Phase 5: Apex Programming

360-Degree Restaurant Lifecycle Management Platform

1. Introduction

The fifth phase of this project focuses on advanced Apex programming for the Restaurant Lifecycle Management System. In the earlier phases, we identified requirements, analyzed the industry, and built the foundational architecture of the application. By this stage, we are implementing the logic that makes the system dynamic, automated, and capable of handling real-world scenarios.

This includes the development of triggers, handler classes, asynchronous processes, batch jobs, exception handling mechanisms, and test classes. The intention is to ensure that the system is not only functional but also scalable, resilient, and capable of running smoothly in production environments where reliability is critical.

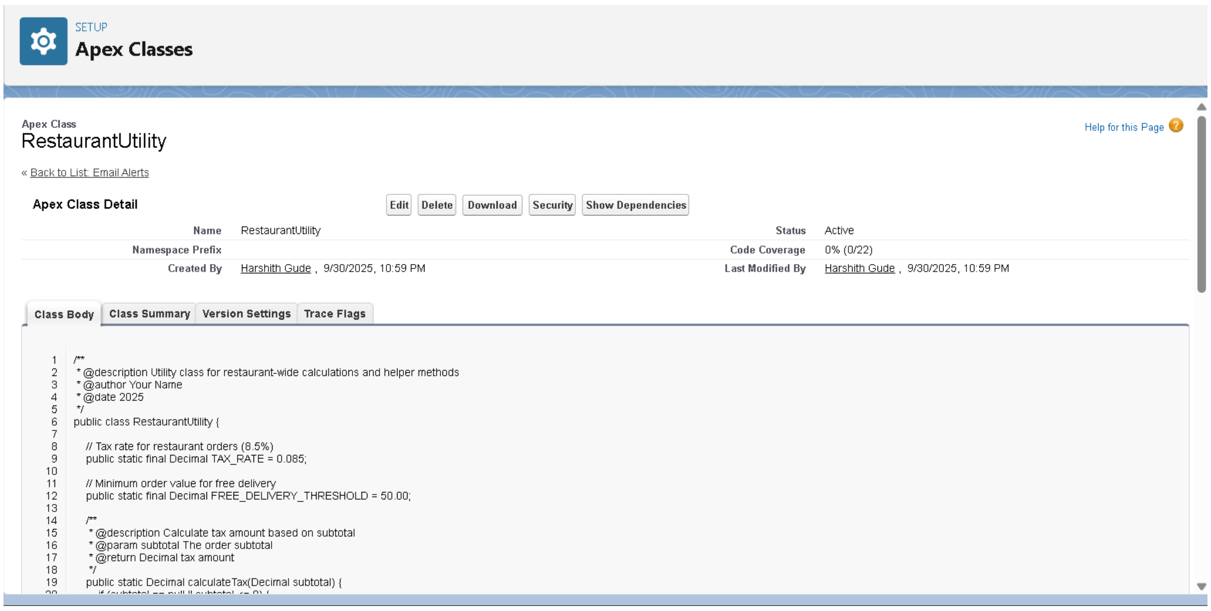
2. Apex Classes

Apex classes form the backbone of the platform's logic. Multiple classes were implemented to handle order processing, recipe costing, inventory management, reporting, and error handling. Each class has been designed with separation of concerns in mind, which allows the system to be modular and easier to maintain.

The main classes developed in this phase include:

- Utility classes for performing calculations such as totals, margins, and stock checks.

- Trigger handler classes that encapsulate business logic and prevent clutter in the trigger files.
- Batch Apex classes for processing large datasets such as expired stock and sales reports.
- Asynchronous classes to handle heavy operations without impacting user performance.
- Custom exception handler classes to ensure data integrity and user-friendly error messages.

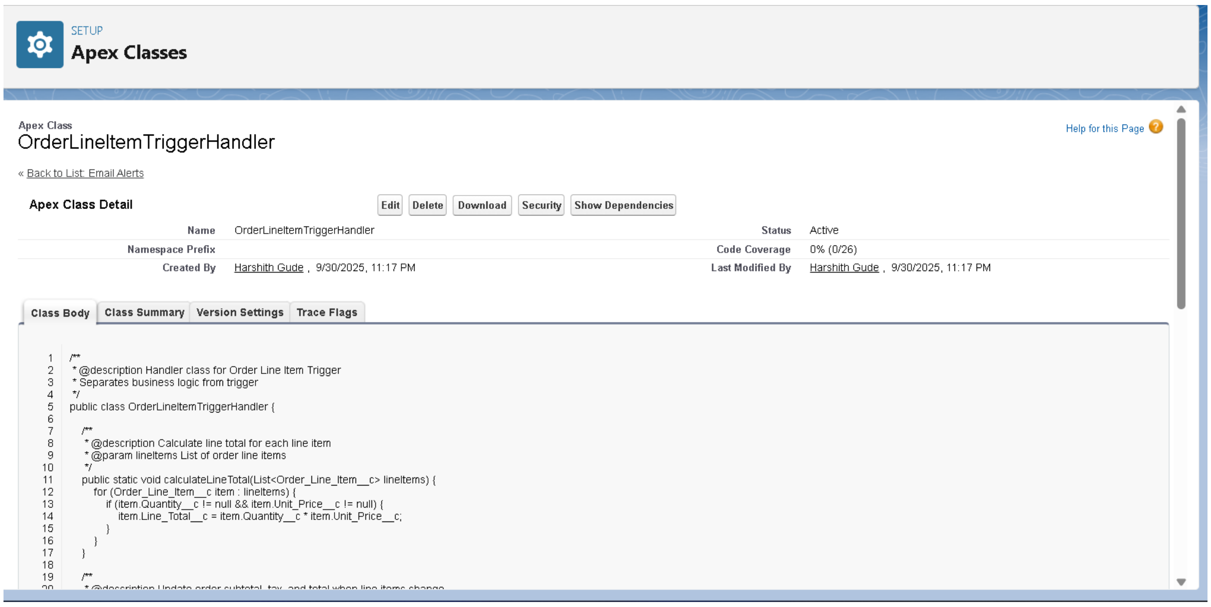


Apex Class
RestaurantUtility

Apex Class Detail

Name	RestaurantUtility	Status	Active
Namespace Prefix		Code Coverage	0% (0/22)
Created By	Harshith Gude , 9/30/2025, 10:59 PM	Last Modified By	Harshith Gude , 9/30/2025, 10:59 PM

```
/**
 * @description Utility class for restaurant-wide calculations and helper methods
 * @author Your Name
 * @date 2025
 */
public class RestaurantUtility {

    // Tax rate for restaurant orders (8.5%)
    public static final Decimal TAX_RATE = 0.085;

    // Minimum order value for free delivery
    public static final Decimal FREE_DELIVERY_THRESHOLD = 50.00;

    /**
     * @description Calculate tax amount based on subtotal
     * @param subtotal The order subtotal
     * @return Decimal tax amount
     */
    public static Decimal calculateTax(Decimal subtotal) {
```



Apex Class
OrderLineItemTriggerHandler

« Back to List: Email Alerts

Apex Class Detail

Name	OrderLineItemTriggerHandler	Status	Active
Namespace Prefix		Code Coverage	0% (0/26)
Created By	Harshith Gude , 9/30/2025, 11:17 PM	Last Modified By	Harshith Gude , 9/30/2025, 11:17 PM

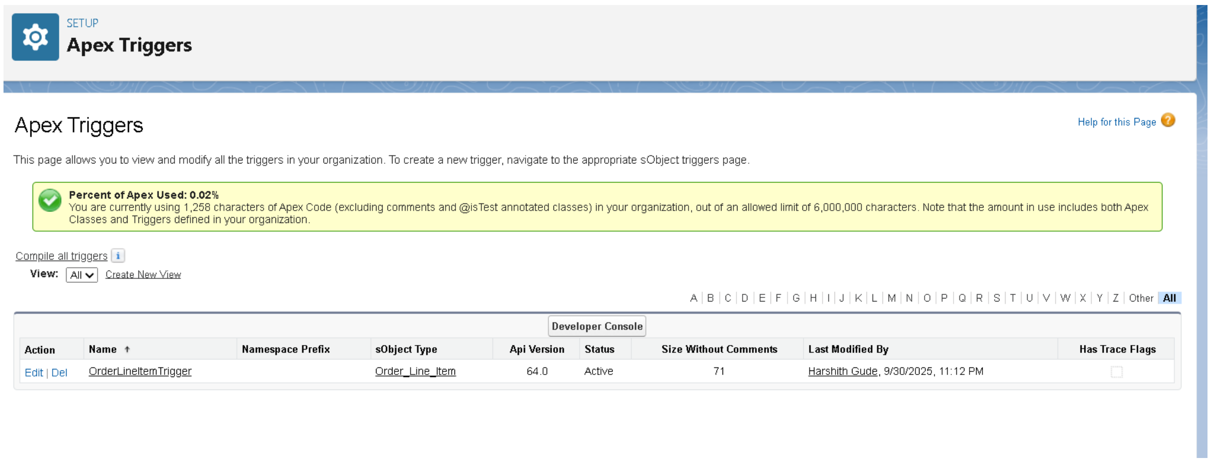
```
/**
 * @description Handler class for Order Line Item Trigger
 * Separates business logic from trigger
 */
public class OrderLineItemTriggerHandler {

    /**
     * @description Calculate line total for each line item
     * @param lineItems List of order line items
     */
    public static void calculateLineTotal(List<Order_Line_Item__c> lineItems) {
        for (Order_Line_Item__c item : lineItems) {
            if (item.Quantity__c != null && item.Unit_Price__c != null) {
                item.Line_Total__c = item.Quantity__c * item.Unit_Price__c;
            }
        }
    }

    /**
     * @description Update order subtotal, tax, and total when line items change
```

3. Apex Triggers

Three core triggers were developed as part of this system.

1. **OrderLineItemTrigger** – Handles line item calculations, ensuring that order totals always reflect the correct amounts whenever items are inserted, updated, or deleted.
2. **InventoryItemTrigger** – Monitors ingredient stock levels. When the stock drops below the minimum threshold, it automatically raises a purchase order.
3. **RestaurantOrderTrigger** – Ensures that order rules are respected. For instance, completed orders cannot be modified, and order confirmation reduces stock automatically.

Each trigger delegates its logic to a corresponding handler class. This keeps triggers lightweight and reduces the risk of recursion or performance issues.


SETUP

Apex Triggers

Apex Triggers

Help for this Page

This page allows you to view and modify all the triggers in your organization. To create a new trigger, navigate to the appropriate sObject triggers page.

Percent of Apex Used: 0.02%
You are currently using 1,258 characters of Apex Code (excluding comments and @isTest annotated classes) in your organization, out of an allowed limit of 6,000,000 characters. Note that the amount in use includes both Apex Classes and Triggers defined in your organization.

Compile all triggers

View: All Create New View

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Other | All

Developer Console

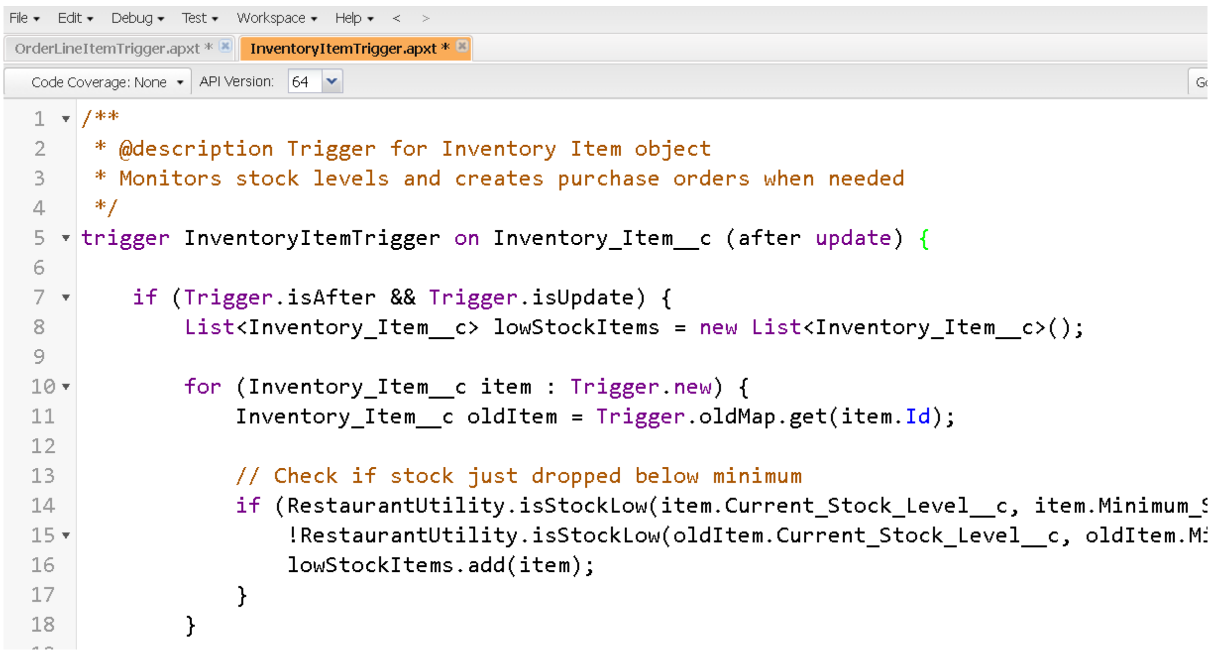
Action	Name ↑	Namespace Prefix	sObject Type	Api Version	Status	Size Without Comments	Last Modified By	Has Trace Flags
Edit | Del	OrderLineItemTrigger		Order_Line_Item	64.0	Active	71	Harshith Gude, 9/30/2025, 11:12 PM	☐

4. Trigger Handlers

For every trigger, there is a dedicated handler class that performs the actual business logic. For example:

- The OrderLineItemTriggerHandler calculates line totals and updates parent order totals.
- The InventoryTriggerHandler ensures that purchase orders are raised when required.
- The OrderTriggerHandler enforces validation rules, assigns default values, and manages post-order confirmation processes.

This structure makes the application easier to extend in the future and aligns with Salesforce best practices.



```
/**
 * @description Trigger for Inventory Item object
 * Monitors stock levels and creates purchase orders when needed
 */
trigger InventoryItemTrigger on Inventory_Item__c (after update) {

    if (Trigger.isAfter && Trigger.isUpdate) {
        List<Inventory_Item__c> lowStockItems = new List<Inventory_Item__c>();

        for (Inventory_Item__c item : Trigger.new) {
            Inventory_Item__c oldItem = Trigger.oldMap.get(item.Id);

            // Check if stock just dropped below minimum
            if (RestaurantUtility.isStockLow(item.Current_Stock_Level__c, item.Minimum_S
                !RestaurantUtility.isStockLow(oldItem.Current_Stock_Level__c, oldItem.Mi
                lowStockItems.add(item);
            }
        }
```

5. Utility Classes

Utility classes provide commonly used methods across the platform. Examples include methods to calculate subtotals, tax amounts, total amounts, and profit margins. They also include helper methods for validating stock levels and generating unique reference numbers for purchase orders.

By keeping these methods in a utility class, code duplication is avoided and consistency is ensured across the application.

6. Batch Apex

Batch Apex was implemented for tasks that need to handle large volumes of data or need to be scheduled at fixed intervals.

One of the main batch jobs implemented is responsible for scanning the inventory every night. It identifies expired ingredients and sets their available stock to zero so that they are not used in new orders.

It also checks for low stock items and prepares alerts or initiates purchase requests.

Batch Apex ensures that these operations are performed efficiently without hitting governor limits, even when thousands of records need to be processed.

8. Reporting Classes

Dedicated classes were also written to prepare analytical data for restaurant managers. These reports cover sales trends, inventory usage, and profit margins. They use aggregate queries to present summarized information in dashboards and management reports.

This makes it easier for decision-makers to track business health and identify which items are most profitable or most frequently ordered.

9. Recipe Cost Calculator

A special class was created to handle recipe costing. Each menu item is linked to its required ingredients, and this class calculates the cost of producing that item. Whenever ingredient prices are updated, the cost of menu items is recalculated automatically.

This ensures that the restaurant always has accurate profit margins for every dish and can make pricing decisions accordingly.

10. Exception Handling

Errors are inevitable, but they must be handled gracefully. For this reason, a centralized exception handler class called **RestaurantExceptionHandler** was implemented.

This class defines and manages several types of custom exceptions:

- **InsufficientInventoryException** – Raised when the available stock is not enough to fulfill an order.
- **InvalidOrderException** – Raised when an order is incomplete or its totals are invalid.
- **General DML and Unexpected Exceptions** – Captured and logged with rollbacks to maintain system stability.

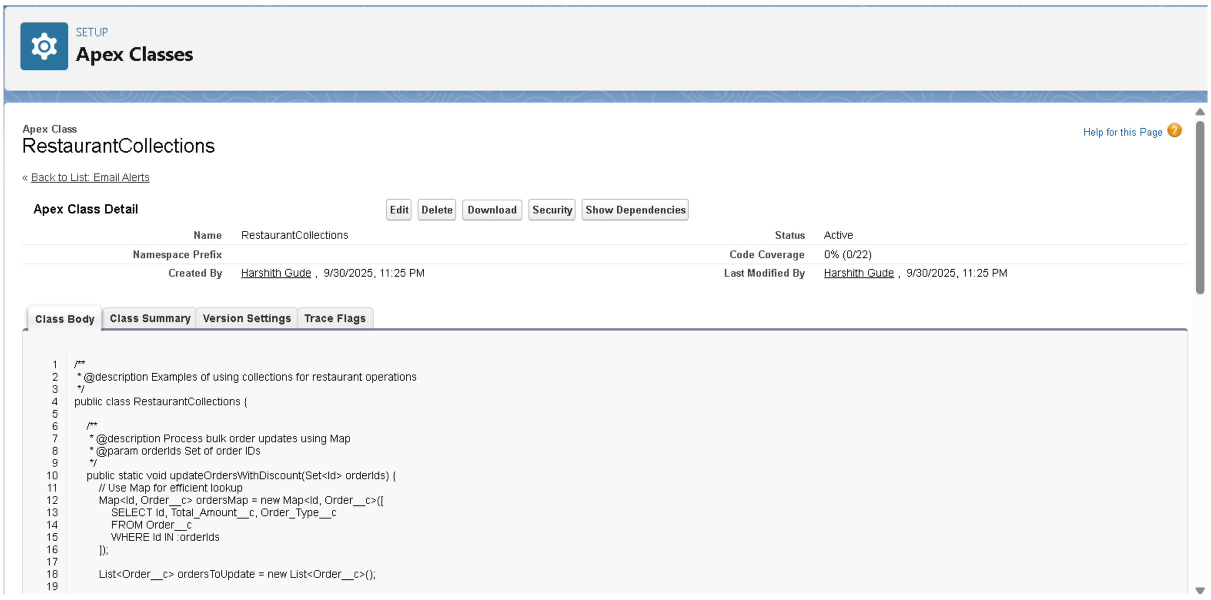
By having a structured approach to error management, the system ensures both data integrity and user clarity.

11. Test Classes

Salesforce requires that all Apex classes and triggers be tested before deployment. Test classes confirm that the system behaves as expected in various scenarios. The following test classes were created:

1. **RestaurantUtilityTest** – Validates calculations for totals, tax amounts, profit margins, and stock checks.
2. **OrderLineItemTriggerTest** – Ensures that order totals are recalculated correctly when line items are added, modified, or deleted.

3. **InventoryTriggerTest** – Verifies that purchase orders are generated when inventory levels fall below minimum thresholds.
4. **BatchApexTest** – Tests nightly batch jobs for expired stock management and low stock alerts.



Apex Class
RestaurantCollections

Apex Class Detail

Name	RestaurantCollections	Status	Active
Namespace Prefix		Code Coverage	0% (0/22)
Created By	Harshith Gude , 9/30/2025, 11:25 PM	Last Modified By	Harshith Gude , 9/30/2025, 11:25 PM

```
/**
 * @description Examples of using collections for restaurant operations
 */
public class RestaurantCollections {

    /**
     * @description Process bulk order updates using Map
     * @param orderIds Set of order IDs
     */
    public static void updateOrdersWithDiscount(Set<Id> orderIds) {
        // Use Map for efficient lookup
        Map<Id, Order__c> ordersMap = new Map<Id, Order__c>([
            SELECT Id, Total_Amount__c, Order_Type__c
            FROM Order__c
            WHERE Id IN :orderIds
        ]);

        List<Order__c> ordersToUpdate = new List<Order__c>();

```

Test Class	What It Tests	Expected Outcome
RestaurantUtilityTest	Tax, totals, stock check, profit margin	Accurate financial calculations (8.5% tax, correct totals, margin % accurate)
OrderLineItemTriggerTest	Insert/update/delete of line items	Order subtotal, tax, total update correctly after each line item change
InventoryTriggerTest	Low stock detection & PO auto-creation	PO created only when stock < min, no duplicate orders if stock is healthy
BatchApexTest	Nightly batch jobs (expired & low stock handling)	Expired stock reset to 0, low stock flagged, system cleanup runs correctly
AsyncOrderProcessor (Future)	Async order confirmation & inventory reduction	Orders confirmed in background, stock reduced without blocking user operation
RecipeCostCalculator	Menu item recipe-based costing	Menu item cost auto-updated, detailed ingredient breakdown generated
OrderTriggerHandler	Trigger framework validations (before/after logic)	Default order date set, no editing completed orders, confirmed orders reduce inventory