Project: Financial Filings & Earnings Call RAG Assistant

One-liner:

"An LLM-powered assistant that answers investor-style questions over annual reports, 10-Ks/10-Qs, and earnings call transcripts, with cited sources and basic evaluation."

This hits:

- ✅ **Finance domain** (reports, calls, ratios, risk factors, segments, etc.)
- ✅ **RAG & LLMs** (embeddings, vector search, context injection, prompt design)
- ✅ **Realistic 2–3 week scope** for 2 people
- ✅ Looks like something a fintech or quant/ML team would *actually* use

---

What the app actually does (feature-level)

MVP features:

1. **Company-specific Q&A**
    - User chooses a company (e.g. "AAPL", "TSLA" – or EU equivalents).
    - Asks questions like:
        - "What are the main risks highlighted in the latest annual report?"
        - "How did revenue from Europe change vs last year?"
        - "What did management say about AI strategy in the last earnings call?"
    - App returns:
        - Short, clear answer.
        - **Cited paragraphs** from the filings/calls.
2. **Cross-company comparisons (stretch but impressive)**
    - "Compare the revenue growth and margin trends of Company A vs Company B."
    - System retrieves context from both companies and outputs a structured answer.
3. **Source viewer**
    - For each answer, show which **document + section** it came from.

    - E.g. “2024 Annual Report → Risk Factors → Page 35–36”.

4. **Basic evaluation dashboard**
    - A small list of predefined questions + “reference answers”.
    - You log:
        - Whether the RAG answer is correct (manually labelled).
        - Whether citations are relevant.
    - Show simple metrics (accuracy %, hallucination rate).

That’s enough to strongly demonstrate RAG/LLM in a finance context.

High-level architecture

You can keep this **simple and clean**:

1. Data ingestion

- Input sources:
    - Annual reports (PDF or HTML).
    - Quarterly reports.
    - Earnings call transcripts (plain text or PDF).
- Pipeline:
    - Convert PDF → text.
    - Split into chunks (e.g. 500–1,000 tokens with some overlap).
    - Attach metadata: { company, document_type, year, section, page, chunk_id }.

2. Embeddings + Vector Store (RAG)

- Use a sentence/paragraph embedding model (e.g. a common finance-capable one).
- Store chunks + embeddings in:
    - ChromaDB / Qdrant / FAISS (any simple vector DB).

3. Query pipeline

Given: company(s), user question:

1. Build a “search query” (maybe just the user question + company name).

2. Embed the query.
3. Retrieve top-k chunks (e.g. k = 5–10).
4. Construct prompt:
    - System message: “You are a financial analyst. Only use the provided CONTEXT. If info is missing, say you don’t know. Always cite sources with [company, document, year, section, page, chunk_id].”
    - Context: concatenated retrieved chunks (with metadata).
    - User message: original question.
5. Call LLM → answer + citations.

4. Backend

- Python + FastAPI (or Flask).
- Endpoints:
    - /ask – POST { question, companies } → answer + sources.
    - /eval/questions – GET predefined evaluation questions.
    - /eval/answer – POST manual rating of an answer.

5. Frontend / UI

- Option A (fast): **Streamlit** app:
    - Left sidebar: choose company/companies, document type filters.
    - Main area: chat interface + answer + citations + “show source” expander.
    - Evaluation tab: list predefined Qs, show model answers, let user mark “correct/incorrect”.
- Option B: Simple React UI consuming FastAPI.

For 2–3 weeks, I’d go with **Streamlit + FastAPI or even pure Streamlit** for speed.

Concrete 2–3 week plan (for 2 people)

Assume ~2.5 weeks total. I’ll call you **Dev A** and your friend **Dev B**, but you can swap.

Days 1–2: Scope + data + skeleton

Both

- Decide on:

    - 3–5 companies.
    - 2–3 document types (e.g. last 2 annual reports + last 2 earnings calls).
- Set up:
    - GitHub repo.
    - Basic project structure (backend/, frontend/, data_pipeline/).
    - requirements.txt or pyproject.toml.

Dev A

- Write a small script:
    - download_reports.py (even if you manually save PDFs, script moves them into project structure).
    - Convert PDF to text (e.g. pdfplumber, pypdf).

Dev B

- Create skeleton Streamlit app / basic frontend with:
    - Dropdown for companies.
    - Text area for question.
    - "Ask" button.
    - Placeholder answer box.

Days 3–6: RAG core pipeline

Dev A – RAG backend

- Implement chunking:
    - Functions for splitting text into 500–800 token chunks with overlap.
    - Store chunks as JSONL or in a small DB.
- Implement embedding + vector store:
    - Choose embedding model.
    - Build index (Chroma/FAISS/etc.).
    - Write utility: retrieve_chunks(company, question, top_k=5).

Dev B – LLM + prompt design

- Integrate LLM API (OpenAI / local model).
- Design prompts:
    - One for single-company Q&A.
    - (Optional) variant for cross-company comparison.
- Implement answer_question(question, companies):
    - Call retrieval,
    - Build prompt with context,
    - Call LLM,
    - Return structured answer + citations.

End of Day 6 target:
You can run a Python script like:

python demo_cli.py "What are the main risks for Company X in the latest report?"

and get a semi-decent answer with citations in the terminal.

Days 7–10: UI, evaluation, polishing

Dev B – UX

- Connect UI to backend function:
    - When user clicks “Ask”, call your Python function (or API) and display:
        - Answer text.
        - List of source snippets (expanders with metadata).
- Make it nice but simple:
    - Show company, document type, section, page.
    - Maybe highlight numerical values in bold.

Dev A – Evaluation + logging

- Create a small eval_questions.json:
    - ~20 questions per company, like:
        - “What was total revenue in 2024?”
        - “How did management describe macroeconomic risks?”

    - Write your “ideal” short answer for each.
- Implement evaluation script:
    - For each Q, call your RAG pipeline.
    - Save: question, ref_answer, model_answer, retrieved_chunks.
- Build Streamlit “Evaluation” page:
    - Show each Q + model answer + context.
    - Dropdown: Correct / Partially correct / Incorrect.
    - Display aggregate metrics (counts, percentages).

Days 11–14: Hardening + CV polish

Both

- Refactor into clean modules: ingestion.py, retrieval.py, rag_pipeline.py, app.py.
- Add **basic tests**:
    - Chunking doesn’t lose text.
    - Retrieval returns chunks only for selected company.
- Write a clear README.md:
    - Project description.
    - Tech stack.
    - How to run locally (commands).
    - Some example questions + screenshots (optional).
- Add **Dockerfile** (nice for employers):
    - Simple: base Python image → install deps → run app.

Stretch ideas (if time remains)

- Add **company comparison** mode:
    - “Compare leverage between Company A and B.”
- Add **simple financial metric extraction**:
    - Use regex/LLM to parse “Total Revenue”, “Operating Income”, etc. and show a small table.

How to describe it on your CV / LinkedIn

CV bullets (example)

Financial Filings RAG Assistant (personal project) — 2025
• Built an LLM-powered question-answering system over annual reports and earnings call transcripts for multiple public companies using retrieval-augmented generation (RAG).
• Implemented ingestion pipeline (PDF → text → chunking → embeddings), vector search (Chroma/FAISS), and a FastAPI/Streamlit interface that returns cited answers and source snippets.
• Designed a small evaluation suite with labeled questions to measure answer correctness and hallucination rate, iterating on chunking and prompt strategies to improve performance.

LinkedIn "About" or "Projects" snippet

Developed an LLM-based assistant for financial analysis that answers investor-style questions using RAG over annual reports and earnings call transcripts. Implemented end-to-end pipeline (data ingestion, embeddings, vector search, prompt engineering, evaluation) with a web UI for interactive exploration and manual grading of answer quality.

This directly checks the boxes for:

- "LLM experience"
- "RAG"
- "Finance domain"
- "End-to-end ML/AI product skills"

If you'd like, I can next:

- Suggest a **concrete tech stack** (exact libraries + directory structure), or
- Help you design the **prompt templates** and metadata schema so you can start coding right away.