

Project Title: Sales Performance Reporting

Phase 5: Apex Programming (Developer)

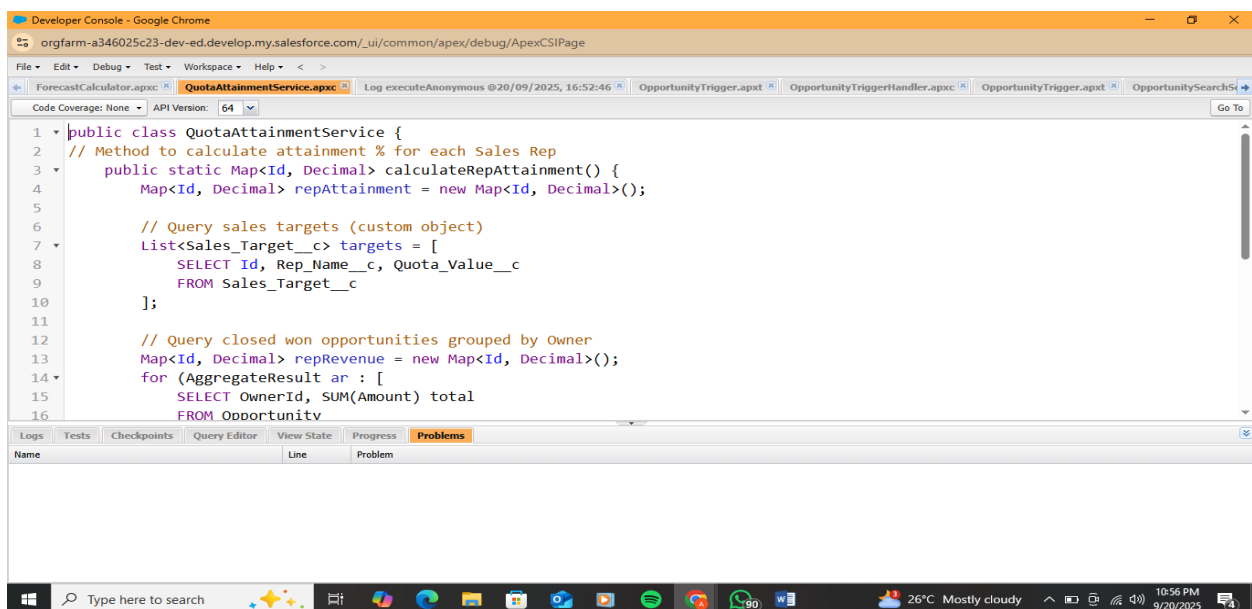
Objective:

Phase 5 focused on extending Salesforce functionality beyond declarative tools by leveraging **Apex programming**. Apex allowed us to implement advanced logic, process large volumes of data, run background jobs, and handle complex scenarios that standard automation tools (like Flows and Workflows) could not manage.

1. Apex Classes

Two main service classes were developed:

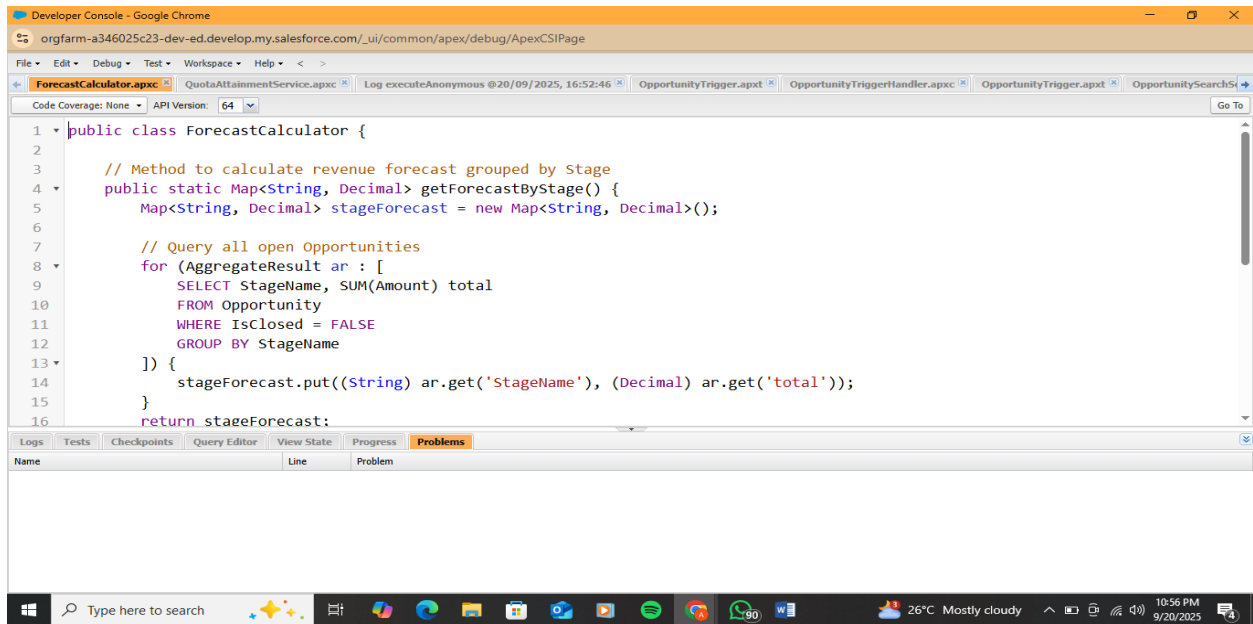
- **ForecastCalculator.cls** → Calculates revenue forecasts grouped by Stage and Region using SOQL.
- **QuotaAttainmentService.cls** → Compares Closed Won revenue against individual Sales Rep quotas stored in the SalesTarget__c custom object and calculates attainment %.



The screenshot displays the Salesforce Developer Console in Google Chrome. The browser address bar shows the URL: `orgfarm-a346025c23-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage`. The console interface includes a menu bar (File, Edit, Debug, Test, Workspace, Help), a toolbar with options like 'Log executeAnonymous', and a code editor. The code editor shows the following Apex code for the `QuotaAttainmentService` class:

```
1 public class QuotaAttainmentService {
2     // Method to calculate attainment % for each Sales Rep
3     public static Map<Id, Decimal> calculateRepAttainment() {
4         Map<Id, Decimal> repAttainment = new Map<Id, Decimal>();
5
6         // Query sales targets (custom object)
7         List<Sales_Target__c> targets = [
8             SELECT Id, Rep_Name__c, Quota_Value__c
9             FROM Sales_Target__c
10        ];
11
12        // Query closed won opportunities grouped by Owner
13        Map<Id, Decimal> repRevenue = new Map<Id, Decimal>();
14        for (AggregateResult ar : [
15            SELECT OwnerId, SUM(Amount) total
16            FROM Opportunity
```

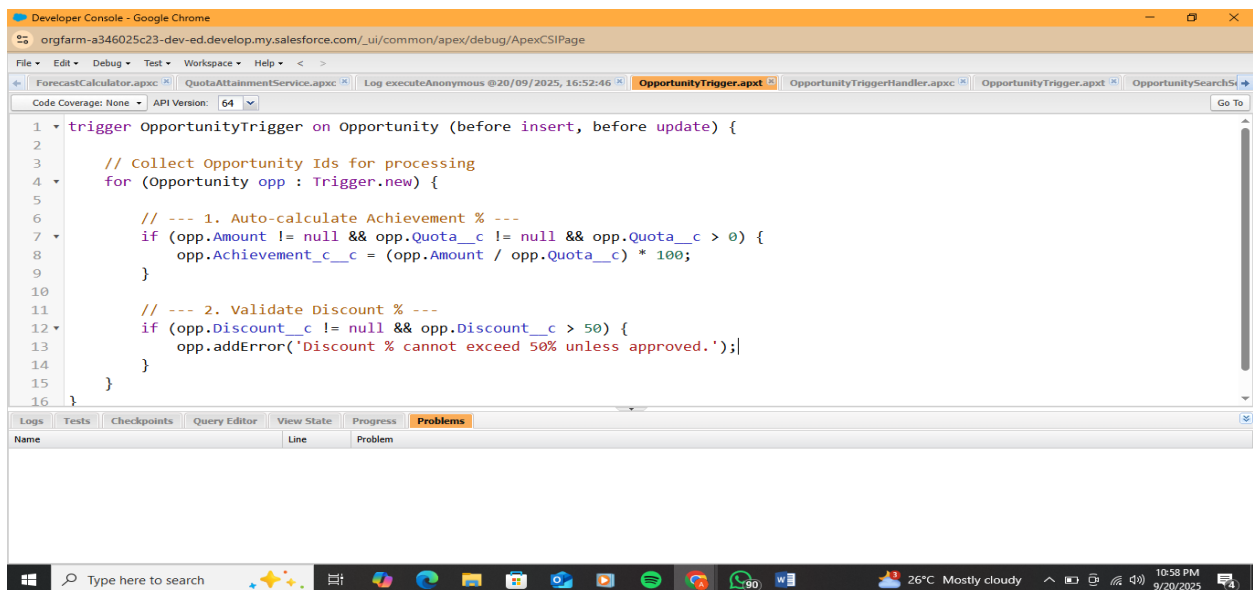
Below the code editor, there are tabs for 'Logs', 'Tests', 'Checkpoints', 'Query Editor', 'View State', 'Progress', and 'Problems'. The 'Problems' tab is currently selected, showing a table with columns 'Name', 'Line', and 'Problem'. The Windows taskbar at the bottom indicates the system time as 10:56 PM on 9/20/2025, with a temperature of 26°C and weather 'Mostly cloudy'.



2. Apex Triggers

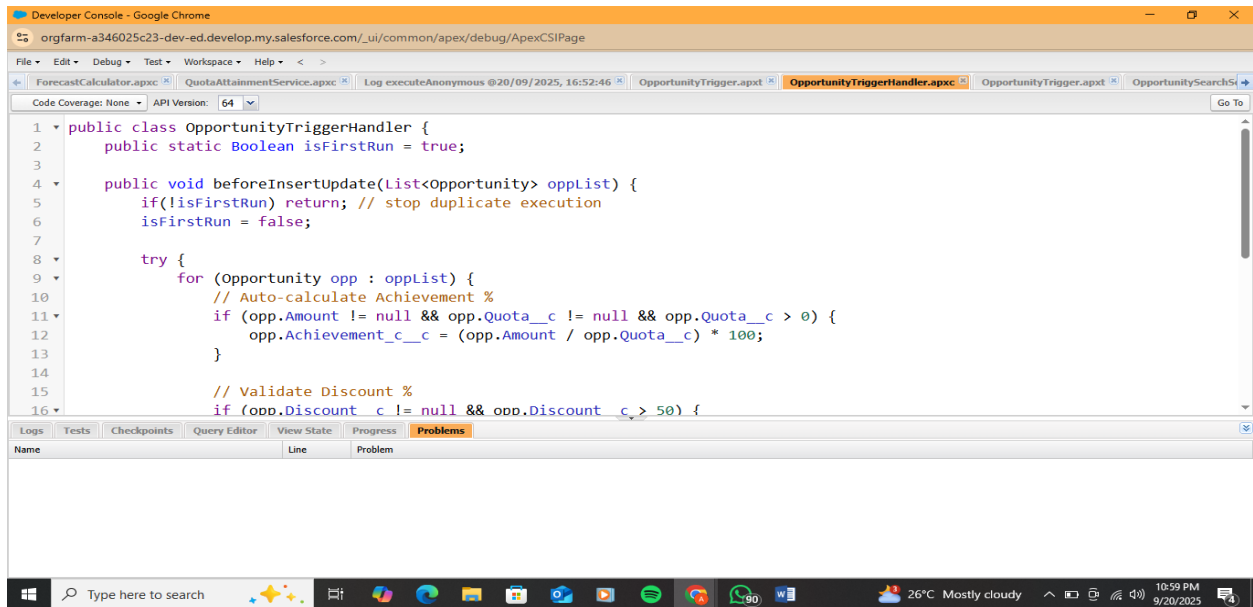
A trigger was created on the **Opportunity** object:

- Auto-calculates **Achievement %** whenever Amount and Quota are filled.
- Prevents users from entering a **Discount % > 50** unless formally approved.



3. Trigger Design Pattern

Instead of placing all logic inside the trigger, a **Trigger Handler class** (OpportunityTriggerHandler) was implemented. The trigger simply delegates work to the handler.



The screenshot shows the Salesforce Developer Console with the OpportunityTriggerHandler class open. The class contains the following code:

```
1 public class OpportunityTriggerHandler {
2     public static Boolean isFirstRun = true;
3
4     public void beforeInsertUpdate(List<Opportunity> oppList) {
5         if(!isFirstRun) return; // stop duplicate execution
6         isFirstRun = false;
7
8         try {
9             for (Opportunity opp : oppList) {
10                // Auto-calculate Achievement %
11                if (opp.Amount != null && opp.Quota__c != null && opp.Quota__c > 0) {
12                    opp.Achievement__c = (opp.Amount / opp.Quota__c) * 100;
13                }
14
15                // Validate Discount %
16                if (opp.Discount__c != null && opp.Discount__c > 50) {
```

The console also shows a table with columns 'Name', 'Line', and 'Problem'.

4. SOQL & SOSL

- **SOQL:** Used to aggregate sales pipeline by Stage and Region.
- **SOSL:** Implemented to search Accounts and Opportunities by keyword across multiple objects.

The screenshot shows the Salesforce Developer Console with the 'OpportunitySearchService.apxc' file open. The code defines a class with a static method that uses SOQL to fetch opportunities grouped by stage. The interface includes tabs for various files, a toolbar with 'Code Coverage' and 'API Version' settings, and a 'Problems' panel at the bottom.

```
1 public class OpportunitySearchService {
2
3     // SOQL Example: Fetch Opportunities grouped by Stage
4     public static Map<String, Decimal> getOpportunitiesByStage() {
5         Map<String, Decimal> results = new Map<String, Decimal>();
6
7         for (AggregateResult ar : [
8             SELECT StageName, SUM(Amount) total
9             FROM Opportunity
10            WHERE IsClosed = FALSE
11            GROUP BY StageName
12        ]) {
13            results.put((String)ar.get('StageName'), (Decimal)ar.get('total'));
14        }
15        return results;
16    }
```

5. Collections (List, Set, Map)

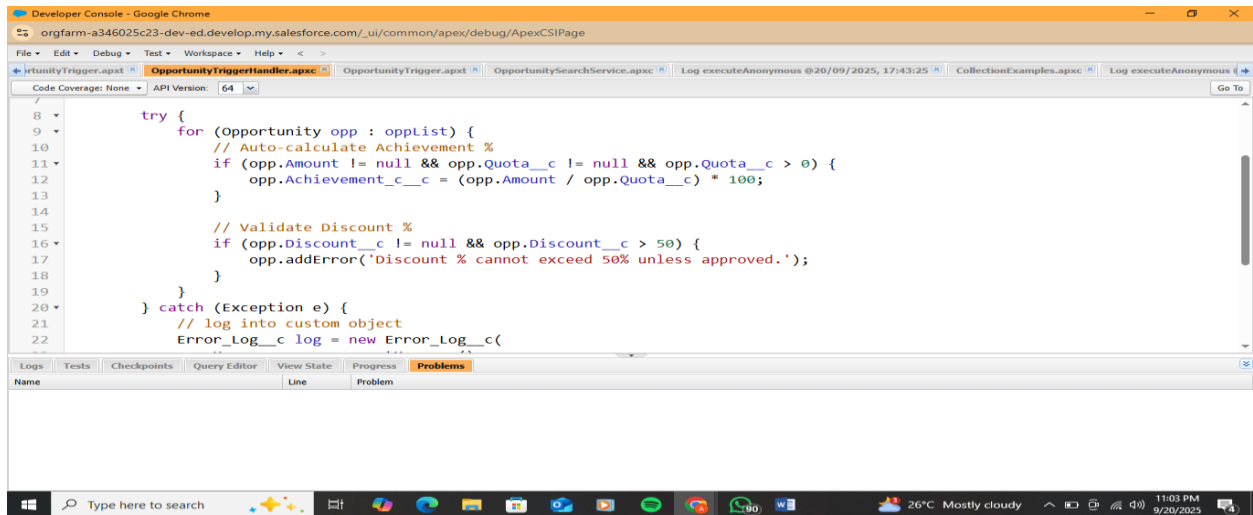
- **List:** Stored multiple Opportunities in memory.
- **Set:** Ensured only unique Account Ids were processed.
- **Map:** Linked Users with their assigned quota records.

The screenshot shows the Salesforce Developer Console with the 'CollectionExamples.apxc' file open. The code defines a class with a static method that uses SOQL to fetch open opportunities and then iterates through them using a for loop. The interface is similar to the previous screenshot, showing the same tabs and toolbar.

```
1 public class CollectionExamples {
2
3     public static void calculateAchievement() {
4
5         // Get Open Opportunities
6         List<Opportunity> opps = [
7             SELECT Id, Name, Amount, Quota__c
8             FROM Opportunity
9             WHERE IsClosed = FALSE
10        ];
11
12        // Loop through each opportunity
13        for (Opportunity opp : opps) {
14
15            // If-Else statement
16            if (opp.Quota__c != null && opp.Quota__c > 0) {
```

6. Control Statements

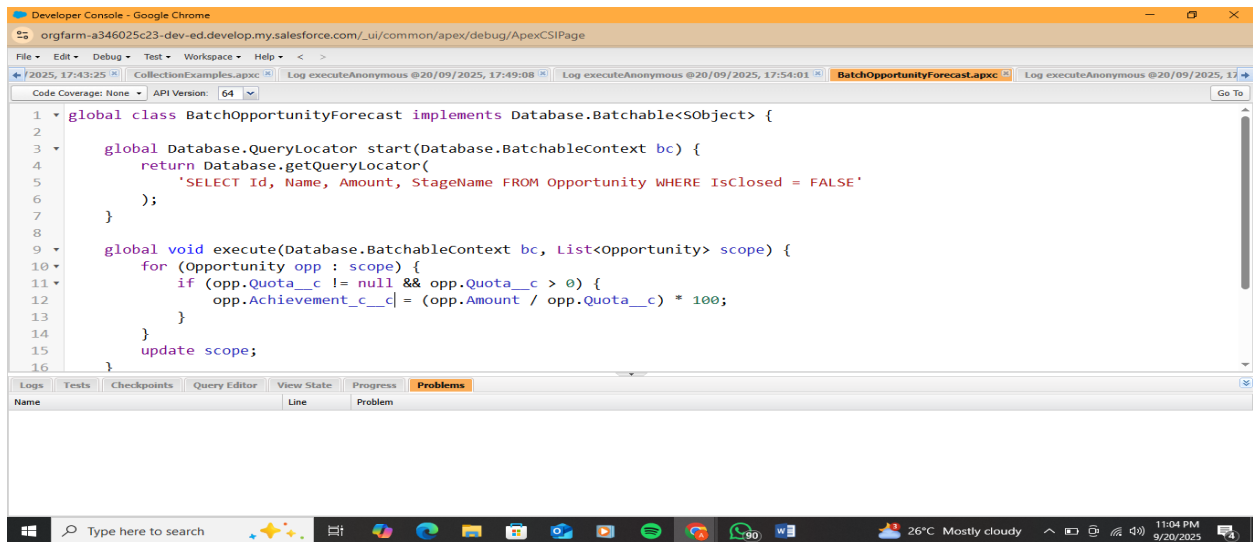
- **For Loops:** Iterated over Opportunities for calculations.
- **If-Else Conditions:** Checked quota values before calculating Achievement %.



```
8      try {
9          for (Opportunity opp : opplist) {
10             // Auto-calculate Achievement %
11             if (opp.Amount != null && opp.Quota__c != null && opp.Quota__c > 0) {
12                 opp.Achievement__c = (opp.Amount / opp.Quota__c) * 100;
13             }
14
15             // Validate Discount %
16             if (opp.Discount__c != null && opp.Discount__c > 50) {
17                 opp.addError('Discount % cannot exceed 50% unless approved.');
```

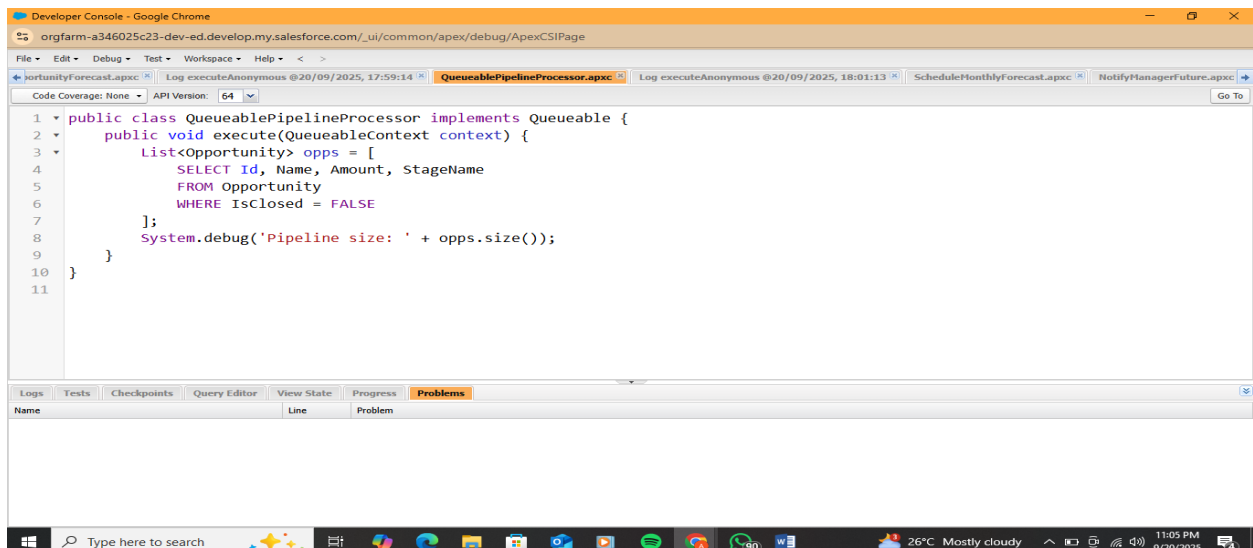
7. Batch Apex

A **Batch Class** (BatchOpportunityForecast) was created to process Opportunities in chunks of 200. It recalculates Achievement % for large data volumes at once.



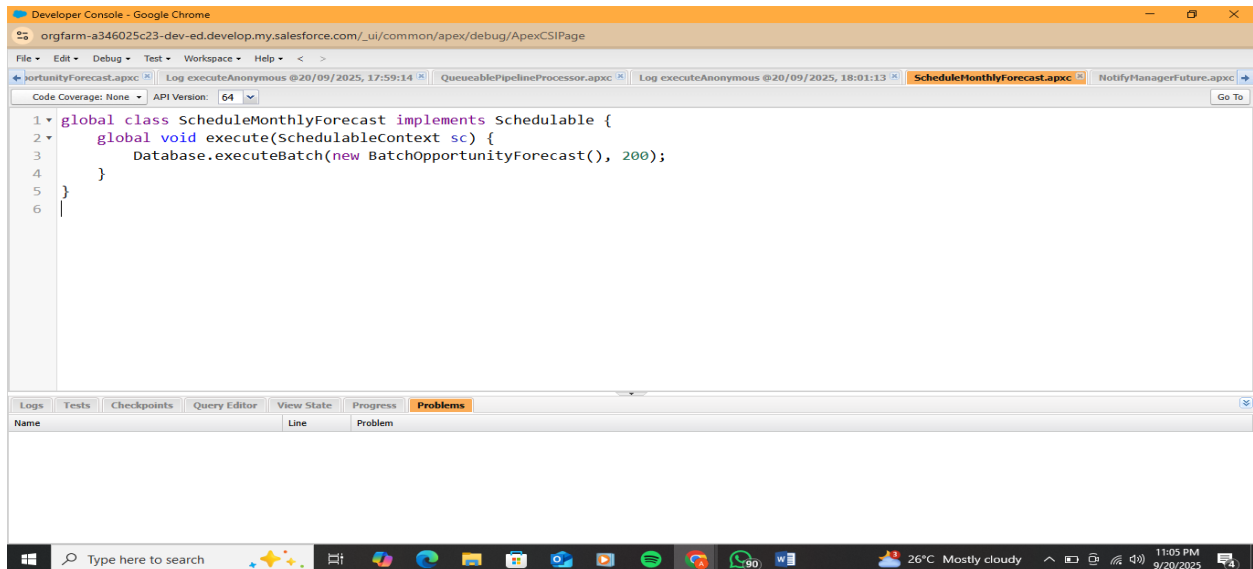
8. Queueable Apex

A **Queueable Class** (`QueueablePipelineProcessor`) processed live pipeline data asynchronously.



9. Scheduled Apex

A **Scheduler Class** (ScheduleMonthlyForecast) was created to run the batch job automatically on the 1st of every month at 12:00 AM.

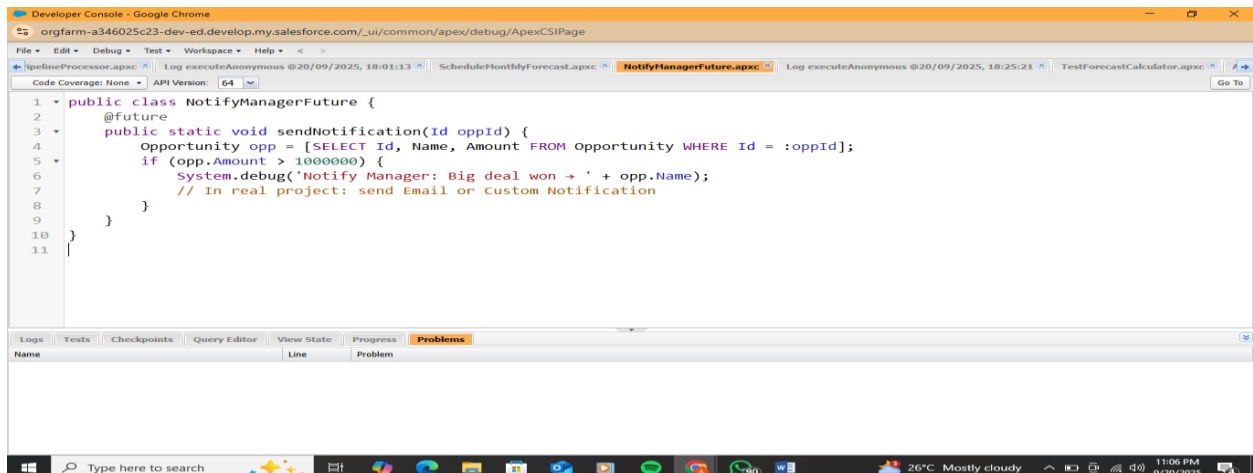


The screenshot shows the Salesforce Developer Console with the 'ScheduleMonthlyForecast.apxc' file open. The code defines a global class that implements the Schedulable interface. The execute method calls Database.executeBatch with a new BatchOpportunityForecast object and a batch size of 200.

```
1 global class ScheduleMonthlyForecast implements Schedulable {
2     global void execute(SchedulableContext sc) {
3         Database.executeBatch(new BatchOpportunityForecast(), 200);
4     }
5 }
6
```

10. Future Methods

A **Future Class** (NotifyManagerFuture) was built to send notifications asynchronously when a large Opportunity (> ₹10 Lakhs) is won.

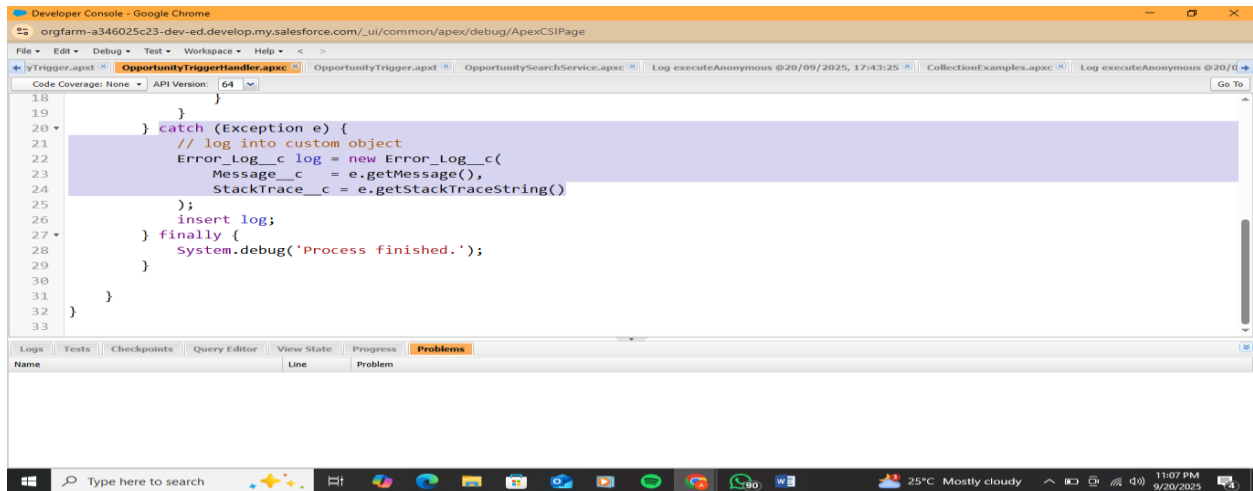


The screenshot shows the Salesforce Developer Console with the 'NotifyManagerFuture.apxc' file open. The code defines a public class with a @future annotation and a static void method sendNotification. This method queries for an Opportunity by ID, checks if the amount is greater than 10,000,000, and then logs a debug message. A comment indicates that in a real project, an email or custom notification would be sent.

```
1 public class NotifyManagerFuture {
2     @future
3     public static void sendNotification(Id oppId) {
4         Opportunity opp = [SELECT Id, Name, Amount FROM Opportunity WHERE Id = :oppId];
5         if (opp.Amount > 10000000) {
6             System.debug('Notify Manager: Big deal won → ' + opp.Name);
7             // In real project: send Email or custom Notification
8         }
9     }
10 }
11
```

11. Exception Handling

A custom object **Error_Log__c** was created with fields **Message__c** and **StackTrace__c**. In all Apex classes, try-catch-finally blocks were implemented to log exceptions.

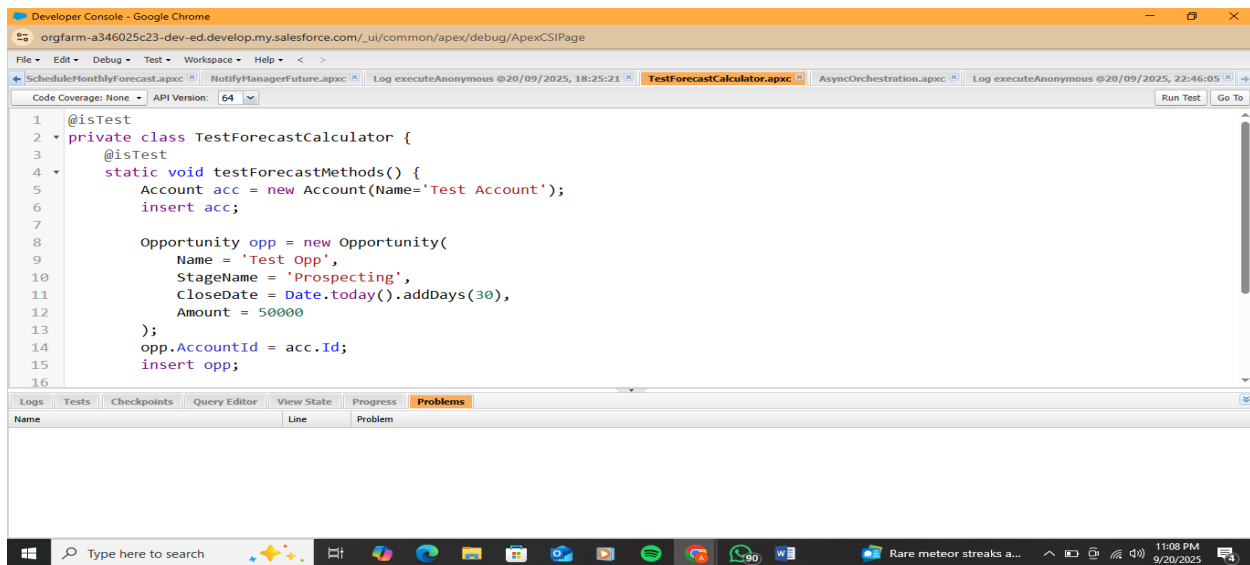


12. Test Classes

Dedicated test classes were written for each Apex component:

- TestForecastCalculator
- TestOpportunityTrigger
- TestBatchOpportunityForecast
- TestQueueablePipelineProcessor

All achieved **≥80% code coverage**, meeting Salesforce deployment requirements.



13. Asynchronous Orchestration

An **AsyncOrchestration** class was developed to combine Batch, Queueable, and Future executions in one command.

