AlgorithmX Assessment

(AI Engineer Intern)

Design and deliver a **PDF‑based Retrieval‑Augmented Generation (RAG) application** that lets a user upload one or more PDFs, indexes them into a vector database, and then **chat** with the content via a **Streamlit** UI powered by **Gemini**. Persist run/user history and evaluation telemetry in **PostgreSQL**.

Basic Requirements-

- **Stores & retrieves embeddings** in a **vector database (Qdrant)**.
- **Persists user/run history** and metrics in a **relational database (PostgreSQL)**.
- **Ingests a pdf file** and stores the embeddings in a vector database.
- A simple chat interface (**Streamlit)**
- **Gemini API keys** for LLM interaction.

---

1) Objectives

- **Primary**: Accurate retrieval over PDF content and coherent LLM answers grounded in retrieved passages.
- **Secondary**: Clean architecture, robust database usage, and a repo that is easy to run and review.

2) Tech & Services (required unless marked optional)

- **Vector DB:** Qdrant
- **Relational DB:** PostgreSQL
- **LLM:** Gemini (API key via env)
- **Embeddings:** sentence‑transformers (model via env)
- **UI:** Streamlit
- **API:** FastAPI or Flask (FastAPI preferred)

- **Containerization:** Docker Compose (recommended, not necessary)
- **PDF parsing:** pypdf / pdfminer.six / pymupdf (your choice, justify in README)

3) Streamlit UI Requirements

- Upload area for **PDFs** with progress indicator.
- Files panel listing indexed documents & their status.
- Chat area with message **bubbles**.
- Each answer must include citations (doc/page) and a collapsible “**show context**” section.
- Session switcher and clear chat button.
- Settings drawer: **top_k**, filter by document, model selector, and toggle for “only answer if sources found”.

4) Scoring Rubric (100 pts)

1. **Accuracy of Retrieval (35 pts)**
   - Correct chunks in top_k on sample questions; clear citations; sensible chunking & filters.
2. **Architecture (25 pts)**
   - Separation of concerns, configurability, error handling, clean prompts, docstrings.
3. **DB Interactions (20 pts)**
   - Proper schema, non-blocking writes, parameterized queries/ORM, minimal migrations.
4. **Repo Maintenance & Ease of Use (20 pts)**
   - One-command run (`docker compose up`), `.env.example`, README with screenshots, make targets, tests.

Bonus (up to +10): auth, streaming tokens to UI, filters UI, doc viewer with page highlighting, Helm chart, or a tiny load test script.

5) Timeline-

You have **24 hours from when you receive the email** to submit. **Late submissions won’t be accepted**

6) How to submit?

Create a **Public Github repository** and push your finished code into the repo and share the link as a reply to this email along with a working video demo of the finalised product.