Technical Review Report

Example Workshop Portal: Architecture & Code Health Check

Code Confidence · Mark Jackson · Illustrative format

Fictional sample — not client work. This is an independently authored example. The project, observations and findings are invented to demonstrate a review format. No software was examined or tested for this sample. It is not a complete assessment or a specification of what every review includes.

Executive Summary

Example Workshop Portal is a fictional application for a small maintenance company. Customers submit repair requests, staff allocate work and an AI feature drafts short job summaries. The next milestone is a limited customer pilot.

In this imagined scenario, the basic structure is understandable and the data model suits the workflow. The main concern is access to job records: the example API checks that someone is signed in, but not that a requested job belongs to their organisation. That would need attention before real customer information is introduced.

The second priority is a repeatable release process. The fictional team currently checks changes against a developer database and then releases directly to its live instance. A separate test environment and a verified recovery procedure would reduce avoidable pilot risk.

Other findings concern test coverage, attachment handling and checks around AI-generated summaries. These do not all require the same response. The review separates immediate concerns from improvements that can be scheduled later, avoiding an unnecessary rewrite.

Example scope: application architecture, selected access paths, release configuration, code structure and data handling. A real report would identify the actual code version, materials reviewed, checks performed and evidence supporting each finding.

Ratings used in this sample

RED Address before the relevant milestone.

AMBER Worth improving; priority depends on intended use.

GREEN A positive feature to preserve within the illustrated scope.

Ratings are illustrative priorities, not certification, a numerical security score or an assurance that unmentioned areas are safe.

Architecture & Infrastructure

Example coverage: data model, organisation boundaries, hosting, storage and running costs.

Database structure GREEN

In this fictional design, organisations, users, jobs and attachments are separate entities, with explicit relationships. That gives the application a straightforward foundation and avoids a single catch-all table becoming responsible for unrelated concepts.

Recommendation: preserve those boundaries. In a real review, check that required relationships and unique identifiers are enforced in the database, not only by the interface.

Organisation-level access RED

The illustrative job lookup accepts a job identifier and returns a record after checking only that the caller is signed in. A customer could potentially request a job belonging to another organisation.

Recommendation: enforce organisation ownership on every server-side read and write. Add tests using two organisations and direct requests, including attempts to access attachments. A real report would cite the exact handlers and checks supporting this finding.

Attachment handling AMBER

The fictional upload flow limits file extensions but has no clear policy for file size, permitted content types or link expiry. Extension checks alone would not establish that an uploaded file is safe or appropriate.

Recommendation: agree accepted formats and limits, keep storage private and authorise downloads. Where risk warrants it, include specialist file scanning in the agreed scope.

Hosting and cost profile GREEN

A single application service, relational database and private file store are proportionate to this fictional pilot. There is no demonstrated need to introduce a distributed architecture.

Recommendation: retain the simple design, establish a baseline for storage and AI usage, and set appropriate spending alerts. Capacity would need measurement before wider rollout.

Deployment Process

Example coverage: releases, test environments, configuration and recovery.

Release traceability GREEN

The fictional deployment uses a versioned release tied to a source commit. This makes it possible to identify what is running and compare it with the code that was reviewed.

Recommendation: retain immutable release references and record the selected version alongside release notes.

Separate test environment RED

In this example, configuration and database changes are first exercised against the live pilot service. A mistake could interrupt work or alter customer records before it is noticed.

Recommendation: introduce a separate test environment using synthetic data. Exercise database changes, access rules and critical user journeys there before releasing to customers.

Backup and recovery checks AMBER

The fictional hosting setup schedules backups, but the team has not recorded a restore exercise. A backup being present is not evidence that the application can be recovered.

Recommendation: agree a recovery objective and rehearse a restore in isolation. Record the procedure, the result and any limitations, without using customer data in an unapproved environment.

Code Quality

Readability and responsibility GREEN

The illustrative code separates HTTP handlers, data access and external-service calls. This is a useful structure for understanding behaviour and making focused changes without a broad rewrite.

Regression-test coverage AMBER

The fictional tests cover successful job creation but not invalid requests, service failures or cross-organisation access. Prioritise tests around these risks rather than aiming for a coverage percentage alone.

Data Correctness

Example coverage: AI-assisted summaries, source records and state transitions.

AI-generated job summaries AMBER

In this fictional workflow, a generated summary is displayed next to the customer's original request, but it is not clearly marked as a draft. Staff could treat an incomplete or inaccurate summary as authoritative.

Recommendation: label generated text, retain the original request and require staff confirmation before it influences job instructions. Validate structured outputs and define behaviour when the model is unavailable.

Record provenance GREEN

The illustrative data model retains the submitting user, original description and creation time. This would help staff resolve questions without relying only on generated text.

Recommendation: preserve this context and apply a deliberate retention policy to both source records and derived summaries.

Application Layer

Job-state changes AMBER

The fictional interface prevents an already closed job being reassigned, but the server accepts the same state change through a direct request.

Recommendation: enforce allowed transitions on the server and test concurrent updates. Interface controls should support, not replace, business rules.

Error responses GREEN

The imagined application returns neutral public error messages rather than stack traces or configuration values. Preserve this separation when adding diagnostics.

Support diagnostics AMBER

The fictional support flow lacks a request reference. Add correlation identifiers and content-minimised operational logs so failures can be investigated without logging passwords, tokens or customer message bodies.

Actionable Recommendations

For this fictional project, focus on reducing pilot risk before expanding features. The order below reflects the illustrated milestone, not a universal checklist.

Do this first

- Enforce organisation ownership on job and attachment requests.
- Verify the rules with two separate organisations, unauthenticated requests and invalid identifiers.
- Move release checks into an isolated test environment before introducing real customer data.

Do this next

- Define file-upload limits and private-download behaviour.
- Mark AI-generated summaries as drafts and retain the original request.
- Enforce job-state transitions on the server and add relevant regression tests.

Near term

- Rehearse recovery from backup and record what was verified.
- Add request references and appropriate diagnostic events.
- Review actual usage and cost before changing the hosting architecture.

Scope & Limitations

This document demonstrates a report format only. It contains no evidence about a real system. A real review would distinguish verified findings, assumptions and untested areas, and provide references appropriate to the access and scope agreed.

Specialist penetration testing, compliance assessment, production load testing and contractual investment due diligence are not demonstrated here. A technical review is not a guarantee of security or launch readiness.

Walkthrough & Next Decision

A real walkthrough would explain the findings, answer questions and help you distinguish work needed for your next milestone from changes that can wait. The purpose is an informed decision, not a rewrite by default.

Code Confidence by Mark Jackson
Independent technical reviews for founders and small teams.
mcj.me.uk/ai-code-review/