BANKING SYSTEM PROJECT - DAY 5: AUDIT LOGS IMPLEMENTATION

==========================================================

OVERVIEW

--------

Day 5 focuses on implementing comprehensive audit logging capabilities for the banking system. This implementation ensures complete traceability of all system activities, compliance with regulatory requirements, and provides detailed audit trails for security and operational purposes.

AUDIT LOGGING ARCHITECTURE

---------------------------

1. DUAL STORAGE STRATEGY

- MongoDB: Primary audit log storage for detailed records

- DynamoDB: Event log storage for high-volume, time-series data

- Real-time synchronization between both systems

- Redundancy for data protection and compliance

2. AUDIT LOG COMPONENTS

- AuditService: Core audit logging service

- AuditLogRepository: MongoDB data access layer

- DynamoDBService: DynamoDB integration service

- AuditLogController: REST API for audit log access

AUDIT LOG DATA MODEL

--------------------

1. AUDITLOG ENTITY STRUCTURE

public class AuditLog {

private String id; // MongoDB ObjectId

private String actionId; // Unique action identifier

private String userId; // User who performed the action

private String actionType; // Type of action (DEPOSIT, WITHDRAWAL, etc.)

private String entityType; // Entity type (CUSTOMER, ACCOUNT, TRANSACTION)

private String entityId; // ID of the affected entity

private String actionDetails; // Detailed description of the action

private LocalDateTime timestamp; // When the action occurred

private String ipAddress; // IP address of the user

private String userAgent; // User agent/browser information

private AuditStatus status; // SUCCESS or FAILURE

private String errorMessage; // Error details if action failed

private LocalDateTime createdDate; // Record creation timestamp

}

2. AUDIT STATUS ENUMERATION

public enum AuditStatus {

SUCCESS, // Action completed successfully

FAILURE // Action failed with error

}

AUDIT SERVICE IMPLEMENTATION

----------------------------

1. CORE AUDIT SERVICE CLASS

@Service

public class AuditService {

private final AuditLogRepository auditLogRepository;

private final DynamoDBService dynamoDBService;

private final ObjectMapper objectMapper;

@Autowired

public AuditService(AuditLogRepository auditLogRepository,

DynamoDBService dynamoDBService,

ObjectMapper objectMapper) {

this.auditLogRepository = auditLogRepository;

this.dynamoDBService = dynamoDBService;

this.objectMapper = objectMapper;

}

}

2. SUCCESS LOGGING METHOD

public void logSuccess(String userId, String actionType, String entityType,

String entityId, String details) {

try {

AuditLog auditLog = createAuditLog(userId, actionType, entityType,

entityId, details, AuditStatus.SUCCESS, null);

// Save to MongoDB

auditLogRepository.save(auditLog);

// Save to DynamoDB asynchronously

CompletableFuture.runAsync(() -> {

try {

dynamoDBService.saveAuditLog(auditLog);

} catch (Exception e) {

log.error("Failed to save audit log to DynamoDB: {}", e.getMessage());

}

});

} catch (Exception e) {

log.error("Failed to log successful action: {}", e.getMessage());

}

}

3. FAILURE LOGGING METHOD

public void logFailure(String userId, String actionType, String entityType,

String entityId, String details, String errorMessage) {

try {

AuditLog auditLog = createAuditLog(userId, actionType, entityType,

entityId, details, AuditStatus.FAILURE, errorMessage);

// Save to MongoDB

auditLogRepository.save(auditLog);

// Save to DynamoDB asynchronously

CompletableFuture.runAsync(() -> {

try {

dynamoDBService.saveAuditLog(auditLog);

} catch (Exception e) {

log.error("Failed to save audit log to DynamoDB: {}", e.getMessage());

}

});

} catch (Exception e) {

log.error("Failed to log failed action: {}", e.getMessage());

}

}

4. AUDIT LOG CREATION HELPER

private AuditLog createAuditLog(String userId, String actionType, String entityType,

String entityId, String details, AuditStatus status, String errorMessage) {

AuditLog auditLog = new AuditLog();

auditLog.setActionId(generateActionId());

auditLog.setUserId(userId);

auditLog.setActionType(actionType);

auditLog.setEntityType(entityType);

auditLog.setEntityId(entityId);

auditLog.setActionDetails(details);

auditLog.setTimestamp(LocalDateTime.now());

auditLog.setStatus(status);

auditLog.setErrorMessage(errorMessage);

auditLog.setCreatedDate(LocalDateTime.now());

// Set request context information

HttpServletRequest request = getCurrentHttpRequest();

if (request != null) {

auditLog.setIpAddress(getClientIpAddress(request));

auditLog.setUserAgent(request.getHeader("User-Agent"));

}

return auditLog;

}

AUDIT LOG INTEGRATION

---------------------

1. TRANSACTION SERVICE INTEGRATION

@Service

public class DepositService {

@Autowired

private AuditService auditService;

public Transaction processDeposit(String accountNumber, BigDecimal amount,

String description, String userId) {

try {

// Process deposit logic

Transaction transaction = performDeposit(accountNumber, amount, description, userId);

// Log successful deposit

auditService.logSuccess(userId, "DEPOSIT", "TRANSACTION",

transaction.getId(),

"Deposit of Rs. " + amount + " processed successfully for account " + accountNumber);

return transaction;

} catch (Exception e) {

// Log failed deposit

auditService.logFailure(userId, "DEPOSIT", "TRANSACTION",

accountNumber,

"Deposit of Rs. " + amount + " failed for account " + accountNumber,

e.getMessage());

throw e;

}

}

}

2. TRANSFER SERVICE INTEGRATION

@Service

public class TransferService {

@Autowired

private AuditService auditService;

public Transaction processTransfer(String sourceAccountNumber, String destinationAccountNumber,

BigDecimal amount, String description, String userId) {

try {

// Process transfer logic

Transaction transaction = performTransfer(sourceAccountNumber, destinationAccountNumber,

amount, description, userId);

// Log successful transfer

auditService.logSuccess(userId, "TRANSFER", "TRANSACTION",

transaction.getId(),

"Transfer of Rs. " + amount + " from " + sourceAccountNumber +

" to " + destinationAccountNumber + " processed successfully");

return transaction;

} catch (Exception e) {

// Log failed transfer

auditService.logFailure(userId, "TRANSFER", "TRANSACTION",

sourceAccountNumber,

"Transfer of Rs. " + amount + " from " + sourceAccountNumber +

" to " + destinationAccountNumber + " failed",

e.getMessage());

throw e;

}

}

}

3. CUSTOMER SERVICE INTEGRATION

@Service

public class CustomerService {

@Autowired

private AuditService auditService;

public Customer createCustomer(Customer customer, String userId) {

try {

// Create customer logic

Customer savedCustomer = customerRepository.save(customer);

// Log successful customer creation

auditService.logSuccess(userId, "CREATE\_CUSTOMER", "CUSTOMER",

savedCustomer.getId(),

"Customer " + customer.getFirstName() + " " + customer.getLastName() +

" created successfully");

return savedCustomer;

} catch (Exception e) {

// Log failed customer creation

auditService.logFailure(userId, "CREATE\_CUSTOMER", "CUSTOMER",

null,

"Failed to create customer " + customer.getFirstName() + " " + customer.getLastName(),

e.getMessage());

throw e;

}

}

}

AUDIT LOG QUERIES AND RETRIEVAL

-------------------------------

1. AUDIT LOG REPOSITORY METHODS

@Repository

public interface AuditLogRepository extends MongoRepository<AuditLog, String> {

Optional<AuditLog> findByActionId(String actionId);

List<AuditLog> findByUserId(String userId);

List<AuditLog> findByActionType(String actionType);

List<AuditLog> findByEntityType(String entityType);

List<AuditLog> findByStatus(AuditStatus status);

List<AuditLog> findByTimestampBetween(LocalDateTime startDate, LocalDateTime endDate);

@Query("{ 'userId': ?0, 'timestamp': { $gte: ?1, $lte: ?2 } }")

List<AuditLog> findByUserIdAndTimestampBetween(String userId, LocalDateTime startDate, LocalDateTime endDate);

@Query("{ 'actionType': ?0, 'status': 'SUCCESS' }")

List<AuditLog> findSuccessfulActionsByType(String actionType);

@Query("{ 'entityType': ?0, 'entityId': ?1 }")

List<AuditLog> findByEntityTypeAndEntityId(String entityType, String entityId);

}

2. AUDIT LOG SERVICE QUERY METHODS

public List<AuditLog> getAuditLogsByUser(String userId, LocalDateTime fromDate, LocalDateTime toDate) {

return auditLogRepository.findByUserIdAndTimestampBetween(userId, fromDate, toDate);

}

public List<AuditLog> getAuditLogsByActionType(String actionType) {

return auditLogRepository.findByActionType(actionType);

}

public List<AuditLog> getAuditLogsByEntity(String entityType, String entityId) {

return auditLogRepository.findByEntityTypeAndEntityId(entityType, entityId);

}

public List<AuditLog> getFailedAuditLogs(LocalDateTime fromDate, LocalDateTime toDate) {

return auditLogRepository.findByStatusAndTimestampBetween(AuditStatus.FAILURE, fromDate, toDate);

}

AUDIT LOG CONTROLLER

--------------------

1. REST API ENDPOINTS

@RestController

@RequestMapping("/api/audit")

public class AuditController {

@Autowired

private AuditService auditService;

@GetMapping("/logs")

public ResponseEntity<List<AuditLog>> getAllAuditLogs(

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "20") int size) {

Pageable pageable = PageRequest.of(page, size, Sort.by("timestamp").descending());

Page<AuditLog> auditLogs = auditService.getAllAuditLogs(pageable);

return ResponseEntity.ok(auditLogs.getContent());

}

@GetMapping("/logs/user/{userId}")

public ResponseEntity<List<AuditLog>> getAuditLogsByUser(

@PathVariable String userId,

@RequestParam @DateTimeFormat(iso = DateTimeFormat.ISO.DATE\_TIME) LocalDateTime fromDate,

@RequestParam @DateTimeFormat(iso = DateTimeFormat.ISO.DATE\_TIME) LocalDateTime toDate) {

List<AuditLog> auditLogs = auditService.getAuditLogsByUser(userId, fromDate, toDate);

return ResponseEntity.ok(auditLogs);

}

@GetMapping("/logs/action/{actionType}")

public ResponseEntity<List<AuditLog>> getAuditLogsByActionType(

@PathVariable String actionType) {

List<AuditLog> auditLogs = auditService.getAuditLogsByActionType(actionType);

return ResponseEntity.ok(auditLogs);

}

@GetMapping("/logs/entity/{entityType}/{entityId}")

public ResponseEntity<List<AuditLog>> getAuditLogsByEntity(

@PathVariable String entityType,

@PathVariable String entityId) {

List<AuditLog> auditLogs = auditService.getAuditLogsByEntity(entityType, entityId);

return ResponseEntity.ok(auditLogs);

}

@GetMapping("/stats")

public ResponseEntity<AuditStats> getAuditStatistics(

@RequestParam @DateTimeFormat(iso = DateTimeFormat.ISO.DATE\_TIME) LocalDateTime fromDate,

@RequestParam @DateTimeFormat(iso = DateTimeFormat.ISO.DATE\_TIME) LocalDateTime toDate) {

AuditStats stats = auditService.getAuditStatistics(fromDate, toDate);

return ResponseEntity.ok(stats);

}

}

AUDIT LOG STATISTICS

--------------------

1. AUDIT STATISTICS MODEL

public class AuditStats {

private long totalActions;

private long successfulActions;

private long failedActions;

private Map<String, Long> actionsByType;

private Map<String, Long> actionsByUser;

private Map<String, Long> actionsByEntityType;

private LocalDateTime fromDate;

private LocalDateTime toDate;

// Constructors, getters, and setters

}

2. STATISTICS CALCULATION

public AuditStats getAuditStatistics(LocalDateTime fromDate, LocalDateTime toDate) {

List<AuditLog> auditLogs = auditLogRepository.findByTimestampBetween(fromDate, toDate);

AuditStats stats = new AuditStats();

stats.setFromDate(fromDate);

stats.setToDate(toDate);

stats.setTotalActions(auditLogs.size());

long successfulCount = auditLogs.stream()

.filter(log -> log.getStatus() == AuditStatus.SUCCESS)

.count();

stats.setSuccessfulActions(successfulCount);

stats.setFailedActions(auditLogs.size() - successfulCount);

// Group by action type

Map<String, Long> actionsByType = auditLogs.stream()

.collect(Collectors.groupingBy(AuditLog::getActionType, Collectors.counting()));

stats.setActionsByType(actionsByType);

// Group by user

Map<String, Long> actionsByUser = auditLogs.stream()

.collect(Collectors.groupingBy(AuditLog::getUserId, Collectors.counting()));

stats.setActionsByUser(actionsByUser);

// Group by entity type

Map<String, Long> actionsByEntityType = auditLogs.stream()

.collect(Collectors.groupingBy(AuditLog::getEntityType, Collectors.counting()));

stats.setActionsByEntityType(actionsByEntityType);

return stats;

}

SECURITY AND COMPLIANCE

-----------------------

1. AUDIT LOG SECURITY

- Tamper-proof audit trail

- Immutable audit records

- Secure storage and transmission

- Access control for audit log viewing

- Encryption of sensitive audit data

2. REGULATORY COMPLIANCE

- RBI (Reserve Bank of India) compliance

- KYC (Know Your Customer) audit requirements

- Anti-money laundering (AML) audit trails

- Data privacy and protection compliance

- Financial audit and reporting requirements

3. DATA RETENTION

- Configurable retention periods

- Automated archival processes

- Secure deletion procedures

- Compliance with legal requirements

- Backup and recovery procedures

PERFORMANCE OPTIMIZATION

------------------------

1. ASYNCHRONOUS LOGGING

- Non-blocking audit log operations

- Background processing for DynamoDB writes

- Queue-based audit log processing

- Batch processing for high-volume operations

2. INDEXING STRATEGY

- Optimized database indexes

- Composite indexes for common queries

- Time-based partitioning

- Efficient query performance

3. CACHING IMPLEMENTATION

- Frequently accessed audit logs

- Statistics caching

- Query result caching

- Performance optimization

MONITORING AND ALERTING

-----------------------

1. AUDIT LOG MONITORING

- Real-time audit log generation

- Failed audit log detection

- Performance monitoring

- Storage capacity monitoring

2. ALERTING MECHANISMS

- Failed audit log alerts

- High-volume activity alerts

- Security incident alerts

- System performance alerts

3. REPORTING CAPABILITIES

- Automated audit reports

- Compliance reporting

- Security incident reports

- Performance analytics

This comprehensive audit logging implementation ensures complete traceability, regulatory compliance, and security for the banking system operations.