# Speech-to-Text (STT) Architecture & Implementation (Whisper AI & Sarvam AI)

**Scope:** This guide documents how the voice pipeline works end‑to‑end in the Revival365 stack, covering data flow, dependencies, environment setup, API contracts, error handling, and how to create/configure a new Sarvam STT API key and wire it into the app via .env.

## 1) High‑Level Data Flow

[Browser/Client]  
 └─▶ (1) POST /api/chat/voice  
 body: { audioBase64, language\_mode, session\_id, patient\_id, ... }  
  
[API Layer]  
 ├─▶ (2) Decode base64 → bytes, detect mime, normalize → 16kHz mono WAV  
 ├─▶ (3a) If language\_mode = "International": Whisper STT  
 │ - local/CLI: whisper <wav> --model <...> → transcript  
 │ - or service invocation depending on your setup  
 ├─▶ (3b) If language\_mode = "Regional": Sarvam STT  
 │ - HTTP multipart to SARVAM\_STT\_URL with SARVAM\_API\_KEY  
 │ - returns transcript (optionally translated → English)  
 ├─▶ (4) Build chat context (RBAC + patient/doctor mapping)  
 ├─▶ (5) Route message → agent executor (diet routing, protocol-only if needed)  
 └─▶ (6) Return JSON { transcript, final\_response, metadata }

## 2) API Contract: /api/chat/voice

**Method:** POST

**Content-Type:** application/json

**Request Body (accepted aliases):** - audioBase64 *(alias: audio\_base64)* — **required**, base64 Data URL or raw base64 audio buffer (e.g., data:audio/webm;base64,<...> or just <...>) - languageMode *(alias: language\_mode)* — "International" (Whisper), "Regional" (Sarvam) - sessionId *(alias: session\_id)* — optional - patientId *(alias: patient\_id)* — optional; if provided, used for RBAC/route conditioning - doctorId *(alias: doctor\_id)* — optional; used if the role is doctor - token — optional bearer for role/context (front‑end typically sets this)

**Response Body:**

{  
 "transcript": "patient said ...",  
 "response": "agent reply text ...",  
 "language\_mode": "International" | "Regional",  
 "stt\_engine": "whisper" | "sarvam",  
 "session\_id": "...",  
 "patient\_id": "...",  
 "meta": {  
 "audio\_mime": "audio/webm",  
 "duration\_ms": 5321,  
 "pipeline\_ms": 842,  
 "whisper": { "model": "base|small|medium|large" },  
 "sarvam": { "endpoint": ".../speech-to-text-translate", "lang": "auto→en" }  
 }  
}

**HTTP Status Codes:** - 200 OK — success - 400 Bad Request — missing/invalid audio, unsupported mime - 401/403 — auth/role issues - 415 Unsupported Media Type — cannot parse - 429 — rate‑limit from upstream (e.g., Sarvam) - 5xx — server or upstream failure

## 3) Audio Normalization

To provide consistent STT quality and avoid upstream rejections, the server normalizes inputs to **16 kHz mono WAV**:

1. **Base64 decode** → bytes
2. **MIME detection** (audio/webm, audio/ogg, audio/mpeg, audio/wav, etc.)
3. **Transcode to WAV** using **ffmpeg**:
   * Sample rate: 16000 Hz
   * Channels: mono (1)
   * PCM 16‑bit LE

**CLI equivalence:**

ffmpeg -y -i input.any -ac 1 -ar 16000 -f wav output.wav

## 4) Whisper AI Path (“International”)

### 4.1 Overview

* Designed for English (or English‑dominant) voice input.
* Runs locally via the Whisper CLI or a service wrapper depending on environment.
* Low latency for short queries; model size choice balances speed/accuracy.

### 4.2 Invocation (reference flow)

1. Save bytes → tmp file (e.g., tmp/input.wav).
2. Ensure WAV normalization (see §3).
3. Call Whisper:

* whisper tmp/input.wav --model small --language en --task transcribe --fp16 False

1. Parse stdout / generated .txt to obtain transcript.

**Note:** If you use openai-whisper Python package, you can also load the model in‑process:

import whisper  
model = whisper.load\_model("small")  
result = model.transcribe("tmp/input.wav", language="en")  
transcript = result["text"].strip()

### 4.3 Config knobs

* **Model size:** tiny, base, small, medium, large
* **Language:** en (fixed) or auto‑detect
* **FP16:** disable on CPU (--fp16 False)
* **Device:** GPU/CPU selection (environment specific)

### 4.4 Error handling

* **Missing ffmpeg:** return 500 with actionable hint
* **Long audio:** either reject with 400 or chunk
* **Timeouts:** kill the subprocess and return fallback error
* **Empty transcript:** respond with 400 (“Audio too short or silent”)

## 5) Sarvam AI Path (“Regional”)

### 5.1 Overview

* Optimized for Indian regional languages.
* Server posts audio to **Sarvam STT API**; receives transcript (optionally **translated to English** for downstream LLM consistency).

### 5.2 HTTP Call (reference flow)

* **Method:** POST
* **URL:** ${SARVAM\_STT\_URL} (e.g., https://api.sarvam.ai/speech-to-text-translate)
* **Headers:** Authorization: Bearer ${SARVAM\_API\_KEY}
* **Body:** multipart/form-data with file + params

**Pseudo‑code:**

import requests  
  
files = {  
 "file": ("input.wav", open("tmp/input.wav", "rb"), "audio/wav")  
}  
# params vary by plan; this is a common pattern  
payload = {  
 "target\_language": "en", # translate to English for LLM  
 "source\_language": "auto" # let Sarvam detect  
}  
headers = {"Authorization": f"Bearer {SARVAM\_API\_KEY}"}  
resp = requests.post(SARVAM\_STT\_URL, files=files, data=payload, headers=headers, timeout=60)  
resp.raise\_for\_status()  
transcript = resp.json().get("text", "").strip()

### 5.3 Error handling

* **401/403:** invalid/expired API key → instruct to rotate in .env
* **413:** payload too large → pre‑compress or chunk
* **415:** rejected mime → ensure WAV as in §3
* **429:** rate limit → exponential backoff, jitter, show friendly retry message
* **5xx:** upstream issue → degrade gracefully, suggest retry

## 6) Environment Variables & Configuration

Add the following to your **.env**:

# Whisper (local)  
WHISPER\_MODEL=small  
WHISPER\_LANGUAGE=en  
WHISPER\_USE\_CLI=true # if true, invokes CLI; false uses Python API  
  
# Sarvam  
SARVAM\_API\_KEY=sk\_sarvam\_xxx # generated from Sarvam console  
SARVAM\_STT\_URL=https://api.sarvam.ai/speech-to-text-translate  
SARVAM\_SOURCE\_LANG=auto  
SARVAM\_TARGET\_LANG=en # translate to English for LLM  
  
# Voice endpoint behavior  
VOICE\_MAX\_DURATION\_MS=120000 # 2 minutes cap  
VOICE\_ACCEPTED\_MIMES=audio/webm,audio/ogg,audio/mpeg,audio/wav

**Tip:** In code, map camelCase aliases (e.g., audioBase64) to snake\_case via Pydantic alias to keep the front‑end flexible.

## 7) Dependencies

### 7.1 System

* **ffmpeg** (required for audio normalization/transcode)
* **Python 3.10+**

**Install ffmpeg:** - Ubuntu/Debian: sudo apt-get update && sudo apt-get install -y ffmpeg - macOS (Homebrew): brew install ffmpeg - Windows (choco): choco install ffmpeg

### 7.2 Python Packages

Add to your backend environment (e.g., requirements.txt):

# Core web  
fastapi  
uvicorn  
pydantic  
python-multipart  
  
# Audio/STT  
openai-whisper # or faster-whisper for performance  
numpy  
soundfile  
pydub  
requests  
  
# Utilities  
python-dotenv  
loguru

If you choose **faster-whisper**, also install ctranslate2 and set device flags; adjust code accordingly.

## 8) Creating a New Sarvam API Key & Wiring .env

1. **Create/Sign in** to your Sarvam developer account (console).
2. **Navigate** to *API Keys* (or *Developers* → *Credentials*).
3. **Generate** a new key (scope: STT/Translate). Copy the key value.
4. **Record** the base STT endpoint URL from Sarvam docs (e.g., /speech-to-text-translate).
5. **Update .env:**

* SARVAM\_API\_KEY=sk\_sarvam\_live\_xxx  
  SARVAM\_STT\_URL=https://api.sarvam.ai/speech-to-text-translate  
  SARVAM\_SOURCE\_LANG=auto  
  SARVAM\_TARGET\_LANG=en

1. **Reload** the backend (so env is re‑read). For Docker, rebuild or restart the service with the updated env.
2. **Rotate** keys periodically and on suspected leakage; keep .env out of VCS or use secrets manager.

## 9) Integration Notes (Backend)

* **Pydantic model** accepts both camelCase and snake\_case via alias.
* **MIME detection**: infer from data URL prefix; fallback by magic bytes.
* **WAV enforcement**: centralize in \_ensure\_wav(...) helper; return path + derived mime.
* **Routing**: choose STT engine based on language\_mode.
* **RBAC context**: propagate patient\_id/doctor\_id/token into the agent layer before invoking.
* **Diet questions**: if text matches food intent (e.g., “can I/should I eat …”), route to protocol‑only agent.

## 10) Security & Privacy

* **Do not log** raw audio or transcripts in production logs.
* **Mask** API keys in logs; store in a secret manager for prod.
* **Purge** temp audio files after processing (use NamedTemporaryFile(delete=True) or async cleanup).
* **TLS** everywhere; Sarvam calls over https only.
* **Rate limit** the voice endpoint (e.g., per IP/session) to control cost & abuse.

## 11) Troubleshooting

| Symptom | Likely Cause | Fix |
| --- | --- | --- |
| 415 Unsupported Media Type | Raw bytes not recognized | Ensure data URL prefix or specify MIME; enforce WAV (§3) |
| Empty transcript | Silent/low volume | Encourage user to speak closer; normalize gain; reject <1s audio |
| ffmpeg: command not found | Missing system dep | Install ffmpeg; verify in PATH |
| 401 from Sarvam | Bad/expired key | Rotate key; check .env + deployment secrets |
| 429 from Sarvam | Rate limit | Add retry with backoff; surface friendly message |
| High latency (Whisper) | Large model on CPU | Switch to base/small; try faster-whisper with GPU |
| Unicode gibberish | Wrong language hints | Set SARVAM\_SOURCE\_LANG=auto, target en; or pass correct source |

## 12) Minimal Reference Implementations

### 12.1 Whisper (Python in‑process)

import base64, tempfile, whisper  
from pathlib import Path  
  
def transcribe\_whisper(audio\_b64: str, model\_name: str = "small") -> str:  
 hdr, b64 = (audio\_b64.split(',', 1) + [None])[:2]  
 raw = base64.b64decode(b64 or audio\_b64)  
 with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as f:  
 f.write(raw)  
 wav\_path = Path(f.name)  
 model = whisper.load\_model(model\_name)  
 result = model.transcribe(str(wav\_path), language="en")  
 wav\_path.unlink(missing\_ok=True)  
 return result.get("text", "").strip()

### 12.2 Sarvam (requests)

import base64, tempfile, requests  
from pathlib import Path  
  
SARVAM\_STT\_URL = "https://api.sarvam.ai/speech-to-text-translate"  
SARVAM\_API\_KEY = "<set via env>"  
  
def transcribe\_sarvam(audio\_b64: str, src="auto", tgt="en") -> str:  
 hdr, b64 = (audio\_b64.split(',', 1) + [None])[:2]  
 raw = base64.b64decode(b64 or audio\_b64)  
 with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as f:  
 f.write(raw)  
 wav\_path = Path(f.name)  
 headers = {"Authorization": f"Bearer {SARVAM\_API\_KEY}"}  
 files = {"file": (wav\_path.name, open(wav\_path, "rb"), "audio/wav")}  
 data = {"source\_language": src, "target\_language": tgt}  
 r = requests.post(SARVAM\_STT\_URL, headers=headers, files=files, data=data, timeout=60)  
 wav\_path.unlink(missing\_ok=True)  
 r.raise\_for\_status()  
 return r.json().get("text", "").strip()

## 13) Testing Checklist

* ✅ Base64 input with and without data: prefix
* ✅ MIME: webm/ogg/mp3/wav → normalized to wav
* ✅ Short (<1s) and long (>2m) inputs
* ✅ Whisper path (International)
* ✅ Sarvam path (Regional) with translation → EN
* ✅ RBAC context present (patient\_id/doctor\_id)
* ✅ Protocol‑only routing fired on “can I/should I eat …”
* ✅ Key rotation without server restart (if using secrets store)

**End of guide.**