Docgen Amendments Reprised

This document outlines the key modifications and improvements made to the document generation feature after transitioning from Streamlit to Next.js. It focuses on enhancing both the frontend and backend integration, ensuring a more responsive and user-friendly experience.

Server Implemented on:
(http://ec2-54-79-231-211.ap-southeast-2.compute.amazonaws.com)

(Site: http://54.79.231.211/)

1) (main.py) Backend:

Path: cd danswer/docgen_backend/main.py:

Summary: In the **backend enhancements** for **FastAPI** (**main.py**) focused on implementing **incremental document generation**, allowing **sections** of the document to be **streamed** to the **frontend** in **real-time** as they are **generated**, by using **FastAPI's StreamingResponse** and a route for **handling POST requests** from the frontend, **triggering** the **Init Document Generation** process.

❖ Key Modifications:

- **FastAPI Route for Streaming:** The /**generate-doc endpoint streams** the **document generation results** using **FastAPI's StreamingResponse**. This ensures that the data is sent to the **frontend** as it is **generated**.

```
@app.post("/generate-doc")
async def generate_document(request: DocumentRequest):
    return
StreamingResponse(process_document_incremental(request.document_tit
le, request.document_info), media_type="application/json")
```

- **Streaming Document Generation:** The **core functionality** of the **backend** was **restructured** to allow **streaming** the **generated document** back to the **frontend** in real-time.

```
def process_document_incremental(document_title, document_info):
   # logic to process document generation in chunks
    yield json.dumps({ "output": clean_result, "title": title })
```

The **process_document_incremental function generates** and **streams document sections** and output incrementally by **yielding** each **chunk** of data as it's generated. This allows the **frontend** to display **content** as soon as it's available, rather than **waiting** for the **entire document** to be ready.

- **ChromaDB Integration:**

```
chroma_client = chromadb.Client()
collection = chroma_client.get_or_create_collection("summaries", embedding_function=embedder)
```

The **backend** uses **ChromaDB** to **query** and store **document sections** and **summaries**, ensuring the generation process is **optimized**. The collection is **queried** for additional data to generate more information **document sections**.

- **Document Prompt and Section Handling:**

```
init_prompt =
read_file("init_prompt.txt").format(document_title=document_title, document_info=document_info)
output = run_inference(init_prompt)
titles = get_titles(read_file(init_filepath))
```

The **document logic** uses **predefined templates** from **init.prompt.text** and **step.txt** to **generate** each section of the document **iteratively**, querying **ChromaDB** for additional information.

2) (page.tsx) Frontend:

Path: cd danswer/web/src/app/docgen/page.tsx:

Summary: In the **frontend** (**page.tsx**), **various updates** were made to support the new **real-time document generation process**, form **validation**, **UI improvements**, and user **experience enhancements**. These changes ensure better user **interaction** and more **dynamic updates** as the **backend streams results.**

❖ Key Modifications:

- **State Variables for Document Handling:**

```
const [documentType, setDocumentType] = useState<string>("Service agreement");
```

These **states** hold the **values** for the **document type** and **description** that the user **inputs**.

```
const [description, setDescription] = useState<string>("");
```

- **New State for Loading and Completion Feedback:**

```
const [isGenerating, setIsGenerating] = useState<boolean>(false); // Spinner state
const [isCompleted, setIsCompleted] = useState<boolean>(false);   // Completion message
const [warning, setWarning] = useState<string>("");               // Warning for validation
```

The **isGenerating state** manages whether a **spinner** is **shown** to indicate a **loading process**, and **isCompleted signals** when the **document generation** is **complete**. **warning** handles field **validation messages**.

- **Form Validation Before Submission:**

```
if (!documentType.trim() || !description.trim()) {
  setWarning("Please fill in both Document Type and Description.");
  return;
}
```

The **form validation logic checks** if both the **document type** and **description** are filled before **initiation** of the **request**. If not, a **warning message** is **shown**, preventing the **request** from being **sent**.

- **Handling Document Generation Request:**

```
const response = await fetch("http://54.79.231.211:8000/generate-doc", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ document_title: documentType, document_info: description }),
});
```

This **initiates POST request** to the **backend**, sending the user-provided **document title** and **description** to generate the **document**. You can access the **backend site** form here (**http://54.79.231.211:8000/generate-doc**).

- **Incremental Data Handling:**

```
const reader = response.body?.getReader();
const decoder = new TextDecoder();

if (reader) {
  let done = false;
  while (!done) {
    const { value, done: readerDone } = await reader.read();
    done = readerDone;
    const chunk = decoder.decode(value, { stream: true });
    const data = JSON.parse(chunk);
    if (data.title) setTitles((prev) => [...prev, data.title]);
    if (data.output) setOutput((prev) => prev + data.output);
  }
}
```

This **code** handles **reading** the **backend's streamed responses**. As each section of the **document** is **generated**, it is **immediately appended** to the **UI** for both the **titles** and the **generated output**.

- **Updates to docker-compose.dev.yml and Dockerfile in docgen_backend:**

Path: cd danswer/deployment/docker_compose:
Path: cd danswer/docgen_backend/Dockerfile:

Summary: As part of the **Docgen integration feature**, we also made **updates** to the **docker-compose.dev.yml file** and the **Dockerfile** inside the **docgen_backend** directory to **streamline** the **backend deployment** and integration into the **development environment**.

❖ **Key Modifications:**

- **Docker-compose.dev.yml Service Addition:** In the **docker-compose.dev.yml** file, a service for the **document generation backend** (**docgen_backend**) was **added**. This service ensures that the **FastAPI-based backend** runs in a **containerized environment**, simplifying the **development** process.

```
docgen_backend:
  build:
    context: ./danswer/docgen_backend
    dockerfile: Dockerfile
  ports:
```

- "8000:8000"
environment:
- APP_ENV=development
volumes:
- ./danswer/docgen_backend:/app
command: uvicorn main:app --host 0.0.0.0 --port 8000 --reload

Note: If needed we can relocate the cd danswer/docgen_backend to cd danswer/backend/danswer inside of here or keep it inside the home directory itself open for suggestions according to the relocated file we need to re-mention the context and configure docker-compose.dev.yml.

- **Dockerfile in docgen_backend Directory:**

The **Dockerfile** inside the **docgen_backend directory** defines the **backend service environment**, ensuring that **FastAPI** and its **dependencies** are correctly **installed** and executed **within** the **container** using **python:3.9-slim** working dir set to **/app** dependencies to be installed inside **requirements.txt** by using **–no-cache-dir** entire source does form **docgen_backend directory** is copied into the **container's** /app directory after that runs **FastAPI** with **Uvicorn CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000", "--reload"]** serving the app on port **8000 default**.

- **Ibrahim Sultan**