AI Document QA System

A professional, production-ready document-based question answering system built with FastAPI, LangChain, Google AI, and FAISS. Features real-time streaming responses, semantic search, and a comprehensive REST API.

Features

Core Functionality

- **Document Upload & Processing**: Support for PDF, TXT, and Markdown files
- **Semantic Search**: Vector-based similarity search using Google/OpenAI embeddings
- **AI-Powered QA**: Answer generation using Google Gemini 2.5 Flash with LangChain
- **Real-time Streaming**: Live response streaming with Server-Sent Events
- **RESTful API**: Complete CRUD operations with OpenAPI documentation

Advanced Features

- **LangChain Integration**: Advanced prompt management and output parsing
- **Multi-provider Support**: Google AI (primary) with OpenAI fallback
- **Intelligent Caching**: File-based embedding cache for performance
- **Authentication**: JWT-based security with Bearer tokens
- **Document Management**: Upload, list, view, and delete documents
- **Health Monitoring**: Comprehensive service health checks
- **Built-in Web UI**: Professional interface for testing and interaction

Technical Highlights

- **Async/Await**: Full asynchronous processing throughout
- **Vector Storage**: FAISS for efficient similarity search
- **Error Handling**: Comprehensive exception management and logging
- **Production Ready**: Docker support, logging, and monitoring
- **Scalable Architecture**: Modular service-oriented design

Quick Start

Prerequisites

- Python 3.11+
- Google AI API Key, get it from: https://aistudio.google.com/app/apikey
- OpenAI API Key (optional)

Installation

1. **Clone the repository**

```
git clone https://github.com/nabibukhsh-ai/ai-document-qa-system.git
cd ai-document-qa-system
```

2. **Create virtual environment**

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

3. **Install dependencies**

```
pip install -r requirements.txt
```

4. **Set up environment variables**

```
cp .example.env .env
# Edit .env with your API keys
```

5. **Run the application**

```
python -m app.main
```

The Interface will be available at `http://localhost:8000` or 127.0.0.1:8000
The Swagger will be available at `http://localhost:8000/docs` or 127.0.0.1:8000/docs

Project Structure

```
ai-document-qa-system/
   app/
       __init__.py
       main.py                 # FastAPI application entry point
       api/
           __init__.py
           dependencies.py     # Dependency injection
           routes/
               __init__.py
               documents.py    # Document CRUD endpoints
               query.py        # QA and search endpoints
       core/
           __init__.py
           config.py          # Application configuration
           security.py        # Authentication & JWT handling
       models/
           __init__.py
           schemas.py         # Pydantic models
       services/
           __init__.py
```

```
        document_service.py    # Document management logic
        embedding_service.py   # LangChain embeddings integration
        vector_store_service.py # FAISS vector operations
        llm_service.py         # LangChain LLM integration
    utils/
        __init__.py
        document_processor.py  # File parsing & text chunking
        exceptions.py          # Custom exception classes
data/                      # Vector store and cache (created at runtime)
uploads/                   # Temporary file uploads (created at runtime)
requirements.txt
.env.example
.gitignore
README.md
```

Environment Configuration

Create a `.env` file with the following variables:

```
# Required API Keys
GOOGLE_API_KEY=your_google_api_key_here
OPENAI_API_KEY=your_openai_api_key_here  # Optional fallback

# Security
SECRET_KEY=secret-key

# Provider Selection
EMBEDDING_PROVIDER=google  # or "openai"
LLM_PROVIDER=google       # or "openai"

# Optional Configuration
CHUNK_SIZE=1000
CHUNK_OVERLAP=200
MAX_FILE_SIZE=52428800
ACCESS_TOKEN_EXPIRE_MINUTES=30
```

API Documentation

Authentication

Get access token (default credentials: `admin`/`secret`):

```
curl -X POST "http://localhost:8000/api/v1/token" \
     -H "Content-Type: application/x-www-form-urlencoded" \
     -u "admin:secret"
```

Document Operations

Upload Document

```
curl -X POST "http://localhost:8000/api/v1/documents/upload" \
     -H "Authorization: Bearer <token>" \
     -F "file=@document.pdf"
```

Add Document (JSON)

```
curl -X POST "http://localhost:8000/api/v1/documents/" \
     -H "Authorization: Bearer <token>" \
     -H "Content-Type: application/json" \
     -d '{"title": "Sample Doc", "content": "Document content..."}'
```

List Documents

```
curl -X GET "http://localhost:8000/api/v1/documents/" \
     -H "Authorization: Bearer <token>"
```

Delete Document

```
curl -X DELETE "http://localhost:8000/api/v1/documents/{doc_id}" \
     -H "Authorization: Bearer <token>"
```

Query Operations

Ask Question

```
curl -X POST "http://localhost:8000/api/v1/query/" \
     -H "Authorization: Bearer <token>" \
     -H "Content-Type: application/json" \
     -d '{"question": "What is the main topic?"}'
```

Stream Response

```
curl -X GET "http://localhost:8000/api/v1/query/stream?question=What%20is%20AI&token=<token>
     -H "Accept: text/event-stream"
```

Web Interface

Access the built-in web interface at `http://localhost:8000` for: - Document upload and management - Interactive question answering - Real-time response streaming - Service health monitoring

Key Technologies

- **FastAPI**: Modern, fast web framework for Python APIs
- **LangChain**: Framework for developing LLM applications

- **Google Generative AI**: Embeddings and LLM services
- **FAISS**: Efficient similarity search and clustering
- **Pydantic**: Data validation using Python type annotations
- **JWT**: JSON Web Tokens for authentication
- **Server-Sent Events**: Real-time streaming responses

Architecture Overview

Service Layer

- **DocumentService**: Handles document lifecycle and metadata
- **EmbeddingService**: Manages text embeddings with caching
- **VectorStoreService**: FAISS operations and similarity search
- **LLMService**: AI response generation with structured output

Data Flow

1. **Document Ingestion**: Upload → Parse → Chunk → Embed → Store
2. **Query Processing**: Question → Embed → Search → Context → Generate
3. **Response Delivery**: Answer with source citations and metadata

Performance Features

- **Intelligent Caching**: Embedding cache for repeated queries
- **Batch Processing**: Efficient handling of multiple documents
- **Async Operations**: Non-blocking I/O throughout
- **Rate Limiting**: API protection and provider compliance
- **Error Recovery**: Graceful handling of service failures

Development

Code Quality

```
pip install black isort flake8
black app/
isort app/
flake8 app/
```

Environment Setup

```
# Development with auto-reload
uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload

# Production
gunicorn app.main:app -w 4 -k uvicorn.workers.UvicornWorker
```

Monitoring

Health Checks

- API Health: `GET /api/v1/health`
- Service Health: `GET /api/v1/query/health`
- Document Health: `GET /api/v1/documents/health/status`

Logging

Structured logging with configurable levels:

```
import logging
logging.basicConfig(level=logging.INFO)
```

Contributing

1. Fork the repository
2. Create a feature branch: `git checkout -b feature/amazing-feature`
3. Commit changes: `git commit -m 'Add amazing feature'`
4. Push to branch: `git push origin feature/amazing-feature`
5. Open a Pull Request

License

This project is licensed under the MIT License - see the LICENSE file for details.

Support

For issues, questions, or contributions: - Create an issue in the repository - Check the API documentation at `/docs` - Review examples in the built-in web interface

Changelog

v1.0.0

- Initial release with complete QA functionality
- LangChain integration for embeddings and LLM
- Real-time streaming responses
- Professional web interface
- Comprehensive documentation