Smart Document Q&A Assistant - User Requirements Document

1. Project Overview

1.1 Project Name

DocuChat - Smart Document Q&A Assistant

1.2 Project Description

DocuChat is an intelligent document management and question-answering system that allows users to upload documents (PDFs, Word docs, text files) and interact with them through natural language queries. The system uses RAG (Retrieval Augmented Generation) technology powered by Langchain to provide accurate, context-aware answers from uploaded documents.

1.3 Project Goals

- Enable users to extract insights from documents without reading entire files
- Provide accurate, source-referenced answers to user queries
- Support multiple document formats and collections
- Offer document summarization and key insights extraction
- Create accessible web and mobile interfaces
- Demonstrate end-to-end ML/AI application development skills

1.4 Target Users

- Students researching academic papers
- Professionals managing multiple reports and documents
- Researchers analyzing literature
- Anyone needing quick information extraction from documents

---

2. User Personas

Persona 1: Sarah - Graduate Student

- **Age**: 25
- **Tech Savvy**: High
- **Need**: Quick research and citation finding from 50+ academic papers
- **Goal**: Save time by asking questions instead of reading entire papers

Persona 2: Mark - Business Analyst

- **Age**: 35
- **Tech Savvy**: Medium
- **Need**: Extract insights from quarterly reports and business documents
- **Goal**: Prepare presentations quickly with data from multiple sources

---

3. Functional Requirements

3.1 User Management

- **FR-1.1**: Users can register with email and password
- **FR-1.2**: Users can log in and log out securely
- **FR-1.3**: Users can reset forgotten passwords
- **FR-1.4**: Users have personal dashboard with their documents and chat history
- **FR-1.5**: User sessions persist across devices

3.2 Document Management

- **FR-2.1**: Users can upload documents (PDF, DOCX, TXT)
- **FR-2.2**: Maximum file size: 25MB per document
- **FR-2.3**: Users can organize documents into collections/folders
- **FR-2.4**: Users can view list of uploaded documents with metadata (name, size, upload date)
- **FR-2.5**: Users can delete documents
- **FR-2.6**: System extracts text from documents automatically
- **FR-2.7**: System creates vector embeddings for RAG functionality

- **FR-2.8**: Users can see document processing status (uploading, processing, ready)

3.3 Question & Answer System

- **FR-3.1**: Users can ask questions about their uploaded documents
- **FR-3.2**: Users can select specific documents or collections to query
- **FR-3.3**: System provides answers with source references (page numbers/sections)
- **FR-3.4**: Users can view chat history for each document/collection
- **FR-3.5**: Users can start new chat sessions
- **FR-3.6**: System indicates when processing/generating answers
- **FR-3.7**: Users can copy answers to clipboard
- **FR-3.8**: Users can rate answers (helpful/not helpful)

3.4 Document Insights

- **FR-4.1**: System generates automatic summary for each document
- **FR-4.2**: System extracts key topics/themes from documents
- **FR-4.3**: Users can view document statistics (word count, estimated read time)
- **FR-4.4**: System provides suggested questions for each document

3.5 Search and Filter

- **FR-5.1**: Users can search documents by name
- **FR-5.2**: Users can filter documents by upload date
- **FR-5.3**: Users can filter documents by collection
- **FR-5.4**: Users can search through chat history

---

4. Non-Functional Requirements

4.1 Performance

- **NFR-1.1**: Document upload should complete within 30 seconds for files up to 25MB
- **NFR-1.2**: Question responses should be generated within 10 seconds
- **NFR-1.3**: Web pages should load within 2 seconds
- **NFR-1.4**: System should support 100 concurrent users

4.2 Security

- **NFR-2.1**: All passwords must be hashed using bcrypt
- **NFR-2.2**: API endpoints must be authenticated using JWT tokens
- **NFR-2.3**: HTTPS must be enforced in production
- **NFR-2.4**: User documents must be private (not accessible by other users)
- **NFR-2.5**: Uploaded files must be scanned for malware

4.3 Usability

- **NFR-3.1**: Interface must be intuitive with minimal learning curve
- **NFR-3.2**: Mobile app must be responsive and work on Android 8.0+
- **NFR-3.3**: Error messages must be clear and actionable
- **NFR-3.4**: System must provide loading indicators for all async operations

4.4 Reliability

- **NFR-4.1**: System uptime should be 99%+
- **NFR-4.2**: System should handle errors gracefully without crashing
- **NFR-4.3**: Automated backups of database every 24 hours

4.5 Scalability

- **NFR-5.1**: Architecture should support horizontal scaling
- **NFR-5.2**: Database should be optimized for growth
- **NFR-5.3**: Vector database should handle 10,000+ documents

---

5. Technical Requirements

5.1 Backend Stack

- **Python 3.9+** - Core programming language
- **Flask** - Web framework and API
- **Langchain** - RAG implementation
- **OpenAI API / Ollama** - LLM for answer generation
- **ChromaDB / Pinecone** - Vector database for embeddings

- **PostgreSQL** - Relational database for user data
- **SQLAlchemy** - ORM for database operations
- **PyPDF2 / pdfplumber** - PDF text extraction
- **python-docx** - Word document processing
- **JWT** - Authentication tokens
- **Flask-CORS** - Cross-origin resource sharing

5.2 Frontend Stack (Web)

- **HTML5, CSS3, JavaScript**
- **Bootstrap 5** or **Tailwind CSS** - UI framework
- **Fetch API** - Backend communication
- **Local Storage** - Token management

5.3 Mobile Stack

- **Android Studio** - Development environment
- **Java/Kotlin** - Programming language
- **Retrofit** - API communication
- **Material Design** - UI components

5.4 DevOps & Tools

- **Git & GitHub** - Version control
- **Digital Ocean** - Cloud hosting (Droplet)
- **Gunicorn** - WSGI server
- **Nginx** - Reverse proxy
- **Jupyter Notebook** - ML experimentation
- **PyCharm / Cursor** - Development IDEs
- **Postman** - API testing

5.5 Machine Learning Components

- **Sentence Transformers** - Text embeddings
- **HuggingFace Models** - Summarization models
- **TF-IDF / spaCy** - Keyword extraction

---

6. User Stories

Epic 1: User Authentication

- **US-1.1**: As a new user, I want to register an account so I can save my documents
- **US-1.2**: As a registered user, I want to log in so I can access my documents
- **US-1.3**: As a user, I want to reset my password if I forget it

Epic 2: Document Upload

- **US-2.1**: As a user, I want to upload PDF documents so I can ask questions about them
- **US-2.2**: As a user, I want to see upload progress so I know when my document is ready
- **US-2.3**: As a user, I want to organize documents into collections so I can manage them better
- **US-2.4**: As a user, I want to delete documents I no longer need

Epic 3: Question & Answer

- **US-3.1**: As a user, I want to ask questions about my documents in natural language
- **US-3.2**: As a user, I want to see which part of the document my answer comes from
- **US-3.3**: As a user, I want to view my previous questions and answers
- **US-3.4**: As a user, I want to select multiple documents to query at once

Epic 4: Document Insights

- **US-4.1**: As a user, I want to see an automatic summary of my document
- **US-4.2**: As a user, I want to see key topics in my document
- **US-4.3**: As a user, I want suggested questions to help me explore the document

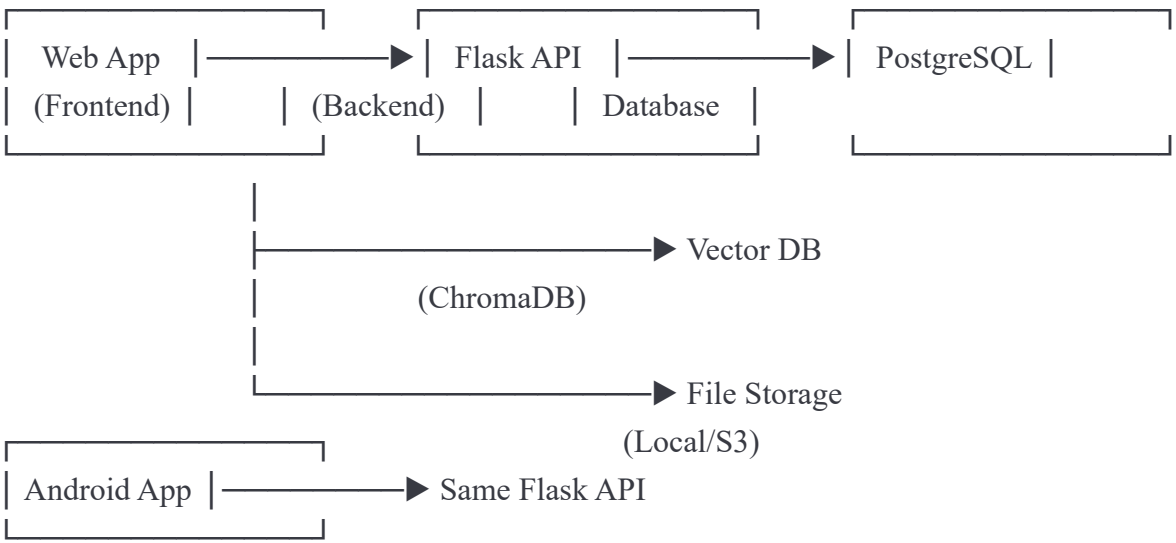
Epic 5: Mobile Access

- **US-5.1**: As a user, I want to access my documents from my Android phone
- **US-5.2**: As a user, I want to upload documents from my phone
- **US-5.3**: As a user, I want to ask questions on-the-go

---

7. System Architecture

7.1 High-Level Architecture



7.2 RAG Pipeline

Document Upload → Text Extraction → Chunk Text →
Generate Embeddings → Store in Vector DB

User Query → Generate Query Embedding →
Similarity Search → Retrieve Relevant Chunks →
Send to LLM with Context → Generate Answer

---

8. Database Schema

8.1 Users Table


sql

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP
);
```

8.2 Documents Table


sql

```sql
CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    filename VARCHAR(255) NOT NULL,
    file_path VARCHAR(500) NOT NULL,
    file_size INTEGER,
    file_type VARCHAR(50),
    status VARCHAR(50) DEFAULT 'processing',
    summary TEXT,
    upload_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    collection_id INTEGER REFERENCES collections(id)
);
```

8.3 Collections Table

sql

```sql
CREATE TABLE collections (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    name VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

8.4 Chat_History Table

sql

```sql
CREATE TABLE chat_history (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    document_id INTEGER REFERENCES documents(id),
    question TEXT NOT NULL,
    answer TEXT NOT NULL,
    sources TEXT,
    rating INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

9. API Endpoints

9.1 Authentication

- `POST /api/auth/register` - Register new user
- `POST /api/auth/login` - User login
- `POST /api/auth/logout` - User logout
- `POST /api/auth/reset-password` - Password reset

9.2 Documents

- `GET /api/documents` - Get all user documents
- `POST /api/documents/upload` - Upload new document
- `DELETE /api/documents/<id>` - Delete document

- `GET /api/documents/<id>` - Get document details
- `GET /api/documents/<id>/summary` - Get document summary

9.3 Collections

- `GET /api/collections` - Get all collections
- `POST /api/collections` - Create new collection
- `PUT /api/collections/<id>` - Update collection
- `DELETE /api/collections/<id>` - Delete collection

9.4 Q&A

- `POST /api/query` - Ask question about documents
- `GET /api/chat-history/<document_id>` - Get chat history
- `POST /api/chat-history/<id>/rate` - Rate an answer

---

10. Development Phases

Phase 1: Backend Foundation (Week 1-2)

- Set up Flask project structure
- Implement user authentication (register, login, JWT)
- Create PostgreSQL database and models
- Basic API endpoints for users

Phase 2: Document Processing (Week 2-3)

- Implement file upload functionality
- Text extraction from PDF/DOCX/TXT
- Set up vector database (ChromaDB)
- Create embeddings pipeline

Phase 3: RAG Implementation (Week 3-4)

- Integrate Langchain
- Implement semantic search

- Connect to LLM (OpenAI/Ollama)
- Build Q&A pipeline with source attribution

Phase 4: Advanced Features (Week 4-5)

- Document summarization
- Key topics extraction
- Collections management
- Chat history

Phase 5: Web Frontend (Week 5-6)

- Create responsive UI
- Implement all screens (login, dashboard, upload, chat)
- Connect to backend API
- Add error handling and loading states

Phase 6: Android App (Week 6-8)

- Set up Android Studio project
- Implement authentication screens
- Create document management UI
- Implement chat interface
- Connect to Flask API

Phase 7: Deployment & Testing (Week 8-9)

- Deploy backend to Digital Ocean
- Set up Nginx and Gunicorn
- Configure SSL certificates
- Testing and bug fixes
- Documentation

---

11. Success Metrics

11.1 Technical Metrics

- API response time < 10 seconds for queries
- Document processing success rate > 95%
- System uptime > 99%

11.2 User Experience Metrics

- Answer accuracy (based on user ratings) > 80%
- Average session duration > 10 minutes
- User retention rate > 60% after first week

11.3 Learning Objectives

- Successfully implement RAG using Langchain
- Deploy full-stack application to cloud
- Create functional Android application
- Demonstrate proficiency in Python, Flask, ML, and web/mobile development

---

12. Future Enhancements (Post-MVP)

- Multi-language document support
- Voice input for questions
- Document comparison feature
- Collaborative collections (share with team)
- Export answers to PDF/Word
- Browser extension for quick document uploads
- Integration with Google Drive, Dropbox
- Advanced analytics dashboard
- Fine-tuned models for specific domains

---

13. Risks and Mitigation

Risk	Impact	Probability	Mitigation
LLM API costs exceed budget	High	Medium	Use Ollama (local) for development, implement rate limiting
Document processing fails for complex PDFs	Medium	High	Implement fallback extractors, error handling
Vector DB performance degrades	High	Low	Optimize chunk size, implement caching
Security vulnerabilities	High	Medium	Follow OWASP guidelines, regular security audits
Mobile app complexity	Medium	Medium	Start with core features, iterate gradually

---

Document Version Control

- **Version**: 1.0
- **Last Updated**: October 20, 2025
- **Author**: Project Team
- **Status**: Approved