LEASE MANAGEMENT

College Name: KPR College of Arts Science and Research

College Code: bruaz

TEAM ID: NM2025TMID21390

TEAM MEMBERS:

Team Leader Name: Raksha S

Email: 23bda049@kprcas.ac.in

Team Member 1: Rubiha M

Email: 23bda062@kprcas.ac.in

Team Member 2: Sandhiya C

Email: 23bda063@kprcas.ac.in

Team Member 3: Sanjay G

Email: 23bda065@kprcas.ac.in

1.INTRODUCTION

1.1 Project Overview

The Lease Management System is a Salesforce-based application designed to streamline the processes associated with leasing real estate properties. It handles tenant management, lease contracts, payments, and communication with automation features such as flows, approval processes, and email alerts.

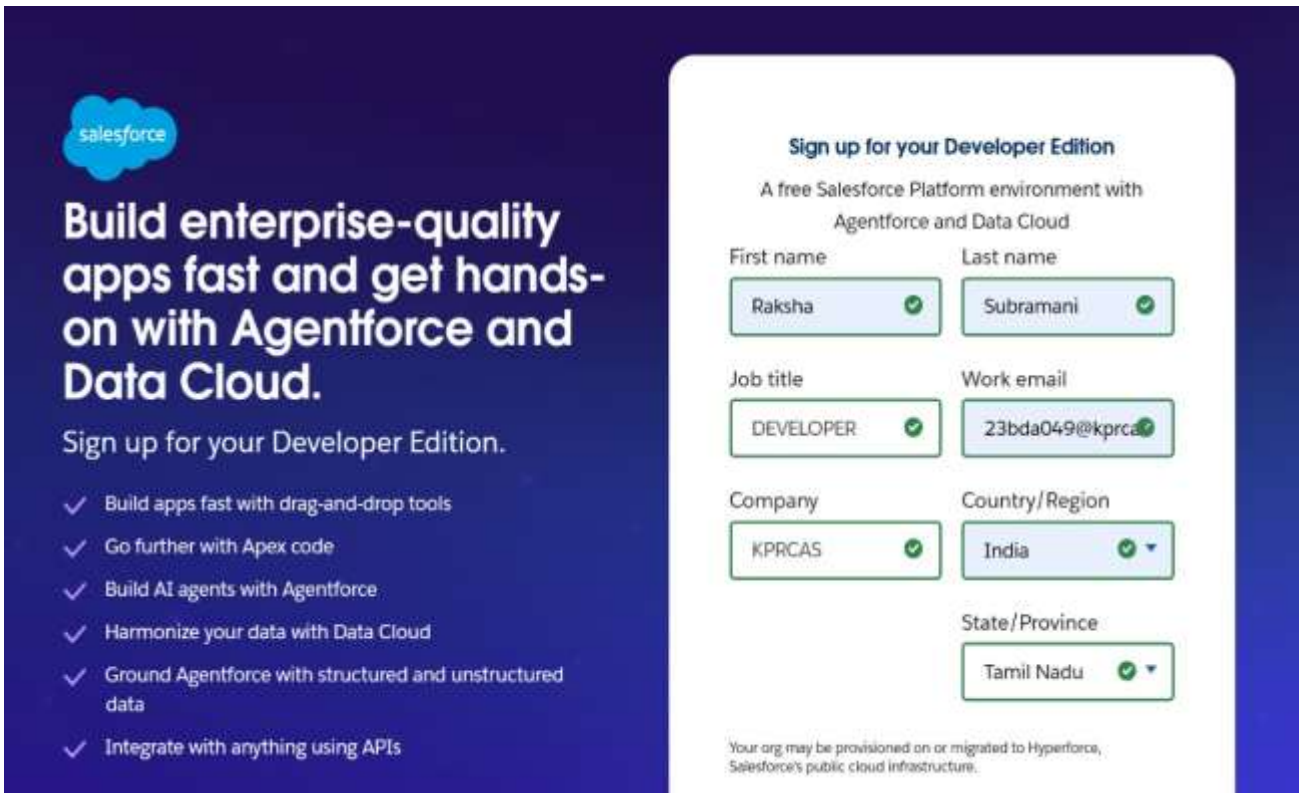
1.2 Purpose

The main objective of the project is to enable organizations to efficiently manage properties, tenants, and lease-related activities. It reduces manual intervention, improves accuracy, and ensures better compliance and communication.

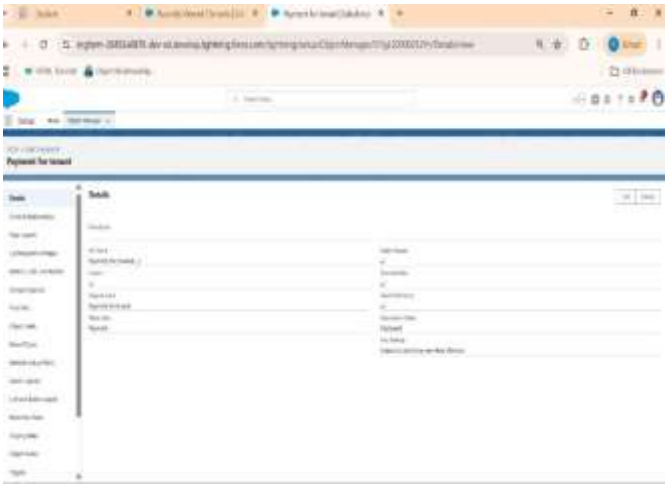
DEVELOPMENT PHASE

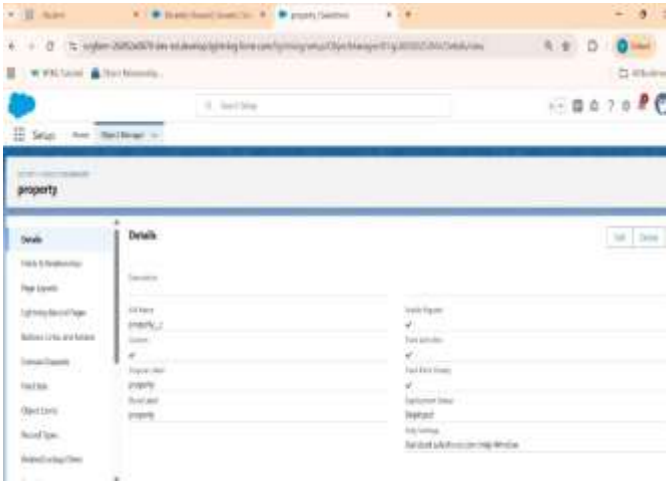
Creating Developer Account:

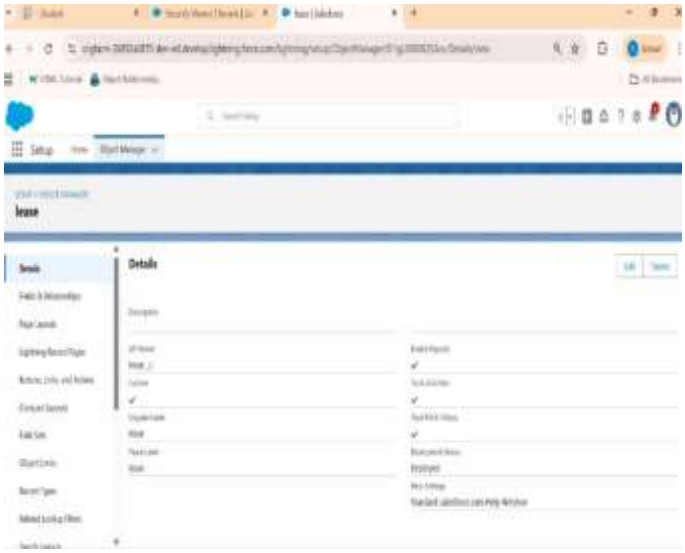
By using this URL - **https://www.salesforce.com/form/developer-signup/?d=pb**

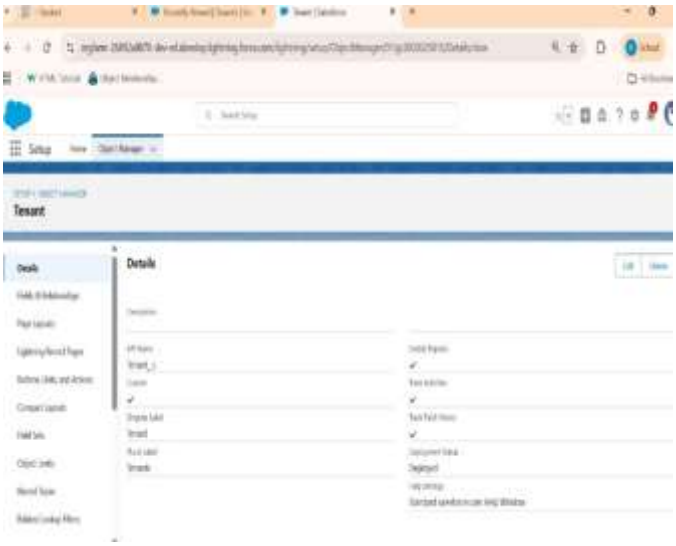


- Created objects: Property, Tenant, Lease, Payment

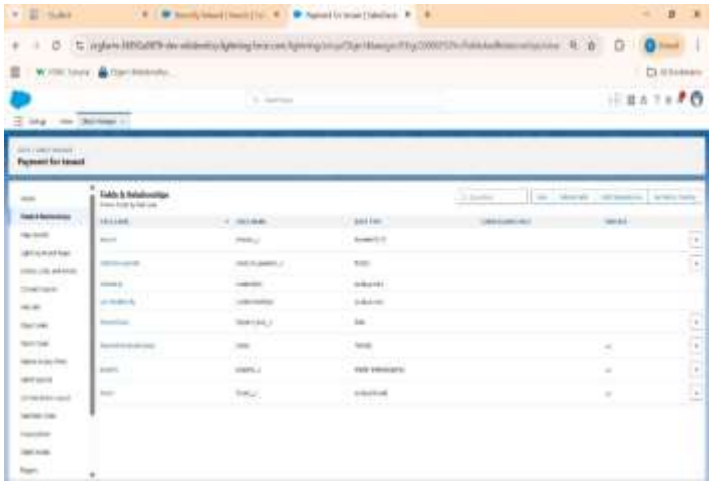


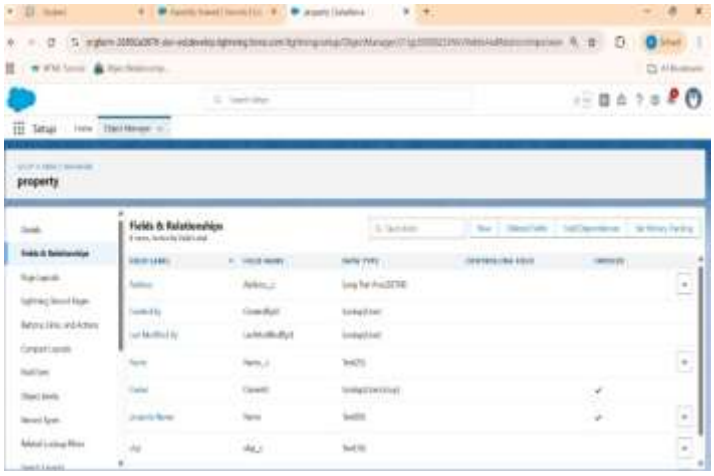


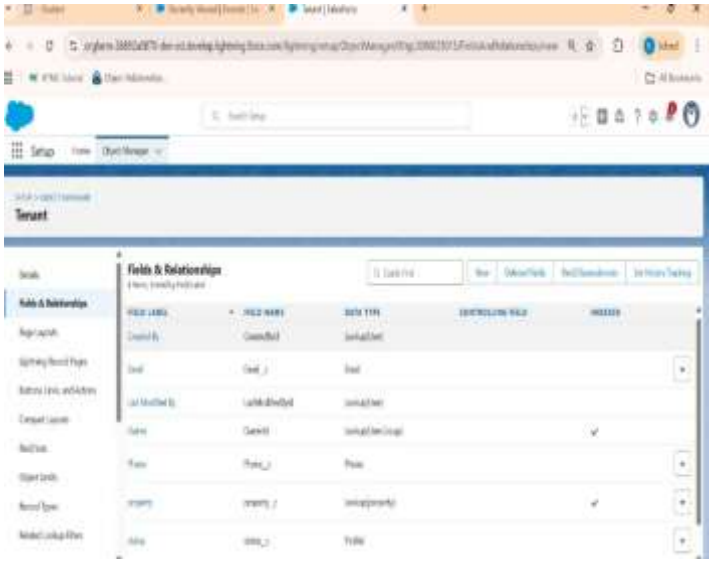


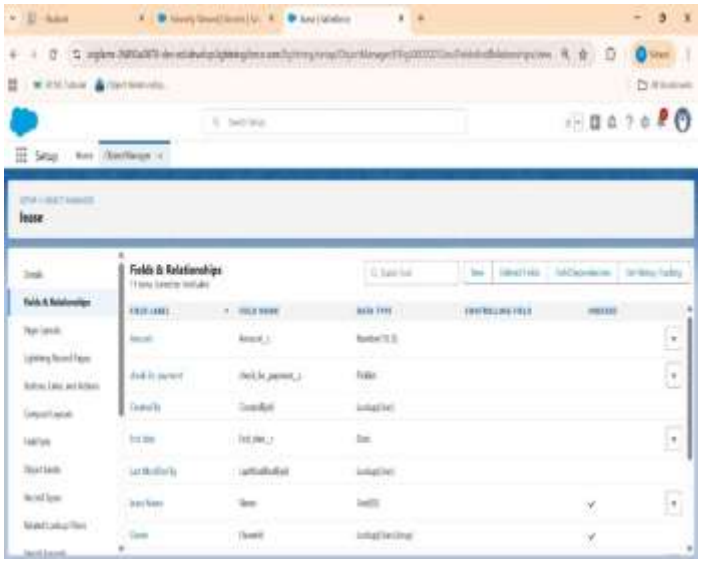


- Configured fields and relationships

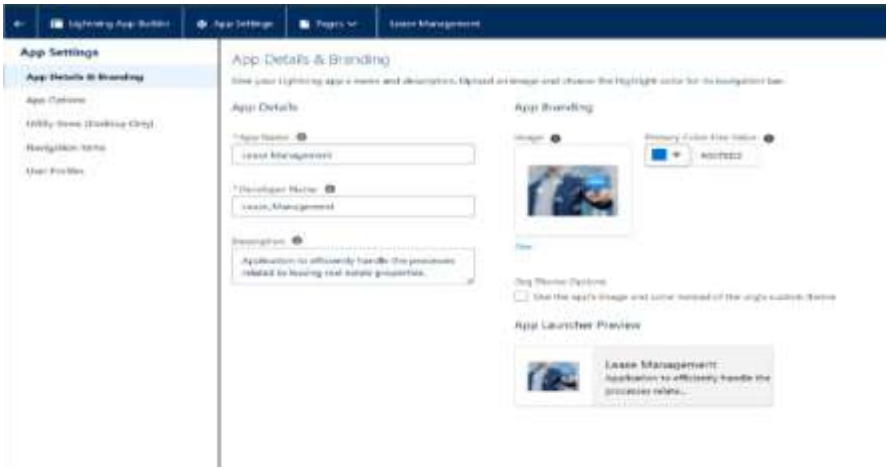


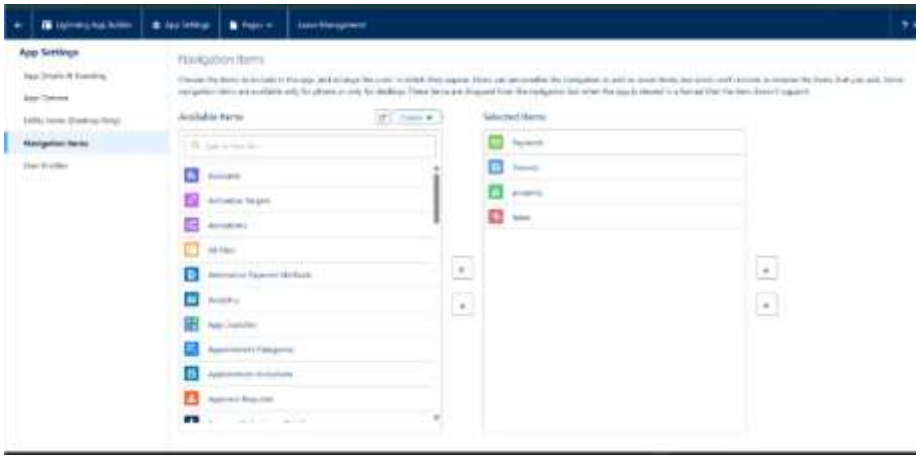


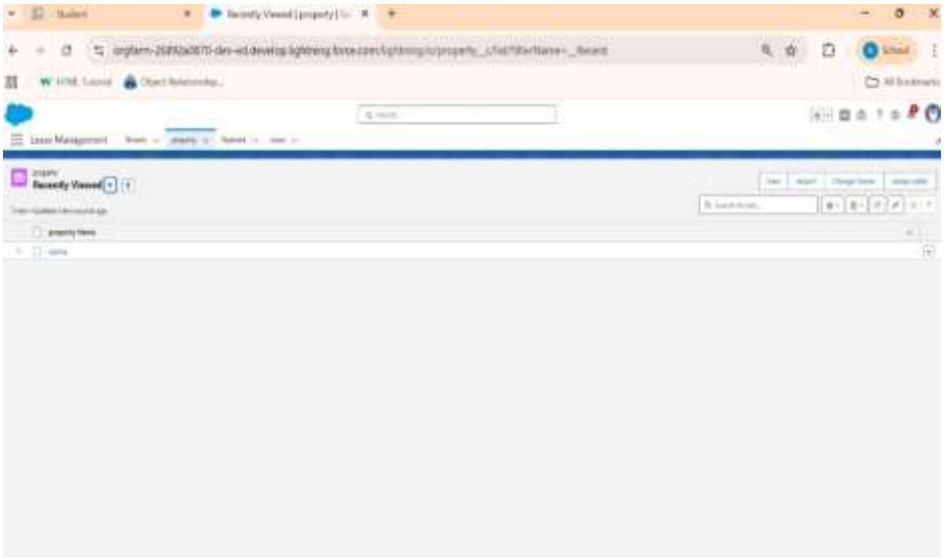




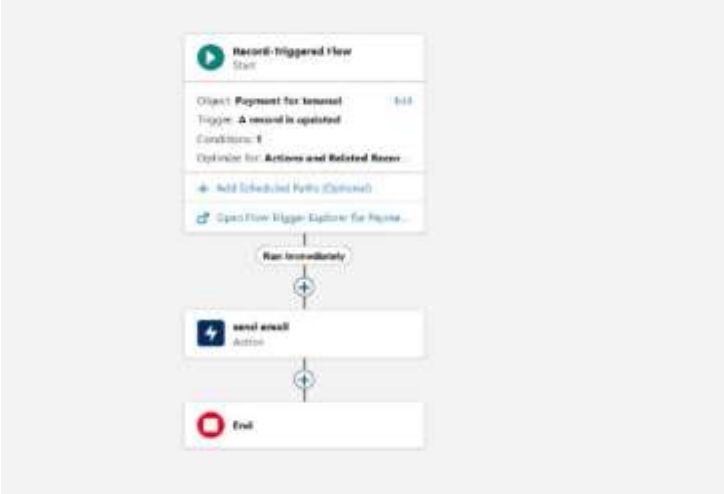
- Developed Lightning App with relevant tabs



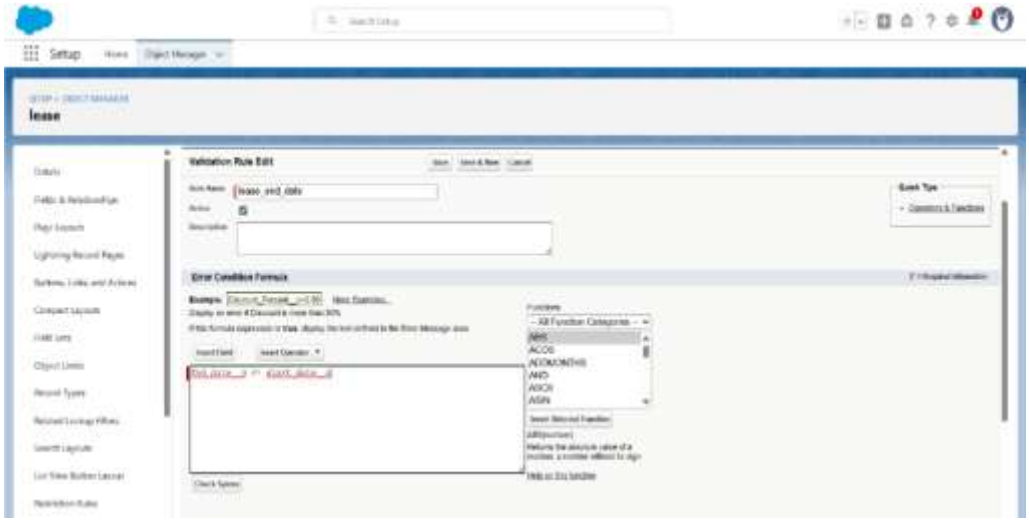




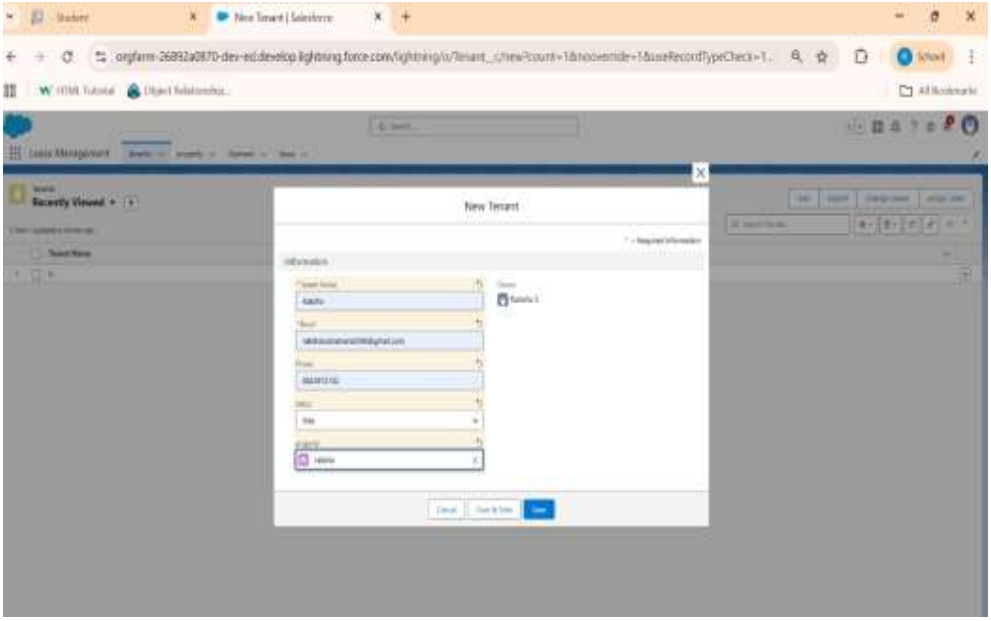
- Implemented Flows for monthly rent and payment success



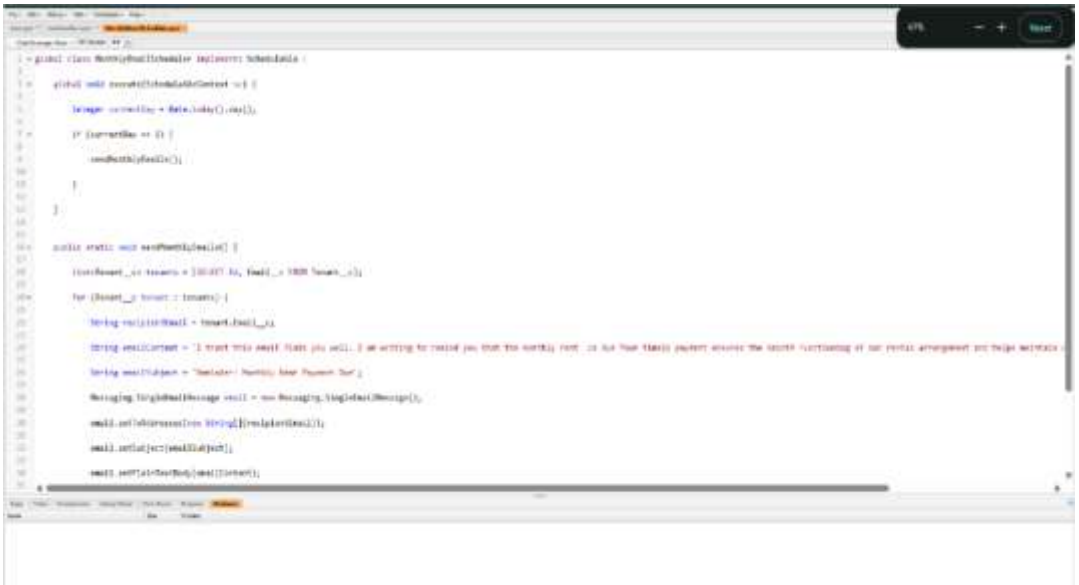
- To create a validation rule to a Lease Object



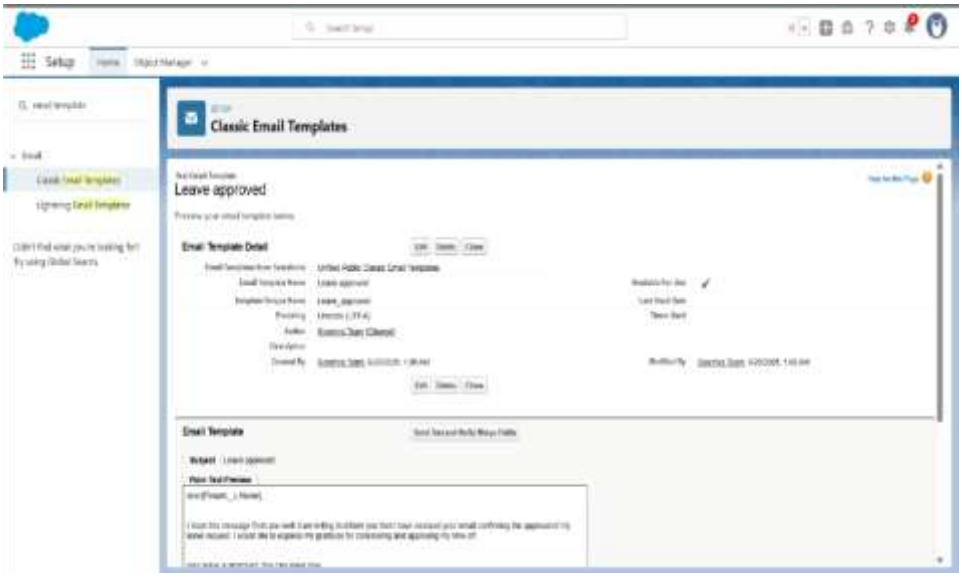
- Added Apex trigger to restrict multiple tenants per property

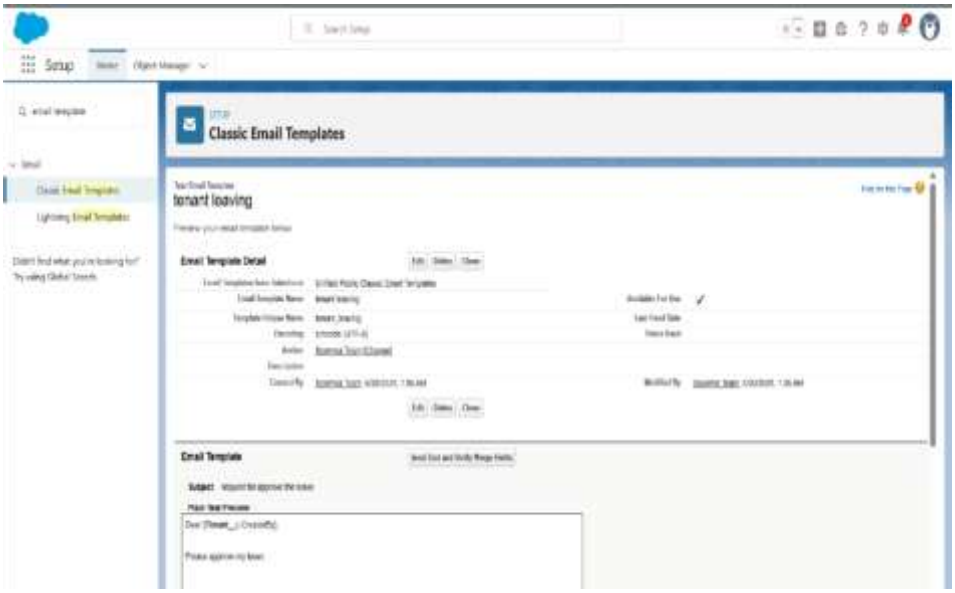


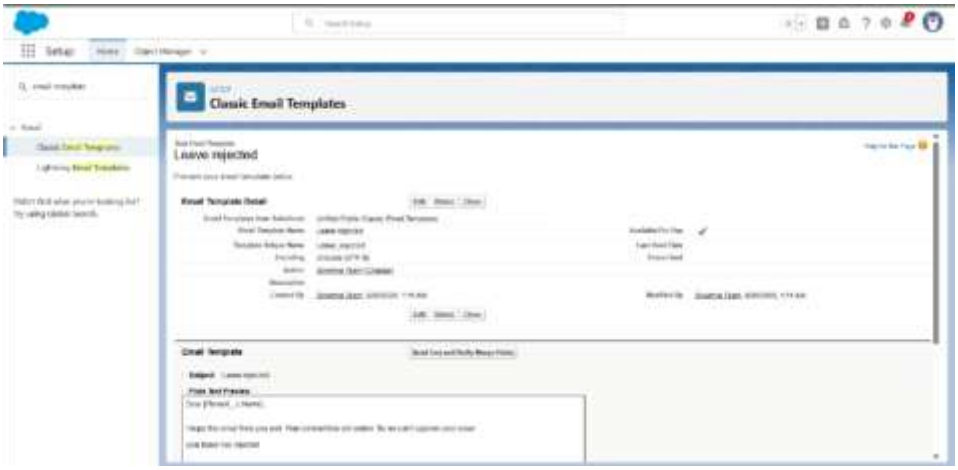
- Scheduled monthly reminder emails using Apex class

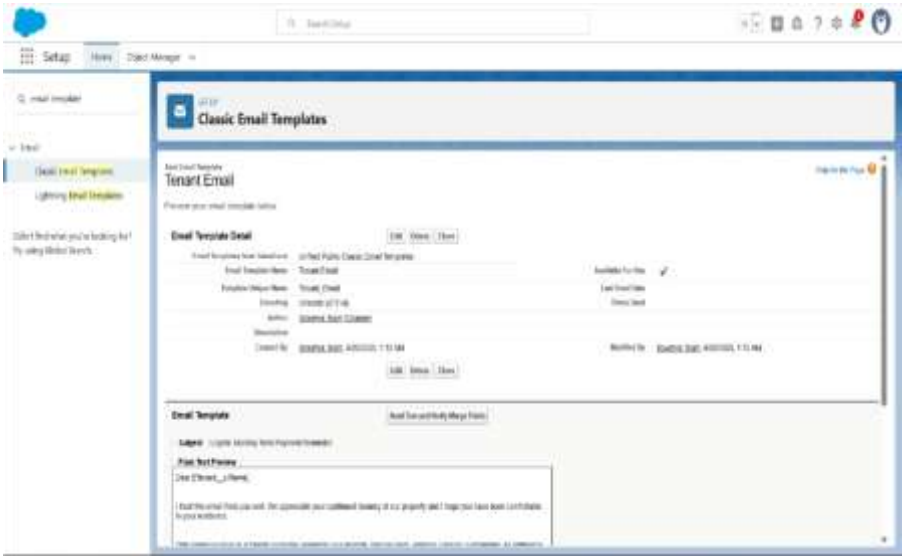


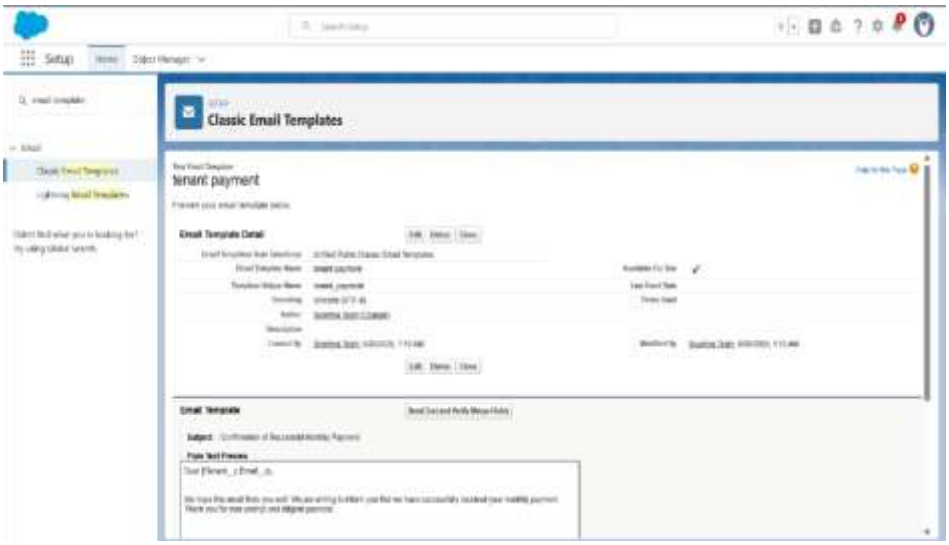
- Built and tested email templates for leave request, approval, rejection, payment, and reminders

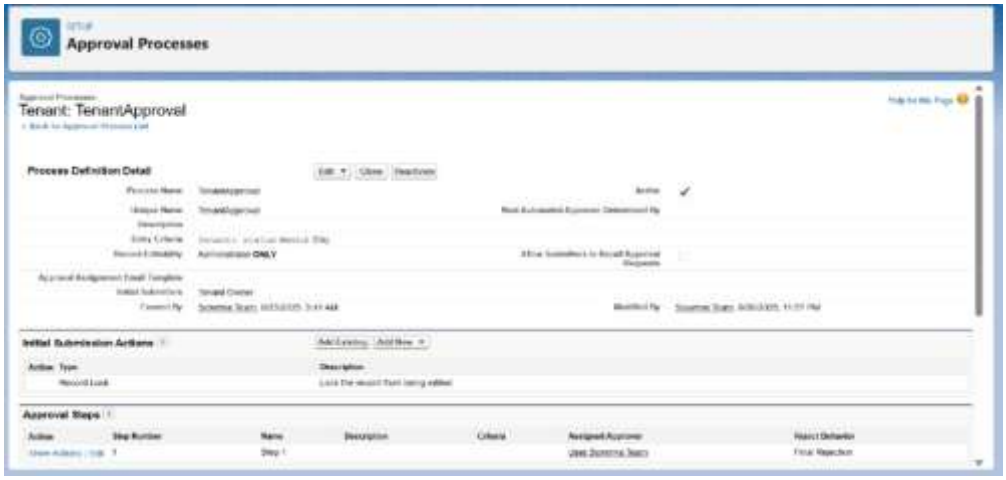





Setup
Home
Object Manager
Classic Email Templates
Leave rejected
Email Template Detail
Email Template

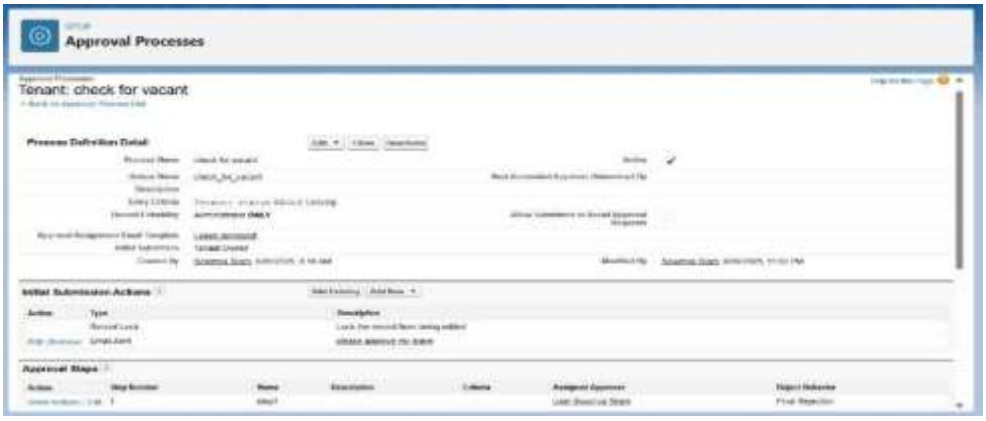

Setup
Home
Object Manager
Classic Email Templates
Tenant Email
Email Template Detail
Email Template

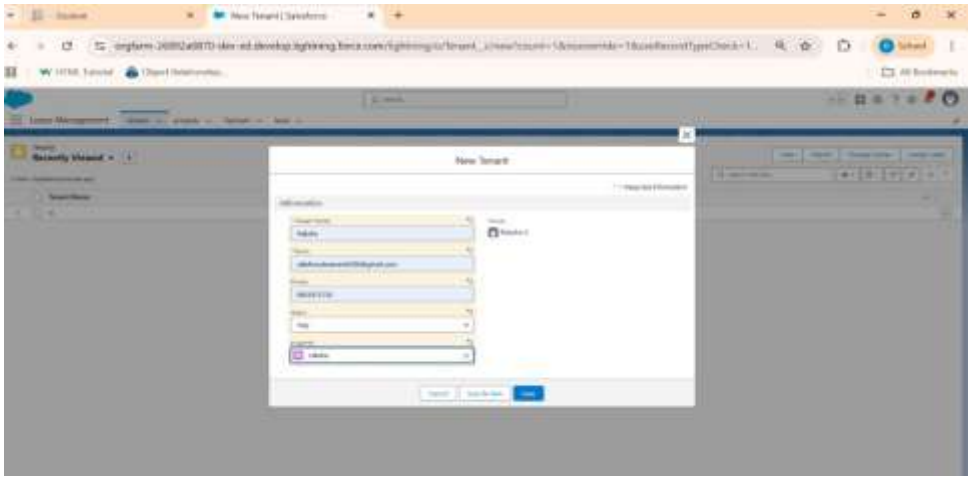

Setup
Home
Object Manager
Classic Email Templates
tenant payment
Email Template Detail
Email Template



Approval Process creation

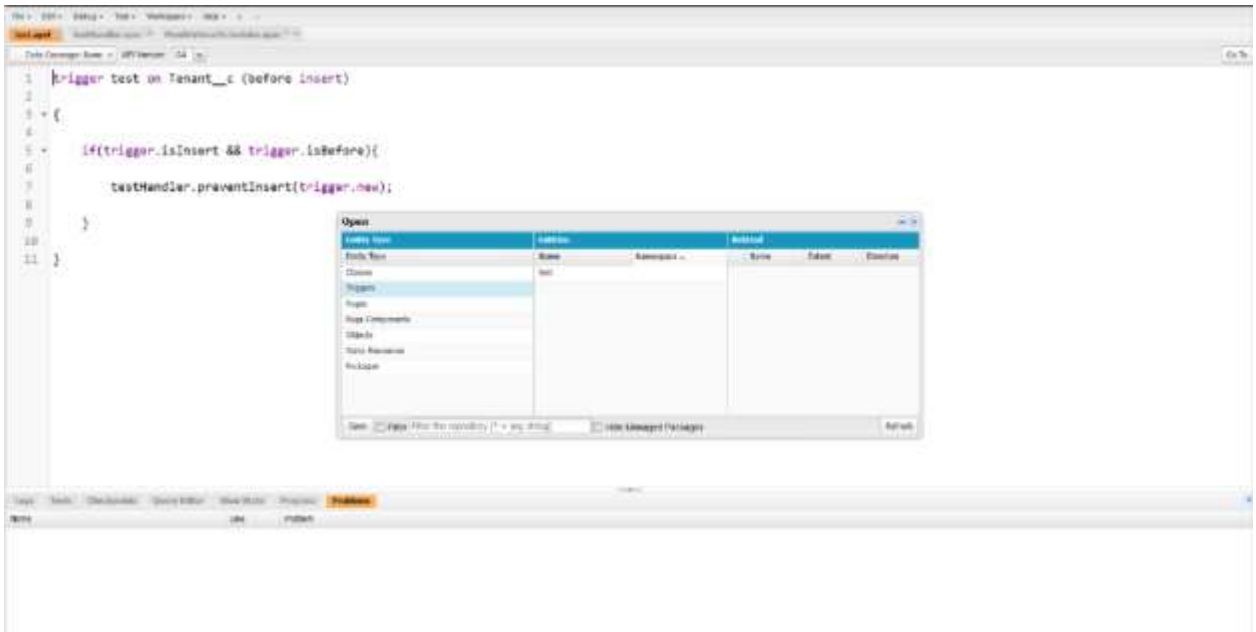
For Check for Vacant:



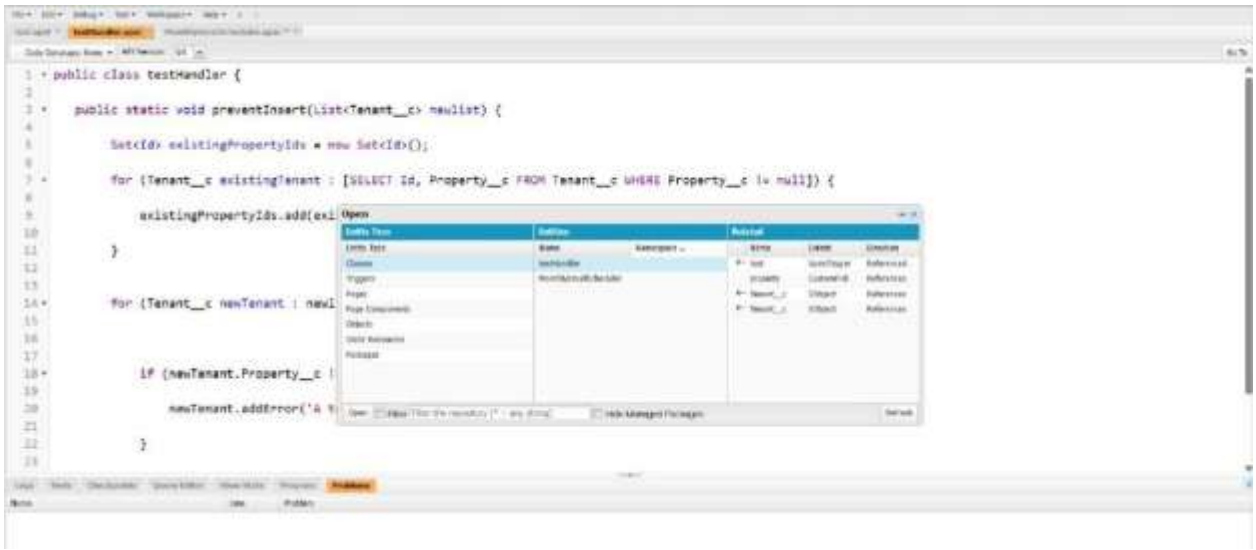


Apex Trigger

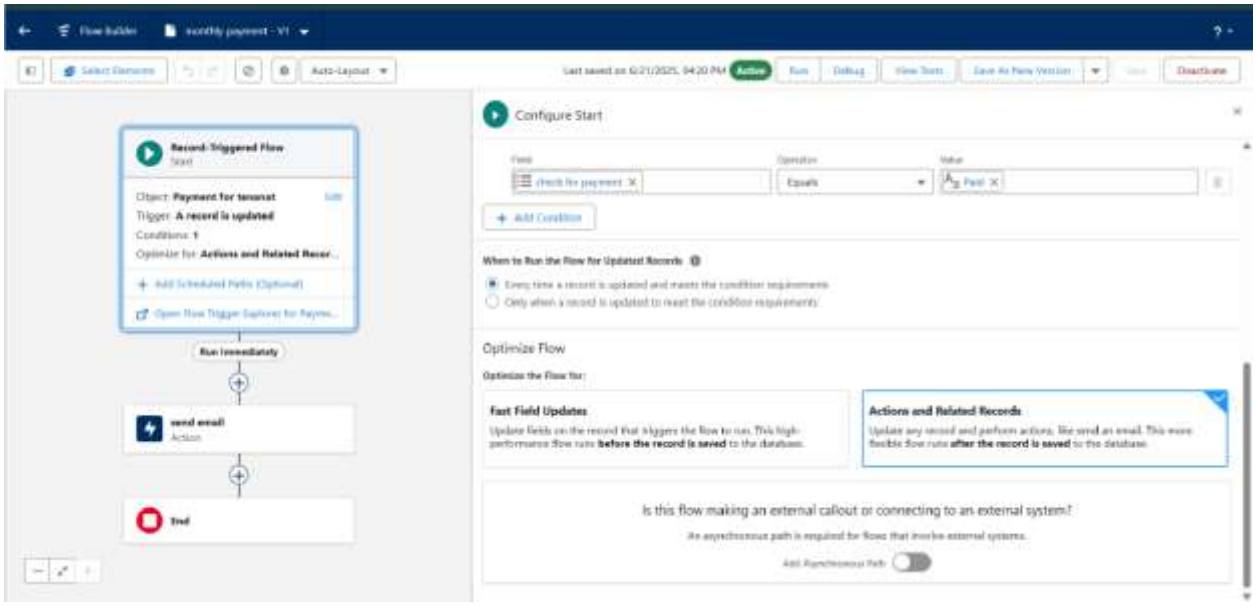
Create an Apex Trigger

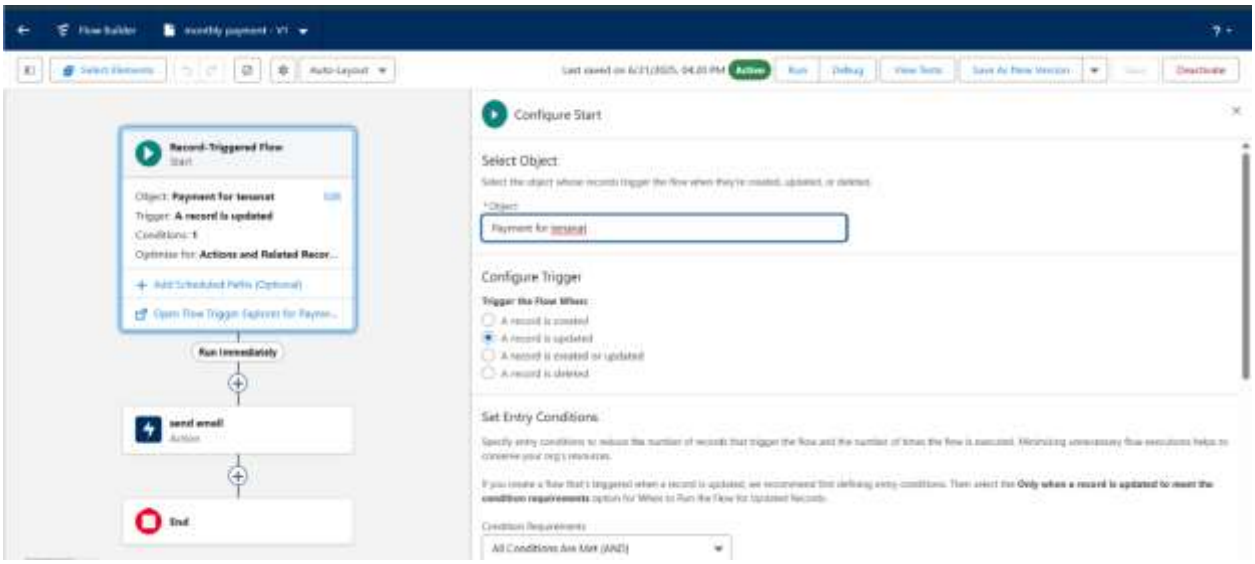


Create an Apex Handler class



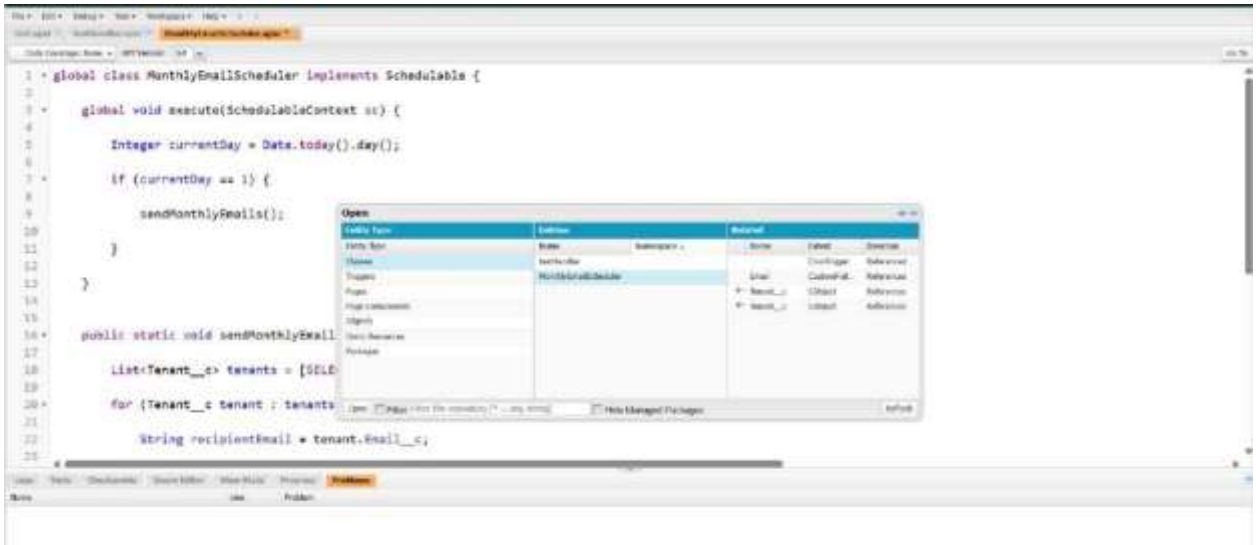
FLOWS


Flow Builder
monthly payment - V1
Auto-Layout
Record-Triggered Flow
Start
Object: Payment for tenant
Trigger: A record is updated
Conditions: 1
Optimize for: Actions and Related Recor...
Add Scheduled Paths (Optional)
Open Flow Trigger Explorer for Payme...
Run immediately
send email
Action
End
Configure Start
Field
Operator
Value
check for payment
Equals
Paid
Add Condition
When to Run the Flow for Updated Records
Every time a record is updated and meets the condition requirements
Only when a record is updated to meet the condition requirements
Optimize Flow
Optimize the Flow for:
Fast Field Updates
Update fields on the record that triggers the flow to run. This high-performance flow runs before the record is saved to the database.
Actions and Related Records
Update any record and perform actions, like send an email. This more flexible flow runs after the record is saved to the database.
Is this flow making an external callout or connecting to an external system?
An asynchronous path is required for flows that involve external systems.
Add Asynchronous Path

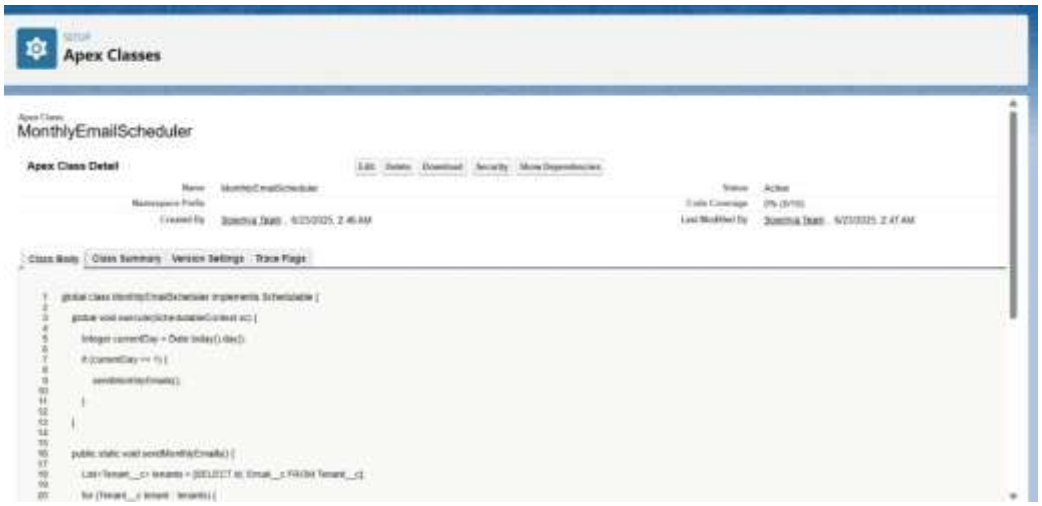

Flow Builder
monthly payment - V1
Auto-Layout
Record-Triggered Flow
Start
Object: Payment for tenant
Trigger: A record is updated
Conditions: 1
Optimize for: Actions and Related Recor...
Add Scheduled Paths (Optional)
Open Flow Trigger Explorer for Payme...
Run immediately
send email
Action
End
Configure Start
Select Object
Select the object whose records trigger the flow when they're created, updated, or deleted.
*Object
Payment for tenant
Configure Trigger
Trigger the Flow When:
A record is created
A record is updated
A record is created or updated
A record is deleted
Set Entry Conditions
Specify entry conditions to reduce the number of records that trigger the flow and the number of times the flow is executed. Minimizing unnecessary flow executions helps to conserve your org's resources.
If you create a flow that's triggered when a record is updated, we recommend first defining entry conditions. Then select the Only when a record is updated to meet the condition requirements option for When to Run the Flow for Updated Records.
Condition Requirements
All Conditions Are Met (AND)

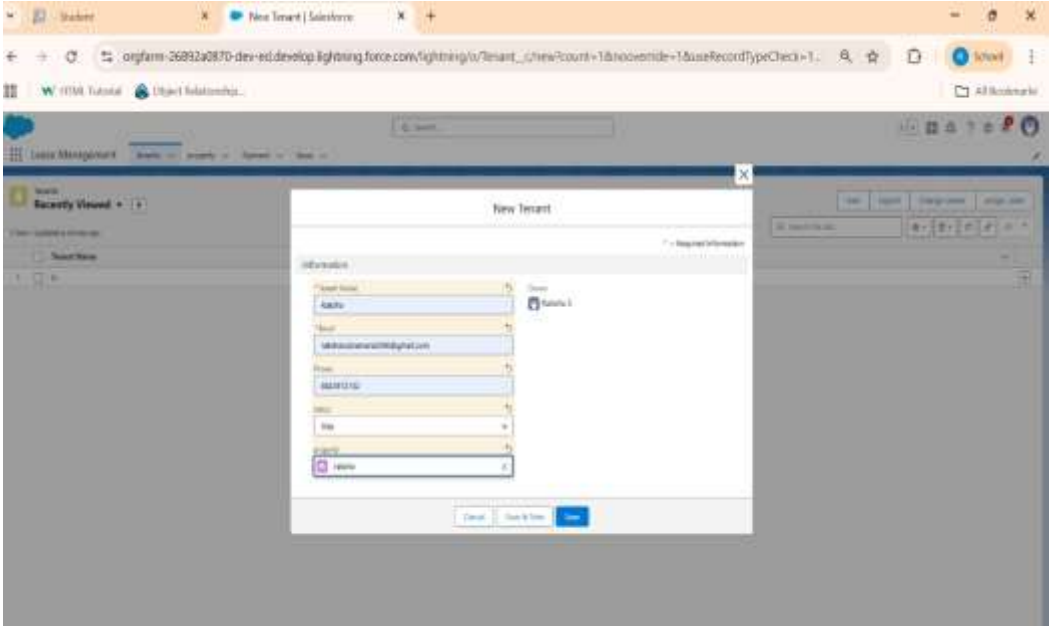
- Schedule class:

  Create an Apex Class



Schedule Apex class

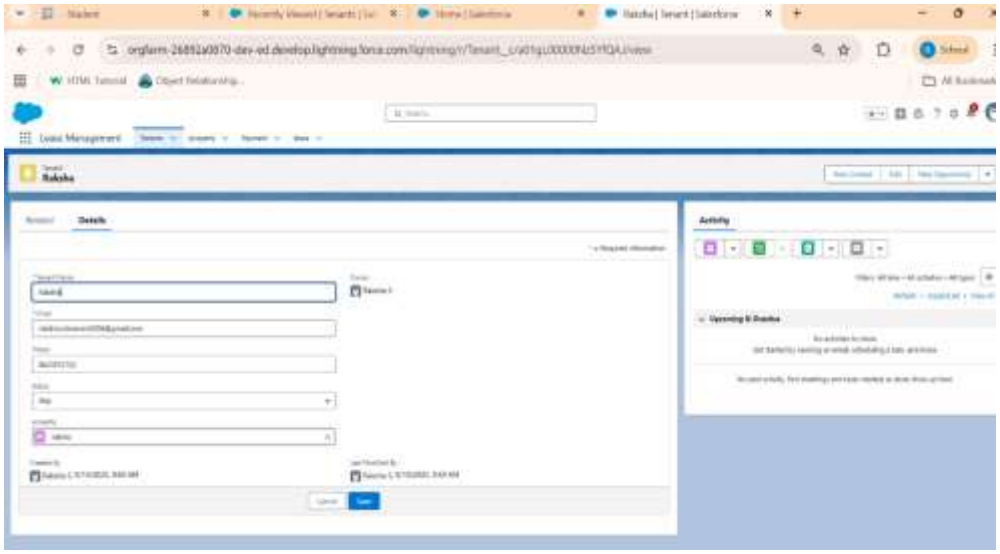




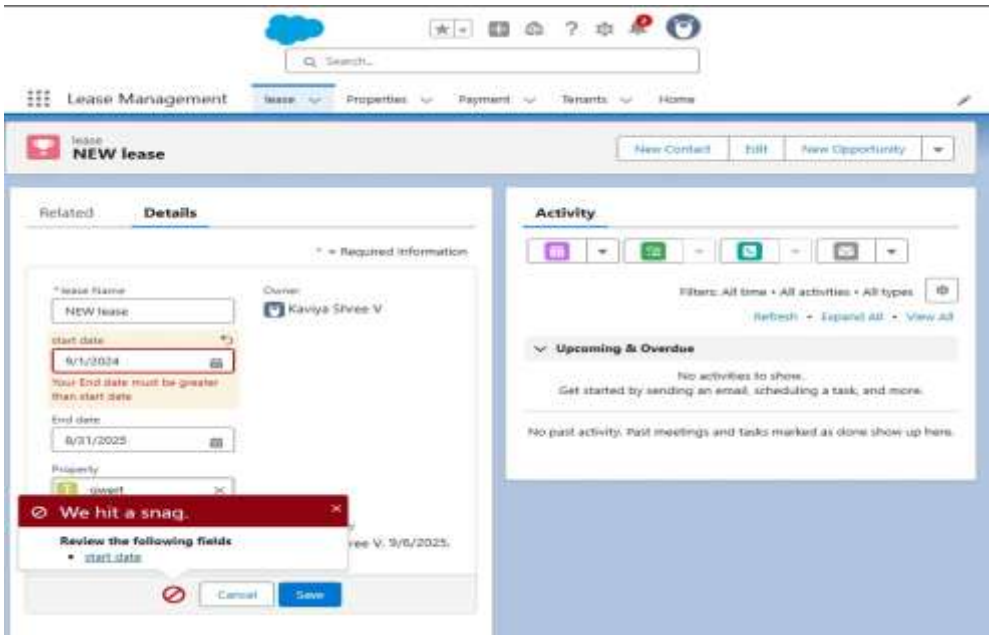
FUNCTIONAL AND PERFORMANCE TESTING

Performance Testing

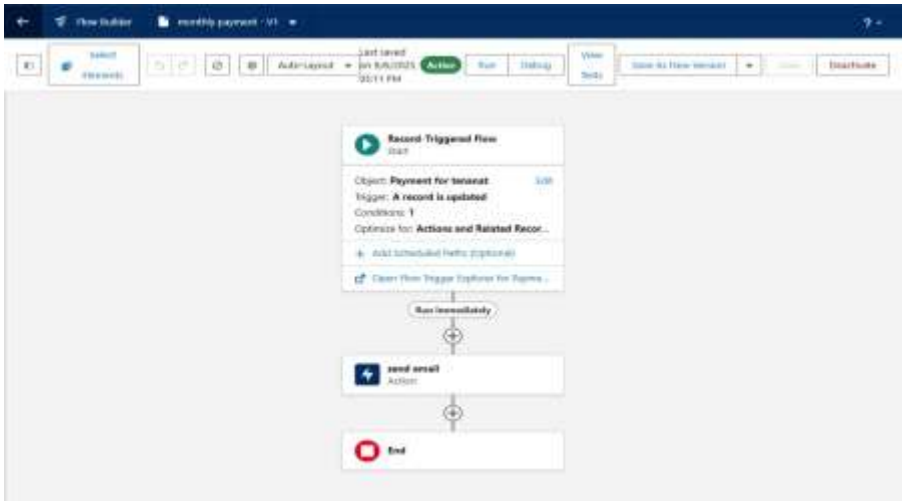
- Trigger validation by entering duplicate tenant-property records

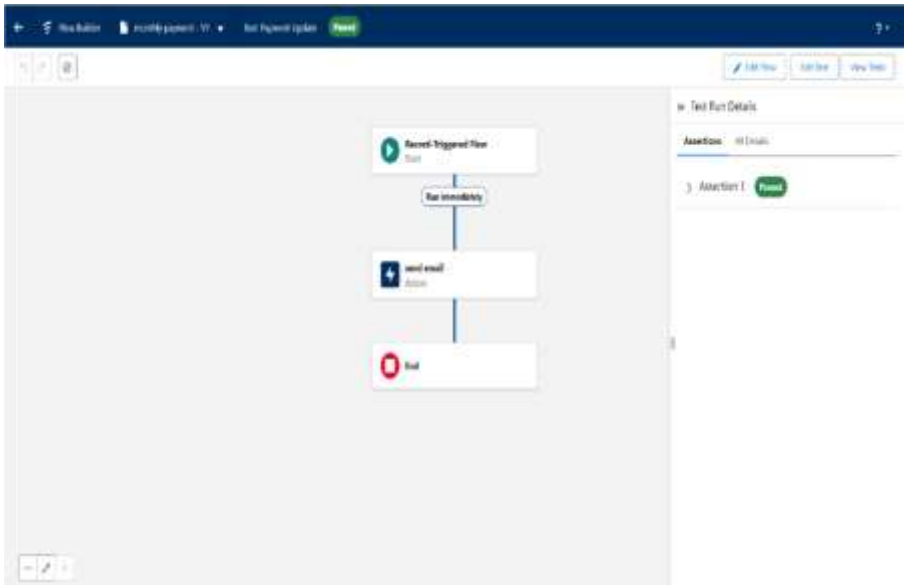


- Validation Rule checking

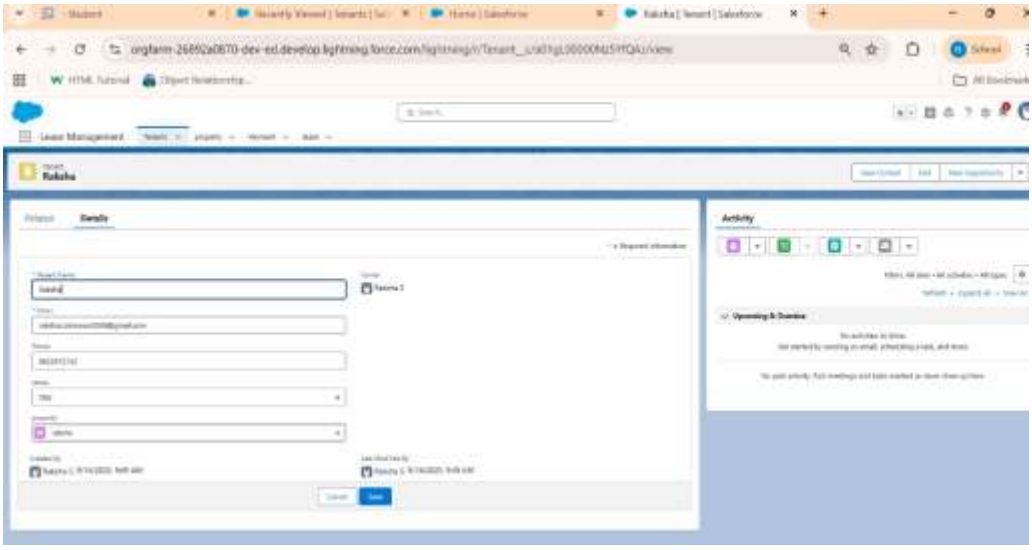


- Test flows on payment update





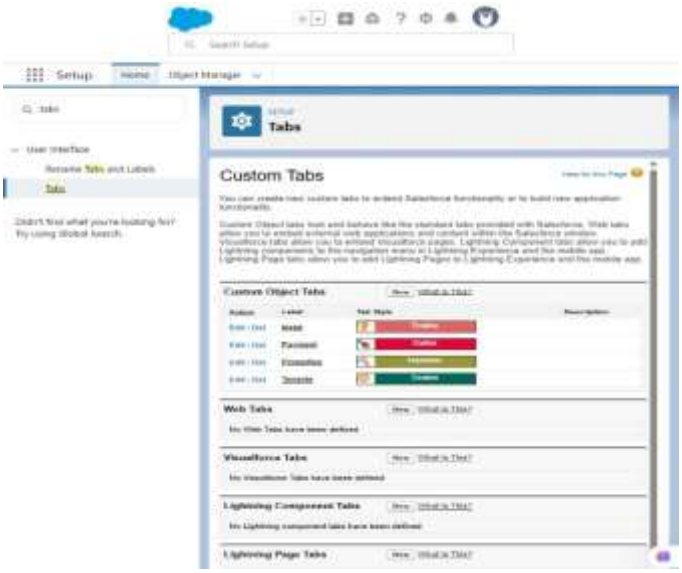
- Approval process validated through email alerts and status updates



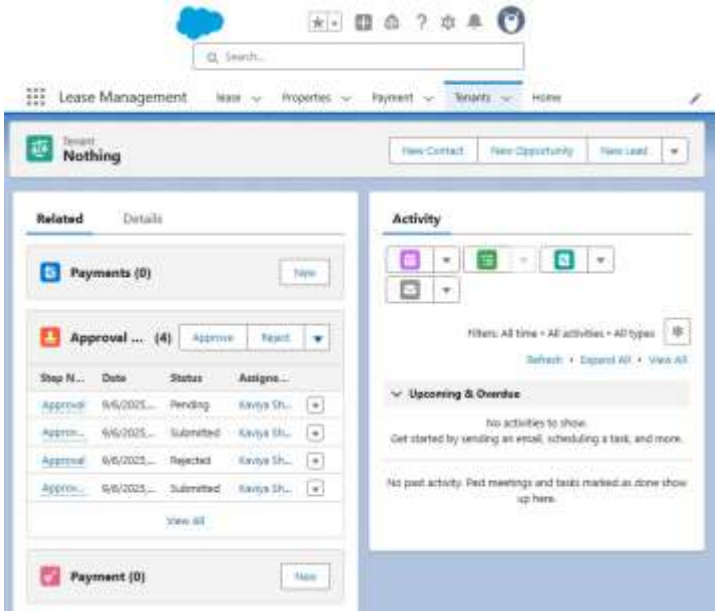
RESULTS

Output Screenshots

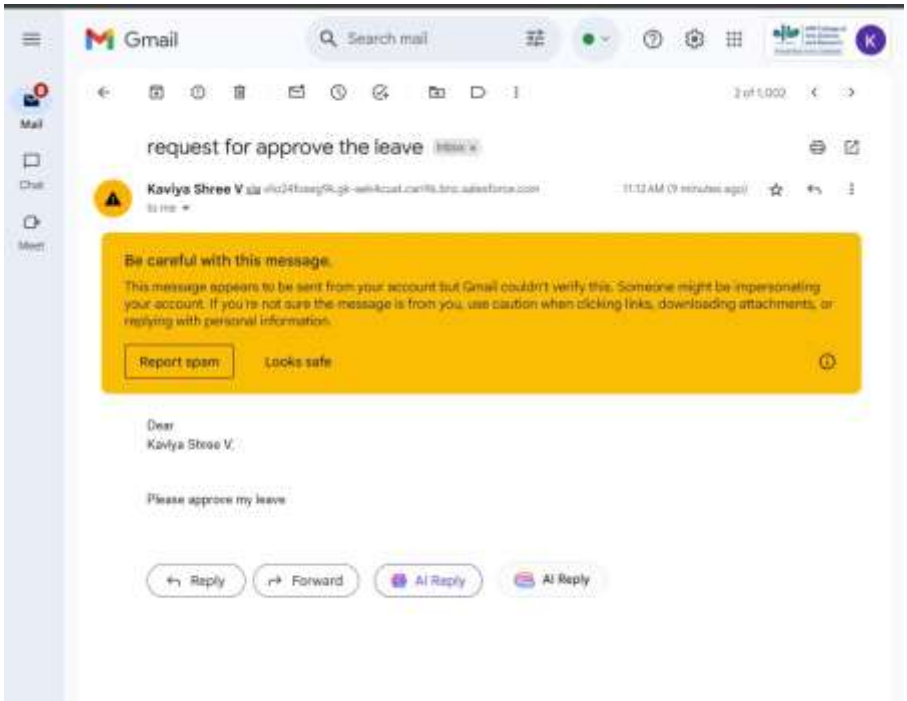
- Tabs for Property, Tenant, Lease, Payment



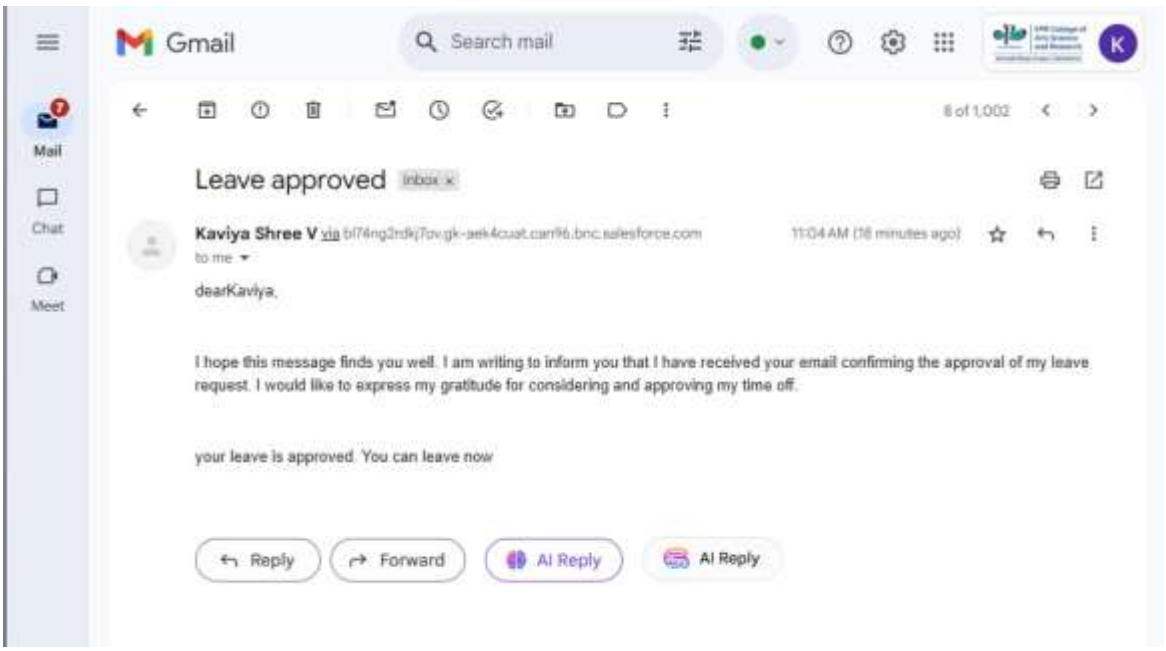
- Email alerts



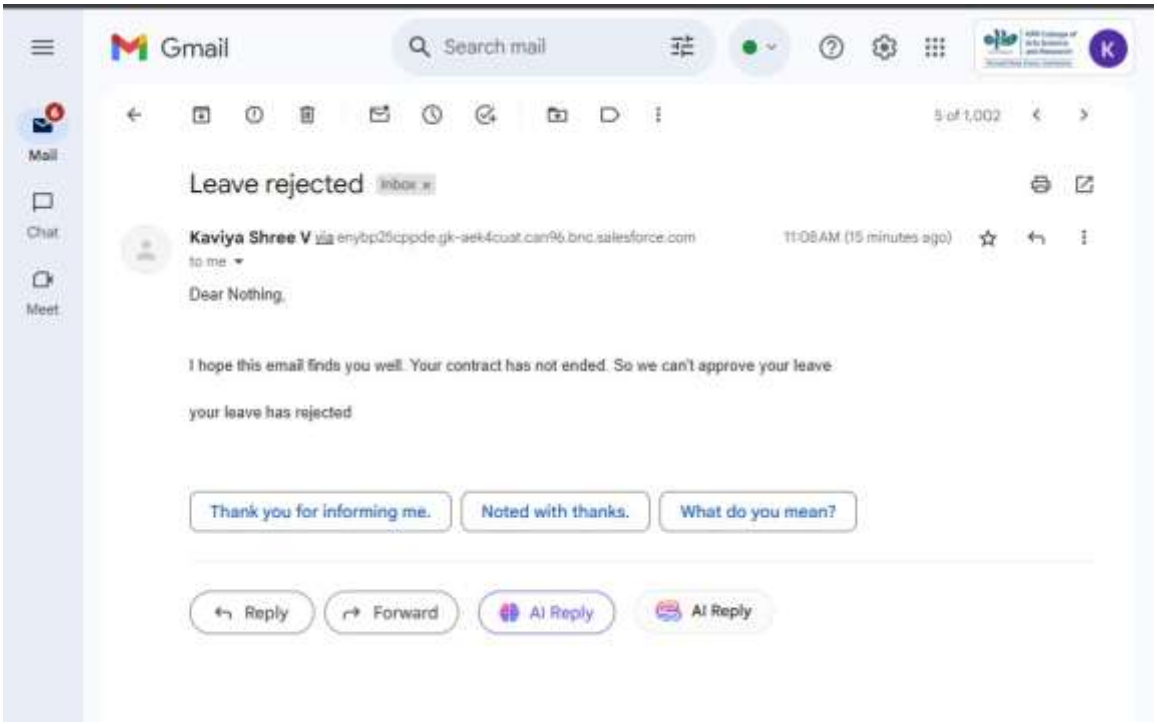
- Request for approve the leave



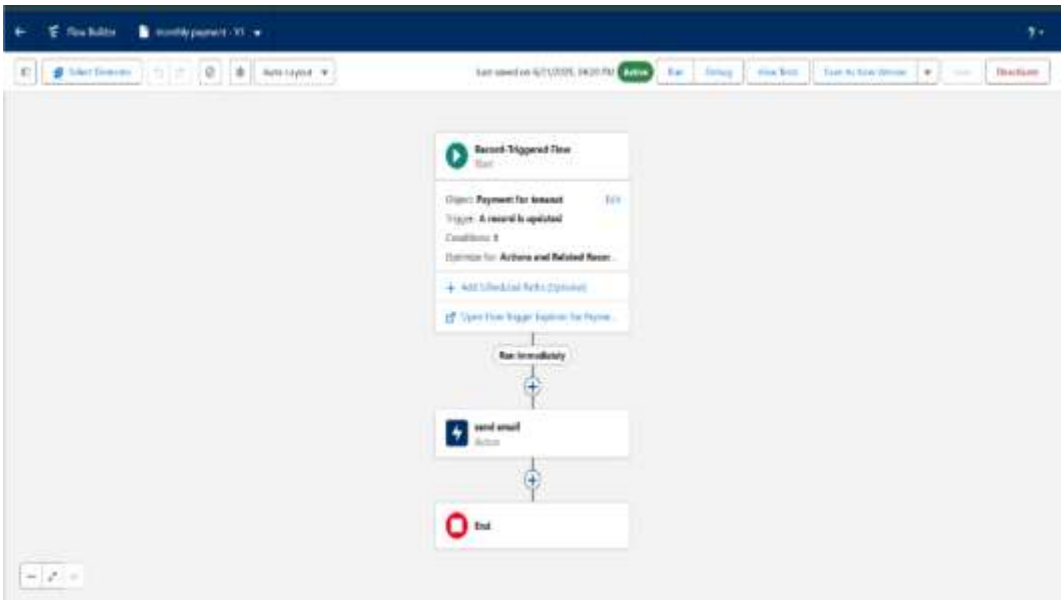
- Leave approved



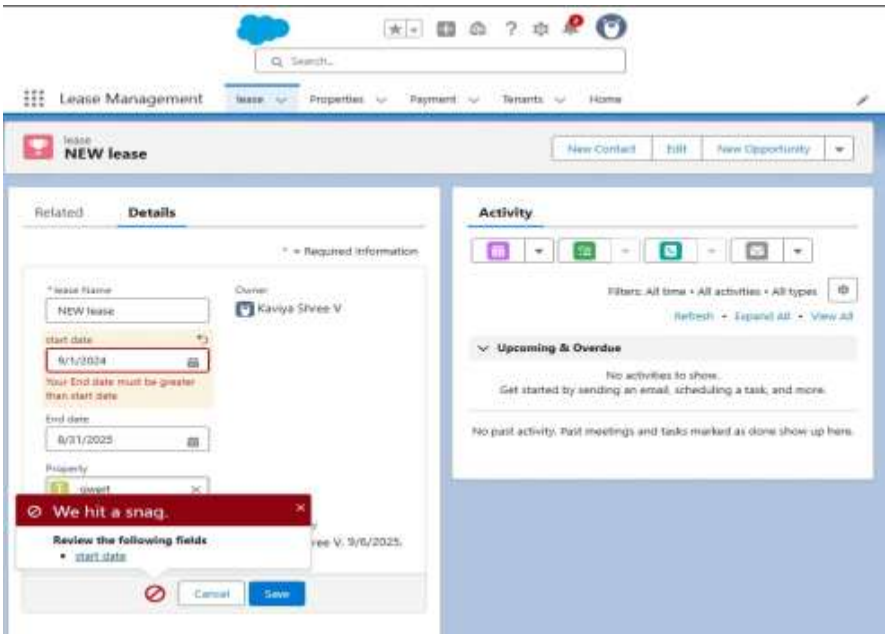
➢ Leave rejected



- Flow runs



- Trigger error messages



CONCLUSION

The Lease Management System successfully streamlines the operations of leasing through a structured, automated Salesforce application. It improves efficiency, communication, and data accuracy for both admins and tenants.

APPENDIX

- **Source Code:** Provided in Apex Classes and Triggers

Test.apxt: trigger test on Tenant__c

(before insert)

{ if (trigger.isInsert &&

trigger.isBefore)

{ testHandler.preventInsert(trigger.new);

}

}

testHandler.apxc:

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');

}

}

}

}
```

MothlyEmailScheduler.apxc:

```
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});

        }

    }

}
```