Requirement Document: Centralized Logging REST API

Project Name: Centralized Logging REST API
Prepared By: Julio Diaz
Date: Oct 7, 2024

Objective

Develop a Spring Boot-based REST API that serves as a centralized logging solution for distributed components. The API should support both detailed and basic log entries through a single endpoint, process these entries using the ArcCommonLogger Java library, and forward the logs to an upstream system for integration with Azure HDInsight and Big Panda.

Scope

The developer is required to:

1. Implement a single `/api/logs` endpoint that can handle both basic and detailed logging requests.
2. Integrate the ArcCommonLogger Java library to process and forward logs.
3. Dynamically parse incoming log entries to determine the logging type (basic or detailed).
4. Convert basic log entries into a detailed format with default values.
5. Ensure appropriate error handling, logging, and validation for incoming requests.

Functional Requirements

1. **Unified Logging Endpoint**

    - **Endpoint:** `POST /api/logs`
    - **Description:** Accepts log entries from external services, handling both basic and detailed logging formats.
    - **Request Body:** JSON object that can represent either a basic or detailed log entry.
    - **Response Codes:**

        - `200 OK`: Log successfully processed and forwarded.
        - `400 Bad Request`: Validation error in the input data.
        - `500 Internal Server Error`: Errors while processing the log.

2. **Log Parsing and Mapping**

    - **Requirement:** Implement logic to parse the incoming JSON request into a `Map<String, Object>` or a generic `LogRequest` wrapper class.
    - **Detailed Logs:** If fields like `processName` or `metrics` are present, map the request to a `LogEntry` object.
    - **Basic Logs:** If these fields are absent, treat the request as a `BasicLogEntry` and populate missing fields with default values.

- **Mapping Logic:**
    - `timestamp`: Use the current timestamp if not provided.
    - `processName`: Use "DEFAULT_PROCESS" if not provided.
    - `subSystem`: Use "DEFAULT_SUBSYSTEM" if not provided.
    - `metrics`: Generate default metrics.
    - `severityLevel`: Default to 1 if not provided.

3. **Integration with ArcCommonLogger**
    - **Library:** ArcCommonLogger (Java library provided)
    - **Details:**
        - Use the ArcCommonLogger to create `InfoEvent` and `ErrorEvent` objects.
        - Log entries based on `logLevel`:
            - For `logLevel` values of "ERROR", create and send an `ErrorEvent`.
            - For other log levels, create and send an `InfoEvent`.

4. **Error Handling**
    - **Validation:** Ensure that `logLevel`, `message`, and `componentName` are present in every request.
    - **Exception Handling:** Implement error handling to catch and return user-friendly error messages for:
        - Missing required fields (`400 Bad Request`).
        - JSON parsing errors.
        - Internal errors during log processing (`500 Internal Server Error`).

Non-Functional Requirements

1. **Security**
    - **Authentication:** Implement OAuth2 or JWT-based authentication for accessing the `/api/logs` endpoint.
    - **Authorization:** Use role-based access control (RBAC) to restrict access.
    - **Data Protection:** Ensure that sensitive information is not logged or exposed in error messages.

2. **Performance**
    - **Response Time:** The API should process log entries and respond within 200ms under normal load conditions.
    - **Scalability:** The API should be capable of handling a high volume of concurrent log requests.

3. **Testing Requirements**
    - **Unit Tests:**

        - Test with minimal required fields to ensure basic logging works.
        - Test with detailed fields to ensure detailed logging works.
    - **Integration Tests:**
        - Use `MockMvc` to test `POST` requests to `/api/logs` with various payloads.
        - Validate that logs are correctly processed and forwarded.
    - **Validation Tests:**
        - Test scenarios with missing required fields to ensure appropriate error responses.
    - **Error Handling Tests:**
        - Simulate internal server errors to verify the correct response (`500`).

Technical Details

1. **Data Models**
    - **BasicLogEntry.java**
        - **Fields:** `logLevel`, `message`, `componentName`
        - This class represents a simplified log entry.
    - **LogEntry.java**
        - **Fields:** `timestamp`, `logLevel`, `componentName`, `processName`, `subSystem`, `message`, `metrics`, `severityLevel`, `exception`, `customProperties`
        - This class represents a detailed log entry.
        - `Metrics` and `ExceptionDetails` should be nested classes to handle respective details.
2. **Service Class**
    - **Class Name:** `LogService`
    - **Responsibilities:**
        - Parse and validate incoming JSON log entries.
        - Convert `BasicLogEntry` to `LogEntry` with default values.
        - Handle `InfoEvent` and `ErrorEvent` creation using ArcCommonLogger.
        - Implement methods `processLog(Map<String, Object> logRequest)`, `processLog(LogEntry logEntry)`, and `processLog(BasicLogEntry basicLogEntry)`.
3. **Controller Class**
    - **Class Name:** `LogController`
    - **Endpoint:** `POST /api/logs`
    - **Responsibilities:**
        - Accept JSON requests and forward them to the `LogService` for processing.

        - Handle exceptions and return appropriate HTTP status codes.

4. **Example Implementation Snippet**
```
java
@RestController
@RequestMapping("/api/logs")
public class LogController {
    private final LogService logService;

    public LogController(LogService logService) {
        this.logService = logService;
    }

    @PostMapping
    public ResponseEntity<String> receiveLog(@RequestBody Map<String, Object> logRequest) {
        try {
            logService.processLog(logRequest);
            return ResponseEntity.ok("Log processed successfully.");
        } catch (IllegalArgumentException e) {
            return ResponseEntity.badRequest().body("Invalid log request: " + e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("An error occurred while processing the log.");
        }
    }
}
```

Deliverables

- Spring Boot Application with the `/api/logs` endpoint.
- `LogService` and `LogController` classes implementing the logic described.
- Unit and Integration Tests covering all scenarios.
- Deployment Documentation including instructions for running the service and configuring ArcCommonLogger.

Notes

- Review the provided ArcCommonLogger documentation to ensure proper integration.
- Follow the coding standards and best practices for Java and Spring Boot.
- Ensure proper logging and documentation within the code for future maintainability.

Please reach out if you have any questions or need further clarification on the requirements.