DOCUMENT READER

Transforming Static Documents into Dynamic Conversations
Unlock Insights Effortlessly with Agentic AI & MCP Coordination

www.documentreader.com

Project Overview

The Document Reader Chatbot System with Agentic MCP is an advanced, intelligent solution designed to transform static documents into dynamic, interactive conversations. It empowers users to seamlessly upload documents in formats such as PDF, DOCX, and TXT, and then interact with their content using a natural language chatbot interface. Behind the scenes, the system leverages an agentic AI architecture coordinated via a well-defined Model Context Protocol (MCP), enabling autonomous and goal-driven operations. The system intelligently parses document text and metadata, converts it into semantic embeddings, and stores structured data in a relational database and vector store. With this foundation, the chatbot delivers accurate, context-aware responses to user queries, while continuously tracking session context for a coherent multi-turn conversation. Designed for scalability and security, the system provides role-based access control, encrypted storage, and API endpoints for integration with external applications. The overall solution simplifies knowledge extraction from large volumes of documents, enabling businesses to save time and enhance decision-making through intelligent automation.

Problem Statement

In today's data-driven world, businesses and professionals handle a large volume of unstructured documents such as contracts, reports, manuals, and research papers. Extracting meaningful insights from these documents is a time-consuming, manual process that requires domain expertise and effort. Traditional document management systems store files in a passive manner, providing little to no interactive capability for searching or understanding document content. As a result, users struggle to quickly find relevant information, leading to inefficient workflows, delayed decisions, and knowledge silos. Furthermore, existing chatbot solutions are typically limited to predefined datasets or FAQs, lacking the ability to dynamically interpret new document content in a meaningful way.

Methodology

04

Document Ingestion & Upload

Document Parsing & Text Extraction

Embedding Generation & Vector Indexing

Natural Language Understanding (NLU) & Question Processing

Model Context Protocol (MCP) for Agent Coordination

Data Storage (PostgreSQL + Vector Database)

Contextual Chatbot Response Generation

Authentication & Role-Based Access Control (RBAC)

Analytics & Monitoring System

Continuous Autonomous Operations & Updates

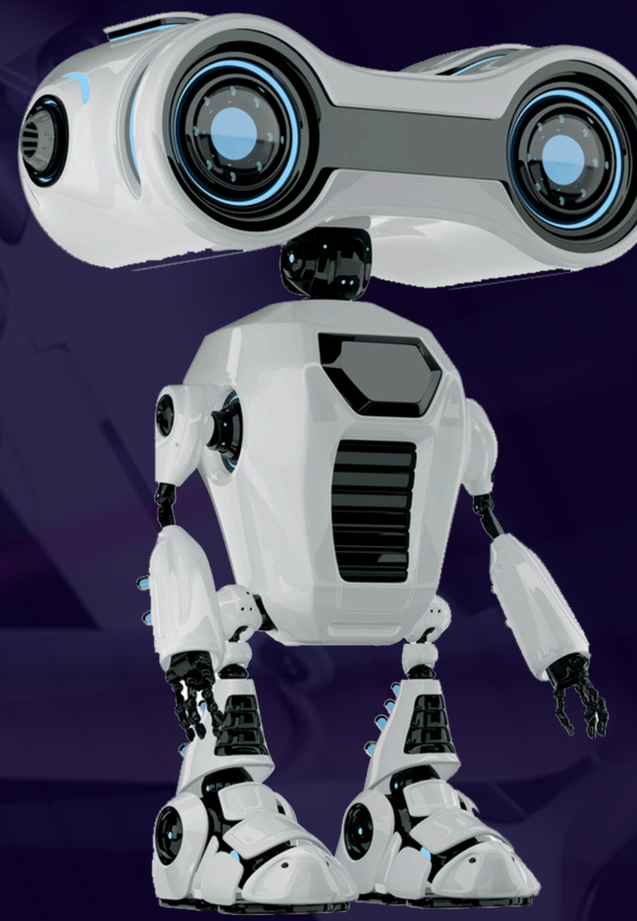
API Design for Integration

Deployment & Scalability Considerations

Data Collection and Preprocessing

The system collects data by allowing users to upload documents in formats such as PDF, DOCX, and TXT through a user-friendly web interface or secure API endpoints. Each document is assigned a unique ID and stored in a reliable storage system (e.g., Blob Storage or File System). Along with the file itself, metadata such as user ID, filename, upload timestamp, and document type are collected automatically. This structured metadata is stored in a relational database (PostgreSQL) to track document ownership, upload history, and facilitate easy retrieval in the future.

Once uploaded, the document enters the preprocessing pipeline. The Document Parser Agent extracts raw text and metadata from the document using specialized parsers. The extracted text is then cleaned by removing unnecessary elements like headers, footers, page numbers, and special characters, and normalized to a consistent format. Afterwards, the text is split into smaller logical chunks (paragraphs or sections) for efficient processing. Each chunk, along with relevant metadata, is stored in a structured database table. In parallel, a semantic embedding is generated for each text chunk using an embedding model, and the resulting vector is stored in a dedicated Vector Database to enable fast, semantic search during question answering.



Model Development

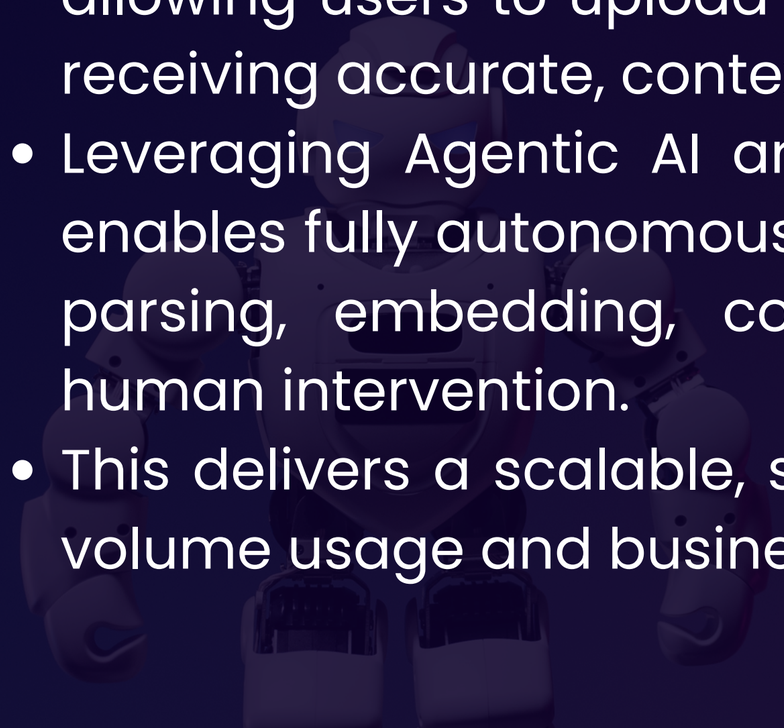
The model development focuses on creating an intelligent system capable of understanding document content and providing context-aware answers to user queries. The core of this solution is a combination of specialized models coordinated via the Model Context Protocol (MCP). First, the Document Parsing Model extracts structured text from unstructured documents using tools like PDFMiner (for PDFs) or python-docx (for Word files). The text is then preprocessed and segmented into meaningful chunks for downstream tasks. Next, the Embedding Generation Model converts each text chunk into high-dimensional semantic vectors using pre-trained models such as OpenAI Embeddings or Sentence Transformers. These embeddings enable efficient and accurate semantic similarity search later during question answering.

The heart of the system is the Conversational Question Answering Model, which receives the user query along with a set of contextually relevant document chunks (retrieved by vector similarity search). Using large language models (LLMs), the system generates a human-like, context-aware response based on both the query and the document context. Additionally, a Context Management Model tracks the entire user session, storing the history of questions and answers to maintain a coherent multi-turn conversation. The MCP ensures seamless coordination between the different models, passing structured context between them so that each model operates independently but in harmony, enabling autonomous and intelligent behavior throughout the system.

. Business Objective

02

- In modern enterprises, the volume of unstructured documents—contracts, reports, manuals, and whitepapers—is growing exponentially. Manually sifting through this data to extract actionable insights is inefficient, error-prone, and time-consuming.
- Our system’s goal is to enable effortless interaction with documents by allowing users to upload files and directly ask natural language questions, receiving accurate, context-driven responses in real time.
- Leveraging Agentic AI architecture and a Model Context Protocol (MCP) enables fully autonomous behavior where individual intelligent agents (e.g., parsing, embedding, context tracking) seamlessly coordinate without human intervention.
- This delivers a scalable, secure, and intelligent solution designed for high-volume usage and business agility.



. System Overview

The system is designed as a document-centric, conversational intelligence platform, powered by autonomous agents operating under a strict Model Context Protocol (MCP).

At its core, it consists of the following integrated components:

- 📤 Document Upload API & UI: Allows easy file upload through a user-friendly interface or API.
- 📑 Document Parser Agent: Automatically extracts structured text and document metadata.
- ⚡ Embedding & Vector Store Agent: Converts text into semantic vector representations for fast retrieval.
- 🧱 Context Management Agent: Maintains conversation state and session context.
- 🧠 Chatbot Q&A Agent: Generates intelligent, context-aware answers using LLMs.
- 📊 Analytics & Monitoring Agent: Provides real-time dashboards with usage metrics, query patterns, and system health.
- 🔗 API Gateway & Authentication Layer: Manages secure external integrations and user authentication.

. User Roles & Personas

👥 User Roles & Personas (Simplified)

- End User: Uploads documents and asks questions in natural language to get quick answers.
- Admin: Manages users, monitors system usage, and configures system settings.
- System Agent: Automatically handles tasks like parsing documents, generating embeddings, tracking conversation context, and providing answers, without needing human help.

✅ . Functional Requirements

- Users can upload documents in PDF, DOCX, or TXT formats via an intuitive UI or secure API.
- The system automatically extracts text and metadata from uploaded documents without manual effort.
- Structured document data is stored in PostgreSQL, while semantic vector embeddings are stored in a Vector Database for fast retrieval.
- Parsed text is converted into high-dimensional embeddings using pretrained models (e.g., OpenAI Embeddings, Sentence Transformers).
- The chatbot generates answers dynamically by using contextual embeddings combined with large language models (LLMs).

✅ . Functional Requirements

- The system tracks the full session history (questions and answers) to maintain coherent multi-turn conversations.
- Autonomous system agents proactively index new documents, update embeddings, and manage stale data without manual intervention.
- On-demand document summarization feature provides automatic summaries of the document content.
- An analytics dashboard visualizes key metrics like active users, most asked questions, document access patterns, and agent health.
- Provides secure API endpoints for document upload, querying, and session management for external integrations.
- The Model Context Protocol (MCP) ensures well-defined, deterministic coordination between intelligent agents, enabling seamless autonomous workflows.

. Non-Functional Requirements

- The system is designed for scalability, capable of handling up to 100 concurrent document uploads and 1000 simultaneous chat sessions without degradation in performance.
- It ensures high performance, with a target query response time of ≤ 500 milliseconds after a document has been processed and indexed.
- The system provides high availability, aiming for 99.9% uptime, which is critical for enterprise-grade, mission-critical operations.
- Security is enforced through TLS-encrypted communication, JWT-based authentication, and Role-Based Access Control (RBAC) to restrict actions based on user roles.

. Non-Functional Requirements

- Strong data consistency is maintained by ensuring that the structured data in the relational PostgreSQL database and the semantic vectors in the Vector Database remain fully synchronized at all times.
- The system is designed for extensibility, making it easy to support additional file types, new embedding models, or integrate new intelligent agents in the future without major architecture changes.

. Data Flow – PSSQL (Process → State → Structure → Query Language)

Document Upload

- The user uploads a document through the interface or API. The system receives the document and stores the file in Blob Storage or the File System. At the same time, document metadata like document ID, user ID, filename, and upload time is saved in a metadata table in PostgreSQL.

Parsing & Structuring

- The Document Parser Agent reads the uploaded document and extracts structured text and metadata automatically. The cleaned and structured content is stored in the documents table in PostgreSQL.

. Data Flow – PSSQL (Process → State → Structure → Query Language)

Embedding Generation

- Each chunk of parsed text is processed by the Embedding Model, which generates a high-dimensional semantic vector. These vectors are stored in a specialized Vector Database to enable fast semantic search later.

Question-Answer Process

- When the user asks a question, the Context Manager Agent retrieves the most relevant document vectors by querying the semantic vector index. The top matching documents are selected, combined into context, and sent to a language model (LLM) to generate an intelligent answer.

. Data Flow – PSSQL (Process → State → Structure → Query Language)

Context Tracking

- Each question and its corresponding answer are saved in the session_context table, allowing the system to maintain the full conversation history and provide coherent multi-turn responses as the session progresses.

. User Stories (Agile Format)

- As an end-user, I want to upload a document via the user interface or API, so that I can ask questions about the document content.
- As an end-user, I want to ask questions in natural language and receive accurate, context-aware answers based on the document.
- As an admin, I want to view analytics about document uploads and user activity, so I can monitor system usage and performance trends.
- As a system agent, I want to automatically process and index newly uploaded documents without needing manual intervention.
- As a system, I want to store embedding vectors of document content, so that I can perform fast and accurate semantic searches when answering questions.

. Definition of Done (DoD)

- ✅ Document upload working with supported formats (PDF, DOCX, TXT).
- ✅ Parsing pipeline successfully extracts and stores structured data in PostgreSQL.
- ✅ Embedding vectors generated and stored in Vector DB.
- ✅ Chatbot delivers intelligent responses based on context.
- ✅ Session context (multi-turn conversations) tracked and stored.
- ✅ MCP handles agent coordination in a seamless, autonomous manner.
- ✅ Authentication & role-based access control fully implemented (JWT/RBAC).
- ✅ Analytics dashboard displays system health and usage metrics.

. Security & Privacy Requirements

- 🔐 TLS Encryption: All communications between clients, APIs, and the system are secured using HTTPS to prevent data interception.
- 🗝️ JWT Authentication: Secure token-based authentication ensures that only authorized users and systems can access the APIs and services.
- 👥 Role-Based Access Control (RBAC): Access is restricted based on roles (Admin, User, System Agent), ensuring that each user type has appropriate permissions.
- 🔒 Data Encryption: Sensitive data, including documents and session history, is stored in encrypted form to protect it at rest.
- ✅ GDPR Compliance: The system supports features for deleting documents and personal data, ensuring compliance with data privacy regulations like GDPR.

✅ 🔟. Analytics & Monitoring

- 👫 Active Users: Tracks the number of unique user sessions over time to monitor system engagement.
- 📂 Documents Uploaded: Monitors document upload statistics, showing trends by time period (daily, weekly, monthly).
- ❓ Most Asked Questions: Keeps track of the most frequently asked questions to analyze user needs and improve the system.
- ⏱️ System Latency: Measures the average time taken per API operation to ensure system performance remains optimal.
- ⚙️ Agent Activity: Logs detailed actions performed by each system agent (e.g., parsing, embedding generation, answering) for monitoring and debugging purposes.

. Next Steps

1. ✅ Prioritize User Stories → Build a detailed Product Backlog.
2. ✅ Conduct Sprint Planning → Define clear goals for the first MVP sprint.
3. ✅ Create a System Architecture Diagram → Visualize agents, databases, and API flow.
4. ✅ Define API Contracts → OpenAPI specification for all endpoints.
5. ✅ Start Implementation → Build MVP focusing on document upload, parsing, embedding, and chatbot answering.
6. ✅ Setup Monitoring Tools → Prometheus for metrics, Grafana dashboards for real-time monitoring.