



**SKP ENGINEERING COLLEGE**

**PROJECT TITLE:**

**LEASE MANAGEMENT**

**TEAM MEMBERS:**

**BANU PRIYA .P**

**CHANDRU .J**

**CHANDRU .N**

**DEENADHAYALAN .E**



**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.

1. **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.

1. **Dynamic Email Templates:** 
   * Create templates for lease expiration reminders, renewal offers, and approval notifications.
2. **Dashboard and Reporting:** 
   * Provide interactive dashboards to track key metrics, including the number of active leases, upcoming expirations, and approval statuses.
3. **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. o 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. o Prevent the deletion of leases tied to active tenants.
* Calculate penalties for late renewals.

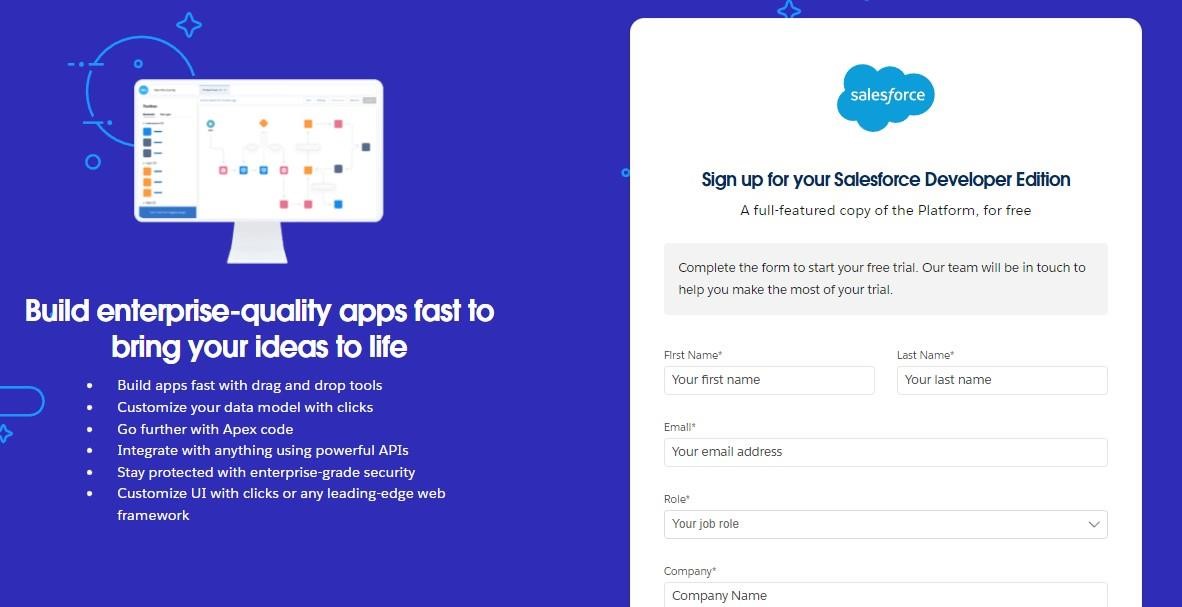
# 9. Schedule Class

• A Schedule Class automates periodic tasks, such as:

o Sending lease expiration reminders. o 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>



1. **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**: Lease ID (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

Lease

Term

Formula(Number)

Formula:

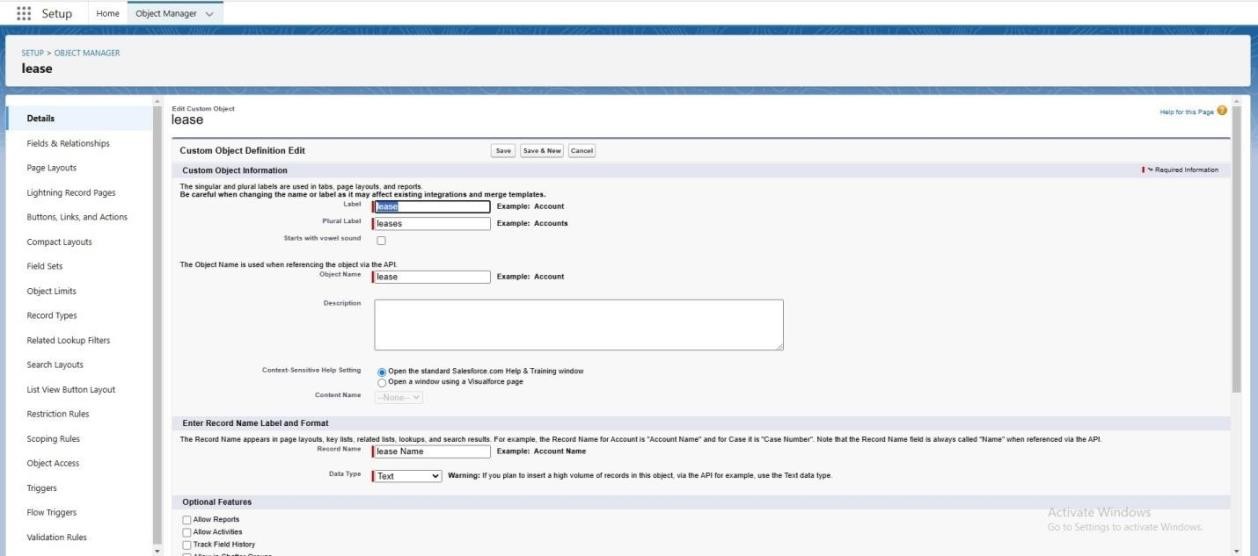
End\_Date

c

-

Start\_Date

c



1. ***Property Object***

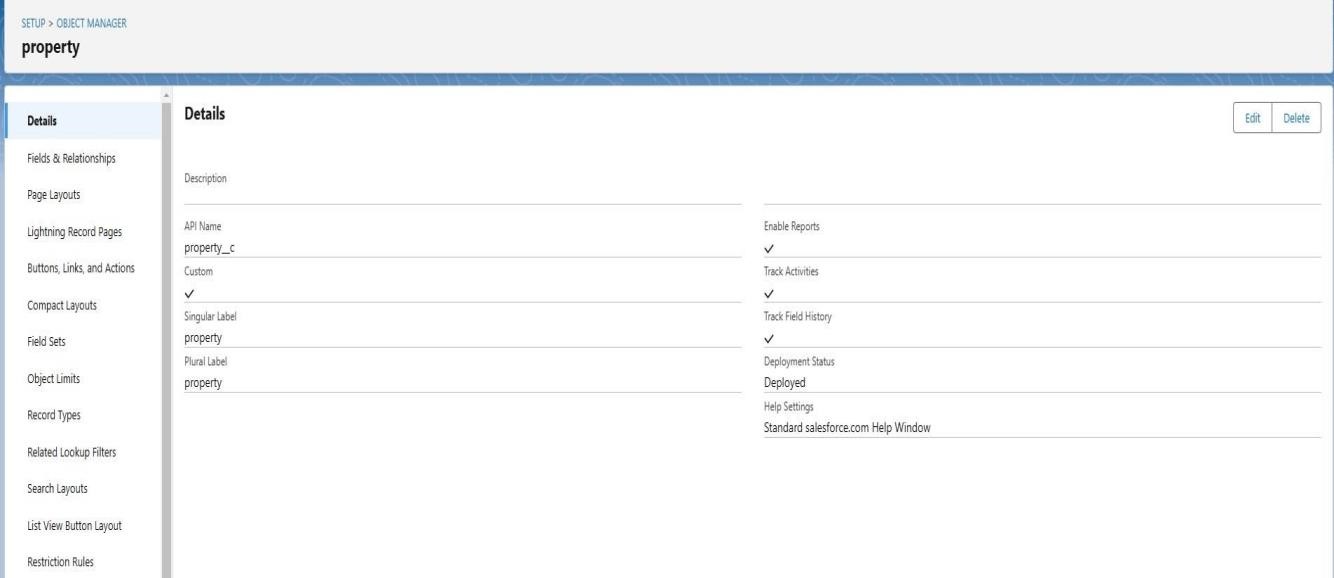
## Steps to Create

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



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

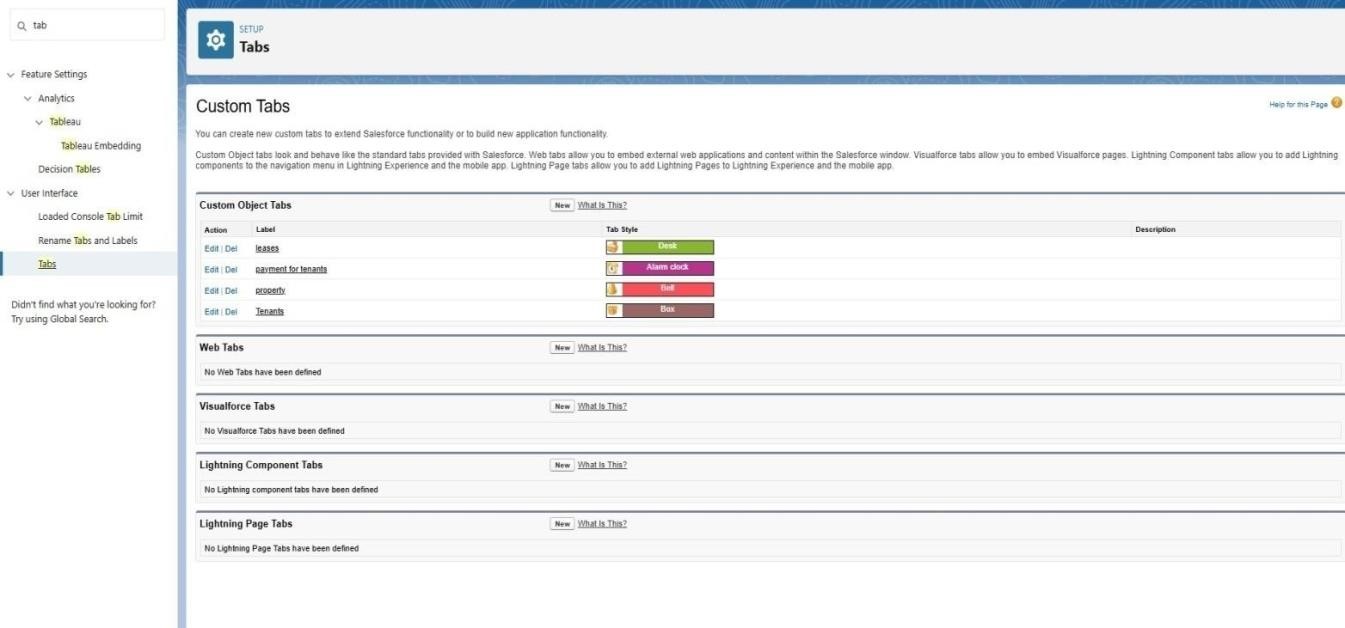
## 1. Lease Tab

***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.



1. **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.

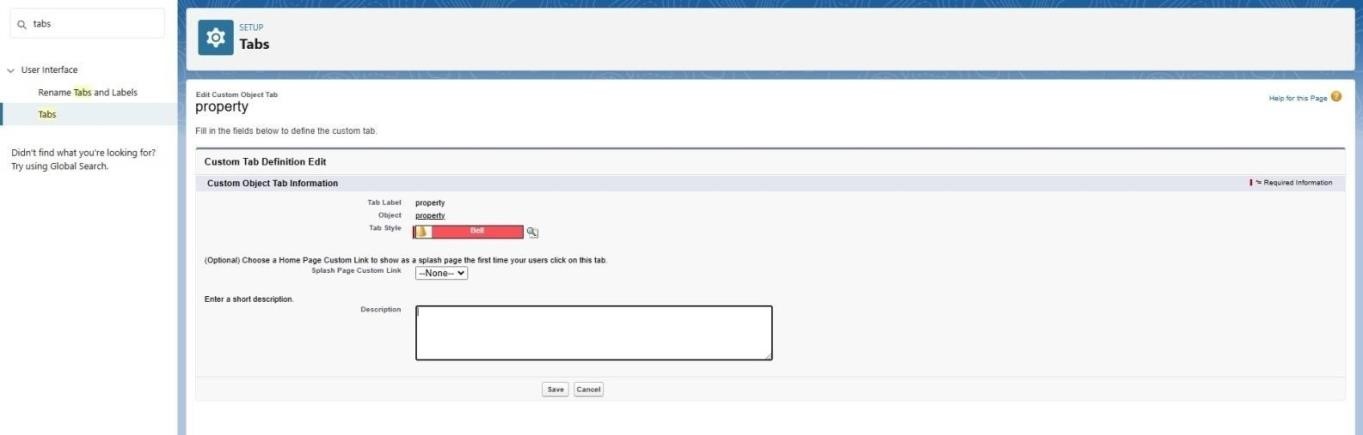
1. **Property Tab**

***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.



1. **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.

1. **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** | Status of the lease (Active, Pending Renewal,  Picklist  Terminated). |
| **Lease Term** | Formula  Formula: End Date - Start Date (calculated lease term). (Number) |
| **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.

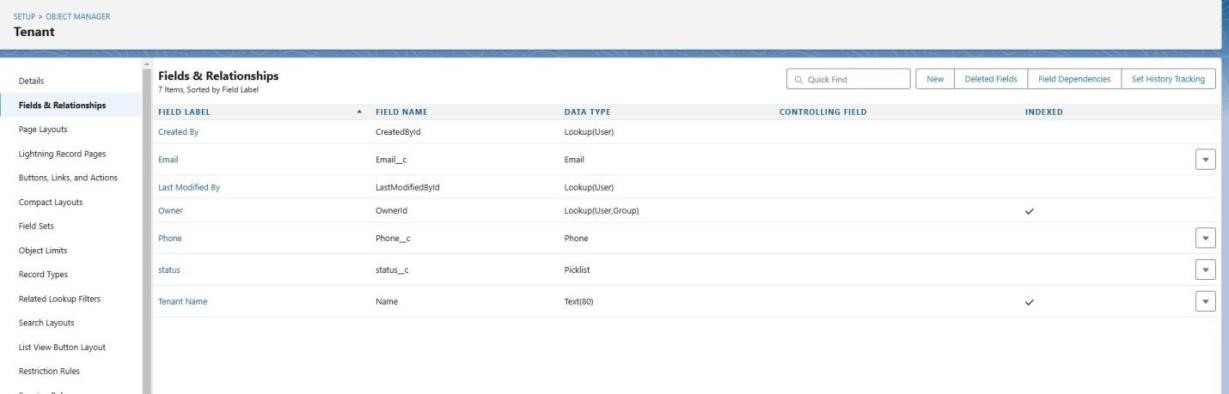
**Contact Phone** Phone Phone number of the tenant.

**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



## 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** Lookup (User) **Manager** | 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** Roll-Up  **Leases** 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

1. **Formula**:

plaintext

Copy code

End\_Date c <= Start\_Date c

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

**4.1Validation 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

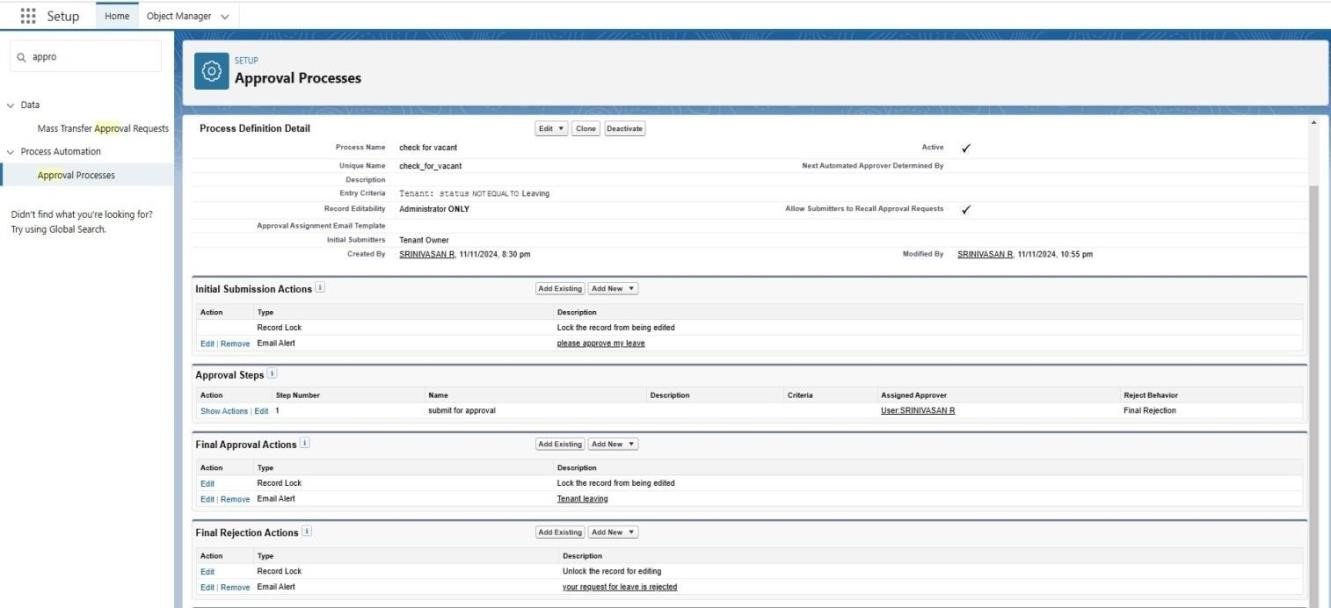
1. **Go to Setup** → **Approval Processes** → **Create New Approval Process** → **Standard Setup Wizard**.
2. **Approval Process Name**: Lease Approval 3. **Entry Criteria**: o Status = "Pending Approval."

4. **Approval Steps**:

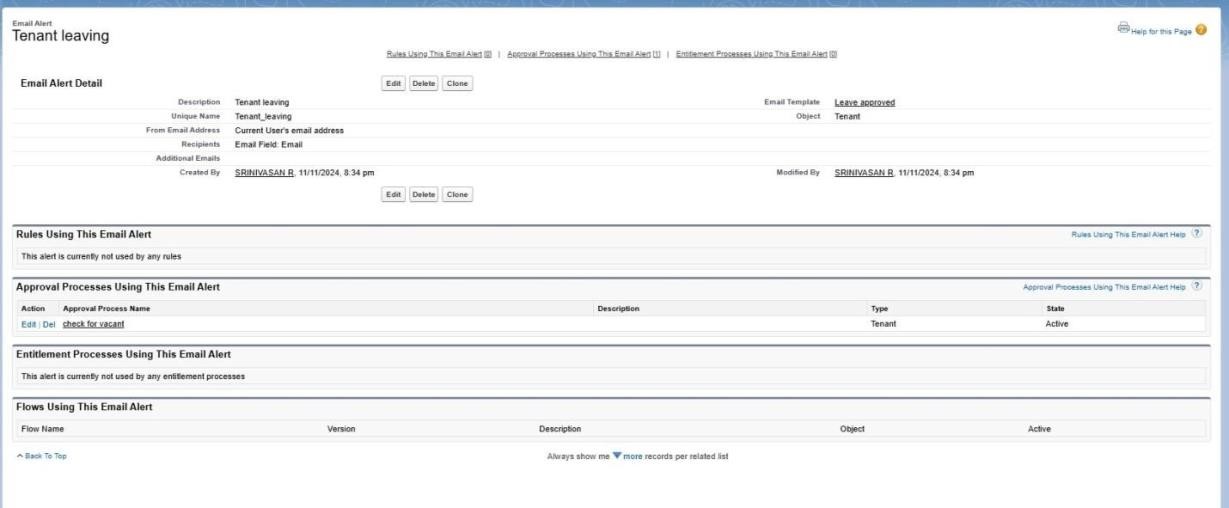
o **Step 1**: Approval by Property Manager o **Step 2**: Approval by Legal Team

### Email Notifications

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



* Email Alert for tenant leaving



**6. Flows**

### 1. Scheduled Flows

* **Purpose**: Notify tenants about lease expiration and automate renewal reminders.
* **Steps**: o Create a Flow with the trigger set to run daily. o Query leases expiring in the next 30 days.

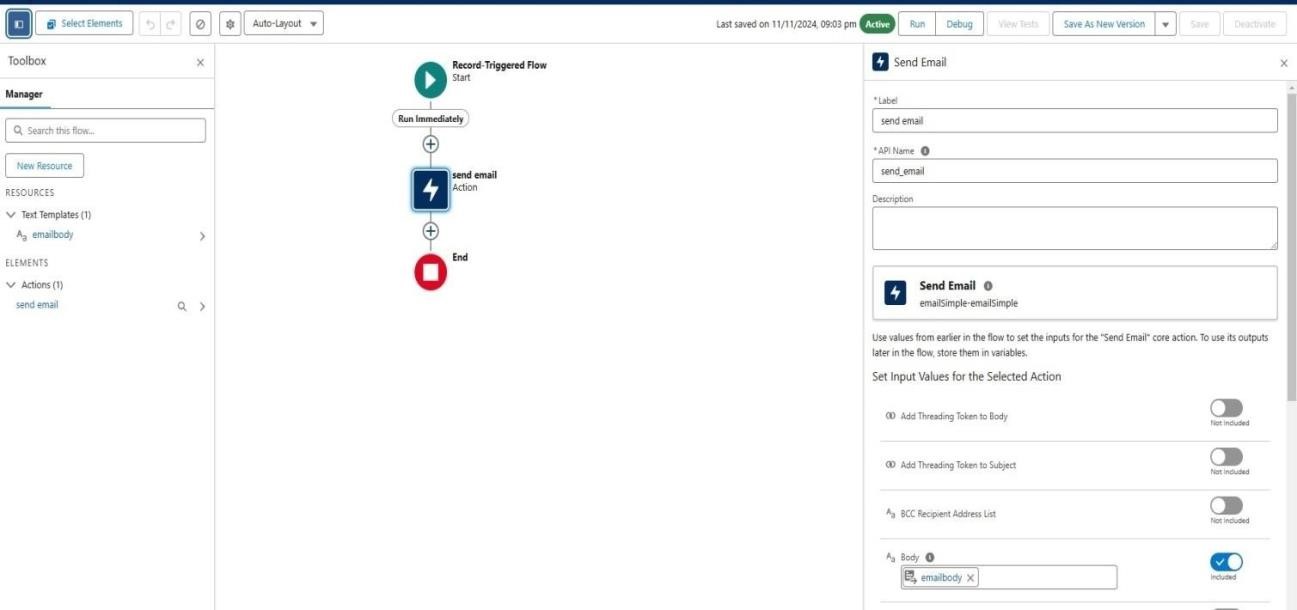
o Send an email notification using dynamic templates.

### 2. Screen Flows

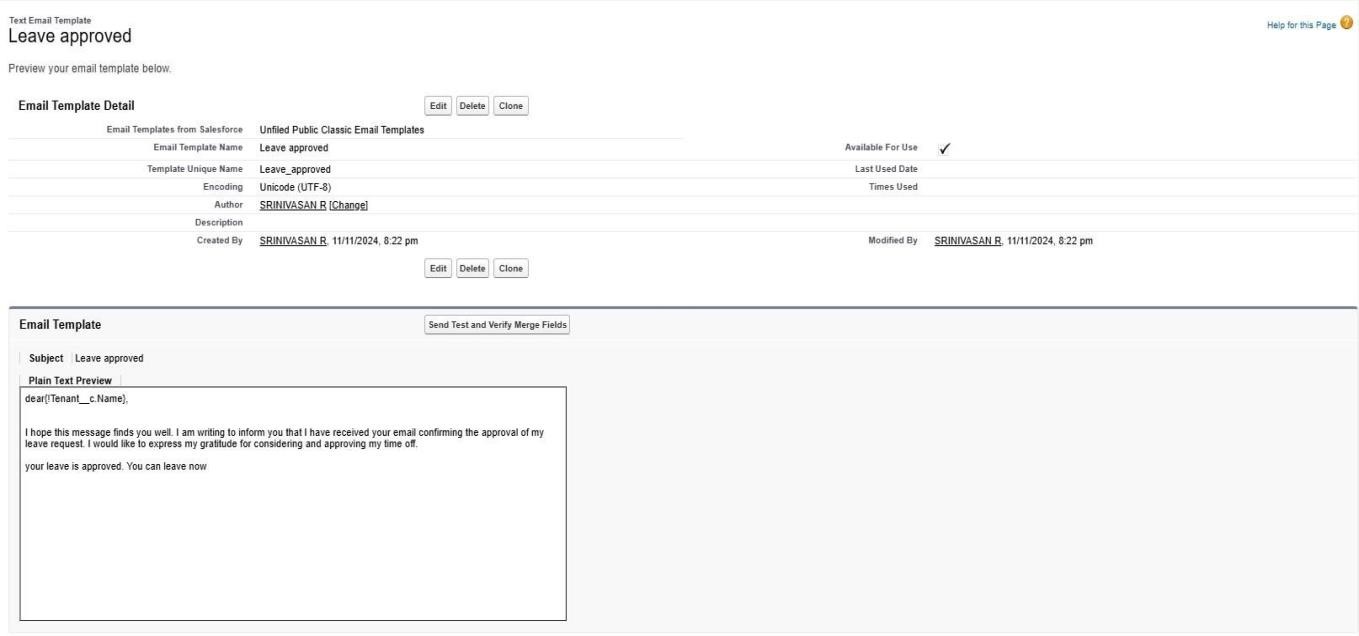
* **Purpose**: Interactive form for creating or updating leases.
* **Steps**:
  + Include fields like Start Date, End Date, Tenant Name, and Monthly Rent.
  + Validate data dynamically before submission.

### 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



**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 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

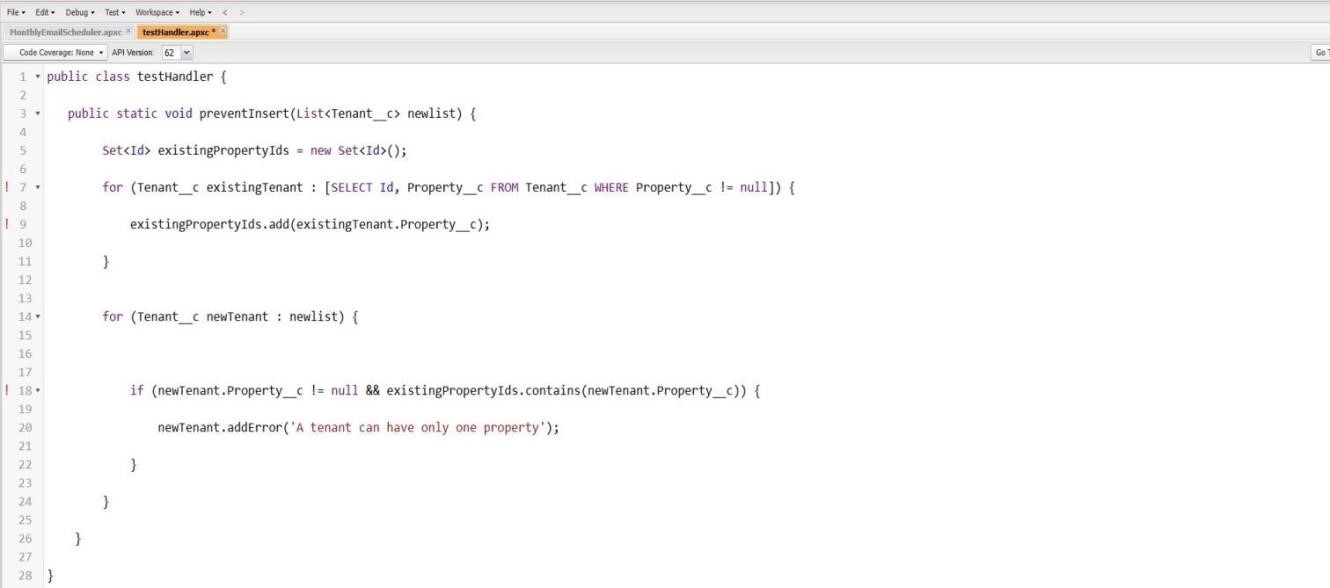
System.assert(e.getMessage().contains('A lease already exists for this tenant and property.'));

}

Test.stopTest();

}

}



***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

);

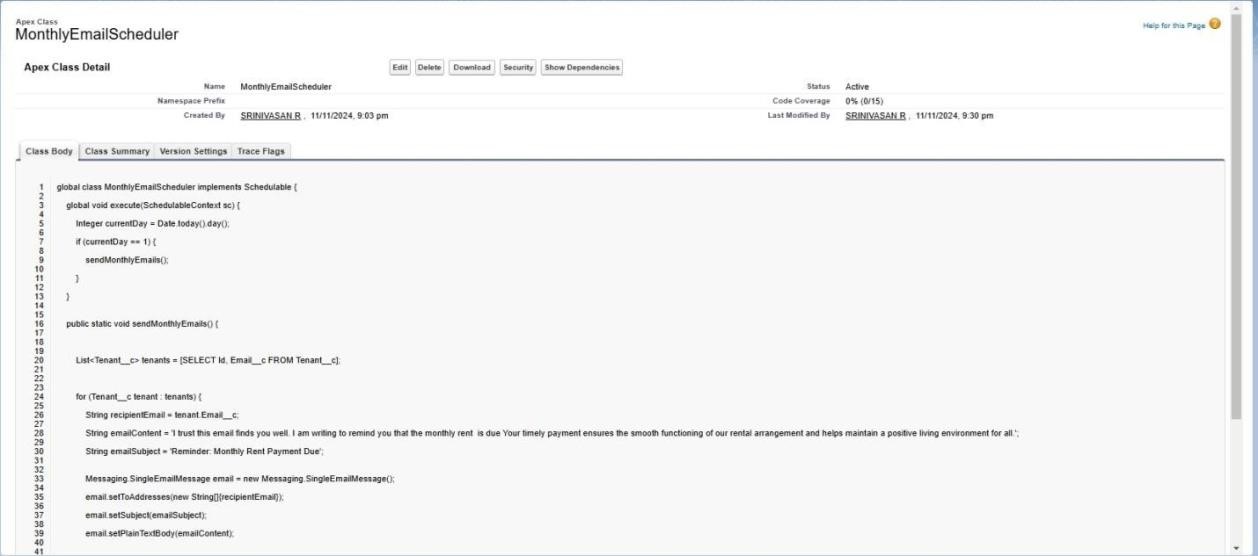
payments.add(payment);

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:** 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';

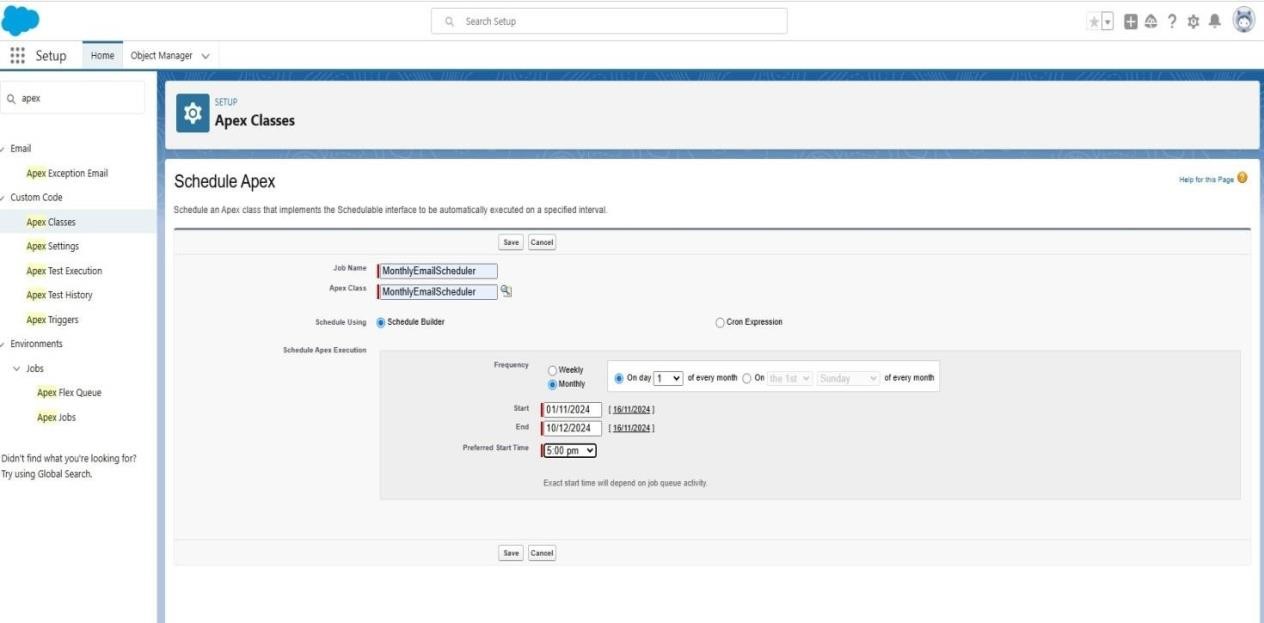
Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage(); email.setToAddresses(new String[]{recipientEmail}); email.setSubject(emailSubject); email.setPlainTextBody(emailContent);

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

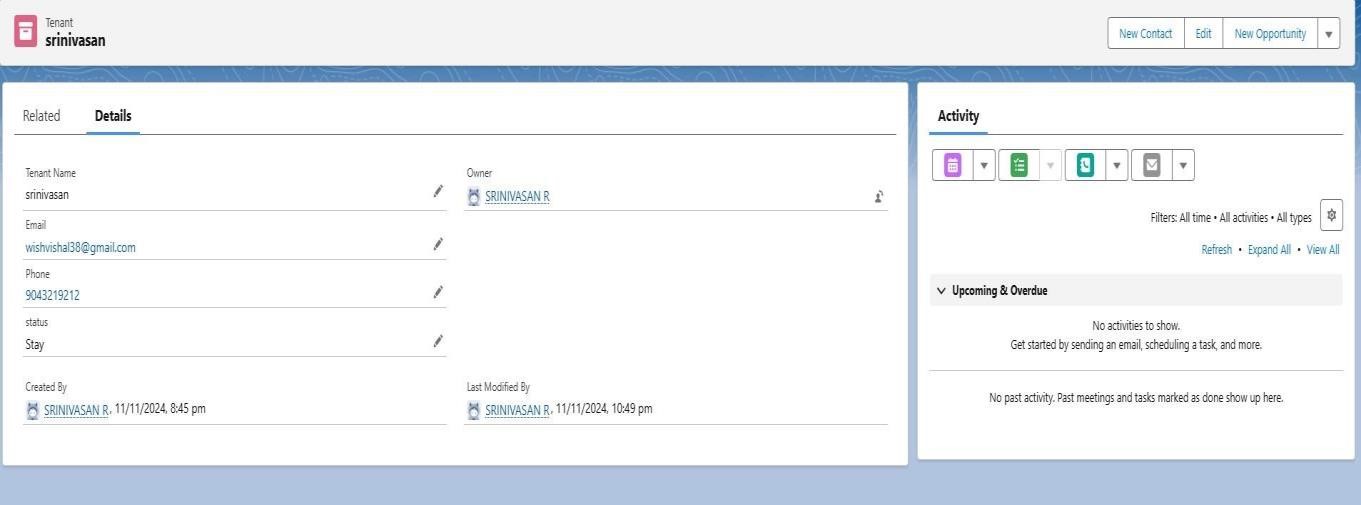
}

}

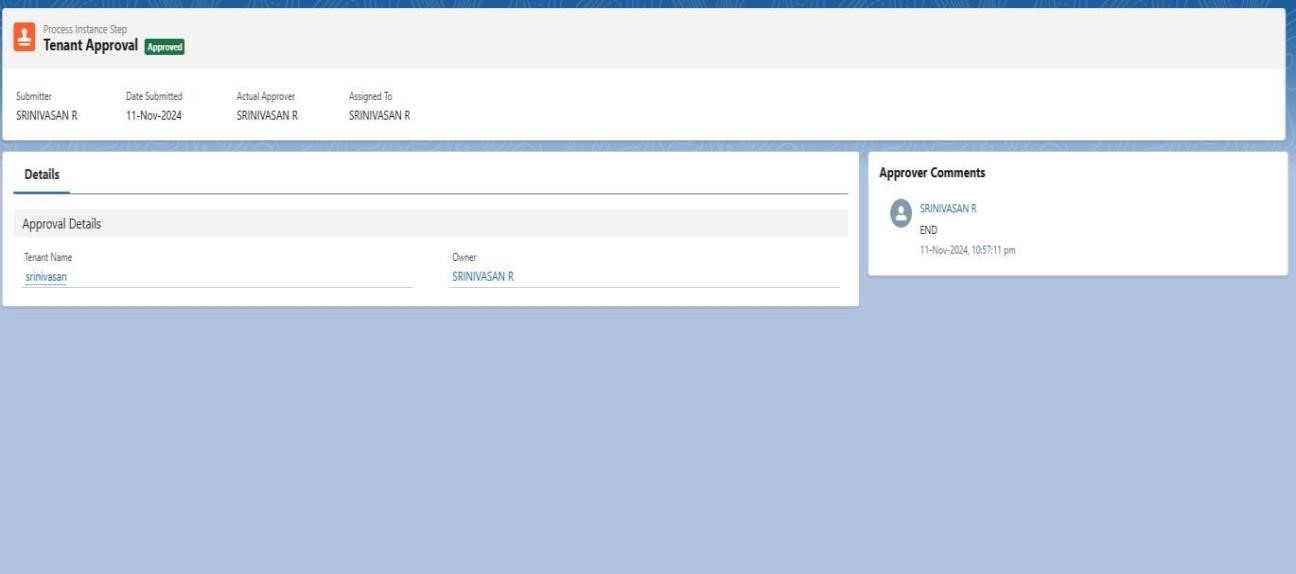
**10. Schedule APEX class:**



* Testing the approval



* Make an approval



1. **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.

1. **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.