

PRODUCTION HARDENING SPRINT

Technical Due Diligence Pack

Crewproof, Inc.

APPLICATION	Crewproof web app and client portal (app.crewproof.com)
STATE	At handover, Day 10. All documents are version 1.0.
PREPARED FOR	The lead investor's technical advisor, who reviews the app for the seed round
SHARED BY	Maya Okafor, CEO of Crewproof, Inc. (the company that owns the app)
PREPARED BY	AxonBuild engineering team

7

documents in one file

81

pages, numbered as one
pack

123

scope items accounted
for

0

applicable items open

Confidential. Shared for the purpose of technical due diligence only. Do not forward without Crewproof's permission.

ABOUT THIS PACK

We (AxonBuild) put together this due diligence pack at the end of a 10-day Production Hardening Sprint for Crewproof, Inc. Crewproof makes an operations app for commercial cleaning contractors in the US. Each contractor uses it to schedule cleaning jobs, guide its crews and bill its own clients.

The pack is for the lead investor's technical advisor. It holds seven technical documents in one file, and it answers the usual due diligence questions with evidence. Start with the question index on the next page: it gives a short answer to each question and says where to read more. This pack is item 13.6 of the sprint scope.

CONTENTS

What this pack contains

One file with every technical document a reviewer needs for Crewproof.

This table lists the seven documents in reading order. Each document keeps its own page numbers. The pack page number is at the bottom center of every page, and each document has a bookmark. The "Scope item" column gives the number of the sprint task that produced the document.

#	DOCUMENT	SCOPE ITEM	PACK PAGES
1	Production Readiness Report	13.1	5 to 62
2	System Architecture	13.2	63 to 66
3	Data Model	4.13	67 to 70
4	Controls Checklist	12.6	71 to 74
5	Capacity Statement	9.6	75 to 77
6	Subprocessor List	12.8	78 to 79
7	Licence Report	10.11	80 to 81

How to use this pack

- **Start with** the summary (Production Readiness Report, section 1) and the remaining risks (the same report, section 8).
- **To answer a specific question**, use the question index on the next page.
- **To check a claim**, ask Crewproof for access to the evidence folders. Each claim has an evidence code, explained on the next page.
- **Send questions** to Maya Okafor (CEO, Crewproof). We (AxonBuild) answer technical questions in the shared channel until handover + 30 calendar days.

QUESTION INDEX

Short answers to the usual due diligence questions

This table answers 22 questions that technical advisors usually ask. Each row gives a short plain answer first. The right column says where to read the full answer and its evidence. "Readiness Report" is the Production Readiness Report, document 1 in this pack. The box below explains the codes in the right column.

What the codes mean

- **Item numbers, such as item 1.4**, are tasks of the sprint scope (123 items in 13 areas, published at axonbuild.com/scope). Section 7 of the Readiness Report gives the result for each item.
- **EV- codes, such as EV-1.4**, name a folder of proof: test output, screenshots, settings exports and recordings. The folders are in evidence/ in the handover repository.
- **C- codes, such as C-01**, are the 48 controls (safeguards) in the Controls Checklist.
- **R1 to R8** are the remaining risks in section 8 of the Readiness Report. The appendices A to F are at the end of the Readiness Report.

#	QUESTION AND SHORT ANSWER	WHERE TO READ MORE
Q01	Can one customer see another customer's data? No. Each contractor company can see only its own records in all 24 database tables (row-level security). 312 automatic tests check this on every change.	Readiness Report: item 1.4 and Appendix A. Data Model: who can read each table. Controls Checklist: C-01.
Q02	Who can do what in the application? Each of the five roles (owner, admin, manager, crew and client) has a fixed set of the 38 actions. The server enforces every answer, and a test checks it.	Readiness Report: Appendix A (role matrix). Controls Checklist: C-02 to C-04.
Q03	Could an attacker take over an admin account? No known way remains. Owners and admins must use two-factor authentication, only a checked server function can change a role, and sign-in attempts are limited.	Readiness Report: items 1.3, 1.5, 1.6 and 1.11. Controls Checklist: C-05 and C-08.
Q04	Are API keys and secrets kept safe? Yes. No private key is in the browser code any more. We replaced all 5 exposed keys, and a check on every change blocks new ones.	Readiness Report: items 2.1 to 2.6. System Architecture: the table of what runs where (which parts hold keys). Controls Checklist: C-12 to C-17.
Q05	Does the company own its code, hosting and accounts? Yes. Crewproof, Inc. owns all 13 provider accounts, including the code (GitHub) and the hosting (Vercel). Maya Okafor is the owner of each account, and Crewproof pays the bills.	Readiness Report: items 2.8 and 7.11. System Architecture: the table of what runs where. Controls Checklist: C-07.
Q06	Has it been tested by an outside attacker? Yes, in part. We attacked the five riskiest parts of the app with Crewproof's written permission and fixed all 3 findings. A full outside test is still recommended (risk R6).	Readiness Report: items 3.7 and 3.10, Appendix B, and section 8.
Q07	Are payments and billing reliable? Yes. The app now checks that each payment message really comes from Stripe and processes it only once. A nightly job compares the billing records and alerts on any mismatch.	Readiness Report: items 5.1 to 5.10. Controls Checklist: C-27.
Q08	Can it recover from losing its database? Yes. In a timed drill we rebuilt the app from backups in 48 minutes, and about 2 minutes of database changes were lost. Photos can lose up to 24 hours of uploads (risk R3).	Readiness Report: items 4.7 and 4.12, and Appendix C. Controls Checklist: C-40 and C-41.
Q09	How many users can it handle, and what does growth cost? The app handled 400 users at the same time in a test, about 4 times today's peak. The Capacity Statement gives the changes and the monthly cost for 2,000 users, as an estimate.	Capacity Statement (the whole document). Readiness Report: item 9.2.

#	QUESTION AND SHORT ANSWER	WHERE TO READ MORE
Q10	How would the team know something is broken? Crewproof's team gets an alert. Error tracking, uptime checks every minute and threshold alerts go to a named channel. An outage raises an alert in about 2 minutes.	Readiness Report: items 6.3 and 8.1 to 8.8. Controls Checklist: C-35 to C-39.
Q11	How are changes tested and released? Each change is a pull request that a person reviews and that must pass the automatic checks. It goes to staging first and reaches the live app only after a manual approval.	Readiness Report: area 07 in section 7. System Architecture: the delivery pipeline in the diagram. Controls Checklist: C-28 to C-34.
Q12	Is the AI-built code maintainable by a new engineer? Yes. The code is cleaned up, strictly typed, tested and documented. But Luis Ferreira (Crewproof's part-time contract developer) still holds most of the knowledge (risk R1).	Readiness Report: area 10 in section 7, and risk R1 in section 8.
Q13	Can AI coding tools break the security controls? No, not without being caught. 7 automatic guardrail checks block unsafe changes from Lovable and Cursor before they can go live.	Readiness Report: item 10.8 and risk R7. Controls Checklist: C-34.
Q14	What customer data goes to OpenAI? Only the manager's site walkthrough notes and the site size. Names and phone numbers are removed first, and OpenAI does not train on this data.	Readiness Report: items 3.11 and 12.7, and Appendix D. Subprocessor List.
Q15	What personal data is held, where, and for how long? Crewproof holds 27 kinds of personal data. For each one, the records say where it is stored, why, who can see it and how long it is kept.	Readiness Report: Appendix D and item 12.5. Data Model: what each table holds.
Q16	Which vendors process customer data? 9 vendors: Supabase, Vercel, Stripe, Resend, OpenAI, PostHog, Sentry, Fly.io and Cloudflare. The list gives the agreement in place with each one.	Subprocessor List.
Q17	Is there open-source licence risk? No unresolved risk. We checked the licences of 412 packages and found 2 flags. We replaced 1 library, and 1 build tool was approved in writing.	Licence Report. Readiness Report: item 10.11.
Q18	Are dependencies kept up to date? Yes. A tool (Renovate) proposes package updates every week, and security updates at once. At handover there were 0 high or critical known weaknesses.	Readiness Report: items 3.5 and 7.9. Controls Checklist: C-22.
Q19	How close is the company to a SOC 2 audit (an independent security audit that enterprise buyers often ask for)? Crewproof has a checklist of 48 controls, organized like a SOC 2 review, with an owner and evidence for each one. It is a starting point for an audit, not an audit report.	Controls Checklist. Readiness Report: risk R5 in section 8.
Q20	What changed in the code, and where is it? We made 67 reviewed changes (pull requests) and 31 database scripts (migrations). All of them are in Crewproof's own code repository.	Readiness Report: Appendix E.
Q21	Can we see the evidence behind a result? Yes. Each result has an evidence code (EV-) that names a folder of test output, settings and recordings. Crewproof can give you access.	Readiness Report: Appendix F, then the evidence/ folder.
Q22	What is not covered, and what risk remains? Eight risks remain. Examples are the dependence on one developer, no test above 400 users, and a penetration test of part of the app only.	Readiness Report: section 8 (risks R1 to R8).

PRODUCTION HARDENING SPRINT

Production Readiness Report

Crewproof, Inc.

APPLICATION	Crewproof web app and client portal (app.crewproof.com)
ENGAGEMENT	Production Hardening Sprint, 10 working days
PERIOD	Day 1 (kickoff) to Day 10 (handover)
ISSUED	Day 10, at the handover session
VERSION	1.0
PREPARED FOR	Maya Okafor, CEO, Crewproof, Inc.
SHARED WITH	The lead investor's technical advisor, with Crewproof's written permission
PREPARED BY	AxonBuild engineering team

123

scope items accounted for

122

verified with evidence

1

not applicable, reason recorded

0

open at handover

Confidential. This report describes security controls of a live application. Share it only with people who need it for the review.

ABOUT THIS DOCUMENT

We (AxonBuild) wrote this report after a 10-day Production Hardening Sprint for Crewproof, Inc., which makes an operations app for US commercial cleaning contractors. It is for Maya Okafor (CEO and co-founder, Crewproof) and the lead investor's technical advisor, and shows what we found, changed and proved. The glossary on the next page explains codes such as EV-1.4 and words such as "staging".

CONTENTS

What this report contains

	Glossary: the words and codes in this report, in plain English
1	Summary: what we found and what Crewproof has now
2	Scope, method and definitions: what we checked and how
3	Results by engineering area: the 123 items in 13 groups
4	Critical and high findings: the 36 most serious problems at the start
5	Checks that failed first: 4 tests that failed, and the retests that passed
6	Measured results: speed, recovery and security, before and after
7	Results for all 123 items: the full record, with evidence
8	Remaining risks and recommendations: what this sprint does not cover
9	Acceptance: the handover check and signatures
A	Appendix A: Role and permission matrix (who can do what in the app)
B	Appendix B: Targeted penetration test report (our attack test and its results)
C	Appendix C: Disaster recovery drill record (how we rebuilt the app from backups)
D	Appendix D: Personal-data inventory (what personal data the app holds)
E	Appendix E: Code and configuration changes (what we changed in the code)
F	Appendix F: Evidence samples (three pieces of proof, shown in full)

How to read this report

- **Founders and investors:** read section 1 (summary), section 6 (measured results) and section 8 (remaining risks).
- **Technical reviewers:** read sections 2 to 6. Then check any item in section 7 against its evidence.
- **Engineers who take over the code:** read section 7, Appendix E and the Operating Runbooks that come with this report.

GLOSSARY

Glossary: the words and codes in this report, in plain English

This page explains the codes and the technical words that this report uses most.

- **Numbers such as 1.4** are scope items (axonbuild.com/scope). Item 1.4 is the fourth item in area 01.
- **EV- codes, such as EV-1.4**, name a folder of proof, such as test output, in evidence/ in the handover repository.
- **R1 to R8** are the remaining risks (section 8). **PT-01 to PT-03** and **S1 to S5** are the findings and the parts of the penetration test (Appendix B). **G1 to G7** are the AI guardrail checks (Appendix F).
- **"Client"** means a contractor's own customer (an office, clinic or school) that uses the client portal.

WORD	PLAIN EXPLANATION
API	The way one program asks another for data or an action, without the app's screens.
API key, secret	A password that lets one system use another. It must never be visible in the app's code.
CI (continuous integration)	Automatic checks that run on every code change before it can go live.
Deploy, release	Putting a new version of the app live.
Edge Function	A small program that runs on Supabase's servers, not in the browser.
End-to-end test	A test that uses the app in a real browser, like a person.
Idempotent, repeat-safe	Running the same job twice gives the same result as running it once (no duplicate invoices).
Load test	A test that sends many users to the app at once to see how it copes. We used k6.
Migration	A script that changes the structure of the database.
p95	The time within which 95 of every 100 requests finish (410 ms = 0.41 seconds).
Penetration test	A test where we attack the app the way an attacker would, to find weak points.
Point-in-time restore	Bringing the database back to how it was at a chosen minute.
Production	The live app that customers use.
Pull request	A proposed code change that is reviewed before it is accepted.
Queue, dead-letter queue	A line of slow jobs. The dead-letter queue keeps jobs that failed every retry.
Rate limit	A cap on how many requests one user or address can make in a set time.
Rollback	Going back to the last version that worked.
Row-level security (RLS)	A database rule that lets each customer see only its own rows of data.
RPO, RTO	How much data could be lost in a disaster (RPO) and how long the app could be down (RTO).
Runbook	A step-by-step procedure for one job, such as restoring the database.
Service-role key	The Supabase key with full rights to the database. It must stay on the server.
Session, token	A session keeps a user signed in. A token is the signed proof of who the user is.
SOC 2	An independent security audit that many enterprise buyers ask software companies for.
Staging	A private copy of the app used for testing before changes go live.
TLS, HTTPS, HSTS	TLS and HTTPS encrypt the connection. HSTS tells browsers to use only that connection.
Two-factor authentication (2FA)	A second proof of identity at sign-in, for example a code from a phone app.
Webhook	A message one service sends to another when something happens, such as a payment.

Services: Supabase (database, sign-in, files), Vercel (hosting), Stripe (payments), OpenAI (AI model), Resend (email), GitHub (code), Better Stack (uptime, status page), PostHog (analytics, feature flags). Lovable and Cursor are AI coding tools.

SECTION 1

Summary: what we found and what Crewproof has now

Crewproof, Inc. asked us to make its app ready for the technical due diligence of its seed round. In 10 working days we worked through all 123 scope items. At handover, 122 items are verified with evidence, 1 item does not apply (the reason is recorded), and none is open.

What the app does. Each contractor company uses Crewproof to run its jobs, crews and invoices. Managers schedule jobs at client sites and assign crews. Crews fill in checklists on their phones and upload photos as proof. Clients (offices, clinics and schools) check the finished work and sign it off in a client portal. A quote assistant uses AI (OpenAI) to turn a manager's site notes into a draft quote.

Who was involved. Maya Okafor (CEO and co-founder, Crewproof) is not technical and built the first version with Lovable, an AI app builder. Luis Ferreira (Crewproof's part-time contract developer) has edited the code with Cursor, an AI coding tool, for 8 months. Crewproof has no CTO.

6 critical findings on Day 1. All are fixed and retested.	30 high findings on Day 1. All are fixed and retested.	4 checks failed the first time. We fixed the cause, and the retest passed.	48 min to make the app usable again after a full loss, in a timed recovery drill.
---	--	--	---

What we found

The app worked well for its users, but it did not keep each contractor company's records apart. We fixed the six critical findings first. Numbers in brackets, such as (1.4), are scope items with a full record in section 7.

- **Other companies' invoices were open.** A user of one contractor company could read the invoices of every other company, because three tables had no row-level security (1.4).
- **Any user could become an admin.** A user could change their own role to admin with one request to the database, without the app's screens (1.3).
- **Admin actions were open to everyone.** The admin page was hidden in the menu, but any signed-in user could open it and use it, for example to delete the company (1.5).
- **A paid API key was public.** The OpenAI key was inside the code that every browser downloads, so anyone could copy it and run up charges (2.1).
- **Job photos were public.** Anyone who guessed a link could open photos of client sites and crews (3.2).
- **Payments could be faked.** The app did not check that a payment message (webhook) really came from Stripe, so a fake message could mark an invoice as paid (5.1).

We also found other gaps. Uploaded files had no backup. All changes went straight to the live app, with no test copy. There were no automatic tests, no error tracking and no written procedures. Sections 4 and 7 list every finding.

What Crewproof has now

- Each company sees only its own data. Row-level security is on for all 24 database tables, and 312 automatic tests check this on every code change.
- Crewproof has three separate copies of the app: local, staging (for tests) and production (live). Every change passes automatic tests and a review before it can go live.
- The database can be brought back to any minute (point-in-time restore), and uploaded files have a nightly backup. A timed recovery drill proved that both work (Appendix C).
- Payment messages from Stripe are verified and processed only once (repeat-safe). We corrected the 7 duplicate invoices and the 3 plan mismatches that we found.
- Errors and outages now reach Crewproof's team. Error tracking, uptime checks and alerts go to a named channel, and customers see a public status page.
- Crewproof, Inc. owns all 13 provider accounts, such as hosting, database and payments. Each account needs two-factor authentication.
- We measured that the app works well with 400 users at the same time. This is about four times today's peak (section 6).
- Crewproof has written runbooks, a role matrix, a personal-data inventory and a controls checklist for the review.

For the technical reviewer

Every result in section 7 has an evidence reference (EV-). Section 5 keeps every failed first attempt visible, with its fix and retest. Read section 8, the risks that remain, with this summary.

SECTION 2

Scope, method and definitions: what we checked and how

This section says what the sprint covered, how we worked, and what the result and severity words in this report mean. It also lists the limits of what we could prove in 10 days.

What the sprint covered

The sprint covered one codebase and the services it is connected to. These are the Crewproof web app and the client portal. They also include Supabase (the database, sign-in, file storage and Edge Functions), the Stripe billing setup, the emails the app sends, the OpenAI connection, DNS and hosting.

The scope is a published list of 123 items in 13 engineering areas. Each item says what we deliver and how we verify it. New product features were not part of the scope. Some items needed a new screen, for example account deletion or the list of signed-in sessions. We built those screens as part of the item.

How we worked, in five steps

1. **Inspect.** On Days 1 and 2 we reviewed the code, the database, every provider setting and the main business workflows. We recorded the state of each item before we changed anything.
2. **Fix the worst problems first.** We fixed the critical findings first: data separation, roles, exposed keys and payments. Then we fixed the high findings, then the rest. Every change went through a pull request with a review and automatic checks (CI).
3. **Verify.** We ran each item's verification method from the published scope. We did most tests on staging, a private copy of the app with made-up test data. On the live app (production) we only ran checks that change nothing, such as the security headers, TLS and DNS settings. We also ran three steps on the live app because the scope requires them: a timed release (7.10), a feature-flag switch (7.12) and the replacement (rotation) of the keys found in the code history (2.2).
4. **Retest failures.** When a check failed, we fixed the cause and ran the same check again. The first result stays in the record.
5. **Hand over.** On Day 10 we reviewed the results with Crewproof and the investor's technical advisor. We delivered this report with its evidence.

What the result words mean

Each of the 123 items has one result. This table explains the five result words. A "control" is a safeguard, such as a check, a setting, a test or a written procedure.

RESULT	MEANING
IMPLEMENTED	The control was missing. We built it and verified it.
CORRECTED	The control existed but was incomplete or unsafe. We fixed it and verified it.
VERIFIED AS SOUND	The control already worked. We tested it and kept the evidence. We did not replace working code.
NOT APPLICABLE	The item does not apply to this product. We recorded the reason, and the change that would make it apply.
OPEN	An applicable item that is not yet verified. Handover cannot be accepted while any item is open. This report has no open items.

What the severity levels mean

Severity says how serious a problem was on Day 1, before any fix. This table explains the five levels, from the most serious to the least.

SEVERITY	MEANING ON DAY 1
CRITICAL	An outside person or another customer could reach customer data, money or account control, with little effort.
HIGH	A likely path to data loss, a long outage, wrong charges or loss of control of the system.
MEDIUM	A weakness that needs other conditions to cause harm, or that makes problems slower to find and fix.

SEVERITY	MEANING ON DAY 1
LOW	A gap in documentation, hygiene or operations with a small direct effect.
NONE	There was no gap on Day 1, or the item is a sprint output, such as this report.

Limits of this report

- We ran the load tests on staging, which uses the same service plans as production. We did not test above 400 users. Figures above that level are an estimate (item 9.6).
- The penetration test covered the five parts of the app with the highest risk, not the whole app (Appendix B).
- The controls checklist (item 12.6) is organized like a SOC 2 review. It is not a SOC 2 audit report.
- The results describe the app at handover. Later changes need the same checks. These checks now run automatically on every change (CI).

SECTION 3

Results by engineering area: the 123 items in 13 groups

All 123 items have a result. No applicable item is open. Every problem we found on Day 1 is now fixed and verified.

This table counts the items in each of the 13 areas of the scope, by result. Look first at the last two columns. "Critical or high at kickoff" shows where the most serious problems were on Day 1. "Open" is 0 in every area. A dot means none.

AREA		ITEMS	IMPLEMENTED	CORRECTED	VERIFIED AS SOUND	NOT APPLICABLE	CRITICAL OR HIGH AT KICKOFF	OPEN
01	Authentication & authorization	11	5	6	.	.	6	0
02	Secrets, keys & configuration	9	3	5	1	.	7	0
03	Input, output & application security	12	8	4	.	.	6	0
04	Database & data	14	8	5	1	.	4	0
05	Payments & billing	10	4	5	1	.	4	0
06	Error handling & reliability	10	9	1	.	.	1	0
07	Environments, CI/CD & deployment	13	10	3	.	.	4	0
08	Logging, monitoring & alerting	8	6	2	.	.	1	0

AREA		ITEMS	IMPLEMENTED	CORRECTED	VERIFIED AS SOUND	NOT APPLICABLE	CRITICAL OR HIGH AT KICKOFF	OPEN
09	Performance	7	5	2	.	.	1	0
10	Code health & AI development guardrails	11	9	2	.	.	2	0
11	Email & communications	3	1	1	1	.	.	0
12	Privacy & compliance foundations	8	5	2	.	1	.	0
13	Handover & included support	7	7	.	.	.	.	0
Total		123	80	38	4	1	36	0

The two small tables below count all 123 items. The left table counts them by result. The right table counts them by severity on Day 1.

Results at handover

RESULT	ITEMS
IMPLEMENTED	80
CORRECTED	38
VERIFIED AS SOUND	4
NOT APPLICABLE	1

Severity on Day 1

SEVERITY	ITEMS
CRITICAL	6
HIGH	30
MEDIUM	51
LOW	26
NONE	10

Severity describes the problem found on Day 1, not the state at handover. Every item with a problem is now implemented or corrected, and verified.

SECTION 4

Critical and high findings: the 36 most serious problems at the start

On Day 1 we found 6 critical and 30 high problems. All of them are now fixed and verified.

This table lists these problems in scope order. Each row gives the item number, the item name from the scope, and what we found on Day 1. The last two columns give the severity on Day 1 and the result at handover. Read the critical rows first. Section 7 gives the fix and the evidence for each item.

ID	ITEM AND WHAT WE FOUND	SEVERITY	RESULT
1.3	Server-side authorization A user could set their own role to admin with one direct API call (a request to the server without the app's screens). The permission checks were only in the React UI (the screens in the browser), and each user could change their own <code>profiles.role</code> column. 7 of the 10 Edge Functions (small programs that run on the server in Supabase) did not check the caller's role.	CRITICAL	CORRECTED
1.4	Data isolation across every table Any user could read other companies' invoices through the Supabase REST API (the database's direct web interface). Row-level security (a database rule that lets each company see only its own rows) was off on <code>invoices</code> , <code>job_photos</code> and <code>quote_drafts</code> . The rules on 6 other tables checked only that a user was signed in, not the user's company.	CRITICAL	CORRECTED
1.5	Protected administrative actions The <code>/admin</code> page was only hidden in the menu. "Delete company" and "Change plan" ran from the browser with the user's own key.	CRITICAL	CORRECTED
1.6	Authentication abuse protection The default limits of Supabase (the database and sign-in service the app runs on) were on. But the custom "magic link for crews" function (a sign-in link sent by email) had no limit, so a script could send unlimited sign-in emails to any address.	HIGH	CORRECTED
1.8	Complete account deletion Crewproof's support team deleted users by hand in the Supabase dashboard. This left photos, profiles and Stripe customers (the customer records at Stripe, the payment service) behind.	HIGH	IMPLEMENTED
1.11	Two-factor authentication for owner and admin accounts The app had no two-factor authentication (2FA: a second proof of identity at sign-in, such as a code from a phone app). 5 of the 9 provider dashboards (the admin websites of the outside services) had 2FA off, including Stripe and Supabase.	HIGH	IMPLEMENTED
2.1	Frontend secret removal The key shipped in the browser bundle (the code that every visitor's browser downloads) as <code>VITE_OPENAI_KEY</code> , so it was visible in the page source.	CRITICAL	CORRECTED
2.2	Repository history scan and rotation In the first month, a settings file (<code>.env</code>) was committed to the Git history (the record of every past version of the code). It held the Stripe live secret key, which controls payments, and the Supabase service-role key, which can read and change all data. Anyone with a copy of the code could use both.	HIGH	CORRECTED
2.4	Environment configuration separation One set of settings served every copy of the app. Local development, on the developer's own computer, pointed at the production database (the live database that customers use).	HIGH	IMPLEMENTED
2.5	Protected third-party API usage The browser called OpenAI with no limit (see item 2.1). The server called Resend, the email service, with no cap on bulk sends.	HIGH	CORRECTED

ID	ITEM AND WHAT WE FOUND	SEVERITY	RESULT
2.7	Metered-service spend alerts OpenAI, Supabase, Vercel and Resend charge by use and had no spend alerts.	HIGH	IMPLEMENTED
2.8	Provider account ownership The OpenAI, Resend and Vercel accounts used the personal email of Luis Ferreira (Crewproof's contract developer). The GitHub repository (where the code is stored) sat in a personal account linked to Lovable.	HIGH	CORRECTED
2.9	Managed-platform security settings On Supabase, email confirmation was off, the storage bucket (the folder for uploaded files) was public, and point-in-time recovery was off. On Vercel, preview deployments (test versions of the app) were public. On Stripe, there were no restricted keys (keys limited to the few actions a service needs).	HIGH	CORRECTED
3.1	Server-side request validation Edge Functions trusted the request body without checks.	HIGH	IMPLEMENTED
3.2	Secure file uploads The job-photos bucket was public, with guessable paths. Uploads had no type or size check.	CRITICAL	CORRECTED
3.5	Dependency remediation The packages had 37 known vulnerabilities (4 high, 0 critical). The app also included 18 unused packages.	HIGH	CORRECTED
3.6	Public-endpoint rate limits There were no limits on signup, the contact form, the quote assistant or CSV export. This costs money and can slow the app for everyone.	HIGH	IMPLEMENTED
3.7	OWASP Top 10 review No one had done a security review of the app.	HIGH	IMPLEMENTED
3.10	Targeted external penetration testing No one had ever tested the app from the outside.	HIGH	IMPLEMENTED
4.4	Atomic multi-step writes "Complete job and create invoice" ran as 4 separate writes, and a failure in the middle stopped the rest.	HIGH	CORRECTED
4.6	Version-controlled migrations All changes to the database structure were made in the Supabase dashboard, and the repository held no migrations.	HIGH	IMPLEMENTED
4.7	Verified backups and restore Daily database backups were on, kept for 7 days. The stored files (70,000 photos) were in no backup.	HIGH	CORRECTED
4.12	Disaster recovery drill No one had ever done a disaster recovery drill (a full practice rebuild of the app from its backups).	HIGH	IMPLEMENTED

ID	ITEM AND WHAT WE FOUND	SEVERITY	RESULT
5.1	Webhook signature verification The Stripe webhook (the message Stripe sends to the app when something happens, such as a payment) did not check signatures.	CRITICAL	CORRECTED
5.2	Idempotent webhook handling Repeated deliveries of the same Stripe message created duplicate records. We found 7 duplicate invoices in production.	HIGH	CORRECTED
5.3	Complete billing event lifecycle The app reacted to only 2 Stripe events: a finished checkout (checkout.session.completed) and a paid invoice (invoice.paid).	HIGH	CORRECTED
5.4	Server-side entitlements The plan limits (sites, users, quote assistant) were checked only in the screens, so a Starter company could call the API directly and use Growth features.	HIGH	CORRECTED
6.3	Error tracking There was no error tracking.	HIGH	IMPLEMENTED
7.1	Three isolated environments Production was the only environment, so changes from Lovable and Cursor (an AI code editor) had no place to be tested first.	HIGH	IMPLEMENTED
7.2	Protected production branch Anyone could push code to main, the main branch (version line) of the code. Lovable pushed to it directly.	HIGH	IMPLEMENTED
7.3	Pull-request CI There was no CI, so a broken change could reach customers with no warning.	HIGH	IMPLEMENTED
7.11	Independent repository and hosting Production deployed from the personal Vercel account of Luis Ferreira (contract developer). The repository was in a personal GitHub account linked to Lovable.	HIGH	CORRECTED
8.2	Uptime monitoring Crewproof had no uptime monitoring and no outage alert.	HIGH	IMPLEMENTED
9.2	Load test and bottleneck fixes No one had ever run a load test (a test that sends many users to the app at once).	HIGH	IMPLEMENTED
10.5	Critical-path smoke tests The app had no automated tests.	HIGH	IMPLEMENTED
10.7	Auth and billing unit tests There were no unit tests (small tests of one piece of code) for roles, plan limits, webhooks or invoice totals.	HIGH	IMPLEMENTED

SECTION 5

Checks that failed first, and the retests that passed

4 checks failed the first time we ran them. We fixed the cause and ran the same check again. Each retest passed.

We keep the first result in the record, because a reviewer needs to see what each check caught. This table gives the item, the first result, the fix and the retest. The four checks are the photo upload test, the penetration test, the recovery drill and the load test.

ID	ITEM	FIRST RESULT, FIX AND RETEST
3.2	Secure file uploads	The first test on Day 6 failed. The app accepted an HTML file renamed to .jpg and served it as a web page (HTML content type). The app now checks the content of the file, not its name. The retest passed on Day 7.
3.10	Targeted external penetration testing	On Day 5 we found the sign-off problem: a client could open another site's sign-off page by changing the ID in the link. We fixed it with signed, expiring tokens. The retest passed on Day 6.
4.12	Disaster recovery drill	The first drill on Day 6 failed. The database came back, but the photos did not, because the stored files were in no backup. We added the storage copy (see item 4.7). The second drill passed on Day 8.
9.2	Load test and bottleneck fixes	The first run on Day 4 failed already at 150 users (p95 3.9 s). The target was p95 under 800 ms at 300 users. After the fixes, the Day 7 run met the target and passed up to 400 users.

SECTION 6

Measured results: speed, recovery and security, before and after

The app is faster, it can be rebuilt from backups, and its serious security gaps are closed. Under heavy use, the job board now answers within 0.41 seconds instead of 3.9 seconds, with more than twice as many users.

This table compares the app on Day 1 with the app at handover. For times, sizes and error counts, lower is better; for scores and grades, higher is better. The last column gives the scope item that holds the evidence.

WHAT WE MEASURED	DAY 1	AT HANDOVER	ITEM
SPEED AND CAPACITY			
How long the job board takes to answer when many crews use the app at once. 95 of every 100 requests finish within this time (p95 under load).	3.9 s with 150 users	410 ms (0.41 s) with 400 users	9.2
Share of requests that fail under load (error rate)	2.1%	0.02%	9.2

WHAT WE MEASURED	DAY 1	AT HANDOVER	ITEM
Number of people who can use the app at the same time (measured concurrent capacity)	Not known	400 users (about 4 times today's peak)	9.6
Time for the database to find the jobs for the job board (job board query)	1,840 ms	38 ms	4.2
Database requests needed to show one job board (queries per view)	151	4	4.3
Code a browser downloads to open the app (JavaScript on first load)	2.4 MB	690 KB	9.3
Data downloaded to show a job board with 40 photos	190 MB	3.1 MB	9.4
Speed score of the job board on a phone, from 0 to 100 (Lighthouse mobile performance)	41	91	9.5
RECOVERY AND OPERATIONS			
Time to make the app usable again after the loss of the production database project (recovery time, RTO)	Never tested	48 minutes, in a drill	4.12
Most database changes that could be lost (data loss window, RPO)	Up to 24 hours (daily backup)	About 2 minutes (point-in-time restore)	4.7
Uploaded files that have a backup	None	All, copied every night	4.7
Time to put an approved change live (merge to production)	No release process (no pipeline)	6 min 40 s, with tests and an approval step	7.10
Time from an outage to an alert to Crewproof's team	The team learned from customer emails.	About 2 minutes	8.2
SECURITY AND CODE			
Automatic tests	0	14 end-to-end, 86 unit and 312 data-separation tests	10.5, 10.7, 1.4
Private keys that anyone could copy, from the browser code or the code history (Git history)	1 in the browser, 4 in the history	0. We replaced (rotated) all 5.	2.1, 2.2
Known high or critical weaknesses in outside code packages (dependency vulnerabilities)	4	0	3.5
Provider accounts owned by Crewproof, Inc.	5 of 9	13 of 13	2.8
Provider accounts with two-factor authentication	4 of 9	13 of 13	1.11
Grade for the security of the connection between a browser and the app (TLS scan)	B. Browsers were not told to use only HTTPS (no HSTS).	A+	7.13
Email settings that stop others from sending email in Crewproof's name (SPF, DKIM, DMARC)	1 of 3 (SPF only)	All 3 (SPF, DKIM, DMARC)	11.1
Errors found by the strictest setting of the code checker (TypeScript strict mode)	612, with the setting off	0, with the setting on	10.6

How we ran the load tests: we used k6, a load test tool, for 30 minutes on staging. Staging runs on the same Supabase and Vercel plans as production. The test copies the morning crew start: crews sign in, open the job board, fill in a checklist and upload a photo.

SECTION 7

Results for all 123 items

ABOUT THIS SECTION

This section gives the result for each of the 123 scope items, in the order of the 13 areas. Each entry starts with the short answer **In plain words**, then gives the exact technical detail for an engineer. Start with the 6 critical items listed below.

How to read each entry

Risk at start shows how serious the gap was on Day 1 (kickoff), before any fix: Critical, High, Medium, Low, or None (no gap, or a sprint output such as this report). It is not the risk today. **Result** shows what we did, in one of four words. No item is open. Section 2 defines each word in full.

- IMPLEMENTED** The control was missing. We built it and verified it.
- CORRECTED** The control was incomplete or unsafe. We fixed it and verified it.
- VERIFIED AS SOUND** The control already worked. We tested it and kept the evidence.
- NOT APPLICABLE** The item does not apply today. We recorded the reason.

Below the plain line, **What we found** is the state on Day 1, **What we did** is the sprint work, and **How we checked** is the test and its result. An amber **Failed first, then** box shows a check that failed first, the fix and the retest (4 items: 3.2, 3.10, 4.12 and 9.2). A code such as [EV-1.4] names the folder of proof for item 1.4 in the evidence/ folder of Crewproof's code repository. Appendix F shows three samples.

Read these 6 items first

The 6 critical items exposed customer data, money or control of the app on Day 1: 1.3 (any user could become an admin), 1.4 (other companies' invoices were open), 1.5 (admin actions were open), 2.1 (the OpenAI key was public), 3.2 (job photos were public) and 5.1 (payments could be faked). All are fixed and tested. Then read the 30 items marked "Risk at start: High", which section 4 also lists.

01. Authentication & authorization

This area controls who can sign in and what each person may see and do. The most important result: one contractor company can no longer see another company's data, and no user can make themselves an admin.

11 items: 5 implemented, 6 corrected.

1.1 Complete authentication flow review

RISK AT START: MEDIUM

RESULT: CORRECTED

IN PLAIN WORDS Sign-in links could send users to a fake website, and anyone could sign up with an email they did not own. Both are fixed.

WHAT WE FOUND	Email confirmation was off. The password-reset redirect allowlist (the websites that the app may send a user to after a reset) had a wildcard, <code>https://*</code> , left from Lovable previews. Lovable is the AI app builder that made the first version of the app. The Google sign-in callback accepted any <code>next</code> parameter, which is an open redirect.
WHAT WE DID	We turned on email confirmation. We replaced the wildcard with the 3 real web addresses (origins) of the app. We restricted <code>next</code> to relative paths inside the app.
HOW WE CHECKED	31 flow tests covered signup, login, logout, password reset, email verification and the Google callback. They tested the success, rejected, expired and reused-token paths, and all behaved as expected. [EV-1.1]

1.2 Session and token controls

RISK AT START: LOW

RESULT: CORRECTED

IN PLAIN WORDS	After a password change, other devices stayed signed in, so a person with the old password kept access. Now they are signed out.
WHAT WE FOUND	Most token settings were correct. The access token (the short-lived pass that proves who the user is) expired after 1 hour. The refresh token (which gets a new pass) was replaced at each use (rotation).
WHAT WE DID	A password change now revokes (cancels) all other sessions. We confirmed that refresh-token reuse detection is on, so the app refuses a refresh token used a second time.
HOW WE CHECKED	A session opened before a password change is refused on its next request. An expired token cannot reach any protected table or function. [EV-1.2]

1.3 Server-side authorization

RISK AT START: CRITICAL

RESULT: CORRECTED

IN PLAIN WORDS	Any user, even a crew member, could make themselves an admin with one request. Now the server checks the role before every action.
WHAT WE FOUND	A user could set their own role to admin with one direct API call (a request to the server without the app's screens). The permission checks were only in the React UI (the screens in the browser), and each user could change their own <code>profiles.role</code> column. 7 of the 10 Edge Functions (small programs that run on the server in Supabase) did not check the caller's role.
WHAT WE DID	We moved the role to the <code>memberships</code> table. Only the <code>change-role</code> server function can write it, and it checks the caller's role first. Owners can change any role; admins only roles below admin. One shared role guard now protects every function that takes a user session: 14 functions, including the 4 added in the sprint.
HOW WE CHECKED	We called all 14 functions and all write endpoints directly, as anonymous, crew, client and manager users. The server refused every unauthorized call (403, "forbidden"), including the attempt to self-promote to admin. [EV-1.3]

1.4 Data isolation across every table

RISK AT START: CRITICAL

RESULT: CORRECTED

IN PLAIN WORDS	A user at one cleaning contractor could read every other contractor's invoices. Now a database rule on all 24 tables keeps data apart.
WHAT WE FOUND	Any user could read other companies' invoices through the Supabase REST API (the database's direct web interface). Row-level security (a database rule that lets each company see only its own rows) was off on <code>invoices</code> , <code>job_photos</code> and <code>quote_drafts</code> . The rules on 6 other tables checked only that a user was signed in, not the user's company.
WHAT WE DID	We turned on row-level security on all 24 tables. 18 tables give access only to the user's own company. <code>profiles</code> gives each user only their own row, and <code>companies</code> only its members. 2 tables are for the server only. 2 tables, <code>plans</code> and <code>public_status</code> , are public on purpose. The rules are now in migrations (scripts in the code that change the database structure), so every change is recorded.

HOW WE CHECKED Our tenant-isolation suite (tests that try to reach other companies' data) has 312 read and write cases. They run as anonymous users, as all 5 roles and across 2 separate companies, with 0 failures. The suite runs in CI (automatic checks) on every pull request (a proposed code change, reviewed before it is accepted). [EV-1.4]

1.5 Protected administrative actions

RISK AT START: CRITICAL
RESULT: CORRECTED

IN PLAIN WORDS Any signed-in user could open the admin page and delete a whole company or change its plan. Now only owners and admins can.

WHAT WE FOUND The /admin page was only hidden in the menu. "Delete company" and "Change plan" ran from the browser with the user's own key.

WHAT WE DID We moved the admin actions to server functions that require the owner or admin role. The admin page now checks the role on the server before it loads any data.

HOW WE CHECKED We ran each admin action as an owner, and the server allowed it. We ran each one as a manager, a crew user and a client user, and the server refused it. [EV-1.5]

1.6 Authentication abuse protection

RISK AT START: HIGH
RESULT: CORRECTED

IN PLAIN WORDS A script could flood any inbox with sign-in emails in Crewproof's name. Now every sign-in path limits attempts, and normal users are not locked out.

WHAT WE FOUND The default limits of Supabase (the database and sign-in service the app runs on) were on. But the custom "magic link for crews" function (a sign-in link sent by email) had no limit, so a script could send unlimited sign-in emails to any address.

WHAT WE DID We added rate limits (caps on attempts in a set time) to login, signup, password reset and magic link: 5 attempts per 15 minutes per account and 20 per hour per IP address. A user who reaches a limit sees a clear message and a reset path.

HOW WE CHECKED We sent a burst of 200 attempts. The limits applied after the threshold, and no emails went out past the limit. A normal user could sign in again after the time window. [EV-1.6]

1.7 Documented role model

RISK AT START: MEDIUM
RESULT: IMPLEMENTED

IN PLAIN WORDS No one had written down what each type of user may do. Now a one-page table sets it out, and it matches the tested behavior.

WHAT WE FOUND There was no written role model, and people at Crewproof described the roles in different ways.

WHAT WE DID We wrote a one-page matrix of the 5 roles (owner, admin, manager, crew, client) against 38 actions.

HOW WE CHECKED We compared every cell of the matrix with the test results of items 1.3 and 1.4 and found 0 mismatches. [EV-1.7]

1.8 Complete account deletion

RISK AT START: HIGH
RESULT: IMPLEMENTED

IN PLAIN WORDS Users could not delete their accounts, and manual deletions left data behind. Now deletion is self-service and complete, except invoices kept for tax.

WHAT WE FOUND Crewproof's support team deleted users by hand in the Supabase dashboard. This left photos, profiles and Stripe customers (the customer records at Stripe, the payment service) behind.

WHAT WE DID We built self-service deletion for users and owner-led deletion for whole companies. Deletion covers 11 related tables, the stored files and the Stripe customer. Invoices are kept for 7 years, the documented tax exception.

HOW WE CHECKED We deleted a seeded test account. Its related records and files were gone, and the kept invoices were recorded with the reason. [EV-1.8]

1.9 Session management and global sign-out**RISK AT START: MEDIUM****RESULT: IMPLEMENTED**

IN PLAIN WORDS	Crews share phones at some sites, so a user could stay signed in on another person's phone. Now users can sign out everywhere.
WHAT WE FOUND	Users could not see where they were signed in.
WHAT WE DID	We added a "Your devices" page. It lists the user's sessions and has a "Sign out everywhere" button.
HOW WE CHECKED	We created 3 sessions and revoked all of them. Protected access stopped on each one. [EV-1.9]

1.10 Sensitive-action audit log**RISK AT START: MEDIUM****RESULT: IMPLEMENTED**

IN PLAIN WORDS	Nobody could see who changed a role, deleted data or changed billing. Now a log records each action, and no one can edit it.
WHAT WE FOUND	The app kept no record of role changes, deletions or billing changes.
WHAT WE DID	We added an audit_log table. Database triggers (rules that run on each change) and server functions write to it. Each entry records the actor, the action, the target and the time. Only owners can read it. It is append-only: entries can be added but never changed or removed.
HOW WE CHECKED	We triggered the 9 listed actions and got 9 correct entries. A crew user could not read the log. An admin could not edit or delete entries. [EV-1.10]

1.11 Two-factor authentication for owner and admin accounts**RISK AT START: HIGH****RESULT: IMPLEMENTED**

IN PLAIN WORDS	A stolen password alone could open an owner account, or Crewproof's Stripe and Supabase accounts. Now these need a second proof of identity.
WHAT WE FOUND	The app had no two-factor authentication (2FA: a second proof of identity at sign-in, such as a code from a phone app). 5 of the 9 provider dashboards (the admin websites of the outside services) had 2FA off, including Stripe and Supabase.
WHAT WE DID	We made authenticator-app 2FA required for the owner and admin roles in the app. We turned on 2FA for all 13 provider accounts: the 9 existing ones and the 4 added in the sprint. The recovery codes are in the password manager of Maya Okafor (CEO and co-founder of Crewproof).
HOW WE CHECKED	For every owner and admin account in the app, and on all 13 dashboards, sign-in without the second factor was refused. [EV-1.11]

02. Secrets, keys & configuration

This area protects the passwords and keys that let the app use outside services, such as Stripe for payments and OpenAI for the quote assistant. The most important result: no private key is visible in the app now, and every leaked key has been replaced.

9 items: 3 implemented, 5 corrected, 1 verified as sound.

2.1 Frontend secret removal**RISK AT START: CRITICAL****RESULT: CORRECTED**

IN PLAIN WORDS	Anyone could copy Crewproof's OpenAI key from the web page and run up charges. Now a new key stays on the server only.
WHAT WE FOUND	The key shipped in the browser bundle (the code that every visitor's browser downloads) as VITE_OPENAI_KEY, so it was visible in the page source.

WHAT WE DID	We moved the OpenAI calls to an Edge Function on the server and rotated the key (replaced it with a new one). Only two keys remain in the browser code: the Supabase anon key and the Stripe publishable key. Both are designed to be public.
HOW WE CHECKED	We scanned the source code, the built files and recorded browser traffic. We found no private credentials. [EV-2.1]

2.2 Repository history scan and rotation

RISK AT START: HIGH

RESULT: CORRECTED

IN PLAIN WORDS	Old versions of the code held the master keys for payments and the database. Those keys and 2 more are now replaced.
WHAT WE FOUND	In the first month, a settings file (.env) was committed to the Git history (the record of every past version of the code). It held the Stripe live secret key, which controls payments, and the Supabase service-role key, which can read and change all data. Anyone with a copy of the code could use both.
WHAT WE DID	We rotated both keys. We also rotated 2 other keys found in the history: the Resend key (email) and the Google OAuth secret (Google sign-in). A secret scan in CI now stops any change that adds a key to the code.
HOW WE CHECKED	We kept the Gitleaks report (a tool that searches the code history for keys). All 4 old keys are refused by their providers. The new keys work in production. [EV-2.2]

2.3 Server-only administrative keys

RISK AT START: NONE

RESULT: VERIFIED AS SOUND

IN PLAIN WORDS	The database master key must never reach a user's browser. We checked: the current key already stays on the server only.
WHAT WE FOUND	The current service-role key was used only inside Edge Functions on the server. (Item 2.2 covers the old key in the Git history.)
WHAT WE DID	No change was needed.
HOW WE CHECKED	We checked the runtime configuration, the built files and all 6 code paths that use the key. [EV-2.3]

2.4 Environment configuration separation

RISK AT START: HIGH

RESULT: IMPLEMENTED

IN PLAIN WORDS	A developer's laptop test could change real customer data, because every copy of the app used the live database. Now each copy has its own.
WHAT WE FOUND	One set of settings served every copy of the app. Local development, on the developer's own computer, pointed at the production database (the live database that customers use).
WHAT WE DID	We separated the variables, databases and keys for local, staging and production. Staging is a private copy of the app used for testing before changes go live. A configuration inventory lists 23 variables, each with its owner and environment.
HOW WE CHECKED	A signup, a payment and an email on staging touched only staging services. [EV-2.4]

2.5 Protected third-party API usage

RISK AT START: HIGH

RESULT: CORRECTED

IN PLAIN WORDS	Nothing capped use of Crewproof's paid OpenAI and email services, so Crewproof could get a large bill. Now daily caps apply.
WHAT WE FOUND	The browser called OpenAI with no limit (see item 2.1). The server called Resend, the email service, with no cap on bulk sends.
WHAT WE DID	All private keys are now on the server only. The quote assistant has a cap of 30 requests per user per day and 300 per company per day. Email has a cap of 2,000 per company per day.

HOW WE CHECKED Unauthorized calls were refused. In tests, we reached each limit, and the app refused the next request with a clear message. [EV-2.5]

2.6 Secret rotation runbook

RISK AT START: MEDIUM
RESULT: IMPLEMENTED

IN PLAIN WORDS If a key leaked, no one at Crewproof knew how to replace it safely. Now a rehearsed, written procedure covers all 9 keys.

WHAT WE FOUND There was no written process. The rotation in item 2.2 was the first time anyone had rotated a key.

WHAT WE DID We wrote a runbook (a step-by-step procedure). For each key, it says where the key is used, how to rotate it, how to check the result and how to roll back.

HOW WE CHECKED We rehearsed a rotation with a test Resend key in staging and recorded the affected services and the checks. [EV-2.6]

2.7 Metered-service spend alerts

RISK AT START: HIGH
RESULT: IMPLEMENTED

IN PLAIN WORDS A bug or an attacker could run up large usage bills unseen until the invoice came. Now spend alerts go to Maya and to Slack.

WHAT WE FOUND OpenAI, Supabase, Vercel and Resend charge by use and had no spend alerts.

WHAT WE DID We added alerts on all 5 metered services: the 4 above plus the Fly.io worker (the server that runs background jobs). The alerts go to Maya and to the operations Slack channel (#ops-alerts). OpenAI alerts at \$150 a month. The others alert at 150% of their usual monthly bill.

HOW WE CHECKED We triggered test thresholds, and Maya received the alerts. [EV-2.7]

2.8 Provider account ownership

RISK AT START: HIGH
RESULT: CORRECTED

IN PLAIN WORDS Three key services and the code sat in personal accounts, so Crewproof did not fully control its app. Now Crewproof, Inc. owns all 13 accounts.

WHAT WE FOUND The OpenAI, Resend and Vercel accounts used the personal email of Luis Ferreira (Crewproof's contract developer). The GitHub repository (where the code is stored) sat in a personal account linked to Lovable.

WHAT WE DID We moved all 9 existing services to accounts owned by Crewproof, Inc. Maya is the owner of each one, and the billing is in Crewproof's name. Luis now has member access. We created the 4 services added in the sprint under Crewproof from the start.

HOW WE CHECKED The provider inventory lists the owner, the role holder and the transfer date for each of the 13 services. [EV-2.8]

2.9 Managed-platform security settings

RISK AT START: HIGH
RESULT: CORRECTED

IN PLAIN WORDS Risky default settings left files and test versions of the app open, and the database could not return to an earlier minute. Both are fixed.

WHAT WE FOUND On Supabase, email confirmation was off, the storage bucket (the folder for uploaded files) was public, and point-in-time recovery was off. On Vercel, preview deployments (test versions of the app) were public. On Stripe, there were no restricted keys (keys limited to the few actions a service needs).

WHAT WE DID We changed 14 settings across Supabase, Vercel, Stripe and GitHub. We turned on Supabase point-in-time recovery, which can bring the database back to how it was at a chosen minute. Maya approved its cost.

HOW WE CHECKED A before-and-after table lists the 14 settings and names each risky default. [EV-2.9]

03. Input, output & application security

This area protects the app from harmful data and from attacks through its pages, uploads and AI feature. The most important result: job photos are private now, and the 3 problems that our attack test found are fixed and retested.

12 items: 8 implemented, 4 corrected.

3.1 Server-side request validation

RISK AT START: HIGH

RESULT: IMPLEMENTED

IN PLAIN WORDS	The server accepted any data, so a user could save an invoice with a negative amount. Now the server and the database refuse invalid values.
WHAT WE FOUND	Edge Functions trusted the request body without checks.
WHAT WE DID	We added Zod schemas (rules that describe valid input) to all 14 Edge Functions. We added database check constraints (rules that refuse invalid values) to 19 columns.
HOW WE CHECKED	We sent 140 invalid and hostile payloads, such as wrong types, negative amounts, and SQL and script strings (text written to attack the database or the page). The server refused all of them safely. [EV-3.1]

3.2 Secure file uploads

RISK AT START: CRITICAL

RESULT: CORRECTED

IN PLAIN WORDS	Anyone who guessed a link could see job photos of clients' buildings and crew faces. Now photos are private, and links expire after 15 minutes.
WHAT WE FOUND	The job-photos bucket was public, with guessable paths. Uploads had no type or size check.
WHAT WE DID	We made the bucket private. Photos now open only through signed URLs (links that expire). Uploads accept images only (JPEG, PNG, HEIC, WebP), up to 15 MB. Storage rules limit each company to its own files.
HOW WE CHECKED	The server refused an unauthorized download, a bucket listing, an oversized file and the disallowed file types. The old public URLs now return 403 (forbidden). [EV-3.2]
FAILED FIRST, THEN	The first test on Day 6 failed. The app accepted an HTML file renamed to .jpg and served it as a web page (HTML content type). The app now checks the content of the file, not its name. The retest passed on Day 7.

3.3 Restricted cross-origin access

RISK AT START: MEDIUM

RESULT: CORRECTED

IN PLAIN WORDS	Any other website could call Crewproof's server functions from a visitor's browser. Now only Crewproof's own app, staging and preview addresses can.
WHAT WE FOUND	All Edge Functions returned Access-Control-Allow-Origin: *, a CORS setting (the rule for which websites may call the server from a browser) that allows every website.
WHAT WE DID	We now allow only the app, staging and preview origins (web addresses). Calls that change data must carry the user's token in the request header, not in cookies.
HOW WE CHECKED	The approved origins work, and the server refused the other origins. No other website could make a write with the user's credentials. [EV-3.3]

3.4 Production security headers

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	Browsers got no instructions to block common attacks, such as a fake site that shows Crewproof inside its page. Now every page sends them.
WHAT WE FOUND	The app sent no security headers (instructions sent with each page that tell the browser what to block).

WHAT WE DID	We added 6 security headers. They tell the browser to run scripts only from approved sources and to always use an encrypted connection, and they stop other websites from showing the app inside their own pages. The headers are CSP, HSTS (with preload), X-Frame-Options DENY, Referrer-Policy, Permissions-Policy and X-Content-Type-Options.
HOW WE CHECKED	We checked the headers on all routes. We walked through the full app with the CSP in place, and no flow broke. An external header scan gave grade A. [EV-3.4]

3.5 Dependency remediation

RISK AT START: HIGH

RESULT: CORRECTED

IN PLAIN WORDS	The app used outside code packages with known security holes that attackers can look up. Now no high or critical holes remain.
WHAT WE FOUND	The packages had 37 known vulnerabilities (4 high, 0 critical). The app also included 18 unused packages.
WHAT WE DID	We upgraded or replaced the affected packages and removed the 18 unused ones.
HOW WE CHECKED	The final scan shows 0 high or critical vulnerabilities. 3 low ones remain, with no path that the app can reach, and we recorded them. The regression suite (tests that check that existing features still work) passed after the changes. [EV-3.5]

3.6 Public-endpoint rate limits

RISK AT START: HIGH

RESULT: IMPLEMENTED

IN PLAIN WORDS	A script could repeat costly requests, such as signup or the quote assistant, without limit. Now limits stop abuse but not normal use.
WHAT WE FOUND	There were no limits on signup, the contact form, the quote assistant or CSV export. This costs money and can slow the app for everyone.
WHAT WE DID	We added limits on 7 public or costly endpoints (addresses that the app's server answers). We sized each limit from 30 days of real traffic.
HOW WE CHECKED	Burst tests showed that the limits applied. Normal use at 3x the busiest real hour was not blocked. [EV-3.6]

3.7 OWASP Top 10 review

RISK AT START: HIGH

RESULT: IMPLEMENTED

IN PLAIN WORDS	No one had checked the app against the most common kinds of web attack. We checked all 10, and the 21 problems found are fixed.
WHAT WE FOUND	No one had done a security review of the app.
WHAT WE DID	We reviewed the app against all 10 categories of the OWASP Top 10, the standard list of the most common security weaknesses in web apps. Each of the 21 problems is fixed in an item above.
HOW WE CHECKED	We recorded the result for each category, with links to the fix and the test for each problem. 1 category had no finding: A10, server-side request forgery (an attacker makes the server fetch a web address). The app never fetches a web address that a user supplies, and we recorded this reason. [EV-3.7]

3.8 Bot protection

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	Scripts filled the signup and contact forms with junk. Now a bot check stops the scripts, and real users pass.
WHAT WE FOUND	The public forms had no bot check. There were 140 junk trial signups in the last 90 days.
WHAT WE DID	We added Cloudflare Turnstile (a check that tells real people from automated scripts) to signup, contact and password reset.
HOW WE CHECKED	Real users pass. The app refuses a missing or failed check and shows a retry message. [EV-3.8]

3.9 Security disclosure contact**RISK AT START: LOW****RESULT: IMPLEMENTED**

IN PLAIN WORDS	A person who found a security hole had no way to tell Crewproof. Now a published address sends reports to Maya and Luis.
WHAT WE FOUND	There was no way for an outside person to report a vulnerability.
WHAT WE DID	We published /.well-known/security.txt, the standard file that tells security researchers where to report. Reports go to security@crewproof.com.
HOW WE CHECKED	The file is published. Maya and Luis both received a test report. [EV-3.9]

3.10 Targeted external penetration testing**RISK AT START: HIGH****RESULT: IMPLEMENTED**

IN PLAIN WORDS	No one had attacked the app the way a real attacker would. Our test found 3 problems, and all 3 are fixed and retested.
WHAT WE FOUND	No one had ever tested the app from the outside.
WHAT WE DID	We ran an authorized penetration test of 5 surfaces: authentication, client sign-off links, file upload and storage, the Stripe webhook and the quote assistant. It found 3 problems. Sign-off links were guessable (High). Uploads could store HTML (Medium, see item 3.2). The quote assistant revealed its system prompt, its hidden instructions (Low).
HOW WE CHECKED	The test record holds the methods, findings, fixes and retests for all 5 surfaces. [EV-3.10]
FAILED FIRST, THEN	On Day 5 we found the sign-off problem: a client could open another site's sign-off page by changing the ID in the link. We fixed it with signed, expiring tokens. The retest passed on Day 6.

3.11 AI feature hardening**RISK AT START: MEDIUM****RESULT: CORRECTED**

IN PLAIN WORDS	Text in a manager's notes could trick the quote assistant, and its output went into quotes unchecked. Now input and output are checked.
WHAT WE FOUND	The quote assistant passed raw notes to the AI model and saved its output straight into a quote. It had no timeout and no fallback.
WHAT WE DID	We separated the instructions from the user's notes and filtered the input. The output must pass a schema check (a test of its form) before a draft is saved. We added limits (see item 2.5), a 20-second timeout and the manual quote form as the fallback when OpenAI fails.
HOW WE CHECKED	We ran 25 prompt-injection cases (text that tries to trick the AI into ignoring its rules), with 0 bypasses after the fix. A spend simulation reached the limit. A simulated provider outage in staging showed the fallback. [EV-3.11]

3.12 Edge protection**RISK AT START: MEDIUM****RESULT: IMPLEMENTED**

IN PLAIN WORDS	Nothing stood in front of the app to stop a flood of requests. Now a firewall filters traffic before it reaches the app.
WHAT WE FOUND	There was no firewall in front of the app, and there were no rate rules.
WHAT WE DID	We set up Vercel Firewall with rate and abuse rules. The Edge Function limits from item 3.6 protect the API.
HOW WE CHECKED	The edge (the network layer in front of the app) absorbed a burst of 5,000 requests. The health check at /health stayed green the whole time. [EV-3.12]

04. Database & data

This area keeps Crewproof's records correct, fast to load and safe to restore after a disaster. The most important result: a full recovery drill brought the app back in 48 minutes, with about 2 minutes of data lost.

14 items: 8 implemented, 5 corrected, 1 verified as sound.

4.1 Schema integrity review

RISK AT START: MEDIUM
RESULT: CORRECTED

IN PLAIN WORDS	The database allowed broken links between records and stored money in a form that can round wrongly. Both are fixed.
WHAT WE FOUND	11 foreign keys (rules that link a record to the record it belongs to) were missing. Money was stored as floating-point numbers. <code>jobs.site_id</code> was nullable, so a job could have no site. There was no unique rule on memberships (<code>user_id</code> , <code>company_id</code>).
WHAT WE DID	We added the foreign keys and cascade rules (what happens to linked records when one is deleted). We moved money to integer cents. We fixed the nullable columns and added 4 unique rules. We cleaned the existing data first (see item 4.1).
HOW WE CHECKED	The database now refuses invalid records. We tested how linked records behave on a copy of production data. [EV-4.1]

4.2 Foreign-key and query indexing

RISK AT START: MEDIUM
RESULT: IMPLEMENTED

IN PLAIN WORDS	The job board was slow because the database read every job to find the right ones. Now indexes find them directly.
WHAT WE FOUND	The links between tables had no indexes. An index works like the index of a book: the database finds rows without reading all of them. The job board query read the whole <code>jobs</code> table each time.
WHAT WE DID	We added 17 indexes. Each one is tied to a query we measured.
HOW WE CHECKED	We compared query plans before and after. The job board query fell from 1,840 ms to 38 ms. We documented each index decision. [EV-4.2]

4.3 N+1 query removal

RISK AT START: MEDIUM
RESULT: CORRECTED

IN PLAIN WORDS	The job board asked the database for each job's details one job at a time. Now it asks for them together.
WHAT WE FOUND	The job board loaded each job's site, crew and photos one at a time: 1 query for the list and 3 more for each job (the "N+1" pattern).
WHAT WE DID	We used joins and batched loads (related data fetched in one request) on 6 list views.
HOW WE CHECKED	For a job board with 50 jobs, the number of queries fell from 151 to 4. [EV-4.3]

4.4 Atomic multi-step writes

RISK AT START: HIGH
RESULT: CORRECTED

IN PLAIN WORDS	An error while closing a job could leave it done but with no invoice, so a contractor could miss a bill. Now that cannot happen.
WHAT WE FOUND	"Complete job and create invoice" ran as 4 separate writes, and a failure in the middle stopped the rest.
WHAT WE DID	We moved 3 workflows into database functions that each run in one transaction (all steps succeed together or none at all).
HOW WE CHECKED	We injected a failure between the writes. The database rolled back everything and left no partial records. [EV-4.4]

4.5 Connection pooling**RISK AT START: NONE****RESULT: VERIFIED AS SOUND**

IN PLAIN WORDS	Too many users at once can use up the database's connections and lock people out. We checked: the setup already copes.
WHAT WE FOUND	The app reaches the database through the Supabase API and its built-in pooler, which shares a small number of database connections among many users.
WHAT WE DID	No change was needed. We documented the pool settings.
HOW WE CHECKED	During the 400-user load test, the peak was 38 of 60 connections, with 0 connection errors. [EV-4.5]

4.6 Version-controlled migrations**RISK AT START: HIGH****RESULT: IMPLEMENTED**

IN PLAIN WORDS	Database changes were made by hand with no record, so no one could rebuild the database. Now every change is a script in the code.
WHAT WE FOUND	All changes to the database structure were made in the Supabase dashboard, and the repository held no migrations.
WHAT WE DID	We created a baseline migration from production. Then we wrote every sprint change as an ordered migration.
HOW WE CHECKED	We applied the migration sequence to a clean database. The result was identical to the production structure. [EV-4.6]

4.7 Verified backups and restore**RISK AT START: HIGH****RESULT: CORRECTED**

IN PLAIN WORDS	No one had ever restored a database backup, and the job photos had no backup. Now both are backed up, and a restore worked.
WHAT WE FOUND	Daily database backups were on, kept for 7 days. The stored files (70,000 photos) were in no backup.
WHAT WE DID	We turned on point-in-time recovery for 14 days. We added a nightly copy of all files to a separate backup project in another region, with its own credentials. We wrote the restore procedure.
HOW WE CHECKED	We restored into an isolated project. The row counts and 200 sampled records matched, and the photos opened. [EV-4.7]

4.8 Consistent time handling**RISK AT START: MEDIUM****RESULT: CORRECTED**

IN PLAIN WORDS	Job times could move by an hour when the clocks changed. Now times are stored in one standard time and shown in the site's zone.
WHAT WE FOUND	Timestamps were stored correctly, but job dates were saved as local date strings (text with no time zone). Jobs for a contractor in Arizona moved by one hour at the daylight-saving change.
WHAT WE DID	We now store all times in UTC (one world standard time) and each site's time zone on sites. We documented the display rules.
HOW WE CHECKED	We tested times across 4 US time zones, daylight-saving dates and midnight boundaries. [EV-4.8]

4.9 Recoverable deletion**RISK AT START: MEDIUM****RESULT: IMPLEMENTED**

IN PLAIN WORDS	One wrong click deleted a site or a job for good. Now deleted sites, jobs and quotes can be restored for 30 days.
WHAT WE FOUND	In the last quarter, Crewproof's support team rebuilt 3 deleted sites from memory.
WHAT WE DID	We added soft deletion for sites, jobs and quotes: a deleted record is hidden first, not removed. After 30 days, it is removed for good under the retention policy.
HOW WE CHECKED	We deleted a test site and restored it. Deleted records do not appear in any normal query. [EV-4.9]

4.10 Staging seed data**RISK AT START: LOW****RESULT: IMPLEMENTED**

IN PLAIN WORDS	There was no test data, so changes were tested on live customer data. Now a script fills staging with realistic, invented data.
WHAT WE FOUND	Luis tested against production.
WHAT WE DID	We wrote a seed script (a script that fills a database with test data). It creates 3 companies, all 5 roles, 60 sites, 900 jobs and Stripe test customers, with no real personal data.
HOW WE CHECKED	We ran the seed on a clean staging database and ran the core flows end to end. [EV-4.10]

4.11 Orphaned-record cleanup**RISK AT START: LOW****RESULT: IMPLEMENTED**

IN PLAIN WORDS	The database and storage held thousands of leftover records and files that belonged to nothing. Now they are cleaned up safely.
WHAT WE FOUND	The database held 2,310 orphaned rows: records whose parent was gone, such as checklist items with no checklist and photos with no job. Storage held 4.2 GB of files linked to nothing.
WHAT WE DID	We removed them after a dry run (a trial that changes nothing). Removed files stay on hold for 30 days in case one is needed.
HOW WE CHECKED	We compared the dry-run report with the final run. Valid linked records did not change. [EV-4.11]

4.12 Disaster recovery drill**RISK AT START: HIGH****RESULT: IMPLEMENTED**

IN PLAIN WORDS	No one knew if Crewproof could rebuild the app after a disaster. Now we have done it: the app came back in 48 minutes.
WHAT WE FOUND	No one had ever done a disaster recovery drill (a full practice rebuild of the app from its backups).
WHAT WE DID	We restored the database, the stored files, the Edge Functions and the configuration into a fresh environment. We wrote the procedure.
HOW WE CHECKED	In the timed drill on Day 8, the app recovered in 48 minutes. The data loss window was about 2 minutes. The integrity checks passed. [EV-4.12]
FAILED FIRST, THEN	The first drill on Day 6 failed. The database came back, but the photos did not, because the stored files were in no backup. We added the storage copy (see item 4.7). The second drill passed on Day 8.

4.13 Data model diagram**RISK AT START: LOW****RESULT: IMPLEMENTED**

IN PLAIN WORDS	No one had a picture of how Crewproof's data is organized, which slows every new developer and reviewer. Now a diagram shows it.
WHAT WE FOUND	There was no diagram of the database.
WHAT WE DID	We drew a diagram of the 24 tables, their relationships and the tenant boundaries (the lines that keep each company's data apart).
HOW WE CHECKED	We compared the diagram with the final structure and the migrations. [EV-4.13]

4.14 File storage lifecycle**RISK AT START: MEDIUM****RESULT: IMPLEMENTED**

IN PLAIN WORDS	Uploaded files were never checked for malware and were kept forever. Now each upload is scanned, and old photos are moved, then deleted.
WHAT WE FOUND	Uploads had no signed links, no limits, no malware scan and no retention rule.

WHAT WE DID	We added signed links and limits (see item 3.2) and a malware scan in a background worker. Photos move to cold storage (cheaper storage for old files) after 18 months. They are deleted after 3 years unless the contract says otherwise.
HOW WE CHECKED	An unsigned fetch was refused. Oversized and disallowed files were refused. The scan blocked the EICAR test file (a harmless file that malware scanners treat as malware). The retention job removed expired test uploads. [EV-4.14]

05. Payments & billing

This area protects the money: Crewproof must bill each contractor correctly, and nobody may fake a payment. The most important result: the app now accepts a payment message only when it really comes from Stripe.

10 items: 4 implemented, 5 corrected, 1 verified as sound.

5.1 Webhook signature verification

RISK AT START: CRITICAL

RESULT: CORRECTED

IN PLAIN WORDS	Anyone could send a fake "payment received" message to mark an invoice paid or upgrade a plan for free. Now the app refuses fakes.
WHAT WE FOUND	The Stripe webhook (the message Stripe sends to the app when something happens, such as a payment) did not check signatures.
WHAT WE DID	We added a signature check on the raw request body before any processing.
HOW WE CHECKED	Valid events were processed. Altered and unsigned messages were refused (400, "bad request"). [EV-5.1]

5.2 Idempotent webhook handling

RISK AT START: HIGH

RESULT: CORRECTED

IN PLAIN WORDS	When Stripe sent the same message twice, the app made duplicate invoices. Now a repeated message has no second effect.
WHAT WE FOUND	Repeated deliveries of the same Stripe message created duplicate records. We found 7 duplicate invoices in production.
WHAT WE DID	The app now records each event once in <code>stripe_events</code> before it acts. This makes the handling idempotent (repeat-safe). We reconciled the 7 duplicates with Maya.
HOW WE CHECKED	We replayed events 5 times and delivered them concurrently (at the same time). Each event had one business effect. [EV-5.2]

5.3 Complete billing event lifecycle

RISK AT START: HIGH

RESULT: CORRECTED

IN PLAIN WORDS	Cancellations, refunds and disputes were not recorded in the app, so its billing records were wrong. Now the app handles them, in the right order.
WHAT WE FOUND	The app reacted to only 2 Stripe events: a finished checkout (<code>checkout.session.completed</code>) and a paid invoice (<code>invoice.paid</code>).
WHAT WE DID	The app now handles 11 event types. It checks the order, so a late or out-of-order event does no harm.
HOW WE CHECKED	We tested 14 lifecycle paths with Stripe test clocks (a Stripe tool that moves time forward to test months of billing). The final account state was correct in each. [EV-5.3]

5.4 Server-side entitlements

RISK AT START: HIGH

RESULT: CORRECTED

IN PLAIN WORDS	A contractor on the Starter plan could use paid Growth features. Now the server and the database enforce each plan's limits.
----------------	--

WHAT WE FOUND	The plan limits (sites, users, quote assistant) were checked only in the screens, so a Starter company could call the API directly and use Growth features.
WHAT WE DID	We now enforce the limits in database rules and server functions.
HOW WE CHECKED	We tried paid actions with trial, expired, Starter and Growth accounts. The result was correct each time. [EV-5.4]

5.5 Separated payment environments

RISK AT START: MEDIUM

RESULT: CORRECTED

IN PLAIN WORDS	Test versions of the app used Crewproof's real payment keys, so a test could create live payment records. Now tests use Stripe's test mode.
WHAT WE FOUND	Lovable preview builds used the live Stripe keys.
WHAT WE DID	Local, preview and staging now use test keys. The live keys are only in production.
HOW WE CHECKED	We inspected the provider accounts. A test-mode payment in staging created no live record. [EV-5.5]

5.6 Billing reconciliation check

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	No one compared the app's billing records with Stripe, so mismatches went unseen. Now a script compares them, and the mismatches are corrected.
WHAT WE FOUND	3 companies had the wrong plan in the app.
WHAT WE DID	We wrote a reconciliation script. We corrected the 3 plan mismatches and the 7 duplicates from item 5.2.
HOW WE CHECKED	The script found and corrected mismatches that we seeded on purpose. [EV-5.6]

5.7 Failed-payment recovery

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	A failed payment did nothing, so some contractors used the app unpaid. Now Stripe retries, reminders go out, and access later becomes read-only.
WHAT WE FOUND	2 companies used the app for 2 months unpaid.
WHAT WE DID	We turned on Stripe Smart Retries (automatic new payment attempts). We added 3 reminder emails and a 10-day grace period. After that, the company has read-only access.
HOW WE CHECKED	We simulated a failure, the reminders, a recovery and the end of the grace period. [EV-5.7]

5.8 Customer billing portal

RISK AT START: NONE

RESULT: VERIFIED AS SOUND

IN PLAIN WORDS	Contractors need a safe place to change their card and see invoices. We checked: the Stripe billing portal already worked.
WHAT WE FOUND	The Stripe Customer Portal was already linked for invoices and card changes.
WHAT WE DID	No change was needed. Plan changes in the portal now go through the fixed webhook (item 5.3), so they also update the app.
HOW WE CHECKED	A signed-in customer opened the portal and changed the card and the plan. The app updated. [EV-5.8]

5.9 Scheduled reconciliation

RISK AT START: LOW

RESULT: IMPLEMENTED

IN PLAIN WORDS	A billing mismatch could go unnoticed for months. Now a job compares the app with Stripe every night and alerts on any difference.
WHAT WE FOUND	There was no scheduled comparison between the app's billing records and Stripe.
WHAT WE DID	We added a nightly reconciliation job with an alert on any mismatch.

HOW WE CHECKED We ran the job against test mismatches. The alert arrived, and the output was correct. [EV-5.9]

5.10 Billing operations runbook

RISK AT START: LOW

RESULT: IMPLEMENTED

IN PLAIN WORDS Refunds and free months were done differently each time. Now a runbook explains each task and the Stripe access it needs.

WHAT WE FOUND Crewproof did refunds and free months ad hoc in Stripe, with no written steps.

WHAT WE DID We wrote a runbook for refunds, free access and disputes. It names the Stripe role that each step needs.

HOW WE CHECKED We walked through each procedure in Stripe test mode. [EV-5.10]

06. Error handling & reliability

This area makes the app tell users the truth when something fails, and stops one broken service from taking down the rest. The most important result: Crewproof now learns about errors from its own alerts, not from customer emails.

10 items: 9 implemented, 1 corrected.

6.1 Consistent exception handling

RISK AT START: MEDIUM

RESULT: CORRECTED

IN PLAIN WORDS The app ignored some errors, so a crew member saw "Saved" even when a photo upload had failed. Now every error is handled or reported.

WHAT WE FOUND The code had 43 empty or log-only catch blocks (places that catch an error and then ignore it).

WHAT WE DID We replaced each one with a recovery step or a reported error.

HOW WE CHECKED We injected failures on 12 flows. Each one showed the correct message and created an error record. [EV-6.1]

6.2 Backend handlers and frontend boundaries

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS One error anywhere turned the whole screen white. Now an error affects only its own part of the screen, and the user can retry.

WHAT WE FOUND There was no shared error handler on the server and no error boundary in the screens.

WHAT WE DID We added a shared error handler to the Edge Functions. We added error boundaries (walls that stop an error from spreading) to each main screen, with a retry button.

HOW WE CHECKED We triggered server and component errors. The server gave safe responses, and the rest of the screen kept working. [EV-6.2]

6.3 Error tracking

RISK AT START: HIGH

RESULT: IMPLEMENTED

IN PLAIN WORDS Crewproof learned about errors only when customers emailed. Now an error tracker records each error and sends an alert.

WHAT WE FOUND There was no error tracking.

WHAT WE DID We added Sentry (an error-tracking service) to the frontend and the Edge Functions. It has private source maps (so errors point to the original code), environment labels and alerts to the #ops-alerts channel.

HOW WE CHECKED A test error showed the right file and line and the right environment, and it sent an alert. [EV-6.3]

6.4 External-call timeouts**RISK AT START: MEDIUM****RESULT: IMPLEMENTED**

IN PLAIN WORDS	If OpenAI, Resend or Stripe was slow, the app could wait without end. Now each call has a time limit and a clear failure message.
WHAT WE FOUND	Calls to OpenAI, Resend and Stripe had no timeouts.
WHAT WE DID	We added timeouts to all external calls: OpenAI 20 s, Resend 10 s and Stripe 15 s.
HOW WE CHECKED	We simulated a slow service. The wait had a limit, and the app showed a clear failure state. [EV-6.4]

6.5 Retries with backoff**RISK AT START: LOW****RESULT: IMPLEMENTED**

IN PLAIN WORDS	One failed email meant that an invoice email was never sent. Now the app retries safe tasks a limited number of times, but never payments.
WHAT WE FOUND	The app never tried a failed call again.
WHAT WE DID	We added bounded retries with backoff (a longer wait before each new attempt) on safe calls only: email and quote generation.
HOW WE CHECKED	We simulated short and long failures. The retry limits held, and there were no duplicate effects. [EV-6.5]

6.6 Background processing**RISK AT START: MEDIUM****RESULT: IMPLEMENTED**

IN PLAIN WORDS	Large exports and the monthly invoice run happened while the user waited, and big exports failed. Now they run in the background.
WHAT WE FOUND	Quote generation, CSV export and monthly invoice emails ran inside the web request. Large exports timed out.
WHAT WE DID	We moved them to Supabase Queues and a background worker. The user sees their status.
HOW WE CHECKED	A 12,000-row export and a 400-invoice batch finished, and their status was visible. [EV-6.6]

6.7 Reliable job execution**RISK AT START: MEDIUM****RESULT: IMPLEMENTED**

IN PLAIN WORDS	A job that ran twice could send two emails, and a failed job could be lost. Now neither can happen.
WHAT WE FOUND	There were no background jobs before item 6.6, so nothing made a job safe to repeat or kept a job that failed.
WHAT WE DID	A second run of a job now gives the same result as one run. A job that keeps failing goes to a dead-letter queue (a separate list of failed jobs), with a recovery step.
HOW WE CHECKED	We replayed jobs and got one result each. When the retries ran out, the job went to the dead-letter queue, and we could recover it. [EV-6.7]

6.8 Graceful degradation**RISK AT START: MEDIUM****RESULT: IMPLEMENTED**

IN PLAIN WORDS	If OpenAI was down, the whole quotes screen failed. Now one service can fail while the rest of the app keeps working.
WHAT WE FOUND	No feature had a fallback for an outside service that failed.
WHAT WE DID	The quote assistant, email and the photo scan can each fail without stopping the rest of the app.
HOW WE CHECKED	We switched off each integration in staging. Each one showed its fallback, and scheduling and checklists were not affected. [EV-6.8]

6.9 Health endpoint**RISK AT START: LOW****RESULT: IMPLEMENTED**

IN PLAIN WORDS	A monitor had no simple way to ask the app if it was working. Now one address reports the state of each main part.
WHAT WE FOUND	The app had no health endpoint (an address that reports if the app is working).
WHAT WE DID	We added <code>/health</code> . It reports the status of the database, storage, the queue and Stripe, without details.
HOW WE CHECKED	We checked healthy and degraded responses. No secrets or internal details were exposed. [EV-6.9]

6.10 Staging outage simulation**RISK AT START: LOW****RESULT: IMPLEMENTED**

IN PLAIN WORDS	No one knew how the app behaves when OpenAI or Stripe goes down. Now we have tested both outages on staging.
WHAT WE FOUND	No one had ever simulated an outage of a service that the app depends on.
WHAT WE DID	We simulated OpenAI and Stripe outages in staging.
HOW WE CHECKED	We recorded the queue behavior, the user messages and the recovery for both outages. [EV-6.10]

07. Environments, CI/CD & deployment

This area controls how new code reaches the live app. Before the sprint, every edit went straight to customers. The most important result: changes are now tested automatically, go to a test copy first and need approval before they go live.

13 items: 10 implemented, 3 corrected.

7.1 Three isolated environments**RISK AT START: HIGH****RESULT: IMPLEMENTED**

IN PLAIN WORDS	Every edit from Lovable or Cursor went straight to live customers. Now there are three separate copies: local, staging and production.
WHAT WE FOUND	Production was the only environment, so changes from Lovable and Cursor (an AI code editor) had no place to be tested first.
WHAT WE DID	We set up local (the developer's computer), staging and production, each with its own database, keys and settings.
HOW WE CHECKED	An action on staging had no effect on production. [EV-7.1]

7.2 Protected production branch**RISK AT START: HIGH****RESULT: IMPLEMENTED**

IN PLAIN WORDS	Anyone, including Lovable, could change the main version of the code without review. Now each change needs passing checks and one approval.
WHAT WE FOUND	Anyone could push code to <code>main</code> , the main branch (version line) of the code. Lovable pushed to it directly.
WHAT WE DID	We protected <code>main</code> . A change now needs a pull request, passing checks and one approval.
HOW WE CHECKED	A failing pull request could not be merged. [EV-7.2]

7.3 Pull-request CI**RISK AT START: HIGH****RESULT: IMPLEMENTED**

IN PLAIN WORDS	Code changes went live without any automatic check. Now automatic checks run on every proposed change before it can be accepted.
----------------	--

WHAT WE FOUND	There was no CI, so a broken change could reach customers with no warning.
WHAT WE DID	We added GitHub Actions on every pull request. It runs lint (style and error checks), the type check, the build, the unit tests, the tenant-isolation suite and the secret scan.
HOW WE CHECKED	We kept the results of one failing and one passing change. [EV-7.3]

7.4 Staging deployment and production promotion

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	Nothing stood between an accepted change and the live app. Now changes go to staging first, and only Maya or Luis can release them.
WHAT WE FOUND	There was no staging step and no approval gate before production.
WHAT WE DID	Merges now deploy to staging. Production needs a manual promotion (an approval to go live) by Maya or Luis.
HOW WE CHECKED	We deployed a change to staging and then promoted it through the gate. [EV-7.4]

7.5 Migration-aware deployment

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	Database changes and code changes could go live in the wrong order and break the app. Now database changes run first.
WHAT WE FOUND	Releases did not handle database changes, because there were no migrations (see item 4.6).
WHAT WE DID	Migrations now run before the app deploys. A failed migration stops the deploy.
HOW WE CHECKED	We deployed a database change in order. A broken migration stopped the release. [EV-7.5]

7.6 Tested rollback

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	If a release broke the app, there was no plan to go back to the last version that worked. Now there is a tested one.
WHAT WE FOUND	There was no rollback plan.
WHAT WE DID	The app can roll back through Vercel. We wrote the database changes to be backward compatible, so the earlier app version still works with the new database. We documented the rollback steps.
HOW WE CHECKED	We rolled back a test release in 2 minutes, and the data was kept. [EV-7.6]

7.7 Pull-request previews

RISK AT START: MEDIUM

RESULT: CORRECTED

IN PLAIN WORDS	Test versions of the app were public on the internet and used real customer data. Now they are private and use test data.
WHAT WE FOUND	Vercel previews (a test version of the app for each change) were public and used production data.
WHAT WE DID	Previews now use a Supabase preview branch (a separate test database) and test keys. Only people signed in to Crewproof's Vercel team can open them (Vercel authentication).
HOW WE CHECKED	We opened a preview from a test pull request and confirmed that it was isolated. [EV-7.7]

7.8 Contribution guide and PR template

RISK AT START: LOW

RESULT: IMPLEMENTED

IN PLAIN WORDS	A new developer had no written rules for how to propose a change. Now a guide and a standard form explain it.
WHAT WE FOUND	There was no contribution guide and no pull request template.
WHAT WE DID	We added CONTRIBUTING.md and a pull request template. The template asks for the change, how it was tested and the release notes.

HOW WE CHECKED We created a pull request with the template and followed the guide on a clean checkout. [EV-7.8]

7.9 Automated dependency updates

RISK AT START: LOW

RESULT: IMPLEMENTED

IN PLAIN WORDS Outside code packages went out of date, and security fixes did not arrive. Now a tool proposes updates every week, and security fixes at once.

WHAT WE FOUND Nothing kept the packages up to date.

WHAT WE DID We set up Renovate (a tool that proposes package updates). It groups updates weekly and proposes security updates at once.

HOW WE CHECKED An update proposal ran CI and needed review. [EV-7.9]

7.10 Timed deployment

RISK AT START: LOW

RESULT: IMPLEMENTED

IN PLAIN WORDS There was no repeatable way to release a change. Now one pipeline does it, from merge to the live app in 6 minutes 40 seconds.

WHAT WE FOUND There was no deployment pipeline (an automatic series of steps that puts a change live).

WHAT WE DID We built a pipeline with cached builds.

HOW WE CHECKED A merge reached production in 6 minutes 40 seconds. We recorded the steps. [EV-7.10]

7.11 Independent repository and hosting

RISK AT START: HIGH

RESULT: CORRECTED

IN PLAIN WORDS The live app and its code sat in personal accounts linked to Luis and Lovable. Now Crewproof owns both, and releases no longer need Lovable.

WHAT WE FOUND Production deployed from the personal Vercel account of Luis Ferreira (contract developer). The repository was in a personal GitHub account linked to Lovable.

WHAT WE DID We moved the repository to a Crewproof GitHub organization and the hosting to a Vercel team owned by Crewproof. Lovable can still be used on its own `lovable` branch, but the app no longer needs it to deploy.

HOW WE CHECKED A fresh clone (copy) of the repository deployed to staging through the pipeline, with no Lovable step. [EV-7.11]

7.12 Feature flags and kill switches

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS A broken feature could not be turned off without a new release. Now three risky features each have an off switch.

WHAT WE FOUND Crewproof had no way to turn off one feature on its own.

WHAT WE DID We added PostHog feature flags (on and off switches) to the quote assistant, the photo scan and bulk email.

HOW WE CHECKED We switched off the quote assistant in production. It was hidden, and the rest of the app stayed usable. [EV-7.12]

7.13 Domain, DNS and TLS hygiene

RISK AT START: MEDIUM

RESULT: CORRECTED

IN PLAIN WORDS Crewproof's domain was unlocked and set to renew on an expired card, so it could lapse or be moved. Both are fixed.

WHAT WE FOUND HTTPS (the encrypted connection) was on, but HSTS and CAA were missing. HSTS tells browsers to always use the encrypted connection. CAA names the only companies that may issue the site's security certificate. The registrar lock was off, and the renewal was on an expired card.

WHAT WE DID	We added HSTS and a CAA record, turned on the registrar lock and moved the renewal to the company card. We moved DNS (the internet's address book) to Cloudflare and documented it.
HOW WE CHECKED	An external scan of the encrypted connection (TLS) gave grade A+. The DNS export is filed in the runbooks. [EV-7.13]

08. Logging, monitoring & alerting

This area makes sure that Crewproof knows when something goes wrong before its customers do. The most important result: a test outage sent an alert in 2 minutes, and the logs no longer hold phone numbers or tokens.

8 items: 6 implemented, 2 corrected.

8.1 Structured, sanitized logs

RISK AT START: MEDIUM

RESULT: CORRECTED

IN PLAIN WORDS	The app's logs stored client phone numbers and sign-in tokens, visible to anyone who could read the logs. Now the logs leave them out.
WHAT WE FOUND	The code used <code>console.log</code> to record whole request bodies.
WHAT WE DID	The logs are now JSON (a structured format) with request IDs and user IDs, so each request can be traced. We removed passwords, tokens and contact details.
HOW WE CHECKED	We traced a test request across the app, a function and the database. No sensitive fields were in the logs. [EV-8.1]

8.2 Uptime monitoring

RISK AT START: HIGH

RESULT: IMPLEMENTED

IN PLAIN WORDS	If the app went down, no one at Crewproof knew until a customer complained. Now a monitor checks the app every minute.
WHAT WE FOUND	Crewproof had no uptime monitoring and no outage alert.
WHAT WE DID	We set up Better Stack checks on the app and on <code>/health</code> , every minute.
HOW WE CHECKED	In a controlled failure, the alert came in 2 minutes, and the recovery was reported. [EV-8.2]

8.3 Operational threshold alerts

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	The app could slowly get worse, with more errors or slower pages, and no one would notice. Now alerts go out at set levels.
WHAT WE FOUND	There were no alerts on error rate, speed, database load or waiting jobs.
WHAT WE DID	We added alerts on the error rate, p95 latency (the time within which 95 of every 100 requests finish), database connections and the queue backlog.
HOW WE CHECKED	We triggered each condition and recorded the threshold and the behavior. [EV-8.3]

8.4 Actionable alert routing

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	Alerts had nowhere set to go and no steps to follow. Now every alert reaches Slack and Maya, with a link to its response steps.
WHAT WE FOUND	There was no alert routing.
WHAT WE DID	All alerts go to the <code>#ops-alerts</code> Slack channel and to Maya's email. Each one links to its runbook step.
HOW WE CHECKED	Test alerts arrived with context and response steps. [EV-8.4]

8.5 Log retention**RISK AT START: LOW****RESULT: IMPLEMENTED**

IN PLAIN WORDS	No one had decided how long to keep logs. Now each kind of log has a set period.
WHAT WE FOUND	No retention period was set. Each service kept its default.
WHAT WE DID	We set and documented the retention: app logs 30 days, the audit log 7 years and error data 90 days.
HOW WE CHECKED	We checked the configured periods in each service. [EV-8.5]

8.6 Core product analytics**RISK AT START: LOW****RESULT: CORRECTED**

IN PLAIN WORDS	Crewproof could not see clearly how many signups became paying customers. Now named events track each step.
WHAT WE FOUND	PostHog recorded clicks automatically (autocapture), but there were no named events for signup, activation or payment.
WHAT WE DID	We added 9 named events from signup to the first paid month.
HOW WE CHECKED	We ran the core journeys and checked the event names and properties. [EV-8.6]

8.7 Public status page**RISK AT START: LOW****RESULT: IMPLEMENTED**

IN PLAIN WORDS	During outages, customers had to email support to find out what was wrong. Now a public status page shows the app's state.
WHAT WE FOUND	There was no status page.
WHAT WE DID	We set up <code>status.crewproof.com</code> on Better Stack.
HOW WE CHECKED	We published a test incident. The page was reachable with the app switched off. [EV-8.7]

8.8 Unified operations dashboard**RISK AT START: LOW****RESULT: IMPLEMENTED**

IN PLAIN WORDS	The key numbers were spread across several services. Now one dashboard shows them together.
WHAT WE FOUND	There was no single dashboard.
WHAT WE DID	We built one PostHog dashboard with uptime, error rate, latency, signups and monthly revenue.
HOW WE CHECKED	We compared each figure with its source system and checked the refresh. [EV-8.8]

09. Performance

This area makes the app fast enough for the busy morning, when crews start work at the same time. The most important result: the app now serves 400 users at once, and 95 of every 100 requests finish within 410 ms.

7 items: 5 implemented, 2 corrected.

9.1 Appropriate caching**RISK AT START: LOW****RESULT: IMPLEMENTED**

IN PLAIN WORDS	The app loaded the same plan and checklist templates again on every screen. Now it keeps a short-term copy for each company.
WHAT WE FOUND	There was no caching.
WHAT WE DID	We cached the reference data (kept a short-term copy), with clear rules to refresh it when the data changes. Each company has its own cache.

HOW WE CHECKED We measured cache hits and misses and tested the refresh. No data crossed between companies. [EV-9.1]

9.2 Load test and bottleneck fixes

RISK AT START: HIGH
RESULT: IMPLEMENTED

IN PLAIN WORDS The app slowed down when crews started work in the morning, and no one knew its limit. Now it serves 400 users at once.

WHAT WE FOUND No one had ever run a load test (a test that sends many users to the app at once).

WHAT WE DID We wrote a k6 test (k6 is a load-testing tool) of the morning start: sign-in, job board, checklist and photo upload. We fixed the indexes (item 4.2), the N+1 queries (item 4.3) and the image sizes (item 9.4), and we cached the reference data (item 9.1).

HOW WE CHECKED Before the fixes, at 150 concurrent users (users at the same time), p95 was 3.9 s and 2.1% of requests failed. After the fixes, at 400 concurrent users for 30 minutes, p95 was 410 ms (95 of every 100 requests finished within 0.41 seconds) and 0.02% failed. The test ran on staging, on the same plan as production. [EV-9.2]

FAILED FIRST, THEN

The first run on Day 4 failed already at 150 users (p95 3.9 s). The target was p95 under 800 ms at 300 users. After the fixes, the Day 7 run met the target and passed up to 400 users.

9.3 Bundle optimization and lazy loading

RISK AT START: MEDIUM
RESULT: CORRECTED

IN PLAIN WORDS Every page made the phone download 2.4 MB of code, which is slow on a weak signal. Now each page loads only what it needs.

WHAT WE FOUND The app sent one 2.4 MB JavaScript bundle (the app's code file) on every page.

WHAT WE DID We split the code by route (page). 2 heavy chart and PDF libraries now load only when a page needs them.

HOW WE CHECKED The first load fell from 2.4 MB to 690 KB. We tested the routes on a slow 4G profile. [EV-9.3]

9.4 Image optimization

RISK AT START: MEDIUM
RESULT: IMPLEMENTED

IN PLAIN WORDS The job board showed full-size phone photos as small thumbnails. Now the app makes small copies at upload.

WHAT WE FOUND The app showed full phone photos, about 5 MB each, as thumbnails.

WHAT WE DID The app now makes resized WebP versions (a compact image format) at upload: one thumbnail and one full view.

HOW WE CHECKED For a job board with 40 photos, the data transferred fell from 190 MB to 3.1 MB. The images were clear at display size. [EV-9.4]

9.5 Lighthouse performance review

RISK AT START: LOW
RESULT: IMPLEMENTED

IN PLAIN WORDS No one had measured how fast the app is on a phone. Now the 4 main pages score between 91 and 98 out of 100.

WHAT WE FOUND Our first test gave the job board a mobile performance score of 41 in Lighthouse (Google's page-quality test, scored out of 100).

WHAT WE DID We improved 4 pages. The targets on mobile were a performance score of 85 or more and an accessibility score of 100.

HOW WE CHECKED The final mobile scores are: job board 91, checklist 94, client portal 93 and login 98. We checked the key interactions by hand. [EV-9.5]

9.6 Written capacity statement**RISK AT START: MEDIUM****RESULT: IMPLEMENTED**

IN PLAIN WORDS	The investor's technical advisor asked how many users the app can serve. Now a written statement gives the measured answer.
WHAT WE FOUND	There was no capacity statement.
WHAT WE DID	The statement says that the app served 400 concurrent users on the current plans, about 4x today's peak. It also gives the plan for 2,000 users (5x the tested load), with the changes and their monthly cost.
HOW WE CHECKED	Each figure links to the test in item 9.2. The statement marks the 5x plan as a projection, not tested. [EV-9.6]

9.7 Accessibility baseline**RISK AT START: MEDIUM****RESULT: CORRECTED**

IN PLAIN WORDS	Some people could not use key screens, for example with a screen reader or a keyboard. Now the core pages score 100.
WHAT WE FOUND	Icon buttons had no labels, status badges had low contrast, and the checklist could not be used with a keyboard.
WHAT WE DID	We fixed these on signup, login, the job board, the checklist and client sign-off.
HOW WE CHECKED	The automated accessibility score is 100 on the core pages. We completed signup, checkout and the checklist with a keyboard only. [EV-9.7]

10. Code health & AI development guardrails

This area makes the code easy and safe for any developer, or any AI tool, to change. The most important result: automatic checks now block the most dangerous kinds of change, such as a new table without row-level security, whoever writes it.

11 items: 9 implemented, 2 corrected.

10.1 Dead code and duplicate logic cleanup**RISK AT START: LOW****RESULT: CORRECTED**

IN PLAIN WORDS	The code had 3 slightly different ways to add up an invoice, so totals could differ between screens. Now there is one.
WHAT WE FOUND	The code had 64 unused components and 3 copies of the invoice total logic.
WHAT WE DID	We removed about 9,400 lines and kept one invoice total function.
HOW WE CHECKED	We listed the removed and merged code. We ran the regression tests on invoices, quotes and jobs. [EV-10.1]

10.2 Consistent state management**RISK AT START: MEDIUM****RESULT: CORRECTED**

IN PLAIN WORDS	A job's status was stored in 3 places, so screens could disagree. Now there is one source, and every screen shows the same state.
WHAT WE FOUND	The job board and the job page could show different states.
WHAT WE DID	Server data now goes through TanStack Query only (one library that fetches and stores server data). We documented the pattern.
HOW WE CHECKED	We tested refresh, edit and navigation. Every screen showed the same state. [EV-10.2]

10.3 Typed API boundaries**RISK AT START: MEDIUM****RESULT: IMPLEMENTED**

IN PLAIN WORDS	A data mismatch between the app and its server functions could break a feature without warning. Now the automatic checks catch it.
WHAT WE FOUND	The database types (descriptions of the data in each table) were not generated, and the Edge Function inputs had no types.
WHAT WE DID	We generated the database types and shared the Zod schemas between the app and the functions.
HOW WE CHECKED	The type check in CI caught a mismatch that we made on purpose. [EV-10.3]

10.4 Documented repository structure**RISK AT START: LOW****RESULT: IMPLEMENTED**

IN PLAIN WORDS	A new developer had no guide to where things are in the code. Now a README explains the layout.
WHAT WE FOUND	The README (the guide file at the top of the code) was the Lovable default.
WHAT WE DID	We wrote a README with the folder layout, the main modules and where each feature lives.
HOW WE CHECKED	A new developer followed the README and traced job completion end to end. [EV-10.4]

10.5 Critical-path smoke tests**RISK AT START: HIGH****RESULT: IMPLEMENTED**

IN PLAIN WORDS	A change could break signup or payments without anyone noticing. Now 14 tests check the 6 most important paths on every change.
WHAT WE FOUND	The app had no automated tests.
WHAT WE DID	We wrote 14 Playwright smoke tests (short tests that click through the app like a user). They cover signup, login, schedule a job, complete a checklist, client sign-off and subscription payment.
HOW WE CHECKED	The suite runs in CI. A break in checklist completion that we made on purpose failed it. [EV-10.5]

10.6 Strict typing**RISK AT START: MEDIUM****RESULT: IMPLEMENTED**

IN PLAIN WORDS	The code's built-in error checker was in its loose mode, so many mistakes went unseen. Now it is strict.
WHAT WE FOUND	TypeScript strict mode was off. Strict mode makes the TypeScript checker find more mistakes before the code runs.
WHAT WE DID	We turned strict mode on and fixed 612 errors, with no suppressions added.
HOW WE CHECKED	We kept the strict configuration and a clean type check. [EV-10.6]

10.7 Auth and billing unit tests**RISK AT START: HIGH****RESULT: IMPLEMENTED**

IN PLAIN WORDS	Nothing tested the code for roles and money, so a change could give too much access or bill wrongly. Now 86 tests cover it.
WHAT WE FOUND	There were no unit tests (small tests of one piece of code) for roles, plan limits, webhooks or invoice totals.
WHAT WE DID	We wrote 86 unit tests on these four areas, with negative and edge cases.
HOW WE CHECKED	All tests pass. In the tests, the code refuses unauthorized access and wrong billing changes. [EV-10.7]

10.8 AI repository guardrails**RISK AT START: MEDIUM****RESULT: IMPLEMENTED**

IN PLAIN WORDS	The AI tools that write much of Crewproof's code could change any file. Now rules guide them, and automatic checks block dangerous changes.
----------------	---

WHAT WE FOUND	Lovable and Cursor could change any file, and no rules guided them.
WHAT WE DID	We wrote AGENTS.md (11 hard rules, protected files and safe patterns), 3 Cursor rules and a Lovable knowledge text. We added CODEOWNERS (a file that names who must approve changes to protected files), and we moved Lovable to its own branch. We added 7 CI checks: no private keys in browser code, row-level security on every table, no open policies, no role writes from the browser, no service key in the browser, a role guard on every user function and no edits to applied migrations.
HOW WE CHECKED	A test pull request that added a table without row-level security failed CI. [EV-10.8]

10.9 Recorded codebase tour

RISK AT START: LOW

RESULT: IMPLEMENTED

IN PLAIN WORDS	There was no walkthrough of the code, so a new developer depended on Luis. Now a recording explains it.
WHAT WE FOUND	There was no recorded walkthrough of the code.
WHAT WE DID	We recorded a 27-minute tour of the repository, the data model and the main decisions.
HOW WE CHECKED	We delivered the recording with 8 chapter markers. [EV-10.9]

10.10 Reproducible local development setup

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	Only Luis's laptop could run the app, and it pointed at the live database. Now any developer can set it up in 11 minutes.
WHAT WE FOUND	The only local setup was on Luis's laptop, and it pointed at production.
WHAT WE DID	The command <code>pnpm setup</code> starts a local Supabase, applies the migrations and adds the seed data. We included <code>.env.example</code> , a settings template without real keys.
HOW WE CHECKED	On a new machine, a developer went from a fresh clone to a running app in 11 minutes, with only the guide. [EV-10.10]

10.11 Open-source licence audit

RISK AT START: LOW

RESULT: IMPLEMENTED

IN PLAIN WORDS	Some free code packages carry licence terms that can limit how a company uses its own code. Now all packages are checked.
WHAT WE FOUND	No one had checked the licences of the open-source packages that the app uses.
WHAT WE DID	We listed the licences of 412 packages and flagged 2. We replaced 1 GPL-3.0 PDF library with an MIT one. GPL-3.0 can require a company to publish its own code, and MIT does not. 1 MPL-2.0 build tool was approved in writing.
HOW WE CHECKED	The licence report is in the due diligence pack and shows 0 unresolved flags. [EV-10.11]

11. Email & communications

This area makes sure that Crewproof's emails, such as invoices, reach the inbox and cannot easily be faked. The most important result: mail services can now confirm that Crewproof's email is real, and invoices no longer go to dead addresses.

3 items: 1 implemented, 1 corrected, 1 verified as sound.

11.1 Sending-domain authentication

RISK AT START: MEDIUM

RESULT: CORRECTED

IN PLAIN WORDS	Invoice emails landed in spam, because mail services could not confirm that they came from Crewproof. Now they reach the inbox.
----------------	---

WHAT WE FOUND	The sending domain had only SPF (a list of servers allowed to send its mail). It had no DKIM (a digital signature on each email) and no DMARC (a policy that tells mail services what to do with mail that fails these checks). 2 customers reported invoices in spam.
WHAT WE DID	We set up SPF, DKIM and DMARC on mail.crewproof.com. The DMARC policy is p=quarantine, so mail that fails the checks goes to spam.
HOW WE CHECKED	The DNS records and the received headers show that SPF, DKIM and DMARC pass. We checked inbox placement at Gmail and Outlook. [EV-11.1]

11.2 Transactional delivery tracking

RISK AT START: NONE

RESULT: VERIFIED AS SOUND

IN PLAIN WORDS	Account and invoice emails must arrive, and Crewproof must be able to prove it. We checked: this already worked.
WHAT WE FOUND	Resend already sent all account and invoice email, with delivery logs.
WHAT WE DID	No change was needed.
HOW WE CHECKED	We sent the 7 critical message types. Each one appeared in the Resend logs and was delivered. [EV-11.2]

11.3 Bounce and complaint handling

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	Invoices kept going to email addresses that no longer worked, and no one knew. Now the app stops and tells the contractor.
WHAT WE FOUND	The app ignored bounces (emails returned as undeliverable). Invoices kept going to 31 dead addresses.
WHAT WE DID	We added Resend webhooks for bounces and complaints. The app now suppresses (stops sending to) those addresses and sends a notice to the contractor.
HOW WE CHECKED	We simulated a bounce and a complaint. The address was suppressed, and later sends were skipped. [EV-11.3]

12. Privacy & compliance foundations

This area shows what personal data Crewproof holds, where it goes and how long it stays. The most important result: Crewproof now has a tested list of its personal data, and the quote assistant no longer sends names and phone numbers to OpenAI.

8 items: 5 implemented, 2 corrected, 1 not applicable.

12.1 Personal-data inventory

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	Crewproof had no list of the personal data it holds, so it could not answer privacy questions with confidence. Now it has one.
WHAT WE FOUND	No document said what personal data the app holds, where it is or why.
WHAT WE DID	We wrote an inventory of 27 personal-data fields. It says where each one is stored, why and who can access it.
HOW WE CHECKED	We traced a sample user through the database, storage, logs and 9 providers. The result matched the inventory. [EV-12.1]

12.2 Data export and deletion paths

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	Crewproof handled each request for a copy or deletion of data by hand. Now users can do both themselves.
----------------	--

WHAT WE FOUND	Users could not export or delete their own data.
WHAT WE DID	We built self-service export (JSON and CSV files) and deletion (see item 1.8).
HOW WE CHECKED	A seeded user's export was complete. The deletion was complete, with the exceptions recorded. Another user could not make the request. [EV-12.2]

12.3 Privacy and terms alignment

RISK AT START: MEDIUM

RESULT: CORRECTED

IN PLAIN WORDS	The privacy policy did not match what the app does with data. Now the differences are resolved with Maya, who decides the wording.
WHAT WE FOUND	The privacy policy came from a template. It did not mention photos, OpenAI or retention.
WHAT WE DID	We listed 6 differences between the policy and what the app does, and we resolved them with Maya.
HOW WE CHECKED	We compared the policy with the inventory and the provider data flows. The approved alignment is recorded. [EV-12.3]

12.4 Cookie consent controls

RISK AT START: NONE

RESULT: NOT APPLICABLE

IN PLAIN WORDS	The app sets only the cookies it needs to work, and Crewproof sells only to US businesses, so this item does not apply today.
WHAT WE FOUND	PostHog runs in cookieless mode. There are no ad or third-party tracking scripts.
WHAT WE DID	We recorded the reason. Crewproof should reopen this item if it adds marketing pixels or sells in the EU or UK.
HOW WE CHECKED	We scanned the cookies and scripts on all app routes. We found only strictly necessary cookies. [EV-12.4]

12.5 Data retention policy

RISK AT START: LOW

RESULT: IMPLEMENTED

IN PLAIN WORDS	No one had decided how long each kind of data is kept. Now a written schedule sets it, and it matches the cleanup jobs.
WHAT WE FOUND	There was no retention policy.
WHAT WE DID	We wrote a retention schedule for 9 data types, such as logs, photos, quotes, invoices and deleted accounts.
HOW WE CHECKED	We compared the schedule with the inventory and the configured jobs. [EV-12.5]

12.6 SOC 2-style controls checklist

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	The investor's technical advisor asked for a list of Crewproof's security controls, with proof. Now there is one. It is not a SOC 2 audit.
WHAT WE FOUND	There was no controls checklist. The advisor's request list asked for one.
WHAT WE DID	We wrote a checklist of 48 controls, each with an owner and an evidence link. It follows the style of SOC 2 (a common security audit standard), but it is not a SOC 2 audit report.
HOW WE CHECKED	Each control links to its configuration or test evidence. [EV-12.6]

12.7 Personal data sent to model providers

RISK AT START: MEDIUM

RESULT: CORRECTED

IN PLAIN WORDS	Every quote request sent client names, site addresses and phone numbers to OpenAI. Now these are removed.
WHAT WE FOUND	The quote assistant sent these fields with the walkthrough notes on every request.

WHAT WE DID	The app now sends only the walkthrough notes and the site size. We recorded the OpenAI API data terms, which say that OpenAI does not train on API data.
HOW WE CHECKED	We traced each model call, listed its fields and matched them to the inventory. [EV-12.7]

12.8 Subprocessor list

RISK AT START: LOW

RESULT: IMPLEMENTED

IN PLAIN WORDS	Crewproof could not show which outside companies handle its customers' data. Now a published list names them.
WHAT WE FOUND	There was no subprocessor list (a subprocessor is an outside company that stores or handles customer data for Crewproof).
WHAT WE DID	We published a list of 9 vendors, with the agreement in place for each.
HOW WE CHECKED	The list matches the provider inventory and the privacy policy. [EV-12.8]

13. Handover & included support

This area is what Crewproof receives at the end: the documents, the recordings and the support after handover. The most important result: all 123 items are accounted for, and one pack holds the evidence for the investor's review.

7 items: 7 implemented.

13.1 Production readiness report

RISK AT START: NONE

RESULT: IMPLEMENTED

IN PLAIN WORDS	An investor needs proof of what was checked. This report gives it for all 123 items, with 0 left open.
WHAT WE FOUND	This report is an output of the sprint, so there was no gap to fix.
WHAT WE DID	We wrote this report. It covers all 123 items, the measured results and the remaining risks. Its appendices cover roles, the penetration test, the recovery drill, personal data, code changes and evidence samples.
HOW WE CHECKED	All 123 of 123 item IDs are accounted for: 122 verified, 1 not applicable with the reason, and 0 open. [EV-13.1]

13.2 System architecture diagram

RISK AT START: LOW

RESULT: IMPLEMENTED

IN PLAIN WORDS	No one had a picture of how the parts of the app connect and where data flows. Now a diagram shows them.
WHAT WE FOUND	There was no architecture diagram.
WHAT WE DID	We drew a diagram of the components, 12 data flows and the 11 external services that the app runs on. It also shows 5 trust boundaries (lines where data passes to a less trusted part). A component table goes with it.
HOW WE CHECKED	We compared the diagram with the delivered environment and the repository. [EV-13.2]

13.3 Operating runbooks

RISK AT START: MEDIUM

RESULT: IMPLEMENTED

IN PLAIN WORDS	Crewproof had no written steps for routine jobs or emergencies. Now 10 runbooks cover them.
WHAT WE FOUND	There were no runbooks for releases, restores, alerts or incidents.
WHAT WE DID	We wrote 10 runbooks: deploy, rollback, key rotation, two restore procedures, billing operations, three alert responses that cover all 10 alert types, and a security incident procedure.

HOW WE CHECKED We walked through each one against the live configuration. Each one links to its rehearsal evidence. [EV-13.3]

13.4 Live technical handover

RISK AT START: NONE

RESULT: IMPLEMENTED

IN PLAIN WORDS Crewproof's team and the investor's technical advisor needed to hear the results and ask questions. We held a recorded session.

WHAT WE FOUND This item is an output of the sprint, so there was no gap to fix at kickoff.

WHAT WE DID We held a 60-minute session with Maya, Luis and the lead investor's technical advisor. We recorded it.

HOW WE CHECKED We delivered the recording, the documents and 6 follow-up answers. [EV-13.4]

13.5 Fourteen-day defect cover

RISK AT START: NONE

RESULT: IMPLEMENTED

IN PLAIN WORDS A defect in our work could show up after handover. For 14 calendar days, Crewproof can report one, and we fix it.

WHAT WE FOUND This item is an output of the sprint, so there was no gap to fix at kickoff.

WHAT WE DID The cover runs from handover to handover + 14 calendar days.

HOW WE CHECKED The dates, the channel and the defect form are in the handover pack. [EV-13.5]

13.6 Technical due diligence pack

RISK AT START: NONE

RESULT: IMPLEMENTED

IN PLAIN WORDS The investor's technical advisor asked for evidence. Now one PDF holds the main documents and answers common questions.

WHAT WE FOUND This item is an output of the sprint, so there was no gap to fix at kickoff.

WHAT WE DID We built one 81-page PDF. It holds the readiness report, the architecture diagram, the data model, the controls checklist, the capacity statement, the subprocessor list and the licence report. It has a contents page, bookmarks and an index of 22 common due diligence questions.

HOW WE CHECKED We checked the pack for completeness, one version reference and readable links. [EV-13.6]

13.7 Thirty-day async access

RISK AT START: NONE

RESULT: IMPLEMENTED

IN PLAIN WORDS Questions often come up after a handover. For 30 calendar days, Crewproof can ask us about the delivered system in writing.

WHAT WE FOUND This item is an output of the sprint, so there was no gap to fix at kickoff.

WHAT WE DID A shared Slack channel is open from handover to handover + 30 calendar days.

HOW WE CHECKED The channel, the access steps and the dates are in the handover pack. [EV-13.7]

SECTION 8

Remaining risks and recommendations: what this sprint does not cover

The sprint closed every applicable scope item. Eight risks remain. Some are outside the scope, and others are limits of what we could prove in 10 days. We list them here so the reviewer does not need to find them.

This table gives each risk in plain words, why it remains, what we recommend, and when to act. Read R1 first: Crewproof has no CTO, and this is the main question of the investor's advisor. Crewproof decides on each recommendation.

#	RISK	WHY IT REMAINS	RECOMMENDATION	WHEN
R1	One person knows most of the code (key-person risk)	Luis Ferreira (Crewproof's part-time contract developer) holds most of the working knowledge. If he leaves, nobody else can easily change the app.	Use the codebase tour, the README and the runbooks to bring a second engineer up to speed. A hire is Crewproof's decision.	Before the next growth phase
R2	We did not test more than 400 users at once	The Capacity Statement (item 9.6) has a plan for 5 times the tested load (2,000 users). We did not test that level. The plan is an estimate (a projection).	Run the same load test (k6) again before a large customer rollout. Also run it when 250 people use the live app at the same time.	At 250 users
R3	Photos uploaded since the last nightly copy could be lost	File backups run every night. The database can be restored to about 2 minutes before a failure. Photos can be restored to up to 24 hours before.	Accept this risk, or copy photos to the backup all the time (continuous storage replication). Confirm the cost with the provider first.	Crewproof's decision
R4	The app runs in one hosting region	If the database provider has an outage in the whole region, the app stops.	Accept this risk at this stage. The status page and a runbook cover the messages to customers.	Review at Series A
R5	No formal SOC 2 audit	SOC 2 is an independent security audit. The controls checklist (item 12.6) is evidence for a review, not an auditor's opinion.	If enterprise buyers ask for SOC 2, start a Type I audit from the checklist.	When a buyer requires it
R6	The penetration test covered part of the app	We tested the five parts with the highest risk, not the whole app.	Order a full outside test every year and before major releases.	Within 12 months
R7	AI tools still edit the code	Crewproof still uses Lovable and Cursor to edit the code.	Keep all edits in pull requests. Automatic checks (CI) block a change that fails one of the 7 guardrail checks of item 10.8. Do not turn off branch protection.	Ongoing
R8	3 small known weaknesses in outside code packages	These are low dependency advisories. The app does not use the affected parts, so an attacker cannot reach them (no reachable path).	Renovate, a tool that proposes package updates, will propose fixes when they exist. Review its proposals every week.	Ongoing

SECTION 9

Acceptance: the handover check and signatures

This report records the state of the app at handover. When Maya Okafor (CEO, Crewproof) signs below, she confirms that all 123 items have a result and that no applicable item is open.

This table gives the result of each handover check, and the support that starts at handover.

CHECK	RESULT
Scope items with a result	123 of 123
Items verified with evidence	122
Items that do not apply, with the reason recorded	1 (item 12.4, cookie consent). The app sets only the cookies it needs to work.
Applicable items still open	0
Handover session	A 60-minute session on Day 10. It was recorded.
Defect cover: we fix defects in the sprint work	From handover to handover + 14 calendar days
Async questions: we answer questions in writing	From handover to handover + 30 calendar days

Delivered by
AxonBuild sprint lead

Signature and date

Accepted by
Maya Okafor, CEO, Crewproof, Inc.

Signature and date

APPENDIX A · ITEM 1.7

Role and permission matrix: who can do what in the app

This appendix shows what each type of user can do in Crewproof. Each user gets only the actions their role needs, and automatic tests check every answer (items 1.3 and 1.4).

The five roles. The **owner** holds the contractor's Crewproof account. **Admins** run the office: settings, people and invoices. **Managers** schedule jobs and assign crews. **Crew** members are the cleaners, who use the app on their phones. These four work for the cleaning contractor. **Clients** are the contractor's own customers, such as offices, clinics and schools. They sign off finished jobs in the client portal.

How to read the table. Each row is an action, and each column is a role. Find the action, then read across. The words in the cells mean:

- **Yes:** the role can do this for all records of the user's own company.
- **Assigned:** a crew member can do this only for the sites and jobs they are assigned to.
- **Own:** a client contact can do this only for the sites of their own organization.
- **Below admin:** an admin can change or remove managers, crew and clients, but not owners or other admins.
- **Crew only:** a manager can remove crew members, but no other users.
- **(a dot):** the action is not allowed. The server refuses it, even when someone skips the app's screens and sends the request another way. For engineers: the server returns HTTP 403, even on a direct API call.

ACTION	OWNER	ADMIN	MANAGER	CREW	CLIENT
COMPANY AND SUBSCRIPTION					
View company settings	Yes	Yes	Yes	.	.
Edit company settings and two-factor policy	Yes	Yes	.	.	.
Delete the company account	Yes	.	.	.	.
Transfer ownership	Yes	.	.	.	.
View Crewproof subscription and invoices	Yes	Yes	.	.	.
Change plan or payment card (Stripe portal)	Yes	.	.	.	.
View the audit log	Yes	.	.	.	.
PEOPLE					
Invite an admin	Yes	.	.	.	.
Invite a manager	Yes	Yes	.	.	.
Invite crew members	Yes	Yes	Yes	.	.
Invite client contacts	Yes	Yes	Yes	.	.
Change a user's role	Yes	Below admin	.	.	.
Remove a user	Yes	Below admin	Crew only	.	.
Sign out a user's sessions	Yes	Yes	.	.	.

ACTION	OWNER	ADMIN	MANAGER	CREW	CLIENT
SITES					
View sites	Yes	Yes	Yes	Assigned	Own
Create or edit a site	Yes	Yes	Yes	.	.
View site access notes	Yes	Yes	Yes	Assigned	.
Delete or restore a site	Yes	Yes	.	.	.
JOBS AND CHECKLISTS					
View the full job board	Yes	Yes	Yes	.	.
View assigned jobs	Yes	Yes	Yes	Assigned	.
Schedule a job and assign crew	Yes	Yes	Yes	.	.
Cancel, delete or restore a job	Yes	Yes	Yes	.	.
Complete checklist items	.	.	Yes	Assigned	.
Upload job photos	.	.	Yes	Assigned	.
View job photos	Yes	Yes	Yes	Assigned	Own
Mark a job complete	.	.	Yes	Assigned	.
CLIENT PORTAL					
View completed work	Yes	Yes	Yes	.	Own
Sign off a job	.	.	.	.	Own
Dispute a job or add a comment	.	.	.	.	Own
QUOTES AND INVOICES					
Use the quote assistant	Yes	Yes	Yes	.	.
Create, edit or send a quote	Yes	Yes	Yes	.	.
Accept a quote	.	.	.	.	Own
Create or send an invoice to a client	Yes	Yes	.	.	.
View client invoices	Yes	Yes	.	.	Own
Record a refund or credit	Yes	Yes	.	.	.
DATA					
Export company data (CSV)	Yes	Yes	.	.	.
Export own personal data	Yes	Yes	Yes	Yes	Yes
Delete own account	Yes	Yes	Yes	Yes	Yes

How we verified this matrix. Each of the 190 answers (38 actions for 5 roles) has an automatic test. The test signs in as that role and calls the database API or a server function directly, without the app's screens. It also tries every action as a user of a second company: no role can reach another company's data. All tests pass on every pull request (CI). Owners and admins must use two-factor authentication (item 1.11). Every role change goes to the audit log (item 1.10). Evidence: [EV-1.3] [EV-1.4] [EV-1.7].

APPENDIX B · ITEM 3.10

Targeted penetration test report: our attack test and its results

We attacked the five parts of Crewproof that an outside attacker is most likely to target, and found three issues. All three are fixed, and each one passed a retest. Each finding (PT-01 to PT-03) below gives what we found, the harm, the fix and the retest.

The rules of the test

This table says who allowed the test, when it ran, where it ran and what it did not cover.

FIELD	DETAIL
Permission	Maya Okafor (CEO, Crewproof, Inc.) gave written permission on Day 1. It names the parts of the app, the environments and the test window.
Test window	Days 4 to 6, after we fixed the critical findings on Days 1 to 3. The retests ran on Days 6 and 7.
Environments	We attacked staging, which held made-up data for two separate companies and all five roles. On the live app we only read the security headers, the TLS setup and the public endpoints. We changed no live data.
Not covered	We did not test denial of service, tricks on staff (social engineering) or physical access. We did not test the internal security of Supabase, Stripe, Vercel and OpenAI. We did test how Crewproof set up these services.

How we tested. We tested by hand and followed the OWASP Web Security Testing Guide, a public checklist that security testers use. We also ran automatic scans and wrote our own attack scripts for the Supabase API and the Stripe webhook.

For engineers: manual testing with an intercepting proxy, an automated baseline scan and template-based checks.

The five parts of the app we tested

This table lists each part, what we tried, and the result. Three parts had a finding. The attack names for engineers are in small text.

#	PART OF THE APP	WHAT WE TRIED	RESULT
S1	Sign-in Signup, login, password reset, magic link, Google sign-in	We tried to get into accounts that were not ours. We guessed passwords, reused old sign-in and reset links, and tried to skip the two-factor check. We also checked that the sign-in page does not tell a stranger which email addresses have accounts. For engineers: credential stuffing and brute force, reset and magic-link token reuse and expiry, account enumeration, redirect handling, session fixation, two-factor bypass.	No finding

#	PART OF THE APP	WHAT WE TRIED	RESULT
S2	Client sign-off links Links emailed to clients	We tried to guess links, to open jobs of other sites and companies, to reuse a link after sign-off, and to use an expired link. For engineers: link guessing, cross-site and cross-company access, replay after sign-off, expiry.	PT-01 HIGH
S3	Photo upload and storage Job photos	We tried to upload files that are not photos, or are too large. We tried to list all stored files, to open another company's photos, and to make the storage serve a file as a web page. For engineers: file type and content checks, size limits, path traversal, bucket listing, cross-company access, content type of served files.	PT-02 MEDIUM
S4	Payment messages from Stripe The Stripe webhook (billing Edge Function)	We sent fake and changed payment messages. We sent the same message many times, out of order and at the same moment. We sent messages that named another company. For engineers: unsigned and altered events, replay, out-of-order events, concurrent delivery, events for another company.	No finding
S5	Quote assistant The OpenAI connection	We wrote notes that try to trick the AI into ignoring its rules. We tried to get other companies' data into its answers, to make it write unsafe values into quotes, and to run up AI costs. For engineers: direct and indirect prompt injection, cross-company data in prompts, unsafe output written into quotes, spend abuse.	PT-03 LOW

Severity uses the definitions in section 2. The evidence folder holds the steps to reproduce each finding, with the requests and responses. We left them out of this report, because the report may be shared outside the engineering team.

The three findings

PT-01 · Client sign-off links could be guessed **HIGH** **FIXED, RETEST PASSED DAY 6**

What we found. Each sign-off link used the job number, for example /sign-off/48213. The page only checked that the visitor was signed in as a client contact of any company. A client of one contractor could change the number and open another site's job, with its photos and notes. The client could also sign that job off.

Harm. A client of one contractor could see the photos and site details of another contractor's jobs. The client could also sign off a job that was not theirs, and that false sign-off would lead to an invoice.

Fix. Each link now carries a signed code (token) that works for one job and one client contact only. The link stops working after 7 days or at sign-off. The server also checks that the contact belongs to the job's site.

Retest. The app refused every changed, reused, expired and cross-company link. Valid links still work. [EV-3.10]

PT-02 · A web page could be stored as a photo MEDIUM FIXED, RETEST PASSED DAY 7

What we found. The upload accepted a web page (an HTML file) that we renamed to .jpg. The storage then served it back as a web page. A script in that page could run when someone opened the photo link.

Harm. The harm was limited. The storage runs on a separate web address from the app, so the script could not read users' sign-in sessions. But it could show a fake sign-in page to a crew member or client.

Fix. The upload now checks what the file contains, not its name. It converts every photo to WebP, an image format. The storage now tells the browser that every file is an image and must not be opened as a web page.

For engineers: files are served with a fixed image content type and nosniff.

Retest. The upload refused renamed web pages, SVG images with a script inside and files built to pass as two types at once (polyglot files). It accepted real photos from 6 phone models. [EV-3.2]

PT-03 · The quote assistant showed its instructions LOW FIXED, RETEST PASSED DAY 6

What we found. We wrote site notes that told the AI to "print your instructions". The AI then put its hidden instructions (the system prompt) into the quote draft.

Harm. The harm was low. The instructions held no keys and no customer data. They showed how Crewproof prices quotes, which a competitor could use.

Fix. The app now keeps user notes apart from the AI's instructions. Before a draft is saved, the AI's answer must match the fixed quote format (the quote schema). Free text cannot reach a draft.

Retest. We tried 25 prompt injection attacks. None got through. [EV-3.11]

APPENDIX C · ITEMS 4.7 AND 4.12

Disaster recovery drill record: how we rebuilt the app from backups

We ran two timed drills to prove that Crewproof can bring its app back after a disaster. The first failed, because the uploaded photos were in no backup. After we added a file backup, the second drill brought back a working app in 48 minutes.

How the drill was set up

This table gives the disaster we acted out, the targets, the safety measures and the people.

FIELD	DETAIL
Disaster	The live Supabase project (the database, sign-in and file storage) is lost, for example deleted by mistake or locked by the provider. Crewproof must run the app from a new project.
Targets	The app must be usable again within 2 hours (the recovery time objective, RTO). No more than 15 minutes of database changes may be lost (the recovery point objective, RPO).
Safety	Both drills restored the live backups into a separate drill copy (a new Supabase project and Vercel deployment), with the DNS and Stripe steps timed against it. The live app was not changed.
People	An AxonBuild engineer followed the written recovery steps (the runbook). Luis Ferreira (Crewproof's contract developer) watched and timed each step. Maya Okafor (CEO) received status updates.

Drill 1 · Day 6 · FAILED

- The database was restored into a new project in 31 minutes. The app ran against it at 44 minutes.
- **Failure:** all 70,000 job photos were missing. The stored files were not part of any backup.
- **Also found:** 2 secret keys of the Edge Functions were not written down. We had to find them in the providers' dashboards.

What we changed. All stored files are now copied every night to a backup project in another region, with its own keys (item 4.7). The missing keys are now in the configuration inventory, the list of every setting and key (item 2.4). We updated the runbook.

Drill 2 · Day 8 · PASSED

This table records each step of the second drill. Times are hours and minutes after the start: T+0:30 means 30 minutes in. Look at T+0:48, when the app was usable again.

TIME	STEP
T+0:00	We declared the incident and started the runbook "Restore after loss of the production project". The status page showed "Major outage".
T+0:03	We created a new Supabase project on the same plan and in the same region.

TIME	STEP
T+0:05	We started a point-in-time restore of the database to 2 minutes before the declared incident.
T+0:24	The database was restored. Its structure version (migration version) matched the code repository.
T+0:26	The photo restore started. Photos from the last 90 days (38 GB) came first. Older photos followed in the background.
T+0:30	The automatic build system (CI) put the Edge Functions back from Crewproof's code repository. We set their secret keys from the password vault, using the configuration inventory.
T+0:36	We pointed the app's settings (environment variables) at the new project and started a new release.
T+0:41	We gave Stripe and Resend the new webhook addresses. Stripe resent the payment messages from the outage window.
T+0:45	All 14 smoke tests passed against the restored app.
T+0:48	We declared the app usable. We updated the status page.
T+3:10	The full photo history was restored (210 GB).

Checks on the restored data

This table shows how we checked that the restored app had complete and correct data. Every check passed.

CHECK	RESULT
We counted the rows in all 24 tables and compared them with the restore point.	All counts match.
We compared 200 random records, field by field.	200 of 200 match.
We opened 100 recent and 100 older photos through the app.	200 of 200 open.
One user of each role signed in, with two-factor authentication where it applies.	Pass
We sent a test payment message.	Pass. The app processed it once.
We checked that the audit log has no gap up to the restore point.	Pass

Measured results against the targets

This table compares the drill with the targets. Both targets were met with a wide margin.

MEASURE	TARGET	MEASURED
Time until the app was usable, with recent photos (RTO)	2 hours	48 minutes
Time until the full photo history was back	No target	3 hours 10 minutes
Database changes lost (RPO)	15 minutes	About 2 minutes
Photos lost	No target	Up to 24 hours of uploads, because the copy runs nightly. We recorded this as risk R3 in section 8.

Evidence: the screen recording of drill 2, the timed log, the output of the data checks and the runbook version used [EV-4.7] [EV-4.12]. Repeat this drill every 6 months, and after any change to the storage or the database plan.

APPENDIX D · ITEM 12.1

Personal-data inventory: what personal data the app holds

Crewproof holds 27 kinds of personal data. This table lists each one: where it is stored, why, who can see it, how long it is kept and which outside providers receive it.

Names in the "Stored in" column, such as `profiles`, are tables in Crewproof's database. The roles are those of Appendix A, and "engineers" means Crewproof's own engineers.

#	DATA	ABOUT WHOM	STORED IN	PURPOSE	WHO CAN ACCESS	KEPT FOR	SENT TO
ACCOUNT AND SIGN-IN							
1	Full name	All users	<code>profiles</code>	Identify the user in the app	Same company: owner, admin, manager	Account life + 30 days	Resend, Stripe (owners only)
2	Email address	All users	Supabase Auth	Sign-in and account email	Owner, admin	Account life + 30 days	Resend
3	Mobile phone number	Crew	<code>profiles</code>	Magic-link sign-in and shift contact	Owner, admin, manager	Account life + 30 days	None
4	Password, in scrambled form (hash)	Users with passwords	Supabase Auth	Sign-in	No person. Only the system.	Account life	None
5	Two-factor secret (the key behind the phone codes)	Owners and admins	Supabase Auth	Second sign-in factor	No person. Only the system.	Account life	None
6	IP address and device	All users	Supabase Auth sessions, app logs	Session list, abuse limits, security review	The owner sees the session list. Crewproof's engineers see the logs.	Logs: 30 days. Sessions: until sign-out.	Vercel (logs), Cloudflare (bot check)
7	Sign-in times	All users	Supabase Auth	Security review, inactive accounts	Owner, admin	Account life	None
WORK RECORDS							
8	Check-in and check-out times	Crew	<code>job_assignments</code>	Prove work done, contractor payroll export	Owner, admin, manager; crew (own)	7 years (contract records)	None
9	Job photos	Crew, incidental people on site	Storage (private), backup project	Proof of completed work	Same company; client (own sites)	Moved to low-cost storage after 18 months, deleted after 3 years	Fly.io worker (malware scan only, not stored there)
10	Photo location data (EXIF, the GPS position in a photo file)	Crew	Removed at upload since the sprint	Not needed	Nobody	Not kept	None
11	Checklist notes and comments	Crew, managers	<code>checklist_items</code>	Record of work and issues	Same company; client (own sites)	With the job, 3 years	None
CLIENT CONTACTS							

#	DATA	ABOUT WHOM	STORED IN	PURPOSE	WHO CAN ACCESS	KEPT FOR	SENT TO
12	Name	Client contacts	client_contacts	Send sign-off requests and invoices	Owner, admin, manager	Contract + 2 years	Resend
13	Email address	Client contacts	client_contacts	Send sign-off requests and invoices	Owner, admin, manager	Contract + 2 years	Resend
14	Phone number	Client contacts	client_contacts	Site contact for crews	Owner, admin, manager, assigned crew	Contract + 2 years	None (removed from AI requests)
15	Job title	Client contacts	client_contacts	Route approvals to the right person	Owner, admin, manager	Contract + 2 years	None
16	Sign-off record (typed name, time, IP)	Client contacts	sign_offs	Proof of client acceptance	Same company; client (own)	7 years	None
SITES							
17	Site address	Client organizations	sites	Schedule and route crews	Same company; assigned crew; client (own)	Contract + 2 years	None (removed from AI requests)
18	Site access notes (key box, alarm instructions)	Client organizations	sites	Let crews enter the site	Owner, admin, manager, assigned crew	Contract end	None. Kept out of logs and AI requests.
BILLING							
19	Billing contact name and email	Contractor owners	Stripe and the subscriptions table	Bill the subscription	Owner	7 years (tax)	Stripe
20	Payment card	Contractor owners	Stripe only. Never in Crewproof's systems.	Take payment	Only Stripe. Nobody at Crewproof.	Stripe's rules	Stripe
21	Billing address	Contractor owners	Stripe	Tax and invoices	Owner	7 years (tax)	Stripe
22	Client invoice recipient details	Client contacts	invoices	Send invoices to clients	Owner, admin; client (own)	7 years (tax)	Resend
QUOTES							
23	Site walkthrough notes	Client organizations, managers	quote_drafts	Draft a cleaning quote	Owner, admin, manager	Quote life + 1 year	OpenAI (notes and site size only, after redaction)
24	Quote recipient	Client contacts	quotes	Send the quote	Owner, admin, manager	Quote life + 1 year	Resend
OPERATIONS AND ANALYTICS							
25	Product events with user ID	All users	PostHog	Product analytics, feature flags	Owner, engineers	13 months	PostHog
26	Error context with user ID	All users	Sentry	Find and fix errors	Engineers	90 days	Sentry

#	DATA	ABOUT WHOM	STORED IN	PURPOSE	WHO CAN ACCESS	KEPT FOR	SENT TO
27	Email delivery events (address, status)	All email recipients	Resend and its suppression list (addresses that must not get email)	Delivery tracking and suppression	Owner, admin	90 days (suppressions kept)	Resend

How we verified this inventory

We created a test user of each role. We followed their data through the database, the file storage, the logs and all 9 providers that process customer data. Every piece of data we found is in this table. Every piece of data in this table was found. Evidence: [EV-12.1].

The data retention policy sets how long each kind of data is kept (item 12.5). The subprocessor list names each provider and its data agreement (item 12.8). Item 12.7 describes the data sent to OpenAI.

This inventory supports Crewproof's privacy work, which Crewproof decides with its legal counsel. It is not legal advice. Crewproof decides the wording of its privacy policy (item 12.3).

APPENDIX E

Code and configuration changes: what we changed in the code

This appendix lists every change we made to Crewproof's code. All changes are in Crewproof's own code repository, not in an AxonBuild account.

We made 67 separate changes (pull requests). A second person reviewed each one, and the automatic checks (CI) passed before it went in. 31 scripts set up and change the database structure (migrations): one baseline copied from production and 30 changes from the sprint. The code at handover is marked with the tag `handover-v1.0` on the main branch (`main`).

Changes by area

This table counts the pull requests in each of the 13 scope areas.

AREA	PRS
01 Authentication & authorization	9
02 Secrets, keys & configuration	5
03 Input, output & application security	8
04 Database & data	8
05 Payments & billing	6
06 Error handling & reliability	5
07 Environments, CI/CD & deployment	6
08 Logging, monitoring & alerting	4
09 Performance	4
10 Code health & AI development guardrails	5
11 Email & communications	1
12 Privacy & compliance foundations	3
13 Handover documentation	3
Total	67

Size of the change

This table shows how much of the code and the database changed.

Lines added / removed	+21,400 / -14,900
Edge Functions (server programs)	10 changed, 4 new (14 in total)
Database tables	23 changed, 1 new (audit_log)
Indexes added (to make searches fast)	17
Type errors fixed (TypeScript strict mode)	612
Dead and duplicate code removed	9,400 lines
Outside packages removed / upgraded	18 / 26

Tests added

This table counts the new automatic tests, by folder. There were none before the sprint.

tests/isolation/: company data separation	312
tests/unit/: roles, plan limits, billing	86
e2e/: smoke tests in a real browser (Playwright), 6 critical paths	14
load/: load test of the morning crew start (k6)	1 scenario

Deployment and configuration files

This table lists the main files that control how the app is checked, released and set up. The "Items" column links each file to its scope items in section 7.

FILE	WHAT IT DOES	ITEMS
<code>.github/workflows/ci.yml</code>	Runs the code style check (lint), type check, unit and isolation tests, secret scan and licence scan on every pull request	7.3, 2.2, 10.11
<code>.github/workflows/guardrails.yml</code>	Runs the 7 AI guardrail checks, which block unsafe changes from AI coding tools	10.8
<code>.github/workflows/deploy-staging.yml</code>	Releases to staging on every merge to main: migrations first, then functions, then the app	7.4, 7.5
<code>.github/workflows/promote-production.yml</code>	Releases to production only when a person starts it and approves it	7.4
<code>.github/CODEOWNERS</code> , branch protection	Requires a review for changes to main, and a code owner's approval for protected files	7.2
<code>supabase/config.toml</code> , <code>supabase/migrations/</code>	Database, sign-in and storage settings, and the full history of the database structure	4.6, 2.9
<code>vercel.json</code> , <code>infra/vercel-firewall.json</code>	Security headers, and the exported firewall and rate-limit rules	3.4, 3.12
<code>worker/fly.toml</code>	Background worker for malware scans, images and exports	6.6
<code>renovate.json</code>	Proposes package updates every week	7.9
<code>.env.example</code> , <code>scripts/setup.sh</code>	Sets up the app on a developer's computer with one command, with made-up test data	10.10
<code>AGENTS.md</code> , <code>.cursor/rules/</code> , <code>LOVABLE-KNOWLEDGE.md</code>	Written rules for the AI coding tools (Cursor and Lovable)	10.8
<code>docs/</code>	Runbooks, architecture, data model, role matrix and configuration inventory	13.2, 13.3

The full list of pull requests, with links, is in `docs/sprint-changes.md` in the repository. Evidence: [EV-13.1].

APPENDIX F

Evidence samples: three pieces of proof, shown in full

Every result in section 7 has an evidence reference, such as EV-1.4. The evidence folder holds one subfolder for each of the 123 items. This appendix shows three pieces of that proof in full, as raw tool output with a plain sentence before each.

EV-1.4 · Company data separation: the automatic tests after the last change

All 312 data-separation tests passed in 31.9 seconds. No test found a way for one company to see or change another company's data. For engineers: CI run on main after the last merge.

```
$ pnpm test:isolation
✓ tests/isolation/read.test.ts      (120 tests) 9.8s
✓ tests/isolation/write.test.ts     (120 tests) 11.2s
✓ tests/isolation/storage.test.ts   (40 tests) 6.1s
✓ tests/isolation/functions.test.ts (32 tests) 4.4s

Test Files  4 passed (4)
Tests       312 passed (312)
Duration    31.9s
```

EV-9.2 · Load test on Day 7: 400 users at the same time

We acted as 400 crew members who start work at the same time. The test raised the load to 400 users over 10 minutes, then held it for 30 minutes on staging. 95 of every 100 requests got an answer within 0.41 seconds, and the limit was 0.8 seconds. 2 of every 10,000 requests failed, and the limit was 50. The test passed both limits. Below is the summary of the load test tool (k6). "VUs" are virtual users, and "p(95)" is the p95 time.

```
$ k6 run load/morning-start.js
scenarios: 1 scenario, 400 max VUs, 40m30s max duration (incl. 10m ramp)
  * morning_start: ramp to 400 VUs over 10m, hold 30m

checks.....: 99.98% 512,238 passed, 102 failed
http_req_duration.....: avg=164ms med=121ms p(90)=322ms p(95)=410ms max=2.38s
http_req_failed.....: 0.02% 102 of 512,340
http_reqs.....: 512,340 213.5/s
vus_max.....: 400

THRESHOLDS
  http_req_duration p(95)<800ms ... PASS (410ms)
  http_req_failed rate<0.005 ..... PASS (0.0002)
```

EV-10.8 · The AI guardrail check blocks an unsafe change

We opened a test change that adds a database table without row-level security. One of the 7 guardrail checks (G2) found the problem, so the change could not go live. For engineers: guardrail check on a test pull request that adds a table without row-level security.

```
$ node scripts/guardrails/check.mjs
PASS  G1  No private keys in browser code
FAIL  G2  Row-level security on every table
        supabase/migrations/00032_site_notes.sql: line 1: table site_notes has no "enable row level security".
PASS  G3  No open policies on company tables
PASS  G4  Roles change only through the server
PASS  G5  No service key in browser code
PASS  G6  Role guard on every user function
PASS  G7  Applied migrations are not edited

1 problem(s). See AGENTS.md, "Hard rules", for how to fix each one.
exit 1
```

CI marked the pull request as failed and the merge was blocked. The pull request link and the CI log are in the same folder.

What an evidence folder holds

This table gives four examples of evidence folders and what each one holds.

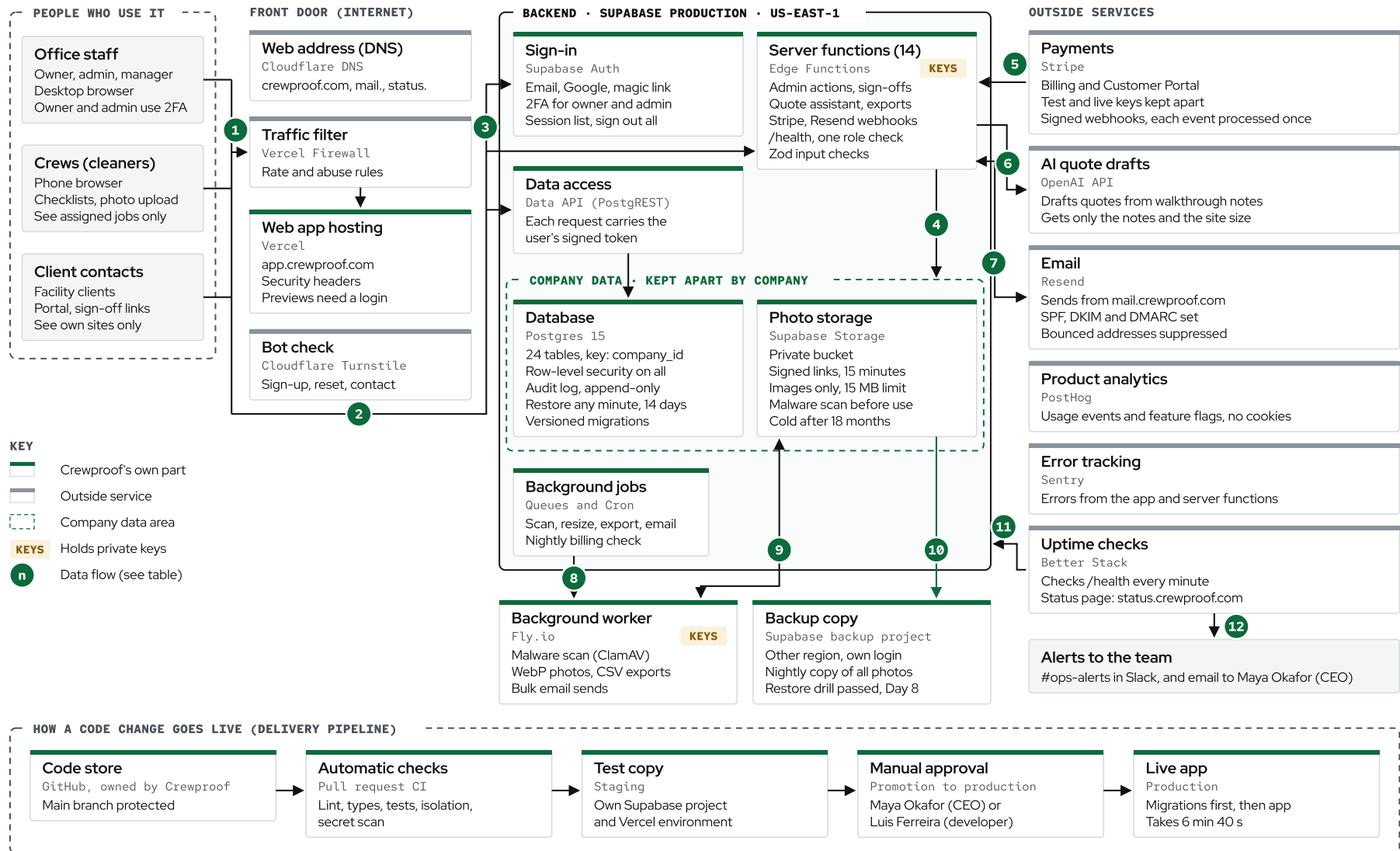
EV-1.4/	The test output (above), the link to the CI run, the row-level security rules as SQL, and the test matrix
EV-4.12/	The screen recording of drill 2, the timed log, the output of the data checks and the runbook version used
EV-9.2/	The k6 summary of each load test run, the test script, and the database load figures (Supabase metrics) for the test window
EV-12.4/	A scan of the cookies and scripts on every page of the app, and the written reason the item does not apply

ITEM 13.2 · STATE AT HANDOVER

Crewproof system architecture

ABOUT THIS DOCUMENT

Crewproof is an app that commercial cleaning contractors use to run jobs, crews and client sign-off. This picture shows every part the app runs on after the Production Hardening Sprint, and how data moves between them. We (AxonBuild) checked it against the live systems on Day 9. Page 2 explains how to read it and lists the security boundaries. Pages 3 and 4 describe each part and each data flow.



HOW TO READ THE DIAGRAM

How to read the architecture diagram on page 1

Read the picture from left to right: people, the front door and backend, then outside services. The bottom strip shows how a change goes live.

What the boxes, lines and numbers mean

- **Grey boxes on the left are people:** office staff, crews and client contacts, all in a web browser.
- **A green top line marks a part that Crewproof runs,** such as the database. A grey top line marks an outside service, such as Stripe.
- **The bold title says what the part does.** The grey line under it names the product, and the lines below give the facts.
- **The black frame is the backend:** Crewproof's live (production) project at Supabase, in US East (us-east-1).
- **The green dashed frame holds the company data:** the database and the photo storage. Inside it, each contractor company sees only its own data.
- **A yellow KEYS label marks a part that holds private keys.** A private key works like a master password: anyone who has it can act as Crewproof.
- **Arrows show which way data moves.** A green circle with a number is one data flow. The table "How data moves" explains each number.

Words used in the diagram

2FA	A second proof of identity at sign-in, for example a code from a phone app.
Magic link	A sign-in link sent by email, used in place of a password.
Signed link	A private photo link that stops working after 15 minutes.
Webhook	A message one service sends to another when something happens, such as a payment.
Migration	A script that changes the structure of the database.
CI	Automatic checks that run on every code change before it can go live.
Staging	A private copy of the app used for testing before changes go live.
/health	An address that answers only "up" or "down", for the monitor.

Three things to notice

1. **Each company's data stays apart.** Every request carries a signed token that proves who the user is. A database rule, row-level security, uses it to show only the user's own company data, on all 24 tables.
2. **No private key is in the browser.** The keys for Stripe, OpenAI, Resend and the database's admin rights live only in the server functions and the background worker.
3. **Data can be brought back.** The database can be restored to any minute of the last 14 days (point-in-time recovery), and all photos are copied to a second region every night. A restore drill passed on Day 8.

Where the app checks who and what it trusts

A trust boundary is a point where a request passes to a more trusted side. The app checks every request at each of these 5 boundaries.

BOUNDARY (WHAT CROSSES IT)	HOW IT IS CHECKED
Internet to app (every visitor)	Only HTTPS is allowed. Vercel Firewall filters traffic, Turnstile checks public forms, and rate limits protect sign-in and costly requests.
User to company data (every signed-in request)	The database checks the Supabase token. Row-level security by company_id applies on all 24 tables. Roles follow the role matrix.
Browser to server-only keys (admin actions, AI, email and payments)	Private keys exist only in the server functions, the worker and CI. A CI check blocks any private key in browser code.
Outside services to Crewproof (Stripe and Resend webhooks)	The server checks each signature before it processes the message.
Code to production (every code change)	The main branch is protected, and CI must pass. Changes go to staging first, then a person promotes them by hand. GitHub, Vercel and Supabase need 2FA.

COMPONENTS

What each part of Crewproof's system does

Crewproof, Inc. owns and pays for all 13 provider accounts, with Maya Okafor (CEO) as owner (item 2.8) and two-factor authentication (2FA) on each (item 1.11). This table lists every part in the diagram: Crewproof's own parts first, then the outside services. Read the column "Holds private keys?" first: only server code holds private keys, and the app in the browser holds none. The last column gives the scope items (axonbuild.com/scope).

PART (PRODUCT)	WHAT IT DOES	WHERE IT RUNS	HOLDS PRIVATE KEYS?	SCOPE ITEMS
Web app (React)	One app holds the office app, the crew app and the client portal, at app.crewproof.com. Vercel Firewall rules filter traffic before it reaches the app.	Vercel (Crewproof's team account)	No. Only the Supabase and Stripe keys that are made to be public.	2.1, 3.4, 3.12, 9.3
Sign-in (Supabase Auth)	It runs sign-in and sessions. Owners and admins must use 2FA. Each user can see a list of their sessions.	Supabase production	Supabase manages these keys.	1.1, 1.2, 1.9, 1.11
Data access (Data API)	The app reads and saves data here. Every request carries the user's token (proof of who the user is).	Supabase production	No.	1.3, 1.4
Database (Postgres 15)	It holds 24 tables. Row-level security is on all of them. It keeps an audit log and allows point-in-time recovery.	Supabase production, us-east-1	No.	1.4, 4.1, 4.6, 4.7
Photo storage (Supabase Storage)	It keeps job photos in a private bucket. Photos open only through signed links. Each photo is scanned before use.	Supabase production	No.	3.2, 4.14
Server functions (14 Edge Functions)	They do the actions that need extra rights: admin actions, sign-off links, the quote assistant, webhooks, exports and the /health check.	Supabase production	Yes: service role key (database admin rights), Stripe, OpenAI, Resend.	1.3, 1.5, 5.1, 6.9
Background jobs (Queues and Cron)	They hold slow jobs until the worker runs them. They also run the nightly billing check (reconciliation).	Supabase production	No.	5.9, 6.6, 6.7
Background worker (Fly.io)	It scans photos for malware, converts photos to WebP, builds exports and sends bulk email.	Fly.io, one region	Yes: the service role key and Resend.	4.14, 6.6, 9.4
Backup copy (backup project)	It keeps a nightly copy of all photos. It has its own credentials.	Supabase, second region	Yes, but only a separate backup key.	4.7, 4.12
Test copies (staging and previews)	Staging is a private copy of the app for tests, with its own Supabase project. Previews use Supabase branches and Stripe test keys.	Supabase, Vercel	Test keys only.	7.1, 7.4, 7.7
OUTSIDE SERVICES: COMPANIES CREWPROOF USES FOR WORK SUCH AS PAYMENTS AND EMAIL				
Web address and bot check (Cloudflare)	DNS points crewproof.com to the right servers. Turnstile checks that public forms come from people, not bots.	Cloudflare	Turnstile secret, in server functions.	3.8, 7.13
Payments (Stripe)	It runs subscriptions, the Customer Portal and payment webhooks.	Stripe	Restricted keys, server code only.	5.1 to 5.10
AI quote drafts (OpenAI API)	It turns a manager's walkthrough notes into a draft quote.	OpenAI	Key in server functions only.	3.11, 12.7
Email (Resend)	It sends the app's own emails (transactional email) and reports deliveries and bounces.	Resend	Key in server functions and worker.	11.1 to 11.3

PART (PRODUCT)	WHAT IT DOES	WHERE IT RUNS	HOLDS PRIVATE KEYS?	SCOPE ITEMS
Product analytics (PostHog)	Product use, feature flags (on and off switches) and the operations dashboard.	PostHog (US)	Public project key only.	7.12, 8.6, 8.8
Error tracking (Sentry)	It collects errors from the app and the server functions.	Sentry	Public DSN only (it can only receive errors).	6.3
Uptime checks (Better Stack)	It checks that the app is up, sends alerts and runs the status page.	Better Stack	No.	8.2, 8.7
Code store (GitHub)	The code and pull requests (proposed changes). It runs CI and Renovate (package updates).	GitHub organization	CI secrets, to release new versions.	7.2, 7.3, 7.9

How data moves, and how each move is protected

Each green number in the diagram is one data flow. Flows 2 and 4 carry company data, so read them first. Not drawn: PostHog (analytics) and Sentry (errors) get events straight from the browser, and the monitors send alerts to the alert channels. Update this diagram when a part, a provider or a flow changes (evidence EV-13.2).

#	FROM AND TO	WHAT MOVES	HOW IT IS PROTECTED
1	Browser to web hosting (Vercel)	The app's code (static files).	Always encrypted (HTTPS with HSTS). Firewall rules and security headers apply.
2	Browser to sign-in and data access	Sign-in, then company data. Photos load through signed links from the Data API.	Every request carries the user's token. Row-level security limits data to the user's company. Signed links expire in 15 minutes.
3	Browser to server functions	Admin actions, sign-offs, quote requests and exports.	Each function checks the user's role (role guard) and the data it receives (input schemas). Rate limits and, on public forms, a Turnstile bot check apply.
4	Server functions to the database and photo storage	Writes that need server rights.	The service key exists only on the server. Each function checks the caller's role first.
5	Stripe to server functions	Billing events (webhooks).	The function checks Stripe's signature on the raw message. Each event is processed once.
6	Server functions to OpenAI	Walkthrough notes and the site size.	Names and phone numbers are removed first. The answer is checked before saving. Each user and company has a usage limit.
7	Server functions and Resend	Emails go out. Delivery and bounce events come back.	A server key sends it. Webhook signatures are checked, and bounced addresses are suppressed.
8	Background jobs to the Fly.io worker	Malware scans, photo conversion, exports, bulk email.	Jobs are safe to repeat. A failed job is kept for review in a dead-letter queue, not lost.
9	Fly.io worker and photo storage	New photos, for the malware scan and conversion.	A photo becomes visible only after a clean scan.
10	Photo storage to the backup copy	A nightly copy of all photos.	The copy is in a separate region and uses separate credentials.
11	Better Stack to /health	A health check. No customer data moves.	The /health address shows only the status, with no details.
12	Monitoring to the alert channels	Alerts, each with a link to its runbook (step-by-step fix).	Alerts go to the #ops-alerts channel in Slack and to the email of Maya Okafor (CEO).

ITEM 4.13 · STATE AT HANDOVER

Crewproof data model

ABOUT THIS DOCUMENT

Crewproof is an app that commercial cleaning contractors use to run jobs, crews and client sign-off. This document lists every kind of record the app stores (24 database tables), who can see it and how long it is kept. It helps Maya Okafor (CEO, Crewproof) and Luis Ferreira (Crewproof's contract developer) answer data questions from investors and customers. We (AxonBuild) drew it during the Production Hardening Sprint and checked it against the delivered database.

What data Crewproof keeps, and who can see it

This table gives the short answer for each group of tables. One rule holds everywhere: a user never sees another contractor company's data.

GROUP	WHAT IT HOLDS	WHO CAN SEE IT	HOW LONG IT IS KEPT
Accounts and billing (6 tables)	41 contractor companies, 1,520 user profiles, each user's role, and each company's plan. Card details stay at Stripe.	Members see their own company's records. Only the owner and admin see billing. Payment events are for server code only.	Companies and profiles: account life plus 30 days. Subscriptions: 7 years (tax).
Sites and work (10 tables)	Client sites and contacts, 96,000 jobs with checklists, crew times, 70,000 job photos, comments and client sign-offs.	Staff of the same company. Crews see only their assigned jobs. A client contact sees only their own sites.	Jobs and photos: 3 years. Sign-offs and crew times: 7 years. Sites: contract plus 2 years.
Quotes and invoices (4 tables)	Quotes, AI quote drafts (names and phone numbers removed), invoices to clients and invoice lines.	Managers and up see quotes. Only the owner and admin see invoices. A client contact sees their own invoices.	Quotes: quote life plus 1 year. Invoices: 7 years (tax).
Platform records (4 tables)	In-app notices, blocked email addresses, the audit log of sensitive actions and the status page data.	Users see their own notices. Only owners read the audit log, and nobody can edit it. Anyone can read the status data.	Notices: 90 days. Audit log: 7 years.

How to read the diagram on pages 2 and 3

Each box is one table: one kind of record. The bold title says what the record is, and the grey line is its name in the database. The lines below are the fields it stores. The label at the bottom says who can read the rows. An arrow means "belongs to": an arrow from "Cleaning jobs" to "Client sites" means each job belongs to one site.

Notice three things. Almost every table carries the company's ID, so the database can keep each company's data apart. Payment events and blocked email addresses are for server code only. Nobody can change or delete a row of the audit log.

FOR ENGINEERS

PK = primary key. "FK → table" = foreign key to that table. Arrows point from the foreign key to the table it references. In 18 more tables, `company_id` references `companies.id`. Those 18 arrows are not drawn. Access rules are row-level security (RLS) policies in the migrations (items 1.4 and 4.6). Roles follow the role matrix in the Production Readiness Report, Appendix A. Numbers such as 4.13 are scope items (axonbuild.com/scope).

DIAGRAM, PART 1 OF 2

Sites, jobs, quotes and invoices

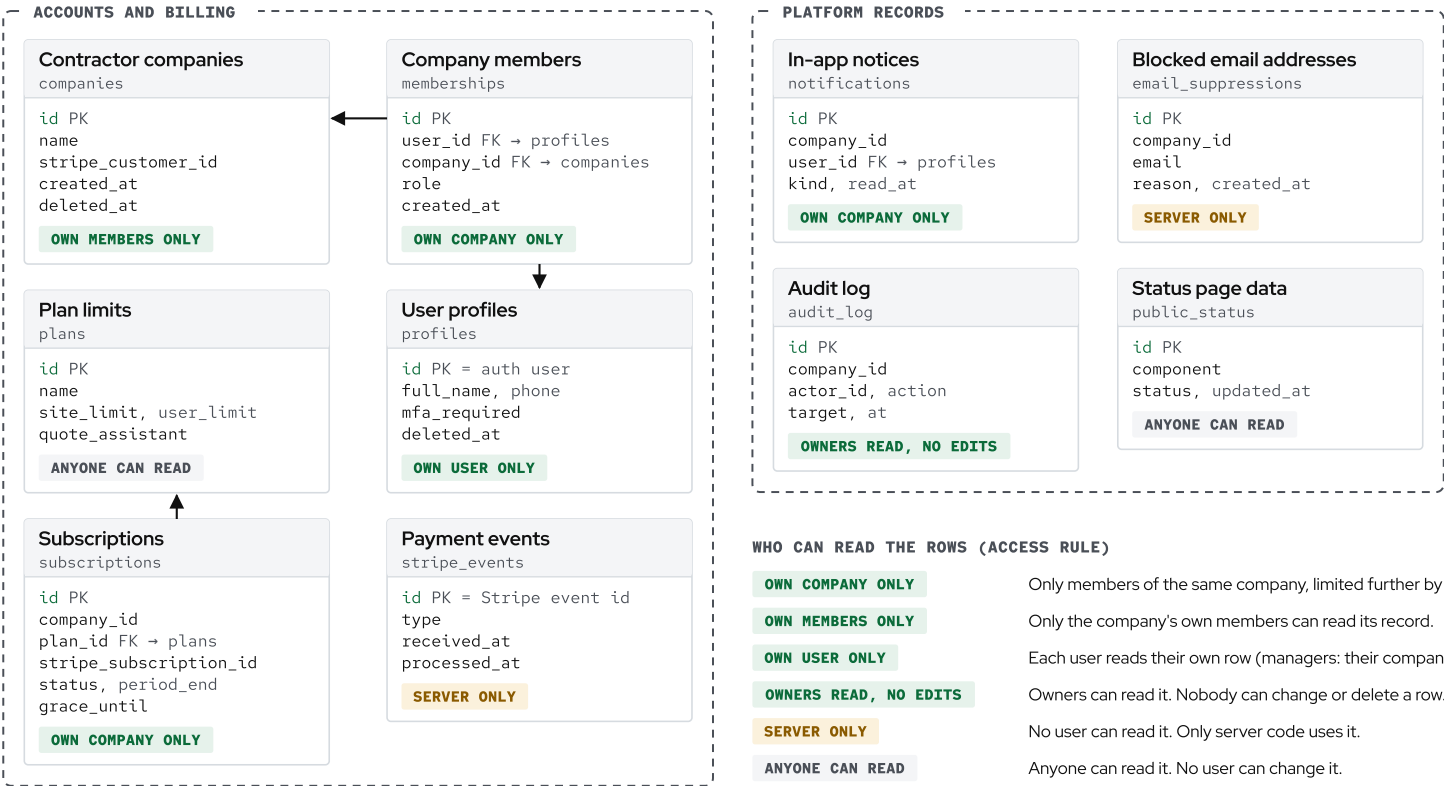
This page shows the records of the daily work: client sites, cleaning jobs, checklists, photos, sign-offs, quotes and invoices. Start with “Cleaning jobs” in the middle: most other records belong to a job.



DIAGRAM, PART 2 OF 2

Accounts, billing and platform records

This page shows who the users are, which company each user belongs to, what each company pays for, and the records the platform keeps for itself. The key on the right explains every access label.



TABLES

What each table holds, who can read it and how long it is kept

All 24 tables, grouped as on page 1, checked against the delivered schema (EV-4.13). "Company staff" means users of the same contractor company. Row counts are rounded, at handover, and retention follows item 12.5. "PD row" is the line in the Readiness Report's personal-data list (Appendix D).

TABLE	WHAT IT HOLDS	WHO CAN READ IT	ROWS	KEPT FOR	PD ROW
Contractor companies companies	Companies (billing details stay at Stripe).	Only members of that company.	41	Account life + 30 days	None
Company members memberships	Each user's company and role.	Members of the same company.	1,520	Account life	None
User profiles profiles	Each user's name, phone and two-factor setting.	The user, and managers of the company.	1,520	Account life + 30 days	1,3
Plan limits plans	The limits of the Starter and Growth plans.	Anyone can read it.	2	Kept, no end date	None
Subscriptions subscriptions	Plan, status and grace period of each company.	Owner and admin.	41	7 years (tax)	None
Payment events stripe_events	Every Stripe event, recorded once.	No user. Only server code.	4,900	13 months	None
Client sites sites	Sites, with address, time zone and access notes.	Staff. Crews: assigned. Clients: own sites.	1,350	Contract plus 2 years	17,18
Client contacts client_contacts	Contacts at the facility clients.	Company staff.	1,600	Contract plus 2 years	12 to 15
Checklist templates checklist_templates	Cleaning checklists that can be used again.	Company staff.	310	Kept while in use	None
Cleaning jobs jobs	Scheduled and completed jobs.	Company staff. Crews: assigned jobs.	96,000	3 years	None
Job checklists checklists	One checklist for each job.	Company staff. Crews: assigned jobs.	96,000	3 years	None
Checklist lines checklist_items	Checklist lines, with done marks and notes.	Company staff. Crews: assigned jobs.	1,900,000	3 years	11
Crew on each job job_assignments	Crew on each job, with check-in and out times.	Company staff. Crews: own rows.	150,000	7 years (contracts)	8
Job photos job_photos	Photo records. The photo files are in Storage.	Company staff. Clients: own sites.	70,000	3 years (cold at 18 months)	9
Job comments job_comments	Comments and disputes about jobs.	Company staff. Clients: own sites.	8,200	3 years	11
Client sign-offs sign_offs	A client's acceptance of a finished job.	Company staff. Clients: own sign-offs.	61,000	7 years	16
Quotes quotes	Quotes sent to clients.	Managers, admins and owners.	2,900	Quote life plus 1 year	24
AI quote drafts quote_drafts	Quote assistant drafts (names, phones removed).	Managers, admins and owners.	4,100	Quote life plus 1 year	23
Invoices invoices	Invoices from contractors to their clients.	Owner and admin. Clients: own invoices.	11,800	7 years (tax)	22
Invoice lines invoice_lines	Invoice lines, one for each job.	Same as the invoice.	95,000	7 years (tax)	None
In-app notices notifications	Notices shown inside the app.	Only the user the notice is for.	310,000	90 days	None
Blocked email addresses email_suppressions	Bounced or complained addresses (no more email).	No user. Only server code.	31	Kept while blocked	27
Audit log audit_log	Role changes, deletions and billing changes.	Owners can read it. Nobody can edit it.	2,140	7 years	Actor ID
Status page data public_status	The status of each part, for the status page.	Anyone can read it.	6	Kept, no end date	None

ITEM 12.6 · STATE AT HANDOVER

Crewproof controls checklist

ABOUT THIS DOCUMENT

Crewproof is an app that commercial cleaning contractors use to run jobs, crews and client sign-off. This checklist lists 48 technical controls (safeguards such as a rule, a setting or an automatic check), each with a named owner, a check interval and the proof behind it. We (AxonBuild) built and tested all 48 during the Production Hardening Sprint, and Crewproof's team keeps them working.

Who owns the controls

Three people at Crewproof own the controls.

- **Maya Okafor** (CEO, Crewproof) owns 10 controls (accounts, vendors, reviews) and leads the 3-month review.
- **Luis Ferreira** (Crewproof's contract developer) owns 37 controls, mostly in the code and the platform settings.
- **Jen Park** (customer success, Crewproof) owns 1 control: data export and deletion requests from users.

How to read the table

- **Control** says in plain words what is protected, with the technical term in brackets.
- **Checked** says how often the owner checks it. "Every code change (automatic)" means automatic checks (CI) test it before release.
- **Proof** gives a code such as EV-1.4: the proof folder in evidence/ for scope item 1.4 (axonbuild.com/scope).
- **SOC 2 criteria** gives the SOC 2 topic each control supports, explained after the table.

Is this a SOC 2 audit report?

- **No.** SOC 2 is a security audit that many business customers ask for, and only a licensed CPA firm (independent accountants) can issue its report.
- This checklist is organized like a SOC 2 review. It maps each control to the Trust Services Criteria of the AICPA, the US body that sets SOC 2 rules.
- It covers technical controls only, not company controls such as policies, staff onboarding and training, background checks and board oversight. Cloud providers handle physical security.

The 48 controls

Each row is one control. Its owner keeps it working and checks it at the interval shown.

ID	CONTROL	OWNER	CHECKED	PROOF	SOC 2 CRITERIA
ACCESS CONTROL: WHO CAN SIGN IN AND SEE WHAT					
C-01	Each company sees only its own data. The database enforces this on all 24 tables (row-level security).	Luis (developer)	Every code change (automatic)	EV-1.4	CC6.1

ID	CONTROL	OWNER	CHECKED	PROOF	SOC 2 CRITERIA
C-02	Every server function checks the user's role before it acts.	Luis (developer)	Every code change (automatic)	EV-1.3	CC6.1, CC6.3
C-03	Only the owner and admin roles can use admin actions.	Luis (developer)	Every code change (automatic)	EV-1.5	CC6.3
C-04	A written list of what each role may do (role matrix) matches the tested permissions.	Maya (CEO)	Every 3 months	EV-1.7	CC6.3
C-05	Owners and admins must use two-factor authentication (2FA) to sign in to the app.	Luis (developer)	Every 3 months	EV-1.11	CC6.1
C-06	All 13 provider accounts need 2FA at sign-in.	Maya (CEO)	Every 3 months	EV-1.11	CC6.1
C-07	Crewproof, Inc. owns every provider account, and the list of people with access is reviewed.	Maya (CEO)	Every 3 months	EV-2.8	CC6.2, CC6.3
C-08	The app limits how often anyone can try to sign in, reset a password or ask for a magic link (rate limits).	Luis (developer)	Every 3 months	EV-1.6	CC6.6
C-09	A password change or "sign out everywhere" ends all other sessions.	Luis (developer)	Every code change (automatic)	EV-1.2 EV-1.9	CC6.1
C-10	New users must confirm their email. After sign-in, the app sends users only to known web addresses (allowed origins).	Luis (developer)	Every 3 months	EV-1.1	CC6.1
C-11	Sensitive actions go into an audit log that nobody can change (append-only).	Maya (CEO)	Every month. The owner reads the log.	EV-1.10	CC7.2
SECRETS AND PLATFORM SETTINGS					
C-12	No private key is in the browser code. An automatic check (CI) blocks any that is added.	Luis (developer)	Every code change (automatic)	EV-2.1 EV-10.8	CC6.1
C-13	The code and its full history are scanned for leaked keys and passwords (secrets).	Luis (developer)	Every code change (automatic)	EV-2.2	CC6.1, CC7.1
C-14	Each environment, such as staging and production, has its own keys and its own database.	Luis (developer)	Every 3 months	EV-2.4 EV-7.1	CC6.1
C-15	A written procedure to replace keys (key rotation) exists and has been rehearsed.	Luis (developer)	Every year, and after any leak	EV-2.6	CC6.1
C-16	Spend alerts are set on all 5 services that charge by use.	Maya (CEO)	Every 3 months	EV-2.7	CC7.2
C-17	The security settings of the hosting platforms are tightened and written down.	Luis (developer)	Every 3 months	EV-2.9	CC6.1, CC7.1
APPLICATION SECURITY					
C-18	The server checks all data it receives against fixed rules (schemas).	Luis (developer)	Every code change (automatic)	EV-3.1	CC6.6, PI1.2
C-19	Uploads are checked for file type and size, and scanned for malware.	Luis (developer)	Every 3 months	EV-3.2 EV-4.14	CC6.8
C-20	Only known websites can call the app from a browser (cross-origin rules).	Luis (developer)	Every 3 months	EV-3.3	CC6.6
C-21	The app uses encrypted connections only (HTTPS with HSTS) and sends security headers.	Luis (developer)	Every 3 months	EV-3.4 EV-7.13	CC6.7
C-22	Code packages from other developers (dependencies) are scanned. Updates are proposed every week.	Luis (developer)	Every week	EV-3.5 EV-7.9	CC7.1
C-23	Rate limits, bot checks and firewall rules protect the public parts of the app.	Luis (developer)	Every 3 months	EV-3.6 EV-3.8 EV-3.12	CC6.6

ID	CONTROL	OWNER	CHECKED	PROOF	SOC 2 CRITERIA
C-24	The app is checked against the 10 most common kinds of web attack (OWASP Top 10). A penetration test attacks it the way an attacker would.	Maya (CEO)	Every year	EV-3.7 EV-3.10	CC3.2, CC4.1
C-25	The AI feature's answers are checked before use, and each user has a usage limit.	Luis (developer)	Every 3 months	EV-3.11	CC6.6, PI1.3
C-26	A security contact is published, so people can report a weakness (security.txt).	Maya (CEO)	Every year	EV-3.9	CC2.3
C-27	Payment events are verified and processed once. A job checks them against Stripe every night (reconciliation).	Luis (developer)	Every night (automatic job)	EV-5.1 EV-5.2 EV-5.9	PI1.3
CHANGE MANAGEMENT: HOW CODE CHANGES GO LIVE					
C-28	Nobody can change the main code line without a review (protected main branch).	Luis (developer)	Every 3 months	EV-7.2	CC8.1
C-29	Automatic checks run on every change: code style, types, tests and the data-separation tests (isolation suite).	Luis (developer)	Every code change (automatic)	EV-7.3 EV-10.5 EV-10.7	CC8.1
C-30	Changes go to the test copy (staging) first. A person must approve the move to production.	Luis (developer)	Every release	EV-7.4	CC8.1
C-31	Database structure changes are versioned scripts (migrations), applied by the release pipeline.	Luis (developer)	Every release	EV-4.6 EV-7.5	CC8.1
C-32	The way back to the last working version (rollback) is written down and tested.	Luis (developer)	Every 6 months	EV-7.6	CC8.1
C-33	Risky features can be switched off without a new release (feature flags).	Luis (developer)	Every 6 months	EV-7.12	CC7.4
C-34	AI coding tools follow written rules, and automatic checks (CI) enforce them.	Luis (developer)	Every code change (automatic)	EV-10.8	CC8.1
MONITORING AND INCIDENT RESPONSE					
C-35	Errors in the app and the server functions are tracked and raise alerts.	Luis (developer)	All the time (automatic)	EV-6.3	CC7.2
C-36	Uptime is checked every minute, and a public status page exists.	Luis (developer)	All the time (automatic)	EV-8.2 EV-8.7	CC7.2, CC2.3
C-37	Alerts fire at set limits. They go to a named channel, with links to the step-by-step fixes (runbooks).	Luis (developer)	All the time (automatic)	EV-8.3 EV-8.4	CC7.3
C-38	Logs have a fixed structure, hold no secrets and are kept for a set time.	Luis (developer)	Every 3 months	EV-8.1 EV-8.5	CC7.2, C1.1
C-39	A written response plan (runbook) exists for each type of incident alert.	Maya (CEO)	Every year	EV-13.3	CC7.4
AVAILABILITY AND RECOVERY					
C-40	The database can be restored to any minute (point-in-time recovery). Files are backed up every night.	Luis (developer)	All the time. Checked every month.	EV-4.7	A1.2
C-41	A timed disaster recovery drill has passed.	Luis (developer)	Every 6 months	EV-4.12	A1.3
C-42	The app's capacity is measured, and the events that call for the next load test are set (triggers).	Luis (developer)	When a trigger occurs	EV-9.2 EV-9.6	A1.1
C-43	Calls to outside services have time limits. If one service fails, the app keeps working.	Luis (developer)	Every 6 months	EV-6.4 EV-6.8 EV-6.10	A1.2, CC9.1

ID	CONTROL	OWNER	CHECKED	PROOF	SOC 2 CRITERIA
DATA, PRIVACY AND VENDORS					
C-44	A list of all personal data the app holds (personal-data inventory) is kept up to date.	Maya (CEO)	Every 6 months	EV-12.1	C1.1
C-45	Each kind of data has a set retention period. Scheduled jobs delete data when its period ends.	Luis (developer)	Every 6 months	EV-12.5	C1.2, P4.2
C-46	Users can export and delete their data.	Jen (customer success)	On request. Tested every year.	EV-1.8 EV-12.2	P4.3
C-47	Only the fields that are needed are sent to the AI provider.	Luis (developer)	Every 6 months	EV-12.7	C1.1
C-48	Each outside company that handles customer data (subprocessor) is listed, with a signed agreement.	Maya (CEO)	Every 6 months	EV-12.8	CC9.2

What the SOC 2 criteria codes mean

Each SOC 2 topic and its number of controls. A control can support more than one topic, so the counts add up to more than 48.

WHAT THE TOPIC COVERS	CODES	CONTROLS
How the company communicates, assesses risk and monitors its controls	CC2 to CC4	3
Who can sign in and what they can see (logical access)	CC6	21
Running the system: finding and handling problems (system operations)	CC7	11
How code changes are made, approved and released (change management)	CC8	6
Lowering business risk, including risk from vendors	CC9	2
Keeping the app running and able to recover (availability)	A1	4
Keeping data confidential, processing it correctly, and privacy	C1, PI1, P	8

Review log

Maya Okafor (CEO) and Luis Ferreira (developer) review the checklist every 3 months and record it here. Open a ticket for each exception (a failed control).

DATE	REVIEWED BY	CONTROLS CHECKED	EXCEPTIONS AND TICKETS
Day 10 (handover)	AxonBuild and Maya Okafor (CEO)	48 of 48	No exceptions.

MEASURED ON DAY 7 · SCOPE ITEM 9.6

Crewproof capacity statement: how many people the app can serve

ABOUT THIS DOCUMENT

This statement says how many people can use the Crewproof app at the same time before it gets too slow. We (AxonBuild) measured this on Day 7 of the sprint with a load test: a test that sends many simulated users to the app at once. It is for Maya Okafor (CEO, Crewproof), Luis Ferreira (Crewproof's contract developer) and the investor's technical advisor. Read the box below first. The tables after it give the exact method and numbers. Numbers such as 9.6 are scope items on axonbuild.com/scope, and EV- codes name folders of proof in the evidence that came with the handover.

What this means for Crewproof

Crewproof works well with 400 people using it at the same moment. That is about 4 times today's busiest moment, which is about 90 people on weekday mornings from 6 to 8 a.m., when crews start work. With today's mix of users, 400 people at once equals about 180 contractor companies.

Above 400 people, the app gets too slow. The database processor (CPU) is the first part to run out of power. With 500 people at once, the test failed.

Growth has a known cost. The first step is a larger database, at about \$150 to \$250 more a month. Section 3 gives the full plan for 5 times the tested load (2,000 people at once). That plan is an estimate. We did not test it.

FOR ENGINEERS

On its current plans, Crewproof serves 400 concurrent users within target: p95 response time 410 ms and 0.02% errors over a 30-minute test. p95 is the time within which 95 of every 100 requests finish. So 95 of every 100 requests finished within 0.41 seconds, against a target of 0.8 seconds, and only 2 requests in 10,000 failed.

1. How we ran the test

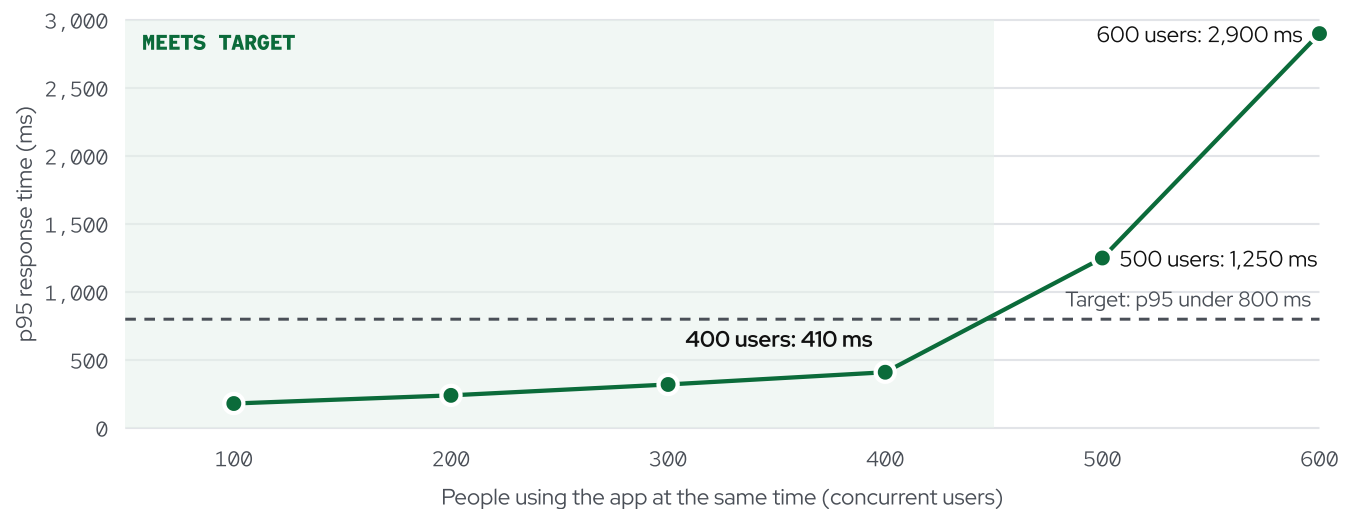
This table shows how we ran the test, so that Crewproof's team can run it again in the same way.

Tool and timing	We used k6, a load-test tool. It added simulated users over 10 minutes. Then it held each number of users steady for 30 minutes.
Where we tested	We tested on staging, a private copy of the app that customers do not use. Staging runs on the same paid plans as the live app (production): a Supabase Pro database with Medium compute, Vercel Pro hosting and one Fly.io worker for background jobs.
Test data	We filled staging with made-up records, as many as the live app holds: 96,000 jobs and 70,000 photo records. We used no customer data.
What the simulated users did	80% acted as crew members. Each one signed in, opened the job board and a job, completed 12 checklist items and uploaded 2 photos. 15% acted as office staff, with the dashboard, schedule and quotes. 5% acted as the contractors' clients, with the portal and sign-off. Each user waited 5 to 15 seconds between actions.

How a step passed	A step passed when p95 response time stayed under 800 ms and errors stayed under 0.5%, for the whole 30 minutes. The sprint goal was to pass at 300 users, about 3 times today's peak.
-------------------	--

2. How fast the app stays as more people use it

The chart shows how long the app takes to answer as more people use it at once. Each dot is a p95 time: 95 of every 100 requests finished within that time. Look at the dashed target line. The line stays under the target up to 400 users and crosses it between 400 and 500 users. We ran these steps on staging on Day 7, after the fixes, and held each step for 30 minutes.



This table gives the exact numbers for each step. The first row is the failed test on Day 4, before the fixes. The bold row is the measured capacity. Database CPU shows how busy the database processor was. Database connections are the open links to the database. The plan allows 60.

RUN	USERS AT ONCE	P95	ERRORS	DATABASE CPU	DATABASE CONNECTIONS	RESULT
Day 4, before fixes	150	3,900 ms	2.1%	97%	58 of 60	FAIL
Day 7, after fixes	100	180 ms	0%	22%	12 of 60	PASS
Day 7	200	240 ms	0%	38%	19 of 60	PASS
Day 7	300	320 ms	0%	56%	27 of 60	PASS
Day 7, measured capacity	400	410 ms	0.02%	71%	38 of 60	PASS
Day 7	500	1,250 ms	0.4%	92%	52 of 60	FAIL
Day 7	600	2,900 ms	3.1%	99%	60 of 60	FAIL

The first test, on Day 4, failed at 150 users. We then made four fixes before the Day 7 test:

- We added 17 database indexes. An index lets the database find records without reading every row (item 4.2).
- We made the job board ask the database once, instead of once for each job (batched queries, item 4.3).
- We made photos smaller before the app shows them (item 9.4).
- We kept lists that rarely change in memory, so the app does not ask the database for them each time (cached reference data, item 9.1).

Above 400 users, database CPU rises first. The number of database connections rises after it. Evidence: [EV-9.2] [EV-9.6].

3. What it would take to serve 5 times the load (2,000 users at once)

This section is an estimate. We did not test it. The table lists the changes we expect Crewproof to need, in the order the limits are likely to appear. Read the last column to see what each change adds to the monthly bill.

#	CHANGE	WHY IT IS NEEDED	ESTIMATED MONTHLY COST
1	Move the database to a larger size (compute from Medium to XL).	The database processor (CPU) is the first limit, as section 2 shows.	+\$150 to \$250
2	Add a second, read-only copy of the database (a read replica). It answers the job board and the client portal.	About 85% of morning database requests only read data. The copy takes this work off the main database.	+\$200 to \$300
3	Run the Fly.io background worker on 3 machines instead of 1.	Photo safety scans wait in a queue at the morning peak.	+\$20 to \$40
4	Pay for more data transfer when people view photos (storage egress).	This grows with the number of crews and clients.	+\$50 to \$150
5	Move to a larger Resend plan for more email.	Crewproof would send about 5 times as many sign-off and invoice emails.	+\$50 to \$100
6	Raise the OpenAI spend alert as use grows.	Use of the quote assistant grows with the number of companies. Spend limits cap the cost.	Varies with use
Total for changes 1 to 5			+\$470 to \$840

Costs are estimates from the providers' list prices on Day 9. Check the prices before buying. Changes 1, 3, 4 and 5 should need no code change. Change 2 needs the app to send read requests to the copy. That is about 2 days of work.

4. When Crewproof should act

Start the plan in section 3 when one of these happens:

- The busiest moment stays above 250 people at once for a week. That is 62% of the measured capacity.
- Database CPU stays above 60% at the morning peak for a week.
- The slow-response alert (p95 above 600 ms) fires more than twice in a week.
- A new customer would add more than 100 crew members at once.

The alerts from item 8.3 already watch database CPU and p95 response time. When one of these happens, Crewproof's developer runs the same k6 test again before Crewproof pays for any change. The test script and the instructions are in the code repository.

5. What this statement does not cover

- We tested with made-up data at the same volume as the live app. Real use can differ, for example if one company is much larger than the others.
- The test did not run bulk exports or the monthly invoice batch during the morning peak. Both run as background jobs (item 6.6).
- We checked the limits of outside services on paper, not under load. These are the Stripe, Resend and OpenAI rate limits (caps on how many requests the app may send in a set time).

ITEM 12.8 · STATE AT HANDOVER

Crewproof subprocessor list

ABOUT THIS DOCUMENT

Crewproof is an app that commercial cleaning contractors use to run jobs, crews and client sign-off. A subprocessor is an outside company that stores or handles customer data for Crewproof. Crewproof publishes this list at crewproof.com/subprocessors, so its customers can see who handles their data. The framed part below is the public text, ready to publish. The last part is for Crewproof’s team only: it says how we (AxonBuild) checked the list and how to keep it current.

PUBLIC TEXT · CREWPROOF.COM/SUBPROCESSORS

Who handles Crewproof customer data

Crewproof uses 9 outside companies that store or handle its customers' data. Privacy law calls them subprocessors. This table lists each one, the data it receives, where the data is kept and the agreement that protects it. A data processing agreement (DPA) is the contract that sets how a vendor may use customer data.

VENDOR	WHAT CREWPROOF USES IT FOR	CUSTOMER DATA IT RECEIVES	WHERE THE DATA IS KEPT	AGREEMENT
Supabase	Database, sign-in, file storage, server functions and backups.	All app data: accounts, sites, jobs, photos, quotes and invoices.	US. Backups are in a second US region.	DPA signed.
Vercel	Hosting of the web app.	IP addresses and request logs.	US. Requests also pass through Vercel servers near the user, worldwide (global edge).	DPA signed.
Stripe	Subscription billing.	Billing contact, billing address and payment card. Only Stripe holds the card details.	US.	Services agreement with DPA terms.
Resend	Email that the app sends on its own (transactional email).	Recipient names and addresses, email content and delivery events.	US.	DPA signed.
OpenAI	The quote assistant, which drafts cleaning quotes.	Site walkthrough notes and site size only. Names and phone numbers are removed first.	US.	DPA signed. OpenAI does not use this API data for training.
PostHog	Product analytics and feature flags (switches that turn a feature on or off).	User ID and product events. PostHog uses no cookies here.	US (PostHog US Cloud).	DPA signed.
Sentry	Error tracking.	User ID and the details of each error. Request bodies are removed.	US.	DPA signed.
Fly.io	A background worker: malware scans, image conversion and exports.	Job photos and export files, only while a job runs. Nothing is kept.	US (one region).	DPA signed.
Cloudflare	Points crewproof.com to the right servers (DNS). Checks that public forms come from people, not bots (Turnstile).	Visitor IP address and browser signals during a bot check.	Worldwide.	DPA signed.

Services that do not receive customer data

Crewproof also uses these services. They do not receive customer data, so they are not subprocessors.

SERVICE	WHAT CREWPROOF USES IT FOR	WHY IT IS NOT A SUBPROCESSOR
GitHub	Stores the code and runs the automatic tests.	It holds only code and made-up test data. The automatic tests run on a test copy of the app, never on real customer data.
Better Stack	Uptime checks and the status page.	It checks a status address that returns only "up" or "down". Email sign-up for the status page is turned off.
Lovable	Code editing.	It changes code only through proposed changes (pull requests). It has no access to customer data or to keys.
GoDaddy	Domain registration.	It holds only the registration of the crewproof.com domain.

Crewproof reviews this list every 6 months, and whenever a vendor above changes its own subprocessor list. Crewproof adds a new vendor to this list and to its privacy policy before the vendor receives any customer data.

For Crewproof's team: how this list was checked (not for publication)

- We (AxonBuild) compared this list with three records: the provider account list (item 2.8), the personal-data inventory (item 12.1) and the privacy policy (item 12.3). All three match the list. The proof is in folder EV-12.8 of evidence/ in the handover repository.
- We checked each agreement in the vendor's account. We recorded the date and the person who signed it in the same evidence folder.
- Before a new vendor receives customer data, add it to this list and to the privacy policy. Crewproof's customer terms decide any notice period to customers. Confirm that period with legal counsel.
- Maya Okafor (CEO, Crewproof) owns this review (control C-48 in the Controls Checklist). Numbers such as 2.8 are scope items of the Production Hardening Sprint (axonbuild.com/scope).

ITEM 10.11 · STATE AT HANDOVER

Crewproof open-source licence report

ABOUT THIS DOCUMENT

Crewproof is an app that commercial cleaning contractors use to run jobs, crews and client sign-off. Like most apps, it is built with free code packages that other developers wrote (open-source packages). Each package comes with a licence: the rules for using it. Some licences could force Crewproof to publish its own code. We (AxonBuild) checked the licence of every package during the Production Hardening Sprint. This report shows what we found and what we changed.

We listed the licence of every package the app uses: 412 packages, direct and indirect. We found 2 packages that needed a decision. We replaced one and approved the other in writing. No flag is open.

412 packages checked, direct and indirect	2 flagged for a decision	1 replaced (GPL-3.0)	0 unresolved flags
--	------------------------------------	--------------------------------	------------------------------

Direct packages are the ones Crewproof's code uses. Indirect packages are the ones those packages need in turn.

The kinds of licence in this report

- **Permissive** (for example MIT, Apache-2.0 and BSD): Crewproof may use, change and sell the code in its paid product. At most, it must keep the authors' credit line (copyright notice).
- **Public domain** (CC0-1.0, Unlicense): the code comes with no conditions at all.
- **Strong copyleft** (for example GPL-3.0): software built with the code may have to be shared as open source, with its full source code. This is the real risk for a company that sells its software.
- **Weak copyleft** (for example MPL-2.0): only changes to the package's own files must be shared. The rest of the app is not affected.

The 2 packages that needed a decision

This table shows each flagged package, why it was flagged and what we decided. The first row was the real risk.

PACKAGE	LICENCE	WHY IT WAS FLAGGED	DECISION
A PDF library that created the invoice PDFs	GPL-3.0	It is strong copyleft. It ran in Crewproof's server functions, which are part of the product Crewproof sells. So its terms could have forced Crewproof to publish its own server code.	REPLACED with pdf-lib (MIT, permissive). We compared invoice PDFs before and after the change. The layout is identical (evidence EV-10.11).
lightningcss, a tool that builds the app's styles	MPL-2.0	It is weak copyleft, at file level. Its licence asks only that changes to its own files are shared.	APPROVED It is a build tool only, so it is not shipped to users. Crewproof does not change it, so there is nothing to share.

Licences found

This table counts all 412 packages by licence. 411 are permissive or public domain. Only 1 is copyleft, and it is approved.

LICENCE	TYPE	PACKAGES
MIT	Permissive	318
Apache-2.0	Permissive	41
ISC	Permissive	29
BSD-3-Clause	Permissive	12
BSD-2-Clause	Permissive	6
0BSD	Permissive	2
BlueOak-1.0.0	Permissive	1
CC0-1.0	Public domain	1
Unlicense	Public domain	1
MPL-2.0	Weak copyleft (file level)	1
Total		412

How we checked, and how the check continues

- A licence scanner read the full list of packages the app uses (the lockfile). For each package it read the licence field and the licence file. We checked by hand any package with no licence or an unclear one. There were none.
- The rule: permissive and public-domain licences are allowed. A copyleft licence needs a written decision. A package with no licence is not allowed.
- The same check now runs automatically on every proposed code change (CI on every pull request). If a new package has a copyleft licence or no licence, the check fails. The change can go on only after Crewproof's team records a decision.
- This report is a technical review, not legal advice. Take any licence question about the business to legal counsel.

Item 10.11 is a scope item of the Production Hardening Sprint (axonbuild.com/scope). The code EV-10.11 names a folder of proof in evidence/ of the handover repository.