📘 Assignment: API Monitoring & Observability Platform

Tech Stack

- **Backend:** Spring Boot (Kotlin)
- **Frontend:** Next.js (Node.js)
- **Database:** Two separate MongoDB databases
- **Auth:** JWT

🎯 Objective

Build a complete platform that tracks API requests across multiple microservices, stores their performance metrics, analyzes issues, and displays them on a dashboard.

🧩 System Overview

Your solution must include:

1. **API Tracking Client**
2. **Central Collector Service**
3. **Next.js Web Dashboard**

--

🚀 1. API Tracking Client (Spring Boot + Kotlin)

A reusable library/interceptor that can be added to any Spring Boot service.

It must track:

- API endpoint
- Request method
- Request size
- Response size
- Status code
- Timestamp
- Latency
- Service name

It must send logs to the collector using REST or gRPC.

⚡ Rate Limiter Requirement

Implement a **per-service rate limiter** inside the tracking layer.

- Default allowed rate: **100 requests/second**
- If the limit is exceeded:
    - The request should still continue normally
    - Log an event `"rate-limit-hit"` to the collector

Rate limits must be configurable through application.yaml.

Example:

```
monitoring:
  rateLimit:
    service: orders
    limit: 120
```

🏢 2. Central Collector Service (Spring Boot + Kotlin)

This service receives logs from multiple microservices and stores + analyzes them.

🔥 Mandatory: Dual MongoDB Connections

Your collector must connect to **two separate MongoDB instances**:

1. Logs Database (Primary MongoDB)

Stores:

- Raw API logs
- Rate limit hit logs

2. Metadata Database (Secondary MongoDB)

Stores:

- User accounts
- Rate limiter config overrides

- Slow/broken API incidents
- Resolved status audit trail
- Alerts history

You must implement:

- Two `MongoTemplate` beans
- Two repositories
- Two MongoTransactionManagers
- Separate schemas/collections

🔄 Concurrency Requirement

Multiple developers may mark an issue as resolved simultaneously.

You must ensure safe writes using:

- **Optimistic locking** (preferred) OR
- Mongo transactions OR
- Atomic operators

📈 3. Dashboard (Next.js)

Build a frontend dashboard that interacts with the collector service.

The dashboard must show:

A. API Request Explorer

- List all API logs
- Filters:
    - Service
    - Endpoint
    - Date range
    - Status
    - Slow APIs (> 500ms)
    - Broken APIs (5xx)
    - Rate-limit hits

B. Dashboard Widgets

- Slow API count
- Broken API count
- Rate-limit violations
- Avg latency per endpoint
- Top 5 slow endpoints
- Error-rate graph

C. Issue Management

- Developers can mark a slow/broken endpoint as "Resolved"
- This must update the metadata DB safely

🚨 Alerting Rules

The collector must detect and log alerts when:

1. API latency > **500ms**
2. Status code is **5xx**
3. Rate limit exceeded

Alerts must appear on the dashboard.

📌 Core Requirements Checklist

Backend

- Spring Boot + Kotlin
- API tracking interceptor
- Rate limiter
- Two MongoDB connections
- Concurrency safety
- JWT Auth

- Alert generation
- REST APIs for frontend

Frontend (Next.js)

- Login page
- Logs table with filters
- Dashboard widgets
- Alert viewer
- API issue resolution UI

⚙️ Non-functional Requirements

- Handle **50 concurrent log writes** without failure
- README must describe:
    - Architecture
    - DB schemas
    - Decisions taken
    - How dual MongoDB setup works
    - How rate limiter works
- Code must be reasonably modular and clean