NPONTU TECHNOLOGIES INTERVIEW ASSIGNMENT

Engineering Department - App/Ops Officer Intern

Name of Interviewee: Mark Drah

Date of Assignment: 4 September 2025

Question 1

Analyze a set of application logs (you can provide sample logs with errors), identify the root cause of a performance issue, and create a runbook for troubleshooting similar problems in the future.

Sample Application Logs (with errors)

```
2025-09-04 10:12:15 [INFO] Server started on port 3000
2025-09-04 10:12:45 [INFO] User login request from IP 192.168.1.20
2025-09-04 10:13:01 [ERROR] Database connection timeout after 5000ms
2025-09-04 10:13:05 [WARN] Retrying database connection...
2025-09-04 10:13:10 [ERROR] Database connection failed: Too many connections
2025-09-04 10:13:22 [INFO] Memory usage at 85% capacity
2025-09-04 10:13:35 [ERROR] API request /api/orders failed: Response time exceeded
3000ms
2025-09-04 10:13:50 [WARN] High CPU usage detected: 92%
2025-09-04 10:14:02 [ERROR] Application crashed due to unhandled exception:
OutOfMemoryError
```

Root Cause Analysis

From the logs, the following performance issues were identified:

- Database Connection Issues: Multiple timeouts and 'too many connections' errors.

- High Memory & CPU Usage: Application consumed excessive system resources.

- Slow API Responses: Likely caused by database slowness and resource exhaustion.

- Crash: Application terminated due to unhandled OutOfMemoryError.

Root Cause: Database connection saturation combined with memory leaks caused cascading failures, leading to slow performance and eventual system crash.

Runbook: Troubleshooting Performance Issues

Step 1: Check Database Connections

- Review database connection pool settings.
- Verify maximum allowed connections in DB configuration.
- Run `SHOW PROCESSLIST;` (MySQL) or `pg_stat_activity` (Postgres) to see active sessions.
- Kill idle or stuck connections if necessary.

Step 2: Monitor System Resources

- Use `top` / `htop` to check CPU and memory usage.
- Restart services if memory leak is detected.
- Increase server memory or optimize application code.

Step 3: Investigate API Response Times

- Check which endpoints are failing.
- Run profiling tools or enable request logging.
- Optimize queries and add indexes if DB queries are slow.

Step 4: Restart and Stabilize Application

- Clear cache/temp files.
- Restart the app service (`systemctl restart app.service`).
- Validate by sending test API requests.

Step 5: Prevent Recurrence

- Implement database connection pooling.
- Add monitoring tools (Prometheus, Grafana, NewRelic, or Datadog).
- Set up alerts for CPU > 80%, Memory > 75%, or DB errors.
- Fix unhandled exceptions in code to avoid future crashes.