# **DocChat AI - Project Documentation**

## **Table of Contents**

1. Project Abstract
2. System Architecture
3. Project Structure
4. Flow Diagrams
5. File Descriptions
6. Setup and Installation
7. API Endpoints
8. Database Schema
9. Security Implementation
10. Troubleshooting

## **Project Abstract**

DocChat AI is an intelligent document question-answering system that allows users to upload documents (PDF, DOCX, DOC, TXT) and engage in conversational interactions with their content. The system uses advanced NLP techniques including document parsing, text chunking, vector embeddings, and OpenAI's GPT models to provide accurate, context-aware responses to user queries.

### **Key Features**

* **Multi-format Document Support**: PDF, DOCX, DOC, TXT
* **Secure User Authentication**: JWT-based authentication with refresh tokens
* **Intelligent Text Processing**: Automatic chunking and OCR capabilities
* **Vector-based Similarity Search**: Semantic search for relevant document sections
* **Conversational AI**: Context-aware responses using OpenAI's GPT models
* **Session Management**: Persistent chat sessions with history
* **Responsive Web Interface**: Modern, mobile-friendly UI

### **Technology Stack**

* **Backend**: Flask (Python)
* **Authentication**: JWT tokens with bcrypt password hashing
* **Database**: MySQL
* **AI/ML**: OpenAI GPT-4 and text-embedding models
* **Document Processing**: PyPDF2, python-docx, OCR capabilities
* **Frontend**: HTML5, CSS3, JavaScript, Bootstrap 5
* **Vector Storage**: Local file-based embeddings storage

## **File Descriptions**

### **Core Application Files**

#### **app2.py - Main Flask Application**

**Purpose**: Central application entry point and route definitions

**Key Responsibilities**:

* Flask app configuration and initialization
* Route definitions for all endpoints
* Component integration (auth, session, document processing)
* Error handling and logging
* Environment variable management

**Key Routes**:

* / - Main dashboard
* /login, /signup - Authentication pages
* /auth/\* - Authentication API endpoints
* /upload - Document upload
* /chat - Chat interface and API
* /api/sessions - Session management

#### **auth.py - Authentication Management**

**Purpose**: Complete user authentication and authorization system

**Key Components**:

* AuthManager class - Core authentication logic
* JWT token generation and validation
* Password hashing with bcrypt
* Database operations for users and refresh tokens
* @token\_required decorator for route protection

**Security Features**:

* Access tokens (1 hour expiry)
* Refresh tokens (30 day expiry)
* Token revocation
* Password strength validation
* SQL injection protection

#### **document\_parser.py - Document Processing**

**Purpose**: Extract and process text from various document formats

**Key Features**:

* Multi-format support (PDF, DOCX, DOC, TXT)
* Text chunking with configurable overlap
* OCR integration for image-based PDFs
* File validation and metadata extraction
* Error handling for corrupted files

**Processing Pipeline**:

1. File format detection
2. Text extraction using appropriate parser
3. Content cleaning and normalization
4. Intelligent chunking with overlap
5. Metadata extraction

#### **vector\_store.py - Vector Embeddings Management**

**Purpose**: Handle document vectorization and similarity search

**Core Functions**:

* Generate embeddings using OpenAI's text-embedding models
* Store embeddings in local file system
* Perform similarity search for relevant chunks
* Manage embedding metadata and indexing
* Optimize search performance

**Search Algorithm**:

* Cosine similarity calculation
* Configurable result count
* Relevance scoring
* Context preservation

#### **session\_manager.py - Session and History Management**

**Purpose**: Manage user sessions and chat history

**Features**:

* Session creation and tracking
* Chat history persistence
* User-session association
* Session cleanup and management
* Last accessed tracking

**Data Management**:

* In-memory session storage
* Chat message threading
* Session metadata tracking
* User access control

#### **chat\_engine.py - AI Conversation Engine**

**Purpose**: Generate intelligent responses using OpenAI's GPT models

**Capabilities**:

* Context-aware response generation
* Conversation history integration
* Relevance scoring
* Token usage optimization
* Error handling and fallbacks

**Response Pipeline**:

1. Query processing
2. Context preparation from relevant chunks
3. Conversation history integration
4. OpenAI API interaction
5. Response post-processing

### **Frontend Files**

#### **templates/login.html - Login Interface**

**Purpose**: User authentication interface

**Features**:

* Responsive login form
* Client-side validation
* Error handling
* Automatic redirection
* Password visibility toggle

#### **templates/signup.html - Registration Interface**

**Purpose**: New user registration

**Features**:

* User registration form
* Real-time validation
* Password strength checking
* Email format validation
* Success/error messaging

#### **templates/index2.html - Main Dashboard**

**Purpose**: Primary user interface after login

**Features**:

* Document upload interface
* Session management
* Drag-and-drop file upload
* Session list with metadata
* File type validation
* Progress indicators

#### **templates/chat.html - Chat Interface**

**Purpose**: Interactive chat interface for document Q&A

**Features**:

* Real-time messaging
* Chat history display
* Auto-scrolling
* Message formatting
* Error handling
* Session information display

#### **app.js - Frontend Utilities**

**Purpose**: Shared JavaScript utilities and functions

**Common Functions**:

* Authentication helpers
* API request wrappers
* UI utility functions
* Error handling
* Local storage management

## **API Endpoints**

### **Authentication Endpoints**

#### **POST /auth/signup**

Register a new user account.

**Request Body**:

json

{

"email": "user@example.com",

"password": "secure\_password",

"firstName": "John",

"lastName": "Doe"

}

**Response**:

json

{

"success": true,

"message": "Account created successfully"

}

#### **POST /auth/login**

Authenticate user and return tokens.

**Request Body**:

json

{

"email": "user@example.com",

"password": "secure\_password"

}

**Response**:

json

{

"success": true,

"user": {

"id": 1,

"email": "user@example.com",

"first\_name": "John",

"last\_name": "Doe"

},

"access\_token": "jwt\_access\_token",

"refresh\_token": "jwt\_refresh\_token"

}

#### **POST /auth/refresh**

Refresh access token using refresh token.

#### **GET /auth/verify-token**

Verify current access token validity.

#### **POST /auth/logout**

Revoke refresh tokens and logout.

### **Document Management Endpoints**

#### **POST /upload**

Upload and process a new document.

**Headers**: Authorization: Bearer <access\_token> **Content-Type**: multipart/form-data **Form Data**: file - Document file

**Response**:

json

{

"success": true,

"session\_id": "uuid",

"file\_info": {

"name": "document.pdf",

"size": 1024000,

"pages": 10

},

"message": "Document uploaded successfully"

}

#### **GET /api/sessions**

Get all user sessions.

**Headers**: Authorization: Bearer <access\_token>

**Response**:

json

{

"success": true,

"sessions": [

{

"id": "session\_uuid",

"filename": "document.pdf",

"created\_at": "2025-07-25T10:08:38"

}

]

}

#### **GET /api/sessions/<session\_id>**

Get session details and chat history.

#### **DELETE /api/sessions/<session\_id>**

Delete a session and associated data.

### **Chat Endpoints**

#### **POST /chat**

Send a message and get AI response.

**Headers**: Authorization: Bearer <access\_token> **Request Body**:

json

{

"session\_id": "session\_uuid",

"message": "What is the main topic of this document?"

}

**Response**:

json

{

"success": true,

"response": "The main topic of this document is...",

"session\_id": "session\_uuid",

"sources\_used": 3,

"confidence": 0.85,

"usage": {

"prompt\_tokens": 150,

"completion\_tokens": 75,

"total\_tokens": 225

}

}

## **Troubleshooting**

### **Common Issues**

#### **"Token is missing" Error**

**Cause**: Frontend not sending Authorization header or token not stored properly. **Solution**:

1. Check browser localStorage for access\_token
2. Verify token is being sent in Authorization header as "Bearer <token>"
3. Check if token has expired

#### **Database Connection Error**

**Cause**: Incorrect DATABASE\_URL or MySQL server not running. **Solution**:

1. Verify MySQL server is running
2. Check DATABASE\_URL format in .env file
3. Verify database user permissions

#### **File Upload Fails**

**Cause**: File size exceeds limit or unsupported format. **Solution**:

1. Check file size is under MAX\_FILE\_SIZE\_MB
2. Verify file format is supported (PDF, DOCX, DOC, TXT)
3. Check upload directory permissions

#### **OpenAI API Errors**

**Cause**: Invalid API key or quota exceeded. **Solution**:

1. Verify OPENAI\_API\_KEY in .env file
2. Check OpenAI account quota and billing
3. Monitor API usage in OpenAI dashboard