Ticket BQGP-291🧠 Prototyping Graph RAG

Comment:

- Designed and implemented the core **Retrieval-Augmented Generation (RAG)** architecture.
- Defined the pipeline stages: document ingestion → chunking → embedding → indexing → retrieval → LLM response generation.
- Validated query-to-vector workflows and ensured compatibility with Vertex AI's Matching Engine.
- Setup project-level resources and initialized GCP environment using `aiplatform.init()`.

Ticket BQGP-309📄 Chunking, Embedding and Vector Storage for Prototyping Graph RAG

Comment:

- Extracted structured text from the Forex Market Screener PDF using **PyMuPDF** (`fitz`).
- Split content into semantically coherent chunks using **LangChain's RecursiveCharacterTextSplitter** (chunk size: 512, overlap: 25).
- Embedded each chunk using Vertex AI `text-embedding-005`.
- Indexed embeddings with metadata into **Vertex AI Matching Engine**, allowing fast ANN search.
- Verified by querying index manually with test vectors.

Ticket BQGP-310🖥️ Implementing Front End UI for ChatBot

Comment:

- Built a **Streamlit**-based chatbot UI to enable real-time query input and response output.
- UI includes textbox input, response display area, and status feedback for errors or loading.
- Connected to backend Python logic to trigger end-to-end retrieval and generation on submit.
- Included debug sections to test embedding and nearest neighbor match interactively.

Ticket BQGP-311🔗 Connecting the ChatBot UI with Graph RAG implementation

Comment:

- Integrated frontend and backend through structured function calls from Streamlit UI to embedding, Matching Engine, and Gemini model.
- Passed query through embedding → vector search → chunk ID resolution → prompt injection → LLM response generation.
- Validated retrieval by printing nearest chunk ID and manually verifying relevance.
- Ensured data flows via `get_answer()` and modularized embedding and matching components.

Ticket BQGP-312⚙️ Improving Performance ChatBot/ Agent

Comment:

- Introduced **fallback handling**: if no chunk is returned, Gemini provides a generic or summarized answer.
- Refined prompt formatting to ensure better alignment with chunk context and user question.
- Reduced LLM hallucinations by scoping context using `datapoint_id`-driven lookups.

- Tested on multiple edge-case queries to measure improvement in precision and relevance.

- Tuned neighbor count, tokenizer handling, and embedding configuration for performance gains.