**MMA 865: Individual Assignment – Part 2**

**Jose Chua**

**January 29, 2025**

**Moez Ali**

**Section 1: Problem Statement**

During my time working on BMO’s API Delivery Team, Digital Core, I have realized how much our team relies on effective documentation to support our delivery efforts. While this may seem like something simple to manage, our current documentation is unorganized, difficult to access, and scattered across multiple platforms — making it challenging to find the information we need to work efficiently. This applies to all types of documentation which we may require such as code documentation, process workflows, data dictionaries, instructions, and meeting minutes, all of which are critical to ensuring smooth API development, integration, and deployment.­ ­­­­­

Currently, our documentation can be found in multiple locations such as Confluence, SharePoint, Git Repositories and even Excel sheets stored on One Drive. These documents are also stored with no standardized structure or indexing system. As a result, we spend valuable time searching for key information, leading to delays, miscommunication, and duplicated work. Code documentation is often buried in repositories without clear linkage to relevant processes, while data dictionaries are incomplete or outdated, making it difficult to understand API dependencies.

Similarly, deployment procedures, testing frameworks, and tool instructions are scattered, outdated, or unclear, leading to inefficiencies and mistakes. Meeting minutes and action items are inconsistently recorded, making it difficult to track decisions, ultimately resulting in repeated discussions and rework.

If we were to solve this issue, we would improve efficiency, reduce delays, and improve onboarding. A centralized, structured documentation system would streamline knowledge sharing and decision-making, enabling faster API deliveries. If we were to continue down our current path and not implement a solution, we would risk continued inefficiencies, slower development cycles, and disorganized onboarding, ultimately impacting the team's ability to deliver high-quality APIs on time.

**Section 2: Proposed Solution**

To address these problems a Retrieval-Augmented Generation (RAG) System can be implemented. A RAG system is AI-powered approach that combines a search engine and generative AI to provide accurate and context-aware responses.

Workflow

First, we will centralize all documentation by indexing content from Confluence, SharePoint, Git repositories, and local files into a unified knowledge base. This will be done using vector embeddings, which transform text into searchable numerical representations, allowing us to perform efficient and intelligent information retrieval.

Next, when a team member performs a search — whether it's deployment procedures, API dependencies, or meeting notes — our system will first retrieve the most relevant documents using a smart search engine. Then, a generative AI model will summarize and reformat the information, providing clear and concise answers instead of just dumping raw documents.

To maintain accuracy, real time updates will be performed by continuously re-indexing new content and flagging outdated documentation. A feedback component must also be added to allow users to rate responses. Feedback systems are used on almost all BMO internal products to ensure quality outputs.

By integrating the RAG system, we will reduce the time spent on searching for information, reduce miscommunication, and prevent duplication of work. Furthermore, structured, easy-to-access documentation will make onboarding team members and knowledge sharing seamless and accelerate API development. This solution ensures that our documentation remains a living, intelligent resource, empowering teams to work smarter, not harder.

**Section 3: Quantifying Impact**

1. Time Savings:
   * Currently, employees spend 15% of their workweek (6 hours per week) searching for or recreating documentation.
   * With the RAG system, A 50% reduction in search time is expected.
   * Each team member saves 3 hours per week → For a team of 50, that’s 7,800 hours annually (50 × 3 × 52).
2. Productivity Gains:
   * Assuming an average hourly salary of $60, the time savings translate to $468,000 per year in recovered productivity (7,800 hours × $60).
3. Fewer Errors & Rework:
   * Miscommunication and outdated documentation contribute to 10% of API development delays (e.g., debugging incorrect implementations, redeployments).
   * A structured system reduces these issues by 30%, leading to faster API rollouts and fewer costly errors.
4. Faster Decision-Making:
   * Meeting follow-ups and action tracking improve, reducing repeated discussions by 20%.
   * This results in more streamlined project timelines and better alignment across teams.

Key Assumptions

* Employees work 40 hours per week.
* Search inefficiencies consume 15% of work time.
* 10% of delivery delays are attributed to outdated documents and miscommunication.
* The AI system achieves at least 50% improvement in retrieval speed.
* The team consists of 50 members.
* The hourly cost of an employee is estimated at $60.

Conclusion

By reducing search time, preventing duplicate work, and improving documentation accuracy, we expect an annual productivity gain of nearly $500K, alongside faster API delivery and improved collaboration.

**Section 4: Implementation**

To implement the RAG system, we will start with collecting and centralizing documentation by indexing content from Confluence, SharePoint, Git repositories, and local files. This includes process workflows, API specifications, deployment procedures, meeting notes, and data dictionaries. Data preprocessing will involve text extraction, removing any duplicates, metadata tagging (author, timestamp, source), and transforming text into vector embeddings for efficient retrieval.

A RAG pipeline can be utilized to complete our modeling. The retrieval component will use a vector database with an embedding model (e.g., OpenAI, BERT) to find relevant documents. The generation component will leverage LLMs (e.g., GPT-4, Llama) to summarize, reformat, and provide structured responses. To keep the information relevant and accurate, we’ll refine the search process and prioritize results based on document details and team feedback.

To measure model performance, we will track retrieval accuracy and response quality (human evaluation). We can also compare different embedding models and LLMs based on latency, accuracy, and usability. The final model will be selected based on the highest retrieval precision and lowest hallucination rate while continuous feedback from users will allow iterative improvements, ensuring the system remains accurate and efficient over time.

By streamlining documentation access with AI, we’ll save time, reduce errors, and improve collaboration—empowering our team to deliver high-quality work faster and more efficiently.