TAHAI Mobile Mission Control

Android / Google Play first - iOS / App Store after Google Play launch

Project-source pass plan for building a best-in-class mobile Mission Deck, Evidence Pack, OpsTools, secure handoff, and tablet Mission Control system.

Prepared: May 1, 2026

Canonical product split: Desktop TAHAI is the IT/DevOps command browser. Mobile TAHAI is the field/operator companion. Phones use Mission Deck. Tablets can graduate to Mission Control.

This PDF is intended for project sources and future coding chats. It preserves the existing Mission Tabs / Mission Control guardrails, then extends them into a native Android-first mobile roadmap with an iOS migration path after Google Play launch.

1. Non-negotiable mobile product thesis

TAHAI Mobile should not attempt to be a smaller desktop Chromium shell. The winning mobile product is a secure operator companion that lets users start missions, follow runbooks, capture evidence, run field tools, export sanitized handoffs, and later sync authorized records through IT Docs.

- Phone UX: Mission Deck - one active role card at a time, swipeable panes, bottom Mission Dock, capture-first workflow.
- Tablet UX: Mission Control - 1-Up, 2-Up, 3-Up, Quad, Focus, runbook rail, evidence rail, keyboard support, and external-display posture where platform support allows.
- Browser lane: Android uses Custom Tabs for external provider consoles and WebView only for first-party/controlled surfaces. iOS uses SwiftUI plus WKWebView/Safari-compatible auth flows unless a later EU-only entitlement path is deliberately pursued.
- Security lane: no PSA tokens, cloud-provider tokens, OAuth refresh tokens, raw cookies, Authorization headers, or copied session material inside mission state or evidence metadata.
- Store lane: Android ships first through Google Play. iOS begins after Android internal/closed/production readiness proves the product and policy posture.

1.1 Hard inheritance from desktop Mission Control guardrails

The existing desktop project source defines the unique system as TAHAI Mission Control = Mission Tabs + Mission Views + Mission Tools + Mission Evidence. It also sets the security boundary: implement browser-side UX, local persistence, validated links, export, redaction, and IT Docs/PSA reference contracts, but do not implement IT Docs backend features, PSA connectors, or secret storage in the browser/mobile client.

- Mobile mission files are untrusted local input. They require schema validation, size limits, URL protocol validation, enum validation, and safe migrations.
- IT Docs references are opaque display references only; server-side IT Docs authorization remains the authority for org/project/runbook/evidence access.
- PSA references are display/deep-link metadata only; all PSA writeback must route through IT Docs-authorized server-side connectors.
- Evidence export must run redaction preview before sync/export and must never include raw cookies, auth headers, bearer tokens, or cloud/API secrets in automated metadata.
- Open-source posture means no secrets, generated runtime artifacts, customer data, or credential-bearing fixtures in source.

1.2 Mobile naming model

Surface	Canonical mobile meaning
Mission Home	Start/restore active missions, view recent evidence, launch recipes, see IT Docs/offline/security state.
Mission Deck	Phone-first replacement for desktop Quad View: swipeable role cards plus bottom Mission Dock.
Mission Dock	Thumb-first actions: Runbook, Evidence, Tools, Notes, Links, Export, Sync.
Evidence Pack	Captured URLs, screenshots, photos, scans, notes, tool results, timestamps, hashes, and redaction status.
OpsTools	Local or privacy-aware tools: DNS, TLS, headers, redirect chain, JSON/YAML, JWT warning decoder, CIDR, status checks.
Tablet Mission Control	Adaptive tablet layout with 1/2/3/4 panes, focus pane, role labels, and active-pane routing.

2. Mobile repo, architecture, and build posture

2.1 Recommended repo structure

Use a new mobile repo or top-level mobile workspace rather than mixing native Android/iOS with the Electron desktop browser source. Keep shared specifications portable and testable.

```
C:\dev\tahai-mobile-mission-control
  apps/
    android/                # Kotlin + Jetpack Compose Android app
    ios/                    # SwiftUI app added after Google Play launch
  shared/
    mission-schema/         # JSON Schema, fixtures, redaction fixtures, source-of-truth enums
    docs/                   # product/security/project-source docs
  tools/
    verify-mobile-sources.*  # repo hygiene, secrets, schemas, store policy gates
    make-android-artifacts.* # build AAB/APK + evidence bundle
  evidence/
    smoke/                  # generated evidence is excluded from commits; copied into release bundle only
  docs/
    android-play-release.md
    mobile-security-spec.md
    privacy-data-map.md
    app-store-ios-follow-on.md
```

Placeholder reminder: replace paths like C:\dev\tahai-mobile-mission-control with the actual repo path once created.

2.2 Android technical stack

- Language/UI: Kotlin + Jetpack Compose + Material 3, with a TAHAI design layer for neon/electric operator visuals without burying usability.
- Architecture: MVVM or unidirectional state flow, repository layer, local-first mission store, explicit sync adapters later.
- Storage: SQLCipher/Room or encrypted file-backed mission store using Android Keystore-backed keys; no secrets in exported mission JSON.
- Browser integration: Custom Tabs for external provider surfaces; WebView only for first-party controlled content or signed local documentation views with a strict allowlist.
- Security: Android Network Security Config with cleartext disabled in release, least permissions, no broad WebView JavaScript bridge, no logging of secrets, no analytics until privacy map is complete.
- Testing: unit tests, instrumentation tests, screenshot/golden tests for key UI, static source verifiers, release AAB proof, Play policy checklist evidence.

2.3 Store policy facts that shape the plan

- Google Play requires new apps to publish with Android App Bundles, not APK-only production submissions. The plan therefore builds an AAB release path from the beginning.
- Current Google Play target API documentation says new apps and updates must target Android 15 / API level 35 or higher. The release gate must re-check this before submission because Google updates target requirements annually.
- Google Play Data Safety requires developers to declare how the app collects and handles user data, including data handled by third-party SDKs; the plan therefore blocks telemetry/analytics until a data map exists.
- New personal Google Play developer accounts may need a closed test with at least 12 opted-in testers for 14 continuous days before applying for production access.
- Google Play policy rejects apps with limited functionality and apps whose primary purpose is merely WebView/affiliate traffic. TAHAI Mobile must ship substantial native mission/evidence/tool functionality, not just a web wrapper.
- Android vitals and technical quality affect app visibility; the release plan includes crash/ANR/battery gates before production launch.

3. Android-first pass plan

Execution rule: each implementation pass should return a full repo ZIP, a smoke-evidence folder, a short verification result, and the remaining pass count. No placeholder-only passes.

ANDROID A00 - Mobile SSOT and repo creation

Goal: Create the new mobile source lane and prevent drift from desktop Electron, TAHAI OS-SENTINEL, IT Docs backend, and PSA connector lanes.

Build:

- Create repo skeleton, README, LICENSE/NOTICE/TRADEMARKS carry-forward where appropriate, SECURITY.md, CONTRIBUTING.md, and mobile docs folder.
- Add docs/mobile-product-ssot.md, docs/mobile-security-spec.md, docs/android-play-release.md, docs/privacy-data-map.md.
- Add source-exclusion rules for build outputs, keystores, local evidence, profiles, captures, and test account data.

Acceptance:

- Repo opens cleanly and does not contain generated installers, keystores, .env files, tokens, or customer data.
- Android-first and iOS-later sequencing is explicitly documented.
- Desktop guardrails are referenced without importing Electron-specific implementation into mobile.

Verification / evidence:

- Run initial repo hygiene verifier.
- Produce evidence: repo tree, gitignore/exclusion proof, docs index.

Notes / guardrails:

- Do not put Google Play signing keys or upload keys in the repo.

ANDROID A01 - Canonical mobile mission schema

Goal: Establish a platform-neutral mission schema that Android implements first and iOS reuses later.

Build:

- Create shared/mission-schema/mission.schema.json with schemaVersion, missionId, name, missionType, mode, role cards, links, notes, timeline, evidence, export profile, and redaction status.
- Create allowed enums for missionType, mode, role, evidenceKind, exportProfile, linkType, syncState.
- Create safe fixtures and malicious fixtures: javascript URLs, data URLs, oversized names, fake token fields, invalid UUIDs.

Acceptance:

- Valid fixtures pass; malicious fixtures fail closed with safe errors.
- Schema includes no token, refreshToken, cookie, authorizationHeader, psaApiKey, cloudSecret, or password field.
- All mission IDs are local identity only, never authorization proof.

Verification / evidence:

- Run schema validation tests.
- Run secret-field grep verifier.
- Attach fixture test output.

ANDROID A02 - Android Gradle and Compose scaffold

Goal: Create the Android app with a boring, stable build before adding product complexity.

Build:

- Initialize apps/android with Kotlin, Jetpack Compose, Material 3, JUnit, instrumentation test support, and release/debug build variants.
- Set package name, versionName, versionCode, minSdk policy, targetSdk current Google Play requirement, app icon placeholders, and signing placeholders.

- Add Gradle tasks for lint, unit test, assembleDebug, bundleRelease, and evidence copy.

Acceptance:

- Debug APK builds locally; release AAB task exists but uses placeholder signing until a real upload key is configured outside source.
- No secret signing material committed.
- App launches to a basic branded shell.

Verification / evidence:

- ./gradlew :app:lint :app:testDebugUnitTest :app:assembleDebug
- ./gradlew :app:bundleRelease or documented blocked state until external signing is configured.

Notes / guardrails:

- Placeholder reminder: replace Android signing config values locally; never commit keystore files.

ANDROID A03 - TAHAI mobile design system

Goal: Build a clean native design layer that can carry the electric/neon brand without sacrificing enterprise usability.

Build:

- Create color tokens, typography tokens, spacing, icons, status chips, mission cards, warning banners, empty states, and dark-mode-first theme.
- Add accessibility sizes, contrast checks, large-font resilience, and reduced-motion mode.
- Add phone and tablet preview screens.

Acceptance:

- Mission Home, Mission Card, status chip, action button, and warning banner components render across compact/medium/expanded widths.
- No critical copy is hidden or clipped at large font sizes.
- Lightweight animation exists but honors reduced-motion.

Verification / evidence:

- Compose preview screenshots.
- Accessibility lint where available.
- Screenshot evidence for phone and tablet widths.

ANDROID A04 - Local encrypted mission store

Goal: Implement safe local persistence for missions and evidence metadata.

Build:

- Add mission repository with create/read/update/archive/list operations.
- Use Android Keystore-backed encryption or SQLCipher-backed Room with documented threat model.
- Store mission metadata and evidence metadata separately from raw evidence blobs.
- Add migration strategy for schemaVersion.

Acceptance:

- App can create, persist, close, and reopen local-only missions.
- Invalid mission JSON is rejected and does not crash the app.
- No raw secrets are written in logs or crash breadcrumbs.

Verification / evidence:

- Unit tests for repository.
- Instrumentation smoke for create/reopen.
- Verifier scans test DB/export fixtures for forbidden fields.

ANDROID A05 - Mission Home v1

Goal: Create the default mobile landing surface.

Build:

- Implement active missions list, recent missions, start mission button, Launch Recipe entry point, Local Only status, offline indicator, and privacy/security summary.
- Add empty state that explains Mission Deck, Evidence Pack, and OpsTools in plain language.
- Add settings shortcut and app build info.

Acceptance:

- First launch is useful without sign-in.
- User can start a local-only mission in two taps.
- No broken IT Docs/PSA buttons appear before those contracts exist.

Verification / evidence:

- UI smoke test for first launch.
- Screenshot evidence for empty and populated states.

ANDROID A06 - Mission create/edit/archive flow

Goal: Build core mission lifecycle actions.

Build:

- Create mission form with name, mission type, optional description, and template/recipe selector.
- Add rename, change type, archive, restore, and delete-with-confirmation actions.
- Add validation for name length, allowed mission type, and safe display text.

Acceptance:

- All mission lifecycle actions are local-first and work offline.
- Delete/archive cannot occur accidentally.
- Oversized or unsafe names are rejected with clear copy.

Verification / evidence:

- Unit tests for validators.
- Instrumentation tests for create/rename/archive/restore.
- Evidence: mission lifecycle smoke report.

ANDROID A07 - Mission Deck shell

Goal: Create the phone-first operational surface.

Build:

- Build swipeable role-card deck: Overview, Runbook, Evidence, Tools, Notes, Links by default.
- Add bottom Mission Dock with primary actions and active role marker.

- Add mission status strip with Local Only / Offline / Linked / Session Expired states.

Acceptance:

- Phone navigation is thumb-first and not a shrunken desktop grid.
- Swipes and dock taps preserve mission context.
- Back behavior is predictable and cannot lose unsaved notes.

Verification / evidence:

- Navigation test for role-card switching.
- Screenshot evidence for compact phone.

ANDROID A08 - Role cards and mission link model

Goal: Add operational role cards that mirror desktop Mission Tabs without requiring desktop panes.

Build:

- Implement role cards for primary-console, docs, runbook, ticket, monitoring, evidence, live-target, vendor-portal, and tool.
- Add safe link model: title, URL, role, createdAt, optional source app, and validated protocol.
- Block javascript:, data:, vbscript:, unsafe file paths, and unknown protocols.

Acceptance:

- User can add safe HTTPS links to a mission role.
- Unsafe URLs are blocked before opening or saving.
- Role assignment uses canonical enums.

Verification / evidence:

- URL validator tests.
- Malicious fixture tests.
- UI smoke: add Cloudflare/GitHub/docs-style links.

ANDROID A09 - Runbook card v1

Goal: Make mobile useful during actual work, not just a link bucket.

Build:

- Implement local checklist sections, steps, stop/rollback condition, owner, due/sequence hints, and completion timestamps.
- Allow recipe-provided runbooks and user-created steps.
- Add undo for step completion and clear completed filters.

Acceptance:

- Runbooks work fully offline.
- Completion events are appended to the mission timeline.
- Stop/rollback condition is visually prominent.

Verification / evidence:

- Unit tests for runbook model.
- Instrumentation smoke for complete/uncomplete step.
- Screenshot evidence for active runbook.

ANDROID A10 - Notes card and mission timeline

Goal: Implement operational notes with timeline traceability.

Build:

- Add local notes with title/body, createdAt, updatedAt, pinned flag, and optional role/source link.
- Append safe timeline events for mission creation, link add, runbook step, note add/update, export, redaction preview, and evidence capture.
- Do not auto-sync or auto-write notes anywhere.

Acceptance:

- Notes survive app restart.
- Timeline is readable and filtered by event type.
- Potential secrets in notes are flagged before export but not blocked during local drafting.

Verification / evidence:

- Notes repository tests.
- Timeline event tests.
- Evidence: local restart smoke.

ANDROID A11 - Evidence model and hash chain v1

Goal: Build the evidence foundation before adding capture sources.

Build:

- Define evidence entries: kind, source role, source URL if safe, title, createdAt, local blob pointer, SHA-256 hash, redactionStatus, operatorNote.
- Add hash generation for captured blobs and exported evidence records.
- Add evidence list, detail view, and delete-with-confirmation.

Acceptance:

- Evidence metadata never stores cookies, auth headers, tokens, or raw request secrets.
- Each evidence item has stable local ID and SHA-256 when a blob exists.
- Evidence can be exported as metadata without raw blobs.

Verification / evidence:

- Unit tests for evidence hashing.
- Verifier scans evidence fixtures for forbidden fields.
- UI smoke for add/list/delete metadata item.

ANDROID A12 - Android Share Sheet capture

Goal: Make TAHAI Mobile useful from day one across the phone.

Build:

- Implement ACTION_SEND and ACTION_SEND_MULTIPLE support for URL/text capture into selected or newly created mission.
- Implement chooser flow: Add to existing mission, Start new mission, Save as note, Save as link.
- Capture source app/package when available without overclaiming identity.

Acceptance:

- User can share a URL/text from Chrome/Gmail/Files into a mission.

- Shared content is redaction-scanned before export, not silently uploaded.
- App handles no selected mission gracefully.

Verification / evidence:

- Instrumentation test for share intents where feasible.
- Manual smoke script: share URL and text into Mission Deck.

ANDROID A13 - Screenshot/photo/document evidence capture

Goal: Turn the phone into the field evidence device.

Build:

- Add camera capture into mission evidence using scoped storage and least permissions.
- Add import image/document from Android photo/file picker.
- Add optional document-scan lane if platform/library choice is privacy-reviewed.
- Add evidence note prompt after capture.

Acceptance:

- Captured files remain local unless explicitly exported/shared/synced.
- File permissions are scoped; no broad storage permission unless absolutely necessary and documented.
- User sees file type, timestamp, hash, and redaction status.

Verification / evidence:

- Camera/import manual smoke.
- Permission rationale screenshots.
- Verifier confirms no broad storage permission without documented exception.

ANDROID A14 - Redaction engine v1

Goal: Build secret detection before any polished export.

Build:

- Implement scanner for bearer tokens, Authorization/Cookie headers, AWS-like keys, GitHub token patterns, JWT-like strings, private key blocks, emails, IP addresses, tenant/account IDs.
- Add redaction preview UI for notes, links, tool results, and evidence metadata.
- Add export profiles: Internal Markdown, Sanitized Handoff, Incident Packet, Change Record.

Acceptance:

- High-risk private key blocks block export unless removed or explicitly acknowledged in internal-only mode.
- Sanitized Handoff redacts by default.
- No secret text is echoed into logs.

Verification / evidence:

- Redaction fixture test suite.
- UI smoke: note with fake bearer token triggers warning.
- Verifier: no sensitive fixture leak in generated sanitized output.

ANDROID A15 - Markdown and JSON export v1

Goal: Create useful local handoffs before PDF export.

Build:

- Generate mission export as Markdown plus machine-readable JSON manifest.
- Include mission summary, runbook status, timeline, evidence manifest, links, redaction summary, and hash list.
- Use Android Sharesheet / Files API for user-selected save/share destination.

Acceptance:

- Exports are explicit user actions.
- Export path is user-selected; app does not write arbitrary filesystem paths.
- Sanitized export removes or masks sensitive classes.

Verification / evidence:

- Unit tests for export renderer.
- Golden output tests for sanitized profile.
- Manual share/save smoke.

ANDROID A16 - PDF export v1

Goal: Add polished field-ready packets that users can send or save.

Build:

- Generate PDF packet from sanitized mission data: cover, summary, runbook, timeline, evidence index, redaction report, hash appendix.
- Use safe embedded rendering; escape all dynamic text.
- Add optional include-images toggle with warnings for sensitive screenshots/photos.

Acceptance:

- PDF has no clipped/overlapped text in sample missions.
- PDF generation works offline.
- PDF export clearly labels Sanitized vs Internal profile.

Verification / evidence:

- Golden PDF smoke with sample mission.
- Rendered PDF visual inspection evidence.
- Verifier checks dynamic text escaping.

ANDROID A17 - OpsTools pack 1 - text and network-safe tools

Goal: Build native utility value that prevents Play Store WebView-wrapper risk.

Build:

- Add JSON formatter, YAML formatter, Base64 decode, URL parser, CIDR calculator, JWT warning decoder, timestamp converter.
- Keep outputs local by default and allow Add result to mission evidence.
- Warn that JWT decoding is local display only and not validation.

Acceptance:

- Each tool can save sanitized result to mission evidence.
- Invalid input fails safely with no crash.
- No tool logs raw secret-like input.

Verification / evidence:

- Unit tests per tool.
- UI smoke for each tool.
- Verifier checks no network use in local-only tools.

ANDROID A18 - OpsTools pack 2 - DNS/TLS/headers/redirects

Goal: Add high-value IT/DevOps field diagnostics.

Build:

- Add DNS lookup, TLS certificate summary, HTTP headers summary, redirect chain, status pulse, and robots/security headers checklist.
- Add timeouts, cancellation, offline errors, and safe output trimming.
- Never capture Authorization/Cookie headers from user sessions.

Acceptance:

- Tools work against public test endpoints and fail gracefully offline.
- Only safe response metadata is stored.
- User can add results to Evidence Pack.

Verification / evidence:

- Network tool tests with mock server where feasible.
- Manual smoke with example domains.
- Redaction scan of saved tool result.

ANDROID A19 - Launch Recipes v1

Goal: Turn common IT/DevOps work into one-tap mobile mission setup.

Build:

- Add recipes: DNS Migration, Website Outage, M365 Admin Check, Cloudflare Incident, GitHub/Vercel Deploy Verify, Documentation Sprint, Client Support Call, Security Review.
- Each recipe creates role cards, default runbook steps, suggested tools, and safe placeholder links.
- No recipe includes secrets or tenant-specific data.

Acceptance:

- Recipes produce usable local missions in one tap.
- Recipe content is editable by user.
- No hardcoded private customer or credential values exist.

Verification / evidence:

- Recipe fixture tests.
- UI smoke for every recipe.
- Secret scan of recipe catalog.

ANDROID A20 - External browsing lane - Custom Tabs

Goal: Integrate provider consoles safely without pretending to control them.

Build:

- Open external mission links in Android Custom Tabs by default.
- Add branded toolbar color, back-to-TAHAI affordance, and Add to Mission actions where platform allows.
- Document session model: Custom Tabs share browser state with user browser; mission evidence does not copy cookies/session headers.

Acceptance:

- External provider URLs open reliably.
- Return to Mission Deck is obvious.
- User understands that provider authentication remains in the browser, not in mission state.

Verification / evidence:

- Manual smoke: open Cloudflare/GitHub/docs links.
- Verifier confirms no cookie extraction code exists.

ANDROID A21 - Controlled WebView lane

Goal: Use WebView only where it is safe and valuable.

Build:

- Implement first-party WebView wrapper for TAHAI-owned docs/help/status pages if needed.
- Add HTTPS allowlist, disable file access, restrict navigation, no broad addJavascriptInterface bridge, clear error states.
- Add WebView security verifier.

Acceptance:

- WebView cannot navigate to arbitrary untrusted sites unless intentionally allowed through Custom Tabs instead.
- No raw JavaScript bridge exposes native mission APIs to remote content.
- WebView errors are user-friendly and not crashy.

Verification / evidence:

- Static source verifier for WebView settings.
- Manual navigation-block smoke.
- Policy evidence showing app is not a webview-only wrapper.

ANDROID A22 - Deep links and QR mission handoff v1

Goal: Enable field-to-desktop and desktop-to-field workflows without cloud dependency.

Build:

- Add tahai-mobile:// link parser for safe internal actions only: open mission, import packet, add link, start recipe.
- Add QR export/import for sanitized mission handoff packets.
- Add signature/hash display for imported packets; imported content is untrusted until validated.

Acceptance:

- Deep links cannot execute arbitrary commands.
- QR/import rejects malformed or unsafe packets.
- User sees clear preview before importing anything.

Verification / evidence:

- Deep-link parser tests.
- Malicious packet tests.

- Manual QR round-trip smoke.

ANDROID A23 - IT Docs capability contract placeholder

Goal: Prepare the server-authorized future without adding backend drift.

Build:

- Add IT Docs status screen: Not signed in, Signed in, Org selected, Linked, Offline, Session expired, Permission denied.
- Define client interfaces for GET /api/browser/mission-capabilities and future append evidence/runbook-note calls.
- All write actions remain disabled or mocked until real IT Docs API is available and authorized.

Acceptance:

- Local-only missions remain fully useful without sign-in.
- No raw token appears in mission state, logs, or exports.
- Disabled controls show clear reasons.

Verification / evidence:

- Contract tests using fake capability responses.
- Static verifier for forbidden token fields.
- UI smoke for all auth states.

ANDROID A24 - PSA reference display contract

Goal: Implement PSA references as safe display/deep-link metadata only.

Build:

- Add optional PSA reference fields: provider, ticketId, displayKey, title, deepLink, status.
- Validate HTTPS deep links and provider enum/server-provided safe values.
- Add Append Mission Summary to PSA as disabled/future IT Docs-routed action with explanation.

Acceptance:

- No PSA API client, API keys, OAuth refresh tokens, or vendor credentials exist in mobile source.
- PSA deep links open externally only after validation.
- User copy makes clear writeback requires IT Docs authorization.

Verification / evidence:

- Static grep for PSA direct fetch/client patterns.
- Fixture tests for safe/unsafe PSA links.
- UI smoke for disabled writeback state.

ANDROID A25 - Incident Mode and notifications

Goal: Add high-value operational behavior without noisy automation.

Build:

- Add mission type incident with persistent optional notification while active.
- Notification actions: Add note, Add evidence, Open runbook, End incident.
- Add quiet hours/respectful notification defaults.

Acceptance:

- Notifications are user-initiated and easy to stop.

- No unsolicited spam or background polling by default.
- Incident timeline captures user actions.

Verification / evidence:

- Manual notification smoke.
- Permission/rationale screenshots.
- Battery/background behavior review.

ANDROID A26 - Tablet adaptive Mission Control v1

Goal: Bring the desktop advantage to tablets without hurting phone UX.

Build:

- Add adaptive layout: compact uses Mission Deck, medium uses two-column deck/rail, expanded uses 2-Up/3-Up/Quad-like panes for role cards and first-party/tool surfaces.
- Add active pane/card marker and role labels.
- External provider consoles still open through Custom Tabs unless a controlled surface is explicitly safe.

Acceptance:

- Tablet screen can view runbook + evidence or tools + notes side by side.
- Phone remains simple and unchanged.
- No hidden overflow makes critical controls unreachable.

Verification / evidence:

- Screenshot evidence: phone, small tablet, large tablet.
- Adaptive layout tests where feasible.

ANDROID A27 - Hardware keyboard and power-user actions

Goal: Support tablets, Chromebooks, foldables, and DeX-style usage.

Build:

- Add keyboard shortcuts for search, new mission, add note, add evidence, switch role cards, export, and find tool.
- Add command/search sheet equivalent to desktop Ctrl+K.
- Add shortcuts discoverability panel.

Acceptance:

- Keyboard shortcuts do not break normal text entry.
- Command sheet shows target mission/action and disabled reason.
- No destructive action without confirmation.

Verification / evidence:

- Unit tests for command registry.
- Manual keyboard smoke on emulator/tablet profile.

ANDROID A28 - Settings, privacy, and local vault controls

Goal: Give users clear trust controls.

Build:

- Add settings for biometric lock, auto-lock timeout, export defaults, redaction defaults, offline mode, clear local evidence, and diagnostics.
- Add local vault status and storage usage summary.
- Add privacy policy link and Data Safety summary mirror.

Acceptance:

- Biometric/local device credential gate works when enabled.
- Diagnostics are sanitized and do not include mission content unless user explicitly exports a sanitized bundle.
- User can delete local missions/evidence intentionally.

Verification / evidence:

- Biometric/manual smoke.
- Diagnostics redaction tests.
- Settings screenshot evidence.

ANDROID A29 - Mobile source security verifier

Goal: Add automated guardrails that fail fast.

Build:

- Create tools/verify-mobile-sources script.
- Check forbidden fields, direct PSA clients, raw token/cookie storage, broad WebView bridges, cleartext release config, committed keystores, .env files, generated artifacts, unsafe URL protocols in fixtures.
- Add CI workflow for lint/test/verify.

Acceptance:

- Verifier fails on intentional bad fixtures and passes clean source.
- CI blocks obvious security drift.
- Verifier output is readable and actionable.

Verification / evidence:

- Run verifier in clean repo.
- Add CI evidence artifact.
- Document expected false-positive handling.

ANDROID A30 - Crash-safe diagnostics and logging

Goal: Prepare for testers without leaking sensitive material.

Build:

- Add local diagnostic bundle: app version, device class, feature flags, sanitized recent errors, build config, verifier version.
- Use privacy-preserving crash reporting only after data map and policy approvals, or keep manual diagnostics for MVP.
- Add no-secret logging wrapper.

Acceptance:

- No notes/evidence/raw URLs with sensitive query params appear in diagnostics by default.
- User can preview diagnostics before sharing.
- Crash handling does not create corrupt mission state.

Verification / evidence:

- Diagnostics fixture tests.
- Manual crash/restart smoke.
- Log scan evidence.

ANDROID A31 - Testing matrix and emulator/device farm baseline

Goal: Stop guessing before Play review.

Build:

- Create device matrix: compact phone, large phone, foldable/tablet, low-memory emulator, Android current, Android previous major, dark/light, large font.
- Add smoke checklist for Mission Home, create mission, share capture, evidence capture, redaction, export, Custom Tabs, WebView allowlist, offline.
- Add release-candidate checklist template.

Acceptance:

- Every RC has a completed matrix or explicit exception.
- Known issues are documented before tester builds.
- Critical flows work on at least one physical Android device before Play submission.

Verification / evidence:

- Generate smoke evidence folder.
- Attach screenshots and command logs.
- Known-issues doc updated.

ANDROID A32 - Play Store compliance pack

Goal: Prepare listing, policy, privacy, and review materials before first upload.

Build:

- Write Google Play short description, full description, feature graphic copy, screenshot plan, privacy policy, support URL, app category, Data Safety draft, permissions rationale, and review notes.
- Document why app is not a WebView wrapper: native mission store, evidence capture, redaction, tools, offline export, runbooks.
- Confirm package name, app signing path, upload key outside repo, versionCode, targetSdk, AAB build.

Acceptance:

- All Play Console content exists before internal testing.
- Data Safety draft maps every SDK and data type.
- Permissions match actual features and rationale.

Verification / evidence:

- Checklist evidence: target API, AAB, Data Safety, privacy URL, screenshots, policy review notes.
- Run release verifier.

ANDROID A33 - Internal app sharing / internal testing build

Goal: Put the first real build in tester hands with limited blast radius.

Build:

- Build signed AAB/APK as appropriate for internal testing path.

- Upload to Play internal app sharing or internal testing track.
- Collect feedback on first-run, Mission Deck, share capture, evidence capture, redaction, and export.

Acceptance:

- Install flow succeeds for internal testers.
- No crash on first launch or core mission flow.
- Feedback issues are triaged into blocker/non-blocker.

Verification / evidence:

- Play upload evidence.
- Tester install screenshots or notes.
- Crash/ANR check after test window.

ANDROID A34 - Closed testing production-access pass

Goal: Complete required tester cycle if account requires it.

Build:

- Recruit/confirm tester group.
- Run closed test for required period if applicable.
- Maintain release notes, tester instructions, feedback form, and issue triage.

Acceptance:

- Required tester threshold and duration are satisfied if Play Console requires them.
- No blocker crash/ANR issues remain from closed testing.
- Core value is demonstrable to non-developer testers.

Verification / evidence:

- Closed testing dashboard screenshots.
- Tester feedback summary.
- Bugfix log and production-access answers draft.

ANDROID A35 - Release Candidate hardening

Goal: Make the RC boring, stable, and review-safe.

Build:

- Fix closed-test blockers.
- Run full source verifier, lint, unit tests, instrumentation tests, smoke matrix, redaction fixtures, export fixtures, and policy checklist.
- Freeze public copy and screenshots.

Acceptance:

- No critical crash/ANR, permission, privacy, broken functionality, or policy blockers.
- No known secret leaks in exports/diagnostics/logs.
- AAB builds from clean checkout.

Verification / evidence:

- Final Android RC evidence pack.
- Build checksum and versionCode/versionName proof.

- Known issues limited to non-blocking items.

ANDROID A36 - Google Play production submission

Goal: Submit deliberately with reviewer-friendly context.

Build:

- Upload production AAB.
- Complete content rating, target audience, Data Safety, app access if login required, privacy policy, testing notes, and review instructions.
- Use staged rollout where available.

Acceptance:

- Production submission accepted into review or documented with exact rejection/remediation if not.
- Reviewer can access all necessary app functionality without needing secret credentials.
- App listing accurately describes features and data use.

Verification / evidence:

- Play Console submission screenshot/evidence.
- Archive submitted AAB checksum and metadata copy.
- Review outcome log.

ANDROID A37 - Post-launch Android vitals and support loop

Goal: Treat launch as the beginning of hardening, not the finish line.

Build:

- Monitor Android vitals, crash/ANR, reviews, support emails, and tester feedback.
- Prioritize stability/security/export bugs before new features.
- Publish patch releases through staged rollouts.

Acceptance:

- Crash/ANR signals stay under Google Play bad-behavior thresholds.
- Support issues are captured into a release triage board.
- Privacy/Data Safety updates are made if practices change.

Verification / evidence:

- Weekly vitals screenshot/report during first month.
- Patch release evidence if needed.
- Post-launch retrospective.

ANDROID A38 - Android v1.1 feature polish after Play launch

Goal: Add differentiators after the first store proof exists.

Build:

- Polish animations, tablet Mission Control, Launch Recipes, OpsTools, and export templates based on real feedback.
- Add optional desktop handoff integration once desktop browser supports corresponding packet format.
- Consider IT Docs live auth only after backend endpoints are stable and security-reviewed.

Acceptance:

- No new feature weakens local-only/offline behavior.
- No IT Docs/PSA direct secret storage is introduced.
- Play listing and Data Safety remain accurate.

Verification / evidence:

- Feature-specific tests.
- Store metadata diff review.
- Security verifier passes.

4. Android release commands and local verification starter

These are starter commands for the eventual Android repo. Adjust module names only after the repo is actually created.

```
# Optional WSL entry pattern if building from Fedora WSL instead of Android Studio/Windows shell
wsl -d FedoraLinux-43 --cd /mnt/c/dev/tahai-mobile-mission-control

# Android repo root
cd C:\dev\tahai-mobile-mission-control\apps\android

# Clean dependency/build verification
./gradlew --version
./gradlew :app:lint :app:testDebugUnitTest
./gradlew :app:assembleDebug

# Instrumentation tests require an emulator or physical device
./gradlew :app:connectedDebugAndroidTest

# Release artifact for Google Play production path
# Placeholder reminder: configure upload signing outside the repo before release.
./gradlew :app:bundleRelease

# Mobile source guardrails
cd C:\dev\tahai-mobile-mission-control
./tools/verify-mobile-sources.ps1
# or
bash ./tools/verify-mobile-sources.sh
```

5. iOS/App Store pass plan after Google Play launch

Do iOS after Android has proven product value and store-readiness. The iOS app should reuse the mobile mission schema and lessons learned, not blindly port Android screens.

IOS I00 - iOS feasibility and App Store source plan

Goal: Start iOS only after Android production or stable closed-test evidence exists.

Build:

- Review Android post-launch findings and freeze mobile schema compatibility.
- Document iOS constraints: SwiftUI native shell, WKWebView/Safari auth, App Store privacy details, no assumption of non-WebKit browser engine outside special entitlement paths.
- Create docs/ios-app-store-plan.md and docs/ios-privacy-map.md.

Acceptance:

- iOS plan explicitly maps what is reused, redesigned, or deferred.
- App Store minimum functionality risk is addressed with native mission/evidence/tools, not a website wrapper.
- Privacy labels and SDK inventory plan exists.

Verification / evidence:

- Review doc signed off as project source.
- Schema compatibility checklist.

IOS I01 - SwiftUI app scaffold

Goal: Create a clean native iOS source lane.

Build:

- Add apps/ios with SwiftUI, app lifecycle, dark-mode-first TAHAI theme, build schemes, unit test target, UI test target.
- Add bundle ID, versioning, placeholder icons, and no secret signing material in repo.
- Add README and build commands.

Acceptance:

- App builds and launches to branded shell.
- No provisioning profiles/certs/secrets committed.
- iOS source is separate from Android-specific implementation.

Verification / evidence:

- Xcode build/test evidence.
- Source hygiene verifier.

IOS I02 - Mission schema implementation

Goal: Implement the canonical shared mission schema in Swift.

Build:

- Create Swift models from mission schema with Codable validation.
- Port malicious/valid fixtures from Android shared schema.
- Implement URL/protocol/enums/size validation.

Acceptance:

- iOS accepts the same valid mission packets as Android.
- iOS rejects malicious fixtures the same way Android does.
- No token/secret fields added.

Verification / evidence:

- Unit tests using shared fixtures.
- Compatibility report against Android schema.

IOS I03 - Secure local mission store

Goal: Build local-first encrypted persistence.

Build:

- Use Keychain/Secure Enclave/local protected storage strategy appropriate for mission metadata and evidence pointers.
- Add create/list/update/archive mission repository.
- Add migration discipline and corruption recovery.

Acceptance:

- Local-only missions survive restart.
- Invalid mission packets are rejected safely.
- No secrets are logged.

Verification / evidence:

- Unit tests and launch/restart smoke.
- Log scan evidence.

IOS I04 - Mission Home and Mission Deck

Goal: Port the proven mobile UX natively.

Build:

- Implement Mission Home, active mission list, start mission, status chips, and Mission Deck role-card navigation.

- Adapt gestures to iOS conventions.
- Preserve phone-first design and iPad adaptive path.

Acceptance:

- Core UX feels native to iOS, not like Android copied mechanically.
- Local-only first launch is useful.
- No broken integration buttons.

Verification / evidence:

- UI screenshots for iPhone and iPad.
- UI smoke tests.

IOS I05 - Runbook, notes, timeline

Goal: Add operational local work surfaces.

Build:

- Implement checklist/runbook, notes, and timeline with local persistence.
- Port redaction pre-scan hooks.
- Add undo and safe delete/archive confirmations.

Acceptance:

- Runbooks and notes work offline.
- Timeline captures local actions only.
- No auto-sync or PSA writeback.

Verification / evidence:

- Unit tests and UI smoke.
- Restart persistence test.

IOS I06 - Share Extension capture

Goal: Match Android share capture with iOS-native flow.

Build:

- Add Share Extension to capture URLs/text/files into new or existing mission.
- Add import preview and redaction warning.
- Handle extension storage handoff safely to main app.

Acceptance:

- User can share from Safari/Mail/Files into TAHAI mission.
- Extension cannot corrupt mission store.
- Shared data remains local unless explicitly exported/synced.

Verification / evidence:

- Share extension manual smoke.
- Extension storage tests.

IOS I07 - Evidence capture

Goal: Port photo/document/file evidence behavior.

Build:

- Add camera/photo/file import using iOS permission and picker patterns.
- Add evidence hashing and metadata display.
- Add note-after-capture prompt.

Acceptance:

- No unnecessary broad permissions.
- Evidence can be exported and deleted intentionally.
- Hash and timestamp are visible.

Verification / evidence:

- Permission screenshot evidence.
- Capture/import smoke.

IOS I08 - Redaction and export

Goal: Port redaction engine and exports.

Build:

- Implement redaction fixtures in Swift.
- Generate Markdown/JSON and PDF exports or use platform-appropriate PDF rendering.
- Add iOS share sheet export.

Acceptance:

- Sanitized handoff output matches Android behavior as closely as possible.
- PDF export escapes dynamic text.
- Private key/high-risk secret behavior is consistent with Android.

Verification / evidence:

- Redaction fixture tests.
- Export golden tests.
- PDF visual smoke.

IOS I09 - OpsTools iOS pack

Goal: Port the tools that made Android review-safe and useful.

Build:

- Add text tools and network-safe tools consistent with Android.
- Use iOS network APIs with timeouts and cancellation.
- Add Add result to Evidence Pack.

Acceptance:

- Tools work offline/local where possible.
- Network tools do not capture sensitive headers/cookies.
- Invalid input fails safely.

Verification / evidence:

- Tool unit tests.

- UI smoke.

IOS I10 - Safari/WKWebView browsing lanes

Goal: Implement iOS web strategy without fighting platform reality.

Build:

- Use Safari-compatible auth/open flows for external provider consoles.
- Use WKWebView only for first-party/controlled content where needed.
- Add allowlists, navigation controls, and no broad JavaScript bridge.

Acceptance:

- External consoles open safely.
- First-party web surfaces are controlled and policy-safe.
- No arbitrary remote content receives privileged mission APIs.

Verification / evidence:

- Navigation allowlist tests.
- Source verifier for unsafe WKWebView patterns.

IOS I11 - iPad Mission Control

Goal: Bring tablet Mission Control to iPad after phone UX is stable.

Build:

- Add adaptive layouts for iPad: deck + rail, 2-Up, 3-Up, role-card dashboard, and export/evidence sidebars.
- Support hardware keyboard shortcuts where appropriate.
- Preserve iPhone simplicity.

Acceptance:

- iPad gives meaningful multi-surface productivity.
- iPhone remains uncluttered.
- No hidden panes or unreachable controls.

Verification / evidence:

- iPhone/iPad screenshot evidence.
- Keyboard shortcut smoke.

IOS I12 - IT Docs and PSA display contracts

Goal: Port safe integration contracts only.

Build:

- Add IT Docs status/capability states and PSA display reference model.
- Keep writeback disabled or routed through authorized IT Docs endpoints only when backend exists.
- No stored OAuth refresh tokens/PSA credentials/cloud provider secrets.

Acceptance:

- Local-only missions remain useful without sign-in.
- Unauthorized/expired states are clear.
- No direct PSA client exists.

Verification / evidence:

- Contract tests.
- Static grep/verifier.

IOS I13 - App Store privacy and review pack

Goal: Prepare Apple submission materials early.

Build:

- Create privacy policy mapping, App Privacy details, support URL, screenshots, reviewer notes, content rating, sign-in/test account plan if needed.
- Document native functionality beyond WebView: Mission Deck, evidence, redaction, tools, offline export, share extension.
- Inventory third-party SDKs and privacy manifests where applicable.

Acceptance:

- App privacy answers are accurate and match app behavior.
- Reviewer can test core app without private credentials.
- App has real native functionality and is not a wrapper.

Verification / evidence:

- App Store Connect checklist evidence.
- Privacy map review.
- Screenshot set.

IOS I14 - TestFlight beta

Goal: Run controlled iOS beta after Android learning is incorporated.

Build:

- Upload TestFlight build.
- Collect feedback on first launch, share extension, evidence capture, export, iPad layout, and browsing handoff.
- Fix blockers before App Store production submit.

Acceptance:

- TestFlight install succeeds.
- No critical crash/permission/export issues remain.
- Feedback is triaged.

Verification / evidence:

- TestFlight dashboard evidence.
- Crash report review.
- Bugfix log.

IOS I15 - App Store release candidate

Goal: Freeze production candidate.

Build:

- Run full tests, source verifier, privacy map review, App Store metadata review, screenshot review, and device matrix.
- Archive build and metadata.

- Prepare staged rollout / phased release if appropriate.

Acceptance:

- No blocker known issues.
- No privacy mismatch.
- No secret leakage in logs/diagnostics/exports.

Verification / evidence:

- Final RC evidence pack.
- Build archive checksum.
- Known issues final review.

IOS I16 - App Store submission and post-launch

Goal: Submit and monitor.

Build:

- Submit production build with clear reviewer notes.
- Respond to review issues with exact changes, not vague appeals.
- Monitor crashes, reviews, support, and privacy/reporting changes after launch.

Acceptance:

- App accepted or rejection/remediation is documented.
- Post-launch support loop exists.
- Android/iOS roadmap converges after both stores are live.

Verification / evidence:

- App Review result evidence.
- Post-launch metrics report.
- Patch-release plan if needed.

6. Cross-platform hard guardrails

6.1 Mobile security invariants

- Never store PSA/API/provider secrets in mission state, local files, exports, logs, or diagnostics.
- Never copy raw cookies, Authorization headers, refresh tokens, or cloud-provider tokens from a browser surface.
- Never use local mission IDs, IT Docs org IDs, project IDs, ticket IDs, or deep links as proof of authorization.
- Never auto-write to IT Docs or PSA. All future writeback requires explicit user action and server-side authorization.
- Never expose mission APIs to arbitrary remote web content through WebView/WKWebView bridges.
- Always validate imported mission packets as untrusted input.
- Always run redaction preview before export/sync.
- Always make offline/local-only/session-expired/permission-denied states visible without blocking local work.

6.2 Mobile permissions rule

Every permission needs a feature, a user-facing reason, a test, and a store-disclosure entry. If a feature can use Android/iOS pickers instead of broad storage permissions, use the picker. If evidence capture can stay local, keep it local.

6.3 Data collection default

Default posture for MVP: no advertising SDKs, no behavioral analytics SDKs, no third-party crash SDK until the privacy/data map and store disclosures are finished. Manual sanitized diagnostics are acceptable and safer for the first release.

6.4 Done means store-real, not demo-real

- Builds from a clean checkout.
- No secret or generated artifact in source.
- Unit/instrumentation/UI smoke tests pass or exceptions are documented.
- Store policy checklist is current.
- Data Safety / App Privacy answers match actual behavior.
- Redaction fixtures pass.
- Export output is visually inspected.
- Known issues are documented and not release-blocking.
- Every returned repo ZIP is verified for non-empty, no nested accidental zip-only source, and no runtime caches.

7. New-chat starter for Android implementation

```
We are starting TAHAI Mobile Mission Control - Android / Google Play first.

Canonical product split:
- Desktop TAHAI = IT/DevOps command browser.
- Mobile TAHAI = field/operator companion.
- Phone UX = Mission Deck, not shrunken desktop Quad View.
- Tablet UX = adaptive Mission Control after phone core is solid.

Repo target:
C:\dev\tahai-mobile-mission-control

Hard scope:
- Android first using Kotlin + Jetpack Compose.
- iOS comes after Google Play launch/stable proof.
- Preserve Mission Control model: Mission Tabs, Mission Views, Mission Tools, Mission Evidence.
- Preserve IT Docs/PSA boundary: client references only unless IT Docs server authorizes future writeback.
- No PSA/API/cloud/provider secrets in app source, mission state, logs, diagnostics, exports, or fixtures.
- No raw cookies/Auth headers copied from browser surfaces.
```

```
- No WebView-only wrapper product. Native mission/evidence/tools/offline export are required for Play Store value.

Current pass:
ANDROID A00 - Mobile SSOT and repo creation

Expected output each pass:
- Full repo ZIP
- Smoke evidence
- Verifier result
- Remaining pass count

Baseline commands once repo exists:
cd C:\dev\tahai-mobile-mission-control\apps\android
./gradlew :app:lint :app:testDebugUnitTest :app:assembleDebug
cd C:\dev\tahai-mobile-mission-control
./tools/verify-mobile-sources.ps1

Do not create placeholder-only passes. Build real source changes and keep the app store/security posture clean.
```

8. Source references consulted

TAHAI Browser Mission Tabs / Mission Control Guardrails PDF
Project-source PDF uploaded in this project, April 26, 2026.

Android App Bundle requirement
https://developer.android.com/guide/app-bundle

Google Play target API level requirement
https://developer.android.com/google/play/requirements/target-sdk

Google Play Data Safety declaration
https://support.google.com/googleplay/android-developer/answer/10787469

Google Play closed testing for new personal developer accounts
https://support.google.com/googleplay/android-developer/answer/14151465

Google Play Webviews and Affiliate Spam policy
https://play.google.com/about/spam-min-functionality/spam/made-for-ads/

Google Play Functionality, Content, and User Experience
https://support.google.com/googleplay/android-developer/answer/9898783

Android technical quality / vitals
https://developer.android.com/quality/technical

Android Custom Tabs overview
https://developer.chrome.com/docs/android/custom-tabs

Android insecure WebView native bridges risk guidance
https://developer.android.com/privacy-and-security/risks/insecure-webview-native-bridges

Apple App Review Guidelines
https://developer.apple.com/app-store/review/guidelines/

Apple App Privacy Details
https://developer.apple.com/app-store/app-privacy-details/

Apple alternative browser engines in the EU
https://developer.apple.com/support/alternative-browser-engines/

9. Final anti-drift clause

When uncertain, choose the safer local-only mobile implementation and document what must wait for IT Docs server support, Google Play/App Store review, or later platform-specific capabilities. The Android release should prove native mobile value before any iOS port or deep integration expansion.