**1. High-level project layout**

Your repo root is:  
C:\Users\uipek\PycharmProjects\DocuChat\

Inside that, you have:

* .venv/ → your virtual environment (site-packages, torch, transformers, etc.)
* docuchat\_backend/
  + app/
    - core/
    - routers/
    - uploads/
    - auth.py
    - database.py
    - jobs.py
    - main.py
    - models.py
    - rag.py
    - schemas.py
    - storage.py
    - **init**.py
  + chroma\_db/ (Chroma persistence with sqlite + collection dirs)

That is VERY good. You’ve already split “backend app” code from vector DB persistence. This is a real project now, not a toy script.

**2. Backend app structure (this is the heart)**

From docuchat\_backend/app/ I see these Python modules:

* main.py
* auth.py
* database.py
* jobs.py
* models.py
* rag.py
* schemas.py
* storage.py
* \_\_init\_\_.py  
  Plus two internal packages:
* core/
* routers/

**core/**

* core/\_\_init\_\_.py
* core/config.py
* core/\_\_pycache\_\_/...  
  This tells me you already externalised config (likely env vars like JWT secret, DB URL, etc.).

→ I will treat core.config as the single source of truth for settings (JWT\_SECRET, DB connection string, upload folder path, Chroma path, etc.). Any new code I write will import from core.config, not hardcode values.

**routers/**

Your API is split by concern (clean 👌):

* routers/auth.py
* routers/documents.py
* routers/health.py
* routers/query.py
* routers/\_\_init\_\_.py
* routers/uploads/ (which currently contains uploaded sample docs and test PDFs/TXT you’ve been using for OCR and telco content)

→ I will extend this pattern.  
When you ask me for a new endpoint, I’ll put it in the correct router file:

* login/register → routers/auth.py
* upload/list/delete docs → routers/documents.py
* ask a question / RAG answer → routers/query.py
* health check / status → routers/health.py

Exactly matching your naming.

**3. Persistence and data layer**

You have:

* database.py
* models.py
* schemas.py

That strongly suggests:

* models.py = ORM models (probably SQLAlchemy models for users, documents, etc.).
* schemas.py = Pydantic (or similar) request/response models for validation.
* database.py = session / engine / Base.

So going forward:

* If I add a new table (e.g. collections), I will add it to models.py.
* If I add a new API request/response shape, I will add the Pydantic model to schemas.py.
* I will NOT create new random folders like services/db/ unless we explicitly decide to refactor.

**4. RAG / AI logic**

You have:

* rag.py in app/
* chroma\_db/ at docuchat\_backend/chroma\_db/ containing:
  + chroma.sqlite3
  + generated UUID folders like 709d98e2-50c2-4b81-bae1-97da8f4deacc/...

So:

* You’re already persisting embeddings locally (Chroma persistent client, not in-memory).
* rag.py is almost certainly doing:
  + chunking
  + embedding
  + similarity search
  + answer synthesis

→ Any retrieval or Q&A work I generate (like /api/query) will call helper functions from rag.py, not re-implement RAG logic inside the router.

That’s perfect separation. You’re already thinking like production.

**5. File handling / OCR**

I see two upload roots:

**A. app/uploads/**

Example files:

* fraud\_detection\_demo\_script3.txt
* Radisys internal PDFs / guides
* user1\_20251024\_222055\_459016\_Raising Radisys MRF Support Ticket.pdf
* user1\_20251024\_222307\_506099\_vMS\_User\_Guide\_...pdf  
  and so on. These look like runtime user uploads, saved with timestamp+random suffix (good for uniqueness).

**B. app/routers/uploads/**

Example test files:

* OCRtest.pdf
* Feature Description - NR CBRS...pdf
* test1.pdf
* etc. These look like manual test artifacts you were using while building/exercising the OCR flow.

→ Interpretation:

* Real prod uploads should live in app/uploads/ (which matches your timestamped naming).
* routers/uploads/ is more like “dev sandbox while I was testing OCR/endpoint code in the router”.

Going forward, I will assume:

* official upload directory = app/uploads/
* temporary test assets can go away / won’t be referenced in code paths I generate

Also, I saw evidence of OCR test docs (OCRtest.pdf, pdf\_digitalizado.pdf) which tells me you’re already doing OCR on scanned PDFs, and you confirmed that earlier in chat.

So when I write documents.py upload logic, I will:

1. Save file under app/uploads/ with a safe unique name.
2. Pass that saved path to an OCR/text extraction step.
3. Push extracted chunks into Chroma (under chroma\_db/).
4. Create/update the DB entry for that document via models.Document (status=ready etc.), matching what’s in your requirements doc (FR-2.6, FR-2.7, FR-2.8).

**6. Other notable modules**

* storage.py exists at the app root. That likely manages disk paths / saving files / maybe S3 later. I’ll reuse it for any new file persistence instead of inventing a new utility.
* jobs.py is interesting. That sounds like background or queued work (e.g. post-upload embedding generation). Even if you’re not running a queue yet, I’ll respect that boundary and keep “slow tasks” in there so it scales to async workers later.

**7. Health check and status**

You already have routers/health.py. That usually means you’re exposing something like /health or /ready. I will keep using that for uptime checks and monitoring (NFR-4.1 uptime, NFR-4.2 graceful error handling from your requirements doc).

**8. Environment / config**

Your core/config.py is going to be critical. I saw earlier you had:

SECRET\_KEY = os.getenv("JWT\_SECRET", "change-this-in-production")

ALGORITHM = os.getenv("JWT\_ALG", "HS256")

(from your previous message in an earlier session, when we debugged JWT)

So:

* I will assume auth depends on JWT\_SECRET and JWT\_ALG from env.
* Any new auth/refresh-token logic I build will import from that config, not hardcode.

**9. TL;DR canonical structure I will now stick to**

I will treat your backend like this:

docuchat\_backend/

app/

core/

\_\_init\_\_.py

config.py

routers/

\_\_init\_\_.py

auth.py # /auth endpoints (login/register/etc)

documents.py # /documents upload/list/delete/summary

query.py # /query RAG Q&A

health.py # /health status

uploads/ # (dev/test files right now)

uploads/ # real stored user documents (timestamped names)

auth.py # JWT helpers / token validation

database.py # DB engine/session/Base

jobs.py # async-ish processing hooks

main.py # app entrypoint / FastAPI-or-Flask app wiring

models.py # ORM models (users, documents, etc.)

rag.py # embedding + retrieval logic

schemas.py # Pydantic-style request/response models

storage.py # file persistence helpers

\_\_init\_\_.py

chroma\_db/

chroma.sqlite3 # persisted Chroma vector store

<uuid>/\* # collection data

Plus the project root:

DocuChat/

.venv/ # local venv with transformers, torch, etc.

All of that is coming directly from your uploaded project\_structure.txt.

**What happens next**

From this point:

* When you ask me “write the /api/documents/upload route”, I will add/modify routers/documents.py.
* When you ask for DB changes (like Collections table), I’ll extend models.py and schemas.py.
* When you ask for query answering, I’ll implement it in routers/query.py but call into rag.py.
* When you ask for config/env, I’ll wire it through core/config.py.

In other words: I now understand your real structure, and I will not fight it. We’re building on top of *your* codebase, not starting fresh.

You did great by exporting that file, by the way. This gives me enough to behave like a teammate inside your repo.