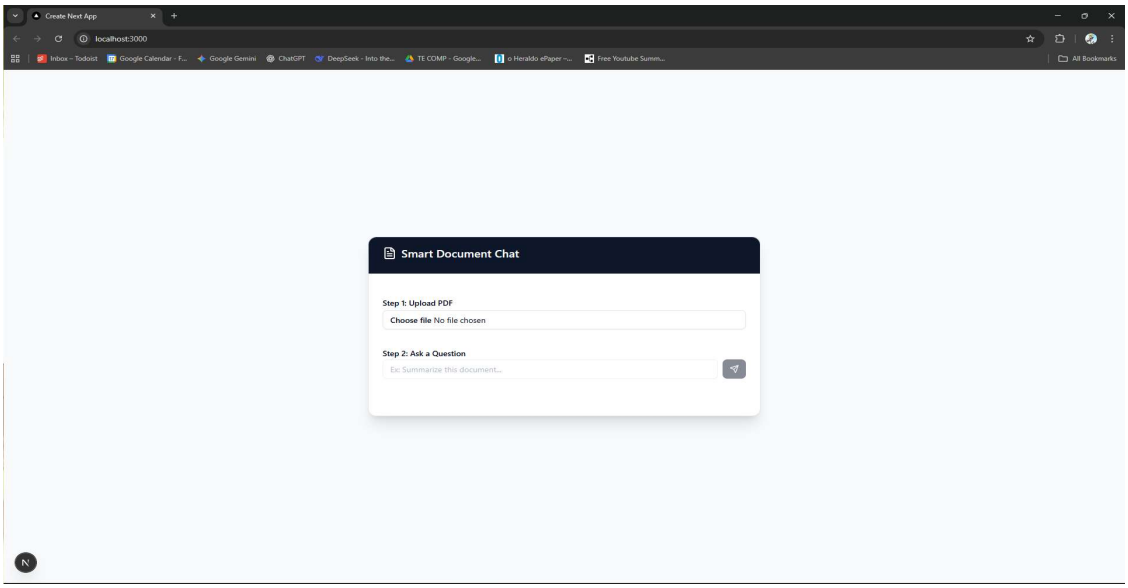
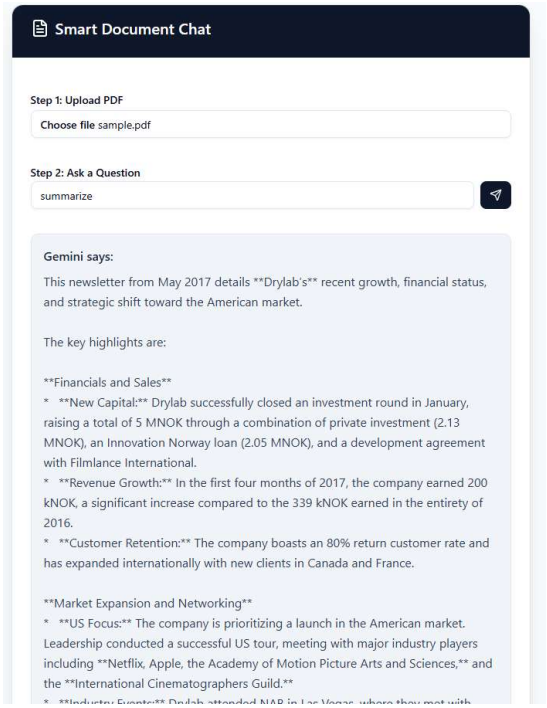
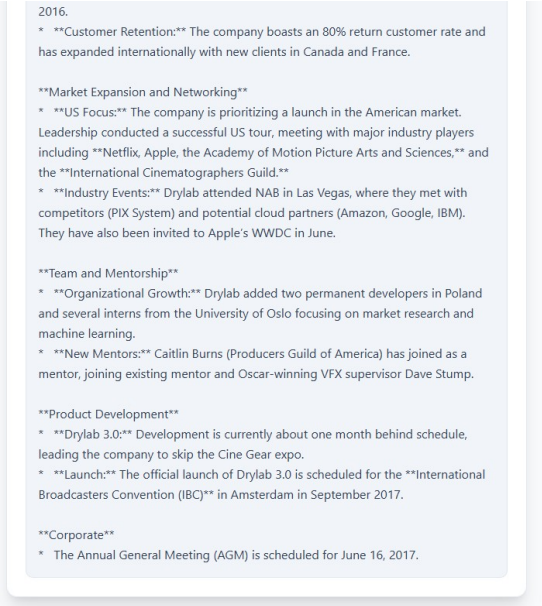
evolution

19 January 2026 11:23 PM

19-01-26





- 

- This started as a very basic project
- User enters pdf and question as input
- Text from pdf is extracted and sent to gemini.
- Response from gemini is presented to user .

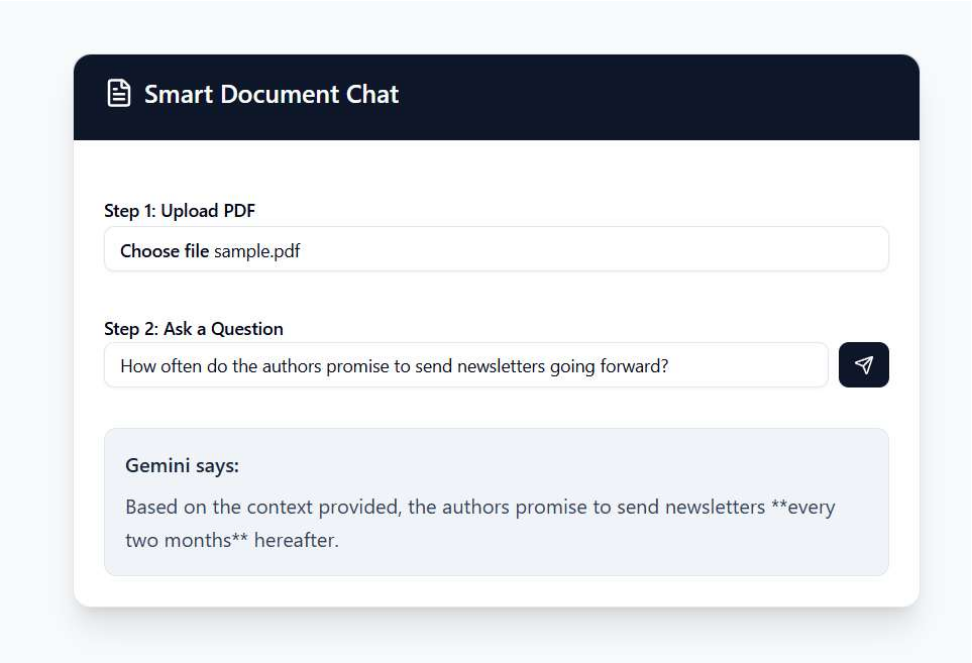
- Very inefficient , can also be done by gemini , no need for separate website.

Only main.py is used

20-01-26
MAKING A RAG

RAG (Retrieval-Augmented Generation)

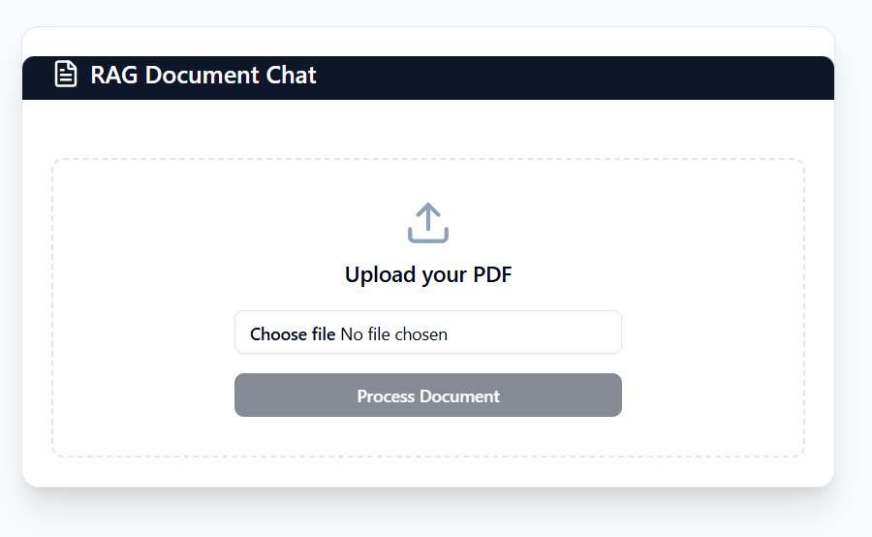
Retrieval: The system searches a database for the most relevant sections of your documents based on your query. Augmentation: It adds this retrieved information to your prompt (e.g., "Using this document, answer..."). Generation: The LLM uses that targeted information to create a precise, informed answer.

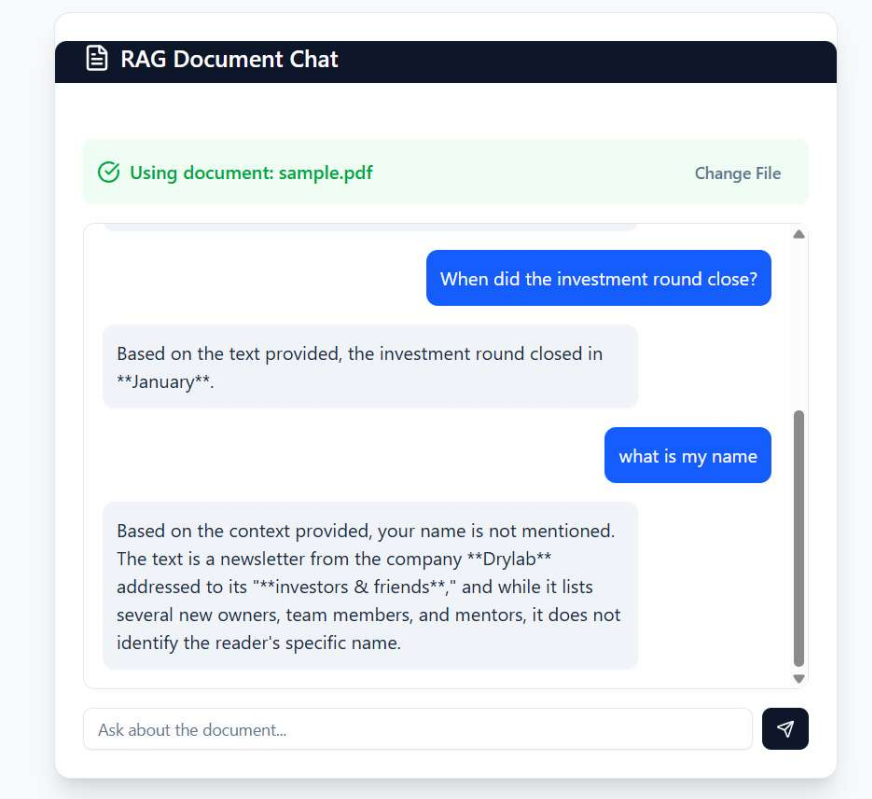


- In this version , I used vector embeddings for efficient data handling.

- Workflow :
- Pdf is hardcoded , and user simply asks questions.
- Before question is processed , pdf is divided into chunks , and stored in data base along with its vector embedding, created by gemini
- Question is then processed (converted to embedding, we set threshold for match , and max number of chunks accepted)
- This then sent to supabase , it compares it with every chunk in database and gives a similarity score. And only send top [max number of chunks accepted] answer
- This answer is then added to user question and sent to gemini
- And response is pasted

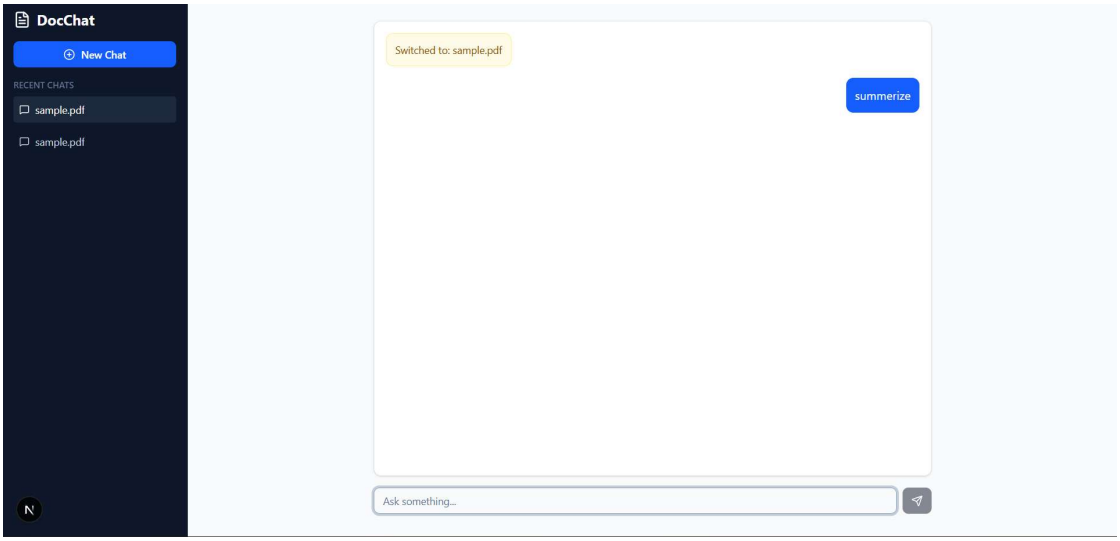
- **Ingestion (Done once):** PDF -> Chunks -> Vectors -> Database. (ingest.py)
- **Retrieval (Done every question):** Question -> Vector -> **SQL Comparison** -> Top Matches -> LLM Answer. (main.py)

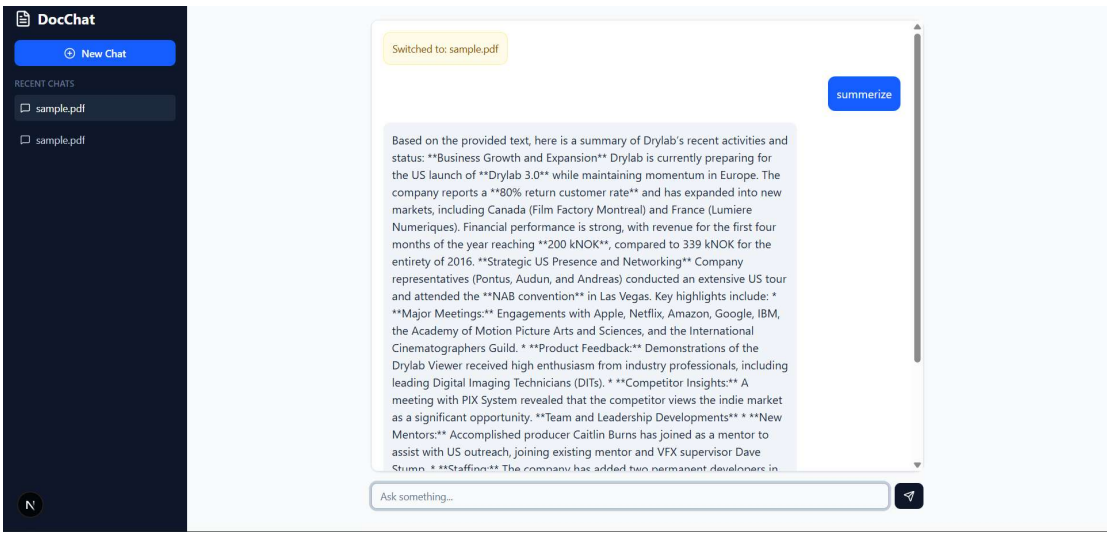




- Ingest.py is fused with main.py
- I have 2 routes not /upload and /chat
- /upload : take file as input, divide it into chunks , create vectors and store it in database.
- All the messages showed are stored in an array on client side.

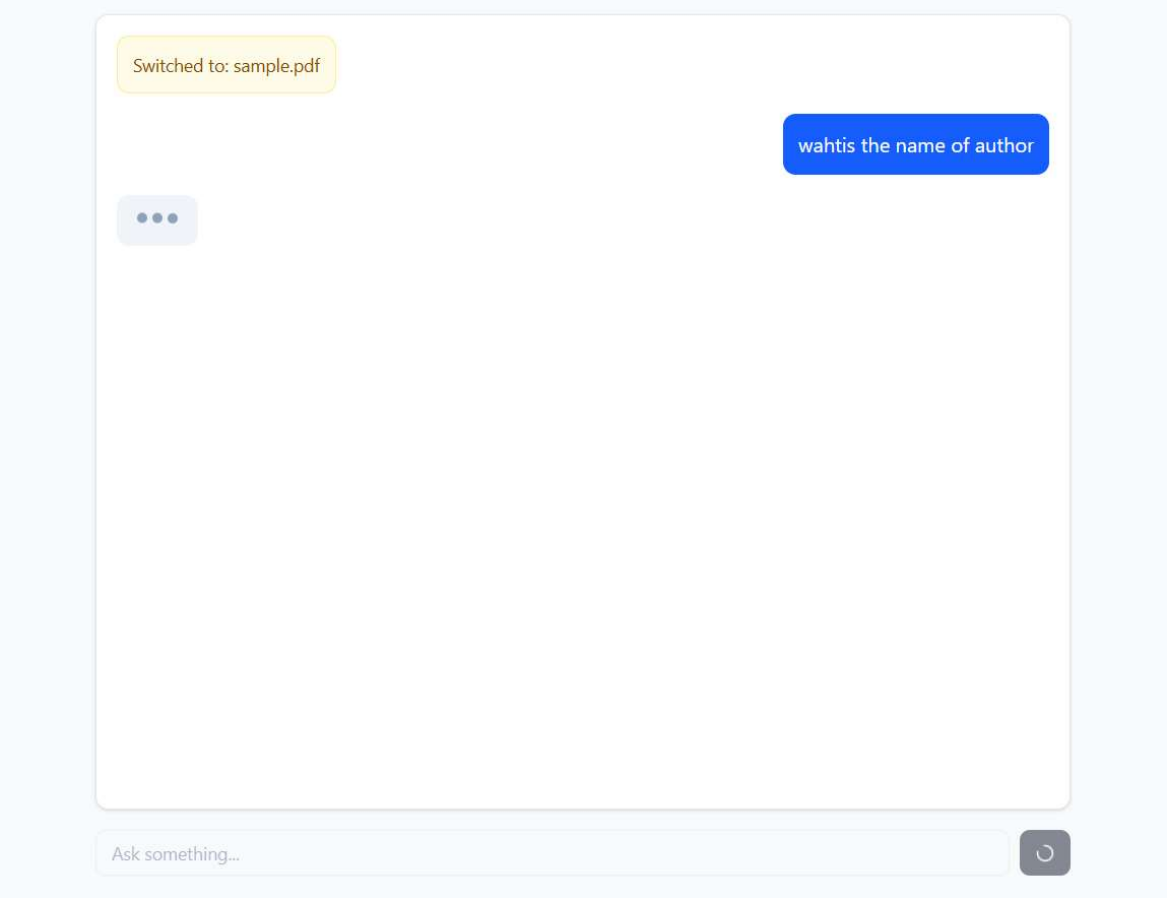
- Cons :
  - Only one fiel can be uploaded
  - Sessions are not stored.

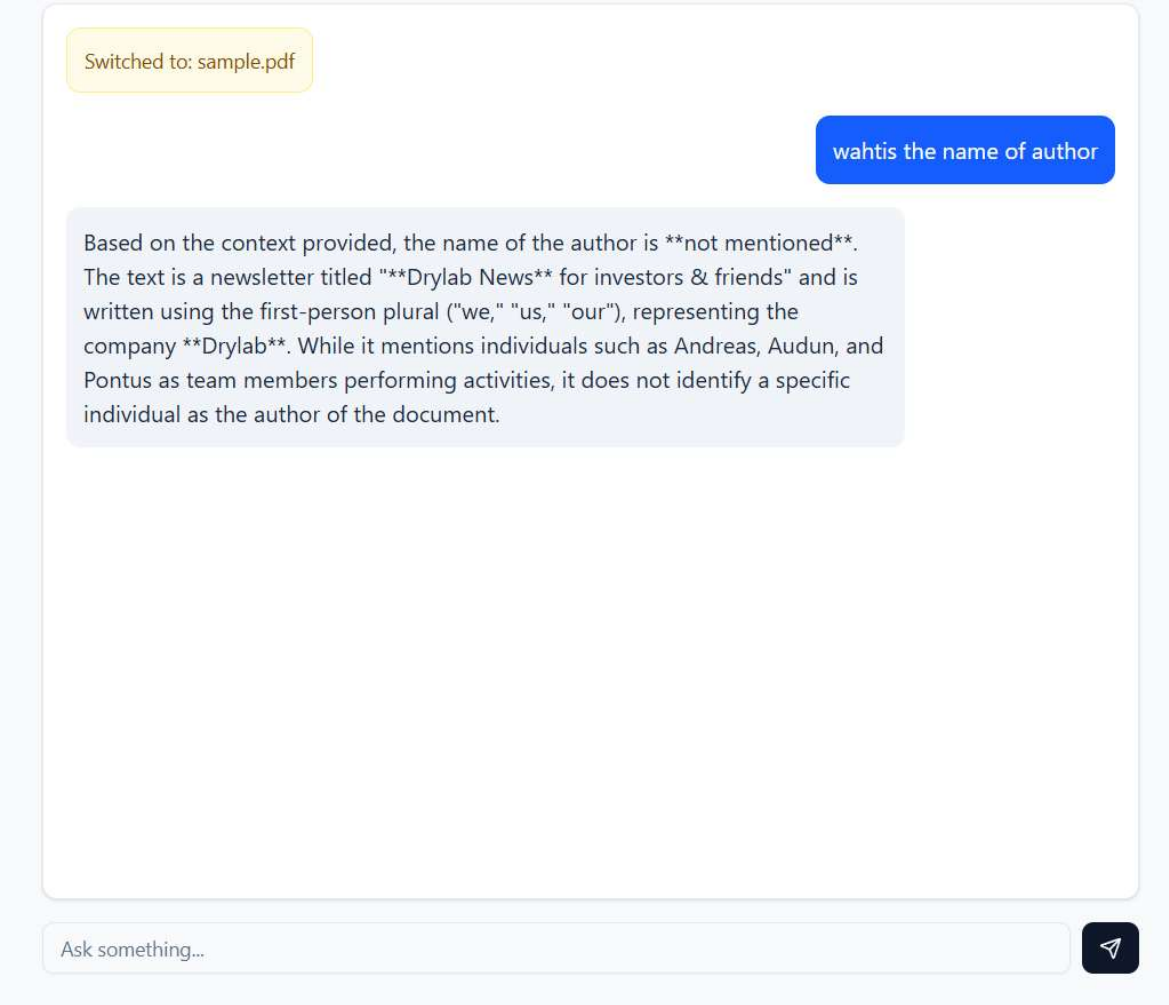


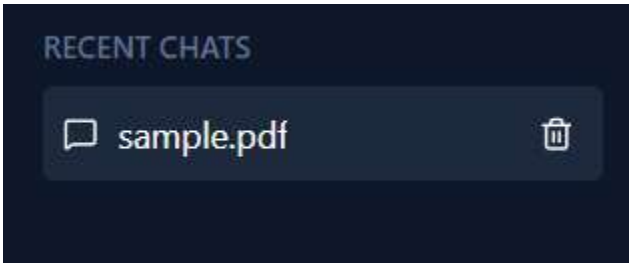
- 

- Sessions created

- No loading options from system side
- No delete session option
- Chats are not saved , wh8ile switching it vanishes

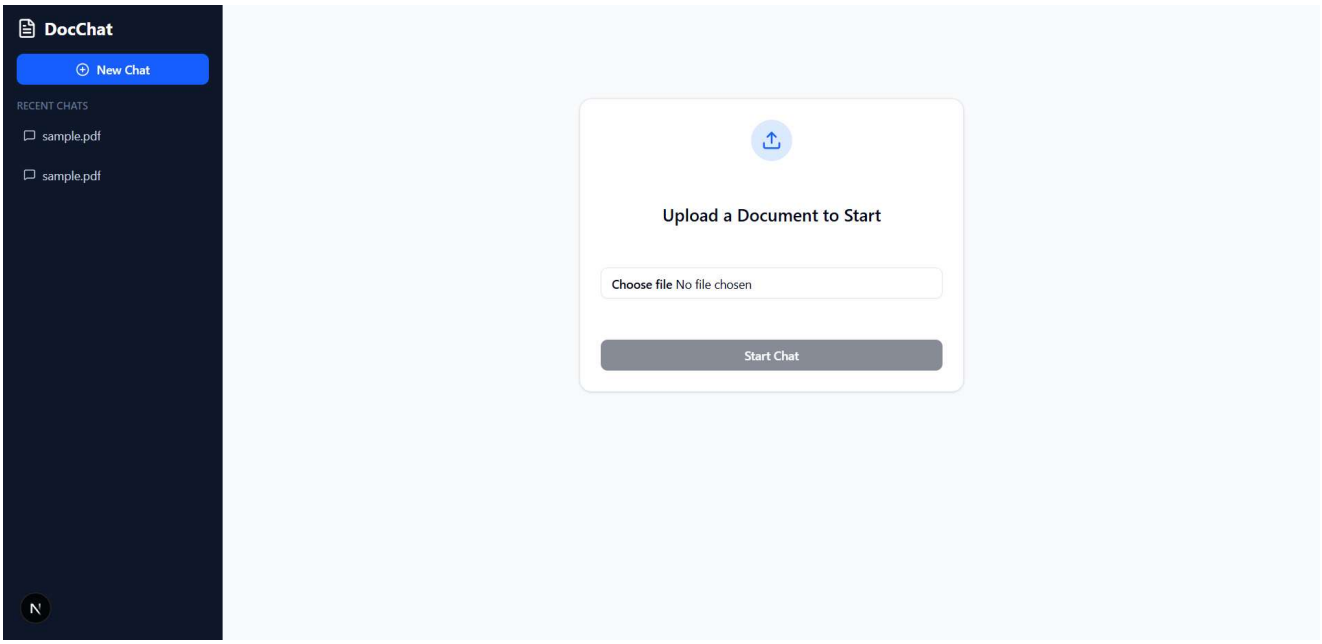

Switched to: sample.pdf
wahtis the name of author
Ask something...





- Loading added
- Delete session added

- Session id is created when I uload a document and all session ids are stored in array.



SUMMERY:

In backend I have

```
@app.delete("/sessions/{session_id}")

@app.get("/sessions")


@app.get("/sessions/{session_id}/messages")


@app.post("/chat")



@app.post("/upload")
```

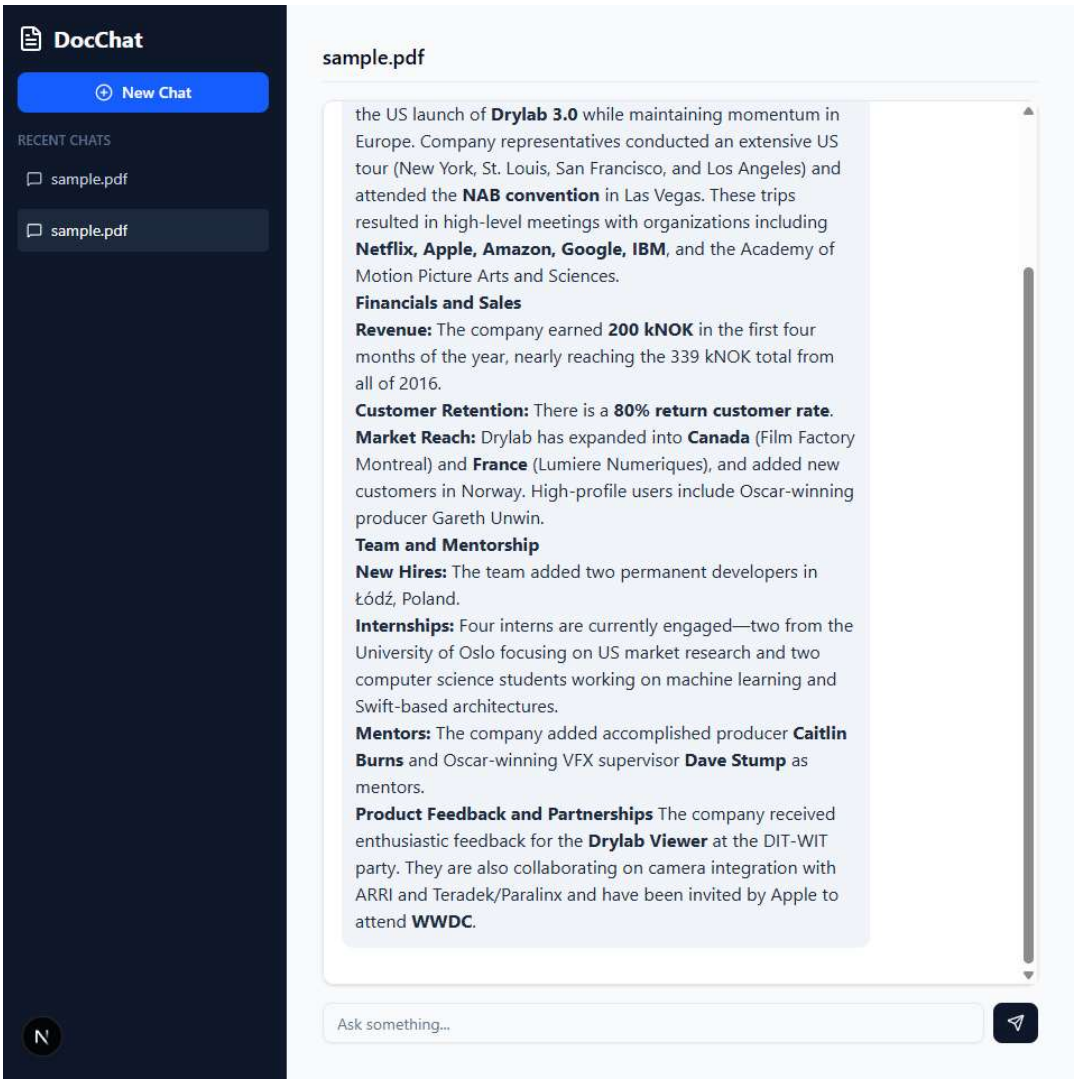
And functions like :

```
def get_text_from_pdf(pdf_bytes): extract text


def split_text(text, chunk_size=1000, overlap=100): split into
chunks and returns it


async def ingest_document(file: UploadFile, session_id: str):calls
both above function , and load it in database

get_relevant_context(user_question: str, session_id: str):
```

Convert qeustion to vector , Calls SQL , get the context

FLOW:

We use `@app.get("/sessions")` and store it in array sessions in front end.
Initially you have no sessions , when you upload a file
`@app.post("/upload")`, session is created. This also make vectir embedding and chunks and store it in database.
Now a session is get and we call `@app.get("/sessions/{session_id}/messages")` store the messages in array and dsplay it.
`@app.post("/chat")` to a qeustion.
This functions sends vector embedding of this question to SQL in supabase , which further compares and returns relevant context.



React markdown added

Here there is a problem I faced , a flaw

Flaw : The "Sentence Chopping" Problem

Your current chunking logic (text[start:end]) is "naive." It blindly cuts text at character 1,000, regardless of what is happening.

The Risk: Imagine a sentence in your PDF:

*"The secret code to launch the rocket is **[CUT HERE]** 12345."*

- **Chunk A** has: "The secret code to launch the rocket is"
- **Chunk B** has: "12345."

If you ask *"What is the code?"*, the AI might find Chunk A (which has no answer) or Chunk B (which has no context), and fail to answer.

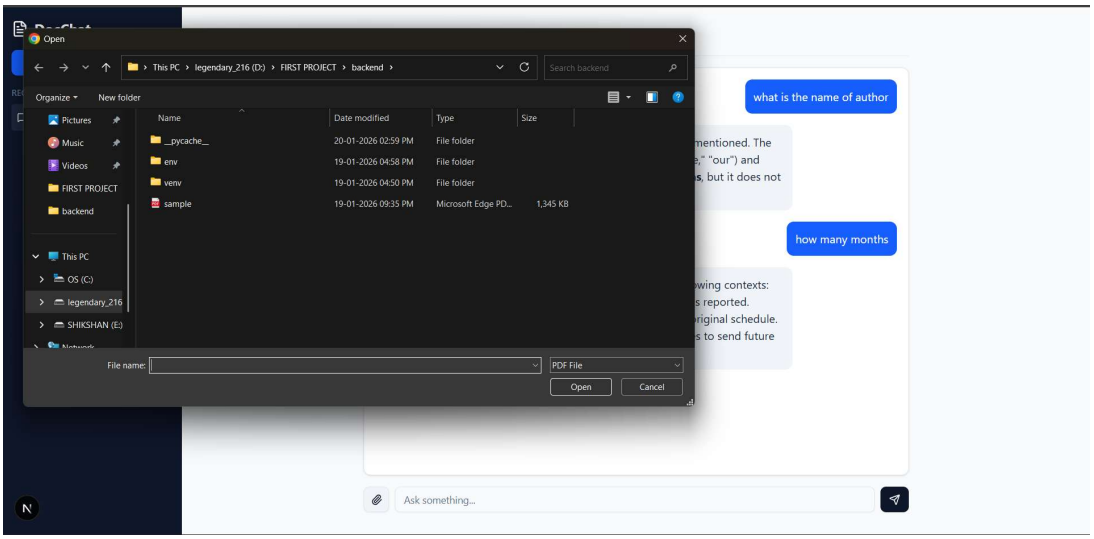
The Fix: Recursive Character Splitter Use a smarter splitter (like LangChain's logic) that tries to split by **paragraphs** first. If that's too big, it splits by **sentences**. If that's too big, *then* it splits by words. This keeps ideas together.

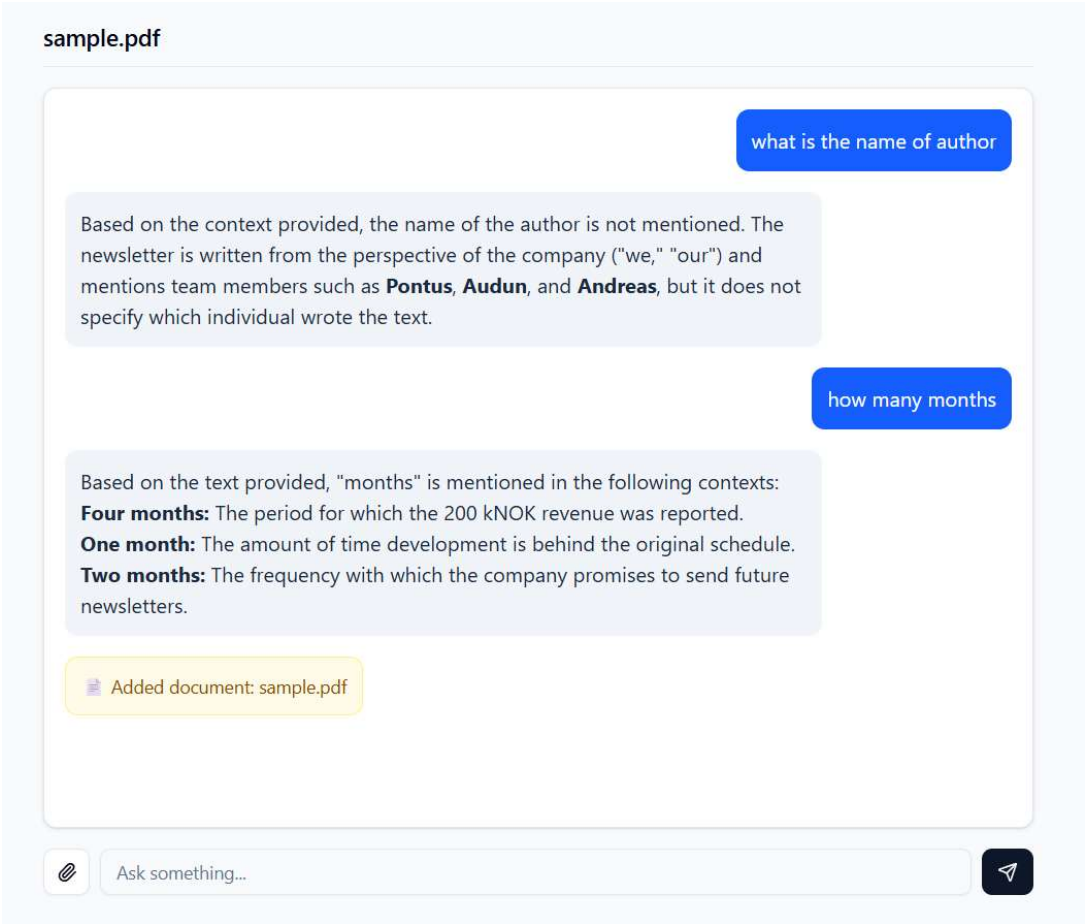
Instead of blindly cutting at character 1,000, the new splitter will try to split in this order of priority:

1. **Double Newlines (\n\n)**: Keep paragraphs together.
2. **Single Newlines (\n)**: Keep lines together.
3. **Periods (.)**: Keep sentences together.
4. **Spaces ( )**: Keep words together.
5. **Characters**: Only cut a word in half if absolutely necessary.

```
pip install langchain-text-splitters
```

```
from langchain_text_splitters import RecursiveCharacterTextSplitter

def split_text(text):
    """
    Splits text recursively: checks for paragraphs, then sentences, then words.
    """
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=1000,      # Aim for ~1000 characters per chunk
        chunk_overlap=100,    # Keep 100 characters of history between chunks
        length_function=len,
        is_separator_regex=False,
    )
    chunks = text_splitter.split_text(text)
    return chunks
```





EXTRA FEATURES TO TAND THIS OUT

Feature 2: "Hybrid Search" (The Accuracy Booster)

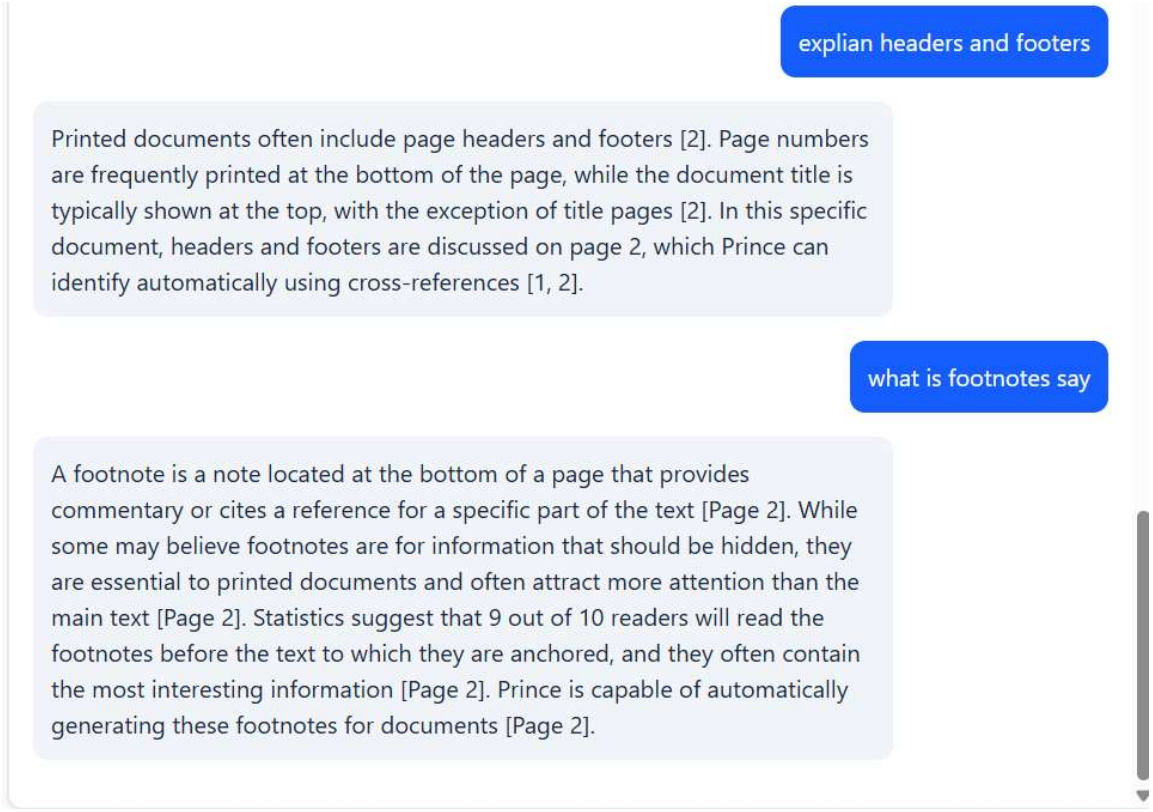
Right now, you use **Vector Search** (searching for meaning).

- *User asks:* "Error 504".
- *Vector Search might find:* "Network timeout issues" (Related meaning).

- *But it might miss:* "Error 504" exactly if the math isn't perfect.

The Upgrade: Combine **Vector Search** + **Keyword Search** (Ctrl+F).

- If the user searches for a specific ID ("Invoice #9928"), Keyword Search finds it instantly.
- If the user searches for a concept ("Why is the bill high?"), Vector Search finds it.
- **Why:** It fixes the "Dumb AI" moments where it misses exact matches.
- **Difficulty:** High (Requires complex SQL in Supabase).



- ## Citations added

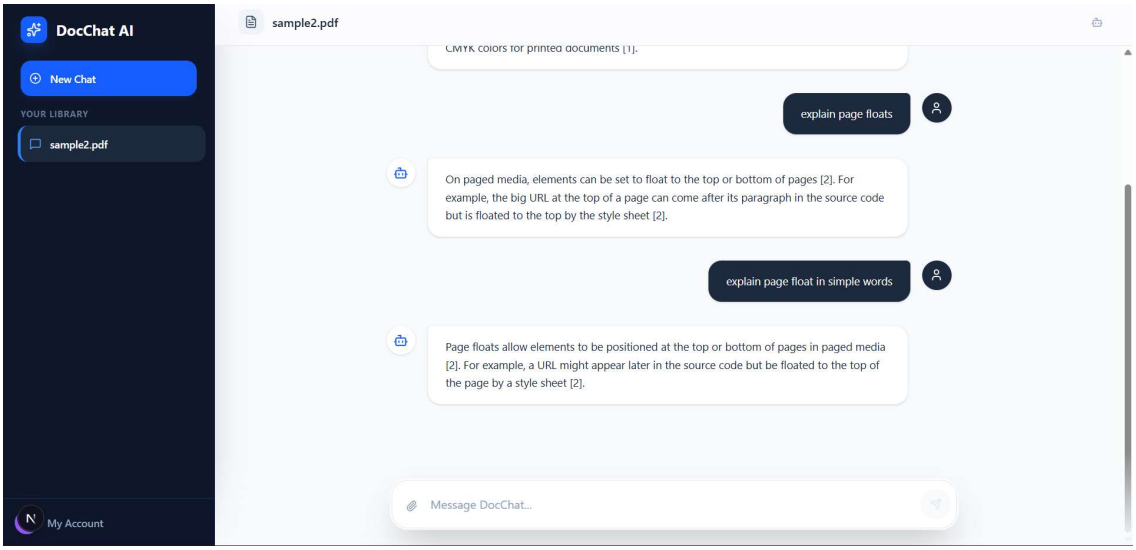
  - **Chunking is done page wise now. We store the page number in additional column we created in database , table document.**

Feature : "Real-Time Typing" (Streaming)

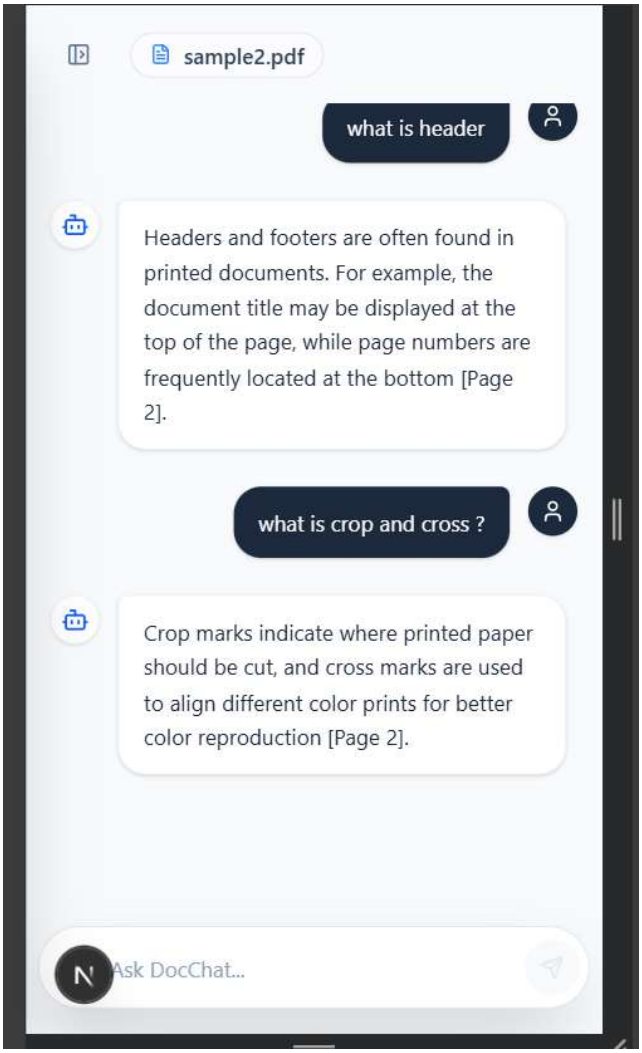
Right now, you click "Send"... wait 3 seconds... and **BAM**, the whole text appears at once. ChatGPT doesn't do that. It "types" the answer word-by-word.

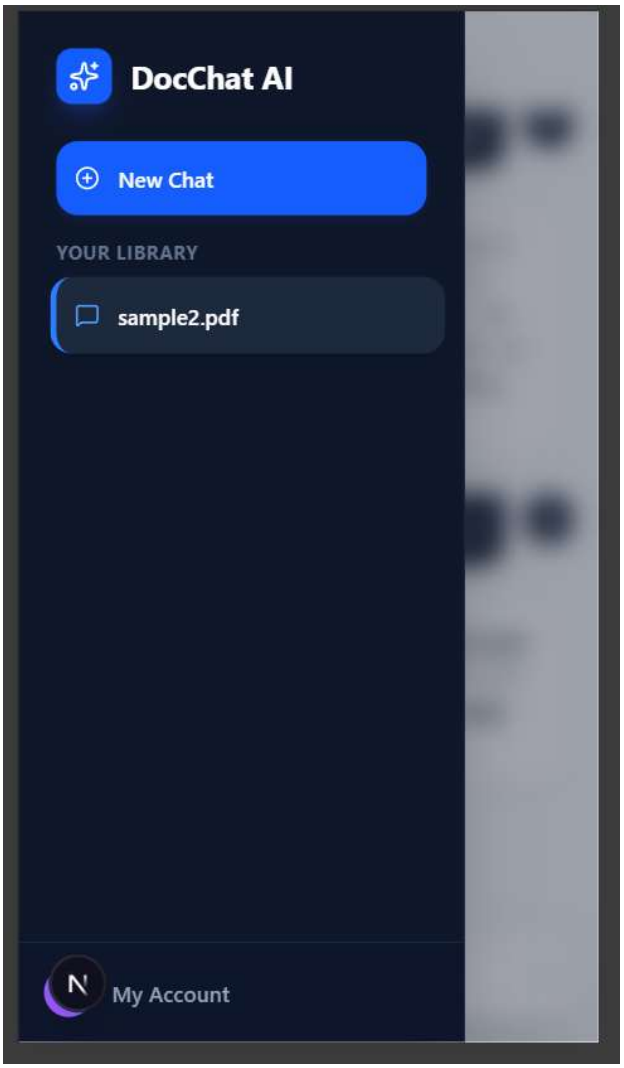
The Upgrade: Stream the response from the backend to the frontend.

- **Why:** It feels **10x faster**. The user starts reading immediately instead of staring at a spinner.
- **Difficulty:** Medium (Requires changing FastAPI to StreamingResponse).

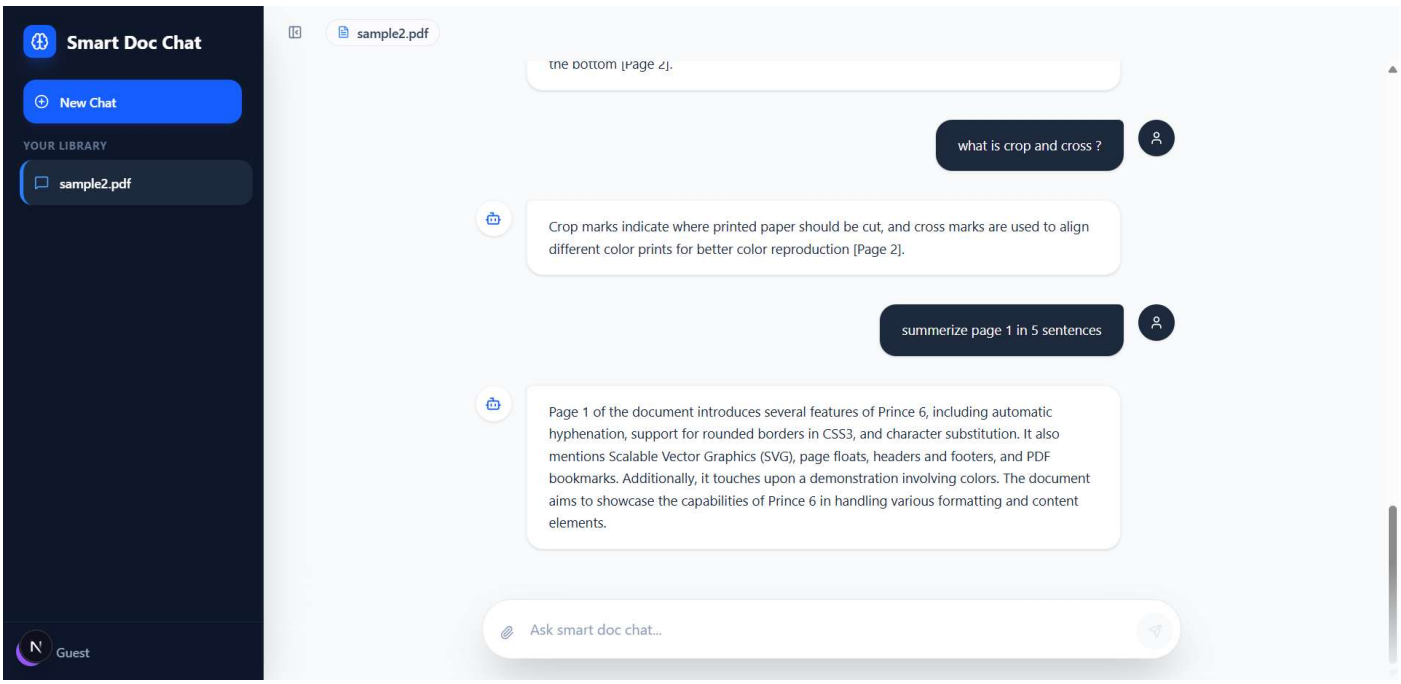


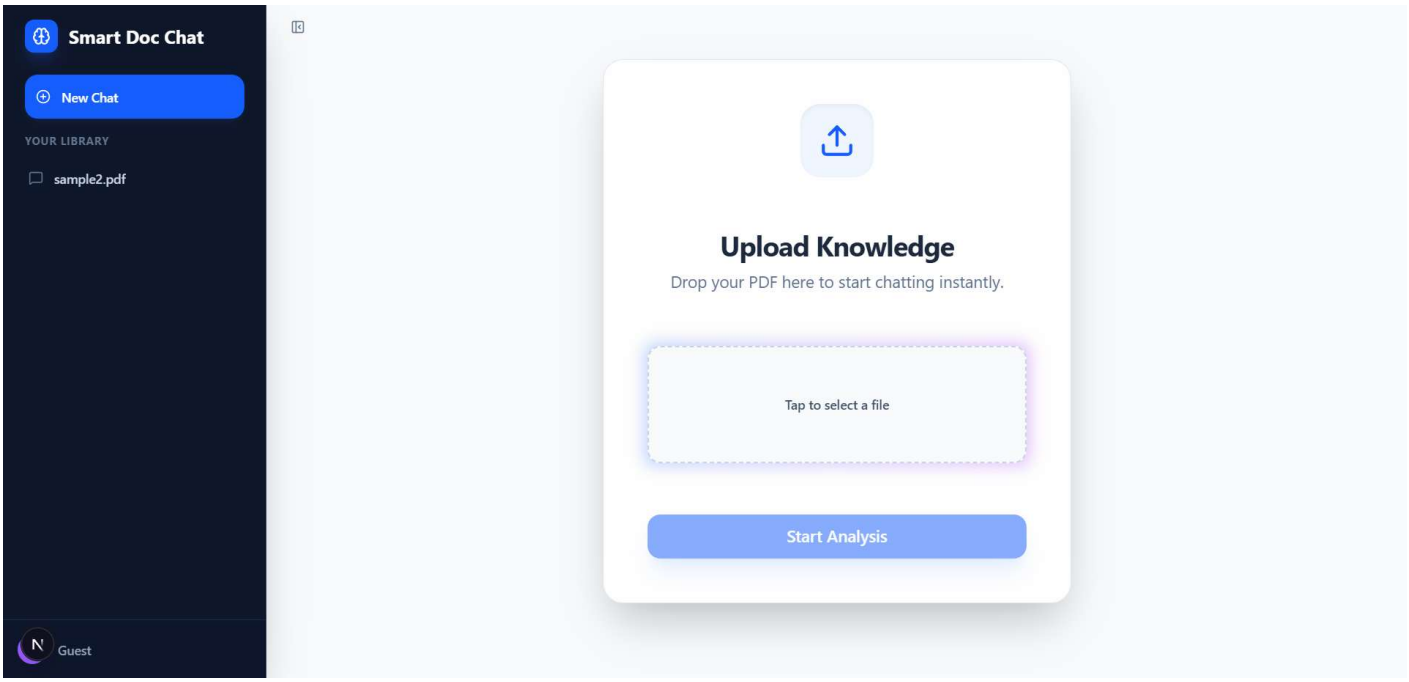
- UI added


sample2.pdf
what is header
Headers and footers are often found in printed documents. For example, the document title may be displayed at the top of the page, while page numbers are frequently located at the bottom [Page 2].
what is crop and cross ?
Crop marks indicate where printed paper should be cut, and cross marks are used to align different color prints for better color reproduction [Page 2].
Ask DocChat...



- Responsiveness added





https://legendary216-smart-doc-chat.vercel.app/