Report On

Institutional Service Management Platform

By

Hrishiksh Jadhav

Github https://github.com/17Hrishi/SalesforceDeveloper

Video .....

Enrolled From A CRM Application to Manage the Services offered by an Institution

Index of Content

Introduction

Institutional Service Management Platform

The Institutional Service Management Platform is designed to enhance operational efficiency and improve service delivery within educational and institutional environments. As organizations strive to streamline their internal processes and provide exceptional service to students, staff, and stakeholders, this platform serves as a comprehensive solution for managing service requests, tracking operational tasks, and facilitating effective communication among team members. Built on the Salesforce ecosystem, the platform leverages robust CRM capabilities to ensure seamless interactions and management of institutional services.

At its core, the platform enables users to submit service requests across various categories, such as maintenance, IT support, and administrative inquiries. Each request is tracked through a well-defined workflow, ensuring timely resolutions and accountability. With automated ticket assignment, approval processes, and real-time notifications, service agents can prioritize and address requests efficiently, leading to enhanced service quality and user satisfaction.

The platform is equipped with powerful reporting and dashboard features that provide insights into service request trends, response times, and operational performance. Decision-makers can utilize these analytics to identify areas for improvement, optimize resource allocation, and implement strategies to elevate the overall service experience. Additionally, the platform supports integration with third-party tools, expanding its capabilities to meet diverse institutional needs.

To ensure user accessibility, the Institutional Service Management Platform is designed to be mobile-friendly, allowing users to manage service requests on the go. By fostering collaboration and transparency among various teams, the platform not only enhances service delivery but also builds a culture of responsiveness and accountability within the institution.

In summary, the Institutional Service Management Platform represents a significant advancement in service management for educational and institutional organizations. By streamlining processes, automating tasks, and providing actionable insights, this platform empowers institutions to meet the evolving demands of their stakeholders while maintaining high standards of service excellence.

Chapter 1
Requirements

Here are the **requirements** for building the "Institutional Service Management Platform" in Salesforce:

1. User Requirements:

Service Request Submission: Users can submit different types of service requests (e.g., IT support, maintenance).

Ticket Status Tracking: Users can track the status of their requests (New, In Progress, Resolved).

Mobile Access: Accessible through the Salesforce mobile app.

Automated Notifications: Users receive email notifications on request submission and updates.

2. Administrator Requirements:

Ticket Assignment: Admins can assign service requests to teams based on request type.

Priority Setting: Admins can set request priorities (High, Medium, Low).

Approval Processes: Multi-step approval processes for high-priority requests (e.g., facility maintenance, IT purchases).

Escalation Rules: Define rules for overdue tickets to ensure timely resolution.

3. Technical Requirements:

Custom Objects:

Service_Request__c: Custom object for managing service requests (fields: Request Type, Priority, Status, Assigned To).

Automations:

Flows/Process Builder: Automate ticket creation, assignment, notifications, and escalations.

Apex Triggers: Use triggers for advanced automation like ticket auto-assignment.

Reports and Dashboards:

Create custom reports for tracking service requests by status and agent performance.

Dashboards to visualize metrics like time-to-resolution and open tickets by priority.

Mobile App Configuration:

Configure the Salesforce mobile layout to display key fields (Request Status, Assigned Team).

4. Security Requirements:

Role-Based Access Control:

Only authorized users (agents, admins) can access specific service requests.

Field-Level Security:

Protect sensitive fields (e.g., personal information) with field-level security.

5. Testing Requirements:

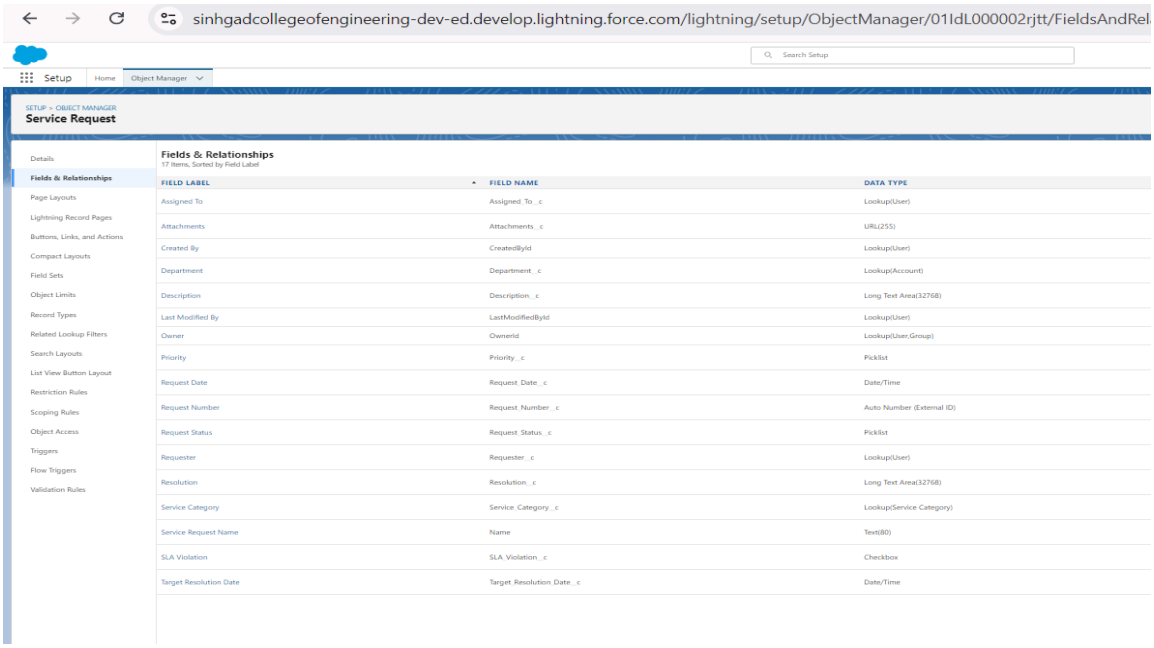
Unit Testing: Test custom Apex code and automations.

Chapter 2
Create Objects

For the "Institutional Service Management Platform" project, here are the custom and standard Salesforce objects that would be used to implement the functionality:

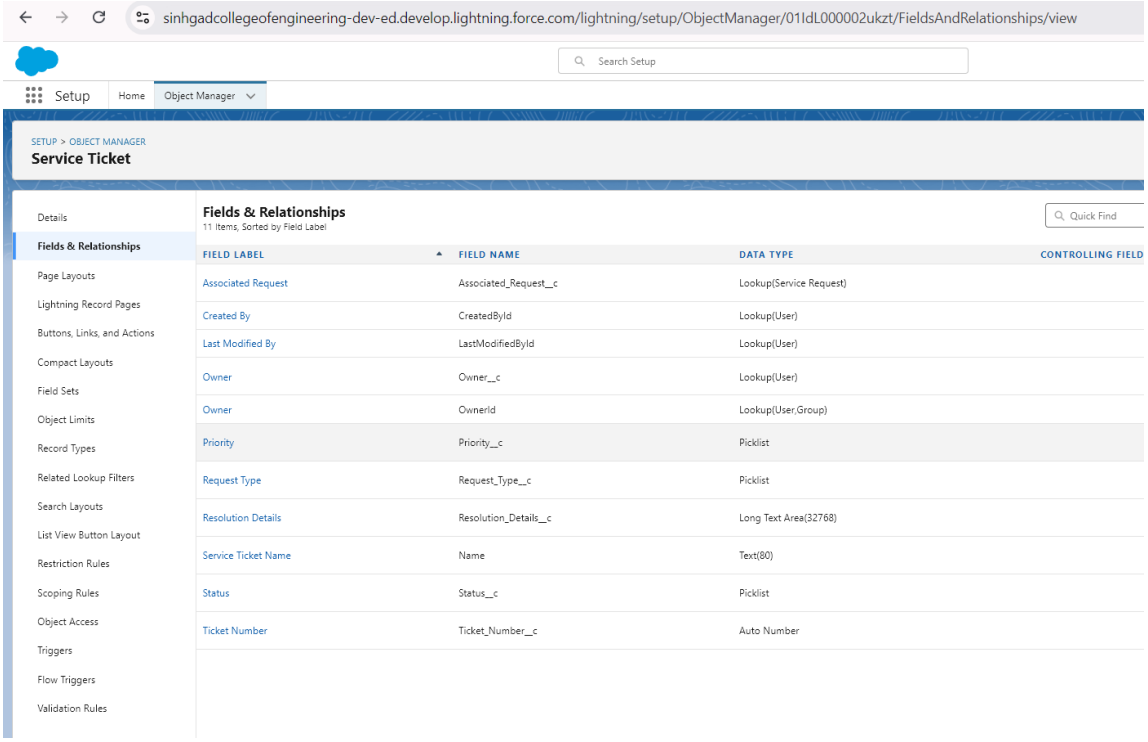
2.1.Custom Objects

1. **Service_Request__c**
    - **Purpose**: This object tracks all the service requests made within the institution.
    - **Key Fields**:
        - Request_Name__c (Text)
        - Description__c (Long Text)
        - Request_Type__c (Picklist: Maintenance, IT Support, General Inquiry, etc.)
        - Status__c (Picklist: New, Submitted, Approved, Rejected, Closed)
        - Priority__c (Picklist: Low, Medium, High)
        - Submitted_By__c (Lookup: User)
        - Assigned_To__c (Lookup: User or Queue)
        - Creation_Date__c (Date/Time)
        - Closure_Date__c (Date/Time)--

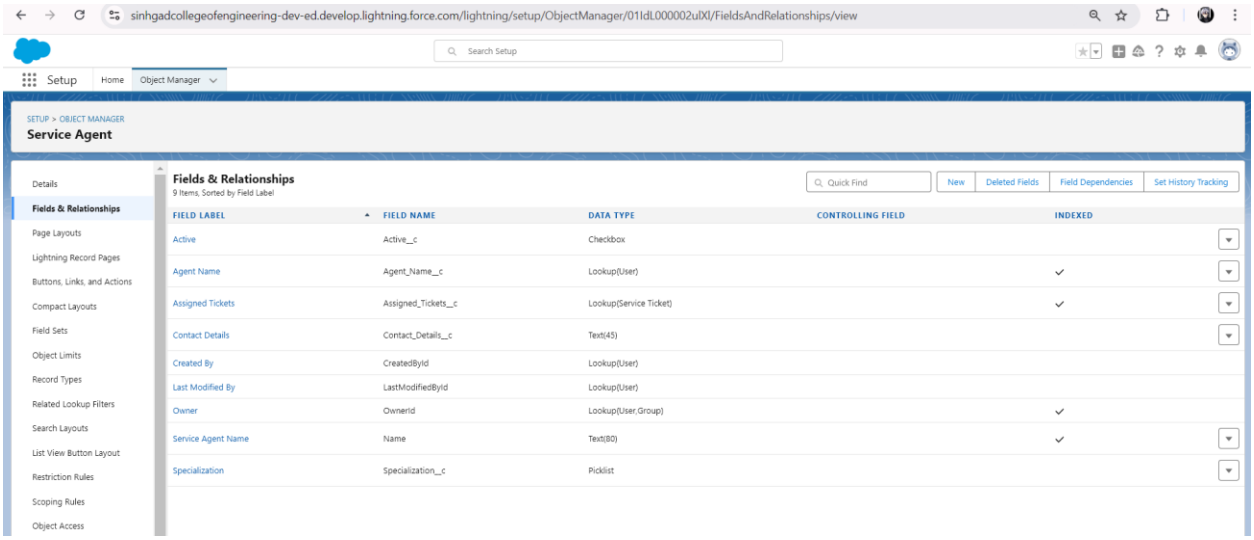


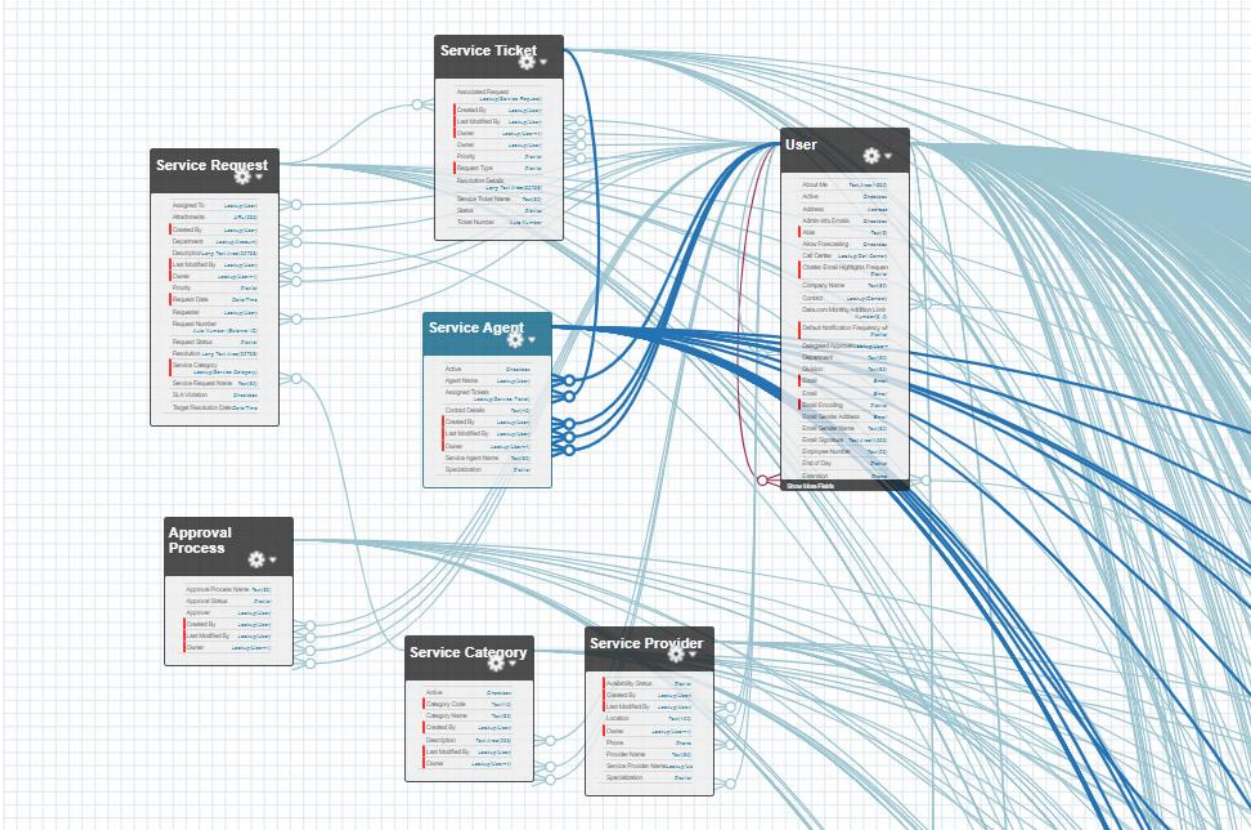
2. **Service_Ticket__c**

- **Purpose**: This object handles the ticketing system related to service requests.
- **Key Fields**:
  - Ticket_Number__c (Auto Number)
  - Request_Type__c (Picklist: Maintenance, IT Support, etc.)
  - Owner__c (Lookup: User or Queue)
  - Status__c (Picklist: Open, In Progress, Resolved, Closed)
  - Priority__c (Picklist: Low, Medium, High)
  - Associated_Request__c (Lookup: Service_Request__c)
  - Resolution_Details__c (Long Text)
  - Created_Date__c (Date/Time)



3. **Service_Agent__c**
   - **Purpose**: This object tracks the service agents responsible for handling requests.
   - **Key Fields**:
     - Agent_Name__c (Lookup: User)
     - Specialization__c (Picklist: Maintenance, IT, General Support)
     - Active__c (Checkbox)
     - Contact_Details__c (Text)
     - Assigned_Tickets__c (Lookup: Service_Ticket__c)

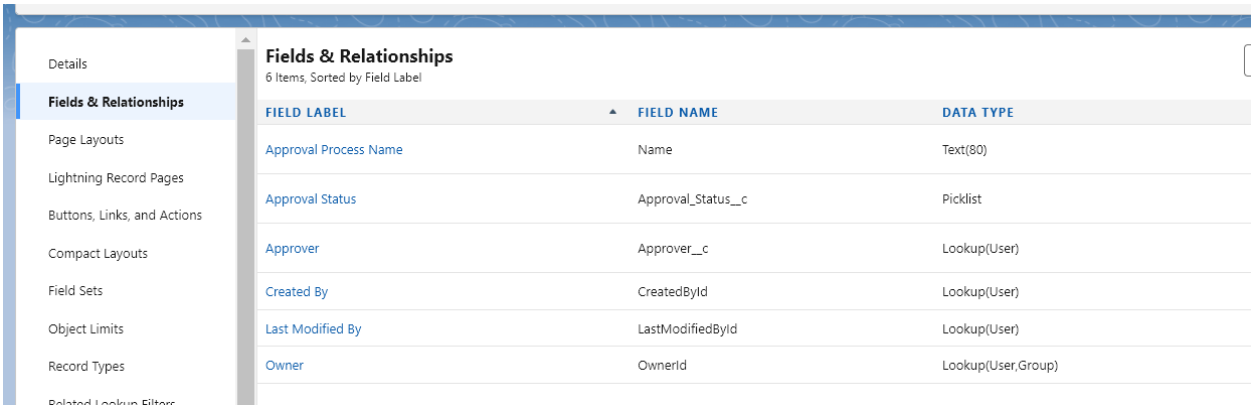




2.2.Standard Objects

1. **User**
   - **Purpose**: Represents the users (students, staff, agents, etc.) interacting with the platform.
   - **Key Fields**:
     - Name

        - Email
        - Role
        - Profile
2. **Group**
    - **Purpose**: Represents queues that are used to assign service tickets to different teams or departments.
    - **Key Fields**:
        - Name
        - Type (Queue)
        - Group Members
3. **Approval Process** (Standard)
    - **Purpose**: Handles the approval workflow for service requests, such as approvals required from department heads or administrators.
    - **Key Fields**:
        - Approval Status
        - Approver
        - Submitted Date



These objects will form the foundation of the Institutional Service Management Platform, helping to streamline request handling, ticket management, and agent assignment.

2.3.Create a Lightning App:

For the **"Institutional Service Management Platform"** in Salesforce, a **Lightning App** will provide a user-friendly interface to manage various aspects of the platform, such as service requests, ticket management, and reporting. Below are the detailed components that can be used in the Lightning App for this project.

Lightning App Name: Institutional Service Management

App Description:

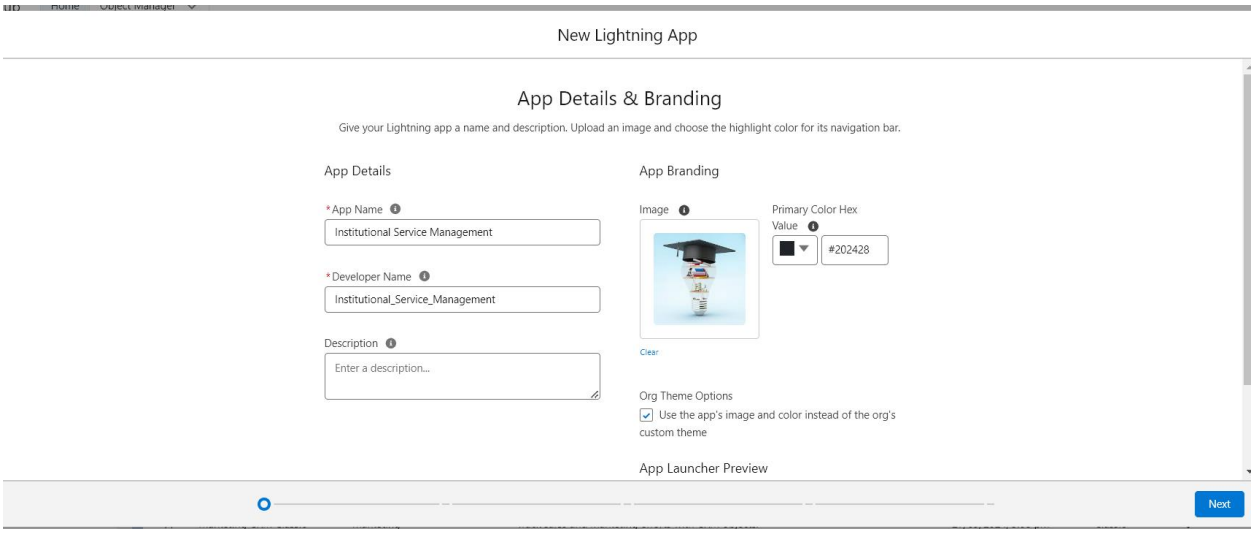
A Lightning App designed to manage institutional service requests, automate ticket assignments, monitor service status, and generate reports on institutional operations. This app is tailored to enhance service management processes within educational institutions or large organizations.

Lightning App Name: Institutional Service Management

App Components:

1. App Label:

Institutional Service Management



2. App Visibility:

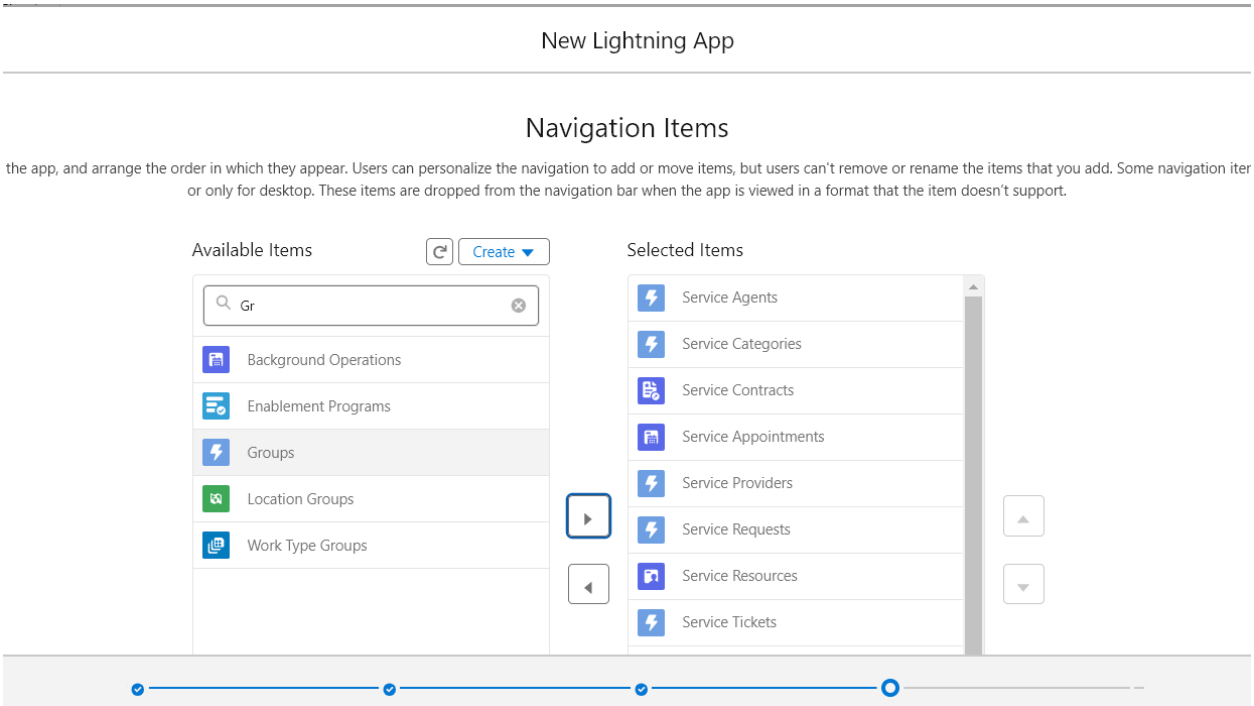
Available for: **System Administrators**, **Service Managers**, **Service Agents**, and **End Users** (based on profiles and permissions).

Navigation Style: **Standard Navigation** (for desktop use) and **Console Navigation** (for agents and service managers who need to work on multiple cases at once).

3. App Branding:

App Icon: Custom icon representing service management.

App Colors: Institutional colors, reflecting the organization's branding.



4. Standard and Custom Objects Used:

Service_Request__c: Tracks all service requests submitted by users.

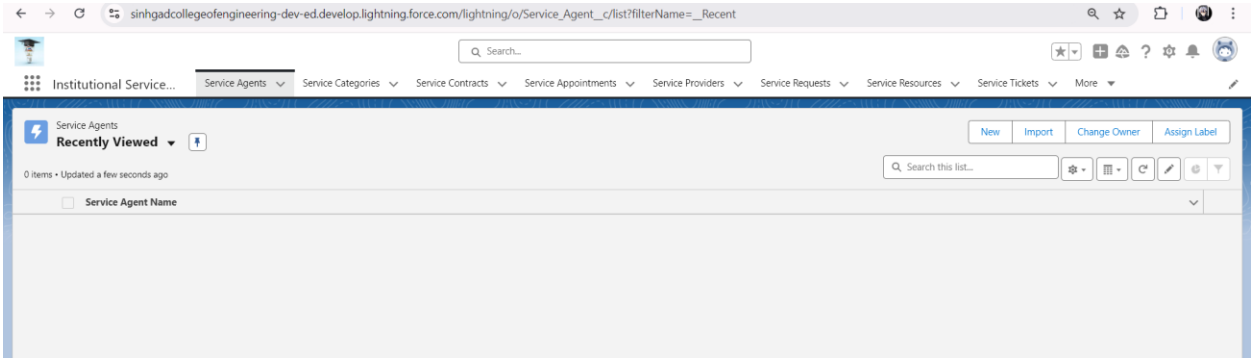
Service_Ticket__c: Manages service tickets for different departments.

Account: Represents departments or units within the institution.

Contact: Holds information about users or agents handling service requests.

Case: For tracking more complex service issues that require escalation.

Task: Assigns specific tasks to agents related to service management.



Chapter 3
Create a ScreenFlow

For the "Institutional Service Management Platform" in Salesforce, a Screen Flow can be used to guide users through submitting a service request, providing necessary information, and automating ticket creation. Here's how the Screen Flow would work in detail:

Screen Flow for Service Request Submission

3.1.Flow Overview:

This Screen Flow will allow users to create a service request by filling in details like the request type, description, priority, and any additional attachments. Once submitted, it will create a new `Service_Request__c` record, assign it to the appropriate team or queue, and notify the requester.

3.2.Flow Structure:

1. Start Element:

Flow Type: Screen Flow (Runs in user interface)

Trigger: The user manually launches the flow from a service portal or Salesforce record page.

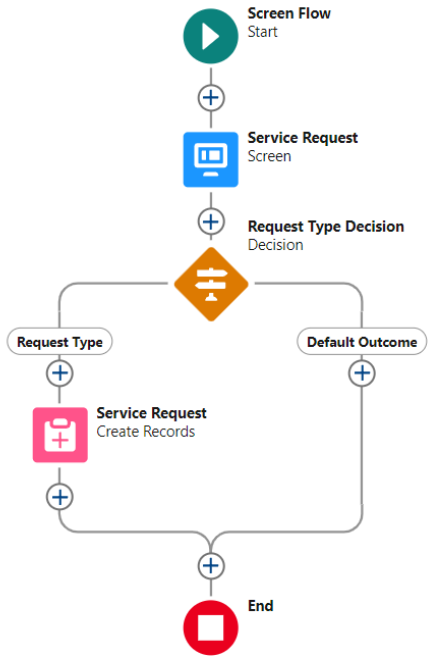
2. Screen 1: Service Request Information

Screen Components:

1. Text Field (Service Request Title): Capture a brief title for the service request.
2. Picklist (Request Type): Let the user select the type of service request. Example values:
   - IT Support
   - Maintenance
   - Administrative
   - Other
3. Picklist (Priority): Let the user set the urgency of the request. Example values:
   - Low
   - Medium
   - High

4. Text Area (Description): Capture a detailed description of the issue or request.
5. File Upload (Optional): Allow the user to upload attachments, such as images or documents related to the request.

Validation Rules: Ensure required fields like Title, Request Type, and Description are completed.



3. Decision Element: Request Type Decision

Logic: Based on the selected "Request Type" (e.g., IT Support, Maintenance, etc.), the flow will assign the request to a different queue.

Decision Branches:

1. IT Support → Assign to IT Support Team
2. Maintenance → Assign to Maintenance Team
3. Administrative → Assign to Admin Team
4. Other → Assign to General Support Team

4. Create Records Element: Create Service Request

Action: Creates a new `Service_Request__c` record in Salesforce.

Field Mappings:

- **Title** → `Service_Request_Title__c`
- **Request Type** → `Service_Request_Type__c`
- **Priority** → `Priority__c`
- **Description** → `Description__c`
- **Attachments (if uploaded) → Related List for Files**

5. Screen 2: Confirmation Message

Screen Components:

- Display Text: Thank the user for submitting the request and provide a confirmation message.
- Dynamic Text: Show the generated Service Request ID for reference.

Chapter 4
Create Users

For the "Institutional Service Management Platform" in Salesforce, user creation is an essential step to ensure that the right individuals have access to the system with appropriate roles and permissions. Below are the details for creating and managing users specific to this platform:

Create Users for the "Institutional Service Management Platform"

1. **User Roles**:

   Define different roles based on the responsibilities of users in the institution. Common roles for the platform might include:

   - **Admin**: Full access to manage the platform, create new users, oversee workflows, and run reports.
   - **Service Agent**: Responsible for handling service requests and managing tickets.
   - **Department Manager**: Oversees requests in their specific department and approves requests where necessary.
   - **End User (Requestor)**: Users who will log service requests for various needs within the institution.

2. **Profiles and Permissions**:

   Custom Profiles: Define permissions for each role. For example:

   - **Admin Profile**: Full access to all objects, fields, and system settings.
   - **Service Agent Profile**: Access to the Service Request, Service Ticket objects, and related workflows, but restricted from managing users or system settings.
   - **End User Profile**: Can create service requests and view their status but cannot access other users' requests.

Custom Object Permissions

	Basic Access				Data Administration	
	Read	Create	Edit	Delete	View All	Modify All
Approval Process	✓	✓	✓	✓	✓	✓
Groups	✓	✓	✓	✓	✓	✓
Placements	✓	✓	✓	✓	✓	✓
Request Status	✓	✓	✓	✓	✓	✓
Services	✓	✓	✓	✓	✓	✓
Service Agents	✓	✓	✓	✓	✓	✓

	Basic Access				Data Administration	
	Read	Create	Edit	Delete	View All	Modify All
Service Categories	✓	✓	✓	✓	✓	✓
Service Providers	✓	✓	✓	✓	✓	✓
Service Requests	✓	✓	✓	✓	✓	✓
Service Tickets	✓	✓	✓	✓	✓	✓
Students	✓	✓	✓	✓	✓	✓

3. **Permission Sets**:

   Use **Permission Sets** to grant additional permissions if needed, without changing the user's profile. For example:

   - Grant **Report Access** to a Service Agent for additional reporting capabilities.
   - Assign **Mobile Access** to users who need to access the platform via the Salesforce mobile app.

4. **Creating Users**:

   To create a user in Salesforce, navigate to **Setup** → **Users** → **Users** → **New User**.

   Here's the required information for creating users for the platform:

   - **Name**: Full Name of the user (e.g., John Doe).
   - **Username**: A unique email format username (e.g., john.doe@institution.com).
   - **Email**: User's email address (e.g., john.doe@institution.com).
   - **Role**: Assign the appropriate role (e.g., Admin, Service Agent, Department Manager).
   - **Profile**: Assign a profile that defines the user's access (e.g., Admin Profile, Service Agent Profile).
   - **License Type**: Choose a Salesforce license type (e.g., Salesforce Platform License).
   - **Active**: Ensure the user is marked as active.
   - **Locale Settings**: Set the user's language, time zone, and currency based on the institution's region.
   - **Mobile Settings**: Enable mobile access if the user will be managing requests on the Salesforce mobile app.

Chapter 5
Create a Approval Process

To create an Approval Process for the "Institutional Service Management Platform" in Salesforce, the process will handle the approval of various service requests made by users (e.g., maintenance, IT support, etc.). The approval process will include different stages for review and approval, based on the nature of the service request.

5.1.Step-by-Step Guide: Creating an Approval Process for Service Requests

1. Define the Approval Process Requirements

Object: `Service_Request__c` (Custom object for handling service requests)

Criteria: Approval required for any service request that meets a particular condition (e.g., requests with a cost exceeding a certain amount or those marked as "High Priority").

Approval Steps:

1. **Initial Submission**: Request is submitted by the user.
2. **Approval by Department Head**: The head of the department to which the request is assigned (e.g., IT or Maintenance) must approve the request.
3. **Final Approval**: If the request requires further escalation, the request is sent to an institutional leader for final approval.

2. Prepare Custom Fields for the Approval Process

Approval Status Field: Add a custom picklist field on `Service_Request__c` to track the status of the request:

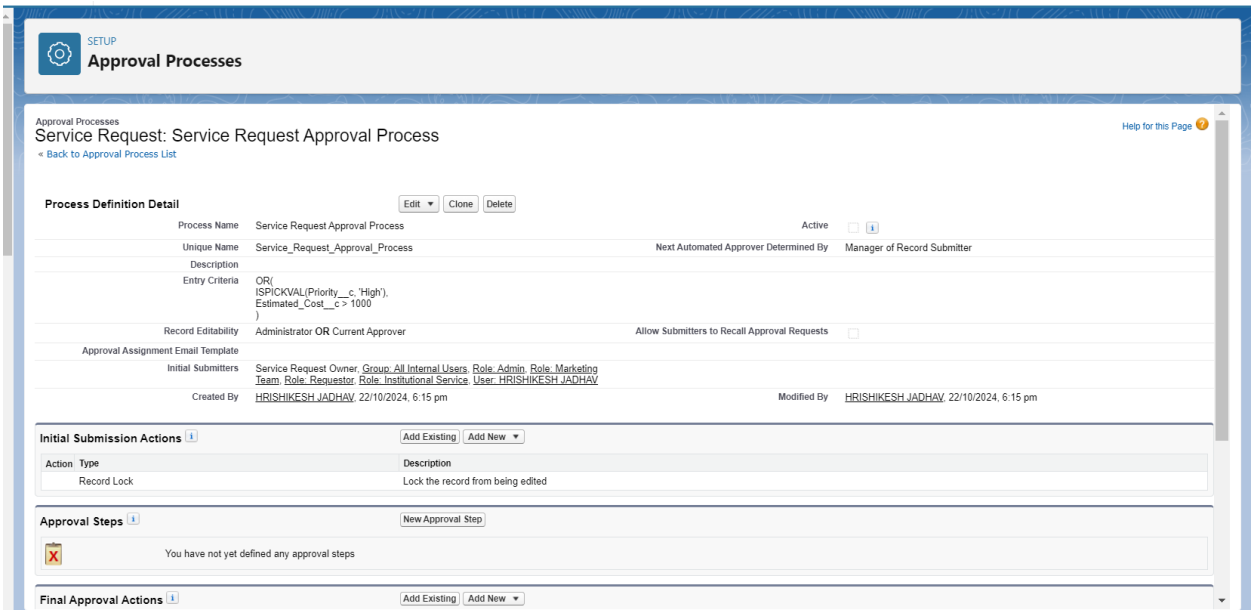
- New
- Submitted
- Approved
- Rejected

3. Create the Approval Process in Salesforce

1. **Go to Setup**:
   - Navigate to **Setup** > **Process Automation** > **Approval Processes**.
   - Select **Approval Processes** and choose **Create New Approval Process**.
2. **Choose the Object**:

- Select `Service_Request__c` as the object for which the approval process will be created.
- Choose **Standard Setup Wizard**.

3. **Enter the Name and Unique Name**:
   - **Name**: `Service Request Approval Process`
   - **Unique Name**: Automatically populated.
4. **Specify Entry Criteria**:
   - Define the criteria under which the service request will enter the approval process. For example:
     - `Service_Request__c.Priority__c = 'High'` OR
     - `Service_Request__c.Estimated_Cost__c > 1000`
5. **Select the Initial Submitter**:
   - Allow service request creators (end users) to submit requests for approval.
   - Optionally, allow administrators to submit on behalf of others.
6. **Select the Approval Field for the Record**:
   - Choose a field (e.g., `Approval_Status__c`) to track the approval status of the request.



7. **Determine the Approval Steps**:
   - **Step 1**: **Department Head Approval**:
     - Assign the request to the appropriate department head based on the request type or department field in the service request.
     - Criteria: `Service_Request__c.Department__c != NULL`
   - **Step 2**: **Institutional Leader Final Approval** (If necessary):

        - This step is required for requests exceeding a certain cost or marked as critical.
        - Criteria: `Service_Request__c.Estimated_Cost__c > 5000`
8. **Assign Approvers**:
    - **Department Head**:
        - Assign the request to a user based on a lookup field (`Service_Request__c.Department_Head__c`).
    - **Institutional Leader**:
        - Assign to a specific institutional leader or use a dynamic assignment based on the request type.
9. **Set Final Approval Actions**:
    - Update the status of the service request to `Approved` upon final approval.
    - Optionally, send an email notification to the requester and other relevant stakeholders.
    - You can also automatically assign the approved request to a service agent for resolution.
10. **Set Final Rejection Actions**:
    - Update the status of the service request to `Rejected`.
    - Notify the requester via email with a reason for the rejection.
11. **Enable Record Locking**:
    - Enable record locking during the approval process to ensure that no changes are made to the request while it is under review.
12. **Activate the Approval Process**:
    - Once all steps, actions, and criteria are configured, activate the approval process.

Chapter 6
Create a Record Triggered Flow

To implement a **Record-Triggered Flow** in Salesforce for the "Institutional Service Management Platform," we'll automate a process such as sending an email notification when a new service request is created.

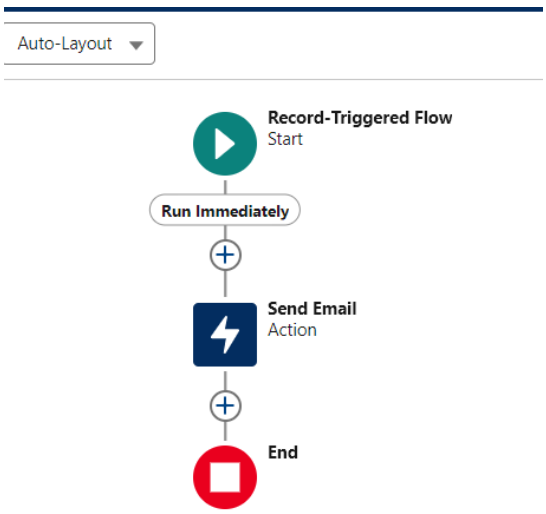
6.1.Steps to Create a Record-Triggered Flow:

1. **Navigate to Salesforce Setup**:
    - Go to **Setup** in Salesforce.
    - In the Quick Find box, type **Flows**.
    - Select **Flows** under **Process Automation**.
2. **Create a New Flow**:
    - Click **New Flow**.
    - Choose **Record-Triggered Flow** as the flow type.
    - Click **Create**.
3. **Define the Trigger**:
    - In the **Trigger** section, select **A record is created**.
    - **Object**: Select the custom object **Service_Request__c**.
    - **Condition Requirements**: Set the condition to trigger only when the **Status** field is set to "New" or any other relevant status.
        - **Field**: `Service_Request__c.Status__c`
        - **Operator**: Equals
        - **Value**: 'New' (or any other initial status you want to trigger the flow)
    - Choose **Actions and Related Records** as the optimization option.
    - Click **Done**.
4. **Add a New Action (Send Email)**:
    - Click the **+** icon to add an element after the trigger.
    - Select **Action**.
    - In the **Action** field, search for **Send Email**.
    - Configure the email:
        - **Recipient Email Address**: Set the recipient to the requestor's email. Use the field reference **{!$Record.CreatedBy.Email}** or specify a custom field like **{!$Record.Contact__r.Email}** if the requestor is associated with a Contact.
        - **Subject**: "Your Service Request has been created"
        - **Body**: Create the email body that includes key details of the service request:

          Dear {!$Record.CreatedBy.Name},

Your service request with ID {!$Record.Id} has been successfully created.
We will get back to you shortly.
Regards,
Institutional Service Management Team

- Click **Done**.



-

5. Set Entry Conditions (Optional):

- If you want the flow to only trigger under certain conditions (e.g., only for specific types of service requests), add entry conditions. For example, you can specify that the flow should run only if the **Request Type** field equals "IT Support."

6. Activate the Flow:

- Once the flow is designed and you're satisfied with the configuration, click **Save**.
- Provide a name for your flow, such as "Service_Request_Created_Notification".
- Activate the flow by clicking **Activate**.

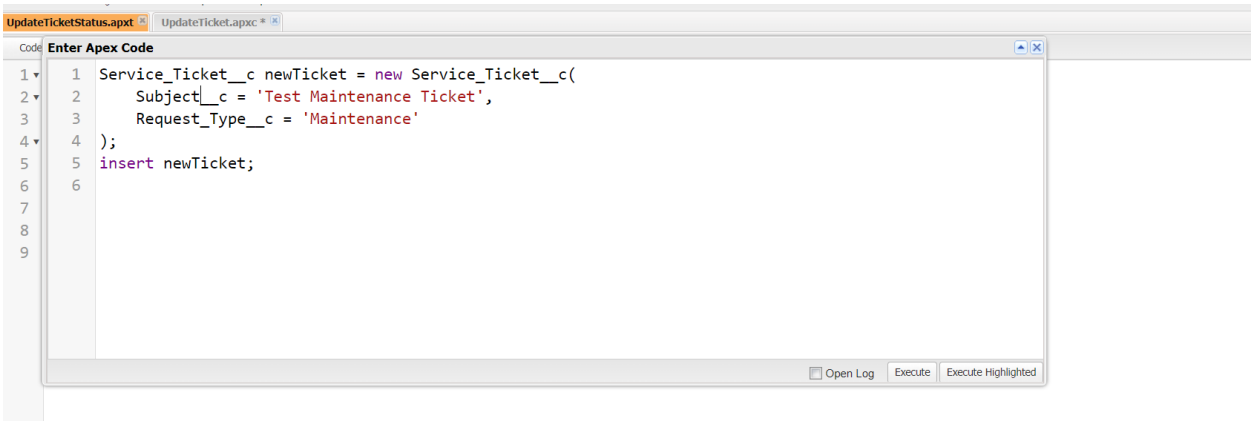
Chapter 7
Create a Trigger

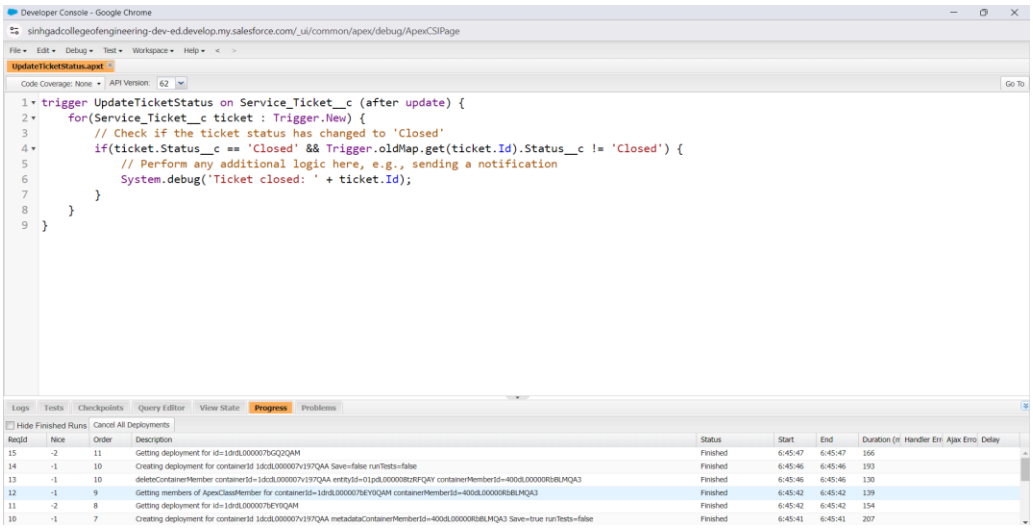
Here are the details for creating triggers related to the "Institutional Service Management Platform" in Salesforce. These triggers will help automate processes such as updating ticket statuses, sending notifications, and handling approval submissions.

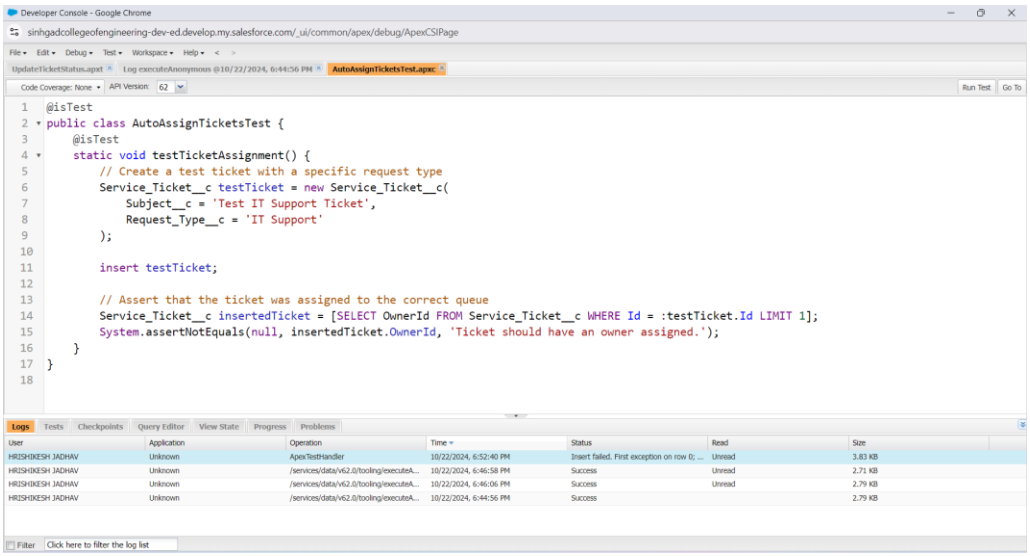
Trigger 1: Update Ticket Status on Closure

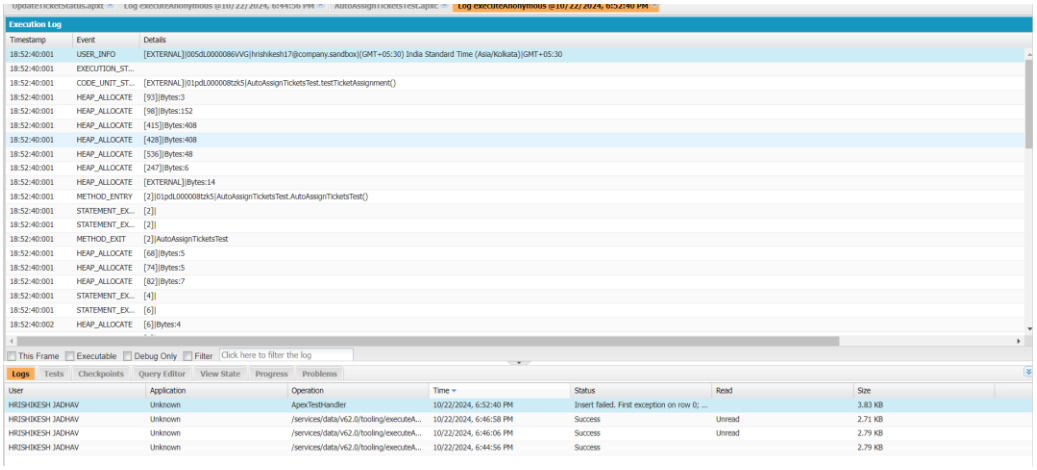
Purpose: Automatically update the status of a service ticket when it is closed.

```
trigger UpdateTicketStatus on Service_Ticket__c (after update) {

    for(Service_Ticket__c ticket : Trigger.New) {

        // Check if the ticket status has changed to 'Closed'

        if(ticket.Status__c == 'Closed' && Trigger.oldMap.get(ticket.Id).Status__c !=
'Closed') {

            // Perform any additional logic here, e.g., sending a notification

            System.debug('Ticket closed: ' + ticket.Id);

        }

    }

}
```





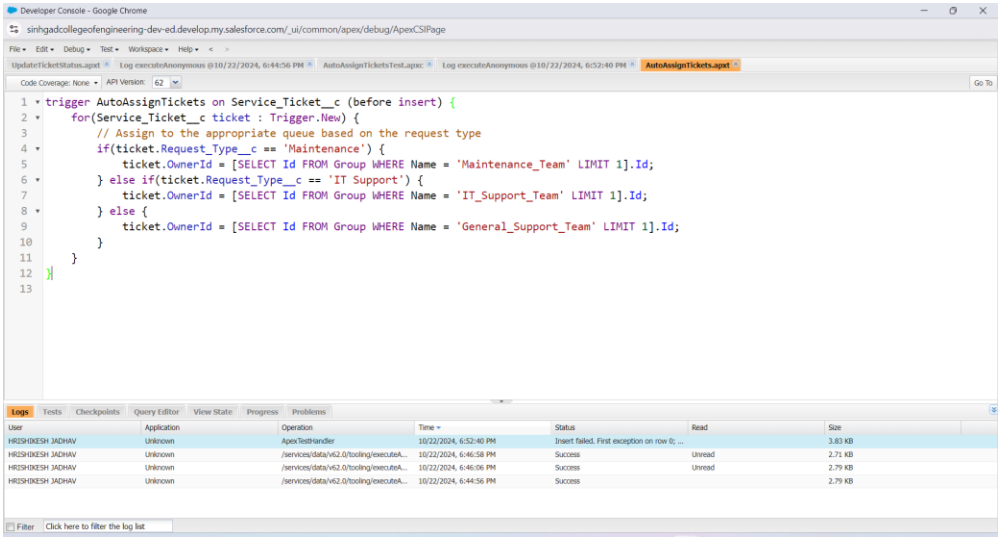




Trigger 2: Auto-assign Tickets Based on Request Type

Purpose: Assign the service ticket to a specific queue or user based on the request type when a new ticket is created.

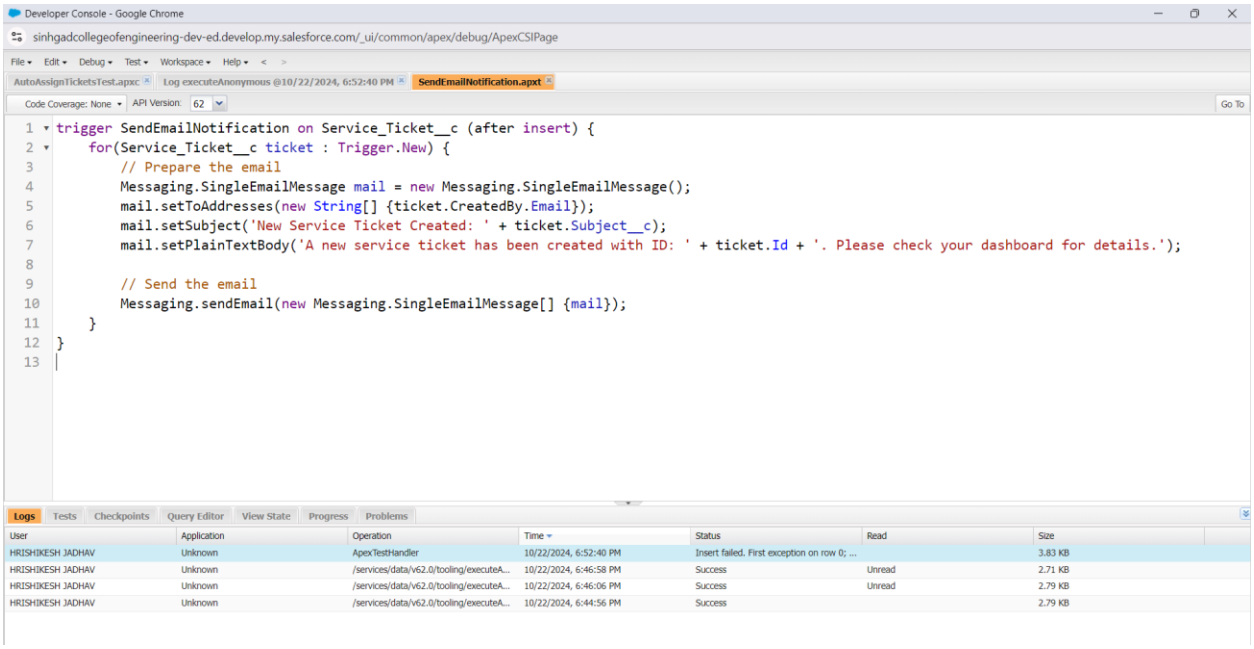
```
trigger AutoAssignTickets on Service_Ticket__c (before insert) {

    for(Service_Ticket__c ticket : Trigger.New) {

        // Assign to the appropriate queue based on the request type

        if(ticket.Request_Type__c == 'Maintenance') {

            ticket.OwnerId = [SELECT Id FROM Group WHERE Name =
'Maintenance_Team' LIMIT 1].Id;

        } else if(ticket.Request_Type__c == 'IT Support') {

            ticket.OwnerId = [SELECT Id FROM Group WHERE Name =
'IT_Support_Team' LIMIT 1].Id;

        } else {

            ticket.OwnerId = [SELECT Id FROM Group WHERE Name =
'General_Support_Team' LIMIT 1].Id;

        }

    }

}
```



Trigger 3: Send Notification Email When a New Ticket is Created

Purpose: Send an email notification to the creator of the service ticket when a new ticket is created.

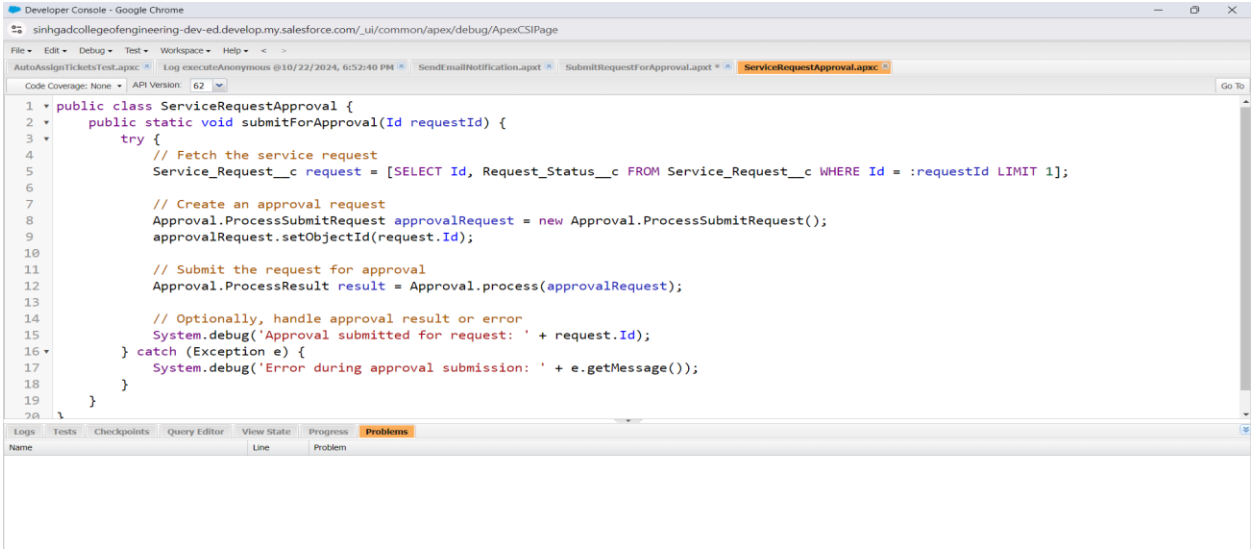
```
trigger SendEmailNotification on Service_Ticket__c (after insert) {

    for(Service_Ticket__c ticket : Trigger.New) {

        // Prepare the email

        Messaging.SingleEmailMessage mail = new
Messaging.SingleEmailMessage();

        mail.setToAddresses(new String[] {ticket.CreatedBy.Email});

        mail.setSubject('New Service Ticket Created: ' + ticket.Subject__c);

        mail.setPlainTextBody('A new service ticket has been created with
ID: ' + ticket.Id + '. Please check your dashboard for details.');



        // Send the email

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

    }

}
```

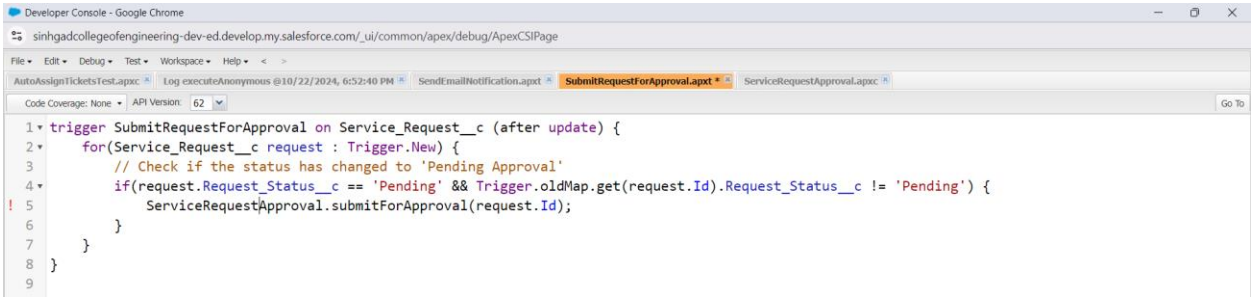


Trigger 4: Submit for Approval on Specific Status Change

Purpose: Automatically submit a service request for approval when its status changes to 'Pending Approval.'

```
trigger SubmitRequestForApproval on Service_Request__c (after update) {

    for(Service_Request__c request : Trigger.New) {

        // Check if the status has changed to 'Pending Approval'

        if(request.Status__c == 'Pending Approval' &&
Trigger.oldMap.get(request.Id).Status__c != 'Pending Approval') {

            ServiceRequestApproval.submitForApproval(request.Id);

        }

    }

}
```





Trigger 5: Log Changes to Service Requests

Purpose: Create a log entry whenever a service request is updated, to track changes made over time

```
trigger LogServiceRequestChanges on Service_Request__c (after update) {

    List<Service_Request_Log__c> logEntries = new
List<Service_Request_Log__c>();

    for(Service_Request__c request : Trigger.New) {

        Service_Request__c oldRequest = Trigger.oldMap.get(request.Id);

        // Log changes if there are differences in key fields

        if(request.Status__c != oldRequest.Status__c || request.Priority__c !=
oldRequest.Priority__c) {

            Service_Request_Log__c logEntry = new Service_Request_Log__c(

                Request__c = request.Id,

                Old_Status__c = oldRequest.Status__c,

                New_Status__c = request.Status__c,

                Old_Priority__c = oldRequest.Priority__c,

                New_Priority__c = request.Priority__c

            );

            logEntries.add(logEntry);

        }

    }

    if(!logEntries.isEmpty()) {

        insert logEntries;

    }

}
```

Developer Console - Google Chrome

sinhgadcollegeofengineering-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

File Edit Debug Test Workspace Help

AutoAssignTicketsTest.apxc | Log executeAnonymous @10/22/2024, 6:52:40 PM | SendEmailNotification.apxt | SubmitRequestForApproval.apxt * | ServiceRequestApproval.apxc | LogServiceRequestChanges.apxt

Code Coverage: None API Version: 62 Go To

```
trigger LogServiceRequestChanges on Service_Request__c (after update) {
    List<Service_Request_Log__c> logEntries = new List<Service_Request_Log__c>();

    for(Service_Request__c request : Trigger.New) {
        Service_Request__c oldRequest = Trigger.oldMap.get(request.Id);
        // Log changes if there are differences in key fields
        if(request.Request_Status__c != oldRequest.Request_Status__c || request.Priority__c != oldRequest.Priority__c) {
            Service_Request_Log__c logEntry = new Service_Request_Log__c(
                Request__c = request.Id,
                Old_Status__c = oldRequest.Request_Status__c,
                New_Status__c = request.Request_Status__c,
                Old_Priority__c = oldRequest.Priority__c,
                New_Priority__c = request.Priority__c
            );
            logEntries.add(logEntry);
        }
    }

```

Logs Tests Checkpoints Query Editor View State Progress Problems

Name	Line	Problem

Developer Console - Google Chrome

sinhgadcollegeofengineering-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

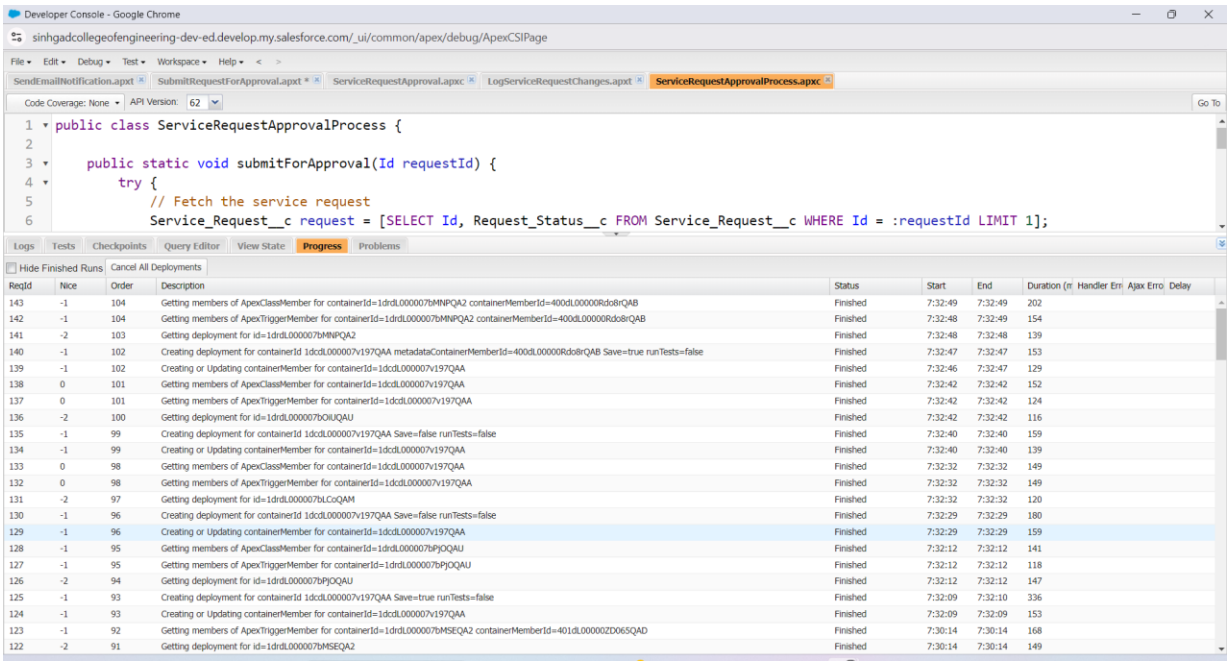
File Edit Debug Test Workspace Help

AssignTicketsTest.apxc | Log executeAnonymous @10/22/2024, 6:52:40 PM | SendEmailNotification.apxt | SubmitRequestForApproval.apxt * | ServiceRequestApproval.apxc | LogServiceRequestChanges.apxt | ServiceRequestApprovalProcess.apxc

Code Coverage: None API Version: 62 Go To

```
public class ServiceRequestApprovalProcess {

    public static void submitForApproval(Id requestId) {
        try {
            // Fetch the service request
            Service_Request__c request = [SELECT Id, Request_Status__c FROM Service_Request__c WHERE Id = :requestId LIMIT 1];

            // Check if the request is not already submitted
            if(request.Request_Status__c != 'Submitted') {
                // Create an approval process instance
                Approval.ProcessSubmitRequest req = new Approval.ProcessSubmitRequest();
                req.setObjectId(request.Id);

                // Submit the request for approval
                Approval.ProcessResult result = Approval.process(req);

                // Update the service request status to 'Submitted'
                request.Request_Status__c = 'Submitted';
                update request;

                System.debug('Service request submitted for approval');
            }
        } catch (Exception e) {
            System.debug('Error during approval process: ' + e.getMessage());
        }
    }
```

Logs Tests Checkpoints Query Editor View State Progress Problems



Summary

These triggers will help automate various processes within your Institutional Service Management Platform by ensuring that tickets are managed efficiently, users are notified of important updates, and approval workflows are handled seamlessly. You can customize the logic further based on specific requirements for your platform.

Conclusion

The "Institutional Service Management Platform" has been designed and developed to streamline and enhance the management of services within an educational or institutional environment. Leveraging Salesforce’s powerful CRM capabilities, the platform provides a comprehensive solution for handling service requests, automating processes, and improving operational efficiency.

Through the implementation of various features, including service request management, a ticketing system, approval processes, dashboards, and automated notifications, the platform addresses the needs of both service providers and users. By employing triggers for automatic ticket assignments, status updates, email notifications, and logging changes, we have ensured that the platform operates smoothly and responsively to user actions.

The custom objects created for service requests and tickets facilitate structured data management, while the integrated reporting and dashboard functionalities allow stakeholders to gain valuable insights into service operations, track performance metrics, and make informed decisions.

The use of Apex classes and triggers not only automates repetitive tasks but also enhances the user experience by providing timely information and reducing the likelihood of errors. The mobile accessibility ensures that users can manage service requests efficiently, regardless of their location.

Overall, the "Institutional Service Management Platform" embodies a robust solution that promotes collaboration, accountability, and responsiveness, ultimately leading to improved service delivery within the institution. The platform is positioned to adapt to evolving needs, ensuring continued relevance and effectiveness in managing institutional services.