**Deliverables for E-Commerce Microservices Demo with Kafka**

[1. Project Overview (PPT Slide) 1](#_Toc1763864945)

[2. System Architecture Diagram (PPT Slide) 1](#_Toc1460661636)

[3. Sample Test Data 1](#_Toc322651317)

[4. Entity Relationship Diagram (ERD) 2](#_Toc1307004007)

[5. Process Flows / Sequence Diagrams 2](#_Toc1761642203)

[6. Codebase Organization 2](#_Toc665764160)

[7. GitHub Repositories 3](#_Toc1347476916)

[8. Kafka Topics Documentation 3](#_Toc1710773715)

[9. Demo Plan / Test Cases 3](#_Toc1577701154)

[10. Feature Summary (Excel or Word Table) 4](#_Toc1183002885)

[11. Retrospective Notes (Optional Slide) 4](#_Toc1563632950)

[12. Final Submission Format (for Demo Only) 4](#_Toc903824422)

[13. Live Project Folder Structure (Real Repos) 5](#_Toc872660024)

### **1. Project Overview (PPT Slide)**

* **Title**: E-Commerce Shopping Site – Microservices Architecture
* **Team Members**: Gavish, Piyush, Saurabh, Naveen, Prajwal, Bharti, Chagam
* **Stack**:
  + Backend: Spring Boot
  + Messaging: Apache Kafka (running on WSL2 Ubuntu)
  + REST-based microservices (running on Windows)
  + DB: MySQL (based on use case)
  + GitHub: Individual repositories per service
* **Key Features**: Product catalog, ordering, recommendations, vendor evaluation

### **2. System Architecture Diagram (PPT Slide)**

* Show:
  + 3 Microservices:
    - CatalogService
    - OrderingService
    - VendorService
    - PaymentService
    - DeliveryService
    - EnhancementService
  + Kafka Broker (separate service on WSL2 Ubuntu)
  + Kafka Topics (e.g., order-events, product-updates, vendor-evaluation)
  + REST APIs exposed per microservice
  + Communication: Kafka + REST
  + DB connections (if any)
  + Integrations: Swagger UI
* Tools: Lucidchart / Draw.io

### **3. Sample Test Data**

* Each microservice should include sufficient sample data in its DB:
  + CatalogService: Categories, Products, Inventory
  + OrderingService: Customers, Cart Items, Orders
  + VendorService: Vendors, Mappings, Evaluations
* Data should be auto loaded via data.sql or via Flyway for testing/demo

### **4. Entity Relationship Diagram (ERD)**

* Minimum 6 ERDs (1 per microservice)
* Include:
  + **CatalogService**: Product, Category, Inventory
  + **OrderingService**: Cart, Order, Customer, Address
  + **VendorService**: Vendor, Rating, ProductMapping
* Tools: dbdiagram.io / Draw.io

### **5. Process Flows / Sequence Diagrams**

* 3–4 core flows (PPT slides):
  + **Product Browsing & Search**  
     → UI → CatalogService → DB
  + **Add to Cart & Place Order**  
     → UI → OrderingService → Kafka → VendorService
  + **Vendor Evaluation via Kafka Event**  
     → Kafka Event → VendorService → Save Evaluation
  + **Admin Modifying Product Info**  
     → Admin Portal → CatalogService → Kafka Event → Re-index/Search updates

### **6. Codebase Organization**

Each microservice should include:

* Spring Boot REST APIs
* KafkaProducer and KafkaConsumer (Spring Kafka Template)
* DTOs & Entity Models
* Services Layer & Event Handlers
* application.yml with Kafka and DB configs
* Testing (unit/integration optional)
* Error handling and logging
* Swagger for API docs

### **7. GitHub Repositories**

* Each service in separate repository:
  + catalog-service
  + ordering-service
  + vendor-service
  + payment-service
  + delivery-service
  + enhancement-service
* README.md for each with:
  + Setup instructions
  + Kafka topic details
  + REST API endpoints
  + Sample request/response
  + Local run guide (including Kafka setup on WSL2)

### **8. Kafka Topics Documentation**

* Table listing:
  + Topic Name
  + Key / Value structure
  + Producer Service
  + Consumer Service
  + Example payload (JSON)
  + Sample Kafka console commands (to test manually if needed)

### **9. Demo Plan / Test Cases**

* Scenarios to cover during the demo (in PPT or Word):
  + Product browsing & filtering
  + Add to cart → Place order → Event triggers
  + Vendor rating auto-evaluation via Kafka
  + Admin updates triggering Kafka messages
* Optionally include Postman collection
* Demonstrate testing of API endpoints on Swagger UI

### **10. Feature Summary (Excel or Word Table)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story** | **Microservice** | **Priority** | **Status** | **Assigned** |
| Browse Products | Catalog | High | Completed | Gavish |
| Place Order | Ordering | High | Completed | Prajwal |
| Vendor Rating | Vendor | Medium | In Progress | Bharti |
| *(and so on...)* |  |  |  |  |

### **11. Retrospective Notes (Optional Slide)**

* What worked well
* Challenges faced (e.g., Kafka config, cross-platform setup)
* Lessons learned
* Improvements/future roadmap

### **12. Final Submission Format (for Demo Only)**

* **Folder Structure (ZIP or GitHub OR GDrive link)**:

ecommerce-microservices-demo/

├── architecture-diagram.pptx

├── process-flows/

│ ├── place-order-sequence.png

│ └── vendor-evaluation.png

├── erd-diagrams/

│ ├── catalog-erd.png

│ ├── ordering-erd.png

│ └── vendor-erd.png

├── kafka-topics.md

├── demo-plan.docx

├── feature-summary.xlsx

├── catalog-service-link.txt # Link to GitHub repo

├── ordering-service-link.txt # Link to GitHub repo

├── vendor-service-link.txt # Link to GitHub repo

└── README.md

├── postman-collection.json

├── **api-docs/**

│ ├── catalog-service-api.md

│ ├── ordering-service-api.md

│ ├── vendor-service-api.md

| ├── payment-service-api.md

| ├── delivery-service-api.md

| ├── enhancement-service-api.md

│ └── common-errors.md

### **Contents of api-docs/\*.md**

Each .md file should document:

* List of endpoints
* HTTP method, path, description
* Request/response samples
* Status codes
* Notes on Kafka events (if triggered)

# **Sample Catalog Service API Documentation**

This document contains the REST API endpoints for the **Catalog Service**.

## **Base Path: /api/catalog**

### **1. Get All Products**

* **Method**: GET
* **Endpoint**: /products
* **Description**: Retrieves a list of all products in the catalog.
* **Response**:

[  
 {  
 "id": 101,  
 "name": "iPhone 15",  
 "category": "Mobiles",  
 "price": 79999,  
 "inventory": 20  
 }  
]

### **2. Get Product by ID**

* **Method**: GET
* **Endpoint**: /products/{id}
* **Description**: Fetches a single product by its ID.
* **Path Variable**: id – Product ID
* **Response**:

{  
 "id": 101,  
 "name": "iPhone 15",  
 "category": "Mobiles",  
 "price": 79999,  
 "inventory": 20  
}

### **3. Search Products by Keyword**

* **Method**: GET
* **Endpoint**: /products/search?q={keyword}
* **Description**: Returns products matching a search keyword.
* **Query Param**: q – Keyword
* **Response**:

[  
 {  
 "id": 102,  
 "name": "Samsung Galaxy",  
 "category": "Mobiles"  
 }  
]

### **4. Get All Categories**

* **Method**: GET
* **Endpoint**: /categories
* **Description**: Returns the list of product categories and subcategories.

### **5. Create a Product**

* **Method**: POST
* **Endpoint**: /products
* **Description**: Creates a new product (Admin only).
* **Request Body**:

{  
 "name": "iPhone 15",  
 "categoryId": 1,  
 "price": 79999,  
 "inventory": 10,  
 "imageUrl": "<https://example.com/iphone15.png>"  
}

* **Response**: 201 Created

### **6. Update Product Info**

* **Method**: PUT
* **Endpoint**: /products/{id}
* **Description**: Updates an existing product (Admin only).
* **Request Body**: Same as create
* **Response**: 200 OK

### **7. Delete a Product**

* **Method**: DELETE
* **Endpoint**: /products/{id}
* **Description**: Deletes a product by ID (Admin only).
* **Response**: 204 No Content

## **Kafka Integration**

* **Topic Name**: product-updates
* **Producer**: CatalogService
* **Consumer**: SearchService (if any)
* **Triggered When**:
  + Product Created
  + Product Updated
  + Product Deleted
* **Sample Payload**:

{  
 "eventType": "PRODUCT\_UPDATED",  
 "productId": 101,  
 "name": "iPhone 15 Pro",  
 "price": 84999  
}

## **Status Codes**

|  |  |
| --- | --- |
| **Code** | **Description** |
| 200 | OK |
| 201 | Created |
| 204 | No Content |
| 400 | Bad Request |
| 404 | Not Found |
| 500 | Server Error |

## **Authentication & Roles**

* POST, PUT, and DELETE endpoints require role: ADMIN

## **Tools / Testing**

* **Swagger UI**: /swagger-ui.html
* **Postman Collection**: Included in demo repo as postman-collection.json

# **Common API Errors & Conventions**

This document lists standard API error formats and conventions across all microservices (Catalog, Ordering, Vendor).

## **Standard Error Response Format**

All services should return structured error responses as JSON.

{  
 "timestamp": "2025-07-15T18:30:00Z",  
 "status": 400,  
 "error": "Bad Request",  
 "message": "Product name must not be empty",  
 "path": "/api/catalog/products"  
}

## **Common HTTP Status Codes**

|  |  |  |
| --- | --- | --- |
| **Code** | **Meaning** | **Typical Use Case** |
| 200 | OK | Successful GET, PUT |
| 201 | Created | Resource successfully created (POST) |
| 204 | No Content | Successful DELETE |
| 400 | Bad Request | Validation or input format errors |
| 401 | Unauthorized | Missing or invalid authentication |
| 403 | Forbidden | Authenticated but no access to resource |
| 404 | Not Found | Resource does not exist |
| 409 | Conflict | Duplicate records / constraint violations |
| 500 | Internal Server Error | Server crash, unexpected issues |

## **Validation Errors (400)**

**Cause**: Bad/missing fields, failed validation rules

{  
 "status": 400,  
 "error": "Bad Request",  
 "message": "Category ID must not be null",  
 "path": "/api/catalog/products"  
}

## **Authentication Errors (401/403)**

**Cause**: No token or wrong role

{  
 "status": 403,  
 "error": "Forbidden",  
 "message": "Access denied: Requires ADMIN role",  
 "path": "/api/vendor/evaluate"  
}

## **Not Found (404)**

**Cause**: Requested ID/resource doesn't exist

{  
 "status": 404,  
 "error": "Not Found",  
 "message": "Order ID 501 not found",  
 "path": "/api/orders/501"  
}

## **Internal Errors (500)**

**Cause**: Unhandled exception or downstream failure

{  
 "status": 500,  
 "error": "Internal Server Error",  
 "message": "Kafka broker unavailable",  
 "path": "/api/orders"  
}

## **Recommendations**

* Always use consistent timestamp, status, error, message, and path fields
* Include input validation in DTOs using javax.validation
* Use global @ControllerAdvice for error handling (Spring Boot)

## **Author**

* Team: Catalog Microservice
* GitHub: [catalog-service-link.txt]

### **13. Live Project Folder Structure (Real Repos)**

Each team's **real development repo** (e.g., ordering-service) should follow this kind of internal structure:

ordering-service/

├── src/

│ ├── main/java/com/yourorg/ordering/

│ ├── resources/

│ │ └── application.yml

├── test/

├── build.gradle / pom.xml

├── README.md

├── scripts/ # Kafka test scripts or DB init

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*\*\*\*