Building an Auditable Operational Platform *for Distressed Contexts*

A note on the design principles, technical architecture, and operational rationale behind the Foundation's ACES™ place-based regeneration platform

AUTHORS
Mariyah Acka'a
with V-Innovation Labs

PUBLISHED
19 May 2026

STATUS
Open for review

PAGES
~12 pp.

ABSTRACT

Most software designed for the NGO sector is built for stable infrastructure and digitally fluent users. Programmes operating in distressed contexts — intermittent connectivity, low device standardisation, limited technical support, high turnover among local staff — need a different design philosophy. The dominant approach in the sector, which assumes off-the-shelf SaaS deployed via professional services, fails in these contexts more often than it succeeds, and the failures are expensive. This paper documents the architectural choices behind the ACES™ operational dashboard: single-page progressive web application, offline-first state management with cloud synchronisation, multi-currency reporting layer, integrated assurance and audit-pack generation, participant-facing scorecards, and a layered backup and integrity system. We position these choices as a design pattern for the sector rather than a product, and we discuss the open governance, openness, and sector-value questions that follow from this approach.

I. The software gap in place-based programmes

The development sector spends a great deal on software it then fails to use. The pattern is consistent enough to have a name in the implementation literature: *ICT4D failure*.[1] The technology is procured against the assumptions of a developed-world enterprise context — reliable broadband, modern devices, professional IT support, fluent administrative staff — and then deployed into a context where those assumptions do not hold. The system works in the demo room and fails in the field.

Richard Heeks's foundational work on information systems in developing countries identified what he termed the *design-reality gap*: the larger the gap between the assumptions designed into the system and the reality of the context, the more likely the failure.[2] Kentaro Toyama's later critique extended this with the observation that technology amplifies existing human capacity rather than substituting for it — a system that requires capabilities the operating context does not have will not perform regardless of how well-designed it is in the abstract.[3]

> *The development sector spends a great deal on software it then fails to use.*

What the Foundation needed for ACES™ was a dashboard a programme manager could use on a low-end Android tablet over an intermittent 3G connection, with offline persistence, with an audit trail any funder could inspect, and with reporting good enough that a UK trustee could read the same numbers as the Adikpo team. We did not find a fit-for-purpose product. We built one. This paper documents the design choices, not as a product specification, but as a pattern other foundations and implementers may adapt to their own contexts.

II. Design principles

Five principles guided every architectural decision.

Offline-first, network-tolerant

Every action a user takes — recording a payment, logging a household visit, completing a survey — must succeed locally even if the device has no network. Synchronisation to the cloud is a background activity, not a precondition for the work. The user is never blocked by the absence of connectivity, and the user is never asked to remember that they were offline.

Single-file deployment

The entire application, including its CSS and JavaScript, is delivered as one HTML file. No build chain, no node_modules, no Docker image, no compile step. A new deployment to a new zone is one file copy. A bugfix is one file replacement. The cost of this constraint is a larger single artefact than a chunked modern web application; the benefit is operational simplicity in environments where build chains and tooling assumptions are themselves a source of fragility.

Auditable by default

Every meaningful action writes to an append-only audit log: the user, the role, the timestamp, the action, the affected record. The audit log is never deletable from the user interface and is included in every audit pack export. Operations whose audit trail is missing or compromised are operations a funder cannot trust.

Beneficiary-visible

The participants of the programme — the resource stewards in Adikpo — can see their own data: their earnings, their tier, their progress. A system that records people without showing them what it records reduces them to data subjects. A system that shows them their own record returns some of the agency the data extraction otherwise removes.

Funder-ready exports

Reporting to funders should not require a separate effort. The system generates the artefacts funders need — board packs, audit packs, grant progress reports — from the same operational records that run the programme. The board pack a UK trustee reads in May is mechanically derived from the same dashboard the Adikpo team used in March.

III. Architecture overview

The platform is built on five technology choices, each made against the design principles above.

LAYER	CHOICE	RATIONALE
Client	Single-page HTML application; vanilla JavaScript; PWA with service worker.	No build chain. Runs on low-end devices. Installable. Works offline.
Local state	Browser localStorage with a structured state object.	Survives loss of network. Survives loss of session. Available immediately on page load.
Cloud backend	Supabase (managed PostgreSQL with row-level security).	Generous free tier appropriate to a small foundation. Open source. RLS gives per-role access control without bespoke API layer.
Hosting	Netlify static hosting; serverless functions for FX rate retrieval where required.	Zero-cost deployment. Atomic releases. Branch-based previews.
Authentication	Supabase Auth (magic link & OAuth) for staff; PIN-gated section access for sensitive modules.	Removes password management burden from local staff. PIN gates for finance and assurance add a second factor for sensitive areas.

The client carries the entire user-facing application. Operations write first to the local state object, immediately persist that state to localStorage, and then attempt to synchronise the relevant key to Supabase. When synchronisation fails (lost network, expired session, server error), the local write is preserved and is retried on the next opportunity. The user sees an unobtrusive online/offline indicator but is not interrupted by network state.

The application is also a Progressive Web App. A service worker caches the application shell and core assets, allowing the dashboard to load even with no network at all. A user who has previously opened the dashboard can open it again offline and operate normally; on next connection the queued writes synchronise.

IV. The resilience layer: backups, snapshots, integrity

Data loss is not a hypothetical in field deployments. Devices break, accounts get reset, browsers clear localStorage, services experience outages. The resilience layer is built on the assumption that any one storage location may fail, and the system must continue without it.

Programme data lives in four locations, ordered by recency:

1. **Browser localStorage** — the current working copy, updated on every action. Lost if the user clears browser data.
2. **Supabase cloud** — the shared source of truth across users and devices. Lost only if the cloud project is destroyed.
3. **Automatic daily snapshots** — the system takes a snapshot to localStorage on first session of each day; the most recent seven snapshots are retained. Lost with localStorage but independent of the working state.
4. **Manual backup files** — downloadable JSON files held in administrator-controlled offsite storage (cloud drive, encrypted USB). Lost only if the administrator loses access to their own offsite storage.

Restoring from any of these locations is supported through the same restore interface. The interface validates the backup file before applying it, takes a pre-restore snapshot of the current state as a fallback, and replays the restored data to the cloud after applying it locally.

A data integrity check runs on each opening of the resilience interface and on request. It scans the live state for orphan references (attachments pointing to deleted records, deliverables pointing to deleted grants), duplicate identifiers within collections, and structural inconsistencies. Detected issues are reported with their type and the affected record. A cleanup action removes orphans automatically; duplicates are flagged for human review because the correct resolution is not generally automatable.

The resilience layer is, on its own, not novel. What is unusual is the integration with the rest of the platform: backups are administrator-facing rather than operations-facing, the daily snapshot is automatic and silent, and the integrity check is part of the routine operating loop rather than an audit-time exercise. The result is a system that is genuinely robust to the kinds of failure that produce most NGO data losses.

V. The multi-currency reporting layer

ACES™ operates in Nigerian Naira (₦). Funders to the Foundation report in pounds, dollars and euros. A naive approach would convert at the data-entry layer, with all the problems that creates: lost precision, double conversions, irreproducible historical figures.

The platform takes a different approach. The **functional currency** is the Naira and remains so throughout the data model; every transaction, balance, and ledger entry is stored in ₦ at the rate that applied when the transaction occurred. The **reporting currency** is a presentation-layer choice applied at report generation time, using administrator-set rates with explicit source attribution and a date stamp. Reports rendered in non-functional currencies carry a footnote stating the rate used, the date it was set, and the source.

The administrator sets rates in a configuration interface in the Users section. Each rate carries a "source / note" field intended for auditor-facing attribution — for example, *"Bank settlement rate, May 2026, from First Bank of Nigeria"*. The Foundation's working convention is to update rates monthly and before each funder report. Reports inherit the currency selected at the time of generation; board packs and audit packs persist the selected currency with the artefact, so that a trustee viewing the same pack months later sees the same numbers in the same currency.

This layered approach respects the auditor's expectation that the books of account remain in the functional currency, while giving funders the presentation they need without contaminating the underlying ledger.

VI. The Participant Scorecard: making the beneficiary visible

One of the architectural choices that has been most appreciated in the field is the Participant Scorecard. Each participant has a downloadable, branded, certificate-style document that shows their income to date, their tier (Resource Steward, Resource Recovery Partner, Circular Steward, Community Resource Partner, Community Regeneration Manager), their progress, and an authenticated verification identifier. The document is printable, shareable, and recognisable as belonging to the participant.

The scorecard does work along three axes. *Operationally*, it gives the participant a tangible record of their relationship with the programme — something they can show to a family member, a future employer, a microcredit officer. *Politically*, it asserts that the data the programme holds about a person belongs in part to that person. *Practically*, it gives funders a beneficiary-facing artefact that closes the loop on programmatic reporting in a way that aggregate statistics cannot.

The technical implementation is modest — the scorecard is generated on-demand from the participant's record in the dashboard, rendered with the Foundation's letterhead and design tokens, with a verification identifier of the form `ACES-{participant-id}-{date}` that an external party could in principle check against the Foundation's records. The substantive point is not the technical implementation; it is the design commitment to treat the participant as a party to the data, not only its subject.

VII. The assurance layer: risk, safeguarding, compliance, conflicts

Operational systems often treat assurance as an afterthought — a quarterly spreadsheet maintained separately from the operating ledgers. The ACES™ platform integrates four assurance registers directly:

- **Risk register** — a 5×5 likelihood/impact matrix with owners, mitigations, and review dates.
- **Safeguarding cases** — a confidential workflow with restricted access; only counts are visible in standard reports.
- **Conflicts of interest** — declared by board members and senior staff, refreshed annually, with material conflicts triggering recusal records.
- **Compliance calendar** — recurring statutory deadlines (annual return, audit submission, etc.) with auto-rolling due dates upon completion.

The integration matters because assurance carried out in a separate system is assurance the operating team forgets to do. Embedded in the same dashboard the operating team uses daily, the registers become live working documents rather than annual rituals.

The audit pack export bundles all four registers into a single letterheaded document over a chosen period, alongside the financial transaction ledger, grants activity, the audit log, and an evidence manifest. Safeguarding details are intentionally redacted to counts only. The result is a document an auditor or funder can request and receive without the operating team having to compile it from scratch — and without the temptation to selectively curate what the auditor sees.

VIII. Open questions: governance, openness, and sector value

We close with three questions we have not resolved, and which we put to peer foundations and implementers.

Open source?

The platform is currently maintained as a closed asset of the Foundation. The arguments for open-sourcing it are familiar: amplification of impact, peer scrutiny, sector value. The arguments for caution are also real: maintenance burden, the risk that a partial release will be deployed by less-equipped teams to contexts it is

not calibrated for, and the security implications of public scrutiny of an unaudited codebase. We have not yet decided. We are open to conversation with foundations interested in adapting the pattern.

Data governance across jurisdictions

The platform stores personal data about Nigerian residents in cloud infrastructure that is not, by default, Nigerian. The legal and ethical implications — the Nigeria Data Protection Act 2023, the UK Data Protection Act 2018, the Foundation's own governance commitments — require a more thorough treatment than this paper can give. A separate governance note is in preparation.

The product-versus-pattern question

We have built a platform; we have not built a product. The distinction matters. A product would be packaged, supported, sold or distributed to other foundations. A platform is an architecture — specific choices, specific trade-offs — that other implementers might learn from in building their own. We currently think the sector is better served by patterns than products, on the grounds that a product imposes its assumptions, while a pattern invites adaptation. We may be wrong about this.

HOW TO CITE THIS WORK

SUGGESTED · AUTHOR–DATE

Acka'a, M. (2026). *Building an auditable operational platform for distressed contexts: The ACES™ dashboard architecture* (Methodology Paper v1.0). The Acka'a-Hitchman Foundation, with V-Innovation Labs. https://vheritage.online/methodology/aces-platform-architecture.html

SUGGESTED · FOOTNOTE

Mariyah Acka'a, *Building an Auditable Operational Platform for Distressed Contexts*, Acka'a-Hitchman Foundation Methodology Paper No. 4 (v1.0, 19 May 2026), https://vheritage.online/methodology/aces-platform-architecture.html.

REFERENCES

1 Dodson, L. L., Sterling, S. R., & Bennett, J. K. (2013). "Considering failure: Eight years of ITID research." *Information Technologies & International Development*, 9(2), 19–34.

2 Heeks, R. (2002). "Information systems and developing countries: Failure, success, and local improvisations." *The Information Society*, 18(2), 101–112.

3 Toyama, K. (2015). *Geek Heresy: Rescuing Social Change from the Cult of Technology*. New York: PublicAffairs.

4 Heeks, R. (2010). "Do information and communication technologies (ICTs) contribute to development?" *Journal of International Development*, 22(5), 625–640.

5 Avgerou, C. (2008). "Information systems in developing countries: A critical research review." *Journal of Information Technology*, 23(3), 133–146.