RACETRAK IMTUF

RaceTrak Operations Manual

Offline-first race tracking for ultra-marathons in country with no signal

Revision	2026-08-12 · `1303eb4`
Applies to	Field app (Android & browser), command console, portal
Audience	Part 1–2 race director and installer · Part 3 aid station and radio operators

PART 1

Installation

This part is for whoever stands the system up: one race director, once, at a desk with a network. Nobody working an aid station needs to read it.

Work through it in order. Each step is verifiable, and the verification is given, because the failure mode this system must never have is one that stays hidden until a tablet is at mile 45 with no signal.

1.1 What the system is made of

Five things run, and they are all the same event log seen from different places.

Piece	Where it runs	What it is for
Supabase Postgres	Hosted	The source of truth. Every event ends up here.
PowerSync	Hosted	Replicates Postgres to each device, and decides what each device is allowed to receive.
Field app	Android tablets	What an aid station uses. Works offline for hours.
Field app (browser)	Any phone or laptop	The same screens, for a station with signal and no tablet.
Command console	Browser, at basecamp	The whole race at once. Privileged views, roster import.
Portal	Browser, at a desk	Setting a race up: accounts, codes, imports.

Two facts about this shape are worth holding on to, because most of the procedures below only make sense in their light.

Every client holds a real database. Not a cache — a local SQLite database with the race in it. A tablet that loses signal keeps working, keeps recording, and sends its work when it can. This is why the install order below puts the backend first: a client cannot be meaningfully tested until there is something for it to replicate.

What a device may see is decided by the server, not the app. A station tablet does not hide a runner's phone number; it never receives it. That boundary lives in `powersync/sync-rules.yaml`, and §2.7 covers how to verify it rather than assume it.

> **NOTE**
>
> The event log is append-only, enforced by a database trigger. Nothing in this manual will ever ask you to edit or delete an event, because the database will refuse. Mistakes are corrected by recording a correction — see §3.6.

1.2 Before you start

You need, in this order:

1. **Node.js and npm**, on the machine doing the install. The repo is an npm workspace; everything below is an npm script.
2. **A Supabase account** and a project for the environment you are building. Free tier is sufficient for a race.
3. **A PowerSync account** and an instance pointed at that Supabase project. Free tier is sufficient.
4. **A Cloudflare account**, for hosting the three web apps.
5. **Android tablets**, if you are using them, with developer mode enabled so builds can be installed over USB or wireless debugging.

> **CAUTION**
>
> PowerSync sits **on top of** Supabase and the wiring between them is the one part general Supabase familiarity does not cover: logical replication has to be enabled on Postgres, PowerSync pointed at it, and Supabase Auth tokens threaded through so the sync rules know who is calling. Budget an unhurried block for this, well before the race — not the week of it.

1.3 Getting the code

```
git clone <your fork of the repository>
cd RaceTrak-IMTUF
npm run bootstrap
```

`npm run bootstrap` installs dependencies for every workspace at once. Do not run `npm install` inside an individual app; the workspaces share a single lockfile.

Confirm the domain logic is sound before going further:

```
npm run test
npm run lint
```

The test suite covers the event-log folds — the code that answers "where is bib 147" — and runs in seconds with no backend. If it fails, stop; nothing downstream will behave.

1.4 The backend

`docs/backend-setup.md` is the authority on this and goes deeper than a manual should. The shape of it:

1. **Create the Supabase project** and record its reference id and database password in `infra/env/<env>.env`. Start from `infra/env/example.env`; the real files are gitignored and never committed.
2. **Apply the schema.** `npm run supabase:migrate dev` runs every migration in `supabase/migrations/` in order. This creates the event table, the row-level security policies, and the append-only triggers.
3. **Deploy the pairing function.** `npm run supabase:functions dev` publishes `redeem-pairing-code`, which is how every client — tablet, console and portal — turns a code into a session. Nothing can pair until this exists.
4. **Enable logical replication** on the Postgres instance and create the read-only replication role PowerSync will connect as.
5. **Point PowerSync at it**, then deploy the sync rules with `npm run powersync:deploy dev`.

Current schema state:

Generated from the source of record. Do not edit by hand.

16 migrations, through `0016_note_events.sql`.

> **CAUTION**
>
> `npm run powersync:deploy <env>` publishes the RBAC boundary. Never deploy it from a dirty working tree — what you deploy should be exactly what is committed, because this is the file that decides whether a phone number reaches a volunteer's tablet.

1.4.1 Verifying the backend

Do not take the console's word for it. Two checks, both cheap:

- **Replication is live.** The PowerSync instance should report a connected replication slot with lag at or near zero.
- **The append-only rule holds.** Attempt an `UPDATE` and a `DELETE` against `events` as a privileged user. Both must be refused by trigger. If either succeeds, the migrations did not all apply.

1.5 The web clients

Three separate apps, three separate Cloudflare Pages projects per environment. They are separate on purpose: the portal writes directly to Postgres and is used at a desk, while the console replicates through PowerSync and keeps working when the uplink does not.

```
npm run build:web dev          # command console
npm run build:portal dev       # portal
npm run build:fieldweb dev     # field client in a browser

npm run deploy:web dev
npm run deploy:portal dev
npm run deploy:fieldweb dev
```

Every deploy targets the project's production branch, so each app keeps one stable hostname that does not change when you deploy. Cloudflare also mints a per-deployment URL with a hash in front of it — those are build artifacts, not addresses.

> **STOP**
>
> A deployment-specific URL (`abc12345.<project>.pages.dev`) is a **different origin** from the project URL, and a client's local database belongs to the origin that created it. An operator who pairs against a hashed URL and later opens the plain one finds an empty, unpaired app, with their unsynced work stranded at the old address. Only ever hand out the plain project URL.

> **CAUTION**
>
> These apps must be served over HTTPS. PowerSync guards its database with the Web Locks API, which browsers refuse outside a secure context — over plain HTTP the app loads and then never opens its database. This is also why, in local development, you use `http://localhost:5173` and not the LAN address the dev server also prints.

1.6 The tablets

The field app is React Native under Expo. The `android/` and `ios/` directories are generated by `expo prebuild` and are not committed — never edit them by hand, because the next prebuild discards the change. Anything that must reach the native project goes through `app.json` or a config plugin.

```
RACETRAK_BUILD_VARIANT=release npm run build:android dev
adb install -r build/android/dev/app-release.apk
```

> **CAUTION**
>
> Build **release**, not debug. A debug build fetches its JavaScript bundle from a development server on the network, which means it needs a laptop within reach to start at all. That is fine on a bench and useless at an aid station. Only a release build is a real test.

1.7 Verifying the whole chain

The install is not finished when things are deployed. It is finished when one tablet has carried a record all the way to Postgres. Do this once, deliberately:

1. Install a release build on a tablet that has never been paired.

2. In **Settings → Race pairing**, enter a station code from the portal.
3. Confirm the tablet syncs down: the roster appears and the station name is shown in the header.
4. Record one arrival.
5. Query the `events` table in Postgres and find that row, carrying the tablet's own device id.

That path — code, function, session, sync rules, local database, screen, upload queue, Postgres — is the system. If it completes, the install is good.

1.8 Commands, in full

Every script takes an environment (`dev`, `staging`, `prod`) and prompts before touching production.

Generated from the source of record. Do not edit by hand.

Command	Runs
`npm run bootstrap`	`./scripts/common/bootstrap.sh`
`npm run supabase:start`	`./scripts/supabase/start-local.sh`
`npm run supabase:migrate`	`./scripts/supabase/migrate.sh`
`npm run supabase:functions`	`./scripts/supabase/deploy-functions.sh`
`npm run powersync:deploy`	`./scripts/powersync/deploy-sync-rules.sh`
`npm run console:dev`	`./scripts/web/dev-web.sh`
`npm run build:web`	`./scripts/web/build-web.sh`
`npm run studio`	`./scripts/android/studio.sh`
`npm run build:android`	`./scripts/android/build-android.sh`
`npm run build:ios`	`./scripts/ios/build-eas.sh`
`npm run deploy:web`	`./scripts/web/deploy-web.sh`
`npm run deploy:portal`	`./scripts/web/deploy-portal.sh`
`npm run submit:android`	`./scripts/android/submit-android.sh`
`npm run submit:ios`	`./scripts/ios/submit-ios.sh`
`npm run test`	`npm run test --workspace @racetrak/core`
`npm run portal:dev`	`./scripts/web/dev-portal.sh`
`npm run build:portal`	`./scripts/web/build-portal.sh`
`npm run fieldweb:dev`	`./scripts/web/dev-field-web.sh`
`npm run build:fieldweb`	`./scripts/web/build-field-web.sh`
`npm run deploy:fieldweb`	`./scripts/web/deploy-field-web.sh`
`npm run lint`	`eslint .`
`npm run manual:build`	`node scripts/manual/build.mjs`
`npm run manual:check`	`node scripts/manual/check.mjs`

Command	Runs
`npm run deploy:manual`	`./scripts/web/deploy-manual.sh`

PART 2

Configuration Management

This part is for the race director and anyone helping set a race up. It covers what you configure, where each setting lives, and which settings are safety boundaries rather than preferences.

The dividing line worth learning first: **the portal is where a race is set up, the console is where it is run.** The portal writes straight to Postgres at a desk with a network. The console replicates through PowerSync and keeps working when the uplink does not. They share every importer and every calculation, so they cannot disagree about what a spreadsheet row means or where a runner is.

2.1 Environments

Three environments, separated by *project* rather than by branch: a separate Supabase project, PowerSync instance and set of Cloudflare Pages projects for each. That separation is what gives each environment a stable hostname and makes it impossible to point a production tablet at development data by accident.

Environment	For
`dev`	Everyday work. Test races, throwaway data, permissive tokens.
`staging`	Optional rehearsal against production-shaped configuration.
`prod`	The race.

Configuration lives in `infra/env/<env>.env`. Only `example.env` is committed — the real files hold credentials and are gitignored. Copy the example, fill it in, and keep the production file somewhere a second person can reach it.

> **STOP**
>
> Development environments are deliberately permissive: temporary sync tokens are allowed and a preset pairing may be baked in so a client can connect before real credentials exist. Neither may be true in production. §2.9 is the checklist, and it is not optional.

2.2 Setting a race up

In the portal, in this order. Each step depends on the one before it.

1. **Create an account** and sign in.
2. **Create the race** — name and date.
3. **Add the aid stations**, in course order. Order matters: it is what makes “this runner never left the previous checkpoint” a question the app can ask.
4. **Import the roster.**
5. **Add the crews** — who is working which station.
6. **Generate the station codes** and distribute them.

All three imports — aid stations, crews and roster — accept a spreadsheet and offer a downloadable template, either blank or filled with the race’s current data. The filled template is the easier path for an edit: download, change, re-import.

2.2.1 Aid stations

Each station carries a name, its order on the course, and optionally a mile mark, opening hours, coordinates, and tags such as `radio`, `water`, `medical` or `drop_bags`. Cutoff times are set here too, and are what the field app counts down against.

2.2.2 The roster

Imported from CSV, with a mapping step: your file’s column headings are matched onto the fields the system needs — bib number, name, age, phone. Unpredictable headings are the norm and the mapper exists for exactly that; it can also combine two columns into one name.

> **NOTE**
>
> A station may assert that a bib **exists**; only the console may say who is wearing it. A tired volunteer typing a runner’s name at 3am is not a data source. This is enforced in the database, not just the interface: the insert policy refuses a row that carries a name, so identity cannot come in through the one door left open for unknown bibs.

2.2.3 Codes

Two kinds of code exist and they are not interchangeable.

Code	Grants	Given to
Station code	One station, one race. Records movements there.	Aid station and radio volunteers
Share code	The whole race, at a privileged role.	Race director, medical command

One code per station, carrying whether that station is `in`, `out` or `radio`. The code says *where* the device is; the operator chooses what the device is *doing* (§2.8). Codes are generated in the portal — one button produces one code per station — and can be sent by SMS or email.

> **CAUTION**
>
> Station codes are read aloud over radio and written on clipboards. Treat them as visible. They scope a device to one station and nothing more, which is what makes that acceptable — but a share code must never be handled the same way.

2.3 Roles, and what each one receives

This is the security model. It is enforced in the sync rules, which decide what each device ever receives, and again in row-level security in the database.

Generated from the source of record. Do not edit by hand.

Role	What it is	Sync buckets received
`aid_station`	Works a checkpoint. Receives the roster without phone numbers, the whole event log, and emergency contacts.	`race_core`
`radio`	Net Control duty at a checkpoint. Same data as an aid station; the tablet is set to Radio so the queue is its default screen.	`race_core`
`race_director`	Runs the race. Everything an aid station receives, plus runner contact details, the pairing codes and the membership list.	`race_core`, `race_contacts`, `race_admin`
`medical_command`	Medical oversight. The same privileged view as the director, for the same reason: they may need to reach a runner.	`race_core`, `race_contacts`, `race_admin`

The important property is negative and worth stating plainly: **a station device does not receive runners' phone numbers at all.** They are not hidden in the interface — the rows are absent from the tablet's database. There is nothing to leak, nothing to render by accident, and nothing recoverable by someone holding the device.

Emergency contacts are different, and are sent to station devices deliberately: the moment one is needed is at the checkpoint where somebody is on the ground, not at basecamp. Revealing one is a deliberate act that writes a permanent record of who looked (§3.6).

2.3.1 Verifying the boundary

Do not infer this from the configuration file. Confirm it:

1. Redeem a real station code and, with the session it returns, query the contacts table. It must come back empty.
2. With the same session, attempt to insert a runner. It must be refused.
3. On a paired tablet, confirm the local database holds zero contact rows.

If all three hold, both layers agree independently, which is the only form of this assurance worth having.

2.4 What a tablet is set to

Pairing decides the race and the station. Everything else is chosen on the device, in **Settings**, and can be changed mid-race.

Setting	What it controls
Race pairing	The code this device redeemed. Determines race, station and role.
Aid station	Which station this device is working.
Operator	Who is holding it this shift, and their radio call sign.
This device is	The default direction — see below.
Display	Whether runner names are shown, grid density, clock seconds.
Station mode	Station lock and its PIN.
Station Wi-Fi sync	Peer sync with the other tablet at this station.

2.4.1 Device role

Generated from the source of record. Do not edit by hand.

Setting	What the tablet says it does	When to use it
IN	Tap records an arrival	Set this on the tablet working arrivals.
OUT	Tap records a departure	Set this on the tablet working departures.
Radio	Relaying to Net Control	The only role that is offered the Radio screen.
Other	No default direction	Nothing is recorded without an explicit choice.

The device role sets what a single tap on the Record grid does. It is a default, not a restriction — every other action stays available behind a long press. An operator can override the direction for their shift, and that choice outranks the paired role.

> **NOTE**
>
> Only a device set to **Radio** is offered the Radio screen. A station running IN and OUT tablets should not have the call-in queue competing for space on either of them.

2.4.2 Display: names are off by default

A station screen is a public surface — propped on a table, photographed, visible to anyone in the tent. Runner names are personal data, and the bib is what the job actually needs, so names are off unless someone turns them on.

2.4.3 Station lock

Station lock keeps a tablet on the recording screens so it cannot be wandered away from mid-shift. It is unlocked with a PIN, or with the race admin code.

> **CAUTION**
>
> Station lock is an operational guard, not a security control. The PIN is stored in the clear on the device, and anyone holding the tablet can clear its data and get back in — which also destroys any records that have not yet synced. Real lockdown needs Android screen pinning or a device policy.

2.5 Two tablets at one station

The IN and OUT operators are routinely a hundred feet or more apart — too far to share a tablet, and exactly the pair who most need each other's data. When both are paired to the same station, they find each other over the station's Wi-Fi and exchange events directly, with no internet involved.

The receiving device decides what it will accept: events for another race, unknown types, or events about nobody are refused regardless of what the sender claims.

> **CAUTION**
>
> This works between **tablets only**. The browser field client has no peer sync, because a browser cannot open the necessary network connections. A station running the web client at a dark checkpoint is isolated until signal returns.

2.6 Before a real race

Development settings are permissive by design. Every one of these must be changed before production carries real runners.

Gate	Why
A production Supabase project and PowerSync instance exist	Development data and production data must never share a database.
Temporary sync tokens **disabled** on the production instance	They let a client connect without proving who it is.
Preset pairing empty	Otherwise a build ships already paired to something.
Email confirmations **on**, with SMTP configured	Otherwise a portal signup is trusted without a verified address.
Release builds signed with a real keystore	The debug keystore is not an identity.
Offline token lifetime checked against the real race window	A 30-hour race with dark stations must not outlive its credentials.

> **STOP**
>
> On the last point: recording must **never** block on an expired credential. A client with no valid token is required to keep recording locally and sync later. Whatever you change about token lifetime, that behaviour has to survive, because the alternative is a station that silently stops working in the middle of the night.

PART 3

Operations

This part is for the people working a race: aid station operators, radio operators, and the director at basecamp. It assumes nothing about the rest of this manual.

Three things are true of this app, and they explain most of how it behaves.

It works with no signal. Everything you record is saved on the tablet the instant you record it. Sync happens in the background whenever there is connectivity. You never wait for the network, and losing it costs you nothing.

Nothing is ever deleted. Mistakes are fixed by recording a correction, so the log always shows both what was written and that it was put right. You cannot destroy a record by getting something wrong.

It will not stop you. Unknown bibs, duplicates, runners who appear out of order — all of these are normal at an aid station. The app tells you what it noticed and lets you decide. It never blocks a record.

3.1 Starting your shift

1. **Check the tablet is paired.** The station name appears at the top of the screen. If it says the device needs pairing, get the station code from the race director and enter it in **Settings → Race pairing**.
2. **Set your name.** Tap **Set operator** at the top, or go to **Settings → Operator**. This is how a record is attributed to you; it is not a login.
3. **Check the sync status.** Also at the top: `Synced`, `Syncing…` or `Offline`. Offline is not a problem — it is the expected state at most stations. It means what you record is waiting on the tablet, which is safe.
4. **Check what the tablet is set to do. Settings → This device is** decides what a single tap records. Make sure it matches the job you are doing.

> **NOTE**
>
> There is no password and no login. The tablet is what is authorised; your name is for attribution. A login prompt at 3am in a canyon with no signal would be a way to lose data, not a way to protect it.

3.2 The screens

Generated from the source of record. Do not edit by hand.

Tab	What it is for
Keypad	Type a bib, confirm the name, record IN or OUT. The fallback when a list is too long to scan.
Record	One square per bib. One tap logs the default direction; a long press opens everything else. The workhorse.
Log	One row per bib — in, out, called in, state. Tap a row for the full history and every correction.
Radio	The queue of movements not yet passed to Net Control, with multi-select and batch call-in. Only on a Radio device.
Race	Read-only reference at the station: race details, bib ranges, roster, stations, crews, cutoffs.
Settings	Pairing, operator name, device role, display, station lock, backend.

Whatever you type on the Keypad survives switching tabs and coming back. An operator moving between IN and OUT halfway through a bib number does not lose the digits they had already entered.

3.3 Recording arrivals and departures

Two ways, and they are for different moments.

3.3.1 The Record grid — the fast way

A square for every bib. One tap records the direction the tablet is set to. That is the whole interaction, and it is what you will use for hundreds of runners.

- **Tap** a square to record the default direction.
- **Long press** a square for everything else — corrections, notes, DNF, the runner’s history.
- Switch between **Present** and **All** to narrow the grid to runners currently at your station.
- **Tap = IN / Tap = OUT** at the top shows what a tap will do, and can be changed for your shift.

3.3.2 The Keypad — when the grid is too long

Type a bib. The runner's name appears as confirmation. Then **Record IN**, **Record OUT**, or **Other** for anything else.

Recording OUT for a runner who was never recorded IN will record both, so a runner passing straight through is one action rather than two.

> **NOTE**
>
> After every record, a short **UNDO** appears at the bottom of the screen. Tapping it retracts what you just did. It is there for a few seconds and is the fastest way to fix a slip — no menus.

3.4 What the app tells you as you type

As you enter a bib, the app assesses it and tells you what it noticed. None of these stop you.

What you see	What it means	What to do
The runner's name	The bib is on the roster.	Record normally.
Not on the roster	No runner with this bib. A bandit, a typo, or a late registration.	Record it anyway. It is saved against the bib number and matched to a person later.
Already recorded here	This runner is already logged in at your station and has not left.	Usually a double-tap. If they really did arrive twice, record it.
Out of sequence	They never left the previous station.	Almost always a missed record upstream, not your problem to solve. Record and carry on.

> **NOTE**
>
> "Not Arrived" means the app has not seen this runner reach your station. It does not mean anything is wrong.

3.5 Radio: calling in to Net Control

The Radio screen shows movements that have **not yet been passed to Net Control**. It is a queue: work from the top.

1. Read the queue. Oldest first, except that drops jump the queue — a DNF is the thing Net Control most needs to hear.
2. Select the bibs you are about to transmit. Several at once is normal — real traffic is “bibs 47, 112 and 203 through Aid 4”, not one at a time.
3. Call them in.
4. Mark them called in. They leave the queue.

Each bib appears as one line per occasion. If the same bib is recorded several times within a few minutes, those collapse into one line with a count — that is two operators confirming the same runner, not two passes.

3.5.1 When a bib comes back

A bib that genuinely appears a second time is flagged, because it is more often a correction than a second lap:

- **Amber** — your station has two times for this runner and has transmitted neither. Check which one is right before calling either in.
- **Red** — the earlier time has already gone to Net Control. The flag names the old time and when it was sent, so you can correct the record over the air rather than adding a second one.

> **CAUTION**
>
> Two operators marking the same batch as called in is not a problem and needs no coordination. It is recorded as two separate acts and the queue treats the batch as sent. Do not hold off calling something in because you think someone else might have.

3.6 Fixing mistakes

Long press a bib on the Record grid, or tap its row in the Log, or use **Other** on the Keypad. Every correction is recorded as a correction: the original stays visible in the history and is marked as superseded.

Action	Use it when
UNDO (the toast)	You have just this second mis-tapped.
Correct time	The record is right, the time is wrong.
Reset bib	The records at your station for this bib should not exist. Clears them so the bib can be recorded again.
Add a note	Something needs explaining. Changes nothing about the record.
DNF	The runner stopped. Asks where — often a station behind yours.
DNS	The runner never started.
History	Everything recorded for this bib, including corrections.

3.6.1 Correcting a time

Operators log in bursts and the tablet's clock is not the runner's clock. A time recorded a few minutes late is normal and worth correcting.

When more than one time could be the one you mean — an arrival, a departure, a radio call-in — the app asks **which time is wrong** rather than guessing. Pick the one you mean, then set the correct time.

> NOTE
>
> If a correction would be overruled by another time sitting after it, the app says so before writing anything, and offers to move the others with it. Say yes if they are all the same mistake.

3.6.2 Notes

A note is for the context that otherwise lives in somebody's head until they go off shift: the bib was upside down and might be 52, the runner said their knee had gone, the time came off a radio call rather than the tablet.

A note changes nothing. It is safe to add at any moment, including in the middle of a correction, and the director sees it in the console alongside the record it explains.

3.6.3 Emergency contact

A runner's emergency contact can be revealed on a station tablet. It is deliberately a two-step action, and **it permanently records who looked and when**. That record cannot be removed by anyone.

Use it when someone is on the ground and you need to reach their people. Do not use it out of curiosity.

> **NOTE**
>
> The runner's own phone number is not on your tablet at all, and no amount of tapping will produce it. Only the director and medical command receive it.

3.7 Working with no signal

This is the normal state at most stations, and nothing about it needs managing.

- Everything you record is saved on the tablet immediately.
- The status at the top will read `Offline`. Records accumulate safely.
- When signal returns, they upload on their own. You do not have to do anything.
- If a second tablet is working your station on the same Wi-Fi, the two share records with each other even with no internet at all.

> **STOP**
>
> Do not clear the app's data, and do not uninstall the app, while records are waiting to sync. That is the one action that destroys a station's work, and nothing can recover it. If a tablet is misbehaving, hand it to the race director rather than resetting it.

3.8 When something looks wrong

What you see	What it usually is	What to do
Screen says the device needs pairing	Not paired, or app data was cleared.	Enter the station code from the director.
Status stuck on `Offline`	No signal. Normal.	Nothing. Keep recording.
Status stuck on `Syncing…` with signal	Weak connection.	Nothing. It retries on its own.
A time shows a weekday, or a small mark after it	The record is from a **previous day**, not today.	Nothing — it is telling you that deliberately.
A runner is at a station they cannot have reached	Usually a mistyped bib somewhere upstream.	Add a note. Do not delete anything.
A bib you recorded is not in the Radio queue	Somebody already called it in.	Check the bib's history.
The tablet will not leave the recording screen	Station lock is on.	Unlock PIN, or the race admin code.

> **CAUTION**
>
> A race that runs more than 24 hours passes the same clock time twice, so a bare `21:48` is ambiguous. Every time in this app that is not from today is marked — either with its weekday, or with a small mark where a weekday will not fit. If you are reading a time to Net Control, read the day too.

3.9 For the race director on race day

The command console shows the whole race at once, and keeps working from its own local copy if basecamp loses connectivity.

- **Whole race** — every runner's position across every station, sortable, with full history per runner. Tallies, cutoffs and unknown bibs. Every count is clickable through to the bibs behind it, and the field can be filtered to one station.
- **Aid stations** — the course, and the import wizard for changing it.

- **Crews** — station personnel.
- **Roster** — CSV import with the column mapper.

Two jobs only the console can do:

Naming unknown bibs. A bandit recorded at a station is saved against the bib number with no name. Matching them to a person is a console job, because a station is not permitted to assign identity.

Reading the contact details. The runner's own phone number reaches the director and medical command and nowhere else.

> **NOTE**
>
> The console gates what it shows on what the code you redeemed actually granted. If you signed in with a station code you will not see the privileged views — that is the boundary working, not a fault.

RaceTrak Operations Manual · revision 2026-08-12 (`1303eb4`). Sections marked as generated are produced from the source of record and verified by `npm run manual:check`.