

How I rebuilt MapChat as a production-grade app

Mayur Mehta · Founding Product Manager, MapChat Social · mayur.d.mehta@gmail.com · [linkedin.com/in/mehta-mayur](https://www.linkedin.com/in/mehta-mayur)

At MapChat I took a consumer social app from a no-code web app to native iOS and Android apps on both stores in three months, as its only product-and-engineering owner. I designed the product, the data platform and the bar every release had to clear, and I used AI coding agents to build at startup speed without lowering it.

RESULTS

3 months

from native rebuild to live on both app stores (May 2026)

0 rollbacks

across post-launch releases, June to August 2026

95.87% → 99.45%

crash-free sessions across releases, tracked in Sentry

1,925 → 17,000+

venues and 100,000+ events and specials, no licensed feed

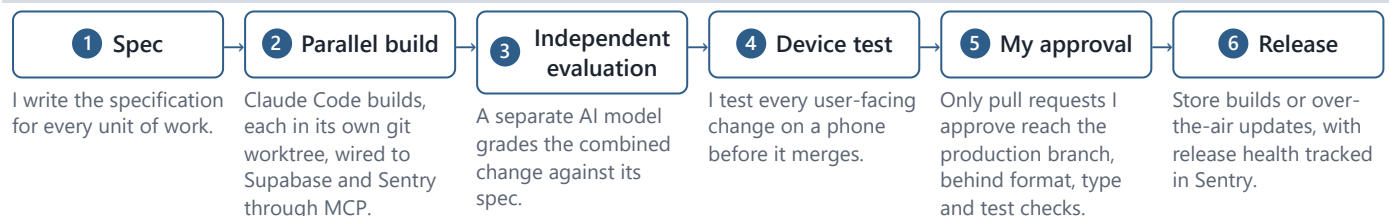
THE PROBLEM

MapChat helps people meet at bars and events. Its first version was a no-code web app whose backend could not carry the product. No licensed feed covered the venues and events the map needed, and the store date could not move. The app had to be production-grade from day one, with a founding team of four.

WHAT I FIGURED OUT

- **Honest on an empty network.** With few users at launch, I had the app lead with curated picks, never social proof it could not back, like ratings or “people here now”.
- **When to change course.** Five weeks into the native rebuild the no-code backend hit its limits, so I reversed my own plan and moved to Supabase and PostgreSQL in two days while the web app stayed live.
- **A data platform two teams share.** I designed a 160-table PostgreSQL model with row-level security on every table, where our venue CRM writes the catalog and the app owns the social data.
- **A catalog with no licensed feed.** I designed the recurrence engine and the LLM-classified scraping harnesses our CRM team runs, which admit a recurring event only when schedule confidence clears about 80%.
- **Fast at 9x the supply.** I made a Docker replica of production the gate for every database performance change, and map and feed now load in under a second on a cold launch.
- **A date that held.** I closed the store-rejection risks in one audit before submission and cut five features to hold the store date.

THE ENGINEERING BAR



HOW THE BAR HOLDS

- **No one grades their own work.** After a builder marked its own work as verified, only the independent evaluator may write that stamp.
- **Checks run on every commit.** An audit found changes merged past a failing lint check, so I moved the checks into git hooks every commit must pass.
- **Autonomy has a fence.** Large multi-agent runs need my approval, run by run.

HOW I EVALUATE AI

I evaluate AI work four ways, with deterministic checks, LLM-as-judge, rubrics and human review.

- **Risk sets the review.** Database migrations, security work and releases also go to a four-lens adversarial AI review.
- **Model choices are experiments.** Before changing the default builder model I commissioned a pre-registered blind comparison, with criteria fixed before results. Spec quality, not the model, decided output quality.
- **It started in production ML.** At Robert Half I lifted Invoca's NLP call classifier from 65% to 90% accuracy with its data scientists and a golden set I hand-labeled.

ACROSS TEAMS

Shared context. I built a GitHub Issues relay that let our agents and the games team's answer each other's questions from the code, with our side designed to run sandboxed and read-only. Three integrations shipped through it, and cross-team delivery ran roughly twice as fast, by my estimate.

Whatever writes the code, I decide what gets built, how it is checked and when it ships.