# Phase 5 — Apex Programming (Developer) Smart Task & Email Summarizer for Executives

Purpose  
This document is a step-by-step, copy‑ready developer guide for Phase 5 of the "Smart Task & Email Summarizer for Executives" project.  
It contains architecture notes, Apex code samples (Trigger, Handler, Service, Batch, Queueable, Scheduled, @future), test classes, best practices and a deployment checklist.  
Use the code samples as starting points — adapt field/API names exactly to your org (this guide assumes objects named Booking\_\_c and Car\_\_c and fields like Start\_Date\_\_c, End\_Date\_\_c, Car\_\_c lookup, Booking\_Status\_\_c, Daily\_Rate\_\_c, Total\_Amount\_\_c).

## Quick checklist (must-do, prioritized)

• Create / verify custom objects and fields: Car\_\_c (Daily\_Rate\_\_c, Status\_\_c), Booking\_\_c (Car\_\_c lookup, Start\_Date\_\_c, End\_Date\_\_c, Booking\_Status\_\_c, Total\_Amount\_\_c, Discount\_\_c).

• Implement Trigger + Trigger Handler (bulkified) to prevent overlapping bookings.

• Create BookingService for reusable logic (date helpers, queries, email senders, callouts).

• Write Batch Apex to mark overdue rentals nightly.

• Create Queueable job to calculate discounts for bulk updates.

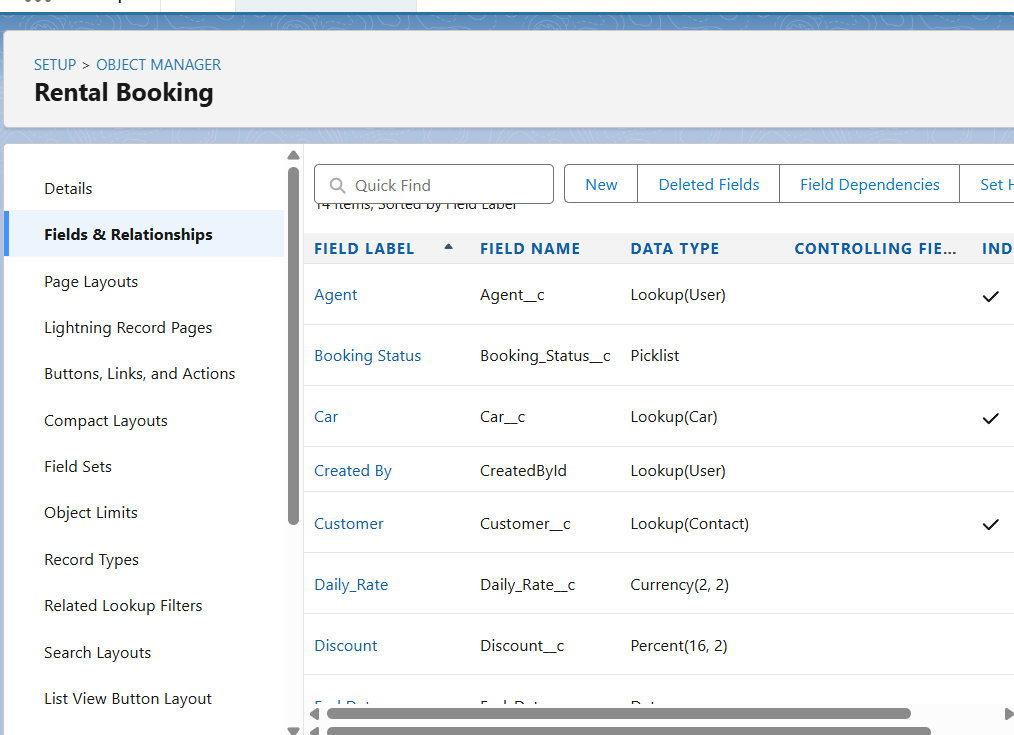
• Write Scheduled Apex to run the morning report and/or schedule the batch.

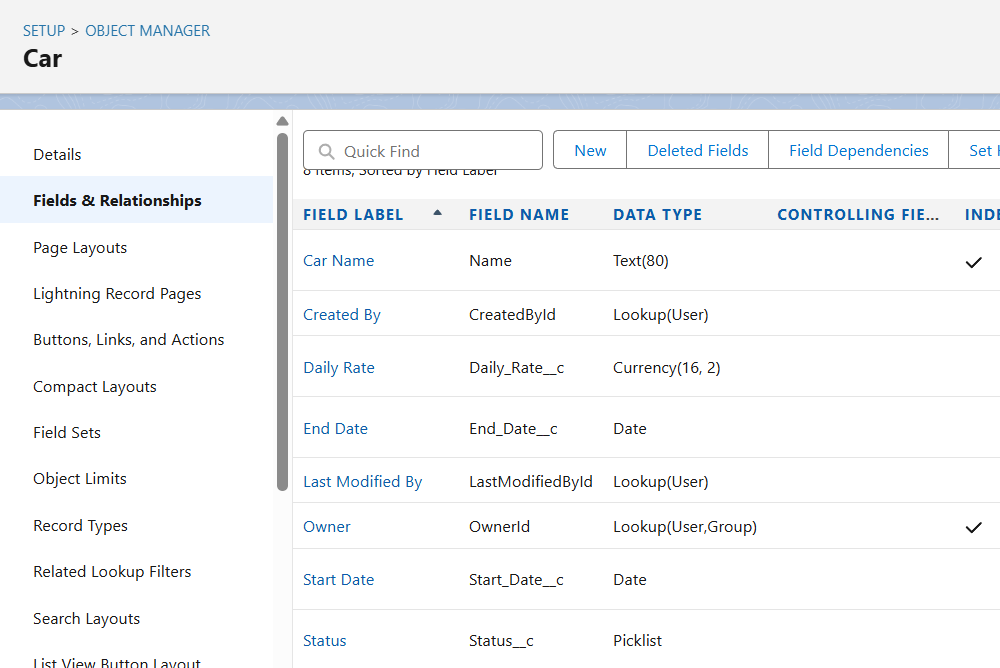
• Implement @future(callout=true) or better: use Queueable with callout for insurance API.

• Write robust Test Classes (use Test.startTest()/stopTest(), @testSetup, HttpCalloutMock for callout tests).

• Create Named Credential / Remote Site for external callouts and update Remote Site Settings or use Named Credentials.

• Deploy to QA sandbox and run all tests; confirm 75%+ coverage.

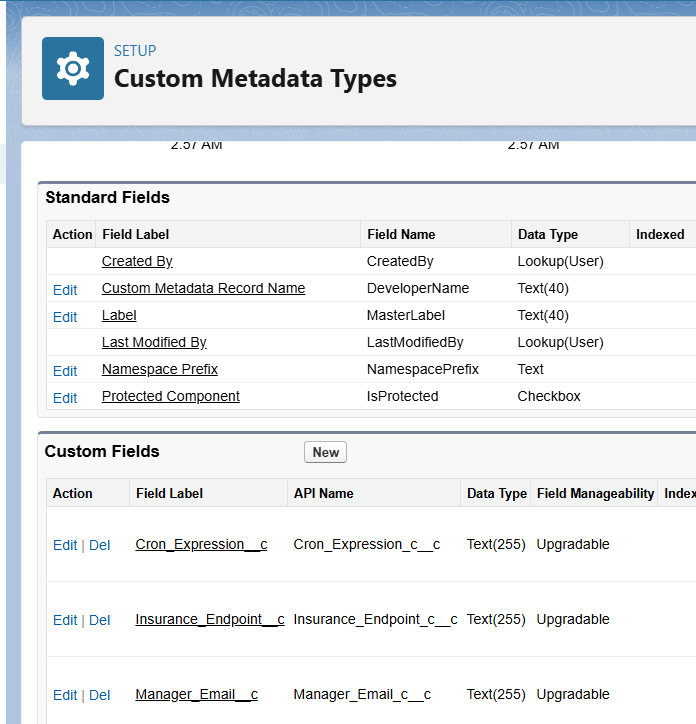


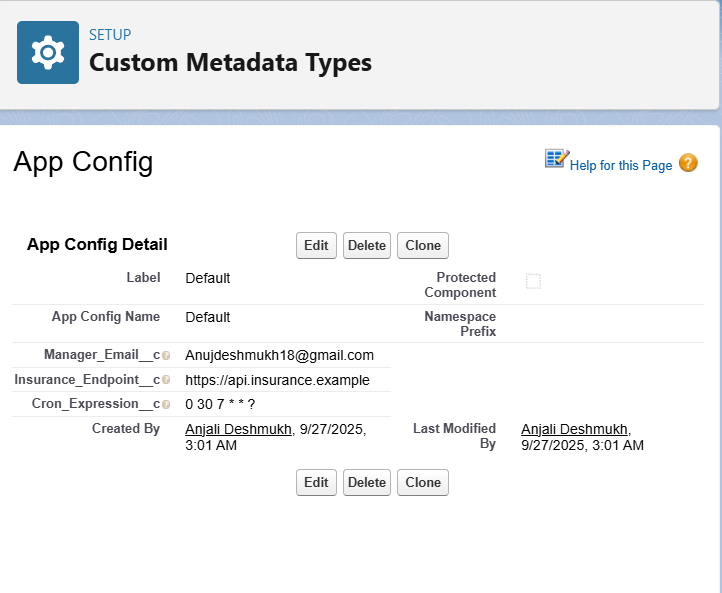


## Architecture & important notes

- Use Booking\_\_c and Car\_\_c as canonical names in this guide. Change them if your org uses different API names.  
- Bulkify everything: never run SOQL/DML inside loops. Use Maps/Sets/Lists.  
- Avoid callouts inside triggers. Queue the work and call out asynchronously (@future or Queueable callout).  
- Store configuration (manager emails, API endpoints) in Custom Metadata / Custom Settings or Named Credential for safe changes without code edits.  
- Use descriptive Booking\_Status\_\_c picklist values such as 'Confirmed', 'Cancelled', 'Returned', 'Overdue'.  
- Use seeAllData=false in tests and create test records programmatically.

Usage: In Apex, query the custom metadata:



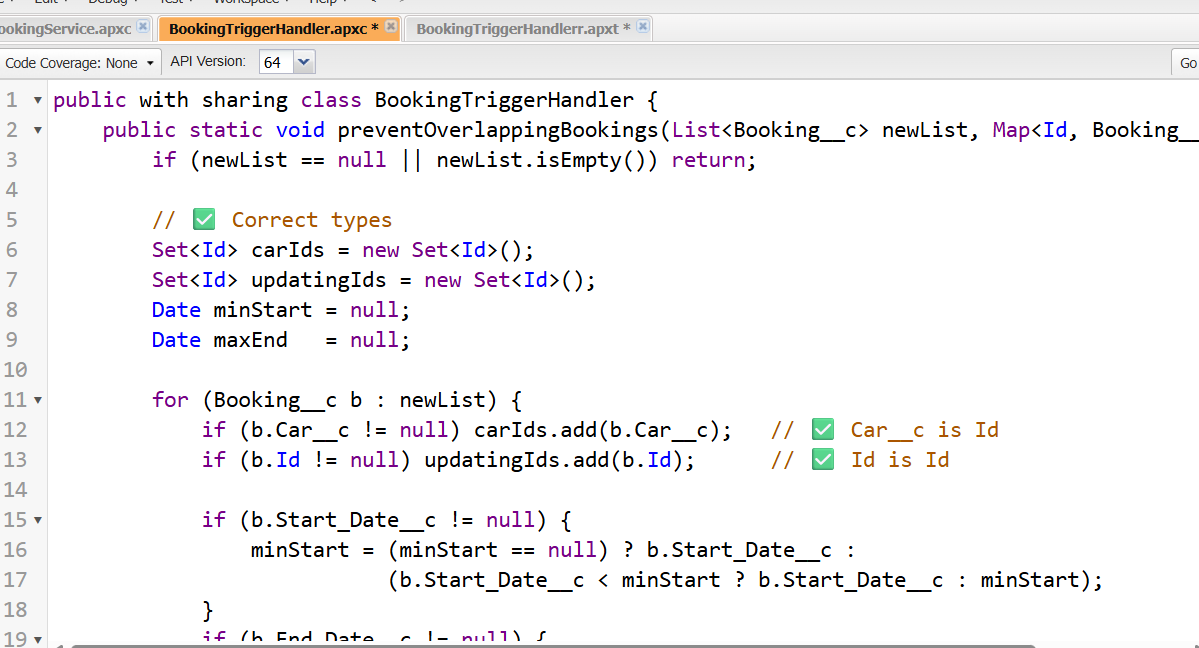


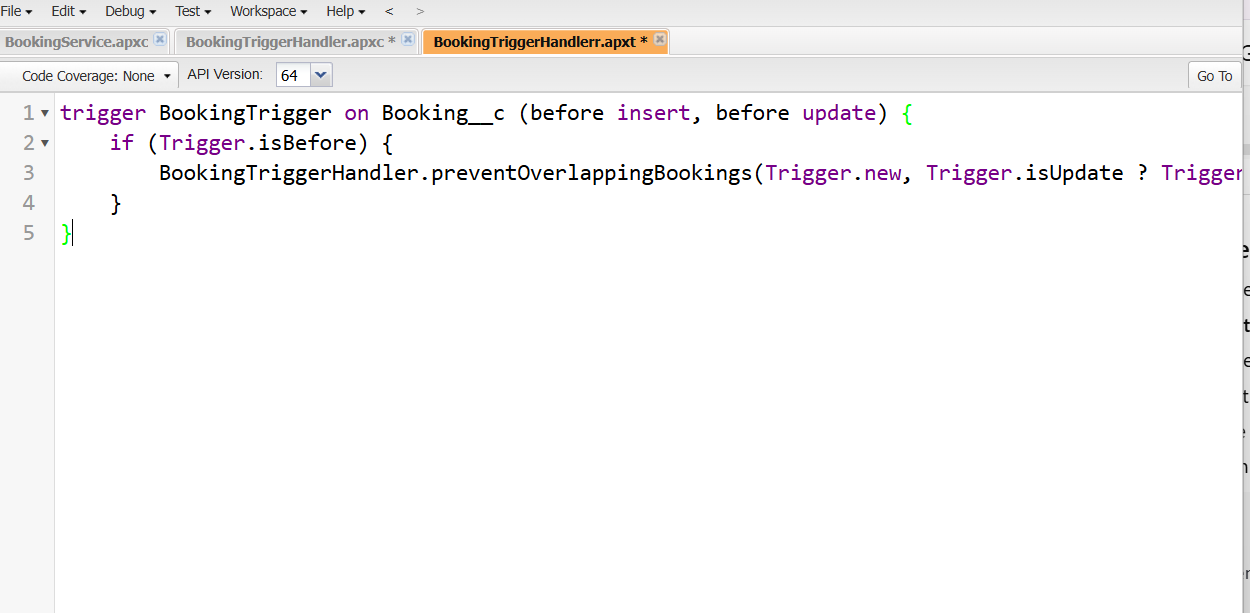
## 1) Trigger + Trigger Handler — prevent overlapping bookings (bulkified)

Create a before insert, before update trigger that delegates to a handler class.

trigger BookingTrigger on Booking\_\_c (before insert, before update) {  
 if (Trigger.isBefore) {  
 if (Trigger.isInsert || Trigger.isUpdate) {  
 BookingTriggerHandler.preventOverlappingBookings(Trigger.new, Trigger.isUpdate ? Trigger.oldMap : null);  
 }  
 }  
}

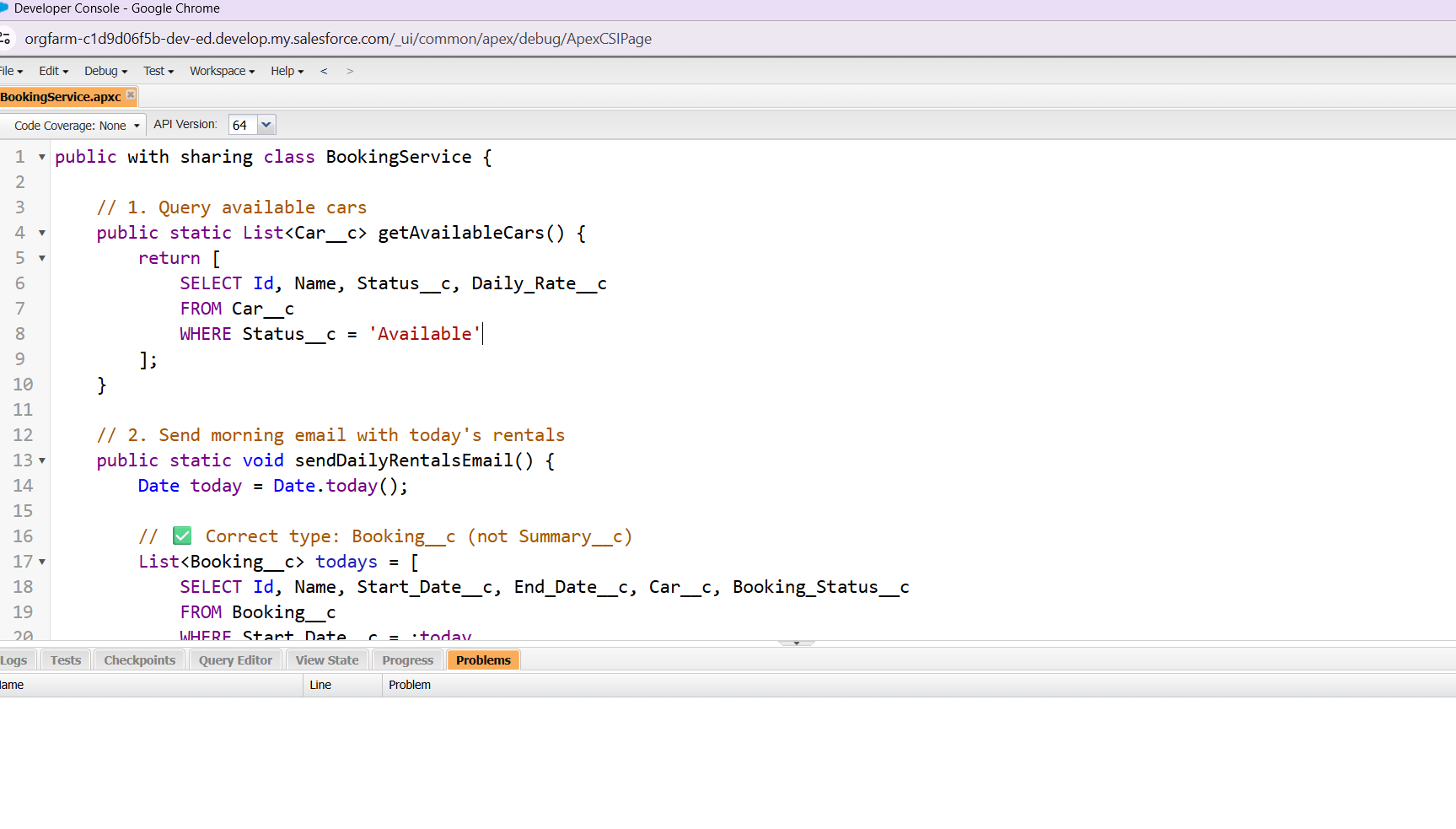
public with sharing class BookingTriggerHandler {  
  
 // Prevent overlapping bookings for same car. Bulk-safe.  
 public static void preventOverlappingBookings(List<Booking\_\_c> newBookings, Map<Id, Booking\_\_c> oldMap) {  
 // Collect involved car Ids (avoid nulls)  
 Set<Id> carIds = new Set<Id>();  
 for (Booking\_\_c b : newBookings) {  
 if (b.Car\_\_c != null) carIds.add(b.Car\_\_c);  
 }  
 if (carIds.isEmpty()) return;  
  
 // Query existing bookings for those cars (exclude Cancelled). Exclude records that are being updated by checking oldMap keys.  
 List<Booking\_\_c> existing = [  
 SELECT Id, Start\_Date\_\_c, End\_Date\_\_c, Booking\_Status\_\_c, Car\_\_c  
 FROM Booking\_\_c  
 WHERE Car\_\_c IN :carIds AND Booking\_Status\_\_c != 'Cancelled'  
 ];  
  
 // Group existing bookings by car  
 Map<Id, List<Booking\_\_c>> existingByCar = new Map<Id, List<Booking\_\_c>>();  
 for (Booking\_\_c e : existing) {  
 if (!existingByCar.containsKey(e.Car\_\_c)) existingByCar.put(e.Car\_\_c, new List<Booking\_\_c>());  
 existingByCar.get(e.Car\_\_c).add(e);  
 }  
  
 // Validate each incoming booking for overlap  
 for (Booking\_\_c nb : newBookings) {  
 // Basic date validation  
 if (nb.Start\_Date\_\_c == null || nb.End\_Date\_\_c == null) continue;  
 if (nb.Start\_Date\_\_c > nb.End\_Date\_\_c) {  
 nb.addError('Start Date cannot be after End Date.');  
 continue;  
 }  
 List<Booking\_\_c> listForCar = existingByCar.get(nb.Car\_\_c);  
 if (listForCar == null) continue;  
 for (Booking\_\_c ex : listForCar) {  
 // If updating, ignore comparing to the same record  
 if (oldMap != null && ex.Id == nb.Id) continue;  
 if (datesOverlap(nb.Start\_Date\_\_c, nb.End\_Date\_\_c, ex.Start\_Date\_\_c, ex.End\_Date\_\_c)) {  
 nb.addError('This booking overlaps with existing booking with Id: ' + ex.Id);  
 break;  
 }  
 }  
 }  
 }  
  
 // Helper to determine interval overlap (Date objects)  
 public static Boolean datesOverlap(Date aStart, Date aEnd, Date bStart, Date bEnd) {  
 if (aStart == null || aEnd == null || bStart == null || bEnd == null) return false;  
 return !(aEnd < bStart || aStart > bEnd);  
 }  
}





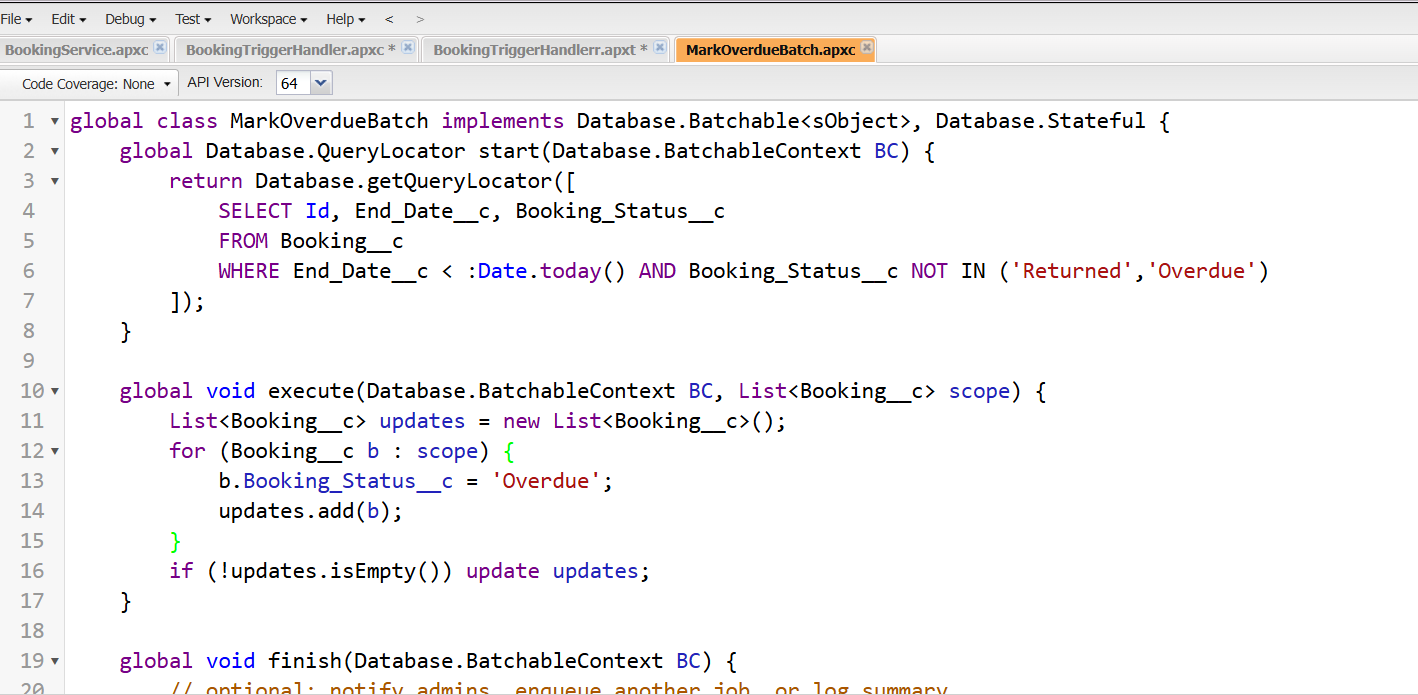
## 2) BookingService — reusable logic & queries

public with sharing class BookingService {  
 // Example SOQL for available cars  
 public static List<Car\_\_c> getAvailableCars() {  
 return [SELECT Id, Name, Status\_\_c, Daily\_Rate\_\_c FROM Car\_\_c WHERE Status\_\_c = 'Available'];  
 }  
  
 // Send morning email with today's rentals (simple example)  
 public static void sendDailyRentalsEmail() {  
 Date today = Date.today();  
 List<Booking\_\_c> todays = [SELECT Id, Name, Start\_Date\_\_c, End\_Date\_\_c, Car\_\_c, Booking\_Status\_\_c FROM Booking\_\_c WHERE Start\_Date\_\_c = :today];  
 if (todays.isEmpty()) return;  
  
 String body = 'Today\'s Bookings:\n';  
 for (Booking\_\_c b : todays) {  
 body += b.Name + ' (CarId: ' + String.valueOf(b.Car\_\_c) + ') Start: ' + String.valueOf(b.Start\_Date\_\_c) + '\n';  
 }  
  
 Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();  
 mail.setToAddresses(new List<String>{ 'manager@example.com' }); // Replace or look up managers dynamically  
 mail.setSubject('Today\'s Rentals (' + String.valueOf(today) + ')');  
 mail.setPlainTextBody(body);  
 Messaging.sendEmail(new List<Messaging.SingleEmailMessage>{ mail });  
 }  
  
 // Example @future call to external insurance API (async callout)  
 @future(callout=true)  
 public static void callInsuranceApi(Set<Id> bookingIds) {  
 Http http = new Http();  
 for (Id bId : bookingIds) {  
 try {  
 HttpRequest req = new HttpRequest();  
 req.setEndpoint('https://example-insurance.com/api/quote');  
 req.setMethod('POST');  
 req.setHeader('Content-Type','application/json');  
 req.setBody('{' + '"bookingId":"' + bId + '"}');  
 HttpResponse res = http.send(req);  
 // Optionally parse res.getBody()  
 } catch (Exception e) {  
 System.debug('Insurance call failed for ' + bId + ': ' + e.getMessage());  
 }  
 }  
 }  
}



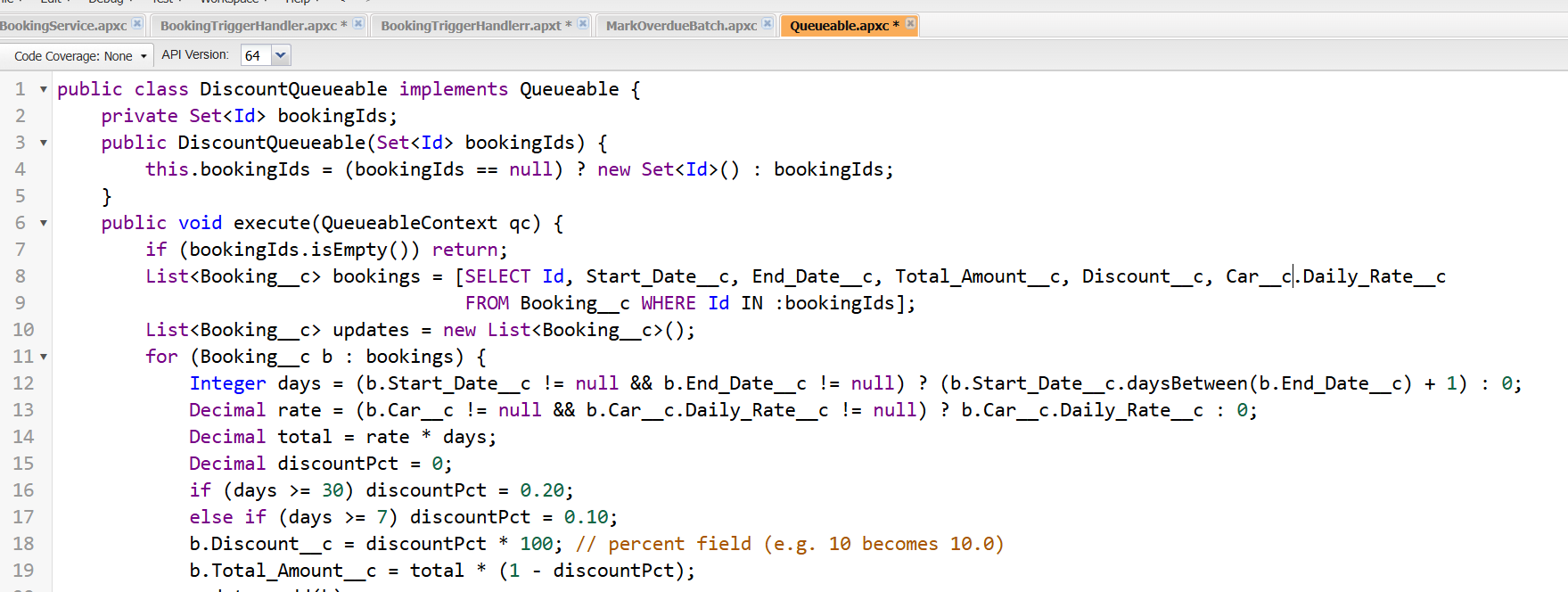
## 3) Batch Apex — nightly job to mark overdue rentals

global class MarkOverdueBatch implements Database.Batchable<sObject> {  
  
 global Database.QueryLocator start(Database.BatchableContext BC) {  
 // Find bookings ended before today and not yet marked Returned or Overdue  
 return Database.getQueryLocator([  
 SELECT Id, End\_Date\_\_c, Booking\_Status\_\_c  
 FROM Booking\_\_c  
 WHERE End\_Date\_\_c < :Date.today() AND Booking\_Status\_\_c NOT IN ('Returned','Overdue')  
 ]);  
 }  
  
 global void execute(Database.BatchableContext BC, List<Booking\_\_c> scope) {  
 List<Booking\_\_c> updates = new List<Booking\_\_c>();  
 for (Booking\_\_c b : scope) {  
 b.Booking\_Status\_\_c = 'Overdue';  
 updates.add(b);  
 }  
 if (!updates.isEmpty()) update updates;  
 }  
  
 global void finish(Database.BatchableContext BC) {  
 // Optional: notify admins or chain Queueables  
 }  
}



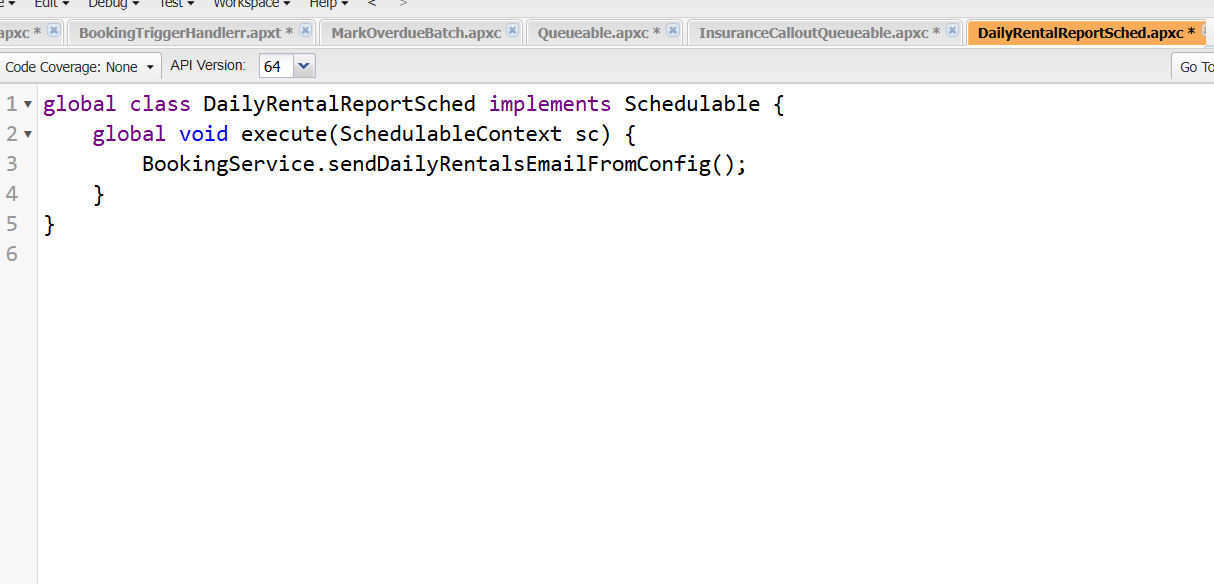
## 4) Queueable — async discount calculation for bulk rentals

public class DiscountQueueable implements Queueable {  
 private Set<Id> bookingIds;  
  
 public DiscountQueueable(Set<Id> bookingIds) {  
 this.bookingIds = bookingIds;  
 }  
  
 public void execute(QueueableContext qc) {  
 List<Booking\_\_c> bookings = [SELECT Id, Start\_Date\_\_c, End\_Date\_\_c, Total\_Amount\_\_c, Car\_\_r.Daily\_Rate\_\_c FROM Booking\_\_c WHERE Id IN :bookingIds];  
 List<Booking\_\_c> updates = new List<Booking\_\_c>();  
 for (Booking\_\_c b : bookings) {  
 Integer days = b.Start\_Date\_\_c.daysBetween(b.End\_Date\_\_c) + 1;  
 Decimal rate = (b.Car\_\_r != null && b.Car\_\_r.Daily\_Rate\_\_c != null) ? b.Car\_\_r.Daily\_Rate\_\_c : 0;  
 Decimal total = rate \* days;  
 Decimal discountPct = 0;  
 if (days >= 30) discountPct = 0.20;  
 else if (days >= 7) discountPct = 0.10;  
 b.Total\_Amount\_\_c = total \* (1 - discountPct);  
 updates.add(b);  
 }  
 if (!updates.isEmpty()) update updates;  
 }  
}



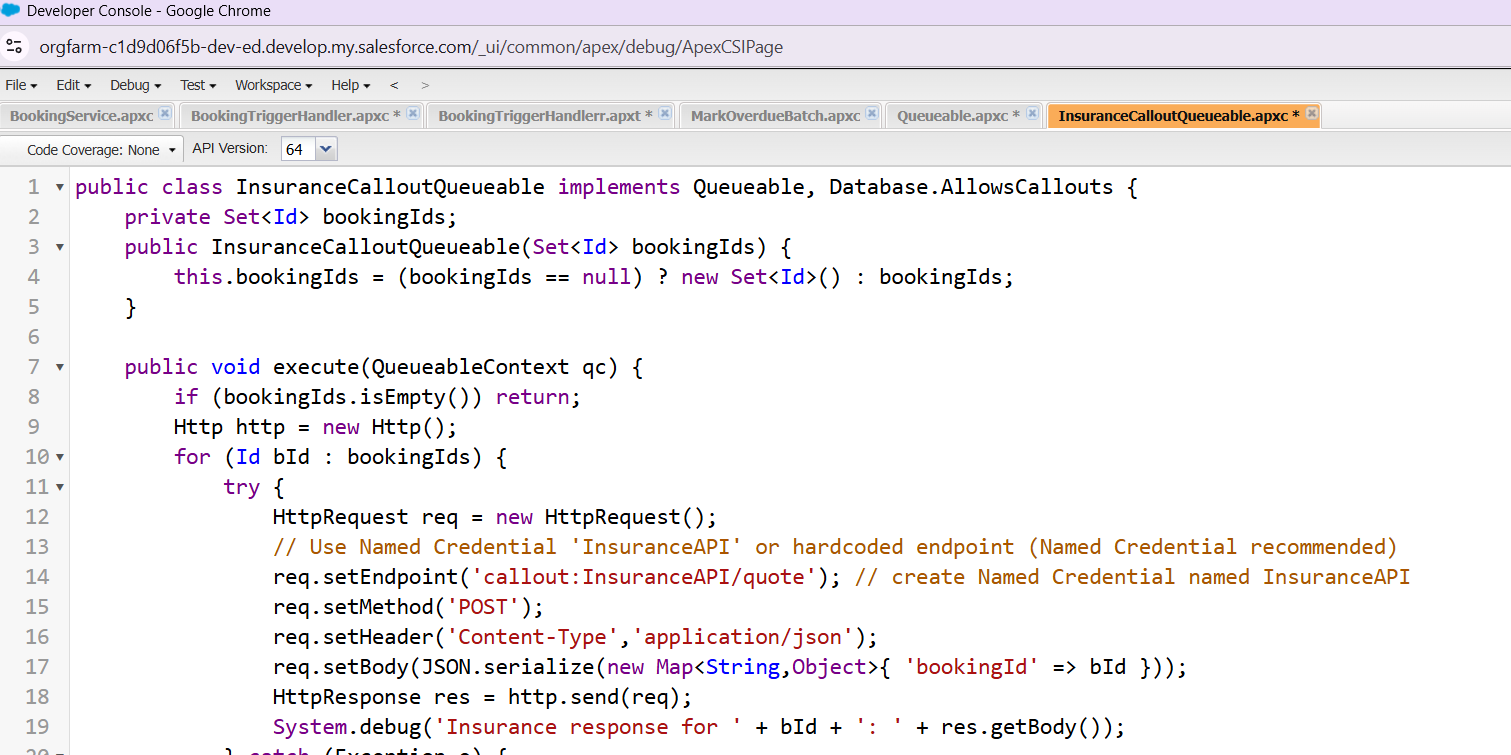
## 5) Scheduled Apex — schedule daily email report

global class DailyRentalReportSched implements Schedulable {  
 global void execute(SchedulableContext sc) {  
 BookingService.sendDailyRentalsEmail();  
 }  
}  
  
// Example to schedule: System.schedule('Daily Rentals', '0 30 7 \* \* ?', new DailyRentalReportSched());  
// The cron above runs at 7:30 AM every day (adjust timezone/cron as needed).



## 6) Future methods (callouts) — notes & best practice

- Use @future(callout=true) for simple fire-and-forget callouts (shown in BookingService). For more complex callouts prefer Queueable with callout (allows chaining).  
- Use Named Credentials and Remote Site Settings for endpoints. Tests must use HttpCalloutMock to avoid real callouts.



## 7) Exception Handling & Control Statements

- Use try/catch where external calls or risky operations occur and log the error using System.debug or Platform Events.  
- Use nb.addError(...) in triggers to stop DML for specific records with informative messages.  
- Example: if (booking dates overlap) nb.addError('...') — this prevents the record from being saved and returns a clear message to the UI/API caller.

## 8) Test Classes — required patterns & examples

Guidance:  
- Use @isTest and seeAllData=false. Create all test data in @testSetup or inside the test method.  
- Use Test.startTest()/Test.stopTest() around asynchronous or time-sensitive operations (Queueable, Batch, @future).  
- For callouts, implement HttpCalloutMock and register it with Test.setMock before the callout.  
- Verify both positive and negative flows: overlapping booking rejected, non-overlapping booking allowed, batch updates statuses, queueable updates totals.

@isTest  
private class TestBookingTrigger {  
  
 @testSetup static void setupData() {  
 Car\_\_c car = new Car\_\_c(Name='Test Car', Status\_\_c='Available', Daily\_Rate\_\_c=100);  
 insert car;  
 // existing booking: day 1..3  
 Booking\_\_c existing = new Booking\_\_c(Name='Existing', Car\_\_c=car.Id, Start\_Date\_\_c=Date.today().addDays(1), End\_Date\_\_c=Date.today().addDays(3), Booking\_Status\_\_c='Confirmed');  
 insert existing;  
 }  
  
 static testMethod void testOverlapPrevention() {  
 Car\_\_c car = [SELECT Id FROM Car\_\_c LIMIT 1];  
 Booking\_\_c newBk = new Booking\_\_c(Name='Overlap', Car\_\_c=car.Id, Start\_Date\_\_c=Date.today().addDays(2), End\_Date\_\_c=Date.today().addDays(4));  
 // Use partial DML to capture addError without raising exception  
 Database.SaveResult[] results = Database.insert(new List<Booking\_\_c>{ newBk }, false);  
 System.assertEquals(false, results[0].isSuccess(), 'Overlapping booking should be rejected');  
 System.assert(results[0].getErrors()[0].getMessage().contains('overlaps'), 'Error should mention overlap');  
 }  
}

@isTest  
private class TestMarkOverdueBatch {  
  
 @testSetup static void setup() {  
 Car\_\_c c = new Car\_\_c(Name='B', Status\_\_c='Available', Daily\_Rate\_\_c=100);  
 insert c;  
 // booking ended yesterday -> should be marked Overdue by batch  
 Booking\_\_c b = new Booking\_\_c(Name='OldBooking', Car\_\_c=c.Id, Start\_Date\_\_c=Date.today().addDays(-5), End\_Date\_\_c=Date.today().addDays(-1), Booking\_Status\_\_c='Confirmed');  
 insert b;  
 }  
  
 static testMethod void testBatchMarksOverdue() {  
 Test.startTest();  
 MarkOverdueBatch batch = new MarkOverdueBatch();  
 ID batchId = Database.executeBatch(batch, 50);  
 Test.stopTest();  
  
 Booking\_\_c changed = [SELECT Booking\_Status\_\_c FROM Booking\_\_c WHERE Name='OldBooking' LIMIT 1];  
 System.assertEquals('Overdue', changed.Booking\_Status\_\_c, 'Batch should mark ended bookings as Overdue');  
 }  
}

