# Technology Stack

Backend Technologies

• Language: Python 3.8+

• Web Framework: FastAPI with Uvicorn ASGI server

• WebSocket Support: Real-time communication via WebSockets

• Template Engine: Jinja2 for HTML templates

• CORS: Cross-Origin Resource Sharing middleware

AI/ML Models (All Free & Open Source)

• Embeddings: paraphrase-multilingual-MiniLM-L12-v2 (SentenceTransformers)

• Summarization: sshleifer/distilbart-cnn-12-6 (Hugging Face)

• Question Answering: distilbert-base-cased-distilled-squad (Hugging Face)

• Text Classification: cardiffnlp/twitter-roberta-base-sentiment-latest (Hugging Face)

• Language Detection: papluca/xlm-roberta-base-language-detection (temporarily disabled)

• ML Framework: PyTorch + Transformers

Document Processing

• PDF Processing: PyPDF2

• DOCX Processing: python-docx

• PPTX Processing: python-pptx

• Image Processing: Pillow (PIL), OpenCV

• OCR Engines:

• Tesseract (via pytesseract) for traditional OCR

• EasyOCR for multilingual text recognition

• Layout Analysis: LayoutParser (optional, for advanced document structure detection)

Vector Storage & Search

• Vector Database: FAISS (Facebook AI Similarity Search)

• Search: Semantic similarity search using embeddings

• Chunking: Document content chunked for efficient storage and retrieval

Frontend Technologies

• Core: Vanilla JavaScript (ES6+)

• CSS Framework: Custom CSS with modern design patterns

• Fonts: Inter from Google Fonts

• Icons: Font Awesome 6.0

• Architecture: Single Page Application (SPA) approach

• Real-time: WebSocket client for live chat

File Storage

• Upload Directory: Local filesystem (uploads/)

• Vector Store: Local FAISS indices (vector\_store/)

• Static Files: CSS, JS, and assets (static/)

Supported Document Formats

• PDF, DOCX, PPTX

• Images: PNG, JPG, JPEG, TIFF, BMP

• Multi-language OCR support for 10+ languages

# User Flow

## Initial startup Flow

User starts application → Python main.py

↓

FastAPI server starts on localhost:8000

↓

Background AI model initialization begins

↓

Web interface becomes available with loading screen

↓

Models finish loading → System ready indicator

## Document Upload Flow

User lands on main interface

↓

Drag & drop OR click to select document

↓

File validation (format, size checks)

↓

Document upload via multipart form

↓

Server processing pipeline:

├── Save file to uploads/

├── Route to appropriate processor (PDF/DOCX/PPTX/Image)

├── Extract text & layout information

├── Apply OCR if needed (for images/scanned docs)

├── Detect languages

├── Generate document chunks

├── Create embeddings for each chunk

├── Store in FAISS vector database

├── Generate AI summary

└── Calculate document statistics

↓

Return processing results to frontend

↓

Document appears in document list

## Real-time Chat Flow

User clicks on processed document

↓

Chat interface opens

↓

WebSocket connection established (/ws/{client\_id})

↓

User types question and hits enter

↓

Frontend sends message via WebSocket

↓

Server processing:

├── Receive chat message

├── Generate question embedding

├── Search vector store for relevant chunks

├── Extract context with page numbers

├── Send to QA model with context

├── Generate answer with confidence score

├── Format response with source citations

└── Send back via WebSocket

↓

Real-time response appears in chat

↓

User can ask follow-up questions (cycle repeats)

## Document Management Flow

Documents List View:

├── View all processed documents

├── See document statistics (pages, languages, chunks)

├── Access document analysis

├── Delete documents

└── Search across all documents

Document Analysis:

├── Detailed document structure analysis

├── Language detection results

├── Element type breakdown

├── Readability scores

└── Processing statistics

## API Integration Flow

External Application

↓

REST API Calls:

├── POST /upload (document upload)

├── POST /chat (question answering)

├── GET /documents (list documents)

├── GET /search (semantic search)

├── GET /analyze/{doc\_id} (document analysis)

├── GET /stats (system statistics)

└── GET /health (health check)

OR

WebSocket Connection:

└── /ws/{client\_id} (real-time chat)

## System Architecture Data Flow

Document Input

↓

Document Processor Layer

├── Multi-format parsers

├── OCR engines

└── Layout analysis

↓

AI Processing Layer

├── Text chunking

├── Embedding generation

├── Language detection

└── Content analysis

↓

Vector Storage Layer

├── FAISS indexing

├── Similarity search

└── Context retrieval

↓

Chat Interface Layer

├── Question processing

├── Context-aware responses

└── Real-time delivery