Online Food Delivery CRM Project

Problem Statement:

The food delivery industry handles thousands of daily orders from websites, apps, and partner channels. Without a centralized CRM, restaurants struggle to manage orders, assign deliveries, track status, and handle customer complaints. Delivery agents rely on manual updates, managers lack visibility, and executives don't get real-time insights.

A **Salesforce-based Online Food Delivery CRM** is needed to:

- Automate order capture and assignment
- Enable real-time delivery tracking
- Centralize customer feedback and complaints
- Help managers monitor operations
- Provide executives with performance dashboards

Phase 1: Problem Understanding & Industry Analysis

Requirement Gathering

- Talk to stakeholders (restaurant managers, delivery agents, executives, customers).
- **Example requirements:**
  - Track all restaurants and their menu items.
  - Allow managers to assign orders to delivery agents.
  - Prevent delivery conflicts (no double assignment).
  - Capture customer complaints/feedback.
  - Generate sales and delivery performance reports.

Stakeholder Analysis

- **Restaurant Managers** → Manage restaurants, menu items, orders, and customer complaints.
- **Delivery Agents** → Update delivery status, ensure timely order completion.
- **Executives / Management** → View performance dashboards, monitor KPIs.
- **System Administrator (CEO)** → Configure Salesforce org, oversee security and data integrity.

Business Process Mapping

- **Order Placement** → Customer places order → System captures order → Assigned to Restaurant Manager.
- **Delivery Assignment** → Restaurant Manager assigns order to Delivery Agent.
- **Delivery Update** → Delivery Agent updates status (Assigned → Out for Delivery → Delivered).
- **Feedback Handling** → Complaints/feedback logged and monitored by Manager.
- **Reporting** → Executives analyze performance metrics via dashboards.

Industry-Specific Use Case Analysis

- Real-time order tracking similar to Swiggy/Zomato.
- Menu & restaurant management within CRM.
- Delivery agent assignment and status updates.
- Customer complaints handling for service improvement.
- Performance dashboards for management review.

AppExchange Exploration

- Explore AppExchange packages:
- Food Delivery Management Apps for reference.
- Maps & Route Optimization apps for delivery tracking.
- SMS/Email Notification apps for customer communication.

.....

Phase 2: Org Setup & Configuration

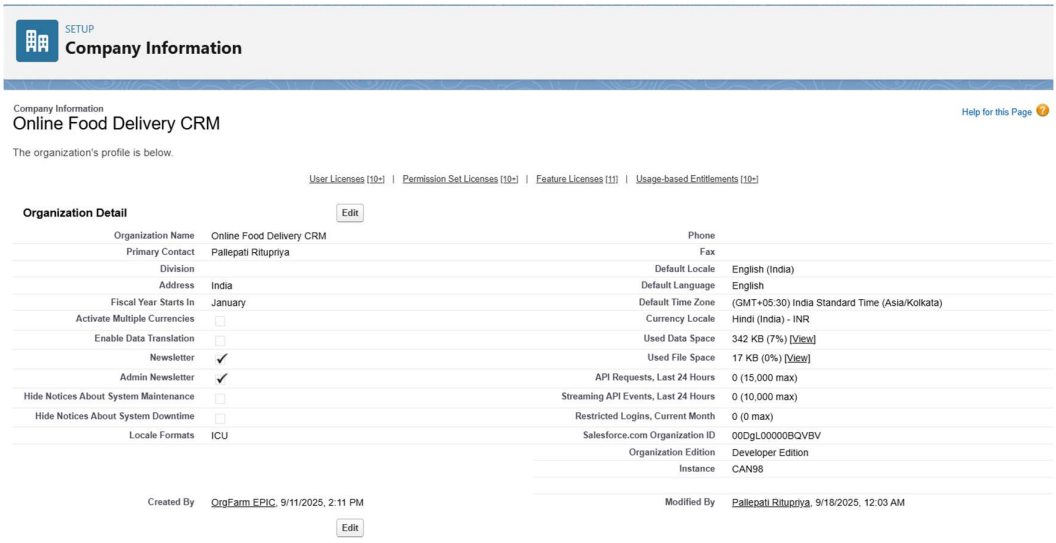
Salesforce Editions

- Using Salesforce Developer Edition Org (free, permanent).
- Supports custom objects, flows, Apex, and dashboards.

Company Profile Setup

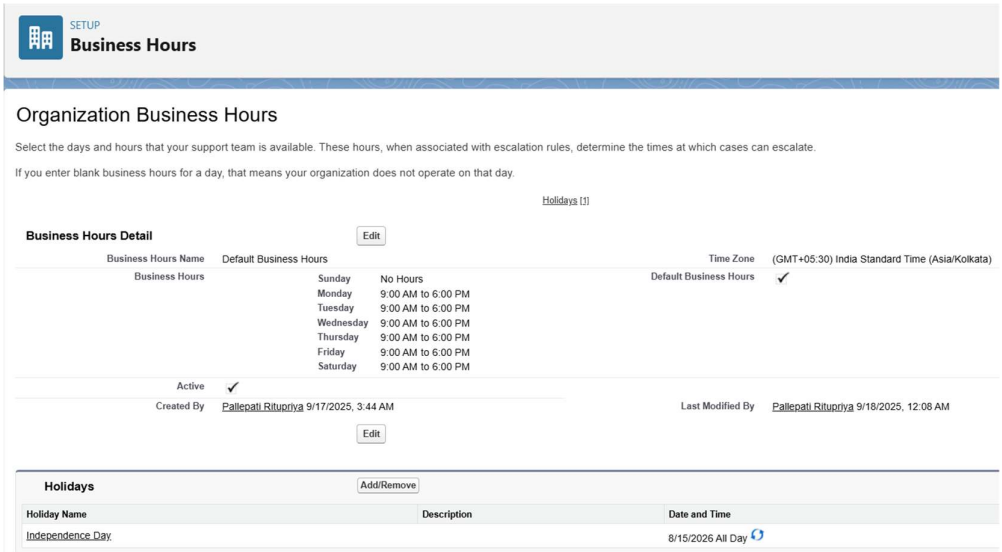
- Setup → Company Information → Edit.
- **Company Name:** Online Food Delivery CRM

- **Default Locale:** English (India)
- **Time Zone:** Asia/Kolkata (IST)
- **Currency:** INR
- Saved changes for org-wide consistency.



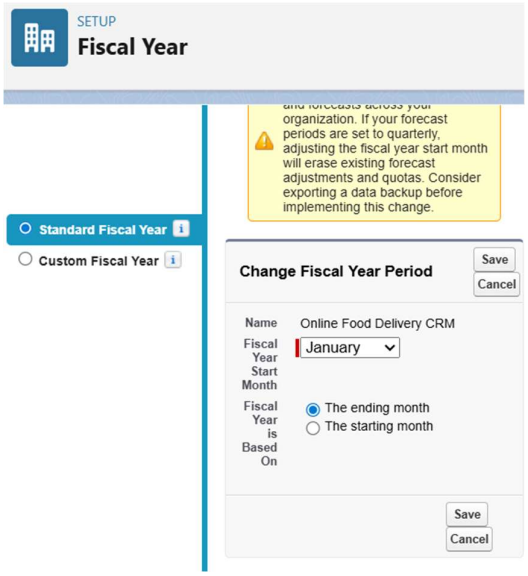
Business Hours & Holidays

- Setup → Business Hours → New Business Hours.
- **Working Hours:** 9:00 AM – 6:00 PM (Mon–Sat).
- Setup → Holidays → Add national holidays (e.g., 15th August – Independence Day).
- Ensures workflows/approvals don't trigger during holidays.



Fiscal Year Settings

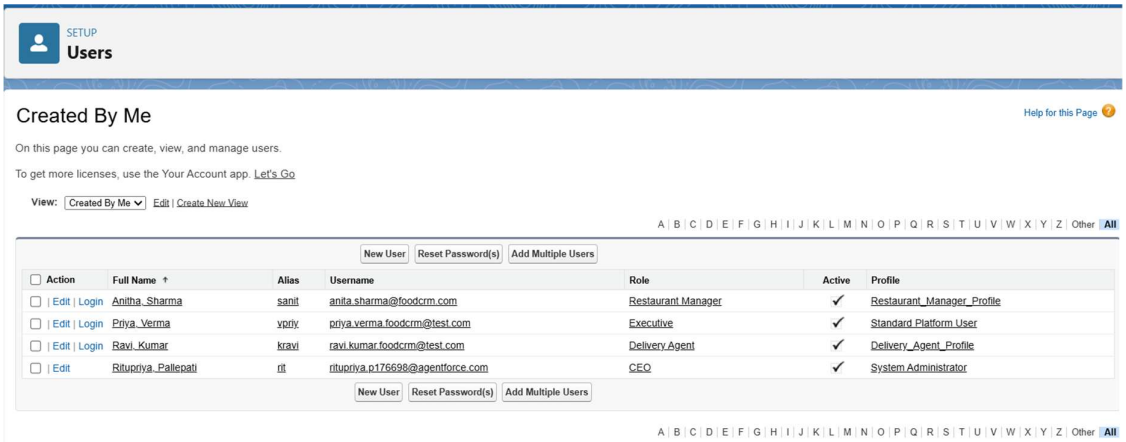
- Setup → Fiscal Year → Standard (Jan–Dec).
- Useful for revenue and sales reporting.



User Setup & Licenses

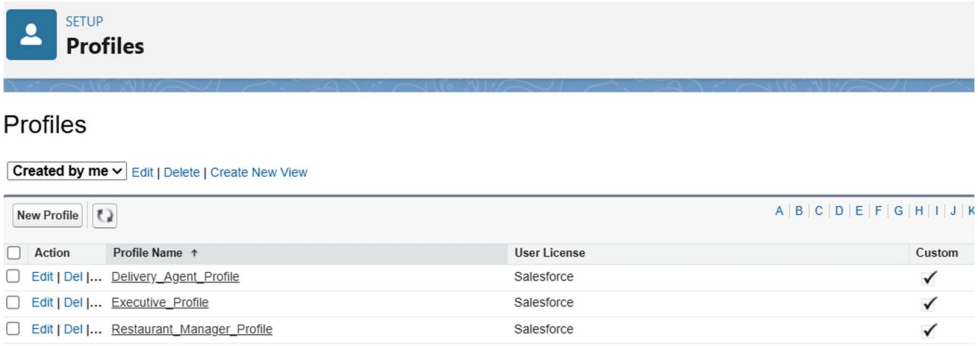
Created project-specific users:

- Me (Pallepati Ritupriya) = System Administrator (CEO) → Full access
- Anita Sharma (Restaurant Manager) → Salesforce License
- Ravi Kumar (Delivery Agent) → Salesforce License
- Priya Verma (Executive/Management) → Salesforce Platform License



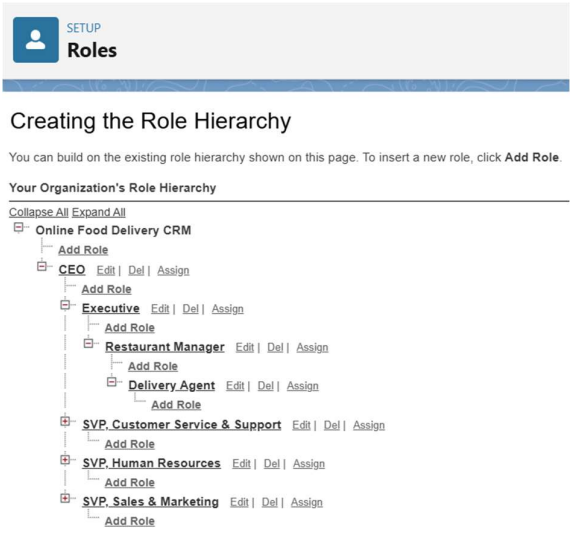
Profiles

- Cloned **Standard User** into custom profiles:
- Restaurant_Manager_Profile
- Delivery_Agent_Profile
- Executive_Profile



Roles

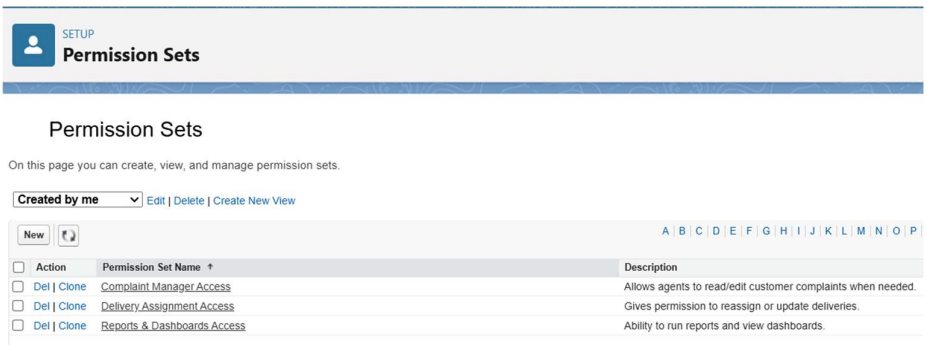
- **CEO (System Admin)** → Top-level, full access to all records.
- **Executive / Management** → Reports to CEO, sees all managers & agents records.
- **Restaurant Manager** → Reports to Executive, manages restaurants, menus, orders, and complaints.
- **Delivery Agent** → Reports to Manager, sees and updates only their assigned orders/deliveries.



Permission Sets

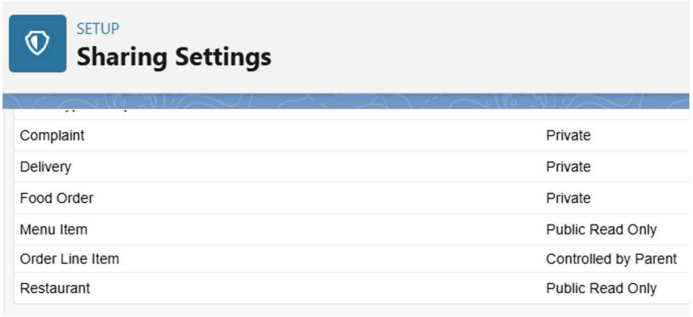
Permission sets provide extra access without changing user profiles.

- **Reports & Dashboards Access** → Grants executives (Priya Verma) ability to run reports and view dashboards.
- **Complaint Manager Access** → Allows delivery agents (Ravi Kumar) to read/edit customer complaints when needed.
- **Delivery Assignment Access** → Gives restaurant managers (Anita Sharma) permission to reassign or update deliveries.



Organization-Wide Defaults (OWD)

- Setup → Sharing Settings → Organization-Wide Defaults
- Configure default access for each object:
- **Restaurant__c:** Public Read Only
  - All users can view restaurants, but only managers/admins can edit.
- **Menu_Item__c:** Public Read Only
  - All users can see menu items.
- **Food_Orders__c:** Private
  - Only the owner (customer) and assigned staff can see the order.
- **Order_Line_Item__c:** Controlled by Parent
  - Inherits access from Food_Orders__c.
- **Delivery__c:** Private
  - Only assigned delivery agent and manager/admin can see.
- **Complaint__c:** Private
  - Only owner (customer) and support/admin can see.
- This ensures baseline record visibility and data security for all users.

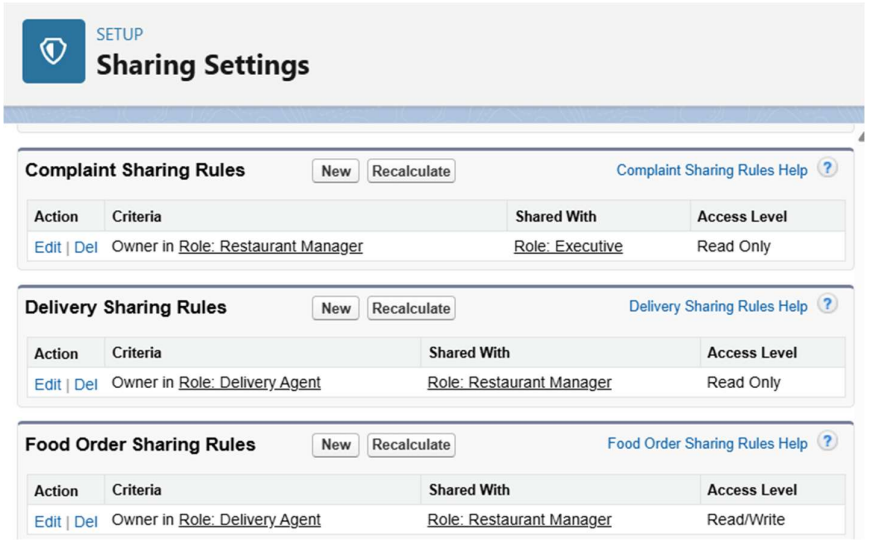


Sharing Rules

- Setup → Sharing Settings → Sharing Rules
- Sharing Rules grant additional access beyond OWD for specific business scenarios, ensuring that users in certain roles can view or edit records as needed.

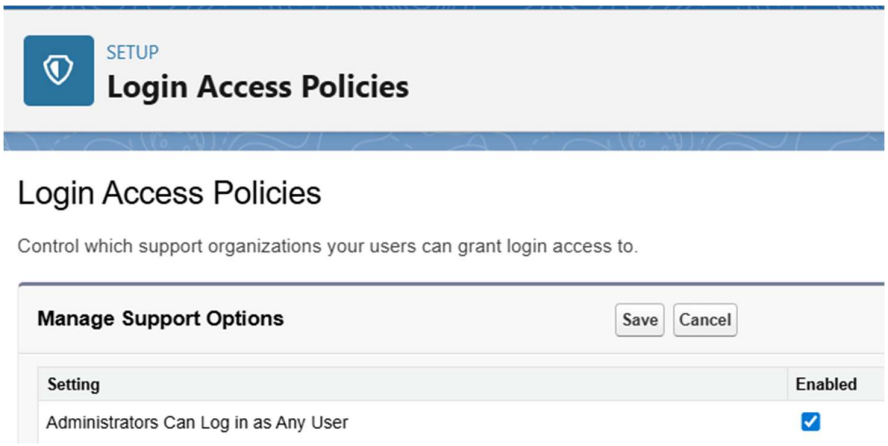
Rules:

1. **Complaint Sharing Rule**
    - **Access Level:** Read Only
    - Allows executives to view complaints raised by restaurant managers.
2. **Delivery Sharing Rule**
    - **Access Level:** Read Only
    - Enables restaurant managers to view deliveries assigned to their agents.
3. **Food Order Sharing Rule**
    - **Access Level:** Read/Write
    - Lets restaurant managers view and update orders handled by delivery agents.



Login Access Policies

- Setup → Login Access Policies.
- Enabled Administrators Can Log in as Any User.
- Allows Admin to test features as Manager, Agent, or Executive.



Deployment Basics

- Development in VS Code + Salesforce CLI.
- Metadata synced via package.xml.
- Source code stored in GitHub repo.
- **Deployment flow:**
  VS Code (local changes) → Push to Salesforce Org → Commit to GitHub.

.....

Phase 3: Data Modeling & Relationships

Standard & Custom Objects

Standard Objects:

- **Contact:** Represents customers placing orders.
- **User:** Represents internal users like Restaurant Managers, Delivery Agents, and Executives.

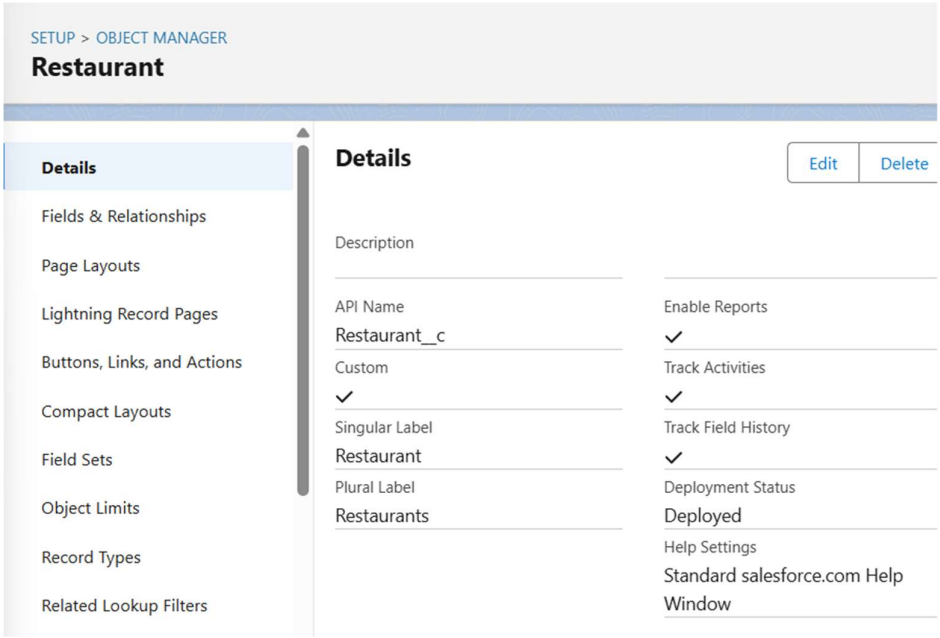
Custom Objects:

- Setup → Object Manager → Create → Custom Object
- Used to manage business-specific data that is not captured by standard objects like Account, Contact, or User.

1. Restaurant__c

Purpose:

Stores information about restaurants in the Food Delivery CRM, including contact details, location, and cuisine type.

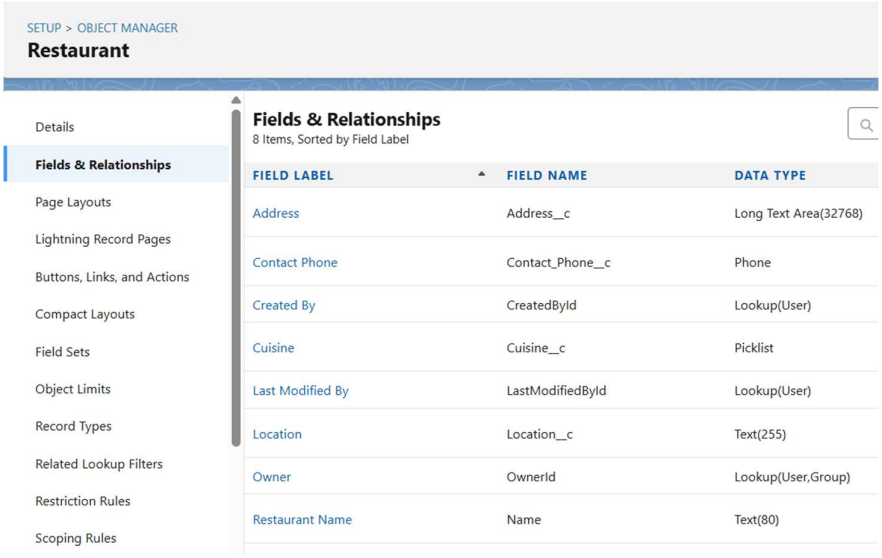


Fields:

- Restaurant Name (Name, Text 80): The name of the restaurant.
- Address__c (Long Text Area 32768): Full address of the restaurant.
- Location__c (Text 255): Short location or area name.
- Contact_Phone__c (Phone): Restaurant contact number.
- Cuisine__c (Picklist): Type of cuisine offered (e.g., Indian, Italian).
- OwnerId (Lookup to User/Group): The user or group who owns the record.
- CreatedById (Lookup to User): System field indicating who created the record.
- LastModifiedById (Lookup to User): System field indicating who last modified the record.

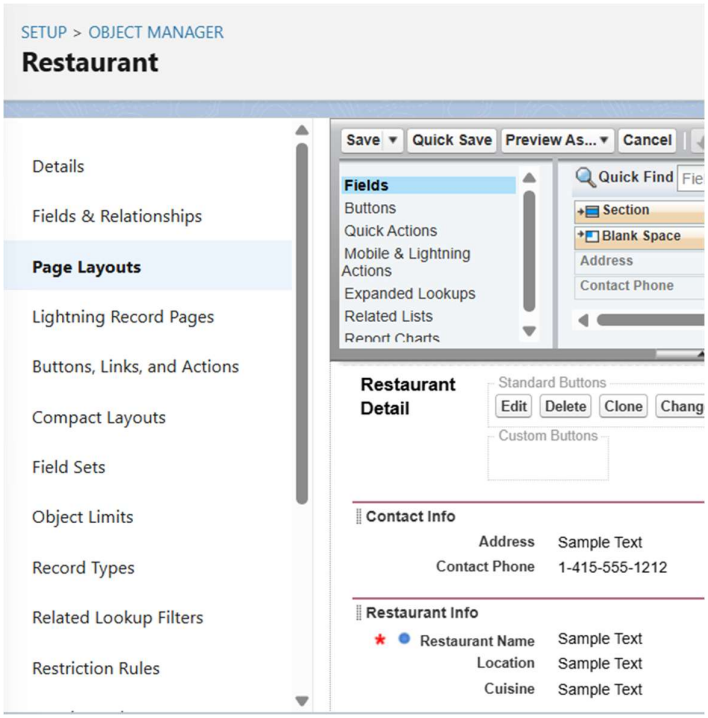
Relationships:

- Menu_Item__c → Restaurant__c (Lookup): Links menu items to the restaurant.
- Food_Orders__c → Restaurant__c (Lookup): Links orders to the restaurant.



Page Layouts:

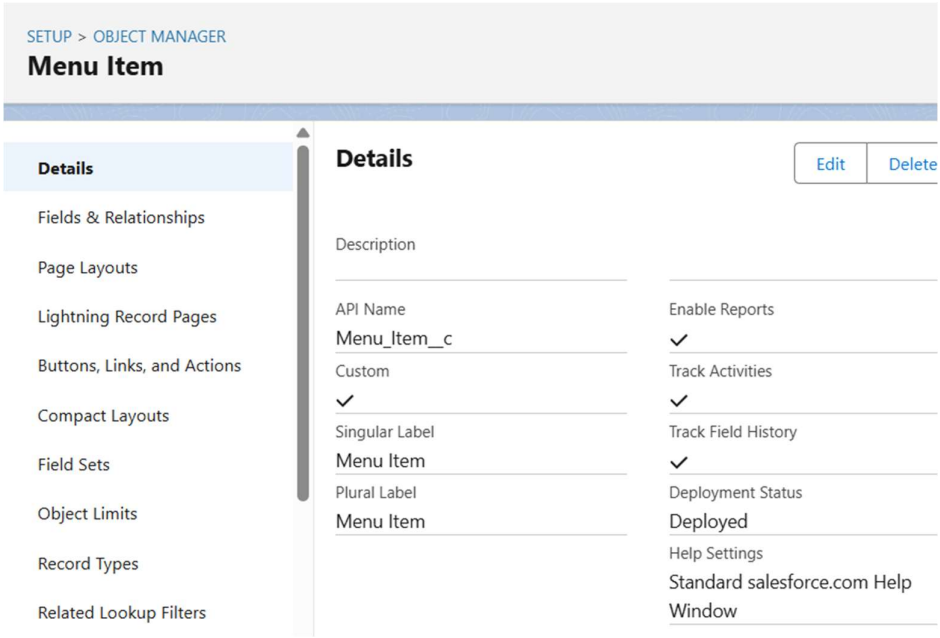
- Contact Info Section: Address, Contact Phone
- Restaurant Info Section: Restaurant Name, Location, Cuisine



2. Menu_Item__c

Purpose:

Stores details of food items offered by restaurants, including price, category, availability, and description.

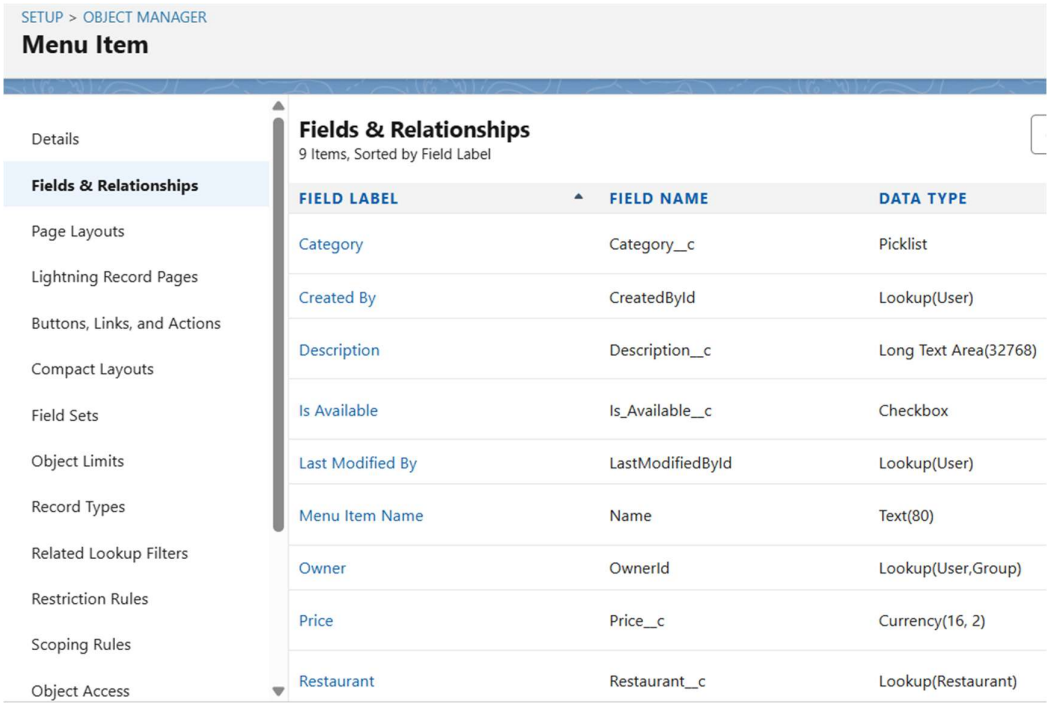


Fields:

- Menu Item Name (Name, Text 80): Name of the menu item.
- Price__c (Currency 16,2): Price of the menu item.
- Category__c (Picklist): Type/category of the food item (e.g., Appetizer, Main Course).
- Description__c (Long Text Area 32768): Detailed description of the item.
- Is_Available__c (Checkbox): Indicates if the item is currently available.
- Restaurant__c (Lookup to Restaurant__c): Associated restaurant for the menu item.
- OwnerId (Lookup to User/Group): Owner of the record.
- CreatedById (Lookup to User): System field indicating who created the record.
- LastModifiedById (Lookup to User): System field indicating who last modified the record.

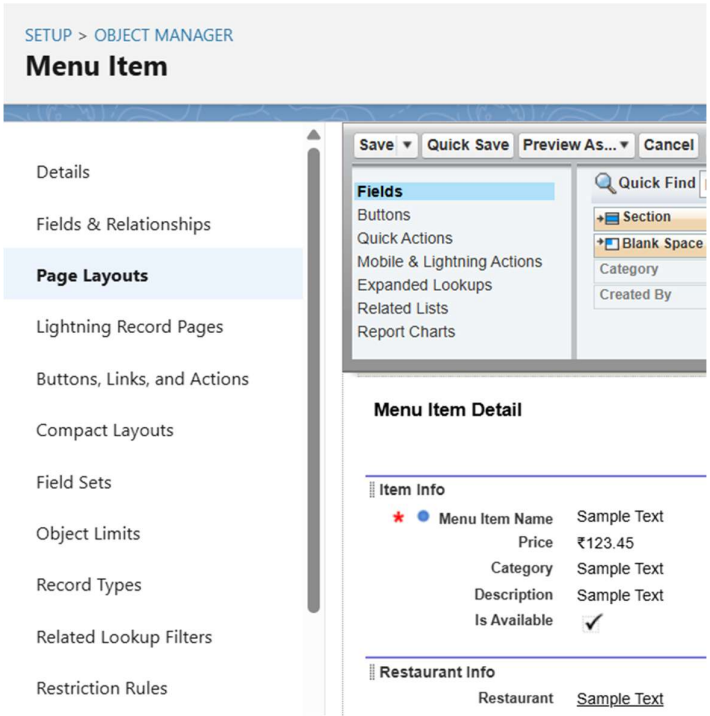
Relationships:

- Menu_Item__c → Restaurant__c (Lookup): Links menu items to the restaurant.
- Order_Line_Item__c → Menu_Item__c (Lookup): Links line items to menu items in orders.



Page Layouts:

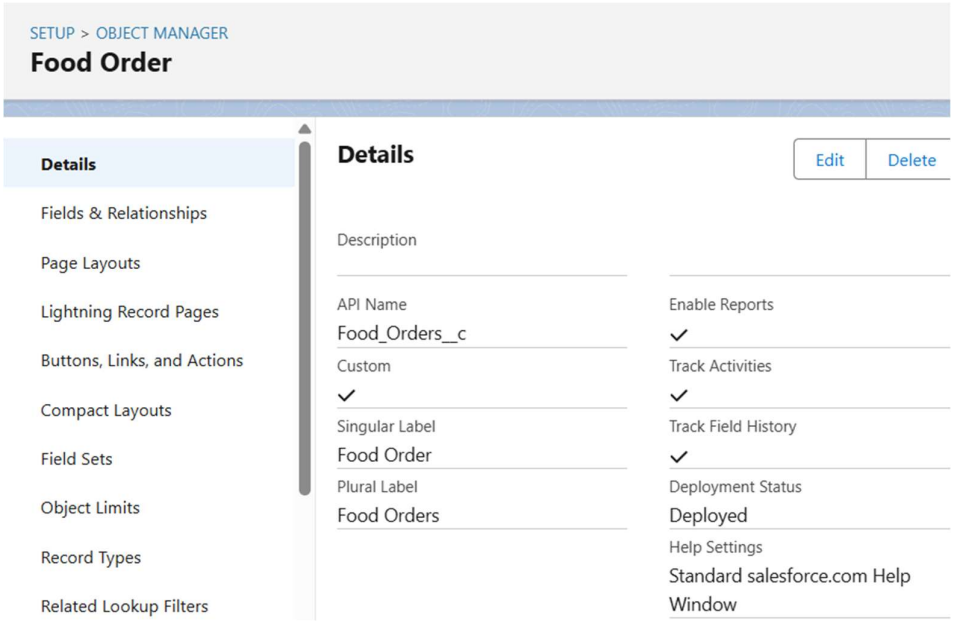
- Item Info Section: Menu Item Name, Price, Category, Description, Is Available
- Restaurant Info Section: Restaurant



3. Food_Orders__c

Purpose:

Represents customer orders in the Food Delivery CRM, including order details, customer information, restaurant, and payment status.

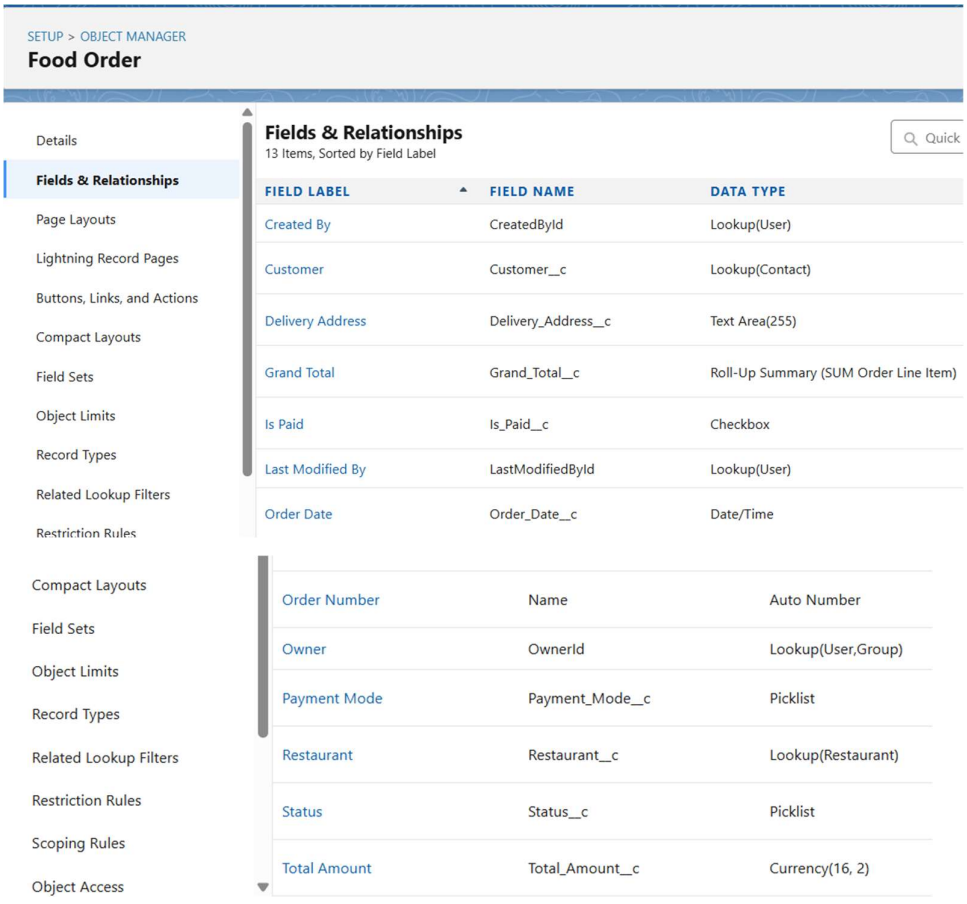


Fields:

- Order Number (Name, Auto Number): Unique identifier for the order.
- Customer__c (Lookup to Contact): Customer who placed the order.
- Delivery_Address__c (Text Area 255): Address where the order should be delivered.
- Restaurant__c (Lookup to Restaurant__c): Restaurant fulfilling the order.
- Order_Date__c (Date/Time): Date and time when the order was placed.
- Status__c (Picklist): Current status of the order (e.g., Pending, Completed).
- Total_Amount__c (Currency 16,2): Total amount for the order.
- Grand_Total__c (Roll-Up Summary, SUM of Order Line Items): Total of all line items.
- Payment_Mode__c (Picklist): Mode of payment (e.g., Cash, Card, Online).
- Is_Paid__c (Checkbox): Indicates whether the order has been paid.
- OwnerId (Lookup to User/Group): Owner of the record.
- CreatedById (Lookup to User): System field indicating who created the record.
- LastModifiedById (Lookup to User): System field indicating who last modified the record.

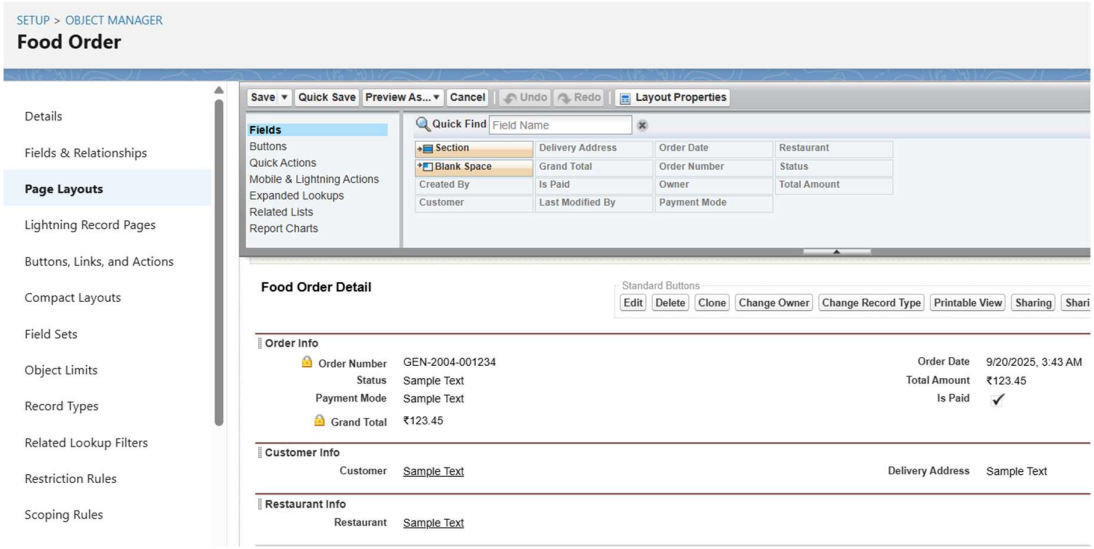
Relationships:

- Food_Orders__c → Customer__c (Lookup to Contact): Links order to customer.
- Food_Orders__c → Restaurant__c (Lookup): Links order to restaurant.
- Order_Line_Item__c → Food_Orders__c (Master-Detail): Links line items to the parent order.
- Delivery__c → Food_Orders__c (Lookup): Links delivery record to the order.
- Complaint__c → Food_Orders__c (Lookup): Links complaints to the order.



Page Layouts:

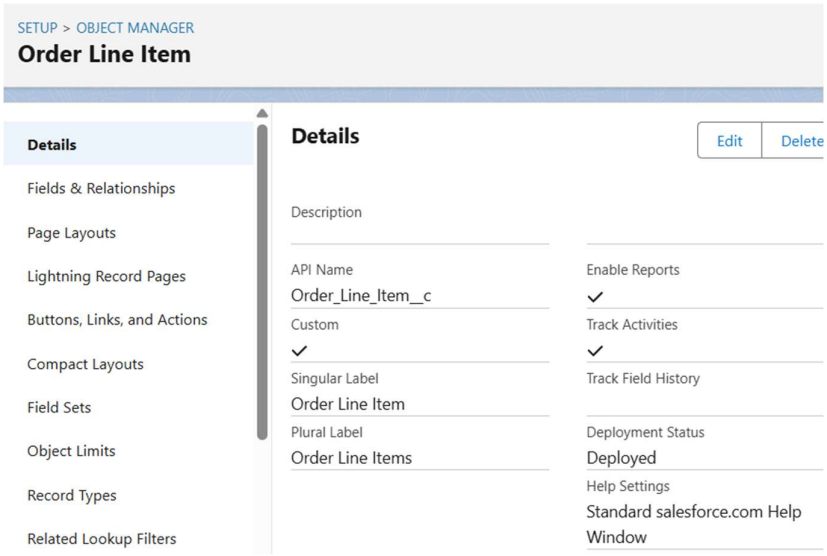
- Order Info Section: Order Number, Order Date, Status, Total Amount, Payment Mode, Is Paid, Grand Total
- Customer Info Section: Customer, Delivery Address
- Restaurant Info Section: Restaurant



4. Order_Line_Item__c

Purpose:

Represents individual items within a food order, including the menu item, quantity, pricing, and calculated total.



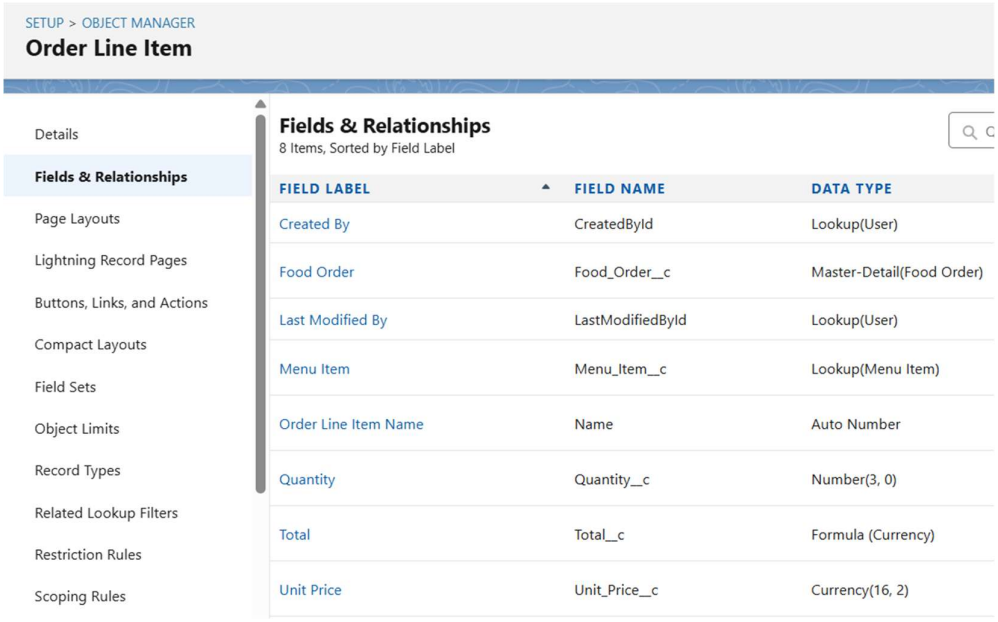
Fields:

- Order Line Item Name (Name, Auto Number): Unique identifier for the line item.
- Food_Order__c (Master-Detail to Food_Orders__c): Parent order to which this line item belongs.

- Menu_Item__c (Lookup to Menu_Item__c): Menu item included in this line item.
- Quantity__c (Number 3,0): Number of units ordered.
- Unit_Price__c (Currency 16,2): Price per unit of the menu item.
- Total__c (Formula, Currency): Calculated total for this line item (Quantity × Unit Price).
- OwnerId (Lookup to User/Group): Owner of the record.
- CreatedById (Lookup to User): System field indicating who created the record.
- LastModifiedById (Lookup to User): System field indicating who last modified the record.

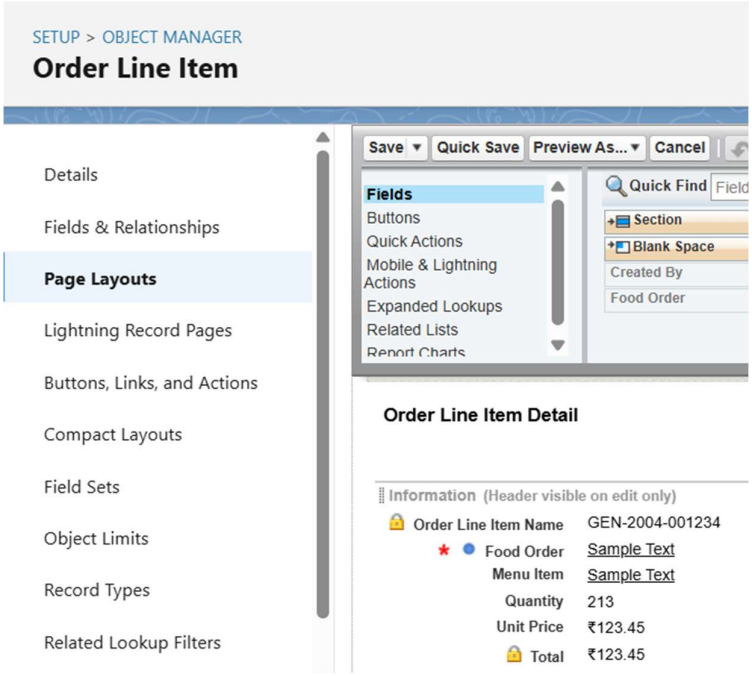
Relationships:

- Order_Line_Item__c → Food_Orders__c (Master-Detail): Links line item to parent food order.
- Order_Line_Item__c → Menu_Item__c (Lookup): Links line item to the menu item being ordered.



Page Layouts:

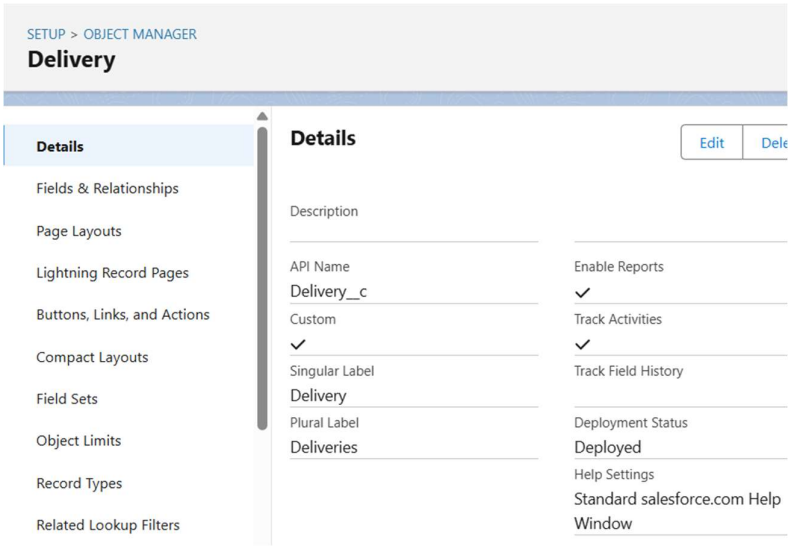
- Information Section: Order Line Item Name, Food Order, Menu Item, Quantity, Unit Price, Total



5. Delivery__c

Purpose:

Tracks the delivery details of food orders, including assigned delivery agent, estimated time of arrival (ETA), and delivery status.



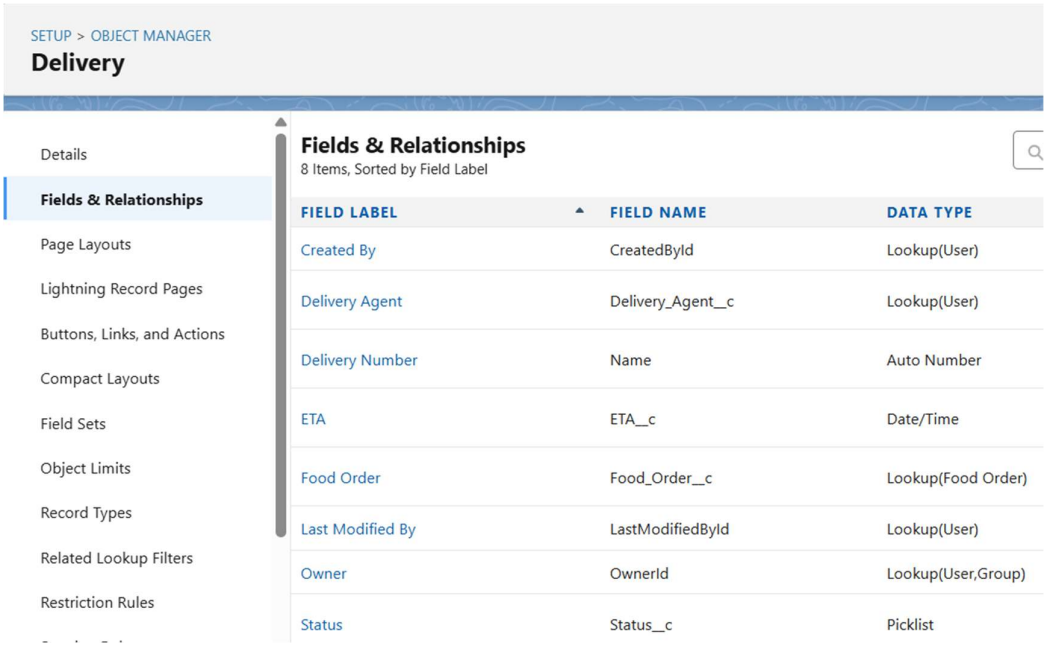
Fields:

- Delivery Number (Name, Auto Number): Unique identifier for each delivery.
- Delivery_Agent__c (Lookup to User): Delivery agent assigned to this delivery.

- Food_Order__c (Lookup to Food_Orders__c): Links the delivery to its associated food order.
- ETA__c (Date/Time): Estimated time of arrival for the delivery.
- Status__c (Picklist): Current status of the delivery (e.g., Assigned, Out for Delivery, Delivered).
- OwnerId (Lookup to User/Group): The owner of the record.
- CreatedById (Lookup to User): System field indicating who created the record.
- LastModifiedById (Lookup to User): System field indicating who last modified the record.

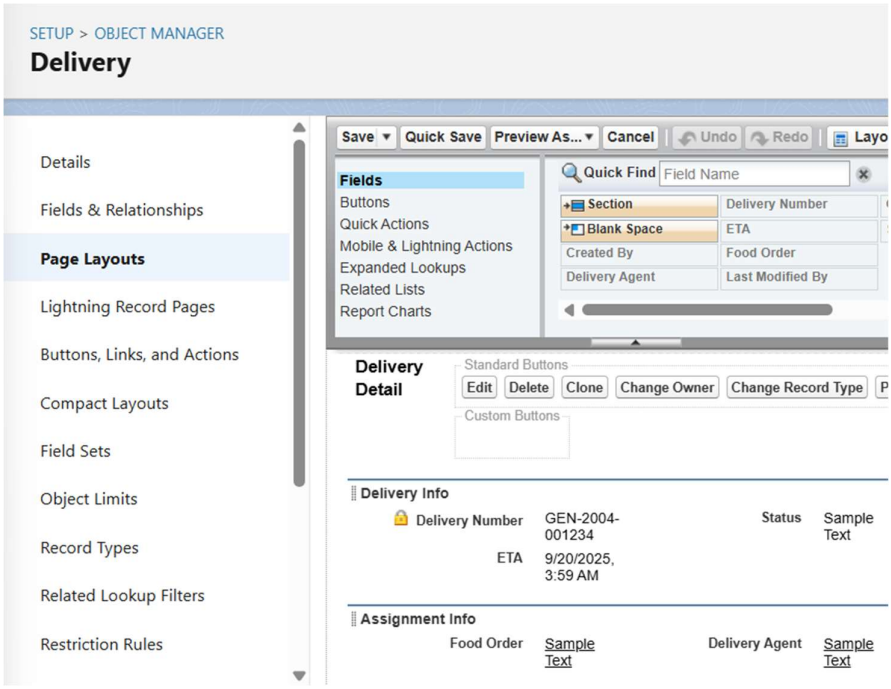
Relationships:

- Delivery__c → Food_Orders__c (Lookup): Associates each delivery with a specific food order.
- Delivery__c → User (Lookup): Links the delivery record to the assigned delivery agent.



Page Layouts:

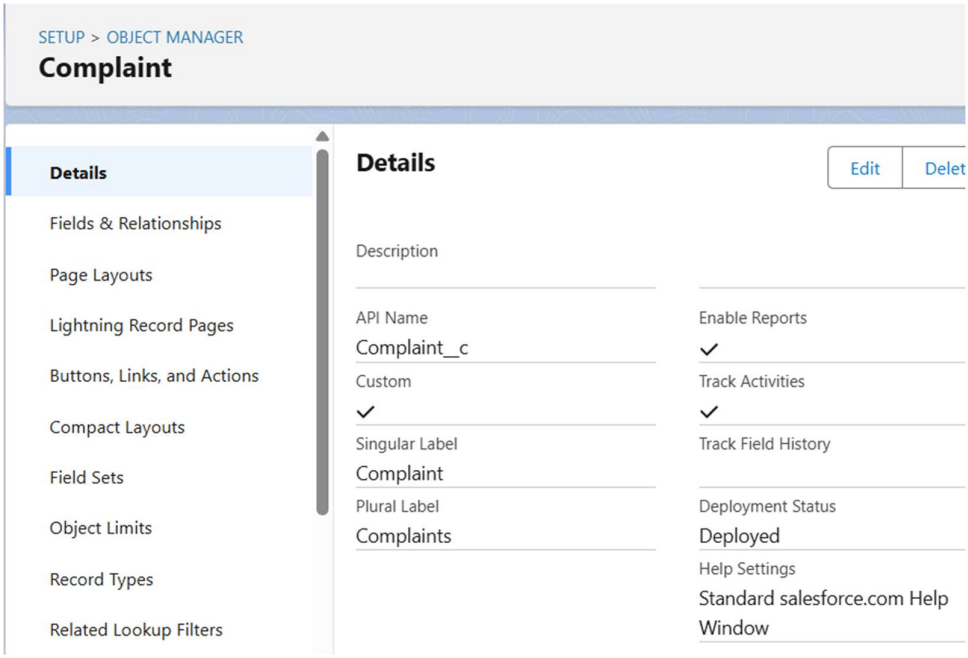
- Delivery Info Section: Delivery Number, Status, ETA
- Assignment Info Section: Food Order, Delivery Agent



6. Complaint__c

Purpose:

Captures customer complaints related to food orders, including complaint type, status, and customer feedback.

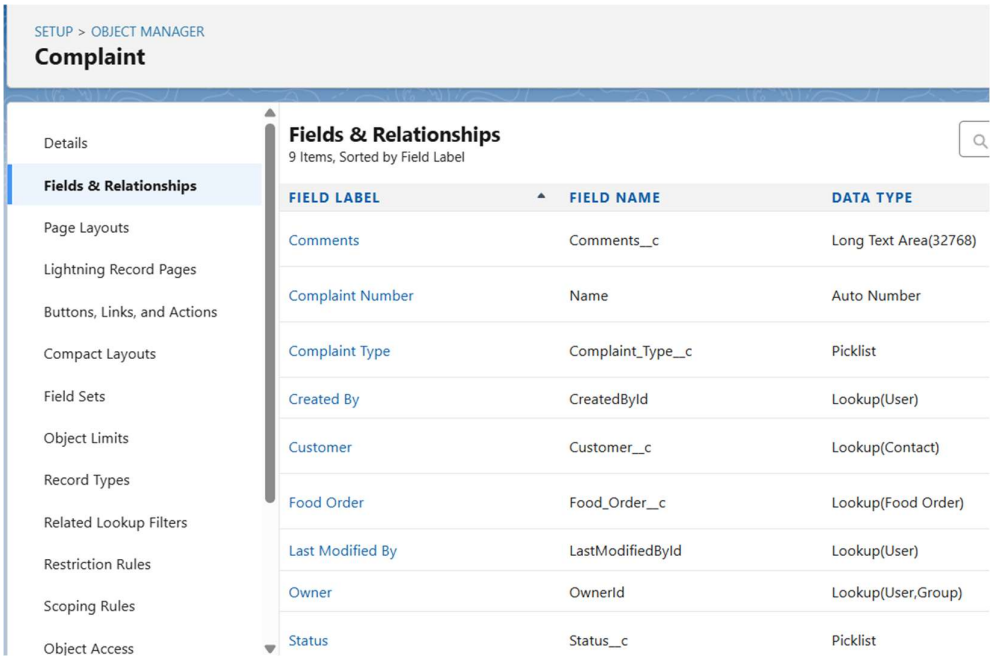


Fields:

- Complaint Number (Name, Auto Number): Unique identifier for each complaint.
- Complaint_Type__c (Picklist): Type of complaint (e.g., Late Delivery, Wrong Item, Quality Issue).
- Status__c (Picklist): Current status of the complaint (e.g., Open, In Progress, Resolved, Closed).
- Comments__c (Long Text Area 32768): Detailed description or comments provided by the customer.
- Food_Order__c (Lookup to Food_Orders__c): Links the complaint to the related food order.
- Customer__c (Lookup to Contact): Customer who raised the complaint.
- OwnerId (Lookup to User/Group): Owner of the record (e.g., Support Agent).
- CreatedById (Lookup to User): System field indicating who created the record.
- LastModifiedById (Lookup to User): System field indicating who last modified the record.

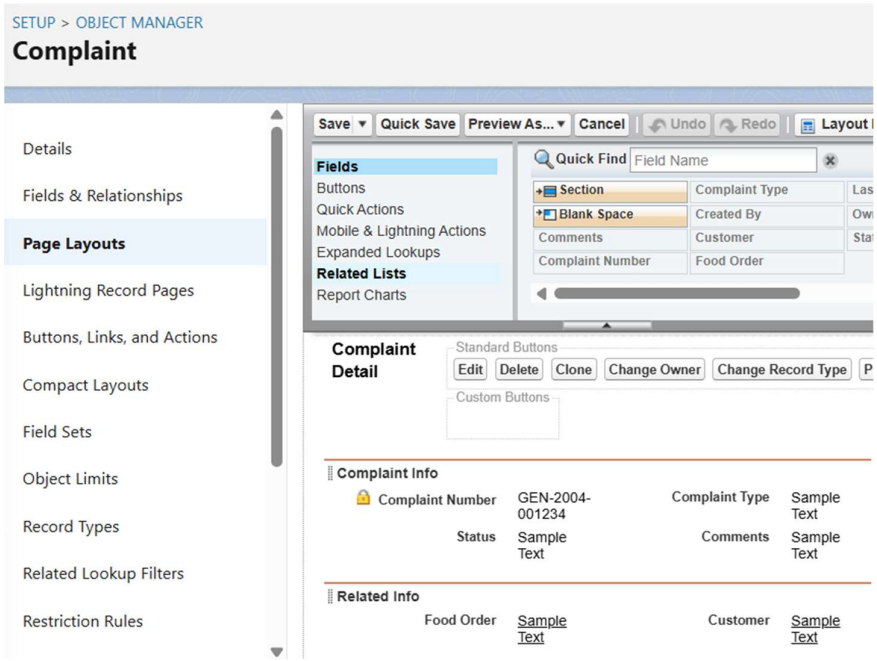
Relationships:

- Complaint__c → Food_Orders__c (Lookup): Associates the complaint with a specific order.
- Complaint__c → Contact (Lookup): Links the complaint to the customer who submitted it.



Page Layouts:

- Complaint Info Section: Complaint Number, Complaint Type, Status, Comments
- Related Info Section: Food Order, Customer



Lookup vs Master-Detail vs Hierarchical Relationships

- **Restaurant__c ↔ Menu_Item__c → Lookup**

  A menu item is linked to a restaurant, but deleting a restaurant doesn't automatically delete its menu items.
- **Food_Orders__c ↔ Restaurant__c → Lookup**

  An order is associated with a restaurant, but restaurants don't own the orders.
- **Food_Orders__c ↔ Contact (Customer) → Lookup**

  Orders are linked to customers (Contacts), but customers don't "own" the order records.
- **Food_Orders__c ↔ Order_Line_Item__c → Master-Detail**

  Line items cannot exist without a parent order. Deleting a Food Order deletes all related Order Line Items.
- **Delivery__c ↔ Food_Orders__c → Lookup**

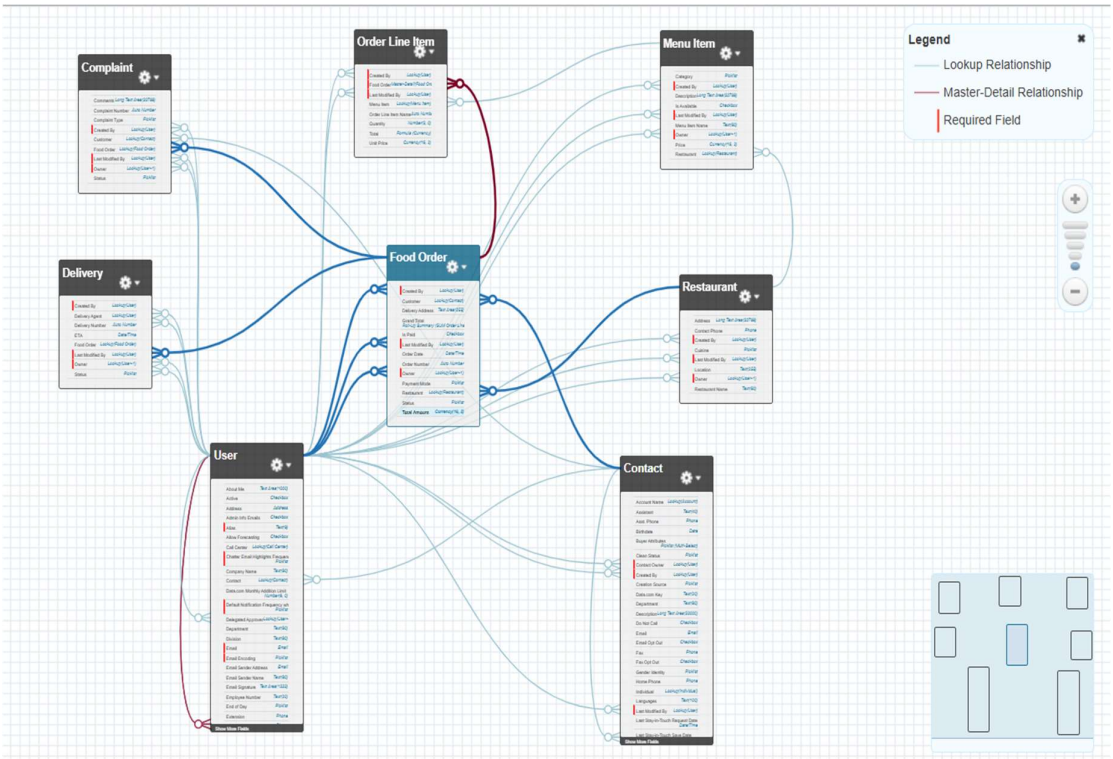
  Each delivery record is linked to an order, but the delivery doesn't own the order.
- **Complaint__c ↔ Food_Orders__c → Lookup**

  Complaints are tied to specific orders, but they can exist independently.

- **Complaint__c ↔ Contact (Customer) → Lookup**
  Complaints are linked to the customer who raised them, but customers don't own the complaints.

Schema Builder

- Setup → Object Manager → Schema Builder
- A visual tool in Salesforce that lets you view and manage objects, fields, and relationships in a single interface.



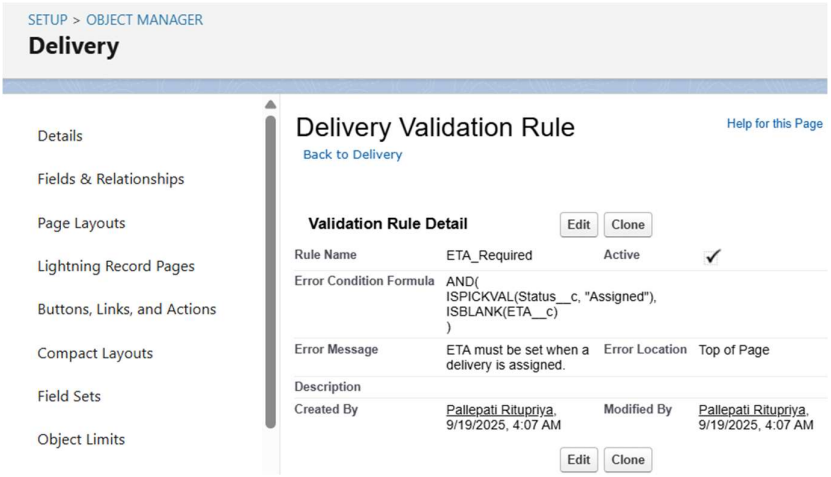
Phase 4: Process Automation (Admin)

Validation Rules

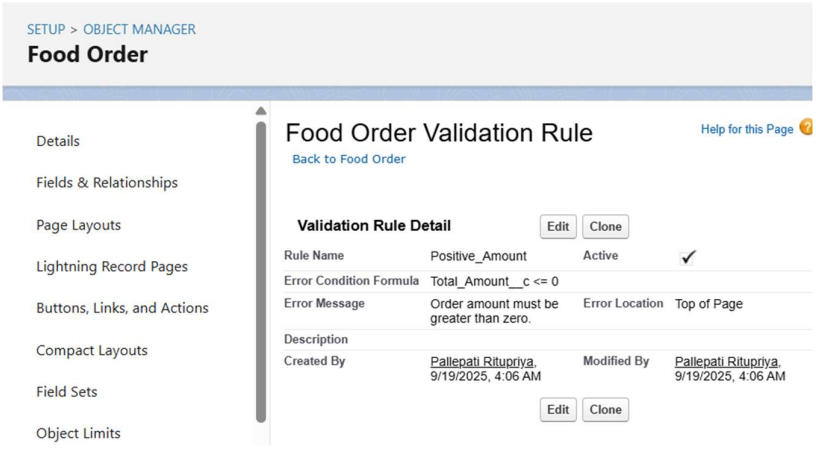
• Setup → Object Manager → Select Object → Validation Rules
• Validation Rules enforce data integrity and business logic, preventing incomplete or invalid data entry.

Rules:

1. **Delivery – ETA_Required**
   - Error Condition Formula: AND(ISPICKVAL(Status__c, "Assigned"), ISBLANK(ETA__c))
   - Error Message: ETA must be set when a delivery is assigned.



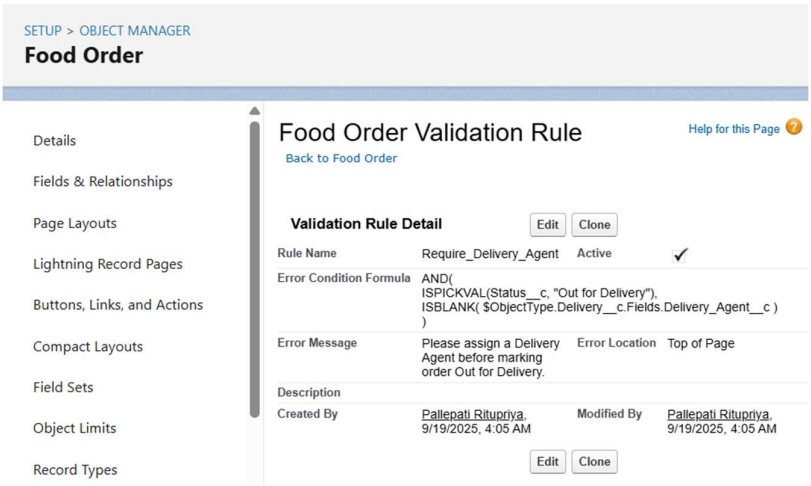
2. **Food Order – Positive_Amount**
   - Error Condition Formula: Total_Amount__c <= 0
   - Error Message: Order amount must be greater than zero.



3. **Food Order – Require_Delivery_Agent**
   - Error Condition Formula: AND(ISPICKVAL(Status__c, "Out for Delivery"),

ISBLANK($ObjectType.Delivery__c.Fields.Delivery_Agent__c))

• Error Message: Please assign a Delivery Agent before marking order Out for Delivery.



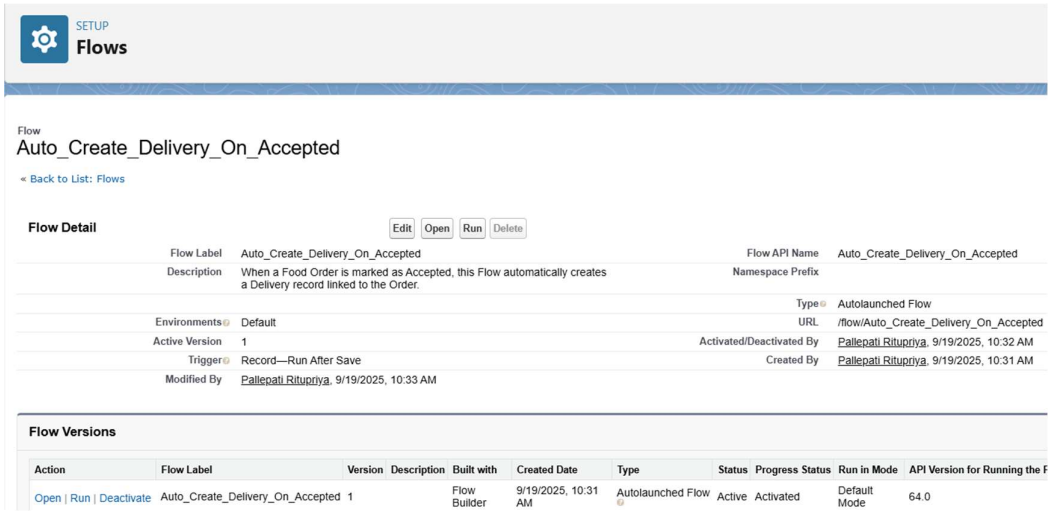
Flow Builder

- Setup → Flow Builder → New Flow → Record-Triggered → Auto-launched Flow
- Record-Triggered Flows run automatically when a record is created or updated, enabling automation without user intervention.

1. **Auto_Create_Delivery_On_Accepted**

   • **Trigger:** Record—Run After Save

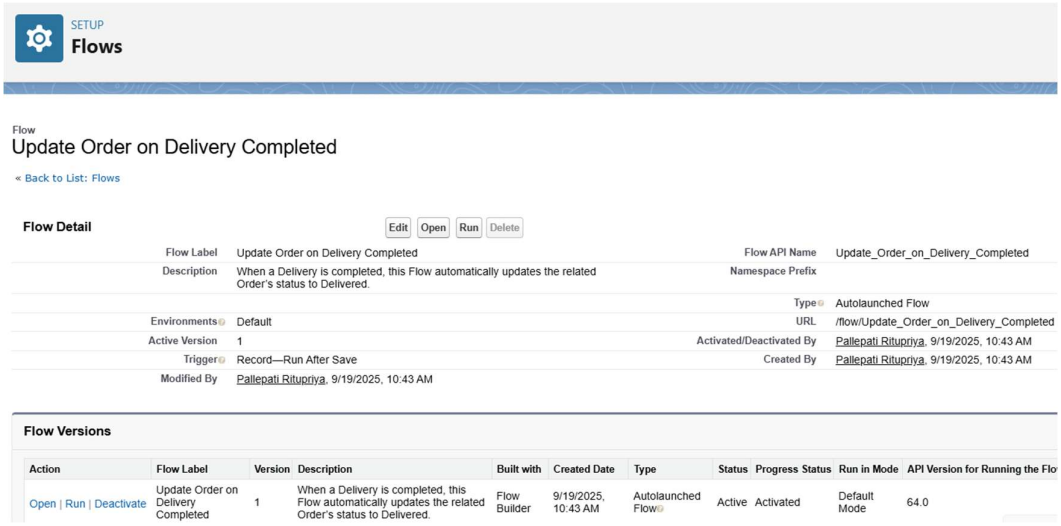
   • **Description:** When a Food Order is marked as Accepted, this flow automatically creates a Delivery record linked to the order.



2. **Update_Order_on_Delivery_Completed**

   • **Trigger:** Record—Run After Save

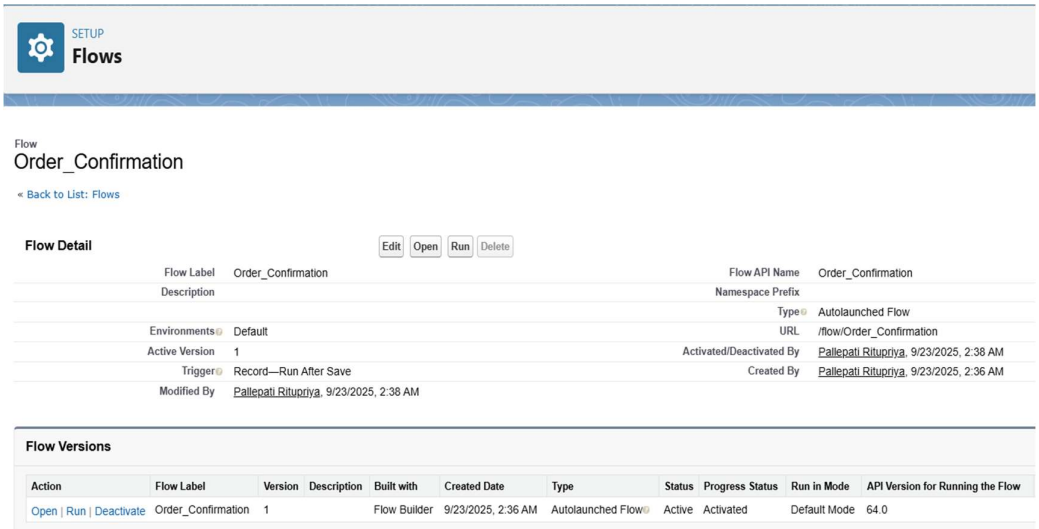
   • **Description:** When a Delivery is completed, this flow automatically updates the related Food Order status to Delivered.



3. **Order_Confirmation Flow**

   • **Trigger:** Record—Run After Save

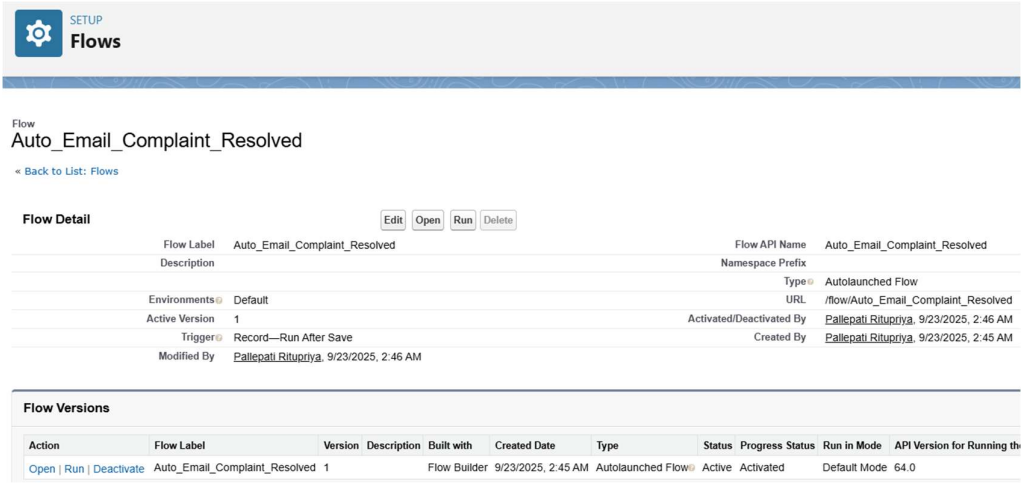
   • **Description:** When a Food Order is placed, this flow automatically sends an order confirmation email to the customer.



4. **Auto_Email_Complaint_Resolved**

• **Trigger:** Record—Run After Save

• **Description:** When a Complaint status is updated to “Resolved”, this flow automatically sends an email alert to notify the customer.



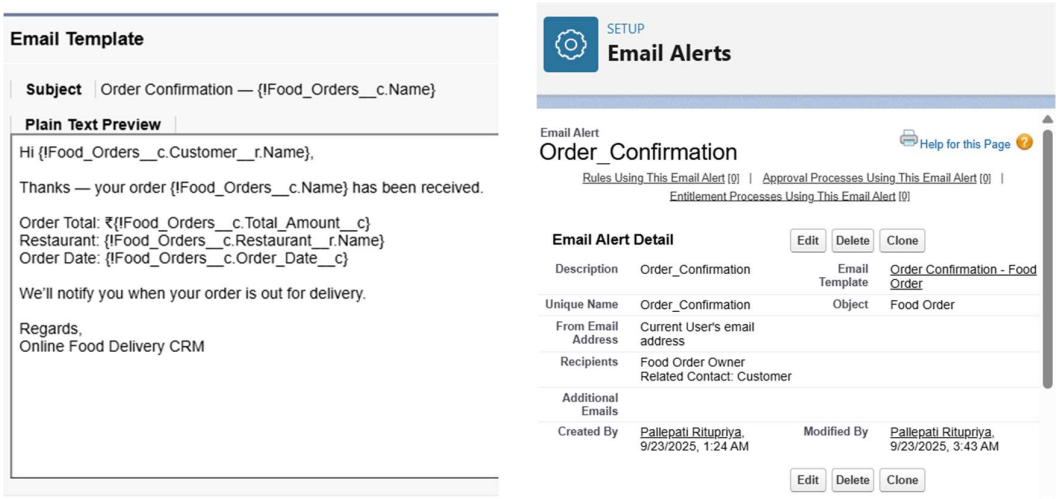
Purpose / Outcome:

- Automates business processes to reduce manual work.
- Ensures data consistency and real-time updates across related objects.
- Supports accurate tracking of orders and deliveries.
- Improves customer engagement and operational efficiency.

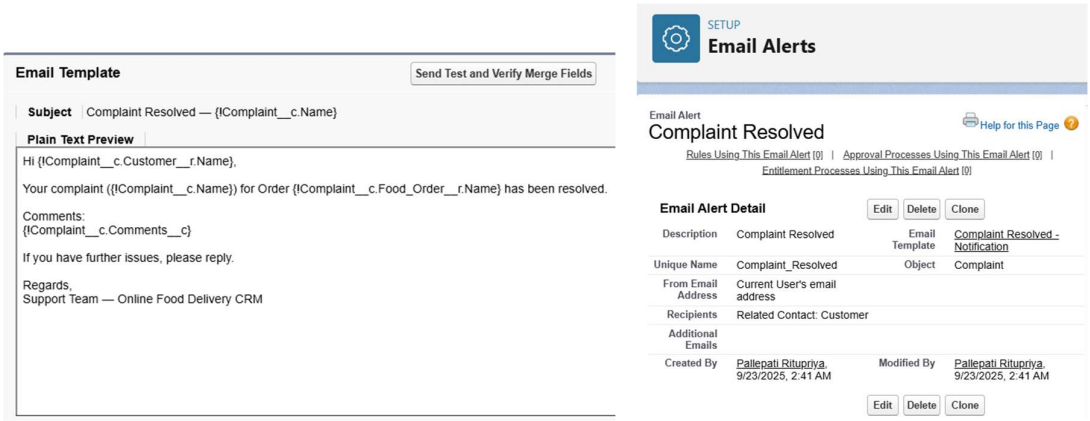
Email Alerts

- Setup → Workflow & Approvals → Email Alerts → New Email Alert.
- Email alerts automatically send emails based on specific conditions or triggers in Salesforce, ensuring timely communication with customers or internal users.

1. **Order Confirmation Email Alert**
    - **Purpose:** Sends confirmation email to customer when a new order is placed.
    - **Object**: Food_Orders__c.
    - Setup → Email → Classic Email Templates → New Template.
    - **Template Name**: Order Confirmation – Food Orders
    - Template used to confirm orders to customers.



2. **Complaint Resolved Email Alert**
    - **Purpose:** Sends notification email to customer when a complaint is resolved.
    - **Object:** Complaint__c
    - **Template Name**: Complaint Resolved - Notification
    - Template used to notify customers when their complaint is resolved.



.....

Phase 5: Apex Programming (Developer)

Apex Classes & Objects

- Developer Console → File → New → Apex Class.

- **Purpose:** These classes encapsulate all business logic for Food Orders, Order Line Items, Menu Items, and Complaints. They ensure that triggers remain clean and maintainable while allowing reusability and separation of concerns.

Apex Classes:

1. FoodOrderService

- **Object:** Food_Orders__c
- **Purpose:** Handles business logic for Food Orders, including calculating totals, updating statuses, and other operations triggered by Apex Triggers.

FoodOrderService.apxc

Code Coverage: None | API Version: 64

```
public class FoodOrderService {
    public static void calculateGrandTotal(Set<Id> foodOrderIds){
        List<Food_Orders__c> ordersToUpdate = [SELECT Id,
                                               (SELECT Quantity__c, Unit_Price__c FROM Order_Line_Items__r)
                                               FROM Food_Orders__c WHERE Id IN :foodOrderIds];
        for(Food_Orders__c order : ordersToUpdate){
            Decimal total = 0;
            for(Order_Line_Item__c item : order.Order_Line_Items__r){
                total += item.Quantity__c * item.Unit_Price__c;
            }
            order.Total_Amount__c = total;
        }
        try {
            update ordersToUpdate;
        } catch(DmlException e){
            System.debug('Error updating Food Orders total: ' + e.getMessage());
        }
     }
}
```

2. FoodOrderManagement

- **Object:** Food_Orders__c
- **Purpose:** Acts as a centralized handler for Food Order triggers to keep triggers clean and maintainable.

FoodOrderManagement.apxc *

Code Coverage: None | API Version: 64

```
public class FoodOrderManagement {
    // Create a new order
    public static Food_Orders__c createOrder(Contact customer, Restaurant__c restaurant, Decimal totalAmount) {
        Food_Orders__c order = new Food_Orders__c(
            Customer__c = customer.Id,
            Restaurant__c = restaurant.Id,
            Total_Amount__c = totalAmount,
            Status__c = 'New',
            Order_Date__c = System.now()
        );
        insert order;
        return order;
    }
    // Update order status
    public static void updateOrderStatus(Id orderId, String newStatus) {
        Food_Orders__c order = [SELECT Id, Status__c FROM Food_Orders__c WHERE Id = :orderId];
        order.Status__c = newStatus;
        update order;
    }
```

```
    // Calculate Grand Total from Order Line Items
    public static Decimal calculateGrandTotal(Id orderId) {
        Decimal total = 0;
        List<Order_Line_Item__c> items = [SELECT Total__c FROM Order_Line_Item__c WHERE Food_Order__c = :orderId];
        for(Order_Line_Item__c item : items){
            total += item.Total__c;
        }
        return total;
    }
}
```

3. ComplaintHandler

- Object: Complaint__c
- Purpose: Manages business logic for complaints, including default values, validation, and related operations.

ComplaintHandler.apxc *

Code Coverage: None API Version: 64

```
public class ComplaintHandler {
    public static void updateOrderOnComplaint(List<Complaint__c> newComplaints,
                                              Map<Id, Complaint__c> oldMap) {
        Set<Id> orderIds = new Set<Id>();
        for(Complaint__c comp : newComplaints){
            Complaint__c oldComp = oldMap.get(comp.Id);
            if(comp.Status__c == 'Resolved' && oldComp.Status__c != 'Resolved'){
                if(comp.Food_Order__c != null)
                    orderIds.add(comp.Food_Order__c);
            }
        }
        if(orderIds.isEmpty()) return;

        List<Food_Orders__c> ordersToUpdate = [SELECT Id, Status__c FROM Food_Orders__c WHERE Id IN :orderIds];
        for(Food_Orders__c order : ordersToUpdate){
            order.Status__c = 'Delivered';
        }
        try {
            update ordersToUpdate;
        } catch(DmlException e){
            System.debug('Error updating Food Orders from Complaint: ' + e.getMessage());
        }
    }
}
```

4. OrderLineItemHandler

- Object: Order_Line_Item__c
- Purpose: Calculates totals for order line items and updates parent Food Orders. Ensures consistent data integrity.

OrderLineItemHandler.apxc

Code Coverage: None API Version: 64

```apex
public class OrderLineItemHandler {

    public static void updateFoodOrderTotal(Map<Id, Order_Line_Item__c> newMap,
                                            Map<Id, Order_Line_Item__c> oldMap,
                                            Boolean isDelete) {
        Set<Id> foodOrderIds = new Set<Id>();

        if(isDelete){
            for(Order_Line_Item__c oldItem : oldMap.values()){
                if(oldItem.Food_Order__c != null)
                    foodOrderIds.add(oldItem.Food_Order__c);
            }
        } else {
            for(Order_Line_Item__c item : newMap.values()){
                if(item.Food_Order__c != null)
                    foodOrderIds.add(item.Food_Order__c);
            }
        }

        FoodOrderService.calculateGrandTotal(foodOrderIds);
    }
}
```

Apex Triggers

- Developer Console → File → New → Apex Trigger.
- **Purpose:** Apex triggers automate business processes when records are created, updated, or deleted. In your Food Delivery CRM, they handle actions like updating Order totals and creating Delivery records, while keeping logic centralized and bulk-safe via handler classes.

ComplaintTrigger

- **Object:** Complaint__c
- **Purpose:** Automates actions when complaints are created or updated, such as assigning owners or sending notifications via the handler class.

ComplaintTrigger.apxt *

Code Coverage: None API Version: 64

```apex
trigger ComplaintTrigger on Complaint__c (after update) {
    ComplaintHandler.updateOrderOnComplaint(Trigger.new, Trigger.oldMap);
}
```

FoodOrderTrigger

- **Object:** Food_Orders__c
- **Purpose:** Handles actions after an order is updated, like calculating the grand total or triggering other business processes via FoodOrderService.

FoodOrderTrigger.apxt *

Code Coverage: None API Version: 64

```
trigger FoodOrderTrigger on Food_Orders__c (after update) {
    Set<Id> foodOrderIds = new Set<Id>();

    for(Food_Orders__c order : Trigger.new){
        Food_Orders__c oldOrder = Trigger.oldMap.get(order.Id);

        // If any relevant field changed (e.g., Status), we can add it here
        if(order.Status__c != oldOrder.Status__c){
            foodOrderIds.add(order.Id);
        }
    }

    if(!foodOrderIds.isEmpty()){
        // Recalculate totals or other logic in your service class
        FoodOrderService.calculateGrandTotal(foodOrderIds);
    }
}
```

OrderLineItemTrigger

- **Object:** Order_Line_Item__c
- **Purpose:** Updates the parent Food Order's total whenever order line items are inserted, updated, or deleted, keeping order amounts accurate.

OrderLineItemTrigger.apxt *

Code Coverage: None API Version: 64

```
trigger OrderLineItemTrigger on Order_Line_Item__c (after insert, after update, after delete) {
    if(Trigger.isInsert || Trigger.isUpdate){
        OrderLineItemHandler.updateFoodOrderTotal(Trigger.newMap, Trigger.oldMap, false);
    } else if(Trigger.isDelete){
        OrderLineItemHandler.updateFoodOrderTotal(null, Trigger.oldMap, true);
    }
}
```

Trigger Design Pattern

In this project, we used the Trigger + Handler design pattern to keep business logic separated from triggers.

- Triggers only handle context (before/after insert, update, delete).
- Handler classes contain the actual business logic.

This makes the code more readable, reusable, and easier to maintain.

Where used:

- ComplaintTrigger → ComplaintHandler (manage complaint creation and updates)
- FoodOrderTrigger → FoodOrderService (automate total calculation, validations on orders)

- OrderLineItemTrigger → OrderLineItemHandler (update order totals when line items change)

SOQL & SOSL

SOQL (Salesforce Object Query Language)

- Similar to SQL, but used only for Salesforce data.
- Allows you to query records from standard and custom objects.
- You can filter, order, and select specific fields.

SOSL (Salesforce Object Search Language)

- Used for full-text search across multiple objects at once.
- Useful when you don't know which object/field has the data.
- Returns results grouped by object.

Collections

Collections are data structures that hold multiple records in memory. In Salesforce Apex, we mainly use List, Set, and Map.

1. List

- Ordered collection (like an array in Java).
- Can contain duplicate records.
- Indexed (you can access by position).

Used in SOQL queries, inserting multiple records at once, and storing results from Apex methods.

2. Set

- Unordered collection of unique elements.
- Automatically removes duplicates.
- Useful when working with Ids.

Used in Service classes and Triggers to avoid duplicate processing.

3. Map

- Key-value pair collection.

- Keys must be unique; values can repeat.
- Very powerful when working with relationships.

Useful in Trigger Handlers when you need to update parent Food Order totals based on child Order Line Items.

Control Statements in Apex

Control statements in Apex determine the flow of execution in the program. They allow conditional execution and iteration over data.

- **If–Else:** Used to make decisions based on conditions.
- **For Loop:** Executes repeatedly for a fixed number of iterations or over a collection.

In our project:

We used if–else for checking order and complaint statuses, and for loops for iterating through records in triggers and handlers.

Exception Handling in Apex

Exception handling in Apex ensures that runtime errors don't break execution and are managed gracefully.

- **Try-Catch-Finally:** Used to catch and handle exceptions.
- **System.debug & Custom Messages:** Help identify and log issues.
- **Throw:** Used to raise custom exceptions.

In our project:

We applied exception handling in service classes and triggers to make sure errors (like invalid record updates) are handled properly without failing the transaction.

Test Classes

- Developer Console → File → New → Apex Class → Add @isTest Annotation`
- **Purpose:** Test classes validate that your Apex triggers and classes work as expected. Salesforce requires test coverage to deploy to production, and they also ensure your business logic is correct for different scenarios.

FoodOrderServiceTest

- **Purpose**: Tests the FoodOrderService class and FoodOrderTrigger.
- **Key Actions Tested:**

- Creating sample Restaurant, Food Order, Menu Item, and Order Line Item records.
- Verifying that order totals are correctly calculated when line items are inserted or updated.
- Updating food order status to ensure trigger logic runs properly.

FoodOrderServiceTest.apxc *

Code Coverage: None | API Version: 64

```apex
@isTest
public class FoodOrderServiceTest {

    @testSetup
    static void setupData(){
        // Create a Restaurant
        Restaurant__c res = new Restaurant__c(
            Name = 'Test Restaurant',
            Location__c = 'Test Area',
            Cuisine__c = 'Italian'
        );
        insert res;

        // Create a Food Order
        Food_Orders__c order = new Food_Orders__c(
            Status__c = 'Pending',
            Restaurant__c = res.Id,
            Total_Amount__c = 0
        );
        insert order;

        // Create Menu Item
        Menu_Item__c menu = new Menu_Item__c(
            Name = 'Pizza',
            Price__c = 100,
            Restaurant__c = res.Id
        );
        insert menu;

        // Create Order Line Item
        Order_Line_Item__c line = new Order_Line_Item__c(
            Food_Order__c = order.Id,
            Menu_Item__c = menu.Id,
            Quantity__c = 2,
            Unit_Price__c = 100
        );
        insert line;
    }

    static testMethod void testCalculateGrandTotal(){
        Food_Orders__c order = [SELECT Id, Total_Amount__c FROM Food_Orders__c LIMIT 1];

        // Call the service method
        Set<Id> orderIds = new Set<Id>{order.Id};
        FoodOrderService.calculateGrandTotal(orderIds);

        // Verify the total
        order = [SELECT Id, Total_Amount__c FROM Food_Orders__c WHERE Id = :order.Id];
        System.assertEquals(200, order.Total_Amount__c);
    }

    static testMethod void testFoodOrderTrigger(){
        Food_Orders__c order = [SELECT Id, Status__c, Total_Amount__c FROM Food_Orders__c LIMIT 1];

        // Update status to simulate trigger (no Delivery)
        order.Status__c = 'Processing';
        update order;

        // Verify totals remain correct
        Food_Orders__c updatedOrder = [SELECT Id, Total_Amount__c FROM Food_Orders__c WHERE Id = :order.Id];
        System.assertEquals(200, updatedOrder.Total_Amount__c);
    }
}
```

ComplaintHandlerTest

- **Purpose:** Tests ComplaintHandler and ComplaintTrigger.
- **Key Actions Tested:**
  - Creating sample Restaurant, Food Order, and Complaint records.
  - Updating complaint status to verify related Food Order status or notifications are handled correctly.

ComplaintHandlerTest.apxc *

Code Coverage: None API Version: 64

```
@isTest
public class ComplaintHandlerTest {
    @testSetup
    static void setupData(){
        // Restaurant
        Restaurant__c res = new Restaurant__c(Name='Test Res', Location__c='Area', Cuisine__c='Italian');
        insert res;
        // Food Order
        Food_Orders__c order = new Food_Orders__c(Status__c='Pending', Restaurant__c=res.Id, Total_Amount__c=0);
        insert order;
        // Complaint
        Complaint__c comp = new Complaint__c(Food_Order__c=order.Id, Status__c='Open');
        insert comp;
    }

    static testMethod void testUpdateOrderOnComplaint(){
        Complaint__c comp = [SELECT Id, Status__c, Food_Order__c FROM Complaint__c LIMIT 1];
        // Resolve complaint
        comp.Status__c = 'Resolved';
        update comp;
        // Verify Food Order status updated (if your handler does that)
        Food_Orders__c order = [SELECT Id, Status__c FROM Food_Orders__c WHERE Id = :comp.Food_Order__c];
        System.assertEquals('Delivered', order.Status__c);
    }
}
```

OrderLineItemHandlerTest

- **Purpose:** Tests OrderLineItemHandler and OrderLineItemTrigger.
- **Key Actions Tested:**
  - Creating sample Restaurant, Food Order, Menu Item, and Order Line Item records.
  - Updating order line item quantity to ensure Food Order totals are updated automatically.

OrderLineItemHandler.apxc *

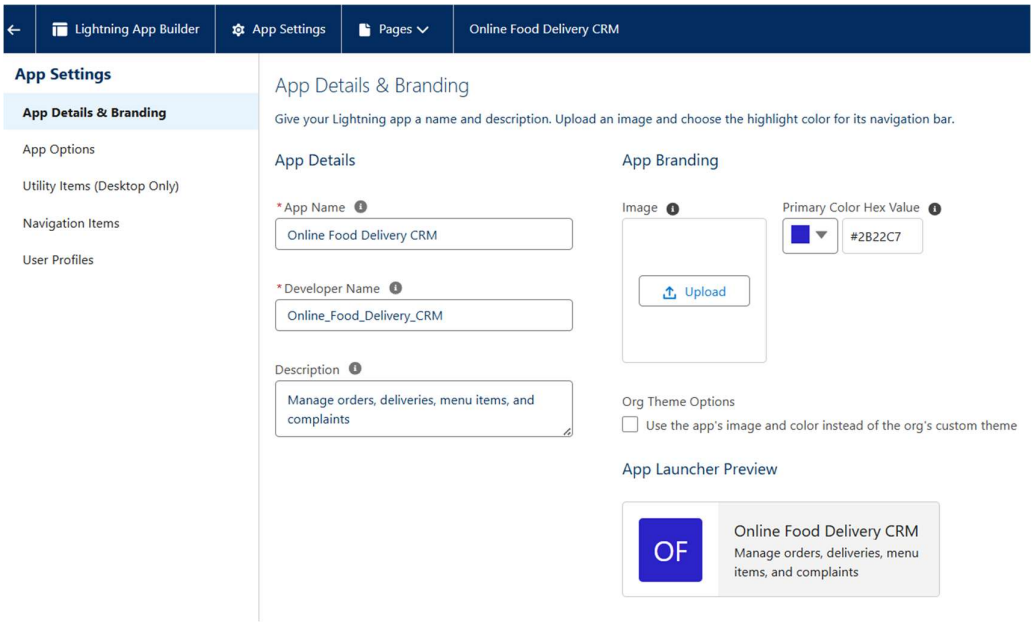
Code Coverage: None API Version: 64

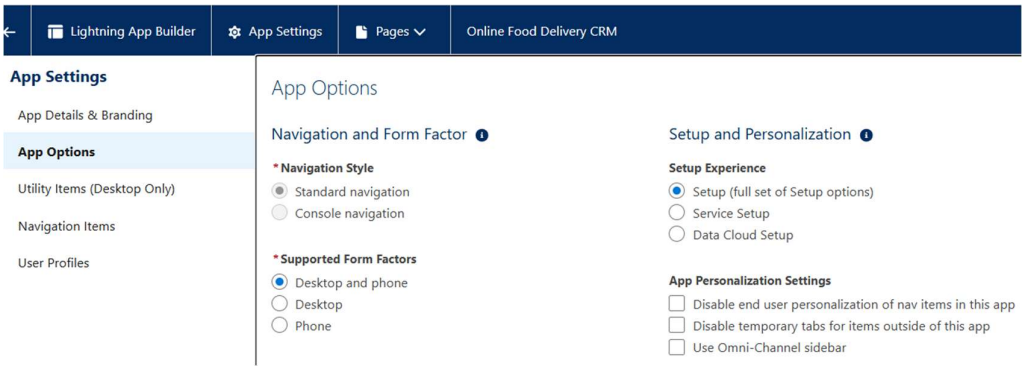
```
public class OrderLineItemHandler {
    public static void updateFoodOrderTotal(Map<Id, Order_Line_Item__c> newMap,
                                            Map<Id, Order_Line_Item__c> oldMap,
                                            Boolean isDelete) {
        Set<Id> foodOrderIds = new Set<Id>();

        if(isDelete){
            for(Order_Line_Item__c oldItem : oldMap.values()){
                if(oldItem.Food_Order__c != null)
                    foodOrderIds.add(oldItem.Food_Order__c);
            }
        } else {
            for(Order_Line_Item__c item : newMap.values()){
                if(item.Food_Order__c != null)
                    foodOrderIds.add(item.Food_Order__c);
            }
        }

        FoodOrderService.calculateGrandTotal(foodOrderIds);
    }
}
```

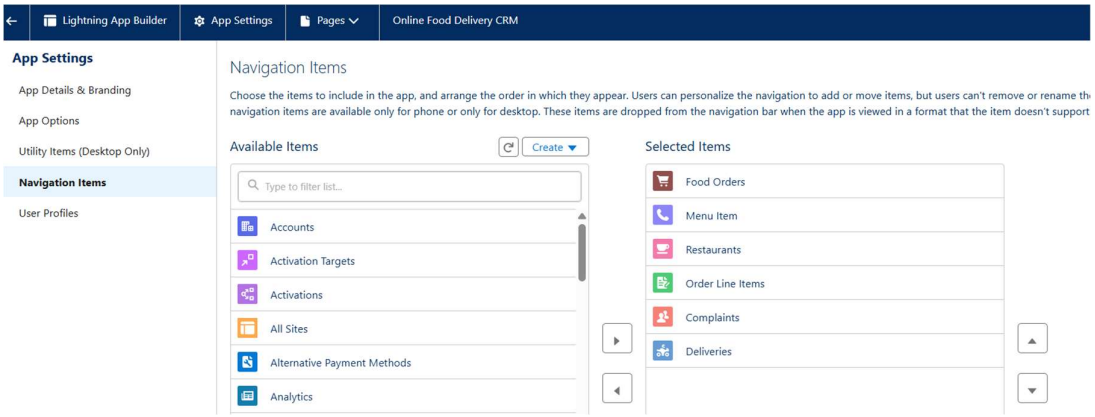
Phase 6: User Interface Development

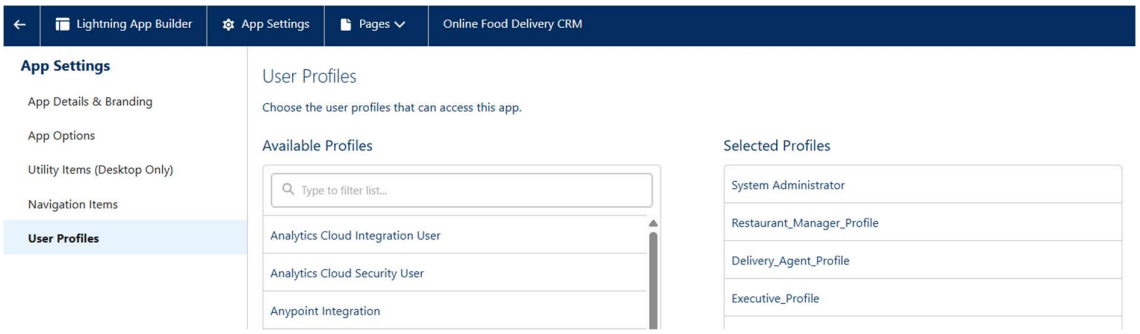
Lightning App Builder

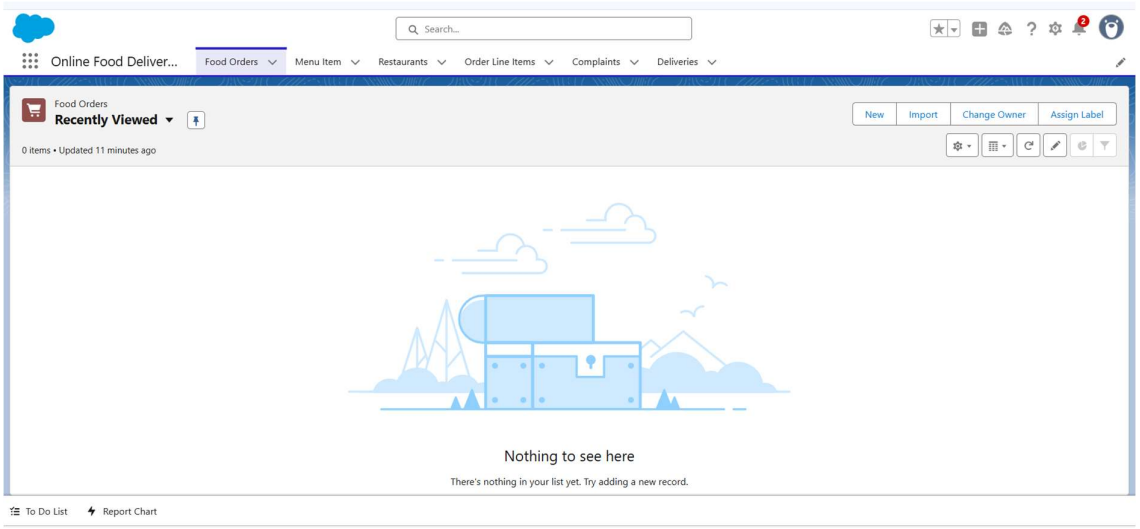
- Setup → App Manager → New Lightning App
- Used Lightning App Builder to create the Online Food Delivery CRM app.
- Added tabs for key objects: Food Orders, Order Line Items, Menu Items, Complaints, Restaurants.
- Configured navigation and page assignments for relevant profiles (Restaurant Manager, Delivery Agent, Executive).






Lightning App Builder
App Settings
Pages
Online Food Delivery CRM
App Settings
App Details & Branding
App Options
Utility Items (Desktop Only)
Navigation Items
User Profiles
Navigation Items
Choose the items to include in the app, and arrange the order in which they appear. Users can personalize the navigation to add or move items, but users can't remove or rename th
navigation items are available only for phone or only for desktop. These items are dropped from the navigation bar when the app is viewed in a format that the item doesn't support
Available Items
Create
Type to filter list...
Accounts
Activation Targets
Activations
All Sites
Alternative Payment Methods
Analytics
Selected Items
Food Orders
Menu Item
Restaurants
Order Line Items
Complaints
Deliveries


Lightning App Builder
App Settings
Pages
Online Food Delivery CRM
App Settings
App Details & Branding
App Options
Utility Items (Desktop Only)
Navigation Items
User Profiles
User Profiles
Choose the user profiles that can access this app.
Available Profiles
Type to filter list...
Analytics Cloud Integration User
Analytics Cloud Security User
Anypoint Integration
Selected Profiles
System Administrator
Restaurant_Manager_Profile
Delivery_Agent_Profile
Executive_Profile


Search...
Online Food Deliver...
Food Orders
Menu Item
Restaurants
Order Line Items
Complaints
Deliveries
Food Orders
Recently Viewed
0 items • Updated 11 minutes ago
New
Import
Change Owner
Assign Label
Nothing to see here
There's nothing in your list yet. Try adding a new record.
To Do List
Report Chart

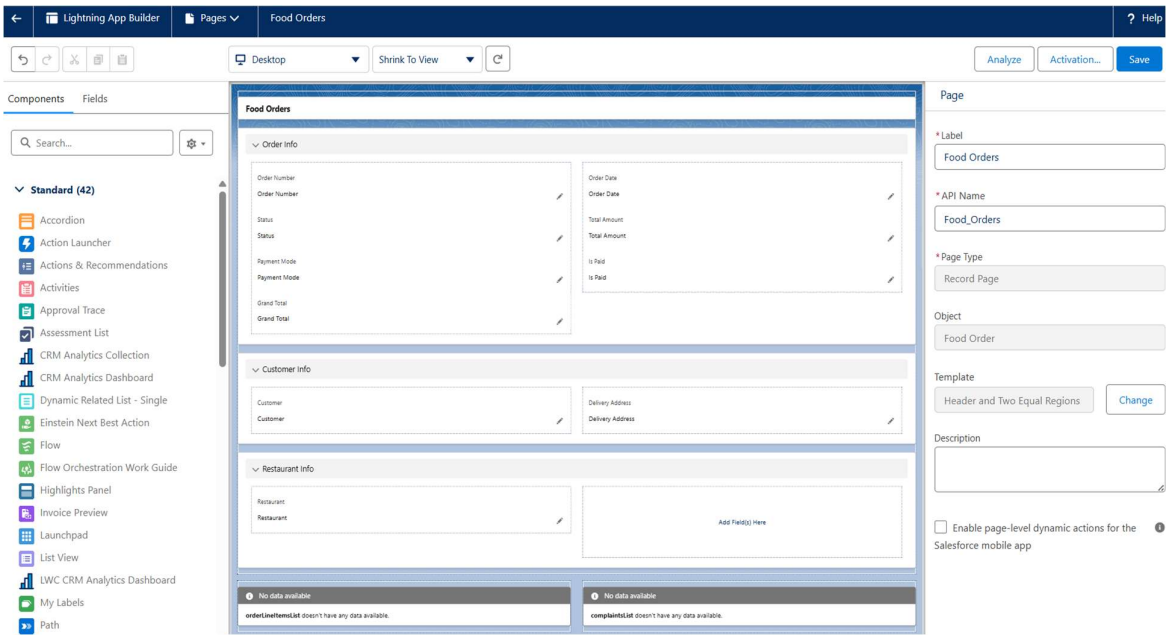
Record Pages

- Record Pages in Salesforce provide a customized view of individual records for each object.

Food Orders Record Page

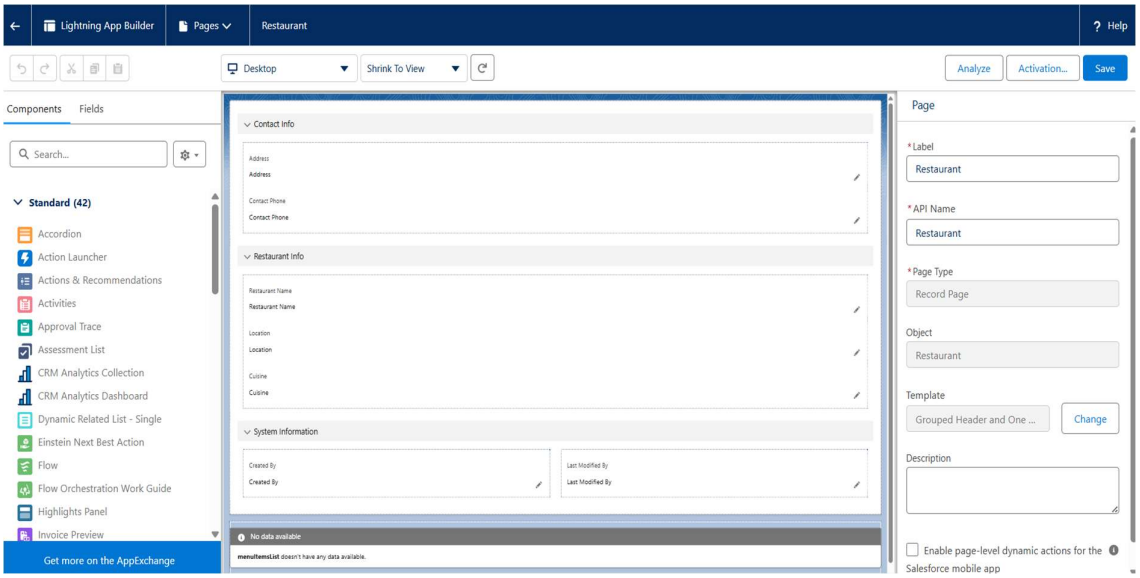
- Setup → Object Manager → Food_Orders__c → Lightning Record Pages → New
- Added Record Detail component to show Order Info and Customer Info.
- Added three LWCs to display:

1. **Food Order Summary** – key order details.
2. **Order Line Items List** – items in the order.
3. **Complaints List** – complaints related to the order.



Restaurant Record Page

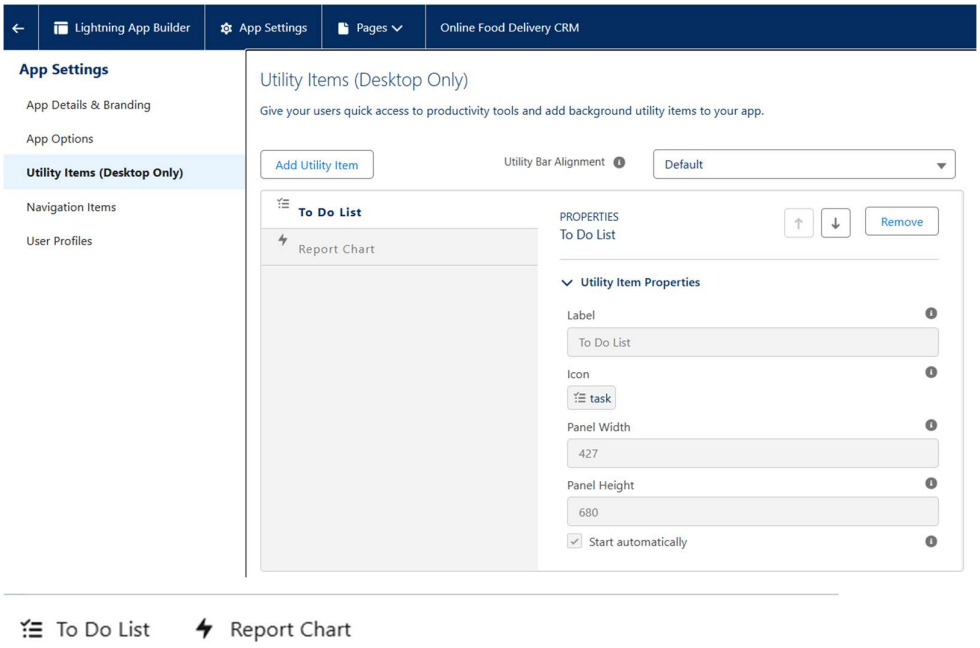
- Setup → Object Manager → Restaurant__c → Lightning Record Pages → New
- Added Record Detail component to show Restaurant details (Name, Location, Cuisine).
- Added one LWC to display: Menu Items List – menu items available for that restaurant.



Utility Bar

- Setup → App Manager → Food Delivery CRM App → Edit → Utility Bar
- Added To-Do List → for quick task management.
- Added Report Chart → to show key metrics like total orders or complaints.

Purpose: Provides users with quick access to frequently used tools, improving productivity and navigation.



Lightning Web Components (LWC)

Purpose: LWCs provide a modern, responsive, and reusable UI for displaying and interacting with Salesforce data in the Food Delivery CRM app.

1. **Food Order Summary**
    - Displays a list of food orders with key details like name and total amount.
    - Added to the Food Orders Record Page.

foodOrderSummary.html

force-app > main > default > lwc > foodOrderSummary > foodOrderSummary.html > ...

```
<template>
    <lightning-card title="Food Orders">
        <template if:true={orders}>
            <ul>
                <template for:each={orders} for:item="order">
                    <li key={order.Id}>
                        {order.Name} - ₹{order.Total_Amount__c}
                    </li>
                </template>
            </ul>
        </template>
        <template if:false={orders}>
            <p>No orders available.</p>
        </template>
    </lightning-card>
</template>
```

foodOrderSummary.js

force-app > main > default > lwc > foodOrderSummary > foodOrderSummary.js > ...

```
import { LightningElement, wire } from 'lwc';
import getFoodOrders from '@salesforce/apex/FoodOrderController.getFoodOrders';

export default class FoodOrderSummary extends LightningElement {
    orders; // stores the food orders
    error;  // stores errors if any

    @wire(getFoodOrders)
    wiredOrders({ error, data }) {
        if (data) {
            this.orders = data;
            this.error = undefined;
        } else if (error) {
            this.error = error;
            this.orders = undefined;
        }
    }
}
```

foodOrderSummary.js-meta.xml

force-app > main > default > lwc > foodOrderSummary > foodOrderSummary.js-meta.xml > ...

```xml
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>64.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
    </targets>
</LightningComponentBundle>
```

2. Order Line Items List

- Shows items (menu items, quantity, unit price) linked to a food order.
- Added to the Food Orders Record Page.

orderLineItemsList.html

force-app > main > default > lwc > orderLineItemsList > orderLineItemsList.html > ...

```html
<template>
    <lightning-card title="Order Line Items">
        <template if:true={lineItems.length}>
            <ul>
                <template for:each={lineItems} for:item="item">
                    <li key={item.Id}>
                        {item.Menu_Item__r.Name} - Qty: {item.Quantity__c} - ₹{item.Total__c}
                    </li>
                </template>
            </ul>
        </template>
        <template if:false={lineItems.length}>
            <p>No line items found.</p>
        </template>
    </lightning-card>
</template>
```

orderLineItemsList.js

force-app > main > default > lwc > orderLineItemsList > orderLineItemsList.js > ...

```js
import { LightningElement, api, wire } from 'lwc';
import getOrderLineItems from '@salesforce/apex/OrderLineItemController.getOrderLineItems';

export default class OrderLineItemsList extends LightningElement {
    @api recordId;
    lineItems;

    @wire(getOrderLineItems, { orderId: '$recordId' })
    wiredItems({ error, data }) {
        if(data){
            this.lineItems = data;
        } else if(error){
            this.error = error;
        }
    }
}
```

orderLineItemsList.js-meta.xml

force-app > main > default > lwc > orderLineItemsList > orderLineItemsList.js-meta.xml > ...

```xml
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>64.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
    </targets>
</LightningComponentBundle>
```

3. Complaints List

- Displays complaints related to a food order.
- Added to the Food Orders Record Page.

complaintsList.html

force-app > main > default > lwc > complaintsList > complaintsList.html > ...

```html
<template>
    <lightning-card title="Complaints">
        <template if:true={complaints.length}>
            <ul>
                <template for:each={complaints} for:item="c">
                    <li key={c.Id}>
                        {c.Complaint_Number__c} - {c.Status__c} ({c.Complaint_Type__c})
                    </li>
                </template>
            </ul>
        </template>
        <template if:false={complaints.length}>
            <p>No complaints found.</p>
        </template>
    </lightning-card>
</template>
```

complaintsList.js

force-app > main > default > lwc > complaintsList > complaintsList.js > ...

```js
import { LightningElement, api, wire } from 'lwc';
import getComplaints from '@salesforce/apex/ComplaintController.getComplaints';

export default class ComplaintsList extends LightningElement {
    @api recordId; // Food Order Id
    complaints;

    @wire(getComplaints, { orderId: '$recordId' })
    wiredComplaints({ error, data }) {
        if(data){
            this.complaints = data;
        } else if(error){
            this.error = error;
        }
    }
}
```

complaintsList.js-meta.xml

force-app > main > default > lwc > complaintsList > complaintsList.js-meta.xml > ...

```xml
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>64.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
    </targets>
</LightningComponentBundle>
```

4. Menu Items List

- Shows all menu items available for a restaurant.
- Added to the Restaurant Record Page.

menuItemsList.html

force-app > main > default > lwc > menuItemsList > menuItemsList.html > ...

```html
<template>
    <lightning-card title="Menu Items">
        <template if:true={menuItems.length}>
            <ul>
                <template for:each={menuItems} for:item="menu">
                    <li key={menu.Id}>
                        {menu.Name} - ₹{menu.Price__c} ({menu.Category__c})
                    </li>
                </template>
            </ul>
        </template>
        <template if:false={menuItems.length}>
            <p>No menu items found.</p>
        </template>
    </lightning-card>
</template>
```

menuItemsList.js

force-app > main > default > lwc > menuItemsList > menuItemsList.js > ...

```js
import { LightningElement, api, wire } from 'lwc';
import getMenuItems from '@salesforce/apex/MenuItemController.getMenuItems';

export default class MenuItemsList extends LightningElement {
    @api recordId; // Restaurant Id
    menuItems;

    @wire(getMenuItems, { restaurantId: '$recordId' })
    wiredItems({ error, data }) {
        if(data){
            this.menuItems = data;
        } else if(error){
            this.error = error;
        }
    }
}
```

menuItemsList.js-meta.xml

force-app > main > default > lwc > menuItemsList > menuItemsList.js-meta.xml > ...

```xml
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>64.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
    </targets>
</LightningComponentBundle>
```

Apex with LWC

Purpose:

Apex controllers provide backend logic for LWCs. They handle SOQL queries, apply business logic, and return data to components. This allows LWCs to display dynamic Salesforce data.

1. **FoodOrderController**
    - **Method:** getFoodOrders()
    - Returns the latest Food Orders with Id, Name, and Total Amount.
    - Used by Food Order Summary LWC.

FoodOrderController.apxc *

Code Coverage: None | API Version: 64

```
public with sharing class FoodOrderController {
    @AuraEnabled(cacheable=true)
    public static List<Food_Orders__c> getFoodOrders() {
        return [SELECT Id, Name, Total_Amount__c FROM Food_Orders__c LIMIT 10];
    }
```

2. **OrderLineItemController**
    - **Method:** getOrderLineItems(Id orderId)
    - Returns items linked to a specific Food Order.
    - Used by Order Line Items LWC.

OrderLineItemController.apxc

Code Coverage: None | API Version: 64

```
public with sharing class OrderLineItemController {
    @AuraEnabled(cacheable=true)
    public static List<Order_Line_Item__c> getOrderLineItems(Id orderId) {
        return [SELECT Id, Quantity__c, Total__c, Menu_Item__r.Name
                FROM Order_Line_Item__c
                WHERE Food_Order__c = :orderId];
    }
}
```

3. **ComplaintController**
    - **Method:** getComplaints(Id orderId)
    - Returns complaints raised for a given Food Order.
    - Used by Complaints List LWC.

ComplaintController.apxc

Code Coverage: None | API Version: 64

```
public with sharing class ComplaintController {
    @AuraEnabled(cacheable=true)
    public static List<Complaint__c> getComplaints(Id orderId) {
        return [SELECT Id, Name, Status__c, Complaint_Type__c
                FROM Complaint__c
                WHERE Food_Order__c = :orderId];
    }
}
```

4. **MenuItemController**
    - **Method:** getMenuItems(Id restaurantId)
    - Returns all menu items for a restaurant.
    - Used by Menu Items List LWC.

MenuItemController.apxc

Code Coverage: None | API Version: 64

```
public with sharing class MenuItemController {
    @AuraEnabled(cacheable=true)
    public static List<Menu_Item__c> getMenuItems(Id restaurantId) {
        return [SELECT Id, Name, Price__c, Category__c
                FROM Menu_Item__c
                WHERE Restaurant__c = :restaurantId];
    }
}
```

Wire Adapters

- **Purpose:** Provide a reactive way to fetch Salesforce data directly into LWCs without manually calling Apex.
- Used @wire with getFoodOrders in Food Order Summary LWC.
- Orders list auto-refreshes whenever data changes in Salesforce.

Imperative Apex Calls

- **Purpose:** Used for on-demand server calls triggered by user actions (button clicks, form submissions).

- Example: When a user clicks “Place Order”, an imperative call to an Apex method can insert a Food_Orders__c record and return confirmation.
- Allows handling logic like validation and conditional record creation.

Navigation Service

- **Purpose:** Lets LWCs programmatically navigate users to record pages, list views, or other Lightning pages.
- After placing a Food Order, the user can be redirected to that Food Order Record Page.
- Ensures smooth user flow from creation → viewing order details.

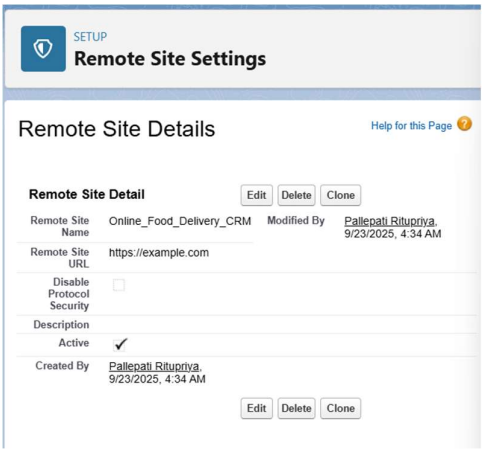
.....

Phase 7: Integration & External Access

Although external integrations are not fully implemented in this project, Salesforce provides powerful features to connect with outside systems. For the Food Delivery CRM, we have prepared the environment for future integrations.

Remote Site Settings

- Setup → Security → Remote Site Settings → New.
- Added a sample external URL (e.g., https://example.com) to allow Salesforce to make callouts.
- **Purpose:** Ensures Salesforce can securely communicate with external APIs like payment gateways or delivery tracking systems.



Future Scope (Not Implemented)

- Named Credentials – to securely store authentication details for external services.
- REST Callouts – to fetch delivery partner status or payment confirmation.
- External Services / Salesforce Connect – to link with external databases or microservices.

Purpose / Outcome

- Prepares the CRM for real-world integration with payment systems, Google Maps, or delivery tracking.
- Provides a pathway for future scalability without changing the core application.

.....

Phase 8: Data Management & Deployment

Data Import Wizard

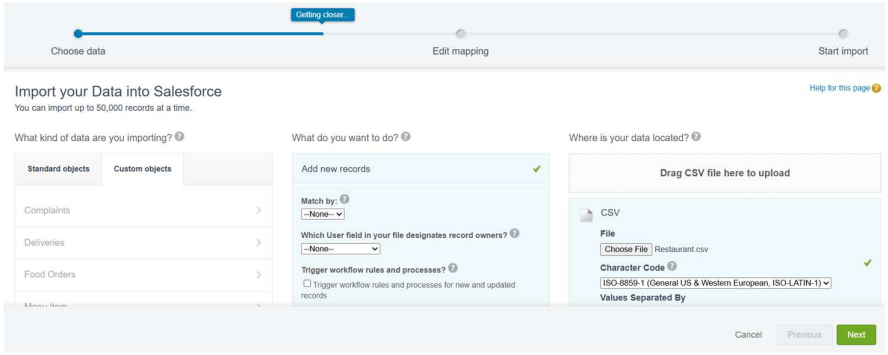
Purpose:

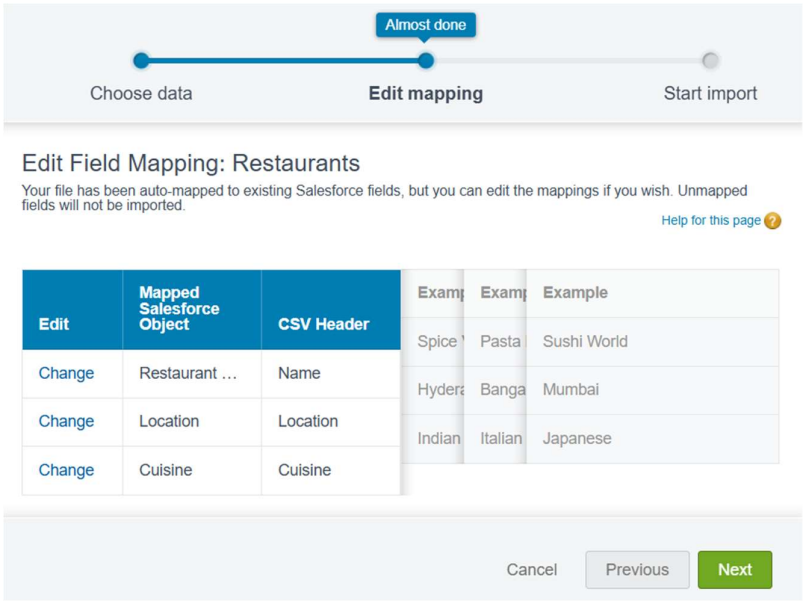
- To quickly import small datasets (like Restaurants, Menu Items, Food Orders) into Salesforce without using external tools.
- Helps in setting up demo data for testing your flows, LWCs, triggers, and reports.

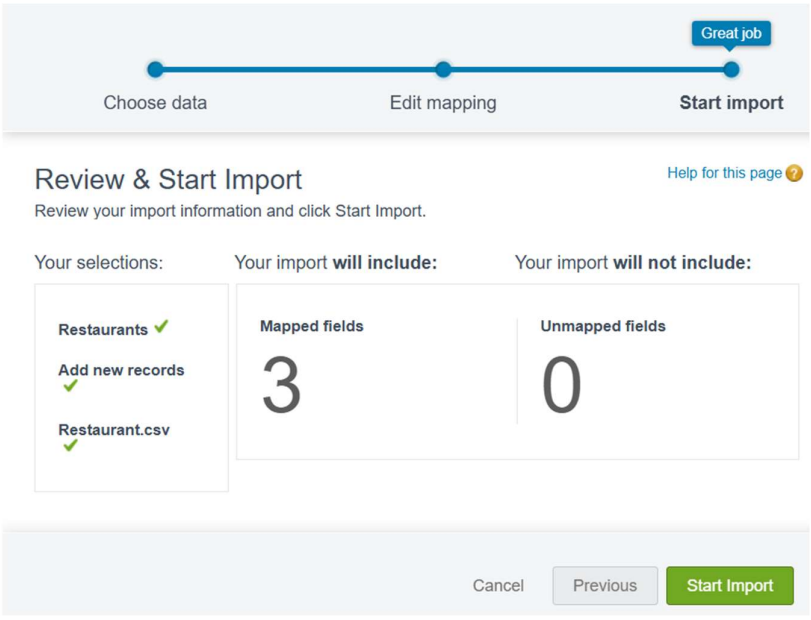
Steps:

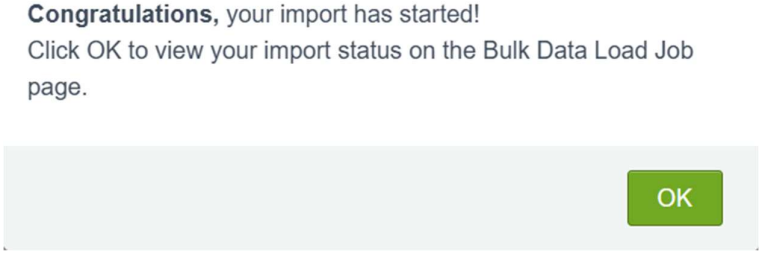
1. Setup → Data → Data Import Wizard → Launch Wizard.
2. Uploaded CSV file(Restaurant.csv) containing the records.
3. Map the CSV columns to Salesforce fields.

Object: **Restaurant__c**

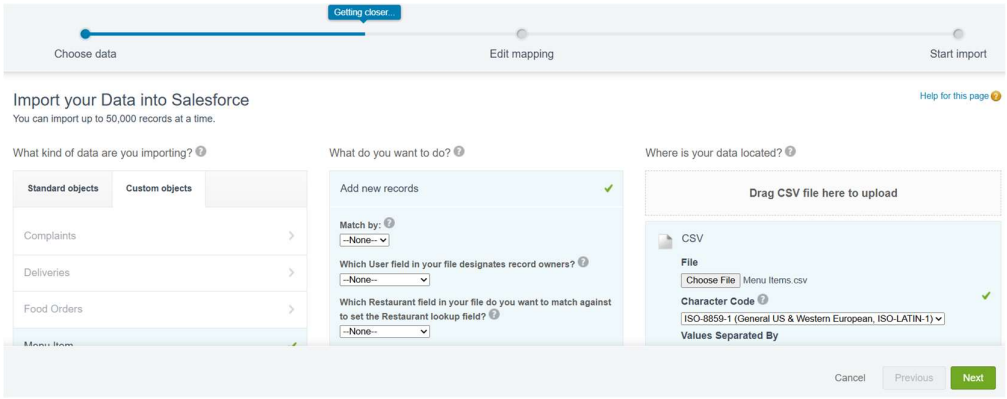


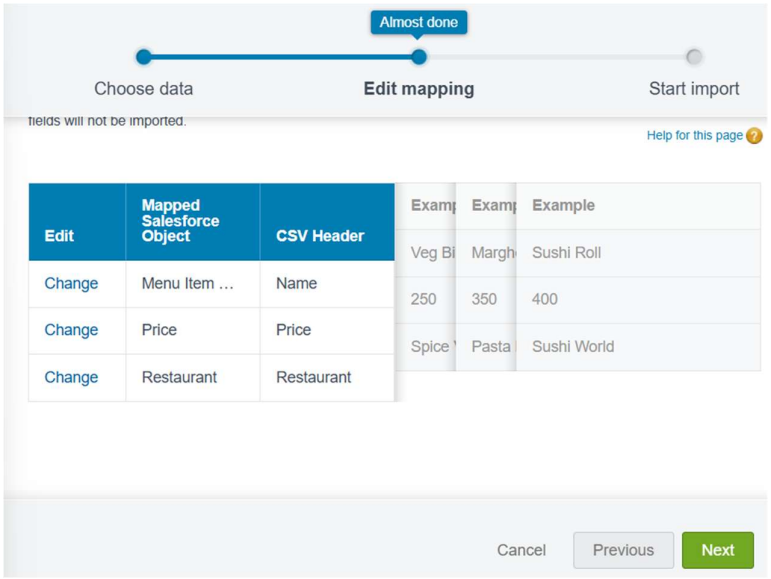

Almost done
Choose data
Edit mapping
Start import
Edit Field Mapping: Restaurants
Your file has been auto-mapped to existing Salesforce fields, but you can edit the mappings if you wish. Unmapped fields will not be imported.
Help for this page
Edit
Mapped Salesforce Object
CSV Header
Example
Change
Restaurant …
Name
Sushi World
Change
Location
Location
Mumbai
Change
Cuisine
Cuisine
Japanese
Cancel
Previous
Next

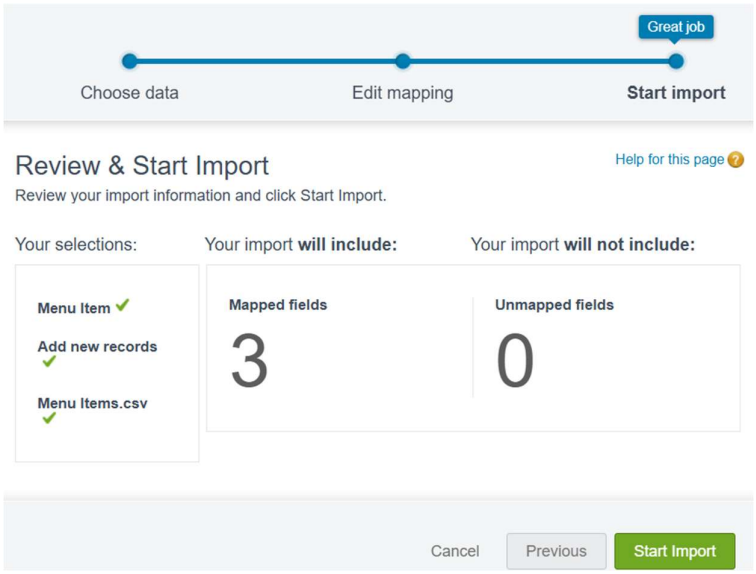

Great job
Choose data
Edit mapping
Start import
Review & Start Import
Help for this page
Review your import information and click Start Import.
Your selections:
Your import will include:
Your import will not include:
Restaurants
Add new records
Restaurant.csv
Mapped fields
3
Unmapped fields
0
Cancel
Previous
Start Import


Congratulations, your import has started!
Click OK to view your import status on the Bulk Data Load Job page.
OK

Object: Menu_Item__c







Duplicate Rules

- Setup → Duplicate Rules → New Rule → Object
- Duplicate Rules in Salesforce ensure data quality by preventing users from creating or importing duplicate records.

Restaurant Duplicate Rule

- Prevents the same restaurant (with same name and location) from being entered twice.
- **Matching Criteria:**
  - Restaurant Name → Exact Match
  - Location → Exact Match

VS Code & SFDX

- VS Code with Salesforce DX (SFDX) allows developers to retrieve, edit, and deploy Salesforce metadata in a local development environment. It is particularly useful for version control, collaborative development, and deploying changes to different orgs.

Package.xml

- Acts as a manifest listing Salesforce metadata to retrieve or deploy, enabling VS Code & SFDX to move components between orgs efficiently.

package.xml

manifest > package.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>*</members>
        <name>ApexClass</name>
    </types>
    <types>
        <members>*</members>
        <name>ApexTrigger</name>
    </types>
    <types>
        <members>*</members>
        <name>CustomObject</name>
    </types>
    <types>
        <members>*</members>
        <name>CustomField</name>
    </types>
    <types>
        <members>*</members>
        <name>Flow</name>
    </types>
    <types>
        <members>*</members>
        <name>EmailTemplate</name>
    </types>
    <types>
        <members>*</members>
        <name>PermissionSet</name>
    </types>
    <types>
        <members>*</members>
        <name>CustomApplication</name>
    </types>
    <version>59.0</version>
</Package>
```

Terminal

```
PS C:\Users\ritup\OneDrive\Documents\OnlineFoodDelivery\Online Food Delivery CRM> sf project retrieve start --mani
fest manifest/package.xml
>>
 »   Warning: @salesforce/cli update available from 2.104.6 to 2.106.6.

 Retrieving v59.0 metadata from ritupriya.p176698@agentforce.com using the v64.0 SOAP API

 - [4/4] Done 0ms

 Status: Succeeded

 ———————— Retrieving Metadata ————————

 Retrieving v59.0 metadata from ritupriya.p176698@agentforce.com using the v64.0 SOAP API

 ✓ Preparing retrieve request 10ms
 ✓ Sending request to org 505ms
 ✓ Waiting for the org to respond 6.57s
 ✓ Done 0ms

 Status: Succeeded
 Elapsed Time: 7.04s
```

.....

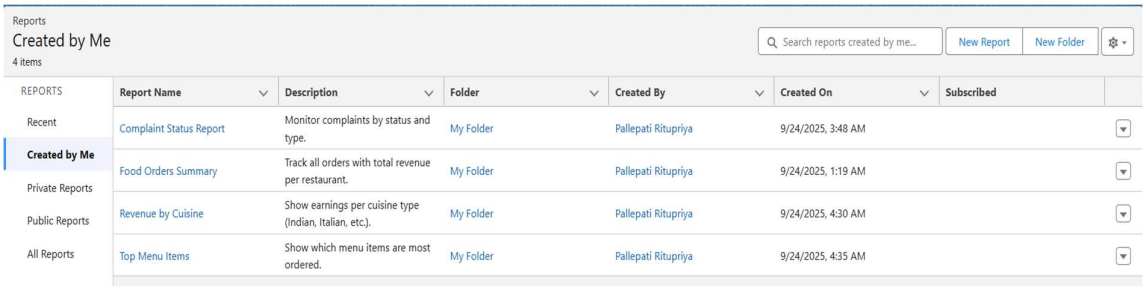
Phase 9: Reporting, Dashboards & Security Review

Reports

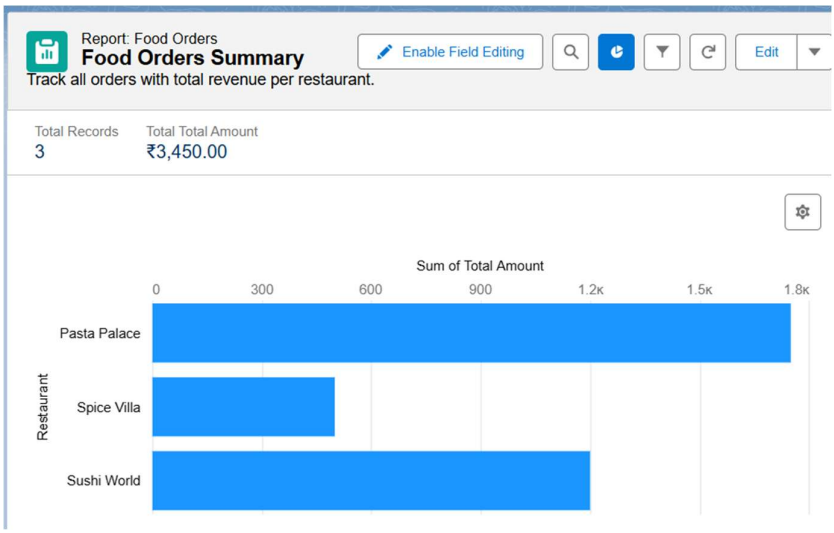
- Monitor business activity, track orders, revenue, and complaints.

Steps:

- **Reports → New Report → Select object/report type.**
- Apply filters and choose columns.
- Group rows, summarize numeric fields.
- Add charts
- Save to public folder.

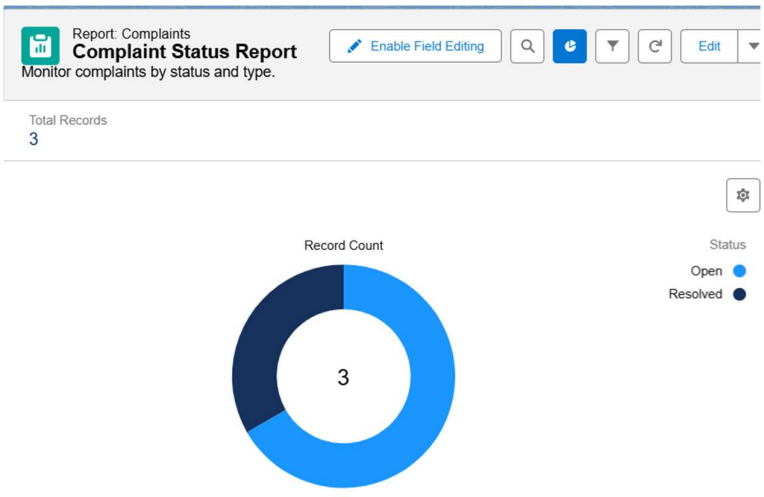


1. **Food Orders Summary** – Track all orders with total revenue per restaurant.
   - Shows total revenue per restaurant; grouped by Restaurant Name, with bar chart.



Restaurant	Food Order: Order Number	Customer	Total Amount	Status
Pasta Palace (1)	ORD-0002	-	₹1,750.00	Out for Delivery
Subtotal			₹1,750.00	
Spice Villa (1)	ORD-0001	-	₹500.00	Delivered
Subtotal			₹500.00	
Sushi World (1)	ORD-0003	-	₹1,200.00	Accepted
Subtotal			₹1,200.00	
Total (3)			₹3,450.00	

2. **Complaint Status** – Tracks complaints by status; grouped by Status, with pie chart.

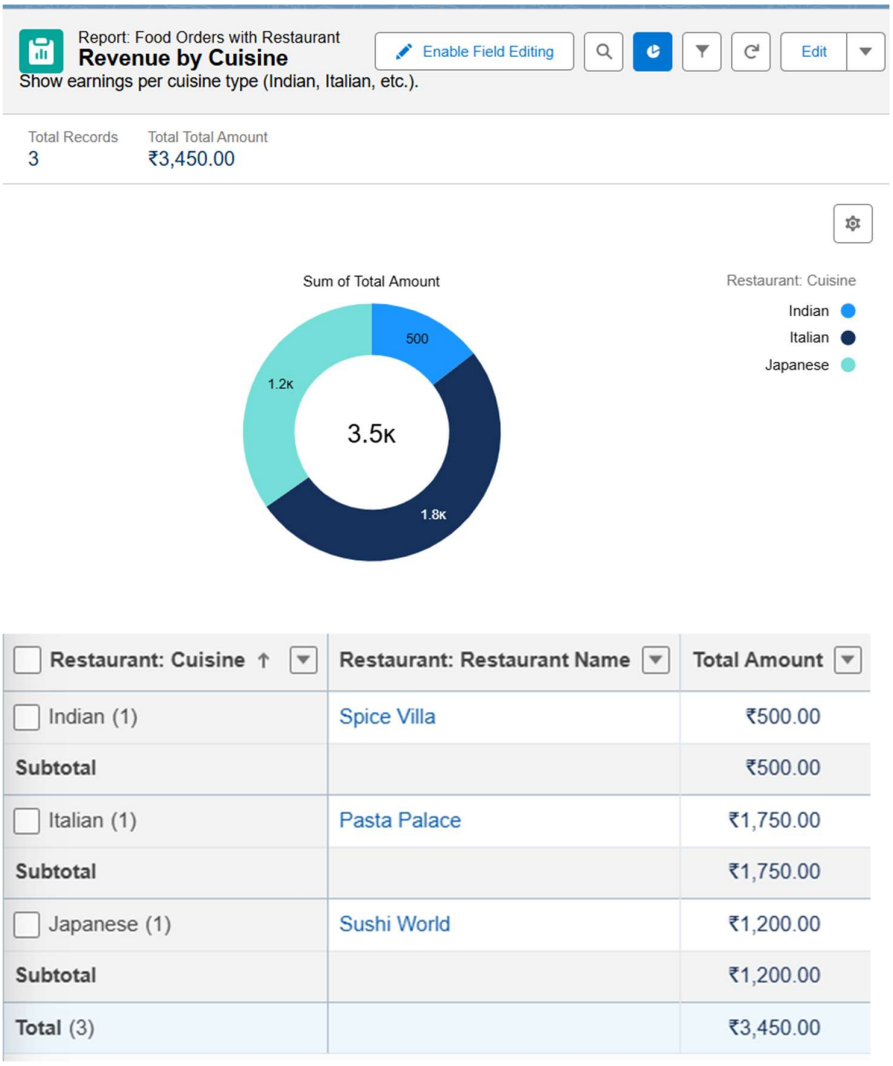


Status	Complaint: Complaint Number	Food Order	Complaint Type	Complaint: Created Date
Open (2)	CMP-0001	ORD-0001	Late Delivery	9/24/2025
	CMP-0003	ORD-0003	Quality Issue	9/24/2025
Subtotal				
Resolved (1)	CMP-0002	ORD-0002	Wrong Item	9/24/2025
Subtotal				
Total (3)				

3. **Top Menu Items** – Shows most ordered items; grouped by Menu Item, with horizontal bar chart.

Menu Item: Menu Item Name	Restaurant	Quantity	Total Amount
Margherita (1)	Pasta Palace	5	₹1,750.00
Subtotal		5	₹1,750.00
Sushi Roll (1)	Sushi World	3	₹1,200.00
Subtotal		3	₹1,200.00
Veg Biryani (1)	Spice Villa	2	₹500.00
Subtotal		2	₹500.00
Total (3)		10	₹3,450.00

4. **Revenue by Cuisine** – Shows earnings per cuisine; grouped by Cuisine, with pie chart



Restaurant: Cuisine ↑	Restaurant: Restaurant Name	Total Amount
Indian (1)	Spice Villa	₹500.00
Subtotal		₹500.00
Italian (1)	Pasta Palace	₹1,750.00
Subtotal		₹1,750.00
Japanese (1)	Sushi World	₹1,200.00
Subtotal		₹1,200.00
Total (3)		₹3,450.00

Dashboards

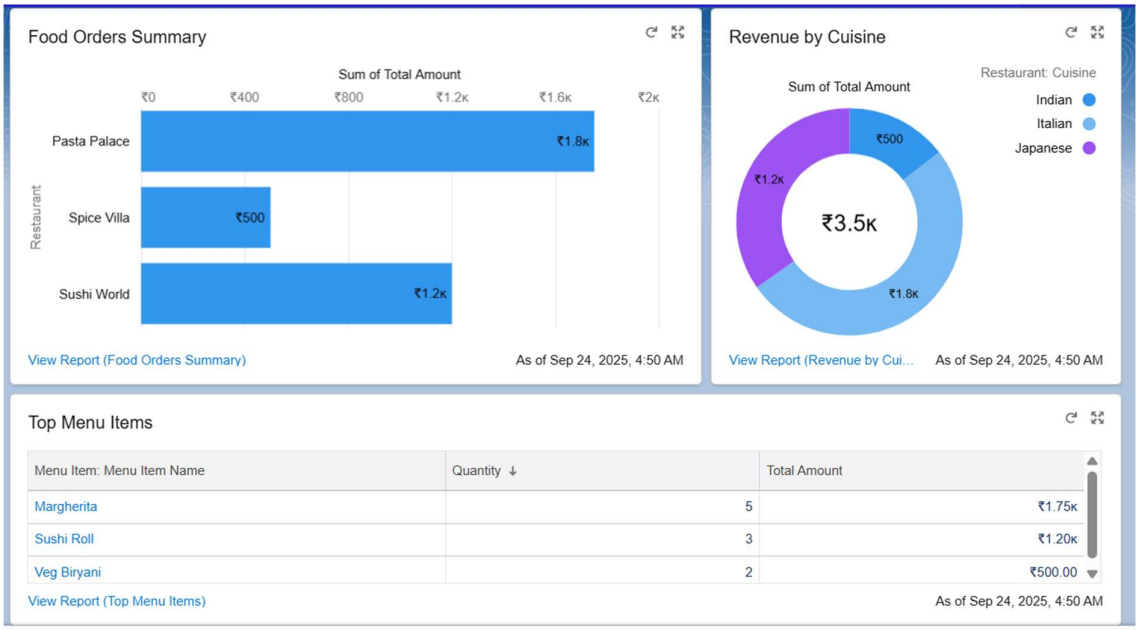
- Visualize key metrics for quick insights.

Steps:

- **Dashboards → New Dashboard**.
- Add components (charts, tables) using saved reports.
- Configure filters and display options.
- Save & run in a public folder.

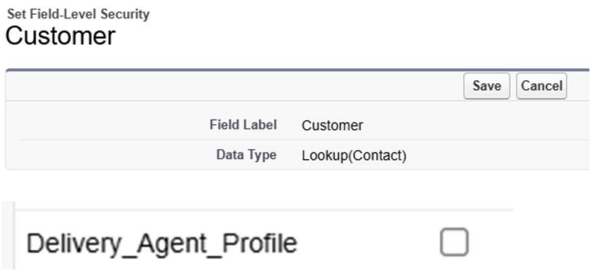
Restaurant Business Overview

Provides a quick view of overall restaurant performance, including total orders, revenue, top menu items, and complaints status.



Field-Level Security

- Setup → Object Manager → Food_Orders__c → Fields & Relationships → Select Customer__c
- Click Set Field-Level Security → Uncheck Visible for Delivery Agent Profile → Save changes



Login IP Ranges

- Setup → Users → Profiles → Delivery Agent Profile
- Scroll to Login IP Ranges section → Click New
- Enter:
    - **Start IP Address** → e.g., 203.0.113.1 (your office network start)
    - **End IP Address** → e.g., 203.0.113.255 (your office network end)
- Save changes

Phase 10: Final Presentation & Demo Day

- In this final phase, I showcased my Salesforce project **Online Food Delivery CRM** in a live demo.
- The system simplifies food ordering and restaurant management by providing:
  - **Custom Lightning Application** – Centralized app for managing restaurants, orders, and complaints
  - **Food Order & Menu Management** – Easy creation and tracking of orders and menu items
  - **Role-Based Access** – Secure access for Restaurant Manager, Delivery Agent, and Executive roles
  - **Complaint Handling** – Tracks customer complaints and updates order status automatically
  - **Data Management** – Import, export, and maintain restaurant, menu, and order data efficiently using Salesforce tools like Data Import Wizard.
  - **Automated Notifications** – Email alerts for order confirmation and complaint resolution
  - **Reports for Analysis** – Detailed reports on orders, complaints, and top-selling menu items
  - **Dashboards** – Real-time insights through visual dashboards for restaurant performance and revenue.

The project demonstrates how Salesforce can streamline restaurant workflows, improve operational efficiency, and provide actionable business insights.

Project PPT: https://docs.google.com/presentation/d/1xwLux-4_Jmjat2mKe13E3LiLrOidkAWq/edit?usp=sharing&ouid=113298212239433324406&rtpof=true&sd=true

Project Video:
https://drive.google.com/file/d/1cO3LnHkoLl6ZyOulSv6hFVrFy82F_2jI/view?usp=drive_link