1. Hero

Entry

Architectures of Android, iOS, and Windows

Architectural overview of Android, iOS, and Windows for designers, developers, and learners.

- Android layers (use green for quick scanning)
- iOS layers (cool blue for stability)
- Windows layers (teal for desktop heritage)

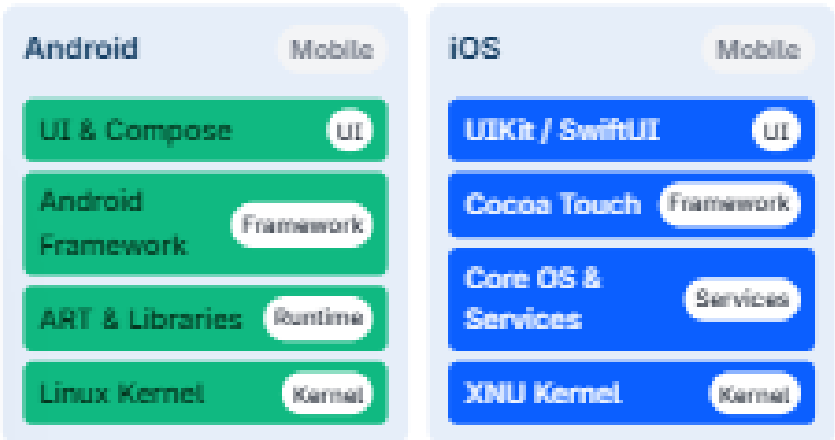
Overview | Developer perspective | System perspective

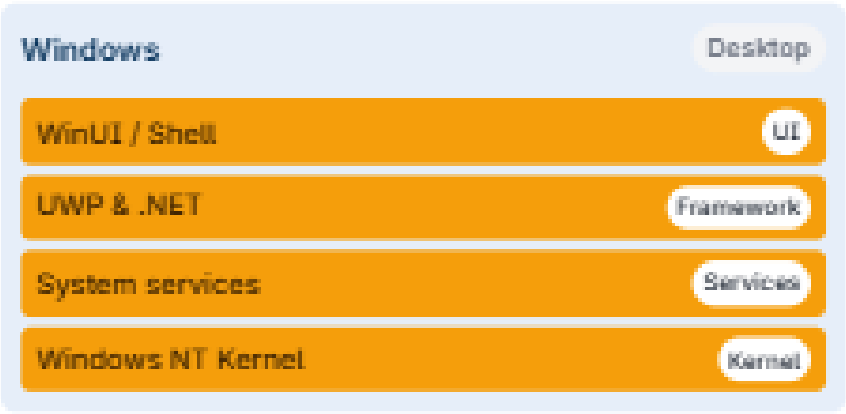
(1) Color coding and tabs define hierarchy. Keep labels < 30 characters for readability.

2. High-level comparison

Stack

Kernel	Linux vs XNU vs NT
UI layer	Views / UIKit / WinUI
Store	Play / App Store / Microsoft Store
Device scope	Phones, tablets, more





(2) Read each column top-down: UI → Framework → Runtime → Kernel. Use equal heights for quick comparison.

3. Architecture deep dives

Layers

Platform **Android**

UI Layer

Activities, fragments, and Jetpack Compose build screens and navigation.

Framework

Android Framework manages lifecycle, permissions, and system services.

Runtime & Libraries

ART executes bytecode and provides core Java/Kotlin APIs.

Linux Kernel

Low-level drivers, scheduling, and memory serve all processes.

Key strengths

- Flexible hardware support
- Rich background services
- Strong Google ecosystem

Key constraints

- OEM fragmentation
- Complex lifecycle handling

UI Layer

UIKit and SwiftUI define views, layout, and gestures.

Cocoa Touch

High-level frameworks for app structure, input, and animation.

Core Services

Networking, data, and cloud services integrated with the OS.

Core OS & XNU

Secure kernel and low-level services isolate apps.

Key strengths

- Tight hardware-software integration
- Consistent UX patterns

Key constraints

- Closed ecosystem
- Strict app review process

Tap to expand full iOS stack

Accordion (expand/collapse)

Platform

Windows

UI & Shell

Desktop shell, windows, and WinUI define the user experience.

App Frameworks

Win32, .NET, and UWP support legacy and modern apps.

System Services

Drivers, networking, and system APIs serve many device types.

NT Kernel

Multitasking kernel built for desktops and servers.

4. Cross-platform comparison

Matrix

Security	App sandbox + permissions	Strong sandbox + code signing	User accounts + UAC
Distribution	Play + side-loading	App Store only	Store + direct installs
UI rules	Material Design	Human Interface	Fluent Design
Hardware	Many OEMs	Apple silicon only	Wide PC range

(4) Icons + 2–3 word labels keep the matrix scannable, even on small screens.

5. Action storyboard

Timeline

User taps a button

Android	Touch event → View	Activity / ViewModel	Framework → OS
iOS	Touch → UIKit	Target-action / View	Run loop → Core OS
Windows	Input → Window	Message loop	System API → Kernel

System updates UI

Android	State updated	Compose re-renders	GPU draws frame
iOS	State change	UIKit / SwiftUI diff	Display refreshed
Windows	Event handled	UI invalidated	Window redraw

(5) Each lane shows the same moment across platforms—read horizontally for story, vertically for platform.

6. Summary & key takeaways

Wrap-up

- All three platforms share a layered design: UI → Framework → OS → Hardware.
- Android emphasizes flexibility; iOS emphasizes control and consistency; Windows balances legacy and modern needs.
- Security and distribution models strongly influence developer workflows.
- UI guidelines differ, but core interaction patterns are similar.

(6) Overlapping cards hint at shared concepts while keeping platform identities distinct.

Tooltip layer examples

Hovers

Framework layer

i

Info

(7) Convert these hover notes into CSS tooltips. Use generous padding for touch.

Comments & review

Clarity — Add notes...

Technical accuracy — Add notes...

Visual hierarchy — Add notes...