CHRIS PHONE NUMBER: 6692204908

RICCARDO ROBERT, jay and I are at one of the tables closer to taylor swift (swift ventures).

devpost link: https://devpost.com/software/1308572/joins/kAaxCdEwOXEBy80o8mf8aA

Project Name: OTOSCRIBE, VOXPENSE, OTOMEDI

Value proposition:

- Cut down unnecessary time to do simple tasks that can be automated.
- Tens / hundreds of missed calls pile up by the end of the day at hospital, time consuming to listen to every call, write down important material, execute the requests of the patient, especially when:
    - Majority of missed calls consist of: prescription refill requests, schedule adjustment, request to relay message to doctor. (eg, I don't feel well)
    - To execute the steps requires a series of clicking and reading, that is more time consuming than what the action requires.
- Product goal:
    - Goes through every voicemail and collects it into a dashboard where one approval click from a certified healthcare worker (CHW) will execute repetitive tasks, and organize edge cases for CHW to read and take action.
    - Organize unfinished and finished tasks into the same dashboard.

Mentor notes:
- regression testing

Multi-agents structure will probably just use claude maybe incorporate agentspan.

- **Intake Agent** -> receives inbound call simulation (or text input for demo), extracts: patient name, reason for visit, preferred times, insurance, urgency signal
    - Sponsor to use for voice to text: Deepgram
    - This agent's job is to strictly extract the fields from the stt json
        - First and last name
        - Date of birth
        - Phone number
        - Details of request
        - Insurance

```
{
```

```
"first_name":    { "value": "Maria",        "confidence": 0.98 },
"last_name":     { "value": "Gonzalez",     "confidence": 0.98 },
"date_of_birth": { "value": "1985-03-04",   "confidence": 0.95 },
"phone_number":  { "value": "+14155550123", "confidence": 0.90 },
"request": {
  "type": "refill",
  "orders": [
      "paracetamol",
      "vitamin A"

  ]
  }
}
```

- **Combine this with intake agent Eligibility Agent** -> checks if the information the patient presented is valid i.e crosscheck the json output contains the needed information i.e First name dob phone number, details of requests and check patient's insurance is accepted, flags missing info, generates a "what we need" checklist
    - Inbound calls that qualify for automation should include: First and Last Name, date of birth, phone number (or other contact information), details of request.
    - Inbound calls that don't qualify goes through:
        - Manual review by the maintainer
        - Perhaps use poke to text the caller and ask for the missing details
    -

- **AGENTS THAT CHECK IF THE REQUEST FALLS INTO ONE OF THE THREE ACTIONABLE ITEMS (**prescription refill requests, schedule adjustment, request to relay message to doctor.)
    - **Prescription Requirement Eligibility Agent**-> checks whether patient meets requirements to get a refill:
        - Has a previous visit within the past 6/12 months (depends on doctor but usually chooses 6 or 12 months based on age).
        - Has an upcoming visit within the next year.
        - Has been prescribed of the medication before with the dosage and instructions identical to before
        - Has not been prescribed conflicting medication (a warning will usually pop on the screen of MyChart/ EclinicalWorks).
        - Agent's Goal: same role as the scheduling one, but for refill requests: "Can this prescription refill go through automatically, or does a human need to look at it first?"
        - It checks four things, all against the real database:
            - Recent visit — has the patient been seen within the required window? The window itself isn't fixed: patients under 65 need a

visit in the last 6 months, patients 65+ get 12 months (mirrors real clinic policy where chronic, established patients get more leeway).
    - Upcoming visit — is there a visit already scheduled within the next year? No upcoming visit means no plan for follow-up care, even if recently seen.
    - Identical dosage match — has this exact medication been prescribed before, at the same dosage and instructions? A first-time prescription or a dosage change can't be auto-refilled.
    - Drug interaction warning — is the patient currently on another medication known to conflict with the one requested? This is the one check that's a warning, not a blocker — it mirrors the popup MyChart/eClinicalWorks shows; it doesn't stop an otherwise-eligible refill, it just flags it for the physician to notice.
- Expected output
    - Eligible (3 of 3 hard requirements pass): eligible: true, status: "pending_approval", a proposed_action describing exactly what to write if a CHW approves it (medication, dosage, instructions, prescribing provider). If there's also a drug interaction, it shows up in agent_checks as conflict: true / conflict_medication: "..." — visible to the reviewer, but doesn't change the eligible/pending_approval outcome.
    - Ineligible (any hard requirement fails): eligible: false, status: "escalated", proposed_action: null, and flagged_reason spells out exactly which requirement(s) failed (e.g. "no visit within the required recent-visit window; no upcoming visit scheduled within the next year") — so staff don't have to guess why it stalled.

- Test cases
    - Pure logic tests (no database) — confirm the age-based window selection (under 65 → 6 months, 65+ → 12 months), that a visit just inside the window passes and one far outside fails, that identical dosage matches but a changed dosage doesn't, that never-prescribed-before fails outright, and that the known conflict pair is flagged while an unrelated medication isn't.
    - Service tests with a fake database — two of them deliberately mirror the real seeded patients to prove this isn't just theoretically correct:
        - Maria Gonzalez's scenario — eligible, but the Lisinopril/Amlodipine interaction still surfaces as a warning.
        - James Okafor's scenario — escalated, because his last visit was 19 months ago and he has nothing upcoming.
        - Plus: never-prescribed-before escalates; a missing patient raises a clear error instead of crashing.

    - Write-back tests — confirm that passing a task_id writes the result into that tasks row, and critically, that it merges with whatever another agent (e.g. the insurance Eligibility Agent) already wrote there instead of erasing it. Also confirm that omitting task_id touches nothing — safe to preview without committing.

- **Schedule Adjustment Eligibility Agent** -> checks whether patient meets requirement to adjust schedule:
    - Check if there are conflicts within the existing calendar (ie another patient already booked in preferred time, doctor availability, holiday/ non-work days).
    - Check if a patient has been continuously asking for a schedule change (sometimes they do this to meet the requirement of prescription refill without showing up). If more than 2 consecutive requests, must call patient to confirm reason for change.
    - **Status**: Completed
    - **Expected Output:**

    When you call this agent with a patient, a provider, and a requested time slot, you get back one of three outcomes:

    1. Clear to proceed — no conflict, patient isn't a repeat-rescheduler. The output says eligible: true, status: "pending_approval", includes a suggested_timeslot confirming the requested time is actually bookable, a proposed_action describing exactly what to write to the calendar if a CHW approves it, and a plain-English summary from Claude.

    2. Needs a different time — the slot is taken, outside provider hours, or on a holiday. Output says eligible: false, but status stays "pending_approval" (not escalated — it's a routine fix, just needs a different slot). suggested_timeslot and proposed_action are both empty, because picking the alternate time isn't this agent's job — that's the separate Scheduling Agent's responsibility.

    3. Needs a phone call — the patient has rescheduled more than 2 times in a row without ever completing a visit (the trick some patients use to keep "renewing" prescription eligibility without showing up). Output says eligible: false, status: "escalated", and flagged_reason explains why — staff need to call and confirm the real reason before touching the schedule again.

    Where the output goes for the next agent
    It's written into the tasks table in Supabase — specifically, it updates the one row that the Intake Agent already created when the voicemail came in (identified by

task_id). The columns it writes are status, agent_summary, agent_checks, proposed_action, and flagged_reason.

There's no direct agent-to-agent messaging — every agent reads and writes that same shared tasks row.

- Testing:
    - Tested correctly! With Robert Martinez case in DB

- **Message Relay Eligibility Agent** -> checks whether message is worth relaying:
    - Filters out unnecessary requests that do not fall into:
        - Update on status becoming negative, unreasonable reaction to medication, any and all discomfort, request for some form of accommodation.
    - Returns whether it's worth relaying to doctor, patient first name, last name, dob, the draft of message to be relayed to doctor. Concise message.

Tech Architecture:

- Demo won't have an actual phone number to call, however it's possible to implement it via a voip number + a service like twilio, for our demo we're just gonna upload audio messages as "voicemails"

Test Cases:

1. Maria Gonzalez — fully automated, gets the SMS
Lisinopril refill. Insurance valid → passes the gate. Prescription Eligibility Agent: recent visit ✓, upcoming visit ✓, dosage matches ✓ → eligible: true, status: pending_approval. Also flags a drug interaction with her active Amlodipine — but that's a note, not a blocker. CHW sees the one-click approval queue with the interaction warning visible, clicks approve → backend inserts the new prescriptions row → (once built) Confirmation Agent texts her "refill ready for pickup." This is the case that proves a warning and an escalation aren't the same thing.

2. James Okafor — escalated, never reaches a human-approval click
Metformin refill. Insurance valid. Prescription Eligibility Agent: last visit 19 months ago, no upcoming visit → eligible: false, status: escalated, flagged_reason spells out exactly why. No

proposed_action exists, so there's nothing for a CHW to one-click approve — they have to actually call him to book a visit first. No SMS either, since nothing got approved.

3. Linda Chen — escalated before the type-specific agent even runs
Wants to reschedule. The insurance Eligibility Agent runs first, finds Kaiser Permanente isn't accepted, and short-circuits straight to escalated — the Schedule Adjustment Eligibility Agent never even gets called. This is the case that shows why agent order matters: don't burn a Claude call checking calendar conflicts for a patient who can't be seen here at all.

4. Robert Martinez — neither escalated nor approved, stuck in between
Wants June 24, 3pm — already booked. Schedule Eligibility Agent: conflict: true, but only 1 prior reschedule (under the abuse threshold) → eligible: false, status: pending_approval (not escalated). This sits in a third bucket: "needs a different slot, not a human call." The not-yet-built Scheduling Agent is supposed to pick this up, search providers.availability + appointments, and propose June 25 9am — which I manually verified works when I fed that slot in by hand. Once that's wired up, CHW approves the proposed slot, and Confirmation Agent texts him the new time.

5. (Hypothetical) Someone who's rescheduled 3 times in a row — escalated for a different reason than Linda or James
Same Schedule Eligibility Agent, but requires_manual_call: true this time → escalated with flagged_reason about calling to confirm the real reason. Three different patients can all land in "escalated," but the reason — insurance, missed visit, suspected no-show pattern — is what tells the CHW what to actually do next. The dashboard needs to surface flagged_reason, not just the status badge.

6. Priya Sharma vs. a hypothetical "just calling to say hi" voicemail — the filter that doesn't exist yet
Priya reports dizziness/nausea after starting Sertraline — that's an adverse reaction, one of the four categories worth relaying. The (unbuilt) Message Relay Eligibility Agent should pass this through to a CHW-reviewable task. A voicemail that's just chit-chat with no actionable content should get filtered out silently — closed out without ever bothering a human. Right now nothing does that filtering; every message_relay voicemail would currently just sit there unsorted.

Resources:
Hackathon guide link

Robert: 626-677-7409

https://miro.com/welcomeonboard/SkVGcXNLd1VpczZjdFdjZkdOcCs3c3Q5YlNCSG1oaGtqUUI5RW1YZWlTT01ncGNvbytCdmFWSGI5TjhEVVhWTVEyOEhiZTY1YjFlcVVvdVY2blpSU3dBbG

9DaDY4ZW1GQmNvSElZSkpLL0VGbjRndWpxdDVwYjF2UklJKytCaGRBd044SHFHaVlWYWk0d3NxeHNmeG9BPT0hdjE=?share_link_id=571292253523

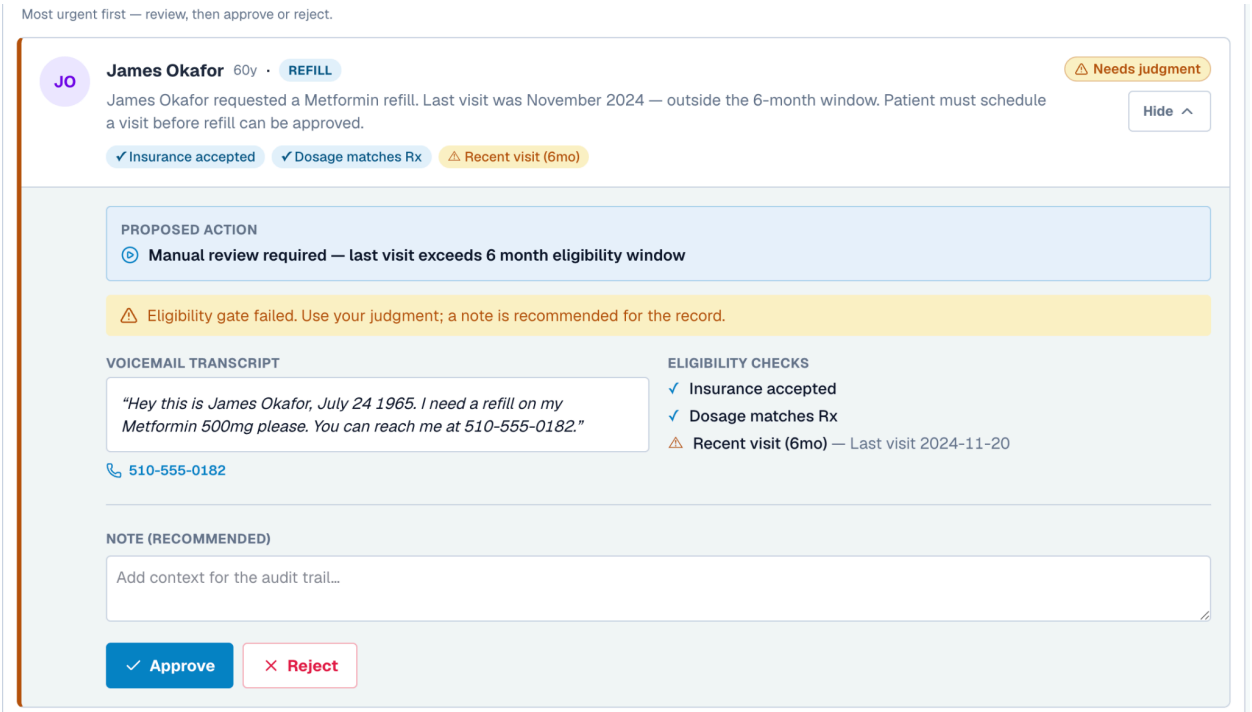
API testing

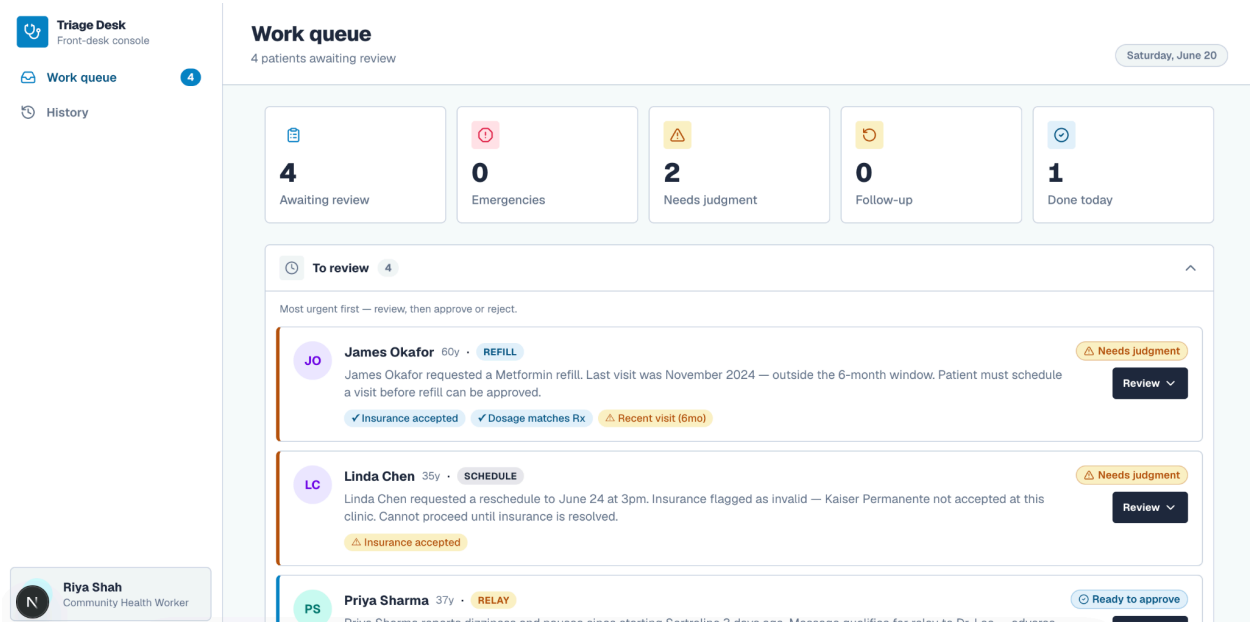
curl -sS -X POST \
  -F path_to_file \
  https://voice-to-text-b3d4b8ef0ae3.herokuapp.com/api/voicemail/intake | python3 -m json.tool

TEXT MESSAGE FOR PRESCRIPTION REFILLS:

- Hello, [Name]. Your prescription refill request for [dosage] [medicine name] has been approved and sent to [pharmacy].
- **DENIED:** Hello, [Name]. Your prescription refill request for [dosage] [medicine name] has been denied. You need an appointment before the medication is approved.

Button	Dashboard API route	Supabase write	FastAPI backend call
Approve (refill)	✅	✅	✅ /api/prescriptions
Approve (reschedule)	✅	✅	✅ /api/appointments
Approve (relay)	✅	✅	❌ *(not built yet — piece a)*
Reject	✅	✅	❌ *(denial SMS not built — piece b)*
Action taken / Mark done	✅	✅	❌


Most urgent first — review, then approve or reject.
JO
James Okafor 60y · REFILL
James Okafor requested a Metformin refill. Last visit was November 2024 — outside the 6-month window. Patient must schedule a visit before refill can be approved.
✓ Insurance accepted
✓ Dosage matches Rx
⚠ Recent visit (6mo)
⚠ Needs judgment
Hide
PROPOSED ACTION
Manual review required — last visit exceeds 6 month eligibility window
⚠ Eligibility gate failed. Use your judgment; a note is recommended for the record.
VOICEMAIL TRANSCRIPT
"Hey this is James Okafor, July 24 1965. I need a refill on my Metformin 500mg please. You can reach me at 510-555-0182."
510-555-0182
ELIGIBILITY CHECKS
✓ Insurance accepted
✓ Dosage matches Rx
⚠ Recent visit (6mo) — Last visit 2024-11-20
NOTE (RECOMMENDED)
Add context for the audit trail…
Approve
Reject


Triage Desk
Front-desk console
Work queue 4
History
Riya Shah
Community Health Worker
Work queue
4 patients awaiting review
Saturday, June 20
4
Awaiting review
0
Emergencies
2
Needs judgment
0
Follow-up
1
Done today
To review 4
Most urgent first — review, then approve or reject.
JO
James Okafor 60y · REFILL
James Okafor requested a Metformin refill. Last visit was November 2024 — outside the 6-month window. Patient must schedule a visit before refill can be approved.
✓ Insurance accepted
✓ Dosage matches Rx
⚠ Recent visit (6mo)
⚠ Needs judgment
Review
LC
Linda Chen 35y · SCHEDULE
Linda Chen requested a reschedule to June 24 at 3pm. Insurance flagged as invalid — Kaiser Permanente not accepted at this clinic. Cannot proceed until insurance is resolved.
⚠ Insurance accepted
⚠ Needs judgment
Review
PS
Priya Sharma 37y · RELAY
Ready to approve

/btw what is dashboard reading from the database prior to this edit and what would it be reading after this?

Here's the read picture, before and after the planned Option B refactor.

Before (current state)

The dashboard reads directly from Supabase itself, server-side, using the service-role key — all funneled through lib/tasks-repo.ts:

- getTasks() — reads the tasks table, then joins:
  - patients (for first_name, last_name, date_of_birth, phone)
  - voicemails (for transcript, audio_url)
  - …and maps each row into the UI Task shape.
  - Called by app/page.tsx (queue), app/history/page.tsx (history), and app/api/tasks/route.ts (GET).
- getTask(id) — single task with the same joins; called by the PATCH route.
- getPatientIdentity(patient_id) — reads patients for first/last/dob; called by lib/executor.ts.

So today the dashboard holds a direct DB connection (lib/supabase.ts) and queries tasks + patients + voicemails itself.

After (Option B)

The dashboard stops talking to Supabase entirely. Instead it reads through the **FastAPI backend** over HTTP:

- The pages/routes would call a backend endpoint like GET /api/tasks, which returns the already-joined, already-mapped Task[] (the tasks + patients + voicemails join + mapping logic moves into the backend).
- **The dashboard receives JSON tasks — it no longer queries tasks, patients, or voicemails directly.**
- **lib/supabase.ts and lib/tasks-repo.ts get deleted, and the service-role key is removed from the dashboard.**

In short: **before**, the dashboard reads tasks/patients/voicemails straight from Postgres via the service-role key; **after**, it reads the same data as pre-assembled Task objects from the FastAPI backend, and has no database access (or key) of its own.

One caveat from what's been established: the backend currently has no GET /api/tasks endpoint and the join/mapping logic lives only in the dashboard's tasks-repo.ts — so that read endpoint + mapping would need to be built on the backend as part of this change.

Here's the precise picture. Good news: it's very contained — the dashboard's DB access lives in **exactly two files**, and there's only **one write**.

Where the dashboard touches the DB right now

All of it is server-side, using the service-role key, funneled through lib/tasks-repo.ts (the only file with queries) + lib/supabase.ts (the only file with the connection):

Reads (getTasks / getTask / getPatientIdentity):
- app/page.tsx → getTasks() (queue)
- app/history/page.tsx → getTasks() (history)
- app/api/tasks/route.ts → getTasks() (GET)
- app/api/tasks/[id]/route.ts → getTask()
- lib/executor.ts → getPatientIdentity()

Writes — there is exactly one: patchTask() in lib/tasks-repo.ts:157 (db.from("tasks").update(...)), called only from app/api/tasks/[id]/route.ts for:
- **reject / action_taken / mark_done / reopen** → fully written by the dashboard (backend not involved at all)
- **approve** → *partly*: it calls the FastAPI executor (which inserts the prescription + flips status), but the dashboard *also* writes the audit fields (approved_by/reviewed_at/chw_note) itself at line 69.

So today: reject/handle/mark-done/reopen are 100% dashboard-written; approve is split between backend and dashboard.

What "backend owns all writes" requires

The target is: the dashboard never calls .update()/.insert(); the FastAPI service performs every mutation.

1. **Add one backend decision endpoint** — e.g. PATCH /api/tasks/{id}/decision taking {decision, note, chw} — that absorbs the logic currently split across app/api/tasks/[id]/route.ts (patchFor) and patchTask. It would handle **all** decisions: approve (→ existing executors), reject (+ denial SMS), action_taken, mark_done, reopen — with idempotency. *This is the same work as the three backend pieces (a/b/c) we discussed; this endpoint is their natural home.*
2. **Turn the dashboard's PATCH /api/tasks/[id] into a thin proxy** that forwards to that FastAPI endpoint, and **delete** patchTask. The browser/React code barely changes — it still calls /api/tasks/[id]; that route just forwards instead of writing.
3. **The backend returns the updated, joined task** (so the optimistic UI reconciles) — which means the row→Task mapping currently in tasks-repo.mapRow needs to exist backend-side, or the dashboard just refetches the list after a decision.
4. **Fold lib/executor.ts into the backend** — the dashboard would no longer route prescription-vs-appointment itself; it just says "approve" and the backend picks the executor.

One decision you need to make

"Front end never makes database **changes**" literally means **writes only** — so the simplest version moves just patchTask and leaves reads (and the service-role key) in the dashboard.

But the cleaner, more secure version also **moves the reads** (getTasks/getTask) behind a backend GET /api/tasks, which lets you **remove the service-role key from the dashboard entirely**. That key bypasses row-level security, so having it in a frontend app is the real smell — moving reads too is what fully fixes it.

Option	What moves	Service-role key in dashboard?	Effort
A — writes only	patchTask → backend	Still there (for reads)	Smaller
B — full (recommended)	writes **and** reads → backend	**Removed**	Larger, but "correct"

Message:

[dashboard] falling back to fixtures: Error: BACKEND_API_URL is not set
    at <unknown> (.next/server/chunks/ssr/[root-of-the-server]__0j1rw7u._.js:1:1018)
    at d (.next/server/chunks/ssr/[root-of-the-server]__0j1rw7u._.js:1:1063)
    at f (.next/server/chunks/ssr/[root-of-the-server]__0j1rw7u._.js:1:10349)
    at stringify (<anonymous>)

This confirms exactly what the `HANDOFF.md` flagged: since the "Option B" cutover, the dashboard has no Supabase fallback anymore — it needs `BACKEND_API_URL` pointing at a deployed `backend/api`, and that backend isn't deployed yet.

The fix has two parts, both required:

1. Deploy `backend/api` somewhere with a public URL (Railway, Render, or Fly — it already has a `Procfile`: `uvicorn main:app --host 0.0.0.0 --port $PORT`). Set these env vars on whichever host you pick:

- `SUPABASE_URL`
- `SUPABASE_SERVICE_ROLE_KEY`
- `ANTHROPIC_API_KEY`
- `DEEPGRAM_API_KEY`
- `TEXTBELT_API_KEY`

2. Set `BACKEND_API_URL` on the Vercel `dashboard` project to that new public URL, then redeploy.

Without step 1, step 2 has nothing to point at — the error you're seeing is just the dashboard confirming the env var is literally unset on Vercel right now, which is expected since `backend/api` doesn't have a host yet.

I can't do either step myself — I don't have Railway/Render/Fly or Vercel credentials in this environment. Want me to walk you through the Railway (or Render) deploy step by step as you click through it, or do you already have a host in mind and just need the env var list confirmed?

BACKEND_API_URL=https://voice-to-text-b3d4b8ef0ae3.herokuapp.com

Pitch

Let me show you what happens to a single voicemail. A patient calls asking for a refill. Deepgram transcribes the call. Our intake agent then extracts key details—her name, date of birth, medication, and insurance information—as structured data with confidence scores. From there, our eligibility engine checks the extracted information against the patient database to identify policy violations, unusual patterns, or anything that required followup.

Here is the important part. Nothing executes without a human approval. When a task is flagged, it enters the review queue—the CHW dashboard. The reviewer sees the full picture: what the agent checked, what it proposes to do, and why it's asking. They approve or reject.

Reject closes it out, logged. Approve sends it to the execution engine, which runs the action

against external services pharmacy SMS, calendar write, EHR update—with no further human steps. And everything, both paths, lands in an audit trail.