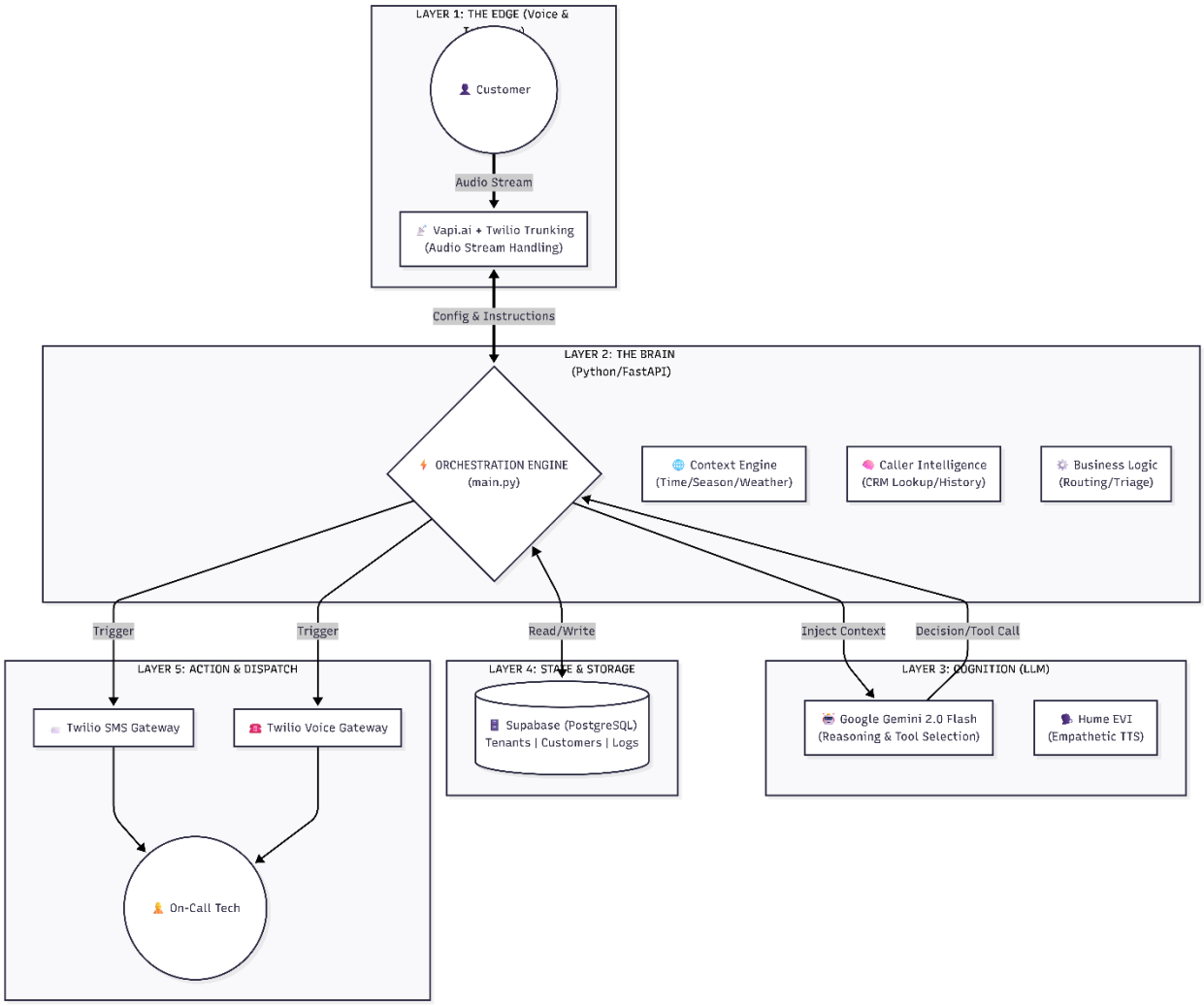


AI Voice Dispatch System – Technical Architecture

1. Executive Summary

This system is an **Event-Driven AI Voice Orchestrator** designed to replace legacy answering services. It utilizes **Vapi.ai** for low-latency voice interaction, **FastAPI** for logic orchestration, and **Supabase** for persistence.

The architecture decouples **Intelligence** (Prompt Engineering, Context Gathering) from **Action** (Dispatching, Logging). This ensures that the AI possesses situational awareness (weather, caller history) before the conversation begins, and executes deterministic logic (SMS dispatch, DB logging) when specific intents are detected.

High-Level Stack

Component	Technology	Role
Voice Gateway	Vapi.ai	Telephony, ASR (Speech-to-Text), TTS (Text-to-Speech), LLM Stream management.
Orchestrator	Python / FastAPI	The "Brain." Handles routing, context injection, and prompt generation.
Database	Supabase (PostgreSQL)	Stores tenant configs, caller history (ANI lookup), and call logs.
External APIs	NWS Weather / Twilio	Provides environmental context (Weather) and output channels (SMS/Voice).
Infrastructure	Railway / Docker	Hosting environment for the FastAPI service.

Key Call-Outs:

1. **Multi-Tenant Isolation**

   The architecture is natively multi-tenant.
   - **Routing:** Inbound calls are routed via DNIS (Dialed Number Identification Service).
   - **Data Security:** All database queries are scoped by tenant_id. A plumber in Knoxville will never see the leads of an HVAC tech in Nashville.
   - **Configurability:** Each tenant has unique business rules (e.g., "Emergency" means <50°F for Tenant A, but <60°F for Tenant B).

2. **Dynamic Context Injection (The "Zero-Shot" Strategy)**

   Most AI systems hallucinate because they lack ground truth. We solve this by injecting a "State of the World" payload into the system prompt *before* the AI speaks a single word.

   The OL compiles a JSON payload containing:

   1. **Temporal Grounding:** Server-side calculation of Time, Day, and Season. (Prevents the AI from thinking it's 2 PM when it's 2 AM).
   2. **Environmental Grounding:** Real-time query to the National Weather Service API. If there is a "Freeze Warning," the AI logic creates a bias toward "Emergency" for heating calls.

3. **CRM Grounding:** Real-time lookup of the caller's ANI (Phone Number) in Supabase. We inject their Name, Address, and Equipment History.

Strategic Value: This reduces token usage (cost) and latency while increasing trust. The AI sounds like a tenured employee, not a call center script.

Comprehensive Feature Catalog

Core Telephony & Interaction

- **24/7 Answering:** Instant pickup with zero hold times.
- **Human-Parity Voice:** Utilizes highly trained voice. Does not sound robotic.
- **Interruption Handling:** The AI handles "barge-in" (users interrupting) gracefully.
- **Noise Robustness:** Filters out background noise to focus on intent.

Intelligence & Triage

- **Semantic Triage:** Distinguishes between "Urgent" (Water leak) and "Routine" (Maintenance quote) using semantic meaning, not just keywords.
- **Safety Guardrails:** Hard-coded detection of "Gas," "Fire," or "CO Alarm" triggers an immediate fail-safe script directing the user to 911.
- **Commercial Detection:** Automatically flags calls from businesses (Restaurants, Hotels) as High Priority.
- **Vulnerable Person Detection:** Listens for context clues ("baby," "elderly," "oxygen tank") to escalate priority.

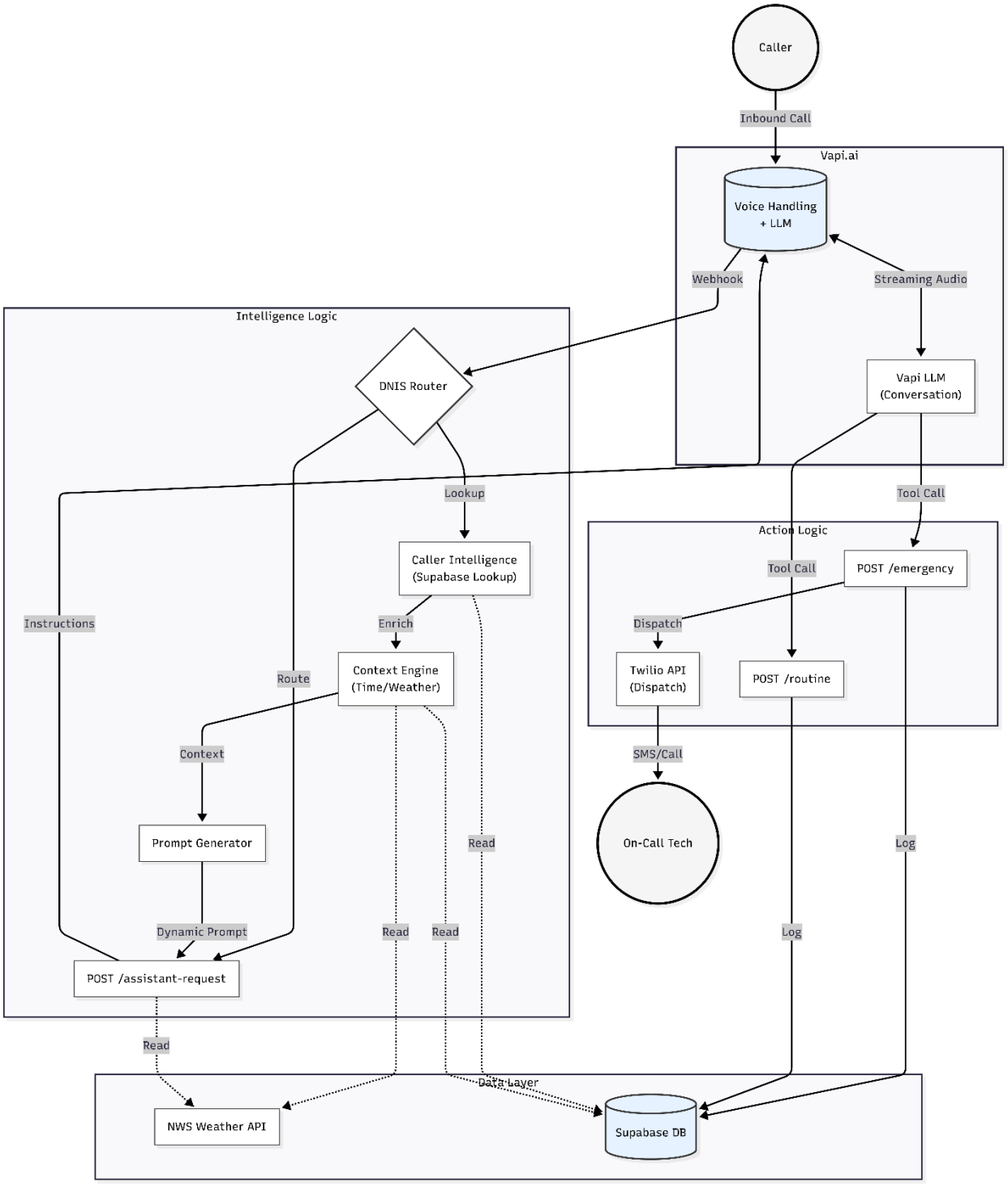
Dispatch & Action

- **Dual-Channel Dispatch:**
    1. **SMS:** Instant text payload with Name, Address, Issue Summary, and Classification.
    2. **Voice Page:** A robotic call to the technician to ensure they wake up. Escalation available, when page is not acknowledged.
- **CRM Sync:** Automatically creates or updates customer records in Supabase based on the conversation.
- **Routine Logging:** Non-emergencies are logged to a dashboard for next-day follow-up (protecting the owner's sleep).

Admin & Observability

- **Live Dashboard:** React-based frontend for owners to view call logs and recordings.
- **"Time Travel" Testing:** A developer tool allowing us to simulate different times of day and weather conditions to test logic without waiting for a blizzard.
- **Cost Analytics:** Real-time tracking of spend per call (LLM + Voice + Telephony costs).

Architecture Diagram



Core Components & Logic Flow

The architecture is divided into three distinct phases: **Initialization (The Brain)**, **Conversation (The Voice)**, and **Execution (The Actions)**.

Phase 1: Initialization & Context (POST /assistant-request)

This is the "Cold Start" of the call. Before the AI speaks, the Orchestrator runs a logic chain to build a dynamic persona.

1. **DNIS Routing:** The system identifies the customer.number (Inbound DNIS) to determine which Tenant is being called (e.g., "HVAC Corp" vs. "Dr. Smith").
2. **Caller Intelligence:** The system queries the Customers table in Supabase using the caller.phone (ANI).
    - *Hit:* Returns name and address (e.g., "Oh, hi Bob").
    - *Miss:* Returns generic context.
3. **Environmental Context:** The system queries the National Weather Service (NWS) API based on the Tenant's zip code to inject weather alerts (e.g., "Severe Freeze Warning").
4. **Prompt Assembly:** A Jinja2 or f-string template combines:
    - Tenant System Prompt
    - Caller Data
    - Weather Alerts
    - Time of Day
5. **Return:** The endpoint returns a generic Vapi Assistant Configuration JSON containing the newly baked system message.

Phase 2: The Conversation (Vapi LLM)

Vapi maintains the WebSocket connection with the caller. The LLM (e.g., GPT-4o or Haiku) streams audio based on the prompt generated in Phase 1. The Orchestrator is idle during this phase until the user triggers an intent.

Phase 3: Action Execution (Tool Calls)

When the LLM detects a specific intent, it pauses the conversation to hit specific API endpoints defined in main.py.

A. Emergency Dispatch (POST /emergency)

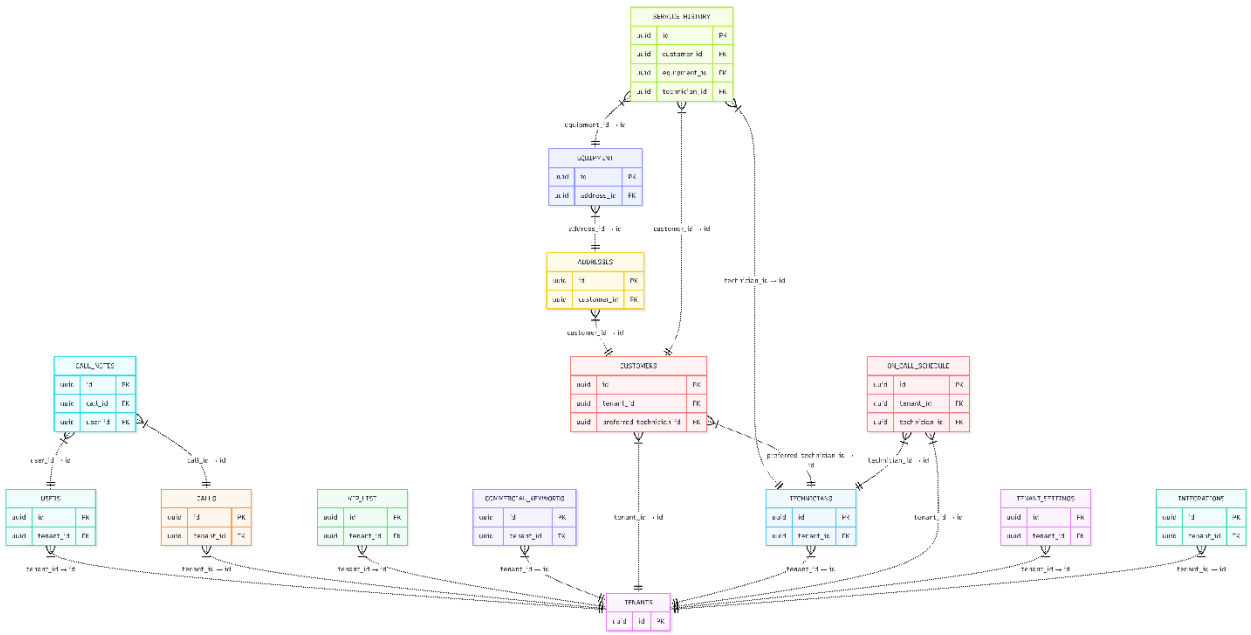
- **Trigger:** Caller says "I have no heat" or "Water is leaking."
- **Payload:** { "summary": "No heat", "address": "123 Main", "callback": "555-0199" }
- **Logic:**
    1. Log incident to Call_Logs table in Supabase (Status: URGENT).
    2. Format an SMS payload.
    3. Call Twilio API to dispatch SMS to the On-Call Technician.

4. Return success message to Vapi (e.g., "Technician notified").

B. Routine Message (POST /routine)

1. **Trigger:** Caller says "I just need to schedule a checkup."
2. **Payload:** { "message": "Requesting tune-up", "availability": "Tuesday mornings" }
3. **Logic:**
    1. Log entry to Call_Logs table (Status: ROUTINE).
    2. (Optional) Send email digest or slack notification.
    3. Return success message to Vapi.

Database Schema (Supabase)



The application relies on a relational schema to handle multi-tenancy.

tenants

Stores configuration for the companies using the system.

- id (UUID, PK)
- dnis_number (String, Unique) - The phone number assigned to this tenant.

- business_name (String)
- base_prompt (Text) - The core personality instructions.
- zip_code (String) - Used for weather lookups.
- tech_phone (String) - Destination for emergency SMS.

customers

Known callers for ANI lookup.

- phone_number (String, PK)
- full_name (String)
- address (String)
- tenant_id (FK)

call_logs

Audit trail of all actions.

- call_id (UUID, PK)
- timestamp (ISO8601)
- caller_number (String)
- transcript_summary (Text)
- classification (Enum: EMERGENCY, ROUTINE)
- dispatch_status (Boolean)

API Specifications (main.py)

Endpoint: POST /assistant-request

Purpose: Webhook target for Vapi "Server URL".
Request:

```
codeJSON
{
  "message": {
    "call": {
      "customer": { "number": "+15550001111" },
      "phone_call_provider_id": "vapi-call-uuid"
    }
  }
```

```
}
```
Response:
```
codeJSON
{
  "assistant": {
    "name": "Dynamic Dispatcher",
    "model": {
      "provider": "openai",
      "model": "gpt-4",
      "systemMessage": "You are a dispatcher for HVAC Corp. It is currently
freezing (-5F)..."
    }
  }
}
```
Endpoint: POST /emergency
Purpose: Tool definition for high-priority handling.
Request:
```
codeJSON
{
  "tool_call_id": "call_12345",
  "parameters": {
    "issue_summary": "Boiler failure",
    "customer_name": "John Doe"
  }
}
```

Environment Variables

The application requires the following configuration to run:

```
codeBash
# Database
SUPABASE_URL="https://xyz.supabase.co"
SUPABASE_KEY="ey..."

# Voice & AI
VAPI_PRIVATE_KEY="vapi-..."
OPENAI_API_KEY="sk-..."

# Dispatch
TWILIO_ACCOUNT_SID="AC..."
TWILIO_AUTH_TOKEN="cx..."
TWILIO_FROM_NUMBER="+1555..."

# External Data
NWS_USER_AGENT="(myweatherapp.com, contact@email.com)"
```

Security & Risk Mitigation

- **Prompt Injection Defense:** The System Prompt (night_watch_prompt_v4.txt) is structured with strict "Role Locking" to prevent the AI from being tricked into unauthorized actions.
- **Fail-Safe Mode:** If the Database or Logic Layer fails, the system falls back to a "Safe Mode" static response, ensuring the caller is never met with dead air.
- **PII Handling:** Customer data is encrypted at rest in Supabase and in transit via TLS 1.3.

APPENDIX:

Mermaid code for Arch Diagram

```
---
config:
  layout: dagre
---
flowchart TB
 subgraph S_VAPI["Vapi.ai"]
        VoiceCore[("Voice Handling<br>+ LLM")]
        VapiLLM["Vapi LLM<br>(Conversation)"]
  end
 subgraph S_BRAIN["Intelligence Logic"]
    direction TB
        DNIS{"DNIS Router"}
        CallerIntel["Caller Intelligence<br>(Supabase Lookup)"]
        ContextEng["Context Engine<br>(Time/Weather)"]
        PromptGen["Prompt Generator"]
        EP_Assist["POST /assistant-request"]
  end
 subgraph S_ACTION["Action Logic"]
    direction TB
        EP_Emerg["POST /emergency"]
        EP_Rout["POST /routine"]
        Twilio["Twilio API<br>(Dispatch)"]
  end
 subgraph S_DATA["Data Layer"]
        NWS["NWS Weather API"]
        Supabase[("Supabase DB")]
  end
    Caller(("Caller")) -- Inbound Call --> VoiceCore
    VoiceCore <-- Streaming Audio --> VapiLLM
    VoiceCore -- Webhook --> DNIS
    DNIS -- Route --> EP_Assist
    DNIS -- Lookup --> CallerIntel
    CallerIntel -- Enrich --> ContextEng
    ContextEng -- Context --> PromptGen
    PromptGen -- Dynamic Prompt --> EP_Assist
    EP_Assist -- Instructions --> VoiceCore
    VapiLLM -- Tool Call --> EP_Emerg & EP_Rout
    EP_Emerg -- Dispatch --> Twilio
```

```
    EP_Emerg -- Log --> Supabase
    EP_Rout -- Log --> Supabase
    CallerIntel -. Read .-> Supabase
    ContextEng -. Read .-> Supabase & NWS
    EP_Assist -. Read .-> NWS
    Twilio -- SMS/Call --> Tech(("On-Call Tech"))
    EP_Emerg ~~~ VoiceCore
    VoiceCore ~~~ EP_Assist

     Caller:::term
     VoiceCore:::cylinder
     VapiLLM:::default
     DNIS:::diamond
     Supabase:::cylinder
     Tech:::term
  %% --- PRINTER FRIENDLY STYLING ---
    %% Standard boxes (White background, dark grey text)
    classDef default fill:#ffffff,stroke:#333333,stroke-width:1px,color:#000000;

    %% Containers/Subgraphs (White background, thick black border)
    classDef dark fill:#ffffff,stroke:#000000,stroke-width:2px,color:#000000;

    %% Databases (Very pale blue fill to distinguish them)
    classDef cylinder fill:#e6f2ff,stroke:#333333,stroke-
width:2px,color:#000000,shape:cylinder;

    %% Decision Diamonds (White background)
    classDef diamond fill:#ffffff,stroke:#333333,stroke-width:2px,color:#000000,shape:diamond;

    %% Start/End Circles (Light grey fill, thick border)
    classDef term fill:#f5f5f5,stroke:#000000,stroke-width:3px,color:#000000,shape:circle;
```

Mermaid code for Database Schema Diagram

```
erDiagram
  TENANTS {
    uuid id PK
  }

  USERS {
    uuid id PK
    uuid tenant_id FK
  }

  CALLS {
    uuid id PK
    uuid tenant_id FK
  }

  CALL_NOTES {
    uuid id PK
    uuid call_id FK
    uuid user_id FK
  }

  VIP_LIST {
    uuid id PK
    uuid tenant_id FK
  }
```

```
COMMERCIAL_KEYWORDS {
  uuid id PK
  uuid tenant_id FK
}

CUSTOMERS {
  uuid id PK
  uuid tenant_id FK
  uuid preferred_technician_id FK
}

ADDRESSES {
  uuid id PK
  uuid customer_id FK
}

EQUIPMENT {
  uuid id PK
  uuid address_id FK
}

SERVICE_HISTORY {
  uuid id PK
  uuid customer_id FK
  uuid equipment_id FK
  uuid technician_id FK
}

TECHNICIANS {
  uuid id PK
  uuid tenant_id FK
}

ON_CALL_SCHEDULE {
  uuid id PK
  uuid tenant_id FK
  uuid technician_id FK
}

TENANT_SETTINGS {
  uuid id PK
  uuid tenant_id FK
}

INTEGRATIONS {
  uuid id PK
  uuid tenant_id FK
}

USERS }|..|| TENANTS : "tenant_id → id"
CALLS }|..|| TENANTS : "tenant_id → id"
VIP_LIST }|..|| TENANTS : "tenant_id → id"
COMMERCIAL_KEYWORDS }|..|| TENANTS : "tenant_id → id"
CUSTOMERS }|..|| TENANTS : "tenant_id → id"
TECHNICIANS }|..|| TENANTS : "tenant_id → id"
ON_CALL_SCHEDULE }|..|| TENANTS : "tenant_id → id"
TENANT_SETTINGS }|..|| TENANTS : "tenant_id → id"
INTEGRATIONS }|..|| TENANTS : "tenant_id → id"

CALL_NOTES }|..|| CALLS : "call_id → id"
```

```
  CALL_NOTES }|..|| USERS : "user_id → id"

  ADDRESSES }|..|| CUSTOMERS : "customer_id → id"
  EQUIPMENT }|..|| ADDRESSES : "address_id → id"

  SERVICE_HISTORY }|..|| CUSTOMERS : "customer_id → id"
  SERVICE_HISTORY }|..|| EQUIPMENT : "equipment_id → id"
  SERVICE_HISTORY }|..|| TECHNICIANS : "technician_id → id"

  CUSTOMERS }|..|| TECHNICIANS : "preferred_technician_id → id"
  ON_CALL_SCHEDULE }|..|| TECHNICIANS : "technician_id → id"
```

Mermaid code for Database Schema Diagram