**Scenario-Based Response Document**

## 1. Important Metrics and Their Significance

To monitor and improve a high-response-time/error-rate REST API endpoint, the following metrics are crucial:

**a) Latency Metrics**

- **P99, P95 and P50 Response Time:** Helps identify latencies affecting the users.
- **DB Query Time:** Measures the delays in database calls.

**b) Error Metrics**

- **HTTP Error Rate (4xx, 5xx codes):** Helps track user and server-side failures.
- **Dependent API Failure Rate:** Monitors external API failures affecting the endpoint.

**c) Throughput Metrics**

- **Requests per Second:** Helps in capacity planning and identifying bottlenecks.

**d) Resource Utilization**

- **CPU & Memory Usage:** Identifies resource caused bottlenecks.

## 2. Logging: What to Keep, Remove and Add?

**What Logs to Keep:**

Request/Response Logs, Database Query Logs, Error Logs with Stack Traces.

**What Logs to Remove:**

Verbose Debug Logs, Unstructured Logs.

**What Logs to Add:**

User Context Information, Performance Metrics in Logs.

## 3. Sample Dashboard

**Image Link -** Sample Grafana Dashboard

**Dashboard Sections:**

- **API Performance Overview:** P99, P95, Avg. Response Time.
- **Error Analysis:** Error Distribution (4xx, 5xx).
- **Request Trends:** Request Per Second, Active Connections, Peak Traffic Times.
- **Dependency Monitoring:** External API and DB calls failure rates.
- **Resource Health:** CPU, Memory and Network Performance.

## 4. Improvements to Address High Response Time and Errors

### a) Code & Query Optimizations

- Optimize slow database queries (indexing, query restructuring).
- Reduce unnecessary external API calls (caching, bulk requests).

### b) Scaling & Load Balancing

- Use auto-scaling policies based on CPU and RPS.
- Implement rate limiting for API stability.

### c) Observability & Monitoring Enhancements

- Implement distributed tracing (OpenTelemetry + Tempo) to track request flow.
- Set up error detection alerts (Grafana/Prometheus alerts on high latency/errors).

## 5. Tracking the Impact of Improvements

To measure how much these optimizations helped:

- **Compare Before & After Metrics:** Look at latency, error rates, and RPS improvements.
- **Monitor Incident Frequency:** Reduction in escalations & downtime.
- **Voluntary Team Feedback:** Survey developers on debugging experience improvements.

**Follow-Up Questions**

## 1. Reducing Repetitive Work for Expanding Dashboards

- Use templated dashboards in Grafana to dynamically add new API endpoints.
- Automate log parsing and metric extraction using OpenTelemetry.
- Standardize logging format for easy ingestion across services.

## 2. Potential Issues after Fixing API Performance

- More logs, traces, and metrics can lead to higher storage and processing costs.
- Too many alerts can overwhelm teams if not fine-tuned.
- Fixing one issue might reveal other weak points.

## 3. Preventing Recurrence of These Issues

- Ensure new releases don’t reintroduce old issues.
- Monitor API performance during deployments.
- Remove outdated logs, refine alert thresholds.
- Follow best practices to avoid similar issues in the future.