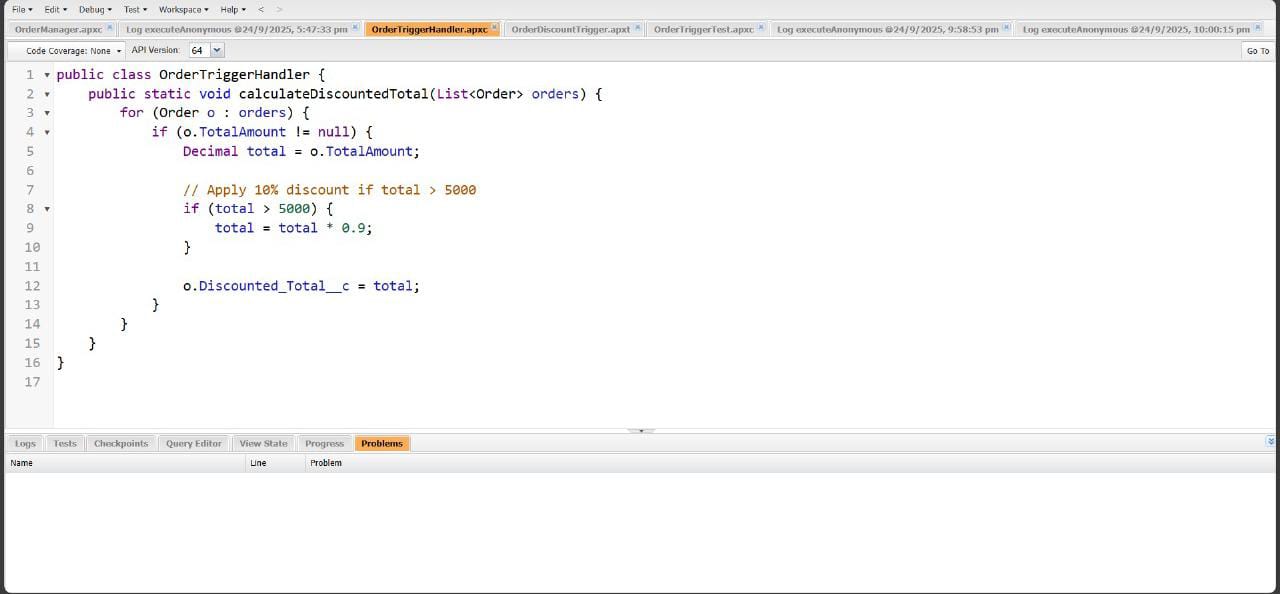
**Phase 5: Apex Programming (Developer)**

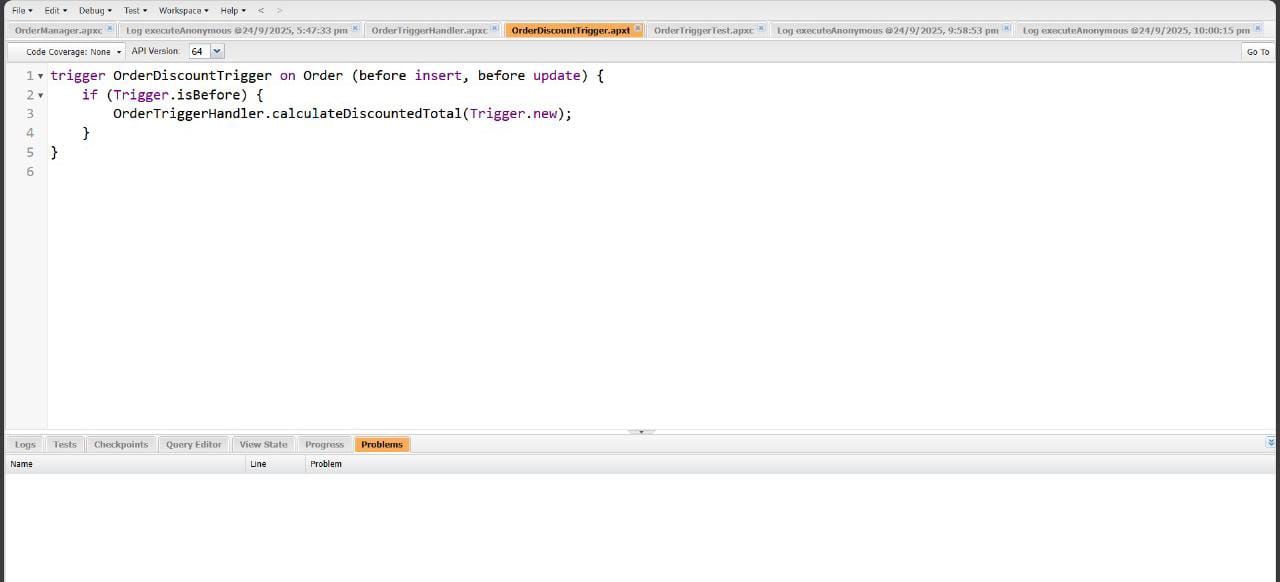
In your e-commerce app, when a customer places an order with products, Salesforce automatically calculates the **TotalAmount** on the Order record (based on OrderItems).

But you want an **extra custom field** (Discounted\_Total\_\_c) that applies a **10% discount if the total is greater than 5000**.

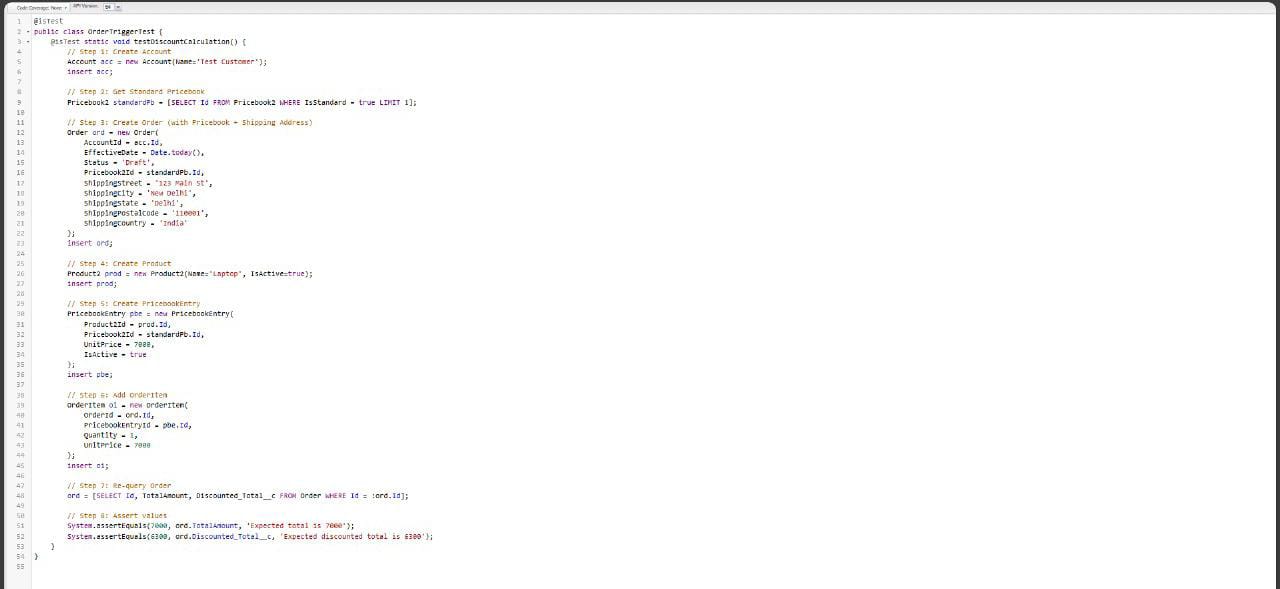
* **Handler Class**



* **Trigger Handler Pattern:** Used a separate handler class OrderTriggerHandler to maintain clean code and follow best practices (Separation of Concerns).
* **Trigger**



* **Apex Trigger:** Implemented OrderDiscountTrigger on the **Order** object to auto-calculate discounted totals whenever an order is inserted or updated.
* **Test Class**



* Created a **test class** to insert sample Accounts, Orders, Products, PricebookEntries, and OrderItems.
* Verified that **OrderDiscountTrigger** calculates and updates **Discounted\_Total\_\_c** correctly.
* **How to Run / Verify**

**Quick Test in Execute Anonymous**

// Step 1: Create Account

Account acc = new Account(Name='Quick Test Customer');

insert acc;

// Step 2: Get Standard Pricebook

Pricebook2 standardPb = [SELECT Id FROM Pricebook2 WHERE IsStandard = true LIMIT 1];

// Step 3: Create Order with Pricebook + Shipping Address

Order ord = new Order(

AccountId = acc.Id,

EffectiveDate = Date.today(),

Status = 'Draft',

Pricebook2Id = standardPb.Id,

ShippingStreet = '123 Main St',

ShippingCity = 'New Delhi',

ShippingState = 'Delhi',

ShippingPostalCode = '110001',

ShippingCountry = 'India'

);

insert ord;

// Step 4: Create Product

Product2 prod = new Product2(Name='Mobile', IsActive=true);

insert prod;

// Step 5: Create PricebookEntry

PricebookEntry pbe = new PricebookEntry(

Product2Id = prod.Id,

Pricebook2Id = standardPb.Id,

UnitPrice = 6000,

IsActive = true

);

insert pbe;

// Step 6: Add OrderItem

OrderItem oi = new OrderItem(

OrderId = ord.Id,

PricebookEntryId = pbe.Id,

Quantity = 1,

UnitPrice = 6000

);

insert oi;

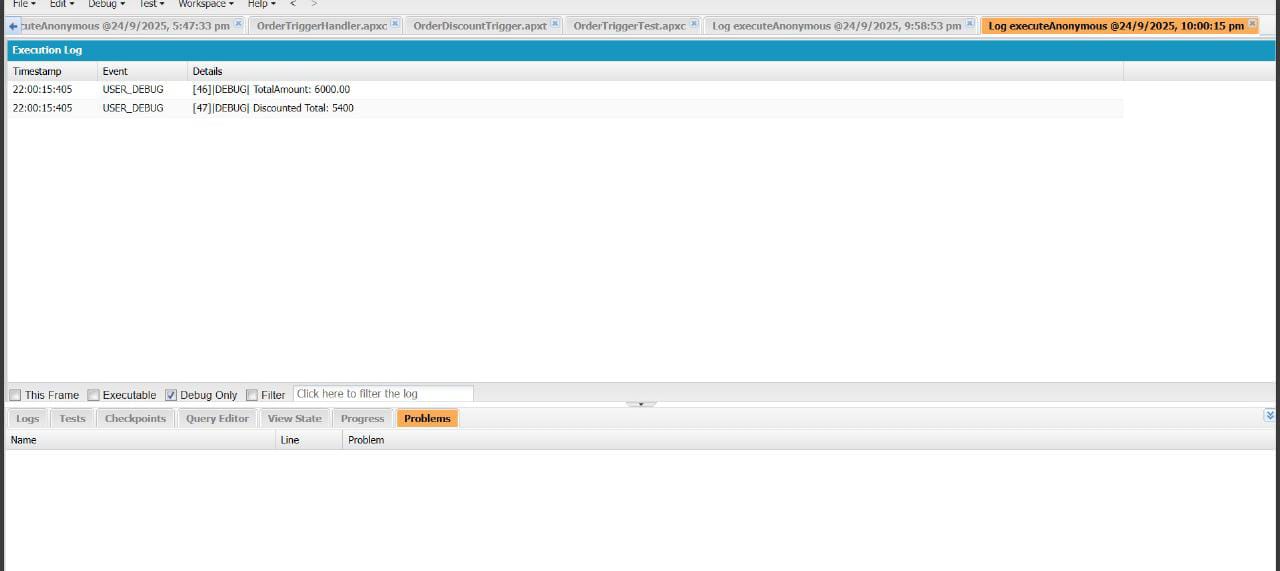
// Step 7: Re-query Order

ord = [SELECT Id, TotalAmount, Discounted\_Total\_\_c FROM Order WHERE Id = :ord.Id];

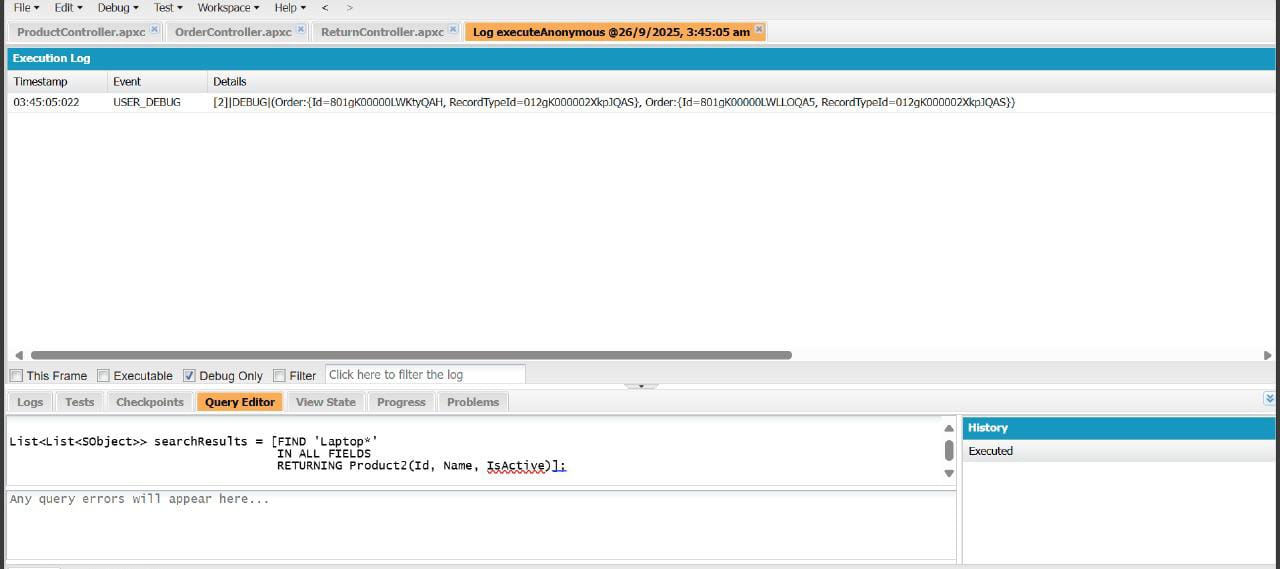
System.debug(' TotalAmount: ' + ord.TotalAmount);

System.debug(' Discounted Total: ' + ord.Discounted\_Total\_\_c);

* **Testing in Anonymous Window:** Verified functionality by inserting sample **Account, Order, Product, PricebookEntry, and OrderItem** records and checking debug logs.
* **Output in Logs will show:**

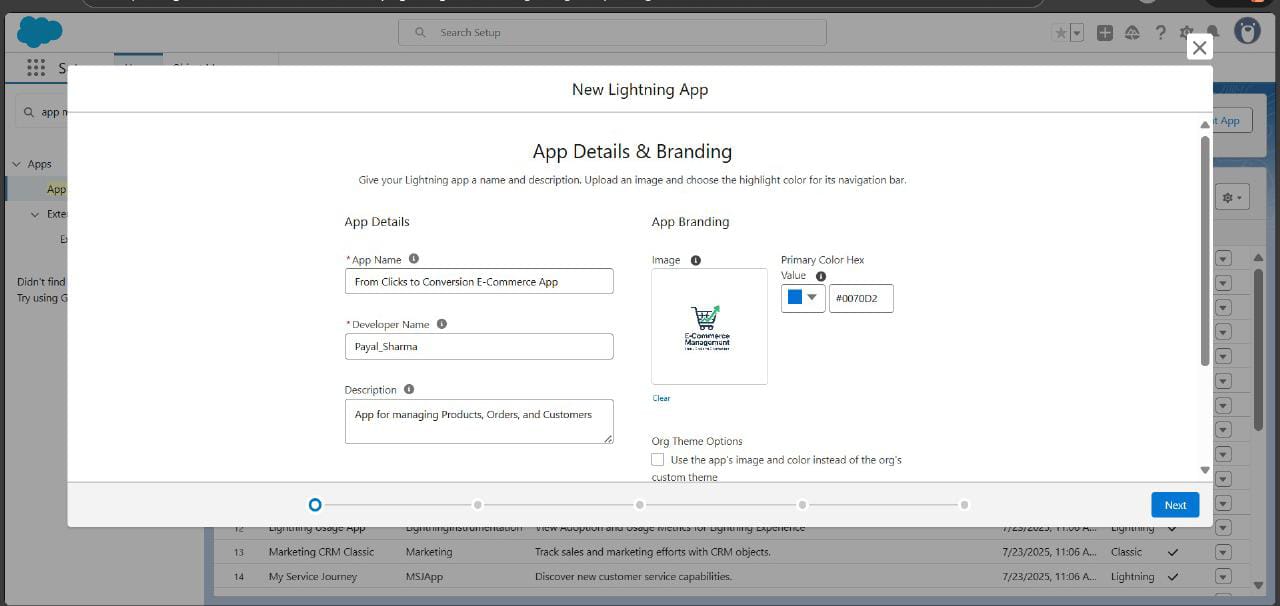
****

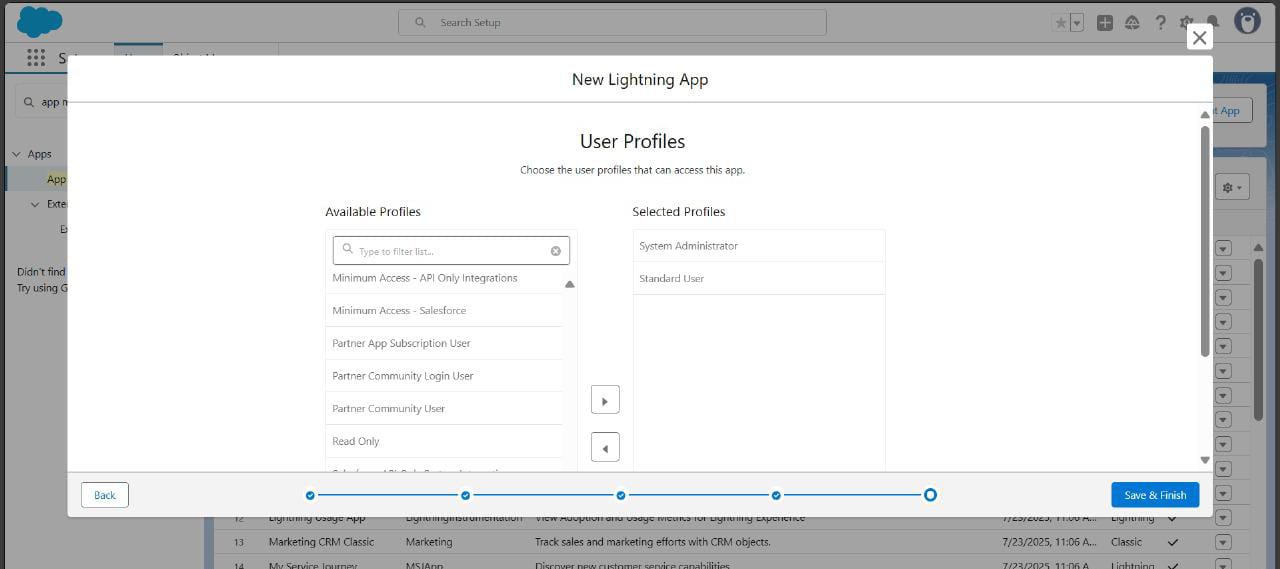
* **Results**
* Successfully automated discount calculation on Orders.
* Reduced manual effort and ensured consistency in pricing logic.
* Built a reusable, scalable solution aligned with Salesforce development best practices.
* **SOQL & SOSL**

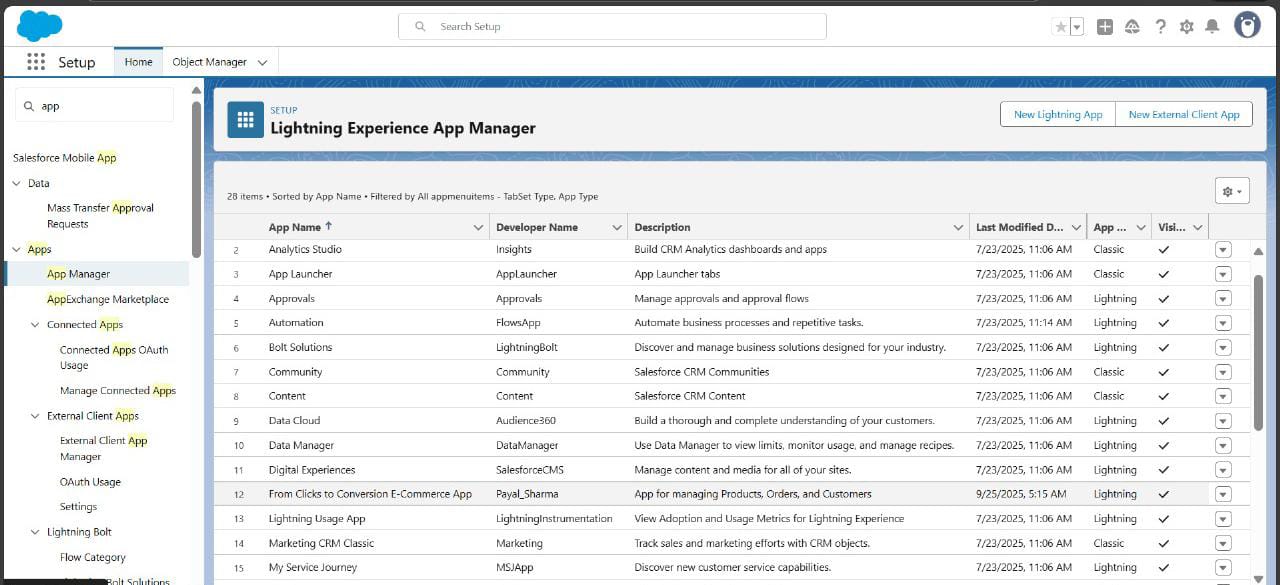
****

**Phase 6: User Interface Development**

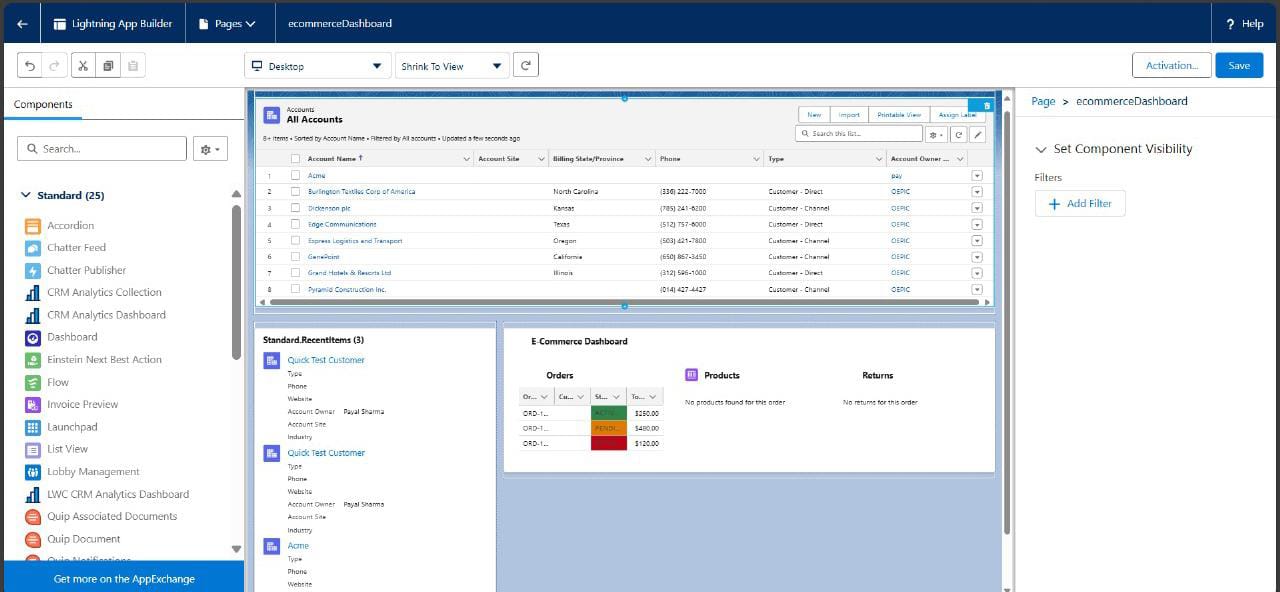
* **Lightning App Builder**







# Phase **LWC with Apex Integration:**



# **E-CommerceDashboard in Salesforce Lightning**:

# Built using Lightning App Builder with standard drag-and-drop components

# Displays “All Accounts” with details like account name, site, phone, and customer type

# Recent Items panel shows quick access to recently viewed or modified records

# Orders section lists order numbers, status with color codes, and total amounts

# Products and Returns sections show relevant items or indicate none found

# 

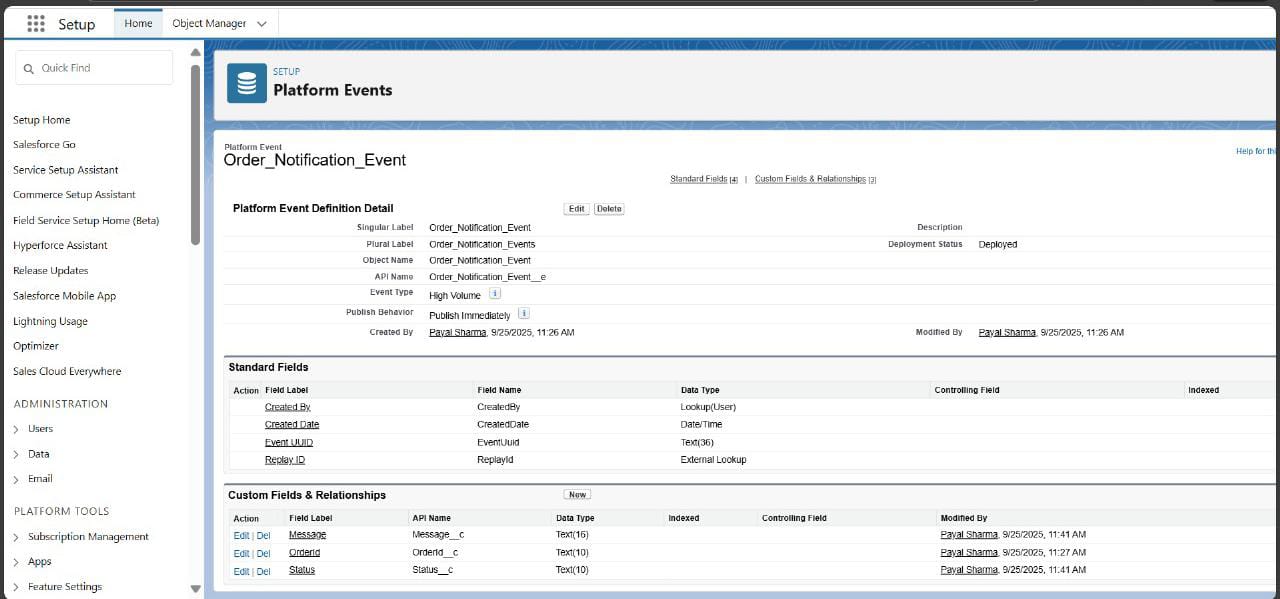
# 

* **Apex classes** handle server-side logic and fetch Salesforce data (Orders, Products, Returns).
* **Parent-child hierarchy:** ecommerceDashboard (parent) communicates with child LWCs via **@api properties** and **events**.
* Child LWCs call **Apex methods** using **@wire adapters** or **imperative Apex calls** for dynamic data loading.
* Ensures **modular, reusable components** with **real-time data interaction**.
* Provides a **scalable architecture** for extending features in the future.

# 7: Integration & External Access

* Platform Event Setup

| **Field Name** | **Data Type** | **Notes** |
| --- | --- | --- |
| OrderId\_\_c | Text | Store the Salesforce Order ID |
| Status\_\_c | Text | Store "Activated", "Shipped", etc. |
| Message\_\_c | Text | Optional message for notifications |
|  |  |  |

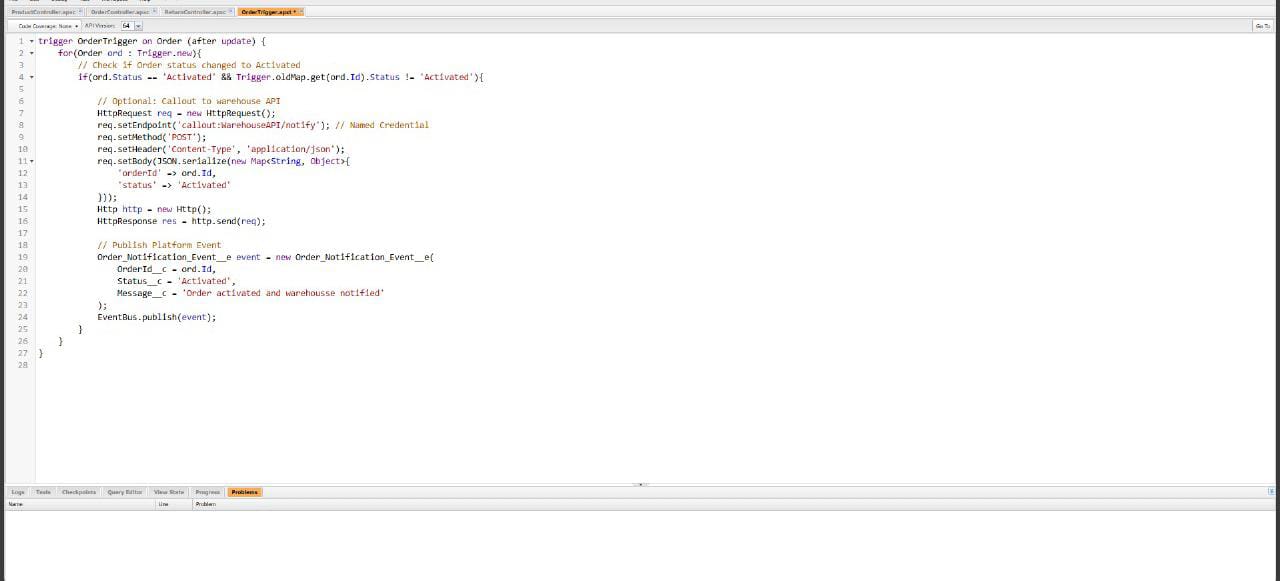


* Remote Site / Named Credential

If you plan a callout to an external system (e.g., warehouse API):

* **Remote Site:** <https://api.warehouse.com>

# **Apex Trigger on Order**



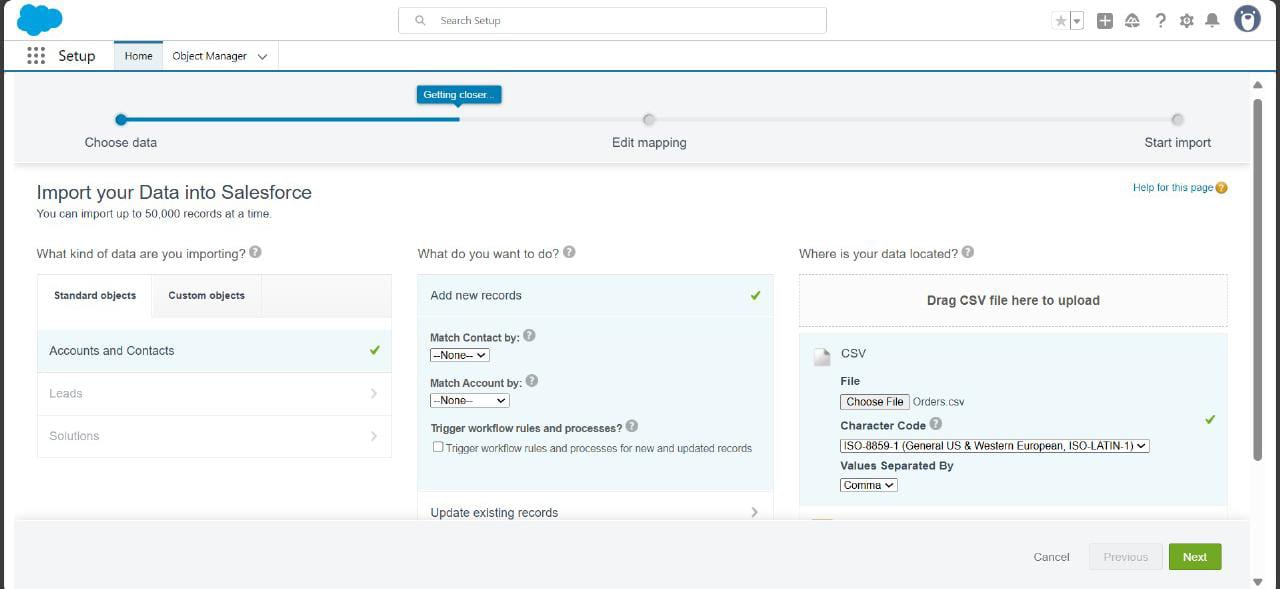
* **. Key Benefits / Takeaways**
* Demonstrates **real-time communication** using Platform Events.
* Shows **integration with external APIs** through callouts.
* Uses **standard Salesforce objects**, keeping the design simple and maintainable.
* Provides a **scalable foundation** for adding more complex workflows like payment processing or shipping updates.

**Phase 8: Data Management & Deployment**

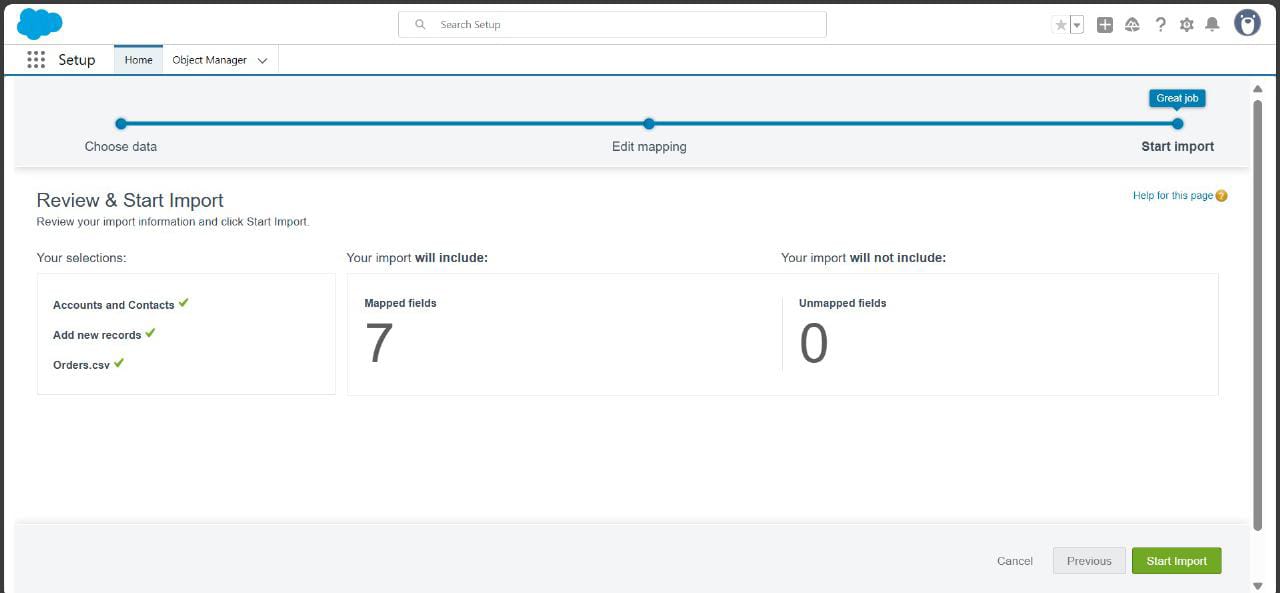
* Data Import Wizard (Import Orders & Customers)

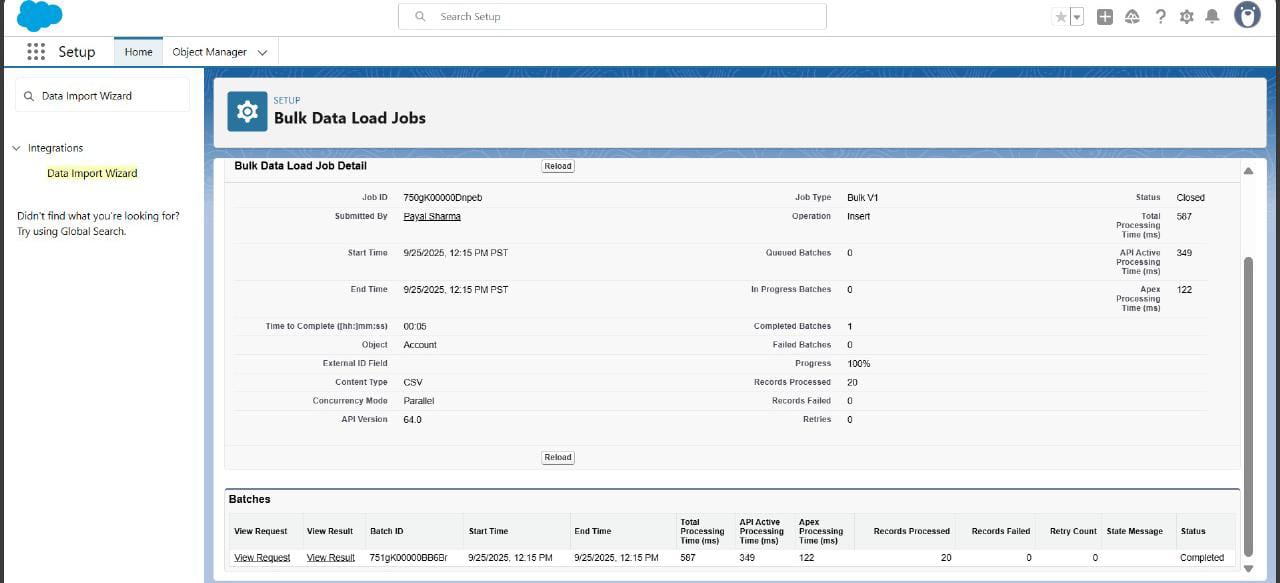
We used the **Data Import Wizard** to easily import Orders, Customers, and Products via a simple UI, ensuring smooth onboarding of records.

**1.Uploading data**

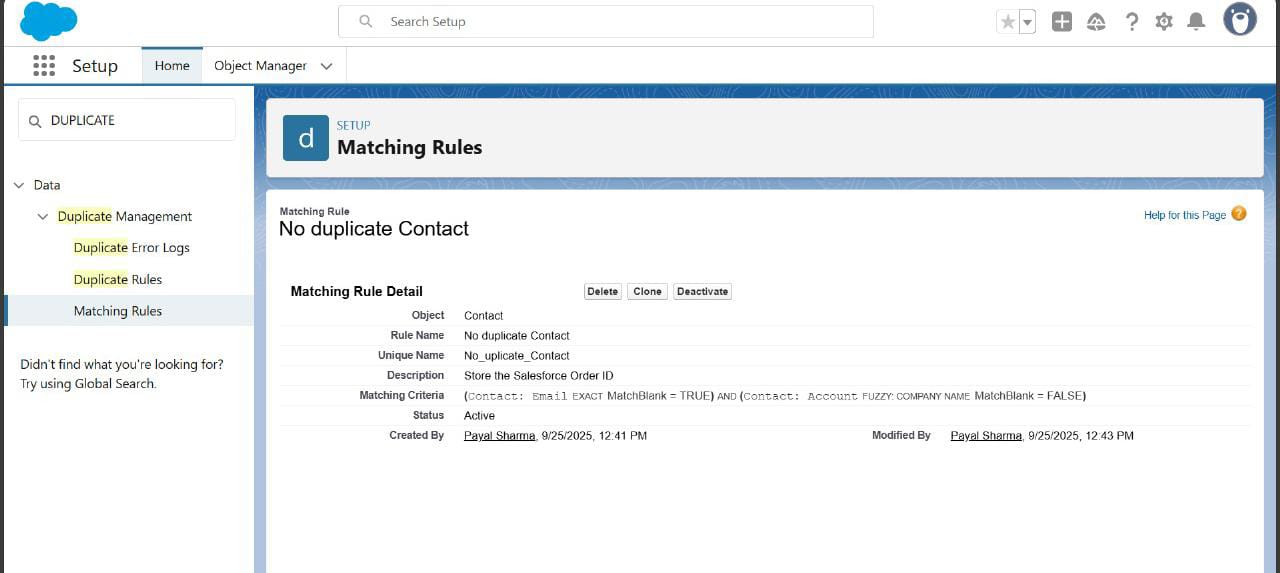


**2. Mapping objects**



**3.RESULTS** 

* Duplicate Rules (Prevent Duplicate Customers)

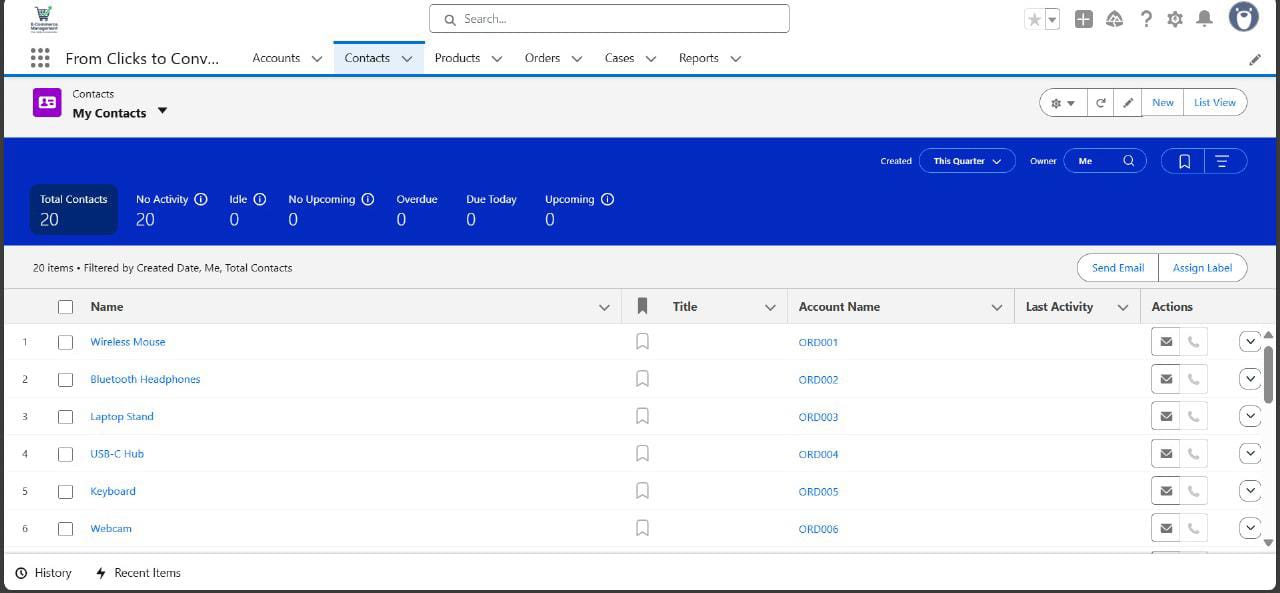


Actions:

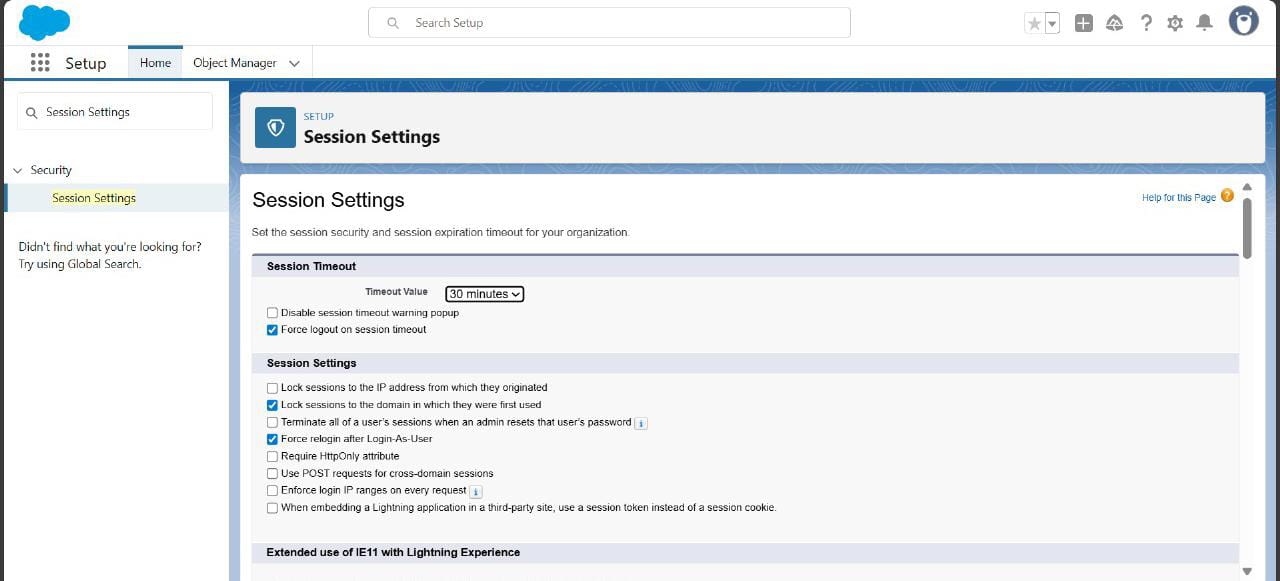
* + On Create → **Block** (stop duplicate customer creation).
  + On Edit → **Allow but Alert** (warn users if a customer’s email is changed to an existing one).

Phase 9: Reporting, Dashboards & Security Review

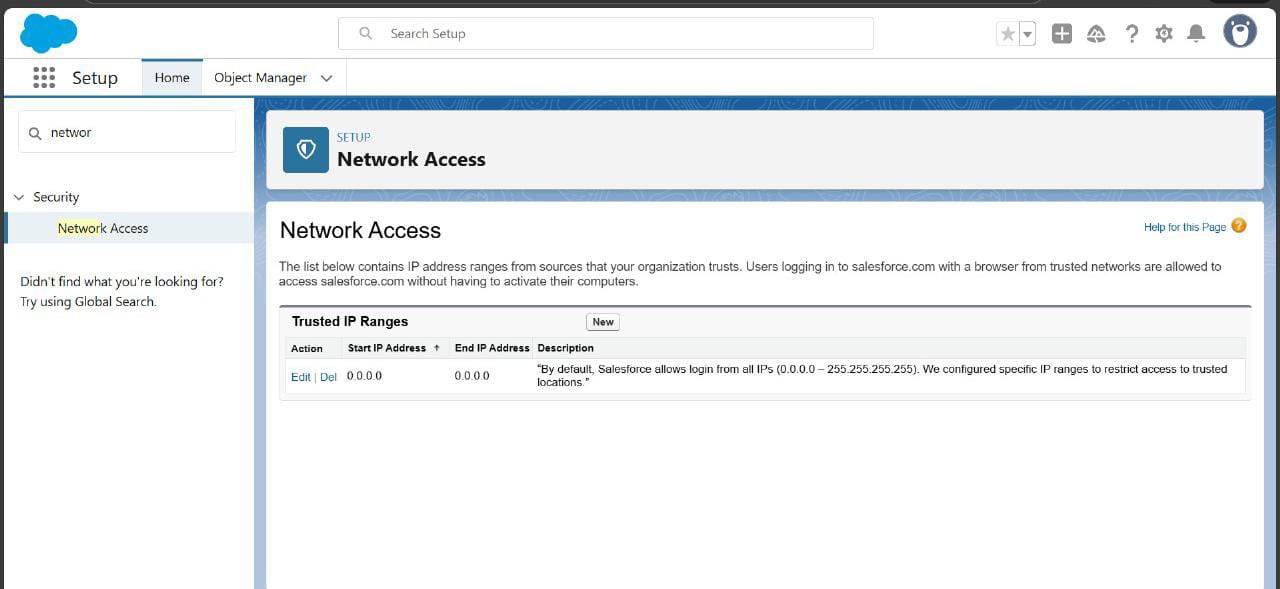
* Reports



* The "My Contacts" page provides an organized overview of all product orders and their associated accounts, showing a total of 20 contacts for this quarter
* Products listed include items such as Wireless Mouse, Bluetooth Headphones, Laptop Stand, etc., each linked to a unique account name (ORD001, ORD002, etc.) for straightforward tracking.
* **Session Settings**



* **Implementation:** Configure session timeout (e.g., 30 min), force logout on browser close, lock sessions to originating IP, and enable secure browser caching.
* **Use Case:** Protects sensitive data by automatically ending inactive sessions and preventing unauthorized access.
* Login IP Ranges



* **Implementation:** By default, Salesforce allows login from all IPs. For the project, we restricted access to **trusted office IP ranges** and configured specific profiles (e.g., Sales, Support) to access only allowed IPs.
* **Use Case:** Ensures that only authorized users from secure locations can access **Orders and Customer data**, enhancing data security.

“Thank you for your guidance and support Sir!”