Here are **bullet-point notes** for designing an **Amazon-like E-commerce System**, a large-scale, distributed system with many moving parts:

**🧩 1. Problem Statement**

* Design a scalable system for online shopping:
  + Product browsing
  + Search
  + Cart & checkout
  + Order tracking
  + Payment & inventory management

**📎 2. Requirements**

**✅ Functional**

* Product catalog with filtering and search.
* Shopping cart management.
* Order placement, confirmation, and tracking.
* Inventory and stock updates.
* Payment processing.
* User account, wishlists, reviews/ratings.

**❌ Non-Functional**

* High availability, scalability, and fault-tolerance.
* Low latency (especially for search and cart).
* Data consistency (especially for inventory/orders).

**🧠 3. Core Microservices**

* **User Service**: Registration, login, profile, auth (OAuth/JWT).
* **Product Service**: Product info, categories, filters.
* **Search Service**: Full-text search (ElasticSearch), filters.
* **Cart Service**: Temporary items in cart, TTL-based cleanup.
* **Inventory Service**: Stock levels, reservations.
* **Order Service**: Create orders, status tracking.
* **Payment Service**: Process payments, third-party gateways.
* **Notification Service**: Email/SMS confirmations.
* **Review & Rating Service**: User reviews, moderation.

**🗃 4. Database Design**

**Product Table**

* product\_id, name, description, price, category, stock, images, rating

**User Table**

* user\_id, name, email, password\_hash, address, orders

**Order Table**

* order\_id, user\_id, items, status, timestamp, payment\_id, shipping\_info

**Inventory Table**

* product\_id, warehouse\_id, quantity

Use **relational DBs** (Postgres/MySQL) for core transactions, **NoSQL** for product catalogs/search metadata.

**🔍 5. Search and Filtering**

* Use **ElasticSearch** for product indexing.
* Support filters: category, price range, rating, etc.
* Autocomplete suggestions with prefix trees or trie.

**🛒 6. Shopping Cart**

* Temporary storage of product selections.
* Stored in:
  + In-memory store (Redis) for guest users.
  + Persistent DB for logged-in users.
* TTL cleanup for abandoned carts.

**📦 7. Inventory Management**

* Check inventory before confirming orders.
* Reserve stock upon adding to cart or at checkout (based on policy).
* Use **eventual consistency** for replicated stock across warehouses.

**💸 8. Payment Handling**

* Integrate with payment gateways (Stripe, PayPal).
* Handle failures, retries, chargebacks.
* Use **idempotency tokens** to prevent duplicate charges.

**📬 9. Order Lifecycle**

* States: CREATED → PAID → SHIPPED → DELIVERED → RETURNED
* Order events published to message queues (Kafka/SQS) for decoupled processing.

**🛠 10. Architecture Components**

**Frontend**

* Web/mobile apps, React or Flutter.

**Backend**

* REST/gRPC microservices.
* API Gateway for routing & rate limiting.

**Message Queue**

* Kafka, RabbitMQ, or SQS for:
  + Payment confirmations
  + Inventory updates
  + Notifications

**CDN & Caching**

* Serve images and product pages via **CDN** (CloudFront, Akamai).
* Use **Redis** for caching popular products and metadata.

**🔐 11. Security**

* HTTPS everywhere.
* Encrypt sensitive user data.
* Token-based authentication (JWT).
* Role-based access for admin dashboards.

**📊 12. Scalability & Availability**

* Auto-scaling services (Kubernetes, ECS).
* DB sharding and read replicas.
* Caching layers (Redis/Memcached).
* Circuit breakers and retries.

**🎯 13. Advanced Features**

* Product recommendations (collaborative filtering).
* AB testing for UI/UX.
* Fraud detection using machine learning.
* Personalized pricing and offers.