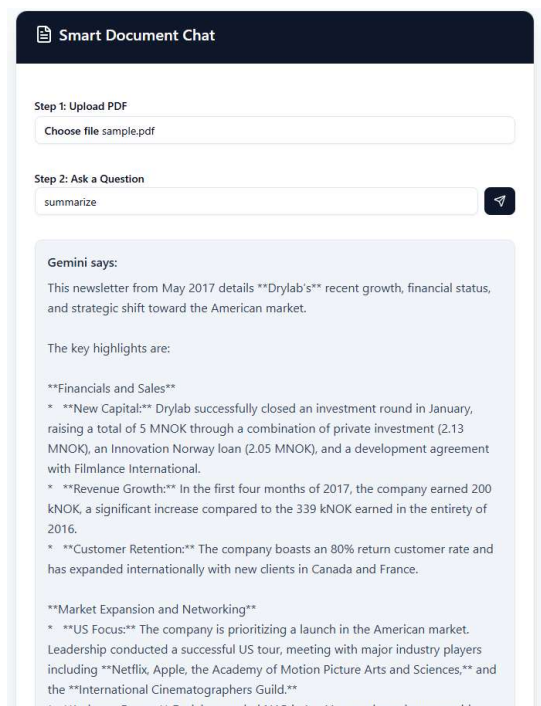
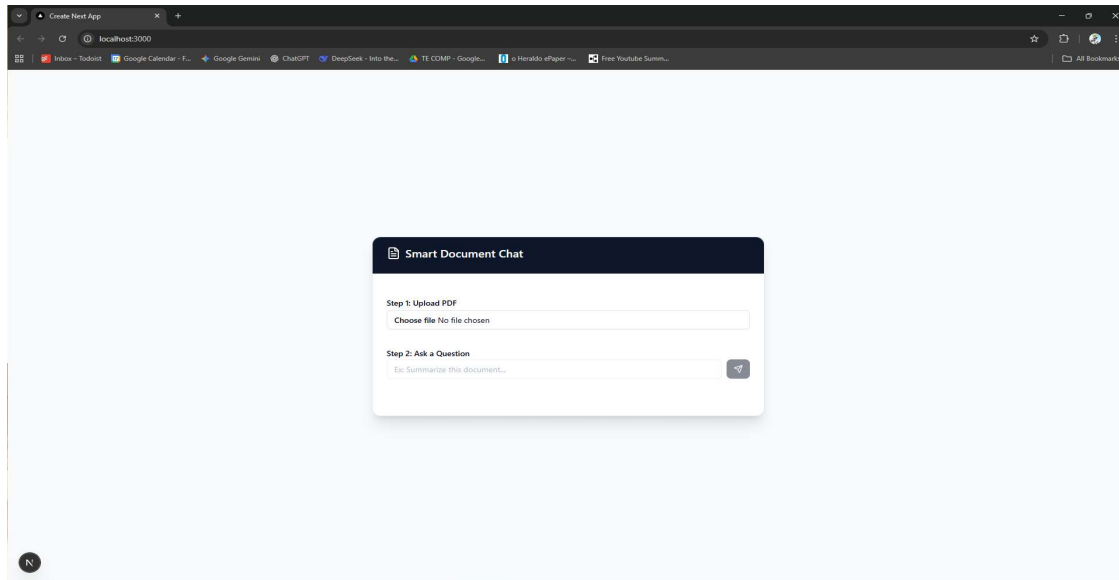


# evolution

19 January 2026 11:23 PM

19-01-26



2016.

- \* **Customer Retention:** The company boasts an 80% return customer rate and has expanded internationally with new clients in Canada and France.

**Market Expansion and Networking**

- \* **US Focus:** The company is prioritizing a launch in the American market. Leadership conducted a successful US tour, meeting with major industry players including Netflix, Apple, the Academy of Motion Picture Arts and Sciences, and the International Cinematographers Guild.
- \* **Industry Events:** Drylab attended NAB in Las Vegas, where they met with competitors (PIX System) and potential cloud partners (Amazon, Google, IBM). They have also been invited to Apple's WWDC in June.

**Team and Mentorship**

- \* **Organizational Growth:** Drylab added two permanent developers in Poland and several interns from the University of Oslo focusing on market research and machine learning.
- \* **New Mentors:** Caitlin Burns (Producers Guild of America) has joined as a mentor, joining existing mentor and Oscar-winning VFX supervisor Dave Stump.

**Product Development**

- \* **Drylab 3.0:** Development is currently about one month behind schedule, leading the company to skip the Cine Gear expo.
- \* **Launch:** The official launch of Drylab 3.0 is scheduled for the International Broadcasters Convention (IBC) in Amsterdam in September 2017.

**Corporate**

- \* The Annual General Meeting (AGM) is scheduled for June 16, 2017.

- 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.

**Smart Document Chat**

Step 1: Upload PDF

Choose file sample.pdf

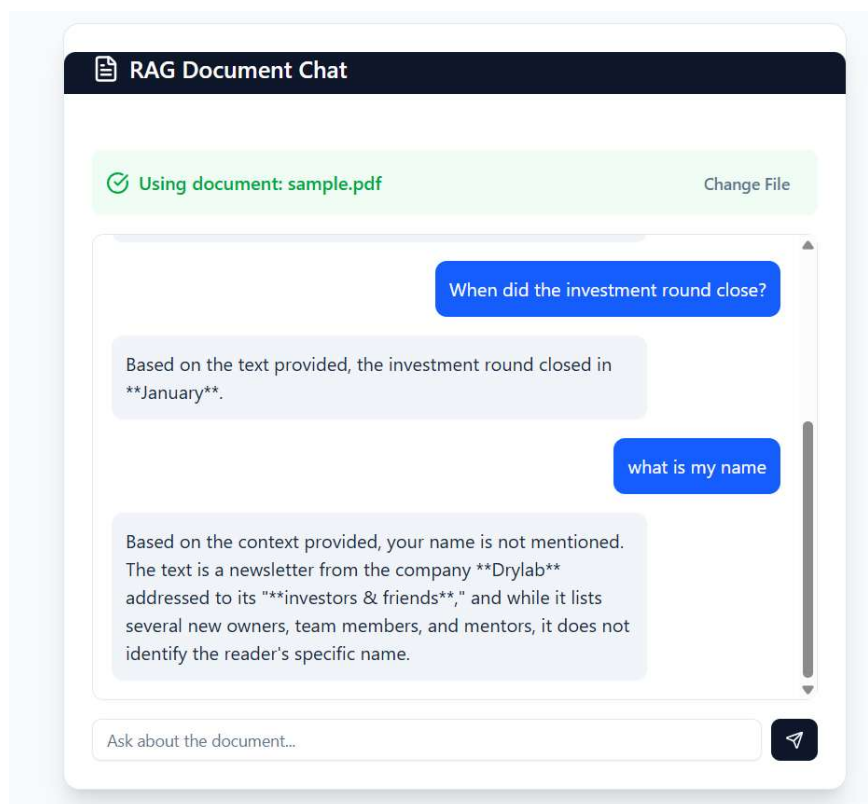
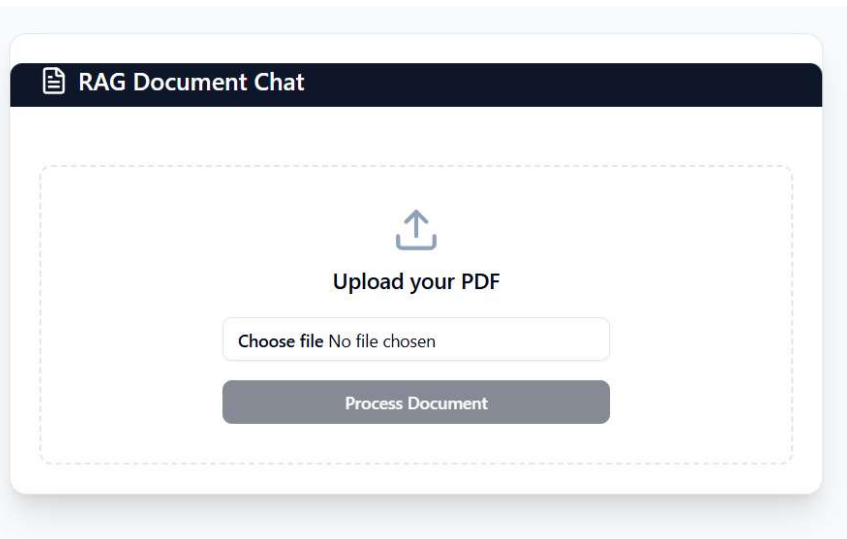
Step 2: Ask a Question

How often do the authors promise to send newsletters going forward?

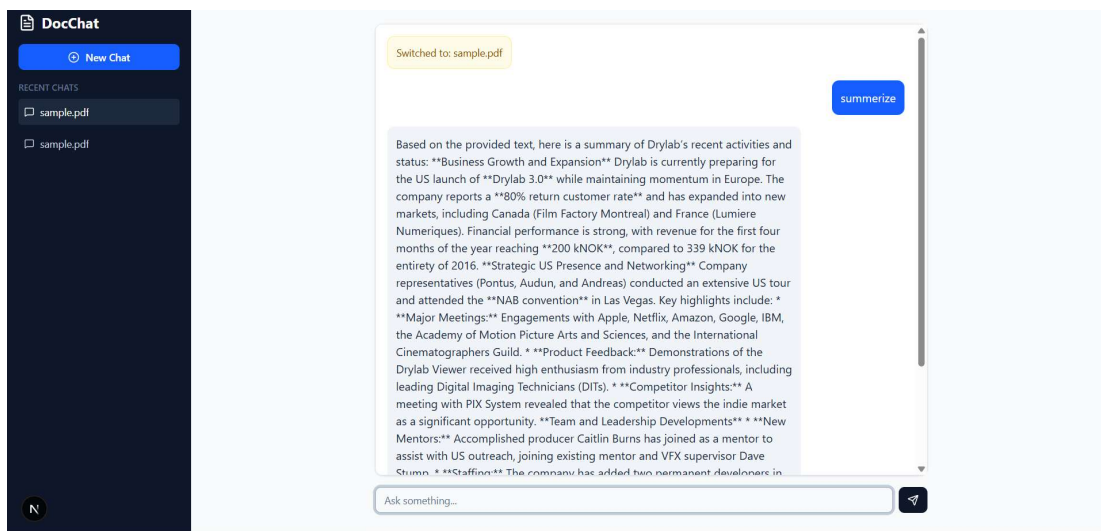
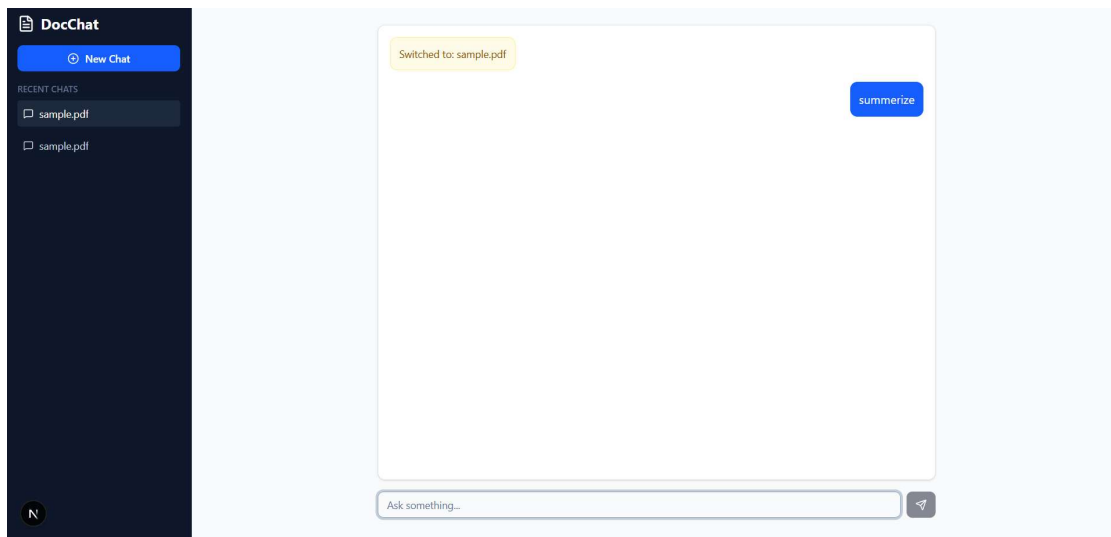
**Gemini says:**

Based on the context provided, the authors promise to send newsletters **\*\*every two months\*\*** hereafter.

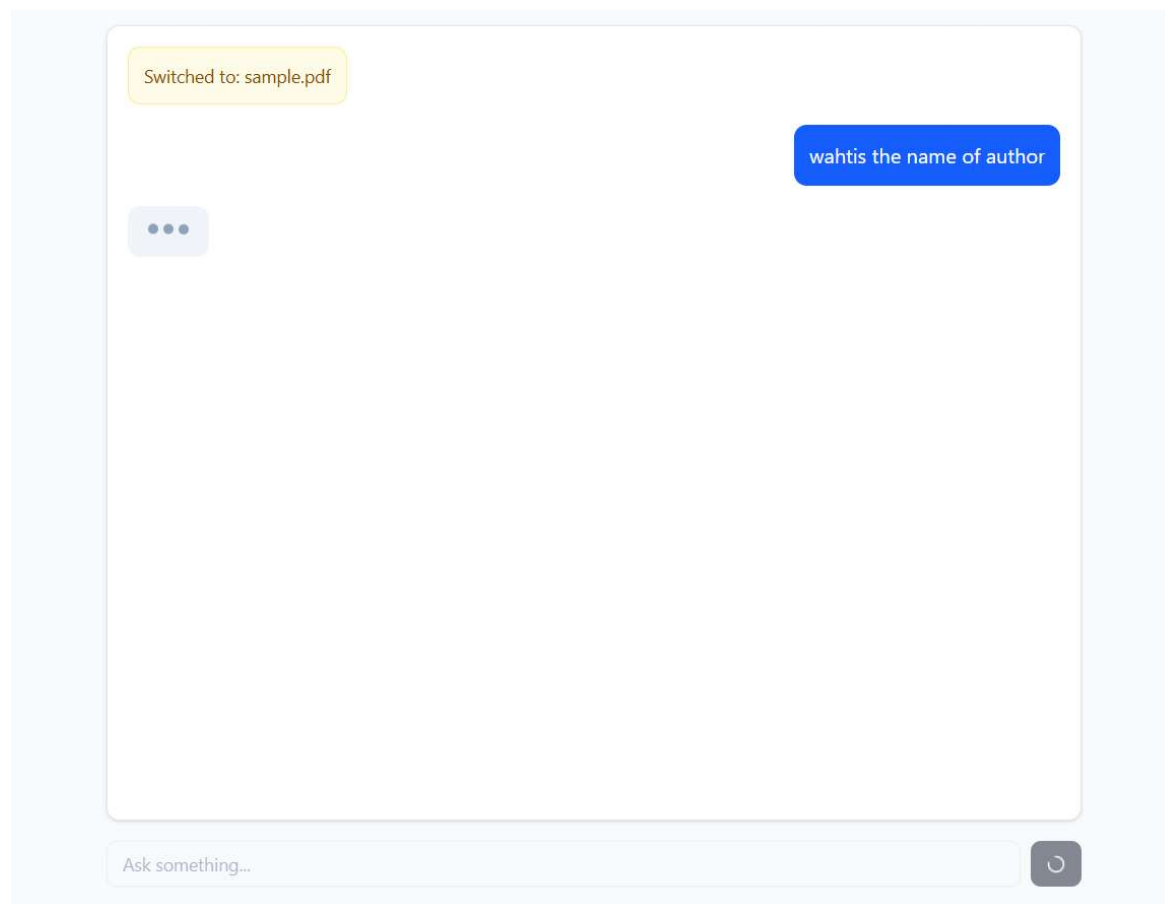
- In this version , I used vector embeddings for efficient data handling.
- Workflow :
- Pdf is hardcoded , and user simply asks questions.
- Before question is processed , pdf is divided into chunks , and stored in data base along with its vector embedding, created by gemini
- Question is then processed ( 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)

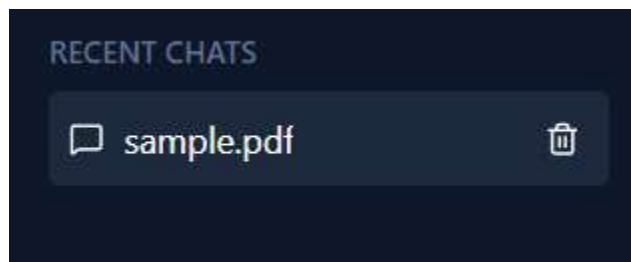
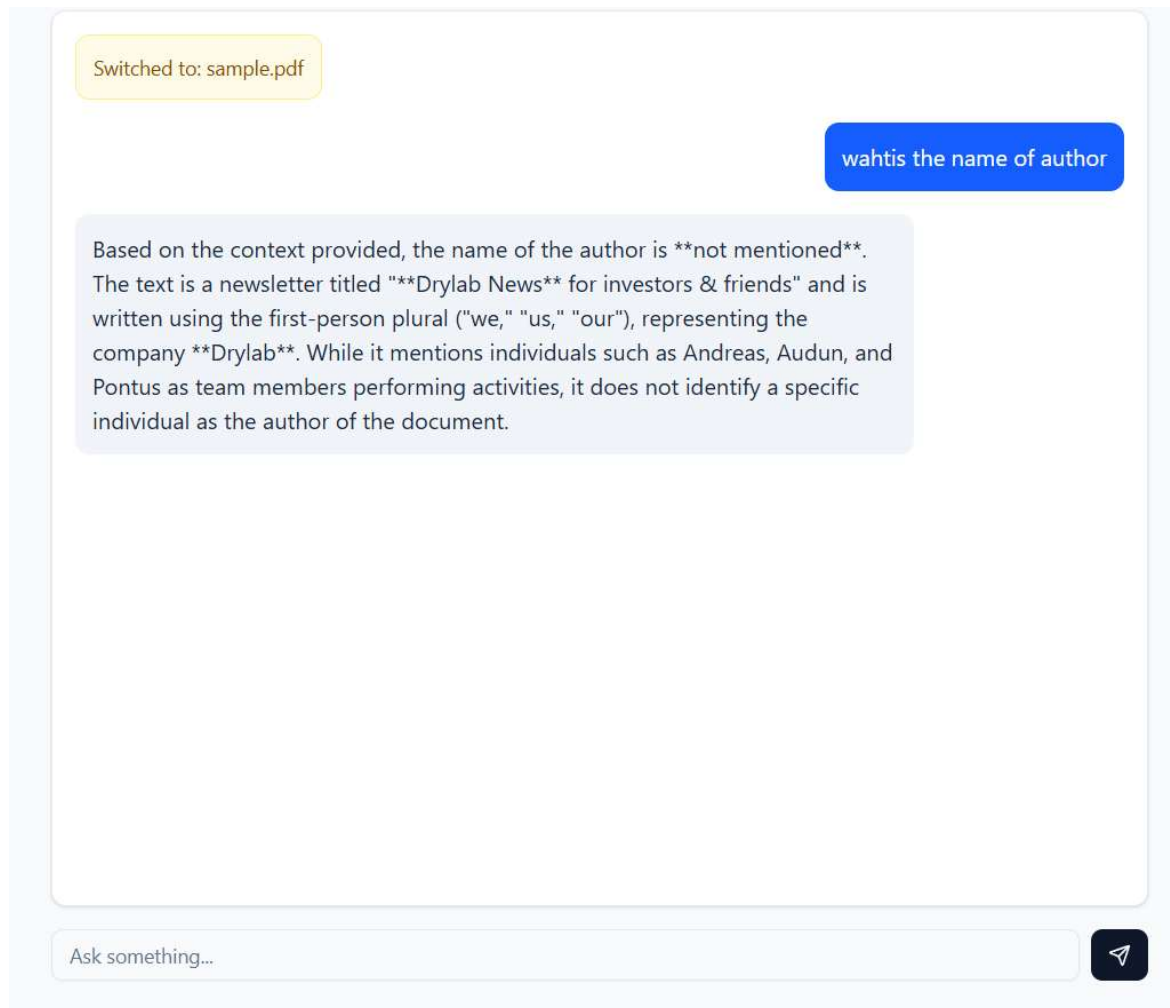


- Ingest.py is fused with main.py
- I have 2 routes not /upload and /chat
- /upload : take file as input, divide it into chunks , create vectors and store it in database.
- All the messages showed are stored in an array on client side.
- Cons :
  - Only one fiel can be uploaded
  - Sessions are not stored.

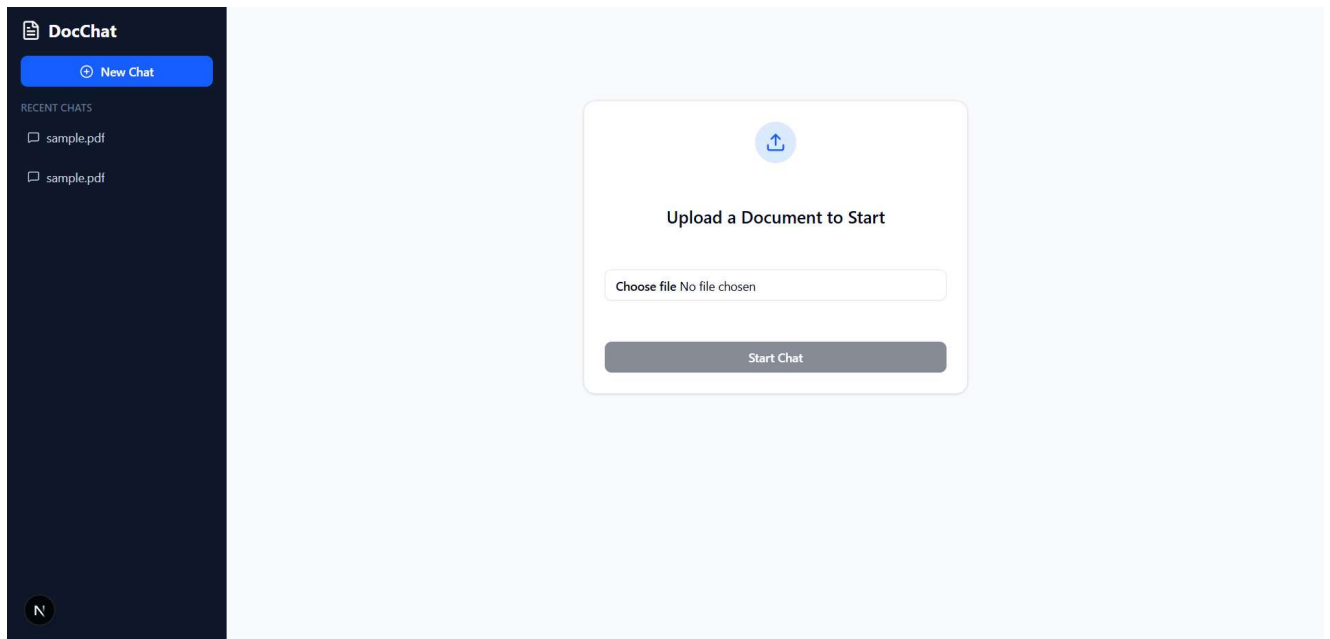


- Sessions created
- No loading options from system side
- No delete session option
- Chats are not saved , while switching it vanishes





- 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

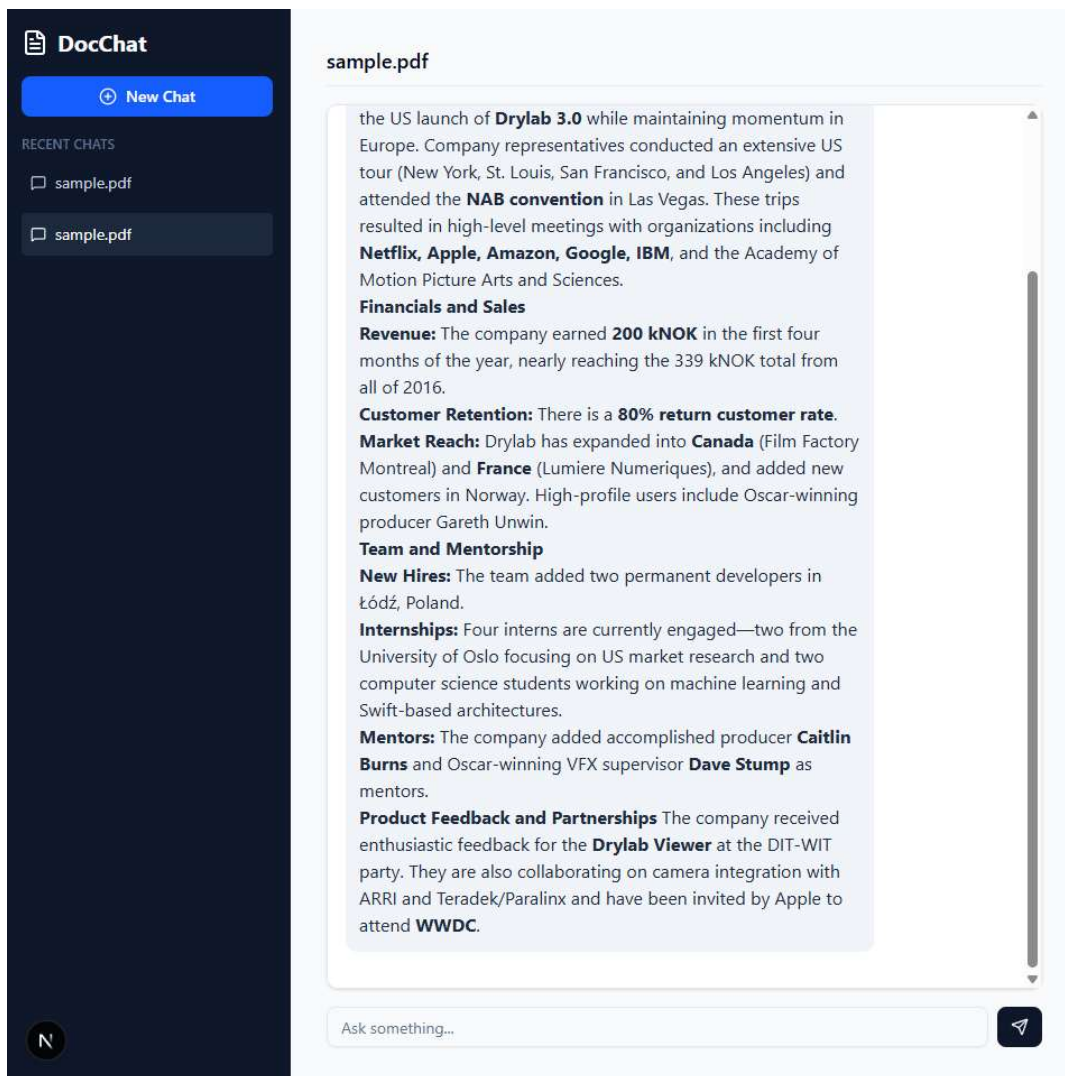
get_relevant_context(user_question: str, session_id: str):
```



Convert question 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
```

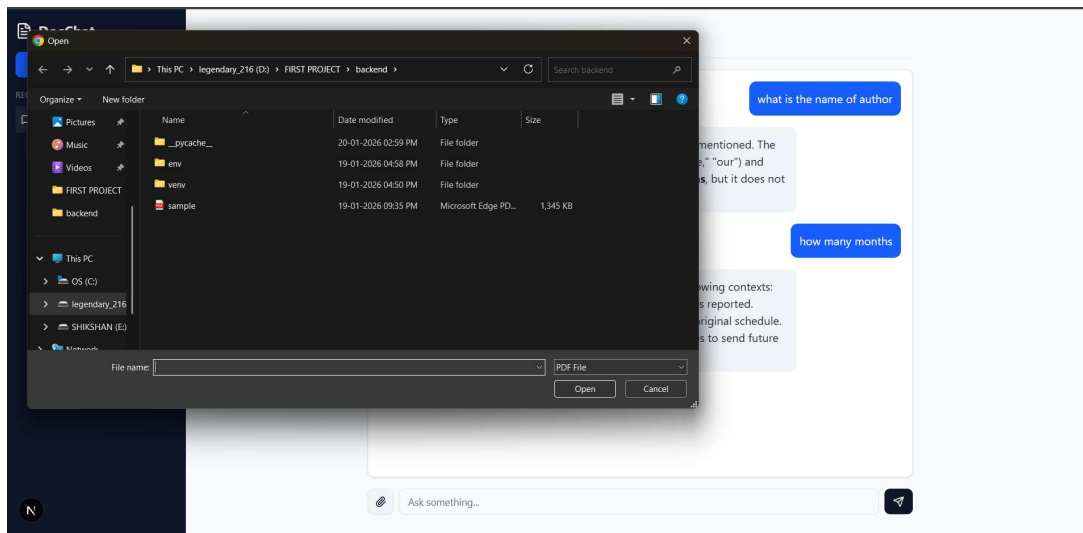
```
        length_function=len,
```

```
        is_separator_regex=False,
```

```
    )
```

```
    chunks = text_splitter.split_text(text)
```

```
    return chunks
```



sample.pdf

what is the name of author

Based on the context provided, the name of the author is not mentioned. The newsletter is written from the perspective of the company ("we," "our") and mentions team members such as **Pontus**, **Audun**, and **Andreas**, but it does not specify which individual wrote the text.

how many months

Based on the text provided, "months" is mentioned in the following contexts:  
**Four months:** The period for which the 200 kNOK revenue was reported.  
**One month:** The amount of time development is behind the original schedule.  
**Two months:** The frequency with which the company promises to send future newsletters.

Added document: sample.pdf

Ask something...

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).

explain headers and footers

Printed documents often include page headers and footers [2]. Page numbers are frequently printed at the bottom of the page, while the document title is typically shown at the top, with the exception of title pages [2]. In this specific document, headers and footers are discussed on page 2, which Prince can identify automatically using cross-references [1, 2].

what is footnotes say

A footnote is a note located at the bottom of a page that provides commentary or cites a reference for a specific part of the text [Page 2]. While some may believe footnotes are for information that should be hidden, they are essential to printed documents and often attract more attention than the main text [Page 2]. Statistics suggest that 9 out of 10 readers will read the footnotes before the text to which they are anchored, and they often contain the most interesting information [Page 2]. Prince is capable of automatically generating these footnotes for documents [Page 2].

## • Citations added

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



**Quota exceeded. Please wait 39 seconds.**

explain page 2 in 3 sentences

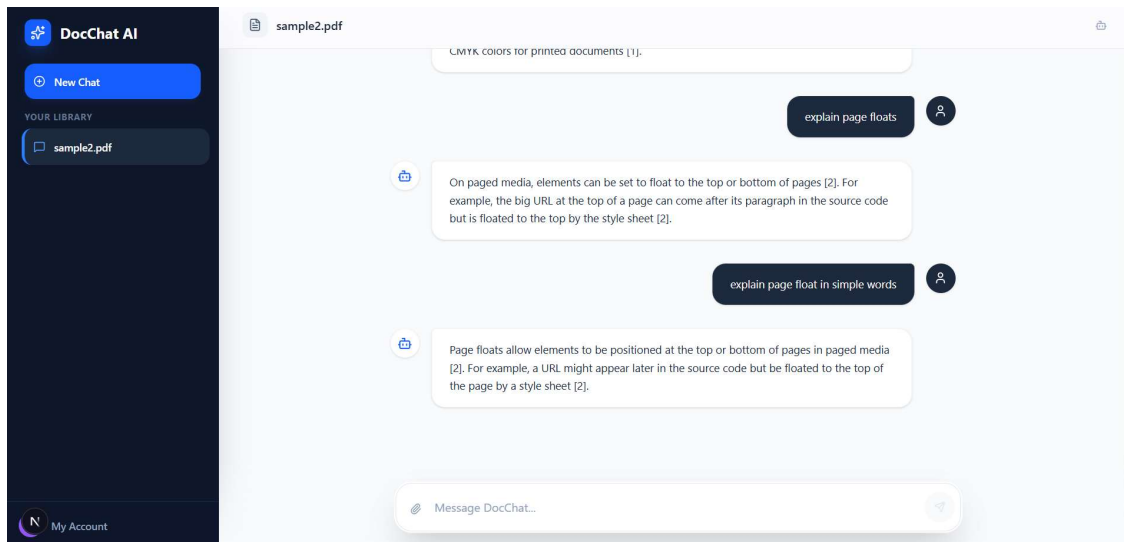
Drylab achieved an 80% return customer rate, generated 200 kNOK in revenue for the first four months, and expanded its team with permanent developers and various interns [2]. Caitlin Burns joined as a new mentor, following Oscar-winn

## Feature : "Real-Time Typing" (Streaming)

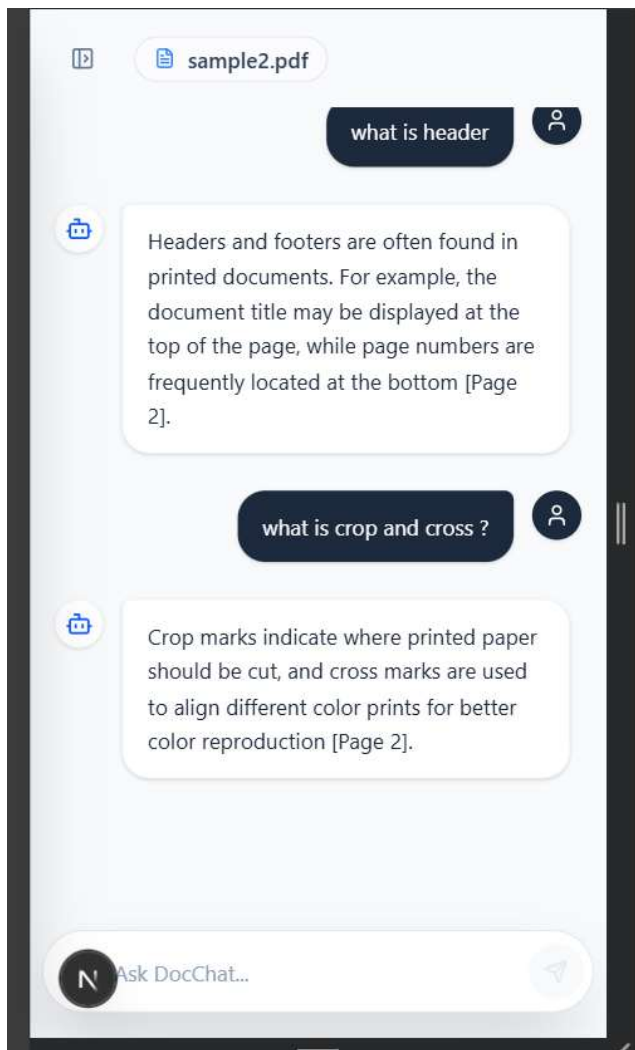
Right now, you click "Send"... wait 3 seconds... and **BAM**, the whole text appears at once. ChatGPT doesn't do that. It "types" the answer word-by-word.

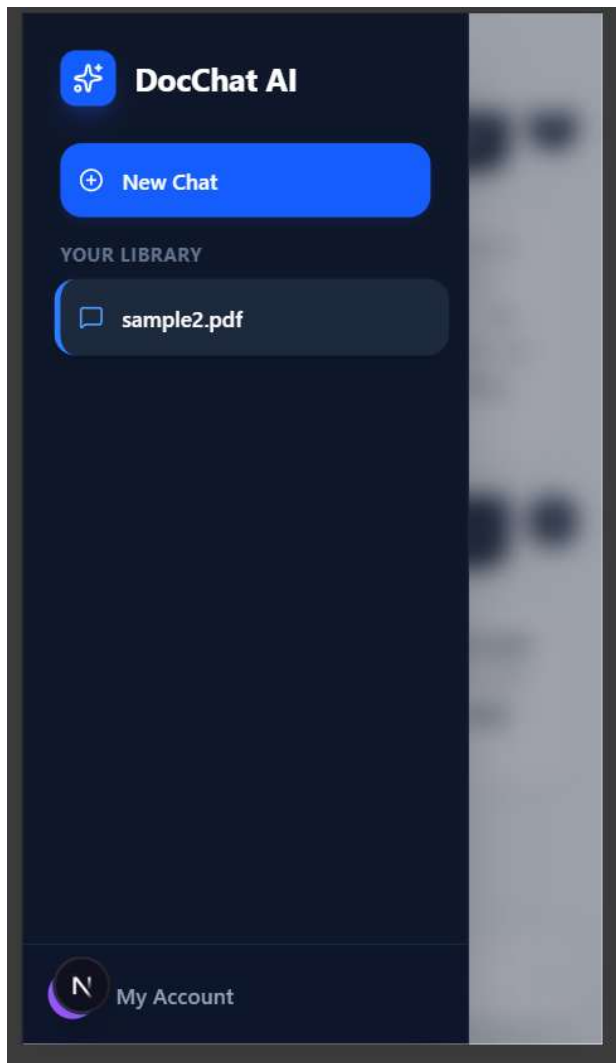
**The Upgrade:** Stream the response from the backend to the frontend.

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

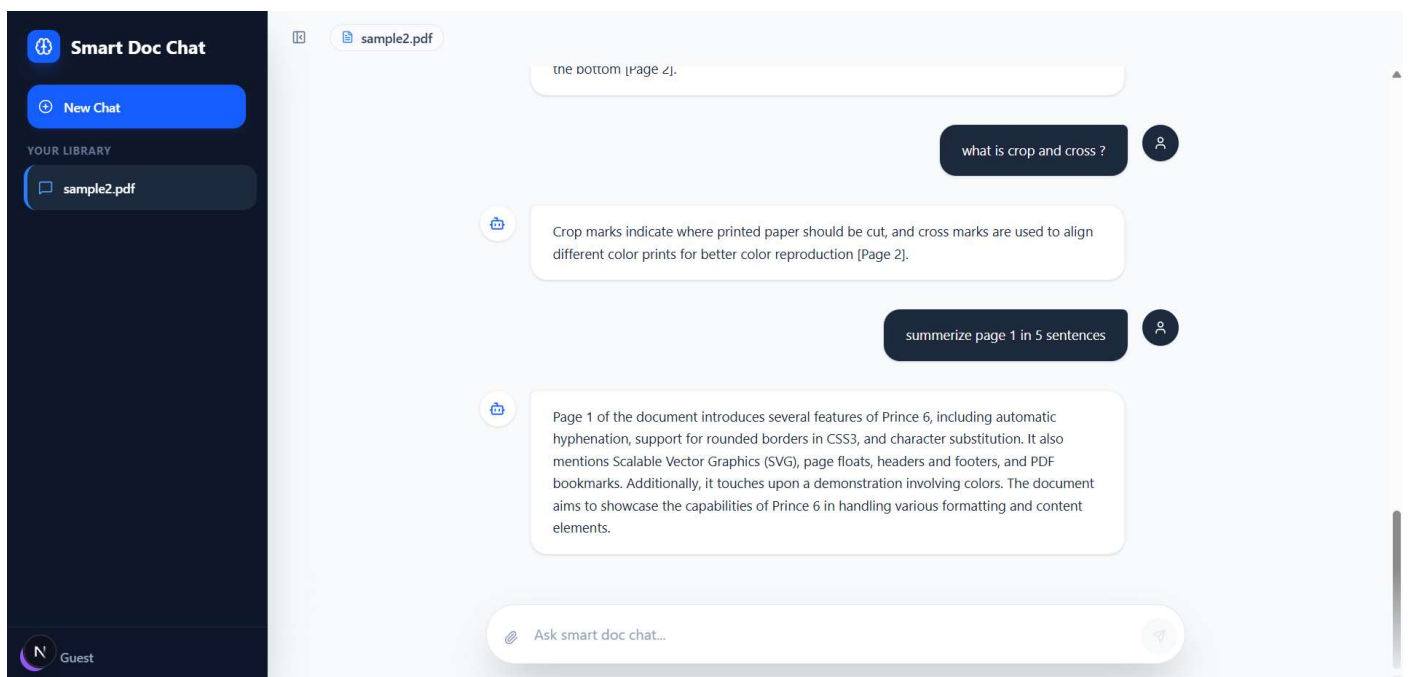


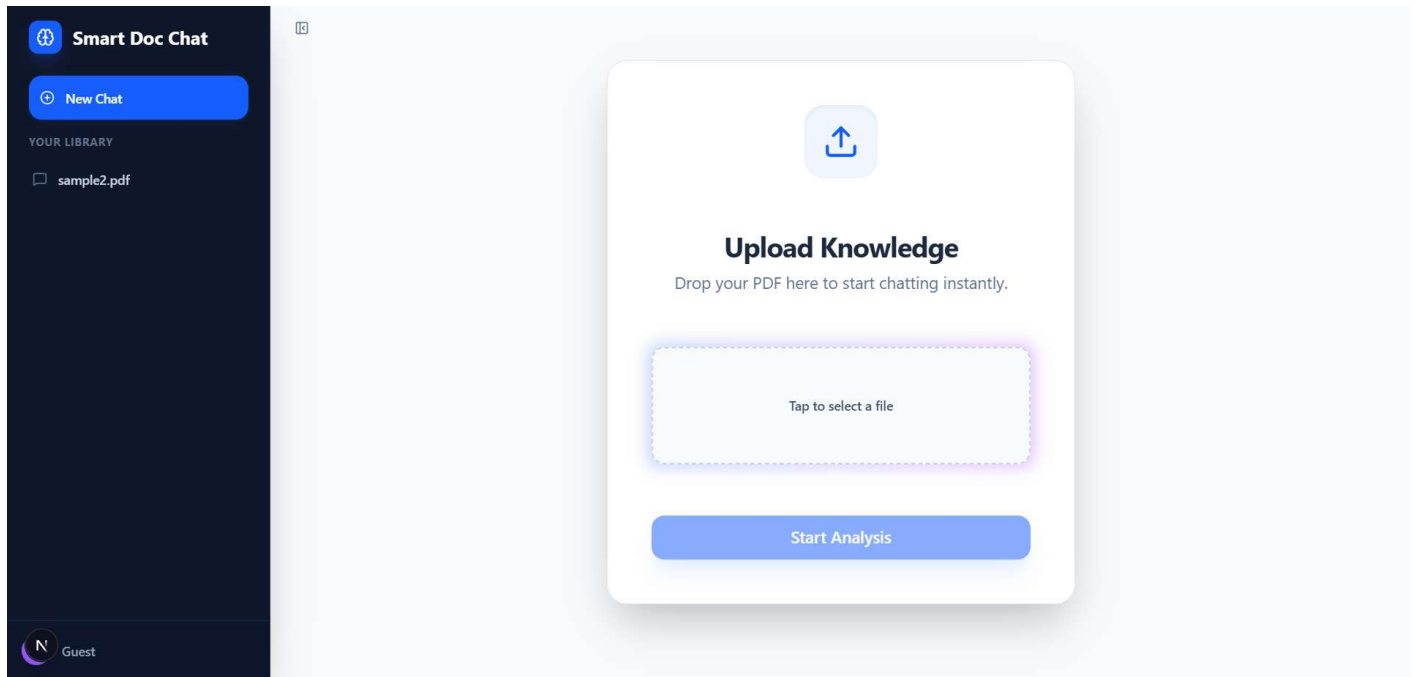
- UI added





- Responsiveness added





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