Securing a small web app with AI-assisted engineering

IkigaiApp — security & privacy case study

Salvatore Vivolo · Living document, version 0.4 · Last updated 4 November 2026

> This document describes principles, decisions and results. It deliberately leaves out configuration details, internal names and anything that would help an attacker.

1. In short

IkigaiApp is a personal web app (self-assessment based on the Japanese concept of *ikigai*) with about 170 registered users. In October 2026 a security incident at its hosting provider became the starting point for a complete security and privacy review. In two weeks, working with AI tools under human review, the project went from an informal "it works" state to a documented, tested and verifiable security posture.

Area	Before (October 2026)	Now (November 2026)
Secrets after the provider incident	Potentially exposed	All rotated, including the master encryption key
Access control	Never audited	Every API route reviewed; all findings fixed and covered by tests
Admin account	Password or email link	Mandatory two-factor authentication (TOTP, RFC 6238)
Personal data in logs	E-mail addresses, sometimes secrets	IDs and keyed hashes only (pseudonymous)
Account deletion	Partial	Complete, verified by automated tests
Data retention	Indefinite for several data sets	Automatic expiry, as stated in the privacy policy
Known vulnerable dependencies	Dozens of advisories	Zero known in production, checked automatically on every change
Automated tests passing	659	1,119, all green and all blocking on every change
Dependency updates	Manual, occasional	Automatic and weekly, with tests and a safety delay
Public legal texts	Partly inaccurate	Rewritten to match what the code actually does

2. Context

- **Product:** a survey that produces an "Ikigai profile", an AI-written narrative of the result, examples from similar profiles, and personal tools (journal, to-dos, flashcards).
- **Architecture:** a web frontend on a managed edge platform, an API on a managed container platform in the EU, a managed document database in the EU, a transactional e-mail provider and an AI provider.
- **Team:** one person, plus AI assistants: one for analysis, specification and review, one for implementation.
- **Constraint:** no budget for paid security tooling; everything had to be achievable with free tiers and good engineering.

3. The incident that started it

In October 2026 the hosting provider disclosed that environment variables of some customers might have been read by a third party. Whether this project was affected could not be determined, so the response assumed the worst case:

1. **Every secret treated as exposed.** Database credentials, session-signing keys, export and link-signing keys, API keys, notification tokens — all regenerated. The master key protecting the database encryption keys was rotated with a dedicated, tested procedure.
2. **Logs and provider dashboards reviewed** from the start of the exposure window: no sign of misuse.
3. **Multi-factor authentication enabled** on every provider account involved (hosting, database, code repository, DNS, e-mail, AI).
4. **GDPR accountability:** an entry in the personal data breach register (Art. 33(5) GDPR) documents the facts, the risk assessment and why notification to the supervisory authority was not required.

The lesson: field-level encryption is only as strong as the place where its keys live. When that place may be compromised, rotate the keys too, not just the passwords.

4. Security by design

Encryption at rest, field by field. Personal data (e-mail, names, nickname, income range) and everything users write (journal, to-dos, daily anchors, flashcards) is encrypted by the application before it reaches the database, with AES-256-GCM. Keys follow an *envelope* scheme: data keys are themselves encrypted by a master key that never lives in the database, and every key has a version, so keys can be rotated without downtime. A copy of the database alone does not reveal these fields.

Searching without decrypting. To find a user by e-mail or check that a nickname is unique without decrypting the whole table, the app stores a *blind index*: a keyed HMAC of the normalised value. Same input, same index. Without the key the index cannot be turned back into the value, but equal values produce the same index, which is why the index and its key are protected together.

Minimal data leaves the server. A single helper builds every view of "another user": an ID and a display name, nothing else. No ciphertext, e-mail or role ever reaches a client that does not need it.

Sessions and requests. Session tokens live in HttpOnly, Secure, SameSite cookies, out of reach of scripts. State-changing requests are checked for their origin. Links that act without a login (for example, unsubscribing) are signed and expire. The server refuses to start in production if a required secret is missing, so a forgotten setting cannot silently fall back to a weak default.

Abuse resistance. Rate limits on authentication, account checks, analytics and AI features; uniform responses that do not reveal whether an e-mail address has an account.

5. Authentication

- Users sign in with an e-mail link or a password.
- **Administrators must use two-factor authentication**: time-based one-time codes per RFC 6238 (the open standard behind Google Authenticator, Authy, Microsoft Authenticator, Apple Passwords and others), implemented with the platform's standard cryptography and verified against the official test vectors. Codes cannot be replayed; ten single-use recovery codes are stored only as slow hashes; the secret is encrypted like the rest of the personal data.
- A **break-glass procedure** exists for a lost device. It can only be run by the owner from the hosting console, and every use is written to the audit log.

6. Access control: auditing every route

The single most valuable step was a systematic review of every API route — 283 of them — asking three questions: who may call it, whose data can it touch, and what does it return? The audit found 14 problems and 7 points to verify, from an administrator check that a missing pair of parentheses had made always true, to records of other users reachable by changing an ID in the request.

All were fixed in four batches. Each fix came with a test that **fails on the old code and passes on the new one**, so the hole is proven to exist and proven to be closed.

7. Privacy and GDPR

Privacy was treated as a design constraint, sometimes at the expense of convenience:

- **Analytics without identity.** Usage statistics store neither the user ID nor the IP address: only pages, a random visit identifier and coarse context. Per-user analytics were deliberately given up. A single first-party cookie remains, holding a random value that lasts 30 minutes; statistics are used only in aggregate and are neither combined with other data nor shared. Assessed against the Italian data protection authority's guidelines, the cookie is treated as a technical cookie, with no consent banner.
- **Automatic retention.** Usage statistics expire after 13 months, technical data after 90 days, AI usage logs after 12 months, security logs between 90 days and 12 months — enforced by the database, not by good intentions.
- **Real deletion.** Deleting an account removes everything linked to it, across every collection. A test creates data everywhere and checks that nothing survives. Accounts deleted in the past were cleaned up retroactively. What remains is security logs until they expire and encrypted backups until they are replaced, within 6 months, as the privacy policy states.
- **Export.** Users can download all their data (JSON or CSV) at any time.

- **Pseudonymised examples with opt-out.** The "similar profiles" feature uses copies of survey results without names or contacts, linked to the account only by a keyed code, so they can be deleted on request. A single switch lets users opt out.
- **No fake switches.** Consent toggles that did nothing were removed. The app does no marketing, sells no data and trains no models.
- **Logs with minimal personal data.** Audit and event logs keep IDs and keyed hashes, which are pseudonymous data with an expiry; e-mail addresses and secrets are redacted on write and on read.
- **Documentation that matches the code.** Privacy policy, cookie policy, data-rights page, AI transparency page and terms were rewritten against the implementation. An internal record of processing, a processor list, a retention schedule and a data-subject-request procedure are kept and versioned.
- **A public promise.** The "Our promise" page sums up, in seven plain-language commitments, what the app does to protect data, with links to the full policy.

8. AI, responsibly

- AI is used for three features: the narrative profile, flashcard generation and translation of administrator messages. It receives only what each feature needs. For the profile that means derived categories, never names, e-mail, user ID or free-text answers.
- The provider does not train on this data; no automated decision with legal effects is taken.
- Everything generated is labelled "Generated with AI", and the label survives sharing. Blog articles state when AI tools helped with the text or images.
- Based on the assessment made against the EU AI Act, the features are not high-risk; labels make the use of AI visible. Archetype calculation and similarity matching are deterministic and explainable, not machine learning.

9. Where the app stands against 2026 threats

AI-assisted attacks. Attackers now use AI to scan applications at scale for authorisation gaps, to automate credential stuffing and to personalise phishing. The response here is to remove what such tools look for:

- every route has been checked for authorisation;
- responses do not reveal whether accounts exist;
- rate limits slow automation;
- the admin account requires a second factor;
- encrypted fields and minimal logs limit what a breach would expose.

The AI features generate text only and cannot act on data, so prompt injection has no path to cause damage.

Software supply chain. Recent waves of malicious package releases showed that a single compromised dependency can steal secrets at install time. Practices in place:

- exact versions and committed lockfiles;
- a minimum age of seven days before adopting a new release;

- install scripts disabled when upgrading;
- reproducible clean installs in deployment;
- continuous integration: every change runs the blocking tests, the build and a vulnerability check on production dependencies, and a change that fails does not get in;
- automatic weekly updates, grouped and with a seven-day delay, while security fixes arrive straight away; major versions remain manual decisions;
- CI actions pinned to exact versions (commit hashes);
- zero known vulnerabilities in production dependencies.

Infrastructure. The provider incident showed that secrets in a hosting platform must be treated as a single point of failure. In place:

- MFA on all provider accounts;
- rotated and sealed secrets;
- fail-fast configuration;
- a health endpoint that reveals nothing;
- debug pages removed;
- encrypted, verified backups.

Honest limits:

- no external penetration test yet;
- rate limits are kept per server instance;
- anomaly alerting is basic;
- data encryption keys are still to be rotated, and backups are still manual.

10. How the work was done: AI-assisted, human-reviewed

- **Roles.** One AI assistant analysed the code, wrote specifications and reviewed every change; a second implemented them on separate branches. The owner made every decision, merged, deployed and ran every production operation personally.
- **Rules in the repository.** A rules file tells coding agents what they may never do: no secrets in code or logs, no weakening of security controls, tests for every behaviour change, every index change documented, progress reported.
- **Evidence over trust.** Every batch ended with a report: tests added, full test suite compared with the main branch **test by test** (not just totals), commands to run in production.
- **Production changes in two steps.** Every data change ships as a script that defaults to a dry run printing only counts, never personal data. The real run comes after reviewing the numbers, and after a fresh encrypted backup.
- **Review catches what tests miss.** One batch passed all its own tests but deleted from three collections under wrong names: the tests used the same wrong names. Human review of the diff caught it before production, and the fix moved the code to names derived from the data models. On another occasion, verifying a backup revealed that the backup script had been mixing a newline into the encryption key; it was fixed before any restore was needed.

- **CI finds what the developer's machine does not.** On its first run on a Linux server, four test files that were green on the Mac failed: a macOS-only path, missing dependencies, a timing-sensitive test. No bug in the app, but exactly the kind of surprise CI is there to catch before a merge.
- **Reviving old tests finds real bugs.** Bringing 183 long-failing tests back to green meant separating, one by one, outdated tests from real bugs. A few real ones surfaced: a submitted survey could stay a draft without scores, a registration stage did not update after e-mail verification, and a copy of encrypted content reached a response without being decrypted. Fixes touching authentication and encryption were stopped, documented and made in a separate, reviewed batch. Before becoming blocking, flaky tests were made deterministic and run twenty times on two operating systems.

11. Status and next steps

Item	Status
Secret rotation after the incident, MFA on providers	Done
Access-control audit and fixes	Done
Two-factor authentication for administrators	Done
Privacy: deletion, export, retention, minimal logs	Done
Legal and compliance documents aligned with the code	Done
Public "Our promise" page	Done
Continuous integration with blocking tests and automatic dependency updates	Done
Full test suite green, blocking merges	Done
Rotation of the data encryption keys	Planned
Automatic encrypted backups with restore test	Planned
Automatic alerts on anomalies (failed-login or rate-limit spikes, new admin logins, backup failures)	Next, after backups
Minimum release age in npm, CI actions pinned to exact versions	Done
Package signature verification, code and secret scanning	Next
Security headers review, periodic restore drills, incident runbook	Next
Passkeys for administrators, optional two-factor for users	Later
External penetration test or automated web scan	Later

12. Lessons for any new project

1. Audit authorisation route by route before adding features: it is where real bugs hide.
2. Encrypt sensitive fields in the application, and plan key rotation from day one.
3. Collect less. Data you do not store cannot leak, needs no retention policy and needs no consent.

4. Make the privacy policy describe the code, then keep them in sync.
5. Use AI for speed, people for judgement. Ask for evidence (tests that fail first, before/after comparisons), not reassurance.
6. Every production data change: backup, dry run, review, run, verify.

Living document: updated as each remaining step is completed.