

A real-time voice AI platform, live in a regulated retail sector

Sixteen pharmacies across eight Italian regions run an AI assistant on their own phone line, WhatsApp and web chat — each answering as itself, with its own catalogue, services, hours and staff. Separate instance, separate database, per site. Architected, built and operated by one engineer: voice pipeline, data layer, admin application, telephony and deployment.

A further thirty-plus have run the demo and cite one reservation before signing — integration with their incumbent retail management software. Not price, not trust in the assistant, not the change to their phone line. One named technical dependency, currently in development.

SITES LIVE	TENANCY	CHANNELS	TEAM
16 eight regions	1 : 1 instance and database per customer	3 voice · WhatsApp · web chat	1 sole engineer, end to end
ROLE	Architect and lead developer — real-time voice pipeline, catalogue and data layer, admin application, telephony topology, deployment, test tooling and onboarding automation.		
CORE STACK	Bun / TypeScript · Laravel 12 + Filament 5 · Next.js · MySQL · Postgres + pgvector · Redis · Python · SIP/PSTN telephony · real-time LLM, streaming TTS and vision APIs		
NOTE	<i>Client and product names withheld. Architecture and figures are as-built.</i>		

THE PROBLEM, AND WHY NOBODY HAS MEASURED IT

A pharmacy has two people behind the counter and a queue in front of it. The phone rings — someone asking whether a product is in stock, whether the pharmacist is in, whether they can book a test. Nobody can reach it. The caller hangs up and buys elsewhere, and the pharmacy never learns the call happened.

This is structural, not a staffing failure: the busiest hours at the counter are the busiest hours on the line.

It is also invisible. No pharmacy knows how many calls it loses. The line is a landline or a consumer VoIP handset, there is no PBX logging inbound, and the operator's records cover outbound. The owner has an impression and nothing else.

Which is why every site goes live on forward-on-busy-or-no-answer, with staff still answering first. The assistant only ever receives calls that had already rung out. From go-live, the missed-call count stops being an impression and becomes a row in a table.

WHAT IT DOES

The platform answers as the specific pharmacy — its catalogue, its services, its opening hours, its staff names — and either completes the request or hands the caller to a human.

CAPABILITY	WHAT ACTUALLY HAPPENS	CHANNELS
Product availability	Caller describes a product in everyday language; resolved against that site's live stock file, answered, then offered as a reservation.	Voice · WhatsApp · Chat
Reservations and orders	Takes name and contact, writes the order to the admin panel, notifies staff — item ready at the counter.	Voice · WhatsApp · Chat
Service booking	Knows the site's bookable services and their durations, computes real availability from the pharmacy's own calendar, writes a confirmed appointment into it.	Voice · WhatsApp · Chat
Prescription intake	Customer photographs a prescription; content read automatically and routed to staff for fulfilment.	WhatsApp · Chat
Escalation and transfer	Transfers to a human on request, on a named staff member, or whenever confidence is insufficient — with the pharmacy's own number shown as caller ID.	Voice

Alongside these: clinical-fact guardrails asserted from structured data rather than generated, so the assistant cannot improvise a fasting or prescription requirement; statutory after-hours duty rosters maintained per region; event and promotion announcements from a calendar the pharmacy maintains itself; and a per-site staff panel with call log, transcripts and recordings, built for non-technical owners.

Per-site catalogues run 5,700–12,400 items, imported from the incumbent retail software, normalised, enriched and re-synchronised on a repeatable monthly pass.

Go-live changes nothing the customer depends on. They keep their number, staff still answer first, and the assistant picks up only the calls that would otherwise have been lost. Setup is reversible in seconds.

PATH OF A SINGLE CALL

Pharmacy line → carrier layer → real-time voice service (audio streamed both ways) → speech-to-speech model and streaming synthesis → authenticated tool calls → business-logic API → catalogue lookup, calendar write, escalation decision → spoken response, with the call persisted as transcript and recording.

Four systems, four runtimes, one conversation — and a latency budget measured in hundreds of milliseconds across all of it. Owning every hop is what made that budget tractable.

COMPONENT	TECHNOLOGY	WHAT IT OWNS
Voice service	Bun / TypeScript, containerised	Audio streaming, turn-taking and interruption handling, model session lifecycle, tool dispatch, transfer control, call persistence
Admin & API	Laravel 12, Filament 5, MySQL	Catalogue, services, bookings, orders, settings, staff UI, the tool-call API the voice and chat layers consume

COMPONENT	TECHNOLOGY	WHAT IT OWNS
Chat & knowledge base	Next.js, Postgres + pgvector, Redis	Web chat, WhatsApp inbound webhook, thresholded vector retrieval, prescription vision endpoint
Telephony	SIP / PSTN, per-customer carrier isolation	Inbound routing, verified caller ID for transfers, per-customer separation and billing
Data pipelines	Python, model-assisted enrichment	Catalogue import and normalisation from incumbent retail software, search enrichment, regional data ingestion
Test harness	Fixture-driven synthetic caller	End-to-end call simulation with assertions against persisted results, every run recorded for review

FOUR DECISIONS THAT MATTERED

01 One deployment per customer, not one multi-tenant application

Each customer gets its own instance and its own database. No tenant identifier on any table, no row-level scoping to get wrong. The gain is blast radius, a procurement conversation that ends quickly on data isolation, and the freedom to let customers diverge without a feature-flag matrix. The cost is fleet drift, paid for with discipline instead of abstraction: a canonical repository as source of truth, every fix landing there first, and a written porting queue tracking each outstanding backport as a numbered item rather than as tribal memory.

02 Deleting the free-text prompt field

An editable system-prompt box in the admin looked like flexibility and was the largest source of behavioural drift on the platform — no bug was reproducible, because no two installations said the same thing. I removed it fleet-wide. Identity is now assembled deterministically by one code path from structured settings: capabilities, catalogue, services, hours, escalation rules. Same inputs, same instruction, every time. A whole class of bug disappeared rather than being fixed.

03 Making the system refuse

The worst failure available to an assistant like this is not silence — it is a fluent, specific, wrong answer. Telling a caller an item is in stock when it is not sends them on a wasted journey and costs the pharmacy the relationship. Early behaviour was too eager: a single matching token between what the caller said and a real catalogue entry was enough to produce a confident answer, and an invented product name could resolve to an unrelated real product. I rebuilt resolution so that confidence has to be earned from the data, with an explicit coverage threshold below which the assistant declines and offers a human instead. Refusal became the default rather than the fallback, and it is now a tuned parameter of the product rather than an accident of the model.

04 Latency as an architecture problem, not a model problem

Conversational voice dies on delay: a pause of a second reads as a broken line and the caller hangs up. Part of the budget is fixed by the model vendor and cannot be negotiated, which means the entire remaining budget has to be won back from the infrastructure around it. Every topology decision on this platform was taken against that constraint — same-region placement, the fewest hops I could get away with between carrier and model, and the vendor's floor treated as something to design around rather than argue with. The same reasoning overturned an inherited assumption about how this market has to be served at the carrier level — one that was costing money and trust, and turned out simply not to be true.

IN PRODUCTION

Sixteen sites, eight regions, one engineer. Instance provisioning, catalogue import, telephony setup and knowledge-base seeding are a repeatable per-customer sequence rather than a bespoke project each time — which is what makes a fleet of single-tenant deployments maintainable by one person at all, and what the sites waiting on the retail-software integration will arrive into.

Variable cost per minute was measured against thirty days of metered traffic rather than estimated, cross-checked against provider billing, and computed separately for competing speech vendors — so commercial pricing rests on real numbers.

Also in production: integration with incumbent retail software, and a second-language variant of the platform.

WHAT THIS MEANS IF YOU'RE HIRING

The interesting part of this system was never the model. It was the four runtimes underneath it, a latency budget spread across hops I had to own in order to control, the decision to make the thing refuse rather than guess, and production deployments that stay up while one person maintains them. That is backend and integration engineering with an LLM in the middle of it — not prompt work.

Richard Trujillo • Desemax. Senior backend and AI-integration engineer: PHP / Laravel / Symfony, TypeScript / Node / Bun, real-time and LLM-backed systems, Python data pipelines. Eleven years of backend engineering, independent since 2015. Based in Milan, working across Europe. Available for architecture, backend and AI-integration engagements.

Milano • Remoto • Partita IVA • richard@desemax.it • +39 366 301 8596 • richardtrujillo.desemax.it