Laporan Review Final — AITI Forms App

Comprehensive Deep Dive: Semua Temuan dari 3 Sesi Review

Ditulis oleh: Claude Sonnet (Anthropic)
Tanggal: 18 Februari 2026
Repo: `formsman6kotakupang` — AITI Global Nexus
File yang Dianalisis: src/worker.js, src/server.js, src/app/routes.js, src/lib/db/bootstrap.js (600+ baris), src/lib/db/sql.js, src/lib/db/client.js, src/lib/security/hash.js, src/lib/security/signature.js, src/lib/http/request-guards.js, src/lib/http/session-cookie.js, src/modules/auth/service.js, src/modules/auth/repository.js, src/modules/auth/middleware.js, src/modules/tenants/service.js, src/modules/tenants/repository.js, src/modules/schools/service.js, src/modules/schools/repository.js, src/modules/questionnaires/service.js (700+ baris), src/modules/questionnaires/repository.js, src/modules/questionnaires/schema.js, src/modules/submissions/service.js, src/modules/submissions/repository.js, src/modules/submissions/validation.js, src/modules/submission/service.js (LEGACY), src/modules/forms/core.js, src/modules/forms/repository.js, src/modules/forms/service.js, src/modules/form/schema.js (LEGACY), src/modules/ai/service.js, src/modules/ai/repository.js, src/modules/ai-prompts/service.js, src/modules/ai-prompts/repository.js, src/modules/analytics/service.js, public/admin/*.html, public/forms/*.html

---

Daftar Isi

---

1. Kondisi App Saat Ini — Gambaran Jujur

App ini adalah sistem form feedback multi-tenant berbasis Cloudflare Workers + Neon PostgreSQL. Secara fungsional **sudah berjalan di production** dan sudah melayani beberapa sekolah. Fiturnya lengkap: form builder dinamis, analytics, AI analysis via Gemini, PDF export, multi-tenant, role-based access.

Yang perlu dipahami: App ini lahir dari satu sekolah (SMAN 6 Kota Kupang) lalu berkembang menjadi platform multi-tenant. Transisi itu masih 70% selesai. 30% sisanya adalah technical debt yang bisa meledak kalau tidak ditangani sebelum scale.

Dua sistem dalam satu repo:

Sistem	Stack	Dipakai?
Legacy Express	`src/server.js` + `src/app/routes.js`	❌ Tidak di production
Production Hono Worker	`src/worker.js`	☑ Aktif
Proxy lama	`worker.js` (root)	❌ Zombie file

2. Masalah Kritis — Harus Diperbaiki Segera

🟤 KRITIS #1: PBKDF2 dengan 10.000 Iterasi — Kelemahan Keamanan Nyata

File: `src/lib/security/hash.js`

```
const DEFAULT_PBKDF2_ITERATIONS = 10000;
```

NIST SP 800-132 (2023) merekomendasikan minimum **600.000 iterasi** untuk PBKDF2-SHA256. Dengan hardware GPU modern, 10.000 iterasi bisa di-crack dalam hitungan menit dengan serangan brute force offline.

Ini melindungi password semua admin di seluruh platform. Ini bukan masalah kecil.

Solusi:

```
const DEFAULT_PBKDF2_ITERATIONS = 600000;
```

Lalu re-hash password saat login berikutnya:

```
// Di auth/service.js setelah verifyPassword berhasil:
if (storedIterations < 600000) {
  const newHash = await hashPassword(password, user.password_salt, 600000);
  await updateUserPassword(env, { userId: user.id, passwordHash: newHash, ... });
}
```

🟤 KRITIS #2: Tidak Ada Rate Limiting di Login Endpoint

File: `src/worker.js` (route handler untuk `/forms/admin/api/login`)

Endpoint login tidak ada rate limiting. Siapapun bisa brute force password admin tanpa hambatan:

```
# Ini bisa dijalankan tanpa batas:
for i in $(seq 1 10000); do
  curl -X POST https://app.com/forms/admin/api/login \
    -d '{"email":"admin@school.com","password":"'$i'"}'
done
```

Solusi A (Cloudflare WAF — paling mudah): Buat rule di Cloudflare Dashboard:

- Path: `/forms/admin/api/login`
- Method: POST
- Limit: 5 requests per IP per 60 detik
- Action: Block (429)

Solusi B (In-Worker dengan KV):

```
// Di worker.js sebelum route login
const loginKey = `ratelimit:login:${clientIP}`;
const attempts = Number(await env.KV.get(loginKey) || 0);
if (attempts >= 5) return c.json({ error: 'Too many attempts' }, 429);
await env.KV.put(loginKey, String(attempts + 1), { expirationTtl: 60 });
```

🟠 KRITIS #3: Analytics In-Memory Tanpa Limit — Bisa Crash Worker

File: `src/modules/questionnaires/service.js`

```
// getTenantQuestionnaireAnalyticsSummary
const responses = await listQuestionnaireResponsesForAggregation(env, filters,
null);
// null = SEMUA response diambil ke memory JavaScript!
const distribution = computeDistribution(fields, responses);
```

Cloudflare Workers punya **memory limit 128MB** dan **CPU limit 50ms per request** (free) atau **30s** (paid). Kalau ada tenant dengan 50.000 response, semua akan di-load ke memory Worker, masing-masing response adalah JSON object dengan `answers`, `respondent`, `payload` — bisa 500KB+ total, bahkan bisa MB.

`computeDistribution` sendiri adalah fungsi 200+ baris yang iterasi setiap response untuk menghitung statistik. Ini O(n) complexity — semakin banyak response, semakin lama.

Solusi Segera (tambah hard limit):

```
// Ganti null dengan batas aman
const responses = await listQuestionnaireResponsesForAggregation(env, filters,
10000);
```

Solusi Ideal (jangka menengah): Pindahkan computation ke SQL aggregation di database.

🟠 KRITIS #4: `session.is_active` — Naming Menyesatkan (Tapi Bukan Bug Fatal)

File: `src/modules/auth/service.js`

```
const session = await getSessionByTokenHash(c.env, tokenHash);
if (session.revoked_at || expired || !session.is_active) {
  await revokeSessionByTokenHash(c.env, tokenHash);
  return null;
}
```

Setelah baca `auth/repository.js` secara mendalam:

```
// auth/repository.js - getSessionByTokenHash
SELECT s.id, s.user_id, s.expires_at, s.revoked_at, u.email, u.is_active
FROM sessions s JOIN users u ON u.id = s.user_id
```

`session.is_active` sebenarnya adalah `users.is_active` (apakah akun user aktif), bukan kolom dari tabel `sessions`. Ini **bukan bug yang menyebabkan semua user logout** — ini pengecekan yang valid.

Tapi naming-nya sangat menyesatkan. Developer baru akan mengira ada kolom `is_active` di tabel `sessions` yang tidak ada. Ini adalah **ticking time bomb** — kalau ada yang refactor query dan lupa include `u.is_active`, pengecekan ini akan silently hilang.

Solusi:

```
// Di auth/repository.js — ubah alias kolom
SELECT s.id, s.user_id, s.expires_at, s.revoked_at,
       u.email,
       u.is_active AS user_is_active  -- rename untuk kejelasan
FROM sessions s JOIN users u ON u.id = s.user_id
```

```
// Di auth/service.js — gunakan nama yang jelas
if (session.revoked_at || expired || !session.user_is_active) {
```

KRITIS #5: Tidak Ada CSRF Protection

File: `src/lib/http/request-guards.js`

```
// monitorAdminOrigin — HANYA LOGGING, BUKAN PROTECTION
export async function monitorAdminOrigin(c, next) {
  console.warn(`[ORIGIN_MONITOR] origin=${origin} host=${requestHost}`);
  await next(); // ← tetap lanjut meski origin berbeda!
}
```

Fungsi ini hanya memonitor, tidak memblokir. Admin endpoints bisa dipanggil dari domain manapun selama ada session cookie yang valid. CSRF attack bisa dilakukan dengan membuat user yang sudah login mengunjungi halaman berbahaya yang mengirim request ke admin API.

Solusi:

```
export async function enforceAdminOrigin(c, next) {
  const origin = c.req.header('origin');
  const host = new URL(c.req.url).hostname;
  if (origin) {
    const originHost = new URL(origin).hostname;
    if (originHost !== host) {
      return c.json({ error: 'Forbidden: Cross-origin request' }, 403);
    }
  }
  await next();
}
```

---

3. Masalah Serius — Perlu Perhatian Bulan Ini

⚠ SERIUS #6: `auth/service.js` — Service Layer Coupling ke HTTP Framework

File: `src/modules/auth/service.js`

```
export async function loginWithEmailPassword(c) {
  // c adalah Hono Context object
  const payload = await c.req.json().catch(() => null);
  setCookie(c, SESSION_COOKIE_NAME, signedToken, ...);
  return { ok: true, user: {...} };
}
```

Service layer langsung menerima `c` (Hono Context) dan memanggil `c.req.json()` dan `setCookie(c, ...)`. Akibatnya:

1. Tidak bisa di-test tanpa mock Hono context
2. Tidak bisa dipakai di script CLI (scripts/bootstrap-superadmin.js harus bypass fungsi ini)
3. Kalau Hono diganti framework lain, seluruh auth/service.js harus ditulis ulang

Solusi:

```
// Pisahkan concern HTTP dari business logic
export async function loginWithEmailPassword(env, email, password) {
  // hanya terima data plain, kembalikan data plain
  return { ok: true, signedToken, user, memberships };
}
```

```
// Di router/handler:
const result = await loginWithEmailPassword(c.env, email, password);
if (result.ok) {
  setCookie(c, SESSION_COOKIE_NAME, result.signedToken, ...);
}
```

⚠ SERIUS #7: `forms/repository.js` Import dari `bootstrap.js`

File: `src/modules/forms/repository.js`

```
import { ensureDraftVersion, ensurePublishedVersion } from
'../../lib/db/bootstrap.js';
```

Repository layer seharusnya hanya import dari `lib/db/sql.js`. Tapi `forms/repository.js` import dari `bootstrap.js` (600+ baris). Ini berarti:

1. Setiap kali `forms/repository.js` dipakai, seluruh `bootstrap.js` di-load ke memory
2. Bootstrap logic tersebar di dua tempat yang berbeda
3. Coupling yang tidak perlu antara repository dan infrastruktur bootstrap

Solusi: Pindahkan `ensureDraftVersion` dan `ensurePublishedVersion` ke `forms/repository.js` langsung, atau buat file terpisah `src/lib/db/form-versions.js`.

⚠ SERIUS #8: `schools/service.js` — Circular-Like Dependency

File: `src/modules/schools/service.js`

```
import { ensureTenantQuestionnaireInitialized } from
'../questionnaires/service.js';
import { syncTenantSuperadmins } from '../tenants/repository.js';
```

`schools/service.js` import dari `questionnaires/service.js`. Dan `questionnaires/service.js` import dari `submissions/validation.js`. Dan `submissions/service.js` dipakai oleh `analytics/service.js`.

Dependency chain yang dalam dan tidak perlu. Kalau `questionnaires/service.js` berubah signature-nya, `schools/service.js` bisa terdampak secara tidak terduga.

Solusi: Buat event-driven initialization — setelah `createNewSchool` berhasil, emit event `school.created`, lalu questionnaire module listen dan inisialisasi sendiri. Atau lebih sederhana: pindahkan `ensureTenantQuestionnaireInitialized` ke caller (worker route handler), bukan di service.

⚠ SERIUS #9: `submissions/repository.js` — Schema Hardcoded 12 Pertanyaan

File: `src/modules/submissions/repository.js`

```
INSERT INTO responses (
  school_id, form_version_id,
  nama_guru, lama_mengajar, mata_pelajaran,
  q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12,  // HARDCODED!
  extra_answers, payload
)
```

Tabel `responses` punya kolom `q1`-`q12` hardcoded. Kalau questionnaire punya 15 pertanyaan, q13-q15 hanya masuk `extra_answers`. Kalau questionnaire punya 8 pertanyaan, q9-q12 akan NULL atau 0.

Ini bertentangan dengan sistem questionnaire dinamis yang baru (tabel `responses_v2` yang pakai JSONB `answers`). Tabel lama ini adalah legacy untuk SMAN 6 Kota Kupang saja — tapi masih dipakai oleh `submissions/service.js` untuk dual-write.

Solusi: Setelah migrasi data selesai, stop dual-write ke tabel `responses` dan hanya tulis ke `responses_v2`.

⚠ SERIUS #10: `submissions/repository.js` — Business Logic di SQL

File: `src/modules/submissions/repository.js`

```
// getSummaryStats — business logic hardcoded di SQL!
ROUND(AVG((q7 + q8 + q9 + q11) / 4.0)::numeric, 2) AS avg_ai_adoption,
ROUND(
  (COUNT(*) FILTER (WHERE q10 IN ('Sangat Berminat', 'Berminat'))
   * 100.0 / NULLIF(COUNT(*), 0))::numeric, 2
) AS interested_pct
```

"AI adoption = rata-rata q7, q8, q9, q11" adalah business rule yang dikubur dalam SQL di repository layer. Kalau definisi "AI adoption" berubah (misal: tambah q13), harus cari dan ubah SQL ini. Tidak ada dokumentasi kenapa q7, q8, q9, q11 yang dipilih.

Solusi: Pindahkan business rule ke service layer. Repository hanya return raw averages per pertanyaan, service yang menghitung composite metric.

⚠ SERIUS #11: `auth/repository.js` — Superadmin Disimpan dengan `school_id = NULL`

File: `src/modules/auth/repository.js`

```
export async function grantSuperadminRole(env, userId) {
  await sql`
    INSERT INTO school_memberships (user_id, school_id, role)
    VALUES (${userId}, NULL, 'superadmin')  // school_id = NULL untuk superadmin!
  `;
}
```

Superadmin disimpan di tabel `school_memberships` dengan `school_id = NULL`. Ini adalah design yang aneh yang menyebabkan:

1. Logic authorization harus selalu cek `hasSuperadmin` dulu sebelum cek `school_id`
2. Tabel `school_memberships` punya data yang semantically berbeda (superadmin tidak punya school)
3. Confusing bagi developer baru

Design yang lebih baik: superadmin seharusnya ada di `tenant_memberships` dengan `tenant_id = NULL` dan `role = 'superadmin'`, atau ada tabel `user_roles` tersendiri.

⚠ SERIUS #12: `ai-prompts/service.js` — 6 Query Paralel per Request

File: `src/modules/ai-prompts/service.js`

```
const [globalDraft, globalPublished, tenantDraft, tenantPublished,
       questionnaireDraft, questionnairePublished] = await Promise.all([
  getPromptByStatus(env, { mode, scope: 'global', status: 'draft' }),
  getPromptByStatus(env, { mode, scope: 'global', status: 'published' }),
  getPromptByStatus(env, { mode, scope: 'tenant', tenantId, status: 'draft' }),
  getPromptByStatus(env, { mode, scope: 'tenant', tenantId, status: 'published'
}),
  getPromptByStatus(env, { mode, scope: 'questionnaire', ..., status: 'draft' }),
  getPromptByStatus(env, { mode, scope: 'questionnaire', ..., status: 'published'
}),
]);
```

6 query paralel setiap kali admin membuka AI prompt management. Di Neon serverless (HTTP-based), setiap query adalah HTTP round trip tersendiri. Ini bisa dikurangi menjadi 1 query dengan JSONB aggregation atau 2 query dengan filter yang lebih cerdas.

Solusi:

```
-- Satu query untuk ambil semua prompts yang relevan
SELECT mode, scope, tenant_id, questionnaire_id, status, template, published_at
FROM ai_prompt_versions_v2
WHERE mode = $1
  AND (
    (scope = 'global')
    OR (scope = 'tenant' AND tenant_id = $2)
    OR (scope = 'questionnaire' AND tenant_id = $2 AND questionnaire_id = $3)
  )
  AND status IN ('draft', 'published')
ORDER BY scope DESC, created_at DESC;
```

⚠ SERIUS #13: `bootstrap.js` — Data Migration Tanpa Limit

File: `src/lib/db/bootstrap.js`

```
async function syncResponsesV2FromResponses(sql) {
  await sql`
    INSERT INTO responses_v2 (...)
    SELECT ... FROM responses r
    WHERE NOT EXISTS (SELECT 1 FROM responses_v2 rv2 WHERE rv2.legacy_response_id
= r.id);
    // TIDAK ADA LIMIT — bisa INSERT jutaan row dalam satu transaksi
  `;
}
```

Kalau ada 100.000 response di tabel `responses`, query ini akan INSERT semua sekaligus. Neon serverless punya timeout default 30 detik. Partial migration yang terpotong di tengah akan menyebabkan inconsistency data yang sangat sulit di-debug.

Solusi: Tambahkan batched migration:

```
async function syncResponsesV2FromResponses(sql, batchSize = 1000) {
  let offset = 0;
  while (true) {
    const result = await sql`
      INSERT INTO responses_v2 (...) SELECT ... FROM responses r
      WHERE NOT EXISTS (...) LIMIT ${batchSize};
    `;
    if (result.count < batchSize) break; // selesai
    offset += batchSize;
  }
}
```

⚠ SERIUS #14: `tenants/service.js` — Langsung Tulis ke Tabel `schools`

File: `src/modules/tenants/service.js`

```
// Tenant service langsung tulis ke tabel schools — bypass schools/service!
const sql = getSqlClient(env);
await sql`
  INSERT INTO schools (id, slug, name, is_active)
  VALUES (${created.id}, ${created.slug}, ${created.name}, ${created.is_active})
  ON CONFLICT (id) DO UPDATE ...
`;
```

Module `tenants` seharusnya tidak tahu tentang tabel `schools`. Kalau ada validasi atau business logic di `schools/service.js`, ini akan di-bypass. Kalau schema tabel `schools` berubah, harus update dua tempat.

Solusi:

```
// tenants/service.js
import { upsertSchoolFromTenant } from '../schools/service.js';
// Panggil schools/service, bukan langsung ke DB
await upsertSchoolFromTenant(env, created);
```

---

4. Masalah Arsitektur — Technical Debt Jangka Menengah

🟡 ARSITEKTUR #15: Duplikasi Module Legacy vs Baru

Module Lama (LEGACY)	Module Baru	Status
`src/modules/submission/`	`src/modules/submissions/`	Legacy harus dihapus
`src/modules/form/`	`src/modules/forms/`	Legacy harus dihapus
`src/lib/db/client.js`	`src/lib/db/sql.js`	client.js harus dihapus
`worker.js` (root)	`src/worker.js`	Root worker harus dihapus
`src/server.js`	`src/worker.js`	server.js harus dihapus
`src/app/routes.js`	`src/worker.js` (routing)	routes.js harus dihapus

Semua file "lama" di atas **tidak dipakai di production** tapi masih ada di codebase. Ini membuat navigasi codebase sangat membingungkan.

🟡 ARSITEKTUR #16: `questionnaires/service.js` adalah God Module (700+ Baris)

Satu file melakukan 8 hal berbeda:

1. CRUD questionnaire & version management
2. Response submission
3. Analytics summary & distribution
4. Analytics trend
5. CSV export
6. AI source data preparation
7. Segmentation analysis (criteria, score band, respondent dimensions)
8. Filter/pagination normalization utilities

Bagian yang harus dipecah:

```
SEKARANG:
src/modules/questionnaires/service.js (700+ baris)

SEHARUSNYA:
src/modules/questionnaires/service.js        ← Hanya CRUD & version management
(~150 baris)
src/modules/analytics/distribution.js        ← computeDistribution,
```

```
buildSegmentSummary (~300 baris)
src/modules/analytics/service.js              ← Summary, trend, aggregation (bukan
wrapper kosong)
src/lib/utils/filters.js                      ← normalizeFromFilter, normalizeDays,
dll
```

🟤 ARSITEKTUR #17: `analytics/service.js` adalah Wrapper Kosong

```
// SELURUH ISI analytics/service.js — hanya 3 fungsi wrapper!
export async function getAnalyticsSummary(env, schoolId) {
  return getSummary(env, schoolId); // delegate ke submissions
}
export async function getAnalyticsDistribution(env, schoolId) {
  return getDistribution(env, schoolId);
}
export async function getAnalyticsTrend(env, schoolId, days) {
  return getTrendAnalytics(env, schoolId, days);
}
```

Module analytics yang nyata (distribution computation, segmentation analysis) ada di `questionnaires/service.js`. Ini terbalik — analytics seharusnya ada di `analytics/`, bukan di `questionnaires/`.

🟤 ARSITEKTUR #18: Duplikasi Fungsi Utility di 3 File

Fungsi-fungsi ini didefinisikan ulang di beberapa tempat:

Fungsi	Lokasi
`escapeCsvValue`	`submission/service.js`, `submissions/service.js`, `questionnaires/service.js`
`normalizeFromFilter`	`questionnaires/service.js`, `submissions/service.js`, `ai/service.js`
`normalizeToFilter`	`questionnaires/service.js`, `submissions/service.js`, `ai/service.js`
`normalizeDays`	`questionnaires/service.js`, `submissions/service.js`
`normalizePage`	`questionnaires/service.js`, `submissions/service.js`
`slugify`	`schools/service.js`, `tenants/service.js`

Semuanya seharusnya ada di `src/lib/utils/`.

🟤 ARSITEKTUR #19: `bootstrap.js` Terlalu Besar dan Terlalu Banyak Tanggung Jawab

File `bootstrap.js` (600+ baris) melakukan:

1. Schema creation (CREATE TABLE)
2. Schema migration (ALTER TABLE)
3. Index creation
4. Data normalization (dedup superadmin, normalize statuses)
5. Data migration (migrate legacy responses, sync v2 tables)
6. Seeding (default AI prompt templates)
7. Schema verification (verifySchemaReady)

Ini seharusnya dipisah menjadi:

- `src/lib/db/schema.js` — table definitions
- `scripts/migrate-*.sql` — versioned migrations
- `src/lib/db/seed.js` — default data seeding
- `src/lib/db/bootstrap.js` — hanya runtime verification

🟡 ARSITEKTUR #20: `ai/service.js` — 8+ Dependency (Orchestrator Salah Tempat)

`ai/service.js` import dari 8+ module berbeda:

- `ai/repository`
- `forms/service`
- `analytics/service`
- `submissions/service`
- `lib/db/bootstrap`
- `schools/service`
- `ai-prompts/service`
- `questionnaires/service`
- `questionnaires/repository` (bypass service layer!)

Module dengan 8+ dependency adalah tanda bahwa dia adalah **orchestrator yang seharusnya ada di router layer**, bukan di service layer. Service seharusnya punya dependency maksimal 3-4 module.

🟡 ARSITEKTUR #21: `public/` Folder Tidak Terorganisir

```
public/
├── dashboard.html          ← Dashboard LAMA (legacy, auth via Basic Auth HTTP)
├── dashboard.js            ← Dashboard LAMA
├── index.html              ← Form submission LAMA (single school)
├── script.js               ← Script LAMA
├── admin/                  ← Admin panel BARU (multi-tenant)
└── forms/                  ← Form submission BARU (multi-tenant)
```

Dua UI untuk hal yang sama. Tidak ada dokumentasi mana yang aktif. `dashboard.html` lama masih bisa diakses via URL langsung.

---

5. Status Modular Monolith — Verdict Final

Apakah App Ini Sudah Modular Monolith?

> **VERDICT: 4/10 — Strukturnya ada, implementasinya bocor di banyak tempat.**

Yang SUDAH modular:

- `auth/` — batas domain jelas (authentication & session)
- `ai-prompts/` — single responsibility (template management)
- `questionnaires/repository.js` — clean SQL, hanya import dari `sql.js`
- `submissions/validation.js` — pure function, tidak ada side effect

Yang BELUM modular:

- `questionnaires/service.js` — God Module, 8 tanggung jawab berbeda
- `analytics/service.js` — empty wrapper, tidak ada value
- `ai/service.js` — 8+ dependency, harus jadi orchestrator di router layer
- `forms/repository.js` — import dari bootstrap (coupling salah)
- `tenants/service.js` — tulis langsung ke tabel `schools`
- `schools/service.js` — import dari `auth/repository` (user management bukan tanggung jawab schools)

Prinsip Modular Monolith yang dilanggar:

1. ❌ Module punya batas jelas → `questionnaires` melakukan analytics
2. ❌ Tidak saling tahu internal → `tenants` tahu skema tabel `schools`
3. ❌ Repository hanya import dari DB client → `forms/repository` import dari `bootstrap`
4. ❌ Service tidak coupling ke framework → `auth/service` coupling ke Hono
5. ☑ Tidak ada circular dependency (hampir — ada circular-like yang dangerous)

---

6. Koreksi dari Review Sebelumnya

Temuan di Review 1 & 2	Status Sebenarnya Setelah Review 3
"Bug `session.is_active` — semua user akan logout"	**KOREKSI:** `session.is_active` sebenarnya adalah `users.is_active` dari JOIN query. Bukan bug fatal, tapi naming sangat menyesatkan dan berpotensi menyebabkan bug di masa depan.
"Advisory lock bisa deadlock di Neon"	**PERLU VERIFIKASI:** Neon mendukung `pg_advisory_xact_lock` karena mereka kompatibel dengan PostgreSQL wire protocol. Tapi di HTTP pooler mode, perlu dipastikan connection tidak di-recycle sebelum lock dilepas.
"Bootstrap berbahaya di production"	**KONFIRMASI + KOREKSI:** Di production `DB_BOOTSTRAP_MODE=check` — aman. Di local, `full` mode bisa race condition kalau ada concurrent cold starts. Ini low risk untuk single developer.

---

7. Peta Dependency Antar Module

Dependency yang SALAH (harus diperbaiki):

```
forms/repository ──────────────────────→ lib/db/bootstrap   ← SALAH (repo→infra)
tenants/service ──────────────────────→ auth/repository    ← SALAH (cross-domain)
tenants/service ──────────────────────→ schools table (SQL) ← SALAH (bypass
module)
schools/service ──────────────────────→ auth/repository    ← SALAH (cross-domain)
schools/service ──────────────────────→ questionnaires/service ← membuat coupling
dalam
ai/service ──────────────────────→ questionnaires/repository ← bypass
service layer
ai/service ──────────────────────→ lib/db/bootstrap   ← SALAH
(service→infra)
auth/service ──────────────────────→ hono/cookie        ← SALAH
(service→framework)
```

Dependency yang BENAR (tidak perlu diubah):

```
questionnaires/repository → lib/db/sql        ← BENAR
submissions/validation    → zod               ← BENAR
ai-prompts/service        → ai/modes          ← BENAR (shared constant)
analytics/service         → submissions/service ← BENAR (consumer pattern)
```

---

8. Action Plan Bertahap

FASE 1 — Quick Wins Security (Estimasi: 1-2 hari)

Tidak ada risiko breaking change.

#	Aksi	File	Effort
1	Naikkan PBKDF2 ke 600.000 iterasi	`src/lib/security/hash.js`	5 menit
2	Rename `session.is_active` → `session.user_is_active`	`auth/repository.js`, `auth/service.js`	15 menit
3	Tambah limit 10.000 di `listQuestionnaireResponsesForAggregation`	`questionnaires/service.js`	5 menit
4	Tambah Cloudflare WAF rate limiting rule untuk login endpoint	Cloudflare Dashboard	10 menit
5	Ubah `monitorAdminOrigin` menjadi actual CSRF check	`request-guards.js`	30 menit

FASE 2 — Bersihkan Legacy (Estimasi: 2-4 jam)

Risiko: Rendah — file-file ini tidak dipakai di production.

```
HAPUS (atau pindahkan ke legacy/):
├── src/server.js
├── src/app/routes.js
├── src/modules/form/          (seluruh folder)
├── src/modules/submission/    (seluruh folder)
└── src/lib/db/client.js

TANDAI DEPRECATED (jangan hapus dulu, perlu verifikasi):
└── worker.js (root) — tambahkan komentar // DEPRECATED: pakai src/worker.js
```

Sebelum hapus, pastikan dengan:

```
# Cek apakah ada yang masih import dari file-file ini
grep -r "from.*src/server" src/
grep -r "from.*modules/form'" src/
grep -r "from.*modules/submission'" src/
grep -r "from.*db/client" src/
```

FASE 3 — Shared Utilities (Estimasi: 1-2 hari)

Risiko: Rendah setelah test.

```
BUAT:
src/lib/utils/
├── csv.js          ← pindah dari questionnaires/service.js &
submissions/service.js
│   export: escapeCsvValue, formatAnswerForCsv
├── filters.js      ← pindah dari questionnaires/service.js &
submissions/service.js
│   export: normalizeFromFilter, normalizeToFilter, normalizeDays, normalizePage,
normalizePageSize, normalizeSearch
└── slug.js         ← pindah dari schools/service.js & tenants/service.js
    export: slugify
```

FASE 4 — Perbaiki Module Boundaries (Estimasi: 1-2 minggu)

Risiko: Sedang — perlu test setelah setiap perubahan.

4a. Pisahkan `auth/service.js` dari Hono

```
// SEBELUM:
export async function loginWithEmailPassword(c) { ... setCookie(c, ...) }

// SESUDAH:
export async function loginWithEmailPassword(env, email, password) {
  // return { ok, signedToken, user, memberships }
}
// Di router handler: setCookie(c, result.signedToken, ...)
```

4b. Pindahkan `ensureDraftVersion` keluar dari `bootstrap.js`

Buat `src/lib/db/form-versions.js` yang berisi fungsi-fungsi ini, lalu:

- `forms/repository.js` import dari `form-versions.js`
- `bootstrap.js` juga import dari `form-versions.js`

4c. Buat module `users/`

```
src/modules/users/
├── service.js    ← createUser, findUser (pindah dari auth/repository)
└── repository.js ← DB operations untuk tabel users
```

Update `tenants/service.js` dan `schools/service.js` untuk import dari `users/service` bukan `auth/repository`.

4d. Fix `tenants/service.js` — jangan tulis langsung ke `schools`

```
// HAPUS direct SQL ke schools dari tenants/service.js
// GANTI dengan memanggil schools/service:
import { upsertSchoolRecord } from '../schools/service.js';
await upsertSchoolRecord(env, { id: created.id, slug, name, isActive: true });
```

FASE 5 — Refactor God Module (Estimasi: 2-3 minggu)

Risiko: Tinggi — perlu test coverage yang baik sebelum mulai.

Pecah `questionnaires/service.js` menjadi:

```
src/modules/questionnaires/
├── service.js              ← Hanya CRUD + version management (~200 baris)
└── response-service.js     ← Submission logic (~100 baris)

src/modules/analytics/
├── service.js              ← Orchestrator (hapus wrapper kosong, isi dengan logic
```

```
nyata)
├── distribution.js         ← computeDistribution, buildSegmentSummary (~350
baris)
└── trend.js                ← getTrend, aggregation

src/modules/ai/
├── service.js              ← Orchestrator saja (~100 baris)
├── gemini-client.js        ← callGemini, HTTP client
└── prompt-builder.js       ← buildPrompt*, prompt construction
```

FASE 6 — Pindahkan Analytics ke Database (Estimasi: 3-4 minggu)

Risiko: Tinggi — perlu validasi hasil identik dengan implementasi JS saat ini.

```
-- Contoh: distribution dihitung di DB, bukan di JS
SELECT
  kv.key AS field_name,
  kv.value AS answer_value,
  COUNT(*) AS total
FROM responses_v2,
  jsonb_each_text(answers) AS kv(key, value)
WHERE questionnaire_id = $1
  AND (created_at >= $2 OR $2 IS NULL)
GROUP BY kv.key, kv.value
ORDER BY kv.key, total DESC;
```

---

9. Target Arsitektur Ideal

```
src/
├── worker.js                   ← Router + middleware setup ONLY
│
├── lib/
│   ├── db/
│   │   ├── sql.js              ← Satu-satunya DB client
│   │   ├── bootstrap.js        ← Hanya schema check (verifySchemaReady)
│   │   └── form-versions.js    ← ensureDraftVersion, ensurePublishedVersion
│   ├── http/
│   │   ├── request-guards.js   ← Real CSRF protection (bukan hanya logging)
│   │   └── session-cookie.js
│   ├── security/
│   │   ├── hash.js             ← PBKDF2 600k iterations
│   │   └── signature.js
│   └── utils/
│       ├── csv.js              ← Shared CSV utilities
│       ├── filters.js          ← Shared filter normalization
│       ├── pagination.js       ← Shared pagination utilities
│       └── slug.js             ← Shared slug utilities
```

```
│
└── modules/
    ├── auth/                    ← Authentication & session ONLY (no HTTP
coupling)
    ├── users/                   ← User management BARU (pindah dari
auth/repository)
    ├── tenants/                 ← Tenant CRUD (tidak tulis ke schools langsung)
    ├── schools/                 ← School CRUD (tidak import dari auth/repository)
    ├── questionnaires/          ← Questionnaire CRUD & version ONLY
    ├── responses/               ← Response submission & retrieval (RENAME dari
submissions)
    ├── analytics/               ← Analytics computation (DIISI konten nyata dari
questionnaires)
    │   ├── service.js           ← Orchestrator
    │   ├── distribution.js      ← computeDistribution, segmentation
    │   └── trend.js             ← trend analytics
    ├── ai/                      ← AI orchestration
    │   ├── service.js           ← Coordinator (max 4 dependency)
    │   ├── gemini-client.js     ← HTTP client ke Gemini BARU
    │   └── prompt-builder.js    ← Prompt construction BARU
    ├── ai-prompts/              ← AI prompt template management
    └── forms/                   ← Form version management (legacy compat)
```

5 Prinsip yang Harus Dijaga Setelah Refactor:

1. **Repository hanya import dari** `lib/db/sql.js` — tidak boleh import dari module lain atau dari bootstrap
2. **Service tidak import dari** `lib/db/bootstrap.js` — bootstrap hanya dipakai di startup
3. **Service tidak coupling ke HTTP framework** — tidak ada `setCookie`, `c.req.json()` di service layer
4. **Setiap module hanya menulis ke tabelnya sendiri** — tenants tidak tulis ke schools, schools tidak tulis ke tenant_memberships
5. **Shared code ada di** `lib/utils/` — tidak boleh diduplikasi di setiap module

---

10. Nilai Akhir per Aspek

Aspek	Nilai	Catatan
Fungsionalitas	8/10	Fitur lengkap, sudah production
Security	4/10	PBKDF2 lemah, no rate limit, no CSRF
Architecture	5/10	Struktur ada, coupling bocor
Modular Monolith	4/10	Belum tercapai — God Module, coupling salah
Code Quality	5/10	Duplikasi, legacy code, business logic di SQL
Reliability	5/10	In-memory analytics berbahaya, migration tanpa limit
Maintainability	4/10	Sulit dipahami tanpa panduan, legacy code membingungkan
Testability	3/10	Service coupling ke framework, sulit di-unit test

Aspek	Nilai	Catatan
Developer Experience	4/10	14 modules, dua set legacy, tidak ada test

Nilai Keseluruhan: 4.7/10

Kesimpulan Final

App ini **berhasil dalam hal fungsionalitas** — sudah production, sudah melayani user nyata, fiturnya impressive (AI analysis, PDF export, multi-tenant, questionnaire builder). Developer yang membangun ini jelas kompeten.

Tapi **arsitekturnya sedang dalam kondisi transitional** yang belum selesai. 70% sudah benar, 30% masih merupakan beban legacy yang jika tidak segera dibersihkan akan semakin sulit diubah seiring bertambahnya data dan user.

3 hal yang paling mendesak:

1. Naikkan PBKDF2 ke 600.000 iterasi — 5 menit, efeknya besar untuk keamanan
2. Tambah hard limit pada analytics in-memory — 5 menit, mencegah crash production
3. Hapus legacy modules dan file — 2-4 jam, mengurangi confusion developer

Jika ketiga hal itu sudah dilakukan, lanjutkan dengan Fase 4 (Module Boundaries) untuk mencapai modular monolith yang sesungguhnya.

Laporan ini menggabungkan temuan dari 3 sesi review mendalam terhadap seluruh source code termasuk file yang paling jarang dilihat: bootstrap.js (600+ baris), questionnaires/service.js (700+ baris), auth/repository.js, auth/service.js, hash.js, forms/repository.js, dan semua module lainnya.

Beberapa temuan di review pertama dan kedua dikoreksi setelah review ketiga yang lebih mendalam — khususnya tentang `session.is_active` yang ternyata adalah `users.is_active` dari JOIN query, bukan kolom yang tidak ada.
