# Phân tích Đặc tính Chất lượng - Event Sourcing Order Management System

## 1. Các đặc tính chất lượng mong muốn đạt được với thiết kế Event Sourcing

### 1.1 Khả năng truy vết (Auditability)

**Mục tiêu**: Cung cấp lịch sử đầy đủ của tất cả thay đổi.

- **Đặc tính:**

**Complete event log với timestamp và version**

- Mọi sự kiện được ghi lại đầy đủ với thời gian và phiên bản
- Giúp xác định chính xác thứ tự thay đổi và trạng thái hệ thống ở từng thời điểm
- *Ví dụ: "User A chuyển 100k vào lúc 10:05, sau đó rút 50k lúc 10:07"*

**Rollback history tracking**

- Có thể theo dõi lịch sử các hành động để biết được "ai đã làm gì và khi nào"
- Thường được dùng cho kiểm toán (audit log) hoặc forensic investigation
- *Ví dụ: Phát hiện nhân viên nội bộ đã thay đổi trạng thái đơn hàng bất hợp pháp*

**Event metadata preservation**

- Không chỉ lưu dữ liệu sự kiện, mà còn cả metadata như: userId, IP, correlationId, service origin
- Điều này giúp tái tạo lại ngữ cảnh của sự kiện
- *Ví dụ: Lưu lại IP gốc khi người dùng gửi request đặt hàng*

**Immutable event storage**

- Một khi sự kiện đã ghi, không được thay đổi/xoá (append-only)
- Giúp đảm bảo tính trung thực và minh bạch
- Nếu có lỗi, phải ghi một sự kiện bù trừ (compensating event) thay vì sửa log cũ

---

### 1.2 Khả năng phục hồi (Recoverability)

**Mục tiêu**: Có thể khôi phục trạng thái tại bất kỳ thời điểm nào.

- **Đặc tính:**

**Event replay capability**

- Có thể phát lại toàn bộ event từ đầu để dựng lại trạng thái hiện tại
- Giúp phục hồi khi database state bị hỏng hoặc mất
- *Ví dụ: Replay toàn bộ giao dịch để khôi phục số dư tài khoản*

**Time travel functionality**

- Có thể dựng lại trạng thái hệ thống ở một thời điểm cụ thể trong quá khứ
- Dùng để kiểm toán hoặc mô phỏng "what-if"
- *Ví dụ: "Tại ngày 01/08, số dư tài khoản khách hàng là bao nhiêu?"*

**State reconstruction từ event history**

- Toàn bộ state hiện tại không lưu trực tiếp mà được dựng từ event history
- Cho phép tái tạo nhiều read model khác nhau từ cùng một event log
- *Ví dụ: Từ cùng dữ liệu sự kiện bán hàng, có thể dựng báo cáo doanh thu theo ngày, theo nhân viên, theo sản phẩm*

**Rollback to specific version/timestamp**

- Có thể quay lại version cụ thể hoặc mốc thời gian cụ thể
- Dùng khi phát hiện bug logic, cần khôi phục hệ thống về trạng thái an toàn
- *Ví dụ: Rollback order system về thời điểm trước khi có lỗi double-payment*

---

### 1.3 Khả năng mở rộng (Scalability)

**Mục tiêu**: Hệ thống có thể xử lý tăng trưởng về dữ liệu và tải.

◇ **Đặc tính:**

**Stateless application design**

- Ứng dụng xử lý sự kiện không phụ thuộc vào state cục bộ, mà state được dựng lại từ event log
- Cho phép dễ dàng scale-out bằng cách thêm nhiều instance
- *Ví dụ: Nhiều worker cùng subscribe vào event stream để xử lý song song*

**Database indexing cho performance**

- Vì event log có thể rất lớn, cần index hợp lý để truy vấn sự kiện nhanh hơn
- Giúp giảm độ trễ khi query event theo aggregateId, timestamp
- *Ví dụ: Index theo aggregateId để lấy ra lịch sử thay đổi của một tài khoản cụ thể*

**Connection pooling**

- Khi lượng request tăng, cần tái sử dụng kết nối với database/event store thay vì mở mới
- Giúp hệ thống chịu tải cao mà không nghẽn tài nguyên
- *Ví dụ: 10k request/s vẫn giữ ổn định nhờ connection pool*

**Pagination support**

- Event log có thể chứa hàng triệu sự kiện → cần chia nhỏ khi load
- Giúp tránh tình trạng OOM (out of memory) khi xử lý hoặc replay toàn bộ log cùng lúc
- *Ví dụ: Replay event theo batch 1000 events/lần*

---

## 2. Công cụ và bước thực hiện kiểm tra đặc tính chất lượng

### 2.1 Kiểm tra Khả năng truy vết (Auditability)

**Công cụ:**

- **Event Store API**: Debug endpoints
- **Logging tools**: Winston, Bunyan
- **Database queries**: Event history analysis

**Bước thực hiện:**

```
# 1. Kiểm tra complete event log
curl -X GET "http://localhost:3001/api/debug/events" | jq

# 2. Kiểm tra order-specific events
curl -X GET "http://localhost:3001/api/debug/orders/{orderId}/events" | jq

# 3. Kiểm tra rollback history
curl -X GET "http://localhost:3001/api/debug/orders/{orderId}/skipped-versions" |
jq

# 4. Database audit query
psql -U postgres -d order_management -c "
SELECT aggregate_id, event_type, version, timestamp,
       event_data->>'orderId' as order_id,
       event_data->>'status' as status
FROM events
WHERE aggregate_id = 'order-001'
ORDER BY version;"
```

### 2.2 Kiểm tra Khả năng phục hồi (Recoverability)

**Công cụ:**

- **Rollback API**: Time travel functionality
- **Event replay testing**: State reconstruction
- **Snapshot testing**: Performance validation

**Bước thực hiện:**

```
# 1. Test rollback to specific version
curl -X POST "http://localhost:3001/api/debug/orders/{orderId}/rollback" \
  -H "Content-Type: application/json" \
  -d '{"toVersion": 3}' | jq

# 2. Test rollback to specific timestamp
curl -X POST "http://localhost:3001/api/debug/orders/{orderId}/rollback" \
  -H "Content-Type: application/json" \
  -d '{"toTimestamp": "2025-01-15T10:30:00.000Z"}' | jq
```

```
# 3. Test event replay
curl -X GET "http://localhost:3001/api/debug/orders/{orderId}/rebuild" | jq

# 4. Validate state consistency
curl -X GET "http://localhost:3001/api/orders/{orderId}" | jq
```

### 2.3 Kiểm tra Khả năng mở rộng (Scalability)

**Công cụ:**

- **Load testing**: Artillery, Apache Bench
- **Database monitoring**: pg_stat_statements
- **Performance profiling**: Node.js profiler

**Bước thực hiện:**

```
# 1. Load testing với Artillery
npm install -g artillery
artillery quick --count 100 --num 10 http://localhost:3001/api/orders

# 2. Database performance monitoring
psql -U postgres -d order_management -c "
SELECT query, calls, total_time, mean_time
FROM pg_stat_statements
WHERE query LIKE '%events%'
ORDER BY total_time DESC;"

# 3. Index usage analysis
psql -U postgres -d order_management -c "
SELECT schemaname, tablename, indexname, idx_scan, idx_tup_read, idx_tup_fetch
FROM pg_stat_user_indexes
WHERE tablename = 'events';"

# 4. Connection pool monitoring
psql -U postgres -d order_management -c "
SELECT * FROM pg_stat_activity
WHERE datname = 'order_management';"
```