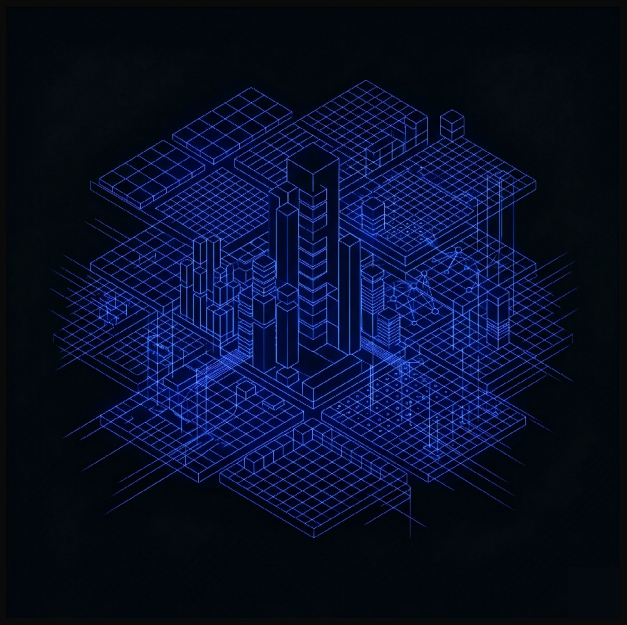
01 · ANALYTICS & MONITORING

CHAPTER 1.3

Only what changes the decision of the day

STUDENT ACTIVITY MONITOR

A bot that watches a student's learning pace every five minutes and pings via Telegram only when it's worth a look.



STATUS	STACK	DATA	DELIVERY
Production	Node.js 18 · 0 deps	Platform API · Telegram	Telegram · low latency

01 THE PROBLEM

THE DATA EXISTED; NO ONE SAW IT IN TIME

Tracking a student on an adaptive practice platform was, by default, a manual student-by-student review: someone had to log into the dashboard, open the profile, read the day's XP, skim the topics worked on, and infer whether the pace was healthy or stalling.

For a Middle School student carrying accumulated conceptual gaps, that tracking could not be weekly nor depend on someone remembering to look.

The concrete problem was one of *latency* and *attention*: when a session went off the rails — repeated low accuracy on the same topic, little progress by mid-afternoon, early abandonment — the data existed in the API, but no one saw it in time to intervene the same day.

Checking the platform by hand every five minutes was unworkable.

And receiving all the raw detail would have been useless noise. The challenge was not accessing the data, but distilling from it only the signals that warrant an action.

02 THE OBJECTIVE

A DEFINITION OF SUCCESS IN SIGNAL-TO-NOISE TERMS

PRIMARY OBJECTIVE

Close the distance between the data and the intervention: detect actionable signals in near real time and push only those signals — not the noise — to a channel where whoever supports the student sees them effortlessly.

I defined success in advance and in signal-to-noise terms: a day of monitoring that produced between **six and ten meaningful messages** — a briefing at the start, progress milestones, critical alerts, session close — instead of the fifteen to twenty trivial pings a naive system would generate, or the total silence of a manual review.

The system had to run on its own, without anyone turning it on, within school hours.

03 THE USERS

THE ALERT ARRIVES WHERE THE PERSON ALREADY IS

The direct user is the person who supports the student's learning: they need to know, without opening any dashboard, whether today's session is on track and where to step in.

The monitored subject is a Middle School student with prior gaps — anonymized as **R-0388** in this chapter — whose recovery plan depended on close, daily support.

The chosen channel was **Telegram** precisely because it lives on the supporter's phone: alerts arrive where the person already is, not on a dashboard they have to remember to visit.

Later the same engine went on to track several students in parallel, each with their own daily XP goal — without touching the core logic.

That choice of channel is not cosmetic: it defines the product's contract. If the notification interrupts, it had better be worth the interruption.

04 THE ARTIFACT

A FULL DAY IN THE TELEGRAM FEED

The product is, for the person using it, a conversation. The bot opens the day with a **morning briefing** that situates the day against the week, marks goal milestones as they are crossed, fires a red alert when a topic genuinely gets stuck, and closes with the session summary. Between six and eight bubbles that tell the story of the day without anyone opening a dashboard.

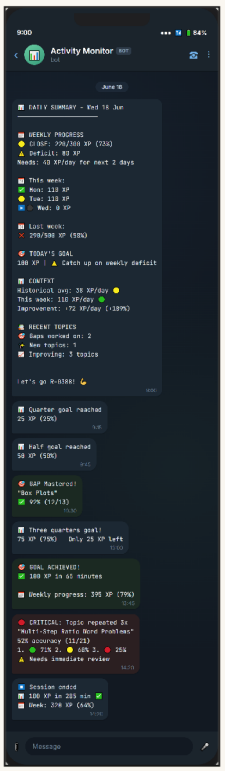


FIG. 1 — A DAY'S TELEGRAM THREAD FOR R-0388 · BRIEFING, XP MILESTONES, CRITICAL ALERT AND CLOSE · RAW DETAIL DISTILLED INTO ACTIONABLE SIGNALS

05 THE FOUNDATION

AN ALERT THAT ALWAYS ARRIVES STOPS BEING AN ALERT

The notification design is anchored in a principle of cognitive load and relevance. The system does not report continuous progress but **significant milestones** — goal quarters at 25, 50, 75 and 100 percent of XP — that correspond to moments when progress changes phase, not to arbitrary increments.

PRINCIPLE	SIGNAL	WHEN IT FIRES
Relevance / cognitive load	`Goal milestone`	On crossing 25, 50, 75 and 100% of the day's XP — phases of progress, not increments.
Deliberate practice	`Critical topic`	Same topic repeated in the day with accuracy below 60% over ≥8 questions — the pattern, not the isolated error.
Deep failure	`Hard alert`	Accuracy below 30% on a topic, or a marked regression against history.
Temporal context	`Briefing`	Start of day situated against the weekly trend and the historical average.
Cycle close	`Session end`	Summary on detecting 30 min of inactivity within school hours.

> *Detecting problem topics rests on deliberate practice: what matters is not an isolated error, but the pattern that recurs with sustained low accuracy.*

The morning briefing situates today against the weekly trend and the historical average, because a metric without context does not allow a decision: **29 XP mean different things** depending on whether the week is running behind or ahead.

06 THE DESIGN

AN AUSTERE ARCHITECTURE AND THE THREE DECISIONS THAT DEFINE IT

The architecture is deliberately austere. The monitor is a single Node.js process **with not a single external dependency** — only native modules — that on each run queries the platform API, compares the freshly read activity against a state persisted on disk, and derives from that difference which notifications are warranted. State lives in JSON files, which keeps the system database-free and trivially inspectable.

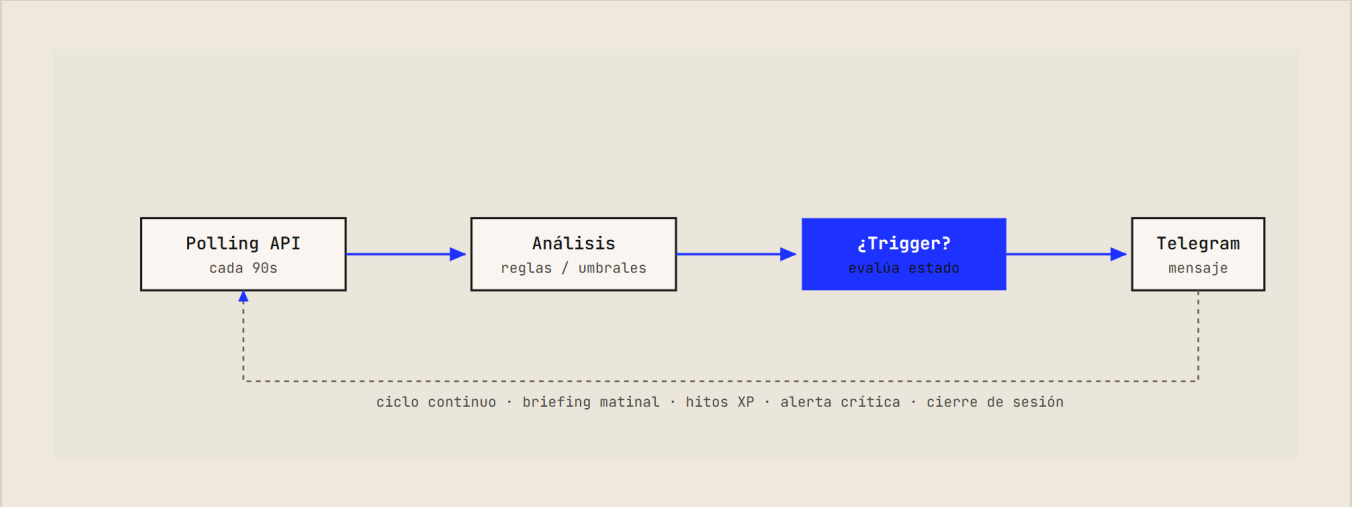


FIG. 2 — THE CYCLE EVERY 5 MIN: CRON-RUNNER → API POLLING → DIFF AGAINST STATE → ANALYSIS → ALERT TO TELEGRAM

Three architecture decisions that define the product

Separate scheduler from monitor. A cron-runner keeps an HTTP server with a health endpoint and fires the monitor every five minutes; that *health check* is what keeps the hosting platform from sleeping the service for inactivity.

Idempotent notifications per day. Flags in the state prevent spam structurally — a milestone once sent is not resent, a critical alert is not repeated — rather than through fragile filters.

Fit within school hours. Monitoring between 9:00 and 16:10 Campus B time, with its own computation of the EST/EDT transition, so as not to generate noise outside class. The network layer uses retries with exponential backoff that distinguishes client errors — which are not retried — from server errors.

07 CONSTRUCTION

TWELVE COMMITS, TWO LESSONS THAT HURT

I built the monitor in an AI-first flow, leaning on Claude to iterate quickly over the notification logic and the API handling. The bulk of the work concentrated in **twelve commits between March 12 and May 21, 2026**. Thresholds and triggers live centralized, so recalibrating relevance does not require touching the logic.

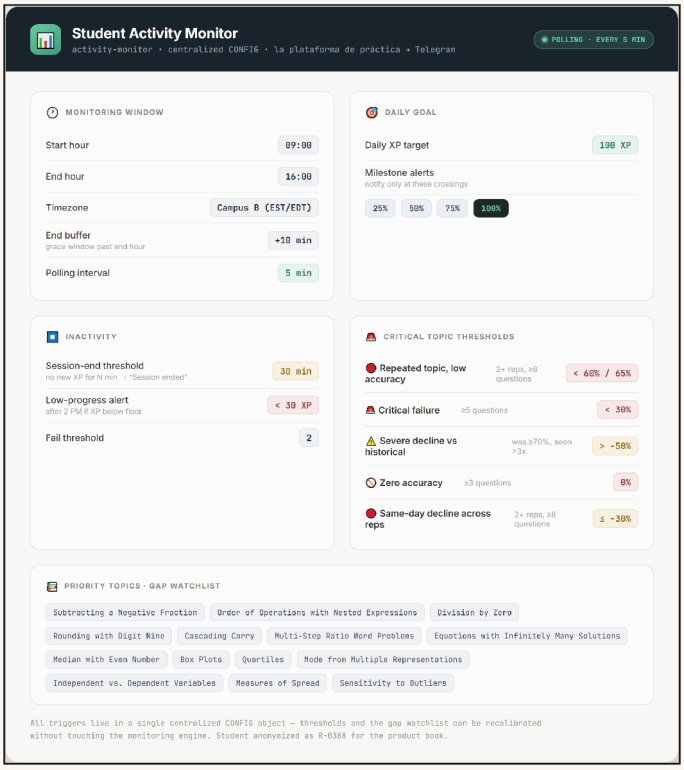


FIG. 3 — TRIGGER CONFIGURATION · MILESTONES, CRITICAL-TOPIC THRESHOLDS AND SCHOOL-HOURS WINDOW CENTRALIZED TO RECALIBRATE WITHOUT TOUCHING THE ENGINE

The first real obstacle was the API itself: requests returned a persistent 400 error until I identified that the *Accept* and *Content-Type* headers were missing and that the base URL had to point directly to the version prefix of the practice platform; I reused the correct configuration already validated in an earlier tracking project. The second lesson came from the persisted state: when migrating the service to the cloud it inherited a local state with all flags already set, so the monitor ran flawlessly but generated **zero notifications** because it believed it had already sent them. That forced me to treat state as a first-class citizen, with a clean reset per instance. The notification system reached a version 4.0 that cut the volume from fifteen-to-twenty pings down to six-to-eight messages with context.

08 VALIDATION

IN PRODUCTION, AGAINST THE EDGE CASES

I validated the system in real production, not in a test environment. The activity log shows the cycle working end to end: query to the API, reading of the day's XP and questions, comparison against state, and effective delivery

of the message via Telegram. I explicitly tested the edge cases that break this kind of bot — running outside hours, critical-pattern detection, days with no activity, API inconsistencies returning totals from different timestamps — with a battery of dedicated scripts before trusting the automatic flow.

The measure of success I cared about was not technical but **signal-to-noise**: that a full day produced the right number of actionable messages. The system proved it could sustain the five-minute cycle within school hours and deliver the morning briefing, the XP milestones, the critical-topic alerts and the inactivity session-close as differentiated and timely messages.

09 THE RESULT AND THE LEGACY

FROM ONE STUDENT TO SEVERAL, WITHOUT TOUCHING THE ENGINE

An individual learning monitor in production that turns the API of an adaptive practice platform into low-latency, high-relevance support. Whoever supports the student stops reviewing dashboards by hand and receives on their phone only what changes the decision of the day.

Monitored student	`R-0388 · Middle School`
Construction	AI-first · no external dependencies · centralized thresholds
Integrations	the practice platform's API · Telegram Bot API
Signal volume	6-8 messages/day (down from 15-20)
Deployment	Railway, migrated to Render
Status	**In production**

The same engine scaled from one student to several in parallel — each with their own daily goal, state file, history and log — without touching the core logic, which confirmed that the dependency-free, per-student state-oriented architecture was the right decision. The service survived a change of hosting platform while keeping the health check that keeps it awake. It rests on the same visibility layer that underpins the analytics portfolio `Ch. 01` `Ch. 02`.

LESSONS

- **Product.** The value of an alert system lies in what it decides NOT to send. Designing relevance — milestones instead of increments, patterns instead of isolated errors, weekly context instead of loose figures — was harder and more important than connecting the API.
- **Technical.** In a stateful monitor, persisted state is part of the design, not an implementation detail: the inherited-flags bug — a system that ran flawlessly while producing zero output — showed that the silent errors live there.
- **Architecture.** The payoff of deliberate simplicity: zero dependencies, JSON files and a health check were enough for a service that runs on its own every school day — and that austerity was what made it trivial to scale from one student to several.

2 TRAINING PLATFORMS

The instruments that produce change: tackling the most expensive difficulty leaps, from the SAT's 650→800 to the AP's 3→5.