DocuChat Project

DocuChat Project (Merged MVP + Production Hardening + Agent)

1) Overview

Build **DocuChat**, a small knowledge assistant where authenticated users upload technical documents and chat with an AI that answers using **Retrieval-Augmented Generation (RAG)**, returning **verifiable citations** from the uploaded docs. The single delivery includes:

- A **shippable MVP** (end-to-end happy path)
- **Production-leaning hardening** (security, reliability, operability)
- An **agentic workflow** (basic planning + tool use on top of RAG)

2) Timeline & Effort Guidance

- **12 calendar days total**
- **~34–48 hours of effort total**
- Partial but well-documented work is acceptable—clearly list trade-offs and next steps.
- Include a brief **high-level time log** in `docs/DECISIONS.md`.

---

3) Stack (Fixed)

- **Frontend:** React + TypeScript, Ant Design, Tailwind CSS
- **Backend:** Python + Django (DRF allowed), WebSockets (Django Channels)
- **Infra:** Docker, NGINX, Redis, PostgreSQL
- **Auth:** Keycloak (OIDC)
- **AI:** LLM via API, embeddings, RAG, plus a lightweight agent that can plan + call simple tools
- **DevEx:** GitHub (branching + Actions), basic tests
- **Docs:** Technical docs + ADRs (Architecture Decision Records)

---

4) Deliverables & Repo Layout (Required)

GitHub repo with:

```
/frontend        # React+TS, AntD, Tailwind
/backend         # Django (+DRF), Channels/WebSockets
/infra           # docker-compose.yml, nginx configs, keycloak realm export/scripts
/docs            # README.md, ARCHITECTURE.md, API.md, DECISIONS.md, OPERATIONS.md, ADRs
/.github/workflows/ci.yml

```

Docs required:

- `docs/README.md` – how to run locally (clean clone → running)
- `docs/ARCHITECTURE.md` – diagrams + data flow + components
- `docs/API.md` – endpoints + request/response examples + websocket events

- `docs/DECISIONS.md` – time log + cost cap + major trade-offs
- `docs/OPERATIONS.md` – runbooks: env vars, migrations, reindexing, troubleshooting
- **ADRs: 5–6 total** (merged scope). Each ADR should mention if an LLM helped draft it or generate code (per LLM policy).

---

5) Responsibilities (Setup & Infrastructure)

You are responsible for:

A) Docker + NGINX (must work from clean clone)

Provide a **working** `docker-compose.yml` that brings up:

- frontend
- backend
- postgres
- redis
- keycloak
- nginx

NGINX must reverse-proxy:

- `/` → frontend
- `/api` → backend HTTP
- `/ws` → backend WebSocket (Channels)

B) Keycloak realm/client configuration

- Configure a Keycloak **realm + OIDC client** for frontend login.
- Provide either:
  - a **realm export JSON** in `/infra/keycloak/realm-export.json`, or
  - a reproducible scripted setup documented in `/docs/OPERATIONS.md`.

C) Seed dataset (3–5 docs)

Include **3–5 small technical docs** (PDF/Markdown/Text) under something like:

- `/backend/seed_docs/` or `/docs/seed/`

D) Environment samples

Provide `.env.sample` for:

- frontend
- backend
- infra (compose / nginx / keycloak if needed)

Document every variable in `docs/OPERATIONS.md`.

E) Secrets policy

- Do **not** commit secrets.

- `.env.sample` should contain placeholders.

---

6) Product Requirements

Core user story

> "As an authenticated user, I can upload docs, see indexing progress, and ask questions that are answered with citations to my docs."

Must-haves (end-to-end functional)

1. **Auth (Keycloak OIDC)**

- Frontend login via Keycloak (OIDC Authorization Code + PKCE recommended).
- Backend (Django) validates JWT access tokens:
  - Signature verification using Keycloak JWKS
  - Issuer + audience validation
  - Role mapping (at least basic `user` role)
- API endpoints require auth by default (explicit allowlist for health endpoints if any).

2. **Document Upload & Indexing (≤20 files/user or per workspace)**

- Upload up to **20 files** (pdf/md/txt at minimum).
- Store original file metadata (name, size, type, upload time, status).
- Extract text:
  - PDF text extraction (best-effort; handle failures gracefully)
  - Markdown/text straightforward parsing
- Chunking strategy:
  - Deterministic chunking with overlap
  - Store chunk text + chunk index + source doc reference
- Embeddings:
  - Generate embeddings for chunks via embedding model API
  - Store embeddings in a queryable store (recommended: Postgres + `pgvector`)

3. **RAG Ask Endpoint**

- `POST /api/chat/ask`
- For each question:
  - Retrieve **top-k** relevant chunks scoped to the user/workspace
  - Call LLM with a prompt that instructs the model to:
    - Answer only using provided context
    - Produce citations (doc + chunk references)
    - Say "I don't know" when context is insufficient
- Return:
  - `answer` (text)
  - `citations[]` including doc id/name + chunk ids + short quoted snippet (short, not huge)

- optional `debug` fields behind a flag (retrieval scores, model used, token usage)

4. **Realtime progress (WebSockets)**

- During indexing, emit progress events:
  - upload received
  - text extracted
  - chunked (N chunks)
  - embedding progress (i/N)
  - completed / failed with reason
- Frontend must display progress per file and overall queue status.

5. **UI**

- Pages:
  - **Login**
  - **Uploads** (list docs + upload button + progress)
  - **Chat** (ask questions, show answers with citations)
- Citations must be clearly visible and linkable to the source doc/chunk.

6. **Local run via Docker**

- `docker compose up` should bring the full system up locally with NGINX as the single entrypoint.

7. **GitHub workflow**

- Branching → PRs
- `.github/workflows/ci.yml` must run:
  - frontend: install + lint + TypeScript typecheck + basic tests
  - backend: install + lint/format check + unit/API tests
  - (optional) docker build smoke check if time allows

---

7) "Production-Leaning" Hardening Requirements

Reliability & async processing

- Indexing should not block request threads:
  - Use a background worker pattern (e.g., Django + Celery strongly recommended) with Redis as broker.
  - If you avoid Celery, document the alternative clearly (and its limitations).
- Idempotency:
  - Re-uploading the same file should not silently duplicate chunks (define behavior; document it).
- Retries:
  - Embedding/LLM API calls should have bounded retries with exponential backoff.

Security & access control

- Enforce multi-user separation (at least per-user scoping; optional workspace/team scoping).
- Validate file type/size limits; store safely.
- Rate limiting (basic):

- `/api/chat/ask` and uploads should have server-side limits (even simple in-memory or Redis-based).
- Audit-ish logging:
  - Record who uploaded what and who asked what (minimal metadata; avoid storing sensitive content unnecessarily).

Operability

- Health endpoints (at least backend health + worker health if applicable).
- Structured logs (JSON preferred) and correlation IDs if feasible.
- Clear runbook actions:
  - reindex a doc
  - delete a doc and its chunks
  - rotate keys / update Keycloak client settings
  - change embedding/LLM models

Data & schema quality

- Use migrations; no manual DB setup steps.
- Define clear models (example):
  - `Document`
  - `DocumentChunk`
  - `IndexJob` / `IndexTask` (status/progress)
  - `ChatSession` + `ChatMessage` (optional but helpful)

---

8) Agentic Workflow Requirements

Goal

Add an **agent mode** that can:

- Take a user goal/question
- Decide whether plain RAG is enough
- If needed, run a short plan and use simple tools safely

Minimal agent behavior (must implement)

- Endpoint: `POST /api/agent/run`
- Inputs:
  - `goal` (string)
  - optional `constraints` (e.g., “only use my docs”, “max 3 steps”)
- Agent loop (bounded):
  - Create a plan of 2–5 steps
  - Use tools with strict limits
  - Produce a final answer with citations when doc-grounded

Tools (simple + safe)

Implement **at least 2 tools**:

1. `search_docs(query)` → returns top-k chunks (same retrieval layer)
2. `open_citation(doc_id, chunk_id)` → returns the chunk text + metadata (for quoting/verification)

Optional tools (nice-to-have):

- `summarize_doc(doc_id)`
- `list_docs()`

Guardrails (required)

- Hard cap on steps (e.g., max 5 tool calls)
- Hard cap on retrieved context size
- Always cite sources when claims rely on docs
- If tools don't provide enough evidence, agent must say so

---

9) API Contract (Minimum)

Document in `docs/API.md` with examples.

HTTP

- `GET /api/me` – returns authenticated user info and roles
- `POST /api/docs/upload` – multipart upload
- `GET /api/docs` – list docs + statuses
- `DELETE /api/docs/{id}` – delete doc + chunks
- `POST /api/chat/ask` – RAG answer with citations
- `POST /api/agent/run` – agent run with tool traces (optional trace field)

WebSocket

- `/ws/indexing` (or similar)
- Events (example payload fields):
  - `type`: `"index_progress" | "index_complete" | "index_failed"`
  - `doc_id`
  - `stage`
  - `progress`: `{current, total}`
  - `message`

---

10) Testing Requirements (Meaningful but scoped)

Backend (required)

- Unit tests for:
  - token validation / auth middleware
  - chunking logic determinism
  - retrieval query scoping (user can't access others' chunks)

- API tests for:
  - upload → index job created
  - ask → returns answer + at least one citation when docs exist

Frontend (required)

- Basic test(s):
  - renders login state correctly
  - chat view renders citations
- TS typecheck must pass in CI.

---

11) ADR Requirements

Create **5–6 ADRs** in `/docs/adrs/` (format: `ADR-0001-*.md`).

Recommended ADR topics:

1. **Auth approach** (OIDC PKCE, token validation strategy, roles)
2. **Vector store choice** (e.g., pgvector vs external; why)
3. **Indexing architecture** (Celery vs alternative; progress events)
4. **Retrieval strategy** (chunk size/overlap, top-k, re-ranking yes/no)
5. **Prompting & citation format** (how citations are produced + verified)
6. **Agent design** (bounded tool loop, guardrails, tool interface)

Each ADR should include a short note like:

- "LLM assistance: used for initial draft / code scaffolding; manually reviewed and adjusted."

---

12) Cost Cap (Required)

In `docs/DECISIONS.md`, specify:

- LLM provider + model(s)
- Embedding model
- A sensible local-dev cost cap (example: "target ≤$5 total during evaluation")
- How you avoid runaway usage (rate limits, step caps, max tokens)

---

13) Acceptance Criteria

From a clean clone, using your `.env.sample`, reviewers can:

1. `docker compose up`
2. Open `http://localhost/`
3. Login via Keycloak
4. Upload the included seed docs (3–5)
5. See **indexing progress** in real-time
6. Ask a question and receive:
   - a coherent answer

- **at least one citation** pointing to the uploaded docs

7. Run agent mode on a goal and see:
   - bounded steps
   - tool usage (optionally visible)
   - citations when doc-grounded

---

14) DECISIONS.md Time Log (High-Level Template)

In `docs/DECISIONS.md`, add something like:

- Day 1: Infra + Keycloak + auth wiring (Xh)
- Day 2: Upload/index pipeline + progress events (Xh)
- Day 3: RAG endpoint + UI wiring + citations (Xh)
- Day 4–6: Hardening (async worker, retries, rate limit, tests) (Xh)
- Day 7–9: Agent mode + tools + guardrails (Xh)
- Day 10: Docs polish + CI stabilization (Xh)