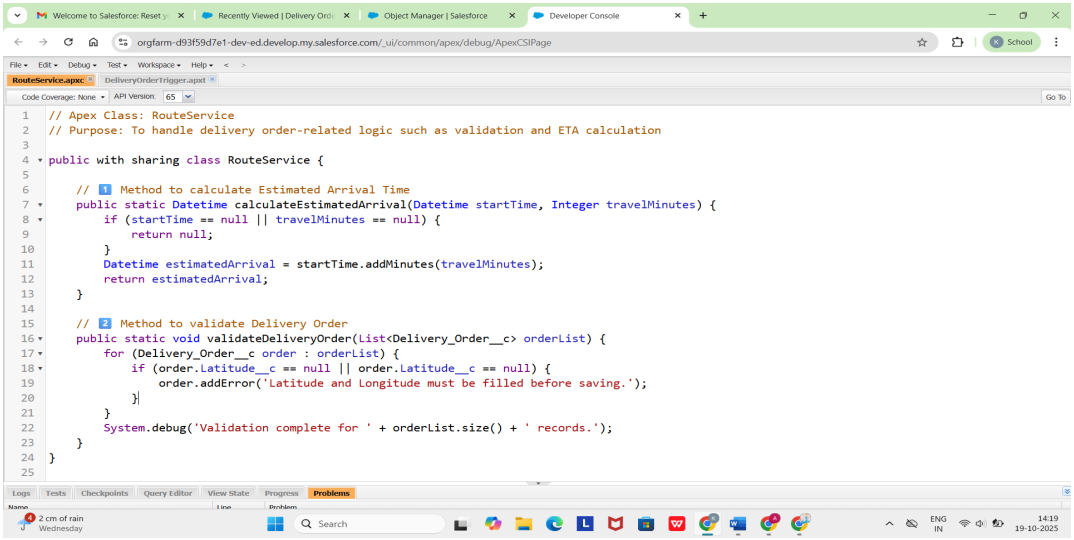
Dynamic Route Optimization — Phase 5 Apex Development

This document provides a detailed explanation of Apex programming concepts implemented in Phase 5 of the Dynamic Route Optimization project. It includes Apex Classes, Triggers, SOQL/SOSL queries, and asynchronous Apex such as Batch, Queueable, Scheduled, and Future methods, along with Exception Handling examples. Each section contains explanations, sample code, and corresponding screenshots.

1. Apex Classes & Objects

Apex Classes are used to encapsulate business logic. In this project, the RouteService class performs calculations such as estimating delivery times and validating delivery orders. It ensures that key fields like Latitude and Longitude are not left empty before saving the record.

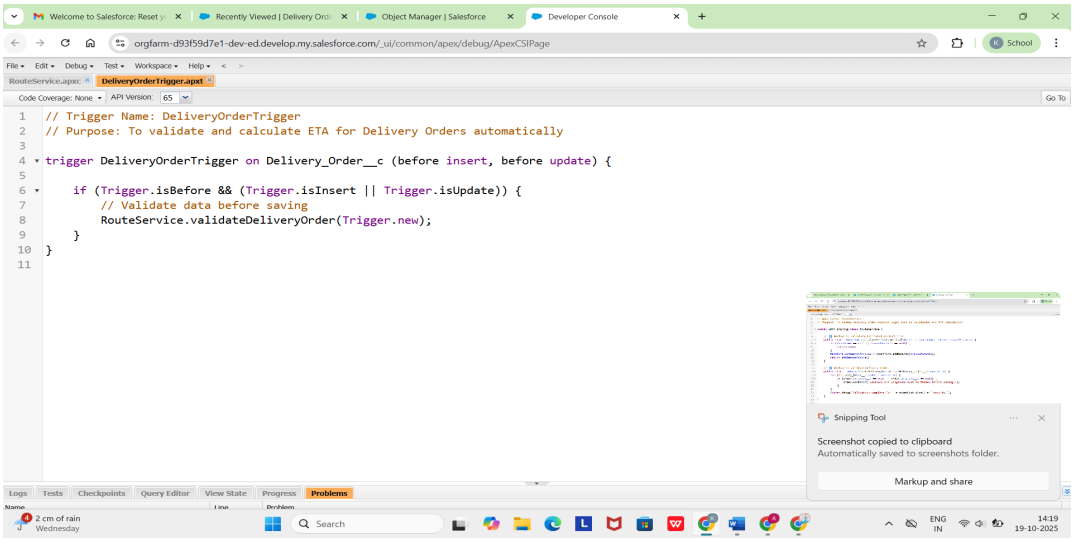
public with sharing class RouteService { public static Datetime calculateEstimatedArrival(Datetime startTime, Integer travelMinutes) { if (startTime == null || travelMinutes == null) return null; return startTime.addMinutes(travelMinutes); } public static void validateDeliveryOrder(List orderList) { for (Delivery_Order__c order : orderList) { if (order.Latitude__c == null || order.Longitude__c == null) { order.addError('Latitude and Longitude must be filled before saving.'); } } } }



2. Apex Triggers

Apex Triggers automate actions before or after database operations. The DeliveryOrderTrigger calls the RouteService class to validate data before inserting or updating records.

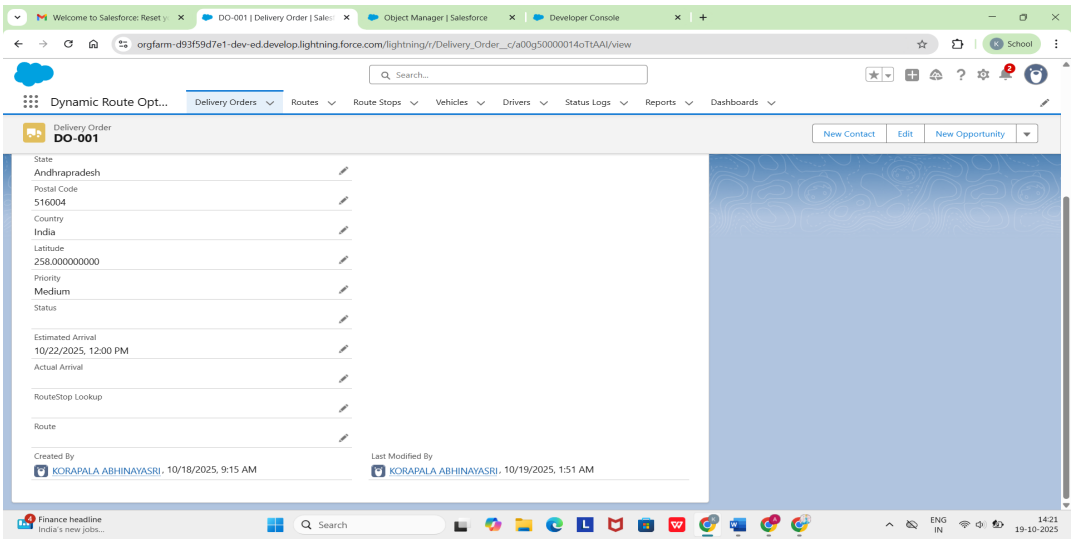
trigger DeliveryOrderTrigger on Delivery_Order__c (before insert, before update) { if (Trigger.isBefore && (Trigger.isInsert || Trigger.isUpdate)) { RouteService.validateDeliveryOrder(Trigger.new); } }



3. SOQL & SOSL Queries

SOQL (Salesforce Object Query Language) retrieves records from objects, while SOSL searches text across multiple objects. Below is a simple query example to fetch assigned Delivery Orders.

List orders = [SELECT Id, Name, Status__c FROM Delivery_Order__c WHERE Status__c = 'Assigned']; System.debug(orders);

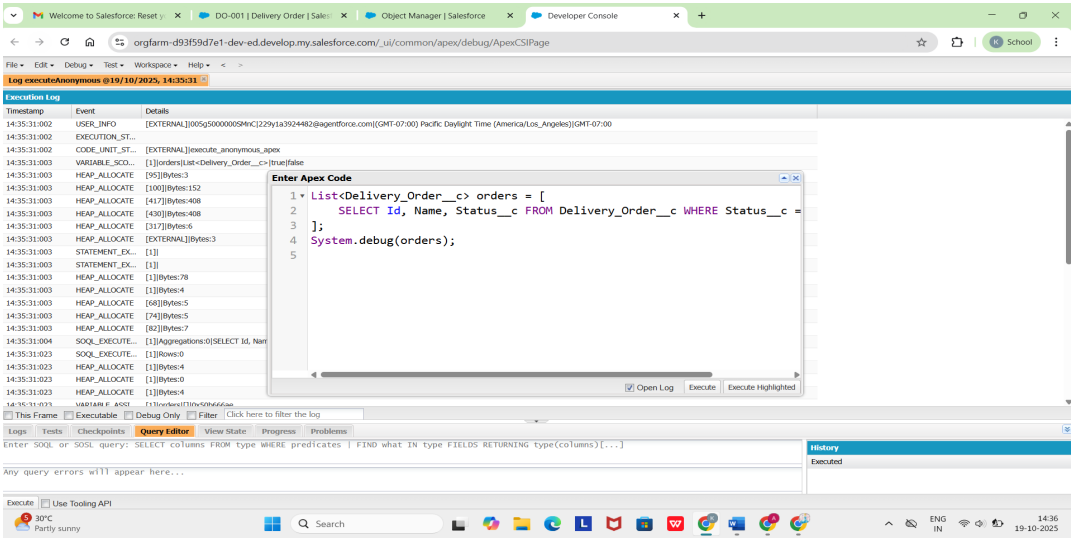


4. Batch Apex

Batch Apex is designed for handling large data volumes asynchronously. It processes records in manageable batches and executes logic efficiently.

global class DeliveryOrderBatch implements Database.Batchable { global Database.QueryLocator start(Database.BatchableContext bc) { return

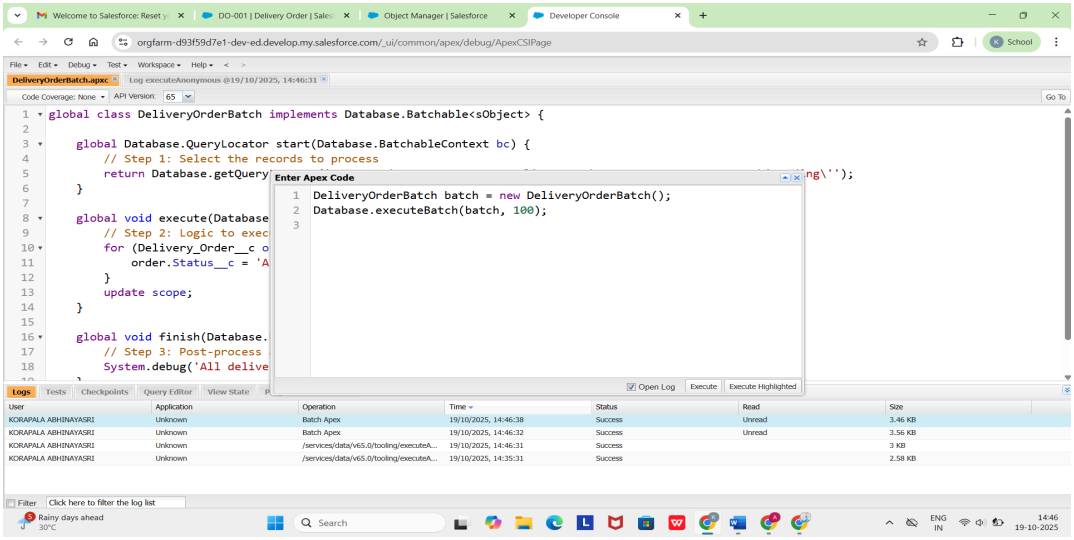
Database.getQueryLocator('SELECT Id, Status__c FROM Delivery_Order__c'); } global void execute(Database.BatchableContext bc, List scope) { for (Delivery_Order__c order : scope) { order.Status__c = 'Assigned'; } update scope; } global void finish(Database.BatchableContext bc) { System.debug('Batch Process Completed.'); } }



5. Queueable Apex

Queueable Apex provides an easy way to run asynchronous jobs. It allows for more complex logic and chaining of multiple jobs. Here, we update delivery orders to 'In Transit' when queued.

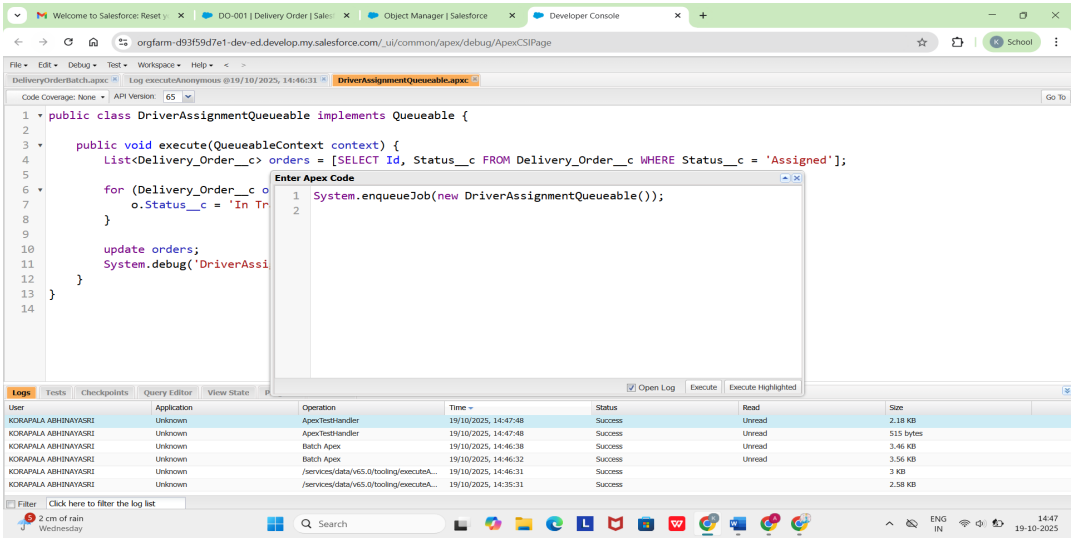
public class DriverAssignmentQueueable implements Queueable { public void execute(QueueableContext context) { List orders = [SELECT Id, Status__c FROM Delivery_Order__c WHERE Status__c = 'Assigned']; for (Delivery_Order__c o : orders) { o.Status__c = 'In Transit'; } update orders; } }



6. Scheduled Apex

Scheduled Apex allows you to run classes at specific times automatically. The DailyDeliveryScheduler executes the Batch Apex daily to process delivery orders.

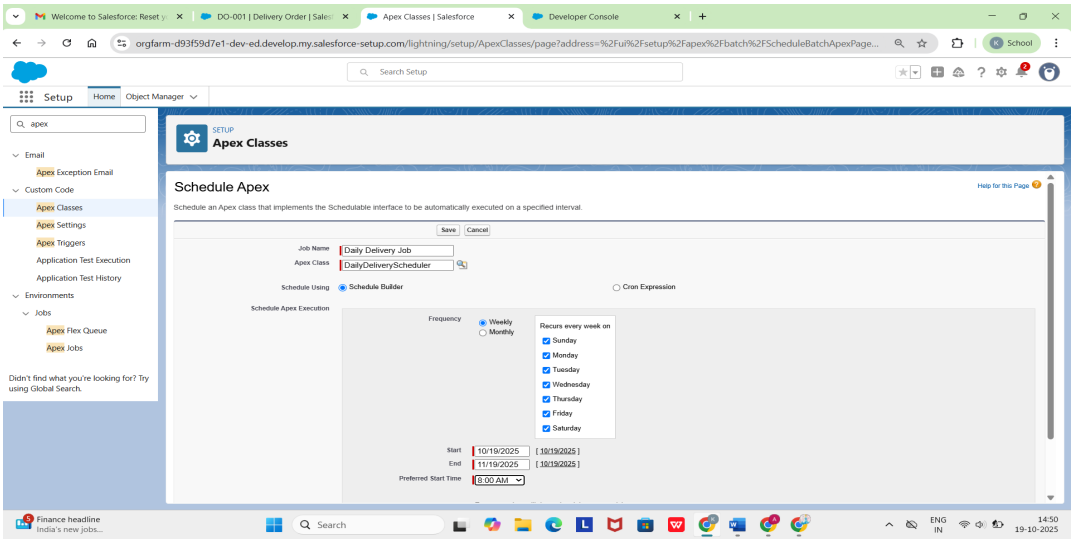
global class DailyDeliveryScheduler implements Schedulable { global void execute(SchedulableContext sc) { Database.executeBatch(new DeliveryOrderBatch(), 100); } }



7. Future Methods & Exception Handling

Future methods allow asynchronous operations like sending notifications or external calls without blocking the main execution. Exception handling ensures smooth error management.

public class DeliveryNotificationService { @future public static void sendEmailAsync(String email, String message) { System.debug('Email sent to: ' + email + ' | Message: ' + message); } } public class DeliveryUpdateService { public static void updateOrderStatus(Delivery_Order__c order, String newStatus) { try { order.Status__c = newStatus; update order; } catch (DmlException e) { System.debug('Error updating order: ' + e.getMessage()); } } }



8. Summary

This phase covered the implementation of core Apex programming concepts such as Classes, Triggers, SOQL/SOSL, and asynchronous Apex features. The integration of Batch, Queueable, and Scheduled Apex provides efficient background processing. Exception handling and future methods add reliability and performance to the project.