

Spiti by OC Travel — Event Concierge Portal

An enterprise-grade, multi-role portal for running the full VIP guest journey of a luxury event: nomination → multi-level approval → concierge orchestration → scoped vendor execution → the guest's white-glove digital itinerary. Built so **the client owns the codebase outright** — clean source, no vendor lock-in, no secrets in the tree.

New here and not a developer? Open `START-HERE.md`, then `RUN.md`. This README is the developer-facing companion. Server deployment (systemd, reverse proxy, backups, the Postgres scale route) lives in `DEPLOYMENT.md`.

Schema note. `migrate()` runs on every boot and is additive and idempotent: it creates missing tables, adds missing columns (guarded), and performs one guarded rebuild of the `messages` table to make `event_guest_id` nullable (general lane threads carry `event_id` instead). No existing row is rewritten. Take a DB backup before the first upgrade boot as a habit.

Run it locally

```
npm install      # install dependencies
npm start        # start the portal on http://localhost:3000
```

On first boot the database is created and seeded automatically with a full example event (LV Cruise 2027 Preview, Paris). No configuration is required to run — the flight feed is a labelled simulation and sign-in links appear on screen in dev mode.

```
npm run seed      # (re)seed if empty
npm run reset     # wipe and reseed the demo data
npm run test:rbac # run the adversarial RBAC test suite (the security proof)
```

Architecture

- **One Express app**, server-rendered HTML. No build step, no bundler: `node server.js` and it runs. This keeps the client's ownership real, and routes live in `src/routes/` (one module per role: admin, events-admin, concierge, approver, store, vendor, guest, director, zone, plus auth and home). Nothing is compiled or hidden.
- **SQLite** (`better-sqlite3`) for persistence — a single portable file at `data/spiti.db`. Swap to Postgres by isolating the data layer (`src/db.js`); the model is standard relational.
- **RBAC is the spine**. Enforcement is server-side, field-level, deny-by-default (`src/rbac.js`). Vendors and brand roles receive projected DTOs (`hotelGuestView`, `transportGuestView`, `restaurantGuestView`, `approverGuestView`, `brandOrgView`, `zoneNominationView`) that contain *only* their granted fields; a denied field never crosses the wire, so it cannot be rendered. The brand-org DTOs never join a guest table at all, so a Client Director reaches a guest COUNT and never a roster, and a Zone Manager reaches a judge-only projection across their zone and never a logistics field.
- **Ten roles, ten walls**. OC Admin, OC Concierge, Client Director, Zone Manager, Brand Approver, Store Manager, three vendor types (hotel, transport, restaurant/experience), and Guest. People-management writes (appoint / suspend / remove) are role-restricted, scope-checked from the *session* user record (never a request param), and audited. Message lanes (`guest_facing`, `internal`, `hotel_oc`, `brand_oc`) are participant-gated in the query, with a counterparty binding, never filtered in the browser.
- **The signature propagation** (`src/derive.js`): `Flight.actual_arrival` is the single source of truth. Transport pickup time, hotel expected arrival, and derived itinerary times are *computed* from it — there is no code path that stores them independently, so they cannot drift. One write, everything downstream recomputes.
- **The flight feed** (`src/flight-sim.js`) is a realistic simulation behind an adapter, labelled "simulated" in the UI. Connect FlightAware AeroAPI or Cirium by setting `FLIGHT_PROVIDER` + `FLIGHT_API_KEY` — config, not a rewrite.
- **Auth** (`src/auth.js`): passwordless magic-link, with a demo role-picker layered on top for walking in as any role.
- **The skin** is Elodie's design tokens (`public/design-tokens.css`), imported verbatim — two density registers (efficient for ops/vendors, spacious for the guest), co-branding via a `data-brand` attribute.

Project layout

server.js	App entry – mounts routes, seeds on first boot
src/	
db.js	Schema + at-rest encryption for passport/PNR/DOB
rbac.js	The walls: scoped queries + field-projection DTOs
derive.js	The single-write flight propagation (pure functions)
flight-sim.js	Flight adapter (simulated; real providers by config)
auth.js	Magic-link + sessions + role guards
seed.js	The LV Cruise 2027 Preview demo data (named cast)
nav.js, view.js	Shared rendering (shells, components, wordmark)
src/routes/	One module per role (admin, events-admin, concierge, approver, store, vendor, guest, director, zone) + auth + home
public/	design-tokens.css, app.css, app.js (static)
test/adversarial-rbac.test.js	The §2.5 checklist as executable tests
data/	SQLite file (gitignored; created on first run)

Security posture (learned from the previous build)

1. **Client owns the codebase** — plain, documented source; no lock-in.
2. **Secrets never in source** — only `.env.example` (placeholders) is committed; `.gitignore` excludes real env files and the database. Rotate every provider key at handover.
3. **Auth + email on an owned domain** — magic-link, resilient to a single mailbox outage; RUN.md walks connecting email on an OC-Travel-owned domain.
4. **Need-to-know enforced server-side** — the adversarial checklist in `test/` is the passing proof, not a promise.

All names, flights, figures, and the two example brands (Louis Vuitton, Cartier) are **illustrative sample data** to make the walls read as a story — clearly marked throughout the product, and replaced with real events in use. No real brand assets or claimed brand colour specs are reproduced.