ILLUSTRATIVE SAMPLE · A FICTIONAL SYSTEM, NOT A CLIENT'S

Architecture review and handover check: the marketplace

Prepared by John Karanja · fixed scope · Sample, November 2026

One product: a marketplace where buyers book and pay for services from independent providers. One developer built it in about five months, working with AI coding agents. Paying customers use it now. The review covers how it is built, who holds what, what breaks first, and a plan for the first 30 days. No code was changed during the review. Out of scope: security certification, penetration and load testing, and hiring or employment terms.

Summary

The marketplace works. Its code is tidier than such a fast build would suggest. The main booking journey is tested, and the API follows one pattern throughout. The risk is not in the code but around it. One developer holds every key. Releases are copied to the server by hand, with no way back. The nightly backups have never been restored. None of this needs a rewrite. In the first 30 days: share ownership, add a release pipeline and prove that a restore works.

Sound: a tested booking journey, payment confirmed by the provider, and a consistent API.

Fragile: one person and one laptop hold every account, and no release can be undone.

Do first: an access register and a second owner on every account.

Scope and method

In scope: the web app, the API, the database, the link to the payments provider, the release process and every account the product depends on. Not in scope: code changes, the payments provider's own systems, security certification, penetration and load testing, and legal questions about who owns the code.

Materials reviewed:

- Read-only access to the code repository, as it was on 20 October.
- The release notes the developer posts in the team chat for the last 90 days.
- A list of accounts made with the developer on the first call. It became the access register.
- The backup settings and a listing of stored backups.

- An export of 2,310 bookings from the last 90 days, private details removed, and the API logs for 20 to 26 October.
- Two calls, one with the founder and one with the developer.

Evidence has three labels. Observed: I saw it in the materials, and I name the source. Reported: the team told me, and I did not check it. Unknown: the evidence was not available. I never count an unknown as zero.

The map

1 **Web app** A single-page app where buyers search, book and pay. It runs on the same server as the API.

2 **API** (2) One service handles search, bookings, provider payouts and the admin pages. It is deployed by hand.

3 **Database** (3) One managed database, with a nightly backup kept for seven days. The restore has never been tested.

4 **Payments provider** It takes card payments and tells the API. The API confirms a booking only when that notice arrives.

5 **One laptop** (1) The developer's laptop holds every key: the server key, the cloud and domain accounts, and the settings file with every password.

Findings

1. One person holds every key

HIGH

EVIDENCE	Observed. Live-system access, release rights and the domain account sit with one developer. The access register lists 14 accounts; 12 are registered to the developer's personal email, and the founder is an administrator on only 2. The server accepts a key held only on the developer's laptop. The domain renews in seven weeks on the developer's personal card.
IMPACT	If the developer is ill, leaves or loses the laptop, nobody can ship a fix, renew the domain or change a password. The product depends on one person's health and goodwill.
UNKNOWN	Nothing important.
RECOMMENDATION	An access register, shared ownership of every account, and releases from a pipeline. Every account moves into the company's name, with at least two administrators.
DONE WHEN	The founder signs in as an administrator on all 14 accounts, and a second person ships a change without the developer's laptop.

2. No way back from a bad release

HIGH

EVIDENCE	Observed. Changes are copied to the server by hand; there is no previous version to return to. The release notes show 41 releases in 90 days. Three of them were followed within an hour by a second release labelled "hotfix".
IMPACT	A bad release can only be fixed by writing another one under pressure, while customers wait.
UNKNOWN	Nothing important.
RECOMMENDATION	Add a release pipeline that can roll back in one step. It runs the tests, builds once, keeps the last five versions, and returns to any of them with one command.
DONE WHEN	A broken build, released to a test server on purpose, is rolled back in under five minutes, timed.

3. The backups have never been restored

MEDIUM

EVIDENCE	Observed. Nightly backups exist. Seven are kept, and the latest is 2.1 GB. No restore is recorded anywhere, and the developer could not remember one.
IMPACT	A backup that has never been restored is a hope, not a recovery plan.
UNKNOWN	Whether one has ever been restored, and how long it takes.
RECOMMENDATION	Restore into a separate database once a month, and time it.
DONE WHEN	Last night's backup is restored, the booking count matches the live database, and the time taken is written down.

4. Passwords were once saved in the code

HIGH

EVIDENCE	Observed. The settings file holds the database password and the payments provider and email keys. The repository history shows the file was added in the first month and deleted later, so the values are still in the history. I did not open or copy them.
IMPACT	Anyone with access to the repository, now or in the past, can read those keys.
UNKNOWN	Whether the keys have changed since. Reported: they have not.
RECOMMENDATION	Change every key that was in the history, remove the file from the history, and keep secrets in a store the company controls.
DONE WHEN	The old keys are refused, and a scan of the full history finds no secret.

5. Payments without a booking go unnoticed

MEDIUM

EVIDENCE	Observed. The API log shows 6 payment notices refused with errors during a release on 22 October. The provider sends notices again, but nothing compares payments taken with bookings confirmed. The export shows 2 bookings that were paid but still say "awaiting payment".
IMPACT	A customer can be charged with no booking to show for it, and the business finds out from the complaint.
UNKNOWN	Nothing important.
RECOMMENDATION	A daily job that compares the provider's payment report with confirmed bookings, and refuses a second notice for the same payment.
DONE WHEN	The first run lists the 2 known cases, and the job runs for seven days with every mismatch explained.

6. Search will slow down first as use grows

LOW

EVIDENCE	Observed. The main search filters on two columns with no index. The busiest hour in the logs was a Saturday morning with 61 bookings. Its slowest search took 2.4 seconds; the median was 180 milliseconds.
IMPACT	Search is the first part likely to fail under more use.
UNKNOWN	How it behaves with more traffic. Load testing was out of scope.
RECOMMENDATION	Add the index, then measure the slowest search each week.
DONE WHEN	At the next Saturday peak, the slowest search takes under one second.

What is sound

- The booking journey is tested: 38 tests cover search, booking and cancellation, and all pass at the version reviewed.
- A booking is confirmed only when the payments provider says so, never on the browser's word.
- Every API endpoint follows the same pattern, so a newcomer can read the code easily.
- The database is managed, and its nightly backups run without anyone touching them.

Action plan

First

1. Week one: an access register and a second owner on every account.
2. Weeks two and three: a release pipeline with one-step rollback.

3. Done when: last night's backup is restored, timed and written down.

Next

1. Change the keys that were in the history and move secrets to a company-held store.
2. Start the daily comparison of payments and bookings.
3. Add the search index and the monthly restore test.

Later

1. A two-page architecture note and a release checklist, so a second developer can start in a week.
2. If hired, a part-time technical lead would own the access register, release checklist and monthly restore, and review every release in the first month.

Open questions

- Is the developer staying, and for how long? The answer sets the pace of the handover.
- Who owns the code the developer wrote? That is a question for your legal adviser, not this review.
- How long could the business accept the site being down? That sets the target for the restore test.

Next steps

I walk you through the findings and the 30-day plan on a call, then answer one round of written questions. If you hire me as your part-time technical lead within 60 days, this review's full fee comes off that price.

This is a sample: a fictional system, written like a real report. A real report follows the scope we agree in writing.

John Wainaina Karanja · UTC+3 · info@johnkaranja.com · johnkaranja.com/book