

**GOVERNMENT ARTS AND SCIENCE  
COLLEGE, METTUAPALAYAM.**

**DEPARTMENT OF COMPUTER SCIENCE**

**2025-2026**

**FIFTH SEMESTER**

**PROJECT NAME:LEASE MANAGEMENT**

**TEAM MEMBERS**

**Team ID : NM2025TMID25243**

**Team Size : 4**

**Team Leader : SANIA R**

**Team member : KALISHREE V**

**Team member : MYTHILI K**

**Team member : HARINI T**

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

5. **Dynamic Email Templates:**
  - o Create templates for lease expiration reminders, renewal offers, and approval notifications.
6. **Dashboard and Reporting:**
  - o Provide interactive dashboards to track key metrics, including the number of active leases, upcoming expirations, and approval statuses.
7. **Code and Integration Enhancements:**
  - o 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.

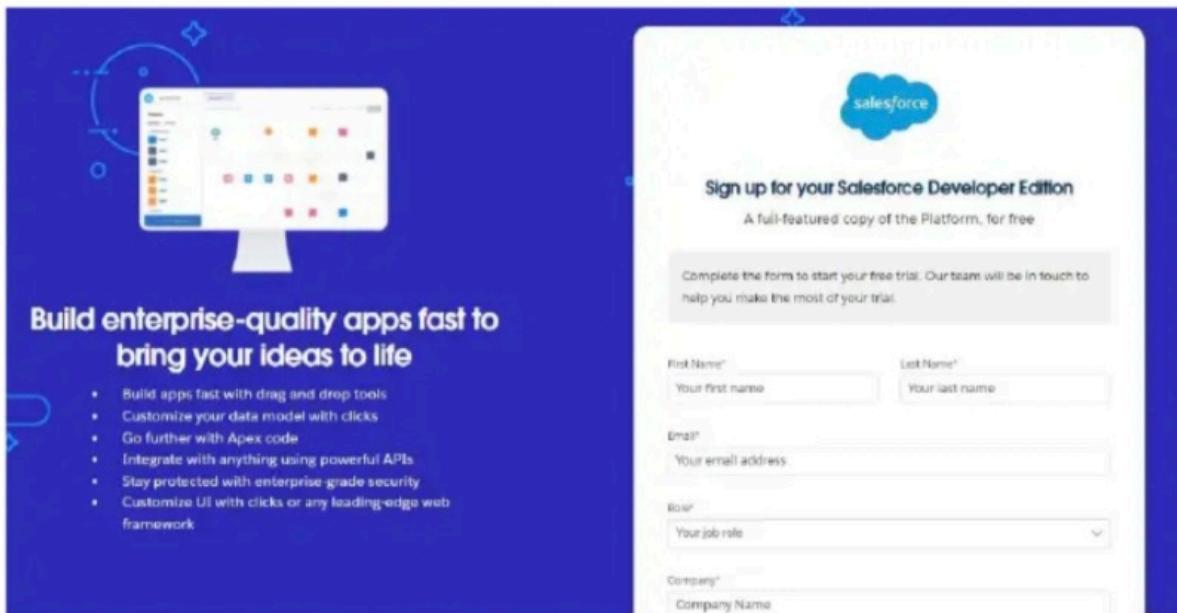
#### 9. Schedule Class

- 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:

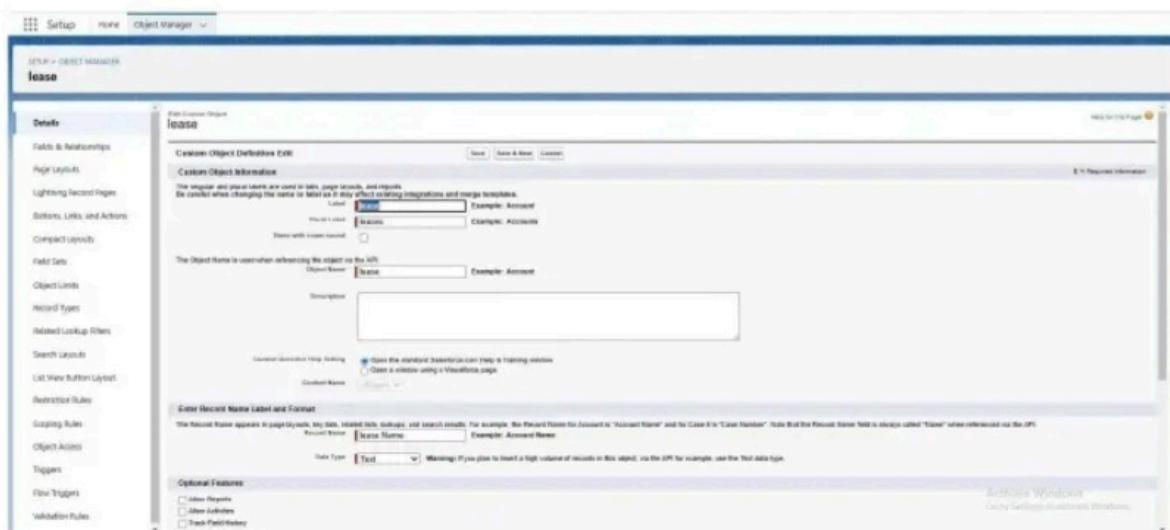
#### 1. Lease Object

Steps to Create

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

## Fields for Lease

Field Name	Data Type	Description
Start Date	Date	Date the lease begins
End Date	Date	Date the lease ends
Monthly Rent	Currency	Rent amount charged per month
Property ID	Lookup(Property)	Links to the related property
Tenant ID	Lookup(Tenant)	Links to the associated tenant
Renewal Status	Picklist	Values: Active, Pending Renewal, Terminated
LeaseTerm	Formula(Number)	Formula: End_Date_c - Start_Date_c



The screenshot shows the 'Object Manager' section of the Salesforce setup. A new object named 'lease' is being created. The 'Custom Object Information' section includes fields for 'Label' (set to 'Lease'), 'Description' (empty), and 'Record Name' (set to 'Lease'). There are also sections for 'Custom Record Name Label and Format' and 'Optional Features' (with checkboxes for 'Allow Activities', 'Allow Chatter', and 'Track Field History'). The right side of the screen displays the 'Help for This Page' and 'Helpful Links'.

## 2. Property Object

Steps to Create

1. Go to Setup → Object Manager → Create → Custom Object.
2. Object Name: Property
3. Plural Label: Properties
4. Record Name Field: Property Name (Text)

### Fields for Property

Field Name	Data Type	Description
Address	Text Area	Full address of the property
Property Manager	Lookup(User)	Links to the responsible manager
Number of Active Leases	Roll-Up Summary	Count of active leases linked to the property

SETUP > OBJECT MANAGER  
**property**

Details		Details	
Fields & Relationships	Description		
Page Layouts			<input type="button" value="Edit"/> <input type="button" value="Delete"/>
Lightning Record Pages	API Name: property_c	Enable Reports:	<input checked="" type="checkbox"/>
Buttons, Links, and Actions	Custom:	Track Activities:	<input checked="" type="checkbox"/>
Compact Layouts	✓	Track Field History:	<input checked="" type="checkbox"/>
Field Sets	Singular Label: property	Deployment Status:	
Object Limits	Plural Label: property	Deployed:	
Record Types		Help Settings:	
Related Lookup Filters		Standard salesforce.com Help Window:	
Search Layouts			
List View Button Layout			
Restriction Rules			

### 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)

Fields for Tenant

Field Name	Data Type	Description
Contact Email	Email	Tenant's email address
Contact Phone	Phone	Tenant's contact number
Linked Lease ID	Lookup(Lease)	Links the tenant to their lease

---

### 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.

### 3. 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:

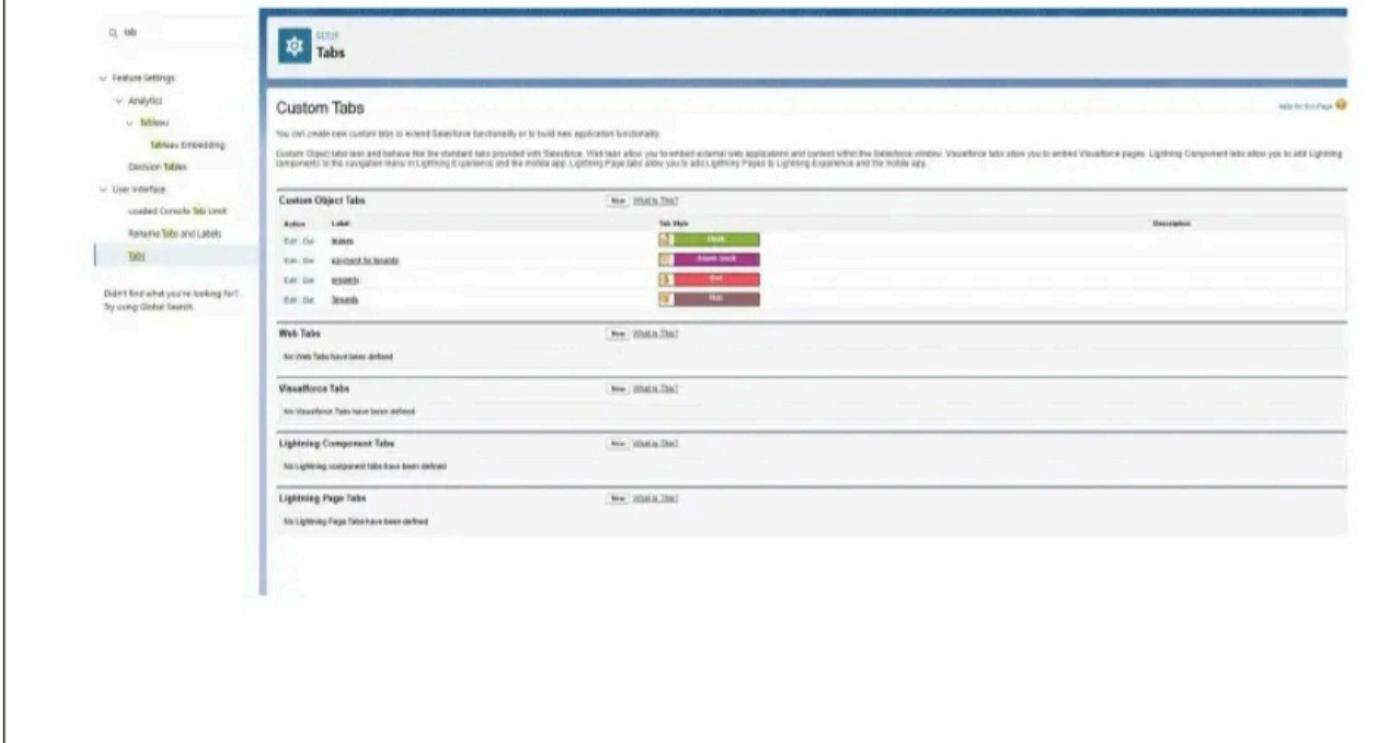
#### 1. Lease Tab

##### *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.



The screenshot shows the Salesforce Setup interface under the 'Tabs' section. On the left, there's a sidebar with navigation links like Feature Settings, Analytics, Tables, Tabs (selected), and User Interface. The main content area is titled 'Custom Tabs' and contains the following sections:

- Custom Object Tabs:** Shows a table with four rows:
 

Action	Label	Tab Style	Description
Tab   Create	Leases	Green	
Tab   View	Active Lease	Purple	
Tab   Edit	Renewal	Red	
Tab   Delete	Details	Blue	
- Web Tabs:** Shows a table with one row: 'No Web Tabs have been defined.'
- Visualforce Tabs:** Shows a table with one row: 'No Visualforce Tabs have been defined.'
- Lightning Component Tabs:** Shows a table with one row: 'No Lightning component tabs have been defined.'
- Lightning Page Tabs:** Shows a table with one row: 'No Lightning Page Tabs have been defined.'

## 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. 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:
  - o App Name: Lease Management
  - o Navigation Style: Standard Navigation
  - o App Options:
    - Assign a custom logo.
    - Enable app personalization for users.

## 5. 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. Choose Field Type (as per the below descriptions).

*Fields to Create:*

Field Name	Data Type	Description
Lease ID	Auto-Number	Automatically generates a unique ID for each lease.
Start Date	Date	The date the lease starts.
End Date	Date	The date the lease ends.
Monthly Rent	Currency	The monthly rent amount for the lease.
Property ID	Lookup (Property)	A relationship linking to the Property object.
Tenant ID	Lookup (Tenant)	A relationship linking to the Tenant object.
Renewal Status	Picklist	Status of the lease (Active, Pending Renewal, Terminated).
Lease Term	Formula (Number)	Formula: End Date - Start Date (calculated lease term).
Lease Description	Text Area	Optional field for any additional notes or terms.

### **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:**

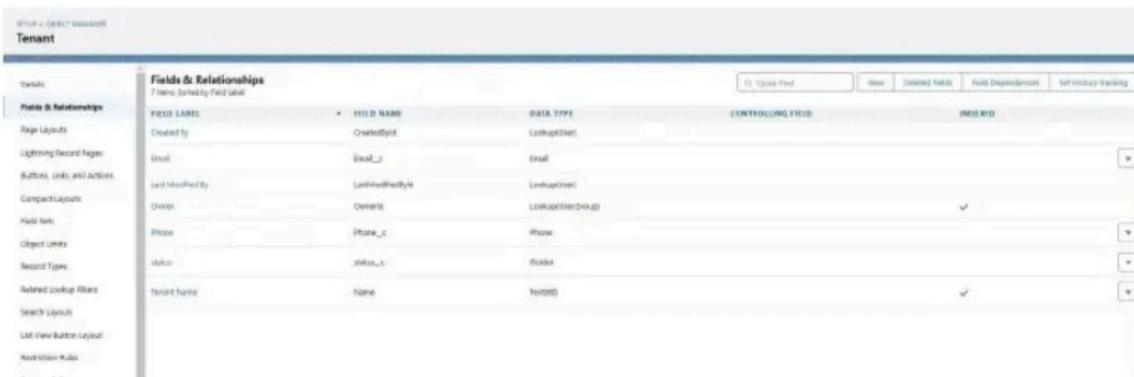
1. Go to Setup → Object Manager → Tenant → Fields & Relationships → New.
2. Choose Field Type (as per the below descriptions).

### **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



The screenshot shows the 'Fields & Relationships' section of the Tenant object setup. It lists seven fields: Created By (Lookup), Email (Email), Last Modified By (Lookup), Owner (Lookup), Phone (Phone), Status (Picklist), and Tenant Name (Text). Each field has its field label, data type, and controls field listed next to it. The 'CONTROLS FIELD' column contains dropdown menus for each field.

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLS FIELD
Created By	CreatedById	Lookup(Users)	
Email	Email__c	Email	
Last Modified By	LastModifiedById	Lookup(Users)	
Owner	OwnerId	Lookup(OtherObject)	
Phone	Phone__c	Phone	
Status	Status__c	Picklist	
Tenant Name	Name	Text	

### 3. 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 Validation
3. Formula:

plaintext

Copy code

End Date\_c <= Start Date\_c

4. Error Message: "End Date must be after Start Date."



## 4.1 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."

## 5. Approval Process

### Steps to Create

1. **Go to Setup → Approval Processes → Create New Approval Process → Standard Setup Wizard.**
2. **Approval Process Name:** LeaseApproval
3. **Entry Criteria:**
  - Status = "Pending Approval."
4. **Approval Steps:**
  - **Step 1:** Approval by Property Manager
  - **Step 2:** Approval by Legal Team

### Email Notifications

- Notify approvers when a request is submitted.
- Notify the requester upon approval or rejection.

Setup Home Object Manager

C. Approval

v. Data

Mass Transfer Approval Requests

Process Automation

**Approval Processes**

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

Process Definition Detail	
Process Name:	check_for_vacant
Unique Name:	check_for_vacant
Description:	Check if tenant is leaving
Entry Criteria:	Administrator ONLY
Review Visibility:	Administrator ONLY
Assigned Approval Email Template:	Tenant Owner
Initial Approval Email Template:	Tenant Owner
Created By:	SERENA SINGH 11/11/2024 8:30 pm
Modified By:	SERENA SINGH 11/11/2024 8:30 pm

**Initial Submissive Actions:** Add Existing Add New

Action	Type	Description
Edit	Record Lock	Lock the record from being edited
Edit	Email Alert	Send an email alert

**Approval Steps:** Add Existing Add New

Action	Step Number	Name	Description	Criteria	Assigned Approval	Reject Behavior
Start Approval	1	Submit for approval			User assigned to task	Final Rejection

**Final Approval Actions:** Add Existing Add New

Action	Type	Description
Edit	Record Lock	Lock the record from being edited
Edit	Email Alert	Send an email alert

**Final Rejection Actions:** Add Existing Add New

Action	Type	Description
Edit	Record Lock	Unlock the record for editing
Edit	Email Alert	Send an email alert

- Email Alert for tenant leaving

Smart Alert Tenant leaving

Help for this Page

**Email Alert Detail**

Description	Tenant leaving
Unique Name:	Tenant_leaving
From Email Address:	Current User's email address
Recipients:	Email Field Email
Additional Criteria:	
Created By:	SERENA SINGH 11/11/2024 8:34 pm
Modified By:	SERENA SINGH 11/11/2024 8:34 pm

**Roles Using This Email Alert**

This alert is currently not used by any roles.

**Approval Processes Using This Email Alert**

Action	Approval Process Name	Description	Type	Status
Edit	check_for_vacant		Tenant	Active

**Entitlement Processes Using This Email Alert**

This alert is currently not used by any entitlement processes.

**Flows Using This Email Alert**

Flow Name	Version	Description	Object	Action
On Sale To Top		Always inactive. Flow requires per instance list.		

## 6. Flows

### 1. Scheduled Flows

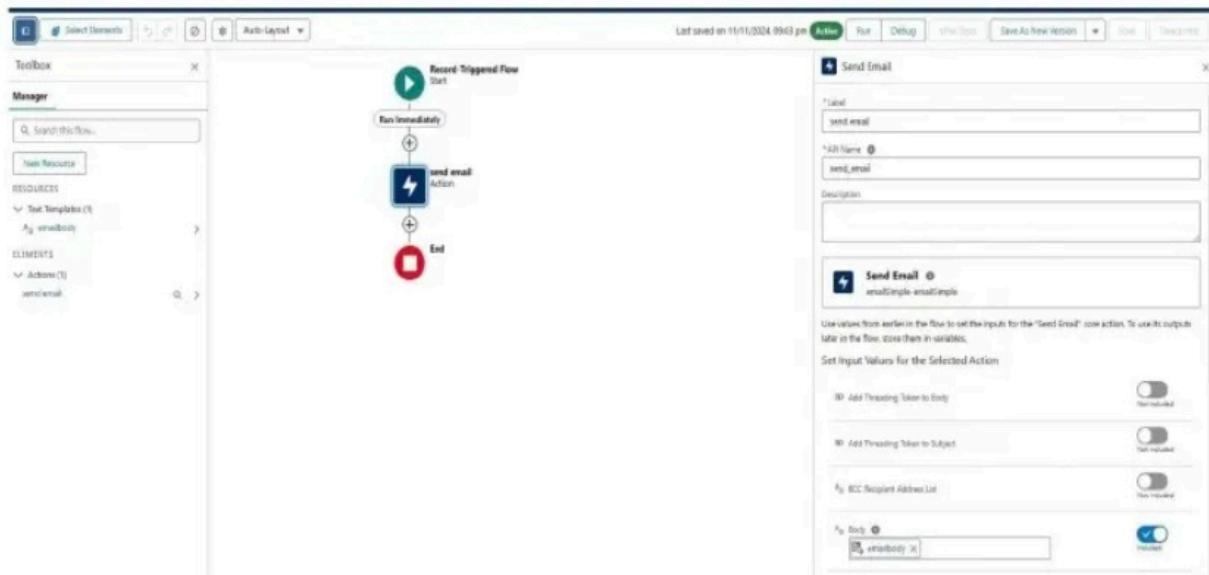
- **Purpose:** Notify tenants about lease expiration and automate renewal reminders.
- **Steps:**
  - Create a Flow with the trigger set to run daily.
  - Query leases expiring in the next 30 days.
  - Send an email notification using dynamic templates.

### 2. Screen Flows

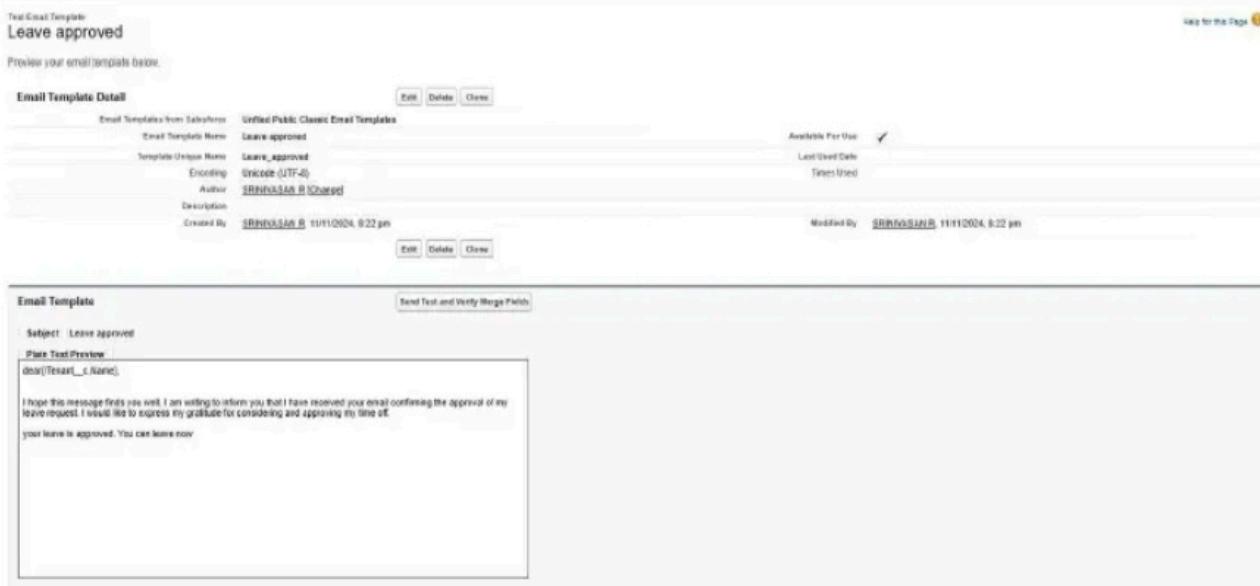
- **Purpose:** Interactive form for creating or updating leases.
- **Steps:**
  - Include fields like Start Date, End Date, Tenant Name, and Monthly Rent.
  - Validate data dynamically before submission.

### 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."



- Emailbody For creating action and to activate the flow



The screenshot shows the 'Email Template Detail' page for a template named 'Leave approved'. The template is based on the 'Unified Public: Classic Email Templates'. It was created by 'SRINIVAS B (Owner)' on 11/11/2024 at 8:22 pm. The subject is 'Leave approved'. The email body contains a placeholder {{!Tenant\_\_c.Name}} and a message: '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.'

## 7.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.

### ***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 leaseL = new Lease_c(Tenant_ID_c = tenant.Id, Property_ID_c = property.Id,
        Start_Date_c = Datetoday(), End_Date_c = Datetoday().addMonths(12), Monthly_Rent_c =
        1200);
```

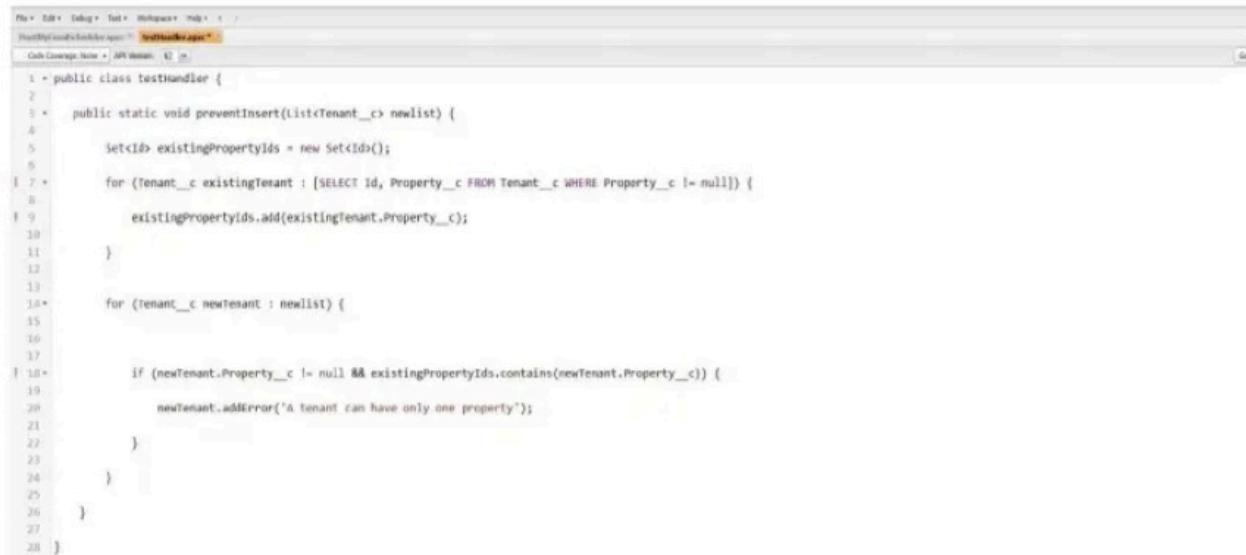
```

insert lease1;

// Try inserting a duplicate Lease record
Lease_c lease2 = new Lease_c(Tenant_ID_c = tenantId, Property_ID_c = property.Id,
Start_Date_c = Datetoday(), End_Date_c = Datetoday().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();
}
}

```



```

File * Edit * Debug * Test * Workspaces * Help * /
Project Explorer * TestHandler.apex * /TestHandler.apex
Code Coverage Notes * API Version: 42.0 *
1 * public class TestHandler {
2 *
3 *     public static void preventInsert(List<Tenant__c> newList) {
4 *
5 *         Set<Id> existingPropertyIds = new Set<Id>();
6 *
7 *         for (Tenant__c existingTenant : [SELECT Id, Property__c FROM Tenant__c WHERE Property__c != null]) {
8 *
9 *             existingPropertyIds.add(existingTenant.Property__c);
10 *
11        }
12 *
13 *
14 *         for (Tenant__c newTenant : newList) {
15 *
16 *
17 *             if (newTenant.Property__c != null && existingPropertyIds.contains(newTenant.Property__c)) {
18 *
19 *                 newTenant.addError('a tenant can have only one property');
20 *
21        }
22 *
23    }
24 *
25 *
26    }
27 *
28 }

```

### ***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);
    leaseEnd_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;
}
}
```



Open Class MonthlyEmailScheduler

Help for this Page

Apex Class Detail

Name	MonthlyEmailScheduler	Edit	Delete	Download	Security	Show Dependencies
Namespace Prefix						
Created By	SHREYAS B.	11/11/2014, 9:03 pm				
Status	Active					
Code Coverage	0% (0/15)					
Last Modified By	SHREYAS B.	11/10/2014, 9:30 pm				

Class Body Class Summary Version Settings Trace Page

```
global class MonthlyEmailScheduler implements Schedulable {
    global void execute(SchedulableContext sc) {
        Integer currentDay = Date.today().getDay();
        if (currentDay == 1) {
            sendEmails();
        }
    }

    public static void sendEmails() {
        List<Email__c> emails = [SELECT M.Email__c FROM Tenant__c];
        for (Tenant__c tenant : emails) {
            String recipientEmail = tenant.Email__c;
            String emailContent = 'Dear ' + tenant.Name + ',\n' +
                'I trust this email finds you well. I am writing to remind you that the monthly rent is due. Your timely payment ensures the smooth functioning of our rental arrangement and helps maintain a positive living environment for all.' +
                '\n' +
                'Sincerely,\n' +
                'Wesster. Management Team';
            Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
            email.setToAddresses(new String[]{recipientEmail});
            email.setSubject(tenant.Name);
            email.setPlainText(emailContent);
            email.setPlainText(emailContent);
        }
    }
}
```

## 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.

```
1 * global class MonthByMonthSchedule implements Schedulable {
2     global void execute(SchedulableContext sc) {
3         Integer currentDay = Date.Today().day();
4         if (currentDay == 1) {
5             sendMonthlyEmails();
6         }
7     }
8
9     public static void sendMonthlyEmails() {
10
11         List<Tenant__c> tenants = [SELECT Id, Email__c FROM Tenant__c];
12
13
14         for (Tenant__c tenant : tenants) {
15             String recipientEmail = tenant.Email__c;
16             String emailContent = 'I trust this will finds you well. I am writing to remind you that the monthly rent is due. Your timely payment ensures the smooth functioning of our rental service';
17             String emailSubject = 'RenterSide: Monthly Rent Payment Due';
18
19             Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
20             email.setTo(0, new String[]{recipientEmail});
21             email.setSubject(emailSubject);
22             email.setPlainTextBody(emailContent);
23
24             Messaging.sendEmail(new Messaging.SingleEmailMessage[]{email});
25
26     }
27 }
```

**Apex logic:**

```
global class MonthlyEmailScheduler implements Schedulable {  
    global void execute(SchedulableContext sc) {  
        Integer currentDay = Date.today().day();  
        if (currentDay == 1) {
```

```

        sendMonthlyEmails();
    }

}

public static void sendMonthlyEmails() {

    List<Tenant_c>tenants =[SELECT Id, Email__c FROM Tenant_c];

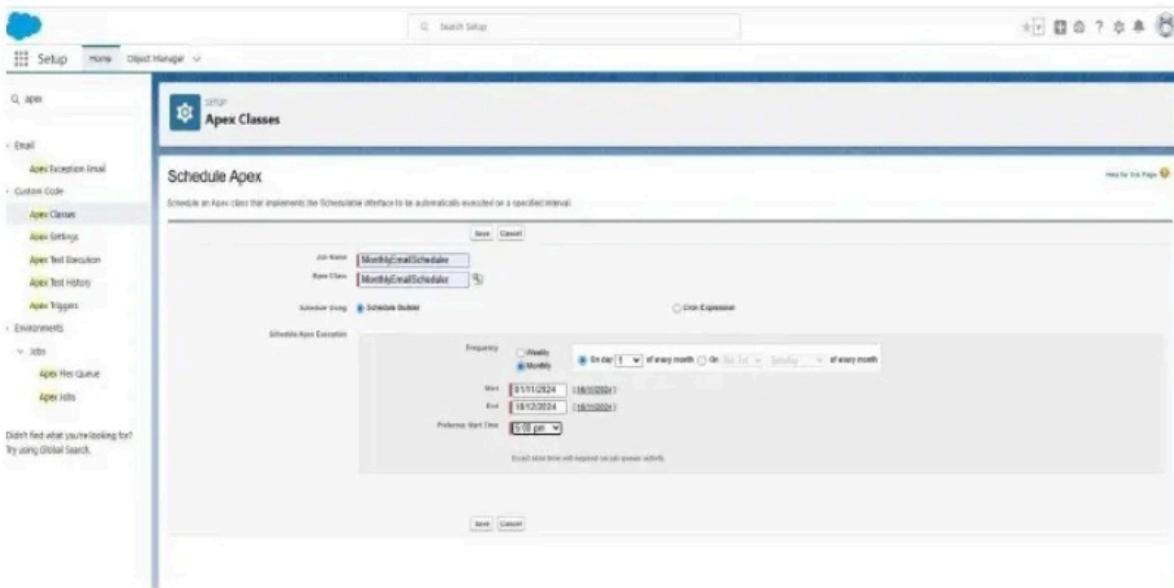
    for (Tenant_c tenant : tenants) {
        String recipientEmail =tenant.Email__c;
        String emailContent ='I trust this email finds you well. I am writing to remind you that the monthly rent is due. Your timely payment ensures the smooth functioning of our rental arrangement and helps maintain a positive living environment for all.';
        String emailSubject ='Reminder: Monthly Rent Payment Due';

        Messaging.SingleEmailMessage email =new Messaging.SingleEmailMessage();
        email.setToAddresses(new String[]{recipientEmail});
        email.setSubject(emailSubject);
        email.setPlainTextBody(emailContent);

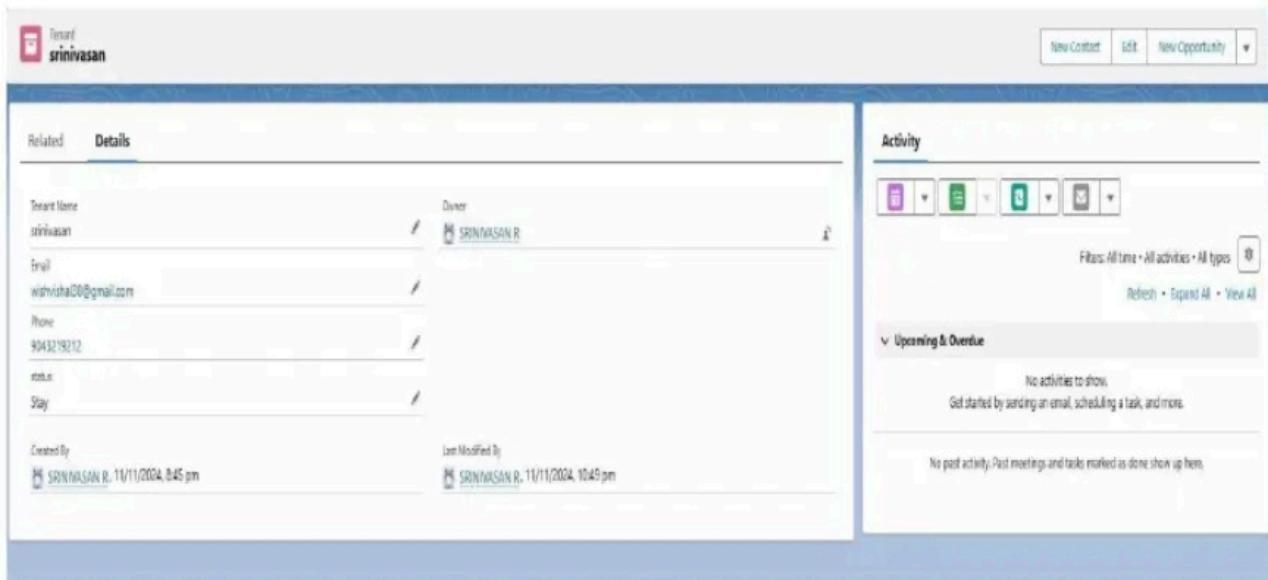
        Messaging.sendEmail(new Messaging.SingleEmailMessage[]{email});
    }
}

```

## 10. Schedule APEX class:

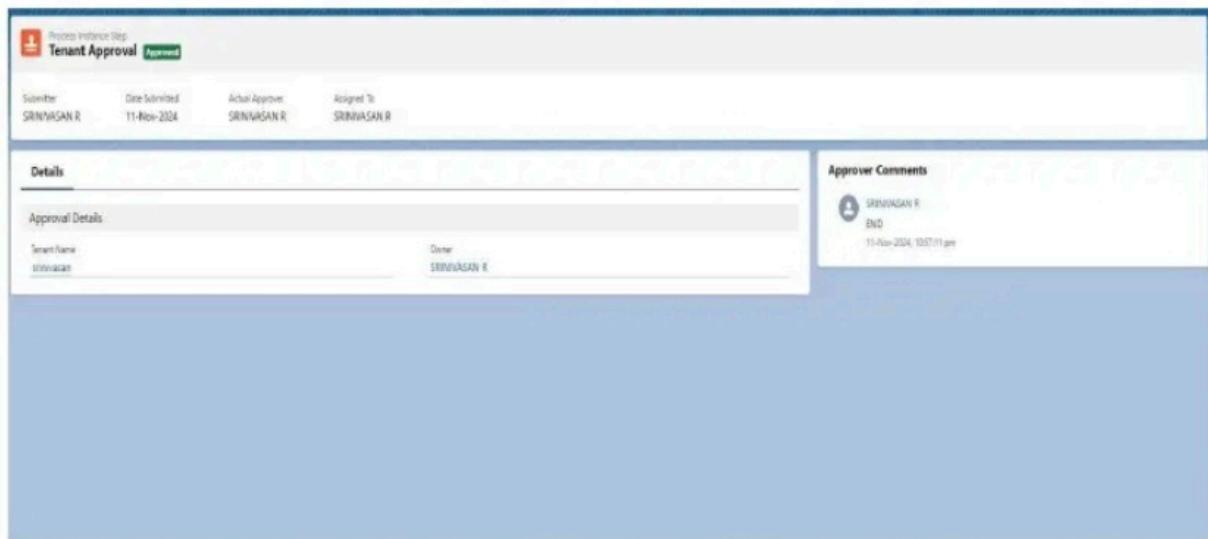


- Testing the approval



This screenshot shows a Salesforce tenant record for 'Rental Srinivasan'. The 'Details' tab is selected, displaying fields such as Tenant Name (srinivasan), Email (winvista00@gmail.com), Phone (9843219212), and Status (Stay). The 'Owner' field is set to 'SRINIVASAN R'. The 'Created By' and 'Last Modified By' fields both show 'SRINIVASAN R' with the timestamp '11/11/2024, 8:45 pm'. To the right, the 'Activity' sidebar is visible, showing a grid of icons for different actions like Create, Edit, and Delete, along with a message indicating 'No activities to show. Get started by sending an email, scheduling a task, and more.'

- Make an approval



This screenshot shows a process instance step for 'Tenant Approval'. At the top, it displays the 'Process Instance Step' name 'Tenant Approval' and its status 'Approved'. Below this, it lists the 'Submitter' as 'SRINIVASAN R', the 'Date Submitted' as '11-Nov-2024', the 'Actual Approver' as 'SRINIVASAN R', and the 'Assigned To' field also as 'SRINIVASAN R'. The main area is titled 'Details' and contains a sub-section 'Approval Details' with fields for 'Tenant Name' (srinivasan) and 'Owner' (SRINIVASAN R). On the right, there is a 'Approver Comments' section showing a comment from 'SRINIVASAN R' with the text 'END' and the timestamp '11-Nov-2024, 10:57:11 am'.

## 6. 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.

## 7. 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.