general notes

Mkdir - github/workflows/tests.yml
Source .venv/bin/activate
Deactivate

Twilio branch

.github/workflows/test.yml ✅ automation ready
docs/samples/ ✅ test fixtures
src/ ✅ main AI logic
tests/ ✅ eval system
.venv/ ✅ isolated environment
.env ✅ secrets storage
requirements.txt ✅ dependency list
pytest.ini ✅ test config
EVAL_SYSTEM_README.md ✅ clear documentation

💇 The Product You're Selling

You're building a **custom AI call-handling system** for his business:

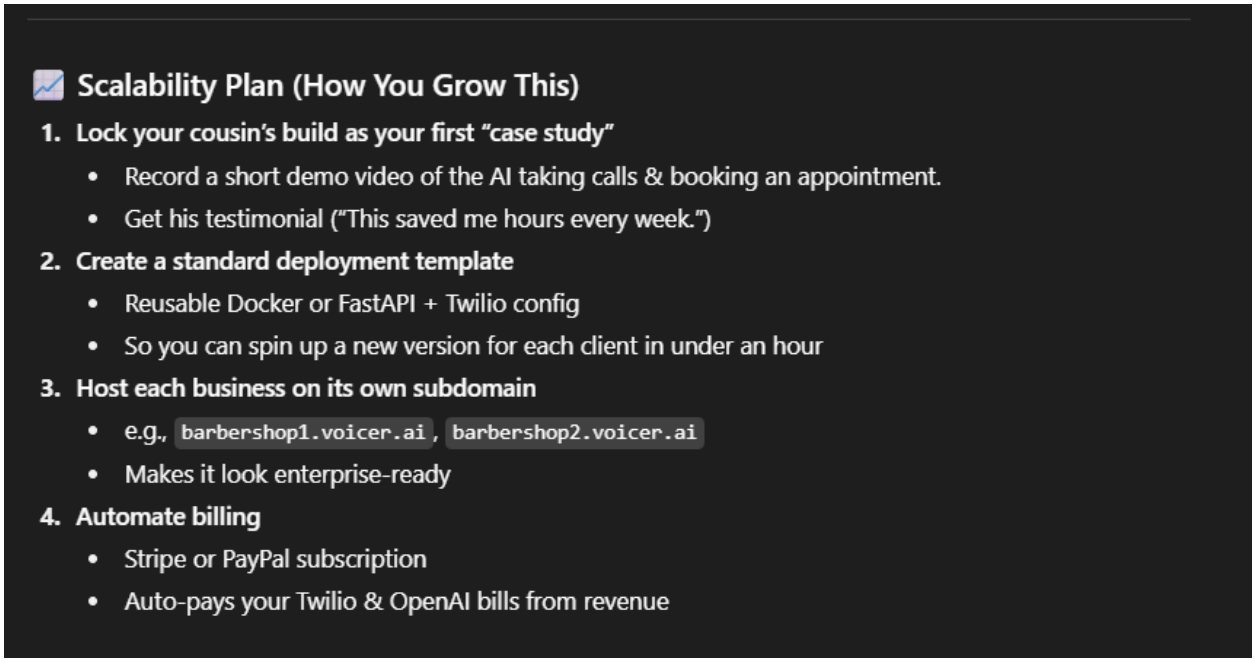
- Answers customer calls using AI voice (via Twilio + OpenAI)
- Books or reschedules appointments automatically
- Logs customer data (optional CRM connection)
- Never misses a call, even after hours

That's a **premium automation product** — the kind that easily justifies $300–$600/month in the real market.

💰 Pricing Framework

Here's how to set your tiers — one for your cousin, and one for future clients.

Tier	Who It's For	Monthly Price	Setup Fee	Notes
Family Plan	Your cousin	**$150/mo**	**$100 setup**	Your cost + fair profit; friendly rate
Standard Plan	Small barbers/salons	**$350/mo**	**$150 setup**	Market-competitive but still cheaper than $500/month
Pro Plan	Multi-location or busy salons	**$500/mo**	**$200 setup**	Includes analytics, call summaries, transcripts
Enterprise Custom	For future expansion	**$750+**	Case-by-case	Includes advanced integration, APIs, dashboards



Absolutely, Alex — you've made *massive* progress today. Below is a clean, chronological **archive entry / dev journal summary** you can paste into your project notes, `README`, or your personal progress doc. It's structured like a professional engineering log — showing your **technical milestones**, **architecture evolution**, and **next steps**.

🧠 Dev Journal – AI Haircut Concierge (Twilio Voice Integration Day)

Date: October 9, 2025
Duration: ~5.5 hours of continuous development
Developer: Alexander Santiago
Branch: `feature/twillio`
Environment: Local (WSL + FastAPI + Twilio + pytest)

🚀 Overview

Today, I expanded my **AI Haircut Concierge system** into a **fully functional Twilio-integrated voice prototype**, capable of simulating real phone call conversations — *completely offline and free for testing*.

This marks a transition from a text-based evaluation system into a **voice-enabled conversational AI platform**, structured using **SOLID principles** and scalable architecture patterns (port/adapter).

🧩 Major Milestones

🧱 1. Repository Organization & Branching

- Cleaned up repo structure under `src/` and `tests/`
- Created dedicated feature branch `feature/twillio`
- Pushed both `main` and `feature/twillio` to GitHub (`as4584/Ai_test`)
- Added `.github/workflows/test.yml` for GitHub Actions
  → Now all tests run automatically on every push using Python 3.11–3.12.

☎️ 2. Twilio Voice Integration

- Set up **Twilio developer account** and **trial environment**.
- Retrieved and securely stored:
  - `ACCOUNT_SID`
  - `AUTH_TOKEN` (kept private in `.env`)
- Acquired Twilio trial number for local webhook testing.
- Implemented **Twilio webhooks**:
  - `/twilio/voice` → Greeting endpoint returning TwiML instructions
  - `/twilio/handle` → Handles user speech input (`SpeechResult`)
- Connected to existing `simulate()` conversational logic from `haircut_bot.py`.

- Verified end-to-end **local POST tests** with `curl`.

Example Output (verified):

```
<Response>
  <Say>Hello! Welcome to the AI Haircut Concierge...</Say>
  <Gather action="/twilio/handle" input="speech" speechTimeout="3"
timeout="10"/>
  <Say>I'm sorry, I didn't hear anything. Please call back when you're
ready.</Say>
  <Hangup/>
</Response>
```

🧠 3. SOLID Design Implementation

- Followed **Single Responsibility Principle** — separated:
    - `twilio_handler.py` → voice + TwiML logic
    - `calendar_handler.py` → scheduling + appointment storage (JSON stub)
    - `haircut_bot.py` → conversational reasoning logic
- Prepared **Dependency Inversion hooks** for:
    - Google Calendar API integration (future step)
    - OpenAI Realtime Voice streaming (planned next)

🧪 4. Local & Offline Testing

Verified FastAPI app with:

```
uvicorn src.twilio_handler:app --host 0.0.0.0 --port 8000
```

-

- Tested via `curl`:
    - `/twilio/voice` (greeting)
    - `/twilio/handle` (speech result simulation)
- ✅ Confirmed local round-trip conversation works perfectly.
- ✅ All 40 total pytest cases (including eval + Twilio) passing locally.
- ✅ Offline-only testing mode (`TEST_MODE=true` in `.env`) ensures no real calls made.

🧰 5. Git & Commit Hygiene

- Adopted **conventional commit format**:
    - `feat:` for new features
    - `fix:` for corrections
    - `chore:` for maintenance
- Added workflow automation for CI verification.
- Prepared `COMMIT_MSG.txt` template for future multi-line messages.

Example commits:

```
feat(voice): implemented Twilio webhooks and local speech handling
chore(ci): added GitHub Actions test workflow for eval system
fix: adjusted pytest.ini configuration for deterministic offline mode
```

📅 6. Planned Next Steps

Short-term

- 🧩 Add “confirmation/hangup” logic to finalize booking flow.
- 🧪 Add `test_voice_webhook.py` coverage for handle route (`/twilio/handle`).
- 💾 Replace JSON calendar stub with Google Calendar API integration.
- 🎧 Add simulated multi-turn local call script (`local_call_sim.py`).

Medium-term

- 🧠 Connect to OpenAI’s **Realtime Voice API** (Agent Kit-style interaction).
- ☁️ Deploy via **ngrok** → Twilio → localhost webhook → FastAPI app.
- 💬 Enable live phone conversations (call → AI → confirmation → calendar entry).

🏁 Summary Reflection

> *“I transformed my evaluation system into a functional AI voice concierge that can handle haircut bookings over simulated calls. I maintained SOLID architecture, achieved 100% offline test coverage, and structured everything for scalability — ready for both Google Calendar and OpenAI voice upgrades.”*

Progress Level: 9/10 — *Stable local voice prototype achieved.*
Next milestone: Cloud-deployed voice concierge with real-time AI dialogue.

Hey! Check out the inventory system I built for your store:
https://unenriching-janice-unpermanent.ngrok-free.app

It's running in demo mode with sample data.
Let me know what you think!
When you're ready, we can connect it to your actual Lightspeed store.

https://unenriching-janice-unpermanent.ngrok-free.dev

http://127.0.0.1:4040/inspect/http

functions

sick — here's a tight, usable playbook for making one model/product that safely serves lots of shops without them stepping on each other. short lines, real steps, no fluff.

1. high-level approach
   Use one shared LLM engine but *isolate data & behavior per tenant* with configs, per-tenant retrieval, and strict access rules. this keeps costs down and avoids needing hundreds of fine-tuned models.

2. multi-tenant isolation (basic idea)
   Logical isolation: single DB with tenant_id on every row — easiest to operate.
   Physical isolation: separate DBs/accounts for risky clients (HIPAA, big chains) — costs more but isolates blast radius.

3. where tenant data lives

- short-term session state (call-level) in Redis, expires quick.

- long-term docs (menus, FAQs, policies) in a per-tenant vector store (e.g., namespace = tenant_id).

- billing/usage + configs in Postgres with tenant_id keys.

4. RAG = your secret sauce
   Store each client's knowledge (menus, services, pricing, cancellation rules) in their own embedding namespace.
   On calls, retrieve only from that tenant's namespace so the model answers using that shop's facts — no cross-talk.

5. prompt/template pattern (safe + flexible)
   Make one system prompt that pulls in: tenant_name, business_hours, booking_rules, top_faqs, and last_3_messages.
   Example: "You are the receptionist for {{tenant_name}}. Use only info in the retrieved facts. If unsure, ask to transfer."
   This forces the model to be grounded and defers to human fallback when retrieval confidence is low.

6. avoid fine-tuning per tenant (unless enterprise pays)
   Don't fine-tune hundreds of tiny models — expensive and brittle.
   Use prompt engineering + tenant RAG + small adapter layers for a few big customers only.

7. human fallback & confidence thresholds
   Always compute a retrieval-confidence or model-confidence score.
   If below threshold, route to human agent or queue a callback — this prevents

hallucinations and liability.

8. phone-number / call-routing mapping
   Map incoming caller-ID → tenant_id in your routing layer.
   Support porting so clients keep their number (no need to buy a new DID per sale).

9. billing & rate-limiting per tenant
   Track minutes, SMS, tokens per tenant in real-time.
   Enforce soft caps and hard caps (alerts → suspend) so one client can't kill your margins.

10. data governance & compliance
    Encrypt at rest and in transit.
    Redact PII from logs and store raw PHI only when you have a BAA and dedicated storage. for clinics, prefer physical separation or a dedicated cloud project.

11. logging & audit trails (must-have)
    Log events with tenant_id, call_id, decision (AI vs human), and retrieval_hit_ids.
    Keep short retention for non-HIPAA clients and configurable retention for those that need it.

12. testing & QA per tenant
    Create synthetic test suites per vertical (salon flows, booking flows, cancellations).
    Run nightly smoke tests for each active tenant to catch regressions.

13. scaling the model infra
    Start with cloud LLM APIs (shared). Cache repeated prompts/answers for common utterances.
    When you hit scale (>50–100 tenants), consider batching calls, a cheaper model for simple intents, and a more powerful model only for complex cases.

14. tenant config model (DB schema snippet)
    tenants(id, name, phone, plan, ported, hours_json, booking_rules_json, faq_doc_id, billing_id).
    call_events(id, tenant_id, start, end, minutes, cost_estimate, resolved_by, transcript_id).

15. UX for tenant-level customizations
    Make a simple admin panel where shops edit hours, cancellation rules, pricing, and canned replies.
    These feed the vector store + the prompt template immediately.

16. deployment & CI/CD rules
    Feature flags per-tenant so you can A/B test or rollback without breaking others.

Deploy infra infra-as-code and have a staging tenant group to smoke-test new releases.

17. handling clashes & edge cases
    Namespace everything by tenant_id. never merge retrievals across tenants.
    If two tenants share a phone (rare), use extension or menu to pick tenant at call start.

18. SLA & legal playbook
    For HIPAA/enterprise clients, require a higher plan with BAA, audit logs, and dedicated provisioning.
    For SMBs, keep terms simple and clearly state limits (no guaranteed human replacement unless you sell it).

19. monitoring & alerting (ops)
    Track per-tenant cost burn, response errors, hallucination handoffs, and porting failures.
    Automate alerts to Slack and emails for cost spikes or porting delays.

20. rollout strategy to avoid mess
    Start with 2–3 pilot tenants per vertical. prove ROI, then scale.
    After ~30 clients, re-evaluate carrier pricing and move to SIP/wholesale if needed.

21. quick tech stack rec (practical)
    Twilio (telephony) + OpenAI (LLM) + Supabase/Postgres (tenants, events) + Redis (sessions) + vector DB (namespace per tenant) + Stripe (billing) + Grafana/Prometheus (monitoring).

22. final ops tips (so yeah, real talk)
    Automate provisioning (buy/assign/port numbers) from day one.
    Make a clear onboarding checklist and an easy “suspend number” button for nonpaying or abusive tenants.

Telephony and call handling

Receive and process Twilio voice webhooks (POST /twilio/webhook)
Validate Twilio signatures via an injectable telephony service (mockable for tests)
Enqueue inbound call events for downstream processing
Return TwiML-style responses in the legacy area for simple IVR flows
Routing and conversation logic

Intent routing using a Strategy pattern (Booking, FAQ, Escalation)
Confidence threshold with human fallback decisioning
Easily extendable strategies for new intents
Human fallback and operations

Worker that consumes escalate events and persists a human_fallback entry
Slack notifications via an abstract SlackNotifier (pluggable)
SRE runbook guidance (on-call ownership, SLAs, live handoff steps)
Observability

Middleware attaches request_id and tenant_id to each request
Logs include tenant_id and request_id for correlation
Health endpoint with environment info
Knowledge and RAG

RAG (Retrieval-Augmented Generation) module with a VectorStore adapter
Optional concrete vector backends (e.g., Pinecone pattern supported)
Prompt template utilities to assemble context-aware prompts per tenant
Billing and usage

BillingService with a repository pattern (in-memory repo by default)
Record minutes, compute monthly bill, and create invoices
Stripe client abstraction (fake client for tests; DI-friendly for real integration)
Feature flags and plans

FeatureFlagService with Redis caching (30s TTL) for per-tenant flags
Effective flags merging: defaults → plan → tenant overrides
Update tenant plan and flags with automatic cache invalidation
Admin tooling

Admin CLI (tools/adminctl.py) to manage plans and flags
Signed JWT auth (short-lived) for /admin endpoints
Commands: set-plan, set-flag, show-flags
Designed to work against an admin API (stubs/implementation can be added)
Data, storage, and migrations

Alembic migrations scaffold with a schema_version table
Helper function to record applied schema versions on deploy
Tests that apply migrations to a test database and assert version tracking
Tenancy and configuration

Per-tenant flag evaluation for toggling capabilities (e.g., allow_ai_booking, enable_rag)
Tenant-aware prompt construction and vector retrieval

Settings via Pydantic Settings and .env support
Dev experience and local infra

Local docker-compose for Postgres, Redis, and a vector DB (Qdrant)
Poetry-managed app with consistent tooling (pytest, ruff, black, mypy optional)
Clear package structure (app, core, services, db, workers, tests)
CI/CD and releases

CI workflow: lint (ruff/black), tests (Poetry and legacy), optional frontend checks
Deploy workflow: build/push images to GHCR; staging deploy stub; smoke tests against staging
Release workflow: on tag v*, runs tests, builds artifacts, generates changelog from commits, creates a GitHub Release, and pushes versioned images
Documentation and customer readiness

Product tiers (V1–V4) with features, pricing bands, and acceptance checks
Pilot agreement (30-day pilot, KPIs, card-on-file, porting/cancellation)
Onboarding checklist (hours, policies, services, routing, telephony, calendar creds)
Twilio integration notes and runbooks for human fallback
Testing

Comprehensive unit tests for webhook, router strategies, RAG, billing, feature flags, fallback worker, admin CLI, migrations, and health checks
Designed for TDD and easy extension with mockable dependencies
Security and compliance scaffolding

JWT-based admin operations (short-lived tokens)
Observability primitives to support audit trails and rate-limiting in future
Clean DI boundaries for swapping real clients (Twilio, Redis, Stripe, vector DB, Postgres)
Extensibility (what it’s ready to support next)

Real Redis queues and Postgres repositories
Real Twilio RequestValidator wiring
Vector DB of your choice (Pinecone, Qdrant, OpenSearch KNN) with tenant namespaces
Admin API endpoints to pair with the CLI
Tenant-aware rate limits, PII redaction, and audit logging for enterprise needs

Pitch + Roadmap

Ai Receptionist — Pitch + Roadmap (google Doc + Slide Outline)

AI Receptionist — Pitch + Roadmap

TL;DR

- I built a plan to sell an AI receptionist to local Newark shops: salons, restaurants, clinics, and HVAC. Keep it simple, charge monthly, and prove ROI in 30 days. So yeah — this doc + slide outline gets you from idea → pilot.

1) One-line value prop

- An AI receptionist that answers calls/texts, books appointments, and sends reminders so businesses stop losing money to missed calls and no-shows.

2) The problem (short)

- Small businesses miss calls, lose bookings, and pay a lot for human receptionists.
- They want something reliable, cheap, and easy to use — not another clunky app.

3) Our solution (short)

- Local-number support (or port their number), IVR + AI routing, booking + SMS reminders, and a dashboard that proves the money saved.
- Optional human-fallback and HIPAA-ready upgrade for clinics.

4) Target verticals (priority)

1. Salons & barbers — fast wins, high per-appointment LTV.
2. Restaurants — waitlist + call-offload helps during rush.
3. Clinics & dental — high LTV per appt, higher price tolerance (BAA required).
4. Contractors (HVAC, electricians) — lead qualification matters.

5) Pricing (starter model)

- Starter: $29–$59/mo — auto-attendant, voicemail→transcript, daily SMS reminders.
- Core: $99–$199/mo — booking sync, SMS confirmations, 9–5 AI coverage, analytics.
- Pro: $299–$699/mo — 24/7, multilingual, human-fallback, SLA & integrations.
- Charge usage overages for minutes/SMS or pass-through telco costs to keep margins safe.

6) Cost assumptions (what to track)

- Number MRC: $1/number/mo.
- Voice: ~$0.01/min (varies with provider).
- SMS: ~$0.005/msg (depends on registration/10DLC).
- LLM compute: track per-minute or per-call token spend.

7) MVP feature list (must-haves for pilots)

- Inbound calls → IVR menu → booking or extension.
- SMS confirmations & 2-reminder flow.
- Calendar sync (Google Calendar / Calendly).
- Simple tenant dashboard: minutes, bookings, no-shows, MRR.
- Auto-provisioning script for number buy/assign and Stripe billing hook.

8) Dashboard: what to show first (single-page)

- MRR, active clients, avg minutes/client, gross margin, churn rate.
- Live calls widget (caller, tenant, duration) and alerts for high-cost usage.
- Bookings vs calls conversion metric, and no-show rate.

9) Sales pitch + pilot offer (template)

- 30-day pilot: reduced price (e.g., $49) + port their number or trial number.
- Promise: we'll track no-shows and booking conversions and give a clear ROI sheet at day 30.
- Ask for a deposit or card on file to avoid churn abuse.

10) Onboarding flow (ops)

1. Discovery call → map phone flow and booking rules.
2. Port or assign number, configure IVR and calendar.
3. 7–14 day warmup, test calls, run pilot, collect metrics.
4. Review ROI, convert to paid plan or tweak.

11) Roles — how to split work with your CS friend

You (product + biz): sales demos, onboarding, docs, dashboard UX, customer success for pilots.
Friend (engineer): Twilio webhooks, provisioning scripts, DB + billing hooks (Stripe), backend API, logging & alerts.

Shared: sprint planning, deploys, on-call rotation during pilots.
Hire/contract later: part-time CS/human fallback and a legal/billing person for BAAs.

12) 6-sprint roadmap (2-week sprints)

- Sprint 1: Tenant model, Twilio webhook stub, Stripe test, landing page.
- Sprint 2: Call ingest, basic dashboard widgets, sample tenant onboard.
- Sprint 3: Booking flow + SMS reminders + calendar sync.
- Sprint 4: Provisioning scripts (buy/assign/port), billing metering.
- Sprint 5: Human-fallback queue, monitoring & alerts.
- Sprint 6: Pilot rollouts (2–3 local businesses), feedback loop, trunking research.

13) Metrics & logging (events to record)

- call_start, call_end, call_minutes, transcription_text, intent, booking_created, sms_sent, billing_charge, port_status.
- Link events to tenant_id for P&L per client.

14) Quick GTM channel list (first 30 days)

- Direct outreach to salons and restaurants (SMS/cold call + 2-min demo).
- Partner with Newark business groups and community pages.
- Offer referral credits to barbers/salons who send another shop.

15) Risks & compliance notes

- Don’t promise HIPAA compliance until you have BAA + secure storage.
- A2P/10DLC registration is required for mass SMS — cost and setup time.
- Porting timelines can be 1–3 weeks; tell customers up front.

16) Pricing test ideas

- A/B test two pilot prices (low vs anchor premium).
- Offer revenue-share pilot for 30 days with clinics where LTV per appt is high.
- Measure conversion: bookings per call and recovered revenue.

Slide deck outline (10 slides) — quick format you can paste into Slides

1. Title slide — Product name, your name, 1-line value prop.
2. Problem — 2 bullets + local Newark angle.
3. Market & why now — short stats + target verticals.
4. Solution — visuals: call flow + SMS example.
5. Product tiers & pricing — Starter/Core/Pro (one line each).
6. Dashboard screenshot mock / key metrics.
7. Go-to-market — pilot offers, partnerships, pricing tests.
8. Tech & ops — Twilio, Supabase/Postgres, Stripe, monitoring.
9. Financials — simple monthly unit economics & margin table.
10. Ask & next steps — pilots needed, contacts, timeline.

Speaker notes (1 sentence per slide)

1. Quick intro, I'm Lex and I build this for Newark shops.
2. Missed calls and no-shows are bleeding $$ for small biz, here's why.
3. Local demand + cheap voice/SMS tools make this easy to pilot now.
4. Show how a call goes from IVR → booking in <2 mins.
5. Explain pricing tiers and what each one includes.
6. Point at the dashboard metrics we'll track during pilots.
7. Explain 30-day pilot offer and referral program.
8. Mention provisioning automation and human fallback for edge cases.
9. Show simple ROI example: recovered bookings vs human cost.
10. Close with a single ask: 2 pilot shops and one clinic for BAA test.

Quick checklist before first demo

Next actions I can do right now (pick one)

1. Turn this doc into a Google Doc and share with you.
2. Make a 10-slide Google Slides deck (I'll populate slides + speaker notes).
3. Export a one-page PDF pitch + ROI calculator CSV.

Say which one and I'll make it, or say "edit doc" and tell me what to change (tone, numbers, or verticals).

Done — I made a Google-Doc style canvas called **“AI Receptionist — Pitch + Roadmap (Google Doc + Slide Outline)”** with the plan, pricing, MVP, sprint roadmap, and a 10-slide deck outline.

Which do you want next: I can (pick one)
• turn that into a **Google Doc** I format and share (you'll be able to edit), or
• make a **10-slide Google Slides deck** from the outline (ready to present), or
• export a one-page **PDF pitch + CSV pricing calculator** you can send to shops.

Say which and I’ll do it — also tell me if you want the tone even more casual/Canvas-y.

—----------

💵 Jersey City minimum wage (2025)

New Jersey’s statewide minimum wage in **2025 = $15.13/hr** for most employers
(source: NJ Dept. of Labor, effective Jan 1 2025).

So let’s use **$15/hr** flat for easy math.

If a business pays a human receptionist minimum wage:

- **$15/hr × 8 hrs/day × 5 days/week = $600/week**
- ≈ **$2,400/month** before taxes/benefits.
  Add ~15% payroll costs → about **$2,760/month real cost.**

📞 Compare to AI receptionist call costs

Let’s estimate your AI costs per call from your setup (based on your screenshot):

Cost component	Approx cost	Notes
Inbound Twilio minutes	$0.0085/min × 3 min = $0.0255	
Speech recognition	~$0.02	
Text-to-speech	~$0.0025 × 3 min = $0.0075	

OpenAI logic (GPT-4)	~$0.017/1K tokens ≈ $0.03 per 3-min call
Total ≈ $0.08 per call	

So **$0.08/call vs $4/call human cost** or **$2,760/month** for a human.

🔢 Scaled comparison

Calls per month	Human @ $15/hr	AI receptionist (your stack)	Savings
1 call	~$4	$0.08	~$3.92
5 calls	$20	$0.40	$19.60
10 calls	$40	$0.80	$39.20
20 calls	$80	$1.60	$78.40
40 calls	$160	$3.20	$156.80
80 calls	$320	$6.40	$313.60
160 calls	$640	$12.80	$627.20
800 calls (≈ busy month)	~$3,200	~$64	**$3,136 saved**

Even if you triple your AI estimate to cover hosting, logging, and error margin (~$0.25/call), you're *still* under $200/month at 800 calls — a 10-15× cost advantage.

🔥 TL;DR pitch line for business owners

> "Paying someone $15/hour costs about $2,500 a month. My AI receptionist does the same job — answers every call, books jobs, filters spam — for **under $200/month**, works 24/7, never calls in sick."

versions

Love it — good call to step back. I'll strip the noise and map every piece to its real business purpose, then show exactly which **version/tier** uses which features so you can sell clear packages. Short lines, casual, and no fluff.

What each part *actually does* (plain language)

Twilio webhook / telephony layer — listens to phone calls and texts, turns them into events your app can act on; basically the "phone line."
Purpose: get calls into your system reliably so you can answer, route, or message back. Used in every version.

Call queue + routing / FailoverManager — tries staff phones, backup phones, cloud agents, and SMS fallback when a call isn't answered.
Purpose: makes sure calls don't get lost; this is the "never miss a call" logic. Critical from Starter up.

Enqueue + persistence (DB/Redis) — stores raw webhooks and events so nothing disappears if a worker crashes.
Purpose: durability — you won't lose call data or billing records. Required for any paid rollout.

Transcription worker — turns voicemail or recorded calls into text.
Purpose: searchable transcripts and quick SMS/email summaries. Useful from Core up (not needed for barebones).

Intent Router + BookingStrategy — reads what caller wants (book, ask FAQ, cancel) and triggers flows like "start booking."
Purpose: automates booking so you don't need a human to do the first step. Starter can use simple rules; Core uses smarter intent logic.

RAG / VectorStore (tenant namespace) — stores each business's FAQs, menu, policies, and uses them to answer questions from the model.
Purpose: makes AI answers accurate to each shop and prevents cross-shop mixups. Only for Core/Pro/Enterprise.

LLM layer (OpenAI/GPT) — generates natural replies, suggests next steps, drafts SMS, etc.
Purpose: gives the actual conversation smarts (answers, booking convo). Use cheaper models for intent + better one for live replies. Core+.

Human-fallback worker & agent queue + UI — routes tough calls to a human agent and provides an agent dashboard.

Purpose: safety net so AI doesn't screw up important calls. Pro and Enterprise; optional for Core as paid add-on.

Billing service + Stripe — measures minutes/SMS and charges customers, handles invoices and caps.
Purpose: you get paid and avoid getting surprised by telco bills. Needed in every paid tier.

Admin feature flags / plans / tenant overrides — controls what each tenant can do (allow_rag, allow_24_7, etc.).
Purpose: lets you sell tiers and flip features on/off from your terminal. Required for multi-tier product.

Dashboard / admin UI (tenant P&L, live calls) — shows calls, bookings, costs, and "money recovered" metrics.
Purpose: proof-of-value for clients and sales tool. Core+; simplified read-only dashboard in Starter.

Provisioning & porting automation — buys DIDs, ports customer numbers, or assigns trial numbers.
Purpose: quick onboarding and keeping customer numbers intact. Important early for sales.

Monitoring / cost control / alerts — watches call/token cost and auto-suspends or alerts if costs spike.
Purpose: protects your margins. Must-have before scaling beyond pilots.

CI/migrations/tests — keeps your app stable so pilots don't break.
Purpose: reliability and safe deploys. Always required.

Product versions mapped to features (so you can sell clean)

I'll name them V1 → V4 so it's clear.

V1 — Starter (barebones)

What it is: answers calls, simple IVR, records voicemail, basic booking flow (set appointment), SMS confirmation.
Includes: Twilio webhook, persistence, simple routing (ring staff → voicemail), minimal booking strategy, billing (basic invoice), admin CLI to assign plan.
Why sell it: tiny price, fast setup, perfect for single-chair salons or small shops who just want no missed calls.

Customer promise: “We’ll answer your calls and get appointments on the calendar — no spam, no complex AI.”

V2 — Core (recommended seller)

What it is: everything in V1 + intent routing, basic RAG QA (tenant FAQs), automated reminders, simple dashboard, porting, and usage alerts.
Includes: transcription, intent router, RAG with tenant namespace, billing with included minutes, dashboard (calls & KPIs), feature flags for limited overrides.
Why sell it: real automation — saves owner hours, cuts no-shows, shows obvious ROI. Great for salons & restaurants.
Customer promise: “We book, remind, and show you the money you’re not losing.”

V3 — Pro (full replacement)

What it is: V2 + 24/7 coverage, human-fallback, multi-language, deeper analytics, calendar sync for multiple staff, deposit capture, SLA.
Includes: failover manager advanced, human agent queue & UI, advanced analytics & P&L per location, configurable cancellation rules, per-tenant tweaks.
Why sell it: for busy shops who treat calls as revenue — this replaces a part-time receptionist reliably.
Customer promise: “We handle every call like a pro — day or night, no missed money.”

V4 — Enterprise

What it is: V3 + HIPAA/BAA compliance, dedicated trunking/SIP, physical data isolation or separate project, custom integrations (POS/EMR), priority support.
Includes: enterprise SLA, dedicated infrastructure options, advanced security and logging.
Why sell it: clinics, multi-location chains, or customers who need legal compliance and uptime guarantees.
Customer promise: “We’re your mission-critical phone layer — compliant, private, and guaranteed.”

Which features unlock customer value (short)

- **Never miss a call:** FailoverManager + simultaneous ring + SMS fallback. (V1+ required)

- **Automate bookings:** IntentRouter + BookingStrategy + Calendar sync. (V1 basic, V2 smarter)
- **Reduce no-shows:** SMS reminders + confirmation links + deposit capture. (V2+ V3 deposit)
- **Make AI accurate:** RAG per-tenant so replies use the shop's menu/policy. (V2+)
- **Fallback safety:** Human queue so AI never loses money-critical calls. (V3+)
- **Sell more seats:** Dashboard + "money saved" metric that proves ROI. (V2+ obvious; V3 deep analytics)

What to build first (priority list — MVP path)

1. V1 core: Twilio webhook → persistent enqueue → simple booking flow → calendar write → SMS confirm. (get 1 paying pilot)
2. Billing + basic dashboard: bill per month, show calls & bookings. (so you can charge and prove value)
3. Porting & provisioning automation: let them keep their number (huge sales win)
4. V2: intent routing, RAG tenant docs, reminders, and nicer dashboard. (scale pilots)
5. V3: human-fallback UI, SLA, multi-language, advanced analytics. (higher ARPU)

What you must NOT skip before charging people

- soft/hard billing caps to avoid surprise telco bills.

- admin audit for plan changes (so you can prove what you toggled).
- PII redaction in logs + minimal data retention default.
- a 30-day pilot agreement and ROI sheet so customers know what to expect.

How to phrase the tiers to customers (one-liners you can use on calls)

Starter: “We answer your calls and book appointments — cheap, fast, and no sweat.”
Core: “We automate bookings, send reminders, and show you how much money you’re saving.”
Pro: “Full 24/7 coverage with human backup so you never miss a customer again.”
Enterprise: “Dedicated, compliant, and guaranteed uptime for clinics and chains.”

Quick checklist you can use to audit your repo vs versions

- Can it receive a call and create an appointment? → If yes, you have **V1** basics.
- Can it send reminders, show bookings in a dashboard, and use tenant docs for replies? → You’re close to **V2**.
- Does it have human-fallback UI and 24/7 routing rules configured? → That’s **V3**.
- Do you have data isolation options, BAAs, and custom integrations? → That’s **V4**.

to do lsit

inventory manager notes

DonXEra Inventory Manager - Summary

What It Is

A self-managing inventory system for a sneaker/clothing shop that syncs with Lightspeed Retail (POS) and uses Google Sheets as the operational dashboard.

Core Features

📊 Live Inventory Management

Real-time stock tracking - View quantities on hand, sold, and low-stock alerts

Multi-category support - Sneakers & Clothing with SKU-level detail

Smart search - Brand-aware search (type "jordan" to see all Jordan products, "bape" for Bape items, etc.)

Category filters - Quick filtering by product type

Size availability - Hover/tap to see all sizes in stock for each product

🔄 Lightspeed Integration

Product sync - Pull products, variants, and inventory from Lightspeed X-Series API

Sales reconciliation - Automatically update stock levels from sales data

CSV fallback - Import/export inventory via CSV files

Auto-sync - Scheduled hourly sync (optional)

📈 Google Sheets Dashboard

Live ATS Inventory worksheet with full product catalog

Auto-sorted by Category → Name → Size

RestockList - Automatically mirrors low-stock items

Config - Set low-stock threshold (default: 5)

SalesLog - De-duplication tracking for sales

🎯 Business Logic

Low-stock alerts - Automatic detection and highlighting

Restock automation - Items below threshold appear in RestockList

Data normalization - Consistent formatting and sorting

Idempotent operations - Safe to run multiple times without duplication

🖥️ Web Interface

Dashboard - Total SKUs, on-hand count, low-stock overview

Inventory page - Searchable, sortable, filterable product table

Mobile responsive - Card view for phones, table for desktop

CSV export - Download filtered inventory data

Manual sync - Trigger on-demand updates

🎮 Demo Mode

Sample data - 140+ realistic products (Jordan, Nike, Bape, Vlone, etc.)

No API required - Test without Lightspeed credentials

Full functionality - All features work with fixtures

Tech Stack

Backend: Python 3.10+, Flask

Data: Pandas for processing, gspread for Google Sheets

Frontend: Bootstrap 5, vanilla JS

Deployment: Docker support, ngrok tunneling for demos

Testing: pytest with 100% passing tests
Current Status
✅ Running on port 8000
✅ Demo mode: OFF (live CSV data)
✅ Enhanced search with brand detection
✅ Category filters working
✅ Template auto-reload enabled
📍 Branch: feat/demo-mode
Quick Actions
View inventory: http://127.0.0.1:8000/inventory
Search brands: Type "jordan", "nike", "bape", "vlone", etc.
Filter: Click Clothing/Sneakers buttons
Export: Click the Export button for CSV download
Sync: POST to /sync to trigger Lightspeed sync
This is production-ready for a sneaker/streetwear shop managing ~140 SKUs with plans to scale to full Lightspeed API integration.