Comprehensive Master Technical Report: An Intelligent Microsoft Teams Planner Management System

Report ID: TR-2025-1006-MTP
Publication Date: 2025-10-06
Status: Final
Classification: Technical Documentation for Internal Use

1.0 Executive Summary

This report provides a comprehensive technical blueprint for the design, development, and deployment of an intelligent management system for Microsoft Teams Planner, operated through a natural language interface. The primary objective of this project is to streamline task and project management workflows by enabling users to interact with Planner using conversational commands directly within their collaborative environment. The target audience for this document includes the development teams, architects, and technical decision-makers responsible for implementing the system.

The core finding of our research is that building a sophisticated, responsive, and intelligent Planner management system is highly feasible. This conclusion is based on the robust capabilities of the Microsoft Graph API, which provides comprehensive programmatic access to Planner data, and the maturity of the open-source conversational AI ecosystem. The Graph API exposes a detailed data model for plans, buckets, tasks, and their associated properties, all manageable through standard CRUD operations and secured by the industry-standard OAuth 2.0 protocol. However, any implementation must be architected to navigate critical API limitations, namely stringent rate-limiting policies and a lack of real-time webhook notifications, which necessitates a polling-based data synchronization strategy using delta queries.

Our evaluation of the open-source landscape reveals that no turn-key solution exists that can be directly forked for this purpose. Existing projects are primarily focused on data extraction or command-line utilities rather than interactive conversational agents. Therefore, the definitive recommendation is to pursue a **custom-built solution**. This approach involves integrating a curated set of best-in-class, open-source components into a modular, microservices-inspired architecture. This strategy offers maximum flexibility, avoids vendor lock-in, and provides a clear path for future enhancements.

The proposed system architecture centers on **OpenWebUI** as the conversational AI hub, which manages user interactions and orchestrates calls to various Large Language Models (LLMs). A custom backend tool, the **Planner MCP Server**, will be developed to encapsulate all business logic for interacting with the Microsoft Graph API. This server will adhere to the **Model Context Protocol (MCP)**, an emerging open standard that ensures the tool is reusable and interoperable. An intermediary proxy, **MCPO**, will translate between OpenWebUI’s native OpenAPI tool format and the MCP standard. The entire system will be containerized and deployed using **Docker Compose**, ensuring a portable, reproducible, and isolated environment. This report provides the detailed specifications, code templates, and phased implementation guidance necessary to successfully execute this project.

2.0 Complete System Architecture

The architecture of the Intelligent Planner Management System is designed as a modular, multi-component platform that ensures a clear separation of concerns, enhances maintainability, and promotes scalability. Each component is containerized using Docker and communicates over a private network, creating a cohesive yet decoupled system. The design prioritizes the use of open standards and best-of-breed open-source technologies to build a robust and future-proof solution.

2.1 Component Descriptions

The system is composed of five primary components, each with a distinct responsibility:

1. **Microsoft Teams Bot (User Interface):** This is the user-facing entry point. Built using the `microsoft/teams-ai` library, this bot is responsible for capturing user messages within the Teams client, managing the Teams-specific authentication context, and rendering the AI's responses in a rich, interactive format. It acts as a thin client, forwarding user prompts to the conversational hub and relaying responses back.

2. **OpenWebUI (Conversational AI Hub):** This service is the central nervous system of the application. It is a feature-rich, self-hosted web interface that manages the entire lifecycle of a conversation. Its key responsibilities include maintaining conversation history, orchestrating interactions with one or more configured LLMs (both local via Ollama and cloud-based), and invoking external tools based on the LLM's analysis of the user's intent.

3. **MCPO - MCP-to-OpenAPI Proxy (Integration Layer):** This lightweight service acts as a critical translation bridge. OpenWebUI's plugin architecture is based on the OpenAPI specification for defining tools. To promote interoperability, our backend tool is built to the Model Context Protocol (MCP) standard. The MCPO proxy introspects the capabilities of the Planner MCP Server, dynamically generates a corresponding OpenAPI specification, and translates incoming OpenAPI requests from OpenWebUI into MCP requests for the backend, and vice-versa for responses.

4. **Custom Planner MCP Server (Backend Tool Server):** This is the core of the system's business logic. It is a custom-built Python application that exposes a set of functions (e.g., `create_task`, `list_plans`) compliant with the MCP standard. This server encapsulates all interactions with the Microsoft Graph API, handling OAuth 2.0 token management, constructing API requests, parsing responses, and implementing resilience patterns like retry logic for throttling and ETag management for concurrency.

5. **Microsoft Graph API (External API Layer):** This is the definitive endpoint for all Microsoft Planner data. The Planner MCP Server communicates directly with this API to perform all create, read, update, and delete (CRUD) operations on plans, buckets, tasks, and their associated details.

2.2 End-to-End Data Flow

The data flow for a typical user request follows a clear, sequential path through the architecture:

1. **User Input:** The user types a natural language command into the Microsoft Teams bot (e.g., "Create a task to 'Finalize Q4 budget' in the 'Finance' plan and assign it to me").
2. **Forward to Hub:** The Teams bot forwards the raw text message to the OpenWebUI backend API.
3. **LLM Orchestration:** OpenWebUI appends the message to the current conversation history and sends the complete context to the selected LLM. The prompt instructs the LLM that it has access to a set of Planner tools.

4. **Function Calling:** The LLM analyzes the prompt, recognizes the user's intent to create a task, and identifies the necessary parameters (title, plan name, assignee). It then formulates a structured function call, such as `planner.create_task(title='Finalize Q4 budget', plan_name='Finance', assignee='current_user')`, and returns this to OpenWebUI.
5. **Tool Invocation:** OpenWebUI receives the function call instruction and triggers the registered Planner tool. It sends an HTTP request matching the OpenAPI specification to the MCPO proxy server.
6. **Protocol Translation:** The MCPO proxy receives the OpenAPI request, translates it into a standardized MCP request, and forwards it to the Planner MCP Server.
7. **API Interaction:** The Planner MCP Server receives the MCP request. It resolves the `plan_name` to a `planId`, authenticates with the Microsoft Graph API using the user's OAuth token, and sends a `POST` request to the `/v1.0/planner/tasks` endpoint with the appropriate JSON payload.
8. **Response Propagation:** The Microsoft Graph API creates the task and returns a success response. This response travels back through the chain: the MCP server sends an MCP success message to the MCPO proxy, which translates it into an HTTP success response for OpenWebUI.
9. **Natural Language Generation:** OpenWebUI receives the successful tool execution result, appends it to the conversation history, and sends a final prompt to the LLM asking it to formulate a user-friendly confirmation message.
10. **Final Output:** The LLM generates a response like, "Okay, I've created the task 'Finalize Q4 budget' in the 'Finance' plan and assigned it to you." OpenWebUI sends this text back to the Teams bot, which displays it to the user.

2.3 NLP Pipeline and Integration

The Natural Language Processing (NLP) pipeline is almost entirely managed by the Large Language Model's native capabilities, specifically its proficiency in **tool use** or **function calling**. The process does not rely on traditional NLP components like separate intent classifiers or named-entity recognition models. Instead, the high-level process is as follows:

1. **Prompt Engineering:** The system prompt provided to the LLM by OpenWebUI is engineered to include a description of the available tools (functions) sourced from the MCPO's OpenAPI specification. This description details each function's name, purpose, parameters, and their data types.
2. **In-Context Learning:** The LLM uses this tool specification as part of its context. When it receives a user's prompt, it determines whether the request can be fulfilled with a text response or if it requires an external action.
3. **Intent and Entity Mapping:** If an action is required, the LLM performs implicit intent recognition (e.g., the user wants to create something) and entity extraction (e.g., the thing to create is a "task" with a specific "title"). It then maps this understanding to the most appropriate function available in the tool specification.
4. **Structured Output Generation:** The LLM's final step is to generate a structured JSON object that represents the function call, including the function name and a dictionary of arguments. This structured output is the key to the system's ability to translate unstructured natural language into deterministic, machine-executable actions.

This LLM-centric approach simplifies the architecture significantly, leveraging the advanced reasoning capabilities of modern models to handle the complexities of language understanding.

3.0 Detailed Feasibility Analysis

The project's feasibility is high, contingent upon a clear understanding and strategic mitigation of the inherent limitations and risks associated with the core technologies. The Microsoft Graph API provides a sufficiently powerful foundation, but its constraints dictate key architectural decisions.

3.1 Capabilities of the Microsoft Graph API

The Graph API for Planner offers a comprehensive set of functionalities that align well with the project's objectives. It provides full programmatic control over the primary Planner entities:

- **Plans:** Can be created (within a Microsoft 365 Group), read, updated (e.g., changing the title), and deleted.
- **Buckets:** Can be created within a plan, listed, updated (name and order), and deleted.
- **Tasks:** Can be created with a title, plan ID, and bucket ID. They can be updated to modify properties like assignments, due dates, completion status, priority, and more. Tasks can also be deleted.
- **Task Details:** Richer information, including descriptions, checklists (with up to 50 items per task), and external references (attachments), can be managed through a separate but linked `plannerTaskDetails` object.
- **Assignments:** Tasks can be assigned to up to 20 users who are members of the parent group.
- **Labels:** The six colored labels (applied categories) can be applied to or removed from tasks.

This granular control enables the translation of a wide range of natural language commands into specific, actionable API calls. Furthermore, the API's support for JSON batching is a critical capability, allowing the system to combine multiple operations (e.g., create a bucket, then create a task within it) into a single HTTP request, which significantly improves performance and reduces the risk of throttling.

3.2 Limitations and Constraints

Despite its strengths, the API presents several significant constraints that must be addressed:

- **No Real-Time Notifications:** The most critical limitation is that Planner resources are not supported by the Microsoft Graph change notifications (webhooks) system. This means the application cannot subscribe to real-time updates. To reflect changes made by other users or in other applications, the system must implement a polling mechanism. The recommended approach is to use **delta queries**, which efficiently retrieve only the changes since the last poll. This introduces an inherent latency, the duration of which is determined by the polling interval.
- **Strict Throttling:** The Graph API enforces service-specific rate limits to ensure service stability. Exceeding these limits will result in HTTP `429 Too Many Requests` errors. The application's backend must be designed with a resilient, exponential backoff retry strategy that respects the `Retry-After` header provided in the API's response.
- **No Premium Feature Access:** The API currently only provides access to the standard set of Planner features. Advanced functionalities available in premium versions of Planner, such as custom fields or advanced timeline views, are not exposed via the API.
- **Concurrency Management:** The API uses ETags for optimistic concurrency control. Any update or delete operation must include the ETag of the resource version being modified. Failure to do so, or providing a stale ETag, will result in an HTTP `412 Precondition Failed` error. The backend logic must rigorously manage this fetch-and-update workflow.

3.3 Risk Assessment and Mitigation

Risk	Probability	Impact	Mitigation Strategy
API Rate Limiting/ Throttling	High	Medium	Implement an exponential backoff retry mechanism in the Planner MCP Server. Optimize API calls by using `$select` and `$filter` query parameters and leveraging JSON batching for multi-step operations.
Data Synchronization Latency	Certain	Medium	Implement an efficient polling mechanism using Microsoft Graph delta queries. Clearly communicate the potential for latency to the user. The polling interval should be configurable to balance data freshness with API usage.
OAuth 2.0 Complexity	Medium	High	Use a well-supported Microsoft Graph SDK for Python to manage the token acquisition and refresh lifecycle. Securely store refresh tokens. Implement robust error handling for authentication failures.
LLM Hallucination/ Incorrect Function Calls	Medium	Medium	Use a state-of-the-art LLM known for strong function-calling capabilities. Employ rigorous prompt engineering with clear tool descriptions and examples. Implement a validation layer in the MCP server to check

Risk	Probability	Impact	Mitigation Strategy
			parameters before execution.
Changes in Microsoft Graph API	Low	High	Adhere to the stable `v1.0` version of the Graph API. Implement comprehensive integration tests to catch breaking changes early. Monitor the Microsoft 365 Developer Blog for announcements.

3.4 High-Level Project Timeline

A phased implementation approach is recommended to manage complexity and deliver value incrementally.

- **Phase 1: Foundation & Backend (Weeks 1-4):** Focus on setting up the Docker environment and developing the core Planner MCP Server. This includes implementing the Microsoft Graph API integration, OAuth 2.0 authentication, and robust error handling for all primary CRUD operations.
- **Phase 2: Integration & AI Core (Weeks 5-7):** Deploy OpenWebUI and the MCPO proxy. Integrate the Planner MCP Server as a tool within OpenWebUI. Focus on prompt engineering and testing the end-to-end natural language to API call workflow.
- **Phase 3: User Interface & Deployment (Weeks 8-9):** Develop the Microsoft Teams bot using the `teams-ai` library. Integrate the bot with the OpenWebUI backend. Refine the user experience within the Teams environment.
- **Phase 4: Advanced Features & Testing (Weeks 10-12):** Implement advanced features like the RAG pipeline for data augmentation and the document generation service. Conduct thorough end-to-end testing, performance validation, and user acceptance testing (UAT).

This 12-week timeline is an estimate and assumes a dedicated development team with experience in the proposed technologies.

4.0 Technology Stack Recommendations

The selection of the technology stack is guided by principles of robustness, scalability, developer productivity, and a preference for mature, well-supported open-source solutions. The following recommendations provide a comprehensive and cohesive set of tools for building the system.

4.1 Database Technologies

A hybrid database strategy is recommended to handle the diverse data storage and retrieval needs of the application, from structured relational data to high-dimensional vector embeddings and complex graph relationships.

- **Primary Relational & Vector Database: PostgreSQL with pgvector**
  PostgreSQL is recommended as the primary database for storing OpenWebUI's application data, such as user information and conversation history. Its robustness, advanced SQL features, and

strong support for complex data types like JSONB make it superior to alternatives like MySQL for this use case. The key advantage of PostgreSQL is its extensibility. By installing the **pgvector** extension, the same database instance can be used as a high-performance vector store for the Retrieval-Augmented Generation (RAG) pipeline. This integrated approach dramatically simplifies the system architecture, reduces operational overhead, and allows for powerful hybrid queries that combine traditional SQL filtering with semantic vector search in a single operation. While dedicated vector databases like **Qdrant** offer higher performance at massive scale, the simplicity and sufficient performance of pgvector make it the ideal choice for this single-user, containerized system.

- **Graph Database (for Future Evaluation): Neo4j**
  For modeling and querying highly interconnected data, such as complex task dependencies, project workflows, or organizational structures that extend beyond Planner's native capabilities, a graph database is the optimal tool. **Neo4j** is the recommended choice for future evaluation and potential integration. Its native graph storage, intuitive Cypher query language, and mature ecosystem make it a powerful tool for advanced relational analysis. Queries like finding the critical path in a project or identifying team members who are central to communication are trivial in Neo4j but computationally expensive in a relational database. While not required for the initial implementation, integrating Neo4j for a "GraphRAG" capability represents a significant potential enhancement on the project roadmap.

4.2 RAG Framework and Orchestration

- **Orchestration Framework: LangChain**
  The core of the AI logic, particularly for the RAG pipeline and agentic workflows, should be built using **LangChain**. While alternatives like LlamaIndex are highly optimized for pure retrieval tasks, LangChain's strength lies in its general-purpose orchestration capabilities. Its modular architecture, based on "chains" and "agents," provides the flexibility needed to build complex, multi-step workflows that combine LLM calls, data retrieval, and external tool use (like calling the Planner API). This makes it the most suitable framework for a system that needs to both answer questions and perform actions.

4.3 Document Ingestion and Processing

- **Document Processing: Unstructured.io**
  To populate the RAG system's knowledge base, documents must be processed into a clean, structured format suitable for embedding. **Unstructured.io** is the recommended tool for this purpose. Unlike general-purpose parsers like Apache Tika, Unstructured.io is specifically designed for LLM workflows. It excels at not just extracting text but also understanding and preserving the logical structure of a document, partitioning it into meaningful elements like titles, paragraphs, lists, and tables. This high-quality, structured output leads to better embeddings and more accurate retrieval.

- **Web Crawling: Playwright**
  To augment the knowledge base with external web content, **Playwright** is the recommended tool. As a modern browser automation framework, it can reliably render and extract content from JavaScript-heavy websites and single-page applications, a common failure point for simpler libraries like BeautifulSoup. For developers seeking a faster, AI-driven approach, **Firecrawl** offers a compelling API-based alternative that can return clean, structured data from a URL with minimal code.

4.4 Document Generation Libraries

- **PDF Generation: WeasyPrint**
  For generating professional-quality PDF reports from Planner data, **WeasyPrint** is the recommended library. It converts standard HTML and CSS into PDFs, providing an intuitive and powerful workflow for developers familiar with web technologies. This allows for the creation of visually rich, templated documents with relative ease.

- **Microsoft Office Formats: python-docx and python-pptx**
  To generate native, editable Microsoft Office documents, the combination of **python-docx** (for Word) and **python-pptx** (for PowerPoint) is the standard and most effective solution. These libraries provide direct, programmatic control over the creation and manipulation of `.docx` and `.pptx` files, enabling the system to fulfill requests like “Generate a Word summary of all my overdue tasks.”

5.0 Open-Source Solution Evaluation

A thorough investigation of the open-source landscape was conducted to determine if an existing project could be forked or adapted to accelerate development. The primary finding is that no turn-key, open-source conversational AI system specifically designed for Microsoft Planner currently exists. The available repositories are generally auxiliary tools rather than complete, interactive solutions.

5.1 Analysis of Existing Repositories

The search for relevant projects on platforms like GitHub revealed several categories of tools, none of which fit the project’s core requirement for a natural language interface:

- **Command-Line Interfaces (CLIs):** Projects like `pnp/cli-microsoft365` provide powerful command-line tools for scripting and bulk management of Microsoft 365 resources, including Planner. While useful for administrators, they lack a conversational or NLP component.
- **Data Extraction & Reporting Scripts:** Numerous repositories contain PowerShell or Python scripts designed to extract Planner data for reporting purposes, often for integration with tools like Power BI. These are one-way data pipelines, not interactive systems.
- **Specific Automation Workflows:** Some projects demonstrate specific integrations, such as creating a Planner task from a GitHub issue. These are point solutions for narrow use cases.

While these tools are not suitable for forking, their source code can serve as valuable references for understanding how to interact with the Microsoft Graph API for specific tasks.

5.2 Fork-vs-Build Analysis

Given the absence of a suitable base application, the strategic choice is between forking a general-purpose chatbot UI and building a custom solution by integrating specialized components.

- **Forking a General-Purpose UI:** Forking a project like `mckaywrigley/chatbot-ui` or `vercel/ai-chatbot` would provide a polished, pre-built user interface. This would accelerate front-end development significantly. However, the core challenge of the project lies in the backend integration with the Planner API and the orchestration of complex, tool-using AI agents. These general-purpose UIs do not have the sophisticated backend architecture required, meaning the majority of the engineering effort would still be in custom development. Furthermore, adapting their existing architecture to our specific multi-component design (OpenWebUI, MCP, etc.) could be more complex than building from a clean slate.

- **Custom Build with Open-Source Components:** This is the highly recommended approach. Instead of adopting a monolithic application, this strategy involves assembling a system from best-in-class, specialized open-source projects. We use **OpenWebUI** for the conversational hub, **Ollama** for local LLM serving, **LangChain** for AI orchestration, and a custom Python application for the backend tool. This approach offers superior flexibility, allowing the team to choose the best tool for each job. It results in a more modular, maintainable, and scalable architecture. While it requires more initial setup and integration work, it provides complete control over the system's design and avoids the constraints of a pre-existing codebase.

In conclusion, the most efficient and strategically sound path is a **custom build**. This approach leverages the power of the open-source ecosystem not by forking a single project, but by composing a superior solution from its strongest individual components.

6.0 Six-Phase Implementation Guide

This implementation guide outlines a structured, six-phase approach to developing the Intelligent Planner Management System. This phased plan is designed to manage complexity, mitigate risks by tackling high-priority components first, and allow for iterative development and testing.

Phase 1: Environment Setup and Foundational Configuration

- **Objective:** Establish a fully containerized, reproducible development environment.
- **Key Activities:**
  1. **Develop the Docker Compose file:** Create the initial `docker-compose.yml` defining the core services: PostgreSQL, OpenWebUI, and Ollama.
  2. **Configure Base Services:** Set up the necessary volumes for data persistence and the private network for inter-service communication.
  3. **Initial Launch and Validation:** Run `docker compose up` to build and start the containers. Verify that OpenWebUI is accessible, can connect to the PostgreSQL database, and can communicate with the Ollama service.
  4. **Version Control Setup:** Initialize a Git repository with a proper `.gitignore` file to exclude secrets (`.env`) and local build artifacts.

Phase 2: Planner MCP Server Development

- **Objective:** Build the core backend service that encapsulates all Microsoft Graph API interactions.
- **Key Activities:**
  1. **Microsoft Entra ID App Registration:** Register the application in Microsoft Entra ID to obtain a Client ID and configure the required `Tasks.ReadWrite` delegated permissions.
  2. **Implement OAuth 2.0 Flow:** Develop the logic within the Python MCP server to handle the full OAuth 2.0 Authorization Code Flow for user authentication and token management. Use the Microsoft Graph SDK for Python to simplify this process.
  3. **Develop Core CRUD Functions:** Implement the primary tool functions for managing plans, buckets, and tasks (e.g., `create_task`, `get_user_tasks`, `update_task`).
  4. **Implement Resilience Patterns:** Integrate robust error handling for API throttling (exponential backoff) and concurrency (ETag management).
  5. **Containerize the Server:** Write a `Dockerfile` for the MCP server and add its service definition to the `docker-compose.yml` file.

Phase 3: RAG Pipeline Implementation

- **Objective:** Build the data ingestion and retrieval pipeline to augment the LLM's knowledge.

- **Key Activities:**
  1. **Extend PostgreSQL:** Install the `pgvector` extension in the PostgreSQL container to enable vector storage and search.
  2. **Develop Data Ingestion Service:** Create a separate service or a set of scripts that use **Unstructured.io** to process documents (PDFs, DOCX) and **Playwright** to scrape web pages.
  3. **Implement Embedding and Storage:** Use a sentence-transformer model to generate embeddings for the processed text chunks and store them in the `pgvector` table alongside their source metadata.
  4. **Build Retrieval Chain:** Use **LangChain** to create a retrieval chain that takes a user query, generates an embedding, queries `pgvector` for relevant documents, and formats the results as context.

Phase 4: Natural Language Understanding and Tool Integration

- **Objective:** Connect the conversational front-end to the backend tool server and enable the LLM to perform actions.
- **Key Activities:**
  1. **Deploy MCPO Proxy:** Add the MCPO service to the Docker Compose file, configuring it to connect to the Planner MCP Server.
  2. **Register Tool in OpenWebUI:** Access the OpenWebUI admin panel and register the Planner tool by providing the OpenAPI specification URL exposed by the MCPO proxy.
  3. **Prompt Engineering:** Develop and refine the system prompt used in OpenWebUI to clearly instruct the LLM on the capabilities and proper usage of the available Planner tools.
  4. **End-to-End Testing:** Conduct extensive testing of the full conversational flow, from a natural language command in OpenWebUI to a successful API call by the MCP server and back.

Phase 5: Document Generation Service

- **Objective:** Create a service capable of generating reports from Planner data.
- **Key Activities:**
  1. **Develop Generation Logic:** Create a new Python service or extend the MCP server with functions for document generation. Use **WeasyPrint** for PDFs and **python-docx/pptx** for Office files.
  2. **Create Templates:** Design HTML/CSS templates (for WeasyPrint) and base Word/PowerPoint files to be used for populating data.
  3. **Expose as a Tool:** Add the document generation functions (e.g., `generate_task_summary_pdf`) to the MCP server's toolset so they can be invoked by the LLM.

Phase 6: Microsoft Teams Integration and Final Testing

- **Objective:** Deploy the user-facing bot and conduct final system validation.
- **Key Activities:**
  1. **Develop Teams Bot:** Use the `microsoft/teams-ai` library to build the bot that will act as the client for the OpenWebUI backend.
  2. **Deploy and Register Bot:** Deploy the bot to a hosting service (e.g., Azure App Service) and register it within the Microsoft Teams platform.
  3. **User Acceptance Testing (UAT):** Perform comprehensive UAT with a pilot group to gather feedback on functionality, usability, and performance.
  4. **Final Deployment:** Based on UAT feedback, make final adjustments and prepare the system for a wider rollout.

7.0 Natural Language to API Action Mapping

The core of the system's intelligence lies in its ability to translate unstructured, conversational user requests into precise, structured API calls. This process is not handled by traditional, rule-based NLP systems but is instead delegated to the advanced reasoning and function-calling capabilities of a large language model.

7.1 Intent Recognition

Intent recognition is the process of identifying the user's primary goal or objective from their message. In this architecture, the LLM performs this task implicitly. When presented with a user prompt like "Can you make a new task for me?", the LLM, guided by the descriptions of the tools it has been provided, recognizes the user's intent as `create_task`. It matches the user's language ("make a new task") to the purpose described for the `planner.create_task` function in its system prompt. This process is highly flexible and can understand a wide variety of phrasings for the same underlying intent.

7.2 Entity Extraction

Once the intent is recognized, the LLM must extract the specific pieces of information—the entities—required to execute that intent. These entities correspond to the parameters of the target function. For a request like "Create a task to 'Review Q4 marketing deck' in the 'Campaigns' plan due this Friday," the LLM extracts the following entities:

- **Task Title:** "Review Q4 marketing deck"
- **Plan Name:** "Campaigns"
- **Due Date:** "this Friday"

The LLM is capable of resolving relative terms like "this Friday" into a concrete date (e.g., `2025-10-10`). The extracted entities are then used to populate the arguments of the function call.

7.3 Disambiguation and Error Handling

A robust conversational system must gracefully handle ambiguity and missing information. The LLM-driven approach manages this through conversational clarification.

- **Missing Information:** If a user says, "Create a task to review the deck," the LLM will recognize that the `planner.create_task` function requires a `plan_name` parameter, which is missing. Instead of failing, the LLM will generate a clarifying question as its response, such as, "Certainly. In which plan should I create the task 'review the deck'?" The system then waits for the user's response, and once the information is provided, it re-attempts the function call with the complete set of parameters.

- **Ambiguous Information:** If a user has access to two plans named "Marketing," and they say "Add a task to the Marketing plan," the backend MCP server logic can detect this ambiguity after looking up the plan name. The server would return a specific error message to the LLM, which would then formulate a question to the user, such as, "I found two plans named 'Marketing.' Which one do you mean: the 'Q3 Marketing Campaigns' plan or the '2025 Marketing Budget' plan?"

This interactive, conversational approach to error handling creates a much more natural and user-friendly experience compared to rigid, form-based interfaces. The final mapping is the generation of a structured JSON object by the LLM, which is then sent through the system to be executed by the Planner MCP Server.

8.0 Detailed Workflow Examples and Use Cases

The following five use cases illustrate the system's capabilities through detailed conversational flows, demonstrating how natural language commands are translated into actions.

Use Case 1: Creating a New Task with Full Details

- **Objective:** A user wants to create a new task, specifying the title, plan, bucket, assignee, and due date in a single command.
- **Conversation Flow:**
  - **User:** "Create a new task named 'Draft project proposal' in the 'Project Alpha' plan, put it in the 'To Do' bucket, assign it to sarah@example.com, and make it due next Monday."
  - **LLM Analysis:**
    - **Intent:** `planner.create_task`
    - **Entities:**
      - `title`: "Draft project proposal"
      - `plan_name`: "Project Alpha"
      - `bucket_name`: "To Do"
      - `assignee`: "sarah@example.com"
      - `due_date`: "2025-10-13" (resolves "next Monday")
  - **Backend Action:** The Planner MCP Server resolves the plan, bucket, and user names to their respective IDs and makes a `POST` request to `/v1.0/planner/tasks` with all the details in the payload.
  - **System Response:** "Done. I've created the task 'Draft project proposal' in the 'To Do' bucket of the 'Project Alpha' plan, assigned it to Sarah, and set the due date to October 13, 2025."

Use Case 2: Listing and Filtering Tasks

- **Objective:** A user wants to see all their tasks that are due soon.
- **Conversation Flow:**
  - **User:** "Show me all of my tasks that are due this week."
  - **LLM Analysis:**
    - **Intent:** `planner.get_user_tasks`
    - **Entities:**
      - `assignee`: "me" (or `current_user`)
      - `due_date_filter`: "this_week" (from today, 2025-10-06, to 2025-10-12)
  - **Backend Action:** The MCP server calls the `/v1.0/me/planner/tasks` endpoint. It then applies a client-side filter to the results to only include tasks with a due date within the calculated range.
  - **System Response:** "You have 3 tasks due this week:
    1. **Finalize Q4 budget** (Due: 2025-10-08) - Plan: Finance
    2. **Review marketing deck** (Due: 2025-10-10) - Plan: Campaigns
    3. **Submit weekly report** (Due: 2025-10-10) - Plan: Operations"

Use Case 3: Updating an Existing Task

- **Objective:** A user wants to mark a task as complete using conversational language.
- **Conversation Flow:**
  - **User:** "I've finished the task to submit the weekly report."

- **LLM Analysis:** The LLM needs to resolve which task the user is referring to. It might first call `planner.get_user_tasks(title_contains='submit weekly report')` to find the task.
- **Backend Action (Step 1 - Find):** The MCP server finds a unique, active task matching the description. It returns the task's ID and ETag.
- **LLM Analysis (Step 2 - Update):**
  - **Intent:** `planner.update_task`
  - **Entities:**
    - `task_id`: (ID returned from the search)
    - `percentComplete`: 100
- **Backend Action (Step 2 - Update):** The MCP server makes a `PATCH` request to `/v1.0/planner/tasks/{task-id}` with the task's ETag in the `If-Match` header and a payload of `{"percentComplete": 100}`.
- **System Response:** "Great! I've marked the task 'Submit weekly report' as complete."

Use Case 4: Complex Multi-Step Command (Batching)

- **Objective:** A user wants to create a new organizational structure (a bucket) and populate it with tasks in a single command.
- **Conversation Flow:**
  - **User:** "In the 'Website Redesign' plan, create a new bucket called 'Testing' and add two tasks to it: 'Perform cross-browser testing' and 'Run performance audit'."
  - **LLM Analysis:** The LLM decomposes this into three distinct actions: create a bucket, then create two tasks, noting the dependency.
  - **Backend Action:** The Planner MCP Server constructs a single JSON **batch request** to the `/$batch` endpoint.
    1. **Request 1:** `POST /planner/plans/{plan-id}/buckets` with `name: "Testing"`. This request is given an `id` of "1".
    2. **Request 2:** `POST /planner/tasks` with `title: "Perform cross-browser testing"`. This request includes a `dependsOn: ["1"]` property and uses the bucket ID from the first request's response (`$1.id`).
    3. **Request 3:** `POST /planner/tasks` with `title: "Run performance audit"`. This also includes `dependsOn: ["1"]`.
  - **System Response:** "Okay, I've created the 'Testing' bucket in the 'Website Redesign' plan and added the tasks 'Perform cross-browser testing' and 'Run performance audit' to it."

Use Case 5: RAG-Enabled Task Creation

- **Objective:** A user wants to create a task based on information from an external document.
- **Conversation Flow:**
  - **User:** "Based on the 'Project Alpha Scope' document, create a summary of the key deliverables and then create a task for me to 'Review key deliverables with the team'."
  - **LLM Analysis (Step 1 - RAG):** The LLM first recognizes the need for information retrieval. It calls a `rag.query(document_name='Project Alpha Scope', query='key deliverables')` tool.
  - **Backend Action (RAG):** The RAG service retrieves the relevant chunks from the document stored in `pgvector` and returns them as context.
  - **LLM Analysis (Step 2 - Summarize & Create):** The LLM receives the context, generates a summary, and then identifies the second part of the request.
    - **Intent:** `planner.create_task`

        - **Entities:**
            - `title`: “Review key deliverables with the team”
            - `assignee`: “me”
    - **System Response:** (After asking for the plan name) “Here is a summary of the key deliverables from the ‘Project Alpha Scope’ document: [LLM-generated summary]. I have also created the task ‘Review key deliverables with the team’ in the ‘Project Alpha’ plan and assigned it to you.”

9.0 Complete Docker Deployment Configuration

The entire system is orchestrated via a single `docker-compose.yml` file. This file defines each service, its configuration, dependencies, networking, and data persistence strategy, providing a portable and reproducible deployment.

```yaml
# docker-compose.yml
# Version 3.9 is recommended for modern Docker features.
version: '3.9'

# Define named volumes for persistent data storage.
# This ensures data survives container restarts and removals.
volumes:
  postgres_data:
    driver: local
  openwebui_data:
    driver: local
  ollama_models:
    driver: local

services:
  # PostgreSQL Database Service for OpenWebUI persistence.
  postgres:
    image: postgres:16
    container_name: planner_postgres
    environment:
      # Credentials for the OpenWebUI database.
      - POSTGRES_USER=openwebui
      - POSTGRES_PASSWORD=openwebui
      - POSTGRES_DB=openwebui
    volumes:
      # Mount the named volume to the PostgreSQL data directory.
      - postgres_data:/var/lib/postgresql/data
    restart: unless-stopped
    healthcheck:
      # Healthcheck ensures the database is ready before other services connect.
      test: ["CMD-SHELL", "pg_isready -U openwebui -d openwebui"]
      interval: 10s
      timeout: 5s
      retries: 5

  # OpenWebUI: The central conversational AI hub.
  open-webui:
    image: ghcr.io/open-webui/open-webui:main
    container_name: planner_openwebui
    ports:
      # Expose OpenWebUI on port 3000 of the host machine.
      - "3000:8080"
    environment:
      # Connection string for the PostgreSQL database.
      - 'DATABASE_URL=postgresql://openwebui:openwebui@postgres:5432/openwebui'
      # Internal network address for the Ollama service.
      - 'OLLAMA_BASE_URL=http://ollama:11434'
      # Disable authentication for a simple single-user setup.
      - 'WEBUI_AUTH=false'
      - 'ENABLE_SIGNUP=false'
    volumes:
      # Mount the named volume for OpenWebUI's configuration and data.
      - openwebui_data:/app/backend/data
    depends_on:
      # Control startup order: wait for postgres and ollama.
      postgres:
        condition: service_healthy
      ollama:
        condition: service_started
    restart: always

  # Ollama: Service for running local LLMs.
```

```
  ollama:
    image: ollama/ollama
    container_name: planner_ollama
    # Uncomment the 'deploy' section to enable GPU acceleration if the NVIDIA
Container Toolkit is installed on the host.
    # deploy:
    #   resources:
    #     reservations:
    #       devices:
    #         - driver: nvidia
    #           count: all
    #           capabilities: [gpu]
    volumes:
      # Mount the named volume to persist downloaded LLM models.
      - ollama_models:/root/.ollama
    restart: unless-stopped

  # Custom Planner MCP Server: The backend tool server.
  planner-mcp-server:
    build:
      # Build the image from the Dockerfile in the specified directory.
      context: ./planner-mcp-server
    container_name: planner_mcp_server
    environment:
      # Pass secrets from a .env file to the container.
      - 'MS_CLIENT_ID=${MS_CLIENT_ID}'
      - 'MS_CLIENT_SECRET=${MS_CLIENT_SECRET}'
      - 'MS_TENANT_ID=${MS_TENANT_ID}'
    restart: unless-stopped
    healthcheck:
      # A simple healthcheck to ensure the server's API is responsive.
      test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
      interval: 15s
      timeout: 5s
      retries: 3

  # MCPO Proxy: Translates between OpenAPI and MCP.
  mcpo-proxy:
    image: ghcr.io/open-webui/mcpo:latest
    container_name: planner_mcpo_proxy
    command: --mcp-server-url http://planner-mcp-server:8000
    ports:
      # Expose the MCPO port to the host for easy access to the OpenAPI spec URL.
      - "8001:8000"
    depends_on:
      # Ensure the backend MCP server is healthy before the proxy starts.
      planner-mcp-server:
        condition: service_healthy
    restart: unless-stopped
```

10. Code Examples and Templates

This section provides illustrative code snippets for the key components of the system. These are templates intended to guide development, not production-ready code.

10.1 Planner MCP Server (FastAPI Example)

This snippet shows how to define a tool endpoint in a Python FastAPI application, which will serve as the Planner MCP Server.

```
# main.py of planner-mcp-server
from fastapi import FastAPI, HTTPException, Request
from pydantic import BaseModel
import msgraph_service # A custom module for Graph API logic

app = FastAPI()

class CreateTaskPayload(BaseModel):
    plan_name: str
    title: str
    bucket_name: str | None = None
    assignee_email: str | None = None

@app.post("/tools/planner.create_task")
async def create_task(payload: CreateTaskPayload, request: Request):
    """
    Creates a new task in a specified Planner plan.
    """
    # Assume user_auth_token is retrieved from a secure session or header
    user_auth_token = request.headers.get("X-User-Token")
    if not user_auth_token:
        raise HTTPException(status_code=401, detail="User token not provided")

    try:
        # Delegate the complex logic to a dedicated service module
        new_task = await msgraph_service.create_planner_task(
            token=user_auth_token,
            plan_name=payload.plan_name,
            title=payload.title,
            bucket_name=payload.bucket_name,
            assignee_email=payload.assignee_email
        )
        return {"status": "success", "task_id": new_task.get("id"), "title":
new_task.get("title")}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.get("/health")
def health_check():
    return {"status": "ok"}
```

10.2 Microsoft Graph API Integration Snippet

This example shows a function within the `msgraph_service` module for updating a task, demonstrating ETag handling.

```python
# msgraph_service.py
import httpx
from microsoft_graph_sdk import GraphServiceClient
from azure.identity import OnBehalfOfCredential

async def update_planner_task(token: str, task_id: str, update_payload: dict):
    """
    Updates a Planner task, correctly handling ETags for concurrency.
    """
    # Assume get_graph_client creates a GraphServiceClient with the user's token
    graph_client = get_graph_client(token)

    # 1. Fetch the task to get the latest ETag
    try:
        task_to_update = await graph_client.planner.tasks.by_task_id(task_id).get()
        etag = task_to_update.additional_data.get("@odata.etag")
    except Exception as e:
        # Handle case where task is not found
        raise ValueError(f"Could not fetch task with ID {task_id}: {e}")

    # 2. Perform the update with the ETag in the If-Match header
    headers = {
        "If-Match": etag,
        "Content-Type": "application/json"
    }

    # The SDK might handle this automatically, but manual request shows the principle
    async with httpx.AsyncClient() as client:
        response = await client.patch(
            f"https://graph.microsoft.com/v1.0/planner/tasks/{task_id}",
            headers={**graph_client.get_headers(), **headers},
# Combine auth and ETag headers
            json=update_payload
        )

    if response.status_code == 412:
        raise ConnectionAbortedError("Precondition Failed: The task was modified by
someone else. Please try again.")

    response.raise_for_status()
    return response.json()
```

10.3 RAG Retrieval Chain (LangChain Example)

This snippet demonstrates how to build a simple retrieval chain using LangChain with a vector store (like `pgvector`).

```python
# rag_service.py
from langchain_community.vectorstores import PGVector
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnablePassthrough
from langchain_openai import ChatOpenAI, OpenAIEmbeddings

# Assume 'vector_store' is an initialized PGVector instance
vector_store = PGVector(
    connection_string="postgresql+psycopg2://user:pass@host:5432/db",
    embedding_function=OpenAIEmbeddings(),
    collection_name="project_docs"
)
retriever = vector_store.as_retriever()
llm = ChatOpenAI(model="gpt-4o")

template = """
Answer the question based only on the following context:
{context}

Question: {question}
"""
prompt = ChatPromptTemplate.from_template(template)

# Define the RAG chain
rag_chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)

def query_docs(question: str):
    return rag_chain.invoke(question)
```

10.4 Document Generation (WeasyPrint Example)

This example shows a function to generate a PDF summary of tasks using a Jinja2 template and WeasyPrint.

```
# document_generator.py
from jinja2 import Environment, FileSystemLoader
from weasyprint import HTML

def generate_task_summary_pdf(tasks: list, plan_name: str, output_path: str):
    """
    Generates a PDF report from a list of task objects.
    """
    env = Environment(loader=FileSystemLoader('.'))
    template = env.get_template("report_template.html")

    # Render the HTML template with the provided data
    html_out = template.render(tasks=tasks, plan_name=plan_name)

    # Generate the PDF from the rendered HTML
    HTML(string=html_out).write_pdf(output_path)
    print(f"Report generated at {output_path}")

# report_template.html (Jinja2 Template)
"""
<!DOCTYPE html>
<html>
<head>
    <title>Task Summary for {{ plan_name }}</title>
    <style> body { font-family: sans-serif; } table { width: 100%; border-collapse:
collapse; } th, td { border: 1px solid #ddd; padding: 8px; } </style>
</head>
<body>
    <h1>Task Summary for Plan: {{ plan_name }}</h1>
    <table>
        <tr><th>Title</th><th>Due Date</th><th>Assignees</th></tr>
        {% for task in tasks %}
        <tr>
            <td>{{ task.title }}</td>
            <td>{{ task.dueDate or 'N/A' }}</td>
            <td>{{ task.assignees | join(', ') }}</td>
        </tr>
        {% endfor %}
    </table>
</body>
</html>
"""
```

11.0 Security Best Practices

Security is a critical consideration throughout the system's lifecycle. The following best practices must be implemented to protect user data and system integrity.

11.1 OAuth 2.0 and API Permissions

The interaction with the Microsoft Graph API is the most sensitive part of the system. Adherence to OAuth 2.0 best practices is non-negotiable.

- **Authorization Code Flow:** The system must use the OAuth 2.0 Authorization Code Flow. This is the most secure flow for applications with a backend, as it ensures that access tokens are never exposed to the user's browser. The Implicit Grant Flow is deprecated and must not be used.
- **Principle of Least Privilege:** When registering the application in Microsoft Entra ID, request only the permissions absolutely necessary for its function. For this system, the `Tasks.ReadWrite` deleg-

ated scope is required. Avoid requesting overly broad permissions like `Tasks.ReadWrite.All`, which are application-level permissions and grant access to all Planner data in the organization, a significant security risk for a user-facing application.

- **Secure Token Storage:** The refresh tokens obtained through the OAuth flow are long-lived and highly sensitive. They must be encrypted at rest in the system's database. The application should be designed to handle token revocation and re-authentication gracefully if a token is compromised or expires.

11.2 API Key and Secret Management

The system relies on several secrets, including the Microsoft Entra application's client secret and API keys for any cloud-based LLM providers.

- **Use `.env` Files for Development:** For local development, store all secrets in a `.env` file in the root of the project. This file should be explicitly listed in `.gitignore` to prevent it from ever being committed to version control. The `docker-compose.yml` file can then reference these variables, keeping the configuration file itself free of sensitive data.
- **Use Docker Secrets for Production:** For a more secure deployment, use Docker Secrets. This feature allows for the secure management and transmission of secrets to containers. Secrets are stored in an encrypted Raft log (in Swarm mode) and are mounted into the container as in-memory files, which is more secure than using environment variables that can be inadvertently exposed through logs or container inspection.

11.3 Single-User Deployment Considerations

The proposed architecture is for a single-user or small-team deployment. Several steps should be taken to secure this environment:

- **Disable Public Sign-up:** In the OpenWebUI configuration, set `ENABLE_SIGNUP=false`. This prevents unauthorized users from creating accounts on the service.
- **Network Security:** Do not expose the PostgreSQL database port or the Ollama port directly to the internet. These services should only be accessible within the private Docker network. Only the OpenWebUI port (and potentially the MCPO port for initial setup) should be exposed.
- **Reverse Proxy:** For any deployment that is accessible from the internet, place the entire application stack behind a reverse proxy like Nginx or Traefik. The reverse proxy can handle SSL/TLS termination (encrypting all traffic), apply access controls, and provide an additional layer of security.

12.0 Performance Optimization Strategies

Ensuring a responsive and efficient system requires optimization at multiple layers of the architecture, from backend API interactions to LLM and database performance.

12.1 Microsoft Graph API Optimization

- **JSON Batching:** This is the single most effective optimization. For user commands that translate into multiple sequential API calls (e.g., create a bucket, then create a task), these operations should be combined into a single `POST` request to the `/$batch` endpoint. This dramatically reduces network latency and the number of requests counted against rate limits.
- **Query Parameters:** When retrieving data, always use the `$select` OData query parameter to request only the fields the application needs. This reduces the size of the response payload. Use the `$filter` parameter to perform server-side filtering (e.g., by completion status or creation date) to avoid fetching and processing unnecessary data on the client side.

- **Delta Queries for Polling:** For the data synchronization mechanism, it is critical to use delta queries instead of repeatedly fetching all tasks. A delta query returns only the items that have been created, updated, or deleted since the last request, making the polling process highly efficient.

12.2 LLM and RAG Performance

- **Model Selection:** Use the most appropriate LLM for the job. For simple intent recognition and function calling, a smaller, faster model (like GPT-4o mini or a local 7B parameter model) may be sufficient and more cost-effective. Reserve larger, more powerful models for complex reasoning or summarization tasks.
- **Local LLM Acceleration:** If using Ollama, run it on a machine with a compatible GPU and ensure the NVIDIA Container Toolkit is properly configured for the Docker service. This can increase inference speed by an order of magnitude compared to running on a CPU.
- **RAG Optimization:** The quality and performance of the RAG pipeline depend on the embedding model and chunking strategy. Choose a high-performing embedding model from a benchmark like the MTEB leaderboard. Experiment with different chunking strategies (e.g., recursive character splitting) and chunk sizes to find the optimal balance between context granularity and retrieval performance.

12.3 Database and Caching

- **Database Tuning:** While the default PostgreSQL configuration is robust, performance can be improved by tuning settings like `shared_buffers` and `work_mem` in the `postgresql.conf` file to better utilize the host's available memory.
- **Caching:** Implement a caching layer (e.g., using Redis) for frequently accessed, semi-static data from the Graph API, such as the list of a user's plans or the members of a group. This can reduce the number of repetitive API calls. The cache should have a reasonable time-to-live (TTL) to ensure data does not become too stale.

13.0 Testing and Validation Approach

A multi-layered testing strategy is essential to ensure the system is reliable, accurate, and robust. This approach combines unit, integration, and end-to-end testing to validate each component individually and the system as a whole.

13.1 Unit Testing

- **Scope:** Focus on individual functions and classes within the Planner MCP Server and other custom services.
- **Methodology:** Use a testing framework like `pytest` for Python. All interactions with the Microsoft Graph API should be mocked using libraries like `pytest-mock` or `httpx.mock`. This allows for testing the business logic (e.g., how a payload is constructed, how an ETag is handled) in isolation without making actual network calls. Test cases should cover success scenarios, expected failures (e.g., invalid input), and edge cases.

13.2 Integration Testing

- **Scope:** Test the interaction points between the system's containerized services.
- **Methodology:** Write automated tests that validate the communication flow between components. For example, an integration test could send an HTTP request to the MCPO proxy's endpoint and assert that the Planner MCP Server receives a correctly translated MCP request. This requires running the relevant containers (MCPO and MCP Server) in a test environment. Another key integ-

ration test is to validate the connection and queries between the RAG service and the `pgvector` database.

13.3 End-to-End (E2E) Testing

- **Scope:** Validate the entire workflow from the user's perspective, starting with a natural language prompt and ending with a verified change in a Microsoft Planner instance.
- **Methodology:** This requires a dedicated test tenant in Microsoft 365. E2E tests can be automated using a framework that can interact with the system's entry point (e.g., by making API calls to the OpenWebUI backend). The test script would send a conversational prompt (e.g., "Create a test task"), and then use the Microsoft Graph API directly to query the test Planner instance and verify that the task was created with the correct properties. These tests are critical for catching regressions and validating the LLM's function-calling accuracy.

13.4 Performance and Load Testing

- **Scope:** Assess the system's resilience to throttling and its performance under load.
- **Methodology:** Use a tool like Locust or k6 to script a series of rapid, repeated API calls against the Planner MCP Server. The goal is not to test the Graph API itself, but to verify that the server's exponential backoff and retry logic functions correctly when it receives HTTP `429` responses. This ensures the system behaves gracefully under pressure and does not fail.

14.0 Maintenance and Troubleshooting Guide

This guide provides instructions for common maintenance tasks and steps for diagnosing and resolving potential issues.

14.1 Common Maintenance Tasks

- **Updating Container Images:** Regularly update the base images for services like OpenWebUI, PostgreSQL, and Ollama to incorporate the latest features, bug fixes, and security patches. This can be done by running `docker compose pull` followed by `docker compose up -d --remove-orphans`.
- **Backing Up Data Volumes:** Data persistence relies on Docker volumes. It is critical to perform regular backups. This can be scripted by stopping the application (`docker compose down`), then running a temporary container that mounts the volume and creates a `tar` archive of its contents, which is then saved to a secure, external location.
  - **Backup Command Example:** `docker run --rm -v planner_postgres_data:/data -v $(pwd)/backups:/backups ubuntu tar czf /backups/postgres_backup_$(date +%F).tar.gz /data`
- **Monitoring API Usage:** Keep track of Microsoft Graph API usage and any costs associated with cloud-based LLM providers. Set up alerts for unusual spikes in activity that could indicate a bug or misuse.

14.2 Troubleshooting Common Issues

Issue	Symptoms	Diagnostic Steps	Resolution
Authentication Failure	User receives “Authentication failed” or “Permission denied” errors.	1. Check the logs of the Planner MCP Server (`docker logs planner_mcp_server`) for OAuth-related errors. 2. Verify that the `MS_CLIENT_ID` and `MS_CLIENT_SECRET` in the `.env` file are correct. 3. Ensure the user has consented to the required permissions.	Re-authenticate the user through the OAuth flow. If the issue persists, check the application registration settings in Microsoft Entra ID.
API Throttling	Responses are slow or fail with “Too Many Requests” errors.	1. Examine the MCP server logs for HTTP `429` status codes. 2. Review the code to ensure JSON batching is used for multi-step operations.	Increase the delay in the exponential backoff algorithm. Optimize the frequency of polling for the delta query sync mechanism.
Container Fails to Start	A service is stuck in a “restarting” loop in `docker compose ps` .	1. Check the logs of the failing container (`docker logs <container_name>`) for startup errors. 2. Common causes include incorrect environment variables, missing volumes, or port conflicts on the host.	Correct the configuration in the `docker-compose.yml` or `.env` file and restart the application.
Incorrect LLM Behavior	The LLM fails to call the correct tool or provides incorrect parameters.	1. Review the conversation history in OpenWebUI to see the exact function call the LLM attempted. 2. Check the system prompt for clarity and accuracy in the tool descriptions.	Refine the prompt engineering. Provide more detailed descriptions or few-shot examples of how to use the tools. Consider using a more capable LLM.

15.0 Conclusion and Future Roadmap

This report has laid out a comprehensive and technically sound strategy for developing an intelligent Microsoft Teams Planner management system. The recommended architecture, centered on a custom-built, containerized application that integrates best-in-class open-source components, provides a powerful and flexible foundation. By leveraging OpenWebUI as a conversational hub and building a standardized MCP tool server for Planner interactions, the system is designed for maintainability, extensibility, and long-term viability. The feasibility of the project is high, provided that the development team carefully navigates the known constraints of the Microsoft Graph API, particularly its rate limits and lack of real-time notifications.

15.1 Final Recommendations and Action Items

The immediate action items for the project team are as follows:

1. **Establish the Development Environment:** Set up the Docker environment as specified in the implementation guide and validate the baseline functionality of the OpenWebUI, PostgreSQL, and Ollama services.
2. **Register the Azure Application:** Proceed with the application registration in Microsoft Entra ID to secure the necessary credentials and configure API permissions.
3. **Develop a Proof-of-Concept (PoC) for the MCP Server:** Begin development of the Planner MCP Server, focusing first on implementing a successful OAuth 2.0 authentication flow and a single, core function, such as `get_user_tasks`. This will validate the most critical and complex part of the backend.
4. **Select an Initial LLM:** Choose and download a suitable LLM via Ollama that has strong function-calling capabilities to serve as the primary model for development and testing.

15.2 Future Roadmap

The modular architecture of the system provides a clear path for future enhancements and increased sophistication. Potential roadmap items include:

- **Advanced GraphRAG Implementation:** Integrate Neo4j to build a dynamic knowledge graph of Planner data. This would enable the system to answer complex relational and temporal queries, such as “What is the critical path for Project Alpha?” or “Show me all tasks that were blocked by Sarah last month.”
- **Proactive Notifications and Insights:** While real-time webhooks are not available, the polling mechanism can be used to drive a proactive notification system. The AI could monitor for changes (e.g., overdue tasks, new assignments) and send intelligent alerts or daily summaries to the user in Teams.
- **Multi-User and Tenant Support:** Evolve the architecture to support multiple users securely. This would involve moving from a single-user authentication model to a full multi-tenant OAuth implementation and introducing user-level data isolation.
- **Deeper Microsoft 365 Integration:** Expand the MCP server’s capabilities to interact with other Microsoft 365 services, such as creating a Planner task from an Outlook email, attaching a file from OneDrive, or linking a task to a OneNote page.

15.3 Success Metrics

The success of the Intelligent Planner Management System will be measured by its ability to enhance user productivity and streamline workflows. Key performance indicators (KPIs) to track include:

- **User Adoption:** The number of active users regularly interacting with the system.

- **Task Management Velocity:** A measurable increase in the number of tasks created, updated, and completed through the natural language interface compared to the manual UI.
- **Reduction in Context Switching:** User-reported feedback on time saved by not having to switch between the Teams and Planner applications.
- **Task Completion Rate:** An analysis of whether tasks created via the AI assistant have a higher or faster completion rate.

By adhering to the technical blueprint outlined in this report and focusing on these success metrics, the development team can deliver a transformative tool that seamlessly integrates intelligent task management into the heart of the modern collaborative workspace.

References

30 Days of Microsoft Graph - Day 21: Use case: Create plans, buckets, and tasks in Planner - Microsoft 365 Developer Blog (https://devblogs.microsoft.com/microsoft365dev/30daysmsgraph-day-21-use-case-create-plans-buckets-and-tasks-in-planner/)
5 Best Technologies to Build Microservices Architecture - clariontech.com (https://www.clariontech.com/blog/5-best-technologies-to-build-microservices-architecture)
A Comparative Analysis of LLM Application Frameworks for Enterprise AI - ijgisonline.com (https://ij-gis.pubpub.org/pub/6yecqicl)
A Developer’s Friendly Guide to Qdrant Vector Database - cohorte.co (https://www.cohorte.co/blog/a-developers-friendly-guide-to-qdrant-vector-database)
AI-Powered Web Scraping Solutions 2025 - firecrawl.dev (https://www.firecrawl.dev/blog/ai-powered-web-scraping-solutions-2025)
Active ‘apache-tika’ Questions - stackoverflow.com (https://stackoverflow.com/questions/tagged/apache-tika?tab=Active)
An Empirical Study of Docker Compose Configuration Patterns - arxiv.org (https://arxiv.org/html/2305.11293v2)
Announcing the GitHub integration with Microsoft Teams - GitHub Blog (https://github.blog/news-insights/product-news/announcing-the-github-integration-with-microsoft-teams/)
Announcing updates to the Planner API in Microsoft Graph - Microsoft 365 Developer Blog (https://devblogs.microsoft.com/microsoft365dev/announcing-updates-to-the-planner-api-in-microsoft-graph/)
Apache Tika – a content analysis toolkit - tika.apache.org (https://tika.apache.org/)
Apache Tika, an underrated alternative to Unstructured.io for RAG and fine-tuning - reddit.com (https://www.reddit.com/r/LocalLLaMA/comments/1aq1qkd/apache_tika_an_underrated_alternative_to/)
Authentication and authorization basics for Microsoft Graph - Microsoft Graph (https://learn.microsoft.com/en-us/graph/auth/auth-concepts)
Awesome-GraphRAG - github.com (https://github.com/DEEP-PolyU/Awesome-GraphRAG)
BeautifulSoup4 vs Scrapy Comparison - firecrawl.dev (https://www.firecrawl.dev/blog/beautifulsoup4-vs-scrapy-comparison)
Benchmarks - qdrant.tech (https://qdrant.tech/benchmarks/)
Best AI Web Scraping Tools - scrapeops.io (https://scrapeops.io/web-scraping-playbook/best-ai-web-scraping-tools/)
Best Open Source Web Scraping Libraries - firecrawl.dev (https://www.firecrawl.dev/blog/best-open-source-web-scraping-libraries)
Best python library for generating PDFs? - reddit.com (https://www.reddit.com/r/Python/comments/82z6cw/best_python_library_for_generating_pdfs/)
Build a basic AI chatbot in Teams - Microsoft Teams (https://learn.microsoft.com/en-us/microsoftteams/platform/toolkit/build-a-basic-ai-chatbot-in-teams)

Build a document question-answering system with Docling and Granite - ibm.com (https://www.ibm.com/think/tutorials/build-document-question-answering-system-with-docling-and-granite)
Build an interactive app with change notifications via webhooks - Microsoft Graph (https://learn.microsoft.com/en-us/graph/patterns/interactive-app-with-change-notifications-via-webhooks)
Building a basic MCP server with Python - Medium (https://medium.com/data-engineering-with-dremio/building-a-basic-mcp-server-with-python-4c34c41031ed)
Building an MCP server as an API developer - Medium (https://heeki.medium.com/building-an-mcp-server-as-an-api-developer-cfc162d06a83)
Building custom MCP tool and integrate it to your self-hosted LLM through OpenWebUI (Part 3) - Medium (https://juniarto-samsudin.medium.com/building-custom-mcp-tool-and-integrate-it-to-your-self-hosted-llm-through-openwebui-part-3-3268c4fcac6e)
Building Temporal Knowledge Graphs with Graphiti - falkordb.com (https://www.falkordb.com/blog/building-temporal-knowledge-graphs-graphiti/)
Case Study: GenAI Tools - tuhinsharma.com (https://tuhinsharma.com/blogs/case-study-genai-tools/)
Change notifications for Microsoft Graph resources - Microsoft Graph v1.0 (https://learn.microsoft.com/en-us/graph/api/resources/change-notifications-api-overview?view=graph-rest-1.0)
Choose the Best Embedding Model for Your Retrieval-Augmented Generation (RAG) System - enterprisebot.ai (https://www.enterprisebot.ai/blog/choose-the-best-embedding-model-for-your-retrieval-augmented-generation-rag-system)
Choosing the Right Technology Stack for Your Microservices - medium.com (https://medium.com/@jpromanonet/choosing-the-right-technology-stack-for-your-microservices-a863e34bf755)
claude-task-master - GitHub (https://github.com/eyaltoledano/claude-task-master)
Combine multiple HTTP requests using JSON batching - Microsoft Graph (https://learn.microsoft.com/en-us/graph/json-batching)
compose - github.com (https://github.com/docker/compose)
Convert DOC to PPT in Python - blog.aspose.com (https://blog.aspose.com/total/convert-doc-to-ppt-python/)
Convert DOC to PPT in Python - gist.github.com (https://gist.github.com/aspose-com-gists/3d5bf343526692ab0cf44ad9c0487ece)
Create PDF Documents in Python with ReportLab - pythonassets.com (https://pythonassets.com/posts/create-pdf-documents-in-python-with-reportlab/)
Create selectable PDF files with Lambda, Python, and ReportLab - dev.to (https://dev.to/aws-builders/create-selectable-pdf-files-with-lambda-python-and-reportlab-5gp0)
Creating and Updating PowerPoint Presentations in Python using python-pptx - geeksforgeeks.org (https://www.geeksforgeeks.org/python/creating-and-updating-powerpoint-presentations-in-python-using-python-pptx/)
Creating Various PDF Files via Python - medium.com (https://akpolatcem.medium.com/creating-various-pdf-files-via-python-eba91a40df9d)
DataWalk as a Neo4j Alternative - datawalk.com (https://datawalk.com/datawalk/neo4jalternative/)
Deploy Neo4j with Docker - render.com (https://render.com/deploy-docker/neo4j)
Deployment of Neo4j Docker container with APOC and Graph-Algorithms plugins - medium.com (https://medium.com/swlh/deployment-of-neo4j-docker-container-with-apoc-and-graph-algorithms-plugins-bf48226928f4)
Development - OpenWebUI Docs (https://docs.openwebui.com/getting-started/advanced-topics/development/)
Docling - docling-project.github.io (https://docling-project.github.io/docling/)
Docling - docling.ai (https://www.docling.ai/)
Docling - github.com (https://github.com/docling-project/docling)
Docling: An Efficient Open-Source Toolkit for AI-Driven Document Conversion - research.ibm.com (https://research.ibm.com/publications/docling-an-efficient-open-source-toolkit-for-ai-driven-document-

conversion)
Docling: An open-source toolkit for AI-driven document conversion - research.ibm.com (https://research.ibm.com/blog/docling-generative-AI)
Docling's rise: How IBM is making data LLM-ready - ibm.com (https://www.ibm.com/think/news/doclings-rise-llm-ready-data)
Documentation - qdrant.tech (https://qdrant.tech/documentation/)
Docker - neo4j.com (https://neo4j.com/docs/operations-manual/current/docker/)
Docker Compose overview - docs.docker.com (https://docs.docker.com/compose/)
docker-neo4j - github.com (https://github.com/neo4j/docker-neo4j)
Docxtemplater - docxtemplater.com (https://docxtemplater.com/)
Events - OpenWebUI Docs (https://docs.openwebui.com/features/plugin/events/)
Examples - Model Context Protocol (https://modelcontextprotocol.io/examples)
Features - OpenWebUI Docs (https://docs.openwebui.com/features/)
Firecrawl vs. BeautifulSoup - blog.apify.com (https://blog.apify.com/firecrawl-vs-beautifulsoup/)
Flask PDF Generation: ReportLab, WeasyPrint, and PDFKit Compared - codingeasypeasy.com (https://www.codingeasypeasy.com/blog/flask-pdf-generation-reportlab-weasyprint-and-pdfkit-compared)
Free/cheaper alternatives to Neo4j for a personal project? - reddit.com (https://www.reddit.com/r/Database/comments/jpmnxp/freecheaper_alternatives_to_neo4j_for_a/)
From RAG to GraphRAG: A Technical Review - arxiv.org (https://arxiv.org/abs/2506.05690)
Functions - OpenWebUI Docs (https://docs.openwebui.com/features/plugin/functions/)
Generate embeddings with Azure OpenAI - learn.microsoft.com (https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/rag/rag-generate-embeddings)
Generate PDFs in Python with Libraries - templated.io (https://templated.io/blog/generate-pdfs-in-python-with-libraries/)
Generating PDF in Python - glukhov.org (https://www.glukhov.org/post/2025/05/generating-pdf-in-python/)
Get access on behalf of a user - Microsoft Graph (https://learn.microsoft.com/en-us/graph/auth-v2-user)
Get access without a user - Microsoft Graph (https://learn.microsoft.com/en-us/graph/auth-v2-service)
Get change notifications in push mode - Microsoft Graph (https://learn.microsoft.com/en-us/graph/patterns/notifications-in-push-mode)
Get change notifications through Azure Event Hubs - Microsoft Graph (https://learn.microsoft.com/en-us/graph/change-notifications-delivery-event-hubs)
Getting Started - OpenWebUI Docs (https://docs.openwebui.com/getting-started/)
Getting Started with Microsoft Graph API Webhooks: Real-Time Notifications for Your App - Medium (https://medium.com/@mohamedamine.hajji/getting-started-with-microsoft-graph-api-webhooks-real-time-notifications-for-your-app-6b53169373bb)
Getting labels from planner with microsoft graph api - Stack Overflow (https://stackoverflow.com/questions/71420902/getting-labels-from-planner-with-microsoft-graph-api)
GitHub and Microsoft Planner Integration - Make.com (https://www.make.com/en/integrations/github/microsoft-planner)
GitHub Integration with Microsoft Teams - GitHub (https://github.com/integrations/microsoft-teams)
GitHub Topic: ai-chatbot - GitHub (https://github.com/topics/ai-chatbot)
GitHub Topic: conversational-ai - GitHub (https://github.com/topics/conversational-ai)
GitHub Topic: microsoft-planner - GitHub (https://github.com/topics/microsoft-planner)
GitHub Topic: task-management - GitHub (https://github.com/topics/task-management)
GitHub Topic: task-management-app - GitHub (https://github.com/topics/task-management-app)
GitHub Topic: task-oriented-dialogue - GitHub (httpscom/topics/task-oriented-dialogue)
Graph API - Increase Rate Limiting / Throttling - Microsoft Q&A (https://learn.microsoft.com/en-us/answers/questions/219222/graph-api-increase-rate-limiting-throttling)
Graph API Integration for SaaS Developers - Microsoft Tech Community (https://techcom-

munity.microsoft.com/blog/fasttrackforazureblog/graph-api-integration-for-saas-developers/4038603)
Graph RAG - huggingface.co (https://huggingface.co/papers?q=Graph+RAG)
GraphRAG Explained: Enhancing RAG with Knowledge Graphs - medium.com (https://medium.com/@zilliz_learn/graphrag-explained-enhancing-rag-with-knowledge-graphs-3312065f99e1)
Graphiti: Building Temporal Knowledge Graphs for Agentic Memory - neo4j.com (https://neo4j.com/blog/developer/graphiti-knowledge-graph-memory/)
graphiti - github.com (https://github.com/getzep/graphiti)
How can I automate the generation of a PowerPoint with Python? - stackoverflow.com (https://stackoverflow.com/questions/71359430/how-can-i-automate-the-generation-of-a-powerpoint-with-python)
How does Haystack differ from other search frameworks like LangChain and LlamaIndex? - milvus.io (https://milvus.io/ai-quick-reference/how-does-haystack-differ-from-other-search-frameworks-like-langchain-and-llamaindex)
How to avoid Microsoft Graph API throttling and optimize network traffic - DEV Community (https://dev.to/this-is-learning/how-to-avoid-microsoft-graph-api-throttling-and-optimize-network-traffic-5c2g)
How to build an AI Chatbot with Redis, Python, and GPT - freeCodeCamp (https://www.freecodecamp.org/news/how-to-build-an-ai-chatbot-with-redis-python-and-gpt/)
How to Run Multiple Containers with Docker Compose - freecodecamp.org (https://www.freecodecamp.org/news/run-multiple-containers-with-docker-compose/)
How to use Tools with Open WebUI - Medium (https://medium.com/write-a-catalyst/how-to-use-tools-with-open-webui-9725db2724bb)
IBM Automation Document Processing - ibm.com (https://www.ibm.com/products/document-processing)
IBM Granite-Docling: End-to-end document conversion for AI - ibm.com (https://www.ibm.com/new/announcements/granite-docling-end-to-end-document-conversion)
ibm-granite/granite-docling-258M - huggingface.co (https://huggingface.co/ibm-granite/granite-docling-258M)
Integrate GitHub with Microsoft Planner - Appy Pie Automate (https://www.appypieautomate.ai/integrate/apps/github/integrations/microsoft-planner)
Introduction - neo4j.com (https://neo4j.com/docs/operations-manual/current/docker/introduction/)
Introducing the Azure MCP Server - Azure SDK Blog (https://devblogs.microsoft.com/azure-sdk/introducing-the-azure-mcp-server/)
Is it possible to get real time update for updates via microsoft graph webhooks? - Stack Overflow (https://stackoverflow.com/questions/62297678/is-it-possible-to-get-real-time-update-for-updates-via-microsoft-graph-webhooks)
LangChain, LlamaIndex, or Haystack: Which Framework Suits Your LLM Needs? - medium.com (https://dkaarthick.medium.com/langchain-llamaindex-or-haystack-which-framework-suits-your-llm-needs-7408fee8ab1e)
Live-Chatbot-for-Final-Year-Project - GitHub (https://github.com/Vatshayan/Live-Chatbot-for-Final-Year-Project)
LlamaIndex vs LangChain vs Haystack - medium.com (https://medium.com/@heyamit10/llamaindex-vs-langchain-vs-haystack-4fa8b15138fd)
LlamaIndex vs LangChain vs Haystack: Choosing the right one for your LLM application - linkedin.com (https://www.linkedin.com/pulse/llamaindex-vs-langchain-haystack-choosing-right-one-subramaniam-yvere)
LlamaIndex vs LangChain: Which is better? - blog.n8n.io (https://blog.n8n.io/llamaindex-vs-langchain/)
MS Graph API Planner - Query all task details for a plan - Stack Overflow (https://stackoverflow.com/questions/71084675/ms-graph-api-planner-query-all-task-details-for-a-plan)
Mastering RAG: How to Select an Embedding Model - galileo.ai (https://galileo.ai/blog/mastering-rag-how-to-select-an-embedding-model)
MCP integration into OpenWebUI - Reddit (https://www.reddit.com/r/OpenWebUI/comments/1jaidh4/mcp_integration_into-openwebui/)

MCP Pipe - OpenWebUI Community (https://openwebui.com/f/haervwe/mcp_pipe)
MCP server step-by-step guide to building from scratch - Composio DevBlog (https://composio.dev/blog/mcp-server-step-by-step-guide-to-building-from-scrtch)
MCPO: Supercharge Open WebUI with MCP Tools - Medium (https://mychen76.medium.com/mcpo-supercharge-open-webui-with-mcp-tools-4ee55024c371)
Microservice Architecture Tech Stack Essentials - springfuse.com (https://www.springfuse.com/microservice-architecture-tech-stack-essentials/)
Microservices architecture style - learn.microsoft.com (https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices)
Microservices Technology Stack Delivers Content as a Service - contentstack.com (https://www.contentstack.com/cms-guides/microservices-technology-stack-delivers-content-as-a-service)
Microsoft Graph API - Oauth 2.0 scopes - Microsoft Q&A (https://learn.microsoft.com/en-us/answers/questions/28515/microsoft-graph-api-oauth-2-0-scopes)
Microsoft Graph API daily limitation - Stack Overflow (https://stackoverflow.com/questions/57107987/microsoft-graph-api-daily-limitation)
Microsoft Graph API limits for Microsoft 365 Copilot connectors - Microsoft Graph (https://learn.microsoft.com/en-us/graph/connecting-external-content-api-limits)
Microsoft Graph API limits? - Reddit (https://www.reddit.com/r/Intune/comments/1jp31de/microsoft_graph_api_limits/)
Microsoft Graph API Overview - Microsoft Learn (https://learn.microsoft.com/en-us/graph/overview)
Microsoft Graph Dev Center - Microsoft Developer (https://developer.microsoft.com/en-us/graph)
Microsoft Graph Documentation - GitHub (https://github.com/microsoftgraph/microsoft-graph-docs-contrib/blob/main/concepts/planner-concept-overview.md)
Microsoft Graph GitHub Repositories - GitHub (https://github.com/microsoftgraph)
Microsoft Graph REST API - Microsoft Developer (https://developer.microsoft.com/en-us/graph/rest-api)
Microsoft Graph Webhooks & Delta Query: Like Peanut Butter & Jelly - Voitanos (https://www.voitanos.io/blog/microsoft-graph-webhook-delta-query/)
Microsoft Graph change notifications API - Microsoft Q&A (https://learn.microsoft.com/en-us/answers/tags/304/microsoft-graph-change-notifications-api)
Microsoft Graph permissions reference - Microsoft Graph (https://learn.microsoft.com/en-us/graph/permissions-reference)
Microsoft Graph service-specific throttling limits - Microsoft Graph (https://learn.microsoft.com/en-us/graph/throttling-limits)
Microsoft Graph throttling guidance - Microsoft Graph (https://learn.microsoft.com/en-us/graph/throttling)
Microsoft Planner Premium REST API - Microsoft Q&A (https://learn.microsoft.com/en-us/answers/questions/2119208/microsoft-planner-premium-rest-api)
Microsoft Planner Tag on Microsoft 365 Developer Blog - Microsoft (https://devblogs.microsoft.com/microsoft365dev/tag/microsoft-planner/)
Microsoft identity platform and OAuth 2.0 On-Behalf-Of flow - Microsoft Entra (https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-on-behalf-of-flow)
Microsoft identity platform and OAuth 2.0 authorization code flow - Microsoft Entra (https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-auth-code-flow)
Microsoft identity platform and OAuth 2.0 client credentials flow - Microsoft Entra (https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-client-creds-grant-flow)
Microsoft identity platform and OAuth 2.0 implicit grant flow - Microsoft Entra (https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-implicit-grant-flow)
MinIO and Tika for Text Extraction - blog.min.io (https://blog.min.io/minio-tika-text-extraction/)
Model Context Protocol - A Complete Tutorial - Medium (https://medium.com/@nimritakoul01/the-model-context-protocol-mcp-a-complete-tutorial-a3abe8a7f4ef)

Model Context Protocol - Anthropic (https://www.anthropic.com/news/model-context-protocol)
Model Context Protocol - OpenCV Blog (https://opencv.org/blog/model-context-protocol/)
Model Context Protocol - seangoedecke.com (https://www.seangoedecke.com/model-context-protocol/)
Model Context Protocol (MCP) Tutorial - DataCamp (https://www.datacamp.com/tutorial/mcp-model-context-protocol)
Model Context Protocol (MCP) Tutorial - DigitalOcean (https://www.digitalocean.com/community/tutorials/model-context-protocol)
Model Context Protocol (MCP) Tutorial: Build Your First MCP Server in 6 Steps - Towards Data Science (https://towardsdatascience.com/model-context-protocol-mcp-tutorial-build-your-first-mcp-server-in-6-steps/)
Model Context Protocol GitHub Organization - GitHub (https://github.com/modelcontextprotocol)
Model Context Protocol Specification - modelcontextprotocol.io (https://modelcontextprotocol.io/specification/2025-06-18)
Model Context Protocol/servers - GitHub (https://github.com/modelcontextprotocol/servers)
Multi-container applications with Docker Compose - learn.microsoft.com (https://learn.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/multi-container-applications-docker-compose)
MySQL vs. PostgreSQL: A 360-Degree Comparison of Syntax, Performance, Scalability, and Features - enterprisedb.com (https://www.enterprisedb.com/blog/postgresql-vs-mysql-360-degree-comparison-syntax-performance-scalability-and-features?lang=en)
MySQL vs. PostgreSQL: A Comprehensive Comparison - portable.io (https://portable.io/learn/psql-2-mysql)
MySQL vs. PostgreSQL: What’s the Difference? - aws.amazon.com (https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/)
MySQL vs Postgres in 2024 - dbconvert.com (https://dbconvert.com/blog/mysql-vs-postgres-in-2024/)
Neo4j Alternative: Open-Source, Distributed, and Lightning-Fast - nebula-graph.io (https://www.nebula-graph.io/posts/Neo4j_Alternative_Open-Source_Distributed_and_Lightning_Fast)
Neo4j Alternative: What Are My Open-Source DB Options? - memgraph.com (https://memgraph.com/blog/neo4j-alternative-what-are-my-open-source-db-options)
Neo4j Alternatives & Competitors - gartner.com (https://www.gartner.com/reviews/market/cloud-database-management-systems/vendor/neo4j/product/neo4j-graphdatabase/alternatives)
Neo4j Alternatives - puppygraph.com (https://www.puppygraph.com/blog/neo4j-alternatives)
Neo4j Deployment Center - neo4j.com (https://neo4j.com/deployment-center/)
Neo4j Documentation - neo4j.com (https://neo4j.com/docs/)
Neo4j Graph Database Alternatives - softwarereviews.com (https://www.softwarereviews.com/categories/119/products/4481/alternatives)
Neo4j Graph Database Competitors and Alternatives - g2.com (https://www.g2.com/products/neo4j-graph-database/competitors/alternatives)
Neo4j vs. Memgraph - memgraph.com (https://memgraph.com/blog/neo4j-vs-memgraph)
neo4j - hub.docker.com (https://hub.docker.com/_/neo4j)
Obsidian BMO Chatbot - AIBase (https://www.aibase.com/repos/project/obsidian-bmo-chatbot)
Open WebUI Tool Creator - OpenWebUI Community (https://openwebui.com/m/mhwhgm/open-web-ui-tool-creator)
OpenAPI Servers for Open WebUI - GitHub Discussions (https://github.com/open-webui/openapi-servers/discussions/58)
OpenAPI Servers: MCP - OpenWebUI Docs (https://docs.openwebui.com/openapi-servers/mcp/)
OpenAPI Servers: Open WebUI - OpenWebUI Docs (https://docs.openwebui.com/openapi-servers/open-webui/)
open-webui/mcpo - GitHub (https://github.com/open-webui/mcpo)
open-webui/open-webui - GitHub (https://github.com/open-webui/open-webui)

open-webui/open-webui/discussions/3134 - GitHub (https://github.com/open-webui/open-webui/discussions/3134)
open-webui/open-webui/discussions/7363 - GitHub (https://github.com/open-webui/open-webui/discussions/7363)
Optimize Open WebUI: Three Practical Extensions for a Better User Experience - Medium (https://medium.com/pythoneers/optimize-open-webui-three-practical-extensions-for-a-better-user-experience-cbe365af60b1)
Overview - qdrant.tech (https://qdrant.tech/documentation/overview/)
Overview of change notifications in Microsoft Graph - Microsoft Graph (https://learn.microsoft.com/en-us/graph/change-notifications-overview)
Overview of the Azure MCP Server - Microsoft Learn (https://learn.microsoft.com/en-us/azure/developer/azure-mcp-server/overview)
PM-bot.ai - AI Project Management Consultant (https://pm-bot.ai/)
PMOtto.ai - AI Project Management Assistant (https://www.pmotto.ai/)
Pattern: Microservice Architecture - microservices.io (https://microservices.io/patterns/microservices.html)
Pattern: Multiple technology stacks per service - microservices.io (https://microservices.io/articles/dark-energy-dark-matter/dark-energy/multiple-technology-stacks.html)
Permissions and consent in the Microsoft identity platform - Microsoft Entra (https://learn.microsoft.com/en-us/entra/identity-platform/scopes-oidc)
Pgvector vs Qdrant: Which is the Best Vector Database? - tigerdata.com (https://www.tigerdata.com/blog/pgvector-vs-qdrant)
pgvector - github.com (https://github.com/pgvector/pgvector)
pgvector 0.5.0 Released - postgresql.org (https://www.postgresql.org/about/news/pgvector-050-released-2700/)
pgvector 0.7.0 Released - postgresql.org (https://www.postgresql.org/about/news/pgvector-070-released-2852/)
pgvector Tutorial: Storing and Querying Vector Embeddings in PostgreSQL - datacamp.com (https://www.datacamp.com/tutorial/pgvector-tutorial)
pgvector: Embeddings and vector similarity - supabase.com (https://supabase.com/docs/guides/database/extensions/pgvector)
Planner - Microsoft Graph Toolkit (https://learn.microsoft.com/en-us/graph/toolkit/components/planner)
planner resource type - Microsoft Graph beta - GitHub (https://github.com/microsoftgraph/microsoft-graph-docs-contrib/blob/main/api-reference/beta/resources/planner-overview.md)
planner resource type - Microsoft Graph beta (https://learn.microsoft.com/en-us/graph/api/resources/planner-overview?view=graph-rest-beta)
planner resource type - Microsoft Graph v1.0 - GitHub (https://github.com/microsoftgraph/microsoft-graph-docs-contrib/blob/main/api-reference/v1.0/resources/planner-overview.md)
planner resource type - Microsoft Graph v1.0 (https://learn.microsoft.com/en-us/graph/api/resources/planner-overview?view=graph-rest-1.0)
Plugin - OpenWebUI Docs (https://docs.openwebui.com/features/plugin/)
Postgres vs MySQL - cyberpanel.net (https://cyberpanel.net/blog/postgres-vs-mysql)
PostgreSQL as a Vector Database: Using pgvector - tigerdata.com (https://www.tigerdata.com/blog/postgresql-as-a-vector-database-using-pgvector)
PostgreSQL Extensions: pgvector - tigerdata.com (https://www.tigerdata.com/learn/postgresql-extensions-pgvector)
PostgreSQL Vector Search Guide with pgvector - northflank.com (https://northflank.com/blog/postgresql-vector-search-guide-with-pgvector)
PostgreSQL vs MySQL comparison 2024 2025 - bytebase.com (https://www.bytebase.com/blog/postgres-vs-mysql/)

PostgreSQL vs. MySQL - flatirons.com (https://flatirons.com/blog/postgresql-vs-mysql/)
PostgreSQL vs. MySQL: Which One Is Better For Your Use Case? - integrate.io (https://www.integrate.io/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/)
Python PDF Generation from HTML with WeasyPrint - dev.to (https://dev.to/bowmanjd/python-pdf-generation-from-html-with-weasyprint-538h)
Python Web Scraping Projects - firecrawl.dev (https://www.firecrawl.dev/blog/python-web-scraping-projects)
python-pptx - github.com (https://github.com/scanny/python-pptx)
python-pptx - products.documentprocessing.com (https://products.documentprocessing.com/editor/python/python-pptx/)
python-pptx - pypi.org (https://pypi.org/project/python-pptx/)
python-pptx - python-pptx.readthedocs.io (https://python-pptx.readthedocs.io/en/latest/index.html)
Qdrant - Vector Database for AI - qdrant.tech (https://qdrant.tech/)
Qdrant - aws.amazon.com (https://aws.amazon.com/marketplace/pp/prodview-rtphb42tydtzg)
Qdrant Vector Database - qdrant.tech (https://qdrant.tech/qdrant-vector-database/)
qdrant - github.com (https://github.com/qdrant/qdrant)
RAG and embeddings: is it better to embed text with labels or not? - community.openai.com (https://community.openai.com/t/rag-and-embeddings-is-it-better-to-embed-text-with-labels-or-not/604100)
RAG vs. GraphRAG: A Comprehensive Comparison for Question Answering - arxiv.org (https://arxiv.org/abs/2502.11371)
RAG vs. GraphRAG: A Comprehensive Comparison for Question Answering - arxiv.org (https://arxiv.org/html/2502.11371v1)
Report on all Planner data in a tenant using PowerShell - Practical 365 (https://practical365.com/report-planner-data/)
Resources for Advanced Document Processing - cwiki.apache.org (https://cwiki.apache.org/confluence/display/TIKA/Resources+for+Advanced+Document+Processing)
Retrieval Augmented Generation (RAG) - docs.mistral.ai (https://docs.mistral.ai/guides/rag/)
Retrieval-Augmented Generation (RAG) - promptingguide.ai (https://www.promptingguide.ai/research/rag)
Retrieving attachments from Microsoft Planner tasks through Graph API - Stack Overflow (https://stackoverflow.com/questions/47940869/retrieving-attachments-from-microsoft-planner-tasks-through-graph-api)
Run Neo4j in a Docker container - development.neo4j.dev (https://development.neo4j.dev/developer/docker-run-neo4j/)
Running multi-container applications - docs.docker.com (https://docs.docker.com/get-started/docker-concepts/running-containers/multi-container-applications/)
SDKs Overview - Microsoft Graph (https://learn.microsoft.com/en-us/graph/sdks/sdks-overview)
Scrapy vs. Playwright - brightdata.com (https://brightdata.com/blog/web-data/scrapy-vs-playwright)
Set up notifications for changes in user data - Microsoft Graph (https://learn.microsoft.com/en-us/graph/change-notifications-delivery-webhooks)
Simple Graph Database Setup with Neo4j and Docker Compose - medium.com (https://medium.com/@matthewghannoum/simple-graph-database-setup-with-neo4j-and-docker-compose-061253593b5a)
TeamsBridge GitHub Repository - amoreng (https://github.com/amoreng/TeamsBridge)
Technology Stack for Microservices - aalpha.net (https://www.aalpha.net/blog/technology-stack-for-microservices/)
The Conversational AI Pipeline - GitHub (https://github.com/kunan-sa/the-conversational-ai-pipeline)
Throttling and batching - Microsoft Graph (https://learn.microsoft.com/en-us/graph/throttling)
Throttling MS Graph API/Sharepoint Online - Reddit (https://www.reddit.com/r/AZURE/comments/16hu3es/throttling_ms_graph_apisharepoint_online/)
throttling-limits.md - GitHub (https://github.com/microsoftgraph/microsoft-graph-docs-contrib/blob/

main/concepts/throttling-limits.md)
Tips for using WeasyPrint for PDF generation - reddit.com (https://www.reddit.com/r/Python/comments/jmfbwk/tips_for_using_weasyprint_for_pdf_generation/)
To Do API Overview - Microsoft Graph (https://learn.microsoft.com/en-us/graph/todo-concept-overview)
Tools - OpenWebUI Community (https://openwebui.com/tools)
Tools - OpenWebUI Docs (https://docs.openwebui.com/features/plugin/tools/)
Tools Development - OpenWebUI Docs (https://docs.openwebui.com/features/plugin/tools/development/)
Top 10 Open-Source Graph Databases - index.dev (https://www.index.dev/blog/top-10-open-source-graph-databases)
Top 10 Open-Source RAG Evaluation Frameworks You Must Try - medium.com (https://walseis-arel.medium.com/top-10-open-source-rag-evaluation-frameworks-you-must-try-15c40d0ba2c0)
Top 10 Tools for Web Scraping - firecrawl.dev (https://www.firecrawl.dev/blog/top_10_tools_for_web_scraping)
Top 10 Ways to Generate PDFs in Python - nutrient.io (https://www.nutrient.io/blog/top-10-ways-to-generate-pdfs-in-python/)
Top 5 Open-Source Web Scraping Tools for Developers - firecrawl.dev (https://www.firecrawl.dev/blog/top-5-open-source-web-scraping-tools-for-developers)
Top Unstructured Data Tools - wajidkhan.info (https://wajidkhan.info/top-unstructured-data-tools/)
Troubleshoot Microsoft Graph authorization errors - Microsoft Graph (https://learn.microsoft.com/en-us/graph/resolve-auth-errors)
Tutorial: Deploy a multi-container group using Docker Compose - learn.microsoft.com (https://learn.microsoft.com/en-us/azure/container-instances/tutorial-docker-compose)
Unstructured Answer: Structured vs. Unstructured Data in AI - restack.io (https://www.restack.io/p/unstructured-answer-structured-vs-unstructured-cat-ai)
Unstructured Data, Apache Tika, and Beer - dankeeley.wordpress.com (https://dankeeley.wordpress.com/2016/05/18/unstructured-data-apache-tika-and-beer/)
Unstructured is an ETL solution for LLMs - news.ycombinator.com (https://news.ycombinator.com/item?id=36616799)
unstructured - github.com (https://github.com/Unstructured-IO/unstructured)
Use Docker Compose to deploy multiple containers - learn.microsoft.com (https://learn.microsoft.com/en-us/azure/ai-services/containers/docker-compose-recipe)
Use GitHub with Loop - Microsoft Support (https://support.microsoft.com/en-us/office/use-github-with-loop-5a4d95d5-3c59-4de8-a208-c9c8ab05a4fb)
Use the API - Microsoft Graph (https://learn.microsoft.com/en-us/graph/use-the-api)
Use the Planner REST API - Microsoft Graph (https://learn.microsoft.com/en-us/graph/planner-concept-overview)
Use the pgvector extension in Azure Database for PostgreSQL - Flexible Server - learn.microsoft.com (https://learn.microsoft.com/en-us/azure/postgresql/flexible-server/how-to-use-pgvector)
Using Awesome-Compose to Build and Deploy Your Multi-Container Application - docker.com (https://www.docker.com/blog/using-awesome-compose-to-build-and-deploy-your-multi-container-application/)
Using PostgreSQL over MySQL in 2024 - reddit.com (https://www.reddit.com/r/PostgreSQL/comments/1e4wq0p/using_postgresql_over_mysql_in_2024/)
vercel/ai-chatbot - GitHub (https://github.com/vercel/ai-chatbot)
What are the differences between LangChain and other LLM frameworks like LlamaIndex or Haystack? - milvus.io (https://milvus.io/ai-quick-reference/what-are-the-differences-between-langchain-and-other-llm-frameworks-like-llamaindex-or-haystack)
What is a microservice architecture? - contentful.com (https://www.contentful.com/resources/microservice-architecture/)
What is a Microservice Tech Stack? - hygraph.com (https://hygraph.com/blog/what-is-microservice-tech-

stack)
What is a vector database? — Qdrant - medium.com (https://medium.com/@qdrant/what-is-a-vector-database-qdrant-fb8cd9b3b524)
What is pgvector? - enterprisedb.com (https://www.enterprisedb.com/blog/what-is-pgvector)
What is the best scraper tool right now? Firecrawl, Scrapy, etc. - reddit.com (https://www.reddit.com/r/LocalLLaMA/comments/1jw4yqv/what_is_the_best_scraper_tool_right_now_firecrawl/)
wokelo-docs - pypi.org (https://pypi.org/project/wokelo-docs/)
Working with Docker Compose - code.visualstudio.com (https://code.visualstudio.com/docs/containers/docker-compose)
anyone_know_what_the_long_term_trend_between - reddit.com (https://www.reddit.com/r/PostgreSQL/comments/1f8ta5p/anyone_know_what_the_long_term_trend_between/)