AI Personal CFO (Rekhon)

Blueprint Summary — the ten-minute read

23 October 2026 · v0.13.10 · Readiness: **READY WITH RISKS**, 0 blocking questions · Android, Kotlin, 60 modules, no server

Full detail: the twelve documents in `docs/project-blueprint/`

The short edition. Ten minutes instead of two hours. Every claim here is expanded, with its evidence, in `docs/project-blueprint/`.

*Version examined: `v0.13.10` · 2026-10-23 · Readiness: **READY WITH RISKS***

1. The product, in one paragraph

An Android app that acts as a full-time CFO for one person or household in India, and runs entirely on the phone. It records money in three taps, reads receipts and — with explicit, revocable permission — bank alerts on-device, and then does the work a ledger does not: forecasts 90 days and names the days the money runs short, sizes an emergency fund from real essential spend, computes what a goal needs each month, splits an EMI to the paise, measures a holding's money-weighted return, decides where the next surplus rupee should go, and answers "can I afford this?" with the gate-by-gate trace behind the verdict. There is no account, no sign-up, and no server holding user data.

2. The seven rules that override everything

These are binding on the code, not aspirations, and most of them fail a build when broken.

Rule	What it means
Numbers from math, words from AI	Deterministic engines compute every figure. A language model may only put figures into sentences — it never produces one
Guardrail every AI output	Every sentence passes a numeric check before display. A figure the engines did not produce never reaches the user
Privacy first	Nothing leaves the device without explicit, per-feature, revocable consent. Default is fully offline
Advice, never orders	The app recommends and simulates. It never moves money
Offline first	Every core feature works in airplane mode. The build and the tests pass with the backend absent
Show the work	Every recommendation shows its inputs, the rule that fired, and a plain-language reason
Deterministic and testable	Fixed input, fixed output. Randomness only through an injected, seedable source

Two more carry the same weight at the code level: **money is integer paise end to end** (a floating-point amount fails the build) and **time is UTC millis through an injected clock** (a wall-clock read in domain code fails the build).

3. What it does

Capture — three-tap entry · transfers as one record · splits that always sum exactly · future-dated entries that post themselves · on-device receipt OCR · opt-in SMS drafts · recurring-series proposals · statement import with provenance.

Ledger — eleven account types · daily net worth · classification by merchant, by nature (need/want/investment/asset/liability) and by stream (fixed/semi-fixed/variable).

Plan — budgets with pace, projection and once-per-band alerts · goals with required monthly, feasibility and a prioritisation waterfall · an emergency-fund target and runway · an eight-stage surplus order of operations.

Wealth — credit-card cycle and utilisation · loan amortisation to the paise · holdings with dated lots and XIRR · allocation and concentration against a published rulebook · vehicles and appliances with service prediction.

Intelligence — a 90-day forecast with P10/P50/P90 and named crunch days · seasonality from the user's own history · a 0–1000 health score with its contributions and strongest lever · a ranked insight feed · a notification policy · the purchase advisor with a kept trace · simulators, tax, insurance and market signals · a chat assistant behind the guardrail.

Platform — biometric app lock · a privacy mask on every amount · capture-guarded screens · a consents dashboard with one-tap revoke · lossless export and import · E2EE backup with a drilled restore · crypto-shredding erase · a widget · four languages · household and business profiles.

4. The architecture, in five lines

- **60 Gradle modules**, dependencies one way: `feature` → `domain` → `data/core`.
- **30 engines in pure Kotlin**, no Android imports — enforced by a guard in CI, which is also what made an iOS port even attemptable.
- **A six-layer AI pipeline**: data → analytics → rules → predictions → decisions → language. Determinism increases downward; nothing above the decision layer may produce a figure.
- **Room over SQLCipher**, key in the Android Keystore (StrongBox where available). 40 tables, 435 columns, schema version 33, 32 non-destructive migrations.
- **No server in the product's path.** The one specified network endpoint ships with a blank base URL.

5. How the rules are kept

A rule that is only written down drifts. These are tasks that fail a build:

Mechanism	What it stops
6 custom lint detectors	Floating-point money · `GlobalScope` · a wall-clock read in domain code · an amount in a log line · a hardcoded UI string · hand-rolled crypto
A TestKit architecture guard	An Android import reaching an engine
36 drift-test classes, with their data files declared as build inputs	A typed Kotlin mirror diverging from the `ai/` row it claims to copy
Golden-file tests per engine version	An engine's answer changing unnoticed — and the harness refuses to rewrite its own fixture
Property tests	Splits that do not sum; balances that do not equal their ledger
115 Paparazzi renders	A UI change nobody looked at, in light, dark and at 200% font
`koverVerify` with a real bound	Coverage below 85%, or below **100% on money maths**
An offline smoke on a real device	A core feature quietly needing the network
A restore drill before every promotion	A backup that does not actually restore
An OSV scan over resolved coordinates	A known vulnerability in a version that actually ships
A DEX-level log-strip check	Trusting the build config instead of reading the artefact

3,986 tests in 386 test classes. Twice, the project has found a gate that could not fail — coverage verification with zero rules, and a screenshot baseline nothing compared against — and fixed both. That history is why two Definition-of-Done items are documented *as* unenforced rather than implied to be gates.

6. By the numbers

Blueprint		Code	
Functional requirements	95	Gradle modules	60
Non-functional requirements	20	Kotlin files	885
Business rules · validation rules	28 · 22	Room entities (v33)	40 (435 columns)
Acceptance criteria	115	DAOs · migrations	34 · 32
Use cases (all with failure paths)	55	Engines	30
Data entities	40	Feature modules · screens	13 · 25
Components	60	Typed routes · workers	21 · 9
API operations	12 (1 implemented)	Custom lint detectors	6
Test cases	124	ADRs	82
Open tasks · gaps (1 closed) · open questions	25 · 19 · 8	Tests · screenshot renders	3,986 · 115

7. Status

Delivered. Epics 1–13 — foundations, onboarding and security, capture, categorisation and budgets, the dashboard and export, wealth, goals, backup, the AI core, the advisor and chat, privacy hardening, release quality, and five of seven expansion features.

Open, and why.

Item	State
The market-data backend (SRS §22)	**Not built.** Eleven of twelve operations unimplemented; the client ships unconfigured. Everything degrades to cache with a staleness label, which is a path the app was built for — except the market backtest, which is genuinely removed
Performance targets	Three NFRs carry numbers that **nothing measures**. The architecture is right and tested; the bounds are not
Account Aggregator	Designed, reserved for in the schema, unbuilt
iOS through KMP	**Blocked.** "Android-free" did not mean "KMP-portable", and the blocker is the exact-paise `Money` type
Six SRS engine sections	Specified, unbuilt, and in neither a milestone nor a descope note
19 files over the 500-line rule	Led by one at 4,723 lines and one at 2,666. A provable no-op refactor is available

8. Verdict

READY WITH RISKS. Thirteen completeness checks pass, five are partial, none fails. **Zero blocking open questions. Zero critical gaps** of nineteen.

Not `READY`, because test traceability is partial: two requirements have no test and four non-functional targets have no measurement — each listed rather than omitted. Not `NOT READY`, because nothing missing makes the work unsafe to continue: every requirement has an acceptance criterion, the data model is read off the schema rather than imagined, 93 of 95 requirements have a test, and the two High gaps are *a decision not yet taken* and *a measurement not yet made* — neither is a defect in what exists.

The three questions worth answering first

1. Will the market-data proxy be hosted, or formally descoped? A "no" is a good answer; silence is the only bad one.
2. Will the three performance targets be measured, or restated without numbers?
3. Does exactness yield to portability, or does iOS leave the roadmap?

Full set: `docs/project-blueprint/` *— twelve documents, five diagrams, and a validator that checks every identifier in them. The unflattering companion is* `docs/architecture-and-gap-report.md`*.*