1.1. AWS Design Question

Architecture Overview:

Compute: Use Amazon ECS (Fargate) for containerized Dropwizard service.

Database: Use Amazon RDS (Aurora MySQL) for high availability and automatic backups.

Storage: Store uploaded session materials in Amazon S3.

Security:

Use IAM Roles for ECS task permissions.

Use AWS API Gateway (if going serverless) with WAF for request filtering.

Restrict access via Security Groups and NACLs.

Scaling:

Use Auto Scaling on ECS tasks based on CPU/memory.

Use Aurora for built-in read replicas.

Use ALB (Application Load Balancer) for traffic distribution.

2. Debugging & Performance Optimization

Steps to troubleshoot /sessions slowness:

Profile the query: Use EXPLAIN on SQL to check if table scans are happening. Add indexes if needed.

Check DB connection pool: Ensure Dropwizard's HikariCP is configured properly.

Pagination: Implement pagination instead of loading all sessions at once.

Monitor metrics: Use Dropwizard metrics or AWS CloudWatch to track response time, DB latency.

Thread blocking: Investigate any synchronous I/O or long operations in the service layer.

3. Security Scenario – DDoS Mitigation

Infrastructure-level:

Use AWS WAF with rate-limiting and IP blocking.

Use CloudFront (CDN) to cache and absorb load.

Enable AWS Shield for automatic DDoS protection.

Code-level:

Validate API key early in the request lifecycle.

Return minimal error details to avoid info disclosure.

4. **Resource Leak**: No try-with-resources it may leave DB connections open.

-Use
try (Connection conn = ...; Statement stmt = ...; ResultSet rs = ...) {}

Hardcoded DB credentials → Security risk
- Externalize to config file (config.yml).

Using raw Statement → Vulnerable to SQL injection

- Use PreparedStatement or JDBI3 DAO for safety and readability.