# DocuChat Architecture Reference

## 1. Directory Layout Overview

Root Directory: C:\Users\uipek\PycharmProjects\DocuChat\  
│  
├── .venv/ → Local virtual environment  
│  
├── docuchat\_backend/  
│ ├── app/  
│ │ ├── core/ → Configuration, environment variables, constants  
│ │ ├── routers/ → API route definitions (auth, documents, query, health)  
│ │ ├── uploads/ → User-uploaded files (runtime storage)  
│ │ ├── auth.py → JWT auth helper functions  
│ │ ├── database.py → SQLAlchemy engine, Base, and session  
│ │ ├── jobs.py → Background/async tasks (embedding generation, etc.)  
│ │ ├── main.py → Flask app entry point and route registration  
│ │ ├── models.py → ORM models (users, documents, chat history, collections)  
│ │ ├── rag.py → RAG logic: chunking, embedding, retrieval, and LLM query  
│ │ ├── schemas.py → Pydantic request/response models  
│ │ ├── storage.py → File handling and disk management  
│ │ └── \_\_init\_\_.py  
│ │  
│ └── chroma\_db/ → Chroma persistent database (chroma.sqlite3 + embeddings)  
│  
└── project\_structure.txt → Generated file showing folder tree for reference

## 2. Module Responsibilities

• core/config.py:  
 - Central configuration management.  
 - Loads environment variables (JWT\_SECRET, DB\_URL, UPLOAD\_PATH, etc.).  
 - Provides global constants and settings.  
  
• routers/\*.py:  
 - auth.py: /api/auth routes (register, login, logout, reset-password).  
 - documents.py: /api/documents routes (upload, list, delete, summary).  
 - query.py: /api/query routes (document Q&A via RAG).  
 - health.py: /api/health route for system readiness checks.  
 - uploads/: test assets and dev-only OCR materials (not used in production).  
  
• models.py:  
 - SQLAlchemy ORM models: Users, Documents, Collections, ChatHistory.  
 - Relationships handled via foreign keys and lazy loading.  
 - Follows DocuChat database schema from requirements document.  
  
• schemas.py:  
 - Pydantic-style data validation for API input/output.  
 - Ensures type safety and consistency across endpoints.  
  
• database.py:  
 - Initializes SQLAlchemy engine, Base, and SessionLocal.  
 - Centralized connection to PostgreSQL.  
  
• storage.py:  
 - Handles file save, delete, and retrieval operations.  
 - Manages unique filenames and upload directories.  
  
• rag.py:  
 - Handles text extraction, chunking, embedding, and similarity search.  
 - Integrates LangChain + ChromaDB for retrieval.  
 - Generates LLM responses with contextual grounding.  
  
• jobs.py:  
 - Defines long-running or asynchronous processes (e.g., embedding generation post-upload).  
 - Designed for scalability with background workers (e.g., Celery, RQ).  
  
• main.py:  
 - Flask app entry point.  
 - Registers routers, CORS, middleware, and error handlers.  
 - Imports configuration from core/config.py.

## 3. Coding Conventions

• Follow PEP8 naming and spacing rules.  
• Use consistent route prefixes: /api/auth, /api/documents, /api/query, /api/health.  
• Use snake\_case for Python variables and camelCase for JSON keys.  
• Environment variables loaded only through core/config.py.  
• All database transactions handled via SessionLocal context manager.  
• Logging uses Flask logger with JSON output (to be extended later).  
• JWT secrets and tokens never hardcoded.  
• Exception handling delegated to centralized middleware.  
• OCR and text extraction fallbacks handled gracefully in storage/rag modules.

## 4. Extension & Maintenance Rules

• New API endpoints → add to appropriate routers/\*.py file.  
• New database entities → define in models.py and schemas.py together.  
• Heavy or asynchronous tasks → offload to jobs.py or background queue.  
• Configuration changes → always modify core/config.py, never inline.  
• New AI logic or pipeline extensions → expand rag.py or create rag\_utils/.  
• Unit tests should reside in tests/ (to be added in next phase).  
• Temporary test assets must not be committed under routers/uploads/.  
• Use docstrings and comments for all public functions and classes.