Project Title: Mediswift - Hyperlocal Medicine Delivery CRM

Phase 1: Problem Understanding & Industry Analysis

➢ Problem Statement:

In the modern healthcare ecosystem, the demand for convenient, reliable, and timely access to medicines has grown significantly. Traditional pharmacy management processes often involve manual order handling, inventory tracking, and delivery coordination, which lead to inefficiencies, delays, and poor customer experience. Existing systems like online pharmacy apps (e.g., PharmEasy or 1mg) solve this problem for customers but are complex and customer-facing, requiring significant backend automation and integrations.

However, for small or mid-sized pharmacy networks or healthcare administrators, there is a lack of a centralized Salesforce-based management system to handle medicine inventory, pharmacy records, order tracking, payment management, and delivery operations — all in one platform.

The absence of an integrated solution results in challenges such as:

- Manual tracking of orders and deliveries
- Lack of visibility into pharmacy performance and medicine stock levels
- Difficulty in monitoring payments and delivery partner status
- Poor coordination between pharmacies and delivery staff

Hence, there is a strong need for a **Salesforce-based Pharmacy Management System (MediSwift)** that automates end-to-end operations — from order creation to delivery — while maintaining centralized control, data consistency, and analytical visibility for pharmacy administrators.

➢ Stakeholder Analysis:

Role	Description
System Administrator	Manages all users, data, and automation in Salesforce. Responsible for configuration, access control, and reports.
Pharmacy Manager	Handles pharmacy registration, medicine catalog, and stock management.
Delivery Partner	Delivers medicine orders, updates delivery status.
Customer (Record Only)	Represents end-users placing orders (not system users).

Salesforce Platform	Backend environment for managing all data and automation.

➢ Business Process Mapping:

1. **Customer Record Creation**
   - Admin records new customers in the *Customer* object (no login access).
   - Captures contact details, address, and location for order delivery.
2. **Pharmacy Onboarding**
   - Admin/Pharmacy Manager adds pharmacy details — license, contact info, and city.
   - Each pharmacy is linked with its available medicines in the *Medicine* object.
3. **Medicine Management**
   - Pharmacy Manager maintains stock levels, price, expiry, and prescription requirements.
   - Medicines are categorized (e.g., Antibiotic, Painkiller, Supplement) using picklists.
4. **Order Creation**
   - Admin creates an *Order* record for a specific customer and pharmacy.
   - Adds order line items (*Order Items*) for each medicine selected.
   - The *Total Amount* is automatically calculated from *Order Items*.
5. **Order Processing**
   - The order moves through various statuses: Pending → Confirmed → Packed → Out for Delivery → Delivered / Cancelled.
   - Automated workflows update statuses and notify relevant stakeholders.
6. **Delivery Assignment**
   - An available *Delivery Partner* is assigned to the order.
   - Partner details and current status (Available, Busy, Offline) are tracked.
   - Once the delivery is complete, the status changes to Delivered.
7. **Payment Processing**
   - Payment record is created automatically or manually linked to an order.
   - Captures *Payment Mode*, *Amount*, *Status*, and *Transaction ID*.
   - Status options: Pending, Paid, Failed.
8. **Post-Delivery Updates**
   - Order status updates to *Delivered*.
   - Delivery Partner's total deliveries increase.
   - Customer satisfaction/rating data can be logged later for reporting.

➢ AppExchange Exploration:

AppExchange was explored to identify ready-to-use Salesforce apps that could enhance MediSwift's functionality for pharmacy management, automation, and analytics.

App Name	Purpose / Benefit
Health Cloud	Reference for healthcare data model and compliance.
Conga Composer	Generate invoices and order reports automatically.
SMS-Magic Interact	Send SMS alerts for order and delivery status.
Mailchimp for Salesforce	Email notifications for pharmacy or order updates.
Tableau CRM (Einstein Analytics)	Advanced dashboards and performance analytics.
DocuSign	Digital signatures for pharmacy license verification.

Phase 2: Org Setup & Configuration

➢ Salesforce Editions:

Edition Used: Salesforce Developer Edition (Free)

The **Developer Edition** provides a fully functional Salesforce environment with:

- Access to Salesforce CRM Core features (Leads, Accounts, Contacts, Opportunities, etc.).
- Customization tools like Objects, Fields, Flows, Validation Rules, and Reports.
- Apex and Lightning Components development capability.
- API access for integrations.

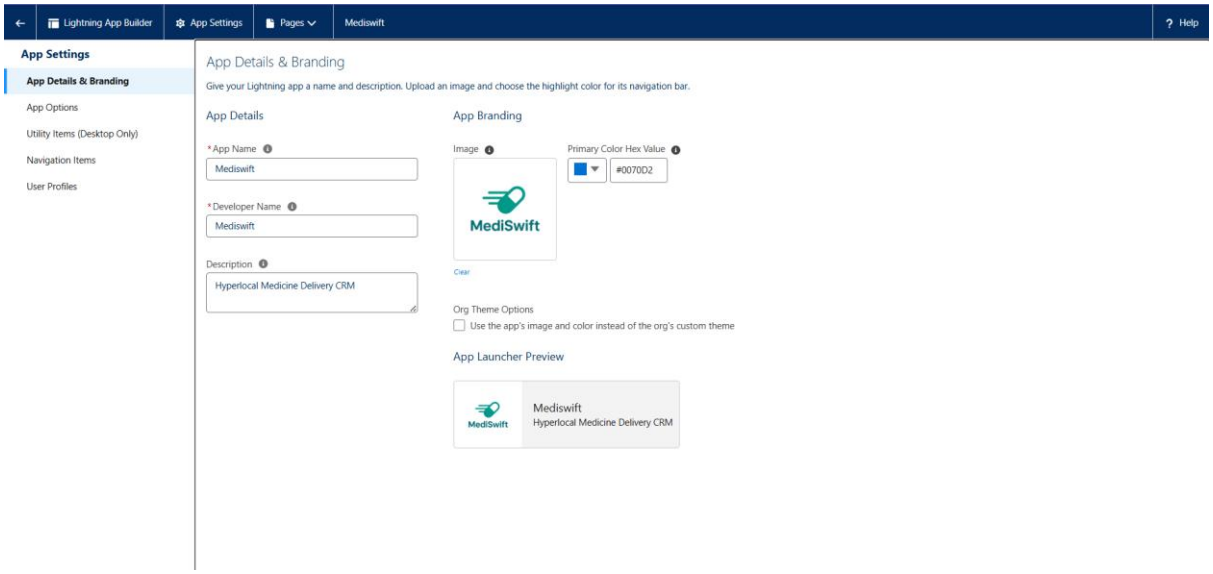
➢ Lightning App Creation – MediSwift

Objective: To create a dedicated Lightning App named *MediSwift* for managing medicines, pharmacies, orders, and deliveries efficiently within Salesforce.

Steps:

1. Open App Manager:
   Go to Setup → App Manager → New Lightning App.
2. App Details:
   Enter App Name: *MediSwift*, add a short description, and upload a logo if needed.

3. Navigation Settings:
   Choose Standard Navigation and continue.
4. Add Items (Tabs):
   Add custom and standard objects such as:
   - Customer
   - Medicine
   - Order
   - Delivery
   - Reports / Dashboards
5. Assign Profiles:
   Give access to specific profiles (e.g., *System Administrator*, *Sales User*).
6. Finish and Verify:
   Click Save & Finish, then open the App Launcher → search for *MediSwift* → verify all tabs and layouts.



➢ Company Profile Setup

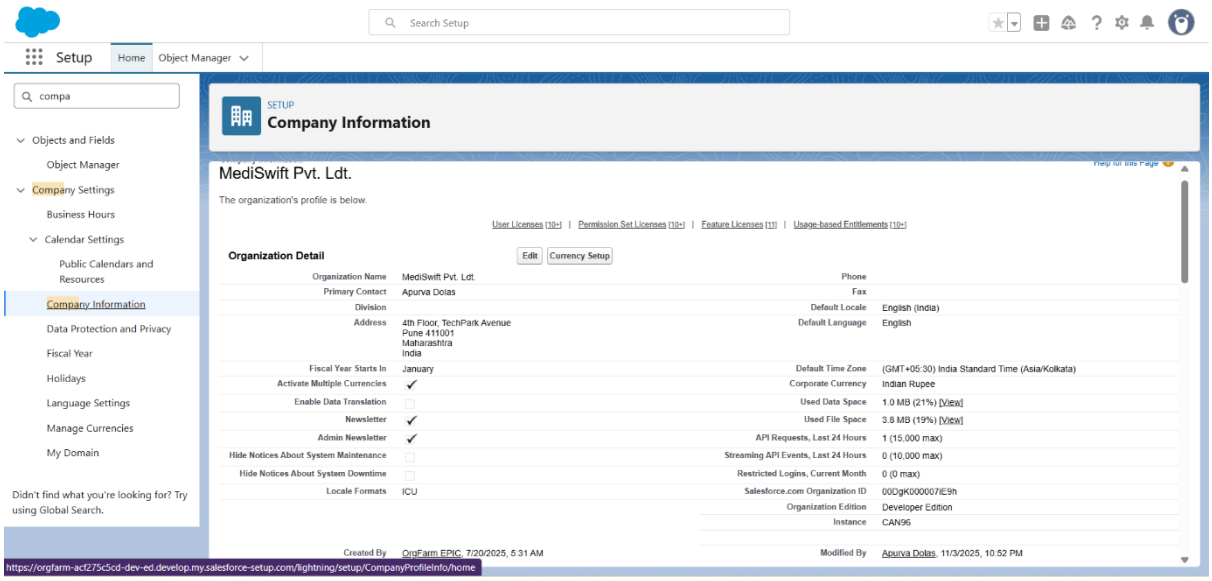
Objective: To configure the company information of *MediSwift Pvt. Ltd.* in Salesforce, ensuring correct organizational details such as name, address, time zone, and currency.

Steps:

1. Go to Setup → Company Settings → Company Information.
2. Click Edit to update organization details.
3. Enter the following:
   - Organization Name: MediSwift Pvt. Ltd.
   - Primary Contact: Apurva Dolas

- Address: 4th Floor, TechPark Avenue, Pune 411001, Maharashtra, India
- Default Locale: English (India)
- Default Time Zone: (GMT+05:30) India Standard Time
- Corporate Currency: Indian Rupee (INR)

4. Save the changes.



➢ Business Hours & Holidays

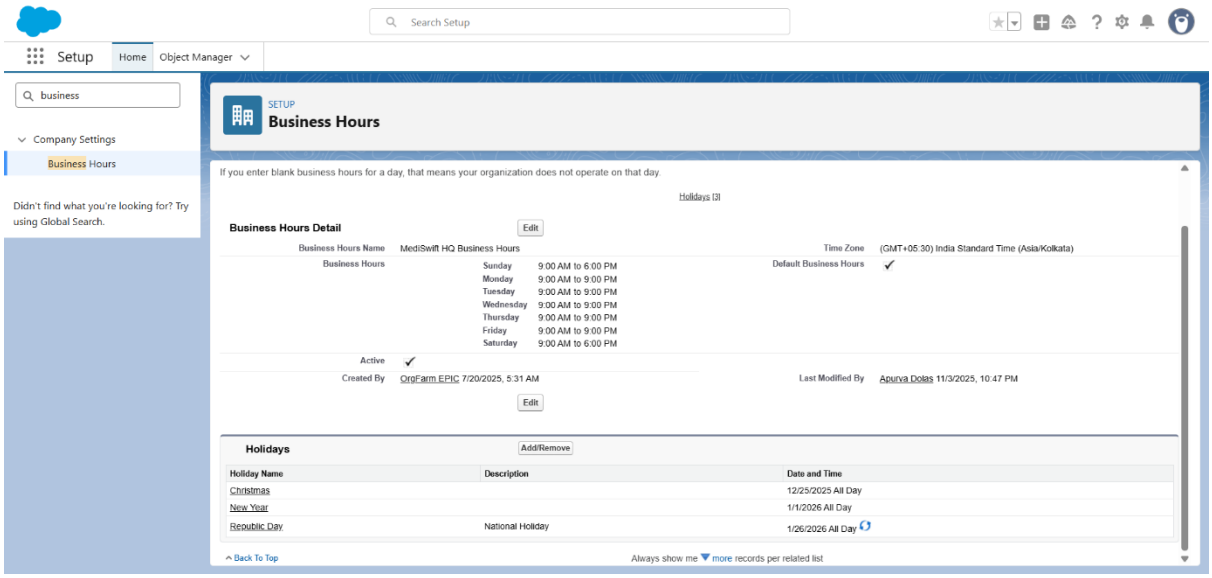
Objectives: To define MediSwift's official operating hours and holidays in Salesforce, ensuring accurate scheduling for customer service, order delivery, and workflow automation.

Steps:

1. Go to Setup → Company Settings → Business Hours.
2. Click New Business Hours or edit the default record.
3. Enter the following details:
   - Business Hours Name: MediSwift HQ Business Hours
   - Time Zone: (GMT+05:30) India Standard Time (Asia/Kolkata)
   - Active: ✔
4. Set daily timings:
   - Monday–Friday: 9:00 AM – 9:00 PM
   - Saturday–Sunday: 9:00 AM – 6:00 PM
5. Click Save.

Holidays Setup:

1. Under the Holidays related list, click Add/Remove.
2. Add common holidays such as:
    - Christmas – 25 December 2025
    - New Year – 1 January 2026
    - Republic Day – 26 January 2026
3. Save the configuration.

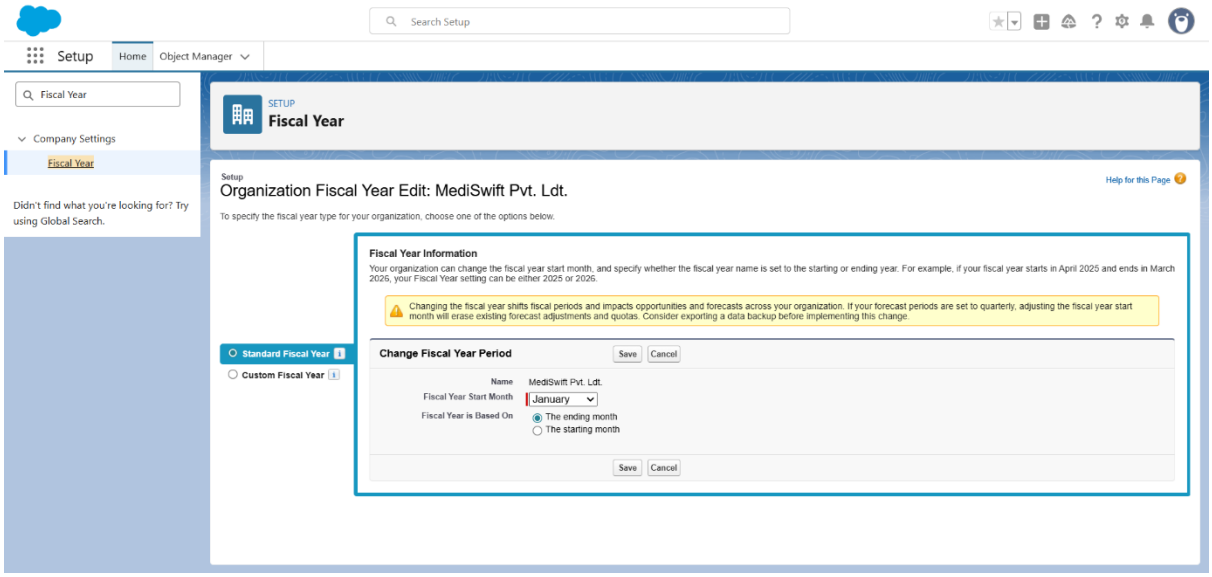


➢ Fiscal Year Settings

Objective: To configure the fiscal year in Salesforce for *MediSwift Pvt. Ltd.* to align business reports, forecasting, and financial planning with the organization's accounting cycle.

Steps:

1. Go to Setup → Company Settings → Fiscal Year.
2. Choose Standard Fiscal Year (suitable for most organizations).
3. Set the following details:
    - Fiscal Year Start Month: January
    - Fiscal Year is Based On: The ending month
4. Click **Save** to apply the settings.

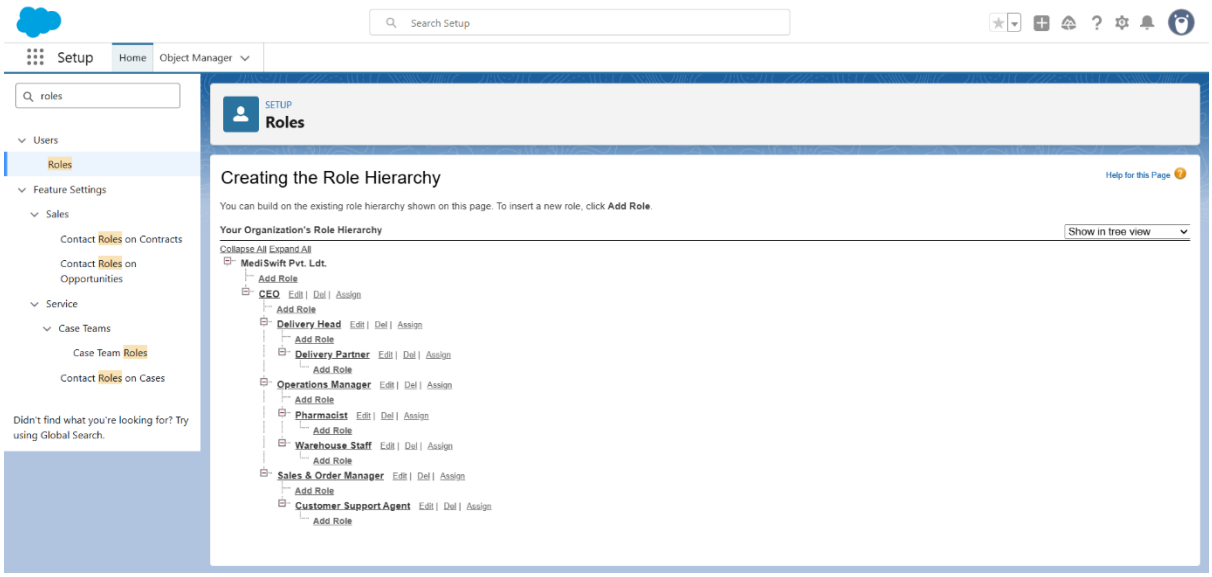


➢ Roles

Objective: To define the role hierarchy in Salesforce for *MediSwift Pvt. Ltd.* so that data access and visibility are managed according to organizational levels.

Steps:

1. Go to Setup → Users → Roles.
2. Click Set Up Roles → Add Role.
3. Create a hierarchical structure for MediSwift as follows:
    - **CEO**
        - **Delivery Head**
            - **Delivery Partner**
        - **Operations Manager**
            - **Pharmacist**
            - **Warehouse Staff**
        - **Sales & Order Manager**
            - **Customer Support Agent**
4. For each role, click **Save** after entering the **Role Name** and **Parent Role**.



➢ Profiles

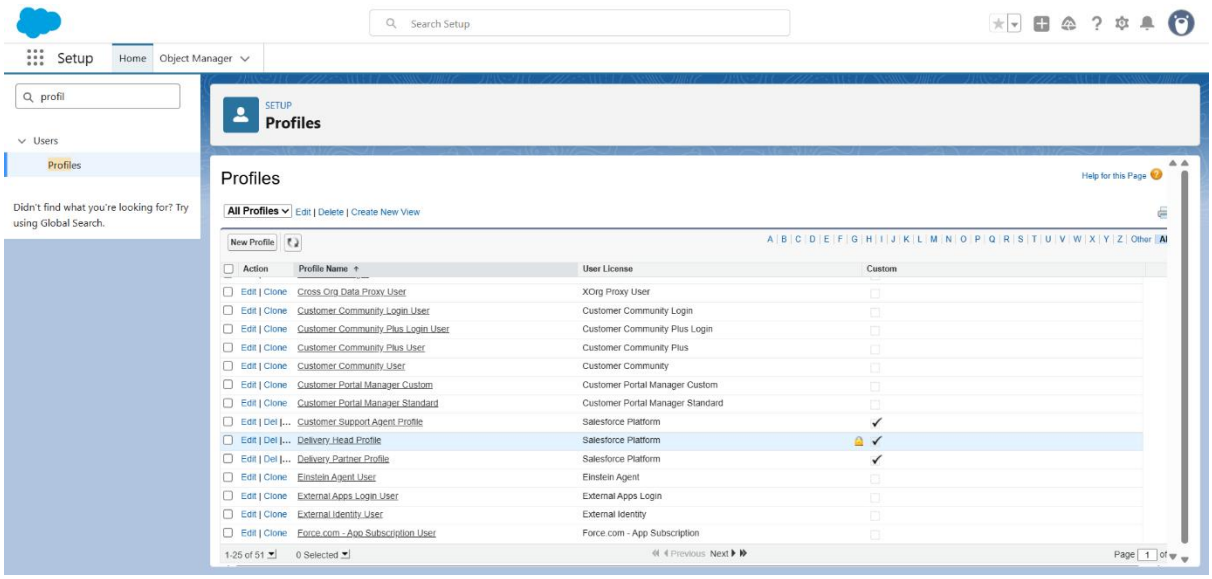
Objective: To create and assign user profiles in Salesforce for *MediSwift Pvt. Ltd.* to control object permissions, field access, and administrative rights according to job roles.

Steps:

1. Go to Setup → Users → Profiles.
2. Click Clone on the *Standard User* or *System Administrator* profile to create a new custom profile.
3. Enter a Profile Name and configure access permissions such as:
    - Object permissions (Read, Create, Edit, Delete)
    - Field-level security
    - Tab visibility and app access
4. Click Save and repeat for each role-based profile.

Profiles Created in MediSwift:

- **System Administrator** – Full access to all data and setup configuration.
- **Delivery Head Profile** – Manages delivery team and oversees partner activities.
- **Delivery Partner Profile** – Handles assigned delivery orders and updates status.
- **Operation Manager Profile** – Supervises order processing and inventory flow.
- **Pharmacist Profile** – Manages medicine inventory and prescription validation.
- **Warehouse Staff Profile** – Updates stock availability and shipment preparation.
- **Sales and Order Manager Profile** – Oversees customer orders and sales reports.
- **Customer Support Agent Profile** – Manages customer queries and complaint records.



➢ User Setup & Licenses

Objectives: To create and assign Salesforce users with appropriate roles, profiles, and licenses in the MediSwift CRM system, ensuring each team member has proper access according to their responsibilities.

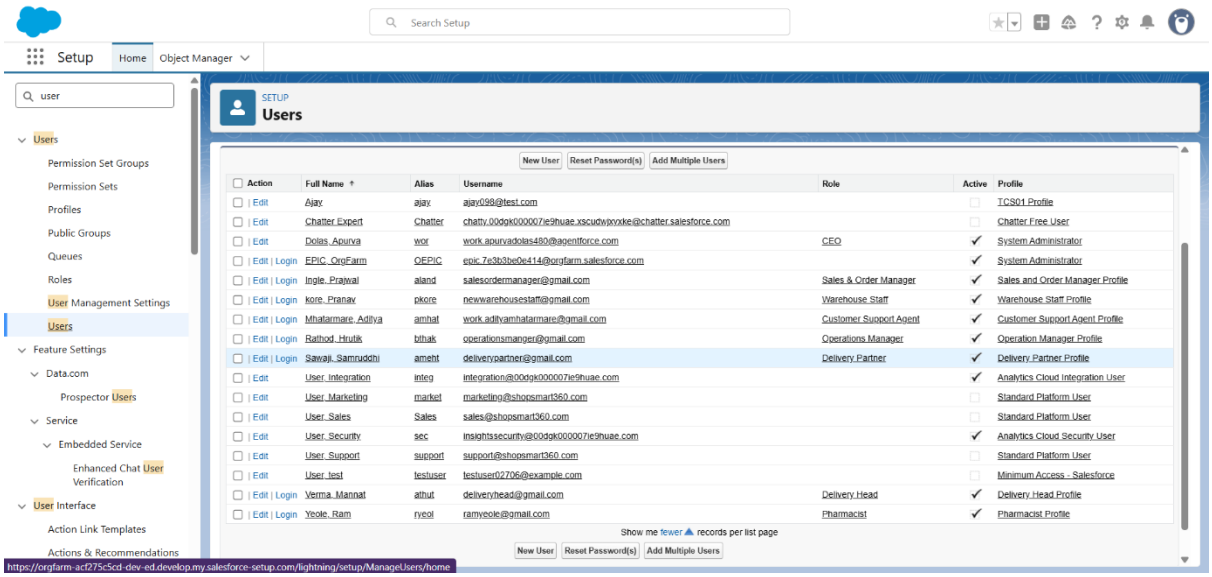
Steps:

1. Navigate to Setup → Users → Users.
2. Click New User.
3. Enter details such as Full Name, Email, Username, and Alias.
4. Assign the appropriate Role (e.g., Delivery Partner, Operations Manager).
5. Choose the corresponding Profile (e.g., Delivery Partner Profile, Operation Manager Profile).
6. Select a User License (typically *Salesforce* license for full CRM access).
7. Check Active to enable the user account.
8. Click Save.

Users Created in MediSwift:

Full Name	Role	Profile	License Type
Apurva Dolas	CEO	System Administrator	Salesforce
Mannat Verma	Delivery Head	Delivery Head Profile	Salesforce Platform
Samruddhi Sawaji	Delivery Partner	Delivery Partner Profile	Salesforce Platform
Hrutik Rathod	Operations Manager	Operation Manager Profile	Salesforce Platform

Aditya Mhatarmare	Customer Support Agent	Customer Support Agent Profile	Salesforce Platform
Pranav Kore	Warehouse Staff	Warehouse Staff Profile	Salesforce Platform
Prajwal Ingle	Sales & Order Manager	Sales & Order Manager Profile	Salesforce
Ram Yeole	Pharmacist	Pharmacist Profile	Salesforce Platform



➢ Login Access Policies

Objectives: To define how administrators and Salesforce support can access user accounts securely for troubleshooting or maintenance within the MediSwift system.

Steps to Configure:

1. Go to Setup → Security → Login Access Policies.
2. Enable the following options:
    - Administrators Can Log in as Any User – allows the System Administrator to log in as any user for issue resolution or testing.
    - Grant Login Access to Salesforce Support – allows Salesforce Support to access the org when support cases are raised.
3. Click Save to apply changes.

Purpose in MediSwift:

- Helps administrators assist team members (e.g., Delivery Partner, Pharmacist) by logging in directly to their accounts.
- Supports faster issue resolution during testing and deployment phases.
- Ensures secure and auditable access for troubleshooting while maintaining data privacy.

Phase 3: Data Modelling & Relationships

➢ Standard & Custom Objects

Objective:
To define and manage the data structure of the MediSwift application using Salesforce custom objects.

Overview:
The MediSwift application does **not** utilize any standard Salesforce objects. Instead, it is built entirely on **custom objects** to meet specific business needs related to pharmacy order management and delivery.

Custom Objects Created in MediSwift:

Object Label	API Name	Purpose
Customer	Customer__c	Stores details of customers using MediSwift services.
Pharmacy	Pharmacy__c	Contains information about registered pharmacies partnered with MediSwift.
Delivery Partner	Delivery_Partner__c	Manages delivery personnel data responsible for order deliveries.
Order	Order__c	Records details of medicine orders placed by customers.
Order Item	Order_Item__c	Tracks individual items within an order.
Payment	Payment__c	Maintains payment and transaction details for customer orders.
Feedback	Feedback__c	Stores customer feedback and service ratings.
Medicine	Medicine__c	Stores information about medicines, including type, composition, stock availability, and pricing.

➢ Fields

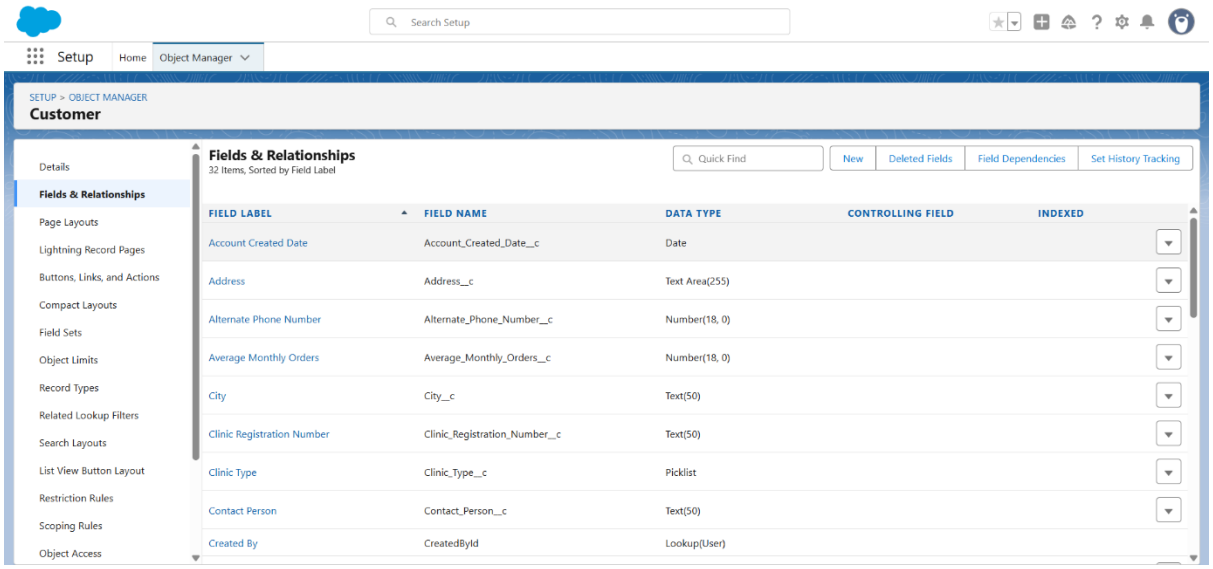
Objectives: To define and configure fields for storing and managing data in Salesforce custom objects used in the MediSwift application.

Purpose in MediSwift:

- To store structured information for customers, pharmacies, orders, and delivery partners.
- To establish relationships between different records (for example, Customer → Order).
- To automate calculations and data display using formulas and picklists.
- To ensure data accuracy and consistency across all business modules.

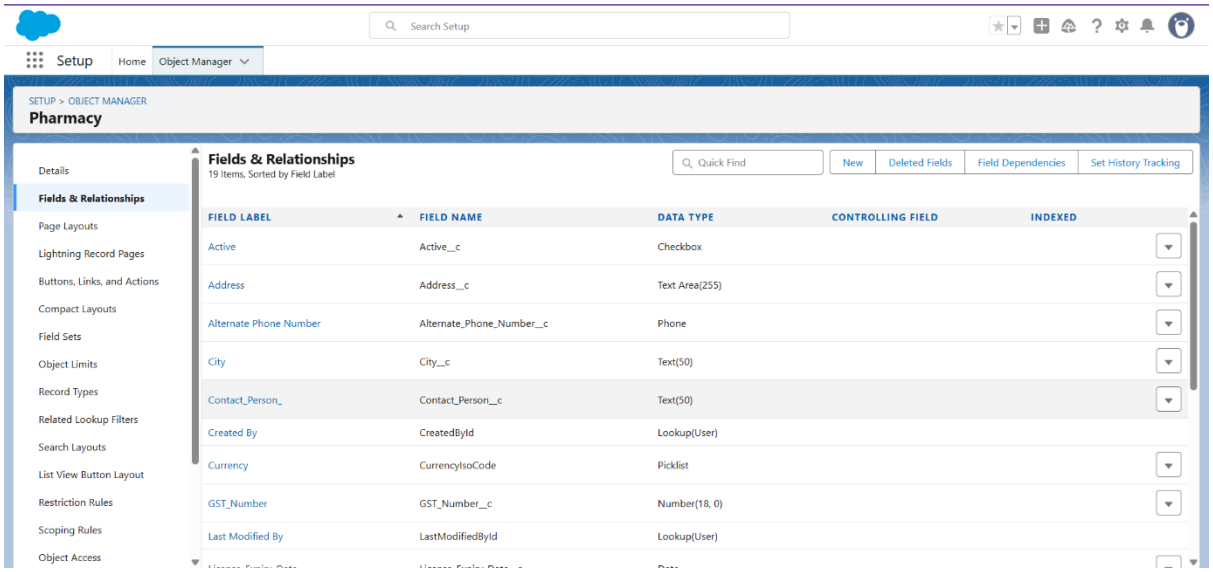
1. Fields in Customer Object

- Account Created Date (Date) — Captures the date when the customer account was created.
- Address (Text Area – 255) — Stores the full address of the customer.
- Alternate Phone Number (Number 18,0) — Secondary contact number for the customer.
- Average Monthly Orders (Number 18,0) — Tracks the customer's average number of monthly orders.
- City (Text 50) — Specifies the customer's city of residence.
- Clinic Registration Number (Text 50) — Registration ID for clinic customers.
- Clinic Type (Picklist) — Defines the type of clinic (e.g., Private, Government).
- Contact Person (Text 50) — Name of the main contact person for the customer.
- Credit Limit (Currency 16,2) — Specifies the maximum credit limit assigned to the customer.
- Customer ID (Auto Number) — Unique system-generated identifier for each customer.
- Customer Status (Picklist) — Indicates whether the customer is *Active* or *Inactive*.
- Date of Birth (Date) — Records the customer's date of birth.
- Email (Email) — Primary email address used for communication.
- Email Send Status (Text 50) — Tracks whether a confirmation or alert email was sent.
- First Name (Text 50) — Customer's first name.
- Full Name (Formula – Text) — Automatically combines name fields to display the full name.
- Gender (Picklist) — Specifies the gender of the customer.
- GST / Tax Number (Text 50) — Holds the GST or tax registration number.
- Last Name (Text 50) — Customer's last name.
- Middle Name (Text 50) — Customer's middle name (if applicable).
- Notes (Long Text Area – 32768) — Stores additional remarks or important comments.
- Phone (Phone) — Customer's primary phone number.
- Pincode (Number 18,0) — Postal or ZIP code of the customer's address.
- Preferred Delivery Time (Picklist) — Customer's chosen delivery time slot.
- Preferred Payment Mode (Picklist) — Defines the preferred payment option (e.g., Cash, Card, Online).
- State (Text 50) — Specifies the customer's state or region.



2. Fields in Pharmacy Object

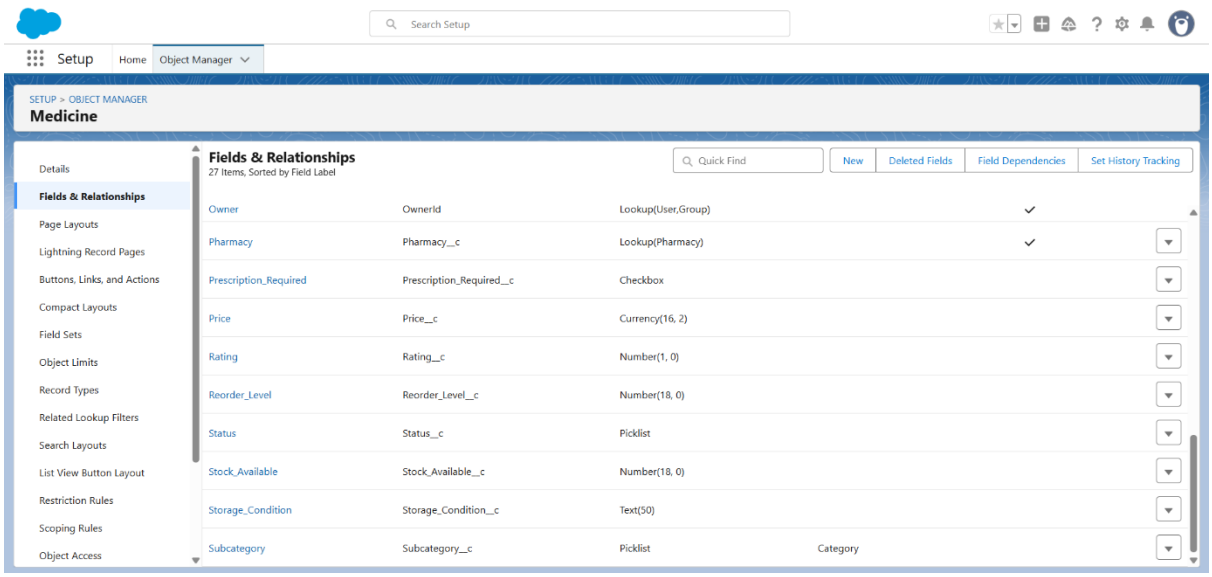
- Active (Checkbox) — Indicates whether the pharmacy is currently active or inactive.
- Address (Text Area – 255) — Stores the complete address of the pharmacy.
- Alternate Phone Number (Phone) — Secondary contact number for the pharmacy.
- City (Text 50) — City where the pharmacy is located.
- Contact Person (Text 50) — Name of the primary contact person at the pharmacy.
- Currency (Picklist) — Defines the currency used for transactions.
- GST Number (Number 18,0) — GST identification number of the pharmacy.
- License Expiry Date (Date) — Expiration date of the pharmacy's operating license.
- License Number (Number 18,0) — Unique registration or license number.
- Notes (Text Area – 255) — Additional notes or remarks related to the pharmacy.
- Operating Hours (Text 50) — Specifies the pharmacy's working hours.
- Pharmacy ID (Auto Number) — Unique system-generated ID for each pharmacy record.
- Pharmacy Name (Text 50) — Official name of the pharmacy.
- Pharmacy Type (Picklist) — Defines the type of pharmacy (e.g., Retail, Hospital-based, Online).
- Phone (Phone) — Primary contact number for the pharmacy.
- Rating (Number 2,0) — Indicates customer or system-assigned rating for the pharmacy's services.



3. Fields in Medicine Object

- Active (Checkbox) — Indicates whether the medicine is currently active or discontinued.
- Batch Number (Number 18,0) — Unique batch identifier for the medicine.
- Brand Name (Text 50) — The commercial brand name of the medicine.
- Category (Picklist) — Defines the medicine's category (e.g., Antibiotic, Painkiller, Supplement).
- Currency (Picklist) — Specifies the currency used for pricing.
- Description (Long Text Area – 32768) — Detailed description or composition of the medicine.
- Discount (Percent 16,2) — Applicable discount percentage on the medicine's price.
- Expiry Date (Date) — Expiration date of the medicine batch.
- Final Price (Formula – Currency) — Automatically calculates the final price after applying discounts.
- Image URL (URL 255) — Link to the image of the medicine for easy identification.
- Last Stock Update (Date/Time) — Tracks the most recent date and time the stock was updated.
- Manufacture Date (Date) — Date when the medicine was manufactured.
- Manufacturer (Text 50) — Name of the company that produces the medicine.
- Medicine ID (Auto Number) — System-generated unique ID for each medicine record.
- Medicine Name (Text 50) — Name of the medicine as displayed in the catalog.
- Pharmacy (Lookup – Pharmacy) — Links the medicine to the associated pharmacy.
- Prescription Required (Checkbox) — Indicates whether a prescription is mandatory for purchase.
- Price (Currency 16,2) — Base price of the medicine before discounts.
- Rating (Number 1,0) — Customer or system rating for the medicine.

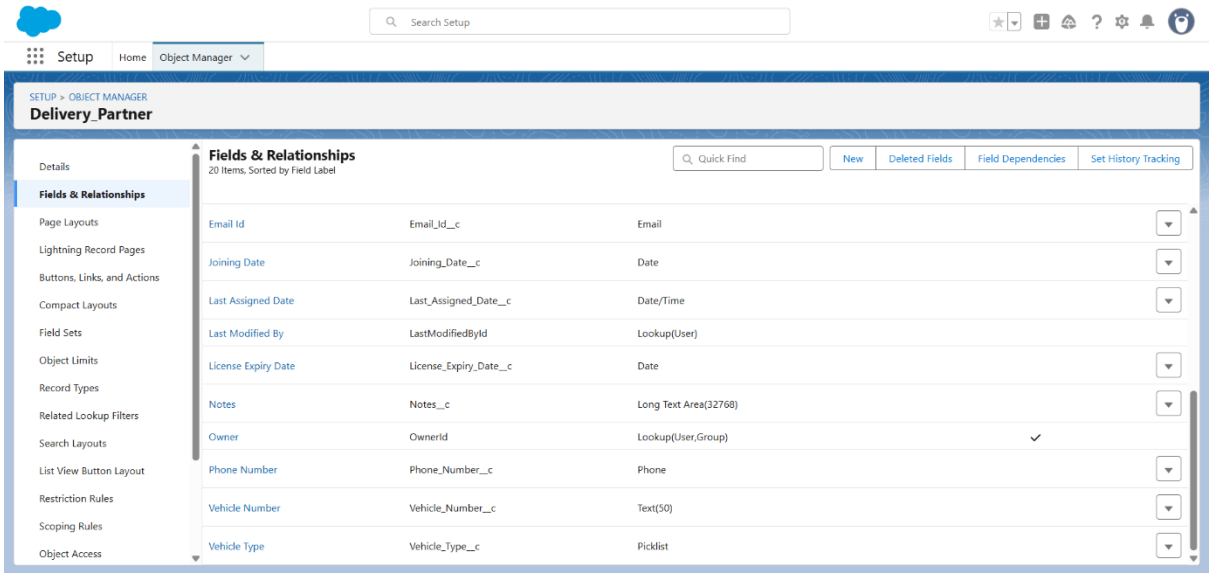
- Reorder Level (Number 18,0) — Defines the minimum stock level to trigger restocking alerts.
- Status (Picklist) — Specifies the availability status (e.g., In Stock, Out of Stock, Discontinued).
- Stock Available (Number 18,0) — Displays the current stock quantity.
- Storage Condition (Text 50) — Recommended storage condition (e.g., Cool, Dry Place).
- Subcategory (Picklist) — Further classifies the medicine within its main category.



4. Fields in Delivery Partner Object

- Active (Checkbox) — Indicates whether the delivery partner is active or inactive.
- Availability Status (Picklist) — Displays the current status (e.g., Available, On Delivery, Offline).
- Average Rating (Number 2,0) — Represents the delivery partner's average rating based on performance and feedback.
- Currency (Picklist) — Defines the currency used for payments or compensation.
- Current Location (Text 50) — Records the delivery partner's current location for tracking.
- Delivery Partner Name (Text 50) — Full name of the delivery partner.
- Delivery Zone (Picklist) — Defines the service zone or area assigned to the delivery partner.
- Delivery Partner ID (Auto Number) — Unique system-generated identifier for each delivery partner.
- Driving License Number (Text 50) — Official driving license number of the delivery partner.
- Email ID (Email) — Registered email address of the delivery partner.
- Joining Date (Date) — Date when the delivery partner joined MediSwift.

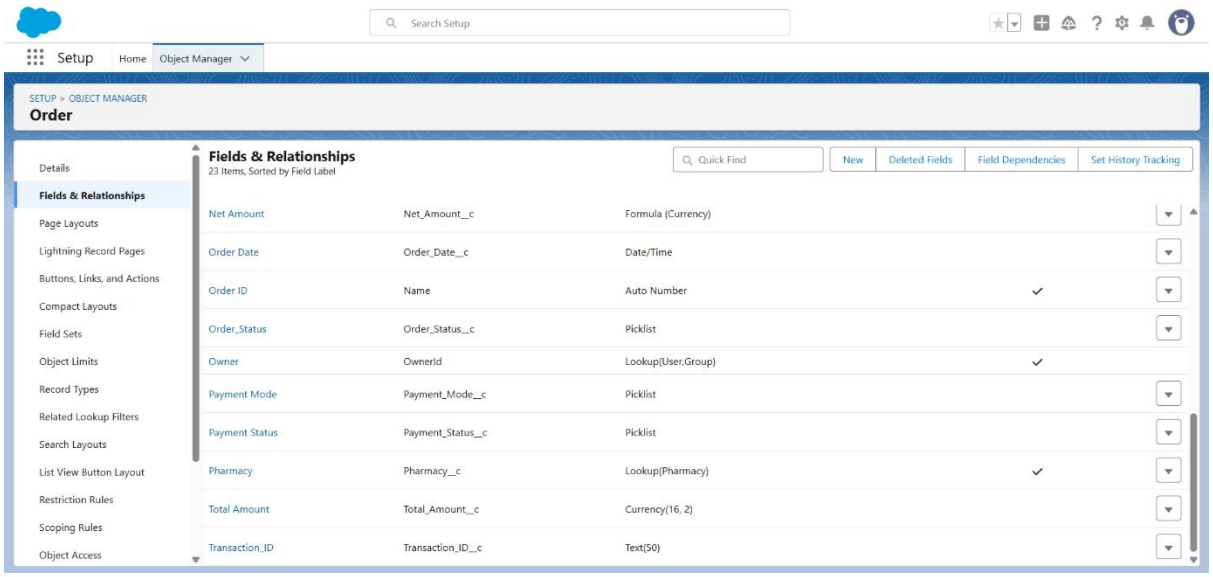
- Last Assigned Date (Date/Time) — Date and time of the most recent delivery assignment.
- License Expiry Date (Date) — Expiration date of the delivery partner's driving license.
- Notes (Long Text Area – 32768) — Additional remarks or notes related to the delivery partner.
- Phone Number (Phone) — Contact number for the delivery partner.
- Vehicle Number (Text 50) — Vehicle registration number used for deliveries.
- Vehicle Type (Picklist) — Type of vehicle used (e.g., Bike, Car, Van).



5. Fields in Order Object

- Approval Status (Picklist) — Displays the order's approval stage (e.g., Pending, Approved, Rejected).
- Currency (Picklist) — Defines the transaction currency for the order.
- Customer (Lookup – Customer) — Links the order to the corresponding customer record.
- Delivered Date (Date) — Captures the date when the order was successfully delivered.
- Delivery Status (Picklist) — Indicates the current status of delivery (e.g., In Progress, Delivered, Cancelled).
- Delivery Address (Long Text Area – 32768) — Stores the complete delivery address for the order.
- Delivery Partner (Lookup – Delivery Partner) — Associates the order with the assigned delivery partner.
- Discount (Currency 16,2) — Records any discount applied to the order total.
- Expected Delivery Date (Date) — Estimated date for order delivery.
- Feedback Comments (Long Text Area – 32768) — Customer's comments or feedback about the order.

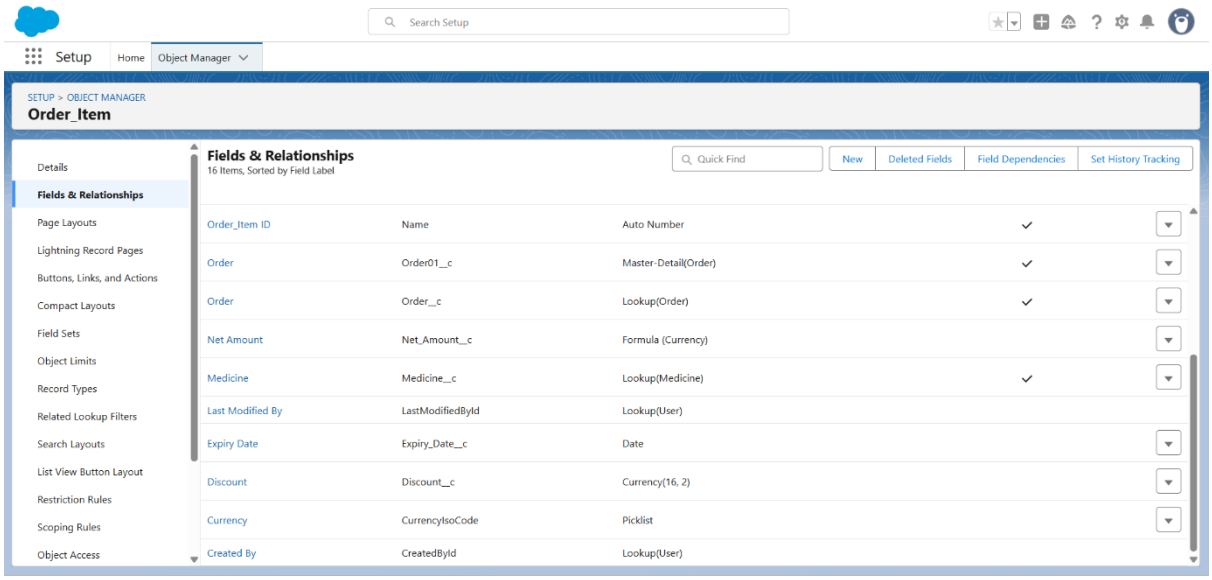
- Feedback Rating (Number 2,0) — Customer rating of the order experience.
- Net Amount (Formula – Currency) — Automatically calculates the payable amount after applying discounts.
- Order Date (Date/Time) — Timestamp when the order was placed.
- Order ID (Auto Number) — Unique system-generated identifier for each order.
- Order Status (Picklist) — Shows the current stage of the order (e.g., New, Processing, Completed).
- Payment Mode (Picklist) — Defines the mode of payment (e.g., Cash, Card, Online).
- Payment Status (Picklist) — Indicates whether the payment is Pending, Paid, or Failed.
- Pharmacy (Lookup – Pharmacy) — Connects the order to the fulfilling pharmacy.
- Total Amount (Currency 16,2) — Displays the total order amount before discount.
- Transaction ID (Text 50) — Unique transaction reference number for payment tracking.



6. Fields in Order Items Object

- Unit Price (Currency 16,2) — Base price of the medicine per unit.
- Subtotal (Formula – Currency) — Automatically calculates the total amount before discounts.
- Status (Picklist) — Displays the item's current status (e.g., Pending, Processed, Delivered).
- Quantity (Number 18,0) — Number of medicine units included in the order.
- Prescription Required (Checkbox) — Indicates whether a prescription is required for this item.
- Pharmacy (Lookup – Pharmacy) — Links the item to the associated pharmacy.
- Order Item ID (Auto Number) — Unique system-generated identifier for each order item.

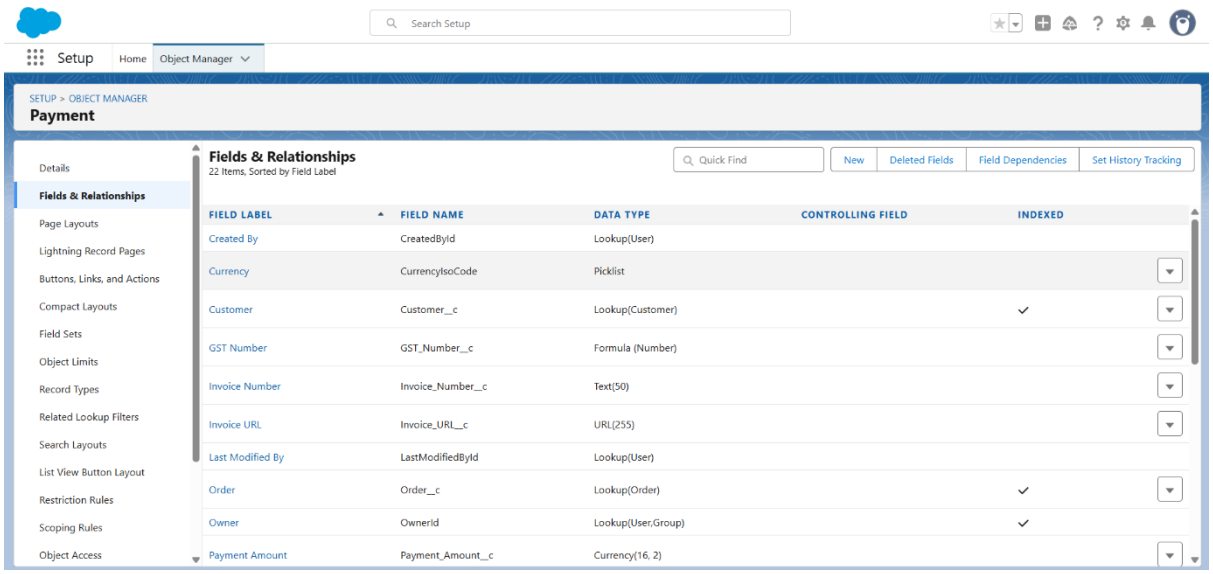
- Order (Master-Detail – Order) — Establishes a master-detail relationship with the Order object.
- Order (Lookup – Order) — Provides an additional lookup reference to the parent order.
- Net Amount (Formula – Currency) — Calculates the final amount after discounts.
- Medicine (Lookup – Medicine) — Connects the order item to the specific medicine record.
- Expiry Date (Date) — Displays the expiration date of the medicine batch.
- Discount (Currency 16,2) — Amount of discount applied on the item.
- Currency (Picklist) — Defines the transaction currency used.



7. Fields in Payment Object

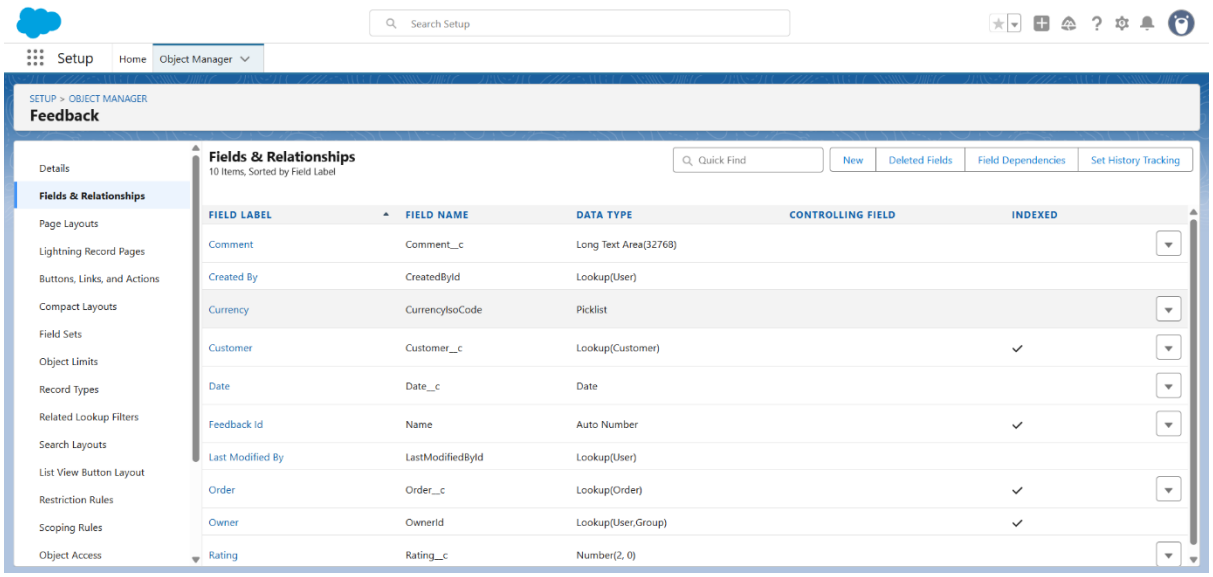
- Currency (Picklist) — Specifies the currency used for the payment transaction.
- Customer (Lookup – Customer) — Links the payment to the related customer.
- GST Number (Formula – Number) — Displays the customer's GST identification number derived automatically.
- Invoice Number (Text 50) — Unique reference number for the generated invoice.
- Invoice URL (URL 255) — Provides a link to download or view the invoice document.
- Order (Lookup – Order) — Associates the payment record with a specific order.
- Payment Amount (Currency 16,2) — Records the total payment amount made by the customer.
- Payment Date (Date/Time) — Captures the exact date and time when the payment was completed.
- Payment ID (Auto Number) — System-generated unique identifier for each payment record.
- Payment Mode (Picklist) — Defines the method of payment (e.g., Cash, Card, UPI, Online).

- Payment Name (Text 50) — Descriptive name for the payment entry.
- Payment Status (Picklist) — Indicates the current payment status (e.g., Pending, Completed, Refunded).
- Pharmacy (Lookup – Pharmacy) — Links the payment to the related pharmacy.
- Refund Amount (Currency 16,2) — Records the refunded amount, if applicable.
- Refund Date (Date) — Specifies the date when the refund was processed.
- Remarks (Long Text Area – 32768) — Additional notes or comments regarding the payment.
- Tax Amount (Currency 16,2) — Displays the calculated tax amount on the payment.
- Total With Tax (Formula – Currency) — Automatically computes the total amount including tax.
- Transaction ID (Text 50) — Unique transaction reference number generated by the payment gateway.



8. Fields in Feedback Object

- Comment (Long Text Area – 32768) — Customer's detailed feedback or remarks about their order or experience.
- Currency (Picklist) — Indicates the currency used in the related transaction, if applicable.
- Customer (Lookup – Customer) — Links the feedback record to the specific customer who provided it.
- Date (Date) — The date when the feedback was submitted.
- Feedback ID (Auto Number) — Unique identifier automatically generated for each feedback record.
- Order (Lookup – Order) — Connects the feedback to the corresponding order.
- Rating (Number 2,0) — Numeric rating provided by the customer (e.g., 1–5 scale).



➢ Lookup vs Master-Detail Relationship

1. Lookup Relationships

These are found in your custom objects where one object references another:

- Customer__c → (No lookups mentioned)
- Pharmacy__c → (No lookups mentioned)
- Medicine__c → Lookup to Pharmacy__c
- Delivery_Partner__c → (No lookups mentioned)
- Order__c → Lookup to:
    - Customer__c
    - Delivery_Partner__c
    - Pharmacy__c
- Order_Item__c → Lookup to:
    - Pharmacy__c
    - Order__c (has both Lookup and Master-Detail)
    - Medicine__c
- Payment__c → Lookup to:
    - Customer__c
    - Order__c
    - Pharmacy__c
- Feedback__c → Lookup to:
    - Customer__c
    - Order__c

2. Master-Detail Relationship

- Found only in Order_Item__c → Order__c (Master-Detail (Order) field).
  So, one Master-Detail Relationship exists.

➢ Schema Builder

The Schema Builder in Salesforce visually represents all the custom objects, their fields, and relationships within the MediSwift application. It provides an interactive way to view and manage the data model, including the object structure and connections between them.

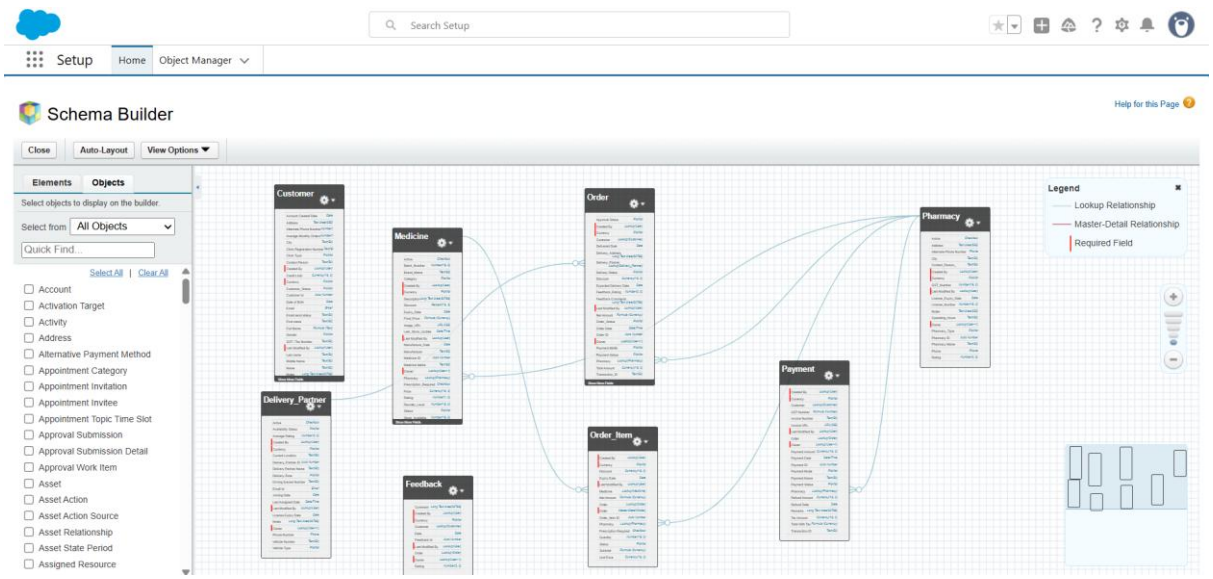
In the MediSwift Schema Builder, the following custom objects are displayed:

- Customer__c
- Pharmacy__c
- Medicine__c
- Delivery_Partner__c
- Order__c
- Order_Item__c
- Payment__c
- Feedback__c

Each object is connected through defined Lookup and Master-Detail relationships:

- Order__c links to Customer__c, Pharmacy__c, and Delivery_Partner__c using lookup relationships.
- Order_Item__c connects to Order__c using a Master-Detail relationship and to Medicine__c and Pharmacy__c through lookups.
- Payment__c connects to Customer__c, Order__c, and Pharmacy__c via lookups.
- Feedback__c connects to Customer__c and Order__c through lookups.
- Medicine__c connects to Pharmacy__c using a lookup.

These relationships help visualize how various components of the MediSwift system interact — such as how a customer places an order, the pharmacy fulfills it, and the delivery partner delivers it — all linked together in the Schema Builder view.



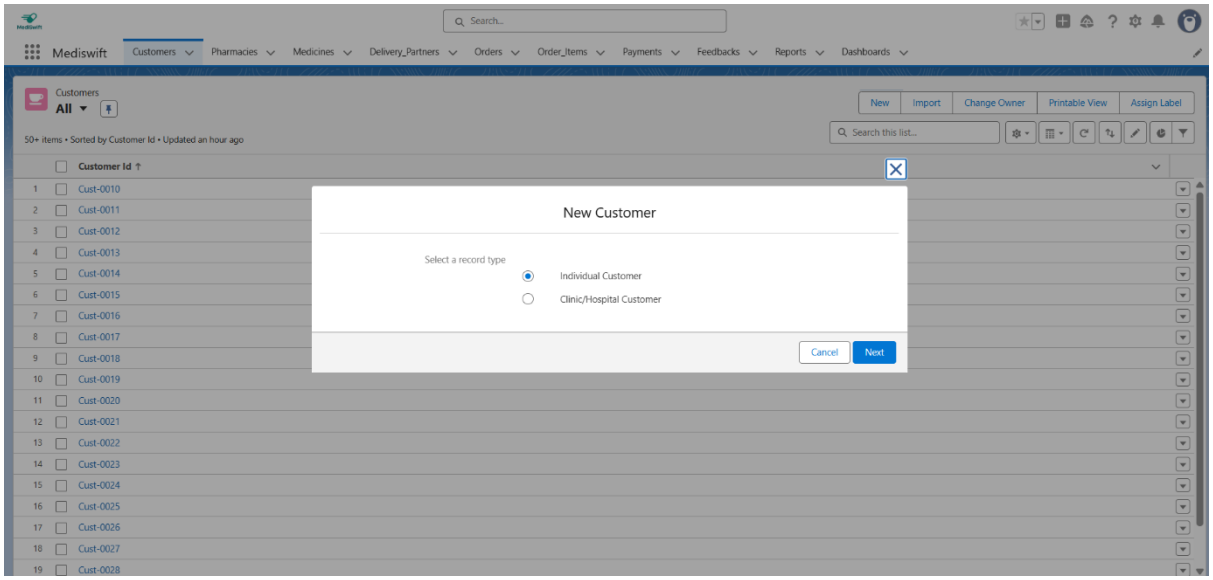
➢ Record Types

In the MediSwift application, Record Types are used to differentiate between different types of customers within the Customer__c object. This allows the system to display customized page layouts, picklist values, and business processes based on the customer type.

Record Types in MediSwift

- Individual Customer:
  Used for personal customers purchasing medicines for self-use. This record type captures basic personal details such as name, contact number, and address.
- Clinic/Hospital Customer:
  Designed for medical institutions, clinics, or hospitals that place bulk medicine orders. This record type includes additional fields like Clinic Registration Number, Clinic Type, and Contact Person.

By using record types, MediSwift ensures that data entry screens and workflows are tailored to the specific needs of individual and institutional customers, improving efficiency and data organization.



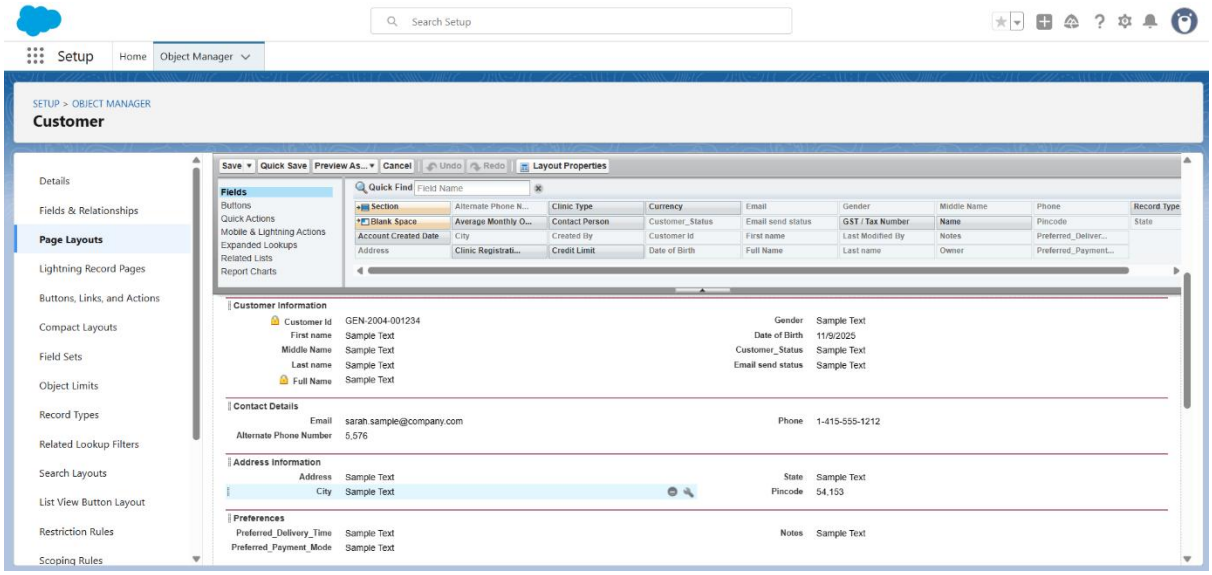
➢ Page Layouts

In MediSwift, Page Layouts are used to organize and display fields, related lists, and buttons on each object's record page. They define how information is presented to users based on their role and record type, ensuring a clear and efficient user experience.

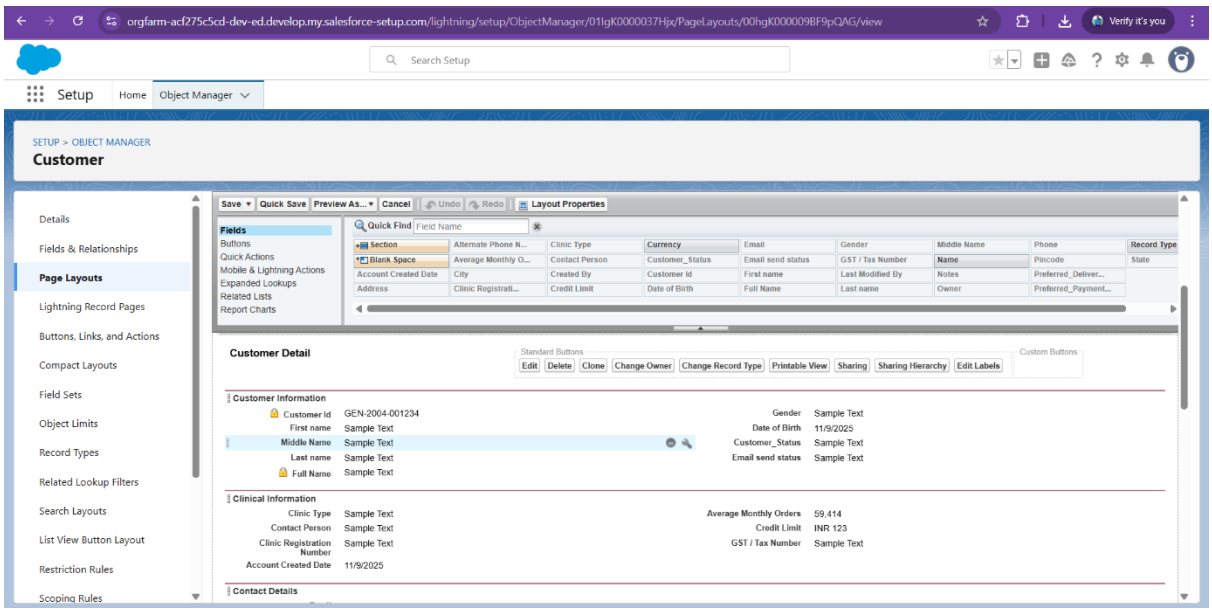
Page Layouts in MediSwift
Each major custom object in MediSwift has a dedicated page layout designed for ease of use and logical data flow:

- **Customer__c Page Layout:**
  Displays customer details such as personal information, clinic details, contact information, and preferences. Different layouts are customized for Individual Customers and Clinic/Hospital Customers based on their record types.

Individual Customer:

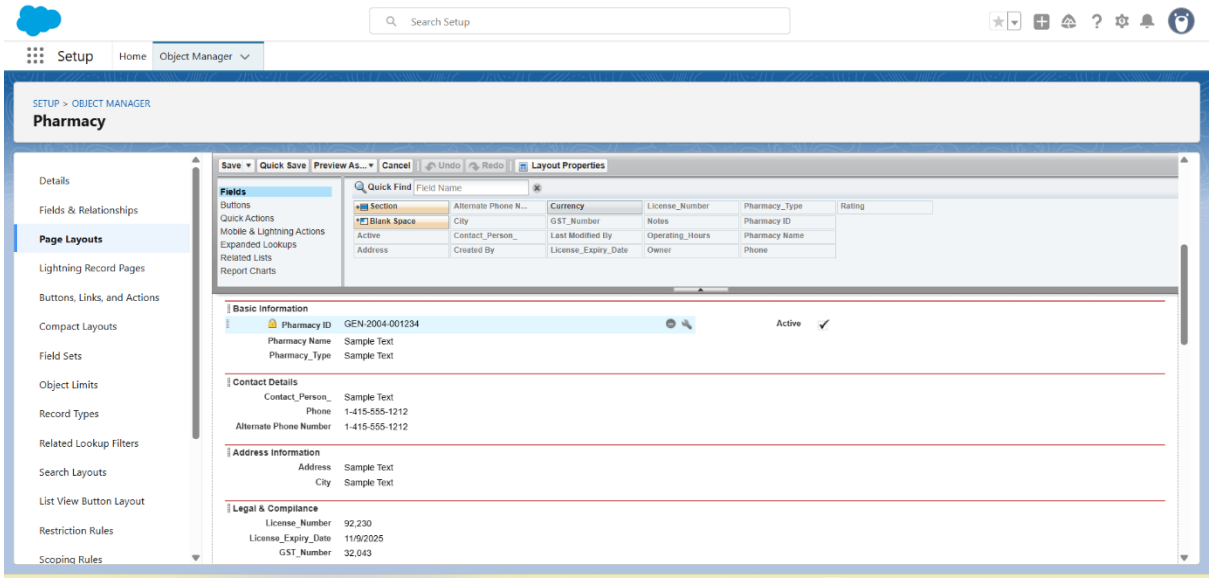


Clinic/Hospital Customer:

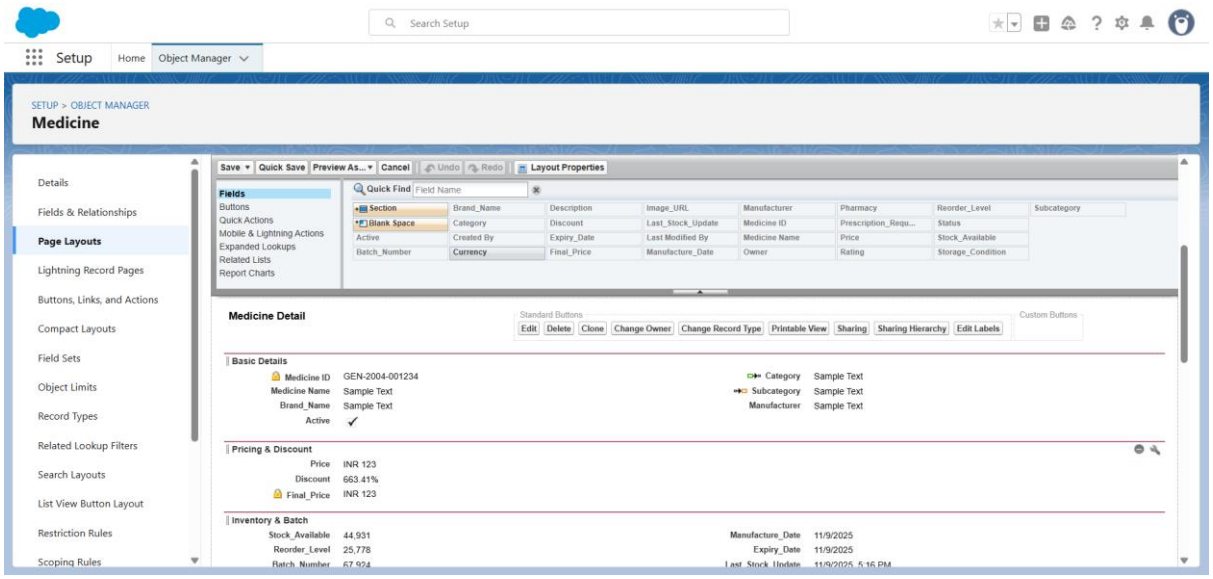


- **Pharmacy__c Page Layout:**

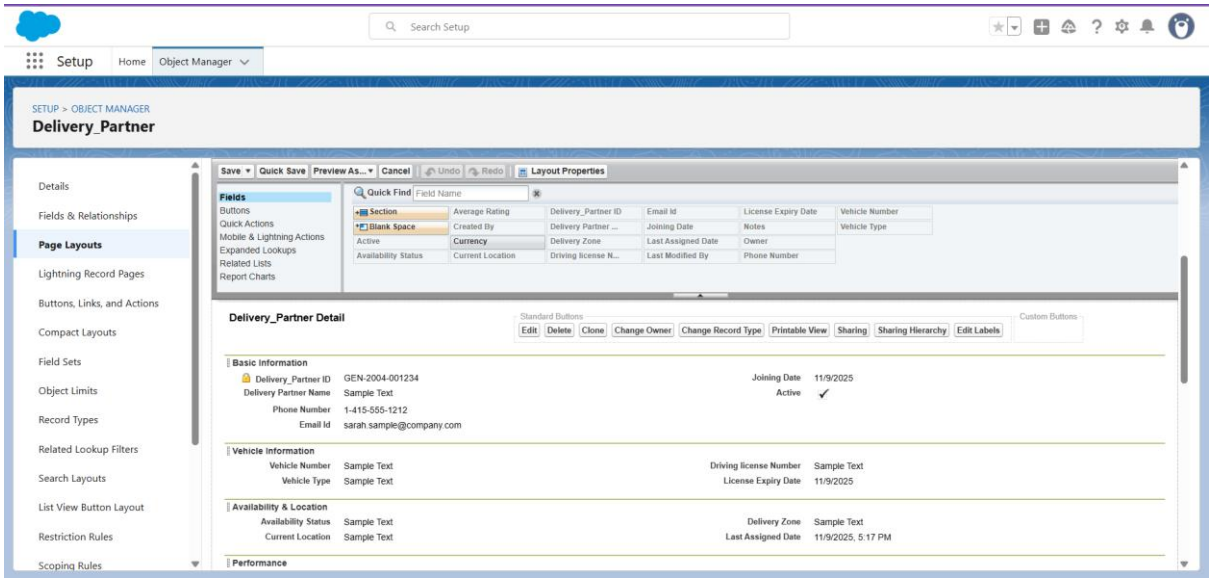
  Includes pharmacy details like address, license information, contact person, operating hours, and rating.



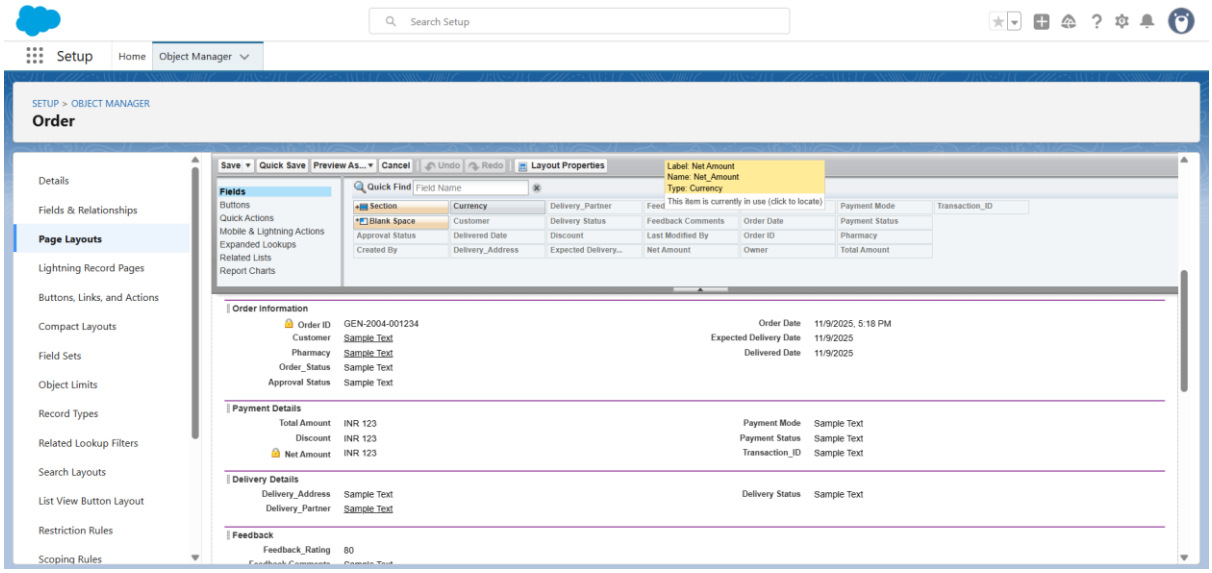
- **Medicine__c Page Layout:**
  Shows details such as brand, category, expiry date, stock availability, and price.



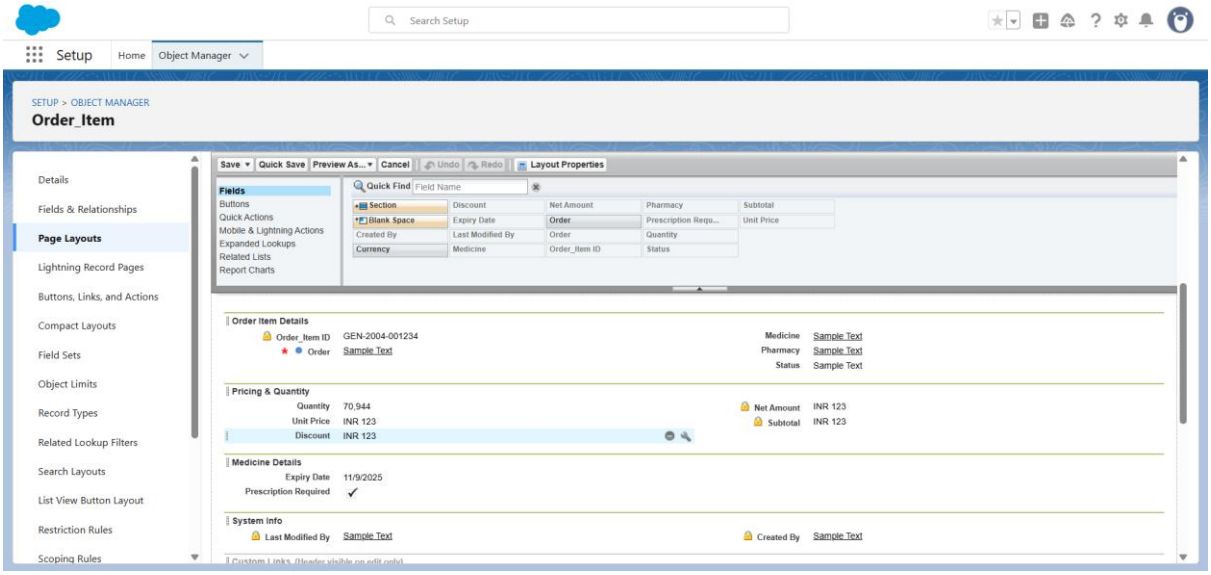
- **Delivery_Partner__c Page Layout:**
  Displays partner information like contact details, vehicle details, license number, and current availability status.



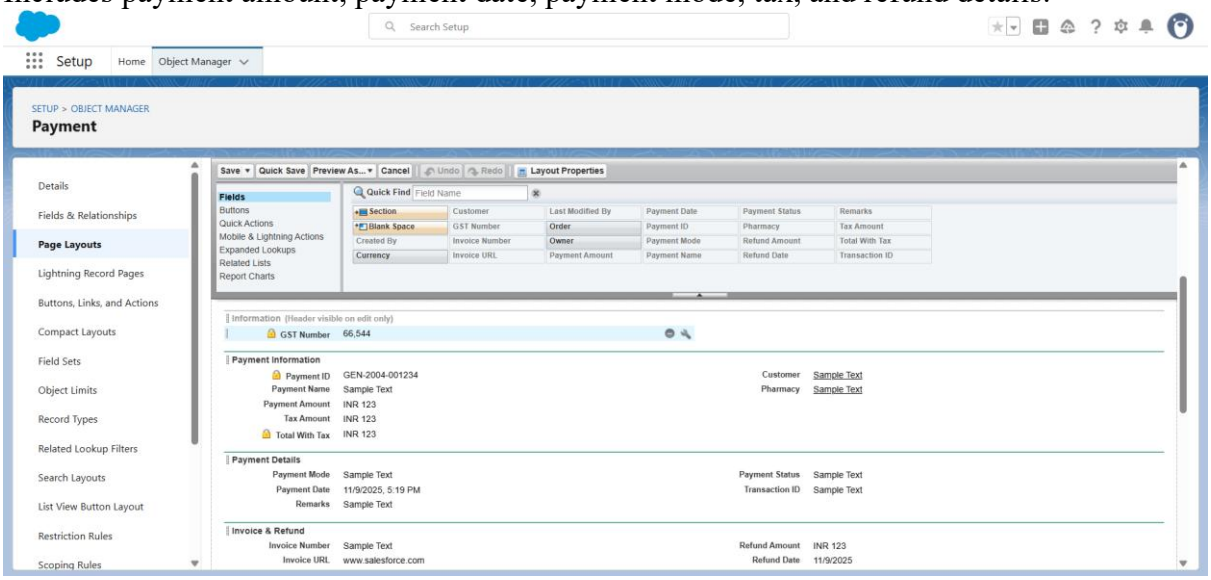
- **Order__c Page Layout:**
  Provides order-related details including customer, pharmacy, delivery partner, order status, payment mode, and delivery address.



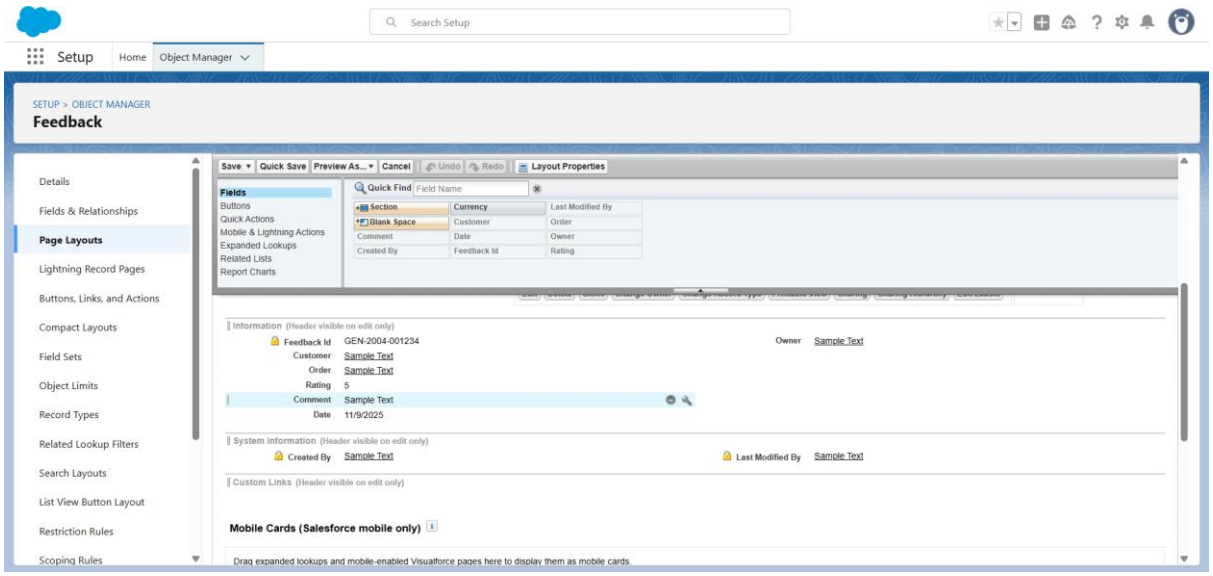
- **Order_Item__c Page Layout:**
  Lists individual items within an order, including medicine, quantity, unit price, and subtotal.



- **Payment__c Page Layout:**
  Includes payment amount, payment date, payment mode, tax, and refund details.



- **Feedback__c Page Layout:**
  Displays customer feedback, rating, and comments linked to related orders.



➢ Hierarchical Relationship

- This relationship type is only available for the User object.
- Your MediSwift custom objects (Customer, Pharmacy, etc.) don't use it.

Not present in MediSwift.

➢ Junction Objects

- A junction object requires two Master-Detail relationships (to link two parents).
- In your case, only one Master-Detail relationship exists (Order_Item → Order).
- Therefore, no true junction object is present.

Not present in MediSwift.

➢ External Object

- None of your shared objects (`Customer__c`, `Pharmacy__c`, `Medicine__c`, etc.) have an external ID reference or "__x" suffix (which indicates external objects).
- All are standard custom objects stored inside Salesforce.

Not present in MediSwift.

Phase 4: Process Automation (Admin)

➢ Validation Rules

In MediSwift, validation rules are used to ensure data accuracy and enforce business logic while users enter records. A validation rule checks that the data entered into a record meets specific criteria before saving it.

Validation Rule on Customer Object

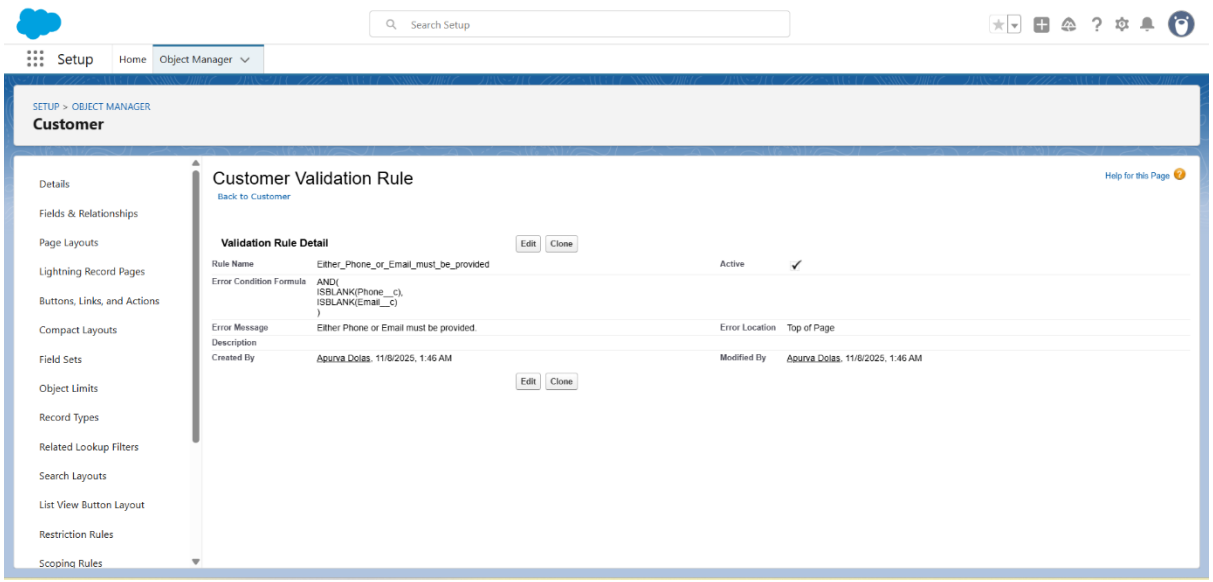
- Rule Name: Either_Phone_or_Email_must_be_provided
- Object: Customer__c
- Active: Yes
- Error Condition Formula:

```
AND(
   ISBLANK(Phone__c),
   ISBLANK(Email__c)
)
```

- Error Message: *Either Phone or Email must be provided.*
- Error Location: Top of Page
- Created By: Apurva Dolas
- Created On: 8th November 2025

Description:

This rule ensures that a customer record cannot be saved without providing at least one mode of contact — either a phone number or an email address. It maintains data completeness and supports better communication with customers.



➢ Workflow Rules

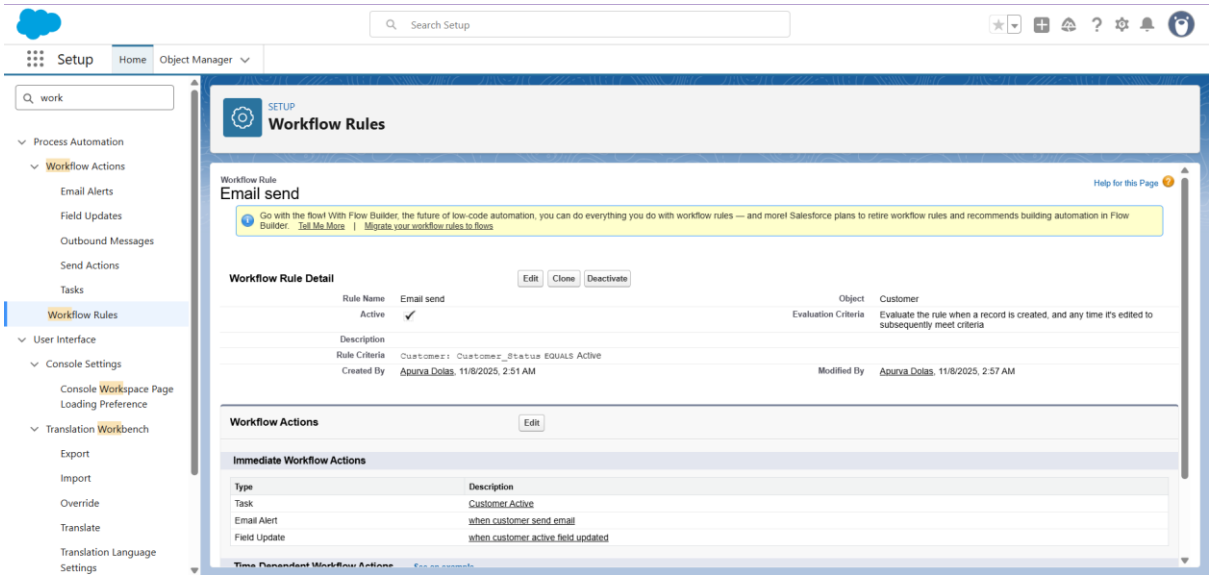
The Email Send workflow rule in MediSwift is designed to automate actions whenever a customer's status changes to Active. This rule ensures timely communication and efficient management of customer activation processes.

Detailed Explanation

- **Rule Name: Email send**
  This is the name assigned to the workflow rule. It indicates that the rule's purpose is to send an email or trigger actions when certain conditions related to customer activity are met.
- **Object: Customer__c**
  The workflow rule is created on the Customer object, meaning it will monitor changes made to customer records.
- **Active: Checked**
  This indicates that the rule is currently active and running in the system. Once active, it automatically evaluates customer records according to the defined criteria.
- **Evaluation Criteria:**
  "Evaluate the rule when a record is created, and any time it's edited to subsequently meet criteria."
  This means the rule is triggered both when a new customer record is created and when an existing record is edited to meet the rule's condition — in this case, when the customer's status becomes *Active*.
- **Rule Criteria:**
  Customer: Customer_Status equals Active
  The workflow rule checks the field Customer_Status__c on the Customer object. If the value changes to Active, the rule criteria are met and the workflow actions are executed.
- **Created By: Apurva Dolas**
  Created On: 8th November 2025
  This shows the author and creation date of the rule, helping with audit tracking and configuration management.
- **Modified By: Apurva Dolas**
  Modified On: 8th November 2025
  Indicates that the rule was last modified by the same user, possibly to refine or test the automation behavior.

Purpose

The purpose of this workflow rule is to automatically trigger actions (such as sending emails, creating tasks, or updating fields) whenever a customer becomes active. This reduces manual effort and ensures that customers receive timely communication from the MediSwift system.



➢ Email Alerts – When Customer Send Email

The Email Alert feature in Salesforce is used to automatically send email notifications to designated recipients when a specific condition or rule is met. In MediSwift, this email alert is linked to the "Email send" workflow rule on the Customer object to notify customers when their status becomes active.

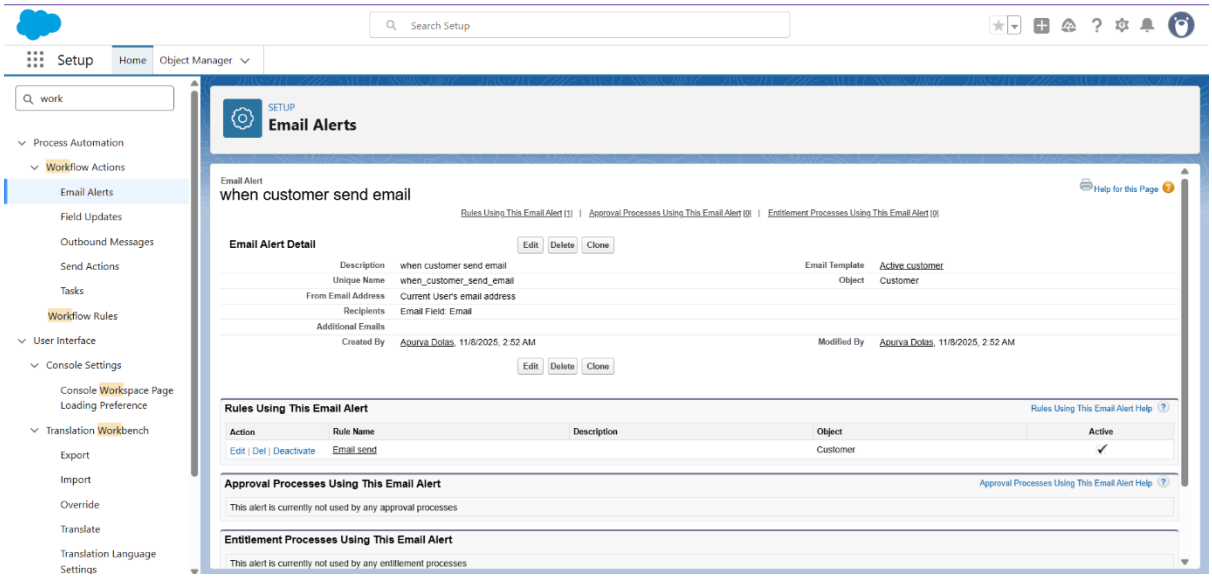
Email Alert Details

- **Email Alert Name: When Customer Send Email**
  This is the name assigned to the email alert, clearly indicating its purpose — to send an automated email when the customer meets specific activation conditions.
- **Description: When customer send email**
  This description briefly explains that the alert is triggered when the system sends an email to the customer upon activation.
- **Email Template: Active Customer**
  The alert uses a predefined email template named *Active Customer*. This template contains the content of the email sent to the customer, ensuring consistency and a professional message format.
- **Unique Name: when_customer_send_email**
  This is the system-generated unique identifier for the email alert, used internally by Salesforce.
- **Object: Customer__c**
  Indicates that the email alert is associated with the Customer object and is triggered by changes or actions related to customer records.
- **From Email Address: Current User's Email Address**
  The email notification is sent from the email address of the currently logged-in Salesforce user who triggered the rule.
- **Recipients: Email Field – Email__c**
  The recipient of the email is the customer's email address specified in the Email__c field on the Customer record.

- **Additional Emails:** *None specified*
  No extra recipients are added, so the email is sent only to the customer.

Purpose

This email alert ensures that whenever a customer's status is set to Active, an automated email is sent to notify them of their activation. It enhances communication and provides immediate acknowledgment to the customer without requiring manual intervention.



➢ Field Updates – When Customer Active Field Updated

The Field Update action in Salesforce automatically modifies the value of a specific field when a workflow rule is triggered. In MediSwift, this field update is linked to the "Email send" workflow rule on the Customer object. It updates the customer record once an email has been successfully sent.

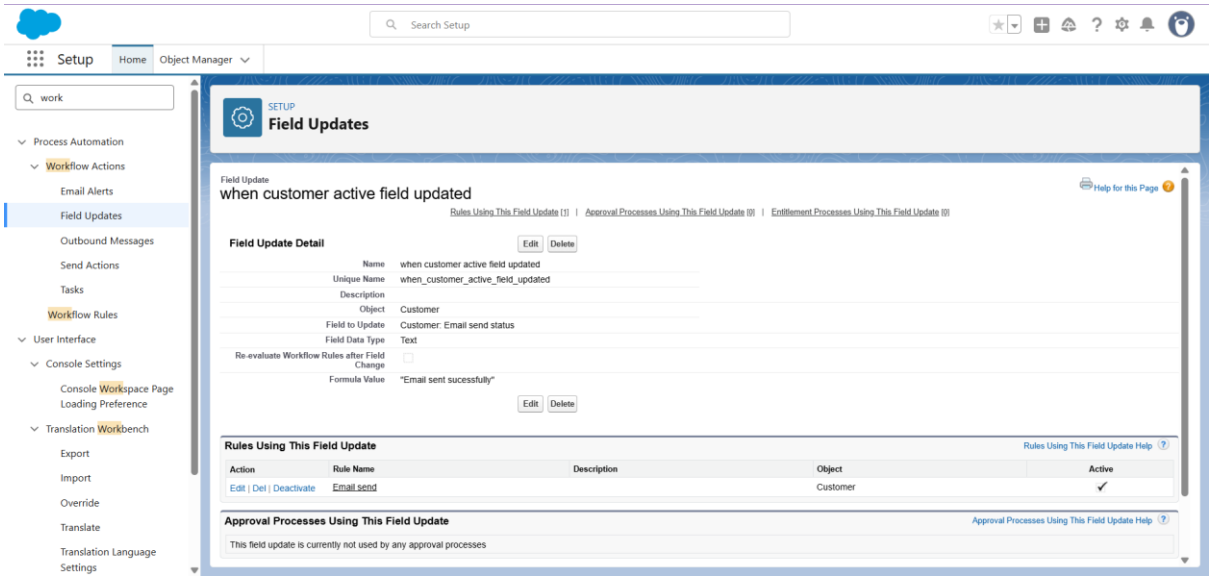
Field Update Details

- **Name: When Customer Active Field Updated**
  This is the name given to the field update action, indicating that it updates a field when the customer record becomes active and the workflow rule is triggered.
- **Unique Name: when_customer_active_field_updated**
  The system-generated unique identifier used internally by Salesforce to reference this field update.
- **Object: Customer__c**
  The field update is applied to the **Customer** object, where customer details and communication statuses are maintained.
- **Field to Update: Customer: Email_send_status__c**
  The specific field that will be updated by this workflow action. It records whether the system successfully sent an activation email to the customer.

- **Field Data Type: Text**
  Indicates that the target field accepts text values.
- **Formula Value:**
  "Email sent successfully"
  When the workflow rule is triggered, this value is automatically entered into the **Email_send_status__c** field to confirm the email action was completed.
- **Re-evaluate Workflow Rules After Field Change: Not Checked**
  This means that the update to the field will not trigger any other workflow rules, preventing recursive automation loops.

Purpose

This field update ensures that whenever a customer's status is set to Active and an email alert is sent, the Email Send Status field is automatically updated to *"Email sent successfully."*
It helps track communication history, supports reporting, and ensures transparency in the activation process.



➢ Tasks - Customer Active

In MediSwift, a Workflow Task is used to automatically create a follow-up activity when a specific workflow rule is triggered. This ensures that users are reminded to take timely action without manual task creation. The Customer Active task is linked to the "Email send" workflow rule on the Customer object.

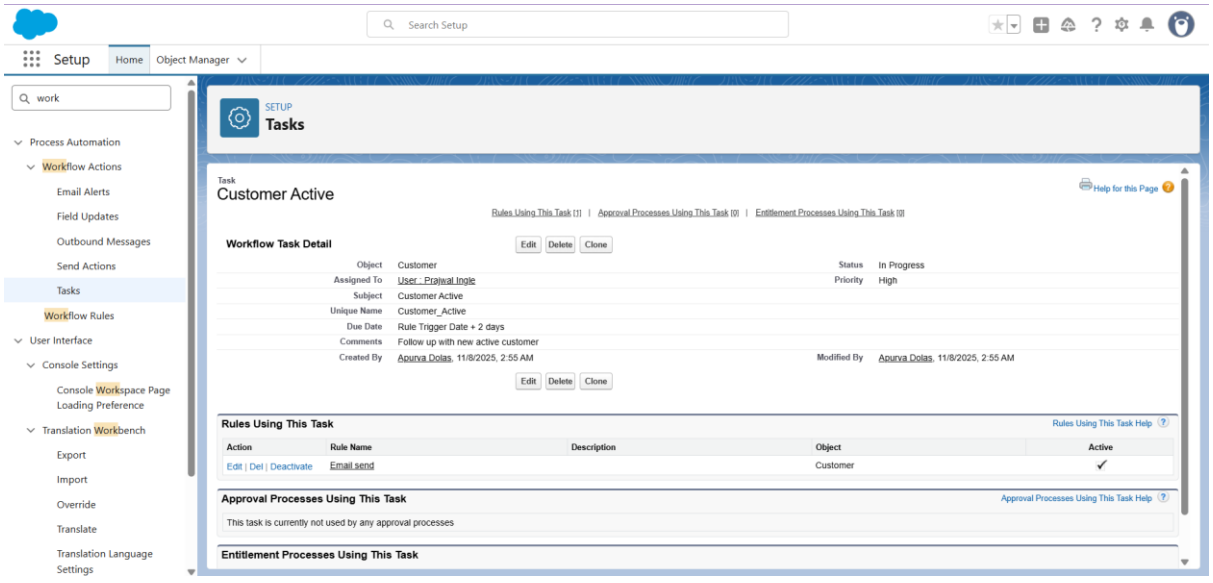
Task Details

- **Task Name: Customer Active**
  The task is automatically created when a customer's status becomes *Active* through the workflow rule.
- **Object: Customer__c**
  Indicates that this task is related to the Customer object and will be attached to the corresponding customer record.

- **Assigned To: User: Prajwal Ingle**
  Specifies that the task will be assigned to the user Prajwal Ingle, who is responsible for following up with newly activated customers.
- **Status: In Progress**
  The task is automatically set to *In Progress* when created, indicating that follow-up action is required.
- **Priority: High**
  The priority level is set to *High* to ensure that the task receives immediate attention.
- **Subject: Customer Active**
  The subject line clearly identifies the purpose of the task — to follow up with a customer whose account has just been activated.
- **Unique Name: Customer_Active**
  This is the system-generated unique name used internally to identify the task configuration.
- **Due Date: Rule Trigger Date + 2 Days**
  The task is automatically due two days after the workflow rule is triggered, allowing time for a timely follow-up.
- **Comments: Follow up with new active customer**
  This provides clear instructions for the assigned user, emphasizing the need to engage with the customer after activation.

Purpose

This workflow task ensures that every new active customer receives timely attention and personalized communication from the team. It supports effective customer onboarding and helps maintain a high level of service quality within the MediSwift system.



➢ Approval Process

Order: Total Amount Exceeds ₹10,000

In the **MediSwift application**, an **Approval Process** is implemented on the **Order** object to ensure that high-value orders undergo proper review and authorization before final processing. This helps maintain financial control and prevents unauthorized order approvals.

Process Details

- **Process Name: Total Amount Exceeds ₹10,000**
  This approval process is triggered when an order's total amount exceeds ₹10,000.
- **Object: Order__c**
  The process is associated with the **Order** custom object in MediSwift.
- **Active: Yes**
  The approval process is currently active and functional.
- **Entry Criteria:**
  Order: Total_Amount__c > "INR 10,000"
  The process starts only when the total order amount is greater than ₹10,000.
- **Record Editability: Administrator ONLY**
  Once an order enters the approval process, only system administrators can edit it until the process completes.
- **Allow Submitters to Recall Approval Requests: Not Allowed**
  The submitter cannot recall the approval request once submitted.
- **Initial Submitter: Order Owner**
  The owner of the order record is responsible for submitting it for approval.

Initial Submission Actions

When the approval process is submitted:

- **Record Lock:** The order record is locked to prevent editing.
- **Field Update:** The Approval Status field is updated to "Pending" to indicate that the order is awaiting approval.

Approval Step

Step 1: Review by Approver

- **Step Name:** Step 1
- **Assigned Approver:** User: Pranav Kore
- **Description:** The order is routed to Pranav Kore for approval.
- **Reject Behavior:** Final Rejection – If the approver rejects the order, the process ends immediately with a rejection.

Final Approval Actions

When the order is approved:

- **Record Lock:** The order record remains locked to prevent further edits.
- **Field Update:** The Approval Status field is updated to “Approved.”

Final Rejection Actions

When the order is rejected:

- **Record Lock:** The record is unlocked to allow modifications if needed.
- **Field Update:** The Approval Status field is updated to “Rejected.”

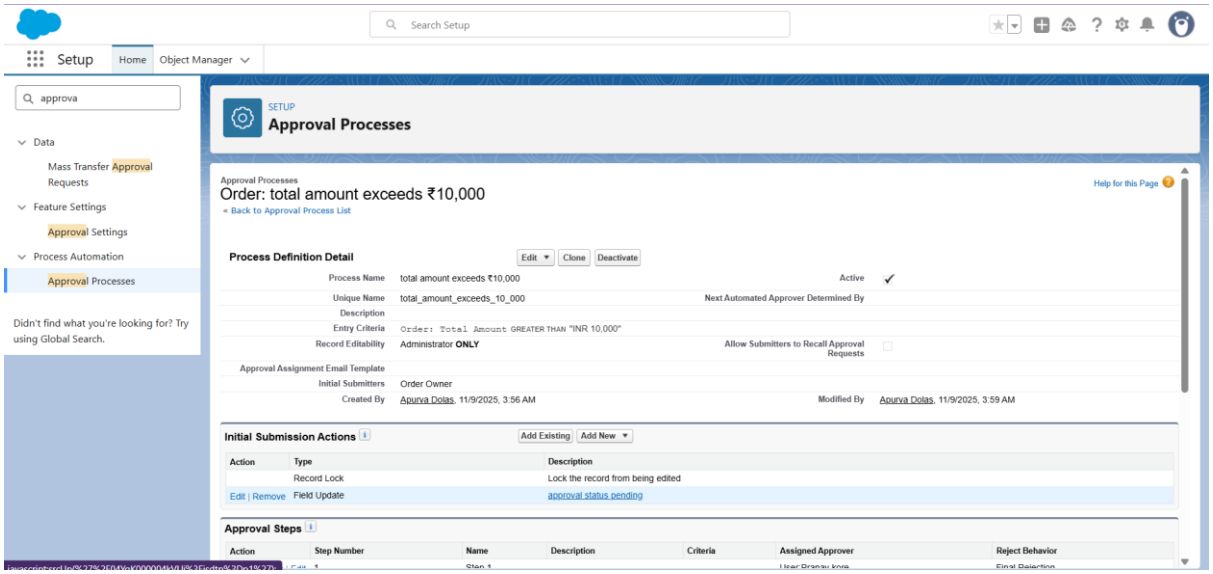
Recall Actions

If the approval process is recalled (by an administrator):

- **Record Lock:** The record is unlocked for editing.

Purpose

This approval process ensures that all orders exceeding ₹10,000 are verified by an authorized user before being finalized. It helps maintain accountability, prevents unauthorized processing, and aligns with MediSwift’s internal compliance standards for large transactions.



➢ Process Builder

The Process Builder in MediSwift automates key business processes by performing defined actions when specific conditions are met. It reduces manual effort, ensures data consistency, and helps maintain smooth workflow automation within the system.

Process Name: Update Order Status Automatically

Object: Order__c

This process is built on the Order object and is designed to automatically update the Order Status when certain criteria are met.

Process Details

- Process Type: Record Change Process
- Trigger: The process starts when a record is created or edited.

Criteria for Executing Actions

- Criteria Name: *Payment Completed*
- Condition:
  Payment_Status__c = 'Completed'
- Evaluation Criteria:
  Execute the actions only when specified changes are made to the record.
- Purpose:
  To automatically update the Order Status to *Delivered* once the payment has been successfully completed.

Immediate Actions

When the above criteria are met:

- Action Type: Field Update
- Field to Update: Order_Status__c
- New Value: Delivered

Purpose

This Process Builder ensures that once a payment is completed, the related order is automatically marked as Delivered without requiring manual intervention. It helps streamline the order fulfillment process, improve accuracy, and enhance customer satisfaction.

Benefits

- Eliminates manual data entry for order status updates
- Reduces processing time and human errors
- Improves workflow efficiency and reliability
- Ensures consistent synchronization between Payment and Order records

➢ Flow Builder

The Flow Builder in MediSwift automates complex business logic through visual workflows. It connects multiple objects and performs actions like record creation, updates, and email alerts — all without Apex code.

Flow Name: New Customer Registration Flow

Flow Type: Record-Triggered Flow

Flow Details

- **Object:** Customer__c
- **Trigger:** When a Customer record is created.
- **Flow Type: After Save** (runs automatically after record creation)
- **Purpose:** To automate actions when a new customer is added to the system.

Flow Criteria

Condition:
Customer_Status__c = 'Active'

Logic:
The flow runs only when a newly created customer is marked as Active.

Flow Actions

1. **Send Email Notification**
    - **Action Type:** Email Alert
    - **Template:** *Active Customer*
    - **Recipient:** Customer's registered email address
    - **Purpose:** To notify the customer that their account is now active.
2. **Update Field**
    - **Field to Update:** Email_Send_Status__c
    - **New Value:** "Email sent successfully"
    - **Purpose:** To track that the notification was sent.
3. **Create Follow-Up Task**
    - **Assigned To:** Sales/User
    - **Subject:** "Follow up with new active customer"
    - **Due Date:** 2 days from record creation
    - **Priority:** High

Purpose

This flow helps automate customer onboarding by:

- Sending a welcome email to active customers
- Updating status fields automatically
- Creating follow-up tasks for internal teams

Benefits

- Saves time by eliminating manual follow-ups
- Ensures consistent communication with customers
- Improves data accuracy and team productivity
- Provides a smooth, automated onboarding experience

Phase 5: Apex Programming (Developer)

➢ Classes & Objects

An **Apex Class** is a blueprint that defines the behavior and functionality of a system component.
Classes in MediSwift are used for performing complex logic such as sending notifications, updating records, or integrating with external systems.

Objects represent the data model of MediSwift.
Each object contains fields and relationships that store specific business information.

➢ Apex Triggers – CustomerTrigger

The CustomerTrigger is an Apex trigger implemented on the Customer__c custom object in Salesforce. Its primary purpose is to automatically send a personalized welcome email to every new customer upon creation. This ensures seamless communication and enhances customer engagement.

Trigger Overview

Trigger Name: CustomerTrigger
Object: Customer__c
Trigger Event: after insert

Description:
The trigger fires after a new Customer__c record is inserted. It loops through the newly created customer records, checks for a valid email address, and sends a personalized welcome email in bulk.

Trigger Context:

- **after insert:** Ensures that the record ID and other system-generated fields are available.
- **Bulk-safe design:** Can handle multiple records inserted at once.

➢ Apex Trigger Concepts

1. Trigger Events

Event	Description
before insert	Executes before saving new records
after insert	Executes after saving new records
before update	Executes before updating existing records
after update	Executes after updating existing records
before delete	Executes before deleting records
after delete	Executes after deleting records
after undelete	Executes after restoring records from recycle bin

2. Trigger Context Variables

Variable	Description
Trigger.new	List of new records being inserted or updated
Trigger.old	List of existing records before update or deletion
Trigger.isInsert	Boolean indicating trigger is running on insert
Trigger.isUpdate	Boolean indicating trigger is running on update
Trigger.isDelete	Boolean indicating trigger is running on delete

3. Collections in Apex

- **List:** Ordered collection, allows duplicates (List<Messaging.SingleEmailMessage> used for emails).
- **Set:** Unique collection, no duplicates.
- **Map:** Key-value pair collection (Map<Id, Customer__c>).

Collections allow bulk operations, reducing governor limit risks.

4. Control Statements

- for loops: Iterate over multiple records (e.g., Trigger.new).
- if statements: Conditional logic to validate data (e.g., checking if an email exists).

Trigger Code:

```
trigger CustomerTrigger on Customer__c (after insert) {

    List<Messaging.SingleEmailMessage> emails = new List<Messaging.SingleEmailMes
sage>();

    for (Customer__c c : Trigger.new) {
        if (c.Email__c != null) {
            Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
            email.setToAddresses(new String[] { c.Email__c });
            email.setSubject('Welcome to Mediswift — Your Trusted Healthcare Partner!');

            // 📧 Long formatted email body
            String body =
                'Dear ' + c.Full_Name__c + ',\n\n' +
                'Welcome to *Mediswift*! 🎉\n\n' +
                'We're thrilled to have you as part of our growing healthcare family.' +
                'Mediswift is designed to make your pharmacy experience simple, fast, and reli
able — ' +
                'whether you're ordering essential medicines, managing prescriptions, or tracki
ng your deliveries in real time.\n\n' +

                'Here's what you can do with your Mediswift account:\n' +
                '• Browse and order medicines easily from our verified pharmacies.\n' +
                '• Track your order and delivery status in real-time.\n' +
                '• View your past orders and manage payments securely.\n' +
                '• Contact our customer support team anytime you need assistance.\n\n' +

                'Getting Started:\n' +
                '1️⃣ Log in to your Mediswift account.\n' +
                '2️⃣ Explore our catalog of medicines and health products.\n' +
                '3️⃣ Place your first order — and let us handle the rest!\n\n' +

                '💡 Tip: Save your favorite medicines for quick reordering and enable notificat
ions for exclusive discounts.\n\n' +

                'If you ever have questions, feedback, or suggestions, feel free to reach out at *s
upport@mediswift.com*.\n\n' +

                'We're committed to making healthcare accessible, convenient, and trustworthy
for everyone.\n\n' +
                'Stay Healthy,\n' +
                '— The Mediswift Team\n\n' +
                '"Delivering Care. Swiftly."';

            email.setPlainTextBody(body);
            emails.add(email);
        }
```

```
    }

    if (!emails.isEmpty()) {
        Messaging.sendEmail(emails);
    }
}
```

1. Trigger Declaration

- Defines the trigger on the Customer__c object.
- Executes after insert to ensure the record exists in the database.

2. Email List Initialization

List<Messaging.SingleEmailMessage> emails = new
List<Messaging.SingleEmailMessage>();

- Creates a list to store multiple email messages.
- Supports bulkification by sending all emails in one call.

3. Loop Through Trigger Records

for (Customer__c c : Trigger.new) {

- Iterates over each new customer record.
- Ensures all records inserted in bulk are processed efficiently.

4. Conditional Check

if (c.Email__c != null) {

- Sends emails only to customers with a valid email address.
- Avoids runtime errors from null email fields.

5. Compose Email

Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();

email.setToAddresses(new String[] { c.Email__c });

email.setSubject('Welcome to Mediswift — Your Trusted Healthcare Partner!');

email.setPlainTextBody(body);

- Creates a personalized, plain-text email.
- setToAddresses() accepts an array of recipients, even for single emails.
- setSubject() sets the email's subject line.
- setPlainTextBody() sets the email content.

6. Add Email to List

emails.add(email);

- Stores the email in a list to send all emails in bulk.
- Avoids calling Messaging.sendEmail() inside a loop (prevents hitting governor limits).

7. Send Emails

```
if (!emails.isEmpty()) {

   Messaging.sendEmail(emails);

}
```

- Sends all collected emails in one operation.
- Ensures no empty email list is processed.

➢ Working of the Trigger (Step by Step)

- Trigger fires after new customers are inserted.
- Initializes an empty email list.
- Iterates through each newly created customer record.
- Checks if the customer has provided an email.
- Composes a personalized welcome email.
- Adds the email to the bulk email list.
- Sends all emails in a single bulk operation.

Analogy: The trigger works like a "welcome bot" that writes letters for each customer and delivers them together in one batch.

The CustomerTrigger efficiently automates the process of welcoming new customers via email. It is bulk-safe, simple, and aligned with Salesforce best practices. Future enhancements could include:

- Using HTML emails for richer formatting.
- Moving the email logic to a handler class for cleaner trigger design.
- Implementing error handling to log failed email sends.

➢ Batch Apex

Definition:
Batch Apex allows you to process large volumes of records asynchronously in smaller manageable chunks to avoid hitting governor limits.

Use Case:

- Processing millions of records for data cleanup.
- Mass updating or deleting records.
- Generating reports or archiving data.

Syntax:

```
global class CustomerBatch implements Database.Batchable<sObject> {


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

        return Database.getQueryLocator('SELECT Id, Name FROM Customer__c');

    }


    global void execute(Database.BatchableContext BC, List<Customer__c> scope) {

        for(Customer__c c : scope) {

            c.Status__c = 'Processed';

        }

        update scope;

    }


    global void finish(Database.BatchableContext BC) {

        System.debug('Batch Process Finished');

    }

}
```

Key Points:

- **start:** Query records to process.
- **execute:** Logic applied to each batch.
- **finish:** Post-processing logic (e.g., sending email notifications).
- Maximum 50 million records per batch.

➢ Queueable Apex

Definition:
Queueable Apex is a lightweight asynchronous job that allows chaining jobs and supports complex objects as parameters.

Use Case:

- Performing asynchronous operations with complex objects.
- Chaining multiple jobs in a sequence.

Syntax Example:

```
public class CustomerQueueable implements Queueable {

   public void execute(QueueableContext context) {

      List<Customer__c> customers = [SELECT Id FROM Customer__c WHERE Status__c='New'];

      for(Customer__c c : customers) {

         c.Status__c = 'Processed';

      }

      update customers;

   }

}



// Enqueue the job

ID jobId = System.enqueueJob(new CustomerQueueable());
```

Key Points:

- Supports chaining via System.enqueueJob(new NextJob()); in execute().
- More flexible than Future methods.

➢ Scheduled Apex

Definition:
Scheduled Apex allows you to schedule Apex classes to run at specific times using the Schedulable interface.

Use Case:

- Daily data cleanup.
- Sending periodic email notifications.
- Batch processing on schedule.

Syntax Example:

```
global class DailyCustomerJob implements Schedulable {

   global void execute(SchedulableContext sc) {

      List<Customer__c> newCustomers = [SELECT Id FROM Customer__c WHERE Status__c='New'];

      for(Customer__c c : newCustomers) {

         c.Status__c = 'Processed';

      }

      update newCustomers;

   }

}



// Schedule via Apex

String cronExp = '0 0 2 * * ?'; // Daily at 2 AM

System.schedule('Daily Customer Job', cronExp, new DailyCustomerJob());
```

Key Points:

- Cron expression determines schedule.
- Can combine with Batch Apex for large data sets.

➢ Future Methods

Definition:
Future methods run asynchronously in the background. Useful for callouts or long-running operations that don't need immediate completion.

Syntax Example:

```
public class CustomerService {

   @future(callout=true)

   public static void sendWelcomeEmail(List<Id> customerIds) {

      List<Customer__c> customers = [SELECT Id, Email__c FROM Customer__c WHERE Id IN :customerIds];

      for(Customer__c c : customers) {

         // Send email logic

      }

   }

}



// Call future method

CustomerService.sendWelcomeEmail(new List<Id>{'001xx000003DHP0AAO'});
```

Key Points:

- Must be static and void.
- Limited to 50 calls per transaction.
- No guarantee of immediate execution.

➢ Exception Handling

Definition:
Exception handling ensures that code handles unexpected errors gracefully using try-catch blocks.

Syntax Example:

```
try {
   List<Customer__c> customers = [SELECT Id FROM Customer__c];
   update customers;
} catch(DmlException e) {
   System.debug('Error updating customers: ' + e.getMessage());
} finally {
   System.debug('Execution completed');
}
```

Key Points:

- try block: Code that might throw an exception.
- catch block: Handle the error.
- finally block: Runs regardless of success or failure.
- Helps prevent unhandled errors in asynchronous jobs.

➢ Test Classes

Definition:
Test classes are Apex classes that validate the correctness of other classes or triggers. Salesforce requires at least 75% code coverage for deployment.

Syntax Example:

```
@isTest
public class CustomerTriggerTest {
   @isTest static void testCustomerTrigger() {
      Customer__c c = new Customer__c(Name='Test Customer', Email__c='test@example.com');
      insert c;

      // Verify trigger logic
```

```
        List<Customer__c> customers = [SELECT Id, Name FROM Customer__c
    WHERE Id=:c.Id];

        System.assertEquals('Test Customer', customers[0].Name);

    }

}
```

Key Points:

- Use @isTest annotation.
- Create test data inside the test method.
- Test all scenarios: success, failure, bulk operations.

➢ Asynchronous Processing in Salesforce

Definition:
Asynchronous processing allows long-running or bulk operations to run in the background, avoiding governor limits and improving user experience.

Types:

1. **Batch Apex:** Large record processing.
2. **Queueable Apex:** Lightweight asynchronous jobs.
3. **Scheduled Apex:** Runs Apex on a schedule.
4. **Future Methods:** Background processing, especially for callouts.

Best Practices:

- Always bulkify code.
- Avoid DML and SOQL inside loops.
- Use try-catch to handle exceptions.
- Prefer Queueable Apex over Future methods for flexibility.

➢ SOQL & SOSL

1. Salesforce Object Query Language (SOQL)

Definition:
SOQL is used to query records from a single object or multiple related objects in Salesforce. It is similar to SQL but optimized for Salesforce's multi-tenant environment.

Purpose:

- Retrieve records based on conditions.
- Support reporting, triggers, Apex logic, and integrations.

Key Features:

- Works on one object at a time.
- Can query related objects using relationship queries.
- Supports aggregate functions like COUNT(), SUM(), AVG().

Basic Syntax:

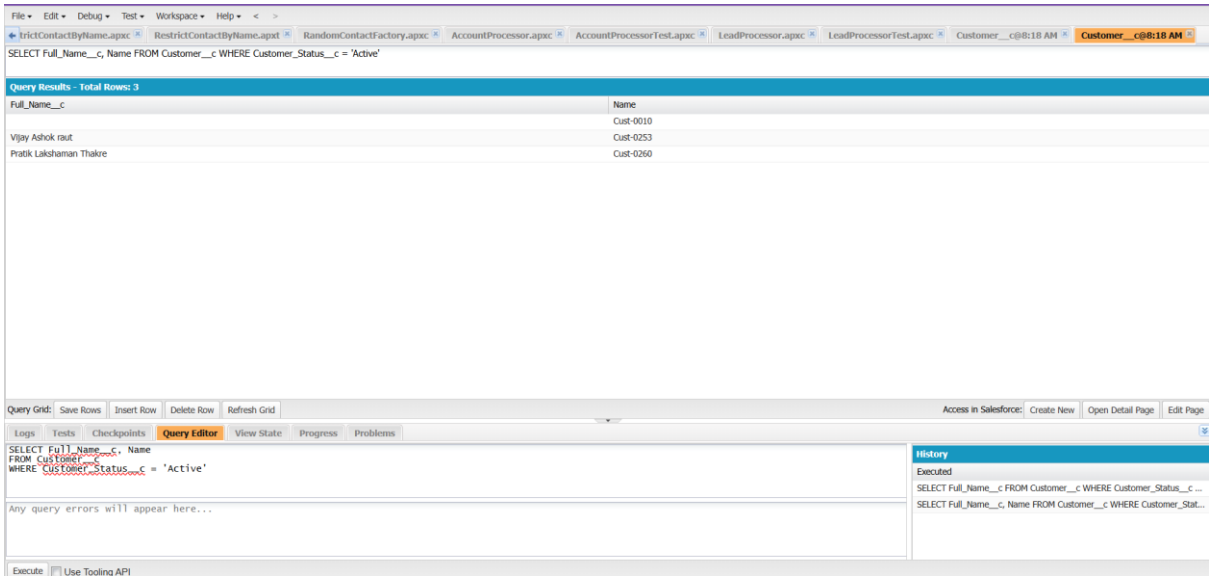
SELECT field1, field2, field3

FROM ObjectName

WHERE condition

ORDER BY field ASC|DESC

LIMIT number



2. Salesforce Object Search Language (SOSL)

Definition:
SOSL is used to search text, email, and phone fields across multiple objects in Salesforce. It is optimized for full-text searches.

Purpose:

- Locate records across multiple objects with a single query.
- Return records quickly based on keywords.

Key Features:

- Can search across multiple objects at once.
- Supports wildcards and exact matches.
- Returns lists of lists, grouped by object.

Basic Syntax:

FIND 'searchString'

IN ALL FIELDS

RETURNING Object1(field1, field2), Object2(field1, field2)

Note: Phases 6 (User Interface Development) and 7 (Integration & External Access) are intentionally omitted from this documentation as they are out of the current project scope.

Phase 8: Data Management & Deployment

➢ Data Import Wizard

Definition:
The Data Import Wizard is a point-and-click tool in Salesforce that allows users to import data for standard and custom objects without coding.

Features:

- Supports objects like Accounts, Contacts, Leads, and custom objects.
- Allows upsert operations (insert new and update existing records).
- Supports CSV file upload.
- Provides mapping interface to match CSV columns with Salesforce fields.

Use Case:

- Importing small to medium datasets (<50,000 records).
- Quick imports without requiring technical expertise.

Steps:

1. Log in to Salesforce and go to Setup → Data Import Wizard.
2. Select the object to import (e.g., Accounts, Contacts, Leads, or Custom Objects).
3. Upload your CSV file.
4. Map CSV columns to Salesforce fields.
5. Choose operation type:
    - Insert (new records)
    - Update (existing records)
    - Upsert (insert or update)
6. Review mappings and click Start Import.
7. Monitor import progress and download the import results for errors or success.

➢ Data Loader

Definition:
Data Loader is a client application used to insert, update, delete, and export Salesforce records in bulk.

Features:

- Supports large volumes of data (>50,000 records).
- Command-line interface for automated tasks.
- Supports upsert, delete, hard delete, and export operations.
- Works with CSV files.

Use Case:

- Migrating large datasets from other systems.
- Bulk updating records with automation.
- Regular data cleanup and backup.

Steps:

1. Download and install Salesforce Data Loader.
2. Open Data Loader and log in with your Salesforce credentials.
3. Select operation:
    - Insert – create new records
    - Update – modify existing records
    - Upsert – insert or update
    - Delete / Hard Delete – remove records
    - Export / Export All – extract data from Salesforce
4. Choose the object (e.g., Customer__c).
5. Browse and select CSV file.
6. Map CSV columns to Salesforce fields.
7. Run operation and check the success and error logs.

➢ Duplicate Rules

Definition:
Duplicate rules in Salesforce prevent or alert users when duplicate records are being created.

Components:

- Matching Rules: Define criteria to identify duplicates (e.g., same email, same phone number).
- Duplicate Rules: Define actions when a duplicate is detected:
    - Block record creation.
    - Allow creation with alert.

Use Case:

- Maintaining data quality.
- Avoiding duplicate Leads, Contacts, or Accounts.

Steps:

1. Go to Setup → Duplicate Rules.
2. Click New Rule and select the object (e.g., Leads, Contacts).
3. Define matching criteria (e.g., Email, Phone, Name).
4. Set actions when duplicates are found:
    - Block the creation
    - Allow with alert
5. Activate the rule.
6. Test by trying to create a duplicate record

➢ Data Export & Backup

Definition:
Salesforce provides tools for exporting and backing up data to ensure business continuity and compliance.

Options:

- Weekly Data Export: Scheduled export of all data and attachments.
- Data Loader Export: Manual export of selected objects.
- Third-party backup tools for additional security.

Best Practices:

- Regularly back up critical objects.
- Store exports securely outside Salesforce.
- Test backup restores processes periodically.

Steps:

1. Go to Setup → Data Export.
2. Click Export Now or Schedule Export (weekly/monthly).
3. Select objects and data to export.
4. Choose Include Attachments / Documents if required.
5. Click Start Export.
6. Download the exported ZIP files securely.

➢ Change Sets

Definition:
Change Sets allow metadata migration between Salesforce orgs (e.g., Sandbox → Production).

Features:

- Point-and-click deployment of components like:
  - Objects, Fields, Validation Rules, Apex classes, Triggers, Workflows.
- Requires connected orgs.

Limitations:

- Only works between related orgs (Sandbox & Production).
- Cannot deploy all components (some require ANT or VS Code).

Use Case:

- Deploying tested components from Sandbox to Production.
- Small to medium deployments.

Steps:

1. Go to Setup → Outbound Change Sets in source org (Sandbox).
2. Click New and enter Name and Description.
3. Add components to the change set:
   - Objects, fields, validation rules, Apex classes, triggers, workflows.
4. Upload change set to the target org (Production).
5. In the target org, go to Inbound Change Sets, validate, and deploy.
6. Monitor deployment status.

➢ Unmanaged vs Managed Packages

Type	Description
Unmanaged Package	Editable, used for open-source distribution or learning. Changes are permanent in target org.
Managed Package	Created by Salesforce ISV partners, upgradeable, has namespace prefix, protected components.

Use Cases:

- Unmanaged: Sample apps, internal deployments, learning exercises.
- Managed: AppExchange apps, controlled distribution, version upgrades.

Steps:

1. **Unmanaged Package:**

   - Go to Setup → Packages → New.
   - Add components (objects, fields, Apex, etc.).
   - Distribute package via installation link.
   - Editable in target org.

2. **Managed Package:**

   - Use Developer Edition org to create the package.
   - Add components and namespace prefix.
   - Upload to AppExchange or install via link.
   - Version upgrades supported, components partially protected.

➢ ANT Migration Tool

Definition:
ANT Migration Tool is a command-line utility provided by Salesforce for metadata deployment using XML and Java.

Features:

- Supports deployment, retrieval, and destructive changes.
- Ideal for CI/CD automation.
- Works with version control systems like Git.

Use Case:

- Automated deployments for complex orgs.
- Migration of components not supported by change sets.

Steps:

1. Download ANT Migration Tool from Salesforce.
2. Install Java and set environment variables.
3. Configure build.properties with Salesforce credentials.
4. Create package.xml to specify metadata components.
5. Use ant retrieve to get metadata from source org.
6. Use ant deploy to push metadata to target org.
7. Review deployment logs for errors.

➢ VS Code & SFDX

Definition:
VS Code with Salesforce DX (SFDX) is a modern development environment for Salesforce, supporting source-driven development.

Features:

- Salesforce CLI: Deploy, retrieve, and test metadata from VS Code.
- Scratch Orgs: Temporary orgs for development and testing.
- Version control integration: Git-based workflows.
- Extension Pack: Apex, Lightning Web Components, Visualforce support.

Use Case:

- Modern Salesforce development with CI/CD.
- Source-driven deployment for team collaboration.
- Testing and development in scratch orgs before production deployment.

Steps:

1. Install VS Code and Salesforce Extension Pack.
2. Install Salesforce CLI (SFDX).
3. Authenticate with Salesforce org: sfdx auth:web:login
4. Create or open a Salesforce DX project in VS Code.
5. Use commands to:

- Retrieve metadata: sfdx force:source:retrieve
- Deploy metadata: sfdx force:source:deploy
- Run tests: sfdx force:apex:test:run

6. Use scratch orgs for development:
7. Use Git for version control integration.

Phase 9: Reporting, Dashboards & Security Review

➢ Reports

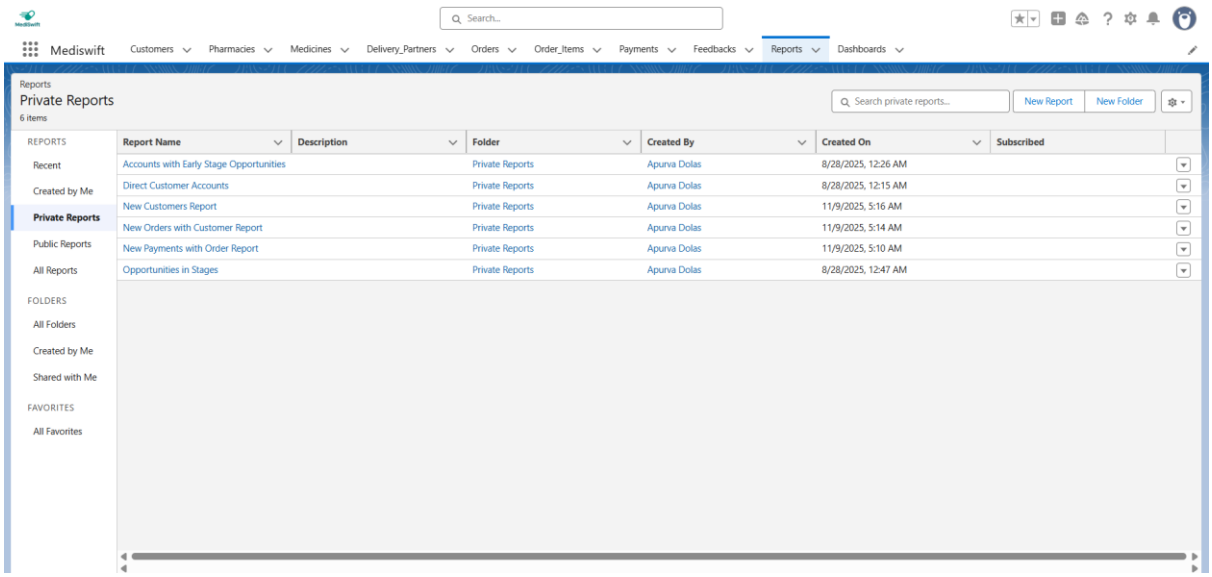
Definition:
Reports allow you to analyze Salesforce data in different formats to support business decisions.

Types of Reports:

Report Type	Description
Tabular	Simple list of records; best for exporting or viewing a list of data.
Summary	Groups data by rows; allows subtotals and aggregations.
Matrix	Groups data by rows and columns; ideal for comparing data across two dimensions.
Joined	Combines multiple report blocks for related data; allows viewing different perspectives in a single report.

Steps to Create a Report:

1. Navigate to Reports → New Report.
2. Select the report type (standard or custom).
3. Add filters to limit data (e.g., Status = Active).
4. Add columns or grouping fields.
5. Run the report to preview data.
6. Save and optionally schedule future runs.



➢ Report Types

Definition:
Report Types define which objects and fields are available in a report.

Types:

- **Standard Report Types:** Predefined for standard objects (Accounts, Contacts, Opportunities).
- **Custom Report Types:** Created to include custom objects or specific relationships.

Steps to Create Custom Report Type:

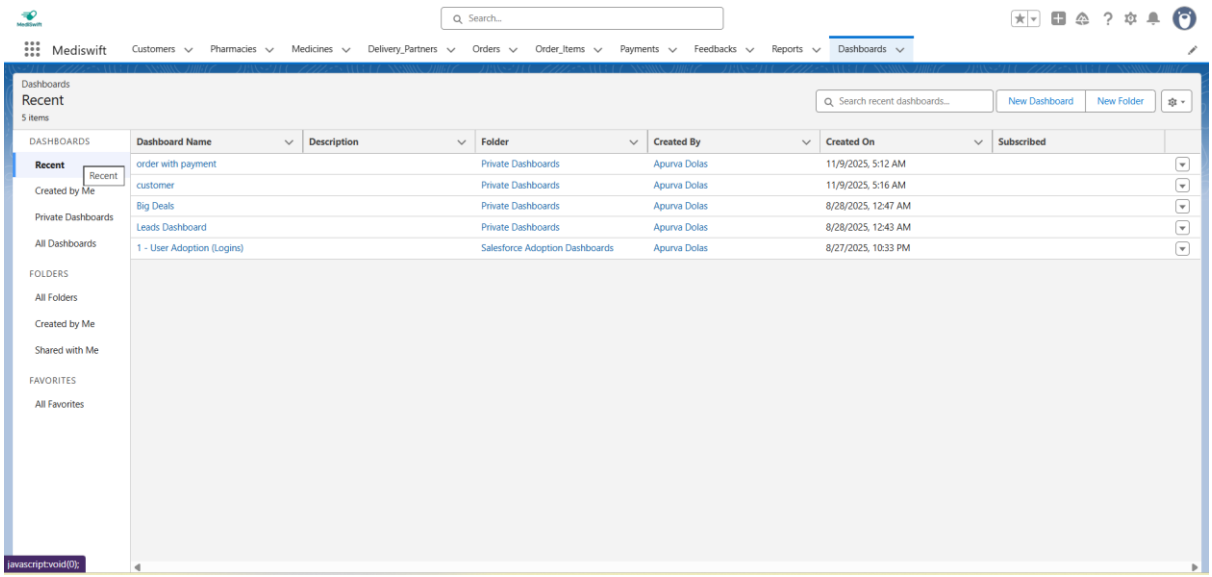
1. Go to Setup → Report Types → New Custom Report Type.
2. Select Primary Object.
3. Define relationships with other objects (e.g., Accounts with Contacts).
4. Choose deployment status (In Development / Deployed).
5. Save and use it when creating reports.

➢ Dashboards

Definition:
Dashboards visually display report data in charts, graphs, and tables.

Steps to Create a Dashboard:

1. Navigate to Dashboards → New Dashboard.
2. Select folder and provide dashboard name.
3. Add components (charts, gauges, tables) and select source reports.
4. Configure filters and component properties.
5. Save and run the dashboard.



➢ Dynamic Dashboards

Definition:
Dynamic dashboards allow a single dashboard to display data based on the running user.

Steps to Create a Dynamic Dashboard:

1. Open an existing dashboard or create a new one.
2. Click View Dashboard As → Dynamic.
3. Select Run as logged-in user.
4. Save and run.

Benefits:

- Eliminates the need for multiple dashboards for different users.
- Maintains security by respecting user access levels.

➢ Sharing Settings

Definition:
Sharing settings control record-level access in Salesforce.

Key Types:

- Organization-Wide Defaults (OWD): Base access level for objects (Public/Private).
- Role Hierarchy: Access based on hierarchy roles.
- Sharing Rules: Automatic sharing based on criteria.
- Manual Sharing: User-specific access grants.

Steps to Configure Sharing Settings:

1. Go to Setup → Sharing Settings.
2. Select object and modify default access.
3. Create criteria-based sharing rules if needed.
4. Test access with different user roles.

➢ Field Level Security

Definition:
Field-level security determines which users can view or edit individual fields.

Steps:

1. Navigate to Setup → Object Manager → Object → Fields & Relationships → Set Field-Level Security.
2. Select profiles to restrict or allow read/edit access.
3. Save changes.

Best Practices:

- Hide sensitive fields for lower-privilege users.
- Enforce data protection compliance requirements.

➢ Session Settings

Definition:
Session settings control security, timeout, and login behavior for Salesforce users.

Steps:

1. Go to Setup → Session Settings.
2. Configure:
    - Session timeout (e.g., 30 minutes of inactivity).
    - Force logout on browser close.
    - Session security levels.
3. Save changes.

➢ Login IP Ranges

Definition:
Login IP ranges restrict Salesforce access to specific network IPs.

Steps:

1. Go to Setup → Profiles → Login IP Ranges.
2. Add start and end IP addresses.
3. Save and enforce access restrictions.

Benefits:

- Prevent unauthorized access.
- Enhance organizational security compliance.

➢ Audit Trail

Definition:
Login IP ranges restrict Salesforce access to specific network IPs.

Steps:

1. Go to Setup → Profiles → Login IP Ranges.
2. Add start and end IP addresses.
3. Save and enforce access restrictions.

Benefits:

- Prevent unauthorized access.
- Enhance organizational security compliance.

Phase 10: Final Presentation & Demo Day

➢ Pitch Presentation

Purpose:
To present the Mediswift project objectives, solutions, and outcomes to stakeholders, mentors, or peers.

Content Overview:

- **Project Goal:** Build a comprehensive Salesforce-based solution for **Mediswift**, enabling efficient customer onboarding, pharmacy order management, and real-time tracking.
- **Key Features Highlighted:**
  - Automated welcome emails for new customers via Apex triggers.
  - Data management using Data Loader, import/export, and duplicate rules for accurate records.
  - Reports and dashboards for monitoring active customers, orders, and pharmacy performance.
  - Secure access management with sharing rules, field-level security, and session settings.
- **Business Impact:** Reduced manual effort, improved data quality, and enabled real-time customer engagement.

➢ Demo Walkthrough

Purpose:
To demonstrate the Mediswift Salesforce system in action, showing real-world functionality.

Demo Highlights:

1. **Customer Onboarding:**
   - Insert a new Customer__c record.
   - Show trigger sending welcome email automatically.
2. **Active Customer Reports:**
   - Run SOQL-based reports to display active customers.
   - Show dashboards summarizing customer status and order tracking.
3. **Data Management:**
   - Demonstrate importing new customer data via Data Import Wizard.
   - Show duplicate rules preventing duplicate customer entries.
4. **Security Review:**
   - Display field-level security for sensitive fields.
   - Verify login IP restrictions and session settings.

➢ Feedback Collection

Purpose:
To gather stakeholder inputs for future improvements in Mediswift.

Methods Used:

- Feedback form with ratings on usability, features, and overall system performance.
- Discussion session during demo to capture qualitative insights.

Key Feedback Collected:

- Appreciation for automated workflows and email triggers.
- Suggestions for enhanced reporting and dashboard filters.
- Requests for integration with pharmacy inventory systems in future phases.

➢ Handoff Documentation

Purpose:
To ensure continuity and maintainability of Mediswift Salesforce implementation.

Included in Handoff:

- Apex triggers, classes, and batch/queueable processes.
- Flow builder and process automation documentation.
- Reports, dashboards, and their source report types.
- Security configuration: sharing rules, field-level security, session settings.
- Step-by-step setup and deployment instructions for admins.