Designing a REST API

Variant: Server Logging System

Done by Sych Denys

The Server Logging System is a RESTful API designed to collect, store, query, and manage server log data generated by distributed systems. The system supports structured logs with severity levels, timestamps, sources, and metadata, enabling efficient monitoring, debugging, and auditing.

Functional Requirements (F)

F1. The system must accept new log entries from external services via API.

F2. Users should be able to obtain a list of logs filtered by severity, service name, and time range.

F3. The ability to view detailed information about a specific log entry by its ID.

F4. The administrator should be able to delete old logs or archive them.

Non-Functional Requirements (NF)

NF1. The API must be able to handle high write loads (write-heavy system).

NF2. Log writing must be asynchronous or very fast (<50ms) so as not to slow down core services.

NF3. Only authorised services can write logs; only authorised users can read them.

NF4. Use of standard JSON format and REST principles.

Model Description

Entity 1: LogEntry

Represents a single logged event.

- **id** (UUID): Unique identifier.
- **timestamp** (ISO 8601 String): Event time.
- **severity** (Enum): INFO, WARN, ERROR, FATAL.
- **message** (String): The log message.

- **metadata** (JSON): Additional context.
- **sourceId** (UUID): Reference to the Source entity.

Entity 2: Source

Represents a service that sends logs.

- **id** (UUID): Unique identifier.
- **serviceName** (String): Service human-readable name.
- **apiKey** (String): Encrypted authentication key.
- **status** (Enum): ACTIVE, INACTIVE.

Entity 3: User

Represents a system user.

- **id** (UUID): Unique identifier.
- **username** (String): Login name.
- **role** (Enum): ADMIN (can delete/archive), USER (read-only).

Operations Description & Status Codes

The API uses standard HTTP headers:

- Content-Type: application/json
- Authorization: Bearer <token> (for Users) or X-API-Key: <key> (for Sources).

3.1. Ingest Log (Create)

- **Method:** POST
- **URI:** /api/v1/logs
- **Description:** Accepts a new log entry from a Source.
- **Status Codes:**
  - 201 Created: Log successfully queued/saved.
  - 400 Bad Request: Missing mandatory fields.
  - 401 Unauthorized: Invalid or missing API Key.

3.2. Search Logs (Read Collection)

- **Method:** GET
- **URI:** /api/v1/logs

- **Description:** Retrieves a paginated list of logs based on filters.
- **Parameters:** ?severity=ERROR&sourceId={id}&startTime={iso}&limit=50
- **Status Codes:**
    - 200 OK: Successful retrieval.
    - 400 Bad Request: Invalid date format or query parameter.

3.3. Get Log Detail (Read Resource)

- **Method:** GET
- **URI:** /api/v1/logs/{id}
- **Description:** Retrieves full details of a specific log.
- **Status Codes:**
    - 200 OK: Success.
    - 404 Not Found: ID does not exist.
    - 304 Not Modified: Resource has not changed (Caching).

3.4. Delete Log (Delete)

- **Method:** DELETE
- **URI:** /api/v1/logs/{id}
- **Description:** Permanently removes a log entry. Restricted to ADMIN.
- **Status Codes:**
    - 204 No Content: Deletion successful.
    - 403 Forbidden: User does not have ADMIN role.
    - 404 Not Found: Log already deleted or does not exist.

Richardson Model Application (HATEOAS)

The API implements **Level 3** of the Richardson Maturity Model. Every response includes a _links object, guiding the client on available next actions (state transitions).

Example Response for GET /api/v1/logs/123:

```
{
  "id": "123",
  "severity": "ERROR",
  "message": "Database timeout",
  "_links": {
    "self": { "href": "/api/v1/logs/123", "method": "GET" },
    "collection": { "href": "/api/v1/logs", "method": "GET" },
    "delete": { "href": "/api/v1/logs/123", "method": "DELETE" }
  }
}
```

Authentication

The system uses a hybrid authentication approach:

1. **For Sources (Ingestion):**
   - Method: **API Key**. The Source sends a header X-API-Key.
   - Error: If the key is missing or invalid, the API returns 401 Unauthorized.
2. **For Users (Management/Querying):**
   - Method: **JWT (JSON Web Token)**. The User sends a header Authorization: Bearer <token>.
   - The JWT Payload contains: { "sub": "userId", "role": "ADMIN", "exp": 1735689600 }.
   - Error: If the token is expired, returns 401 Unauthorized. If the role is insufficient, returns 403 Forbidden.

Pagination

Because log data volume is massive, the GET /api/v1/logs endpoint **mandates** pagination using the Offset/Limit strategy.

- **Request:** GET /logs?page=2&limit=50
- **Default Behavior:** If parameters are omitted, defaults to page=1 and limit=20.
- **Max Limit:** The API enforces a maximum limit=100 to prevent performance degradation.
- **Response Metadata:** The response body includes total_pages, current_page, and total_items.

Caching

Caching is implemented via HTTP Headers to reduce load on the database.

1. **Cacheable Methods (GET):**

- **Single Log (GET /logs/{id}):** Logs are immutable (historical data).
    - Header: Cache-Control: public, max-age=31536000 (1 year).
    - Validation: Uses ETag. If the client sends If-None-Match, server returns 304 Not Modified.
- **Source List (GET /sources):** Metadata changes rarely.
    - Header: Cache-Control: public, max-age=3600 (1 hour).

2. **Non-Cacheable Methods:**
    - **Log List (GET /logs):** Not cached (no-store) because new logs arrive milliseconds apart, making the list instantly stale.
    - **POST/DELETE:** Never cached as they alter the server state.