

Sutradhar

Every app on one laptop, on and off, from anywhere.

Stack	Python 3 standard library and one cryptography package. No pip, no npm, no build step. A <code>systemd --user</code> daemon serving a local page, plus a hand-written PHP half on navyasharma.org that holds a snapshot and a queue
Status	Complete. A live service on the laptop it manages, since September 2026
Size	Under a megabyte of source, built in a day and redesigned two days later

The brief

Several things I have built run on the same laptop as long-lived services: a WhatsApp assistant for a student club, another for me, a node in a lab's calculation cluster, a local language model that two of those share. Each has its own units, timers and logs, and each has a small number of things one might want to do to it – restart it, run its health check, switch one of its behaviours off – that were until now a terminal command someone had to remember.

I wanted to switch them on and off from my phone and see what each was costing in memory and GPU. That was the whole brief. It started as a script inside my personal workspace and became a project on the afternoon it turned out to have a security boundary.

Sūtradhāra is the stage manager of Sanskrit drama, the one who holds the strings. He starts and stops the play and knows every player's state.

Nothing calls the laptop

A daemon on the laptop serves a page on the machine itself and on my private tailnet. For the phone there is a small PHP half on this site that holds the laptop's most recent snapshot and a queue of commands; the laptop polls it every two seconds while someone is watching the page and every twenty seconds otherwise, picks up what has been queued, runs it, and pushes a fresh snapshot. Shared hosting cannot keep a process alive and the laptop has no public address, so an outbound poll is the only shape that works. It is the same shape my personal assistant uses when WhatsApp is unavailable, which is why the two were built to one pattern.

An **app** is a bundle of `systemd` units named in a registry file, and “off” stops all of them. An app that has no unit of its own gives a command and gets one rendered for it. Readings – memory, CPU, since when – come from the control group rather than from a process id, so an app that forks is counted whole, and an app whose child wanders out of the group is noticed. Each project carries a short file saying how it registers, and a system unit can be registered read-only: it gets a lock on the board instead of a switch.

The registry can also describe **switches**. A project gives one command that prints its toggles as JSON and one that sets a toggle by name; the daemon reads them in the background at most every thirty seconds and again right after one is set, so a snapshot never waits on a subprocess. A flag deep in another project's configuration thereby becomes a toggle on the board without that project knowing the board exists. The club assistant's four behaviour switches – answer, act, speak first, use the local model – are on the board this way, and can be flipped from a phone without a restart.

The page

The first page was one scroll of cards showing everything at once. A button's output landed in a results panel at the bottom, and the only toggle on a card was “start at boot,” which read as the app's switch and was not. It was neither intuitive nor navigable, and I use it on the laptop far more than on the phone, which the first design had assumed the other way round.

The second is three columns, desktop first. Three directions were drawn before one was built – a board, one page of collapsible rows, and a feed like a chat – and the board won because the laptop is the surface.

- **The board**, left: one row per app with its state, one reading, and *the* switch – start and stop, or a lock for a system unit.

- **The app**, middle: its switches as toggles with their state; its buttons under *Do*, with each result appearing under the button that asked for it; anything that needs typing under *Ask*; its readings and its units.
- **The side**: everything asked of the laptop, newest first, and the shell.

One sentence in the header answers the glance question: “Nothing is wrong.” or “Setu has failed.” The vitals are a single line and the pause-everything control is deliberately quiet. Under a thousand pixels the three columns stack.

The one dangerous button

The shell is the only thing on the page that can do arbitrary harm, and it is treated accordingly. From the site it needs the admin login *and* a PIN, and the command is sealed in the browser to the laptop’s public key before it is sent, so the site’s database only ever holds ciphertext with nothing to guess against; a compromise of the site yields a queue of commands nobody can read. The laptop counts wrong PINs and locks at five, until it is reset at the keyboard. On the tailnet, where the browser’s crypto API is unavailable over plain HTTP, the command goes in the clear inside the tailnet’s own encryption; on the machine itself it needs no PIN at all.

Every other button is a named action from the registry with fixed arguments. The page cannot invent a command; it can only choose among the ones each project offered.

Tested in a real browser, on all three surfaces

The whole thing was driven end to end over the browser’s DevTools protocol, signed in through the site’s own login form, exercising every command kind and every error path: start, stop, restart, enable, disable, health, log, pause, resume, reload, a system unit refused, an unknown app, an unknown kind, a shell that times out, a shell that exits non-zero.

That found four bugs on the first day, and none of them was visible in a diff. The page gave a sealed shell command a second id after sealing it with the first, so the laptop refused every one. The site’s copy of the HTML had drifted from the local one, so the site page threw on load – the body is now one fragment cut from the local page at sync time, so there is nothing to drift. “On at boot” also started the app. And two PIN checks racing could count one wrong answer twice, so the check now runs under a lock.

It also found a fifth bug that was not its own: starting the lab node on this laptop died with a name error in that project’s code, and the board said “failed, exit 1.” Reporting somebody else’s failure accurately is the first useful thing a monitor does, and it was the moment the project felt finished.