Simplycity Platform — Proof of Concept Guide

VERSION	0.9 (PoC)
DATE	21 September 2026
STATUS	working sandbox; not an operational system

> Simplycity is a coordination, assurance and evidence platform. It does not manufacture aircraft, operate aircraft, or replace the safety, clinical or regulatory responsibilities of its partners. Everything in this sandbox uses simulated SC-17 aircraft and illustrative organisations, sites and data.

1. What this is

A running implementation of the design in SIMPLYCITY-Platform-Design.md: Flow (mission orchestration and release policy), Trust (role-based views, tamper-evident ledger, alerts, controlled configuration) and Impact (metrics computed from mission records, noise, community sentiment, controlled exports) for one care-corridor network: OLVG West <-> Amsterdam UMC AMC Lab / VUmc Lab.
It goes beyond MVP Release 1 of the design: it also contains the connected-corridor and reporting features of Releases 2 and 3, against a simulated aircraft instead of a real OEM integration.

2. Where to find it

What	Where
Website	http://<vm-public-ip>:8080/
Platform console (sign in by role)	/platform/
Guided demo (start here)	/platform/ -> "Watch a guided demo"
Community transparency page (public, no login)	/community/
OEM / operator integration contract	/platform/api.html (rendered from /api/v1/spec)
Unit-economics sensitivity tool	/platform/economics.html
Design library and this guide	/resources/deliverables/

The VM has no fixed IP: the public address changes if the instance is stopped and started. Attach an Elastic IP before sharing the link widely. Traffic is plain HTTP; put TLS in front (load balancer or reverse proxy) before any non-demo use.
Privileged roles. Operator manager and Administrator require a sandbox passcode, generated at first start and stored in simplycity/data/admin_passcode.txt (mode 0600). The other roles sign in with one click because the sandbox holds no real data. The passcode is stored hashed in the database.
Operating it (on the VM, from /home/ubuntu/simplycity):

```
./run.sh status | start | stop | restart | logs
python3 -m unittest discover -s tests -v # 29 backend tests (isolated server on :8092)
```

A cron entry runs ./run.sh ensure every minute and at reboot, so the site restarts after a crash or VM restart. Code lives in server/ (Python standard library only, SQLite), the public web root in web/, and the database in data/. Only web/ is ever served.

3. Build status against the design

Design item	Status
Organisations, users, roles, corridor/hub configuration	Built: 6 organisations, 10 users across 6 roles, 3 hubs, 3 corridors, 7 contingency sites
Clinical request, cassette custody, mission state machine	Built: explicit states and owners; every transition ledgered; exceptions, holds, returns, diversions, incident review
Release policy: 12 mandatory gates, snapshot, named approval	Built: gates evaluate live data, stale or missing data fails closed, snapshot stores inputs and rule version, release needs a named operator and explicit confirmation, launch re-checks all gates
Operator console, alerts, escalation	Built: acknowledge / resolve with ownership; unacknowledged criticals escalate operator -> manager -> administrator
Tamper-evident ledger	Built: SHA-256 hash chain plus database triggers refusing UPDATE/DELETE; whole-chain verification; non-destructive tamper demonstration
Telemetry ingest, live map, energy/health state	Built: POST /api/v1/telemetry contract; the simulator is a client of the same function
Weather / manual data source with expiry	Built with a simulated feed and a manual-observation path (1–30 min expiry)

Exception workflow	Built: hold, abort (pre-launch), return, divert, incident review, aircraft inspection/recovery
Service, energy, adherence, exception reporting	Built: computed from mission records, with scope filter (live vs synthetic)
Noise attachment with method	Built: method text mandatory
Authority view, controlled PDF/CSV export	Built: role-masked; every export hashed and recorded in the ledger
Configuration/version approval workflow	Built: two-person rule, partner-approval reference, hard OEM envelope limits, rollback via new version
Document vault	Built: content-hash storage; uploads and downloads ledgered
Community feedback and public transparency page	Built (goes beyond the design)
One real OEM/operator integration	Not built: the contract exists and works, but only against the simulator
Real weather / airspace / UTM provider	Not built (simulated feed and manual entry only)
SSO / hardware MFA, TLS, hosted infrastructure	Not built (TOTP is implemented; sandbox uses a passcode)

4. How it works

Mission states — REQUESTED -> PAYLOAD_READY -> PREFLIGHT_CHECK -> RELEASED -> LAUNCHING -> AIRBORNE -> APPROACHING -> LANDED -> HANDED_OVER -> CLOSED, with ON_HOLD, RETURNING, DIVERTING, INCIDENT_REVIEW, ABORTED, REJECTED, CANCELLED. Illegal transitions are refused; each state has a named owner role.
Release gates — eligible payload · origin hub ready · destination hub ready · route available · weather within limits · aircraft healthy · energy sufficient (route + wind + 30 % reserve) · maintenance valid · crew available · communications ready · route configuration current · no active hold. Each gate reports observed value, threshold, data source and data age, and who owns the fix.
Aircraft model — the SC-17 reference numbers from the blueprint (0.42 kWh battery at 80 % usable, 26.4 m/s cruise at 0.47 kW, 2.5 kW vertical legs). A simulated 18 km corridor mission consumes about 0.165–0.170 kWh, matching the blueprint's 0.170 kWh. These are planning values, not validated performance.
Roles — clinical requester, hub attendant, remote operator, operator manager, authority viewer, administrator, plus an OEM service key. Clinicians and attendants see only their own organisation's or hub's missions; the authority view masks clinical references and seal ids; managers configure but do not release; the author of a configuration change can never approve it.
Trust ledger — every event stores the previous event's hash. Editing or removing history breaks every later link, and verification finds it. Denied releases, custody exceptions and refused requests are recorded as evidence, not rolled back.
Time in the sandbox — flights run faster than real time (default 8x; 16x during guided demos). Flight duration is simulated; ground steps use real time. Reports say so.

5. Security posture (honest summary)

Implemented: role-based access with data scoping; PBKDF2 password hashing; TOTP MFA capability; CSRF token on every write; HttpOnly SameSite cookies; per-IP rate limiting (reads, writes, sign-in); strict input validation with plausibility ranges; API keys stored hashed and revocable; a static file server confined to one web root with path-traversal tests; Content-Security-Policy and standard security headers; append-only ledger.
Known gaps for any real deployment: no TLS; one-click sandbox logins for non-privileged roles; single-process SQLite (adequate for the pilot's scale, not for a city network); no SSO; no formal penetration test; no data-protection impact assessment. Retention, deletion and data-sharing terms remain open decisions (design document, section 15).
Incident note (21 Sept 2026). The earlier static site was served with python3 -m http.server from the home directory, which exposed the whole home directory (including tool credentials) to the internet while it ran. The new server serves only web/. Credentials that lived in the home directory during that window should be rotated.

6. Test evidence

- 29 backend tests (tests/test_platform.py): energy model against the blueprint, corridor length, RFC 6238 TOTP vector, permission and CSRF checks, data scoping and masking, path-traversal attempts, custody exceptions that

persist their evidence, denied releases that persist their snapshot, stale-data fail-closed, hold blocking, energy gate, full mission end to end, return-to-origin with incident review, two-person configuration rule and hard limits, API key lifecycle and telemetry validation, hashed and ledgered exports, tamper detection including offline database edits.

- UI tests (tests/ui_smoke.mjs, tests/ui_shell.mjs): every view rendered for every role, and the app shell driven through sign-in, navigation, a refused request and a scan through the modal UI.
- Guided demos: four scripted scenarios run through the real platform (tests/demo_run.py).
- Layout was inspected in a headless browser at desktop and phone widths.

7. Limits of what this proves

It shows the workflow, the rules engine, the evidence model and the operator experience working together. It does not show that a real aircraft can fly the route, that weather and airspace data can be integrated as assumed, that an operator will accept the release model, or that the numbers in Impact would look like this in the field. Synthetic history is flagged wherever it appears and excluded when scope is "live".

8. Recommended next steps

1. Put the platform behind TLS with a fixed address and a real domain.
2. Choose the anchor care partner, operator and OEM; replace the simulator with one real integration against /api/v1.
3. Replace the simulated weather feed with an approved source; agree the airspace data source with the authority.
4. Run the four guided scenarios with the operator's safety manager and record what they would change in the release policy.
5. Add SSO and hardware MFA; commission a security review and a data-protection assessment.
6. Decide retention, deletion and sharing rules per role with each partner.