Order Management System using Salesforce

1. Problem Statement

In many organizations, the process of creating, approving, and managing client orders is manual and error-prone.

- Sales teams face difficulty in handling orders and checking product stock.
- Business managers struggle to monitor approvals and track order progress.
- Support teams lack visibility into order status and client communication.

This results in delayed approvals, stock mismatches, and reduced customer satisfaction.

A Salesforce-based Order Management CRM can streamline the order-to-invoice cycle by automating approvals, updating product inventory in real-time, and providing dashboards for business insights.

Phase 1: Problem Understanding & Industry Analysis

Goal: Clearly understand the Order Management business process and why we're building this CRM.

1. Requirement Gathering

- Talk to Sales teams, Business managers, Support staff.
- Example requirements:
  - Sales reps should create new orders and search/add products.
  - Orders below $100,000 → Auto-approved.
  - Orders above $100,000 → Sent for manager approval.
  - Stock levels must auto-update when order items are added/removed.
  - Support team should track order issues and invoice status.
  - Managers need dashboards for sales, pending approvals, and invoices.

2. Stakeholder Analysis

- **Sales User** → Creates orders, adds/removes products.
- **Business User (Manager)** → Approves/rejects high-value orders.
- **Support User** → Resolves order-related exceptions, monitors status.
- **Admin (you)** → Configures org, builds LWCs, triggers, reports.

3. Business Process Mapping

Order Lifecycle Flow:
Customer places order → Sales user creates order →
If amount < $100k → Auto-approved → Invoice generated → Stage updated.
If amount ≥ $100k → Manager approval required → Approved/Rejected → Invoice generated → Stage updated.

4. Industry-Specific Use Case Analysis

- **Retail/Wholesale industry**: Orders must sync with stock availability.
- **B2B sales**: Large deals often require manager approval.
- **Support teams**: Need real-time visibility into order progress for handling escalations.

5. AppExchange Exploration

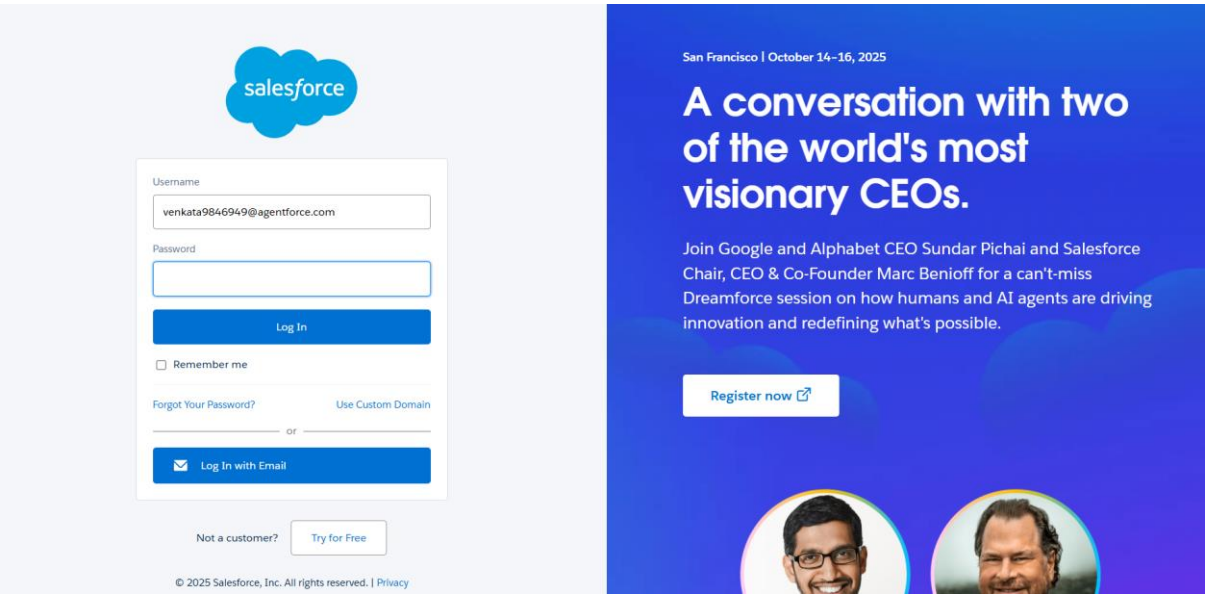
- Check if existing Order Management / CPQ apps exist.
- Decide to build a custom solution (simpler, educational, tailored to your scenario).

Phase 2: Org Setup & Configuration

Goal: Prepare the Salesforce environment for your Order Management CRM.

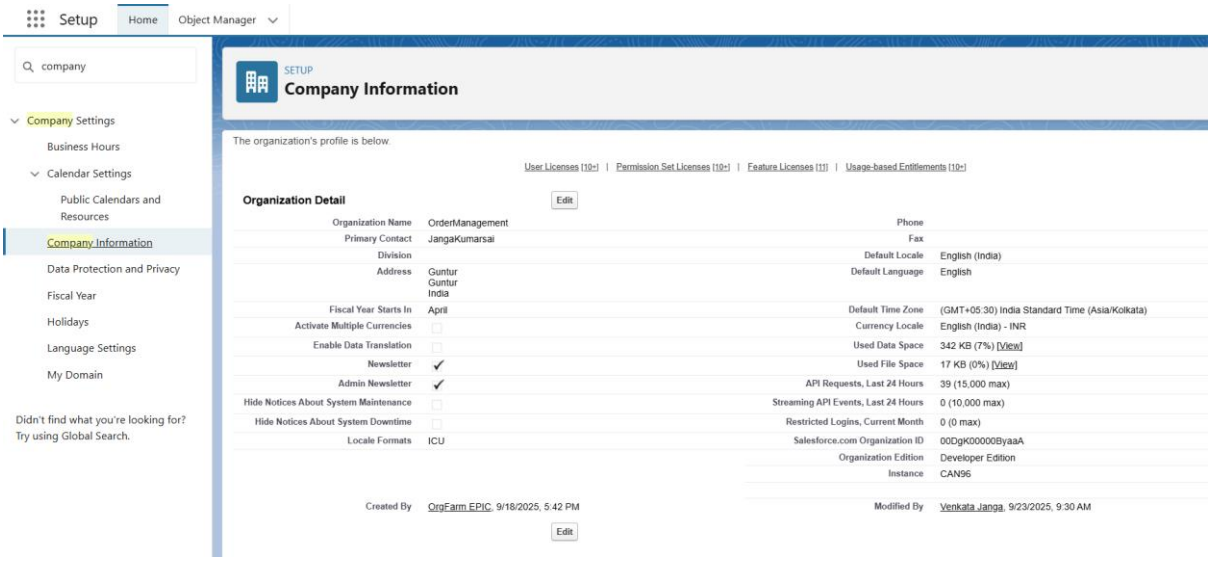
1. Salesforce Edition

- Use Developer Org (free) for building & testing.



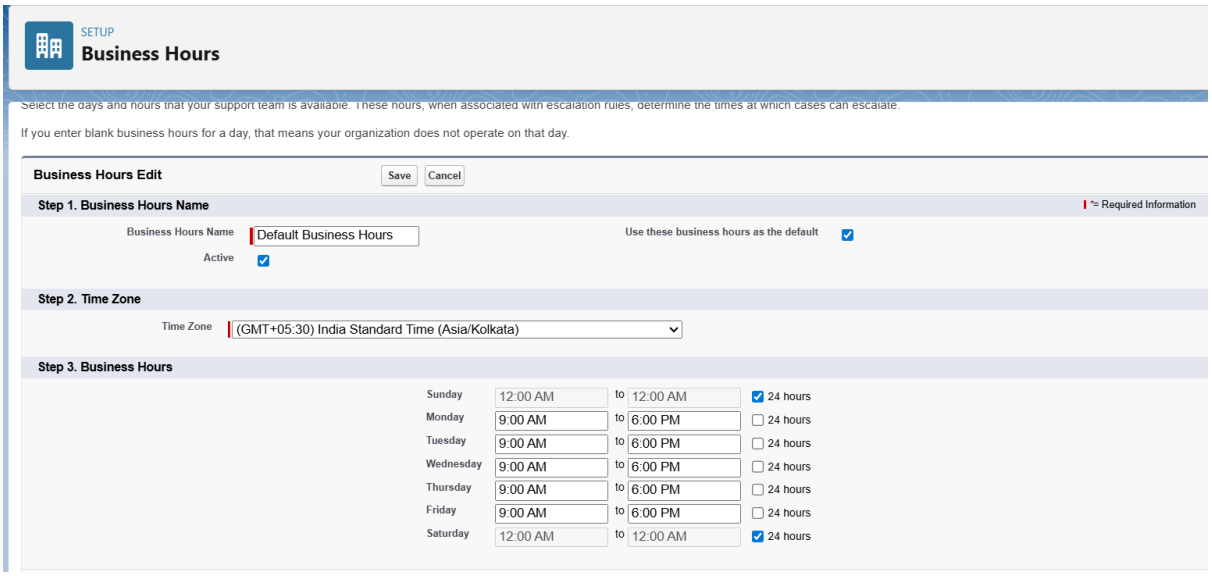
2. Company Profile Setup

- Configure company info, time zone, currency (USD/INR).

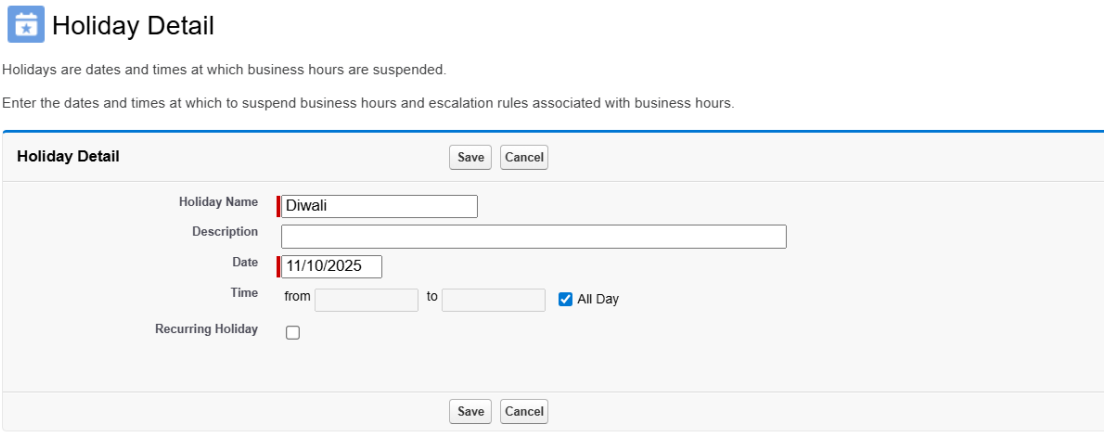


3. Business Hours & Holidays

- Define working hours (e.g., 9 AM – 6 PM).

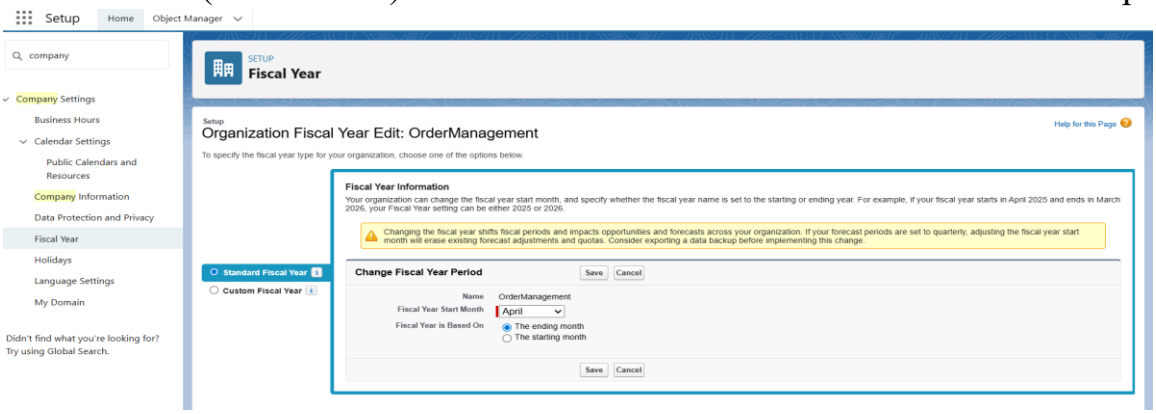


- Add holidays (no approvals during these days).



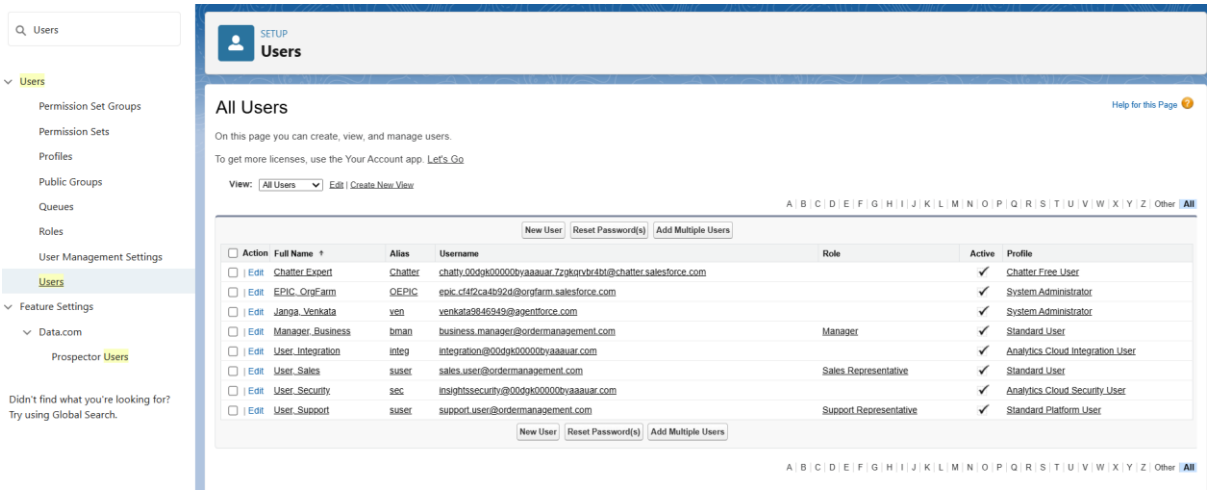
4. Fiscal Year Settings

- Standard (APR–MAR) → for order & revenue reporting



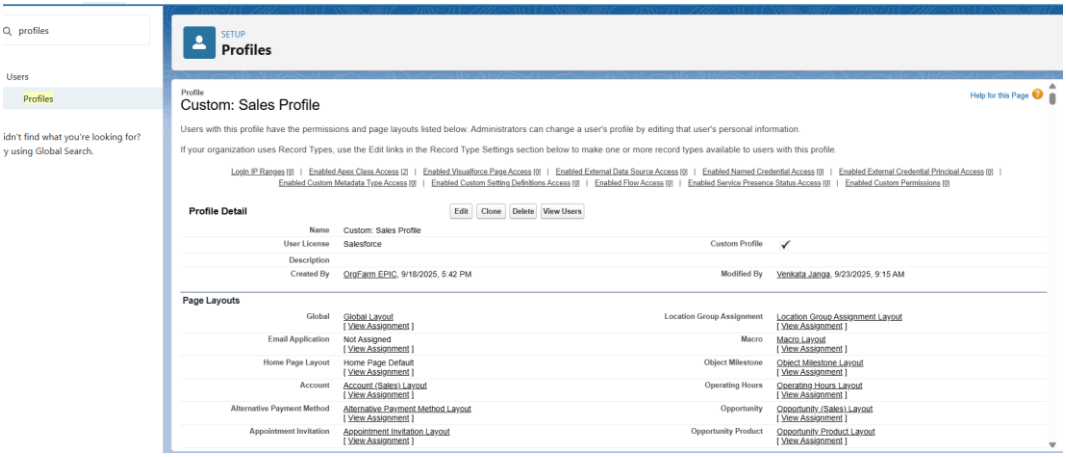
5. User Setup & Licenses

- Create sample users: Sales User, Business Manager, Support User.
- Assign Salesforce licenses.

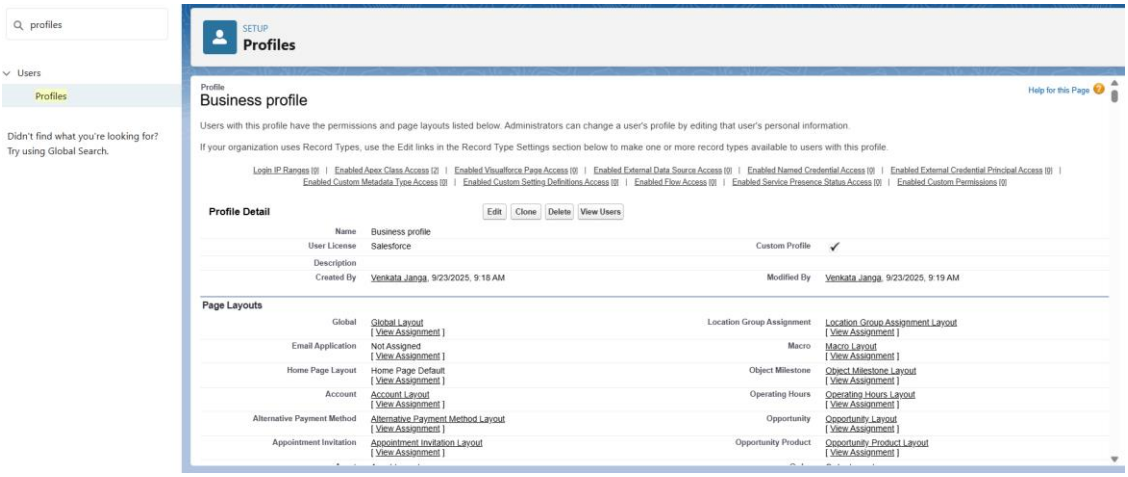


6. Profiles

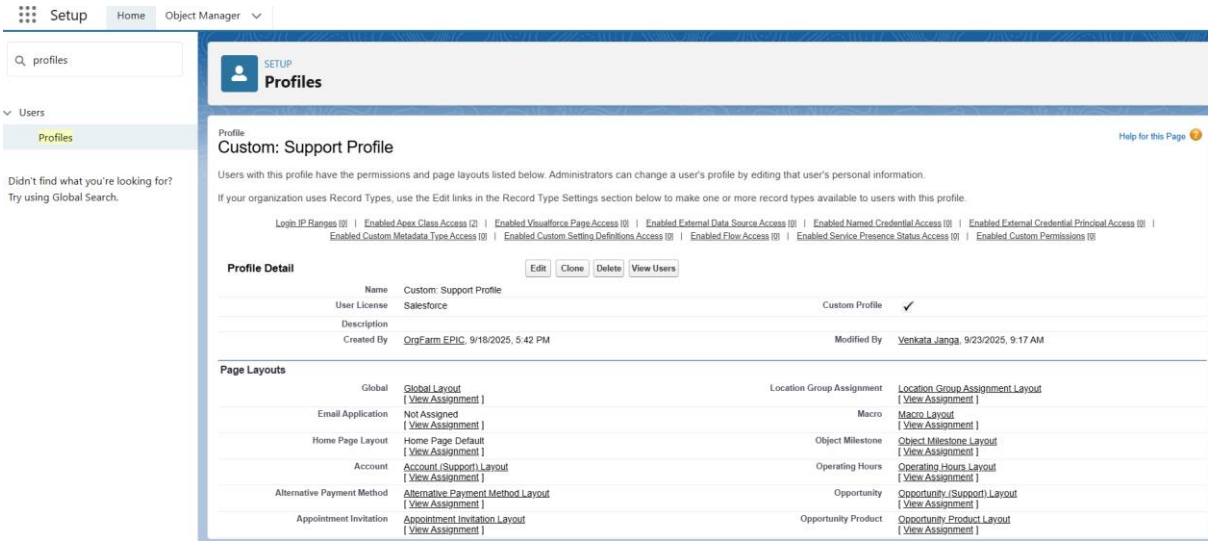
- **Sales Profile** → Create/Edit Orders, search products.



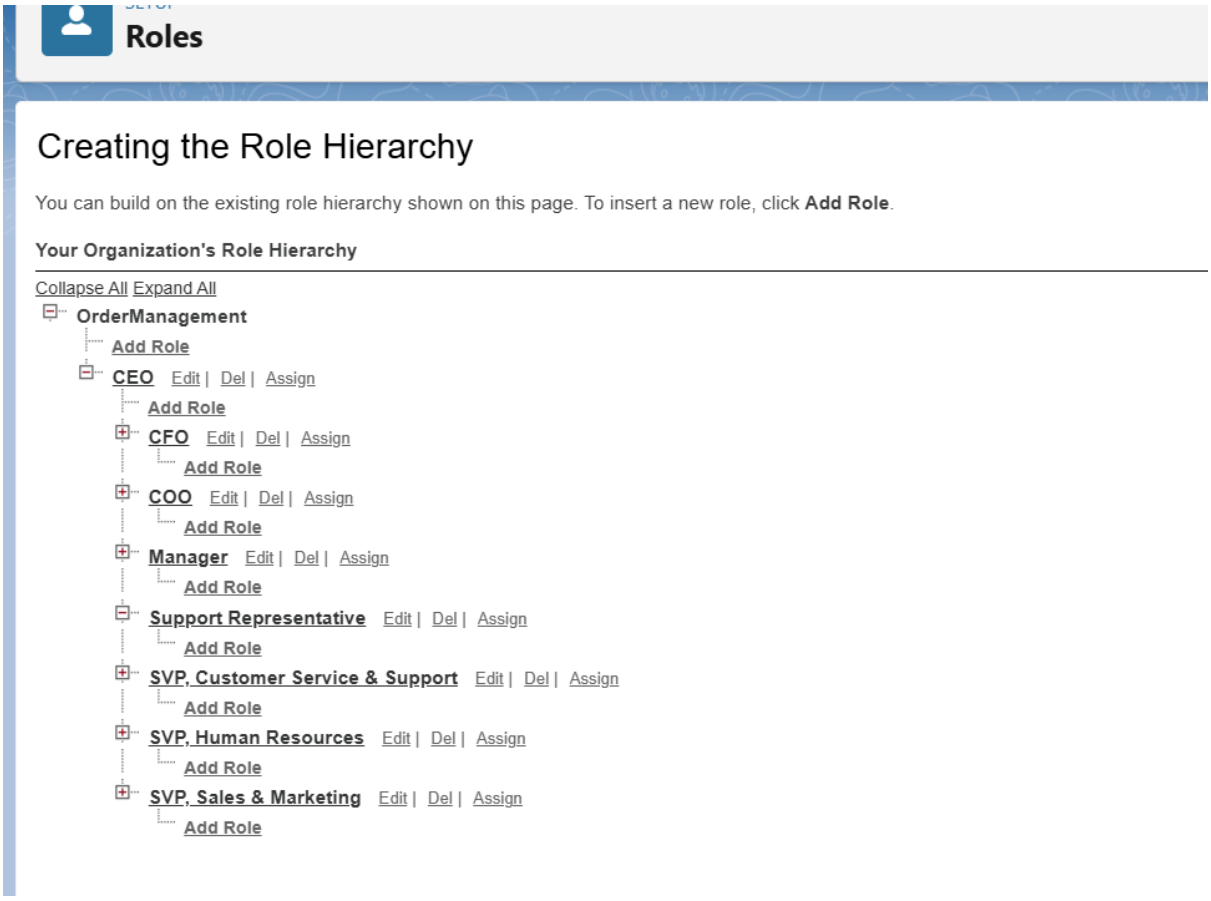
- **Business Profile** → Full access to approve orders.



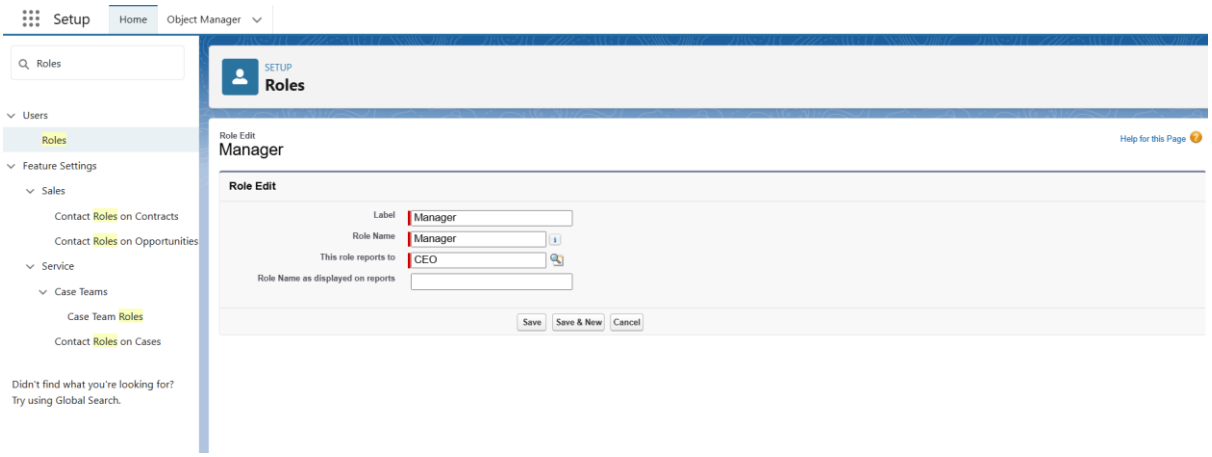
- **Support Profile** → View orders, update issue records.



7. Roles



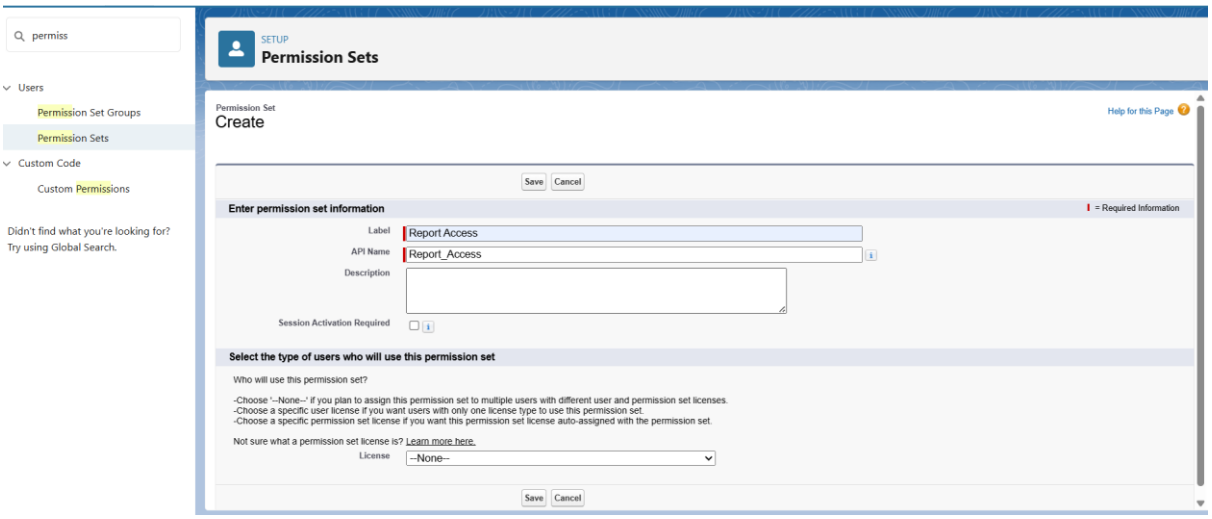
- **Manager (top)** → sees all orders.



- **Sales Rep (below)** → sees only own orders.
- **Support (parallel role)** → sees order issues.

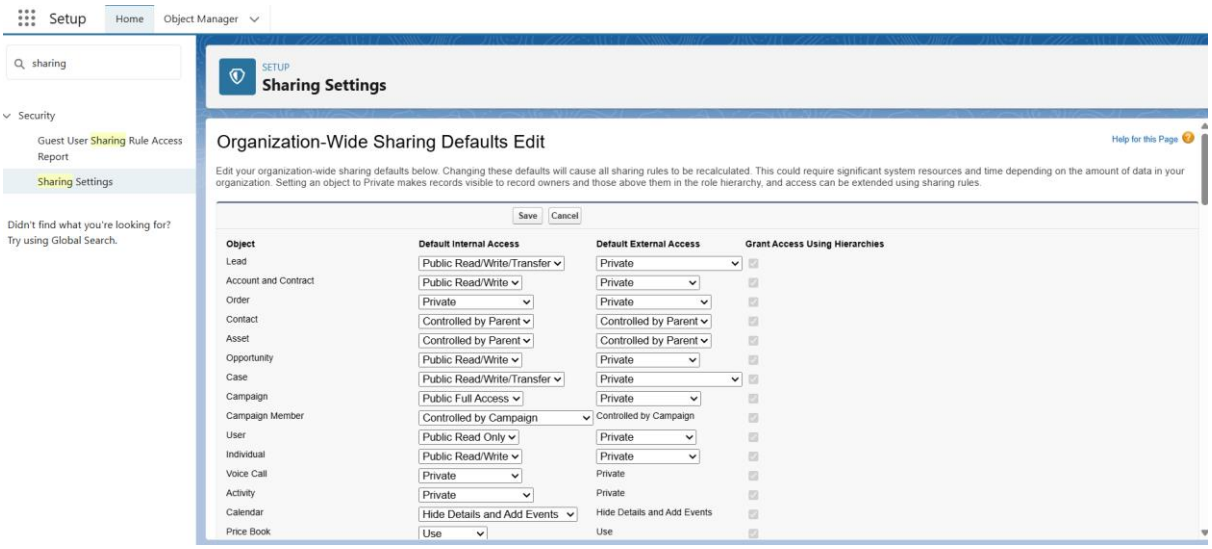
8. Permission Sets

- If Sales Users need access to dashboards, assign Report Access Permission Set instead of modifying profile.



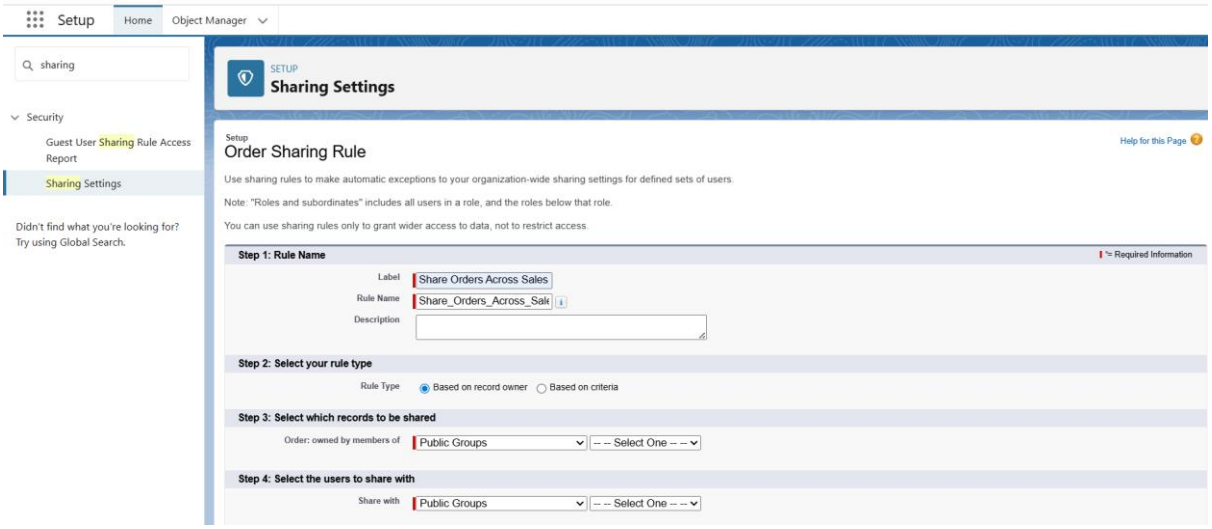
9. Org-Wide Defaults (OWD)

- **Order Object** → Private (only owner + manager can see).
- **Product Object** → Public Read Only (all can view).
- **Invoice Object** → Private.



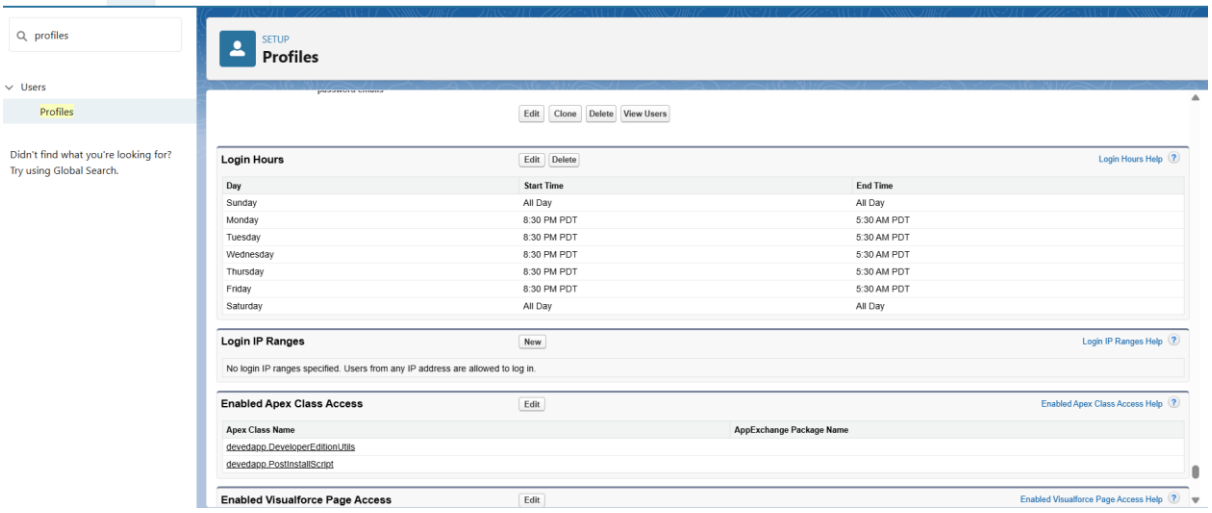
10. Sharing Rules

- Allow sharing of orders across sales teams when required.



11. Login Access Policies

- Restrict sales user logins (e.g., only business hours).



12. Dev Org Setup

- This org acts as sandbox for development & testing.

13. Sandbox & Deployment

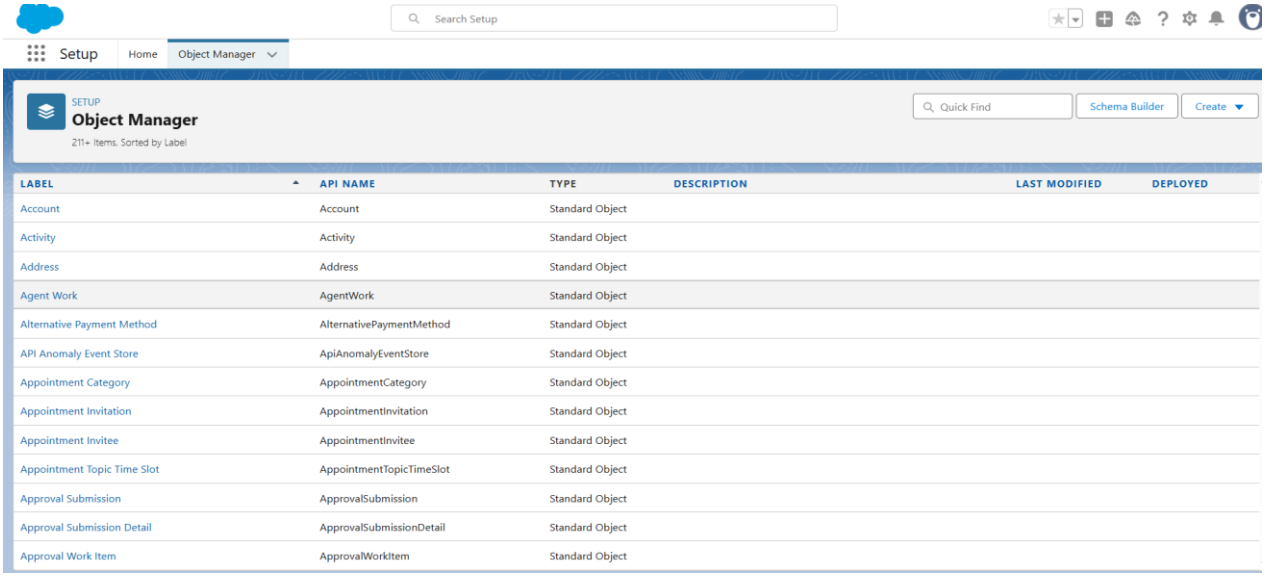
- If enterprise project: Build in Sandbox → Deploy via Change Sets.
- For developer project: Work directly in Dev Org.

Phase 3: Data Modeling & Relationships

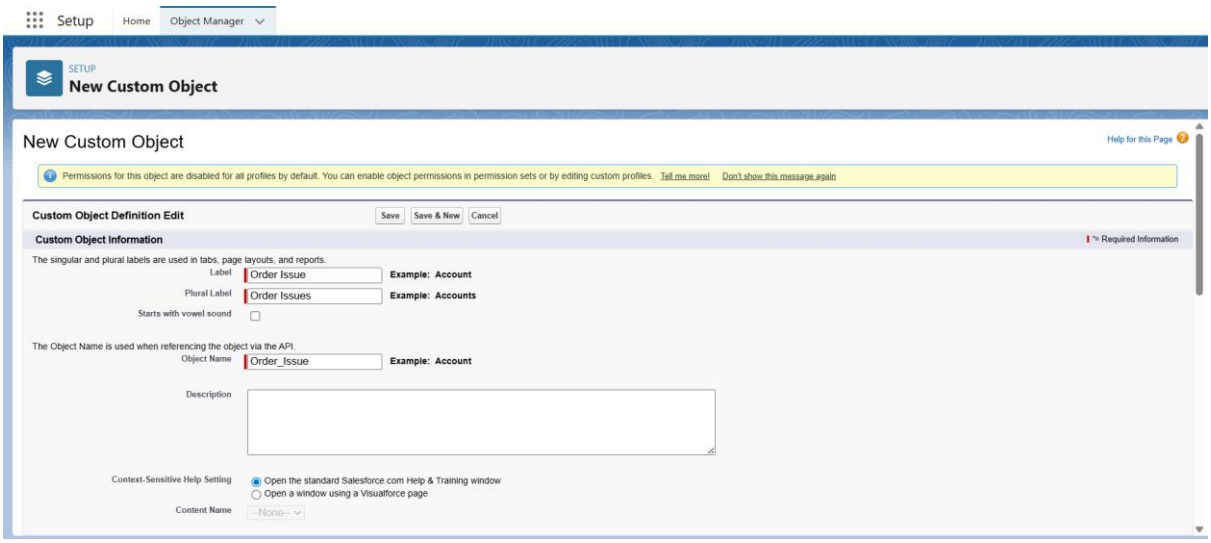
Standard & Custom Objects

Use Case for Order Management Project

- **Standard Objects**: Already provided by Salesforce (Accounts, Contacts, Products, Orders, Invoices, Cases).
    - Example: Order object stores customer orders.
    - Example: Product object stores catalog products.



- **Custom Objects**: Created to store project-specific data not covered by standard objects.
    - Example: Order Issue object to track issues or complaints for a specific order.

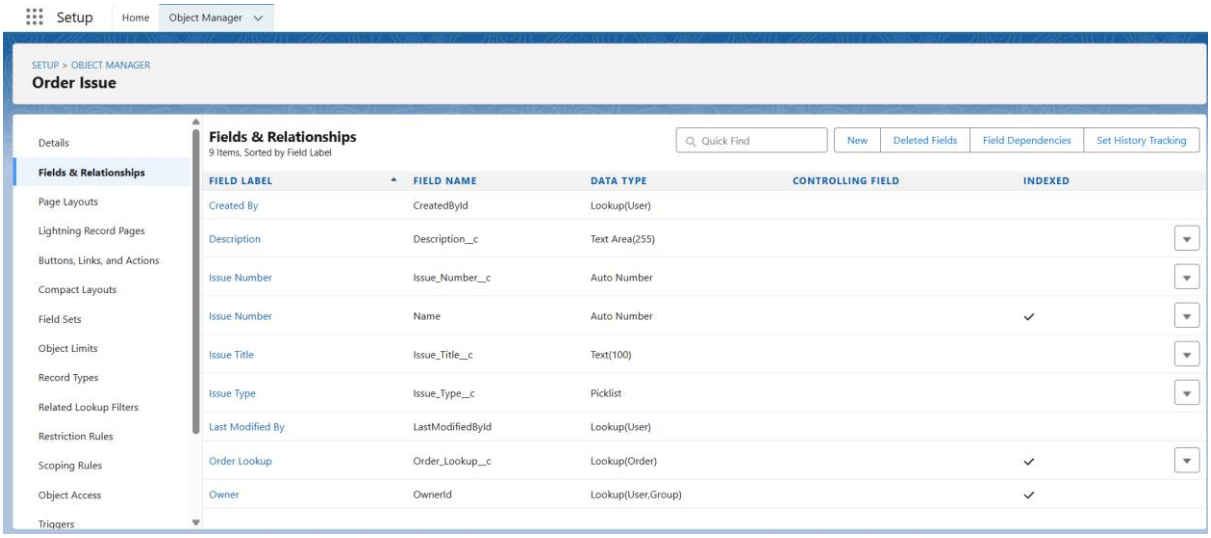


Fields

Use Case for Order Management Project

Fields store **specific pieces of information** in each object.

- **Standard Object Fields:**
  - Order object: Order Date, Status, Total Amount
  - Product object: Product Name, Price, Stock Quantity
- **Custom Object Fields (Order Issue):**
  - Issue Type → Picklist (e.g., Delay, Damage, Incorrect Item)
  - Description → Text Area (for detailed issue info)
  - Order Lookup → Lookup field to connect the issue to a specific order
  - Issue Number → Auto Number (we already discussed)



Why it's needed:

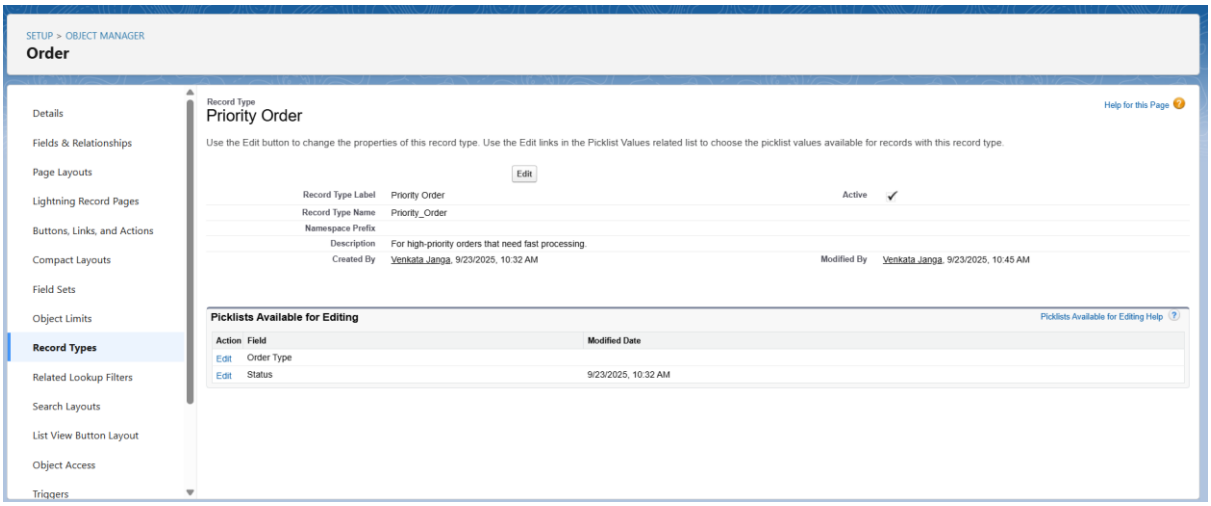
- Fields define what data each record will hold.
- Lookup or Master-Detail fields define relationships between objects.

Record Types

Use Case for Order Management Project

- **Record Types** let you have **different business processes, picklists, and page layouts** for the same object.
- Example:
  - Order object could have:
    - **Standard Orders** → Normal orders from regular customers
    - **Priority Orders** → High-priority orders that need fast processing

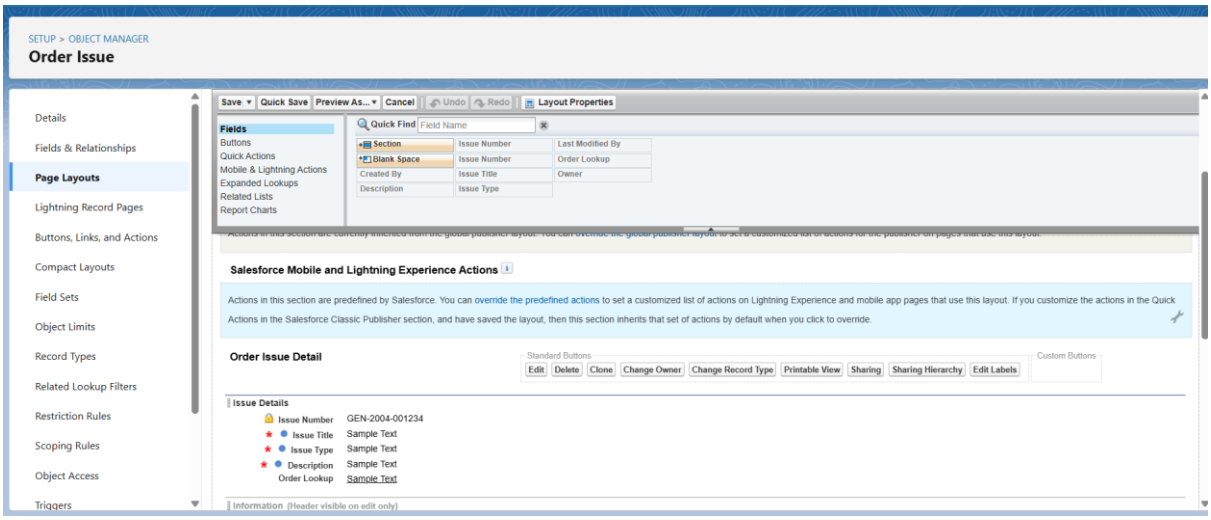
- Different Record Types can show **different picklist values** and **different page layouts** for each type.

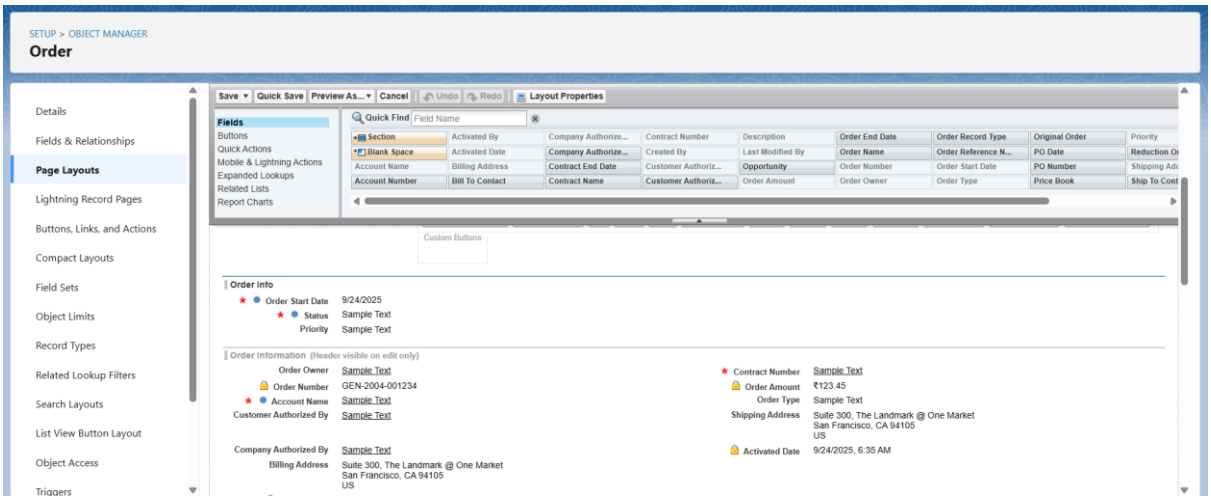


Page Layouts

Use Case

- **Page Layouts** control:
  1. **Which fields are visible on the record detail page**
  2. **Field order**
  3. **Sections** (grouping related fields)
  4. **Related lists** (e.g., show all Order Issues under an Order)
- Example:
  - Order object layout for **Sales User** may show only Order Date, Status, and Product
  - Order layout for **Business Manager** may also include Approval Status and Total Amount





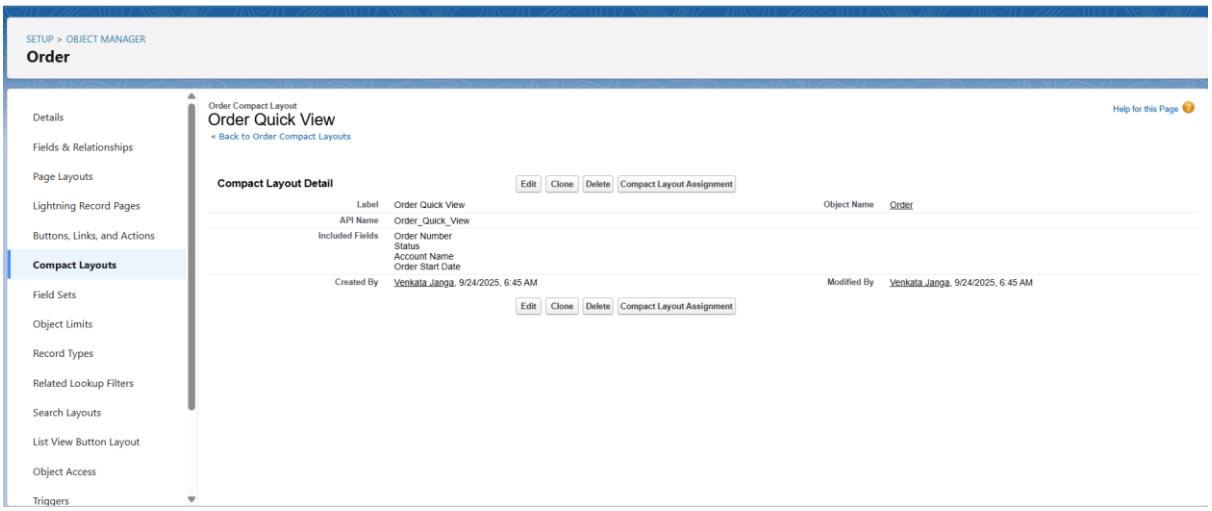
Compact Layouts in Salesforce

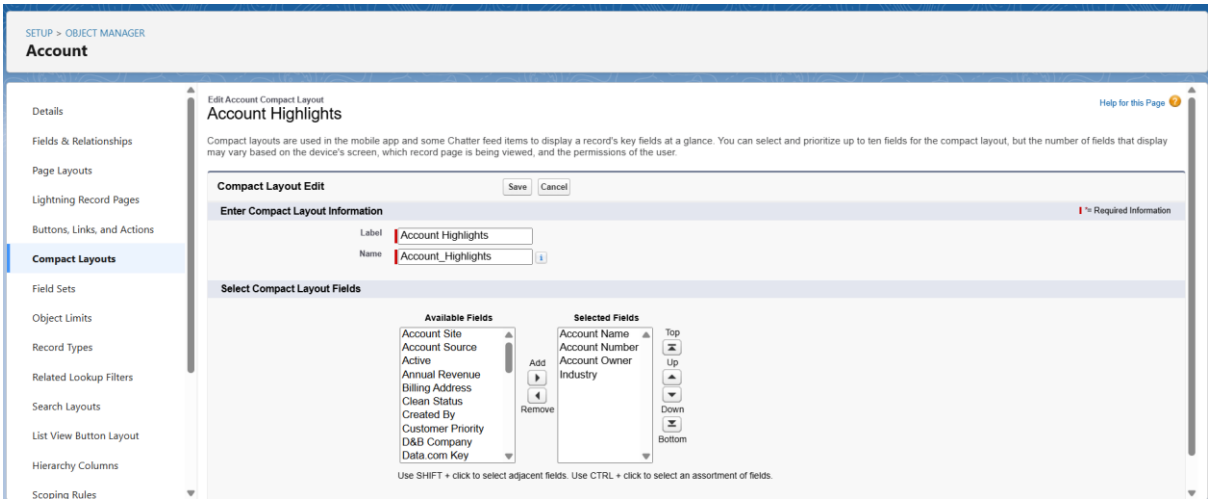
What are Compact Layouts?

- **Page Layouts** → control the **record detail page** (big form when you open a record).
- **Compact Layouts** → control the **highlights panel** at the top of the record.
  - This is the **first few fields** users see when they open a record in Lightning Experience or Salesforce Mobile.

Example:

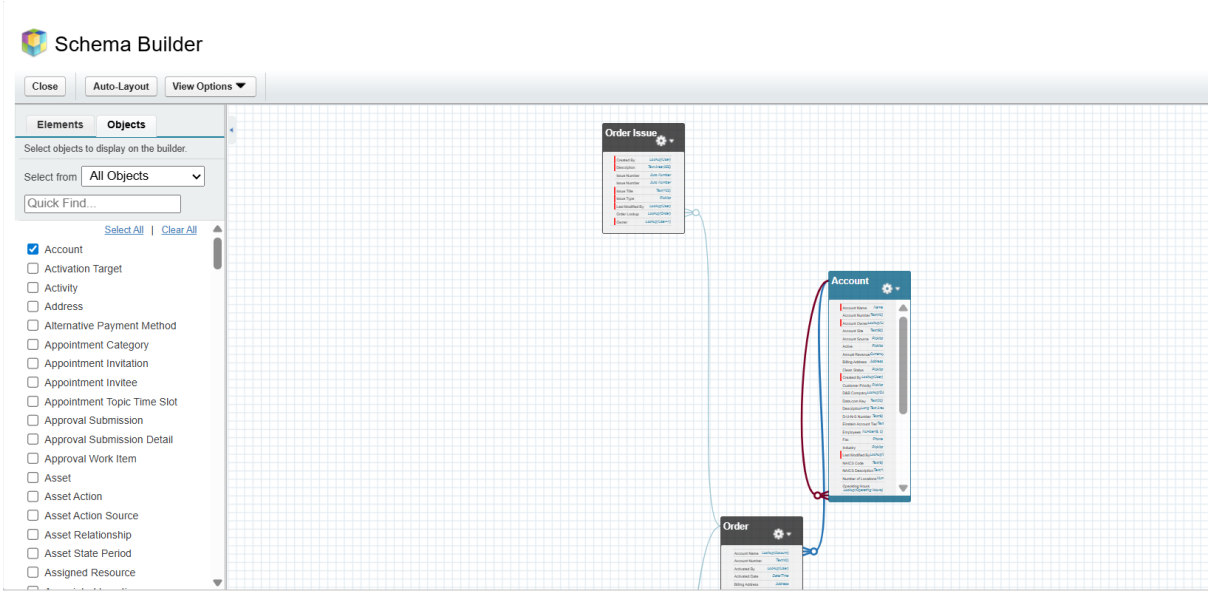
- For an **Order record**, the compact layout could show: **Order Name, Order Date, Status, Priority**.
- For an **Order Issue**, it could show: **Issue Number, Issue Title, Status, Related Order**.





Schema Builder

- Visualize how **Orders, Order Issues, Accounts, Products** are connected.
- Quickly check **relationships** (lookup, master-detail, hierarchical).
- Easier than navigating multiple object managers.
- Useful for **new learners** to understand the data structure.



Lookup vs Master-Detail vs Hierarchical in Order Management

1. **Order → Order Item → Master-Detail**
   - Reason: Order Items cannot exist without an Order.
   - Behavior: Deleting the Order deletes all related Order Items automatically.
2. **Order → Customer → Lookup**
   - Reason: An Order references a Customer, but Customers exist independently of Orders.
   - Behavior: Deleting a Customer may or may not affect Orders (based on configuration).
3. **Payment → Order → Master-Detail**
   - Reason: Payments are always tied to a specific Order.
   - Behavior: Deleting an Order deletes all associated Payments.

Recovery → Order → Lookup

- Reason: Recoveries (like refunds or audits) are linked to Orders but can exist independently.
- Behavior: Deleting an Order doesn't necessarily delete Recovery records.

Order Assignment → Order → Master-Detail

- Reason: Assignment records are only relevant for a particular Order.
- Behavior: Deleting an Order deletes the Assignment record.

Order Assignment → User → Lookup

- Reason: The assigned User exists independently.

- Behavior: Deleting a User may or may not remove the Assignment record.

Hierarchical Relationship

- Typically used for self-referencing objects, e.g., **User → Manager (User)**.
- In Order Management, you may not need this unless modeling something like a chain of approvals.

Junction Objects

- **Purpose:** Used to model **many-to-many relationships** between two objects.
- **Example in Order Management:** Order Product links **Orders** and **Products**.
- **Key Points:**
  - Must have **two Master-Detail relationships**.
  - Allows an Order to have multiple Products and a Product to belong to multiple Orders.
  - Additional fields like quantity, price, or discount can be added to the junction object.
- **Behavior:** Acts as a child of both parent objects; deletion of a parent record removes associated junction records.

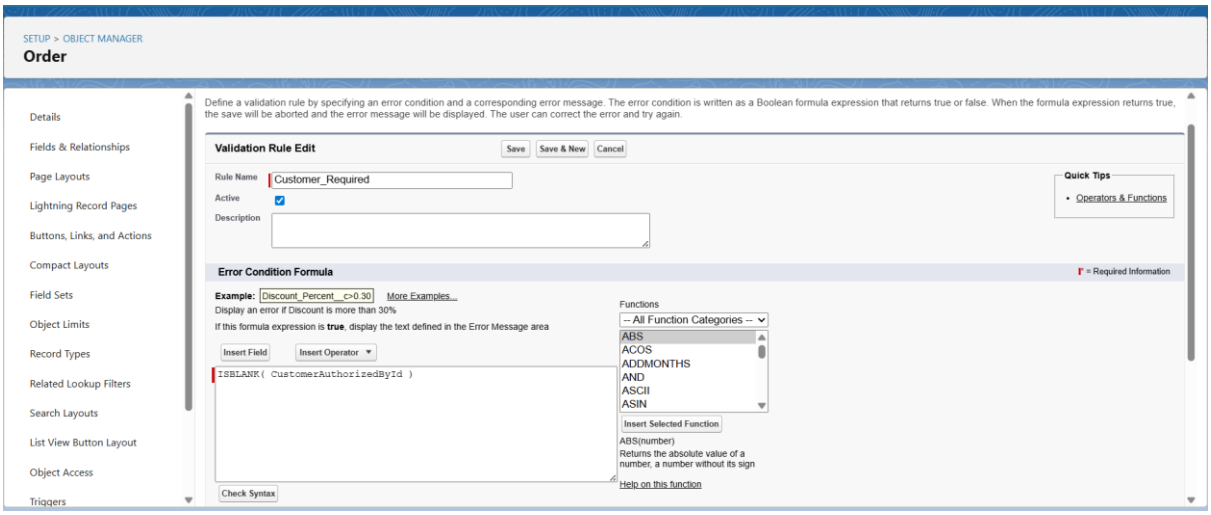
External Objects

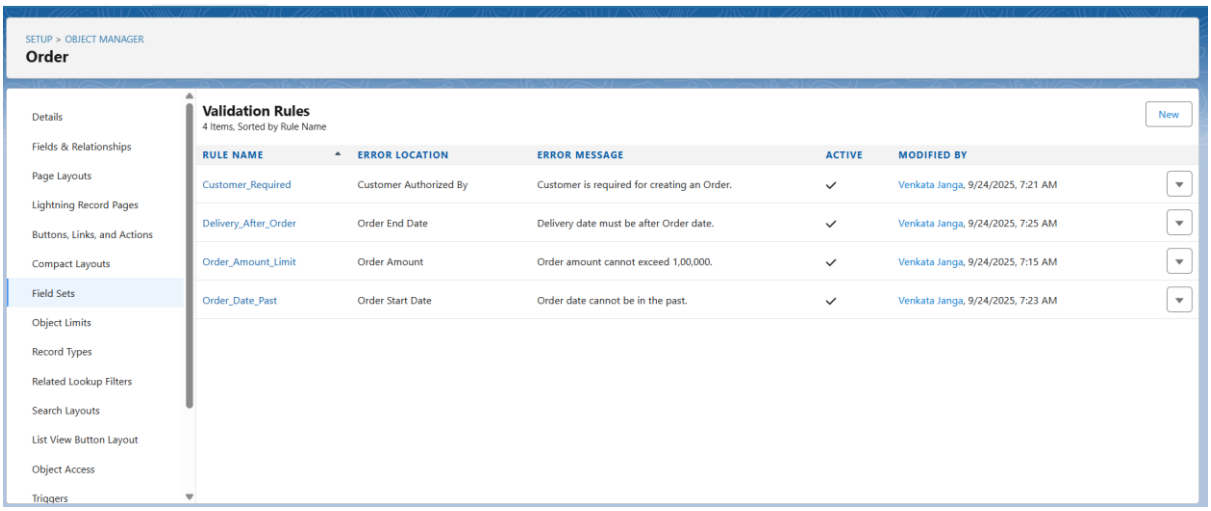
- **Purpose:** Used to access **data stored outside Salesforce** (like external databases or ERP systems) without storing it in Salesforce.
- **Example in Order Management:** External Product Catalog or External Customers.
- **Key Points:**
  - Requires **External Data Source** setup in Salesforce (e.g., OData connection).
  - Supports **Lookup relationships** to link with standard or custom objects.
  - Data is **read-only by default**, unless write operations are enabled.
- **Behavior:** Provides real-time access to external data while keeping Salesforce storage optimized.

Phase 4: Process Automation (Admin)

Validation Rules

Use Case:
Ensure that the Order Amount cannot exceed a certain limit for high-value orders. This prevents data entry errors and maintains data integrity.

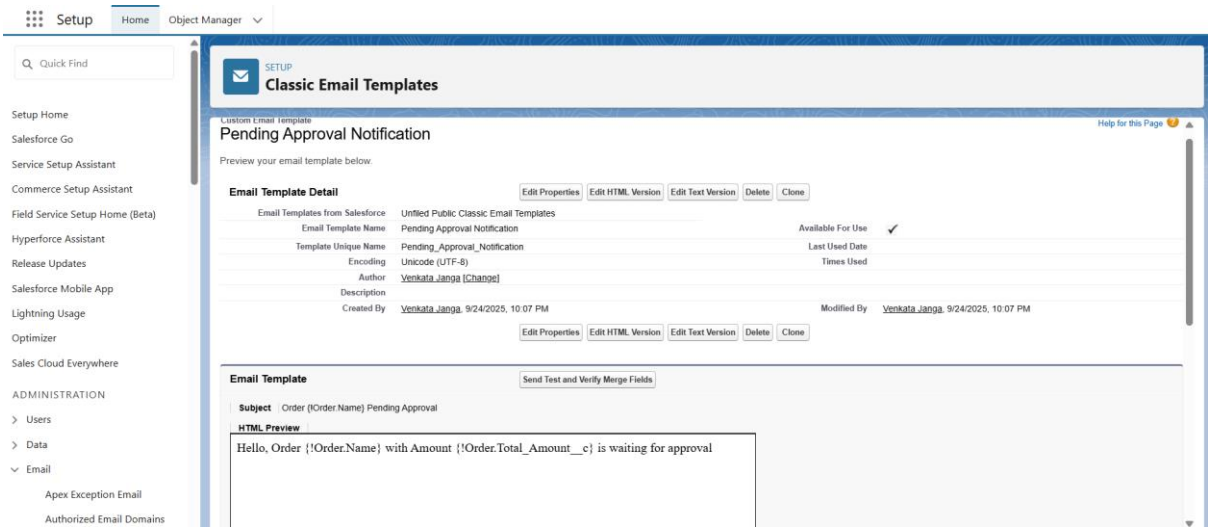


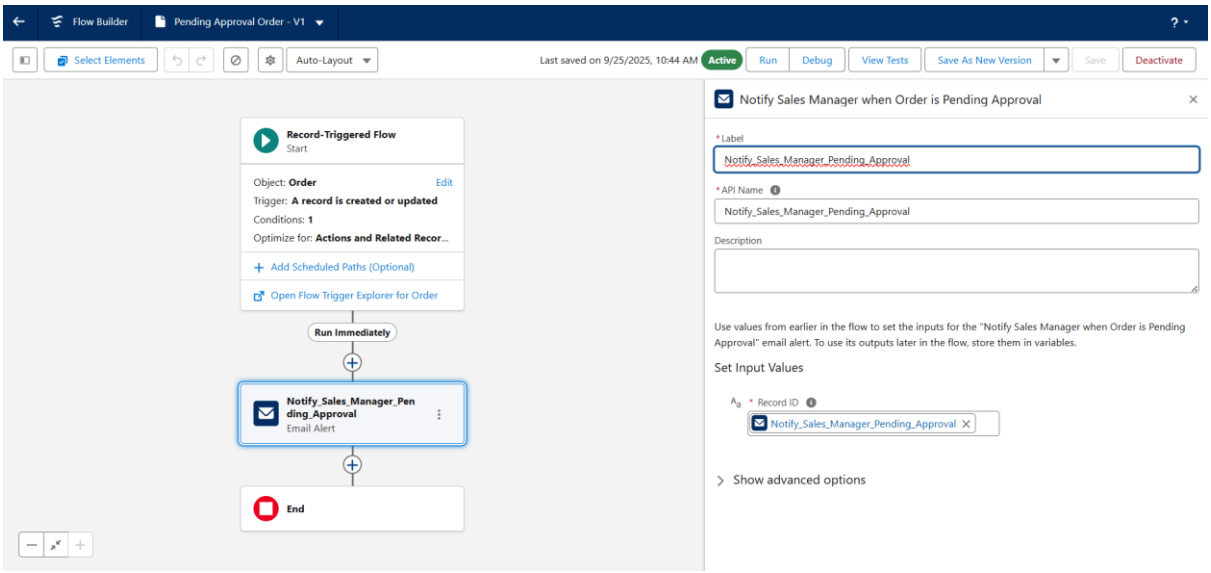


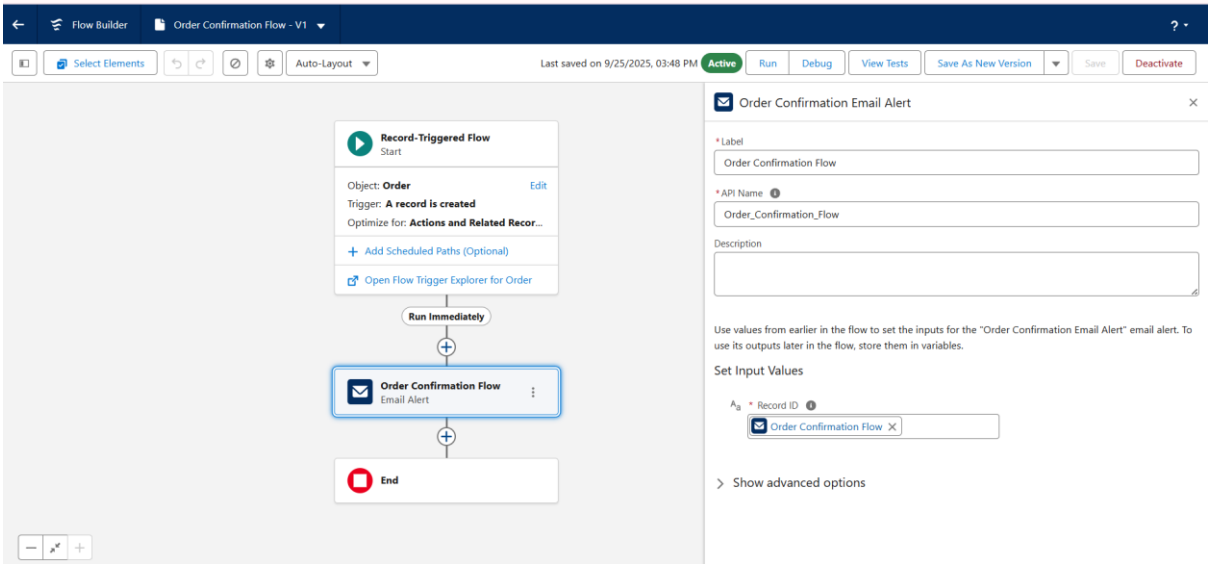
Workflow Rules & Email Alerts

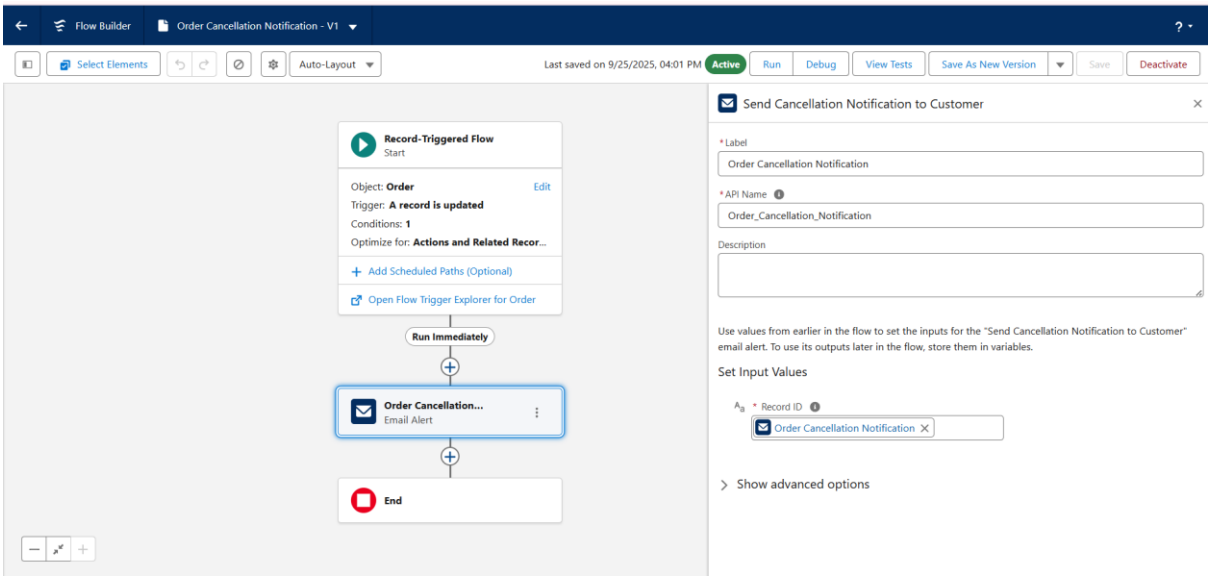
Use Case 1: Notify Sales Manager for Pending Approval Orders

- **Trigger:** When an Order is created or updated to Pending Approval.
- **Action:** Send email to Sales Manager.


Setup
Home
Object Manager
Quick Find
Setup Home
Salesforce Go
Service Setup Assistant
Commerce Setup Assistant
Field Service Setup Home (Beta)
Hyperforce Assistant
Release Updates
Salesforce Mobile App
Lightning Usage
Optimizer
Sales Cloud Everywhere
ADMINISTRATION
Users
Data
Email
Apex Exception Email
Authorized Email Domains
SETUP
Classic Email Templates
Custom Email Template
Pending Approval Notification
Help for this Page
Preview your email template below.
Email Template Detail
Edit Properties
Edit HTML Version
Edit Text Version
Delete
Clone
Email Templates from Salesforce
Unfiled Public Classic Email Templates
Email Template Name
Pending Approval Notification
Available For Use
Template Unique Name
Pending_Approval_Notification
Last Used Date
Encoding
Unicode (UTF-8)
Times Used
Author
Venkata Janga [Change]
Description
Created By
Venkata Janga, 9/24/2025, 10:07 PM
Modified By
Venkata Janga, 9/24/2025, 10:07 PM
Email Template
Send Test and Verify Merge Fields
Subject
Order {!Order.Name} Pending Approval
HTML Preview
Hello, Order {!Order.Name} with Amount {!Order.Total_Amount__c} is waiting for approval


Flow Builder
Pending Approval Order - V1
Select Elements
Auto-Layout
Last saved on 9/25/2025, 10:44 AM
Active
Run
Debug
View Tests
Save As New Version
Save
Deactivate
Record-Triggered Flow
Start
Object: Order
Edit
Trigger: A record is created or updated
Conditions: 1
Optimize for: Actions and Related Recor...
Add Scheduled Paths (Optional)
Open Flow Trigger Explorer for Order
Run Immediately
Notify_Sales_Manager_Pending_Approval
Email Alert
End
Notify Sales Manager when Order is Pending Approval
* Label
Notify_Sales_Manager_Pending_Approval
* API Name
Notify_Sales_Manager_Pending_Approval
Description
Use values from earlier in the flow to set the inputs for the "Notify Sales Manager when Order is Pending Approval" email alert. To use its outputs later in the flow, store them in variables.
Set Input Values
* Record ID
Notify_Sales_Manager_Pending_Approval
Show advanced options


Flow Builder
Order Confirmation Flow - V1
Select Elements
Auto-Layout
Last saved on 9/25/2025, 03:48 PM
Active
Run
Debug
View Tests
Save As New Version
Save
Deactivate
Record-Triggered Flow
Start
Object: Order
Edit
Trigger: A record is created
Optimize for: Actions and Related Recor...
Add Scheduled Paths (Optional)
Open Flow Trigger Explorer for Order
Run Immediately
Order Confirmation Flow
Email Alert
End
Order Confirmation Email Alert
* Label
Order Confirmation Flow
* API Name
Order_Confirmation_Flow
Description
Use values from earlier in the flow to set the inputs for the "Order Confirmation Email Alert" email alert. To use its outputs later in the flow, store them in variables.
Set Input Values
* Record ID
Order Confirmation Flow
Show advanced options



Process Automation

We still need to cover the other automation types for your Order Management Project:

1. Field Updates

- Example:
  When an order status is set to Shipped, automatically set the Delivery Date = Today + 7 days.
  (This saves manual work for users.)

2. Tasks

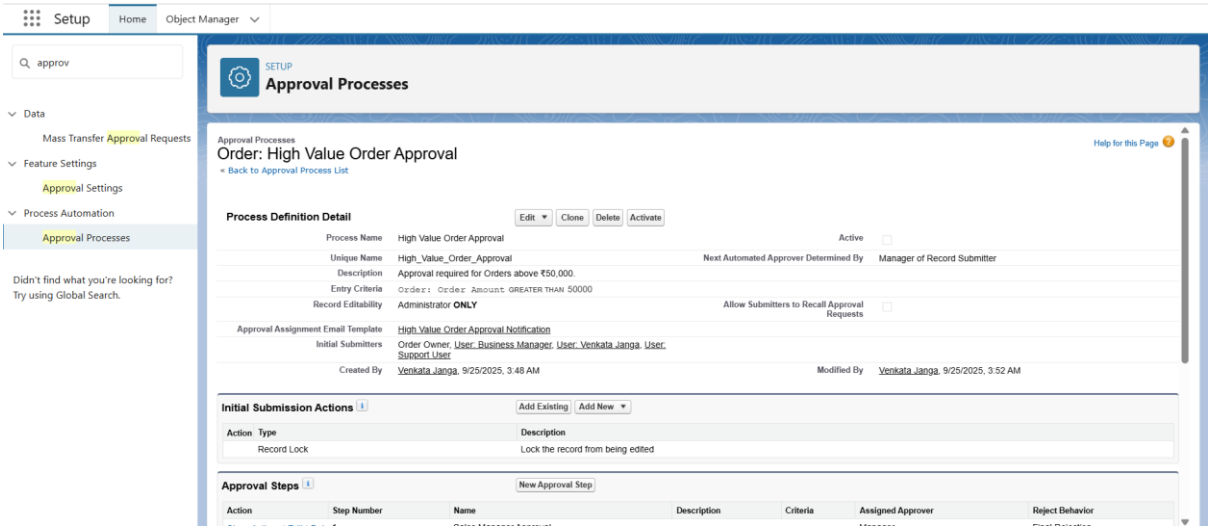
- Example:
  When an order is cancelled, create a Follow-up Task for the Sales Rep to call the customer.

3. Custom Notifications (In-App / Mobile)

- Example:
  When an order is in Pending Approval, send a Custom Notification to the Sales Manager in Salesforce mobile/desktop app.

4. Approval Process

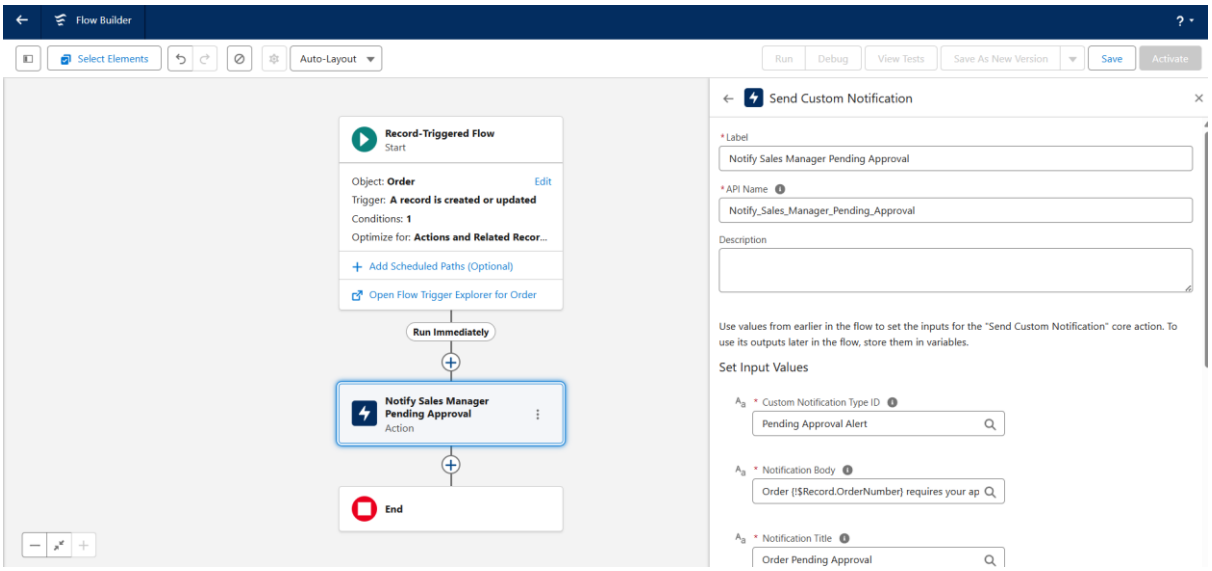
- Example:
  When an order > ₹50,000 is submitted, it must go through an Approval Process (Sales Manager approves/rejects).



Custom Notifications (In-App / Mobile Alerts)

Use Case

- Notify the Sales Manager inside Salesforce (bell icon or mobile app) whenever an order is Pending Approval.
- This is useful in addition to email, so they don't miss high-value approvals.



Phase 5: Apex Programming (Developer)

Classes & Objects

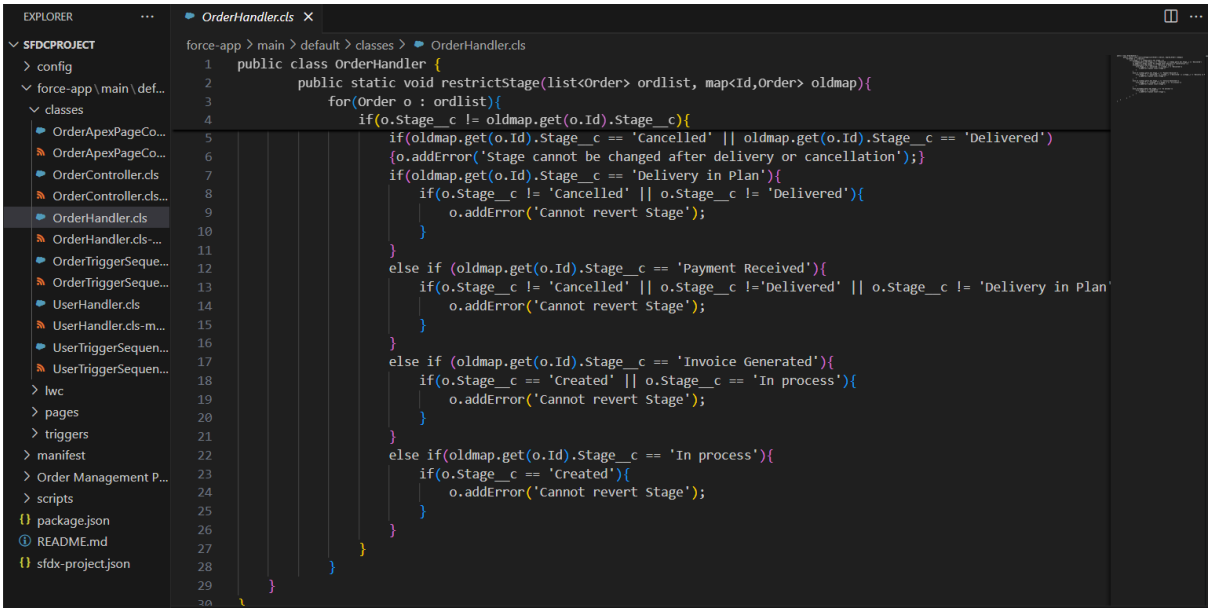
Use Case

- Classes in Apex allow you to encapsulate logic that can be reused in multiple places, such as calculating order discounts, validating order totals, or processing shipments.
- Objects represent your Salesforce data, like Order, Customer, Product, etc.

OrderHandler.cls

```
public class OrderHandler {
  public static void restrictStage(list<Order> ordlist, map<Id,Order> oldmap){
    for(Order o : ordlist){
      if(o.Stage__c != oldmap.get(o.Id).Stage__c){
        if(oldmap.get(o.Id).Stage__c == 'Cancelled' || oldmap.get(o.Id).Stage__c
== 'Delivered')
        {o.addError('Stage cannot be changed after delivery or cancellation');}
        if(oldmap.get(o.Id).Stage__c == 'Delivery in Plan'){
          if(o.Stage__c != 'Cancelled' || o.Stage__c != 'Delivered'){
            o.addError('Cannot revert Stage');
          }
        }
        else if (oldmap.get(o.Id).Stage__c == 'Payment Received'){
          if(o.Stage__c != 'Cancelled' || o.Stage__c !='Delivered' || o.Stage__c !=
'Delivery in Plan'){
            o.addError('Cannot revert Stage');
          }
        }
        else if (oldmap.get(o.Id).Stage__c == 'Invoice Generated'){
          if(o.Stage__c == 'Created' || o.Stage__c == 'In process'){
            o.addError('Cannot revert Stage');
```

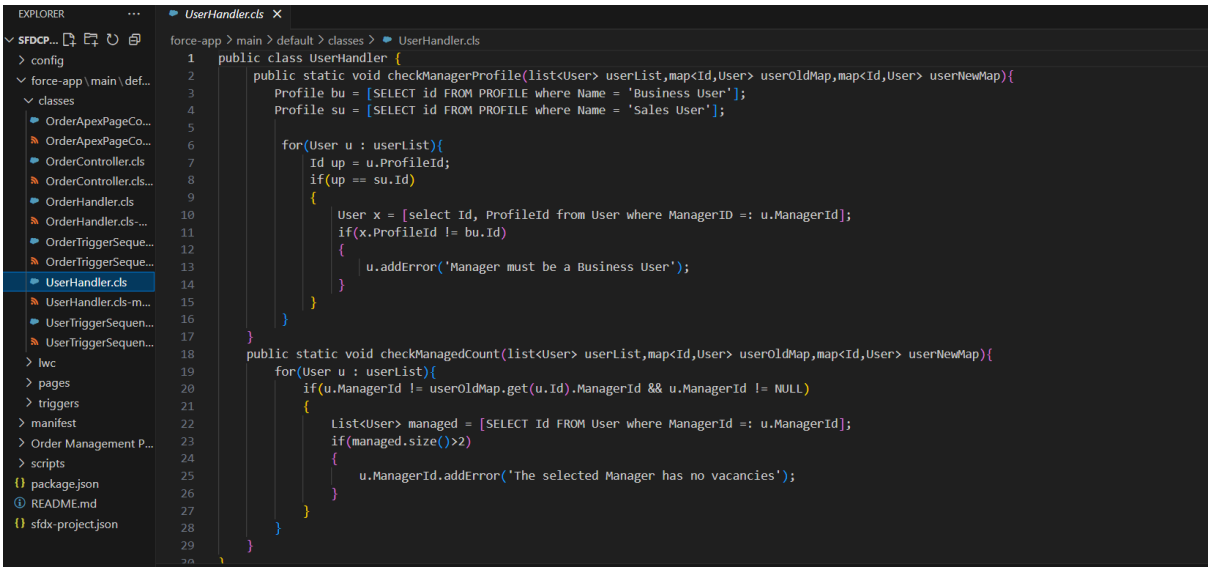
```
                }
            }
            else if(oldmap.get(o.Id).Stage__c == 'In process'){
                if(o.Stage__c == 'Created'){
                    o.addError('Cannot revert Stage');
                }
            }
        }
    }
  }
}
```



UserHandler.cls

```
public class UserHandler {
    public static void checkManagerProfile(list<User> userList,map<Id,User> userOldMap,map<Id,User> userNewMap){
        Profile bu = [SELECT id FROM PROFILE where Name = 'Business User'];
        Profile su = [SELECT id FROM PROFILE where Name = 'Sales User'];
```

```
        for(User u : userList){
            Id up = u.ProfileId;
            if(up == su.Id)
            {
                User x = [select Id, ProfileId from User where ManagerID =: u.ManagerId];
                if(x.ProfileId != bu.Id)
                {
                    u.addError('Manager must be a Business User');
                }
            }
        }
    }
    public static void checkManagedCount(list<User> userList,map<Id,User> userOldMap,map<Id,User> userNewMap){
        for(User u : userList){
            if(u.ManagerId != userOldMap.get(u.Id).ManagerId && u.ManagerId != NULL)
            {
                List<User> managed = [SELECT Id FROM User where ManagerId =: u.ManagerId];
                if(managed.size()>2)
                {
                    u.ManagerId.addError('The selected Manager has no vacancies');
                }
            }
        }
    }
}
```



OrderController.cls

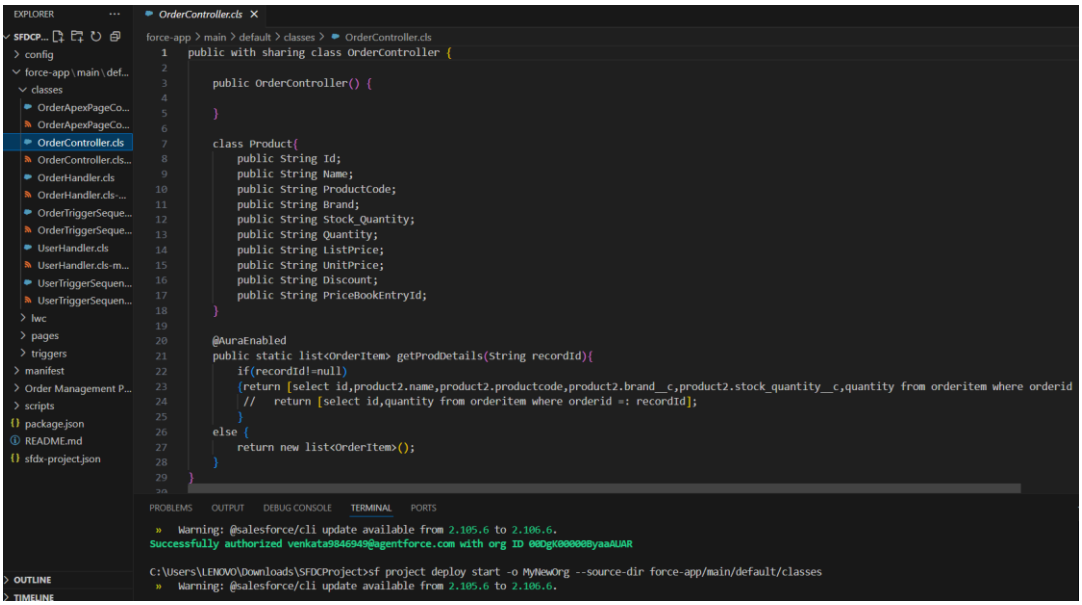
```
public with sharing class OrderController {
    public OrderController() {
    }
    class Product{
        public String Id;
        public String Name;
        public String ProductCode;
        public String Brand;
        public String Stock_Quantity;
        public String Quantity;
        public String ListPrice;
        public String UnitPrice;
        public String Discount;
        public String PriceBookEntryId;
    }

    @AuraEnabled
    public static list<OrderItem> getProdDetails(String recordId){
```

```
        if(recordId!=null)
                                                    {return [select id,product2.name,product2.productcode,product2.brand__c,product2.stock_quantity__c,quantity from orderitem where orderid =: recordId];
         //   return [select id,quantity from orderitem where orderid =: recordId];
        }
    else {
        return new list<OrderItem>();
    }
}


    @AuraEnabled
    public static list<Order> getOrdDetails(String recordId){
        if(recordId!=null){
        return [select id,totalamount,total_qty__c from order where id=:recordid];
    }
    else{
        return new list<Order> ();
    }
}


    // @AuraEnabled
    // public static Integer getTotalQty (String recordId){
     //     AggregateResult[] agResults = [select sum(quantity) from orderitem where orderid=:recordId];
    //     Integer qty = (Integer)agResults[0].get('qty');
    //     return qty;
    // }
    @AuraEnabled
```

```
public static string getRecordId(){
    list<Order> recId = [SELECT Id FROM Order ORDER BY CreatedDate DESC LIMIT 1];
    String rId = '' + recId[0].Id;
    return rId;
}

@AuraEnabled
public static string getProducts(String searchBy, String searchText, String pbId){
    map<Id,Product2> prodList;
    Map<Id,PriceBookEntry> priceMap;
    if(searchBy.equals('Name')){
        prodList= new map<Id,Product2>([select Id,ProductCode,Name,Brand__c,Stock_Quantity__c from Product2 where Name LIKE :'%'+searchText+'%']);
    }
    if(searchBy.equals('Brand__c')){
        prodList= new map<Id,Product2>([select Id,ProductCode,Name,Brand__c,Stock_Quantity__c from Product2 where Brand__c LIKE :'%'+searchText+'%']);
    }
    if(searchBy.equals('UnitPrice')){
        if(searchText.isNumeric()){
            Decimal dec = Decimal.valueOf(searchText);
        priceMap = new Map<Id,PriceBookEntry>([select Product2Id,name,unitprice from pricebookentry where unitprice =: dec]);
        prodList= new map<Id,Product2>([select Id,ProductCode,Name,Brand__c,Stock_Quantity__c from Product2 where Id in : priceMap.keySet()]);
        }
    }
```

```
        list<PricebookEntry> pbeList=[select Id,Product2Id,UnitPrice from PricebookEntry where Product2Id IN :prodList.keySet() AND Pricebook2Id=:pbId];
    map<Id,String> priceBookList=new map<Id,String>();
    map<Id,Decimal> unitPriceList=new map<Id,Decimal>();
    for(PricebookEntry item:pbeList){
        priceBookList.put(item.Product2Id,item.Id);
        unitPriceList.put(item.Product2Id,item.UnitPrice);
    }
    String str='[';
    for(Id key:prodList.keySet()){
        str+='{"Id":"'+key+'","'+'Name":"'+prodList.get(key).Name+'","'+'ProductCode":"'+prodList.get(key).ProductCode+'","'+'ListPrice":"'+unitPriceList.get(key)+'","'+'Quantity":"","Brand__c":"'+prodList.get(key).Brand__c+'","'+'Stock_Quantity__c":"'+prodList.get(key).Stock_Quantity__c+'","'+'PriceBookEntryId":"'+priceBookList.get(key)+'"}'+',';
    }
    str = str.substring(0, str.length()-1);
    str+=']';
    System.debug(str);
    if(str.equals(']')){
        str='';
    }
    return str;
  }

  @AuraEnabled
    public static string createOrderProducts(String selectedProducts,String priceBookId, String orderId){

    List<Product> productList = (List<Product>)JSON.deserialize(selectedProducts, List<Product>.class);
```

```
        Order order=[Select Stage__c,Pricebook2Id from Order where Id=:orderId];
        order.Pricebook2Id = priceBookId;
        order.Stage__c = 'In process';
        update order;

        list<OrderItem> orderItemList = new list<OrderItem>();
        try{
            for(Product product:productList){
                                                                    orderItemList.add(new
OrderItem(OrderId=orderId,Product2Id=product.Id,Quantity=Decimal.valueOf(prod
uct.Quantity),UnitPrice=Decimal.valueOf(product.UnitPrice),PricebookEntryId=pro
duct.PriceBookEntryId));
        }
        insert orderItemList;
        return 'Your order is ready';}
        catch(Exception e){
            return 'Error';
        }
    }
}
```

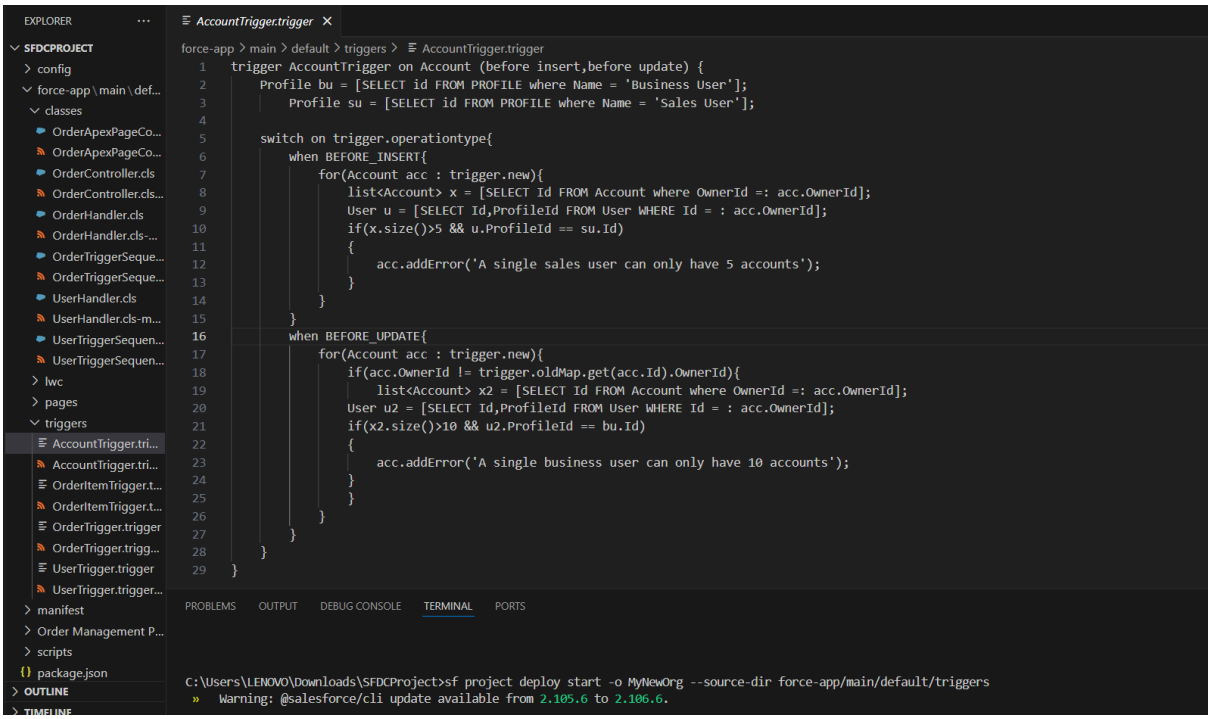


Triggers

i)AccountTrigger.trigger

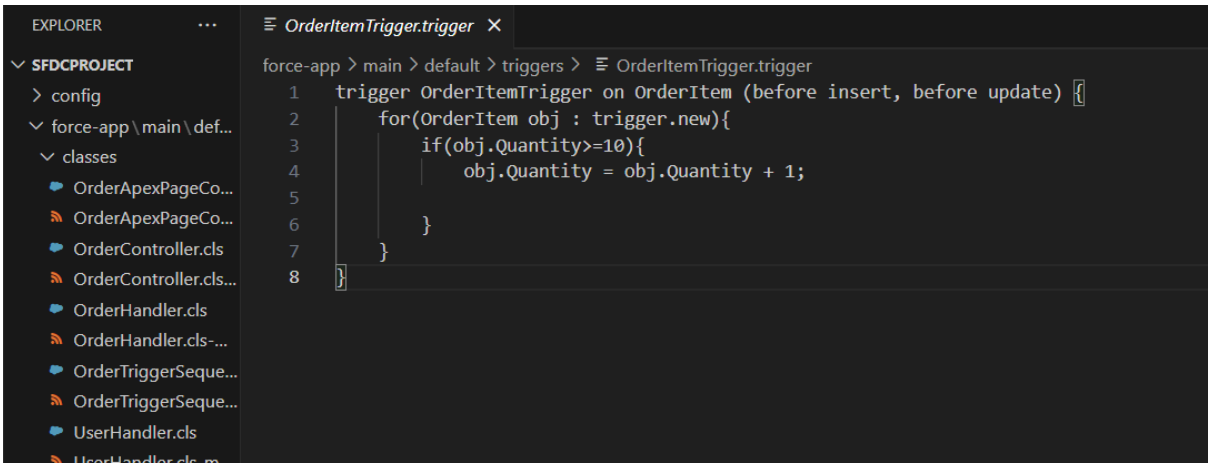
```
trigger AccountTrigger on Account (before insert,before update) {
   Profile bu = [SELECT id FROM PROFILE where Name = 'Business User'];
      Profile su = [SELECT id FROM PROFILE where Name = 'Sales User'];


   switch on trigger.operationtype{
      when BEFORE_INSERT{
         for(Account acc : trigger.new){
            list<Account> x = [SELECT Id FROM Account where OwnerId =: acc.OwnerId];
            User u = [SELECT Id,ProfileId FROM User WHERE Id = : acc.OwnerId];
            if(x.size()>5 && u.ProfileId == su.Id)
            {
               acc.addError('A single sales user can only have 5 accounts');
            }
         }
      }
      when BEFORE_UPDATE{
         for(Account acc : trigger.new){
            if(acc.OwnerId != trigger.oldMap.get(acc.Id).OwnerId){
               list<Account> x2 = [SELECT Id FROM Account where OwnerId =: acc.OwnerId];
            User u2 = [SELECT Id,ProfileId FROM User WHERE Id = : acc.OwnerId];
            if(x2.size()>10 && u2.ProfileId == bu.Id)
            {
               acc.addError('A single business user can only have 10 accounts');
            }
            }
```

```
            }
        }
    }
}
```



ii) OrderItemTrigger.trigger

```
trigger OrderItemTrigger on OrderItem (before insert, before update) {
    for(OrderItem obj : trigger.new){
        if(obj.Quantity>=10){
            obj.Quantity = obj.Quantity + 1;
        }
    }}
```

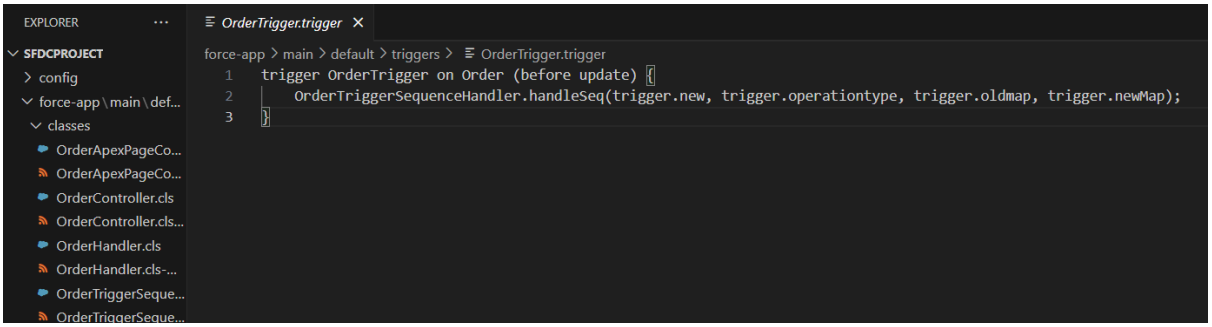


iii) OrderTrigger.trigger

trigger OrderTrigger on Order (before update) {

OrderTriggerSequenceHandler.handleSeq(trigger.new, trigger.operationtype, trigger.oldmap, trigger.newMap);

}

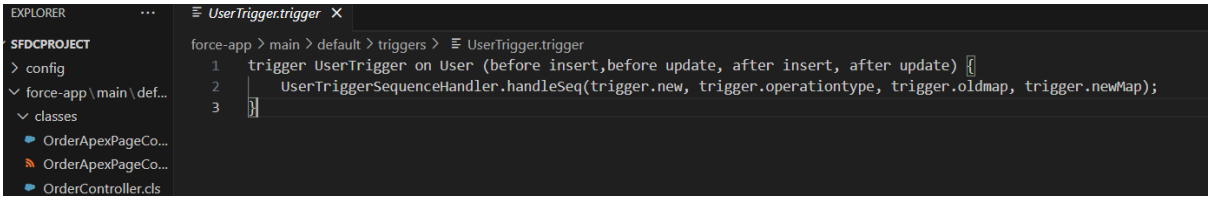


iv) UserTrigger.trigger

trigger UserTrigger on User (before insert,before update, after insert, after update) {

UserTriggerSequenceHandler.handleSeq(trigger.new, trigger.operationtype, trigger.oldmap, trigger.newMap);

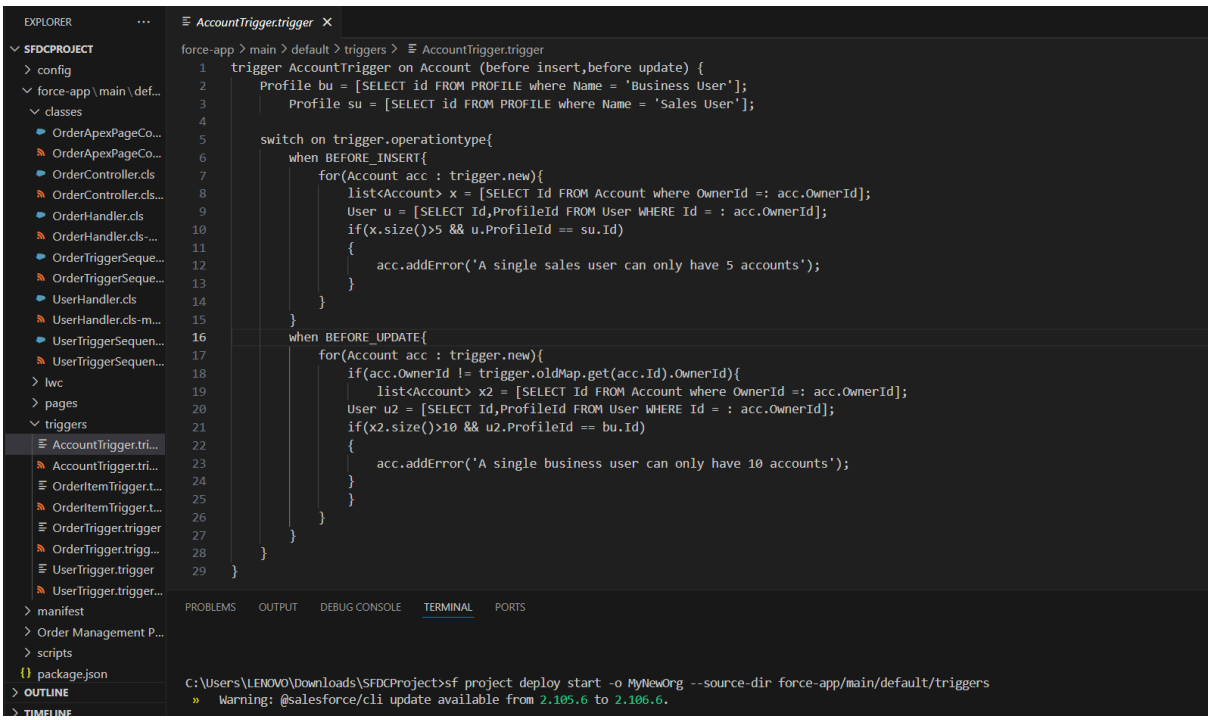
}



Trigger Design Pattern

In Salesforce, triggers can quickly become hard to maintain if all logic is written directly inside the trigger.

- The Trigger Design Pattern separates the business logic from the trigger execution.
- Benefits:
    1. One Trigger Per Object – keeps triggers clean and avoids multiple triggers firing for the same object.
    2. Reusability – business logic is centralized in a Handler Class.
    3. Easier Testing & Maintenance – changes in logic don't require modifying the trigger itself.
- Example Use Case for Order Management:
    - Before Update: Restrict stage changes (using OrderHandler.restrictStage)
    - After Insert: Automatically send notification emails to Sales Manager



SOQL (Salesforce Object Query Language)

Purpose / Use Case

- SOQL is used to query Salesforce records from a single object or related objects.
- In your Order Management Project, you might use SOQL to:
    1. Retrieve all orders for a particular customer.
    2. Get all high-value orders (e.g., Total Amount > ₹50,000).
    3. Fetch related product details for orders.

Example 1: Get Orders for a Customer

```
Id customerId = '001XXXXXXXXXXXX'; // Example Account Id
List<Order__c> orders = [SELECT Id, Name, Stage__c, Total_Amount__c
                    FROM Order__c
                    WHERE Customer__c = :customerId];
```

Example 2: Get High-Value Orders

```
List<Order__c> highValueOrders = [SELECT Id, Name, Total_Amount__c
                        FROM Order__c
                        WHERE Total_Amount__c > 50000];
```

Example 3: Get Products Linked to Orders

```
List<Order_Item__c> orderItems = [SELECT Id, Product__r.Name, Product__r.Brand__c, Quantity__c
                        FROM Order_Item__c
                        WHERE Order__c = :orderId];
```

SOSL (Salesforce Object Search Language)

Purpose / Use Case

- SOSL is used to search across multiple objects and fields simultaneously.
- Useful when you don't know exactly which object the data resides in, e.g., searching for a customer by name across Accounts, Contacts, and Orders.

Example: Search for a Customer Across Multiple Objects

```
List<List<SObject>> searchResults = [FIND 'John Doe*' IN ALL FIELDS
                        RETURNING
                        Account(Id, Name),
                        Contact(Id, Name, Email),
                        Order__c(Id, Name, Stage__c)];
```

- searchResults[0] → Accounts
- searchResults[1] → Contacts
- searchResults[2] → Orders

```
    @AuraEnabled
    public static list<OrderItem> getProdDetails(String recordId){
        if(recordId!=null)
        {return [select id,product2.name,product2.productcode,product2.brand__c,product2.stock_quantity__c,quantity from orderitem where orderid =:
         //   return [select id,quantity from orderitem where orderid =: recordId];
        }
    else {
        return new list<OrderItem>();
    }
}

    @AuraEnabled
    public static list<Order> getOrdDetails(String recordId){
        if(recordId!=null){
        return [select id,totalamount,total_qty__c from order where id=:recordid];
    }
    else{
        return new list<Order> ();
    }
}

    // @AuraEnabled
    // public static Integer getTotalQty (String recordId){
    //     AggregateResult[] agResults = [select sum(quantity) from orderitem where orderid=:recordId];
    //     Integer qty = (Integer)agResults[0].get('qty');
    //     return qty;
    // }
```

Collections: List, Set, Map

Collections are used to store multiple records or values efficiently.

- In your project:
  - List: Store multiple orders to process in batch.
  - Set: Avoid duplicates (e.g., customer IDs).
  - Map: Store key-value pairs for fast lookup (e.g., Order Id → Stage).

Examples

```
// List
List<Order__c> orders = [SELECT Id, Name, Total_Amount__c FROM Order__c];
// Set
Set<Id> customerIds = new Set<Id>();
for(Order__c ord : orders){
    customerIds.add(ord.Customer__c);
}
// Map
Map<Id, Order__c> orderMap = new Map<Id, Order__c>(orders);
```

Control Statements

Conditional logic and loops to control Apex code execution.

- Example: Apply discounts based on order amount.

```
for(Order__c ord : orders){
```

```
    if(ord.Total_Amount__c >= 50000){
        ord.Discount__c = ord.Total_Amount__c * 0.10;
    } else if(ord.Total_Amount__c >= 20000){
        ord.Discount__c = ord.Total_Amount__c * 0.05;
    } else {
        ord.Discount__c = 0;
    }
}
```

Batch Apex

Process large data sets asynchronously (50,000+ records).

- Example: Update all shipped orders without a delivery date nightly.

```
global class BatchUpdateDeliveryDate implements Database.Batchable<sObject> {
    global Database.QueryLocator start(Database.BatchableContext BC){
        return Database.getQueryLocator('SELECT Id FROM Order__c WHERE Stage__c
= \'Shipped\' AND Delivery_Date__c = NULL');
    }
    global void execute(Database.BatchableContext BC, List<Order__c> scope){
        for(Order__c ord : scope){
            ord.Delivery_Date__c = Date.today().addDays(7);
        }
        update scope;
    }

    global void finish(Database.BatchableContext BC){
        System.debug('Batch Completed.');
    }
}
```

Queueable Apex

Asynchronous processing like Batch Apex, but simpler and allows chaining jobs.

- Example: Send notifications after inserting high-value orders.

```
public class OrderNotificationQueueable implements Queueable {
   List<Id> orderIds;
   public OrderNotificationQueueable(List<Id> ids){
      this.orderIds = ids;
   }
   public void execute(QueueableContext ctx){
      OrderHandler.notifySalesManager(orderIds);
   }
}
```

Future Methods

- Run methods asynchronously. Useful for sending emails or callouts.
- Example: Notify Sales Manager after order insert.

```
@future
public static void notifySalesManager(List<Id> orderIds){
   OrderHandler.notifySalesManager(orderIds);
}
```

Exception Handling

- Handle errors gracefully without stopping the execution.
- Example: Wrap update logic in try-catch.

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

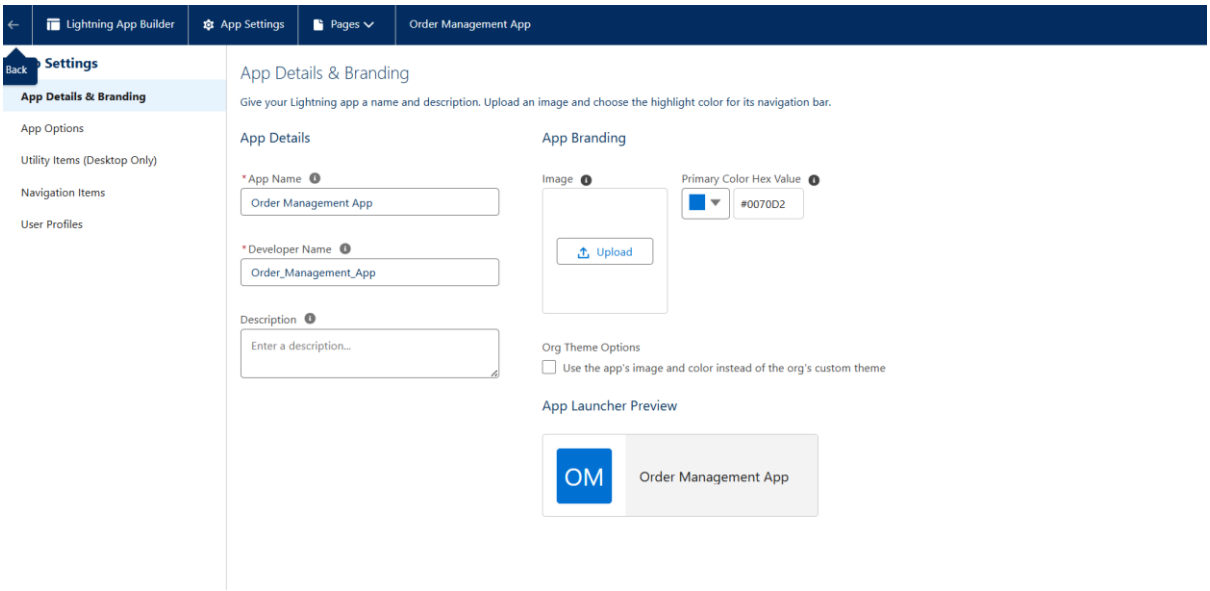
Test Classes

```
@IsTest
public class OrderHandlerTest {
    @IsTest
    static void testRestrictStage(){
        Order__c ord = new Order__c(Name='Test Order', Stage__c='Created');
        insert ord;
        ord.Stage__c = 'Delivered';
        Test.startTest();
        try {
            update ord;
        } catch(DmlException e){
            System.assert(e.getMessage().contains('Stage cannot be changed'));
        }
        Test.stopTest();
    }
}
```

Phase 6: User Interface Development

Lightning App Builder

We want to create a dedicated Order Management App for users (Sales Reps, Sales Manager, Support Team) so they can easily access Orders, Products, Customers, and Issues in one place without navigating multiple areas.

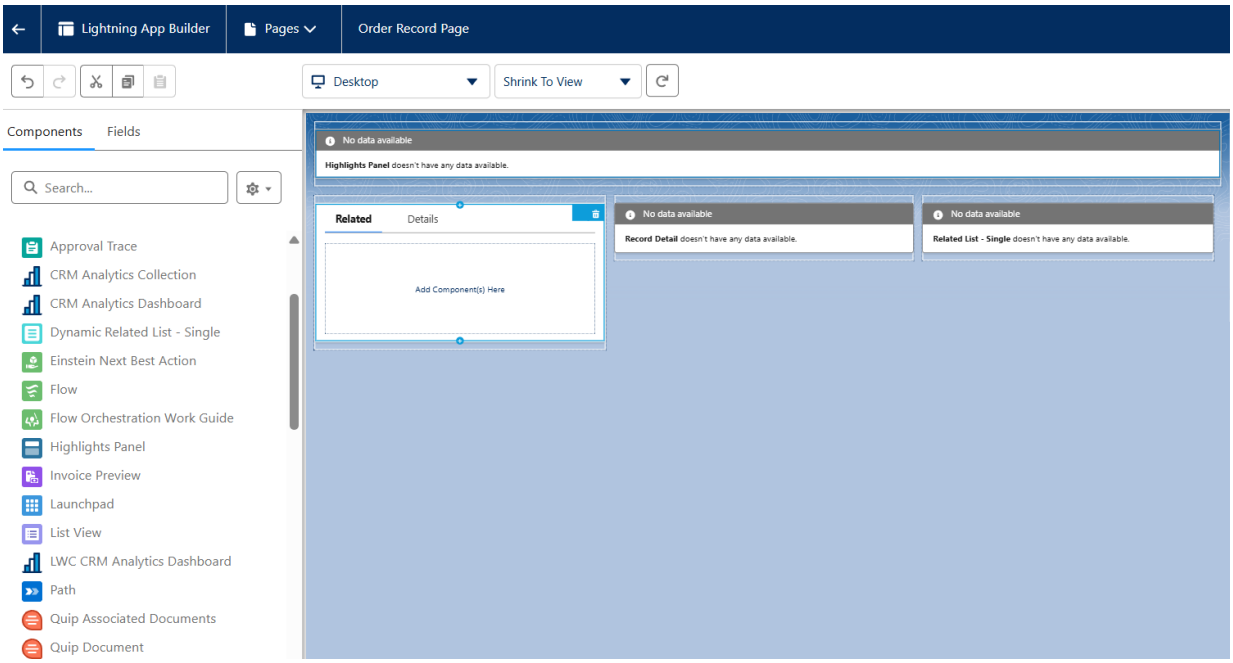


Record Pages

When a Sales Rep or Manager opens an Order, they should see:

- Key fields (Order Status, Amount, Customer).
- Related lists (Products, Order Issues).
- Quick actions (Cancel, Approve, etc.).
- A clean, customized page instead of Salesforce's default layout.

We'll use Lightning App Builder to create a custom Order Record Page.

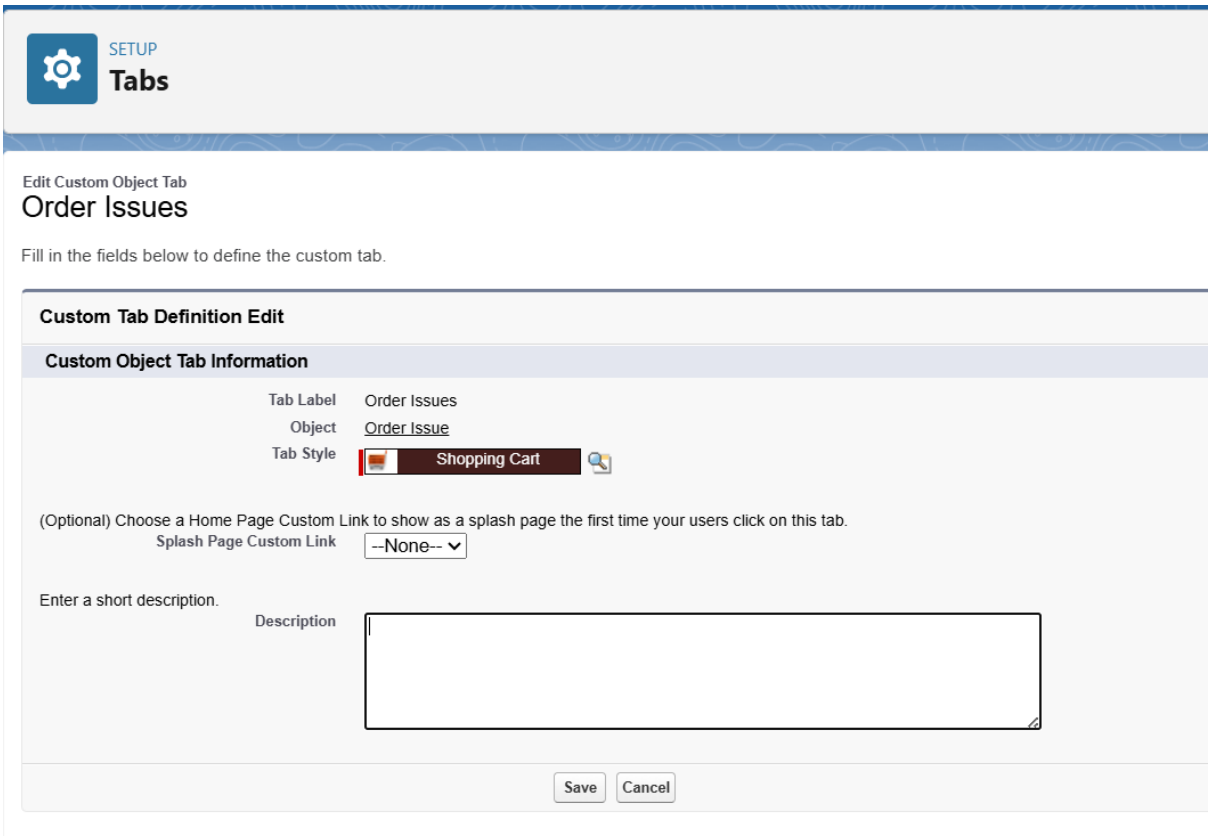


Tabs in Salesforce

Tabs are what make your objects or pages visible in the Salesforce App Navigation Bar (so users can easily access Orders, Order Issues, etc.).

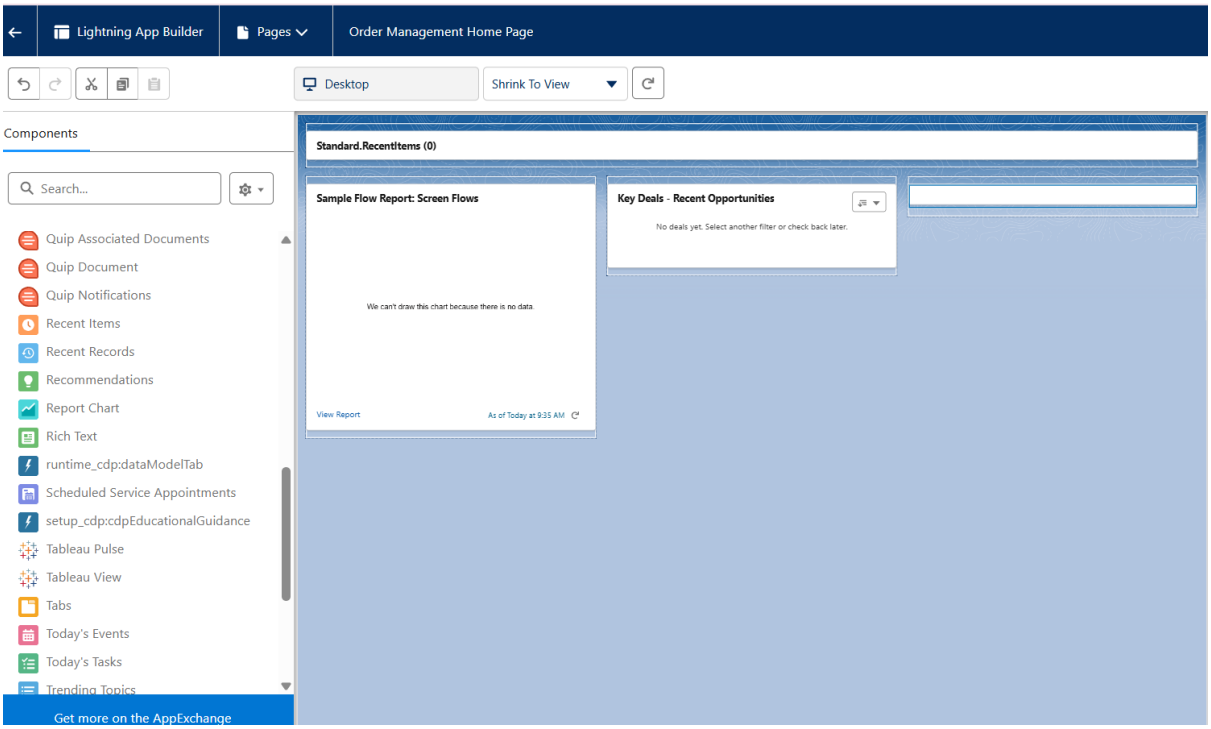
There are 3 main types of tabs:

1. Custom Object Tabs → for objects like Order__c, OrderIssue__c.
2. Lightning Page Tabs → for specific record/home pages.
3. Web Tabs / Visualforce Tabs → for external apps or VF pages (less used in your case).



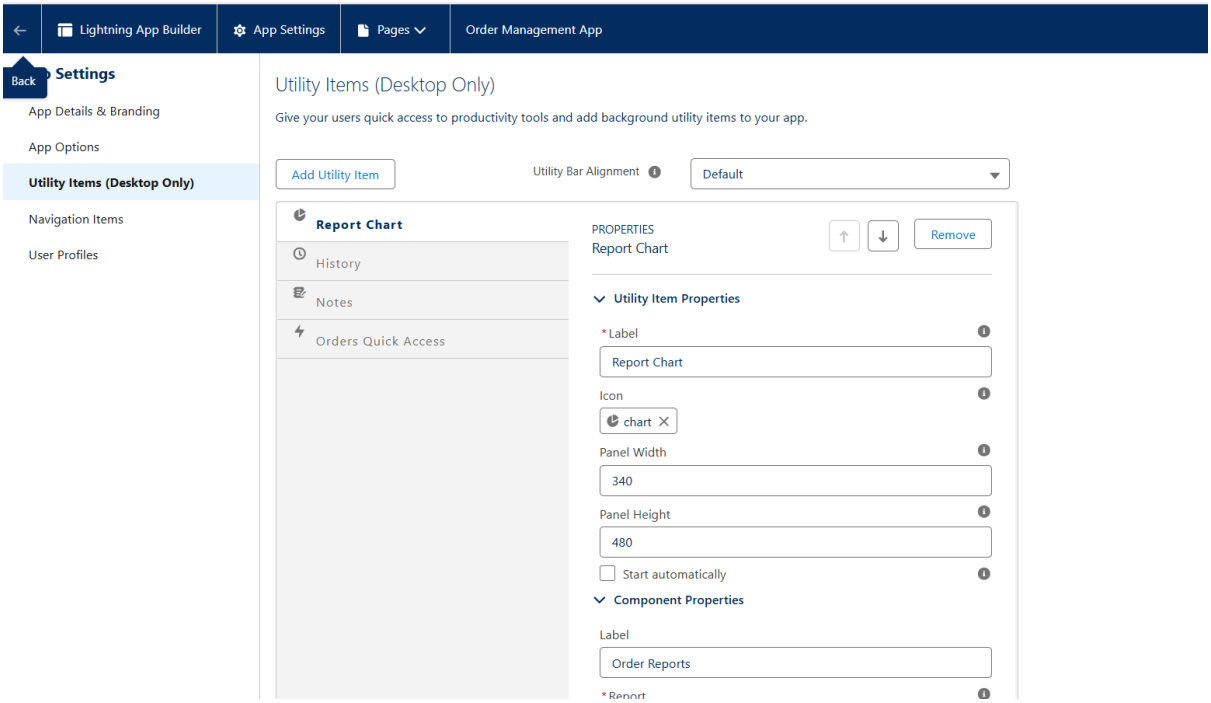
Home Page Layouts

The Home Page in Salesforce Lightning is the first screen users see when they log in. Customizing it helps show Order-related insights, shortcuts, and reports so that users can act quickly.



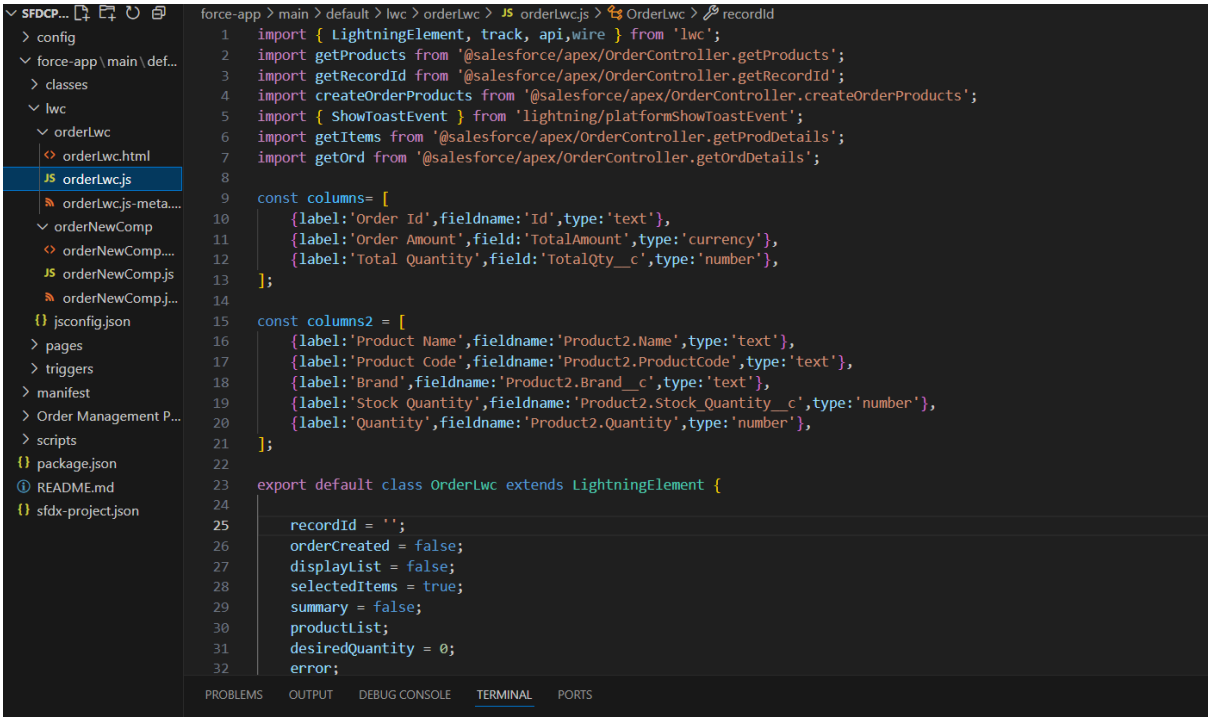
Utility Bar

The Utility Bar in Salesforce Lightning is a fixed footer at the bottom of the screen. It allows quick access to important tools like Notes, Recent Items, Open Orders, Chat, or Custom Components (like Order Calculator) from anywhere in the app.



LWC (Lightning Web Components)

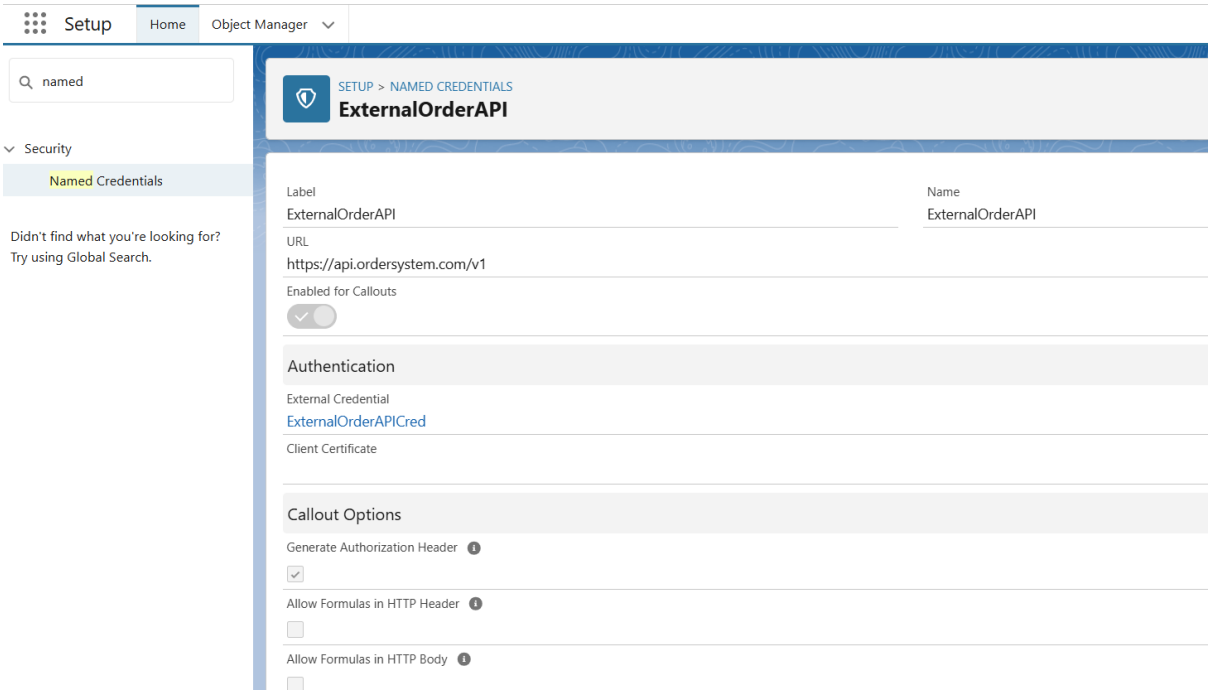
Navigation Service



Phase 7: Integration & External Access

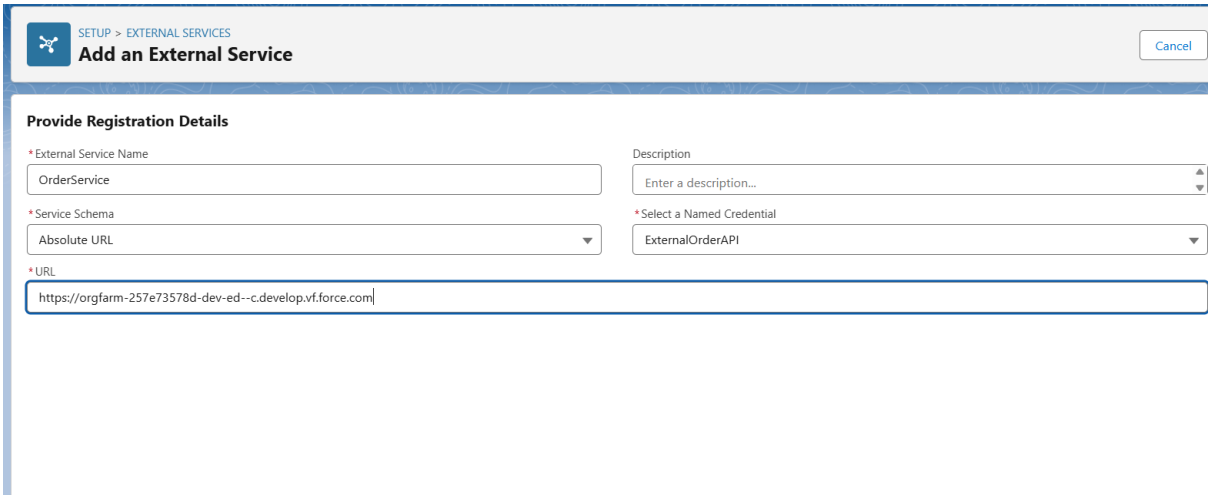
Named Credentials

Purpose: Securely store authentication info for external systems.



External Services

Purpose:
Connect Salesforce to external APIs (like your ExternalOrderAPI) and use them in Flows, LWC, or Apex. Salesforce reads an OpenAPI schema and automatically generates actions you can use.



Web Services (REST / SOAP)

Salesforce can both:

1. **Consume external APIs** (call outside services from Salesforce).
2. **Expose Salesforce as an API** (let outside systems call Salesforce).

A. Consuming REST APIs (Call external services)

This is the most common case (like connecting to an Order System, Payment Gateway, etc.).

OrderAPIService.cls:

```
public with sharing class OrderAPIService {
    @AuraEnabled(cacheable=true)
    public static String getOrderStatus(String orderId) {
        Http http = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('callout:ExternalOrderAPI/orders/' + orderId);
        req.setMethod('GET');

        HttpResponse res = http.send(req);

        if (res.getStatusCode() == 200) {
            return res.getBody(); // JSON response from external API
        } else {
            return 'Error: ' + res.getStatus();
        }
    }
}
```

Callouts

What are Callouts?

- A callout is when Salesforce makes an HTTP request to an external system (REST or SOAP).
- Example: When a new Order is created in Salesforce, automatically send a callout to an external Order Management System.

OrderCallout.cls:

```
public with sharing class OrderCallout {
    @future(callout=true)
```

```
    public static void sendOrderToExternal(Id orderId) {
        Order__c ord = [SELECT Id, Name, Status__c FROM Order__c WHERE Id =
:orderId];

        Http http = new Http();
        HttpRequest req = new HttpRequest();

        // Using Named Credential
        req.setEndpoint('callout:ExternalOrderAPI/orders');
        req.setMethod('POST');
        req.setHeader('Content-Type', 'application/json');

        // Convert to JSON
        Map<String, Object> body = new Map<String, Object>();
        body.put('orderId', ord.Id);
        body.put('status', ord.Status__c);
        req.setBody(JSON.serialize(body));

        HttpResponse res = http.send(req);
        System.debug('Response: ' + res.getBody());
    }
}
```

orderLwc.js-meta.xml | UserTriggerSequenceHandler.cls | OrderCallout.cls

force-app > main > default > classes > OrderCallout.cls

```
public with sharing class OrderCallout {
    @future(callout=true)
    public static void sendOrderToExternal(Id orderId) {
        Order__c ord = [SELECT Id, Name, Status__c FROM Order__c WHERE Id = :orderId];

        Http http = new Http();
        HttpRequest req = new HttpRequest();

        // Using Named Credential
        req.setEndpoint('callout:ExternalOrderAPI/orders');
        req.setMethod('POST');
        req.setHeader('Content-Type', 'application/json');

        // Convert to JSON
        Map<String, Object> body = new Map<String, Object>();
        body.put('orderId', ord.Id);
        body.put('status', ord.Status__c);
        req.setBody(JSON.serialize(body));

        HttpResponse res = http.send(req);
        System.debug('Response: ' + res.getBody());
    }
}
```

Platform Events

When an **Order is created in Salesforce**, publish an event → external Order System consumes it.

- When the external system updates delivery → it publishes an event → Salesforce consumes it.

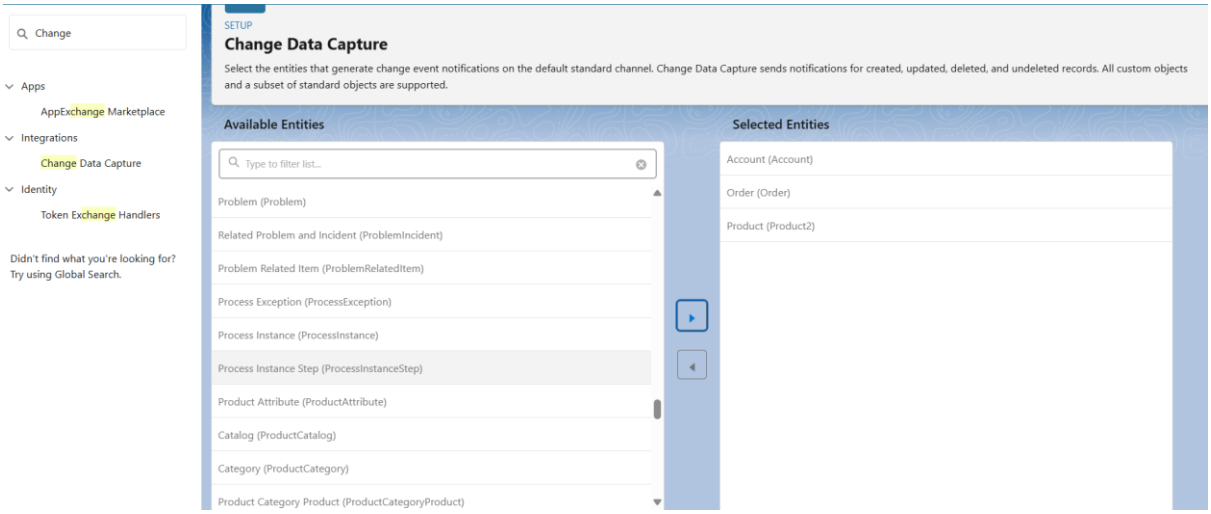
Eg:

```
trigger OrderTrigger on Order__c (after insert) {
    for (Order__c ord : Trigger.new) {
        Order_Event__e event = new Order_Event__e(
            OrderId__c = ord.Id,
            Status__c  = ord.Status__c
        );
        Database.SaveResult result = EventBus.publish(event);
        System.debug('Event Published: ' + result.isSuccess());
    }
}
```

```
trigger OrderTrigger on Order__c (after insert) {
    for (Order__c ord : Trigger.new) {
        Order_Event__e event = new Order_Event__e(
            OrderId__c = ord.Id,
            Status__c  = ord.Status__c
        );
        Database.SaveResult result = EventBus.publish(event);
        System.debug('Event Published: ' + result.isSuccess());
    }
}
// This trigger publishes a platform event whenever a new Order__c record is created.
```

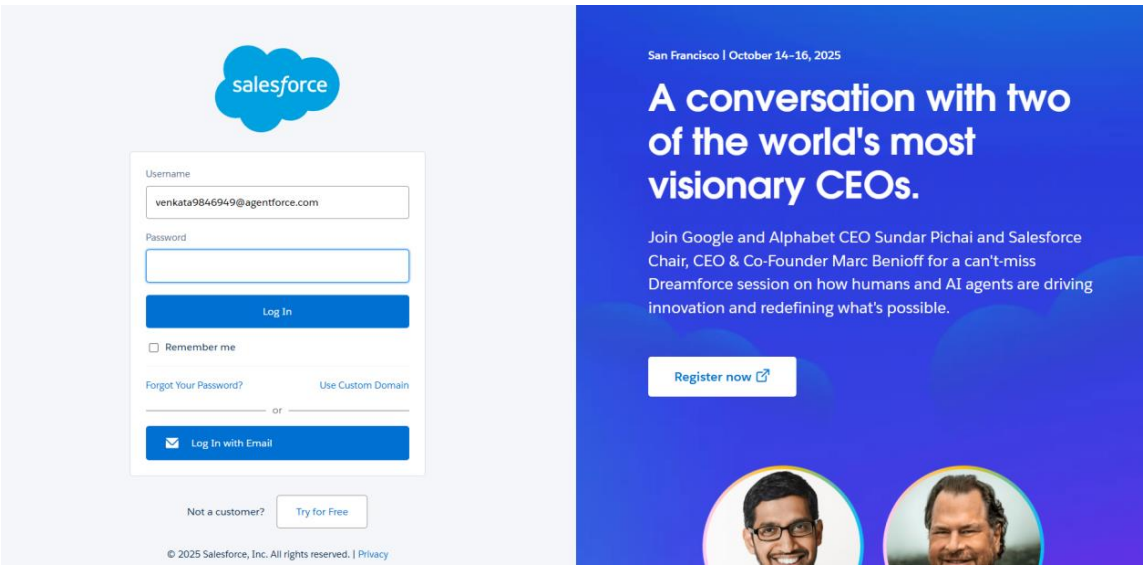
Change Data Capture (CDC)

- If an Order__c record is updated in Salesforce → a CDC event fires → external Order System gets notified instantly.
- If the external system updates Order__c → Salesforce users can subscribe to CDC to react (e.g., Flow updates UI).



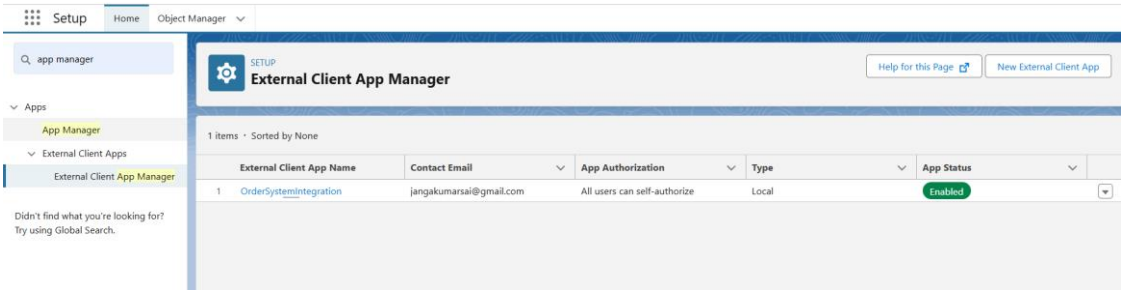
Salesforce Connect

- Provides real-time view of external data without storing in Salesforce.



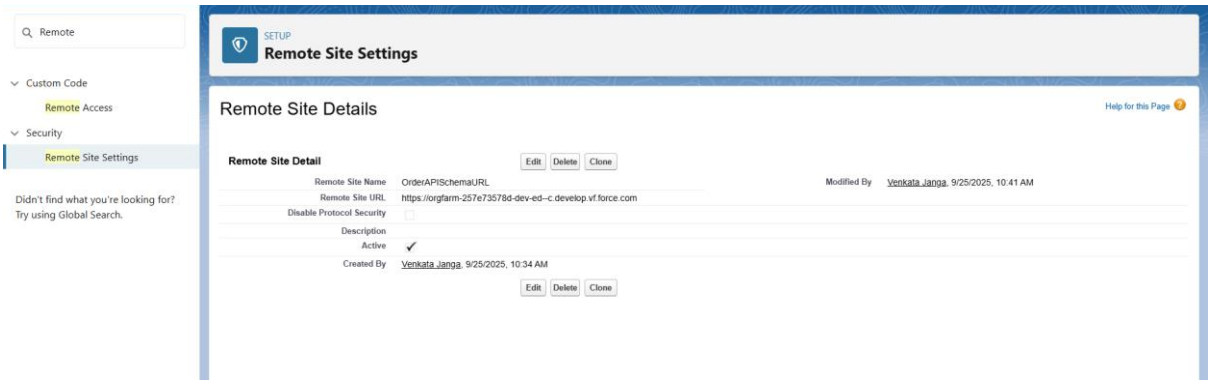
OAuth & Authentication

When your Salesforce org connects with an external system (like your Order System API), both systems need to trust each other. Salesforce uses OAuth 2.0 as the standard authentication framework.



Remote Site Settings

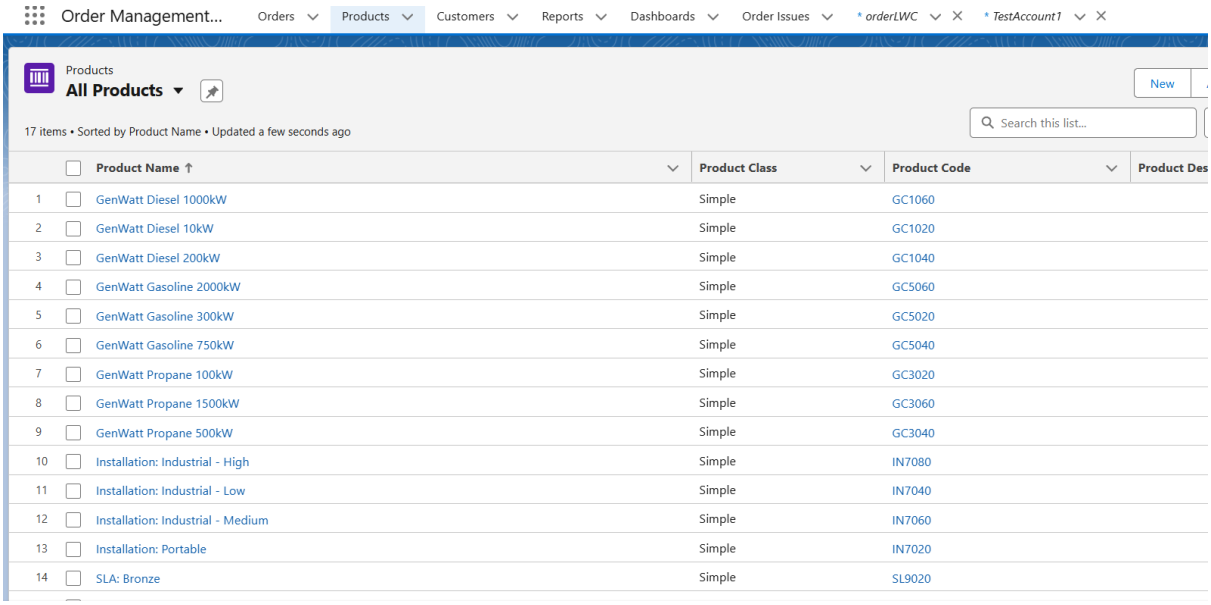
- Required to allow outbound callouts to untrusted domains.
- Add URL as a Remote Site Setting before making callouts.



Phase 8: Data Management & Deployment

Data Import Wizard

Added some products using Data Import Wizard



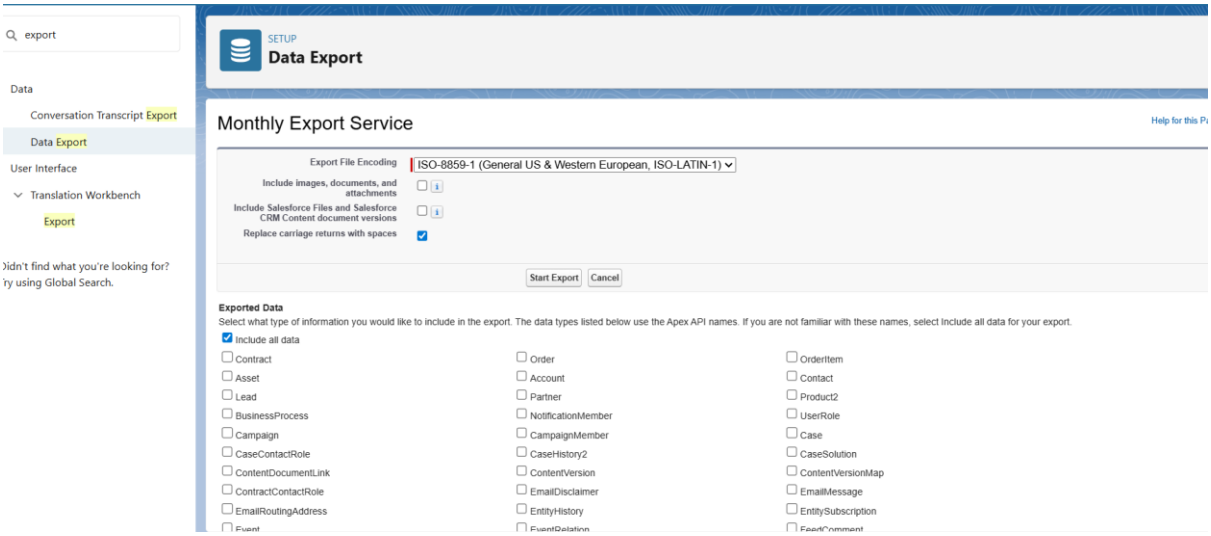
	Product Name	Product Class	Product Code	Product Des
1	GenWatt Diesel 1000kW	Simple	GC1060	
2	GenWatt Diesel 10kW	Simple	GC1020	
3	GenWatt Diesel 200kW	Simple	GC1040	
4	GenWatt Gasoline 2000kW	Simple	GC5060	
5	GenWatt Gasoline 300kW	Simple	GC5020	
6	GenWatt Gasoline 750kW	Simple	GC5040	
7	GenWatt Propane 100kW	Simple	GC3020	
8	GenWatt Propane 1500kW	Simple	GC3060	
9	GenWatt Propane 500kW	Simple	GC3040	
10	Installation: Industrial - High	Simple	IN7080	
11	Installation: Industrial - Low	Simple	IN7040	
12	Installation: Industrial - Medium	Simple	IN7060	
13	Installation: Portable	Simple	IN7020	
14	SLA: Bronze	Simple	SL9020	

Data Loader

- Data Loader for bulk upload, but not required since records were limited.

Data Export & Backup

- Verified Data Export option in Setup for backup.



VS Code & SFDX

- Used VS Code with SFDX CLI for writing and deploying Apex Classes, Triggers, and LWC.
- Successfully pushed/pulled metadata between local project and Salesforce Org

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(c) Microsoft Corporation. All rights reserved.

C:\Users\LENOVO\Downloads\SFDCProject>sf project deploy start -o MyNewOrg --source-dir force-app/main/default/classes
```

Phase 9: Reporting, Dashboards & Security Review

Reports (Tabular, Summary, Matrix, Joined)

- Created different report formats (Grouped using Amount, Payment Date, Status and added columns Order ID, Status, Order Number, Account Name, Account Owner, Order Start Date, OrderType, Order Amount, Created Date)

Report: Orders
New Orders Report

Enable Field Editing

Total Records
7

Order ID	Status	Order Number	Account Name	Account Owner	Order Start Date	Order Type	Order Amount	Created Date
801gL00000Omat2 (1)	Draft (1)	00000100	Test Account 1	Kumarsai Janga	9/1/2025	-	$0.00	9/22/2025
	Subtotal							
Subtotal								
801gL00000OpeVW (1)	Draft (1)	00000101	Test Account 1	Kumarsai Janga	9/8/2025	-	$0.00	9/22/2025
	Subtotal							
Subtotal								
801gL00000P4nyF (1)	Draft (1)	00000102	Test Account 1	Kumarsai Janga	9/23/2025	-	$0.00	9/24/2025
	Subtotal							
Subtotal								
801gL00000P58Pz (1)	Draft (1)	00000104	Test Account 1	Kumarsai Janga	9/1/2025	-	$0.00	9/24/2025
	Subtotal							
Subtotal								
801gL00000P5DeA (1)	Draft (1)	00000103	Test Account 1	Kumarsai Janga	9/15/2025	-	$0.00	9/24/2025

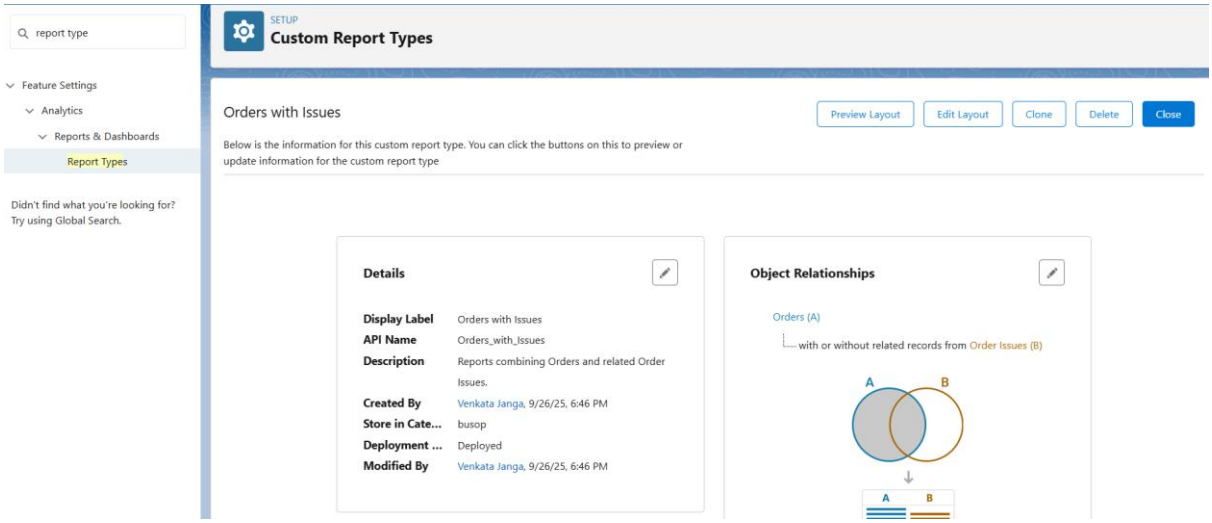
Row Counts Detail Rows Subtotals Grand Total

Report Types

Purpose:
Defines which objects and fields are available for reporting.

Explanation:

- Standard report types are prebuilt for common objects.
- Custom report types allow combining custom objects or multiple objects.
- Example: “Orders with related Order Issues”

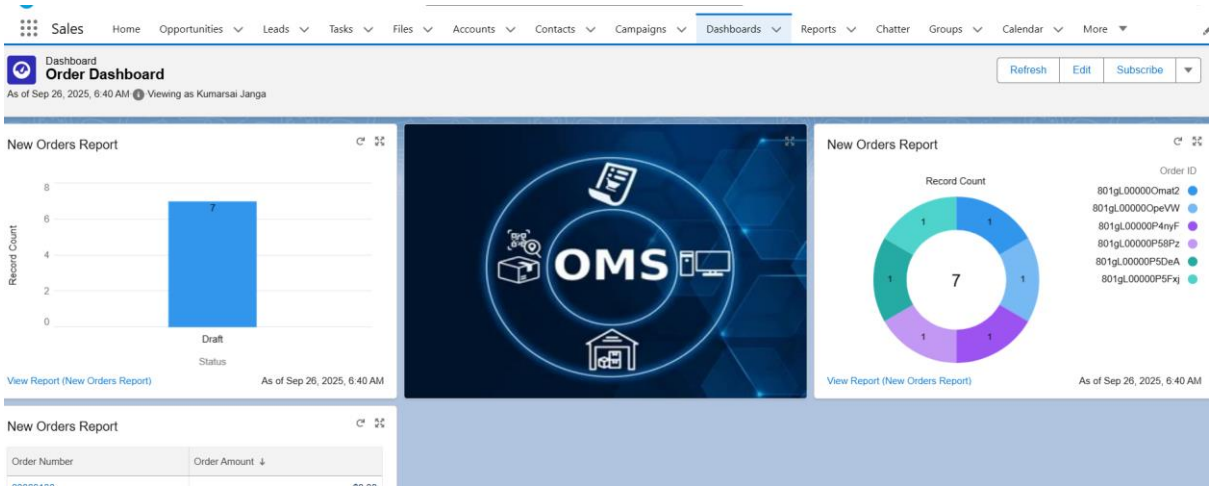


Dashboards

Purpose:
Dashboards visually display data from one or more reports in charts, tables, or graphs for quick analysis and monitoring of key metrics.

Explanation:

- Dashboards combine multiple reports in one page.
- Components can be bar charts, pie charts, line charts, gauges, tables, etc.
- Useful for managers to track KPIs like total orders, order status distribution, or top customers.



Profiles

profiles

Users

Profiles

Didn't find what you're looking for?
Try using Global Search.

SETUP
Profiles

Profile
Business profile

Help for this Page

Users with this profile have the permissions and page layouts listed below. Administrators can change a user's profile by editing that user's personal information.

If your organization uses Record Types, use the Edit links in the Record Type Settings section below to make one or more record types available to users with this profile.

Login IP Ranges [0] | Enabled Apex Class Access [2] | Enabled Visualforce Page Access [0] | Enabled External Data Source Access [0] | Enabled Named Credential Access [0] | Enabled External Credential Principal Access [0] | Enabled Custom Metadata Type Access [0] | Enabled Custom Setting Definitions Access [0] | Enabled Flow Access [0] | Enabled Service Presence Status Access [0] | Enabled Custom Permissions [0]

Profile Detail Edit | Clone | Delete | View Users

Name	Business profile		
User License	Salesforce	Custom Profile	✓
Description			
Created By	Venkata Janga, 9/23/2025, 9:18 AM	Modified By	Venkata Janga, 9/25/2025, 6:00 AM

Page Layouts

Standard Object Layouts

Global	Global Layout [View Assignment]	Location Group	Location Group Layout [View Assignment]
Email Application	Not Assigned [View Assignment]	Location Group Assignment	Location Group Assignment Layout [View Assignment]
Home Page Layout	Home Page Default [View Assignment]	Macro	Macro Layout [View Assignment]
Account	Account Layout [View Assignment]	Object Milestone	Object Milestone Layout [View Assignment]

profiles

Users

Profiles

Didn't find what you're looking for?
Try using Global Search.

SETUP
Profiles

Profile
Custom: Sales Profile

Help for this Page

Users with this profile have the permissions and page layouts listed below. Administrators can change a user's profile by editing that user's personal information.

If your organization uses Record Types, use the Edit links in the Record Type Settings section below to make one or more record types available to users with this profile.

Login IP Ranges [0] | Enabled Apex Class Access [2] | Enabled Visualforce Page Access [0] | Enabled External Data Source Access [0] | Enabled Named Credential Access [0] | Enabled External Credential Principal Access [0] | Enabled Custom Metadata Type Access [0] | Enabled Custom Setting Definitions Access [0] | Enabled Flow Access [0] | Enabled Service Presence Status Access [0] | Enabled Custom Permissions [0]

Profile Detail Edit | Clone | Delete | View Users

Name	Custom: Sales Profile		
User License	Salesforce	Custom Profile	✓
Description			
Created By	OrgFarm EPIC, 9/18/2025, 5:42 PM	Modified By	Venkata Janga, 9/25/2025, 6:00 AM

Page Layouts

Standard Object Layouts

Global	Global Layout [View Assignment]	Location Group	Location Group Layout [View Assignment]
Email Application	Not Assigned [View Assignment]	Location Group Assignment	Location Group Assignment Layout [View Assignment]

Roles

Roles

Creating the Role Hierarchy

You can build on the existing role hierarchy shown on this page. To insert a new role, click **Add Role**.

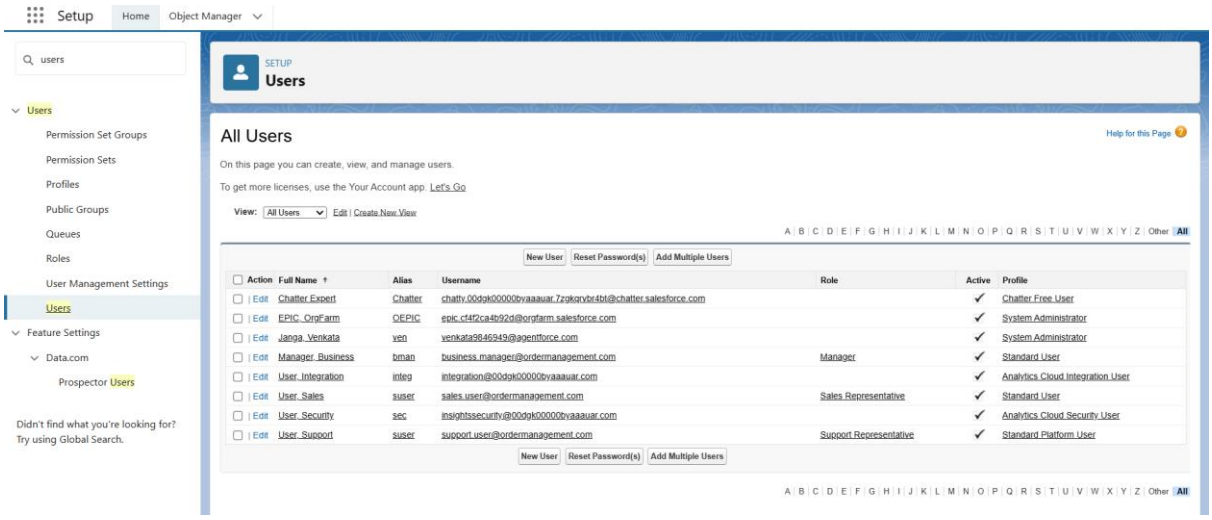
Your Organization's Role Hierarchy

Collapse All Expand All

- **OrderManagement**
  - Add Role
  - **CEO** Edit | Del | Assign
    - Add Role
    - **CFO** Edit | Del | Assign
      - Add Role
    - **COO** Edit | Del | Assign
      - Add Role
    - **Manager** Edit | Del | Assign
      - Add Role
    - **Support Representative** Edit | Del | Assign
      - Add Role
    - **SVP, Customer Service & Support** Edit | Del | Assign
      - Add Role
    - **SVP, Human Resources** Edit | Del | Assign
      - Add Role
    - **SVP, Sales & Marketing** Edit | Del | Assign
      - Add Role

Users

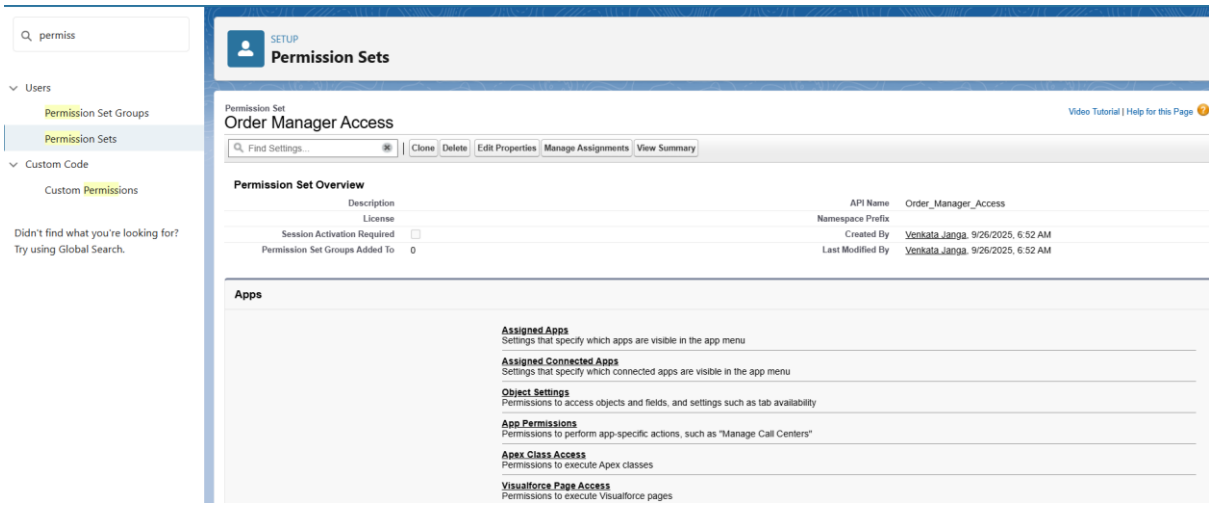
Created users (Manager, Sales, Support and Sales users).



Permission Sets

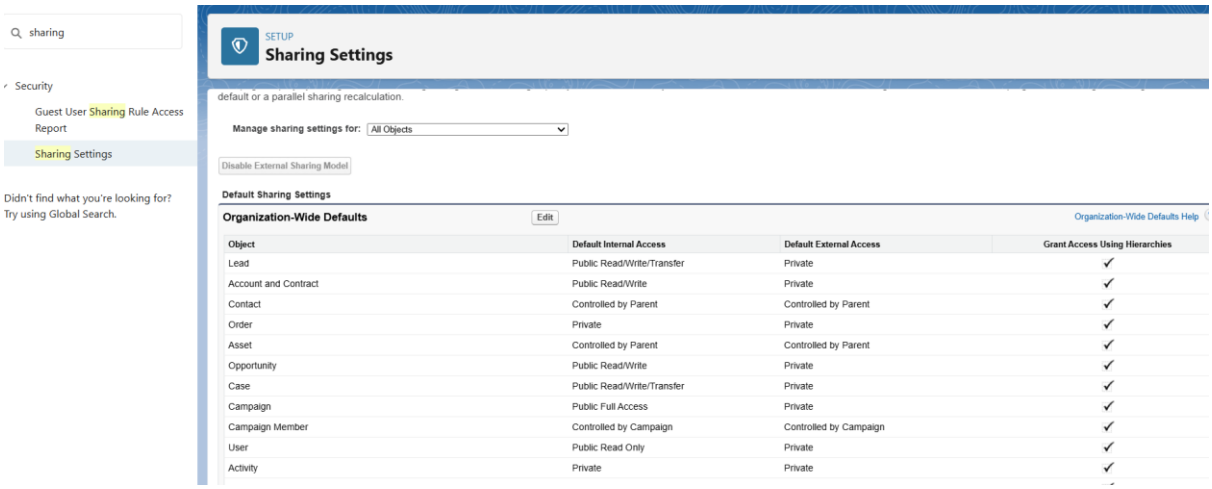
Purpose:

Permission Sets allow you to grant additional permissions to users without changing their profile. This is useful when some users need extra access temporarily or for specific objects/fields.



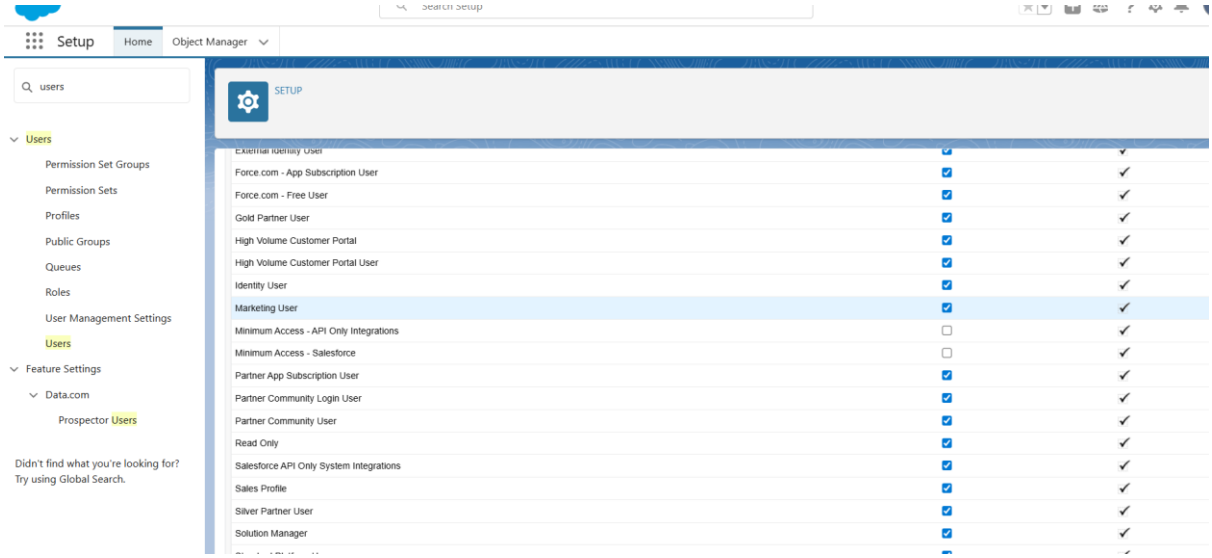
Organization-Wide Defaults (OWD)

Purpose:
OWD defines the baseline level of access for all records of an object. It controls who can see records by default.



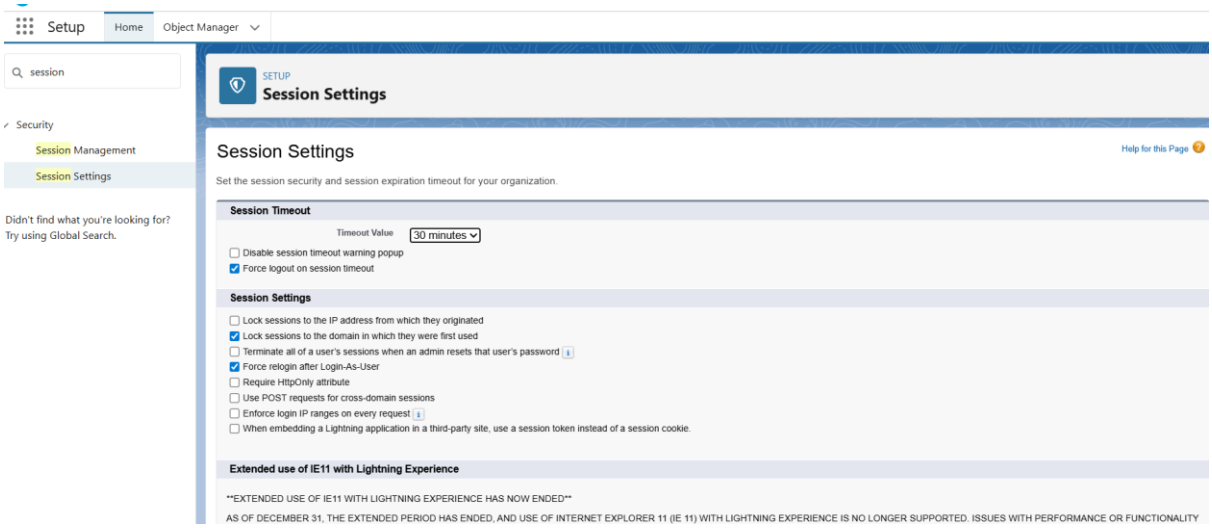
Field-Level Security (FLS)

Purpose:
Field-Level Security controls which users can view or edit specific fields on an object.



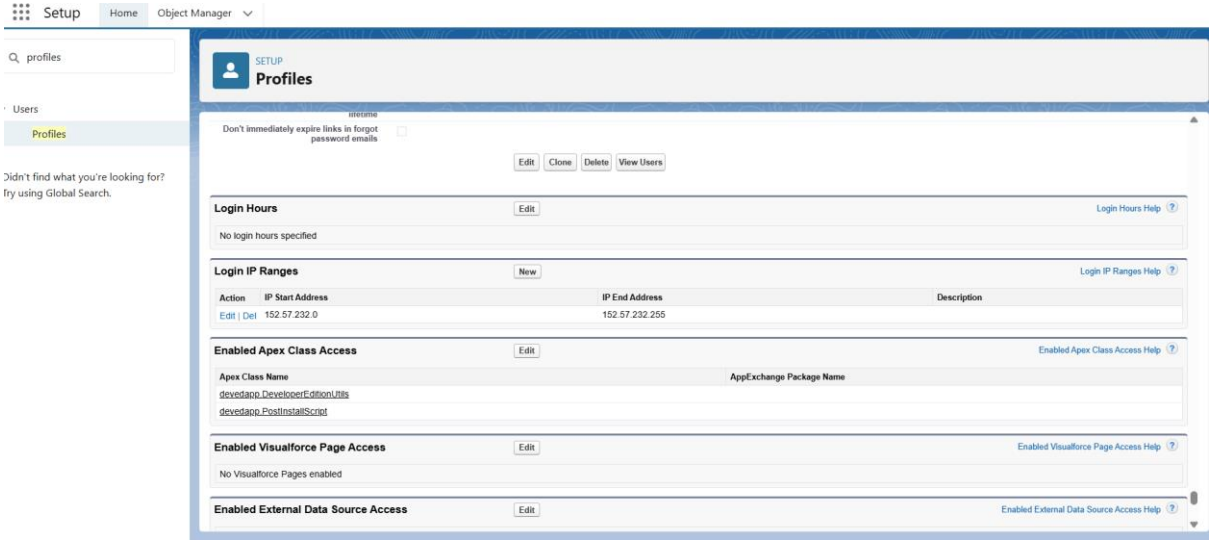
Session settings

Set session timeout for 30 minutes.



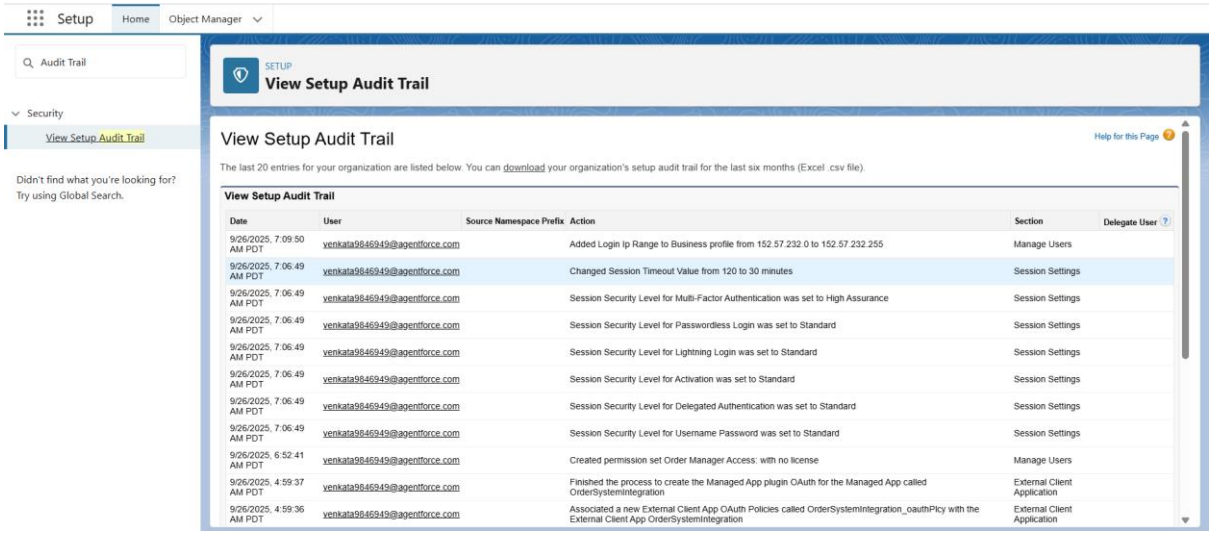
Login IP Ranges

- Configured login IP ranges for each profile.



Audit Trail

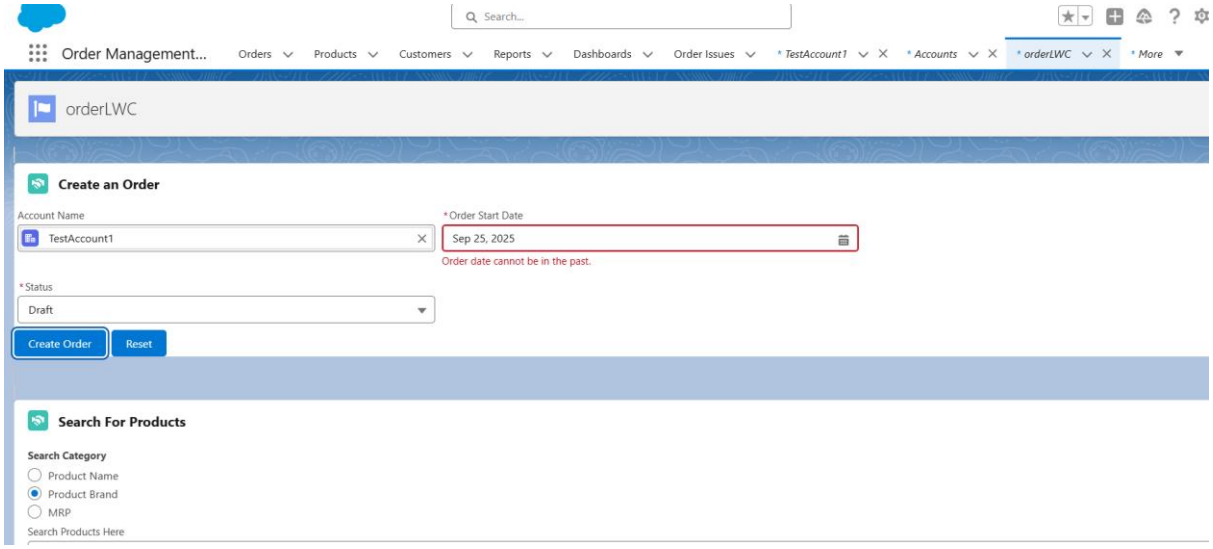
- Enabled Setup Audit Trail to track all admin-level changes.



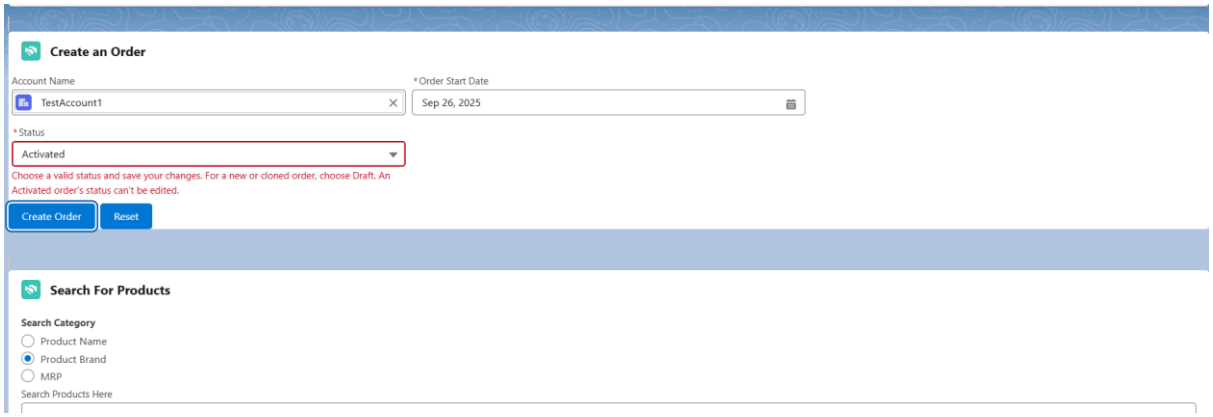
Date	User	Source Namespace Prefix	Action	Section	Delegate User
9/26/2025, 7:09:50 AM PDT	venkata9846949@agentforce.com		Added Login Ip Range to Business profile from 152.57.232.0 to 152.57.232.255	Manage Users	
9/26/2025, 7:06:49 AM PDT	venkata9846949@agentforce.com		Changed Session Timeout Value from 120 to 30 minutes	Session Settings	
9/26/2025, 7:06:49 AM PDT	venkata9846949@agentforce.com		Session Security Level for Multi-Factor Authentication was set to High Assurance	Session Settings	
9/26/2025, 7:06:49 AM PDT	venkata9846949@agentforce.com		Session Security Level for Passwordless Login was set to Standard	Session Settings	
9/26/2025, 7:06:49 AM PDT	venkata9846949@agentforce.com		Session Security Level for Lightning Login was set to Standard	Session Settings	
9/26/2025, 7:06:49 AM PDT	venkata9846949@agentforce.com		Session Security Level for Activation was set to Standard	Session Settings	
9/26/2025, 7:06:49 AM PDT	venkata9846949@agentforce.com		Session Security Level for Delegated Authentication was set to Standard	Session Settings	
9/26/2025, 7:06:49 AM PDT	venkata9846949@agentforce.com		Session Security Level for Username Password was set to Standard	Session Settings	
9/26/2025, 6:52:41 AM PDT	venkata9846949@agentforce.com		Created permission set Order Manager Access: with no license	Manage Users	
9/26/2025, 4:59:37 AM PDT	venkata9846949@agentforce.com		Finished the process to create the Managed App plugin OAuth for the Managed App called OrderSystemIntegration	External Client Application	
9/26/2025, 4:59:36 AM PDT	venkata9846949@agentforce.com		Associated a new External Client App OAuth Policies called OrderSystemIntegration_oauthPlcy with the External Client App OrderSystemIntegration	External Client Application	

Phase 10: Quality Assurance Test

An order Date cannot be in past if we try to keep past date It will throw an error



For a new Order Status must be Drat And later Owner Can change the status to Activated or cancelled



Choose What products we need to add to order

We can search products based on Product name, brand name or MRP

Search For Products

Search Category

- Product Name
- Product Brand
- MRP

Search Products Here

ph

Product Code	Product Name	Brand	MRP	In Stock	Action
null	Philips A2	Philips	null	12	Add Item
PA1	Philips A1	Philips	1000.00	28	Add Item
GC3040	GenWatt Propane 500kW	null	50000.00	null	Add Item
SL9080	SLA: Platinum	null	40000.00	null	Add Item
GC3020	GenWatt Propane 100kW	null	15000.00	null	Add Item
GC3060	GenWatt Propane 1500kW	null	120000.00	null	Add Item
IN7020	Installation: Portable	null	5000.00	null	Add Item

We can Add or Remove Selected Products for our Order

Selected Products List

Product Code	Product Name	Brand	Stock Quantity	Quantity	Discount(%)	Action
PA1	Philips A1	Philips	28	4	0	Remove

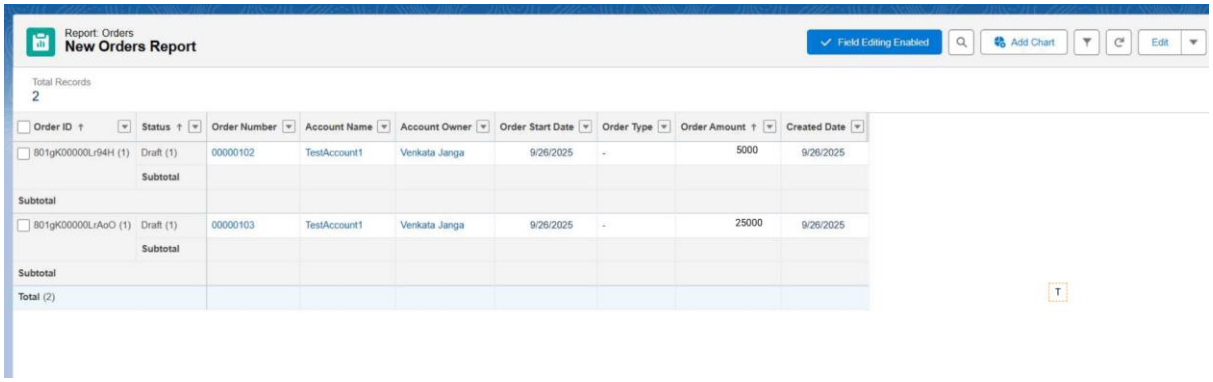
Confirm Order Cancel Order

Finally Order Details are Show as below(Order Summary)

Order Details:
Order ID: 8012w000000Cn5mAAC Order Amount : 11000 Total Order Quantity : 11

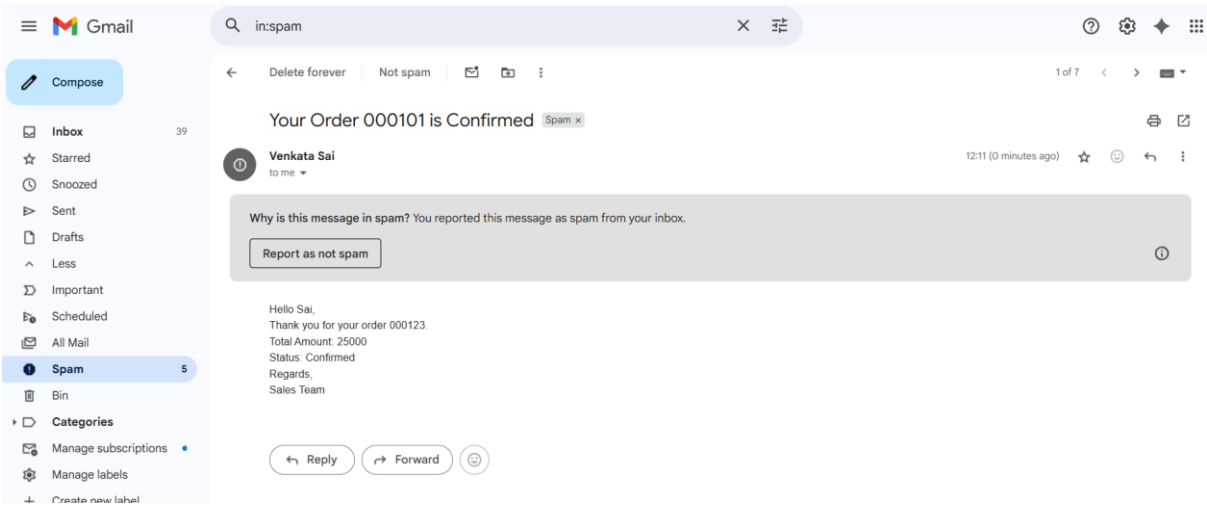
Product Details:
Product Name : Philips A1 Product Brand : Philips Stock Quantity : 944 Quantity : 11

Once orders are created we can view them in reports as follow

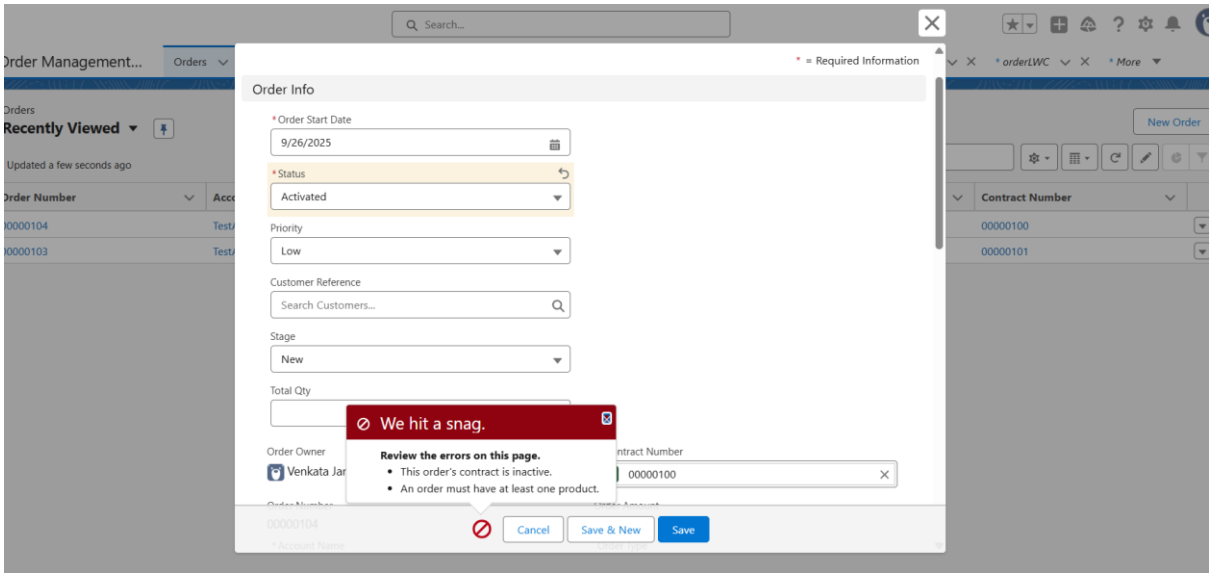


Updates in dashboard

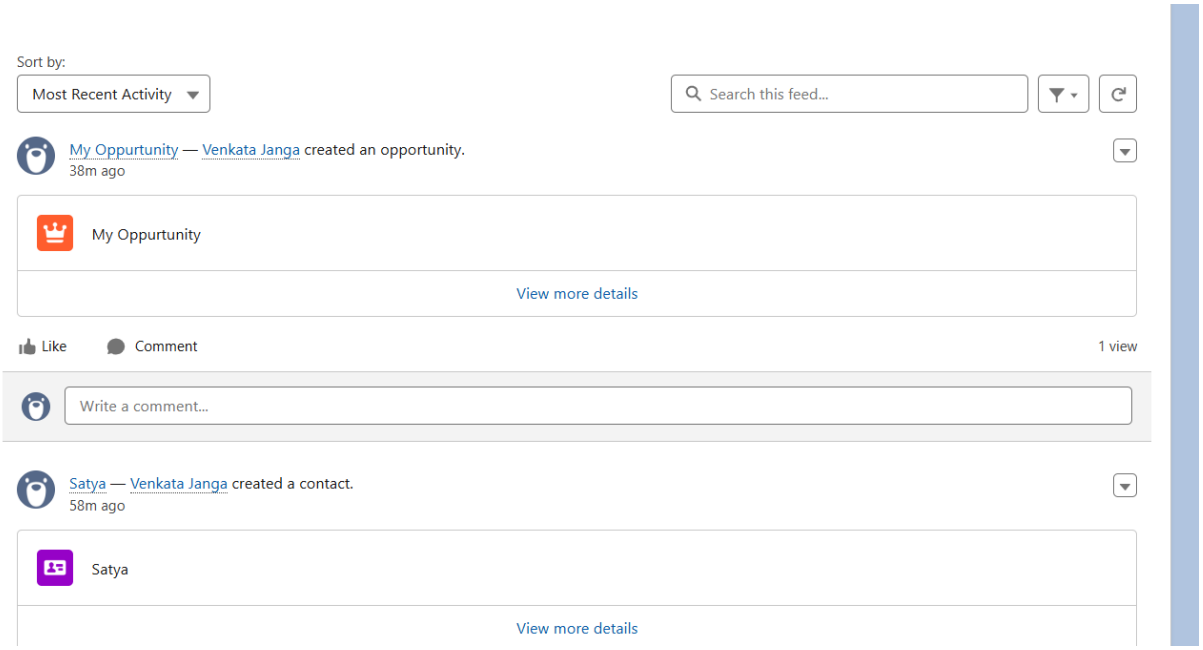
An Order Confirmation is also sent to email When an order is created



We cannot make an order Active unless it has a product selected



All Activities are visible owner or business manager



Final Conclusion

The **Order Management CRM Project** implemented in Salesforce successfully automates and streamlines the process of managing customer orders. It enables users to efficiently create, track, and update orders while maintaining accurate customer and product information. Key outcomes of the project include:

- **Order Creation & Management:** Users can create orders with associated products, quantities, and customer details. The system ensures all mandatory fields are validated and stored correctly.
- **Automation & Notifications:** Workflow rules and email templates automatically notify customers when their orders are confirmed, ensuring timely communication and reducing manual effort.
- **User Interface & Accessibility:** The Lightning App Builder, Record Pages, and utility components provide a clear, user-friendly interface for managing orders.
- **Reporting & Tracking:** Custom fields and record layouts allow users to monitor order status, total amounts, and other key metrics for informed business decisions.
- **Integration & Scalability:** The project structure allows for future integration with external systems and can scale with organizational growth.

The project demonstrates how Salesforce can be leveraged to automate business processes, improve customer communication, and maintain accurate order records. It provides a practical, real-world application of CRM concepts, enhancing both operational efficiency and user experience.