# SYNOPSIS

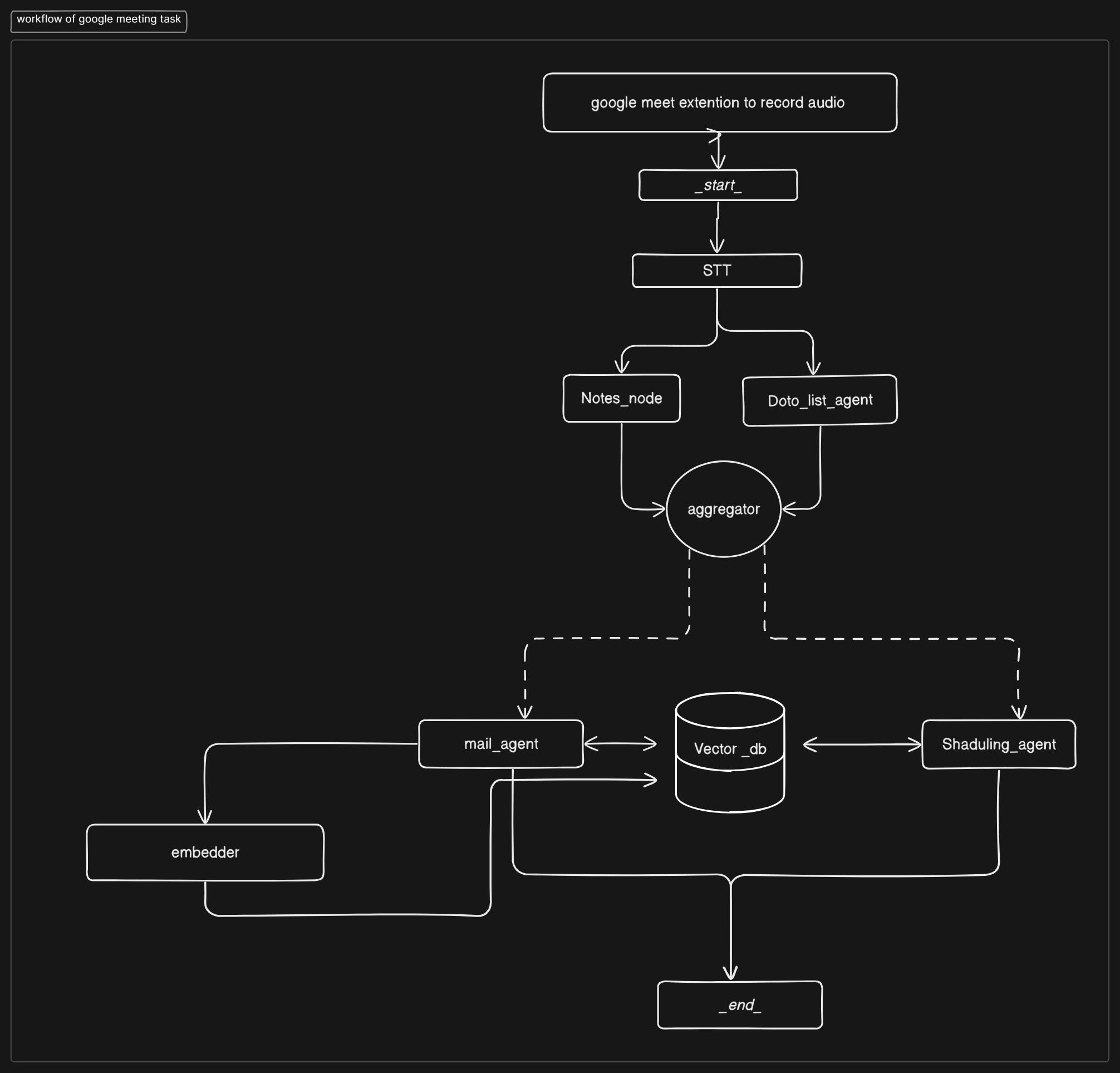
# ASEP PROJECT AIDS-C GROUP-5

# Synopsis: AI-Powered Digital Twin for Productivity Automation

The project titled "AI-Powered Digital Twin for Productivity Automation" aims to develop a web-based platform that harnesses artificial intelligence to enhance user productivity through automation and personalization. The core of the platform is an AI-powered digital twin that learns from a user’s online behavior, communication style, and decision-making patterns. Leveraging retrieval augmented reality. the AI adapts over time to closely mimic the user’s tone and writing style.  
  
This digital twin serves as a virtual assistant capable of automating various day-to-day activities. Key functionalities include generating context-aware email responses, composing emails based on user interests, managing calendar events, scheduling meetings, setting reminders. It can summers Google meet and add make todo-list which task also can be done by the workflow.  
  
The solution is designed to significantly enhance user productivity by streamlining tasks such as communication, scheduling, and information management, thereby allowing users to focus on more strategic and creative activities. Over time, the system becomes more intelligent and user-centric, offering a seamless and personalized user experience.

**System Architecture Overview**

The diagram represents a **client-server based email system** designed to facilitate both sending and receiving emails through a structured interaction between the frontend interface, backend server, and external mail servers.



### 1. Frontend Interface (Client Side)

The system's user interface is a web application where users can:

* Compose and send emails.
* View incoming messages.
* Receive feedback or status messages.

This interface collects input from the user and sends it to the backend using HTTP requests.

### 2. Backend Server (Application Layer)

The backend is developed using the **FastAPI framework** in Python. Its responsibilities include:

* Handling incoming requests from the frontend.
* Executing email-related logic such as formatting, validation, and routing.
* Managing communication with mail servers using standard protocols.

### 3. Email Protocol Integration

The backend utilizes standard email protocols:

* **SMTP (Simple Mail Transfer Protocol)** for sending emails.
* **IMAP (Internet Message Access Protocol)** or **POP3** for retrieving messages from the mail server.

These protocols ensure reliable communication between the application and external email services.

### 4. External Mail Servers

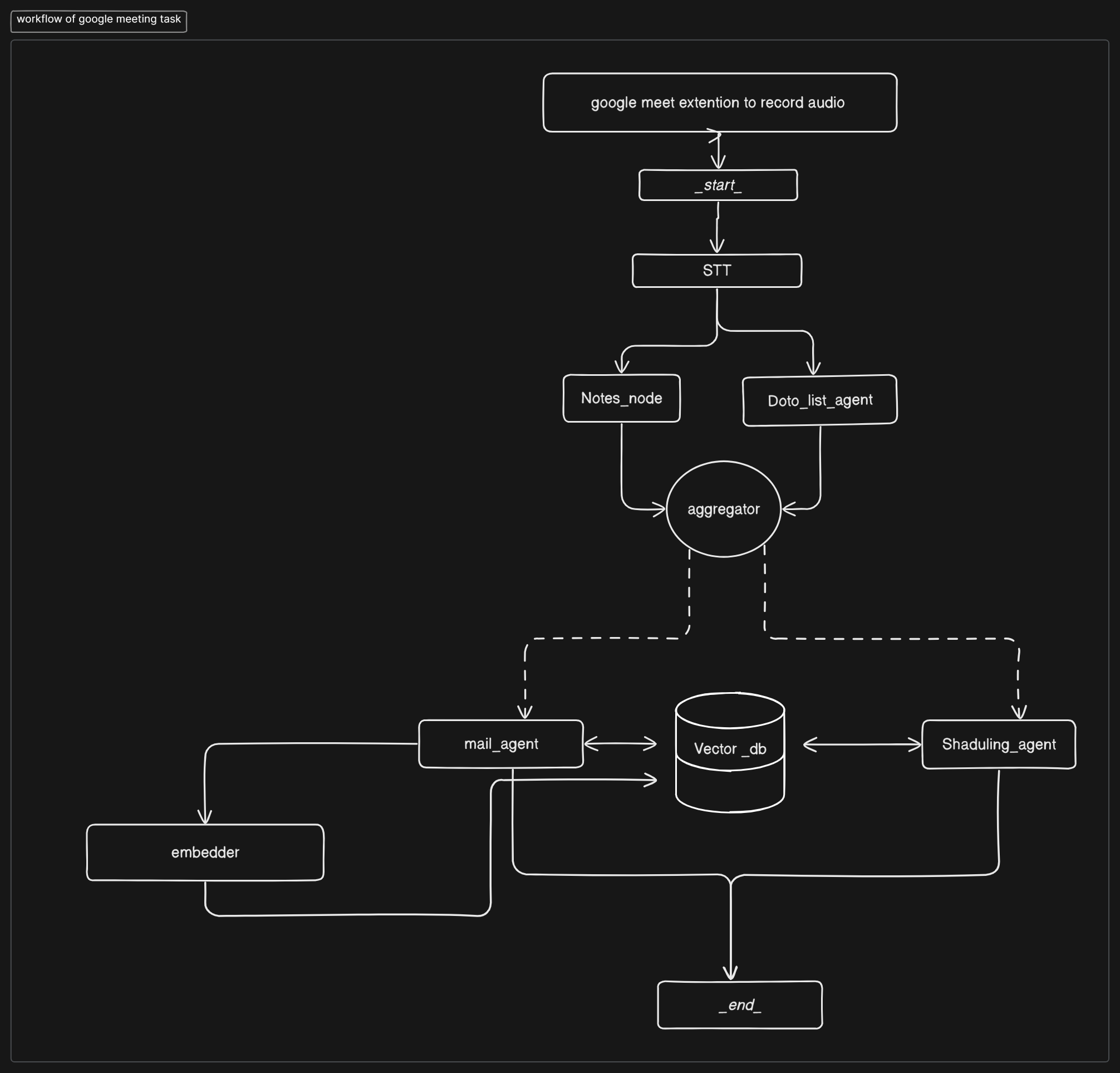
Mail servers (e.g., Gmail, Outlook) handle:

* Delivering outgoing messages to recipients.
* Storing and syncing incoming messages for retrieval.
* Ensuring security, delivery status, and inbox management.

### 5. Data Flow Summary

* **Sending Emails:**  
  User → Web Interface → FastAPI Server → SMTP → Mail Server → Recipient
* **Receiving Emails:**  
  Mail Server → IMAP/POP3 → FastAPI Server → Web Interface → User

**Automated Workflow for Google Meet Task Management**  
Powered by LangChain and LangGraph

**

### ****1. Audio Recording****

The system initiates with a **Google Meet extension** that records the meeting’s audio. When the recording starts, the workflow is triggered (\_start\_).

### ****2. Speech-to-Text (STT) Conversion****

The recorded audio is processed through a **Speech-to-Text (STT)** engine that transcribes the spoken content into text. This serves as the input for downstream language models.

### ****3. LangGraph-Based Information Routing****

The transcribed text flows through a **LangGraph**-based architecture that distributes data to two primary nodes:

* **Notes\_node**: Extracts concise summaries and key points from the meeting.
* **Todo\_list\_agent** Identifies and structures actionable tasks or follow-ups.

These nodes act as **LangChain agents** within a controlled workflow.

### ****4. Aggregation****

Outputs from both the Notes and Todo agents are sent to an **aggregator module**, which merges the structured information for downstream usage.

### ****5. Embedding and Vector Storage****

The aggregated data is passed through an **embedder**, which converts text into high-dimensional vectors. These embeddings are stored in a **Vector Database (Vector\_db)** using tools like **ChromaDB** or **Pinecone**, enabling future semantic search and recall.

### ****6. Agent-Based Task Execution****

Two intelligent agents, powered by LangChain, operate on the stored vectorized data:

* **mail\_agent**: Composes and sends context-aware follow-up emails based on extracted insights.
* **scheduling\_agent** (corrected from “shaduling\_agent”): Interfaces with calendar APIs to schedule meetings, set reminders, and allocate tasks.

These agents not only retrieve data from the vector store but also write updates back, ensuring a persistent state.

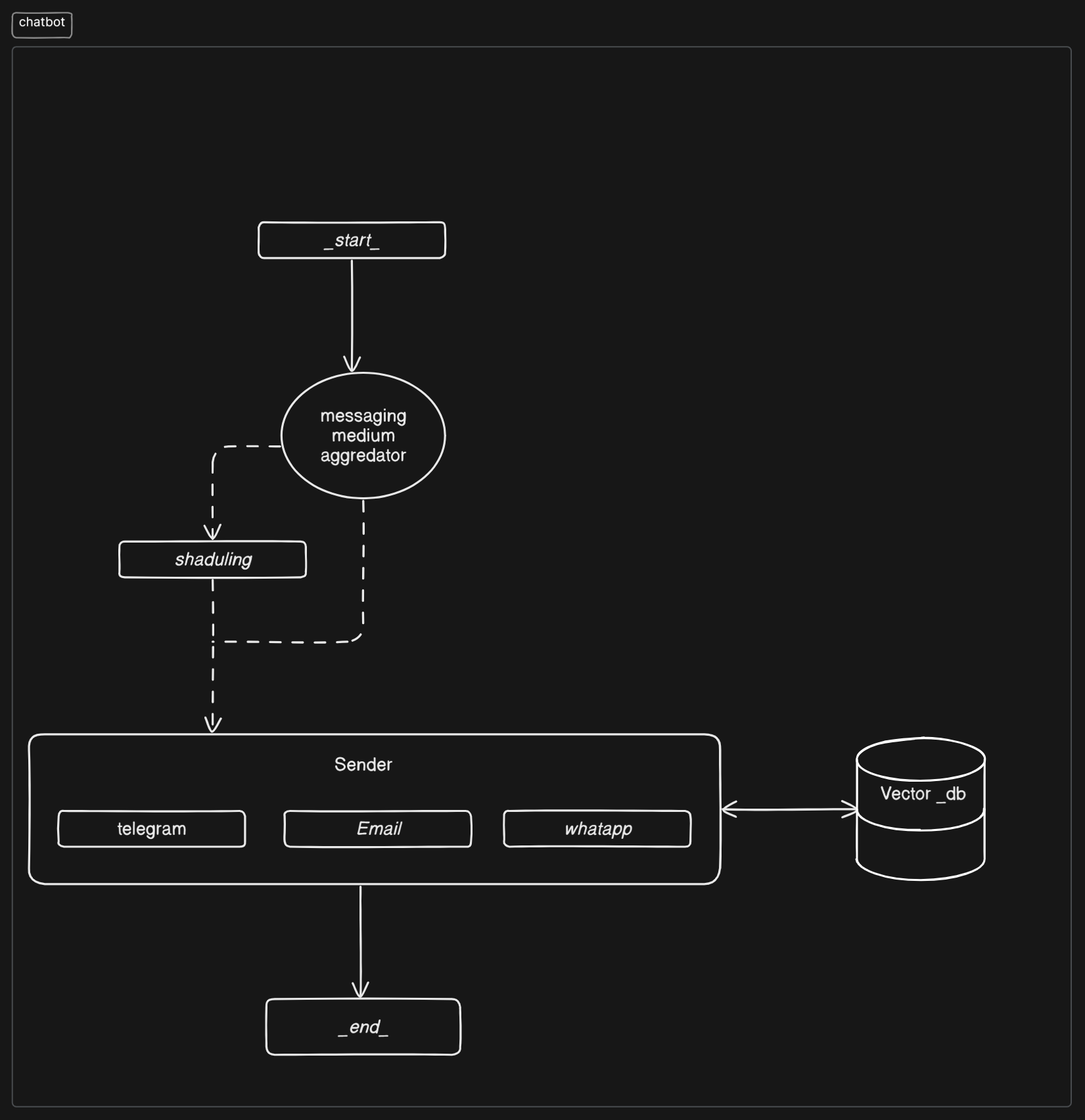
### ****7. Workflow Termination****

Once all tasks are executed—email dispatch, scheduling, and note logging—the workflow concludes with an \_end\_ marker.

### ****Technologies Used****

| **Component** | **Technology/Tool** |
| --- | --- |
| Audio Capture | Google Meet Extension |
| Transcription | Speech-to-Text API (e.g., Whisper, Google STT) |
| Workflow Management | **LangGraph** |
| Task Parsing Agents | **LangChain Agents** |
| Embedding & Storage | Embedding Models + **Vector DB** (ChromaDB, Pinecone) |
| Task Automation | Email and Calendar APIs via LangChain |

**Chatbot Messaging Workflow**  
*Powered by LangChain, LangGraph, and VectorDB*



### ****1. Workflow Initialization****

The process begins at the \_start\_ node, where the chatbot system is triggered to process and deliver a message across various platforms.

### ****2. Messaging Medium Aggregation****

The **messaging medium aggregator** determines the appropriate communication channels (e.g., Telegram, Email, WhatsApp) based on user preferences, message type, or urgency. This module acts as a LangGraph node responsible for intelligent routing.

### ****3. Scheduling Agent****

A **scheduling module** (labelled as “shaduling” in the diagram, should be “scheduling”) receives input from the aggregator and determines the appropriate time for sending the message. This may involve time-zone handling, delivery windows, or planned batch sends.

### ****4. Sender Module****

Once the medium and time are decided:

* The **Sender** module routes the message to the selected channel:
  + **Telegram**
  + **Email**
  + **WhatsApp**

This component accesses user-specific vectorized data from the **Vector Database (Vector\_db)** to personalize and format the message content before sending.

### ****5. Vector Database Integration****

The **Vector\_db** stores embeddings of historical conversations, message templates, or user preferences. It supports:

* Semantic message personalization
* Retrieval of context-aware responses
* Consistent communication tone across platforms

The Sender module reads from and optionally writes to this database for future use.

### ****6. Completion****

After the message is dispatched through the selected medium(s), the workflow concludes at the \_end\_ node.

### ****Technologies and Tools Used****

| **Component** | **Description / Tools** |
| --- | --- |
| Workflow Management | **LangGraph** |
| Message Aggregation | LangChain Node for medium selection |
| Scheduling | LangChain Agent + calendar/time APIs |
| Channel Communication | Telegram, Email (SMTP/SendGrid), WhatsApp API |
| Vector Intelligence | **Vector Database** (ChromaDB,superbase) |