Software Requirements Specification

for

SMARTDOCQ

**Keshav Memorial Institute Of Technology**

**Version 1.0 approved**

(Group No.: G552)

Team Members:

23BD1A0567-KOMAL

23BD1A0568-RUCHITHA

23BD1A056B-UDAYA SRI

23BD1A1296-ANANYA

23BD1A12C1-SUSHMITHA

Software Requirements Specification for <SmartDocQ> Page ii

# Table of Contents

[Table of Contents 2](#_Toc6145)

[Revision History 3](#_Toc6146)

[1. Introduction 1](#_Toc6147)

[1.1 Purpose 1](#_Toc6148)

[1.2 Document Conventions 1](#_Toc6149)

[1.3 Intended Audience and Reading Suggestions 1](#_Toc6150)

[1.4 Product Scope 2](#_Toc6151)

[1.5 References 3](#_Toc6152)

[2. Overall Description 4](#_Toc6153)

[2.1 Product Perspective 4](#_Toc6154)

[2.2 Product Functions 4](#_Toc6155)

[2.3 User Classes and Characteristics 5](#_Toc6156)

[2.4 Operating Environment 6](#_Toc6157)

[2.5 Design and Implementation Constraints 6](#_Toc6158)

[2.6 User Documentation 7](#_Toc6159)

[2.7 Assumptions and Dependencies 7](#_Toc6160)

[3. External Interface Requirements 8](#_Toc6161)

[3.1 User Interfaces 8](#_Toc6162)

[3.2 Hardware Interfaces 12](#_Toc6163)

[3.3 Software Interfaces 12](#_Toc6164)

[3.4 Communications Interfaces 14](#_Toc6165)

[4. System Features 15](#_Toc6166)

[4.1 System Feature 1 **Error! Bookmark not defined.**](#_Toc6167)

[4.2 System Feature 2 (and so on) **Error! Bookmark not defined.**](#_Toc6168)

[5. Other Nonfunctional Requirements 20](#_Toc6169)

[5.1 Performance Requirements 20](#_Toc6170)

[5.2 Safety Requirements 21](#_Toc6171)

[5.3 Security Requirements 21](#_Toc6172)

[5.4 Software Quality Attributes 22](#_Toc6173)

[5.5 Business Rules 23](#_Toc6174)

[6. Other Requirements **Error! Bookmark not defined.**](#_Toc6175)

[Appendix A: Glossary **Error! Bookmark not defined.**](#_Toc6176)

[Appendix B: Analysis Models **Error! Bookmark not defined.**](#_Toc6177)

[Appendix C: To Be Determined List **Error! Bookmark not defined.**](#_Toc6178)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Reason For Changes | Version |
| G552 | 28-07-2025 | Introduction , Overall Description , External Interface Requirements | 1.0 |
| G552 | 30-07-2025 | System Features, Other Non-Functional Requirements | 1.1 |

# **Introduction**

## **Purpose**

This Software Requirements Specification (SRS) defines the requirements for SmartDocQ – Your AI Assistant for Smarter Document Q\&A (v1.0). This system allows users to upload documents such as PDFs, Word files, text files, and URLs, and ask natural language questions based on their contents. Using AI technologies like Google Gemini and Retrieval-Augmented Generation (RAG), the system retrieves relevant content segments and produces intelligent, context-aware responses.

This document focuses on the core subsystem responsible for document ingestion, semantic embedding, vector storage, and intelligent question answering. The target audience includes developers, testers, and stakeholders involved in designing, implementing, and evaluating SmartDocQ.

## **Document Conventions**

Bold is used for section headings and important terms.

Times New Roman indicate referenced components or modules.

Monospaced text is used for code, APIs, and system functions.

Each requirement will be identified with a unique identifier (e.g., FR-1, NFR-3).

All requirements are assumed to inherit the priority of their parent categories unless explicitly stated.

Font Size: 16 and Bold for Titles, 14 and Bold for heading and 12 for content

## **Intended Audience and Reading Suggestions**

This document is intended for:

Developers: To implement backend (FastAPI) and frontend (React) components.

Testers: To design and run tests on functional and non-functional modules.

Project Managers: To track milestones and ensure delivery aligns with requirements.

End Users and UI Designers: To understand user interaction expectations.

Stakeholders: To assess the product scope, goals, and deliverables.

Reading Order Suggestion:

1. Start with Section 1 (Introduction) to understand the project context.

2. Continue to Section 2 for detailed system description.

3. Move to Sections 3 & 4 for functional and non-functional requirements.

4. Reference Appendices or References for supporting materials or external resources.

## **Product Scope**

SmartDocQ is a smart, AI-based question answering platform for documents. Its key objective is to transform static documents into interactive learning tools. By enabling semantic search and contextual Q\&A using AI models like Google Gemini, it enhances document comprehension for students, researchers, educators, and legal/corporate professionals.

The tool:

Simplifies studying by providing quick, accurate answers.

Converts documents into searchable vector embeddings.

Supports memory tracking and feedback loops.

Can be scaled for enterprise or academic applications.

This aligns with broader goals of digital transformation in education and corporate knowledge management.

## **References**

The following published research papers, technical documentation, and online standards have been referenced during the development of SmartDocQ – Your AI Assistant for Smarter Document Q&A:

1)Lewis, Patrick, et al.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.

Proceedings of NeurIPS 2020.

URL: <https://arxiv.org/abs/2005.11401>

2)Brown, Tom, et al.

Language Models are Few-Shot Learners.

NeurIPS 2020.

URL: <https://arxiv.org/abs/2005.14165>

3)Bommasani, Rishi, et al.

On the Opportunities and Risks of Foundation Models.

Stanford Institute for Human-Centered AI, 2021.

URL[: https://arxiv.org/abs/2108.07258](file:///C:\Users\Dell\AppData\Local\Microsoft\Windows\INetCache\IE\BMRVFY5U\%20https\arxiv.org\abs\2108.07258)

4)Google AI Blog.

Introducing Gemini 1.5: Our Next Generation Multimodal Model.

Published by Google DeepMind, 2024.

URL: [https://deepmind.google/technologies/gemini/gemini-1-5](https://deepmind.google/technologies/gemini/gemini-1-5%20)

# **Overall Description**

## **Product Perspective**

SmartDocQ is a new, self-contained AI-based document Q\&A system designed to enable users to interact intelligently with textual documents by uploading files and asking natural language questions. It is not part of an existing product family or system, but it can be extended to integrate with other academic, research, or content management systems via APIs.

The system architecture includes:

A React.js frontend for user interaction

A FastAPI backend for handling file processing, question answering, and AI logic

A vector database (FAISS/Chroma) for semantic search

An LLM (e.g., Mistral via Ollama) for local and free answer generation

**Product Functions**

SmartDocQ provides the following high-level functions:

Upload document files (PDF, DOCX, TXT)

Extract and chunk content into meaningful segments

Generate semantic embeddings using local models

Store and retrieve data using a vector database (FAISS or Chroma)

Use a local LLM (like Mistral) to generate accurate natural-language answers

Provide an interactive chat interface for question-answering

Optional memory handling for multi-turn conversation

Accept feedback from users to rate responses

Log chat history and answers for future analysis

## **User Classes and Characteristics**

|  |  |
| --- | --- |
| User Class | Characteristics |
| Students | Basic computer knowledge; use SmartDocQ to ask questions from textbooks/notes |
| Teachers | Moderate tech experience; upload lecture notes and create dynamic Q\&A sets |
| Researchers | Experienced users; analyze research papers and technical documents |
| General Users | Casual use; no advanced skills needed; expect intuitive UI and fast responses |

Most users are expected to have basic English literacy and use SmartDocQ through a browser interface.

No programming knowledge is required.

## **Operating Environment**

SmartDocQ will operate in the following environment:

Frontend : Browser-based (Chrome, Firefox, Edge), built with React.js

Backend : Python-based FastAPI server running on Windows/Linux/macOS

Database : FAISS or ChromaDB for vector storage

Embedding Model : sentence-transformers (e.g., MiniLM)

LLM Model : Mistral/OpenHermes running via Ollama or LM Studio

Deployment : Can run locally or be hosted on cloud platforms (optional)

## **Design and Implementation Constraints**

Must use only free and offline-compatible AI tools (e.g., Mistral via Ollama, MiniLM for embeddings).

Must work on systems with limited RAM/CPU (e.g., 8 GB).

Frontend must be built using React.js; backend must use FastAPI.

APIs must be RESTful and documented.

Embedding models must not exceed a reasonable size (e.g., MiniLM \~100MB).

No use of paid services like OpenAI or Gemini unless explicitly authorized.

Must support basic file types only: .pdf, .docx, .txt.

## **User Documentation**

The following user documentation will be provided:

User Manual : PDF or HTML document explaining how to use the system.

Quick Start Guide : Embedded in the UI or linked from the homepage.

In-App Tooltips : Inline hints on upload and chat functionality.

Developer README : Contains setup instructions, architecture details, and code usage for developers.

API Reference : Swagger UI (auto-generated via FastAPI)

## **Assumptions and Dependencies**

Assumptions:

Users will upload readable, text-based documents (OCR for images is not included).

All AI models (embedding and LLM) will run locally without API keys.

Users will operate the application on a modern system (8+ GB RAM recommended).

All inputs will be in English.

Dependencies:

Relies on open-source Python libraries such as sentence-transformers, faiss-cpu, PyMuPDF, and ollama.

System performance may depend on hardware specs.

If deployed online, external dependencies (e.g., hosting, SSL, domain) may be required.

# **External Interface Requirements**

## **User Interfaces**

SmartDocQ provides a clean, modern, and interactive web interface built using React.js. It is designed for seamless interaction with AI-driven features such as document Q\&A, summarization, voice input, multilingual support, and downloadable reports. The UI is user-friendly and accessible across devices.

**Core Interface Components**

Document Upload Panel

Accepts: PDF, DOCX, TXT, and URLs

Drag & drop or manual file selection

Shows progress bar, validation messages, and file metadata

Multiple file upload support

**Chat & Q\&A Interface**

Text Input Field: For typing natural language questions

Voice Input : Users can ask questions via microphone (Speech-to-Text API)

Submit Button: Sends query to the backend (FastAPI + Gemini)

**Answer Display:**

Natural language response

Highlights the referenced document snippet

Allows feedback

Shows processing time

**Summarization Panel**

Appears after document upload

Options:

Full document summary

Section-wise summary

Summary can be copied or downloaded

**Multi-Language Support**

Language dropdown or settings gear icon

Interface and responses available in multiple languages

Supports both question input and AI response localization

Languages: English, Hindi, Telugu, and others (based on user preference)

Downloadable Reports

**Allows exporting:**

Individual Q\&A pairs

Full chat history

Document summaries

Formats: PDF, TXT

Button: Download Report shown in chat panel and summary screen

**Navigation / Sidebar**

Menu: Home, Upload, Chat History, Summary, Downloads, Settings, Help, Logout

**Chat History Panel**

Lists past interactions

Searchable by keyword

Allows restoring a previous session

Design Standards & Guidelines

Material UI / Google Material Design

Color contrast and typography for accessibility

**Responsive across:**

Desktop (Chrome, Firefox)

Mobile (Android/iOS browsers)

Tablet devices

Error Handling & Notifications

Inline Errors & Toast Alerts for:

Unsupported files

Empty input or failed response

Speech recognition issues

Clear loading indicators and fallback messages

## **Hardware Interfaces**

SmartDocQ is a web-based application and requires no direct hardware integration. However, the system will interact indirectly with the following hardware:

Client-Side Hardware (Users):

Desktop, Laptop, Tablet, or Smartphone with a modern web browser.

Microphone (for future voice support).

Server-Side Hardware:

Backend runs on cloud-hosted virtual servers (e.g., AWS, GCP).

GPU-accelerated machines for AI inference (for Gemini model calls or embedding generation).

SSD-based storage for speed in document vector retrieval.

Data Exchange Characteristics:

All data is transferred over HTTP/HTTPS protocols.

Uploaded documents are stored temporarily in server memory or disk for pre-processing.

## **Software Interfaces**

SmartDocQ interacts with several software components:

**Backend and AI Integration:**

FastAPI (v0.95+): Handles routing, document upload, processing, and API services.

Google Gemini API: Provides language understanding and generation (Q&A, summarization).

ChromaDB/Pinecone: Vector database for storing and retrieving document embeddings.

MongoDB (v6+): Stores user metadata, document metadata, and chat logs.

Redis/In-memory caching: Manages session state and memory module (short-term history).

**Frontend:**

React.js (v18+): Provides the web-based interactive interface.

Communicates with FastAPI via RESTful APIs.

**Libraries and Tools**:

Tika, pdfplumber: For document parsing

LangChain/Sentence Transformers: For embedding generation

dotenv, logging, uvicorn, pydantic: Environment and validation tools

**Data Flow Summary:**

Input: Documents (uploaded), Queries (text)

Output: Answers (text), Summaries, Highlight references, Chat history

All API routes are RESTful and secured using JWT-based authentication.

## **Communications Interfaces**

SmartDocQ is a cloud-hosted web application that communicates over the internet using secure protocols.

**Communication Mechanisms:**

HTTP/HTTPS: Used for all frontend-backend and backend-AI API interactions.

WebSockets (optional for future real-time chat updates)

JWT Tokens: Used for authentication between client and server.

Data Uploads: Documents are uploaded over HTTPS with MIME type verification.

API Calls: RESTful requests to Google Gemini and vector DB.

**Security Considerations:**

HTTPS enforced: All communication is SSL/TLS encrypted.

File Validation: Prevents uploading of malicious files.

CORS Policy: Configured to restrict unauthorized cross-origin access.

Rate Limiting & API Throttling: Prevents misuse of AI APIs.

**Performance Metrics:**

Document upload response within 2–4 seconds (average)

Q&A response latency under 5 seconds for <10MB documents

**Supported Browsers:**

Chrome, Firefox, Safari, and Edge (latest 2 versions recommended)

# **System Features**

## **Document Upload and Parsing**

### **Description and Priority**

This feature allows users to upload documents (PDF, DOCX, or TXT) into the system for analysis and question answering. It includes content extraction and chunking.

Priority: High

Benefit: 9

Penalty: 8

Cost: 3

Risk: 2

### **Stimulus/Response Sequences**

Stimulus: User selects and uploads a document.

System Response:

Parses document text

Divides text into manageable chunks (e.g., 300 words)

Stores original file for reference

Shows “Upload successful” message

### **Functional Requirements**

REQ-1.1: The system shall accept files with .pdf, .docx, and .txt extensions.

REQ-1.2: The system shall reject files exceeding 10 MB

REQ-1.3: The system shall extract readable text from the uploaded file.

REQ-1.4: The system shall chunk extracted text into smaller segments for embedding.

REQ-1.5: The system shall notify the user of successful or failed upload.

**4.2 Embedding Generation and Vector Storage**

**4.2.1 Description and Priority**

Generates semantic vector embeddings from text chunks and stores them for later retrieval.

Priority: High

Benefit: 9

Penalty: 7

Cost: 2

Risk: 3

**4.2.2 Stimulus/Response Sequences**

Stimulus: A document is successfully parsed and chunked.

System Response:

Generates semantic vectors for each chunk

Stores vectors in FAISS or ChromaDB

Logs vector generation completion

**4.2.3 Functional Requirements**

REQ-2.1: The system shall use a sentence-transformer model to convert text chunks into vectors.

REQ-2.2: The system shall store all generated vectors in a vector database.

REQ-2.3: The system shall store reference metadata (document name, chunk number) with each vector.

REQ-2.4: The system shall log errors in vector generation or storage.

**4.3 Question Answering Interface**

**4.3.1 Description and Priority**

Allows users to type in natural language questions and receive accurate, context-based answers.

Priority: High

Benefit: 10

Penalty: 9

Cost: 3

Risk: 4

**4.3.2 Stimulus/Response Sequences**

Stimulus: User enters a question in the chat input box.

System Response:

Converts question into a vector

Retrieves top-matching document chunks

Sends content to the LLM

Displays generated answer on the frontend

**4.3.3 Functional Requirements**

REQ-3.1: The system shall allow the user to input a question via text box.

REQ-3.2: The system shall convert the question into a semantic vector.

REQ-3.3:The system shall retrieve top-K similar vectors from the vector store.

REQ-3.4: The system shall pass relevant content and the question to the local LLM (e.g., Mistral).

REQ-3.5: The system shall display the generated answer in the frontend chat interface.

REQ-3.6: The system shall return an error message if no document is uploaded.

**4.4 Answer Feedback and Session Tracking**

**4.4.1 Description and Priority**

Enables users to rate answers and maintains session history for future reference (optional in v1).

Priority: Medium

Benefit: 6

Penalty: 2

Cost: 3

Risk: 2

**4.4.2 Stimulus/Response Sequences**

Stimulus: User clicks 👍 or 👎 on an answer.

System Response:

Logs feedback

Optionally stores Q\&A pair in session memory for context tracking

**4.4.3 Functional Requirements**

REQ-4.1: The system shall allow users to rate an answer using a thumbs-up/down button.

REQ-4.2: The system shall log feedback in a structured format.

REQ-4.3: The system shall maintain chat history during a single session.

REQ-4.4: (Optional) The system may associate a session ID for long-term memory storage.

# **Other Nonfunctional Requirements**

## **Performance Requirements**

SmartDocQ is designed for high performance and responsiveness to ensure smooth and efficient interactions. Performance goals are as follows:

**Response Time:**

Document upload and chunking: ≤ 4 seconds for files up to 10MB

Question-answer generation: ≤ 5 seconds per query (using Gemini API + vector retrieval)

Summarization: ≤ 7 seconds for standard-length documents (\~5 pages)

**Concurrency:**

Supports simultaneous interaction by at least 100 concurrent users.

Load balancing handled via API gateway and scalable cloud instances.

**Availability:**

99.5% uptime with fallback error messages in case of AI model/API failure.

**Resource Utilization:**

Memory-efficient in-memory caching (e.g., Redis) for session context.

GPU acceleration used for embedding generation and AI inference when available.

## **Safety Requirements**

SmartDocQ is primarily a web-based document assistant and poses minimal physical safety risks. However, digital safety is prioritized through the following measures:

**File Validation:**

Uploaded documents are scanned for malicious content and unsupported formats.

Scripts or embedded executable content are blocked.

**Data Isolation:**

Users cannot access others’ documents or data under any circumstance.

Session boundaries are strictly enforced.

**Fallback Behavior:**

In the event of AI model failure, the system provides top matching document snippets as a fallback mechanism instead of returning blank output.

**Compliance:**

Adheres to best practices recommended by OWASP and relevant data safety norms (ISO/IEC 27001 guidelines if scaled).

## **Security Requirements**

Given that SmartDocQ processes potentially sensitive educational or legal documents, strict security controls are enforced:

**Authentication & Authorization:**

JWT-based user login with email/password credentials.

Role-based access (e.g., Admin, Student, Educator) for future extensions.

**Data Security:**

All data is transmitted via HTTPS (SSL/TLS encryption).

Uploaded documents and metadata are stored in secured databases (MongoDB).

**Session Management:**

Sessions are automatically invalidated after a period of inactivity (e.g., 30 minutes).

Rate limiting to prevent DDoS attacks or abuse of Gemini API.

**Compliance Requirements**:

Designed to align with GDPR and Indian IT Act 2000 (with 2023 amendments) for data protection.

**Software Quality Attributes**

|  |  |
| --- | --- |
| Attribute | Description |
| Usability | Intuitive, minimal-click interface accessible to students, teachers, and researchers. |
| Reliability | Recovers gracefully from partial failures (e.g., AI fallback, offline alerts). |
| Maintainability | Modular FastAPI backend with clear routing structure and logging |
| Portability | Runs on any modern web browser; backend deployable on AWS, GCP, or local Linux servers |
| Interoperability | Supports standard file formats (PDF, DOCX, TXT) and integrates with Gemini, ChromaDB. |
| Testability | Unit-tested APIs; frontend components tested via Jest or Cypress. |
| Availability | Target availability is 24/7 with minimal maintenance downtime. |
| Scalability | Easily scales via Docker containers and Kubernetes (for future enterprise expansion). |
| Flexibility | Plugin-ready architecture for adding future features like collaboration, analytics, etc |

## **Business Rules**

Only authenticated users can upload and query documents.

Uploaded documents are private to each user unless shared explicitly in future collaborative modules.

Feedback submission is allowed only after a complete answer is received.

Multi-language responses must match the language selected in user preferences.

Voice input must be allowed only on browsers that support speech recognition.

Downloadable content (summaries or Q\&A reports) is restricted to the session owner.

Admin roles (future feature) can view anonymized usage analytics but not document contents

1. List of Stakeholders

Doctor – Uses the app to capture handwritten medical notes and convert them into digital format.

Hospital Admin / Medical Records Officer – Manages and reviews stored patient records.

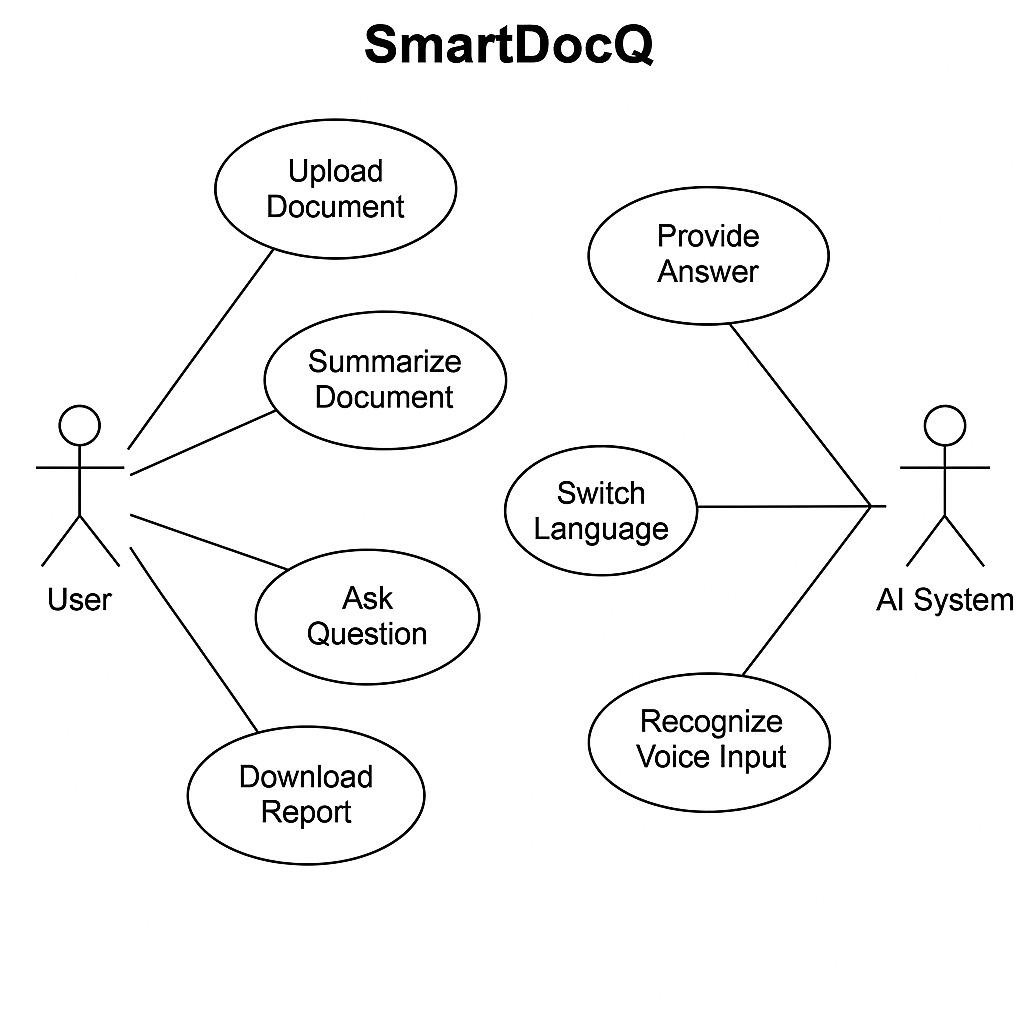
Patient – Can view their own digitized medical reports (optional feature depending on system setup).

System Administrator – Maintains the app, database, and integrations with EMR systems.

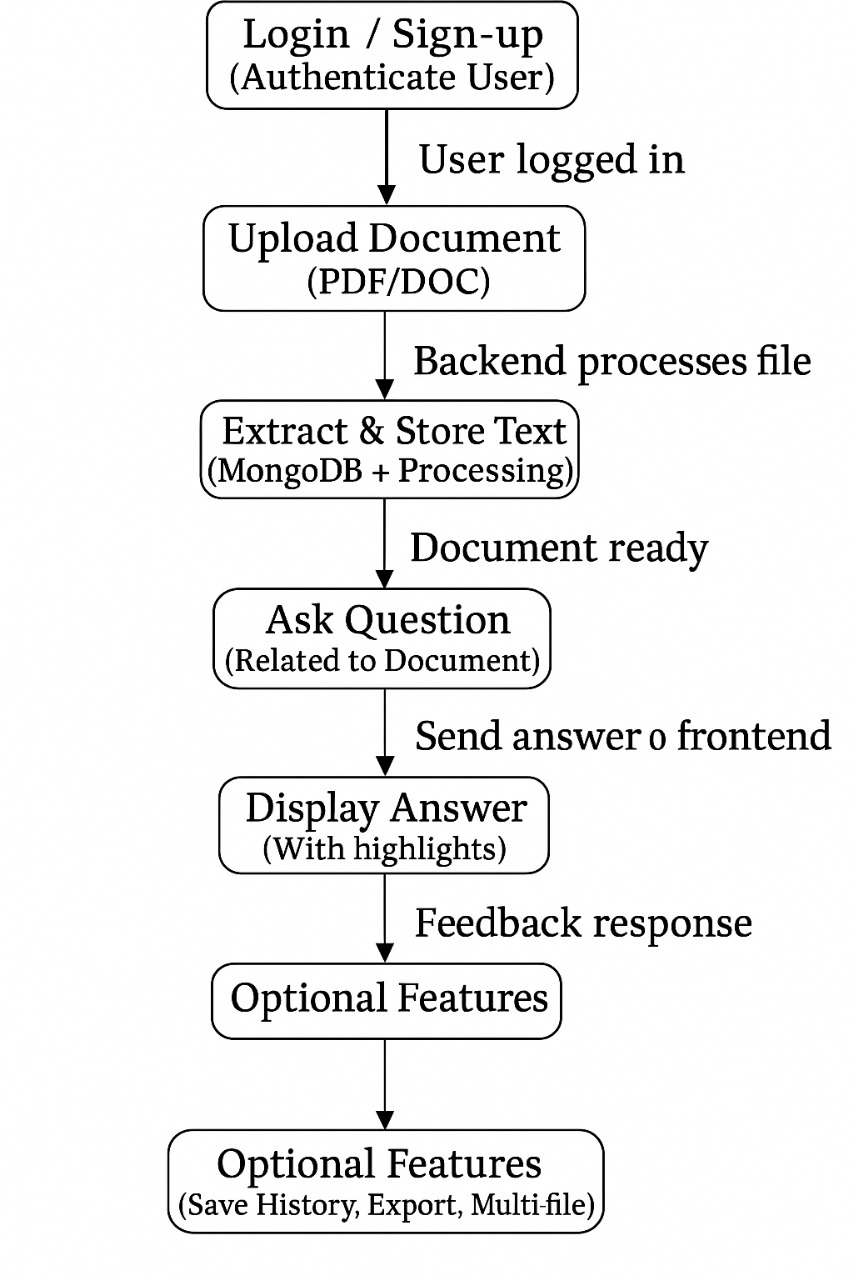
AI/OCR System (Backend Service) – Processes handwritten notes and returns structured text.

EMR System – Receives the digitized data from the app for permanent storage and further usage.

Use-Case for Each Stakeholder



workflow



MOCK SCREENS

