**Use Case 4:**

Implementing Robust Item Loading Functionality for an E-Commerce Platform

Enabling Seamless Item Integration and Management

# Overview

Below are the typical CRUD (Create, Read, Update, Delete) operations—mapped to HTTP verbs—for each microservice in the Item Management System:

**Item Service:**

GET: Retrieve item details

POST: Create a new item

PUT: Update item information

DELETE: Remove an item

**Inventory Service:**

GET: View inventory levels and transactions

POST: Add inventory for an item

PUT: Update inventory records

DELETE: Remove inventory records

**Notification Service:**

GET: View notification logs

POST: Send a new notification

PUT: Update notification settings

DELETE: Remove notification entries

**Audit & Logging Service:**

GET: Retrieve audit logs

POST: Log a new event

PUT: Update log information

DELETE: Remove log entries

Microservice Architecture and Operations

🧾 Overview

This system manages item data and related business processes using a modern microservice architecture. Each domain area—such as items, inventory, notifications, and audit logging—is managed by an independent service for scalability, maintainability, and clear separation of concerns.

🏗️ Architecture

- Spring Boot microservices for each domain

- Authentication & Authorization for microservices

- RESTful APIs with structured JSON payloads

- Dedicated database tables per service

- API Gateway for central routing (optional)

- Service Discovery (optional)

- Swagger/OpenAPI for documentation

- Each service is independently deployable

🧩 Microservices Included

1. Item Service

- Manages item details and operations.

🗄️ DB Schema: items

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | name | description | category | created\_at |
| (BIGINT, PK, AUTO\_INCREMENT) | (VARCHAR(150), NOT NULL) | (TEXT, NULL) | (VARCHAR(100), NOT NULL) | (TIMESTAMP, NOT NULL) |

🧾 Sample JSON:

{

"name": "Wireless Mouse",

"description": "Ergonomic wireless mouse with USB receiver.",

"category": "Electronics",

"createdAt": "2025-07-28T09:00:00"

}

2. Inventory Service

- Handles inventory tracking and stock operations.

🗄️ DB Schema: inventory

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | item\_id | quantity | warehouse\_location | last\_updated |
| (BIGINT, PK, AUTO\_INCREMENT) | (BIGINT, FK) | (INT, NOT NULL) | (VARCHAR(100), NOT NULL) | (TIMESTAMP, NOT NULL) |

🧾 Sample JSON:

{

"itemId": 1005,

"quantity": 150,

"warehouseLocation": "A1-North",

"lastUpdated": "2025-07-28T10:30:00"

}

3. Notification Service

- Sends notifications based on system events.

🗄️ DB Schema: notifications

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | event\_type | recipient | message | sent\_at |
| (BIGINT, PK, AUTO\_INCREMENT) | (VARCHAR(100), NOT NULL) | (VARCHAR(150), NOT NULL) | (TEXT, NOT NULL) | (TIMESTAMP, NOT NULL) |

🧾 Sample JSON:

{

"eventType": "LowStock",

"recipient": "inventory.manager@company.com",

"message": "Stock for item 1005 has dropped below threshold.",

"sentAt": "2025-07-28T11:00:00"

}

4. Audit & Logging Service

- Tracks and records critical operations.

🗄️ DB Schema: audit\_logs

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | service\_name | operation | record\_id | timestamp | details |
| (BIGINT, PK, AUTO\_INCREMENT) | (VARCHAR(100), NOT NULL) | (VARCHAR(50), NOT NULL) | (BIGINT, NOT NULL) | (TIMESTAMP, NOT NULL) | (TEXT, NULL) |

🧾 Sample JSON:

{

"serviceName": "Item Service",

"operation": "DELETE",

"recordId": 1005,

"timestamp": "2025-07-28T12:00:00",

"details": "Item deleted by admin user"

}

🔧 API Operations and Integration

Each service exposes RESTful APIs with well-structured JSON formats for interoperability.

All CRUD operations follow REST conventions—POST (create), GET (read), PUT (update), DELETE (remove).

Validation is enforced at the service layer to ensure data integrity (e.g., required fields, format checks).

⚠️ Error Handling

- Descriptive error messages returned as JSON.

- No stack traces exposed to clients.

- HTTP status codes correctly implemented (201 Created, 200 OK, 400 Bad Request, 404 Not Found).

🧪 Testing

- Integration tests for end-to-end API flows.

- Unit tests for business logic coverage.

Note: Each microservice is independently deployable and can be extended for future needs such as analytics, reporting, or third-party integrations.