# Objective

Develop a **high-performance task management system** where users can create tasks, assign them to different users, and track their progress. The system should also process large data sets asynchronously using **multi-threading** and **Celery for background processing**.

---

# Functional Requirements

1. **User Registration & Authentication**
    - Implement JWT-based authentication using **Django REST Framework (DRF)**.
    - Users can **register, log in, and manage their profiles**.
2. **Task Management**
    - Users can **create tasks** with details like **title, description, priority, due date, and status**.
    - Each task should be **assigned to a user**.
    - Users should be able to **view, update, and delete their tasks**.
    - Implement **pagination and filtering** (e.g., by priority, status, or due date).
3. **Multi-Threaded Report Generation**
    - A **report generator API** should process **100,000+ tasks** and return insights:
        - Number of completed tasks
        - Number of pending tasks
        - Tasks categorized by priority
    - This API should use **multi-threading** to improve performance.
4. **Asynchronous Notifications (Using Celery & Redis)**
    - Implement a **Celery task** that runs in the background to **send email notifications** when:
        - A task is assigned to a user.
        - A task's deadline is within 24 hours.
    - Use **Redis** as the Celery broker.
5. **Caching (Using Redis)**
    - Cache the task list API response for **fast retrieval**.
    - If a user requests the task list **multiple times within a minute**, serve data from Redis instead of querying the database.
6. **WebSocket for Real-Time Updates**
    - Implement **Django Channels** to send real-time updates when a task is updated or assigned.

---

# Non-Functional Requirements (NFRs)

1. **Performance Optimization**
    - Ensure API response times are minimal by using **threading, caching, and optimized queries**.
2. **Fault Tolerance**
    - Implement **exception handling** and retry mechanisms for Celery tasks.
3. **Scalability**
    - The architecture should allow for future expansions, such as adding more task statuses, notifications, or integrations.

---

# Technical Requirements

1. **Django & Django REST Framework** for API development.
2. **PostgreSQL** as the database.
3. **Celery & Redis** for background tasks.
4. **Django Channels & WebSockets** for real-time notifications.
5. **Multi-threading** for handling large dataset processing.
6. **Caching** using Redis.
7. **Docker** for containerization.

---

# Bonus (Optional Enhancements)

1. **Rate Limiting** - Prevent abuse by limiting API requests using Django's `throttling`.
2. **Task Export** - Implement an endpoint to **export tasks as CSV** in an asynchronous manner.
3. **GraphQL Support** - Implement a GraphQL endpoint using Django Graphene.

---

# Deliverables

1. **Source Code** (GitHub repository).
2. **Postman Collection** for API testing.
3. **Dockerfile & docker-compose.yml** for easy deployment.
4. **README.md** with setup instructions.

---

Please share the details on aditya@tratoli.com in 48 hours of getting this assignment.