

Guide UX Mobile Complete 2024–2026 With Concrete iOS & Android Values

Scope, sources, and measurement rules

This guide consolidates concrete, measurable UX implementation rules for mobile apps across iOS and Android, grounded in official platform/documentation sources from `["company","Apple","consumer electronics and software company"]`, `["company","Google","technology company"]`, AndroidX/AOSP, and `["organization","World Wide Web Consortium","web standards body (w3c)"]`, plus selectively used archival “Programming Guide” PDFs when they provide the only explicit numeric defaults (e.g., Core Animation transaction duration). ¹

Units & cross-mapping (practical): - iOS layout: **pt** (points). Android layout: **dp** (density-independent pixels). Treat $pt \approx dp$ for *human-scale UI* decisions, then validate with device testing. (Practical convention; platform docs emphasize semantic tokens rather than fixed hex/color values.) ²

- Timing: iOS/UIKit/Core Animation often uses a **0.25 s “implicit default”** when no transaction duration is set; Android Material 3 tokens define explicit duration tiers in **ms**. ³

Visual themes and dark mode

Dark mode

Pattern	Rule	Value iOS	Value Android	Source
Semantic colors (background)	Use semantic dynamic background colors; do not hardcode hex values for surfaces	<code>UIColor.systemBackground</code> , <code>secondarySystemBackground</code> , <code>tertiarySystemBackground</code> (choose by layering depth)	Material 3 <code>ColorScheme</code> : <code>Surface</code> , <code>SurfaceVariant</code> , <code>Background</code> (and corresponding “on” colors)	⁴
Semantic colors (text)	Use semantic label colors for text hierarchy	<code>UIColor.label</code> , <code>secondaryLabel</code> , <code>tertiaryLabel</code> , <code>quaternaryLabel</code>	<code>OnSurface</code> , <code>OnSurfaceVariant</code> , <code>OnBackground</code> (avoid custom near-white/near-black text without contrast checks)	⁵

Pattern	Rule	Value iOS	Value Android	Source
Dividers	Prefer system separator/divider colors (often translucent)	<code>UIColor.separator</code> (may be partially transparent)	Use theme-based dividers (e.g., tokens mapping to <code>OnSurfaceVariant</code> / alpha depending on component)	6
Fills/overlays on existing backgrounds	Use system “fill” colors (they include transparency and adapt)	<code>UIColor.systemFill</code> , <code>secondarySystemFill</code> , <code>tertiarySystemFill</code> , <code>quaternarySystemFill</code>	Prefer M3 state layers/tokens on top of container colors (state-driven rather than generic opacity hacks)	7
Contrast (text)	Meet WCAG contrast in both light/dark	Normal text $\geq 4.5:1$; Large text $\geq 3:1$	Same ratios; validate against final rendered colors (including overlays/alpha)	8
Contrast (UI components)	UI components & graphical objects need sufficient contrast vs adjacent colors	$\geq 3:1$ for essential UI components/graphics	Same	9
Elevation cue in dark theme	Don't rely on “stronger shadows” in dark mode; use platform-supported elevation cues	Prefer system materials + subtle shadows; avoid pure black surfaces that remove depth cues	M3 tonal elevation: elevation primarily via tonal overlays (plus shadows)	10
Fixed elevation scale	Use platform token scale (measurable)	No single official dp scale; keep shadows subtle and consistent; prefer system components	M3 elevation tokens: Level0=0dp, Level1=1dp, Level2=3dp, Level3=6dp, Level4=8dp, Level5=12dp	11

Pattern	Rule	Value iOS	Value Android	Source
Icons that adapt automatically	Prefer template/tinted icons that inherit semantic colors	Use SF Symbols / template rendering + semantic tint (e.g., <code>.label</code> , <code>.secondaryLabel</code>)	Prefer vector icons tinted from <code>OnSurface</code> / <code>OnSurfaceVariant</code> ; use M3 tokens for active/inactive states	12
When to provide separate assets	Provide separate images only when meaning/brand requires different artwork	Use asset variants (Light/Dark) only when needed; otherwise use semantic colors	Use <code>-night</code> resources only when needed (avoid duplicating all assets)	(Guidance aligns with “semantic purpose over fixed appearance” principle.) 13
Common dark-mode bug	Avoid “off-black on off-black” + disabled text failing contrast	Don’t use custom gray-on-black without measuring contrast; use semantic colors	Same; watch disabled state contrast (frequent issue if you rely on default disabled alpha without testing)	14

iOS snippet (SwiftUI) — semantic colors in dark mode

```
import SwiftUI

struct ThemedCard: View {
    var body: some View {
        VStack(alignment: .leading, spacing: 12) {
            Text("Title")
                .foregroundColor(Color(UIColor.label))
                .font(.headline)

            Text("Secondary text that remains readable in Dark Mode.")
                .foregroundColor(Color(UIColor.secondaryLabel))
                .font(.subheadline)

            Divider()
                .background(Color(UIColor.separator))

            Button("Primary action") { }
                .buttonStyle(.borderedProminent)
        }
    }
}
```

```

        .padding(16)
        .background(Color(UIColor.secondarySystemBackground))
        .clipShape(RoundedRectangle(cornerRadius: 16, style: .continuous))
    }
}

```

(Uses iOS semantic label/separator/background color roles.) 15

Android snippet (Kotlin/Compose) — Material 3 color roles + elevation

```

@Composable
fun ThemedCard() {
    Card(
        colors = CardDefaults.cardColors(
            containerColor = MaterialTheme.colorScheme.surfaceVariant
        ),
        elevation = CardDefaults.cardElevation(defaultElevation = 3.dp) //
        token-aligned scale
    ) {
        Column(Modifier.padding(16.dp)) {
            Text(
                "Title",
                color = MaterialTheme.colorScheme.onSurface,
                style = MaterialTheme.typography.titleMedium
            )
            Text(
                "Secondary text.",
                color = MaterialTheme.colorScheme.onSurfaceVariant,
                style = MaterialTheme.typography.bodyMedium
            )
        }
    }
}

```

(Material 3 defines a measurable elevation token scale; Compose tokens show Level2=3dp.) 16

Checklist (dark mode)

- [] All surfaces/text/icons use **semantic roles** (no hardcoded black/white UI). 2
- [] Contrast verified: $\geq 4.5:1$ text, $\geq 3:1$ large text and UI components (including alpha overlays). 8
- [] Dividers use `separator` /token equivalents (no 1px pure-white lines in dark). 6
- [] Dark theme elevation uses tonal/elevation tokens (Android) and subtle hierarchy cues (iOS). 17
- [] Disabled states still readable and perceivable (test common states: disabled, pressed, focused).

18

Anti-patterns (dark mode)

Avoid “custom dark palette” that breaks system semantics; pure #000 backgrounds everywhere; using opacity-only disabled styles that drop below 3:1 contrast; shipping separate dark assets for the entire UI instead of using semantic roles; forgetting separator translucency and ending up with heavy dividers. ¹⁹

Official sources (dark mode)

Apple HIG Color (semantic/dynamic intent) ²⁰

iOS semantic colors: `label`, `secondaryLabel`, `separator`, `background` and `fill` families ²¹

Material 3/Compose color roles & tokens (`ColorSchemeKeyTokens`) ²²

Material 3 elevation tokens (dp scale) ¹¹

WCAG contrast requirements (text + non-text) ⁸

Tactile feedback and motion

Haptics and tactile feedback

Pattern	Rule	Value iOS	Value Android
“Impact” haptics	Use impact haptics for physical metaphor moments (snap, collision, “thud”)	<code>UIImpactFeedbackGenerator</code> styles: <code>light</code> , <code>medium</code> , <code>heavy</code> , <code>soft</code> , <code>rigid</code>	<code>HapticFeedbackConstant</code> where available; otherwise vibration APIs (<code>VibrationEffect</code>)
“Notification” haptics	Use for success/warning/error outcomes	<code>UINotificationFeedbackGenerator</code> types: <code>success</code> , <code>warning</code> , <code>error</code>	Use distinct, short patterns (don’t create “musical” patterns); respect user settings
“Selection” haptics	Use only when selection changes , not when it’s confirmed	<code>UISelectionFeedbackGenerator.selectionChanged()</code> (explicitly warned by Apple docs)	Use selection haptics for discrete value changes (pickers, scrubbers)
Latency control	Pre-warm where timing matters	<code>UIFeedbackGenerator.prepare()</code> is optional but “highly recommended” to minimize latency	Avoid calling vibration on every frame; coalesce repeated events

Pattern	Rule	Value iOS	Value Android
Frequency	Avoid adding motion/haptics on extremely frequent micro-interactions	“Generally avoid adding motion” to frequent interactions; system already animates standard controls	Apply haptics sparingly; maintain consistent mapping action→feedback
Accessibility fallback	Provide a visual alternative; don't rely only on vibration	Pair haptic with visible state change (color, icon, text, animation)	Same; vibration can be disabled or muted

iOS snippet (Swift) — coherent haptics mapping

```
import UIKit

final class Haptics {
    static let shared = Haptics()
    private let selection = UISelectionFeedbackGenerator()
    private let notify = UINotificationFeedbackGenerator()

    func prepare() {
        selection.prepare()
        notify.prepare()
    }

    func selectionChanged() {
        selection.selectionChanged()    // for discrete value changes only
    }

    func success() { notify.notificationOccurred(.success) }
    func warning() { notify.notificationOccurred(.warning) }
    func error() { notify.notificationOccurred(.error) }

    func impact(_ style: UIImpactFeedbackGenerator.FeedbackStyle) {
        let gen = UIImpactFeedbackGenerator(style: style)
        gen.prepare()
        gen.impactOccurred()
    }
}
```

(Apple explicitly cautions selection feedback usage; `prepare()` reduces latency.) 29

Android snippet (Kotlin) — haptic constants + fallback vibration

```
fun View.hapticConfirm() {  
    // Prefer system haptic constants (consistent across apps) when available:  
    performHapticFeedback(HapticFeedbackConstants.CONFIRM)  
}  
  
fun View.hapticReject(context: Context) {  
    performHapticFeedback(HapticFeedbackConstants.REJECT)  
  
    // If you need explicit vibration patterns (rare), use Vibrator/  
    VibrationEffect.  
    val vibrator = context.getSystemService(Vibrator::class.java)  
    vibrator?.vibrate(VibrationEffect.createOneShot(40L,  
        VibrationEffect.DEFAULT_AMPLITUDE))  
}
```

(Android provides `HapticFeedbackConstants` and `VibrationEffect` APIs.) 30

Checklist (haptics)

- [] Every haptic has a **clear semantic meaning** (impact vs selection vs result). 31
- [] Haptics are NOT used for frequent interactions (scrolling lists, text cursor moves). 32
- [] `prepare()` used only when timing matters (e.g., just before the interaction). 33
- [] Always paired with a visible state change (color, text, icon, animation). 34
- [] Tested with vibration disabled / device in silent modes. 35

Anti-patterns (haptics)

Avoid using haptics as “decoration”; firing multiple haptics in rapid succession; using success/warning/error haptics for neutral navigation; using `selectionChanged()` on “Confirm” button taps (Apple explicitly says not to). 36

Official sources (haptics)

Apple feedback generator docs (impact/notification/selection + `prepare()`) 37

Android vibration & haptic APIs (`VibrationEffect` / `HapticFeedbackConstants`) 38

Animations and motion

Pattern	Rule	Value iOS	Value Android
Duration scale (tokenized)	Use a small, consistent set of durations; don't invent per-screen timings	Core Animation default behavior uses transaction duration or 0.25s if not set (baseline)	Material 3 motion tokens (ms): Short1=50, Short2=100, Short3=150, Short4=200; Medium1=250, Medium2=300, Medium3=350, Medium4=400; Long1=450, Long2=500, Long3=550, Long4=600; ExtraLong1=700... ExtraLong4=1000
iOS practical mapping (recommended)	Map common UI moments to a measurable "house scale" aligned to iOS baseline	Micro-feedback: 0.20–0.25s ; Standard transitions: 0.30–0.35s ; Large/interruptible: 0.45–0.60s	Prefer token tiers: 100–200ms micro, 250–350ms standard, 450–600ms large
Easing	Keep easing consistent; use platform curves instead of random bezier	Prefer system/standard curves; avoid excessive motion in frequent interactions	M3 easing tokens (cubic-bezier): Standard = (0.2,0.0,0.0,1.0); StandardDecelerate=(0,0,0,1); EmphasizedDecelerate=(0.05,0.7,0,1) etc.
Elevation & motion consistency	Elevation changes should animate with the same scale and easing across app	Keep subtle; avoid "popping" elevation/shadows	Use tonal + shadow elevation; base levels: 0/1/3/6/8/12

Pattern	Rule	Value iOS	Value Android
"Reduce motion" support	Allow reduced motion; do not force motion if user opted out	<code>UIAccessibility.isReduceMotionEnabled</code> + <code>reduceMotionStatusDidChangeNotification</code>	Respect system-wide "animators disabled" (e.g., duration scale 0); g non-essential motion
WCAG guidance for interaction-triggered motion	Motion triggered by interaction should be disable-able unless essential	Provide "Reduce motion" compliant alternative	Same

iOS snippet (Swift) — reduced motion gating + baseline durations

```
import UIKit

struct Motion {
    /// Practical baseline aligned with Core Animation defaults (0.25s).
    static let micro: TimeInterval = 0.25
    static let standard: TimeInterval = 0.35
    static let large: TimeInterval = 0.50
}

func animateIfAllowed(_ animations: @escaping () -> Void, completion: ((Bool) -> Void)? = nil) {
    if UIAccessibility.isReduceMotionEnabled {
        // Reduced motion: apply end state without animation.
        UIView.performWithoutAnimation {
            animations()
            completion?(true)
        }
    } else {
        UIView.animate(withDuration: Motion.standard, animations: animations, completion: completion)
    }
}
```

(iOS exposes Reduce Motion state; Core Animation transaction defaults to ~0.25s if not set, which is a sensible baseline for "micro" feedback.) ⁴³

Android snippet (Compose) — tokenized duration + easing

```
@Composable
fun AnimatedVisibilityTokenized(visible: Boolean) {
    val alpha by animateFloatAsState(
        targetValue = if (visible) 1f else 0f,
        animationSpec = tween(
            durationMillis = MotionTokens.DurationMedium2.toInt(), // 300ms
            easing = MotionTokens.EasingStandardCubicBezier
        ),
        label = "alpha"
    )
    Box(Modifier.alpha(alpha)) { /* content */ }
}
```

(Material 3 token durations and easing curves are explicit in `MotionTokens`.) 44

Checklist (motion)

- [] App uses a small, documented duration scale (token tiers) rather than per-feature timings. 45
- [] Easing is standardized (standard vs emphasized; no random cubic-beziers). 44
- [] All “frequent interactions” avoid extra motion beyond platform defaults. 32
- [] Reduced motion is respected; non-essential animation can be disabled. 46
- [] Elevation transitions use consistent dp scale (Android) and subtle hierarchy cues (iOS). 40

Anti-patterns (motion)

Avoid 700ms+ animations for basic navigation; stacking multiple animations (opacity+scale+blur) for every interaction; making animations essential to use a feature; ignoring Reduce Motion; and creating motion that triggers vestibular discomfort without an off switch. 47

Official sources (motion)

Core Animation duration behavior: transaction duration or **0.25s** default (archival Apple documentation)

48

Apple HIG Motion (avoid extra motion on frequent interactions) 32

Android/Compose Material 3 MotionTokens (durations + easing) 44

iOS Reduce Motion APIs 49

WCAG guidance on interaction-triggered animation 42

Keyboard and form UX

Keyboard handling

Pattern	Rule	Value iOS	Value Android	Source
Layout avoids keyboard	Use system-provided keyboard layout guides / insets	<code>keyboardLayoutGuide</code> / <code>UIKeyboardLayoutGuide</code> for constraints	<code>WindowInsets</code> and IME insets; in Compose use <code>WindowInsets.ime</code>	50
Don't hardcode keyboard height	Derive from system metrics/ notifications	Observe keyboard frame changes if needed; prefer layout guides	Use insets APIs, not "estimated keyboard dp"	51
Scroll active field into view	Ensure focused field is visible above keyboard	Use scroll + focus state; consider <code>UIScrollView</code> content inset via keyboard guide	Use <code>imePadding()</code> / inset-aware scrolling; <code>bringIntoView</code> patterns	52
Dismiss keyboard	Provide predictable dismissal	Tap outside / scroll / explicit UI action; keep consistent	Same, but avoid stealing focus unexpectedly	(Platform guidance favors consistent behaviors and system patterns.) 53

iOS snippet (SwiftUI) — keyboard-safe footer input

```
struct ChatComposer: View {
    @State private var text = ""
    var body: some View {
        VStack(spacing: 0) {
            ScrollView { /* messages */ }

            Divider()

            HStack {
                TextField("Message", text: $text)
                    .textFieldStyle(.roundedBorder)
                Button("Send") { /* send */ }
            }
        }
    }
}
```

```

        .padding(12)
        .background(Color(UIColor.secondarySystemBackground))
    }
    // iOS 15+ provides keyboard layout guides; in SwiftUI prefer safe area
    keyboard handling.
    }
}

```

(Use semantic backgrounds and avoid manual keyboard height constants; on UIKit you'd anchor to `keyboardLayoutGuide`.) ⁵⁴

Android snippet (Compose) — IME padding

```

@Composable
fun ChatComposerScreen() {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .windowInsetsPadding(WindowInsets.ime) // pushes content above
        keyboard
    ) {
        // Messages...
        Spacer(Modifier.weight(1f))
        // Composer...
    }
}

```

(Compose IME insets are the recommended mechanism.) ⁵⁵

Checklist (keyboard)

- [] No hardcoded keyboard heights; only system guides/insets. ⁵¹
- [] Focused field is never obscured (scrolls into view). ⁵²
- [] Return key behavior set to match flow (Next/Done). (See forms section for mapping to content types.) ⁵⁶
- [] Dismiss behavior consistent and non-surprising. ³²

Anti-patterns (keyboard)

Avoid “keyboard avoidance” with magic numbers; jumping content that causes layout jitter; auto-dismissing keyboard while user is typing; trapping focus with no way to dismiss. ⁵³

Official sources (keyboard)

iOS keyboard layout guide docs ⁵⁷

Android IME insets guidance ⁵⁵

Forms mobile best practices

Pattern	Rule	Value iOS	Value Android	Source
Autofill / semantic meaning	Declare field meaning so OS can autofill and show correct keyboard	<code>UITextContentType</code> (e.g., <code>emailAddress</code> , <code>password</code> , <code>oneTimeCode</code>)	Autofill framework + <code>autofillHints</code> (and Compose autofill integration)	58
One-time codes	Use dedicated content type/hint for OTP	<code>UITextContentType.oneTimeCode</code>	Use OTP hints/ autofill where applicable	59
Error messaging (supporting text)	Place errors adjacent to input; keep linked to field	Use accessible label + error text; color not the only signal	Material text fields define visible states and support error state patterns	60
Validation timing	Validate "as early as helpful, as late as necessary"	Prefer inline after user interaction (e.g., on submit or after leaving field)	Same	(Material field states emphasize visible state; avoid disruptive early errors.) 60
Keyboard type	Use native keyboard types per field	Use keyboard types + <code>textContentType</code> (email, phone, number, URL)	Use <code>inputType</code> / Compose <code>KeyboardOptions</code> (e-mail, number, phone)	56

iOS snippet (SwiftUI) — autofill and OTP

```
struct LoginForm: View {
    @State private var email = ""
    @State private var password = ""
    @State private var otp = ""

    var body: some View {
        Form {
            TextField("Email", text: $email)
                .textInputAutocapitalization(.never)
                .keyboardType(.emailAddress)
                .textContentType(.emailAddress)

            SecureField("Password", text: $password)
                .textContentType(.password)

            TextField("One-time code", text: $otp)
                .keyboardType(.numberPad)
                .textContentType(.oneTimeCode)
        }
    }
}
```

(`UITextContentType` exposes semantic meaning including `oneTimeCode`.) 61

Android snippet (Compose) — keyboard options + autofill intent

```
@Composable
fun LoginForm() {
    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }

    OutlinedTextField(
        value = email,
        onValueChange = { email = it },
        label = { Text("Email") },
        keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Email,
            imeAction = ImeAction.Next
        )
    )

    OutlinedTextField(
        value = password,
        onValueChange = { password = it },
```

```
        label = { Text("Password") },
        visualTransformation = PasswordVisualTransformation(),
        keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Password,
            imeAction = ImeAction.Done
        )
    )
}
```

(Android autofill guidance focuses on declaring field meaning; Compose text fields support IME actions and keyboard types.) 62

Checklist (forms)

- [] Every input declares its semantic meaning (email/password/OTP) to enable autofill. 56
- [] Error states are adjacent to the field and not “color-only.” 63
- [] IME/Return actions match flow (Next/Done). 64
- [] Validation is not aggressively “red while typing” for normal latency networks/user flows. 65

Anti-patterns (forms)

Avoid turning every field red on each keystroke; using placeholder as label only; preventing paste for OTP; masking/formatting that blocks selection/cursor movement; hiding errors at the top of the page away from the field. 66

Official sources (forms)

iOS: `UITextContentType` (including `oneTimeCode`) 61
Android: autofill framework optimization guidance 64
Material text field states guidance 60

Authentication and permissions

Biometrics and authentication

Pattern	Rule	Value iOS	Value Android
System prompt only	Always use the system-managed prompt (trust + consistency)	<code>LAContext.evaluatePolicy</code> with <code>deviceOwnerAuthenticationWithBiometrics</code> (or <code>deviceOwnerAuthentication</code>)	Use <code>BiometricPrompt</code> system

Pattern	Rule	Value iOS	Value Android
Fallback credential	Always provide fallback to device credential where appropriate	Use <code>LAPolicy.deviceOwnerAuthentication</code> (biometry + passcode/Watch per policy)	Use <code>setAllowedAuthenticators(DEVICE_CREDENTIAL)</code> conce
Biometric strength	Don't treat all biometrics as equal for high-risk actions	Managed by iOS policy; app shouldn't "grade" Face ID vs Touch ID	Android defines auth classes (Strong, Convenience) and restricts APIs a
Secure storage	Store secrets/tokens in platform secure storage	Keychain access classes (e.g., <code>kSecAttrAccessibleWhenUnlockedThisDeviceOnly</code>)	Android Keystore for non-exporta with user auth if needed
UX rollout	Biometrics should be opt-in inside settings or after first successful login	Offer "Enable Face ID/Touch ID" after value is clear	Offer toggle after login; don't blo biometric isn't available

iOS snippet (Swift) — Face ID/Touch ID with passcode fallback

```
import LocalAuthentication

func authenticate(reason: String, completion: @escaping (Bool) -> Void) {
    let ctx = LAContext()
    var error: NSError?

    // Use deviceOwnerAuthentication for biometry + device passcode fallback.
    guard ctx.canEvaluatePolicy(.deviceOwnerAuthentication, error: &error) else
    {
        completion(false)
        return
    }
}
```

```

        ctx.evaluatePolicy(.deviceOwnerAuthentication, localizedReason: reason) {
            success, _ in
                DispatchQueue.main.async { completion(success) }
        }
    }
}

```

(`LAPolicy.deviceOwnerAuthentication` supports biometry or device credential depending on conditions.) ⁷²

Android snippet (Kotlin) — BiometricPrompt with device credential fallback

```

val promptInfo = BiometricPrompt.PromptInfo.Builder()
    .setTitle("Sign in")
    .setSubtitle("Use biometrics or your device PIN")
    .setAllowedAuthenticators(
        BiometricManager.Authenticators.BIOMETRIC_STRONG or
        BiometricManager.Authenticators.DEVICE_CREDENTIAL
    )
    .build()

biometricPrompt.authenticate(promptInfo)

```

(Android documentation recommends the Biometric library and system prompt; allowed authenticators are declared via constants.) ⁷³

Checklist (biometrics)

- [] Use only system prompts (no “fake Face ID UI”). ⁷¹
- [] Provide device credential fallback when appropriate. ⁷⁴
- [] High-value actions require strong authentication (Android: `BIOMETRIC_STRONG`). ⁷⁵
- [] Secrets stored in Keychain/Keystore, not plain prefs. ⁷⁰

Anti-patterns (biometrics)

Avoid forcing biometrics at first launch; blocking login behind biometric enrollment; storing long-lived session tokens outside Keychain/Keystore; implementing arbitrary “3 tries” counters instead of relying on system lockout and fallbacks. ⁷⁶

Official sources (biometrics)

Apple LocalAuthentication policies ⁷⁷

Keychain access class example (`WhenUnlockedThisDeviceOnly`) ⁷⁸

Android biometric sign-in guidance ⁷⁹

Android biometric authenticators constants ⁸⁰

Android Keystore system overview ⁸¹

Permissions strategy

Pattern	Rule	Value iOS	Value Android	Source
Ask “in context”	Request permission right before the user needs the capability	Align prompt with user action (camera → scan, location → nearby)	Same; request only when feature is invoked	Android permission UX explicitly supports educational UI before requesting
Educational UI (“priming”)	Explain why before system prompt when risk of denial is high	Use a short in-app explanation screen; then system prompt	Use “educational UI” / rationale patterns before requesting	82
Rationale logic	Don’t always show rationale—use platform signal	Detect denied state and route to Settings	Use <code>shouldShowRequestPermissionRationale()</code> to decide whether to show educational UI	82
“Don’t ask again”	Detect and provide Settings route	iOS: user must manually re-enable in Settings once denied	Android: if denied and rationale returns false after prior request, treat as “don’t ask again” case and guide to Settings	82

iOS snippet (Swift) — open app settings after denial

```
import UIKit

func openAppSettings() {
    guard let url = URL(string: UIApplication.openSettingsURLString) else {
        return
    }
    UIApplication.shared.open(url)
}
```

(Use for guiding users after denial; complements in-context request strategy.) 82

Android snippet (Kotlin) — rationale + request flow

```
if (shouldShowRequestPermissionRationale(Manifest.permission.CAMERA)) {  
    // Show an educational UI explaining why, then request.  
}  
requestPermissions(arrayOf(Manifest.permission.CAMERA), REQ_CAMERA)
```

(Official Android guidance explicitly references educational UI and shouldShowRequestPermissionRationale().) 82

Checklist (permissions)

- [] No permission prompts at cold start unless the app cannot function without it (rare). 82
- [] Each permission has an educational UI tied to user intent when denial risk is high. 82
- [] Denial state has a graceful degraded mode (read-only, manual entry, etc.). 82
- [] Settings fallback is available when user has blocked prompts. 82

Anti-patterns (permissions)

Avoid requesting multiple permissions in a row; asking for location when user taps “Sign up”; blocking the UI behind an un-dismissable permission wall; repeating a denied prompt without explaining. 82

Official sources (permissions)

Android runtime permissions request + educational UI + rationale signal 82

Offline, empty states, and launch experiences

Offline mode and sync

Pattern	Rule	Value iOS	Value Android
Connectivity detection	Monitor connectivity changes via system APIs	NWPathMonitor	ConnectivityManager.registerDefaultNetworkCallback() / NetworkCallback
Offline-first baseline	Reads must work offline for “offline-first”	Use local cache/store as source for reads	Android offline-first guidance: local data source as canonical source of truth

Pattern	Rule	Value iOS	Value Android
Queue writes	Queue writes when offline; drain later	Implement local queue; sync later	Android guidance: queued writes + WorkManager; describe strategies
Sync strategies	Choose pull vs push vs hybrid based on domain	Prefer explicit refresh or opportunistic sync	Android offline-first doc: pull-based vs push-based, hybrid; conflict resolution section
HTTP caching	Use platform HTTP cache where it fits	<code>URLCache</code> for URL loading with caching policy	Use HTTP cache + local DB (Room/Datastore)

iOS snippet (Swift) — connectivity monitor

```
import Network

final class Connectivity {
    private let monitor = NWPathMonitor()
    private let queue = DispatchQueue(label: "ConnectivityMonitor")

    var onChange: ((Bool) -> Void)?

    func start() {
        monitor.pathUpdateHandler = { [weak self] path in
            self?.onChange?(path.status == .satisfied)
        }
        monitor.start(queue: queue)
    }

    func stop() { monitor.cancel() }
}
```

(`NWPathMonitor` is the recommended API to observe network path changes.) 86

Android snippet (Kotlin) — default network callback

```
val cm = context.getSystemService(ConnectivityManager::class.java)

cm.registerDefaultNetworkCallback(object :
```

```

ConnectivityManager.NetworkCallback() {
    override fun onAvailable(network: Network) { /* online */ }
    override fun onLost(network: Network) { /* offline */ }
})

```

(Android guidance: use `NetworkCallback` with `registerDefaultNetworkCallback` to listen to network events.) ⁸⁷

Checklist (offline & sync)

- [] App remains usable offline for “read” paths (lists, cached content). ⁸⁴
- [] UI clearly indicates offline/online state (banner, icon, disabled actions). ⁸⁸
- [] Writes either: online-only (block when offline), queued, or lazy-write—chosen per domain. ⁸⁴
- [] Sync has backoff + retry policy; no infinite tight retry loops. ⁸⁴
- [] Cache strategy documented (what is cached, invalidation, TTL). ⁸⁵

Anti-patterns (offline)

Avoid showing empty UI that looks like “no data” when it’s actually “offline”; overwriting local changes after reconnect; retrying aggressively on 401/403; making the UI wait for first network call before showing cached data (explicitly discouraged by offline-first guidance). ⁸⁴

Official sources (offline)

iOS `NWPathMonitor` ⁸⁶

Android “Read network state” / callbacks ⁸⁷

Android offline-first architecture guide (sync, conflict resolution, queues, local source of truth) ⁸⁴

iOS `URLCache` ⁸⁹

Splash and launch screens

Pattern	Rule	Value iOS	Value Android	Source
Use system launch mechanism	Launch screen is not a “marketing splash”	Use storyboard launch screens; legacy <code>UILaunchImages</code> is deprecated	Android 12+ SplashScreen API	⁹⁰
Purpose	Launch screen’s job is perceived speed and readiness	“Sole function... enhance perception... quick to launch”	Splash shows during cold/warm start; dismissed when app draws first frame	⁹¹
Animation limits	Keep icon animation short	(iOS: avoid prolonged fake loading; no official ms cap stated in cited sources)	Recommended icon animation \leq 1000 ms ; delayed start \leq 166 ms	⁹²

Pattern	Rule	Value iOS	Value Android	Source
Measurable icon specs	Follow platform spec for dimensions	(Use storyboard constraints; focus on first UI parity rather than separate art)	Branding image 200×80 dp ; icon w/ bg 240×240 dp in 160dp circle; icon w/o bg 288×288 dp in 192dp circle	⁹²

iOS snippet (Info.plist / Xcode concept)

Use Xcode's launch screen storyboard configuration; do not rely on deprecated launch images (`UILaunchImages`). ⁹³

Android snippet (Kotlin) — install SplashScreen

```
override fun onCreate(savedInstanceState: Bundle?) {
    val splashScreen = installSplashScreen()
    super.onCreate(savedInstanceState)
    setContentView(R.layout.main_activity)
}
```

(Android SplashScreen API describes lifecycle + customization.) ⁹²

Checklist (launch/splash)

- [] No “fake loading spinner” on top of launch screen unless absolutely necessary. ⁹¹
- [] Branding does not delay app readiness; launch transitions into real first UI fast. ⁹¹
- [] Android icon animation $\leq 1000\text{ms}$; delayed start $\leq 166\text{ms}$. ⁹²
- [] iOS uses storyboard launch screen; deprecated launch images removed. ⁹³

Anti-patterns (launch/splash)

Avoid long logo movies; marketing copy on launch screen; blocking first frame while doing network calls that could be deferred; adding extra spinners on top of Android 12 splash (Android guidance warns this can feel jarring). ⁹⁴

Official sources (launch/splash)

iOS: `UILaunchImages` deprecation (use launch storyboard) ⁹⁵

Apple HIG Launching (purpose statement) ⁹⁶

Android SplashScreen API + specs + timing recommendations (updated 2026-02-06) ⁹²

Xcode guide for specifying launch screen ⁹⁷

Empty states

Pattern	Rule	Value iOS	Value Android	Source
Structure	Empty state should educate + guide next step	Apple guidance: empty state can welcome + educate; keep copy helpful	Material guidance: basic empty state includes image + tagline; keep tone appropriate; provide actions	98
Types	Differentiate: first use vs no results vs error vs offline	Tailor message and CTA to the cause	Same	99
CTA	Provide a primary action when user can resolve it	iOS: clear action labeled with verb	Android: use primary button when resolution exists; otherwise provide secondary/help	(Empty state guidance emphasizes guidance and next action.) 98

iOS snippet (SwiftUI) — empty state with CTA

```
struct EmptyStateView: View {
    let title: String
    let message: String
    let actionTitle: String
    let action: () -> Void

    var body: some View {
        VStack(spacing: 12) {
            Image(systemName: "tray")
                .font(.system(size: 48))
                .foregroundColor(Color(UIColor.secondaryLabel))

            Text(title).font(.headline)
            Text(message)
                .font(.body)
                .foregroundColor(Color(UIColor.secondaryLabel))
                .multilineTextAlignment(.center)

            Button(actionTitle, action: action)
                .buttonStyle(.borderedProminent)
        }
        .padding(24)
    }
}
```

(iOS semantic label colors support hierarchy in empty states.) 100

Android snippet (Compose) — empty list state

```
@Composable
fun EmptyState(
    title: String,
    message: String,
    onAction: (() -> Unit)? = null
) {
    Column(
        Modifier.fillMaxSize().padding(24.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Icon(Icons.Outlined.Inbox, contentDescription = null)
        Text(title, style = MaterialTheme.typography.titleMedium)
        Text(message, style = MaterialTheme.typography.bodyMedium)

        if (onAction != null) {
            Spacer(Modifier.height(12.dp))
            Button(onClick = onAction) { Text("Create") }
        }
    }
}
```

(Empty state guidance emphasizes a basic image + text; Wear guidance notes empty states aren't always errors.) 101

Checklist (empty states)

- [] Empty states explain **what's happening** and what will appear here. 102
- [] If user can fix it, provide a primary CTA; else provide help/learn more. 102
- [] Offline empty states are clearly labeled (not confused with “no results”). 103
- [] Tone fits context (first use vs error). 104

Anti-patterns (empty states)

Avoid blank screens; “No data” without explanation; CTAs that do nothing or lead to dead ends; using humor for error/offline states that need clarity; hiding the reason (filters, permissions, offline). 105

Official sources (empty states)

Apple HIG Writing (mentions empty states as an opportunity to welcome/educate) 106

Material empty states guidance (basic structure: image + tagline) 107

Android Wear empty state guidance (contextual framing) 108

Large screens: iPad, tablets, and foldables

Tablets and iPad

Pattern	Rule	Value iOS	Value Android	Source
Multitasking support	Support Split View / Slide Over; handle size class changes	iPad multitasking changes size classes; test Split View/Slide Over	Support multi-window; build adaptive layouts	109
Sidebar navigation	Use multi-column navigation patterns on large screens	<code>NavigationSplitView</code> / split views / sidebars guidance	Navigation rail recommended for larger window sizes; 3–7 top-level destinations	110
Navigation rail measurable width	Treat rail width as a fixed measurable token	(iOS: sidebar width is adaptive; use system components)	Navigation rail container width 80dp ; icon size 24dp ; active indicator 56×32dp	111
Pointer/trackpad	Support pointer interactions on iPad	Pointer interactions API for iPad input devices	(Android: pointer support varies; focus states must be clear on large screens)	112

iOS snippet (SwiftUI) — `NavigationSplitView` baseline

```
struct RootView: View {
    var body: some View {
        NavigationSplitView {
            List {
                NavigationLink("Inbox", value: "inbox")
                NavigationLink("Settings", value: "settings")
            }
        } detail: {
            Text("Select an item")
        }
    }
}
```

(Apple provides `NavigationSplitView` for 2–3 column navigation, aligning with sidebar patterns.) 113

Android snippet (Compose) — navigation rail token alignment

```
@Composable
fun TabletScaffold() {
    Row {
        NavigationRail(
            modifier = Modifier.width(80.dp) // tokenized
        ) {
            NavigationRailItem(
                selected = true,
                onClick = { },
                icon = { Icon(Icons.Default.Home, contentDescription =
"Home") },
                label = { Text("Home") }
            )
        }
        // Main content...
    }
}
```

(Compose tokens define rail width 80dp and indicator sizes.) 111

Checklist (tablets/iPad)

- [] Layout adapts to Split View/Slide Over sizes; no clipped sidebars or broken grids. 114
- [] Large-screen navigation uses split/sidebars (iPad) or rails (Android) rather than stretched bottom tabs. 115
- [] Navigation rail: 3–7 top-level destinations, consistent placement; width aligned to token (80dp). 116
- [] Pointer support on iPad: hover/highlight states are clear and not “touch-only.” 112

Anti-patterns (tablets)

Avoid phone UI simply scaled up; keeping a single-column list for everything; hiding primary navigation behind hamburger on large screens; ignoring pointer/keyboard input on iPad. 117

Official sources (tablets/iPad)

Apple iPad multitasking guide (Split View/Slide Over and size class implications) 118

Apple pointer interactions 112

Apple split views/sidebars and `NavigationSplitView` 119

Android large screens “User interface” guidance 120

Android navigation rail guidance 121

Compose Material 3 NavigationRailTokens (80dp width + measurable indicator sizes) 122

Foldables (Android)

Pattern	Rule	Value iOS	Value Android	Source
Postures	Support key postures: flat, tabletop, book	(iOS: foldables not a mainstream platform target)	Foldables: flat + half-open (tabletop/book) postures	123
Continuity	Preserve app state across posture changes	Maintain state on size changes; avoid restart-like experience	WindowManager provides display features; preserve UI state	124
Hinge awareness	Don't place critical controls under hinge/fold	N/A	Use Jetpack WindowManager to detect folding features and avoid hinge occlusion	124
Testing	Test multiple device configs and emulator profiles	Test multitasking size classes	Use foldable emulator + WindowManager samples/guides	125

Android snippet (Compose) — fold-aware layout (conceptual hook)

```
@Composable
fun FoldAwareScreen(windowInfoTracker: WindowInfoTracker) {
    // Observe FoldingFeature via Jetpack WindowManager (see official guide).
    // Then switch between one-pane vs two-pane or tabletop layouts.
}
```

(Official guidance: make your app fold-aware using Jetpack WindowManager.) [124](#)

Checklist (foldables)

- [] App handles posture changes without losing state. [124](#)
- [] Layout avoids hinge area; critical content not hidden. [125](#)
- [] Uses canonical adaptive layouts (list-detail, supporting pane) when screen expands. [126](#)
- [] Tested on emulator + at least one real foldable if shipping to that segment. [125](#)

Anti-patterns (foldables)

Avoid forcing a single phone layout in all postures; resetting navigation on unfold; placing FAB or primary CTA right at the hinge; ignoring tabletop/book opportunities where UI can split naturally. [123](#)

Official sources (foldables)

Learn about foldables (postures) [127](#)

Make your app fold-aware (Jetpack WindowManager) [125](#)

- 1 2 10 13 19 20 <https://developer.apple.com/design/human-interface-guidelines/color>
<https://developer.apple.com/design/human-interface-guidelines/color>
- 3 48 https://leopard-adc.pepas.com/documentation/Cocoa/Conceptual/Animation_Types_Timing/Articles/Timing.html
https://leopard-adc.pepas.com/documentation/Cocoa/Conceptual/Animation_Types_Timing/Articles/Timing.html
- 4 54 <https://developer.apple.com/documentation/uikit/uicolor/systembackground>
<https://developer.apple.com/documentation/uikit/uicolor/systembackground>
- 5 12 15 21 <https://developer.apple.com/documentation/UIKit/UIColor/label>
<https://developer.apple.com/documentation/UIKit/UIColor/label>
- 6 <https://developer.apple.com/documentation/UIKit/UIColor/separator>
<https://developer.apple.com/documentation/UIKit/UIColor/separator>
- 7 https://developer.apple.com/documentation/uikit/uicolor/systemfill?changes=_2
https://developer.apple.com/documentation/uikit/uicolor/systemfill?changes=_2
- 8 34 63 <https://www.w3.org/WAI/WCAG22/Understanding/contrast-minimum.html>
<https://www.w3.org/WAI/WCAG22/Understanding/contrast-minimum.html>
- 9 14 <https://www.w3.org/TR/WCAG22/>
<https://www.w3.org/TR/WCAG22/>
- 11 16 40 <https://android.googlesource.com/platform/frameworks/support/%2B/HEAD/compose/material3/material3/src/commonMain/kotlin/androidx/compose/material3/tokens/ElevationTokens.kt>
<https://android.googlesource.com/platform/frameworks/support/%2B/HEAD/compose/material3/material3/src/commonMain/kotlin/androidx/compose/material3/tokens/ElevationTokens.kt>
- 17 <https://developer.android.com/develop/ui/compose/designsystems/material3>
<https://developer.android.com/develop/ui/compose/designsystems/material3>
- 18 <https://m3.material.io/foundations/interaction/states>
<https://m3.material.io/foundations/interaction/states>
- 22 <https://android.googlesource.com/platform/frameworks/support/%2B/c06c17769261251cc37627b07577dced5344298a/compose/material3/material3/src/commonMain/kotlin/androidx/compose/material3/tokens/ColorSchemeKeyTokens.kt>
<https://android.googlesource.com/platform/frameworks/support/%2B/c06c17769261251cc37627b07577dced5344298a/compose/material3/material3/src/commonMain/kotlin/androidx/compose/material3/tokens/ColorSchemeKeyTokens.kt>
- 23 <https://developer.apple.com/documentation/uikit/uiimpactfeedbackgenerator/feedbackstyle/heavy>
<https://developer.apple.com/documentation/uikit/uiimpactfeedbackgenerator/feedbackstyle/heavy>
- 24 <https://developer.apple.com/documentation/uikit/uinotificationfeedbackgenerator/feedbacktype/success>
<https://developer.apple.com/documentation/uikit/uinotificationfeedbackgenerator/feedbacktype/success>
- 25 29 31 36 <https://developer.apple.com/documentation/uikit/uiselectionfeedbackgenerator/selectionchanged%28%29>
<https://developer.apple.com/documentation/uikit/uiselectionfeedbackgenerator/selectionchanged%28%29>
- 26 33 37 <https://developer.apple.com/documentation/uikit/uifeedbackgenerator/prepare%28%29>
<https://developer.apple.com/documentation/uikit/uifeedbackgenerator/prepare%28%29>

27 28 32 39 <https://developer.apple.com/design/human-interface-guidelines/motion>
<https://developer.apple.com/design/human-interface-guidelines/motion>

30 35 <https://developer.android.com/develop/ui/views/haptics/haptic-feedback>
<https://developer.android.com/develop/ui/views/haptics/haptic-feedback>

38 <https://developer.apple.com/documentation/uikit/uinotificationfeedbackgenerator>
<https://developer.apple.com/documentation/uikit/uinotificationfeedbackgenerator>

41 43 46 49 <https://developer.apple.com/documentation/UIKit/UIAccessibility/isReduceMotionEnabled>
<https://developer.apple.com/documentation/UIKit/UIAccessibility/isReduceMotionEnabled>

42 47 <https://www.w3.org/WAI/WCAG22/Understanding/animation-from-interactions.html>
<https://www.w3.org/WAI/WCAG22/Understanding/animation-from-interactions.html>

44 45 <https://android.googlesource.com/platform/frameworks/support/%2Bc06c17769261251cc37627b07577dced5344298a/compose/material3/material3/src/commonMain/kotlin/androidx/compose/material3/tokens/MotionTokens.kt>
<https://android.googlesource.com/platform/frameworks/support/%2Bc06c17769261251cc37627b07577dced5344298a/compose/material3/material3/src/commonMain/kotlin/androidx/compose/material3/tokens/MotionTokens.kt>

50 52 57 <https://developer.apple.com/documentation/uikit/uiview/keyboardlayoutguide>
<https://developer.apple.com/documentation/uikit/uiview/keyboardlayoutguide>

51 <https://developer.apple.com/documentation/uikit/uikeyboardlayoutguide>
<https://developer.apple.com/documentation/uikit/uikeyboardlayoutguide>

53 55 <https://developer.android.com/develop/ui/compose/system/insets>
<https://developer.android.com/develop/ui/compose/system/insets>

56 58 61 <https://developer.apple.com/documentation/uikit/uitextcontenttype>
<https://developer.apple.com/documentation/uikit/uitextcontenttype>

59 <https://developer.apple.com/documentation/uikit/uitextcontenttype/onetimecode>
<https://developer.apple.com/documentation/uikit/uitextcontenttype/onetimecode>

60 65 <https://m3.material.io/components/text-fields/overview>
<https://m3.material.io/components/text-fields/overview>

62 64 <https://developer.android.com/identity/autofill/autofill-optimize>
<https://developer.android.com/identity/autofill/autofill-optimize>

66 <https://m3.material.io/components/text-fields/accessibility>
<https://m3.material.io/components/text-fields/accessibility>

67 77 <https://developer.apple.com/documentation/LocalAuthentication/LAPolicy/deviceOwnerAuthenticationWithBiometrics>
<https://developer.apple.com/documentation/LocalAuthentication/LAPolicy/deviceOwnerAuthenticationWithBiometrics>

68 72 74 <https://developer.apple.com/documentation/LocalAuthentication/LAPolicy/deviceOwnerAuthentication>
<https://developer.apple.com/documentation/LocalAuthentication/LAPolicy/deviceOwnerAuthentication>

69 75 <https://source.android.com/docs/security/features/biometric>
<https://source.android.com/docs/security/features/biometric>

70 78 <https://developer.apple.com/documentation/security/ksecattraccessiblewhenunlockedthisdeviceonly>
<https://developer.apple.com/documentation/security/ksecattraccessiblewhenunlockedthisdeviceonly>

71 73 76 79 <https://developer.android.com/identity/sign-in/biometric-auth>
<https://developer.android.com/identity/sign-in/biometric-auth>

80 <https://developer.android.com/reference/android/hardware/biometrics/BiometricManager.Authenticators>
<https://developer.android.com/reference/android/hardware/biometrics/BiometricManager.Authenticators>

81 <https://developer.android.com/privacy-and-security/keystore>
<https://developer.android.com/privacy-and-security/keystore>

82 <https://developer.android.com/training/permissions/requesting>
<https://developer.android.com/training/permissions/requesting>

83 86 <https://developer.apple.com/documentation/network/nwpathmonitor>
<https://developer.apple.com/documentation/network/nwpathmonitor>

84 88 103 <https://developer.android.com/topic/architecture/data-layer/offline-first>
<https://developer.android.com/topic/architecture/data-layer/offline-first>

85 89 <https://developer.apple.com/documentation/foundation/urlcache>
<https://developer.apple.com/documentation/foundation/urlcache>

87 <https://developer.android.com/develop/connectivity/network-ops/reading-network-state>
<https://developer.android.com/develop/connectivity/network-ops/reading-network-state>

90 93 95 <https://developer.apple.com/documentation/bundleresources/information-property-list/uilaunchimages>
<https://developer.apple.com/documentation/bundleresources/information-property-list/uilaunchimages>

91 96 <https://developer.apple.com/design/human-interface-guidelines/launching>
<https://developer.apple.com/design/human-interface-guidelines/launching>

92 94 <https://developer.android.com/develop/ui/views/launch/splash-screen>
<https://developer.android.com/develop/ui/views/launch/splash-screen>

97 <https://developer.apple.com/documentation/xcode/specifying-your-apps-launch-screen>
<https://developer.apple.com/documentation/xcode/specifying-your-apps-launch-screen>

98 106 <https://developer.apple.com/design/human-interface-guidelines/writing>
<https://developer.apple.com/design/human-interface-guidelines/writing>

99 104 108 <https://developer.android.com/design/ui/wear/guides/surfaces/tiles/states>
<https://developer.android.com/design/ui/wear/guides/surfaces/tiles/states>

100 <https://developer.apple.com/documentation/uikit/uicolor/secondarylabel>
<https://developer.apple.com/documentation/uikit/uicolor/secondarylabel>

101 102 105 107 <https://m2.material.io/design/communication/empty-states.html>
<https://m2.material.io/design/communication/empty-states.html>

109 118 <https://developer.apple.com/library/archive/documentation/WindowsViews/Conceptual/AdoptingMultitaskingOniPad/index.html>
<https://developer.apple.com/library/archive/documentation/WindowsViews/Conceptual/AdoptingMultitaskingOniPad/index.html>

110 113 <https://developer.apple.com/documentation/SwiftUI/NavigationSplitView>

<https://developer.apple.com/documentation/SwiftUI/NavigationSplitView>

111 122 <https://android.googlesource.com/platform/frameworks/support/%2B/c06c17769261251cc37627b07577dced5344298a/compose/material3/material3/src/commonMain/kotlin/androidx/compose/material3/tokens/NavigationRailTokens.kt>

<https://android.googlesource.com/platform/frameworks/support/%2B/c06c17769261251cc37627b07577dced5344298a/compose/material3/material3/src/commonMain/kotlin/androidx/compose/material3/tokens/NavigationRailTokens.kt>

112 <https://developer.apple.com/documentation/uikit/pointer-interactions>

<https://developer.apple.com/documentation/uikit/pointer-interactions>

114 <https://developer.apple.com/library/archive/documentation/WindowsViews/Conceptual/AdoptingMultitaskingOniPad/QuickStartForSlideOverAndSplitView.html>

<https://developer.apple.com/library/archive/documentation/WindowsViews/Conceptual/AdoptingMultitaskingOniPad/QuickStartForSlideOverAndSplitView.html>

115 <https://developer.apple.com/design/human-interface-guidelines/sidebars>

<https://developer.apple.com/design/human-interface-guidelines/sidebars>

116 121 <https://developer.android.com/develop/ui/compose/components/navigation-rail>

<https://developer.android.com/develop/ui/compose/components/navigation-rail>

117 120 126 <https://developer.android.com/guide/topics/large-screens/user-interface>

<https://developer.android.com/guide/topics/large-screens/user-interface>

119 <https://developer.apple.com/design/human-interface-guidelines/split-views>

<https://developer.apple.com/design/human-interface-guidelines/split-views>

123 127 <https://developer.android.com/develop/ui/compose/layouts/adaptive/foldables/learn-about-foldables>

<https://developer.android.com/develop/ui/compose/layouts/adaptive/foldables/learn-about-foldables>

124 125 <https://developer.android.com/develop/ui/compose/layouts/adaptive/foldables/make-your-app-fold-aware>

<https://developer.android.com/develop/ui/compose/layouts/adaptive/foldables/make-your-app-fold-aware>