Assignment 1:

Design a high-level architecture diagram for a Retrieval-Augmented Generation (RAG) pipeline powering a Slack bot that answers queries using:

Code from GitHub repositories
Documentation stored in Obsidian (files available in Google Drive)
The diagram should clearly illustrate the data flow and system components involved.

For Assignment 1, there is no need to include code—focus only on the high-level system design.

Assignment 2:

File Handling REST API

We are looking for a solution that creates a robust and secure API manage file transfers between remote devices and our server. Your assignment is to build this API (HTTP protocol) using FastAPI, ensuring it meets the following business needs and technical requirements.

Business Needs

- **Reliable File Uploads:**
  Devices should be able to send files in small pieces. The system must combine these pieces to rebuild the original file. If an upload is interrupted, the system must save what has been received and let the device resume from where it left off.
- **Secure and Efficient File Downloads:**
  The API must support file downloads with the ability to download specific parts (partial downloads). This feature is essential for efficient handling of large files.
- **Real-Time File Status Monitoring:**
  There must be a mechanism to check whether a file is fully received, partially received, or pending. This will inform devices where to resume an interrupted upload.
- **Continuous Maintenance and Cleanup:**

The system should regularly review file transfers. Any incomplete uploads that haven't been updated for a defined period should be saved permanently, and unused data should be cleared to free resources.

- **Security:**
  All interactions with the API must be secure. Only authorized devices should be allowed to upload or download files.

Assumptions

- File chunks will include a custom binary header containing the start byte, end byte, and a checksum (computed as the sum of the bytes modulo 256) for verifying data integrity.
- Devices may experience intermittent connectivity. The API should support resumable uploads by tracking the last received byte.
- Partial uploads are persisted to disk when necessary, and the design should allow for potential integration with cloud storage solutions in the future.

Technical Requirements

- **Framework:**
  Build the API using FastAPI, taking advantage of its asynchronous capabilities to efficiently manage I/O operations.
- **File Chunk Upload:**
  Validate the custom header in each file chunk, verify the checksum, Once all pieces are received, assemble and save the complete file. Handle interruptions gracefully by persisting partial uploads.
- **Partial File Download:**
  Create an endpoint that supports the HTTP Content-Range header. This will allow clients to request and stream specific portions of a file via FastAPI's StreamingResponse.
- **File Status Monitoring:**
  Develop an endpoint to report the current state of a file transfer, indicating whether the file is complete, partially stored (in memory or on disk), or pending, along with the next expected byte for resuming uploads.
- **Background Cleanup Process:**
  Implement a background process that periodically checks for file chunks that haven't been updated within a specified time frame. This process should persist incomplete files to disk and free up any stale data from memory.
- **Security:**
  Use token-based authentication (for example, JWT Bearer) to ensure that only authorized devices can interact with the API.

Evaluation Criteria

Your solution will be evaluated based on the following criteria:

- **Reliability:**
  Does the API consistently handle file uploads and downloads without data loss or corruption?
- **Data Integrity:**
  Is checksum validation implemented correctly to ensure that files are accurately reconstructed?
- **Efficiency:**
  Are asynchronous operations and resource management (including background cleanup tasks) handled effectively?
- **Security:**
  Is the API properly secured so that only authenticated devices can access its endpoints?
- **Code Quality:**
  Is the code well-organized, modular, and clearly documented? Does it follow best practices in modern Python development?

Your submission should include the complete FastAPI application source code along with a README file detailing your setup instructions, design decisions, and any assumptions you made during development. We look forward to reviewing your solution. Good luck!