# Experiment 2

**Student Name: Tejinderpal Singh** **UID: 23BCS80343**
**Branch: CSE** **Section/Group: KRG 3-B**
**Semester: 6th** **Date of Performance:14/01/2026**
**Subject Name: System Design** **Subject Code: 23CSH-314**

1. **Aim**: To design and analyze a scalable E-commerce platform similar to Amazon / Flipkart that allows users to search products, manage carts, place orders, perform payments, and track order status while ensuring high availability, consistency, scalability, and low latency.

## 2. Objective:

- To identify functional and non-functional requirements of an online shopping system.
- To design a scalable architecture using **microservices**.
- To analyze **CAP theorem trade-offs** for different system components.
- To design **high-level architecture (HLD)** for product search, cart, order, payment, and inventory.
- To understand the role of **Kafka, ElasticSearch, and CDC pipeline** in large-scale systems.
- To handle **race conditions** during flash sales and limited inventory scenarios.

## 3. 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.

## 4. System Requirements:

### A. Functional Requirements:

1) User should be able to **search products** using product name or title.
2) User should be able to **view product details** such as description, images, price, available quantity, and reviews.
3) User should be able to **select quantity** and add products to the **cart**.
4) User should be able to **checkout and make payment** successfully.
5) User should be able to **track order status** after placing an order.
6) System should be able to **manage limited inventory stock** efficiently.

7) System should handle **concurrent transactions during flash sales** without overselling.

### B. Non-Functional Requirements:

1) Scalability
    a. Target scale: 100 million DAU
    b. Order processing rate: ~10 orders/sec
2) Availability & Consistency
    a. High Availability required for:
        i. Product search
        ii. Product listing
    b. Strong Consistency required for:
        i. Payment processing
        ii. Order placement
        iii. Inventory management
3) Latency
    a. System response time should be ≤ 200 ms
4) Reliability
    a. System should tolerate failures and recover automatically.
5) Scalability Type

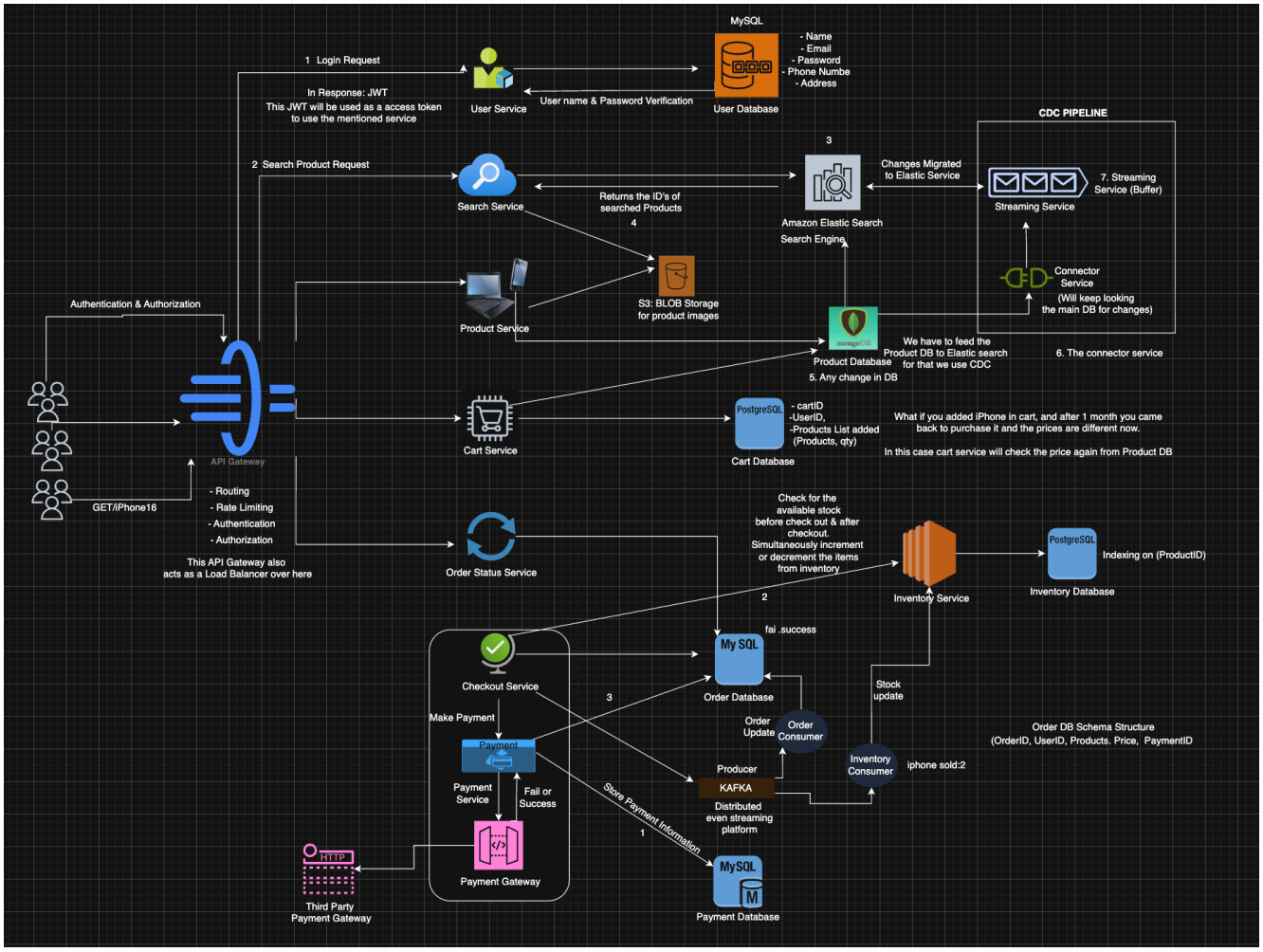
    - Horizontal and vertical scaling supported.

## 5. High Level Design (HLD):

- The system follows a microservices-based architecture:
- API Gateway
    - Handles routing, authentication, authorization, and rate limiting.
- User Service
    - Manages user authentication and profile data.
- Product Service
    - Manages product catalog and metadata.
- Search Service
    - Uses ElasticSearch for fast product search.
- Cart Service
    - Manages user cart and price validation.
- Order Service

- Handles order creation and status tracking.

- Inventory Service
  - Maintains stock levels and prevents overselling.
- Payment Service
  - Handles payment processing via third-party gateway.
- Kafka + CDC Pipeline
  - Syncs product data changes to ElasticSearch.

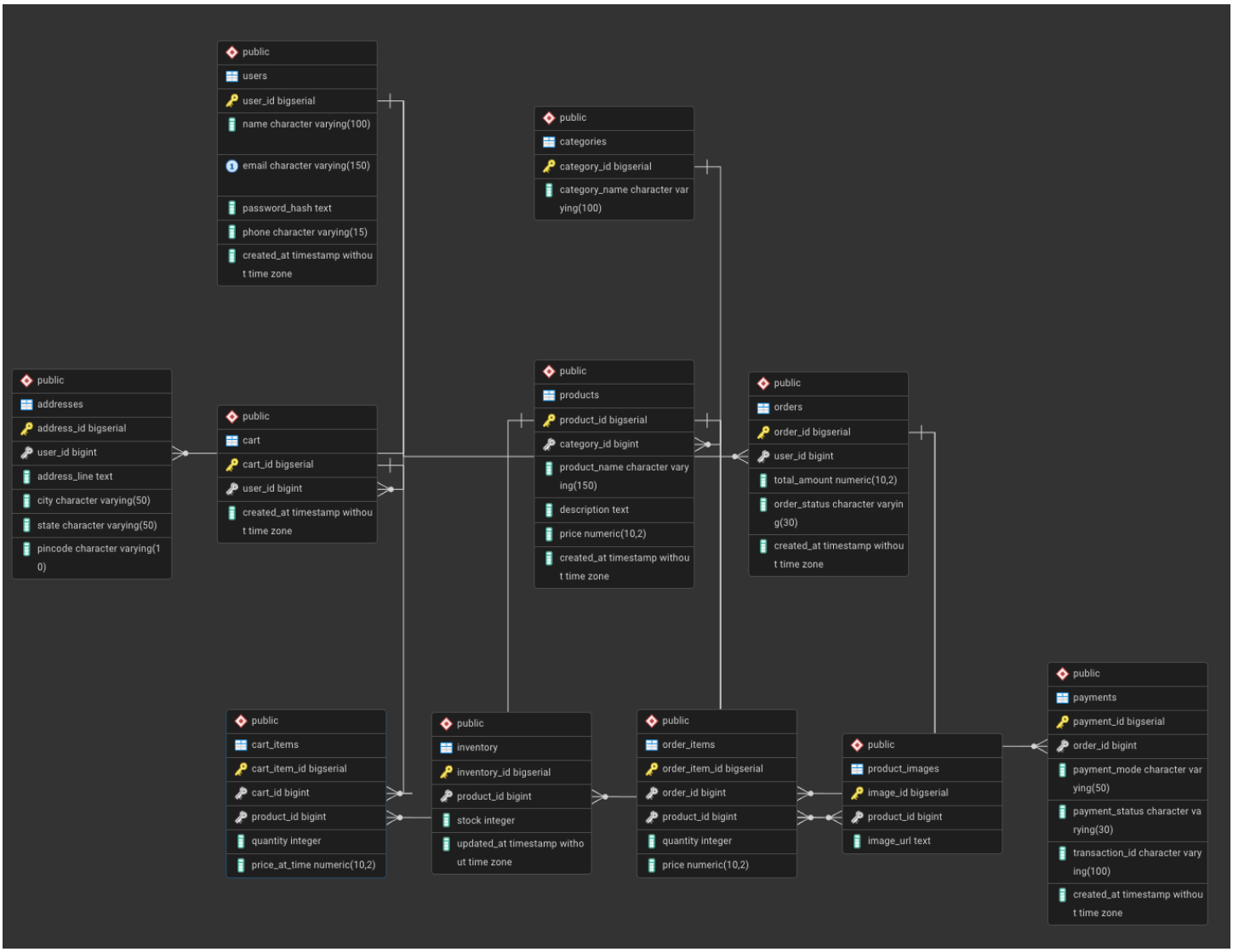



## 6. Database Design (ER Diagram):

- The database for the E-commerce platform is designed using an **Entity Relationship (ER) model**. The ER diagram represents the **core entities, their attributes, and relationships** required to support user management, product catalog, cart handling, order processing, inventory control, and payment management.
- The ER diagram is generated using **PostgreSQL (pgAdmin ERD Tool)** based on tables defined with **primary keys and foreign keys**, ensuring referential integrity.

**Relationships Between Entities:**

- A User can have multiple Addresses (One-to-Many).
- A User can have one Cart (One-to-One).
- A Cart can contain multiple Products through Cart Items (Many-to-Many).
- A User can place multiple Orders (One-to-Many).
- An Order can contain multiple Products through Order Items (Many-to-Many).
- Each Product belongs to one Category (Many-to-One).
- Each Product has one Inventory record (One-to-One).
- An Order has one Payment record (One-to-One).
- A Product can have multiple Images (One-to-Many).



## 7. Low Level Design (LLD):

- Inventory Management (Flash Sale Handling)
- Inventory is updated using atomic operations.
- Stock decrement happens only after successful order confirmation.
- Database transactions and row-level locking are used to prevent race conditions.
- Kafka events ensure eventual synchronization across services.

## 8. Scalability Solution

- Horizontal scaling **of microservices.**
- API Gateway **acts as load balancer.**
- ElasticSearch **used for read-heavy search traffic.**
- Kafka **used for asynchronous communication.**
- CDC pipeline **ensures near real-time data consistency between Product DB and Search index.**

## 9. Learning Outcomes (What I Have Learnt)

- Learned how to design a **real-world scalable E-commerce system**.
- Understood **functional vs non-functional requirements**.
- Gained knowledge of **CAP theorem trade-offs**.
- Learned how **ElasticSearch and Kafka** are used in production systems.
- Understood handling of **concurrency and race conditions**.
- Learned importance of **low latency and high availability** in distributed systems.