STDISCM P4

Distributed Fault Tolerance

Dayrit, Jason Rafael S11
Escober, Matthew David S14
Lababidi, Mezen Carlos S11

System Overview & Architecture

1. Microservices:
    a. Auth Service: JWT, OAuth2, Redis for token sharing
    b. Course Service: REST CRUD, optimistic locking, read replicas
    c. Enrollment Service: Distributed locking (Redis), transactions, idempotency
    d. Grading Service: Concurrent grade handling and event sourcing
    e. API Gateway: Routes requests; implements circuit breakers

1. Infrastructure:
    a. Dockerized services (using Docker Compose)
    b. PostgreSQL with replication and HikariCP
    c. Redis (for caching and locking) and RabbitMQ (for messaging)

Key Implementation Steps

1. Phase 1: Planning & Architecture:
    a. Define service boundaries and failure domains
    b. Design resilient database schemas and communication patterns
2. Phase 2: Infrastructure Setup:
    a. Dockerfiles/Compose for local development
    b. PostgreSQL containers (with replication)
    c. Redis cluster and RabbitMQ configuration

1. Phase 3: Backend Development:
    a. Implement Auth, Course, Enrollment, & Grading services (Spring Boot)
    b. Configure Spring Cloud Gateway for routing/resilience
    c. Integrate JWT-based security and role-based access control

Fault Tolerance Mechanisms

1. Database Resilience:
    a. PostgreSQL replication, PgPool-II, HikariCP connection pooling
2. Service Resilience:
    a. Circuit breakers (Resilience4j) and retry logic
    b. Bulkhead pattern to isolate failures
    c. Distributed locking (via Redis) for concurrent enrollment

1. Transaction & Consistency:
    a. Spring's '@Transactional' for atomic operations
    b. Idempotency via request IDs to avoid duplicate enrollments
2. Monitoring & Load Balancing:
    a. Actuator, Prometheus, Grafana for health monitoring
    b. HAProxy/Nginx for load balancing and sticky sessions