

Technical Due Diligence Report

SampleCo (anonymized), prepared by Alexandria Solutions LLC for KYE Ventures. Sample report, October 2026. Signed reports carry the reviewer's name and date.

Target: SampleCo, a studio booking and management platform (web admin, public booking page, early mobile app) **Prepared by:** Alexandria Solutions LLC **Date:** 2026-10-19 **Basis:** Read-only review of the source repository at one point in time. Nothing was run, installed, or modified. Secrets files were not opened.

1. Executive summary

SampleCo is a small, well-structured codebase (about 25k lines of Go, 11k of SQL, 31k of TypeScript) with a strong payment and booking core and weak surrounding operations. The money paths (processor webhooks, server-side pricing, seat capacity under concurrency) are verified by tests against a real database. Almost everything around them (environments, monitoring, access control hygiene, team continuity) is at prototype maturity.

Axis	Rating	Reason
Architecture	Medium	Clean layering and atomic booking logic, but hard-wired to one studio and keeps state in process memory.
Security	High	A real owner login and customer data are committed to the repo; no rate limiting; day-long tokens in browser storage with no refresh or revocation. Payment security itself is good.
Code quality and tests	Medium	Backend tests exceed production code in size. Frontend has zero unit tests; end-to-end tests run against production.
Infrastructure and operations	High	No staging, deploys not gated on CI, no error monitoring, backups unverified on what appears to be a free database tier.

Axis	Rating	Reason
Team and bus factor	High	One human committer for roughly four in five commits, the rest from automated coding agents. Dormant for several months before the review.

Three findings that most affect valuation

- 1. Bus factor of one, project dormant.** Under 200 commits, nearly all landed in two short bursts about five months apart. The only other human commit is a planning document. The buyer acquires code, not a team, and most of that code was produced by AI agents under one reviewer. Budget a 4 to 8 week ramp for any new engineer.
- 2. Hygiene failures that are cheap to fix but costly as signals.** The committed seed file holds a real owner's email, password hash, and plaintext password in a comment; the same credentials drive end-to-end tests that default to the production site. One day to fix; what it says about process is the real cost.
- 3. Single-tenant by construction.** Every tenant check compares against one hardcoded studio ID. If the thesis is "platform for many studios," the data model does not support it yet (15 to 25 engineer-days).

Recommendation: Caution. Proceed only if: (a) credentials are rotated and the seed scrubbed before data room access widens; (b) a holdback is tied to the 30-day plan in section 5; (c) backup retention and a tested restore are confirmed in writing; (d) IP assignment covers AI-generated code and the bundled analysis tool whose container labels point at a third-party repository.

2. Scope and method

Examined: README and planning docs; all package manifests, the lockfile, and the Go module file; CI workflows, Dockerfiles, hosting configs, docker-compose; all migration pairs (about three dozen), with five read in depth (base tables and RBAC, booking spine, webhook idempotency, per-occurrence capacity, system-role grants); API entry point, router, auth and error middleware, config, database layer; login, registration, payment webhook, payment intent, and booking creation paths; shared API client and auth stores; web routing; test directories, Playwright config and fixtures; git history and remote branches; the seed file header; the team's own progress ledger.

Not examined: `.env*` and secret-like files (by policy); running services, the live database, hosting dashboards, the payments processor account, DNS, contracts. No tests, linters, or builds were run; coverage figures are the team's claims unless stated.

Effort: about six senior-engineer hours. Statements based on comments or configuration rather than observed behavior are marked *inferred* or *not verified*.

Identifying details, vendor names and revision identifiers have been removed from this sample; a client report carries them in full.

3. Findings by axis

Severity: Critical, High, Medium, Low. Effort in engineer-days.

3.1 Security

S1. Real credentials and customer PII committed. High. Evidence:

`apps/api/seeds/<owner>_data.sql` lines 2 and 30 to 33 (filename redacted; it contains a real name) hold a real owner email, bcrypt hash, and the plaintext password in a comment.

`apps/web/e2e/fixtures/auth.fixture.ts` lines 10 to 11 reuse the login;

`apps/web/playwright.config.ts` line 17 targets production by default;

`apps/api/internal/integration/harness_test.go` loads the seed. Impact: anyone with repo access can log into the production owner account. Also a consent question about whose data sits in fixtures. Fix: rotate, replace with synthetic data, read fixture credentials from environment, treat history as burned. **1 to 2 days.**

S2. Day-long bearer tokens in localStorage, no refresh, no revocation. Medium. Evidence:

day-long expiry hardcoded in `apps/api/internal/usecase/auth/login.go` line 71 while

`JWT_EXPIRATION_MINUTES` (`apps/api/internal/infrastructure/config/config.go` line 22) is never read. A refresh token is minted (`login.go` lines 92 to 99) but

`apps/api/internal/adapters/handler/router.go` lines 197 to 199 expose only `/login`. Tokens persist in `window.localStorage` (`packages/stores/src/auth-store.ts` lines 112 to 116). HS256 secret has no length check (`config.go` lines 59 to 62). No password reset, email verification, logout, or MFA found. Fix: honor expiry, add refresh and logout-all, enforce secret length, add reset and logout. **3 to 5 days.**

S3. No rate limiting, including on unauthenticated writes. Medium. Evidence:

no rate-limit middleware anywhere in the API. `POST /api/v1/public/bookings` (`router.go` lines 226 to 229) runs under the trusted `system` role (`apps/api/internal/adapters/handler/booking_handler.go` lines 132 to 147) and creates customers and bookings for anyone. Login and registration do bcrypt per request. Impact: spam customers, seat squatting, email cost, CPU exhaustion on a single small instance. **1 to 2 days.**

S4. Role checks live only in the database and are not total. Low (not verified per function). Evidence: 178 `check_role` calls across 203 SQL functions in `apps/api/migrations/*.up.sql`; the HTTP `RequireRole` middleware (`apps/api/internal/adapter/middleware/auth.go` line 83) has zero call sites; three repositories bypass the function layer with direct pool queries (`user_repository.go`, `customer_auth_repository.go`, `service_repository.go`). About 25 functions need individual review. **2 days.**

S5. Payment path is sound. Informational. Evidence: signature verification refuses to run without a webhook secret (the payments client under `apps/api/internal/infrastructure/`, lines 300 to 311); idempotency ledger with primary key and `ON CONFLICT DO NOTHING` (the webhook idempotency migration under `apps/api/migrations/`, lines 19 to 20 and 50); charged amount read from the booking, client value ignored (the payment intent use case under `apps/api/internal/usecase/`, lines 53 to 73). One gap: fees are estimated by formula (the webhook use case, line 129) rather than read from the processor, so recorded margins drift. **1 day.**

S6. Customer names and emails go to a third-party LLM. Low.

`apps/api/internal/infrastructure/ai/prompts.go` lines 222 to 250. Needs a data processing agreement and disclosure. **0.5 day plus legal.**

S7. Other. Secrets handling is otherwise sound (env-driven config, no secret literals in app code). Errors mask internals (`middleware/error.go` lines 109 to 117). No security headers anywhere. Thirty-two handlers use `uuid.MustParse` on path IDs, so a malformed ID returns 500 not 400. **1 day combined.**

3.2 Architecture

A1. Single-tenant by construction. High if multi-studio is the thesis, else Low. Evidence: `apps/api/internal/pkg/tenant/tenant.go` line 9 hardcodes the studio ID; `login.go` line 74 stamps it into every token; `studio_id` exists only on users (the studios table migration under `apps/api/migrations/`, line 118). Bookings, customers, services, payments have no tenant column. **15 to 25 days.**

A2. In-process state blocks scaling. Medium. Chat history and OAuth state are in memory (`apps/api/cmd/api/main.go` lines 378, 442, with a “replace with Redis” comment) while the container host config allows a second machine to start. **2 to 3 days.**

A3. Logic split between Go and PL/pgSQL. Medium (maintainability). Pricing, capacity, credits, waivers, and refunds live in 10.8k lines of SQL functions (for example the per-occurrence capacity migration under `apps/api/migrations/`, lines 179 to 253). Defensible and correct, but it requires engineers fluent in both. Hire accordingly.

A4. Mobile app is a shell. Low. 34 files, placeholder build-service project ID in the mobile app manifest (line 40), no API client wiring found. *Inferred*: never shipped. Cut or fund.

3.3 Code quality and tests

Q1. Backend tests are an asset. 63 test files, 27,375 test lines vs 25,206 production lines; 12 integration suites run against Postgres in CI (`.github/workflows/ci.yml` lines 22 to 35, 60 to 64); a 90 percent use-case coverage gate in the root `Makefile`. The ledger (`docs/prd/PROGRESS.md` line 19) claims 98.9 percent; *not verified*.

Q2. No frontend unit tests; E2E not in CI. Medium. Zero `*.test.*` files under `apps/web/src`, `apps/mobile`, `packages`; CI test steps commented out (`ci.yml` lines 98 to 100, 131 to 133). Seventeen Playwright specs exist, no workflow runs them, default target is production (see S1). **3 days plus I1.**

Q3. Non-reproducible tooling, EOL versions. Low. `golangci-lint` pinned to `latest` (`ci.yml` line 52) with no config; ESLint 8 (EOL); CI pins Go 1.22 (`ci.yml` line 9) while `go.mod` says 1.24 and `.tool-versions` says 1.25.6. The in-repo static analysis tool was dropped from CI before the reviewed revision after reporting over 200 violations (`CI_STATUS.md`). **1.5 days.**

Q4. Silent stubs and stale docs. Low. `postgres/user_repository.go` lines 121 to 152: `Create`, `Update`, `Delete` return `nil` and do nothing (unrouted today). Root-level `CI_STATUS.md`, `TEST_STATUS.md`, `FINDINGS.md`, and a payments implementation summary describe superseded states and will mislead new hires. **1 day.**

3.4 Infrastructure and operations

I1. No staging; deploys not gated on CI. High. `deploy-api.yml` and `deploy-web.yml` fire on push to `main` with no dependency on the CI workflow. No staging config exists. Migrations auto-run at boot (`main.go` lines 63 to 71) with no rollback runbook. A red build still ships. **3 to 4 days.**

I2. No error monitoring, metrics, or alerting. High. An error-monitoring DSN is declared (`config.go` line 41) and never used; logging is JSON to stdout; the web error boundary logs to console (`apps/web/src/components/ErrorBoundary.tsx` lines 34 to 35). The `/health` route does ping the DB (`main.go` lines 484 to 497). **2 days.**

I3. Backups unverified. High until verified. Comments in the container host config (lines 9 to 13) describe the hosted Postgres provider's free-tier pooler limit. *Inferred*: free tier, so limited or no point-in-time recovery. No backup or restore doc in the repo. **1 day plus hosting.**

I4. EOL runtimes and images. Medium. `apps/api/Dockerfile` line 20 uses `alpine:3.19` (EOL November 2025); `.nvmrc` pins Node 20 (EOL April 2026). Eight Dependabot branches sit unmerged. **1.5 days.**

3.5 Team and bus factor

T1. One human. High. Under 200 commits, concentrated in two short bursts about five months apart; roughly four in five from one person under multiple identities, the rest from automated coding agent identities with matching agent-named remote branches, plus one docs commit from the business owner. `.github/CODEOWNERS` names teams that *inferred* do not exist. Mitigation: unusually thorough written conventions (the repo’s agent conventions file, `docs/coding/`, `docs/prd/`). Insist on a documented handover and 60 to 90 days of founder availability.

T2. Licensing and provenance. Medium. README says “Proprietary”; no LICENSE file; `package.json` has no license field. The in-repo static analysis tool’s Dockerfile labels it MIT with a source label pointing at a third-party organization, so ownership needs confirming. Runtime dependencies appear permissively licensed; *not verified* by SBOM. Ensure the agreement assigns AI-generated code explicitly.

4. Stack and architecture map

A pnpm and Turborepo monorepo: three apps, three shared packages. The API is Go with Chi and pgx in Clean Architecture form: handlers validate and delegate, use cases orchestrate, repositories call PostgreSQL functions by name through a helper that sets `app.current_user_id` and `app.current_user_role` on the connection first (`apps/api/internal/pkg/dbutil/dbutil.go` lines 54 to 66). Authorization and most invariants live in those SQL functions across about three dozen migrations on PostgreSQL 16. The web admin and public booking pages are a React 18 and Vite SPA on a static hosting provider, calling the API on a single small container host (one shared-CPU machine, one region). Mobile is a React Native toolchain. External services: the payments processor (payments, subscriptions, and the processor’s connected-accounts product), a transactional email API, and a third-party LLM API (analytics chat). The database is *inferred* from config comments to be a hosted Postgres provider on a free tier.

Dependency inventory (manifests and lockfile):

Component	Version	Note
Go	1.24.12 toolchain; 1.22 in CI	Both outside Go’s two-release support window; align to 1.25
go-chi/chi	v5.0.12	Fine
jacke/pgx	v5.5.4	Several minors behind; check advisory database
golang-jwt/jwt	v5.3.1	Current

Component	Version	Note
payments processor Go SDK	current major	Supported
golang-migrate	v4.19.1	Fine
Node	20.18.1	Past EOL; move to 22
pnpm	9.15.1	Fine
React	18.2	Fine
Vite	^5.1 (esbuild 0.21.5)	Two majors behind; esbuild dev-server advisory below 0.25; check advisory database
Vitest	^1.3	Two majors behind
ESLint	^8.57	End of life
TypeScript	^5.3	Fine, strict on
TanStack Query / Zustand / Zod	^5.17 / ^4.5 / ^3.22	Fine
react-router-dom	^6.22	Fine
Playwright	^1.58	Fine
payments processor browser SDK	current major	Fine
React Native toolchain	several SDK releases behind	Check vendor support window and store build rules
PostgreSQL	16	Supported to 2028
alpine runtime image	3.19	Past EOL
Transitive of note	ws 8.19, undici 6.23, rollup 4.57, send 0.18.0	send 0.18 predates a path-handling fix; check advisory database

5. 30-60-90 day plan

Assumes one senior contractor at about 1,200 USD per day.

Days 0 to 30 (14 days, about 17k USD). Rotate the owner credential; synthetic seed and fixture (S1). Rate limiting (S3). Honor token expiry, add refresh and logout, enforce secret length (S2 part one). Gate deploys on CI; enable frontend tests in CI (I1 part, Q2). Wire error monitoring and an

uptime check (I2). Upgrade Alpine, align Go, pin golanci-lint, add security headers (I4, Q3, S7). Confirm DB plan, enable PITR, run one restore (I3).

Days 31 to 60 (18 days, about 22k USD). Staging environment and Playwright against it in CI (I1, Q2). Move in-memory stores to Postgres or Redis (A2). Password reset, verification, logout (S2 part two). Audit functions lacking `check_role` (S4). Upgrade Node, Vite, Vitest, ESLint; clear Dependabot queue (I4). Validated UUID parsing; fees from the processor's balance transactions (S7, S5). Prune stale docs (Q4).

Days 61 to 90 (25 days, about 30k USD). Decide tenancy: add tenant columns everywhere (A1) or formally position as one deployment per studio. Decide the mobile app (A4). Runbooks and an incident drill. Data processing agreement and PII minimization for the LLM feature (S6). SBOM, license scan, tool provenance (T2). Onboard a second engineer against the written conventions (T1).

Total: about 57 engineer-days, 65k to 75k USD, excluding hosting and legal.

6. Appendix: 40-item checklist

#	Check	Result	Evidence
1	Parameterized SQL only	Pass	<code>dbutil.go</code> validates function names; placeholders throughout
2	Request body validation	Pass	<code>middleware/validation.go</code> , struct tags on inputs
3	Auth on protected routes	Pass	<code>router.go</code> line 235
4	Role-based authorization	Partial	DB checks on 178 of 203 functions; HTTP <code>RequireRole</code> unused
5	Tenant isolation	Fail	Hardcoded studio ID (<code>tenant.go</code> line 9)
6	Password hashing	Pass	<code>bcrypt</code> (<code>register.go</code> line 46)
7	Token expiry configurable and honored	Fail	Day-long expiry hardcoded; config unused
8	Token refresh and revocation	Fail	No refresh route
9	Password reset and email verification	Fail	Not found

#	Check	Result	Evidence
10	Lockout or MFA	Fail	Not found
11	Rate limiting	Fail	Not found
12	Secrets excluded from git	Partial	<code>.gitignore</code> correct; plaintext password in seed comment
13	No secret literals in app code	Pass	Only in tooling fixtures and docs
14	CORS restricted	Pass	Env allow list (<code>main.go</code> line 475); prod value not verified
15	Security headers	Fail	None set
16	Errors do not leak internals	Pass	<code>error.go</code> lines 109 to 117
17	Panic recovery	Pass	<code>middleware.Recoverer</code>
18	Webhook signature verification	Pass	Payments client lines 300 to 311
19	Webhook idempotency	Pass	The webhook idempotency migration
20	Server-side pricing	Pass	<code>payment_intent.go</code> lines 53 to 73
21	Concurrency-safe capacity	Pass	Advisory locks in the per-occurrence capacity migration
22	PII minimization to third parties	Fail	Names and emails sent to LLM
23	Backend unit tests	Pass	63 files, 27k lines
24	Backend integration tests in CI	Pass	Postgres service in <code>ci.yml</code>
25	Coverage gate	Pass (claimed)	<code>Makefile</code> 90 percent gate; not run
26	Frontend unit tests	Fail	Zero files; CI step commented out
27	E2E in CI against non-production	Fail	Not in CI; targets production
28	Lint and typecheck in CI	Pass	<code>ci.yml</code>

#	Check	Result	Evidence
29	Reproducible toolchain pins	Fail	Go versions disagree; lint on <code>latest</code>
30	Dependency update automation	Partial	Dependabot on; branches unmerged
31	No EOL runtimes or images	Fail	Node 20, Alpine 3.19, ESLint 8, Go 1.22
32	Deploy gated on CI	Fail	Independent push triggers
33	Staging environment	Fail	None found
34	Migration rollback strategy	Partial	Down files exist; auto-run at boot, no runbook
35	Health check	Pass	DB ping on <code>/health</code>
36	Error monitoring and alerting	Fail	Error-monitoring DSN unused
37	Backups and tested restore	Not verified	No evidence in repo
38	Runbooks and incident process	Fail	None found
39	LICENSE file and license scan	Fail	Neither present
40	Onboarding documentation	Pass	README, agent conventions file, <code>docs/coding/</code> , <code>docs/prd/</code>

Tally: 17 pass, 4 partial, 18 fail, 1 not verified.

Limits of this report

One reviewer, one repository snapshot, about six hours, static reading only. No code, tests, or linters were run; no live infrastructure, hosting dashboard, payments processor account, DNS, or contract was inspected; environment and secrets files were not opened. Coverage and “gates green” claims come from the team’s ledger and were not reproduced. Dependency risk was judged from manifests without a scanner; resolve every “check advisory database” note with a scan before closing. Line numbers will drift. Items marked inferred rest on comments and configuration, not observed behavior.

A full assessment should add a penetration test of the public booking and auth endpoints, a restore drill, and an interview with the founding engineer.