Senior Software Development Engineer - Technical Assignment

Objective

Build a scalable Task Management System with Dynamic Rule-Based Task Assignment. The goal of this assignment is to evaluate your architectural thinking, database design, performance optimisation, background processing, and clean coding practices.

Suggested Tech Stack

Backend: Python (Django or FastAPI)
Database: PostgreSQL
Caching & Queues: Redis
Background Processing: Celery / RQ / Worker
Authentication: JWT + Refresh Tokens
Frontend: React
Infra: Docker & Docker Compose

Core Features

1. Authentication & Authorization
   - User signup and login
   - Roles: Admin, Manager, User

2. Task Management
   - CRUD operations on tasks
   - Status: Todo → In Progress → Done
   - Due dates and priority

3. Dynamic Rule-Based Task Assignment Engine (Important)
   - Tasks are NOT manually assigned to users.
   - Each task defines dynamic assignment rules.

User Profile Attributes

Each user has:
- Department (Finance, HR, IT, Operations)
- Experience in years

- Location
- Current number of assigned tasks

User Stories

Story 1 - Admin Creates Task with Rules
Admin creates a task and defines rules such as:
- Department = Finance
- Experience >= 4 years
- Active Tasks < 5

The system automatically computes eligible users in the background and assigns the task to the eligible user. What will happen if there are multiple eligible users? What will happen if there are no eligible users?

Story 2 – User Views Eligible Tasks
User should see only tasks they are eligible and assigned for via:
GET /my-eligible-tasks
This API must be highly optimised.

Story 3 – User Data Changes
If user attributes change, eligibility must be recomputed automatically.

Story 4 – Admin Updates Rules
If task rules change, the system must recompute eligible users efficiently.

Required APIs

POST /tasks/ → Create task with rules
GET /tasks/{id}/eligible-users → Highly optimised
GET /my-eligible-tasks
POST /tasks/recompute-eligibility

Performance Expectations

System should be designed assuming:
- 100k users
- 1M tasks
- APIs must respond under 200ms using caching, indexing, and background processing.

Deliverables

1. Public GitHub repository
2. Docker setup
3. DB migrations
4. README explaining:
   - Architecture decisions
   - Indexing strategy
   - Caching strategy
   - Rule engine design
   - Recompute strategy
5. Seed data
6. API documentation

Evaluation Criteria

- Architecture quality
- Database design & indexing
- Performance optimisation
- Clean code and structure
- Rule engine implementation
- Background processing design