Assignment: Inventory KPI Calculation System

Subjects: Systems Programming – Jan 2026

Minor: For all bachelor programs

1. Assignment Overview

In this assignment, you will design and implement a Key Performance Indicator (KPI) calculation system for an Inventory Management System.

The system processes JSON files containing invoices and purchase orders, performs data analysis, and computes key inventory performance metrics.

The goal of this assignment is to help you apply file processing, data modeling, memory-efficient coding, and asynchronous programming in a real-world backend scenario.

2. Learning Objectives

After completing this assignment, you should be able to:

- Read and parse structured JSON files in C#
- Model real-world business entities using classes and structs
- Process large datasets efficiently
- Apply memory and performance best practices
- Use LINQ, async/await, and Tasks correctly
- Calculate and interpret inventory KPIs
- Design clean, maintainable, and extensible code

3. Business Scenario

A company stores its sales invoices and purchase orders as JSON files exported from different systems.

Management wants a program that can analyze these files and compute inventory KPIs to evaluate stock efficiency and operational performance.

Your task is to build a console application or class library that reads these files and generates KPI results.

4. Input Data

You are provided with two types of JSON files:

4.1 Purchase Orders (purchase_orders.json): Each purchase order contains: Order ID, Product ID, Quantity purchased, Unit cost, Purchase date.

4.2 Sales Invoices (invoices.json): Each invoice contains: Invoice ID, Product ID, Quantity sold, Unit selling price, Invoice date

The files may contain thousands of records, so efficiency matters.

5. Required KPIs

Your system must calculate the following inventory KPIs based on invoice and purchase order data.

1. Total SKUs

Description: The total number of unique products managed by the inventory system.

Formula: Total SKUs = Count of distinct ProductId

Data Source: Purchase Orders + Sales Invoices

2. Cost of Inventory (Stock Value)

Description: The total monetary value of current inventory on hand.

Formula: Stock Value = Σ (Unsold Quantity × Unit Cost)

Data Source: Purchase Orders (unit cost) + Sales Invoices (quantity sold)

3. Out-of-Stock Items

Description: The number of products with zero or negative inventory balance.

Formula: Out-of-Stock Items = Count of SKUs where (Purchased − Sold) ≤ 0

Data Source: Purchase Orders + Sales Invoices

4. Average Daily Sales

Description: The average quantity of items sold per day over the analysis period.

Formula: Average Daily Sales = Total Quantity Sold / Number of Sales Days

Data Source: Sales Invoices (invoice date, quantity)

5. Average Inventory Age

Description: The average number of days inventory items remain unsold.

Formula: Inventory Age = Average(Current Date − Purchase Date of Unsold Items)

Data Source: Purchase Orders + Sales Invoices

Assessment Emphasis

- KPIs may be calculated **per SKU** and **system-wide**
- Correct interpretation of KPI definitions
- Accurate aggregation and calculations
- Clear handling of edge cases (zero sales, missing data)
- Clean and extensible KPI calculation design

6. Real-Time / Near Real-Time Data Processing

To simulate a real-world inventory system, the application must support **near real-time processing of data files** that are **automatically sent to the system every hour**.

Business Requirement

The inventory system receives new invoice and purchase order JSON files every hour from external systems (e.g., POS systems, ERP systems, suppliers).

The application must detect, process, and incorporate these files into KPI calculations without restarting the system.

7. Functional Requirements

Your application must:

- Load JSON files from disk
- Deserialize data into strongly typed C# models
- Handle invalid or missing data gracefully
- Calculate KPIs per product
- Output results in a clear, readable format (console table or JSON report)
- Use asynchronous file I/O where appropriate
- Monitor one or more input directories for new data files
- Automatically detect newly added JSON files
- Process files as soon as they arrive (hourly batches)
- Update KPI calculations incrementally
- Prevent duplicate processing of the same file
- Continue running reliably over long periods

8. Technical Requirements

Language: C#

Platform: .NET 8 or later

Data format: JSON

1. File Monitoring

Use FileSystemWatcher to monitor directories:

- /data/invoices/
- /data/purchase-orders/

Trigger processing when new files are created or renamed

2. Asynchronous Background Processing

Process incoming files using background tasks

Avoid blocking the main application thread

Use Task, async/await, and a producer–consumer pattern

3. Incremental KPI Updates

Do not reprocess all historical data every hour

Update KPIs only with newly received records

Maintain in-memory aggregates or persisted snapshots

4. Idempotency and Duplicate Detection

Track processed files using:

File name + checksum, or

A processed-files registry (JSON or lightweight database)

Ensure each file is processed exactly once

5. Fault Tolerance

Handle partial or corrupted files gracefully

Retry failed file processing

Log errors without stopping the system

9. Non-Functional Requirements

Code must be readable and well-structured

Classes and methods must have meaningful names

No hard-coded business logic values

The solution must be easy to extend with new KPIs

The system must run continuously for long periods

Memory usage must remain stable over time

File processing must be thread-safe

KPI calculations must remain consistent under concurrent updates

10. Deliverables

Students must submit:

1. Source code project (.sln)

2. KPI files in JSON format
3. A short report presentation (4 pages – 15 minutes) describing:
    - KPIs implemented
    - Design decisions
    - Performance considerations
    - Assumptions made

11. Marking Criteria (Example)

Component	Weight
Correct KPI calculations	30%
Code structure and design	25%
Performance and memory efficiency	15%
Error handling and robustness	10%
Use of async/await and LINQ	10%
Documentation and report	10%

12. Extension (Bonus – Optional)

Support multiple input files

Add date-range filtering for KPIs

Use Span<T> or streaming deserialization for large files