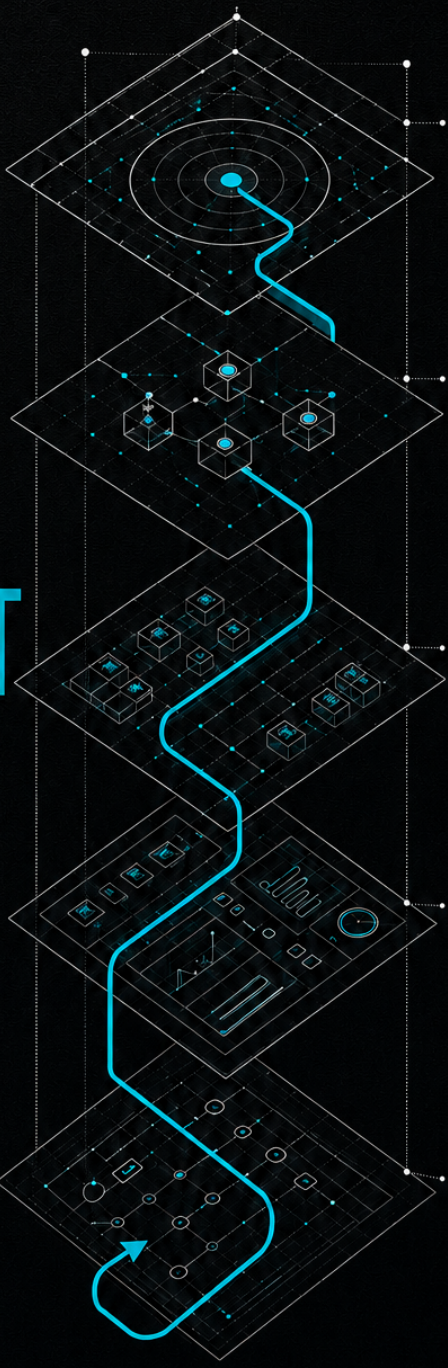
MIKE SALARI

MOBILE SYSTEM DESIGN BLUEPRINT



A Staff engineer's working laboratory for evolving, migrating, releasing, and operating mobile systems.

INCLUDES

20 Chapters

12 Real-World Scenarios

Architecture Decision Records

Checklists & Playbooks

Reference Implementations

FIRST EDITION FRAME • DECIDE • MIGRATE • OPERATE • REVERSE MASTER EDITION

salari.dev

A short sample

One case. One idea. This is a brief taste of Mobile System Design Blueprint, not the whole book. It follows a single decision, a "trending" badge in a photo-sharing app, far enough to show how the book thinks, then it stops. If it changes how you see where a value is computed, the rest will do the same across payments, messaging, migrations, incidents, security, cross-platform, and on-device AI.

The full book is at https://salari.dev/books/mobile-system-design-blueprint.

Chapter 1. The architecture decision loop

The screenshot that killed a sponsorship

Halo is a photo-sharing app with roughly eight million monthly actives and a creator economy bolted onto the side. In app version 5.4 the team shipped a small, beloved thing: a "trending" badge that appears on a post when it catches fire. The badge was computed on the device. Each client watched the like count arrive over the post's lifetime, estimated a `likeVelocity1h`, and if that crossed a threshold it set `post.trendingBadge = true` and drew the little flame. Clean, cheap, no new backend. It demoed beautifully and shipped on a Thursday.

The escalation arrived eleven days later, and it was not a crash. Crash-free sessions held at 99.91 percent the entire time. A creator, @maren_frames, had a post cross the threshold on her phone, screenshotted the flame, and sent it to a brand manager as proof the post was trending, part of a pitch for a paid campaign. The brand manager opened Halo on his own phone, looked at the same post, and saw no badge. His device had joined late, estimated a lower velocity off a shorter window, and never crossed the threshold. To him the creator had faked a screenshot. The deal died. Maren posted about it. Within a day support had two hundred tickets that all said some version of "the trending badge is a lie."

The engineer who picked up the escalation spent an afternoon proving the code was working exactly as written. That is the worst kind of incident, the kind with no bug to fix. Two devices watching the same post computed two different truths because each ran its own estimator over its own slice of time, and nothing in the design said the badge had to mean the same thing to everyone looking at the post. Where a value is computed is an architectural decision. The team had made it by reflex, to avoid a backend, and the reflex quietly promised something the product could not keep: that a public badge is a shared fact.

The recurring failure, stated plainly: a senior engineer can draw the target architecture but cannot say what evidence justifies it, which assumption is most likely to be wrong, how the first ten percent ships, or what happens when adoption stalls. The badge is a small instance of a large habit. This chapter is about making that decision on purpose, and building the loop that revisits it as the app grows.

The durable principle

> Architecture is a governed sequence of reversible and irreversible decisions, not a final diagram.

Hold on to the word *governed*. A diagram is a snapshot of one moment's opinion. What survives contact with production is not the boxes but the rules around them: what promise each box makes, who owns the promise, how you would know it broke, and what evidence would make you tear the box down. The client-side badge was not wrong because on-device computation is bad. It was wrong because nobody decided whether the badge was a private hint or a public claim, and no signal existed that would have caught the disagreement before a brand manager did.

Mobile makes this sharper than the backend does. A server can be redeployed in minutes and the old version simply stops existing. A mobile population cannot be assumed to update, reconnect, or even launch during your remediation window. The night of the Maren incident, three Halo clients were live: 5.4 with the on-device badge, 5.3 at about 31 percent of actives with no badge at all, and 4.9 at roughly 4 percent, held back on older devices whose owners had auto-update off. Any decision about the badge had to hold across all three at once. Code, persisted data, SDK contracts, config, and store-distributed binaries all evolve at different speeds, and the slowest one sets your real turning radius.

Five layers to separate before you name a component

Most failed reviews collapse into a fight about mechanism. The move that separates a Staff answer from a senior one is to place a decision on five layers first, so the argument about tools happens last.

The **invariant** is the user or business outcome that must stay true. For the badge: *two people looking at the same post at the same moment must see the same badge state.* Notice this names no technology. It is testable without knowing whether the flame is drawn from a device estimate or a server field.

The **constraint** is the limit code style cannot negotiate away. Halo's creators screenshot badges into contracts, so the badge is evidence, not decoration. That is a constraint, not a preference. Another constraint:

5.3 and 4.9 are in the field and will stay there for months.

The **mechanism** is the current implementation. On-device `likeVelocity1h` crossing a threshold. Mechanisms are the layer that changes most and matters least to the argument.

The **operating model** is ownership, rollout, observability, support, and incident response. Who is paged when badges disagree, who can halt a rollout, what dashboard answers "how often does this happen."

The **revisit rule** is the evidence that expands, modifies, or abandons the decision. Without it, every decision is permanent by accident.

A proposal that contains only the mechanism layer is an implementation proposal wearing an architecture costume. Run all five decisions below through these layers, and keep asking the review's sharpest question: strip the preferred technology out of the document, and does the contract still make sense? If not, you were shown a tool.

Decision 1: state the user-visible and operational problem before naming components

The purpose of this decision is to turn an ambiguous expectation into an observable contract. "Add a trending badge" is an expectation. "Two viewers of one post see one badge state, and we can measure disagreement" is a contract. The difference is that the second one can fail a review and can fail a test.

You have at least three ways to say what the badge *is*, and they are not interchangeable. It can be a **private hint** ("posts moving fast for you"), in which case per-device computation is correct and the Maren incident is a documentation problem, not an architecture one. It can be a **shared public claim**, in which case the value must have a single authority. Or it can be a **ranked-feed side effect**, computed wherever the feed is ranked and carried alongside the post. Halo's product team, once asked directly, said the badge was a public claim that creators monetize. That one sentence eliminated the on-device mechanism entirely, before anyone drew a box.

The negative case that exposes a weak answer here is the one Halo lived: **the architecture review becomes a diagram review.** If the review had only checked that the badge box connected to the feed box with a tidy arrow, it would have passed, because the diagram was never wrong. Diagrams do not encode "same post, same badge, everywhere."

Contracts do.

The evidence to require is **decision lead time**: the elapsed time from "we have enough production signal to decide the badge's authority" to "the decision is recorded and owned," measured per initiative, with a threshold that a decision blocking a rollout may not sit unreviewed for more than five working days. Record the outcome in the Architecture Decision Packet with the client versions it touches (5.4, 5.3, 4.9), the persisted field it governs (`post.trendingBadge`), and any temporary bridge. Then apply the challenge: remove the words "server" and "on-device" from the statement. If "two viewers see one badge" still reads as the point, you have a contract, not a tool.

The sample stops here

That is the first of five decisions in this one chapter. The remaining four, choosing the evidence and the revisit date before writing code, and designing adoption, rollback, ownership, and measurement into the architecture, continue in the full chapter, along with the worked Architecture Decision Packet filled in for Halo, the failure modes to review explicitly, the operational signals with owners and thresholds, and the Swift and Kotlin the decision compiles down to.

Before you go, one tool from the book worth keeping on your desk.

The tool you would keep on your desk

Every chapter and scenario in the book is scored on one rubric. It is how you tell a competent senior design from a Staff one, and it is the thing readers pin next to their screen before a real review. Every exercise and every longitudinal scenario is scored on the same seven dimensions, each from 0 to 4. Here are the seven, with what a top score of 4 actually looks like. The full 0 to 4 ladder for each row is in the book.

Dimension	What a 4 looks like
Problem and invariants	Invariants tied to business and operating policy, not a feature list

Dimension	What a 4 looks like
State and authority	Reconciliation, trust, and failure ownership are explicit
Compatibility and evolution	Reversal economics and alternate futures are planned
Reliability and operations	Cohort-aware recovery and long-tail remediation
Security and privacy	Risk policy with false-positive handling and support design
Organization and adoption	Decision rights, incentives, exceptions, and deprecation
Evidence and learning	A decision policy and an architecture learning loop

Add the seven scores and read the band. 0 to 8 is an incomplete design, 14 to 18 is competent senior work, 19 to 23 is Staff-level system ownership, and 24 to 28 is Principal-level decision quality. Anything above 23 takes more than technical depth. It takes credible migration, operations, evidence, and organizational adoption, which is exactly what the badge decision above was missing.

Get the full book

Mobile System Design Blueprint is a working laboratory for evolving, migrating, releasing, and operating iOS and Android systems. It is the work that starts after the diagram is drawn.

Inside the full edition:

- 20 decision chapters, each built around one worked failure case like

the one you just read
- 12 longitudinal scenarios, from payments to on-device AI, each evolving across five stages with an incident
- 50 scored exercises with a complete answer key
- six named frameworks, a professional toolkit, and printable field cards
- a tested companion Swift and Kotlin package, one module per chapter decision
- light and dark editions, in PDF and EPUB

This sample was one case from Chapter 1. The book takes the same method all the way through.

Get the full book at https://salari.dev/books/mobile-system-design-blueprint

Frame. Decide. Migrate. Operate. Reverse.

Mike Salari, a Staff mobile engineer and tech lead. salari.dev.