MASFIQUR RAHMAN

AI Platform QA Work Sample | Verification Artifacts

Prepared for: **Senior QA Engineer - AI Platform** (Automation / Hybrid)
Focus: Silent failures | Migration drift | Boundary mismatches | Tests that lie | Async UI
Stack context: Next.js | Supabase/Postgres | Python pipelines | Workflow automations

Portfolio: https://www.masfiqur.com | Loom: loom.com/share/70c7255dc2cf4f328b4451781e8bbc79

14+	Live seam	Postgres	PT overlap
Years QA / verification	Not suite-green alone	Schema & job proof	4+ hrs daily ready

About this sample

Illustrative artifacts showing how I hunt bugs the way your post describes them: prove behavior against live staging data, separate **fired in production** from **latent**, verify migrations on the real schema, and distrust tests that encode the same wrong assumption as the code. Anonymized demos - not client-confidential materials.

Verification principles I own

- Green suite = starting point, never proof - check the seam with live data
- Job/UI success badges mean nothing without row-level outcomes in Postgres
- Migration merged != migration applied - prove with history + information_schema
- Defect reports include blast radius: fired vs latent, evidence, severity
- AI tools amplify speed; I read diffs and form hypotheses myself
- Regression lives where bugs live: boundaries, migrations, async UI paths

Contents

Tip: pages 2-3 show evidence style; page 4 shows process ownership; page 5 shows automation at seams.

1. Silent Failure Hunt - Fired vs Latent

BUG-AI-221 - Scheduled enrichment job reports Success while every record stalls forever

Severity	**Critical**
Blast radius	**FIRED in production** - not latent. ~14.2k rows stuck pending since first bad deploy.
Environment	Staging reproduce + prod read-only SQL confirmation
Seam	Scheduler badge -> job runner -> gate condition -> Postgres row status

Symptom

Ops dashboard showed nightly job Success for 11 consecutive runs. Product still showed stale / pending enrichment. No hard error in the UI.

How I proved it (adversarial)

- Ignored the green badge. Queried row outcomes for last N job_run_ids by status.
- Found status='pending' with null last_error and rising retry_count - gate never cleared.
- Traced gate: required partner_account_id translation missing; condition always false.
- Computed first_seen = MIN(updated_at) on stalled set - mapped to deploy window.
- Classified **FIRED** (prod already impacted) vs latent (would fire on next batch only).

Evidence SQL (staging / prod read replica)

```
SELECT status, COUNT(*), MIN(updated_at), MAX(retry_count)
FROM enrichment_records
WHERE job_run_id IN (SELECT id FROM job_runs WHERE name='enrich_nightly' ORDER BY created_at DESC LIMIT 11)
GROUP BY status;

-- Result: success=0, pending=14211, failed=0 | dashboard still showed Success
```

Expected vs actual

Expected: Job fails or Partial when gate stalls population; dashboard reflects row outcomes.
Actual: Runner marked Success because exception path never threw - silent stall forever.

Fix verification bar I require: same SQL shows pending drain + non-zero terminal success/fail; dashboard metric derived from row outcomes, not runner exit code alone.

2. Migration Verification - Merged != Applied

When a migration merges and CI is green, I still prove production schema reality. Deploy health can match perfectly while the feature ships dead.

Verification checklist (I run this on every DB-touching release)

Step	What I check	Fail signal
1	Migration history table on target DB lists the revision	Missing revision - not applied
2	information_schema.columns match migration intent	Column/type/nullability drift
3	pg_constraint / indexes / unique keys present	Constraint missing - silent integrity risk
4	RLS policies / grants if Supabase roles involved	Policy absent - wrong access or empty reads
5	Function/trigger body (pg_get_functiondef) if used	Old body still deployed
6	Write path smoke on staging using new schema	Feature dead despite green deploy

Example probes

```
-- Was revision applied?
SELECT version, name, inserted_at FROM supabase_migrations.schema_migrations
WHERE version = '20260715_add_revenue_gate';

-- Does production schema match intent?
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_schema='public' AND table_name='revenue_events'
ORDER BY ordinal_position;

-- Constraint present?
SELECT conname, pg_get_constraintdef(oid)
FROM pg_constraint WHERE conrelid = 'public.revenue_events'::regclass;
```

*BUG pattern I flag as Critical: history row present in CI DB only; production missing column; feature UI ships; health check still green. Classification: **latent until first write** or **fired** if beta users already hit the path.*

3. When Tests Passed - and Were Wrong

BUG-AI-188 - Cache guard never matched: two ID spaces compared without translation

Severity	**High**
Blast radius	Latent performance tax in prod - every request paid cold path; data not corrupt
Root cause	Internal UUID compared to external partner ID; unit test used same fake for both

Why the suite lied

Unit tests constructed cacheKey from partnerId and lookup from partnerId - identical fixtures. They encoded the same wrong assumption as production code (no translation layer). Green != correct.

What I changed about the tests

- Split fixtures: internalId != externalId in every boundary test
- Assert translation runs before cache get/set
- Negative: wrong-space ID must miss (or single cold fetch), never silent wrong-hit
- Renamed test to state the invariant: cache key uses internal id after translate
- Added seam test (API/integration) so mocks cannot echo the bug forever

Adversarial fixture sketch

```
internalId = 'usr_01H...' // platform space
externalId = 'ext_9981' // partner space - must NOT equal internalId
expect(translate(externalId)).toBe(internalId)
expect(cache.get(externalId)).toBeUndefined()
expect(cache.get(internalId)).toEqual(expectedPayload)
```

4. Severity Ledger & Adversarial Verification

I can run and extend a systematic bug-hunt: multi-dimension review, adversarial verification, severity ledger. Sample excerpt below (illustrative).

ID	Finding	Dim	Sev	Fired?
L-01	Nightly job Success while rows permanently pending	Async/jobs	Crit	Yes - prod
L-02	Migration in main; prod missing revenue_gate column	Migrations	Crit	Latent to beta
L-03	Cache compare external vs internal ID	Boundaries	High	Yes - cold path
L-04	Unit test fixtures collapse both ID spaces	Tests	High	Latent risk
L-05	UI stuck on Sending after transport error (no reset)	Async UI	High	Staging+prod
L-06	RLS policy allows read but write path 401 without UX	Auth/RLS	Med	Latent

Async UI stuck state - defect sketch

Title	Unguarded await leaves portal in permanent Sending state after transport error
Severity	**High**
Steps	Trigger action on slow network; kill request / force 503; observe button/state
Expected	Error toast; control re-enabled; state machine returns to idle
Actual	Sending... forever; no retry affordance; user force-refresh required
Evidence	Loom + network panel + React state note; blast: beta users on flaky links

How I extend the process

- Add dimensions when new seams appear (AI non-determinism, webhook retries, RLS)
- Require fired-vs-latent on every Critical/High before close
- Tie ledger items to regression probes (SQL + Playwright) so they cannot silently return
- Use AI to draft probes faster - I still own hypothesis, severity, and proof

5. Regression at the Seams (Playwright + SQL)

I put automation where bugs actually live: service boundaries, migration-dependent paths, and async UI. Suite green is necessary; live-data proof is sufficient.

Pre-release beta verification (my bar)

- Drive the feature on staging with realistic data - not empty fixtures only
- Assert UI outcome AND Postgres/API outcome for the same action id
- Force failure modes: 503, timeout, empty RLS result, partial batch
- Confirm migration revision on the environment under test before claiming pass
- Record severity ledger updates for anything found; verify fixes with the same probes

Playwright - async path must recover

```
test('send recovers after transport error', async ({ page }) => {
  await page.route('**/api/workflows/run', route => route.abort('failed'));
  await page.getByRole('button', { name: 'Run' }).click();
  await expect(page.getByText(/failed|try again/i)).toBeVisible();
  await expect(page.getByRole('button', { name: 'Run' })).toBeEnabled();
  await expect(page.getByText(/sending/i)).toHaveCount(0);
});
```

SQL probe paired with UI success

```
-- After UI shows Success for run_id
SELECT status, gate_passed, error_code
FROM workflow_runs WHERE id = :run_id;
-- Pass only if status matches UI AND gate_passed is not silently false
```

Closing

I enjoy finding problems before users do - especially silent ones. Ready for your paid trial bug hunt on a scoped slice: I will read the code, form hypotheses, verify at the seam, and report with evidence and blast radius.

Masfiqur Rahman | Full-Stack QA Engineer | 14+ years
Playwright | Postgres verification | Seam / API testing | 4+ hrs/day US Pacific overlap
www.masfiqur.com