

Today, we'll learn how to build a **RAG** application that lets you chat with your PDF files. Using Langchain and Node.js, we'll create a system that can read and understand your documents, allowing you to ask questions and get answers directly from the text. This guide will cover everything from setting up the project to querying your PDFs effectively.

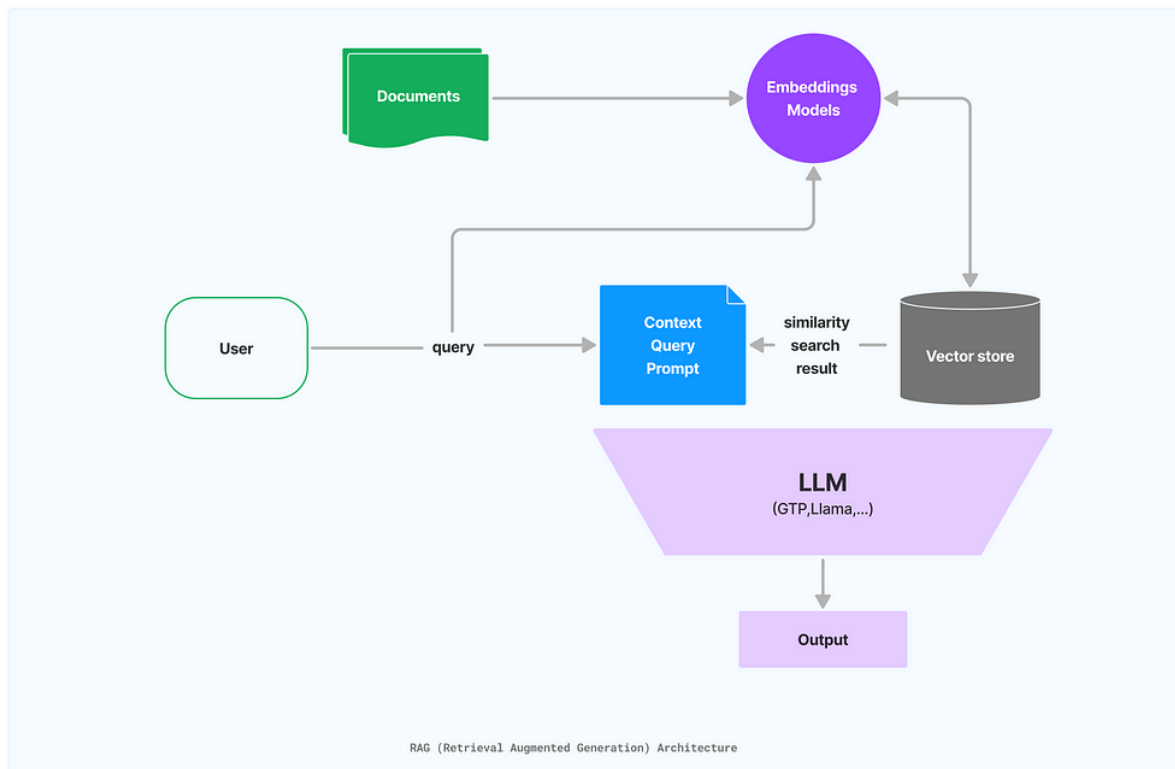
In our application, we will chat with my book “**Designing Scalable Systems**”.

What is a RAG Application?

A **Retrieval-Augmented Generation** (RAG) application combines the power of information retrieval with language generation to enhance the capabilities of conversational AI. In essence, a RAG system first retrieves relevant information from a data source — in our case, a PDF document — and then uses this context to generate accurate and informed responses. This approach enables the AI to provide answers that are not only contextually aware but also deeply rooted in the specific content it has access to.

By leveraging RAG, our application can transform static text from “**Designing Scalable Systems**” into dynamic conversations, making the information more accessible and interactive.

Here is the architecture of a RAG Application



RAG (Retrieval Augmented Generation) Architecture

Understanding Langchain

Langchain is a powerful framework designed to simplify the integration of language models into applications that interact with structured or unstructured data. It's particularly suited for building applications that process, understand, and make decisions based on large volumes of text. Langchain supports a variety of **Natural Language Processing (NLP)** tasks and integrates seamlessly with tools and models from leading AI providers, making it an ideal choice for developers looking to leverage AI in document-based applications.

To explore more about how Langchain can enhance your projects and dive deeper into its capabilities, visit the official Langchain documentation [here](#).

Before we start building a RAG app, let's first learn about embeddings and vector stores.

What are Embeddings and Vector Stores?

Embeddings transform text into numerical vectors that capture deep meanings, enabling computers to process language like humans. They are crucial in machine learning for identifying text relationships and supporting tasks like searching, classifying, and grouping text by similarity.

Vector stores are databases optimized to manage and quickly access these numerical vectors. In applications like Retrieval-Augmented Generation (RAG), vector stores efficiently find text most related to user queries, enabling AI models to respond accurately and swiftly.

In this article, we're using **Langchain's in-memory vector store** and **OpenAI's embeddings** to quickly handle and search through text from a PDF. This setup helps our app find the right information fast and accurately.

Let's start building a RAG Application

In this application, we will chat with my book “**Designing Scalable Systems**”.

1. Setting Up Your Environment

Before diving into the code, ensure your development environment is set up with the necessary tools:

- **Node.js:** Ensure Node.js is installed on your system.
 - **Package Installation:** Install the required Node.js packages.
Navigate to your project directory in the terminal and run:

```
npm install dotenv openai @langchain/openai @langchain/community  
langchain pdf-parse dotenv
```

This will install all the necessary packages including the Langchain libraries and OpenAI SDK.

2. Configuring the Application

Create a .env file in your project directory to store sensitive configuration settings such as your OpenAI API key:

```
OPENAI_API_KEY=your-api-key-here
```

3. Writing the Application Code

Let's analyze the code you provided and set it up for execution:

Import Dependencies

Begin by importing the necessary modules in your main application file:

```
import { openai } from './openai.js';
import { MemoryVectorStore } from 'langchain/vectorstores/memory';
import { OpenAIEmbeddings } from '@langchain/openai';
import { CharacterTextSplitter } from 'langchain/text_splitter';
import { PDFLoader } from
'@langchain/community/document_loaders/fs/pdf';
```

where **open.js** will be:

```
import { configDotenv } from 'dotenv'
configDotenv()
```

```
import OpenAI from 'openai'
export const openai = new OpenAI(process.env.OPENAI_API_KEY)
```

Load and process the PDF:

Load the PDF file you want to interact with:

```
const pdf = 'Designing Scalable Systems.pdf';
const loader = new PDFLoader(pdf);
const loadedDoc = await loader.load();
```

- Split the loaded document into manageable chunks using a character text splitter:

```
const splitter = new CharacterTextSplitter({
  separator: ' ',
  chunkSize: 2500,
  chunkOverlap: 200
});
const pdfDocs = await splitter.splitDocuments(loadedDoc);
```

Create the vector store:

Convert the document chunks into vectors and store them

```
const createStore = (docs) =>
  MemoryVectorStore.fromDocuments(docs, new
  OpenAIEmbeddings());
const store = await createStore(pdfDocs);
```

Set Up the Query Function:

Define a function to handle queries and fetch responses based on the document content:

```
const query = async () => {
  const question = process.argv[2] || 'How can i design a scalable real
  time app?';
  const results = await store.similaritySearch(question, 2);

  const response = await openai.chat.completions.create({
    model: 'gpt-4o',
    temperature: 0,
    messages: [
      {
        role: 'system',
        content: 'You are an AI assistant. Answer questions or else!',
      },
      {
        role: 'user',
```

content: ` Answer the following question using the provided context. If you cannot answer the question with the context, don't lie and make up stuff. Just say you need more context.

Question: \${question}

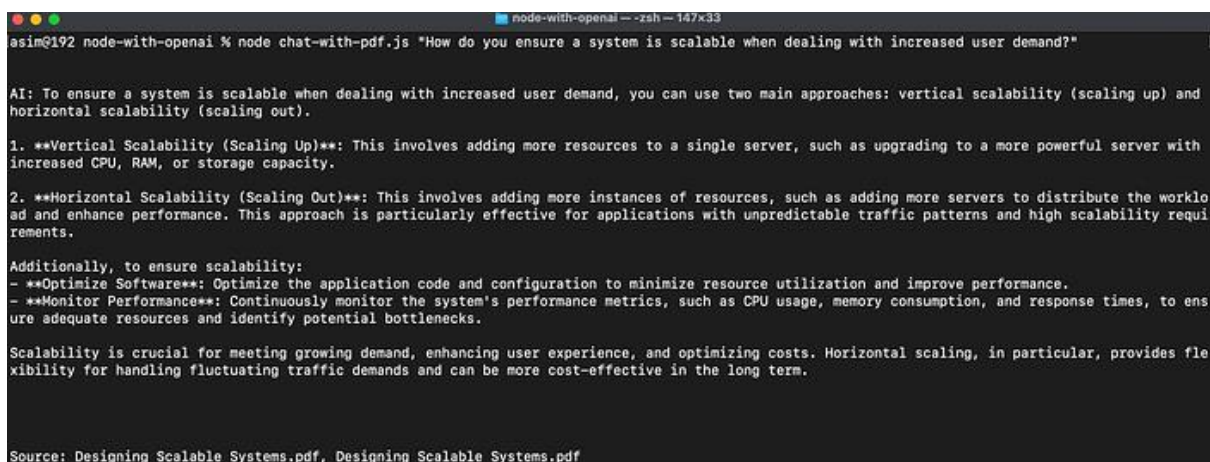
Context: \${results.map((r) => r.pageContent).join('\n')}

```
` ,  
},  
],  
});
```

```
console.log(`\n\nAI: ${response.choices[0].message.content}\n\n` );  
console.log(`\n\nSource: ${results.map((r) =>  
r.metadata.source).join(', ')}\n\n` );  
}  
query();
```

The main **query** function in our RAG app processes user questions by searching for relevant content within a PDF using a **vector store**. It then uses **OpenAI's GPT-4** to generate contextually aware responses based on the information retrieved, providing precise answers that are informed directly by the document's content.

Demo: 1



```
node-with-openai -- zsh -- 147x33  
asim@192 node-with-openai % node chat-with-pdf.js "How do you ensure a system is scalable when dealing with increased user demand?"  
  
AI: To ensure a system is scalable when dealing with increased user demand, you can use two main approaches: vertical scalability (scaling up) and horizontal scalability (scaling out).  
  
1. Vertical Scalability (Scaling Up): This involves adding more resources to a single server, such as upgrading to a more powerful server with increased CPU, RAM, or storage capacity.  
  
2. Horizontal Scalability (Scaling Out): This involves adding more instances of resources, such as adding more servers to distribute the workload and enhance performance. This approach is particularly effective for applications with unpredictable traffic patterns and high scalability requirements.  
  
Additionally, to ensure scalability:  
- Optimize Software: Optimize the application code and configuration to minimize resource utilization and improve performance.  
- Monitor Performance: Continuously monitor the system's performance metrics, such as CPU usage, memory consumption, and response times, to ensure adequate resources and identify potential bottlenecks.  
  
Scalability is crucial for meeting growing demand, enhancing user experience, and optimizing costs. Horizontal scaling, in particular, provides flexibility for handling fluctuating traffic demands and can be more cost-effective in the long term.  
  
Source: Designing Scalable Systems.pdf, Designing Scalable Systems.pdf
```

Demo1

Demo: 2

```
node-with-openai — zsh — 126x24
asim@192 node-with-openai % node chat-with-pdf.js "who are the authors of this book?"

AI: The authors of the book "Designing Scalable Systems" are Huzaifa Asif and Asim Hafeez.

Source: Designing Scalable Systems.pdf, Designing Scalable Systems.pdf
```

Demo: 2

Conclusion

So far, we have built an app that lets you chat with a PDF, almost like talking to a friend. We used Langchain and Node.js to make it possible to ask questions and get answers straight from the text of “**Designing Scalable Systems.**” It’s like turning your book into a smart assistant that can help you find information quickly.

This could be super useful for students who need quick answers while studying or for professionals who need facts from manuals quickly. The possibilities are endless. You could adapt this app to work with other documents or add even smarter features to make the interactions more helpful.

If you found the article helpful, don’t forget to share the knowledge with more people! 🙌