

TECHNICAL DUE DILIGENCE

Fieldline

Field service scheduling platform (web, iOS, Android)

PREPARED FOR
Sample Ventures

PREPARED BY
Kurb Media

DATE
October 2026

CONFIDENTIAL

SAMPLE REPORT

The organization, people, and findings in this report are fictional.

Summary

OVERALL

Proceed, with conditions

Fieldline's core product is well built for its stage, and the team ships steadily. The stack is modern and widely supported, the code is consistent and tested where it matters most, and releases go out weekly with review on every change.

Two issues should be resolved before close. Production database credentials are shared across the team, and most infrastructure knowledge sits with one engineer. Neither is unusual at this stage, and both are fixable without slowing the roadmap.

STRENGTHS

- Modern, well-supported stack: TypeScript, React, React Native, Node.js, and PostgreSQL on AWS.
- Clean, consistent codebase with strong automated tests around the scheduling engine, the heart of the product.
- Healthy delivery habits: weekly releases, code review on every change, and tests that run before anything ships.

TOP RISKS

- Shared production database credentials
◆ **CRITICAL** R1
- Key-person dependency on infrastructure
◆ **HIGH** R2
- No multi-factor authentication on the AWS root account
◆ **HIGH** R3

Scope and method

WHAT WE REVIEWED

- Source code for the web app, mobile app, and API
- AWS infrastructure, deployment pipeline, and backups
- Security practices, access control, and data handling
- Third-party services and open-source licenses
- Interviews with the CTO and two senior engineers
- The current product roadmap

NOT IN SCOPE

- Penetration testing
- Formal legal review of intellectual property
- Financial review

How we worked. We reviewed the code and infrastructure hands-on with read-only access, then walked through our findings with the engineering team to confirm context before writing this report.

Findings by area

RATINGS AT A GLANCE

Architecture ◆ SOUND	Code quality ◆ STRONG	Security ◆ CRITICAL	Scalability and performance ◆ NEEDS ATTENTION
Infrastructure and DevOps ◆ SOUND	Team and process ◆ NEEDS ATTENTION	Third-party services and licensing ◆ SOUND	Tech debt ◆ NEEDS ATTENTION

Architecture

SOUND

A single API service with a React web app and a React Native mobile app. A good fit for the company's stage, with clear seams for growth.

- Web and mobile share types and business rules with the API, which keeps behavior consistent across platforms.
- Notification sending is tied to web requests and will need to move to a background queue as volume grows **R5**.

Code quality

STRONG

Consistent, typed, and readable. The scheduling engine is well tested and well documented.

- Linting and formatting are enforced automatically on every change.
- The older billing module has few tests and is changed rarely and carefully **R7**.

Security

CRITICAL

Good fundamentals around customer data, undercut by weak access control to production systems.

- Customer data is encrypted at rest and in transit.
- Six people share one set of production database credentials, with no record of who did what **R1**.
- The AWS root account does not require multi-factor authentication **R3**.
- Two dependencies have known high-severity vulnerabilities with patches available.

Scalability and performance

NEEDS ATTENTION

The database has plenty of headroom. The bottleneck is how notifications are sent.

- Reminder texts and emails are sent while the user waits, which slows the app at peak scheduling times **R5**.
- Load on the database is modest, and indexes are in good shape.

Infrastructure and DevOps

SOUND

Most infrastructure is defined in code, and the deployment pipeline is reliable.

- Web and API deploys are automated. Mobile releases still involve manual steps.
- Nightly database backups run, but a restore has never been tested **R4**.

Team and process

NEEDS ATTENTION

A capable team of five engineers with good review habits, and a single point of failure on infrastructure.

- One engineer holds most of the knowledge needed to deploy, debug, and recover production **R2**.
- There is no documented on-call rotation or incident process.

Third-party services and licensing

SOUND

Mainstream, well-supported vendors and no licensing concerns in the shipped product.

- Payments, maps, and messaging run on established providers with standard contracts.
- An internal admin tool uses an AGPL-licensed library. Low risk while the tool stays internal

R8

Tech debt

NEEDS ATTENTION

Manageable overall, concentrated in two places.

- The mobile app is two major versions behind its framework, which will get harder to upgrade the longer it waits R6.
- The legacy billing module works but is brittle R7.

Risk register

Every risk we found, ranked by severity. Severity reflects the impact on the business if the risk is left alone. Definitions are in the appendix.

ID	RISK	AREA	SEVERITY
R1	Shared production database credentials Six people use the same login. Access can't be revoked for one person or audited after the fact.	Security	CRITICAL
R2	Key-person dependency on infrastructure If one engineer is unavailable, the team would struggle to deploy or recover from an outage.	Team and process	HIGH
R3	No multi-factor authentication on the AWS root account A single leaked password would give full control of all production infrastructure.	Security	HIGH
R4	Backups have never been test-restored Backups run nightly, but there is no evidence they can be restored when needed.	Infrastructure and DevOps	MEDIUM
R5	Notifications sent during web requests Slows the app at peak times and will limit growth without a background queue.	Scalability and performance	MEDIUM
R6	Mobile framework two major versions behind Upgrades get harder over time, and app store requirements may eventually force one.	Tech debt	MEDIUM
R7	Untested legacy billing module Changes to billing carry more risk than the rest of the codebase.	Code quality	LOW
R8	AGPL library in an internal admin tool Fine while internal. Would need review if the tool were ever offered to customers.	Third-party services and licensing	LOW

Recommendations

Resolve before close

We recommend making these conditions of the investment.

01. Give each person their own production database access, with audit logging, and rotate the shared credentials. **R1**
02. Require multi-factor authentication on the AWS root account and every user with production access. **R3**
03. Document how to deploy and recover production, and train a second engineer to do both. **R2**

Address after close

01. Test a full database restore, then repeat it on a regular schedule. **R4**
02. Move notification sending to a background queue. **R5**
03. Plan and schedule the mobile framework upgrade. **R6**
04. Add tests around billing before the next change to it. **R7**
05. Patch the two vulnerable dependencies and keep dependencies current.
06. Keep the admin tool internal, or replace the AGPL library before sharing it. **R8**

Questions for the founding team

Worth asking before close. The answers will sharpen or soften several of the risks above.

1. Today, who besides your infrastructure engineer can deploy and recover production?
2. Is the billing rewrite on the roadmap, and who would own it?
3. Have any customers asked for a security review or SOC 2 report?
4. How are mobile releases tested before they reach the app stores?

Appendix: how to read this report

AREA RATINGS

◆ STRONG	Better than we'd expect at this stage. No action needed.
◆ SOUND	Right for this stage. Minor improvements only.
◆ NEEDS ATTENTION	Will hold the company back if left alone. Plan to address.
◆ CRITICAL	A material risk to the business today. Address now.

RISK SEVERITY

◆ CRITICAL	Could cause a serious breach, outage, or loss of data. Resolve before close.
◆ HIGH	Likely to cause real harm or slow the business. Resolve before or soon after close.
◆ MEDIUM	Will limit growth or add cost over time. Plan to address.
◆ LOW	Worth knowing. Address when convenient.