Phase 5: Apex Programming (Developer)

5.1 Overview

This phase concentrated on developing and integrating **Apex programming logic** within the Salesforce environment. While previous phases (Org Setup, Data Modeling, and Declarative Automation) established the system's foundation, this phase ensured that the CRM could handle **complex business requirements** which are not achievable through declarative features alone.

Apex, being Salesforce's proprietary, strongly typed, object-oriented language, was leveraged to implement:

- **Custom automation** beyond workflow rules and Process Builder.
- **Robust data processing** using governor-limit friendly patterns.
- **Scalable asynchronous operations** for long-running tasks.
- **Test-driven development** ensuring deployment readiness.

This phase transformed the PlayStation Gaming CRM into a **customizable, efficient, and enterprise-ready platform**.

5.2 Key Apex Components Implemented

1. Apex Classes & Objects

- **Apex Classes** were developed to encapsulate business logic into reusable components.
- Core functionalities like subscription validation, repair request handling, and game-to-gamer mappings were written inside **classes** for modularity.
- Salesforce **sObject types** (standard and custom objects) were used directly in Apex for **CRUD operations** (Create, Read, Update, Delete).
- Example: A SubscriptionHandler class was created to auto-calculate subscription expiry and send notifications.

---

2. Apex Triggers

- **Before Triggers**: Used for data validation (e.g., ensuring subscription start date < end date).
- **After Triggers**: Used for post-save actions (e.g., logging support case history or sending alerts).
- Implemented using the **Trigger Handler Pattern** (one trigger per object, delegating logic to a handler class).

- Benefits: Cleaner structure, easier maintenance, scalable when new requirements arise.

---

3. SOQL (Salesforce Object Query Language) & Collections

- **SOQL queries** were optimized to fetch only necessary fields to avoid governor limit exceptions.
- **Collections (List, Set, Map)** were used for bulk processing of records in loops.
- Examples:
    - Lists to store multiple subscriptions.
    - Sets to check for unique game IDs.
    - Maps to quickly link gamer IDs with their active subscriptions.
- Bulkification ensured that triggers/processes handled **hundreds of records at once** without failing.

---

4. Asynchronous Processing

Certain tasks needed to run **outside normal transaction limits** or on schedules. These were implemented as:

- **Future Methods**: For lightweight asynchronous tasks like sending email/SMS alerts.
- **Queueable Apex**: For chaining complex background jobs (e.g., recalculating subscription usage stats).
- **Scheduled Apex**: To automatically run jobs like subscription expiry checks or monthly data cleanup.
- **Batch Apex**: For handling **large datasets** such as recalculating thousands of repair requests or gamer activity logs.

This ensured the system could scale for **enterprise-level data volumes**.

---

5. Control Statements & Exception Handling

- Used **control statements (IF-ELSE, SWITCH, loops)** to build dynamic business logic.
- Implemented **try-catch blocks** to handle unexpected errors gracefully.
- Exception handling ensured system reliability — errors were logged in custom error objects or sent as admin notifications instead of breaking transactions.
- Example: If a Repair record failed due to missing gamer info, an error log was generated instead of halting the entire batch.

---

6. Test Classes & Deployment Readiness

- Created **Apex Test Classes** to validate triggers, classes, and asynchronous logic.
- Achieved **>75% code coverage** (Salesforce minimum requirement for deployment).
- Tests covered:
    - Positive scenarios (valid subscription creation).
    - Negative scenarios (invalid dates, missing fields).
    - Bulk scenarios (inserting/updating hundreds of records).
- Benefits: Reliable deployments, easier debugging, long-term maintainability.

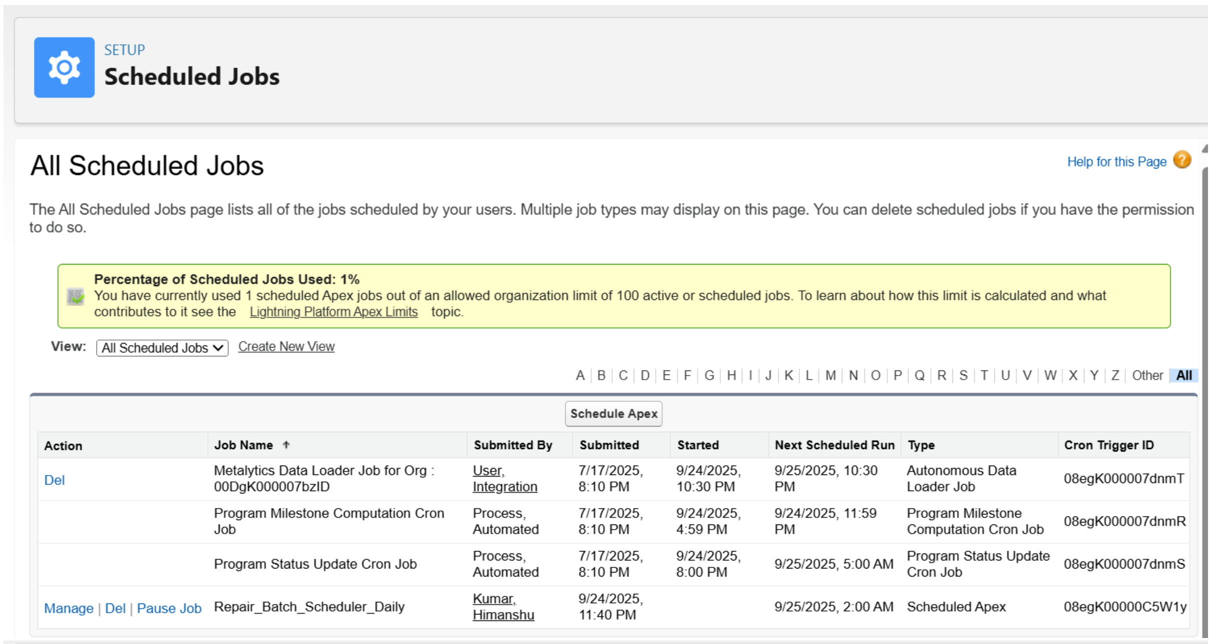
5.3 Integration with Previous Phases

- **From Phase 2 & 3 (Setup & Data Modeling):** Apex logic extended the objects and relationships by adding **dynamic automation** that complements Page Layouts and Relationships.
- **From Phase 4 (Testing):** Test classes verified that the Apex logic works correctly across all modules and remains governor-limit compliant.
- Apex ensured that the system's functionality aligned with **business requirements**, moving it beyond simple configurations into a **robust enterprise application**.

5.4 Conclusion

Phase 5 demonstrated the power of **Apex programming** in making the PlayStation Gaming CRM a scalable, intelligent, and production-ready solution. By combining classes, triggers, SOQL, and asynchronous processes, the system gained:

- **Efficiency** (bulkified operations, optimized queries).
- **Automation** (custom business logic beyond declarative tools).
- **Reliability** (exception handling, test coverage).
- **Scalability** (batch jobs, scheduled jobs for large data sets).


SETUP
Scheduled Jobs

All Scheduled Jobs

Help for this Page

The All Scheduled Jobs page lists all of the jobs scheduled by your users. Multiple job types may display on this page. You can delete scheduled jobs if you have the permission to do so.

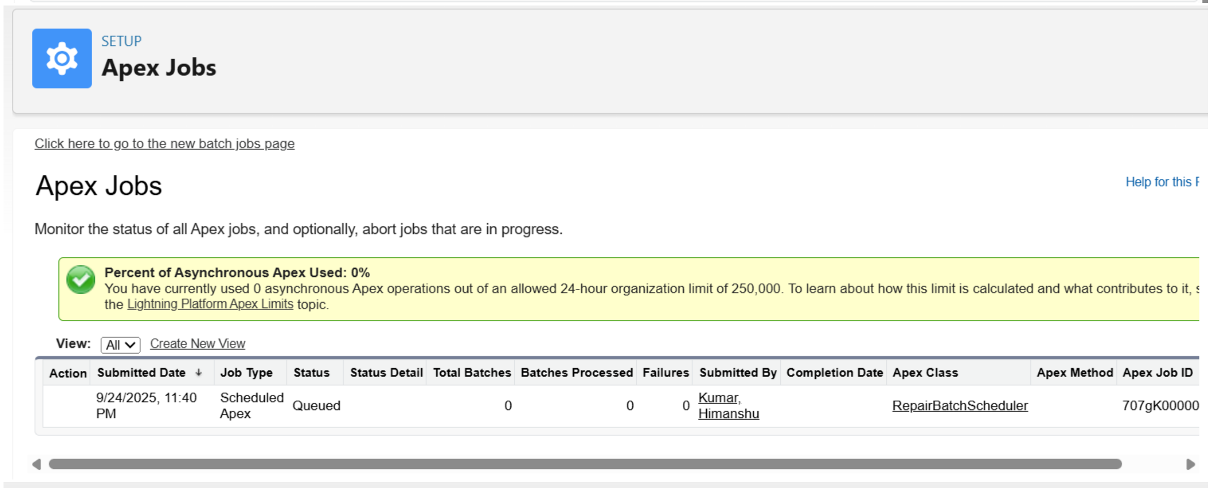
Percentage of Scheduled Jobs Used: 1%
You have currently used 1 scheduled Apex jobs out of an allowed organization limit of 100 active or scheduled jobs. To learn about how this limit is calculated and what contributes to it see the Lightning Platform Apex Limits topic.

View: All Scheduled Jobs 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**

Schedule Apex

Action	Job Name ↑	Submitted By	Submitted	Started	Next Scheduled Run	Type	Cron Trigger ID
Del	Metalytics Data Loader Job for Org : 00DgK000007bzID	User, Integration	7/17/2025, 8:10 PM	9/24/2025, 10:30 PM	9/25/2025, 10:30 PM	Autonomous Data Loader Job	08egK000007dnmT
	Program Milestone Computation Cron Job	Process, Automated	7/17/2025, 8:10 PM	9/24/2025, 4:59 PM	9/24/2025, 11:59 PM	Program Milestone Computation Cron Job	08egK000007dnmR
	Program Status Update Cron Job	Process, Automated	7/17/2025, 8:10 PM	9/24/2025, 8:00 PM	9/25/2025, 5:00 AM	Program Status Update Cron Job	08egK000007dnmS
Manage | Del | Pause Job	Repair_Batch_Scheduler_Daily	Kumar, Himanshu	9/24/2025, 11:40 PM		9/25/2025, 2:00 AM	Scheduled Apex	08egK00000C5W1y


SETUP
Apex Jobs

Click here to go to the new batch jobs page

Apex Jobs

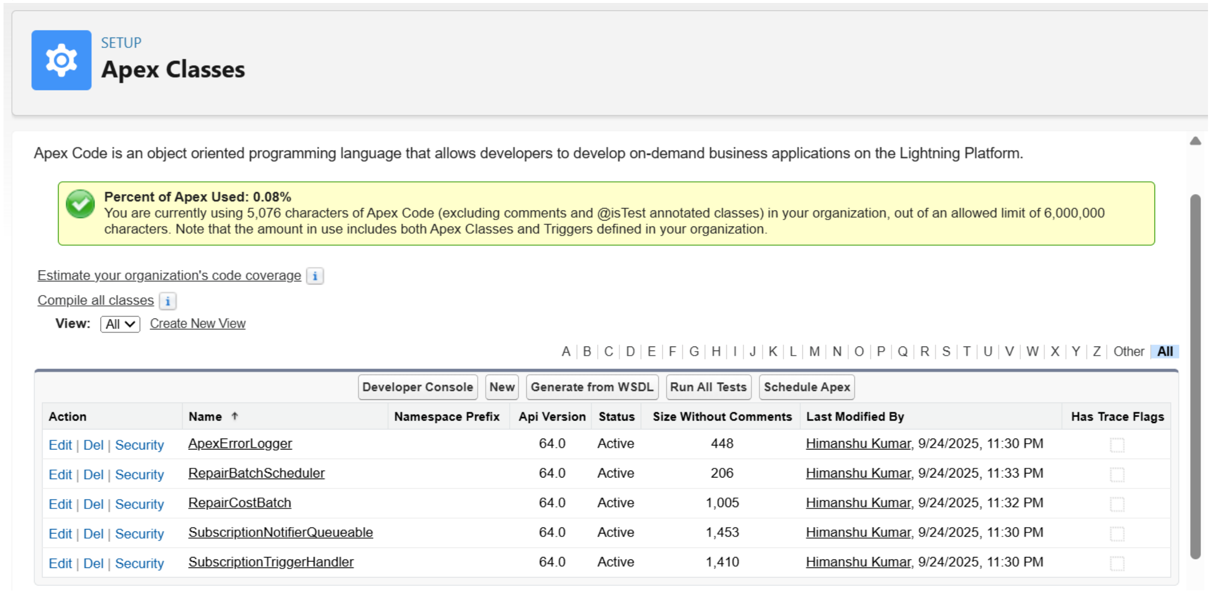
Help for this F

Monitor the status of all Apex jobs, and optionally, abort jobs that are in progress.

Percent of Asynchronous Apex Used: 0%
You have currently used 0 asynchronous Apex operations out of an allowed 24-hour organization limit of 250,000. To learn about how this limit is calculated and what contributes to it, s the Lightning Platform Apex Limits topic.

View: All Create New View

Action	Submitted Date ↓	Job Type	Status	Status Detail	Total Batches	Batches Processed	Failures	Submitted By	Completion Date	Apex Class	Apex Method	Apex Job ID
	9/24/2025, 11:40 PM	Scheduled Apex	Queued		0	0	0	Kumar, Himanshu		RepairBatchScheduler		707gK00000


SETUP
Apex Classes

Apex Code is an object oriented programming language that allows developers to develop on-demand business applications on the Lightning Platform.

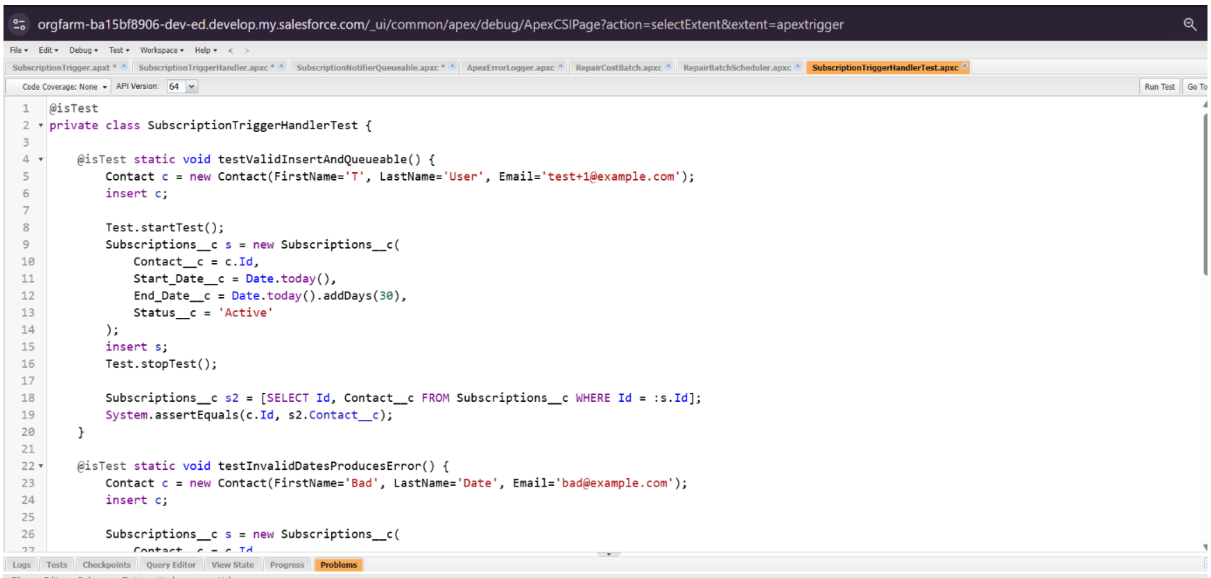
Percent of Apex Used: 0.08%
You are currently using 5,076 characters of Apex Code (excluding comments and @isTest annotated classes) in your organization, out of an allowed limit of 6,000,000 characters. Note that the amount in use includes both Apex Classes and Triggers defined in your organization.

Estimate your organization's code coverage
Compile all classes
View: All Create New View

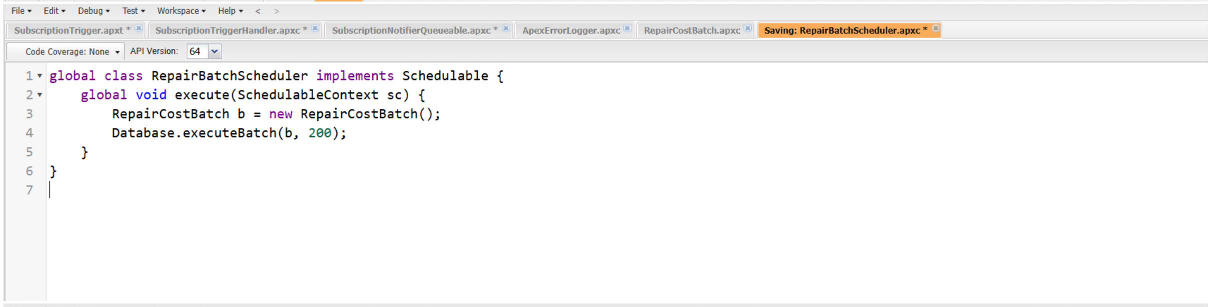
A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Other | **All**

Developer Console | New | Generate from WSDL | Run All Tests | Schedule Apex

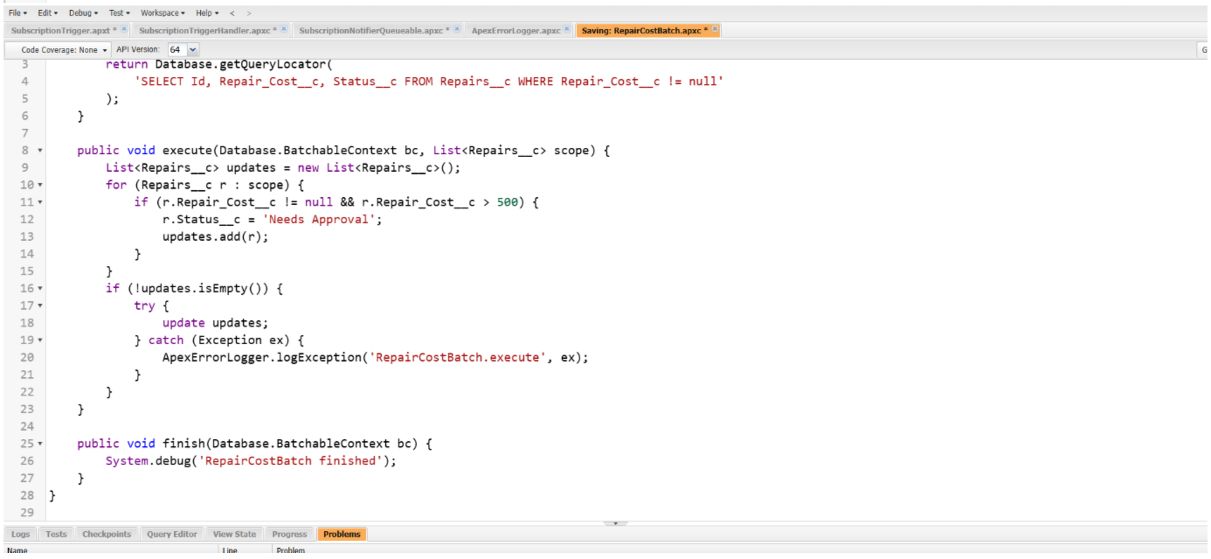
Action	Name ↑	Namespace Prefix	Api Version	Status	Size Without Comments	Last Modified By	Has Trace Flags
Edit | Del | Security	ApexErrorLogger		64.0	Active	448	Himanshu Kumar, 9/24/2025, 11:30 PM	☐
Edit | Del | Security	RepairBatchScheduler		64.0	Active	206	Himanshu Kumar, 9/24/2025, 11:33 PM	☐
Edit | Del | Security	RepairCostBatch		64.0	Active	1,005	Himanshu Kumar, 9/24/2025, 11:32 PM	☐
Edit | Del | Security	SubscriptionNotifierQueueable		64.0	Active	1,453	Himanshu Kumar, 9/24/2025, 11:30 PM	☐
Edit | Del | Security	SubscriptionTriggerHandler		64.0	Active	1,410	Himanshu Kumar, 9/24/2025, 11:30 PM	☐


orgfarm-ba15bf8906-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage?action=selectExtent&extent=apextrigger
File Edit Debug Test Workspace Help
SubscriptionTrigger.apxt * | SubscriptionTriggerHandler.apxc * | SubscriptionNotifierQueueable.apxc * | ApexErrorLogger.apxc | RepairCostBatch.apxc | RepairBatchScheduler.apxc | SubscriptionTriggerHandlerTest.apxc
Code Coverage: None | API Version: 64
Run Test | Go To
Logs | Tests | Checkpoints | Query Editor | View State | Progress | Problems

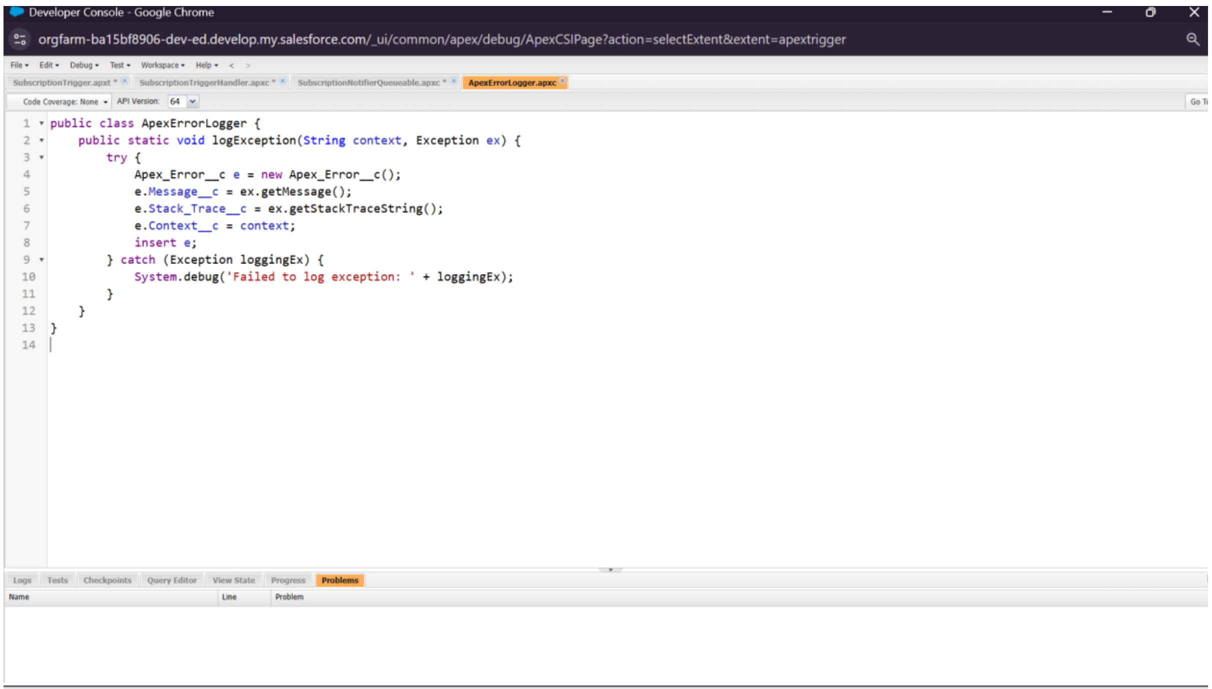
```
@isTest
private class SubscriptionTriggerHandlerTest {

    @isTest static void testValidInsertAndQueueable() {
        Contact c = new Contact(FirstName='T', LastName='User', Email='test+1@example.com');
        insert c;

        Test.startTest();
        Subscriptions__c s = new Subscriptions__c(
            Contact__c = c.Id,
            Start_Date__c = Date.today(),
            End_Date__c = Date.today().addDays(30),
            Status__c = 'Active'
        );
        insert s;
        Test.stopTest();

        Subscriptions__c s2 = [SELECT Id, Contact__c FROM Subscriptions__c WHERE Id = :s.Id];
        System.assertEquals(c.Id, s2.Contact__c);
    }

    @isTest static void testInvalidDatesProducesError() {
        Contact c = new Contact(FirstName='Bad', LastName='Date', Email='bad@example.com');
        insert c;

        Subscriptions__c s = new Subscriptions__c(
            Contact__c = c.Id
```


File Edit Debug Test Workspace Help
SubscriptionTrigger.apxt * | SubscriptionTriggerHandler.apxc * | SubscriptionNotifierQueueable.apxc * | ApexErrorLogger.apxc | RepairCostBatch.apxc | Saving: RepairBatchScheduler.apxc *
Code Coverage: None | API Version: 64

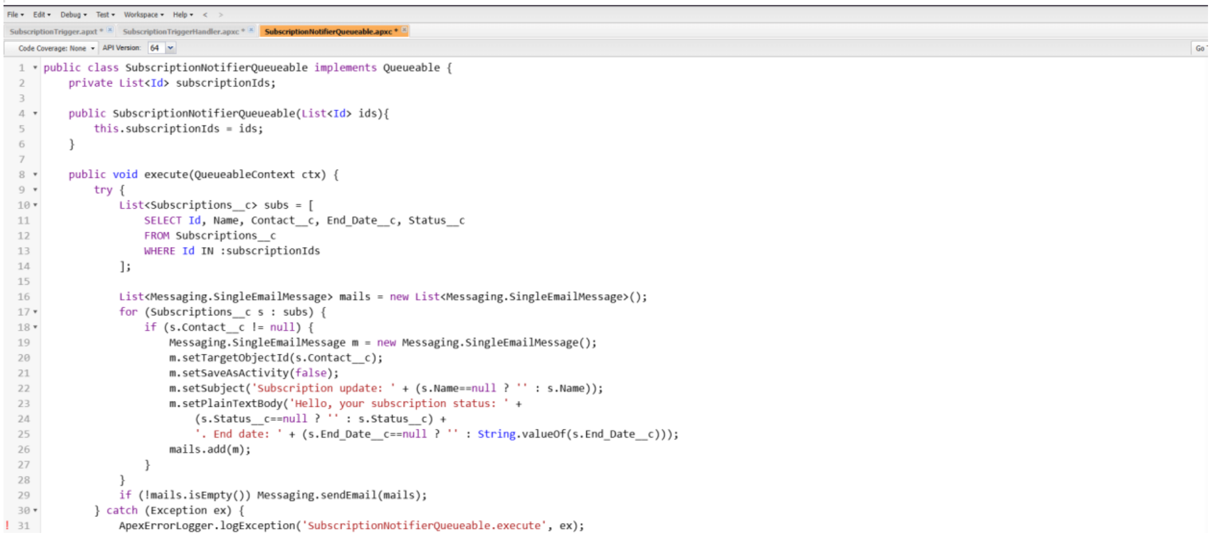
```
global class RepairBatchScheduler implements Schedulable {
    global void execute(SchedulableContext sc) {
        RepairCostBatch b = new RepairCostBatch();
        Database.executeBatch(b, 200);
    }
}
```


File Edit Debug Test Workspace Help
SubscriptionTrigger.apxt * | SubscriptionTriggerHandler.apxc * | SubscriptionNotifierQueueable.apxc * | ApexErrorLogger.apxc | Saving: RepairCostBatch.apxc *
Code Coverage: None | API Version: 64
Logs | Tests | Checkpoints | Query Editor | View State | Progress | Problems
Name | Line | Problem

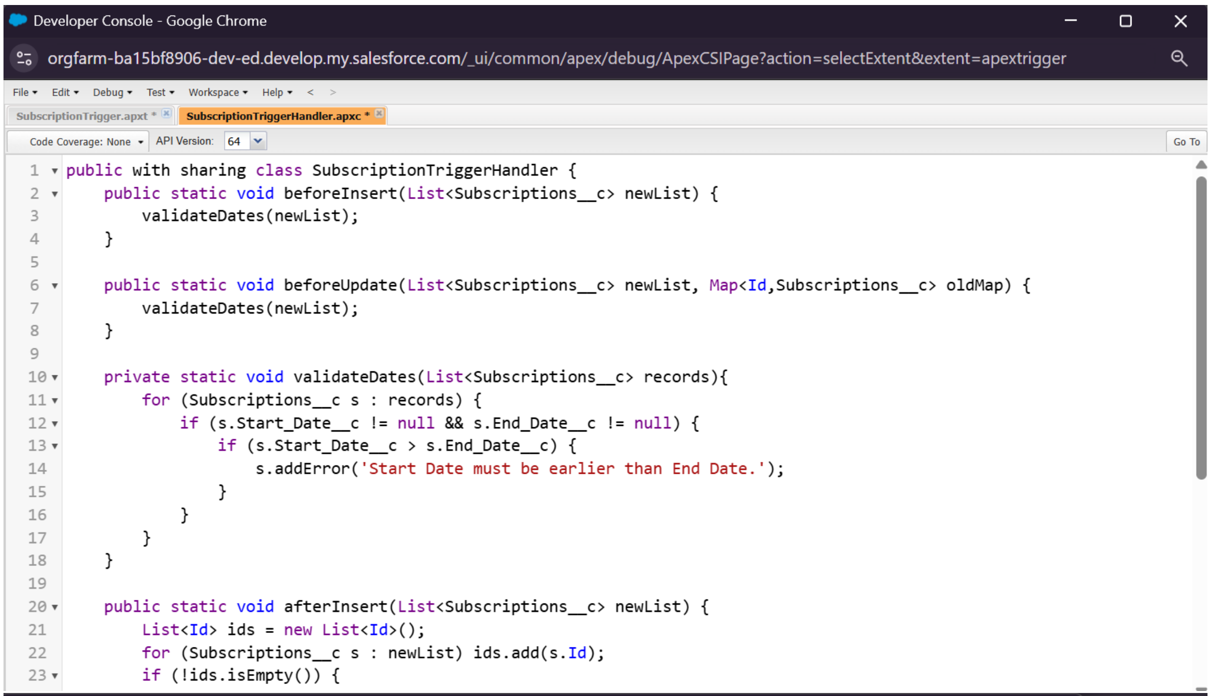
```
        return Database.getQueryLocator(
            'SELECT Id, Repair_Cost__c, Status__c FROM Repairs__c WHERE Repair_Cost__c != null'
        );
    }

    public void execute(Database.BatchableContext bc, List<Repairs__c> scope) {
        List<Repairs__c> updates = new List<Repairs__c>();
        for (Repairs__c r : scope) {
            if (r.Repair_Cost__c != null && r.Repair_Cost__c > 500) {
                r.Status__c = 'Needs Approval';
                updates.add(r);
            }
        }
        if (!updates.isEmpty()) {
            try {
                update updates;
            } catch (Exception ex) {
                ApexErrorLogger.logException('RepairCostBatch.execute', ex);
            }
        }
    }

    public void finish(Database.BatchableContext bc) {
        System.debug('RepairCostBatch finished');
    }
}
```


Developer Console - Google Chrome
orgfarm-ba15bf8906-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage?action=selectExtent&extent=apextrigger
File Edit Debug Test Workspace Help
SubscriptionTrigger.apxt * SubscriptionTriggerHandler.apxc * SubscriptionNotifierQueueable.apxc * ApexErrorLogger.apxc
Code Coverage: None API Version: 64
Logs Tests Checkpoints Query Editor View State Progress Problems
Name Line Problem

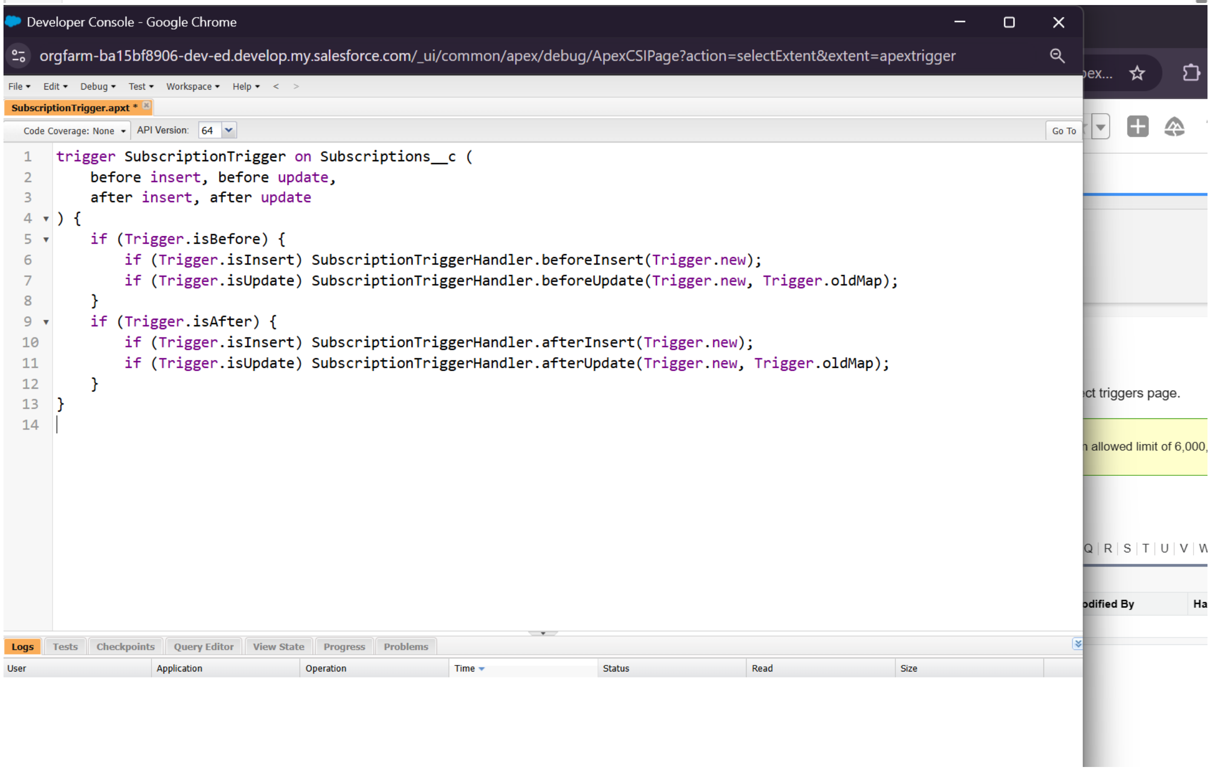
```
public class ApexErrorLogger {
    public static void logException(String context, Exception ex) {
        try {
            Apex_Error__c e = new Apex_Error__c();
            e.Message__c = ex.getMessage();
            e.Stack_Trace__c = ex.getStackTraceString();
            e.Context__c = context;
            insert e;
        } catch (Exception loggingEx) {
            System.debug('Failed to log exception: ' + loggingEx);
        }
    }
}
```


File Edit Debug Test Workspace Help
SubscriptionTrigger.apxt * SubscriptionTriggerHandler.apxc * SubscriptionNotifierQueueable.apxc *
Code Coverage: None API Version: 64

```
public class SubscriptionNotifierQueueable implements Queueable {
    private List<Id> subscriptionIds;

    public SubscriptionNotifierQueueable(List<Id> ids){
        this.subscriptionIds = ids;
    }

    public void execute(QueueableContext ctx) {
        try {
            List<Subscriptions__c> subs = [
                SELECT Id, Name, Contact__c, End_Date__c, Status__c
                FROM Subscriptions__c
                WHERE Id IN :subscriptionIds
            ];

            List<Messaging.SingleEmailMessage> mails = new List<Messaging.SingleEmailMessage>();
            for (Subscriptions__c s : subs) {
                if (s.Contact__c != null) {
                    Messaging.SingleEmailMessage m = new Messaging.SingleEmailMessage();
                    m.setTargetObjectId(s.Contact__c);
                    m.setSaveAsActivity(false);
                    m.setSubject('Subscription update: ' + (s.Name==null ? '' : s.Name));
                    m.setPlainTextBody('Hello, your subscription status: ' +
                        (s.Status__c==null ? '' : s.Status__c) +
                        '. End date: ' + (s.End_Date__c==null ? '' : String.valueOf(s.End_Date__c)));
                    mails.add(m);
                }
            }
            if (!mails.isEmpty()) Messaging.sendEmail(mails);
        } catch (Exception ex) {
            ApexErrorLogger.logException('SubscriptionNotifierQueueable.execute', ex);
```


Developer Console - Google Chrome
orgfarm-ba15bf8906-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage?action=selectExtent&extent=apextrigger
File Edit Debug Test Workspace Help
SubscriptionTrigger.apxt *
SubscriptionTriggerHandler.apxc *
Code Coverage: None
API Version: 64
Go To

```
public with sharing class SubscriptionTriggerHandler {
    public static void beforeInsert(List<Subscriptions__c> newList) {
        validateDates(newList);
    }

    public static void beforeUpdate(List<Subscriptions__c> newList, Map<Id,Subscriptions__c> oldMap) {
        validateDates(newList);
    }

    private static void validateDates(List<Subscriptions__c> records){
        for (Subscriptions__c s : records) {
            if (s.Start_Date__c != null && s.End_Date__c != null) {
                if (s.Start_Date__c > s.End_Date__c) {
                    s.addError('Start Date must be earlier than End Date.');
                }
            }
        }
    }

    public static void afterInsert(List<Subscriptions__c> newList) {
        List<Id> ids = new List<Id>();
        for (Subscriptions__c s : newList) ids.add(s.Id);
        if (!ids.isEmpty()) {
```


Developer Console - Google Chrome
orgfarm-ba15bf8906-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage?action=selectExtent&extent=apextrigger
File Edit Debug Test Workspace Help
SubscriptionTrigger.apxt *
Code Coverage: None
API Version: 64
Go To
Logs
Tests
Checkpoints
Query Editor
View State
Progress
Problems
User
Application
Operation
Time
Status
Read
Size

```
trigger SubscriptionTrigger on Subscriptions__c (
    before insert, before update,
    after insert, after update
) {
    if (Trigger.isBefore) {
        if (Trigger.isInsert) SubscriptionTriggerHandler.beforeInsert(Trigger.new);
        if (Trigger.isUpdate) SubscriptionTriggerHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
    }
    if (Trigger.isAfter) {
        if (Trigger.isInsert) SubscriptionTriggerHandler.afterInsert(Trigger.new);
        if (Trigger.isUpdate) SubscriptionTriggerHandler.afterUpdate(Trigger.new, Trigger.oldMap);
    }
}
```