Project: QuickSale – Flash Sale Backend

🎯 Objective

Build a **microservice-based backend system** for a flash-sale scenario. The system must support **secure authentication, order placement without overselling, reliable event processing, and real-time user notifications**.

Core Requirements

1. Authentication Service

- User registration and login endpoints.
- Issue **JWT access & refresh tokens**.
- Token refresh endpoint and logout.
- Access token required for all protected APIs.

2. Catalog & Inventory Service

- Product management: list and add products.
- Stock management: update stock levels.
- **Stock reservation API**: must handle concurrent requests safely (no overselling).
- Emit stock reservation result events to Kafka.

3. Order Service

- `POST /orders` to create new orders.
- Must require an **Idempotency-Key** header.
- If the same key is used again, return the same result (no duplicate orders).
- Implement the **Outbox pattern**: write order + outbox event atomically.
- Dispatch order events from outbox to Kafka.
- Consume inventory results to update order status.

4. Event Flow (via Kafka)

- **order.created** → published by Order service.
- **inventory.result** → published by Inventory service after reservation.
- **order.status** → published by Order service when status changes.

- Other services subscribe and react accordingly.

5. Notifier Service (WebSockets)

- Provide a WebSocket endpoint for clients.
- Authenticate users on connection (JWT).
- Each user joins their own channel/room.
- Consume Kafka **order.status** events and push `order.update` messages to the correct user in real time.

6. API Gateway

- Verify JWT on all incoming requests.
- Apply **rate limiting** (store counters in Redis).
- Forward requests to the appropriate internal service.

7. Database Design (required)

- Candidate must design the schema for:
    - Users & authentication
    - Products & stock
    - Orders
    - Outbox (for reliable events)
    - Notifications
- Schema must support concurrency safety (e.g., optimistic/pessimistic locking).

📝 Deliverables

- Working code for all services.
- README with setup instructions and sample API calls.
- Clear description of the database schema.
- Example flow demonstrating:
    1. User registers & logs in.
    2. User creates an order with Idempotency-Key.
    3. Order events go through Kafka → inventory reserves stock → order status updates.
    4. User receives live order status update via WebSocket.