



# Fieldra

Technical guide

Architecture, data model, protocols and how-tos  
for developers continuing the phone app, Fieldra Desktop and the website.

Version 0.5 · October 2026 · Luminex Ltd

Repositories: [github.com/luminex-ltd/fieldra](https://github.com/luminex-ltd/fieldra) · [fieldra-desktop](#) · [fieldra-web](#)

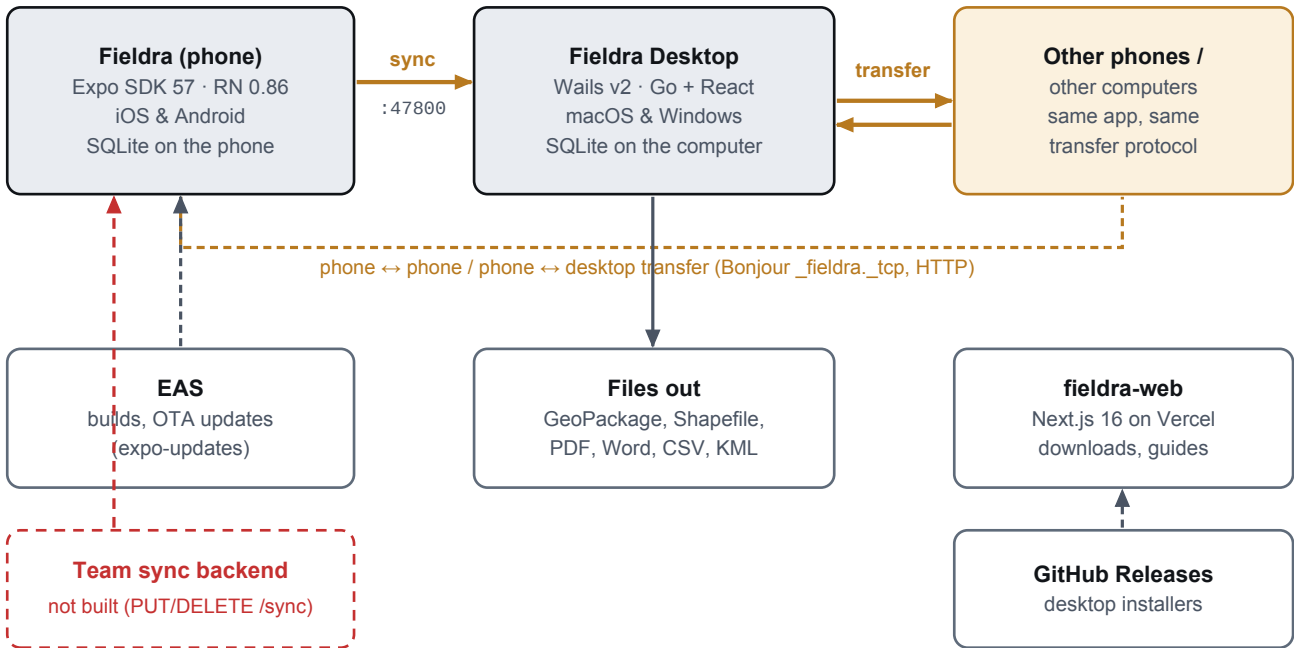


# Contents

- 1. The system at a glance . . . . . 3
- 2. Setting up a development machine . . . . . 4
- 3. Phone app architecture . . . . . 5
- 4. Fieldra Desktop architecture . . . . . 14
- 5. Protocols and data contracts . . . . . 27
- 6. Website (fieldra-web) . . . . . 29
- 7. Building and releasing . . . . . 30
- 8. How to... . . . . 32
- 9. Testing and quality . . . . . 33
- 10. Known limitations and next steps . . . . . 34
- 11. Terms . . . . . 35

# 1. The system at a glance

Fieldra is offline-first field survey software. Data is created on phones, lives in SQLite on each device, and moves between devices over the local network. There is no cloud dependency: a phone, a laptop and a Wi-Fi router (or a phone hotspot) are enough.



Components and how data moves between them. Red: planned, not built.

| Repository      | Stack                                                                                           | Purpose                                                       |
|-----------------|-------------------------------------------------------------------------------------------------|---------------------------------------------------------------|
| fieldra (phone) | Expo SDK 57, React Native 0.86, TypeScript, Bun, expo-router, expo-sqlite, @rnmapbox/maps       | Field data collection, offline, all modules                   |
| fieldra-desktop | Wails v2.12 (Go 1.25 + WebView), React 19, Vite 8, Tailwind 4, shadcn/ui, MapLibre 5, deck.gl 9 | Receive, map, edit, analyse, drone mapping, reports, releases |
| fieldra-web     | Next.js 16 (App Router), React 19, Tailwind 4, Vercel                                           | Marketing, downloads, guides (PDF)                            |

## Design principles

- **Offline first.** Every write goes to the local database first; sending is a background concern. No screen waits on the network.
- **One record model everywhere.** A record is a camelCase JSON payload with `id`, `projectId` and `updatedAt`, stored per table. The phone, the desktop and the transfer protocol all use the same payload.
- **Idempotent, last-write-wins.** Every upload can be repeated safely; the newest `updatedAt` wins.
- **Plain language.** UI text and code comments are written for non-specialists (field teams, not surveyors).
- **Monochrome UI** with a brown secondary; colour only for data and state. Removals always ask first, with a black (not red) button.

## 2. Setting up a development machine

| Tool                   | Used by              | Notes                                                                                                 |
|------------------------|----------------------|-------------------------------------------------------------------------------------------------------|
| Bun 1.3+               | phone, web           | Package manager and script runner (bun.lock). Use bunx, not npx.                                      |
| Node 22 + npm          | desktop frontend     | Wails runs <code>npm install / npm run build</code> in frontend/.                                     |
| Go 1.25                | desktop              | go.mod declares 1.25; builds need macOS 12+ to run.                                                   |
| Wails CLI v2.12        | desktop              | <code>go install github.com/wailsapp/wails/v2/cmd/wails@v2.12.0</code>                                |
| Xcode / Android Studio | phone (local builds) | Optional: EAS builds in the cloud. Continuous Native Generation: never edit ios/ or android/ by hand. |
| EAS CLI                | phone                | <code>bunx eas-cli</code> · account owner luminex-ltd                                                 |

### Phone app

```
cd fieldra
bun install
cp .env.example .env.local      # set EXPO_PUBLIC_MAPBOX_TOKEN (pk.)
bun run build:dev:android       # EAS dev build (Expo Go can't run it)
bun start                       # expo start --dev-client
bun run typecheck               # before every commit
bun run lint                    #
bunx expo-doctor                # dependency / config health
```

### Fieldra Desktop

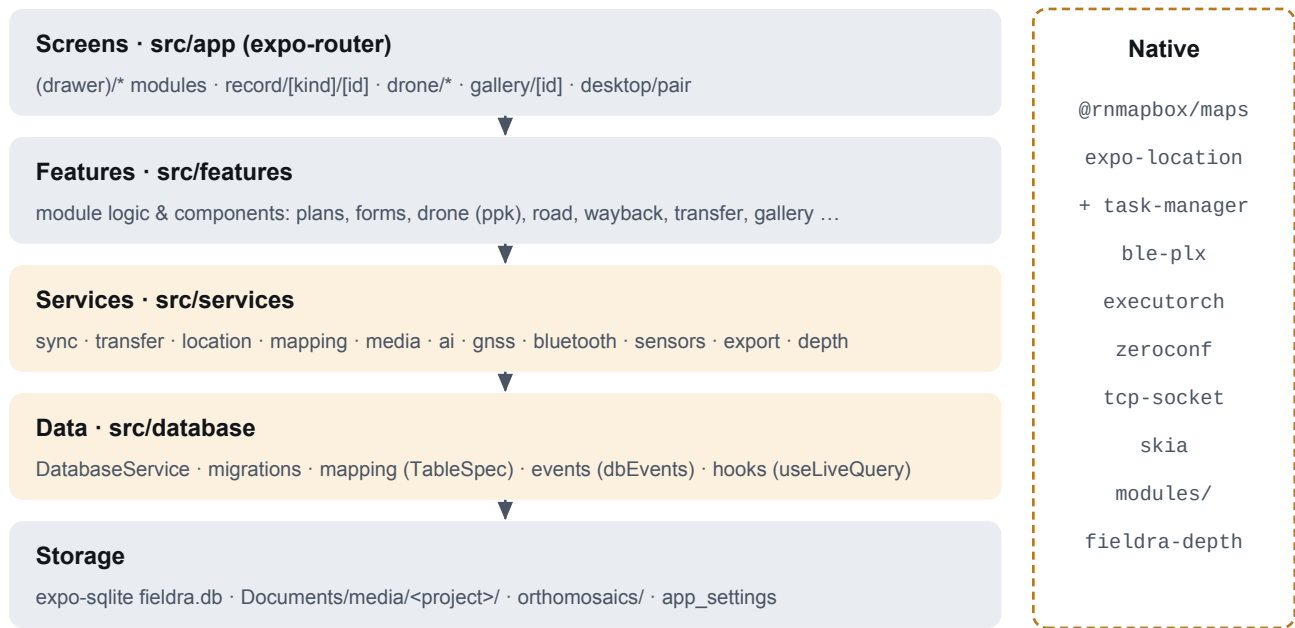
```
cd fieldra-desktop
wails dev                      # hot reload; also http://localhost:34115
go test ./internal/...        # Go tests
cd frontend && npx tsc -b       # typecheck the UI
wails build -platform darwin/arm64
scripts/release.sh all        # every build into dist/<version>/
```

**Tip:** In the browser at localhost:34115, Go bindings are available as `window.go.main.App.*`. In development builds `window.__fieldra` exposes the app store and map, and `window.__fieldraAssets` the assets store, for scripted UI checks (e.g. Puppeteer).

### Website

```
cd fieldra-web
bun install && bun run dev      # http://localhost:3000
bun run lint && bun run build
```

### 3. Phone app architecture



Layers of the phone app. Each layer only calls the one below it.

### Folder layout

| Path                               | What lives there                                                                                                                                                                      |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| src/app/                           | Routes only (expo-router). ( drawer )/ holds the module screens in the side menu; stack screens (forms, record details, viewers) sit beside it. _layout .tsx files define navigators. |
| src/features/                      | Module-specific logic and components (plans, forms, drone, road, wayback, transfer...).                                                                                               |
| src/services/                      | Cross-cutting services: sync, transfer, location, mapping, media, AI, GNSS, sensors, export, depth.                                                                                   |
| src/database/                      | SQLite access: DatabaseService, migrations, table specs, change events, hooks.                                                                                                        |
| src/components/ui/                 | App design system on @expo/ui: Button, FieldGroup, Select, TextField, Dialog, Toast, OptionSheet, BottomSheetModal.                                                                   |
| src/theme/,<br>src/types/domain.ts | Colour tokens (light/dark) and the domain types for every record.                                                                                                                     |
| modules/fieldra-depth/             | Local Expo module: LiDAR (ARKit) on iOS, ARCore Depth on Android.                                                                                                                     |
| docs/                              | PDF guides and their generators (build_guide .py, build_tech_guide .py).                                                                                                              |

## Start-up

`src/app/_layout.tsx` wraps the app in: `GestureHandlerRootView` → `KeyboardProvider` → `SQLiteProvider` `databaseName="fieldra.db"` `onInit={migrateDatabase}` → `ProjectProvider` → theme, toasts, dialogs, option sheets. It also mounts `useAutoSync()`, the plan notices, the desktop sync banner, and imports `services/location/backgroundTracking` so the background task is defined at start-up.

## Data layer

**Tables.** 24 tables: `projects`, `survey_sessions`, `locations`, `observations`, `compass_measurements`, `road_tracks`, `road_defects`, `water_samples`, `photos`, `sensor_readings`, `sync_queue`, `app_settings`, `scene_surveys`, `bio_surveys`, `bio_sightings`, `measurements`, `control_points`, `flights`, `scan_stations`, `survey_plans`, `survey_forms`, `form_responses`, `land_parcel`s, `layers`. Every record table has `id`, `project_id`, `created_at`, `updated_at`, `deleted_at`, `sync_status`. `layers` holds Fieldra Desktop layers received with a project (one row per layer: GeoJSON `geojson`, `style`, `visible`, `sort_order`), shown on the map as “Office layers”.

**Record kinds.** `RECORD_TABLES` in `database/mapping.ts` maps each kind to its table and lists which camelCase fields are JSON or booleans. `toRow/fromRow` convert between camelCase objects and snake\_case rows. Generic screens (records list, details, export) are driven by this table.

**Writes.** Always through `DatabaseService` (`createRecord`, `updateRecord`, `softDeleteRecord`, `addPhoto`, `deletePhoto`...). Each write runs in one transaction with a `sync_queue` entry (repeated edits collapse into one pending operation), then calls `dbEvents.emit()`.

**Reads.** `useLiveQuery(fn, deps)` runs a query and re-runs it after any local write — screens never poll.

**Mobile sample site.** `database/sampleProject.ts` builds 21 typed, fictional records at a fixed Freetown site. `DatabaseService.openSampleProject` checks `app_settings.sample_project_id` and creates the project, records, sync entries and marker in one queued exclusive transaction. Reopening preserves edits and renames; a deleted sample can be recreated, and a same-named user project is never adopted. `Projects` and `Welcome` expose `Open sample site`. Map draws only saved project data; changing projects remounts its workspace and starts at mapped records before falling back to GPS. `tests/sampleProject.test.js` runs the shipped migrations against real SQLite to check isolation, links, reopening, deletion and rollback.

**Mobile sizing.** `BottomSheetModal` defaults to full app width; option sheets inherit it. `textSizing.ts` defines a 16-point body base and 14-point supporting base. `Shared EmptyState` applies supporting size, line spacing and wrapping across module empty lists. `useScaledTextStyle` adjusts fixed numeric iOS native font sizes and line height for the system font scale; Android native `sp` scaling is retained. Map toolbar labels can wrap, with button width and height adapting to text size. Tests check default sizes and single scaling on iOS versus native `sp`/custom-font scaling. Native graphics require matching JavaScript and CocoaPods Expo UI versions after dependency changes.

**Migrations.** `MIGRATIONS` is an append-only list of SQL strings; the applied count is `PRAGMA user_version`; `SCHEMA_VERSION` is derived from the migration count. Never edit a shipped migration; add one.

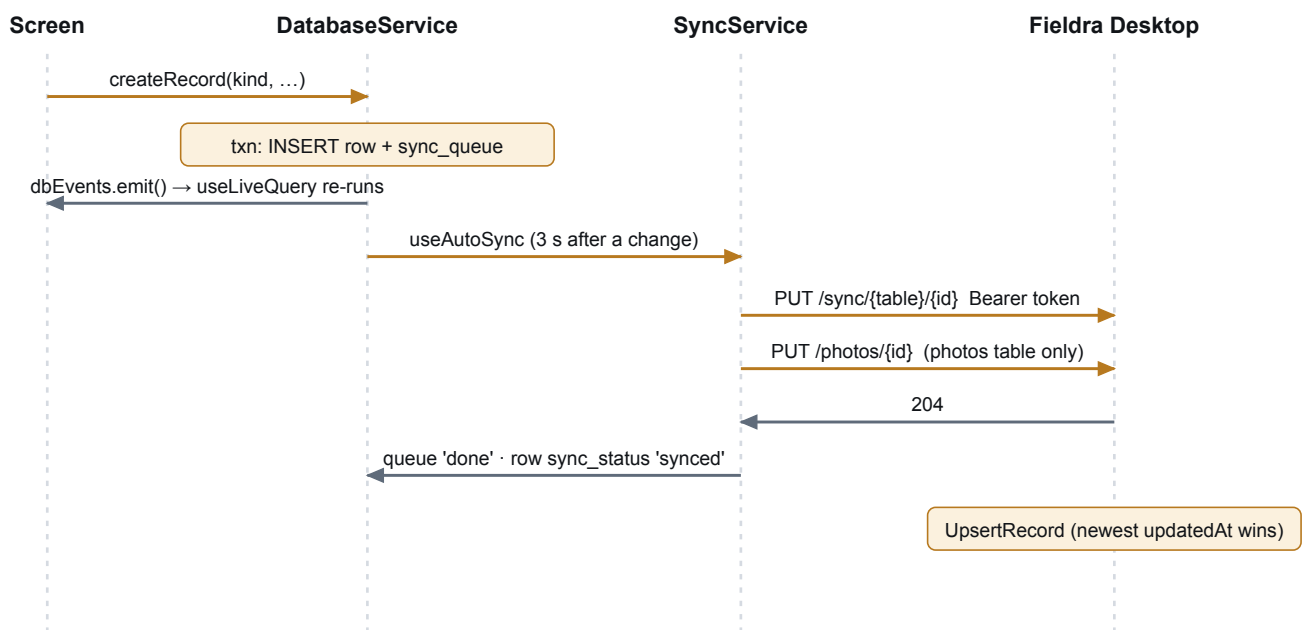
**Projects.** `DatabaseService.renameProject` (unique names, case-insensitive) and `deleteProject` (soft-deletes the project and every row with its `project_id`, and deletes its photo files); `ProjectContext` opens another project when the open one is deleted.

**Watch out:** Deletes are soft (`deleted_at`) so they can sync. Queries must filter `deleted_at IS NULL`.

## Photos and files

Captured photos are copied into `Documents/media/<projectId>/` (`MediaService`) and a photos row links them to a record (`recordKind`, `recordId`). Optionally a copy goes to the device gallery album “Fieldra” (`expo-media-library`). Received photos (transfer) land in the same folder.

## Sync to Fieldra Desktop



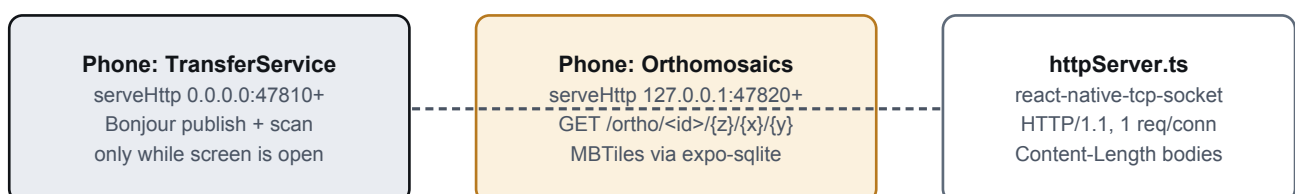
*A write, the queue, and the upload to Fieldra Desktop.*

- **Pairing** (`services/sync/desktopLink.ts`): the desktop QR holds `{fieldra:"desktop", url, code, name}`. The phone calls `POST /api/pair {code, deviceId, deviceName, platform}` and stores the returned bearer token in `app_settings`.
- **Queue runner** (`SyncService.run`): drains `sync_queue` oldest first in batches of 50, one `PUT/DELETE` per item with an `Idempotency-Key`; photos also upload the file. Items are retried up to 8 times. `X-Fieldra-Progress: n/total` and `X-Fieldra-Session` headers let the desktop show progress.
- **When** (`useAutoSync`): when connectivity returns, and — while paired — 3 s after any local change, plus every 30 s.
- **Server target**: the paired desktop or team server (typed address and code; the hello kind tells them apart), else `EXPO_PUBLIC_SYNC_API_URL`.
- **Office layers** (`services/layers/officeLayerEdits.ts`): edits to desktop layers queue as layer edits; `withShapePart` rewrites one part of a line or area from `app/layer/shape.tsx` (which also splits lines at a corner and one-ring areas between two corners, over `lays.ts`

splitRing; splitFeature gives the second part a new fid). Layers downloaded on the phone have kind phone and skip the queue. **Overlays** (features/layers/overlays.ts, pure): clip and erase with polygon-clipping for areas, lines cut at edge crossings, points by ray casting; countWithin adds count, sum\_ and mean\_ of a field. Planar in lon/lat (site scale); results are new phone layers.

- **Data files and tracks:** features/layers/layerFiles.ts (pure) reads GeoJSON (WGS 84 only), KML/KMZ (Placemark points, lines, polygons with holes, gx:Track, Data/SimpleData fields), GPX (wpt, trk, rte), CSV (lat/lon columns, comma or semicolon) and zipped Shapefiles (gisFiles.ts: .shp/.dbf/.cpg, reprojected from the .prj) and writes GeoJSON, KML and CSV; layerFileIO.ts picks and shares, and opens GeoPackages (expo-sqlite, each feature table a layer, WKB geometries reprojected from their SRS). Layers of kind phone are queued whole as table phone\_layers on every change (Fieldra Desktop stores them as layers, a team server relays them). features/tools/track.ts keeps a fix when accurate enough and at least 3 m (8 m in a vehicle) from the last; app/tools/track.tsx records through BackgroundTracking and appends to the Tracks layer. features/mapping/placeSearch.ts: Nominatim search, biased to about 50 km around the map.
- **Web layers** (services/mapping/WebLayers.ts): kept in app\_settings web\_layers; WMS and ArcGIS export templates become raster sources under the orthomosaics; **Feature service edits** (services/mapping/serviceEdits.ts): downloadService fetches ArcGIS query (f=geojson), OGC API items or WFS GetFeature (up to 20,000) and keeps ServiceSource {url, kind, base, at} in the layer style, with base a hash of each feature as downloaded. serviceChanges compares current features with it; before saving, conflicts re-downloads and flags features changed on the service too. Saves: ArcGIS applyEdits (Esri JSON), OGC API Part 4 POST/PUT/DELETE, WFS-T 2.0/1.1 Transaction (DescribeFeatureType for field order; responses parsed with regex — Hermes has no DOMParser). afterSave stores a phone|server hash pair so a server that rewrites values doesn't look like a conflict next time.

## Transfer between devices (Nearby Devices)



*The phone runs two tiny HTTP servers on react-native-tcp-socket.*

services/transfer/TransferService.ts advertises and browses \_fieldra.\_tcp with react-native-zeroconf (DNSSD on Android), and serves the transfer protocol (chapter 5) with httpServer.ts on port 47810 (next free up to +5) while the Nearby Devices screen is open. Imports filter payload keys to the table's real columns (PRAGMA table\_info), fill required timestamps, mark rows synced and do **not** enqueue them. Pending sends, accepted offers and history are kept in app\_settings (transfer\_pending, transfer\_accepted, transfer\_history).

Sends include soft-deleted rows (their payload has `deletedAt`), so deletions reach the other device; a received photo deletion also removes the local file. Counts and the “choose what to send” list leave deleted rows out. The `layers` table uses a transfer-only spec (geojson, style as JSON, visible as boolean).

**Offline maps** travel after the photos: `PUT /api/transfer/{token}/maps/{id}?offset=N` (8 MB pieces; 409 with `{have}` when `N` is not what has arrived, so the sender re-aligns) and `POST .../maps/{id}/done` with `{name, size, bounds, minzoom, maxzoom, format}`. The offer carries maps; its answer's `have.maps` gives bytes held per map (full size = complete), so a dropped send resumes mid-file. The phone appends with `File.open(FileMode.Append)` to `orthomosaics/<id>.mbtiles.part`, then `Orthomosaics.receiveDone` moves it and registers it for the project. `httpServer.ts` joins body packets once, so large bodies aren't re-copied per packet. Desktop: `internal/transfer/maps.go`; a desktop receiver keeps the map as an MBTiles asset.

**Datum shift grids** go to phones before the maps: the answer's `have.grids` lists `"code/name"`: bytes held, and the desktop sends each NTV2 (.gsb) grid of its geodesy setting that differs with `PUT /api/transfer/{token}/grids/{code}?name=...`. The phone keeps them in `documents/grids/<code>__<name>` and `ShiftGrids.loadAll` registers them at start-up. GeoTIFF grids stay on the desktop. Desktop: `Manager.Grids`, `gridFiles` in `assets.go`.

## Maps

- **Renderer:** `@rnmapbox/maps` only (`features/mapping/SurveyMap.tsx`), loaded lazily; needs `EXPO_PUBLIC_MAPBOX_TOKEN`.
- **Offline areas:** Mapbox offline packs (`services/mapping/OfflineMaps.ts`), up to 6000 tiles per area, zoom 10–17.
- **Your orthomosaics:** `Orthomosaics.ts` copies an MBTiles file into `Documents/orthomosaics/`, opens it with `expo-sqlite` and serves tiles from 127.0.0.1:47820 (TMS row flip); `SurveyMap` adds a `RasterSource` below the survey layers.
- **Survey data** is one GeoJSON `ShapeSource` built by `MapService.toGeoJSON`. Features may carry their own color (office layers, coloured per feature from the desktop's `style.renderer` — categories or graduated classes — by `services/mapping/officeStyle.ts`); the colour expression is `coalesce(color, layer colour)`. Office-layer features have kind `office_layer` and show their name when tapped instead of opening a record.
- **Map screen:** `features/mapping/MapToolbar.tsx` is the bottom tool bar (groups like the desktop ribbon, scrolls sideways); `LayersSheet.tsx` is the Contents sheet (check boxes, colours, counts, empty layers apart); `GoToSheet.tsx` parses latitude, longitude. `SurveyMap` exposes `controlRef` (`flyTo`, `fitBounds`, `measure(rect)`: ground width and bearing of a screen rectangle via `getCoordinateFromView`) and a pin.
- **Sharing:** `SurveyMap` exposes `snapshotRef` (Mapbox `takeSnap`). **Layout:** `mapLayout.ts` holds the page geometry in millimetres (A4, A3, Letter; portrait or landscape), scale bar and scale; the map screen frames that shape on the live map, crops the snapshot with `expo-image-manipulator`, and `MapLayoutModal.tsx` previews the page; `Export image` draws the preview with `Skia.makeImageFromView`. `services/export/Printables.ts` builds HTML and prints it with `expo-print` (the layout page at its paper size, response PDF with photos as base64, blank form PDF), then shares with `expo-sharing`.

- **Map tools:** `features/mapping/useMapTools.tsx` holds every shared tool (navigate, guide me, bookmarks, base map, colour by, time, terrain, measure, buffer, snap, save image, offline) for the main map and each module's expanded map. It returns `tools` (toolbar groups), `mapProps` (spread on `SurveyMap`), `top/bottom overlays` and `helpers`. Pure maths is in `mapTools.ts` (path length, snapping, buffers, time ranges, profile sampling and stats, legend colours).
- **Expanded maps** live inside the app's single option sheet, so their pickers open in `InlineSheet.tsx` (an in-map panel with the same request shape). Tapping a record there opens a menu: open, add photo, guide me, buffer.
- **Guide me** uses `features/wayback/directions.ts` (`useWalkingRoute`, Mapbox walking directions, refetched after 40 m) and `utils/navigation.ts` (`followPath`, `turnHint`); offline it falls back to a straight line. `GuideCard.tsx` turns the arrow with the heading from expo-location. `BookmarksSheet.tsx` saves views per project in `app_settings` (`map.bookmarks.<projectId>`).
- **Terrain:** `SurveyMap` props `terrain` (`RasterDemSource` `mapbox-terrain-dem-v1` + `Terrain`, exaggeration 1.5, 1 while profiling), `relief` (`HillshadeLayer`), `contours` (`VectorSource` `mapbox-terrain-v2`, source layer `contour`, labels on `index ≥ 5`). `controlRef.elevations` calls `queryTerrainElevation`; the profile fits the line on screen, waits for tiles, samples 80 points and shows `ProfilePanel.tsx`. All need internet.
- **Colour by:** `MapService.loadProject` attaches readable fields to points (water, roads, observations, sightings, and `select_one / yes_no` form answers as `form.<formId>.<question>`) and lists them in `ProjectMapData.fields`; the tool sets each point's color from `fieldLegend`. Every map item also carries `at` (`createdAt`) for the time filter.
- **Photos layer:** photos with a position become points of kind `photo` (`PHOTO_KIND`); tapping one opens the viewer (`features/gallery/openMapPhoto.ts`). `features/gallery/addPhoto.ts` adds photos to a record or to the project (`record_kind = project`, as on the desktop), several at once from the library (`MediaService.pickMany`).
- **Android:** embedded (compact) maps use a `TextureView` (`surfaceView={false}`) and mount after interactions, because a `GLSurfaceView` inside the Compose-hosted form could crash on first open. `Mapbox.UserLocation` is only rendered once location is allowed.

## Location, icons and messages

- **Performance (phone):** `dbEvents.emit` coalesces writes within 50 ms into one version bump, so a burst (a track saving every fix, edits in a row) re-runs open queries once. `MapService.loadProject` is shared per project and version, so the dashboard, module maps and Map do one load; desktop layers are parsed once per change (`officeParsed`, keyed by `id` and `updated_at`).
- **Performance (desktop):** `reuseUnchangedLayers` keeps the `FeatureCollection` of every phone-data table whose records did not change, so MapLibre only re-receives changed layers; full-screen views (Layout, Forms, Dashboard, Compare, Georeference, Photos, Devices, Accuracy, Land survey) are `React.lazy` chunks, taking the start-up bundle from 5.5 to 3.4 MB.
- **Crash reports:** `services/crash/sentry.ts` runs `Sentry.init` before anything else (imported first in the root layout), only with `EXPO_PUBLIC_SENTRY_DSN`; no personal data. `Sentry.wrap(RootLayout)` and the Expo Router navigation container add taps and screens. Source maps upload at build time with `SENTRY_ORG / SENTRY_PROJECT /`

SENTRY\_AUTH\_TOKEN (metro.config.js uses getSentryExpoConfig); the development profile skips uploads. Reports are tagged with the running EAS update (expo-update-id, expo-update-group-id, a link to it on expo.dev), and the update:\* scripts run sentry-expo-upload-sourcemaps dist after eas update.

- **3D Orbit:** features/mapping/OrbitBall.tsx, shown in 3D or whenever the map is tilted. Gesture-handler pans call controlRef.orbit(heading, pitch) (Mapbox setCamera without animation); free drag turns and tilts, side handles only turn, top and bottom only tilt, double-tap is flat north-up. The map reports the settled camera through onViewChange. Fieldra Desktop has the same ball (components/app/OrbitBall.tsx, MapLibre jumpTo).
- **Map toolbar:** Liquid Glass (expo-glass-effect GlassView) on iOS 26+, elsewhere a translucent surface with a light rim and no Android elevation (a shadow shows through translucent views).
- **Location permission:** services/location/locationPermission.ts. The app asks once after the welcome survey (location.askedAtStart); useLocation asks again on a GPS screen at most once per launch while the system allows it, and restarts GPS when permission arrives (after asking or back from Settings). Map tools ask through ensureLocationPermission.
- **Icons:** import SymbolView from components/ui/Symbol, not expo-symbols: SF Symbols on iOS, Material Symbols **Rounded** on Android (@expo-google-fonts/material-symbols-rounded). SF Symbols may only be used on Apple platforms. Option sheets take an optional symbol per option; Select sheets use radio buttons, MultiSelect check boxes.
- **Toasts over modals:** a native Modal is a separate window, so BottomSheetModal, the layout modal and the transfer modals each mount a ToastHost; the newest host shows messages and the root one takes over when they close.

## Land survey (compass and tape)

- **Record:** land\_parcel → land\_parcells: parcel details, declination, tape length, the first beacon (GeoFix from averaged GPS), the field book as JSON legs (TraverseLeg: bearing, back bearing, source and spread, tape count and remainder, slope, beacon mark, GPS check), and a stored summary (geometry, perimeter, area, misclosure, precision) worked out on every change by features/land/parcells.ts.
- **Maths:** features/land/traverse.ts has no imports and is copied unchanged to the desktop (frontend/src/lib/traverse.ts): circular mean and spread of compass samples, slope reduction, forward/back mean with a 2° local-attraction flag, declination, local plane E/N from the first beacon, Bowditch adjustment, precision grades (1:1000 good, 1:300 fair), shoelace area and inside angles. Keep both copies identical.
- **Screens:** app/(drawer)/land (list), app/land/new (details; also edits), app/land/[id] (Skia TraverseSketch, closure, field book, beacons), app/land/leg (bearing capture, line guide, tape counter, slope, back bearing). useBearingCapture averages 3 s of heading or DeviceMotion pitch. LandReport.ts makes the PDF (SVG plan) and the beacon CSV.
- **Survey calculators:** features/tools/coords.ts (UTM inverse, MGRS both ways — checked against GeoTrans values — and a parser for DD, DMS, UTM and MGRS, also used by the map's Go to) and features/tools/crs.ts (proj4 with every EPSG system as epsg.json, code/name/proj4/area, ≈1.2 MB, loaded on first use; definitionOf swaps towgs84/datum for +nadgrids=key when shiftGrids.ts has loaded the system's NTV2 grid

with proj4.nadgrid, keeping the datum's ellipsoid), features/tools/cogo.ts (inverse, offset point; resection and intersection by weighted Gauss–Newton least squares on bearings and distances together, a 95 % error ellipse ( $K = \sqrt{5.991}$ ) and residuals; seeded grid/random/stratified sampling with Neyman allocation; sampleSize for a margin at 90/95/99 %, simple or stratified; declination from expo-location true – magnetic heading). Screens app/tools/coords, cogo and sampling; points are measurement records of type point, sampling points are survey\_plan points.

## Location, GNSS and sensors

- LocationService (expo-location) for fixes and watches; useLocation for screens.
- backgroundTracking.ts: one TaskManager task (fieldra-line-tracking) shared by road tracks and transects; Android foreground service with notification; falls back to foreground watching without “Always” permission.
- External RTK receivers: BLE NMEA via react-native-ble-plx (services/gnss), averaged in useGnssAveraging.
- Compass, barometer, magnetometer, sound and roughness in services/sensors.

## AI

On-device models via react-native-executorch: a CLIP-style classifier for land cover, waste and vegetation, and an object counter (services/ai). Models are downloaded once in Settings (about 250 MB) and run offline. AI output is only ever a suggestion; the user's answer is stored separately.

## Native module: fieldra-depth

| Platform | Implementation                                                     | Emits (onMeasurement, ~5 Hz)                               |
|----------|--------------------------------------------------------------------|------------------------------------------------------------|
| iOS      | ARKit sceneDepth / smoothedSceneDepth (LiDAR devices)              | tracking, distance, confidence 0–2, x/y/z in session space |
| Android  | ARCore 1.56 Depth API (AUTOMATIC), GLSurfaceView camera background | same fields; y is up (gravity)                             |

**Watch out:** The Android implementation has not been compiled or run on a device yet; ARCore APIs were checked against the 1.56.0 library. Build it with `eas build --profile development --platform android` and test on an ARCore phone.

## UI conventions

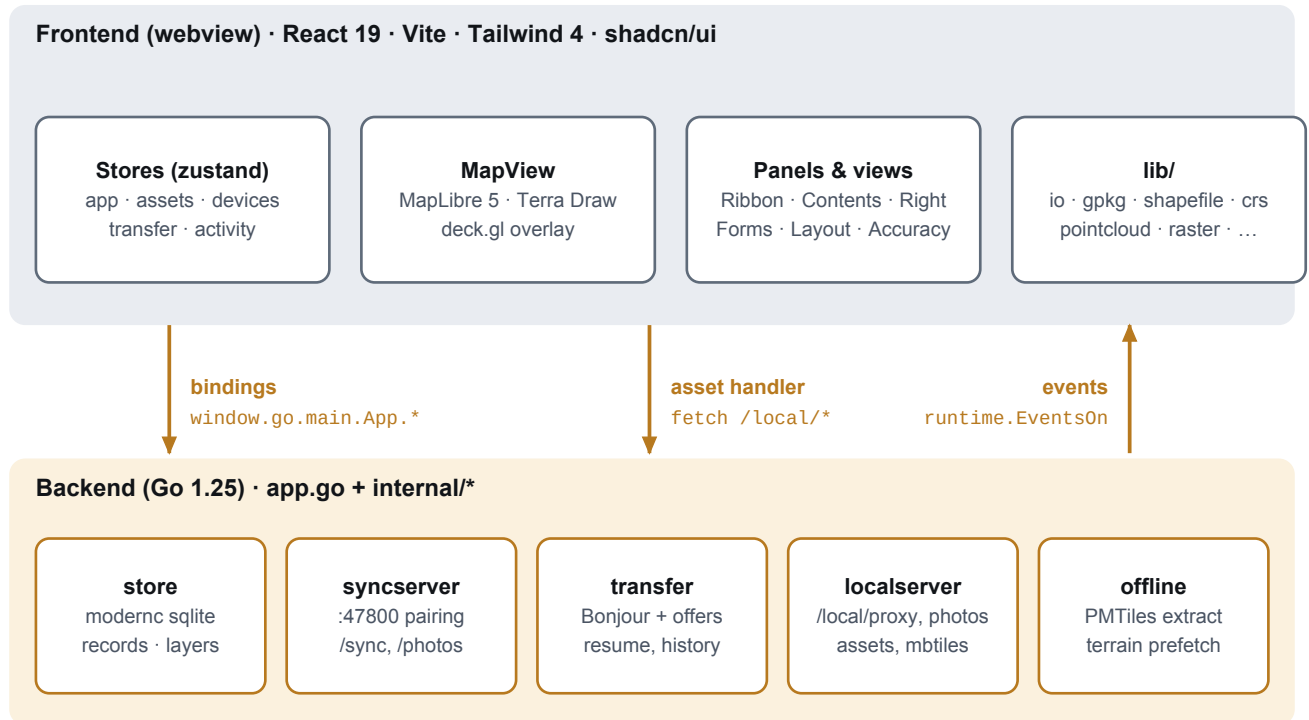
- Use the components in components/ui (they wrap @expo/ui for native iOS/Android look) and colours from useTheme().
- Forms: FieldGroup sections; scrolling screens use KeyboardAwareScrollView so inputs avoid the keyboard.
- Choice questions: ChoiceList lists every option with radio buttons (choose one, yes/no) or check boxes (choose several).
- Android buttons: Compose paddingAll and border take whole dp, so the outline of outlined buttons is 1 dp (0.75 rounded to 0 and vanished).

- Android system bars: expo-navigation-bar with enforceContrast: false and `<NavigationBar style="auto"/>`; the root window background follows the theme (expo-system-ui), so the bottom inset matches light and dark mode.
- Confirmations: `Alert.alert` from `components/ui/Dialog`; destructive buttons render black.
- Help: add a topic to `features/help.ts` keyed by the route name; words for the (i) glossary in `features/glossary.ts`.

## Configuration and releases (phone)

| Item        | Where                                                                                                                |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| Identity    | app.config.ts: owner luminex-ltd, bundle/package com.luminex.fieldra, scheme fieldra                                 |
| Secrets     | .env.local / EAS secrets: EXPO_PUBLIC_MAPBOX_TOKEN, EXPO_PUBLIC_SYNC_API_URL (optional)                              |
| Permissions | app.config.ts plugins: location (background on), camera, media library, Bonjour _fieldra._tcp, local networking      |
| Builds      | eas.json profiles: development (dev client), preview (APK, simulator), beta, production (app bundle, auto-increment) |
| OTA updates | expo-updates, runtimeVersion policy fingerprint; channels staging, beta, production                                  |

## 4. Fieldra Desktop architecture



*The webview UI and the Go backend, and the three ways they talk.*

## Go packages

| Package / file           | Responsibility                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| main.go                  | Opens the store in the user config dir (macOS ~/Library/Application Support/Fieldra, Windows %AppData%\Fieldra), starts Wails with the embedded frontend, asset handler, bindings and (macOS) menu.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| app.go                   | The App struct bound to the UI; startup wires syncserver + transfer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| internal/store           | SQLite (modernnc, pure Go): records(table_name, id, project_id, payload, updated_at, deleted, device_id, received_at), devices, layers (with project_id and a deleted_at marker), settings; photos in photos/<id>.<ext>. DeleteRecord stamps deletedAt/updatedAt into the payload so deletions win in transfers.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| internal/syncserver      | HTTP on :47800 (configurable): hello, pairing (6-digit code → token), record/photo upload from phones; emits sync:event.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| internal/transfer        | Bonjour register/browse (grandcat/zeroconf), offer/accept, sending and receiving projects, resume, history.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| internal/localserver     | Wails asset handler for /local/*: tile proxy with disk cache, offline basemaps, photos, assets (PUT/GET/DELETE with ranges), MBTiles tiles, and file geodatabases the user opened (/local/gdb/<token>/<file>, read-only, geodatabase file names only).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| internal/mssql, mssql.go | Microsoft SQL Server / Azure SQL through microsoft/go-mssqldb (database/sql). Tables from sys.tables with the first geometry/geography column (SRID and type from a row); shapes read with STAsBinary and written with STGeomFromWKB(...).MakeValid() (own WKB codec in wkb.go); columns read by type (decimals as float, dates as ISO text, GUIDs as text, binary left out). No reprojection on the server: shapes come in the table system and lib/dbClient.ts converts them, map boxes and edits. Write: fid INT IDENTITY, multi-row INSERTs under 2,000 parameters, a GEOMETRY_AUTO_GRID spatial index over the data. Live editing: row fingerprint = md5 of the values read; Apply reads each row WITH (UPDLOCK, HOLDLOCK), compares, then updates or deletes; inserts return the key with OUTPUT ... INTO a table variable; SAVE TRANSACTION per change. Tests script a mocked server with go-sqlmock (each connection the package opens is the next mock).                                                                                        |
| internal/postgis         | pgx v5. Tables, Read/Write, live editing (ReadLive, Apply: md5(t::text) row fingerprints, a savepoint per change). Versioning (versioning.go): <schema>.fieldra_history (editor, db_user, row_key, op, old/new jsonb, note) written in the same transaction as each saved change; Restore rebuilds a row from old (jsonb_populate_record, geometry via ST_GeomFromGeoJSON) with an upsert; <schema>.fieldra_proposals (changes jsonb, status open → approving → approved / rejected): Review claims an open proposal, applies it under the author with a note, and stores the per-row result. Keys may be composite (columns joined by commas; the key text is a JSON array). SetTracking adds <schema>.fieldra_track() as an AFTER trigger (editor and note from set_config('fieldra.editor') inside Fieldra's transactions); RestoreTo / RevertProposal upsert each row's earliest old state after a time or a proposal; fieldra_reviewers is checked in Review. Tests run against a real server when FIELDRA_PG_TEST="socketdir,port,db,user" is set. |
| weblayers.go             | FetchCapabilities and FetchText (≤512 MB, 3 min, Accept geo+json/json/xml) for WMS/WMTS capabilities and WFS / OGC API Features; AddWebHost lets the tile proxy cache a host.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

| Package / file                                        | Responsibility                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| internal/localserver/auth.go, rangeproxy.go           | Sign-ins per host (Auth: basic, bearer, header, query; saved in web_credentials) applied to the tile cache, capabilities and SendText; /local/range?u= streams GETs with their Range header to imagery hosts (Sentinel-2 COGs, Earth Search, the Planetary Computer and its Landsat store) and user hosts.                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| gdalconvert.go, python.go                             | GdalConvert: a data-folder or picked file through the computer's GDAL (tiler.FindGDAL) to COG GeoTIFF or GeoPackage (WGS 84). GdalSubdatasets lists a netCDF/HDF file's SUBDATASETS from gdalinfo -json; GdalConvertPart converts one (the part must name the same file). RunSystemPython: the computer's Python (FindSystemPython) runs a boot script defining an awaitable fieldra module; calls are stdout lines marked \x1eFIELDRA, answers come back on stdin (PythonReply), output and the end as python:<run> events.                                                                                                                                                                                                                                     |
| internal/localserver/pyodide.go                       | /local/pyodide/v<ver>/<file>: Python package wheels fetched from the Pyodide CDN once through the disk cache (64 MB per file), then served offline.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| internal/teamsync, teamserver.go, cmd/fieldra-team    | Optional team server: cmd/fieldra-team runs syncserver with Kind="team" and Relay=true (flags -addr :47800, -data, -code or FIELDRA_TEAM_CODE, -name, -cert/-key). It stores layer edits as layer_edits records instead of applying them. The desktop pairs like a phone (teamsync.Pair checks hello kind "team") and teamLoop pulls every 3 minutes: GET /sync/changes?since=cursor&limit (cursor "receivedAt[table id]", from store.ChangesSince), photos by GET /photos/{id}, layer edits applied with ApplyLayerEdits; the link is kept in setting team_server. Pairing locks an address out after 10 wrong codes in 15 minutes (429), codes compared in constant time. The release workflow cross-compiles Fieldra-TeamServer for Linux, Windows and macOS. |
| websecrets.go                                         | Sign-in secrets (password, token, key) in the system keychain through zalando/go-keyring (service "Fieldra"); storedAuth.InKeychain marks entries whose secret is there, and older settings entries move on first read. Falls back to the settings when the system has no keychain.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| internal/offline                                      | Offline basemap downloads (go-pmtiles extract) and terrain tile prefetch; emits offline:progress.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| menu*.go, projects.go, photos.go, assets.go, forms.go | macOS menu (cgo sendAction for text editing), new projects, photos added on the computer, point cloud / imagery metadata, survey forms made on the computer (stored as survey_forms records, version bumped on every save).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| package.go                                            | Project packages (.fieldra): a zip with manifest.json (format fieldra-project-package, version 1, counts), records.json (project record first), layers.json, assets.json, photos/ and assets/ (stored, not deflated). Import upserts records (newest wins), skips photos and asset files already present, and validates ids and file names.                                                                                                                                                                                                                                                                                                                                                                                                                      |

## Bindings (called from the UI)

Info Records Layers SaveLayer DeleteLayer SyncStatus NewPairCode RestartSync StopSync  
Devices ForgetDevice Peers AddPeer TransferProjects SendProject RespondOffer DeviceName  
SetDeviceName TransferHistory PendingSends DismissPending Basemaps DeleteBasemap  
DownloadBasemap CancelDownload SaveTextFile SaveFiles Setting SetSetting Assets  
SaveAsset DeleteAsset MBTilesInfo NativeEdit PickDataFiles AddPhotos DeletePhoto  
CreateProject SaveForm DeleteForm ExportProjectPackage ImportProjectPackage  
PickGeodatabase SaveGdb AddWebHost FetchCapabilities FetchText PostgisTables  
PostgisRead PostgisWrite PostgisPrimaryKey PostgisReadLive PostgisApply (editor)  
PostgisHistory PostgisRestore PostgisPropose PostgisProposals PostgisReview  
PostgisTracked PostgisSetTracking PostgisPruneHistory PostgisRestoreTo  
PostgisRevertProposal PostgisReviewers PostgisSetReviewers MssqlTables MssqlRead  
MssqlWrite MssqlPrimaryKey MssqlReadLive MssqlApply WebSignIns SetWebSignIn SendText  
SystemGdal GdalConvert GdalSubdatasets GdalConvertPart FindSystemPython RunSystemPython  
PythonReply StopPython TeamServer ConnectTeamServer DisconnectTeamServer PullTeamServer  
After changing a bound method's signature, run `wails generate` module to refresh  
frontend/wailsjs.

## Events (Go → UI)

| Event                                                                            | Payload / use                                                                            |
|----------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| sync:event                                                                       | record / delete / photo / paired, with progress; feeds the transfer card and map refresh |
| offline:progress                                                                 | basemap download progress                                                                |
| transfer:peers · offer · offer-expired · progress · received · pending · history | Nearby devices state                                                                     |
| menu                                                                             | menu bar actions (new-project, add-data, undo, redo, tab-*, zoom-*...)                   |

## Settings keys

device\_id, device\_name, sync\_port, assets, transfer\_accepted, transfer\_pending, transfer\_history (Go) and UI keys such as workspace, team\_server (team link and cursor), web\_credentials (sign-ins; secrets in the keychain), UI `fieldra.analysisDetail`.

## Frontend

- **State:** zustand stores — app (records, layers, selection, tools, views, history/undo, project), assets (point clouds, rasters, tiles, photo sets), devices, transfer, activity, cursor, crsPrompt.
- **Survey data → layers:** `lib/fieldra.ts` turns records into one `FeatureCollection` per table (`KIND_BY_TABLE` holds labels and colours). Layer ids: `f:<table>` (survey), user layers by id.
- **Files:** `io.ts` (router), `gpkg.ts` (sql.js + WKB), `shapefile.ts`, `crs.ts` (proj4; EPSG 4326, 3857, WGS 84 UTM), `commands.ts` (import/export/analysis).
- **Reports:** `formReports.ts` (jsPDF + autotable, docx, CSV), `printLayout.ts` (map capture at print resolution).
- **Feature services:** `lib/featureServices.ts` — OGC API collections and paged items (`rel=next`), WFS 1.0/1.1/2.0 capabilities and `GetFeature` (GeoJSON output when offered, else a GML 2/3 simple-feature parser; urn EPSG:4326 is latitude first), and an axis-order check against

the layer's WGS 84 extent; UI in `WebLayerDialog.tsx`.

- **Feature service link:** `lib/serviceLink.ts` (`_sid` service id, `_base` hash; pending inserts/updates/deletes) and `saveChanges` in `featureServices.ts`: `DescribeFeatureType` → WFS-T 1.1/2.0 Transaction (GML 3, urn EPSG:4326, fields in schema order), or OGC API POST/PUT/DELETE. WFS pages with STARTINDEX; broken or id-less GeoJSON falls back to GML.
- **Warped WMS:** `lib/wmsWarp.ts` — protocol `wmswarp://{z}/{x}/{y}?src&crs;&def;&lf;`; a 17×17 grid of the tile in the source system (proj4), one 512 px `GetMap` over its box via `/local/proxy`, bilinear grid lookup per output pixel.
- **GDAL:** `lib/gdal.worker.ts` (inline worker; `importScripts /gdal/gdal3.js` copied by the `gdal-assets` Vite plugin; takes its `onmessage` back after `gdal3.js` init) and `lib/gdalConvert.ts` (sidcar grouping, bundled first, then `GdalConvert` in Go; vectors without a system ask for one).
- **Satellite:** `lib/satellite.ts` — STAC search (POST via `SendText`): Earth Search for Sentinel-2 L2A, the Planetary Computer for Landsat C2 L2 (files signed with the collection's SAS token from `/api/sas/v1/token`, cached until near expiry); items' raster:bands scale/offset; `geotiff.js` `readRasters({bbox,width,height})` over `/local/range`. Masks from SCL classes 0, 1, 3, 8–10 or QA\_PIXEL bits 0–4, read nearest and grown by one pixel; composites read every scene onto one grid and take the per-band median.
- **Cartography:** `lib/annotation.ts` (label anchors, labels → annotation points, draw-order sort keys for fill/line/circle/symbol); PDF layers in `printLayout.ts` (two captures with base or data layers hidden; OCGs created in `jsPDF`'s `putResources`, `/Properties` added in `putXObjectDict`, `/OCProperties` in `putCatalog`, content wrapped in `BDC/EMC`).
- **Analysis additions:** `network.ts` (end-point connectivity, level fields, forbidden turns with a directed-edge search and a check onto the destination segment; time-of-day costs per edge and period, taken at the time the edge is entered (labels are arrival times); U-turn policy enforced in the directed-edge search, dead ends by node degree, and at stops by leaving towards the end ahead); `kriging.ts` (universal kriging with a scaled linear/quadratic drift, residual variogram, directional variograms for anisotropy); `rasterMore.ts` `panSharpen`; `multiband toGeoTiff`.
- **Vehicle routing:** `lib/vrp.ts` — regret-2 insertion with per-route cached insertion costs, then relocate between routes and 2-opt within them until no gain or the time limit (3 s by default); one schedule simulator checks load along the route, several windows, the break (taken while waiting when possible), reload trips (−1 markers), skills and pick-up → delivery pairs; cost matrix from `odMatrix` or haversine at a speed. A fixed cost per used vehicle keeps the fleet small.
- **Live PostGIS UI:** `LiveLayersBar.tsx` (Save, Propose, History, auto refresh) and `LiveVersionsDialog.tsx` (history diff, restore, proposal preview/approve/reject); the editor name is the `editorName` setting.
- **Plugins:** `plugins.go` keeps them in `<data>/plugins/<id>` (install from a zip, folder or `plugin.json` with path checks, 50 MB / 2,000 files; a new version swapped in by rename; starter templates), `lib/plugins.ts` checks `plugin.json` and tool specs, `lib/plugin.worker.ts` runs a JavaScript plugin (`registerTool`, `registerCommand`, the script calls), `lib/pluginRuntime.ts` (`JsPlugin`, `pluginToolDef`: inputs as plain `FeatureCollections` and grids, results checked and rebuilt) and `lib/pluginHost.ts` (`zustand usePlugins`; tools as `<plugin>/<tool>` via `setTools`, commands in `commandRegistry`; Python tools through a dedicated Pyodide worker, queued, or `RunPluginPython`). Off plugins are in the setting `plugins.disabled`. `lib/scriptHost.ts` answers the `fieldra` calls of scripts and plugins

alike.

- **Python:** `lib/python.worker.ts` loads Pyodide from `/pyodide/` (copied from `node_modules/pyodide` into `dist` by the `pyodide-assets` Vite plugin; served by middleware in `dev`) with `packageBaseUrl = /local/pyodide/v<ver>/`; a `fieldra` module calls the page over `postMessage` with JSON in and out; the worker stays loaded between runs. The Pyodide version is pinned in `package.json` (the CDN path depends on it).
- **Land survey:** `landSurvey.ts` (parcels from `land_parcels` records, office declination or start point, UTM beacon schedule, jsPDF report with a vector plan, CSV, GeoJSON) and `LandSurveyView.tsx`; the arithmetic is the phone's `traverse.ts`. jsPDF's built-in Helvetica has no prime or arrow glyphs, so the report writes `'" →` as `'" -`.
- **Drone:** `pointcloud.ts` (LAS/LAZ header + EPSG from GeoKeys/WKT, loaders.gl `laz-rs/laz-perf`, `METER_OFFSETS`, colour modes), `raster.ts` (geotiff.js range reads, preview  $\leq 4096$  px, DSM ramp + hillshade), `dronephotos.ts` (exifr + DJI XMP  $\rightarrow$  footprints, GSD, overlap, coverage grid), `accuracy.ts` (ASPRS 2023 maths, CSV parsing, PDF), `assetImport.ts` (flows, CRS prompt), `AssetLayers.tsx` (map drawing), `terrain.ts` (DEM read at  $\leq 2048$  px and cached; cut/fill volume with best-fit-plane or flat bases, profiles, marching-squares contours, change between two DEMs), `TerrainTools.tsx` and `CompareView.tsx` (two synced MapLibre maps: swipe, side by side, change).
- **Forms:** `formDesign.ts` (the phone's question helpers and household template, ported), `FormEditor.tsx`, `xlsform.ts` (a small `.xlsx` reader on `jszip` + `DOMParser`, and `XLSForm`  $\rightarrow$  Fieldra mapping with warnings).
- **Dashboard:** `dashboard.ts` (aggregates and water limits) and `DashboardView.tsx` (SVG charts with fixed colours, rasterised into the jsPDF report).
- **GIS tools (ArcGIS Pro parity):** `expression.ts` (a Pratt parser: `[field]`, `$area` `$length` `$perimeter` `$x` `$y` `$id`, AND/OR/NOT, LIKE, IN, IS NULL, BETWEEN and functions; `compile(src)  $\rightarrow$  {value, test}`, unknown functions fail at parse time). `layerFilter.ts` (`LayerStyle.query`; `shownFeatures(l)`, cached in a `WeakMap` by data and query, is what the table, exports and analysis read; the time window uses `createdAt`). `renderer.ts` (single, categories, graduated colours and sizes; Jenks, quantile and equal breaks; builds MapLibre match/step expressions and legend rows). `select.ts`, `fields.ts` (calculate field — survey layers get an editable copy; CSV/XLSX/layer joins), `overlay.ts` (pure `runOverlay` on turf: clip, intersect, union, erase, spatial join, summarize within, near, merge).
- **Raster and coordinates:** `rasterOps.ts` (a Grid type; IDW, zonal statistics, Horn slope/aspect/hillshade, band calculator, reclassify, NDVI, viewshed; results written as float32 GeoTIFF with EPSG and imported as assets). `coords.ts` (UTM, MGRS, DMS/DDM parsing and formatting, ESRI and OGC WKT for WGS 84 UTM), `cogo.ts` (bearings incl. quadrant; traverse on the UTM grid with misclosure), `chainage.ts` (station + offset along a route), `georef.ts` (least-squares affine in UTM with residuals; canvas-warped RGBA GeoTIFF), `plainTiff.ts` (a TIFF without `ModelTiepoint` / `ModelTransformation` is not placed on the map: `importAssets` hands it to Georeference, which decodes it with `geotiff.js readRGB` to a PNG, at most 6000 px a side), `snapping.ts` (vertex/edge snapping fed to Terra Draw's `snapping.toCustom`, and trace along a line), `fieldRules.ts` (`LayerStyle.fields` and meta).
- **Phone tiles:** `lib/phoneTiles.ts` cuts a raster into XYZ tiles (blocks rendered by a pool of `tileWorker.ts` module workers, one per core less one, up to 6, encoding with `OffscreenCanvas` where available; a failed block falls back to the page; one run per image shared between a send and the automatic preparation after processing): max zoom from pixel size ( $\leq 22$ ,

≤ 60,000 tiles), blocks at maxzoom=3 (2048 px) reprojected once through a 17×17 control grid (proj4) and read from the smallest fitting overview (internal or .ovr), halved with alpha weighting down to each block's tile, coarser zooms from four children. Solid tiles JPEG 0.85, edge tiles PNG; DSMs coloured with the ramp and hillshade. Written by Go (CreateMBTiles / PutTiles / FinishMBTiles, localserver/mbwriter.go) to <id>-phone.mbtiles, kept in settings.phoneTiles; made when the project is first sent (lib/offlineMaps.ts).

- **Photogrammetry options:** photogrammetry/options.go — templates  
fast/map/survey/3d/multispectral → ODM args: always cog, build-overviews, optimize-disk-space; orthophoto/dem-resolution (0 → 0.1, capped at GSD by ODM); dsm, dtm+pc-classify, pc-quality, feature-quality, gltf+texturing-single-material (else skip-3dmodel unless use-3dmesh), 3d-tiles, use-3dmesh, radiometric-calibration camera+sun, rolling-shutter, boundary (one-polygon FeatureCollection, WGS 84) or auto-boundary, orthophoto-kmz, no-gpu, split 400 / split-overlap 150 above 1,500 photos. GCPs upload as gcp\_list.txt (EPSG header; x y z px py image name) from lib/gcp.ts: points parsed from CSV/TXT, lon/lat written in their UTM zone, candidate photos and first guesses from a nadir pinhole model (GPS, relative altitude, gimbal yaw, 35 mm focal), check points left out; marks saved per folder (gcp:<path> setting). stats.json → Report (shots used, GSD, area, points, reprojection px, GPS/GCP RMS and per-GCP errors, warnings). all.zip also yields report.pdf, shots.geojson (camera positions layer), the GLB, model.zip (odm\_texturing \*\_geo\*), tiles3d.zip and the KMZ. Local engine (Windows x64, no Docker): native.go reads the latest ODM release from the GitHub API, downloads ODM\_Setup\_<v>.exe (resumable, Range) and runs it with /VERYSILENT /SUPPRESSMSGBOXES /NORESTART /CURRENTUSER /DIR=%LOCALAPPDATA%\Fieldra\Engine\ODM (Inno Setup, per-user; it may install the VC++ runtime), unpacks NodeODM's nodeodm-windows-x64.zip, and starts nodeodm.exe --odm\_path ... --port 3000 detached (DETACHED\_PROCESS | CREATE\_NEW\_PROCESS\_GROUP, no window; PID in nodeodm.pid, log in nodeodm.log). Started again when Fieldra opens; other PCs reach it on :3000. Macs (no native ODM build): engine\_darwin.go, built for macOS only, finds docker (also Docker Desktop's paths), checks docker info (memory), pulls opendronemap/nodeodm and runs container fieldra-odm (--restart unless-stopped, -p 3000:3000). Events odm:engine.
- **Photogrammetry:** internal/photogrammetry drives a NodeODM server: /task/new/init with those options, photos streamed from disk four at a time with retries, commit, /task/{uuid}/info polled every 10 s (status 10/20/30/40/50), all.zip downloaded and orthophoto, dsm, dtm and georeferenced LAZ extracted. Jobs are kept in photogrammetry/jobs.json and followed again after a restart. The UI adds results through PhotogrammetryResults (picked-file URLs, copied on disk) and MarkPhotogrammetryAdded records completion while retaining the original downloads until explicit job removal. Events: odm:job.
- **Job recovery:** job JSON is written through serialized atomic replacement and deep-copied snapshots. Cancellation is terminal; removal waits for workers. Interrupted uploads are marked failed on restart, while submitted tasks resume polling. HTTP 401/403/404 stops polling with an actionable error. Extraction is context-aware; 3D Tiles retain their GLB payloads. Result imports run through a serial queue with per-file receipts and a captured project ID. Vector saves are awaited before recording successful import. The dialog exposes per-file progress, persistent errors and retry. GCP import requires finite heights and unique point names, and always uses the selected CRS; folder changes cannot restore marks from an earlier request.

- **Native tiler:** `internal/tiler` — a GeoTIFF/BigTIFF reader (tiles or strips; none, LZW, Deflate, ZSTD, JPEG with JPEGTables/YCbCr, WebP, PackBits; predictors 2 and 3; planar; 1-bit masks; internal overviews and `.ovr`), checked against GDAL on 12 layouts. The app sends a control grid (16 steps per block tile, pixel positions computed with proj4 in the image's CRS), so Go needs no projection library. Block concurrency uses an estimated 512 MiB scratch budget (separate shared 512 MiB chunk cache, smallest fitting level, bilinear with nodata/mask re-weighting, percentile stretch for 16-bit/float, ramp + hillshade for DSMs), JPEG for solid tiles and PNG at edges; one SQLite writer. Fixture comparisons allow a mean difference of up to 6 colour levels from GDAL after JPEG encoding; throughput depends on the source and hardware. Files it can't read go to GDAL's own tools when installed (QGIS, OSGeo4W, Homebrew: `gdalwarp` → VRT, `gdal_translate` MBTILES, `gdaladdo`; `gdaldem` color-relief for DSMs); else the JS tiler. Tiles are made when a raster is added, shown at full resolution on the desktop and sent to phones.
- **LiDAR:** `lib/lidar.ts` (in `lidarWorker.ts`) — progressive morphological filter on a lowest-point grid (van Herk min/max, windows doubling to the largest building, slope threshold), Laplace-filled holes; noise by 3×3-cell IQR; DSM (max) / DTM (mean of ground, filled) rasters written as GeoTIFF; class by polygon and height; LAS 1.2 writer with GeoKeys. `internal/lidar` rewrites every record of a LAS 1.0–1.4 file (formats 0–10) from the ground surface and area edits, streaming.
- **3D model:** ODM GLB (Draco, Cesium\_RTC centre in its `proj.txt` CRS) placed at the centre and turned by the grid convergence; `deck.gl` `ScenographLayer` with the Draco decoder shipped in `public/draco`.
- **Quick map:** `photogrammetry/previews.go` makes 1024 px previews on every core (MPF embedded preview when present); `lib/quickMap.ts` draws each nadir photo's central 85% on its footprint (GPS, relative height, gimbal yaw, 35 mm focal) into a north-up canvas in a local frame linear in degrees, saved as an RGBA GeoTIFF in EPSG:4326.
- **Large files:** `lib/importSource.ts` — an `ImportSource` is a browser File or a file picked in the system dialog (`/local/picked/<token>`, served from disk; `PickDataFiles` returns name, size and URL, never base64). `readRange` slices either, `toFile` fetches one whole for formats parsed whole, `storeFile` copies into `assets/` by `?from=<token>` (on disk) or in 16 MB `PUT ?offset=N[&done=1]` chunks (the server checks the offset, writes `.part`, renames at the end). Reads use `?range=start-end` (the web view can drop Range headers): `geotiff.js` through `LocalRangeClient` (from `CustomClient`), LAS through `readLasSampled` (32 MB slices, every n-th record, formats 0–10). LAZ is decoded whole ( $\leq 1.8$  GB). Photos are sent to `AddPhotos` one at a time.
- **Coordinate systems:** `lib/epsg.ts` fetches `public/epsg.json` ( $\approx 6,900$  projected and geographic EPSG systems: proj4, OGC WKT, area, bbox, shift accuracy) as a lazy chunk at start-up; `defForCode`, `epsgName`, `esriWkt/ogcWkt` fall back to it. Imports, exports and saved assets await `ensureEpsg()`. Regenerate with `scripts/build_epsg.py` (`epsg-index` + PROJ's `proj.db`): grid shifts (+nadgrids) become EPSG's best Helmert transformation; 43 systems stay approximate. User datum shifts (`datumShifts` in the workspace) override `+towgs84` and the WKT TOWGS84.
- **Kriging:** `lib/kriging.ts` — empirical semivariogram (12 lags to half the largest distance), weighted least-squares fit of spherical/exponential/Gaussian (nugget  $\geq 0$ ), ordinary kriging from the k nearest points (grid index) with the kriging standard error, leave-one-out cross-validation.
- **Data rules:** `fieldRules.ts` adds subtypes (`LayerStyle.subtypes`: field, kinds with defaults and pick-lists; used when drawing (`MapView`) and editing (`RightPane`)); `topology.ts` checks

LayerStyle.topology rules with turf (overlap, gaps from the union's holes, kinks, dangles within a tolerance, must-be-inside, duplicates) and returns error geometries for an error layer.

- **File geodatabases:** filegdb.ts reads .gdbtable/.gdbtablx per the open FGDB specification (GDAL's OpenFileGDB notes): catalog (a00000001), GDB\_Items for WKIDs, field descriptions, null bitmaps, varint geometry blobs (points, multipoints, lines, polygons, Z; curves as vertices), sparse row maps. Folders come through PickGeodatabase and the /local/gdb route; zipped .gdb through JSZip. Checked against GDAL on ArcGIS-made data.
- **Reading big GeoTIFFs:** raster.ts readScaled is used for previews, analysis grids and elevation models. It takes an overview inside the file (COG) first, then a GDAL .ovr stored with the asset (<id>.ovr, RasterSettings.overview, chosen together with the .tif), else the full image in strips of about 512 rows above 60 MP. A single geotiff.js read of a 1.5 GB image without overviews failed with "invalid stored block lengths"; strips work (about 90 s in Chrome for a 30,000 × 34,000 px DJI Terra orthomosaic, 9 s with its .ovr, 7 s as a COG).
- **Writing .gdb:** filegdbwrite.ts writes the catalog, spatial references, items (with DEFeatureClassInfo XML) and item relationships using GDAL's field descriptions of those system tables (filegdbTemplate.ts, made by scripts/gdb\_template.py), and one table per layer and shape type: compressed shapes (outer rings clockwise), OBJECTID index, and a spatial index (filegdbSpatialIndex.ts: one grid level, cell key = level<<62 | (x cell + 2<sup>29</sup>)<<31 | (y cell + 2<sup>29</sup>), a B-tree of 4 KB pages — leaves of up to 340 keys with row ids at offset 12 and keys at 1372, index pages with child ids from offset 8 — and a 22-byte trailer; FDO\_SHAPE in the .gdbindexes). Field rules become coded value and range domains and subtypes (filegdbDomains.ts: GPCodedValueDomain2 / GPRangeDomain2 items, DomainName in the field infos, DomainInDataset relationships); the reader turns them back into field rules with code labels. Checked by reading back with GDAL 3.8 (ogrinfo lists the domains, and a spatial filter with the index's keys wiped finds nothing, so GDAL answers through the index) and the app's own reader.
- **Editing and quality:** attributeRules.ts (calculated fields applied in updateData to features whose object changed; "insert" rules only to new features and never over a value), topology.ts (12 rules; partial line overlaps found per segment pair in a local metric frame, points-on-line and line-ends-on-points through bounding-box and grid indexes), layer history in the Go store (layer\_versions: the previous geojson kept when the data changed and the last version is older than 2 minutes, the last 20 kept; the window flushes pending saves on close via OnBeforeClose and the app:flush event).
- **Routes, co-kriging, services, 3D:** routes.ts (routes chained from pieces by route id with per-vertex measures; calibration by posts as a piecewise-linear re-measure; events placed with right-hand offsets), cokriging.ts (standardised ordinary co-kriging; the secondary rescaled to the primary by regression on collocated points; a linear model of coregionalisation sharing the primary's model and range, kept positive semi-definite, with a small nugget floor; cross-validation against kriging alone), arcgisRest.ts (FeatureServer/MapServer discovery, returnIdsOnly then objectIds batches, Esri JSON ↔ GeoJSON with ring orientation, applyEdits; MapServer cached tiles or export with {bbox-epsg-3857}), buildings.ts (the roof as the lowest of 1–4 planes over the footprint's minimum-area rectangle; footprint triangles clipped per plane, walls split where the lowest plane changes; drawn as one SimpleMeshLayer per layer; glTF export).
- **This round:** serviceLink.ts keeps a hash per feature as downloaded (bases); findConflicts re-downloads before saving and lists changed features whose server hash moved. toolboxMore.ts adds 13 shape tools. simulation.ts: sequential Gaussian

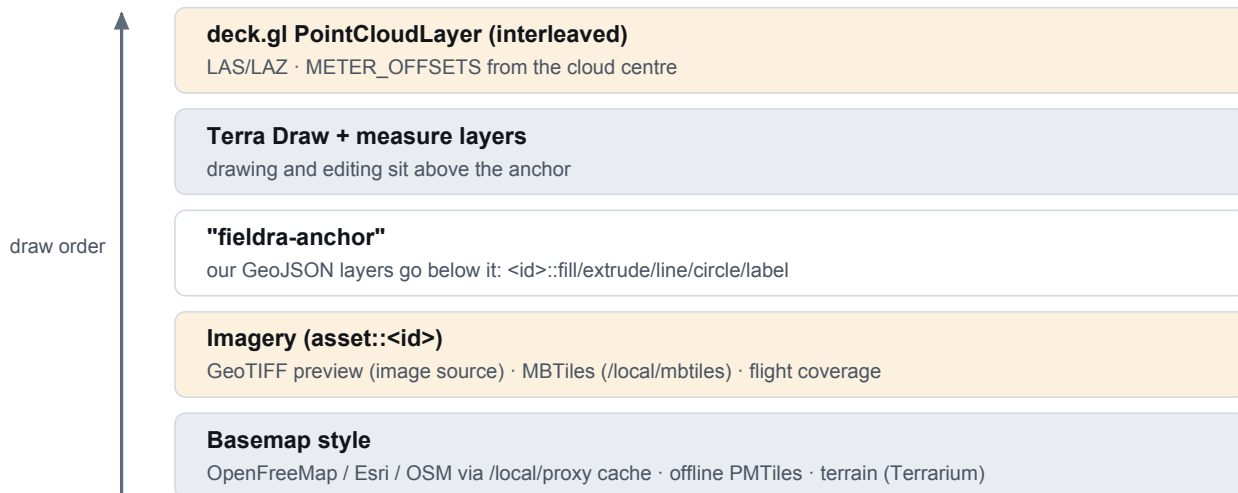
simulation — normal-score transform, a seeded random path, simple kriging (unit sill) from the nearest samples and simulated cells, back-transform; mean, standard deviation, one realisation and exceedance probability. `geocode.ts`: Nominatim search, requests 1.1 s apart, batches of 300. `annotation.ts` `labelPath`: the middle third of the longest part, drawn with symbol-placement line-center. `printLayout.ts` `pdfGroups` / `pdfLayerGroups`: one capture per map layer (others hidden), each in an optional content group (up to 16; /Order top first). `readGrid` takes up to 8,192 cells a side (the Detail choice) and keeps all bands under 80 M cells. `filegdb.ts` reads tables, `DERelationshipClassInfo` items (`parseRelationship`) into `NamedLayer.relates`, and curve modifiers (circular arcs with flags 0x08 ccw / 0x80 interior point, Bézier, elliptic; `filegdbCurves.ts`); `filegdbwrite.ts` writes `DETableInfo` tables and one-to-many relationship classes (type B606A7E1..., `DatasetsRelatedThrough` links), checked with GDAL.

- **Start-up and light mode:** `deck.gl` (`deckLib.ts`), `CodeMirror` and the LAZ decoders are dynamic imports, loaded when first needed. `performance.ts` chooses light mode for  $\leq 4$  cores or  $\leq 4$  GB: pixel ratio 1, no fades, GeoJSON tolerance 1, 1.5 M point budget.
- **Analysis:** `sampling.ts` (seeded random, grid, square/hexagon cells, Thiessen via `turf voronoi`), `spatialStats.ts` ( $G_i^*$ , Moran's I with the randomisation variance, OLS with t and F p-values from the incomplete beta function), `ebk.ts` (empirical Bayesian kriging: subsets of up to 100 points by recursive halving, each estimated from the points nearest its centre; per subset 100 Gaussian simulations from the fitted semivariogram via Cholesky, refitted and weighted by their Gaussian likelihood of the real data, 30 drawn by systematic resampling; predictions mix the drawn models' kriging means and variances, log-normal moments when on logs; seeded so results repeat), `hydrology.ts` (priority-flood fill, D8, accumulation, Strahler and Shreve order in linear time, streams split at confluences, watershed outlines traced on the cell edges; D-infinity after Tarboton (1997) with the facet clamped to its edges and accumulation in topological order; flow length up and down the D8 paths from one cached upstream-first order; the longest flow path walked up from the outlet with the Kirpich time of concentration; basins labelled by outlet; depressions as filled minus original, grouped 8-connected; wetness index  $\ln(a / \tan \beta)$  from the D-infinity area).
- **Background worker:** `background.ts` runs `computeOps.ts` (raster tools, IDW, kriging, hydrology, topology) in an inline module worker (`compute.worker.ts?worker&inline`, started from a blob URL); grids travel as plain data plus the proj4 definitions they need (`registerProj4`). If the worker can't start or crashes, jobs run on the main thread.
- **Symbols and labels:** rule-based renderers and label expressions are Fieldra expressions that MapLibre can't evaluate: `styleData.ts` adds `__rule` and `__label` to a cached copy of the drawn features; the layer's own data is never changed.
- **Layout charts:** `layoutCharts.ts` draws bar, pie and histogram charts as jsPDF vectors; `buildLayout` takes them beside the map (`chartPlacement: 'map'`) or on pages of up to four.
- **Exports:** the export CRS setting (wgs84, utm-auto or any EPSG code) reprojects `GeoPackage` (srs table), `Shapefile` (.prj) and CSV. **GeoPDF:** `printLayout.ts` hooks jsPDF's `putPage` event to write a /VP viewport with a GEO /Measure (GPTS/LPTS) for the map frame; it also draws grids, an inset, table pages and map series.
- **Layers and projects:** user layers carry `projectId`; `layers()` hides other projects' layers; drawings only go into a layer of the open project. Point layers can be drawn as a heat map or clusters (`LayerStyle.pointMode`; clusters use a second, clustering GeoJSON source).

## Layout designer and shared colour controls

`FreeLayoutView.tsx` is the free-form designer, opened through `Share › Designer or Layout › Open designer`. `freeLayout.ts` defines the page, element and template model and PDF renderer. Geometry uses page millimetres; the viewport converts them to screen pixels. `layoutGeometry.ts` holds proportional resizing with opposite edges anchored, readable ruler tick spacing and optional placement quantisation.

- **Workspace:** eight resize handles, Fit page and percentage zoom with scrollable overflow, millimetre rulers, adjustable dot grid and snapping to page/element edges and centres. Alt bypasses placement snapping; Shift-resize preserves proportions. Map panning rotates the drag into the frame camera directions.
- **Current map:** `workingLayoutKey` scopes saved working designs to the project; legacy global layouts are rebased to the current map view. Use current map applies the main map camera and visible layers to the selected or first map frame (creating one if needed), and recenters linked locators while retaining their overview scale ratio. Refresh map previews increments the capture revision without altering geometry.
- **Guides:** `FreeLayout.guides` optionally holds x/y arrays in millimetres. Ruler drags create guides, existing guides can be dragged or removed outside the page. They are saved in the `freeLayout` and `layoutTemplates` settings and scaled with a paper-size change. Guides, rulers and the editor grid are not passed to PDF drawing.
- **Inset extents:** an ordinary `MapElement` optionally stores `extentMapId` and `extentColor`. `newInset` creates an independent overview; `extentPolygon` projects the linked frame corners into overview-local millimetres, handles both rotations and the nearest world copy, and clips to the frame. The same polygon drives the SVG preview and PDF outline. A hidden source frame is not outlined; deleting it clears the link.
- **Colours:** `RichTextEditor.tsx` exports the shared `ColourPicker`, used by layout properties and `ReportDesigner`. `ColourSwatch` is declared at module scope: declaring it inside the picker recreated buttons during focus/selection updates and lost clicks between mouse-down and click. Brand, grid and recent swatches share that stable component.
- **Selection feedback:** native/custom colours apply through `onPick`; the custom value is refreshed when the picker opens. Text editors retain their selection while choosing colours. Toolbar text and highlight colours are read from rendered selection styles, walking backgrounds up to the editor boundary, so the white editing page is not treated as a highlight.
- **Checks:** `freeLayout.test.ts`, `layoutGeometry.test.ts` and `richText.test.ts` cover geometry, rotations, clipping, the antimeridian and rich-text export. Interactive checks on macOS covered preset clicks in the desktop webview and report, full/compact text, highlight removal, recent colours and None in a browser preview. Repeat the manual cases on Windows before a release.



*MapLibre style order in MapView, bottom (basemap) to top (point clouds).*

**Note:** Point clouds use deck.gl's MapboxOverlay with interleaved: true so they share the map's camera, pitch and terrain. Positions are float32 metre offsets from the cloud centre. Clouds over 3 million points get a tile set (below) and are drawn by view.

## Point clouds of any size: tiles

- **Build** (internal/pctiles): an octree like Potree's, out of core. Points are streamed once into chunk files (compact 21-byte records), chunks too full are split, then each chunk is built in memory (three at a time): a node keeps one point per cell of a  $128^3$  grid over its cube, in a shuffled order, and passes the rest to its children. The levels above the chunks take their points out of their children, so every point is in exactly one node. Output: <file>.pct/index.json and one n/d-x-y-z.bin per node (positions float32, intensity, RGB 8-bit, class). About 20 million points in 8 s with ≈600 MB at most, whatever the size.
- **LAZ** (lib/lazChunks.ts, lazStream.worker.ts): the chunk table is read with a port of LASzip's arithmetic decoder; each chunk is decoded alone with laz-perf's ChunkDecoder in a worker, and the records go to the build in batches of up to 32 MB (POST /local/pct-feed/<job>?n=, refused whole if the web view cuts it short; then FeedCloudTiles over the bridge).
- **Drawing** (lib/cloudTiles.ts): selectNodes walks down from the root while a node's points would be more than 1.5 px apart on screen (allowing for pitch), coarsest first, within the point budget; a node is drawn with all its ancestors. Tiles load six at a time into a cache of 15 million points; each is one deck.gl PointCloudLayer. The LiDAR tools work on whole top levels up to 3 million points (sampleKeys).

## Point classes and TIN editing

- **Features** (lib/lidarClasses.ts): per point, from its neighbours within an automatic radius: eigenvalue linearity, planarity, scattering and verticality, height above ground, local density and height spread. **Rules:** roofs grow as planar regions above 2 m (then completed over ridges and walls), wires are linear and thin above 3 m, the rest of the off-ground points is vegetation by height band.

- 
- **Learning:** a random forest (extra-trees style splits, balanced samples per class) trained in the worker on the points the user classed by hand, scored on a held-out part, then applied to all.
  - **Saving to the full LAS:** exact per-point classes when every point is loaded; otherwise a voxel majority grid (`lidar.Voxels`) applied in Go to points still unclassified after the ground step.
  - **TIN editing** (`lib/tin.ts`, `TinDialog.tsx`): the TIN definition is kept on the edits layer's `style.tin`; `applyTinEdits` adds, moves or deletes vertices (0.5 m match), drawn breaklines and holes join the constrained triangulation, and `makeTinOutputs` replaces the data of the existing output layers and DEM asset.

## 5. Protocols and data contracts

### Record payload

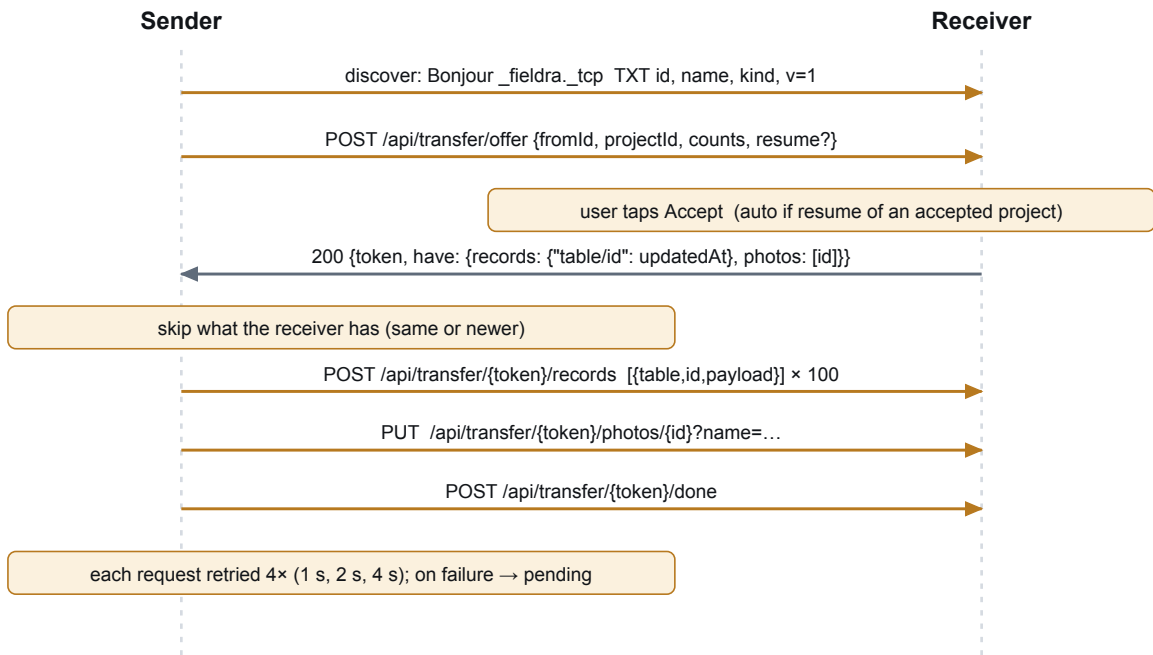
```
{
  "id": "0f9c...",           // UUID (phone) – [A-Za-z0-9_-]{1,64} on the wire
  "projectId": "b41e...",
  "createdAt": "2026-09-27T08:00:00.000Z",
  "updatedAt": "2026-09-27T08:05:12.301Z",    // conflict key: newest wins
  "deletedAt": null,           // soft delete
  "syncStatus": "pending",
  ... fields of the record kind, camelCase; JSON fields as objects
}
```

Tables on the wire are the phone's table names (water\_samples, photos...). A photo's payload has recordKind (e.g. water\_sample, or project for project photos) and recordId; the image file travels separately.

### Sync API (phone → Fieldra Desktop, port 47800)

| Request                                                         | Purpose                                                                                                                                                                                                                                                                          |
|-----------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GET /api/hello                                                  | {app, id, kind:"desktop", name, version, pairing, transfer}                                                                                                                                                                                                                      |
| POST /api/pair {code, deviceId, deviceName, platform}           | Returns {token, desktopName}. The 6-digit code is in the QR.                                                                                                                                                                                                                     |
| PUT /sync/{table}/{id} Authorization: Bearer                    | Upsert a record (newest updatedAt wins). Optional X-Fieldra-Progress, X-Fieldra-Session.                                                                                                                                                                                         |
| DELETE /sync/{table}/{id}                                       | Mark deleted.                                                                                                                                                                                                                                                                    |
| POST /sync/layers/{id}/edits {upserts:[Feature], deletes:[fid]} | Edits made on the phone to a desktop layer, merged feature by feature by properties.fid (the layer is kept in layer_versions first); 404 when the desktop no longer has the layer. Emits sync:event type "layer" with the edits, which the open window merges into its copy too. |
| PUT /photos/{id}?name=...                                       | Image bytes (≤ 40 MB).                                                                                                                                                                                                                                                           |
| GET /sync/changes?since=&limit= (team server)                   | Records changed since a cursor, oldest first, with the next cursor; for the desktop pulling from a team server.                                                                                                                                                                  |
| GET /photos/{id} (team server)                                  | A photo the team server received.                                                                                                                                                                                                                                                |
| PUT / DELETE /sync/phone_layers/{id}                            | A layer made on the phone, whole ({projectId, name, geojson, updatedAt}): stored as a layer of its project; an older copy never overwrites a newer one.                                                                                                                          |

### Transfer protocol (any device ↔ any device)



*Offer, accept, send what the receiver lacks, finish. A stopped send resumes the same way.*

| Item                  | Value                                                                                                                                                                                                                                                                  |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Discovery             | Bonjour/mDNS service <code>_fieldra._tcp</code> , instance = device id, TXT id, name, kind (phone desktop), v=1                                                                                                                                                        |
| Ports                 | Desktop: the sync port (47800). Phone: 47810 (next free up to 47815), only while Nearby Devices is open.                                                                                                                                                               |
| Offer wait            | Up to 90 s for the user to accept; 403 {error:"declined" "busy"}, 408 no answer.                                                                                                                                                                                       |
| Resume                | Offer with <code>resume: true</code> is auto-accepted if the same (fromId, projectId) was accepted in the last 7 days.                                                                                                                                                 |
| Selection             | Sender-side only: {skipTables, skip:["table/id"], noPhotos}; skipping a record skips its photos.                                                                                                                                                                       |
| Limits                | Records: 100 per request (64 MB max). Photos: 40 MB desktop, 60 MB phone.                                                                                                                                                                                              |
| Deletions             | Deleted records travel with <code>deletedAt</code> set (newest <code>updatedAt</code> wins); the receiver marks them deleted and removes photo files.                                                                                                                  |
| Layer edits on phones | The phone keeps unsent edits in <code>layers.pending_edits</code> ({upserts by fid, deletes, added}) and queues one sync item per layer; they are put back on top when the desktop sends the layer again, removed once sent unchanged, and never sent to other phones. |
| Layers                | Table <code>layers</code> : the whole layer as one record (payload = the desktop Layer: name, kind, geojson, style, visible, sortOrder, updatedAt, deletedAt). Desktops and phones both store it.                                                                      |

**Important:** The transfer endpoints are unauthenticated on the local network; the receiving user must accept each new offer. Do not expose these ports to the internet.

## 6. Website (fieldra-web)

| Path                                      | Content                                                                                                                          |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| src/app/page.tsx                          | Home: hero, modules, field features, data flow, desktop teaser, equipment, downloads.                                            |
| src/app/desktop/page.tsx                  | Fieldra Desktop: features, drone mapping, setup, store buttons.                                                                  |
| src/lib/desktop.ts                        | Desktop store links, version and requirements.                                                                                   |
| src/app/beta/page.tsx,<br>src/lib/beta.ts | Test versions (noindex): phone APK and TestFlight, every desktop file with OS/chip detection.                                    |
| src/app/admin/<br>src/components/admin/   | Admin dashboard: email/password sign-in, overview, installations (block, block email, delete, CSV), the free-use switch, admins. |
| src/lib/access.ts,<br>src/lib/firebase.ts | Survey and access logic (shared with the apps); Firebase web SDK set-up with emulator support.                                   |
| firebase/                                 | firebase.rules, indexes, firebase.json and emulator tests (bun run test:firebase).                                               |
| src/lib/stores.ts                         | App store links from env (coming-soon until set).                                                                                |
| public/*.pdf                              | The complete guide and this technical guide.                                                                                     |

Environment: NEXT\_PUBLIC\_SITE\_URL, NEXT\_PUBLIC\_PLAY\_STORE\_URL, NEXT\_PUBLIC\_APP\_STORE\_URL, NEXT\_PUBLIC\_DESKTOP\_DOWNLOAD\_URL, NEXT\_PUBLIC\_DESKTOP\_VERSION, NEXT\_PUBLIC\_MAC\_APP\_STORE\_URL, NEXT\_PUBLIC\_MICROSOFT\_STORE\_URL, NEXT\_PUBLIC\_ANDROID\_APK\_URL, NEXT\_PUBLIC\_TESTFLIGHT\_URL, NEXT\_PUBLIC\_BETA\_VERSION, NEXT\_PUBLIC\_FIREBASE\_API\_KEY, NEXT\_PUBLIC\_FIREBASE\_AUTH\_DOMAIN, NEXT\_PUBLIC\_FIREBASE\_PROJECT\_ID, NEXT\_PUBLIC\_FIREBASE\_APP\_ID — read at build time; redeploy after changes. Pushes to main deploy on Vercel. The production CSP allows the Firebase Auth and Firestore hosts for /admin.

### Welcome survey and the free-use switch (Firebase)

The apps are free and need no account. Each installation signs in to Firebase **anonymously** over REST (no native SDK, so phone changes ship as OTA updates) and owns `installs/{uid}`. `access.ts` is one file with no imports, identical in the phone app, Fieldra Desktop and the website; each app passes in its key-value store (SQLite `app_settings`, the Go settings store), SHA-256 and device details.

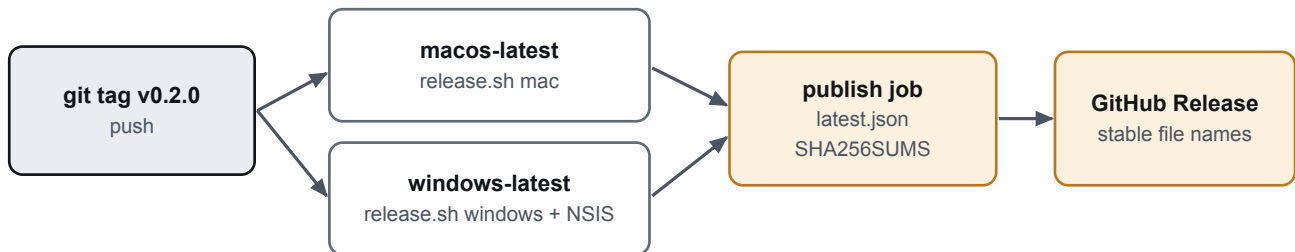
| Step         | What happens                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| First use    | The survey (name, email, organisation, reason) is validated and saved locally with a pending flag; the app is usable at once.                                                                                                                                                                                                                                                                                                      |
| Online check | <code>accounts:signUp</code> (once) or <code>securetoken:refresh</code> → <code>documents:commit</code> : create <code>installs/{uid}</code> with <code>currentDocument.exists=false</code> and server timestamps, or update only the survey/device fields ( <code>updateMask</code> ) and <code>lastSeenAt</code> → GET <code>installs/{uid}</code> , <code>config/access</code> and <code>blockedEmails/{sha256(email)}</code> . |
| Decision     | <code>decideAccess</code> : <code>freeAccess false</code> → <code>free_use_off</code> ; status blocked or email blocked → blocked; otherwise allowed. Saved locally, so it holds offline.                                                                                                                                                                                                                                          |

| Step     | What happens                                                                                                                                             |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Recovery | A deleted record (404, or 403 because the rules see a create) is re-created from the saved survey; a deleted anonymous account is replaced by a new one. |
| When     | App start, back online (NetInfo / window online), back in front, every 15 minutes. Errors keep the saved decision and are shown on the stopped screen.   |

**Rules** (firebase/firestore.rules): an installation may create only its own record as active and later change only the survey keys and lastSeenAt; users/{uid} holds dashboard accounts (email/password, not anonymous), created with role user; admins (role admin) may write anything. config/access is readable by any signed-in app. blockedEmails allows get but not list. The emulator tests cover each refusal and the whole flow.

## 7. Building and releasing

### Fieldra Desktop



fieldra-web reads .../releases/latest/download/<file> (NEXT\_PUBLIC\_DESKTOP\_DOWNLOAD\_URL)

*Tag → GitHub Actions → release with stable file names → the website links to them.*

1. Set info.productVersion in wails.json.
2. Commit, tag vX.Y.Z and push the tag. The Release workflow builds macOS (Universal, arm64, amd64: .dmg/.zip) and Windows (x64, arm64, x86 portable .zip; NSIS installers for x64, arm64 and both).
3. Set NEXT\_PUBLIC\_DESKTOP\_VERSION on Vercel and redeploy the site.

Direct downloads: add MACOS\_CERTIFICATE, MACOS\_CERTIFICATE\_PASSWORD, MACOS\_SIGN\_IDENTITY, APPLE\_ID, APPLE\_TEAM\_ID, APPLE\_APP\_PASSWORD secrets to sign and notarise.

### Mac App Store and Microsoft Store

- **Mac App Store:** scripts/release.sh mas builds a sandboxed Universal app (bundle id com.luminex.fieldra.desktop, build/darwin/entitlements.mas.plist: network client/server, user-selected files), embeds the provisioning profile, signs, and makes Fieldra-MacAppStore.pkg; with the ASC\_\* secrets the workflow uploads it to App Store Connect.
- **Microsoft Store:** scripts/release.sh msix (on Windows) packs x64, ARM64 and x86 MSIX with build/windows/msix/AppxManifest.xml into Fieldra.msixbundle; the Store signs it. Identity values come from Partner Center (repository variables MSSTORE\_\*).

- 
- Store packages are private workflow artifacts; manual runs only build. The account set-up steps are in the fieldra-desktop README.

## Phone app

```
bunx eas-cli build --profile production --platform all
bunx eas-cli submit --platform android      # Play, internal track
bunx eas-cli update --channel production    # OTA JavaScript update
```

**Watch out:** Adding or upgrading a native module changes the fingerprint: ship a new store build, not an OTA update.

## 8. How to...

### Add a new record kind (end to end)

1. **Phone types:** add the interface to `src/types/domain.ts` and the kind to `RecordKind`.
2. **Phone table:** append a migration with `CREATE TABLE` (standard columns + `project_id`) and an index on `project_id`.
3. **Mapping:** add it to `RECORD_TABLES` (table, JSON fields, booleans). Sync, export, records list and transfer then work.
4. **Labels:** `KIND_LABELS` (records/queries), the map layer (`MapService`), help text.
5. **Screen:** a route under `src/app/`; save with `DatabaseService.createRecord`.
6. **Desktop:** add the table to `FIELDRA_KINDS` in `lib/fieldra.ts` (label, colour, description) so it becomes a layer.
7. **Photos:** if records have photos, the desktop maps `recordKind` → table in `lib/photos.ts`, `SendDialog`, `PhotosView` and `Go kindTable`; irregular plurals need an entry.

### Change the phone schema

- Append to `MIGRATIONS`; never edit a shipped one. Use `ALTER TABLE ... ADD COLUMN` for new fields.
- Mention new JSON columns in the table spec. Transfer tolerates missing columns on older phones (keys are filtered).

### Add a desktop feature

- Zustand: never select a derived object (`useApp((s) => s.layer(id))`) — it returns a new object each time and loops (React error #185). Use the memoized `useLayer(id)`.
- Go: a method on `App` → `wails generate module` → call via `api.X(lib/api.ts)`.
- Long work: run in a goroutine and report with `runtime.EventsEmit`; subscribe in `App.tsx`.
- Ribbon: add a `RibbonButton` in `Ribbon.tsx`; for the menu, `menu.go` + the switch in `App.tsx`.
- Big files: upload with `PUT /local/assets/<id>.<ext>` and keep details with `SaveAsset`.

### Add a phone module to the side menu

- Route file `src/app/(drawer)/<name>/index.tsx`; entry in `SCREENS` of `(drawer)/_layout.tsx` (title, SF Symbol / Material icon).
- Help topic in `features/help.ts`; after adding routes, run `expo start` once to regenerate typed routes.

## 9. Testing and quality

| Area       | Today                                                                                                                       | Recommended next                                                                                                                                                                                                                                              |
|------------|-----------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Desktop Go | Unit/integration tests: localserver (assets, MBTiles), syncserver, transfer (real mDNS discovery, send, resume, selection). | Keep adding with each package change.                                                                                                                                                                                                                         |
| Desktop UI | TypeScript build; manual and scripted checks against <code>wails dev</code> (localhost:34115).                              | Vitest for lib/ (accuracy, crs, gpkg, pointcloud header).                                                                                                                                                                                                     |
| Phone      | TypeScript, ESLint, expo-doctor, expo export for both platforms. No tests in the repository yet.                            | Add <code>bun test</code> with mocks for expo-sqlite ( <code>bun:sqlite</code> ), file system and tcp-socket ( <code>node:net</code> ) — this pattern was used during development to test transfer, orthomosaic tiles, PPK migration and background tracking. |
| Website    | Lint and production build.                                                                                                  | —                                                                                                                                                                                                                                                             |

**Note:** Run lint and typecheck before declaring any change done (AGENTS.md). Test native features (Bonjour, background location, ARCore) on real devices; emulators do not do multicast.

## 10. Known limitations and next steps

| Area                          | Status                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Subscriptions                 | Not built. The free-use switch and stopped screen exist; signed-in users with plans (users/{uid}) and sign-in in the apps are the next step.                                                                                                                                                                                                                                                        |
| Team sync                     | Optional team server (cmd/fieldra-team), tested end to end on one machine; not yet run with a crew over mobile networks.                                                                                                                                                                                                                                                                            |
| Fine-tuned AI models          | Not started: train on own tiles, export .pte, drop into AIService.                                                                                                                                                                                                                                                                                                                                  |
| react-native-tcp-socket       | Flagged by expo-doctor as untested on the New Architecture: verify on devices.                                                                                                                                                                                                                                                                                                                      |
| Android depth                 | Implemented, not yet compiled or device-tested.                                                                                                                                                                                                                                                                                                                                                     |
| Desktop distribution          | Mac App Store and Microsoft Store set up; waiting for the store accounts' certificates, profile and listings. Direct downloads are unsigned until certificates are added.                                                                                                                                                                                                                           |
| Project deletion              | Deleting a whole project stays on the device where it was deleted (record deletions do travel).                                                                                                                                                                                                                                                                                                     |
| New screens                   | Desktop terrain tools, compare, dashboard, XLSForm import, heat maps and the GIS tools; phone PDFs and office layers: built and unit-checked, not yet tried on devices.                                                                                                                                                                                                                             |
| GeoPDF                        | Structure checked; not yet opened in ArcGIS / QGIS / Avenza. Only flat north-up frames are georeferenced (not 3D or rotated views).                                                                                                                                                                                                                                                                 |
| Raster tools                  | Supported block tools (terrain, focal statistics, reclassification, calculator and zonal statistics) process full-resolution row bands. Other tools use the selected analysis grid, up to 8,192 px. Native tile concurrency is an estimate, not a hard process-memory cap; very large TIFF strips/windows can exceed it. Native webview interaction and real dataset limits still need measurement. |
| Writing .gdb                  | Checked against GDAL 3.8 (domains, spatial index, tables, relationship classes), not yet opened in ArcGIS Pro. 2D only; relationship classes one-to-many; subtypes are written only for whole-number subtype fields (GDAL ignores subtypes, so they are unchecked).                                                                                                                                 |
| ArcGIS REST                   | Checked against a stand-in service (the network here blocks ArcGIS servers). Paged by object id in batches of up to 500; applyEdits with rollbackOnFailure.                                                                                                                                                                                                                                         |
| Phone edits of desktop layers | Merge tested in Go and TypeScript; the phone screens (values, points, reshaping vertex by vertex, web layers) typecheck and lint but were not run on a device.                                                                                                                                                                                                                                      |
| Reading .gdb                  | Multipatch, rasters and annotation are skipped; curves are densified (2° steps, 24 for Bézier); 64-bit OBJECTID tables with gaps (ArcGIS Pro 3.2+) are read only in the documented layout.                                                                                                                                                                                                          |
| Datum shifts                  | 43 EPSG systems whose only official shift is a grid file are approximate unless a grid is added (desktop: NTv2 or GeoTIFF; phones: NTv2 sent from the desktop); the picker warns. Users can enter their own 7 parameters.                                                                                                                                                                           |
| New phone tools               | COGO with distances, sample size, feature-service saves (against stand-in ArcGIS, WFS-T and OGC responses), grids to phones, area splits and overlays: unit-tested and typechecked, not yet run on a device or a live service.                                                                                                                                                                      |
| Phone receiving               | Only while the Nearby Devices screen is open (by design; no background server).                                                                                                                                                                                                                                                                                                                     |
| Flight check                  | Assumes flat ground at take-off height; estimates only.                                                                                                                                                                                                                                                                                                                                             |

| Area           | Status                                                                                                                                                                                                               |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Photogrammetry | Available through OpenDroneMap / NodeODM. Recovery and imports are checked against a simulated server. A real flight, survey accuracy and comparative throughput against Pix4D or ArcGIS Pro remain to be validated. |

## Performance review: 16 October 2026

Point-cloud tile selection now uses the existing typed-array priority queue instead of repeatedly scanning every candidate. On an Apple M4, a synthetic 21,845-node index took a median 30 ms instead of 264 ms over five measured runs after warm-up. This measures tile selection only, not rendered frame rate or full cloud loading.

High-bit-depth contrast preparation samples decoded TIFF chunks through a separate 32 MiB cache, avoiding a full-image float window. In the 4,096 by 4,096 RGB 16-bit synthetic benchmark, total allocated bytes fell from about 535 MB to 327 MB (39%); this is cumulative allocation, not peak resident memory. Large native tile blocks share an estimated 512 MiB scratch budget; small blocks retain CPU parallelism. Cancellation is checked during rendering and while waiting for worker slots; database write failures stop further production, partial output is removed, and an existing completed output is preserved on cancellation. Large polygon zonal bounds and mobile map bounds use streaming extrema rather than argument spreading.

Keep Go for orchestration and file streaming. Existing Rust/WebAssembly kernels cover expensive numerical work; benchmark representative workloads before any further port. These changes do not establish overall performance or accuracy parity with commercial photogrammetry or GIS packages.

## 11. Terms

| Term          | Meaning here                                                                                                        |
|---------------|---------------------------------------------------------------------------------------------------------------------|
| CNG           | Continuous Native Generation: ios/ and android/ are generated from app.config.ts and plugins; never edited by hand. |
| Dev client    | A development build containing the app's native modules (Expo Go cannot run Fieldra).                               |
| Fingerprint   | Hash of native code/config; OTA updates only reach builds with the same fingerprint.                                |
| Binding       | A Go method on App callable from the webview.                                                                       |
| METER_OFFSETS | deck.gl coordinate system: positions in metres from a lon/lat origin.                                               |
| TMS / XYZ     | Tile row order: MBTiles stores rows bottom-up (TMS); web maps use top-down (XYZ).                                   |
| ASPRS 2023    | Positional Accuracy Standards, Edition 2: RMSE_H, RMSE_V, survey error included.                                    |