Middleware Architecture

(IS3108/SCS3203)

Assignment 4 — Middleware Architecture for "SwiftLogistics"

(Release date: 06 August 2025, Deadline: 06 September 2025)

1. Objective

The primary objective is to design and implement a technology solution based on the knowledge and understanding of middleware architectures and related technologies to realise a prototype solution for a given business scenario.

2. Introduction

This assignment is designed to be a **collaborative group effort**. Its primary objective is to provide a hands-on opportunity to apply the theoretical knowledge of middleware architecture principles and related technologies to a real-world business scenario. Working in a team will allow you to simulate the dynamics of a professional software development environment, where collective problem-solving and the integration of diverse ideas are essential for success.

All teams will be given the same detailed business scenario. Your task is to:

1. **Design a robust and scalable middleware architecture** to solve the business problem.
2. **Implement a functional prototype** of the core components of your proposed solution.

The entire solution—from design to implementation—must be based on **open-source technologies**. This is a deliberate constraint to encourage you to explore the rich ecosystem of open-source tools that are frequently used in the industry. As you will discover, a wide range of open-source tools and frameworks are available to build the essential components of a modern middleware solution, including:

- **Service Communication:** Frameworks for creating SOAP and RESTful services (e.g., Spring Boot, Flask, FastAPI, Node.js Express).
- **Service Orchestration:** Enterprise Service Bus (ESB) technologies or microservices frameworks for managing complex business processes and data flows.
- **Asynchronous Communication:** Publish-subscribe message brokers for high-volume, asynchronous messaging (e.g., RabbitMQ, Apache Kafka).
- **Service Discovery and Registry:** Tools for managing and locating services in a distributed environment.
- **Data Persistence:** Database systems (e.g., PostgreSQL, MongoDB) for storing application data.

Each team is encouraged to research and select the most suitable combination of these technologies to realise an effective and well-justified architecture for the given business scenario. The freedom to choose your technology stack is a key part of the assignment, allowing you to demonstrate your critical thinking and architectural decision-making skills.

3. Duration and Team Composition

The assignment duration is 4 weeks (deadline: 6 September 2025) and should be completed in teams of **8–9 students**.

4. Submission Guidelines

The submission should comply with the following guidelines. The source code, documentation and the presentation screencast should be uploaded to the VLE.

1. The **Solution Documentation** should be submitted in PDF format. The index number of each group member who contributed should be stated in the documentation. A

comprehensive report detailing the proposed solution/middleware architecture. This should include architectural diagrams, a justification for technology choices (e.g., why a specific middleware component is used), and a discussion of how each of the challenges is addressed.

2. The **Source Code** of the solution should be compressed and submitted together with the project documentation.

3. **Presentation of the solution:** An overall presentation (less than 10 minutes) of the proposed solution architecture and technology should be recorded as a screencast and uploaded to the VLE. The presentation should primarily focus on a detailed description of the solution architecture, the rationale for combining the architectural patterns to solve the overall solution, and a brief functional demo. The screencast presentation should be voiced/narrated by the team members.

5. Evaluation Guidelines

The evaluation of the assignment would be done based on the following criteria.

1. **Evaluation of the solution documentation – The documentation should cover the following aspects.**

    a. Introduction to the solution.

    b. **The architecture of the solution** — conceptual and implementation architectures to be drawn and discussed. Propose 2 alternative architectures that could be used to implement the solution and state the rationale for using the proposed final architecture for the implementation.

    c. **The Architectural Patterns used in the solution**. You should describe the architectural and integration patterns used to connect the components of the overall architecture. Describe the rationale for selecting the proposed Architectural and Integration Patterns.

d. **Prototype**: A minimal implementation that mocks the functionalities of the CMS, ROS, and WMS as described in section 6. The prototype should demonstrate a basic order submission flow, showing how the architecture orchestrates the communication between the three mock systems and provides real-time updates to a simple client UI. The implementation of the real-time tracking and notification system should be architecturally described, but only minimally implemented.

e. List the information security considerations taken during the design and implementation of the solution.

2. Evaluation of the solution through a group presentation.

a. The group members should use a voiced/narrated screencast video to describe the aspects described in section (1) above.

b. Each member of the group should participate in the presentation.

6. Assignment Scenario: Middleware Architecture for "SwiftLogistics"

Swift Logistics (Pvt) Ltd. is a rapidly growing logistics company in Sri Lanka, specialising in last-mile delivery for e-commerce businesses. They offer delivery services for a range of clients, from large online retailers to small-scale independent sellers. To scale their operations and stay competitive, they are replacing their siloed, manual systems with a modern, integrated platform. The new platform is a web-based portal and a mobile application for drivers, both branded as "SwiftTrack."

The core challenge is to integrate three critical systems that currently operate independently, seamlessly:

1. **Client Management System (CMS):** A legacy, on-premise system that handles client contracts, billing, and order intake. It exposes a SOAP-based XML API.

2. **Route Optimisation System (ROS):** A modern, cloud-based service from a third-party vendor that takes delivery addresses and vehicle availability to generate the most efficient routes. It has a well-documented RESTful API.

3. **Warehouse Management System (WMS):** A system that tracks packages within the warehouse, from the moment they are received from a client to when they are loaded onto a vehicle. It offers a proprietary messaging protocol over TCP/IP for real-time updates.

Functional Requirements for "SwiftTrack"

The new "SwiftTrack" platform needs to provide the following functionalities:

- **Client Portal:** E-commerce clients should be able to log in, view the status of their submitted orders, and track deliveries in real-time.
- **Driver Mobile App:** Drivers should be able to:
    - View their assigned delivery manifest and optimised route for the day.
    - Receive real-time updates on route changes or new high-priority deliveries.
    - Mark packages as "delivered" or "failed" with reasons (e.g., recipient not available).
    - Capture a customer's digital signature or take a photo as proof of delivery.
- **Order Intake and Processing:**
    - Clients submit orders through the portal, which are then passed to the WMS for processing and to the ROS for route planning.
    - The system must handle high volumes of incoming orders, especially during promotional events like "Black Friday" or "Avurudu" sales.
    - The architecture must ensure that an order is never lost during the process, even if one of the back-end systems is temporarily unavailable.

Specific Architectural Challenges & Considerations

1. **Heterogeneous Systems Integration:** The proposed architecture must bridge the gap between the CMS (SOAP/XML), the ROS (REST/JSON), and the WMS (proprietary TCP/IP messaging). This requires a robust mechanism for protocol and data format translation.

2. **Real-time Tracking and Notifications:** The system needs to provide real-time updates to clients and drivers. For instance, when a driver marks a package as delivered, the client portal should reflect this change immediately. This also includes push notifications to drivers for urgent route changes.

3. **High-Volume, Asynchronous Processing:** Order processing, especially for a large number of orders, can be time-consuming. The system should not block the client portal while waiting for the ROS to optimise a route or for the WMS to confirm a package is ready. The solution should use an architectural pattern that supports this high-volume, asynchronous communication.

4. **Transaction Management:** A single client order triggers multiple actions across different systems (e.g., creating an order in CMS, adding a package to WMS, and adding a delivery point to ROS). The architecture must ensure the consistency of this distributed transaction. Propose the methods the system would use to recover when one or more steps in a transaction fail.

5. **Scalability and Resilience:** The proposed architecture must be able to scale to handle an increasing number of clients, drivers, and daily deliveries. It should also be resilient to the failure of any single component.

6. **Security:** All communication, especially between the client portal and the backend, and with external APIs like the ROS, must be secure.