Laporan Analisis Kode — AITI Forms App

Ditulis oleh: Claude Sonnet (Anthropic)
Tanggal: 18 Februari 2026
Repo: `formsman6kotakupang` — AITI Global Nexus
Tipe Review: Brutally Honest Code Review & Architecture Audit

---

Daftar Isi

---

1. Gambaran Arsitektur Sebenarnya

App ini sebenarnya adalah **dua sistem berbeda yang hidup berdampingan** dalam satu repo. Ini bukan desain yang disengaja — ini adalah hasil dari proses migrasi yang belum selesai.

Layer	Stack	Status
Legacy Backend	Express.js (`src/server.js`)	Rollback-only, sudah deprecated
Production Backend	Cloudflare Worker + Hono (`src/worker.js`)	Aktif di production
Proxy Lama	`worker.js` (root)	Zombie file, tidak relevan lagi
Database	Neon PostgreSQL via `@neondatabase/serverless`	Aktif
AI Engine	Google Gemini API	Aktif
Frontend	Vanilla JS + Chart.js + jsPDF	Aktif

Alur Kerja Production (Cloudflare Worker)

```
User Browser
    │
    ▼
Cloudflare Worker (src/worker.js — Hono)
    │
    ├── /forms/*          → Serve HTML dari public/ via ASSETS binding
```

```
      ├── /forms-static/*   → Serve static assets
      ├── /forms/:slug/api/ → API endpoints (DB queries ke Neon)
      └── /forms/admin/api/ → Admin API (auth required)
                │
                ▼
          Neon PostgreSQL (cloud)
                │
                ▼
          Google Gemini API (untuk AI analysis)
```

Alur Kerja Legacy (Express — TIDAK DIPAKAI DI PRODUCTION)

```
User Browser → Express.js (src/server.js) → Neon PostgreSQL
```

---

2. Masalah Kritis — Harus Diperbaiki Segera

🔴 Masalah #1: Duplikasi Module yang Sangat Membingungkan

Ini adalah masalah terbesar dalam codebase ini. Ada **dua set module yang hampir identik**:

```
src/modules/submission/   ← LEGACY (dipakai Express server)
src/modules/submissions/  ← BARU (dipakai Hono worker)

src/modules/form/         ← LEGACY
src/modules/forms/        ← BARU
```

Dampak konkret:

- `src/modules/submission/repository.js` masih menulis ke tabel `form_responses` (tabel lama yang dibuat manual via `ensureSubmissionTable()`)
- `src/modules/submissions/repository.js` menulis ke tabel `responses` (tabel baru yang proper dengan `school_id`, `form_version_id`)
- `src/app/routes.js` masih import dari `submission/` (singular) — ini path Express legacy yang sudah tidak dipakai di production tapi masih ada di codebase
- Developer baru yang masuk ke project ini **tidak akan tahu mana yang dipakai production** tanpa membaca seluruh codebase

Solusi: Hapus `src/modules/submission/` dan `src/modules/form/` (singular). Konsolidasi ke versi plural yang sudah benar.

🔴 Masalah #2: `worker.js` di Root — Zombie File

File `worker.js` di root adalah **Cloudflare Worker lama** yang berfungsi sebagai proxy dengan Basic Auth HTTP. Ini sudah **tidak dipakai** karena production sekarang pakai `src/worker.js` (Hono dengan auth

berbasis session cookie).

Tapi file ini masih ada, tidak ada komentar yang jelas bahwa ini sudah deprecated, dan bisa membingungkan siapapun yang baru bergabung ke project.

Solusi: Hapus atau pindahkan ke folder `legacy/` dengan komentar jelas.

🟤 Masalah #3: Dua DB Client dengan Signature Berbeda

```
// src/lib/db/client.js — dipakai module legacy
export function getSqlClient() {
  return neon(process.env.DATABASE_URL); // hanya baca process.env
}

// src/lib/db/sql.js — dipakai module baru
export function getSqlClient(env) {
  return neon(env?.DATABASE_URL || process.env.DATABASE_URL); // bisa terima env
dari Worker
}
```

Dua file, fungsi sama, nama sama, tapi signature berbeda. Kalau ada yang salah import `client.js` di dalam Hono Worker, query ke DB akan **gagal di production** karena Cloudflare Workers tidak punya `process.env`.

Solusi: Hapus `src/lib/db/client.js`. Pakai hanya `src/lib/db/sql.js`.

🟤 Masalah #4: Bootstrap yang Terlalu Berat dan Berbahaya

`src/lib/db/bootstrap.js` — fungsi `initializeSchema()` menjalankan **puluhan ALTER TABLE, CREATE TABLE, CREATE INDEX, dan data migration** setiap kali app cold start di mode `full`.

Masalah konkret:

- Di production, `DB_BOOTSTRAP_MODE=check` — aman
- Di local/staging, `DB_BOOTSTRAP_MODE=full` — setiap request pertama menjalankan ~50+ SQL statements
- Cache `schemaPromisesByMode` hilang setiap Worker restart (Cloudflare Workers stateless)
- Artinya setiap cold start di local, ada ~50 SQL queries sebelum request pertama selesai
- Fungsi `migrateLegacyResponses`, `syncResponsesV2FromResponses`, dll adalah **data migration yang dijalankan di runtime**, bukan di migration script — ini pola yang sangat berbahaya

Solusi: Pisahkan bootstrap dari runtime. Migration harus jalan via script (`pnpm migrate:multi`), bukan saat request pertama.

🟤 Masalah #5: PBKDF2 dengan 10.000 Iterasi — Terlalu Lemah

```
// src/lib/security/hash.js
const DEFAULT_PBKDF2_ITERATIONS = 10000;
```

NIST merekomendasikan minimum **600.000 iterasi** untuk PBKDF2-SHA256 (standar 2023). 10.000 iterasi sudah sangat ketinggalan zaman dan rentan terhadap brute force dengan hardware modern.

Ini bukan masalah kecil — ini adalah **kelemahan keamanan nyata** untuk password admin yang melindungi seluruh data feedback sekolah.

Solusi: Naikkan ke 600.000 iterasi. Re-hash semua password yang ada saat login berikutnya.

🟠 Masalah #6: Rate Limiting Tidak Ada

Endpoint `/forms/admin/api/login` tidak ada rate limiting sama sekali. Siapapun bisa melakukan brute force password admin tanpa hambatan apapun.

Di Cloudflare Worker, ini bisa diatasi dengan Cloudflare Rate Limiting rules, tapi tidak ada dokumentasi atau konfigurasi untuk itu di repo ini.

Solusi: Tambahkan Cloudflare WAF Rate Limiting rules untuk endpoint login. Minimal: 5 percobaan per IP per menit.

🟠 Masalah #7: AI Endpoint Tanpa Rate Limiting / Cost Control

```
app.post('/forms/:tenantSlug/admin/api/questionnaires/:questionnaireSlug/ai/analyze', ...)
```

Setiap klik tombol "Analisa" memanggil Gemini API langsung. Tidak ada:

- Cooldown per user/tenant
- Limit berapa kali per hari
- Estimasi cost sebelum eksekusi
- Logika "jangan re-analyze kalau data tidak berubah sejak analisa terakhir"

Kalau ada 100 tenant masing-masing klik "Analisa" 10x sehari, biaya Gemini bisa meledak tanpa peringatan.

Solusi: Tambahkan cooldown minimum (misal: 1 jam per tenant per mode), dan cek apakah ada data baru sejak analisa terakhir sebelum memanggil Gemini.

---

3. Masalah Serius — Perlu Perhatian

⚠ Masalah #8: Bug `session.is_active` — Kolom Tidak Ada di Schema

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

```
const expired = new Date(session.expires_at).getTime() <= Date.now();
if (session.revoked_at || expired || !session.is_active) {
  await revokeSessionByTokenHash(c.env, tokenHash);
  return null;
}
```

Tapi di schema tabel `sessions`:

```
CREATE TABLE IF NOT EXISTS sessions (
  id UUID PRIMARY KEY,
  user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  token_hash TEXT NOT NULL UNIQUE,
  expires_at TIMESTAMPTZ NOT NULL,
  revoked_at TIMESTAMPTZ,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
  -- TIDAK ADA KOLOM is_active!
);
```

Kolom `is_active` **tidak ada di tabel sessions**. Query akan return `undefined` untuk `is_active`, dan `!undefined === true`, sehingga **setiap session akan dianggap tidak aktif** dan selalu di-revoke saat dicek.

Ini adalah **bug yang bisa menyebabkan semua user tidak bisa tetap login** (setiap request akan dianggap unauthorized). Perlu dicek segera apakah kolom ini sudah ditambahkan via migration yang tidak tercatat di `schema.sql`.

Solusi: Tambahkan kolom `is_active BOOLEAN NOT NULL DEFAULT TRUE` ke tabel `sessions`, atau hapus pengecekan `!session.is_active` dari kode.

⚠ Masalah #9: `public/dashboard.html` — Dashboard Lama Tanpa Auth Proper

File `public/dashboard.html` dan `public/dashboard.js` adalah **dashboard lama** yang:

- Tidak ada autentikasi di sisi client
- Langsung hit `/api/analytics/summary`, `/api/ai/analyze` dll
- Proteksi hanya dari Cloudflare Worker proxy lama (`worker.js` root) via Basic Auth HTTP
- Kalau diakses langsung ke backend tanpa melalui proxy, **semua data terbuka**

Di production sekarang (Hono worker), endpoint analytics sudah di-protect dengan `requireAuth()`. Tapi dashboard lama ini masih ada di `public/` dan masih bisa diakses via path `/dashboard.html`.

Solusi: Hapus atau pindahkan `public/dashboard.html` dan `public/dashboard.js` ke folder `legacy/`.

⚠ Masalah #10: `ensureSubmissionTable()` — Anti-Pattern

```
// src/modules/submission/service.js
let tableEnsured = false;
```

```
async function ensureTableOnce() {
  if (tableEnsured) return;
  await ensureSubmissionTable();
  tableEnsured = true;
}
```

Pattern "lazy table creation" ini:

- Membuat tabel `form_responses` yang **berbeda** dari tabel `responses` di schema utama
- Artinya ada **dua tabel responses** yang bisa ada di DB: `form_responses` (legacy) dan `responses` (baru)
- Tidak thread-safe dalam konteks concurrent requests
- Menyembunyikan dependency DB dari startup, membuat debugging lebih sulit

Solusi: Hapus seluruh module `src/modules/submission/` (legacy). Tabel `form_responses` sudah dimigrasikan ke `responses` via bootstrap.

⚠ Masalah #11: Tidak Ada CSRF Protection

`monitorAdminOrigin` di `src/lib/http/request-guards.js` hanya **memonitor dan log** origin yang berbeda, tidak **memblokir** cross-origin requests:

```
// Ini hanya logging, BUKAN protection
console.warn(`[ORIGIN_MONITOR] requestId=... origin=${origin}
host=${requestHost}`);
await next(); // tetap lanjut!
```

Admin endpoints bisa dipanggil dari domain manapun selama ada session cookie yang valid. Ini bukan CSRF protection.

Solusi: Implementasikan CSRF token atau setidaknya blokir requests dengan Origin header yang tidak cocok dengan host.

⚠ Masalah #12: `public/script.js` — Dead Code

```
// public/script.js
const viewDashboardBtn = document.getElementById('view-dashboard-btn');
```

Tapi di `public/index.html` tidak ada element dengan id `view-dashboard-btn`. Fungsi `setDashboardLinkEnabled()` dipanggil berkali-kali tapi tidak pernah berfungsi karena element tidak ada.

Solusi: Hapus referensi `viewDashboardBtn` dan fungsi `setDashboardLinkEnabled()` dari `script.js`.

---

4. Masalah Medium — Technical Debt

🟤 Masalah #13: Dual-Write Pattern yang Rapuh

Setiap submit response, ada dual-write ke `responses` dan `responses_v2`. Kalau `responses_v2` gagal, hanya di-log sebagai warning:

```
console.error(`[DUAL_WRITE_WARNING] responses_v2 gagal sinkron...`);
// tidak ada retry, tidak ada alert, tidak ada tracking
```

Tidak ada retry mechanism, tidak ada alert, tidak ada cara untuk tahu berapa banyak data yang tidak tersinkron. Kalau ada 1000 submission dan 50 gagal dual-write, data di `responses_v2` akan tidak lengkap dan analytics v2 akan salah.

Solusi: Tambahkan retry dengan exponential backoff, atau gunakan database transaction untuk memastikan atomicity.

🟤 Masalah #14: Neon Client Dibuat Ulang Setiap Request

```
// src/lib/db/sql.js
export function getSqlClient(env) {
  return neon(databaseUrl); // dipanggil setiap kali ada query
}
```

`neon()` membuat koneksi baru setiap dipanggil. Tidak ada connection pooling atau singleton pattern. Di Cloudflare Workers ini acceptable karena stateless, tapi ini tetap menambah latency yang tidak perlu.

Solusi: Untuk Cloudflare Workers, pertimbangkan menggunakan Neon's HTTP driver yang sudah dioptimalkan untuk serverless, atau cache client per request context.

🟤 Masalah #15: Bootstrap Migration Tidak Bersih

Di `initializeSchema()`, ada redundansi seperti:

```
-- Kolom dibuat di CREATE TABLE
CREATE TABLE IF NOT EXISTS ai_analysis_v2 (
  legacy_ai_analysis_id BIGINT UNIQUE, -- sudah ada di sini
  ...
);

-- Lalu di-ALTER lagi
ALTER TABLE ai_analysis_v2
ADD COLUMN IF NOT EXISTS legacy_ai_analysis_id BIGINT; -- redundan!
```

Ini menunjukkan bootstrap adalah **akumulasi patch** bukan desain yang bersih. Setiap kali ada perubahan schema, developer menambahkan ALTER TABLE di bawah tanpa membersihkan CREATE TABLE yang sudah

ada.

Solusi: Refactor bootstrap menjadi versioned migrations yang bersih (misal menggunakan format `001_initial.sql`, `002_add_tenants.sql`, dst).

🟡 Masalah #16: Tidak Ada Input Sanitization untuk Nama dan Teks Bebas

Validasi untuk `namaGuru` dan `mataPelajaran` hanya cek `min(1)`:

```
namaGuru: z.string().trim().min(1, 'Nama Guru wajib diisi'),
mataPelajaran: z.string().trim().min(1, 'Mata Pelajaran wajib diisi'),
```

Tidak ada:

- Max length (bisa diisi 10.000 karakter)
- Sanitization XSS
- Validasi karakter yang diizinkan

Kalau data ini ditampilkan di dashboard tanpa escaping yang proper, bisa menjadi XSS vector.

Solusi: Tambahkan `.max(200)` untuk nama, dan pastikan semua output di frontend di-escape dengan benar.

🟡 Masalah #17: `public/forms/` — Status Tidak Jelas

Ada `public/forms/portal.html`, `public/forms/app.js`, `public/forms/portal.js` — ini adalah UI baru untuk multi-tenant. Tapi `public/index.html` dan `public/script.js` adalah UI lama untuk single-school.

Dua UI untuk hal yang sama, tidak ada dokumentasi mana yang aktif untuk path mana, dan tidak ada cara mudah untuk tahu mana yang sedang dikembangkan.

Solusi: Dokumentasikan di README mana yang aktif dan untuk path apa. Pertimbangkan menghapus UI lama setelah migrasi selesai.

---

5. Ringkasan Penilaian

Aspek	Nilai	Komentar
Arsitektur	6/10	Migrasi multi-tenant sudah bagus, tapi transisi belum bersih
Security	4/10	PBKDF2 lemah, no rate limit, no CSRF, bug session
Code Quality	5/10	Duplikasi module, dead code, dua DB client
Reliability	6/10	Dual-write tanpa retry, bootstrap terlalu berat
Maintainability	4/10	Sangat sulit dipahami developer baru
Production Readiness	6/10	Sudah jalan, tapi ada bug session dan security gaps

Nilai Keseluruhan: 5.2/10

---

6. Rekomendasi Prioritas

🔴 SEGERA (Minggu Ini)

#	Aksi	File yang Terdampak
1	**Fix bug** `session.is_active` — tambah kolom ke tabel `sessions` atau hapus pengecekan	`db/schema.sql`, `src/modules/auth/service.js`
2	**Naikkan PBKDF2 ke 600.000 iterasi**	`src/lib/security/hash.js`
3	**Hapus atau tandai jelas** `worker.js` root sebagai deprecated	`worker.js`

🟠 JANGKA PENDEK (Bulan Ini)

#	Aksi	File yang Terdampak
4	**Hapus** `src/modules/submission/` **dan** `src/modules/form/` (singular)	Seluruh folder
5	**Hapus** `src/lib/db/client.js` — pakai hanya `sql.js`	`src/lib/db/client.js`
6	**Pisahkan bootstrap dari runtime** — migration via script saja	`src/lib/db/bootstrap.js`
7	**Tambah rate limiting** di login endpoint	Cloudflare WAF rules
8	**Hapus dead code** `viewDashboardBtn` di `script.js`	`public/script.js`

🟡 JANGKA MENENGAH (3 Bulan)

#	Aksi	File yang Terdampak
9	**Tambah retry mechanism** untuk dual-write failures	`src/modules/submissions/service.js`
10	**Tambah AI cost control** — cooldown, daily limit per tenant	`src/worker.js`, AI routes
11	**Implementasikan CSRF protection** yang sesungguhnya	`src/lib/http/request-guards.js`
12	**Audit dan bersihkan** `public/` — hapus dashboard lama	`public/dashboard.*`
13	**Refactor bootstrap** menjadi versioned migrations	`src/lib/db/bootstrap.js`
14	**Tambah max length** untuk input teks bebas	`src/modules/submissions/validation.js`

---

7. Kesimpulan Jujur

App ini adalah **produk yang sedang dalam transisi besar** — dari single-school Express app ke multi-tenant Cloudflare Worker. Transisi itu sudah sekitar 70% selesai dan secara fungsional sudah berjalan di production.

Tapi **70% selesai artinya 30% masih berantakan**, dan berantakannya ada di area yang paling penting: security, reliability, dan maintainability.

Yang sudah bagus:

- Arsitektur multi-tenant dengan `schools`/`tenants` sudah solid
- Validasi input dengan Zod sudah benar
- Error handling di controller sudah cukup baik
- Dual-write pattern untuk migrasi data sudah ada (meski rapuh)
- AI prompt template system sudah fleksibel
- PDF export dari analisa AI sudah impressive

Yang paling mengkhawatirkan:

- Bug `session.is_active` — kalau ini benar-benar ada di production, maka tidak ada yang bisa tetap login ke admin panel. Perlu dicek segera.
- PBKDF2 10.000 iterasi — ini kelemahan keamanan nyata yang perlu diperbaiki sebelum ada lebih banyak user admin.
- Tidak ada rate limiting di login — ini undangan untuk brute force.

Secara keseluruhan: Kode ini ditulis oleh developer yang tahu apa yang mereka lakukan, tapi sedang dalam tekanan untuk deliver cepat sambil migrasi arsitektur besar. Hasilnya adalah technical debt yang perlu dibayar sebelum scale lebih jauh. Prioritaskan security fixes dulu, baru bersihkan technical debt.

Laporan ini dibuat berdasarkan analisis statis kode sumber. Beberapa masalah mungkin sudah diperbaiki di environment production melalui konfigurasi atau migration yang tidak tercatat di repo.
