Reputation-based trading platform

Jake Tae-moon Jo · Engineering lead · https://jake-jo-portfolio.vercel.app/en/work/reputation-market/

Built a user app, an admin tool and an API server from scratch, with the admin tool in place 13 days after the first commit; the features went in as eight bundles, each built and checked before its commit (20 days, pre-launch), and 238 of 242 planning answers went into the spec.

Pre-launch	238 / 242	25 / 25	2,583
20 days, not deployed	planning answers received	top-priority UX issues fixed	automated tests, all pass on rerun

Results

Thirteen days after the first commit on the new planning documents the admin tool was in, and the last commit came on Oct 30, 20 days in. Push notifications are on hold and four planning questions are still waiting for answers.

Every change request from the client was shipped within 24 hours.

Built a user app (Flutter, iOS and Android), an admin tool (Next.js), and an API server and worker (NestJS). The product is pre-launch; for production, the layout is designed and a rollback runbook is drafted.

Thirteen days after the Oct 10 start the admin tool was committed, and the last commit came on Oct 30, 20 days in. Push notifications are on hold, four planning questions are waiting for answers, and the backend tests that need a database were not part of the Nov 10 rerun.

Exchanged 242 questions with the client's product planner over 8 rounds and wrote the answers to 238 into the spec. Four are still waiting for an answer.

Got two external integrations working on live adapters: real SMS sending and address search, and staging and production were configured to use the same live drivers (Oct 25).

Fixed all 25 top-priority issues found in the UX audit.

All 2,583 automated tests passed on the Nov 10 rerun.

Period

Oct 2026 (20 days)

Role

Only human on the build team; designed and ran the AI agent team

Team

Me plus a 13-role AI agent team

Platforms

iOS and Android app, web admin tool, API server and worker

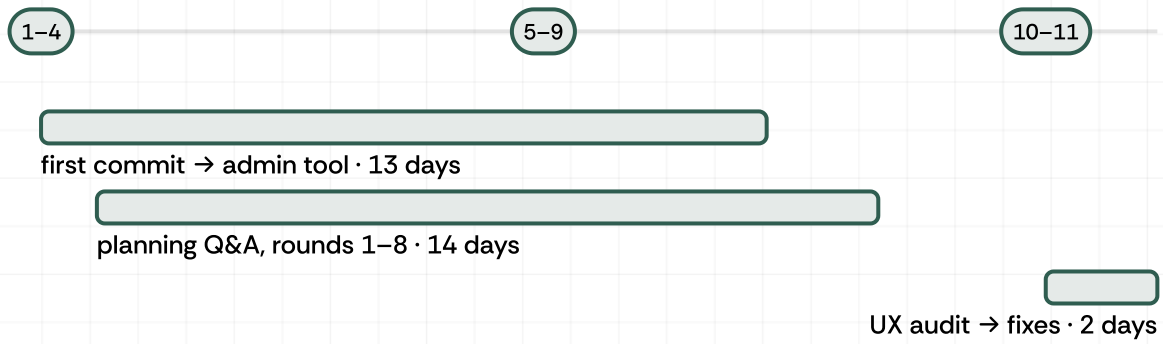
Context

A company preparing to launch a trading platform; its product planner wrote the planning documents and the QA checklist

Twenty days from the first commit

Thirteen days after starting on the new planning documents the admin tool was done, and over 20 days 238 of 242 planning questions were answered and written into the spec.

13 days	20 days	238 / 242
to the admin tool	of development	planning answers received into the spec



1 Oct 10, 2026 — **Start**
17 new planning documents, four repos, the stack and agent harness of an earlier project

2 Oct 11, 2026 — **Topology and data model**
First 109 planning questions; 2 architecture decision records

3 Oct 14, 2026 — **Bundles 1–3: auth, relations, feed**
Spec v0.2; push-queue decision

4 Oct 16, 2026 — **Bundle 4: trades, DM, reports**
Trade confirmation in one transaction

5 Oct 19, 2026 — **Bundle 5: notifications**
Full cross-check against the planning documents (spec v0.5–0.7)

6 Oct 21, 2026 — **Bundle 6: tiers, rankings, profile**

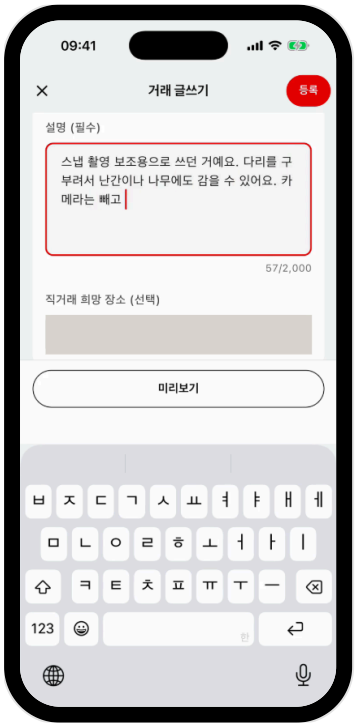
7 Oct 23, 2026 — **Bundle 7: admin tool**

		13 days after the first commit; spec v0.8
8	Oct 24, 2026	**Eighth round of answers** 238 of 242 questions resolved, 4 open
9	Oct 25, 2026	**Two live integrations** Live SMS and address search; staging deleted the day it was built (internal release postponed)
10	Oct 28, 2026	**UX audit** 110 findings, 25 top-priority
11	Oct 30, 2026	**Last commit** All 25 top-priority fixed

Screens

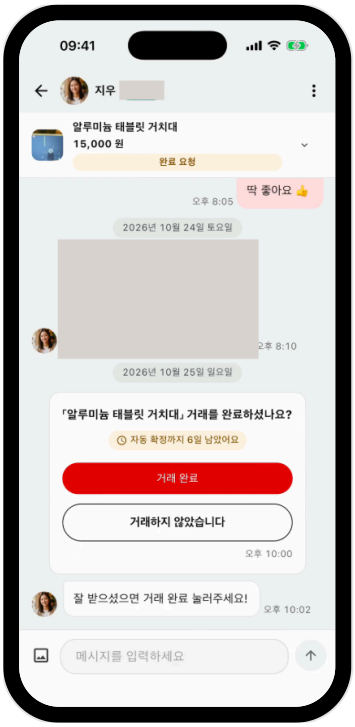
Writing a trade post

Photos, title, price and description, with a preferred meeting place left optional. Demo seed data; the place field is masked.



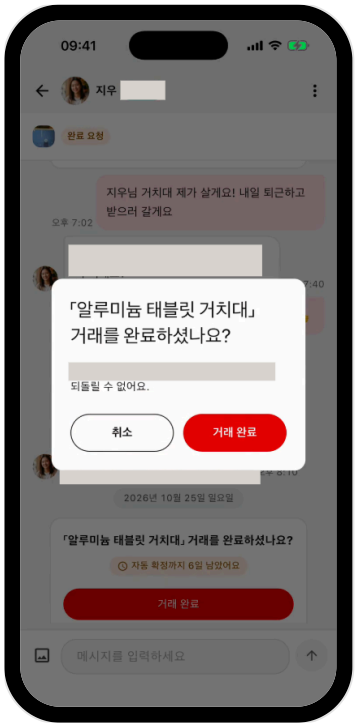
The trade bar inside a DM

The trade status sits at the top, and a card in the conversation offers "Complete trade" or "We did not trade", with the days left before auto-confirm. The relationship-step chip is masked.



Pressing "Complete trade" asks once more, and says it cannot be undone after confirmation

The points sentence, the relationship-step chip and a place name in the chat are masked.



Problem and context

The job was to build the product from scratch in October, on new planning documents and with the same stack and agent harness as an earlier project.

The client is a company preparing to launch a trading platform built on a social graph. Users build a first-degree network out of mutually accepted friends. When a first-degree friend relays a trade, connection or feed post to their own friends, the post spreads to second- and third-degree contacts. The two parties to a trade are connected in DM, and activity builds up as reputation points.

The app only connects the parties. There is no in-app payment, escrow or shipping, and reputation points cannot be bought, transferred or cashed out. The problem the client was aiming at was the distrust of trading with strangers; its answer was to let people trade only with those they are linked to through their network.

Work began from scratch on Oct 10. Its basis was 17 new planning documents and a 1,450-item QA checklist from the client's product planner. In the product, the system works out first- to third-degree relationships itself, there is no in-app payment, and the points are reputation points you can neither buy nor spend. The stack is NestJS, Next.js and Flutter, the same as an earlier project (the earned wage access platform). The reason recorded in the design document is "share the agent harness and operating knowledge."

One trade, end to end

The app never moves money; it only connects people, and the moment a trade is confirmed the trade points and the brokers' share are written in one transaction.

1. **Post** ○Person
   A seller posts a trade
2. **Relay** ○Person
   Direct friends pass it on, reaching 2nd- and 3rd-degree contacts
3. **Start a trade** ○Person
   The relay path and degree are fixed the moment the buyer taps; the post row is locked
4. **Connected in DM** ○Person
   The two parties meet in DM; no in-app payment or shipping
5. **Completion request** ○Person
   The seller picks the buyer and asks

⑥ **Confirm** ○Person
Buyer confirms, says "we did not trade", or it auto-confirms after 7 days (notice 2 days before)

⑦ **One-transaction settle** □System
Done, sibling trades on the post cancelled, trade points, broker points to the relay path

⑧ **Tally-period points → tier** □System
Tier by how many active users have more points in the tally period

My role

The client wrote the planning documents; before any code, I turned their gaps into questions, and I ran the agent team and made the structural calls.

The agent team is the 13-role team brought over from an earlier project. The role names stayed the same, and every definition file was edited for this project. The pipeline is the same eight stages too: intake → architect → design → develop → QA → review → integrate → document. I am the only git author, and all 226 commits carry an AI co-author mark. Counted from the session records, the agents were called 275 times. The main session acted as orchestrator.

Questions came before code. On Oct 11 I decided that, before any development, the ambiguous and conflicting parts of the planning documents would become a list of questions, and the first 109 went out. Seven more rounds followed during development, for 242 questions in all (plus 11 separate corrections to the planning documents in round 2), and the answers to 238 of them went into the spec. Each answer meant editing that story's acceptance criteria, marking it settled and bumping the spec version. Where planning documents disagreed, the order of precedence was rules document > screen document > common UX > overview. The spec has 207 stories (197 Must, 10 Won't), and the 1,450-item QA checklist is linked to them. The planning documents, the QA checklist and the answers are the work of the client's product planner.
I made the major structural and operations calls. On Oct 10 I set the stack, agents and repo layout to match the earlier project, and chose AWS as the deployment target. On Oct 11 I set production to servers that scale with traffic plus a shared database and Redis, with development and staging on a single Docker server. On Oct 14 I asked for a message queue to handle push bursts, and on Oct 23 I had the quiet hours for marketing pushes made a setting (default 21:00–08:00). On Oct 25 I set

the app versioning rule, forced updates and automatically generated release notes.

I also set how the work ran. On Oct 19 I asked for a one-page status report so that people who are not developers could gauge progress and timing. On Oct 26 I set the loop for screen fixes: fix → capture to confirm → independent QA, with the spec winning over the implementation when they disagree, and a question to me after three rounds. On Oct 25–27 came the demo seed data (about four months of activity by 45 fictional users, replayed through the service code) and a Figma design system and screens. The original design came from the client's side. On Oct 29 I adopted the recommendation on 18 of the 19 planning questions raised by the UX audit and held 1. The decision card numbers are the design document's decision number (D10) and ADR numbers.



Who did what

A person made the direction and operations calls, all 226 commits carry the AI co-author mark, and security and code reviewer agents ran 41 reviews. There was no CI and no PR.

Me

Same stack, agents and repo layout as an earlier project (Oct 10)

Topology: servers that scale with traffic plus a shared DB and Redis; dev and staging on one server (Oct 11)

Questions on ambiguities and conflicts in the planning documents before any code (Oct 11)

A message queue for push bursts (Oct 14)

Cheapest possible staging, deleted the same day (Oct 25)

Adopted the recommendation on 18 of 19 UX planning questions (Oct 29)

Agents · 13 roles

Same 13 role names as an earlier project, each definition edited

275 calls (including 67 by the built-in general-purpose and explore agents): app 59 · backend 51 · frontend 17 · docs 15 · designer 13 · architect 9 and others

AI co-author mark on all 226 commits

Review gates

16 security, 25 code and 1 spec-coverage review (Oct 14–29)

25 review findings cited in ADRs, 22 known limits recorded

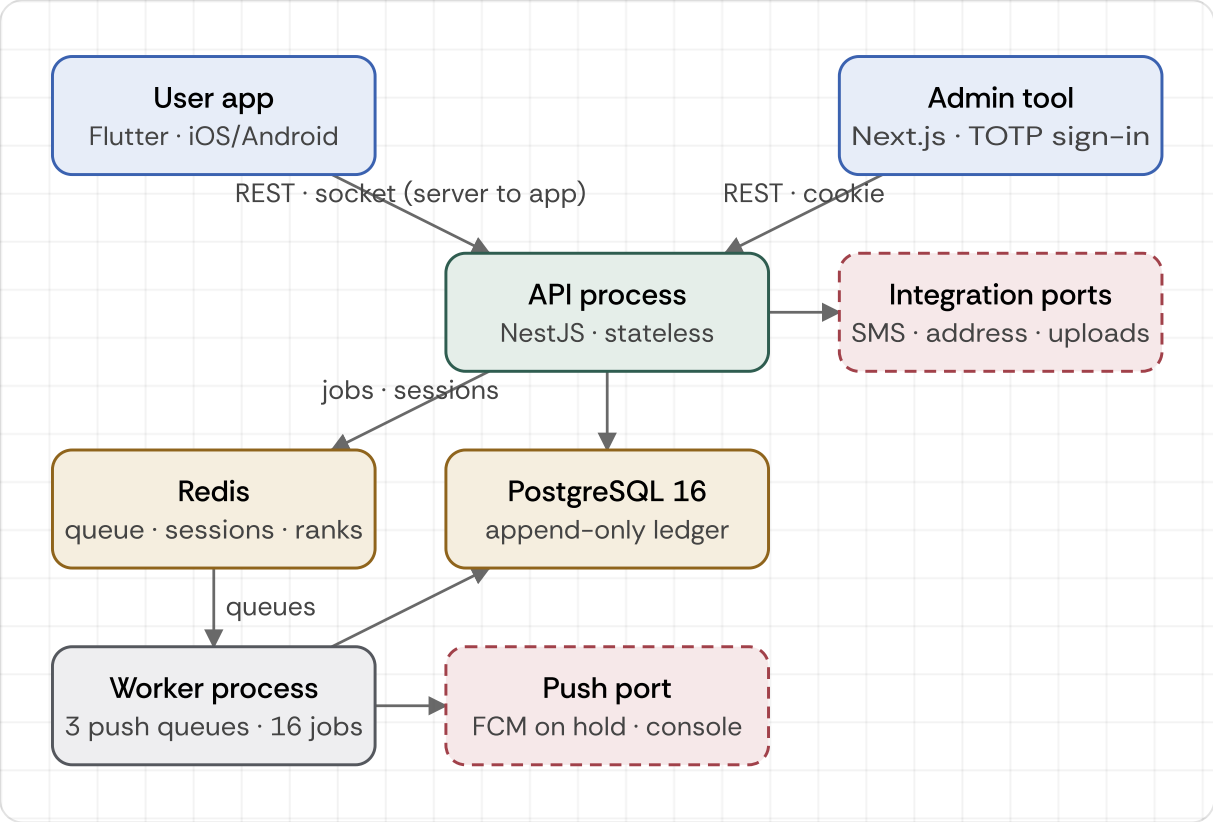
force push, reset --hard, rm -rf and .env reads blocked

No CI, no PR

System architecture

System architecture

The API keeps its state in Redis so it can run as many copies as needed, and push notifications and scheduled jobs run only in the worker process, through queues.



User app

Flutter with Riverpod. Sending goes over REST; receiving over socket.io, used only for server-to-app notifications. App models are written by hand and checked against the OpenAPI snapshot. Contacts are matched by one-way hashes only.

Admin tool

Next.js. Signing in with ID, password and TOTP returns a Redis-backed session cookie. Types are generated from the backend's OpenAPI.

API process

42 NestJS modules. Sessions, rate limits and idempotency keys live in Redis, so any number of copies holds no state. Feature modules never call external SDKs directly; they go through ports. The production layout that scales with traffic was designed but not built.

Integration ports

Of the 7 ports, 3 have live adapters (SMS, address search, LLM release notes); the rest are mocks, console or local disk. The development environment makes 0 external calls, and the production configuration refuses the development adapters.

Redis

BullMQ broker, sessions and rate limits, the socket.io adapter, tier and ranking snapshots. Because it holds queues, memory eviction is off, so large caches live in the database instead.

PostgreSQL 16

41 models. The ledger refuses UPDATE and DELETE by trigger, takes each event exactly once, and accepts nothing for a finished tally period. Second- and third-degree relationships are cached as one array row per user.

Worker process

A separate process from the same code. It runs the 3 push queues (immediate, 5-minute digest, bulk in 500-token chunks) and 16 recurring jobs on schedules stored in Redis, so any number of instances share one schedule. There is no cron inside the API.

Push port

FCM was put on hold because the store developer accounts came late. Until then the push port runs on a console adapter.

`src/trade/trade.service.ts` · Every trade transaction locks users → post → trade, then puts confirmation, sibling cancellation and point payment in one transaction. If the buyer and the auto-confirm batch collide, the second sees an already-confirmed trade and stops. (comments condensed from the source, some lines omitted, names changed)

```
await runTx(this.prisma, async (tx) => {
  // Lock order of every trade transaction (ADR 0005 §4.2):
  // users (id order) → posts → trades → dm_rooms → the rest.
  const payees = [head.buyerId, head.sellerId, ...plan.relayers];
  await lockParties(tx, payees);
  await lockPost(tx, head.postId);
  const t = await lockTrade(tx, tradeId);
  // Buyer and auto-confirm batch race: the second finds COMPLETED.
  if (t.status === 'COMPLETED') return;
  // …
  await writeTrade(tx, t.id, 'COMPLETE_REQUESTED', {
    status: 'COMPLETED', confirmedAt: now,
    confirmMethod: auto ? 'AUTO_CONFIRM' : 'BUYER_CONFIRM',
  });
  // One completion request per post: the others are in progress only.
  const siblings = await tx.trade.updateManyAndReturn({
    where: {
      postId: t.postId, id: { not: t.id }, status: 'IN_PROGRESS',
    },
    data: {
      status: 'CANCELLED', cancelReason: 'OTHER_TRADE_COMPLETED',
    },
  });
  await this.points.pay(tx, t, plan, now); // trade + broker points
  await rederivePost(tx, t.postId);
  // … DM system cards for the trade and every cancelled sibling
}); // any failure rolls all of it back
```

`prisma/migrations/…_init_schema/migration.sql` · A trigger refuses UPDATE and DELETE on the points ledger, and a new row goes only into an unfinished tally period whose range holds its time. Beyond that, a unique constraint on (user, source key) lets each event in exactly once, an exclusion constraint keeps tally periods from overlapping, and a CHECK keeps the point caches from going below zero. (two triggers from the first migration; quotes and names shortened)

```
-- The points ledger is append-only: no UPDATE or DELETE, ever.
CREATE FUNCTION forbid_update_delete() RETURNS trigger
LANGUAGE plpgsql AS $$
BEGIN
  RAISE EXCEPTION '% is append-only (% rejected)', TG_TABLE_NAME, TG_OP
    USING ERRCODE = 'restrict_violation';
END $$;
CREATE TRIGGER ledger_append_only
  BEFORE UPDATE OR DELETE ON ledger_entries
  FOR EACH ROW EXECUTE FUNCTION forbid_update_delete();
-- …
-- A row goes only to the open season whose range holds created_at.
CREATE FUNCTION ledger_check_season() RETURNS trigger
LANGUAGE plpgsql AS $$
BEGIN
  IF NOT EXISTS (
    SELECT 1 FROM seasons s
    WHERE s.id = NEW.season_id
      AND NEW.created_at >= s.starts_at AND NEW.created_at < s.ends_at
      AND s.finalized_at IS NULL
  ) THEN
    RAISE EXCEPTION 'season % does not cover % or is finalized',
      NEW.season_id, NEW.created_at USING ERRCODE = 'check_violation';
  END IF;
  RETURN NEW;
END $$;
CREATE TRIGGER ledger_season_chk BEFORE INSERT ON ledger_entries
  FOR EACH ROW EXECUTE FUNCTION ledger_check_season();
```

Key decisions

Of the five decisions, DM and the push queue started from my own question and request, the auto-confirm batch was changed by the client planner's answer, and the relation cache and tier rule are the agents' recommended defaults.

DM built in-house, its messages stored in Postgres D10 My input

Problem

The parties to a trade meet in DM, and every change in a trade's state has to leave a system message there. On Oct 11 I asked whether an external API would be more stable and cheaper to run than building and operating chat ourselves, and pointed out that building it would add a message queue to the stack.

Options considered

External chat SaaS

Managed real-time service (API Gateway WebSocket, AppSync, managed pub/sub)

✓ Build it: send over REST, socket for server-to-app notifications only

Decision

Built in-house. All that was needed was one-to-one text and images, and trade system messages had to be written together with the trade transaction. An external SaaS would have charged monthly by expected user numbers. Each room has a sequence number (seq), and locking the room row fixes the commit order, so read, unread and catching up on missed messages all come down to one integer. Sending goes over REST; socket.io is used only for the server to notify the app. Sending over the socket and long-polling were also considered and rejected: sending over the socket would mean rebuilding validation, idempotency and error handling, and long-polling had a sticky-session problem. A DM push is written as a notification row that never shows in the notification list, so it rides the same outbox as every other notification.

Result

The trade-confirmation transaction writes the DM system card as well. If confirmation fails, no card is left behind. The message record lives in PostgreSQL, and the design sends a push when the app is closed (FCM is on hold).

Cost & remaining risk

Sends within the same room are handled one after another because of the room row lock. The ADR notes this is harmless for a two-person conversation.

Push via notification rows as an outbox, on three BullMQ queues over the

existing Redis ADR-0003 My input

Problem

Notifications can arrive in bursts, for example when a tally period ends or when marketing pushes held overnight are released in the morning. On Oct 14, at the design stage, I asked for a message queue to handle push bursts.

Options considered

A separate broker such as Kafka, RabbitMQ or SQS

SQS + Lambda, or SNS mobile push

Call FCM straight from the API and retry

✓ BullMQ on the existing Redis, notification rows as outbox, three queues

Decision

BullMQ on the Redis already in use. The SQS family adds AWS services and points of operation, and the 5-minute digest, delays and rate limits would still have to be built by hand. Kafka was judged too much before there is any streaming pipeline. Calling straight from the API cannot stop slow responses and exhausted quotas during a burst. So the notification row itself became the outbox. Feature code only calls NotificationService.create(tx, …), and the worker splits sending across three queues (immediate, 5-minute digest, bulk in 500-token chunks), each with its own rate limit. The 08:00 release of held pushes and the bulk send at the end of a tally period are spread over 10 minutes.

Result

There is no cron at all inside the API process. The 16 recurring jobs also run in the worker on BullMQ's job scheduler, with schedules stored in Redis, so several workers share one schedule. The actual FCM connection was put off because the store developer accounts came late; until then the push port runs on a console adapter.

Cost & remaining risk

With Redis holding the queues, memory eviction has to stay off (noeviction), so large caches cannot live in Redis (next card). The ADR says to revisit this past a million users, when services are split, or when other channels start sharing the pipeline.

Second- and third-degree cache as one array row per user ADR-0002

Agent default · no sign-off on record

Problem

The feed reaches second- and third-degree contacts, so one changed relationship touches the caches of hundreds to thousands of people. Feed ordering had to finish in a single SQL join with the posts table.

Options considered

Redis cache | One row per (user, contact) pair | ✓ One array row per user

Decision

One array row per user. In Redis the feed could not be ordered in one join with the posts, and the shared Redis, with eviction off because of the queues, could not hold a cache measured in gigabytes. Pair rows were estimated at 30 to 100 million rows even assuming 10,000 users, and one changed relationship would mean deleting millions to tens of millions of rows. With arrays, it ends at setting computed_at to NULL on the few thousand affected rows. Invalidation happens in the same transaction as the relationship change, with upserts in id order to avoid deadlocks. Recomputing happens on read; if a recompute is already running, the reader waits up to 300 ms, then uses the previous array and queues a job.

Result

The transaction that changes a relationship only marks the affected rows. The heavy computation happens the next time that user reads the feed.

Cost & remaining risk

A reader may wait up to 300 ms for a recompute. The ADR sets a ceiling: if p95 goes over target on the densest networks, switch to full stale-while-revalidate.

Auto-confirm as a 5-minute batch; confirmation in one transaction

`ADR-0005` Default → revised on a stakeholder answer

Problem

Confirming a trade changes several things together: the trade's completion, automatic cancellation of other trades on the same post, trade and broker points, the contributor record and the DM system card. The buyer's confirmation and the 7-day auto-confirm batch could also reach the same trade at once. If only part of it landed, the points would drift from the trades.

Options considered

Schedule auto-confirmation as a BullMQ job per trade

✓ A batch every 5 minutes finds overdue trades and confirms them

Decision

Auto-confirmation runs as a 5-minute batch. A job per trade means keeping the queue and the database in agreement; a batch gives the same result however many times it runs. Every trade transaction takes its locks in one order: users → posts → trades → DM rooms. Confirmation does all of it in one transaction, and any failure rolls everything back. The broker path is walked back from the buyer's first-degree relay towards the seller with a recursive CTE, and the condition that relay times keep decreasing ends any cycle. Counting daily limits in a counter table or a Redis counter was rejected because it could

drift from the ledger, and a per-user advisory lock was not used because the users row lock already gives the same serialization.

Result

Following the answer to planning question Q6-01 on Oct 19, the broker path and degree are fixed the moment the buyer taps "start a trade", ahead of confirmation. That stops anyone changing relationships mid-trade to inflate the total, or cutting in just before confirmation. On Oct 23 a path went in for operators to claw back trade points when they rule a trade abusive, tied to sanctions.

Cost & remaining risk

Relays made in the same millisecond cannot be told apart by time alone, so a visited set and a depth cap of 50 stay as a safety net.

Tiers by "how many people have more points than me" ADR-0006

Agent default · no sign-off on record

Problem

Tiers come from the distribution of points within a tally period. They had to make sense even with few active users, as at the start of a period, and the boundaries move as points change.

Options considered

Rank percentile: rank / n ≤ boundary

Store the tier on the user row

✓ Compare against boundaries using the number of active users with more points in the tally period

Decision

The third. With rank percentile, a single active user lands in the lowest tier, and with fewer than 100 the top tier stays empty: a cold-start problem. Storing the tier on the user row would mean updating everyone each time the boundaries moved, once a minute. Ties go up to the higher tier, and first place is always in the top tier. Every 60 seconds the worker swaps in the whole distribution and ranking as a Redis snapshot.

Result

However many users are active, first place is in the top tier and equal points land in the same tier. No user rows change; only the snapshot is replaced.

Cost & remaining risk

Near a boundary, a tier change can show up to a minute late.

One notification, end to end

A notification is written as a row in the same transaction as the business data, and a separate worker process sends it through queues and rate limits.

1. **Business event** ☐System
   Feature code only calls the notification service's create(tx, ...)
2. **Notification row as outbox** ☐System
   Same transaction as the business data; DM pushes use the same path as hidden rows
3. **Pick a queue** ☐System
   Immediate · 5-minute digest · bulk in 500-token chunks
4. **Rate limits and quiet hours** ☐System
   Per-queue limits; marketing pushes held overnight (default 21:00–08:00), released over 10 minutes
5. **Worker → push port** ☐External
   FCM on hold until the store accounts arrive; console adapter meanwhile

Quality and verification

Each of the 25 top-priority UX issues got a failing test first and was fixed, and all 2,583 automated tests passed when rerun on Nov 10.

The 2,583 automated tests were rerun on Nov 10 and all passed: 991 backend unit tests (100 Jest suites), 359 admin unit tests (45 Vitest files) and 1,233 app widget and unit tests (130 Flutter files). For the app, only test files tracked in the repo were counted. The same three numbers are recorded as passing in the Oct 29 UX audit report.

The e2e suites were not part of the Nov 10 rerun. The 54 backend e2e spec files (806 it blocks, counted statically) need Postgres and Redis. The single app integration test file exists to capture the screen gallery.

Review was done by agents. Between Oct 14 and 29 the security reviewer ran 16 times and the code reviewer 25 times, and the spec-coverage audit ran once on Oct 19. The API contract checker was never called. The ADRs cite review findings 25 times and record 22 known limits. There was no CI and no PR, and there are 0 merge commits.
Hard-to-undo commands are blocked in configuration. The permission settings are identical, character for character, to the earlier project's, blocking force push, reset --hard, rm -rf and reading .env. An operating note in the README records that Prisma's AI safety guard stops agents running the database-reset command (migrate reset).

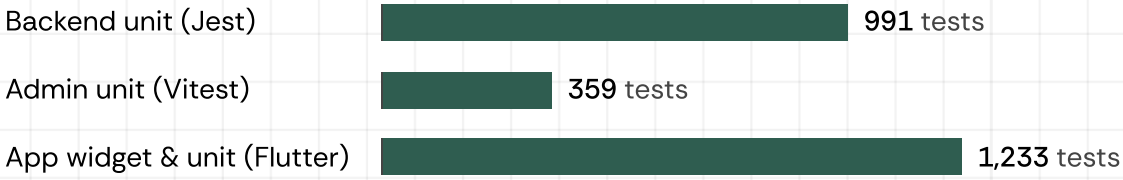
Every external integration goes through one of 7 ports. Three have live adapters (SMS, address search, LLM release notes); the rest are mocks, console or local disk. The development environment makes 0 external calls, and the production configuration refuses the development adapters.

The Oct 28 UX audit reran the existing widget tests under 7 screen conditions (narrow width, large text, keyboard) and collected every overflow. It found 110 issues, sorted into 25 top-priority issues, 19 needing a planning answer, and others. Each item got a failing test first. All 25 top-priority issues were fixed, and the 33 places that overflowed with large text went to 0 under the 6 conditions without a keyboard. The 23 code review findings on the fixes were settled as 14 fixed, 6 not a bug and 3 out of scope.

Automated tests, rerun

All 2,583 unit and widget tests across the three codebases passed on a rerun; the backend e2e suite, which needs a database, was not part of the Nov 10 rerun.

Source: Run on 2026-11-10, all passing. App: tracked test files only. The 54 backend e2e files need Postgres and Redis and were not part of the rerun.



One feature, start to finish

This follows one story from the planning documents to the tests: trade confirmation and broker points.

1 **Planning — documents to spec stories (Oct 10)**

The trade process, point rules and state transitions in the planning documents became the spec's trade stories (USR-TRD-01–07, SYS-TRD-01–06) and two broker-point stories.

2 **Questions — six items settled (Oct 13–24)**

Broker points go to the people on the relay path (Q-005). To prevent abuse there is a cooldown per pair, a per-tally-period count for the same broker and seller, and a per-period cap (Q5-01). Daily limits are counted separately (Q5-08), and the path's starting point was defined (Q5-09). The path and degree are fixed at "start a trade" (Q6-01), and what happens to a withdrawn user's share was settled (Q7-06).

3 **Design — ADR 0005 (Oct 16)**

Set the trade state machine, one-transaction confirmation, a cycle-safe broker path, the lock order and auto-confirmation by a 5-minute batch.

4 **Backend (Oct 16)**

One commit brought the state machine, the post row lock on "start a trade", limits on completion requests, one-transaction confirmation, automatic cancellation of other trades on the post, cancellation reasons that differ by viewer, and three batches (auto-confirm, notice 2 days before, reminder after 3 days). Four trade e2e specs came with it.

5 **App (Oct 16)**

Built the "start a trade" button, the seller's bottom bar and counterpart picker, and completion request, confirm, deny and cancel in the DM trade bar.

6 **Rule changes (Oct 19)**

Following the answers: path and degree fixed, the pair cooldown, a per-period maximum for the same seller, a broker cap per period and separate daily limits. The app gained a deny confirmation dialog and a seller-only deny card showing the remaining count.

7 **Operator clawback (Oct 23)**

When an operator rules a trade abusive, its trade points are clawed back, tied to sanctions.

8 **UX audit (Oct 28–30)**

The buyer's only way to cancel a trade was a hidden swipe. I adopted the recommendation to show a "cancel trade" button in the expanded trade bar (UXQ-07), committed on Oct 30.

9 **Tests (Nov 10)**

All 991 backend unit tests, trades included, passed. The four trade e2e specs exist but were not part of the Nov 10 rerun.

Retrospective

After the session's scratch folder was wiped twice, long-running work files moved somewhere durable, and CI is still on the to-do list.

The session's scratch folder was wiped twice (early October and Oct 27). The second time took the reference images, lists and rules in use for the Figma screen work. Since then, files for work that runs for hours live in a git-excluded folder inside the repo.

Commit dates are not work dates. Under the rule "commit only when asked", each phase was committed at its end, split by feature, so 55 commits landed on Oct 14 alone. The dates on this page come from commits and may trail the actual work.

Staging was built on Oct 25 in the cheapest configuration and deleted the same day when the internal release was postponed; the budget alerts and quota requests were kept. Building it turned up a VM startup script that ran under POSIX sh, where a bash-only option broke the first boot.

The development environment had two traps. The Dockerized admin tool could not sign in because the build arguments were not passed, which baked an empty API address into it; adding the build arguments to compose fixed it (Oct 26). The mock SMS never logged the code and Redis held only its hash, so no person could see a verification code; a fixed code now works only in development and with the mock SMS.

There is still no CI. None of the four repos has a workflow file, and the CI/CD phase on the to-do list has not started. Moving the UX audit harness, currently local only, into CI is also left as follow-up work.

How I work

This project was also built with a role-based AI agent team. The team and its gates differ from project to project; this one's are under "My role" above. Below is the method as written up while building the EWA platform.

See how I set up and ran the work on the 2026 EWA platform →

https://jake-jo-portfolio.vercel.app/en/process/

Contact

whxoans@gmail.com
+82 10-9233-7440