A Project Report

on

SALESFORCE IMPLEMENTATION FOR THE EDTECH INDUSTRY

Submitted as Part of

iLearnings Global Internship Program, Cohort 2 ***(1st Oct 24 - 30 Dec 24)***

by

MUJAWAR MOHD KHIZAR
Salesforce Developer Intern, iLearnings

Under The Guidance

Of

Ms. TANU PRIYA SRIVASTAVA
Project Head, iLearnings

iLearnings Career & Consulting Pvt Ltd

Pune, India

2024

TABLE OF CONTENTS

ABSTRACT ……(1)

1. INTRODUCTION TO SALESFORCE

1.1 Get familiar with Salesforce interface and key features

- Complete basic tutorials and set up personal developer accounts.

1.2 Create a Developer Org

- Name: *iLearnings Test Org*
- Implement all functionalities there.

1.3 Create an App

- App Name: *iLearnings*

2. DATA MODEL

2.1 Create Custom Objects

- Objects: Employee, Clients, Trainer

2.2 Create Custom Fields and Relationships

2.3 Page Layouts

3. DATA MANAGEMENT

3.1 Create Record Types on Leads

3.2 Import Test Records

- Use: *Data Import Wizard*

4. CAMPAIGN AND LEAD

4.1 Create *iLearnings* Campaign

4.2 Add Test Leads to Campaign

4.3 Create a Web-to-Lead Form

- Purpose: Capturing leads
- Form Name: *Enrollment Form*

AUTOMATION

5. Email Templates and Alerts

5.1 **Scenario 1**:

- Upon Web-to-Lead form submission or Lead creation, notify the sales team and candidate.

5.2 **Scenario 2:**

- Send an email with the training schedule attached 1 day before the batch start date.

5.3 **Scenario 3:**

- Send promotional emails related to offers.
- When a Lead is created, send a "Buy 1, Get 1" offer email.

5.4 **Scenario 4:**

- Send birthday emails to trainers and employees.

6. Flows -

Create Flows for Campaign Members:

6.1 **Scenario 1:**

- 24 hours before the webinar (based on Webinar Date on Campaign Member), send "1 Day Before Webinar" email template.

6.2 **Scenario 2:**

- 1 hour before the Webinar Date and Time, send "1 Hour Before" email template.

6.3 **Scenario 3:**

- 5 minutes before the webinar starts, send "We Are Live" email template.

Build complex flows to automate multi-step processes and enhance user interactions:

6.3 **Scenario 4:**

- 1 hour after the webinar (on Campaign Members)

6.3 **Scenario 5:**

- 6 hours after the webinar (on Campaign Members)

6.3 **Scenario 6:**

- 24 hours after the webinar (on Campaign Members)

7. VALIDATION RULES

- Learn about validation rules and their role in ensuring data quality.

7.1 Create Validation Rules to Enforce the Following Business Logic:

- If **Registration Status = Successful**:

- Fields **Registration Date**, **Fees Paid by Candidate**, and **Batch Start Date** are mandatory.

- Learn troubleshooting techniques for Salesforce automation issues, practice error identification and resolution, and perform QA on previous work.

8. LIGHTNING WEB COMPONENT (LWC)

- Develop a Lightning Web Component to **Delete Duplicate Attachments**.

9. APEX

9.1 **Scenario 1:**

- **Account Trigger:** Create default contacts equal to the number of employees.

9.2 **Scenario 2:**

- **Opportunity Trigger:** Enforce future closed dates.

9.3 **Scenario 3:**

- **Product Trigger:** Set default pricebook entry with price 15.

9.4 **Scenario 4:**

- **Contact Trigger:** Update account name by appending last name.

9.5 **Scenario 5:**

- **Enhanced Contact Trigger:** Remove last name from account name upon contact deletion.

10. REPORTING & DASHBOARDS

10.1 Reports:

- Total Fees of Candidates.
- Monthly Revenue.
- Webinar Registrations.
- Lead Conversion Ratio.

10.2 Dashboards:

- Sales Dashboard.
- Trainer Dashboard.
- Employee Dashboard.

11. APEX

11.1 **Scenario 6:**

- **Opportunity Trigger:** Trigger on Opportunity to Alert users when closing an Opportunity without Line Items.

11.2 **Scenario 7:**

- **Opportunity Trigger:** Trigger on Opportunity to Increment “No of Products Sold” when Opportunity is closed-won.

11.3 **Scenario 8:**

- **Opportunity Trigger:** Trigger on Opportunity to Add all Account Contacts to Opportunity Contact Roles during creation.

11.4 **Scenario 9:**

- **Campaign Trigger:** Trigger on Campaign to Close Opportunities as “Won” or “Lost” based on Line Items when a Campaign is completed.

11.5 **Scenario 10:**

- **Contact Trigger:** Trigger on Contact to Copy Account’s Billing Address to Contact’s Mailing Address when enabled.

11. CAMPAIGN AUTOMATION

- Automate marketing campaigns.
- Track campaign performance and analyze results.

12. THIRD-PARTY INTEGRATIONS

12.1 Implement a salary calculator for employees.

12.2 Enable calling functionality for leads.

12.3 Explore payment integration options.

12.4 **Automate:**

- Salary slip emails to employees.
- Certificates to leads upon course completion.

13. FINAL DELIVERABLES

13.1 **Final Presentation:** Findings and recommendations for Salesforce efficiency at iLearnings.

13.2 **Group Discussion:** Reflect on internship experience and outcomes.

13.3 **Internship Review:** Feedback collection and career advice.

ABSTRACT

The project "Salesforce Implementation for the EdTech Industry" focuses on the implementation of Salesforce, a leading Customer Relationship Management (CRM) platform, to address the unique operational challenges of the EdTech industry. Designed to enhance efficiency across student enrollment, lead management, marketing, and analytics, this initiative delivers a comprehensive, data-driven solution tailored to the sector's needs. Key objectives include streamlining the lead-to-enrollment process, automating repetitive tasks, and enabling targeted marketing campaigns through robust data insights.

The implementation encompasses a wide range of functionalities, including custom data models, campaign automation, validation rules, and advanced workflows powered by Salesforce Flows. Automation scenarios address critical business requirements, such as notifying stakeholders upon lead creation, managing webinar communications, and personalizing outreach based on user behavior. Additionally, the project features sophisticated components like Lightning Web Components (LWC) for attachment management and Apex triggers to enforce business logic and improve operational consistency.

By leveraging Salesforce's advanced reporting and dashboard capabilities, the project enables detailed analytics on metrics such as student demographics, enrollment trends, and revenue growth. Third-party integrations, such as a salary calculator and payment functionality, further extend the platform’s utility, ensuring seamless interaction between educational operations and administrative tasks.

This project demonstrates technical proficiency in Salesforce development and a deep understanding of EdTech-specific requirements. By combining automation, personalization, and actionable analytics, the solution empowers the EdTech company to optimize processes, enhance customer relationships, and deliver a superior experience to students and stakeholders alike.

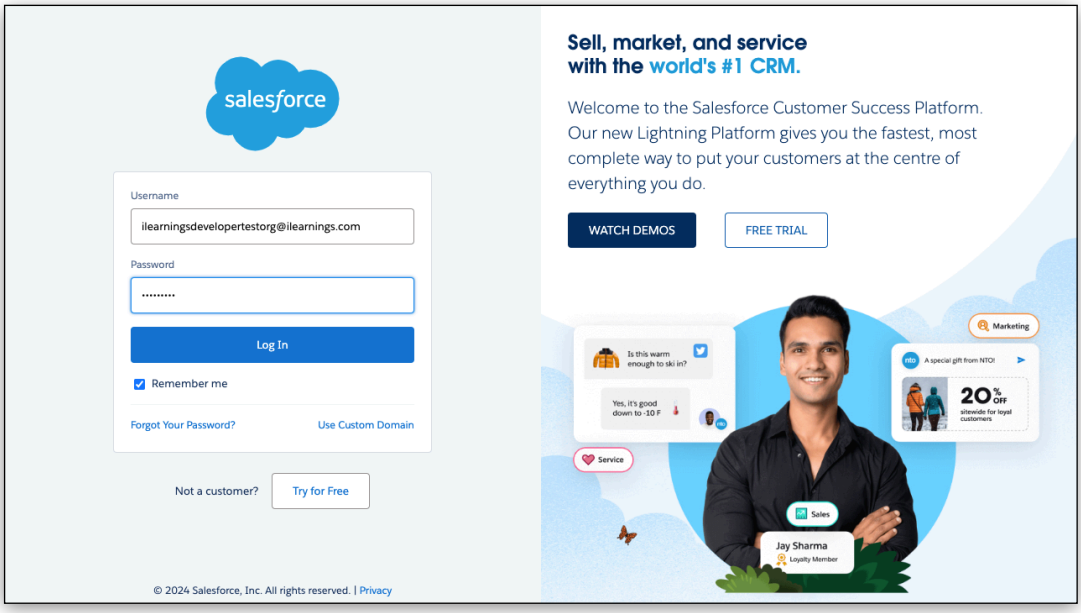
INTRODUCTION TO SALESFORCE

1.1 Get familiar with Salesforce interface and key features

- Complete basic tutorials and set up personal developer accounts.

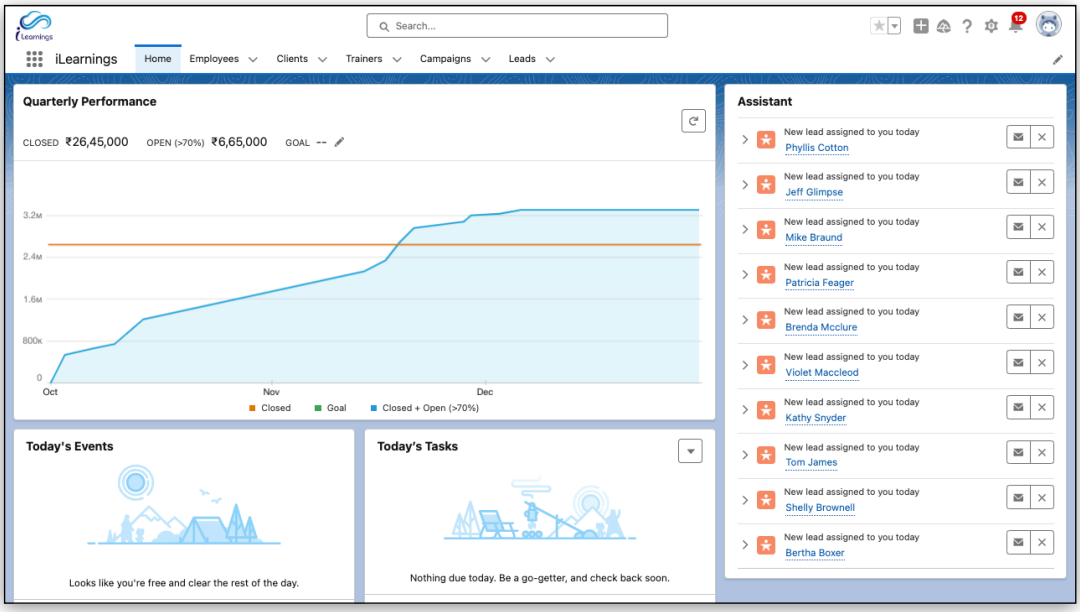
1.2 Create a Developer Org

- Name: *iLearnings Test Org*
- Implement all functionalities there.



1.3 Create an App

- **App Name:** *iLearnings*



DATA MODEL

2.1, 2.2 Create Custom Objects & Custom Fields and Relationships

Objects: Employee, Clients, Trainer

1. Employee Object

Create a custom object named **Employee** and add the following fields:

Field Label	Field Name	Data Type	Additional Notes
Employee Name	Employee_Name__c	Text	
Designation	Designation__c	Text	
Salary	Salary__c	Currency	
Incentives	Incentives__c	Currency	
Birthdate	Birthdate__c	Date	
Joining Date	Joining_Date__c	Date	
Active	Active__c	Checkbox	
Last Working Date	Last_Working_Date__	Date	
Address	Address__c	Text Area	
Payment Made	Payment_Made__c	Formula (Currency)	Formula: Salary__c + Incentives__c
Payment Date	Payment_Date__c	Date	
Mode of Payment	Mode_of_Payment__c	Picklist	E.g., Bank Transfer, Cheque, Cash
Meeting Link	Meeting_Link__c	URL	

2. Trainer Object

Create a custom object named **Trainer** and add the following fields:

Field Label	Field Name	Data Type	Additional Notes
Trainer Name	Trainer_Name__c	Text	
Trainer Email	Trainer_Email__c	Email	
Trainer Phone	Trainer_Phone__c	Phone	
Address	Address__c	Text Area	
Specializations	Specializations__c	Text Area	
Current Course Assigned	Current_Course__c	Text	

Commercials	Commercials__c	Currency	
Payment Released	Payment_Released__c	Currency	
Payment Date	Payment_Date__c	Date	
Payment Mode	Payment_Mode__c	Picklist	E.g., Bank Transfer, Cheque
Advance Payment	Advance_Payment__c	Currency	
Active Batch	Active_Batch__c	Text	
Mode of Training	Mode_of_Training__c	Picklist	Values: Offline,
Client Name	Client_Name__c	Lookup (Client)	
Birthdate	Birthdate__c	Date	
Joining Date	Joining_Date__c	Date	
Incentives	Incentives__c	Currency	

Section: Add a section for **Payment Fields** and group payment-related fields.

3. Client Object

Create a custom object named **Client** and add the following fields:

Field Label	Field Name	Data Type	Additional Notes
Client Name	Client_Name__c	Text	
Contact Number	Contact_Number__c	Phone	
Email ID	Email_ID__c	Email	
Address	Address__c	Text Area	

4. Campaign Object

Enhance the **Campaign** object with the following fields:

Field Label	Field Name	Data Type	Additional Notes
Webinar Date & Time	Webinar_Date_Time__c	Date/Time	
Webinar End Date & Time	Webinar_End_Date_Time__c	Date/Time	
Duration	Duration__c	Number	
Course Name	Course_Name__c	Text	
Perks	Perks__c	Text Area	
Course Link	Course_Link__c	URL	

5. Leads Object

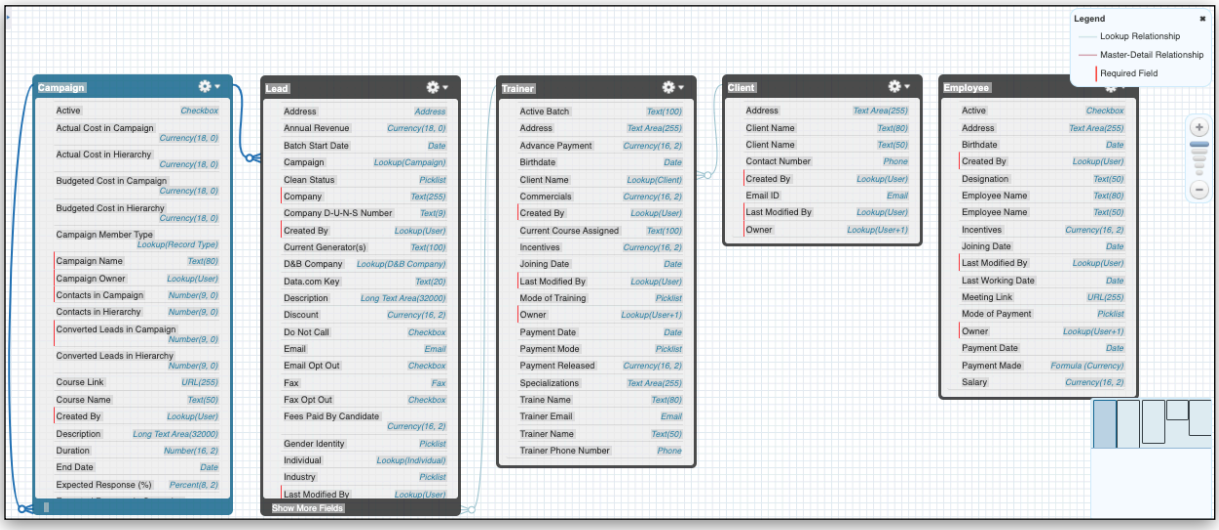
Add the following fields to the **Lead** object:

Field Label	Field Name	Data Type	Additional Notes
Trainer	Trainer__c	Lookup (Trainer)	
Registration Date	Registration_Date__c	Date	
Total Fees	Total_Fees__c	Currency	
Discount	Discount__c	Currency	
Fees Paid by	Fees_Paid__c	Currency	
Pending Fees	Pending_Fees__c	Formula (Currency)	Formula: Total_Fees__c - Fees_Paid__c
Batch Start Date	Batch_Start_Date__c	Date	

Additional Configuration:

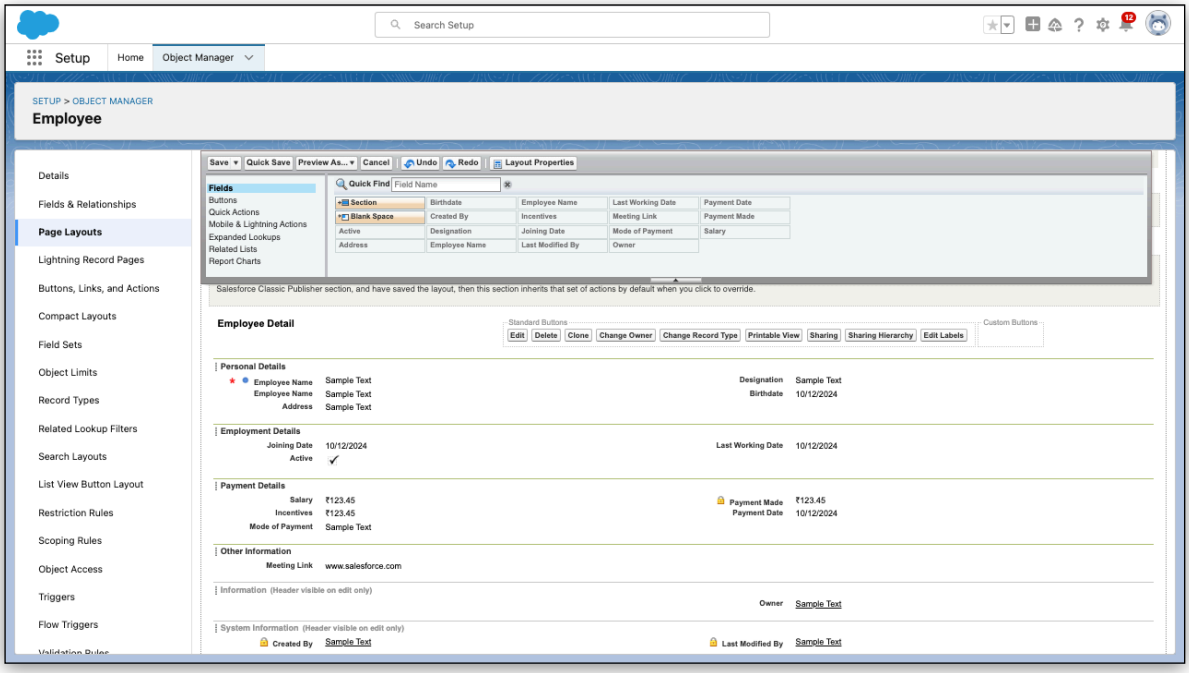
- Add a new picklist value: **Registration Successful** to the **Lead Status** field.

Schema Builder

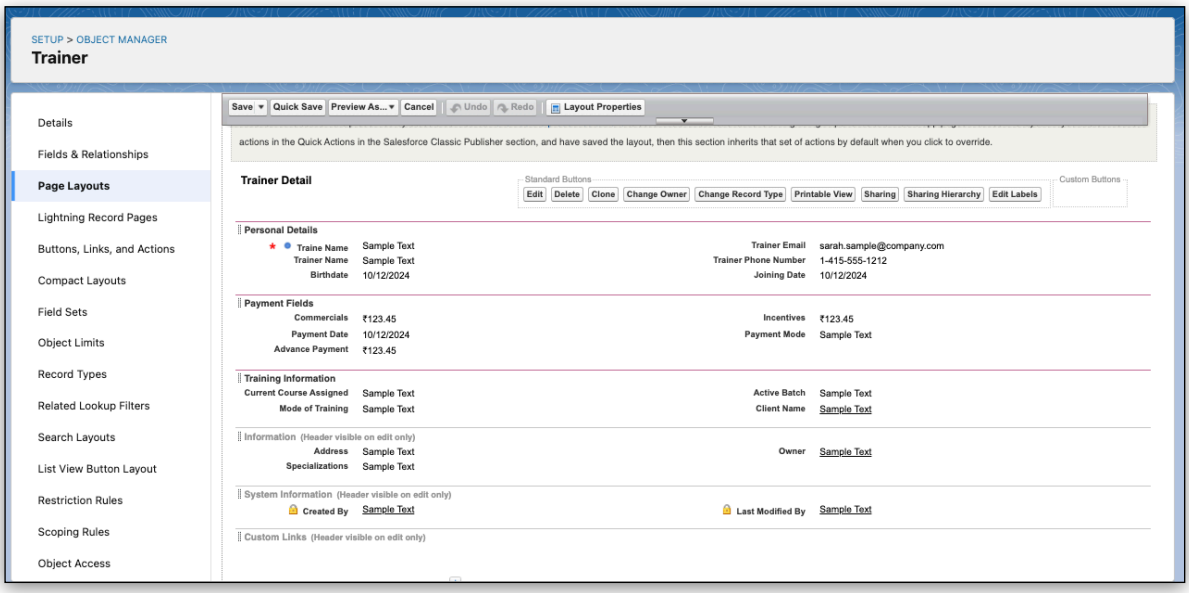


Schema Builder Showing custom objects and fields

2.3 Page Layouts



Employee Object page layout



Trainer Object page layout

AUTOMATION

Email Templates and Alerts

5.1 Scenario 1:

- Upon Web-to-Lead form submission or Lead creation, notify the sales team and candidate.

Solution Overview:

1. **Create Email Templates** for the sales representative and the new lead.
2. Use Email Templates to create corresponding **email alerts**.
3. Send Email Alerts using **flow automation**.

IMPLEMENTATION STEPS :

STEP-1: Create Email Templates for Sales Rep and New Lead:

- **Email Template for Sales Rep**
  - Go to Setup > Quick Find > Classic Email Templates > New Template.
  - **Enter Email Template Details:**
    - **Type:** Text
    - **Template Name:** Email Template for Sales Rep
    - **Available for Use:** True
    - **Subject:** New Lead Alert!
    - **Body:**

```
Hello Sales Team,

A new lead has been submitted via the web form. Below are the details:
- First Name: {!Lead.FirstName}
- Last Name: {!Lead.LastName}
- Instrested in Course: {!Lead.Course_Name__c}
- Registered On: {!Lead.Registration_Date__c}
- Email: {!Lead.Email}
```

- Phone: {!Lead.Phone}

Please review the details and reach out to the candidate at your earliest convenience.

Best regards,
Salesforce System

- **Test the Template:** Test the merge field by clicking **Send Test and Verify Merge Field**, then add test record and your email address.

- **Email Template for New Lead**
  - Go to Setup > Quick Find > Classic Email Templates > New Template.
  - Enter Email Template Details:
    - **Type:** Text
    - **Template Name:** Email Template for New Lead
    - **Available for Use:** True
    - **Subject:** Thank You for Enrolling into the Course
    - **Body**:

Dear {!Lead.LastName},

Thank you for your interest in {!Lead.Course_Name__c} Course. We've successfully received your details, and a member of our sales team will be reaching out to you shortly.

In the meantime, if you have any questions, please don't hesitate to get in touch with us:

- Email: info@ilearningecareer.com
- Phone: +91 9730002508

We're excited about the opportunity to help you advance your career and achieve your goals.

Best regards,
The ILearnings Team

- **Note:** Use merge fields like {!___} to insert lead data.

- Test the merge field by clicking **Send Test and Verify Merge Field**, then add test record and your email address.

STEP-2: Create Email Alerts Using Email Templates:

- **Email Alert for Sales Rep**
  - Go to Setup > Quick Find > Email Alerts > New Email Alert.
  - Enter Email Alert Details:
    - **Description:** Email Alert for Sales Rep
    - **Object:** Lead
    - **Template:** Select "Email Template for Sales Rep”
    - **Recipient:** Receiver of this email alert is sales rep which is lead.Owner, So, Select **Record Owner** as the recipient or reciever.

- **Email Alert for New Lead**
  - Go to Setup > Quick Find > Email Alerts > New Email Alert.
  - Enter Email Alert Details:
    - **Description:** Email Alert for New Lead
    - **Object:** Lead
    - **Template:** Select "Email Template for New Lead”
    - **Recipient:** Receiver of this email alert is lead, So, Select **Lead.Email** as the recipient or reciever address.

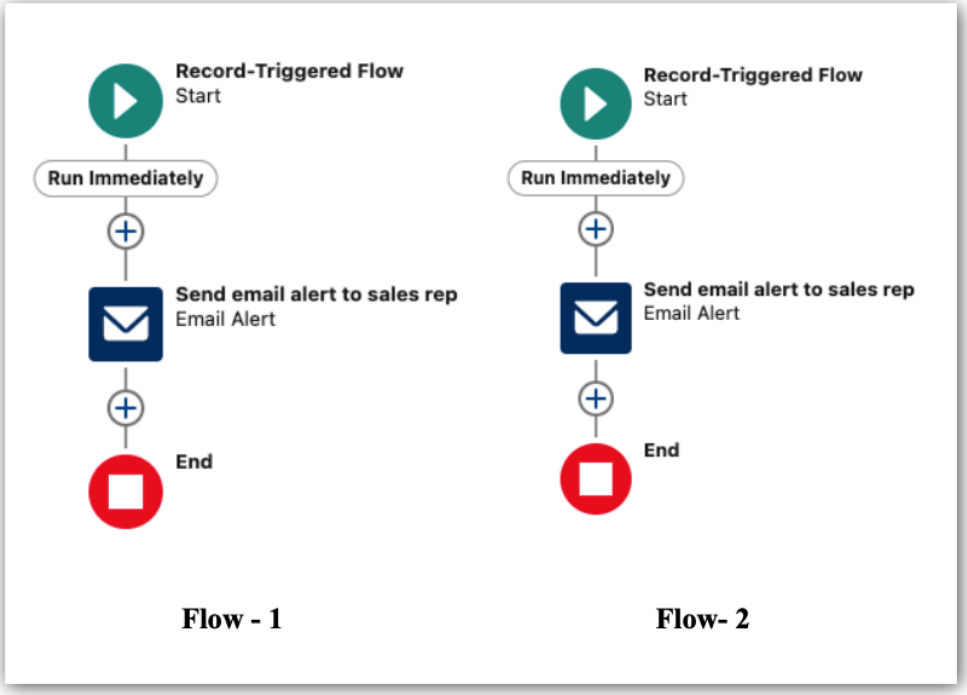
STEP 3: Create Flow to send Email alert

- **Sending email alert to sales rep**
  - Navigate to Flow Builder:
    - Go to Setup → Flow → New Flow.
    - Select **Record-Triggered Flow**.
  - Configure the flow to be triggered when a new Lead Record is created.
    - **Object**: Lead

- **Trigger Condition**: When a lead record is created.

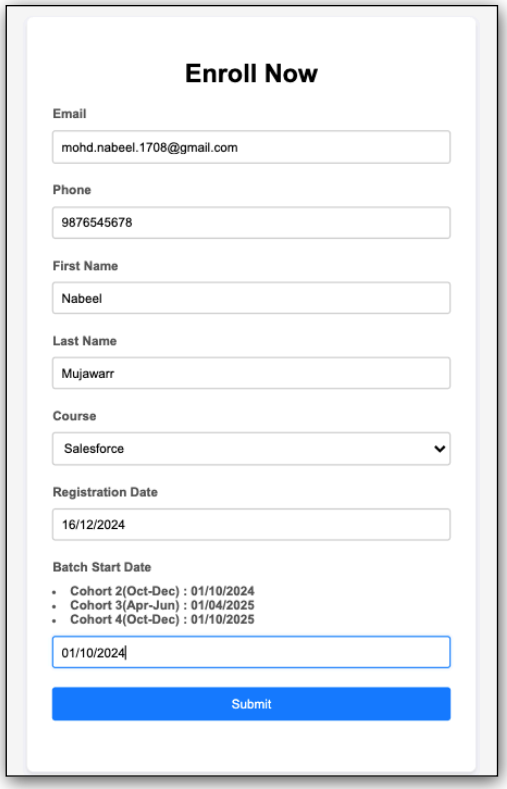
- **Set Up Entry Conditions:** Configure the flow to send an email alert only when a course is selected in the "Web to Lead" form.
  - **Condition**: Course_Name__c IS NOT NULL
- **Add an Email Alert**
  - Select '**Send Email Alerts**' Action Element.
  - Choose created email alert i.e 'Email alert for sales rep'
    - **Label**: Send Email Alert to Sales Rep
    - **Record ID**: Use the Lead Record to provide values for the email merge fields. Set **Record ID** to: Lead.Id.
- Save and Activate the Flow.

- **Sending Email alert to new lead**
  - Create Another Record-Triggered Flow:
    - Navigate to Setup → Flow → New Flow.
    - Choose Record-Triggered Flow.
    - Use the same entry condition as before: Course_Name__c IS NOT NULL.
  - **Add an Email Alert**
    - Select '**Send Email Alerts**' Action Element.
    - Choose created email alert i.e 'Email alert for new lead'
      - **Label**: Send Email Alert to New Lead
      - **Input Record ID**: Use the Lead Record to provide values for the email merge fields. Set **Record ID** to: Lead.Id.
  - Save and Activate the Flow.

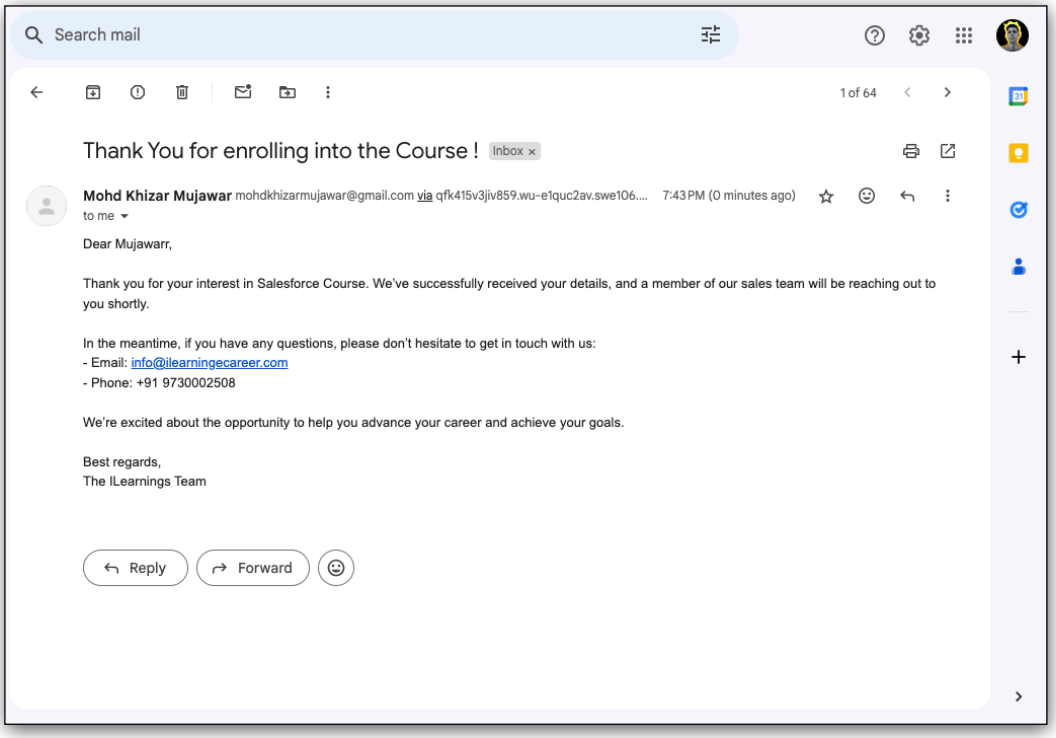


STEP 4: Testing Functionality

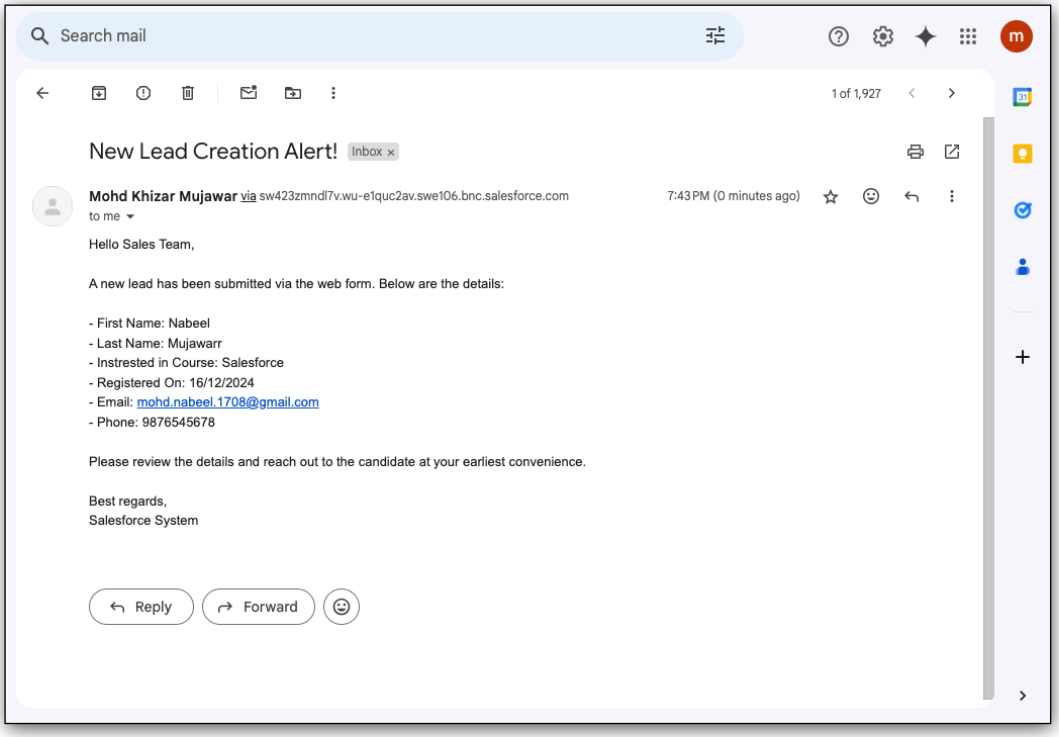
1. When a lead enrolls in the course and submits their details through the web-to-lead enrollment form.



2. Lead receives a thank-you email at the email address they entered.



3. ILearnings sales rep also receives an email to follow up with the lead.



6.2 Scenario 2:

- Send an email with the training schedule attached 1 day before the batch start date.

Solution:

1. **Create Email Template** for the new lead.
2. Use Email Templates to create **email alerts**.
3. Send Email Alerts using **flow automation**.

IMPLEMENTATION STEPS :

STEP-1: Create an Email Template with Attachment for New Lead

1. Go to **Setup > Quick Find > Classic Email Templates > New Template**.
2. Enter Email Template Details:
   - **Type:** Text
   - **Template Name:** Training Schedule Email Template
   - **Available for Use:** True
   - **Subject:** Training Schedule for Your Upcoming Batch
   - **Body:**

> Hello {!Lead.FirstName},
>
> Greetings from ILearnings!!
>
> We're excited to welcome you to the upcoming training batch for the {! Lead.Course_Name__c} course. Please find the training schedule attached.
>
> If you have any questions before the session, feel free to contact us at info@ilearningecareer.com or +919730002508.
>
> Thank You
> The ILearnings Team

3. Test the Template: Test the merge field by clicking **Send Test and Verify Merge Field**, then add test record and your email address.

4. **Add Attachment:**
   - Save the email template.

- Attach the **Training Schedule PDF** to this email template by uploading it in the attachments section which is below email template section.

STEP 2: Create an Email Alert

1. Go to **Setup > Quick Find > Email Alerts > New Email Alert**.
2. Enter Email Alert Details:
   - **Description:** Email Alert for Training Schedule
   - **Object:** Lead
   - **Email Template:** Select '*Training Schedule Email Template'*
   - **Recipient Type:** Lead must receive email, So select 'Email field' of lead object.
     - **Recipient type:** Email field ,& **Recipient**: Email

STEP 3: Use Scheduled Flows to Send Emails

You'll need to set up a **Scheduled Path** in a Record-Triggered Flow to send the email **1 day before the batch start date**.

3.1. Create the Flow

1. Go to **Setup > Quick Find > Flow > New Flow**.
2. Select **Flow Type: Record-Triggered Flow**
3. **Trigger Settings**:
   - **Object:** Opportunity (or the relevant object)
   - **Trigger Condition:** When a record is created or updated.
   - **Entry Conditions:**
     - Batch_Start_Date__c IS NOT NULL

3.2. Add a Scheduled Path

- In the Flow Builder, click the **Start** element ( green circle with the play icon).
- Select + Add Scheduled Paths.
- **Define Scheduled Path Settings:**
  - **Path Name:** Enter 'Send Training Schedule Email'.
  - **Condition:** Set to *1 day before Batch_Start_Date__c*.
    - **Time Source:** Lead: Batch Start Date
    - **Time Offset:** 1 day
    - **Offset Option:** Days Before
    - **Time Selection:** Choose 8:00 AM (or your preferred time).

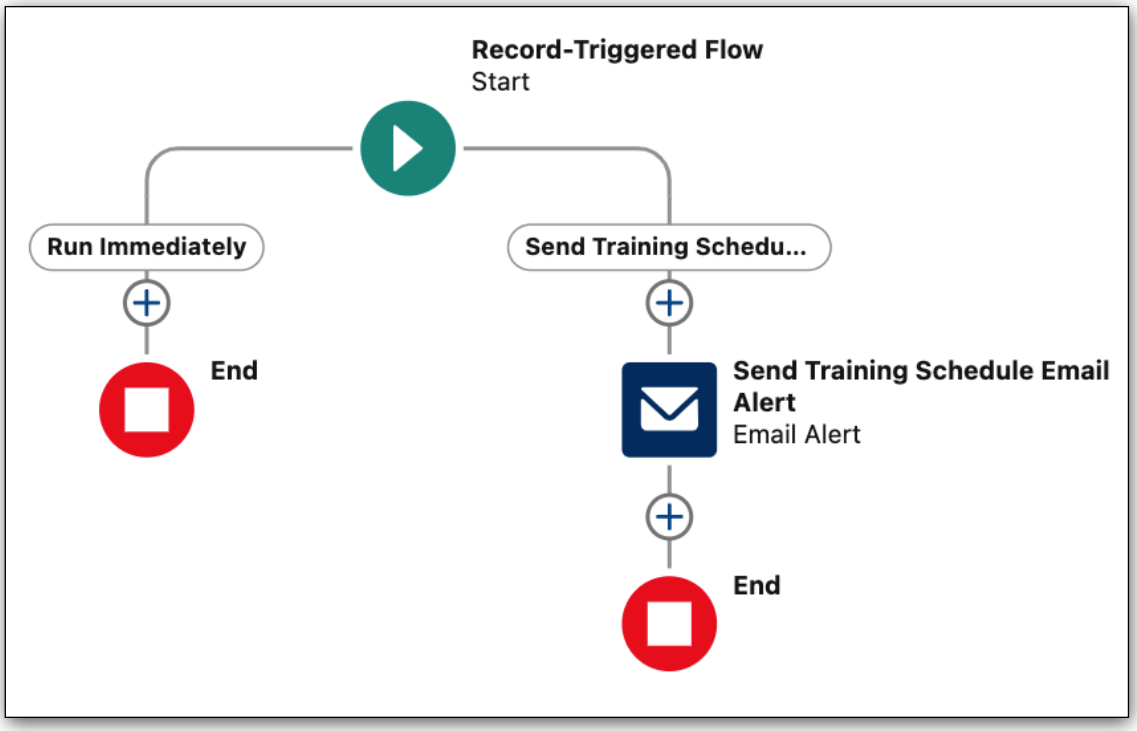
3.3. Add an Email Alert

- **Action Type:** Send Email Alerts
- **Label:** Send Training Schedule Email Alert
- **Email Alert:** Select '*Email Alert for Training Schedule'*.
- **Record ID:** Use the *'Lead Record ID'*.

3.4 Save and Activate the Flow

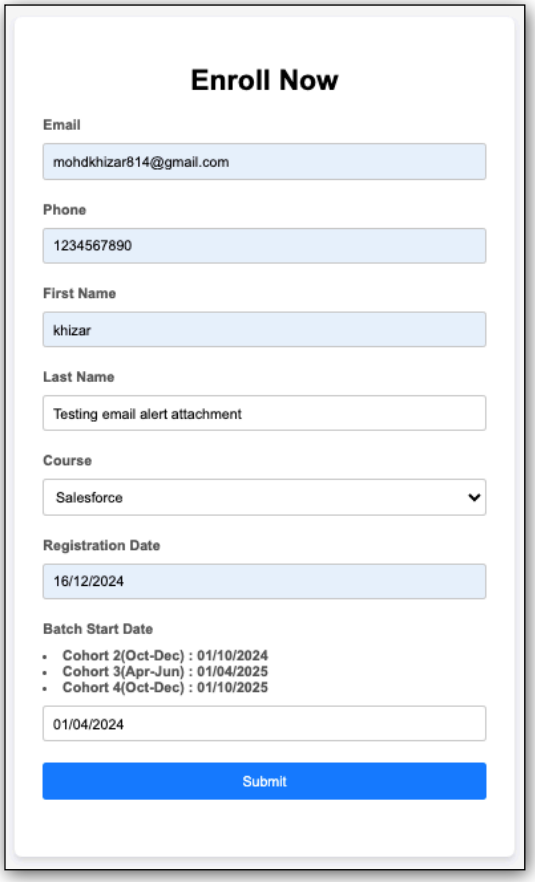
- **Flow Label:** Send Training Schedule Email to Leads

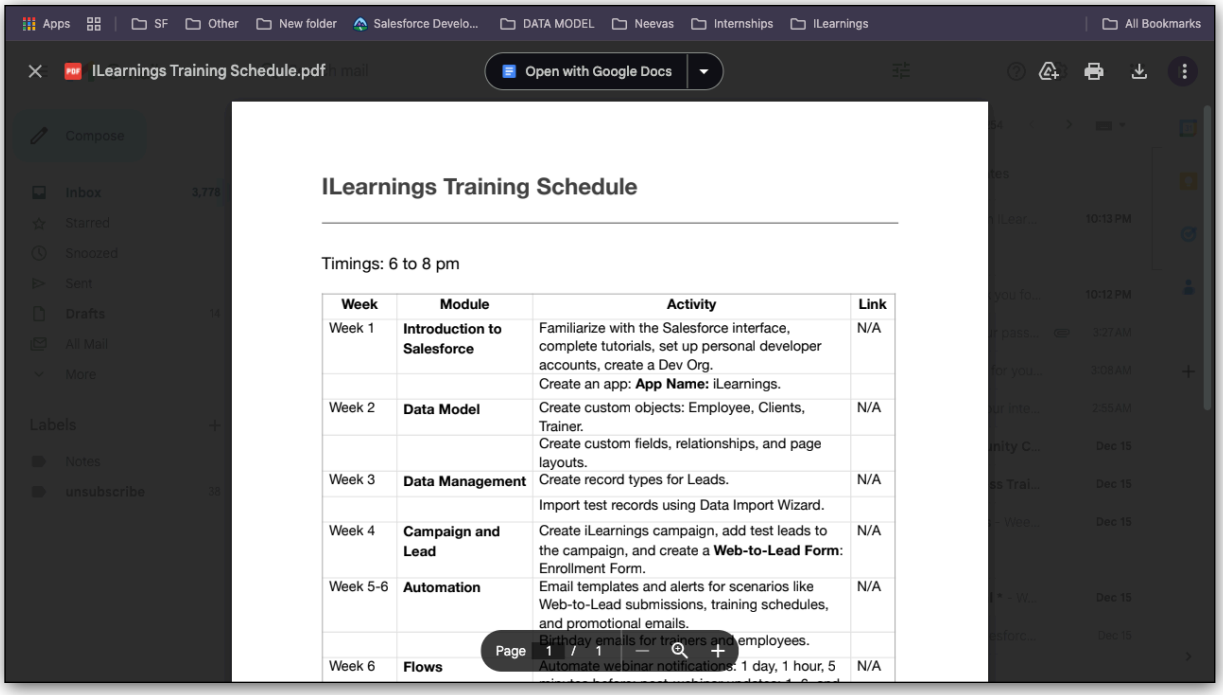
Flow



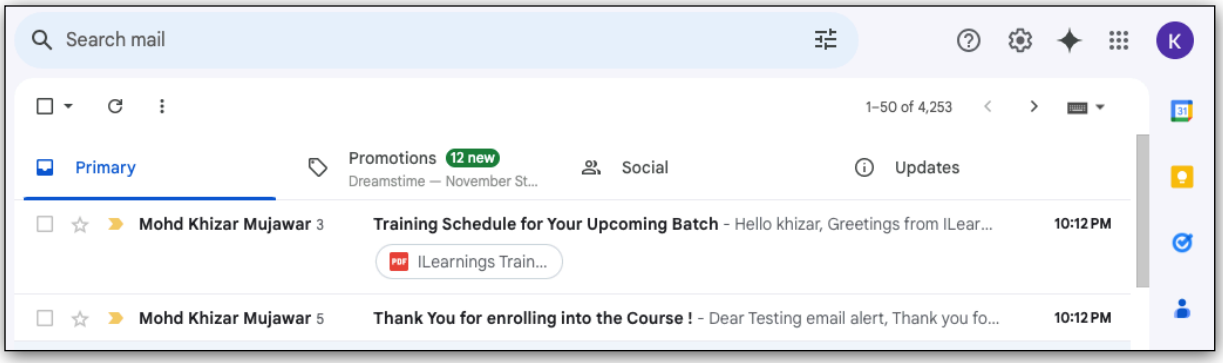
<u>STEP 4</u>: Testing Functionality

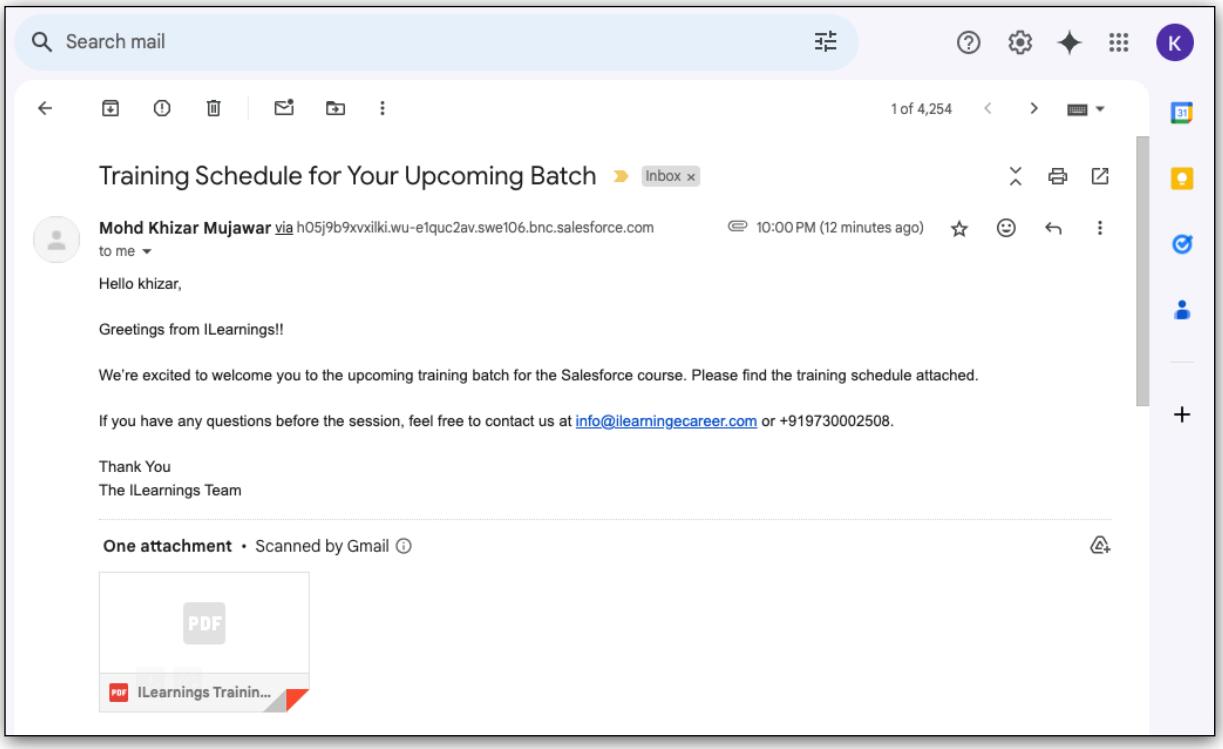
1. **Clone the Created Flow**.
2. **Modify the Trigger**:
   - Remove the **Scheduled Path**.
   - Set the flow to trigger **immediately** when a record is created or updated.
3. **Save and Activate**:
   - Save and activate the cloned flow. Deactivate the original flow temporarily.
4. **Create a Test Record**: Add a Lead record with Batch_Start_Date__c, Email, and other required fields.


Enroll Now
Email
mohdkhizar814@gmail.com
Phone
1234567890
First Name
khizar
Last Name
Testing email alert attachment
Course
Salesforce
Registration Date
16/12/2024
Batch Start Date
Cohort 2(Oct-Dec) : 01/10/2024
Cohort 3(Apr-Jun) : 01/04/2025
Cohort 4(Oct-Dec) : 01/10/2025
01/04/2024
Submit


ILearnings Training Schedule.pdf
Open with Google Docs
ILearnings Training Schedule
Timings: 6 to 8 pm
Week
Module
Activity
Link
Week 1
Introduction to Salesforce
Familiarize with the Salesforce interface, complete tutorials, set up personal developer accounts, create a Dev Org.
N/A
Create an app: App Name: iLearnings.
Week 2
Data Model
Create custom objects: Employee, Clients, Trainer.
N/A
Create custom fields, relationships, and page layouts.
Week 3
Data Management
Create record types for Leads.
N/A
Import test records using Data Import Wizard.
Week 4
Campaign and Lead
Create iLearnings campaign, add test leads to the campaign, and create a Web-to-Lead Form: Enrollment Form.
N/A
Week 5-6
Automation
Email templates and alerts for scenarios like Web-to-Lead submissions, training schedules, and promotional emails.
N/A
Birthday emails for trainers and employees.
Week 6
Flows
Automate webinar notifications: 1 day, 1 hour, 5
N/A
Page 1 / 1

5. **Check Results**:
   - Verify the email is received in your test inbox.
   - Check the email content and attachment.





Email with Training Schedule Attachment:

6. **Revert Changes**:
   - Deactivate the cloned flow and reactivate the original flow.

5.3 Scenario 3:

- Send promotional emails related to offers.
- When a Lead is created, send a "Buy 1, Get 1" offer email.

Solution:

1. **Create Email Template** for the new lead.
2. Use Email Templates to create **email alerts**.
3. Send Email Alerts using **flow automation**.

IMPLEMENTATION STEPS :

STEP-1: Create an Email Template with Attachment for New Lead

1. Go to **Setup > Quick Find > Classic Email Templates > New Template**.
2. Enter Email Template Details:
   - **Type:** Text
   - **Template Name:** BOGO Offer Email Template
   - **Available for Use:** True
   - **Subject:** Exclusive Offer Just for You: Buy 1, Get 1 Free!
   - **Body:**

Hello {!Lead.FirstName},

Thank you for showing interest in our services! We're excited to present an exclusive 'Buy 1, Get 1 Free' offer, just for you.

Here's how you can claim this offer:
- Choose any course of your interest.
- Enroll today to avail a free course of equal or lesser value!

Don't wait too long, as this offer is valid for a limited time only.

To get started, contact us today:
- Email: offers@ilearningecareer.com
- Phone: +91 9730002508

Best regards,
The ILearnings Team

- **Note:** Use merge fields like {!___} to insert lead data.

3. Test the merge field by clicking **Send Test and Verify Merge Field**, then add test record and your email address.

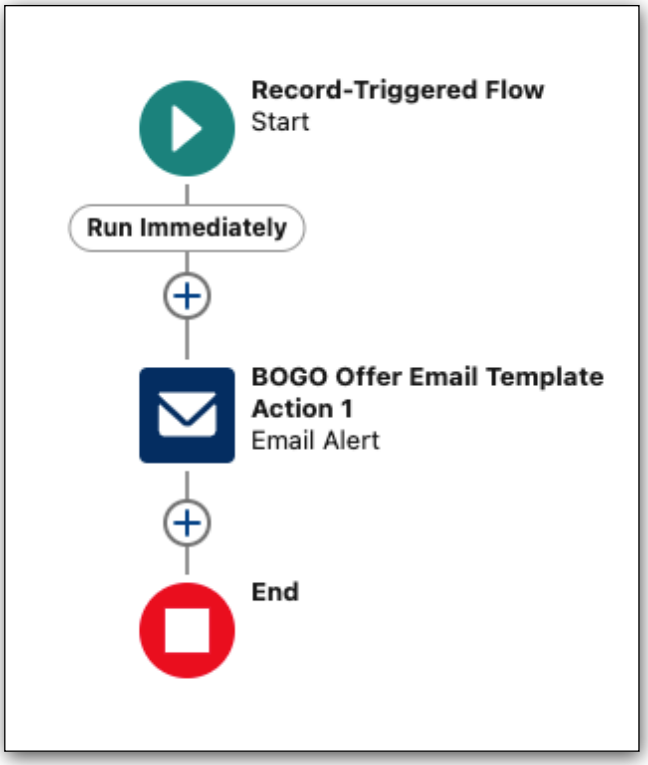
STEP 2: Create an Email Alert

1. Go to **Setup > Quick Find > Email Alerts > New Email Alert**.
2. Enter Email Alert Details**:**
   - **Description:** Email Alert for BOGO Offer
   - **Object:** Lead
   - **Email Template:** Select '*BOGO Offer Email Template*'.
   - **Recipient Type:** Lead must receive email, So select 'Email field' of lead object.
     - **Recipient type:** Email field ,& **Recipient**: Email

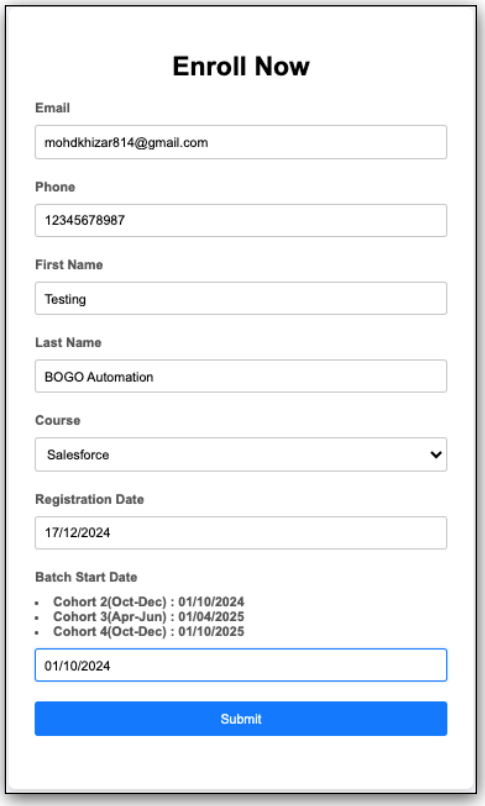
STEP 3: Create Flow to send Email alert

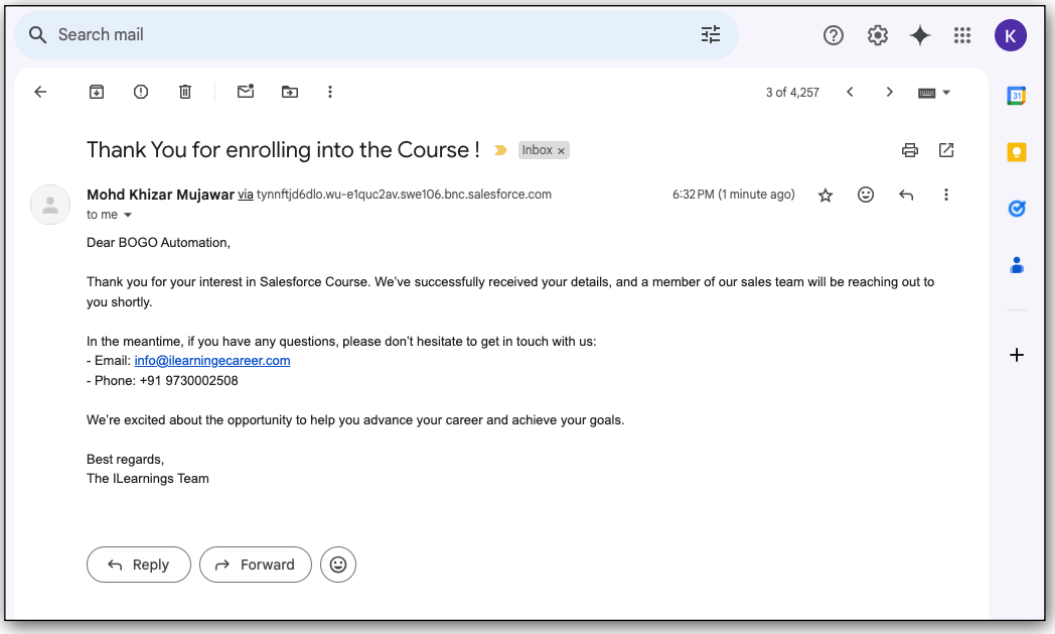
1. **Go to Setup → Flow → New Flow.**
2. **Choose Flow Type:** Select **Record-Triggered Flow**.
3. **Trigger Settings:**
   - **Object:** Lead
   - **Trigger:** When a record is created.
   - **Condition:** No specific condition required unless you want to restrict it to specific leads.
4. **Add an Email Alert**
   - **Action Type:** Send Email Alerts
   - **Email Alert:** Select the '*Email Alert for BOGO Offer'*.
   - **Record ID:** Set the Record ID to Lead.Id.
5. Save and Activate the Flow.

Flow



STEP 4: Testing Functionality




Search mail
3 of 4,257
Thank You for enrolling into the Course !
Inbox ×
Mohd Khizar Mujawar via tynnftjd6dlo.wu-e1quc2av.swe106.bnc.salesforce.com
6:32 PM (1 minute ago)
to me
Dear BOGO Automation,
Thank you for your interest in Salesforce Course. We've successfully received your details, and a member of our sales team will be reaching out to you shortly.
In the meantime, if you have any questions, please don't hesitate to get in touch with us:
- Email: info@ilearningecareer.com
- Phone: +91 9730002508
We're excited about the opportunity to help you advance your career and achieve your goals.
Best regards,
The ILearnings Team
Reply
Forward

5.4 Scenario 4:

- Send birthday emails to trainers and employees.

Solution:

1. Create an **'Is Birthday'** formula field on the **Employee** object.
2. Create a shared **Birthday Email Template** for both trainers and employees.
3. Use the email template to configure an **Email Alert**.
4. Send the Email Alerts using a **Scheduled-Triggered Flow**.
   - Use the formula field to get only records with birthday '**today**'.

IMPLEMENTATION STEPS:

Using Scheduled Flows to Send Email !

STEP 1: Create Formula Field

1. **Navigate to Setup**:
   - Go to **Setup → Object Manager** → Select **Trainer** or **Employee**.
   - For Trainers: Navigate to **Trainer > Fields & Relationships**.
   - For Employees: Navigate to **Employee > Fields & Relationships**.
2. **Create a New Formula Field**:
   - Click **New Field** → Select **Formula** as the field type.
   - Enter a Field Label: Is_Birthday__c.
   - Set the **Formula Return Type**: Checkbox.
3. **Define the Formula Logic**:
   - Use the following formula to check if today's date matches the Birthdate:

```
IF(
 AND(
  DAY(TODAY()) = DAY(Birthdate__c),  /* Checking day */
  MONTH(TODAY()) = MONTH(Birthdate__c)  /* Checking month */
 ),
 TRUE,
 FALSE
)
```

4. **Save the Field**:
   - Mark **Treat Blank Fields as Blank**: True.
   - Save the field.
5. **Repeat for Both Objects**:
   - Perform this step for both the **Trainer** and **Employee** objects.

STEP 2: Create an Email Template

1. **Navigate to Classic Email Templates**:
   - Go to **Setup** → Quick Find → **Classic Email Templates** → Click **New Template**.
2. **Enter Template Details**:
   - **Type**: Text
   - **Template Name**: Birthday Email Template
   - **Available for Use**: True
   - **Subject**: Happy Birthday, {!Contact.FirstName}!
3. **Body**:

> Dear {!Contact.FirstName},
>
> Happy Birthday!
>
> On your special day, we'd like to take a moment to celebrate you and thank you for all your hard work and dedication.
>
> Wishing you a wonderful year ahead filled with success, happiness, and joy. Have an amazing day!
>
> Best wishes,
> The ILearnings Team

4. **Test the Template**:
   - Use the **Send Test and Verify Merge Fields** feature with sample data to ensure proper functionality.

STEP 3: Create an Email Alert

1. **Navigate to Email Alerts**:
   - Go to **Setup** → Quick Find → **Email Alerts** → Click **New Email Alert**.
2. **Enter Alert Details**:
   - **Description**: Birthday Email Alert
   - **Object**: Choose Contact (or the relevant object for trainers and employees).
   - **Email Template**: Select the **Birthday Email Template** created in Step 2.

- **Recipient**: Set the recipient as Contact.Email (or the equivalent email field for trainers and employees).

STEP 4: Create a Flow to Send Email Alerts

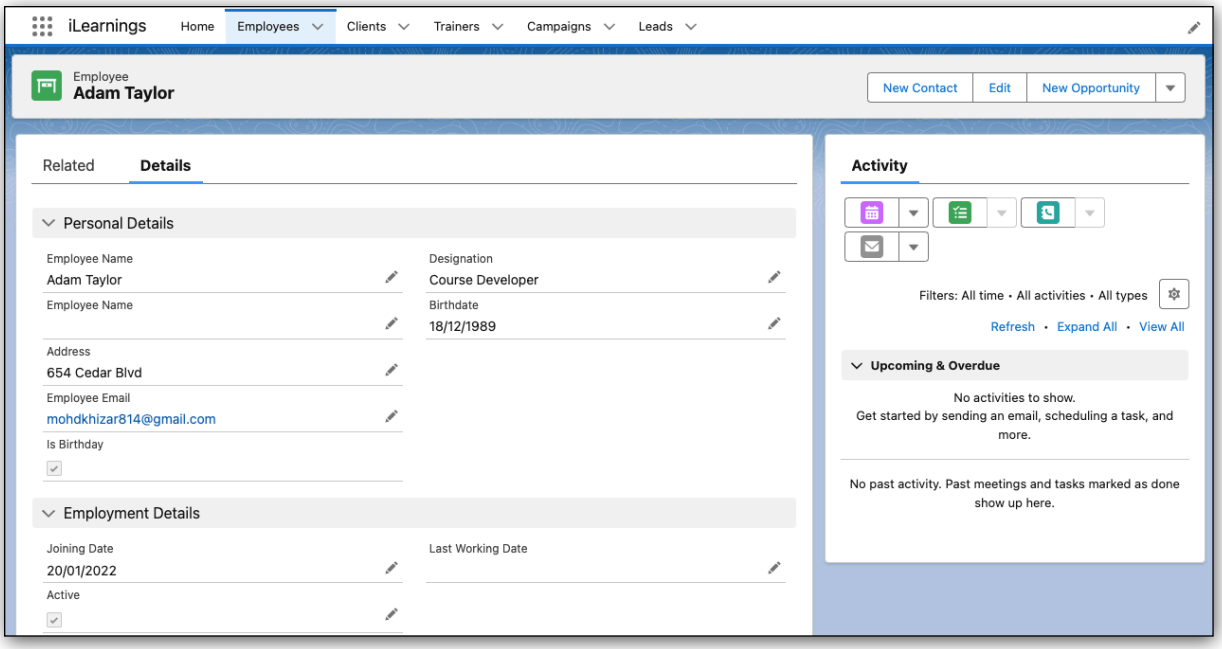
1. **Navigate to Flows**:
   - Go to **Setup** → Quick Find → **Flows** → Click **New Flow**.
2. **Choose Flow Type**:
   - Select **Scheduled-Triggered Flow**.
3. **Configure the Scheduled Flow**:
   - **Trigger Settings**:
     - Object: Choose Trainer (or Employee for the Employee flow).
     - Frequency: Select **Daily** (to check for birthdays every day).
     - Start Date: Choose today's date (or any preferred start date).
     - Start Time: Set a specific time to run the flow (e.g., 8:00 AM).
   - **Entry Conditions**:
     - Set conditions to filter records whose birthdays are today:
       - Field: Is_Birthday__c (Formula field from Step 1).
       - Operator: Equals.
       - Value: True.
   - **Configure the Flow to Send Emails:**
     - In your flow, add an **Email Alert Action**.
     - Click + Add Element → Select Action.
     - In the Search Actions box, type Alert → Select Email Alert.
     - Choose the Birthday Email Alert created in Step 3.
     - Set the **Record ID** as the Employee's ID.
   - **Add the "Get Records" Element:**
     - Fetch the email template by using the **Get Records** element and selecting the email template by its unique name (e.g., "Birthday_Email_Template").
   - **Set Up the "Create Records" Element:**
     - Add a **Create Records** element to create an email message.
     - Use the following fields to configure the email:
       - **From Address**: mohdkhizarmujawar@gmail.com (your verified organization's default address).
       - **To Address**: Employee's email (Triggering Employee__c > Employee Email).
       - **Subject**: "Cheers… It's Your Day!"

- **Text Body**: Insert a dynamic text template to personalize the message, e.g., "Dear {!Employee.Name}, Happy Birthday!"

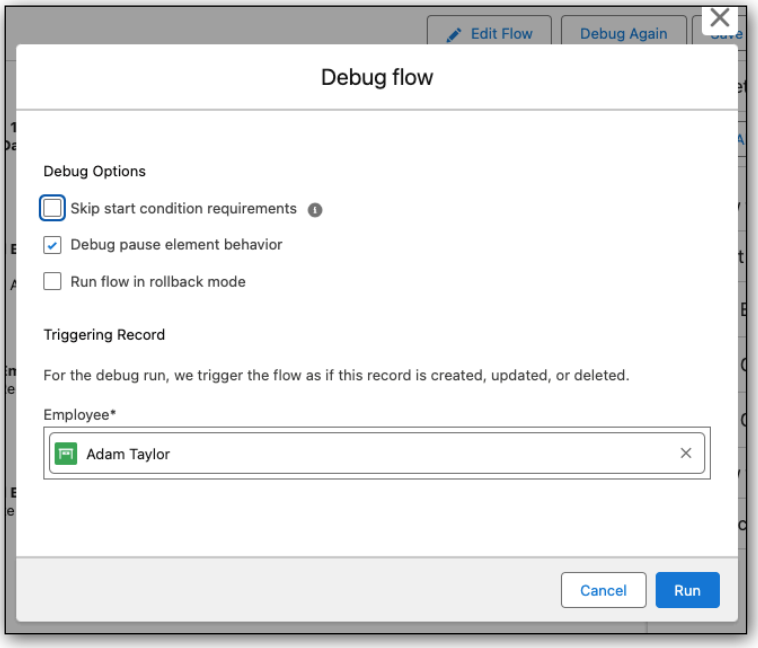
Repeat for Both Objects: Create separate flows for Trainer and Employee objects, each with their respective conditions and actions.

STEP 5: Test and Debug the Flow

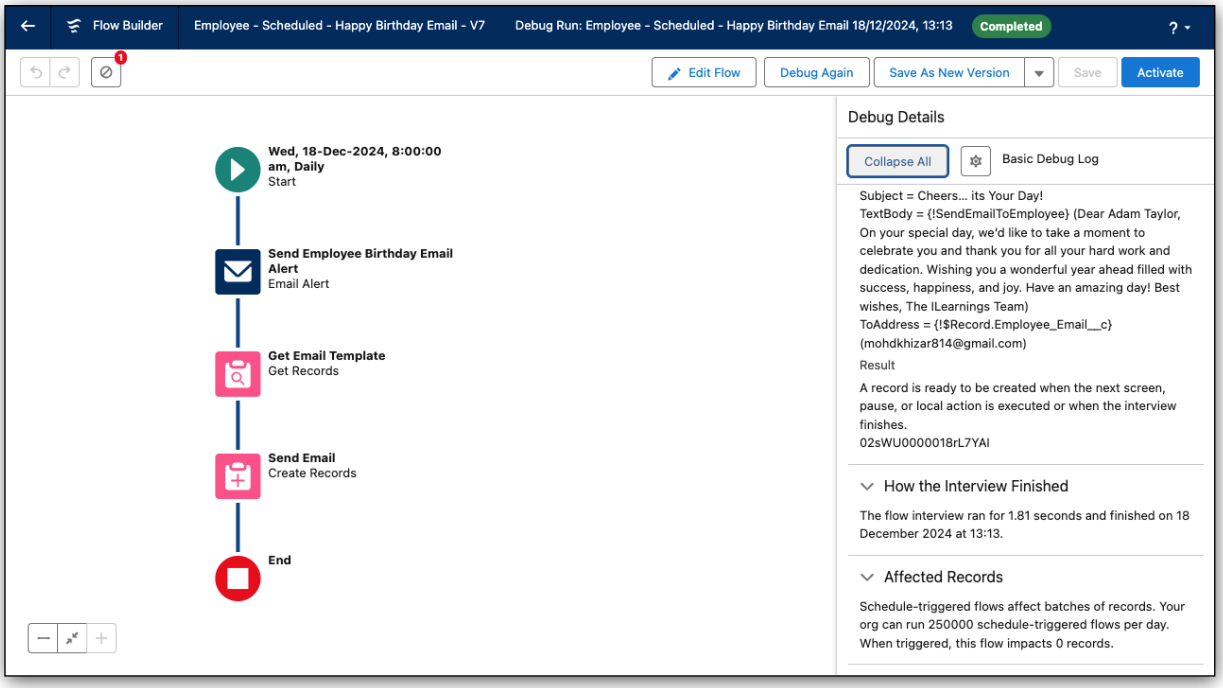
1. **Prepare a Test Record:** Set birthday of some record to current day.



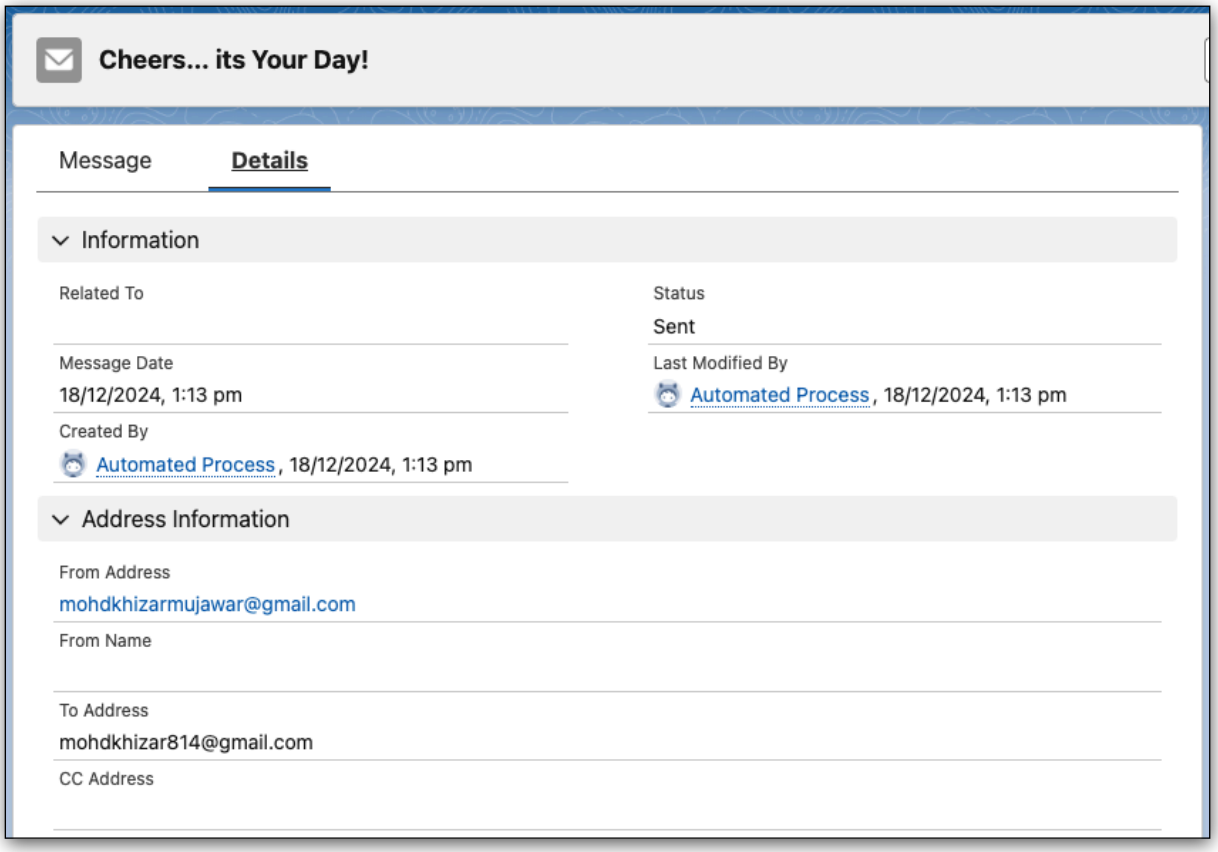
2. Debug Flow > Select the record where the birthday is set to today.

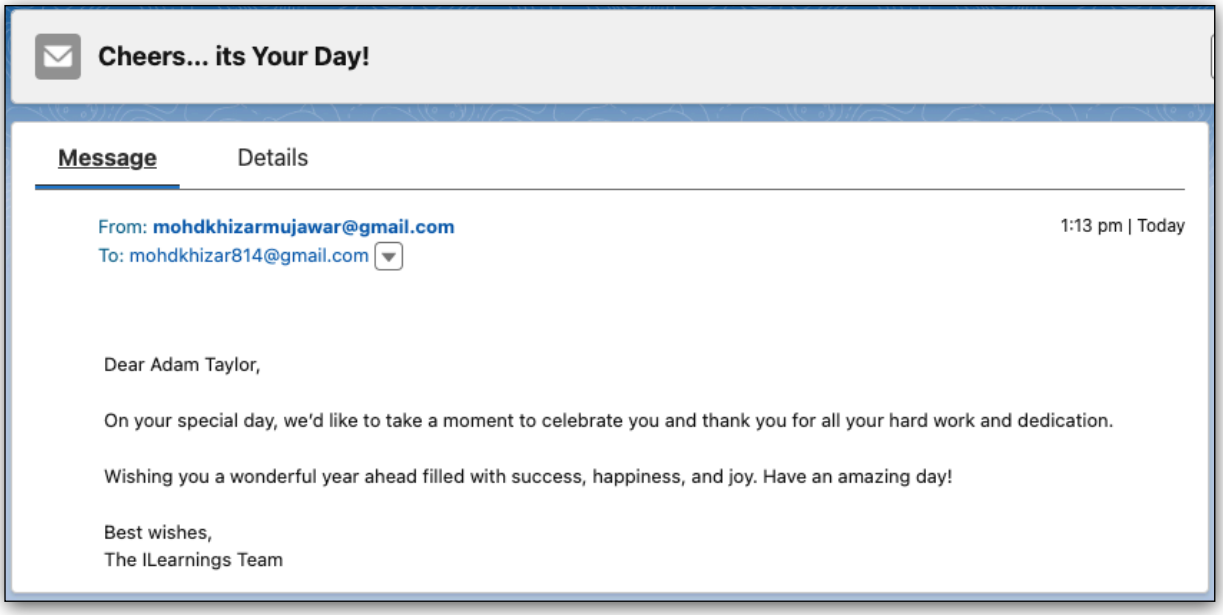


3. Flow executed successfully.



4. **Verify Salesforce Email Logs:** Copy record ID from flow and paste it in salesforce link




Cheers... its Your Day!
Message
Details
From: mohdkhizarmujawar@gmail.com
To: mohdkhizar814@gmail.com
1:13 pm | Today
Dear Adam Taylor,
On your special day, we'd like to take a moment to celebrate you and thank you for all your hard work and dedication.
Wishing you a wonderful year ahead filled with success, happiness, and joy. Have an amazing day!
Best wishes,
The ILearnings Team

AUTOMATION

Flows

Create Flows for Campaign Members:

6.1 **Scenario 1:**

- 24 hours before the webinar (based on Webinar Date on Campaign Member), send "1 Day Before Webinar" email template.

6.2 **Scenario 2:**

- 1 hour before the Webinar Date and Time, send "1 Hour Before" email template.

6.3 **Scenario 3:**

- 5 minutes before the webinar starts, send "We Are Live" email template.

Solution Overview:

Step-1: Initial Setup

- **1.1:** Create Email Templates and Alerts
- **1.2:** Create ‘Webinar Start Date and time’ custom field on campaign member.
- **1.3:** Populate ‘Webinar End Date and Time’ field of campaign member with the date present in ‘Webinar End Date and Time’ field of campaign using flow.
- **1.4: Helper Fields:** Create ‘Hours_Since_Webinar__c’ formula field on campaign member. Use created ‘Webinar Start Date and Time’ custom field in calculation of ‘Hours_Since_Webinar__c’ formula field

Step-2: Create Flow

- Create Scheduled triggered using Helper Fields to send email ‘after one hour’ or ‘after six hour’ or ‘after twenty four hour'

IMPLEMENTATION STEPS :

Step-1: Initial Setup

1.1: Create Email Templates And Alerts

Email Templates

Scenario - 1 Email Template:

Email Template Name: 24 hrs before webinar
Available for use: True

Subject: Reminder: Your Webinar is Tomorrow!

Dear {!CampaignMember.FirstName},

Be ready to join on webinar tomorrow.

Date and Time: {!CampaignMember.Webinar_Start_Date_and_Time__c}

Please ensure you have a stable internet connection and a quiet environment for the best experience.

If you have any questions before the webinar, feel free to contact us
at info@ilearningecareer.com or +919730002508.

We look forward to seeing you tomorrow!

Best regards,
The Learnings Team

Scenario - 2 Email Template:

Email Template Name: 1 hr before webinar
Available for use: True

Subject: Reminder: Your Webinar Starts in 1 Hour!

Dear {!CampaignMember.FirstName},

This is a quick reminder that your webinar starts in just 1 hour!

Date and Time: {!CampaignMember.Webinar_Start_Date_and_Time__c}

Please ensure you:

- Join 5-10 minutes early to avoid any delays
- Have a stable internet connection and a quiet environment

If you encounter any issues, feel free to contact us at info@ilearningecareer.com or +919730002508.

We're excited to see you there!

Best regards,
The Learnings Team

Scenario - 3 Email Template:

Email Template Name: 5 mins before webinar
Available for use: True

Subject: Reminder: Your Webinar Starts in 5 Minutes!

Dear {!CampaignMember.FirstName},

Your webinar is about to begin!

Date and Time: {!CampaignMember.Webinar_Start_Date_and_Time__c}

Please ensure you have a stable internet connection and a quiet environment for the best experience.

If you have any issues connecting, contact us at info@ilearningecareer.com or +919730002508.

We look forward to seeing you shortly!

Best regards,
The Learnings Team

Email Alerts

Scenario - 1 Email Alert:

1. Description: 1 Day before webinar email alert
2. Object: Campaign Member
3. Select '1 Day Before Webinar' email tempalate
4. Recipient type: Email field
5. Selected recipient: Email

Scenario - 2 Email Alert:

1. Description: 1hr before webinar email alert
2. Object: Campaign Member
3. Select '1 Hr Before Webinar' email tempalate
4. Recipient type: Email field
5. Selected recipient: Email

Scenario - 3 Email Alert:

1. Description: 5 mins before webinar email alert
2. Object: Campaign Member
3. Select '5 mins Before Webinar' email tempalate
4. Recipient type: Email field
5. Selected recipient: Email

1.2: Create ‘Webinar Start Date and time’ custom field on campaign member.

- Navigate to **Setup** > **Object Manager** > **Campaign Member**.
- Create a new custom field:
  - **Field Type**: Date
  - **Field Label**: Webinar Start Date and Time
  - **Field Name**: Webinar_Start_Date_and_Time__c
  - Add to page layouts if needed.
- Save the field.

Why create a custom field?

To create a ‘1 Day Before Webinar’ flow on Campaign Member, we need a custom Date/Time field, i.e., ‘Webinar Start Date and Time’ on the same object (Campaign Member).
However, the Campaign Member object does not have a ‘Webinar Start Date’ custom field. Therefore, a new custom Date/Time field, ‘Webinar Start Date and Time,’ must be created on the Campaign Member object.To create **‘1 day before webinar’** flow on Campaign Member, we need custom date field i.e ‘**Webinar Start Date**’ on same object i.e Campaign Member.

1.3: Create a Flow to populate created custom field of Campaign member using 'Start Date and Time’ custom Field of Campaign.

Steps to create Record-Triggered Flow:

1.3.1 Flow Type

- Go to **Setup** > **Flow** > **New Flow**.
- Select **Record-Triggered Flow**.
- Configure the flow to be triggered **on Campaign Member creation and updates**.

1.3.2 Add Trigger Criteria

- Set the triggering condition:
  - **Object**: Campaign Member
  - **Condition Requirements**: None (run every time a Campaign Member is created or updated).
  - **Triggering Actions**: Run flow after the record is saved.

1.3.3 Add Elements to the Flow

1. **Get Records (Fetch Campaign ‘Start Date and Time’)**

- Drag the **Get Records** element to the canvas.
- Configure it as follows:

- **Object**: Campaign
- **Filter Conditions D?**: This Condition ensures the flow fetches the Campaign record associated with the Campaign Member.
  - **Field**: Id (Campaign ID of the Campaign object)
  - **Operator**: Equals
  - **Value**: {!$Record.CampaignId} (Campaign Member's Campaign lookup field)
  - **EXPLANATION**:
    - **$Record**: Refers to the Campaign Member record that triggered the flow.
    - **CampaignId (CORRECT)**: The lookup field on Campaign Member that stores the ID of the related Campaign.
    - **Campaign.Id(WRONG)**: The ID of the Campaign object record fetched in the flow.
- Retrieve **Only the First Record**.

2. **Decision (Check if Campaign 'Start Date and Time' Exists)**

- Add a **Decision** element:
  - **Label:** "Start Date and Time Exists?"
  - Two outcomes:
    1. **Yes**: If the Webinar_Start_Date_and_Time__c field (on the related Campaign) is not null.
       - **Condition**:
         - Field: Campaign.Webinar_Start_Date_and_Time__c (Campaign. Webinar_Start_Date_and_Time__c)
         - Operator: **Is Null**
         - Value: **False**
    2. **No**: Default.
       - No specific condition

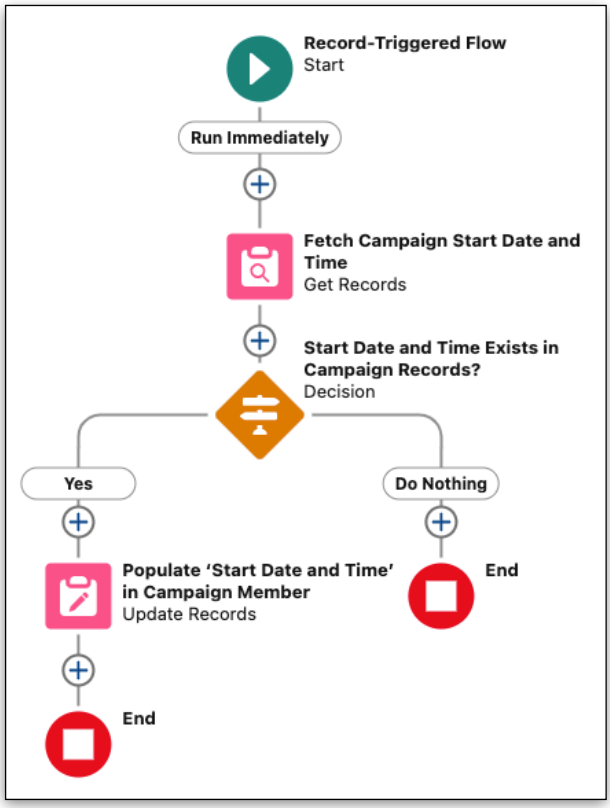
3. **Update Records (Populate 'Start Date and Time' in Campaign Member)**

- Under the **Yes** outcome, add an **Update (Triggered) Record** element.
- Configure it as follows:
  - **Record to Update**: Current Campaign Member.
  - **Field to Update**: Webinar_Start_Date_and_Time__c (of CampaignMember) = FetchedCampaign.Webinar_Start_Date_and_Time__c

4. **Default Path Handling**

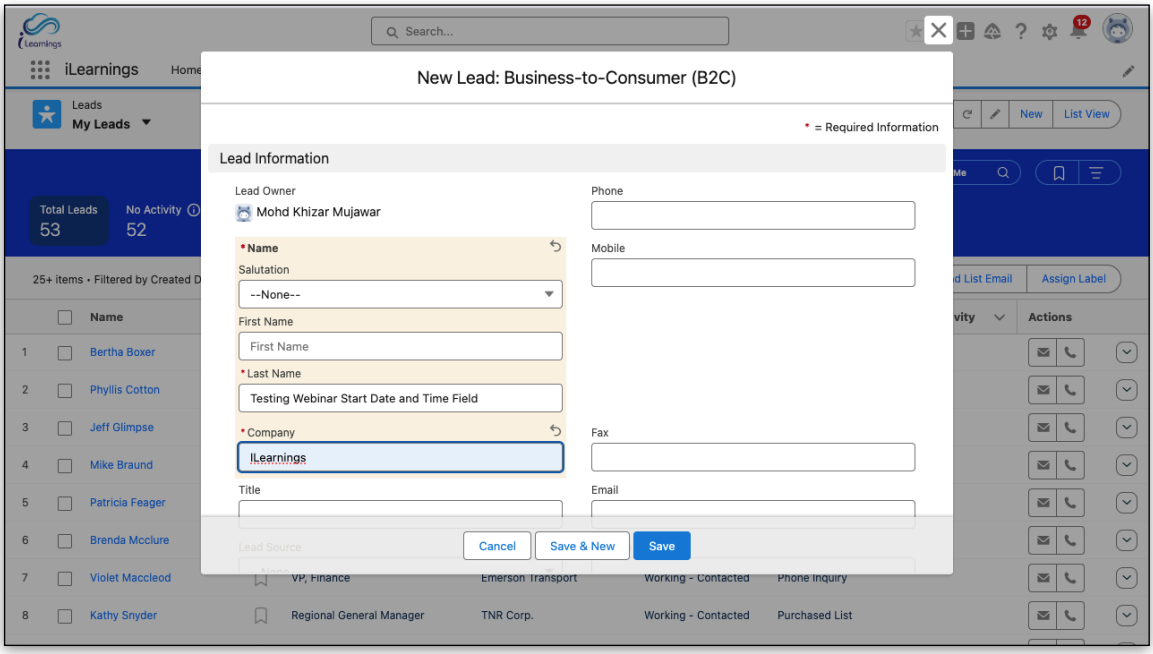
- If the '**Webinar Start Date and Time**' does not exist on the Campaign, you can log an error, (Do nothing) leave the 'Webinar Start Date and Time' field on the Campaign Member blank or skip the record.

Flow

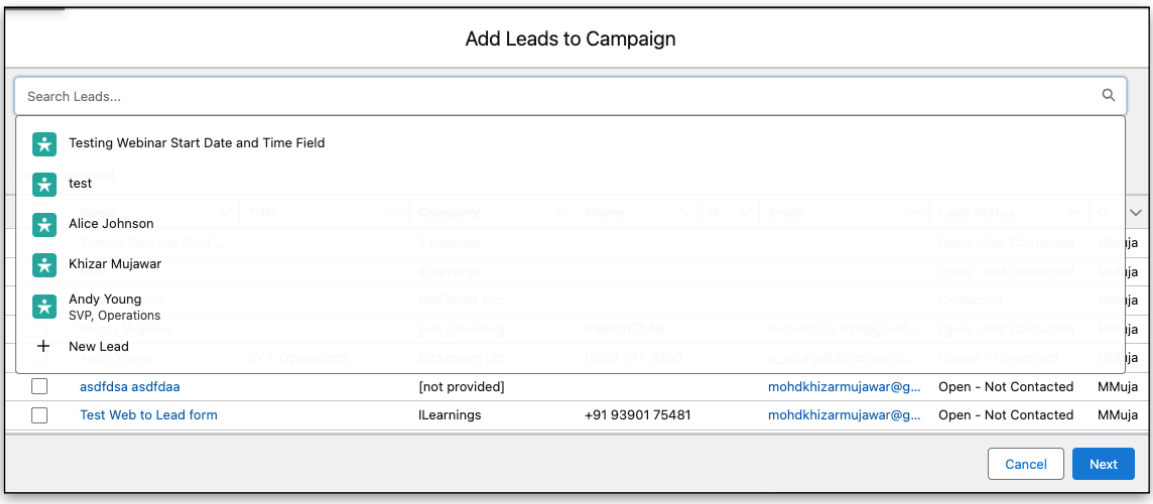


1.3.4 Testing if Webinar Start Date and Time in Campaign Member Object is Populated:

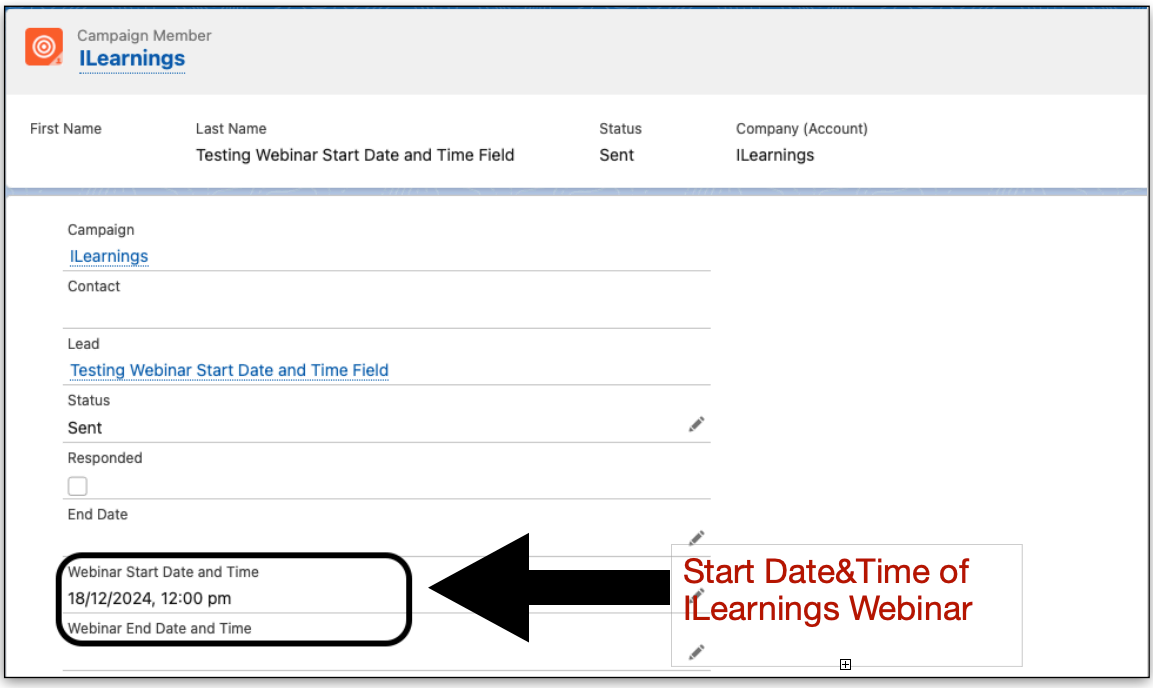
1. Create a new Lead.



2. Add the created Lead to a ILearningsCampaign.



3. The **Webinar Start Date and Time** field in the Campaign Member record is populated with the corresponding date from the **Webinar Start Date and Time** field on the related Campaign record.



1.4: Helper Fields: Create 'Hours_Since_Webinar__c' formula field on campaign member. Use created 'Webinar Start Date and Time' custom field in calculation of 'Hours_Since_Webinar__c' formula field

Fields:

1.4.1 **Days Until Webinar**: Store how many days are left until the webinar starts.

1.4.2 **Hours Until Webinar**: Store how many hours are left until the webinar starts.

1.4.3 **Minutes Until Webinar**: Store how many minutes are left until the webinar starts.

Use Case for These Fields:

These fields can be used to send email notifications at different times, such as:

- If (Days Until Webinar Field = 1) -> Send 1 day before webinar email
- If (Hours Until Webinar Field = 2) -> Send 2 hour before webinar email
- If (Minutes Until Webinar Field = 5) -> Send 5 min before webinar email

1.4.1 **Days Until Webinar**:

- **Name: Days Until Webinar**
- **Datatype:** Formula field
- **Output:** Number
- **Decimal:** 0
- **Formula:**

```
IF(
   ISBLANK(Webinar_Start_Date_and_Time__c),
   NULL,
   FLOOR(
      DATEVALUE(Webinar_Start_Date_and_Time__c) - TODAY()
   )
)
```

1.4.2 **Hours Until Webinar**:

- **Name: Hours Until Webinar**
- **Datatype:** Formula field
- **Output:** Number
- **Decimal:** 0
- **Formula:**

```
IF(
   ISBLANK(Webinar_Start_Date_and_Time__c),
   NULL,
   IF(
      Webinar_Start_Date_and_Time__c < NOW(),
      NULL,
      FLOOR(
         (Webinar_Start_Date_and_Time__c - NOW()) * 24
      )
   )
)
```

1.4.3 **Minutes Until Webinar**:

- **Name: Minutes Until Webinar**
- **Datatype:** Formula field
- **Output:** Number
- **Decimal:** 0
- **Formula:**

```
IF(
   ISBLANK(Webinar_Start_Date_and_Time__c),
   NULL,
   IF(
      Webinar_Start_Date_and_Time__c < NOW(),
      NULL,
      FLOOR(
         (Webinar_Start_Date_and_Time__c - NOW()) * 24 * 60
      )
   )
)
```

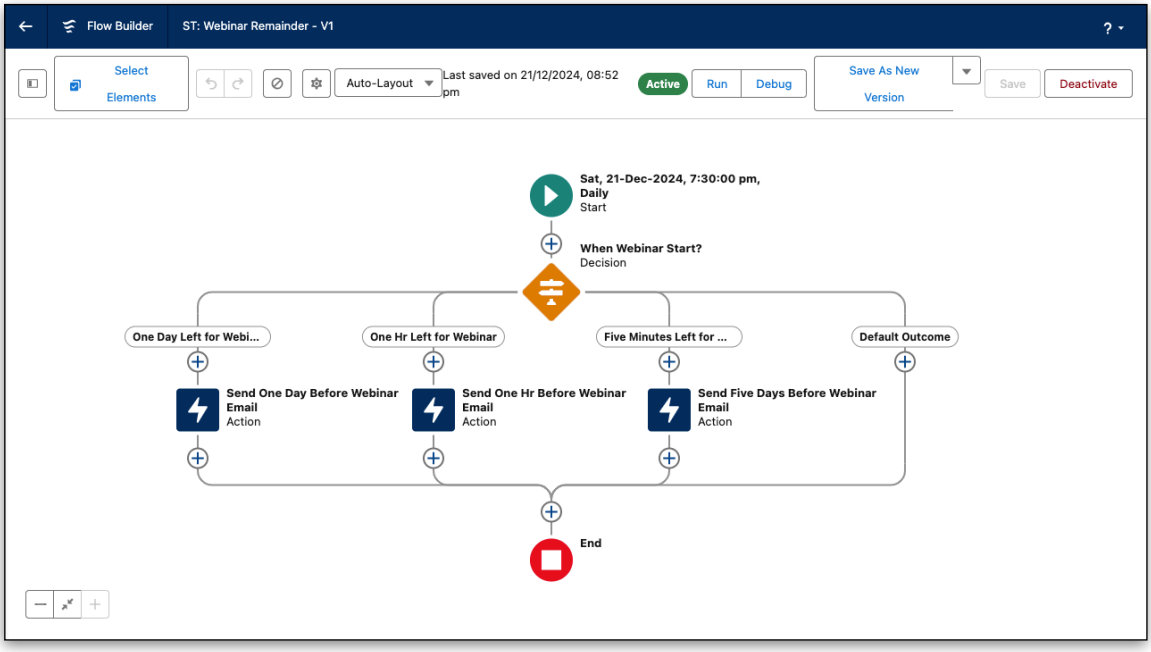
Step-2: Creating Flow To Send Webinar Email Before 1 Day, 2 Hr, 5 Mins:

- **Create a Flow:**
  - Go to Setup > Flows > Click New Flow > Select Scheduled-Triggered Flow and click Create.
  - Record Triggered flow send email at anytime, So we use Scheduled triggered flow for more control like sending email at specific time such as 10.00am.
- **Configure Trigger Settings**:
  - **Object**: Select **Campaign Member**.
  - **Trigger the Flow When**: Choose **A record is updated or created**.
- **Set A Schedule:**
  - **Start Date :** Enter Todays date, So it will start executing flow from today
  - **Start Time :** 7:30 pm (It start checking at 7:30pm)
  - **Frequency :** Daily
- **Click add object:**
  - Object: Campaign Member
  - **Entry Condition**:
    - **Field:** Webinar Start Date and Time (custom field on Campaign Member).
    - **Operator:** Is Null.
    - **Value:** False.

- **Add Decision Element**
  - **Label:** What time is it?
  - **Outcomes**:
    - **Label:** One Day Before Webinar:
      - **Add Condition:** Campaign Member(Triggered record) > Days_Until_Webinar__c is 1

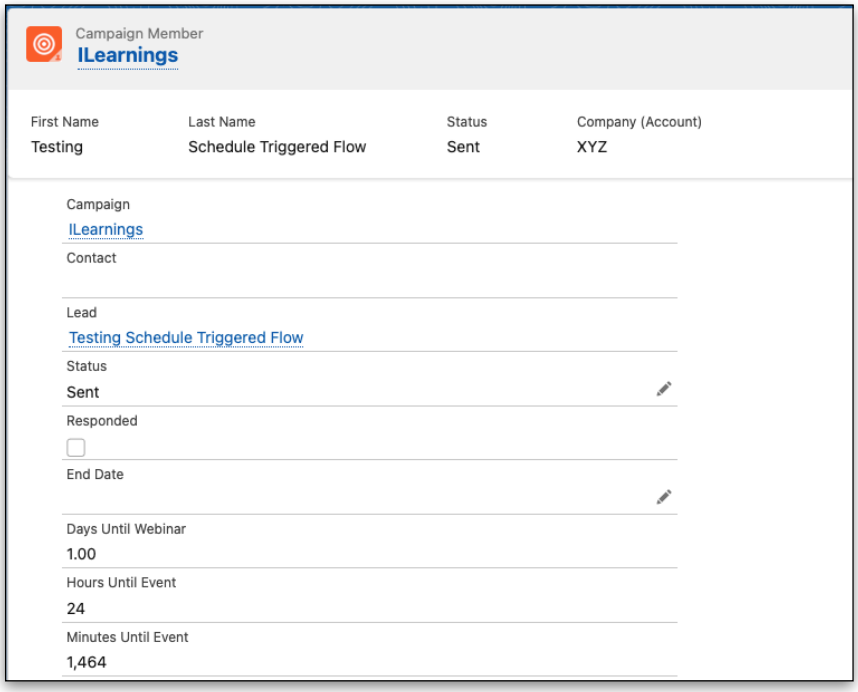
      Add one more outcome
    - **Label:** 2 Hours Before Webinar:
      - **Add Condition:** Campaign Member(Triggered record) > Hours_Until_Webinar__c is 2

      Add one more outcome
    - **Label:** 5 Minutes Before Webinar:
      - **Add Condition:** Campaign Member(Triggered record) > Minute_Until_Webinar__c is 5
- **Add Email Alerts**
  - Under One Day Before Webinar outcome, Add 'One Day Before Webinar Email Alert'
    - **Label:** Send Day Before Webinar Email, **RecordID:** CampaignMemberID {$Record.Id}
  - Under Two Hour Before Webinar outcome, Add '2 Hour Before Webinar Email Alert'
    - **Label:** Two Hour Before Webinar Email, **RecordID:** CampaignMemberID {$Record.Id}
  - Under Five Minutes Before Webinar outcome, Add 'Five min Before Webinar Email Alert'
    - **Label:** Five Minutes Before Webinar Email, **RecordID:** CampaignMemberID {$Record.Id}
- **Save and Activate the Flow**
  - Click **Save**, provide a name (e.g., **ST: Email Remainder About Webinar**) and Activate flow.

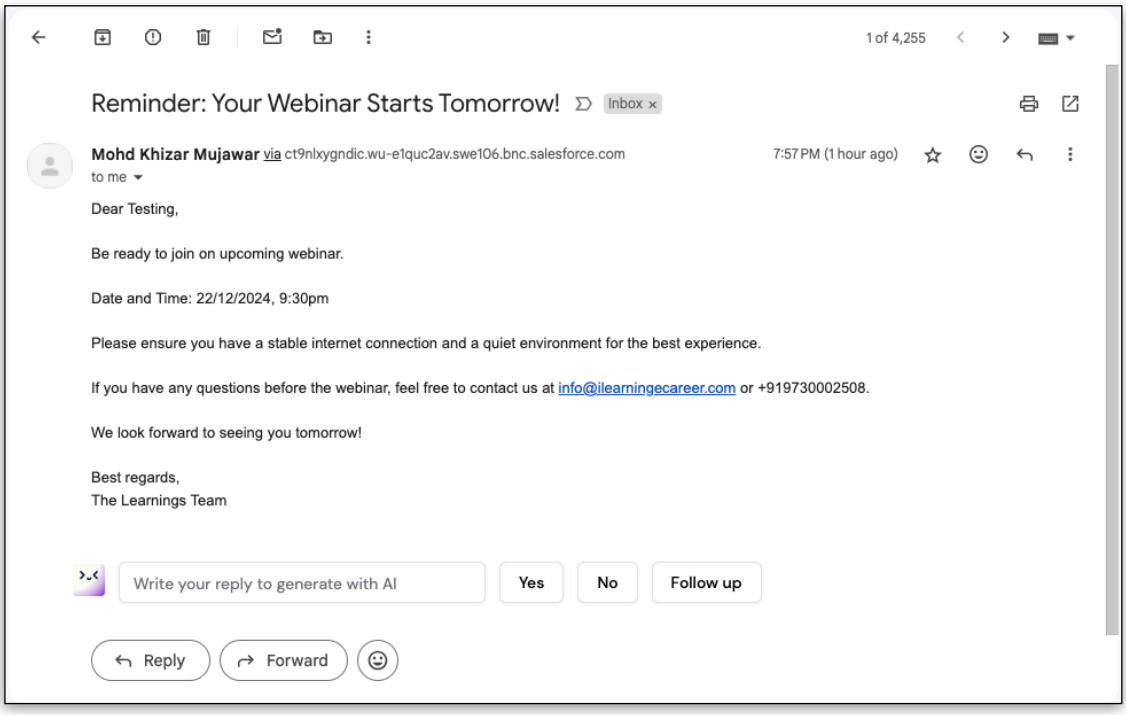


TESTING EMAIL REMAINDER FLOW

- Set time in scheduled triggered flow, keep 5 mins more in flow
  - **Ex:** if current time is 7:25 set to 7:30, Flow check at this time and send email
- Set ‘Webinar Start Date and Time’ field of ILearning campaign record to 1 day ahead
  - **Ex:** If todays date is 21 dec and current time is 7:25pm, Set ‘webinar start date and time’ to 22 dec and time to 7:30pm
- Create a New Lead, Make sure to enter email address.
- Add that new lead to ILearnings campaign.



- Email received at entered email address



AUTOMATION

Complex Flows

Create Flows for Campaign Members:

6.3 **Scenario 4:**

- 1 hour after the webinar on Campaign Members.

6.4 **Scenario 5:**

- 6 hours after the webinar On Campaign Members.

6.5 **Scenario 6:**

- 24 hours after the webinar On Campaign Members.

Solution Overview:

Step-1: Initial Setup

- **1.1:** Create Email Templates and Alerts
- **1.2:** Create ‘Webinar Start Date and time’ custom field on campaign member.
- **1.3:** Populate ‘Webinar End Date and Time’ field of campaign member with the date present in ‘Webinar End Date and Time’ field of campaign using Record Triggered Flow.
- **1.4: Helper Fields:** Create ‘Hours_Since_Webinar__c’ formula field on campaign member. Use created ‘Webinar Start Date and Time’ custom field in calculation of ‘Hours_Since_Webinar__c’ formula field

Step-2: Create Flow

- Create Scheduled triggered using Helper Fields to send email ‘after one hour’ or ‘after six hour’ or ‘after twenty four hour'

IMPLEMENTATION STEPS :

Step-1: Initial Setup

1.1: Create Email Templates And Alerts

Email Templates

1-Hour After Webinar:

Available for use: True
Subject: Thank You for Attending the Webinar!
Body:
Dear {!CampaignMember.FirstName},

Thank you for attending our webinar! We hope you found it insightful and valuable.

We would love to hear your feedback. Please take a moment to fill out our short feedback survey: [Insert Survey Link].

Stay tuned for more events and resources tailored to your interests!

Best regards,
ILearnings

6 Hour After Webinar:
Available for use: True
Subject: Webinar Recording and Resources Now Available!
Body:
Hi {!CampaignMember.FirstName},

Thank you again for joining us for the webinar. If you missed any part of the session or want to revisit the content, we've got you covered!

📹 Webinar Recording: [Insert Recording Link]

📄 Presentation Slides: [Insert Slide Link]

Feel free to reach out if you have any questions or need further assistance. We look forward to seeing you at our next event!

Warm regards,
ILearnings

24 Hours After Webinar:
Available for use: True
Subject: Stay Connected After the Webinar!
Body:
Hello {!CampaignMember.FirstName},

It's been a day since our webinar and we wanted to thank you for being a part of it. We hope you enjoyed the session and found the content useful!

If you haven't already, here are some ways to stay connected with us:
- Follow us on Social Media: Instagram Link, LinkedIn Link
- Explore More Resources: ILearnings Website Link

We look forward to continuing our journey together. If you have any questions or feedback, feel free to reach out.

Thank you,
ILearnings

Email Alerts

Create 1,6,24 Hour After Webinar Email Alerts

1.2: Create ‘Webinar Start Date and time’ custom field on campaign member or check if it Exists:

1. **On Campaign Member**:
   - **Field Name**: Webinar_End_Date_and_Time__c
   - **Type**: Date/Time
   - **Usage**: Stores the end date and time of the webinar for each Campaign Member.
2. **On Campaign**:
   - **Field Name**: Webinar_End_Date_and_Time__c
   - **Type**: Date/Time
   - **Usage**: Stores the end date and time of the webinar for the Campaign.

1.3: Create a Record-Triggered Flow

1.3.1 Flow Configuration

- **Go to Setup** > **Flow** > **New Flow**.
- **Select**: **Record-Triggered Flow**.
- **Configure the Trigger**:
  - **Object**: Campaign Member.
  - **Trigger Condition**: When a record is created or updated.
  - **Run the Flow**: After the record is saved.

1.3.2 Add Flow Elements

- **Get Records**: Fetch the related Campaign record.
  - **Element Label**: Get Campaign Webinar End Date and Time.
  - **Object**: Campaign.
  - **Filter Condition**:
    - Campaign.Id = {!$Record.CampaignId} (Fetch the Campaign associated with the Campaign Member).
  - Tick the ‘Retrieve Only the First Record’ Checkbox.
- **Decision**: Check if Webinar End Date and Time exists on the Campaign.
  - **Element Label**: Webinar End Date Exists?
  - **Conditions**:
    - Path 1: **Yes**
      - Campaign.Webinar_End_Date_and_Time__c IS NOT NULL.
    - Path 2: **No**

- Default path.

- **Update Records**: Populate Webinar End Date and Time on the Campaign Member.
  - **Element Label**: Update Webinar End Date on Campaign Member.
  - **Record to Update**: Current Campaign Member {!$Record}.
  - **Field to Update**:
    - Webinar_End_Date_and_Time__c = {! Get_Campaign_Webinar_End_Date_and_Time.Webinar_End_Date_and_Time__c}.

1.3.3 Default Path Handling:

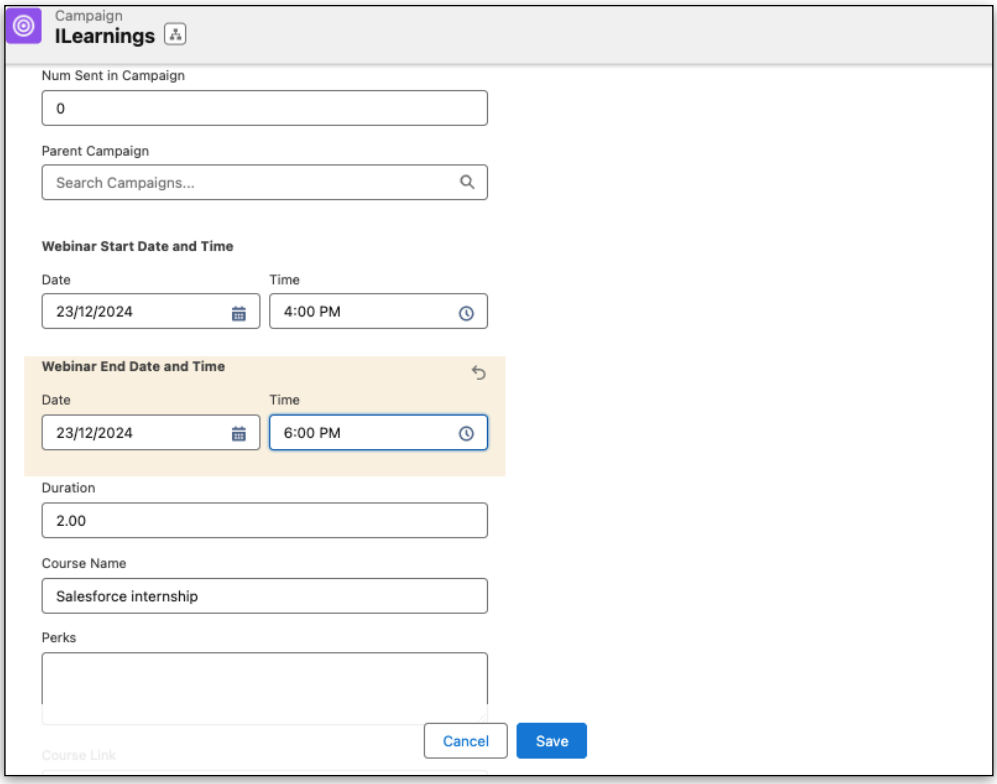
- If the Webinar_End_Date_and_Time__c field on the Campaign is blank, do nothing or log an error.

1.3.4 Save and Activate

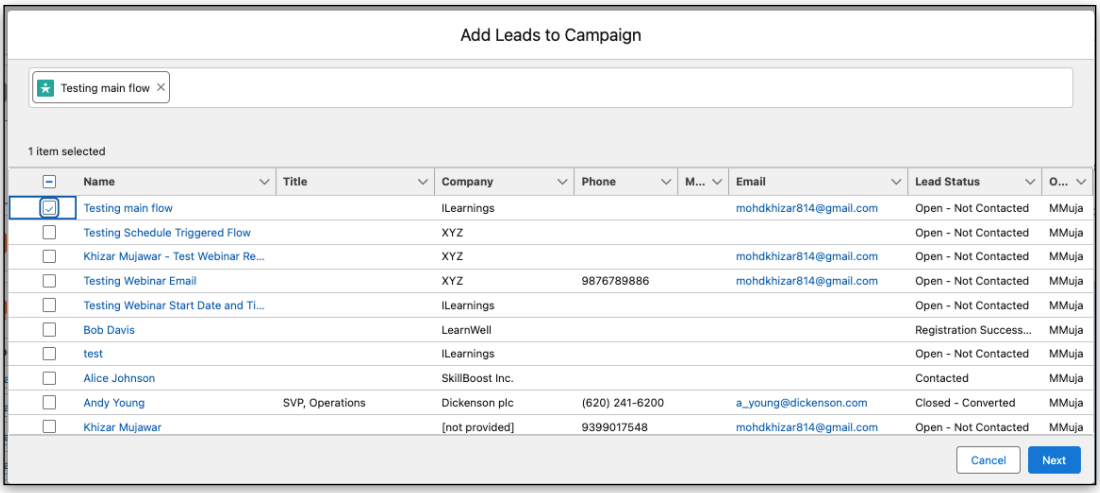
- **Save the Flow**:
  - **Name:** Populate Webinar End Date on Campaign Members.
- **Test the Flow**:
  - Create or update Campaign Members associated with a Campaign having a Webinar_End_Date_and_Time__c value.
  - Verify the field is correctly populated.
- **Activate the Flow**.

1.3.5 Testing if Webinar End Date and Time in Campaign Member Object is Populated:

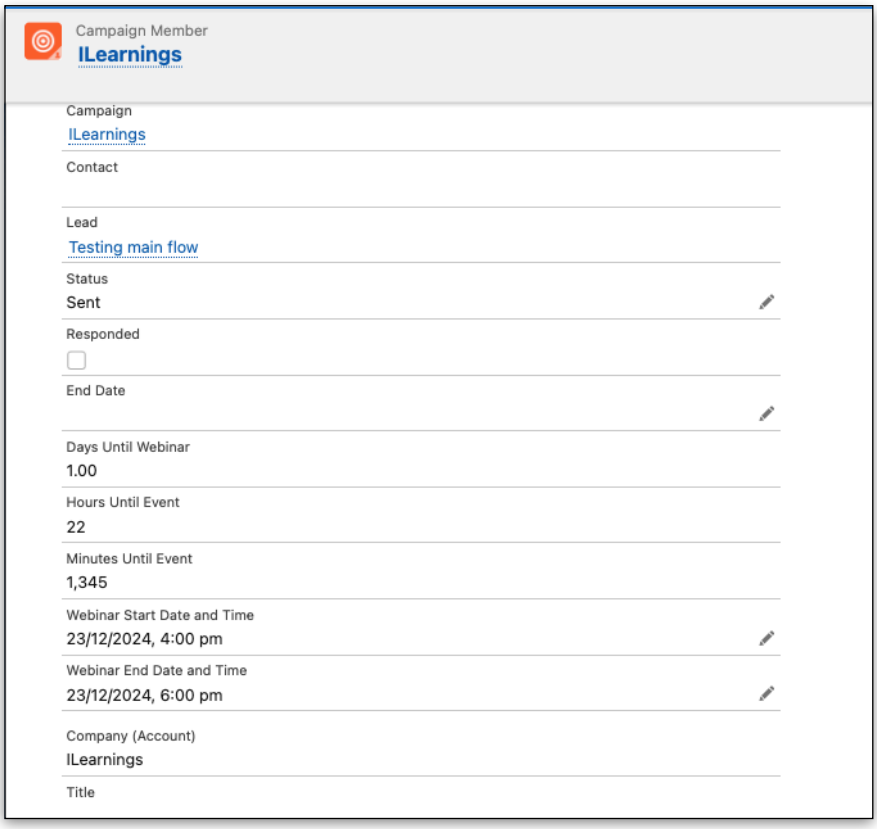
- Enter 'Webinar End Date and Time' in ILearnings campaign record.



- Add random lead record into campaign member.



- Webinar End Date and Time automatically populated from Webinar Date and Time of campaign



1.4: Creating Helper Field:

Add helper formula fields to calculate the time elapsed since the webinar start:

Hours Since Webinar:

- **Field Name**: Hours_Since_Webinar__c
- **Type**: Formula (Number)
- **Decimal Places**: 0
- **Formula**:

```
IF(
   ISBLANK(Webinar_Start_Date_and_Time__c),
   NULL,
   IF(
      NOW() < Webinar_Start_Date_and_Time__c,
      NULL,
      FLOOR((NOW() - Webinar_Start_Date_and_Time__c) * 24)
   )
)
```

These field will determine how much time has passed since the webinar started.

Step-2: Creating a Scheduled-Triggered Flow to Send an Email After 1, 6, and 24 Hours of the Webinar

Create a scheduled-triggered flow to send emails at the specified intervals after the webinar.

2.1 Flow Configuration

- **Flow Type**: Scheduled-Triggered Flow.
  - **Frequency**: Daily.
  - **Start Time**: 7:30 PM or any time of your choice.
- **Trigger Object**: Campaign Member.
- **Entry Conditions**:
  - Webinar_Start_Date_and_Time__c IS NOT NULL.

2.2 Decision Element

Add a decision element to determine which email should be sent based on the time elapsed since the webinar.

- **Label:** What time is it?
- **Outcomes:**
  - **1 Hour After Webinar**:
    - **Condition**: {!$Record.Hours_Since_Webinar__c} = 1
  - **6 Hours After Webinar**:
    - **Condition**: {!$Record.Hours_Since_Webinar__c} = 6
  - **24 Hours After Webinar**:
    - **Condition:** {!$Record.Hours_Since_Webinar__c} = 24

2.3 Add Send Email Alert Flow Element

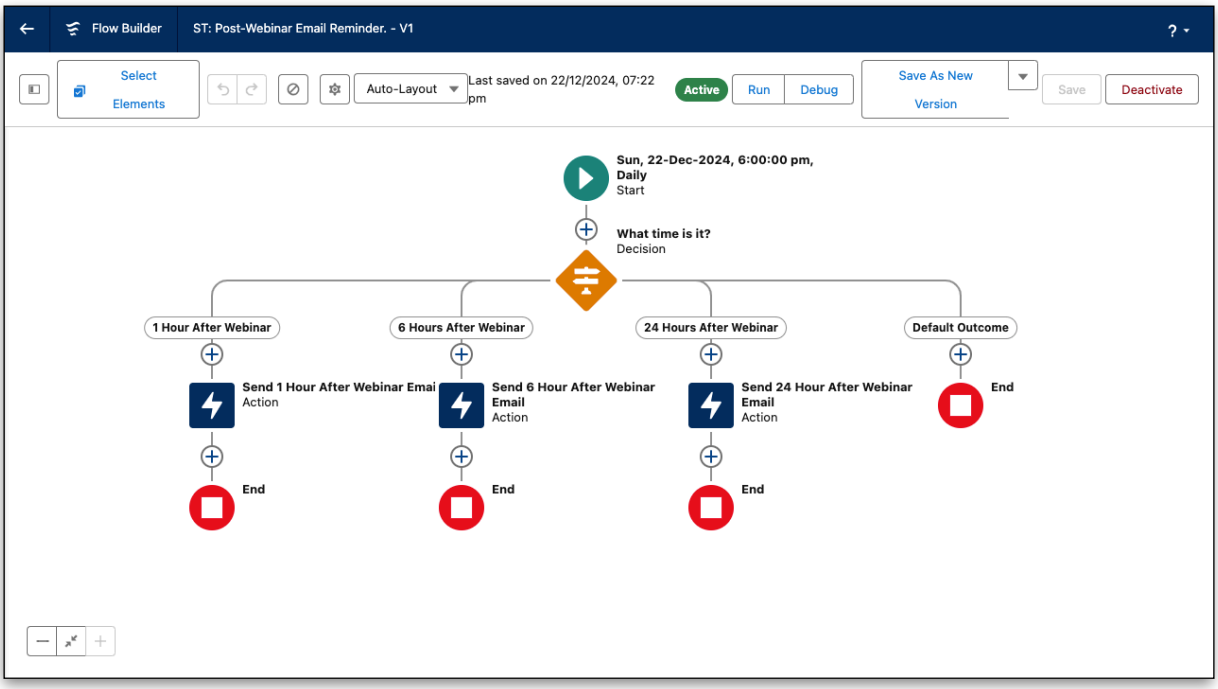
Add actions under each outcome to send the respective email alerts:

- **1 Hour After Webinar**:
  - Label: Send 1 Hour After Webinar Email.
  - Action: Send an Email Alert configured for "1 Hour After Webinar."
  - Record ID: Campaign Member {$Record.Id}.
- **6 Hours After Webinar**:
  - Label: Send 6 Hours After Webinar Email.
  - Action: Send an Email Alert configured for "6 Hours After Webinar."
  - Record ID: Campaign Member {$Record.Id}.
- **24 Hours After Webinar**:
  - Label: Send 24 Hours After Webinar Email.

- Action: Send an Email Alert configured for "24 Hours After Webinar."
- Record ID: Campaign Member {$Record.Id}.

2.4 Save and Activate the Flow

- **Save the Flow**:
  - Name: ST: Post-Webinar Email Reminder.
- **Activate**:
  - Ensure the flow is activated for production use.



2.5 Testing

- **Create Test Data**:
  - Assign webinar start dates in the past for Campaign Members.
- **Validate Email Triggers**:
  - Confirm emails are sent at 1 hour, 6 hours, and 24 hours post-webinar.

VALIDATION RULES

Validation Rules

***Definition*:** Validation rules are used to verify if the data entered by the user meet specified criteria, before record is saved into the database.

Requirement:

If **Registration Status = 'Successful'**, the following fields must be **mandatory** :

- **Registration Date**
- **Fees Paid by Candidate**
- **Batch Start Date**

IMPLEMENTATION STEPS:

STEP-1: Create the Registration_Status__c Field

- **Create a New Field:**
  - Go to **Object Manager > Fields & Relationships** > New > Select **Picklist** as the data type
- **Field Details:**
  - **Field Label:** Registration Status
  - **Field Name:** Registration_Status__c (auto-generated).
  - **Values**: Choose Enter values with each value separated by a new line and input:

    Pending

    Successful

    Failed

    Cancelled
  - **Field Requirement:** Leave unchecked (we'll enforce conditions using validation rules later).

STEP-2: Create Validation Rule

1. **Navigate to Salesforce Setup:**
   - Go to **Object Manager** > **Lead** (or the appropriate object).

- Click on **Validation Rules** in the left panel.
- Click **New** to create a new validation rule.

2. **Validation Rule Details:**
   - **Rule Name:** Mandatory_Fields_Successful_Registration
   - **Error Condition Formula:**

```
AND(
  ISPICKVAL(Registration_Status__c, "Successful"),
  OR(
    ISBLANK(Registration_Date__c),
    ISBLANK(Fees_Paid_by_Candidate__c),
    ISBLANK(Batch_Start_Date__c)
  )
)
```

Explanation:

- **ISPICKVAL(Registration_Status__c, "Successful"):** Checks if the **Registration Status** field equals **Successful**.
- **ISBLANK(...):** Ensures the fields are not empty.
- **OR(...):** Combines the conditions for all three fields, triggering the error if any of them is blank.

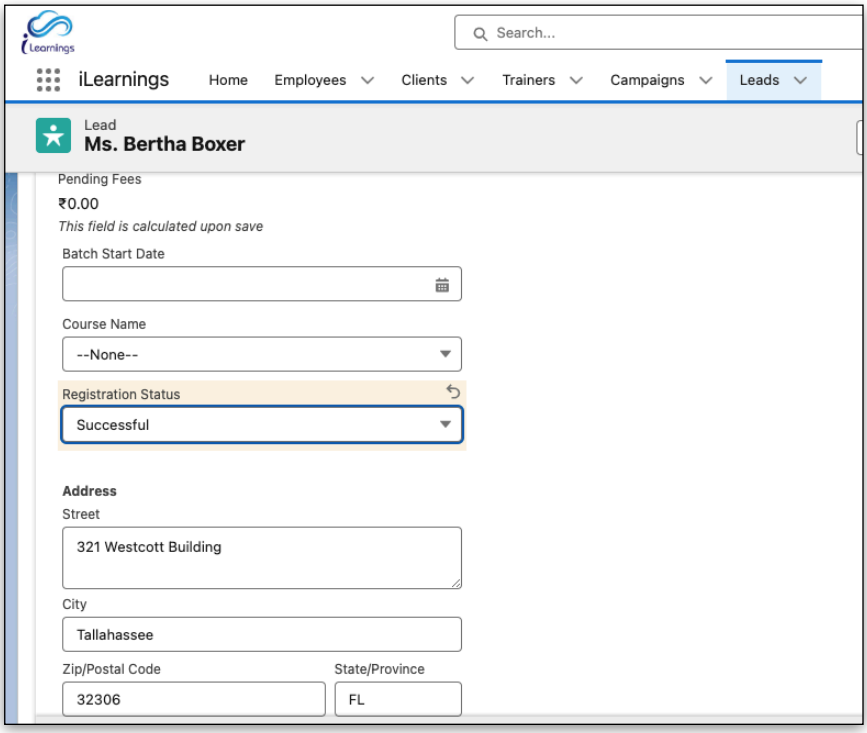
3. **Error Message:**

   “Please fill in Registration Date, Fees Paid by Candidate, and Batch Start Date when Registration Status is ‘Successful’.”

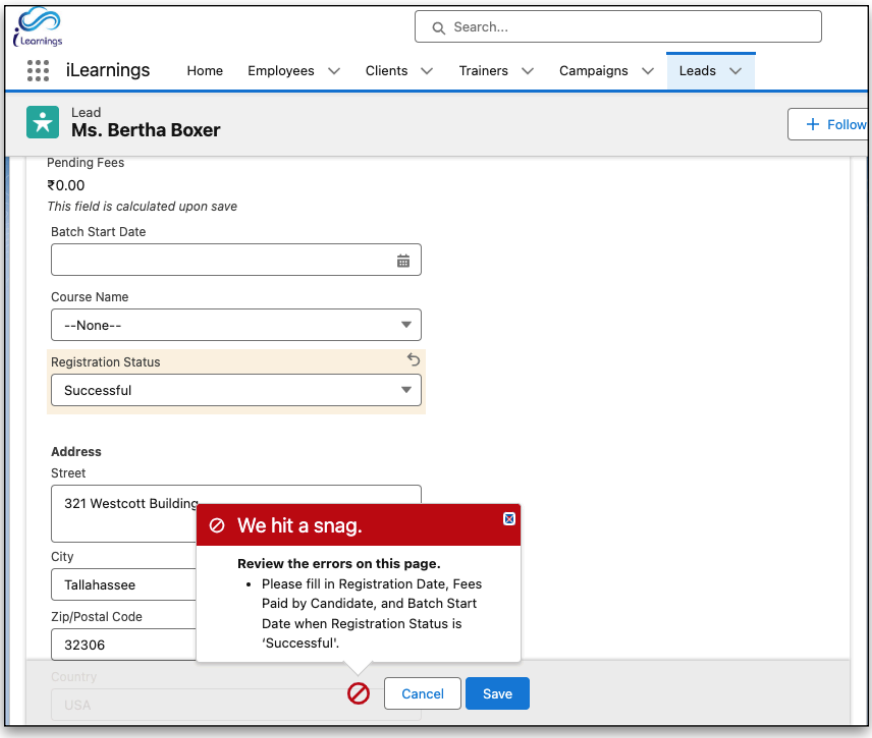
4. **Error Location:** Select **Top of Page** or associate it with a specific field.
5. **Save the Rule.**

STEP-3: Testing the Validation Rule

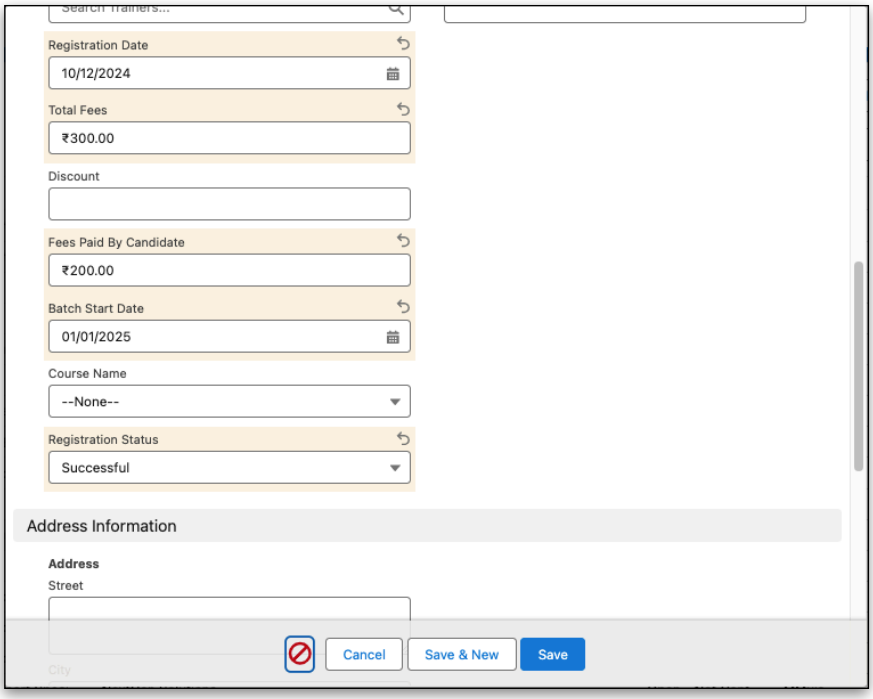
- Create or edit a record.
- Set **Registration Status** to **Successful** and leave any of the mandatory fields blank.

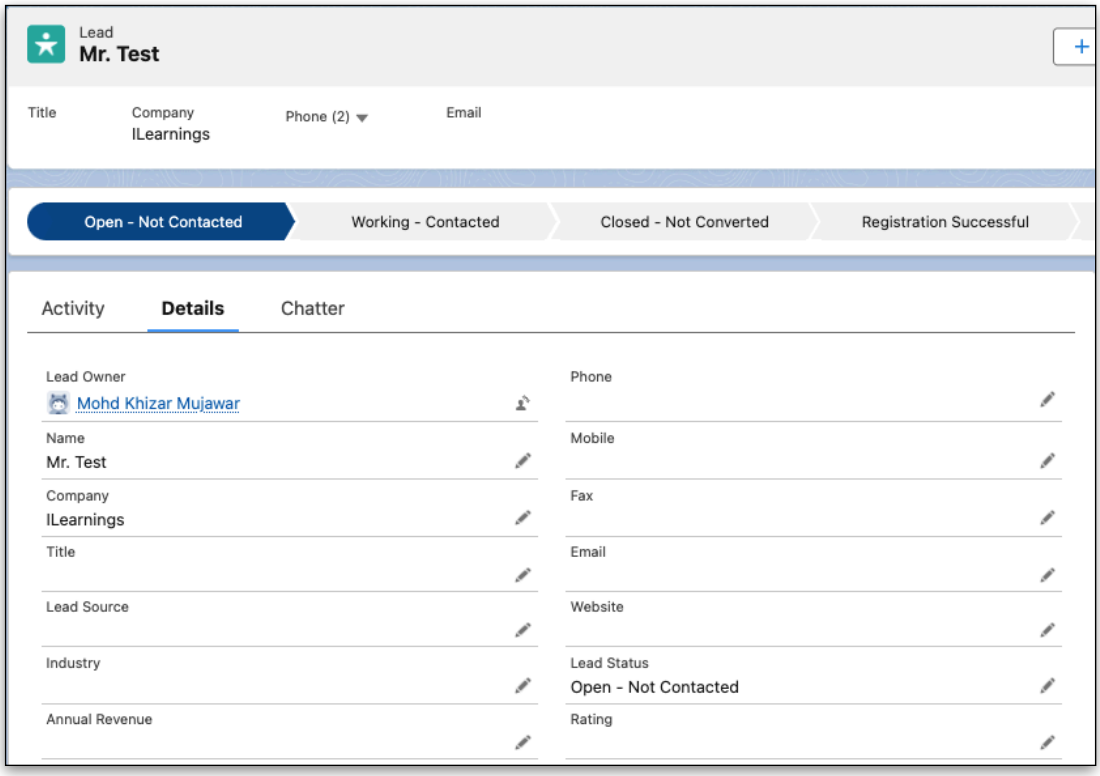


- Attempt to save the record—an error message should appear.



- Fill in all mandatory fields and save again—the record should save successfully.





LIGHTNING WEB DEVELOPMENT (LWC)

Requirement:

Develop a Lightning Web Component to Delete Duplicate Attachments.

Solution:

1. **Duplicate Identification**:
   - Apex groups files by their title to identify duplicates.
   - Files with the same title (more than one entry) are flagged as duplicates.
2. **Interactive UI**:
   - LWC uses lightning-datatable to display duplicate files.
   - Users can select files for deletion and initiate the delete operation.
3. **Error Handling**:
   - Apex and LWC include error handling mechanisms.
   - Toast messages provide feedback on operations like deletion success or errors.
4. **Efficient Data Management**:
   - Cacheable Apex methods ensure optimal performance for fetching data. • Selected files are processed in bulk to minimize resource usage.

IMPLEMENTATION STEPS:

STEP-1: Create a Salesforce Project

- Open VS Code.
- Open Command Palette (cmd+shift+P for Mac).
- Type **'SFDX: Create Project'** and select the Standard Template. • Enter the project name: **'iLearnings App'**.

STEP-2: Create the LWC Component

- Open Command Pallet (cmd+shift+P for Mac).
- Type **'SFDX: Create Lightning Web Component'**
- Enter Component name **'deleteDuplicateAttachment'** (use camel case).

STEP-3: Authorize Your Org

- Open Command Palette (cmd+shift+P for Mac).
- Type **'SFDX: Authorize an Org'**
- Select Production.
- Enter org Alias **'ilearnings org'**
- Log in with your Salesforce account where you are currently working.

STEP-4: Code

Apex Class:

The DeleteDuplicateAttachmentsController class identifies and deletes duplicate attachments:

- **fetchDuplicates():** Groups ContentDocument records by title to find duplicates.
- **deleteDuplicates():** Deletes selected duplicates using a list of file IDs.

DeleteDuplicateAttachmentsController.cls

```
public with sharing class DeleteDuplicateAttachmentsController {

    @AuraEnabled(cacheable=true)
    public static List<ContentDocument> fetchDuplicates() {
        // Map to group files by their title
        Map<String, List<ContentDocument>> fileMap = new Map<String, List<ContentDocument>>();
        List<ContentDocument> duplicateFiles = new List<ContentDocument>();

        // Query for all attachments grouped by Title to find duplicates
        for (ContentDocument doc : [SELECT Id, Title FROM ContentDocument]) {
            if (!fileMap.containsKey(doc.Title)) {
                fileMap.put(doc.Title, new List<ContentDocument>());
            }
            fileMap.get(doc.Title).add(doc);
        }

        // Add duplicates (more than 1 file with the same title) to the result list
        for (String title : fileMap.keySet()) {
            if (fileMap.get(title).size() > 1) {
                duplicateFiles.addAll(fileMap.get(title));
            }
        }

        return duplicateFiles;
    }

    @AuraEnabled
    public static void deleteDuplicates(List<Id> fileIds) {
        try {
            System.debug('File IDs to delete: ' + fileIds);
            List<ContentDocument> filesToDelete = [SELECT Id FROM ContentDocument WHERE Id IN :fileIds];
            System.debug('Files to delete: ' + filesToDelete);
            delete filesToDelete;
        } catch (Exception e) {
            System.debug('Error deleting files: ' + e.getMessage());
            throw new AuraHandledException('Error deleting files: ' + e.getMessage());
        }
    }

}

```

Lightning Web Component

The DeleteDuplicateAttachments LWC provides an interactive UI for managing duplicates:

1. Fetches duplicate files via Apex.
2. Displays duplicates in a lightning-datatable.
3. Allows users to select and delete duplicates.

HTML: deleteDuplicateAttachments.html

```
<template>
    <lightning-card title="Delete Duplicate Attachments" icon-name="utility:delete">
        <!-- Spinner -->
        <template if:true={isLoading}>
            <lightning-spinner alternative-text="Loading" size="medium"></lightning-spinner>
        </template>

        <!-- Display Duplicate Files -->
        <template if:true={duplicates.length}>
            <p><strong>Found Duplicate Files:</strong></p>
            <lightning-datatable
                key-field="Id"
                data={duplicates}
                columns={columns}
                onrowselection={handleRowSelection}>
            </lightning-datatable>
            <lightning-button
                label="Delete Selected"
                variant="brand"
                onclick={handleDelete}>
            </lightning-button>
        </template>

        <!-- No Duplicates Message -->
        <template if:false={duplicates.length}>
            <p>No duplicate files found.</p>
        </template>
    </lightning-card>
</template>

```

JS: deleteDuplicateAttachments.js

```
import { LightningElement, track } from 'lwc';
import fetchDuplicates from '@salesforce/apex/DeleteDuplicateAttachmentsController.fetchDuplicates';
import deleteDuplicates from '@salesforce/apex/DeleteDuplicateAttachmentsController.deleteDuplicates';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';

export default class DeleteDuplicateAttachments extends LightningElement {
    @track duplicates = [];
    @track selectedRecords = [];
    @track isLoading = false;

    columns = [
        { label: 'File Name', fieldName: 'Title', type: 'text' },
        { label: 'File ID', fieldName: 'Id', type: 'text' }
    ];

    connectedCallback() {
        this.loadDuplicateFiles();
    }

    // Load duplicate files
    loadDuplicateFiles() {
        this.isLoading = true;
        fetchDuplicates()
            .then((result) => {
                this.duplicates = result;
            })
            .catch((error) => {
                this.showToast('Error', 'Error fetching duplicates', 'error');
                console.error(error);
            })
            .finally(() => {
                this.isLoading = false;
            });
    }

    // Handle row selection in datatable
    handleRowSelection(event) {
        this.selectedRecords = event.detail.selectedRows.map((row) => row.Id);
    }

    // Handle delete action
    handleDelete() {
        if (this.selectedRecords.length === 0) {
            this.showToast('Warning', 'Please select files to delete', 'warning');
            return;
        }

        this.isLoading = true;
        deleteDuplicates({ fileIds: this.selectedRecords })
            .then(() => {
                this.showToast('Success', 'Selected duplicates deleted successfully', 'success');
                this.loadDuplicateFiles(); // Reload duplicates after deletion
            })
```

```
            .catch((error) => {
                this.showToast('Error', 'Error deleting files', 'error');
                console.error(error);
            })
            .finally(() => {
                this.isLoading = false;
            });
    }

    // Utility method for showing toast messages
    showToast(title, message, variant) {
        this.dispatchEvent(
            new ShowToastEvent({
                title,
                message,
                variant
            })
        );
    }
}

```

XML: deleteDuplicateAttachments.xml

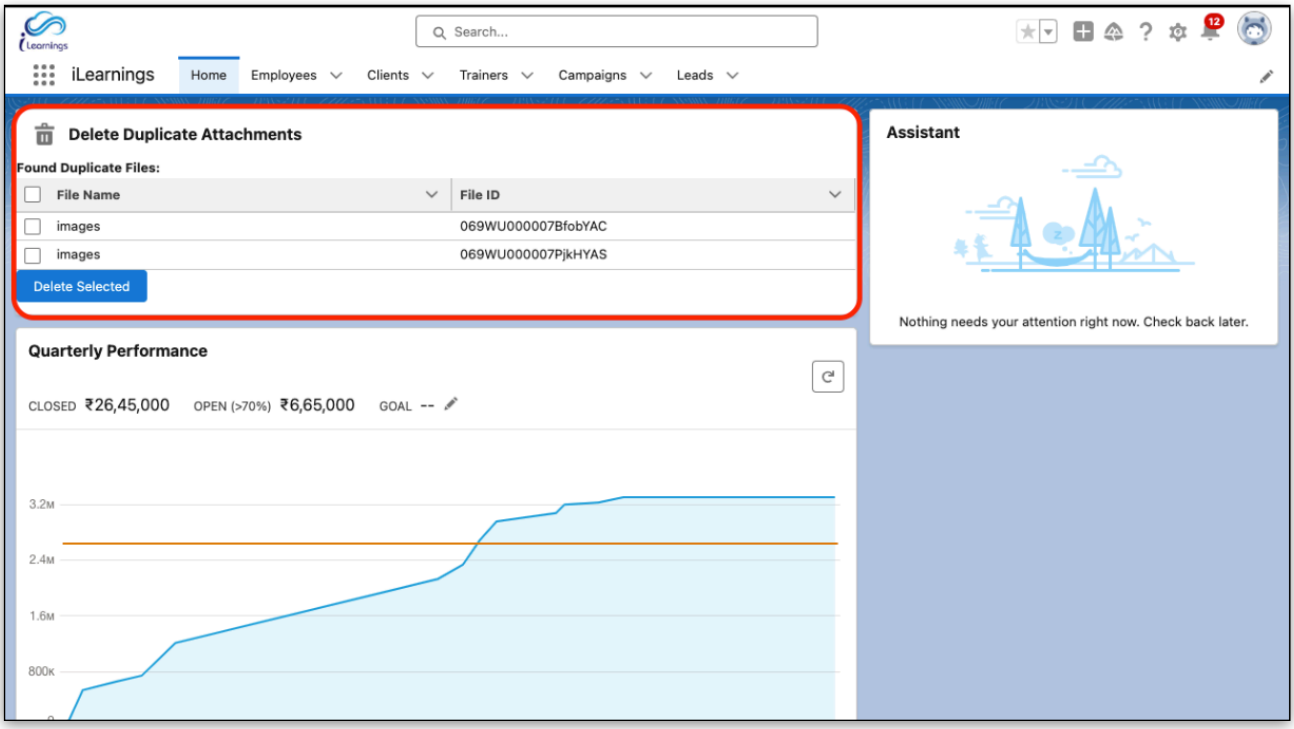
```
<?xml version="1.0" encoding="UTF-8"?>
<ApexClass xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>62.0</apiVersion>
    <status>Active</status>
</ApexClass>

```

STEP-5: Deploy Source Code to Default Org

- Enter "**sf project deploy start**" command in terminal to deploy source code.

STEP-6: Add the created component

- Go to ilearnings App Home Page in Salesforce.
- Open the Lightning App Builder.
- Drag and Drop the ‘deleteDuplicateFileAttachment' Component. • Save and Activate



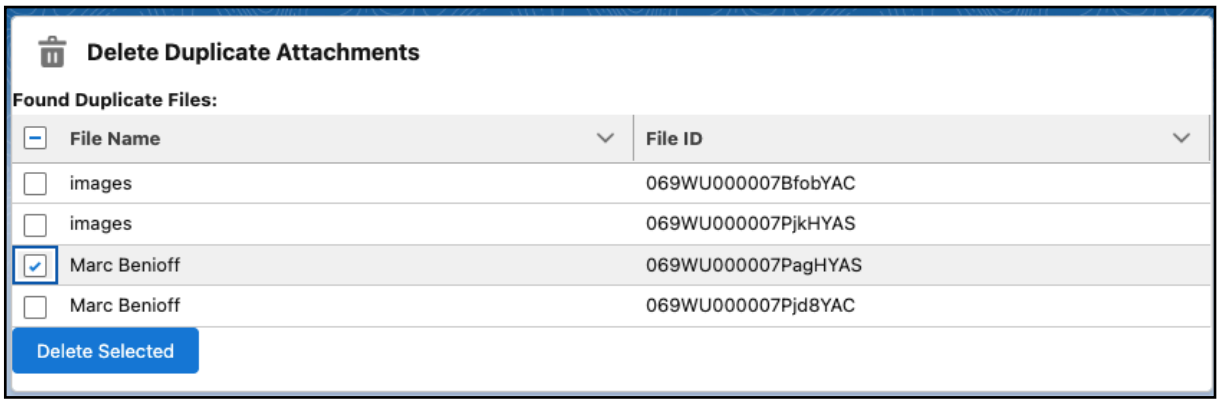
STEP-7 (Testing): Adding Attachments

- Salesforce Classic provides the **Notes & Attachments** related list to upload attachments.
- Navigate to the record where you want to add the attachment (e.g., an Account, Contact, or custom object record).

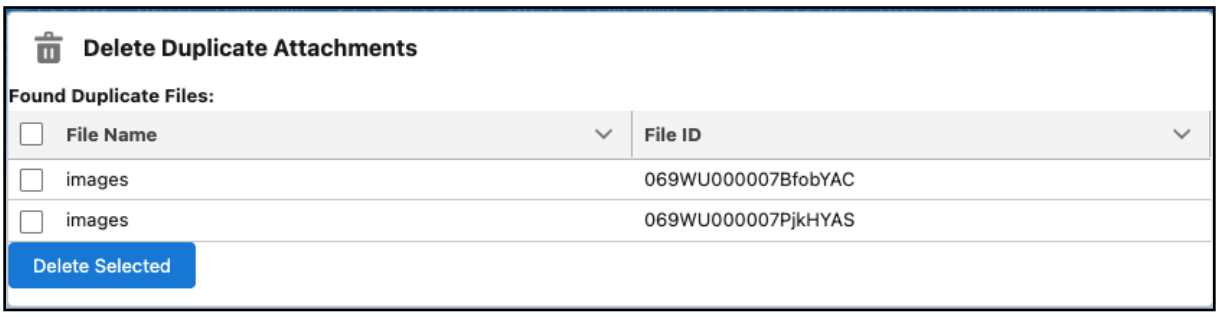
Lets, Add duplicate images for **Marc Benioff**:

- Go to the **Marc Benioff** contact record's related section.
- Select and Upload the following Marc's image **twice**.

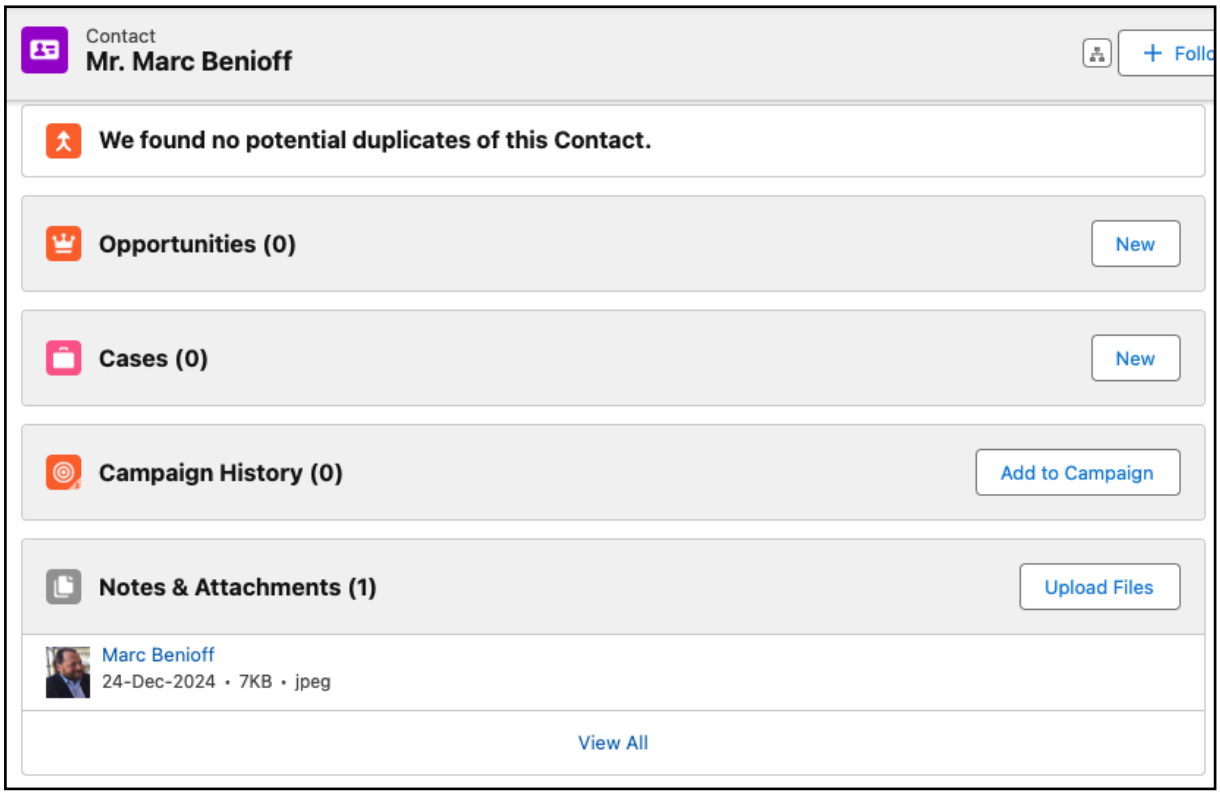
- Navigate to the **Home Page** of the iLearnings app.
- You will see two duplicate Marc images listed.



- Select one of the duplicate images and delete it.
- Refresh the home page



- Verify that the duplicate image has been deleted from Marc's contact **Notes & Attachments** section.



APEX TRIGGERS

Apex Scenarios

9.1 **Scenario 1: Account Trigger:** Create default contacts equal to the number of employees.

9.2 **Scenario 2: Opportunity Trigger:** Enforce future closed dates.

9.3 **Scenario 3: Product Trigger:** Set default pricebook entry with price 15.

9.4 **Scenario 4: Contact Trigger:** Update account name by appending last name.

9.5 **Scenario 5: Enhanced Contact Trigger:** Remove last name from account name upon contact deletion.

Steps to create the Trigger in Salesforce

1. **Navigate to the Object Manager:**
   - In Salesforce Setup, go to Object Manager and find the object you want to create the trigger for.
2. **Open Developer Console:**
   - Click your avatar in the top-right corner.
   - Select Developer Console.
3. **Create a New Trigger:**
   - In Developer Console, go to File → New → Apex Trigger.
   - Name your trigger (e.g., AccountTrigger) and choose the object it will act on.

Trigger-1: Create Default Contacts for Accounts

STEP-1: Create ‘No Of Employee’ Custom field with the Datetype: Number

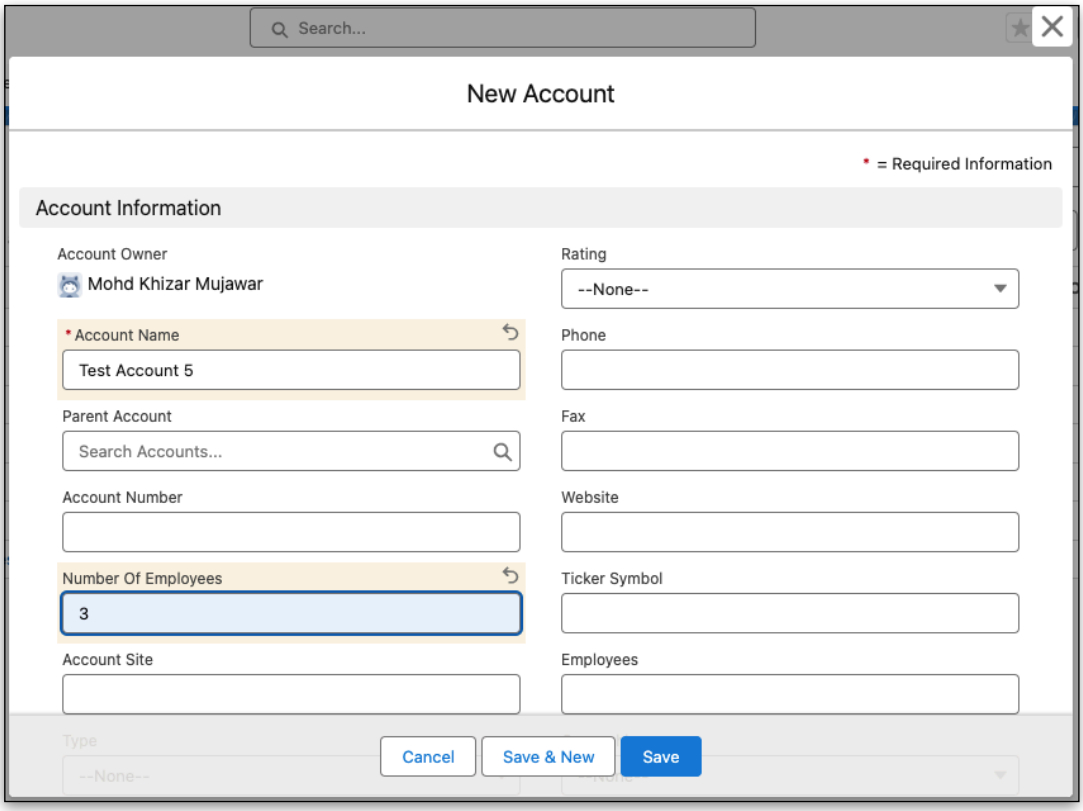
STEP-2: Create Apex Trigger.

CreateDefaultContacts.apxt

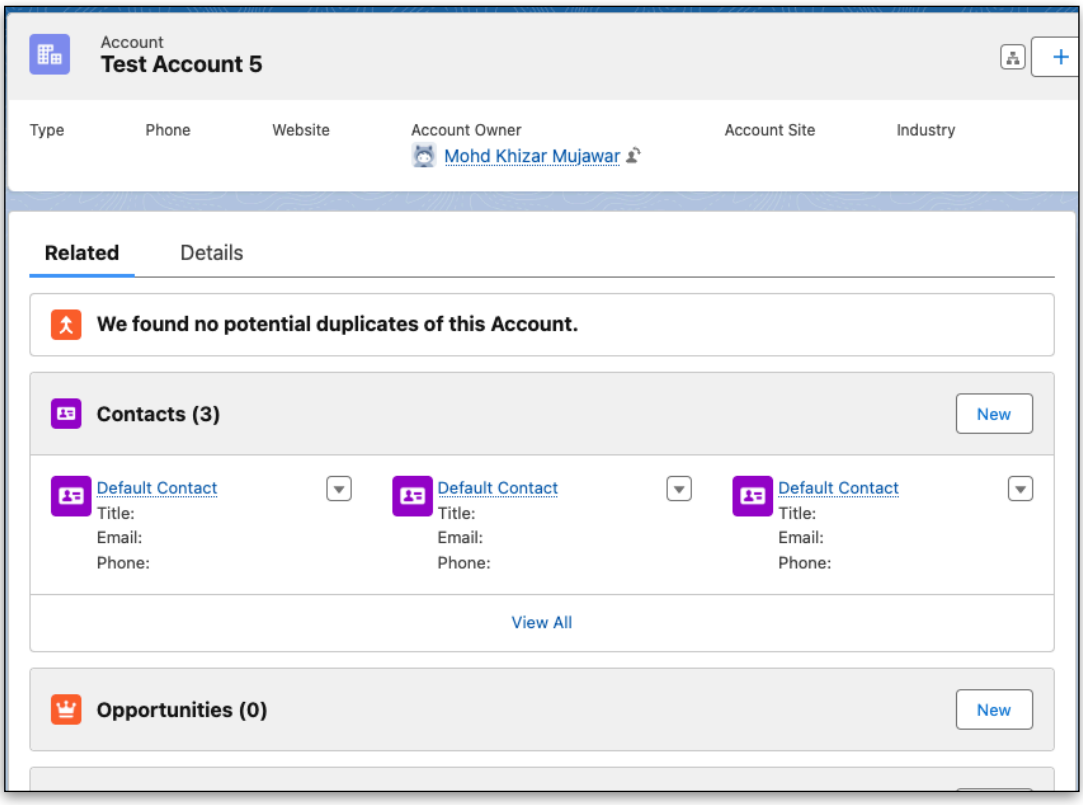
```
trigger CreateDefaultContacts on Account (after insert) {
    List<Contact> contactsToInsert = new List<Contact>();

    for (Account acc : Trigger.new) {
        // Check if NumberOfEmployees is greater than zero
        if (acc.Number_Of_Employees__c != null && acc.Number_Of_Employees__c > 0) {
            for (Integer i = 0; i < acc.Number_Of_Employees__c; i++) {
                // Create a new Contact for each employee
                contactsToInsert.add(new Contact(
                    FirstName = 'Default',
                    LastName = 'Contact',
                    AccountId = acc.Id
                ));
            }
        }
    }

    // Insert the Contacts
    if (!contactsToInsert.isEmpty()) {
        try {
            insert contactsToInsert;
        } catch (DmlException e) {
            System.debug('Error while creating default contacts: ' + e.getMessage());
        }
    }
}

```

Testing Trigger:

1. Create a new **Account** record and Enter value in the **Number of Employees** field.



2. After saving the account, the trigger will automatically create **default contacts** for the account and the number of default contacts created will be equal to the value entered in the **Number of Employees** field.



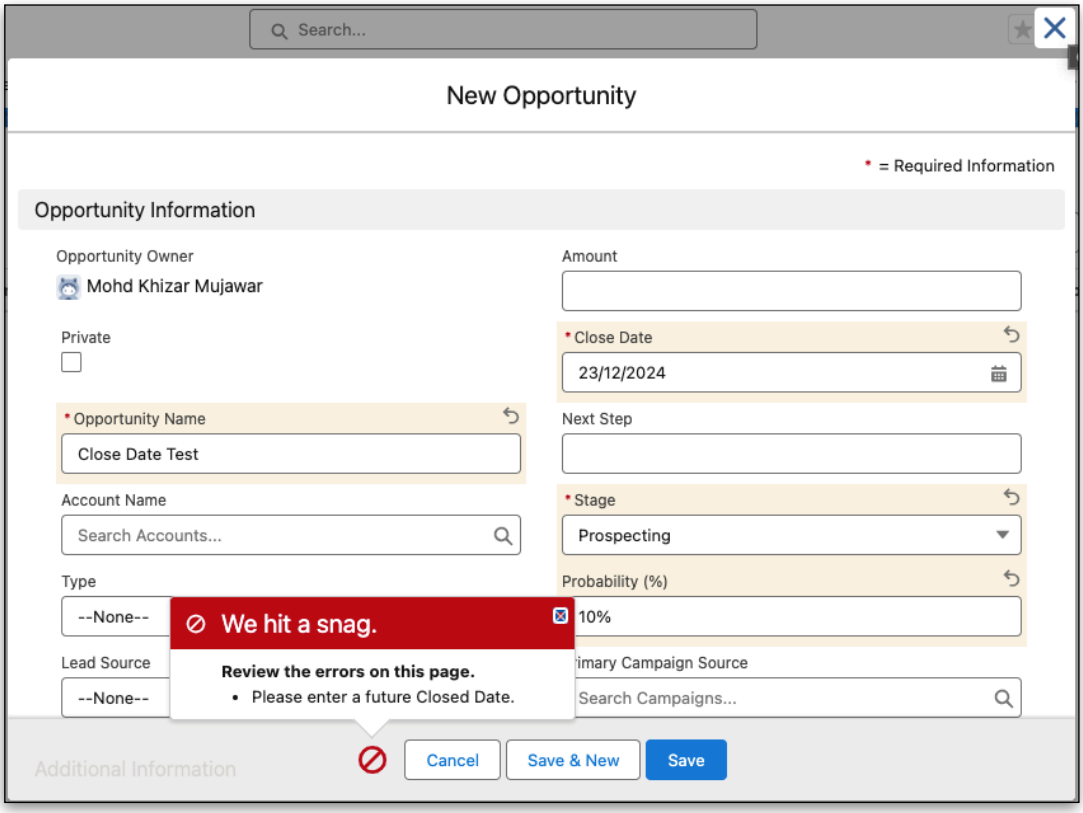
Trigger-2: Enforce future Closed Date on Opportunity

EnforceFutureClosedDate.apxt

```
trigger EnforceFutureClosedDate on Opportunity (before insert, before update) {
    for (Opportunity opp : Trigger.new) {
        // Check if ClosedDate is in the past
        if (opp.CloseDate != null && opp.CloseDate < Date.today()) {
            opp.addError('Please enter a future Closed Date.');
        }
    }
}
```

Testing Trigger:

If we attempt to create or update an Opportunity with a Closed Date in the past, the trigger will prevent the operation by throwing an error with the message: 'Please enter a future Closed Date.



Trigger-3: When a new Product2 record is created, automatically create a PricebookEntry in the "Standard Pricebook" with a default price of **15**. This simplifies the setup for users by ensuring every new product has a default price set in the "Standard Pricebook."

SetDefaultPricebookEntry.apxt

```
trigger SetDefaultPricebookEntry on Product2 (after insert) {
    // Get the Standard Pricebook ID
    Id standardPricebookId = [SELECT Id FROM Pricebook2 WHERE IsStandard = TRUE LIMIT 1].Id;

    List<PricebookEntry> pricebookEntriesToInsert = new List<PricebookEntry>();

    for (Product2 product : Trigger.new) {
        // Create a new PricebookEntry for each Product
        pricebookEntriesToInsert.add(new PricebookEntry(
            Pricebook2Id = standardPricebookId,
            Product2Id = product.Id,
            UnitPrice = 15,
            IsActive = true
        ));
    }

    // Insert the Pricebook Entries
    if (!pricebookEntriesToInsert.isEmpty()) {
        try {
            insert pricebookEntriesToInsert;
        } catch (DmlException e) {
            System.debug('Error while creating Pricebook Entries: ' + e.getMessage());
        }
    }
}

```

Working of Trigger:

A Pricebook Entry is created by the trigger as follows:

1. Trigger Fires on Product2 Insert

- The trigger is defined on the Product2 object and is set to fire **after insert**.
- This ensures that a new Product2 record has been created and has a valid Id before the trigger logic executes.

2. Query for the Standard Pricebook

- The trigger queries the Pricebook2 object to retrieve the Id of the Standard Pricebook:
  Id standardPricebookId = [SELECT Id FROM Pricebook2 WHERE IsStandard = TRUE LIMIT 1].Id;

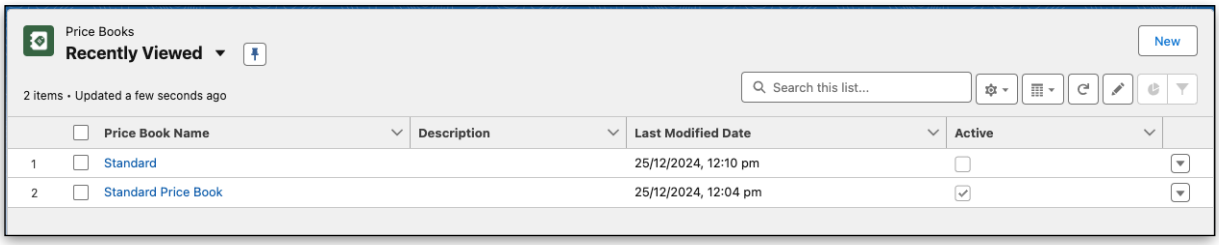
- **Reason:** The Standard Pricebook (IsStandard = TRUE) is required to create a PricebookEntry.

3. **Prepare Pricebook Entry:** For each new product, a **PricebookEntry** record is created with:
    - Pricebook2Id = Standard Pricebook ID
    - Product2Id = Product ID
    - UnitPrice = 15
    - IsActive = true
4. **Insert Pricebook Entries:** The trigger inserts the **PricebookEntry** records into the database.

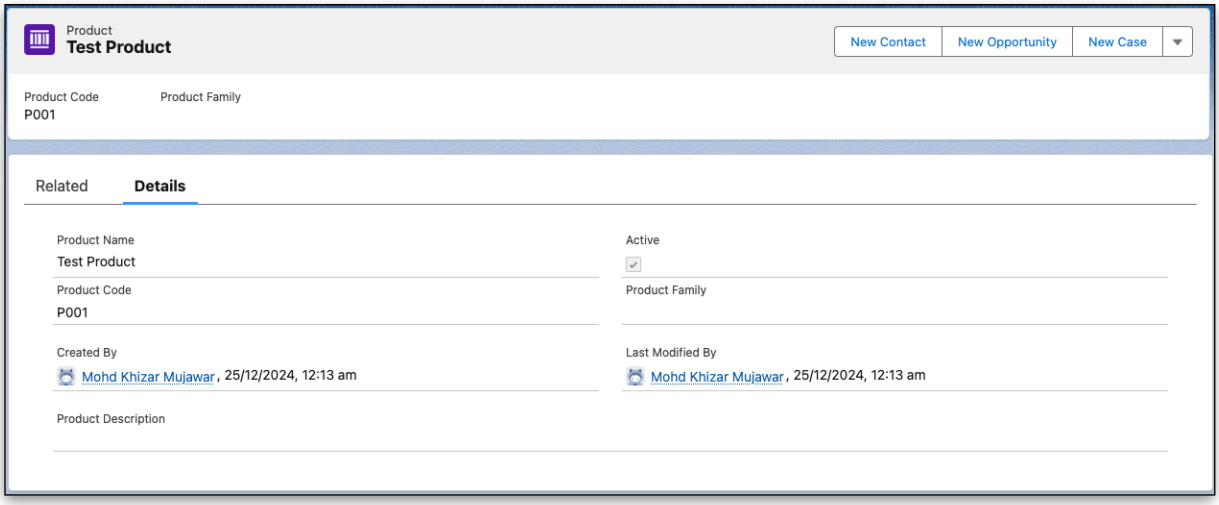
Testing Trigger-3:

To test the SetDefaultPricebookEntry trigger from the UI:

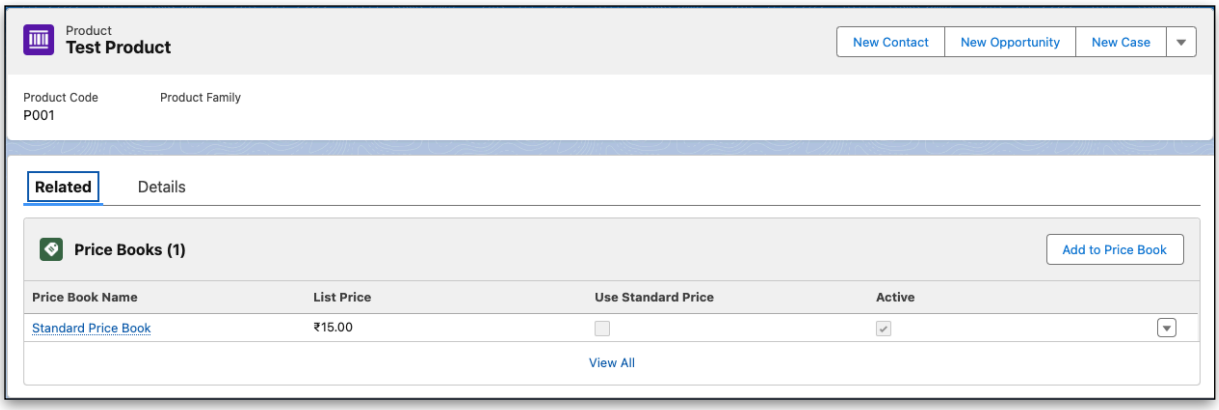
1. **Verify the Standard Pricebook:**
   - Under the **Price Books** tab, Ensure the record named **Standard Pricebook** exists and is active. If its not exist create one, if its not active, mark as active.



2. **Create a New Product:**
   - Go to the **Products** tab. If you can't see product records, create a new tab for products.
   - Create a new record, fill in the required fields (e.g., Product Name), and click **Save**.



3. **Check Pricebook Entries:**
   - Open the newly created product record.
   - Scroll to the **Pricebook Entries** related list and verify:
     - A Pricebook Entry is created.
     - The **Standard Pricebook** is referenced, the **Unit Price** is 15, and it is **Active**.



Trigger-4: Update Account Name with Contact's Last Name

Trigger-5: Remove Contact's Last Name from Account Name

ContactAllEventsTrigger.apxt

```
trigger ContactAllEventsTrigger on Contact (before insert, after insert, after delete) {
    switch on Trigger.operationType {
        when AFTER_INSERT {
            // Trigger-4 Handler Method
            ContactTriggerHandler.afterInsertAction(Trigger.new);
        }
        when AFTER_DELETE {
            // Trigger-5 Handler Method
            ContactTriggerHandler.afterDeleteAction(Trigger.old);
        }
        when else {
        }
    }
}
```

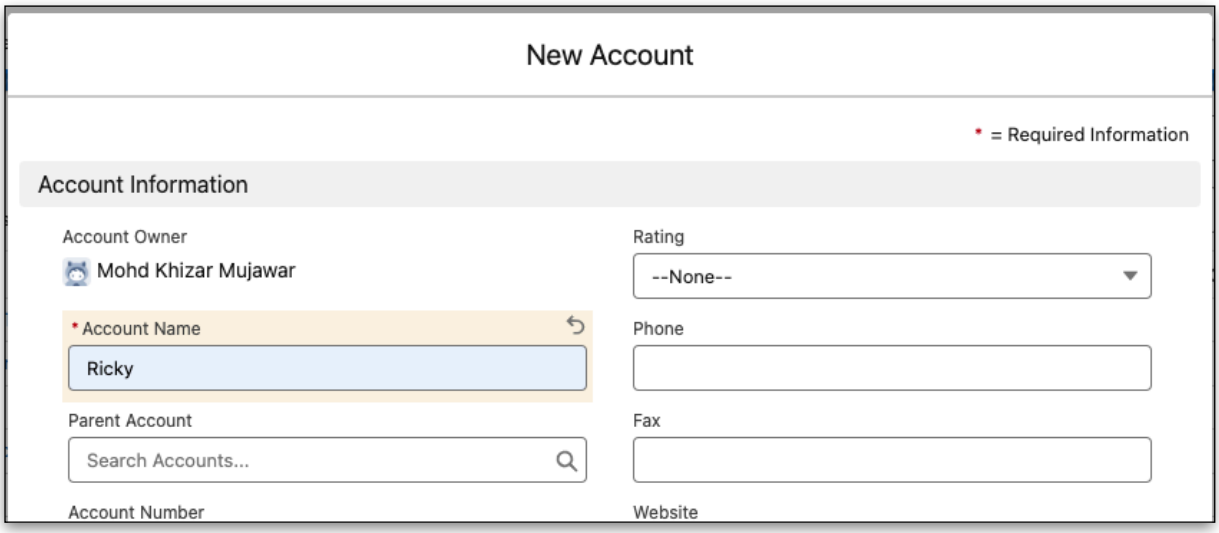
ContactTriggerHandler.apxc

```
public class ContactTriggerHandler {

    public static void afterInsertAction(List<Contact> newContacts) {
        // Collect Account Ids to update
        Set<Id> accountIds = new Set<Id>();
        for (Contact con : newContacts) {
            if (con.AccountId != null) {
                accountIds.add(con.AccountId);
            }
        }

        // If no accounts to update, exit
        if (accountIds.isEmpty()) return;

        // Get the accounts that need to be updated
        Map<Id, Account> accountsToUpdate = new Map<Id, Account>(
            [SELECT Id, Name FROM Account WHERE Id IN :accountIds]
        );

        // Update account names
        for (Contact con : newContacts) {
            if (con.AccountId != null && accountsToUpdate.containsKey(con.AccountId)) {
                Account acc = accountsToUpdate.get(con.AccountId);
                // last name was appending twice,
        //  So use if else to ensure the last name is appended only once
                if (!acc.Name.contains(' ' + con.LastName)) {
                    acc.Name = acc.Name + ' ' + con.LastName;
                }
                accountsToUpdate.put(acc.Id, acc);
            }
        }

        // Update the accounts
        if (!accountsToUpdate.isEmpty()) {
            try {
                update accountsToUpdate.values();
            } catch (Exception e) {
                System.debug('Error updating accounts: ' + e.getMessage());
            }
        }
    }
```

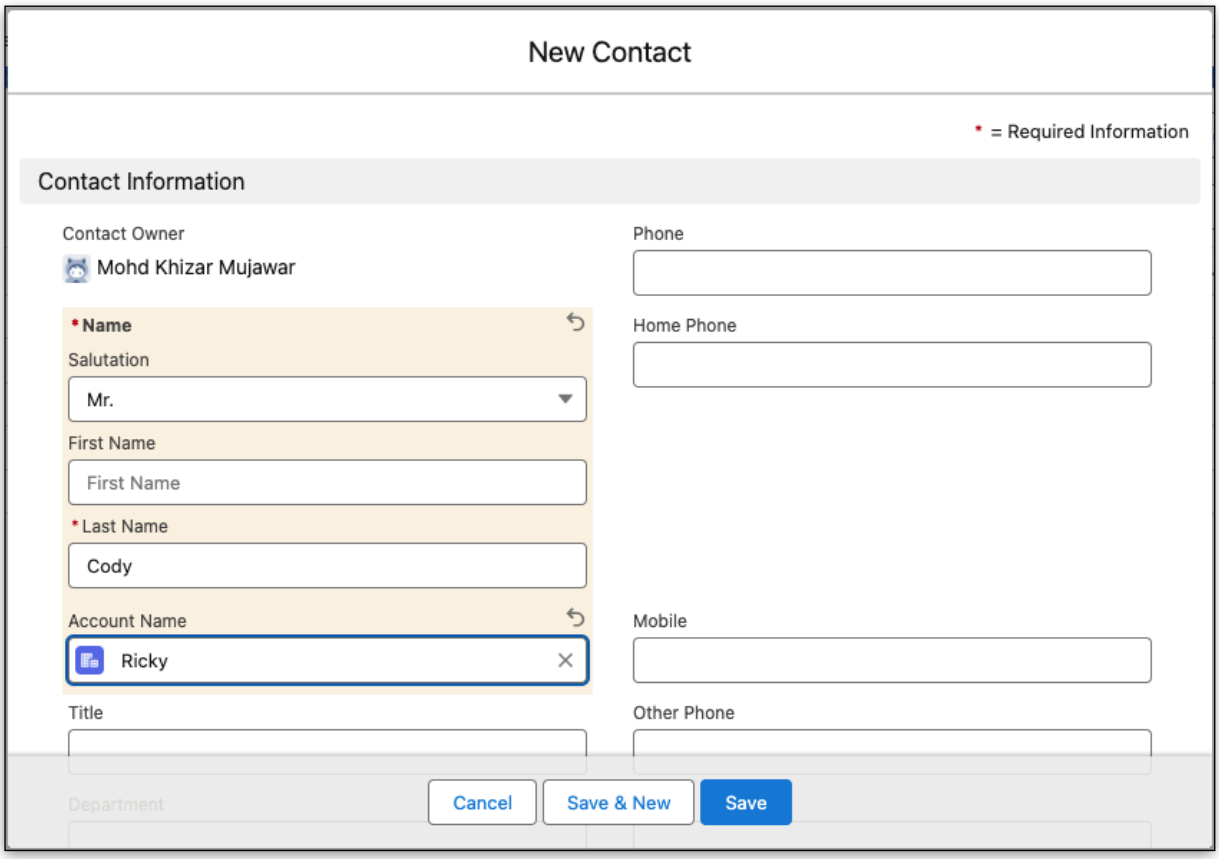
```apex

    public static void afterDeleteAction(List<Contact> oldContacts) {
        // Collect Account Ids to update
        Set<Id> accountIds = new Set<Id>();
        for(Contact con : oldContacts) {
            if(con.AccountId != null) {
                accountIds.add(con.AccountId);
            }
        }

        // If no accounts to update, exit
        if(accountIds.isEmpty()) return;

        // Get the accounts and their related contacts
        Map<Id, Account> accountsToUpdate = new Map<Id, Account>(
            [SELECT Id, Name, (SELECT LastName FROM Contacts)
             FROM Account WHERE Id IN :accountIds]
        );

        // Process each deleted contact
        for(Contact deletedContact : oldContacts) {
            if(deletedContact.AccountId != null &&
            accountsToUpdate.containsKey(deletedContact.AccountId)) {
                Account acc = accountsToUpdate.get(deletedContact.AccountId);
                // Remove the deleted contact's last name from account name
                String updatedName = acc.Name.replace(' ' + deletedContact.LastName, '');
                acc.Name = updatedName;
                accountsToUpdate.put(acc.Id, acc);
            }
        }

        // Update the accounts
        if(!accountsToUpdate.isEmpty()) {
            try {
                update accountsToUpdate.values();
            } catch(Exception e) {
                System.debug('Error updating accounts: ' + e.getMessage());
            }
        }
    }
}
```

Testing Triggers -4:

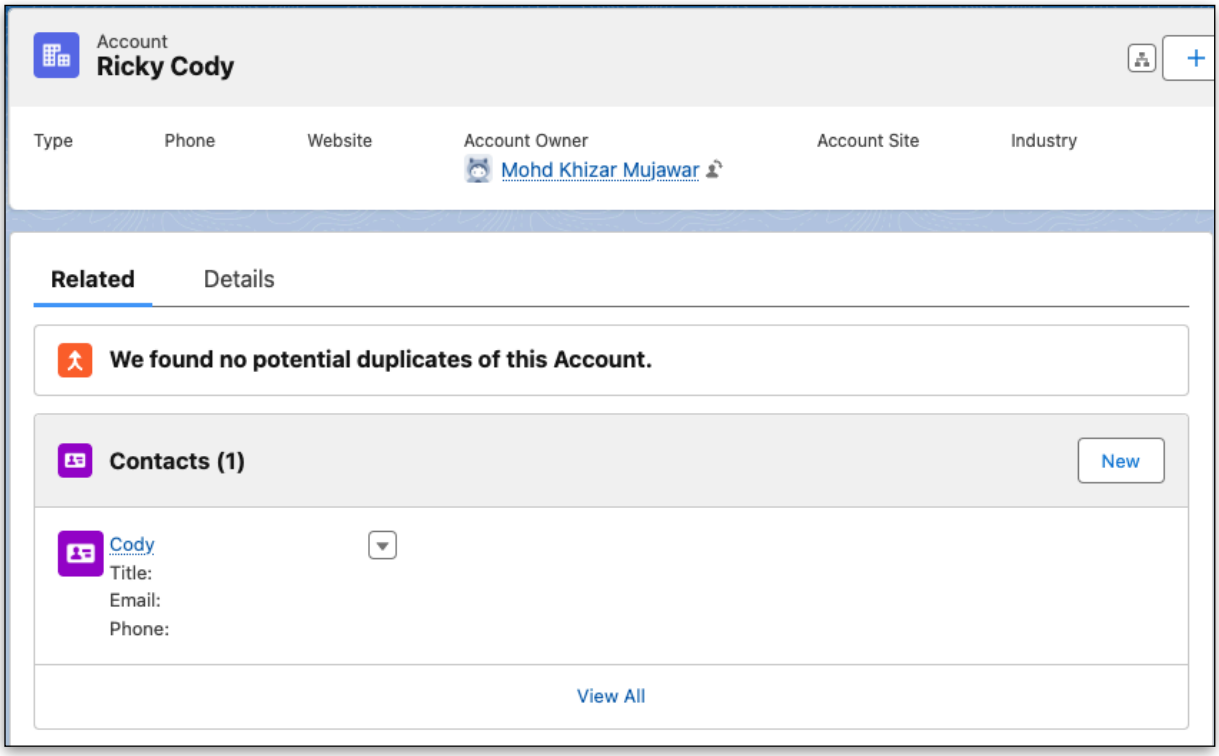
1. Create an account “Ricky"



2. Create a new contact ‘Cody’, & associate that contact with ‘Ricky' account.

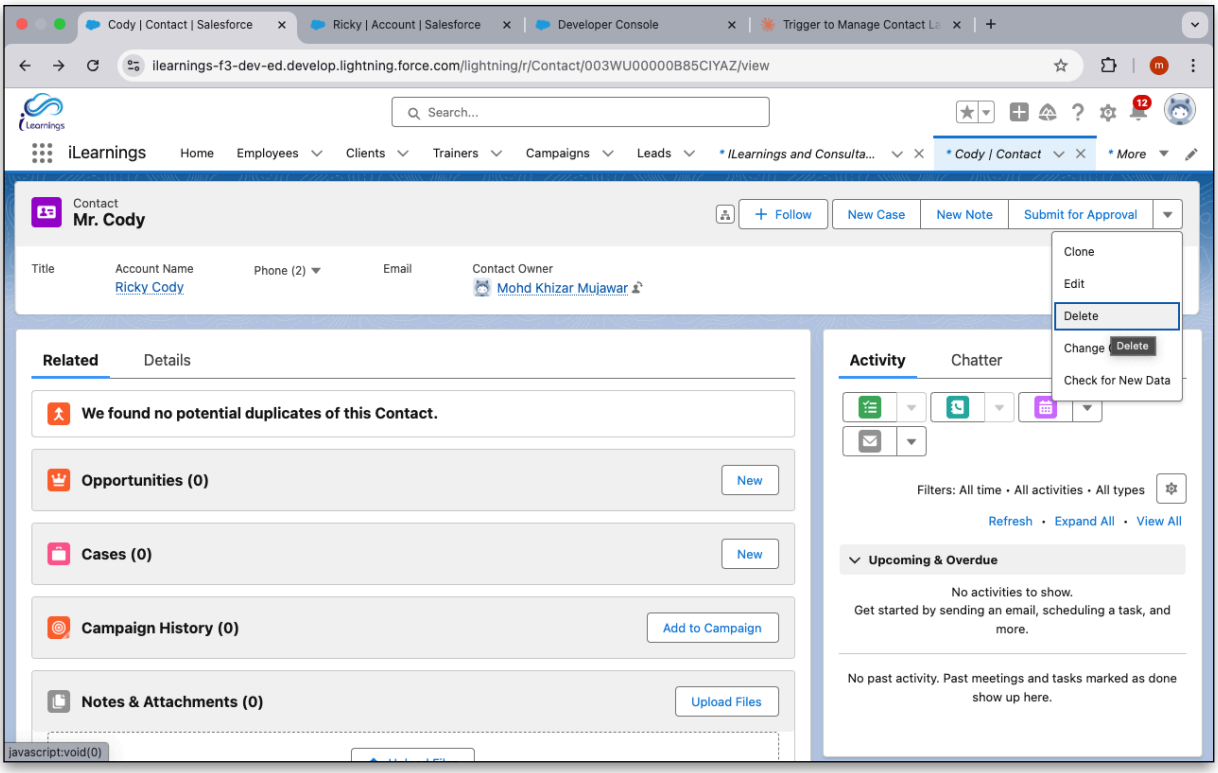


3. Account Name ‘Ricky’ gets updated to ‘Ricky Cody’

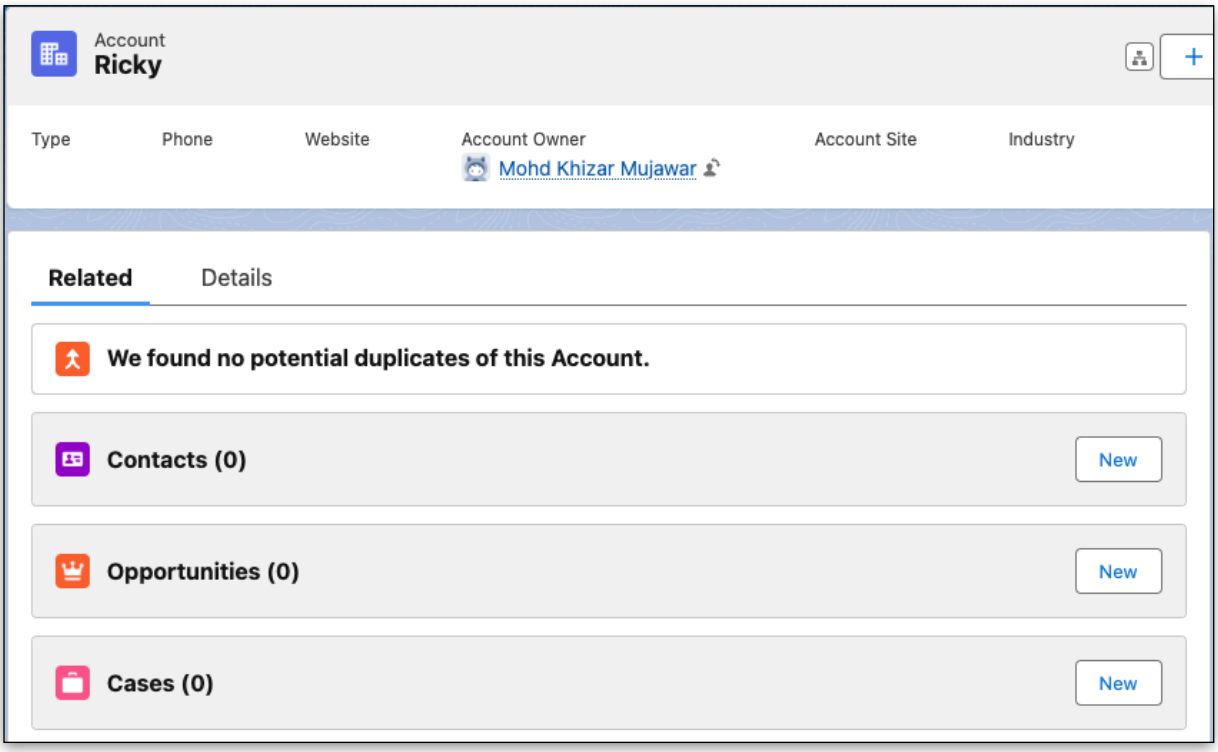


Testing Triggers -5:

4. Now, Delete "Cody" contact and verify contact's last name is removed from account name

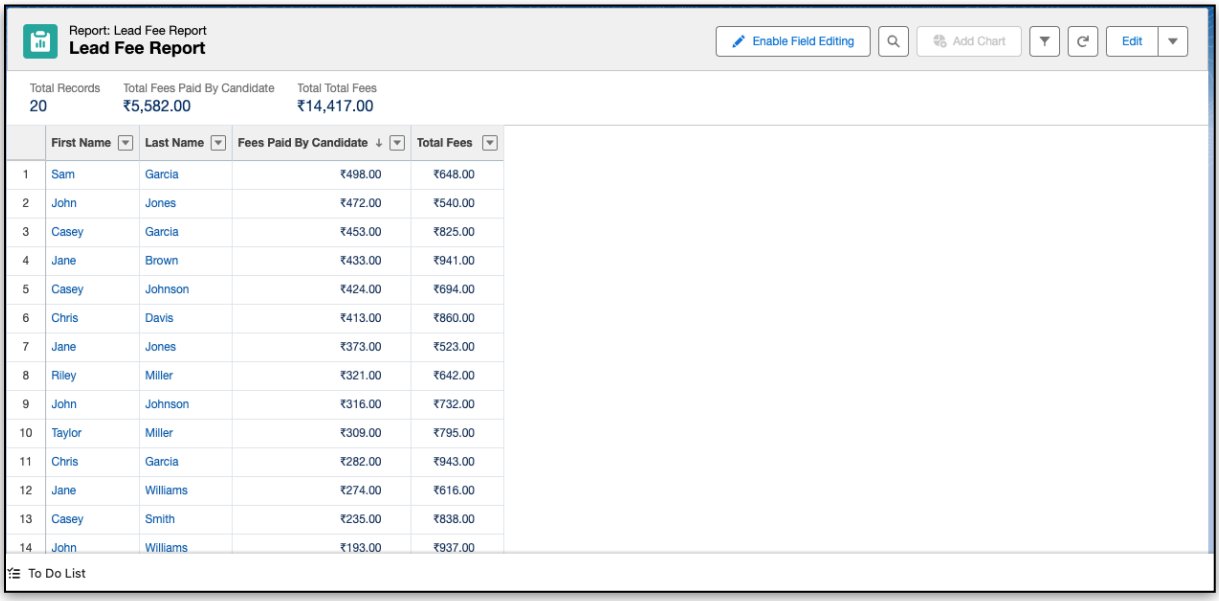


5. LastName of contact is removed from AccountName



REPORTS AND DASHOBOARDS (WEEK-10)

10.1 Create report to calculate total Fees of candidates



Report: Lead Fee Report
Lead Fee Report

Enable Field Editing | Add Chart | Edit

Total Records	Total Fees Paid By Candidate	Total Total Fees
20	₹5,582.00	₹14,417.00

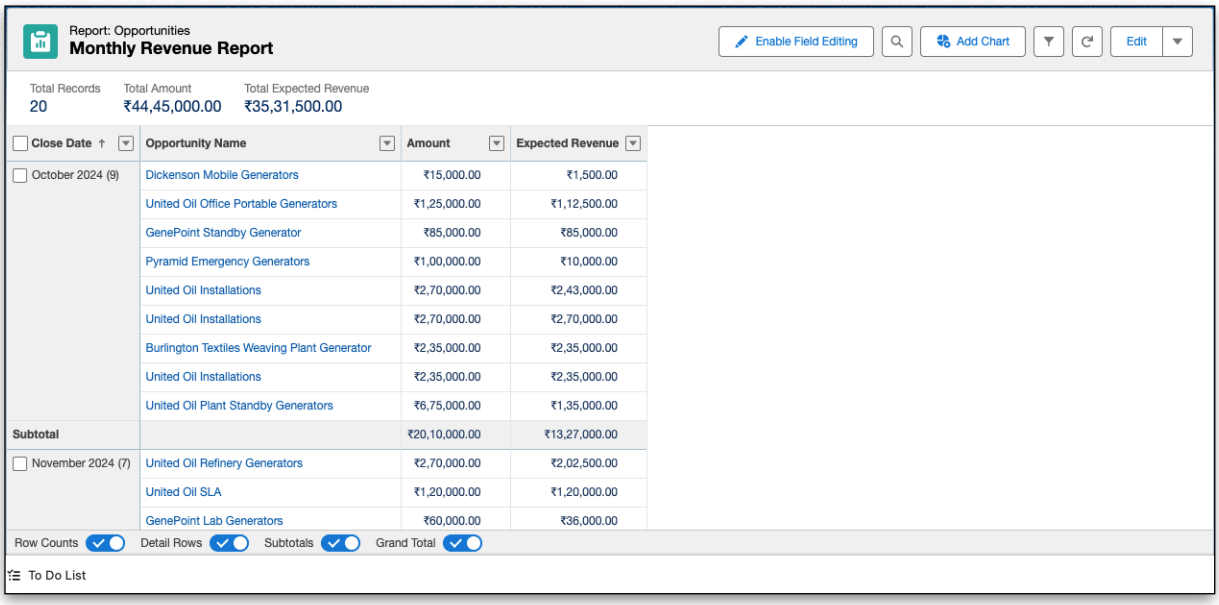
	First Name	Last Name	Fees Paid By Candidate	Total Fees
1	Sam	Garcia	₹498.00	₹648.00
2	John	Jones	₹472.00	₹540.00
3	Casey	Garcia	₹453.00	₹825.00
4	Jane	Brown	₹433.00	₹941.00
5	Casey	Johnson	₹424.00	₹694.00
6	Chris	Davis	₹413.00	₹860.00
7	Jane	Jones	₹373.00	₹523.00
8	Riley	Miller	₹321.00	₹642.00
9	John	Johnson	₹316.00	₹732.00
10	Taylor	Miller	₹309.00	₹795.00
11	Chris	Garcia	₹282.00	₹943.00
12	Jane	Williams	₹274.00	₹616.00
13	Casey	Smith	₹235.00	₹838.00
14	John	Williams	₹193.00	₹937.00

To Do List

Steps:

1. **Navigate to Reports:** In Salesforce, go to the "Reports" tab.
2. **Create a New Report:** Click "New Report" and select "Opportunity" as the report type.
3. **Add Fields:** Drag and drop relevant fields like “First Name," “Last Name," “Fees Paid By Candidate," and “Total fees" into the report.
4. Save and Run the report
5. **Assign Label:** Lead Fee Report

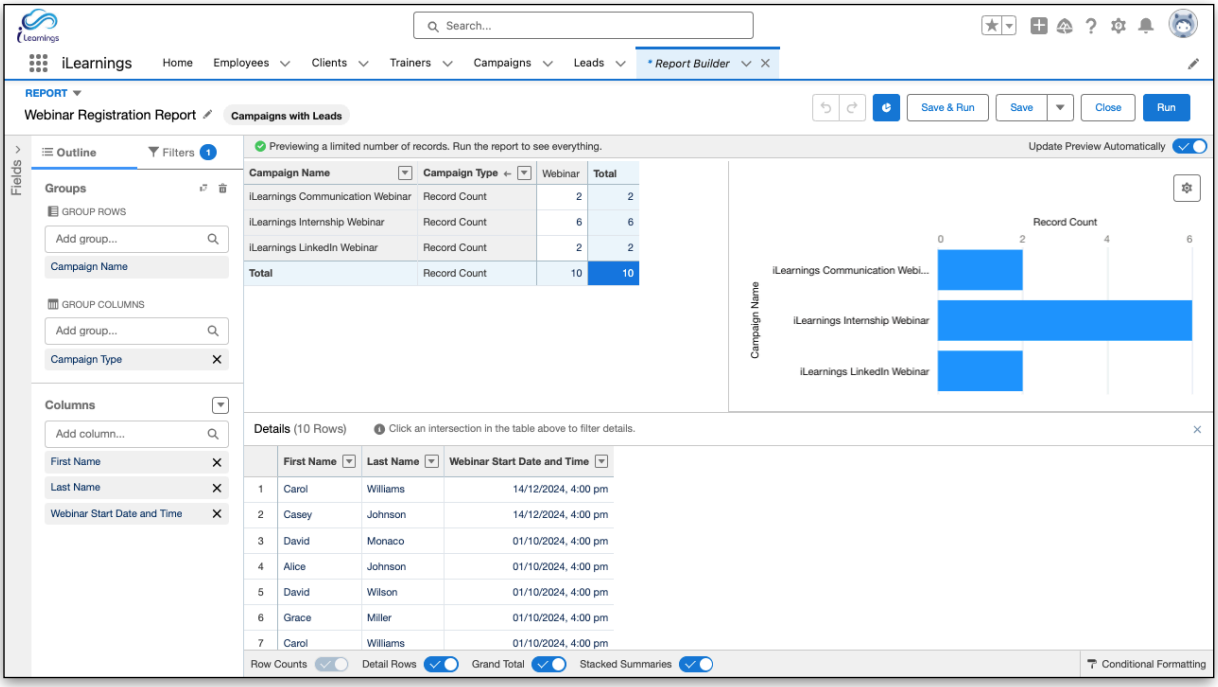
10.2 Create Sales report to calculate monhtly revenue



Steps:

1. **Navigate to Reports:** In Salesforce, go to the "Reports" tab.
2. **Create a New Report:** Click "New Report" and select "Opportunity" as the report type.
3. **Add Fields:** Drag and drop relevant fields like “Close Date," "Opportunity," "Amount," and “Expected Revenue" into the report.
4. Go to Groups Section
   - **Group Rows:**
     - ‘Close Date’ field
5. Save and Run the report.
6. **Assign Label:** Monthly Revenue Report.

10.3 Create Webinar Reports to track how many students registered in each webinar



Steps:

1. **Navigate to Reports:** In Salesforce, go to the "Reports" tab.
2. **Create a New Report:** Click "New Report" and select “Campaign With Leads" as the report type.
3. **Add Fields:** Drag and drop relevant fields like “First Name," “Last Name," and “Webinar Start Date And Time”.
4. Go to Groups Section
   - **Group Rows:**
     - Add ‘Close Date’ field
   - **Group Columns:**
     - Add ‘Campaign Type’ field
5. Save and Run the report.
6. **Assign Label:** Webinar Registration Report.

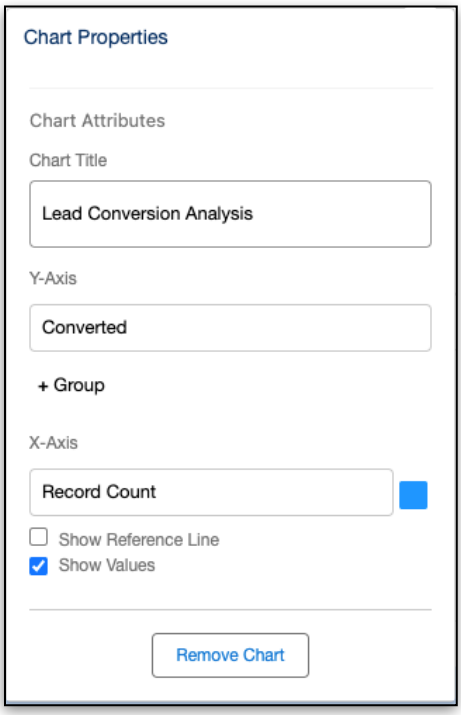
10.4 Create Report on conversion ratio how many leads vs how many converted

Steps:

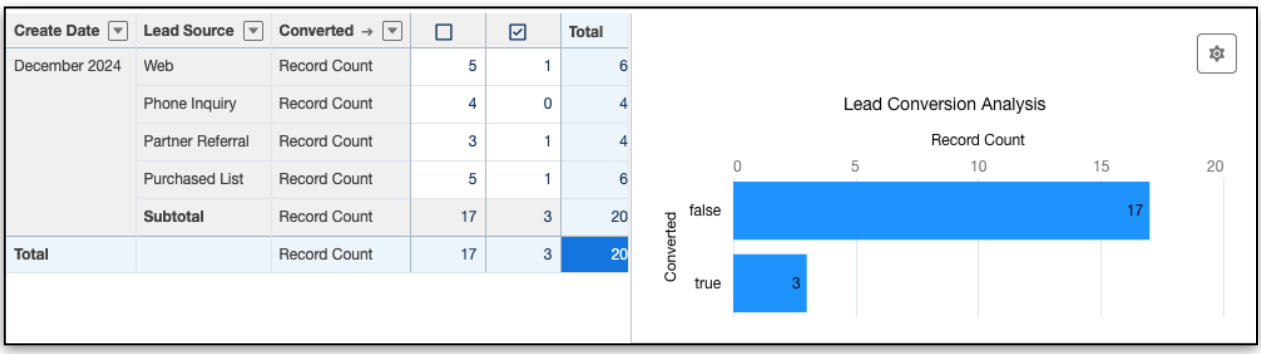
1. **Navigate to Reports:** In Salesforce, go to the "Reports" tab.
2. **Create a New Report:** Click "New Report".
3. **Choose Report Type:** Select "Leads with Converted Lead Information" to track both converted and unconverted leads.
4. **Add Fields:** Drag and drop relevant fields like "Lead Source," "Converted Date," "Owner," and "Status" into the report.
5. **Go to Groups Section**
   - **Group Rows:**
     - 'Close Date' field
       - Click Drop down of 'Create Date' and Group By : Calendar Month
     - Lead Source
   - **Group Column:**
   - Converted
6. **Columns**
   - First Name
   - Last Name
   - Company
   - Lead Status
7. Save and Run the report, **Assign Label:** Lead Conversion Ratio

BAR GRAPH:

Graph Settings



Ratio of Total Leads vs Converted / Unconverted Leads



Create Date	Lead Source	Converted →	☐	☑	Total
December 2024	Web	Record Count	5	1	6
	Phone Inquiry	Record Count	4	0	4
	Partner Referral	Record Count	3	1	4
	Purchased List	Record Count	5	1	6
	Subtotal	Record Count	17	3	20
Total		Record Count	17	3	20

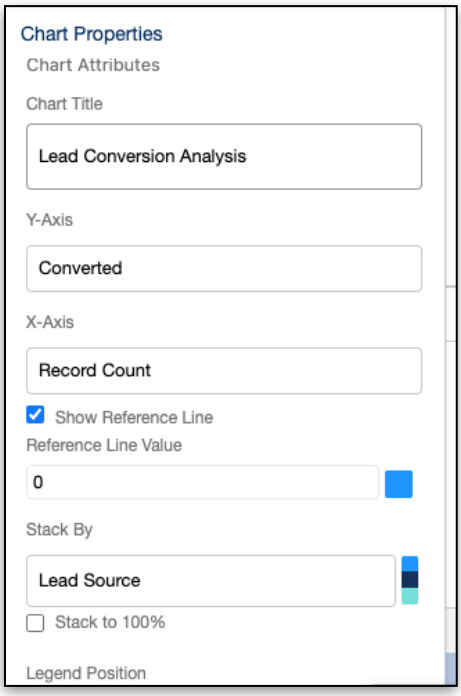
From Graph,

- Ratio of Converted Leads and Total leads = 3 : 20 **[<u>Which is Requirement!</u>]**
- Ratio of Not Converted Leads and Total leads = 17 : 20

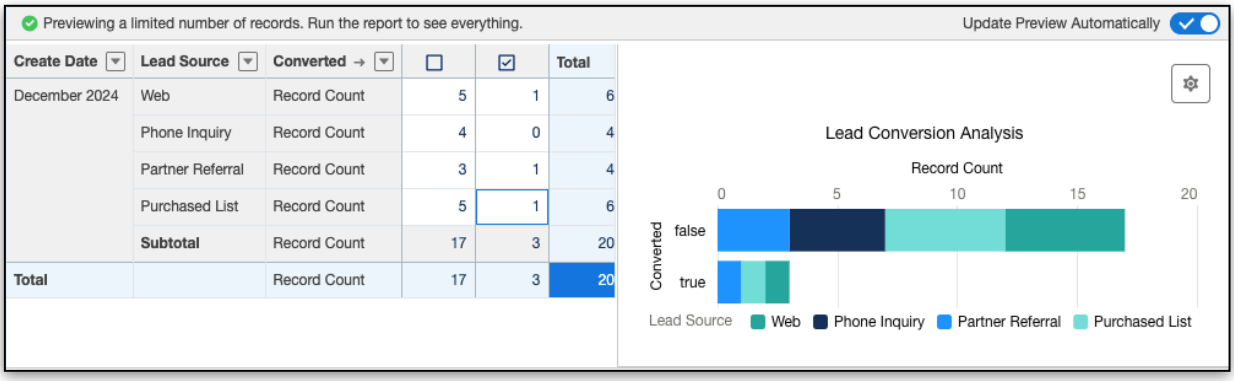
We can also add Lead Source info by using Stacked Bar Graph

STACKED BAR GRAPH:

Graph Settings

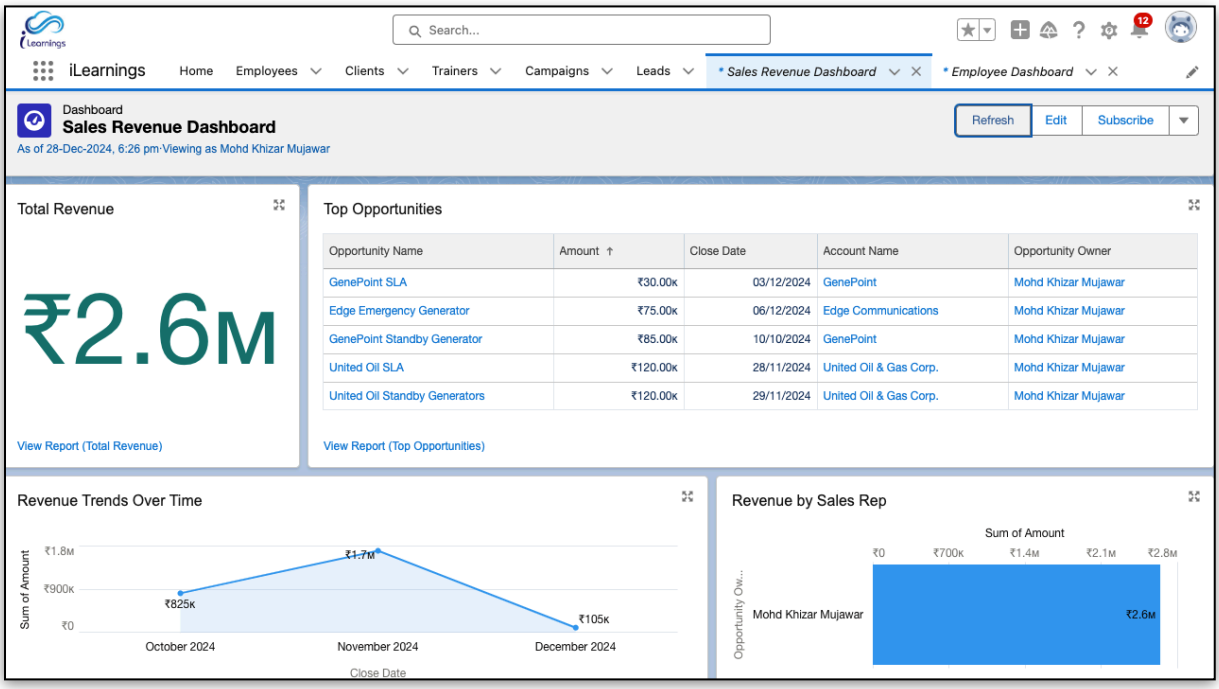


Ratio of Total Leads vs Converted / Unconverted Leads across different lead sources:



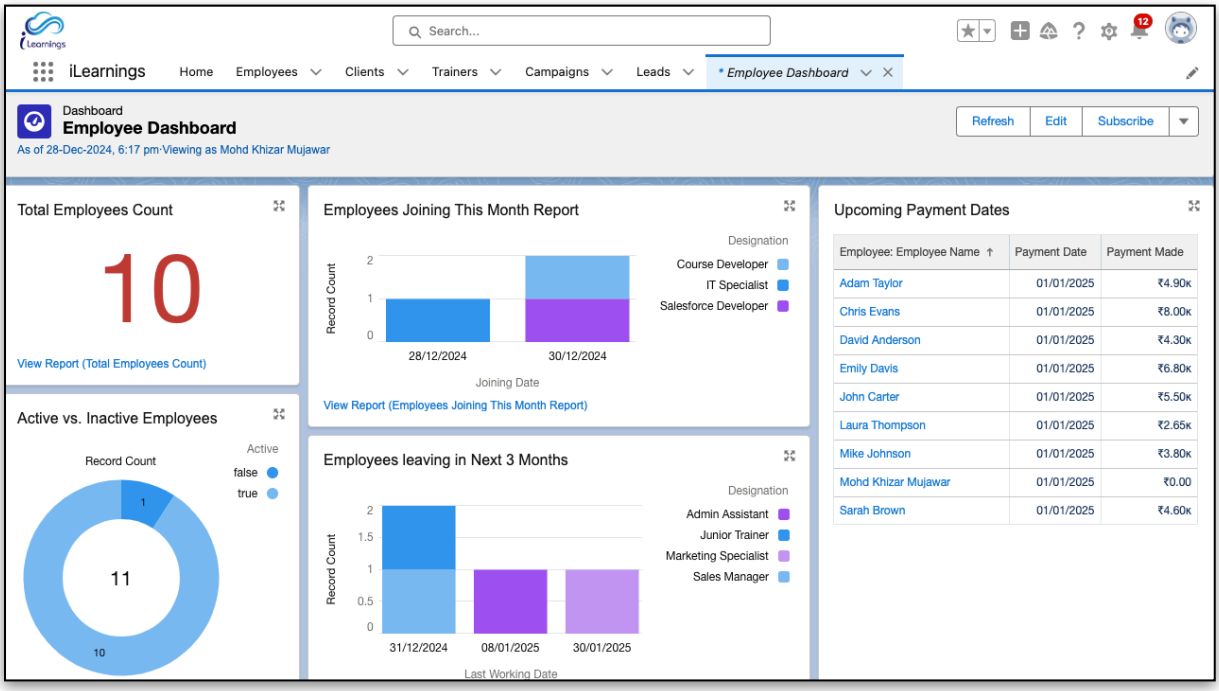
10.5 Create 3 dashboards - Sales, trainers & Employee

Sales Revenue Dashboard



Trainer Dashboard

Employee Dashboard



APEX TRIGGERS

Apex Scenarios

11.1 **Scenario 6: Opportunity Trigger:** Alert users when closing an Opportunity without Line Items.

11.2 **Scenario 7: Opportunity Trigger:** Increment “No of Products Sold” when Opportunity is closed-won.

11.3 **Scenario 8: Opportunity Trigger:** Add all Account Contacts to Opportunity Contact Roles during creation.

11.4 **Scenario 9: Campaign Trigger:** Close Opportunities as “Won” or “Lost” based on Line Items when a Campaign is completed.

11.5 **Scenario 10: Contact Trigger:** Copy Account’s Billing Address to Contact’s Mailing Address when enabled.

Trigger-6: Alert users when closing an Opportunity without Line Items.

Trigger-7: Alert users when closing an Opportunity without Line Items.

Trigger-8: Alert users when closing an Opportunity without Line Items.

OpportunityTrigger.apxt

```
trigger OpportunityTrigger on Opportunity (before update, after insert, after update) {
    OpportunityTriggerHandler handler = new OpportunityTriggerHandler();

    // Trigger - 8 Logic (Add Account Contacts to Opportunity Contact Roles)
    if (Trigger.isAfter && Trigger.isInsert) {
        handler.handleAfterInsert(Trigger.new); // Corrected method name
    }
    // Trigger - 6 Logic (Prevent closing Opportunity without Line Items)
    if (Trigger.isBefore && Trigger.isUpdate) {
        handler.handleBeforeUpdate(Trigger.new, Trigger.oldMap);
    }
    // Trigger - 7 Logic (Increment "No of Products Sold' field)
    if (Trigger.isAfter && Trigger.isUpdate) {
        handler.handleAfterUpdate(Trigger.new, Trigger.oldMap);
    }
}

```

Trigger-8 Logic

```apex
public class OpportunityTriggerHandler {

    // Trigger - 8 Logic
    public void handleAfterInsert(List<Opportunity> newOpps) {
        List<OpportunityContactRole> contactRolesToCreate = new
        List<OpportunityContactRole>();

        for (Opportunity opp : newOpps) {
            if (opp.AccountId != null) {
                // Query all Contacts related to the Account
                List<Contact> accountContacts = [
                    SELECT Id FROM Contact WHERE AccountId = :opp.AccountId
                ];

                // Create OpportunityContactRole records for each Contact
                for (Contact contact : accountContacts) {
                    OpportunityContactRole ocr = new OpportunityContactRole(
                        OpportunityId = opp.Id,
                        ContactId = contact.Id,
                        Role = 'Business User' // Default Role
                    );
                    contactRolesToCreate.add(ocr);
                }
            }
        }

        // Insert the OpportunityContactRole records in bulk
        if (!contactRolesToCreate.isEmpty()) {
            insert contactRolesToCreate;
        }
    }

```

Trigger-6 Logic

```apex
    public void handleBeforeUpdate(List<Opportunity> newOpps, Map<Id, Opportunity> oldOppMap) {
    for (Opportunity opp : newOpps) {
        // Prevent closing without Opportunity Line Items
        if ((opp.StageName == 'Closed Won' || opp.StageName == 'Closed Lost') &&
            (oldOppMap.get(opp.Id).StageName != 'Closed Won' &&
            oldOppMap.get(opp.Id).StageName != 'Closed Lost')) {

            // Query for Opportunity Line Items associated with the Opportunity
            List<OpportunityLineItem> lineItems = [SELECT Id
            FROM OpportunityLineItem
            WHERE OpportunityId = :opp.Id LIMIT 1];

            // Add error if no Opportunity Line Items exist
            if (lineItems.isEmpty()) {
                opp.addError('You cannot close an Opportunity without adding Opportunity Line Items.');
            }
        }
    }
}

```

Trigger-7

INITIAL STEP:

Create ‘No of Products Sold’ custom field

- Field Type: Number
- Field Label: No of Products Sold

Trigger-7 Logic

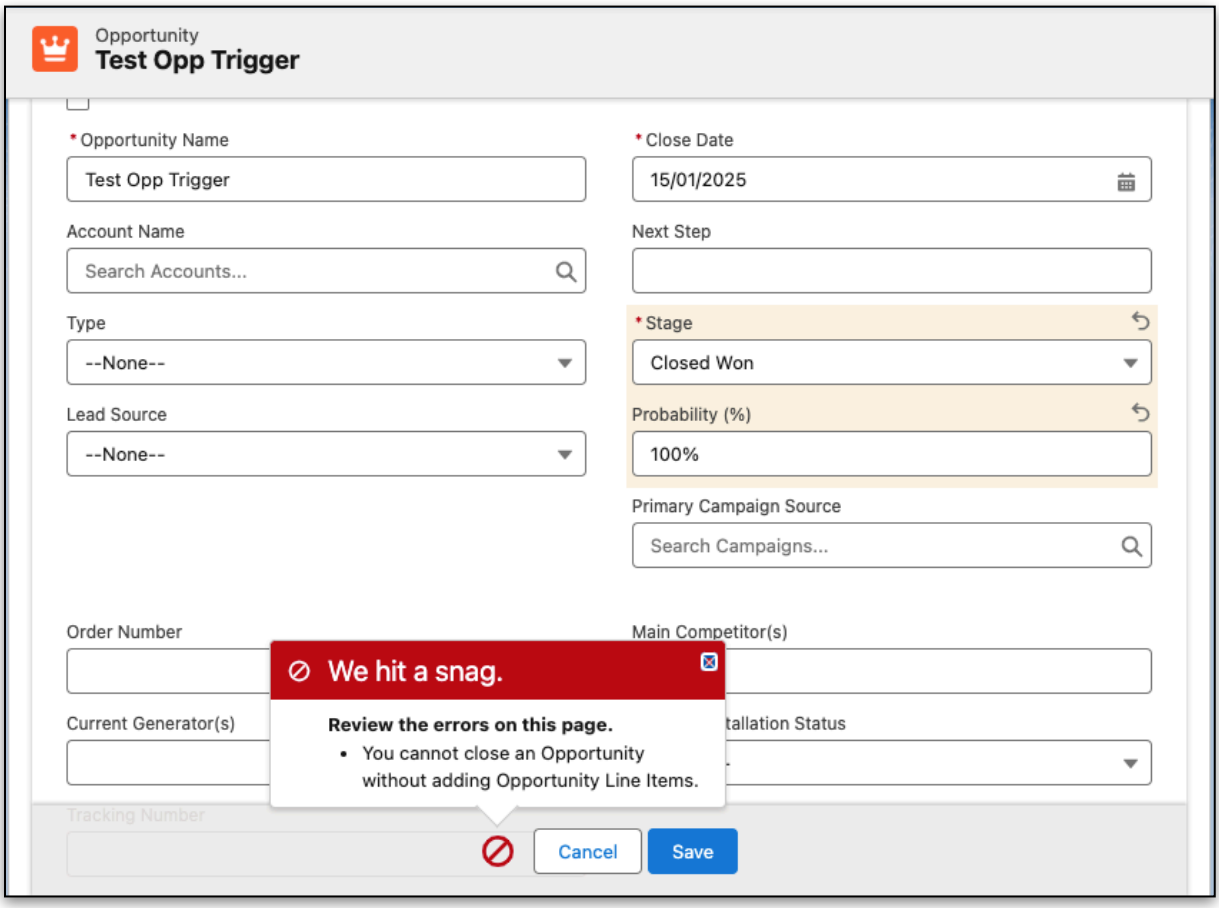
```
    public void handleAfterUpdate(List<Opportunity> newOpps, Map<Id, Opportunity> oldOppMap) {
    List<Product2> productsToUpdate = new List<Product2>();

    for (Opportunity opp : newOpps) {

        // Check if Opportunity is closed as "Closed Won"
        if (opp.StageName == 'Closed Won' && oldOppMap.get(opp.Id).StageName != 'Closed Won')   {
            // Query Opportunity Line Items and associated Products
            List<OpportunityLineItem> lineItems = [
                SELECT Id, Quantity, PricebookEntry.Product2Id
                FROM OpportunityLineItem
                WHERE OpportunityId = :opp.Id
            ];

            for (OpportunityLineItem oli : lineItems) {
                if (oli.PricebookEntry.Product2Id != null) {

                    // Update "No of Products Sold" field
                    Product2 product = [SELECT Id, No_of_Products_Sold__c
                    FROM Product2
                    WHERE Id = :oli.PricebookEntry.Product2Id LIMIT 1];

                    product.No_of_Products_Sold__c =
                    (product.No_of_Products_Sold__c == null ? 0 : product.No_of_Products_Sold__c)
                    + Integer.valueOf(oli.Quantity);

                    productsToUpdate.add(product);
                }
            }
        }
    }

    // Update Product records in bulk
    if (!productsToUpdate.isEmpty()) {
        update productsToUpdate;
    }
  }
}
```

Testing Trigger - 6

1. **Create a New Opportunity**
   - Fill in the Stage **except "Closed Won" or "Closed Lost"** for now.
   - Save the Opportunity.

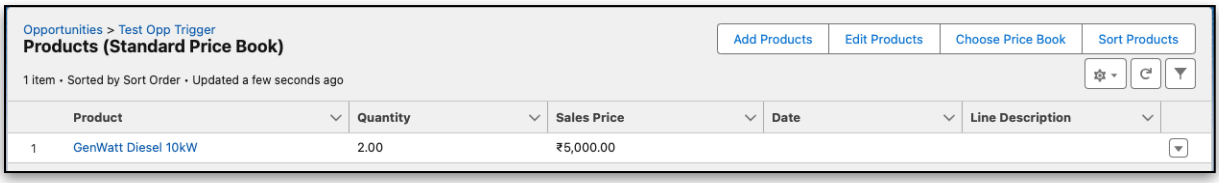
2. **Attempt to Close the Opportunity Without Adding Line Items**
   - Open the newly created Opportunity record.
   - Edit the record and change the **Stage** to either "Closed Won" or "Closed Lost"
   - Click **Save**.

3. **Verify Validation Error**
   - The trigger will fire, and you should see an error message like:
     - **"You cannot close an Opportunity without adding Opportunity Line Items."**
   - The Opportunity record will not be saved with the "Closed Won" or "Closed Lost" Stage.



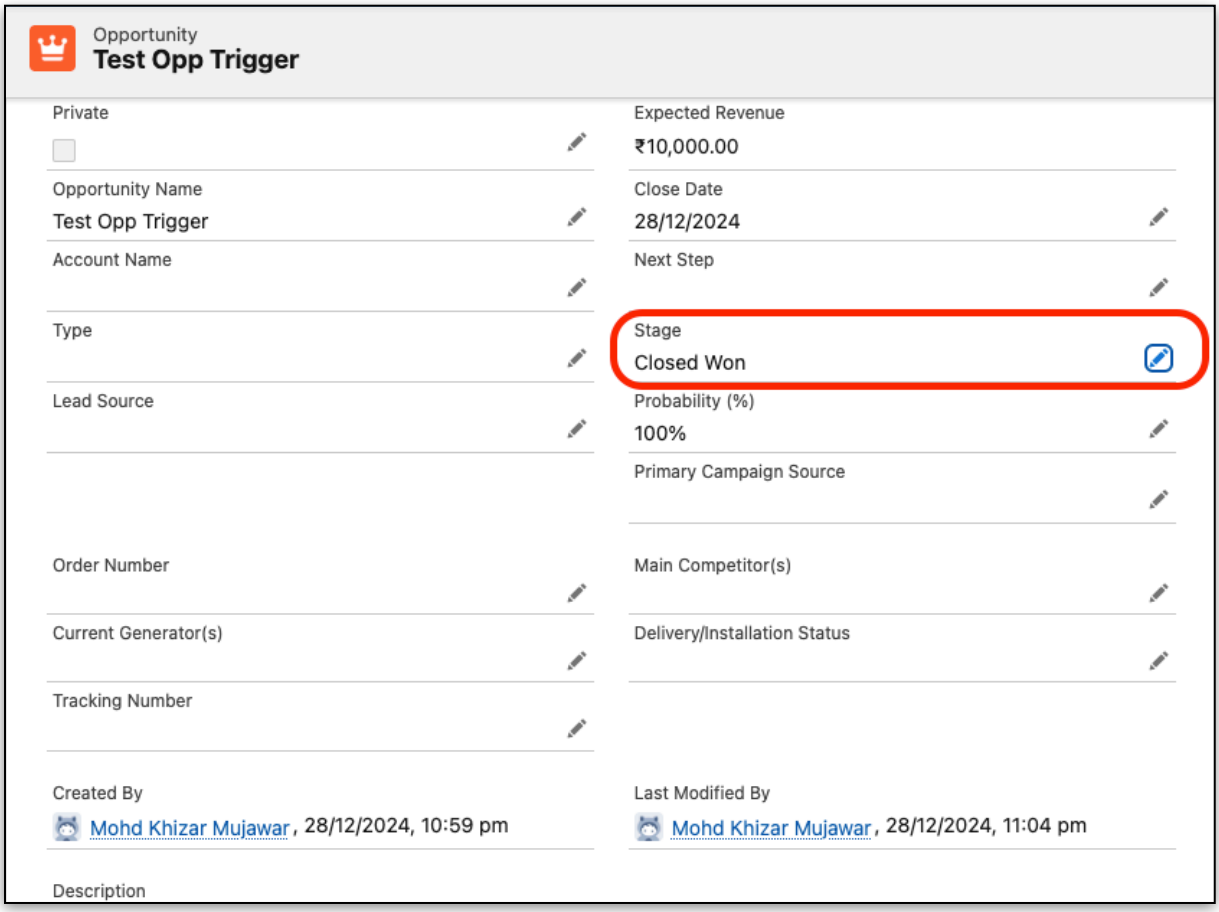
4. **Add Opportunity Line Items (to Validate the Positive Scenario)**

- Open the Opportunity record.
- Go to the **Opportunity Products** related list.
- Click **Add Products** and add at least one product with a quantity and price.
- Save the Opportunity Line Items.



5. Close the Opportunity Again

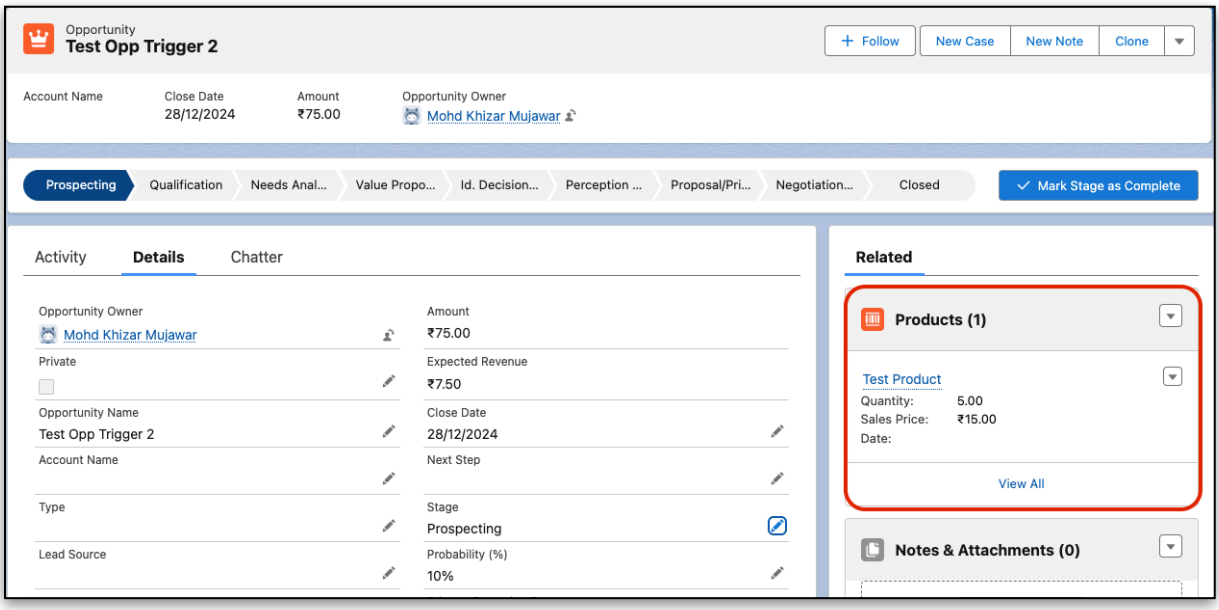
- Edit the Opportunity record and change the **Stage** to "Closed Won" or "Closed Lost."
- Click **Save.**



Testing Trigger - 7

1. **Create a New Opportunity**
   - Set the **Stage** to anything other than "Closed Won."

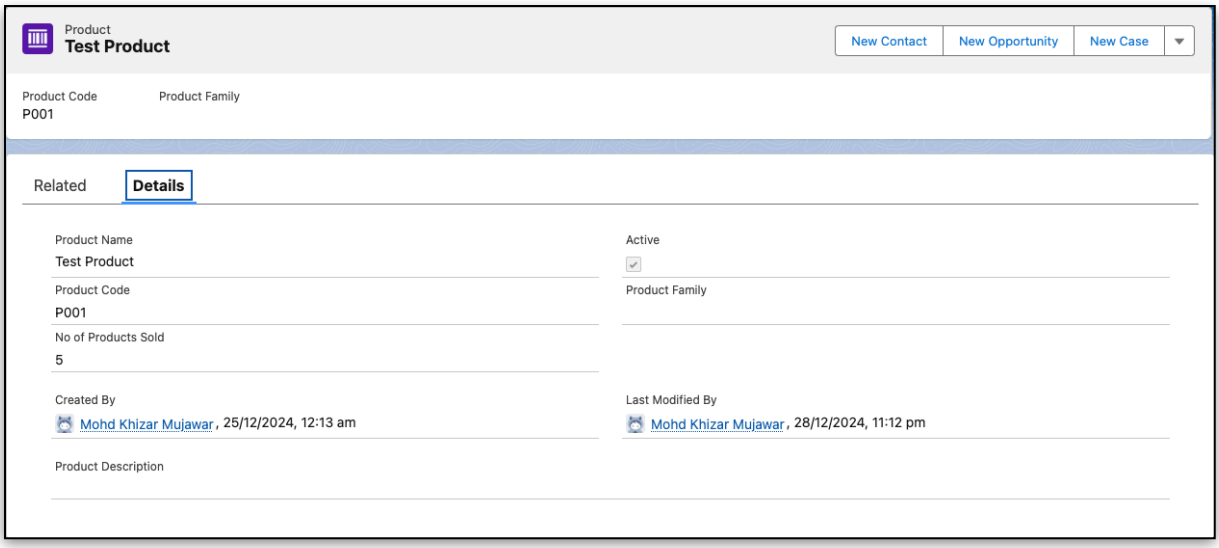
2. **Add Opportunity Line Items**
   - Open the newly created Opportunity record.
   - Go to the **Opportunity Products** related list (this is where Opportunity Line Items are added).
   - Click **Add Products** and:
     - Select one or more products.
     - **Set Quantity** : 5
   - Save the Opportunity Line Items.



3. **Close the Opportunity as "Closed Won"**
   - Edit the Opportunity record and change the **Stage** to **"Closed Won"**.
   - Save the Opportunity.

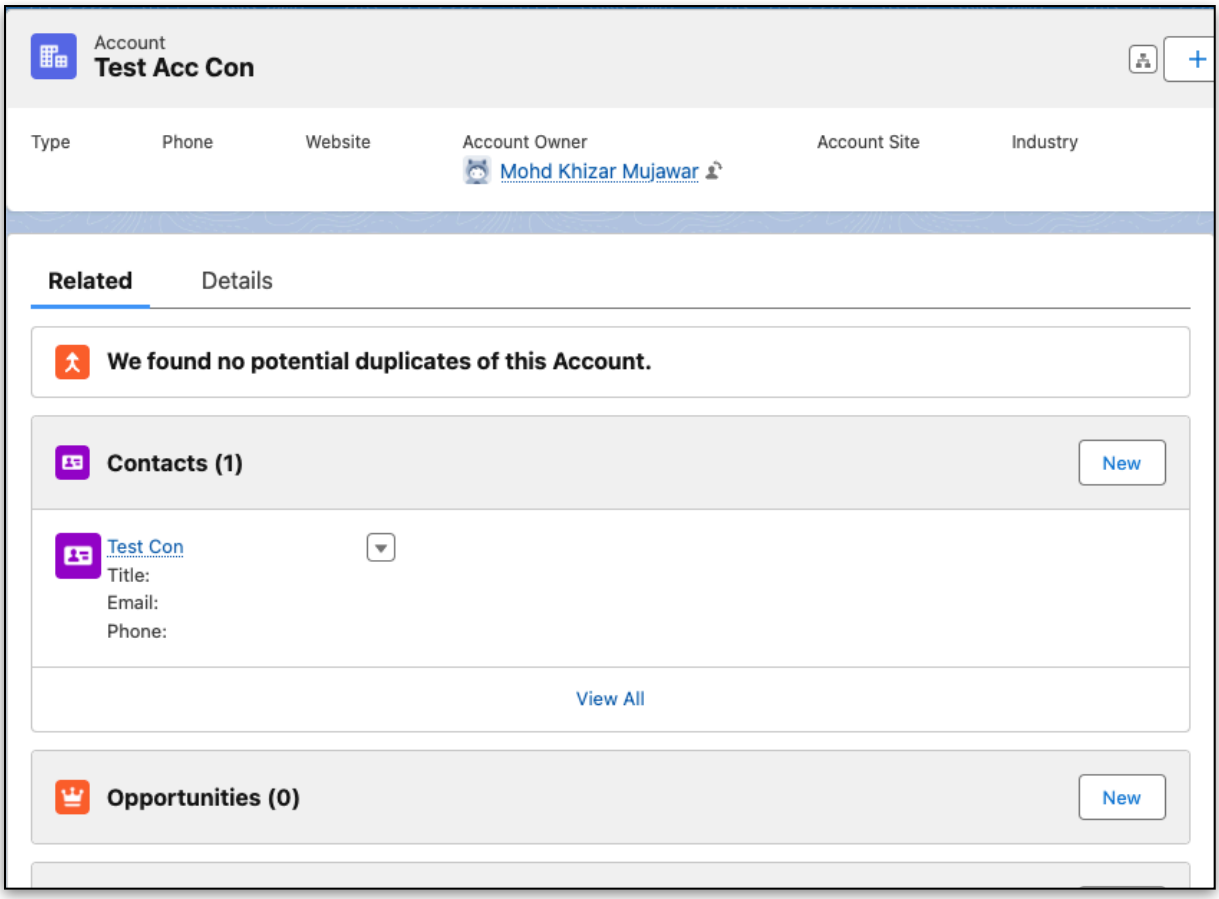
4. **Verify the "No of Products Sold" Field on Products**
   - Navigate to the **Products** related to the Opportunity Line Items.
     - You can find the Product records by:
       - Opening each **Opportunity Line Item** (Product lookup field).
       - Navigating to the Product detail page.
   - Check the **"No of Products Sold"** field on each Product.

- The field should have been incremented by the **Quantity** of the corresponding Opportunity Line Item(s).



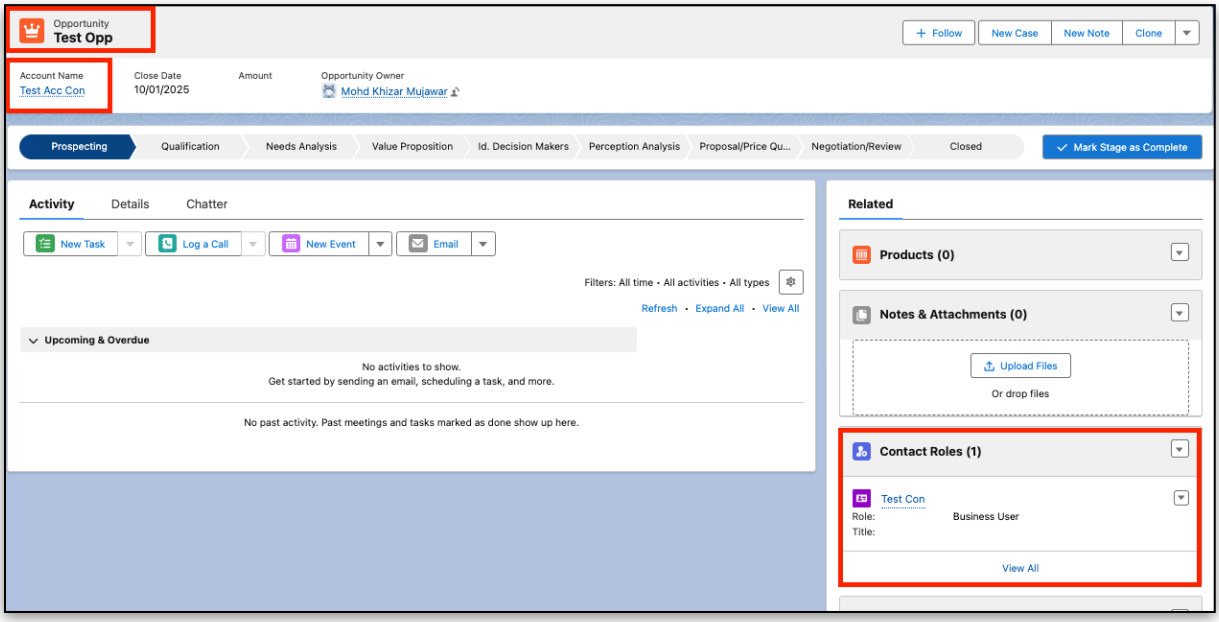
Testing Trigger - 8

1. **Create a test Account with Contacts**
   - Go to **Accounts** and create a new Account (e.g., "Test Acc").
   - In the "Related" tab, add one or more Contacts by clicking **New Contact**, filling in required details, and saving.



2. **Create a New Opportunity**
   - Go to **Opportunities** and create a new Opportunity.
   - Link it to the created Account (e.g., "Test Account"), fill in required fields, and save.

3. **Verify Opportunity Contact Roles**
   - Open the saved Opportunity and navigate to its "Related" tab.
   - Check the **Opportunity Contact Roles** list to ensure all Account Contacts are added with the default Role (e.g., "Business User").


Opportunity
Test Opp
+ Follow
New Case
New Note
Clone
Account Name
Test Acc Con
Close Date
10/01/2025
Amount
Opportunity Owner
Mohd Khizar Mujawar
Prospecting
Qualification
Needs Analysis
Value Proposition
Id. Decision Makers
Perception Analysis
Proposal/Price Qu...
Negotiation/Review
Closed
Mark Stage as Complete
Activity
Details
Chatter
New Task
Log a Call
New Event
Email
Filters: All time • All activities • All types
Refresh • Expand All • View All
Upcoming & Overdue
No activities to show.
Get started by sending an email, scheduling a task, and more.
No past activity. Past meetings and tasks marked as done show up here.
Related
Products (0)
Notes & Attachments (0)
Upload Files
Or drop files
Contact Roles (1)
Test Con
Role:
Business User
Title:
View All

Trigger-9: Close Opportunities as “Won” or “Lost” based on Line Items when a Campaign is completed.

CampaignTrigger.apxt

```
trigger CampaignTrigger on Campaign (after update) {
    CampaignTriggerHandler handler = new CampaignTriggerHandler();

    if (Trigger.isAfter && Trigger.isUpdate) {
        handler.handleAfterUpdate(Trigger.new);
    }
}
```

CampaignTriggerHandler.apxc

```
public class CampaignTriggerHandler {
    public void handleAfterUpdate(List<Campaign> updatedCampaigns) {

        List<Opportunity> opportunitiesToUpdate = new List<Opportunity>();
        Map<Id, List<String>> closureDetails = new Map<Id, List<String>>();

        for (Campaign camp : updatedCampaigns) {
            if (camp.Status == 'Completed') {
                // Query all Opportunities related to the Campaign
                List<Opportunity> associatedOpportunities = [
                    SELECT Id, Name, StageName,
                           (SELECT Id FROM OpportunityLineItems)
                    FROM Opportunity
                    WHERE CampaignId = :camp.Id
                ];

                for (Opportunity opp : associatedOpportunities) {
                    if (!opp.OpportunityLineItems.isEmpty()) {
                        opp.StageName = 'Closed Won';
                        if (!closureDetails.containsKey(camp.Id)) closureDetails.put(camp.Id, new List<String>());
                        closureDetails.get(camp.Id).add('Opportunity "' + opp.Name + '" closed as Won.');
                    } else {
                        opp.StageName = 'Closed Lost';
                        if (!closureDetails.containsKey(camp.Id)) closureDetails.put(camp.Id, new List<String>());
                        closureDetails.get(camp.Id).add('Opportunity "' + opp.Name + '" closed as Lost.');
                    }
                    opportunitiesToUpdate.add(opp);
                }
            }
        }

```

```
1
2           // Update all Opportunities in bulk
3           if (!opportunitiesToUpdate.isEmpty()) {
4               update opportunitiesToUpdate;
5           }
6
7           // Log the summary of closure details (could send email instead)
8           for (Id campId : closureDetails.keySet()) {
9               System.debug('Campaign Id: ' + campId);
10              for (String detail : closureDetails.get(campId)) {
11                  System.debug(detail);
12              }
13          }
14      }
15  }
```

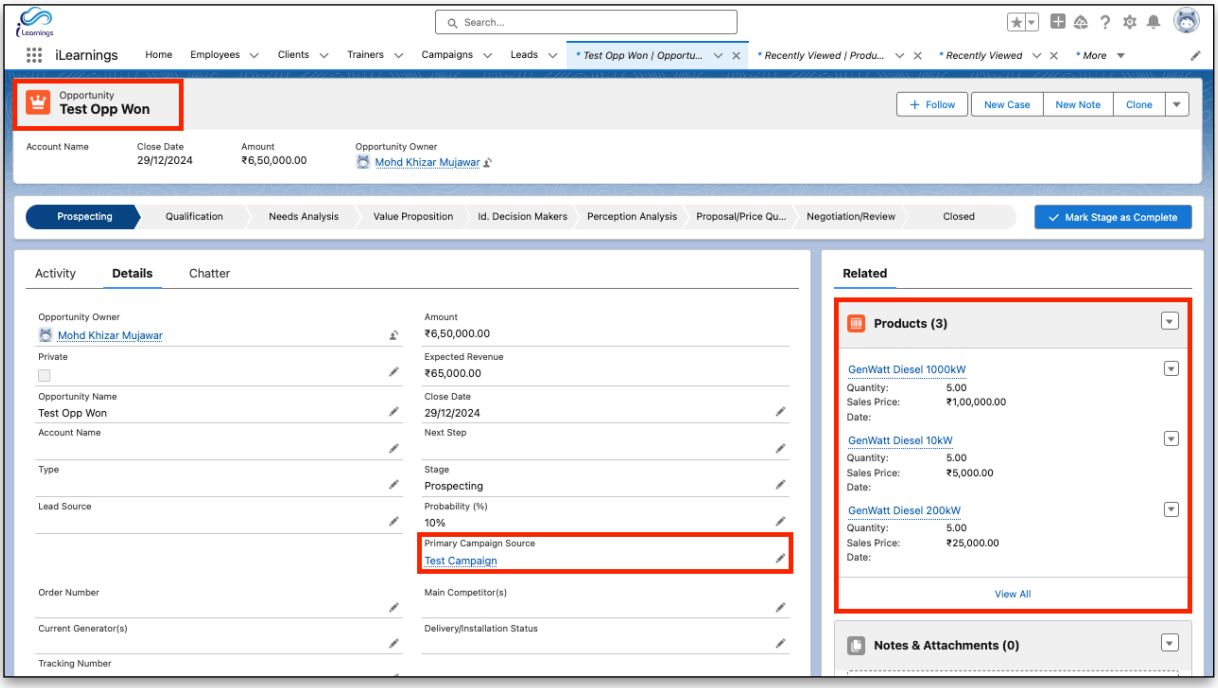
Testing Trigger - 9

1. Create a Campaign

- Create a test campaign, Keep **Status**: In Progress (initially)

2. Create Opportunities Associated with the Campaign

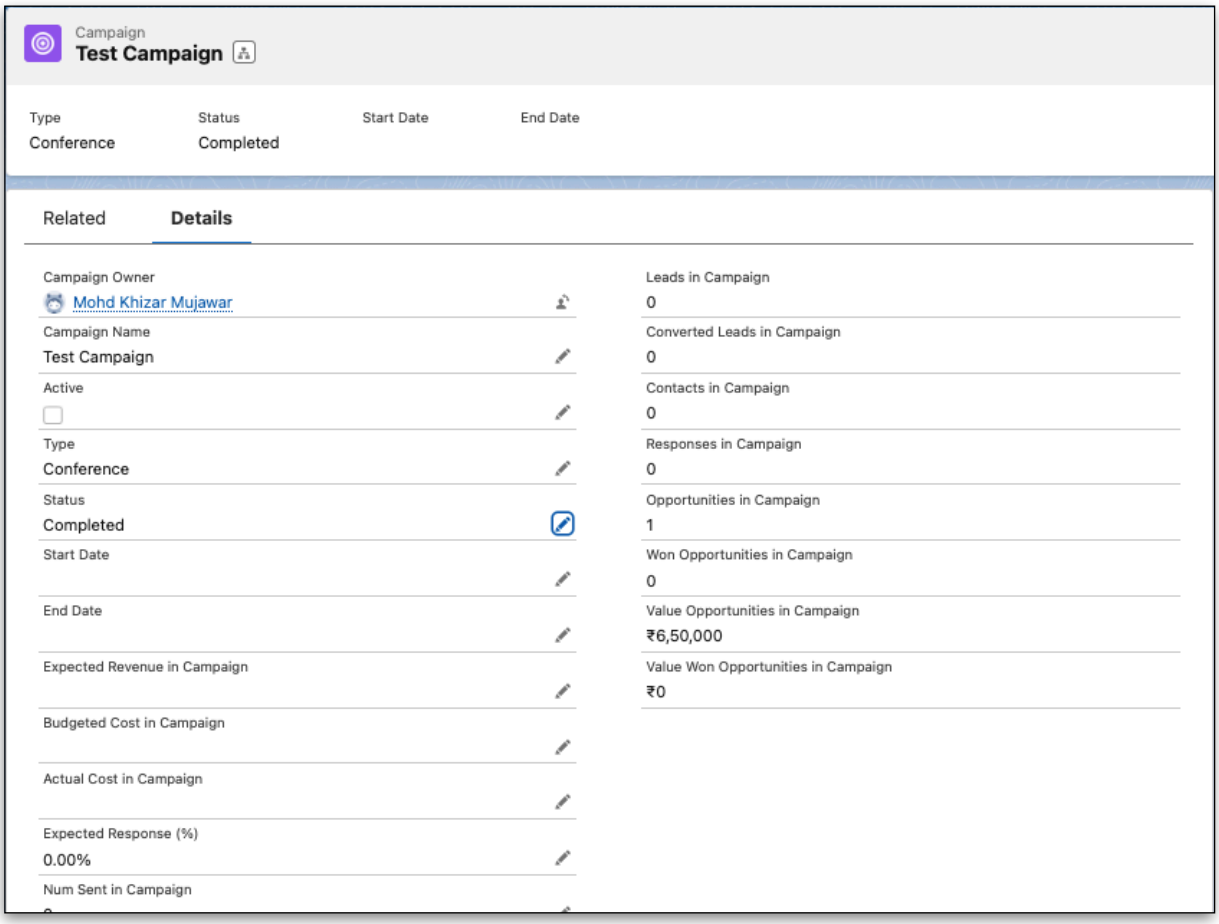
- Create **New Opportunity**.
- Fill in the required fields:
  - **Opportunity Name**: Test Opportunity Won (or Lost)
  - **Stage Name**: Prospecting (initially)
  - **Campaign Source**: Select the Campaign you created in Step 1.
- (Optional) Add **Opportunity Line Items** for the Opportunity:
  - Edit the Opportunity and click **Add Products** (this determines whether it will be closed as "Won" or "Lost" in the trigger logic).



- Repeat these steps to create multiple Opportunities (both with and without Opportunity Line Items) linked to the same Campaign.

3. Update the Campaign to Trigger the Logic

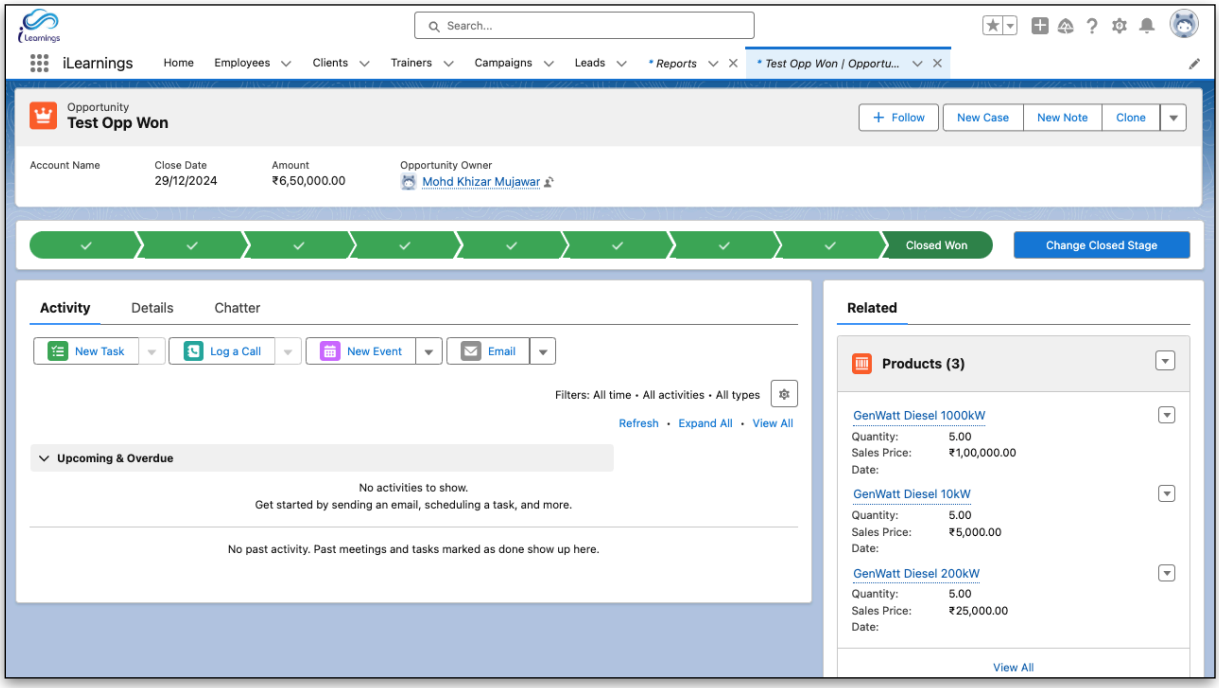
- Navigate back to the Campaign you created in Step-1
- Click **Edit** and change the **Status** to Completed.
- Click **Save**.



4. Verify the Results

Once the Campaign status is updated to Completed, the trigger will execute. You can verify the results as follows:

- **Opportunities Updated:**
  - Go to the related Opportunities for the Campaign (navigate to the Campaign, then view the related list of Opportunities).
- **Verify that:**
  - Opportunities with **Opportunity Line Items** are updated to StageName = Closed Won.
  - Opportunities **without Opportunity Line Items** are updated to StageName = Closed Lost.


Search...
iLearnings
Home
Employees
Clients
Trainers
Campaigns
Leads
* Reports
* Test Opp Won | Opportu...
Opportunity
Test Opp Won
+ Follow
New Case
New Note
Clone
Account Name
Close Date
29/12/2024
Amount
₹6,50,000.00
Opportunity Owner
Mohd Khizar Mujawar
Closed Won
Change Closed Stage
Activity
Details
Chatter
New Task
Log a Call
New Event
Email
Filters: All time • All activities • All types
Refresh • Expand All • View All
Upcoming & Overdue
No activities to show.
Get started by sending an email, scheduling a task, and more.
No past activity. Past meetings and tasks marked as done show up here.
Related
Products (3)
GenWatt Diesel 1000kW
Quantity: 5.00
Sales Price: ₹1,00,000.00
Date:
GenWatt Diesel 10kW
Quantity: 5.00
Sales Price: ₹5,000.00
Date:
GenWatt Diesel 200kW
Quantity: 5.00
Sales Price: ₹25,000.00
Date:
View All

Trigger-10: Copy Account's Billing Address to Contact's Mailing Address when enabled.

INITIAL STEP:
Create 'Copy Billing Address from Account' custom field on Contact Object

- Field Type: CheckBox
- Default value: Uncgked

ContactTrigger.apxt

```
trigger ContactAllEventsTrigger on Contact (before insert, after insert, after delete) {
    switch on Trigger.operationType {
        when BEFORE_INSERT, BEFORE_UPDATE {
            // Trigger-10: Copy Account's Billing Addresses to Contact's Mailing Addresses
            ContactTriggerHandler.beforeInsertUpdateAction(Trigger.new, Trigger.oldMap);
        }
        when AFTER_INSERT {
            // Trigger-4: Update Account Name with Contact's Last Name
            ContactTriggerHandler.afterInsertAction(Trigger.new);
        }
        when AFTER_DELETE {
            // Trigger-5: Remove Contact's Last Name from Account Name
            ContactTriggerHandler.afterDeleteAction(Trigger.old);
        }
        when else {
        }
    }
}
```

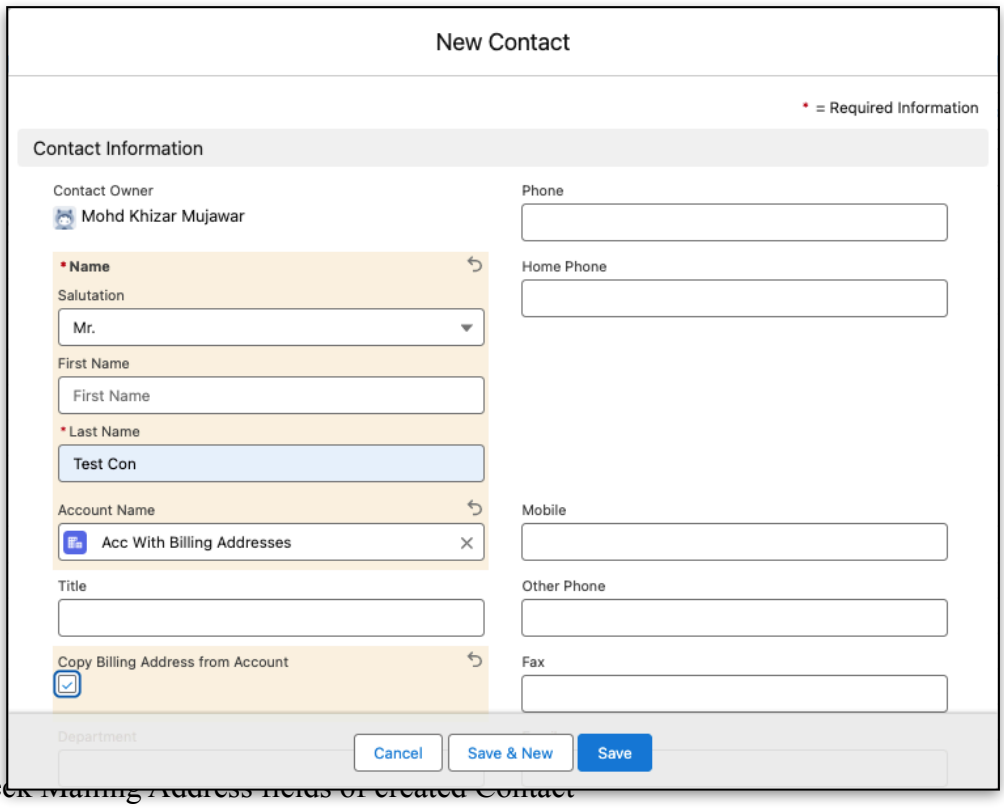
ContactTrigger.apxc

```
public class ContactTriggerHandler {
    // Trigger-10 Logic
    public static void beforeInsertUpdateAction(List<Contact> newContacts, Map<Id, Contact>
oldContactsMap) {
        // Set of Account IDs to query
        Set<Id> accountIds = new Set<Id>();
```

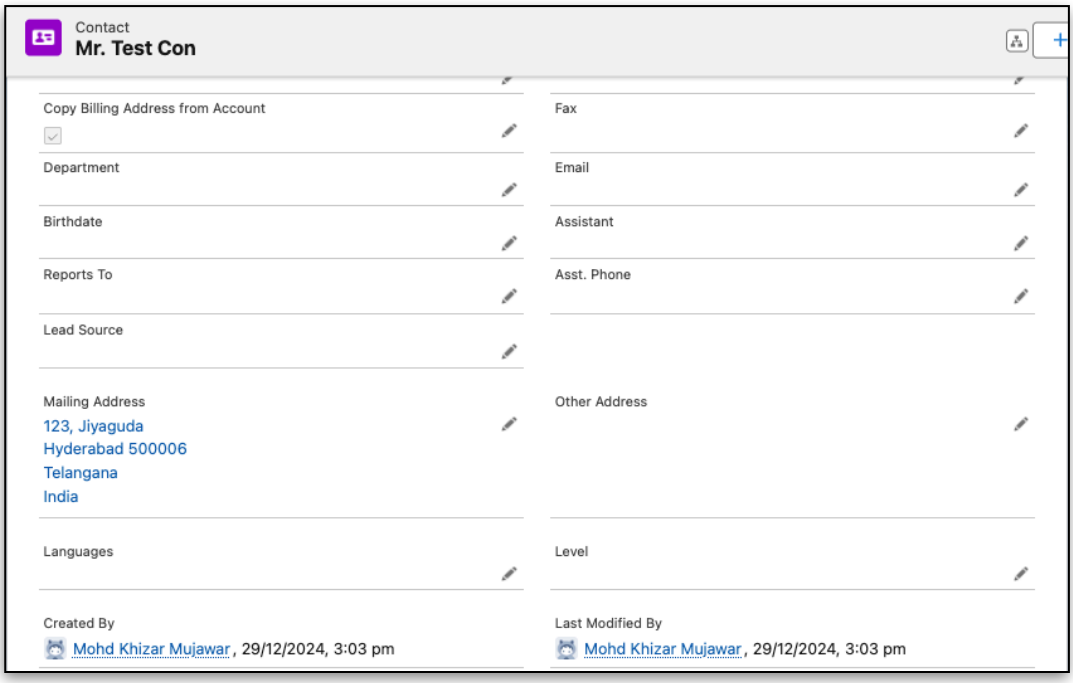
```
    // Gather Account IDs where the checkbox is true
    for (Contact con : newContacts) {
        if (con.Copy_Billing_Address_from_Account__c == true && con.AccountId != null) {
            accountIds.add(con.AccountId);
        }
    }

    // Fetch Account Billing Addresses
    Map<Id, Account> accountMap = new Map<Id, Account>([SELECT Id, BillingStreet,
BillingCity, BillingState, BillingPostalCode, BillingCountry
                                                        FROM Account
                                                        WHERE Id IN :accountIds]);

    // Update the Contact's Mailing Address with the Account's Billing Address
    for (Contact con : newContacts) {
        if (con.Copy_Billing_Address_from_Account__c == true && con.AccountId != null) {
            Account acc = accountMap.get(con.AccountId);
            if (acc != null) {
                con.MailingStreet = acc.BillingStreet;
                con.MailingCity = acc.BillingCity;
                con.MailingState = acc.BillingState;
                con.MailingPostalCode = acc.BillingPostalCode;
                con.MailingCountry = acc.BillingCountry;
            }
        }
    }
  }
  // Trigger-4 Logic …..
  // Trigger-5 Logic …..
 }
```

Testing Trigger - 10

1. Create an Account with a Billing Address
2. Create a New contact,
   - Associate it with Account (Created in Step-1).
   - Mark the ‘'Copy Billing Address from Account” check box.



3. Check Mailing Address fields of created Contact



CAMPAIGN AUTOMATION

Objective

Develop and automate marketing campaigns within Salesforce to effectively promote our products. Monitor campaign performance and analyze results to refine and improve future marketing strategies.

Steps

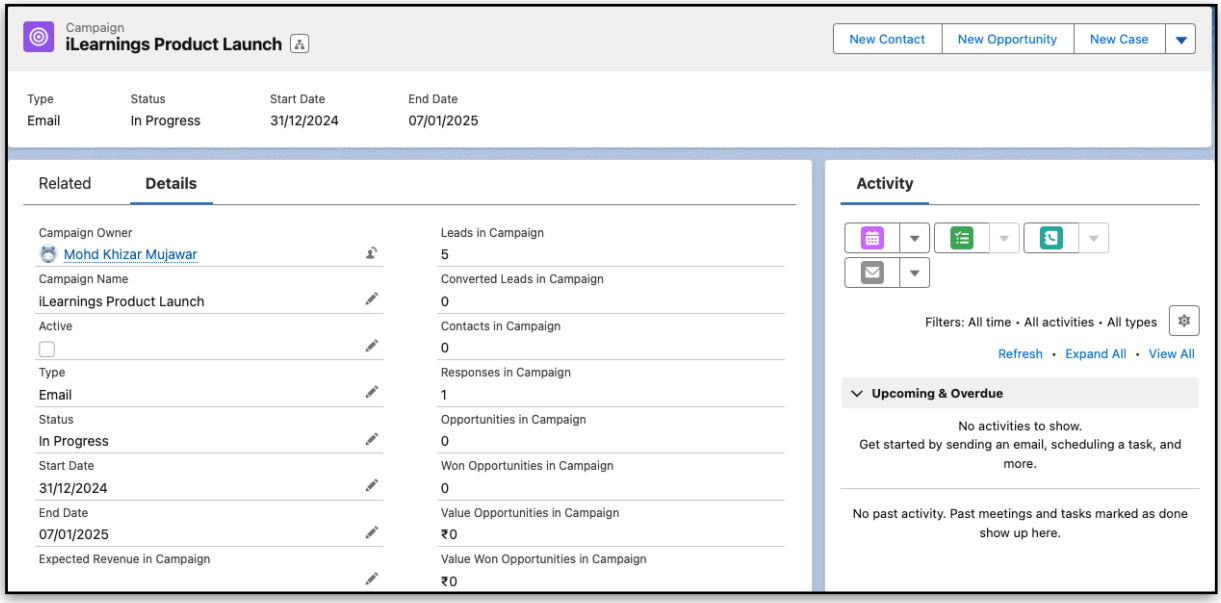
Step 1: Create the Campaign

1. Navigate to Campaigns:

- Go to Salesforce.
- Open the Campaign object (via the App Launcher).

2. Create a New Campaign:

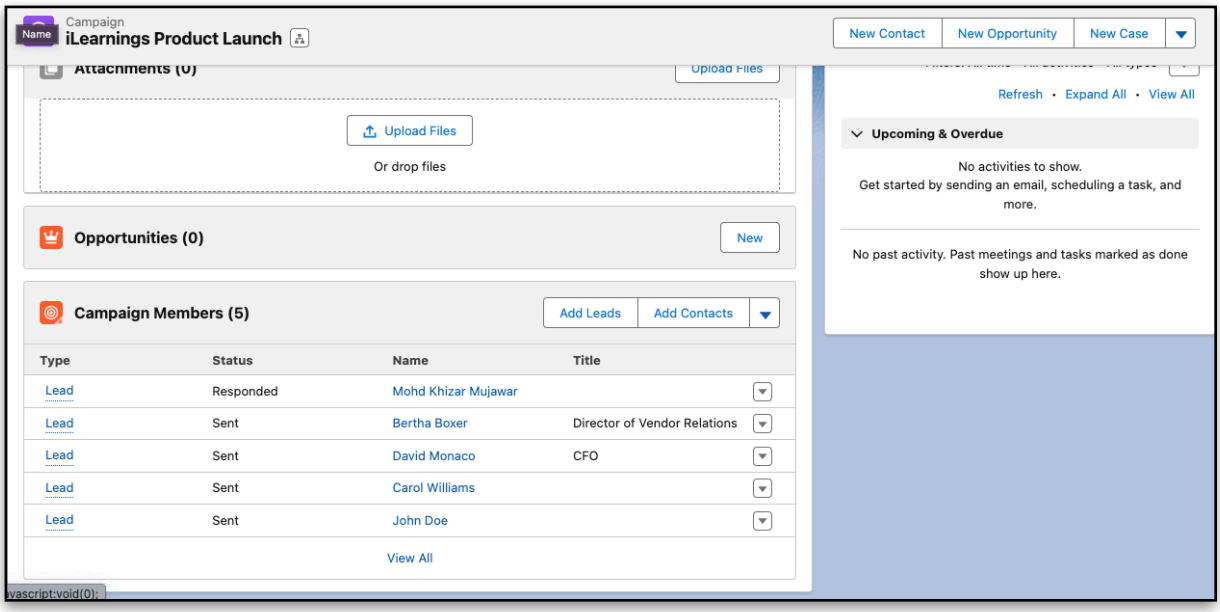
- Click **New**.
- Fill in the required fields:
  - **Campaign Name**: iLearnings Product Launch
  - **Type**: Email
  - **Status**: In Progress
  - **Start Date**: 31/12/2024
  - **End Date**: 07/01/2024
  - **Description**: Promoting our new product line.
- Click **Save**.



3. Add Campaign Members:

- Open the Campaign record.
- Click **Add Leads**:

- Select the target audience for the campaign (Leads).
- Confirm that the list is accurate.



Step 2: Prepare the Email Template

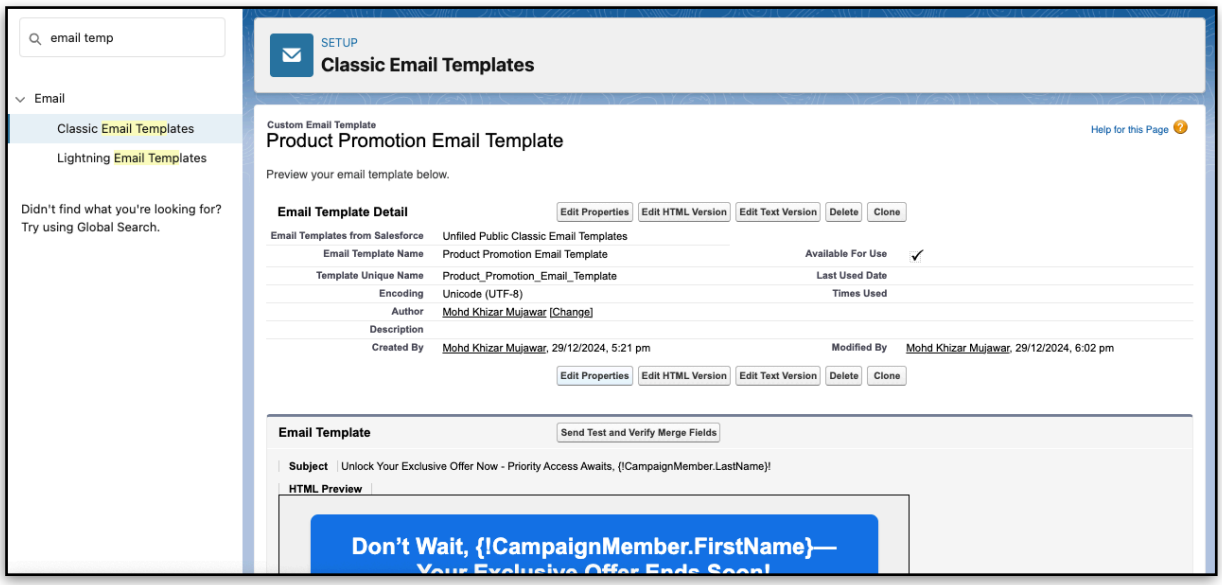
You need an email template to send to your Campaign Members.

1. Navigate to Classic Email Templates:

- Go to **Setup** → Search for **Classic Email Templates**.
- Click **New Template**.
- Choose **HTML with Letterhead** or **Custom**.
- Click **Next**.

2. Create the Email Template:

- **Template Name**: "Product Promotion Email Template”
- Mark Avaialable for Use
- **Subject Line**: Unlock Your Exclusive Offer Now-Priority Access Awaits, (CampaignMember LastName)!
- Add your HTML Content.
- Click **Save**.



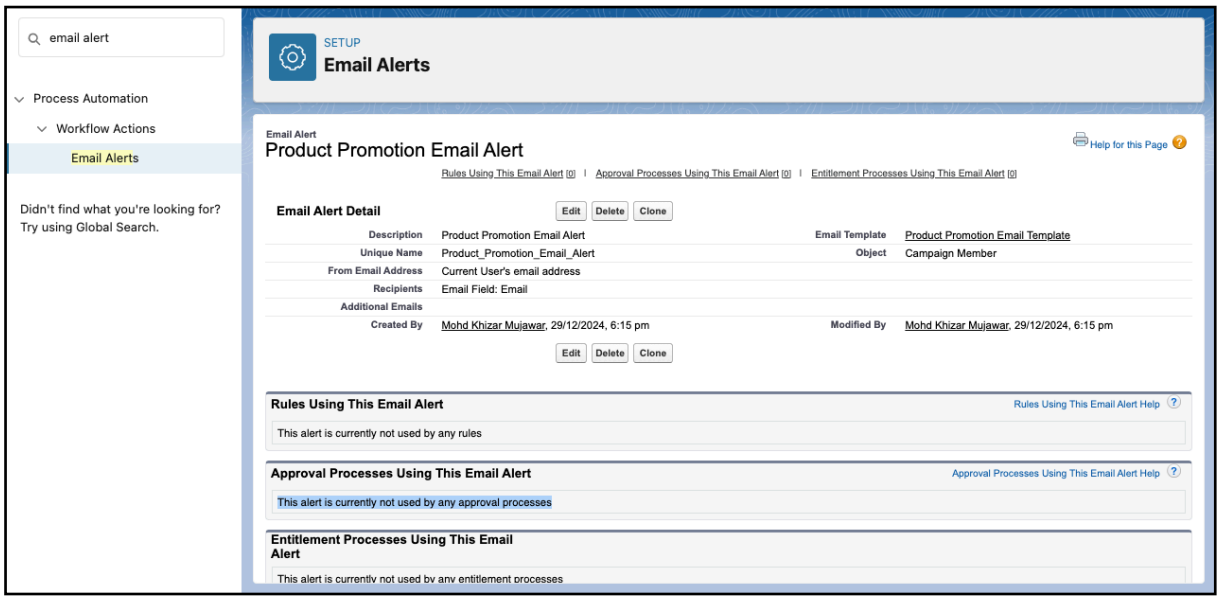
Step 3: Prepare the Email Alert

1. Navigate to Email Alert:

- Go to **Setup** → Search for **Email Alerts**.
- Click **New Email Alert**.

2. Create the Email Template:

- **Description**: "Product Promotion Email Alert"
- **Object:** Campaign Member
- **Recipient**: Email Field: Email of Campaign Member
- Click **Save**.



Step 4: Create the Flow to Automate the Campaign

1. Navigate to Flows:

- Go to **Setup** → Search for **Flows**.
- Click **New Flow**.

2. Choose Flow Type:

- Select **Record-Triggered Flow** (this will trigger when a Campaign Member is added).
- Click **Next**.

3. Configure Flow Trigger:

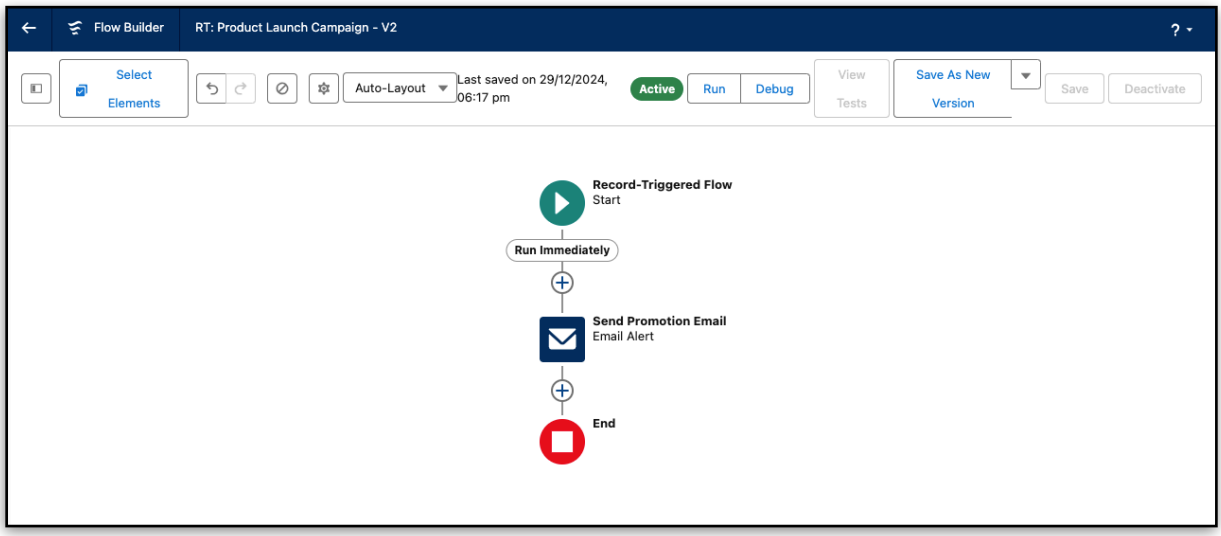
- **Object**: Campaign Member.
- **Trigger**: When a record is created.
- **Set conditions :**
  - **Campaign Status** = Responded AND
  - **Email** is Null False — If there is no email in campaign member record, Flow will not send email

4. Send Email Alert:

- Use the **Send Email Alert** action.
- Select the created Email Alert
- **Label:** Send Promotion Email
- **Record Id:** Triggered Campaign Member Record Id i.e {!$Record.Id}

5. Save and Activate:

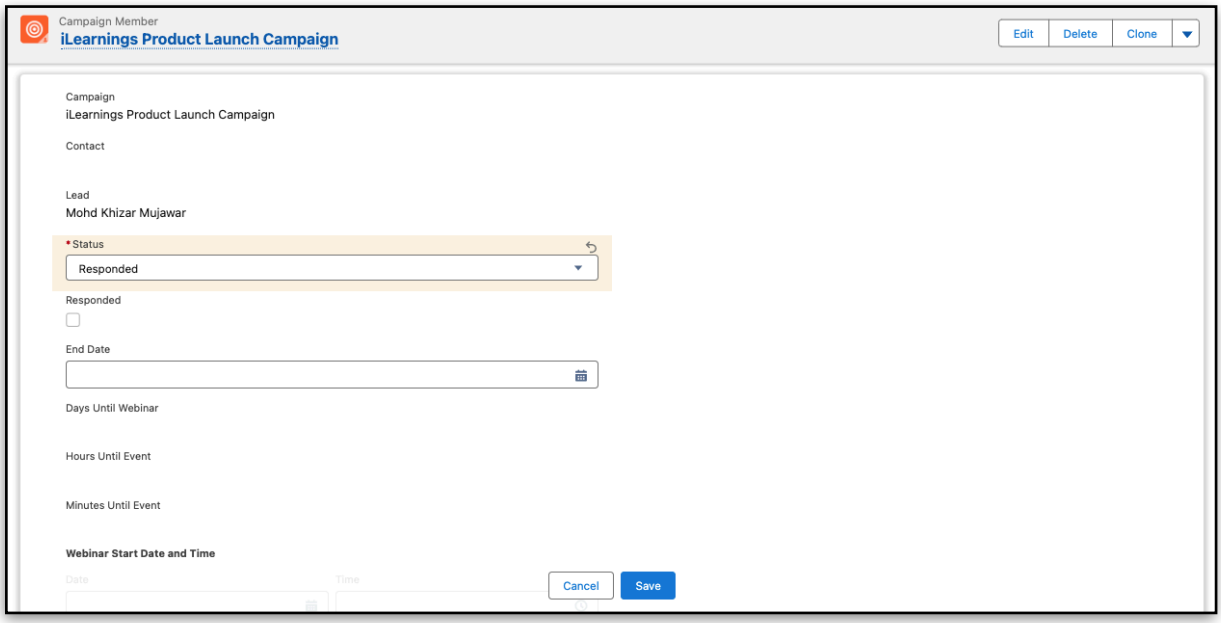
- Name your flow: “RT: Product Launch Campaign Email".
- Activate the flow.



Step 5: Test the Campaign Automation

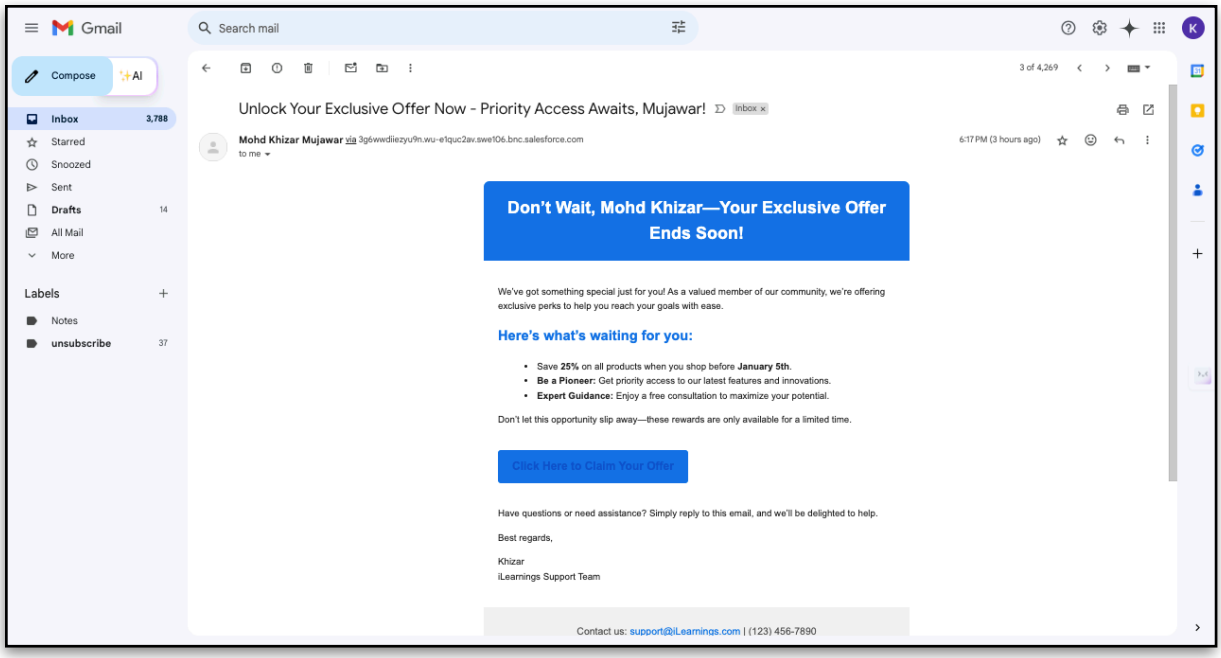
1. Add a Test Campaign Member:

- Add a test Lead to the Campaign.
- Make sure to enter Email Address.



2. Verify Email Delivery:

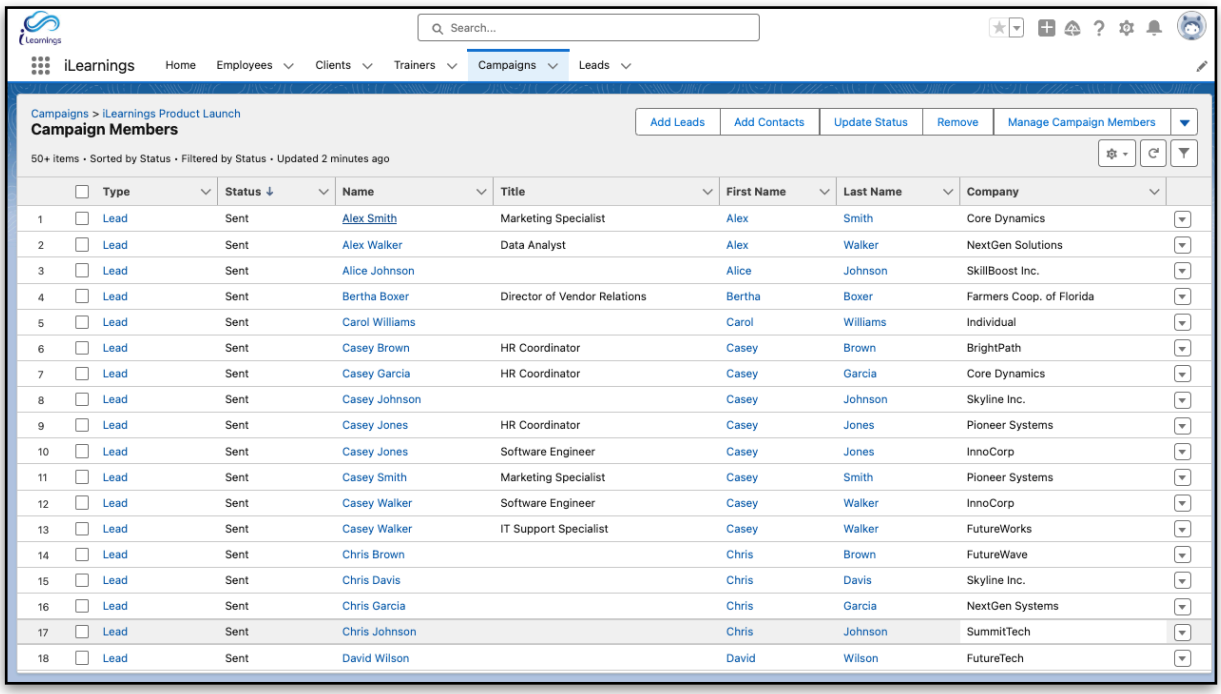
- Check that the email is sent to the recipient's email address.
- Confirm that the content matches the template and includes correct details.



Step 6: Track Campaign Performance

1. Monitor Campaign Members:

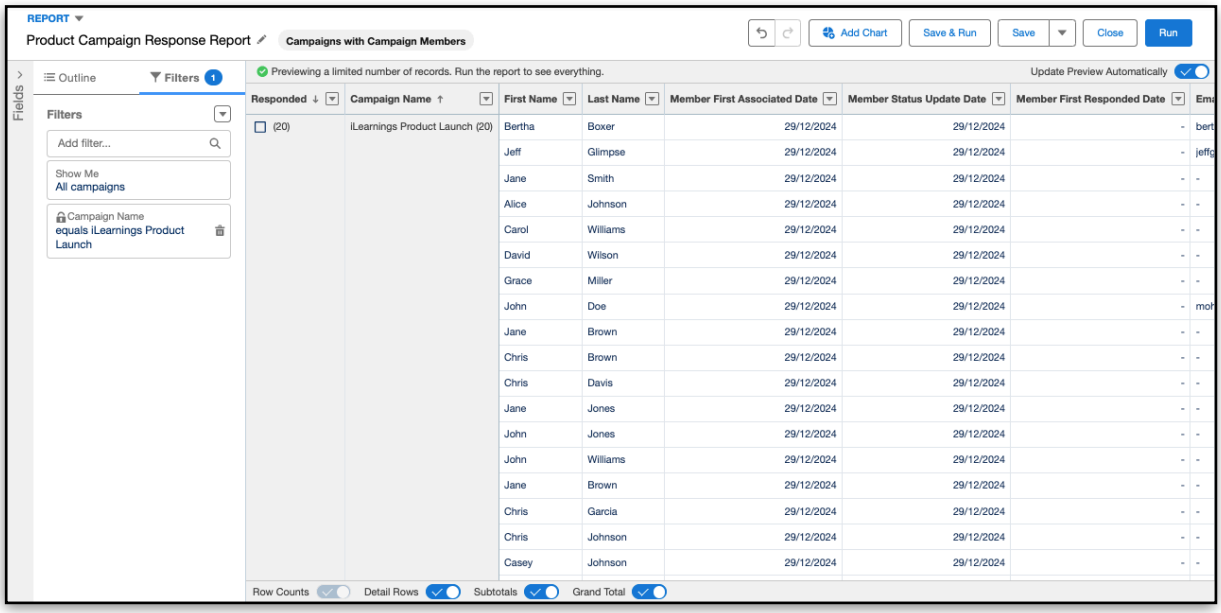
- Open the Campaign record.
- Check the **Campaign Members** list and their statuses.



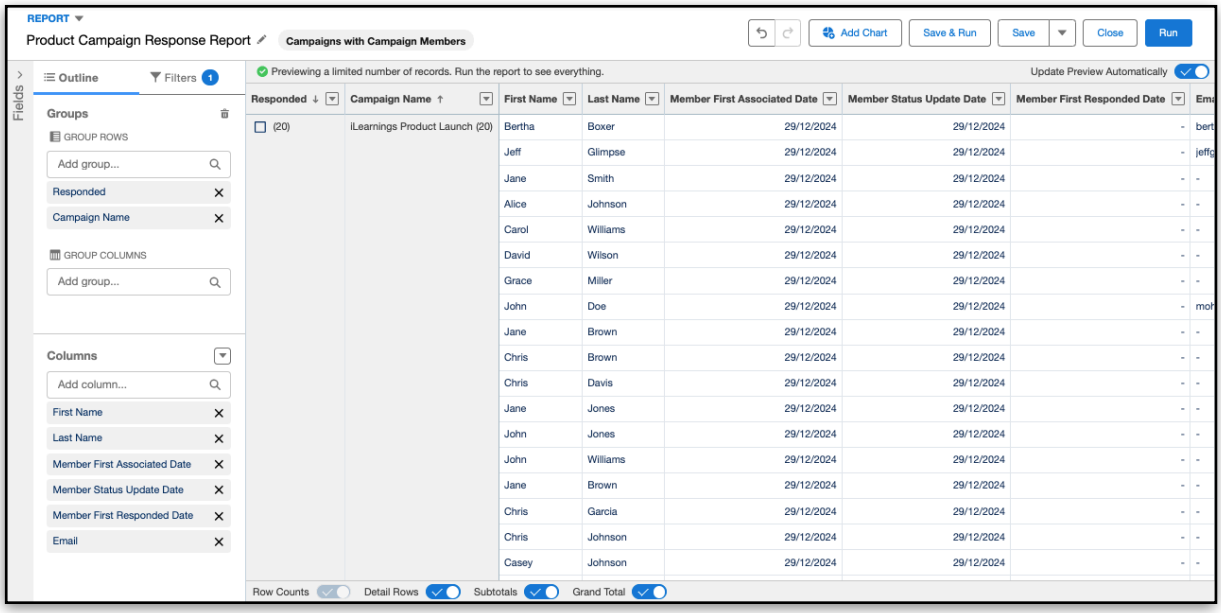
	Type	Status	Name	Title	First Name	Last Name	Company
1	Lead	Sent	Alex Smith	Marketing Specialist	Alex	Smith	Core Dynamics
2	Lead	Sent	Alex Walker	Data Analyst	Alex	Walker	NextGen Solutions
3	Lead	Sent	Alice Johnson		Alice	Johnson	SkillBoost Inc.
4	Lead	Sent	Bertha Boxer	Director of Vendor Relations	Bertha	Boxer	Farmers Coop. of Florida
5	Lead	Sent	Carol Williams		Carol	Williams	Individual
6	Lead	Sent	Casey Brown	HR Coordinator	Casey	Brown	BrightPath
7	Lead	Sent	Casey Garcia	HR Coordinator	Casey	Garcia	Core Dynamics
8	Lead	Sent	Casey Johnson		Casey	Johnson	Skyline Inc.
9	Lead	Sent	Casey Jones	HR Coordinator	Casey	Jones	Pioneer Systems
10	Lead	Sent	Casey Jones	Software Engineer	Casey	Jones	InnoCorp
11	Lead	Sent	Casey Smith	Marketing Specialist	Casey	Smith	Pioneer Systems
12	Lead	Sent	Casey Walker	Software Engineer	Casey	Walker	InnoCorp
13	Lead	Sent	Casey Walker	IT Support Specialist	Casey	Walker	FutureWorks
14	Lead	Sent	Chris Brown		Chris	Brown	FutureWave
15	Lead	Sent	Chris Davis		Chris	Davis	Skyline Inc.
16	Lead	Sent	Chris Garcia		Chris	Garcia	NextGen Systems
17	Lead	Sent	Chris Johnson		Chris	Johnson	SummitTech
18	Lead	Sent	David Wilson		David	Wilson	FutureTech

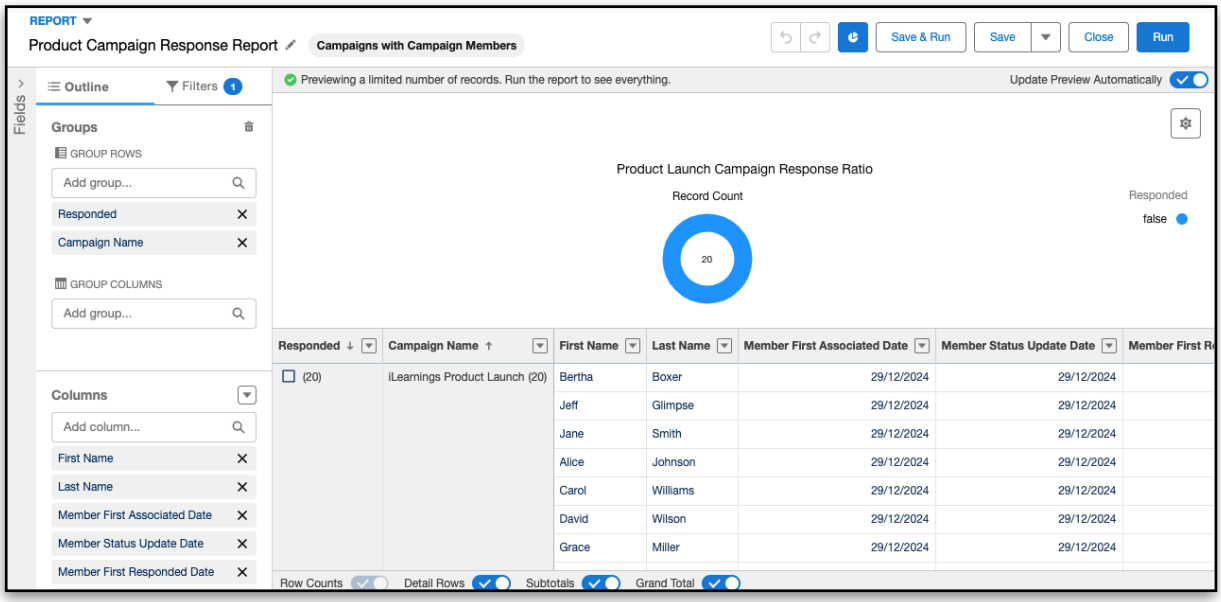
2. Generate Reports:

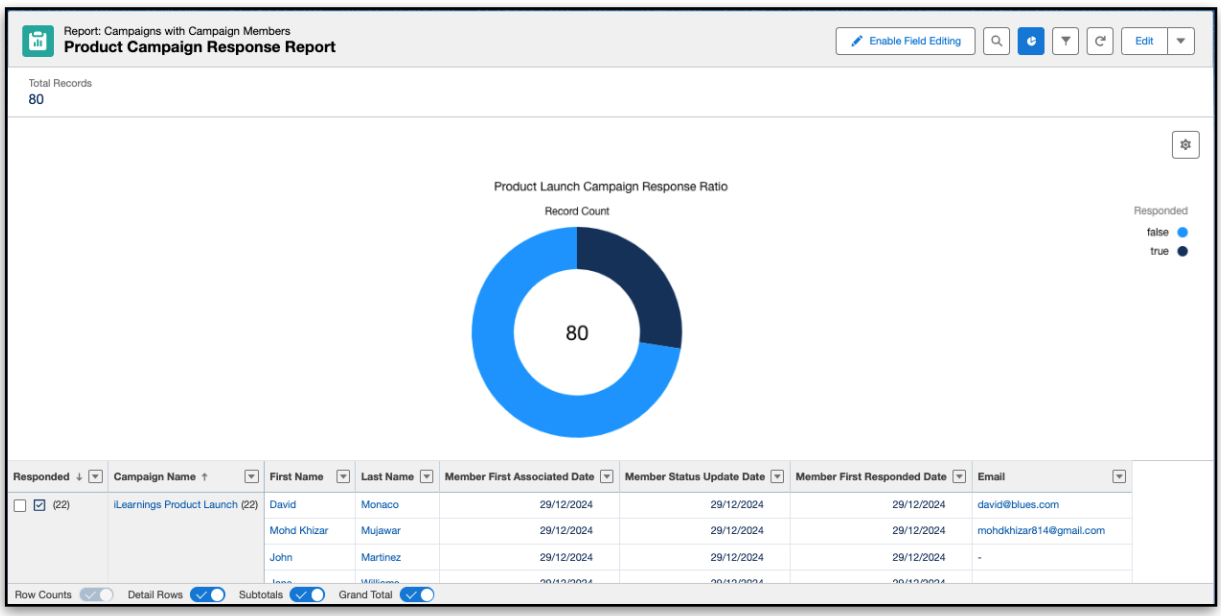
- Go to the **Reports** tab → Select **Campaigns with Campaign Members**.
- Customize the report fields to show the data you need, such as:
  - Number of emails sent.
  - How Campaign Members responded.



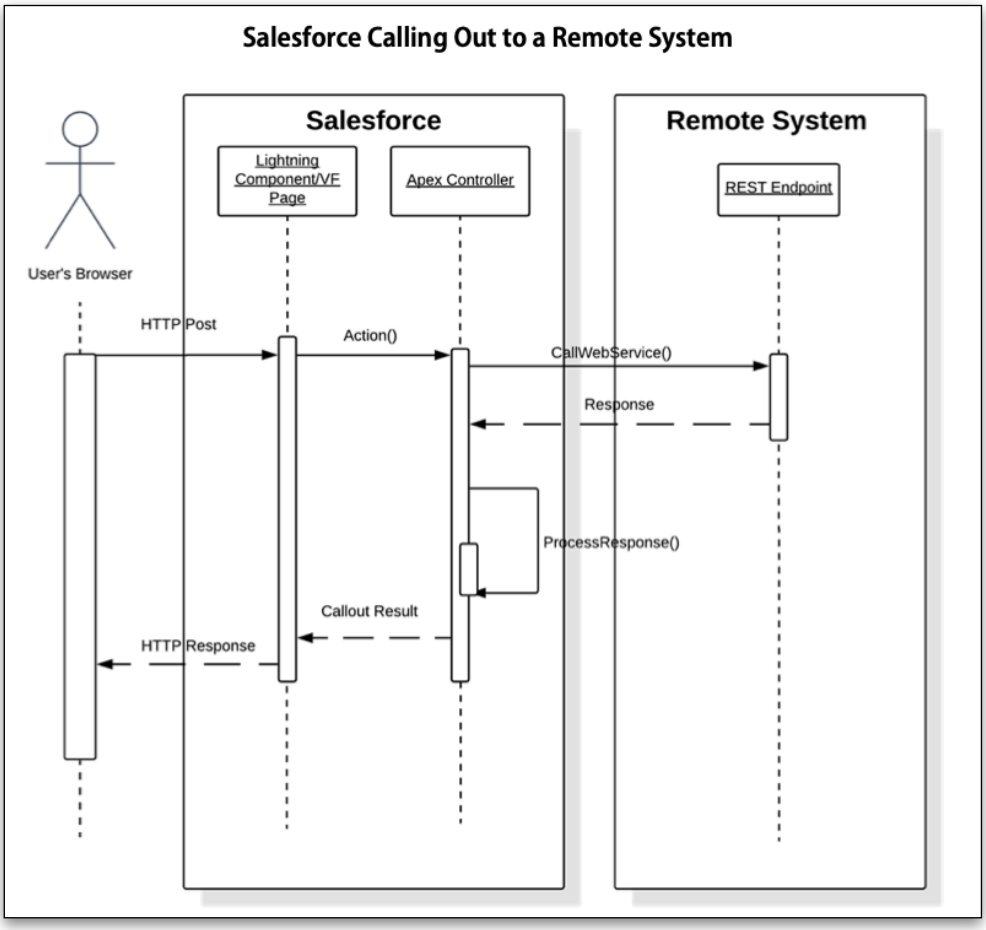
Responded	Campaign Name	First Name	Last Name	Member First Associated Date	Member Status Update Date	Member First Responded Date
(20)	iLearnings Product Launch (20)	Bertha	Boxer	29/12/2024	29/12/2024	-
		Jeff	Glimpse	29/12/2024	29/12/2024	-
		Jane	Smith	29/12/2024	29/12/2024	-
		Alice	Johnson	29/12/2024	29/12/2024	-
		Carol	Williams	29/12/2024	29/12/2024	-
		David	Wilson	29/12/2024	29/12/2024	-
		Grace	Miller	29/12/2024	29/12/2024	-
		John	Doe	29/12/2024	29/12/2024	-
		Jane	Brown	29/12/2024	29/12/2024	-
		Chris	Brown	29/12/2024	29/12/2024	-
		Chris	Davis	29/12/2024	29/12/2024	-
		Jane	Jones	29/12/2024	29/12/2024	-
		John	Jones	29/12/2024	29/12/2024	-
		John	Williams	29/12/2024	29/12/2024	-
		Jane	Brown	29/12/2024	29/12/2024	-
		Chris	Garcia	29/12/2024	29/12/2024	-
		Chris	Johnson	29/12/2024	29/12/2024	-
		Casey	Johnson	29/12/2024	29/12/2024	-


REPORT
Product Campaign Response Report
Campaigns with Campaign Members
Add Chart
Save & Run
Save
Close
Run
Fields
Outline
Filters 1
Previewing a limited number of records. Run the report to see everything.
Update Preview Automatically
Groups
GROUP ROWS
Add group...
Responded
Campaign Name
GROUP COLUMNS
Add group...
Columns
Add column...
First Name
Last Name
Member First Associated Date
Member Status Update Date
Member First Responded Date
Email
Responded
Campaign Name
First Name
Last Name
Member First Associated Date
Member Status Update Date
Member First Responded Date
Email
(20)
iLearnings Product Launch (20)
Bertha Boxer 29/12/2024 29/12/2024
Jeff Glimpse 29/12/2024 29/12/2024
Jane Smith 29/12/2024 29/12/2024
Alice Johnson 29/12/2024 29/12/2024
Carol Williams 29/12/2024 29/12/2024
David Wilson 29/12/2024 29/12/2024
Grace Miller 29/12/2024 29/12/2024
John Doe 29/12/2024 29/12/2024
Jane Brown 29/12/2024 29/12/2024
Chris Brown 29/12/2024 29/12/2024
Chris Davis 29/12/2024 29/12/2024
Jane Jones 29/12/2024 29/12/2024
John Jones 29/12/2024 29/12/2024
John Williams 29/12/2024 29/12/2024
Jane Brown 29/12/2024 29/12/2024
Chris Garcia 29/12/2024 29/12/2024
Chris Johnson 29/12/2024 29/12/2024
Casey Johnson 29/12/2024 29/12/2024
Row Counts
Detail Rows
Subtotals
Grand Total


REPORT
Product Campaign Response Report
Campaigns with Campaign Members
Save & Run
Save
Close
Run
Fields
Outline
Filters 1
Previewing a limited number of records. Run the report to see everything.
Update Preview Automatically
Groups
GROUP ROWS
Add group...
Responded
Campaign Name
GROUP COLUMNS
Add group...
Columns
Add column...
First Name
Last Name
Member First Associated Date
Member Status Update Date
Member First Responded Date
Product Launch Campaign Response Ratio
Record Count
20
Responded
false
Responded
Campaign Name
First Name
Last Name
Member First Associated Date
Member Status Update Date
(20)
iLearnings Product Launch (20)
Bertha Boxer 29/12/2024 29/12/2024
Jeff Glimpse 29/12/2024 29/12/2024
Jane Smith 29/12/2024 29/12/2024
Alice Johnson 29/12/2024 29/12/2024
Carol Williams 29/12/2024 29/12/2024
David Wilson 29/12/2024 29/12/2024
Grace Miller 29/12/2024 29/12/2024
Row Counts
Detail Rows
Subtotals
Grand Total


Report: Campaigns with Campaign Members
Product Campaign Response Report
Enable Field Editing
Edit
Total Records
80
Product Launch Campaign Response Ratio
Record Count
80
Responded
false
true
Responded
Campaign Name
First Name
Last Name
Member First Associated Date
Member Status Update Date
Member First Responded Date
Email
(22)
iLearnings Product Launch (22)
David Monaco 29/12/2024 29/12/2024 29/12/2024 david@blues.com
Mohd Khizar Mujawar 29/12/2024 29/12/2024 29/12/2024 mohdkhizar814@gmail.com
John Martinez 29/12/2024 29/12/2024 29/12/2024 -
Row Counts
Detail Rows
Subtotals
Grand Total

INTEGRATION



Remote Process Invocation—Request and Reply Sequence Diagram

TASK-1

Objective:

Implement a salary calculator for employees using a third-party app

I couldn't found Third Party App to complete this task.
So I tried creating my own REST API Flask Third Party App. It worked fine locally, but I encountered an error after deploying it on Vercel!

Created API:

Structure:

FLASK-API-EXAMPLE/

```
├── api/                  # Folder for API logic
│   └── __index.py        # Initialize the API package (optional)
├── index.py              # Main Flask app file
├── .gitignore            # Git ignore file
├── requirements.txt      # Dependencies for the project
├── runtime.txt           # Specifies Python version (for Vercel)
└── vercel.json           # Vercel deployment configuration
```

Flask REST API:

```python
from flask import Flask, request, jsonify

app = Flask(__name__)

# Root endpoint
@app.route('/')
def home():
    return "Welcome to the Salary Calculator API! Use the /calculate-salary endpoint with a POST request."

# Endpoint for calculating net salary
@app.route('/calculate-salary', methods=['POST'])
def calculate_salary():
    try:
        # Get JSON data from the request
        data = request.get_json()

        # Extract fields from JSON
        basic_pay = data.get('basicPay', 0)
        allowances = data.get('allowances', 0)
        deductions = data.get('deductions', 0)

        # Validate input
        if not isinstance(basic_pay, (int, float)) or not isinstance(allowances, (int, float)) or not isinstance(deductions, (int, float)):
            return jsonify({"error": "Invalid input. All values must be numbers."}), 400

        # Calculate net salary
        net_salary = basic_pay + allowances - deductions

        # Return the result as JSON
        return jsonify({
            "basicPay": basic_pay,
            "allowances": allowances,
            "deductions": deductions,
            "netSalary": net_salary
        }), 200

    except Exception as e:
        return jsonify({"error": str(e)}), 500

# Run the Flask app
if __name__ == '__main__':
    # Run Flask on port 5001 (or any desired port)
    app.run(host='0.0.0.0' , port=5001)
    app.run(debug=True)


```

Tested API using POSTMAN

Use the correct method (e.g., Postman or cURL) to send a POST request with a JSON body.

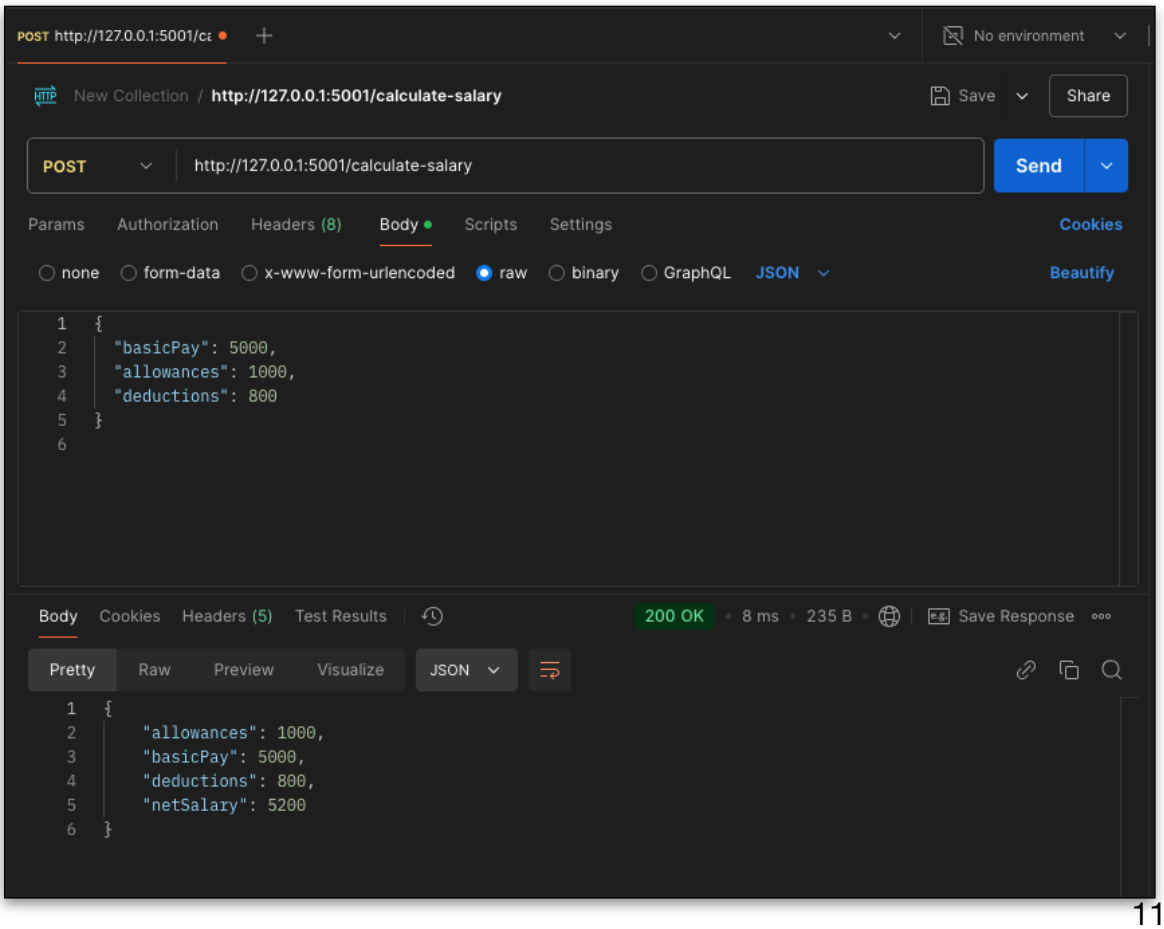
Steps

1. Set the request type to POST.
2. Set the URL to http://127.0.0.1:5000/calculate-salary.
3. Go to the **Body** tab → Select **raw** → Choose JSON.
4. Enter the following JSON and Click **Send**.:

```
{
 "basicPay": 5000,
 "allowances": 1000,
 "deductions": 800
}
```

5. Expected Response for Valid Input:

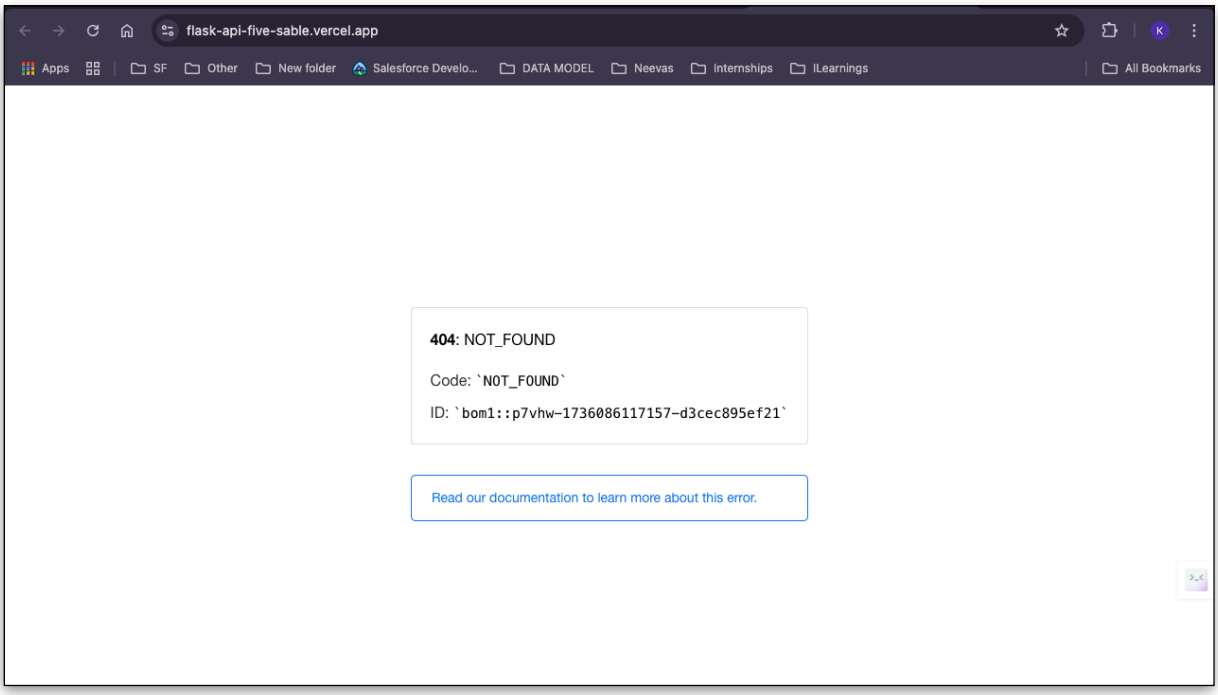
```
{
 "basicPay": 5000,
 "allowances": 1000,
 "deductions": 800,
 "netSalary": 5200
}
```



Deployment:

Steps:

1. **Push Code to GitHub**:
    - Commit your code to a Git repository.
    - Push the repository to GitHub.
2. **Sign Up on Vercel and Go to Vercel Dashboard**
3. **Import Code from GitHub**:
    - In Vercel, click on "Add New Project" or a similar option.
    - Link your GitHub account and import the repository containing your REST API code.
4. **Deploy the REST API App on Vercel**:
    - Follow the deployment steps prompted by Vercel.
    - Once deployed, Vercel will provide a public URL for your REST API.



Encountered an error after deploying REST API App on Vercel

Steps to Calculate Net Salary of Employee Record by Integrating Salesforce with created Third Party REST API App:

1. Prepare Salesforce Environment

Create Custom Fields on the Employee Object:

Navigate to **Object Manager** in Salesforce and add the following fields on the Employee object:

Field Name	API Name	Data Type	Description
Basic Pay	Basic_Salary__c	Currency(18, 2)	Base salary of the employee.
Allowance	Allowance__c	Currency(18, 2)	Fixed allowance.
Deductions	Other_Deductions__	Currency(18, 2)	Additional
Net Salary	Net_Salary__c	Currency(18, 2)	Final salary after deductions.

2. Prepare Your REST API

After correctly deploying REST API. Note the public URL of the /calculate-salary endpoint, for example:

Endpoint: https://your-vercel-app-url.vercel.app/calculate-salary

3. Create an Apex Class in Salesforce

In Salesforce, you'll use Apex to call the REST API and update the Net Salary field on the Employee object.

1. **Go to Developer Console in Salesforce**:
   - Click on your profile picture > Developer Console.
2. **Create a New Apex Class**:
   - In Developer Console, go to File > New > Apex Class.
   - Name the class, e.g., SalaryCalculator.
3. **Write Apex Code:** Use the following example code to integrate with your REST API.

```
public with sharing class SalaryCalculator {
    // Method to calculate and update Net Salary
    public static void calculateNetSalary(Id employeeId) {
        try {
            // Query the Employee record
```

```
        Employee__c employee = [SELECT Basic_Pay__c, Allowance__c, Deductions__c
FROM Employee__c WHERE Id = :employeeId LIMIT 1];

        // Prepare the HTTP request
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://your-vercel-app-url.vercel.app/calculate-salary');
        request.setMethod('POST');
        request.setHeader('Content-Type', 'application/json');

        // Prepare JSON body
        Map<String, Object> requestBody = new Map<String, Object>();
        requestBody.put('basicPay', employee.Basic_Pay__c);
        requestBody.put('allowances', employee.Allowance__c);
        requestBody.put('deductions', employee.Deductions__c);

        // Set the body of the request
        request.setBody(JSON.serialize(requestBody));

        // Send the request
        HttpResponse response = http.send(request);

        // Parse the response
        if (response.getStatusCode() == 200) {
            Map<String, Object> responseBody = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());

            // Update the Employee record with Net Salary
            employee.Net_Salary__c = (Decimal) responseBody.get('netSalary');
            update employee;
        } else {
            System.debug('Error: ' + response.getStatus() + ' - ' + response.getBody());
            throw new Exception('Failed to calculate Net Salary. API returned error.');
        }
    } catch (Exception e) {
        System.debug('Exception: ' + e.getMessage());
        throw e;
    }
```

```
    }
}
```

Key Notes:

- Ensure API names for your custom fields and object are correct.
- Update the endpoint URL (your-vercel-app-url) to your actual deployed API URL.

4. Create a Trigger to Automatically Calculate Net Salary

You can automate the Net Salary calculation using a trigger on the Employee object.

1. **Create a Trigger**:
   - In Developer Console, go to File > New > Apex Trigger.
   - Name the trigger, e.g., EmployeeTrigger.
2. **Write Trigger Code**:

Example trigger to call the calculateNetSalary method after an Employee record is created or updated:

```
trigger EmployeeTrigger on Employee__c (after insert, after update) {
    for (Employee__c emp : Trigger.new) {
        SalaryCalculator.calculateNetSalary(emp.Id);
    }
}
```

4. Test the Integration

- **Step 1**: Create or update an Employee record in Salesforce and populate the Basic Pay, Allowance, and Deductions fields.
- **Step 2**: The trigger will call the REST API, calculate the Net Salary, and update the Net Salary field on the record.

TASK-2

Objective:

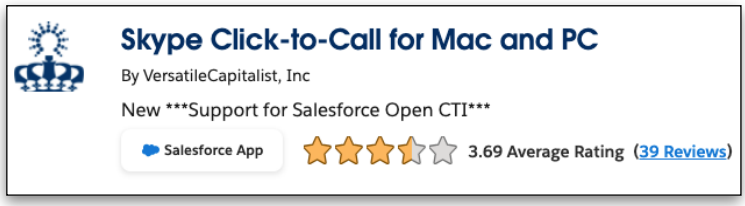
Enable option to call on Leads from Salesforce using a third-party app

Solution:

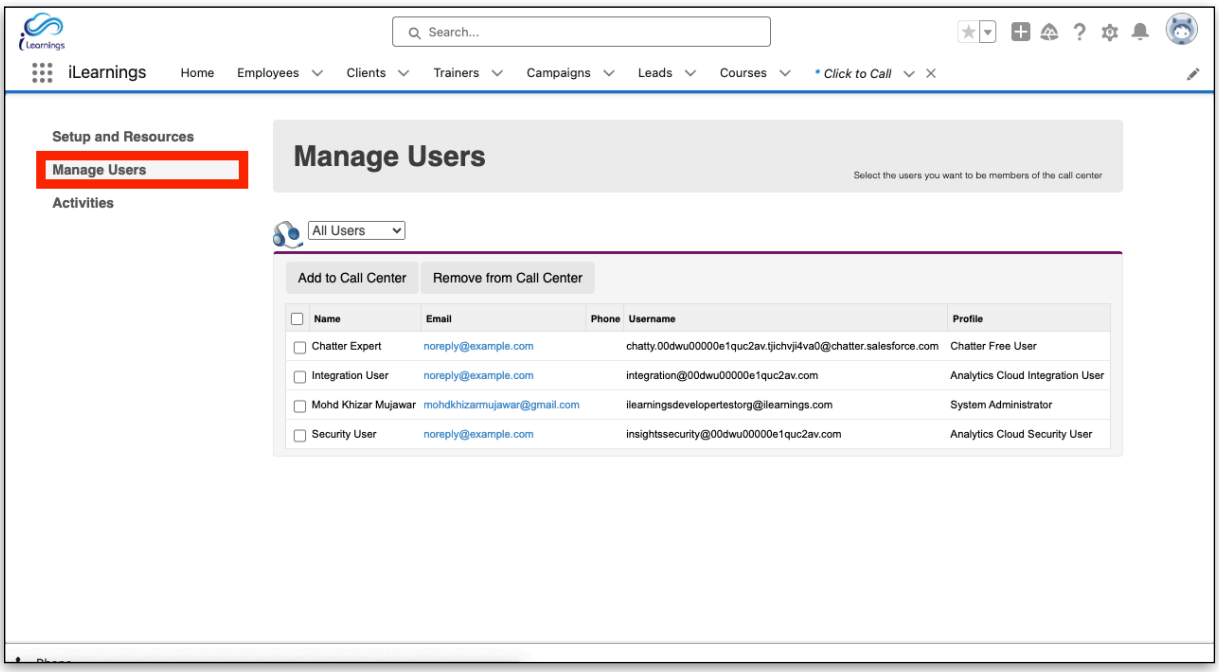
Using Click-To-Call App to complete this task

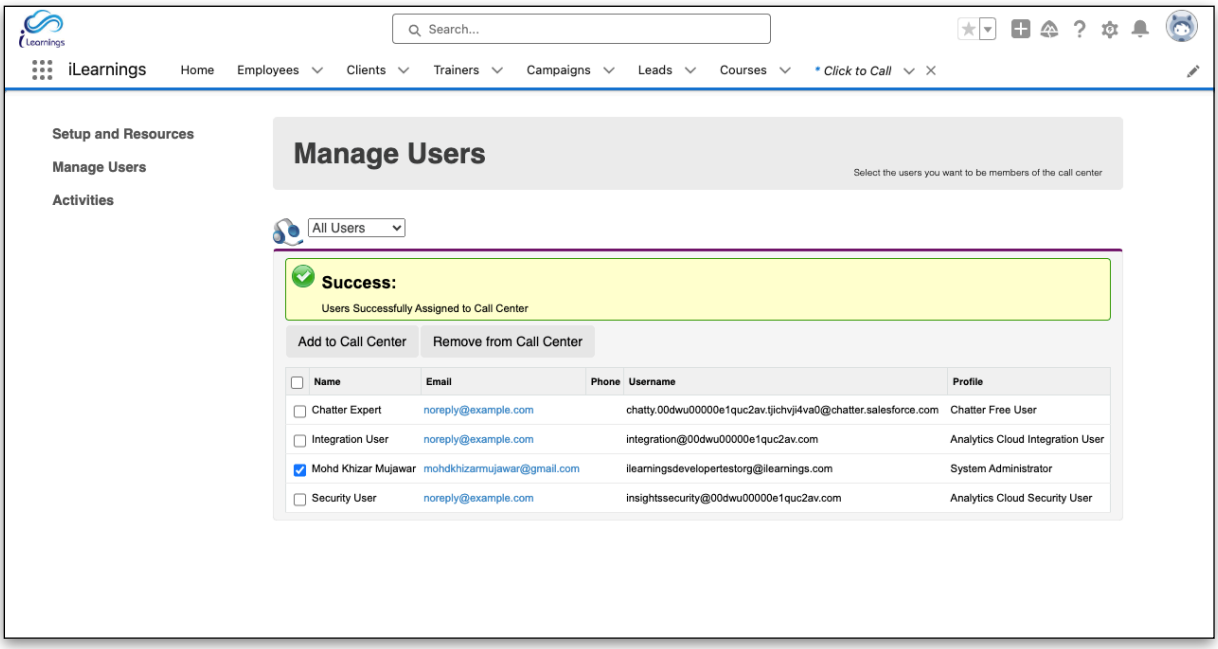
Steps:

1. Download the "Skype Click-to-Call for Mac and PC" application from the App Exchange.

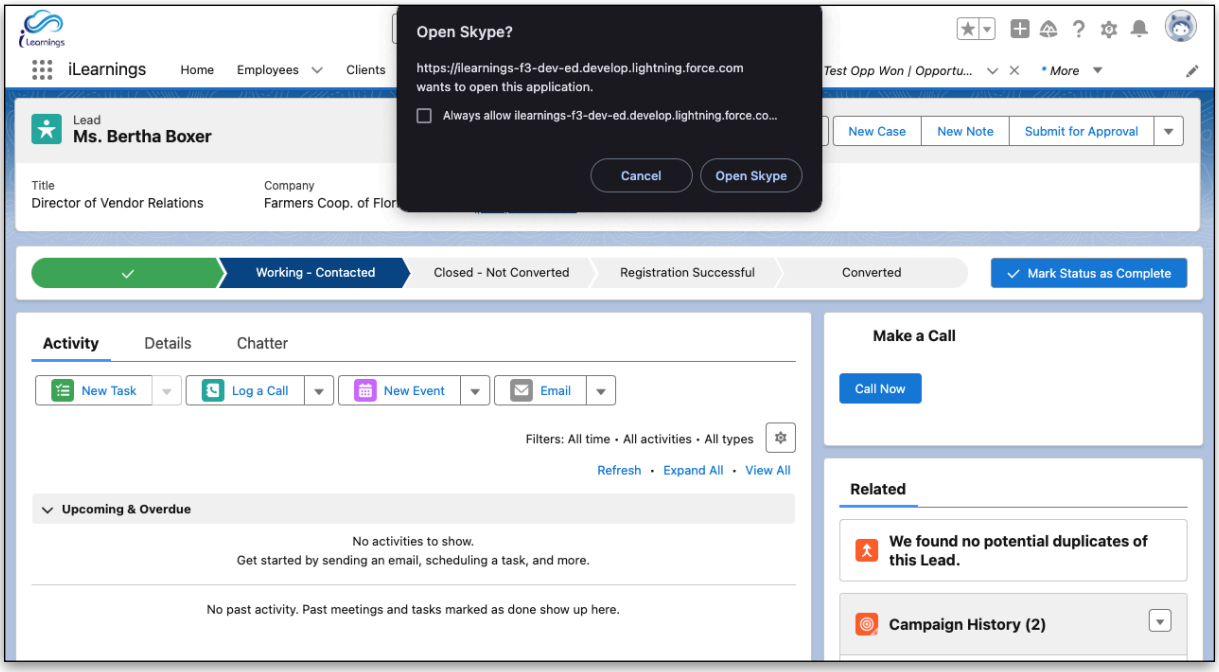


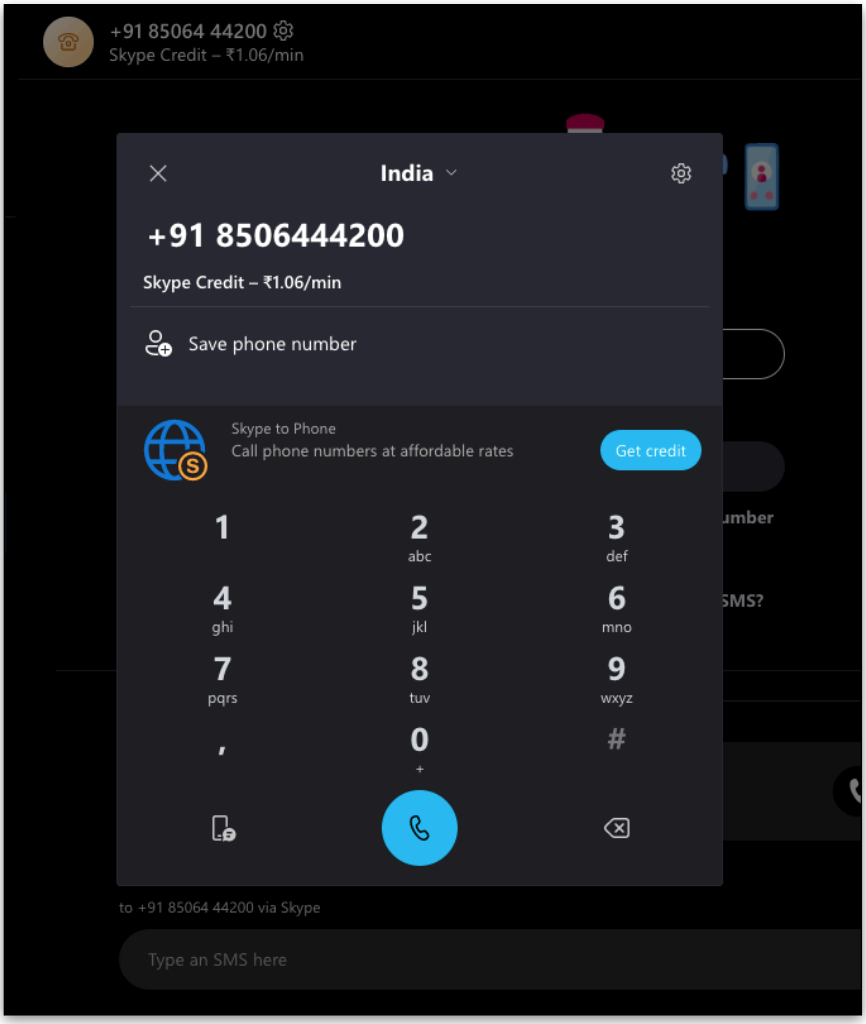
2. Click 'Manage Users', and add the user to the call center that you want to use with this demo adapter.





3. Go to any record page **(e.g. Lead**, in our case), and click the phone number.




+91 85064 44200
Skype Credit – ₹1.06/min
India
+91 8506444200
Skype Credit – ₹1.06/min
Save phone number
Skype to Phone
Call phone numbers at affordable rates
Get credit
1
2 abc
3 def
4 ghi
5 jkl
6 mno
7 pqrs
8 tuv
9 wxyz
,
0 +
#
to +91 85064 44200 via Skype
Type an SMS here

TASK-3

Objective:

Explore the possibility of payment integration

Overview of the Stripe Integration:

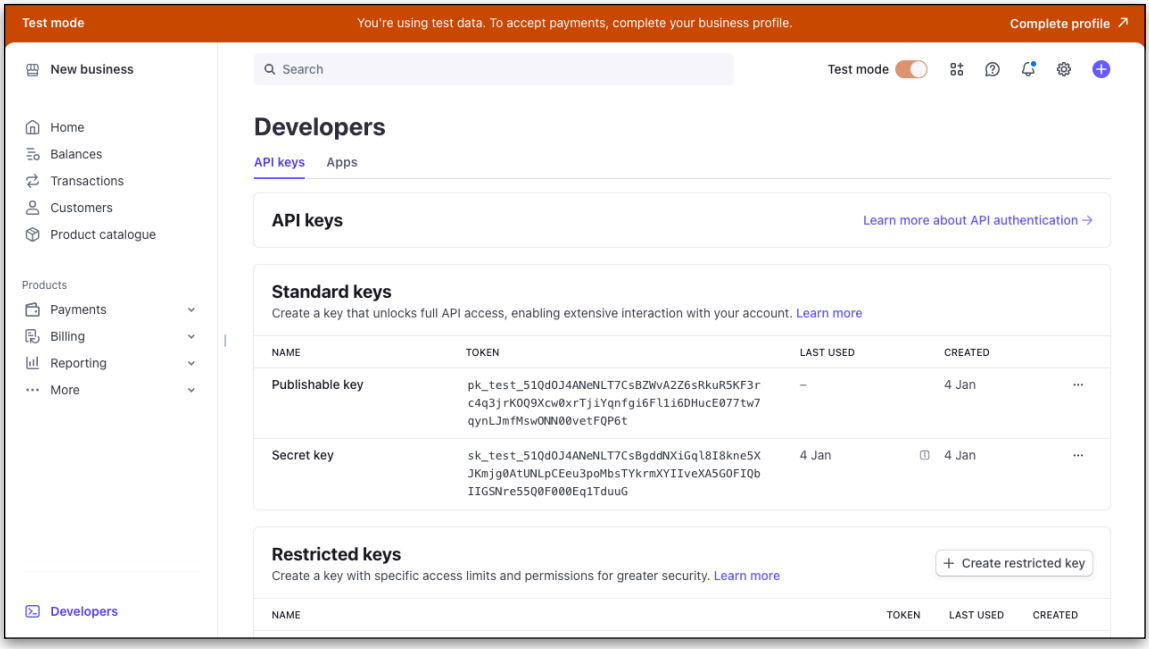
The payment flow for this integration consists of the following steps:

1. **Fetch Opportunity Line Items**: Retrieve the products and their details from a specific Opportunity record in Salesforce.
2. **Create Prices for Products**: Use Stripe's **Prices API** to create prices based on the product information.
3. **Generate Payment Link**: Use Stripe's **Payment Links API** to generate a link that customers can click to complete the payment.
4. **Send Payment Link**: Email the generated payment link to the Opportunity owner or the customer for them to complete the payment.

Step 1: Creating a Stripe Account and API Keys

Before you can integrate Stripe with Salesforce, you need to create a Stripe account. Here's how to get started:

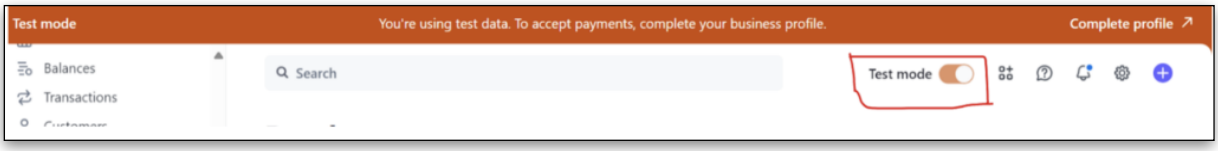
1. Go to Stripe's website(Stripe.com) and sign up.
2. Once you've signed up, you'll be directed to your dashboard.
3. Navigate to the API Section: From the Dashboard, click on the "Developers" tab in the left sidebar.
4. Click on "API keys": Here, you will find your Publishable Key and Secret Key as shown in the screenshot below.



- **Publishable Key**: This key is used on the client-side (browser) to identify your Stripe account and make public API requests.
- **Secret Key**: This key is used on the server-side to make private API requests and handle sensitive information like payment processing.

You will use these keys to integrate Stripe with Salesforce.

Test Mode: Initially, you will have Test mode to simulate payments. When you're ready to start accepting real payments, you'll switch to Live by Toggle Test Mode Button.

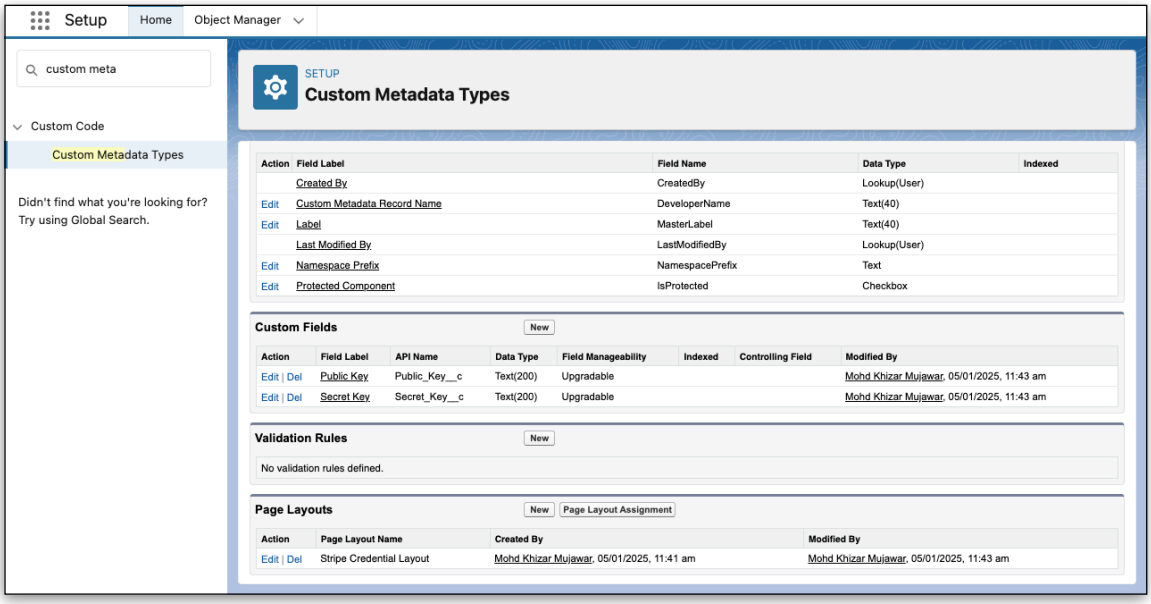


Step 2: Storing API Keys in Salesforce Custom Metadata Types

It's a good practice to store sensitive information, like Stripe's API keys, in Salesforce **Custom Metadata Types**. This makes the keys easily accessible from Apex while ensuring they are not exposed in the code itself.

Steps to Create Custom Metadata Types:

1. Go to Setup > Custom Metadata Types.
2. Create a new metadata type called Stripe_Credentials
3. Add Secret Key (Text) and Public Key (Text) fields to store your Stripe keys.
4. Create a record under Stripe_Credentials to store your actual API keys



Now, in your Apex code, you can retrieve these values securely like so:

```
Stripe_Credential__mdt credentials =
Stripe_Credential__mdt.getInstance('Stripe_Credentials');
secretKey = credentials.secretKey__c;
publicKey = credentials.publicKey__c;
```

Step 3: Building the Frontend: stripePaymentCmp

The frontend of this integration is built using Lightning Web Components (LWC). Specifically, we have the stripePaymentCmp component, which displays the Opportunity products and provides a button for users to initiate the payment process.

LWC HTML Template:

Here's the template of the LWC, which will display the products on the Opportunity and allow users to click the **Pay** button:

```
<template>
    <div class="slds-card">
        <div class="slds-grid slds-wrap">
            <div class="slds-col slds-size_12-of-12 slds-text-align_center">
                <h1 class="slds-text-heading_medium slds-text-title_bold">
                    Products for Payment</h1>
            </div>
        </div>
        <table class="slds-table slds-table_bordered">
            <thead>
                <tr>
                    <th>Name</th>
                    <th>ProductCode</th>
                    <th>Quantity</th>
                    <th>Total Price(USD)</th>
                </tr>
            </thead>
            <tbody>
                <template for:each={records} for:item="rec">
    <tr key={rec.id}>
                        <td>{rec.Product2.Name}</td>
                        <td>{rec.ProductCode}</td>
                        <td>{rec.Quantity}</td>
                        <td>{rec.TotalPrice}</td>
                    </tr>
                </template>
            </tbody>
        </table>
        <button type="button" disabled={isDisabled}
        onclick={handlePay} class="slds-button slds-button_brand">{totalPrice}
        </button>
    </div>
</template>
```

This HTML template is responsible for rendering the list of products with details such as product name, code, quantity, and total price.

LWC JavaScript Logic (stripePaymentCmp.js)

```
import { LightningElement, wire, track, api } from 'lwc';
import { CurrentPageReference } from 'lightning/navigation';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';
import getProducts from '@salesforce/apex/StripePaymentHelper.getOpportunityLineItems';
import sendPaymentRequest from '@salesforce/apex/StripePaymentHelper.sendPaymentRequest';

export default class StripeProductListCmp extends LightningElement {
    error;
    records;
    totalPrice = 0;
    recordId;
    isProcess = false;
    isDisabled = false;

    @wire(CurrentPageReference)
 getStateParameters(currentPageReference) {
        if (currentPageReference) {
            this.recordId = currentPageReference.state.recordId;
            getProducts({ parentId: this.recordId })
                .then((result) => {
                    this.records = result;
                    this.totalPrice = 'Pay($' + result.reduce(
                        (sum, rec) => sum + rec.TotalPrice, 0) + ')';
                })
                .catch((error) => {
                    console.error('Error: ', error);
                });
        }
    }

    handlePay() {
        this.isProcess = true;
        this.isDisabled = true;
        const dataJson = JSON.stringify(this.records);
        sendPaymentRequest({ productsJson: dataJson })
            .then((result) => {
                this.isProcess = false;
                this.showToast('Success',
                    'Payment request successfully sent',
                    'success');
            })
            .catch((error) => {
                this.isProcess = false;
                this.isDisabled = false;
                this.showToast('Error', 'Something went wrong: '
                    + error, 'error');
            });
    }

    showToast(title, message, variant) {
        const evt = new ShowToastEvent({
            title: title,
            message: message,
            variant: variant,
            mode: 'dismissable'
        });
        this.dispatchEvent(evt);
    }
}
```

Fetching Opportunity Products: The component uses **getProducts** (Apex method) to fetch the products related to the Opportunity.
Triggering Payment Request: The **handlePay** method triggers the payment process by sending the product details to the **sendPaymentRequest()** Apex method.

XML (stripePaymentCmp.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>62.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
    <target>lightning__RecordPage</target>
    </targets>
</LightningComponentBundle>
```

Step 4: Building the Backend

4.1 Creating Prices ID using Price API

Price ID:

A Price ID represents the cost of a product or service. It's tied to a specific product in Stripe.

Requirements for Creating a new price:

When creating a new price for an existing product in Stripe, there are a few key elements to define, depending on whether the price is recurring or one-time:

1. **Currency**: Required field.Must be the three-letter ISO currency code in lowercase (e.g., “inr” for Indian Rupees).
2. **Product**: Required unless product_data is provided.This is the unique ID of the product to which the price will be linked.
3. **Unit Amount:** Required unless using unit_amount_decimal or custom_unit_amount.This is the price amount in cents (e.g., 1000 = $10.00 USD). A value of 0 can be used for free products.

By providing the necessary details for these fields, you can successfully create a price in Stripe for both one-time and recurring payments.

Generating Prices API:

- To create it, go to https://docs.stripe.com/api/prices/create.
- Click on  in the right down corner > **API Explorer** > in **Resource**, and fill these fields:

Resource	Price
Method	Create

Required Parameter	Value
Currency	Ins

Optional Parameter	Value
active	True
Billing scheme	perUnit
Product Data_name	iLearning Product
recurring > aggregate_usage	null
recurring > interval	month
recurring > interval_count	1
recurring > trial_period_days	null
recurring > usage_type	licensed

Tax behaviour	unspecified
Unit amount	1,500,000 - 15k ₹

- Click **Send Request**.
- Price Id gets generated.


```
{
 "id": "price_1QdzJWANeNLT7CsBwSZPoDiX",
 "object": "price",
 "active": true,
 "billing_scheme": "per_unit",
 "created": 1736104602,
 "currency": "inr",
 "custom_unit_amount": null,
 "livemode": false,
 "lookup_key": null,
 "metadata": {},
 "nickname": null,
 "product": "prod_RX3QDDHOL05mlM",
 "recurring": {
  "aggregate_usage": null,
  "interval": "month",
  "interval_count": 1,
  "meter": null,
  "trial_period_days": null,
  "usage_type": "licensed"
 },
 "tax_behavior": "unspecified",
 "tiers_mode": null,
 "transform_quantity": null,
 "type": "recurring",
 "unit_amount": 150000,
 "unit_amount_decimal": "150000"
}
```


- Note down the priceId, we will used it later to create Payment Link Id.
- Later, Payment Link Id will be used to create Payment Link in apex code.

4.2 Creating Payment Link in Stripe

Payment Link:

A payment link is a shareable URL that will take your customers to a hosted payment page. A payment link can be shared and used multiple times.

Requirements for Creating a Payment Link:

When creating a payment link in Stripe, you must provide the line items representing what is being sold. Each line item has two key attributes:

- **price:** The ID of the price (e.g., price_1ABCXYZ).
- **quantity:** The number of items being sold.

Generating Payment Link using generated Price API:

Steps

- To create it, go to docs.stripe.com/api/payment-link/create?shell=true&api=true
- Click on  in the right down corner > **API Explorer** > in **Resource**, and fill these fields:

Resource	Price
Payment link	Create

Required Parameter	Value
Price	price_1QdzJWANeNLT7CsBwSZPoDiX{Paste your generated price id}

Optional Parameter	Value
Billing address collection	Auto
customer_creation	if_required
Product Data_name	iLearning Product
recurring > interval	month
recurring > interval_count	1
Unit amount	1,500,000 - 15k ₹

- Upon clicking **Send Request**, **Payment Link** will generated.


```
{
 "id": "plink_1QeDUwANeNLT7CsBd0n90YuR",
 "object": "payment_link",
 "active": true,
 "after_completion": {
  "hosted_confirmation": {
   "custom_message": null
  },
```

```
  "type": "hosted_confirmation"
},
"allow_promotion_codes": false,
"application": null,
"application_fee_amount": null,
"application_fee_percent": null,
"automatic_tax": {
  "enabled": false,
  "liability": null
},
"billing_address_collection": "auto",
"consent_collection": null,
"currency": "usd",
"custom_fields": [],
"custom_text": {
  "after_submit": null,
  "shipping_address": null,
  "submit": null,
  "terms_of_service_acceptance": null
},
"customer_creation": "if_required",
"inactive_message": null,
"invoice_creation": {
  "enabled": false,
  "invoice_data": {
    "account_tax_ids": null,
    "custom_fields": null,
    "description": null,
    "footer": null,
    "issuer": null,
    "metadata": {},
    "rendering_options": null
  }
},
"livemode": false,
"metadata": {},
"on_behalf_of": null,
"payment_intent_data": null,
"payment_method_collection": "if_required",
"payment_method_types": null,
"phone_number_collection": {
  "enabled": false
},
"restrictions": null,
"shipping_address_collection": null,
"shipping_options": [],
"submit_type": "auto",
```

```
  "subscription_data": null,
  "tax_id_collection": {
    "enabled": false,
    "required": "never"
  },
  "transfer_data": null,
  "url": "https://buy.stripe.com/test_8wM2ar9DX6lS1eE7sv"
}
```


- Copy and paste id in following apex code

Second Way:

You can generate **Price IDs** and **Payment Links IDs** directly through the Stripe Dashboard's user interface (UI) & use those payment link id in apex code.

Generating payment link and Sending Payment Link Through Email using Apex:

The payment logic resides in the Apex class. This class interacts with the Stripe API to create prices and generate payment links.

- Once prices are created for each product, the next step is to generate a **payment link** using the **Payment Links API**.
- Copy and paste generated **Payment Links Id** in following apex code.

Apex Code (StripePaymentHelper.cls)

```apex
public class StripePaymentHelper {
    private static final String baseURL = 'https://api.stripe.com';
    private static String getStripeKey() {
        Stripe_Credentials__mdt credentials = Stripe_Credentials__mdt.getInstance('Stripe_Keys');
        return credentials.Secret_Key__c;
    }

    public static String createPrice(OpportunityLineItem product) {
        String apiURL = baseURL + '/v1/prices';
        String requestBody = 'unit_amount_decimal=' + EncodingUtil.urlEncode(String.valueOf(product.TotalPrice), 'UTF-8') +
                             '&currency=' + EncodingUtil.urlEncode(String.valueOf(product.CurrencyIsoCode), 'UTF-8') +
                             '&product_data[name]=' + EncodingUtil.urlEncode(product.Product2.Name, 'UTF-8');

        HttpRequest request = new HttpRequest();
        request.setEndpoint(apiURL);
        request.setMethod('POST');
        request.setHeader('Authorization', 'Bearer ' + getStripeKey());
        request.setHeader('Content-Type', 'application/x-www-form-urlencoded');
        request.setBody(requestBody);

        HttpResponse response = new Http().send(request);

        if (response.getStatusCode() == 200) {
            Map<String, Object> result = (Map<String, Object>)JSON.deserializeUntyped(response.getBody());
            return (String)result.get('id');
        }
        throw new StripeException('Failed to create price: ' + response.getBody());
    }

    public static String createPaymentLink(List<String> priceIds, Opportunity opportunity) {
        String apiURL = baseURL + '/v1/payment_links';

        List<Map<String, Object>> lineItems = new List<Map<String, Object>>();
        for(String priceId : priceIds) {
            lineItems.add(new Map<String, Object>{
                'price' => priceId,
                'quantity' => 1
            });
        }

        Map<String, Object> requestMap = new Map<String, Object>{
            'line_items' => lineItems,
            'after_completion' => new Map<String, Object>{
                'type' => 'redirect',
                'redirect' => new Map<String, Object>{
                    'url' => URL.getSalesforceBaseUrl().toExternalForm() + '/' + opportunity.Id
                }
            }
        };

        HttpRequest request = new HttpRequest();
        request.setEndpoint(apiURL);
        request.setMethod('POST');
        request.setHeader('Authorization', 'Bearer ' + getStripeKey());
        request.setHeader('Content-Type', 'application/json');
        request.setBody(JSON.serialize(requestMap));

        HttpResponse response = new Http().send(request);

        if (response.getStatusCode() == 200) {
            Map<String, Object> result = (Map<String, Object>)JSON.deserializeUntyped(response.getBody());
            return (String)result.get('url');
        }
        throw new StripeException('Failed to create payment link: ' + response.getBody());
    }

    public static void sendPaymentEmail(String paymentURL, Opportunity opportunity) {
        Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
        email.setTargetObjectId(opportunity.OwnerId);
        email.setSubject('Payment Request for: ' + opportunity.Name);
        email.setPlainTextBody('Dear ' + opportunity.Account.Name + ',\n\nClick on this link ' +
                               paymentURL + ' to make a payment.');
        email.setSaveAsActivity(true);

        try {
            Messaging.sendEmail(new List<Messaging.SingleEmailMessage>{ email });
        } catch(Exception e) {
            throw new StripeException('Failed to send email: ' + e.getMessage());
        }
    }

    public class StripeException extends Exception {}
}
```

Step 5: Opportunity Action: “Send Payment” Button

- Deploy source code to org
- Create an Opportunity Action called “Send Payment,” which triggers the “stripePaymentCmp”.
- Navigate to Opportunity Object > Select Button, Links and Action > New Action
  - Action Type: Stripe payment cmp
  - Select created component

Step 6: Testing

- Once the customer clicks the payment link, they will be redirected to Stripe’s payment checkout interface to enter their card details and complete the payment. The page will display the product details, including the price, and you can customize this page to match your branding and design preferences.
- Once the payment is done, we will get the following success screen.
- Success Transactions from Stripe Dashboard as below snap.

AUTOMATIONS

TASK-4

Objective:

Consider all possible ways to create a seamless process for students.

Similar to previously completed task …

TASK-5

Objective:

Automated email for salary slips to be sent to employees with the Attachment.

Steps:

Generating and emailing salary slips in Salesforce can be achieved using a combination of standard Salesforce features, custom development (Apex and Visualforce/Lightning Components), and third-party integrations. Below is a step-by-step approach:

Step 1: Data Preparation

1.1 Fields in Employee Object:

- Assume fields like Basic_Pay__c, HRA__c, Other_Allowances__c, and Total_Salary__c exist in the Employee__c object.
- Add fields for the salary slip period, e.g., Salary_Month__c and Salary_Year__c.

1.2 Ensure Data is Ready:

- Use automation (Apex, Flow, or third-party integrations) to calculate and populate salary data in the Employee object.

Step 2: Generate Salary Slip (PDF)

2.1 Use Visualforce Page:

- Create a Visualforce page to format the salary slip layout.
- Include employee details, salary components, and the company logo.
- Use PDF rendering in the Visualforce page for downloadable slips.

Example:

```
<apex:page renderAs="pdf" standardController="Salary_Slip__c">
  <h1>Salary Slip for {!Salary_Slip__c.Month__c}/{!Salary_Slip__c.Year__c}</h1>
  <p>Employee: {!Salary_Slip__c.Employee__r.Name}</p>
  <p>Basic Pay: {!Salary_Slip__c.Basic_Pay__c}</p>
  <p>HRA: {!Salary_Slip__c.HRA__c}</p>
  <p>Total: {!Salary_Slip__c.Total__c}</p>
</apex:page>
```

2.2 Generate the PDF Programmatically:

- Use PageReference.getContentAsPDF() in Apex to generate the PDF file.

Step 3: Automate Email with PDF Attachment

3.1 Create an Email Template:

- Design an email template for sending salary slips. You can use either a Visualforce email template or an HTML email template.

3.2 Apex Code to prepare email:

- Assume fields like Basic_Pay__c, HRA__c, Other_Allowances__c, and Total_Salary__c exist in the Employee__c object.
- Use the Messaging class in Apex to attach the generated PDF and send the email.

Example:

```
public with sharing class SalarySlipEmailService {
    public static void sendSalarySlip(Id employeeId) {
        Employee__c employee = [
SELECT Id, Name, Email__c, Salary_Month__c, Salary_Year__c, Basic_Pay__c, HRA__c,
Other_Allowances__c, Total_Salary__c
FROM Employee__c
WHERE Id = :employeeId];

        // Generate PDF
        PageReference pdfPage = Page.EmployeeSalarySlipPDF; // Visualforce page name
        pdfPage.getParameters().put('id', employeeId);
        Blob pdfBlob = pdfPage.getContentAsPDF();

        // Prepare Email
        Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
        email.setToAddresses(new String[] {employee.Email__c});
        email.setSubject('Salary Slip for ' + employee.Salary_Month__c + '/' +
employee.Salary_Year__c);
        email.setHtmlBody('Dear ' + employee.Name + ',<br><br>Please find your salary slip for '
                + employee.Salary_Month__c + '/' + employee.Salary_Year__c + '
attached.<br><br>Regards,<br>HR Team');
        email.setFileAttachments(new Messaging.EmailFileAttachment[] {
            new Messaging.EmailFileAttachment(
                'SalarySlip_' + employee.Name + '_' + employee.Salary_Month__c + '_' +
employee.Salary_Year__c + '.pdf',
                pdfBlob
            )
        });

        // Send Email
        Messaging.sendEmail(new Messaging.SingleEmailMessage[] {email});
    }
}
```

3.3 Automate for All Employees:

- Use Batch Apex to loop through employees and send salary slips in bulk.
- **Example Batch Apex skeleton**:

```
public class SalarySlipBatch implements Database.Batchable<SObject> {
    public Database.QueryLocator start(Database.BatchableContext BC) {
```

```
        return Database.getQueryLocator('SELECT Id FROM Employee__c WHERE
    Salary_Month__c = THIS_MONTH');
    }
    public void execute(Database.BatchableContext BC, List<Employee__c> scope) {
        for (Employee__c emp : scope) {
            SalarySlipEmailService.sendSalarySlip(emp.Id);
        }
    }
    public void finish(Database.BatchableContext BC) {
        // Optional: Log or notify completion
    }
}
```

3.4 Schedule the Batch Job:

- Use Schedulable Apex to schedule the batch job monthly.

Step 4: Test and Verify

- **Test with Sample Records:**
  - Test email delivery, PDF attachment formatting, and accuracy of salary details.
- **Verify in Bulk:**
  - Run a batch job for multiple employees to ensure the process works at scale.

Step 5: Enhance with Third-Party Tools [Optional]

- Use apps from the Salesforce AppExchange (e.g., Conga, DocuSign) for advanced document generation and email workflows.
- Integrate with email marketing tools like Mailchimp for batch email delivery.

TASK-6

Objective:

Certificates to be sent to Leads after course completion

Steps:

To send certificates to leads after course completion in Salesforce, you can implement a solution using automation tools like Flows or custom Apex. Here's a step-by-step guide:

Step 1: Prepare Certificate Data

1. **Add Relevant Fields to Lead Object:**
   - Course_Name__c: To store the course name.
   - Completion_Date__c: To store the date of course completion.
   - Certificate_Sent__c (Checkbox): To track if the certificate has been sent.
2. **Store Certificate Templates:**
   - Use Salesforce Files or a Visualforce page to create a certificate template.
   - Alternatively, use a custom PDF generator.

Step 2: Generate Certificates

1. **Use Visualforce Page for Certificate Design:**
   - Create a Visualforce page for the certificate layout, dynamically populated with lead data.

```
<apex:page renderAs="pdf" standardController="Lead">
    <h1>Certificate of Completion</h1>
    <p>This certifies that <strong>{!Lead.Name}</strong> has successfully completed the course:</p>
    <p><strong>{!Lead.Course_Name__c}</strong></p>
    <p>Completion Date: {!Lead.Completion_Date__c}</p>
</apex:page>
```

2. **Generate the Certificate as a PDF:**
   - Use PageReference.getContentAsPDF() in Apex if programmatic generation is needed.

Step 3: Automate Certificate Sending

1. **Flow Solution (Declarative Approach):**
   - Use a **Record-Triggered Flow**:
     - Trigger: When a Lead record is updated, and Completion_Date__c is populated.

- Action: Use a **Send Email** action to send the certificate email with a predefined email template.
- Update the Certificate_Sent__c field to true after sending.

2. Email with Certificate Attachment:

- If you need to attach a PDF, use an Apex action in Flow to generate the PDF and attach it to the email.

3. **Apex Solution (Programmatic Approach):**

```
public with sharing class CertificateEmailService {
    public static void sendCertificate(Id leadId) {
        Lead lead = [SELECT Id, Name, Email, Course_Name__c, Completion_Date__c FROM
Lead WHERE Id = :leadId];

        // Generate PDF certificate
        PageReference pdfPage = Page.CertificatePDF; // Visualforce page name
        pdfPage.getParameters().put('id', leadId);
        Blob pdfBlob = pdfPage.getContentAsPDF();

        // Prepare email
        Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
        email.setToAddresses(new String[] {lead.Email});
        email.setSubject('Certificate of Completion');
        email.setHtmlBody('Dear ' + lead.Name + ',<br><br>Congratulations on completing the
course <strong>' + lead.Course_Name__c + '</strong> on ' + lead.Completion_Date__c +
'.<br>'+ 'Please find your certificate attached.<br><br>Best regards,<br>Course Team');

        email.setFileAttachments(new Messaging.EmailFileAttachment[] {
            new Messaging.EmailFileAttachment(
                'Certificate_' + lead.Name + '.pdf',
                pdfBlob
            )
        });

        // Send email
        Messaging.sendEmail(new Messaging.SingleEmailMessage[] {email});
    }
}
```

4. Batch Processing (Optional for Multiple Leads):

- Use Batch Apex to generate and send certificates for multiple leads at once.

```
public class CertificateBatch implements Database.Batchable<SObject> {
    public Database.QueryLocator start(Database.BatchableContext BC) {
        return Database.getQueryLocator('SELECT Id FROM Lead WHERE Certificate_Sent__c
= false AND Completion_Date__c != NULL');
    }
    public void execute(Database.BatchableContext BC, List<Lead> scope) {
        for (Lead lead : scope) {
```

```
                CertificateEmailService.sendCertificate(lead.Id);
                lead.Certificate_Sent__c = true;
            }
            update scope;
        }
        public void finish(Database.BatchableContext BC) {
            // Notify admin or log completion
        }
    }
```

5. **Schedule the Batch Job:**
   - Schedule the batch job to run daily or weekly.

Step 4: Create and Use Email Templates

- **Email Template:**
  - Use Salesforce Classic or Lightning email templates with merge fields for dynamic content.
  - **Example content:**

    Subject: Certificate of Completion
    Body:
    Congratulations {!Lead.Name},

    You have successfully completed the course {!Lead.Course_Name__c} on {!Lead.Completion_Date__c}.
    Your certificate is attached.

    Regards,
    The iLearnings Team

Step 5: Test and Deploy

1. **Test Individual Emails:**
   - Send a certificate email for a single lead to ensure data is correct and the attachment is included.
2. **Test Bulk Operations:**
   - Run batch processes or flows to verify certificates are sent in bulk.
3. **Monitor and Troubleshoot:**
   - Use email logs to track email delivery.
   - Add debug logs in Apex for error handling.

This approach ensures certificates are automatically sent to leads upon course completion with minimal manual effort.