

This document provides a complete beginner-friendly, end-to-end RAG (Retrieval Augmented Generation) project using **Node.js + Express (backend)** and **React (frontend)**.

1 What We Are Building

Project: Ask-My-PDF (RAG App)

What it does: - Upload a PDF - Ask questions related to that PDF - AI answers only from the uploaded content

Core Concepts Used: - RAG pipeline (Ingestion + Query) - Embeddings - Vector Database (Chroma – local) - OpenAI LLM

2 Prerequisites

Make sure you have: - Node.js v18+ - npm - OpenAI API Key

3 Project Folder Structure

```
rag-app/
|
└── backend/
    ├── src/
    │   ├── app.js
    │   └── server.js
    |
    ├── routes/
    │   ├── upload.routes.js
    │   └── chat.routes.js
    |
    ├── controllers/
    │   ├── upload.controller.js
    │   └── chat.controller.js
    |
    ├── services/
    │   ├── pdfLoader.js
    │   ├── textSplitter.js
    │   ├── embedding.service.js
    │   └── vectorDb.service.js
```

```
|- llm.service.js
|   |
|   |- rag/
|   |   |- ingest.js
|   |   |- query.js
|   |
|   |- config/
|   |   |- openai.js
|   |
|   |- uploads/
|   |- .env
|   |- package.json
|
|- frontend/
|   |- src/
|   |   |- components/
|   |   |   |- UploadPdf.jsx
|   |   |   |- ChatBox.jsx
|   |
|   |- services/api.js
|   |- App.jsx
|   |- main.jsx
|   |- package.json
```

4 Backend Setup

Step 1: Initialize Backend

```
mkdir backend && cd backend
npm init -y
npm install express cors multer pdf-parse dotenv openai chromadb
```

Step 2: .env

```
OPENAI_API_KEY=your_openai_key_here
```

Step 3: `src/app.js`

```
const express = require("express");
const cors = require("cors");

const uploadRoutes = require("./routes/upload.routes");
const chatRoutes = require("./routes/chat.routes");

const app = express();
app.use(cors());
app.use(express.json());

app.use("/upload", uploadRoutes);
app.use("/chat", chatRoutes);

module.exports = app;
```

Step 4: `src/server.js`

```
require("dotenv").config();
const app = require("./app");

app.listen(3000, () => {
  console.log("Backend running on port 3000");
});
```

Step 5: Upload Route

`routes/upload.routes.js`

```
const express = require("express");
const multer = require("multer");
const uploadController = require("../controllers/upload.controller");

const router = express.Router();
const upload = multer({ dest: "uploads/" });

router.post("/", upload.single("file"), uploadController.uploadPdf);

module.exports = router;
```

```
controllers/upload.controller.js
```

```
const ingest = require("../rag/ingest");

exports.uploadPdf = async (req, res) => {
  await ingest(req.file.path);
  res.json({ message: "PDF ingested successfully" });
};
```

Step 6: RAG Ingestion Pipeline

```
rag/ingest.js
```

```
const loadPdf = require("../services/pdfLoader");
const splitText = require("../services/textSplitter");
const { embedAndStore } = require("../services/vectorDb.service");

module.exports = async function ingest(filePath) {
  const text = await loadPdf(filePath);
  const chunks = splitText(text);
  await embedAndStore(chunks);
};
```

```
services/pdfLoader.js
```

```
const fs = require("fs");
const pdfParse = require("pdf-parse");

module.exports = async (path) => {
  const data = await pdfParse(fs.readFileSync(path));
  return data.text;
};
```

```
services/textSplitter.js
```

```
module.exports = (text) => {
  const size = 500;
```

```
const chunks = [];

for (let i = 0; i < text.length; i += size) {
  chunks.push(text.slice(i, i + size));
}

return chunks;
};
```

services/embedding.service.js

```
const OpenAI = require("openai");
const client = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });

exports.getEmbedding = async (text) => {
  const response = await client.embeddings.create({
    model: "text-embedding-3-small",
    input: text,
  });
  return response.data[0].embedding;
};
```

services/vectorDb.service.js

```
const { ChromaClient } = require("chromadb");
const { getEmbedding } = require("./embedding.service");

const client = new ChromaClient();
const collectionName = "pdf-docs";

exports.embedAndStore = async (chunks) => {
  const collection = await client.getOrCreateCollection({ name:
collectionName });

  for (let i = 0; i < chunks.length; i++) {
    const embedding = await getEmbedding(chunks[i]);

    await collection.add({
      ids: [`chunk-${Date.now()}-${i}`],
      embeddings: [embedding],
      documents: [chunks[i]],
    });
  }
};
```

```

    }
};

exports.search = async (query) => {
  const collection = await client.getCollection({ name: collectionName });
  const embedding = await getEmbedding(query);

  const result = await collection.query({
    queryEmbeddings: [embedding],
    nResults: 3,
  });

  return result.documents.flat();
};

```

Step 7: Query Pipeline

rag/query.js

```

const { search } = require("../services/vectorDb.service");
const askLLM = require("../services/llm.service");

module.exports = async function queryRag(question) {
  const docs = await search(question);

  const prompt = `Use the context below to answer the question.\n\nContext:\n${docs.join("\n")}\n\nQuestion:\n${question}`;

  return askLLM(prompt);
};

```

services/llm.service.js

```

const OpenAI = require("openai");
const client = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });

module.exports = async (prompt) => {
  const res = await client.chat.completions.create({
    model: "gpt-4o-mini",
    messages: [{ role: "user", content: prompt }],
  });

```

```
    return res.choices[0].message.content;
};
```

routes/chat.routes.js

```
const express = require("express");
const queryRag = require("../rag/query");

const router = express.Router();

router.post("/", async (req, res) => {
  const answer = await queryRag(req.body.question);
  res.json({ answer });
});

module.exports = router;
```

5 Frontend Setup (React)

```
cd ../
npx create-react-app frontend
cd frontend
npm start
```

src/services/api.js

```
import axios from "axios";

export const uploadPdf = (file) => {
  const form = new FormData();
  form.append("file", file);
  return axios.post("http://localhost:3000/upload", form);
};

export const askQuestion = (question) => {
  return axios.post("http://localhost:3000/chat", { question });
};
```

src/components/UploadPdf.jsx

```
import { uploadPdf } from "../services/api";

export default function UploadPdf() {
  return (
    <input type="file" onChange={(e) => uploadPdf(e.target.files[0])} />
  );
}
```

src/components/ChatBox.jsx

```
import { useState } from "react";
import { askQuestion } from "../services/api";

export default function ChatBox() {
  const [q, setQ] = useState("");
  const [a, setA] = useState("");

  return (
    <div>
      <input value={q} onChange={(e) => setQ(e.target.value)} />
      <button onClick={async () => setA(await askQuestion(q).data.answer)}>Ask</button>
      <p>{a}</p>
    </div>
  );
}
```

src/App.jsx

```
import UploadPdf from "./components/UploadPdf";
import ChatBox from "./components/ChatBox";

export default function App() {
  return (
    <>
      <UploadPdf />
      <ChatBox />
    </>
  );
}
```

```
    );  
}
```

6 How to Run the App

Backend

```
cd backend  
node src/server.js
```

Frontend

```
cd frontend  
npm start
```

7 How to Test

1. Upload PDF
 2. Ask question from PDF
 3. Verify AI answers are context-based
-

8 What You Learned

- Complete RAG pipeline
 - Vector DB usage
 - Prompt augmentation
 - Real AI system design
-

 Congratulations! You built a full RAG application.