

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

EXPERIMENT-2

Student Name: GEETIKA

UID: 23BCS12885

Branch: CSE

Section/Group: KRG-3A

Semester: 6th

Date of Performance: 15/01/26

Subject Name: System Design

Subject Code: 23CSH-314

1. **Aim:** To design an online E-commerce platform similar to Amazon/Flipkart for browsing and purchasing products like mobiles, laptops, cameras, and clothes. To implement Kafka, Elasticsearch, and a CDC pipeline for real-time data processing, fast search, and scalability.

2. **Objective:**

- To develop a scalable online shopping system for product listing, search, and order management.
- To use Apache Kafka for real-time event streaming and inter-service communication.
- To implement Elasticsearch for fast and efficient product search.
- To integrate a CDC pipeline for real-time synchronization between databases and services.

3. **Tools Required:**

- Programming Language: Java / Python / JavaScript
- Backend Framework: Spring Boot / Express.js / Flask
- Database: MySQL / PostgreSQL / MongoDB
- API Testing Tool: Postman
- Design Tool: Draw.io (for HLD diagrams)
- Web Browser
- ElasticSearch
- Kafka
- CDC connector

4. **SYSTEM DESIGN / SYSTEM SPECIFICATION:**

4.1. Functional Requirements:

- User should be able to search and find the products based on product title or names.
- User should be able to view the details of the product like description, image, available quantity, review, accessed.
- User should be able to select the quantity and move the product/item into the cart.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

- User should be able to make the payment and should be able to perform the check out.
- User should be able to check the status of the order.
- System should be able to manage purchase of items having limited stocks.

4.2. Non-functional Requirements:

- Target Scale: 100 Million DAU with 10 orders processed per second. Availability – System should be available 24/7
- Consistency & Availability: Here for this system we need both as per the Target Scale. Now we should specify that, which part of our system needs what?
- Latency: Required: ~200 ms
- Scaling: Horizontal / Vertical Consistency

4.3. Core-Entities of the System:

1. User / Client
2. Product
3. Cart
4. Orders
5. Checkout followed by Payment

4.4. API Endpoints Creation:

1. GET API Call: Prod_Search

Https://Local_Host/products/search_item = {Search_keywords}

```
HTTP Req {  
  GET: <iPhone 16>  
}
```

```
HTTP Res {  
  <ProductID:iPhone>  
}
```

Now, on front-end if multiple data of respective product is coming in that case the FE becomes faulty -> ultimately increasing the LATENCY.

For that: we can use Pagination 1, 2, 3, 4, ----- SO ON



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

2. GET API Call: View Product Details

https://Local_Host/products/{product_id}

```
HTTP Req {  
GET: <Product_id = 17>  
}
```

```
HTTP Res {  
Product_id:17,  
Name: iPhone 17,  
Color: Navy Blue,  
Price: $1099,  
Image Thumbnail: URL_Image  
}
```

3. POST API Call: Item add in cart

https://Local_Host/cart/add_products

```
HTTP Req {  
Product_id = 17, Product_id = 16  
}
```

```
HTTP Req Header {  
User_id: 04  
}
```

```
HTTP Res {  
Cart_id: 101  
}
```

4. PUT API Call: To update any order in the cart

Let's Suppose you want to add one more product into the cart.

5. DELETE API Call: To remove any item from the cart

Let's Suppose you want to delete one more product into the cart.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

6. POST API Call: for check out & Payment

Https://Local_Host/checkout -> {post body}

```
HTTP REQ {  
  All products ID's,  
  Total Quantity,  
  Total Price  
}
```

```
HTTP RES {  
  Order_ID  
}
```

Https://Local_Host/payment -> {post body}

```
HTTP REQ {  
  Order ID, Payment Type, Payment Mode  
}
```

```
HTTP RES {  
  Confirmation_Status: Succes / Fail  
}
```

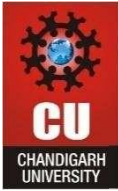
7. GET API Call: Order Status

Https://Local_Host/orde_status = {order_id}

5. Database Schema:

The system is divided into domain-specific schemas to support microservices:

- Identity (MySQL): users table storing credentials and contact info.
- Catalog (MS SQL): products, categories, and sellers tables.
- Media (S3): product_images table linked to image URLs in S3.
- Cart & Inventory (PostgreSQL): cart, cart_items, and inventory (with reserved_quantity).
- Orders & Payment (MySQL): orders, order_items, and payments tables.
- Social: reviews table for user feedback and ratings.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

6. High Level Design (HLD):

The HLD follows a Microservices pattern:

- API Gateway: Acts as a Load Balancer and handles Routing, Rate Limiting, and Authentication (JWT).
- Service Layer: Independent services (User, Search, Product, Cart, Checkout) communicate via the Gateway.
- Persistence Layer: Each service owns its database to prevent a single point of failure.

7. Low Level Design (LLD):

The LLD addresses specific NFR challenges using an Event-Driven Architecture:

- Search Optimization: A CDC (Change Data Capture) Pipeline monitors the Product DB. A Connector service sends changes to a Streaming Buffer (Kafka), which updates ElasticSearch.
- Inventory Logic: Uses an Inventory Consumer to handle Stock update messages from Kafka. It manages reserved_quantity to prevent overselling.
- Payment Flow: Integration with a Third Party Payment Gateway via HTTP. Status updates (Fail/Success) are propagated back to the Order Status Service.

9. Learning Outcomes:

1. Understanding Distributed System Latency
2. Collision Resolution Strategies
3. Architectural Scalability and Reliability
4. Distributed State Management
5. Single Points of Failure (SPOF)