Intelligent PDF Query Application

A System Design and Implementation Report

Abhyudaya Tiwari
IIT Jodhpur
b23cs1085@iitj.ac.in

September 15, 2025

Contents

Abstract

This report details the design, architecture, and implementation of a full-stack MERN application for intelligent PDF document management and querying. The system leverages Google's Generative AI, a Retrieval-Augmented Generation (RAG) pipeline, and MongoDB Atlas Vector Search to provide a sophisticated, dual-mode chat interface. The architecture is designed to be scalable and efficient, handling both small and large documents by dynamically choosing between a fast, summary-based query method and a more accurate, deep-content RAG pipeline. This document covers the system architecture, a detailed backend pipeline explanation, data design, component breakdown, and the rationale behind the chosen technologies.

1 System Architecture

The application is built on a robust three-tier architecture, separating the client, server, and data layers to ensure maintainability and scalability. This model is augmented by external AI services, which act as a fourth, specialized processing layer for advanced tasks like summarization, embedding generation, and conversational AI.

1.1 Overall Architecture Flowchart

The following flowchart illustrates the high-level interaction between the system's main components, from the user's browser to the backend services and database.

2 Backend Pipeline Explained

The core of the backend logic resides in its ability to handle PDF documents through two distinct, event-driven pipelines: an asynchronous Ingestion Pipeline for processing large documents, and a real-time Query Pipeline for handling user chat messages.

2.1 Ingestion Pipeline

The Ingestion Pipeline is a one-time, asynchronous process dedicated to preparing large PDF documents for efficient querying via Retrieval-Augmented Generation (RAG). This process is handled by the 'ragController.js'.

1. **Text Extraction:** Upon upload, the PDF file is read from storage, and its raw text content is extracted page by page.
2. **Chunking:** The full text is passed to a 'RecursiveCharacterTextSplitter' from LangChain, which breaks the content into smaller, semantically coherent chunks with overlap to preserve context.
3. **Embedding Generation:** Each text chunk is converted into a high-dimensional vector embedding using Google's 'text-embedding-004' model. This vector numerically represents the semantic meaning of the chunk.

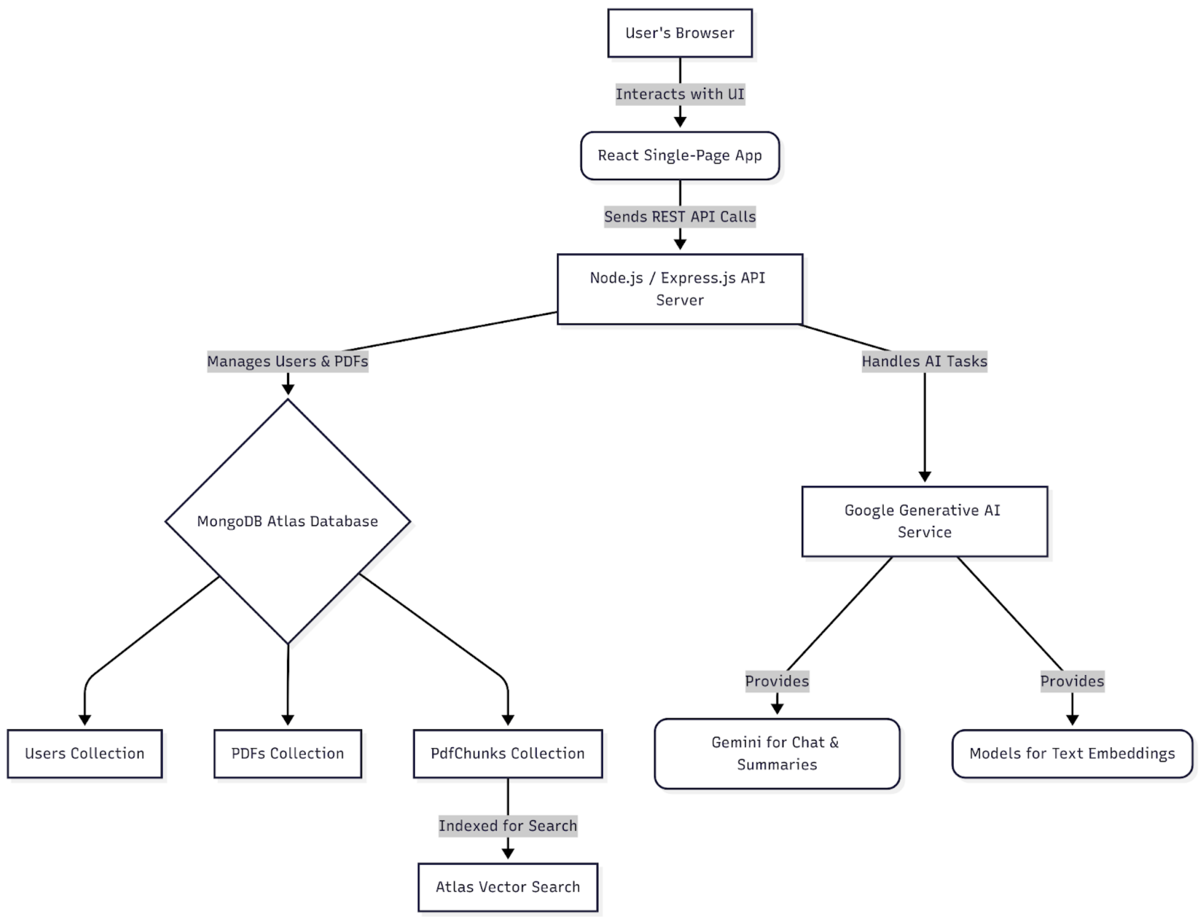


Figure 1: High-Level System Architecture

4. **Storage:** The chunks and their corresponding vector embeddings are stored in the ‘PdfChunk‘ collection in MongoDB, with a reference back to the parent PDF document.

5. **Status Update:** Finally, the parent document’s ‘isChunked‘ flag is set to ‘true‘, signaling that it is ready for the RAG query pipeline.

2.2 Query Pipeline

The Query Pipeline, managed by the ‘pdfController.js‘, is a conditional workflow that activates each time a user sends a chat message. It intelligently selects the best method to answer the user’s question.

1. **MCP / Tool-Using Path:** The system first checks if the user’s message contains a URL. If so, it prioritizes the tool-using pipeline. The Gemini model is invoked, identifies the need to use the `get_webpage_content` tool, and returns a function call request. The backend executes this function, scrapes the web content, and sends the results back to the model to generate a final, context-aware answer.

2. **RAG Path (for Large PDFs):** If no URL is present and the PDF’s ‘isChunked‘ flag is ‘true‘, the RAG pipeline is initiated. The user’s question is converted into a vector

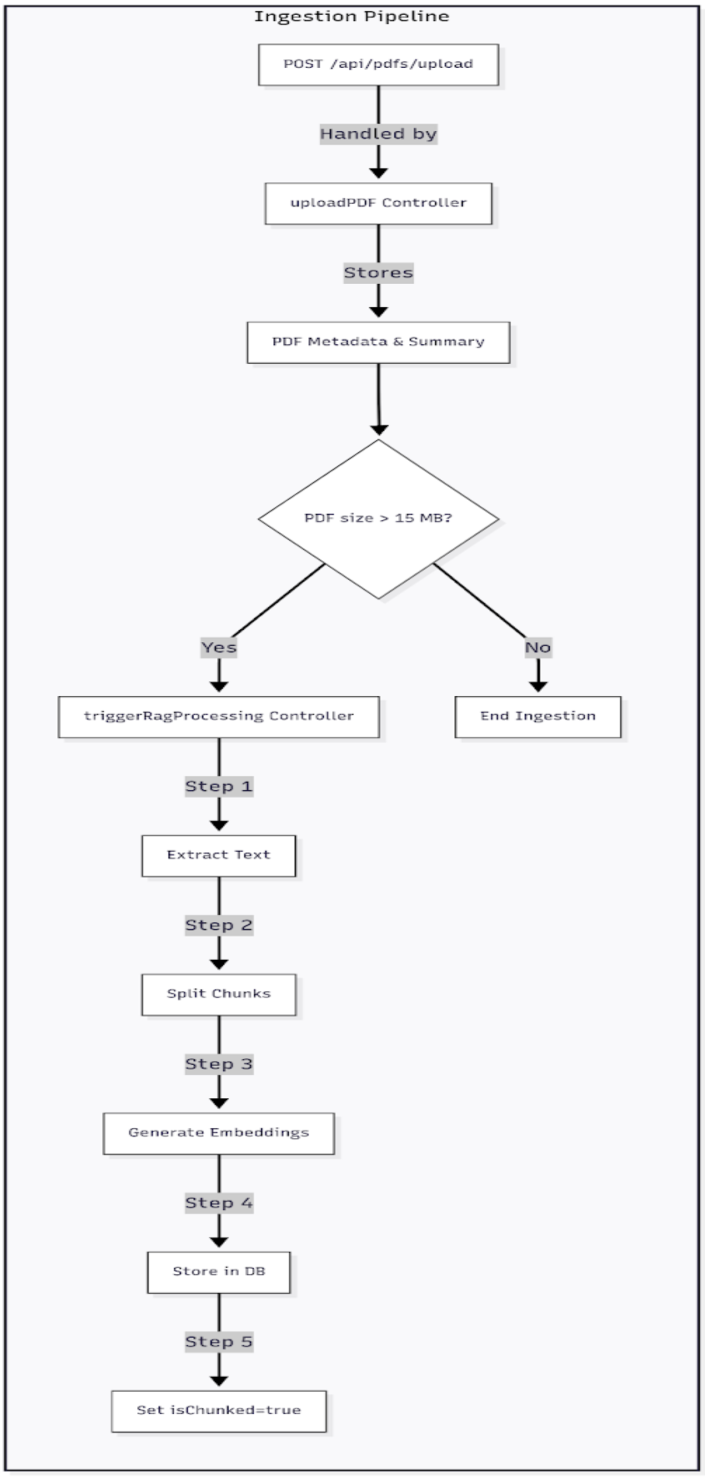


Figure 2: The Asynchronous PDF Ingestion Pipeline

embedding. This vector is then used to perform a similarity search in MongoDB Atlas against the indexed chunks of the specific PDF. The most relevant chunks are retrieved and used to augment a prompt for the Gemini model, which then generates a highly accurate answer based on the provided context.

3. **Standard Path (for Small PDFs):** If the PDF is not chunked, the system uses the

default pipeline. It relies on the pre-generated summary of the PDF and the chat history to provide context to the Gemini model for a fast and efficient response.

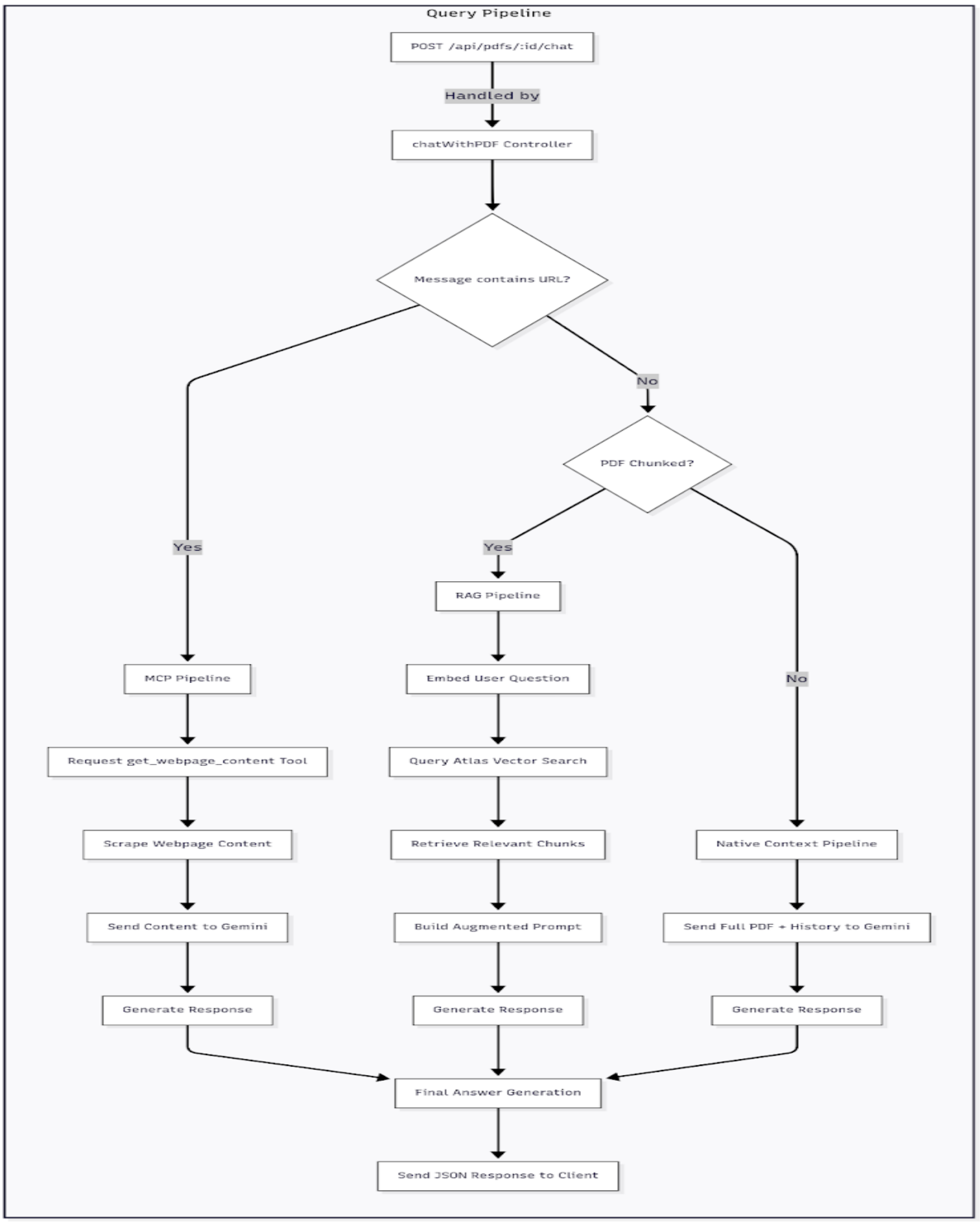


Figure 3: The Real-Time User Query Pipeline

3 Data Design

The data is modeled with three core Mongoose schemas to represent the application's entities and their relationships within MongoDB.

- **User Schema**: Stores essential user credentials and establishes ownership over documents. It contains an array of 'pdfs', linking one user to many PDF documents.
- **PDF Schema**: Acts as the central metadata store for each document. It references its owner via a 'user' ObjectID and, crucially, contains the 'isChunked' boolean flag. This flag is the decider for which backend pipeline to use during a chat session. Chat history is embedded within this document for simplicity in retrieval.
- **PdfChunk Schema**: This collection is the foundation of the RAG pipeline. By storing each text chunk and its vector 'embedding' in a separate document, we avoid MongoDB's 16MB document size limit for large PDFs and enable highly efficient, targeted vector queries. Each chunk maintains a reference to its parent 'pdfId'.

3.1 Entity-Relationship (ER) Diagram

The following diagram illustrates the relationships between the User, PDF, and PdfChunk collections.

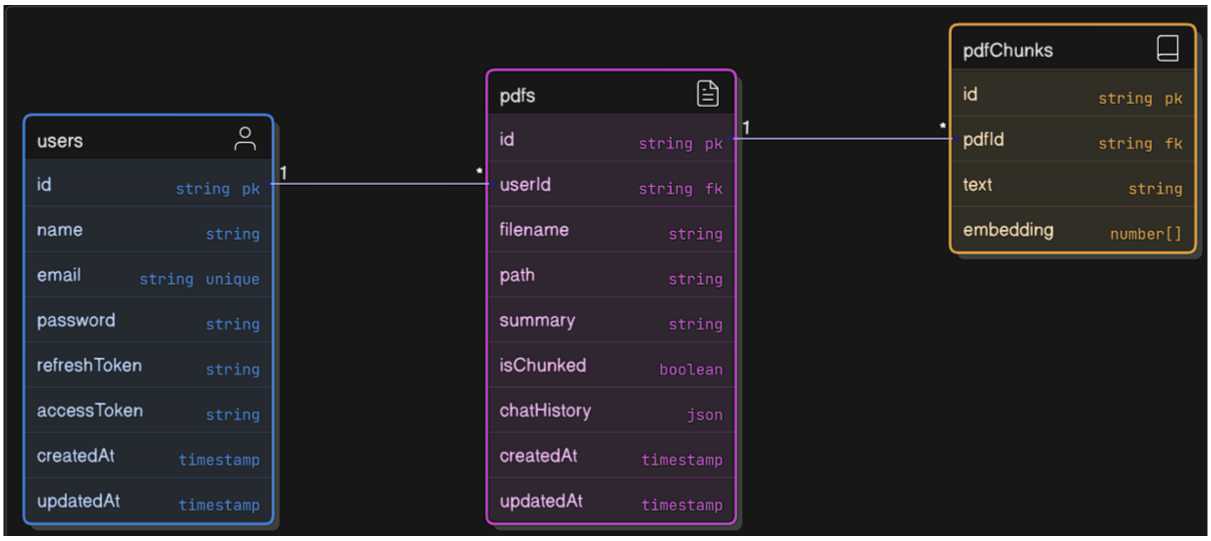


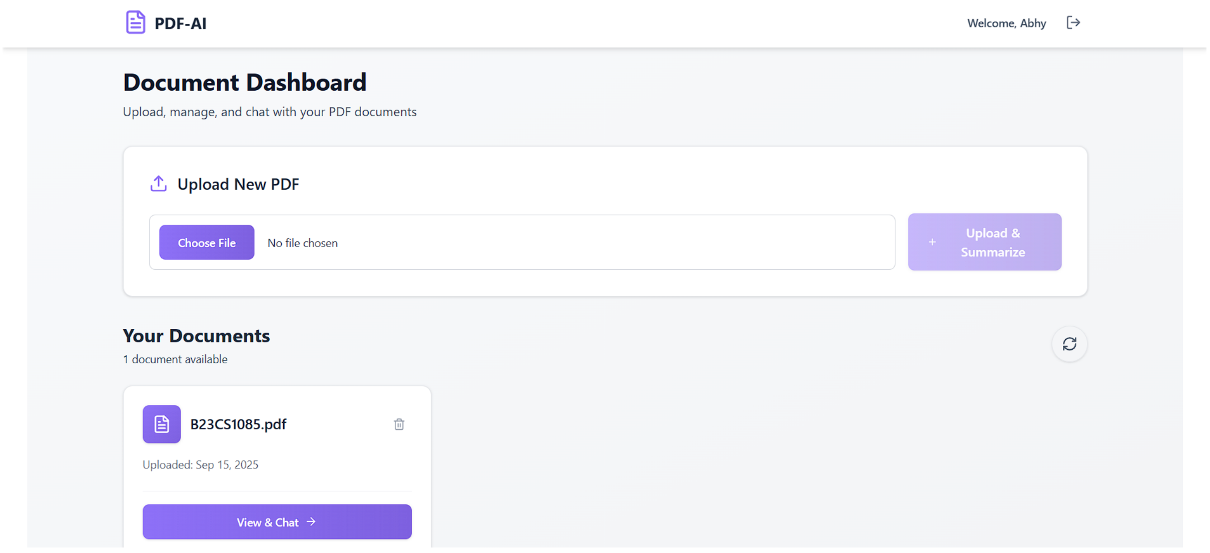
Figure 4: Data Model ER Diagram

4 User Interface (UI)

The frontend is a modern and responsive React single-page application designed for a seamless user experience.

4.1 Dashboard Page

The main dashboard serves as the central hub for document management. It lists all of the user's uploaded PDFs and provides functionality for uploading new documents and deleting existing ones.



PDF-AI Welcome, Abhy

Document Dashboard

Upload, manage, and chat with your PDF documents

Upload New PDF

Choose File No file chosen

Upload & Summarize

Your Documents

1 document available

B23CS1085.pdf

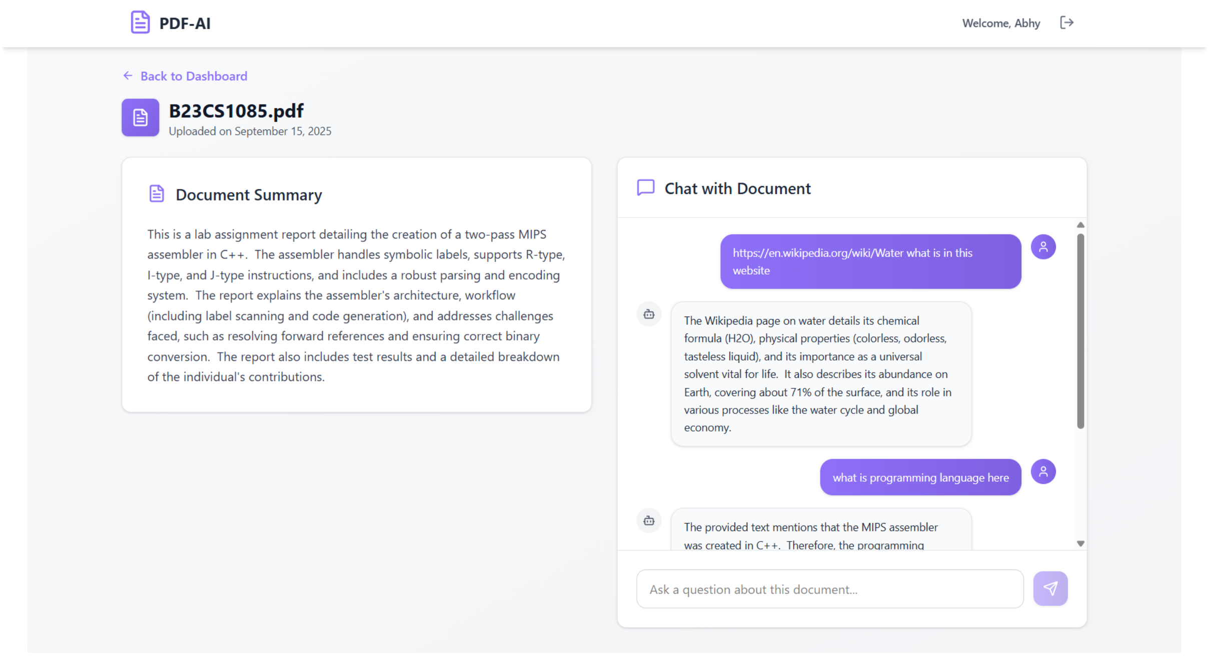
Uploaded: Sep 15, 2025

View & Chat

Figure 5: The User Dashboard

4.2 Chatbox Page

The chat page provides the core interactive experience, allowing users to ask questions in natural language and receive intelligent, context-aware answers.



PDF-AI Welcome, Abhy

Back to Dashboard

B23CS1085.pdf

Uploaded on September 15, 2025

Document Summary

This is a lab assignment report detailing the creation of a two-pass MIPS assembler in C++. The assembler handles symbolic labels, supports R-type, I-type, and J-type instructions, and includes a robust parsing and encoding system. The report explains the assembler's architecture, workflow (including label scanning and code generation), and addresses challenges faced, such as resolving forward references and ensuring correct binary conversion. The report also includes test results and a detailed breakdown of the individual's contributions.

Chat with Document

https://en.wikipedia.org/wiki/Water what is in this website

The Wikipedia page on water details its chemical formula (H2O), physical properties (colorless, odorless, tasteless liquid), and its importance as a universal solvent vital for life. It also describes its abundance on Earth, covering about 71% of the surface, and its role in various processes like the water cycle and global economy.

what is programming language here

The provided text mentions that the MIPS assembler was created in C++. Therefore, the programming

Ask a question about this document...

Figure 6: The PDF Chat Interface

5 Technology Stack and Justification

The technologies for this project were selected to create a modern, scalable, and efficient full-stack application.

Node.js / Express.js Chosen for its non-blocking, event-driven architecture, ideal for an I/O-heavy application that handles file uploads and API calls.

React.js A powerful library for building dynamic, responsive single-page applications. Its component-based architecture makes the UI manageable.

MongoDB A flexible NoSQL database that works seamlessly with JavaScript. Its document-based model is well-suited for the application's data structures.

MongoDB Atlas Vector Search This was a critical architectural choice. It integrates vector search directly into the primary database, simplifying the architecture, reducing costs, and enabling powerful pre-filtered vector queries.

Google Generative AI (Gemini) Provides high-quality models for text generation, summarization, and state-of-the-art text embeddings for the RAG pipeline.

LangChain.js Acts as a crucial abstraction layer that simplifies the implementation of the RAG pipeline, providing robust tools for text splitting and embedding model integration.

6 Conclusion

This project successfully demonstrates the integration of modern AI capabilities into a full-stack web application. By implementing a sophisticated, conditional backend pipeline, the system offers both speed for small documents and high accuracy for large ones, providing an intelligent and intuitive user experience for document interaction. The use of an integrated vector database and a modular design creates a solid foundation for future enhancements.