WhatsApp-based AI Assistant for Google Workspace: System Architecture and Technical Specifications

1. Introduction

This document outlines the system architecture and technical specifications for an AI assistant that integrates with WhatsApp to control Google Workspace services through Natural Language Processing (NLP). Inspired by the Slack-based AI assistant demonstrated in the provided YouTube video and leveraging insights from the OpenClaw project, this design focuses on a robust, scalable, and secure solution.

2. High-Level Architecture

The AI assistant will operate as an intermediary between the user on WhatsApp and the Google Workspace APIs. The core components include a WhatsApp Interface, an NLP Engine, an AI Agent Core, and a Google Workspace Integrator, all supported by a Memory/Context Management system.

mermaid

3. Component Breakdown

3.1 WhatsApp Interface

This component is responsible for receiving messages from users on WhatsApp and sending responses back. Based on the OpenClaw research, the `wacli` method is recommended for personal use due to its simplicity and cost-effectiveness.

Technical Specifications:

- **Method**: `wacli` (WhatsApp Web protocol via Baileys library).
- **Setup**: QR code pairing for linking a personal or business WhatsApp account.

- **Features**: Support for text messages, group chats, and potentially media (images, audio).
- **Security**: Leverages WhatsApp's end-to-end encryption; local processing of messages to maintain privacy.

3.2 NLP Engine

The NLP Engine interprets user commands, extracting intent and relevant entities. This is the core of natural language understanding.

Technical Specifications:

- **Technology**: Large Language Model (LLM) for intent recognition, entity extraction, and natural language generation.
  - **Recommendation**: Anthropic Claude (e.g., Claude 3.5 Sonnet) for its strong performance in long-context understanding and prompt injection resistance, as suggested by OpenClaw documentation.
- **Functionality**: Transforms raw text messages into structured commands or queries for the AI Agent Core.
- **Output**: Structured JSON objects containing `intent` (e.g., `create_event`, `send_email`) and `entities` (e.g., `recipient`, `date`, `subject`, `body`).

3.3 AI Agent Core

The AI Agent Core acts as the orchestrator, receiving parsed intent and entities from the NLP Engine, determining the appropriate action (tool call), executing it, and formulating a coherent response.

Technical Specifications:

- **Framework**: Agentic framework (e.g., inspired by OpenClaw's agent runtime) to manage task execution, tool selection, and response generation.
- **Tool Management**: Manages a set of predefined tools (skills) that correspond to Google Workspace functionalities.
- **Decision Making**: Uses the LLM to decide which tool to invoke based on the user's intent and available context.
- **Error Handling**: Implements robust error handling and fallback mechanisms for failed tool calls or ambiguous requests.

3.4 Google Workspace Integrator

This component is responsible for making authenticated calls to the Google Workspace APIs based on the instructions from the AI Agent Core.

Technical Specifications:

- **APIs**:
  - **Gmail API**: For email operations (send, read, search).
  - **Google Calendar API**: For event management (create, read, update, delete).
  - **Google Drive API**: For file operations (search, create, share).
  - **Google Sheets API**: For spreadsheet data manipulation.
- **Authentication**: OAuth 2.0 for secure access to user's Google Workspace data.
  - **Process**: Requires a Google Cloud Project, enabled APIs, configured OAuth Consent Screen, and OAuth 2.0 Client IDs (Desktop application type).
  - **Token Management**: Secure storage and refresh of access tokens.
- **Client Libraries**: Utilizes official Google API client libraries (e.g., Python client library) for reliable interaction.

3.5 Memory/Context Management

This system stores and retrieves conversational history, user preferences, and other relevant data to enable a personalized and continuous interaction.

Technical Specifications:

- **Types of Memory (inspired by OpenClaw)**:
  - **Short-term Memory**: Stores recent conversational turns for immediate context.
  - **Working Memory**: Holds task-specific information during an ongoing interaction.
  - **Long-term Memory**: Stores user preferences, frequently used contacts, and historical data for personalization.
- **Storage**: A persistent data store (e.g., a simple file-based database or a lightweight NoSQL database) for long-term memory.
- **Retrieval-Augmented Generation (RAG)**: Potentially integrates RAG techniques to retrieve relevant information from memory to augment LLM prompts, improving accuracy and context awareness.

4. Technical Stack and Deployment Considerations

4.1 Programming Languages and Frameworks

- **Primary Language**: TypeScript/Node.js (consistent with OpenClaw's ecosystem).
- **Python**: Potentially used for specific NLP tasks or Google API interactions if more mature libraries are available.

4.2 Deployment Environment

- **Local/Self-hosted**: Suitable for personal use, mirroring OpenClaw's local-first approach.
- **Cloud VM**: For more robust and always-on availability (e.g., a cloud Ubuntu VM with sufficient resources for LLM processing).

4.3 Security Considerations

- **OAuth 2.0**: Strict adherence to Google's OAuth 2.0 best practices.
- **WhatsApp Security**: Rely on `wacli` 's inherent security and WhatsApp's end-to-end encryption.
- **Access Control**: Implement user-specific access control for Google Workspace functionalities.
- **Data Privacy**: Ensure sensitive user data is handled and stored securely, adhering to privacy principles.

5. Implementation Plan (High-Level)

1. **Setup OpenClaw Environment**: Install OpenClaw and its dependencies.
2. **WhatsApp Integration**: Configure `wacli` for WhatsApp connectivity.
3. **Google Cloud Project Setup**: Create a Google Cloud Project, enable necessary APIs, and configure OAuth 2.0 credentials.
4. **Develop Google Workspace Skills**: Create OpenClaw-compatible skills (tools) for Gmail, Calendar, Drive, and Sheets.
5. **NLP Integration**: Integrate the chosen LLM for intent recognition and response generation.
6. **Memory System**: Implement a basic memory system for context management.
7. **Testing**: Thoroughly test each component and the end-to-end workflow.

6. Conclusion

This design provides a comprehensive blueprint for building a WhatsApp-based AI assistant that leverages NLP to control Google Workspace. By adopting a modular architecture and utilizing established technologies like OpenClaw and Google APIs, the assistant can offer a

powerful and intuitive way for users to manage their productivity through natural language commands.

References

[1] YouTube Video: Build Mini ClawdBot from Scratch | Agents | MCP | Memory | Context Engineering -
[2] OpenClaw GitHub Repository:
[3] OpenClaw WhatsApp Integration Guide:
[4] Google Developers: Using OAuth 2.0 to Access Google APIs -
[5] Google Cloud Console: