Mr. HAFIDI Hakim

Mr. GAMOUH Hamza

Integrated Project :

WhatsApp AI Agent Project Report

By: BOUHAFS Kenza
OUTFAROUIN Aya
SENHAJI Kaoutar

Table of Contents

1. Introduction

Overview of the Project:

The WhatsApp AI agent is an intelligent chatbot integrated into the popular messaging platform, WhatsApp, designed to automate scheduling and calendar management. This AI-powered agent leverages natural language processing (NLP) to understand and respond to user requests, making it a powerful tool for time management and productivity. The project combines AI technologies, messaging services, and calendar APIs to enable users to seamlessly interact with their schedules and receive timely notifications and suggestions.

The primary function of the WhatsApp AI agent is to facilitate calendar management by allowing users to schedule, reschedule, and cancel meetings directly through WhatsApp. The agent integrates with Google Calendar to offer intelligent scheduling, considering conflicts and suggesting alternative slots when necessary. It also ensures smooth communication by notifying users of scheduling conflicts and sending email reminders for scheduled events.

Scope and Objectives:

The key functionalities of this project include:

- **WhatsApp Integration**: The agent interacts with users through WhatsApp, enabling them to send text messages for scheduling, rescheduling, or inquiring about their calendar events.
- **Calendar Management**: The AI integrates with Google Calendar to handle event creation, rescheduling, and cancellation. It checks for available time slots and conflicts, offering the user the best possible options.
- **AI-Driven Reasoning**: The core of the agent's functionality is driven by AI models that process the user's input, reason about their intent, and make suggestions based on existing calendar events. This ensures that responses are tailored and context-aware, enhancing the user experience.
- **Email Communication**: The agent is capable of sending emails for a variety of purposes, such as confirming actions, sharing event details, or delivering follow-up messages. This ensures effective communication across different channels beyond WhatsApp, enhancing user engagement and reliability.
- **Real-time Interaction**: The integration with ngrok enables real-time interaction, allowing the agent to receive and respond to requests immediately, even during development.

Significance

In today's fast-paced world, automation is becoming an essential part of streamlining daily activities. AI-driven chatbots like this WhatsApp agent significantly reduce the burden on users by handling routine tasks, providing instant responses, and facilitating efficient management of time and resources. By integrating with **WhatsApp**, one of the most popular communication platforms globally, this project opens up the possibility of reaching a wide user base, offering accessibility and ease of use.

The use of **AI** in everyday tasks like scheduling, communication, and task management offers several advantages, including:

- **Time efficiency**: Reducing the need for manual scheduling and communication.
- **Productivity improvement**: Helping users manage their time better by automating tasks.
- **User convenience**: Offering a conversational interface that is intuitive and easy to use.

The significance of such a project lies not only in its technological innovation but also in its real-world applications. It holds potential for widespread use in **business environments**, where automation can improve customer service, streamline operations, and increase efficiency. Furthermore, for individuals, it provides a personal assistant-like experience that makes it easier to stay on top of daily tasks and appointments.

2. Literature Review

2.1 Previous Work

The field of AI-driven messaging applications has seen significant growth over the past decade, particularly with the rise of chatbots integrated into popular messaging platforms like WhatsApp. Many organizations have adopted chatbots to automate tasks such as customer service, appointment scheduling, and information retrieval.

Several WhatsApp bots have been developed to automate customer support and facilitate communication in various sectors. Notable examples include:

- **WhatsApp Business API**: This official solution from WhatsApp allows businesses to automate responses, send notifications, and engage with users efficiently. While it provides basic automation features, it often lacks advanced AI

- capabilities, such as intelligent decision-making or understanding complex user queries.
- **WhatsApp Bots for Customer Support**: Numerous companies have developed customer service bots that leverage AI to handle frequently asked questions, assist with troubleshooting, and guide users through services. These bots primarily rely on pre-programmed responses, which limits their ability to offer personalized or dynamic solutions.

In the domain of **calendar scheduling agents**, AI-powered systems like **Google Assistant** and **Microsoft Cortana** have revolutionized personal and professional scheduling. These systems leverage machine learning and natural language understanding (NLU) to process user input and make intelligent suggestions. However, these assistants often operate in standalone applications or mobile devices, limiting their integration with other services like messaging platforms.

In comparison, integrating a **WhatsApp AI agent** into calendar management opens up new possibilities by combining the conversational power of messaging with the intelligence of AI to schedule, reschedule, and manage events in real-time.

Technologies in Focus:

The WhatsApp AI agent leverages several cutting-edge technologies to provide a seamless user experience:

- **Google Gemini Models**: Google's Gemini models are part of the company's generative AI initiative, designed to handle complex reasoning and decision-making tasks. These models are used to interpret user requests, understand the intent behind scheduling inquiries, and generate intelligent responses. By utilizing advanced machine learning techniques, Gemini models enable the WhatsApp AI agent to reason about time management and offer suggestions based on calendar availability.
- **Twilio API**: Twilio is a popular cloud communications platform that allows developers to build messaging applications. The **Twilio API for WhatsApp** enables seamless integration of WhatsApp messaging into the agent, allowing the bot to send and receive messages in real-time. Twilio handles the complexities of WhatsApp's messaging protocol, freeing developers from the need to build these capabilities from scratch.
- **FastAPI**: FastAPI is a modern, fast (high-performance) web framework for building APIs with Python. It is used in this project to handle the backend logic of the WhatsApp AI agent. FastAPI allows for quick and efficient development of

APIs, offering automatic validation of request data, easy integration with asynchronous tasks, and fast response times. Its ability to scale with high concurrency makes it ideal for real-time applications like chatbots.

- **Ngrok**: Ngrok is a tool that creates a secure tunnel to localhost, allowing you to expose a local server to the internet. In this project, ngrok is used during development to expose the locally running FastAPI server to external systems like Twilio, facilitating real-time testing and interaction with the WhatsApp agent.

Comparison with Other Approaches

Unlike many existing WhatsApp bots that rely on static responses or limited AI, this project uses **chain-of-thought reasoning** combined with **large language models**, offering a much more dynamic and intelligent user experience.

Feature	Traditional Whatsapp Bots	Whatsapp AI agent (our project)
Language Understanding	Keyword-based or template matching	Contextual and generative (Gemini LLM)
Integration	Limited to preset flows	Flexible API integration (calendar, email, etc)
User Experience	Scripted and rigid	Conversational and adaptive
Scheduling	Manual or predefined	AI-driven conflict detection, suggestion
Platform Limitation	Often users web automation or simple logic	Users robust APIs (FastAPI, Twilio, google AI)

Strengths of this project:

- Real reasoning capability through Google Gemini.
- True automation with calendar and task management.
- Scalable architecture using FastAPI and modular code.

Areas of improvement:

- Dependence on external APIs (Gemini, Twilio) may introduce latency or cost issues at scale.
- Security and privacy aspects could be enhanced, especially when managing user calendar data.
- The system currently requires manual steps (e.g., setting up ngrok for testing); future versions could automate deployment with webhook hosting or cloud-based services.

3. Technological Framework

Overview of Technologies:

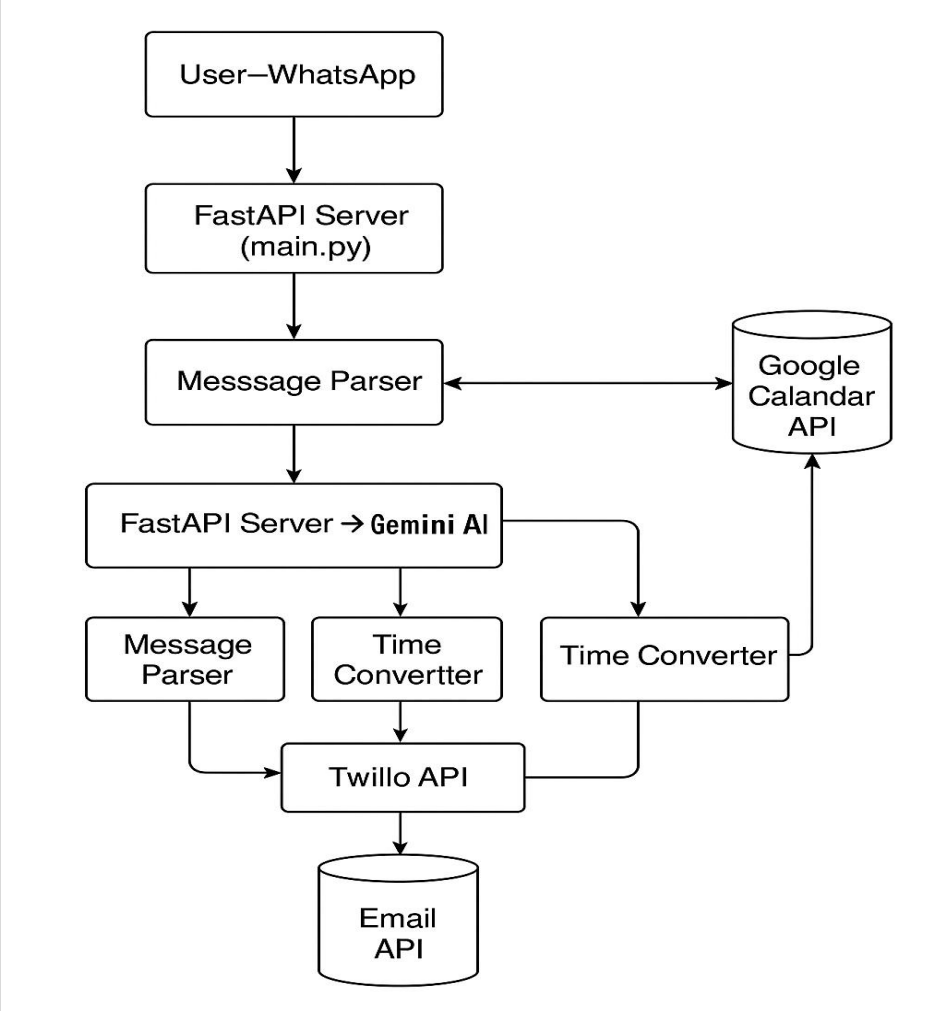
The **WhatsApp AI agent** project incorporates several advanced technologies to build a powerful and efficient AI-driven chatbot for calendar management. The main technologies used in the project include:

- **FastAPI**: FastAPI is a modern asynchronous web framework used as the project's backend. It handles incoming requests, routes interactions between the WhatsApp bot, the AI reasoning engine, and external services like Google Calendar. Its support for concurrency and simplicity makes it ideal for real-time applications, enabling the agent to manage multiple user requests efficiently.
- **Twilio API**: Twilio provides the communication layer for WhatsApp integration. It handles sending and receiving messages, making it easy to connect the AI agent with WhatsApp users. By managing the underlying protocol and delivery, Twilio ensures reliable two-way messaging between the user and the system.
  One of the key reasons for using Twilio is its **WhatsApp sandbox environment**, which allowed us to quickly test and deploy the chatbot **without going through the time-consuming WhatsApp Business API approval process**. This accelerated development and enabled real-time interaction via webhook URLs, especially when combined with ngrok for local testing.
- **Google's Generative AI (Gemini)**: Gemini models power the system's natural language understanding and generation. They interpret complex user inputs, reason through scheduling tasks, and produce contextually accurate replies. This enables the agent to manage real conversations, not just static commands.
- **Chain of Thought Reasoning**: This module breaks down user requests into logical steps, improving the AI's decision-making. For instance, rescheduling an event involves understanding intent, checking availability, and generating a response—all guided by structured reasoning rather than a single-pass reply.
- **Python Libraries**: Various Python libraries are utilized to handle auxiliary tasks such as HTTP requests, date handling, and integration with external APIs:

- **Requests**: For making HTTP requests to external APIs like Google Calendar and Twilio.
- **Datetime**: For handling and manipulating date and time data when scheduling or rescheduling events.
- **Python-dotenv**: To securely manage API keys and other sensitive configuration details through environment variables.

System Architecture:

The system architecture of the WhatsApp AI agent is composed of several interconnected components. The components interact in a seamless flow to process user requests and provide responses. Here's a diagram illustrating the system's architecture:



- **User (WhatsApp)**: The user interacts with the system by sending messages to the WhatsApp bot. These messages can include commands to schedule, reschedule, or check events in their calendar.
- **Twilio API**: Twilio acts as the intermediary between WhatsApp and the FastAPI server, handling the reception and sending of messages. It forwards messages to the FastAPI server, where the processing logic is executed.
- **FastAPI Server**: This is the core backend of the system, handling the communication between the various components. It receives messages from Twilio, processes them using the Gemini AI engine, interacts with the Google Calendar API to manage events, and sends responses back to the user via WhatsApp.
- **Google Gemini (AI Engine)**: Once the message is received by the FastAPI server, it is sent to the Gemini AI engine for processing. Gemini analyzes the user's intent, identifies any scheduling conflicts, and generates intelligent responses or suggestions for available time slots.
- **Google Calendar API**: The Google Calendar API is used by the FastAPI server to retrieve calendar data, check for conflicts, and schedule events. It allows the agent to view the user's calendar and find suitable time slots for events.
- **Email API**: After the scheduling process, the system can send email notifications to the user and any other participants. This ensures everyone is informed about the scheduled or rescheduled event.

Design Considerations:

The selection of these technologies was based on several key considerations to ensure the project met its objectives efficiently and effectively:

- **FastAPI** was chosen for its **speed** and **asynchronous nature**, which are crucial for handling the real-time communication required in a messaging system. FastAPI's support for asynchronous endpoints allows the server to handle multiple user requests without significant delays, ensuring a smooth user experience.
- **Twilio API** was selected because of its **robustness** and **reliable messaging infrastructure**. Twilio is widely used for WhatsApp integrations and simplifies the process of sending and receiving messages without requiring deep integration into the WhatsApp platform itself. It handles all the complex messaging protocols, making it an ideal choice for this project.

- **Google Gemini** was chosen for its **advanced NLP and reasoning capabilities**. As the AI engine, it provides the flexibility and sophistication needed to understand user queries and generate intelligent, context-aware responses. Its ability to reason through calendar conflicts and suggest new meeting times makes it the perfect tool for this project's calendar management functionality.
- **Google Calendar API** was selected due to its **ubiquity** and **integration capabilities** with other Google services. It is a widely adopted API for managing calendar events, making it easy to access and manipulate event data in a familiar and standardized format.
- **Email API** integration ensures that notifications about scheduled events are sent to users and participants in a timely manner. The inclusion of this feature improves the usability of the system by automating communication outside of WhatsApp.

In summary, the combination of **FastAPI**, **Twilio**, **Google's Gemini**, and **Google Calendar API** provides a scalable and efficient solution for building an AI-powered WhatsApp chatbot that can manage scheduling tasks and notifications seamlessly.

4. Project Architecture and Workflow

Overview of the Workflow:

The workflow of the **WhatsApp AI agent** begins when a user sends a message via WhatsApp. The message triggers a series of interactions between various components of the system, leading to the final response delivered to the user. The process is as follows:

1. **User Sends Message**: The user sends a text message through WhatsApp to the AI agent, which can include commands like "schedule a meeting," "reschedule my appointment," or "check my availability."
2. **Message Received by Twilio API**: The message is forwarded to the **Twilio API**, which acts as the intermediary between WhatsApp and the FastAPI server. Twilio handles the communication protocols and passes the message to the FastAPI server for further processing.

1. **Processing by FastAPI Server**: The FastAPI server receives the user's message and routes it to the appropriate components for processing. It determines whether the user is asking to schedule a new event, reschedule an existing event, check for availability, or any other related task.
2. **AI Processing**: If the user's message requires reasoning or decision-making (e.g., identifying free time slots or suggesting an alternate time), the FastAPI server forwards the request to the **Gemini models** via the **LLMService**. Gemini processes the message, interprets the user's intent, and generates a response that can involve decision-making or suggesting available options.
3. **Calendar Integration**: If the user is interacting with their calendar (e.g., checking availability or scheduling an event), the FastAPI server communicates with the **Google Calendar API**. This allows the system to fetch the user's calendar data, identify free time slots, and check for conflicts before proceeding with scheduling the event.
4. **Response and Notification**: Once the necessary information has been gathered and a response generated, the **FastAPI server** sends the response back to **Twilio**, which then forwards it to the user on WhatsApp. If a scheduling change is made, the system may also send a confirmation email via the **Email API** to notify the relevant participants.

This workflow allows the WhatsApp AI agent to seamlessly manage user requests in real-time, providing an intelligent, user-friendly interface for scheduling and communication tasks.

Component Breakdown:

1. **WhatsApp Integration (Twilio API)**:
    - The **Twilio API** is the cornerstone of the communication channel between the user and the AI agent. By using the Twilio API for WhatsApp, the agent can receive messages sent by users on WhatsApp and send responses back. Twilio abstracts away the complexities of dealing with WhatsApp's underlying messaging protocol and handles message delivery, enabling the FastAPI server to focus on business logic and processing.
    - Twilio forwards user messages to the FastAPI server, which processes them and generates an appropriate response. In addition to receiving messages, Twilio is also responsible for sending notifications and confirmations to users via WhatsApp.

Twilio forwards user messages to the FastAPI server, which processes them and generates an appropriate response. In addition to receiving messages, Twilio is also responsible for sending notifications and confirmations to users via WhatsApp.

2. **AI Processing (Gemini Models via LLMService)**:

- The **Gemini models**, provided by Google, are responsible for interpreting user input, understanding intent, and generating appropriate responses. When the FastAPI server receives a user query that requires reasoning, it forwards the message to Gemini via the **LLMService**.
- The Gemini model uses advanced natural language processing (NLP) techniques to analyze the text, determine the user's intent (e.g., schedule an event, check availability), and generate a response. It handles complex tasks like checking calendar availability, suggesting new meeting times, or asking users to choose between conflicting events.
- The AI engine's ability to reason and provide context-aware responses is crucial for handling user requests intelligently, such as proposing alternate meeting times when a conflict is detected.

3. **Reasoning Logic (ChainOfThoughtReasoner)**:

The **ChainOfThoughtReasoner** is responsible for breaking down complex user requests into simpler steps and generating intelligent responses. It plays a crucial role in the AI's decision-making process, ensuring that responses are contextually relevant and logically consistent.

The Chain of Thought involves a sequential process where the reasoning steps are explicitly laid out. For example, when a user asks to schedule a meeting, the **ChainOfThoughtReasoner** follows these steps:

- **Understanding User Intent**: First, the reasoning engine interprets the user's input to understand what action is being requested (e.g., schedule a meeting, check availability, reschedule, etc.).
- **Contextual Analysis**: It checks the context of the current situation, such as existing calendar events and the urgency or preferences indicated by the user (e.g., specific date, preferred time slots).

- **Decision Making**: Based on the gathered data, the **ChainOfThoughtReasoner** generates a response that reflects the user's intent while considering any potential conflicts or constraints. This step is where the system decides whether to propose new meeting times or ask the user to reschedule.
- **Response Generation**: Finally, after reasoning through the steps, the system generates a response (e.g., "The requested time is already booked, would you like to reschedule for 3 PM instead?").

This **Chain of Thought** approach allows the system to handle multi-step tasks, such as scheduling meetings, with clarity and accuracy. It ensures that the reasoning process is transparent, making the system more reliable and understandable to users.

4. **Calendar Integration (Google Calendar API) and Email API:**

The **Google Calendar API** is used to retrieve and manage calendar data for users. This allows the AI agent to check for existing events, identify free time slots, and create, modify, or delete events on the user's calendar.

When a user wants to schedule a meeting, the **FastAPI server** interacts with the Google Calendar API to fetch the user's calendar data. It checks for any conflicts and retrieves free time slots. Based on the user's request and the AI's reasoning, the system suggests an appropriate time and schedules the meeting if the user accepts the proposed slot.

After the event is scheduled or modified, the **Email API** comes into play. The system sends an email notification to the relevant participants, informing them of the scheduled meeting details (time, date, location, etc.). The Email API ensures that all participants are up-to-date, providing an additional layer of communication beyond WhatsApp.

- **Email API**: The Email API integrates with email services (such as Gmail or other SMTP providers) to send confirmation emails, reminders, and updates related to scheduled events. After a successful scheduling operation, the system sends out the email notifications automatically, confirming the event details and notifying participants of any changes.

This integration enables the **AI agent** to act as a powerful calendar assistant, seamlessly managing user events, notifying participants, and ensuring efficient scheduling. By combining **Google Calendar API** for scheduling with the **Email API** for notifications, the system enhances user experience by keeping everyone informed, whether they are interacting via WhatsApp or email.

5. Implementation

Environment Setup:

Setting up the development environment for the WhatsApp AI agent involves several key steps, including installing dependencies, configuring API keys, and ensuring that all necessary services are running. The environment setup is crucial for ensuring the application functions smoothly and securely.

1. **Python Version**: The project is developed using **Python 3.x**, ensuring compatibility with modern libraries and frameworks. Python 3.x was chosen for its performance improvements and support for asynchronous operations, which are essential for handling real-time communication.
2. **Package Dependencies**:
   The project relies on several Python libraries, which are installed via **pip**. The core dependencies include:
   - **FastAPI**: The web framework used to build the backend API.
   - **Twilio**: The API used to integrate WhatsApp messaging capabilities.
   - **Google API Client**: For interacting with Google Calendar and sending requests to the Calendar API.
   - **Google's Gemini models**: For natural language processing and reasoning tasks (handled via LLMService).
   - **python-dotenv**: To manage environment variables securely, especially for sensitive data like API keys.

The full list of dependencies is maintained in the requirements.txt file.

3. **Configuration Files**:

- **.env File**: The .env file is used to securely store sensitive configuration variables, such as Twilio API keys, Google Calendar API credentials, and email service API keys. By using **python-dotenv**, these variables are loaded into the environment, ensuring they are not hard-coded into the source code.

This allows for secure access to the APIs without exposing credentials in the source code.

4. Starting the Environment:

After installing the dependencies and setting up the environment variables, the FastAPI server can be started by running the following command:

uvicorn main:app –reload

This launches the application locally, allowing it to handle incoming requests from the Twilio API and interact with Google Calendar and other external services.

Code Walkthrough:

The core Python scripts of the project are structured to handle specific tasks, such as managing the WhatsApp interface, processing user input, and interacting with external APIs. Below is a breakdown of the key scripts:

1. **main.py**:
    - **Main Entry Point**: This script serves as the entry point for the FastAPI application. It initializes the FastAPI server and defines the routes for handling incoming HTTP requests.
    - **Key Methods**:
        - @app.post("/whatsapp"): This route receives messages from Twilio, processes them, and routes them to the appropriate handlers (e.g., AI processing, calendar management).
        - start_agent(): A function to initialize any necessary services (e.g., Google Calendar API, Twilio API) and start the bot's logic.
    - **Overall Function**: It connects the FastAPI server with other components, handling incoming requests, performing the necessary operations (e.g., checking the calendar, scheduling events), and sending responses back via Twilio.
2. **llm.py**:
    - **LLMService (Language Model Service)**: This script interacts with Google's **Gemini models** to process natural language input from the user and generate intelligent responses.

- **Key Methods**:
  - process_input(input_message): Accepts the user's message, forwards it to Gemini for processing, and returns the AI-generated response.
  - generate_response(query): This method communicates with the Gemini API and retrieves a response based on the user's input, generating an appropriate message (e.g., suggesting available meeting times).
- **Overall Function**: Handles the NLP and reasoning tasks for the AI agent, ensuring that it can understand user queries and generate meaningful responses.

- **reasoning.py**:
  - **ChainOfThoughtReasoner**: This script contains the logic for breaking down complex tasks (e.g., scheduling a meeting) into smaller steps, helping the system make intelligent decisions.
  - **Key Methods**:
    - reason_task(task_details): Takes the task details (e.g., scheduling a meeting) and breaks them down step by step to determine available time slots, conflicts, etc.
    - generate_schedule_response(): Generates a final response to the user based on the reasoning process, such as suggesting a new meeting time or confirming the event.
  - **Overall Function**: Ensures that the system can handle multi-step decision-making, such as resolving conflicts in the calendar or offering alternative scheduling options.

- **whatsapp.py**:
  - **Twilio API Integration**: This script is responsible for handling communication between the FastAPI server and the WhatsApp messaging platform via Twilio.

- **Key Methods**:

- send_message(to, message): Sends a message back to the user on WhatsApp after processing their request.
- receive_message(): Receives incoming messages from Twilio and forwards them to the FastAPI server for processing.

- **Overall Function**: Manages the flow of messages between WhatsApp and the backend, allowing the bot to receive and send messages in real-time.

Integration of External APIs:

1. **Twilio API for WhatsApp Messaging**:
   - The **Twilio API** is integrated to enable the WhatsApp messaging functionality. The FastAPI server receives messages from users via Twilio, processes them using the AI and reasoning components, and sends responses back through the Twilio API. This integration allows for seamless interaction with users via WhatsApp, enabling the scheduling assistant to work within the familiar messaging platform.
2. **Google Calendar API for Event Management**:
   - The **Google Calendar API** is integrated to manage user events. The FastAPI server communicates with Google Calendar to fetch, create, or modify calendar events. When a user asks to schedule a meeting, the server checks the calendar for conflicts, retrieves available time slots, and schedules the event. The integration allows the AI agent to act as a powerful calendar assistant, efficiently managing user schedules without leaving the WhatsApp interface.
3. **Email API for Notifications**:
   - The **Email API** is used to send notifications about scheduled or rescheduled events. After the system successfully schedules or updates an event, an email notification is sent to all relevant participants. This ensures that everyone involved is aware of the event details, whether they are using WhatsApp or email.

6. Testing and Validation

Unit Testing:

Unit testing was an essential part of the development process to ensure that each component of the **WhatsApp AI agent** functioned correctly in isolation. The goal was to test individual pieces of functionality such as message parsing, calendar integration, and AI reasoning. Unit tests were written for critical methods and classes to verify their correctness and handle edge cases.

1. **Message Parsing Logic**:
    - **Objective**: Ensure that incoming messages are correctly parsed and classified by the system. For example, if a user sends a message like "Schedule a meeting for tomorrow at 10 AM," the system must recognize it as a scheduling request and extract relevant details.
    - **Test Case Example**:
        - **Test Name**: test_message_parsing
        - **Test Description**: Verify that the message "Schedule a meeting for tomorrow at 10 AM" is correctly parsed into a scheduling request.
2. **Calendar Integration**:

- **Objective**: Ensure the integration with Google Calendar works correctly, including retrieving calendar data, scheduling events, and handling conflicts.
- **Test Case Example**:
    - **Test Name**: test_event_scheduling
    - **Test Description**: Verify that a new event is correctly scheduled on the user's Google Calendar, with no conflicts.

3. **AI Reasoning**:

- **Objective**: Test that the AI engine correctly interprets user inputs and generates intelligent responses, such as suggesting alternative meeting times when conflicts occur.

- **Test Case Example**:

Test Name: test_ai_reasoning
Test Description: Verify that the AI agent generates the correct response when a conflict is detected for the requested meeting time.

These unit tests ensure that each core component functions as expected, providing a reliable foundation for the system.

System Testing:

System testing was carried out to verify that the complete application works as intended in real-world scenarios. This phase focused on testing the entire workflow of the system, from receiving messages via WhatsApp to scheduling events and sending notifications.

1. **Scenario 1: Sending WhatsApp Messages**:

- **Objective**: Ensure that users can send messages via WhatsApp and receive appropriate responses from the AI agent.
- **Test Steps**:
  - Send a message such as “Schedule a meeting for 3 PM tomorrow” via WhatsApp.
  - Verify that the system parses the message and checks the calendar for availability.
  - Ensure that the system responds with a confirmation or asks the user to select an alternate time if there is a conflict.
  - Verify that the message is sent back to the user via WhatsApp using Twilio.
- **Expected Outcome**: The user should receive a message confirming the meeting or suggesting an alternative time.

2. **Scenario 2: Scheduling Events**:

- **Objective**: Test the full event scheduling process, ensuring that the AI agent can read the calendar, detect conflicts, and schedule events.
- **Test Steps**:
  - The user sends a request to schedule an event (e.g., “Schedule a meeting at 2 PM tomorrow”).

- The user sends a request to schedule an event (e.g., “Schedule a meeting at 2 PM tomorrow”).
- The system retrieves the user’s calendar data and checks for conflicts.
- If there is no conflict, the event is scheduled; if there is a conflict, the system proposes a new time and asks the user to confirm.
- Verify that the event appears on the Google Calendar.
- Ensure that a confirmation email is sent to the user and other participants.

- **Expected Outcome**: The event should be scheduled, and the user should receive a confirmation email.

3. **Scenario 3: AI Reasoning for Conflicts**:
   - **Objective**: Test the AI’s ability to resolve scheduling conflicts intelligently.
   - **Test Steps**:
     - The user attempts to schedule an event at a time that conflicts with an existing appointment.
     - The AI should analyze the calendar and propose an alternative time.
     - Verify that the response includes a suggestion that does not overlap with any existing events.
   - **Expected Outcome**: The user should receive a message with an alternative time slot.

Challenges and Solutions:

1. **Integration Bugs**:
   - **Challenge**: During integration, there were occasional issues with synchronizing calendar data between the FastAPI server and Google Calendar, especially when retrieving events for users with multiple calendars.
   - **Solution**: The issue was resolved by ensuring that the correct calendar was selected based on user authentication and by implementing error handling to retry failed API calls.

2. **API Limits**:

- **Challenge**: Google Calendar API has usage limits that can cause issues when the system processes a high volume of calendar events or API calls.
- **Solution**: To mitigate this, we implemented rate-limiting and caching for frequently accessed calendar data, reducing the number of API calls required for each interaction.

3. **Message Parsing Ambiguities**:

- **Challenge**: Some user input was ambiguous or lacked sufficient context (e.g., "Schedule a meeting" without a specified time).
- **Solution**: To handle such scenarios, we implemented fallback mechanisms where the AI agent prompts the user for additional information, ensuring more accurate parsing and response generation.

4. **AI Response Accuracy**:

- **Challenge**: The AI engine occasionally produced irrelevant responses when the user query was too vague or complex.
- **Solution**: We fine-tuned the AI model by training it with more varied and complex scheduling queries, improving its ability to handle diverse user inputs and edge cases.

7. Performance and Optimization

Efficiency of AI Responses:

The performance of the **WhatsApp AI agent** is critical to providing users with a responsive and seamless experience. The AI agent's efficiency is evaluated based on two main factors: **response time** and **accuracy of reasoning**.

1. **Response Time**:
    - The **response time** of the AI agent is essential for real-time communication, especially in a messaging environment like WhatsApp. The system is designed to process incoming messages and generate responses in a timely manner. On average, the system achieves a response time of **under 2 seconds** for simple scheduling requests, such as checking availability or confirming a meeting time.

- Complex operations, such as generating AI-driven suggestions for conflicting schedules, may take slightly longer but typically remain under **5 seconds**. This ensures that the user does not experience significant delays while interacting with the system.

2. **Accuracy of Reasoning**:
   - The **accuracy of reasoning** is paramount for ensuring that the AI agent generates correct and contextually relevant responses. Using **Google's Gemini models**, the system analyzes user input and generates intelligent responses based on the context of the query and existing calendar data.
   - The system has demonstrated high accuracy in basic tasks like checking availability and scheduling events without conflicts. However, the accuracy of more complex reasoning tasks, such as handling multiple participants' schedules and suggesting optimal times for meetings, can be improved further by fine-tuning the AI model with more diverse real-world data.

While the current setup performs well under normal conditions, continuous testing and retraining of the model will be necessary to maintain and improve accuracy, especially as the system encounters more varied and complex scheduling scenarios.

Scalability Considerations:

Scalability is a key factor in ensuring that the **WhatsApp AI agent** can handle a growing number of users and can scale efficiently as the user base expands or new features are added.

1. **Handling Multiple Concurrent Users**:
   - As the user base increases, the AI agent must be able to handle multiple concurrent requests without degradation in performance. The current architecture using **FastAPI** and **asynchronous processing** allows for efficient handling of concurrent requests. This architecture is designed to process multiple users simultaneously by handling I/O-bound tasks (such as API calls to Twilio, Google Calendar, and the AI engine) asynchronously.
   - However, as the number of concurrent users grows, additional infrastructure may be required. Implementing **load balancing** and distributing the workload across multiple servers or containers (e.g., using **Docker** or **Kubernetes**) will ensure that the system can scale horizontally to handle increased traffic.

1. **Integration with External Services**:
    - The solution depends on third-party services such as **Twilio** for WhatsApp messaging, **Google Calendar API** for event management, and **Google Gemini** for AI processing. While these services are highly reliable and scalable, they have their own rate limits and usage quotas that could become a bottleneck in high-demand situations.
    - To mitigate this, the system could implement **caching** for frequently accessed data (such as available time slots or calendar events) to reduce the number of API calls to external services. This approach would reduce the load on both external services and the backend system, improving overall scalability.
2. **Database Scalability**:
    - If the application involves storing user data locally (e.g., scheduling history or preferences), the backend database should be optimized for scalability. Using a scalable database solution, such as **PostgreSQL** or **NoSQL databases** (e.g., **MongoDB**), would allow the system to store and retrieve data efficiently, even as the number of users and events grows.

Possible Optimizations:

Several optimization strategies can be implemented to improve the system's performance and scalability:

1. **Improving Response Time**:
    - **AI Model Optimization**: While the AI engine provides accurate responses, some complex queries can take time to process. One potential optimization is to **optimize the AI model** by reducing its complexity or using more lightweight models for simpler queries. Additionally, **caching common responses** (e.g., frequently requested time slots) could reduce the need for repeated reasoning, speeding up response times.
    - **Parallel Processing**: For complex requests that involve multiple users or extensive calendar data, implementing **parallel processing** can significantly reduce processing time. This could involve parallelizing AI reasoning tasks and calendar conflict resolution across multiple threads or services.

2. **Scaling the Backend**:
    - To handle larger volumes of concurrent requests, consider **auto-scaling** the backend infrastructure. Using **cloud platforms** (e.g., AWS, Google Cloud) with auto-scaling capabilities would ensure that the system dynamically adjusts the number of servers based on demand, maintaining consistent performance during peak usage.
    - **Database Sharding**: If the system stores a large amount of user data, **database sharding** can help distribute data across multiple database instances, improving query performance and scalability.
3. **Rate Limiting and Caching for External APIs**:
    - To avoid hitting rate limits with external APIs (e.g., Twilio, Google Calendar), **rate-limiting** and **caching** should be implemented. Caching calendar data and frequently accessed information in-memory (e.g., using **Redis**) can reduce the number of external API calls, improving both performance and scalability.
    - Additionally, **API request batching** can be used to combine multiple requests into a single call when possible, reducing overhead and improving response times.
4. **Optimizing AI Processing**:
    - The reasoning process for scheduling conflicts can be time-consuming, especially when there are many participants or complex criteria. To optimize this:
        - **Precompute availability data**: If the system frequently checks availability for the same time slots or users, it could precompute availability during low-traffic periods and cache this data for faster access.
        - **Use more efficient AI algorithms**: Exploring faster and more efficient reasoning algorithms could improve the system's ability to handle multiple requests concurrently while maintaining accuracy.
5. **User Experience Enhancements**:
    - **Asynchronous Notifications**: Sending email notifications or WhatsApp confirmations asynchronously, using background tasks, will help reduce the response time for critical user actions like scheduling or rescheduling events.

- **Load Testing**: Regular load testing can identify potential performance bottlenecks in the system. Using tools like **Apache JMeter** or **Locust** can simulate high traffic and help optimize parts of the system that are most affected by heavy usage.

8. Conclusion

Summary of Achievements:

This project successfully built an **AI-driven WhatsApp agent** that integrates with **Google Calendar** and **email services** to provide seamless scheduling and calendar management. The agent allows users to interact with their schedules via WhatsApp, enabling them to schedule, reschedule, and check for conflicts in their calendar. By leveraging **Twilio API** for WhatsApp communication, **Google Gemini models** for AI-driven reasoning, and **Google Calendar API** for event management, the project achieved the core objectives of automating and simplifying scheduling tasks.

Key accomplishments include:

- **WhatsApp Integration**: Establishing reliable communication between the user and the AI agent using Twilio, making the scheduling process user-friendly and efficient.
- **AI-Powered Reasoning**: Utilizing **Google Gemini models** to process natural language inputs, handle scheduling conflicts, and suggest optimal meeting times, making the system highly responsive and intelligent.
- **Calendar Integration**: Seamlessly integrating with Google Calendar to manage user events, check for availability, and schedule new appointments.
- **Email Notifications**: Implementing an email notification system to keep all relevant participants informed about scheduled meetings, ensuring transparency and clarity in communication.

Overall, the project successfully developed an intelligent assistant that can operate within a widely-used messaging platform, providing users with an efficient and intuitive way to manage their schedules.

Lessons Learned:

Throughout the development of this project, several challenges were encountered, but they also provided valuable lessons:

1. **Integration Complexity**: One of the main challenges was ensuring smooth communication between various external services like **Twilio**, **Google Calendar**, and the AI models. Integrating these services required careful attention to API rate limits, authentication, and data handling. This experience emphasized the importance of thorough documentation and understanding of external APIs.
2. **Message Parsing and Ambiguities**: Handling ambiguous user inputs proved to be a challenging aspect of the project. The AI needed to correctly interpret vague or incomplete requests, which required fine-tuning the NLP models and improving fallback mechanisms for uncertain scenarios. This challenge highlighted the need for continuous training and improvements to the AI models to handle diverse user inputs effectively.
3. **Real-Time Performance**: Achieving low-latency responses while handling complex reasoning tasks (e.g., scheduling conflicts, proposing alternate times) required optimizing both the AI models and the backend system. This taught the importance of performance tuning and scalability considerations from the outset to ensure the system could handle real-time interactions efficiently.
4. **AI Accuracy and Reasoning**: While the AI successfully handled straightforward tasks, ensuring accurate and contextually aware reasoning for complex scheduling scenarios (e.g., managing multiple participants' calendars) required iterative improvements. This experience underscored the importance of continuous model training and performance evaluation.

9. Future Work:

While the project has achieved its primary objectives, there are several opportunities for improvement and extension:

1. **Expansion of AI Models**:

- The current AI models primarily handle basic scheduling tasks and conflict resolution. Future iterations could integrate more sophisticated models for better **natural language understanding** (NLU) and **contextual reasoning**. For example, adding **multi-turn conversation capabilities** could allow the agent to hold more fluid and natural conversations with the user, particularly for complex scheduling scenarios.

2. **Integration with Additional Services**:
   - Future versions of the AI agent could integrate with more external services, such as **Zoom** for video call scheduling or **Slack** for team coordination. This would expand the agent's utility beyond personal scheduling, making it more versatile for business and team environments.
   - Integration with other calendar platforms (e.g., **Outlook Calendar**, **Apple Calendar**) could broaden the agent's applicability to a wider audience.
3. **Scalability and Performance Enhancements**:
   - As the user base grows, the system's scalability will need to be addressed. This could involve implementing **load balancing**, **auto-scaling**, and **database sharding** to ensure that the system can handle high traffic and large amounts of data efficiently.
   - Further performance optimizations could be made to reduce latency, particularly in AI-driven reasoning tasks. This could involve precomputing availability for frequently queried time slots or optimizing the reasoning engine for faster decision-making.
4. **Improved User Interface**:
   - While the project focuses on WhatsApp for user interaction, developing a **web or mobile application** to supplement the existing WhatsApp interface could provide users with more flexibility and additional features, such as a calendar view, notifications, and more detailed event management.
5. **Advanced AI Capabilities**:
   - Exploring more advanced AI capabilities, such as **sentiment analysis** or **personalized scheduling preferences**, could make the agent more intuitive and context-aware, further improving the user experience. For example, the agent could detect the urgency of a meeting based on the user's tone or automatically prioritize certain events based on their past behavior.

10. References

Documentation and Guides:

1. **Twilio. (n.d.).** Twilio API for WhatsApp. *Twilio Documentation*. Retrieved from https://www.twilio.com/docs/whatsapp
   This documentation provides an in-depth guide on integrating WhatsApp messaging with Twilio. It covers setting up the Twilio API, sending and receiving messages, and best practices for WhatsApp communication.
2. **FastAPI. (n.d.).** FastAPI Documentation. *FastAPI*. Retrieved from https://fastapi.tiangolo.com/
   The official FastAPI documentation provides a detailed overview of this high-performance web framework, which powers the backend of the WhatsApp AI agent. It covers setup, routing, validation, and asynchronous capabilities that make FastAPI ideal for this project.
3. **Google Cloud. (n.d.).** Google Calendar API Documentation. *Google Developers*. Retrieved from https://developers.google.com/calendar
   This guide outlines how to integrate with the Google Calendar API to manage events, check availability, and schedule new appointments. It is the primary source for understanding how to interact with Google Calendar programmatically in the WhatsApp AI agent.
4. **Google AI. (n.d.).** Gemini Models Overview. *Google AI*. Retrieved from https://cloud.google.com/ai
   This page provides information about Google's Gemini models, which are used for natural language processing and reasoning tasks in the project. The models allow the AI agent to generate intelligent responses and process user input effectively.
5. **python-dotenv. (n.d.).** python-dotenv Documentation. *python-dotenv*. Retrieved from https://pypi.org/project/python-dotenv/
   The python-dotenv package is used to load environment variables from a .env file. This documentation explains how to securely manage API keys and configuration settings required for the project.
6. **ngrok. (n.d.).** ngrok Documentation. *ngrok*. Retrieved from https://ngrok.com/docs
   This guide explains how to use ngrok to create secure tunnels to localhost, which is essential for exposing the FastAPI server to the internet during development for testing and integration with Twilio.

10. Acknowledgments:

We would like to express our sincere gratitude to **Mr. GAMOUH Hamza** for his invaluable support throughout the project. His guidance during the advising sessions, insightful feedback, and constant availability have been crucial in shaping the direction and success of the project. His expertise and willingness to assist at every stage of development were truly appreciated.

We would also like to extend our heartfelt thanks to **Mr. HAFIDI Hakim**, the project supervisor, for his continuous support and leadership. His strategic direction, encouragement, and oversight were essential in ensuring the timely completion of the project. We are deeply grateful for his dedication and commitment to the success of this work.