CASE STUDY 02 OF 04 | ONE SALE | B2B FOOD ORDERING PLATFORM

Multi-Customer Bulk Order Management

How agents place, manage, and track bulk orders across an entire customer portfolio

#BulkOrdering #OrderManagement #B2BCart #CustomerOrders

Overview

The core operational task in One Sale is placing bulk food orders for multiple customers in the course of a single working day. An agent might visit four shops in a morning and need to place four separate bulk orders, each with different product selections and quantities, all tracked against the right customer account and all needing to be reviewed later to verify delivery and payment.

This case study covers the ordering system from the agent's perspective: how bulk carts are built, how orders are placed across multiple customers efficiently, and how the order history is structured to give agents a clear record of what was ordered for whom and when.

Project Snapshot

Project	One Sale — B2B Food Ordering Platform
Feature Area	Multi-Customer Bulk Order Placement and Management
Role	Senior Fullstack Engineer
Order Pattern	Multiple bulk orders per day, different customers, same agent
Cart Scope	Per-customer cart, persisted between sessions until ordered
Outcome	Efficient bulk ordering workflow handling high daily order volumes per agent

The Problem

Ordering for multiple B2B customers in a single day is operationally demanding. Each customer has different requirements: different products, different quantities, different delivery windows. The ordering workflow needs to make it easy to handle all of these without mixing up which order belongs to which customer, and without requiring the agent to start from scratch for customers who reorder similar products each week.

Consumer cart systems are not built for this pattern. They assume a single cart, a single buyer, and a single checkout event per session. One Sale needed a cart model that could hold multiple active carts simultaneously, one per customer, persisted across sessions, and convertible to orders independently.

What Was Built

Per-Customer Persistent Cart

Every customer account has its own cart. When an agent selects a customer and starts adding items, those items go into that customer's cart, not a generic session cart. The cart persists until the agent places the order. An agent can work on the cart for Customer A, switch to Customer B, build their cart, then come back to Customer A's cart later and continue where they left off.

This means an agent can accumulate carts across multiple customers throughout a field day and batch the actual order placement at the end of the day when they review what was added. The cart becomes a working order draft that matches the rhythm of field work rather than forcing an immediate checkout decision.

Bulk Cart Building

Adding items to a B2B cart is designed for volume. Quantity fields in the cart accept direct typed input for large numbers without requiring repeated increment taps. A quick-add panel lets agents search for a product and enter the quantity in a two-step flow without navigating to the product detail page. For repeat customers, the agent can use the customer's last order as a starting template and adjust quantities from there, which covers the very common pattern of a shop re-ordering the same selection each week with minor variations.

Wishlist to Cart Conversion for B2B

Wishlists built during field visits convert to cart items in bulk. When an agent has captured a customer's needs in the wishlist during a shop visit, they can move the entire wishlist to the cart for that customer with a single action, setting quantities for each item in one step. This converts field-captured notes into a structured order without requiring the agent to re-navigate the catalogue.

Order Submission and Confirmation

Submitting an order from the cart goes through a confirmation step that surfaces:

1. The customer the order is being placed for, shown prominently
2. The complete line-item list with quantities and unit pricing
3. The total order value before and after any applicable bulk discounts
4. The requested delivery address and any delivery notes
5. A final confirm button that submits the order and locks the cart

After submission, the agent receives an order confirmation with a reference number. The customer's cart is cleared and a new empty cart is initialized, ready for the next ordering cycle. The placed order is immediately visible in the agent's order history under that customer.

Order History per Customer and Across the Portfolio

The agent's order history is organized at two levels. At the customer level, the agent can pull up every order ever placed for a specific shop, with dates, items, quantities, and order values. At the

portfolio level, the agent has a consolidated view of all orders across all their customers, filterable by date range, order status, and product category.

This dual-level history is what makes one of the most common agent workflows possible: reviewing what a specific shop ordered last time before placing this week's order, without having to leave the ordering context to look it up elsewhere.

Technical Architecture

Per-Customer Cart Data Model Carts are stored as database records with a composite key of (agent_id, customer_id). A given agent-customer pair always has exactly one active cart. Items within the cart are child records referencing the cart ID, product ID, and quantity.	**Last-Order Template Generation** The 'use last order as template' feature queries the most recent confirmed order for the agent-customer pair and generates a new set of cart item records from it. The agent then modifies quantities before submitting. The original order is never mutated.
Bulk Discount Computation at Cart Level Pricing is computed at cart-load and on each quantity change. For each line item, the pricing engine checks applicable bulk tier thresholds and applies the correct unit price. The total and line subtotals update in real time as quantities change.	**Atomic Order Submission** Order submission is wrapped in a database transaction: stock reservation, order record creation, cart clearance, and confirmation event emission all happen atomically. A failure at any step rolls the entire submission back, preventing partial order states.
Order History Indexing Order history queries are indexed on (agent_id, customer_id, created_at) for customer-level views and on (agent_id, created_at) for portfolio-level views. Both query patterns are covered by the same composite index with different leading columns.	**Cart Draft Notifications** If a customer's cart has been inactive for a configurable period without being converted to an order, the agent receives a reminder notification. This surfaces forgotten draft carts before they become stale without requiring the agent to manually track their open carts.

Key Challenges

Handling the 'Same Items, Different Customer' Pattern Safely

Agents often order similar or identical product selections for multiple customers on the same day. This creates a risk of the agent mistakenly applying the right products to the wrong customer's cart. Beyond the UI enforcement described in CS-01, the ordering system logs a customer context token with each cart mutation, so any cart action that occurs outside the expected customer context is flagged in the audit log. This provides a detection mechanism for systematic attribution errors without blocking the normal workflow.

Re-Order Workflow Performance

Loading a previous order as a template involves querying the last order's line items, checking current stock availability for each, flagging any items that are now out of stock, and applying current pricing. For orders with 30 or more line items, doing this synchronously on button click produced a

noticeable wait. The solution was to precompute the last-order template in the background when the agent opens a customer's profile, so by the time they tap 'use last order', the data is already ready.

Agents Working Offline or on Poor Connectivity

Field agents in warehouses or rural areas sometimes operate on minimal connectivity. Cart additions need to feel immediate even when the network request is slow. A local-first cart update model was implemented: the cart UI updates immediately based on the local action, and the server sync happens asynchronously. If the sync fails, the item is marked as pending-sync and retried on next connectivity, with a visible indicator that the cart has unsynced changes.

Outcome

The per-customer persistent cart model matched the actual rhythm of B2B field sales work in a way that a single-session cart never could. Agents could accumulate orders throughout a field day and review them before submitting, reducing errors and making the end-of-day order confirmation a natural checkpoint rather than a disruptive interruption. The last-order template workflow significantly cut the time required to place repeat orders for established customers. Portfolio-level order history gave agents the visibility they needed to manage their customer relationships proactively.

Engineering Highlights

Per-customer persistent cart model matching the multi-shop field agent workflow
Last-order template with background precomputation keeping re-order initiation instant
Atomic order submission preventing partial order states on payment or stock failures
Local-first cart updates keeping the UI responsive on degraded mobile connections