The WYAR 88.3 FM Station Monitor

What We Built, How It Works, and the Code Behind It.

JEFF FRANZEN · JUNE 21, 2026 · LIVE

A monitoring system for WYAR 88.3 FM that watches three different ways the station can fail and alerts a group the moment any of them happens. It is two pieces working together: a website anyone can open on a phone, and a physical probe that actually listens to the air.

PART 1 · WHAT WE BUILT

Three ways the station can go dark

1. **The internet stream is unreachable** — the public web stream is down.
2. **The stream is connected but silent** — it is "on," but broadcasting dead air.
3. **The over-the-air FM carrier is weak or missing**, or the off-site listener stops checking in.

Two pieces cover all three:

- **A website** — live at `wyar.jbf.com`. A dashboard with three signal cards (stream, audio, FM carrier), a live player, a password-protected chat room, and a recent-readings chart. Built for a phone or a laptop.
- **A physical probe** — a Raspberry Pi with a USB radio receiver that physically listens to 88.3 FM from across the water at Cumberland Foreside and reports what it hears every 60 seconds.

The key design idea: **a monitor at the station cannot raise an alarm if station power and internet both die.** Cumberland has generator-backed power and separate internet, so it can still report "the signal went dark" even during a station-side outage.

> *Status: live.*
>
> The website is built, deployed, and running. The off-site probe went live **June 20, 2026** — a Raspberry Pi 5 + Nooelec NESDR SMArt v5 receiver + antenna, posting real FM readings every minute. The first live report came back healthy, all-clear.

PROOF · OVER SSH

The live probe, seen over SSH

This is the actual off-site Raspberry Pi at Cumberland Foreside, reached over SSH from Jeff's Mac. The monitoring service is **active (running)** and has been posting a healthy reading to `wyar.jbf.com` every minute — no one had to touch it.

> *How to read it: `wyar-probe.service – active (running)` is the probe alive and set to auto-start after any reboot or power blip. Each `posted 200 ... "all-clear"` line is one minute's reading reaching the dashboard. `Main PID 3188 (python)` is the probe program itself, running from `/opt/wyar-station-monitor/probe`.*

PART 2 · HOW IT WORKS

The shape of the system

```
Vercel (the website + the brains)          ← https://wyar.jbf.com
  - checks the public stream URL itself every 5 minutes
  - receives the off-site probe's reports
  - watches the probe heartbeat (alarms if it goes quiet)
  - sends Discord, email, and SMS alerts on any state change
        ^  posts every 60 seconds
        |
Off-site probe (Raspberry Pi at Cumberland Foreside)
  - generator-backed power, separate internet from the station
  - USB SDR tuned to 88.3 MHz - measures real RF carrier strength
  - also checks the internet stream + listens for silence
```

The **website** is a React/TypeScript app hosted on Vercel. Vercel serves the dashboard and runs small backend functions. One function runs on a timer **every 5 minutes** to independently check the public stream — so even with no probe, the internet side is always watched.

The **probe** is a single Python program. Every 60 seconds it:

- pulls a few seconds of the internet stream and confirms real audio bytes are flowing,
- samples 8 seconds of audio and measures loudness to catch silence / dead air,
- sweeps the 88.3 MHz band with the USB radio and measures carrier strength,
- bundles all of it into a report and posts it to the website.

Calibration — the one tricky part

Radio signal strength is relative to the specific dongle and antenna, so we measured it on site. With WYAR present the carrier read about `-6 dBFS`; with the antenna unplugged it dropped to about `-24 dBFS` — an 18 dB gap. We set the "carrier is missing" alarm line at `-15 dBFS`, leaving roughly 9 dB of safety margin on each side so normal fluctuation will not trip a false alarm.

> *Security: the probe and the website share a secret. The website rejects any report that does not present it, so nobody can post fake readings.*

Logging into the Pi — the SSH steps

"SSH" is how you log into the Raspberry Pi remotely from a Mac and type commands. The full copy-paste runbook lives in the project; the core sequence, after flashing the Pi:

Log in, then install the tools the probe needs:

```
ssh wyar@wyar-probe.local
sudo apt update
sudo apt install -y python3-venv ffmpeg rtl-sdr git
```

Confirm the USB radio can hear 88.3 from this spot:

```
rtl_test -t
rtl_power -f 88.0M:88.6M:10k -g 30 -1 ~/wyar_test.csv && cat ~/wyar_test.csv
```

PART 2 · HOW IT WORKS (CONT.)

Run the probe, then leave it running

Put the project on the Pi, build its Python environment:

```
git clone https://github.com/franzenjb/wyar-station-monitor.git /tmp/wyar
cp -R /tmp/wyar/probe /opt/wyar-station-monitor/
cd /opt/wyar-station-monitor/probe
python3 -m venv .venv
.venv/bin/pip install -r requirements.txt
```

Test one report, then install it as a forever-running service:

```
.venv/bin/python wyar_probe.py --once          # one real post
sudo systemctl enable --now wyar-probe         # runs every 60s, restarts on boot
systemctl status wyar-probe --no-pager         # should say: active (running)
```

> *That service auto-restarts after any reboot or power blip and loops every 60 seconds — it is what keeps the probe alive without anyone touching it.*

PART 3 · THE CODE

Everything that makes it work

PIECE	WHERE	WHAT IT DOES
The probe	`probe/wyar_probe.py`	The on-Pi program (~570 lines, Python). Checks stream, audio, and FM carrier; posts a report every minute.
Service	`probe/systemd/`	Tells the Pi to run the probe forever and restart it automatically.
The website	`src/App.tsx`	The dashboard (~1,000 lines, React): signal cards, live player, password gate, chat room, chart.
Backend	`api/*.ts`	Functions on Vercel: receive probe posts, feed the dashboard, and the 5-minute stream check.
Alerts + logic	`api/_shared.ts`	The shared brain: decides health, fans out Discord, email (Resend), and SMS (Twilio) alerts.
Chat storage	`supabase/`	Database tables for the monitor chat room (messages, edits, attachments).
Tests	`e2e/`	Automated browser tests (Playwright) that load the dashboard and verify it renders.

Tech stack: React 19 + TypeScript + Vite frontend; Vercel serverless functions + cron for the backend; Supabase (Postgres) for chat; Python + `rtl-sdr` + `ffmpeg` on the Raspberry Pi. Alerts via Discord, Resend (email), and Twilio (SMS).

> **A note, just in case.** *If anything should ever happen to me, show Shannon this document. She can pass along the Raspberry Pi, the antenna, and the SDR so someone can pick this up and keep WYAR's watch running. Everything needed to rebuild it is written in these pages. — Jeff*