The Factory: Greenfield Use Case

Taking a vibe-coded prototype to secure, compliant production-grade.

Lovable, v0, Bolt, and Replit will get you a working prototype in a weekend. The part that decides whether a product survives real users is the part those tools don't do: real architecture, authorization that works under real access patterns, infrastructure, a deployment pipeline, a compliance story, and moving your existing data across without losing a row.

The below is an example of the Factory taking one prototype the rest of the way. The starting point was an internal operations app (workforce planning, time-tracking, role-gated finance views) built with AI generated code on a hosted Postgres database with row-level security.

How the engagement is shaped

The work is divided by four human-signed gates: Contract, Architecture, Verification, Delivery with the goal of maintaining the initial functionality while adding the security, reliability, observability and flexibility required for the production version.. The Factory runs autonomously between gates; a person gives approval to move onto the next gate.

What makes it trustworthy: Council Mode

Any tool can produce a confident answer. The one that hurts in production is the confidently wrong one: the plausible architecture choice, the migration that looks complete but dropped rows, the auth fix that opens a new hole. Before a load-bearing decision the Factory runs Council Mode: a critic from a different AI provider family attacks the proposal, the original agent rebuts, and the result is re-evaluated before a human gate. The rule is that no single agent can lock in a load-bearing decision. Different families fail in different places, so this is a second prior, not a second opinion. On this engagement it settled the early choices (data layer, authentication, runtime) at a gate arbitrated by the operator, not by one agent's silent preference. Claude orchestrates, and everything on the panel is swappable except the orchestrator itself, which stays on Anthropic for long-horizon context.

What it did

Re-platformed, and found the holes the prototype was hiding. Moving authorization out of the database and into the application forced every access path into the open. It caught and fixed a direct object-reference bug (change the ID in the URL, edit another user's records: critical), an authorization check that failed open, new users created with no role, managers granted global write instead of scope to their own reports, and a reporting view that leaked everyone's data to any employee. Each fixed and covered by a test. A prototype that looked done was leaking data across users underneath.

Migrated the real data and proved it to the row. The dataset lived in the hosted platform as roughly 18 raw CSV files, joins encoded as ID columns. It built a re-runnable harness: clear the target in reverse dependency order, fix the format landmines (locale dates the new database rejects, invalid byte sequences, identity references to a user system that no longer existed), and load in strict dependency order. Then it reconciled source row counts against landed row counts and flagged any mismatch, and reset identity sequences so the first record created after go-live wouldn't collide. "Done" meant every table was reconciled, not that the script finished.

Produced a compliance story an auditor can read. A full security review: findings across critical, high, medium, and low, each with a remediation, mapped to the OWASP Top 10 and SOC 2 common criteria, with a role-by-role authorization matrix that proves "employees can't see finance, managers see only their reports" as a tested property. Dependency vulnerabilities triaged, not ignored.

Built the infrastructure as code. A private network with the database unreachable from the internet, managed Postgres with encryption at rest and enforced TLS, a web application firewall behind a CDN, secrets in a managed store with rotation, and a pipeline that runs lint, type-check, build, and tests and blocks the merge on a red bar. After one incident where the build passed but hid a type error, it added a standalone type-check gate so that class of bug couldn't recur, and wrote down why.

Stayed on after delivery. Monitoring wired to alerts, incident triage handled the same disciplined way as the brownfield engagement, and a running backlog of fixes and enhancements shipped as pull requests under the client's merge authority.

The point

With the right skills and insight into user needs, a skilled product owner can generate a working prototype in a matter of days. As with all large projects, the 80/20 rule applies and it is the last push to get to production that takes the remaining 80 percent of the time. In this case, the Factory was able to efficiently bridge that gap and generate production-grade architecture, authorization, migration, infrastructure, compliance, and the proof that all of it works. The product is up and running with happy users and tight security.