# 🌾 Phase 5: Apex Programming (Developer)

**🧭 What is Apex?**

Apex is Salesforce’s own programming language — similar to Java.  
You use it when **Flows or Process Builder cannot do something** — for example:

* Updating hundreds of records at once
* Performing complex calculations
* Sending automatic messages after records are saved

In your project, Apex helps:

* Auto-update Buyer or Farmer information
* Schedule monthly reminders
* Handle large data updates
* Send background notifications or messages

**⚙️ 1. Classes & Objects**

**📘 What is a Class?**

A class is a *container* for logic (like a box that stores instructions).

**📗 What is an Object?**

An object is a *data table* — like **Farmer\_\_c**, **Buyer\_\_c**, **Order\_\_c**, **Subsidy\_Application\_\_c**.

**💡 Example:**

Let’s calculate the total value of all orders for one farmer.

public class FarmerService {

public static Decimal getTotalOrderAmount(Id farmerId) {

Decimal total = 0;

List<Order\_\_c> orders = [SELECT Amount\_\_c FROM Order\_\_c WHERE Farmer\_\_c = :farmerId];

for(Order\_\_c o : orders) {

total += o.Amount\_\_c;

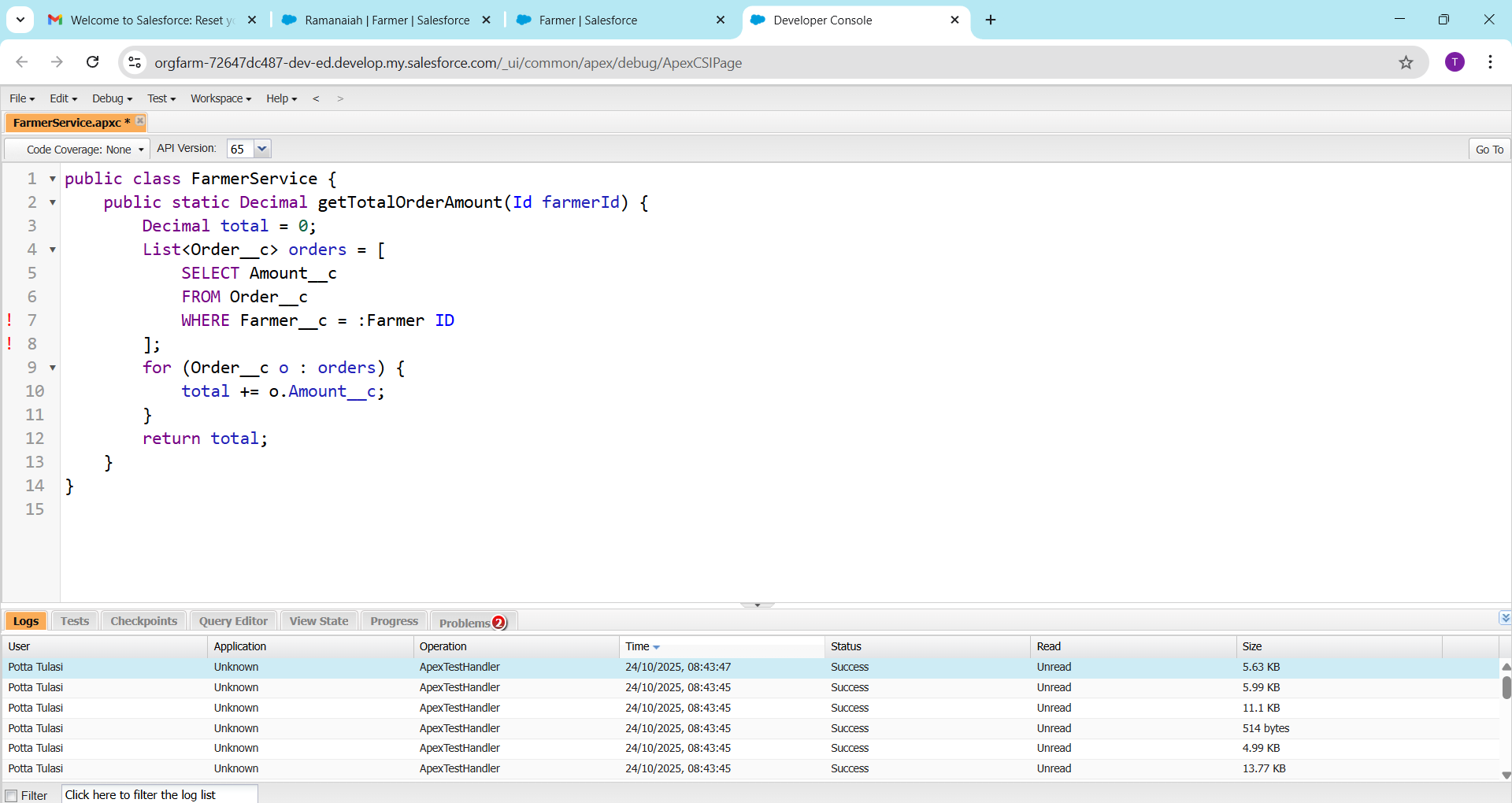
}

return total;

}

}

🟢 This means:  
When you pass a Farmer ID → Apex looks up all that farmer’s orders → adds the total → and gives you the result.



**⚡ 2. Apex Triggers**

**📘 What is a Trigger?**

A Trigger is code that runs **automatically** when data changes in Salesforce.

It can run:

* **Before Insert / Update / Delete**
* **After Insert / Update / Delete**

**💡 Example:**

When a new Order is created → update the Buyer’s “Last Order Date”.

trigger UpdateBuyerDate on Order\_\_c (after insert) {

Set<Id> buyerIds = new Set<Id>();

for(Order\_\_c o : Trigger.new){

if(o.Buyer\_\_c != null){

buyerIds.add(o.Buyer\_\_c);

}

}

List<Buyer\_\_c> buyers = [SELECT Id, Last\_Order\_Date\_\_c FROM Buyer\_\_c WHERE Id IN :buyerIds];

for(Buyer\_\_c b : buyers){

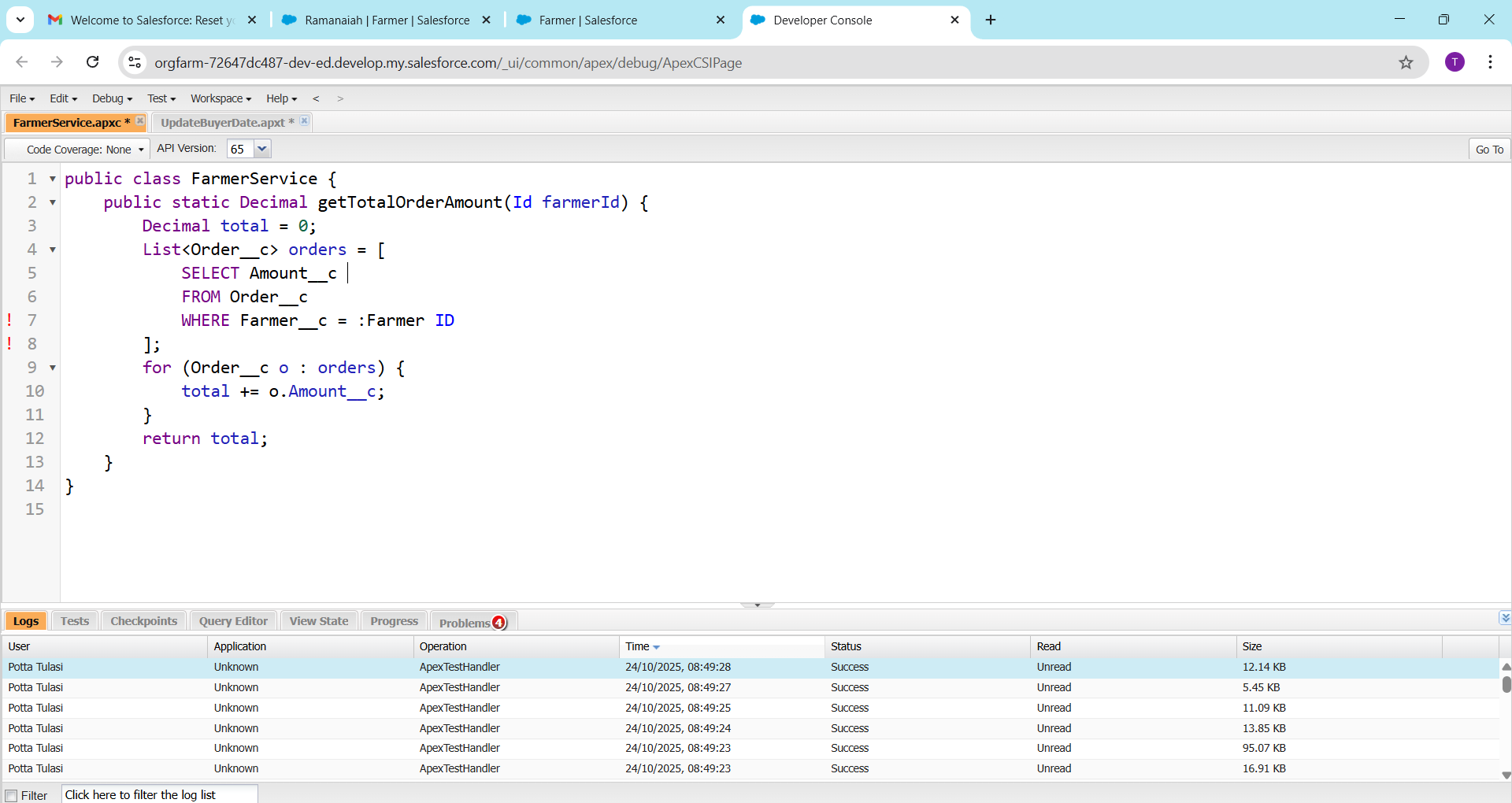
b.Last\_Order\_Date\_\_c = System.today();

}

update buyers;

}

🟢 This means:  
Whenever a new Order is saved → Apex finds who the Buyer is → updates Buyer’s “Last Order Date” automatically.



**🧠 3. Trigger Design Pattern**

Instead of writing long logic inside the trigger, we **move the code to a separate class** (for neatness).

**Trigger File:**

trigger OrderTrigger on Order\_\_c (after insert) {

OrderHandler.afterInsert(Trigger.new);

}

**Handler Class:**

public class OrderHandler {

public static void afterInsert(List<Order\_\_c> orders) {

// Your logic here

}

}

🟢 This makes code easier to read and reuse.

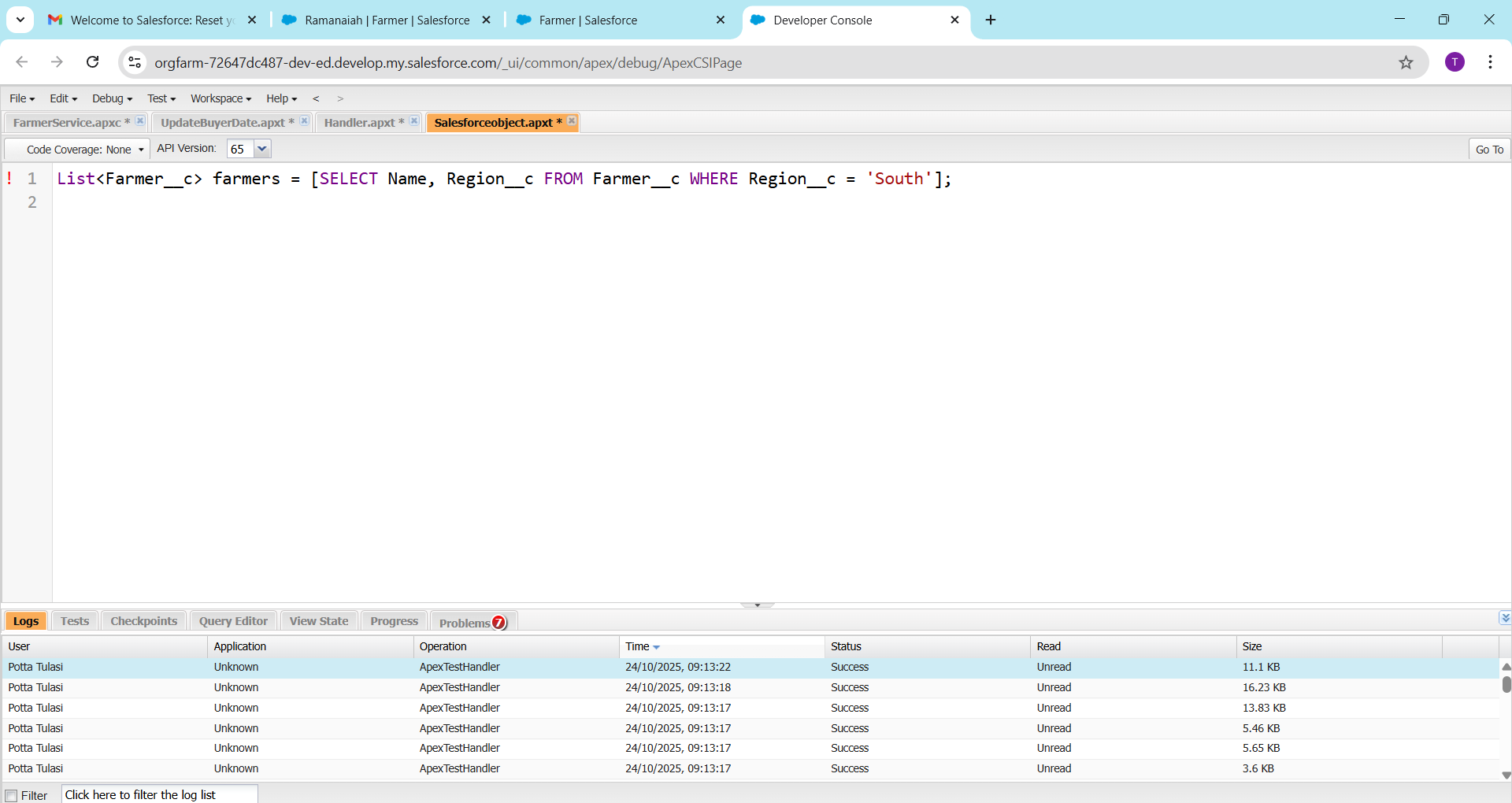
**🔎 4. SOQL and SOSL**

**SOQL (Salesforce Object Query Language)**

Used to **get data from one object**.

List<Farmer\_\_c> farmers = [SELECT Name, Region\_\_c FROM Farmer\_\_c WHERE Region\_\_c = 'South'];

🟢 Meaning: Get all farmers from the “South” region.



**SOSL (Salesforce Object Search Language)**

Used to **search text in multiple objects**.

List<List<SObject>> result = [FIND 'Ravi' IN ALL FIELDS RETURNING Farmer\_\_c(Name), Buyer\_\_c(Name)];

🟢 Meaning: Find “Ravi” in both Farmer and Buyer records.

**📦 5. Collections**

Apex can store multiple records in memory using collections.

| **Type** | **Meaning** | **Example** |
| --- | --- | --- |
| **List** | Ordered list of records | List<Order\_\_c> orders = new List<Order\_\_c>(); |
| **Set** | Unique values only | Set<Id> ids = new Set<Id>(); |
| **Map** | Key → Value pairs | Map<Id, Farmer\_\_c> farmerMap = new Map<Id, Farmer\_\_c>(farmers); |

🟢 In your project: You use a **Set** to store unique Buyer IDs before updating them.

**🔁 6. Control Statements**

These are basic programming conditions — “if”, “for”, “while”.

for(Order\_\_c o : orders){

if(o.Amount\_\_c > 5000){

o.Status\_\_c = 'High Value';

} else {

o.Status\_\_c = 'Normal';

}

}

update orders;

🟢 Meaning:  
If an order is above ₹5000 → mark it as *High Value*, else *Normal*.

**⚙️ 7. Batch Apex**

Used when you have **thousands of records** to update.  
Salesforce runs them in **chunks (batches)** to prevent system overload.

global class UpdateFarmerBatch implements Database.Batchable<SObject> {

global Database.QueryLocator start(Database.BatchableContext bc) {

return Database.getQueryLocator('SELECT Id FROM Farmer\_\_c');

}

global void execute(Database.BatchableContext bc, List<Farmer\_\_c> farmers) {

for(Farmer\_\_c f : farmers){

f.Last\_Updated\_\_c = System.today();

}

update farmers;

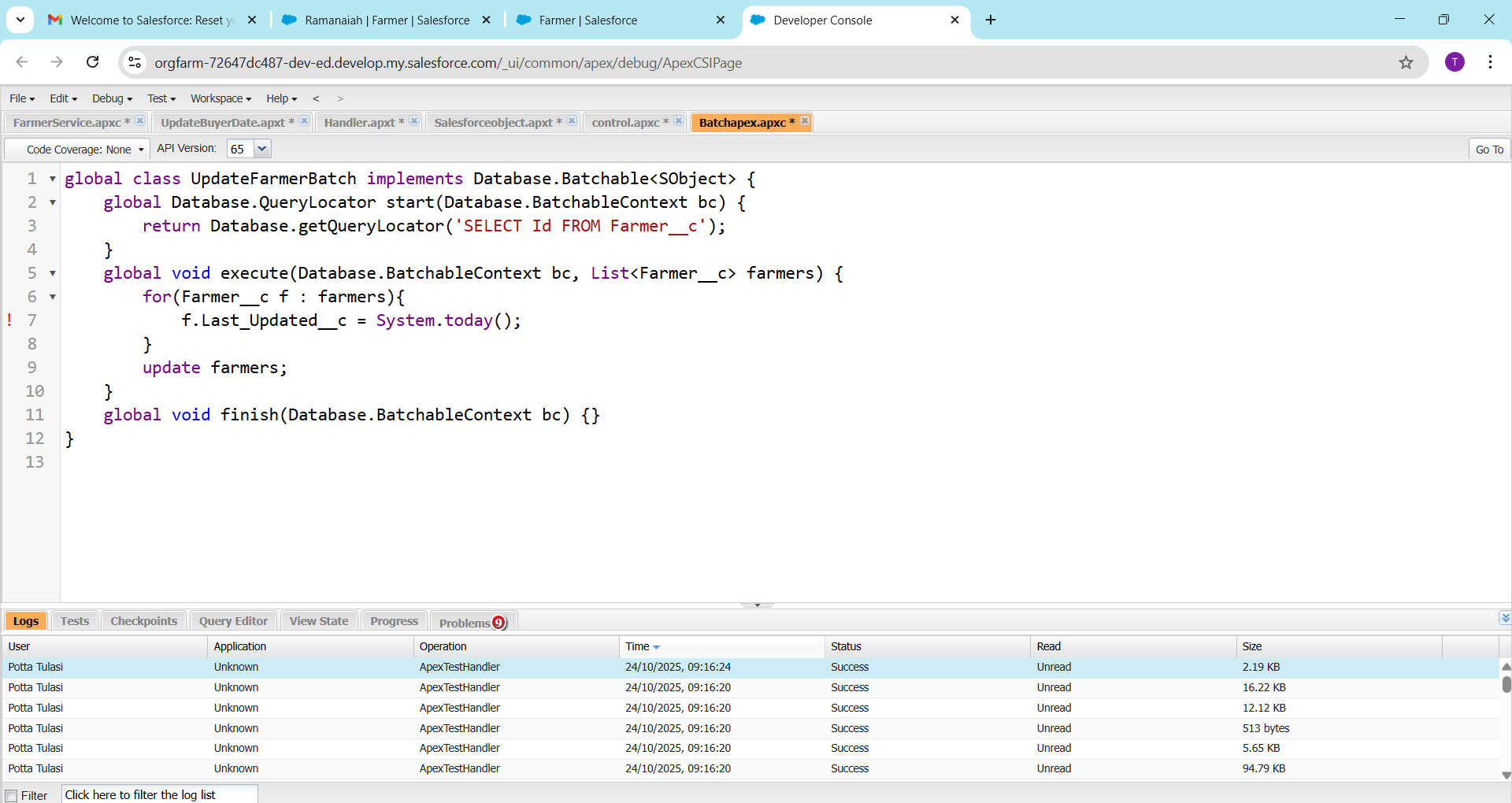
}

global void finish(Database.BatchableContext bc) {}

}

🟢 You can run this batch to refresh farmer info:

Database.executeBatch(new UpdateFarmerBatch(), 100);



**🧩 8. Queueable Apex**

Used for **background jobs** that can be started by system logic.

public class OrderQueueable implements Queueable {

public void execute(QueueableContext qc) {

List<Order\_\_c> orders = [SELECT Id, Status\_\_c FROM Order\_\_c WHERE Status\_\_c='Pending'];

for(Order\_\_c o : orders){

o.Status\_\_c = 'Processed';

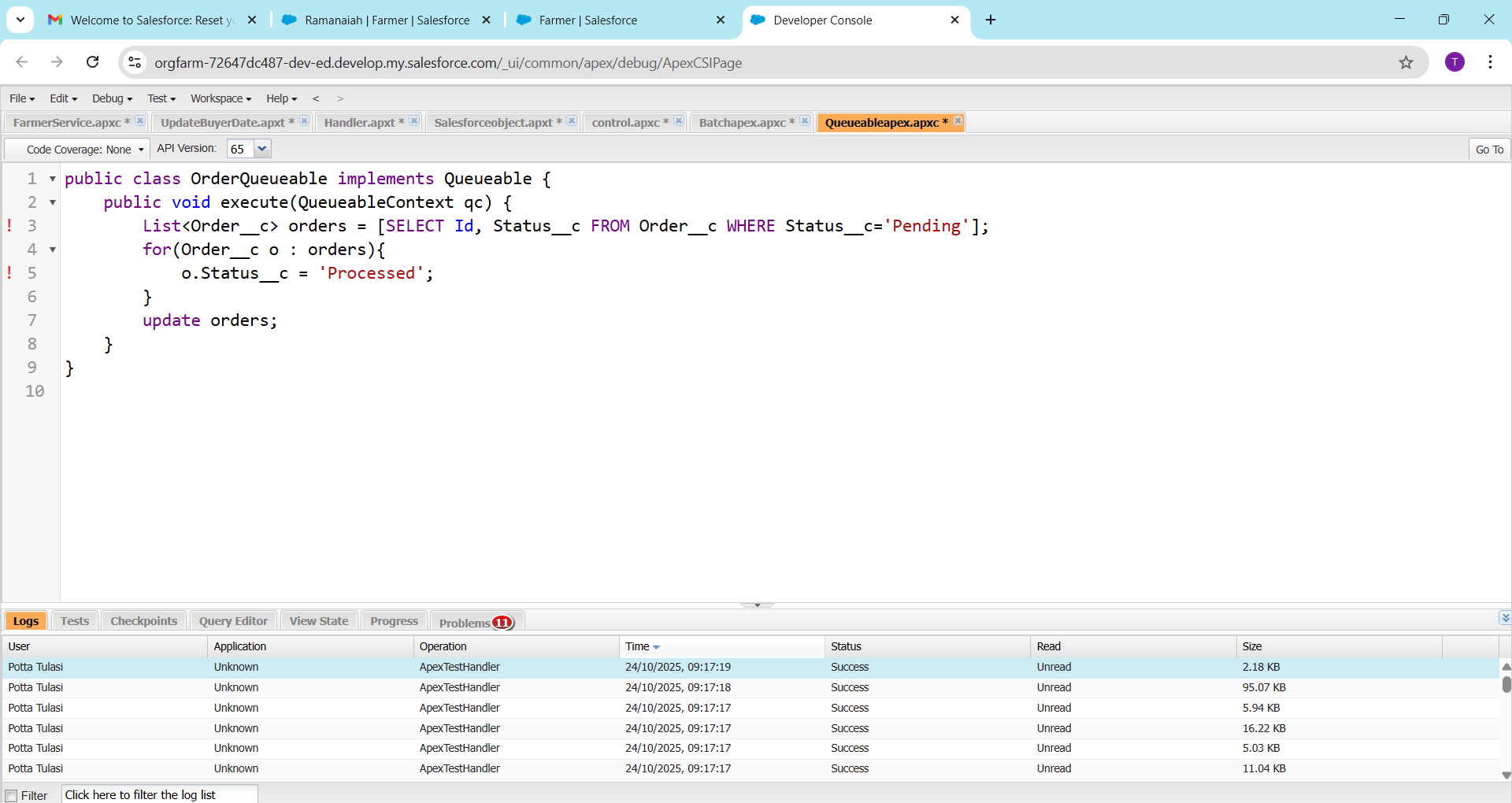
}

update orders;

}

}

🟢 This runs in the background — updating pending orders automatically.



**⏰ 9. Scheduled Apex**

Runs Apex code **at a fixed time** (daily, weekly, monthly).

global class MonthlyReminder implements Schedulable {

global void execute(SchedulableContext sc) {

System.enqueueJob(new OrderQueueable());

}

}

🟢 Schedule it to run on the 1st of every month at 9 AM.

**🚀 10. Future Methods**

Used to run **small tasks in the background**, like sending SMS or API calls.

public class SMSUtility {

@future

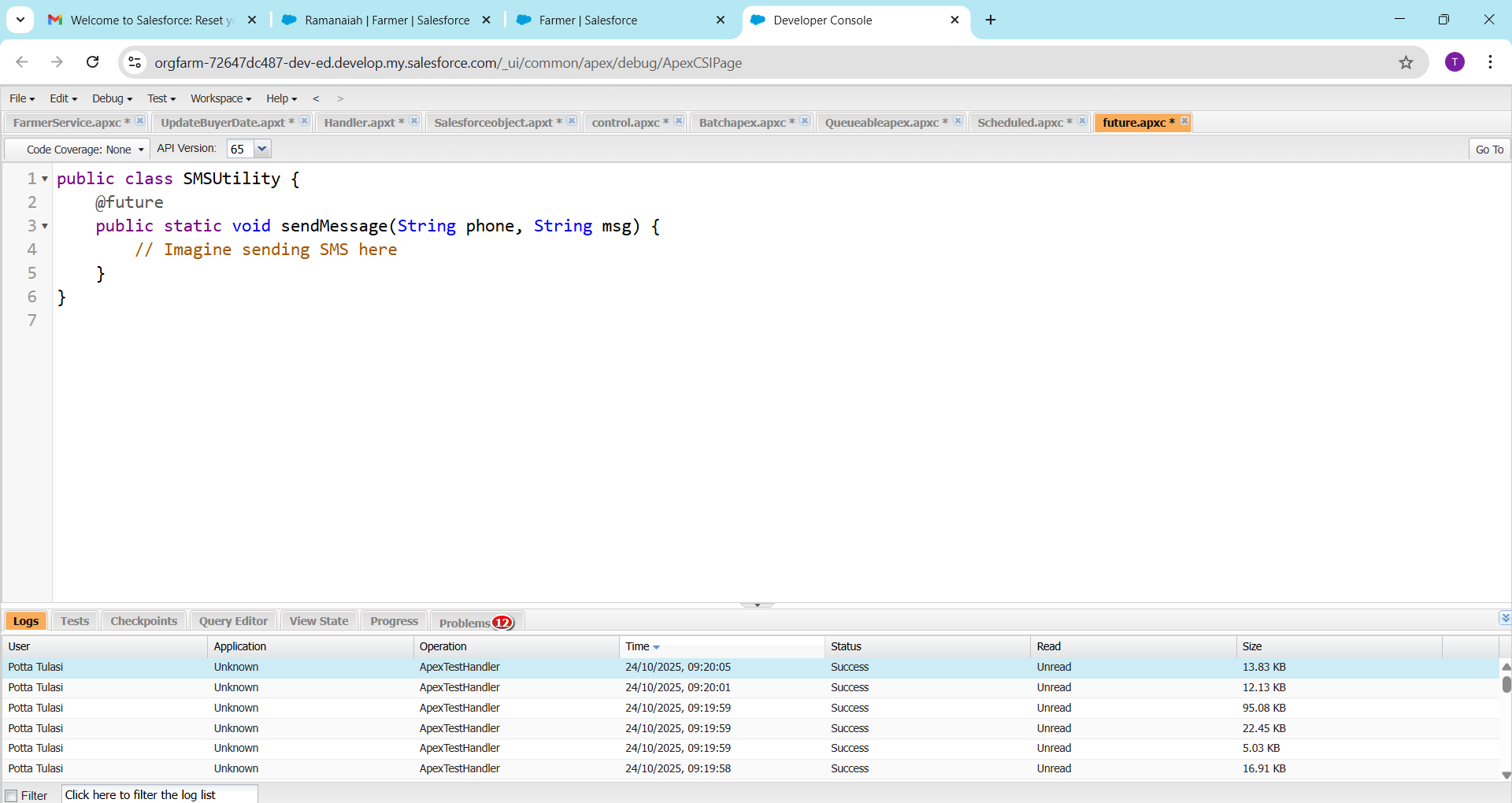
public static void sendMessage(String phone, String msg) {

// Imagine sending SMS here

}

}

🟢 Example: After an order is placed → send a thank-you SMS.



**🧱 11. Exception Handling**

Used to stop code from crashing when there’s an error.

try {

update farmers;

} catch(DmlException e) {

System.debug('Error: ' + e.getMessage());

}

🟢 If update fails, Salesforce shows a friendly message instead of crashing.

**🧪 12. Test Classes**

Before deploying Apex code, Salesforce *requires* 75% test coverage.

**Example Test:**

@isTest

public class TestOrderTrigger {

@isTest static void testOrderCreation() {

Buyer\_\_c b = new Buyer\_\_c(Name='Test Buyer');

insert b;

Order\_\_c o = new Order\_\_c(Name='Test Order', Buyer\_\_c=b.Id, Amount\_\_c=1000);

insert o;

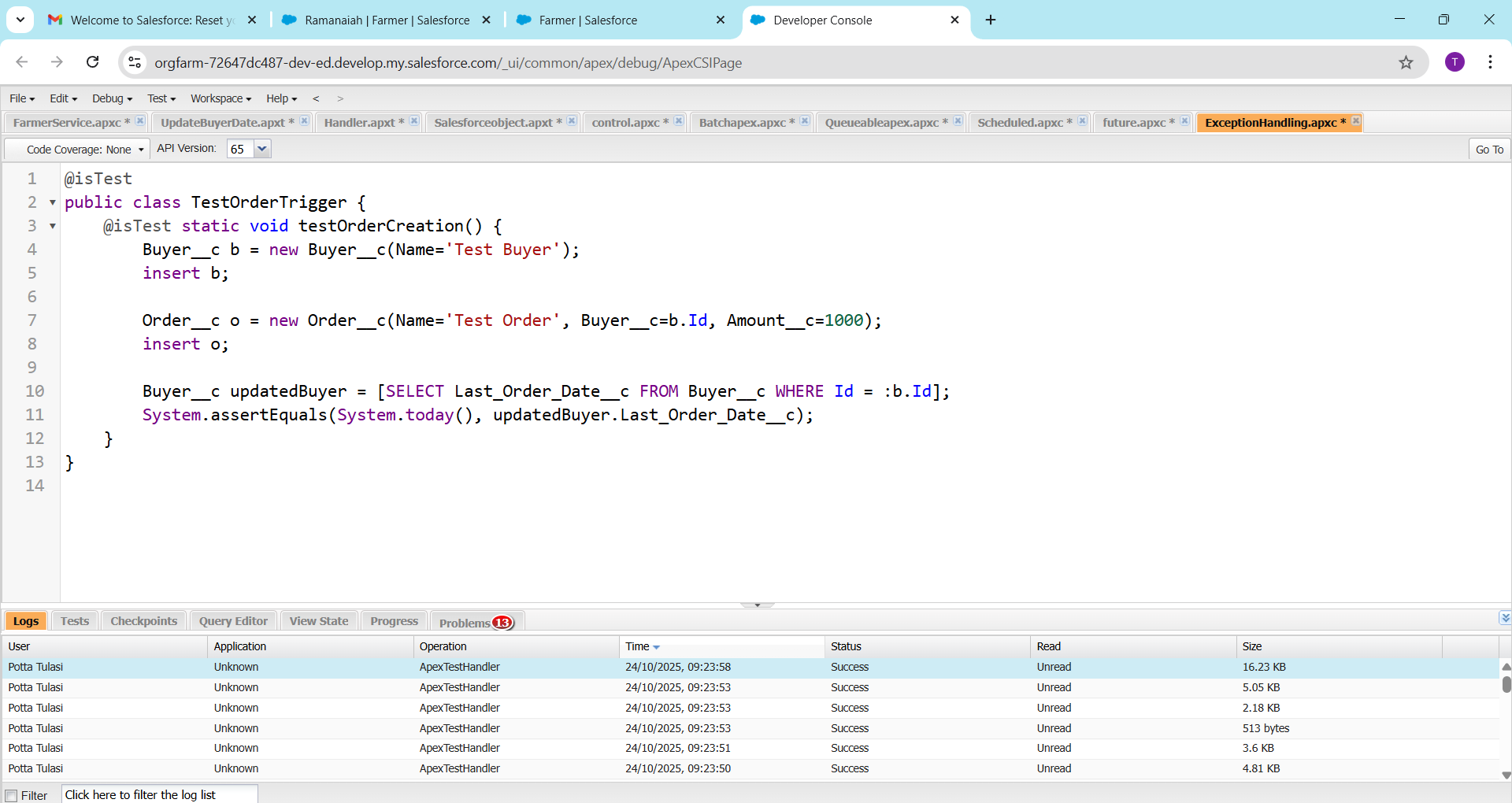
Buyer\_\_c updatedBuyer = [SELECT Last\_Order\_Date\_\_c FROM Buyer\_\_c WHERE Id = :b.Id];

System.assertEquals(System.today(), updatedBuyer.Last\_Order\_Date\_\_c);

}

}

🟢 Tests check that your trigger updates Buyer’s “Last Order Date” correctly.



**⚡ 13. Asynchronous Processing Types**

| **Type** | **Runs In Background** | **Use Case** |
| --- | --- | --- |
| **Future** | ✅ Yes | Simple tasks like SMS |
| **Batch** | ✅ Yes | Large data processing |
| **Queueable** | ✅ Yes | Medium jobs, can chain |
| **Scheduled** | ✅ Yes | Time-based execution |

🟢 Example:  
Batch updates Farmer info → Queueable sends notification → Scheduled runs monthly.

**🔁 14. Full Automation Flow in Project**

1️⃣ User creates Order →

2️⃣ Trigger runs →

3️⃣ Handler updates Buyer →

4️⃣ Queueable Apex sends confirmation →

5️⃣ Batch Apex updates all related Farmers monthly →

6️⃣ Test Class validates logic before deployment

**✅ 15. Outcome**

After Phase 5:

* Your Salesforce backend becomes *intelligent and automated*.
* Farmers, Buyers, and Government officials don’t need to do manual updates.
* All actions (orders, approvals, updates) happen automatically through Apex code.