### ****Tech Stack for Streamlit-based Tool****

1. **Frontend & Backend:**
   * **Streamlit**: For building the entire UI and managing interactions.
   * **Python**: As the backend language integrated within Streamlit for API calls and LLM integration.
2. **Database:**
   * **MongoDB**: For storing user-uploaded documents, agendas, discussion points, etc.
   * **Pinecone/Weaviate**: For storing and retrieving vector embeddings for RAG.
3. **File Storage:**
   * **Cloud Storage (e.g., AWS S3 or Google Cloud)**: For storing the meeting recordings and pre-meeting documents.
4. **LLM & AI Integration:**
   * **OpenAI GPT-4 API**: For generating meeting summaries.
   * **AssemblyAI/Deepgram**: For transcribing meeting recordings.
5. **Authentication:**
   * **Streamlit Authentication/Streamlit-Authenticator**: For managing user logins.
6. **APIs & Libraries:**
   * **LangChain**: For orchestrating the LLM, vector database, and other AI workflows.
   * **Multer (File handling)**: For handling file uploads in Streamlit.
   * **Requests**: For API calls (e.g., OpenAI API, transcription services).

### ****Step-by-Step Implementation****

#### **1. Set Up the Development Environment**

* Install **Streamlit** and the required libraries:

pip install streamlit pymongo pinecone-client openai langchain boto3 assemblyai

* **MongoDB** setup: Use MongoDB Atlas for cloud-based storage or run a local instance for development.
  + Install pymongo to interact with MongoDB:

pip install pymongo

* **Pinecone/Weaviate**: Set up an account on Pinecone or Weaviate for vector storage. Install the client library.

pip install pinecone-client

#### **2. Build the Pre-Meeting Document Management Feature**

* **File Upload with Streamlit**: In Streamlit, you can use the st.file\_uploader() to allow users to upload documents.

python

Copy code

import streamlit as st

import pymongo

client = pymongo.MongoClient("mongodb+srv://<username>:<password>@cluster0.mongodb.net/test")

db = client["meeting\_management"]

st.title("Upload Pre-Meeting Documents")

uploaded\_file = st.file\_uploader("Choose a file", type=["pdf", "docx"])

if uploaded\_file:

st.write("File uploaded successfully!")

# Store the file in MongoDB or cloud storage

db.files.insert\_one({"file\_name": uploaded\_file.name, "file\_data": uploaded\_file.read()})

* **Store Discussion Points**: Use Streamlit forms to capture discussion points:

python

Copy code

st.title("Add Discussion Points")

discussion\_point = st.text\_input("Enter a discussion point")

if st.button("Submit"):

db.discussion\_points.insert\_one({"point": discussion\_point})

st.write("Discussion point added!")

#### **3. Automatically Create an Agenda**

* **LLM-Generated Agenda**: Use GPT-4 to automatically generate the meeting agenda based on discussion points:

python

Copy code

import openai

st.title("Generate Meeting Agenda")

if st.button("Generate Agenda"):

discussion\_points = db.discussion\_points.find()

discussion\_points\_text = "\n".join([p["point"] for p in discussion\_points])

openai.api\_key = "your-openai-api-key"

response = openai.Completion.create(

model="gpt-4",

prompt=f"Generate an agenda from the following discussion points: {discussion\_points\_text}",

max\_tokens=500

)

st.write(response.choices[0].text)

#### **4. Meeting Recording & Tracking**

* **Upload Meeting Recording**: Similar to document upload, use st.file\_uploader() to handle meeting recordings:

python

Copy code

st.title("Upload Meeting Recording")

recording\_file = st.file\_uploader("Upload recording", type=["mp4", "mp3"])

if recording\_file:

st.write("Recording uploaded successfully!")

# Upload file to cloud storage (e.g., AWS S3)

* **Transcribe the Recording** (using AssemblyAI): After uploading, use the AssemblyAI API to transcribe the meeting.

python

Copy code

import requests

st.write("Transcribing Meeting...")

headers = {

'authorization': 'your-assemblyai-api-key',

'content-type': 'application/json'

}

upload\_url = "https://api.assemblyai.com/v2/upload"

response = requests.post(upload\_url, headers=headers, files={"file": recording\_file})

if response.status\_code == 200:

transcription\_id = response.json()['id']

st.write(f"Transcription ID: {transcription\_id}")

* **Track Meeting Progress**: Create a checklist UI to track whether each point is discussed:

python

Copy code

st.title("Track Meeting Progress")

discussion\_points = db.discussion\_points.find()

for point in discussion\_points:

st.checkbox(point["point"])

#### **5. Post-Meeting Summary with LLM**

* **Generate the Summary**: Use GPT-4 to generate a meeting summary, flag unresolved issues, and assign action items.

python

Copy code

st.title("Post-Meeting Summary")

if st.button("Generate Summary"):

transcription\_text = "Meeting transcription..." # from earlier transcription

response = openai.Completion.create(

model="gpt-4",

prompt=f"Summarize the following meeting and assign action items: {transcription\_text}",

max\_tokens=1000

)

st.write(response.choices[0].text)

#### **6. Integrate Vector Database (RAG)**

* **Store Documents as Embeddings**: Store pre-meeting documents and discussion points as vector embeddings in Pinecone:

python

Copy code

import pinecone

pinecone.init(api\_key="your-pinecone-api-key", environment="us-west1-gcp")

index = pinecone.Index("meeting-docs")

embeddings = openai.Embedding.create(input=discussion\_points\_text)["data"]

index.upsert(vectors=[(id, embedding, metadata)])

* **Retrieve Related Documents**: Use embeddings to retrieve relevant documents during summarization:

python

Copy code

query\_embedding = openai.Embedding.create(input=transcription\_text)["data"]

results = index.query(query\_embedding, top\_k=5)

### ****Final Notes****

* **Authentication**: Use **Streamlit-Authenticator** for basic login functionality, managing user sessions.
* **Deployment**: Streamlit makes it easy to deploy via Streamlit Cloud or services like Heroku.

This roadmap will help you create an efficient meeting management tool leveraging Streamlit and integrating AI-powered summaries and workflows.