Earned Wage Access fintech platform, Vietnam

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

Built the employee app, the operator CMS and the employer and partner portals from scratch over five months, then moved the running v1's data onto the v2 production server.

Data migrated	5 months	Matched	7,080
v1 → v2, Nov 7	first commit to the migration	loaded vs expected totals	asserting tests, from run records

Results

Within five months, v1's production data was on v2, and the reconciliation of the loaded counts and amounts against the values expected from v1 matched.

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

Built the employee app (iOS and Android), the operator CMS and the employer and sales-partner portals in three languages. The API and the web run on the staging and production servers, and the app has a build pointing at the production server.

Moved v1's production data to v2 on Nov 7; after the transactions that reached v1 later were added on Nov 8, reconciliation still matched (its scope is on the "The v1 → v2 data migration" decision card). Requests from the v1 app now get an update notice, which retired the v1 app.

Integrated two PSPs. The first end-to-end sandbox transfer succeeded on Aug 23, and on Oct 30 the production connection and a healthy status were confirmed. Failover keeps withdrawals going when one PSP is down, and an ambiguous result is designed to stay with that PSP so two PSPs don't both pay.

All 7,080 asserting tests come with a run record. Backend unit, web and app tests all passed on Nov 10; in the Nov 3 full run of the backend tests that call the API end to end against a real database, 1 of 1,508 failed (it passed on rerun).

Period

Jun 2026 – present

Role

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

Team

Me plus a 13-role AI agent team

Platforms

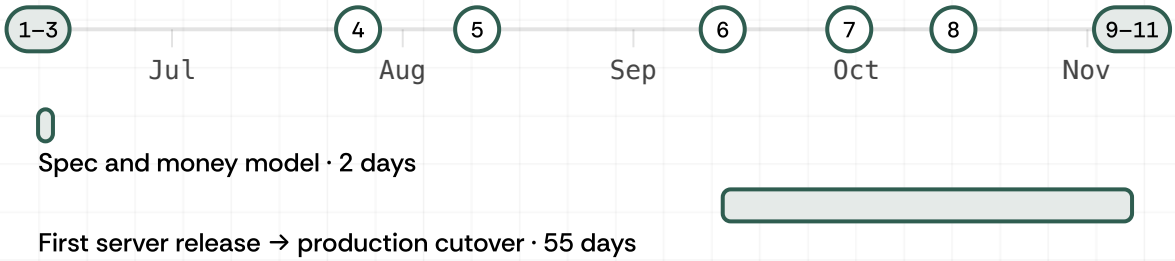
iOS and Android app, web CMS and portals, API server (3 languages)

Context

An earned wage access operator in Vietnam, with a v1 built by another developer already in production

From first commit to production migration

The spec and the money model were settled in the first three days; getting from the first server release in September to the production data migration in November took two months.



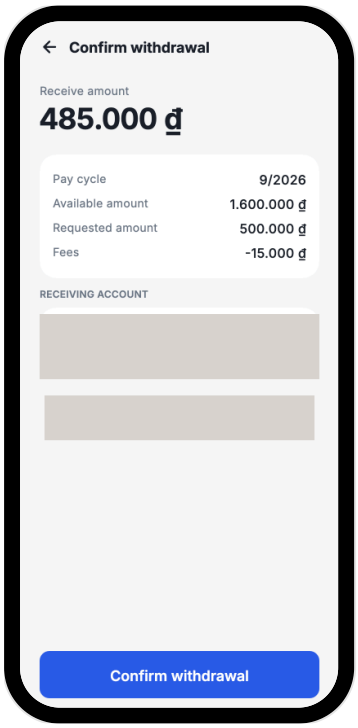
1. Jun 13, 2026 — **Spec v0.2 and the first ADR**
   One spec merged from the v1 and v2 plans
2. Jun 14, 2026 — **Four repos; agent team starts**
   13 roles, 8-stage pipeline
3. Jun 15, 2026 — **Money data model**
   Integer amounts, append-only ledger, withdrawal transaction
4. Jul 26, 2026 — **Staging environment**
   VPS + Caddy
5. Aug 11, 2026 — **Live adapters for both PSPs**
   First end-to-end sandbox transfer on Aug 23
6. Sep 13, 2026 — **PSP failover; first server release 0.0.2**
   App force-update rule the same day
7. Sep 30, 2026 — **PII encryption; v1 migration tool**
   First rehearsal on real v1 data, rolled back
8. Oct 14, 2026 — **Production stack**
9. Nov 7, 2026 — **Production data migration, v1 → v2**
   Run after a rollback rehearsal that morning

10	Nov 8, 2026	**Later transactions moved; v1 app retired** Reconciliation matched; v1 app requests get an update notice
11	Nov 10, 2026	**Now (ongoing)** Server 0.4.22 after 43 releases; 199 logged decisions

Screens

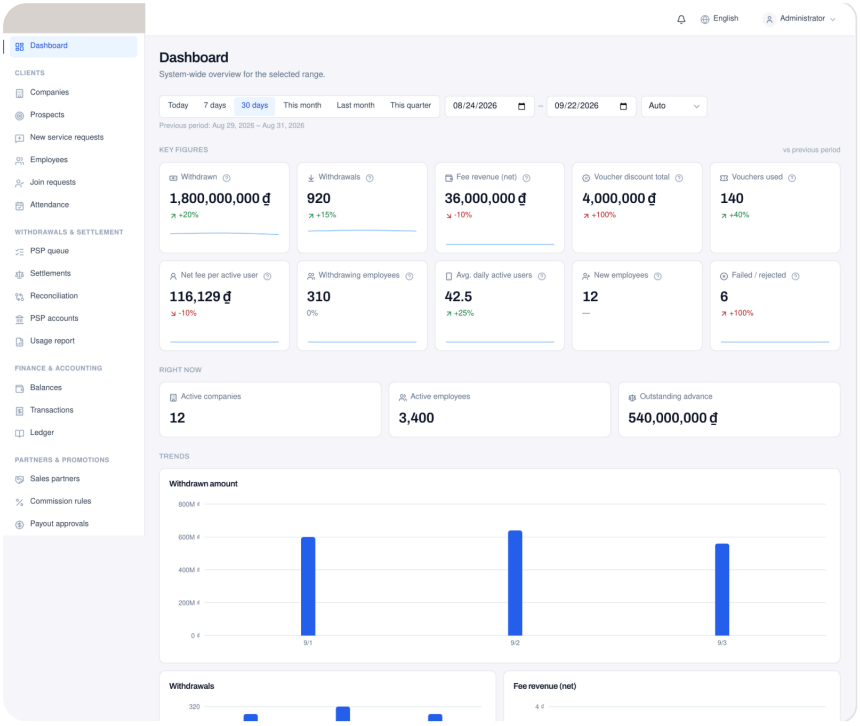
Withdrawal confirmation

Pay cycle, available amount, requested amount, fees and the amount received on one screen. Receiving account details masked.



Operator CMS dashboard: key figures such as withdrawn amount, withdrawal count and fee revenue, with period trend charts

Mock data; logo masked.



Manual withdrawal approval queue

Request time, employee, company, amount, net payout and waiting time sit on one row, with approve and reject buttons per row. Mocked API data; logo masked.



Problem and context

The job was to rebuild, from scratch, a service that moves real money, next to a v1 that was already running.

Workers in Vietnam who need money before payday turn to high-interest loans or to manual pay advances from their employer. Those manual advances also eat up the payroll staff's time.

With Earned Wage Access, workers draw wages they have already earned through an app, and the employer deducts the amount from the next

payroll and settles with the operator. Workers pay a fee per withdrawal. For the employer it is a benefit that costs nothing.

Six actors work on the same flow of money, each with their own screens and permissions: the operator's admin, finance and HR staff, employers, employees and sales partners.

Real money moves, so a double payout or a false success was not acceptable. Regulation required the data to stay on servers inside Vietnam. A v1 built by another developer was already in production, and v2 was a new system built from scratch. The rule for the move was that no v1 data is ever deleted.

One pay cycle

The operator pays out first, the employer repays it from the next payroll, and the employee only pays a fee per withdrawal.

① **Work** ○Person
Wages already earned become available to withdraw

② **Request** ○Person
Pick an amount and a voucher in the app; extra authentication

③ **Payout** □External
Operator's PSP account → employee's bank account, minus the fee

④ **Fee invoice** □System
VAT e-invoice for the fee; vouchers can reduce it

⑤ **Payday deduction** ○Person
Employer submits a settlement file: salary minus EWA deductions

⑥ **Settlement approval** ○Person
Operator approves, then employer; processed after both (auto-processing off)

⑦ **Settlement check** □System
Status monitoring and unsettled alerts

Loops back
↻ ⑦ Settlement check → ① Work: next pay cycle

My role

I designed and ran the team and its pipeline, and made it the rule that releases to production run on my instruction. I read and approved the code on the money paths myself, and the code the agents wrote is mine to answer for once it ships.

I designed the 13-role AI agent team and its 8-stage pipeline (intake → architect → design → develop → QA → review → integrate → document), committed it on Jun 14, and ran the team from then on. I am the only git author, and 1,853 of the 2,130 commits (87%) carry an AI co-author trailer.

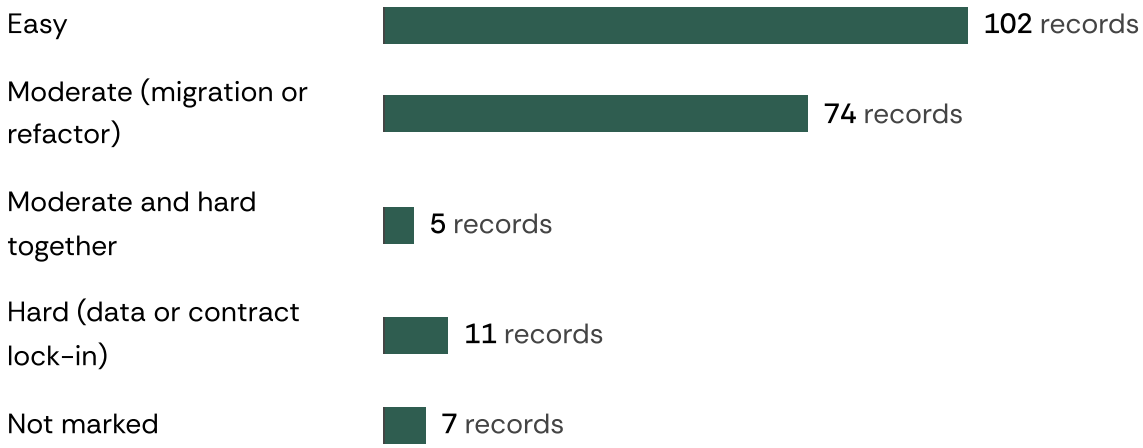
I set the decision mode: so that work never stalls, go ahead on a recommended default, log it in the decision register, and review it later. 68 of the 199 rows (34%) carry a person's confirmation or input, 37 of them with my name. Of the 11 hard-to-reverse decisions, 3 (D-036, D-159, D-161) were confirmed when logged and 8 went ahead on the default. The answers in the stakeholders' decision sheet of 3 July went in together as decisions D-128 to D-131 on 4–5 July.

The rule is that push approvals, production deploys and store submissions run on my instruction; agent sessions carried them out. The production data migration on Nov 7 ran after a rollback rehearsal that morning, and the migration tool does not commit if any reconciliation item differs. The migration rule "no v1 data may be deleted" is in the decision register (D-173) as my input.

How hard the 199 recorded decisions are to reverse

192 of the 199 decisions are tagged with how hard they are to reverse; 11 are hard, locking in data or contracts.

Source: Counted row by row from the reversibility column of the decision register (D-001 to D-199). The 5 rows marked both moderate and hard are grouped on their own.



Who did what

I designed and ran a 13-role agent team, and kept decision sign-off and input and the go-ahead for production myself.

Me

1 person on the build team (me)

Designed the 13-role team and the 8-stage pipeline (Jun 14)
Decision sign-off or input: 68 of 199 rows marked by a person, 37 of them with my name (13 confirmed · 24 input)
Migration rule: no v1 data may be deleted
Go-ahead for production: push approval, production deploy, store submission

Agents

9 roles 87% commits co-authored by AI (1,853 of all 2,130)

Lead: orchestrator
Design: architect, designer
Build: backend, web, app developers
QA, DevOps, docs

Review gate

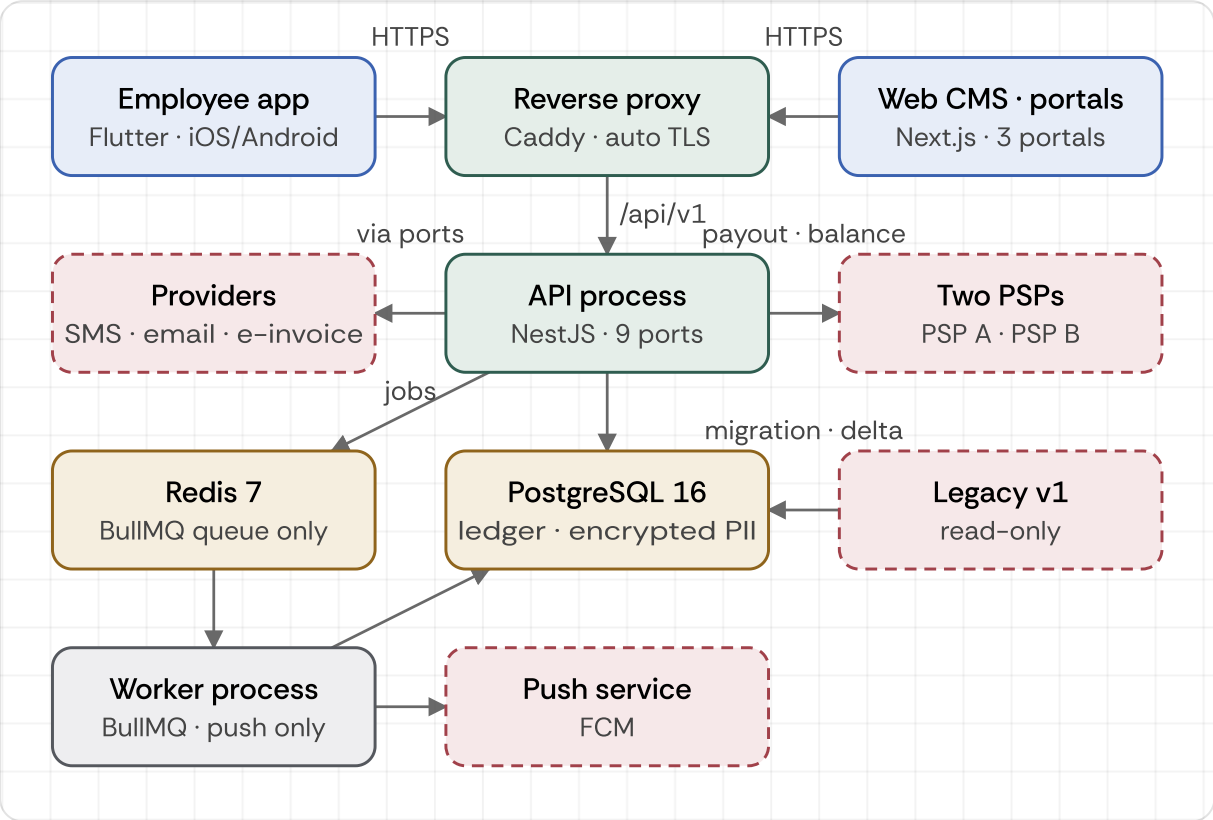
4 reviewers (9 + 4 = 13 roles; 3 in the pipeline)

30 fix commits mention security review or a security fix (Jun 14 – Oct 29)
Three reviewers in the pipeline: security (when money, auth or personal data changes), code (always), API contract (when the contract changes)
PSP failover: fixed 4 findings from 3 security reviews

System architecture

System architecture

The app and the web share one API, and only the API process moves money; the worker just sends push notifications.



Employee app

The Flutter app for employees (iOS and Android): sign-up, the withdrawal limit, withdrawals (two-factor, biometrics, PIN), vouchers, map-based check-in and on-device ID-card text recognition. 45 screens. Its models are written by hand.

Reverse proxy

Caddy adds TLS automatically. On staging a single domain sends only /api/v1 to the API; production splits the API and CMS domains. The proxy also serves the CMS pages.

Web CMS · portals

The operator CMS, the employer portal and the sales-partner portal in one Next.js app, 123 pages. API types are generated from the OpenAPI snapshot (604 operations).

Providers

SMS codes, email, VAT e-invoices for the fee, and business-registry lookups. Every port has a mock and a real adapter, so development makes zero external calls. Payments have no path that falls back to a mock.

API process

42 NestJS modules, 96 controllers, 605 routes. The 18 kinds of scheduled job (settlement reconciliation, voucher issuing, e-invoices, store-version checks, contract expiry and more) run only in this process. Every external call goes through one of 9 integration ports.

Two PSPs

Two PSPs. Requests carry an RSA signature, and one PSP reports results through a callback webhook. A balance query every 2 minutes tracks health and drives failover (D-142).

Redis 7

Only the BullMQ queue broker.

PostgreSQL 16

PostgreSQL 16 with Prisma: 112 models, 150 migrations. Money tables are an append-only ledger, and six employee PII fields are encrypted at rest.

Legacy v1

The previous production system. The migration tool only reads from it. One full migration on Nov 7, then a delta on Nov 8 for what came in between.

Worker process

A separate process started from the same code. It only sends push notifications taken from the queue. Scheduled jobs are switched off here, so no money-moving job ever runs in two processes at once.

Push service

FCM. Only the worker calls it.

`src/integrations/payment-router.ts` · Only a returned decline or outage moves on to the next PSP; an exception stays on that PSP as pending. (comments condensed from the source)

```
  async payout(
    plan: PaymentPartner[],
    req: PayoutRequest,
  ): Promise<RoutedPayout> {
    type Hop = { partner: PaymentPartner; result: PayoutResult };
    const attempts: PayoutAttempt[] = [];
    let last: Hop | null = null;
    let declined: Hop | null = null;
    for (const partner of plan) {
      const result = await this.hop(partner, req);
      attempts.push({ partner, status: result.status });
      last = { partner, result };
      // Only a returned 'failed' or 'unavailable' goes to the next PSP.
      // A throw inside hop() is ambiguous (it may have paid), so it
      // comes back as 'pending' and the walk stops on that PSP.
      if (result.status === 'success' || result.status === 'pending') {
        break;
      }
      if (result.status === 'failed') {
        declined = last;
      }
    }
    // …
  }
```

Key decisions

The three money-path decisions started as agent defaults, two then changed by the stakeholder sheet and security-review findings; PII encryption and the v1 migration started from my input.

Integer money, append-only ledger `ADR-0004` Agent default · no sign-off on record

Problem

The Vietnamese dong has no minor unit, and one company's monthly settlement can exceed the 32-bit integer limit (about 2.1 billion).

Options considered

Full double-entry: the right long-term model for a licensed fintech, but heavier than the current stories need and slower to deliver

✓ Single-entry, append-only ledger, with the fields for a later move to double-entry in place

Decision

The ledger is single-entry but append-only: no updates or deletes, and a cancellation is a REVERSAL row. Account, direction, reference and batch fields are there from the start, so a later move to double-entry needs no rewrite of history. Amounts are stored as BigInt whole dong and rates as integer basis points (60% = 6000). The dong has no minor unit, so Decimal and float are banned from money paths, and a monthly settlement can exceed the 32-bit limit, so the type is BigInt rather than Int.

Result

All limit and fee math is integer arithmetic. Because the ledger is append-only, race tests could later assert "ledger sum = 0" as an invariant.

Cost & remaining risk

If strict double-entry is ever required, it takes a migration that adds a parent transaction table and a balance constraint (the ADR rates it moderate). The ADR is still marked provisional.

One transaction for the withdrawal, payout after commit `ADR-0006`

Default → revised on a stakeholder answer

Problem

A double click or a race could pay twice, and no DB lock may be held during a network call.

Options considered

Saga / outbox: not needed in the first wave, since booking a withdrawal happens entirely inside one database

✓ Single Postgres transaction + payout after commit

Decision

A withdrawal is booked in one Serializable transaction. Inside it, the cycle's withdrawals are summed again to compute the limit, the employee's status is rechecked, and the withdrawal, voucher and ledger rows are written. If two requests overlap, PostgreSQL rolls one back with a serialization failure; after up to 5 retries it returns 409. The ADR text says FOR UPDATE locking, but the implementation has worked this way from the start. The PSP is called after commit, outside the transaction, and a short second transaction applies the result. The Idempotency-Key is unique globally and forever: the same key returns the original result, and a different employee or amount gets 409.

Result

An e2e test checks that when one employee confirms twice at once with different keys, exactly one succeeds. On Sep 4 a race was found and fixed: two manual rejections arriving at once wrote the reversal twice. The fix is a conditional claim that updates only when the status is the expected one and continues only when exactly one row changed,

plus race e2e tests asserting that the ledger sums to zero. Security review approved the fix, and on Sep 13 the same pattern went onto the sibling payout paths it had flagged.

Cost & remaining risk

Instead of row locks, the design relies on Serializable conflict detection: overlapping requests are recomputed, and more than 5 conflicts return 409. Idempotency keys are never garbage-collected and stay unique forever. The ADR said to revisit this design once there were several PSPs; two arrived in September and the ADR was not updated.

Two PSPs with health-based failover D–142 Agent default · no sign-off on record

Problem

One PSP going down blocked every withdrawal.

Decision

Payout results are success, pending, failed or unavailable. Only decline codes on an allow-list count as failed; everything else is pending and is not passed on to the other PSP. The order is the company's PSP, then the other, and only PSPs that are configured, active, healthy and funded are tried. Failed or unavailable moves to the next PSP; pending or an exception stays with that PSP and stops. Health is a balance query every 2 minutes, and two transport errors in a row mark a PSP down. If none is available, the API returns 503 before the two-factor (TOTP) code is consumed and writes nothing to the ledger. Payments have no path that falls back to a mock.

Result

Merged on Sep 13. Three rounds of security review fixed a manual-payout race, a double payout caused by treating unknown codes as failed, flags that came back after being resolved, and payouts stuck in pending for good.

Cost & remaining risk

An ambiguous result is pending: it stays with that PSP until a status query resolves it or an operator closes it, and closing it does not restore the limit. Payments never fall back to a mock: with no usable PSP the API returns 503 before 2FA.

Employee PII encrypted at rest D–172 My input

Problem

v1 had been encrypting some personal data, and a DB dump alone must not expose it.

Decision

AES-256-GCM sits in a Prisma query extension that encrypts on write and decrypts on read, so service code only sees plaintext. Six fields are covered: phone number, account number, account holder, national ID number, date of birth and base salary. A random IV on every field would rule out equality lookups, so only the phone and national ID numbers (the latter since Oct 27), which need lookups, use a deterministic IV, derived from an

HMAC of the field name and value, to keep uniqueness and equality lookup; the rest use a random IV. Only the last four digits of the phone number stay in plain text, for search.

Result

The v1 migration tool commits only after reconciliation confirms that all six fields of every migrated employee are sealed (zero plaintext).

Cost & remaining risk

A lost key cannot be recovered, and partial search works only on the last four phone digits. For the two deterministic fields, the ciphertext alone shows whether two numbers are the same.

The v1 → v2 data migration D–173 My input

Problem

The data in the running v1 had to move to v2. The rule was "no v1 data may be deleted".

Decision

The migration tool fixes only v1's column list and rules, and reads every row at run time. It uses deterministic IDs and stops on any enum value missing from its rules. It loads everything in one transaction, reconciles, and does not commit if anything differs. Reconciliation compares per-company counts, amounts and fees, per-employee payout totals, the number of settled withdrawals, ledger offsets for failed withdrawals, and zero plaintext PII. The expected values are built while transforming the rows read from v1, so it checks the transformed result against what was actually loaded. By default a run is a rehearsal that loads, reconciles and rolls back, and a test fails if any string sent to v1 contains a destructive keyword.

Result

The rehearsals on Sep 30 and the morning of Nov 7 found problems in v1's own data (swapped date fields in attendance records, broken account numbers), and on the tool side withdrawal codes were kept, clashes with numbers already in v2 were handled and unsettled months were listed. Transactions that reached v1 after the one-off migration on Nov 7 were added with a delta migration on Nov 8, and reconciliation matched.

Cost & remaining risk

Reconciliation does not query v1 again to compare against the source. Times, attendance days, notifications and files are not reconciled.

One withdrawal, end to end

Money moves only after the record is committed, and the router is designed to move to the next PSP only on a definite failure; an ambiguous result stays with that PSP so two PSPs don't both pay.

① **Request** ○Person
Amount, voucher, two-factor code; resending the same Idempotency-Key returns the original result

② **Eligibility** □System
Attendance, limit and usage history; company and account checks

③ **PSP plan** □System
Company's PSP first, then the other; only configured, active, healthy and funded. If none: 503 before the code is consumed, zero ledger rows

④ **One transaction** □System
Inside a Serializable transaction: recompute the limit, recheck status, write withdrawal, voucher and ledger; if two overlap, one is rolled back and recomputed (up to 5 times)

⑤ **Pay after commit** □External
Signed request outside the transaction; the same reference on every PSP

⑥ **Classify result** □System
success, pending, failed, unavailable; pending or an exception stays with that PSP and stops

⑦ **Apply result** □System
Short second transaction, payout notice, reconciliation

Loops back
↺ ⑥ Classify result → ⑤ Pay after commit: definite failure or unavailable → next PSP

Quality and verification

Two thirds of the 7,080 automated tests sit in the backend, where the money paths are, and the e2e tests aim at the boundaries of money and permissions.

There are 7,080 automated tests with assertions. The 3,156 backend unit, 1,460 web and 956 app tests were run on Nov 10 and all passed. The backend e2e tests need a real database; the count comes from the full run on Nov 3, where 1 of 1,508 failed and passed when that spec was rerun.

The e2e tests aim at the boundaries of money and permissions. Counted by test name, so approximate: 32 on tenant isolation and cross-company access, 94 expecting a 403, and 45 on idempotency, retries and concurrency.

Every external call goes through one of 9 integration ports. Each port has a mock adapter, so development makes zero external calls. Only the payment port has its path to a mock blocked.

The pipeline's three reviewers (security, code and API contract) ran 24 times since 28 Sep, when records begin. 30 fix commits mention security review or a security fix (Jun 14 – Oct 29). PSP failover (D-142) went through security review three times.

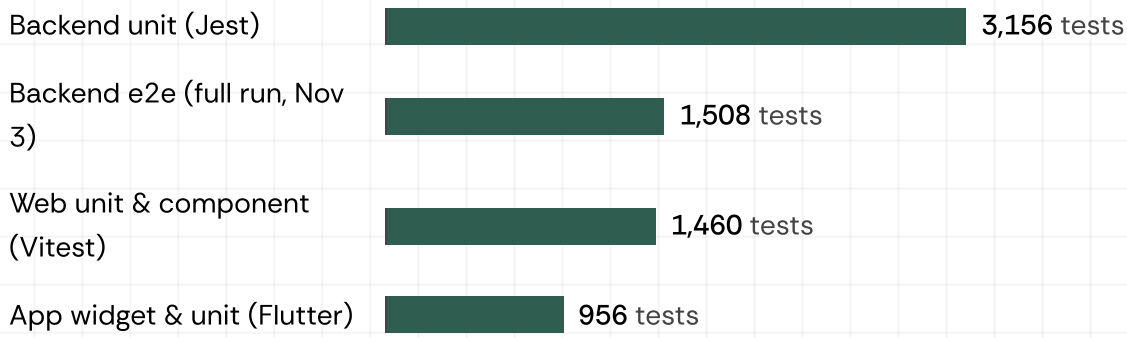
I also built a map of screen states. App golden captures and web Playwright captures drew 131 screens, 597 states and 1,245 transitions as a wireframe map. These captures have no assertions, so they are not counted as tests.

The rule was to write the failing test first: 344 of the 398 non-merge fix commits (86%) also changed a test file. That includes edits to existing tests, so it does not mean a regression test was added each time.

Automated tests by layer

About two thirds of the 7,080 asserting tests sit in the backend, where the money paths are.

Source: Backend unit, web and app run on 2026-11-10 (all passing); backend e2e from the full run on 2026-11-03 (1 of 1,508 failed, passed on rerun of that spec). Capture-only tests without assertions (143 app goldens, 314 web Playwright) are excluded.



One feature, start to finish

This follows one story from the spec to a deploy: the forced app update.

1 **Spec and decision — EMP-SET-07 / ADM-SET-03 → D-143**

An X.Y.Z version rule. A bump to X or Y shows a blocking screen with only a store button; a bump to Z alone shows a dismissible prompt for 5 seconds. If the check fails, the app opens normally (fail-open). The public GET was made a deliberate exception to the "role guard on every endpoint" rule. The trigger was that v1's app had a forced update and v2 did not.

2 **Backend — version registry and a public check API**

One commit added the version registry, a public GET /app-version and an admin PUT with an audit record, together with 3 unit tests and 7 e2e tests. The same commit wrote the reason for the role-guard exception into the project rules file.

3 **Web CMS — latest-version registry page**

The commits went: settings page, then a dialog that confirms the forced update on first registration, then a regenerated OpenAPI snapshot. 7 tests and 1 fix commit.

4 **App — update gate on cold start**

6 widget tests (forced, optional, 5-second auto-dismiss, fail-open on API error, malformed body, no Android row) and 6 unit tests for version comparison. The iOS branch was narrowed with a test variant. Captures of the forced and optional screens came 9 days later in a separate commit, for the wireframes; they are not regression tests.

5 **First use — the rule picks the version on staging**

On Sep 21, removing the invite-code route and blocking account writes meant installed apps had to update. Following the rule, the server went from 0.1.5 to 0.2.0 and the app got a Y bump too. It was a staging round and the build was not in the stores, so CMS registration was skipped on purpose. The production order (store release → server deploy → CMS registration) and why the reverse order is worse were written down as well.

6 **Hardening — the rule enforced in code**

"Register a version only once it is live in the store" was still just a convention. On Oct 26, a job started reading the store version daily at 06:00 (ICT), and registering anything above it now returns 400 (VERSION_ABOVE_STORE). Admins are alerted if the store drops below the forced baseline.

Retrospective

For the flaky e2e suite I set a rule that tells an environment problem from a bug by the shape of the failures, and used it to narrow the cause.

A flaky e2e suite gave me a rule for diagnosis. About 3 failures in 90 runs, a different single test each time. The rule: many failures at once means the environment, one failure that moves around means a bug not yet found. With it, the cause was narrowed from the app to the test transport

(a temporary port opened and closed per request), and the server now binds once.

How I work

This is how I set up and ran the work on the 2026 EWA platform, and how that setup changed across six 2026 projects. Every change request from the client was shipped within 24 hours. I defined an 8-stage pipeline and assigned design, implementation, QA and review to a 13-role AI agent team. The client's stakeholders supplied the requirements and the decision-sheet answers; I confirmed or gave input on decisions taken by default, and gave the go for production.

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