

**Lease Management**

College: 7155 - PSG Institute of Technology and Applied Research

Team ID: NM2024TMID00663

Team Lead: Harshavardhan K K (53399036FEF77E7BD274B6451DCEC1DA)

Team Members:

|  |  |
| --- | --- |
| HARSHAVARDHAN K K | 53399036FEF77E7BD274B6451DCEC1DA |
| KAVIYAVALLI A | F2916DDE85DA574914841EEB3EDF9374 |
| MADHUMITHA S | 54F0A2229B763EBBD834575A393DE970 |
| SANTHOSHWAR R | FA976340C0BC8F9B5C046FDABB82A426 |
| VIGNESHWARAN M | 556695D2FF83B6AA02B42B7A10CA1986 |

1. **Project Overview**

This Lease Management project focuses on optimizing and automating the handling of lease agreements for real estate properties, equipment, or other assets. Built on Salesforce’s platform, it aims to simplify lease management tasks, enhance communication, and maintain regulatory compliance. The system will support efficient tracking, streamlined communication, and accurate record-keeping for all lease transactions, improving both operational workflows and data reliability.

1. **Objectives**

**Business Goals:**

* Develop a streamlined and automated lease management system that enhances tracking, compliance, and tenant relations.
* The project seeks to minimize administrative tasks, strengthen tenant communication, and ensure adherence to lease terms and regulatory standards.

**Specific Outcomes:**

* A centralized dashboard to monitor lease terms, payments, and tenant information.
* Automated alerts and notifications for lease renewals, upcoming payments, and tenant communications.
* Streamlined approval workflows and validation mechanisms to maintain data accuracy and regulatory compliance.

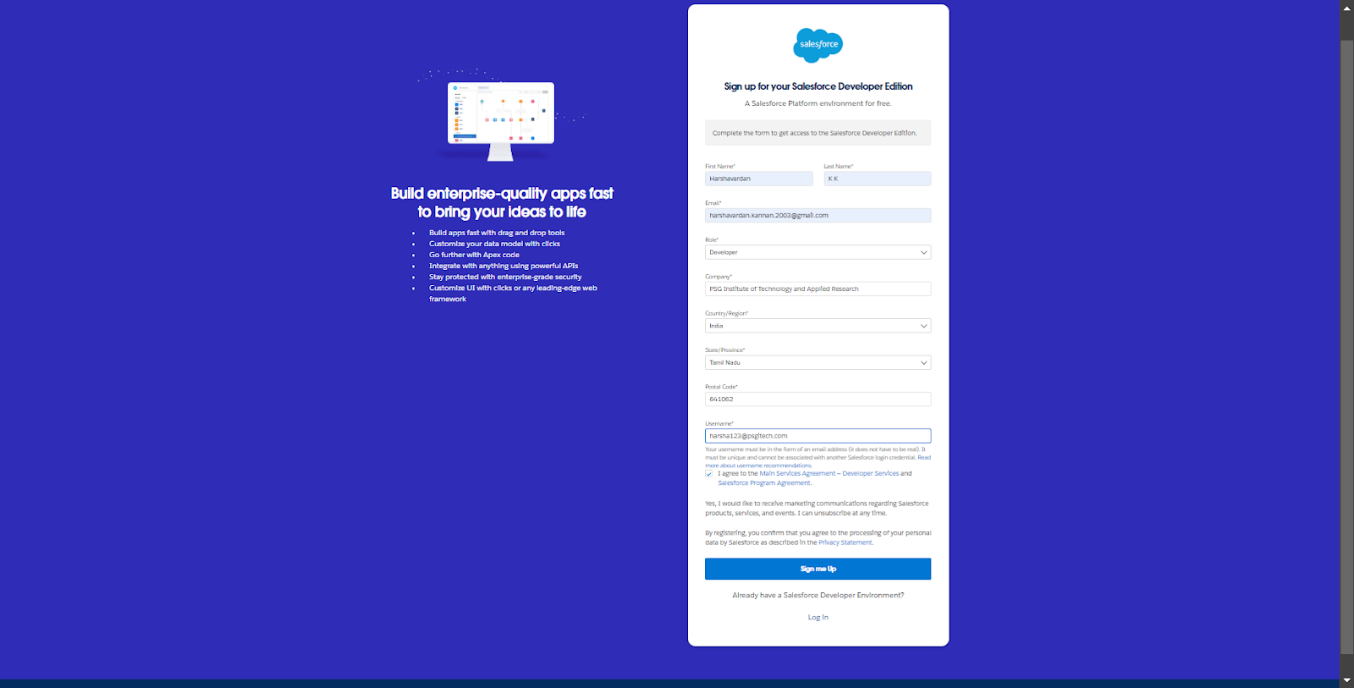
1. **Salesforce Key Features and Concepts Utilized**

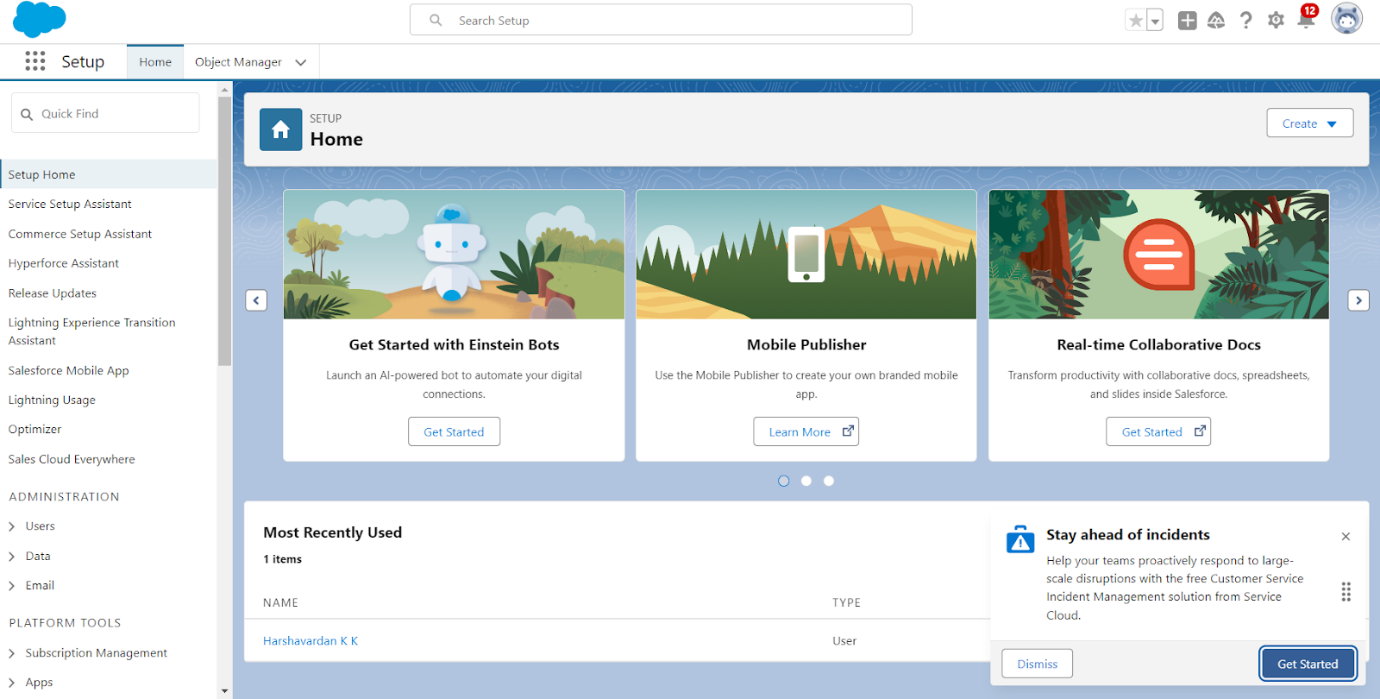
This project leverages essential Salesforce features to build a powerful lease management system:

1. **Custom Objects and Relationships**
   * Created custom objects like Property, Tenant, Lease, and Payment to store crucial lease information.
   * Defined lookup relationships to link lease records with tenants and properties, enabling efficient data access and record association.
2. **Tabs and Lightning App (Lease Management)**
   * Added tabs for each custom object to enhance navigation within the Lease Management Lightning app.
   * Customized the app layout for streamlined functionality and consistent branding.
3. **Validation Rules**
   * Developed validation rules to uphold data integrity, ensuring accurate Lease entries and avoiding incomplete or erroneous submissions.
4. **Email Templates**
   * Created custom email templates for tenant communications, including:
     + Tenant Leaving Notification
     + Leave Approval/Rejection
     + Monthly Payment Reminder
     + Payment Confirmation
5. **Approval Process**
   * Configured an approval process for vacant properties:
     + **Initial Submission Action:** Sends for review upon submission.
     + **Final Approval Action:** Updates property status to leased and initiates tenant communication.
     + **Final Rejection Action:** Notifies tenant with rejection details.
6. **Apex Triggers and Classes**
   * Developed an Apex Trigger for automating key lease tasks.
   * Created an Apex handler class for functions like lease term validation and payment due date calculations.
7. **Flows**
   * Designed a Monthly Payment Flow for streamlined tenant payment processing, with reminders and status tracking.
8. **Scheduled Apex Class**
   * Built an Apex class for scheduled tasks, automating monthly billing reminders and payment validations for timely notifications and compliance.
9. **Detailed Steps to Solution Design**

1. **Created Salesforce Developer Account**

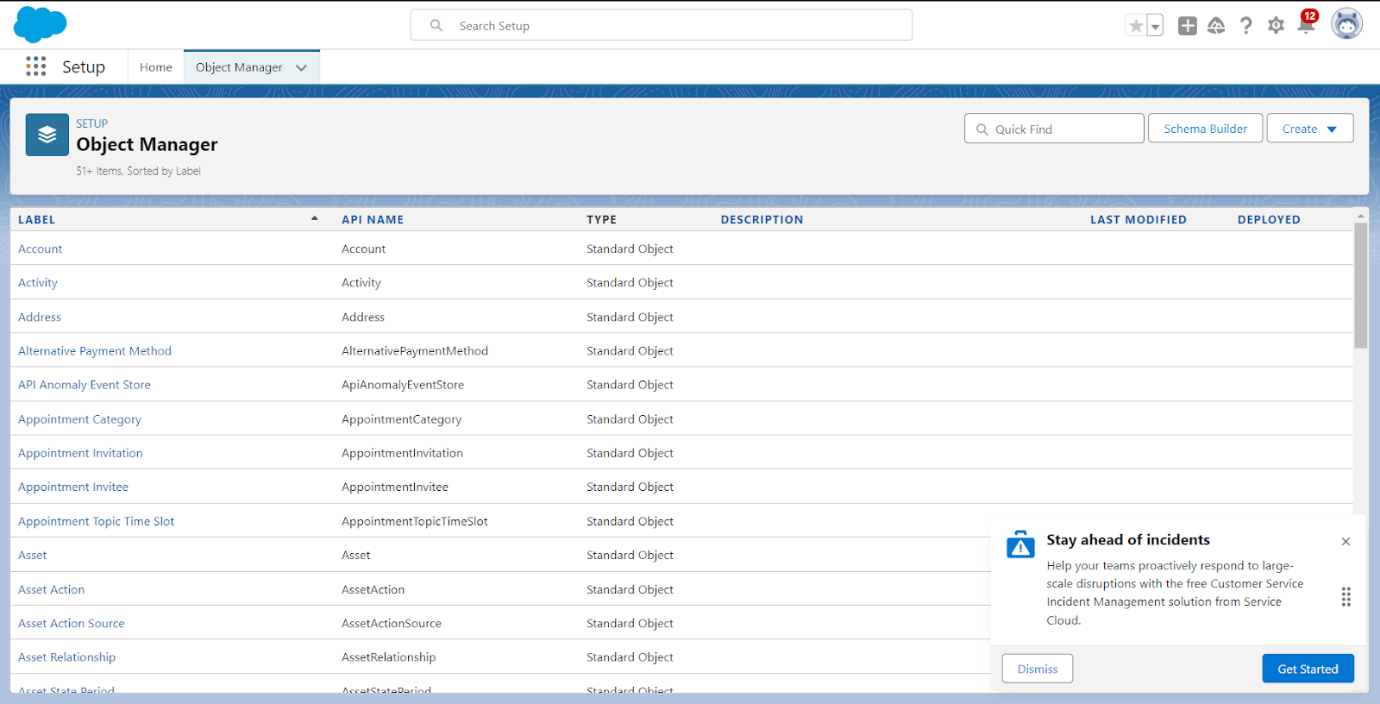
* We began by creating a Salesforce Developer Account to access a development environment where we could build and customize the application. This setup provided the necessary tools to design, test, and deploy the application.

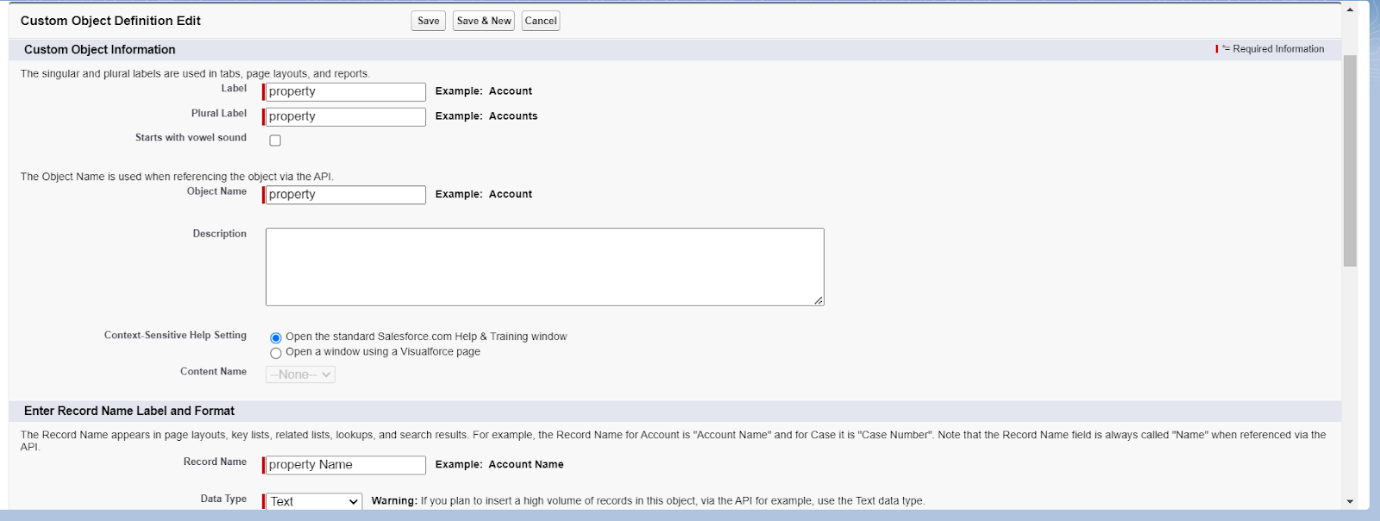


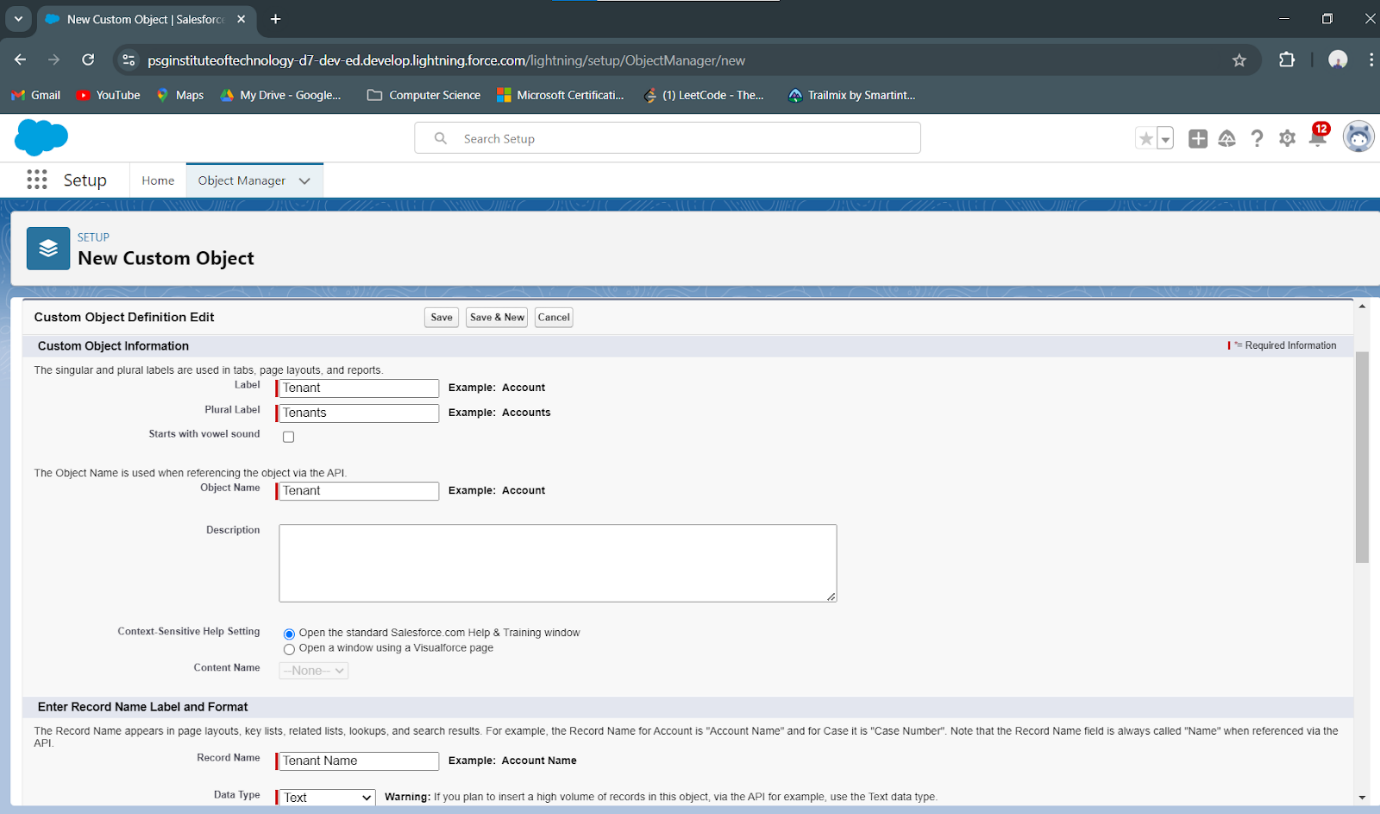


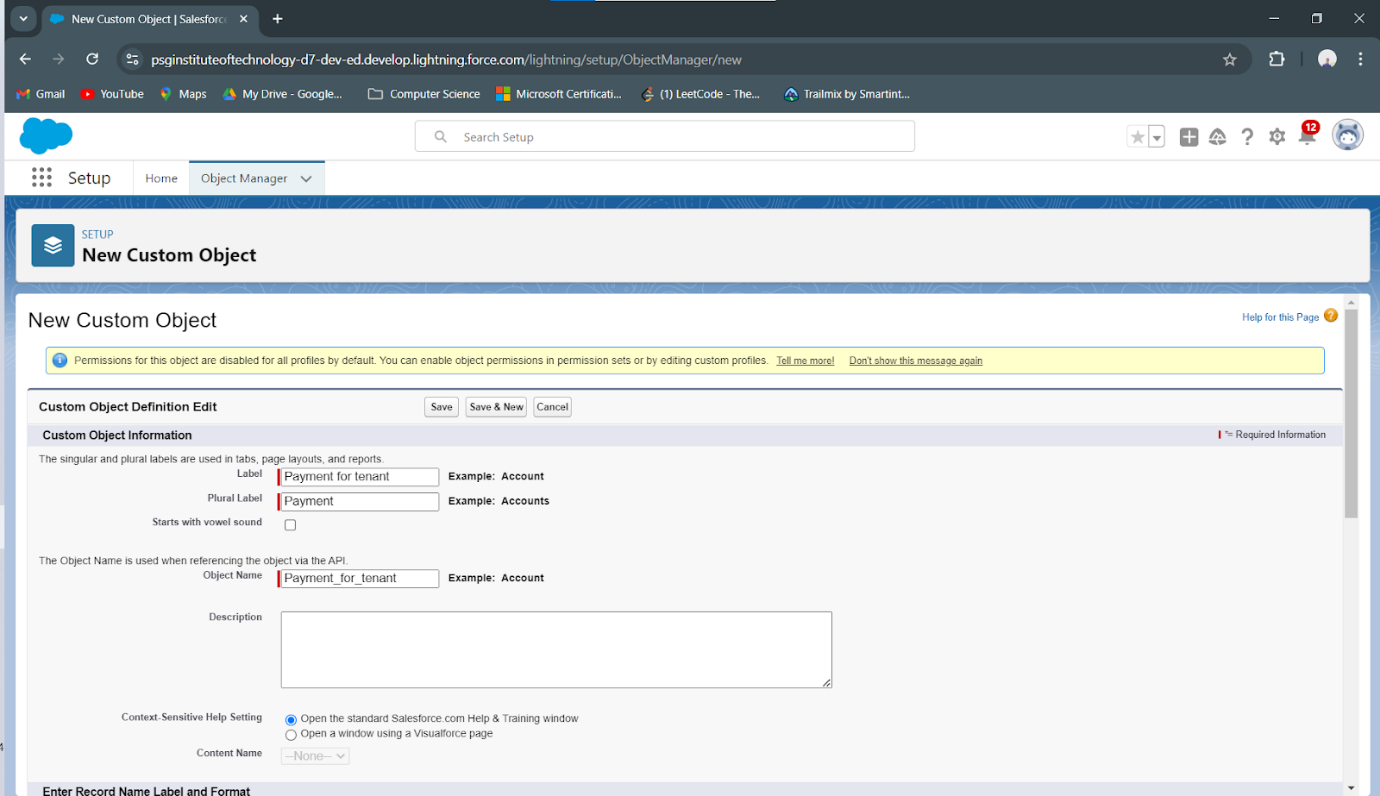
2. **Object Definitions**

* Defined primary objects: **Property**, **Tenant**, **Lease**, and **Payment** within Object Manager.
* Each object was designed to store specific data: Property details, tenant information, lease agreements, and payment records.



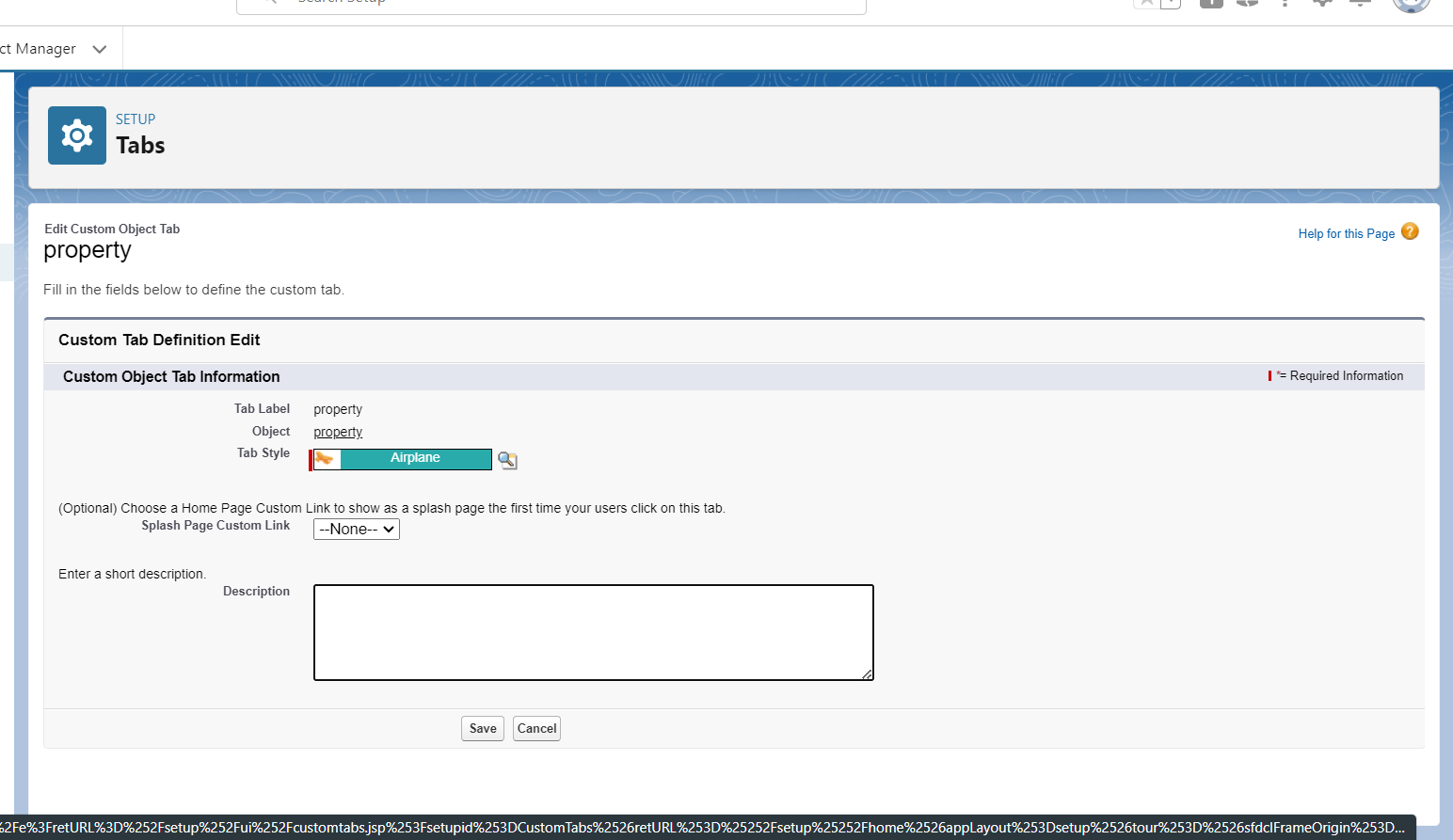


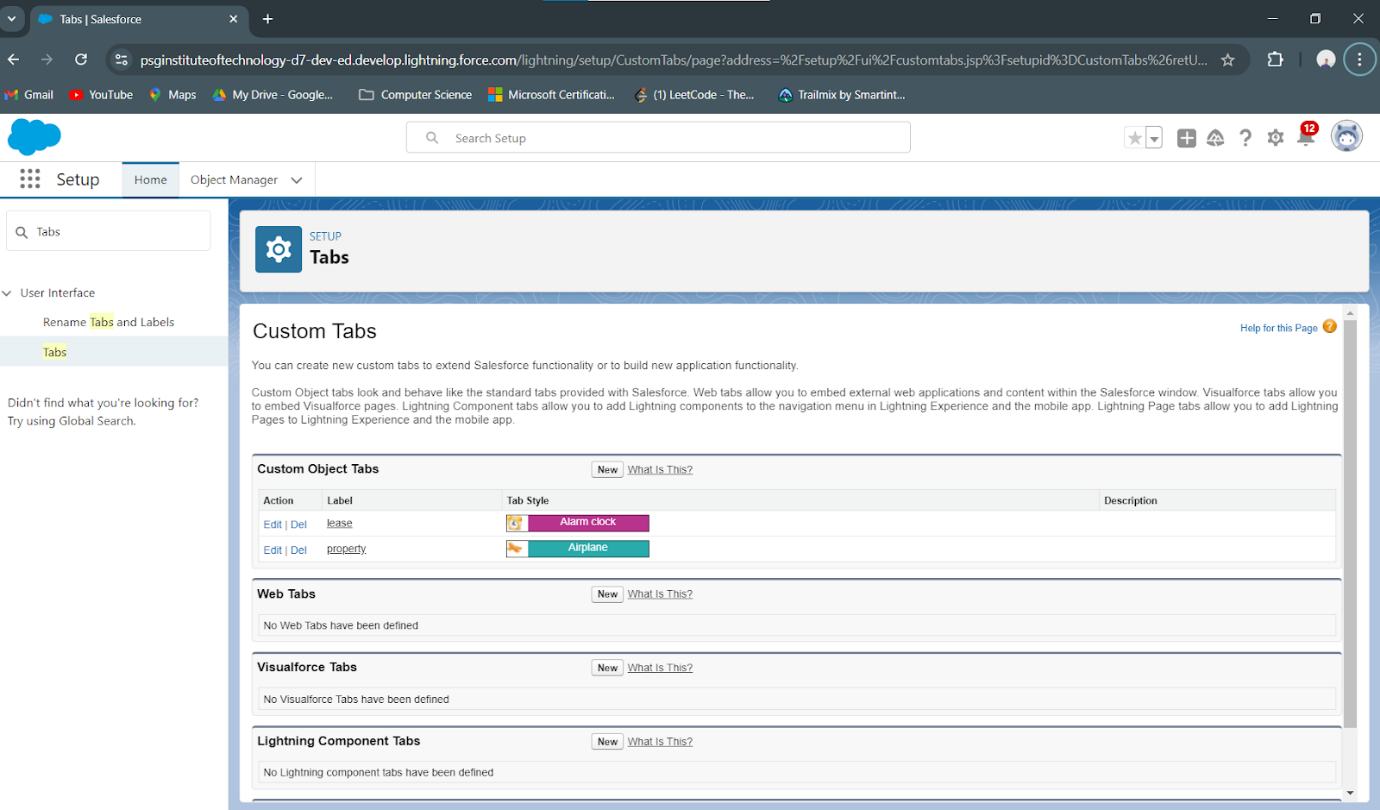




3. **Tab Configuration**

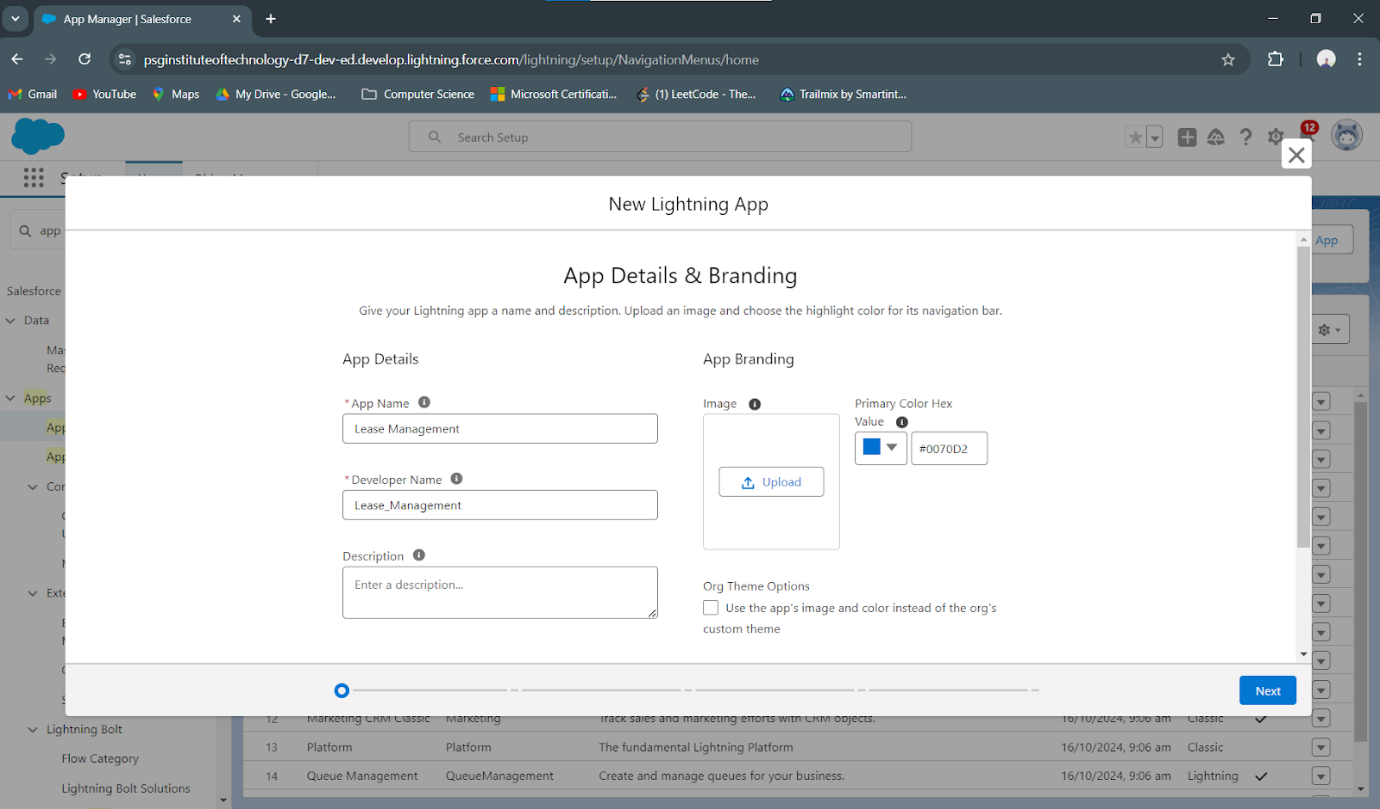
* Configured tabs for each main object to facilitate quick access.
* Enabled tabs for Property, Tenant, Lease, and Payment, allowing users to manage lease data seamlessly.

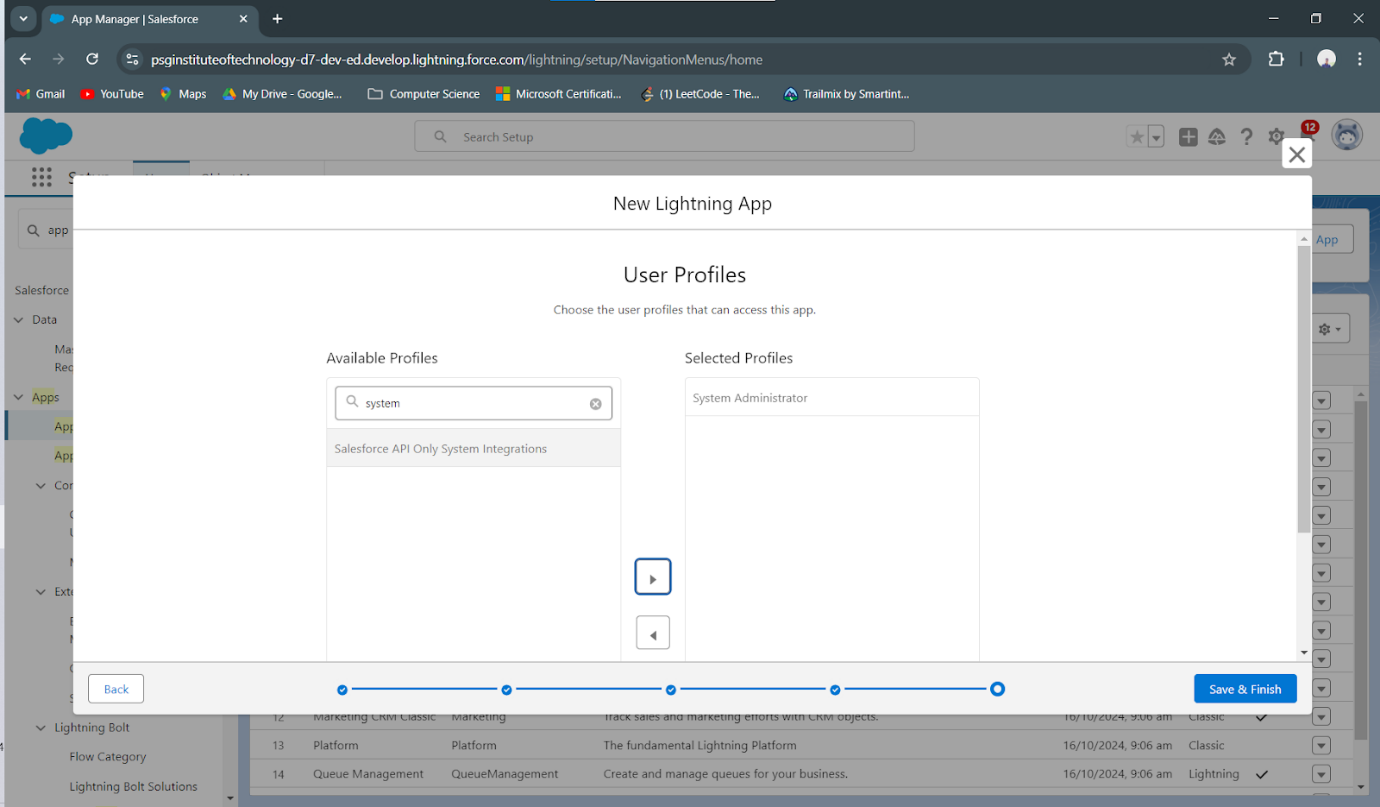




4. **Lightning App Development**

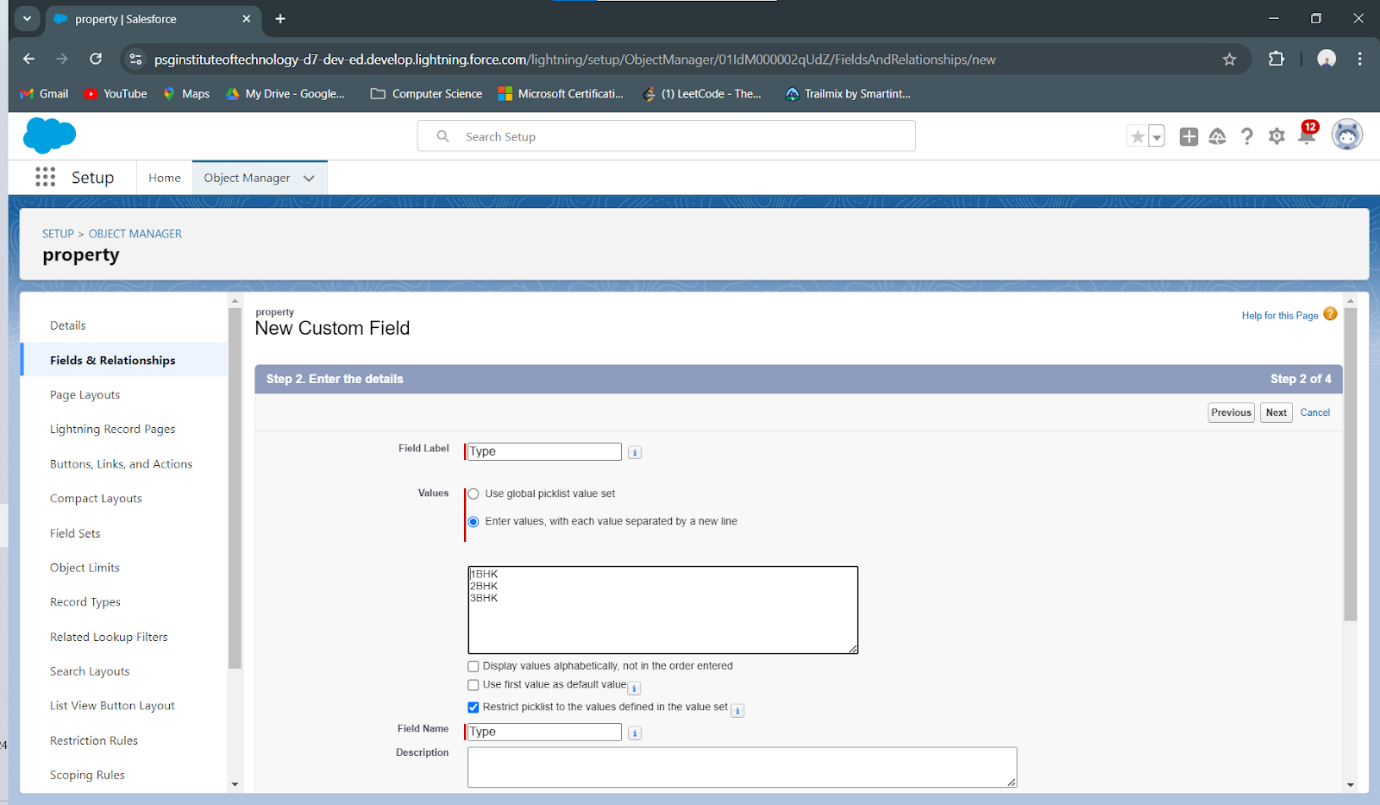
* Created a custom Lightning App named **Lease Management** to consolidate all functionalities.
* Configured branding, navigation items, and access settings to improve user experience and accessibility.



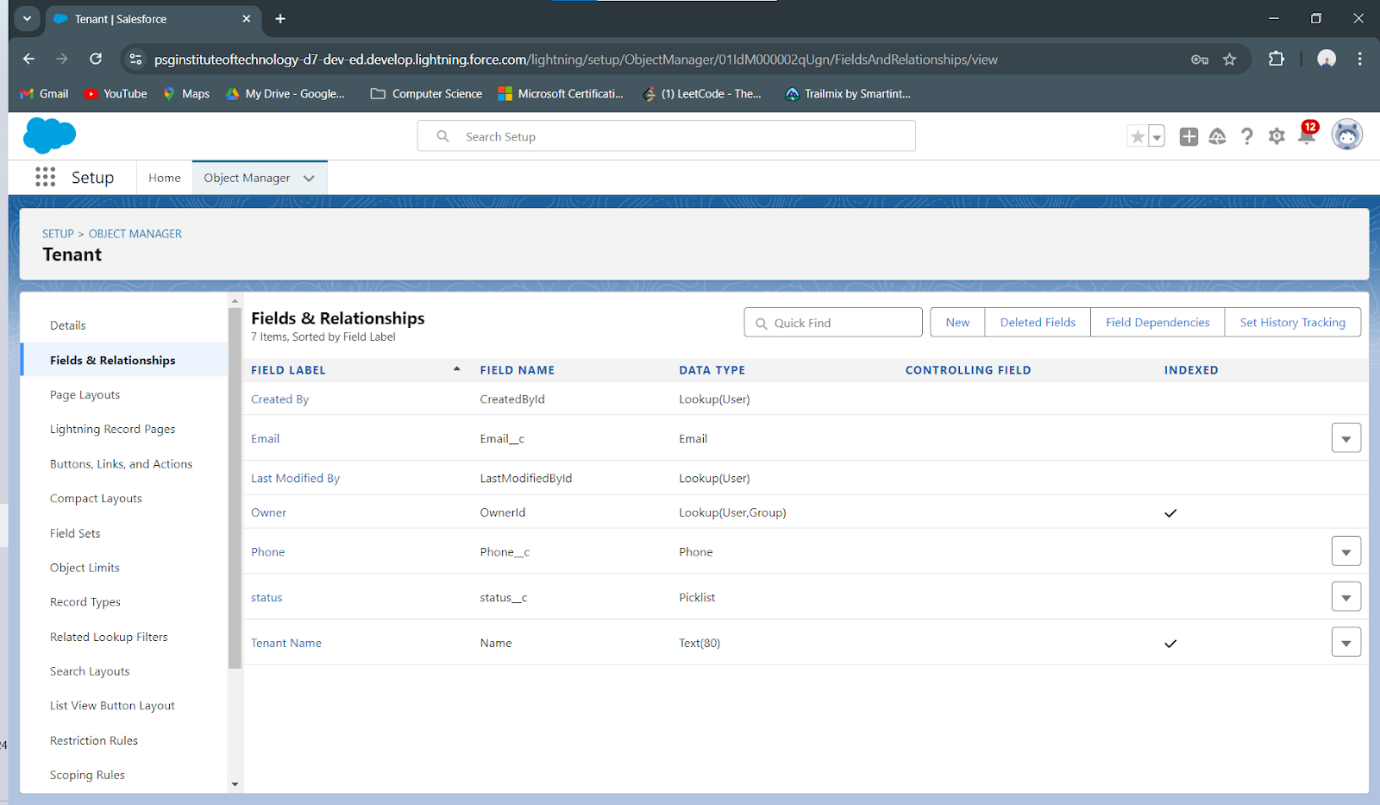


5. **Custom Field Creation**

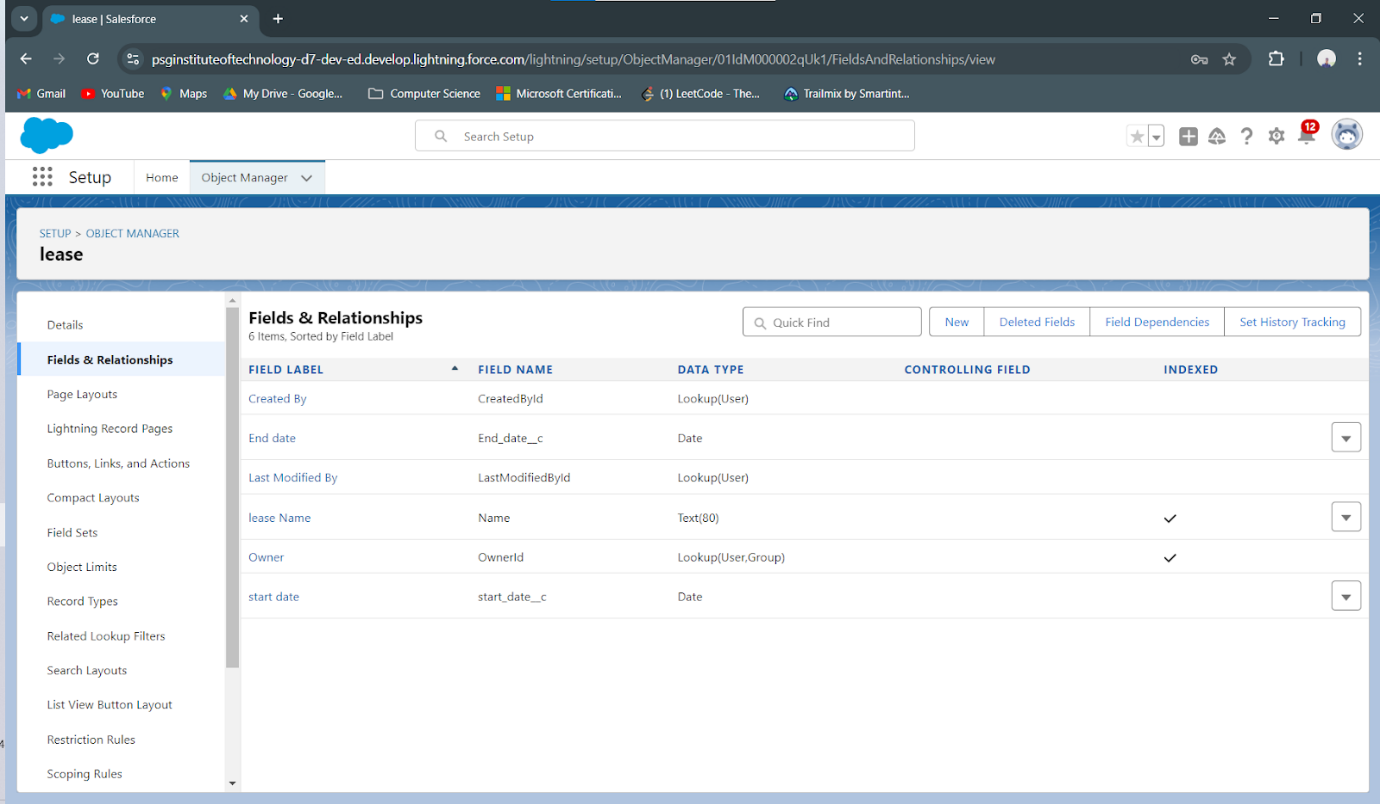
* Created custom fields within each object to capture required details:
  + **Property:** Location, type, rental amount, availability status.



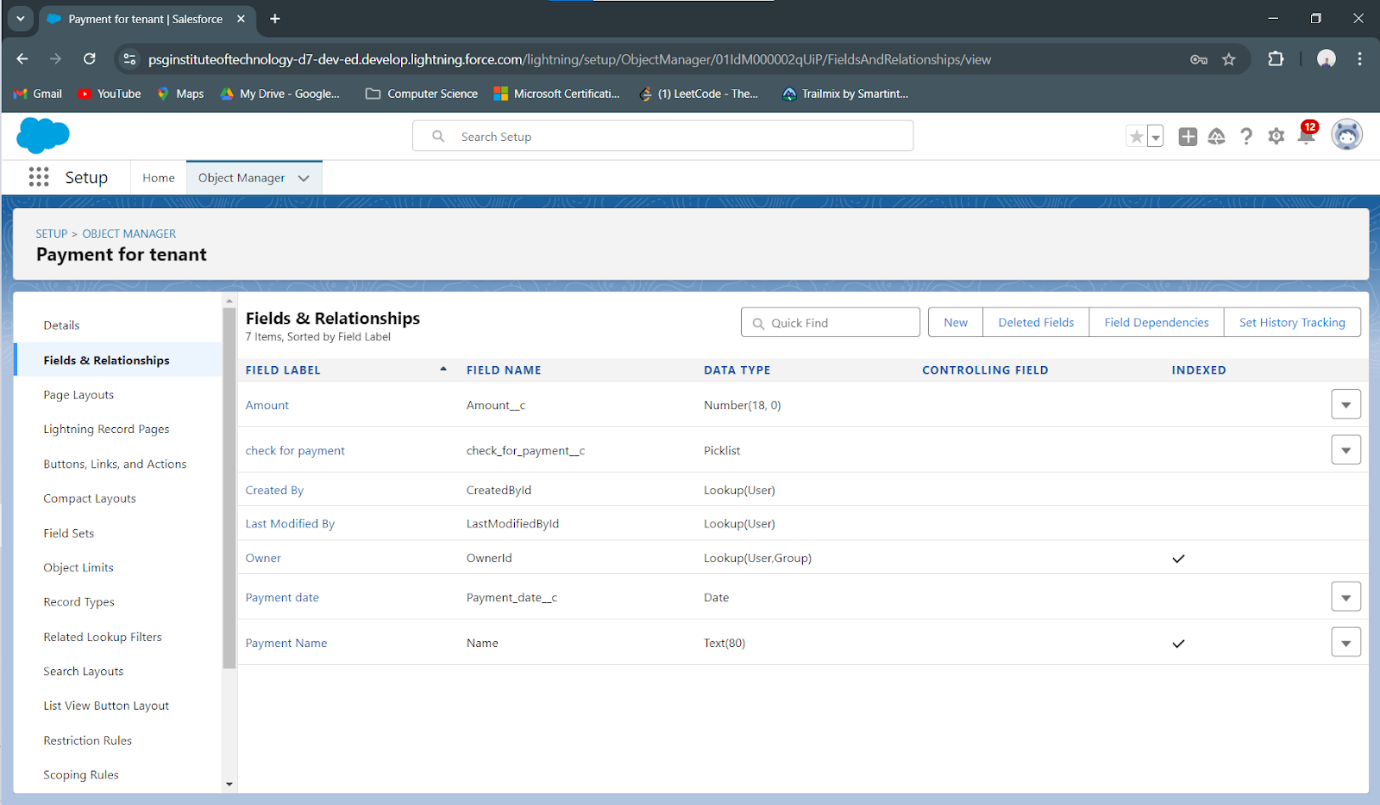
* + **Tenant:** Contact information, lease start/end dates.



* + **Lease:** Lease terms, conditions, and expiration date.



* + **Payment:** Amount due, due date, and payment status.



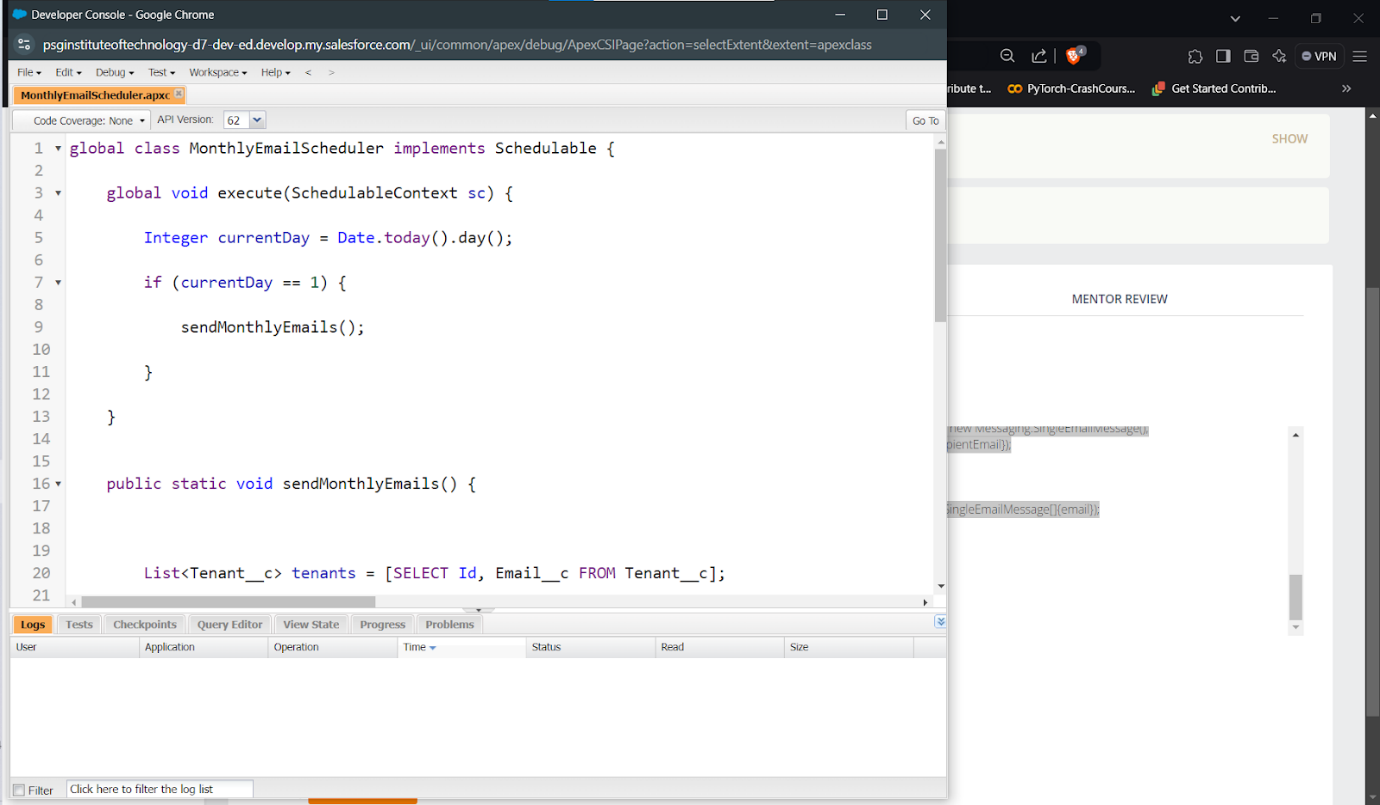
6. **Flow Creation for Monthly Payments**

* Developed a **Monthly Payment Flow** to guide users through the payment process for tenants.
* Included automated reminders and status tracking, streamlining monthly rent collection.



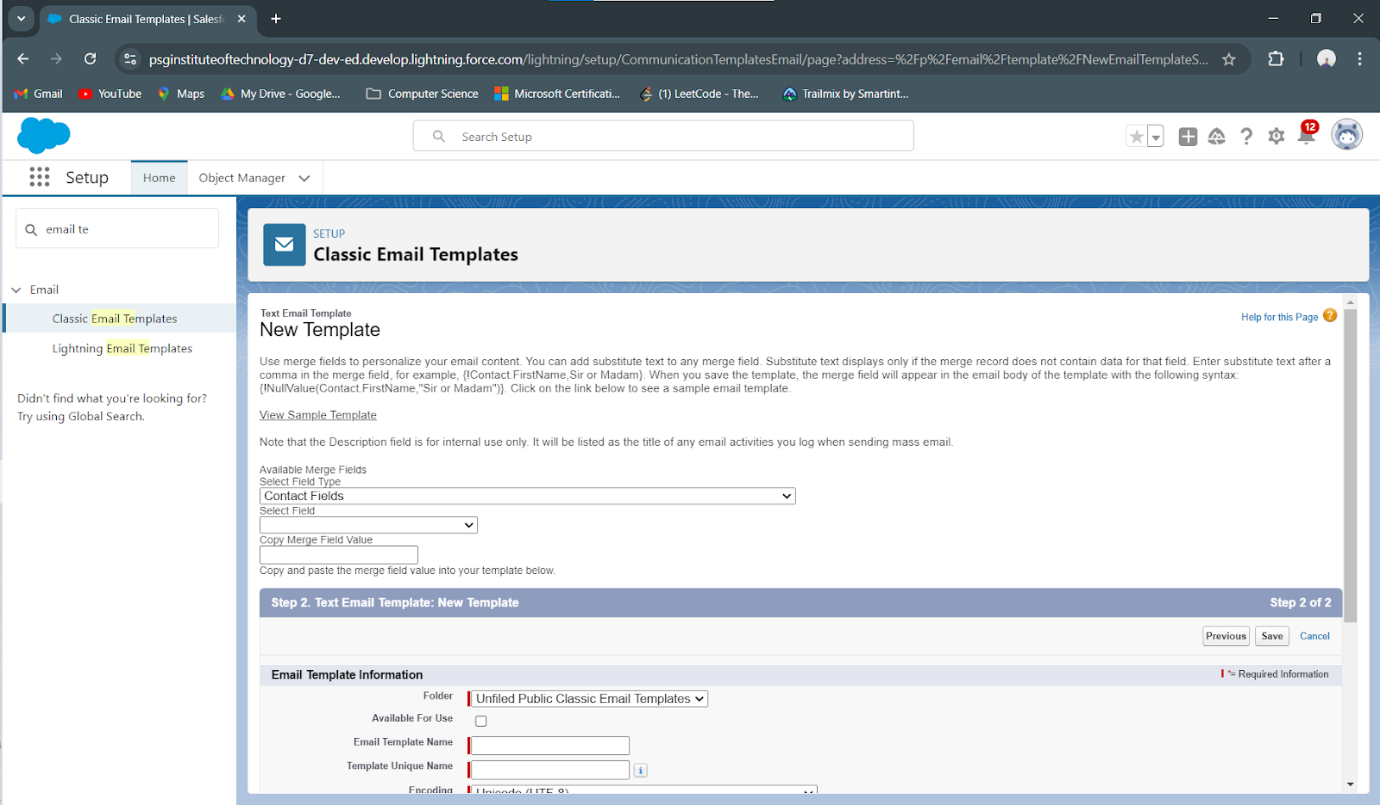
7. **Apex Trigger and Handler Class**

* Created an Apex Trigger to automate lease calculations, such as due dates and reminders.
* Developed an Apex handler class to process complex calculations and manage lease status changes.

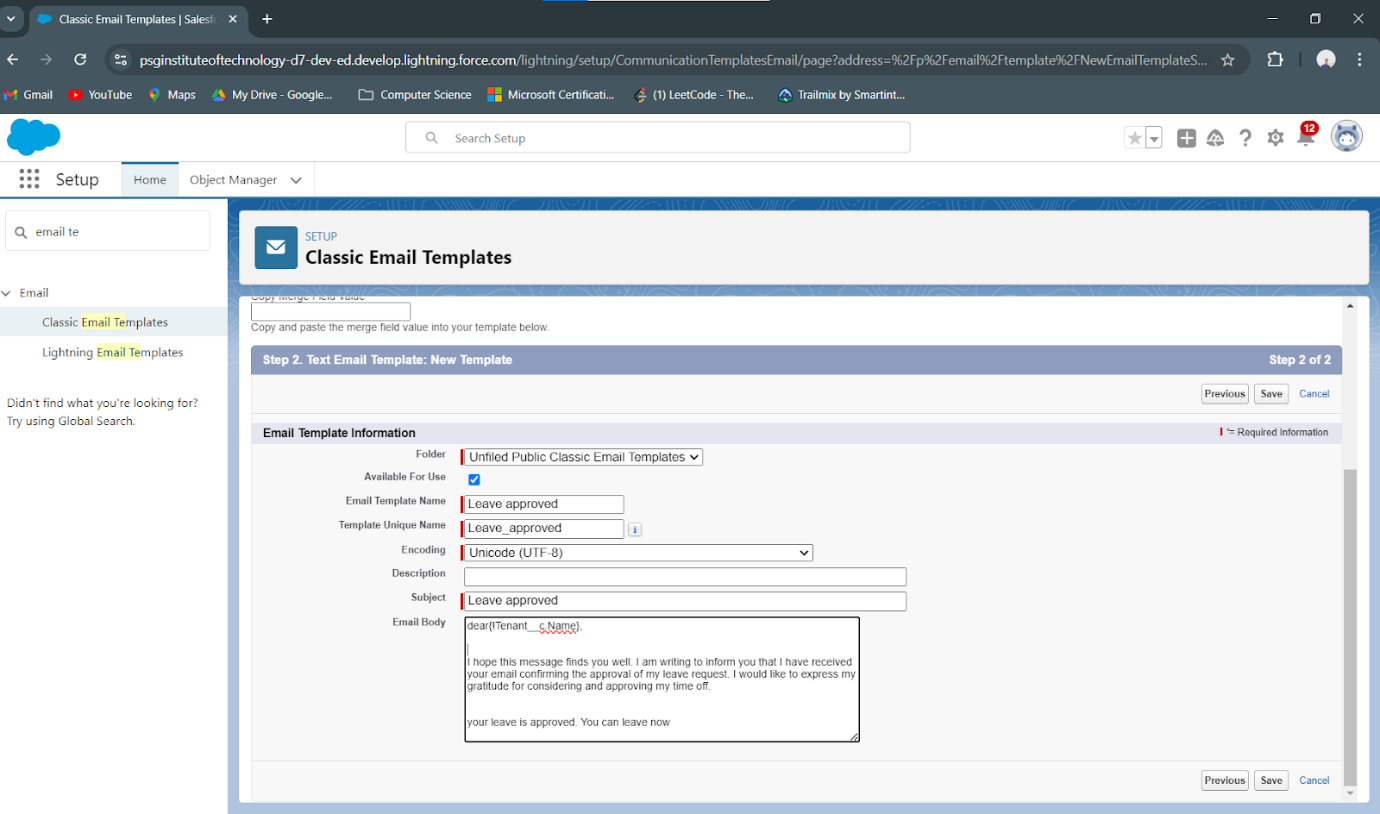


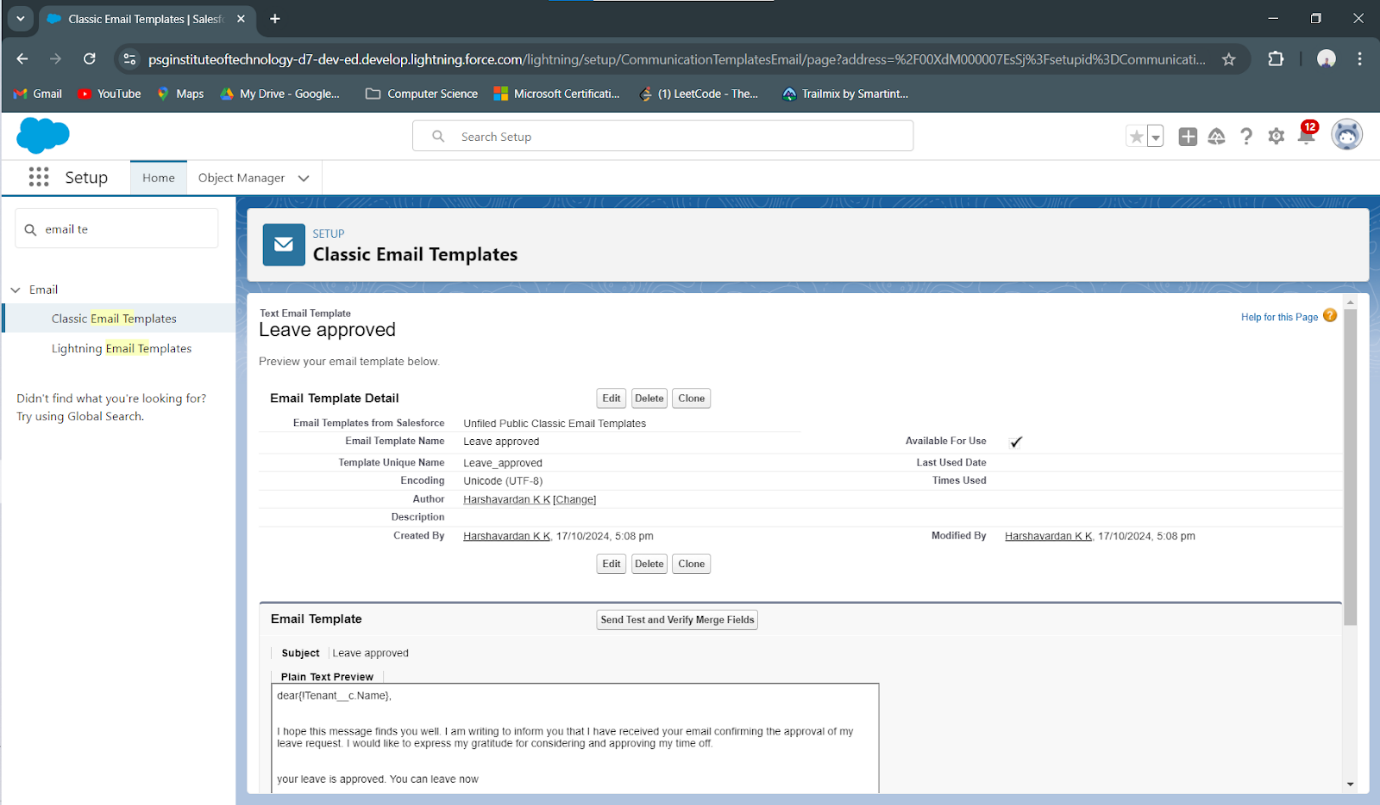
8. **Email Template Setup**

* Designed email templates for tenant communication, including notifications for payment reminders, successful payments, and lease approval or rejection.



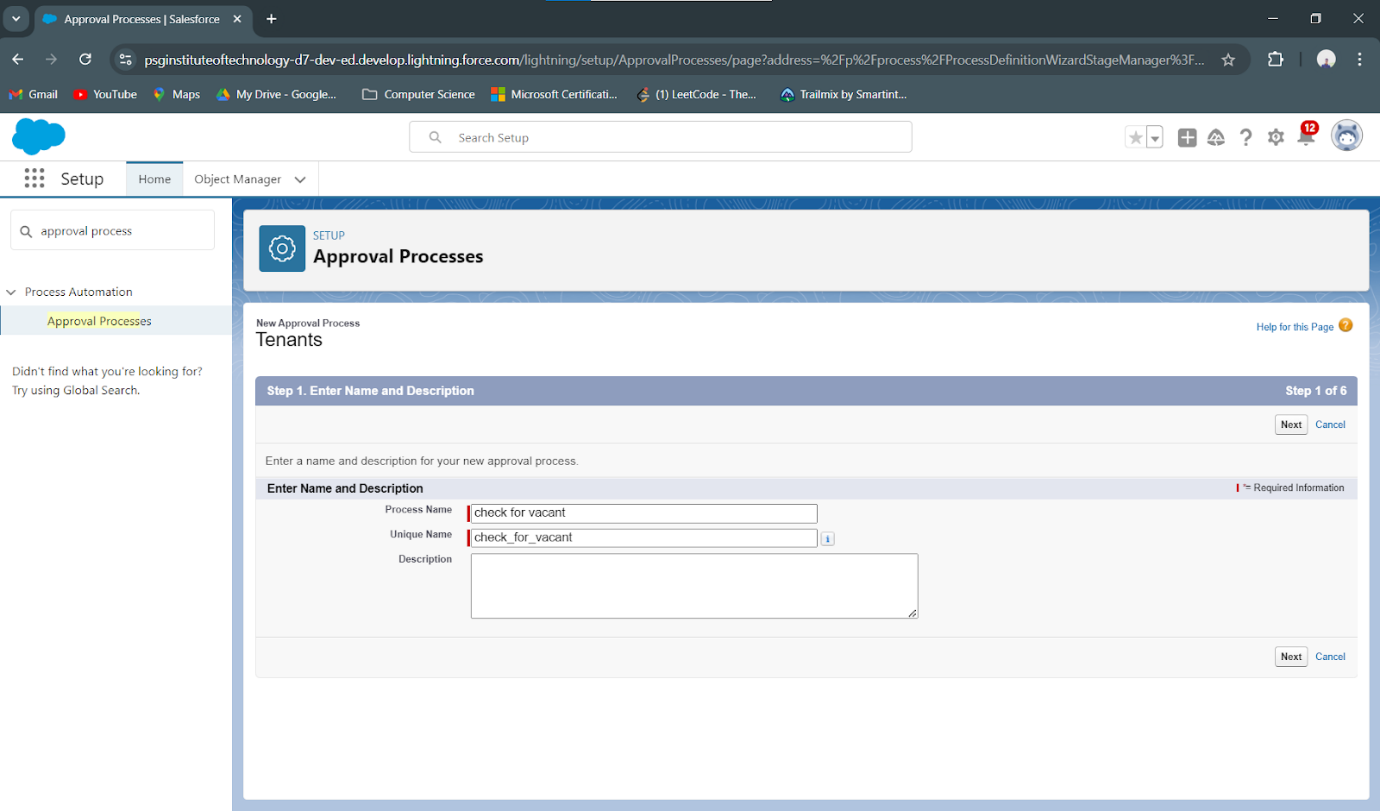






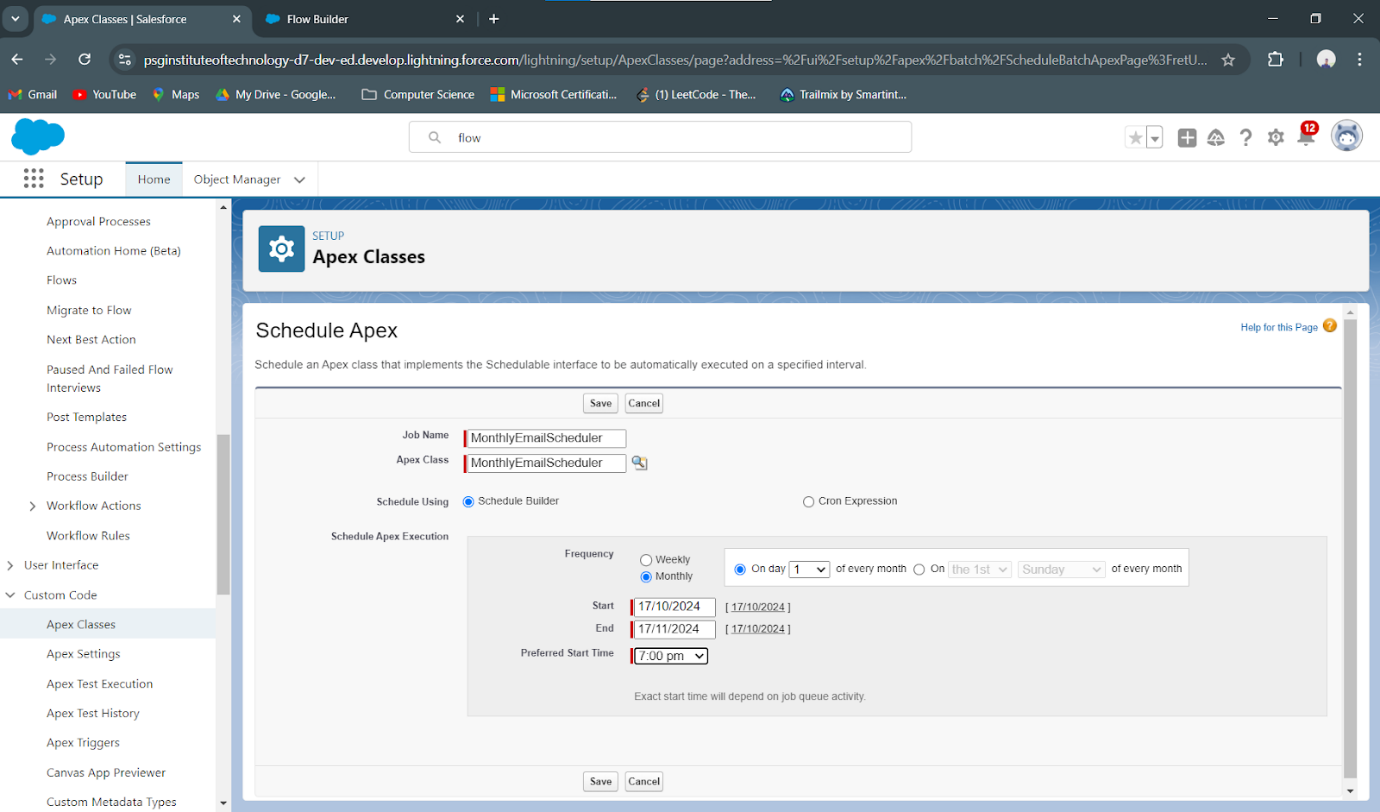
9. **Approval Process for Vacant Properties**

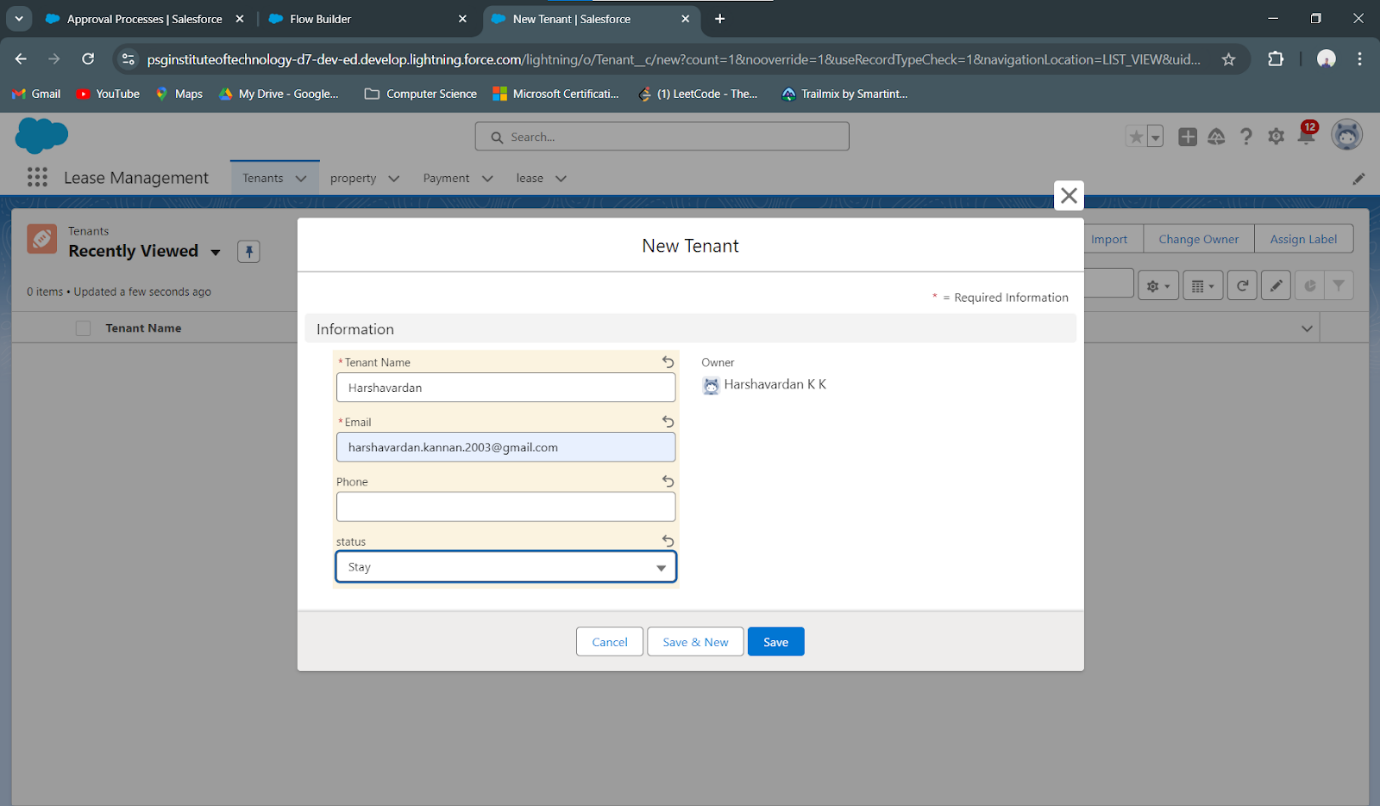
* Established an approval process to verify property availability before leasing:
  + Configured initial submission, final approval, and rejection actions to streamline the lease request process.



10. **Scheduled Apex Class for Monthly Reminders**

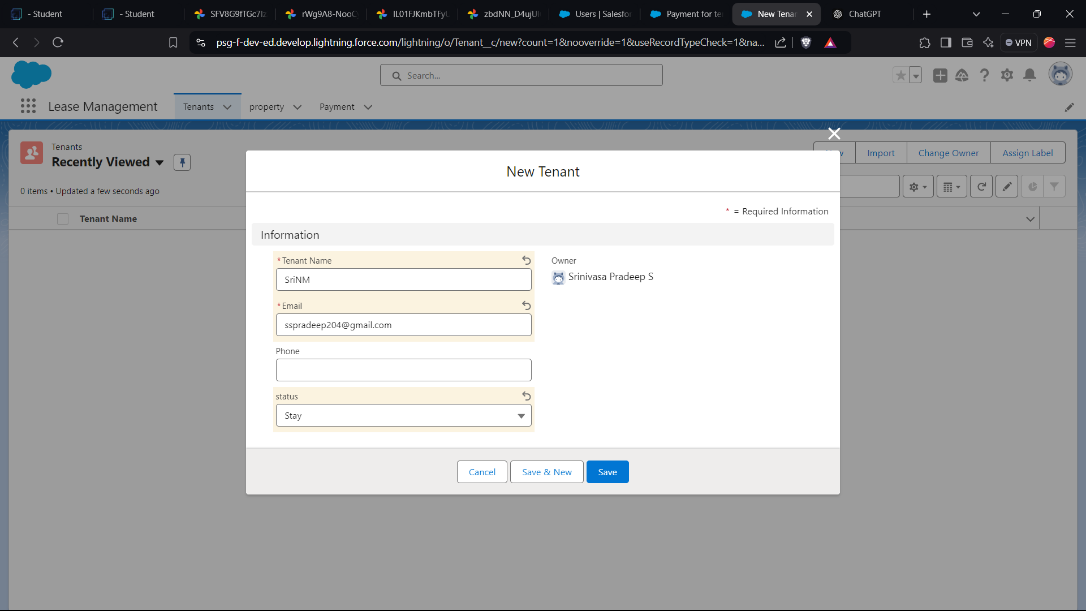
* Scheduled an Apex class to handle monthly billing reminders and lease renewals, ensuring tenants and administrators are up-to-date on upcoming obligations.





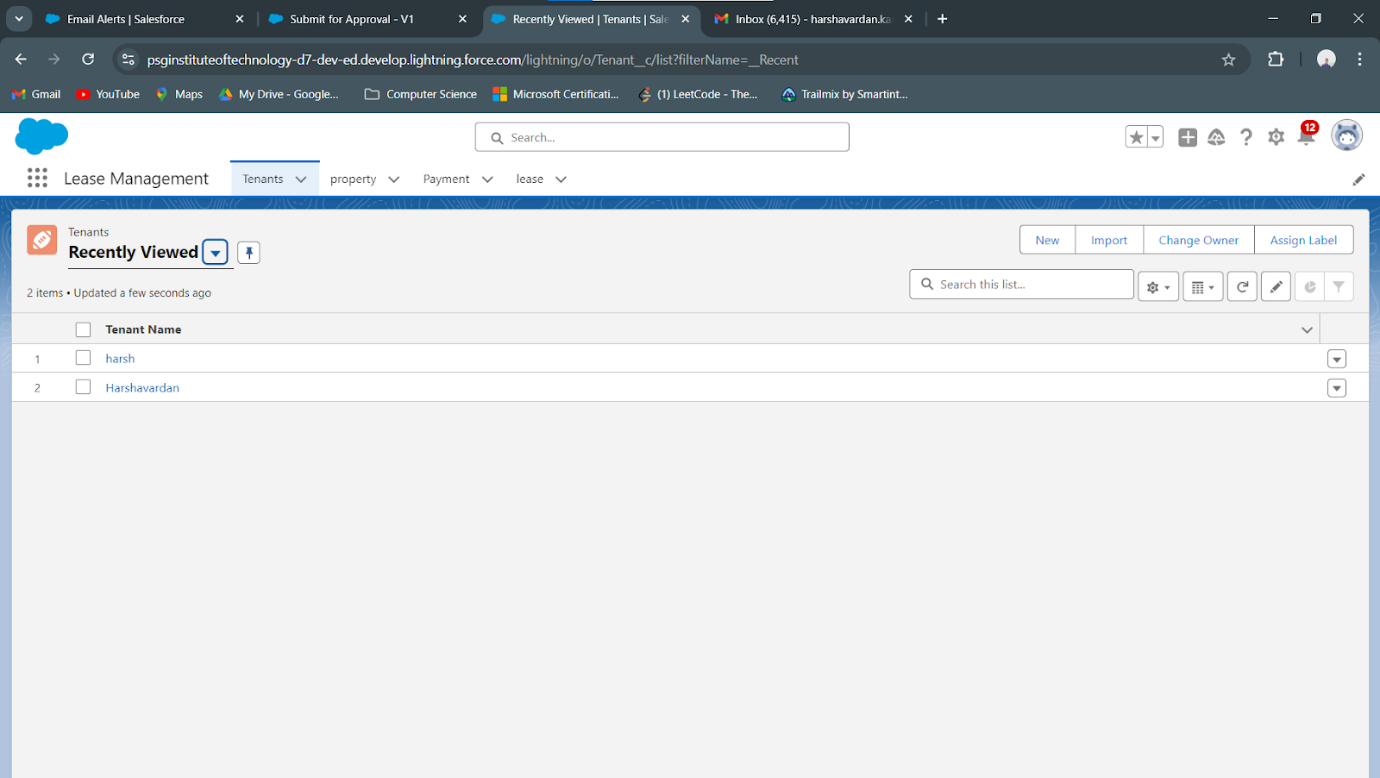
11. **Create a Tenant Record**

* Created a **Tenant** record within the system, entering all relevant details such as contact information, lease start and end dates, and tenant status.
* Ensured that the new tenant record is associated with the appropriate **Property** and **Lease** records through lookup relationships, enabling seamless data access and association.

****

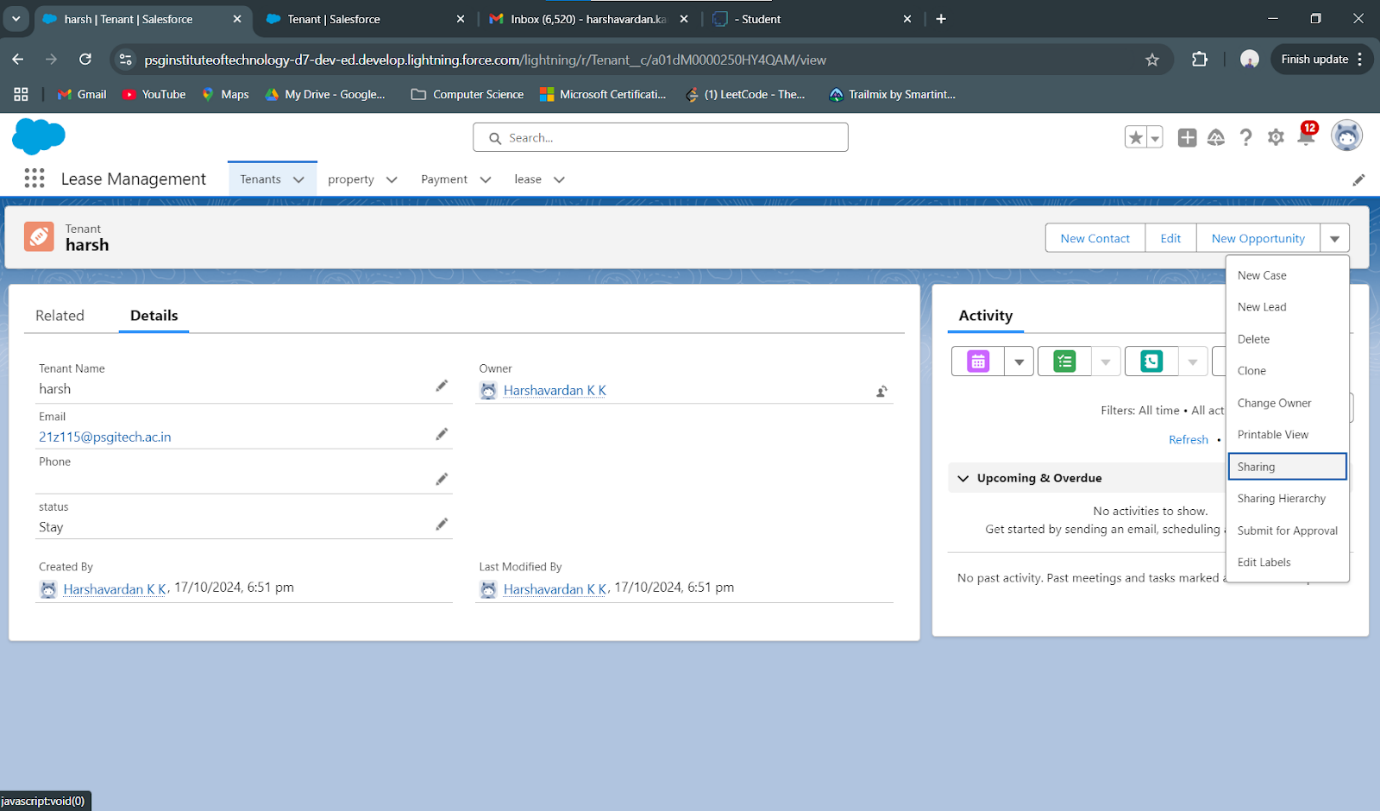
12. **Submit for Approval**

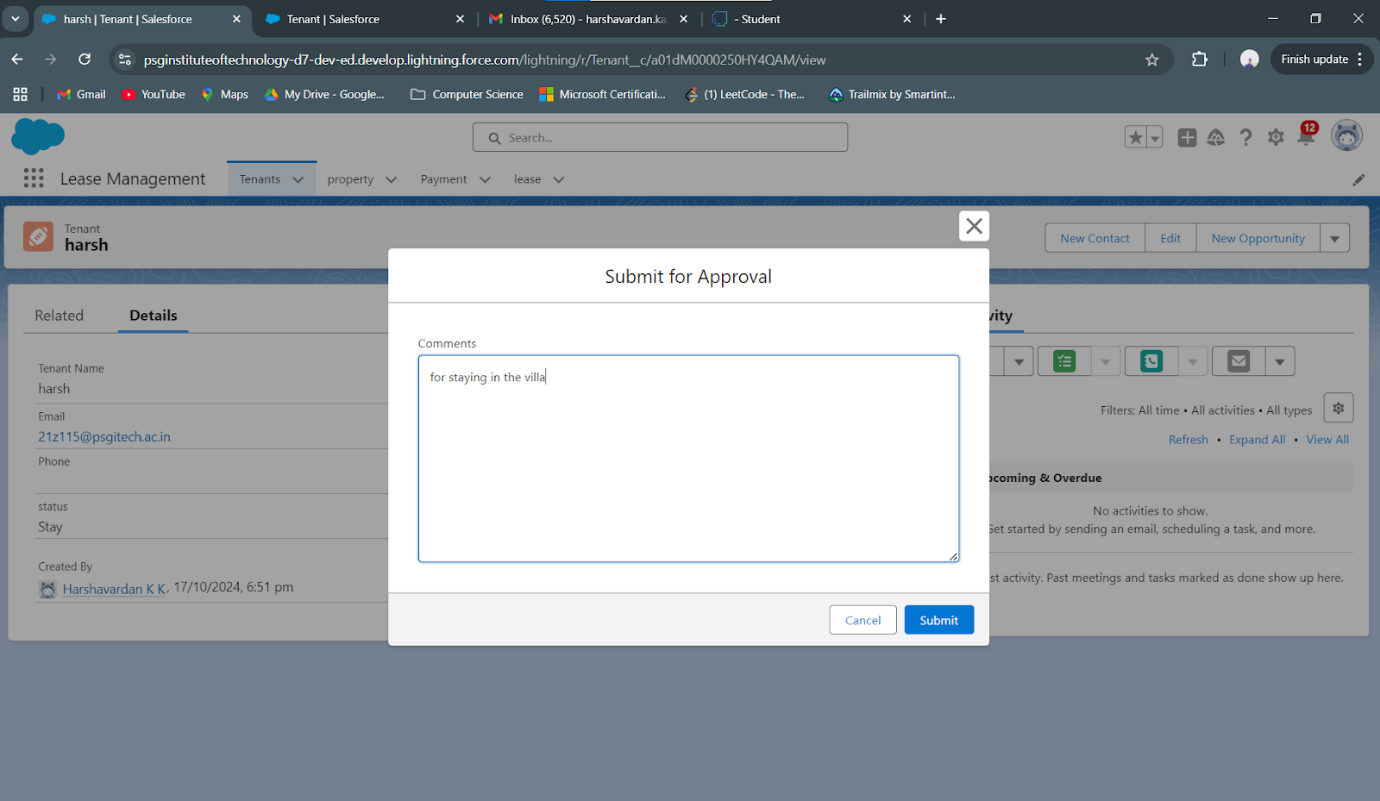
* Initiated the **Approval Process** for the new lease by submitting the **Lease** record for approval.
* The submission triggers an initial review, allowing the leasing team to validate all provided information before final approval.



1. **Wait for Approval**

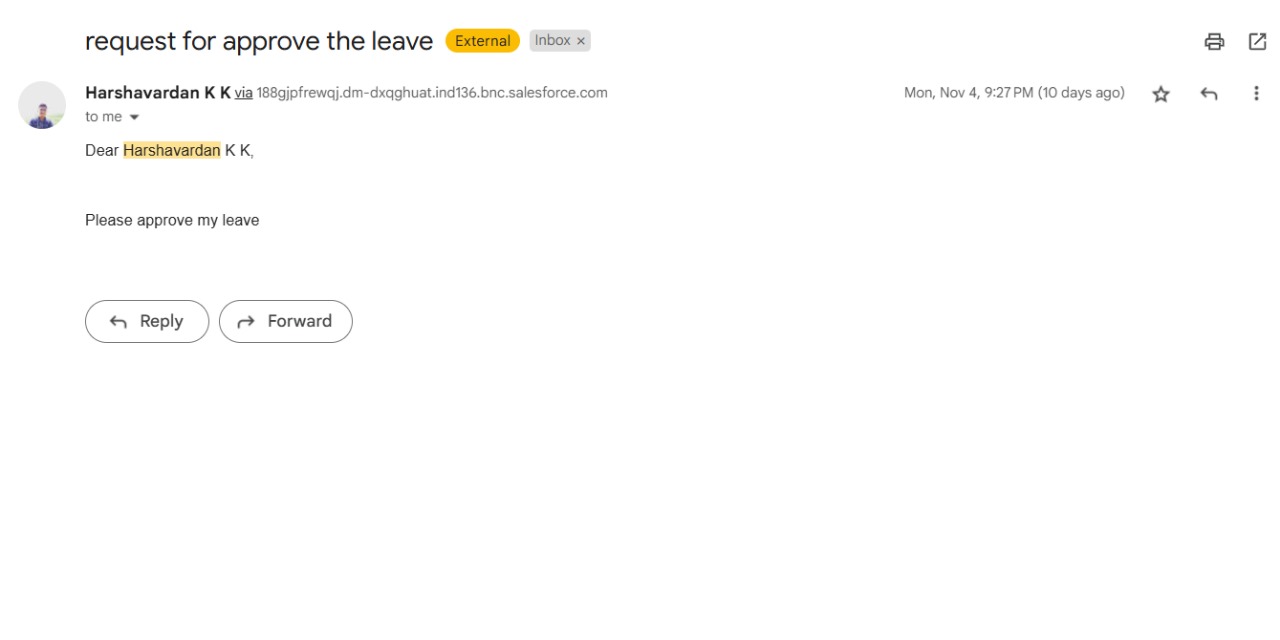
* The **Approval Process** waits for the approver’s action (approval or rejection). During this phase, the lease status remains in a “Pending Approval” state until final action is taken.



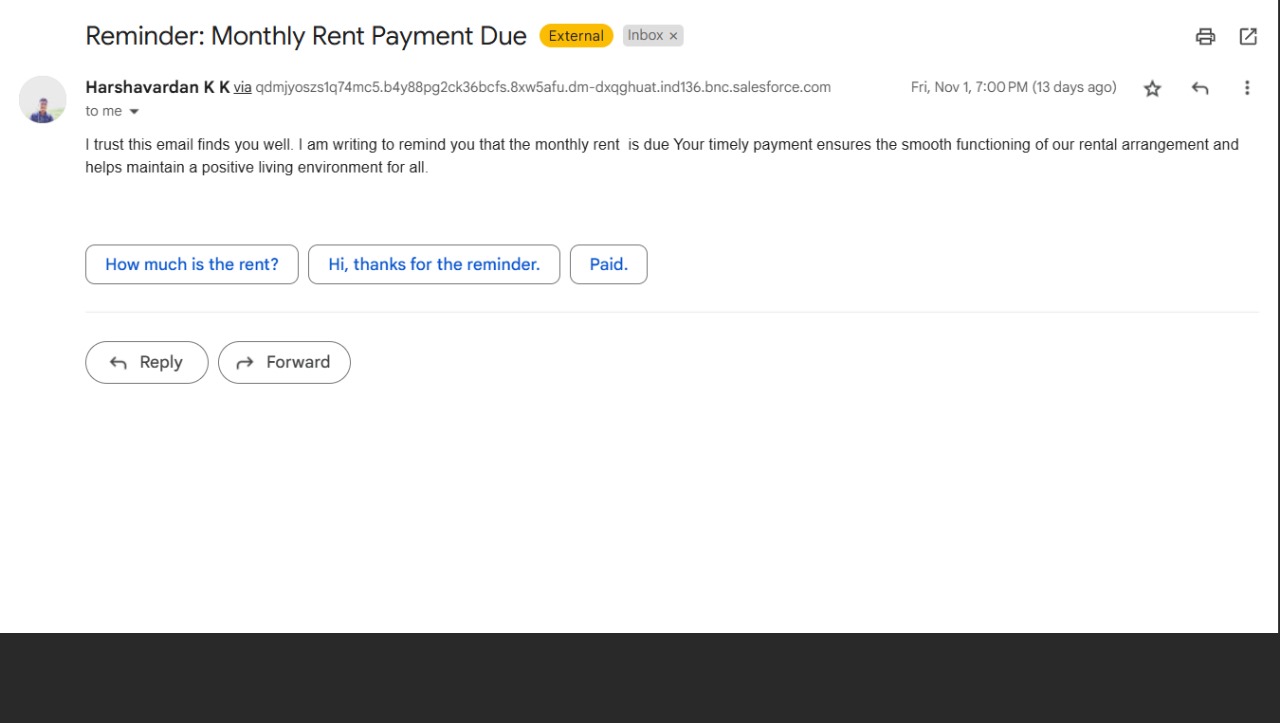


1. **Email Sent for Approval Confirmation**

* Upon approval, an automated **Email Template** for lease approval is triggered and sent to the tenant, confirming the lease’s approval status.



* The email outlines the lease’s terms and conditions, ensuring clear communication with the tenant.



1. **Testing and Validation**

**Unit Testing (Apex Classes, Triggers)**

**Apex Trigger:**

trigger test on Tenant\_\_c (before insert)

{

if(trigger.isInsert && trigger.isBefore){

testHandler.preventInsert(trigger.new);

}

}

**Test Class:**

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');

}

}

}

}

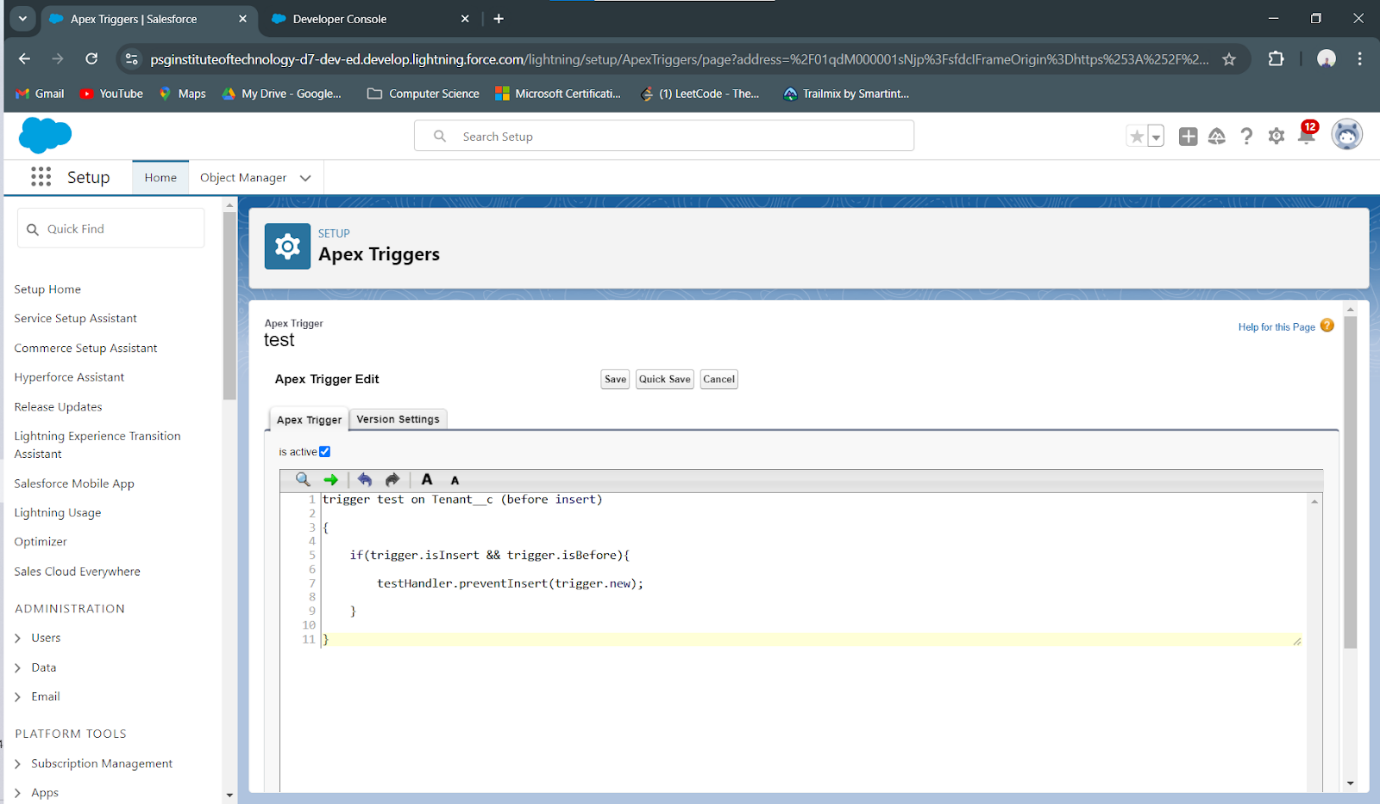
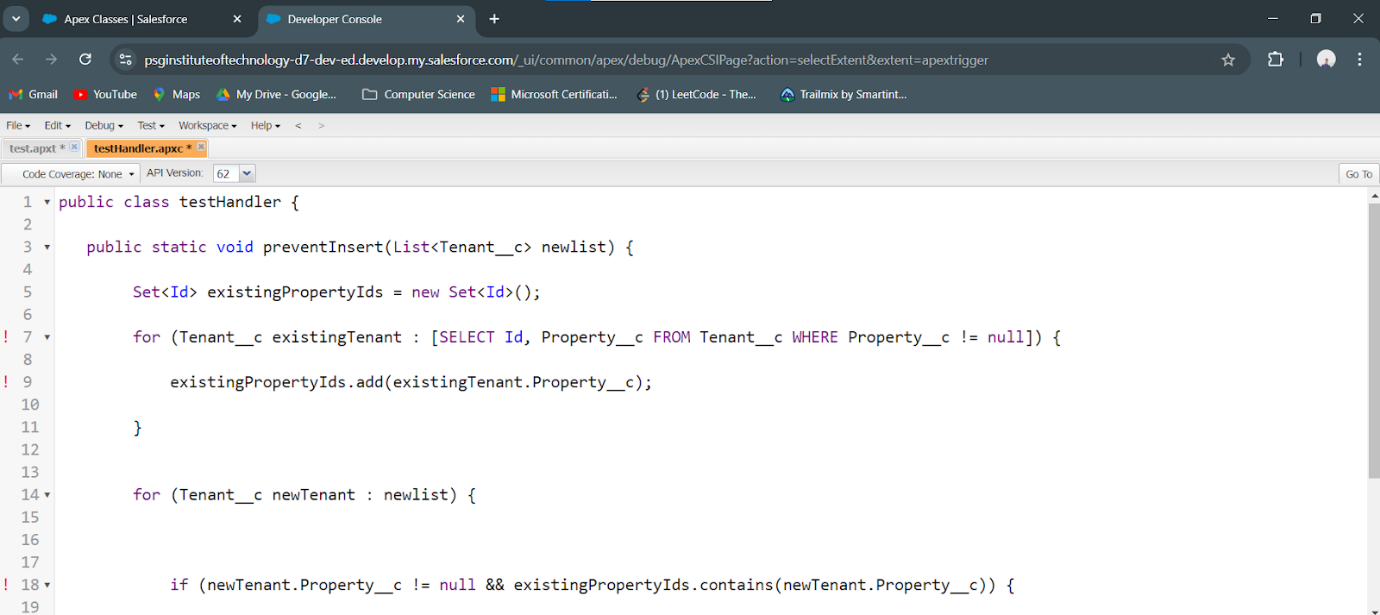
**MonthlyEmailScheduler:**

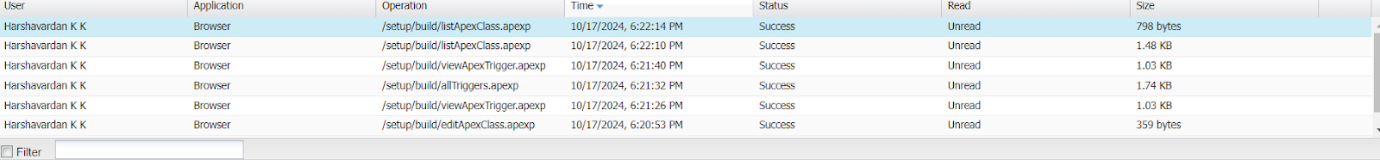
global class MonthlyEmailScheduler implements Schedulable {  
  
    global void execute(SchedulableContext sc) {  
  
        Integer currentDay = Date.today().day();  
  
        if (currentDay == 1) {  
  
            sendMonthlyEmails();  
  
        }  
  
    }  
  
  
    public static void sendMonthlyEmails()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

**Steps**:

* 1. Created a Test Property Record: Created a Property\_\_c record to serve as the property for assigning tenants.
  2. Inserted the First Tenant: Inserted a Tenant\_\_c record linked to the test property, simulating the initial tenant assignment.
  3. Prepared a Second Tenant for the Same Property: Created another Tenant\_\_c record with the same property ID to test the trigger’s restriction on multiple tenants per property.
  4. Inserted the Second Tenant: Attempted to insert the second tenant within a try-catch block to catch the expected error, confirming that the trigger prevented the insertion.
  5. Verified Error Message: Checked if the error message matched the expected output: "A tenant can have only one property," verifying that the trigger works as intended.
  6. Run Apex Test Execution:
* Go to Setup > Apex Test Execution.
* Select TenantTriggerTest and click Run.

Confirm that all assertions pass, indicating that the trigger prevents multiple tenants from being assigned to the same property.



1. **Conclusion**

**Summary of Achievements:** The Lease Management project successfully developed a streamlined, transparent, and scalable solution for managing property leases. Key achievements include:

* **Organized Data Management:** Custom objects, fields, and tabs were created for structured and efficient lease tracking.
* **Process Automation:** Triggers, validation rules, and scheduled classes were implemented to streamline recurring tasks, including monthly payments and notifications.
* **Enhanced Communication:** Approval workflows and email templates were designed to automate tenant communications, improving clarity and tenant satisfaction.
* **Data-Driven Decision-Making:** Custom reports and dashboards provide stakeholders with real-time insights, enabling faster and more informed decision-making.

This project showcases the powerful application of Salesforce in lease management, driving operational efficiency, ensuring data accuracy, and enhancing tenant relations through optimized workflows.