DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

Experiment 2

Student Name: Jetavya Singh Naruka
UID: 23BCS11171
Branch: BE-CSE
Section: KRG-3B
Semester: 6th
Date of performance: 14-01-2026
Subject name: System Design
Subject code: 23CSH-314

1. Aim

   To design a **scalable, highly available, and consistent online shopping (E-commerce) platform** that supports millions of daily users, enables product search, cart management, checkout, payment, and order tracking while handling **high concurrency and race conditions** during flash sales.

2. Objective:
   - Design a **distributed microservices-based E-commerce system**
   - Fulfill all **functional requirements (FRs)** and **non-functional requirements (NFRs)**
   - Handle **100 million DAU** and **10+ orders/sec**
   - Ensure **strong consistency** for critical operations like **payment and inventory**
   - Solve **race conditions** using **Kafka and Inventory locking**
   - Implement **efficient product search** using **Elasticsearch with CDC**

3. System Requirements:

3.1 Functional Requirements

1. User can **search products** by name/title
2. User can **view product details** (price, image, reviews, quantity)
3. User can **add/remove/update items in cart**
4. User can **checkout and make payment**
5. User can **track order status**
6. System must manage **limited inventory** during flash sales

3.2 Non-Functional Requirements

- **Scale**: 100M DAU, 10+ orders/sec
- **Latency**: ~200 ms

- **Availability**:
    1. High availability for **Product Search**
- **Consistency (Strong)**:
    1. Payment processing
    2. Order placement
    3. Inventory management
- **Scalability**: Horizontal scaling (preferred)
- **Fault Tolerance**: Retry, message durability (Kafka)

4. Tools Used:
    - Draw.io Designing system architecture (HLD).
    - PostgreSQL - Relational database design.
    - ElasticSearch Product search and indexing.
    - Apache Kafka - Event streaming.
    - CDC Pipeline - Syncing product data to ElasticSearch.

5. High-Level Architecture (Microservices)

Core Services

- **API Gateway**
    - Routing
    - Rate Limiting
    - Authentication & Authorization (JWT)
- **User Service**
- **Product Service**
- **Search Service**
- **Cart Service**
- **Checkout Service**
- **Payment Service**
- **Order Service**
- **Inventory Service**

Databases

- MySQL (Transactional data)
- Elasticsearch (Search)
- Kafka (Event streaming)

- S3 (Product images)

Core Entities

- User
- Product
- Cart
- Order
- Payment
- Inventory

6. API Design (Key APIs)

6.1 Search Products

```
GET /products/search?query=iPhone16
```

```
Response:
```

```
[
  { "product_id": 17 },
  { "product_id": 18 }
]
```

➡ Pagination used to reduce latency

6.2 View Product Details

```
GET /products/{product_id}
```

```
Response:
```

```
{
  "product_id": 17,
  "name": "iPhone 17",
  "color": "Navy Blue",
  "price": 1099,
  "image_url": "S3_URL"
}
```

6.3 Add Item to Cart

```
POST /cart/add_products

Header: User_ID

Body:

{

  "product_id": 17,

  "quantity": 2

}

Response:

{

  "cart_id": 101

}
```

6.4 Checkout

```
POST /checkout

Body:

{

  "cart_id": 101,

  "total_price": 2198

}

Response:

{

  "order_id": 9001

}
```

6.5 Payment

```
POST /payment

Body:

{

  "order_id": 9001,
```

```
  "payment_mode": "CARD"
}
Response:
{
  "status": "SUCCESS"
}
```

6.6 Order Status

```
GET /order_status/{order_id}
```

7. Search Optimization Using Elasticsearch

Problem

- Searching directly from MySQL is slow (`O(n)`)
- 100M users → DB scanning is not feasible

Solution

- Use **Elasticsearch**
- **Inverted Indexing**
- Tokenization + Multi-term search
- Near `O(1)` performance

8. CDC Pipeline (MySQL → Elasticsearch)

Why CDC?

Elasticsearch is **not a primary DB**, so product data must stay in MySQL.

1. Product DB change (INSERT/UPDATE)
2. **CDC Connector** detects change
3. Event pushed to **Kafka**
4. Search Service consumes event
5. Elasticsearch index updated in real time

9. Inventory & Race Condition Handling

Problem (Flash Sale)

- Multiple replicas read same inventory count
- Concurrent checkout leads to **overselling**

Solution

Inventory Service + Kafka (Producer-Consumer)

Flow

1. Checkout Service sends **Order Event** to Kafka
2. Inventory Service consumes event
3. **Atomic stock decrement** (DB lock / CAS)
4. Inventory updated safely
5. Order status updated

➡ Ensures **strong consistency**

10. Payment Consistency (Critical Section)

Issue

- Payment success but inventory not updated

Solution

- **Event-driven architecture**
- Kafka ensures:
  - No lost messages
  - Retry on failure
  - Exactly-once stock update

11. Scaling Strategy

- **Horizontal Scaling**
- Stateless services
- API Gateway acts as **Load Balancer**
- Kafka enables async processing

- DB sharding on `ProductID / UserID`

12. Output

- Fully functional **E-commerce system design**
- Supports **high traffic & flash sales**
- Strong consistency for payment & inventory
- Low latency product search
- Scalable and fault tolerant architecture

13. Learning Outcome

- Understood **real-world E-commerce system design**
- Learned how to:
    - Handle **race conditions**
    - Use **Kafka for consistency**
    - Implement **CDC with Elasticsearch**
    - Balance **CAP theorem**
- Designed an **interview-ready, production-grade architecture**