arpedon

User Manual ↘ Condition-based monitoring software

Mechbase Manual

English

NOVEMBER 2026

Contents

0 · Welcome to Mechbase 6
What Mechbase is 6
The Mechbase family 7
Where your data comes from 7
How a site is organised 7
What the colours mean 8
Who sees what 8
Good to know 8
How to read this manual 8

1 · Getting started 10
Signing in 10
Choosing which installation to open 11
A tour of the home page 12
Getting around: menu, shortcuts, and search 13
Switching between installations 15
Your name, email, and units 15
Choosing your email notifications 16
Signing out 17

2 · Finding your way around 18
Opening a section 18
Zones and the equipment inside them 19
What the colours mean 20
Needs attention: where to start each day 20
Recent activity: your audit trail 21
Every point on one page 21
Running hours 22
Dashboards: a live board of what you watch 23

3 · Working with assets 26
What's on an asset page 26
Notes, activity, and open issues 28
Measurement points and what each one watches 29
A point's own page 30
Running state and operating hours 30
Adding a section, a zone, or an asset 31
Editing an asset 31
Adding and editing a measurement point 32
How a point gets its data: sources 34

4 · Measurements, trends, and alarms 36
Reading a trend 36
Filters and time range 37
The readings table 37
Inspecting a vibration reading 37
A single reading's page 38

Comparing readings ... 39
Recording a finding ... 39
Bearings and defect frequencies ... 40
Setting alarm limits ... 41
Keeping false alarms down ... 43
How a reading becomes a state ... 43

5 · Inspections and routes ... 44
What a route is ... 44
Seeing your routes ... 44
Building a route ... 46
Running a route ... 47
Surveys: routes you build as you walk ... 49
Logging a non-conformity ... 49
The inspection report ... 49
History and amending a run ... 51

6 · Staying on top of what needs attention ... 53
The Needs attention list ... 53
Silent assets ... 54
Working through them ... 54
How an item clears ... 54
Where it shows up ... 54
Getting alerts by email ... 55

7 · Mechbase on your phone (iOS & Android) ... 56
What the apps are ... 56
Installing and signing in ... 56
The home hub ... 57
Assets and points ... 58
Walking a route ... 59
Capturing readings with sensors ... 61
Finding things by QR ... 63
Working offline ... 64
Your vibration sensor ... 64
iPhone and Android ... 64

8 · MaintNode devices and automatic data ... 66
What MaintNode is ... 66
The MaintNode Devices page ... 66
Claiming a device ... 67
Getting it on the network ... 68
Connecting a channel to a point ... 69
From device to trend ... 70
Is my device healthy? ... 70

9 · Setting up and managing your account ... 71
Two kinds of setup ... 71
Building out a site ... 71
Importing zones and photos ... 72
Managing your installations ... 73
People and access ... 73
Signing in with Microsoft (single sign-on) ... 76
Devices, alarm defaults, and other settings ... 77

10 · Cookbooks ... 78

For your team ... 78
For developers and integrators ... 78

11 · Running an inspection route ... 79
The short version ... 79
Get the app and open your route ... 79
Start or join the run ... 80
Walk the stops ... 81
Complete each kind of check ... 82
Capture photos, notes, and sensor readings ... 82
When a check fails ... 82
Finish the run and sync ... 83
Open the inspection report ... 84
Also on the web ... 84
Quick reference ... 86

12 · Capturing field readings ... 87
Before you start ... 87
Open the point in front of you ... 87
Take a reading by hand ... 88
Capture a thermal image ... 89
Capture a vibration reading ... 90
Add a photo and notes ... 91
Work offline, sync later ... 91
Where the reading lands ... 91

13 · Triaging Needs attention ... 92
The one-minute version ... 92
Set up your alerts once ... 92
Open Needs attention ... 92
Work it top-down, most urgent first ... 93
Open an item to see why it's flagged ... 93
Act, then clear or resolve ... 94
Don't forget the silent ones ... 95
Where the same items show up ... 95

14 · Building a route and setting alarm limits ... 96
The short version ... 96
Build the route ... 96
Set the alarm limits ... 99
Keeping false alarms down ... 100
Quick reference ... 101

15 · Analytics & BI ... 102
The one-paragraph version ... 102
Before you start: two things you need ... 102
Live feed vs bulk export: which do I use? ... 103
Connect recipes ... 103
Prefer a client library? The Mechbase SDK ... 104
Cursor pagination ... 105
Incremental sync (only what changed) ... 105
Filtering ... 106
Bulk export (a whole dataset as one file) ... 106
Spectra (FFT arrays for one measurement) ... 107
OData details (filtering and sorting) ... 108

Field dictionary 108
Known limitation: hard-deletes don't propagate 109
Quick reference 110

16 · Consuming Mechbase data 111
Choose your path 111
What you'll need 111
The first full load 112
Keeping in sync 112
When rows disappear 113
Scheduling and idempotency 113
Syncing findings into a CMMS 114
Quick reference 115

17 · Pushing data into Mechbase 116
The one-paragraph version 116
Install and construct the client 116
Push one measurement 117
Batch upload, with result handling 117
Sync your registry by external ID 118
Drive a route execution 119
Read, correct, and delete a reading 120
Photos, files, and special point types 120
Go, Swift, and Kotlin 121
Quick reference 122

18 · Streaming from a device over MQTT 123
The one-paragraph version 123
Before you start: what you need 123
Connect to the broker 123
The topic format 124
The payloads 124
Size and sample caps 126
How a channel maps to a point 126
Verifying it arrived 127
Quick reference 127

Appendices 128
A. Glossary 129
B. Keyboard shortcuts 131
C. Health states 131
D. Connecting Mechbase to other systems 132

0 · Welcome to Mechbase

Mechbase keeps track of the health of your plant — every pump, compressor, panel, and motor — so you always know what's running well, what needs a look, and what needs action right now. This chapter explains what Mechbase is and the few ideas the rest of the manual builds on. If you only read two things first, read **What the colours mean** and **How a site is organised** — together they're the whole mental model.

What Mechbase is

Mechbase is condition monitoring and inspections in one place. For every piece of equipment you own, it answers one question continuously: *is this asset healthy — and if not, how bad is it, and since when?*

It does that by bringing together two things that usually live in separate tools:

- **Automatic measurements** from sensors — vibration, temperature, process values, running hours, and full vibration waveforms.
- **Manual inspections** that people carry out on scheduled rounds, with readings, photos, and any problems they find.

Both feed the same picture of your plant and the same health assessment, so what you see on screen is always current — not a spreadsheet someone updates after the fact.

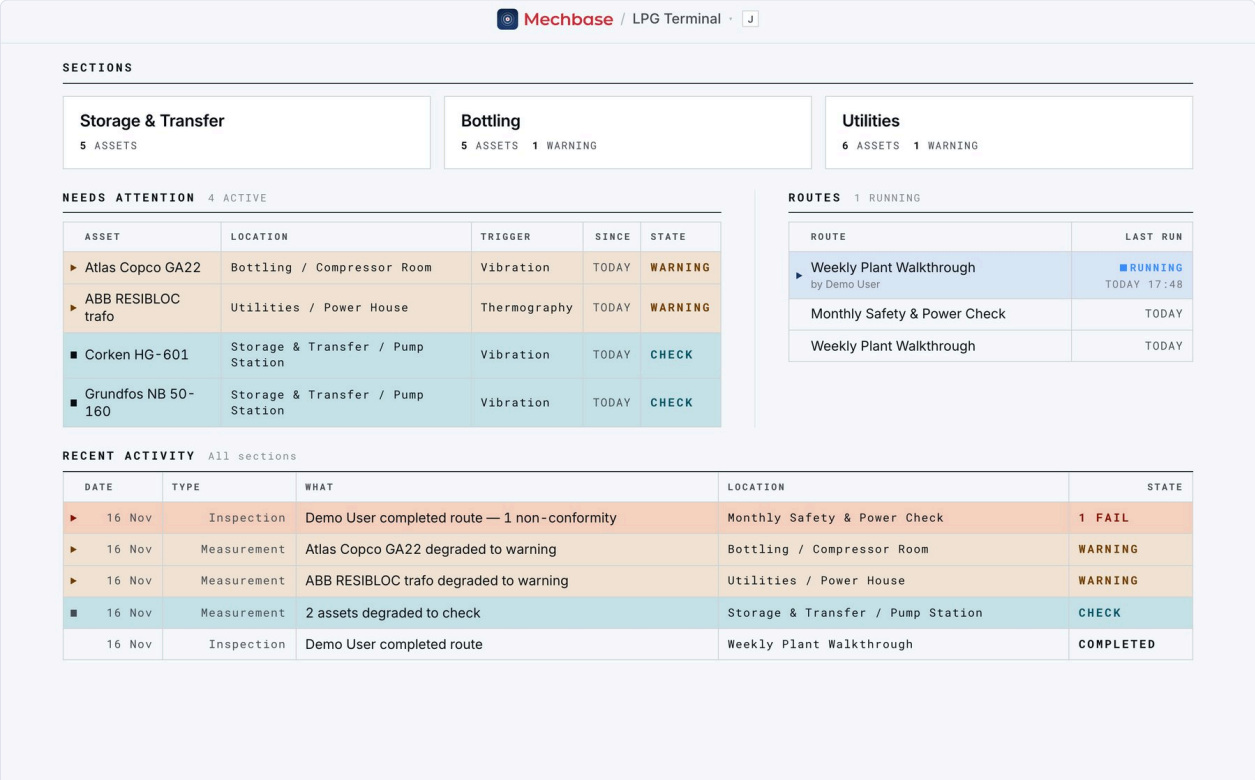


Figure 1 · The Mechbase home page for a site

The payoff is that maintenance stops being guesswork. Instead of running equipment to failure or servicing it on a fixed calendar, you act on evidence: the assets that are actually degrading, in priority order.

The Mechbase family

Mechbase is a small family of connected products, and this manual covers all of them:

- **Mechbase web** — the full system in your browser: the home page, live dashboards, trends and vibration analysis, route building, reports, and setup. It's where most of this manual lives.
- **The iOS app** — a native field app for technicians: walk routes and capture readings — including thermal images and vibration signatures — right at the machine, even offline. *(Mechbase on your phone (iOS & Android))*
- **The Android app** — the same native field app on Android. *(Mechbase on your phone (iOS & Android))*
- **MaintNode** — the edge hardware that measures your equipment automatically and uploads the readings on its own. *(MaintNode devices and automatic data)*

They all share one backend, so a measurement from a MaintNode device, an inspection logged on Android, and a trend you review on the web are all the same data seen from different places.

Where your data comes from

Two streams keep an asset's condition up to date:

- **Sensors, automatically.** MaintNode hardware (or your own devices, over MQTT or the API) measures your equipment and the readings flow into Mechbase on their own. These range from simple values — a temperature, an overall vibration level, whether the machine is running — to **full vibration waveforms** an analyst can open, filter, and inspect.
- **Routes, by hand.** People walk scheduled **inspection routes** and record readings, notes, and **non-conformities** on the mobile apps or the web.

Mechbase treats both the same way: each new piece of data attaches to the right asset, can change that asset's health, and lands in the activity log.

How a site is organised

Nearly every screen in Mechbase is a view onto one simple hierarchy. Learn these five words and the whole product becomes predictable:

```
Installation        a single site – a plant, terminal, factory, or vessel
└── Section         a major area of it            (e.g. Utilities)
    └── Zone        a specific location in it      (e.g. Power House)
        └── Asset   a piece of equipment           (e.g. Caterpillar C9 genset)
            └── Measurement point   what's measured on it, and where
                └── Measurement     one reading, at one moment
```

A few more terms run across that structure:

- **Route** — a recurring inspection round (like *Weekly Plant Walkthrough*). Each time someone performs it you get a **run** (an execution), which may record **non-conformities** — things found out of spec.
- **Needs attention** — the running list of assets that are no longer healthy, or have gone quiet, and need a look.
- **Activity** — the dated log of every state change, inspection, note, and finding.
- **Dashboard** — a live board of the points you choose to watch, for a control room screen or a quick check.

Everything is referred to by **name** — sections, zones, and assets by their names, measurement points by their **location** on the machine (*Motor DE*, *Pump seal*). Locations read as `Section / Zone` wherever space allows.

What the colours mean

Mechbase boils an asset's condition down to one of **four states**, coloured the same way on every screen — and always written out as a word too, so you never have to rely on colour alone:

STATE	MEANING	WHAT TO DO
Nominal	Healthy — within normal limits.	Nothing.
Check	A soft, early signal that something's slightly off.	Keep an eye on it.
Warning	Clearly degraded.	Plan an intervention.
Critical	Severe.	Act now.

You never set these by hand — Mechbase works them out from measurements and inspections, and timestamps every change. That's why it can describe an asset's history in plain words (*"degraded to warning"*, *"returned to nominal"*), and why the **Needs attention** list is a real, current work list rather than someone's opinion.

Two more markers appear alongside the health states. They describe the *data*, not the machine:

- **Silent** — an asset whose points used to report but haven't sent a reading in 30 days, or whose automatic feed has stopped.
- **Stale** — a running-state or device feed that hasn't checked in recently.

Who sees what

Access in Mechbase is simple: each person is a **member** of the installations they work on, and sees those sites and nothing else. The installation picker only lists the sites you belong to.

Inside a site, every member can read and work with everything — trends, routes, notes, setup. A small number of actions are kept for **staff** accounts (the Mechbase administrators at Arpedon and your designated power users): building dashboards, amending other people's inspection runs, and deleting equipment that already has recorded history. Who belongs to which site, and who is staff, is set up by an administrator — that's **Setting up and managing your account**.

Good to know

A few things worth knowing if you're weighing Mechbase up:

- **Full signals, not just numbers.** Mechbase stores complete vibration waveforms, so an analyst can open the time waveform and spectrum of any reading, compare readings side by side, and record a finding against it (Measurements, trends, and alarms).
- **Running hours, automatically.** Mechbase knows when each machine is running — from a tachometer, a run signal, or the vibration itself — and keeps running-hour totals for you.
- **Your history is durable.** A solid SQL data store keeps your measurement history safe and queryable.
- **It talks to other systems.** Mechbase has a REST API, an OData feed for BI tools, an SDK, and a CMMS findings feed, so it can exchange data with your maintenance and analytics tools instead of being a silo. There are pointers in the Appendices and the developer cookbooks.
- **It tells you by email.** Each person chooses which alerts and reports they want emailed (Getting started).

How to read this manual

- This manual covers **Mechbase web** in depth, with the **mobile apps** (Mechbase on your phone) and the **MaintNode device** (MaintNode devices and automatic data) in their own chapters.

- Every web screenshot comes from the Mechbase demo data, on the **LPG Terminal** site. Your own data will look different; the screens won't.
- A few conventions:
  - Screen and panel names are written as they appear.
  - Keys look like `K`; things you click look like **Sign in**.
  - **Note**, **Tip**, and **Heads-up** boxes flag things worth catching.
- There's a **glossary**, a **keyboard-shortcut list**, and a **health-states reference** in the Appendices for quick lookups.

1 · Getting started

This chapter takes you from a browser and a set of credentials to feeling at home in Mechbase: signing in, opening the site you want, reading the home page, moving around quickly, and setting your personal preferences — including which alerts reach your inbox.

Signing in

Mechbase web runs at your organisation's Mechbase address — usually `app.mechbase.io`. Open it in any modern browser. If you're not already signed in, Mechbase takes you to the sign-in screen.

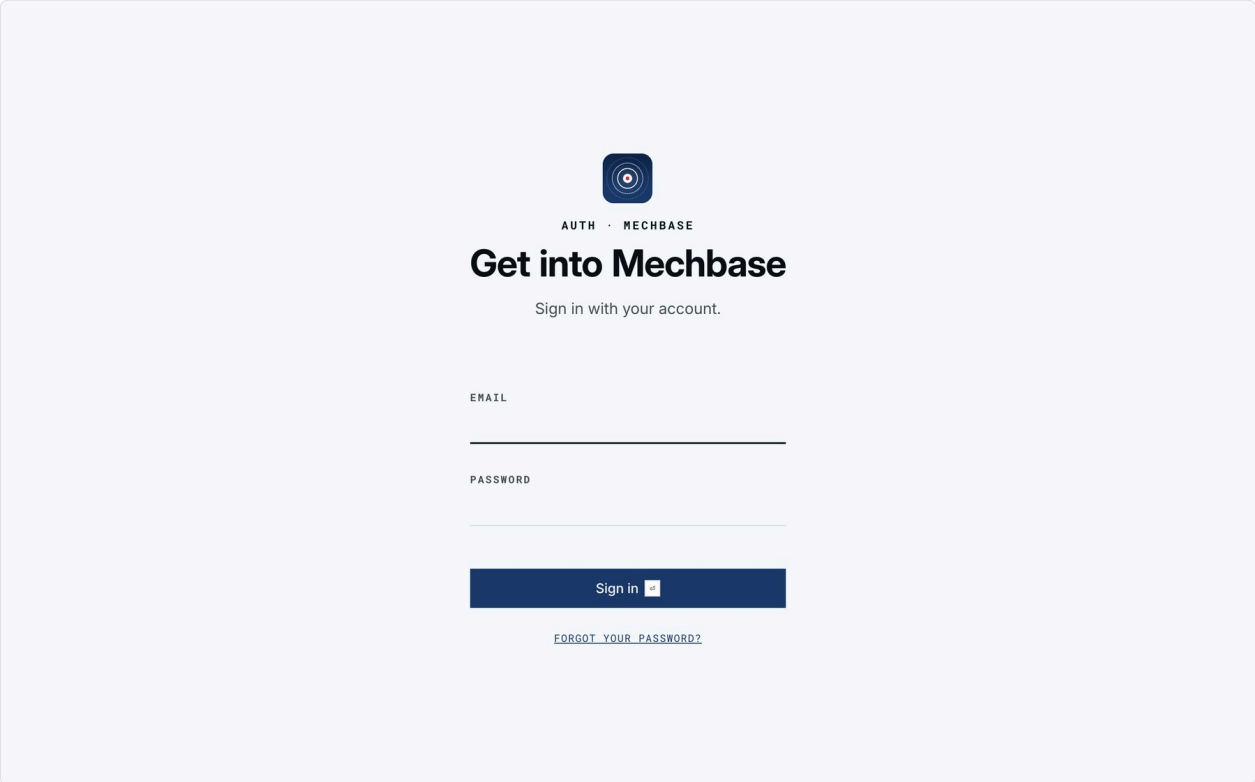


Figure 2 · The Mechbase sign-in screen

To sign in:

1. Type your **Email** — the work address your account was created with.
2. Type your **Password**.
3. Click **Sign in**, or just press **Enter**.

That's it — you go straight to the installation picker below. Mechbase keeps you signed in on that browser until you sign out.

Signing in with Microsoft. If your company uses Microsoft 365 and has single sign-on set up with Mechbase, a **Sign in with Microsoft** button appears under the form as soon as you type an email on your company's domain. Click it and sign in with your usual work account instead of a Mechbase password. You still need a Mechbase account with that email — single sign-on proves who you are, it doesn't create an account or grant access on its own. If you see *"Your Microsoft account isn't set up for Mechbase"*, ask your Mechbase contact to add you.

Forgot your password? Click **Forgot your password?** under the form, enter your email, and click **Send reset link**. Mechbase emails you a link (subject *"Reset your Mechbase password"*); open it, choose a **New password**, confirm it, and click **Set password**. If the link has expired, request a new one from the same page.

> **NOTE**
> Accounts, and the sites each person can open, are managed centrally by an administrator — see Setting up and managing your account. For everyone else the effect is simple: you only ever see the sites you belong to.

Choosing which installation to open

An **installation** is a single physical site Mechbase monitors — a plant, a terminal, a factory, or a vessel. After you sign in, Mechbase shows you every installation your account can open.

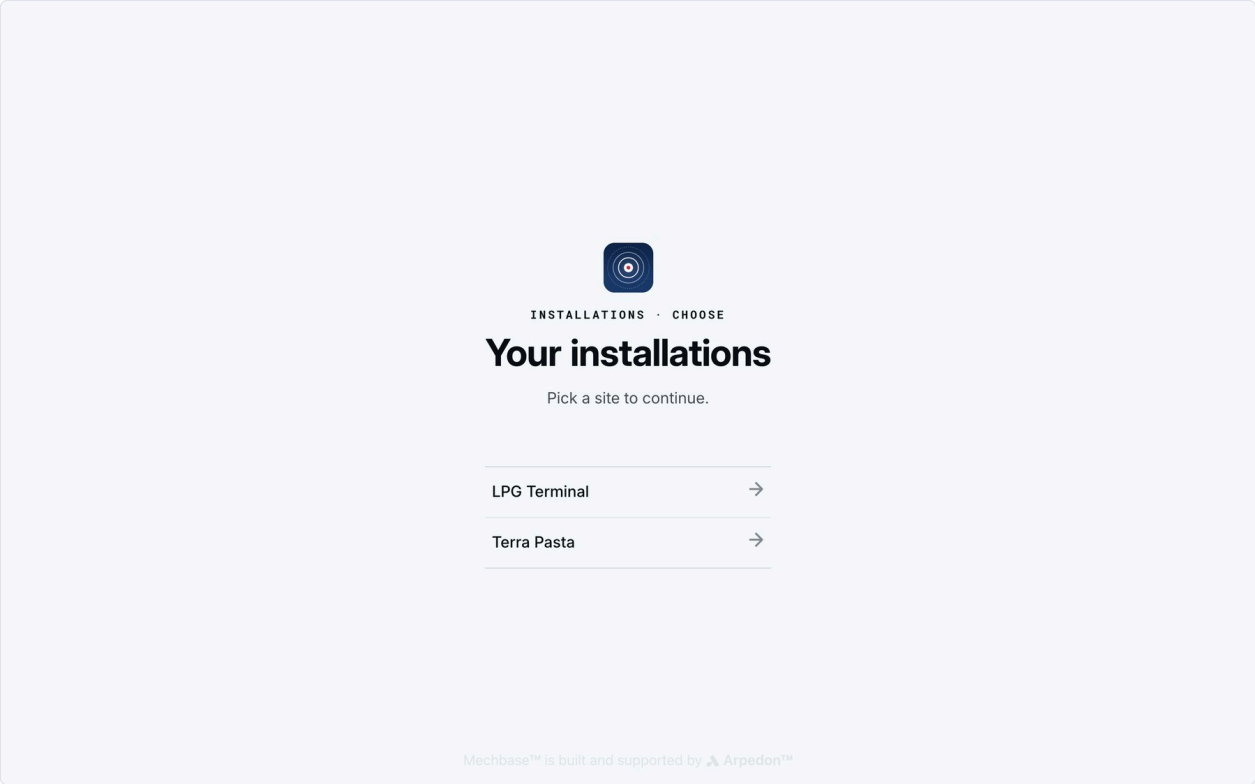


Figure 3 · The installation picker

Click a site's name, or use the arrow keys and **Enter**, to go in. The demo data used in this manual has two: **LPG Terminal** (a storage and transfer terminal) and **Terra Pasta** (a food-processing factory). This manual uses **LPG Terminal** throughout.

Once you pick a site, its home page opens at an address like `app.mechbase.io/2012/`, where `2012` is the installation number — bookmark it to jump straight back next time.

> **NOTE**
> If your account can only open one site, Mechbase skips this screen and takes you straight to its home page. If you don't belong to any site yet, the picker says so — ask your administrator to add you.

A tour of the home page

The home page is your starting point for a site. It's built around one question — *right now, what needs my attention, and where do I go next?* — and it's the same screen whether you're a technician starting a shift or a director checking in.

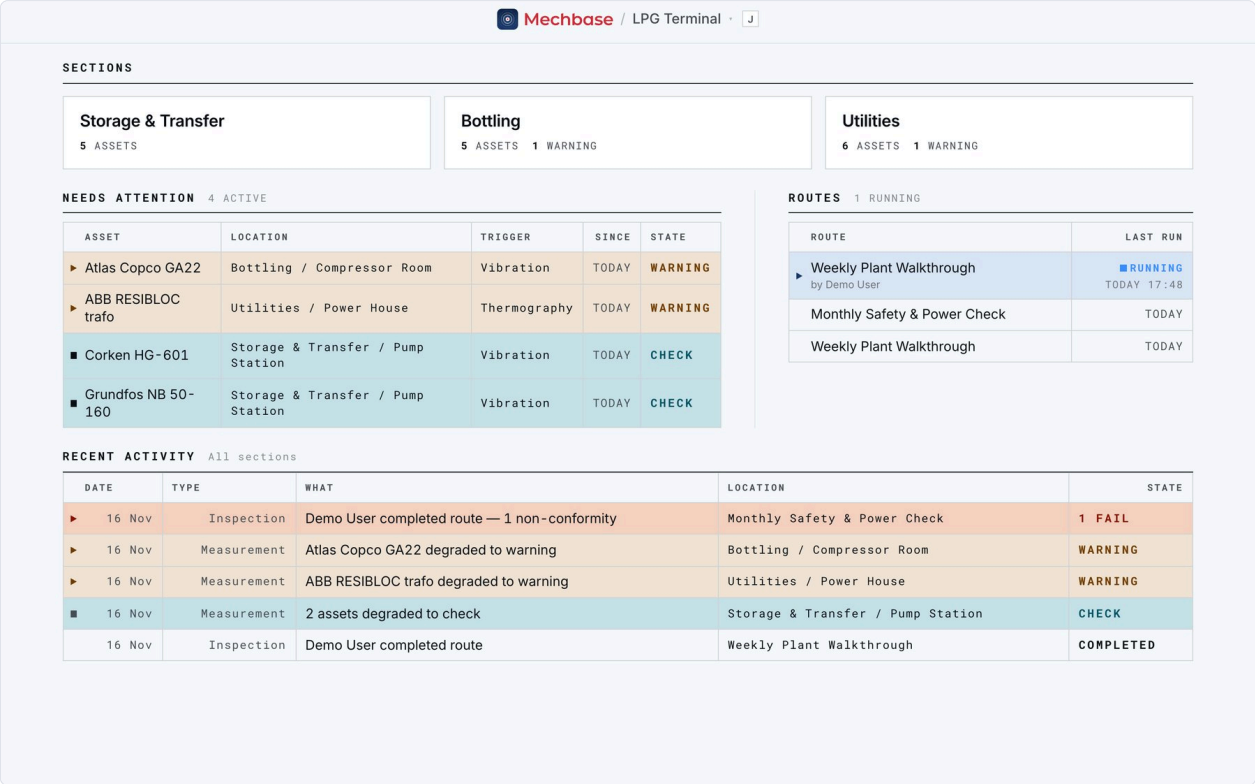


Figure 4 · The home page: sections, Needs attention, routes, and recent activity

The top bar

The bar across the top stays with you on every page. It's a single button: the Mechbase logo, the name of the site you're in (`/ LPG Terminal ▾`), and a `J` key cap. Click anywhere on it — or press `J` — to open the menu (see Getting around).

On any page below the home page you'll also see:

- **Back** on the left, with the place it goes back to (*Back · Compressor Room*) and an `ESC` key cap — press `Esc` to use it.
- A **breadcrumb** on the right showing where you are (`LPG Terminal / Bottling / Compressor Room / Atlas Copco GA22`). Each part is a link, so the site name always takes you home.

Sections

The page opens with one card per **section** — the major areas of the site. Each card shows the section's name and how many assets it holds, plus how many are in **warning** or **critical** (counts of zero are left out). Click a card to open the section (Finding your way around).

Needs attention

Needs attention is your triage list — every asset that isn't healthy, most urgent first. The heading tells you how many are **active**, and how many have gone **silent** (stopped reporting).

COLUMN	WHAT IT TELLS YOU
Asset	The equipment name — click the row to open the asset.
Location	Where it is, as `Section / Zone`.
Trigger	What raised it — the measurement point that's out of limits (*Motor DE*), the inspection check that failed (*Oil level and quality*), or *No reading: …* for a silent asset.
Since	How long it's been in this state.
State	Its health — Check, Warning, Critical — or **Silent**.

The panel shows the top six; if more need attention, **+N more** opens the full list (Staying on top of what needs attention). When everything is healthy it simply says *"No active alerts — All equipment is operating normally."*

Routes

Beside it, **Routes** lists inspection rounds that are in progress or recently done. Walks happening right now come first, with a **Running** marker, who's walking it, and when they started — click one to join or resume it. Below them, each route's latest completed run, linking to its report (Inspections and routes). The panel only appears once a route has been run.

Recent activity

At the bottom, **Recent activity** is the site's running log, newest first, as a table: **Date**, **Type**, **What**, **Location**, and **State**. There are four types of entry:

- **Measurement** — automatic state changes from readings, in plain words: *"Atlas Copco GA22 degraded to warning"*. When several assets change together they're grouped: *"2 assets degraded to check"*.
- **Inspection** — routes people completed: *"Demo User completed route — 1 non-conformity"*.
- **Note** — comments people left on an asset.
- **Finding** — an analyst's recorded diagnosis on a reading (Measurements, trends, and alarms).

Click any row to open what's behind it. For a director, this one feed is a ready-made audit trail of who did what, and how each asset has trended.

> **TIP**
> Times everywhere in Mechbase are shown in your own time zone, and recent dates read naturally — *Today*, *Yesterday*, *3 days ago*, *2 weeks ago*.

Getting around: menu, shortcuts, and search

Mechbase is built to be driven from the keyboard. Rather than hunting through pages, you open the **menu** and jump straight where you need to go.

The menu

Click the top bar — or press `J` (or `⌘J` / `Ctrl+J`) — to open it. You get a filter box up top and every destination below.

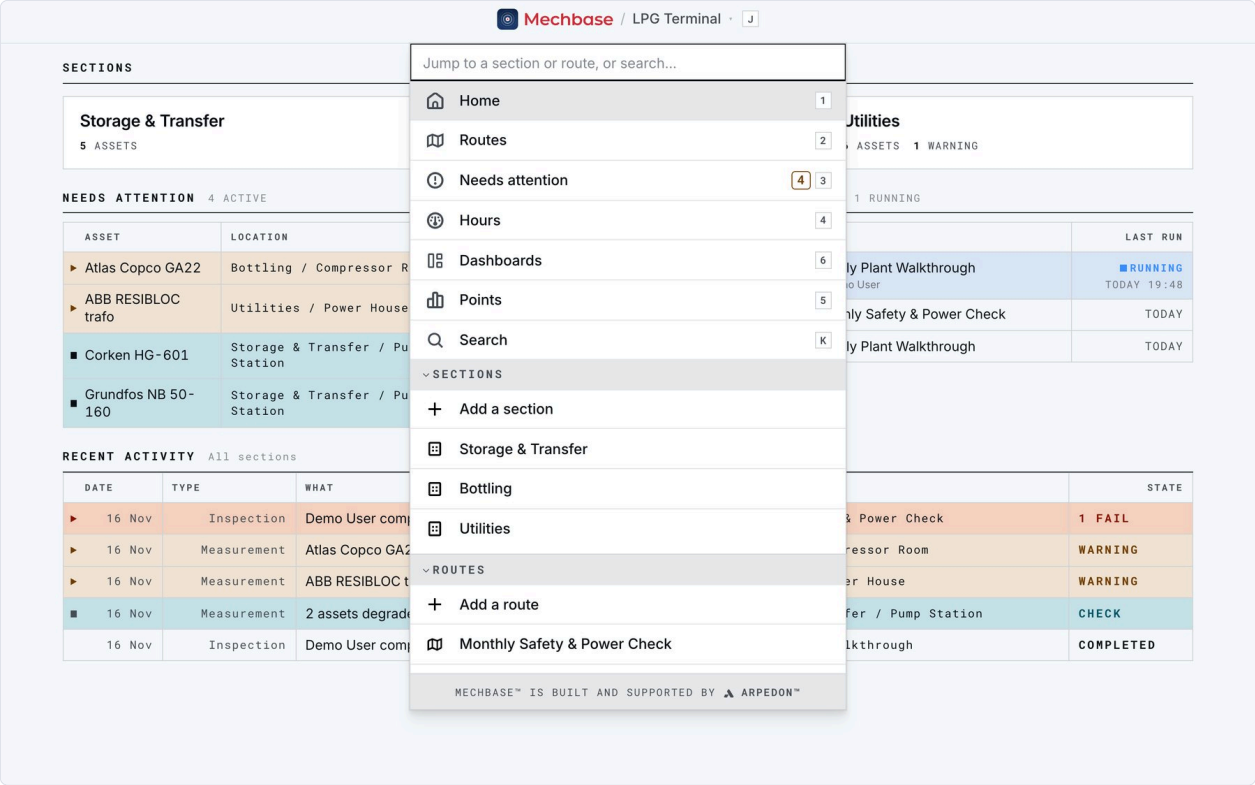


Figure 5 · The menu

From top to bottom:

- A **filter box** — *"Jump to a section or route, or search…"*. Start typing and the list narrows; the last row, **Search for "…"**, runs a full search for what you typed.
- **Home** (`1`), **Routes** (`2`), **Needs attention** (`3` , with a count of active items), **Hours** (`4`), **Points** (`5`), **Dashboards** (`6`), and **Search** (`K`).
- **Sections** — *Add a section*, plus a link to each one.
- **Routes** — *Add a route*, plus a link to each one.
- **Installations** — every site you can open, with the current one ticked (only shown if you have more than one).
- **Setup** — **MaintNode Devices**, **Bearings**, and **Zone types & imports**.
- **Account** — **My Profile** and **Sign out**.

Groups fold away with the small arrow beside their name, and Mechbase remembers which you've folded. Use the arrow keys and **Enter** to move without the mouse.

Keyboard shortcuts

The single-key shortcuts work from anywhere (except while you're typing in a field or a dialog is open):

KEY	ACTION
`J` · `⌘J`	Open or close the menu
`1`	Home
`2`	Routes
`3`	Needs attention
`4`	Hours
`5`	Points
`6`	Dashboards
`K`	Search
`Esc`	Back — up to the page above (or out of the field you're in)

Pages add their own shortcuts, shown as small key caps on their buttons — `E` for Edit, `P` for a new point, `R` to start a route, and so on. The full list is in the Appendices.

Searching the whole site

The filter box in the menu is for going *to* a place you know. When you want to *find* something by name or content, use **Search** (press `K`).

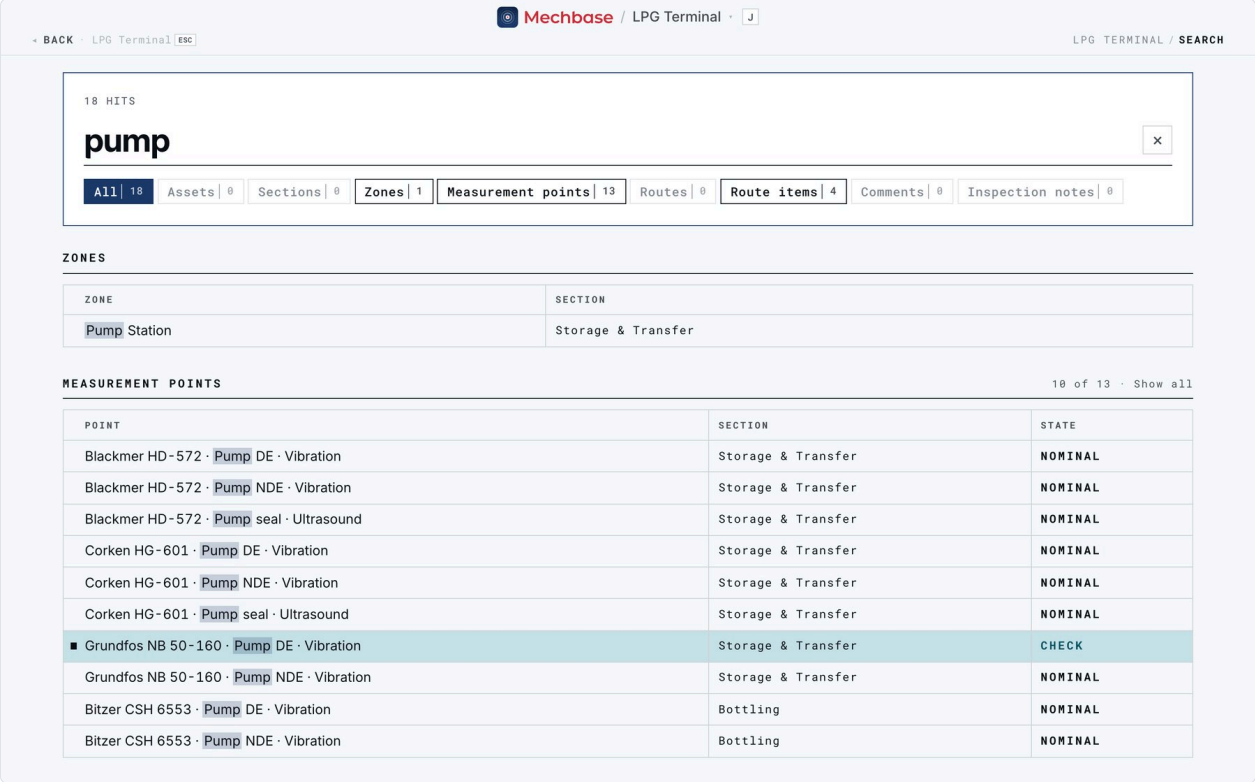


Figure 6 · Search results, filtered by category

Type what you're looking for and press **Enter**. Search covers the whole installation — assets, sections, zones, measurement points, routes, route checks, comments, and inspection notes — and highlights the match. A bar of categories across the top shows how many hits each has; click one (**Zones**, **Measurement points**, …) to see only those. Long groups show the first few with a **Show all** link.

It's the fastest way to reach a specific pump, panel, or note when you only half-remember what it's called. Measurement points are found by their location too, so *"pump seal"* finds every pump seal on the site.

Switching between installations

If your account covers several sites, you can move between them without signing out:

1. Open the menu (`J`).
2. Under **Installations**, click the site you want. The current one has a tick.

You can also go back to the picker at `app.mechbase.io/installation/select/`. Each site keeps its own sections, assets, routes, and dashboards, so switching changes everything below the top bar.

Your name, email, and units

Open **My Profile** from the menu (**Account → My Profile**). This is where you control how Mechbase looks for *you* — it doesn't change anything for anyone else.

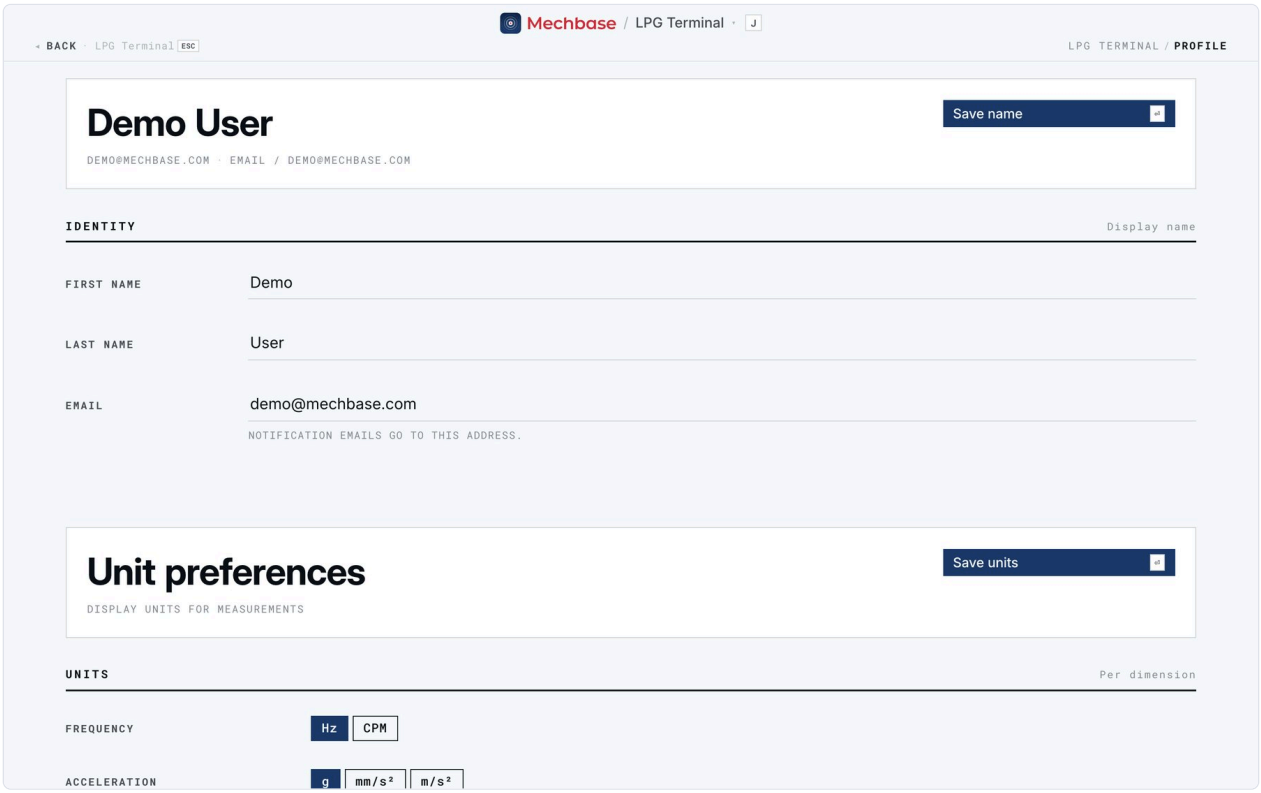


Figure 7 · Your profile: name, email, and units

Your name and email. Your first and last name make up your display name — the one that shows on inspections you complete (*"Demo User completed route"*). Your **Email** is both your sign-in and where notification emails go. Edit any of them and click **Save name**.

Unit preferences. This is the setting engineers and technicians care about most. Mechbase stores every measurement in a standard unit but *shows* it in the unit you choose here, per quantity:

QUANTITY	CHOICES
Frequency	`Hz` · `CPM`
Acceleration	`g` · `mm/s²` · `m/s²`
Velocity	`mm/s` · `in/s`
Temperature	`°C` · `°F`
Pressure	`bar` · `psi` · `kPa`

Pick a unit for each and click **Save units**. From then on, every trend, reading, report, and alarm limit across the app appears in your units — so an analyst who thinks in `CPM` and `in/s` and a process engineer who thinks in `Hz` and `mm/s` can each read the same data in their own terms. Each sheet confirms with *Saved · HH:MM*.

Choosing your email notifications

Further down the same page, **Email notifications** decides which alerts and reports Mechbase emails to you. Every person sets their own, per installation.

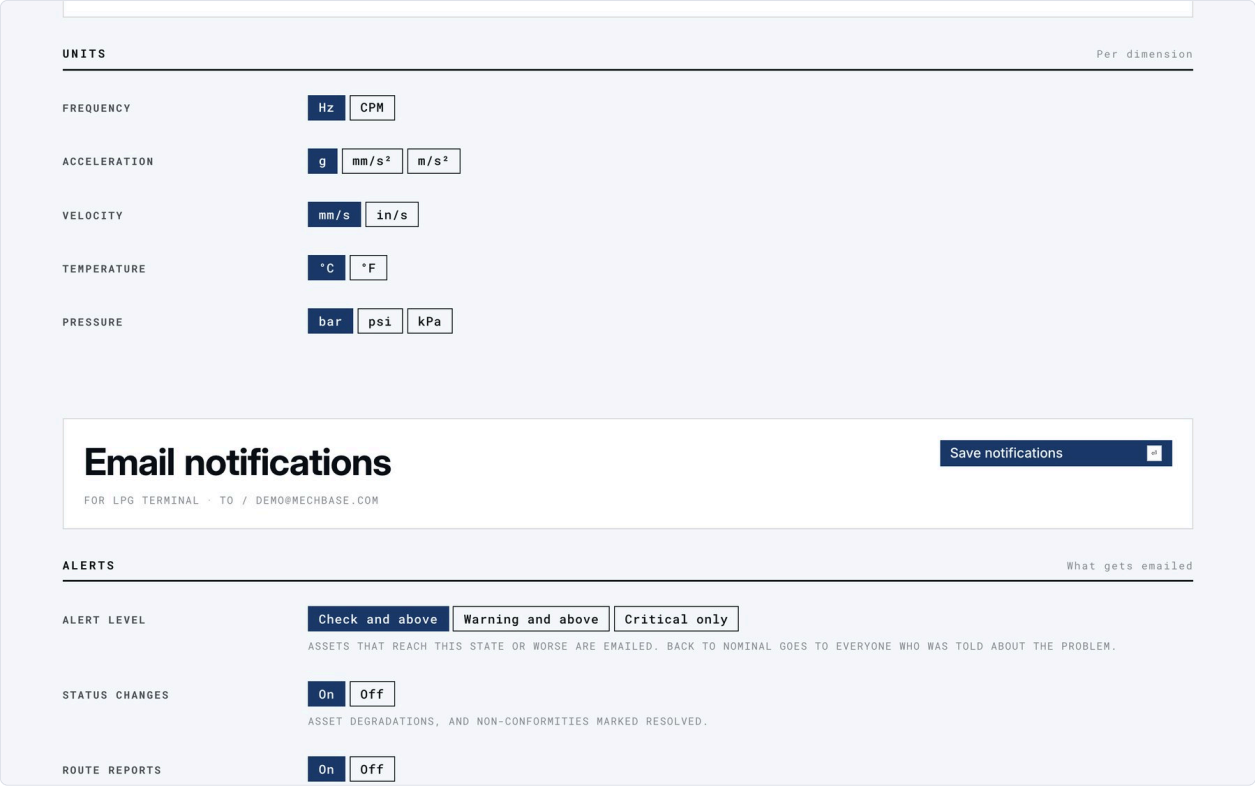


Figure 8 · Email notifications on your profile

- **Alert level** — **Check and above**, **Warning and above**, or **Critical only**. You're emailed when an asset reaches that state or worse. When it goes back to normal, everyone who was told about the problem hears that too.
- **Status changes** — **On** / **Off**: asset degradations, and non-conformities marked resolved.
- **Route reports** — **On** / **Off**: a summary when a route finishes with open non-conformities.
- **Exports** — **On** / **Off**: a download link when an analytics export you requested is ready.
- **All email** — **Active** / **Paused**. Pausing stops every Mechbase notification email to you, across all installations.

Click **Save notifications** to keep your choices. **Send test email** sends a sample to your own address straight away, so you can check it arrives (and isn't caught by a spam filter).

Every notification email has **Manage notifications** and **Unsubscribe** links at the bottom. Unsubscribing pauses all Mechbase email for you; you can switch it back on here at any time.

> **TIP**
> Out of the box everything is on at *Check and above*. A supervisor who only wants to hear about real problems might choose *Warning and above*; a director might choose *Critical only*.

Signing out

To sign out, open the menu (J) and click **Account → Sign out**, or use the **Sign out** button at the bottom of your profile. Mechbase ends your session on that browser and returns you to the sign-in screen.

> **HEADS-UP**
> On a shared workstation, always sign out at the end of a shift — Mechbase otherwise keeps you signed in on that browser.

2 · Finding your way around

Mechbase models every site the same way — **installation → section → zone → asset** — and once that hierarchy and the health colours are clear, everything else (measurements, routes, what needs attention, the field apps) falls into place. This chapter walks that structure, shows how the colours behave, and introduces the site-wide views that cut across it: **Points**, **Hours**, and **Dashboards**.

Opening a section

A **section** is a major area of the site — a process unit, a building, or a functional group like *Utilities*. Open one from its card on the home page, or from **Sections** in the menu.

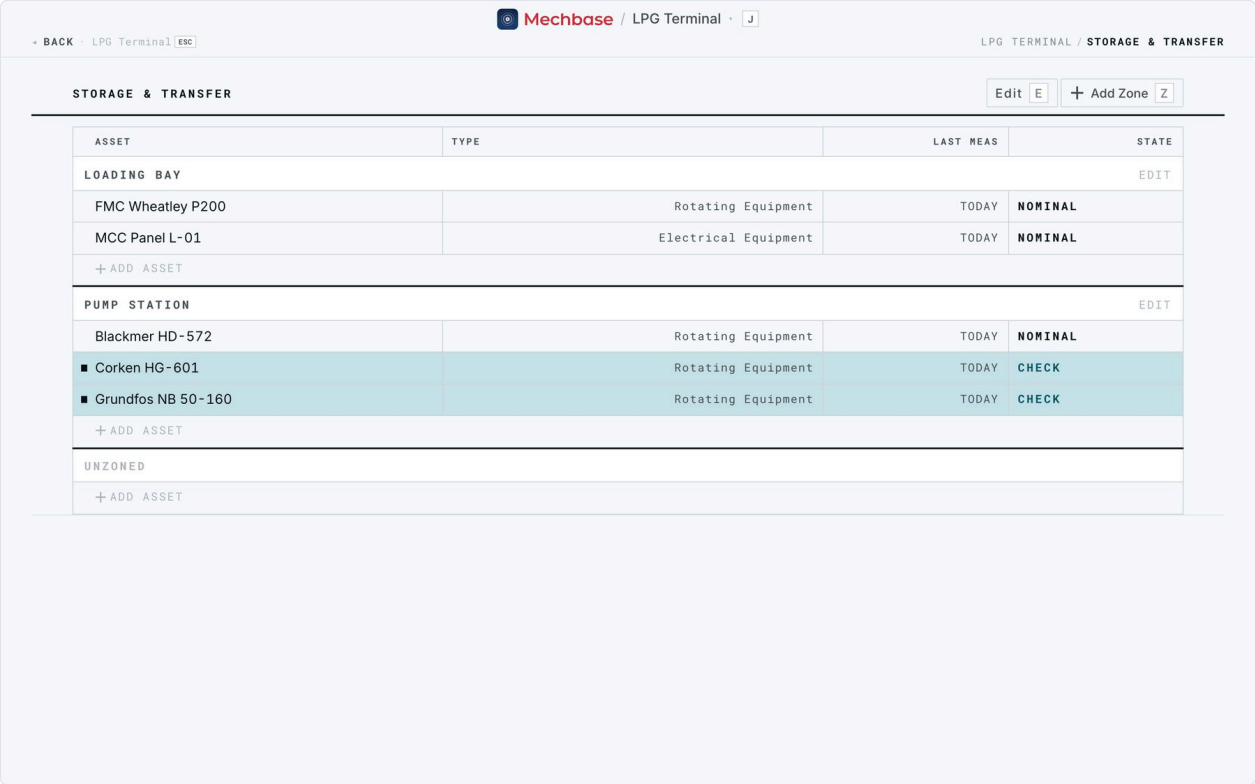


Figure 9 · A section page: zones and their assets

The section page is a single, scannable table of everything in the area, grouped by zone. The header has **Edit** (`E`) to rename or delete the section, and **Add Zone** (`Z`) to create a new zone — give it a name and you're done.

> **NOTE**
> The *Add Zone*, *Add asset*, and *Edit* controls are part of normal navigation, and any member of the site can use them. This chapter is about *reading* the plant; Working with assets and Setting up and managing your account cover *building* it.

Zones and the equipment inside them

Within a section, assets are grouped into **zones** — the specific place the equipment sits, like *Pump Station*, *Loading Bay*, or *Power House*. Zones are why a location anywhere in Mechbase reads as `Section / Zone` (for example *Storage & Transfer / Pump Station*).

On the section page each zone is a labelled block with its own **Edit** button (rename or delete the zone), and each asset is one row:

- **Asset** — the equipment name and model, e.g. *Blackmer HD-572*. Click the row to open the asset page (Working with assets).
- **Type** — its class: *Rotating Equipment*, *Electrical Equipment*, *Static Equipment*, or *Infrastructure*.
- **Last meas** — when it was last measured.
- **State** — its current health.

Every zone ends with an **Add asset** row: type a name in the **New asset** dialog, click **Add asset**, and the new asset appears in place. Assets not yet assigned to a zone sit under **Unzoned**, always at the bottom.

The zone page

Click a zone's name to open its own page — the place for everything about that *location* rather than any one machine.

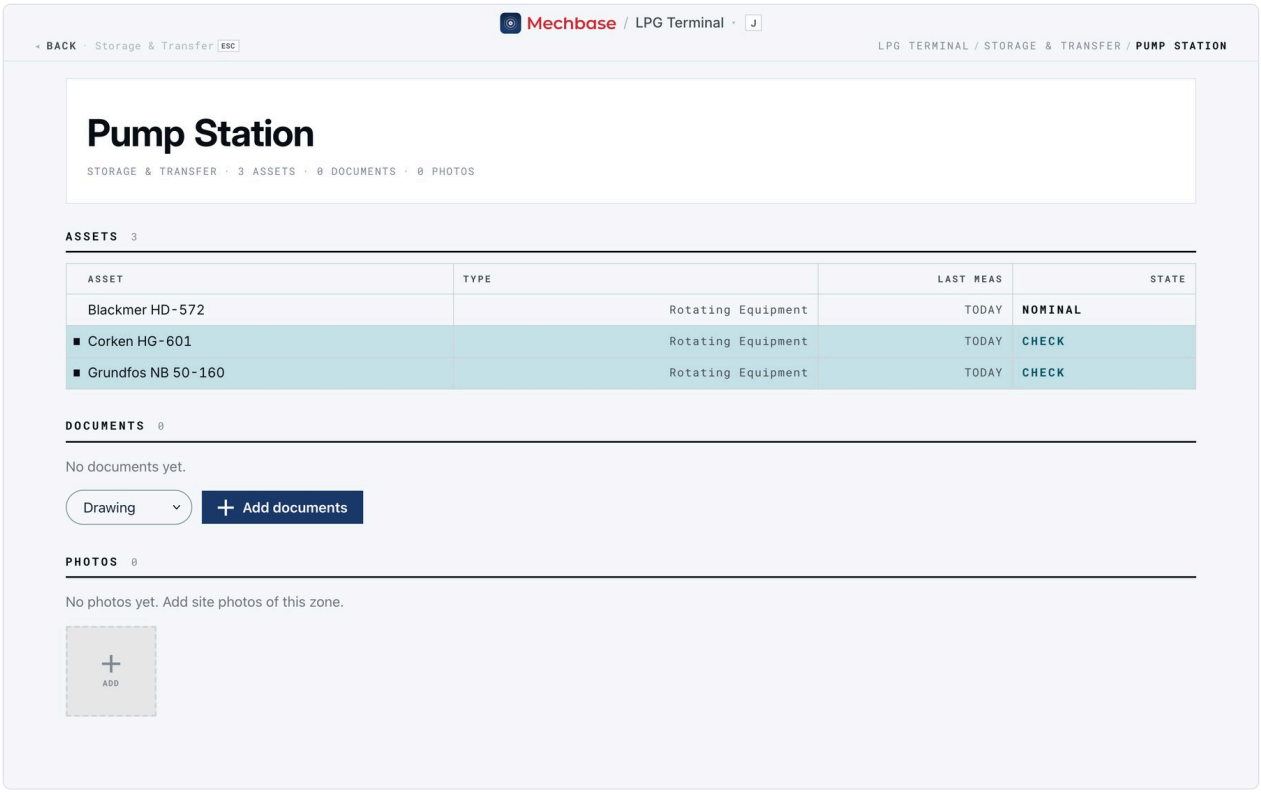


Figure 10 · A zone page: its assets, documents, and photos

- The header shows the section it belongs to and how many assets, documents, and photos it holds. Zones created by a room-list import also show their **type** and any old number they're **also known as** (Setting up and managing your account).
- **Assets** — the same rows as on the section page.
- **Documents** — drawings, datasheets, manuals, check sheets, and test reports for this zone. Pick a kind and click **Add documents** to upload. Documents shared by every zone of the same type appear here too, marked with where they came from.
- **Photos** — site photos of the zone, grouped by the day they were taken. Use the **Add** tile to upload.

TIP

The section page is the quickest “walk the area” view. It lists every asset in a zone with its latest state, so you can plan a physical round zone by zone.

What the colours mean

Every asset is always in exactly one of **four health states**. The same four words and colours appear everywhere — the home page, section pages, asset pages, dashboards, the activity feed, and the field apps.

STATE	COLOUR	WHAT IT MEANS	WHAT TO DO
Nominal	neutral	Within normal limits; healthy.	Nothing.
Check	teal	Slightly off baseline — an early, soft signal.	Keep an eye on it.
Warning	amber	Clearly degraded.	Plan an intervention.
Critical	red	Severe.	Act now.

Mechbase never shows a state by colour alone. In lists, a Check, Warning, or Critical row is tinted *and* carries the state word; Warning and Critical rows also get a small ▸ flag in the margin so they stand out when you scan. State chips put a solid square (■) before the word. A hollow square (□) marks something **Stale** — data that hasn’t arrived recently, not a machine fault.

You never set states by hand. Mechbase derives them from measurements (Measurements, trends, and alarms) and inspections (Inspections and routes), and records every change. That’s what produces the plain-language history in the activity feed — *“degraded to warning”*, *“improved to check”*, *“returned to nominal”*. For a director, it means an asset’s health is a dated, auditable trajectory, not a snapshot opinion.

Needs attention: where to start each day

The home page’s **Needs attention** panel is the most useful view for day-to-day work — every asset that isn’t healthy, most urgent first.

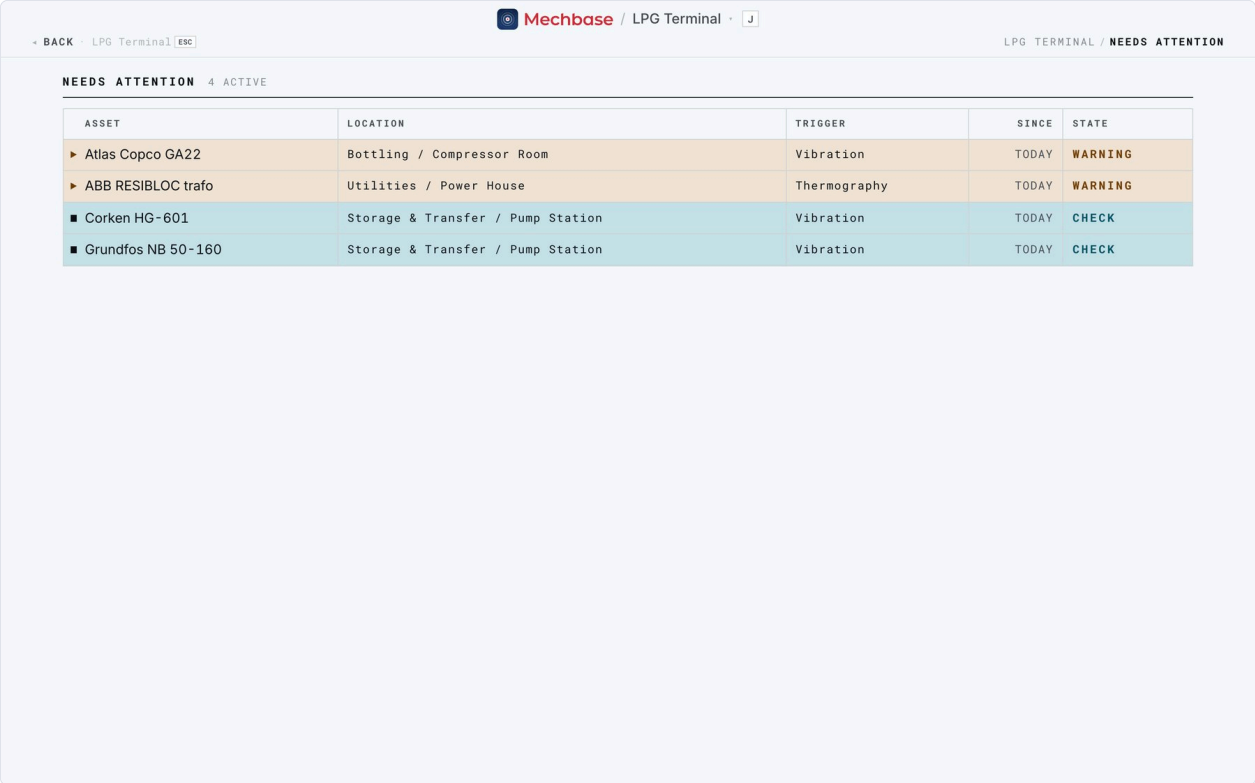


Figure 11 · The Needs attention list

Read it left to right: the **Asset**, its **Location** (`Section / Zone`), the **Trigger** that raised it — the measurement point or the inspection check responsible — how long it's been in that state (**Since**), and the **State** itself. Click any row to go straight to the asset and act.

When more need attention than fit, **+N more** opens the full list — also on the menu as **Needs attention** (`3`). That screen and the triage workflow around it are Staying on top of what needs attention.

Recent activity: your audit trail

Where *Needs attention* answers "what's wrong now?", **Recent activity** answers "what's happened, and who did what?". It's the site's log, newest first, at the bottom of the home page.

Four kinds of entries are mixed together:

- **Measurement** — automatic state changes from readings, e.g. *"Atlas Copco GA22 degraded to warning"*, or *"2 assets degraded to check"* when several change together.
- **Inspection** — completed routes, e.g. *"Demo User completed route — 1 non-conformity"*, linking to the run report.
- **Note** — comments people left on an asset.
- **Finding** — an analyst's diagnosis recorded on a reading.

Because every state change, inspection, and note lands here with a date and a name, the feed is a ready-made audit trail. A director can answer "was this asset inspected last month, and how has it trended since?" without leaving the home page, then click straight through to the evidence.

Every point on one page

Points (menu, or press `5`) lists every measurement point on the site in one table — the fastest way to scan all your readings at once, or to compare like with like across machines.

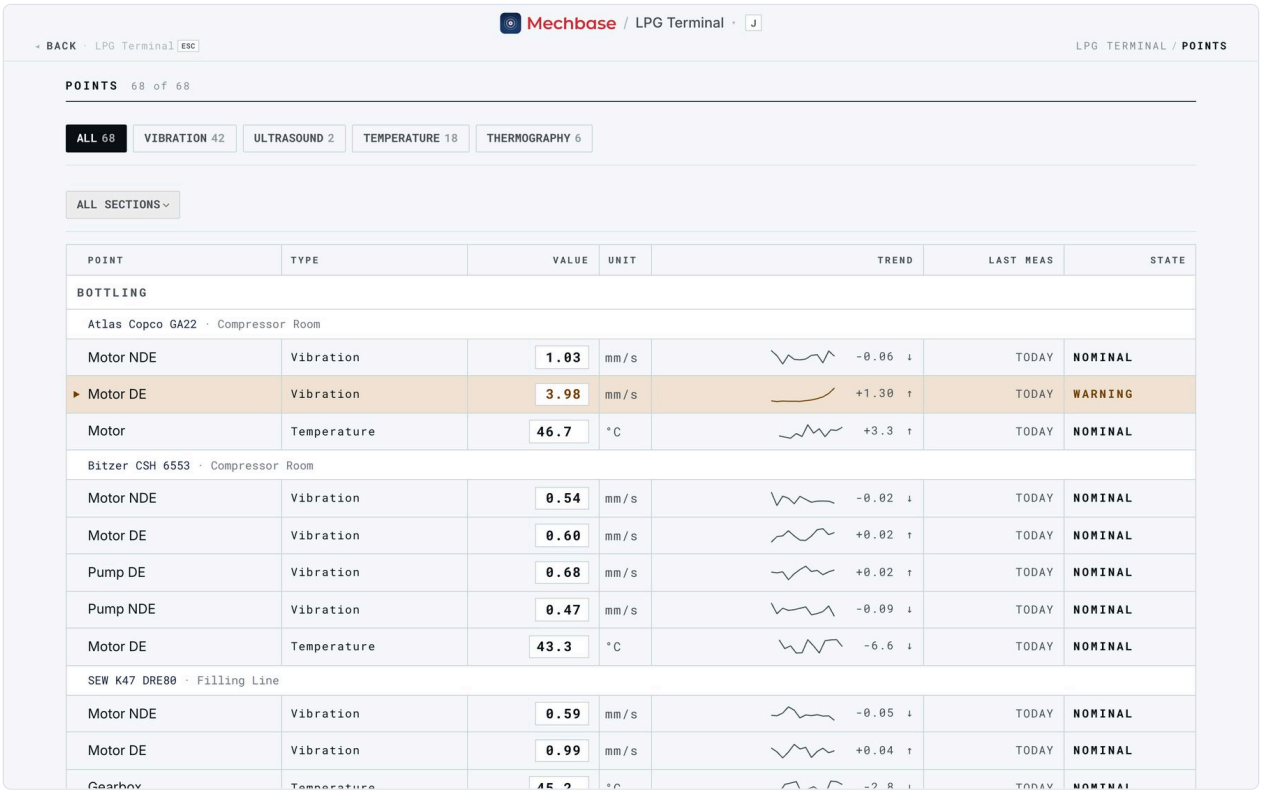


Figure 12 · The Points page

Points are grouped by section, then by asset (with its zone). For each you see the point's **location**, its **Type**, the latest **Value** and **Unit** (in your units), a small **Trend** line with the recent change (↑ / ↓), when it was **Last meas**, and its **State**.

Two filters sit above the table:

- **Type pills** — **All**, or just *Vibration*, *Temperature*, *Ultrasound*, *Thermography*, and so on, each with a count.
- **Section picker** — **All sections**, or one section only.

Click any point to open it. Running-hours (production) points aren't listed here — they have their own page, next.

Running hours

Mechbase keeps track of when each machine is running, and **Hours** (menu, or press `4`) totals it up for every asset that tracks running time.

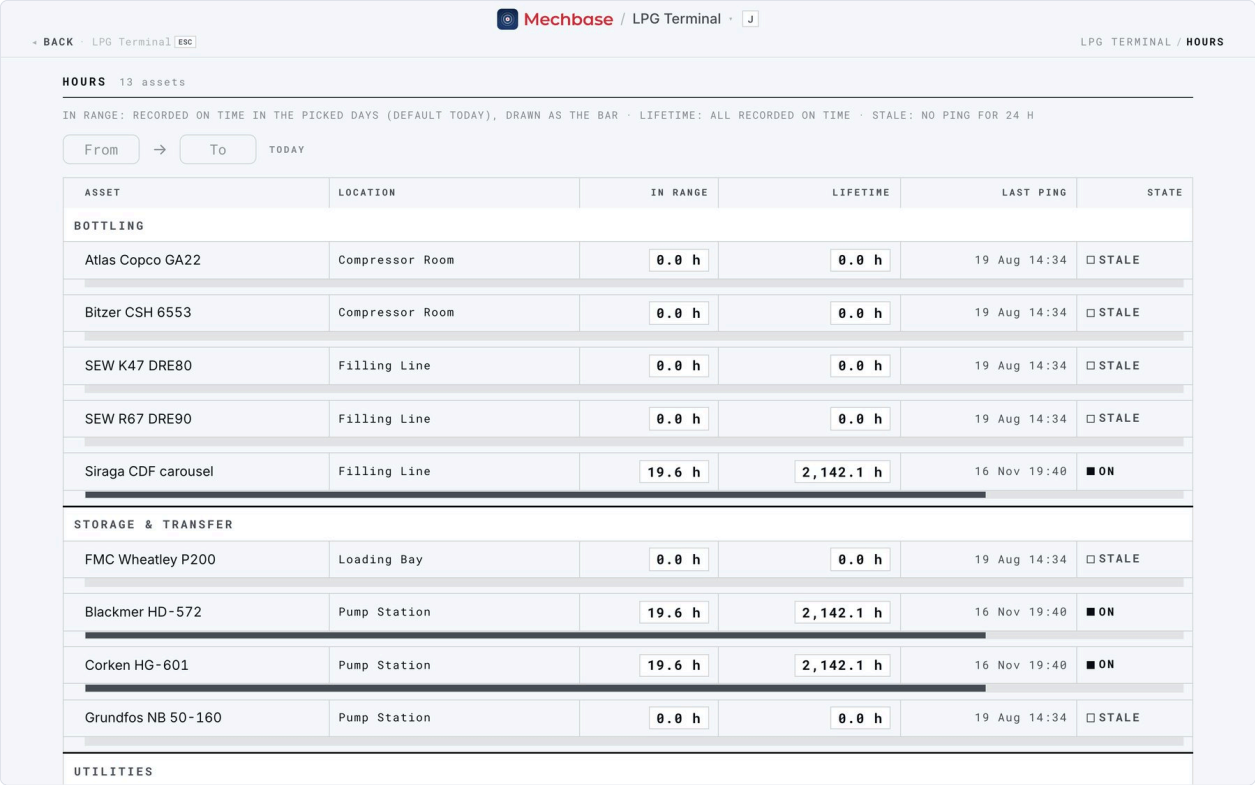


Figure 13 · The Hours page

Assets are grouped by section. For each you see:

- **In range** — running time recorded in the days you pick (today by default), also drawn as the bar under the row.
- **Lifetime** — all running time ever recorded.
- **Last ping** — when the running signal last reported.
- **State** — **On**, **Off**, or **Stale** (no signal for 24 hours). These aren't alarms, so they're shown in ink rather than colour.

Use the **From → To** pickers to total a week, a month, or a maintenance interval. How Mechbase knows a machine is running — a tachometer, a run signal, or the vibration itself — is explained in Working with assets.

Dashboards: a live board of what you watch

A **dashboard** is a live board of the measurement points you choose, grouped into sections — built for a control-room screen, a supervisor's second monitor, or a quick glance at a group of machines. Open it from **Dashboards** in the menu, or press `6`.

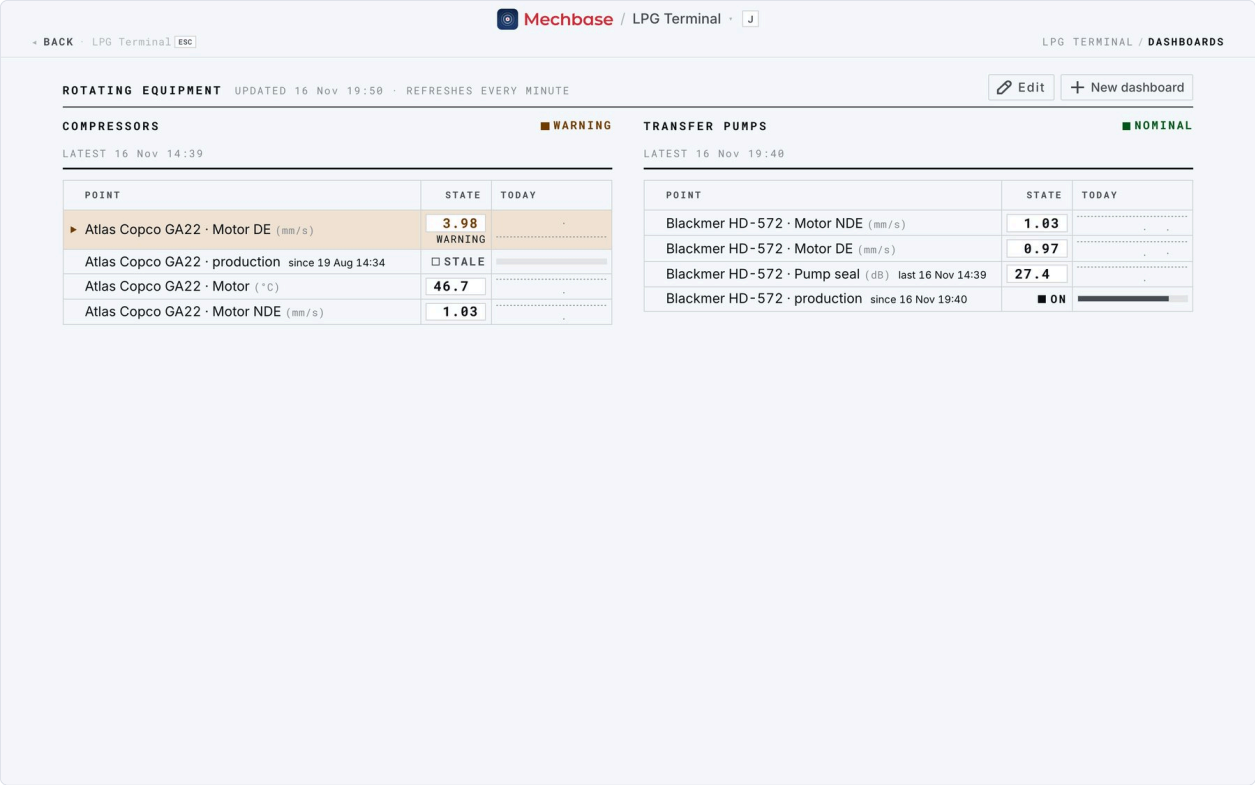


Figure 14 · A dashboard with two sections

Each **section** of the board shows its title, when it last received data, and a status chip (the worst alarm in the section, or **Nominal**). Inside, each point is one compact row:

- **Point** — the label (asset and location by default) and the unit. *"since …"* tells you when an on/off state started; *"last …"* flags a point that's fallen an hour or more behind the rest of the section.
- **State** — the current value. In a section with an alarm, the alarm word sits beside the value. On/off points show **On** / **Off**; a feed that's gone quiet shows **Stale**.
- **Today** — today's trend across a fixed midnight-to-midnight axis, with the warning limit drawn as a dashed line. On/off points show a strip of their periods instead.

Only rows in Warning or Critical are tinted, so a calm board stays calm and an alarm stands out from across the room. The board refreshes itself every minute (**Updated … · Refreshes every minute**), and on a wide screen the sections flow into columns so a whole area fits on one screen. On a phone, a list of *Sections in alarm* appears at the top.

If a site has several boards, the board name becomes a switcher; the **default** board is the one that opens from the menu. Before anyone has built a board, Mechbase shows an automatic one — **All monitored points** — with a section per asset that has live MaintNode sensors.

Building a board

Building and editing boards is a **staff** task. On the board, click **New dashboard** (or **Edit** on an existing one).

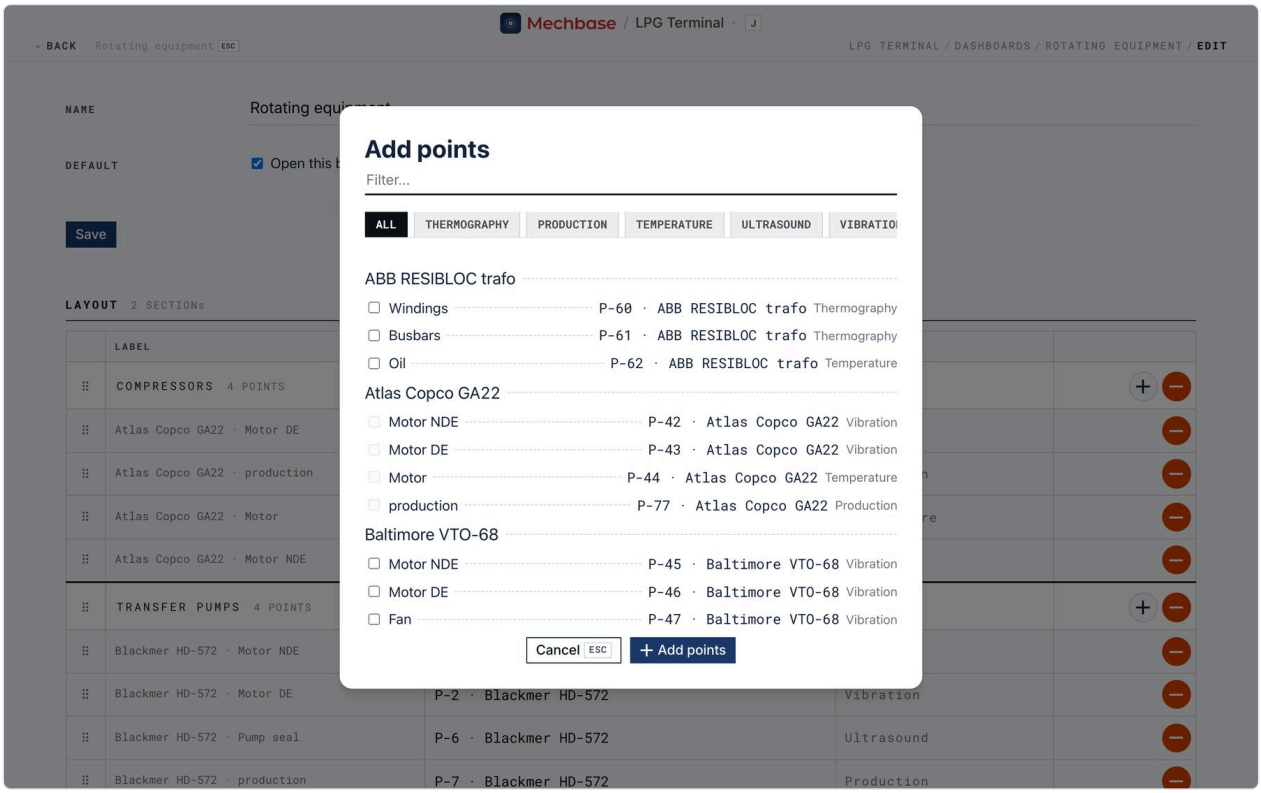


Figure 15 · Editing a dashboard: sections, points, and the Add points dialog

1. Give the board a **Name**, and tick **Default** if it should open from the Dashboards menu. Click **Save**.
2. Type a **Section title** and click **Add section**.
3. Click **+** on a section to open **Add points**. Filter by name or type, tick the points you want, and click **Add points**.
4. Drag the handles to reorder sections and points — or move a point to another section. Every drop saves on its own.
5. Rename a section or give a point a shorter **Label** by typing over it.

Use **–** to remove a point or a section, and **Delete board…** to remove the whole board. Removing things from a board never touches the points or their readings.

> **TIP**
> Build one board per audience — a control room wants every critical machine on one screen; a pump-station supervisor wants just their area. Short labels (*"GA22 DE"*) read better on a wall screen than the full default.

3 · Working with assets

The asset page brings everything about a machine together in one place — its condition, every sensor on it with a week of trend, whether it's running, its history, and any open issues. An **asset** is a single piece of equipment (a pump, a compressor, a genset, a panel), and this chapter walks through its page in detail, then shows how to read a single measurement point and how to build and adjust the structure underneath.

What's on an asset page

Open an asset by clicking its name anywhere it appears — a section or zone page, *Needs attention*, the Points page, search, or the activity feed. You land on a page like this.

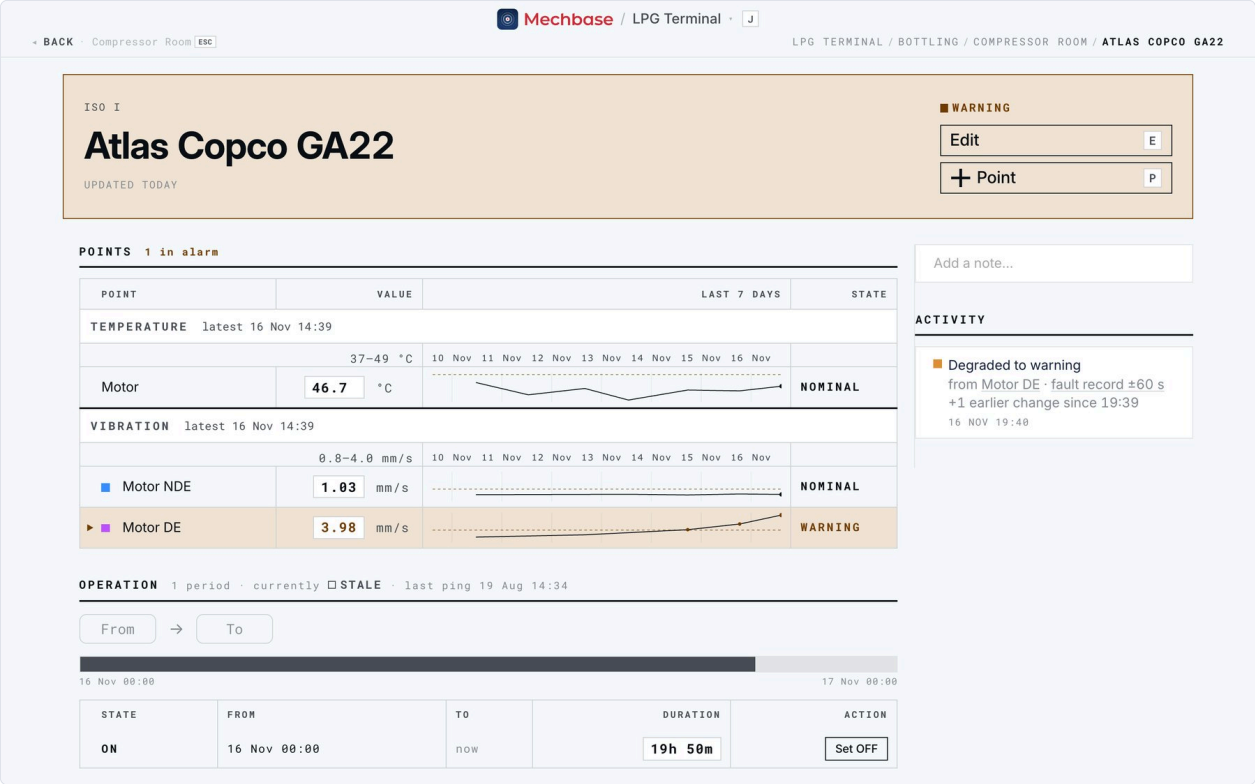


Figure 16 · An asset page: header, points with a week of trend, and activity

The header

The header gives you the asset at a glance:

- Its **ISO 10816 class** (`ISO I`) for rotating equipment — the vibration severity class for this machine — or its equipment type for anything else.
- Its **name**, and a nameplate line (maker, model, rating) if one has been entered.
- Its current **state** (here, **Warning**) and when it was last **updated**.
- **Edit** (`E`) to change the asset, and **Point** (`P`) to add a measurement point.

Where the asset sits — section and zone — is in the breadcrumb at the top right.

Points

The **Points** section lists every sensor on the asset, **one row per point**, grouped by type — here **Temperature** and **Vibration**. The heading counts how many are **in alarm**, and how many are **not reporting**.

Each row shows:

- The point's **location** (*Motor DE*, *Motor NDE*), with a colour swatch that matches its dot on the topology diagram.
- Its latest **Value**, in your units.
- **Last 7 days** — a small trend line. Every row in a group shares one scale and one time axis (printed above the group, with the group's actual low–high), so you can compare sensors directly.
- Its **State**.

Scrub the week. Move the pointer over any trend line: a cursor runs through every row in the group, the time appears above, and each row's Value switches to that sensor's reading at that moment. It's the quickest way to see what every bearing on a machine was doing when one of them jumped.

Only alarm readings, the limits, and the latest point are coloured. Readings taken while the machine was stopped show as faint dots. A point whose live feed has gone quiet shows a **No data** chip and how long it's been silent. Click any row to open the point.

Operation

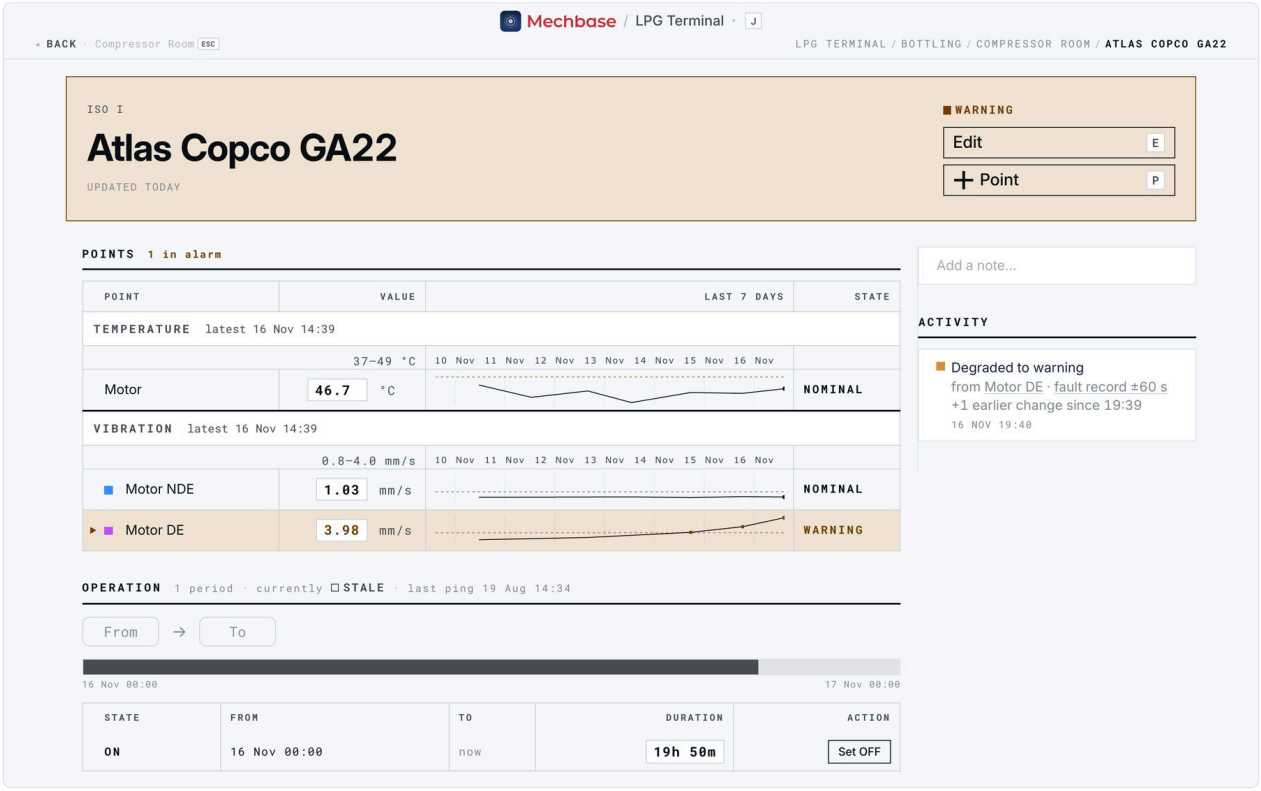


Figure 17 · The Operation panel: running periods for the day

If the asset tracks whether it's running, the **Operation** panel shows it. The heading reads the number of periods, whether it's **currently On**, **Off**, or **Stale**, and when the running signal last pinged. Below it:

- **From → To** — the window to show; today, midnight to midnight, by default.
- A strip of the window, with running and stopped periods drawn differently.
- A table of periods — **State**, **From**, **To**, **Duration** — newest first, with **Set ON** / **Set OFF** on the current period to correct it by hand.

Why it matters: some readings — vibration especially — only make sense while the machine is running. Mechbase knows when the asset was stopped and treats those readings accordingly (see Running state and operating hours).

Topology

For rotating equipment, **Topology** is a simple map of the machine — its **body segments** (non-drive end and drive end of the motor and the driven machine) with a coloured dot for each measurement point. The same colours are used for the points and their trend lines, so you can connect a reading to a physical spot on the machine at a glance.

Attachments

Attachments holds reference **Photos** and **Docs** for the asset — nameplate shots, drawings, manuals. Click **Add photo** or the document tile to upload; click any tile to open it full size.

Notes, activity, and open issues

The right-hand side of the asset page is its conversation and history.

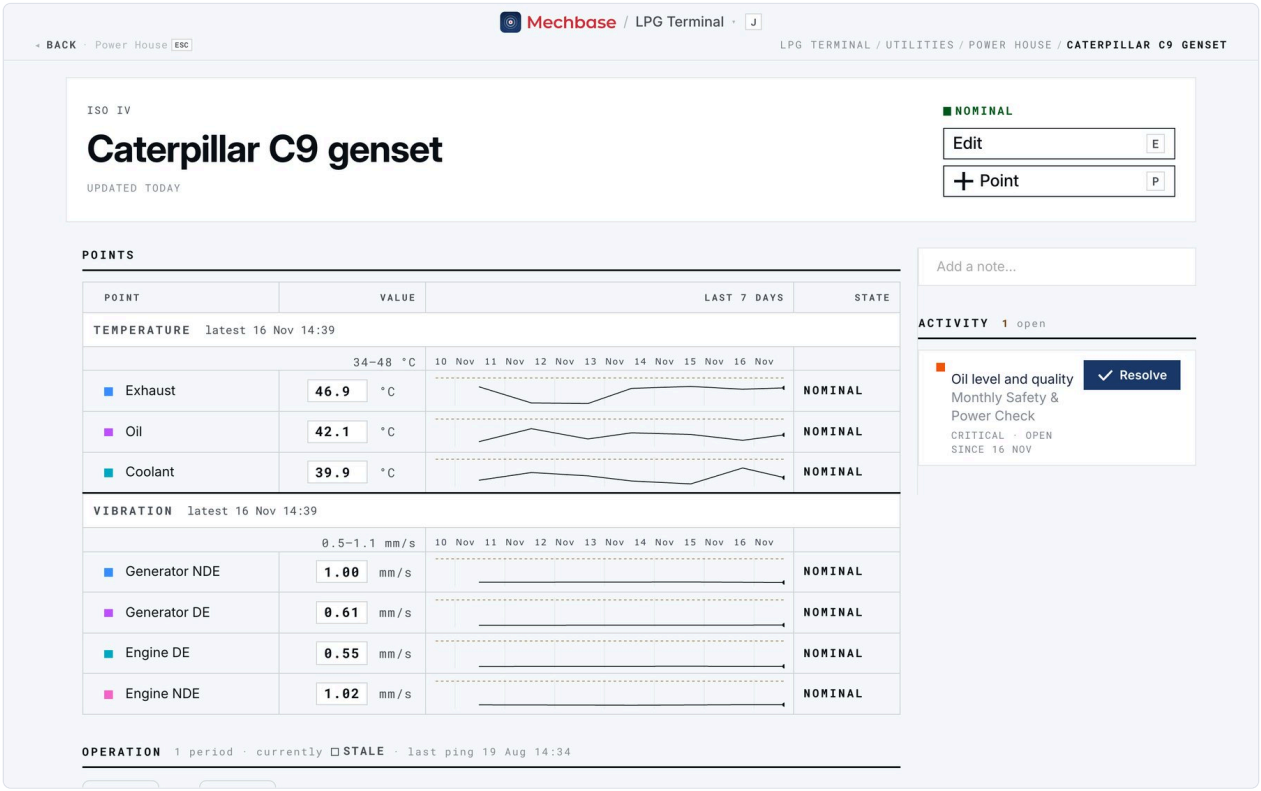


Figure 18 · An asset's activity, with an open non-conformity pinned at the top

Notes. The **Add a note...** field at the top expands when you click it. Write what you saw, optionally open **Reference measurement points** to tag specific points, and click **Post** (or `⌘ Enter`). You can edit or delete your own notes later with the pencil and bin icons; deleting shows a short **Undo** so a slip is never permanent.

Open issues. Any **non-conformities** raised against the asset during inspections are pinned at the top of **Activity**, and the heading counts how many are **open**. Each shows the failed check (linking to the inspection report), the route, its severity — **Warning** or **Critical** — and how long it's been open, with a **Resolve** button. Resolving asks for an optional **Resolution comment**; click **Mark resolved** and the item closes, with your name and comment on the record. Non-conformities come from routes — see Inspections and routes.

Activity. Below the open issues is the asset's complete history, newest first:

- State changes in plain language — *"Degraded to warning from Motor DE"*, with the reason. Changes caused by a reading carry a **fault record ±60 s** link: a page of mini-trends showing every sensor on the asset in the minute before and after the change, so you can see what moved first.

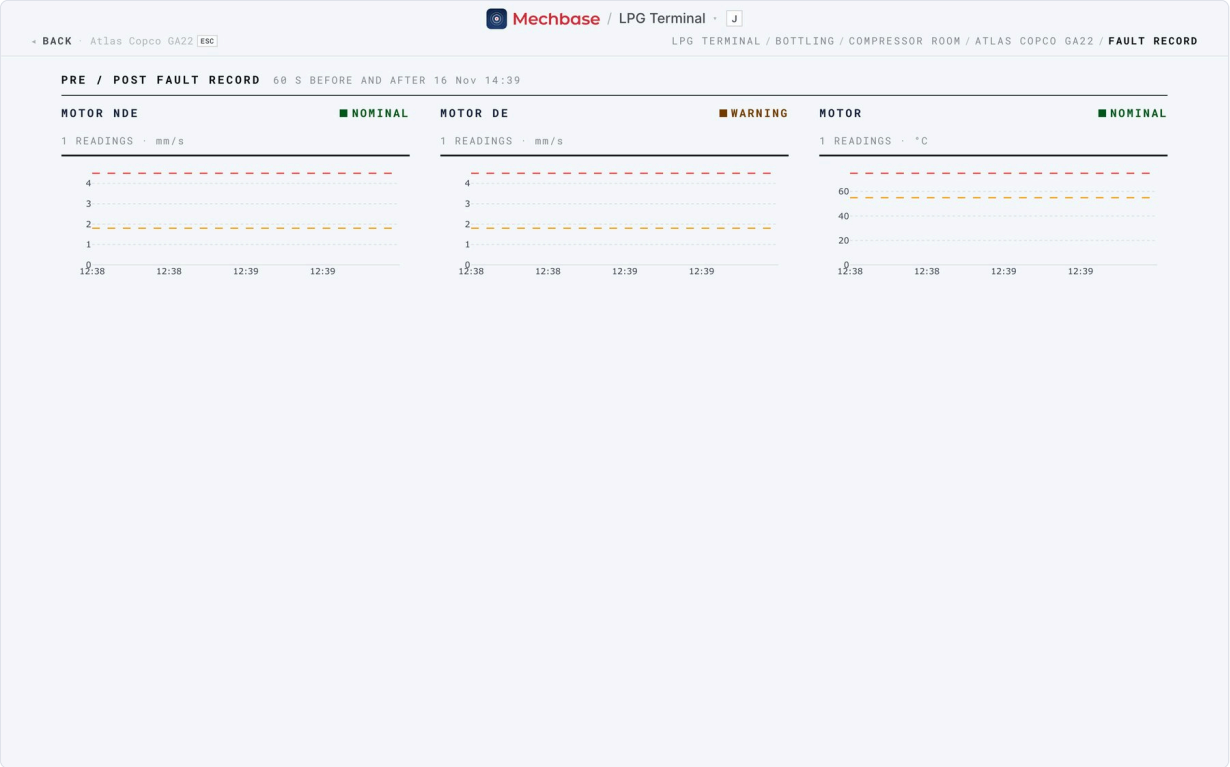


Figure 19 · A fault record: every sensor on the asset, 60 seconds either side of a change

- Changes from inspections — *"via inspection"*, with who and when, linking to the report.
- **No readings** from a point (and how often it was expected), and **Readings resumed** when it comes back.
- Resolved non-conformities, with the resolution comment.

Bursts of changes fold into *"+N earlier changes"* so the list stays readable.

Measurement points and what each one watches

A **measurement point** is one specific thing you measure, in one specific place on the asset — the drive-end bearing's vibration, the motor's temperature, the pump seal's ultrasound. An asset usually has several, and each is named by its **location**.

Each point has a **type** (its transducer), and Mechbase groups points by type on the asset page. Mechbase supports a wide range, so you can capture almost anything:

- **Vibration** — overall levels and full waveforms from rotating equipment.
- **Temperature** and **Thermography** — contact temperature and thermal images.
- **Ultrasound** — high-frequency sound, in dB, e.g. from a failing seal or bearing.
- **Pressure**, **Flow**, **Current**, **Voltage** — process and electrical values.
- **Tachometer** and **Counter** — shaft speed and counts.
- **Boolean** — a simple yes/no reading.
- **State** — an on/off signal such as a fault contact or a run lamp, recorded as periods (*On since 06:12*) rather than a trend.
- **Production** — whether the machine is running; it drives the Operation panel and running hours. It isn't offered when you add a point by hand.

> **NOTE**
> Vibration points only report alarms once the asset has an **ISO 10816 class** set, since that class defines the severity bands. You set it on the asset, and each point can inherit it or override it.

A point's own page

Click a point to open it. This is where you read its numbers and history.

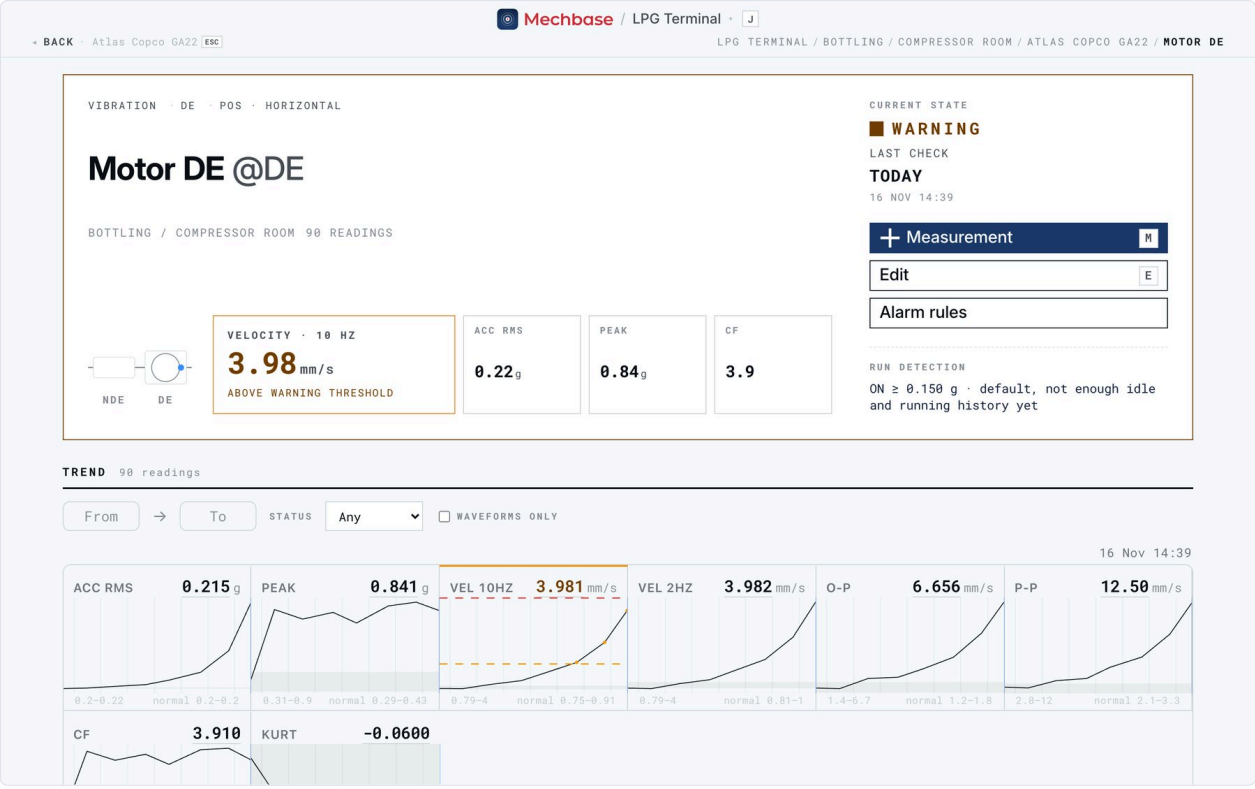


Figure 20 · A vibration point: headline, signal trends, and the inspect panel

At the top you get an honest **headline**: the value that matters for this point (for vibration, the velocity at 10 Hz, with its relation to the limit — *"Above warning threshold"*), the other key values beside it, the **current state**, and when it was **last checked**. If the machine is stopped, the headline says so and shows the state **at last run** instead of a meaningless stopped reading; if the feed has gone quiet, it says **No data** and since when.

Alongside:

- **Measurement** (`M`) — record a reading by hand, right there on the page.
- **Edit** (`E`) — change the point.
- **Alarm rules** — see and change the limits (Measurements, trends, and alarms).
- **Sources** — where data arrives from automatically (*"Compressor room node, CH 0"* or *"MQTT LPG-TM-001, CH 1"*), with a live dot when it's reporting.
- **Run detection** (vibration points) — how this sensor tells running from stopped; see Running state and operating hours.

Below the headline are the point's trends, and below those the list of readings, grouped by day and by run. Reading those properly — and the vibration analysis tools — is the subject of Measurements, trends, and alarms.

Running state and operating hours

Mechbase works out on its own whether each machine is running, because so much depends on it: vibration readings only mean something while the machine runs, stopped readings shouldn't raise alarms, and running hours drive maintenance intervals.

It decides in this order:

1. **A tachometer, if the asset has one reporting.** A shaft-speed reading above zero means running. While the tachometer has reported in the last ten minutes, it has the final say.
2. **A run signal**, such as a contactor or a PLC bit, wired to a MaintNode or sent from your own system.
3. **The vibration itself.** Each vibration sensor learns, from its own history, the level that separates *idle* from *running* on that machine. The point page shows this as **Run detection** — for example *"ON ≥ 0.150 g · default, not enough idle and running history yet"* while it's still learning, and the learned idle and running levels once it has enough. If a sensor only ever fires when a *neighbouring* machine runs, Mechbase says so (*"Not used: … Check where it is mounted"*) — usually a sign it's mounted on the wrong machine.

You can always correct the current period by hand with **Set ON** / **Set OFF** in the Operation panel.

Operating hours. Running time accumulates automatically and is totalled on the **Hours** page (Finding your way around). If a machine has a physical hour meter, record its reading on a route or by hand on the running-hours point: Mechbase takes it as the new total and carries on counting from there.

Adding a section, a zone, or an asset

You build out a site from the top down: add **sections**, add **zones** inside them, then add **assets** inside the zones.

- **Add a section** from the menu, under *Sections → Add a section*.
- **Add a zone** from a section page, using **Add Zone** (`Z`).
- **Add an asset** from the **Add asset** row at the bottom of a zone on the section page: type its name in the small **New asset** dialog and click **Add asset**.

The new asset appears in place, ready to fill in. Open it and click **Edit** to set its type, class, and topology (next section), then add points with **Point**.

> **TIP**
> Large sites don't have to be typed in by hand. A room or equipment list in Excel can be imported as zones in one go — see Setting up and managing your account.

Editing an asset

Open the asset and click **Edit** (`E`).

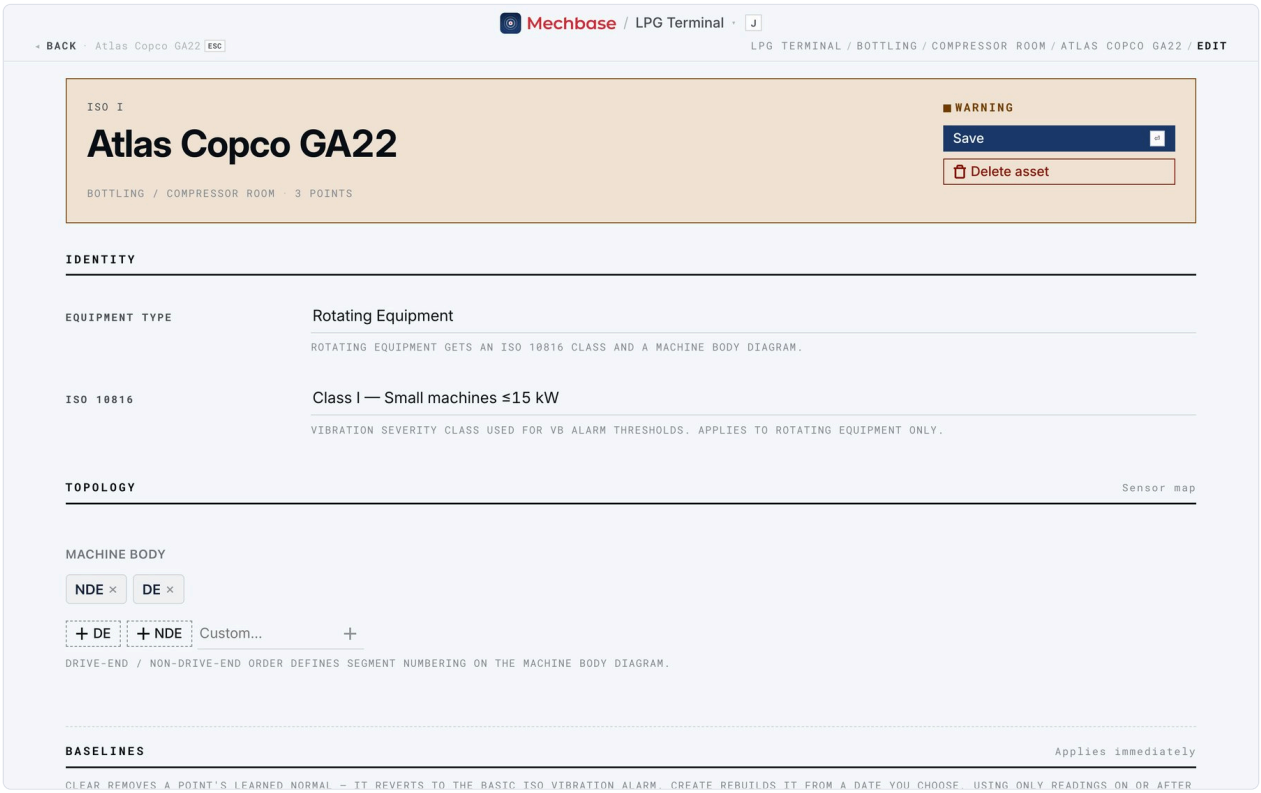


Figure 21 · Editing an asset: identity, topology, and baselines

- **Identity** — the **Equipment type**: *Rotating Equipment*, *Electrical Equipment*, *Static Equipment*, or *Infrastructure*. Only rotating equipment gets an **ISO 10816** class and a machine body.
- **ISO 10816** — the vibration severity class (Class I, small machines, through Class IV) that sets the default vibration alarm bands.
- **Topology** — the **Machine body**: the drive-end and non-drive-end segments that become the bearing diagram and decide where points sit. Add segments with **+ DE**, **+ NDE**, or a **Custom…** name, in order along the shaft.
- **Baselines** — for each vibration point, the learned "normal" that relative alarms compare against. **Clear** reverts a point to the basic ISO alarm; **Create from date…** relearns it from a date you pick — useful after a repair, or to escape a period of bad data. Mechbase tells you how many usable readings there are since that date (it needs 100) before you commit.

Click **Save** to keep your changes.

> **HEADS-UP**
> **Delete asset** removes the asset with all its points and history. An asset that already has recorded readings can only be deleted by a staff user, who must type the asset's name to confirm.

Adding and editing a measurement point

On an asset page, click **Point** (`P`). A small dialog asks for the **Transducer type**, the **Location** (*"e.g. Motor, Bearing 1"*), and — for rotating equipment — where it sits on the machine body. Click **Add point** and it appears in the list.

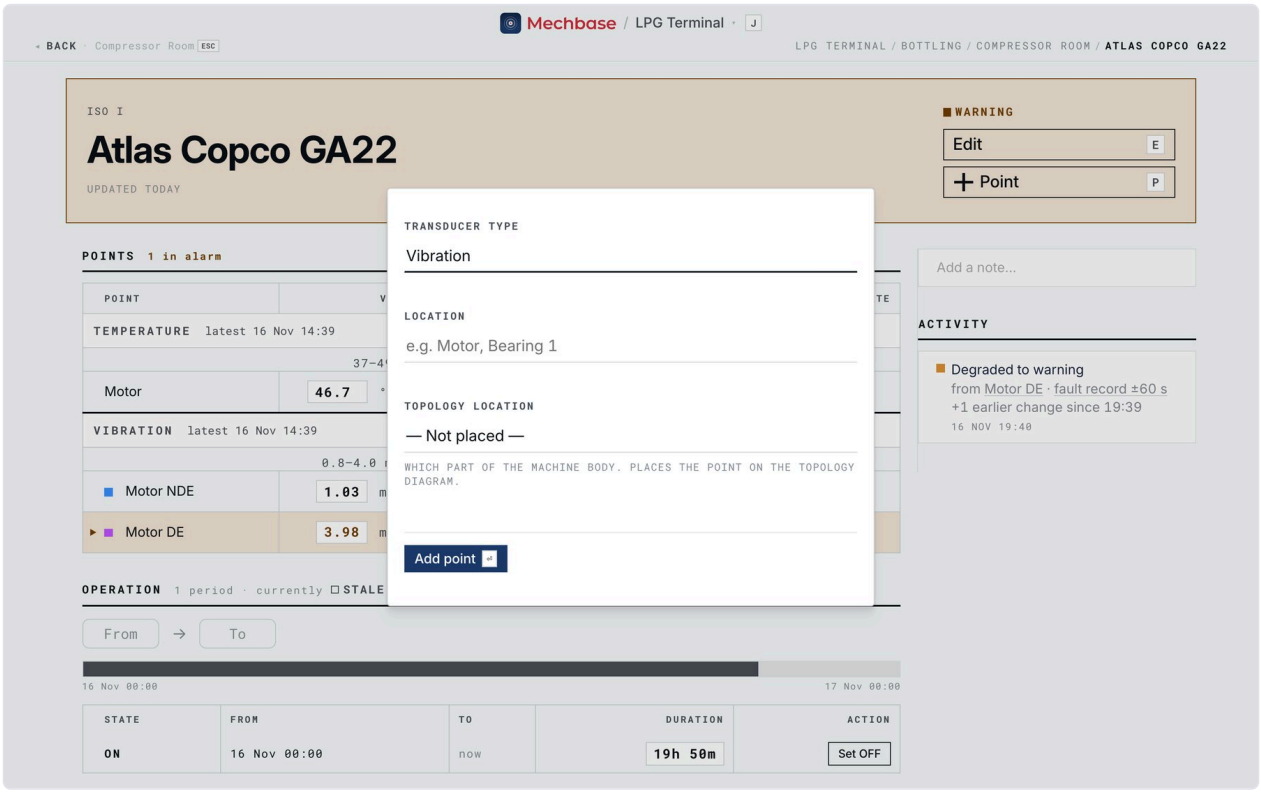


Figure 22 · Adding a measurement point

To change a point, open it and click **Edit** (`E`). The edit page is one scrollable form:

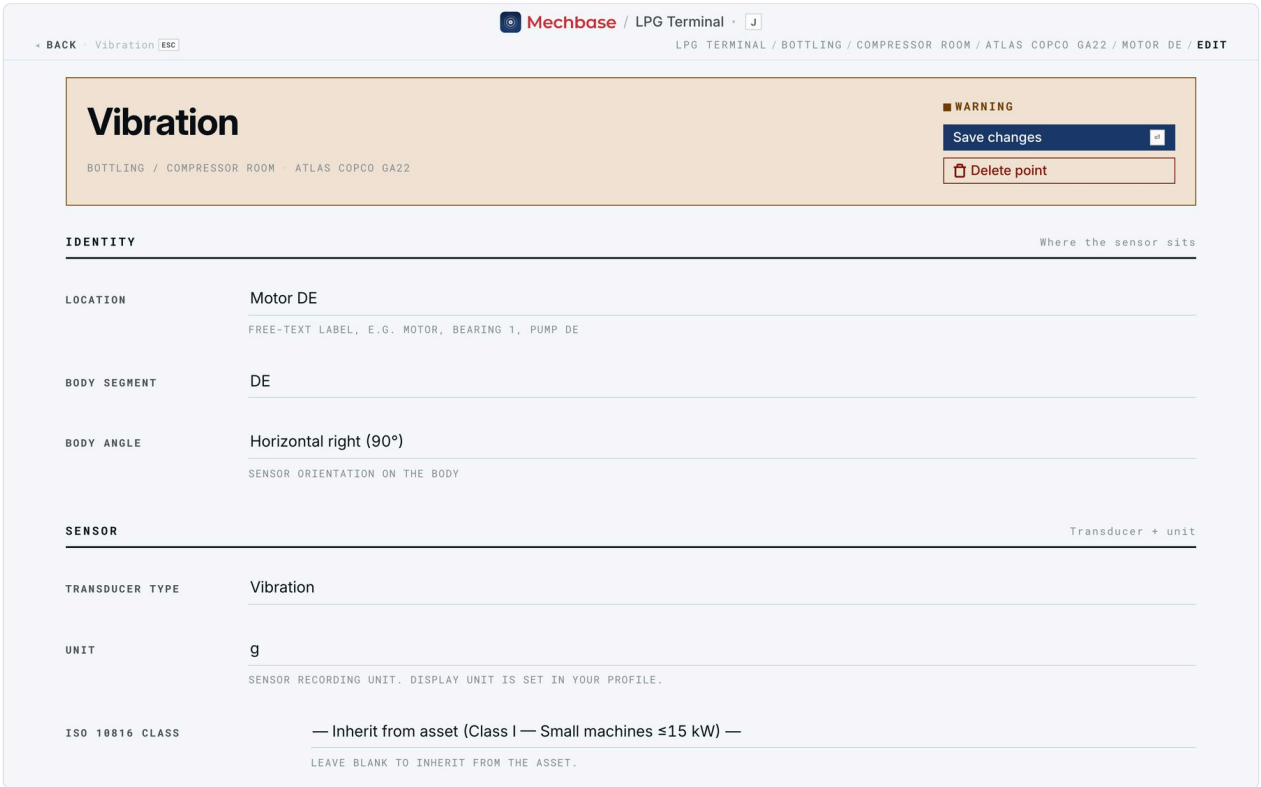


Figure 23 · The measurement-point edit form

- **Identity — where the sensor sits.** The free-text **Location** (*Motor DE*), and optionally the **Body segment** (NDE/DE) and **Body angle** (top, horizontal, bottom, axial) so Mechbase knows exactly where on the machine the sensor is.

- **Sensor — transducer and unit.** The **Transducer type**, the **Unit** the sensor records in (the unit *you* see is set in your profile), and the **ISO 10816 class** (inherit from the asset, or override).
- **Shaft & bearing** (vibration points). The **Nominal speed** in RPM from the nameplate; the **Bearing** fitted at this point, picked from the bearing library (Measurements, trends, and alarms); a **Speed ratio** if the bearing sits behind a gearbox or belt; and the **Envelope band** used to demodulate bearing faults (500 Hz–10 kHz if left blank).
- **Instructions.** Rich-text guidance shown to whoever takes a reading here — on the web and on the phone. Worth filling in for anything non-obvious: where to put the probe, which axis, what to watch for.
- **Photo.** A reference image of where the sensor goes, shown next to the instructions.
- **Sources.** Where automatic data comes from (next section).

Save changes (Enter) keeps your edits. **Delete point** removes it after a confirmation.

> HEADS-UP
> Deleting a point removes its readings and history too. If you only want to stop using it, consider whether editing is enough before you delete.

> TIP
> Good instructions make field readings consistent. For vibration, say where the load zone is and which way the sensor's X axis points; for thermography, say what to compare against (the other phases, the twin bearing); for ultrasound, say which probe and how hard to press. The same spot, the same way, every visit.

How a point gets its data: sources

A point can be fed two ways.

By hand. Anyone taking a reading — on the web with **Measurement**, or out in the field on the mobile app — records the value directly. The **instructions** above are there to make those manual readings consistent.

Automatically, from a source. At the bottom of the edit page, **Sources** lists the live feeds connected to this point. Each row shows where the data comes from — a **MaintNode** device, its acquisition group and channel, and the sensor type; or an **MQTT** device and channel — and when it last reported. Click **Add source** to connect one, or the pencil to change it.

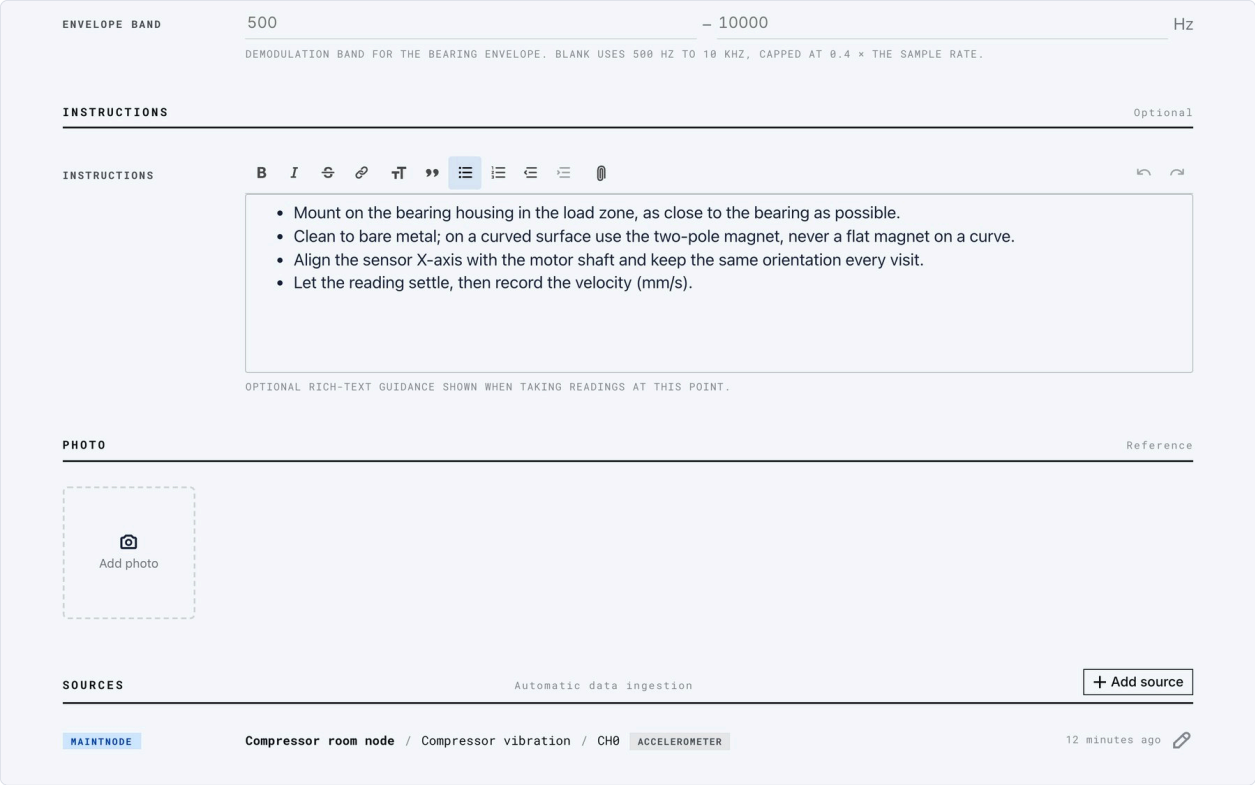


Figure 24 · A point's Sources: one MaintNode channel

These automatic sources are how MaintNode devices and your own systems feed readings straight into Mechbase without anyone typing them in — covered in MaintNode devices and automatic data.

Whichever way a reading arrives, it's treated the same: it's added to the point's history, checked against the alarm limits, and — if it crosses one — it updates the asset's state and shows up in the activity feed.

4 · Measurements, trends, and alarms

This chapter shows you how to read a point's trends, dig into a vibration reading's waveform and spectrum, record what you conclude, set alarm limits, and trust the single, reliable state Mechbase derives from a stream of readings — the part that keeps the whole system worth paying attention to.

Reading a trend

Every measurement point has a **Trend** — its readings plotted over time. It's the single most useful view for spotting a problem early, because a value that looks fine on its own often tells a clear story once you see where it's heading.

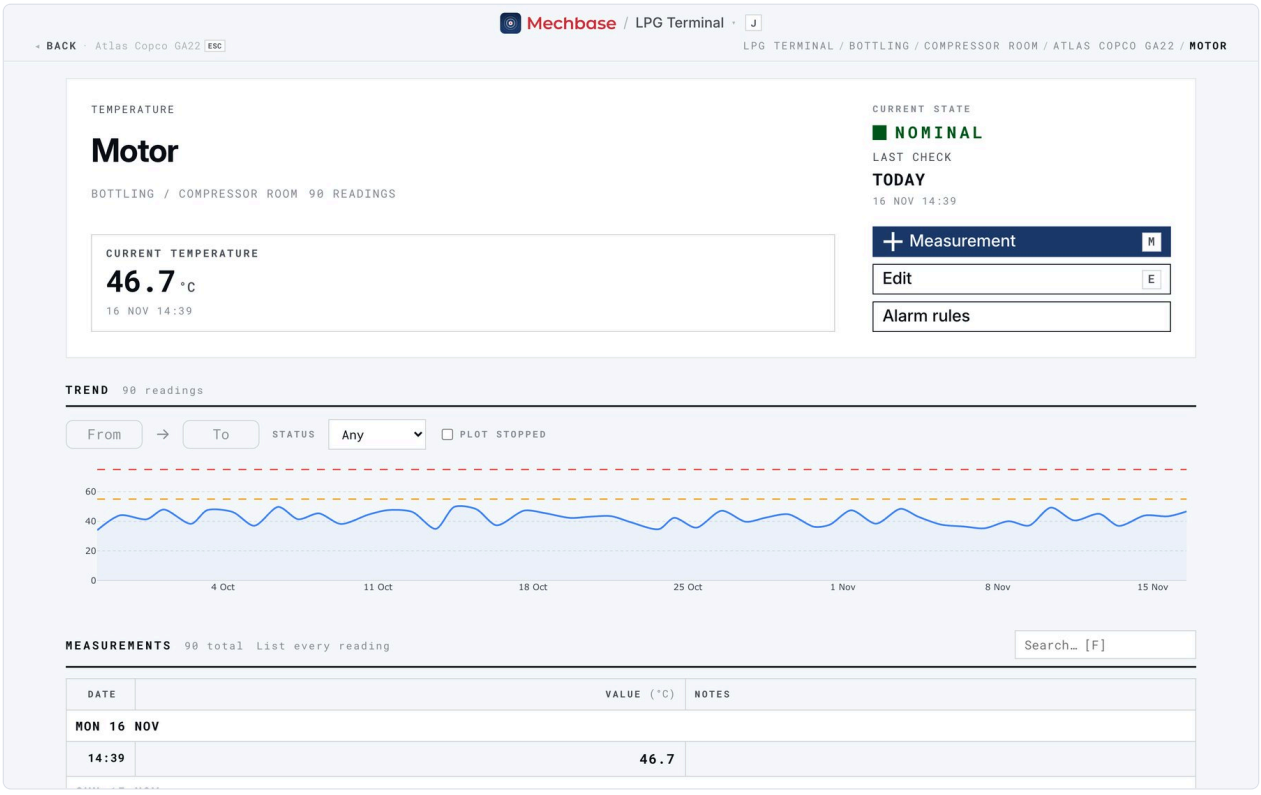


Figure 25 · A temperature point: headline, trend with alarm limits, and readings

For most points — temperature, pressure, ultrasound — the trend is one chart:

- **The line** is the reading over time. You're looking for direction and pattern — a steady climb, a step change, a new wobble — as much as the latest number.
- **The dashed limits** are the alarm thresholds: **amber** for warning, **red** for critical. How much headroom there is between the line and those limits tells you how hard the equipment is being pushed.

Vibration points show a **grid of small charts** instead — one per metric (*Acc RMS*, *Peak*, *Vel 10Hz*, *Vel 2Hz*, *O-P*, *P-P*, *CF*, *Kurt*), each spanning the week up to the latest reading. The metric the alarm watches carries its limits. Under each chart, **normal** gives the band the value usually sits in — the middle 80% of running readings over the previous 30 days — so you can tell at a glance whether today is unusual for *this* machine, not just whether it's under a limit. Hover any chart and a cursor follows on all of them; click a point to inspect that reading below.

When the machine has a tachometer, the grid adds **Speed**, **1X**, **1X phase**, **2X**, and **2X phase**. Each of these has a **Time | RPM** toggle: against RPM it becomes a run-up/coast-down (Bode) plot, where a resonance shows as a peak in amplitude with a swing in phase.

Remember that the units on every axis are *your* units, set in your profile (Getting started) — the same data reads in `Hz` / `mm/s` or `CPM` / `in/s` depending on who's looking.

Filters and time range

The controls above the trend let you focus on exactly the readings you care about:

- **From → To** — set a date range to zoom into a specific period, like the week a problem started.
- **Status** — show only readings of a given state (Nominal, Check, Warning, or Critical). Handy for jumping straight to every warning-level reading.
- **Waveforms only** (vibration) — only readings that carry a full waveform.
- **Plot stopped** (most other types) — by default the chart skips readings taken while the machine was stopped, and tells you how many it left out (*"N readings taken while the machine was stopped are not plotted. Plot them"*). Tick it to show them. On vibration charts, stopped readings are always shown, faintly.

The readings table

Below the chart, **Measurements** lists the readings in the chosen range, newest first, under a heading per day. Instead of hundreds of near-identical rows, Mechbase folds each stretch of one machine state into a **run**: *"06:00 → 14:00 · Running · 48 readings"*, with the low, typical, and high value and the worst state. Click a run to expand its readings. **List every reading** switches to a plain list; **Group by run** switches back.

The columns follow the point's type: for vibration, **RMS**, **Peak**, **Vel**, and **CF**; for ultrasound, the **Level** in dB; for thermography, a thumbnail, the **Max** temperature, **ΔT**, and the number of **Hotspots**; for everything else, the **Value**. Readings taken while the machine was stopped stay in the list, marked **Stopped**. The search box (`F`) finds readings by their notes.

Click a vibration reading to inspect it in place; click any other reading to open its own page.

Inspecting a vibration reading

For vibration, a single number isn't enough to diagnose a fault. Mechbase keeps the **full waveform** behind each reading, and the **Inspect** panel on the point page lets you look at it without leaving the trend. It opens on the latest running reading; click any point on the charts above to switch.



Figure 26 · Inspecting a vibration reading: time waveform and frequency spectrum

The panel's header repeats the reading's key values (RMS, Peak, Vel) and its state. If the reading came with a tachometer, it adds the shaft speed and the **1X** and **2X** amplitude and phase (*"1X 2.1 mm/s ∠ 47°"*); a phase Mechbase doesn't trust — because the speed wandered during the capture — is shown muted with the reason.

- **Accel / Vel / Env** — switch both charts between acceleration, velocity, and the demodulated **envelope** (for bearing defects).
- **Time waveform** — opens on about four shaft revolutions (or 200 ms without a tachometer), with ±RMS and the peak marked and the largest excursion flagged. Tachometer pulses appear as tick marks. Drag the overview strip underneath to move along the capture. **One rev** folds the signal into a single averaged revolution with its 1X fit — the clearest view of unbalance and looseness.
- **Frequency spectrum** — opens zoomed on the band that holds almost all the energy, with the five largest peaks labelled. **Hz | Orders** switches the axis to multiples of shaft speed, with 1X–10X order lines drawn in. Double-click for the full span. If a bearing is assigned to the point, its defect frequencies are marked too (Bearings).
- **Pin** — keep the reading for side-by-side comparison (Comparing readings).
- **Record finding** — write down your diagnosis (Recording a finding).
- **Open** — go to the reading's own page.

If the spectrum shows a strong tone that looks like a bearing but the point has no bearing assigned, Mechbase points it out — *"Possible bearing tone at 3.58×"* — with an **Assign bearing** button that lists the library bearings matching that tone. Nothing is assigned until you choose.

A single reading's page

Every reading has its own page — open it from **Open** in the inspect panel, from the readings table, or from a route report.

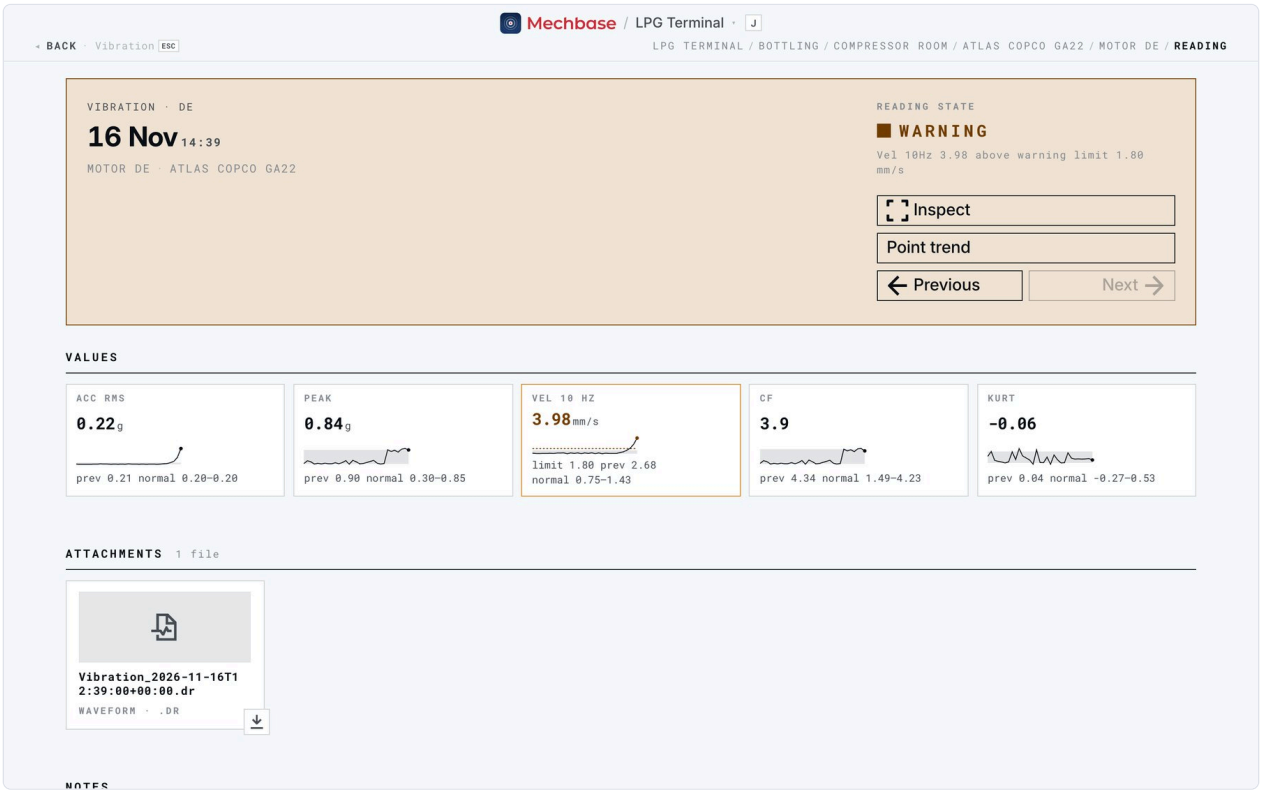


Figure 27 · A reading's page: each value in context

- The header shows the reading's state and, if it's in alarm, why (*"Vel 10Hz 3.98 above warning limit 1.80 mm/s"*), with who recorded it and how (a MaintNode, a route, or by hand). **Previous** and **Next** step through the point's readings; **Point trend** goes back to the point.
- **Values** — every value in context: a small trend of the last 40 running readings, the normal band, the limit, and a line comparing it with the limit, the previous reading, and the normal band (*"limit 1.80 · prev 2.68 · normal 0.79–1.43"*).
- **Attachments** — the files that came with the reading: the raw waveform (`.dr`), photos, and thermograms with their hotspots. Click to download.
- **Notes** — add a note to the record.
- For vibration, **Bench** gives you the waveform and spectrum with a little more control: choose the **Signal** (*Accel*, *Vel*, or *Env*), set a **High-pass** and **Low-pass** filter in Hz, and click **Apply** (or **Reset**).

Delete reading removes it straight away — no "Are you sure?" — and shows an **Undo** button for a few seconds in case it was a slip. The point's state is recalculated from the readings that remain.

Comparing readings

To compare readings — before and after a repair, this month against last — click **Pin** on each one in the inspect panel. The pinned readings collect in a tray; click **Analyze** to open the **Compare** drawer along the bottom of the page.

In the drawer you can switch between **Velocity**, **Accel**, **Envelope**, and **Waveform**, apply a high/low-pass filter, overlay the readings, show the spectrum in **dB**, label the **Top** peaks, and add **harmonic** cursors at a frequency you choose. **Record finding** works from here too.

Recording a finding

A **finding** is an analyst's conclusion about a reading — what's wrong, how serious it is, and what to do — written down where everyone can see it, and in a form a maintenance system can pick up.

Click **Record finding** on the inspect panel (or in the Compare drawer). If you clicked a peak in the spectrum first, its frequency, amplitude, and any matching defect (*"148.3 Hz · 2.31 mm/s · 3.05× · BPFO"*) are attached as evidence.

- **Fault** — *Bearing outer race (BPFO)*, *inner race (BPFI)*, *rolling element (BSF)*, *cage (FTF)*, *Unbalance*, *Misalignment*, *Looseness*, or *Other*. Mechbase suggests one from the peak you clicked.
- **Severity** — **Check**, **Warning**, or **Critical**.
- **Action** — what to do (*"Replace DE bearing"*), and a **Due by** date.
- **Comment** — what you see, in your own words.

Click **Record finding**. It's added to the asset's notes as a card with the severity, fault, action, due date, and evidence, and it appears in the activity feed. A finding can be edited but not deleted — it's part of the record.

Findings also have a work-order lifecycle — **Open**, **Acknowledged**, **Closed** — that your CMMS moves along through the API, attaching its work-order number (see the Analytics & BI and integration cookbooks).

Bearings and defect frequencies

When a vibration point knows which bearing it's watching, Mechbase can mark the bearing's defect frequencies on every spectrum, trend each defect band, and alarm on them.

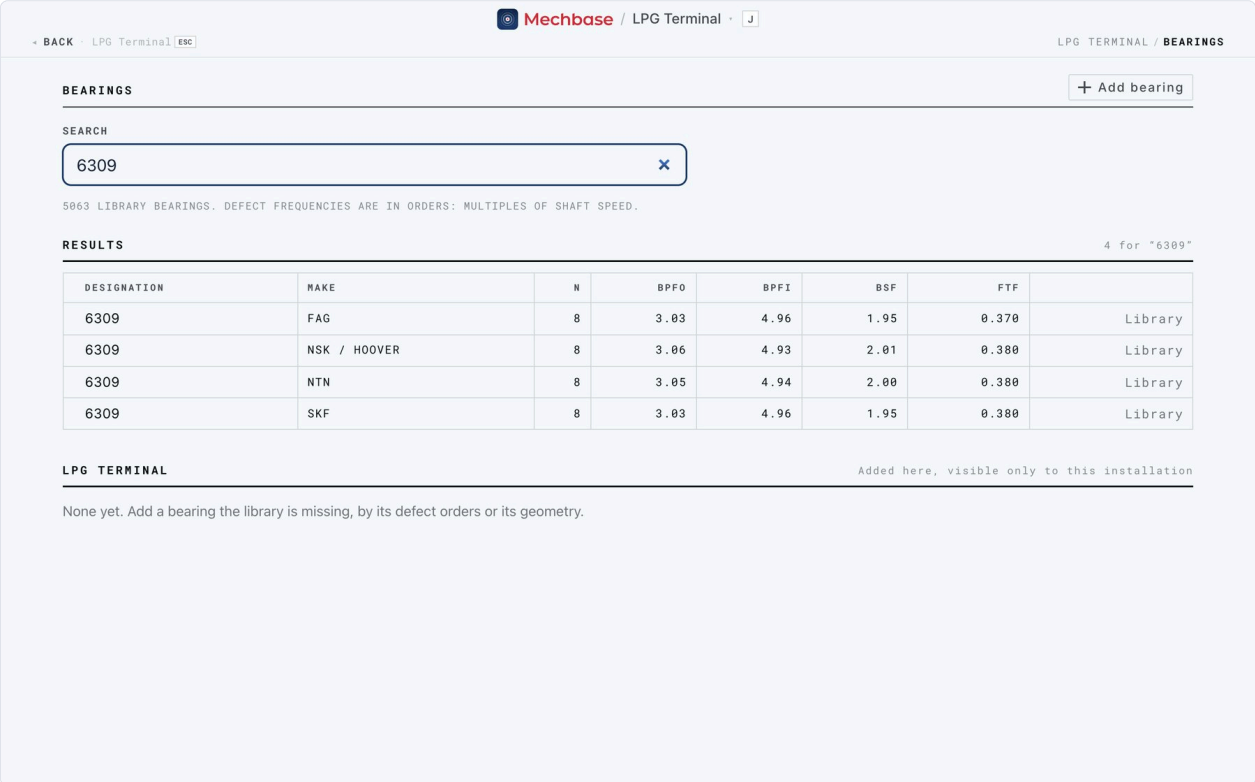


Figure 28 · The bearing library, searched for 6309

The library. Open **Bearings** from the menu (**Setup → Bearings**). Mechbase carries a library of several thousand rolling-element bearings. Search by designation or maker (*"6309"*, *"SKF 6205"*) — press `F` to jump to the search box. Each result shows the number of rolling elements and the four defect frequencies, in **orders** (multiples of shaft speed):

- **BPFO** — outer-race defect.
- **BPFI** — inner-race defect.
- **BSF** — ball spin.
- **FTF** — cage (fundamental train).

Your own bearings. If the library is missing one, click **Add bearing**. Give its **Designation**, **Manufacturer**, and **Rolling elements**, then either the four **Defect orders** from the maker's data, or its **Geometry** (ball diameter, pitch diameter, contact angle) and let Mechbase work the orders out. Bearings you add are visible only to your installation.

Assigning one to a point. Edit the point and, under **Shaft & bearing**, set the **Nominal speed**, search for the **Bearing**, and set a **Speed ratio** if it sits behind a gearbox or belt. From then on:

- Spectra show markers for shaft speed and **BPFO**, **BPFI**, **BSF** (with 2× and 3× harmonics) and **FTF**, placed using each reading's actual measured speed.
- The point's trends gain **BPFO env**, **BPFI env**, **BSF env**, and **FTF env** — the envelope energy in each defect band, reading by reading.
- You can set **bearing band** alarms on them (next section).

Setting alarm limits

Alarms are what make *Needs attention* mean something. They're defined by **alarm rules**, on three levels:

- An **installation default** for each measurement type — for example, "for any temperature point, warn at 55 and go critical at 75".
- For vibration on rotating equipment, the **ISO 10816 default** that follows from the asset's class.
- A **point override** for the cases where one specific point should behave differently.

Open a point and click **Alarm rules** to see what's in effect.

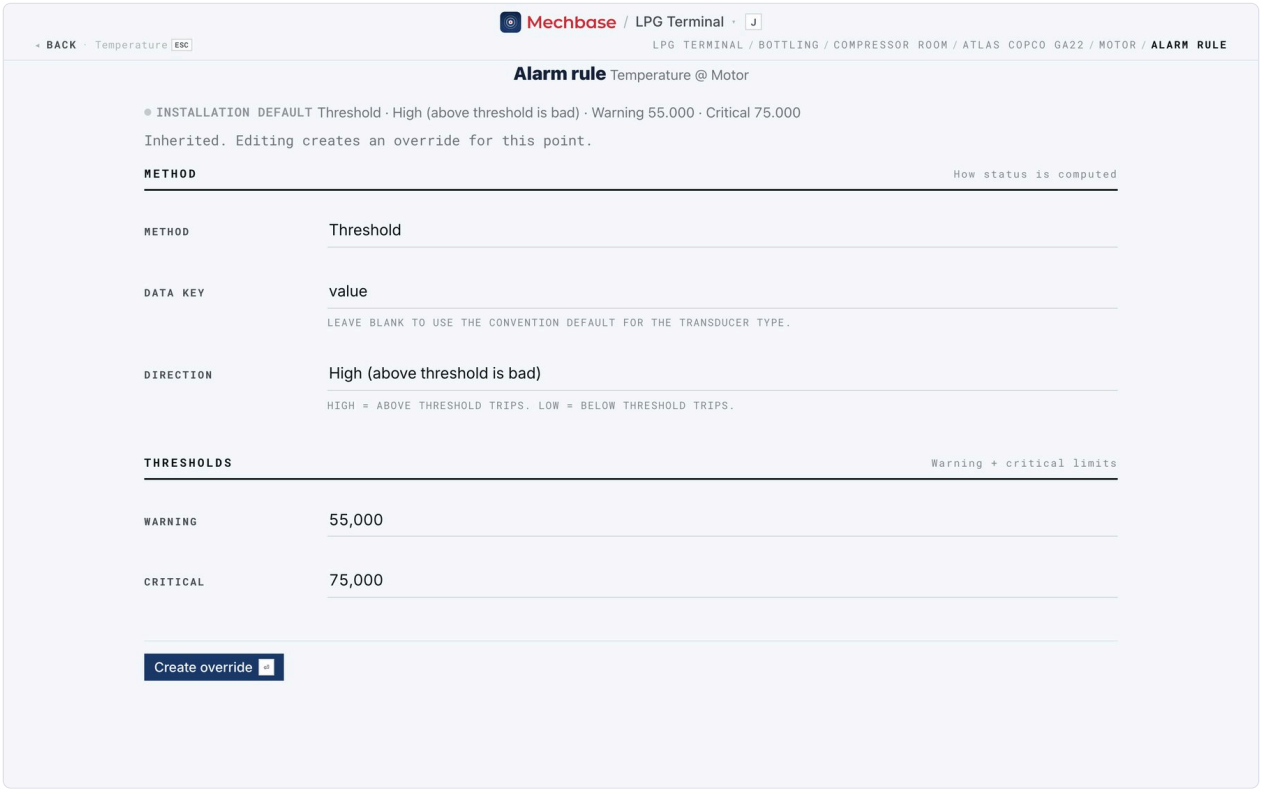


Figure 29 · The alarm-rule page for a temperature point using the installation default

The line at the top tells you where the rule comes from — **Point override**, **ISO … default**, **Installation default**, or **No rule configured** — and what it says (*"Threshold · High (above threshold is bad) · Warning 55 · Critical 75"*). When the point is inheriting a rule, the form below is already filled in with it: *"Inherited. Editing creates an override for this point."*

An alarm rule has two parts:

- **Method** — how the status is worked out:
  - **Threshold** — a simple limit on the value. The common case.
  - **Narrowband Envelope** — watches a specific frequency band, for spotting a particular vibration fault.
  - **Maintenance due** — raises a status based on time or usage, for condition-plus-schedule maintenance.

  Plus the **Data key** (which value of the reading to watch — blank uses the usual one for the type) and the **Direction**: **High** (above the threshold is bad — temperature, vibration) or **Low** (below the threshold is bad — a flow or pressure that shouldn't drop).
- **Thresholds** — the **Warning** and **Critical** limits.

Adjust them and click **Create override** (or **Create rule** where no default applies, or **Save rule** when you're editing an existing override). The new limits appear immediately on the point's trend.

To go back to the inherited rule, use **Remove override** at the bottom. It takes effect at once, with an **Undo** in case you didn't mean it.

Bearing bands. On a vibration point with a bearing assigned, the page adds a **Bearing bands** section. It shows the current verdict for the defect bands and any band rules in force. Each **Band rule** picks a **Band** (or every envelope band), a **Mode** — an **Absolute value**, or a **Multiple of baseline** (3× for warning and 6× for critical if you leave them blank) — and the **Warning** and **Critical** limits; click **Save band rule**. Relative rules compare against a baseline Mechbase learns from the first five waveform readings while running; **Reset baseline** starts a fresh one — do it after a bearing change, a repair, or a remount.

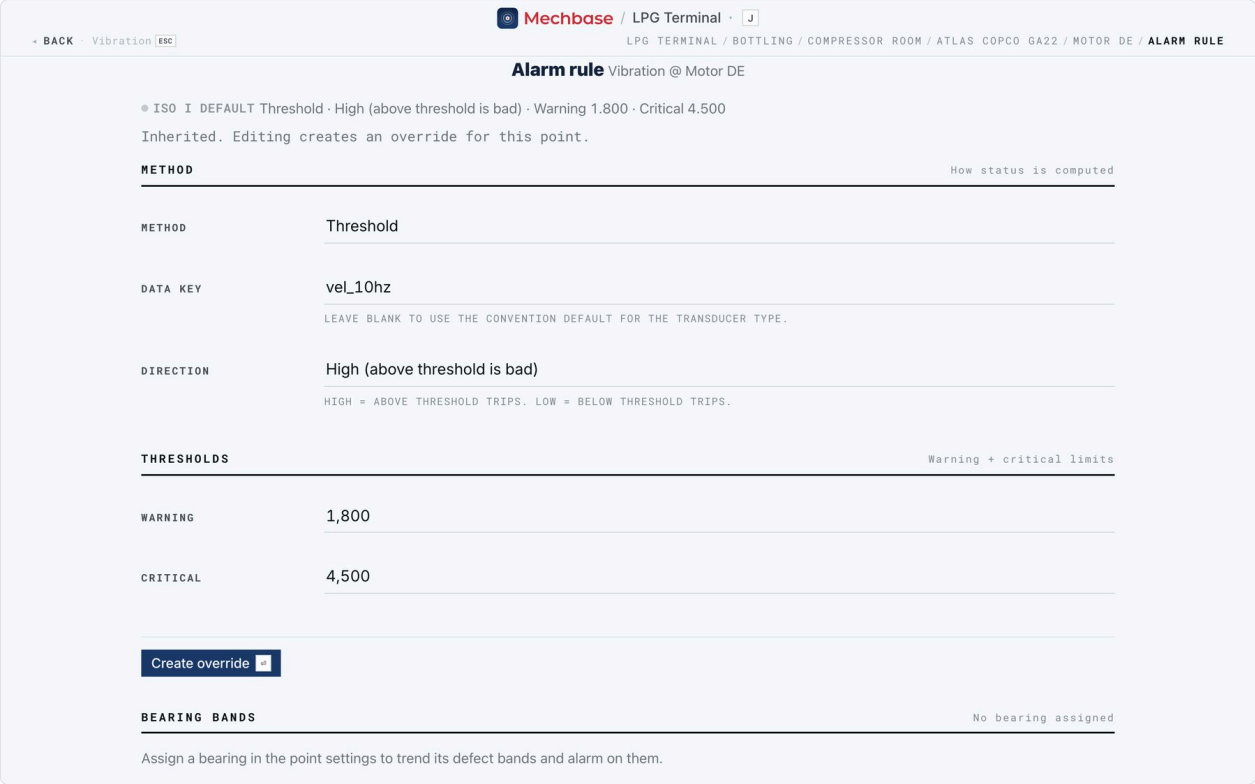


Figure 30 · Alarm rule for a vibration point, with the Bearing bands section

TIP

Base your limits on history and expert judgement, not a guess. The **normal** band under each chart, and the *limit · prev · normal* line on a reading's page, show you where the value really sits. Set warning and critical limits that sit clearly above (or below) that band.

NOTE

Installation-wide defaults are maintained centrally by your Mechbase administrator (Setting up and managing your account). Point overrides are yours to set.

Keeping false alarms down

A predictive-maintenance system is only worth as much as people trust it. If it cries wolf — firing an alarm every time a sensor twitches or a reading blips — teams stop paying attention, and the genuine faults get lost in the noise.

Mechbase is deliberately built to avoid that. Instead of tripping on a **single** reading that crosses a limit, it looks at the **recent pattern**:

1. **Every reading gets a status** against the alarm limits — *normal*, *minor* (a little past the limit), or *major* (well past it).
2. **Mechbase averages the status over the last several readings**, not just the latest one.
3. **The state changes only when that average crosses a threshold** — so a sustained run of off readings moves the asset to Warning or Critical, while a lone spike does not.

On top of that, Mechbase knows when a machine is **stopped** and keeps those readings apart from the running trend, and relative rules compare each machine against **its own learned baseline** rather than a one-size-fits-all number.

That's why the states are **Check / Warning / Critical** rather than a blunt on/off: **Check** is Mechbase telling you "a soft pattern is forming here" before it ever escalates. The payoff is fewer false alarms, warnings you can act on with confidence, and maintenance effort spent on real problems instead of chasing ghosts.

NOTE

Two things shape how responsive this is — the **alarm limits** (from history and expertise) and how many recent readings are averaged. A longer window is steadier but slower to react; a shorter one reacts faster but tolerates less noise. Treat the first weeks on a new point as a "learning" period and review both.

How a reading becomes a state

Pulling the whole chapter together, here's the journey of a single reading — whether it arrived automatically from a MaintNode device or was typed in on a route:

1. **It's recorded** against its measurement point and added to the trend.
2. **It's scored** against the point's alarm rule — normal, minor, or major.
3. **It's combined** with recent readings (the false-alarm logic above) to decide the point's state.
4. **The asset's state** rolls up from its points and its open non-conformities — the worst of them sets the asset's state, and feeds the **in alarm** count.
5. **Everything downstream updates**: the asset's colour, **Needs attention**, the section cards, any dashboards, a plain-language line in the **activity** feed (*"degraded to warning"*), and an email to everyone who asked for one at that level.

Running-state readings (*Production*) and on/off **State** readings take a different path: they open and close periods rather than being scored and averaged.

So the numbers you act on across Mechbase aren't opinions or stale spreadsheet entries — they're the live result of this pipeline, running on every reading.

5 · Inspections and routes

Routes let your team run repeatable inspection rounds and turn what they find into tracked issues — the work that sensors can't do for you: a visual check, a manual reading, a safety test. This chapter covers the whole cycle: what a route is, how to build one, how to run it (on the web here, on the phone in Mechbase on your phone (iOS & Android)), and how findings turn into non-conformities and a report.

What a route is

A **route** is a defined inspection round — a named list of things to check, in a sensible walking order. It's made of two building blocks:

- **Stops** — the places you visit, in order: a **zone** (*Power House*) or a specific **asset** (*Caterpillar C9 genset*).
- **Checks** — the individual items you complete at each stop.

Every check has a **type**, which decides what you record:

- **Pass / Fail** — a yes/no judgement, like *"fire extinguisher in date"*.
- **Numerical** — a number you read off a gauge, like a room temperature, in the unit you set on the check.
- **Multi-check** — several pass/fail steps bundled into one check (for example, a panel's emergency stop, relays, interlocks, and fan).
- **Measurement** — a reading taken on one of the asset's measurement points (a vibration, a temperature, a thermal image), which flows into that point's trend just like an automatic reading. A thermography check is a Measurement check on a thermography point.

Each time someone walks a route, they create a **run** (an execution). The route itself is the template; the runs are the history. A route has **at most one run in progress at a time** — if a colleague has already started today's round, starting it again joins their run rather than creating a second one.

Seeing your routes

Open **Routes** from the menu, or by pressing `2`.

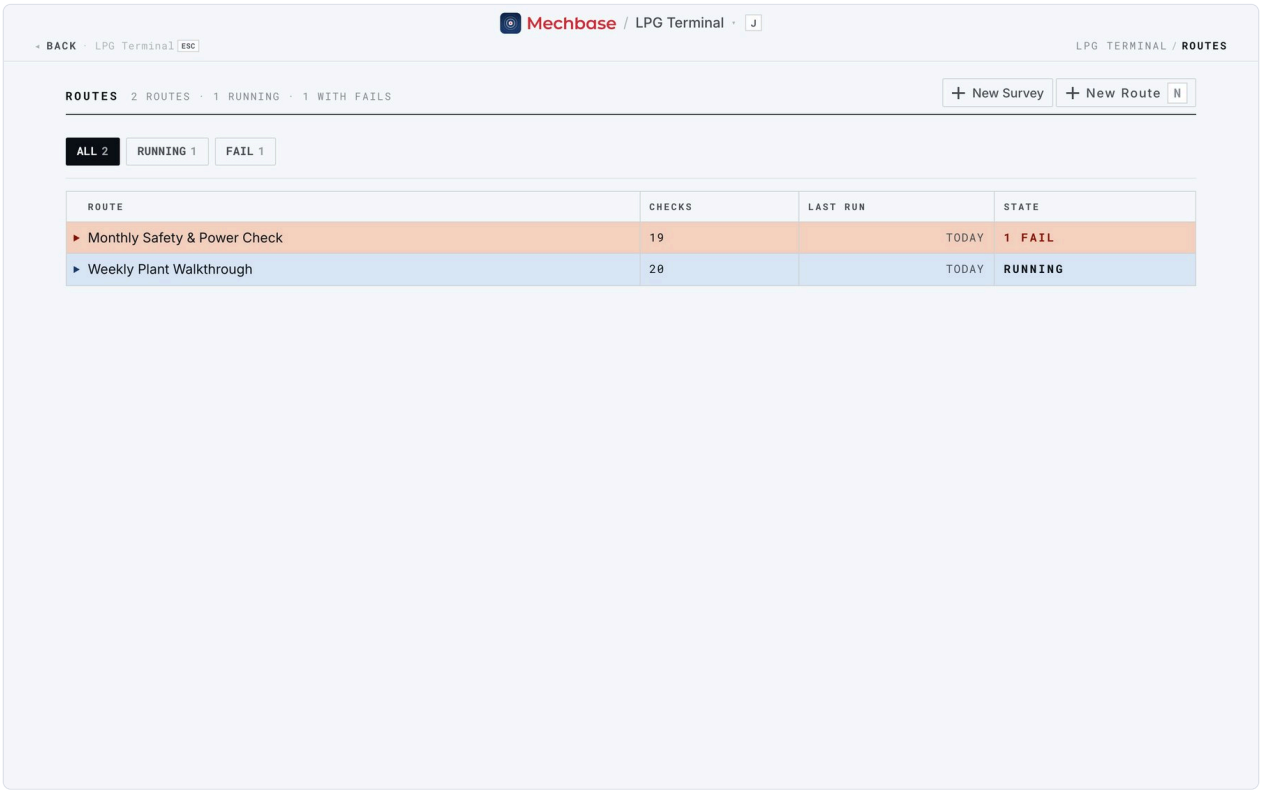


Figure 31 · The routes list

You get every route for the site, with its number of checks, when it last ran, and its state. The filter pills — **All**, **Running**, **Fail**, **Completed**, **Never run** — each show a count, so you can see at a glance what's in progress and what found problems. Start a new route with **New Route** (`N`), or a survey with **New Survey** (Surveys). Click a route to open it.

A route's own page is its launchpad and its latest result:

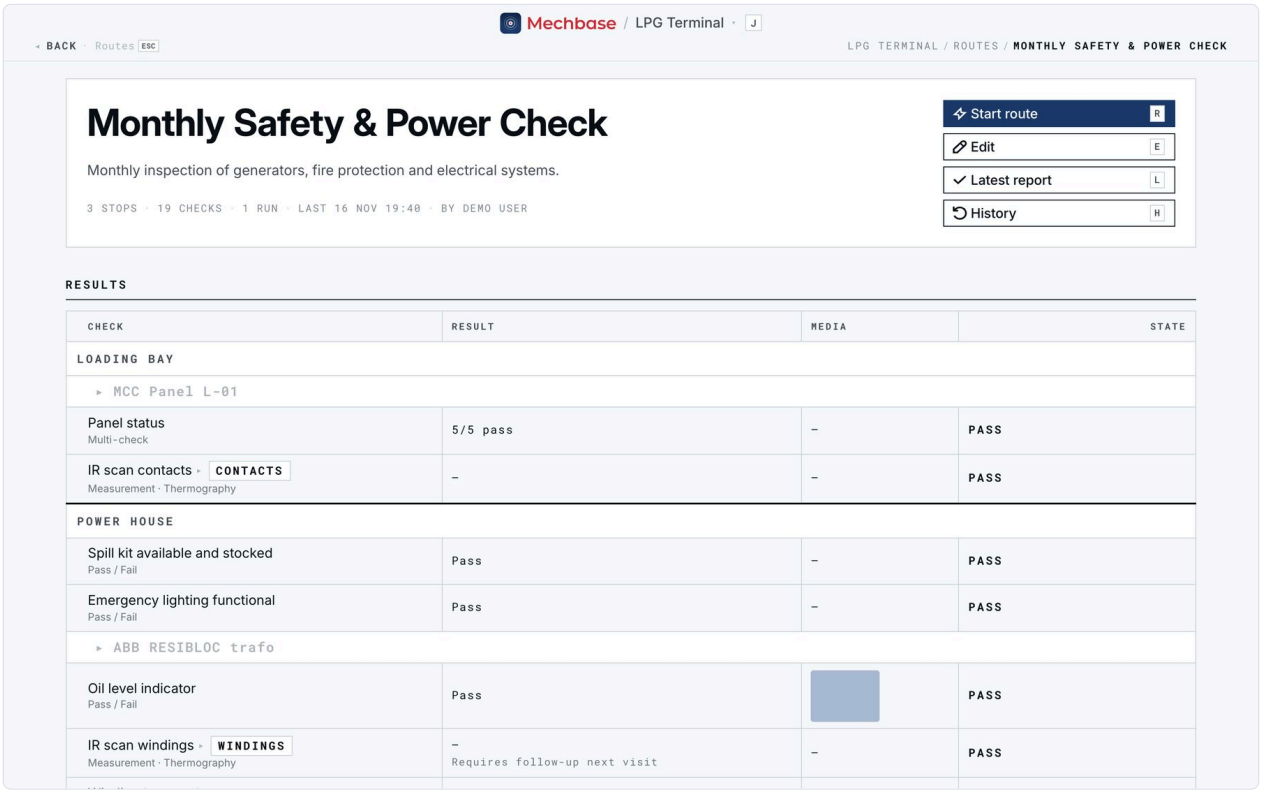


Figure 32 · A route's page: actions, and the latest run's results

At the top are its description and totals — **stops**, **checks**, **runs**, when it last ran and by whom — and four actions:

- **Start route** (`R`) — begin a run. If one is already in progress, the button reads **Resume route** with a **Running** marker, and takes you into it.
- **Edit** (`E`) — change the checklist.
- **Latest report** (`L`) — open the most recent run's report.
- **History** (`H`) — every past run.

Below, once the route has been run, you see the **Results** of its latest run, grouped by stop and then by asset, so the page answers "how did it go last time?" straight away. A route that has never been run shows its **Checklist** instead.

Building a route

Create a route with **New Route** (`N`). Give it a **Route name** and an optional description, then click **Create route**.

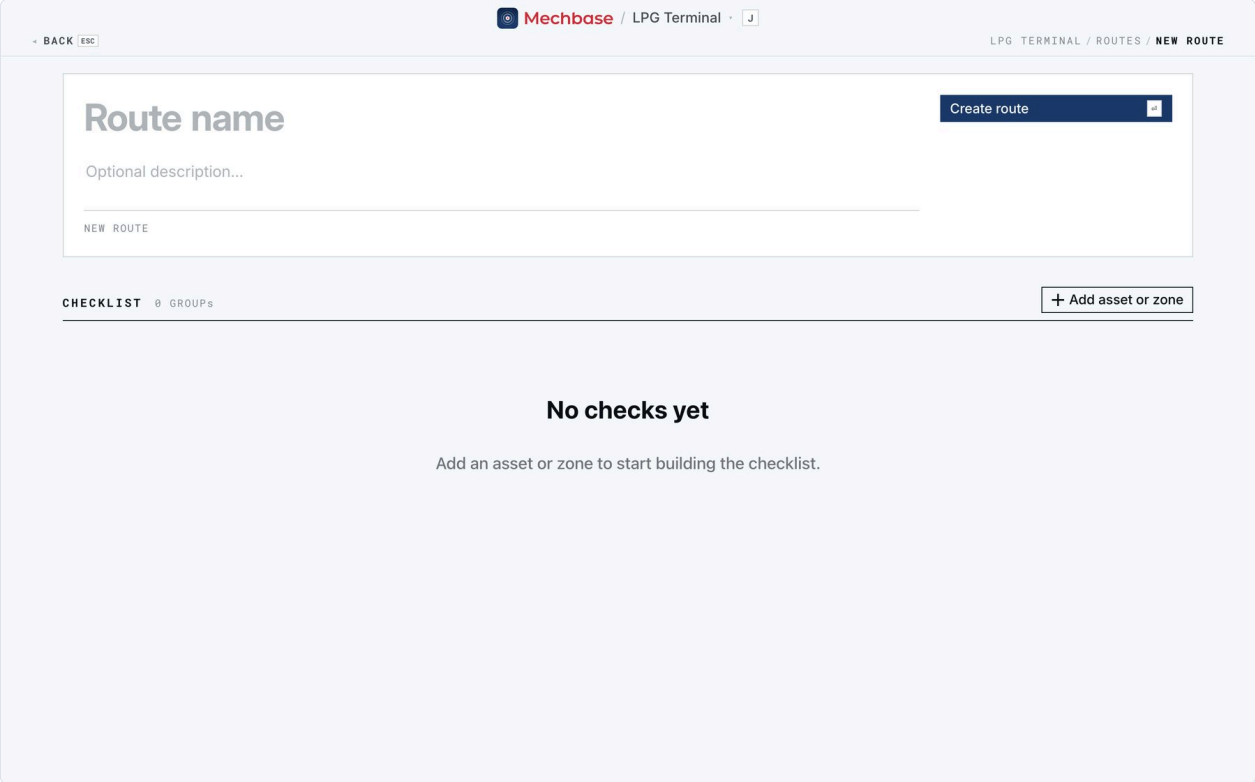


Figure 33 · Starting a new route

A new route starts empty. Build its checklist in the editor (**Edit**, `E`):

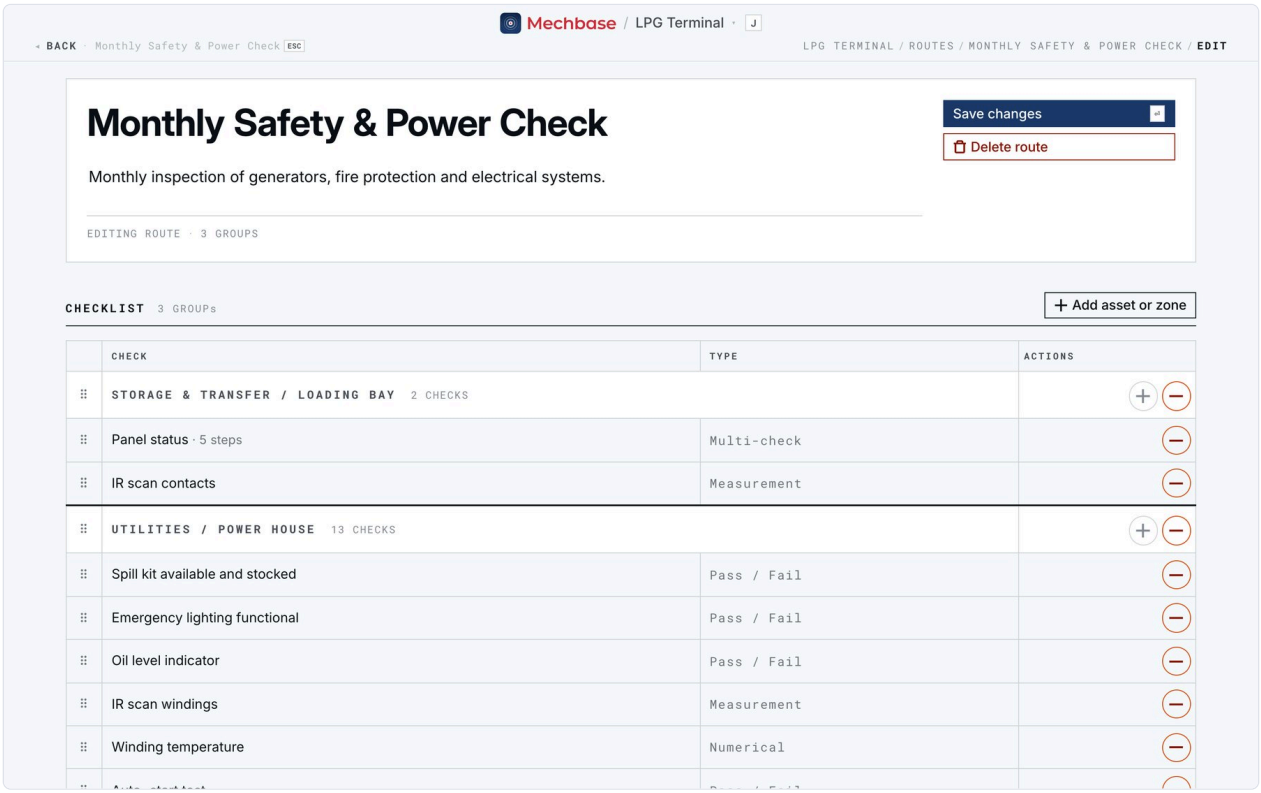


Figure 34 · The route editor

1. Click **Add asset or zone** and pick a stop from the list (type to filter). A stop can be a whole zone or a single asset.
2. On each stop, click **+** to **Add check**. Give it a **Label**, choose its **Type**, and then:
   - for a **Measurement** check, pick the **Measurement point** it reads;
   - for a **Numerical** check, set the **Unit** (*mm/s*, *°C*, *bar*…);
   - for a **Multi-check**, add each **Step** — every step becomes its own pass/fail tick during the run.
3. Drag the handles to reorder stops and checks so the route follows the natural walking path of the plant, or use **Move to different zone or asset**.
4. Click **Save changes**.

A check's type is fixed once it's created — changing it would break the history of past runs. Remove a check with **–**, or open it and use **Delete check**; deleting shows an **Undo**, and past runs keep their answers to it either way. **Delete route** removes the whole route after a confirmation.

> **TIP**
> Order the stops the way someone actually walks the site, and keep each check's name short and unambiguous — it's what the technician reads on their phone at 6 a.m. Put the *how* in the measurement point's **instructions**, which appear on the check during the run.

Running a route

Click **Start route** (`R`) — or **Resume route** if one is in progress — to open the live run.

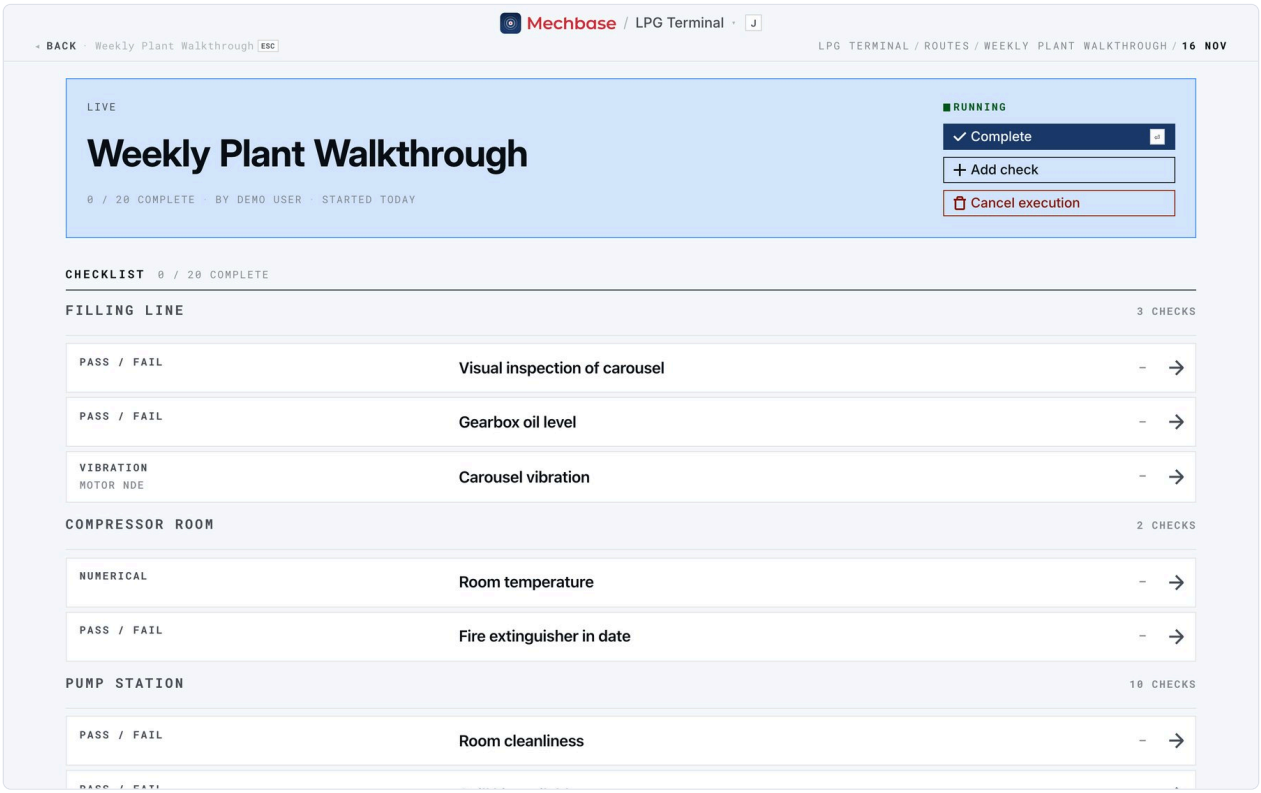


Figure 35 · Running a route — the live checklist

The header shows **Live**, the route, how many checks are complete, who's running it, and when it started. Below is the checklist, grouped by stop, with each check's type and current answer. Click a check to record it:

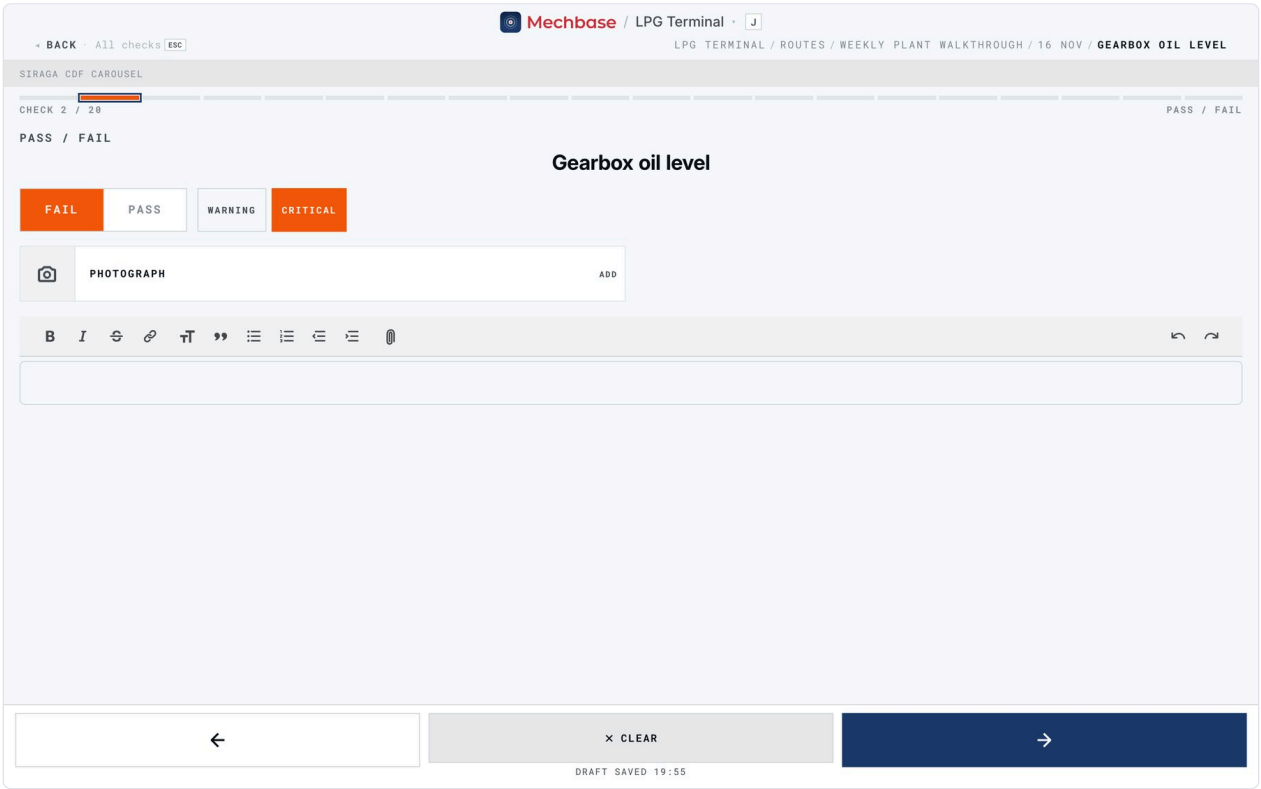


Figure 36 · Recording a check: a failure graded Critical

Each type shows the right input — here a **Pass / Fail** check. When you choose **Fail**, pick how serious it is: **Warning** or **Critical** (Critical is preselected). Multi-check steps are graded the same way, one by

one. Every check also has room for **evidence**: **Photograph** to attach photos, and a notes field — your draft is saved as you type.

Use the arrows at the bottom to move to the previous or next check (on the last check, the arrow reads **Finish route**), and **Clear** to undo an entry. Back on the checklist:

- **Complete** (`Enter`) finishes the run and saves it as a report.
- **Add check** adds something you spotted that isn't on the list: give it a **Label** and a **Kind** — Pass / Fail, Multi-check, or a reading such as *Ultrasound / leak (dB)*, *Temperature (°C)*, *Vibration (mm/s)* — and optionally a **Zone** (or *No zone — triage later*). Click **Add to route**; it's added to the route for next time too.
- **Cancel execution** abandons the run. Mechbase asks first, because every answer in it is discarded.

> **NOTE**
> A run doesn't have to be finished in one go — leave it and **Resume** later, which is exactly what happens when a round spans a shift. Because a route only ever has one run in progress, colleagues picking up the same round all add to the same run. On the mobile app this also works offline; see Mechbase on your phone (iOS & Android).

Surveys: routes you build as you walk

Sometimes there's no route yet — a first walk-down of a new area, a leak survey, an audit. Click **New Survey** on the routes list, keep or change the suggested name (*"Survey — 16 Nov 19:48"*), and click **Start survey**. You go straight into a live run with an empty checklist, and add each check with **Add check** as you find it.

When you complete it, the survey becomes a normal route you can run again. If you cancel a survey before recording anything, it's removed altogether.

Logging a non-conformity

A **non-conformity** is simply a check that didn't pass — a failed Pass/Fail item or step, or a reading that breached its limits. You don't log it separately; it's created the moment you record a failing result, with the severity you chose (**Warning** or **Critical**), and the **photos and notes** you add at that check become its evidence.

From there, a non-conformity travels with the asset, not just the inspection:

- It's pinned at the top of the affected **asset's Activity** (Working with assets), and raises the asset's state — a Critical non-conformity makes the asset Critical.
- It appears on **Needs attention**, with the failed check as the trigger.
- It's cleared with **Resolve** on that asset, once the problem is dealt with, with an optional resolution comment.

This is the link between a person spotting a problem on a round and that problem being tracked to closure against the right asset. If your site is connected to a CMMS, non-conformities can also be pushed there as maintenance tasks automatically (Appendix D).

The inspection report

Every completed run is saved as a report. Reach it with **Latest report** (`L`) or **History** (`H`) on the route, or by clicking the inspection in the activity feed.

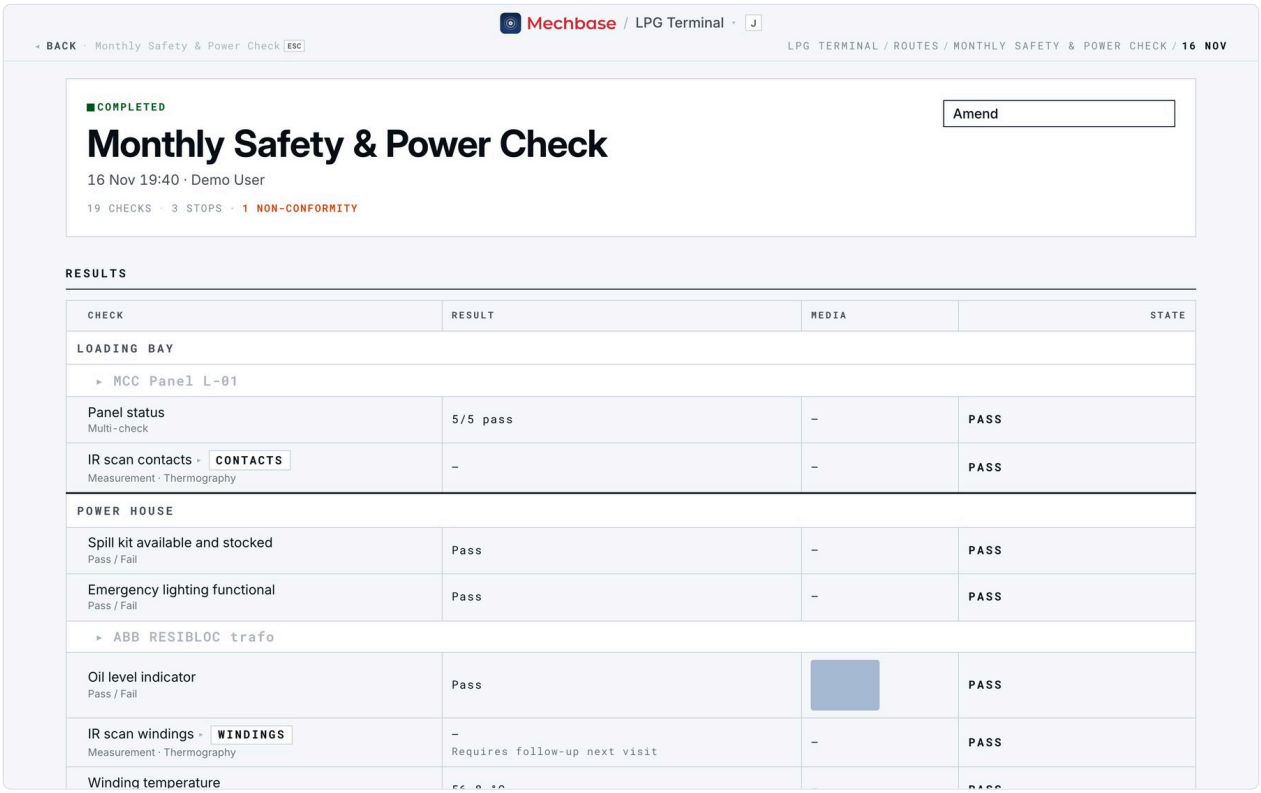


Figure 37 · A completed run — the inspection report

The report records who ran it and when, the totals (checks, stops, and the number of **non-conformities**), and a **Results** table with columns **Check**, **Result**, **Media**, and **State**. Checks are grouped by stop and then by asset — with a small topology diagram of the asset, its points coloured to match. The check's type sits under its name, and any notes under its result. Only failed checks are coloured, so a director skimming the report sees the exceptions immediately.

The **Result** column shows what was actually recorded: *Pass*, a number with its unit, *5/5 pass* for a multi-check, or for a measurement the reading itself (*"Vel 2.1 mm/s · RMS … · Peak …"*, *"Temperature 45.5 °C"*, *"3 hotspots · ΔT …"*).

Open a row for the detail. Click a measurement or a failed check to expand it:

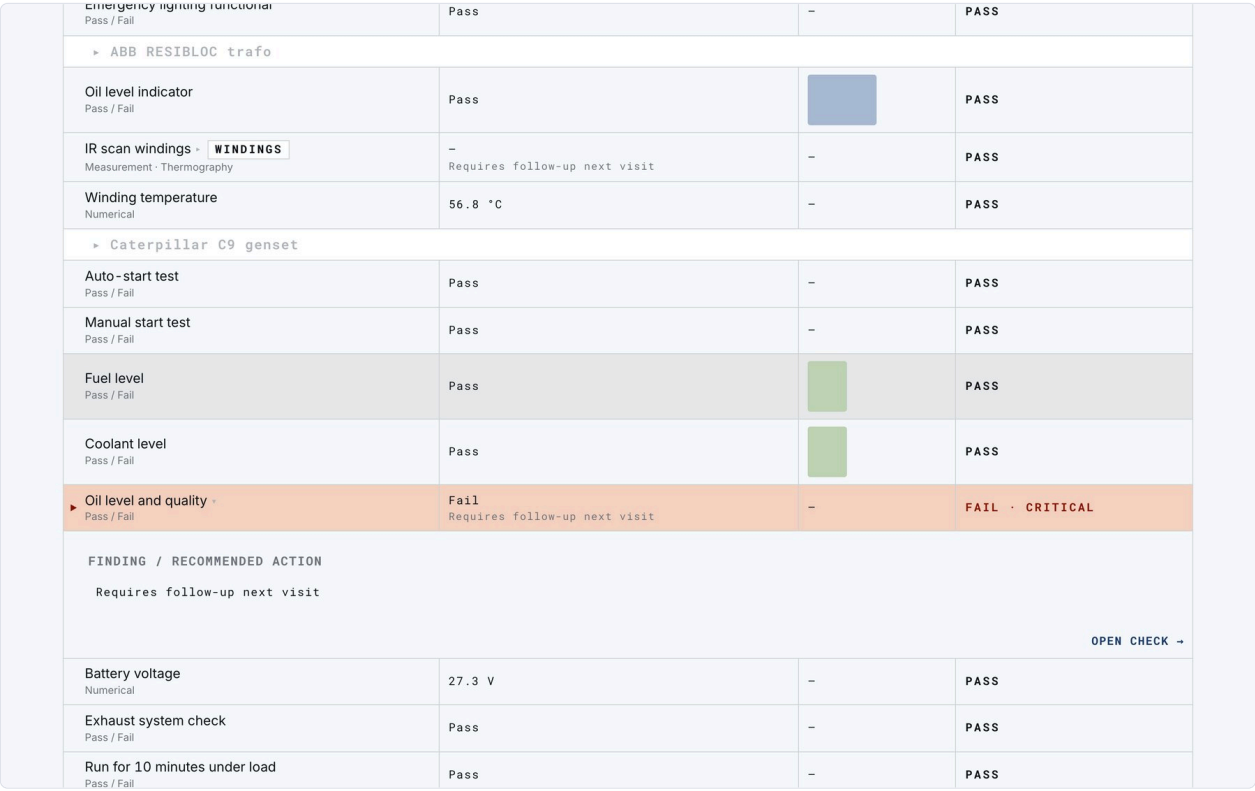


Figure 38 · An expanded report row with its finding and recommended action

- For a measurement: a trend of the point around the run, with its limits; the **Previous**, **Current**, and **Trend** change; for vibration, the full breakdown (*Vel 10Hz/2Hz*, *RMS*, *Peak*, *O-P*, *P-P*, *CF*, *Kurtosis*) and the waveform and spectrum.
- **Finding / Recommended action** — what was found and what should happen next, editable right there on the report.
- **Open check →** to go to the check itself.

The state words read **Pass**, **Fail · Warning**, or **Fail · Critical** for checks, and the reading's state for measurements. **Print** the report from your browser and every row prints expanded, without the charts and controls — ready to hand over or file.

History and amending a run

History (`H`) lists every run of the route as a card: **Running** or **Completed**, when and by whom, the result (*18/19*), and the worst state found.

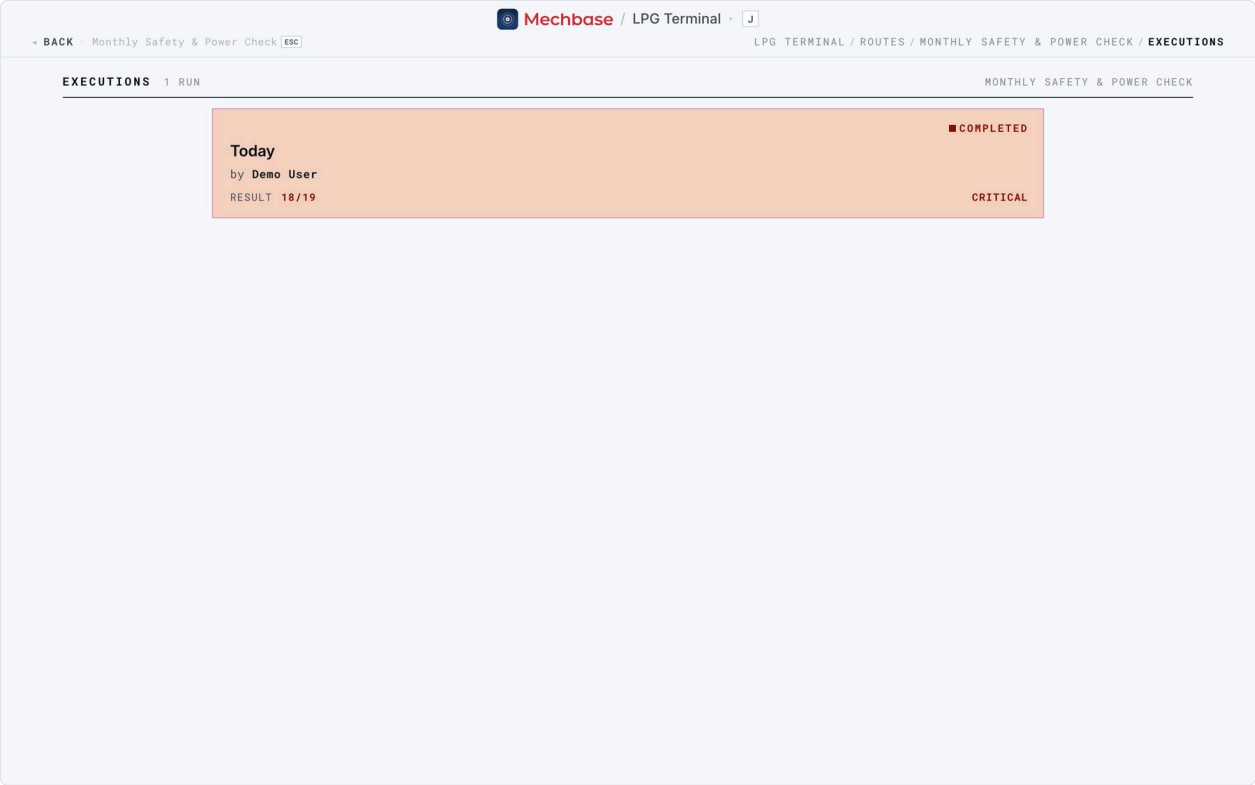


Figure 39 · A route's history

Cancelled runs that never recorded anything are left out. At the foot of a report, **View earlier runs of …** takes you here too.

Amending a run. Things get missed — a skipped check, a photo left on the phone, a reading taken after the run was closed. The technician who ran it (or a staff user) can click **Amend** on the report to:

- answer checks that were skipped (they're counted as *Not answered*);
- record or replace a reading — the old one stays in the point's history;
- add or remove photos — removed photos are kept on the run, not deleted;
- edit notes.

Recorded results themselves can't be changed. Every amendment is logged with who did it and when, and the asset's state is recalculated. Click **Done** to go back to the report.

A few words tie this together:

- A run is **Running** while it's in progress, **Completed** once finished, or **Cancelled**.
- A route shows **Running** while it has a run in progress.

Between the automatic trends of Measurements, trends, and alarms and the human inspections here, Mechbase holds the complete maintenance record for every asset: what the sensors saw, what people found, and what's still open.

6 · Staying on top of what needs attention

Mechbase gathers every asset that's no longer healthy — whether a sensor flagged it or an inspector did — onto one **Needs attention** screen, so you can work through problems by priority, watch them clear as you fix them, and have the important ones reach you by email when you're not looking.

The Needs attention list

The home page's **Needs attention** panel shows the top six items; the full list lives on its own page. Open it from the **+N more** link on the panel, from **Needs attention** in the menu (its badge shows how many are active), or by pressing `3`.

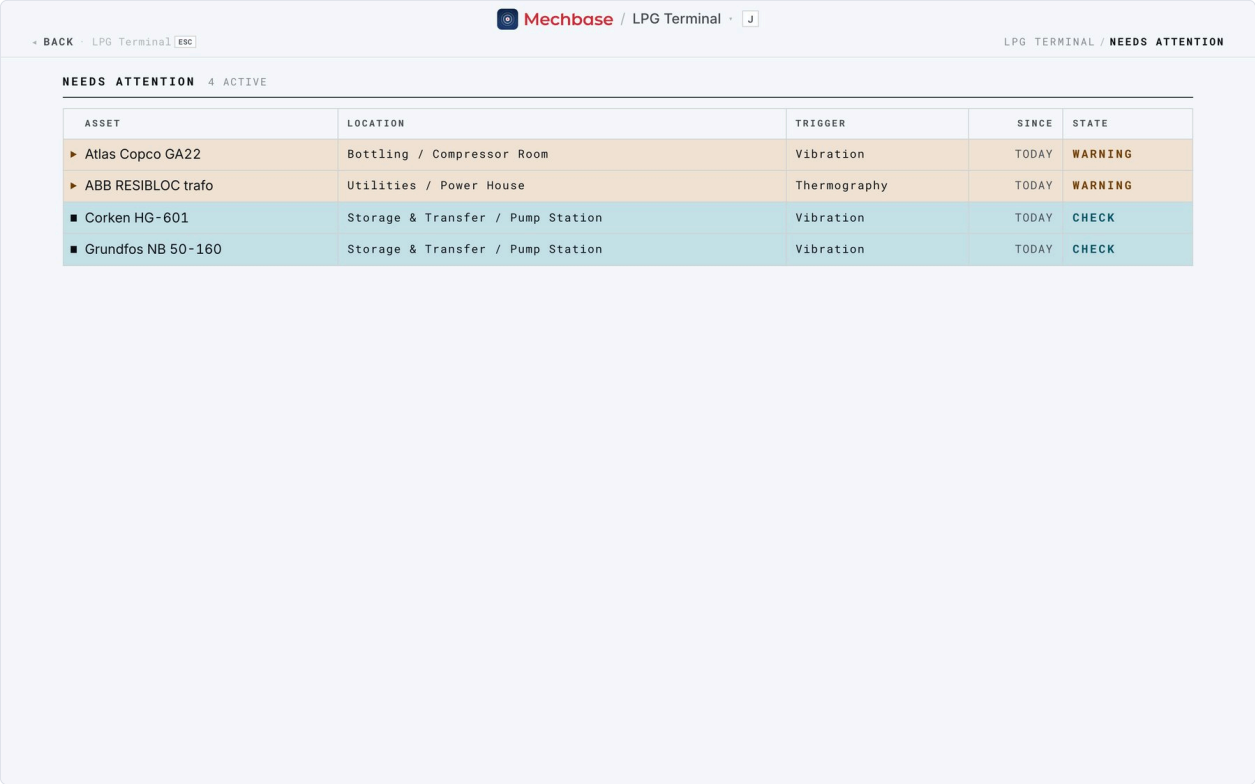


Figure 40 · The full Needs attention list

It's every asset at the site that isn't currently Nominal, ordered most urgent first — Critical, then Warning, then Check, and within each by section and name:

- **Asset** — what it is; click the row to act.
- **Location** — `Section / Zone`.
- **Trigger** — what raised it: the measurement point that's off (*Motor DE*), or the inspection check that failed (*Oil level and quality*) — whichever is worse.
- **Since** — how long it's been in this state.
- **State** — Check, Warning, or Critical.

This is your single, current backlog of asset health. If nothing needs attention, the page says *"All assets are nominal."*

Silent assets

A sensor that stops reporting is a problem of its own — the machine might be fine, or it might be failing unseen. So after the health alerts, the list shows **Silent** assets: ones whose points used to report but haven't sent a reading in 30 days, or whose live MaintNode feed has stopped. Their trigger reads *"No reading: …"* with the points concerned, and the home page heading counts them separately (*"4 active · 1 silent"*).

Treat a silent asset as a job too: check the sensor and its device, or the route that should have measured it.

Working through them

Use the list the way it's sorted — from the top:

1. **Start with Critical, then Warning.** The order already puts the most urgent first, so working top-down is working by priority.
2. **Read the trigger** to know what kind of problem it is — a vibration point, a temperature, a failed inspection check.
3. **Click the asset** to open it. There you get the full picture from Working with assets, Measurements, trends, and alarms, and Inspections and routes: the points that are off (the Points section counts them **in alarm**), their week of trend, the **Activity** history with its *fault record* links, and any open non-conformities pinned at the top.
4. **Act, and let the evidence close it** (next section). Leave a **note** on the asset so the next person knows what's been done.

> TIP
> **Check** items aren't urgent, but don't ignore them. A Check is Mechbase telling you a soft pattern is forming — the cheapest time to look is before it becomes a Warning.

How an item clears

Items on this list reflect reality, not a separate to-do list you tick off — so you clear them by fixing the underlying condition, and they come off the list on their own. There are two cases:

- **Sensor-driven items** (vibration, temperature, and the like) clear **automatically** once readings return to normal. Thanks to the averaging logic from Measurements, trends, and alarms, the state lifts when the *pattern* recovers, not on the first good reading — so a cleared item really means the asset has settled.
- **Inspection-driven issues** (a failed check — a **non-conformity**) are closed by hand. Open the affected asset, find the item pinned at the top of **Activity**, deal with the real-world problem, and click **Resolve**. Add a **Resolution comment** if it helps (*"Topped up with Shell Rimula R4, no leak found"*) and click **Mark resolved**.

Either way, the moment the condition is resolved, the asset drops off Needs attention, the counts tick down, and the change is written to the **activity** feed (*"returned to nominal"*, *"… resolved by …"*) — so there's a dated record of both the problem and its resolution.

Where it shows up

You don't have to go looking for trouble — Mechbase surfaces it wherever you're already working:

- The **Needs attention** panel on the home page (*"N active · M silent"*), the full list, and the count badge on the menu.

- The **warning** and **critical** counts on each section card on the home page.
- The state and **in alarm** count on each **asset** page.
- **Dashboards**, where an alarmed point's row is tinted and its section shows the alarm word.
- The **activity** feed, as each state change happens.
- **Your inbox**, if you've asked for it (next section).

Getting alerts by email

Each person decides which alerts reach their inbox, on **My Profile → Email notifications** (Getting started):

- **Alert level** — be emailed when an asset reaches **Check**, **Warning**, or only **Critical**. The subject says it plainly: *"Atlas Copco GA22 is now Warning · LPG Terminal"*, with the point and reason inside. When it recovers, everyone who was told about the problem gets *"… is back to normal"*.
- **Status changes** — also email when a non-conformity is resolved.
- **Route reports** — a summary when a route finishes with open non-conformities (*"Monthly Safety & Power Check: 1 non-conformity · LPG Terminal"*).

Every member of the site gets these by default at *Check and above*; anyone can turn them down, pause them, or unsubscribe from the link in any email.

7 · Mechbase on your phone (iOS & Android)

The **iOS** and **Android** apps are purpose-built field tools: they let your crew walk routes, take readings, photograph equipment, and capture **thermal images and vibration signatures** right at the machine — fully offline if there's no signal — and sync it all back to Mechbase automatically. This chapter explains what the apps are and how a technician uses one on a round.

What the apps are

The apps are **fully native** — built with each platform's own toolkit (SwiftUI on iPhone and iPad, Material 3 on Android) rather than a web page in a wrapper. They share your Mechbase account and the same plant data — the installations, sections, zones, assets, measurement points, and routes from the earlier chapters — but they are focused on the work that actually happens on the floor:

- **Walking inspection routes** and recording each check.
- **Capturing readings on the spot** — by hand, and from **thermal** and **vibration** sensors connected to your phone.
- **Photographing equipment** — asset photos and each point's reference photo.
- **Working offline**, then syncing when you're back in range.

Think of the web as where you set things up and analyse trends, and the apps as where the readings come *from*. Anything captured on a phone — a value, a photo, a thermal hotspot, a vibration signature — flows into the same measurement points and trends you read on the web (Working with assets and Measurements, trends, and alarms).

Installing and signing in

Install the Mechbase app on the phone or tablet you'll use in the field (your administrator will tell you where to get it for your organisation), then sign in with the same account you use on the web: type your email in the **Username** field, and your password. There's no separate mobile account — your site access comes with you. After signing in you land on the home hub.

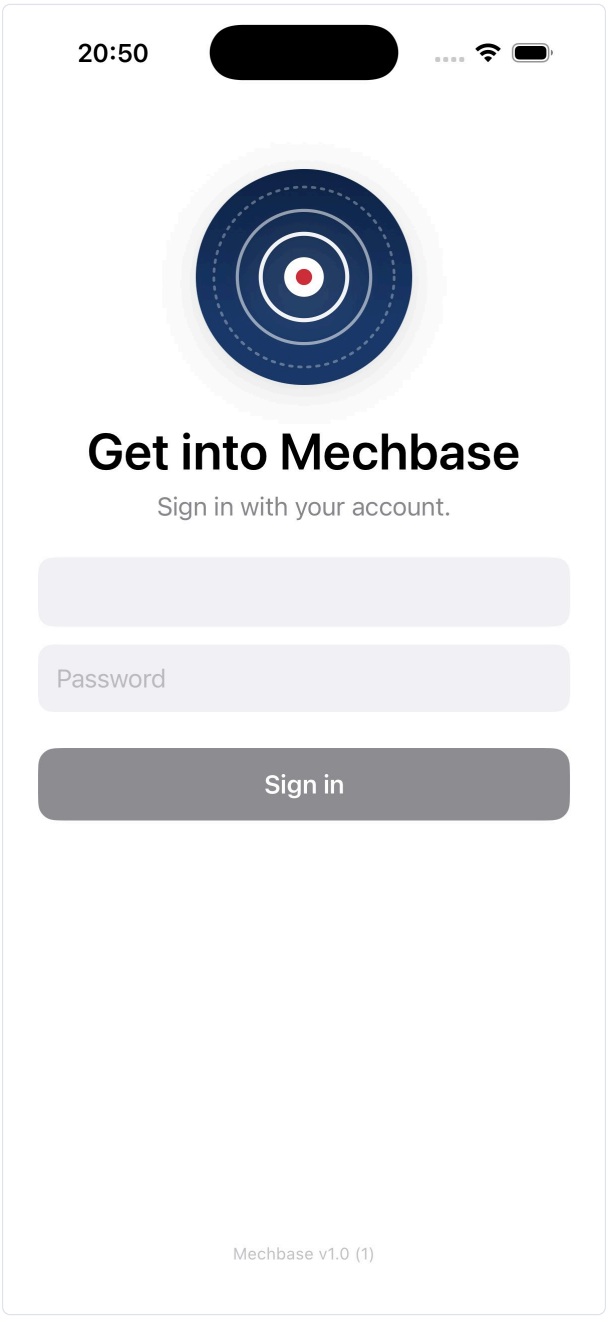


Figure 41 · The sign-in screen (iOS)

NOTE

Microsoft single sign-on is web-only for now; on the phone, sign in with your Mechbase email and password.

The home hub

The home hub is your base on the phone.

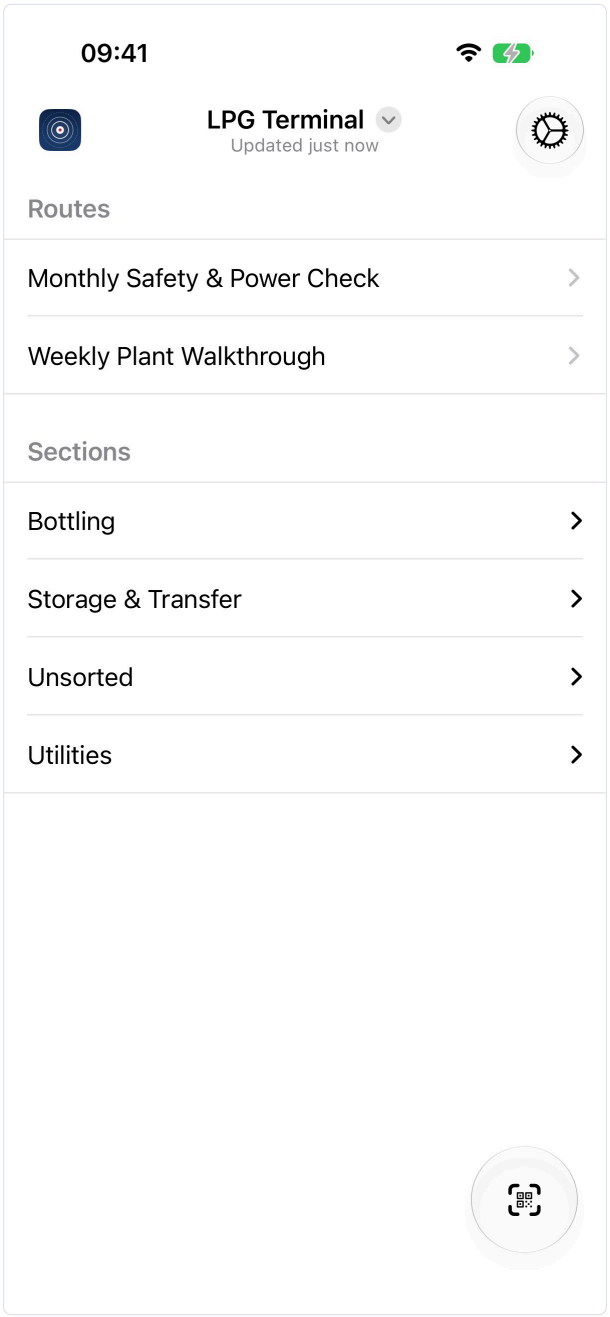


Figure 42 · The home hub — installation, routes, and sections (iOS)

- **The site you're in** is the title at the top. Tap it to **switch installations**. Underneath, a status line tells you how fresh your data is — *"Updated just now"*, *"Updated 11 min ago"*, or *"Updating…"*.
- **Routes** — your inspection routes, ready to walk. A dot marks a route with a run in progress.
- **Sections** — browse the plant: **Sections → Zones → Assets**. Tap an asset to open it (next section).
- **Scan QR** — the round button at the bottom right opens the scanner (see Finding things by QR).
- **Settings** — the gear at the top right.
- **Pull down to refresh** — force a sync whenever you want the latest.

Assets and points

Tap an asset in the tree to open its screen: a breadcrumb (section › zone › equipment type), the asset's **photos**, a **Share link** button that sends the asset's web address to a colleague, and its **Measurement Points**.

- **Add Photo** takes a picture with the camera (**Take Photo**) or picks one from the library (**Choose from Library**). Photos taken offline are marked *Waiting to upload* and go up on the next sync.
- Tap a measurement point to take a one-off reading there, outside any route. The point screen shows its **last** value, a **Recent** list of readings, and its reference photo — tap the photo button to add or replace it.

Walking a route

This is the apps' main job. Open a route to see its **checklist**, grouped by stop, with a count of how many checks are complete.

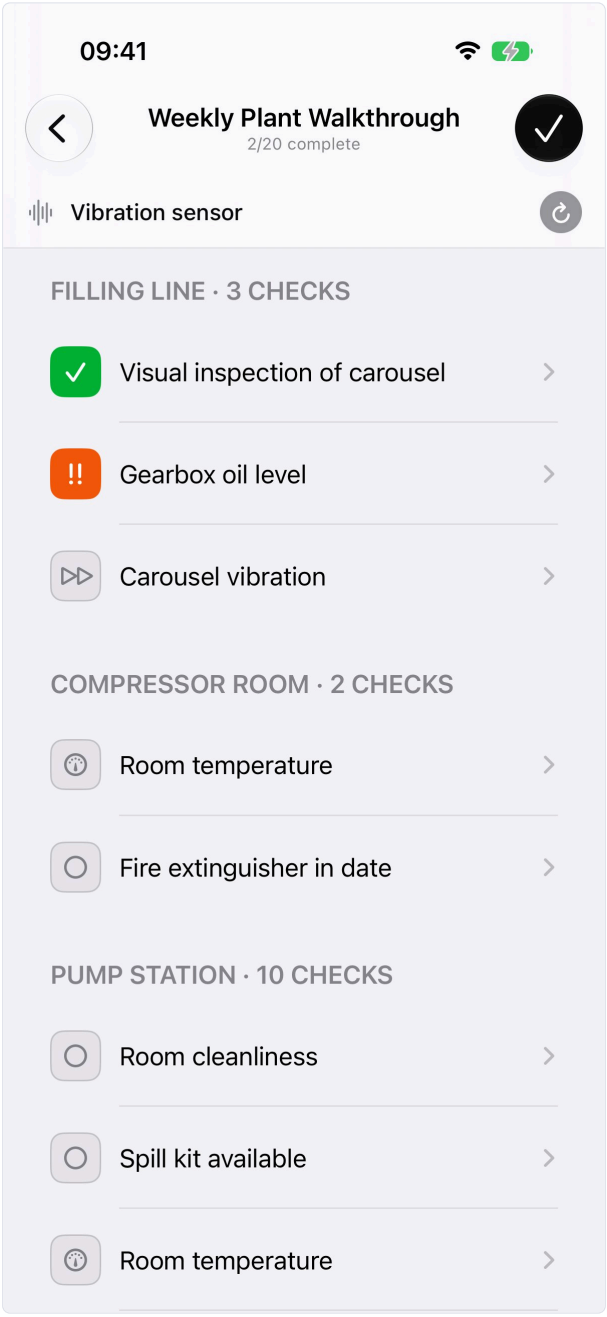


Figure 43 · Walking a route — the checklist (iOS)

Each check's icon tells you its type and, once answered, its result by shape — a tick for a pass, `!` for a minor and `!!` for a major failure, a skip mark for a check you passed over. Tap a check to open it. You

see where you are (*Point 2 / 20*), a progress strip across the top (tap it to jump), the breadcrumb, and the right input for the check's **type**:

- **Pass / Fail** — tap the tick to pass: it records and moves on in one tap. Tap the cross to fail, then grade it **Minor** (`!`) or **Major** (`!!`), which maps to a Warning or Critical non-conformity.
- **Numerical** — type the value you read off the gauge, in large type. If the check has limits, a bar under the value shows where it falls, with the verdict **OK**, **Watch**, or **Alarm**. The **last** reading is shown as a sanity check, and there's a minus key for negative values.
- **Multi-check** — several pass/fail steps bundled into one check (*Checks 2 / 5*), each passed or failed on its own.
- **Measurement** — a reading on the asset's measurement point, which flows into the point's trend. For vibration and thermal points you capture it from a sensor (next section).

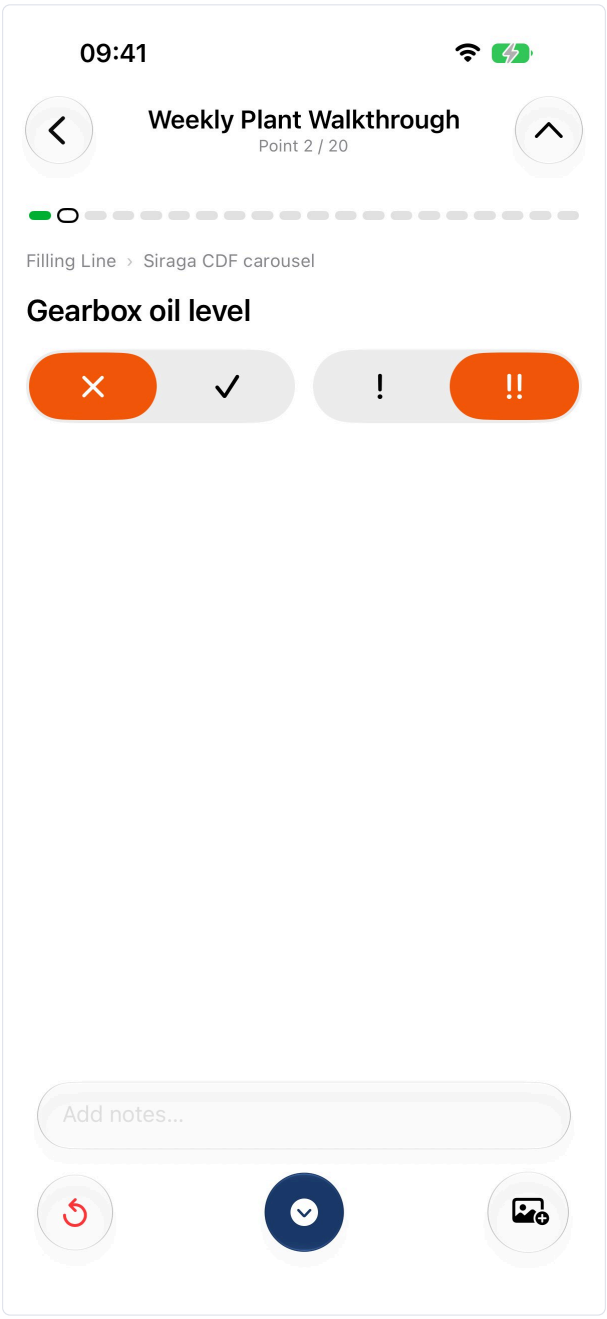


Figure 44 · A failed check graded Major (iOS)

Every check also has **Add notes...** and a photo button. When a point has **Instructions** set on the web, they're shown on the check — open the disclosure to read them.

The bottom bar moves you on: the centre button records the check and goes to the next (or **Skip**s it if you haven't answered), the arrow at the top goes back, and **Clear** wipes an answer — with an **Undo** for a few seconds. Scanning an asset's QR tag while you're in a route jumps straight to its check.

Tap the ✔ at the top right of the checklist to complete the run. On iPhone a **Route completed** bar with **Undo** appears on the home hub for a moment; on Android you're asked to confirm if checks are still to do. You don't have to answer everything — skipped checks are simply left unanswered, and can be filled in later on the web with **Amend** (Inspections and routes).

> **NOTE**
> A route has only one run in progress at a time. If a colleague already started today's round, opening the route **joins their run** (*"Joined a run already in progress on another device"*), so you both add to the same report.

Capturing readings with sensors

The biggest thing the apps do that the web can't is **capture live sensor data** at the machine. A banner at the top of the route (or point) shows the sensor's state — *Connecting…*, connected, *Sensor lost* — with **Connect** / **Reconnect**, and an ⓘ that opens step-by-step instructions for waking and linking it. The **Capture** button stays disabled until the sensor is connected.

Thermography — with a FLIR thermal camera. For a thermal point, the app connects to a **FLIR ONE Edge** camera and shows a **live thermal preview** with a temperature scale down the side. **Options** lets you set the **Color** palette (*Iron* by default), the **Emissivity** (*Matte*, *Semi-matte*, *Semi-glossy*, *Glossy*), and run a **NUC** recalibration. Capture a frame and the app **automatically finds the hotspots** — tap to add one, or tap a marker to edit it and grade it **Minor** or **Major**. **Retake** or **Save**; the thermal image and its hotspots attach to the reading, and a copy is saved to your photo library. A route check can hold several thermal frames.

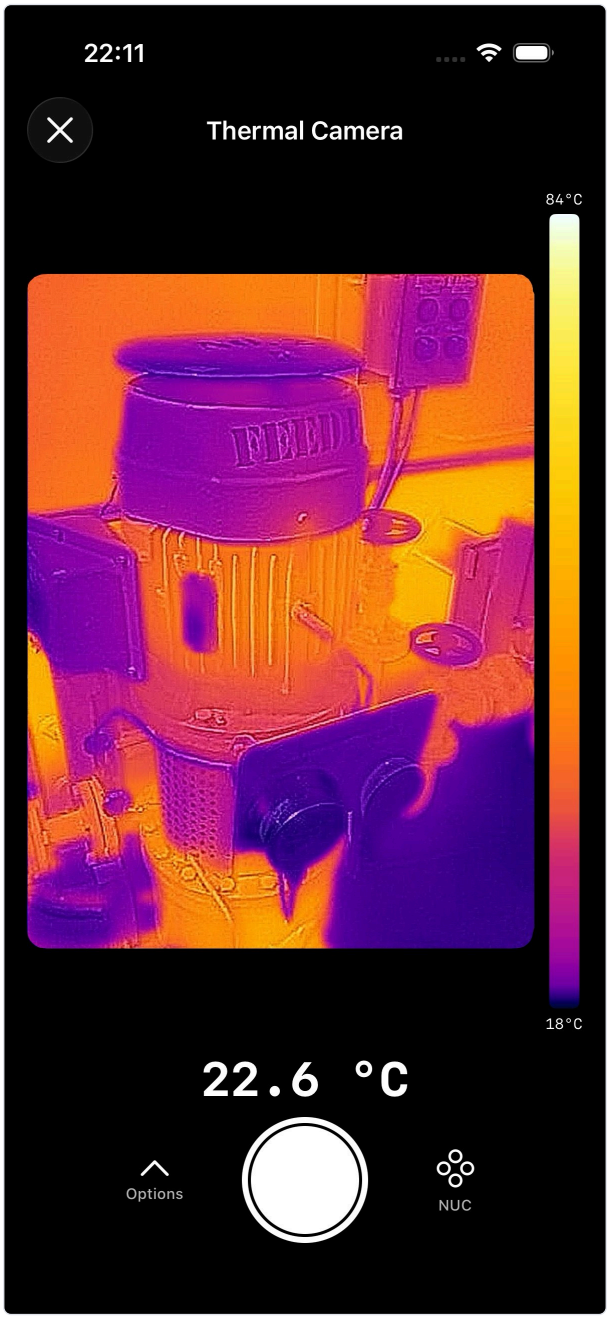


Figure 45 · Live thermal preview from a FLIR camera, with palette and temperature scale (iOS)

Once linked, the camera stays connected while you move between checks, so you don't wait at every stop. Pulling down to refresh on the home hub drops the camera's Wi-Fi link so the phone can reach the internet to sync.

Vibration — with a Bluetooth sensor. For a vibration point, the app pairs with a **Bluetooth accelerometer** (the TE 8531N wireless sensor). Pick the **X**, **Y**, or **Z** axis and capture: the sensor records about one second of waveform, and the app shows the **RMS**, **Peak**, **Crest**, and **Vel 10Hz** values together with the **Waveform** and **FFT magnitude** plots. **Retake** or **Save**; the full signature is attached to the reading — the same waveform an analyst opens on the web (Measurements, trends, and alarms).

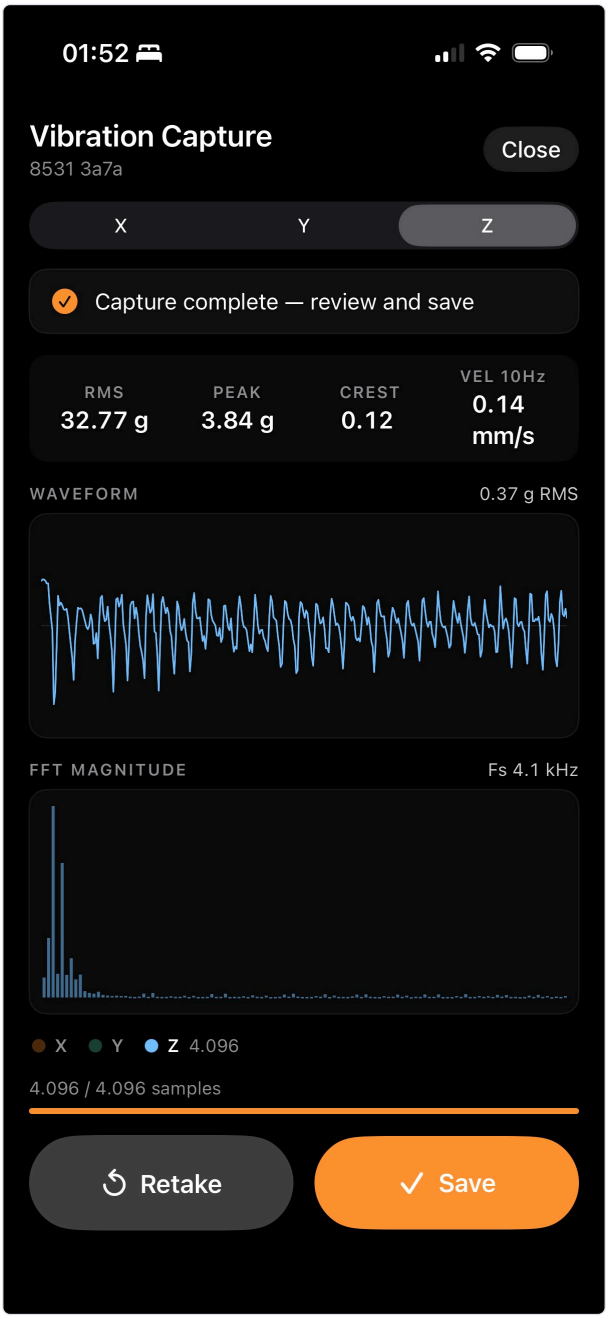


Figure 46 · Vibration capture: RMS, peak, crest and velocity above the live waveform and spectrum (iOS)

> **TIP**
> Wake the vibration sensor with a quick tap of the magnet (it stays awake for about an hour), and keep the phone close. On iPhone the thermal camera starts linking as soon as you open a route that needs it; tap **Connect** on the vibration banner when you reach the first vibration check. Android starts both, if you've already granted Bluetooth permission.

Finding things by QR

Rather than drilling through the tree, you can **scan a QR code** placed on a piece of equipment. Tap the scanner on the home hub and point it at the tag: the app jumps straight to that **asset** or **measurement point** — or, if you're in the middle of a route that includes it, to that check.

If the code belongs to a different installation, the app offers to **Switch** to it; if it isn't a Mechbase code, it tells you so (*"Not a Mechbase code"*) rather than guessing.

Working offline

Plant rooms and remote sites don't always have a signal, so the apps keep a **local copy** of your data and let you work without a connection:

- Open routes, assets, and points, and **walk a whole route offline**.
- Record results, readings, photos, thermal captures, and vibration signatures — all saved on the device first. A value you're typing even survives the app being closed.
- Everything **syncs back to Mechbase automatically** once you're online again, in the background. The home hub's status line tells you where things stand; pulling down forces a sync.

> **HEADS-UP**
> Signing out deletes anything on the device that hasn't uploaded yet. The app warns you first (*"You have N unsynced changes…"*) — on a shared field phone or tablet, let it sync before you sign out.

Your vibration sensor

Sensors are managed in the app's **Settings** (the gear on the home hub), under **Data Acquisition → Vibration Sensor**.

- You don't have to pair anything in advance: with no sensor saved, the app **connects automatically** to the first TE 8531N it finds.
- To fix a particular sensor to the phone, open **Vibration Sensor**, tap **Scan**, and pick it from **Available sensors**. Saved sensors can be removed with **Forget**. If nothing appears, wake the sensor with a magnet tap and keep the phone close.
- The **FLIR thermal camera** needs no setup in Settings — it connects when you open a thermal capture.

Settings also holds your **Account**, with **Sign Out**.

> **NOTE**
> For mounting, wiring, ranges, and battery life of the wireless sensors, follow the sensor's own user manual — it's the authority on the hardware itself.

iPhone and Android

The two apps are built to match — the same screens, the same route and capture workflow, the same offline behaviour, and the same FLIR and Bluetooth sensor support. A few details differ between them:

- **iPhone only:** the **Route completed** bar with **Undo**, and a minus key for negative values on numeric checks.
- **Android only:** a **Complete route?** confirmation when checks are still to do, a flashlight in the QR scanner, **Undo** when you remove a photo, and a *Pending upload* marker on routes waiting to sync.
- Each follows its own platform's navigation, styling, and permission prompts.

Here's a route checklist on **Android**:

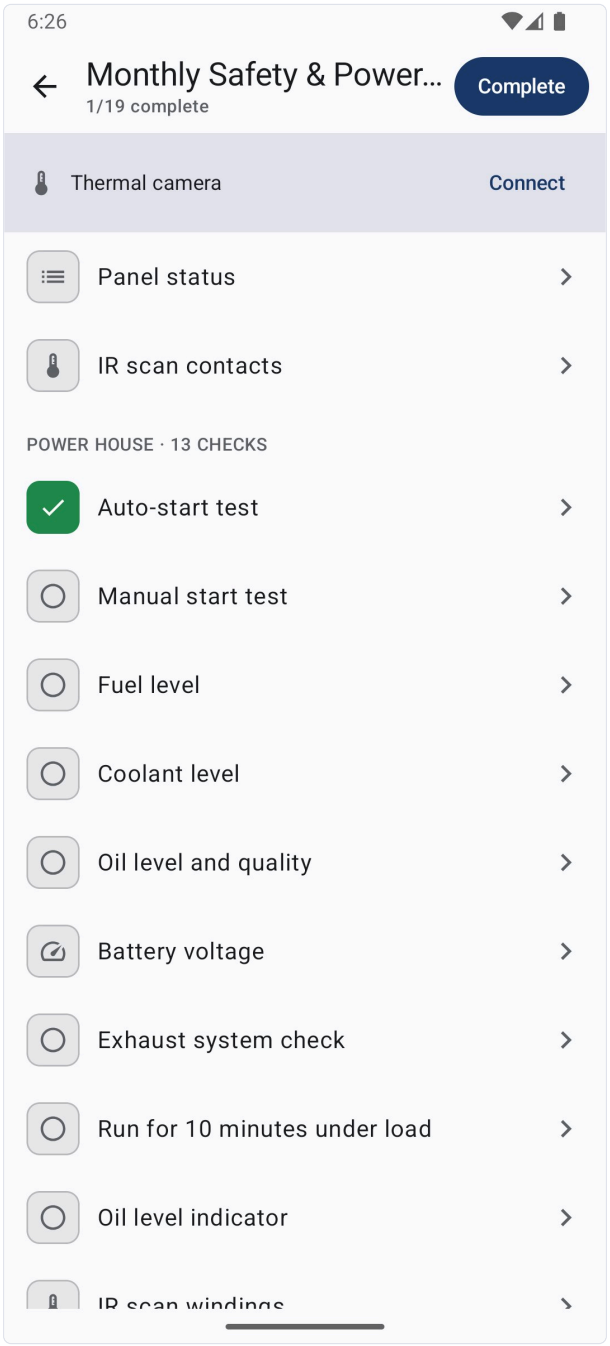


Figure 47 · A route checklist on Android

Anything you learn on one transfers directly to the other, and the data is identical to the web.

8 · MaintNode devices and automatic data

Routes cover the readings a person takes. **MaintNode** covers the rest — the hardware that measures your equipment continuously and sends the readings into Mechbase on its own, with nobody in the loop. This chapter explains what MaintNode is, how you claim a device and connect its inputs to measurement points, and how to tell it's healthy.

What MaintNode is

A **MaintNode** is a small edge computer installed at your site, wired to the sensors on your equipment. It measures on a schedule, keeps every reading on its own storage, and uploads them to Mechbase over a secure internet connection — so a network outage never loses data; the readings simply catch up when the link returns. Those readings land on the matching **measurement points**, where they trend, get checked against alarm limits, and drive asset health exactly like any other reading.

A MaintNode reads three kinds of input:

- **Built-in inputs** — sixteen analog channels (two stacked boards of eight) for accelerometers, 0–10 V and 4–20 mA transmitters, and tachometers.
- **USB DAQ** — an external USB-201 acquisition module.
- **Modbus** — registers and coils read from a PLC, drive, or gateway over **TCP** or **serial (RTU)**.

Three ideas make the rest of this chapter make sense:

- **A device has channels.** Each input is a **channel**, numbered from 0.
- **Channels are grouped.** An **acquisition group** is a set of channels on one device that are measured together, on the same hardware and the same schedule — for example *"Compressor vibration: built-in inputs, every 20 minutes, 4096 samples at 4096 Hz"*.
- **A point is fed by a channel.** You connect the two on the point's **Sources** (Working with assets) — *this point is fed by channel 0 of that group on that device*.

The MaintNode Devices page

Open **MaintNode Devices** from the menu, under **Setup**. It lists every device claimed by the current installation.

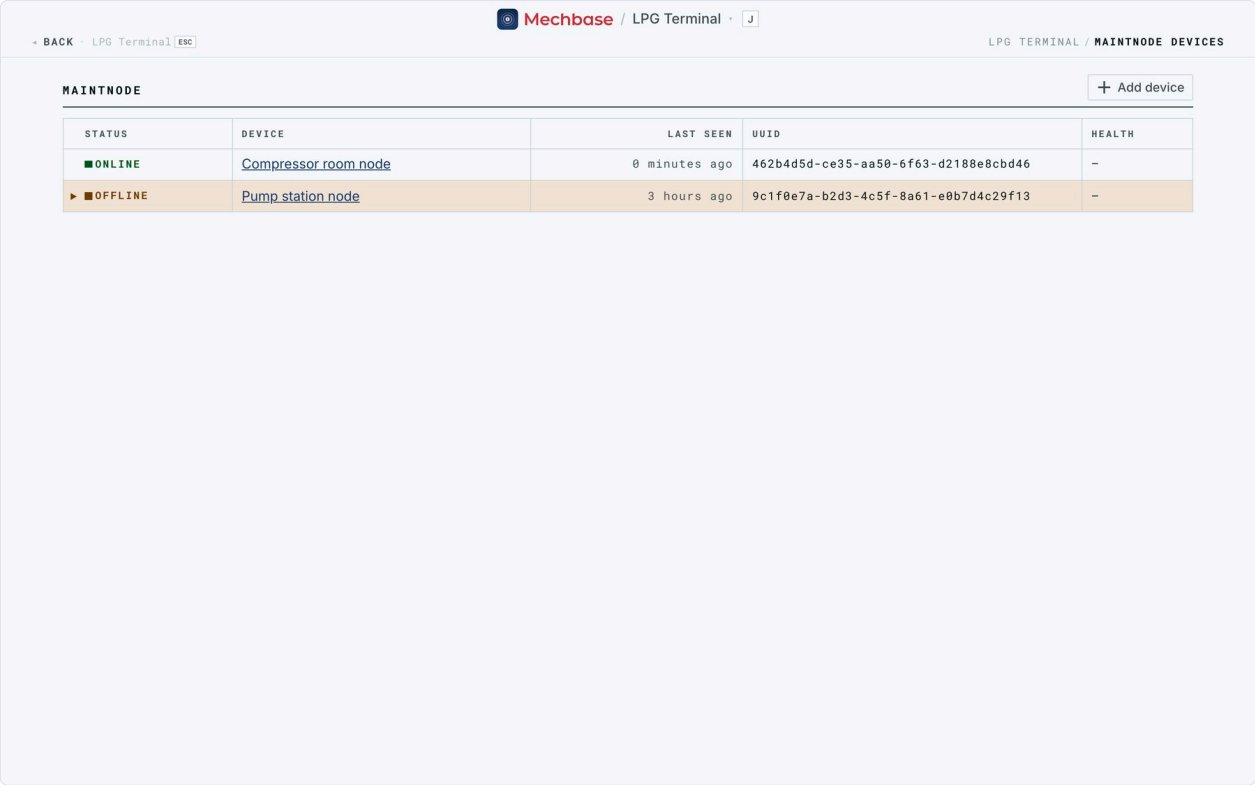


Figure 48 · The MaintNode Devices page

For each device you see:

- **Status** — **Online** if it has checked in within the last five minutes, otherwise **Offline**.
- **Device** — its name; click it to open the device page.
- **Last seen** — when it last checked in.
- **UUID** — the device's unique identity.
- **Health** — a warning chip if the device reports high **CPU**, memory (**MEM**), **DISK** usage, or **TEMP**erature.

A site with no devices yet shows *"No devices linked"* and an **Add device** button.

Claiming a device

Every MaintNode carries a **label** with a QR code, its short ID, and the credentials for its setup hotspot. **Claiming** registers the device to your installation; Mechbase then sends it the credentials it needs to upload readings.

The quickest way is from the device itself: **scan the QR code** on its label with your phone. It opens the device's **Device setup** page in Mechbase (sign in if asked).

From a desk instead, open **MaintNode Devices**, click **Add device**, type the device's UUID (it's on the label), and click **Go to device**.

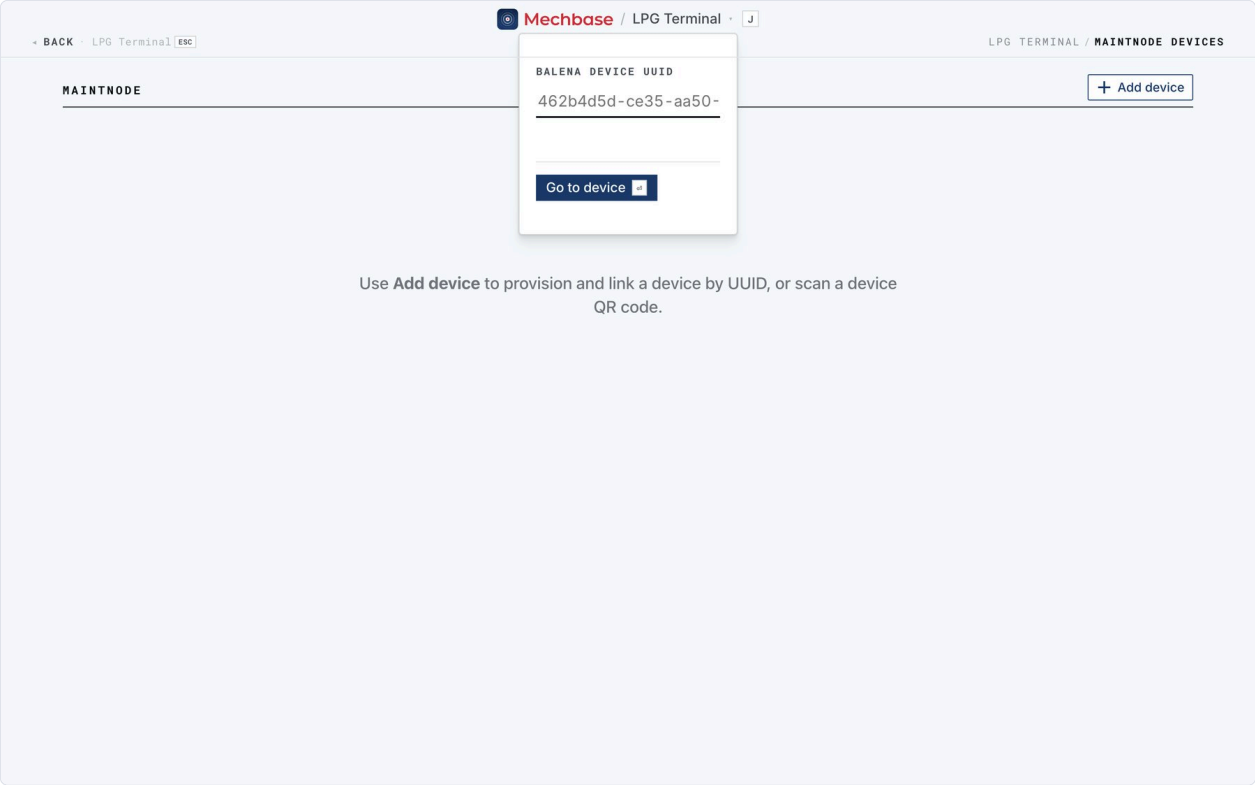


Figure 49 · Adding a device by its UUID

On the **Device setup** page — *"Choose where this device belongs"*:

1. Pick the **Installation** (if you belong to more than one).
2. Give it a **Device name** people will recognise — *"Compressor room node"*.
3. Click **Claim device**.

You're returned to the devices list with *"Compressor room node is now registered to LPG Terminal"*. Once claimed, the same page shows the device's **Online** / **Offline** status, lets you **Rename** it, and has **Send credentials again** — use that if a newly claimed device doesn't come online.

If a device is already registered to another installation, Mechbase says so and shows nothing more about it. A device it doesn't recognise at all reads *"This device isn't registered with Arpedon yet"* — contact Arpedon.

> **NOTE**
> Any member of the site can claim and connect devices, but in practice it's a commissioning job. Day-to-day users don't need this page — they just see the readings it produces.

Getting it on the network

A MaintNode needs a network connection to upload. If it can't reach one on its own (no DHCP on its Ethernet port, or Wi-Fi only), use its built-in **Network setup** page:

1. On your phone or laptop, join the device's Wi-Fi hotspot — **MaintNode-xxxxxxxx**, with the password printed on the label.
2. Browse to **http://10.42.0.1**.
3. Set up **Ethernet** or join a **WiFi** network, review the **Change**, and **Apply** it. Changes that would cut you off can be staged for the next reboot instead; the **History** tab shows what was changed and when.

Once it's online and claimed, the device's status in Mechbase turns **Online**.

Connecting a channel to a point

With the device claimed and wired, connect each channel to the measurement point it measures. Open the point, click **Edit** (E), scroll to **Sources**, and click **Add source**.

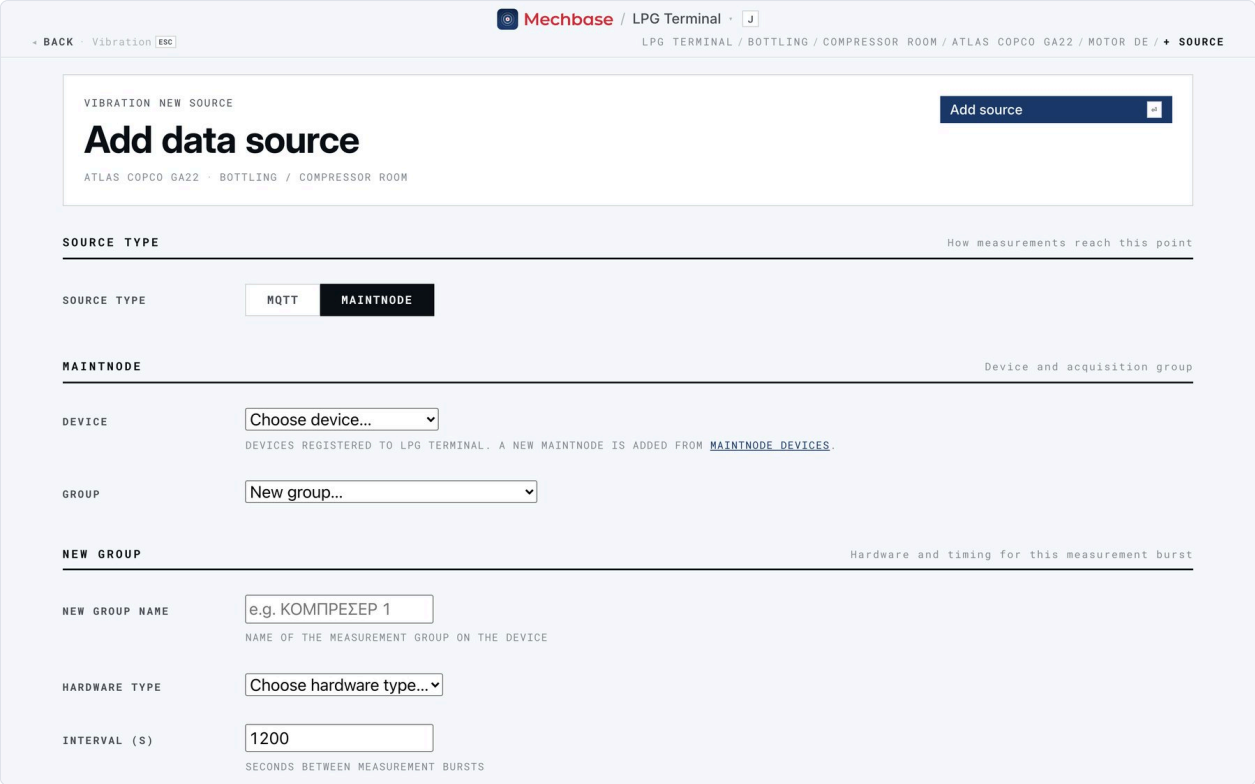


Figure 50 · Adding a MaintNode source to a point

1. **Source type** — choose **MaintNode** (MQTT is for your own devices; see the MQTT cookbook).
2. **Device** — one of the devices claimed by this installation.
3. **Group** — an existing acquisition group on that device, or **New group…**. A new group needs:
   - a **New group name**;
   - the **Hardware type** — **Built-in inputs**, **USB DAQ (USB-201)** (with its hardware ID and address), or **Modbus**;
   - the **Interval (s)** between measurement bursts (20 minutes by default), the **Sample rate (Hz)**, and the **Samples per burst**.
4. **Channel** — channel 0–15 on built-in inputs (0–7 is the first board, 8–15 the second).
5. **Sensor type**, and its settings:
   - **Accelerometer** — **Sensitivity (mV/g)** (100 by default) and **Offset**.
   - **0-10 V**, **4-20 mA**, **0-20 mA** — the **Min value** and **Max value** the signal range represents, and an optional **Aggregation** (mean, RMS, max, min) over the burst.
   - **Tachometer (RPM)** — a once-per-revolution pulse.
   - For Modbus: **Register value** or **Digital**.
6. Click **Add source**.

Modbus groups also need the connection — **TCP** (the **IP address** and **TCP port** of the PLC or gateway, as the MaintNode sees it on the plant network) or **Serial (RTU)** (port, baud rate, parity, stop and data bits) — and the **Unit ID**. Each Modbus source then names its **Register address**, **Register type** (holding, input, coil, or discrete input), **Data type** (16- or 32-bit integer, or float), **Word order**, and an optional **Scale** and **Offset**. Digital channels can be **normally open** or **normally closed**, and are recorded as on/off periods — ideal for a run signal or a fault contact on a **State** point.

Why a tachometer is worth wiring. Put a tachometer in the same group as the accelerometers on a machine and every vibration reading carries the shaft's once-per-turn pulses. Mechbase then mea-

sures the actual shaft speed and the **1X** and **2X** amplitude and phase of each reading (Measurements, trends, and alarms), and the tachometer becomes the authority on whether the machine is running.

To change a source later, click its pencil on the point's edit page. A source keeps its type, device, and group — to move it, remove it and add a new one. **Edit group settings** changes the group's name and timing for every channel in it. **Remove source** stops new readings arriving from it; readings already recorded stay on the point.

From device to trend

Once a device is claimed, online, and its channels connected, its data reaches Mechbase like this:

1. On each group's schedule, the MaintNode measures its channels in a burst.
2. It stores the readings on the device, then **uploads** them to Mechbase — immediately when it's online, or as soon as it's back if not.
3. Each reading lands on the **measurement point** that channel is connected to.
4. From there it's an ordinary reading: it joins the **trend**, is scored against the point's **alarm rule**, and updates the asset's **state** (Measurements, trends, and alarms).

So the join between hardware and Mechbase is one line on a point: *this point is fed by that device's channel*. After that, everything you learned about trends and alarms applies automatically.

Is my device healthy?

Because the readings only keep coming while the device is working, it's worth knowing how to check one:

- On **MaintNode Devices**, a healthy device reads **Online**, with a recent **Last seen** and no **Health** warnings.
- Mechbase watches every connected point. If a point that's normally fed automatically misses three scheduled readings in a row, Mechbase flags it: the point page shows **No data** and since when, the asset's activity records *"No readings from … since …"*, and the asset shows up as **Silent** on Needs attention. When readings come back, the activity says *"Readings resumed"*.
- If a device goes **Offline**, check its power and network first. If it's online but a single point is silent, check the sensor and its wiring, and that the point's **Sources** still names the right channel.

> **NOTE**
> For the wiring, mounting, and sensor specifics, follow the hardware's own user manual — it's the authority on sensor models, ranges, and installation.

9 · Setting up and managing your account

Set Mechbase up so every person sees exactly the sites they're responsible for — and nothing else. This chapter is for owners and administrators: it covers building out a site (including importing it from a spreadsheet), managing installations, and the part unique to admins — managing people, access, and single sign-on.

Two kinds of setup

It helps to know that Mechbase administration happens in two places:

- **In the app** — building out the *content* of a site: its sections, zones, assets, measurement points, alarm overrides, routes, bearings, and devices. Any member of the site can do this through the normal Mechbase screens you've already seen, and **staff** can also build dashboards.
- **In the back office** — managing *people and installations*: user accounts, who belongs to which site, who is staff, single sign-on, and installation-wide alarm defaults. This is done by a system administrator in **Django administration** (`/admin/`), a separate, plainer admin interface behind the main app. For most customers, that's Arpedon.

Building out a site

Everything that makes up a site is created and edited in the app, top-down:

- **Sections** — from the menu, **Sections → Add a section**.
- **Zones** — on a section page, **Add Zone** (`Z`), or import them in bulk (next section).
- **Assets** — the **Add asset** row in a zone: give it a name, then open it and use **Edit** to set its **Equipment type**, **ISO 10816** class, and machine body (Working with assets).

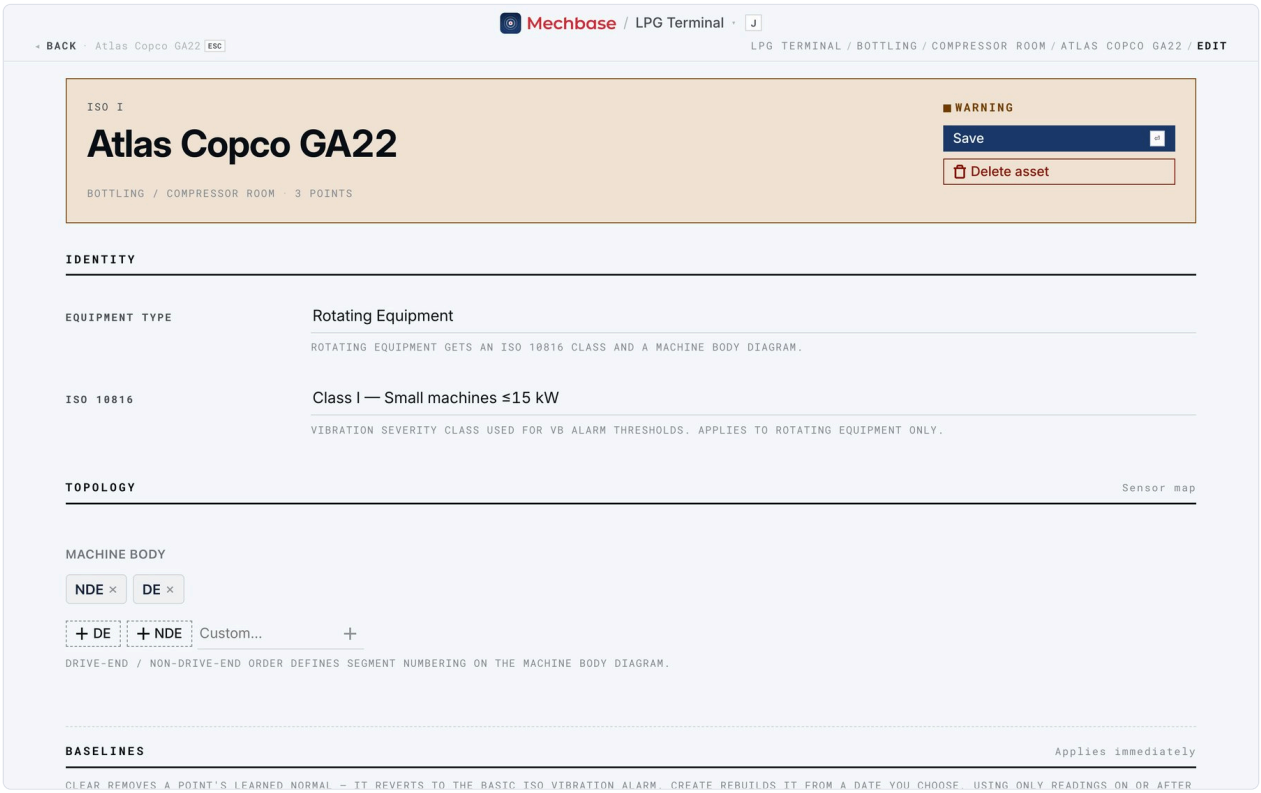


Figure 51 · Editing an asset's identity and topology

- **Measurement points** — on an asset, **Point** (`P`), then **Edit** to set the location, sensor, bearing, instructions, photo, and data source (Working with assets).
- **Bearings** — **Setup → Bearings**, if a bearing you use isn't in the library (Measurements, trends, and alarms).
- **Devices** — **Setup → MaintNode Devices**, to claim hardware and connect it to points (MaintNode devices and automatic data).
- **Alarm overrides** — per point, where the defaults don't fit (Measurements, trends, and alarms).
- **Routes** — **New Route** (`N`), then build the checklist from stops and checks (Inspections and routes).
- **Dashboards** — staff only: **Dashboards → New dashboard** (Finding your way around).

In short, the earlier chapters *are* the site-building guide — this section just points to the order to do it in: sections → zones → assets → points → sources → alarm overrides → routes → dashboards.

Importing zones and photos

For a large site — a hotel, a hospital, a ship with hundreds of rooms — typing zones in one by one isn't practical. **Setup → Zone types & imports** brings them in from the spreadsheets you already have.

Import zones. Click **Import zones**, then **Choose room list**, and pick an Excel (`.xlsx`) or CSV file. Mechbase looks for a header row with a **room** column (*Room number*, *New room number*) and a **Building**, **Level**, or **Section** column; optionally a **Room type** or **Room subtype** column for the zone type, and a **Room number as is** or **Alias** column for an old number. Greek headings (*Δωμάτιο*, *Κτίριο*, *Όροφος*…) work too. Sections are named *Building L*.

Before anything is saved you see a **preview**: how many zones are new, will be updated, or are unchanged, which sections and zone types will be created, and any rows that were skipped and why. Click **Import N zones** to go ahead. The import only ever adds zones and sets their type and alias — it never deletes or moves anything — so if a column was wrong, fix the file and import again.

Zone types. Each zone type from the import is listed on **Zone types**. Open one to add **Shared documents** — drawings, circuit schedules, datasheets — that then appear on every zone of that type.

Import photos. Click **Import photos**, zip a folder of site photos with one sub-folder per room (for example `ΔΩMATIO 1321/PHOTO-2026-10-30-08-50-39.jpg`), and click **Choose zip and import**. Each photo goes to the zone named by its folder (matched on the zone name, then its alias), dated from the file name or the photo's own data. Re-uploading the same folder later only adds the new photos; anything that isn't an image, or doesn't match a zone, is listed as skipped.

Managing your installations

An **installation** is a whole site. The sites a user can open are listed on their installation picker (Getting started) and under **Installations** in the menu.

Creating a new installation, and setting its top-level details, is done in Django administration under **Installation → Installations**. Each installation gets a number automatically (it appears in the web address, like `/2012/`).

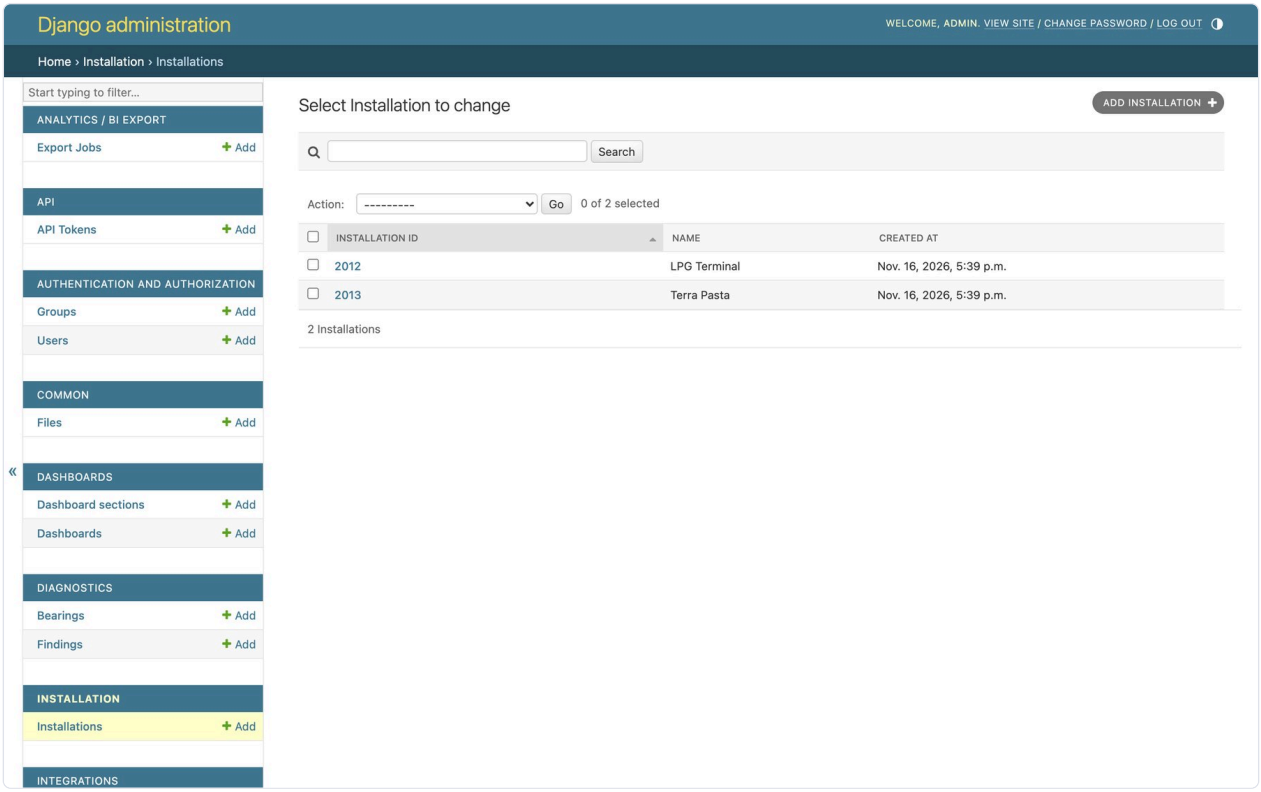


Figure 52 · The installations list in Django administration

Open an installation to change its **Name**, its **Owner**, and its **Users** — the people who can open it. Once a site exists and you belong to it, you build it out using the in-app tools above.

People and access

Access in Mechbase is deliberately simple:

- **Membership.** A user can open an installation if they're in its **Users** list (or are its **Owner**). Inside it, they can see and do everything.
- **Staff.** A user marked as **staff** can also build and edit dashboards, amend other people's inspection runs, delete equipment that already has recorded history, and sign in to Django administration.

That's the whole model — there are no per-feature roles to configure.

Adding a person

1. Open **Django administration** at `/admin/` (you need an administrator account).

2. Go to **Authentication and Authorization → Users** — the list shows everyone's email, name, whether they're staff and active, and when they last signed in — and click **Add user**.
3. Enter the person's **Email address** — it's their sign-in. Choose whether they use **Password-based authentication**: leave it **Enabled** and set a password, or **Disabled** for someone who will only sign in with Microsoft.
4. Click **Save**, then fill in their **First name** and **Last name**. Tick **Staff status** only for people who need it.

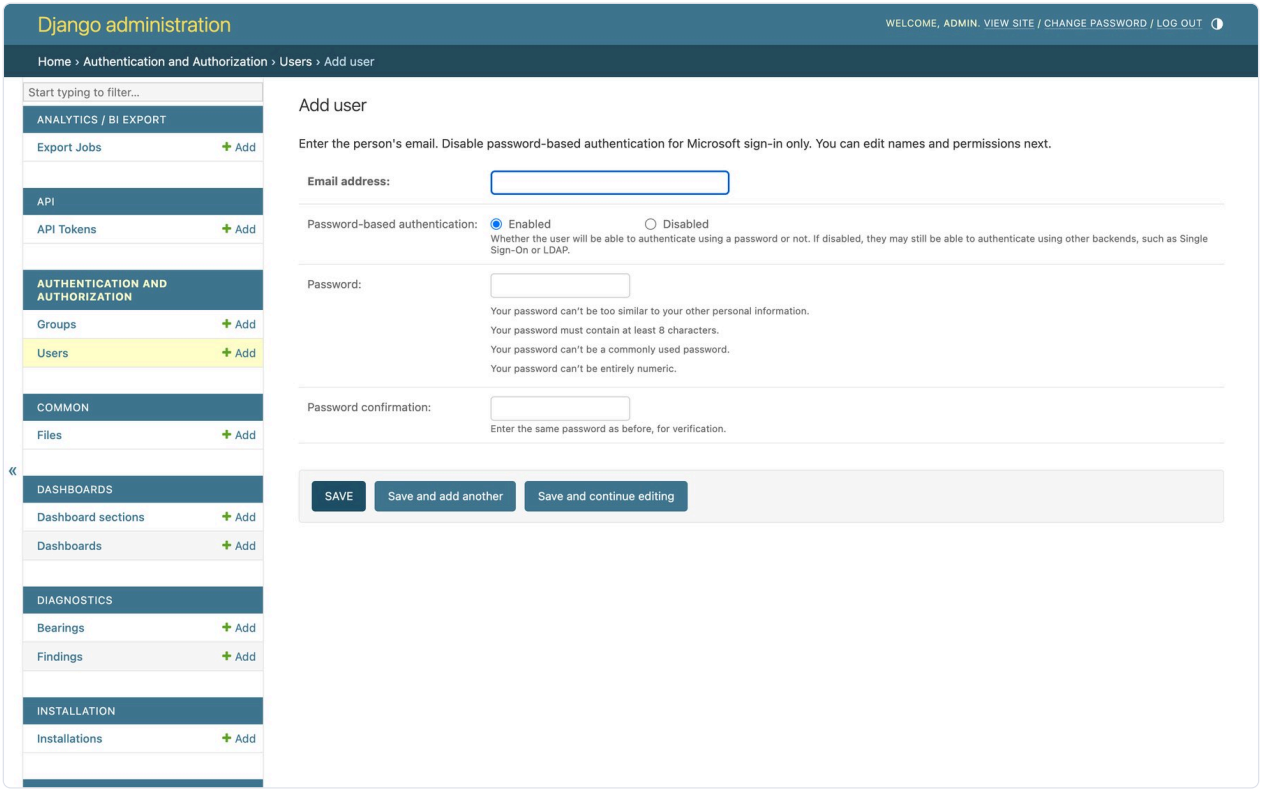


Figure 53 · Adding a user by email in Django administration

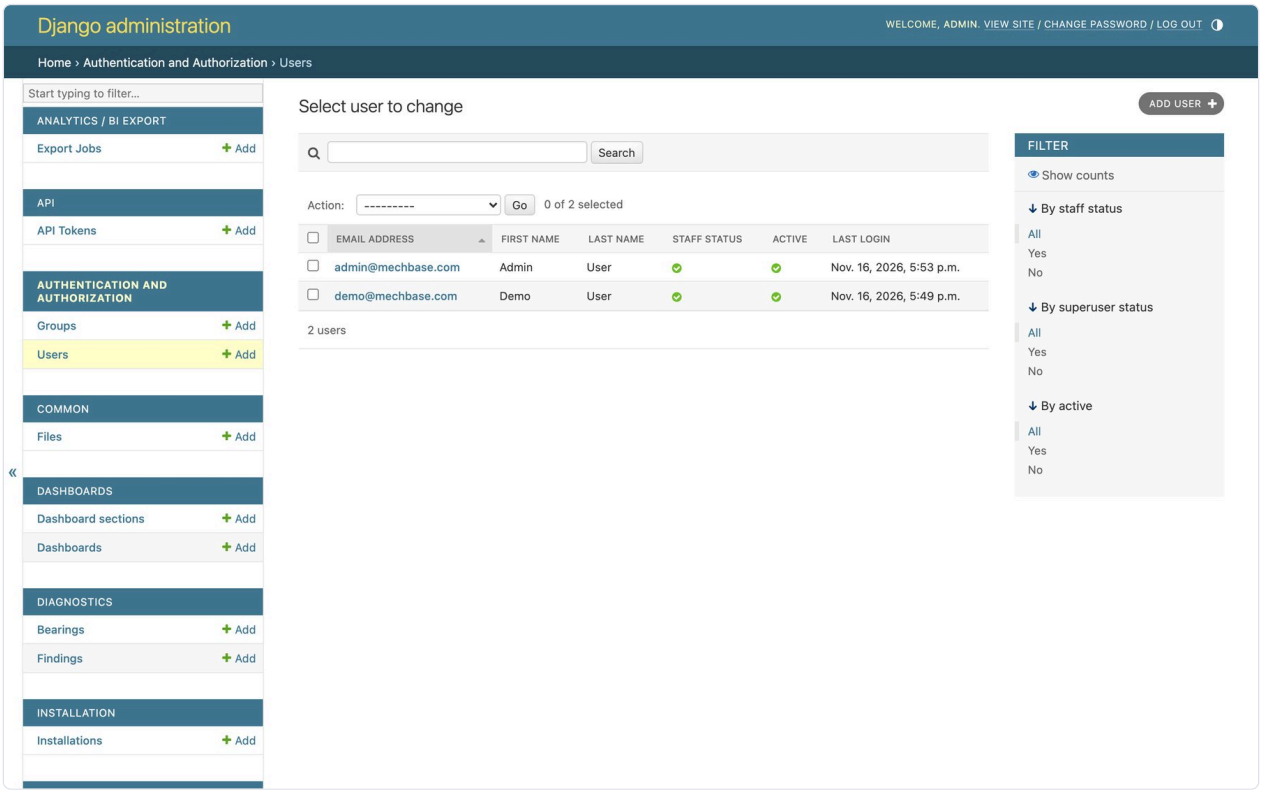


Figure 54 · The Users list in Django administration

Giving them access to a site

Open **Installation → Installations**, open the site, and add the person to **Users**. Click **Save**. Next time they sign in, the site is on their picker.

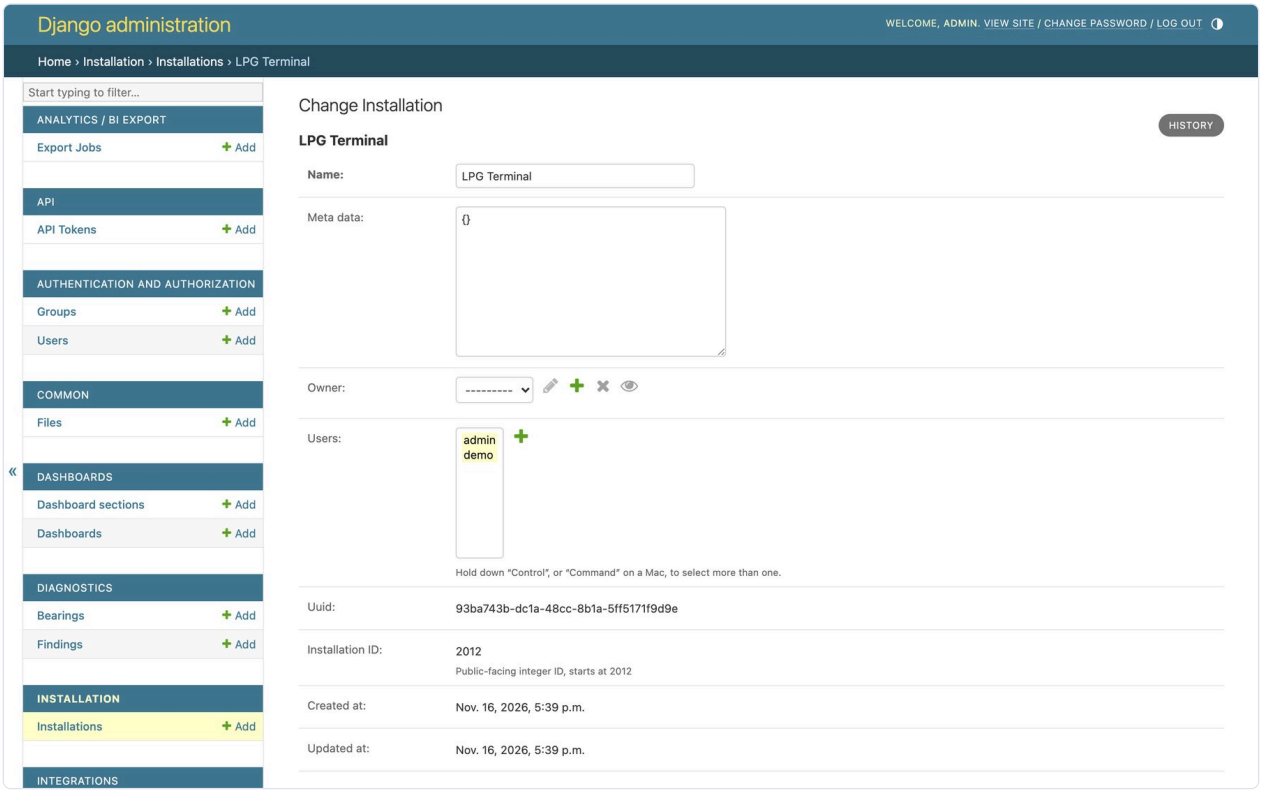


Figure 55 · An installation's Users list in Django administration

To remove someone from a site, take them off its **Users** list. To stop someone signing in at all, untick **Active** on their user account.

HEADS-UP

Membership decides what each person sees across Mechbase, so keep each site's **Users** list to the people who actually work on it — it keeps their view focused on the data that's theirs. Grant **staff** sparingly.

NOTE

Each person manages their own name, email, units, and email notifications on **My Profile** (Getting started). In Django administration, **Notification settings** shows everyone's choices and can send a test email to selected users.

Signing in with Microsoft (single sign-on)

If a customer uses Microsoft 365 (Entra ID), their people can sign in to Mechbase with their work account instead of a separate password. Setting it up takes one entry per customer:

1. In Django administration, open **Single sign-on → Entra tenants** and click **Add**.
2. Enter a **Name**, the customer's **Tenant ID** (the *Directory (tenant) ID* from their Entra admin centre), and their email **Domains**, comma-separated (*hotel.gr, hotel.com*).
3. Leave **Is active** ticked and click **Save**.

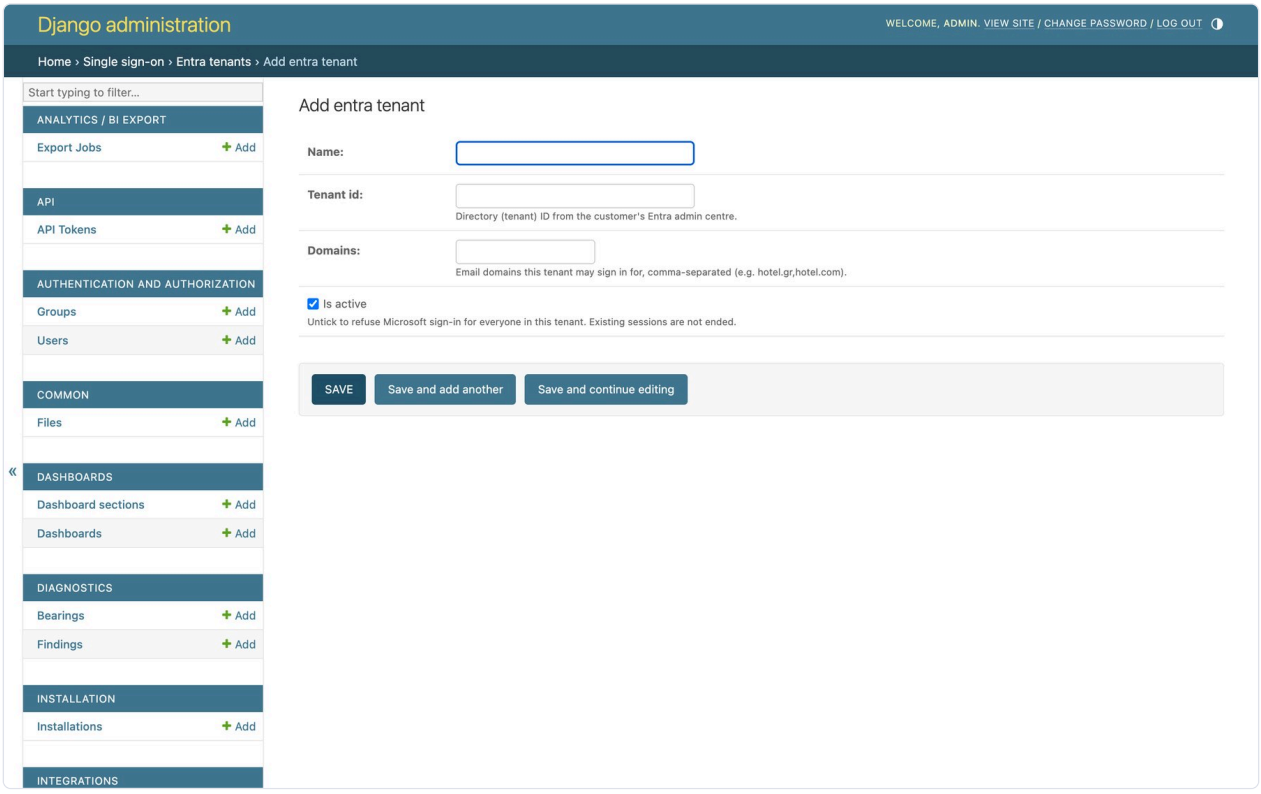


Figure 56 · Adding an Entra tenant for Microsoft sign-in

From then on, when someone types an email on one of those domains on the sign-in screen, a **Sign in with Microsoft** button appears.

A few rules keep it safe:

- Single sign-on **proves who someone is**; it never creates an account or grants access. The person must already have a Mechbase user with that work email, and be on the **Users** list of their sites.
- The first Microsoft sign-in links the account — only if the tenant is active, the email's domain is on the tenant's list, and exactly one active user has that email.

- Unticking **Is active** stops Microsoft sign-in for everyone in that tenant (existing sessions aren't ended).
- To make an account Microsoft-only, disable password-based authentication on it.

Devices, alarm defaults, and other settings

A few more pieces of administration live in their own chapters or screens:

- **MaintNode devices** — claiming and connecting the hardware that feeds automatic readings is covered in MaintNode devices and automatic data (**Setup → MaintNode Devices**).
- **Installation-wide alarm defaults** — the per-type alarm rules every point inherits (Measurements, trends, and alarms). These are maintained in Django administration under **Alarm rules**, so the whole site behaves consistently; points show them as *Installation default*. Set them early — with sensible defaults, most new points need no per-point alarm setup at all.
- **Dashboards** can also be managed in Django administration, as well as on the board itself.
- **Per-user preferences** — display name, units, and email notifications are set by each person on their own profile (Getting started); they don't affect anyone else.

Between the in-app tools and the back office, that's the full picture: build the plant and its rules in the app, manage the people and sites behind it in Django administration, and let everyone work from the same, current view of equipment health.

That's the web platform end to end.

10 · Cookbooks

Short, task-first guides for the jobs you actually do.

A cookbook is a walkthrough of one real job, start to finish — no theory, no tour of every setting, just the steps to get it done. The rest of this manual explains what each feature is; these chapters show you how to use them together. Find the one that matches the job in front of you.

For your team

The everyday cookbooks — written for the people who walk the site, log readings, and clear the work.

- Running an inspection route — for the technician on the floor: pick up a route, work it stop by stop, and close it out.
- Capturing field readings — for anyone taking readings at the machine: type a value by hand, capture a FLIR thermal image or a Bluetooth vibration signature on the phone, attach a note or photo, and see the health state land.
- Triaging Needs attention — for the supervisor at the start of the day: set your alert emails, work the Needs attention list most-urgent-first, act on each item, and resolve it or let it auto-clear.
- Building a route and setting alarm limits — for the person who sets things up: lay out a route from its stops and checks, and set the alarm limits that drive its health states.

For developers and integrators

The technical cookbooks — for getting data in and out of Mechbase and wiring it to your own systems.

- Analytics & BI — for the data analyst: pull your installation's data into pandas, Power BI, Excel, or Grafana.
- Consuming Mechbase data — for the integrator reading from Mechbase: feeds, bulk export, OData, and syncing findings into a CMMS.
- Pushing data into Mechbase — for the integrator writing to Mechbase: upload readings, files, and photos, sync your registry by external ID, and drive routes — with the SDK or the `mechbase` command-line tool.
- Streaming from a device over MQTT — for the firmware or edge engineer: stream readings straight from your own device (MaintNode doesn't need this).

> **NOTE**
> the developer cookbooks are written in English only.

11 · Running an inspection route

A start-to-finish recipe for the technician walking the round — on the phone, where it's normally done.

Routes are built to be walked on your phone. You open the Mechbase app at the machine, pick the route from the home hub, work down the checklist stop by stop, and capture a photo, a reading, or a fault right where you're standing — then it all syncs back to Mechbase on its own. This chapter runs one round start to finish on the app, the way it normally happens, and closes with the web path for running, reviewing, or completing a route from a desk. It's the hands-on companion to Inspections and routes, which covers what a route is and how to build one.

The short version

Open the app, pick your route from the home hub, and start walking. You get the **checklist**, grouped by stop, with a progress count. Tap each check to record it — a **Pass / Fail**, a **Numerical** reading, a **Multi-check**, or a **Measurement** (by hand, or straight off a thermal camera or vibration sensor) — and attach a **photo** and **notes** as you go. Fail a check and you grade it **Minor** or **Major**, which becomes a Warning or Critical non-conformity on the asset. It all works with no signal, saving to the phone first, then **syncing back to Mechbase automatically** once you're online. Tap ✔ to complete, and the run is saved as an inspection report anyone can open later.

The rest of this chapter walks each step on the phone, then shows how to do the same on the web.

Get the app and open your route

Install the Mechbase app on your phone or tablet and sign in with the same email and password you use on the web — there's no separate mobile account, and your site access comes with you. More on the apps in Mechbase on your phone.

You land on the **home hub** — your base on the phone. Your inspection routes are listed at the top, ready to walk (a dot marks one already in progress), with the plant's sections below and a status line under the site name telling you how fresh your data is. Tap your route to open it.

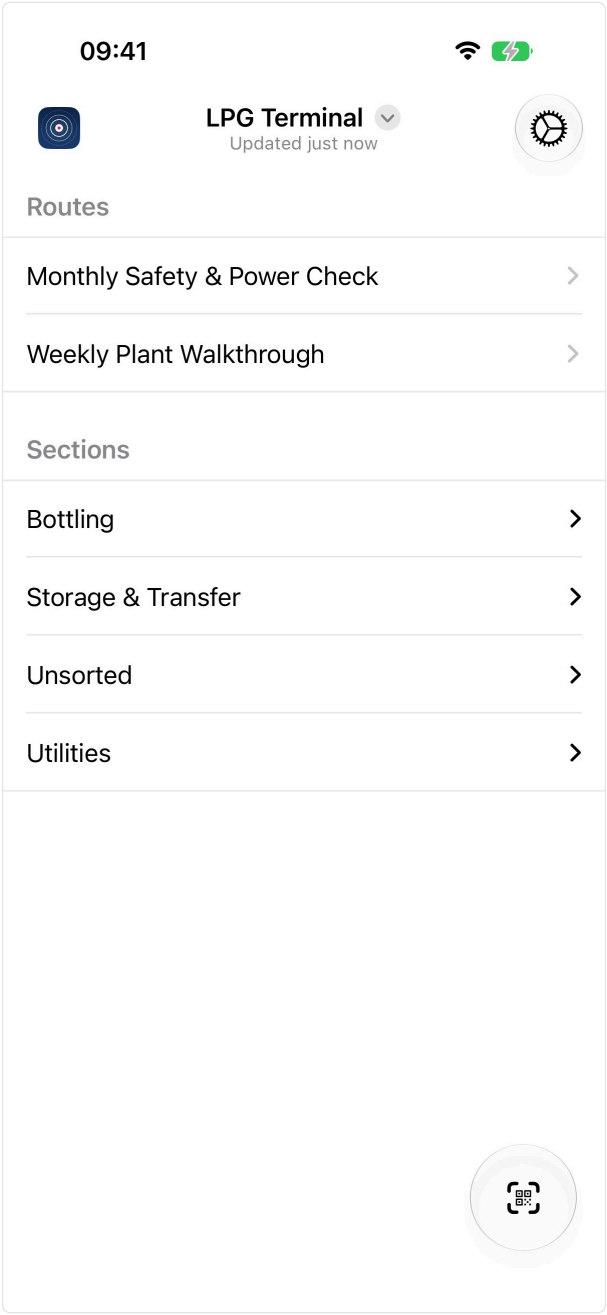


Figure 57 · The home hub — site, routes, and sections (iOS)

TIP

Standing in front of the equipment? **Scan its QR tag** from the home hub to jump straight to the asset or measurement point — and if you're already in a route that includes it, straight to its check.

Start or join the run

Opening the route shows its **checklist**, grouped by stop, with a count of how many checks are done. If nobody has started it, you start a fresh run; if a colleague already has one going, you **join it** — a route only ever has one run in progress, so everyone working the same round adds to the same report.

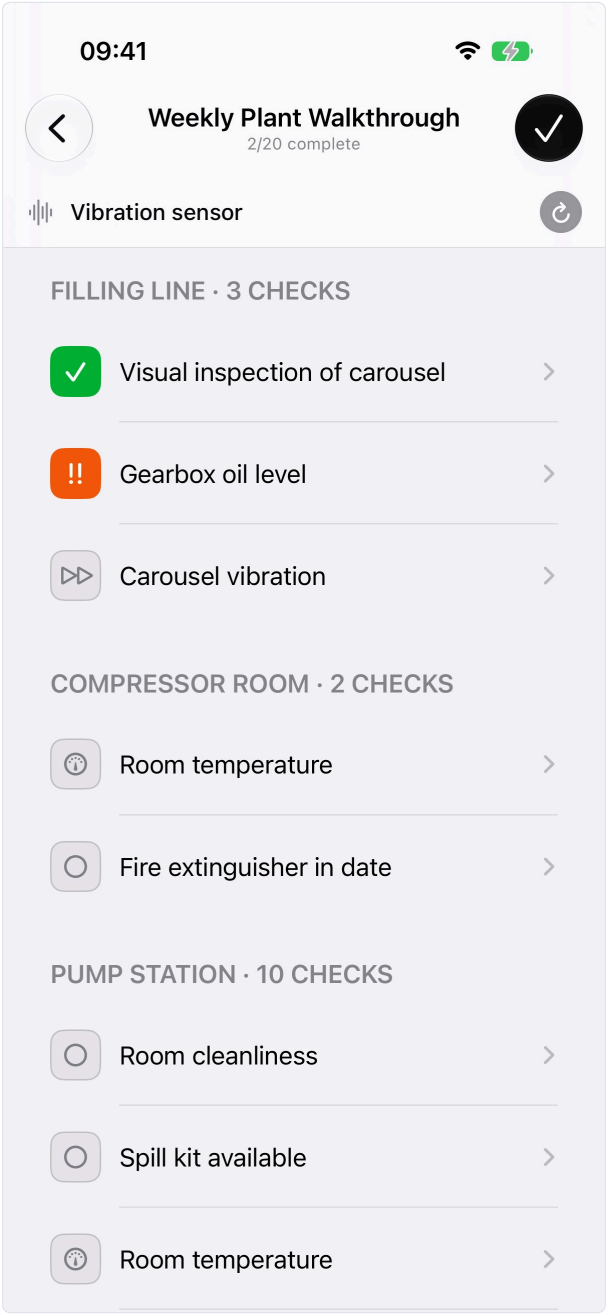


Figure 58 · Walking a route — the checklist (iOS)

> **NOTE**
> A run doesn't have to be finished in one go. Leave it and come back later — exactly what happens when a round spans a shift or you step away. And it all works **with no signal**: the app keeps a local copy of your data, so you can start and walk a whole route offline and it syncs when you're back in range.

Walk the stops

The checklist is grouped by stop in the order someone actually walks the site. Work down it, tapping each check to open and record it. The progress strip across the top shows where you are — tap it to jump to any check. Each check's icon on the checklist shows its result by shape once you've answered it: a tick for a pass, `!` or `!!` for a failure, a skip mark for one you passed over.

At the bottom of each check, the centre button records it and takes you to the next one (or skips it if you haven't answered); the arrow at the top goes back. **Clear** wipes an answer if you recorded something wrong — with a few seconds to **Undo**.

Complete each kind of check

Tap a check and the app shows the right input for its **type**, plus room for a photo and notes. What you record depends on the type:

- **Pass / Fail** — tap the **tick** to pass (it records and moves on in one tap), or the **cross** to fail. A failed one becomes a non-conformity.
- **Numerical** — type the number you read off the gauge (a temperature, a pressure). If the check has limits, a bar shows where your value falls — **OK**, **Watch**, or **Alarm** — and the **last** reading is shown so you can sanity-check yours.
- **Multi-check** — several pass/fail steps bundled into one check (a panel's emergency stop, relays, interlocks, and fan). Mark each step on its own.
- **Measurement** — take a reading on the asset's measurement point (a vibration, a temperature, a thermal image). It flows into that point's trend exactly like an automatic reading. On the phone you can capture this straight from a FLIR camera or a Bluetooth vibration sensor — see Capturing field readings.

If the measurement point has **instructions**, they're on the check too — where to put the probe, which axis, what to compare against. Read them; they're what make this month's reading comparable with last month's.

Capture photos, notes, and sensor readings

Every check has room for **evidence** — attach a **photo** from the camera and add free-text notes in **Add notes...**. It's optional on a passing check, but do it on a failing one: the photo and notes you add here become the non-conformity's evidence, so photograph the fault while you're standing in front of it.

The apps do one thing the web can't — **capture live sensor data at the machine**. A **vibration** check can take a reading straight from a paired Bluetooth sensor, and a **thermography** check can capture frames from a **FLIR thermal camera**, hotspots and all. The full recipe is in Capturing field readings; sensor details are in Mechbase on your phone.

When a check fails

When you **fail** a Pass / Fail check — or a failed step inside a Multi-check — grade how serious it is:

- **Minor** (`!`) — the non-conformity is raised as a **Warning**.
- **Major** (`!!`) — it's raised as a **Critical** non-conformity.

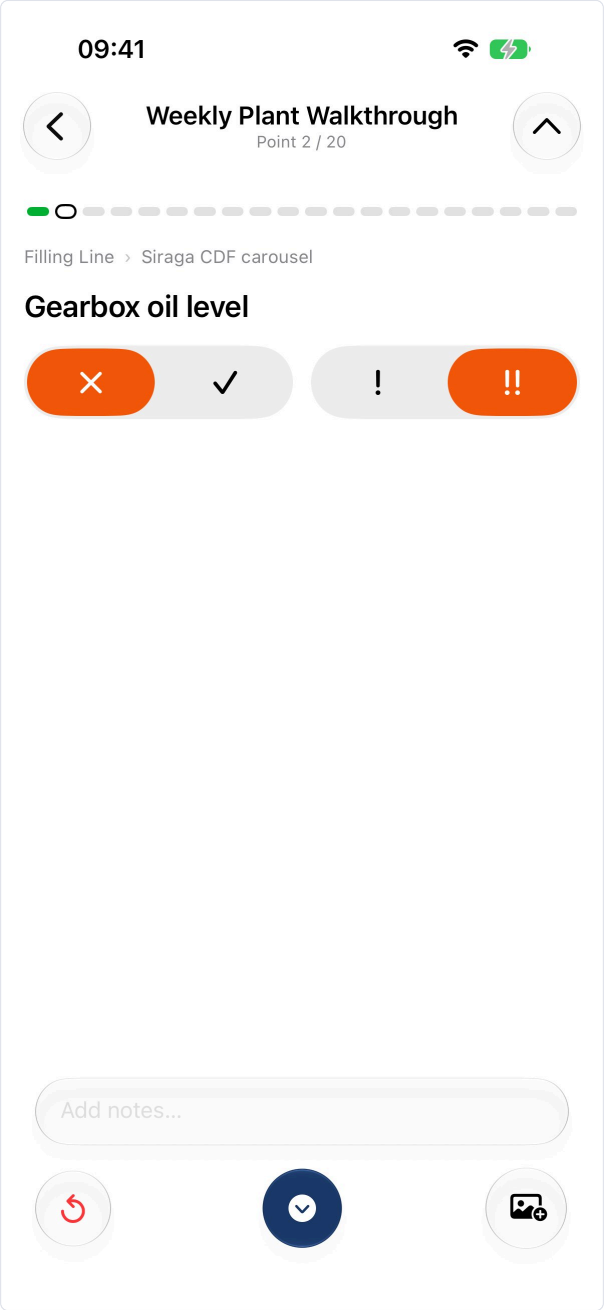


Figure 59 · A failed check graded Major (iOS)

A **non-conformity** is just a check that didn't pass — a failed Pass / Fail item, or a reading that breached its limits. You don't log it separately; it's created the moment you record a failing result, and the photo and notes on that check become its evidence.

From there it travels with the asset, not just the inspection. It's pinned at the top of the affected asset's **Activity**, raises the asset's state, and appears on **Needs attention** — where your team can see it, discuss it in the asset's notes, and clear it with **Resolve** once the problem is dealt with. That's the thread from someone spotting a fault on a round to it being tracked to closure. More on the whole cycle in Inspections and routes.

Finish the run and sync

When you've worked through the list, tap the ✔ at the top right of the checklist to complete the run. You don't have to answer every check — skipped ones are simply left unanswered (and can be filled in later with **Amend** on the web).

Everything you recorded offline — results, readings, photos, and sensor captures — saves to the device first, then **syncs back to Mechbase automatically** once you're online again, in the background. The home hub's status line tells you where things stand, and **pulling down** on it forces a sync.

> **HEADS-UP**
> Signing out deletes anything on the device that hasn't uploaded yet — the app warns you before it does. On a shared field phone or tablet, let a walked route sync before you sign out.

Open the inspection report

Every completed run is saved as a report, which anyone can open on the web — from the route's **Latest report** (`L`) or **History** (`H`), or from the activity feed.

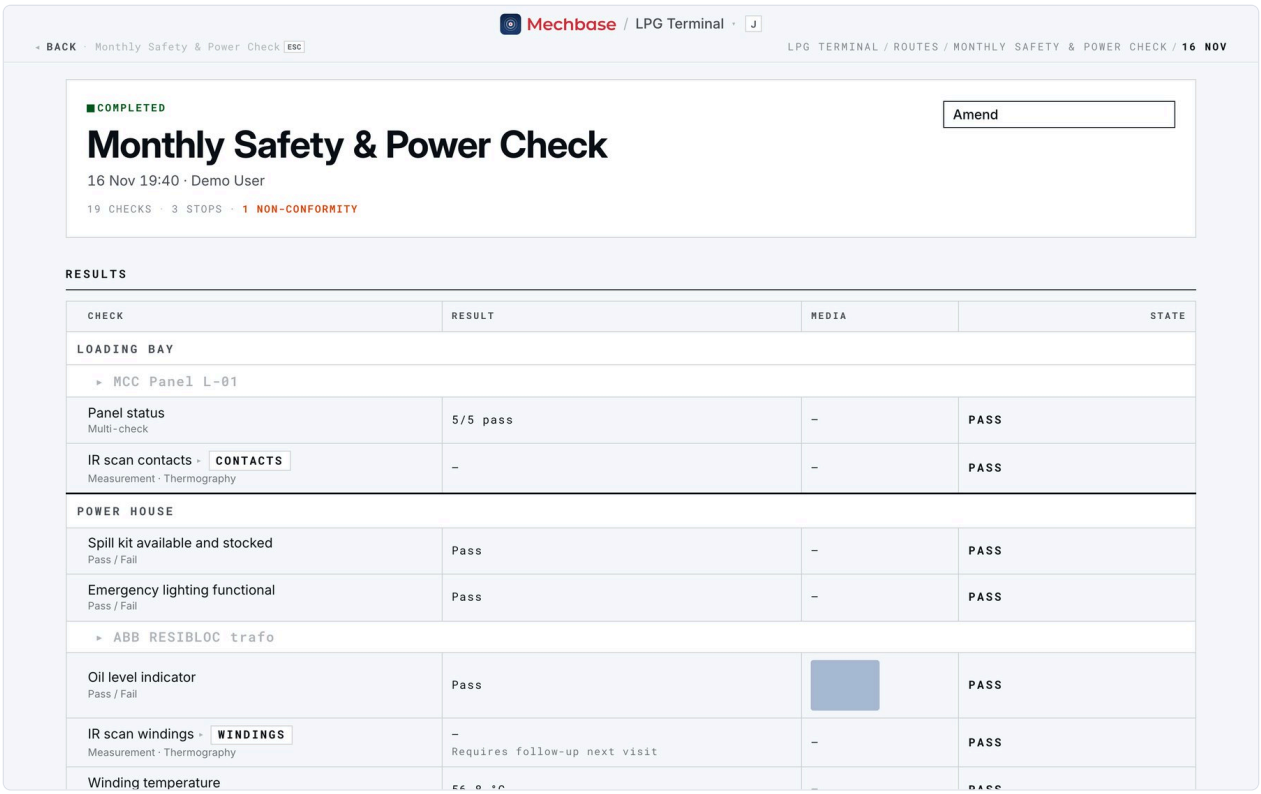


Figure 60 · A completed run — the inspection report

The report records who ran it and when, the totals (checks, stops, and the number of **non-conformities**), and a **Results** table — every check with its result, any photos, and its state, grouped by stop and asset. Failed checks are highlighted, so a director skimming the report sees the exceptions immediately. Click a measurement or a failed check to expand it: you get the point's trend around the run, the previous and current values, and a **Finding / Recommended action** field you can fill in right there.

Also on the web

You can also run, review, or tidy up a route in the browser — handy from a desk, or when you're checking up on a round rather than walking it. Open **Routes** (or press `2`), pick your route, and click **Start route** (`R`) — or **Resume route** if a run is already in progress.

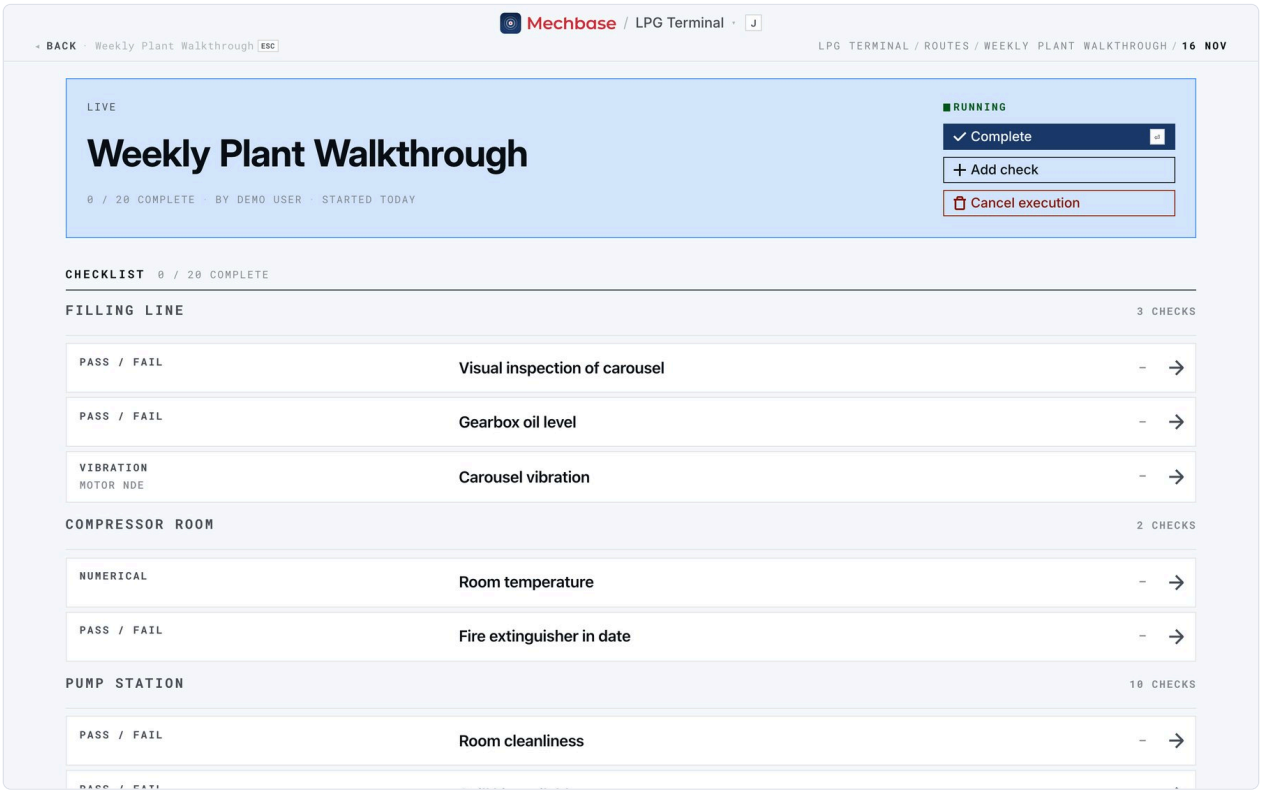


Figure 61 · Running a route — the live checklist (web)

The web run works the same way: open a check, record it, move with the arrows, **Clear** to undo. A failed check asks for its severity too — **Warning** or **Critical** — so grading isn't limited to the phone. **Add check** adds something you spotted that isn't on the list, and **Complete** finishes the run.

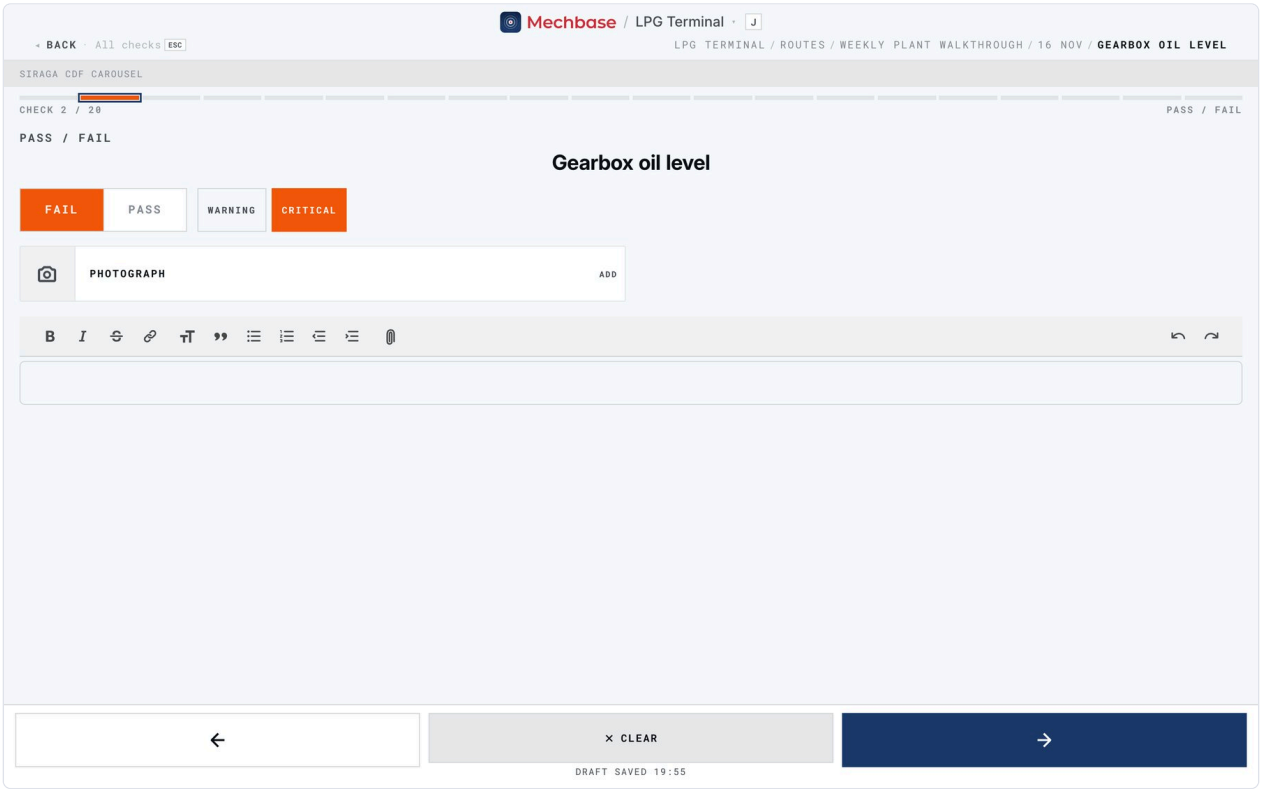


Figure 62 · Failing a check on the web — choose Warning or Critical

After the run, the web is where you finish the paperwork:

- **Latest report** (`L`) opens the report; fill in each row's **Finding / Recommended action**.

- **Amend** on the report lets the technician who ran it (or a staff user) answer skipped checks, add a reading taken afterwards, or add the photos that were left on the phone. Recorded results stay as they were, and every change is logged.
- **Print** the report from your browser for a paper copy, with every row expanded.

Quick reference

STEP	ON THE PHONE
Open your route	Home hub → tap the route (or scan the asset's QR)
Start / continue a run	Open the route — you join a run already in progress
Move between checks	the next button, the back arrow, or the progress strip
Undo one entry	**Clear** (then **Undo**)
Grade a failed check	**Minor** (`!`) or **Major** (`!!`)
Finish the run	✔ at the top of the checklist
Force a sync	pull down on the home hub

CHECK TYPE	WHAT YOU RECORD
Pass / Fail	**Pass** or **Fail**; grade a fail Minor/Major (phone) or Warning/Critical (web)
Numerical	The number off the gauge, judged against its limits
Multi-check	Each bundled pass/fail step on its own
Measurement	A reading on the point — by hand or from a sensor, including thermal images; flows into its trend

ON THE WEB	
Open the routes list	**Routes**, or press `2`
Start / continue a run	**Start route** / **Resume route** (`R`)
Finish the run	**Complete**
Latest report · past runs	**Latest report** (`L`) · **History** (`H`)
Fill in what was missed	**Amend** on the report

12 · Capturing field readings

Taking a reading at the machine — by hand, by thermal camera, and by vibration sensor.

You're standing at a pump with your phone. This chapter is the recipe for turning that moment into a reading Mechbase can act on — a number, a thermal image, or a vibration signature that lands on the point's trend and can move the asset's state. It's the same workflow whether you reached the point from a route or opened it on its own, and it works with no signal at all.

If you only read one thing, read **Where the reading lands** — that's what a reading is *for*.

Before you start

Three things need to be in place — all covered in Mechbase on your phone:

- **The app, signed in.** You're on the home hub, on the right installation (the site you're visiting).
- **Sensors at hand**, if you're capturing thermal or vibration. The **FLIR ONE Edge** camera connects when you open a thermal capture; the **Bluetooth accelerometer** connects automatically — wake it with a tap of the magnet (Your vibration sensor).
- **Nothing else.** No signal required — see Work offline, sync later.

> **TIP**
> Connecting a sensor takes a moment. The banner at the top of the route or point shows its state — *Connecting…*, connected, or *Sensor lost* — with **Connect** / **Reconnect** and an ⓘ for the steps to wake and link it. On iPhone the thermal camera starts linking as soon as you open a route that needs it; tap **Connect** for the vibration sensor when you reach your first vibration check.

Open the point in front of you

You reach a measurement point one of three ways:

- **Browse the plant** on the home hub — **Sections → Zones → Assets**, then tap the point under the asset's **Measurement Points**.
- **Scan its QR code** — tap the scanner and point at the code on the equipment; the app jumps straight to that point (or, if you're in a route that includes it, to its check).
- **Work a route** — open the route and tap a **Numerical** or **Measurement** check; it opens on the right point for you (Walking a route).

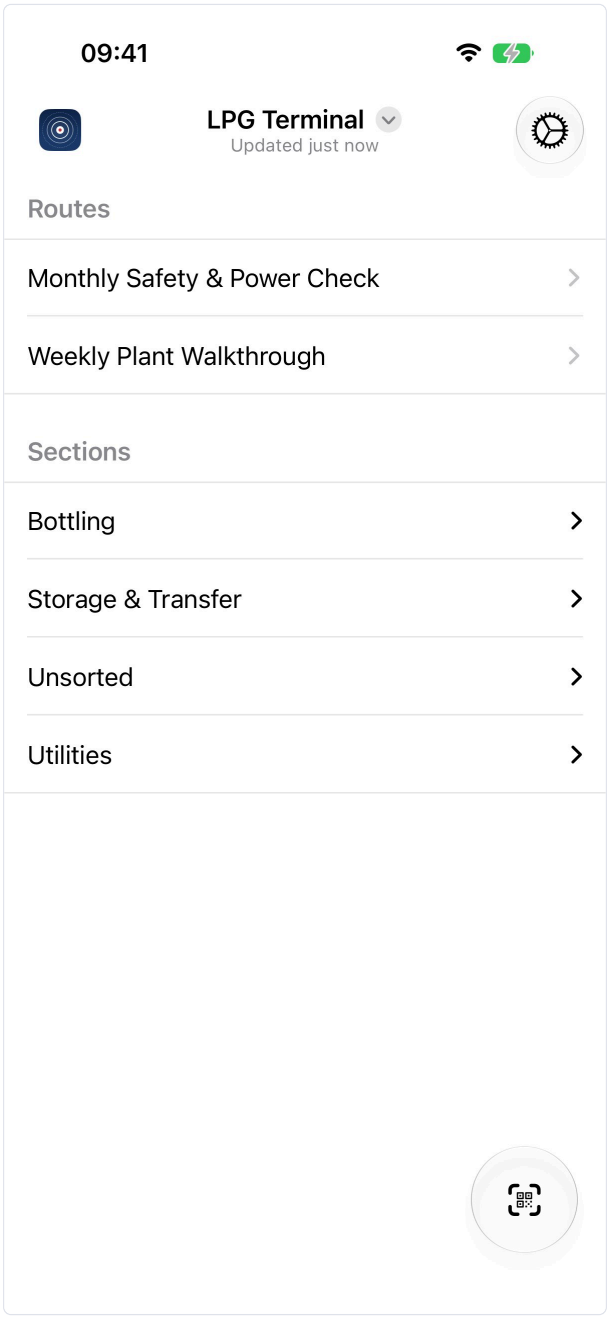


Figure 63 · The home hub — browse down to a measurement point (iOS)

The point opens with the input that matches its **type** — a number field for most points; for thermal and vibration points, a sensor banner and a **Capture** button that stays disabled until the sensor is connected. The next three sections cover each in turn.

Take a reading by hand

The everyday case — a temperature off a gauge, a pressure, a level.

Type the value in the large field and save (✔). Under the field, **last:** shows the previous reading, so you can sanity-check before you commit — if you're about to enter `4.2` where the last reading was `42`, that gap is right there to catch. On a route check with limits, a bar shows where your value falls — **OK**, **Watch**, or **Alarm**. Opened on its own, the point also lists its **Recent** readings below.

The value goes in the point's own unit, shown beside the field (ultrasound is always in dB). On the web, everyone then reads it in their own units (Reading a trend).

Capture a thermal image

For a **thermography** point, the app connects to the **FLIR ONE Edge** camera and shows a **live thermal preview** — with a temperature scale down the side.

1. Frame the equipment in the preview. Under **Options**, pick the **Color** palette that shows the detail you want, and set the **Emissivity** to suit the surface (*Matte*, *Semi-matte*, *Semi-glossy*, *Glossy*). **NUC** re-calibrates the sensor — the click you hear now and then is normal.
2. **Capture** a frame. The app **automatically finds hotspots**, marking the hottest regions for you.
3. Tap to add a hotspot, or tap a marker to edit it and grade it **Minor** or **Major**. Then **Save** — or **Retake**.

The thermal image and its hotspots attach to the reading, and a copy is saved to the phone's photo library. A route check can hold several frames.

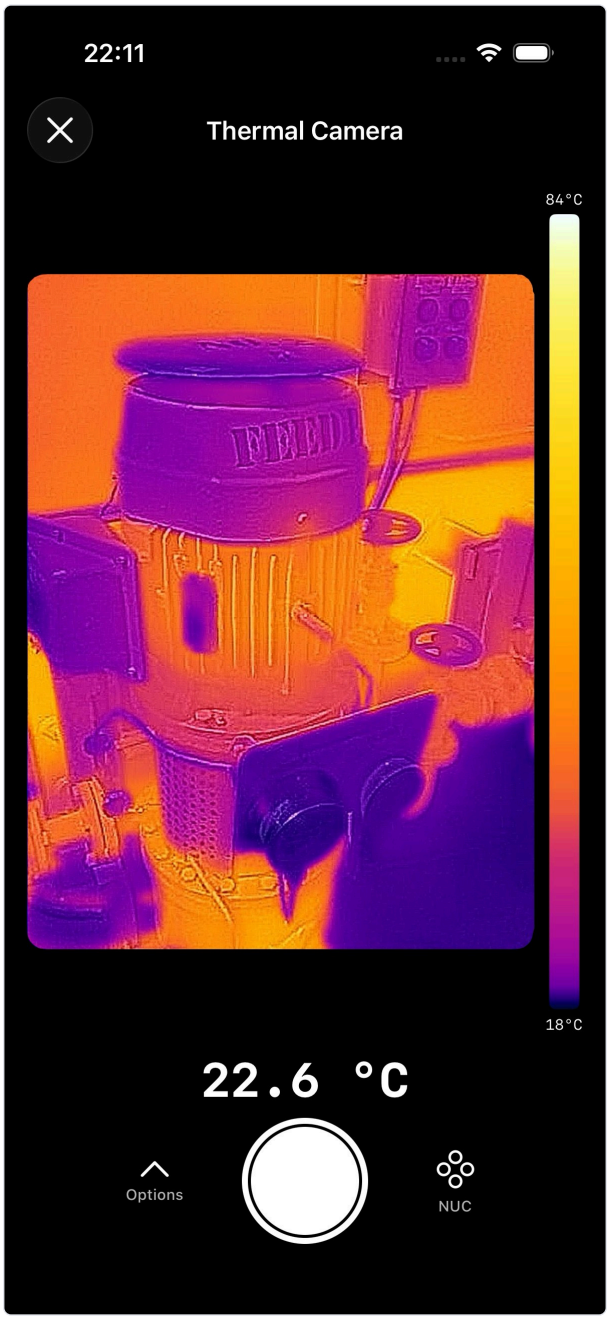


Figure 64 · Live thermal preview from a FLIR camera, with palette and temperature scale (iOS)

Capture a vibration reading

For a **vibration** point, the app connects to the **Bluetooth accelerometer**. Pick the **X**, **Y**, or **Z** axis and capture: the sensor records about one second of waveform.

When the capture finishes you get the sensor's **RMS**, **Peak**, **Crest**, and **Vel 10Hz** figures above the **Waveform** and **FFT magnitude** plots. **Save** attaches the **full signature** — the same waveform an analyst opens on the web — or **Retake** to try again.

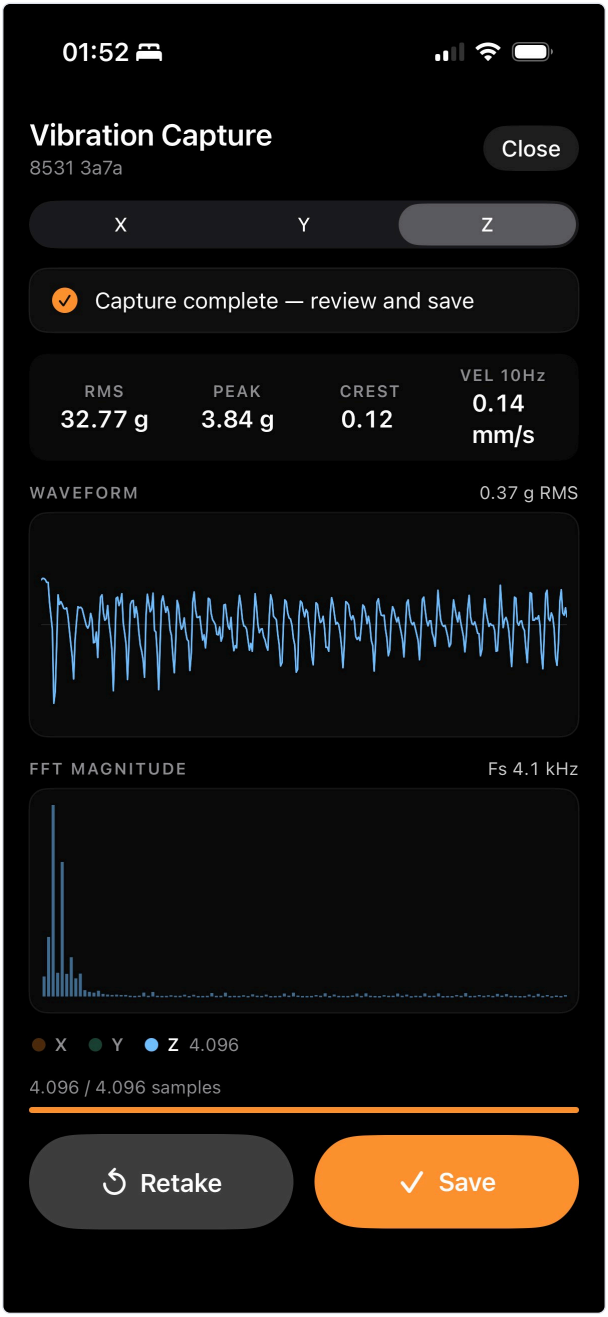


Figure 65 · Vibration capture: per-axis RMS, peak, crest and velocity above the live waveform and spectrum (iOS)

Because the whole waveform comes back with the reading, the point's inspect view on the web — time waveform, spectrum, envelope, order lines, bearing markers — has everything it needs (Inspecting a vibration reading).

HEADS-UP

Mount the sensor on the same spot, the same way, every visit — on clean, bare metal in the bearing's load zone, never on cooling fins or covers, with the X axis along the shaft. On a curved housing use a two-pole magnet and roll it on rather than letting it snap. A vibration reading only compares cleanly to history when it's taken from the same mounting point in the same direction.

Add a photo and notes

Every check — and every point opened on its own — has room for evidence, and it's worth using:

- **Photo** — **Take Photo** with the camera or **Choose from Library**. Snap the gauge face, the nameplate, the puddle under the coupling. One picture saves a paragraph of explanation later; camera photos are also saved to the phone.
- **Notes** — free text in **Add notes…** for anything the number doesn't say: *"reading taken with guard off"*, *"new noise at start-up"*.

Both ride along with the reading and show up against it on the web, so whoever triages it later sees what you saw.

Work offline, sync later

Plant rooms and remote sites don't always have a signal, and you don't need one. The app keeps a **local copy** of your data and saves every reading **on the device first**:

- Open points, take numerical, thermal, and vibration readings, attach photos and notes — all offline.
- Everything **syncs back to Mechbase automatically** once you're online again, in the background. The home hub's **sync status** line tells you where things stand, and **pulling down** forces a sync.

HEADS-UP

Signing out deletes anything on the device that hasn't uploaded yet — the app warns you first. On a shared field phone or tablet, make sure your readings have synced before you sign out.

Where the reading lands

Saving isn't the end of it. Every reading you take runs through the same pipeline as one that arrived automatically from a MaintNode device (How a reading becomes a state):

1. **It's recorded** against the measurement point and added to the point's **trend**.
2. **It's scored** against the point's alarm rule — normal, minor, or major.
3. **It's combined** with recent readings, so a sustained pattern moves the state and a lone spike doesn't.
4. **The asset's state rolls up** from its points and open non-conformities — the worst of them sets the asset's state, across Nominal / Check / Warning / Critical.
5. **Everything downstream updates** — the asset's colour, the section counts, any dashboards, the **Needs attention** list, and alert emails.

So the reading you just took at the machine can be what moves an asset onto **Needs attention** for someone to triage — which is exactly where the next chapter picks up.

13 · Triaging Needs attention

Working the Needs attention list each morning.

Open Mechbase, work the **Needs attention** list from the top, and you'll have dealt with the day's real problems in the order they matter — this chapter is the recipe for doing it fast and in the right sequence.

The one-minute version

Open **Needs attention** (press `3`), work from the top (Critical, then Warning, then Check), click each asset to see why it's flagged and how it's trended, then act. Sensor-driven items clear themselves once readings settle; a failed inspection check you close by hand with **Resolve**. Finish with the **Silent** rows at the bottom. That's the whole loop. The rest of this chapter is the detail.

For the reference on the screen itself, see Staying on top of what needs attention.

Set up your alerts once

Triage starts before you open the app. On **My Profile → Email notifications**, choose your **Alert level** — *Check and above*, *Warning and above*, or *Critical only* — and whether you want **Status changes** and **Route reports**. Click **Save notifications**, then **Send test email** to make sure it arrives. From then on, a new problem at your level reaches your inbox without anyone watching the screen (Getting started).

Open Needs attention

The home page's **Needs attention** panel shows the top six items, with a count of how many are **active** and how many are **silent**. The full list lives on its own page — open it from the **+N more** link on that panel, from **Needs attention** in the menu, or press `3`.

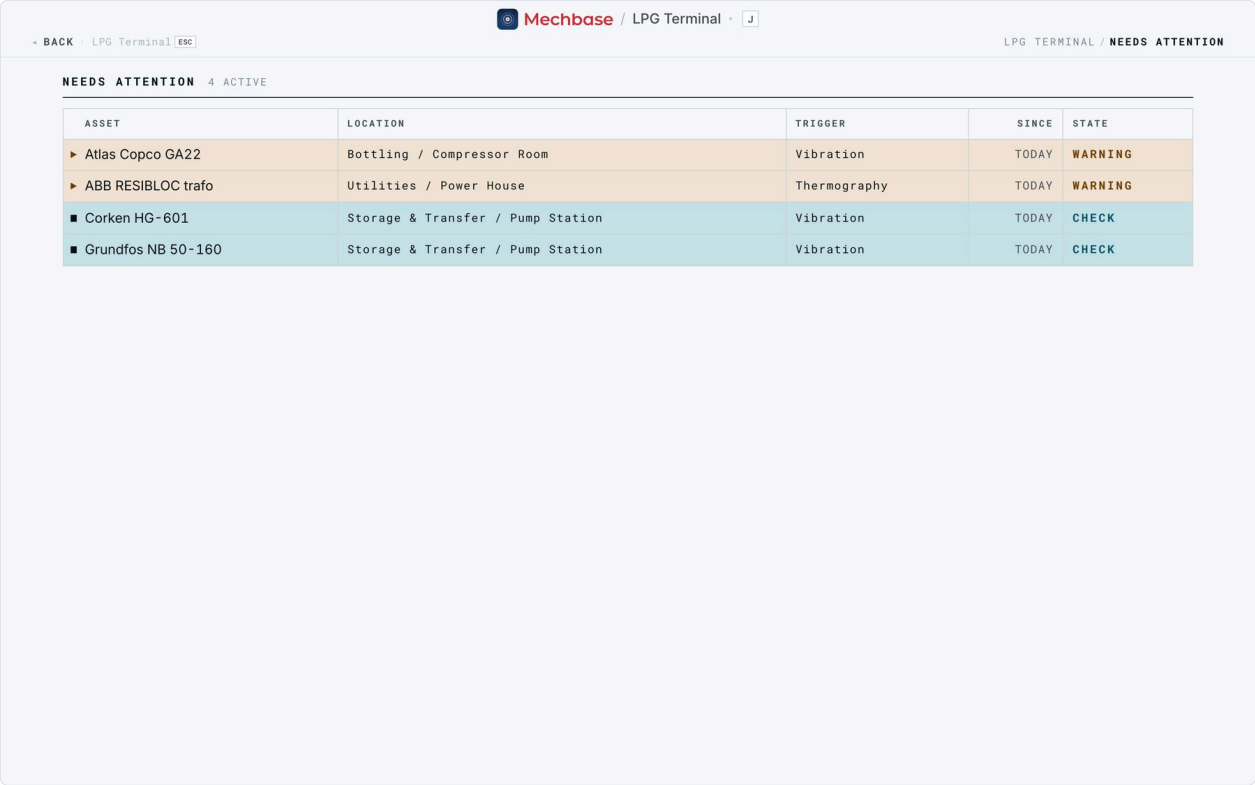


Figure 66 · The full Needs attention list

It's every asset at the site that isn't currently Nominal, most urgent first. If it's empty, it says *"All assets are nominal"* and you're done.

> **TIP**
> Responsible for just one area? Open your section from its card on the home page — its table shows every asset in the area with its state, zone by zone. Or use **Points** (5) with the section filter to see every reading in your area at once.

Work it top-down, most urgent first

The list is already sorted by priority, so working from the top *is* working by urgency. Take it in this order:

1. **Critical first** — red, severe, act now.
2. **Then Warning** — amber, clearly degraded, plan an intervention.
3. **Then Check** — teal, an early soft signal.

Every row carries the state word as well as the colour. Read the **Trigger** column as you go to know what raised each one — the measurement point that's out of limits (*Motor DE*), or the inspection check that failed (*Oil level and quality*). The **Since** column tells you how long it's been in that state.

> **TIP**
> Don't skip the **Check** items. A Check is Mechbase telling you a soft pattern is forming — the cheapest time to look is before it becomes a Warning.

Open an item to see why it's flagged

Click the row to open the asset. There you get the full picture:

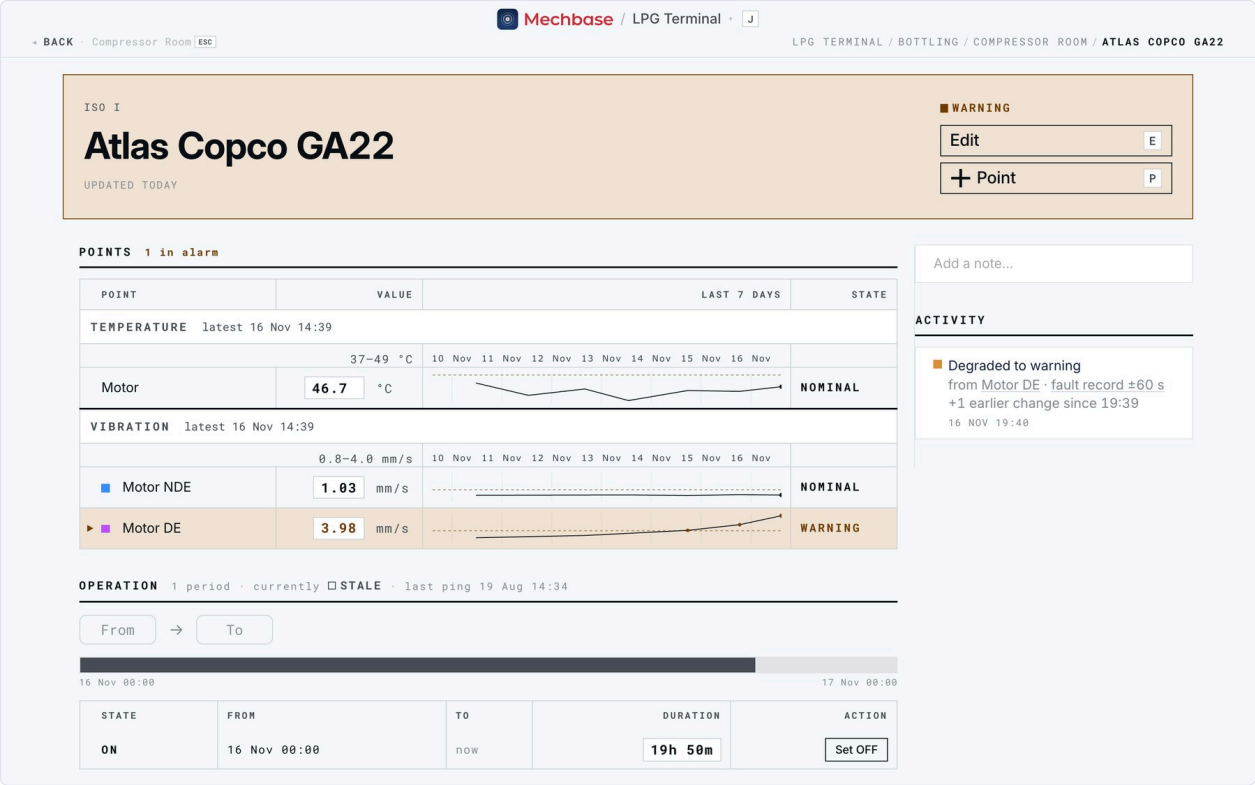


Figure 67 · An asset opened from Needs attention: points in alarm, trends, and activity

- The **Points** section says how many are **in alarm**, with a week of trend on every sensor. Scrub across the trends to see what every sensor was doing at the moment the trouble started.
- **Activity** gives the dated, plain-language trajectory (*"degraded to warning"*, *"improved to check"*), with a **fault record ±60 s** link on changes caused by a reading — a one-page snapshot of every sensor on the machine in the minute before and after.
- Open non-conformities from inspections are pinned at the top of Activity.

That history is what tells you whether this is a one-off spike or a trend that's been building for weeks. For a vibration point, click it and open the **Inspect** panel to look at the waveform and spectrum (Measurements, trends, and alarms).

Act, then clear or resolve

Needs attention reflects reality — it's not a to-do list you tick off. You clear an item by fixing the underlying condition, and it comes off the list on its own. Which of two things happens depends on what raised it:

- **Sensor-driven items** (vibration, temperature, and the like) clear **automatically** once readings return to normal. Because Mechbase waits for the *pattern* to recover rather than the first good reading, a cleared item genuinely means the asset has settled — so act on the real-world cause and let the next readings close it.
- **Inspection-driven issues** — a failed check, a **non-conformity** — you close **by hand**. Open the affected asset, find the item pinned at the top of **Activity**, deal with the real-world problem, and click **Resolve**. Add a **Resolution comment** saying what you did, and click **Mark resolved**.

Either way, leave a **note** on the asset as you go (*"Bearing greased, recheck Thursday"*) — or, if you're the analyst, **Record finding** on the reading with the fault, severity, and recommended action, so it can flow to your maintenance system. When the condition is resolved the asset drops off Needs attention, the counts tick down, and the change is written to the **activity** feed — so there's a dated record of both the problem and its fix.

HEADS-UP

You never set a state by hand to make an item disappear. If a sensor item is still on the list, the readings haven't settled yet — chase the cause, not the flag.

Don't forget the silent ones

After the health alerts, the list shows **Silent** assets — ones whose sensors used to report but have stopped. Their trigger reads *"No reading: …"*. A silent asset might be perfectly healthy, or failing where no one can see it. Check the sensor and its MaintNode (Is my device healthy?), or the route that should have measured it.

Where the same items show up

You won't only meet these items on the list — Mechbase surfaces the same backlog wherever you're already working, so triage stays honest across the whole app:

- The **Needs attention** panel on the home page and the count badge on the menu.
- The **warning** and **critical** counts on each section card.
- The state and **in alarm** count on each **asset** page.
- **Dashboards**, where an alarmed point's row is tinted.
- The **activity** feed, as each state change happens.
- **Your inbox**, at the alert level you chose.

14 · Building a route and setting alarm limits

Two jobs an engineer sets up once: a repeatable inspection round, and limits that fire only when they should.

This is a recipe for an engineer or admin standing up a site. Part 1 builds a **route** — the inspection round your team walks. Part 2 sets the **alarm rules** — the limits that decide when a reading is worth someone's attention. Do both well and the two halves of Mechbase reinforce each other: a measurement check on a route feeds the same trend an alarm rule watches, so a manual reading and an automatic one are judged against exactly the same limits.

The short version

Two jobs, done once. **Part 1:** open **Routes**, create a route, and fill its checklist with the **stops** (zones or assets) you'll visit and the **checks** at each. **Part 2:** open each measurement point's **Alarm rules** and either lean on the default or create an override with **Warning** and **Critical** limits. Then leave both to settle over a short learning period and trim the limits against what you actually see.

You'll need the assets and their measurement points already in place (Working with assets). For the full detail behind each half, see Inspections and routes and Measurements, trends, and alarms — this chapter is the condensed how-to.

Build the route

A route is a named list of things to check, in a sensible walking order. You build it once; every walk-through creates a run against it.

Create it

Open **Routes** (press `2`), click **New Route** (`N`), give it a **Route name** and an optional description, then click **Create route**.

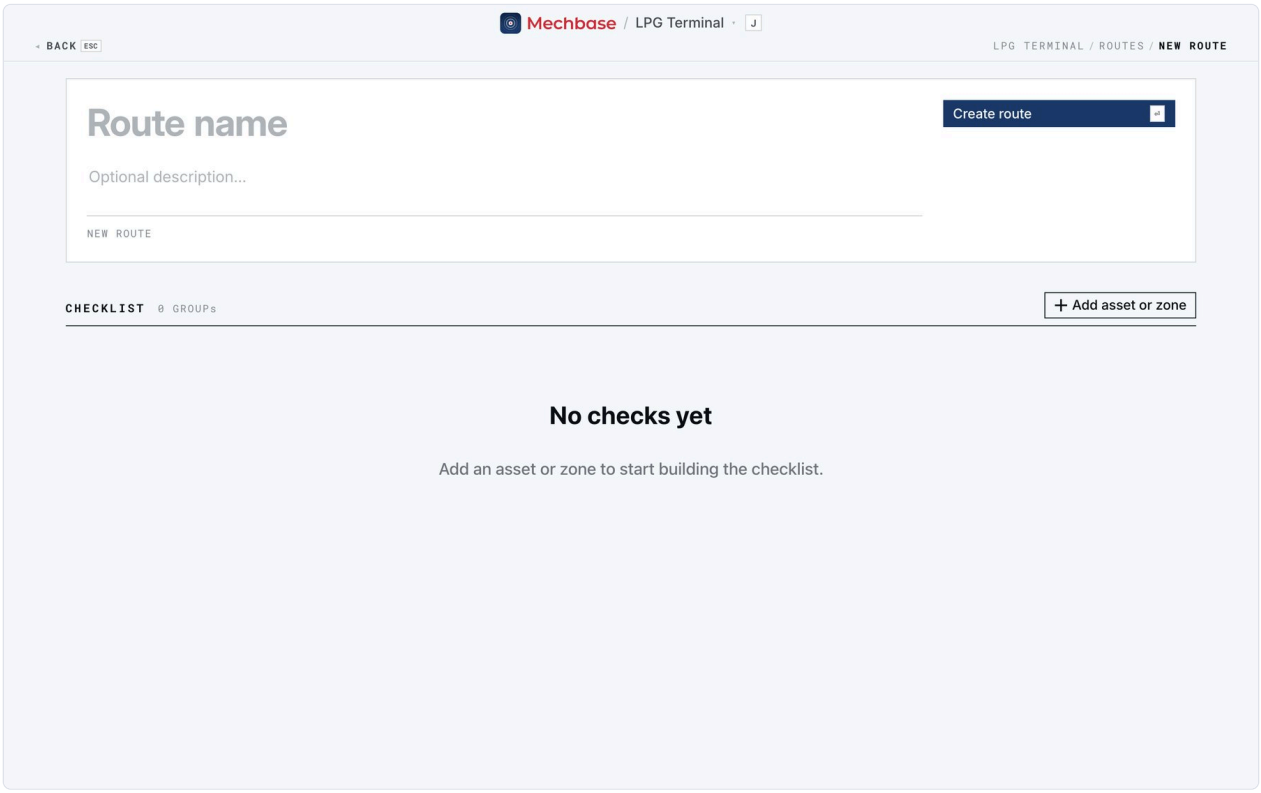


Figure 68 · Starting a new route

Add stops and checks

A new route starts empty. Click **Add asset or zone** and pick the first **stop** — a whole zone, or a single asset. Add the rest the same way. Then, on each stop, click **+** to **Add check**, give it a **Label**, and choose its **Type**:

- **Pass / Fail** — a yes/no judgement, like *"fire extinguisher in date"*.
- **Numerical** — a number read off a gauge, in the **Unit** you set (*bar*, *°C*).
- **Multi-check** — several pass/fail **Steps** bundled into one check (a panel's e-stop, relays, interlocks, fan). Add each step; it becomes its own tick during the run.
- **Measurement** — a reading on one of the asset's measurement points; pick the **Measurement point**. It flows into that point's trend like an automatic reading, and is scored against the same alarm rule you set in Part 2. Thermal images and vibration captures are Measurement checks on thermography and vibration points.

A check's type can't be changed once it's created, so pick carefully.

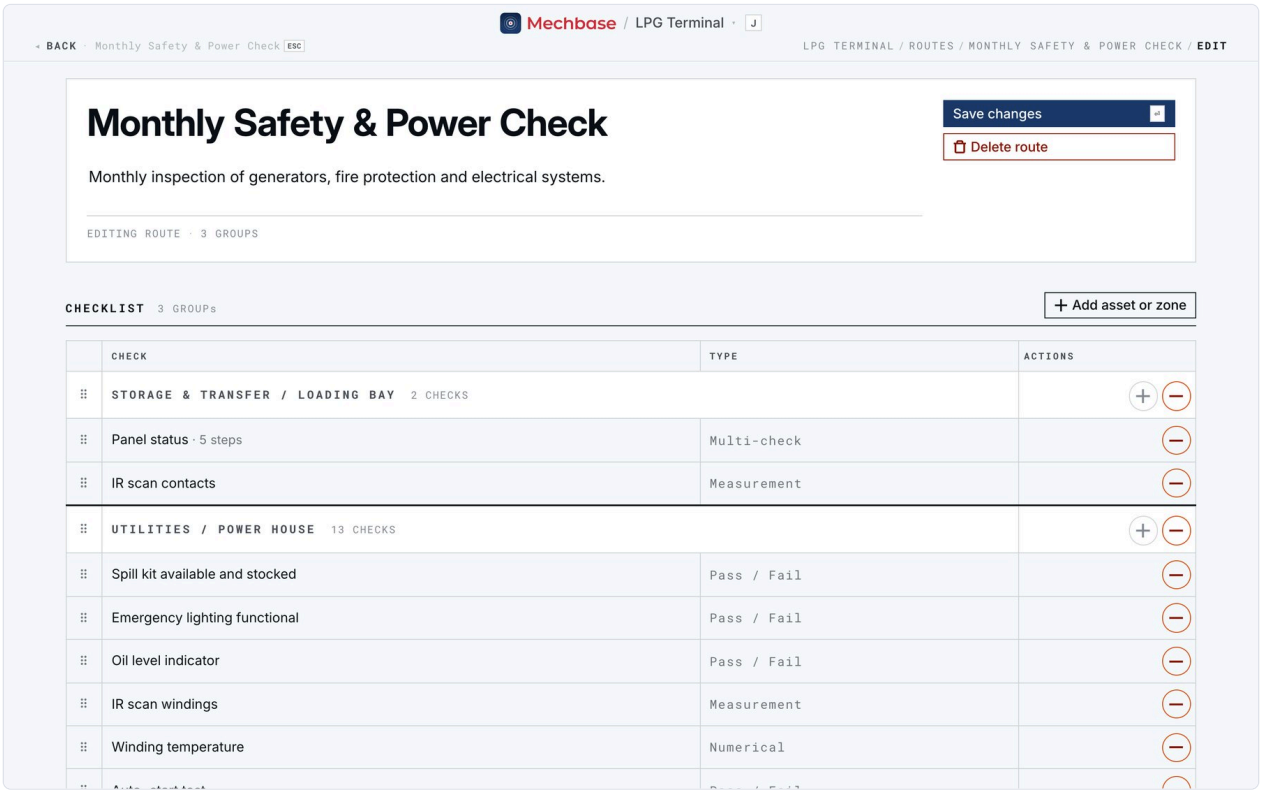


Figure 69 · The route editor, with stops and their checks

Order it and save

Drag the handles to reorder stops and checks so the route follows the natural walking path of the plant (or use **Move to different zone or asset**), then click **Save changes**. The route's page shows its totals — **stops**, **checks**, and **runs** — and, until it's first walked, its **Checklist**. From here **Start route** (`R`) walks it, **Latest report** (`L`) opens the most recent result, and **History** (`H`) shows every run.

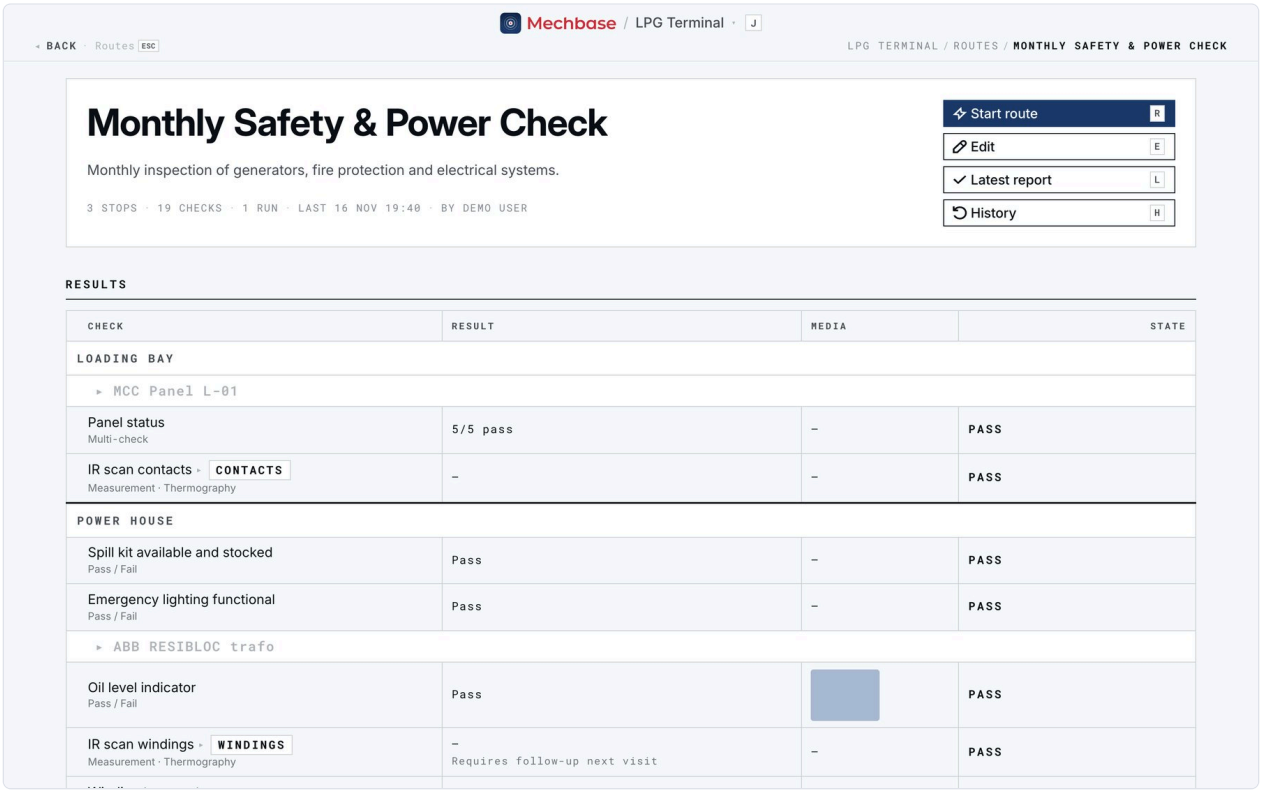


Figure 70 · The finished route — its page and latest results

TIP

Order the stops the way someone actually walks the site, and keep each check's name short and unambiguous — it's what the technician reads on their phone at 6 a.m. Put *how* to take a reading in the measurement point's **Instructions**: they're shown on the check.

TIP

Don't know the route yet? Walk it once as a **New Survey**, adding checks as you go — it becomes a normal route when you complete it.

Set the alarm limits

Alarms are what make **Needs attention** mean something. They're defined by **alarm rules**: an **installation default** per measurement type (and the **ISO 10816** default for vibration), maintained centrally, and a **point override** for the cases where one point should behave differently.

See what's in effect

Open a measurement point and click **Alarm rules**. The line at the top tells you where the rule comes from — **Point override**, **ISO … default**, **Installation default**, or **No rule configured** — and what it says.

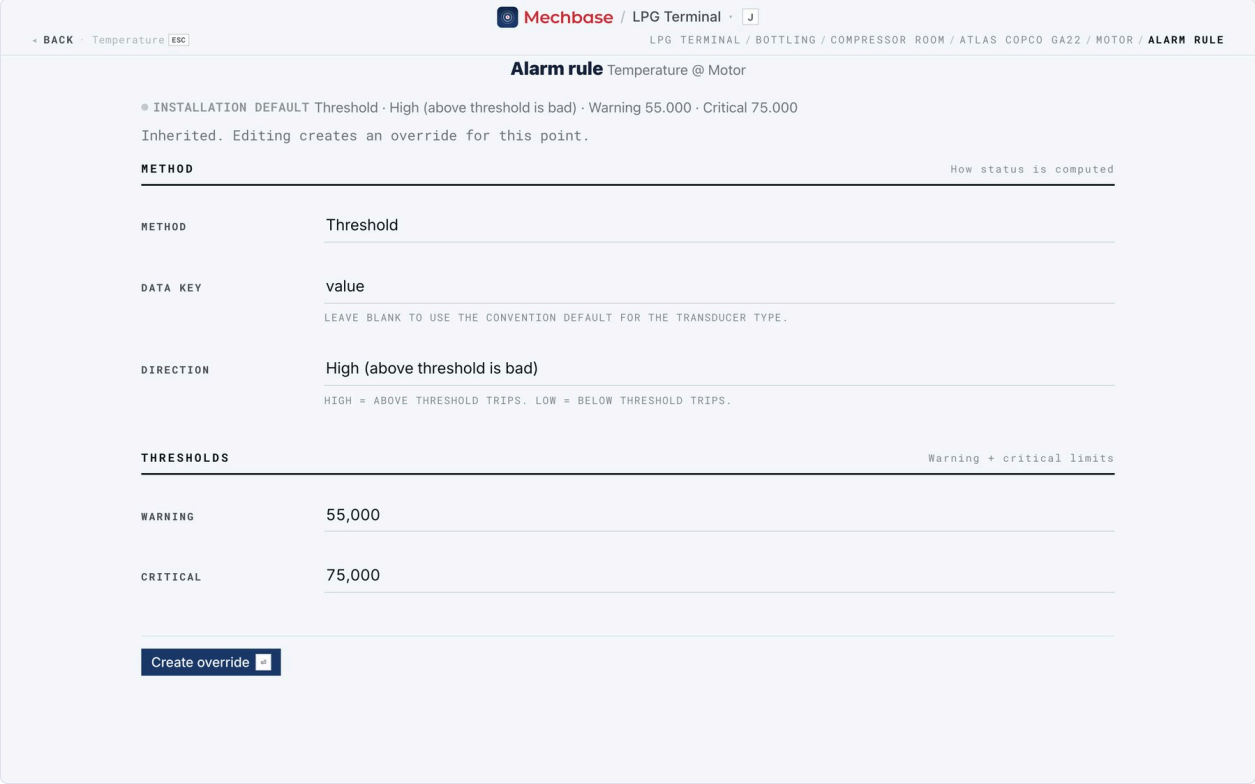


Figure 71 · The alarm rule in effect for a point

Override one point

When the point is inheriting a rule, the form below is already filled in with it — *"Inherited. Editing creates an override for this point."* Adjust it:

- **Method** — how the status is worked out. **Threshold** (a simple limit on the value — the common case), **Narrowband Envelope** (watches a specific frequency band for a particular vibration fault), or **Maintenance due** (raises a status on time or usage).
- **Data key** — which value to watch; leave it blank for the usual one.

- **Direction** — **High** (above the threshold is bad: temperature, vibration) or **Low** (below is bad: a flow or pressure that shouldn't drop).
- **Thresholds** — the **Warning** and **Critical** limits.

Click **Create override** (or **Create rule** where no default applies). The new limits appear immediately on the point's trend. **Remove override** at the bottom puts the point back on the default, with an **Undo** in case you didn't mean it.

> **TIP**
> Base your limits on history and judgement, not a guess. Each chart on the point page shows its **normal** band — the middle 80% of its running readings over the previous 30 days — and a reading's page compares it with the limit, the previous reading, and that band. Set warning and critical limits that sit clearly above (or below) normal.

Bearings: alarm on the defect bands

For a vibration point on a rolling-element bearing, go further. Assign the **Bearing** on the point's **Edit** page (**Shaft & bearing**), then on **Alarm rules** add a **Band rule** under **Bearing bands** — **Multiple of baseline** is the easy start (3× for warning, 6× for critical, against a baseline learned from the first five running readings). Click **Reset baseline** after any bearing change or repair.

Installation defaults

If every point of a type should behave the same — *"for any temperature point, warn at 55 and go critical at 75"* — that's an **installation default**, not an override on every point. Defaults are maintained by your Mechbase administrator (Setting up and managing your account); ask for one when you find yourself overriding the same type again and again. With sensible defaults in place, point overrides only exist for the genuine exceptions.

Keeping false alarms down

A route and its limits are only worth as much as people trust them. If an alarm fires every time a sensor twitches, teams stop paying attention — and the real faults get lost in the noise.

Mechbase avoids that by design: instead of tripping on a **single** reading that crosses a limit, it scores each reading (normal, minor, major), averages that over the last several readings, and moves the point's state only when the average crosses a threshold. So a sustained run of off readings escalates to **Warning** or **Critical**, while a lone spike shows as **Check** — a soft pattern forming — and no more.

Three things shape how well that works, and all are worth reviewing during a new point's first few weeks:

- **The limits** — from history and expertise, as above.
- **How many recent readings are averaged** — a longer window is steadier but slower to react; a shorter one reacts faster but tolerates less noise.
- **The baseline** — for relative and bearing-band rules, what "normal" means for this machine. If the first weeks of data were unrepresentative (a bad bearing, a misaligned coupling since fixed), relearn it from a better date: **Edit** the asset, and under **Baselines** use **Create from date...**.

> **NOTE**
> Treat the first weeks on a new point as a **learning period**. Watch the trend, see where the value really sits, and trim the limits until warnings land on real problems and nothing else. The full mechanism is in Measurements, trends, and alarms.

Quick reference

TASK	WHERE
New route	**Routes** (2) → **New Route** (N)
Route as you walk	**Routes** → **New Survey**
Edit a route's checklist	open the route → **Edit** (E) → **Add asset or zone**, **+** to add a check
Walk it · latest result · past runs	**Start route** (R) · **Latest report** (L) · **History** (H)
Check types	Pass / Fail · Numerical · Multi-check · Measurement
See a point's alarm rule	open the point → **Alarm rules**
Per-point limits	adjust the prefilled form → **Create override**
Back to the default	**Remove override** (with Undo)
Alarm methods	Threshold · Narrowband Envelope · Maintenance due
Direction	High (above is bad) · Low (below is bad)
Thresholds	Warning · Critical
Bearing defect alarms	assign the bearing on **Edit** → **Bearing bands** → **Band rule**
Relearn a baseline	asset **Edit** → **Baselines** → **Create from date...**
Installation-wide default	your Mechbase administrator — Setting up and managing your account

15 · Analytics & BI

Pulling Mechbase data into pandas, Power BI, Excel, and Grafana.

Get your installation's measurements, events, and inspection data out of Mechbase and into the tool you already use. This chapter is written for a **data analyst** — someone comfortable with spreadsheets, SQL, or a notebook, but not necessarily a programmer. Every recipe is copy-paste; replace the `<PLACEHOLDERS>`.

If you only read one section, read **Connect recipes** — that covers the 80%.

The one-paragraph version

Mechbase exposes your data three ways, all read-only:

1. **Live feeds** — paged JSON or CSV you walk page by page (good for a sync job or a quick pull).
2. **Bulk export** — you ask for a whole dataset as one CSV or Parquet file and download it from a link (good for a full load into a warehouse or notebook).
3. **OData feed** — the same data behind a single URL you paste into Power BI or Excel; the tool does the paging for you (the friendliest option if you live in Power BI / Excel).

Everything is scoped to **one installation** and needs an **API token** (except the bulk-export download link, which carries its own signature). Pick the recipe for your tool below.

> **The full API reference.** This chapter is a cookbook for the common jobs. For the complete, always-current list of endpoints, parameters, and response shapes, browse the interactive API docs at `app.mechbase.io/api/docs` (a self-hosted Mechbase serves them at `<your-host>/api/docs`).

Before you start: two things you need

1. Your installation ID

Every URL contains your installation's numeric ID. You can read it from the address bar while using Mechbase — it's the number straight after the host (`app.mechbase.io/2012/` → `2012`) — or ask an administrator. Throughout this chapter it's written `<INSTALLATION_ID>`.

The base URL for everything in this chapter is:

```
https://app.mechbase.io/api/installations/<INSTALLATION_ID>/analytics/
```

> **NOTE**
> examples use the hosted service at `app.mechbase.io`. A self-hosted Mechbase uses its own host — swap it in wherever you see `app.mechbase.io`.

2. An API token

There are two ways to get one:

- **Ask an administrator.** They create an **API Token** for you in the admin area (*API Tokens → Add*), scoped to your user and your installation — or a **service token** for an unattended integration. They hand you a string that looks like `xK3f...`.

- **Mint your own** with the `mechbase` command-line tool (it comes with the Python SDK — `pipx install mechbase`): run `mechbase login`, enter your email and password, and the token is stored in `~/.config/mechbase/config.json`. A token you mint yourself works for every installation you belong to. (Under the hood this is `POST /api/auth/token` with your email as `username`, your password, and a `device_id`.)

Either way, treat the token like a password.

You send it on every request as a header:

```
Authorization: Bearer <TOKEN>
```

> **One exception:** the bulk-export **download link** is pre-signed — the signature in the URL *is* the authorization. `pandas.read_csv(url)` or a browser download works with no header. More on that in Bulk export.

Live feed vs bulk export: which do I use?

YOU WANT TO…	USE
Pull the latest data into a notebook or a sync job, page by page	**Live feed**
Load an entire dataset (millions of rows) into a warehouse or file	**Bulk export**
Point Power BI / Excel at a URL and let it refresh	**OData feed**
Keep a warehouse table in sync with only what changed	**Live feed, incremental mode**

The three feeds and the export cover the same three datasets:

- **measurements** — one row per measurement reading.
- **events** — alarm/audit events.
- **inspections** — route inspection responses (walkthroughs).

Connect recipes

Python / pandas

Quick pull of the current page (CSV) — add `?format=csv` and pandas reads it directly. You still send the token:

```python
import pandas as pd
import requests

BASE = "https://app.mechbase.io/api/installations/<INSTALLATION_ID>/analytics"
HEADERS = {"Authorization": "Bearer <TOKEN>"}

# One page (up to 200 rows) of measurements as a DataFrame
r = requests.get(f"{BASE}/measurements/", headers=HEADERS, params={"format": "csv"})
r.raise_for_status()
from io import StringIO
df = pd.read_csv(StringIO(r.text))
```

Walk every page (JSON) — follow `next_cursor` until it comes back `null`. See Cursor pagination for why there's no row count:

```python
def pull_all(dataset, **filters):
    rows, cursor = [], None
    while True:
        params = {"cursor": cursor, **filters} if cursor else filters
        page = requests.get(f"{BASE}/{dataset}/", headers=HEADERS, params=params).json()
        rows.extend(page["items"])
        cursor = page["next_cursor"]
        if cursor is None:          # null cursor = last page
            break
    return pd.DataFrame(rows)

df = pull_all("measurements", status="major")   # only 'major' rows
```

Big load — for a full dataset, don't page the live feed; use a bulk export and `pd.read_parquet()` the signed link.

> **TIP**
> the analytics feeds are plain HTTP on purpose — the recipe above is all you need. The **Mechbase SDK** is for the operational API (readings, registry, routes), not these feeds.

Power BI / Excel (the OData feed)

This is the recommended path for Power BI and Excel — paste one URL and the tool handles the rest.

1. **Power BI:** *Get Data → OData feed*. **Excel:** *Data → Get Data → From Other Sources → From OData Feed*.
2. Paste the OData root for your installation:

   ```
   https://app.mechbase.io/api/installations/<INSTALLATION_ID>/analytics/odata/
   ```

3. When prompted for credentials, choose **Web API** / **API key** and set the key to:

   ```
   Bearer <TOKEN>
   ```

   (Include the word `Bearer` and a space before the token — it's the same `Authorization` header value as everywhere else.)
4. You'll see three tables — **Measurements**, **Events**, **Inspections**. Pick one, *Load*, and refresh whenever you like. The connector walks all the pages for you.

See OData details for filtering and sorting in the query.

Grafana

Use the **Infinity** data source (it reads JSON/CSV over HTTP).

- **URL:** a feed endpoint, e.g. `https://app.mechbase.io/api/installations/<INSTALLATION_ID>/analytics/measurements/?format=csv`
- **Parser:** CSV (or JSON — then set the *Rows / Root* to `items`).
- **Headers:** add `Authorization` = `Bearer <TOKEN>`.

Infinity pulls one page (200 rows). For dashboards over large ranges, point Grafana at a warehouse table you've loaded from a bulk export instead.

Prefer a client library? The Mechbase SDK

The recipes above talk to the analytics feeds by hand with `requests`, and that's the intended way: the SDK doesn't wrap the feeds, bulk export, or OData. What the **Mechbase SDK** gives you is a typed client for the *operational* API — reading assets, points, and individual readings, fetching a reading's

spectra, writing readings, syncing the registry, and driving routes. It's available for **Python, Go, Swift, and Kotlin** (https://github.com/arpedon/mechbase-sdk).

Install the Python client:

```
pip install mechbase
```

It uses the **same Bearer token** as everything else in this chapter and defaults to the hosted service — pass a `base_url` for a self-hosted installation:

```
from mechbase import Mechbase

client = Mechbase(token="<TOKEN>")          # base_url="https://your-host" if self-hosted
inst = client.for_installation(<INSTALLATION_ID>)

# one reading by the number in its web URL (…/measurementpoints/42/measurements/766/)
m = inst.measurements.get(42, 766)
spectra = inst.measurements.spectra(m.uuid)    # same arrays as the Spectra section below

# push a reading (the SDK also syncs the registry by external ID and drives routes)
inst.measurements.create(point_id=42, data={"rms": 2.3}, notes="manual reading")
```

The Python package also installs the `mechbase` **command-line tool**, handy for a quick look without writing code:

```
mechbase login                       # once; stores a token
mechbase use 2012                    # pick the installation
mechbase points -q "motor"           # find points
mechbase readings 42 -n 5            # latest readings on point 42
mechbase reading show 42 766 --peaks 10   # values, state, and the top spectrum peaks
```

Add `--json` to any command for machine-readable output.

For the read-only analytics feeds in this chapter, the copy-paste recipes above are the shortest path. Reach for the SDK when you're building an integration that also **writes back** to Mechbase — bulk measurement uploads, registry sync, or an edge agent (Pushing data into Mechbase). See the repository README for the full surface.

Cursor pagination

The feeds page with a **cursor**, not page numbers, and they deliberately **do not return a total row count** (counting millions of rows on every request is slow). The rule is simple:

> Read `next_cursor` from each response and pass it back as `?cursor=...`. When `next_cursor` is `null`, you've reached the end.

Each page holds up to **200 rows**. There's no "page 5 of 100" — just keep walking until the cursor is null. A cursor is tied to the mode and dataset it was minted for; don't reuse a measurements cursor on the events feed (you'll get a clean `422`, not wrong rows).

Incremental sync (only what changed)

To keep a warehouse table fresh without re-downloading everything, use **incremental mode**: pass `?updated_since=<ISO-8601 timestamp>`.

- The feed switches to **oldest-changed-first**, walking a "last updated" watermark. A reading that was backdated or re-finalized after your last run is re-emitted, so you never miss an edit.

- The feed is **at-least-once** — the same row can appear twice across runs. **Dedupe in your warehouse** by upserting on the row's UUID column:
  - measurements → `measurement_uuid`
  - events → `event_id`
  - inspections → `response_uuid`

Recipe:

```
# Save this from the last successful run (start with a far-past date the
# first time), then pass it next time.
last_run = "2026-06-01T00:00:00Z"
df = pull_all("measurements", updated_since=last_run)
# ... upsert df into your warehouse on measurement_uuid ...
# then store "now" as the next last_run
```

GOTCHA

don't combine `updated_since` with a `created_from` / `created_to` window for a changed-since sync. The created window also constrains incremental pulls, so a late-edited row whose reading time falls outside the window won't re-emit. Leave `created_*` unset for a complete sync.

Filtering

All feeds accept a `created_from` / `created_to` window (ISO-8601) on the measurement/event time. Beyond that, each dataset has its own filters:

DATASET	FILTERS
measurements	`asset_id`, `asset_external_id`, `point_id`, `point_external_id`, `transducer_type`, `status` (`good` | `elevated` | `minor` | `major`)
events	`action`
inspections	`route_id` (the route's UUID), `status` (`good` | `elevated` | `minor` | `major`)

Example: major-severity readings on one asset in May:

```
.../analytics/measurements/?
asset_id=42&status=major&created_from=2026-05-01T00:00:00Z&created_to=2026-06-01T00:00:00Z
```

Bulk export (a whole dataset as one file)

For a full load, ask Mechbase to build one file. It's a three-step flow:

1. Start the job (POST). Body picks the dataset, the format, and the same filters as the live feed:

```
job = requests.post(f"{BASE}/exports/", headers=HEADERS, json={
    "dataset": "measurements",     # measurements | events | inspections
    "format": "parquet",           # parquet | csv
    "filters": {"status": "major"} # same filters as the feeds (optional)
}).json()
status_url = job["status_url"]
```

2. Poll until it's done. The status response reports `pending` → `running` → `done` (or `failed` / `expired`):

```
import time
while True:
    s = requests.get(status_url, headers=HEADERS).json()
    if s["state"] in ("done", "failed", "expired"):
        break
    time.sleep(3)
```

3. Download the signed link. When `state` is `done`, the status response carries a `download_url`. **That link needs no token** — the signature in the URL is the authorization:

```
import pandas as pd
df = pd.read_parquet(s["download_url"])   # or pd.read_csv for a CSV export
```

Things to know:

- **Row caps:** a normal (interactive) token can export up to **100,000 rows**; a **service token** is capped at **10,000,000**. Ask an administrator for a service token for large warehouse loads.
- **Link lifetime:** the `download_url` signature is short-lived (**~15 minutes**). If it expires, just **poll the status again** — each poll mints a fresh link. The file itself lives for **24 hours** after it's built.
- **Concurrency:** up to **3** export jobs can run at once per installation. Beyond that you get a `429` — wait for one to finish.

Spectra (FFT arrays for one measurement)

Every vibration measurement with a stored waveform has spectra — served from a cache, or computed from the waveform on request. The feed row keeps the heavy arrays out (it gives you a `spectra_url`, and `spectra_available` tells you whether there's a waveform behind it); fetch them by URL when you need them:

```
GET .../analytics/measurements/<MEASUREMENT_UUID>/spectra/
```

Returns the frequency axis plus the velocity / acceleration / envelope arrays and the unit:

```
{
  "measurement_uuid": "…",
  "freq_hz": [0.0, 1.5, 3.0, …],
  "velocity": [...],
  "acceleration": [...],
  "envelope": [...],
  "unit": "mm/s",
  "source": "cache"        // "empty" only when the reading has no waveform
}
```

Raw signal files (`.dr`)

The spectra above are the *processed* arrays. For the **full raw waveform** behind a high-rate reading, download the measurement's Mechbase data-recorder file (`.dr`) — see A single reading's page. To read a `.dr` file in Python, use the **dr-files** package, which converts it into standard formats you can load straight into a notebook:

```
pip install dr-files
```

https://pypi.org/project/dr-files/ · https://github.com/arpedon/dr-files

OData details (filtering and sorting)

The OData feed lives at:

```
https://app.mechbase.io/api/installations/<INSTALLATION_ID>/analytics/odata/
```

with three entity sets: `Measurements`, `Events`, `Inspections`. It's a **read-only subset** of OData v4 — enough for Power BI / Excel, not the whole spec. Supported query options:

- **`$filter`** — comparisons `eq` / `ge` / `le` / `gt` / `lt`, joined with `and`, over the per-set fields below.
- **`$select`** — pick columns.
- **`$orderby`** — only on the time axes: `timestamp` (when it happened) or `synced_at` (when it was last updated; Measurements and Inspections only — Events order by `timestamp`), `asc` or `desc`. Any other field returns `501` (sorting on other columns would break paging).
- **`$top`** — a **total** row cap across all pages (a strict `$top=10` gives exactly 10 rows, not a page size).
- **`$count=true`** — adds `@odata.count` (the full filtered total).

Paging is automatic in Power BI/Excel; if you call OData by hand, follow `@odata.nextLink` until it's absent.

Filterable fields per set:

SET	FIELDS
Measurements	`measurement_uuid`, `asset_id`, `asset_external_id`, `point_id`, `point_external_id`, `transducer_type`, `status`, `timestamp`, `synced_at`
Events	`event_id`, `action`, `timestamp`
Inspections	`response_uuid`, `route_id`, `status`, `timestamp`, `synced_at`

Example — major readings since May, newest first, just two columns:

```
.../analytics/odata/Measurements?$filter=status eq 'major' and timestamp ge
2026-05-01T00:00:00Z&$orderby=timestamp desc&$select=measurement_uuid,value
```

There's also a service document (`/odata/`) listing the sets and a `$metadata` document (`/odata/$metadata`) describing the columns — Power BI reads both automatically.

Field dictionary

The columns you'll meet most often:

FIELD	MEANING
`status`	Severity label: `good`, `elevated`, `minor`, `major`. (Note: `elevated` exists — don't hardcode just the other three.)
`severity_zone`	The ISO/alarm zone the reading fell in (may be empty).
`metric_key`	What the measured value represents (e.g. the metric/channel name).
`value`	The numeric reading (may be empty for non-numeric rows).
`unit`	Unit of `value`.
`data`	A JSON bag of extra per-row fields that don't have a dedicated column.
`*_uuid` / `event_id`	Stable row identity — dedupe on this in your warehouse.
`timestamp`	When the reading/event happened.
`synced_at`	When the row was last updated in Mechbase (the incremental watermark). Measurements and inspections only — events never change, so their watermark is `timestamp`.
`point_sequence`	A measurement's number on its point — the number in the reading's web address (`…/measurements/766/`).
`transducer_type`	The point type code: `VB` vibration, `US` ultrasound, `TM` temperature, `IR` thermography, `PS` pressure, `FL` flow, `BL` boolean, `EI` current, `EV` voltage, `TC` tachometer, `CT` counter, `ST` state, `PR` production.
`*_external_id`	Your own external identifiers (asset/point), when set.

Known limitation: hard-deletes don't propagate

If a row is **hard-deleted** in Mechbase, it does not get announced through the feeds — incremental sync only ever *adds or updates*, it never tells you something disappeared. To stay exactly in sync, do a **periodic full refresh** (a full bulk export, or a feed pull with no `updated_since`) on whatever cadence your accuracy needs — e.g. nightly or weekly.

Quick reference

THING	VALUE
Base URL	`https://app.mechbase.io/api/installations/<INSTALLATION_ID>/analytics/`
Auth header	`Authorization: Bearer <TOKEN>`
API reference	`app.mechbase.io/api/docs`
Python / Go / Swift / Kotlin SDK	`pip install mechbase` · github.com/arpedon/mechbase-sdk
Command-line tool	`pipx install mechbase` → `mechbase login`
Dates in filters	ISO 8601 **with** a time zone (`…Z` or `+02:00`) — a date without one is rejected with `422`
Raw `.dr` files	`pip install dr-files` · pypi.org/project/dr-files
Feed page size	200 rows
End of feed	`next_cursor` is `null`
Dedupe keys	`measurement_uuid` / `event_id` / `response_uuid`
Export row cap	100k (interactive) / 10M (service token)
Export link TTL	~15 min (re-poll for a fresh one)
Export file TTL	24 hours
Max concurrent exports	3 per installation
OData root	`.../analytics/odata/` (sets: Measurements, Events, Inspections)

16 · Consuming Mechbase data

Keeping a warehouse, lake, or BI model in lockstep with one installation.

You're a system integrator, and you want an external store that always matches what's in Mechbase — without re-downloading the whole plant every night. This chapter is the integration playbook: how to do the first full load, how to pull only what changed after that, and how to stay honest when rows get deleted. It's all built on the same **read-only** analytics feeds an analyst uses — no new endpoints, no write access.

For the full endpoint list, parameters, filters, and field meanings, keep Analyst: Analytics & BI open alongside this chapter, and the interactive API docs at `app.mechbase.io/api/docs`. This chapter assumes those and focuses on the sync patterns.

Choose your path

Mechbase exposes the same three datasets — **measurements**, **events**, **inspections** — three ways. Pick the path that matches your destination:

- **Live feeds** — paged JSON or CSV you walk page by page. This is your delta channel; it's the only path that takes `updated_since`, so it's how you pull just-the-changes on a schedule.
- **Bulk export** — a whole dataset built as one CSV or Parquet file behind a signed link. This is your load channel; use it for the first full load and for the periodic full refresh.
- **OData feed** — one URL you point Power BI or Excel at. If your "external store" *is* a BI tool that speaks OData, you may not need a pipeline at all — the tool does the paging and refresh for you. See Analyst: Analytics & BI.

For a warehouse or lake, the pattern that works is a combination:

JOB	PATH
First load of a dataset into a fresh table	**Bulk export**
Every incremental run after that	**Live feed,** `updated_since` **mode**
Periodic full refresh to catch deletes	**Bulk export** again

> **NOTE**
> everything is scoped to **one installation**. If you sync several sites, run the whole pattern once per installation ID.

What you'll need

Two things, both covered in detail in Analyst: Analytics & BI:

- **Your installation ID** — the numeric ID in every URL. The base for everything here is `https://app.mechbase.io/api/installations/<INSTALLATION_ID>/analytics/` (swap the host for a self-hosted Mechbase).
- **An API token**, sent as `Authorization: Bearer <TOKEN>` on every request.

One integrator-specific tip: ask an administrator for a **service token**, not an interactive one. It raises the bulk-export row cap from 100,000 to 10,000,000 — the difference between "a demo" and "the real plant".

> **HEADS-UP**
> the bulk-export **download link is pre-signed** — the signature in the URL is the authorization, so don't send the Bearer header when you fetch the file. Everything else needs the header.

The first full load

Seed each destination table from a bulk export. It's a three-step flow — start the job, poll it, download the signed link:

```python
import requests, time, pandas as pd

BASE = "https://app.mechbase.io/api/installations/<INSTALLATION_ID>/analytics"
HEADERS = {"Authorization": "Bearer <TOKEN>"}

def bulk_export(dataset, fmt="parquet", filters=None):
    job = requests.post(f"{BASE}/exports/", headers=HEADERS, json={
        "dataset": dataset,            # measurements | events | inspections
        "format": fmt,                 # parquet | csv
        "filters": filters or {},
    }).json()
    while True:
        s = requests.get(job["status_url"], headers=HEADERS).json()
        if s["state"] in ("done", "failed", "expired"):
            break
        time.sleep(3)
    if s["state"] != "done":
        raise RuntimeError(s["state"])
    read = pd.read_csv if fmt == "csv" else pd.read_parquet
    return read(s["download_url"])             # signed link — no header
```

Run it once per dataset and write each result to its table:

```python
for dataset in ("measurements", "events", "inspections"):
    df = bulk_export(dataset)
    # ... load df into <dataset> table (create-or-replace) ...
```

Things to plan for, all detailed in the export reference:

- **Row caps** — 100,000 on an interactive token, 10,000,000 on a service token.
- **Link lifetime** — the signed `download_url` is short-lived (~15 minutes). If it expires, poll the status again; each poll mints a fresh link. The built file itself lives 24 hours.
- **Concurrency** — up to 3 export jobs per installation at once; a 4th gets a `429`.

Keeping in sync

After the first load, don't re-export the world every run. Switch to the live feed's **incremental mode** and pull only what changed since last time by passing `?updated_since=<ISO-8601 timestamp>`.

Two properties of this mode drive the whole design:

- It walks a **last-updated watermark** (`synced_at`), oldest-changed-first. A reading that was back-dated or re-finalised after your last run is re-emitted, so you never miss an edit. (Events never change

once written, so the events feed walks `timestamp` instead, and re-emits rows exactly on the boundary — upsert and they're harmless.)
- It's **at-least-once** — the same row can come back across runs. That's fine, because you're going to **upsert on the row's stable key**, never blind-insert:
  - measurements → `measurement_uuid`
  - events → `event_id`
  - inspections → `response_uuid`

The loop is: remember when you last ran, pull everything changed since, upsert it, then remember now.

```
# Walk the paged feed (see 15-cookbook-analytics-bi.md#python--pandas for pull_all).
run_started = "2026-07-01T00:00:00Z"          # the watermark from your last run
df = pull_all("measurements", updated_since=run_started)
# upsert df into your warehouse keyed on measurement_uuid, e.g.
#   MERGE INTO measurements t USING staged s
#     ON t.measurement_uuid = s.measurement_uuid
#     WHEN MATCHED THEN UPDATE SET ...
#     WHEN NOT MATCHED THEN INSERT ...
# then persist "now" as the next run's watermark
```

HEADS-UP

for a changed-since sync, leave `created_from` / `created_to` **unset**. The created window also constrains an incremental pull, so a late-edited row whose reading time falls outside the window won't re-emit — and you'll silently miss the edit.

When rows disappear

The feeds only ever **add or update** — a **hard-delete in Mechbase is never announced**. Your incremental job will happily keep a row that no longer exists.

There's no delete feed to subscribe to, so reconcile on a cadence instead: run a **periodic full refresh** — a fresh bulk export loaded into a new table, then swapped in — on whatever interval your accuracy needs (nightly, weekly). The refresh replaces the whole table, so anything deleted upstream simply isn't in the new copy.

Keep the cheap incremental job on its frequent schedule for freshness, and let the full refresh be the slower correctness backstop.

Scheduling and idempotency

A few rules keep the pipeline safe to re-run and easy to reason about:

- **Advance the watermark only after a run commits.** Store the timestamp you *started* the run, and write it back only once the upsert has landed. If a run crashes mid-way, the next run re-pulls the same window — harmless, because you upsert.
- **Upserts make retries free.** At-least-once delivery plus a `MERGE` on the `*_uuid` / `event_id` key means running the same window twice produces the same table. That's your idempotency guarantee — lean on it instead of trying to pull each row exactly once.
- **Don't persist a cursor between runs.** A cursor is tied to the mode and dataset it was minted for. Start each incremental run fresh from the watermark and walk `next_cursor` to `null` within that run — see cursor pagination.
- **Stagger against the export limit.** Only 3 exports run per installation at once. If your full refresh fans out across datasets or sites, cap concurrency at 3 and back off on a `429`.
- **Download promptly.** The signed link is good for ~15 minutes; fetch the file as soon as the state is `done`, or re-poll for a fresh link.

> **TIP**
> the feeds are plain HTTP and the loop above is all you need — the **Mechbase SDK** doesn't wrap them. Reach for the SDK for the operational API instead: one reading by its number (`inst.measurements.get(point_id, seq)`), a point's history (`inst.measurements.iter_for_point(...)`), or a reading's spectra. See the SDK section.

Syncing findings into a CMMS

When an analyst records a **finding** on a reading in Mechbase — the fault, its severity, the recommended action, and a due date (Measurements, trends, and alarms) — a maintenance system can pick it up as a work request and report back. The findings feed follows the same incremental pattern as the rest of this chapter:

```
GET  /api/installations/<ID>/findings/?since=<ISO-8601>
GET  /api/installations/<ID>/findings/<UUID>/
POST /api/installations/<ID>/findings/<UUID>/status/
```

- **Pull** with `?since=` the time of your last successful run: you get every finding created or changed at or after it, oldest first. Dedupe on `uuid`, and walk `next_cursor` as usual. Narrow with `status=open|acknowledged|closed`, `asset_id` / `asset_external_id`, or `point_id` / `point_external_id`.
- Each finding carries `fault_type`, `severity`, `recommended_action`, `due_by`, the spectral `evidence`, a `web_url` back to the reading, and its `work_order_ref`.
- **Report back** by posting a status: `{"status": "acknowledged", "work_order_ref": "WO-1234"}` when you raise the work order (the reference is required), then `{"status": "closed"}` when it's done. Findings only move forward — open → acknowledged → closed, or straight to closed — and re-sending the same status is harmless. Mechbase shows the state and work-order number on the finding.

Quick reference

JOB	PATH	KEY DETAIL
First full load	Bulk export	POST `/exports/` → poll `status_url` → read pre-signed `download_url`
Incremental delta	Live feed	`?updated_since=<ISO-8601>`; walk `next_cursor` to `null`
Catch deletes	Bulk export	Periodic full refresh, swap the table
Dedupe / upsert keys	—	`measurement_uuid` · `event_id` · `response_uuid`
Watermark column	—	`synced_at` (last updated in Mechbase); `timestamp` for events
CMMS findings	Findings feed	`GET /findings/?since=` · `POST /findings/<uuid>/status/`
Delivery guarantee	Live feed	At-least-once — always upsert, never blind-insert
Export row cap	Bulk export	100k (interactive) / 10M (service token)
Signed link TTL	Bulk export	~15 min (re-poll for a fresh one); file lives 24h
Max concurrent exports	Bulk export	3 per installation, else `429`
Full endpoint & field reference	—	Analyst: Analytics & BI · `app.mechbase.io/api/docs`

17 · Pushing data into Mechbase

Sending readings, syncing your asset tree, and running inspections from your own code.

The analytics chapters pull data out of Mechbase. This one is the write side: how a **system integrator** pushes measurements in, keeps the asset registry in step with an upstream system, and drives an inspection from a script. Everything here uses the **Mechbase SDK** — a typed client that wraps the same API and carries the same Bearer token. The examples are Python; Go, Swift, and Kotlin cover the same ground, with a few differences noted below.

If you only read one section, read **Install and construct the client** — every other recipe starts from the `inst` handle it builds.

The one-paragraph version

Construct one `Mechbase` client with your API token, scope it to an installation with `for_installation(...)`, and you get a handle (`inst`) with a resource for each thing you can write:

- `inst.measurements` — push one reading (`create`) or many at once (`create_batch`).
- `inst.sections`, `inst.zones`, `inst.assets`, `inst.measurement_points` — the registry, each with an `upsert` keyed on your own external ID so a sync job is safe to re-run.
- `inst.routes` — start an inspection, respond to items, and complete it.

Every call goes to **one installation** and carries the **same Bearer token** you use for the read-only feeds in the Analyst: Analytics & BI cookbook. Pick the recipe below.

> **The full API reference.** This chapter is a cookbook for the common write jobs. For the complete, always-current list of endpoints, parameters, and response shapes, browse the interactive API docs at `app.mechbase.io/api/docs` (a self-hosted Mechbase serves them at `<your-host>/api/docs`).

Install and construct the client

Install the Python client:

```
pip install mechbase
```

Construct a client with your token. `token` is keyword-only and required; `base_url` defaults to the hosted service, so pass it only for a self-hosted installation. The client is a context manager — use `with` so it closes cleanly:

```
from mechbase import Mechbase

with Mechbase(token="<TOKEN>") as client:     # base_url="https://mechbase.internal.acme.
com" if self-hosted
    inst = client.for_installation(client.me().current_installation_id)
    # ... every recipe below runs against `inst`
```

What's happening here:

- **`Mechbase(token=...)`** — the token is sent as `Authorization: Bearer <TOKEN>` on every request. The default `base_url` is `https://app.mechbase.io`; there's also a `timeout` (seconds, default `30.0`).

- **`client.me()`** — reads the caller's profile from `GET /api/me/`. For a token scoped to one installation, its `current_installation_id` is that installation, so you don't have to hard-code the numeric ID. A token you minted yourself with `mechbase login` covers every installation you belong to; then `current_installation_id` is simply the first (and is empty if you belong to none) — so pass the ID you want straight to `for_installation(...)`.
- **`inst`** — an installation-scoped handle exposing `.assets`, `.measurement_points`, `.measurements`, `.routes`, `.sections`, and `.zones`. Everything below hangs off it.

> **TIP**
> treat the token like a password. A Mechbase administrator creates it for you, scoped to your user and installation.

Push one measurement

Send a single reading against a measurement point with `inst.measurements.create(...)`. `point_id` and `data` are required; all arguments are keyword-only, and the rest have defaults:

```
m = inst.measurements.create(
    point_id=42,
    data={"rms": 2.3, "peak": 4.1},
    notes="hourly poll",
    idempotency_key="poll-2026-07-01T09:00Z-p42",
)
print(f"ok -> {m.uuid} sequence={m.point_sequence} status={m.status}")
```

The arguments:

- **`point_id`** — the numeric ID of the measurement point.
- **`data`** — a JSON object of the values you're recording (metric name to number).
- **`notes`** — a free-text note (default empty).
- **`session_id`** / **`timestamp`** — set these to group readings or to backdate a reading to when it was actually taken; when omitted, the server records the arrival time. A `timestamp` must carry its time zone (`2026-11-16T09:30:00Z` or `+02:00`); a bare local time is rejected with `422`.
- **`idempotency_key`** — sent as the `X-Idempotency-Key` header. Give each reading a stable key (as above) and a retry after a network blip won't create a duplicate. Reach for this whenever a poll loop might resend.
- **`file`** — pass a path or file object to attach a raw file; the call then multipart-POSTs to `/measurements/upload/` instead, with the JSON under a `payload` form field.

`create` returns a `Measurement` with `uuid`, `measurement_point_id`, `point_sequence` (the per-point running number — the same number as in the reading's web address), `data`, `status` (the resulting health state), `notes`, `created_at`, `external_id`, and `file_url`.

> **NOTE**
> the raw API also accepts an `external_id` in the body of a single reading. With one, the write is idempotent on it: `201` the first time, `200` for a re-send, `409` if that external ID already belongs to a reading on another point. The Python `create()` exposes `idempotency_key` instead; use `create_batch` (below) to send external IDs.

Batch upload, with result handling

Polling a hundred points one HTTP call at a time is slow. Send them together with `inst.measurements.create_batch(items)`, where `items` is a list of dicts. Reference the point by your

own external ID and tag each row with an `external_id` so you can tie the outcome back to your source record:

```
result = inst.measurements.create_batch([
    {"measurement_point_external_id": "PUMP-1-DE", "data": {"rms": 2.3}, "external_id":
"r-1"},
    {"measurement_point_external_id": "PUMP-1-DE", "data": {"rms": 2.9}, "external_id":
"r-2"},
])
print(f"created={result.created} duplicates={result.duplicates} errors={result.errors}")
```

`create_batch` returns a `BatchResult` with the headline counts — `created`, `duplicates`, `errors` — and a `results` list, one `BatchItemResult` per input row (in order). Each item carries `index`, `status`, `measurement_point_id`, `point_sequence`, `uuid`, your `external_id`, and a `detail` message. Walk the results to log anything that wasn't a clean insert:

```
if result.errors:
    for item in result.results:
        if item.detail:      # a per-row message means it wasn't inserted cleanly
            print(f"row {item.index} ({item.external_id}): {item.status} — {item.detail}")
```

HEADS-UP

the SDK forwards each item's dict through as-is — it doesn't validate the per-row shape locally. The keys shown here (`measurement_point_external_id`, `data`, `external_id`) are what the server expects; the API docs are the authority on the accepted fields.

Sync your registry by external ID

To keep Mechbase's asset tree in step with an upstream CMMS or asset register, use `upsert(...)` on each registry resource. `upsert` is an idempotent `PUT` keyed on your `external_id`: run it once to create the record, run it again to update it. That means the **whole sync is safe to re-run** — no "does this already exist?" bookkeeping on your side.

Build the tree top-down, referencing each parent by its external ID:

```
inst.sections.upsert(external_id="HALL-A", name="Hall A")
inst.zones.upsert(external_id="Z-1", name="Pump Row", section_external_id="HALL-A")
inst.assets.upsert(external_id="PUMP-1", name="Feed Pump 1", zone_external_id="Z-1",
                   equipment_type="rotating", machine_class="II")
inst.measurement_points.upsert(external_id="PUMP-1-DE", name="Drive End",
                               asset_external_id="PUMP-1", transducer_type="VB",
                               measurement_unit_code="mm/s")
```

The `upsert` fields per resource:

- **`sections`** — `name`, `external_id`.
- **`zones`** — `name`, `external_id`, plus a parent as `section_id` or `section_external_id`.
- **`assets`** — `name`, `external_id`, a parent as `zone_id` / `zone_external_id` or `section_id` / `section_external_id`, plus `equipment_type` (`rotating`, `electrical`, `static`, or `infrastructure`) and `machine_class` (ISO 10816 class `I`, `II`, `III`, or `IV`).
- **`measurement_points`** — `name`, `external_id`, a parent as `asset_id` / `asset_external_id`, plus `transducer_type`, `measurement_unit_code`, `location`, `body_segment`, `body_angle`, and `machine_class`. The `transducer_type` is a code: `VB` vibration, `US` ultrasound, `TM` temperature, `IR` thermography, `PS` pressure, `FL` flow, `BL` boolean, `EI` current, `EV` voltage, `TC` tachometer, `CT` counter, `ST` state. Leave `measurement_unit_code` out and Mechbase uses the type's default unit (`mm/s` for vibration).

> **HEADS-UP**
> these values are checked. Free text such as `equipment_type="pump"` or `transducer_type="accel"` is rejected with `422`.

Each resource also has `list`, `get`, `create` (a plain `POST`), and `update(id, ...)` (a `PATCH`) if you'd rather not upsert.

Deleting. `delete` takes the **numeric ID**, not the external ID, and returns a `DeleteResult` (`deleted`, `uuid`). Pass `cascade=True` to remove children too:

```
inst.assets.delete(1234, cascade=True)     # removes the asset and its points
```

> **NOTE**
> because the feeds never announce hard-deletes (see the Analyst cookbook), your own delete calls are the record of what you removed — log them.

Drive a route execution

You can run an inspection route from code end to end. In the SDK narrative this is a "survey", but the actual types and methods are `RouteExecution`, `routes.start`, and `execution.complete` — there's no method literally named `survey`.

Start an execution against a route UUID. `start` returns an `Executions` handle, with `.uuid` set to the execution's UUID:

```
from mechbase.errors import ConflictError

try:
    execution = inst.routes.start(route_uuid)
except ConflictError as e:
    # a route has at most one run in progress — join it instead
    execution = inst.routes.execution(e.body["running_execution_uuid"])
print(f"working on execution {execution.uuid}")
```

A route only ever has **one run in progress**. If someone (a technician on the phone, another script) already has one open, `start` fails with `409` — *"This route already has a running execution"* — and tells you its UUID so you can join it. (Your own run that hasn't recorded anything yet doesn't block you; it's simply replaced.) An empty run can be cancelled with `POST …/executions/<uuid>/cancel`.

On that handle you can:

- **`respond(route_item_uuid=..., data=..., notes="", photos=None, idempotency_key=None)`** — answer a defined route item. Pass `photos` to attach images (it switches to a multipart upload); pass `idempotency_key` (sent as `X-Idempotency-Key`) for safe retries. You get the item UUIDs from the route definition via `inst.routes.get(route_uuid)`, which returns a raw dict.
- **`add_field_item(label=..., item_type="pass_fail", data=None, zone_id=None, config=None, notes="")`** — author a new checklist item in the field, e.g. a fault the route didn't anticipate. `zone_id=None` makes it route-level; triage it to an asset later.
- **`triage(route_item_uuid=..., asset_id=...)`** — attach a response to a specific asset.
- **`complete()`** — finish the execution.

A field-authored item then completed:

```
resp = execution.add_field_item(
    label="Oil leak under pump",
```

```
    item_type="pass_fail",
    zone_id=None,                                    # route-level — triage to an asset
later
    data={"passed": False, "severity": "major"},
    notes="under coupling",
)
print(f"added field item -> response {resp.uuid} status={resp.status}")

final = execution.complete()
print(f"completed at {final.completed_at}")
```

`respond`, `add_field_item`, and `triage` each return an `ItemResponse` (`uuid`, `route_item_uuid`, `data`, `notes`, `status`, `created_at`). `complete` returns a `RouteExecution` (`uuid`, `route_uuid`, `status`, `started_at`, `completed_at`, `session_id`).

> **TIP**
> for a long or resumable field session, reattach later with `inst.routes.execution(execution_uuid)` instead of starting a new one.

Read, correct, and delete a reading

Every reading has a number on its point — the `point_sequence`, the same number you see in its web address (`…/measurementpoints/42/measurements/766/`). Use it to fetch or remove one reading:

```
m = inst.measurements.get(42, 766)          # GET  …/measurement-points/42/measurements/766/
print(m.status, m.data)

inst.measurements.delete(42, 766)           # DELETE the same URL
```

Deleting removes the reading and its files, and the point's state rolls back to its newest remaining reading. The `mechbase` command-line tool does the same from a terminal: `mechbase reading show 42 766` and `mechbase reading delete 42 766`.

Photos, files, and special point types

Files on a reading. To attach a file to a reading that already exists (a waveform, a report, a photo), `POST` it as multipart `file` to `/api/installations/<ID>/measurements/<MEASUREMENT_UUID>/files/`. Add a form field `uuid` that you generate, and a retry after a network failure returns the file that already landed instead of attaching a second copy. Files up to 50 MB; common image, document, data, and `.dr` / `.wav` types are accepted.

Photos of assets and points. `POST` an image (`file`, optional `uuid`) to `/api/installations/<ID>/assets/<ASSET_ID>/photos/` to **add** a photo to an asset's gallery, or to `/measurement-points/<POINT_ID>/photos/` to **replace** a point's reference photo. Both appear as `image_urls` when you read the asset or point back.

Running state. A reading sent to a **Production** point is an on/off sample, not a value: `data={"value": 1}` when the machine is running, `0` when it stops. Mechbase opens and closes running periods from them (a repeat of the current state counts as a duplicate) and totals the running hours.

On/off signals. A **State** point (`ST`) works the same way — `data={"value": 1}` for on, `0` for off — and is recorded as periods (a fault contact *On since 06:12*) rather than a trend.

Shaft speed with vibration. If your device records a tachometer alongside an accelerometer, send the pulse positions with the vibration reading as a `tach` object: `edges` (the sample indices of each pulse,

strictly increasing), `fs` (the sample rate), `ppr` (pulses per revolution, default 1), and `lag` (seconds, default 0). Mechbase then works out the measured shaft speed and the 1X and 2X amplitude and phase of the reading. Send it with the `.dr` waveform on `/measurements/upload/`; a missing or unusable tach never costs you the reading. (The Python `create()` has no `tach` argument yet — include it in a `create_batch` item or post the JSON yourself.)

Batch limits. `create_batch` takes up to 1,000 items per request.

Go, Swift, and Kotlin

The SDK ships Go, Swift, and Kotlin clients alongside Python in the same repository. They share the same shapes — the same client, the same installation handle, and the same resource methods — but they've grown at different paces:

LANGUAGE	INSTALL	NOTABLY ADDS	NOT YET
Python 0.3.0	`pip install mechbase`	`measurements.get` / `delete` / `spectra`, the CLI	photo uploads; `uuid` on `add_file`
Kotlin 0.2.6	`com.arpedon:mechbase-sdk:0.2.6`	`addPhoto` on assets and points, `imageUrls`, idempotent `addFile`	getting / deleting one reading
Swift	Swift Package from `https://github.com/arpedon/mechbase-sdk`	`addPhoto`, `imageUrls`, idempotent `addFile`, `Execution.cancel()` — on `main`, newer than the last tag (0.2.2)	—
Go 0.2.2	`go get github.com/arpedon/mechbase-sdk/go@v0.2.2`	the core surface	the newer features above

Learn the shapes here, then use whichever language your integration is written in. See the repository for each language's install and examples.

https://github.com/arpedon/mechbase-sdk

Quick reference

THING	VALUE
pip install	`pip install mechbase`
Import	`from mechbase import Mechbase`
Construct	`Mechbase(token="<TOKEN>", base_url=..., timeout=30.0)`
Default base URL	`https://app.mechbase.io`
Auth	`Authorization: Bearer <TOKEN>` (set for you)
Installation handle	`client.for_installation(client.me().current_installation_id)`
Push one	`inst.measurements.create(point_id=…, data={…})`
Push many	`inst.measurements.create_batch([{…}, …])` → `BatchResult`
Sync registry	`inst.assets.upsert(external_id="…", …)` (also sections / zones / measurement_points)
Delete	`inst.assets.delete(<numeric id>, cascade=False)` → `DeleteResult`
Run inspection	`inst.routes.start(uuid)` → `respond` / `add_field_item` / `triage` / `complete`
Route already running	`start` → `409` `ConflictError`; join `e.body["running_execution_uuid"]`
One reading	`inst.measurements.get(point_id, seq)` · `.delete(point_id, seq)`
Asset / point photos	`POST …/assets/<id>/photos/` (adds) · `…/measurement-points/<id>/photos/` (replaces)
Running state · on/off	`data={"value": 1 or 0}` on a Production or State point
Timestamps	ISO 8601 with a time zone, else `422`
Command line	`pipx install mechbase` → `mechbase login`
Reattach	`inst.routes.execution(<execution uuid>)`
Safe retries	`idempotency_key=…` → `X-Idempotency-Key` header
Languages	Python · Go · Swift · Kotlin
SDK repo	github.com/arpedon/mechbase-sdk
API reference	`app.mechbase.io/api/docs`

18 · Streaming from a device over MQTT

Publishing readings straight from your own hardware into Mechbase.

Get a device you built or bought to stream its readings into Mechbase without a person in the loop. This chapter is written for a **developer** wiring a sensor, gateway, or PLC up to Mechbase over **MQTT** — you publish a small JSON message to a topic, and it lands on a measurement point as an ordinary reading. Every example is copy-paste; replace the `<PLACEHOLDERS>`.

If your hardware is a **MaintNode** edge device, you don't need this chapter — MaintNode uploads to Mechbase on its own (over HTTPS, not MQTT). See MaintNode devices and automatic data. This chapter is for streaming from your *own* device.

The one-paragraph version

Your device connects to an **MQTT broker** and publishes a JSON message to a topic named for your installation, device, and channel. Mechbase subscribes to that broker, reads the message, looks up which **measurement point** the device's channel is mapped to, and creates a reading there. From that point on it's an ordinary reading — it trends, gets scored against the point's **alarm rule**, and drives the asset's health state. Three things make the rest of this chapter make sense:

- **One topic per channel** — `mechbase/{installation_id}/{device_id}/{channel}`.
- **One small JSON payload per message** — every payload carries a `ts` and a `type`, and there are five types.
- **A mapping binds the channel to a point** — you set that up once in Mechbase; without it, the message is rejected.

Before you start: what you need

Your broker details. Mechbase reads its broker from a per-installation setting, and the actual host and credentials aren't fixed in the product — **get them from your Mechbase administrator**:

- **Host** — the broker hostname. There's no default; it's configured per installation.
- **Port** — defaults to `1883`.
- **TLS** — off unless your broker is configured for it. If it is, your administrator will tell you the TLS port.
- **Username / password** — both optional. Some brokers require them, some don't. Use them only if your administrator hands them to you.

Your installation ID. The numeric ID of the installation these readings belong to. You can read it from the address bar while using Mechbase (`.../installations/<ID>/...`), or ask an administrator. It goes into the topic as `{installation_id}`.

A device ID. Any stable identifier for your device — a serial number or name. It goes into the topic as `{device_id}`, and you'll enter the same value in Mechbase when you map the channel to a point.

Connect to the broker

Any MQTT client works. The examples below use the standard **paho-mqtt** client for Python — the point is the connection parameters, not the library:

```
pip install paho-mqtt
```

```
import paho.mqtt.client as mqtt

BROKER_HOST = "<from your administrator>"
BROKER_PORT = 1883          # default; a TLS broker may use a different port
USERNAME    = "<from your administrator>"   # optional — omit if not required
PASSWORD    = "<from your administrator>"   # optional

client = mqtt.Client()
if USERNAME:
    client.username_pw_set(USERNAME, PASSWORD)
# client.tls_set()          # uncomment ONLY if your broker uses TLS
client.connect(BROKER_HOST, BROKER_PORT, keepalive=60)
```

- **Username and password are sent only if set** — leave them out entirely if your broker doesn't require them.
- **TLS is off unless your broker uses it.** Enable it only when your administrator says so.
- **`keepalive`** — pick whatever suits your device. Mechbase's own subscriber-to-broker connection uses a 60-second keepalive, but that's an internal Mechbase setting and is independent of your client's keepalive; you don't have to match it.

> **NOTE**
> Mechbase subscribes to `mechbase/{installation_id}/+/+` for your installation — one broker connection covers **all** your devices and channels. You just publish; Mechbase is already listening.

The topic format

Publish to:

```
mechbase/{installation_id}/{device_id}/{channel}
```

- **`mechbase`** — the topic prefix. `mechbase` is the default; your administrator will tell you if your installation uses a different one.
- **`{installation_id}`** — your installation's numeric ID.
- **`{device_id}`** — your device's identifier.
- **`{channel}`** — the channel number on the device.

Mechbase splits the topic on `/` and requires **exactly four parts** — anything else is ignored. The `{channel}` segment is read as an **integer when it's all digits**; a non-numeric segment is read as **no channel** (`null`), which is how you address a **single-channel** device (more in How a channel maps to a point).

```
INSTALLATION_ID = 42
DEVICE_ID = "pump-07"
CHANNEL = 1
topic = f"mechbase/{INSTALLATION_ID}/{DEVICE_ID}/{CHANNEL}"
```

The payloads

Every message is a JSON object with two required fields — **`ts`** (an integer Unix timestamp in **epoch milliseconds**, `minimum: 0`) and **`type`** — plus the fields for that type. Five `type` values are accepted: `waveform` , `spectrum` , `scalar` , `status` , `production` .

Publish the JSON as the message body:

```
import json, time

# ts is epoch milliseconds — multiply the Unix seconds from time.time() by 1000
payload = {"ts": int(time.time() * 1000), "type": "scalar", "value": 72.5, "unit": "C"}
client.publish(topic, json.dumps(payload))
```

Scalar — a single value

```
{"ts": 1719763200000, "type": "scalar", "value": 72.5, "unit": "C"}
```

Required: `ts`, `type`, `value` (a number). Optional: `unit` (a string). This is the everyday case — one reading, one number.

What Mechbase does with a scalar depends on the **type of the point** it lands on:

- On most points it's simply the reading's value.
- On an **ultrasound** point it's stored as the dB level.
- On a **State** point it's an on/off sample — any value above 0 is *on* — recorded as periods rather than a trend. Ideal for a fault contact or a run lamp.
- On a **Production** (running-hours) point it's taken as an **hour-meter total** that resets the running-hours count. To say *running* or *stopped*, send a `production` message instead (below).

> **NOTE**
> `type` must be **exactly one** of the five values above — it's a closed set. A message with any other `type` fails validation and is rejected, so nothing is stored.

Waveform — a time-domain signal

```
{"ts": 1719763200000, "type": "waveform", "fs": 5120, "samples": [...]}
```

Required: `ts`, `type`, `fs`, `samples`.

- **`fs`** — the sample rate (a number, must be greater than `0`).
- **`samples`** — an array of numbers, at least `1` and at most the waveform cap (see Size and sample caps).

Spectrum — a frequency-domain signal

```
{"ts": 1719763200000, "type": "spectrum", "fmax": 2000, "lines": [...]}
```

Required: `ts`, `type`, `fmax`, `lines`.

- **`fmax`** — the maximum frequency (a number, must be greater than `0`).
- **`lines`** — an array of numbers, at least `1` and at most the spectrum-line cap (see below).

Status — a heartbeat

```
{"ts": 1719763200000, "type": "status", "battery": 85, "firmware": "1.2.3"}
```

Required: `ts`, `type`. Optional: `battery` (integer `0`–`100`), `rssi` (integer), `firmware` (string).

> **HEADS-UP**
> a status message **does not create a reading**. It only updates the device's **last-seen** time on its active mappings — it's a heartbeat, not a measurement. Don't expect it to show on a trend.

Production — running or stopped

```
{"ts": 1719763200000, "type": "production", "on": true}
```

Required: `ts`, `type`, `on` (a boolean). Use it to tell Mechbase whether the machine is running. Repeats of the current state are **deduplicated** — send it whenever you like; only genuine changes are recorded.

Size and sample caps

Arrays are bounded so a buggy or hostile device can't drive an unbounded payload into a single reading:

FIELD	CAP	WHY
Waveform `samples`	`2,097,152` (2 × 1024 × 1024)	Samples persist to a `.dr` file at **2 bytes/sample** — this caps one message at ~2M samples, roughly **4 MB on the wire**.
Spectrum `lines`	`6,400`	Matches the analyser's FFT line cap, so a spectrum message can't bloat a row.

Both arrays also require **at least one** element. Exceed a cap and the message is rejected.

> **HEADS-UP**
> the spectrum `lines` array is stored but currently has no reader anywhere in Mechbase — it's bounded only to stop row bloat, and may be removed. If you need frequency-domain data on a trend, prefer scalar band values or send a `waveform` and let Mechbase's analysis path handle it.

How a channel maps to a point

A message only lands somewhere if the topic's device and channel are **mapped to a measurement point**. The mapping binds `(installation, device_id, channel)` to one point. You create it in Mechbase, not over MQTT:

1. Open the **measurement point** you want the readings to land on and click **Edit** (`E`).
2. Under **Sources**, click **Add source**, and leave **Source type** on **MQTT** — the same Sources list that holds MaintNode channels (Working with assets).
3. Enter the **Device ID** exactly as it appears in your topic's `{device_id}`.
4. Enter the **Channel** number. **Leave it blank for single-channel devices** — a blank channel is stored as `null` and matches a topic whose `{channel}` segment is non-numeric.
5. Click **Add source**. A device ID and channel can feed only one point; Mechbase tells you if that pair is already in use.

At ingest, Mechbase looks up the mapping for the message's device and channel:

- **Found** — it creates the reading against that point. The reading also carries the `sensor_id` (your device ID) and, when a channel is set, the `sensor_channel`, in its metadata.
- **Not found** — the message is **rejected** and logged as `No mapping for {device_id}/{channel}`. Nothing is stored. This is the first thing to check if a reading doesn't appear.

So the join between your hardware and Mechbase is one line on a point: *this point is fed by that device channel.* After that, everything about trends and alarms applies automatically, exactly as in MaintNode devices and automatic data.

Verifying it arrived

Publish a `scalar` message and confirm the loop end to end:

1. **Check the point.** Open the mapped measurement point in Mechbase — the new reading should appear on its trend with the value and timestamp you sent, and the point's **Sources** shows *MQTT , CH* with a live dot. On the edit page, the source row shows when it was last seen. (A `status` message won't add a reading; it only bumps the device's last-seen time.)
2. **No reading?** Check, in order — the topic has exactly four parts and the right installation ID and prefix; the **Device ID** and **Channel** on the point's source match the topic byte-for-byte; and, for arrays, you're under the caps.

For waveforms, the raw samples persist to a Mechbase data-recorder file (`.dr`). Download it from the reading's **Attachments** — see A single reading's page — and read it in Python with the **dr-files** package:

```
pip install dr-files
```

https://pypi.org/project/dr-files/ · https://github.com/arpedon/dr-files

If the `.dr` opens and the sample count and rate match what you published, the whole path is working.

Quick reference

THING	VALUE
Topic	`mechbase/{installation_id}/{device_id}/{channel}`
Topic parts	Exactly 4, split on `/`
Channel	Integer when all digits; otherwise `null` (single-channel)
Default prefix	`mechbase`
Default port	`1883` (TLS off unless configured)
Keepalive	`60` s on Mechbase's subscriber connection (internal; your client's is independent)
Auth	Username/password (both optional), TLS optional — from your administrator
Required in every payload	`ts` (integer, epoch milliseconds, `minimum: 0`) and `type`
Payload types	`waveform` , `spectrum` , `scalar` , `status` , `production`
Waveform	`fs` (> 0), `samples` (1 – 2,097,152 numbers)
Spectrum	`fmax` (> 0), `lines` (1 – 6,400 numbers)
Scalar	`value` (number), `unit` (string, optional)
Status	`battery` 0–100, `rssi` int, `firmware` str — all optional; heartbeat only, no reading
Production	`on` (boolean); repeats deduplicated
Waveform persistence	`.dr` file at 2 bytes/sample (~4 MB max message)
No mapping	Message rejected: `No mapping for {device_id}/{channel}`
Read `.dr` files	`pip install dr-files` · pypi.org/project/dr-files

Appendices

Quick references to keep beside you: the vocabulary, the keyboard shortcuts, the health states, and how Mechbase connects to other systems.

A. Glossary

TERM	WHAT IT MEANS
Installation	A single physical site Mechbase monitors — a plant, terminal, factory, or vessel. Its number appears in the web address (`/2012/`).
Section	A major area of an installation (e.g. *Utilities*).
Zone	A specific location within a section (e.g. *Pump Station*). Locations read as `Section / Zone`. Zones have their own page for documents and photos.
Zone type	A kind of zone (e.g. a room type), created by a room-list import. Documents added to a type appear on every zone of that type.
Asset	A piece of equipment (pump, compressor, panel…). Its **equipment type** is rotating, electrical, static, or infrastructure.
ISO 10816 class	The vibration severity class of a rotating machine (I–IV); it sets the default vibration alarm bands.
Measurement point	One specific thing measured on an asset, in one place — e.g. drive-end vibration. Named by its **location** (*Motor DE*).
Measurement / reading	A single value recorded at a point in time, automatically or by hand. Numbered per point (the number in its web address).
Transducer type	What a point measures: vibration, temperature, thermography, ultrasound, pressure, flow, current, voltage, tachometer, counter, boolean, state, production.
Trend	A point's readings plotted over time, with alarm limits shown. Vibration points show a grid of trends, one per metric.
Normal band	Where a value usually sits — the middle 80% of its running readings over the previous 30 days.
Waveform / spectrum	The full vibration signal behind a reading, and its frequency content. Opened in the **Inspect** panel.
1X / 2X	The vibration at once and twice shaft speed, with its phase — measured when a tachometer is wired.
Bearing	A rolling-element bearing from the library (or added by you), with its defect frequencies **BPFO**, **BPFI**, **BSF**, **FTF** in orders of shaft speed.
Baseline	A point's learned "normal", used by relative alarm rules. Can be cleared or re-learned from a date.
Finding	An analyst's recorded diagnosis on a reading — fault, severity, recommended action, due date — with a work-order status (Open, Acknowledged, Closed).
Alarm rule	How a point's status is judged — a method (threshold, narrowband envelope, maintenance-due), a direction, and Warning/Critical limits. An installation (or ISO) default, or a per-point override.
Bearing band rule	An alarm on a bearing's defect band, as an absolute value or a multiple of the baseline.
Health state	An asset or point's condition: Nominal, Check, Warning, or Critical. See Appendix C.
Silent	An asset whose sensors used to report but haven't for 30 days, or whose live feed has stopped.
Stale	A running-state or device signal that hasn't checked in recently.
Production / running state	Whether a machine is running (**On** / **Off**), from a tachometer, a run signal, or the vibration itself. Totalled as running **hours**.
Run detection	How a vibration sensor tells running from stopped, learned from its own history.
State point	An on/off signal (a fault contact, a run lamp) recorded as periods rather than a trend.
Needs attention	The list of every asset that isn't Nominal, most urgent first, followed by silent assets — on the home page and its own page.
Points page	Every measurement point on the site in one filterable table.
Hours page	Running hours for every asset that tracks them.
Mi[illegible]	A[illegible] meas[illegible]

B. Keyboard shortcuts

Mechbase is built to be driven from the keyboard. Page shortcuts are shown as small key caps on their buttons, and don't fire while you're typing in a field or a dialog is open.

Anywhere

KEY	ACTION
`J` · `⌘J` / `Ctrl+J`	Open or close the menu
`1`	Home
`2`	Routes
`3`	Needs attention
`4`	Hours
`5`	Points
`6`	Dashboards
`K`	Search
`Esc`	Back — up to the page above; also closes dialogs and the menu, and leaves a focused field first
`↑` `↓` `Enter`	Move through and pick from the menu, the installation picker, and pickers
`Enter` (`⏎`)	Submit the current form (sign in, save, create)

On a specific page

KEY	ACTION	WHERE
`E`	Edit	Section, asset, point, route
`Z`	Add a zone	Section page; an empty route
`P`	Add a measurement point	Asset page
`M`	Record a reading	Measurement-point page
`F`	Search	Readings list; Bearings
`N`	New route	Routes
`R`	Start / resume a route	Route page
`L`	Latest report	Route page
`H`	Route history	Route page
`⌘` / `Ctrl` + `Enter`	Post or save a note	Asset notes; reading and check notes

C. Health states

Every asset and measurement point is always in one of four states, coloured the same way throughout Mechbase — and always written out as a word as well, never shown by colour alone.

STATE	MEANING	WHAT TO DO
Nominal	Within normal limits; healthy.	Nothing.
Check	A soft, early signal that something is slightly off baseline.	Keep an eye on it.
Warning	Clearly degraded beyond normal limits.	Plan an intervention.
Critical	A severe deviation.	Act now.

How they look: in tables, Check, Warning, and Critical rows are tinted, and Warning and Critical rows carry a small ▸ flag in the margin; state chips put a solid ■ before the word. A hollow ☐ marks **Stale** data, and **Silent** marks an asset that has stopped reporting — both are about the data, not the machine.

States are derived automatically from measurements and inspections — never set by hand — and a short averaging window means they reflect a sustained *pattern*, not a single reading (Measurements, trends, and alarms). Every change is timestamped and written to the activity feed in plain language:

- **"Degraded to warning / critical"** — the condition got worse.
- **"Improved to check / warning"** — it got better but isn't fully healthy yet.
- **"Returned to nominal"** — it's healthy again.

An asset's state is the worst current state among its points and its open non-conformities. It feeds the **in alarm** count on the asset, the **Needs attention** list and its menu badge, the section cards' warning and critical counts, dashboards, and alert emails.

Times are shown in your own time zone, and recent dates read naturally — *Today*, *Yesterday*, *3 days ago*, *2 weeks ago*.

D. Connecting Mechbase to other systems

Mechbase is designed to fit alongside the rest of your toolset rather than stand apart from it:

- **MaintNode devices** — measure on a schedule, keep readings on the device, and upload them to Mechbase over HTTPS, landing on measurement points through their sources. Devices are claimed to an installation from their label's QR code (MaintNode devices and automatic data).
- **Modbus** — a MaintNode can read registers and coils from a PLC, drive, or gateway over TCP or serial (RTU), and feed them to points like any other sensor.
- **Your own devices** — stream readings over **MQTT** (Streaming from a device over MQTT), or push them through the **REST API** and **SDK** (Pushing data into Mechbase).
- **Analytics & BI feeds** — a read-only API pulls measurements, events, and inspection data into pandas, Power BI, Excel, or Grafana, with bulk export and an OData feed (Analytics & BI, Consuming Mechbase data).
- **CMMS** — two ways to connect a maintenance system:
  - the **findings API**, which lets any CMMS pull analysts' findings and report back a work-order number and status (Consuming Mechbase data);
  - a built-in **AIMMS** connector, set up by Arpedon, which mirrors your AIMMS asset structure into Mechbase, pushes inspection non-conformities back as AIMMS maintenance tasks, and keeps AIMMS operating-hours counters up to date.
- **Ultrasound surveys** — UE Systems DMS exports (`.dmsbak`) can be imported by Arpedon, bringing in the survey points, dB readings with their sound and image files, and the survey rounds as completed route runs.
- **Email** — alert, route-report, and export-ready notifications, chosen by each user (Getting started).
- **Raw-signal files** — every vibration waveform can be downloaded as the native `.dr` data-recorder file for analysis in specialist tools (Measurements, trends, and alarms).
- **Microsoft 365** — single sign-on with Entra ID (Setting up and managing your account).

Where to find each:

RESOURCE	WHERE
API reference (all endpoints)	`app.mechbase.io/api/docs` — a self-hosted install serves them at `<your-host>/api/docs`
Mechbase SDK (Python · Go · Swift · Kotlin)	`pip install mechbase` · github.com/arpedon/mechbase-sdk
`mechbase` **command-line tool**	`pipx install mechbase` → `mechbase login`
dr-files (read raw `.dr` signal files)	`pip install dr-files` · pypi.org/project/dr-files
Analytics & BI cookbook	Chapter 15

> **NOTE**
> Exact integration options, endpoints, and credentials depend on your deployment. For the AIMMS connector, a DMS import, or a custom integration beyond the documented API, contact **Arpedon**.