Requirement Document: Concurrent Order Processing System

Objective

Build a concurrent, in-memory order processing system in Go. The system should support safe concurrent operations, allow order submission and querying, and simulate real-time order processing.

Key Features

1. Submit new customer orders.
2. Retrieve orders by time range.
3. Concurrently process orders (simulate shipping with delay).
4. Track status of each order (pending, processing, shipped).
5. Thread-safe data access.
6. Unit-test coverage for core logic.

Data Model

```
type Item struct {
    Name     string `json:"name"`
    Quantity int    `json:"quantity"`
}

type Order struct {
    ID        string    `json:"id"`
    Customer  string    `json:"customer"`
    Items     []Item    `json:"items"`
    Timestamp time.Time `json:"timestamp"`
    Status    string    `json:"status"` // pending, processing, shipped
}
```

Requirement Document: Concurrent Order Processing System

API Endpoints

1. POST /orders

Create a new order

Request Body:

```
{
  "customer": "John Doe",
  "items": [
    {"name": "Monster Energy", "quantity": 2},
    {"name": "Protein Bar", "quantity": 1}
  ]
}
```

Response:

```
{
  "id": "order_12345",
  "status": "pending"
}
```

2. GET /orders

Retrieve all orders

Optional Query Parameters: from, to (ISO timestamp)

Response: List of orders with metadata.

Requirement Document: Concurrent Order Processing System

3. GET /orders/{id}

Retrieve a single order by ID

Response: Order details.

4. POST /orders/process

Manually trigger processing of all pending orders.

Response: { "message": "Processing started" }

5. GET /orders/status/{id}

Check the status of a specific order

Response: { "id": "order_12345", "status": "processing" }

6. DELETE /orders/{id}

Delete a specific order (only if pending)

Response: { "message": "Order deleted" }

Concurrency Requirements

- Use sync.Mutex or sync.RWMutex for shared data access.

- Order processing must run in separate goroutines.

- Ensure race-free read/write access.

- Track status updates safely.

Requirement Document: Concurrent Order Processing System

Bonus Features (Optional)

- Order cancellation: Allow cancellation if not yet shipped.

- Timeout handling: Cancel processing if an order takes too long (context.WithTimeout).

- Graceful shutdown: Handle safe termination of processing.