Application Logging Guideline

### **1. Purpose**

Define clear, consistent logging practices across all services to aid in troubleshooting, observability, performance analysis, and compliance.

### **2. Log Level Guidelines**

| **Level** | **Purpose** |
| --- | --- |
| TRACE | Very fine-grained, useful for deep debugging ex. Entry/exit of methods, deep internals |
| DEBUG | Detailed info for debugging in dev/test. ex. Parameters, DB queries, cache status |
| INFO | Application flow, startup, shutdown, user actions, service start/stop, flow |
| WARN | Potential issues or unusual situations. ex. Retry attempts, fallbacks, slow response |
| ERROR | Application errors or failures and data inconsistency |
| FATAL | Critical errors causing the system to stop. |

✔ **Rule of thumb:** Log only what's necessary at each level — don’t flood logs with DEBUG in production.

### **3. Log Format**

* **Structured JSON or key-value format,** Makes it easier to parse in centralized logging systems
* Required fields:
  + @timestamp
  + log.level
  + service.name
  + message
  + trace.id, span.id (for distributed tracing)
  + user.id, request.id (if available)
* Example:

| {  "@timestamp": "2025-03-12T08:00:00.853Z",  "log.level": "INFO",  "service.name": "payment-service",  "message": "Payment processed successfully",  "user.id": "USR456"  } |
| --- |

✔ Use **MDC (Mapped Diagnostic Context)** or similar in Java/Spring projects to auto-inject context info.

**4. Don’t Log Sensitive Data / Sensitive Data Masking**

* Mask or avoid logging:
  + Passwords
  + API keys
  + Credit card numbers
  + PII (Personally Identifiable Information)

✔ Apply log sanitizers or filters.

**5. Use Logging Libraries, Not System.out**

* Encourage consistent use of logging frameworks:
  + Java: **SLF4J + Logback / Log4j2**
  + Node.js: **Winston / Pino**
  + Python: **logging**
  + Go: **Zap / Logrus**

### **6. Log Errors with Stack Trace**

Always log **exception messages + stack traces**, but only at ERROR or DEBUG level depending on sensitivity.

Example:

| logger.error("Failed to process payment for order $orderId", ex) |
| --- |

### **7. Avoid Over-Logging**

### Avoid duplicate log entries.

### Avoid logging inside tight loops.

### Reduce noise: don’t log every DB query unless needed.

### **8. Log Rotation and Retention**

* Setup log rotation (daily/size-based).
* Define retention policies (e.g., 30/90 days).
* Archive or purge old logs appropriately.

### **9. Application Events That Must Be Logged**

| Event Type | Log Level | Description |
| --- | --- | --- |
| Incoming API Request | INFO | Log all incoming requests with important context (endpoint, parameters, requestId, userId) |
| Business Process Start/End | INFO | Log entry/exit for business operations or use cases |
| Data Access Events | DEBUG | Log repository calls and queries |
| External Service Calls | INFO or DEBUG | Log outbound requests (URL, payload, response) |
| Validation Failures | WARN | Log when request validation or business rules fail |
| Authorization/Authentication Events | INFO/WARN | Log login attempts, roles, or access failures |
| Exceptions/Errors | ERROR | Log unhandled exceptions or known business errors with full context and stack trace |
| State Changes / Mutations | INFO | Log when entities are created/updated/deleted |
| Scheduled Job Executions | INFO | Log job runs, status, and duration |
| Application Startup / Shutdown | INFO | Log system startup, version, profiles, shutdown time |

### 

### **10. Application Logging Events Checklist**

## ✔ Use This Checklist for:

* Code Reviews
* Logging Conventions in Dev Onboarding Docs
* Unit/Integration Test Observability Checks

### 

### **🔹 1. Incoming API Request Logging**

### Log on every controller entry point

### Include: path, method, requestId, userId, params (exclude sensitive values)

### Example: Fetch all

| //entry  logger.info("Fetching all orders: path={}, method={}, requestId={}, userId={}, filters={}, pageable={}, sort={}", path, method, requestId, userId, filters, pageable, sort) |
| --- |

### Example: Fetch by ID

| //entry  logger.info("Fetching order by ID: path={}, method={}, requestId={}, userId={}", path, method, requestId, userId) |
| --- |

### Example: Create

| //entry  logger.info("Creating new order: path={}, method={}, requestId={}, userId={}", path, method, requestId, userId) |
| --- |

### Example: Update by ID

| //entry  logger.info("Updating order by ID: path={}, method={}, requestId={}, userId={}", path, method, requestId, userId) |
| --- |

### Example: Delete by ID

| //entry  logger.info("Deleting order by ID: path={}, method={}, requestId={}, userId={}", path, method, requestId, userId) |
| --- |

### 

### **🔹 2. Business Logic Execution**

### Log entry/exit success (or failed)

### Include: use-case name, object id(s), requestId, duration

### Example:

| //entry  logger.info("Creating new order with: requestId={}", requestId)  .  .  .  //exit - success  logger.info("Order created successfully: orderId={}, requestId={}, duration={}", savedOrder.orderId, requestId, duration)  //exit - failed  logger.error("Error creating order: requestId={}", requestId, ex) |
| --- |

### 

### 

### **🔹 3. Data Access Layer (Repository Logging)**

* Log read/write operations
* Include: parameters (exclude sensitive values)
* Log at DEBUG level

### Example: Read operation

| //entry  logger.debug("Finding order by ID: orderId={}", orderId) |
| --- |

### Example: Write operation

| //entry  logger.debug("Save order: orderId={}", orderId) |
| --- |

### 

### **🔹 4. External Service Calls**

* Log with request and response metadata
* Include: target service, URL, status code, duration, requestId

### Example:

| //entry  logger.info("Call payment service: serviceName={}, url={}, requestId={}", serviceName, url, requestId)  .  .  .  //exit - success  logger.info("Call payment service successfully: serviceName={}, url={}, requestId={}, status={}, duration={}", serviceName, url, requestId, status, duration)  //exit - failed  logger.error("Error call payment service: serviceName={}, url={}, requestId={}", serviceName, url, requestId, ex) |
| --- |

### 

### 

### **🔹 5. Validation & Business Rule Failures**

* Log when validation failed or business rule violated
* Include: reason, affected field/entity
* Log at WARN level

### Example:

| //validate failed  logger.warn("Invalid requested fields: {}", invalidFields) |
| --- |

### 

### **🔹 6. Authorization & Authentication Events**

* Log login attempts, access grants/denials
* Include: userId, Client IP, path

### Example: Login success

| //login success  logger.info("Login success: userId={}, ip={}", userId, clientIP) |
| --- |

### Example: Access denied

| //access denied  logger.warn("Access denied: userId={}, path={}", userId, path) |
| --- |

### 

### **🔹 7. Exceptions and Failures**

* Log all unexpected errors at ERROR level
* Include: reason, entityId, stack trace
* Always include requestId for correlation

### Example:

| //  logger.error("Error update order by ID: orderId={}, requestId={}", orderId, requestId, ex) |
| --- |

### 

### **🔹 8. State Changes (Create/Update/Delete)**

* Log lifecycle changes of entities
* Include: entityId

### Example:

| //exit  logger.info("Order created successfully: orderId={}, userId={}", savedOrder.orderId, savedOrder.createdBy) |
| --- |

### 

### **🔹 9. Scheduled Jobs**

* Log job start and completion
* Include: event=JobStart, event=JobSuccess, duration, status

### Example:

| //entry  logger.info("JobStart DailyInventorySync”)  .  .  .  //exit - success  logger.info("JobSuccess DailyInventorySync: status={}, duration={}", status, duration)  //exit - failed  logger.error("JobFailed DailyInventorySync", ex) |
| --- |

### 

### **🔹 10. Application Lifecycle Events**

* Log application startup/shutdown
* Include: environment, version

### Example:

| //app startup  logger.info("AppStartup: version={}, environment={}", version, environment)  //app shutdown  logger.info("AppShutdown") |
| --- |

## 

## **🔒 Sensitive Data Logging Guidelines**

* ❌ Do NOT log: passwords, tokens, credit card info, session ids.
* ✅ Mask if needed: userEmail=\*\*\*@\*\*\*.com

### 

### Appendix

* Field Reference - List of Fields

Field Reference

*อ้างอิงมาจาก Elastic Common Schema 8.17*

*https://www.elastic.co/guide/en/ecs/current/index.html*

Base Fileds

Base Field Details

| Field | Description |
| --- | --- |
| @timestamp | Date/time when the event originated.  This is the date/time extracted from the event, typically representing when the event was generated by the source.  If the event source has no original timestamp, this value is typically populated by the first time the event was received by the pipeline.  Required field for all events.  type: date  example: 2016-05-23T08:05:34.853Z |
| message | For log events the message field contains the log message, optimized for viewing in a log viewer.  For structured logs without an original message field, other fields can be concatenated to form a human-readable summary of the event.  If multiple messages exist, they can be combined into one message.  type: match\_only\_text  example: Hello World |
| ?เพิมเติมสำหรับ structured logging |  |

Log Fileds

Log Field Details

| Field | Description |
| --- | --- |
| log.level | Original log level of the log event.  If the source of the event provides a log level or textual severity, this is the one that goes in log.level. If your source doesn’t specify one, you may put your event transport’s severity here (e.g. Syslog severity).  Some examples are warn, err, i, informational.  type: keyword  example: error |
| ? |  |
| ? |  |

### 

Referrence

?