Here's a clear and structured documentation outline for your **Speech-to-Text Lab using Azure AI**

🧪 Speech-to-Text Lab Documentation

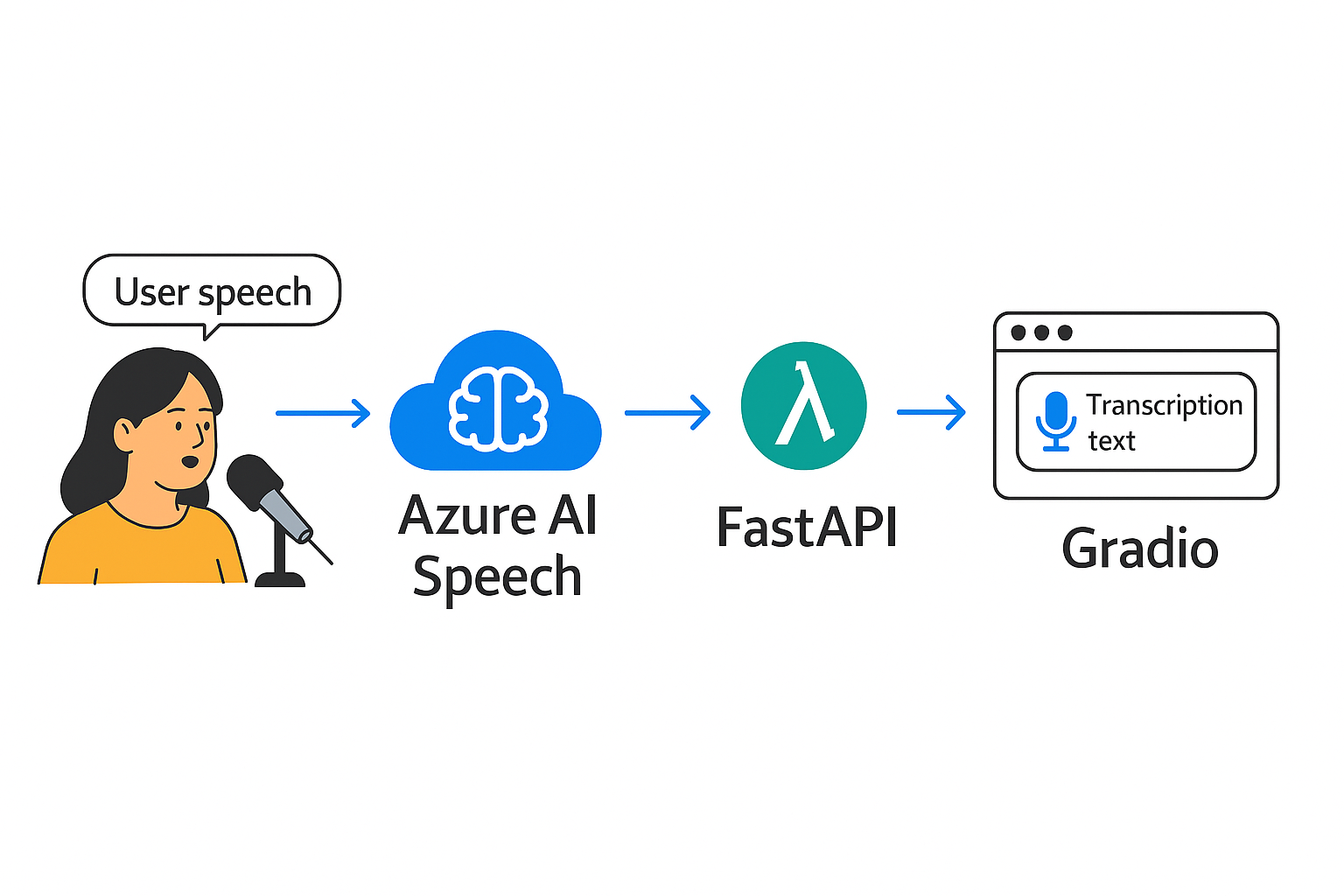
**Project Title:** *Voice to Text Pipeline using Azure AI Speech, FastAPI, and Gradio*

📌 Objective

Build a modular, real-time speech-to-text application using Azure AI Speech SDK, FastAPI for backend orchestration, and Gradio for frontend interaction.

🧠 Architecture Overview

**Visual:**



It shows:

* **User Input → Azure AI Speech → FastAPI → Gradio UI**
* Clear arrows and icons for each component
* Transcription flow from microphone to text

🛠️ Tech Stack

| **Component** | **Technology Used** | **Purpose** |
| --- | --- | --- |
| Speech SDK | Azure AI Speech | Real-time transcription |
| Backend API | FastAPI | Endpoint orchestration (/transcribe) |
| Frontend UI | Gradio | Interactive web interface |

⚙️ Setup Instructions

1. **Install Dependencies**

pip install azure-cognitiveservices-speech fastapi gradio uvicorn

2. **Configure Azure Speech SDK**

import azure.cognitiveservices.speech as speechsdk speech\_config = speechsdk.SpeechConfig(subscription="YOUR\_KEY", region="YOUR\_REGION") audio\_config = speechsdk.AudioConfig(use\_default\_microphone=True) speech\_recognizer = speechsdk.SpeechRecognizer(speech\_config=speech\_config, audio\_config=audio\_config)

3. **Create FastAPI Endpoint**

from fastapi import FastAPI app = FastAPI() @app.get("/transcribe") def transcribe(): result = speech\_recognizer.recognize\_once() return {"text": result.text}

4. **Build Gradio UI**

import gradio as gr def transcribe\_audio(): result = speech\_recognizer.recognize\_once() return result.text gr.Interface(fn=transcribe\_audio, inputs="microphone", outputs="text").launch()

🧪 Testing Flow

1. Launch Gradio UI
2. Speak into the microphone
3. Transcription appears in real-time
4. FastAPI logs the request and response

📈 Use Cases

* Voice-enabled agents
* Real-time transcription tools
* Accessibility features
* AI-powered voice interfaces

**Step 1: Console speech-to-text with Azure AI Speech**

Let’s start simple: a console app that listens to your microphone and prints the transcription. Run this in VS Code on Windows 11. Once you confirm it works, say “Next” and we’ll move to FastAPI.

**Folder structure**

azure-speech-stt/

azure-speech-stt/

├─ .env

├─ requirements.txt

└─ src/

└─ console\_stt.py

**Setup**

* **Prerequisites:** An Azure Speech resource with a Key and Region.
* **Create venv (Windows):**
  + PowerShell:

python -m venv .venv

.\.venv\Scripts\Activate

* **Dependencies (requirements.txt):**

azure-cognitiveservices-speech==1.38.0

python-dotenv==1.0.1

* **Environment file (.env):**

SPEECH\_KEY=your\_speech\_key\_here

SPEECH\_REGION=your\_region\_here # e.g., eastus, westus2

SPEECH\_LANGUAGE=en-US # optional; set your localepython-dotenv==1.0.1

**Code (src/console\_stt.py)**

import os

import sys

import time

from dotenv import load\_dotenv

try:

import azure.cognitiveservices.speech as speechsdk

except ImportError:

print("azure-cognitiveservices-speech is not installed. Run: pip install -r requirements.txt")

sys.exit(1)

load\_dotenv()

SPEECH\_KEY = os.getenv("SPEECH\_KEY")

SPEECH\_REGION = os.getenv("SPEECH\_REGION")

SPEECH\_LANGUAGE = os.getenv("SPEECH\_LANGUAGE", "en-US")

if not SPEECH\_KEY or not SPEECH\_REGION:

print("Missing SPEECH\_KEY or SPEECH\_REGION in environment. Set them in your .env file.")

sys.exit(1)

def make\_recognizer():

speech\_config = speechsdk.SpeechConfig(subscription=SPEECH\_KEY, region=SPEECH\_REGION)

speech\_config.speech\_recognition\_language = SPEECH\_LANGUAGE

audio\_config = speechsdk.audio.AudioConfig(use\_default\_microphone=True)

return speechsdk.SpeechRecognizer(speech\_config=speech\_config, audio\_config=audio\_config)

def single\_utterance():

print("\nSingle-utterance mode: Speak once after the beep. Recognition stops when you pause.\n")

recognizer = make\_recognizer()

result = recognizer.recognize\_once\_async().get()

if result.reason == speechsdk.ResultReason.RecognizedSpeech:

print(f"Transcription: {result.text}")

elif result.reason == speechsdk.ResultReason.NoMatch:

print("No speech could be recognized.")

elif result.reason == speechsdk.ResultReason.Canceled:

cancellation = result.cancellation\_details

print(f"Canceled: {cancellation.reason}")

if cancellation.reason == speechsdk.CancellationReason.Error:

print(f"Error details: {cancellation.error\_details}")

print("Check your key/region, network, and microphone permissions.")

def continuous\_until\_enter():

print("\nContinuous mode: Start speaking. Press Enter to stop.\n")

recognizer = make\_recognizer()

def recognizing(evt):

# Partial (interim) results

if evt.result.text:

print(f"[...] {evt.result.text}")

def recognized(evt):

# Finalized segments

if evt.result.text:

print(f"[Final] {evt.result.text}")

def canceled(evt):

print(f"[Canceled] Reason: {evt.reason}")

if evt.reason == speechsdk.CancellationReason.Error:

print(f"Error details: {evt.error\_details}")

def session\_stopped(evt):

print("[Session] Stopped.")

recognizer.recognizing.connect(recognizing)

recognizer.recognized.connect(recognized)

recognizer.canceled.connect(canceled)

recognizer.session\_stopped.connect(session\_stopped)

recognizer.start\_continuous\_recognition()

try:

input() # Wait for Enter

finally:

recognizer.stop\_continuous\_recognition()

# Give the SDK a moment to flush callbacks

time.sleep(0.5)

def main():

print("Azure AI Speech — Console Transcription")

print(f"- Region: {SPEECH\_REGION} | Language: {SPEECH\_LANGUAGE}")

print("Choose a mode:")

print("1) Single utterance (auto-stops after a pause)")

print("2) Continuous (press Enter to stop)")

choice = input("Enter 1 or 2: ").strip()

if choice == "1":

single\_utterance()

elif choice == "2":

continuous\_until\_enter()

else:

print("Invalid choice.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Run**

* **Install deps:**

pip install -r requirements.txt

* **Run app:**

python src/console\_stt.py

* **Speak and verify:** Try both modes. If you get silence, check Windows microphone privacy settings and your default input device.

**Step 2: Build a FastAPI-based API**

Takes audio input, transcribes it using Azure AI Speech, and returns the result via a REST endpoint. You’ll test it directly in the FastAPI docs UI (/docs) before we move to Gradio.

**🗂 Folder structure update**

azure-speech-stt/

├─ .env

├─ requirements.txt

├─ src/

│ ├─ console\_stt.py

│ └─ api/

│ ├─ main.py

│ └─ speech\_utils.py

**📦 Update requirements.txt**

Add FastAPI and Uvicorn:

Txt

azure-cognitiveservices-speech==1.38.0

python-dotenv==1.0.1

fastapi==0.111.0

uvicorn==0.29.0

Then run:

Bash

pip install -r requirements.txt

**🧠 speech\_utils.py — Transcription logi**

import os

import azure.cognitiveservices.speech as speechsdk

from dotenv import load\_dotenv

load\_dotenv()

SPEECH\_KEY = os.getenv("SPEECH\_KEY")

SPEECH\_REGION = os.getenv("SPEECH\_REGION")

SPEECH\_LANGUAGE = os.getenv("SPEECH\_LANGUAGE", "en-US")

def transcribe\_audio\_file(file\_path: str) -> str:

speech\_config = speechsdk.SpeechConfig(subscription=SPEECH\_KEY, region=SPEECH\_REGION)

speech\_config.speech\_recognition\_language = SPEECH\_LANGUAGE

audio\_config = speechsdk.audio.AudioConfig(filename=file\_path)

recognizer = speechsdk.SpeechRecognizer(speech\_config=speech\_config, audio\_config=audio\_config)

result = recognizer.recognize\_once\_async().get()

if result.reason == speechsdk.ResultReason.RecognizedSpeech:

return result.text

elif result.reason == speechsdk.ResultReason.NoMatch:

return "No speech recognized."

elif result.reason == speechsdk.ResultReason.Canceled:

return f"Canceled: {result.cancellation\_details.error\_details}"

return "Unknown error."

**🚀 main.py — FastAPI ap**

from fastapi import FastAPI, UploadFile, File

from fastapi.responses import JSONResponse

import os

from src.api.speech\_utils import transcribe\_audio\_file

app = FastAPI(title="Azure Speech-to-Text API")

@app.post("/transcribe")

async def transcribe(file: UploadFile = File(...)):

temp\_path = f"temp\_{file.filename}"

with open(temp\_path, "wb") as f:

f.write(await file.read())

try:

transcription = transcribe\_audio\_file(temp\_path)

return JSONResponse(content={"transcription": transcription})

finally:

os.remove(temp\_path)

**▶️ Run the API**

From the root folder:

Bash

uvicorn src.api.main:app --reload

Then open your browser to:

http://127.0.0.1:8000/docs

You’ll see the interactive Swagger UI. Upload a .wav or .mp3 file and test the /transcribe endpoint.

Perfect — now let’s build the **Gradio interface** so users can interact with your Azure Speech-to-Text API using a microphone or file upload, all wrapped in a clean UI

**🎨 Step 3: Add Gradio UI**

**create a new file**

azure-speech-stt/

├─ src/

│ ├─ console\_stt.py

│ ├─ api/

│ │ ├─ main.py

│ │ ├─ speech\_utils.py

│ └─ gradio\_app.py ← NEW

**📦 Update requirements.txt**

Add Gradio:

gradio==4.26.0

Then run:

Bash

pip install -r requirements.txt

**🎤 gradio\_app.py — Microphone + File Upload UI**

import gradio as gr

from src.api.speech\_utils import transcribe\_audio\_file

import tempfile

def transcribe\_gradio(audio):

if audio is None:

return "No audio provided."

with tempfile.NamedTemporaryFile(delete=False, suffix=".wav") as tmp:

tmp.write(audio)

tmp\_path = tmp.name

return transcribe\_audio\_file(tmp\_path)

with gr.Blocks(title="Azure Speech-to-Text") as demo:

gr.Markdown("## 🎙️ Azure Speech-to-Text\nUpload or record audio to transcribe using Azure AI Speech.")

with gr.Row():

audio\_input = gr.Audio(source="microphone", type="filepath", label="Record Audio")

file\_input = gr.Audio(source="upload", type="filepath", label="Upload Audio File")

output\_text = gr.Textbox(label="Transcription")

transcribe\_btn = gr.Button("Transcribe")

def handle\_transcription(mic\_path, file\_path):

audio\_path = mic\_path or file\_path

if not audio\_path:

return "Please provide audio input."

return transcribe\_audio\_file(audio\_path)

transcribe\_btn.click(handle\_transcription, inputs=[audio\_input, file\_input], outputs=output\_text)

demo.launch()

**🚀 Run the Gradio ap**

python src/gradio\_app.py

You’ll get a local URL like http://127.0.0.1:7860 — open it in your browser and test both the microphone and file upload options

**Swagger page**

