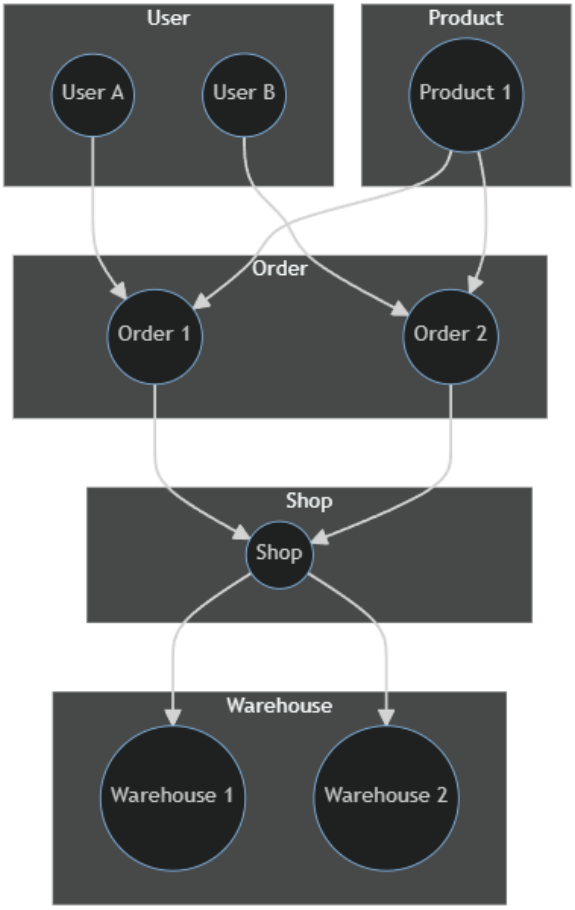
Test Case



1. User Service:
    a. Create simple authentication so user can login using phone or email.
2. Product Service:
    a. API to List Products: Create an API endpoint that allows users to view a list of products along with their stock availability. This endpoint should retrieve product information from a database or storage system.
3. Order Service:
    a. Checkout and Stock Deduction: When a customer places an order, you should implement a checkout process. During checkout, reserve (lock) the stock of the ordered products. If another customer tries to order the same product, the system should check the available stock and deduct the reserved quantity from it. This ensures that stock is not oversold.

b. Release Stock: Implement a mechanism to release the reserved stock if payment is not made within a specified time frame (N minutes). This can be done using a background job or a timer.

4. Shop Service:

a. Shop has one or more warehouse.

5. Warehouse Service:

a. Stock Management
b. Transfer Products: Allow products to be transferred from one warehouse to another. Implement APIs or mechanisms for moving stock between warehouses. Ensure that stock levels are updated accordingly.
c. Active/Inactive Warehouses: Track the status of each warehouse. If a warehouse becomes inactive, make sure that its stock is no longer counted as available. You should have a way to activate or deactivate warehouses.

Common Considerations:

a. Program Language: Go or NodeJS
b. Database: You will need a database to store product information, order details, stock levels, and warehouse data. Choose a suitable database system for your needs.
c. Concurrency and Locking: Implement concurrency control mechanisms to handle multiple users simultaneously accessing and modifying stock levels. This will ensure data consistency.
d. Monitoring and Logging: Implement monitoring and logging to track system activities and errors.
e. Error Handling: Implement robust error handling to gracefully handle failures and exceptions.
f. Testing: Develop comprehensive unit tests, integration tests, and end-to-end tests to ensure the reliability of your services.
g. Scaling: Consider how your system can scale horizontally to handle increased traffic and data volumes.
h. Deployment and Containerization: Decide how you will deploy your services, whether in containers (e.g., Docker) or on virtual machines, and manage them using container orchestration tools (e.g., Kubernetes).

Remember that building a complete e-commerce and stock management system is a complex task that requires careful planning, development, and testing. Additionally, you may want to consider using a microservices architecture to keep your services modular and maintainable.