**Real-Time Deal Assistant Using GROQ API**

**Table of Contents**

1. Introduction
2. Phase 1: Planning
3. Phase 2: Analysis
4. Phase 3: Design
5. Phase 4: Implementation
6. Phase 5: Testing
7. Phase 6: Deployment
8. Code References

**1. Introduction**

The Real-Time Deal Assistant integrates speech-to-text capabilities, real-time data analysis, and Google Sheets for transcription and response management. It leverages threading for concurrency and APIs such as GROQ and Google Sheets to facilitate automated financial recommendations based on mutual fund datasets. By combining advanced speech recognition technologies with intelligent data analysis, the system enhances the efficiency and accuracy of financial advisory services.

This solution allows financial advisors to capture user input seamlessly through voice, analyze complex datasets to identify suitable investment products, and present actionable insights in real-time. The integration with Google Sheets ensures that all interactions are logged systematically, providing transparency and traceability. With its modular and scalable design, the Real-Time Deal Assistant is well-suited for dynamic, high-demand environments, making it a valuable tool for both financial professionals and investors.

**2. Phase 1: Planning**

Objectives:

* Automate financial advisory processes for mutual funds.
* Enable voice-based user input through speech recognition.
* Provide actionable insights using GROQ API and mutual fund data.
* Store queries and responses in Google Sheets and generate HTML-based responses.

Stakeholders:

* Financial advisors
* End-users (mutual fund investors)
* Developers and product managers

**3. Phase 2: Analysis**

Functional Requirements:

1. Speech Recognition: Convert user speech into text.
2. Data Analysis: Process user inputs using GROQ API for insights.
3. Data Storage: Log interactions in Google Sheets.
4. Response Generation: Display recommendations in an HTML format.

Non-Functional Requirements:

* Performance: Real-time processing of speech and data.
* Scalability: Handle multiple user requests simultaneously.
* Reliability: Ensure transcription accuracy and data integrity.

**4. Phase 3: Design**

Architecture:

The system comprises four main modules:

1. Controller Module (controller.py): Manages threading to run speech and data processing concurrently.
2. Speech Module (speech.py): Captures and transcribes user speech.
3. GROQ Integration Module (groq\_integration2.py): Analyzes queries and generates responses.
4. Google Sheets Module (google\_sheets\_util.py): Logs data into Google Sheets.

Sequence Diagram:

* User speaks into the microphone.
* Speech.py converts speech into text.
* Groq\_integration2.py processes the text and generates insights.
* Google Sheets logs the interaction.
* HTML updates display the response.

**5. Phase 4: Implementation**

Technologies:

* Programming Language: Python
* Libraries: threading, pyaudio, speech\_recognition, gspread, pandas
* APIs: GROQ, Google Sheets

**Key Code Features:**

**1. Controller Module:**

Coordinates the speech-to-text and data analysis processes using threading.

import threading

import speech

import groq\_integration2

def run\_speech():

speech.main()

def run\_groq():

groq\_integration2.main()

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

speech\_thread = threading.Thread(target=run\_speech)

groq\_thread = threading.Thread(target=run\_groq)

speech\_thread.start()

groq\_thread.start()

speech\_thread.join()

groq\_thread.join()

**2. Speech Module:**

Captures audio from a Virtual Audio Cable and converts it into text.

import pyaudio

import wave

import speech\_recognition as sr

def capture\_audio\_from\_vac(record\_seconds=10, output\_filename="vac\_audio.wav"):

p = pyaudio.PyAudio()

...

wf = wave.open(output\_filename, 'wb')

wf.writeframes(b''.join(frames))

wf.close()

def speech\_to\_text(audio\_filename="vac\_audio.wav"):

recognizer = sr.Recognizer()

with sr.AudioFile(audio\_filename) as source:

audio\_data = recognizer.record(source)

text = recognizer.recognize\_google(audio\_data)

with open("recognized\_text.txt", "w") as f:

f.write(text)

**3. GROQ Integration Module:**

Processes text and generates insights using GROQ API.

from groq import Groq

from google\_sheets\_util import write\_to\_sheet

def process\_text\_with\_groq(text):

client = Groq(api\_key="YOUR\_API\_KEY")

...

response = client.chat.completions.create(...)

return response.choices[0].message.content

def generate\_html\_response(response):

with open("assistant.html", "r+") as f:

content = f.read()

f.write(f"<li>{response}</li>")

**4. Google Sheets Integration:**

Handles authentication and writes data into Google Sheets.

import gspread

from oauth2client.file import Storage

def authenticate\_google\_sheets():

storage = Storage("storage.json")

credentials = storage.get()

return gspread.authorize(credentials)

def write\_to\_sheet(sheet\_name, data):

client = authenticate\_google\_sheets()

sheet = client.open(sheet\_name).sheet1

sheet.append\_row(data)

**6. Phase 5: Testing**

Test Cases:

1. Speech Recognition:
   * Input: User speaks into the microphone.
   * Expected Output: Transcribed text in recognized\_text.txt.
2. GROQ API Integration:
   * Input: Transcribed text.
   * Expected Output: Valid response based on mutual fund data.
3. Google Sheets Logging:
   * Input: Query and response.
   * Expected Output: Data appended to the specified Google Sheet.
4. HTML Response:
   * Input: Response text.
   * Expected Output: Properly formatted response displayed in the HTML file.

**7. Phase 6: Deployment**

Steps:

1. Push the source code to a secure GitHub repository.
2. Secure all sensitive information, such as API keys, using environment variables.
3. Set up a user-friendly interface for end-users.
4. Monitor and debug post-deployment.

**8. Code References**

1. Speech Module: Located in speech.py. Handles audio input and transcription.
2. GROQ Integration: Implemented in groq\_integration2.py. Processes text and generates actionable responses.
3. Google Sheets Integration: Found in google\_sheets\_util.py. Logs interactions into Google Sheets.
4. HTML Response Management: Updates dynamically in assistant.html.