ChurnGuard AI

An AI-orchestrated system that catches SaaS churn before the cancellation email

Case study · Tabassum Khanum · Built July 2026

Summary

ChurnGuard AI is an end-to-end retention system for a mid-market SaaS product. Instead of waiting for a cancellation request, it scores every customer's usage every six hours, and when the signals turn negative, it puts an AI voice agent on the phone within minutes, briefed on that customer's real usage history, open tickets, and unused features. If the call doesn't land, it falls back to WhatsApp, then email, then a human. Every interaction feeds a live dashboard built for a retention team, not an engineer.

It was built solo over three weeks, covering product design, the data model, prompt engineering, and six orchestration workflows, as a portfolio project meant to demonstrate PM judgment on an agentic system, not just the ability to wire APIs together.

Note: this document covers system design, architecture, and the product decisions behind the build. Dashboard figures shown are illustrative until a live demo cohort produces real numbers.

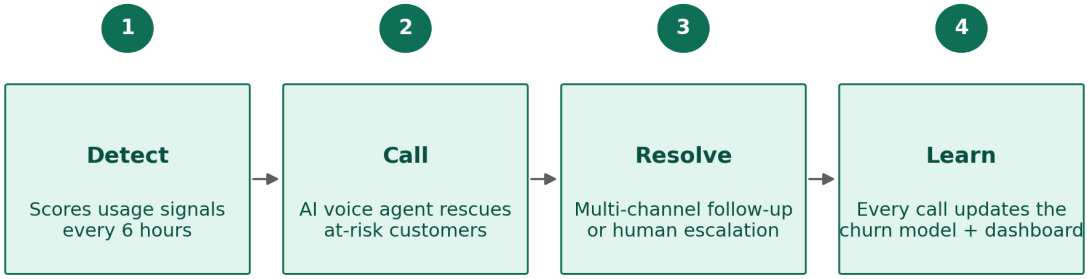
The problem

Most retention tooling is reactive: dashboards flag risk, but the intervention, a generic email or a discount code, arrives after the customer has already decided to leave. The gap usually isn't detection; most teams already have usage data sitting in a warehouse somewhere. The gap is intervention: someone has to notice, personalize an outreach, and act, consistently, within a narrow window, for every at-risk account. That doesn't scale past a handful of high-touch accounts a CS lead can track by hand.

Voice specifically was the interesting bet. Email and in-app banners are easy to ignore; a phone call that opens like a genuine check-in, references what the customer actually uses, and can create a support ticket or escalate to a human mid-conversation is a fundamentally different quality of outreach, but only if it's cheap enough to run on every at-risk account, not just the handful a CS team would call by hand.

The solution

ChurnGuard closes that gap by making the intervention itself automatic and personalized, on a four-step loop:



Detect → Call → Resolve → Learn

Detect. Every 6 hours, a scoring pass looks at login frequency, usage trend, unused features, and open tickets to rank customers by churn risk, and filters out anyone contacted in the last 48 hours.

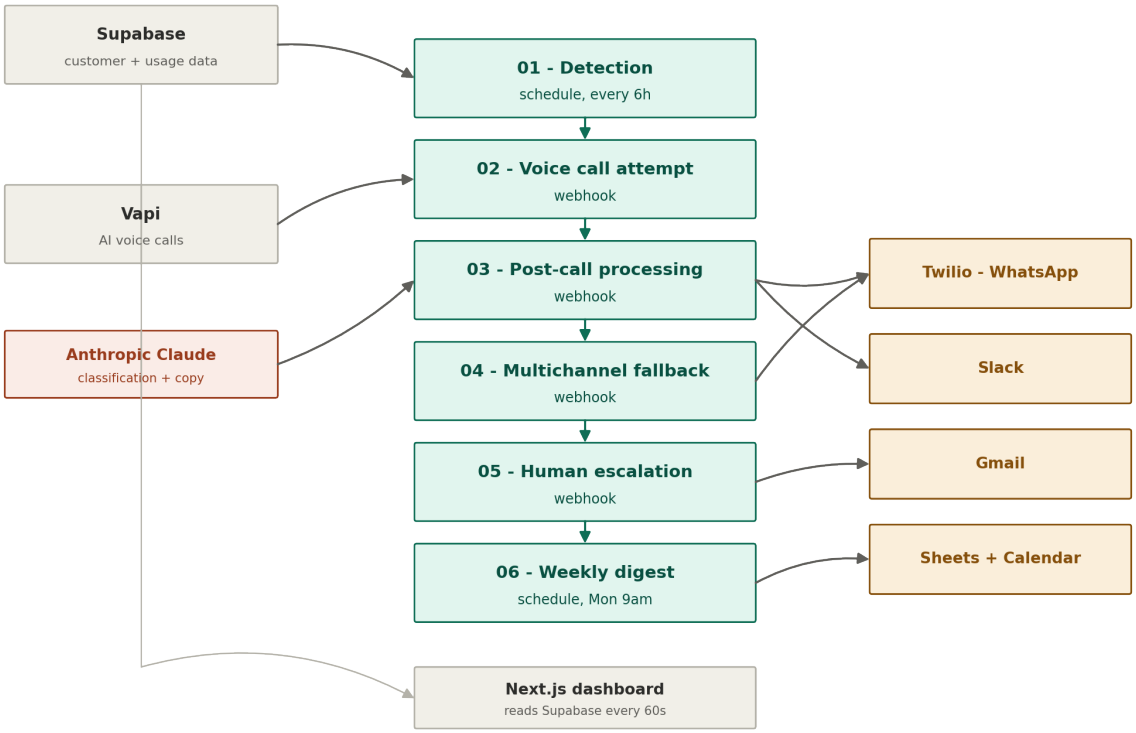
Call. The top 5 at-risk customers get an outbound call from a Vapi voice agent, briefed in real time with that customer's name, plan, days since login, usage trend, and unused features, so the conversation opens specific, not generic.

Resolve. If the call doesn't connect, the system falls back to WhatsApp, then email. If the customer's problem is beyond what the agent can handle, it escalates live to a human with full context already assembled, so there is no cold handoff.

Learn. Every call is classified for outcome and sentiment, updates a rolling per-customer health score, and rolls up into a dashboard and a weekly narrative digest, so the risk model and the talking points can both improve over time.

System architecture

Six n8n workflows sit between the Next.js app and five external services: Vapi for voice, Twilio for WhatsApp, Slack, Gmail, and Google Sheets/Calendar, with Anthropic's Claude doing call classification and copy generation along the way. n8n was chosen deliberately as the orchestration layer rather than more custom backend code; the full rationale is in the accompanying PRD.



The six workflows and the services they orchestrate

- **01 Detection** - scheduled every 6h, scores and ranks at-risk customers.
- **02 Voice call attempt** - places the Vapi call with live customer context.
- **03 Post-call processing** - classifies the transcript and routes by outcome.
- **04 Multichannel fallback** - WhatsApp, then email, if the call doesn't land.
- **05 Human escalation** - Slack alert, calendar hold, and confirmation email.
- **06 Weekly digest** - Monday-morning narrative summary for the founder.

My role and process

I designed and built ChurnGuard end to end over three weeks: the Supabase schema and churn-detection logic, the Vapi voice agent's system prompt and tool calls, six n8n workflows orchestrating five third-party services, and the analytics dashboard. Implementation was done in Claude Code against a phased prompt plan: project setup and schema first, then the detection API and RAG knowledge base, then the Vapi tool endpoints, then the n8n workflows in dependency order (core rescue flow, then fallback and escalation, then the digest), and finally the dashboard and launch assets.

My job in that process was the product decisions, not the code: what to automate versus leave to a human, how aggressive the call cadence should be, and where the experience could tip from helpful into invasive. Every non-obvious call is logged with its rationale below, both as documentation and as interview prep.

Designing the voice agent

The agent has exactly three tools, deliberately kept narrow: **search_knowledge_base** to pull answers from a RAG-indexed support corpus, **create_support_ticket** for issues it can log but not resolve live, and **escalate_to_human** for anything past its authority, such as cancellation threats, anger, or a request the agent isn't confident about. The system prompt explicitly bans two things: mentioning usage analytics as the reason for the call, and offering discounts or concessions the agent isn't authorized to make. Every tool response is written to be spoken aloud: short, no bullet points, no jargon, since a voice agent reading a formatted list sounds broken.

Key product decisions

n8n as orchestrator, not custom backend code. Branching, retries, and five third-party integrations are exactly what a workflow tool is for. Hand-rolling this in Next.js API routes means rebuilding retry logic, OAuth flows, and a monitoring UI n8n already ships with. Trade-off accepted: an extra system to operate and a dependency on n8n's uptime.

48-hour cooldown between calls. Without it, the same customer could be flagged and called again the next cycle, which reads as harassment, not support. Trade-off accepted: a genuinely urgent case might wait up to two days for a second attempt.

The agent never mentions analytics as the reason for calling. Saying we noticed you haven't logged in reads as surveillance. The call opens as a check-in, not a confrontation. Trade-off accepted: the agent has to work harder conversationally to surface the actual problem instead of naming it upfront.

5-call cap per detection cycle. Protects Vapi spend during build and demo, and forces the risk filter to actually rank customers instead of just flagging everyone above a threshold. Trade-off accepted: at higher customer volumes this cap would need to scale with a real budget, not stay fixed at 5.

WhatsApp before email as the first fallback. Higher open and response rates than email for this user base, and it's conversational rather than a broadcast channel. Trade-off accepted: requires a Twilio WhatsApp sender and sandbox approval, more setup than email alone.

Symmetric health-score deltas: +15 saved, -20 churned, 0 escalated, +5 pending. Makes the score a genuine trendline instead of a one-way ratchet that only ever goes down. Trade-off accepted: the specific point values are judgment calls, not derived from real outcome data yet.

Weekly digest instead of real-time notifications for the founder view. Real-time pings for every save or churn event create alert fatigue. A Monday-morning narrative is what a founder actually reads. Trade-off accepted: a critical churn event still waits up to a week to surface in the digest, though it's visible on the live dashboard immediately.

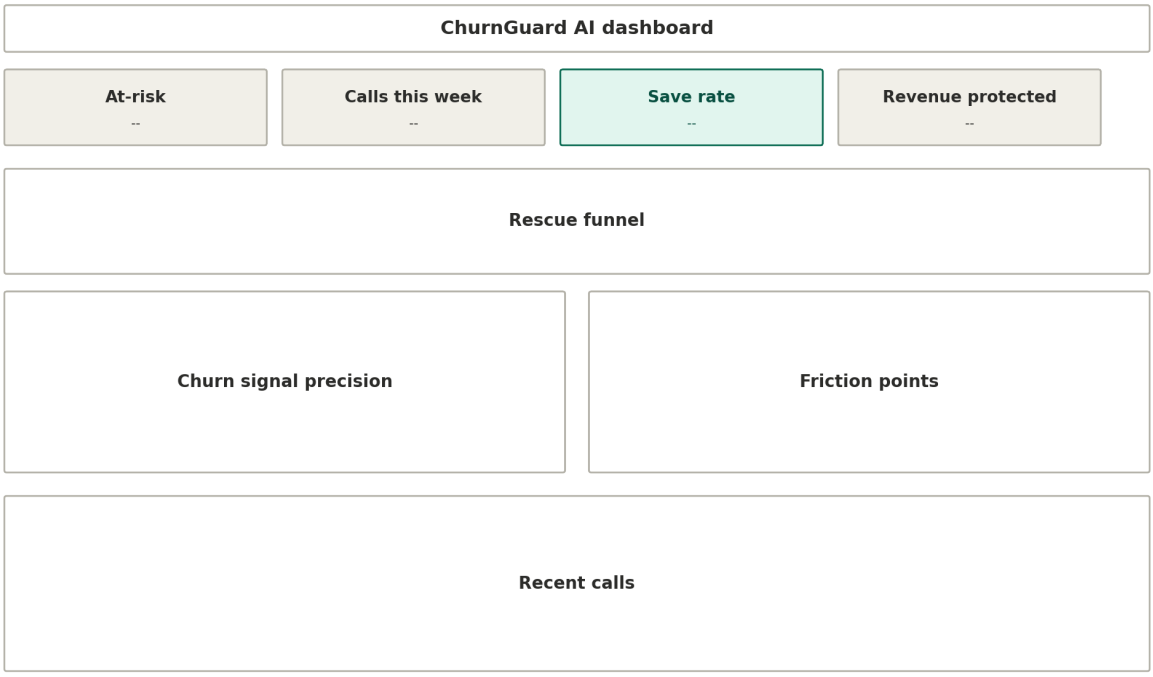
Challenges and what I'd do differently

- **Embedding latency on serverless.** The knowledge-base search endpoint downloads a ~90MB model on cold start, which risked exceeding Vercel's Hobby-plan function timeout. I scoped three fixes: move embedding to a Supabase Edge Function, use a hosted embedding API for queries only, or warm the function before calls. I picked the Edge Function route to keep the Next.js route a thin proxy.
- **Idempotency on the post-call webhook.** Vapi retries webhooks that don't return quickly, and n8n's own processing takes longer than that window. The Vercel proxy returns 200 immediately and hands off to n8n asynchronously, with a pending_webhooks fallback table so nothing is silently dropped if n8n is unreachable.
- **Sequencing inside Workflow 2.** An early version tried to fetch the customer record and their usage history in parallel branches; in practice the health-score update depends on the customer row already being loaded, so those Supabase reads had to be sequential, not parallel. A reminder that visual workflow tools still need real dependency analysis, not just a clean-looking canvas.
- **What I'd do differently:** instrument the risk filter's signal precision from day one instead of adding it as a dashboard panel after the fact, since without it there's no fast way to tell whether a given usage signal is actually predictive or just noisy.

Product surface

The dashboard is the PM-facing centerpiece, built around six panels:

- **Rescue funnel** - flagged, attempted, connected, positive outcome, saved, with drop-off between each stage.
- **Churn signal precision** - which individual usage signals actually correlate with a churn or save outcome, used to prune noisy ones.
- **Friction points** - most-mentioned issues pulled from call transcripts, aggregated across the period.
- **Channel performance** - response rates for voice, WhatsApp, and email side by side.
- **Cohort comparison** - average health-score change for intervened customers versus similar-risk customers who weren't called.
- **Recent calls** - a live table of outcome, sentiment, duration, and timestamp for the last 10 calls.



Dashboard layout (wireframe, populates from live Supabase data)

What's next

Predictive send-time. Call when a customer is statistically likely to answer based on their own usage-time patterns, instead of a flat 6-hour-after-flagging window. Should meaningfully lift the connect rate without adding call volume.

Signal confidence scoring. Feed the signal-precision panel back into the detection filter itself, so weak signals stop triggering calls automatically instead of needing a manual config change.

Self-serve playbook editor. Let a CS lead adjust the voice agent's talking points, tone, and escalation triggers from a simple form, without touching the system prompt or redeploying.

Tech stack

Layer	Choice	Why
App	Next.js on Vercel	Dashboard + thin tool/webhook endpoints only; kept deliberately lean.
Data	Supabase (Postgres + pgvector)	Relational data and the RAG knowledge base in one system.

Voice	Vapi	Managed telephony + LLM voice loop; no custom telephony stack to build.
Orchestration	n8n, 6 workflows	Visual branching, retries, and OAuth for 5 external services.
LLM	Anthropic Claude (Haiku)	Call classification and short-copy generation; fast and cheap enough to run per-call.
Channels	Twilio, Slack, Gmail, Sheets, Calendar	Existing, well-supported n8n nodes for every fallback and reporting surface.

Portfolio assets

Alongside this case study: a README with a Mermaid architecture diagram and setup instructions, a 90-second demo script covering a dashboard walkthrough, an n8n workflow tour, a recorded call snippet, and a Slack alert screenshot, plus a launch post written around the sharpest thing a real call transcript surfaces, once demo data exists to draw from.