Murali Assigned Topics Notes Retail Integration

SAP, MuleSoft, DataWeave & Domain Logic

API-Led

SAP

Retail

Levi's Integration Support-M

Prepared By 2395498- Cognizant

Contents

Chapter 1

The Retail Ecosystem: Buy, Move, Sell

1.1 The Core Lifecycle

Imagine a retail business as a living organism. It breathes in products and breathes out sales. This cycle is often summarized as **Buy, Move, Sell**.

The "Buy-Move-Sell" Paradigm

- **BUY (Procurement):** This is where it starts. The business realizes, "Hey, we're out of jeans!" They create a *Purchase Order (PO)* to a vendor.
- **MOVE (Logistics):** The vendor ships the jeans. They arrive at a warehouse. We have to *Receive* them and *Put them away*.
- **SELL (Omnichannel):** A customer buys those jeans. Maybe in a store, maybe online, maybe on Instagram.

1.2 The "Brain" and The "Muscle"

Two systems dominate the retail landscape. You must know these to pass the interview.

OMS: The Brain (Order Management System)

The OMS doesn't physically hold any products. It holds *information*.

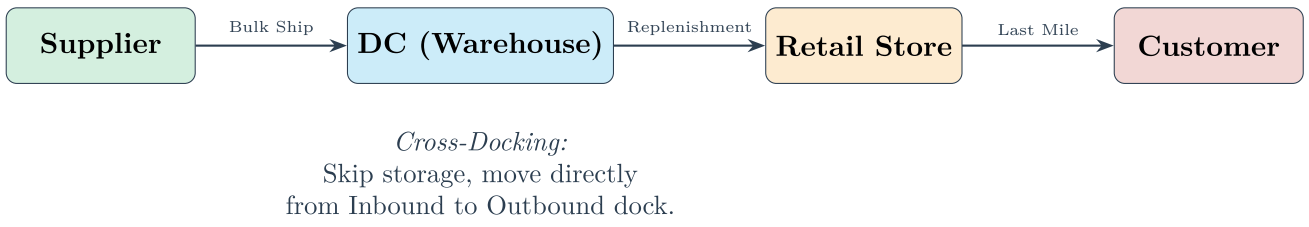
- **Orchestration:** If I buy a shirt online, the OMS decides: "Should I ship this from the New York warehouse or the Boston store?" It picks the cheapest/fastest route.
- **Global Inventory:** It sees stock everywhere.

WMS: The Muscle (Warehouse Management System)

The WMS controls the physical building.

- **Inbound:** ASN (Advance Ship Notice) $\rightarrow$ Receiving $\rightarrow$ Put-away.
- **Outbound:** Wave Planning $\rightarrow$ Picking $\rightarrow$ Packing $\rightarrow$ Shipping.

1.3 Visualizing the Distribution Model



1.4 Critical Terminology (Cheat Sheet)

- **SKU (Stock Keeping Unit):** The internal ID for a product (e.g., "LEVIS-JEAN-BLUE-32").
- **UPC (Universal Product Code):** The barcode scanned at the register. Universal across all retailers.
- **Dropshipping:** Selling a product you don't own. The supplier ships it directly to your customer.
- **BOPIS:** Buy Online, Pickup In-Store. A classic Omnichannel use case.
- **Safety Stock:** A buffer amount kept to prevent running out (stockouts).
- **Reverse Logistics:** The messy business of returns. It costs money to bring items back!

Chapter 2

SAP Business Processes: OTC & PTP

This is the "Hard" skill set. Integration Developers are paid to connect systems to SAP.

2.1 OTC: Order to Cash (Money In)

This is the sales cycle. We want to get paid.

1. **Sales Order (VA01):** A customer wants goods. We confirm the order.
2. **Delivery (VL01N):** We tell the warehouse to prepare the shipment.
3. **PGI (Post Goods Issue) (VL02N):** **CRITICAL STEP!** This is the moment inventory physically leaves the building. Ownership transfers to the customer. Accounts are updated.
4. **Billing (VF01):** We generate the invoice.
5. **Cash:** The customer pays.

2.2 PTP: Procure to Pay (Money Out)

This is the buying cycle. We need to restock.

1. **Purchase Requisition:** "Hey boss, we need laptops."
2. **Purchase Order (ME21N):** The legal contract sent to the Vendor.
3. **Goods Receipt (MIGO):** The truck arrives. We count the boxes. Inventory goes UP.
4. **Invoice Verification (MIRO):** The vendor asks for money. We match the Invoice against the PO and the Goods Receipt (3-Way Match).

2.3 SAP CAR: The Speed Demon

Why do we need SAP CAR?

Problem: SAP ERP (S/4HANA) is big and slow. If you have 1,000 stores sending sales data every minute, ERP will crash.
Solution: SAP CAR (Customer Activity Repository).

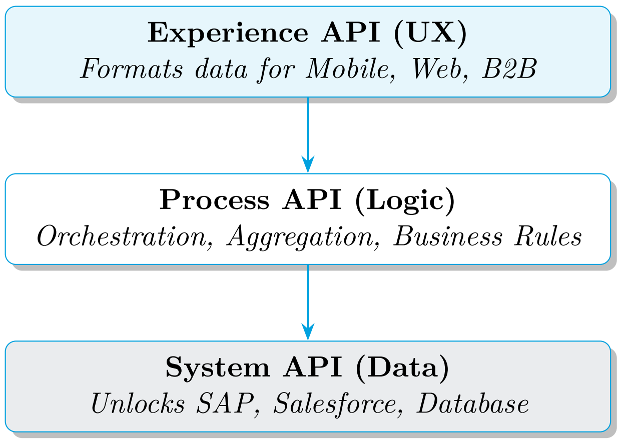
- It sits in the middle.
- It ingests **TLOGs** (Transaction Logs) from POS systems in real-time.
- It calculates **OAA** (Omnichannel Article Availability).
- *Analogy:* SAP ERP is the library archive. SAP CAR is the Twitter feed.

Chapter 3

MuleSoft Architecture & Messaging

3.1 API-Led Connectivity

The 3-layer cake. Never forget this.


Experience API (UX)
Formats data for Mobile, Web, B2B
Process API (Logic)
Orchestration, Aggregation, Business Rules
System API (Data)
Unlocks SAP, Salesforce, Database

Governance: Exp API should NEVER call System API directly. It creates tight coupling!

3.2 Messaging: The Nervous System

Queue vs. Exchange (Anypoint MQ)

- **Queue (Point-to-Point):** Imagine a line at a bank teller. One message goes to One receiver. Used for *Load Balancing*.
- **Exchange (Pub/Sub):** Imagine a radio broadcast. One message goes to Many queues. Used for *Broadcasting* (e.g., New Employee → create Badge AND create Email).

Scenario: Black Friday "Store and Forward"

The Problem: SAP goes down on Black Friday because of too many orders.
The Pattern:

1. Experience API accepts the order immediately. Returns "200 OK" to customer.
2. API puts the order into an **Anypoint MQ Queue**.
3. Customer goes away happy.
4. Process API slowly pulls messages from the Queue and sends them to SAP.
5. If SAP is down, the message stays in the Queue (safe!). It doesn't get lost.

3.3 Transaction Tracking

How do we find a lost order?

- **Correlation ID:** A unique ID generated at the start.
- Passed in the **Header** (e.g., `X-Correlation-ID`).
- Logged using **MDC** (Mapped Diagnostic Context) so it appears in every log line automatically.

Chapter 4

DataWeave Mastery

4.1 The Toolkit

Essential Operators

```
// 1. Map: Transform a list
payload map (item, index) -> {
    id: index,
    name: upper(item.name)
}

// 2. Filter: Select specific items
payload filter ($.price > 100)

// 3. Flatten: Turn nested arrays into one
flatten([[1,2], [3,4]]) // Output: [1, 2, 3, 4]

// 4. DistinctBy: Remove duplicates
payload distinctBy ($.id)

// 5. Default: Handle Nulls (Safety)
payload.address default "No Address Provided"
```

4.2 Complex Scenario: XML to JSON & Grouping

Scenario: You have a list of sales. Group them by Category.

The Solution

```
%dw 2.0
output application/json
var sales = [
    {category: "Electronics", item: "Mouse"},
    {category: "Clothing", item: "Shirt"},
    {category: "Electronics", item: "Keyboard"}
]
---
// GroupBy creates an Object where Keys are the category
sales groupBy ($.category)

/* OUTPUT:
{
  "Electronics": [ {item: "Mouse"}, {item: "Keyboard"} ],
  "Clothing": [ {item: "Shirt"} ]
}
*/
```

4.3 The Reduce Operator

Think of `reduce` as a snowball rolling down a hill, collecting value.

Calculating Total Price

```
payload reduce ((item, accumulator = 0) -> accumulator + item.price)
```