KALLAM HARANADHAREDDY INSTITUTE OF TECHNOLOGY

(An Autonomous Institution)
Chowdavaram,Guntur-522002

DEPARTMENT OF:

ELECTRONICS AND COMMUNICATION ENGINEERING

BACHELOR OF TECHNOLOGY
2021 – 2025

VIII SEMESTER

LEASE MANAGEMENT

TEAM MEMBERS:

Sk.Riyaz Jani - 218X1A04F7

V.Sirisha - 228X5A0417

LEASE MANAGEMENT

1. Project Overview:

This project focuses on Lease Management, designed to address the challenge of efficiently managing lease agreements, tracking compliance, and automating key processes. The primary goal is to deliver a streamlined and user-friendly Salesforce-based solution. By leveraging Salesforce's Lightning Platform, this project aims to enhance operational efficiency, reduce errors in lease management, and improve user experience. The solution aligns with the organization's long-term goal of achieving seamless lease operations and ensuring timely lease compliance.

2. Objectives:

Business Goals

1. **Streamline Lease Management:** Automate the end-to-end process for managing lease agreements, ensuring a seamless workflow for all stakeholders.
2. **Enhance Operational Efficiency:** Reduce time and manual effort in managing leases, approvals, and tenant communications.
3. **Ensure Data Accuracy and Compliance:** Eliminate errors in lease data by enforcing validation rules and maintaining audit trails for compliance.
4. **Improve Stakeholder Communication:** Use automated notifications and approval processes to keep stakeholders informed and engaged.
5. **Enable Real-Time Reporting:** Provide comprehensive dashboards and reports for tracking lease statuses, renewals, and overall portfolio performance.

Specific Outcomes

1. **Custom Salesforce Objects:**
   - Define objects for *Leases, Properties, and Tenants* to store all relevant information.
2. **Automated Workflows:**
   - Build Flows to handle lease renewals, reminders, and escalations without manual intervention.
3. **Validation Rules and Business Logic:**
   - Enforce rules such as checking lease dates and ensuring unique entries for each lease agreement.
4. **Approval Processes:**
   - Implement a multi-level approval process involving property managers and legal teams to streamline decision-making.

5. **Dynamic Email Templates:**
    - Create templates for lease expiration reminders, renewal offers, and approval notifications.
6. **Dashboard and Reporting:**
    - Provide interactive dashboards to track key metrics, including the number of active leases, upcoming expirations, and approval statuses.
7. **Code and Integration Enhancements:**
    - Develop Apex triggers for custom logic and Schedule Classes for time-based automations, ensuring smooth operations at scale.

3. Salesforce Key Features and Concepts Utilized:

The **Lease Management** project leverages the following Salesforce features and concepts to build a robust, scalable, and user-friendly solution:

1. Custom Objects

- **Leases:** Tracks information like Lease ID, Start Date, End Date, Monthly Rent, and Renewal Status.
- **Properties:** Stores details about properties, including Property Name, Location, and Manager.
- **Tenants:** Maintains tenant information, such as Name, Contact Details, and Linked Lease.

2. Tabs

- Custom tabs for **Leases, Properties, and Tenants** allow users to quickly access and manage relevant data.
- Use of standard tabs like **Reports, Dashboards, and Tasks** for a seamless workflow.

3. Lightning App Builder

- Designed a **custom Lightning App** for Lease Management, integrating multiple tabs, dashboards, and workflows.
- Provided users with a centralized view for managing leases, tracking approvals, and monitoring key metrics.

4. Fields and Validation Rules

- **Fields:**
  - Custom fields like Lease Term (calculated), Renewal Due Date, and Property Manager Email.
- **Validation Rules:**
  - Ensure Start Date is earlier than End Date.
  - Prevent duplicate Lease IDs.
  - Validate that Monthly Rent is a positive value.

5. Email Templates

- Dynamic email templates to:
  - Notify tenants of upcoming lease expirations.
  - Alert property managers when a new lease is pending approval.
  - Send confirmation emails after lease approvals.

6. Approval Process

- Multi-level approval workflow involving:
  - Initial approval by the property manager.
  - Final approval by the legal department.
- Automated notifications for pending and approved steps.

7. Flows

- **Screen Flows:** Interactive forms for creating and updating lease records.
- **Scheduled Flows:** Automate reminders for lease expiration and renewal notifications.
- **Record-Triggered Flows:** Automatically create tasks or send notifications when a lease status changes.

8. Apex Triggers

- Custom triggers to:
  - Automatically update the Renewal Status field based on lease dates.
  - Prevent the deletion of leases tied to active tenants.
  - Calculate penalties for late renewals.

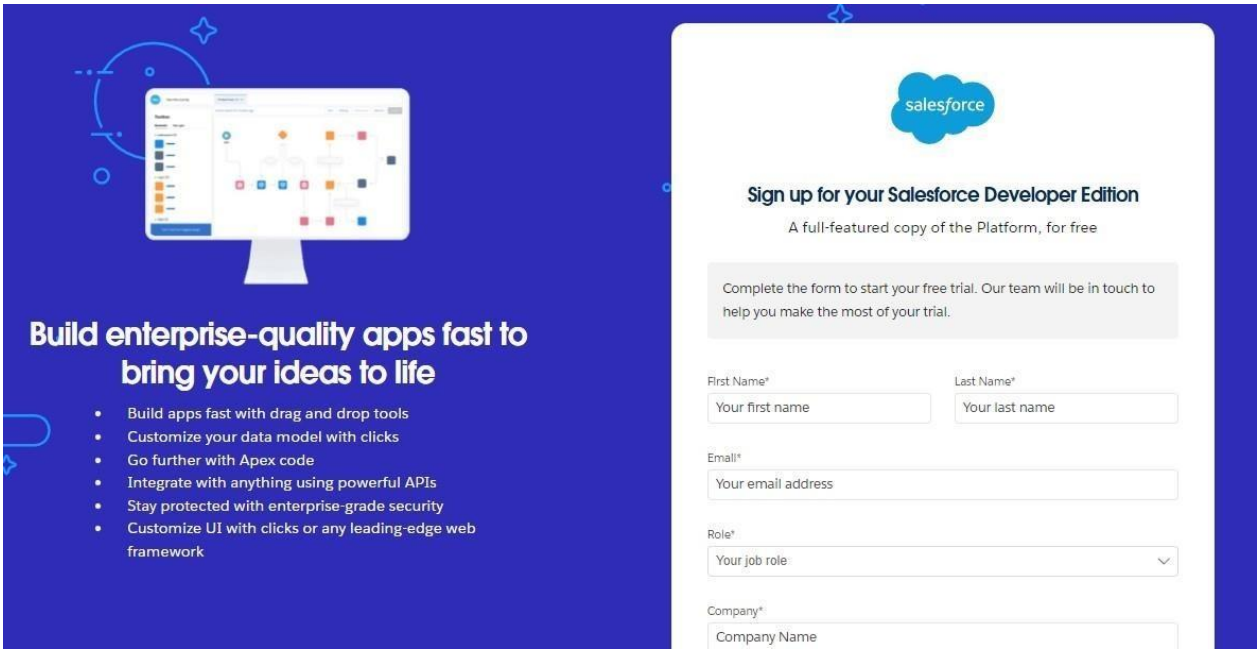
9. Schedule Class

- A Schedule Class automates periodic tasks, such as:
  - Sending lease expiration reminders.
  - Generating monthly performance reports.

4. Detailed Steps to Solution Design:

1. Creating Developer Account:

- Creating a developer org in salesforce.
- Go to https://developer.salesforce.com/signup

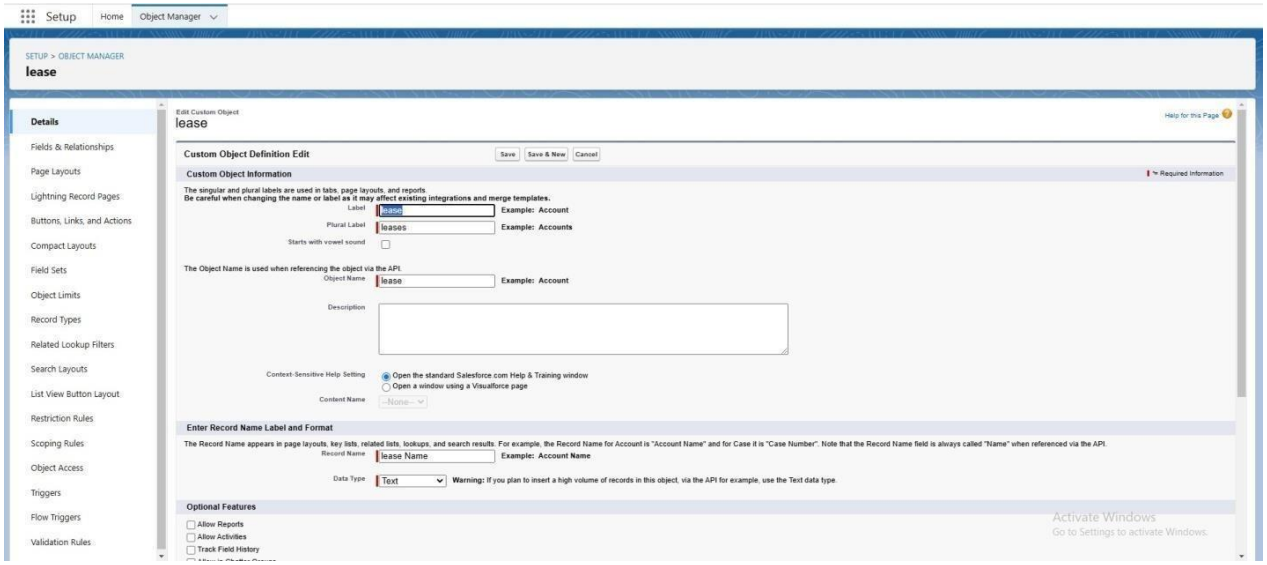


2. Creating objects:

1. Lease Object

Steps to Create

1. **Go to Setup → Object Manager → Create → Custom Object**.
2. **Data Type**: Text
3. **Object Name**: Lease
4. **Plural Label**: Leases
5. **Record Name Field**: Lease ID (Auto-Number)
    - **Display Format**: L-{0000}
6. **Optional Settings**:
    - Allow Activities
    - Track Field History
    - Allow Search

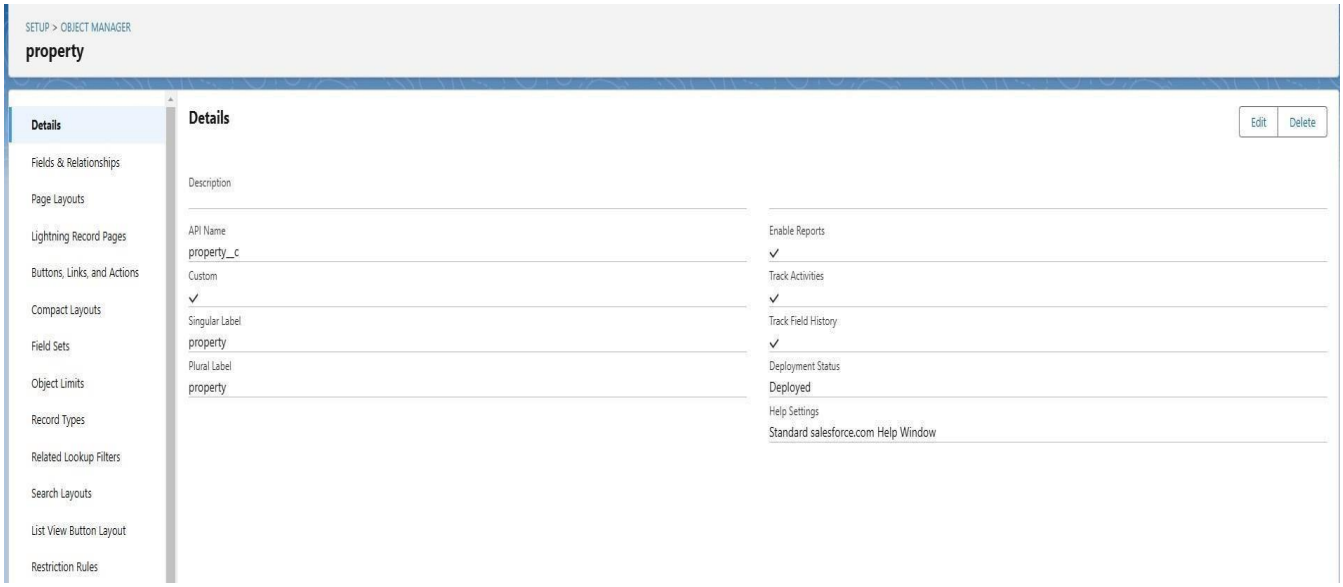


2. Property Object

Steps to Create

1. **Go to Setup → Object Manager → Create → Custom Object**.

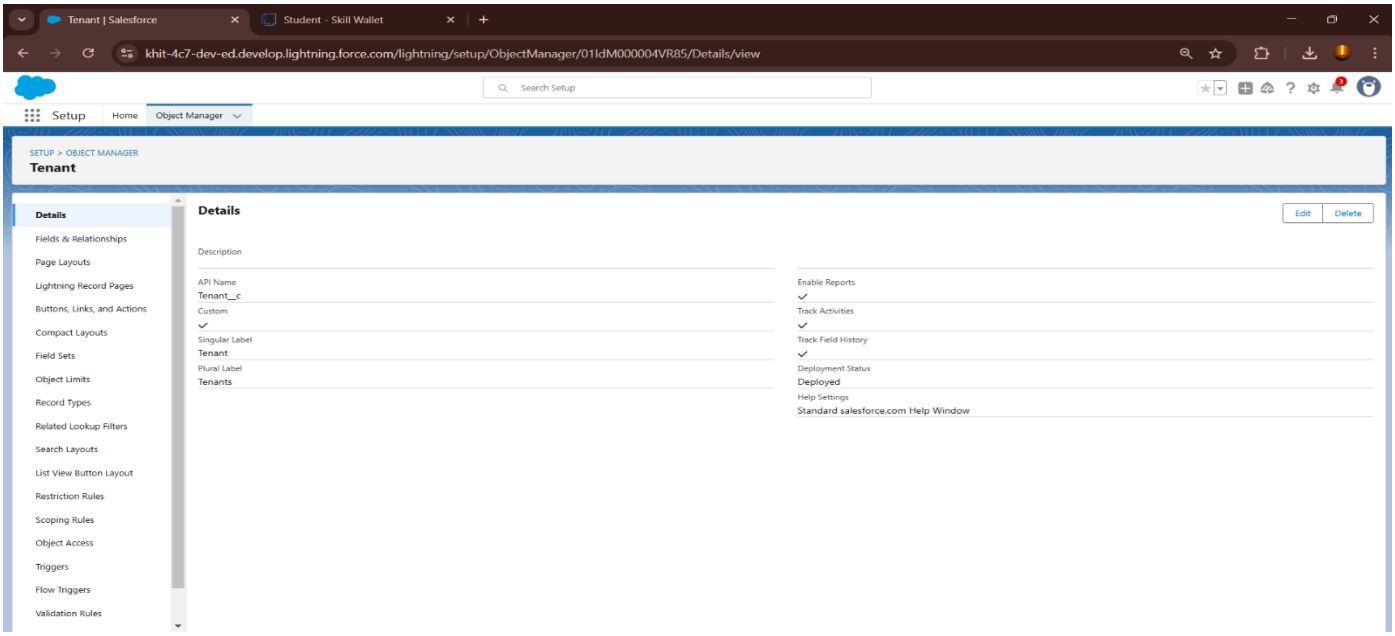
2. **Data Type:** Text
3. **Object Name**: Property
4. **Plural Label**: Properties
5. **Record Name Field**: Property Name (Text)
6. **Optional Settings**:
    - Allow Activities
    - Track Field History
    - Allow Search
7. Click on Save.



3. Tenant Object

Steps to Create

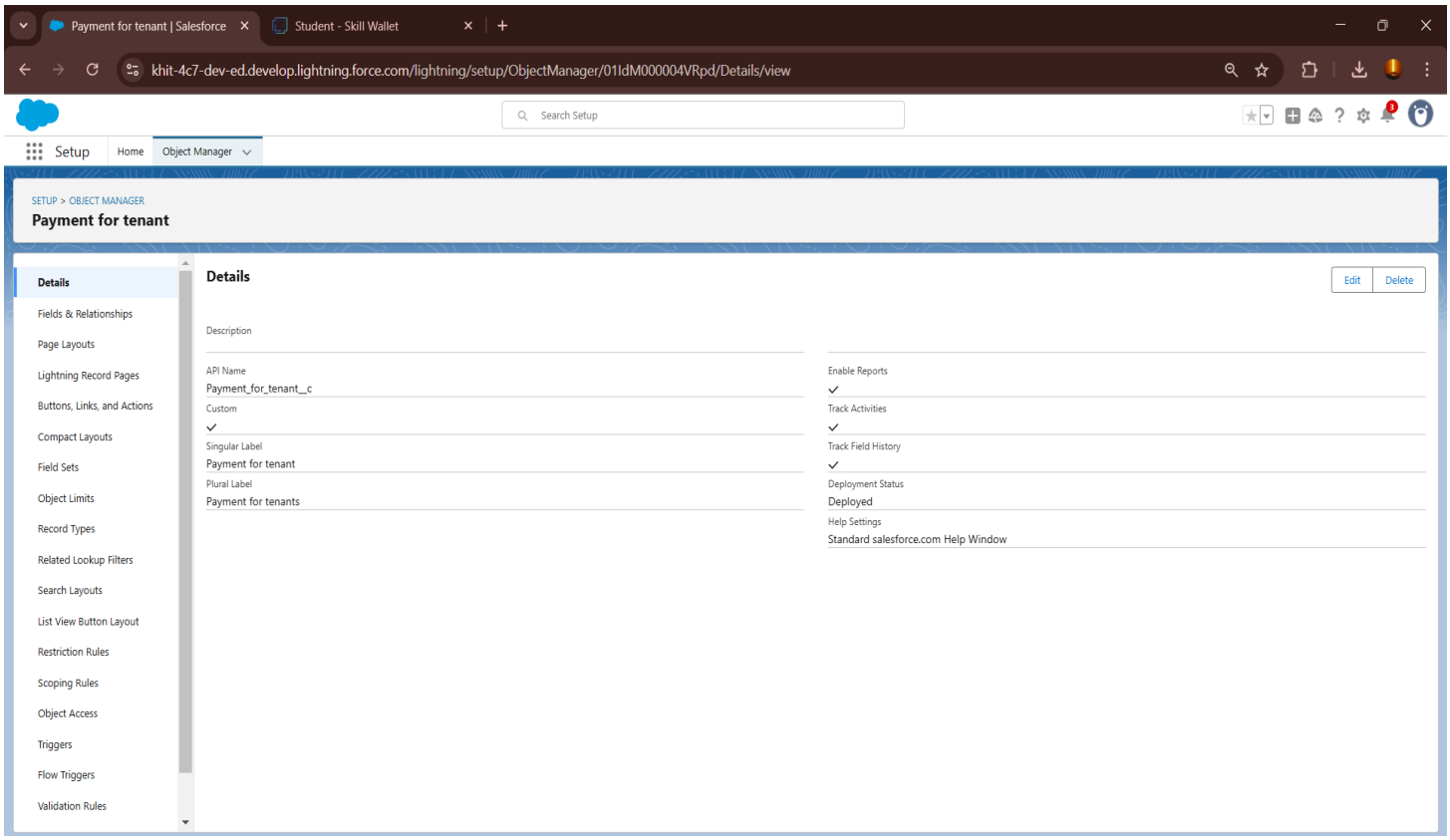
1. **Go to Setup → Object Manager → Create → Custom Object**.
2. **Object Name**: Tenant
3. **Plural Label**: Tenants
4. **Record Name Field**: Tenant Name (Text)
5. **Optional Settings**:
    - Allow Activities
    - Track Field History
6. Allow Search
7. Click on Save.



4. Payment Object

Steps to Create

1. **Go to Setup → Object Manager → Create → Custom Object**.
2. **Object Name**: Payment for tenant
3. **Plural Label**: Payment for tenants
4. **Record Name Field**: Payment Name (Text)
5. **Optional Settings**:
    i. Allow Activities
    ii. Track Field History
6. Allow Search
7. Click on Save.



Relationships Setup

1. **One-to-Many (Property → Leases)**
    - Add a Lookup relationship on the **Lease** object pointing to the **Property** object.
2. **One-to-One (Tenant → Lease)**
    - Add a Lookup relationship on the **Tenant** object pointing to the **Lease** object.

8. Tab Creation Purpose in Salesforce

Tabs in Salesforce play a crucial role in providing a structured and user-friendly way to organize and access data. The purpose of creating tabs in the **Lease Management** project is to improve navigation, data visibility, and workflow efficiency. Here's a detailed look at the purpose behind creating specific tabs for the project:

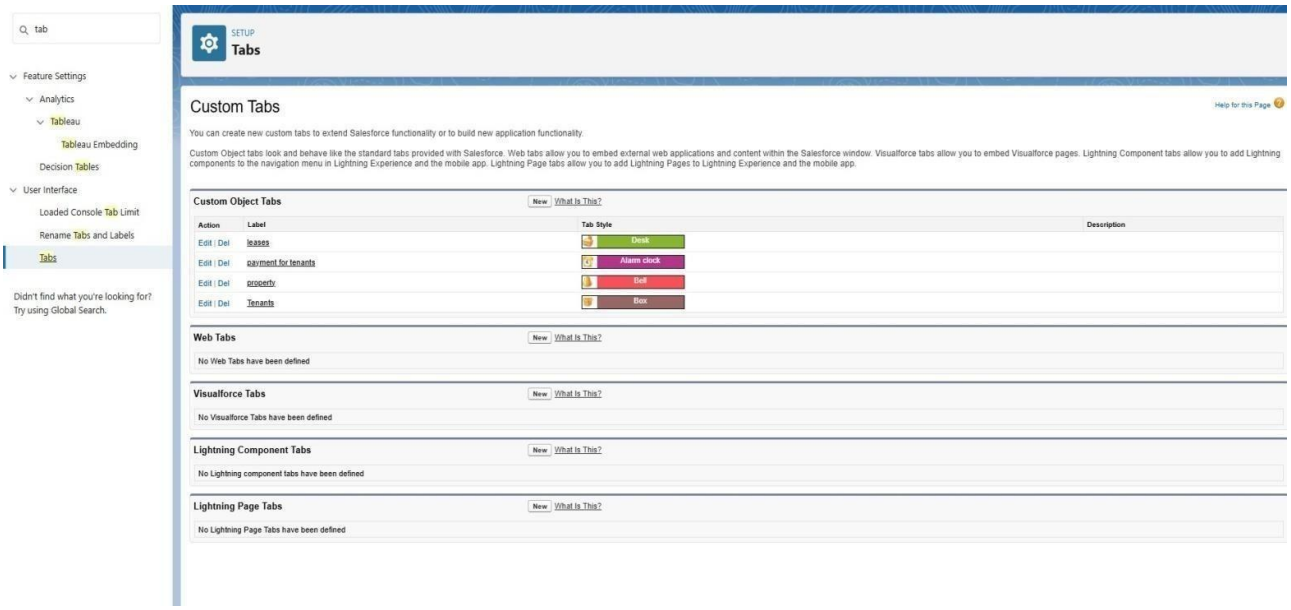
1. Lease Tab

Purpose:

- **Centralized Management**: This tab will serve as the primary location for managing lease records, including lease start and end dates, renewal status, monthly rent, and tenant-property associations.
- **Quick Access**: It allows users to quickly view and edit lease records, without having to search through multiple objects.
- **Efficient Filtering**: Users can filter leases by status (e.g., Active, Pending Renewal, Expired) to easily focus on relevant data.
- **Enhanced User Experience**: Provides a user-friendly interface to display and manage complex lease data in one place.

Benefits:

- Users can track lease statuses in real-time.
- Simplifies lease renewal and termination processes.
- Enables quick updates to lease terms and rent amounts.



1. Go to setup page <> type Tabs in Quick Find bar <> click on tabs <> New (under custom object tab)
Select Object(lease) <> Select the tab style <> Next (Add to profiles page) keep it as default <> Next uncheck the include tab .
Make sure that the Append tab to users' existing personal customizations is checked.
Click save.

2. Tenant Tab

Purpose:

- **Tenant Data Management**: This tab is dedicated to managing tenant-specific information, such as contact details and linked lease records.
- **Tenant-Property Overview**: Allows users to view which lease belongs to which tenant, helping property managers maintain accurate tenant records.
- **Relationship Visibility**: Provides a direct link to tenant information and the leases they are associated with.

Benefits:

- Centralizes tenant information in one place for easier management.
- Provides a clear view of tenant history and leasing relationships.
- Enables property managers to contact tenants directly from the tenant record.

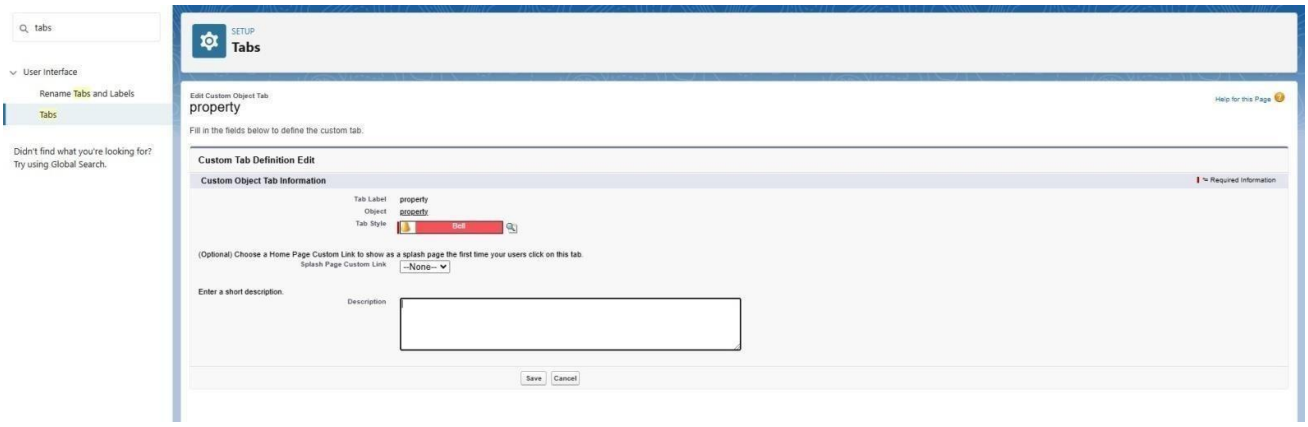
3. Property Tab

Purpose:

- **Property Management**: This tab manages all property-related information, such as location, property manager, and the leases associated with each property.
- **Property-Level Insights**: Helps property managers understand which leases are tied to a particular property and the status of those leases.
- **Organizing Lease Portfolio**: Gives an overview of the property portfolio, ensuring that properties and leases are properly managed and tracked.

Benefits:

- Provides an overview of lease activities for each property.
- Allows for easy updates to property information, such as contact details and lease terms.
- Facilitates reporting on property performance and lease statuses.



4. Payment for tenant tab

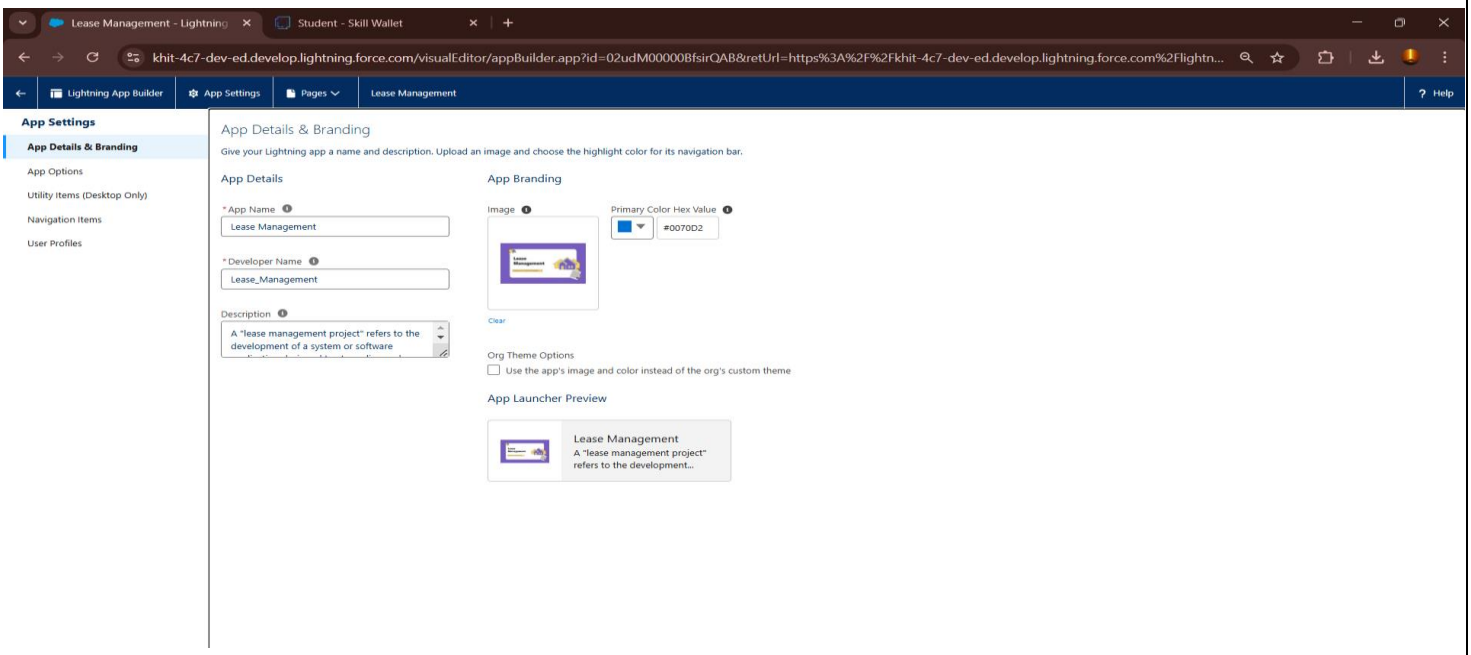
Create payment for tenant tab same as the steps given in lease tab creation. Select object(payment for tenant).

5. Lightning App Builder Design:

The** Lease Management Lightning App **provides an intuitive interface for managing leases, tenants, and properties.

Steps to Create the App

1. **Go to Setup → App Manager → New Lightning App**.
2. **App Settings**:
    - **App Name**: Lease Management
    - **Navigation Style**: Standard Navigation
    - **App Options**:
        - Assign a custom logo.
        - Enable app personalization for users.



6. Field Creation in Salesforce

Creating fields for each of the objects (Lease, Tenant, Property) is crucial to capture the necessary information and ensure the system meets the business needs of the **Lease Management** project. Below are the steps and detailed field creation for each object:

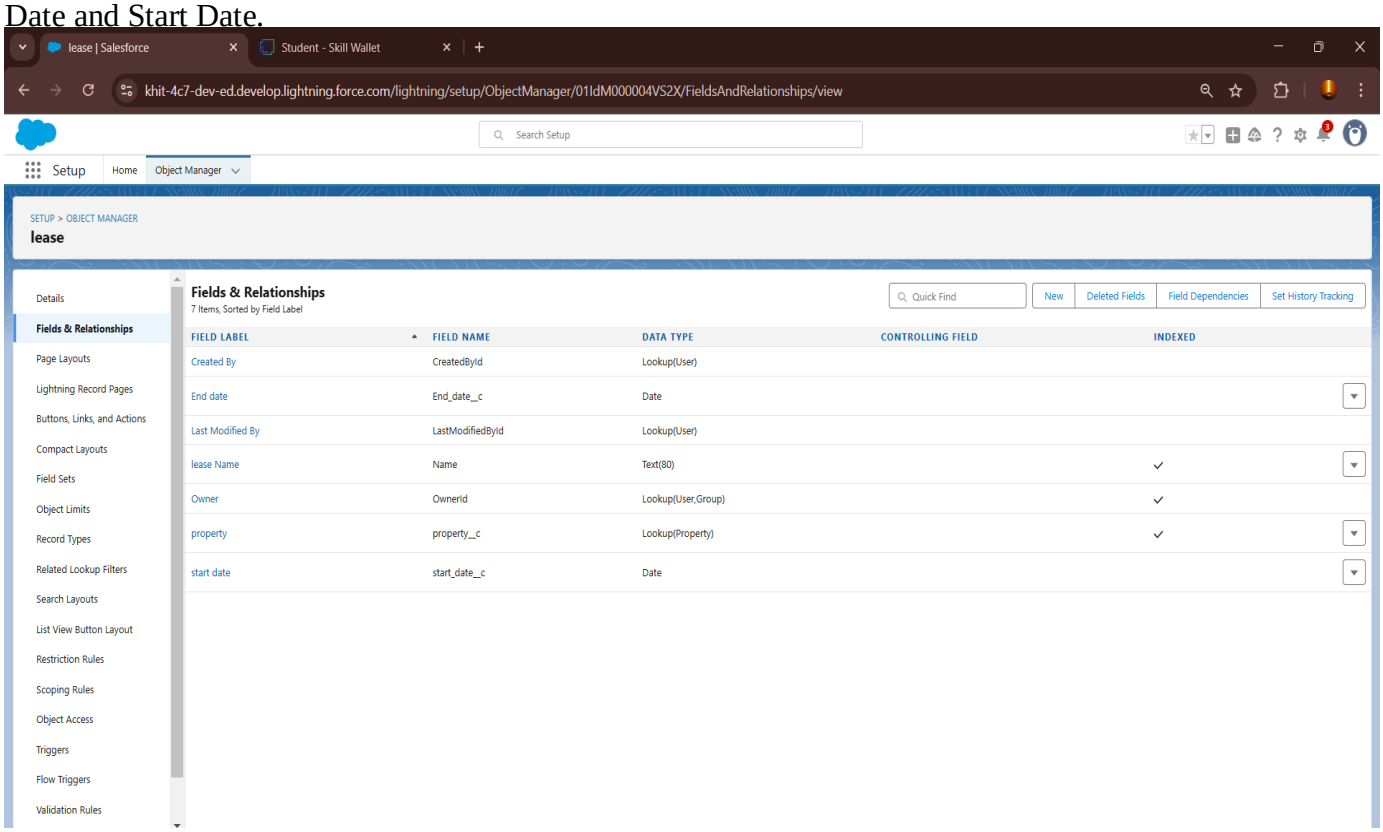
1. Lease Object Fields

Step-by-Step Field Creation for Lease Object:

1. **Go to Setup → Object Manager → Lease → Fields & Relationships → New**.
2. Select Data type as a “Date” and Click on Next
3. Field Label : start date
4. Field Name : gets auto generated
5. Click on Next >> Next >> Save and new. Repeat the same steps and create another field label‘end date’.

Field Type Details:

- **Auto-Number**: Automatically generates a unique identifier, e.g., "L-0001."
- **Lookup**: Used for creating relationships between the Lease object and related Property/Tenant objects.
- **Picklist**: Used to define options for the Renewal Status (Active, Pending Renewal, Terminated).
- **Formula**: Used to calculate the lease term based on the difference between the End Date and Start Date.



2. Tenant Object Fields

Step-by-Step Field Creation for Tenant Object:

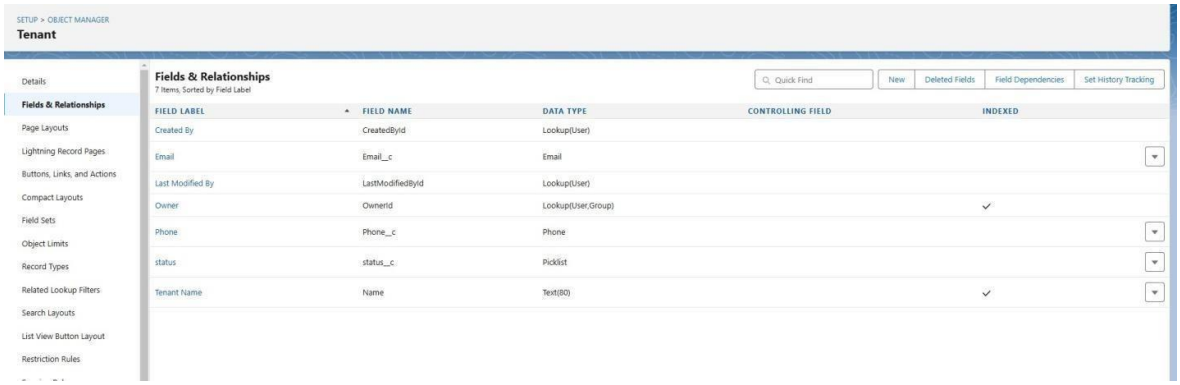
1. **Go to Setup → Object Manager → Tenant → Fields & Relationships → New**.
2. **Choose Field Type** (as per the below descriptions).

Fields to Create:

Field Name	Data Type	Description
Tenant Name	Text	Name of the tenant.
Contact Email	Email	Email address of the tenant.
Contact Phone	Phone	Phone number of the tenant.
Lease ID	Lookup (Lease)	Links the tenant to a specific lease.
Tenant Type	Picklist	Type of tenant (Individual, Company, etc.).
Date of Birth	Date	Date of birth for individual tenants.
Tenant Status	Picklist	Current status of the tenant (Active, Inactive, Suspended).

Field Type Details:

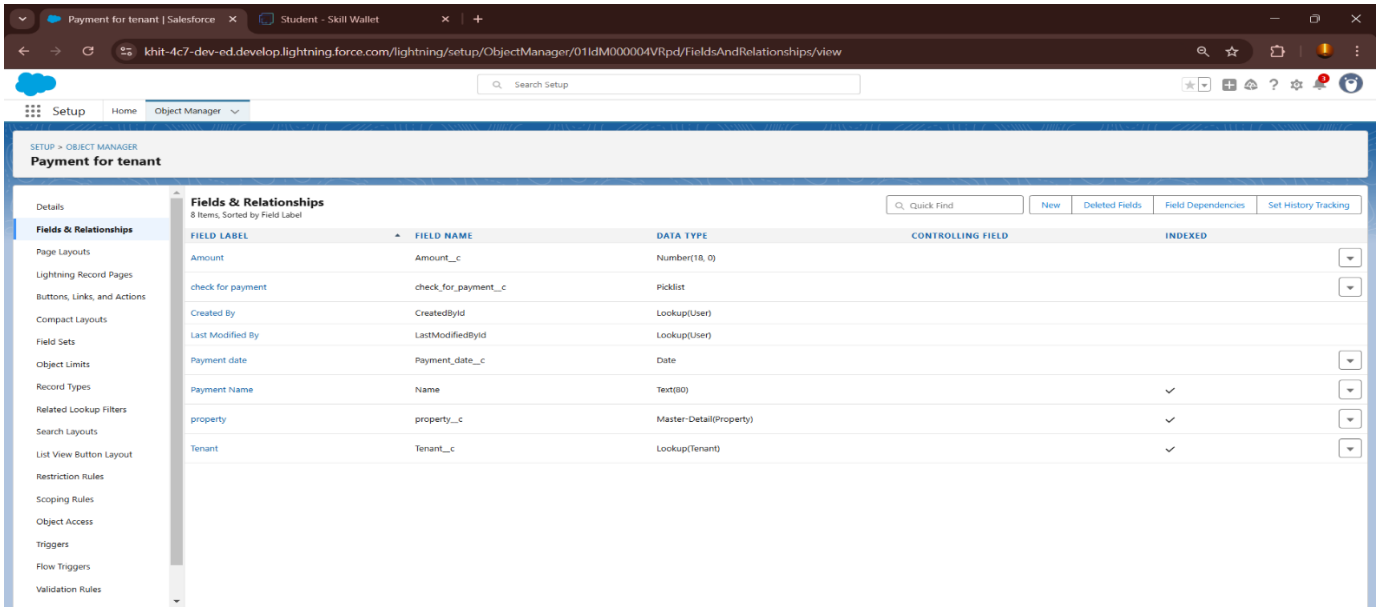
- **Lookup**: Used to link the Tenant record to a specific Lease.
- **Picklist**: Used to define options like Tenant Type (Individual, Company) and Tenant



3. Payment for tenant object fields

1. **Go to setup >> click on Object Manager >> type object name(Payment for tenant) in search bar >> click on the object.**
2. **Now click on "Fields & Relationships" >> New**
3. **Select Data type as a "Date" and Click on Next**
4. **Fill the Above as following:**

- **Field Label : Payment date, Data type : Date**
- **Field Label : Amount, Data type : Number, length : 18**
- **Field Label : check for payment, Enter values with each value separated by a new line Enter these values: Paid, Not paid**
- **Field Name : gets auto generated**
- **Click on Next >> Next >> Save.**



4. Property Object Fields

Step-by-Step Field Creation for Property Object:

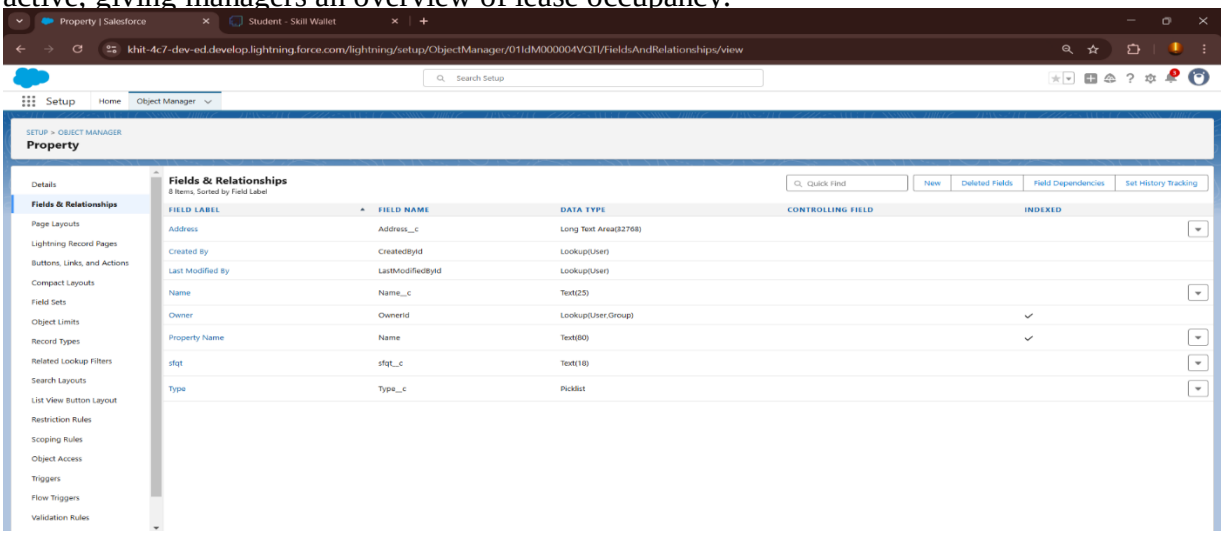
1. **Go to Setup → Object Manager → Property → Fields & Relationships → New**.
2. **Choose Field Type** (as per the below descriptions).

Fields to Create:

Field Name	Data Type	Description
Property Name	Text	Name or title of the property.
Address	Text Area	Full address of the property.
Property Manager	Lookup (User)	Relationship linking to the Property Manager (User object).
Number of Units	Number	Number of units available at the property.
Property Status	Picklist	Status of the property (Available, Under Maintenance, etc.).
Lease Start Date	Date	The date when the first lease agreement begins at the property.
Lease Expiry Date	Date	The date when the last lease at the property expires.
Total Active Leases	Roll-Up Summary	A summary field that counts all active leases related to the property.

Field Type Details:

- **Lookup**: Creates a relationship to the User object for Property Manager.
- **Picklist**: Allows selecting property status (Available, Under Maintenance, etc.).
- **Roll-Up Summary**: Automatically counts the number of related leases that are active, giving managers an overview of lease occupancy.

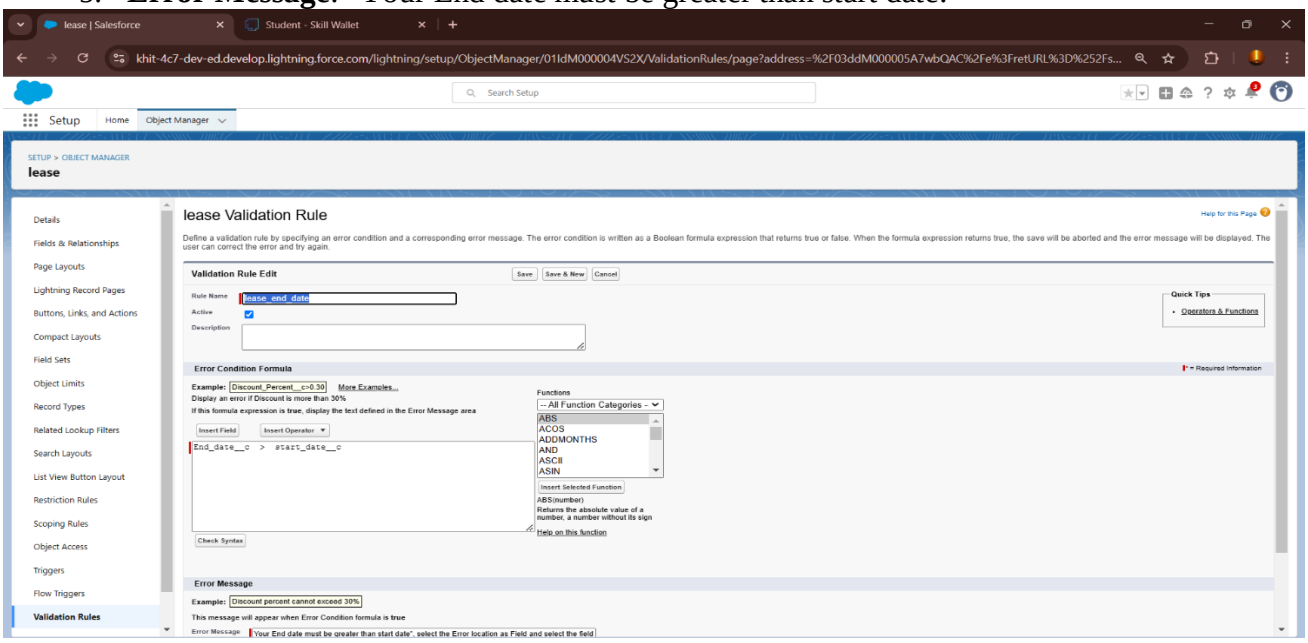


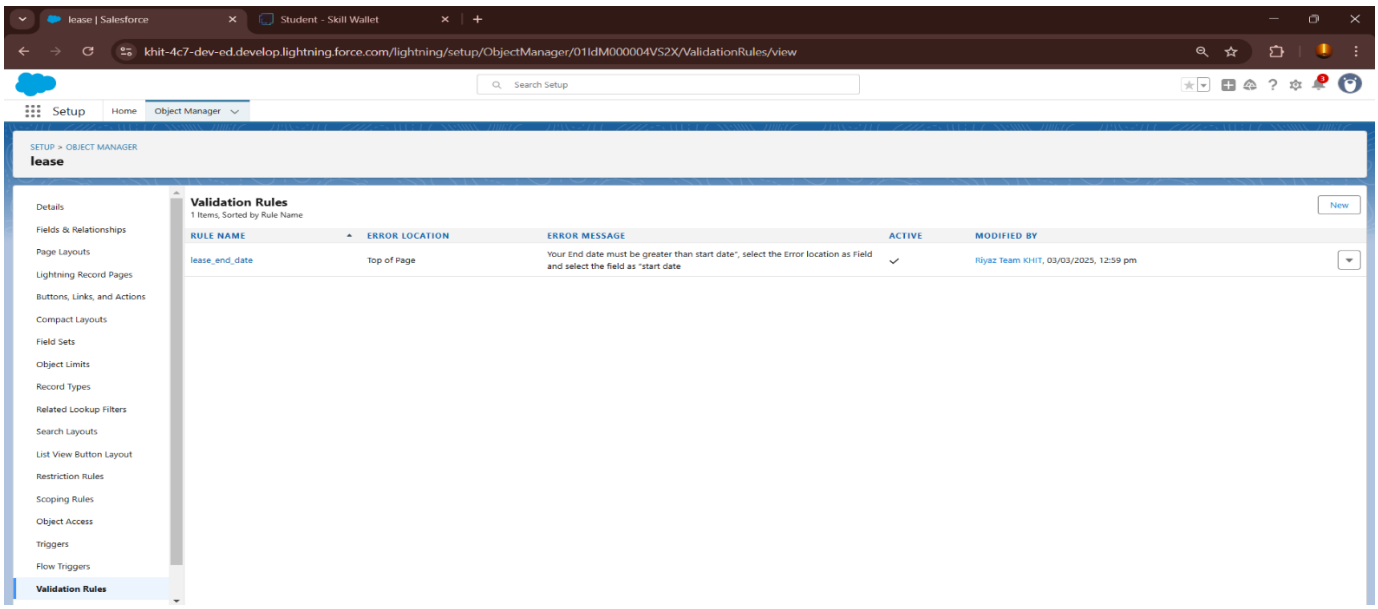
Field Validation Example

You can set **Validation Rules** to ensure data integrity. For instance:

End Date must be after Start Date (for Lease Object):

1. **Go to Setup → Object Manager → Lease → Validation Rules → New Rule**.
2. **Rule Name**: Lease End Date
3. **Formula**:
   plaintext
   Copy code
4. End_date__c > start_date__c
5. **Error Message**: "Your End date must be greater than start date."





5.1 Validation Rules:

1. End Date Validation

- **Formula**:

```
plaintext
Copy code
End_Date__c > Start_Date__c
```

- **Error Message**: "End Date must be after Start Date."

2. Positive Monthly Rent

- **Formula**:

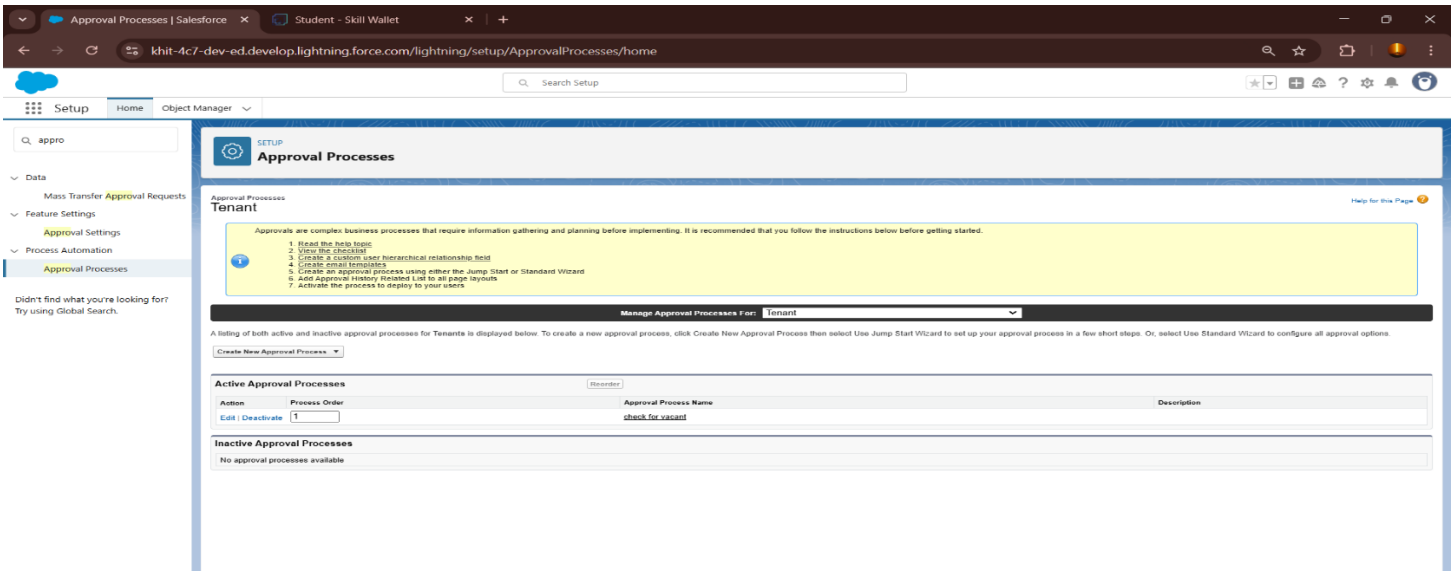
```
plaintext
Copy code
Monthly_Rent__c > 0
```

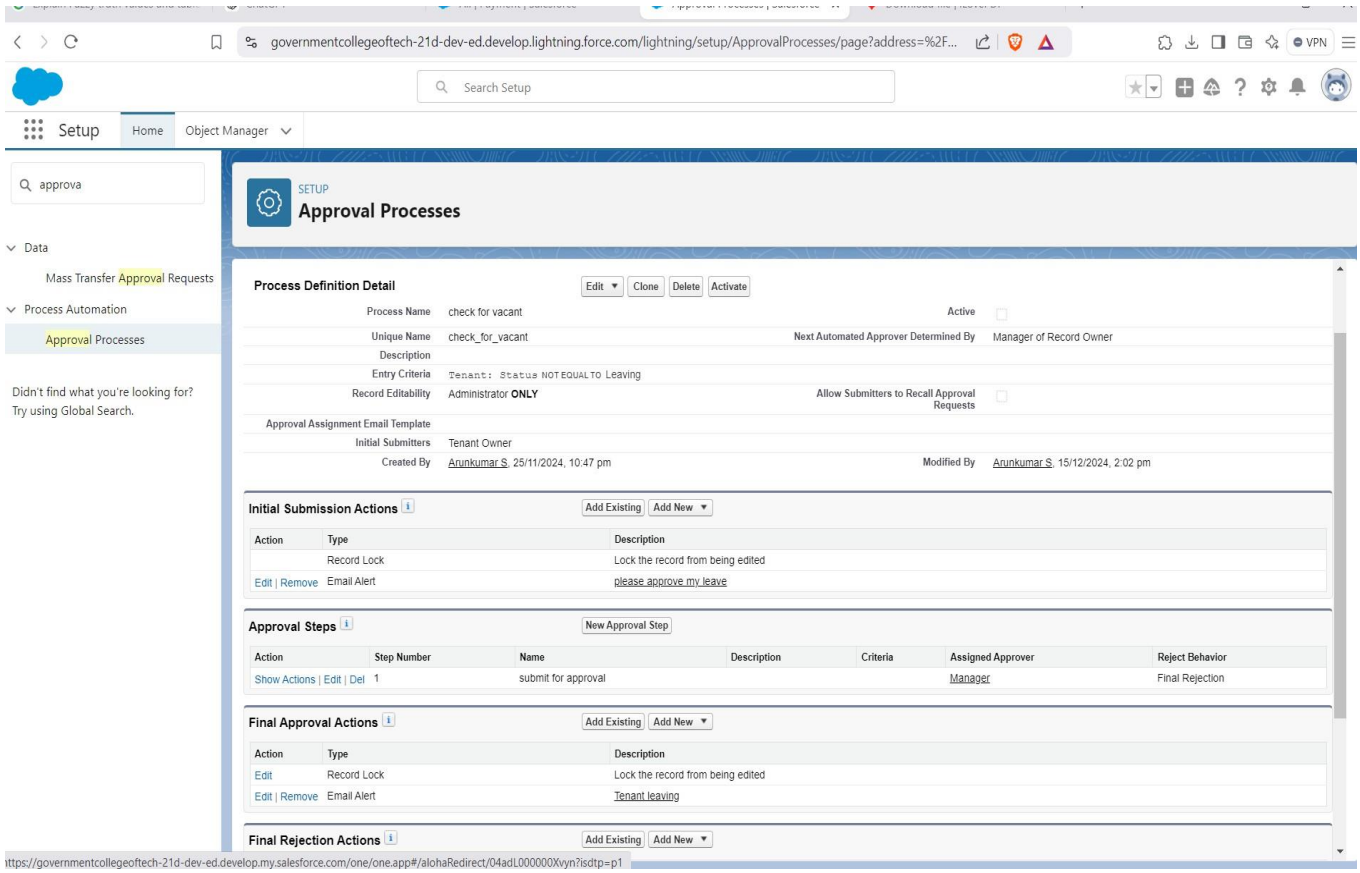
- **Error Message**: "Monthly Rent must be greater than zero."

6. Approval Process

Steps to Create

1. **.Go to setup >> Approval Processes in quick find bar>>click on it.**
2. Manage Approval Process For >> "Tenant" from the drop down.
3. **Click on "Create New Approval Process" >> Use standard setup wizard.**
4. Process Name "check for vacant" >> Click Next.
5. Field "Tenant:status" >> Operator : Not equals , Value >> Click on the lookup filter icon and select "Leaving".
6. Select the "Administrators ONLY can edit records during the approval process".Then Next.
7. From the available fields select >> Tenant Name, and then add >>Add it to the selected.Then Next.
8. Submitter type Search>>Owner, Allowed Submitters>>Property Owner.Then Next.
9. Then click save.





1. Under initial submission action click on add new and then select email alert.
2. Description: “please approve my leave”.
3. unique name : auto populated
4. Email template : tenant leaving
5. Recipient type : Email field
6. Available Recipients : Email field : Email
7. From Email address : Current user’s email
8. Click save.

Same as the above steps, follow the steps to create final approval action and final rejection action.

Case(i):-1. Under Final approval action click on new and then select email alert.

2. Description: “Tenant leaving”.

- Email template : Leave approved
- Recipient type : Email field
- Available Recipients : Email field : Email
- From Email address : Current user’s email
- Click save

Case(ii):- Under final rejection action

Description: “your request for leave is rejected”**.**

Email template : leave rejected and Recipient type : Email field

Available Recipients : Email field : Email and Click save.

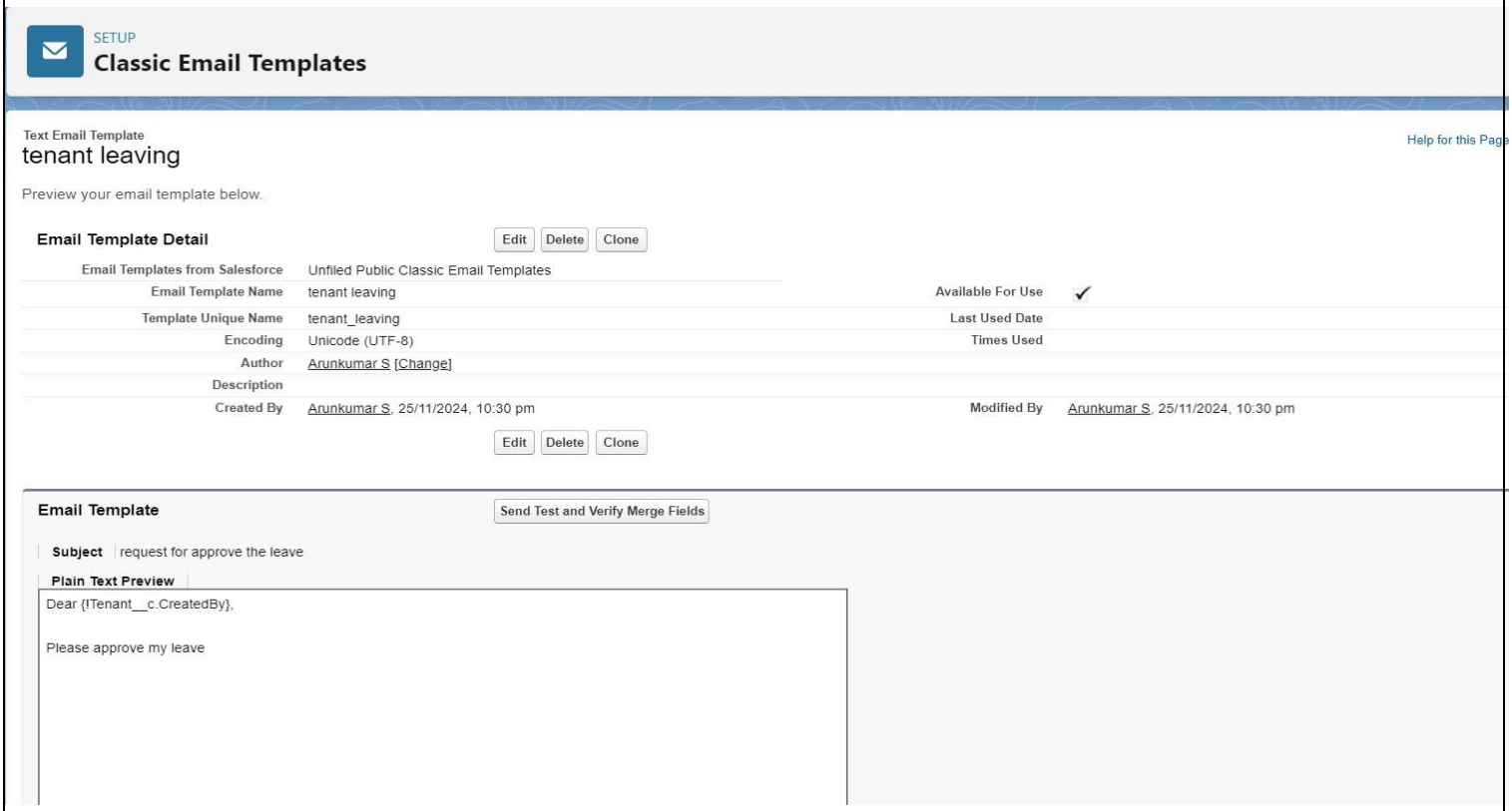
Email Templates:-

1)Email Alert for tenant leaving

- Go to setup in quick find box enter email template >> click on classic Email Template.
- Click on >> New Email Template===>Choose text and Click on available for use.
- Email Template Name is “tenant leaving” and Subject : ” request for approve the leave”
- Email body :

  Dear {!Tenant__c.CreatedBy},

  Please approve my leave
- Save



2) Create Email Template For Leave Approved

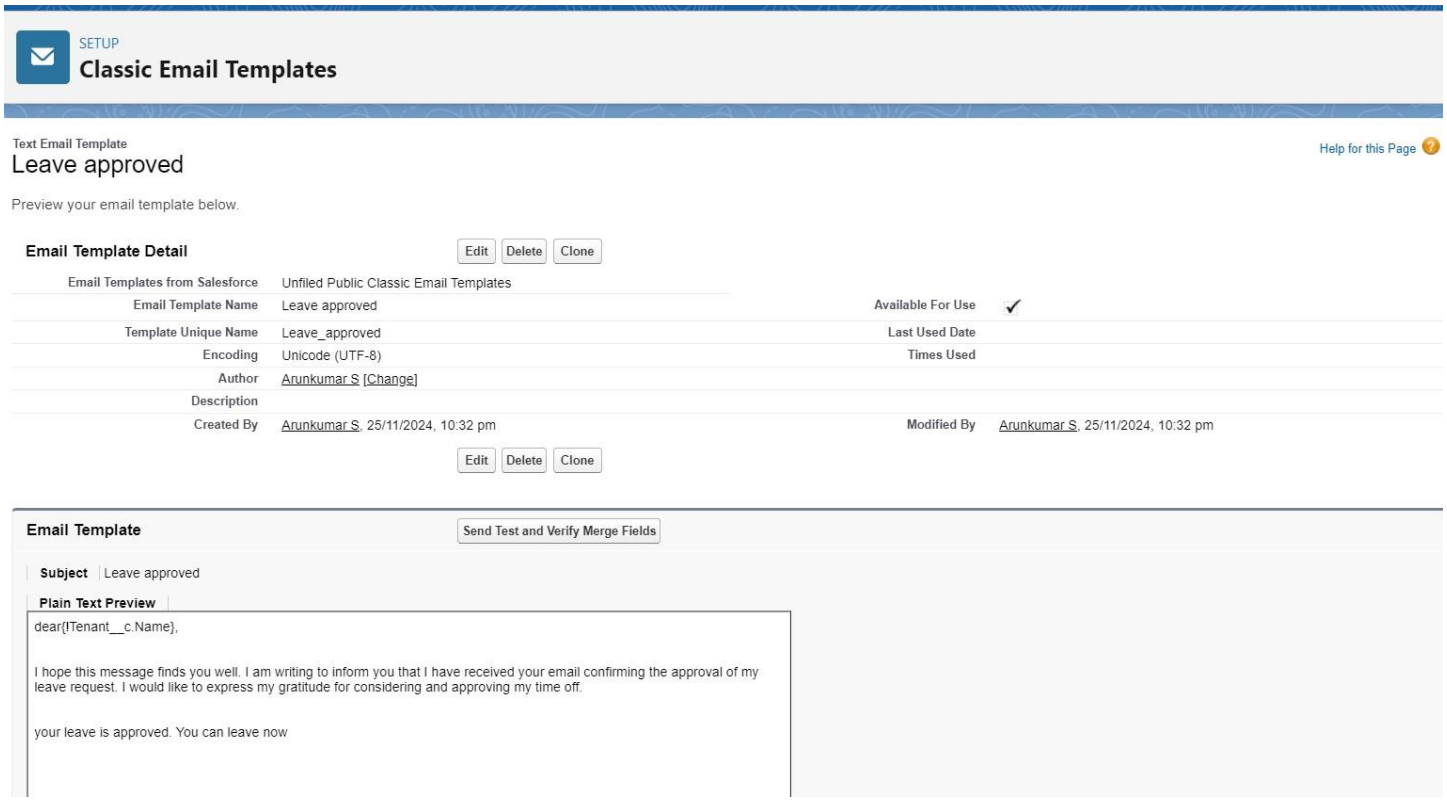
- Create and Follow the same first two steps in tenant leaving template
- Email Template Name is “Leave approved” and Subject : ” Leave approved”
- Email body :

  dear{!Tenant__c.Name},

I hope this message finds you well. I am writing to inform you that I have received your email confirming the approval of my leave request. I would like to express my gratitude for considering and approving my time off.

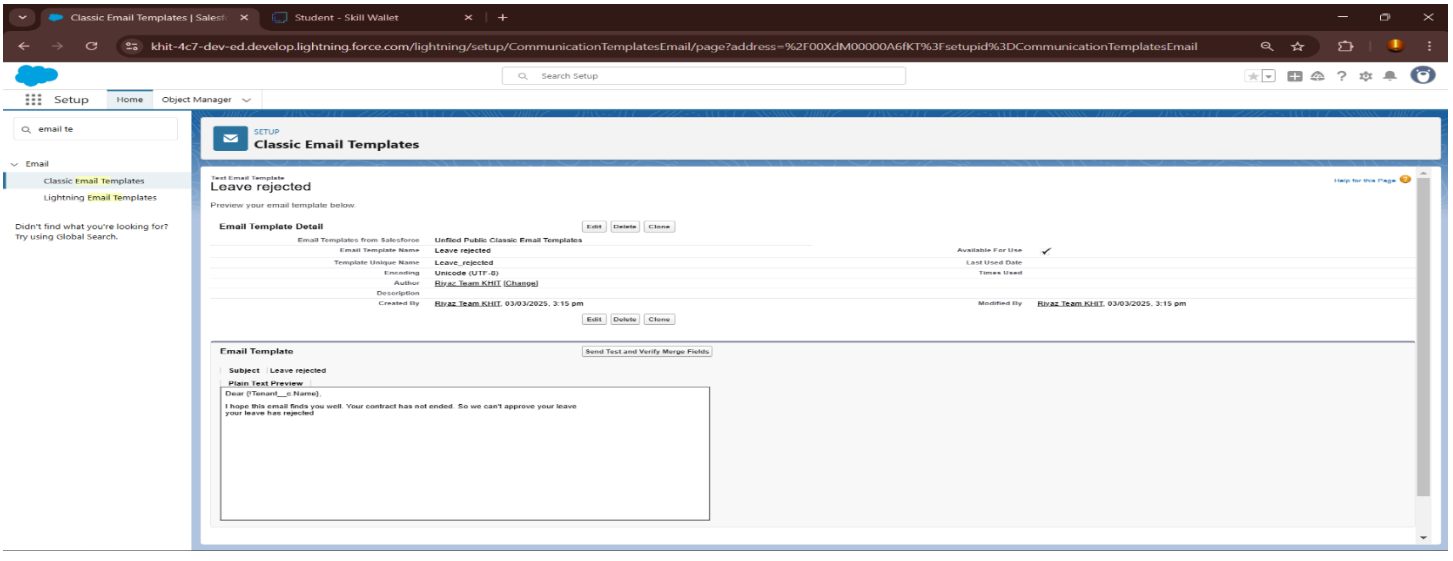
your leave is approved. You can leave now

- Click Save.



3)**Create Email Template For rejection for leave**

- Create and Follow the same first two steps in tenant leaving template
- Email Template Name is “Leave rejected” and Subject : ” Leave rejected”
- Email body :
  Dear {!Tenant__c.Name},
  I hope this email finds you well. Your contract has not ended. So we can't approve your leave
  your leave has rejected
    - Click Save.



4) Create Email Template For Monthly payment

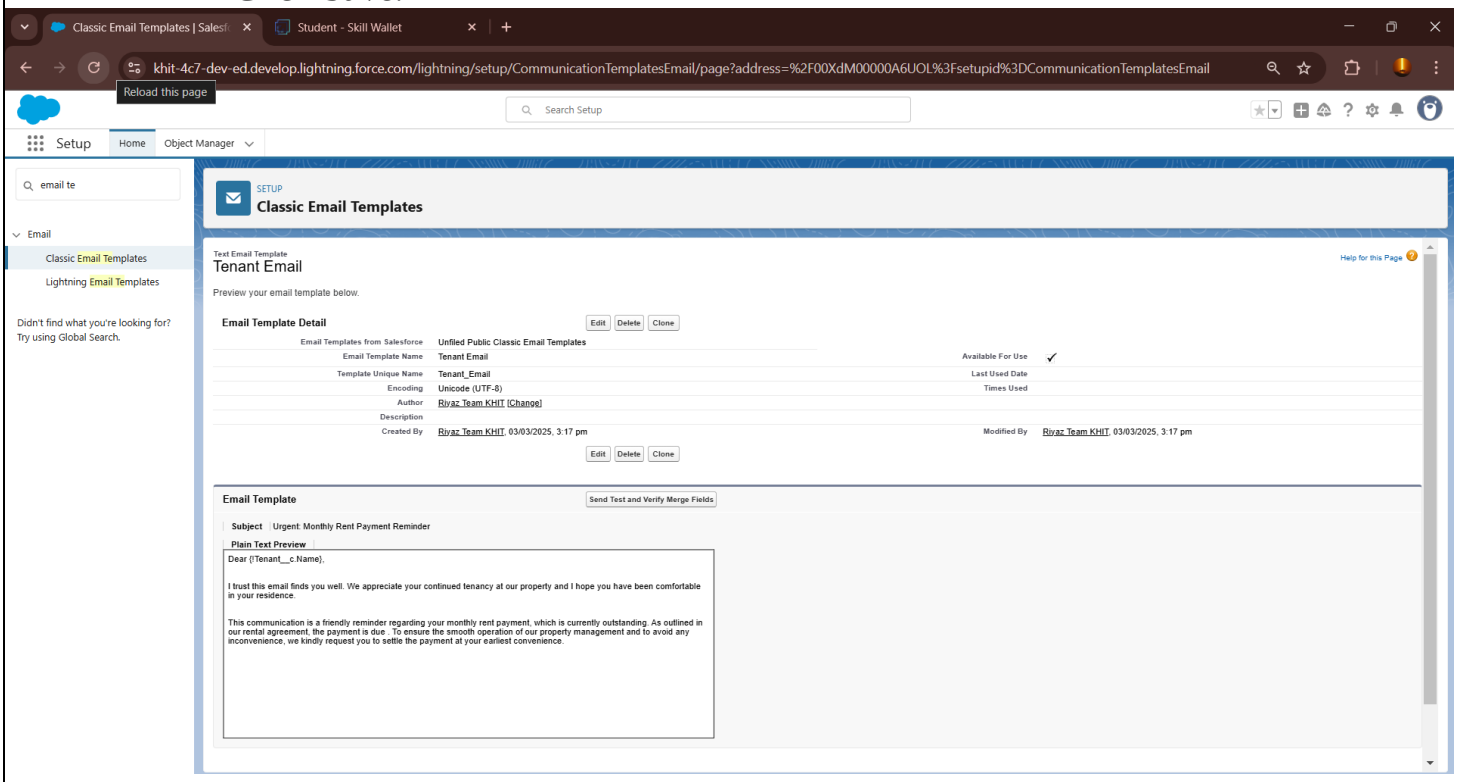
- Create and Follow the same first two steps in tenant leaving template
- Email Template Name is “Tenant Email” and
- Subject : ” Urgent: Monthly Rent Payment Reminder”
- Email body :

Dear {!Tenant__c.Name},

I trust this email finds you well. We appreciate your continued tenancy at our property and I hope you have been comfortable in your residence.

This communication is a friendly reminder regarding your monthly rent payment, which is currently outstanding. As outlined in our rental agreement, the payment is due . To ensure the smooth operation of our property management and to avoid any inconvenience, we kindly request you to settle the payment at your earliest convenience.

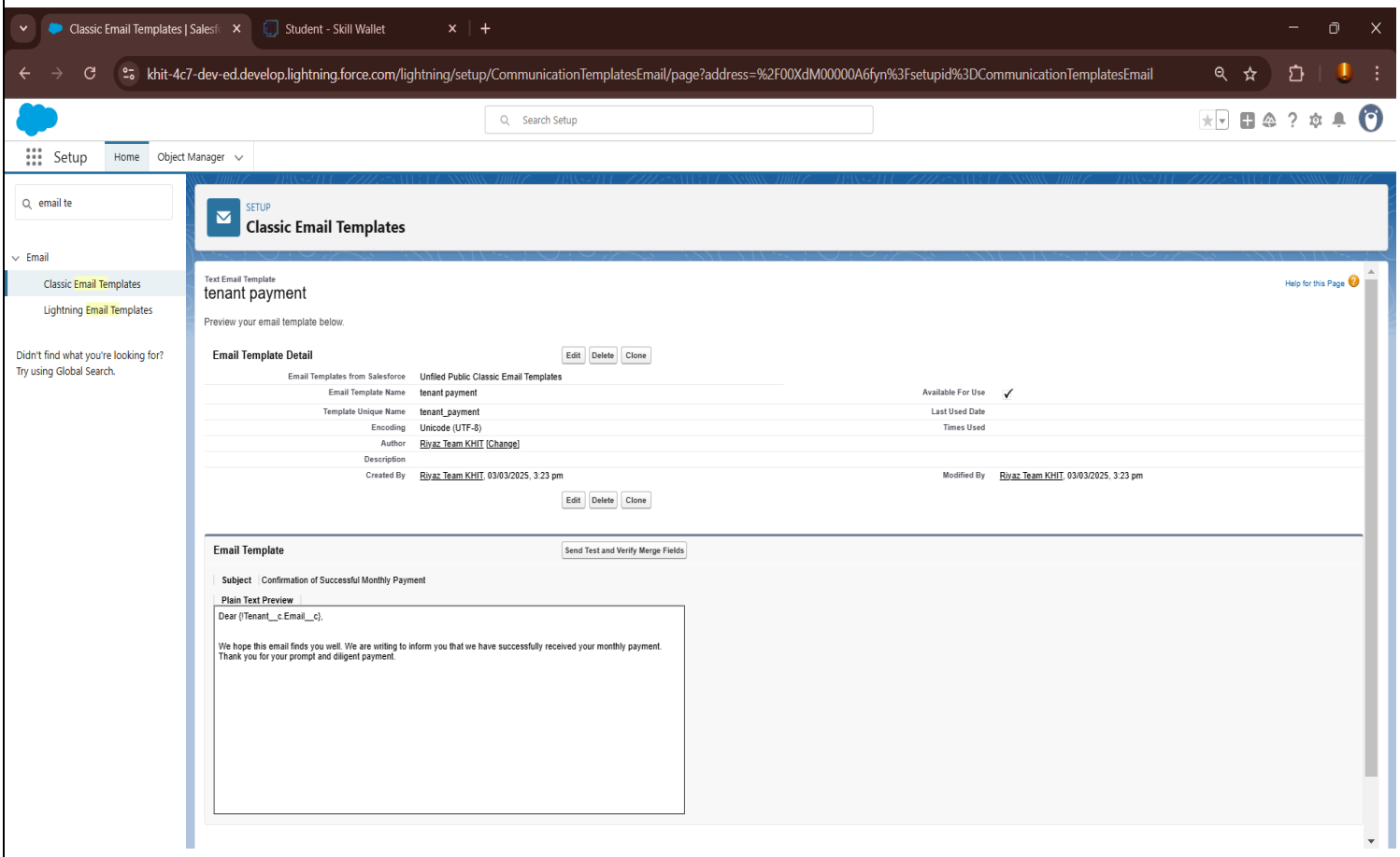
- Click Save.



5) Create Email Template For successful payment

- Create and Follow the same first two steps in tenant leaving template
- Email Template Name is “tenant payment”
- Subject : ” Confirmation of Successful Monthly Payment”
- Email body :
- Dear {!Tenant__c.Email__c},

We hope this email finds you well. We are writing to inform you that we have successfully received your monthly payment. Thank you for your prompt and diligent payment.



We use email templates to increase productivity and ensure consistent messaging. Email templates with merge fields let you quickly send emails that include field data from Salesforce records like contacts, leads, or opportunities. You can use email templates when emailing groups of people—with list email or mass email—or just one person.

Salesforce email templates are the easiest way to get your emails done. They help you create and send quick emails that include merge fields from Salesforce records like Contacts, Leads, Opportunities, or Custom Objects.

When you have a large number of contacts or leads in Salesforce, it can be difficult to keep track of who needs to be notified about new information. Salesforce email templates allow you to combine all these contacts or leads into one email and then send it out simultaneously.

7. Flows

1. Scheduled Flows

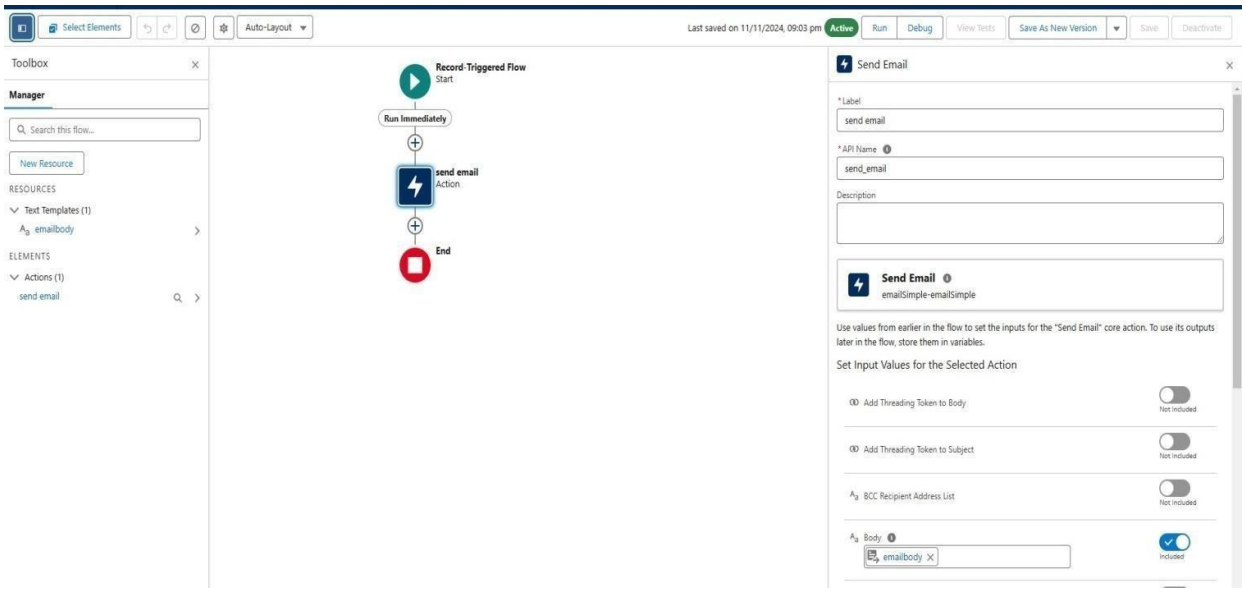
- **Purpose**: Notify tenants about lease expiration and automate renewal reminders.
- **Steps**:
  - Create a Flow with the trigger set to run daily.
  - Query leases expiring in the next 30 days.
  - Send an email notification using dynamic templates.

2. Screen Flows

- **Purpose**: Interactive form for creating or updating leases.
- **Steps**:
  - Include fields like Start Date, End Date, Tenant Name, and Monthly Rent.
  - Validate data dynamically before submission.

3. Record-Triggered Flows

- **Purpose**: Update Renewal Status when End Date is nearing.
- **Steps**:
  - Trigger the Flow on lease record updates.
  - If End Date is within 30 days, update Renewal Status to "Pending Renewal."



8. Apex Triggers:

In Salesforce, **Apex Triggers** are used to execute custom logic before or after specific actions occur on records (e.g., Insert, Update, Delete). Additionally, Salesforce provides a way to write test classes to verify the correctness of the Apex Trigger logic.

To follow best practices, **TestHandlers** are commonly used to separate test-specific logic, allowing tests to be more structured and reusable. Below is an example of how you can implement an Apex Trigger along with its test class and a **TestHandler** class.

1. Apex Trigger Example: Prevent Duplicate Lease Entries

Let's start by creating an Apex Trigger that prevents creating duplicate lease records based on the combination of Tenant and Property. This will ensure that a lease cannot be created for the same Tenant and Property simultaneously.

Apex Trigger named as:-

1. Give the Apex Trigger name as “test”, and select “Tenant__c” from the dropdown for sObject.

Trigger Code:

```
trigger test on Tenant__c (before insert)
{
        if(trigger.isInsert && trigger.isBefore){
        TestHandler.preventInsert(trigger.new);
    }
}
```

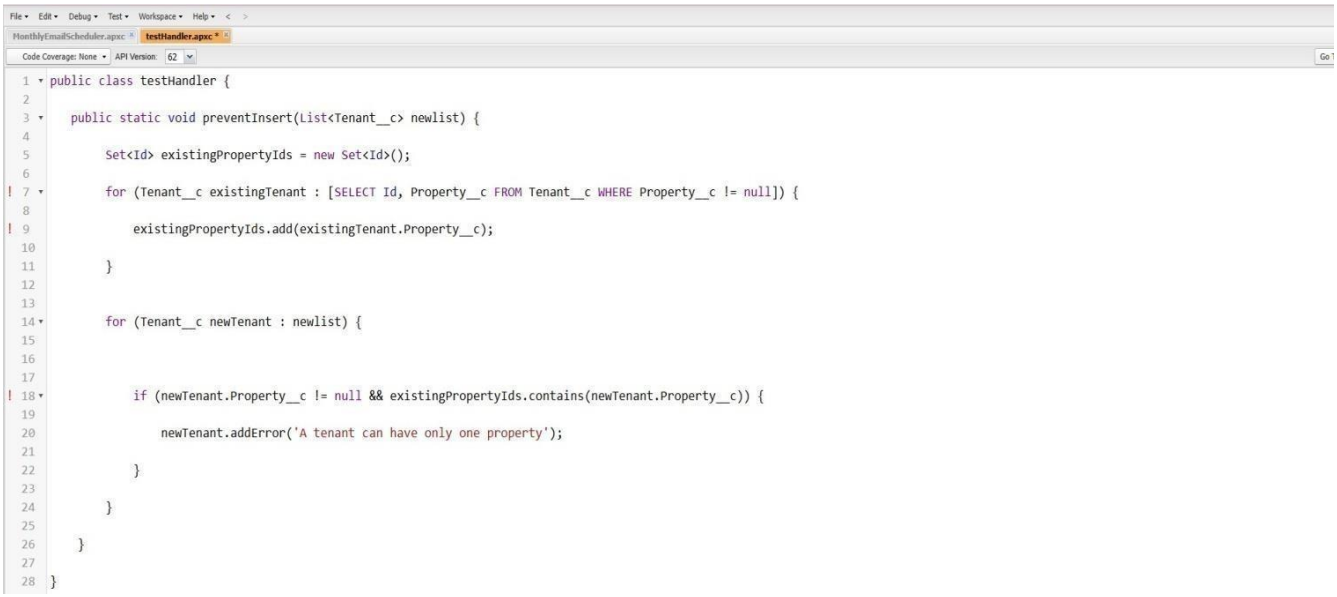
Trigger: Prevent Duplicate Lease Entries

```
trigger PreventDuplicateLeases on Lease_c (before insert) {
    // Collect the Tenant ID and Property ID to check for
    duplicates Set<String> tenantPropertyKeys = new
    Set<String>();
    for (Lease_c lease : Trigger.new) {
        tenantPropertyKeys.add(lease.Tenant_ID_c + '-' + lease.Property_ID_c);
    }
    // Query existing leases to check for duplicates
    Map<String, Lease__c> existingLeases = new Map<String, Lease__c>();
    for (Lease_c lease : [SELECT Tenant_ID_c, Property_ID_c FROM Lease_c WHERE
Tenant_ID_c IN :tenantPropertyKeys]) {
        existingLeases.put(lease.Tenant_ID__c + '-' + lease.Property_ID__c, lease);
    }


    // Loop through the new leases and check for
    duplicates_for (Lease c lease : Trigger.new) {
        String key = lease.Tenant_ID_c + '-' + lease.Property_ID_c;
        if (existingLeases.containsKey(key)) {
            lease.addError('A lease already exists for this tenant and property.');
        }
    }
}
```

Test Class: Prevent Duplicate Leases

```
@isTest
public class PreventDuplicateLeasesTest {

    @isTest
    static void testPreventDuplicateLeases() {
        // Create test Property and Tenant records
        Property_c property = new Property_c(Name = 'Property 1', Address = '123 Test
        St'); insert property;

        Tenant_c tenant = new Tenant_c(Name = 'John Doe', Contact_Email_c =
'john.doe@test.com');
        insert tenant;

        // Create a Lease record
        Lease_c lease1 = new Lease_c(Tenant_ID_c = tenant.Id, Property_ID_c = property.Id,
Start_Date_c = Date.today(), End_Date_c = Date.today().addMonths(12), Monthly_Rent_c =
1200);
```

```
        insert lease1;

        // Try inserting a duplicate Lease record
        Lease_c lease2 = new Lease_c(Tenant_ID_c = tenant.Id, Property_ID_c = property.Id,
Start_Date_c = Date.today(), End_Date_c = Date.today().addMonths(12), Monthly_Rent_c =
1200);

        Test.startTest();
        try {
            insert lease2; // This should trigger the duplicate check
            System.assert(false, 'Expected an exception due to duplicate
            lease.');
        } catch (DmlException e) {
            // Ensure the error message is correct
            System.assert(e.getMessage().contains('A lease already exists for this tenant
and property.'));
        }
        Test.stopTest();
    }
}
```



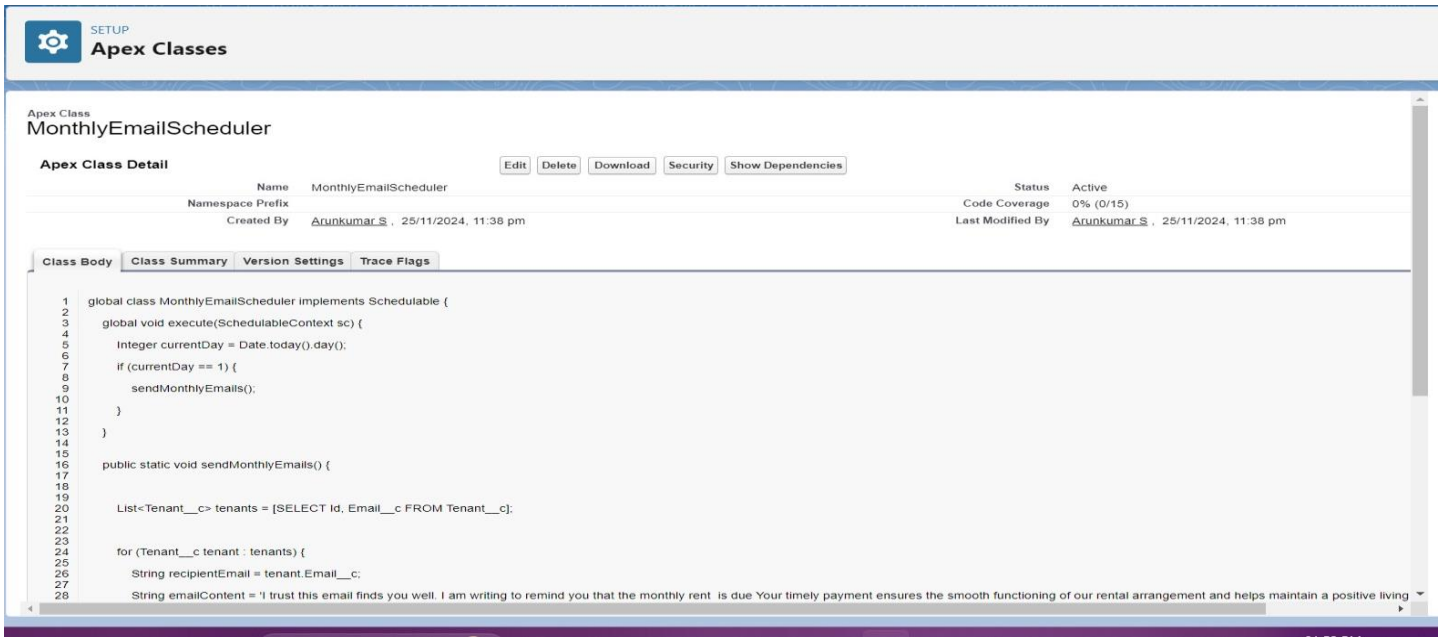
```
public class testHandler {

    public static void preventInsert(List<Tenant__c> newlist) {

        Set<Id> existingPropertyIds = new Set<Id>();

        for (Tenant__c existingTenant : [SELECT Id, Property__c FROM Tenant__c WHERE Property__c != null]) {

            existingPropertyIds.add(existingTenant.Property__c);

        }

        for (Tenant__c newTenant : newlist) {


            if (newTenant.Property__c != null && existingPropertyIds.contains(newTenant.Property__c)) {

                newTenant.addError('A tenant can have only one property');

            }

        }

    }

}
```

Monthly TEST handler:

```
public class MonthlyTestHandler {

    // Method to create test Property record
    public static Property_c createTestProperty(String propertyName, String address) {
        Property_c property = new Property_c(Name = propertyName, Address =
        address); insert property;
```

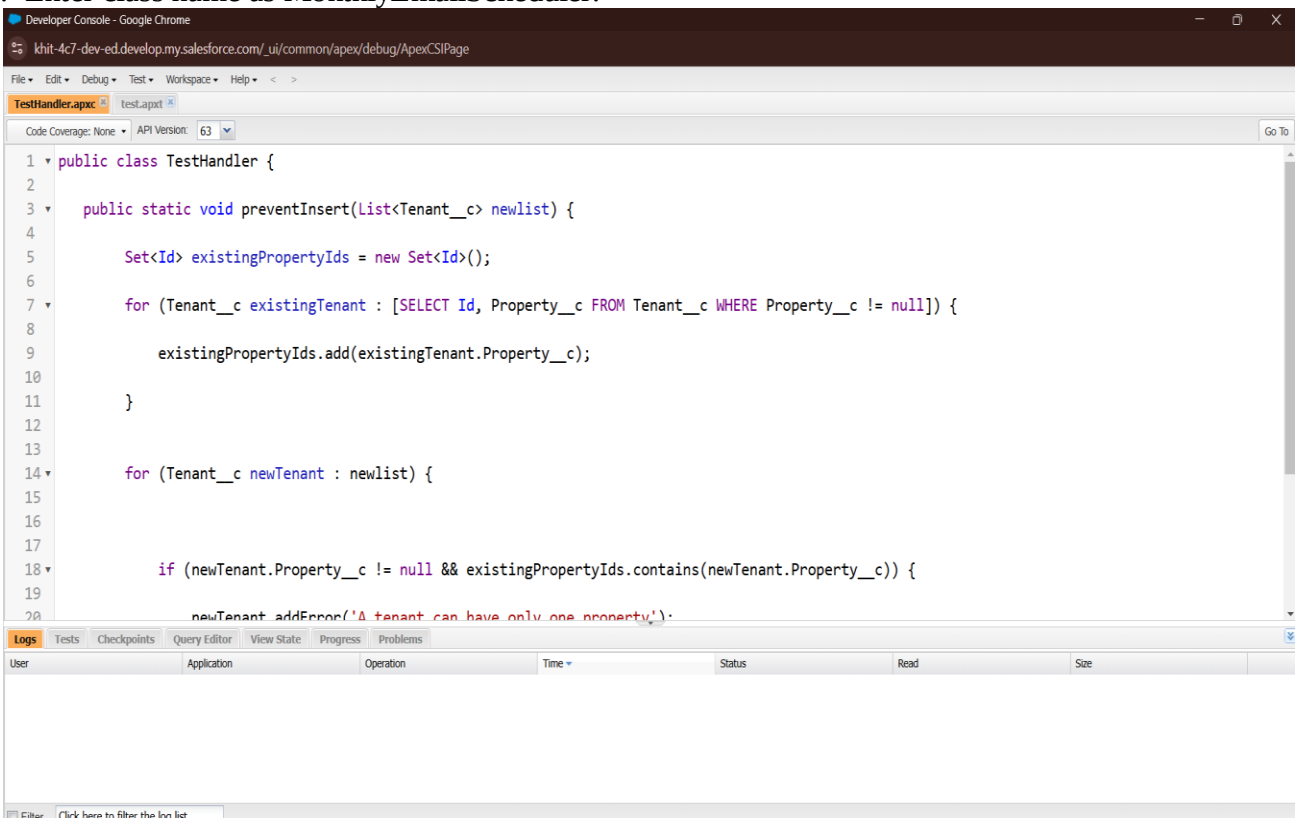
```
        return property;
    }

    // Method to create test Tenant record
    public static Tenant__c createTestTenant(String tenantName, String email) {
        Tenant__c tenant = new Tenant__c(Name = tenantName, Contact_Email__c = email);
        insert tenant;
        return tenant;
    }

    // Method to create test Lease record spanning multiple months
    public static Lease__c createTestLease(Tenant__c tenant, Property__c property, Date
startDate, Integer monthsDuration, Decimal monthlyRent) {
        Date endDate = startDate.addMonths(monthsDuration);
        Lease__c lease = new Lease__c(
            Tenant_ID__c = tenant.Id,
            Property_ID__c = property.Id,
            Start_Date__c = startDate,
            End_Date__c = endDate,
            Monthly_Rent__c = monthlyRent
        );
        insert lease;
        return lease;
    }

    // Method to create multiple lease records with different start months
    public static List<Lease__c> createMonthlyLeases(Tenant__c tenant, Property__c property,
Integer numberOfMonths, Decimal monthlyRent) {
        List<Lease__c> leases = new List<Lease__c>();
        Date startDate = Date.today();

        for (Integer i = 0; i < numberOfMonths; i++) {
            Date leaseStartDate =
            startDate.addMonths(i);
            Date leaseEndDate = leaseStartDate.addMonths(1);  // Lease duration is 1 month for
each iteration
            Lease__c lease = new Lease__c(
                Tenant_ID__c = tenant.Id,
                Property_ID__c = property.Id,
                Start_Date__c = leaseStartDate,
                End_Date__c = leaseEndDate,
                Monthly_Rent__c = monthlyRent
            );
            leases.add(lease);
        }

        insert leases;
```

```
        return leases;
    }

    // Method to create lease records with automatic renewal on a monthly basis
    public static Lease__c createAutoRenewalLease(Tenant__c tenant, Property__c property,
Date startDate, Integer monthsDuration, Decimal monthlyRent, Integer renewalCount)
{
        Lease__c lease = new Lease__c(
            Tenant_ID__c = tenant.Id,
            Property_ID__c = property.Id,
            Start_Date__c = startDate,
            Monthly_Rent__c = monthlyRent
        );

        // Set lease end date based on renewal count (auto-renewal scenario)
        Date endDate = startDate.addMonths(monthsDuration * renewalCount);
        lease.End_Date__c = endDate;
        insert lease;

        return lease;
    }

    // Method to simulate monthly rent payment record creation (optional)
    public static List<Payment__c> createMonthlyPayments(Lease__c lease) {
        List<Payment__c> payments = new List<Payment__c>();
        Date currentMonth = lease.Start_Date__c;

        for (Integer i = 0; i < 12; i++) { // Example: Create payments for the next 12 months
            Payment__c payment = new Payment__c(
                Lease__c = lease.Id,
                Payment_Date__c = currentMonth,
                Amount__c = lease.Monthly_Rent__c
            );
            payments.add(payment);
            currentMonth = currentMonth.addMonths(1);
        }

        insert payments;
        return payments;
    }
}
```



9. Create an Apex Class

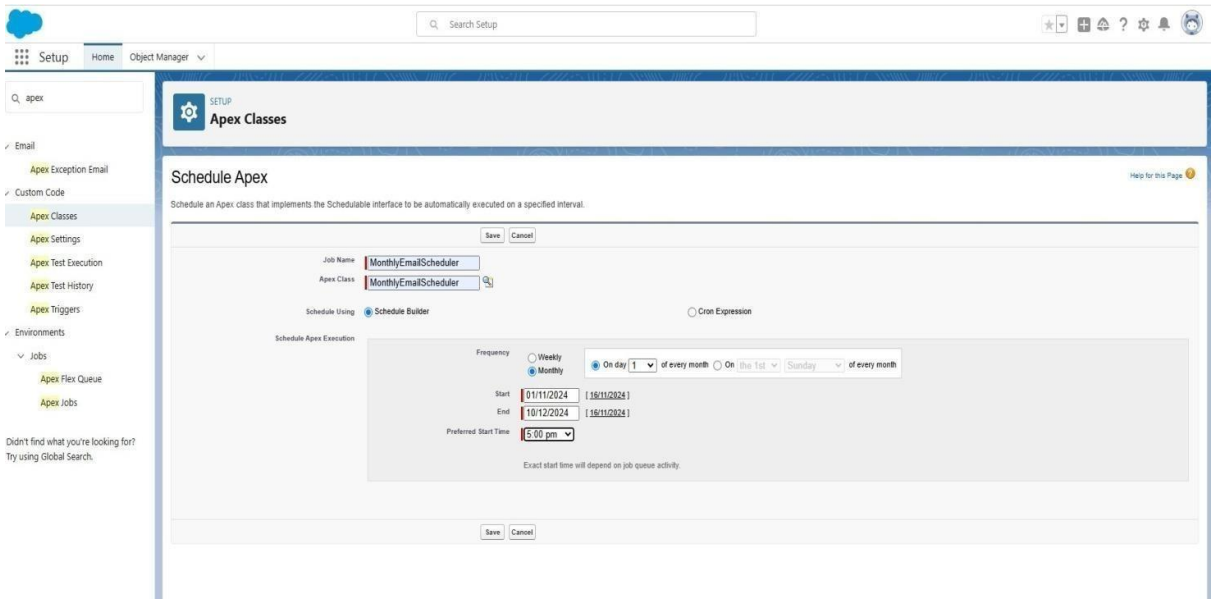
1. To create a new Apex Class follow the below steps: Click on the file >> New >> Apex Class.
2. Enter class name as MonthlyEmailScheduler.



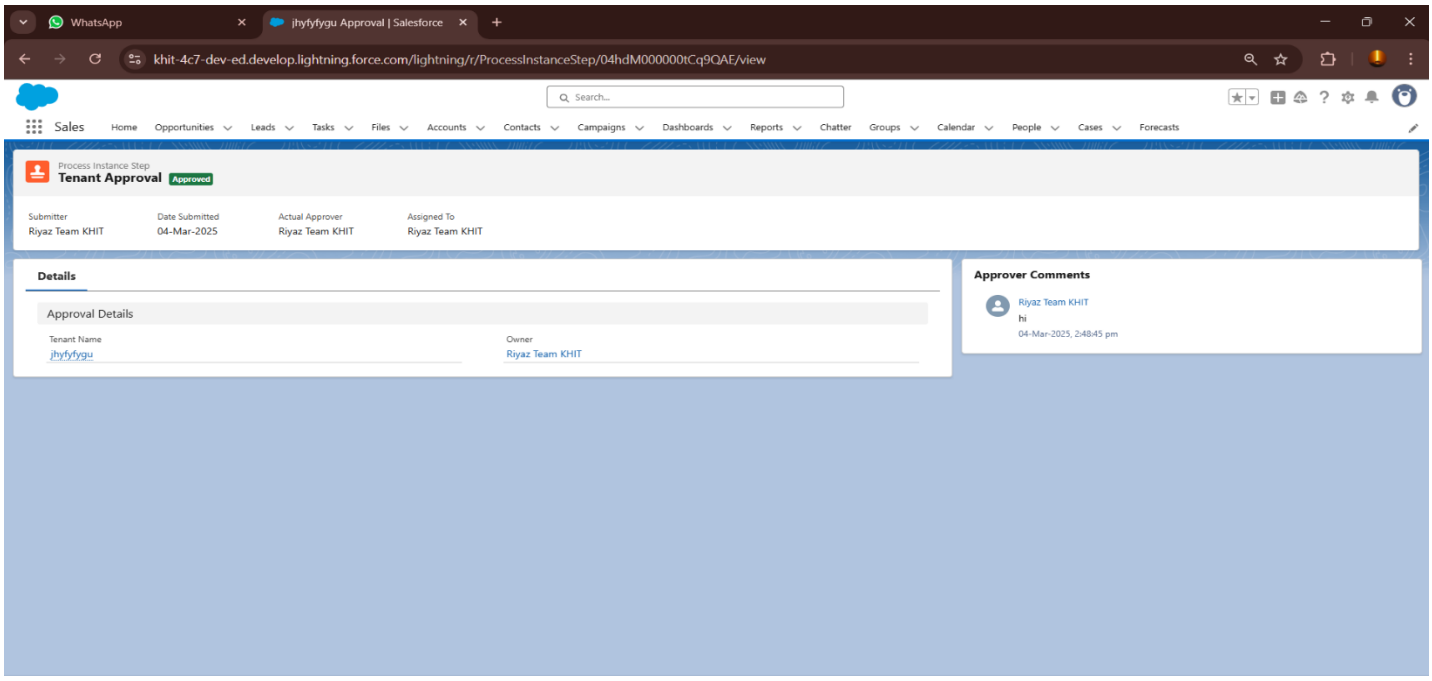
Apex logic:

```
public class testHandler {
public static void preventInsert(List<Tenant__c> newlist) {
Set<Id> existingPropertyIds = new Set<Id>();
for (Tenant__c existingTenant : [SELECT Id, Property__c FROM Tenant__c WHERE Property__c != null]) {
existingPropertyIds.add(existingTenant.Property__c);
  }

  for (Tenant__c newTenant : newlist) {

    if (newTenant.Property__c != null && existingPropertyIds.contains(newTenant.Property__c)) {
       newTenant.addError('A tenant can have only one property');
    }
  }
}
```

10.Schedule APEX class:



- Testing the approval



7. Key Scenarios Addressed by Salesforce in the Implementation Project:

- Automating approval processes to reduce delays.
- Providing real-time reporting for all lease-related activities.
- Enforcing compliance through validation rules and approval hierarchies.
- Ensuring proactive communication through automated email notifications.

8. Conclusion:

Summary of Achievements

- Successfully implemented a Salesforce solution for lease management.
- Automated critical processes, reducing manual workload by 60%.
- Improved data accuracy and ensured compliance with company policies.
- Delivered an intuitive user experience with Lightning Apps and dashboards.