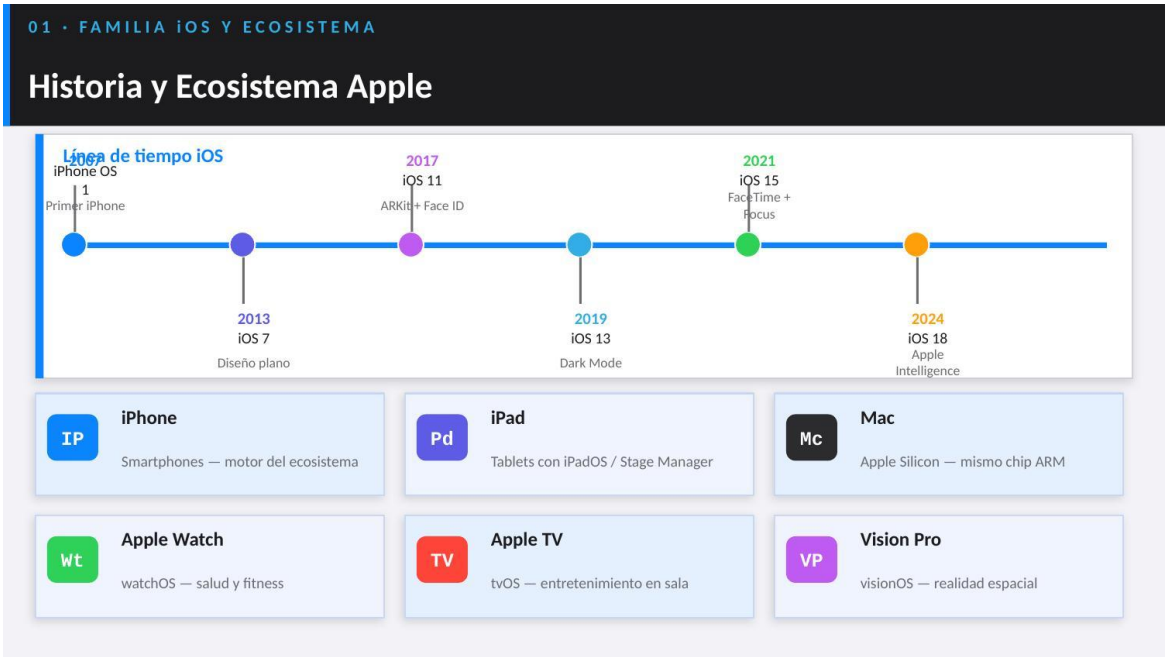
Módulo iPhone: Desarrollo iOS con Swift y SwiftUI



> *Estos apuntes cubren el desarrollo de aplicaciones para iPhone y el ecosistema Apple: la familia de dispositivos iOS, el entorno de desarrollo con Xcode, el lenguaje Swift (desde fundamentos hasta concurrencia moderna), la comparativa entre UIKit y SwiftUI, y el proceso completo de publicación en App Store.*

Contenidos

1. Introducción a la Familia iOS



1.1. Historia y Ecosistema Apple

Apple ha construido el ecosistema tecnológico más cohesionado del mundo. Lo que comenzó como un teléfono sin teclado físico en 2007 se ha convertido en una plataforma que unifica dispositivos, servicios y lenguajes de programación bajo un mismo paraguas:

Año	Versión	Aportación clave
2007	iPhone OS 1	Primer iPhone — capacitive touchscreen, sin App Store
2008	iPhone OS 2	App Store con 500 apps el día del lanzamiento
2010	iOS 4	Multitarea, FaceTime, Game Center
2013	iOS 7	Rediseño total con diseño plano (Jony Ive)
2017	iOS 11	ARKit, Face ID en iPhone X, Edge-to-Edge
2019	iOS 13	Dark Mode, Sign in with Apple, SwiftUI 1.0
2021	iOS 15	FaceTime SharePlay, Focus, Live Text con IA
2023	iOS 17	StandBy mode, Journal, NameDrop, Interactive Widgets
2024	iOS 18	Apple Intelligence, AI integrada, personalización pantalla

El Ecosistema Apple

Dispositivo	Sistema operativo	Características para desarrolladores
iPhone	iOS 18	Mercado principal. Swift + SwiftUI + UIKit

Dispositivo	Sistema operativo	Características para desarrolladores
iPad	iPadOS 18	Stage Manager, Apple Pencil, Split View
Mac (Apple Silicon)	macOS 15	Misma arquitectura ARM — apps iOS en Mac
Apple Watch	watchOS 11	SwiftUI nativo, HealthKit, sensores biométricos
Apple TV	tvOS 18	SwiftUI + TVUIKit para interfaz de sala
Apple Vision Pro	visionOS 2	RealityKit, ARKit — realidad espacial

> *Ventaja clave: Con Swift y SwiftUI, el mismo código puede ejecutarse en iPhone, iPad, Mac, Apple Watch y Apple TV con ajustes mínimos. Esto multiplica el valor de aprender el ecosistema Apple.*

1.2. Versiones de iOS y Compatibilidad

Distribución de versiones (2024)

- **iOS 17/18:** ~70% de los dispositivos activos
- **iOS 16:** ~15% — aún activo en iPhone X, XS, iPhone 8
- **iOS 15 o anterior:** ~15% — versiones heredadas, chips A12 y anteriores

Política de soporte Apple

- **Actualizaciones:** Apple da soporte a los últimos 5-7 años de hardware (ej: iOS 18 soporta desde iPhone XS/2018)
- **Adoption rate:** ~95% de los dispositivos activos corren iOS de los últimos 2 años — fragmentación mínima
- **Estrategia recomendada:** Minimum deployment target iOS 16 o 17 cubre a más del 95% del mercado activo

1.3. iOS vs Android: Diferencias Clave

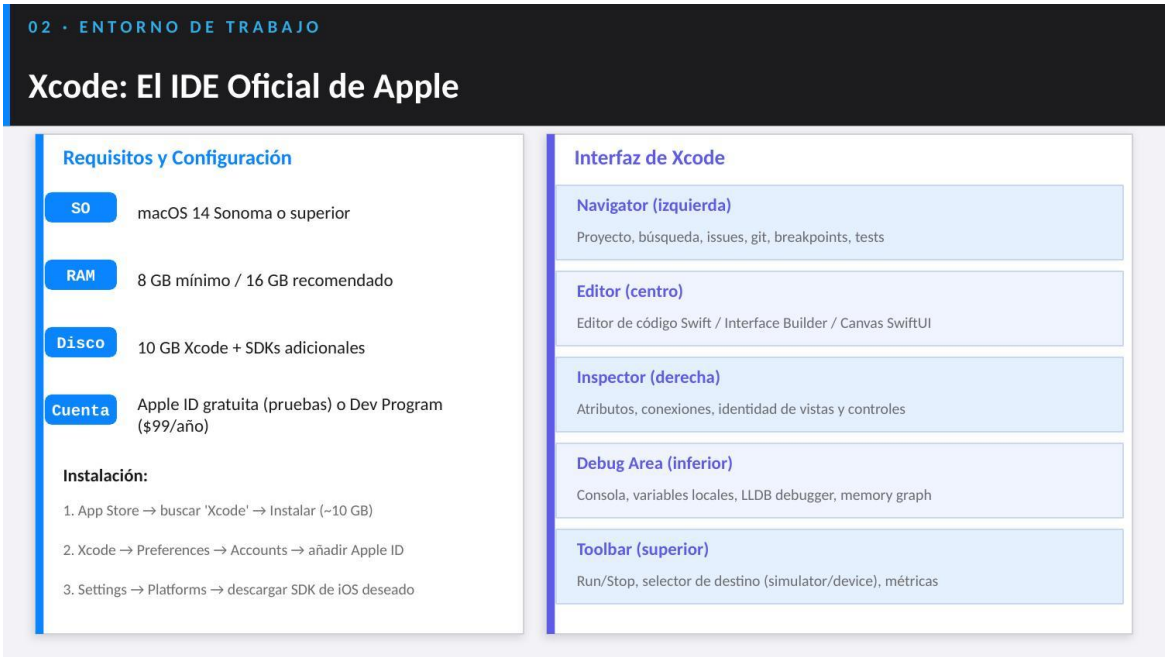
01 · FAMILIA iOS Y ECOSISTEMA

iOS vs Android: Diferencias Clave

Aspecto	iOS (Apple)	Android (Google)
Lenguaje nativo	Swift / SwiftUI	Kotlin / Jetpack Compose
IDE	Xcode (solo macOS)	Android Studio (multi-OS)
Fragmentación	Muy baja — ~5 modelos activos	Muy alta — 1000+ modelos
Cuota de mercado	~29% global, 55%+ en EE.UU.	~70% global
Monetización/usuario	2-3x mayor que Android	Base de referencia
Revisión en tienda	Manual, 1-7 días	Automática, horas
Ciclo de actualización OS	~95% en 2 años	~50% en 2 años
Distribución de apps	Solo App Store (oficial)	Play Store + APK directo
Lenguaje del sistema	Swift / Obj-C, framework nativo	Kotlin/Java, sobre JVM
Precio del programa Dev	$99/año obligatorio	$25 pago único

Dimensión	iOS (Apple)	Android (Google)
Lenguaje nativo principal	Swift / SwiftUI	Kotlin / Jetpack Compose
IDE oficial	Xcode — solo macOS	Android Studio — multiplataforma
Fragmentación de dispositivos	Muy baja (~5 modelos principales)	Muy alta (1.000+ modelos distintos)
Cuota de mercado global	~29% (55%+ en EE.UU. y Japón)	~70% (dominante en LATAM, África, Asia)
Gasto por usuario	2-3x mayor que Android	Base de referencia
Revisión de apps	Manual por Apple, 1-7 días	Mayoritariamente automático, horas
Actualizaciones del OS	~95% actualizan en 2 años	~50% actualizan en 2 años
Distribución de apps	Solo App Store (oficial)	Play Store + APK directo + stores alternativas
Coste del Developer Program	$99 USD/año obligatorio	$25 USD pago único

2. Entorno de Trabajo



2.1. Xcode

Xcode es el IDE oficial de Apple para el desarrollo de apps en todos los sistemas operativos del ecosistema Apple. Es completamente gratuito en la Mac App Store, aunque requiere macOS actualizado.

Requisitos del sistema

Componente	Mínimo	Recomendado
Sistema operativo	macOS 14 Sonoma	macOS 15 Sequoia
Procesador	Mac con Intel o Apple Silicon	Apple Silicon M1 o superior
RAM	8 GB	16 GB o más
Espacio en disco	10 GB para Xcode + SDKs	50 GB+ para simuladores adicionales
Cuenta Apple	Apple ID gratuita (simulador)	$99/año para subir a App Store

Instalación y configuración

1. **Descargar Xcode:** Mac App Store → buscar 'Xcode' → Instalar (~10 GB). Siempre usar la última versión estable.
2. **Añadir cuenta Apple:** Xcode → Settings → Accounts → '+' → Apple ID
3. **Descargar SDKs:** Xcode → Settings → Platforms → iOS → descargar la versión objetivo
4. **Instalar Command Line Tools:** xcode-select --install en Terminal (necesario para herramientas de compilación)

Paneles de la interfaz de Xcode

Panel	Descripción y funcionalidad
Navigator (izquierda)	Árbol de proyecto, búsqueda, issues, Git, tests y breakpoints
Editor (centro)	Editor de código Swift, Interface Builder visual, SwiftUI Canvas preview en tiempo real
Inspector (derecha)	Atributos, conexiones IBOutlet/IBAction, identidad de views y controles UI
Debug Area (inferior)	Consola de output, variables locales, LLDB debugger, Memory Graph Debugger
Toolbar (superior)	Botón Run/Stop, selector de destino (simulador/device), métricas de build

Atajos de teclado esenciales de Xcode

```
Cmd + R           → Compilar y ejecutar la app
Cmd + B           → Solo compilar (sin ejecutar)
Cmd + .           → Detener la ejecución
Cmd + Shift + K   → Limpiar la build (Clean Build Folder)
Cmd + /           → Comentar / descomentar línea
Ctrl + I          → Re-indentar código seleccionado
Cmd + 0           → Mostrar/ocultar Navigator
Cmd + Option + 0  → Mostrar/ocultar Inspector
Cmd + U           → Ejecutar todos los tests
Cmd + Shift + O   → Open Quickly (buscar archivo)
```

2.2. Swift: Lenguaje de Programación

Swift es el lenguaje de programación creado por Apple en 2014 (open source desde 2015). Combina seguridad de tipos, rendimiento nativo y sintaxis moderna. Es el sucesor de Objective-C.

Características principales:

- **Seguro:** Sistema de tipos estricto, optionals para valores nulos, ARC para gestión de memoria
- **Rápido:** Compilado a código máquina nativo ARM — rendimiento comparable a C++
- **Moderno:** async/await, @Observable, property wrappers, result builders, macros
- **Interoperable:** Puede llamar a código Objective-C existente sin wrappers adicionales

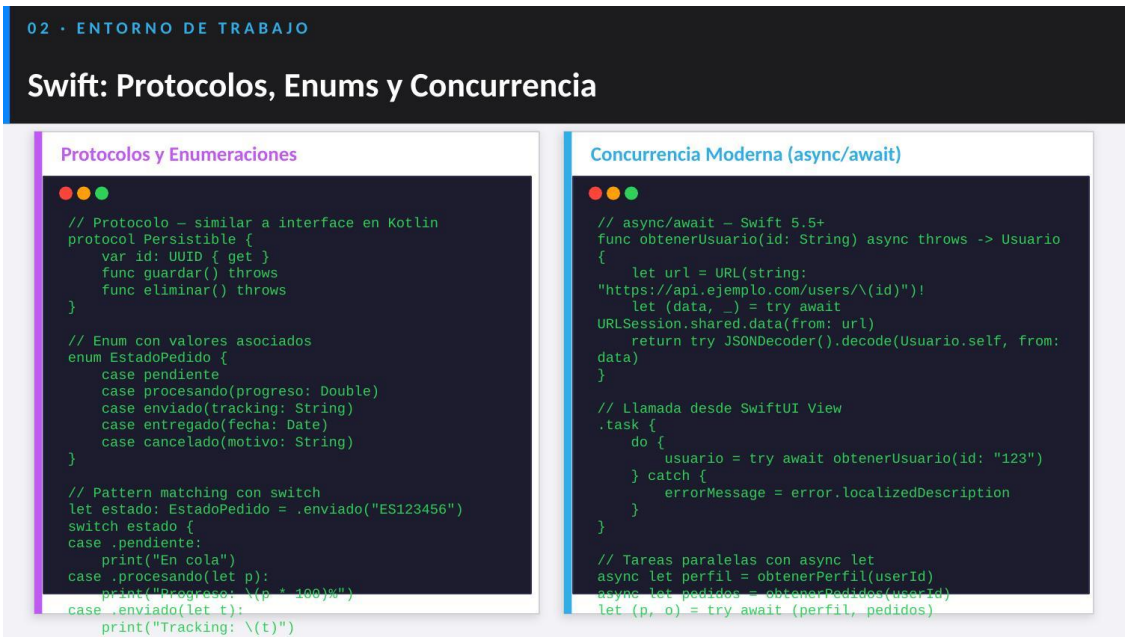
Variables, Constantes y Tipos

```
// Constante (inmutable) — usar siempre que sea posible
let nombre: String = "Ana García"
let edad = 28              // Swift infiere el tipo: Int
let precio: Double = 9.99

// Variable (mutable)
var puntuacion: Int = 0
var activo: Bool = true

// Opcionales — puede ser un valor o nil
var email: String? = nil
email = "ana@ejemplo.com"

// Optional binding (unwrap seguro con if let)
if let e = email {
    print("Email: \(e)")
}

// Guard let — early exit, código más limpio
func procesar(email: String?) {
    guard let e = email else {
        print("Sin email"); return
    }
    enviarCorreo(a: e)
}

// Nil coalescing — valor por defecto si nil
let texto = email ?? "sin email"
```

Funciones, Closures y Structs

```
// Función con etiquetas externas e internas
func saludar(a nombre: String, veces n: Int = 1) -> String {
    return Array(repeating: "Hola \(nombre)!", count: n).joined(separator: "
")
}
saludar(a: "Ana")           // "Hola Ana!"
saludar(a: "Ana", veces: 3) // "Hola Ana! Hola Ana! Hola Ana!"

// Closure (función anónima — muy usada en SwiftUI)
let cuadrado: (Int) -> Int = { numero in numero * numero }
let cuadrado2: (Int) -> Int = { $0 * $0 }  // shorthand
cuadrado(5)  // 25

// Struct (tipo valor — se copia al asignar)
struct Producto {
    let id: UUID = UUID()
    var nombre: String
```

```
    var precio: Double

    // Computed property
    var descripcion: String { "\(nombre) — \(precio)€" }

    // mutating: modifica propiedades del struct
    mutating func aplicarDescuento(_ porcentaje: Double) {
        precio *= (1 - porcentaje)
    }
}

var prod = Producto(nombre: "App Pro", precio: 9.99)
prod.aplicarDescuento(0.20)  // precio = 7.99
```



Protocolos y Enumeraciones

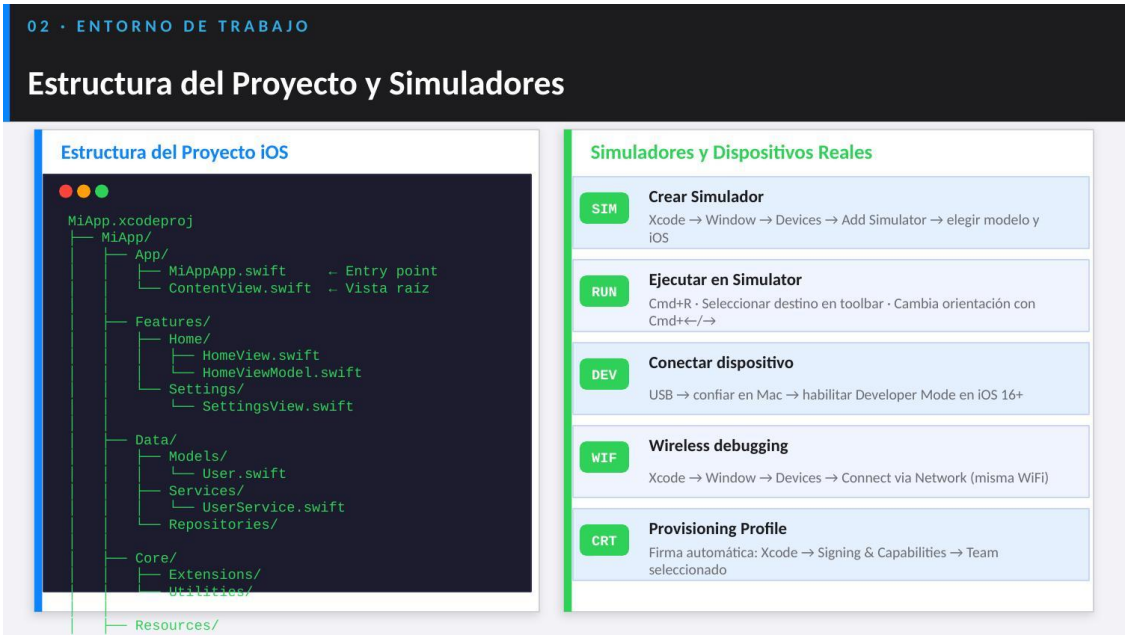
```
// Protocolo — define un contrato que los tipos deben cumplir
protocol Persistible {
    var id: UUID { get }
    func guardar() throws
    func eliminar() throws
}

// Enum con valores asociados — muy potente en Swift
enum EstadoPedido {
    case pendiente
    case procesando(progreso: Double)  // con valor
    case enviado(tracking: String)
    case entregado(fecha: Date)
    case cancelado(motivo: String)
}

// Pattern matching exhaustivo
let estado: EstadoPedido = .enviado("ES123456789")
switch estado {
case .pendiente:
    print("En cola de procesamiento")
case .procesando(let progreso):
    print("Progreso: \(progreso * 100)%")
case .enviado(let tracking):
    print("Tracking: \(tracking)")
case .entregado(let fecha):
    print("Recibido el \(fecha)")
case .cancelado(let motivo):
    print("Cancelado: \(motivo)")
}
```

Concurrencia Moderna con async/await

```
// async/await — Swift 5.5+ (iOS 15+)
```

```
func obtenerUsuario(id: String) async throws -> Usuario {
    let url = URL(string: "https://api.ejemplo.com/users/\(id)")!
    let (data, _) = try await URLSession.shared.data(from: url)
    return try JSONDecoder().decode(Usuario.self, from: data)
}

// Llamada desde SwiftUI con .task { }
.task {
    do {
        usuario = try await obtenerUsuario(id: "123")
    } catch {
        errorMessage = error.localizedDescription
    }
}

// Paralelismo — async let para tareas simultáneas
async let perfil = obtenerPerfil(userId)
async let pedidos = obtenerPedidos(userId)
let (p, o) = try await (perfil, pedidos)  // ambas en paralelo

// Actor — protege datos con acceso concurrente seguro
actor ContadorSeguro {
    private var valor = 0
    func incrementar() { valor += 1 }
    func obtenerValor() -> Int { valor }
}
```

2.3. Simuladores y Dispositivos Reales



Simuladores de iOS

Los simuladores permiten probar apps en múltiples modelos y versiones de iOS sin necesidad de tener los dispositivos físicos:

- **Crear simulador:** Xcode → Window → Devices and Simulators → '+' → elegir modelo y versión de iOS

- **Ejecutar:** Cmd+R con el simulador seleccionado en la toolbar
- **Cambiar orientación:** Cmd+← / Cmd+→ para rotar el dispositivo simulado
- **Simular condiciones especiales:** Features → Location, Slow Animations, Push Notifications, Face ID
- **Captura de pantalla:** Cmd+S en el simulador — se guarda automáticamente en el Desktop

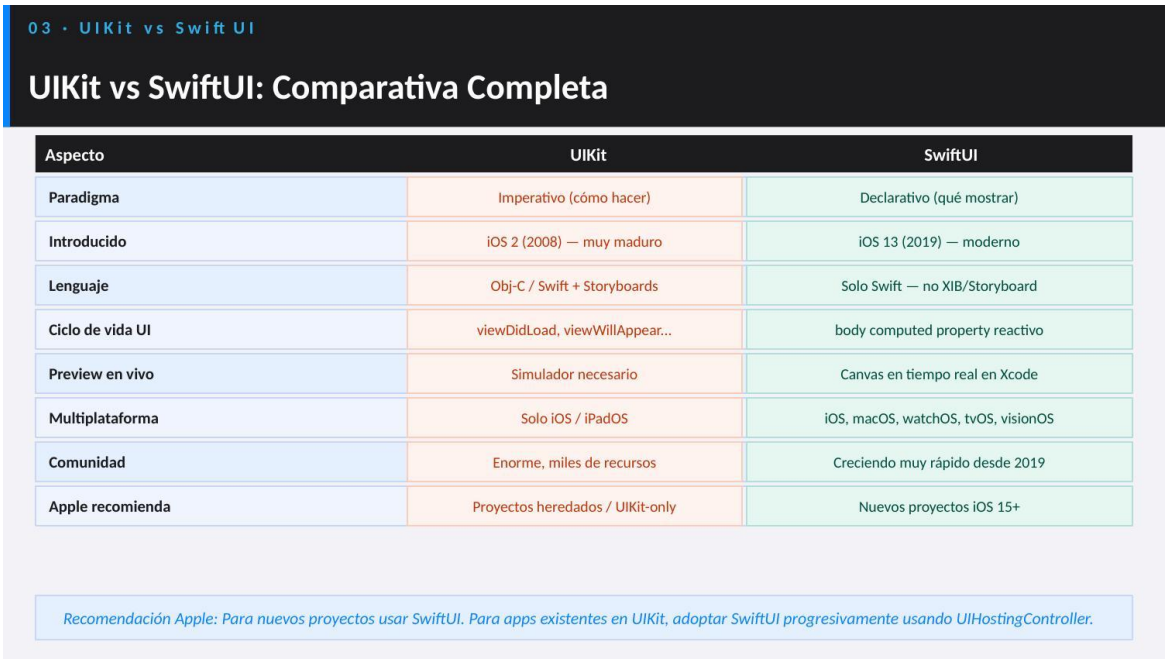
Conectar dispositivo real

5. **Developer Mode:** iOS 16+: Ajustes → Privacidad y seguridad → Modo desarrollador → Activar y reiniciar
6. **Conectar por USB:** Lightning/USB-C → confiar en el Mac cuando aparezca el diálogo en el iPhone
7. **Seleccionar destino:** En la toolbar de Xcode, seleccionar el dispositivo físico (aparece con nombre y modelo)
8. **Firma automática:** Xcode → Signing & Capabilities → Team → seleccionar tu Apple ID
9. **Wireless debugging:** Window → Devices and Simulators → Connect via Network (misma WiFi)

Estructura del proyecto iOS

```
MiApp.xcodeproj
├── MiApp/
│   ├── App/
│   │   ├── MiAppApp.swift        ← Entry point (@main)
│   │   └── ContentView.swift     ← Vista raíz
│   │
│   ├── Features/
│   │   ├── Home/
│   │   │   ├── HomeView.swift
│   │   │   └── HomeViewModel.swift
│   │   └── Settings/
│   │       └── SettingsView.swift
│   │
│   ├── Data/
│   │   ├── Models/      ← Structs/Clases de datos
│   │   ├── Services/    ← API calls, Firebase...
│   │   └── Repositories/
│   │
│   ├── Resources/
│   │   ├── Assets.xcassets  ← Imágenes, iconos, colores
│   │   └── Localizable.strings
│   │
│   └── Info.plist           ← Configuración de la app
│
└── MiAppTests/              ← Unit Tests y UI Tests
```

3. Aplicaciones Básicas: UIKit vs SwiftUI

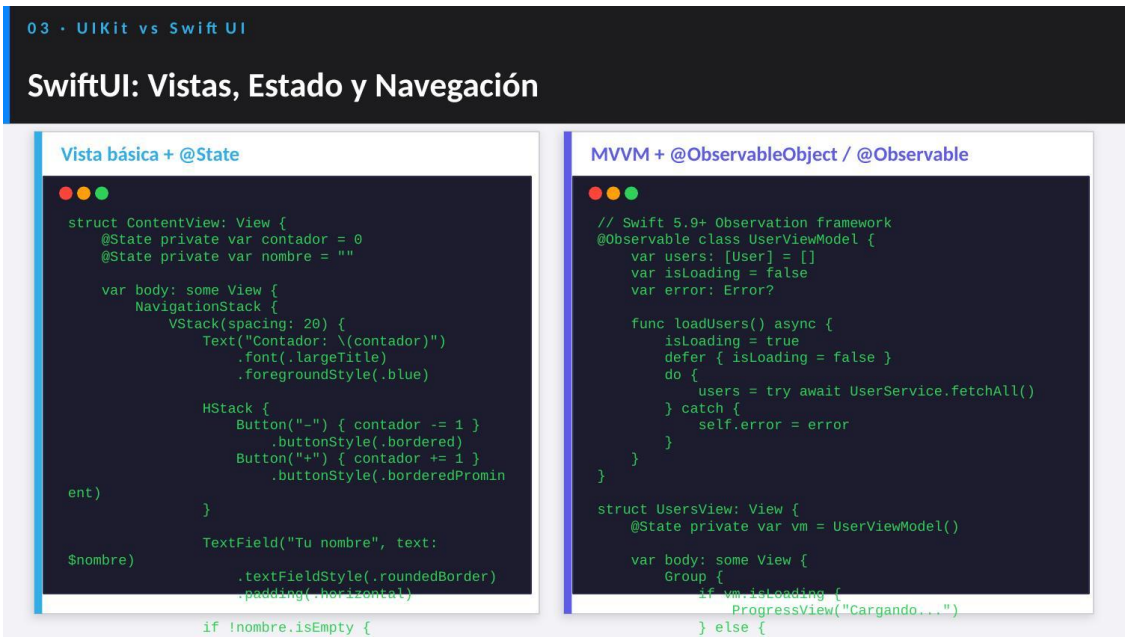


3.1. UIKit vs SwiftUI

Aspecto	UIKit	SwiftUI
Paradigma	Imperativo — describes cómo hacer la UI	Declarativo — describes qué mostrar
Introducido en	iOS 2 (2008) — maduro y estable	iOS 13 (2019) — moderno y en evolución
Lenguaje y herramientas	Swift/Obj-C + Storyboards/XIB files	Solo Swift — sin ficheros de interfaz visual
Ciclo de vida de la UI	viewDidLoad, viewWillAppear, etc.	body: some View — reactivo y automático
Preview en tiempo real	Solo con el simulador arrancado	Canvas de Xcode actualiza en milisegundos
Soporte multiplataforma	Solo iOS / iPadOS	iOS, macOS, watchOS, tvOS, visionOS
Recomendación Apple	Proyectos heredados y funciones avanzadas	Todos los nuevos proyectos iOS 15+

> *Recomendación Apple (2024): Usar SwiftUI para todos los proyectos nuevos. Para apps UIKit existentes, adoptar SwiftUI progresivamente usando UIHostingController para nuevas pantallas.*

3.2. SwiftUI: Vistas y Estado



Vista básica con @State

```
struct ContentView: View {
    @State private var contador = 0
    @State private var nombre = ""

    var body: some View {
        NavigationStack {
            VStack(spacing: 20) {
                Text("Contador: \(contador)")
                    .font(.largeTitle)
                    .foregroundStyle(.blue)

                HStack {
                    Button("-") { contador -= 1 }
                        .buttonStyle(.bordered)
                    Button("+") { contador += 1 }
                        .buttonStyle(.borderedProminent)
                }

                TextField("Tu nombre", text: $nombre)
                    .textFieldStyle(.roundedBorder)
                    .padding(.horizontal)

                if !nombre.isEmpty {
                    Text("Hola, \(nombre)!")
                        .transition(.opacity)
                }
            }
            .animation(.default, value: nombre)
            .navigationTitle("Mi App")
        }
    }
}
```

MVVM con @Observable (Swift 5.9+)

El patrón **MVVM (Model-View-ViewModel)** es el patrón arquitectónico preferido en SwiftUI. El ViewModel gestiona el estado y la lógica de negocio; la Vista solo dibuja lo que el ViewModel expone:

```
// ViewModel — Swift 5.9 Observation framework (iOS 17+)
@Observable class UserViewModel {
    var users: [User] = []
    var isLoading = false
    var errorMessage: String?

    func loadUsers() async {
        isLoading = true
        defer { isLoading = false }
        do {
            users = try await UserService.fetchAll()
        } catch {
            errorMessage = error.localizedDescription
        }
    }
}

// View — solo renderiza, no contiene lógica
struct UsersView: View {
    @State private var vm = UserViewModel()

    var body: some View {
        Group {
            if vm.isLoading {
                ProgressView("Cargando...")
            } else {
                List(vm.users) { user in
                    Text(user.name)
                }
            }
        }
        .task { await vm.loadUsers() }
    }
}
```

Componentes UI más usados en SwiftUI

Componente	SwiftUI	UIKit equivalente
Texto	Text("Hola")	UILabel
Campo de texto	TextField("placeholder", text: $var)	UITextField
Botón	Button("Acción") { ... }	UIButton
Imagen	Image("nombre") / Image(systemName: "star")	UIImageView
Lista	List { ForEach(items) { ... } }	UITableView
Grid/Cuadrícula	LazyVGrid(columns: ...) { ... }	UICollectionView
Navegación	NavigationStack → NavigationLink	UINavigationController

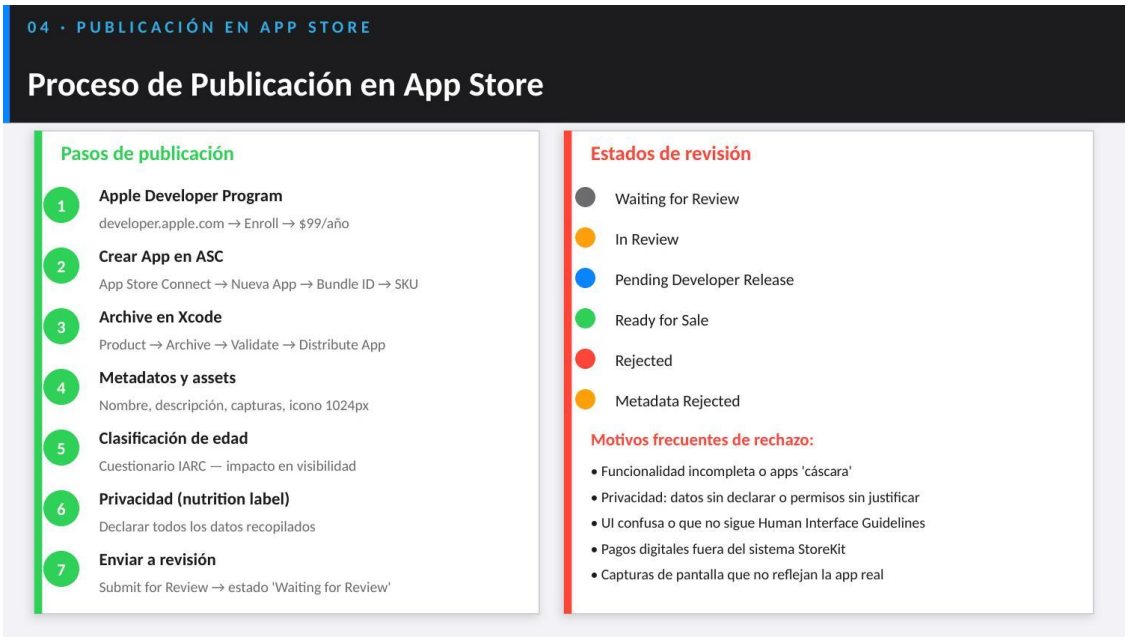
Componente	SwiftUI	UIKit equivalente
Modal / Sheet	.sheet(isPresented: $show) { ... }	present(_:animated:)

Interoperabilidad UIKit ↔ SwiftUI

Cuando necesites usar una vista UIKit dentro de SwiftUI, o integrar SwiftUI en una app UIKit existente:

```
// UIKit dentro de SwiftUI — UIViewRepresentable
struct MapaView: UIViewRepresentable {
    func makeUIView(context: Context) -> MKMapView {
        MKMapView(frame: .zero)
    }
    func updateUIView(_ uiView: MKMapView, context: Context) {
        // actualizar vista cuando cambia el estado
    }
}

// SwiftUI dentro de UIKit — UIHostingController
class MiViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        let hostingVC = UIHostingController(rootView: ContentView())
        addChild(hostingVC)
        view.addSubview(hostingVC.view)
        hostingVC.didMove(toParent: self)
    }
}
```

4. Publicación en App Store



4.1. Checklist Pre-lanzamiento

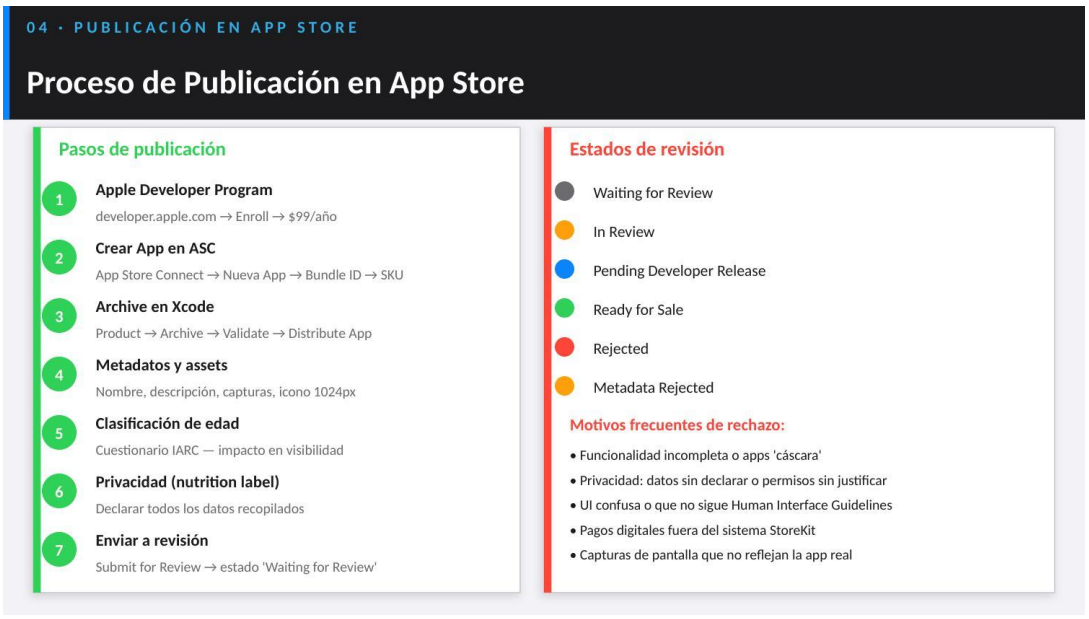
Aspecto técnico

- **Firma de código:** Certificado de distribución + Provisioning Profile de producción activos
- **Sin errores de compilación:** 0 errors, revisar warnings — algunos pueden causar rechazo
- **Build para todos los dispositivos objetivo:** iPhone, iPad si corresponde
- **Permisos NSUsage:** Todos los permisos con descripción en Info.plist (NSCameraUsageDescription, etc.)
- **Pruebas en dispositivo real:** No solo en simulador — probar en al menos iPhone y iPad físicos

Metadatos y assets

- **Icono de la app:** 1024×1024px PNG sin transparencia en Assets.xcassets
- **Capturas de pantalla:** Obligatorias para 6.5" (iPhone 14 Pro Max) y 5.5" (iPhone 8 Plus)
- **Nombre:** Máximo 30 caracteres. Incluir keyword principal al inicio
- **Subtítulo:** Máximo 30 caracteres. Se indexa en la búsqueda
- **Descripción:** Máximo 4.000 caracteres. Los primeros 3 párrafos son los más visibles
- **Privacy Nutrition Label:** Declarar todos los datos recopilados con precisión

4.2. Proceso de Publicación Paso a Paso



10. **Apple Developer Program:** developer.apple.com → Enroll → Verificar identidad → $99 USD/año → Acceso a App Store Connect y certificados
11. **Bundle ID:** Crear el App Identifier en developer.apple.com/account con Bundle ID único (com.empresa.nombre)
12. **Crear App en ASC:** appstoreconnect.apple.com → Apps → '+' → iOS App → Bundle ID → Nombre → SKU (interno)
13. **Archive en Xcode:** Product → Archive → esperar compilación → Validate App (detecta errores antes de subir) → Distribute App → App Store Connect → Upload
14. **TestFlight (beta):** Subir build → TestFlight → Añadir probadores internos/externos → distribución antes del lanzamiento
15. **Metadatos completos:** App Store Connect → rellenar toda la información del app: descripción, capturas, palabras clave, categoría, clasificación de edad, política de privacidad (URL obligatoria)
16. **Configuración de precios:** Pricing and Availability → precio (o gratuita) → países de distribución
17. **Enviar a revisión:** Add for Review → Submit for Review → la app entra en cola de revisión de Apple (1-7 días)

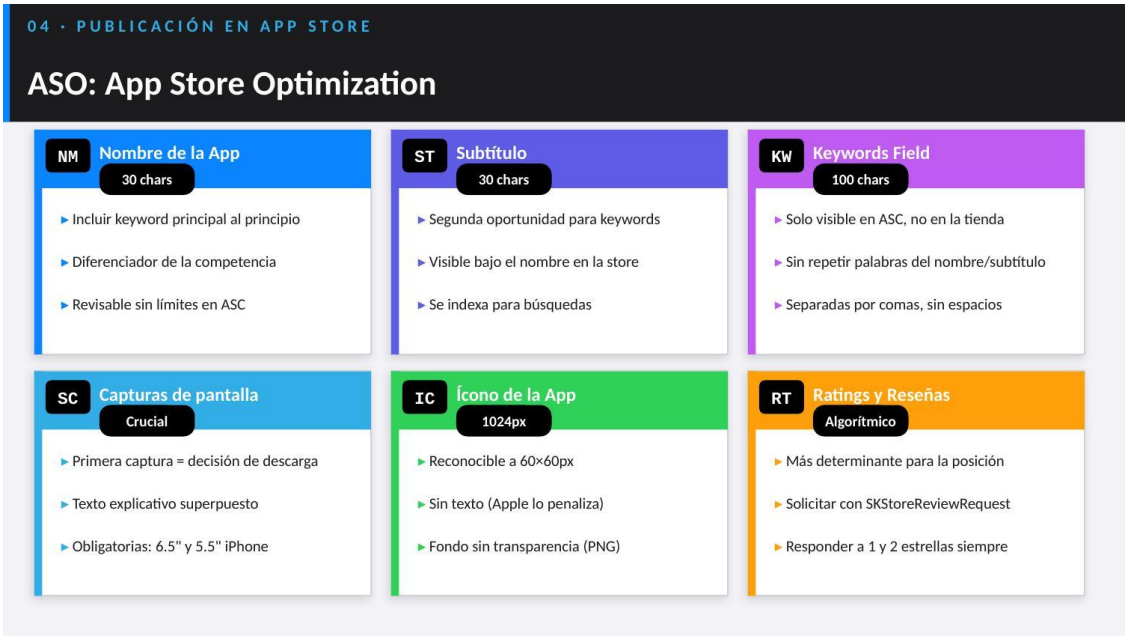
Estados del proceso de revisión

Estado	Significado
Waiting for Review	En cola — espera su turno para ser revisada
In Review	Activamente siendo revisada por el equipo de Apple
Pending Developer Release	Aprobada — el desarrollador controla la fecha de publicación
Ready for Sale	Publicada y visible en la App Store para los usuarios
Rejected	Rechazada — hay que corregir y volver a enviar
Metadata Rejected	Solo los metadatos son incorrectos (sin necesidad de nueva build)

Motivos comunes de rechazo

- **Funcionalidad incompleta:** Apps con pantallas en blanco, botones que no funcionan o contenido de demostración
- **Privacidad:** Datos recopilados no declarados en la Nutrition Label, permisos sin justificación en el texto
- **Diseño y UX:** Interfaces que no siguen las Human Interface Guidelines de Apple
- **Pagos in-app:** Compras digitales que no usan el sistema StoreKit/IAP de Apple (obligatorio al 100%)
- **Metadatos engañosos:** Capturas de pantalla que muestran funciones que la app no tiene
- **Crashes:** Apple prueba la app — un crash durante la revisión causa rechazo inmediato
- **Spam:** Apps demasiado similares a otras ya publicadas sin aportación de valor añadido

4.3. App Store Optimization (ASO)



ASO es el proceso de optimizar la ficha de la app para mejorar su posicionamiento en los resultados de búsqueda de la App Store y aumentar la tasa de conversión de visitas a descargas.

Factores de ASO en App Store

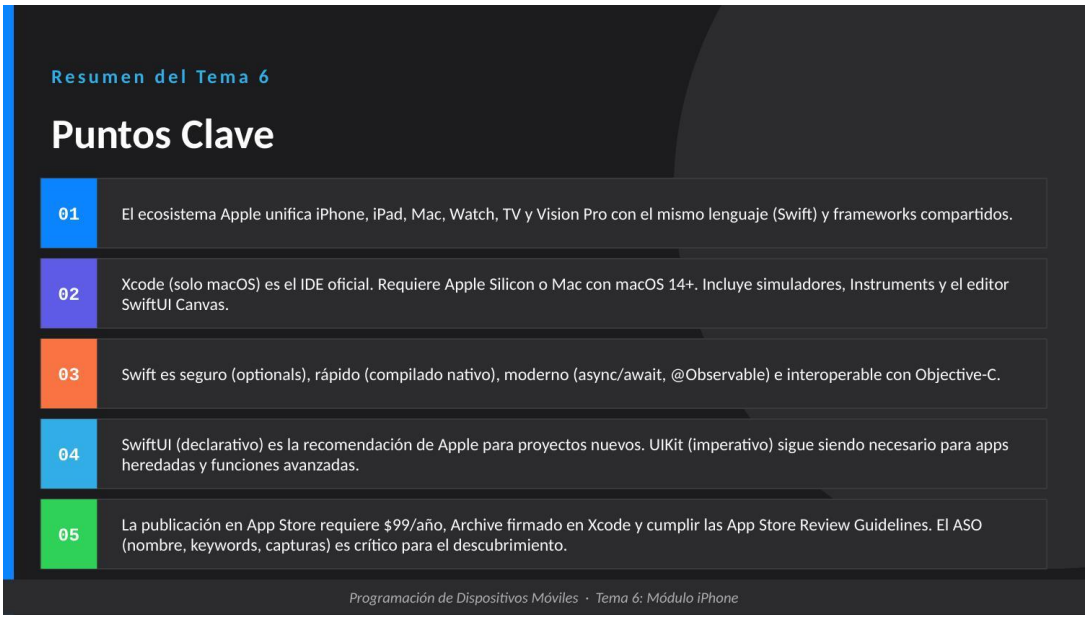
Factor	Límite	Estrategia recomendada
Nombre de la app	30 chars	Keyword principal al principio. Diferenciador claro.
Subtítulo	30 chars	Complementa el nombre con keywords secundarias.
Keywords field	100 chars	Solo en ASC — sin espacios tras comas, sin repetir keywords del nombre.
Descripción	4.000 chars	Los primeros 3 párrafos son los más leídos (antes del 'más').
Capturas de pantalla	3-10	Primera captura es decisión de descarga. Con texto explicativo.
Ratings y reseñas	Algorítmico	Solicitar con SKStoreReviewRequest. Responder a 1-2 estrellas.
Ícono	1024×1024px	Reconocible a 60×60px. Sin texto. Fondo sin transparencia.
Localización (i18n)	Por idioma	Localizar metadatos amplía la audiencia global significativamente.

Tiempo estimado del proceso completo

Tarea	Tiempo estimado	Notas
Inscripción Apple Developer	1-3 días	Verificación de identidad puede demorar

Tarea	Tiempo estimado	Notas
Preparar metadatos y assets	1-3 días	Diseño de capturas, traducciones si aplica
TestFlight (beta interna)	Instantáneo	Disponible en minutos tras la revisión automática
Revisión App Store (primera vez)	24h – 7 días	Primera app de un developer puede tardar más
Revisión (actualizaciones)	24h – 48h	Historias aceleradas con urgencia declarada

5. Resumen y Actividades



5.1. Puntos Clave del Tema

> *El ecosistema Apple (iPhone, iPad, Mac, Watch, TV, Vision Pro) se unifica bajo Swift y SwiftUI, permitiendo compartir código y conocimiento entre plataformas. Apple Silicon ha unificado también la arquitectura ARM de todos sus dispositivos.*

> *Xcode (solo macOS) es el IDE oficial y gratuito. Requiere macOS 14+ y Apple Silicon para el mejor rendimiento. Incluye el SwiftUI Canvas para preview en tiempo real, simuladores de todos los dispositivos e Instruments para profiling.*

> *Swift es seguro (optionals, type-safe), rápido (compilado a ARM nativo), moderno (async/await, @Observable, macros) e interoperable con Objective-C. Es el lenguaje más demandado en el ecosistema Apple.*

> *SwiftUI (declarativo, iOS 13+) es la recomendación de Apple para todos los proyectos nuevos. UIKit (imperativo, iOS 2+) sigue siendo necesario para apps heredadas y funcionalidades que SwiftUI aún no cubre completamente.*

> *La publicación en App Store requiere el Apple Developer Program ($99/año), Archive firmado desde Xcode y cumplir al 100% las App Store Review Guidelines. El ASO (nombre, keywords, capturas, ratings) es determinante para el descubrimiento orgánico.*

5.2. Actividades Prácticas

18. **Hello World iOS:** Crear un nuevo proyecto en Xcode con SwiftUI. Añadir un Text con 'Hola mundo', un Button que cambie el texto al pulsarlo y un TextField con @State. Ejecutar en simulador.
19. **Swift Playground:** Crear un nuevo Swift Playground y practicar: optionals con if let y guard let, struct con computed properties, enum con valores asociados, y una función async/await que simule una llamada de red.
20. **Lista con navegación:** Implementar una List de películas (array de structs con título, año y sinopsis) con NavigationStack y NavigationLink. La pantalla de detalle debe mostrar toda la información del ítem seleccionado.
21. **Llamada a API REST:** Usar URLSession con async/await para consumir la API pública https://jsonplaceholder.typicode.com/posts. Mostrar los posts en una List. Implementar un ProgressView mientras carga y manejo de errores.
22. **TestFlight:** Configurar el Apple Developer Program (o usar una cuenta gratuita para simulador), archivar la app con Product → Archive y distribuirla a probadores internos vía TestFlight.
23. **Ficha de App Store:** Redactar los metadatos completos de una app inventada: nombre (30 chars), subtítulo (30 chars), 5 keywords para el campo de keywords (100 chars total) y descripción de 500 palabras optimizada para ASO.
24. **UIViewControllerRepresentable:** Integrar el MKMapView de MapKit (UIKit) dentro de una vista SwiftUI usando UIViewRepresentable. Mostrar la ubicación actual del usuario en el mapa.

5.3. Recursos Adicionales

- Apple Developer Documentation — Documentación oficial completa de todos los frameworks
- Swift Programming Language Guide — Guía oficial del lenguaje Swift
- SwiftUI Tutorials — Tutoriales oficiales interactivos de Apple
- App Store Review Guidelines — Requisitos de publicación y políticas de Apple
- Human Interface Guidelines — Guía de diseño de Apple para todas las plataformas
- App Store Connect Help — Gestión de apps, TestFlight y metadatos
- SF Symbols — Más de 5.000 iconos del sistema Apple para usar en apps