The React Native App Audit Checklist

The 40 things I check in audit and rescue engagements

Dhairya Senjaliya — Senior React Native + Python + AI Engineer · dhairyasenjaliya.com · Guest Engineer at Expensify · 20+ App Store launches

1. Architecture & Code Health

01 ☐ **Feature-based folder structure, not type-based**
Screens, hooks, and API calls for one feature living together is what keeps a codebase navigable past 50 screens. A giant components/ folder is the first smell of a project that grew without an owner.

02 ☐ **TypeScript strict mode is on — and respected**
strict: false or a codebase full of `any` means the type system is decoration. Most 'random' production crashes in audits trace to a shape mismatch TypeScript would have caught.

03 ☐ **State management fits the app, not the trend**
One giant context re-rendering the whole tree, or Redux boilerplate wrapping two values — both are drag. Server state belongs in a query library; UI state in something small.

04 ☐ **Business logic lives outside components**
Pricing rules and data transforms buried in JSX can't be tested and get duplicated. Components render; logic lives in plain functions that have unit tests.

05 ☐ **Dependency audit: unmaintained and duplicate libraries**
Two date libraries, three icon sets, and a navigation lib abandoned in 2022 — each one is future migration debt. Every dependency should earn its place.

2. Performance

06 ☐ **Long lists are virtualized properly**
Feeds beyond a few hundred items need FlashList or a well-configured FlatList. Blank cells while scrolling is the #1 user-visible performance complaint in rescues.

07 ☐ **No re-render storms on core screens**
A keystroke that re-renders 40 components is why the app 'feels laggy'. Profile first: most RN performance problems are React problems, not native ones.

08 ☐ **Images are sized, cached, and compressed**
Shipping 4000px images into 120px avatars burns memory and scroll performance. Resize on the server, cache on the client, use modern formats.

09 ☐ **Hermes enabled and startup time measured**
Cold start over ~3 seconds loses users before the first screen. Measure TTI on a mid-range Android device, not your simulator.

10 ☐ **JS bundle audited for weight**
Moment with all locales, lodash imported wholesale, dev-only tooling in production — bundle bloat slows startup on every single launch.

3. Reliability & Error Handling

11 ☐ **Crash reporting wired with readable stack traces**
Sentry (or equivalent) with sourcemaps uploaded per release. A crash report pointing at minified line 1 is not observability.

12 ☐ **Error boundaries around risky surfaces**
One thrown error in a deeply nested component should not white-screen the entire app. Boundaries plus a friendly retry screen contain the blast radius.

13 ☐ **Network failures are handled, not assumed away**
Every API call needs a loading, error, and retry story. Apps tested only on office Wi-Fi fall apart on train Wi-Fi.

14 ☐ **Offline behavior is defined**

Even 'this app requires a connection' is a decision — show it deliberately. Cached reads and queued writes are the difference between 4 and 5 stars in reviews.

15 ☐ **Slow frames and ANRs monitored in production**

Crashes get reported; jank silently churns users. Production performance monitoring shows what your test devices never will.

4. Security

16 ☐ **Tokens in SecureStore/Keychain — never AsyncStorage**

AsyncStorage is plaintext and readable from device backups. Auth tokens belong in the platform keychain, full stop.

17 ☐ **No secrets inside the JS bundle**

API keys shipped in the app can be extracted in minutes. Anything secret stays server-side behind endpoints you control.

18 ☐ **Transport security: HTTPS everywhere, pinning where it matters**

Fintech and health apps warrant certificate pinning; every app warrants zero plain-HTTP calls, including that one legacy analytics endpoint.

19 ☐ **Deep links validated before acting**

A deep link is untrusted input. Links that trigger payments, open WebViews, or prefill forms need the same validation as any API input.

20 ☐ **Dependency vulnerabilities reviewed on a schedule**

npm audit noise is real, but so are the criticals hiding in it. A monthly triage habit beats a panicked upgrade the week of a pen test.

5. Auth & Data Integrity

21 ☐ **Token refresh is deduplicated and race-free**

Five parallel 401s must trigger one refresh, not five. Refresh races are the classic cause of 'randomly logged out' bug reports.

22 ☐ **Logout actually clears everything**

Tokens, caches, query stores, persisted state — a logout that leaves the previous user's data behind is a privacy incident on shared devices.

23 ☐ **No PII leaking into logs or analytics**

Emails and tokens in breadcrumbs or event payloads become a compliance problem the day you sign an enterprise customer.

24 ☐ **Persisted local data has a migration strategy**

Changing a persisted store's shape without migrations crashes exactly the users who've had the app longest — your best users.

25 ☐ **API error contract is consistent**

If every endpoint fails differently, the app handles none of them well. One error envelope, handled in one interceptor.

6. Release Engineering

26 ☐ **CI builds and tests both platforms on every PR**

'It builds on my Mac' is not a release process. EAS or equivalent CI catching platform breaks before merge is table stakes.

27 ☐ **OTA update strategy with a rollback plan**

Over-the-air updates fix JS bugs in hours instead of store-review days — but only if rollback is one command and someone owns the button.

28 ☐ **Store rollouts are staged**

Shipping to 100% of users on day one turns any missed bug into a support fire. 10% → 50% → 100% with crash-rate gates between.

29 ☐ **Versioning and release notes are disciplined**

When a user reports a bug, you need to know exactly which JS bundle and native build they run. Chaos here makes every incident slower.

30 ☐ **Store listings are current and owned**

Screenshots from three redesigns ago and an unowned developer account are conversion and continuity risks nobody notices until it hurts.

7. Testing & Quality

31 ☐ **Critical paths have E2E coverage**

Login, purchase, and the core loop covered by Maestro or Detox — the flows where a regression costs money get automated first.

32 ☐ **Business logic has unit tests**

Not 100% coverage — targeted tests on the functions that compute money, permissions, and state transitions.

33 ☐ **API responses validated at runtime**

A zod schema at the network boundary turns 'backend changed a field and the app crashes' into a logged, handled error.

34 ☐ **A real device matrix exists**

Mid-range Android with 3GB RAM is where performance dies first. If every test device costs $1000+, the app is untested for most users.

35 ☐ **Accessibility basics pass**

Labels on touchables, contrast, and font-scaling that doesn't break layouts — legally required in more markets every year, and it widens your user base.

8. Observability & Handoff

36 ☐ **Core funnel events are tracked and trusted**

If signup-to-activation can't be answered from analytics, product decisions are guesses. Five reliable events beat fifty noisy ones.

37 ☐ **Someone owns the dashboards and alerts**

Crash-rate and API-error alerts that page a named human. Monitoring nobody reads is decoration.

38 ☐ **Feature flags gate risky changes**

The ability to turn a misbehaving feature off remotely — without a release — has saved every serious app I've worked on at least once.

39 ☐ **README gets a new engineer running in under an hour**

Setup folklore that lives in one person's head is a bus-factor problem. The audit test: can I run the app from the README alone?

40 ☐ **A runbook exists for the top three incidents**

App down, API down, bad release — written steps, owners, and rollback commands. Incident response designed during an incident goes badly.

Every unchecked box is a risk you now know about. Want this audit run on your app by the person who wrote it? Book a call at **dhairyasenjaliya.com** — or grab the FastAPI and RAG checklists when they ship: dhairyasenjaliya.com/feed.xml