**Implementing Caching (Spring Boot + Redis or in-memory)**

A. **Add caching to account balance method**

@Service

public class AccountService {

@Cacheable(value = "accountBalance", key = "#accountId")

public Double getAccountBalance(String accountId) {

simulateLatency();

return database.findBalanceByAccountId(accountId);

}

private void simulateLatency() {

try { Thread.sleep(2000); } catch (InterruptedException e) { }

}

}

B. **Enable caching in main config**

@SpringBootApplication

@EnableCaching

public class BankApp {

public static void main(String[] args) {

SpringApplication.run(BankApp.class, args);

}

}

C. **Add in-memory or Redis config in application.yml**

spring:

cache:

type: simple # or redis

**Rate Limiting per API Role/User using Bucket4j**

@Component

public class RateLimiterFilter extends OncePerRequestFilter {

private final Map<String, Bucket> buckets = new ConcurrentHashMap<>();

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

throws ServletException, IOException {

String apiKey = request.getHeader("API-Key");

if (apiKey == null) {

response.setStatus(401);

return;

}

Bucket bucket = buckets.computeIfAbsent(apiKey, k -> Bucket4j.builder()

.addLimit(Bandwidth.simple(10, Duration.ofMinutes(1))) // 10 requests per minute

.build());

if (bucket.tryConsume(1)) {

chain.doFilter(request, response);

} else {

response.setStatus(429); // Too Many Requests

}

}

}

**Circuit Breaker (Using Resilience4J)**

<dependency>

<groupId>io.github.resilience4j</groupId>

<artifactId>resilience4j-spring-boot2</artifactId>

<version>1.7.1</version>

</dependency>

Configure in application.yml

resilience4j.circuitbreaker:

instances:

accountService:

registerHealthIndicator: true

slidingWindowSize: 10

minimumNumberOfCalls: 5

failureRateThreshold: 50

waitDurationInOpenState: 10s

Wrap the method:

@CircuitBreaker(name = "accountService", fallbackMethod = "fallbackBalance")

public Double getAccountBalance(String accountId) {

return database.findBalanceByAccountId(accountId);

}

public Double fallbackBalance(String accountId, Throwable ex) {

return -1.0; // or cached/default value

}

**Monitoring (Using Actuator + Prometheus + Grafana)**

**Add Actuator in pom.xml:**

xml

Copy code

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

**Enable in application.yml:**

yaml

Copy code

management:

endpoints:

web:

exposure:

include: "\*"