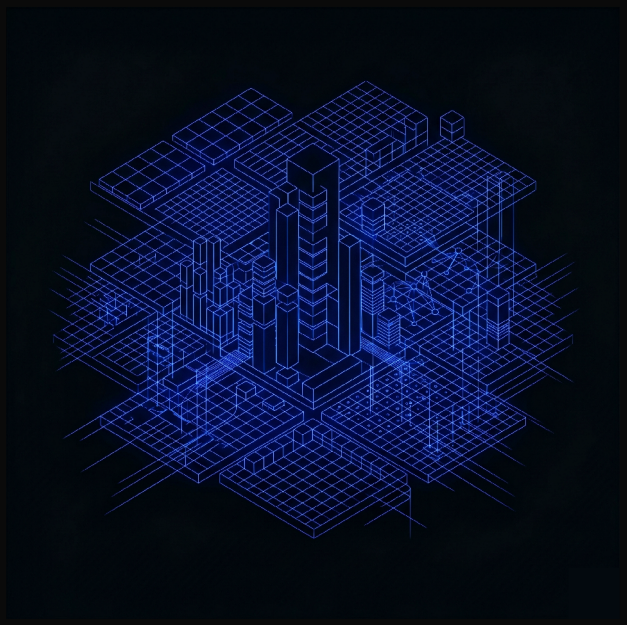
Solo lo que cambia la decisión del día

STUDENT ACTIVITY MONITOR

Un bot que vigila el ritmo de aprendizaje de un estudiante cada cinco minutos y avisa por Telegram solo cuando vale la pena mirar.



ESTADO	STACK	DATOS	ENTREGA
Producción	Node.js 18 · 0 deps	API de la plataforma · Telegram	Telegram · baja latencia

01 EL PROBLEMA

EL DATO EXISTÍA; NADIE LO VEÍA A TIEMPO

El seguimiento de un estudiante en una plataforma de práctica adaptativa era, por defecto, una revisión manual alumno por alumno: alguien tenía que entrar al panel, abrir el perfil, leer el XP del día, hojear los temas trabajados y deducir si el ritmo iba bien o se estaba estancando.

Para una estudiante de Middle School con brechas conceptuales acumuladas, ese seguimiento no podía ser semanal ni depender de que alguien se acordara de mirar.

El problema concreto era de *latencia* y de *atención*: cuando una sesión se torcía —baja precisión repetida en un mismo tema, poco avance a media tarde, abandono temprano— el dato existía en la API, pero nadie lo veía a tiempo para intervenir el mismo día.

Revisar la plataforma cada cinco minutos a mano era inviable.

Y recibir todo el detalle bruto habría sido ruido inservible. El reto no era acceder al dato, sino destilar de él solo las señales que merecen una acción.

02 EL OBJETIVO

UNA DEFINICIÓN DE ÉXITO EN SEÑAL-RUIDO

OBJETIVO PRIMARIO

Cerrar la distancia entre el dato y la intervención: detectar señales accionables en tiempo casi real y empujar solo esas señales —no el ruido— a un canal donde quien acompaña al estudiante las vea sin esfuerzo.

Definí el éxito de antemano y en términos de señal-ruido: un día de monitoreo que produjera entre **seis y diez mensajes con sentido** —un briefing al inicio, hitos de avance, alertas críticas, cierre de sesión— en lugar de los quince a veinte avisos triviales que generaría un sistema ingenuo, o del silencio total de la revisión manual.

El sistema tenía que correr solo, sin que nadie lo encendiera, dentro del horario lectivo.

03 LOS USUARIOS

LA ALERTA LLEGA DONDE LA PERSONA YA ESTÁ

El usuario directo es la persona que acompaña el aprendizaje del estudiante: necesita saber, sin abrir ningún panel, si la sesión de hoy va encaminada y dónde hay que meter mano.

El sujeto monitoreado es una estudiante de Middle School con brechas previas —anonimizada como **R-0388** en este capítulo—, cuyo plan de recuperación dependía de un acompañamiento diario y cercano.

El canal elegido fue **Telegram** precisamente porque vive en el teléfono de quien acompaña: las alertas llegan donde la persona ya está, no a un dashboard que hay que recordar visitar.

Más tarde el mismo motor pasó a seguir a varios estudiantes en paralelo, cada uno con su propia meta diaria de XP — sin tocar la lógica central.

Esa elección de canal no es cosmética: define el contrato del producto. Si la notificación interrumpe, más vale que valga la interrupción.

04 EL ARTEFACTO

UN DÍA COMPLETO EN EL FEED DE TELEGRAM

El producto es, para quien lo usa, una conversación. El bot abre la jornada con un **briefing matinal** que sitúa el día contra la semana, va marcando los hitos de meta al cruzarse, dispara una alerta roja cuando un tema se atasca de verdad, y cierra con el resumen de sesión. Entre seis y ocho burbujas que cuentan el día sin que nadie abra un panel.

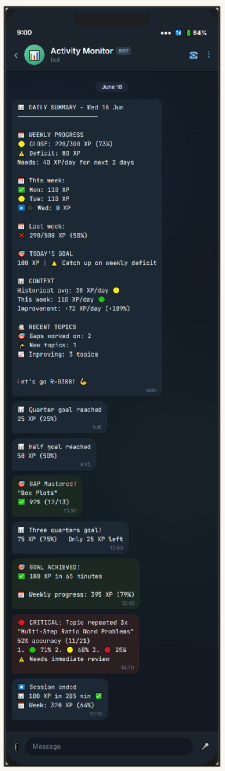


FIG. 1 — HILO DE TELEGRAM DE UN DÍA DE R-0388 · BRIEFING, HITOS DE XP, ALERTA CRÍTICA Y CIERRE · EL DETALLE BRUTO DESTILADO EN SEÑALES ACCIONABLES

05 EL FUNDAMENTO

UNA ALERTA QUE LLEGA SIEMPRE DEJA DE SER UNA ALERTA

El diseño de las notificaciones está anclado en un principio de carga cognitiva y de relevancia. El sistema no reporta avance continuo sino **hitos significativos** —cuartos de meta a 25, 50, 75 y 100 por ciento de XP— que corresponden a momentos en que el progreso cambia de fase, no a incrementos arbitrarios.

PRINCIPIO	SEÑAL	CUÁNDO DISPARA
Relevancia / carga cognitiva	`Hito de meta`	Al cruzar 25, 50, 75 y 100% del XP del día — fases del progreso, no incrementos.
Práctica deliberada	`Tema crítico`	Mismo tema repetido en el día con precisión bajo 60% sobre ≥8 preguntas — el patrón, no el error aislado.
Fallo profundo	`Alerta dura`	Precisión bajo 30% en un tema, o retroceso marcado frente al historial.
Contexto temporal	`Briefing`	Inicio del día situado contra la tendencia semanal y el promedio histórico.
Cierre de ciclo	`Fin de sesión`	Resumen al detectar 30 min de inactividad dentro del horario lectivo.

> *La detección de temas problemáticos se apoya en la práctica deliberada: lo que importa no es un error aislado, sino el patrón que se repite con precisión baja sostenida.*

El briefing matinal sitúa el día de hoy contra la tendencia semanal y el promedio histórico, porque una métrica sin contexto no permite decidir: **29 XP significan cosas distintas** según si la semana viene atrasada o adelantada.

06 EL DISEÑO

ARQUITECTURA AUSTERA Y LAS TRES DECISIONES QUE LA DEFINEN

La arquitectura es deliberadamente austera. El monitor es un único proceso Node.js **sin una sola dependencia externa** —solo módulos nativos— que en cada ejecución consulta la API de la plataforma, compara la actividad recién leída contra un estado persistido en disco y deriva de esa diferencia qué notificaciones corresponden. El estado vive en archivos JSON, lo que mantiene el sistema sin base de datos y trivialmente inspeccionable.

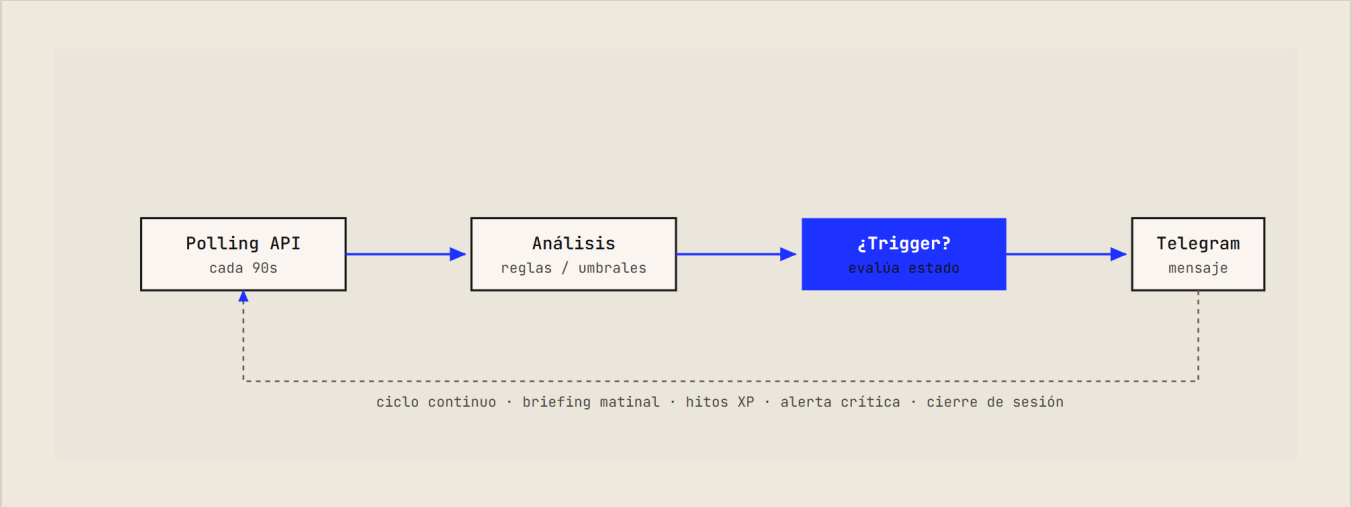


FIG. 2 — EL CICLO CADA 5 MIN: CRON-RUNNER → POLLING DE LA API → DIFF CONTRA ESTADO → ANÁLISIS → ALERTA A TELEGRAM

Tres decisiones de arquitectura que definen el producto

Separar scheduler de monitor. Un cron-runner mantiene un servidor HTTP con endpoint de salud y dispara el monitor cada cinco minutos; ese *health check* es lo que impide que la plataforma de hosting duerma el servicio por inactividad.

Notificaciones idempotentes por día. Banderas en el estado evitan el spam estructuralmente —un hito enviado no se reenvía, una alerta crítica no se repite— en vez de a fuerza de filtros frágiles.

Encajar en el horario lectivo. Monitoreo entre 9:00 y 16:10 hora de Campus B, con cómputo propio de la transición EST/EDT, para no generar ruido fuera de clase. La red usa reintentos con backoff exponencial que distingue errores de cliente —que no se reintentan— de errores de servidor.

07 CONSTRUCCIÓN

DOCE COMMITS, DOS APRENDIZAJES QUE DOLIERON

Construí el monitor en un flujo AI-first, apoyándome en Claude para iterar rápido sobre la lógica de notificaciones y el manejo de la API. El cuerpo del trabajo se concentró en **doce commits entre el 12 de marzo y el 21 de mayo de 2026**. Los umbrales y triggers viven centralizados, de modo que recalibrar la relevancia no exige tocar la lógica.

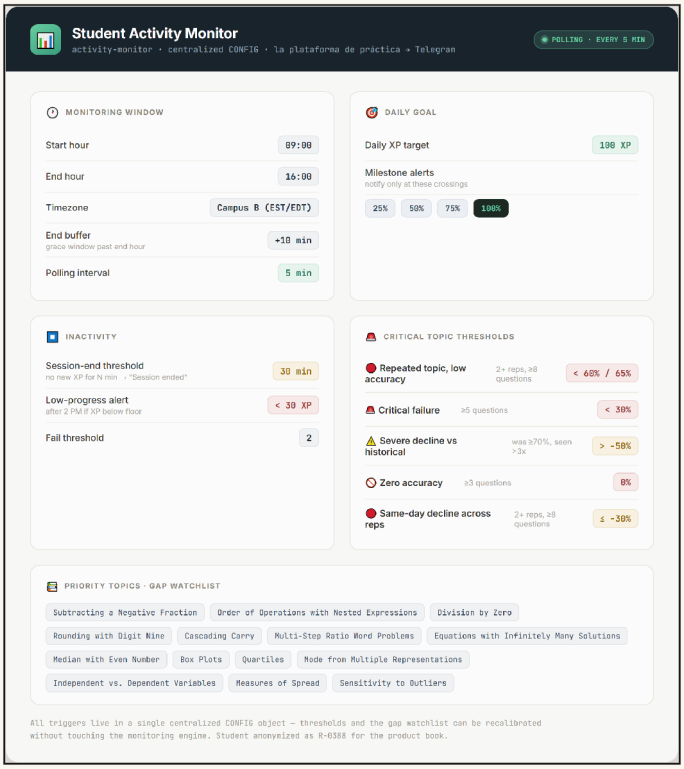


FIG. 3 — CONFIGURACIÓN DE TRIGGERS · HITOS, UMBRALES DE TEMA CRÍTICO Y VENTANA LECTIVA CENTRALIZADOS PARA RECALIBRAR SIN TOCAR EL MOTOR

El primer obstáculo real fue la propia API: las peticiones devolvían error 400 de forma persistente hasta que identifiqué que faltaban las cabeceras *Accept* y *Content-Type* y que la URL base debía apuntar directamente al prefijo de la versión de la plataforma de práctica; reutilicé la configuración correcta ya validada en un proyecto de tracking anterior. El segundo aprendizaje vino del estado persistido: al migrar el servicio a la nube heredó un estado local con todas las banderas ya marcadas, de modo que el monitor corría perfecto pero generaba **cero notificaciones** porque creía que ya las había enviado. Eso me obligó a tratar el estado como ciudadano de primera clase, con reset limpio por instancia. El sistema de notificaciones llegó a una versión 4.0 que redujo el volumen de quince-veinte avisos a seis-ocho mensajes con contexto.

08 VALIDACIÓN

EN PRODUCCIÓN, CONTRA LOS CASOS DE BORDE

Validé el sistema en producción real, no en un entorno de prueba. El registro de actividad muestra el ciclo funcionando de extremo a extremo: consulta a la API, lectura de XP y preguntas del día, comparación con el estado y envío efectivo del mensaje por Telegram. Probé explícitamente los casos de borde que rompen este tipo de bots —ejecución fuera de horario, detección de patrones críticos, días sin actividad, inconsistencias de la API que devuelve totales de distintos timestamps— con una batería de scripts dedicados antes de confiar en el flujo automático.

La medida de éxito que me importaba no era técnica sino de **señal-ruido**: que un día completo produjera la cantidad correcta de mensajes accionables. El sistema demostró sostener el ciclo de cinco minutos dentro del horario lectivo y entregar el briefing matinal, los hitos de XP, las alertas de tema crítico y el cierre de sesión por inactividad como mensajes diferenciados y oportunos.

09 EL RESULTADO Y EL LEGADO

DE UN ESTUDIANTE A VARIOS, SIN TOCAR EL MOTOR

Un monitor de aprendizaje individual en producción que convierte la API de una plataforma de práctica adaptativa en un acompañamiento de baja latencia y alta relevancia. Quien acompaña deja de revisar paneles a mano y recibe en su teléfono solo lo que cambia la decisión del día.

Estudiante monitoreado	`R-0388 · Middle School`
Construcción	AI-first · sin dependencias externas · umbrales centralizados
Integraciones	la plataforma de práctica API de la plataforma · Telegram Bot API
Volumen de señal	6-8 mensajes/día (desde 15-20)
Despliegue	Railway, migrado a Render
Estado	**En producción**

El mismo motor escaló de un estudiante a varios en paralelo —cada uno con su meta diaria, su archivo de estado, historial y log— sin tocar la lógica central, lo que confirmó que la arquitectura sin dependencias y orientada a

estado por estudiante era la decisión correcta. El servicio sobrevivió a un cambio de plataforma de hosting conservando el health check que lo mantiene despierto. Se apoya en la misma capa de visibilidad que sostiene el portafolio de analítica Cap. 01 | Cap. 02.

APRENDIZAJES

- **De producto.** El valor de un sistema de alertas está en lo que decide NO enviar. Diseñar la relevancia —hitos en vez de incrementos, patrones en vez de errores aislados, contexto semanal en vez de cifras sueltas— fue más difícil y más importante que conectar la API.
- **Técnico.** En un monitor stateful, el estado persistido es parte del diseño, no un detalle de implementación: el bug de las banderas heredadas —un sistema que corría perfecto produciendo cero salida— mostró que ahí viven los errores silenciosos.
- **De arquitectura.** El rendimiento de la simplicidad deliberada: cero dependencias, archivos JSON y un health check bastaron para un servicio que corre solo todos los días lectivos — y esa austeridad fue lo que hizo trivial escalar de uno a varios estudiantes.

2 PLATAFORMAS DE ENTRENAMIENTO

Los instrumentos que producen cambio: atacar los saltos de dificultad más caros, del 650→800 del SAT al 3→5 del AP.