ANNAMACHARYA INSTITUTE OF TECHNOLOGY AND SCIENCES, KADAPA

Project Report on

SALESFORCE - Lease Management System

BY – N. Mahesh Babu (22HM1A3324)

Email: mb349930@gmail.com

Table of Contents

S.No. Title

1. PROJECT OVERVIEW

The **Lease Management System** is a comprehensive digital solution designed to simplify and automate the management of lease agreements for property managers and tenants. Developed using Salesforce, the system centralizes all lease-related data and activities—ranging from lease creation and renewal to payment tracking and compliance monitoring—into one cohesive platform.

By replacing manual, paper-based workflows with a structured and automated environment, the system significantly reduces administrative workload, eliminates errors, and ensures timely actions through automated alerts and real-time updates.

This platform is especially beneficial for organizations managing multiple properties, as it offers seamless data access, role-based permissions, and intelligent reporting tools that support data-driven decision-making.

The system covers the entire lease lifecycle from creation to renewal or termination helping teams focus on key tasks instead of paperwork.

1.1 Technical Approach

The Lease Management System is designed to be simple, fast, and secure. It uses a **central database** to store all lease-related data, ensuring quick access and better data management.

To reduce manual work, the system uses **automation** for tasks like payment reminders and compliance checks. This helps avoid delays and improves accuracy.

Real-time updates ensure that any change made like payment status or lease details— is instantly visible to all users, keeping information consistent.

The platform is built to be **flexible and scalable**, allowing future upgrades like adding new reports or integrating with other tools.

The **user interface** is clean and easy to use, designed even for non-technical users to work without confusion.

For security, features like **login authentication, user roles, and data protection** are built in to keep sensitive information safe.

In summary, the system is technically strong, user-focused, and ready to adapt as business needs grow.

1.2 Benefits:

- **Centralized Data Access**
  All lease information is stored in one secure place for easy access and updates.
- **Time-Saving Automation**
  Automatic alerts and checks reduce manual work and prevent missed deadlines.
- **Real-Time Updates**
  Any change is instantly reflected for all users, ensuring accurate and current data.
- **Improved Decision-Making**
  Built-in reports provide insights into leases, helping businesses plan better.
- **User-Friendly Interface**
  Simple design makes it easy for anyone to use, even without technical skills.
- **Enhanced Security**
  Login control, user roles, and data protection keep sensitive info safe.
- **Scalability and Flexibility**
  The system can grow with the business and easily integrate with other tools.

2. OBJECTIVE

This section highlights the clear and measurable goals that the Lease Management System aims to achieve. These objectives focus on solving key business challenges, streamlining operations, and delivering real value to both property managers and tenants. **2.1 Business Goals**

- **Streamline Lease Management:**
  Develop a user-friendly platform that simplifies the management of lease agreements, tenant profiles, and property data, minimizing manual effort and bringing structure to the entire process.
- **Enhance Data Security and Controlled Access:**
  Securely store tenant and property information, ensuring that only authorized individuals can view or modify sensitive data.
- **Automate Lease Renewal and Termination Notifications:**
  Automatically send timely alerts for upcoming lease renewals or terminations to keep property managers and tenants informed and prepared.
- **Efficient Payment Tracking:**
  Monitor rent payments and due dates effectively, giving property managers a clear view of payment history and overall financial performance.
- **Generate Insightful Reports and Analytics:**

  Provide real-time reports on lease status, payment trends, and property performance to support better decision-making and long-term planning.

2.2 Specific Outcomes

- **Intuitive User Interface:**
  Provide a clean, easy-to-navigate interface that allows both tenants and managers to manage and access lease data without confusion.
- **Automatic Reminders:**
  Implement alert systems to notify users of important events like rent due dates, lease renewals, and contract expirations.

- **Live Data Synchronization:**

  Ensure that all updates to lease records are reflected instantly across the platform, giving users access to the most accurate and up-to-date information.

- **Report Generation:**

  Enable property managers to create reports summarizing lease activities, payment statuses, and property performance for better analysis and decision-making.

- **Online Rent Payment:**

  Integrate online payment options to allow tenants to pay rent digitally, making the process more convenient and reducing delays.

- **Expandable System Design:**

  Build the system with a scalable database architecture that can support additional tenants and properties as the business grows.

3. REQUIREMENT ANALYSIS & PLANNING

3.1 Understanding Business Requirements

The first phase of the project focused on understanding the real-world challenges faced by property managers and tenants in managing lease agreements. Through stakeholder meetings, user interviews, and domain research, key pain points were identified to shape the system's core functionalities.

Identified Problems & User Needs:

- Manual handling of lease documents often leads to data entry errors, delays, and lack of consistency.
- Property managers face difficulties in tracking rent payments, lease expirations, and due dates across multiple tenants.
- Lease data is scattered and unstructured, making it hard to access and manage centrally. • No alert mechanism for upcoming lease renewals or terminations causes missed actions and delays.
- Current systems lack data security, putting sensitive tenant and financial data at risk.

Solution Approach:

The Lease Management System addresses these problems by offering an automated, centralized, and secure digital platform. The system is designed to streamline daily operations, enhance visibility, and improve accuracy in lease tracking and management.

3.2 Defining Project Scope and Objectives

This step involves outlining what Phase 1 of the Lease Management System will deliver and how it will align with the overall business goals. The focus is on building a core functional system that can later be expanded.

Phase 1 Scope Includes:

- Centralized platform for managing leases, tenant records, properties, and payments
- User access through secure login based on roles (admin, manager, tenant)
- Lease lifecycle automation: creation, renewal reminders, and termination alerts
- Dashboard displaying lease status, payment due dates, and tenant activity
- Real-time synchronization of updates to ensure consistent and current data
- Flexibility to scale across multiple properties, locations, and user roles

- Data security measures like authentication, access control, and audit trails
- Built-in reporting for financial summaries and lease performance **Objectives:**
- Improve operational efficiency by digitizing manual lease management
- Provide transparency and real-time visibility into lease data
- Enhance communication between tenants and property managers
- Support better decision-making through reporting and analytics
- Ensure system stability and ease of use for non-technical users
- Maintain long-term sustainability with modular and scalable design

The objective is to deliver a reliable MVP (Minimum Viable Product) that solves immediate pain points and forms the foundation for future enhancements.

3.3 Design: Data Model and Security Model

A. Data Model Design

The data model is the backbone of the Lease Management System. It defines how data is structured, stored, and linked across the application. The model ensures efficient data retrieval and clear relationships between different business entities.

Core Entities:

- **Users:** Stores login credentials and role information (Admin, Property Manager, Tenant)
- **Tenants:** Contains personal information, lease history, contact details, and tenant-specific notes
- **Properties:** Stores property details like ID, name, address, and ownership information
- **Leases:** Connects tenants to properties and includes start/end dates, rent amount, and lease status
- **Payments:** Tracks payment records including amount, due date, payment status, method, and remarks **Relationships:**
- One Property → Many Leases
- One Tenant → Many Leases (historical)
- One Lease → Many Payments

These relationships ensure a normalized, scalable data structure that supports reporting and performance optimization.

B. Security Model Design

Given the sensitivity of lease and financial data, the system incorporates strong security practices to safeguard information and ensure data privacy.

Security Features:

- **User Authentication:**
  A secure login system requiring valid username and password combinations, with session control to prevent unauthorized access.
- **Role-Based Access Control (RBAC):**
  Users are assigned roles which define what data they can view or modify.
  - Tenants can view their own leases and payments.
  - Property Managers can manage leases, payments, and tenant profiles.
  - Admins have full access to all modules.
- **Data Encryption:**
  Sensitive information (e.g., payment methods, user credentials) is encrypted both during transmission (SSL/TLS) and at rest in the database.
- **Audit Logging:**
  Every critical operation (e.g., lease updates, payment status changes) is recorded for traceability. This ensures accountability and helps in debugging or compliance.
- **Input Validation & Protection:**
  All user input is validated to prevent common web vulnerabilities such as:
  - SQL Injection
  - Cross-Site Scripting (XSS)
  - CSRF (Cross-Site Request Forgery)

These security layers protect the system from unauthorized access, data loss, and malicious attacks, ensuring the platform is trustworthy and compliant with best practices.

4. SALESFORCE DEVELOPMENT – BACKEND & CONFIGURATIONS

4.1 Setup Environment & DevOps Workflow

We established a robust Salesforce development environment using:

- **Developer Org Strategy**: Created Developer Org sandboxes for development and testing.

  You can create a Developer Org using this official Salesforce link:

  https://developer.salesforce.com/signup

- **Source Control**: Used GitHub to manage code versions, collaborate with the team, and track all changes efficiently.
- **Deployment Tools**: Utilized Change Sets and Salesforce DX (SFDX CLI) to deploy components between Sandbox and Production environments.
- **Testing Environment**: All features were validated in the Sandbox to ensure quality and stability before live deployment.

This setup supports faster development, safer releases, and better tracking of changes during the entire project lifecycle.

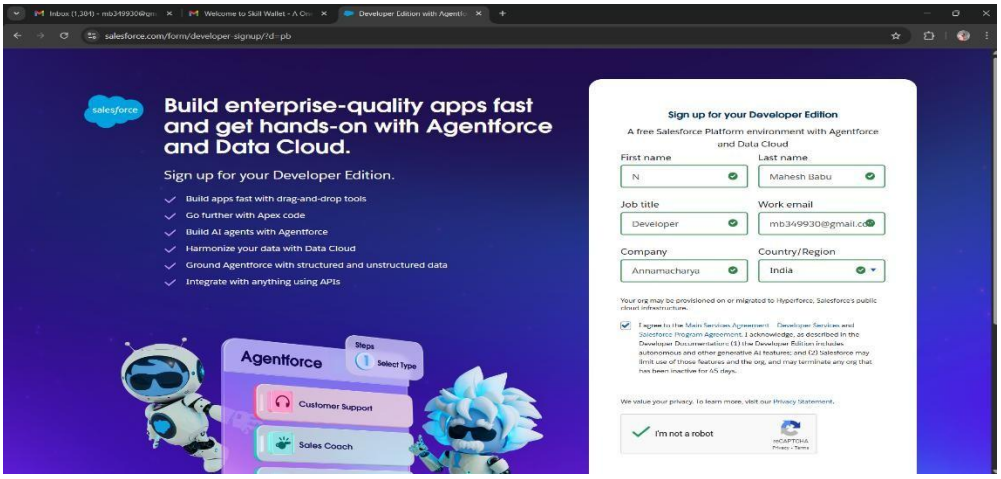


Fig: Signing up into the Developer Org

Now, enter your credentials to login into the Salesforce Developer Org.

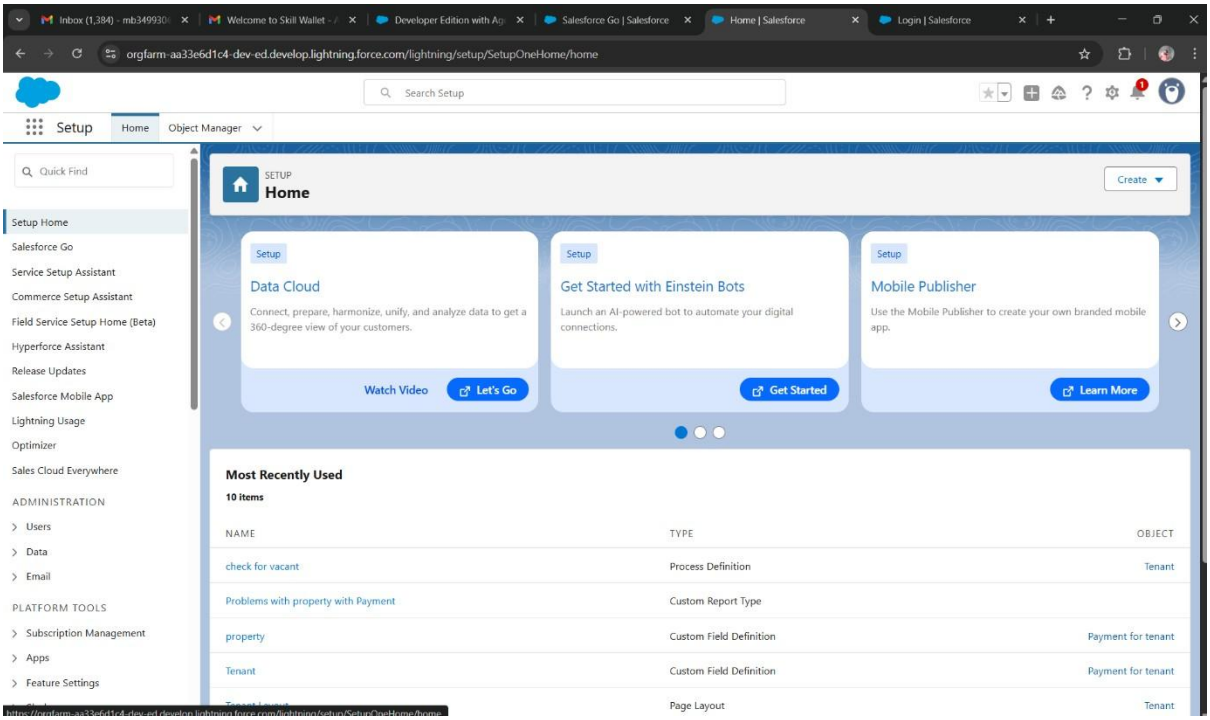


Fig: Logged into the Developer Org

4.2 Customization of Objects, Fields, Validation Rules, and Automation Salesforce's **declarative (click-based)** tools are used to customize the system:

4.2.1 Custom Objects:

To model lease data, the following custom objects are created:

- **Property**
- **Tenant**
- **Lease**
- **Payment**

Salesforce's declarative tools were used to create custom objects tailored to the Lease Management System.

These objects include: **Property**, **Tenant**, **Lease**, and **Payment**, each representing core business data. They enable structured storage and management of lease-related records. Custom fields, relationships, and validations were added to meet specific project needs.

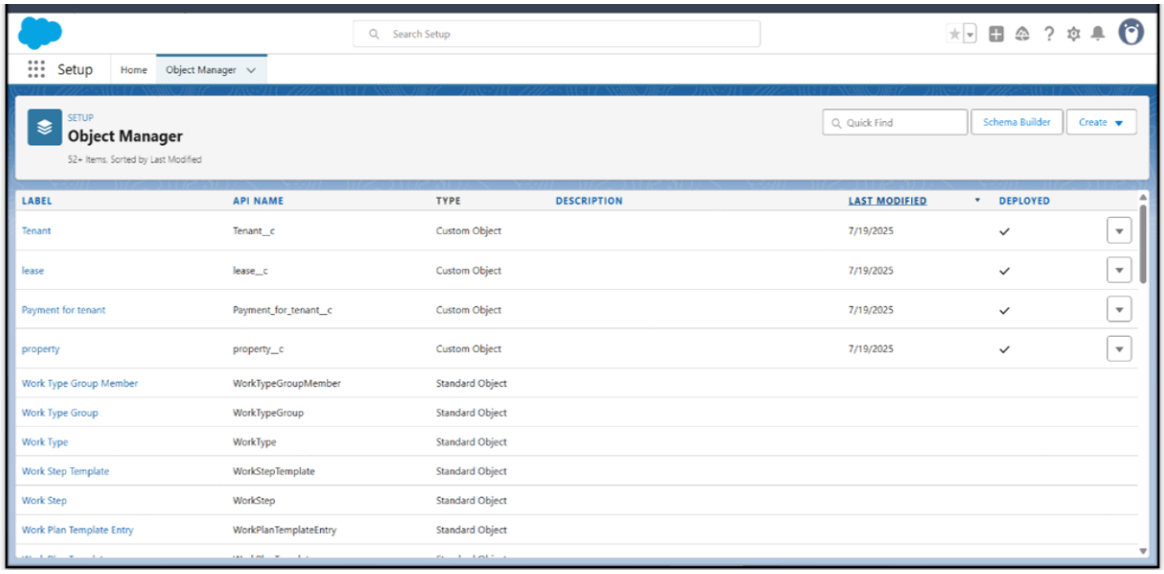


Fig: Custom Objects

4.2.2 Custom Fields:

Each object has custom fields such as:

- **Tenant:**
  - Email__c
  - Phone__c
  - Property__c
  - Status__c

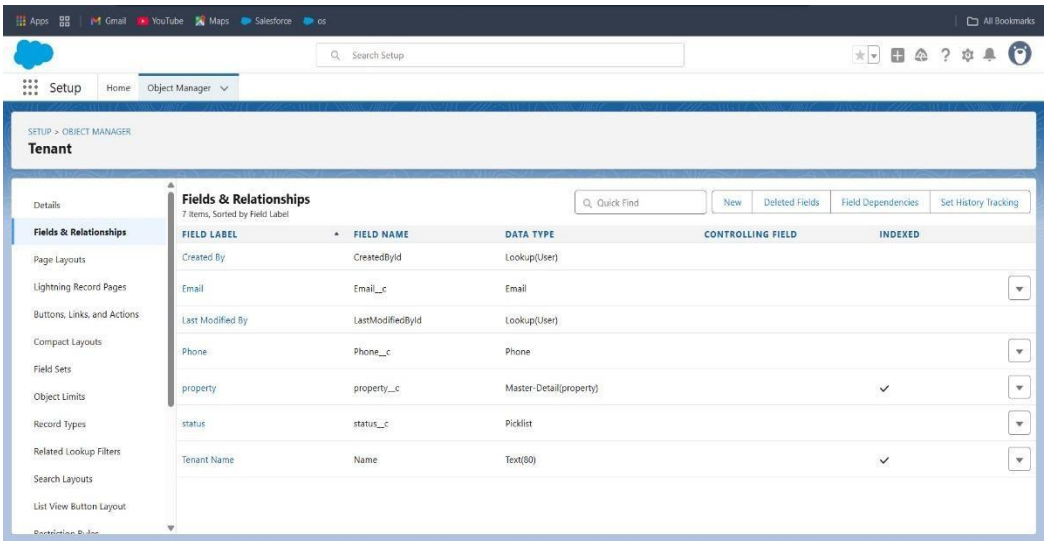


Fig: Custom Fields for Tenant Object

- **Lease:**
  - End_date__c
  - Property__c
  - Start_date__c

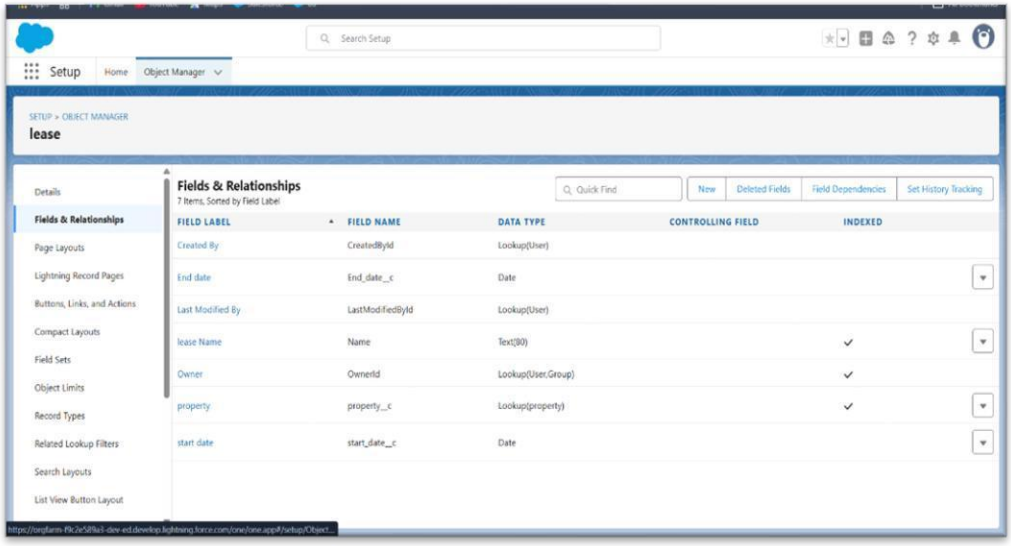


Fig: Custom Fields for Lease Object

Payment for Tenant:

- Amount__c
- Payment_date__c
- Check_for_payment__c
- Property__c
- Tenant__c

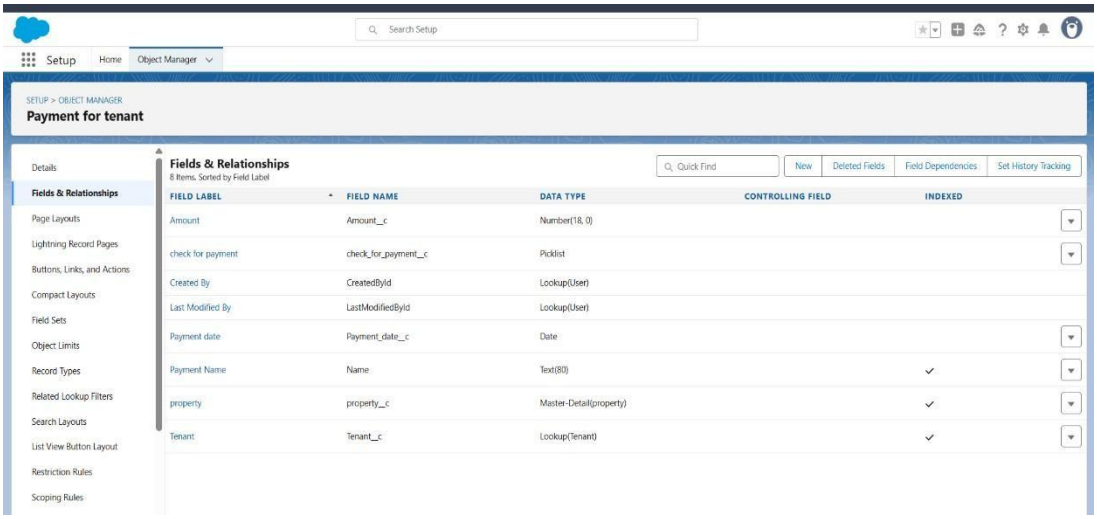


Fig: Custom Fields for Payment for Tenant Object

- **Property:**
  - Address__c
  - Name__c
  - Sqft__c
  - Type__c

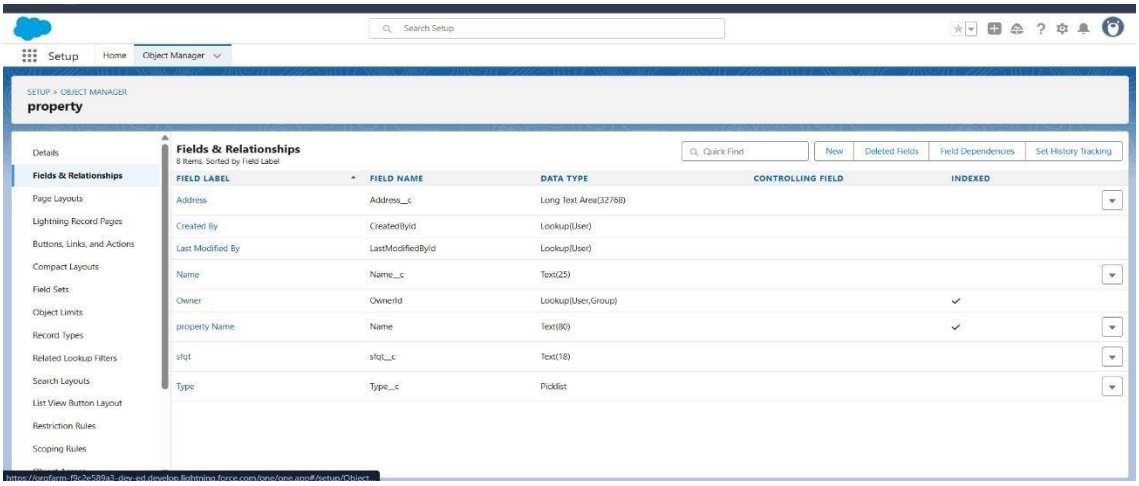


Fig: Custom Fields for Property Object

4.2.3 Validation Rules:

Validation rules ensure data accuracy. Examples:

- Lease end date must be after the start date.
- Rent amount must be greater than zero.
- Payment date cannot be in the future (unless marked as scheduled).

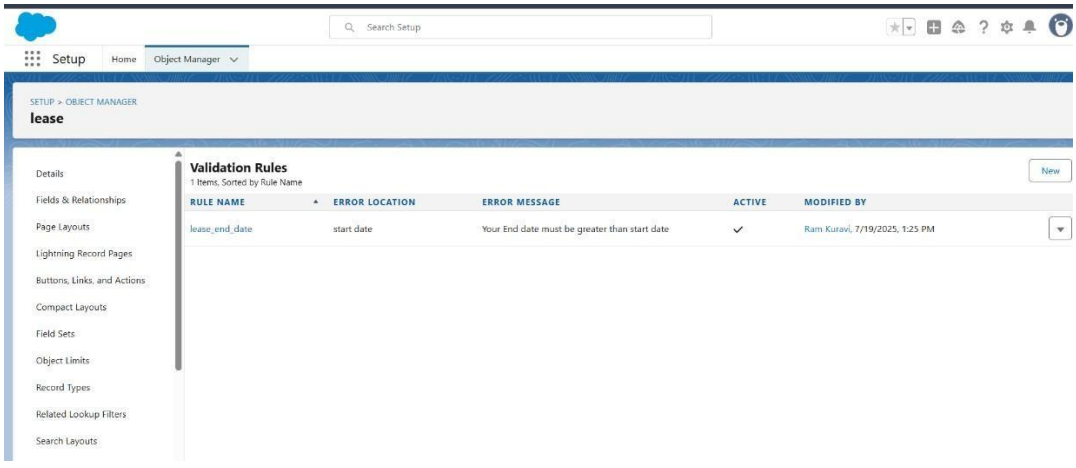


Fig: Validation Rule for Lease Object

4.2.4 Automation Tools:

- **Flows**: Used to collect tenant input, auto-create payment records, and send reminders.

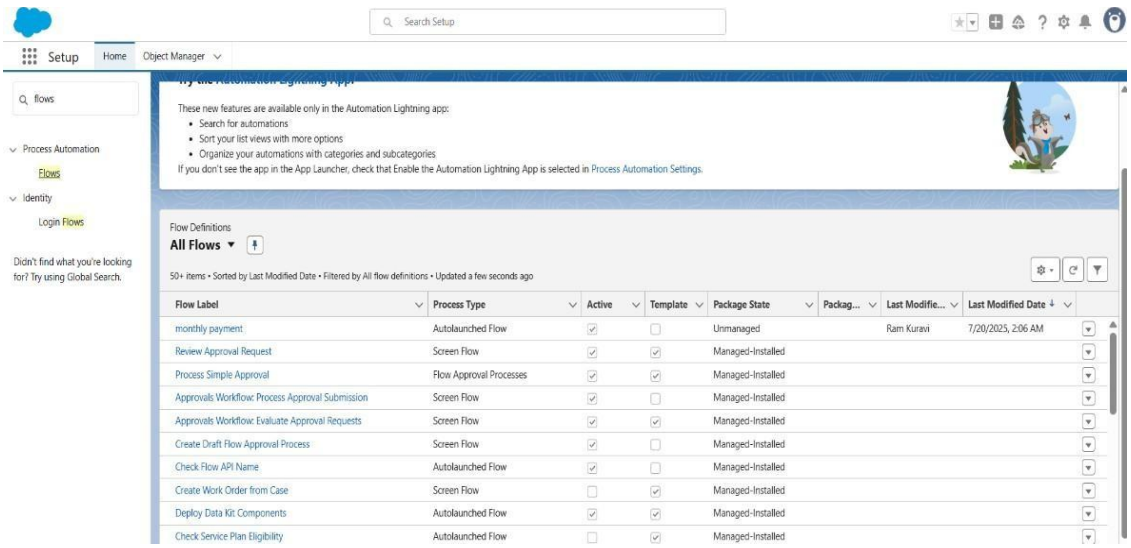


Fig: Record Triggered Flow

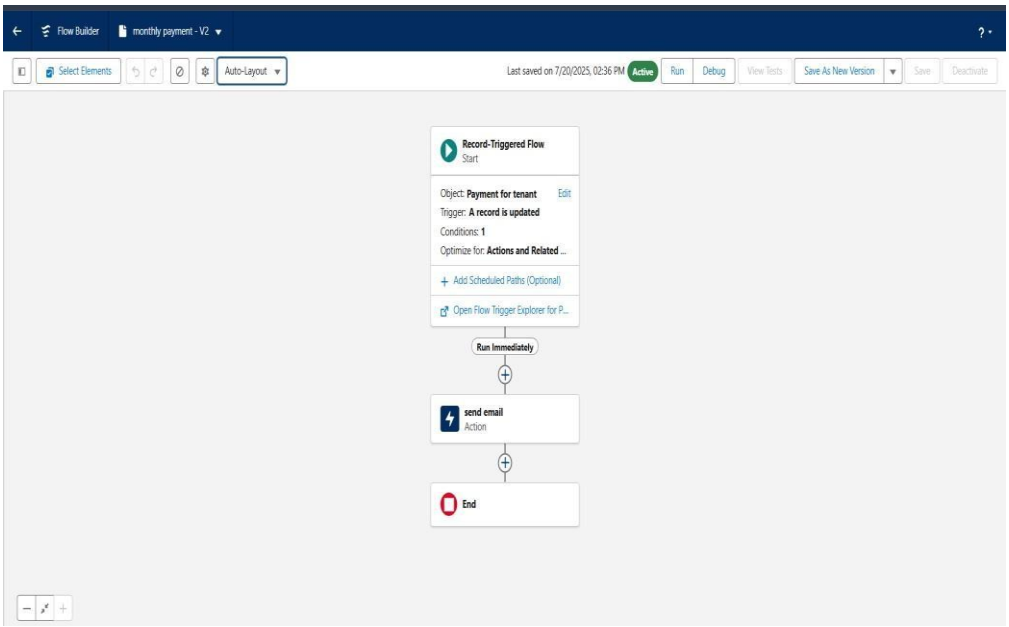


Fig: Monthly Payment Flow

- **Approval Processes**: Used for scenarios like lease renewal approval by a manager.

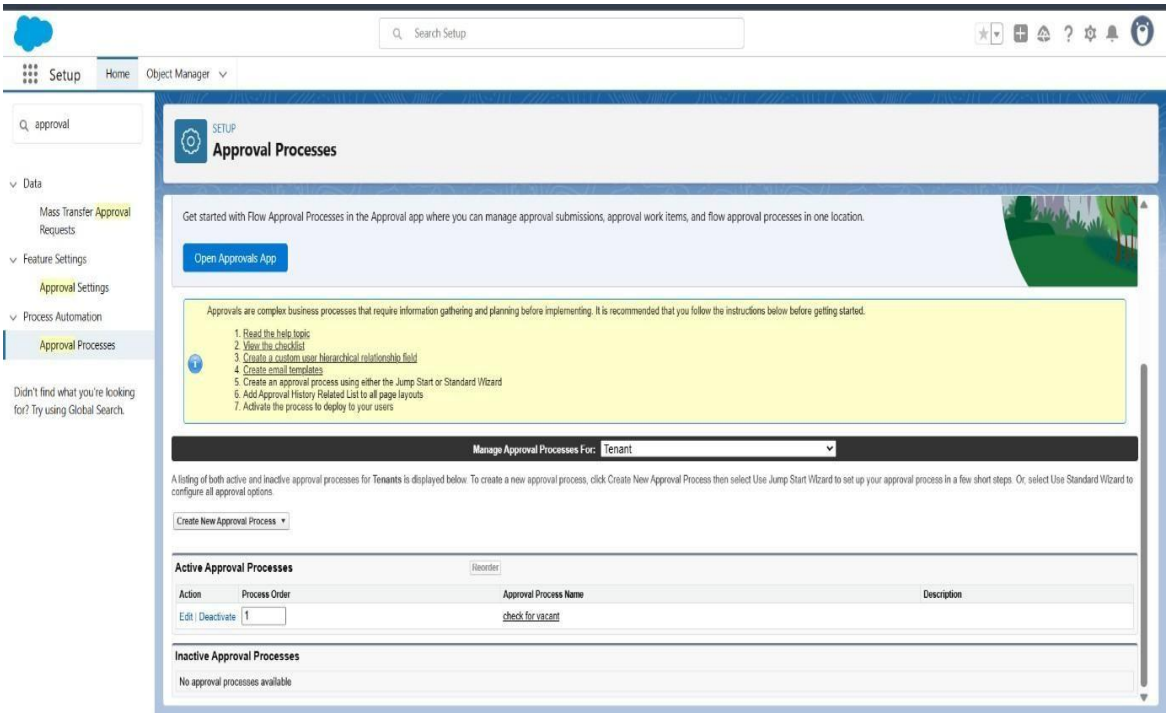


Fig: Creation of Approval Process

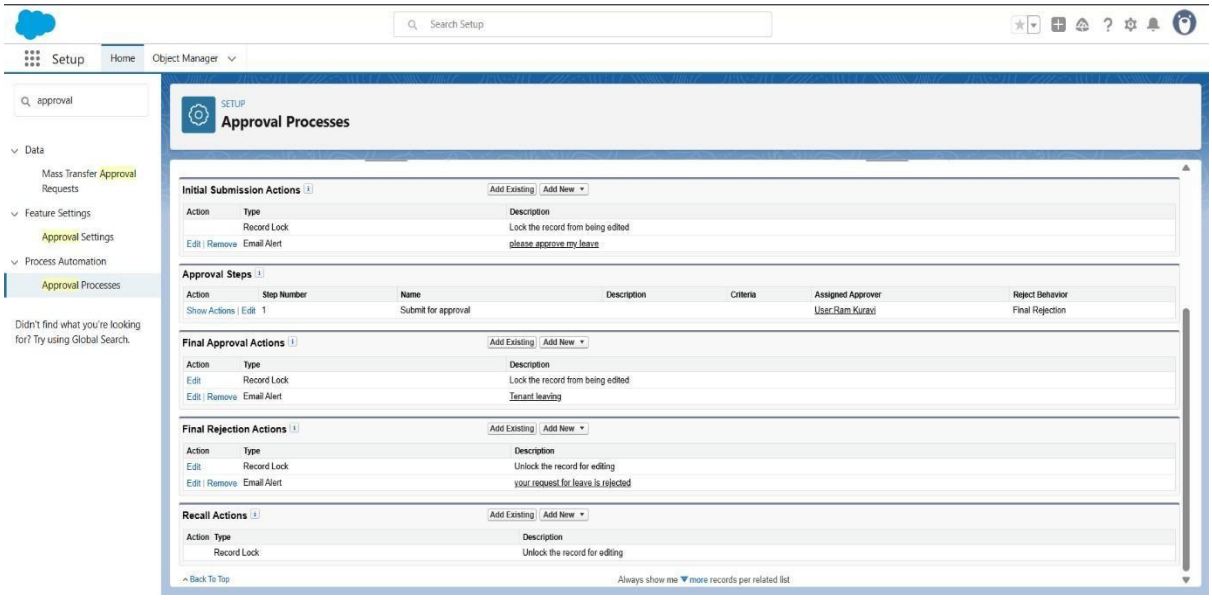


Fig: Check For Vacant Approval Process

4.3 Apex Classes, Triggers, and Asynchronous Apex

When point-and-click tools are not enough, **Apex programming** is used to implement custom logic.

4.3.1 Apex Classes:

These are reusable blocks of code used for custom backend logic

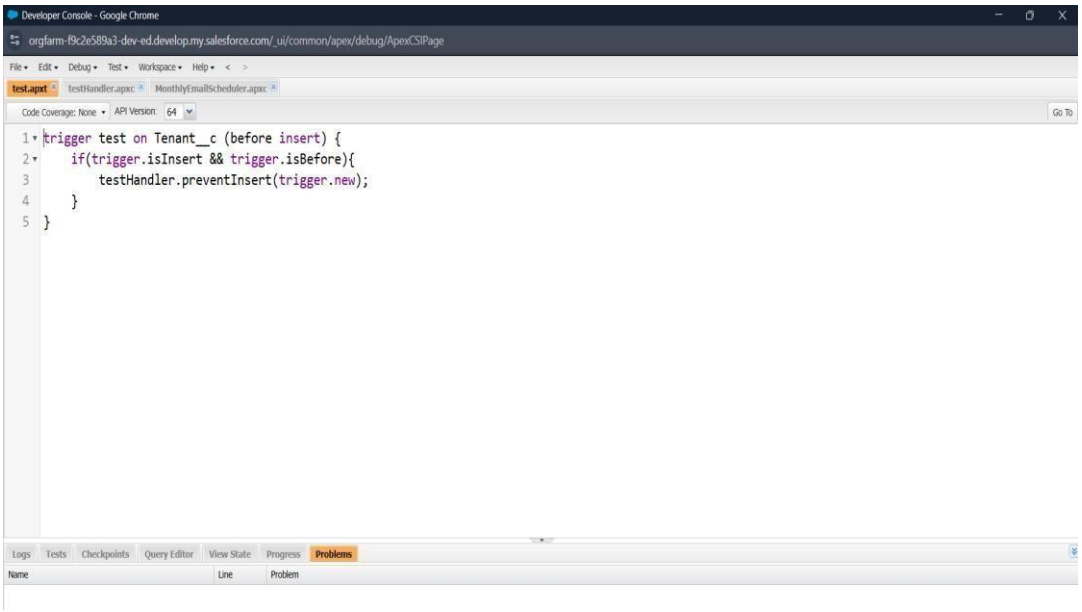


Fig: Test Apex Class

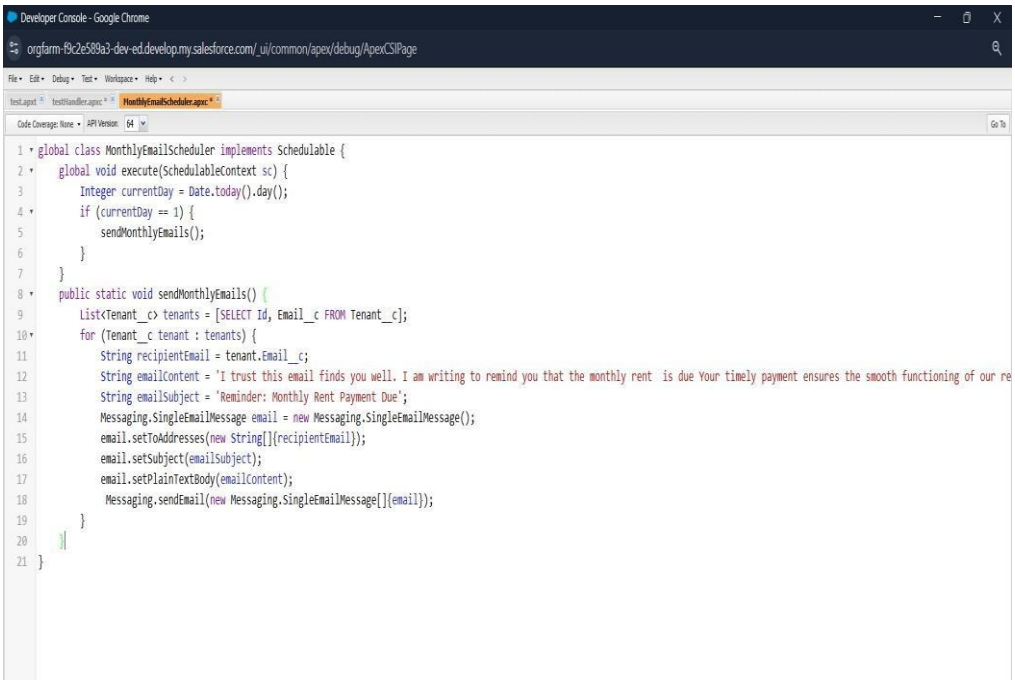
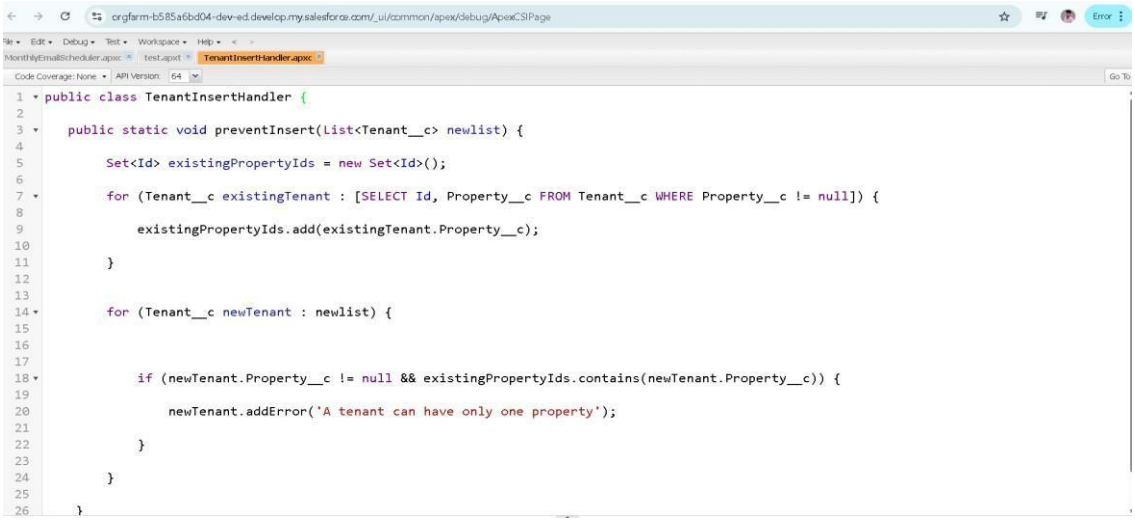


Fig: MonthlyEmailSchedular Apex Class

4.3.2 Apex Triggers:

Triggers are used to perform actions automatically when a record is inserted, updated, or deleted. Example:

- On creation of a Lease record, auto-generate a related Payment schedule.
- On Payment update, update Lease status if fully paid.



```
public class TenantInsertHandler {

    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');

            }

        }
    }
```

Fig: TenantInsertHandler Apex Trigger

4.3.3 Asynchronous Apex:

Used when tasks take longer to run or need to happen in the background:

- **Scheduled Apex**: To check daily for upcoming lease expirations.

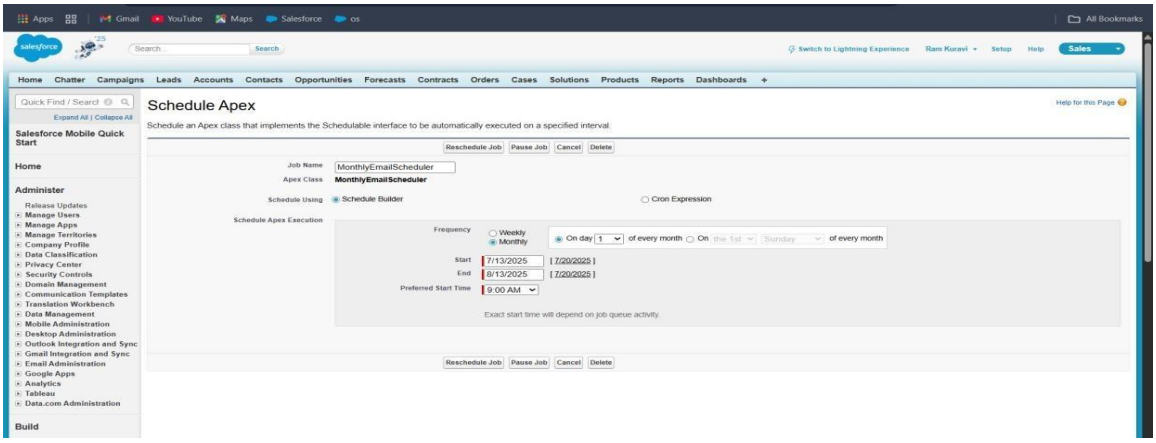


Fig: Scheduled Apex

5. UI/UX DEVELOPMENT & CUSTOMIZATION

This phase focuses on designing and customizing the user interface to provide a smooth and efficient experience for both property managers and tenants. Salesforce Lightning Experience tools are used to build a clean, user-friendly, and role-specific UI.

5.1 Lightning App Setup through App Manager

Using **App Manager**, a custom **"Lease Management" Lightning App** is created with a branded name, logo, and navigation items such as:

- Tenants
- Properties
- Payments

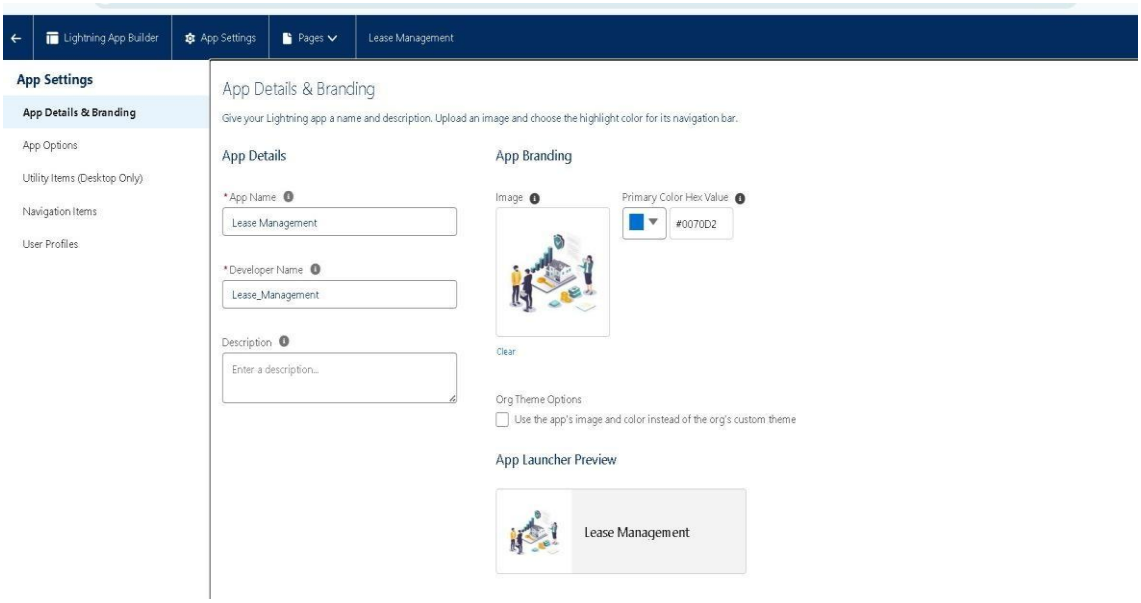


Fig: Lightning App

This setup ensures quick access to all key objects in one place, improving user workflow.

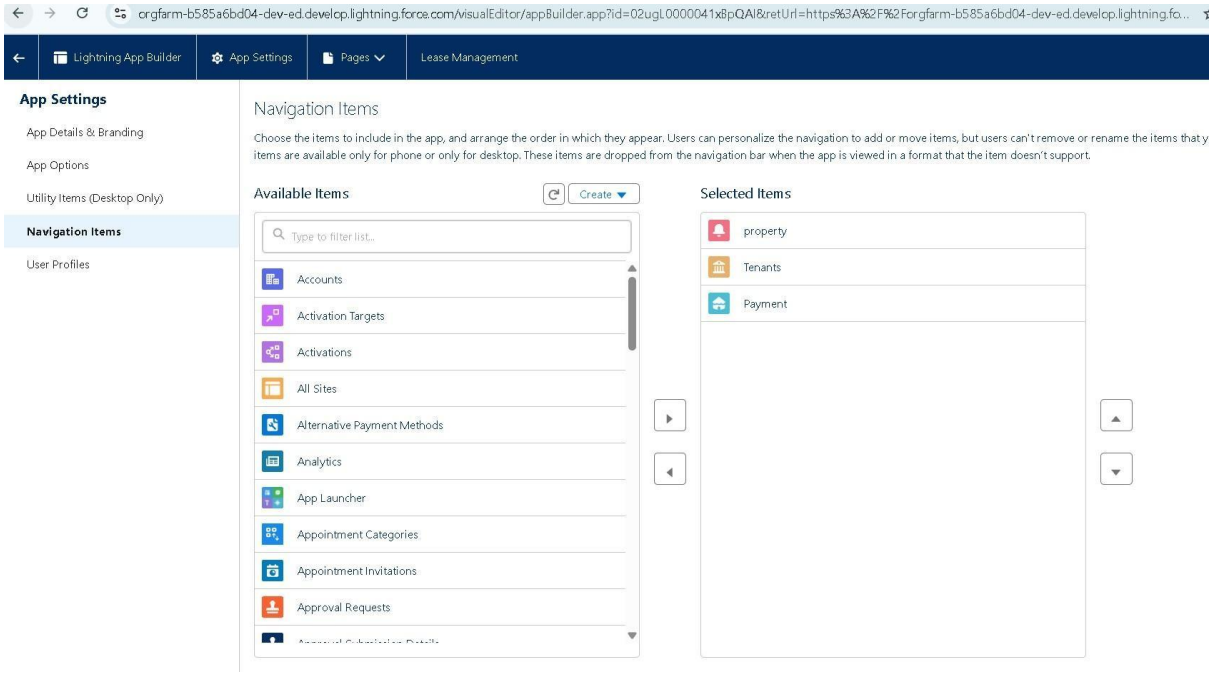


Fig: Objects under Lease Management Application

The next step involves in selecting the system administrator as the default user for the user profiles.

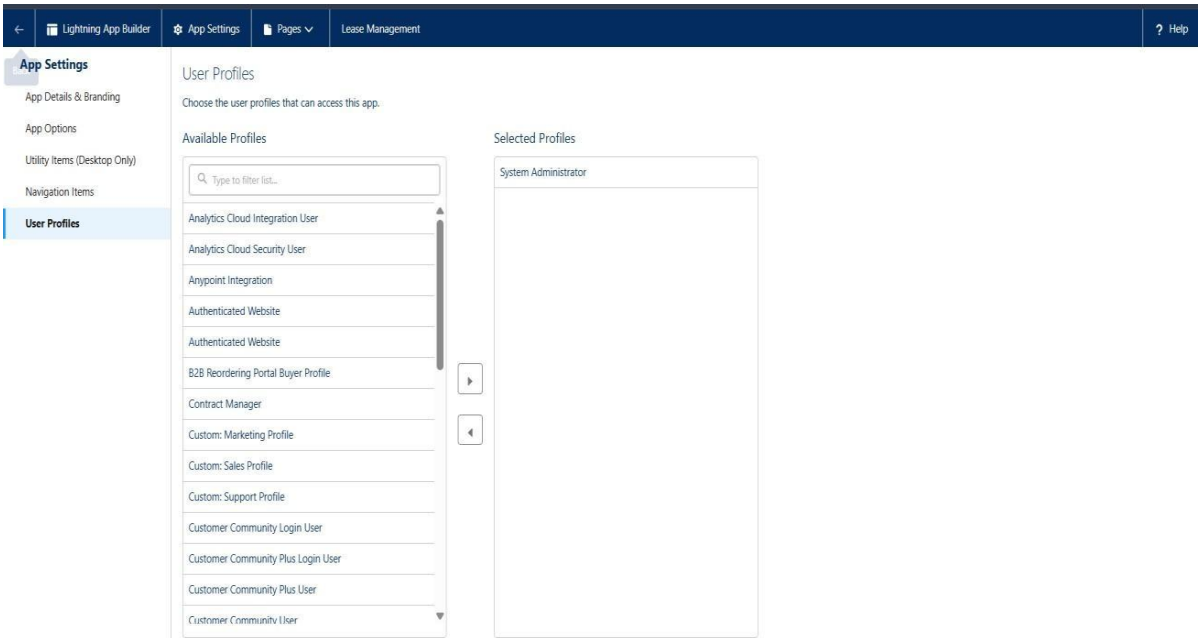


Fig: Users Who can Access the Application

5.2 Page Layouts and Dynamic Forms

Page Layouts are customized for Tenant object to display only relevant fields for each user type.

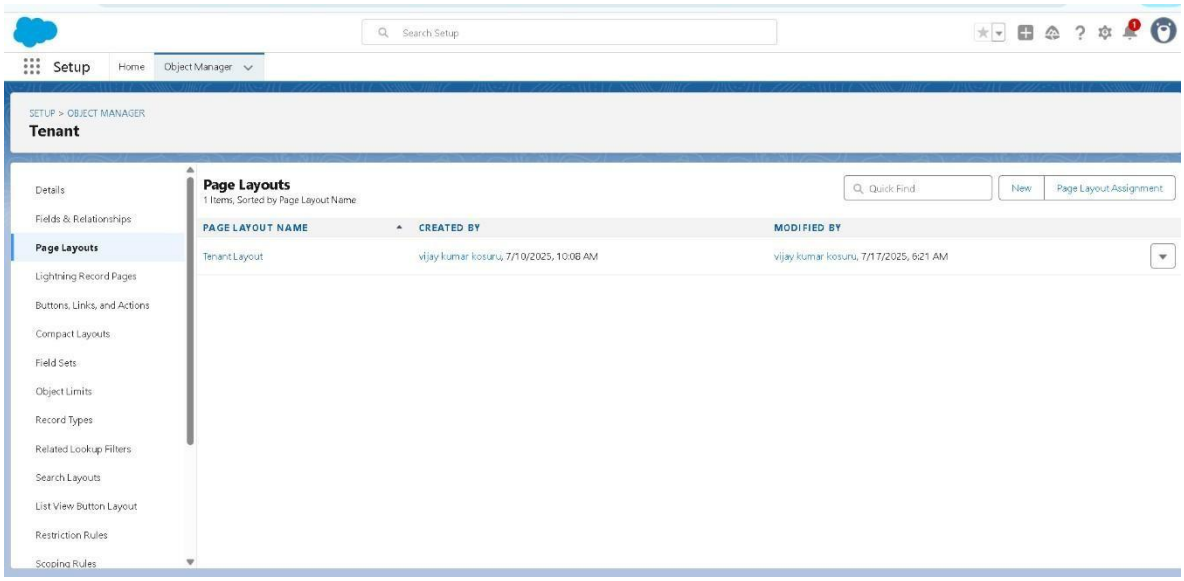


Fig: Page Layout for Tenant Object

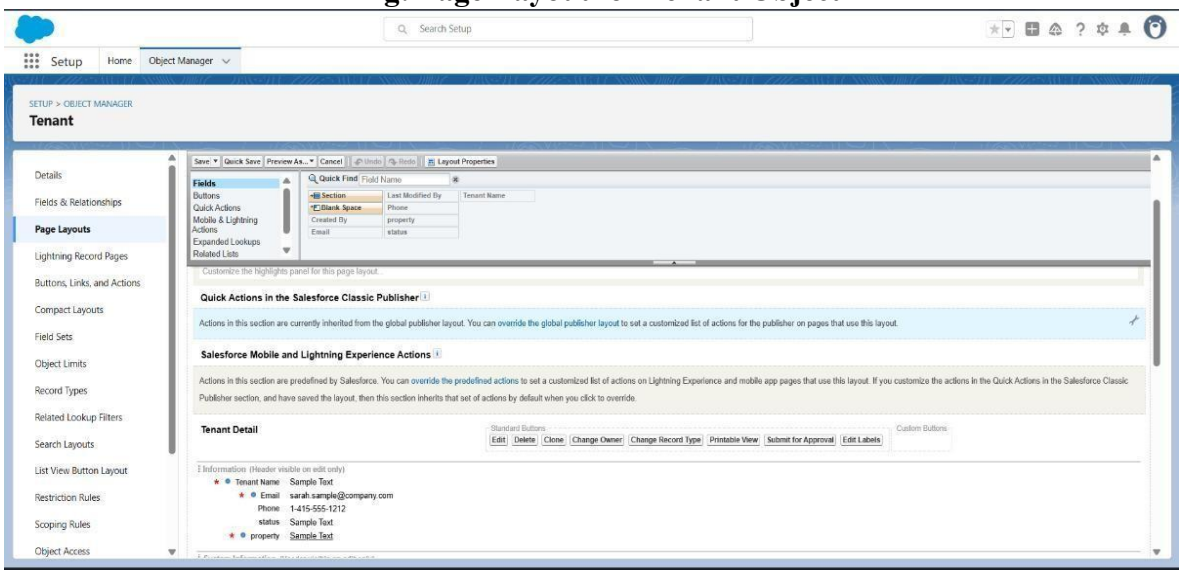


Fig: Submit For Approval Settings

This ensures the interface stays clean and relevant for users, reducing clutter and confusion.

5.3 Lightning Pages

Lightning Record Pages were customized for each object to improve how data is displayed to users:

- **Use of Tabs**: Tabs were used to organize related information clearly within the record page, ensuring a clean and structured layout.

This setup creates a simple, role-friendly interface, making navigation easier and enhancing the overall user experience.

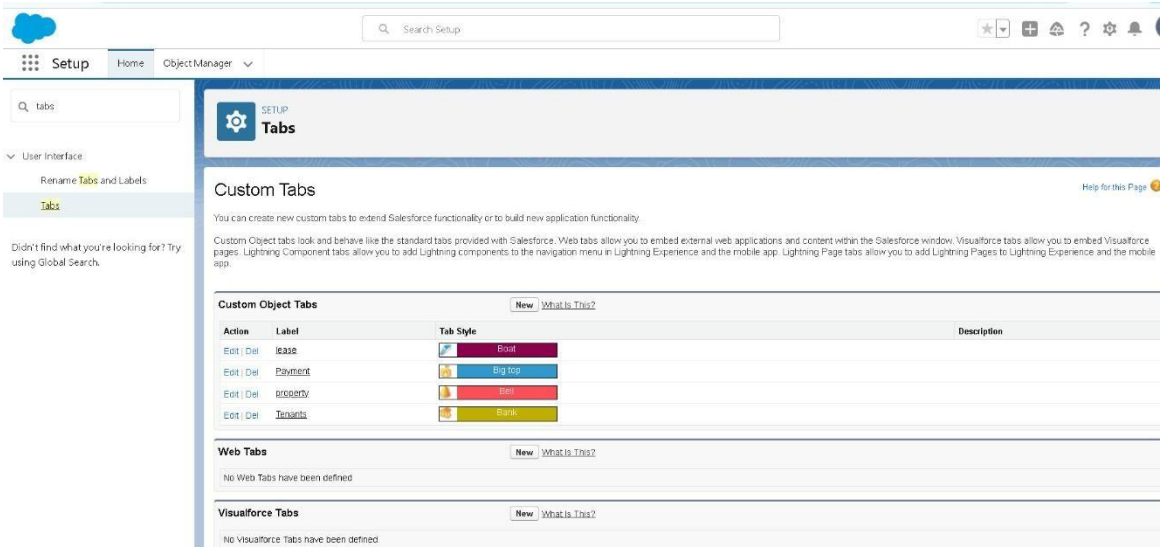


Fig: Custom Tabs Creation for Objects

6. DATA MIGRATION, TESTING & SECURITY

This phase covers the essential activities of migrating lease-related data into Salesforce, thorough testing of the developed features, and ensuring security measures are properly implemented. The objective is to guarantee data accuracy, system reliability, and protection of sensitive information before the system goes live.

6.1 Testing

Testing was performed to validate the functionality and accuracy of all implemented automation and business logic in the system. This phase ensures that the Lease Management System performs as expected under real-world scenarios.

6.1.1 Apex Trigger Testing

A custom Apex trigger was developed to enforce the rule that each property can be assigned to only one tenant at a time. To test this:

- A test class was created simulating insertion of tenant records.
- It confirmed successful creation when a property was free.
- It verified the trigger correctly throws an error if a property is already assigned, preventing data inconsistencies.

Test logs and results demonstrated the trigger's effectiveness in maintaining data integrity.

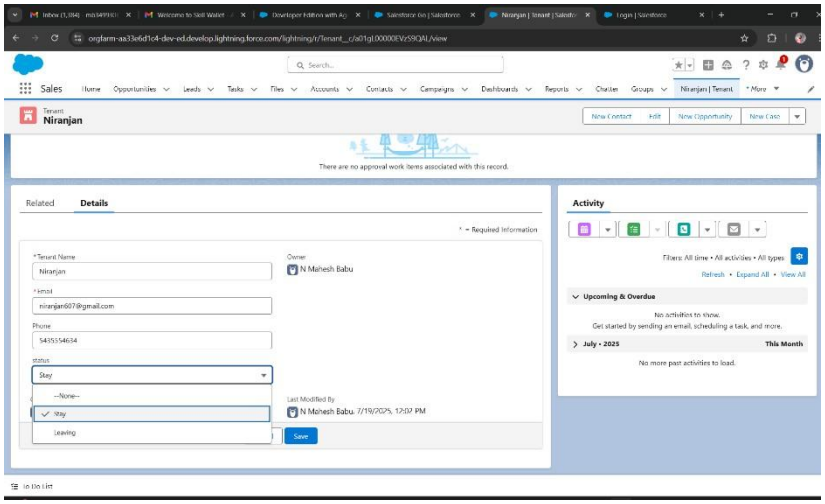


Fig: Testing for property to be unique for tenant

6.1.2 Approval Process Testing

The tenant leave request approval workflow was tested through multiple scenarios:

- Submission of leave requests by tenants triggered the approval process.
- Approvers received real-time notifications and emails.
- Both approval and rejection paths were tested to ensure that tenant status updated accordingly, and the proper email templates were sent to notify tenants.

Screenshots of submission forms, approval screens, and notification emails are documented as evidence of successful workflow execution.

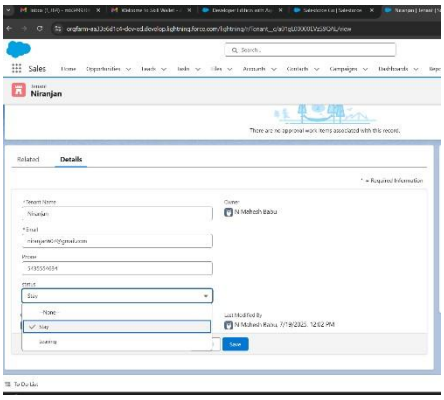


Fig: Submission of leave requests to approver

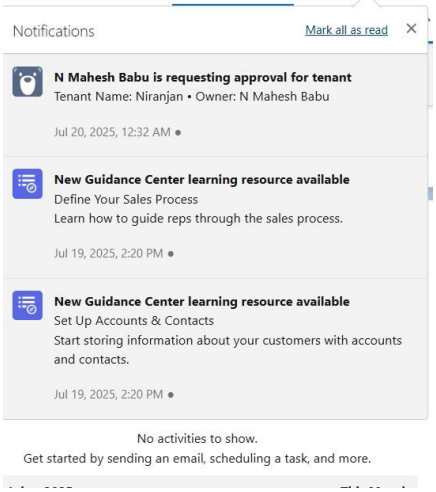


Fig: Mails received to approver

6.1.3 Flow Testing

The system includes a record-triggered flow that sends payment confirmation emails automatically when a tenant's payment status changes to "paid." Testing involved:

- Updating payment records to “paid” status.
- Monitoring flow execution in Salesforce.
- Confirming that tenants received correctly formatted confirmation emails.

This automated communication improves tenant engagement and reduces manual workload.

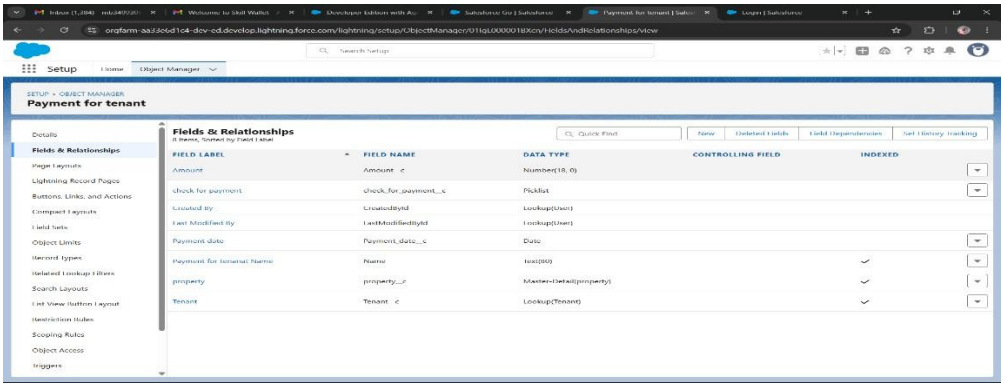


Fig: Fields in Payment Object

A confirmation mail received to the tenant about the payment of the monthly rent.

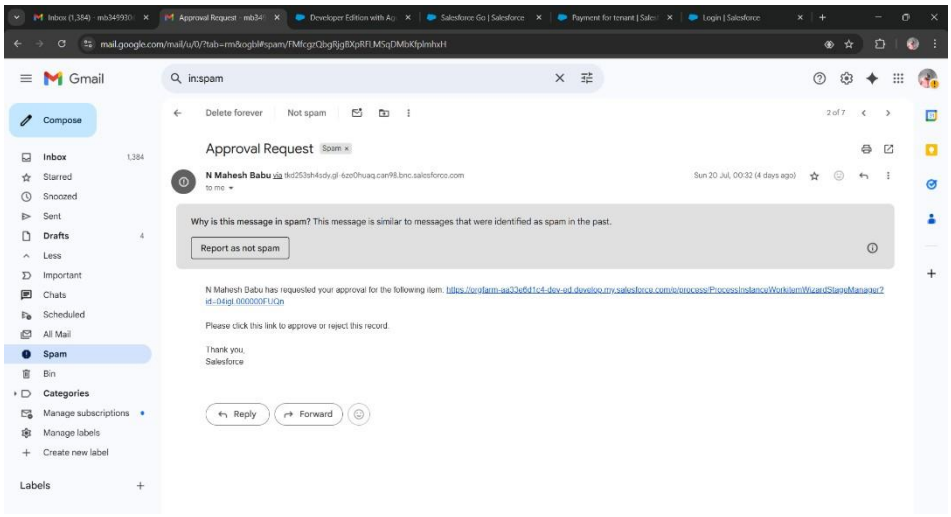


Fig: Confirmation Email Received

6.1.4 Scheduled Apex Testing

To ensure timely reminders for rent payments, a scheduled Apex class was developed and tested. The scheduler runs monthly on the 1st day to send payment reminder emails. Testing involved:

- Manually triggering the scheduled job to simulate its execution.
- Verifying emails were dispatched to all tenants with pending payments.

Test documentation includes logs from the scheduler and screenshots of sent emails.

Fig: Executing MonthlyEmailSchedular Apex Class

The execution of the MonthlyEmailSchedular class.

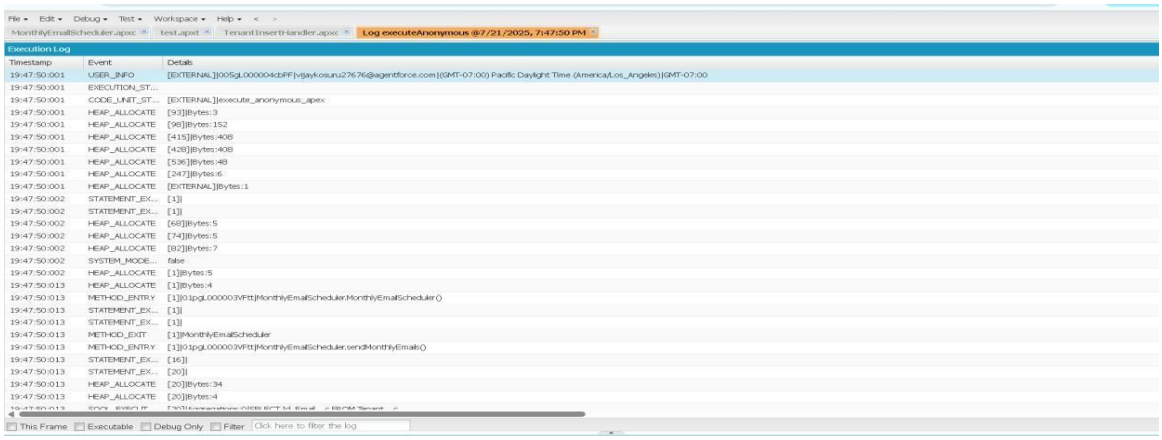


Fig: Execution Logs

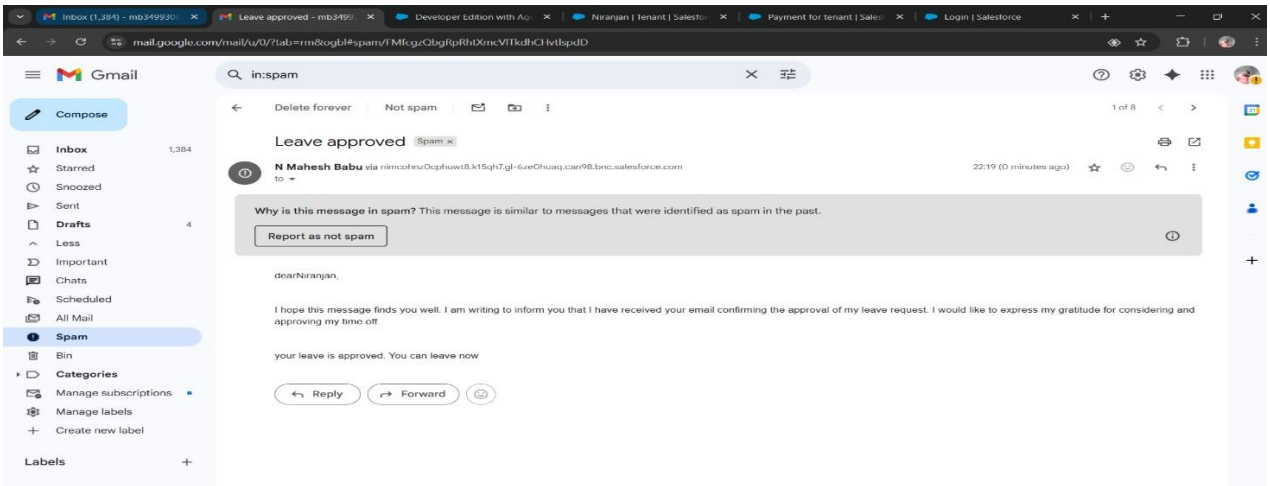


Fig: Remainder Mail Received to Tenant

6.2 Documentation and Evidence

- Detailed test cases were created covering all implemented features.
- Input data, execution steps, and expected outcomes were recorded for each test.
- Screenshots captured actual results to verify correct functionality.
- Documentation ensures transparency and aids future maintenance.
- Provides confidence in system readiness for deployment.

The execution of the Apex class MonthlyEmailSchedular. The following figure shows the execution logs of the Apex class.

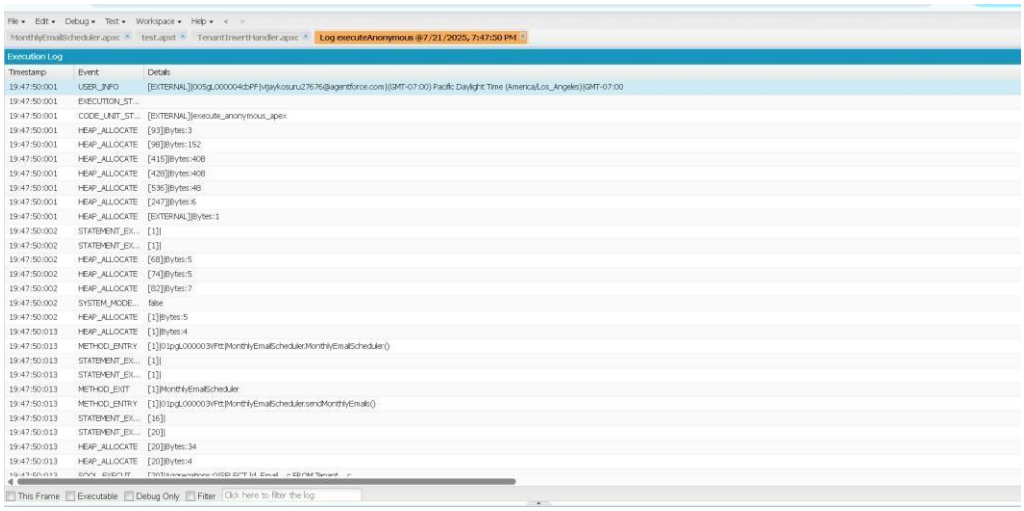


Fig: Execution Logs Evidence

The notifications are received to the user.

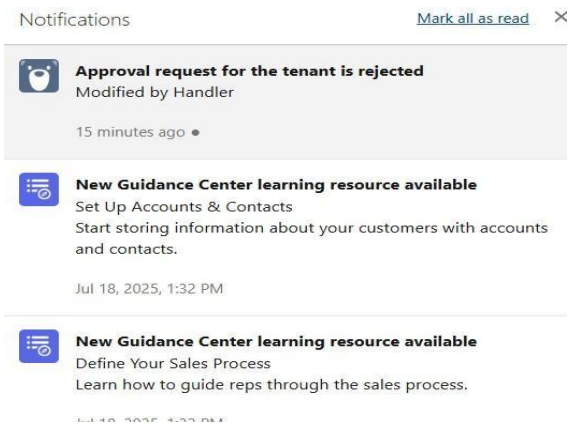


Fig: Notifications Evidence

Emails received by the tenant sent by the user or owner through automated process.

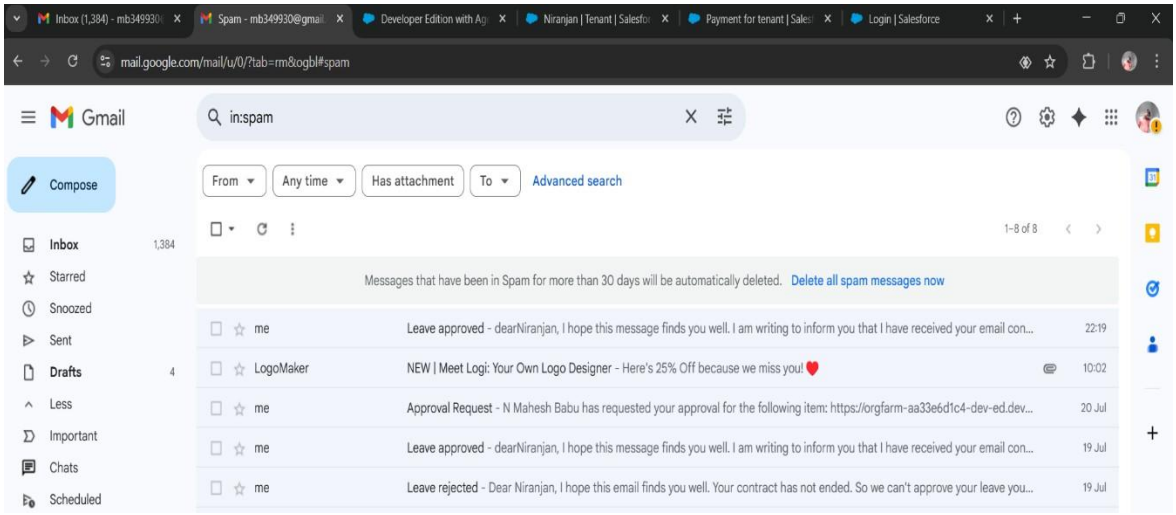


Fig: Emails Received Evidence

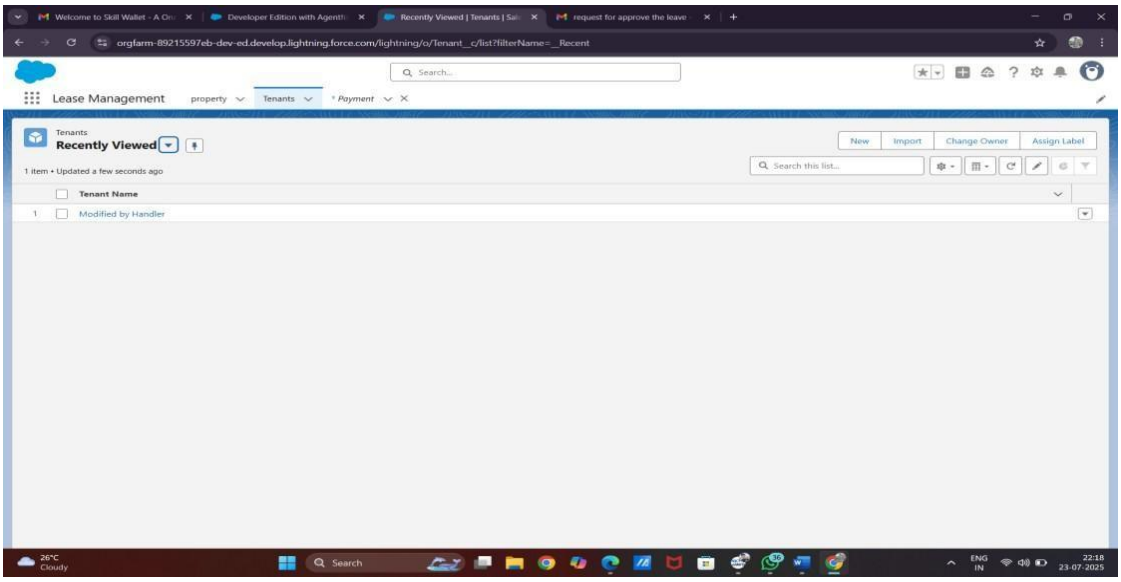


Fig: Approval Page

The above figure determines the approval page for the owner. The owner can approve, reject or reassign the tenant request.

The approver can accept or reject the user request.

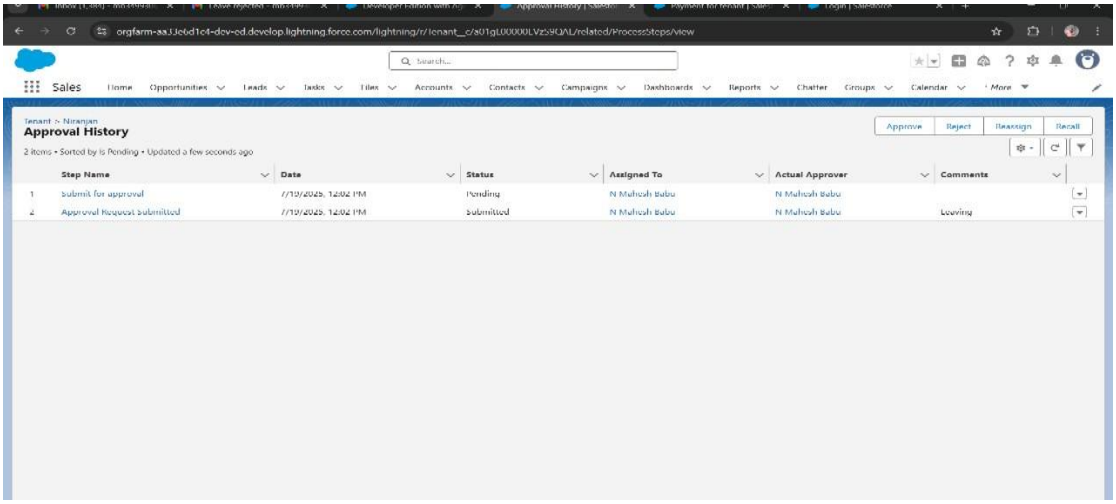


Fig: Leave Rejection by Approver

The approval or rejection email is received to the tenant.

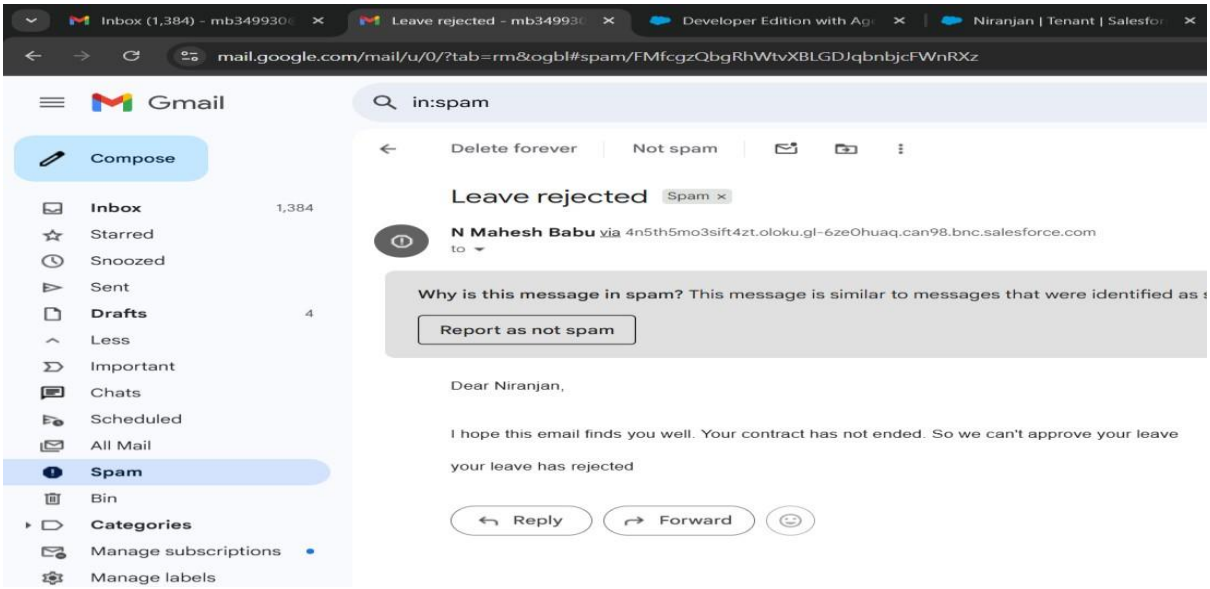


Fig: Leave Rejected Email

7. DEPLOYMENT, DOCUMENTATION & MAINTENANCE

7.1 Document Strategy :

To ensure a smooth transition from development to production, a well-defined deployment strategy has been implemented. The primary method used for deployment is **Salesforce Change Sets**, allowing for the structured movement of metadata between environments. This approach reduces risks, ensures stability, and maintains system integrity.

7.1.1 Key Components of Deployment:

- **Developer Sandbox Usage:**
  All development and testing activities are conducted in Salesforce Developer Edition or Sandbox environments. This isolate testing from the live system and allows teams to experiment and validate without affecting end users.

- **Outbound Change Sets:**
  These are created in the Sandbox environment and include all

  the necessary components such as: ○ Custom Objects and Fields

  ○ Validation Rules ○ Process Builders and Flows ○ Apex

  Classes and Triggers ○ Lightning Pages and Components

- **Inbound Change Sets:**
  The packaged Change Sets are received in the **Production Org** and reviewed. Once verified, they are deployed after proper validation and testing, ensuring that only stable and approved features go live.

- **Pre-deployment Testing:**
  Before any deployment, a complete testing cycle is conducted in a **Full Sandbox**

  environment. This includes:

  ○ Functional Testing

  ○ Integration Testing

  ○ Regression Testing

This helps identify potential bugs, configuration issues, or deployment errors early in the process.

7.2 Advanced Deployment Tools (Optional):

For larger projects or distributed teams, additional tools are considered for improved control and automation:

- **SFDX CLI (Salesforce DX)** for script-based deployment and continuous integration
- **Git** for version control and team collaboration

This deployment approach ensures consistency, minimizes risks, and supports scalable growth in the future.

7.3 System Maintenance & Monitoring

Post-deployment, the system needs to be actively monitored and maintained to ensure continued performance and data reliability. A structured maintenance plan has been put in place:

Key Activities:

- **Regular Data Backups:**
  Scheduled backups are configured to safeguard critical information such as lease contracts, tenant records, payment data, and related documents. Backups ensure business continuity in case of data loss or system failure.
- **Performance Monitoring:**
  Tools provided by Salesforce are used to continuously monitor system performance: ○ **Debug Logs** to track system executions and identify performance bottlenecks ○ **Setup Audit Trail** to monitor configuration changes and user actions ○ **Salesforce Health Check** to assess security settings and overall system health
- **User Feedback Collection:**

Regular feedback is gathered from end-users (property managers, admins, tenants) to identify UX/UI improvements and minor bugs. Feedback is logged and analyzed to drive iterative enhancements.

- **Scheduled Enhancements & Upgrades:**

Based on usage patterns and evolving business needs, new features or process optimizations are scheduled periodically. These may include:

- New report types
- Enhanced dashboards
- Additional automation (e.g., lease expiry follow-ups)
- Workflow optimizations

- **Security & Permission Reviews:**

As teams grow or roles change, it's crucial to ensure access levels are aligned.

Periodic reviews are conducted for:

- Role hierarchy
- Profile settings
- Permission sets and sharing rules

These reviews ensure that users only access the data they're authorized to, maintaining trust and data protection compliance.

7.4 Documentation & Troubleshooting

Proper documentation is essential for smooth adoption, ongoing support, and future scalability. Two levels of documentation have been created:

7.4.1 User Documentation (User Guide & Admin Manual):

Designed for end-users and system administrators, this includes:

- **Step-by-step instructions** for key tasks such as:
  - Creating a new lease record
  - Viewing tenant profiles and payment history

- Generating and exporting reports
- Updating lease statuses (active, expired, terminated)

- **Common Troubleshooting Tips:**
- Handling record saves errors caused by validation rules
- What to do when approval processes or flows don't trigger
- Resolving login issues and access permission problems
- **FAQs Section:**
  Answers to frequently asked questions, separated for general users and system administrators. Helps reduce support requests and onboarding time for new users.

7.4.2 Technical Documentation:

Created for developers and future maintainers, this includes:

- **Data Model Diagram:**
  A visual representation showing relationships between key custom objects like Leases, Tenants, Properties, and Payments.
- **Custom Objects & Fields List:**
  A detailed list of all custom-built objects and fields, including field types, API names, and usage.
- **Automation & Apex Details:**
  Documentation of flows, process builders, triggers, Apex classes, and any scheduled jobs implemented in the system.
- **Deployment Checklist:**
  A pre-deployment and post-deployment checklist used to ensure all components were tested, dependencies addressed, and configurations verified.

8. CONCLUSION

The **Lease Management System** offers a comprehensive solution to modernize and streamline lease operations. By automating routine tasks, enhancing data visibility, and ensuring secure and user-friendly access, it effectively reduces administrative burden while increasing overall efficiency. With features like real-time updates, automated alerts, detailed reporting, and scalable architecture, the system empowers property managers and tenants alike to stay informed and proactive. Its flexible design ensures it can adapt to changing business needs, making it a sustainable and future-ready platform.

In summary, this system not only simplifies lease management but also contributes to smarter decision-making, improved compliance, and long-term operational success.

Future Enhancements:

1. **Tenant Portal**
   Develop a self-service portal where tenants can view lease details, make payments, request maintenance, and track payment history.
2. **Automated Lease Renewal**
   Implement logic to automatically notify tenants of lease expiry and offer renewal options with predefined terms.
3. **Mobile Accessibility**
   Build a mobile app or enhance mobile responsiveness to allow landlords and tenants to manage leases and payments on the go.
4. **Integration with Payment Gateways**
   Integrate secure payment gateways (e.g., Razorpay, Stripe) for seamless rent transactions directly within Salesforce.
5. **Analytics Dashboard**
   Add dashboards and charts to provide insights into property occupancy, rent collection trends, and tenant behaviour.
6. **Maintenance Tracking**
   Introduce a module to log and track maintenance issues, assign tasks to vendors, and monitor service timelines.

7. **Enhanced Notification System**
   Use Salesforce Flow or Apex to send SMS alerts in addition to emails for critical updates like payment reminders or approvals.
8. **Multi-Property Management**
   Extend functionality to support property groups, enabling landlords to manage multiple properties under different companies or regions.

Links:

Google Drive Link: https://youtu.be/Cif896Ud7eY?si=6PbXKkaavQYMYTAa

GitHub Link: https://github.com/Mahesh324-Babu/Lease-Management-system-