Distributed Financial Data Processing

Objective

The goal of this exercise is to assess your ability to design and implement a distributed system using a message broker (**RabbitMQ**), integrate with an external API (**ChatGPT API**), process financial data, and store normalized results in **MongoDB**. This exercise will test your skills in **asynchronous messaging, data extraction, transformation, and API integration**, as well as your ability to structure scalable and maintainable code.

Scenario

You are building a distributed financial data processing pipeline. Your system will process **unstructured financial reports** and extract structured information using OpenAI's ChatGPT API. The processed and normalized financial data will be stored in MongoDB.

Your system will consist of:

1. **A Producer Service** – Accepts raw financial data and sends it to a RabbitMQ queue.
2. **A Worker Service** – Consumes messages from the queue, extracts structured financial data using the ChatGPT API, normalizes it, and stores it in MongoDB.
3. **Database Storage** – Stores the extracted and normalized financial data.

Technical Requirements

- **Python** for implementation.
- **RabbitMQ** as the message broker.
- **FastAPI** or **Flask** for the producer service.
- **Pika** for interacting with RabbitMQ.
- **OpenAI's ChatGPT API** for financial data extraction.
- **Pymongo** for MongoDB interaction.
- **Logging** and **Exception Handling** for robustness.
- **Docker (Optional)** for containerization.

Example Input (Unstructured JSON)

```
Unset
{
  "raw_text": "Company XYZ reported a net income of $5.3 million
for Q1 2024."
}
```

Expected Output (Normalized JSON Stored in MongoDB)

```
Unset
{
  "company": "XYZ",
  "metric": "net income",
  "value": 5300000,
  "currency": "USD",
  "quarter": "Q1 2024"
}
```

Implementation Steps

1. Producer Service (FastAPI or Flask)

- Create a **REST API endpoint** (e.g., `/submit`) that accepts a JSON payload.
- Publish the received message to a **RabbitMQ queue**.

2. Worker Service

- Listen for messages from the RabbitMQ queue.
- Extract financial details using **OpenAI's ChatGPT API**.
- **Normalize** the extracted data (e.g., converting “5.3 million” to `5300000`).
- Store the **structured financial data in MongoDB**.

3. Database Storage (MongoDB)

- Connect to a MongoDB database.

- Store the structured financial data in a collection called `financial_data`.

Bonus Features

- **Containerization**: Dockerize the services.
- **Unit Testing**: Add test cases for key functionalities.
- **Scalability Considerations**: Allow multiple worker instances to process messages in parallel.
- **Data Validation**: Ensure input/output meets predefined financial data standards.

Evaluation Criteria

- **Code Structure & Readability**: Clean and modular code.
- **Messaging System Implementation**: Effective use of RabbitMQ.
- **API Integration**: Correct use of the ChatGPT API.
- **Data Processing Accuracy**: Proper financial data extraction and normalization.
- **Database Interaction**: Correctly storing structured data in MongoDB.
- **Logging & Error Handling**: Robust handling of failures.

Submission Guidelines

- Estimated time to complete: **2 hours**.
- The code must be submitted in a **public Git repository** (e.g., GitHub, GitLab, or Bitbucket).
- Ensure your repository includes a **README.md** explaining:
    - How to set up and run the project.
    - API endpoints and example requests.
    - Any assumptions or enhancements you made.
- Provide a Loom video showcasing your system running locally.