Balance Sheet Analyst

Project Report

Author: Prasunn Dubey

Date: November 2025

The **Balance Sheet Analyst** is an AI-driven platform to analyze corporate balance sheets and income statements using NLP, RAG and multi-LLM reasoning.

Department of Computer Science
AI and Machine Learning Specialization
Birla Institute of Technology

Contents

Chapter 1

Executive Summary

What it is. An AI-powered analyst that interprets balance sheets and annual reports, answers natural-language questions, and visualizes key financial metrics.

Why it matters.

- Reduces manual effort in sifting through 100+ page annual reports.
- Brings explainable, on-demand insights to analysts and leadership.
- Scales across companies with consistent ingestion and retrieval.

How it works.

- Modular **FastAPI** backend + **Streamlit** frontend.
- **RAG** retrieves relevant context from the annual report PDF (FAISS).
- Multi-LLM integration (Groq / OpenAI) generates concise, data-grounded answers.
- Plotly dashboards show trends across assets, liabilities, revenue, profit.

Outcomes (Reliance 2023–24).

- Total Assets: 1,755,986 crore; Liabilities: 830,198 crore; Equity: 925,788 crore.
- Revenue: 914,472 crore; Profit: 79,020 crore.
- Equity ratio 52.7%; strong liquidity; clear profitability drivers identified.

Chapter 2

Key Features

- **Interactive Analyst Chatbot (LLM):** Ask any financial or qualitative question; responses cite retrieved evidence where applicable.
- **RAG-based Contextual Understanding:** Pulls sentences/paragraphs from the 100+ page PDF and fuses them with structured figures.
- **Dynamic Financial Dashboard:** Plotly charts for assets, liabilities, revenue and profit; year-over-year comparisons and KPIs.
- **Secure Authentication:** Role-based access (`analyst@company.com`, `ceo@company.com`) with token-based session.
- **Document Parsing & Embedding Search:** PDF ingestion, chunking and semantic search via FAISS + SentenceTransformers.
- **Multi-modal Context Fusion:** Combines JSON financials with unstructured report text for richer, grounded answers.
- **Extensible Architecture:** Clean separation of concerns; easy to deploy on Render/Railway/AWS EC2 with Streamlit Cloud.

Chapter 3

System Architecture

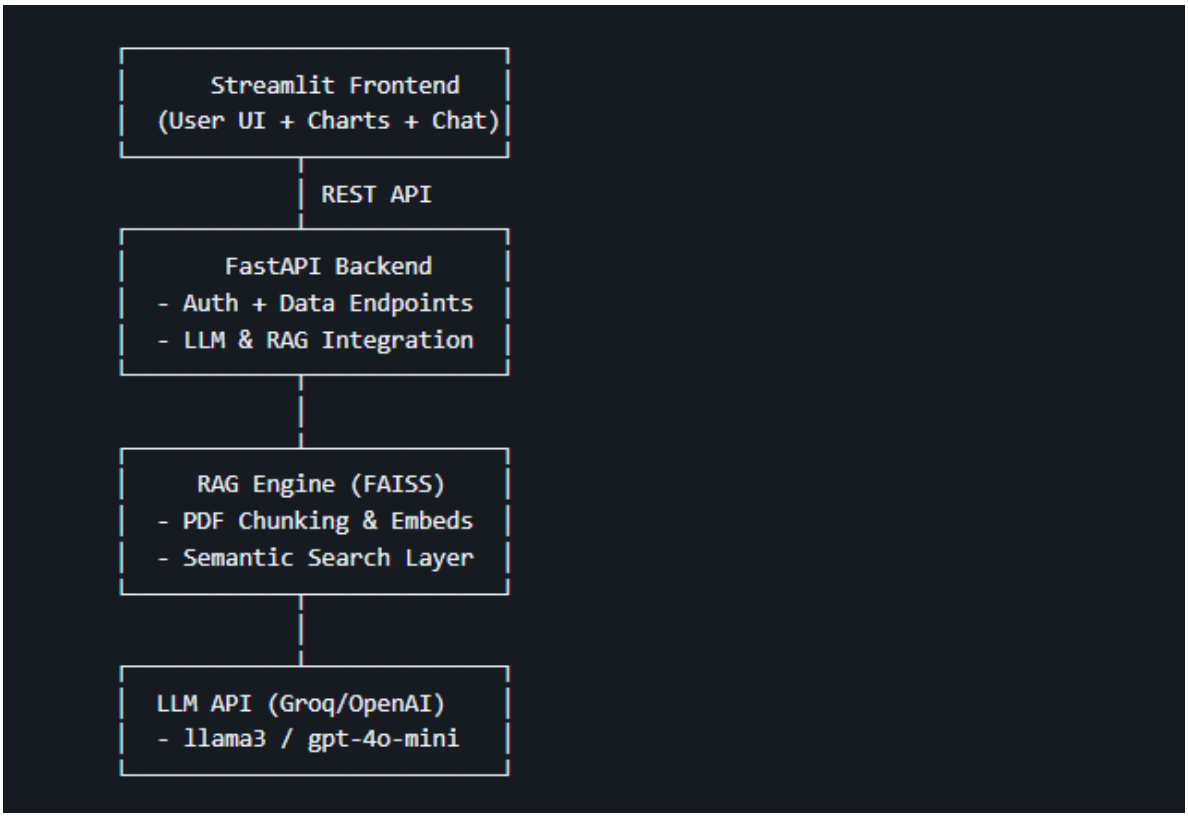

Figure 3.1: High-level architecture: Streamlit UI → FastAPI → RAG (FAISS) → LLM APIs.

3.1 Data Flow

1. User queries and dashboard actions originate in Streamlit.
2. FastAPI authenticates requests, fetches structured figures, and orchestrates RAG.
3. FAISS retrieves top-k chunks; context is fused with financial JSON.
4. The LLM generates grounded answers and the frontend renders visualizations.

Chapter 4

Repository Structure & Tech Stack

4.1 Repository Tree

```
balance-sheet-analyst/
          backend/
                app/
                      main.py
                      model.py
                      rag_utils.py
                      settings.py
                      reliance_consolidated.pdf
                requirements.txt
                ...
          frontend/
                streamlit_app.py
          README.md
          requirements.txt
```

4.2 Tech Stack

Layer	Technology	Purpose
Frontend	Streamlit, Plotly Express	Interactive dashboard and analyst chatbot interface
Backend	FastAPI	REST API with auth, data endpoints, and LLM orchestration
LLM Engine	Groq API / OpenAI GPT	Natural language reasoning and response generation
RAG Engine	FAISS, SentenceTransformers	Embedding-based similarity search over the PDF
Storage	JSON, FAISS index	Financial figures and vectorized chunks
Visualization	Plotly Express	Time series and KPI charts
Language	Python 3.12	Core development language

Chapter 5

Implementation Details

5.1 Backend (FastAPI)

Responsibilities: authentication, data serving, RAG retrieval, prompt assembly, and LLM invocation.

Listing 5.1: Login endpoint with simple role-based token

```
class LoginRequest(BaseModel):
    email: str
    password: str

@app.post("/login")
def login(req: LoginRequest):
    user = USERS.get(req.email)
    if not user or user["password"] != req.password:
        raise HTTPException(status_code=401, detail="Invalid credentials")
    return {"name": user.get("name","Analyst"), "role": user["role"],
            "token": req.email, "companies": user.get("companies",["Reliance"])}
```

Listing 5.2: RAG-fused analysis endpoint (from your main.py)

```
class AnalyzeRequest(BaseModel):
    question: str

@app.post("/analyze")
def analyze(req: AnalyzeRequest, token: str):
    user = USERS.get(token)
    if not user:
        raise HTTPException(status_code=401, detail="Unauthorized")

    structured_context = DATA.get("figures_crore", {})

    retrieved_text = ""
    try:
        retrieved_chunks = query_rag(req.question, top_k=4)
        if retrieved_chunks:
            retrieved_text = "\n\n".join(retrieved_chunks)
    except Exception as e:
        print("[WARN] RAG retrieval failed:", e)

    prompt = f"""
You are a senior financial analyst AI.
Answer using BOTH the structured data and retrieved report excerpts.

Structured Financials (    crore):
{json.dumps(structured_context, indent=2)}

Retrieved Excerpts:
{retrieved_text}
```

```
Question: {req.question}
Guidelines: stay grounded in provided data; quantify when possible; state if data
    is missing.
"""

    # ... Groq call (as in your snippet) and JSON response ...
```

5.2 Frontend (Streamlit)

- Login form and session state for token handling.
- Dashboard with KPIs and Plotly charts (assets, liabilities, revenue, profit).
- Chat panel wired to `/analyze` for question answering.

Listing 5.3: Example call from Streamlit to analyze endpoint

```
def call_analyze(token: str, question: str):
    return api_post("/analyze", json={"question": question}, params={"token":
        token})
```

5.3 RAG Pipeline

Ingestion & Indexing.

- Parse the annual report PDF, extract text, split via `RecursiveCharacterTextSplitter` (e.g., 1000 chars, 200 overlap).
- Embed with SentenceTransformers; persist FAISS index to disk.

Retrieval & Fusion.

- Top-k retrieval per query; concatenate chunks (with source markers).
- Prompt template fuses JSON figures and retrieved text for grounded answers.

Chapter 6

Usage, API & Security

6.1 How to Use

Login. Use `analyst@company.com / analyst123` (demo) to obtain a token.

Explore.

1. Open the dashboard to view KPIs and charts.
2. Ask questions in natural language (e.g., "Compare profit vs revenue YoY").
3. Download CSVs and adjust chart filters interactively.

6.2 API Endpoints

Method Description / Params	Path	Auth
POST `email, password` → token, role, companies	/login	–
GET `token, [company]` → structured figures	/balance-sheet	Token
POST `question` → grounded answer + retrieved chunks	/analyze	Token

6.3 Security & Compliance

- **AuthZ/AuthN:** Role-based access; token identifies user and allowed companies.
- **CORS:** Restricted origins in production; HTTPS enforced by hosting provider.
- **Secrets:** API keys in `.env`; never committed to VCS.
- **Data Handling:** Only necessary fields are logged; PII avoided.

Chapter 7

Deployment, Performance & Findings

7.1 Deployment

Targets. Backend: Render/Railway/AWS EC2; Frontend: Streamlit Cloud. **Pipeline.** Build & push image / deploy from GitHub; run migrations and warm up RAG index.

- **Backend on Render/Railway/EC2:** Auto-deploy from main branch, managed SSL, horizontal scaling.
- **Frontend on Streamlit Cloud:** One-click deploy; set environment variables for API base URL.
- **CORS:** Allowlist deployed Streamlit domain only.
- **Observability:** Basic logs + request latency, error rate, tokens/sec.

7.2 Performance & Caching

- **Retrieval speed:** FAISS index enables sub-second semantic search on local PDFs.
- **Cold-start:** Index ensured at startup; reindex on file change.
- **Response time:** Groq/OpenAI latency dominates; keep prompts concise and deterministic (low temperature).
- **Caching:** Memoize embeddings and frequent queries; CDN for static assets.

7.3 Findings & Insights

- Total Assets: 1,755,986 crore Liabilities: 830,198 crore Equity: 925,788 crore
- Revenue: 914,472 crore Profit: 79,020 crore

Observation: equity ratio $\approx 52.7\%$, healthy liquidity; profitability supported by diversified business lines.

Chapter 8

Future Improvements and Personal Highlights

Future Enhancements

- Time-series forecasting and sensitivity analysis with domain prompts.
- Multi-company portfolio view and peer benchmarking.
- Real-time market data (Yahoo Finance) and on-chart ratio analytics.
- Auto-ingestion across multiple years; deduplication and section-aware splitting.
- Better caching, hybrid reranking, and light domain-tuned models.

Personal Highlights

- Built a production-style FastAPI + Streamlit stack with RAG and multi-LLM.
- Solved CORS, SDK migration, and performance bottlenecks under constraints.
- Designed grounded prompts and retrieval fusion for explainable answers.
- Sharpened debugging discipline and deployment hygiene.

References

- FastAPI — https://fastapi.tiangolo.com/
- Streamlit — https://streamlit.io/
- LangChain — https://python.langchain.com/
- OpenAI API — https://platform.openai.com/docs/
- FAISS — https://faiss.ai/

Appendix: Configuration

Create a `.env` with:

```
GROQ_API_KEY=...
OPENAI_API_KEY=...
RAG_PDF_PATH=backend/app/reliance_consolidated.pdf
RAG_MODEL=llama-3.1-8b-instant
```