# Metis RAG

Metis RAG is an application that combines conversational AI with Retrieval Augmented Generation (RAG) capabilities. It allows users to chat with large language models while providing relevant context from uploaded documents.

## Project Documentation

This repository contains the following key documentation:

* **README.md** (this file): Overview, features, and usage instructions
* [**PLANNING.md**](PLANNING.md): High-level architecture, design decisions, and technical specifications
* [**TASK.md**](TASK.md): Current tasks, priorities, and development roadmap

## Features

* Chat with large language models through Ollama
* Upload and process documents (PDF, TXT, CSV, MD)
* Retrieval Augmented Generation for contextual responses
* User authentication and access control
* Multi-user support with resource ownership
* LLM-enhanced document processing with intelligent chunking strategies
* Advanced query refinement and retrieval enhancement
* Response quality assurance with evaluation and refinement
* Factual accuracy verification and hallucination detection
* Comprehensive audit reports with source tracking
* Document management with tagging and organization
* System monitoring and analytics
* Background Task System for asynchronous processing
* Persistent chat memory across conversation turns
* Explicit memory buffer for storing and recalling user information
* Context-aware memory management with priority-based retrieval
* Responsive UI with light/dark mode
* Enhanced logging and debugging capabilities

## Authentication System

Metis RAG includes a comprehensive authentication system that provides user management and access control:

### User Management

* User registration and login
* Role-based access control (user/admin)
* JWT-based authentication
* Secure password hashing with bcrypt

### Access Control

* Resource ownership validation
* Protected API endpoints
* Route protection middleware
* Multi-user support with data isolation

### Security Features

* Token-based authentication
* Password hashing and validation
* CORS protection
* Security headers
* Route-based access control

For more details, see the [Authentication Documentation](docs/technical/AUTHENTICATION.md).

## LLM-Enhanced RAG System

Metis RAG includes an advanced LLM-enhanced system that improves critical aspects of the RAG pipeline:

### Dynamic Chunking Strategy Selection

The system uses a “Chunking Judge” (an LLM agent) to analyze documents and select the most appropriate chunking strategy and parameters:

* Analyzes document structure, content type, and formatting
* Dynamically selects between recursive, token-based, or markdown chunking strategies
* Recommends optimal chunk size and overlap parameters
* Preserves semantic meaning and document structure
* Adapts to different document types without manual configuration

### Query Refinement and Retrieval Enhancement

The system uses a “Retrieval Judge” (an LLM agent) to improve retrieval quality:

* Analyzes queries and retrieved chunks
* Refines queries to improve retrieval precision
* Evaluates relevance of retrieved chunks
* Re-ranks chunks based on relevance to the query
* Requests additional retrieval when necessary

### Response Quality Pipeline

The system includes a comprehensive response quality pipeline to ensure high-quality responses:

* **Response Synthesis**: Combines retrieval results and tool outputs into coherent responses with proper source attribution
* **Response Evaluation**: Assesses factual accuracy, completeness, relevance, and checks for hallucinations
* **Response Refinement**: Iteratively improves responses based on evaluation results
* **Audit Reporting**: Generates comprehensive audit reports with information source tracking and verification status

### Conversation Memory

The system maintains persistent memory across conversation turns and browser sessions:

* **Context Augmentation**: Includes conversation history in the LLM context for more coherent multi-turn dialogues
* **Memory-Aware Query Analysis**: Analyzes queries in the context of previous conversation turns
* **Memory-Enhanced Planning**: Creates query execution plans that consider conversation history
* **Contextual Response Generation**: Generates responses that maintain continuity with previous exchanges
* **Explicit Memory Commands**: Supports natural language commands for storing and retrieving specific information
  + Remember this: [information] - Explicitly store information in memory
  + Recall [topic] - Explicitly retrieve information from memory
* **Implicit Memory Storage**: Automatically stores all user queries for future reference
* **Implicit Memory Retrieval**: Recognizes and responds to implicit memory queries like “What is my favorite color?”
* **Memory Categories**: Supports common memory categories like favorite colors, foods, names, etc.
* **Persistent Memory Storage**: Stores memories in the database for long-term retention across sessions
* **Memory Diagnostics**: Provides tools for debugging and monitoring memory functionality
* **Session Persistence**: Maintains conversation context across page refreshes and browser sessions

### Quality Assurance Features

* Factual accuracy scoring and verification
* Hallucination detection and removal
* Source attribution and citation tracking
* Iterative refinement until quality thresholds are met
* Comprehensive audit trails for transparency and accountability
* LLM-based process analysis for continuous improvement

These enhancements make the RAG system more adaptable to different document types and query styles, while ensuring responses are accurate, complete, relevant, and free from hallucinations. ## Architecture

Metis RAG is built with the following technologies:

* **Backend**: FastAPI, Python
* **Authentication**: JWT-based authentication with bcrypt password hashing
* **Database**: PostgreSQL with SQLAlchemy ORM
* **Vector Database**: ChromaDB with optimized caching
* **LLM Integration**: Ollama API with enhanced prompt engineering
* **Document Processing**: LangChain with LLM-enhanced dynamic chunking strategies
* **Response Quality**: Comprehensive pipeline for synthesis, evaluation, refinement, and auditing
* **Workflow Orchestration**: LangGraph for adaptive RAG workflows
* **Process Logging**: Detailed logging and audit trail generation
* **Background Task System**: Asynchronous processing with task prioritization and resource management
* **Testing**: Comprehensive test suite for RAG functionality and response quality
* **Deployment**: Docker

## Getting Started

### Prerequisites

* Docker and Docker Compose
* Ollama (optional, can run in Docker)

### Installation

1. Clone the repository:

* git clone https://github.com/yourusername/metis-rag.git  
  cd metis-rag

1. Create a .env file based on .env.example:

* cp .env.example .env
* Key configuration options include:
  + DEFAULT\_MODEL: The LLM model to use for RAG responses (default: gemma3:12b)
  + CHUNKING\_JUDGE\_MODEL: The model to use for document analysis (default: gemma3:12b)
  + USE\_CHUNKING\_JUDGE: Enable/disable the Chunking Judge (default: True)
  + USE\_RETRIEVAL\_JUDGE: Enable/disable the Retrieval Judge (default: True)
  + ENABLE\_RESPONSE\_QUALITY: Enable/disable the Response Quality Pipeline (default: True)
  + QUALITY\_THRESHOLD: Minimum quality score for responses (default: 8.0)
  + MAX\_REFINEMENT\_ITERATIONS: Maximum number of refinement iterations (default: 2)
  + ENABLE\_AUDIT\_REPORTS: Enable/disable audit report generation (default: True)
  + SECRET\_KEY: Secret key for JWT token generation (required for authentication)
  + ALGORITHM: Algorithm for JWT token generation (default: HS256)
  + ACCESS\_TOKEN\_EXPIRE\_MINUTES: Token expiration time in minutes (default: 30)

1. Build and start the application with Docker Compose:

* docker-compose up -d

1. Access the application at <http://localhost:8000>

### Running Without Docker

1. Create a virtual environment:

* python -m venv venv  
  source venv/bin/activate # On Windows: venv\Scripts\activate

1. Install dependencies:

* pip install -r requirements.txt

1. Run the application:

* uvicorn app.main:app --reload

## Usage

### Chat Interface

The main chat interface allows you to: - Send messages to the model - Toggle RAG functionality - View citations from source documents - Control conversation history - See quality scores for responses - View source attributions and citations

### Document Management

The document management page allows you to: - Upload documents - Process documents for RAG - View document information - Delete documents

### Authentication

The authentication system provides: - User registration and login pages - Secure access to personal resources - Role-based access control - Token-based authentication for API access - Protection of sensitive operations

### System Management

The system page allows you to: - View system statistics - Manage models - Check system health

### Audit Reports

The audit reports page allows you to: - View comprehensive audit reports for queries - Examine information sources used in responses - Verify factual accuracy and completeness - Track reasoning traces and execution timelines - Identify potential hallucinations - Export audit reports for compliance and transparency

### Background Task System

The Background Task System allows you to: - Monitor and manage asynchronous tasks - View task status, progress, and results - Prioritize tasks based on importance - Schedule tasks for future execution - Define task dependencies for complex workflows - Monitor system resource usage - View resource alerts and system health

## Development

### Project Structure

metis\_rag/  
├── app/  
│ ├── api/ # API endpoints  
│ │ ├── auth.py # Authentication endpoints  
│ │ └── ... # Other API endpoints  
│ ├── core/ # Core configuration  
│ │ ├── security.py # Authentication and security  
│ │ └── ... # Other core modules  
│ ├── middleware/ # Middleware components  
│ │ └── auth.py # Authentication middleware  
│ ├── db/ # Database components  
│ │ ├── repositories/ # Repository pattern implementations  
│ │ │ ├── user\_repository.py # User data access  
│ │ │ └── ... # Other repositories  
│ │ └── ... # Other database modules  
│ ├── rag/ # RAG engine  
│ │ ├── agents/ # LLM-based agents for RAG enhancement  
│ │ ├── response\_synthesizer.py # Response synthesis  
│ │ ├── response\_evaluator.py # Response evaluation  
│ │ ├── response\_refiner.py # Response refinement  
│ │ ├── audit\_report\_generator.py # Audit reporting  
│ │ ├── response\_quality\_pipeline.py # Quality pipeline  
│ │ └── langgraph\_states.py # LangGraph state models  
│ ├── tasks/ # Background Task System  
│ │ ├── task\_manager.py # Task management  
│ │ ├── task\_models.py # Task data models  
│ │ ├── scheduler.py # Task scheduling  
│ │ ├── resource\_monitor.py # Resource monitoring  
│ │ ├── task\_repository.py # Task database operations  
│ │ └── example\_tasks.py # Example task handlers  
│ ├── models/ # Data models  
│ │ ├── user.py # User models  
│ │ └── ... # Other models  
│ ├── static/ # Static files  
│ │ ├── js/ # JavaScript files  
│ │ │ ├── main.js # Authentication JS  
│ │ │ └── ... # Other JS files  
│ │ └── ... # Other static files  
│ ├── templates/ # HTML templates  
│ │ ├── login.html # Login page  
│ │ ├── register.html # Registration page  
│ │ └── ... # Other templates  
│ └── utils/ # Utility functions  
├── tests/ # Tests  
│ ├── unit/ # Unit tests  
│ │ └── test\_response\_quality.py # Response quality tests  
│ └── integration/ # Integration tests  
│ └── test\_response\_quality\_integration.py # Integration tests  
├── uploads/ # Uploaded documents  
├── chroma\_db/ # ChromaDB data  
└── docker-compose.yml

### Running Tests

Run tests with pytest:

pytest

For testing specific components:

# Test RAG retrieval functionality  
python test\_rag\_retrieval.py  
  
# Test response quality components  
pytest tests/unit/test\_response\_quality.py  
  
# Test response quality integration  
pytest tests/integration/test\_response\_quality\_integration.py  
  
# Test background task system  
python scripts/test\_background\_tasks.py  
  
# Test authentication system  
python scripts/test\_authentication.py  
  
# Test memory functionality  
python scripts/test\_memory\_functionality.py

The memory functionality test script verifies: - Creating users and conversations - Storing explicit memories using the “Remember this” command - Storing implicit memories from all user queries - Processing queries with explicit recall commands - Processing queries with implicit memory-related questions - Retrieving memories with search terms - Retrieving memories with specific labels

These test scripts create test documents, process them, and test various aspects of the system: - RAG retrieval tests verify that the system correctly retrieves and uses information from documents - Response quality tests verify that responses are accurate, complete, relevant, and free from hallucinations - Integration tests verify that all components work together correctly in end-to-end workflows - Background task tests verify that the task system correctly handles task submission, execution, prioritization, and resource management - Authentication tests verify user registration, login, token generation, and access control for resources

## License

[MIT License](LICENSE)

## Acknowledgements

This project builds upon: - Metis\_Chat - rag\_ollama\_mvp