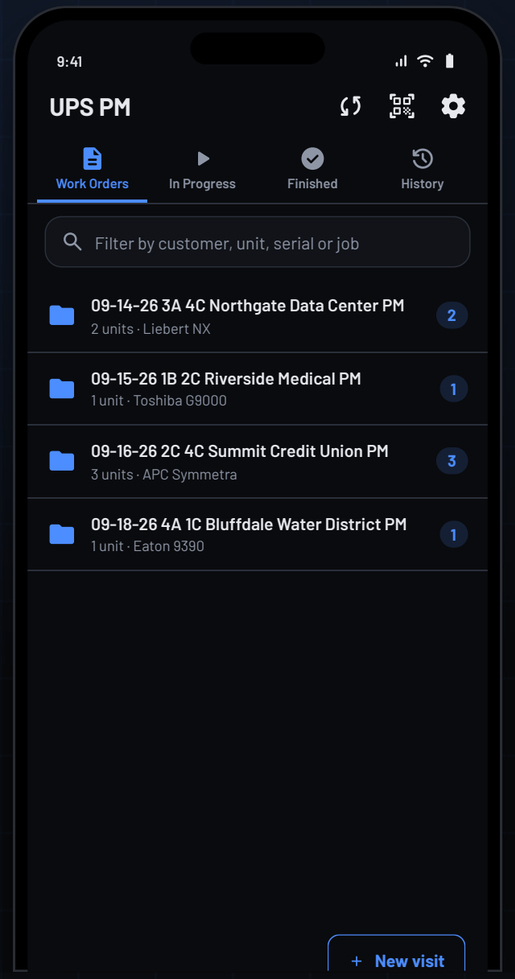
TECHCONNECT

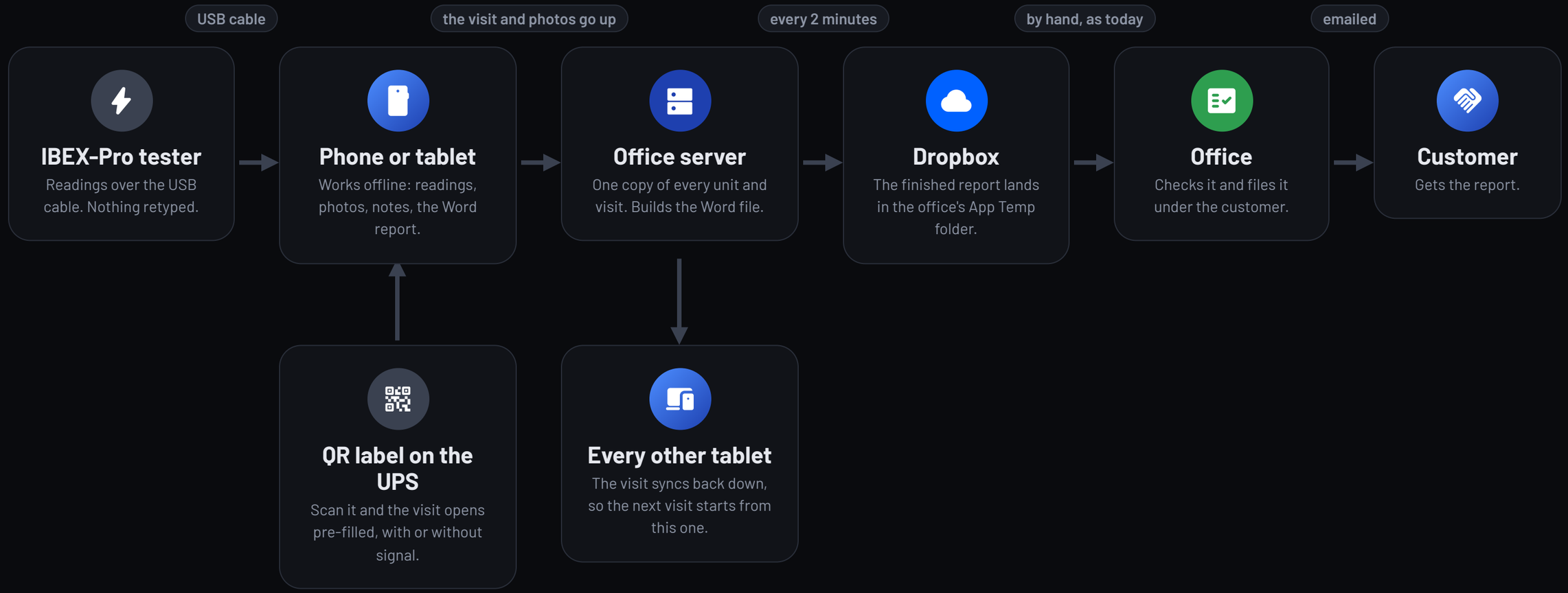
UPS PM

How the app works, from the first install to the report landing in Dropbox, and what keeps the code and the data safe.

A GUIDE FOR TECHNICIANS AND THE OFFICE · APP VERSION 87 · SEPTEMBER 2026



The whole system in one picture



Who does what

Technician

- Opens the job on the phone or scans the QR label
- Plugs in the IBEX-Pro, taps Read from tester, picks Data Transfer; nothing is retyped
- Takes photos; the display gets read
- Checks the drafted notes, edits, taps Generate
- Marks it finished with signal. Done on site.

Office

- Writes the job up in Dropbox as always
- Finds the finished report in App Temp
- Checks it, moves it on to Service Reports by hand
- Sends it on to the customer
- Hands out and replaces access tokens

The server, on its own

- Watches Dropbox every two minutes
- Turns work orders into visits for the phones
- Holds every phone visit plus an imported history of 7,000 past reports
- Delivers finished reports to App Temp
- Hands the AI key to phones, logs every request by name

Nobody retypes anything. The numbers flow from the tester and the photos into the document.

PART 1

Getting started

Install the app, connect it to the office, and do your first visit. About ten minutes the first time; a minute every time after.

Step 1: Install the app

1 **Open techpm.web.app**

Type it into any browser on the phone, or scan the code below. The page shows the current version and one Download button.

2 **Tap Download, then Install**

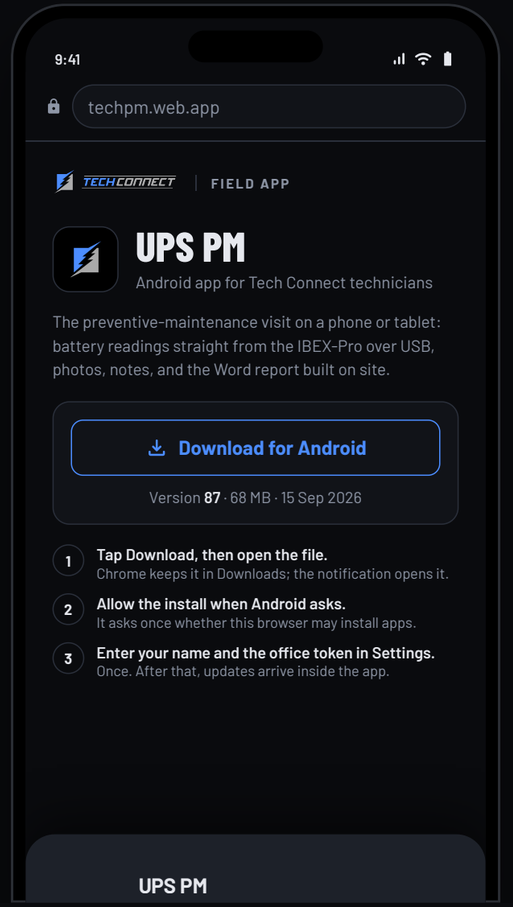
Android asks once whether this browser may install apps. Say yes; every Android app that does not come from the Play Store asks the same.

3 **Updates come from inside the app**

Whenever the app opens with signal it looks for a newer version, and the home screen shows a banner naming it. Tap the banner to download, open the file, allow the install: your visits stay where they are. Settings has Check for update too.

techpm.web.app

Scan it with the phone's camera to open the download page. No store account today; a Google Play listing is on the roadmap.



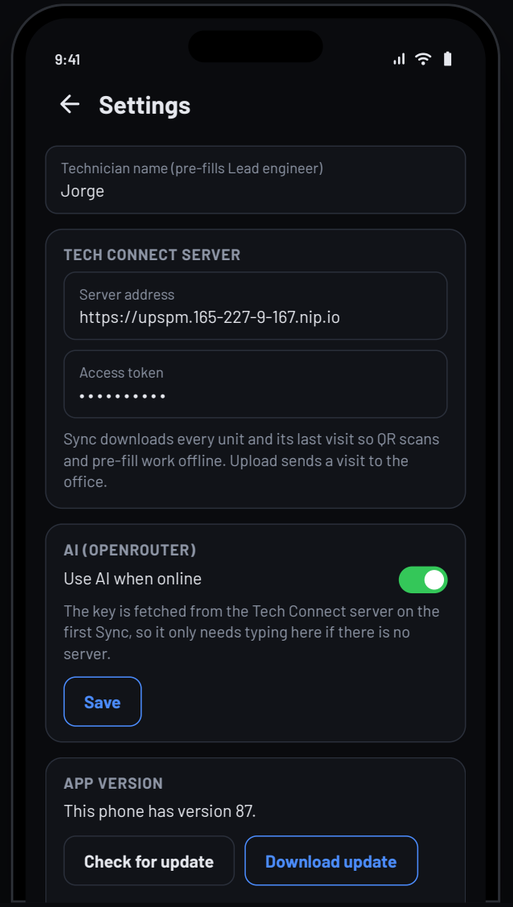
Step 2: Connect to the office

1. **Ask the office for your access token**
   A short code tied to your name, like `Jorge-8940`. Type it once in Settings. That is all the access you need: no sign-up form, no login screen, no password.

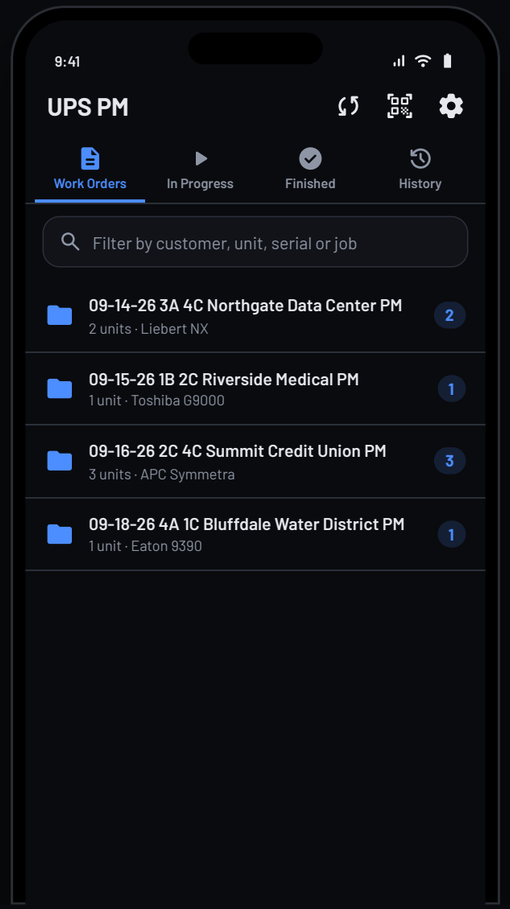
2. **Leave the server address alone, add your name**
   The address is filled in already. On a visit started from a work order the office's engineer names come first, so check the Visit tab.

3. **Save, then sync from the home screen**
   Tap the sync icon at the top, or pull the list down. Your work orders, plus every unit already on this phone, arrive in a few seconds. Nothing syncs by itself: you decide when.

Cloud accounts are the office's job, not yours. Dropbox is the only cloud link today, set up once on the server by the office. OneDrive and Google Drive are on the roadmap, not in the app yet.



The home screen: four tabs



Work Orders

Jobs the office has written up, one folder per job, named the way the office names them. Tap a folder to see its units.

In Progress

Visits you have started and not finished. They stay here with or without signal.

Finished

Visits you marked finished. With signal, the report goes to the office in Dropbox, where it is checked and filed under the customer.

History

Earlier reports for the units on this phone, grouped by customer. The battery outlook is built from these.

Pull down or tap the sync icon to sync. The filter box matches customer, nickname, make, model, serial, job number or address within the tab you are on.

No signal? Keep working

Everything on the phone

Units, history, work orders and your visits live on the phone after a sync. Readings, the tester, the checklist, notes and the Word report all work in a basement with no bars.

AI work waits in a queue

Photos to read and notes to draft are queued. The moment the phone is back online they run by themselves, and the suggestions appear on the visit.

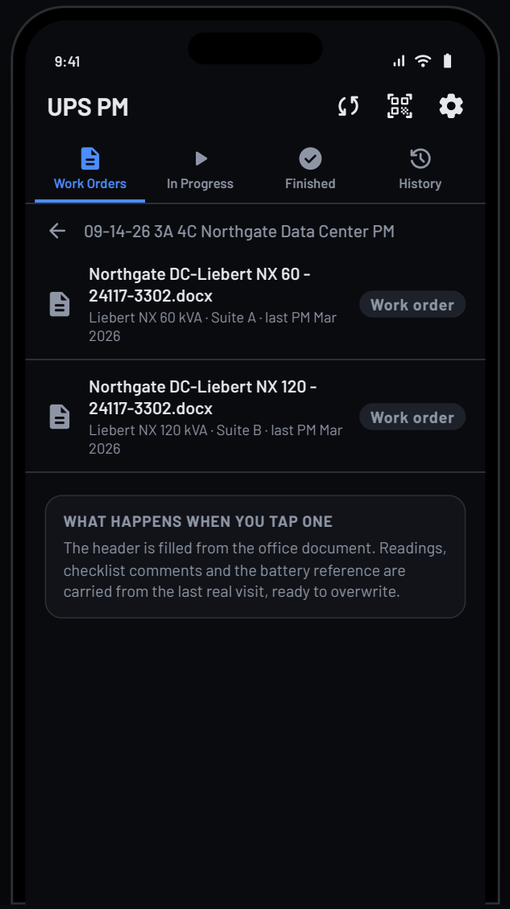
Upload when you can

Marking a visit finished uploads it and the photos, so do that with signal. If you marked it finished in a basement, it goes up at the next sync, or tap **Send to the office now** on the Report tab's **With the office** card once you are back online. Until then the office does not have it.

Sync is small and fast. The server sends only the units with current work orders plus the ones already on this phone, and only what changed since last time. A fresh sync is a few seconds, not minutes.

On site, the only thing that needs signal is the AI. Sync and sending to the office wait until you have it.

Step 3: Start a visit



Open the job folder and tap the unit

A new visit starts from the document the office wrote; the document itself is never changed. Customer, site, contact, job number and unit details are already there.

Or scan the QR label on the UPS

The label starts today's visit for that unit, pre-filled from its history, even with no signal. The phone's own camera app works too: the link opens UPS PM.

Carried over from the last real visit

- Readings and the parameter table, so you overwrite instead of retyping
- Checklist comments word for word, the way the office expects them
- The battery reference and how many strings and jars to expect
- Carried readings and battery strings are flagged on the Report tab until you replace them

QR labels: one sticker per UPS

What is on the label

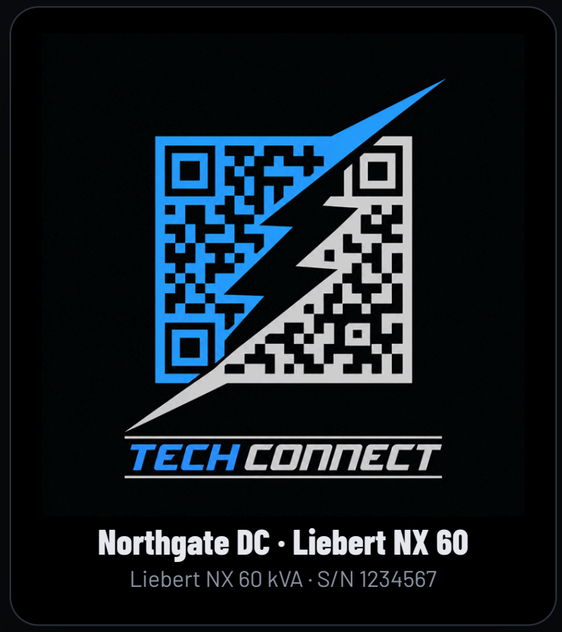
The QR code itself holds only a link with the unit's id, like `upspm://unit/u-c447806c8f9e`; anyone who scans it without the app sees an address and nothing else. The printed label adds the customer, the unit and the serial so a technician can find the right cabinet.

Printing them

The office makes them with the laptop tool: `pmtool qr-sheet` writes one image with every unit at a customer, `pmtool qr` one label. Print, stick it on the UPS.

Scanning

The QR button on the home screen scans with the camera, or reads a QR out of a picture already on the phone. If the unit is not on the phone yet, it is fetched from the server when you have signal.



Inside a visit: seven tabs, in the order of a real visit

01



Visit

Site, contact, unit and the job number. Mostly pre-filled.

02



Photos

Take photo, sorted by the AI: nameplates, displays, issues, infrared.

03



Batteries

Three sub-tabs: battery testing, the outlook, and the strings.

04



Readings

Input, output, battery and load values. Typed, or read from a photo.

05



Checklist

The inspection list with comments carried from last time.

06



Notes

Generate all drafts from the numbers, or dictate. You edit.

07

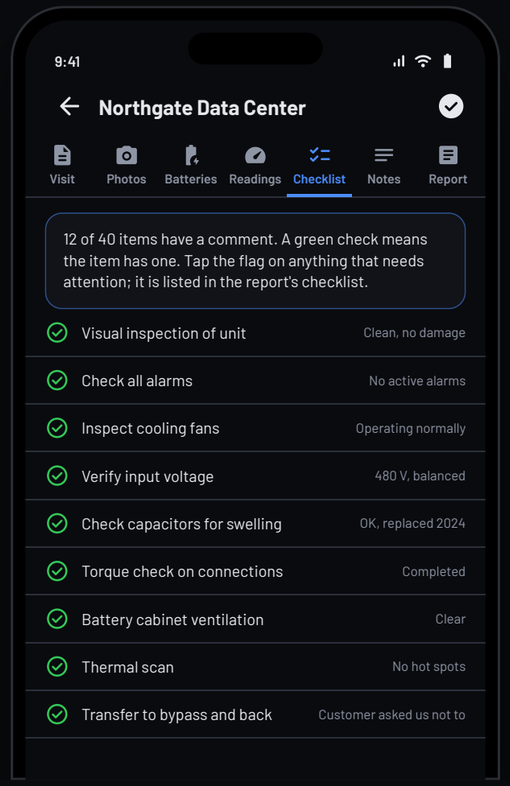


Report

Pre-flight, hours, Generate Word report, Mark report finished.

Edits save about half a second after you stop typing. Leave with the arrow at the top left rather than the phone's Back gesture, which can drop the last edit.

Visit, Readings and Checklist (since v112: Visit, Photos, Batteries, Readings, Checklist, Notes, Report)



Visit

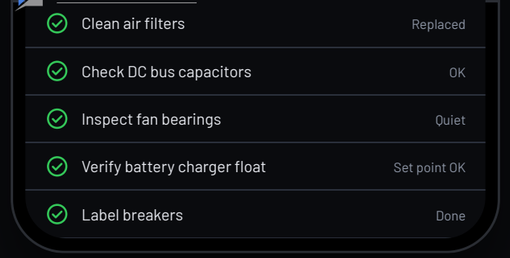
Customer, site, contact, job and contract numbers, the engineers on site, the unit's make, model, serial and nickname. From the work order; correct anything that changed. The **Previous visit** card at the top names the last report on file and its **View previous report** button opens it, as the office wrote it. Everything below the header starts from that report, and the phone fetches it from the office first if it does not have it yet. A **Next** card names the first thing still missing; tap it and the right tab opens. Close the app mid-visit and it reopens on the same visit.

Readings

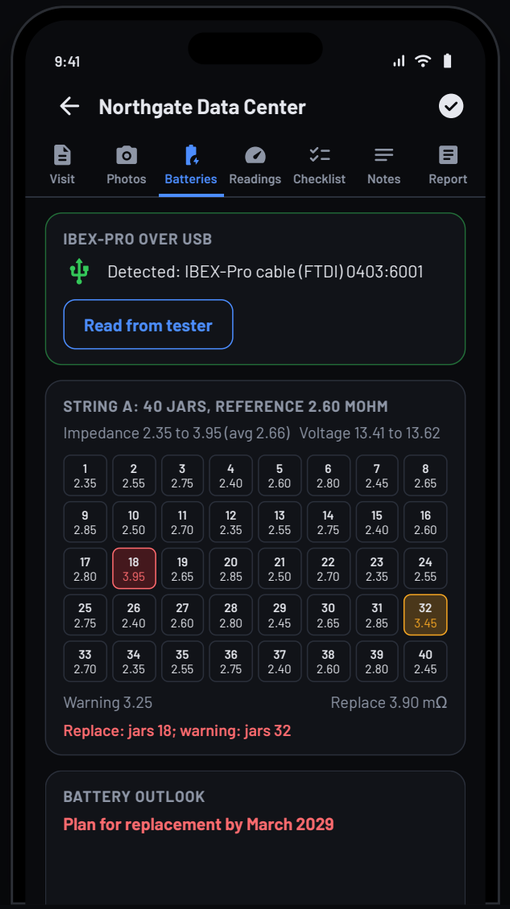
The unit's parameter table, laid out the way the office template prints it for that model. Last visit's values sit in the fields so you overwrite only what moved. A photo of the display can fill them for you. Under each value a line says where it came from: carried from last visit, typed today, or read from a photo (with the photo beside it), and a **Confirm** tap turns a carried value into today's. A reading the unit does not show is marked **Not tested** so the report says so instead of reprinting last year's.

Checklist

Forty inspection items with their comment from the last report, word for word, because the office keeps those constant. Change a comment where something differs today; tap the **flag** on anything that needs attention, so it is listed in the report.


Clean air filters
Replaced
Check DC bus capacitors
OK
Inspect fan bearings
Quiet
Verify battery charger float
Set point OK
Label breakers
Done

Batteries: plug in the tester, the numbers appear



1

Plug the IBEX-Pro into the phone

The USB cable that came with the tester. Open the Batteries tab on **01 Battery testing**: it notices the cable and shows **Read from tester**. The first tap asks once for USB permission.

2

Pick Data Transfer on the tester

The tester pushes every jar it measured, 4,100 bytes in a couple of seconds however many jars there are. Nothing to type. A saved file from the tester's software imports the same way.

3

Read the verdict

03 Strings grades every jar; **02 Battery outlook** has the replacement date and a chart per string. Amber is at or above the warning line, one and a quarter times the reference. Red is at or above the replace line, one and a half times. A jar with a bad clip contact shows as open-probe: retest it before the report, because the app will not build around a bad reading. Chips over the grid filter it to **Over a line**, **Re-test** or **New**. Tap a jar to see that position's readings on every visit; **This jar was replaced today** records the swap, and the old readings stay with the old jar.

The reference impedance comes from the last real report for these batteries. You can change it on this tab. The warning and replace factors, the service life and the battery models come from one assessment profile the office keeps on the server; every report names the profile that judged it.

Battery outlook: which tests count

Only the batteries in the cabinet today

The install date on the report header starts the clock: tests before it belong to the old set and are set aside. With or without a date, the numbers mark a change too: a different number of jars, a reference more than 15% different (another battery model), or a string average falling by more than a fifth between two tests (fresh batteries read low; a worn set never recovers that much). The line starts again from there.

Every test of that set, and nothing else

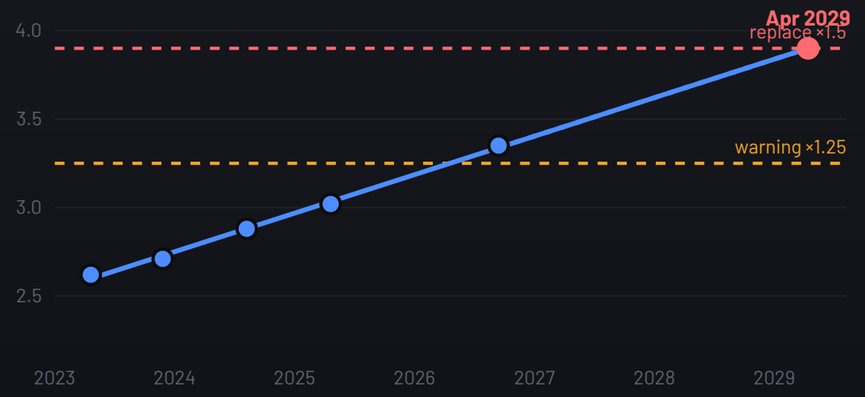
Every report on file for the unit since then, phone visits and the imported archive alike. A visit that only carries last time's numbers is not a test, and neither is an office document with a pasted chart. Tests of batteries that are gone never pull the line.

Every jar, minus the bad clips

Inside a test, a jar with no voltage or an off-scale reading is a clip off the post: it is listed for a re-test and never judged. A jar reading several times the rest of its string is kept off the trend chart so the chart stays readable, but it still counts. One failed battery in a sound string (the rest near the reference) is its own action, replace that battery now, and the set is dated from the rest of the string; two or more over the line is a worn set. The outlook says how many jars stand over each line, and repeats any battery finding from the checklist as evidence in its own right.

The app prints the level of the highest jar, the office history behind the date, how many tests it used and how well they fit a line (R^2), so the outlook can be checked rather than believed. Tests it set aside are counted too.

Highest jar per test, string A mΩ



Five tests of the current set. Reference 2.60 mΩ. The straight line crosses the replace line in 2029; at 1.29 × reference today the office history puts the plan sooner, Mar 2027. Example data.

Battery outlook: the arithmetic

The level today

The highest jar in the string, divided by the reference R the office set for that model: `level = z ÷ R`. Warning at 1.25, replace at 1.5. This one number carries most of the forecast.

What the office's own history says about that level

Every string the office ever tested was followed forward from each test: how many months until it was first seen over the warning line and the replace line. Grouped by level, that is the table on the right: 4,209 tests on 718 strings. The plan date is the first-quartile month among the strings that later crossed. Strings still under the line at their last test are left out, so counting every string far fewer than a quarter had crossed by then. A higher level never gets a later date: each level uses the shortest months of its own band and every band below it.

Why not the straight line

A least-squares line through the earlier tests is printed too, with its slope and R^2. Tested against the archive, it called the crossing later than it came in 54% of real cases, by a median of three years: impedance creeps for years and then turns up. So the trend is the earliest of the line, a curve from the last reading and the last interval's slope, and it may pull the date earlier, never later.

Age

Install date plus the service life in the office's profile, 6 years unless the office sets another. Counting sets still in service, a quarter were replaced by about 4 years. The plan date, Budget by, is the earliest of the three clocks: when the set is likely to reach the replace line or be replaced.

When it refuses to guess

A highest jar under its reference gives no impedance date; age alone speaks. Fewer than two tests, or tests under six months apart, give no line; one test still gives a date from its level.

The office's history, by level of the highest jar

LEVEL (× R)	TESTS	OVER THE WARNING LINE: A QUARTER BY / HALF BY	OVER THE REPLACE LINE: A QUARTER BY / HALF BY
1.0 to 1.1	631	12 mo / 19 mo	18 mo / 28 mo
1.1 to 1.2	371	6 mo / 9 mo	9 mo / 17 mo
1.2 to 1.3	169	6 mo / 7 mo	9 mo / 13 mo
1.3 to 1.4	72	over it	7 mo / 9 mo
1.4 to 1.5	58	over it	5 mo / 6 mo

worked example	highest jar 3.35 mΩ, reference 2.60: level 1.29
plan by	Sep 2026 + 6 months → Mar 2027 (the band's own is 9; 1.2 to 1.25 has 6, and a higher level never gets a later date)
half by	Sep 2026 + 13 months → Oct 2027
the line, for comparison	0.218 mΩ a year → Apr 2029, later than the history says: not used
age	installed Apr 2023 + 6 years → Apr 2029

The reading, the reference, this table and the rule that set the date are all printed, so the date can be checked by hand. The table is rebuilt from the archive as it grows.

Photos: one Take photo button, sorted by the AI



Take photo, sorted by AI

Take photo on the Auto-sort card opens the camera and it stays open between shots, with a count and a torch. Tap the X when you are done. Shoot everything in one go: the AI works out which photos are screens and reads them; the rest go in the report as nameplate, cabinet, infrared, issue or other. Tap any photo to change what it is. Every photo is also copied to the phone's Gallery.

The display gets read for you

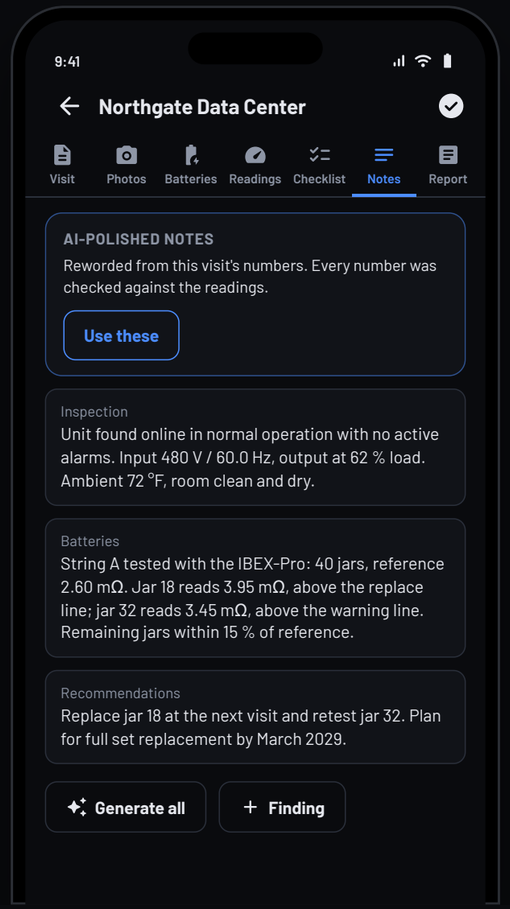
A photo of the UPS screen is read by an AI vision model. Its values arrive on the Readings tab as a card, **Read from photo (6 values)**, with Accept all or Dismiss. Only fields that are blank or still last visit's are ever filled by themselves; a value you typed is never replaced.

Sort by hand when you prefer

A **Sort by hand** switch on the Auto-sort card brings back one tile per kind (display, nameplate, battery cabinet, issue, infrared, other), and the phone remembers the choice. **Add photos from phone** takes pictures from the Gallery, Files or an infrared camera; the AI sorts those too. Only display and unsorted photos go to the AI; nameplate, cabinet, issue and infrared photos never leave the phone until the report.

No signal? Photos wait in the queue and are read the moment the phone is back online, up to five tries each. You can keep working.

Notes: drafted from the numbers, finished by you



The one rule the writer follows

It may only use numbers that are in the visit. Every number in a draft is checked against the readings before you see it. A draft with an invented number is thrown away and the plain draft is used instead.

A plain draft is built from the numbers first: Inspection, Measurements and adjustments, Batteries, Checklist and Event log, plus Recommendations and Action required. The Battery outlook is arithmetic, not prose. The AI only rewords the draft in the office's tone, at most four sentences a section.

Edit anything, in the right order

Generate Word report drafts the notes only while they are blank, still last visit's, or the app's own earlier draft; once you have typed in any note field it leaves all of them alone. **Generate all**, at the top of the Notes tab, starts every section over from the numbers, so use it before you type, not after.

Findings

Tap **Add finding** for anything worth a follow-up quote. Immediate goes into Action required, Next visit into Recommendations, and the office sees every finding on the web page. A finding carries over to the next visit until someone marks it **Resolved**, **Quoted** or **Deferred**, with a note; the report says which are still open and which were resolved this visit.

Notes for the AI, answered

Each section has a **No issues** switch; off, a box takes what you found and the AI states it first. The **Notes for the AI** box takes tone and instructions, typed or spoken with **Dictate notes**; the field shows when it was last sent and the AI's one-line answer on what it did with it, or why it could not, such as a runtime that is not on the visit.

Ask about this visit

A second technician who has read the file

The question mark in the visit's top bar opens a panel that knows the open visit, the last report on file, the unit's history and outlook, the checklist and the photos read so far. Ask it anything about them, by button or in your own words.

One rule, checked on every answer

Every number it states must be on the visit. A number that is not is listed in red under the answer, so nothing is taken on trust. Ask for a figure that is not on file and it says so, and where it would be entered.

It proposes, you decide

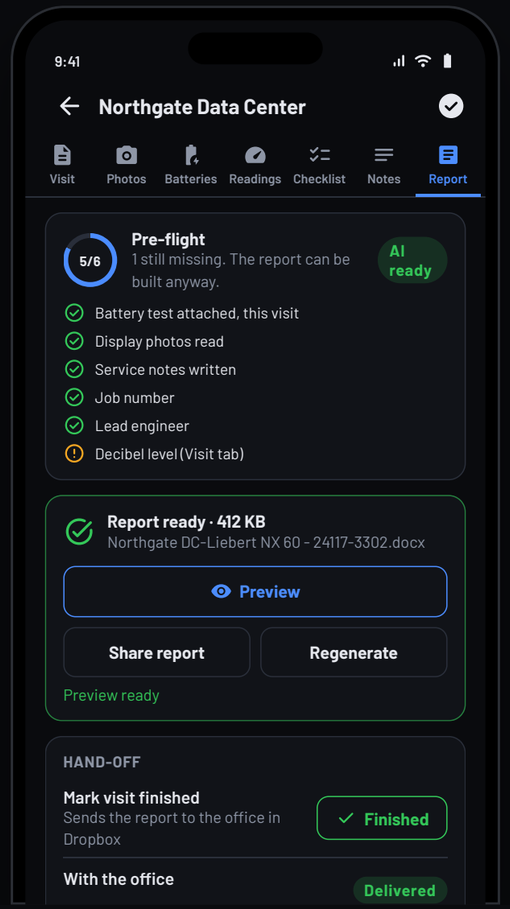
It cannot change the visit. Wording it suggests is for you to copy. The conversation stays with the visit on the phone and can be cleared. About a cent a question, through the office server, logged under your name.

The five buttons

- **What changed since last time?** Readings, load, impedance, strings and checklist answers that moved, and what did not.
- **What needs attention before I generate?** Anything still missing or still carried over, anything over a line, anything in the checklist that reads as a finding.
- **What did we tell them last time?** Last report's recommendations and action required, and whether today's numbers show they were acted on.
- **Explain the battery outlook.** Which jar sets the level, what the office's history says, what the plan date means and does not mean.
- **Suggest tighter wording for the notes.** Shorter, plainer paragraphs with every fact kept, for you to copy.

YOUR OWN QUESTION GOES IN THE BOX UNDERNEATH.

Report: one tap, a Word file, delivered



1 Generate Word report

The button at the foot of the Report tab ("Build visit report"). Reads any waiting photos, drafts the notes, fills the runtime, then builds the Word document with the charts. It is the office's own template, so the customer sees no change. Tap Preview and the pages come back laid out by Word itself, through the office server, so what you see on the tablet is what the office prints.

2 Read the pre-flight list

Green ticks are done, amber is still missing (the report can be built anyway), red stops the build: no battery test, strings still last visit's, open-probe jars, unsorted photos. Each line names the reading or photo, and a tap opens its tab. Generate anyway overrides red.

3 Mark visit finished

Tap **Mark report finished** on the Mark visit finished card; it turns green and reads Finished, and a second tap reopens the visit. With signal, the visit and every photo go to the server, which builds the same report and sends it to the office in Dropbox. The **With the office** card, and the pill on the home row, say exactly where it stands: waiting for signal, uploading, received, or delivered to Dropbox. A change after delivery needs a one-line reason and goes out as a numbered revision; the office's own edits are never overwritten.

A copy of every report and the tester's raw data stays on the phone under Files › Documents › UPS PM (Android 10 and newer). Share works with email, Teams or anything else on the phone.

Where the files go: the Dropbox loop

1 Office writes it up

In Dropbox under Technicians › Work Orders › the job folder, like today.

2 Server picks it up

Within two minutes it is a visit on the server. It reaches the phone's Work Orders tab at the next Sync.

3 Technician does the PM

Opens the folder, tests the batteries, takes photos, generates the report on the phone.

4 Report lands in App Temp

Marking it finished uploads the visit; the server builds the Word file and drops it into Technicians › App Temp under the job folder.

5 Office checks and files it

Moved on by hand to Service Reports, as before. Reports filed under Customers are what the server reads back as history.

The rule the server follows: it only reads Customers and Work Orders, and only ever writes inside App Temp. It never overwrites or deletes a file. The Dropbox history stays exactly as it is.

Technicians › Work Orders — server reads

```
09-14-26 3A 4C Northgate Data Center PM
  └ Northgate DC-Liebert NX 60 - 24117-3302.docx
  └ Northgate DC-Liebert NX 120 - 24117-3302.docx
```

Technicians › App Temp — server writes

```
09-14-26 3A 4C Northgate Data Center PM
  └ Northgate DC-Liebert NX 60 - 24117-3302.docx
  └ never overwritten, never deleted
```

Customers › … › Service Reports — server reads

```
N › Northgate Data Center › Service Reports
  └ Northgate DC-Liebert NX 60 - 23891-3302.docx
  └ 7,000 reports, the history behind every outlook
```

The office keeps working in Dropbox exactly as before. The server watches two folders and delivers into one.

pmtool: the same engine on the office laptop

The same workflow as the phone, on the office laptop, for bulk jobs. Unzip it, double-click **UPS PM Tool.cmd**, and a window opens with the menu. Nothing to install: Java is inside.

Import the archive

Reads old Word reports into visits: the header, the parameter table, the checklist, the battery readings. That is how 7,000 reports became history.

Receive the tester on the laptop

The IBEX-Pro on a COM port, or its saved files.

Labels, reports, samples

QR sheets per customer, reports with trend charts in any style, and the Dropbox link the office does once.

```
UPS PM tool | data folder: Documents\UPS-PM-Data | 11 units, 166 visits
pmtool> units / visits / show list and inspect what is on file
pmtool> import <dir> read old Word reports into visits
pmtool> new <unit> start a visit pre-filled from the last real one
pmtool> ibex COM6 receive the tester on the laptop
pmtool> ibex-file <xls> attach a saved tester file
pmtool> read-photo <jpg> read a display photo with AI
pmtool> narrative --ai draft the service notes
pmtool> compare this visit against the last one
pmtool> report --trend --style lines Word report with a trend chart
pmtool> trend-samples one report with every chart style, to pick from
pmtool> qr / qr-sheet label images for a unit or a whole site
pmtool> dropbox-link <app key> link Dropbox once (office)
```

PART 2

Behind the scenes, explained simply

Where the app's code lives, how a change becomes a new version, and what a repository, a branch and a release actually are.

What is a repository? Think of a recipe book



The recipe book

A repository is the one shared book that holds every recipe, every file, the app is made of.

Ours lives on GitHub, so it is not stuck on one laptop.
Private: only people the company invites can open it.



Every change is a page

Who changed what, when, and why. Nothing is ever erased, only added to. More than 130 pages so far.



You can flip back

Any earlier version can be opened again in seconds. A bad change is never the end.



Draft on a copy first

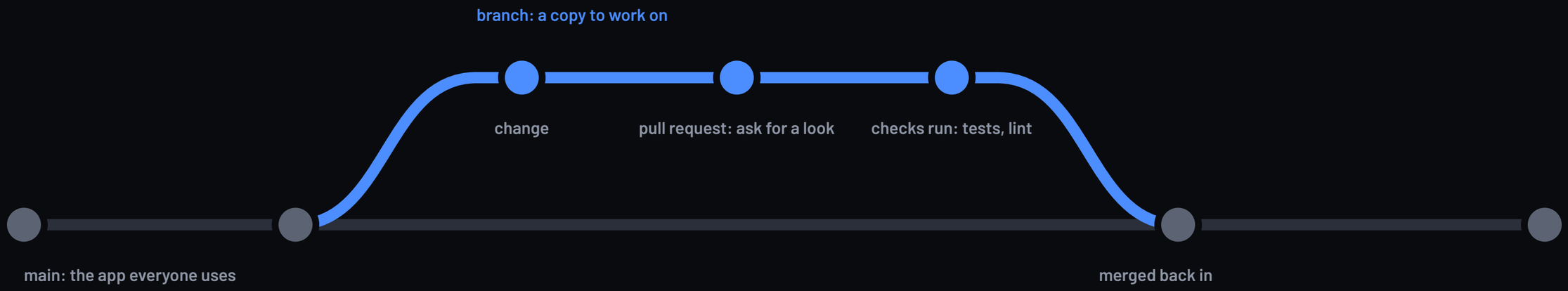
New work goes on a branch, a copy of the book, so the real one keeps working.



Every page is checked

Automatic tests and checks run before a page is allowed into the book. Part 3 shows them.

How a change gets in: branch, pull request, checks, merge



Branch

A copy of the book to write on. The real app is untouched while the change is being made.

Pull request

The note that says: here is my change, please look. It shows exactly which lines changed and why.

Checks

The tests and Android Lint run on the change by themselves. A red tests check keeps the door shut; a red lint is a warning to fix.

Versions and releases: what "Version 31" means

Version 29
Every photo goes through the in-app viewfinder
2026-09-11

Version 30
The AI key comes from the server on every Sync
2026-09-11

Version 31
The app ships with the office AI key built in
2026-09-11

Version 32
Whatever is fixed next
soon

A tag

v31 marks the exact page in the book the phones are running.

A release

The app file and pmtool are attached to the tag on GitHub, with checksums, forever.

A changelog line

Plain words: what a technician or the office would notice in this version.

From version 31 on, every publish does the three at once, so the version in Settings on the phone has a matching release page on GitHub.

The GitHub tools we use, and what each one does for us

Issues

Live

Every known problem and idea as a numbered card with a label: bug, security, field-ux, server, app. 30 filed from the code review, each with the file and the fix proposed.

Project board

Being set up

The roadmap as columns, harden to distribution to multi-tenant to templates, billing, web and connectors. Issues move across as they are done, so anyone can see where things stand.

Releases

Live

One page per version from 31 on, with the app file, pmtool and checksums attached. Version 31 is the first; every publish adds the next.

Actions

Live

Three automatic jobs: the tests on every change, Android Lint on every pull request and every change to main, and the release build when a version is tagged.

Dependabot

Live

Weekly pull requests for newer or safer parts, grouped so they are easy to review. Security alerts when a part has a known problem.

Templates and rules

Live

A Field bug report form, a feature form, a pull request checklist, a code owners file, a security policy and a contributing guide, so the next developer needs no hand-holding.

All of it lives at github.com/collinaggie89/ups-pm-app, to move under a Tech Connect organization.

Reading the changelog

One file, one section per version, one plain line per change. Newest at the top. It is written for the people who use the app, not for programmers.

"When did that change?"

Find the feature, read the version above it, check Settings on the phone.

"What will the update do?"

The Unreleased section at the top lists what is coming in the next version before it is published.

Same words on GitHub

Each release page shows its changelog section, next to the file that was published.

```
# Changelog

## Unreleased
- This field guide, under docs/guide

## Version 31 (2026-09-11)
- The app ships with the office OpenRouter key as its default; Settings is never blank

## Version 30 (2026-09-11)
- The AI key comes from the server on every Sync; a rotated key replaces the old one on every phone

## Version 29 (2026-09-11)
- The in-app viewfinder is the way every photo is taken

## Version 28 (2026-09-11)
- Photo tiles ask for the camera permission before opening the camera


"""
```

PART 3

Keeping it safe

Who can see what, where the secrets live, where AI is allowed to act, and the checks that run on every change before a person ever runs it.

Who can see what

Your token

One per person. Typed once in Settings. Every request is logged by name.

Office token

Admin jobs only: the usage table and the Dropbox link. Kept in the server's locked settings file.

Server over HTTPS

Everything travels encrypted. No token, no answer (only the health check replies without one). Tokens are kept out of the code by rule.

Dropbox rules

- Reads Customers for history
- Reads Work Orders for new jobs
- Writes only inside App Temp
- Never overwrites or deletes

Secrets stay out of the book

Keys and tokens live in the server's settings file, a git-ignored key file on the build laptop and GitHub's locked secret store. The repository is meant to hold none of them, and never holds customers' reports or photos.

If someone leaves, the office replaces one token in one file on the server and restarts the service. Nothing else changes.

Where AI does real work, and where it is not allowed to

What it does

- **Reads UPS displays** from photos, with a confidence on every value, and offers them on the Readings tab.
- **Answers questions about the visit** from the visit record, the last report and the unit's history, with any number not on the visit marked in red.
- **Rewords the service notes.** The app drafts them from the readings by rule; a writing model polishes the draft into the office's tone.
- **Sorts quick-capture photos:** anything showing readings becomes a display photo and is read; the rest print as plain photos.
- **Polishes notes on the office laptop** with pmtool, when asked.

`MODELS › google/gemini-3.8-flash reads photos · anthropic/claude-sonnet-5 writes notes · both through OpenRouter`

What it may never do

- ✕ **Invent a number.** Every figure in a draft is checked against the visit; a draft with a number that is not there is discarded, and a number in an answer that is not there is marked.
- ✕ **Change the visit from a conversation.** It suggests; the technician copies or taps to apply.
- ✕ **Replace a value you typed.** It fills only blank fields or ones still carried from last visit; anything else waits as a suggestion for a tap.
- ✕ **Touch the tester's numbers.** Strings from the IBEX-Pro or the Generex card are never changed by the AI.
- ✕ **Write the battery outlook.** That is arithmetic from the impedance history; the AI may only reword it, and any changed number is rejected.

`ENFORCED BY › the number validator in the app, on every draft, before you ever see it`

Photo reading uses a Gemini vision model and note writing a Claude writing model by default (Settings can change either), both through one OpenRouter account the office holds. The key is handed to phones by the server on every sync and can be replaced in one place. A switch in Settings turns the AI off entirely.

The automatic checks that watch the code, and the AI

Dependabot

The app is built from about sixty ready-made parts written by other people. Every Monday Dependabot checks whether any part has a security recall or a newer version, and writes up the swap as a pull request for us to approve.

`YOU SEE › a pull request titled "Bump …" every Monday, and a red alert if a part has a known hole`

Android Lint

A spell-checker for app code. It reads every screen and flags things that will break on some phones, waste battery, or are unsafe, before a person ever sees them. Its report is kept with every build.

`YOU SEE › a lint report attached to every build on GitHub`

Automatic tests

Sixty-eight taste tests. Every time the code changes, they feed the tester decoder, the report builder, the importer and the battery maths known inputs and check the answers. Red means the change cannot go in.

`YOU SEE › a green check next to every change on GitHub`

Branch protection

A change only merges when the tests are green. History cannot be force-pushed and the main copy cannot be deleted, not even by accident. Only the repository owner can override, and it is logged.

`YOU SEE › a padlock on the main branch; a red check blocks the merge button`

Crash reports

If the app ever stops on a phone, it writes down what happened and sends it to the server on the next launch, with the version and the phone model. Nothing about the customer goes with it.

`YOU SEE › a short note on the next launch, with a Send button if you want to add words`

AI-written code, checked the same way

Much of this app was written with an AI assistant. Every one of those changes goes through the same door as a human one: the tests, the lint, the pull request and a person reading the diff. Nothing ships on trust.

`YOU SEE › the same green checks, whoever or whatever wrote the change`

Five of these run without anyone remembering to run them. The safety net does not depend on a person having a good day.

Something looks wrong? Here is how to report it

1. **Note the version and the job**
   Settings shows the version. The job folder name is enough; no serial numbers or customer contacts in the report.

2. **Say what you did and what you saw**
   A screenshot is worth a paragraph. If the app stopped, the crash was already sent; just say roughly when.

3. **Send it**
   Tell Collin. He files it on GitHub with the Field bug report form, where it joins the same issue list as everything else and gets a number you can ask about.

Field bug report

App version	87
Customer and unit	09-14-26 Northgate DC, NX 60
What happened	Jar 18 shows red but the tester showed it fine. Screenshot attached.
Area	Batteries / tester
Impact	☐ Data was lost or a report went out wrong ☑ The app crashed (the crash was sent on the next launch)

TECH CONNECT

Questions, ideas, something broken?



collin@techconnectpower.com

The long version is `docs/SYSTEM-OVERVIEW.md` in the repository, which is private: ask Collin for a copy.