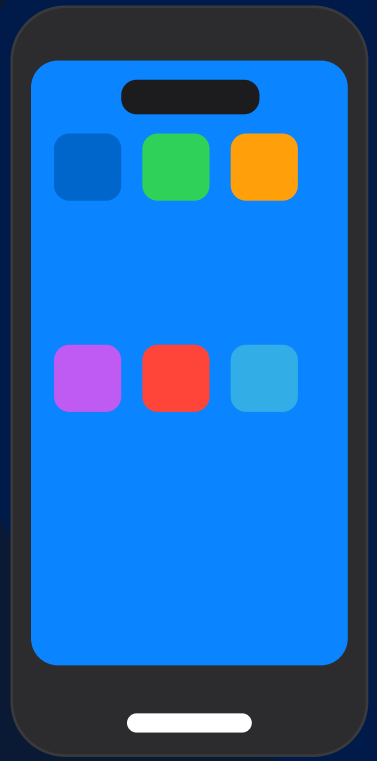
TEMA 6

Módulo iPhone

Desarrollo iOS con Swift y SwiftUI

Programación de Dispositivos Móviles

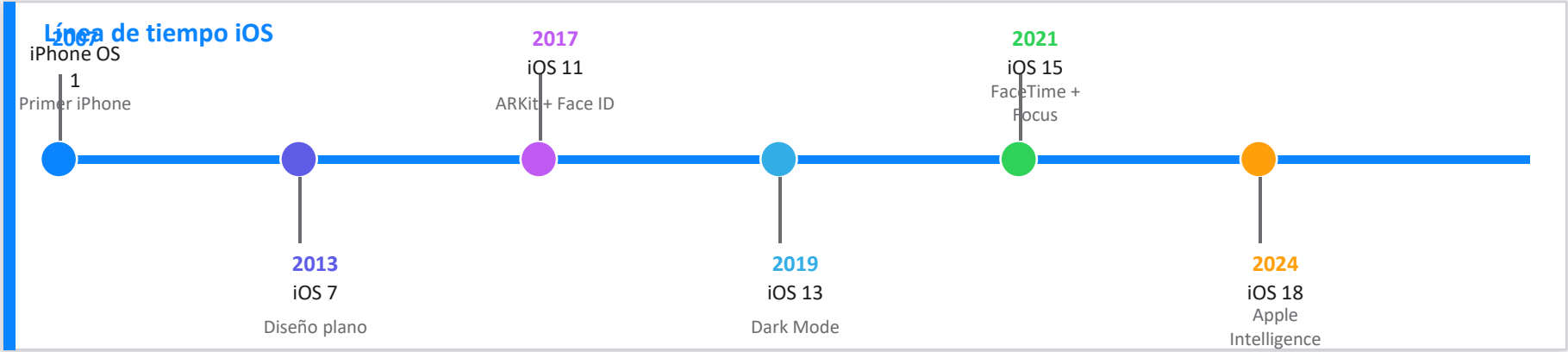


Ecosistema iOS | Xcode & Swift | UIKit vs SwiftUI | App Store

CONTENIDOS

Tema 6

Historia y Ecosistema Apple



IP

iPhone

Smartphones — motor del ecosistema

Pd

iPad

Tablets con iPadOS / Stage Manager

Mc

Mac

Apple Silicon — mismo chip ARM

Wt

Apple Watch

watchOS — salud y fitness

TV

Apple TV

tvOS — entretenimiento en sala

VP

Vision Pro

visionOS — realidad espacial

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

Xcode: El IDE Oficial de Apple

Requisitos y Configuración

SO	macOS 14 Sonoma o superior
RAM	8 GB mínimo / 16 GB recomendado
Disco	10 GB Xcode + SDKs adicionales
Cuenta	Apple ID gratuita (pruebas) o Dev Program ($99/año)

Instalación:

1. App Store → buscar 'Xcode' → Instalar (~10 GB)
2. Xcode → Preferences → Accounts → añadir Apple ID
3. Settings → Platforms → descargar SDK de iOS deseado

Interfaz de Xcode

Navigator (izquierda)

Proyecto, búsqueda, issues, git, breakpoints, tests

Editor (centro)

Editor de código Swift / Interface Builder / Canvas SwiftUI

Inspector (derecha)

Atributos, conexiones, identidad de vistas y controles

Debug Area (inferior)

Consola, variables locales, LLDB debugger, memory graph

Toolbar (superior)

Run/Stop, selector de destino (simulator/device), métricas

Swift: Variables, Tipos y Funciones

Variables, Constantes y Tipos

```swift
// Constante (inmutable) — preferida
let nombre: String = "Ana García"
let edad = 28              // Inferencia de tipos
let pi = 3.14159           // Double

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

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

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

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

Funciones, Closures y Structs

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

// Closure (función anónima)
let cuadrado: (Int) -> Int = { $0 * $0 }
cuadrado(5)  // 25

// Struct (tipo valor — copiado)
struct Producto {
    let id: UUID = UUID()
    var nombre: String
    var precio: Double

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

    mutating func aplicarDescuento(_ pct: Double) {
        precio *= (1 - pct)
    }
}
```

Swift: Protocolos, Enums y Concurrencia

Protocolos y Enumeraciones

```swift
// Protocolo — similar a interface en Kotlin
protocol Persistible {
    var id: UUID { get }
    func guardar() throws
    func eliminar() throws
}

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

// Pattern matching con switch
let estado: EstadoPedido = .enviado("ES123456")
switch estado {
case .pendiente:
    print("En cola")
case .procesando(let p):
    print("Progreso: \(p * 100)%")
case .enviado(let t):
    print("Tracking: \(t)")
```

Concurrencia Moderna (async/await)

```swift
// async/await — Swift 5.5+
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 View
.task {
    do {
        usuario = try await obtenerUsuario(id: "123")
    } catch {
        errorMessage = error.localizedDescription
    }
}

// Tareas paralelas con async let
async let perfil = obtenerPerfil(userId)
async let pedidos = obtenerPedidos(userId)
let (p, o) = try await (perfil, pedidos)
```

UIKit vs SwiftUI: Comparativa Completa

Aspecto	UIKit	SwiftUI
Paradigma	Imperativo (cómo hacer)	Declarativo (qué mostrar)
Introducido	iOS 2 (2008) — muy maduro	iOS 13 (2019) — moderno
Lenguaje	Obj-C / Swift + Storyboards	Solo Swift — no XIB/Storyboard
Ciclo de vida UI	viewDidLoad, viewWillAppear...	body computed property reactivo
Preview en vivo	Simulador necesario	Canvas en tiempo real en Xcode
Multiplataforma	Solo iOS / iPadOS	iOS, macOS, watchOS, tvOS, visionOS
Comunidad	Enorme, miles de recursos	Creciendo muy rápido desde 2019
Apple recomienda	Proyectos heredados / UIKit-only	Nuevos proyectos iOS 15+

Recomendación Apple: Para nuevos proyectos usar SwiftUI. Para apps existentes en UIKit, adoptar SwiftUI progresivamente usando UIHostingController.

SwiftUI: Vistas, Estado y Navegación

Vista básica + @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 {
```

MVVM + @ObservableObject / @Observable

```
// Swift 5.9+ Observation framework
@Observable class UserViewModel {
    var users: [User] = []
    var isLoading = false
    var error: Error?

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

struct UsersView: View {
    @State private var vm = UserViewModel()

    var body: some View {
        Group {
            if vm.isLoading {
                ProgressView("Cargando...")
            } else {
```

Estructura del Proyecto y Simuladores

Estructura del Proyecto iOS

```
MiApp.xcodeproj
├── MiApp/
│   ├── App/
│   │   ├── MiAppApp.swift      ← Entry point
│   │   └── ContentView.swift   ← Vista raíz
│   │
│   ├── Features/
│   │   ├── Home/
│   │   │   ├── HomeView.swift
│   │   │   └── HomeViewModel.swift
│   │   └── Settings/
│   │       └── SettingsView.swift
│   │
│   ├── Data/
│   │   ├── Models/
│   │   │   └── User.swift
│   │   ├── Services/
│   │   │   └── UserService.swift
│   │   └── Repositories/
│   │
│   ├── Core/
│   │   ├── Extensions/
│   │   └── Utilities/
│   │
│   ├── Resources/
```

Simuladores y Dispositivos Reales

SIM — **Crear Simulador**
Xcode → Window → Devices → Add Simulator → elegir modelo y iOS

RUN — **Ejecutar en Simulator**
Cmd+R · Seleccionar destino en toolbar · Cambia orientación con Cmd+←/→

DEV — **Conectar dispositivo**
USB → confiar en Mac → habilitar Developer Mode en iOS 16+

WIF — **Wireless debugging**
Xcode → Window → Devices → Connect via Network (misma WiFi)

CRT — **Provisioning Profile**
Firma automática: Xcode → Signing & Capabilities → Team seleccionado

Proceso de Publicación en App Store

Pasos de publicación

1. **Apple Developer Program**
   developer.apple.com → Enroll → $99/año
2. **Crear App en ASC**
   App Store Connect → Nueva App → Bundle ID → SKU
3. **Archive en Xcode**
   Product → Archive → Validate → Distribute App
4. **Metadatos y assets**
   Nombre, descripción, capturas, icono 1024px
5. **Clasificación de edad**
   Cuestionario IARC — impacto en visibilidad
6. **Privacidad (nutrition label)**
   Declarar todos los datos recopilados
7. **Enviar a revisión**
   Submit for Review → estado 'Waiting for Review'

Estados de revisión

- Waiting for Review
- In Review
- Pending Developer Release
- Ready for Sale
- Rejected
- Metadata Rejected

Motivos frecuentes de rechazo:

- Funcionalidad incompleta o apps 'cáscara'
- Privacidad: datos sin declarar o permisos sin justificar
- UI confusa o que no sigue Human Interface Guidelines
- Pagos digitales fuera del sistema StoreKit
- Capturas de pantalla que no reflejan la app real

ASO: App Store Optimization

NM Nombre de la App

30 chars

- Incluir keyword principal al principio
- Diferenciador de la competencia
- Revisable sin límites en ASC

ST Subtítulo

30 chars

- Segunda oportunidad para keywords
- Visible bajo el nombre en la store
- Se indexa para búsquedas

KW Keywords Field

100 chars

- Solo visible en ASC, no en la tienda
- Sin repetir palabras del nombre/subtítulo
- Separadas por comas, sin espacios

SC Capturas de pantalla

Crucial

- Primera captura = decisión de descarga
- Texto explicativo superpuesto
- Obligatorias: 6.5" y 5.5" iPhone

IC Ícono de la App

1024px

- Reconocible a 60×60px
- Sin texto (Apple lo penaliza)
- Fondo sin transparencia (PNG)

RT Ratings y Reseñas

Algorítmico

- Más determinante para la posición
- Solicitar con SKStoreReviewRequest
- Responder a 1 y 2 estrellas siempre

Resumen del Tema 6

Puntos Clave

01 El ecosistema Apple unifica iPhone, iPad, Mac, Watch, TV y Vision Pro con el mismo lenguaje (Swift) y frameworks compartidos.

02 Xcode (solo macOS) es el IDE oficial. Requiere Apple Silicon o Mac con macOS 14+. Incluye simuladores, Instruments y el editor SwiftUI Canvas.

03 Swift es seguro (optionals), rápido (compilado nativo), moderno (async/await, @Observable) e interoperable con Objective-C.

04 SwiftUI (declarativo) es la recomendación de Apple para proyectos nuevos. UIKit (imperativo) sigue siendo necesario para apps heredadas y funciones avanzadas.

05 La publicación en App Store requiere $99/año, Archive firmado en Xcode y cumplir las App Store Review Guidelines. El ASO (nombre, keywords, capturas) es crítico para el descubrimiento.

ACTIVIDADES PRÁCTICAS

Ejercicios del Tema 6

1 Hello World iOS

Crear app en Xcode con SwiftUI. Mostrar 'Hola mundo' y un botón que cambie el texto.

2 Swift Playground

Practicar optionals, structs, enums con valores asociados y async/await en un Swift Playground.

3 Lista con navegación

Implementar List → NavigationLink → DetailView usando un array de structs como fuente de datos.

4 Llamada a API REST

Usar async/await + URLSession para consumir una API pública y mostrar los datos en una List.

5 TestFlight

Archivar la app, subirla a App Store Connect y distribuirla a probadores vía TestFlight.

6 Ficha de App Store

Redactar metadatos completos de la app: nombre (30), subtítulo (30), keywords (100) y descripción.

7 Migración UIKit→SwiftUI

Integrar una UIViewController existente en SwiftUI usando UIViewControllerRepresentable.