🧠 ✨ Multi-Modal RAG System

📌 Overview

The Multi-Modal RAG (Retrieval Augmented Generation) System is designed to provide intelligent, context-aware responses by combining structured data, documents, and external tools with large language models (LLMs).

This system leverages a Django + FastAPI hybrid architecture to ensure:

- Clean separation of responsibilities
- High scalability for AI workloads
- A stable and secure API layer for frontend consumers

Django acts as the API gateway and middleware, while FastAPI powers the AI and integration logic such as Jira data processing and RAG pipelines.

🏗️ High-Level Architecture

🧑‍💻 React (User Input)

⬇️

🛡️ Django API (Middleware / Gateway)

⬇️

⚡ FastAPI (Multi-Modal RAG Engine / Jira)

⬇️

🛡️ Django (Response Relay)

⬇️

📊 React (Response Display)

🔑 Key Design Principles

- 🔗 Loose coupling between services

- 🧩 Modular and extensible backend
- 🚫 Frontend unaware of FastAPI internals
- ⚡ Optimized AI processing layer

🧩 System Components & Responsibilities

🎨 Frontend (React)

- Captures user queries (text, metadata, filters)
- Communicates only with Django APIs
- Renders AI-generated responses

🛡️ Django (API Gateway & Middleware)

- Acts as a single entry point for frontend requests
- Handles:
  - Authentication & validation 🔐
  - Request forwarding ➡️
  - Response normalization 📦
- Decouples frontend from FastAPI services

⚡ FastAPI (Multi-Modal RAG Engine)

- Executes heavy and async workloads
- Integrates with:
  - 🧾 Jira REST APIs
  - 📚 Document stores
  - 🧠 Vector databases
  - 🤖 LLMs

- Returns structured responses to Django

👤 Assigned Responsibility

🧑‍💻 Developer: Manoj

Manoj is responsible for implementing Django middleware APIs that proxy requests to FastAPI and relay responses back to the frontend.
These endpoints act as clean abstraction layers and ensure future flexibility.

🔌 Django API Endpoints

📊 /api/jira-data/

Purpose
Provides Jira-related insights through a clean Django API.

Request Flow

1. React sends request to Django
2. Django forwards request to FastAPI
3. FastAPI fetches data from Jira REST API
4. Processed data is returned to Django
5. Django relays response to React

FastAPI Responsibilities

- 🔐 Jira authentication
- 🧾 Fetching issues, projects, sprints
- 📊 Data transformation & aggregation

🧠 /api/rag-query/

Purpose
Handles AI-powered knowledge queries using Multi-Modal RAG.

Request Flow

1. React submits a query to Django
2. Django forwards query to FastAPI
3. FastAPI executes the RAG pipeline:
    - 🔍 Retrieval (documents, Jira, knowledge base)
    - 🧠 Context enrichment
    - 🤖 LLM inference
4. Response returned to Django
5. Django relays answer to React

FastAPI Responsibilities

- 📚 Document & metadata retrieval
- 🧮 Vector similarity search
- 🧠 LLM response generation

🌟 Why Multi-Modal RAG?

- 📄 Combines structured & unstructured data
- 🧠 Produces grounded, context-aware answers
- 🔄 Reduces hallucinations
- 📈 Improves answer relevance and accuracy

✅ Architectural Benefits

- 🧩 Clean separation of concerns
- 🔐 Secure and stable API gateway
- ⚡ High-performance AI processing
- 🔧 Easy to extend with new tools or data sources
- 🧩 Frontend remains backend-agnostic

🚀 Future Enhancements

- 🖼️ Image & file-based RAG inputs
- 🔊 Audio / speech-to-text support
- 📈 Caching and response optimization
- 📜 Request logging and observability
- 🔄 Retry & fallback mechanisms

🧾 Summary

The Multi-Modal RAG System architecture combines the reliability of Django with the performance and flexibility of FastAPI to deliver scalable, AI-powered experiences. This design ensures long-term maintainability while enabling rapid innovation in AI workflows.