Phase 5: Apex Programming (Developer)

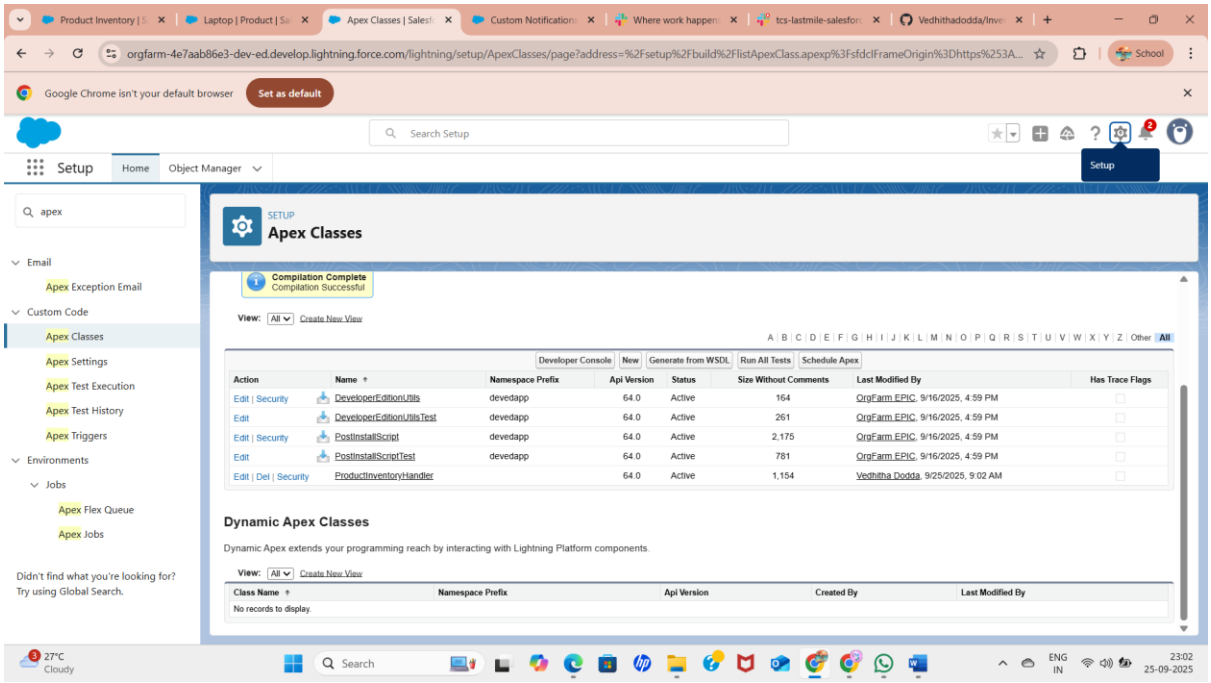
1. Classes & Objects

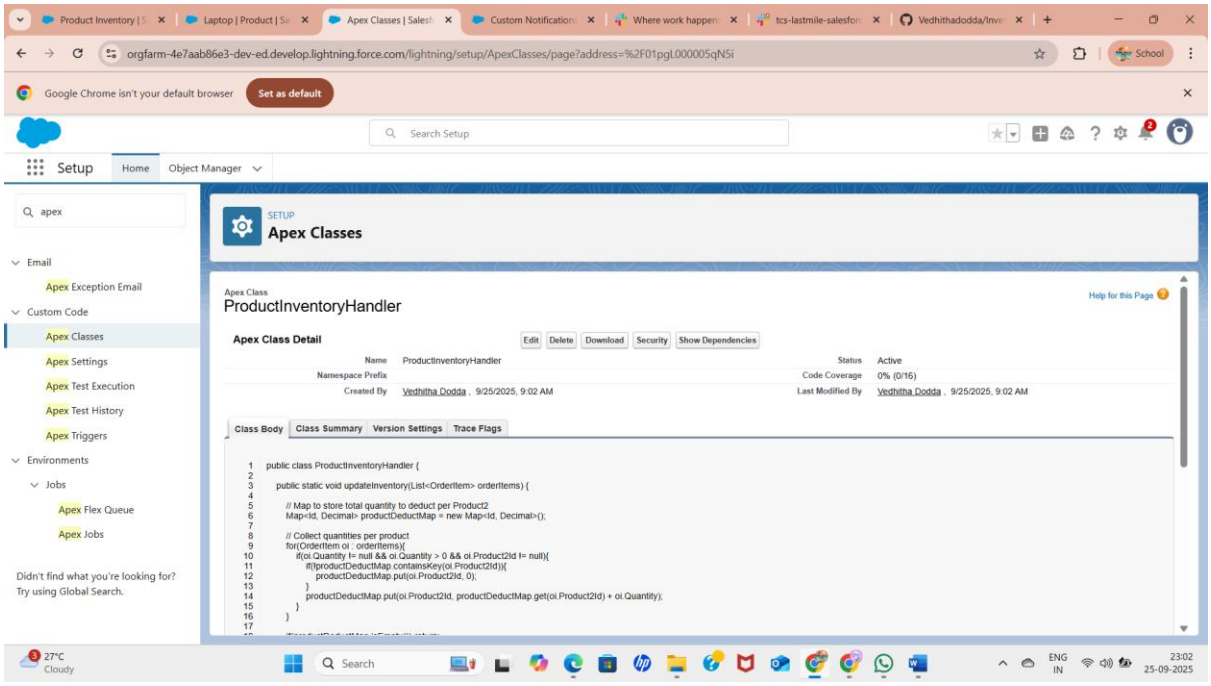
- **Definition:** Apex is an object-oriented language; classes define the blueprint for objects.
- **Example:**

```
public class ProductManager {
    public String productName;
    public Integer quantity;

    public ProductManager(String name, Integer qty){
        this.productName = name;
        this.quantity = qty;
    }

    public void displayProduct(){
        System.debug('Product: ' + productName + ', Quantity: ' + quantity);
    }
}
```

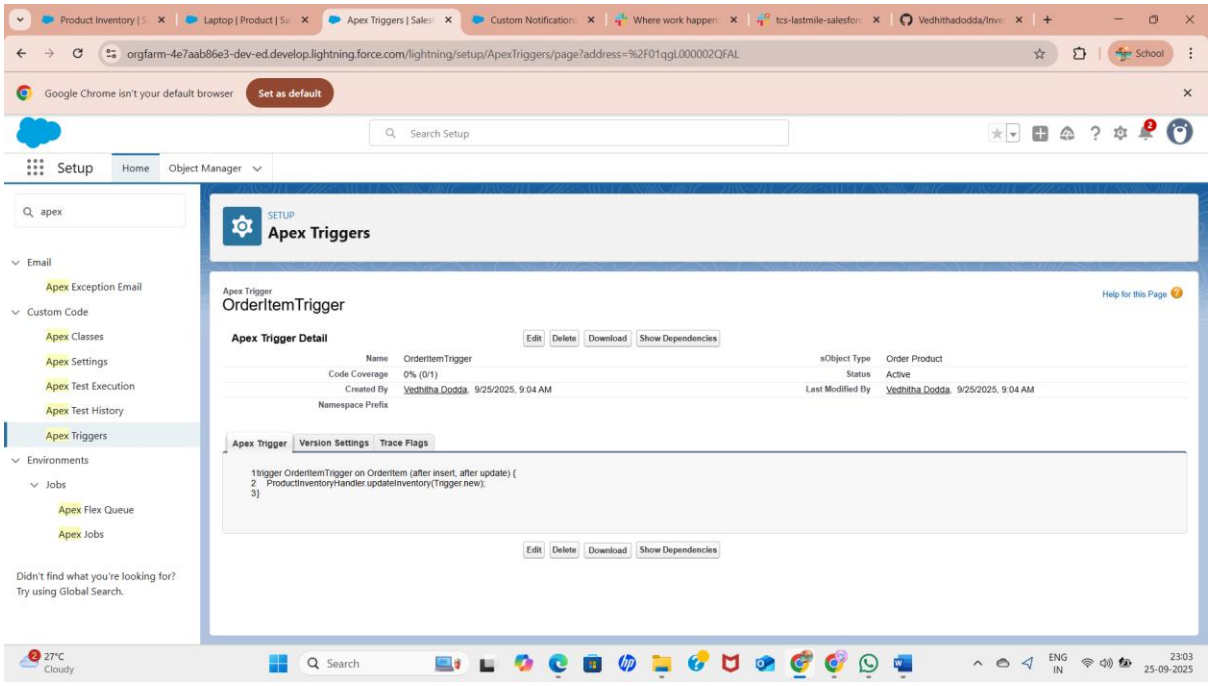




2. Apex Triggers (Before/After Insert/Update/Delete)

- **Definition:** Automatically execute Apex code before or after DML operations.
- **Trigger Events:** before insert, after insert, before update, after update, before delete, after delete, after undelete.
- **Example:**

```
trigger UpdateInventory on OrderItem (after insert) {
    for(OrderItem oi : Trigger.new){
        Product_Inventory__c pi = [SELECT Id, Quantity__c FROM Product_Inventory__c WHERE Product_Lookup__c = :oi.Product2Id LIMIT 1];
        pi.Quantity__c -= oi.Quantity;
        update pi;
    }
}
```



3. Trigger Design Pattern

- **Purpose:** Avoid logic duplication and maintain bulk-safe, maintainable triggers.
- **Pattern:**
    1. **Trigger** → calls **Handler Class**
    2. **Handler Class** → contains all business logic
- **Example Structure:**

Trigger: AccountTrigger → AccountTriggerHandler class

4. SOQL & SOSL

- **SOQL (Salesforce Object Query Language):** Query Salesforce records.

List<Account> accounts = [SELECT Id, Name FROM Account WHERE Industry='Technology'];

- **SOSL (Salesforce Object Search Language):** Search across multiple objects/fields.

List<List<SObject>> searchResults = [FIND 'Laptop*' IN ALL FIELDS RETURNING Product2(Name, Price__c)];

5. Collections: List, Set, Map

- **List:** Ordered collection, allows duplicates.

List<String> fruits = new List<String>{'Apple','Banana'};

- **Set:** Unordered collection, no duplicates.

```
Set<String> colors = new Set<String>{'Red','Blue'};
```

- **Map:** Key-value pairs.

```
Map<Id, Account> accMap = new Map<Id, Account>([SELECT Id, Name FROM Account]);
```

6. Control Statements

- **If-Else**
- **For Loops / While Loops**
- **Switch Statements**

```
for(Account a : accounts){
   if(a.Industry == 'Tech'){
      System.debug(a.Name);
   }
}
```

7. Batch Apex

- **Definition:** Process large data volumes asynchronously in batches.
- **Example:**

```
global class BatchUpdateInventory implements Database.Batchable<sObject>{
   global Database.QueryLocator start(Database.BatchableContext BC){
      return Database.getQueryLocator('SELECT Id, Quantity__c FROM
Product_Inventory__c');
   }
   global void execute(Database.BatchableContext BC, List<Product_Inventory__c> scope){
      for(Product_Inventory__c pi : scope){
         pi.Quantity__c += 10;
      }
      update scope;
   }
   global void finish(Database.BatchableContext BC){}
}
```

- **Use Cases:** Large data updates, mass emailing, recalculations.

8. Queueable Apex

- **Definition:** Asynchronous Apex for jobs requiring chaining or complex processing.
- **Example:**

```
public class InventoryUpdateQueueable implements Queueable{
   public void execute(QueueableContext ctx){
      // logic here
   }
}
System.enqueueJob(new InventoryUpdateQueueable());
```

9. Scheduled Apex

- **Definition:** Run Apex classes at specified times.

```
global class DailyInventoryUpdate implements Schedulable{
   global void execute(SchedulableContext ctx){
      // batch or update logic
   }
}
```

- **Schedule Example:**

```
System.schedule('Daily Inventory', '0 0 6 * * ?', new DailyInventoryUpdate());
```

10. Future Methods

- **Definition:** Run asynchronous code for callouts or heavy processing.

```
@future
public static void updateInventoryAsync(List<Id> productIds){}
```

11. Exception Handling

- **Try-Catch-Finally** blocks for safe error handling.

```
try{
   update products;
```

```
} catch(DmlException e){
   System.debug('Error: '+ e.getMessage());
}
```

12. Test Classes

- **Purpose:** Ensure code works correctly and deployable to production.
- **Requirements:** 75% code coverage.

```
@isTest
public class TestInventoryUpdate{
   static testMethod void testInventory(){
      // create test data
      // call trigger or class
      // assert results
   }
}
```

13. Asynchronous Processing

- Includes **Batch Apex, Queueable, Scheduled, Future Methods**
- Useful for handling **large volumes of data or long-running processes** without hitting governor limits.