Intelligent Document System (DocuMind IN)

Technical Documentation | Version 1.0 | Target Audience: Developers & DevOps

DocuMind IN is a RAG-powered document Q&A system specifically designed for Indian IT consulting and staffing firms. It allows users to upload compliance policies, rate cards, and checklists, and then query them using natural language. The system provides answers with source citations and confidence scores, utilizing a local backend for document processing.

Table of Contents

- 1. Project Overview
- 2. Architecture Overview
- 3. Tech Stack
- 4. Quick Start Guide
- 5. Backend Reference
- 6. API Reference
- 7. Frontend Reference
- 8. Configuration & Environment Variables
- 9. Deployment Guide
- 10. Testing & Validation
- 11. Troubleshooting
- 12. Adding Documents
- 13. Contributing
- 14. Security Considerations
- 15. Future Roadmap
- 16. License

📋 Executive Summary

This technical documentation provides complete developer-focused reference material for the **Intelligent Document System (DocuMind IN)**, a Retrieval-Augmented Generation (RAG) system built with React, FastAPI, and FAISS. The system enables semantic search over plain text documents with confidence scoring and source attribution.

Key Statistics:

- 🏗️ **Architecture:** Client-server with React frontend + FastAPI backend
- 🧠 **ML Models:** 2 embedding models (MiniLM-L6, MPNet-Base) via Sentence Transformers
- 💾 **Vector Store:** FAISS IndexFlatIP for efficient similarity search
- 📄 **Supported Formats:** Plain text (.txt) with UTF-8 encoding
- 🔧 **Chunking Strategy:** Fixed-size sliding window (500 chars, 50 overlap)
- ✅ **Confidence Threshold:** Default 0.6 cosine similarity (adjustable)

⚠️ Citation Policy & Quality Notes

All technical claims in this documentation are sourced from the project's README.md file located at `https://github.com/aditya4232/Intelligent-document-system`. The repository structure, file paths, and architectural descriptions reflect the documented design. Some implementation details could not be verified directly due to repository access limitations.

TODO Items:

- 🔍 **TODO: Clarify** - Exact implementation of `generator.py` answer synthesis logic (source: inferred from README architecture section)
- 🔍 **TODO: Clarify** - Specific Pydantic schemas in `schemas.py` (source: documented API request/response examples)
- 🔍 **TODO: Clarify** - Complete Python dependencies list (source: requirements.txt - file access failed)
- 🔍 **TODO: Clarify** - Exact implementation of Three.js visual components (source: README mentions CelestialBloomShader, NeuralBackground)

1. Project Overview

DocuMind IN solves the challenge of retrieving specific information from unstructured text files (TXT) in a corporate environment. Unlike generic LLMs, this system uses Retrieval-Augmented Generation (RAG) to ground answers in the user's specific documents, reducing hallucinations and providing traceability.

Key Features

- **Semantic Search:** Uses FAISS and Sentence Transformers to find relevant document sections based on meaning, not just keywords.
- **Dual Embedding Models:** Ships with MiniLM-L6 (fast, 384-dim) and MPNet-Base (high recall, 768-dim).
- **Document Locking:** Users can pin specific files to restrict the search scope for a session.
- **Confidence Guardrails:** Automatically blocks answers if the similarity score falls below a cosine-threshold, ensuring quality.

- **Source Citations:** Every response includes the filename of the source document.
- **Visual Interface:** Features a WebGL shader background and interactive particle canvas built with Three.js.

2. Architecture Overview

The system follows a standard Client-Server architecture with a decoupled frontend and backend.

- **Frontend:** A React Single Page Application (SPA) served via Vite. It handles user interaction, file uploads via drag-and-drop, and chat rendering.
- **Backend:** A FastAPI application acting as the RAG engine. It handles document ingestion, vector storage, and answer generation.
- **Data Flow:**
  - **Ingest:** `.txt` files are uploaded or loaded from `data/`.
  - **Chunking:** Text is split into fixed-size windows (500 chars with 50 char overlap).
  - **Embedding:** Chunks are converted to vectors using Sentence Transformers.
  - **Vector Store:** Vectors are indexed in FAISS for efficient similarity search.
  - **Retrieval:** User queries are embedded and compared against the FAISS index.
  - **Generation:** The system constructs a natural language response based on the retrieved context.

PlantUML Architecture Diagrams

Component Diagram

```
@startuml
package "Frontend" {
  [React App] as ReactApp
  [LandingPage] as Landing
  [ChatPage] as Chat
  [Document Upload] as Upload
}

package "Backend (FastAPI)" {
  [API Router] as API
  [Ingest Module] as Ingest
  [Chunking] as Chunk
  [Embeddings] as Embed
  [Vector Store] as Vector
  [Retriever] as Retrieve
  [Generator] as Gen
  [Guardrails] as Guard
}

package "Storage" {
```

```
  database "FAISS Index" as FAISS
  folder "data/" as Data
}

package "External Services" {
  [Sentence Transformers] as ST
}

ReactApp --> API : HTTP/REST
API --> Ingest : Load documents
API --> Retrieve : Query
API --> Upload : Upload files
Ingest --> Chunk : Process text
Chunk --> Embed : Create embeddings
Embed --> ST : Model inference
Embed --> Vector : Store vectors
Vector --> FAISS : Index operations
Retrieve --> Vector : Search
Retrieve --> Guard : Validate
Guard --> Gen : Synthesize answer
Data --> Ingest : Read .txt files
Upload --> Data : Save files

@enduml
```

Sequence Diagram - Query Flow

```
@startuml
actor User
participant "React Frontend" as Frontend
participant "FastAPI Backend" as Backend
participant "Retriever" as Retriever
participant "FAISS" as FAISS
participant "Guardrails" as Guard
participant "Generator" as Gen

User -> Frontend: Submit question
Frontend -> Backend: POST /ask-recruiter
Backend -> Retriever: Search(query, top_k, filters)
Retriever -> FAISS: Similarity search
FAISS --> Retriever: Top K chunks
Retriever -> Guard: Validate confidence
Guard --> Retriever: Pass/Fail
alt Confidence >= Threshold
    Retriever -> Gen: Generate answer(chunks)
    Gen --> Backend: Answer + sources
    Backend --> Frontend: JSON response
    Frontend --> User: Display answer with citations
else Confidence < Threshold
```

```
    Backend --> Frontend: Low confidence error
    Frontend --> User: "No relevant information found"
end

@enduml
```

Project File Structure

```
Intelligent-document-system/
├── backend/
│   ├── app/
│   │   ├── __init__.py
│   │   ├── main.py              # FastAPI app entry point
│   │   ├── api.py               # Route handlers
│   │   ├── schemas.py           # Pydantic models
│   │   ├── ingest.py            # Document loader
│   │   ├── chunking.py          # Text chunking logic
│   │   ├── embeddings.py        # Embedding wrapper
│   │   ├── vector_store.py      # FAISS wrapper
│   │   ├── retriever.py         # Semantic search
│   │   ├── guardrails.py        # Confidence validation
│   │   └── generator.py         # Answer synthesis
│   ├── data/                    # Document storage (.txt files)
│   └── requirements.txt         # Python dependencies
├── src/
│   ├── components/
│   │   ├── CelestialBloomShader.jsx
│   │   ├── NeuralBackground.jsx
│   │   ├── ChatMessage.jsx
│   │   └── ...
│   ├── hooks/
│   │   ├── useDocuments.js
│   │   ├── useSettings.js
│   │   └── useSidebarResize.js
│   ├── pages/
│   │   ├── LandingPage.jsx
│   │   └── ChatPage.jsx
│   └── main.jsx
├── .github/
│   └── workflows/
│       └── deploy.yml           # GitHub Actions CI/CD
├── vite.config.js               # Vite configuration
├── netlify.toml                 # Netlify deployment config
├── package.json                 # Node dependencies
└── README.md
```

3. Tech Stack

Component	Technology	Details
Frontend	React 18, Vite 6	React Router 6, Axios, Lucide React, CSS Modules
Visuals	Three.js	CelestialBloomShader, NeuralBackground
Backend Framework	FastAPI	Python 3.10+, Uvicorn
ML/AI	Sentence Transformers	`all-MiniLM-L6-v2`, `all-mpnet-base-v2`
Vector Database	FAISS	Facebook AI Similarity Search (IndexFlatIP)
Validation	Pydantic v2	Data validation and settings management

4. Quick Start Guide

Prerequisites

- Node.js 18+ and npm 9+
- Python 3.10+
- Git

Installation Steps

1. Clone the Repository

```
git clone https://github.com/aditya4232/Intelligent-document-
system.git
cd Intelligent-document-system
```

2. Backend Setup

The backend requires a Python virtual environment to manage dependencies.

```
# Create virtual environment
python -m venv .venv

# Activate (Windows)
.venv\Scripts\activate
# Activate (macOS/Linux)
source .venv/bin/activate

# Install dependencies
pip install -r backend/requirements.txt
```

```
# Start the server
cd backend
uvicorn app.main:app --reload --port 8000
```

Note: The first run will download approximately 510 MB of model data (MiniLM and MPNet). Subsequent starts are instant.

3. Frontend Setup

Open a new terminal window in the project root.

```
npm install
npm run dev
```

The application will launch at `http://localhost:3000`.

5. Backend Reference

The backend logic is contained within the `backend/app/` directory.

Core Modules

`app/main.py`

Purpose: Application entry point. Configures CORS, initializes the FastAPI app, and triggers startup ingestion.

`app/api.py`

Purpose: Route definitions. Connects HTTP requests to business logic for asking questions and managing documents.

`app/schemas.py`

Purpose: Pydantic models defining request and response structures. Ensures type safety for API inputs like query parameters and guardrail settings.

`app/ingest.py`

Purpose: Document loading logic. Scans the `data/` directory to load `.txt` files into memory for processing.

`app/chunking.py`

Purpose: Text processing strategy. Implements a fixed-size sliding window algorithm.
Configuration: 500 characters per chunk, 50 characters overlap.

`app/embeddings.py`

Purpose: Wrapper for Sentence Transformers. Manages the loading and inference of `MiniLM-L6` and `MPNet-Base`.

`app/vector_store.py`

Purpose: FAISS wrapper. Manages the creation of the `IndexFlatIP` (Inner Product) index and handles vector addition and search operations.

`app/retriever.py`

Purpose: Semantic search logic. Performs the vector search against the store and supports optional source filtering (document locking).

`app/guardrails.py`

Purpose: Quality control. Implements a cosine-threshold gate. If the top result's similarity score is below the configured threshold, the system refuses to answer to prevent hallucination.

`app/generator.py`

Purpose: Answer synthesis. Takes retrieved chunks and constructs a cohesive answer with source attribution.

6. API Reference

The backend serves API documentation automatically at `http://127.0.0.1:8000/docs`.

Endpoints

POST /ask-recruiter

Purpose: Primary Q&A endpoint. Synthesizes an answer from matching document chunks.

Request Body:

```
{
  "question": "What are the BGV requirements for senior hires?",
  "top_k": 3,
  "temperature": 0.7,
  "source_filter": ["compliance_policy.txt"],
  "guardrails_enabled": true,
  "confidence_threshold": 0.6,
  "embedding_model": "MiniLM-L6"
}
```

Response:

```
{
  "answer": "Based on **compliance_policy.txt**:\n\nIf candidate
fails background check...",
  "confidence": "high",
  "source_documents": ["compliance_policy.txt"],
  "similarity_score": 0.83
}
```

Request Parameters:

Parameter	Type	Required	Description
`question`	string	Yes	The natural language question to ask
`top_k`	integer	No	Number of document chunks to retrieve (default: 3)
`temperature`	float	No	Response randomness (0.0-1.0, default: 0.7)
`source_filter`	array[string]	No	List of filenames to restrict search (document locking)
`guardrails_enabled`	boolean	No	Enable confidence validation (default: true)
`confidence_threshold`	float	No	Minimum similarity score (0.0-1.0, default: 0.6)
`embedding_model`	string	No	Model to use: "MiniLM-L6" or "MPNet-Base" (default: MiniLM-L6)

cURL Example:

```
curl -X POST "http://127.0.0.1:8000/ask-recruiter" \
  -H "Content-Type: application/json" \
  -d '{
    "question": "What are the BGV requirements for senior hires?",
    "top_k": 3,
    "temperature": 0.7,
    "source_filter": ["compliance_policy.txt"],
```

```
    "guardrails_enabled": true,
    "confidence_threshold": 0.6,
    "embedding_model": "MiniLM-L6"
  }'
```

GET /documents

Purpose: Lists all currently indexed document filenames.

Response Example:

```
{
  "documents": [
    "compliance_policy.txt",
    "rate_card_2024.txt",
    "screening_checklist.txt",
    "onboarding_pack.txt"
  ]
}
```

cURL Example:

```
curl -X GET "http://127.0.0.1:8000/documents"
```

POST /documents/upload

Purpose: Uploads new files. Accepts `multipart/form-data`. Files are saved to `data/`, chunked, embedded, and indexed immediately.

Response Example:

```
{
  "message": "File uploaded successfully",
  "filename": "new_policy.txt",
  "chunks_created": 15
}
```

cURL Example:

```
curl -X POST "http://127.0.0.1:8000/documents/upload" \
  -F "file=@/path/to/document.txt"
```

GET /models

Purpose: Returns the list of available embedding models (MiniLM-L6, MPNet-Base).

Response Example:

```
{
  "models": [
    {
      "name": "MiniLM-L6",
      "dimensions": 384,
      "description": "Fast, lightweight model suitable for most use
cases"
    },
    {
      "name": "MPNet-Base",
      "dimensions": 768,
      "description": "Higher quality, better recall, slower
inference"
    }
  ]
}
```

cURL Example:

```
curl -X GET "http://127.0.0.1:8000/models"
```

7. Frontend Reference

The frontend is a React application built with Vite, located in the `src/` directory.

Structure

- `components/`: Reusable UI elements.
  - `CelestialBloomShader`, `NeuralBackground`: Three.js background effects.
  - `ChatMessage`: Renders individual chat bubbles with citations.
- `hooks/`: Custom React hooks.
  - `useDocuments`: Manages file lists and upload state.
  - `useSettings`: Manages RAG parameters (top-k, temperature).
  - `useSidebarResize`: Handles UI layout responsiveness.
- `pages/`:
  - `LandingPage.jsx`: Marketing/Entry page.

- `ChatPage.jsx`: The main workspace for Q&A and document management.

8. Configuration

Environment variables are managed via `.env` files. Defaults are provided for local development.

Variable	Context	Default	Description
`VITE_API_URL`	Frontend	`http://127.0.0.1:8000`	Address of the backend API.
`VITE_BASE_PATH`	Frontend	`/`	Base path for GitHub Pages deployment (set by CI).
`ALLOWED_ORIGINS`	Backend	`localhost:3000,5173`	CORS whitelist for browser access.
`DOCS_API_KEY`	Backend	(empty)	Optional key to secure document upload endpoints.

9. Deployment Guide

Frontend (GitHub Pages)

The repository includes a GitHub Actions workflow at `.github/workflows/deploy.yml`.

1. Go to GitHub Repository **Settings** > **Pages**.
2. Set **Source** to `Deploy from branch` and select `gh-pages`.
3. Add a repository secret `VITE_API_URL` pointing to your production backend.
4. Push to `main` triggers a build and deploy.

Backend (Render/Railway)

The backend is Python-based and stateless (assuming ephemeral storage for this demo, or persistent disk if configured).

- **Root Directory:** `backend`
- **Build Command:** `pip install -r requirements.txt`
- **Start Command:** `uvicorn app.main:app --host 0.0.0.0 --port $PORT`
- **Environment:** Set `ALLOWED_ORIGINS` to your frontend's URL.

10. Testing & Validation

The system relies on "Guardrails" defined in `app/guardrails.py` for validation.

- **Confidence Threshold:** Default is set to 0.6. Answers with similarity scores below this are rejected.

- **Validation:** Users can manually adjust the "Confidence Guardrail" in the UI settings panel to test system sensitivity.

11. Troubleshooting

Common Issues

- **Slow Startup:** The first backend run downloads ML models (~500MB). Ensure you have a stable internet connection.
- **CORS Errors:** If the frontend cannot talk to the backend, check `ALLOWED_ORIGINS` in the backend `.env` and ensure the `VITE_API_URL` is correct.
- **No Answers Found:** Check if documents are actually uploaded. Use `GET /documents` to verify index status. Ensure the "Confidence Threshold" isn't set too high (e.g., > 0.9).

FAQ

- **Q: Can I use PDF or Word documents?**
  A: Currently, only `.txt` files are supported. Convert your documents to plain text before uploading.
- **Q: How do I clear the index?**
  A: Delete files from `backend/data/` and restart the server. The index is rebuilt on startup.
- **Q: What's the difference between the two embedding models?**
  A: MiniLM-L6 is faster (384 dimensions) and suitable for most use cases. MPNet-Base is slower but provides better recall (768 dimensions) for complex queries.
- **Q: How large can documents be?**
  A: There's no hard limit, but very large files (>10MB) may slow down indexing. Consider splitting them into smaller logical documents.

Log Locations

- **Backend logs:** Check the terminal where `uvicorn` is running
- **Frontend logs:** Browser Developer Console (F12)
- **Document storage:** `backend/data/`

12. Adding Documents

There are two methods to add documents to the system:

Method 1: UI Upload (Recommended)

1. Open the chat interface at `http://localhost:3000`
2. Drag and drop `.txt` files into the upload area
3. Files are automatically processed, chunked, and indexed

4. Refresh the document list to confirm indexing

Method 2: Manual File Copy

1. Stop the backend server if running
2. Copy `.txt` files to `backend/data/`
3. Restart the backend: `uvicorn app.main:app --reload --port 8000`
4. Files are automatically indexed on startup

> ⚠️ **Important:** Only `.txt` files with UTF-8 encoding are currently supported. Ensure your documents are properly formatted to avoid encoding errors.

13. Contributing

Contributions are welcome! Follow these steps to contribute to the project:

Development Workflow

1. Fork the repository on GitHub
2. Clone your fork locally:

```
git clone https://github.com/YOUR_USERNAME/Intelligent-document-system.git
cd Intelligent-document-system
```

3. Create a feature branch:

```
git checkout -b feat/my-new-feature
```

4. Make your changes and commit:

```
git add .
git commit -m "feat: add my new feature"
```

5. Push to your fork:

```
git push origin feat/my-new-feature
```

6. Open a Pull Request on the main repository

Commit Message Convention

Use conventional commits format:

- `feat:` New features
- `fix:` Bug fixes
- `docs:` Documentation changes
- `style:` Code style changes (formatting, etc.)
- `refactor:` Code refactoring
- `test:` Adding or updating tests
- `chore:` Maintenance tasks

Code Quality

- Follow PEP 8 style guide for Python code
- Use ESLint rules for JavaScript/React code
- Add docstrings to Python functions and classes
- Write meaningful commit messages
- Test your changes locally before submitting

14. Security Considerations

- **API Authentication:** The `DOCS_API_KEY` environment variable can be used to secure document upload endpoints in production deployments.
- **CORS Configuration:** Always set `ALLOWED_ORIGINS` to your specific frontend URL in production. Never use wildcard (*) origins.
- **Data Privacy:** Documents uploaded to the system are stored locally in `backend/data/`. Ensure sensitive data is properly secured at the infrastructure level.
- **Input Validation:** All API inputs are validated using Pydantic schemas to prevent injection attacks.

> ⚠️ **Security Note:** This is a demonstration system. For production use with sensitive data, implement proper authentication, authorization, and data encryption.

15. Future Roadmap

Potential enhancements for future versions:

- ✨ Support for PDF, DOCX, and other document formats
- 🔍 Advanced query operators (AND, OR, NOT)
- 📊 Analytics dashboard for query patterns
- 🔐 User authentication and role-based access control
- 💾 PostgreSQL backend for persistent document storage

- 🤖 Integration with LLMs (OpenAI, Claude) for answer generation
- 📱 Mobile-responsive UI improvements
- 🌐 Multi-language support
- ⚡ Real-time collaborative document editing
- 📈 Relevance feedback and learning from user interactions

16. License
