Internship Assignment — Linkedin Analytics Backend (FastAPI + PostgreSQL)

Goal

Build a backend system (FastAPI + PostgreSQL) that powers a simplified **linkedin analytics platform**. The focus is on **database design, API handling, and scheduling logic**.

Requirements

1. Users & Roles

- Implement **JWT authentication**.
- At least 2 roles:
    - **Admin** → can manage all posts & analytics.
    - **User** → can manage only their own posts & analytics.

2. Posts (Create your own dummy database)

- Implement **CRUD APIs** for posts.
- Posts should be retrievable with **filters** (by user, by time range, etc.).

3. Post Analytics

- Store and calculate multiple reaction types:
  `like, praise, empathy, interest, appreciation`.
- Calculate **engagement metrics**: total reactions, total engagement, total impressions, total shares, total comments.
- Provide APIs to:
    - Post-wise Analytics Graphs
    - Fetch analytics (e.g., top 5 most engaging posts)

4. Internal Post Scheduling

- A user should be able to **schedule a post** by providing:
    - `date`
    - `hour`
    - `minute`
- The post should go live at **any second within that minute**.
- The system should be optimized to handle **many users scheduling posts at the same time**.
- When the post is due:
    - Call an **empty function** (placeholder) that simulates the LinkedIn API post request. (No need to integrate linkedin api's)
    - Change the post status from `scheduled` → `published`.
- **No need to integrate LinkedIn API** — just simulate it.
- Handle edge cases (e.g., scheduling in the past).

5. Database & Queries

- Design **tables** for users, posts, and analytics.
- Use **indexes** for faster retrieval (especially time-based queries).
- Write **optimized queries** (joins, aggregates, group by).

6. API Handling

- Role-based authorization checks in endpoints.
- Input validation with Pydantic.
- Proper error handling with HTTP status codes.

Deliverables

- FastAPI project with a clear structure
- PostgreSQL schema & migration setup
- Postman collection with sample data

Notes

- Focus on **database design + API correctness**, not frontend.
- Keep it simple but **production-like**.
- Bonus if you can deploy the api’s on a server.