

Project Title: *Institute Management System (IMS)*

(*CodeSphere Institute*)

- **Industry:** Education
 - **Project Type:** B2C Salesforce CRM Implementation
 - **Target Users:** Institute Administrators, Admission Officers, Faculty Members, Students, and Parents
-

Problem Statement

An educational institute receives hundreds of student inquiries via its website, offline walk-ins, and social media. However:

- Lead follow-up is slow and inconsistent
- Student records are tracked manually in spreadsheets
- Faculty allocation is done without balancing workloads
- Fee collection and tracking are fragmented
- Management lacks real-time dashboards for admissions, revenue, and faculty performance

To address these challenges, the institute wants to implement a **Salesforce CRM (IMS)** to:

- Automate lead capture and admission qualification
 - Manage student profiles, faculty details, courses, and batches
 - Track fee payments and send automated reminders
 - Provide real-time dashboards for admissions, faculty utilization, and revenue
 - Improve student/parent communication through SMS/Email alerts
-

Use Cases

Lead Management

- Automatically capture leads from website forms, social channels, and walk-ins
- Assign leads to admission officers based on course/territory
- Qualify leads using interest scores

Student Management

- Maintain student profiles with academic history, enrolled courses, and status.
- Convert qualified leads into student records.

Faculty Management

- Maintain faculty profiles with specialization and availability
- Assign faculty to courses/batches and track workload.

Course & Batch Scheduling

- Schedule lectures, exams, and institute events
- Send SMS/Email reminders to students and faculty

Fee Management

- Record payments, track pending dues, and generate receipts
- Send automated reminders for pending fees

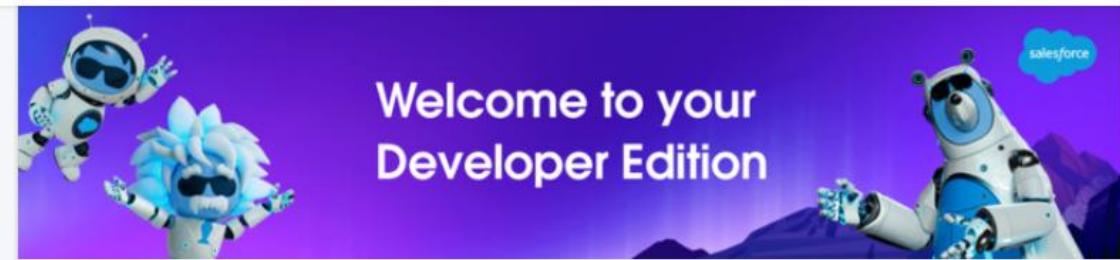
Reporting

- Dashboards for admissions funnel, monthly revenue, and faculty utilization
- Reports for student trends, counselor performance, and fee collection

Phase 2: Org Setup & Configuration

Step 1: Setup Developer Org

1. Go to <https://developer.salesforce.com/signup>
2. Create a free Developer Edition org.
3. Verify your email and login.



Hi Narigammagari,

Thanks for signing up for a Developer Edition. Now you can start building on Salesforce for free and get hands-on with Agentforce and Data Cloud.

There's just one more step. Use the following link to reset the password for your Developer Edition. This link expires in 24 hours.

[Reset Password](#)

To easily log in later, save this URL:

<https://orgfarm-e7cd2615d0-dev-ed.develop.my.salesforce.com>

Here's the username for your Developer Edition:
narigammagarinandini980@agentforce.com

Fig 1: Developer Org

Step 2: Setup Company Profile

1. Navigate: Setup → Company Information.

2. Update:

- Org Name =Codesphere Institute Of Technology
- Currency=English(INR)
- Set Default Time Zone, Fiscal Year.

The screenshot shows the 'Company Information' setup page in Salesforce. At the top, there's a blue header bar with the word 'SETUP' and a small icon. Below it, the title 'Company Information' is displayed. The main content area is titled 'Organization Detail'. It contains various configuration fields such as 'Organization Name' (CodeSphere Institute Of Technology), 'Primary Contact' (OrgFarm EPIC), 'Division' (India), 'Fiscal Year Starts In' (January), 'Activate Multiple Currencies' (unchecked), 'Enable Data Translation' (unchecked), 'Newsletter' (checked), 'Admin Newsletter' (checked), 'Hide Notices About System Maintenance' (unchecked), 'Hide Notices About System Downtime' (unchecked), 'Locale Formats' (ICU), 'Phone' (empty), 'Fax' (empty), 'Default Locale' (English (United States)), 'Default Language' (English), 'Default Time Zone' (GMT-07:00 Pacific Daylight Time (America/Los_Angeles)), 'Currency Locale' (English (India) - INR), 'Used Data Space' (352 KB (7%) [View]), 'Used File Space' (119 KB (1%) [View]), 'API Requests, Last 24 Hours' (0 (15,000 max)), 'Streaming API Events, Last 24 Hours' (0 (10,000 max)), 'Restricted Logins, Current Month' (0 (0 max)), 'Salesforce.com Organization ID' (00DgL00000BY8ph), 'Organization Edition' (Developer Edition), and 'Instance' (CAN98). At the bottom, it shows 'Created By' (OrgFarm EPIC, 9/12/2025, 10:59 PM) and 'Modified By' (Maktumsabgari Juha Fathima, 9/18/2025, 10:15 AM).

- Fig 2: Company profile

3. Configure Business Hours:

- Path: Setup → Business Hours.
- Example: Mon–Fri, 9AM – 4 PM.

4. Configure Holidays:

- Example: 26 Jan, 15 Aug (exam blackout dates).

Step 3: Create Custom Profiles

1. Navigate: Setup → Profiles.

2. Clone Standard Profiles into:

- Faculty Profile → for HOD, Teachers.
- Accounts Profile → for Accountant.
- Student Profile → for Students
- Faculty → Manage Courses, Student.
- Accounts → Manage Fee Payments, Reports.
- Students → Read-only access on their own record.

The screenshot shows the Salesforce Setup interface under the 'Profiles' section. A new profile named 'Faculty Profile' is being created. The profile is set to 'Custom Profile' and is associated with the 'Salesforce' user license. The 'Description' field is empty. In the 'Custom App Settings' section, several apps are listed with checkboxes for 'Visible' and 'Default'. Most apps have their 'Visible' checkbox checked, while 'Automation (standard FlowsApp)' and 'Sales Cloud Mobile' have it unchecked. The 'Default' column contains radio buttons, with most being empty and one checked for 'Sales (standard_Sales)'.

App	Visible	Default
All Tabs (standard__AllTabSet)	<input checked="" type="checkbox"/>	<input type="radio"/>
Analytics Studio (standard__Insights)	<input checked="" type="checkbox"/>	<input type="radio"/>
App Launcher (standard__AppLauncher)	<input checked="" type="checkbox"/>	<input type="radio"/>
Approvals (standard__Approvals)	<input checked="" type="checkbox"/>	<input type="radio"/>
Automation (standard FlowsApp)	<input type="checkbox"/>	<input type="radio"/>
My Service Journey (standard__MSJapp)	<input checked="" type="checkbox"/>	<input type="radio"/>
Queue Management (standard__QueueManagement)	<input checked="" type="checkbox"/>	<input type="radio"/>
Sales (standard__LightningSales)	<input checked="" type="checkbox"/>	<input type="radio"/>
Sales (standard__Sales)	<input checked="" type="checkbox"/>	<input checked="" type="radio"/>
Sales Cloud Mobile	<input type="checkbox"/>	<input type="radio"/>

Fig 3: Creation of profiles

Step 4: Define Roles

1. Navigate: Setup → Roles → Set Up Roles → Expand All.

2. Create hierarchy:

- Principal (Top level).
- HOD (reports to Principal).
- Teacher (reports to HOD).
- Accountant (reports to Principal).
- Student (lowest level).

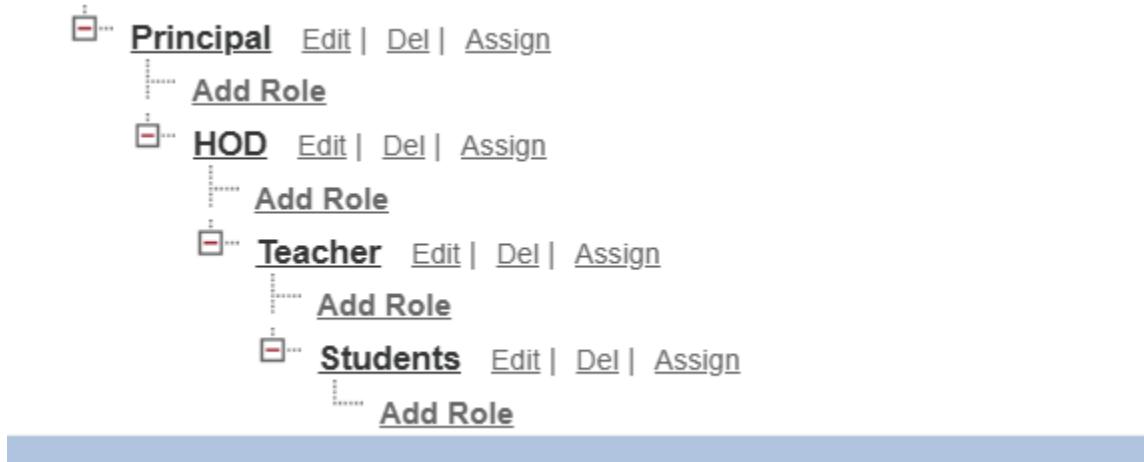


Fig 4: Hierarchy of Roles

Step 5: Create Users

1. Navigate: Setup → Users → New User.
2. Assign details:
 - User License = Salesforce (for staff).
 - Profile = Faculty, Accounts, System Admin, etc.
 - Role = Principal, Teacher, Accountant, Student.
3. Create sample test users: Principal, Teacher.

Action	Full Name	Alias	Username	Role	Active	Profile
<input type="checkbox"/>	Begum_shahida	sbegu	shahid@email.com	Teacher	<input checked="" type="checkbox"/>	Faculty Profile
<input type="checkbox"/>	Chatter_Expert	Chatter	chatty.00dg00000by8phuad.4k2d3kv4b0xa@chatter.salesforce.com		<input checked="" type="checkbox"/>	Chatter Free User
<input type="checkbox"/>	EPIC_OrgFarm	QEPIC	epic.67bb17f44a4@orgfarm.salesforce.com		<input checked="" type="checkbox"/>	System Administrator
<input type="checkbox"/>	Juha_Fathima_Maktumsabgari	zuh	zuhan8899720@agentforce.com		<input checked="" type="checkbox"/>	System Administrator
<input type="checkbox"/>	Maktumsabgari_Mahaboob_Basha	mmB	basha99@blurgmail.com	Principal	<input checked="" type="checkbox"/>	System Administrator
<input type="checkbox"/>	User_Integration	integ	integration@00dg00000by8phuad.com		<input checked="" type="checkbox"/>	Analytics Cloud Integration User
<input type="checkbox"/>	User_Security	sec	insightsecurity@00dg00000by8phuad.com		<input checked="" type="checkbox"/>	Analytics Cloud Security User

Fig 5: Users List

Step 6: Org-Wide Defaults (OWD) & Sharing

1. Navigate: Setup → Sharing Settings.

2. Set Org-Wide Defaults (OWD):

- Student Object = Private.
- Course/Batch = Public Read Only.

With this setup, my Salesforce Educational Org will have:

- Proper institute branding & working hours.
- Clear role hierarchy (Principal → HOD → Teacher → Student).
- Secure sharing model (Students private, Courses public read).
- Profiles tailored for Faculty, Accounts, and Students.

The screenshot shows the 'Sharing Settings' page in the Salesforce Setup. The top navigation bar includes 'SETUP' and the 'Sharing Settings' icon. Below the header, the page title is 'Sharing Settings'. A sub-header 'Sharing Settings' is displayed. A note states: 'This page displays your organization's sharing settings. These settings specify the level of access your users have to each others' data. Go to [Background Jobs](#) to monitor the progress of a change to an organization-wide default or a parallel sharing recalculation.' A dropdown menu 'Manage sharing settings for:' is set to 'Course'. A link 'Disable External Sharing Model' is present. The main section is titled 'Default Sharing Settings' and contains 'Organization-Wide Defaults'. It shows a table:

Object	Default Internal Access	Default External Access	Grant Access Using Hierarchies
Course	Public Read Only	Private	<input checked="" type="checkbox"/>

Below this is the 'Other Settings' section, which includes checkboxes for 'Manager Groups', 'Secure guest user record access' (which is checked), and 'Require permission to view record names in lookup fields'.

Fig 6: The OWD settings

Phase 3

Salesforce Data Model Implementation

Step 1: Create Custom Objects

Navigate to Setup → Object Manager → Create → Custom Object.

Create the following custom objects one by one:

- **Student**
 - Record Name: Student ID (Auto-Number)
 - Enable: Reports & Activities
- **Faculty**
 - Record Name: Faculty ID (Auto-Number)
 - Enable: Reports & Activities
-

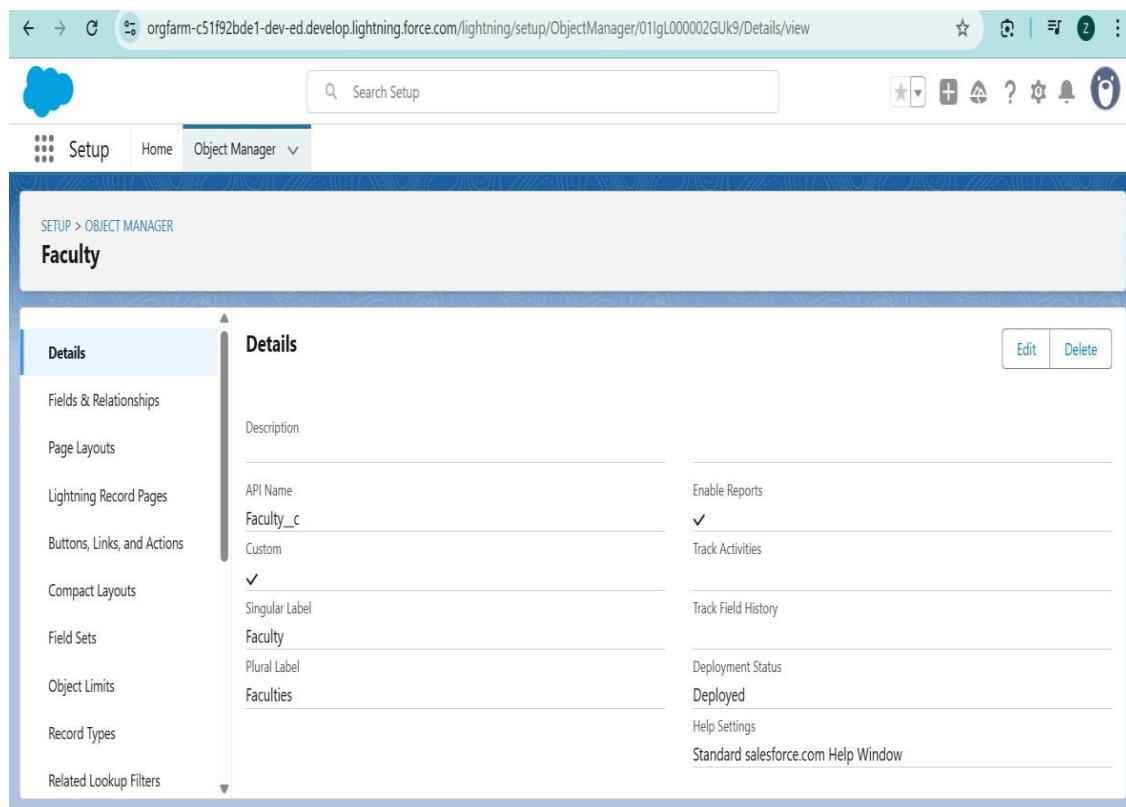


Fig1: Faculty Custom Object

- **Course**
 - Record Name: Course ID (Auto-Number or Text)
 - Enable: Reports & Activities
- **Enrollment**
 - Record Name: Enrollment ID (Auto-Number)
 - Enable: Reports & Activities
- **Fee Payment**
 - Record Name: Payment ID (Auto-Number)
 - Enable: Reports & Activities
-

The screenshot shows the Salesforce Object Manager interface. At the top, there's a navigation bar with a search bar labeled 'Search Setup', followed by icons for Home, Object Manager, and other setup functions. Below the navigation is a breadcrumb trail 'SETUP > OBJECT MANAGER'. The main title is 'Fee Payment'. On the left, a sidebar lists various setup categories like Fields & Relationships, Page Layouts, and Buttons, Links, and Actions. The main content area is titled 'Details' and contains fields for API Name ('Fee_Payment__c'), Custom status ('Custom'), Singular Label ('Fee Payment'), Plural Label ('Fee Payments'), and several checkboxes for reporting and tracking. At the bottom right of the details panel are 'Edit' and 'Delete' buttons.

Details	
Description	
API Name	Enable Reports
Fee_Payment__c	
Custom	Track Activities
✓	Track Field History
Singular Label	
Fee Payment	
Plural Label	Deployment Status
Fee Payments	Deployed
	Help Settings
	Standard salesforce.com Help Window

Fig 2:Fee payment custom Object

Step 2: Add Fields to Custom Objects

Student

- Email → Text
- Phone → Phone
- DOB → Date
- Enrollment Status → Picklist (Admitted, Active, Completed, Dropped)
- Course → Lookup (Course)

The screenshot shows the Salesforce Object Manager interface for the 'Student' object. The left sidebar lists various setup options like Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, etc. The main content area is titled 'Fields & Relationships' and displays a table of 11 items, sorted by Field Label. The table columns are FIELD LABEL, FIELD NAME, DATA TYPE, CONTROLLING FIELD, and INDEXED. The fields listed are:

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Admission Date	Admission_Date_c	Date		
Course	Course_c	Lookup(Course)	✓	
Created By	CreatedBy	Lookup(User)		
DOB	DOB_c	Date		
Email	Email_c	Email		
Last Modified By	LastModifiedBy	Lookup(User)		

Fig 3:Fields of Student object

Faculty

- Email → Email
- Phone → Phone
- Department → Picklist (CSE,ECE,EEE,CE,MEC,AIML,AIDS)
- Faculty ID → Auto-Number

Course

- Course Name → Text
- Course Code → Text
- Duration → Number (Months)
- Credits → Number
- Fees → Currency
- Faculty → Lookup (Faculty)

Enrollment

- Student → Lookup (Student)
- Course → Lookup (Course)

- Enrollment Date → Date
- Status → Picklist (Active, Completed, Dropped)
- Grade → Picklist (A, B, C, D, F)

The screenshot shows the Salesforce Object Manager interface for creating a new custom field. The top navigation bar includes 'SETUP' and 'Object Manager'. The page title is 'Edit Enrollment Custom Field Status'. A progress bar at the top right indicates 'Step 2 of 3'. The main content area is titled 'Step 2. Enter the details'. It shows a table of existing picklist values: Active, Completed, and Dropped, each with a 'Modified By' timestamp of 'Maksumsabgari Juhu Fatima, 9/17/2025, 6:28 AM'. Below the table, there are input fields for 'Field Name' (Status), 'Description' (empty), and 'Help Text' (empty). A note at the bottom states 'Required' with an unchecked checkbox and the text 'Always require a value in this field in order to save a record'.

Action	Values	API Name	Default	Chart Colors	Modified By
<input type="checkbox"/> Edit Del	Active	Active	<input type="checkbox"/>	Assigned dynamically	Maksumsabgari Juhu Fatima, 9/17/2025, 6:28 AM
<input type="checkbox"/> Edit Del	Completed	Completed	<input type="checkbox"/>	Assigned dynamically	Maksumsabgari Juhu Fatima, 9/17/2025, 6:28 AM
<input type="checkbox"/> Edit Del	Dropped	Dropped	<input type="checkbox"/>	Assigned dynamically	Maksumsabgari Juhu Fatima, 9/17/2025, 6:28 AM

Field Label: i

Description:

Help Text:

Required Always require a value in this field in order to save a record

Fig 4: Picklist of Status Field

Fee Payment

- Student → Lookup (Student)
- Payment Date → Date
- Amount → Currency
- Mode of Payment → Picklist (Cash, Card, Online)
- Status → Picklist (Paid, Pending)

Fields & Relationships					
	FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Page Layouts	Amount	Amount__c	Currency(18, 0)		
Lightning Record Pages	Created By	CreatedById	Lookup(User)		
Buttons, Links, and Actions	Fee Payment Name	Name	Text(80)		
Compact Layouts	Last Modified By	LastModifiedById	Lookup(User)		
Field Sets	Mode of Payment	Mode_of_Payment__c	Picklist		
Object Limits	Owner	OwnerId	Lookup(User,Group)		
Record Types					
Related Lookup Filters					

Fig 5: Fields of Fee payment

Step 3: Setup Relationships:

Junction Object → Enrollment:

The Enrollment object enables a many-to-many relationship between Students and Courses.

- Enrollment → Student (Lookup)
- Enrollment → Course (Lookup)

Fields & Relationships					
	FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Page Layouts	Course	Course__c	Lookup(Course)		
Lightning Record Pages	Created By	CreatedById	Lookup(User)		
Buttons, Links, and Actions	Enrollment Date	Enrollment_Date__c	Date		
Compact Layouts	Enrollment Name	Name	Text(80)		
Field Sets	Grade	Grade__c	Picklist		
Object Limits	Last Modified By	LastModifiedById	Lookup(User)		
Record Types					
Related Lookup Filters					

Fig 6: Junction Object

- Course → Faculty (Lookup)
- Fee Payment → Student (Lookup)

This creates the following structure:

- One Student → many Courses (via Enrollment)
- One Course → many Students (via Enrollment)
- One Student → multiple Fee Payments
- One Faculty → many Courses

Step 4: Verify Schema with Schema Builder

1. Go to Setup → Schema Builder
2. Drag & drop objects: Student, Faculty, Course, Enrollment, Fee Payment
3. Confirm relationships visually.

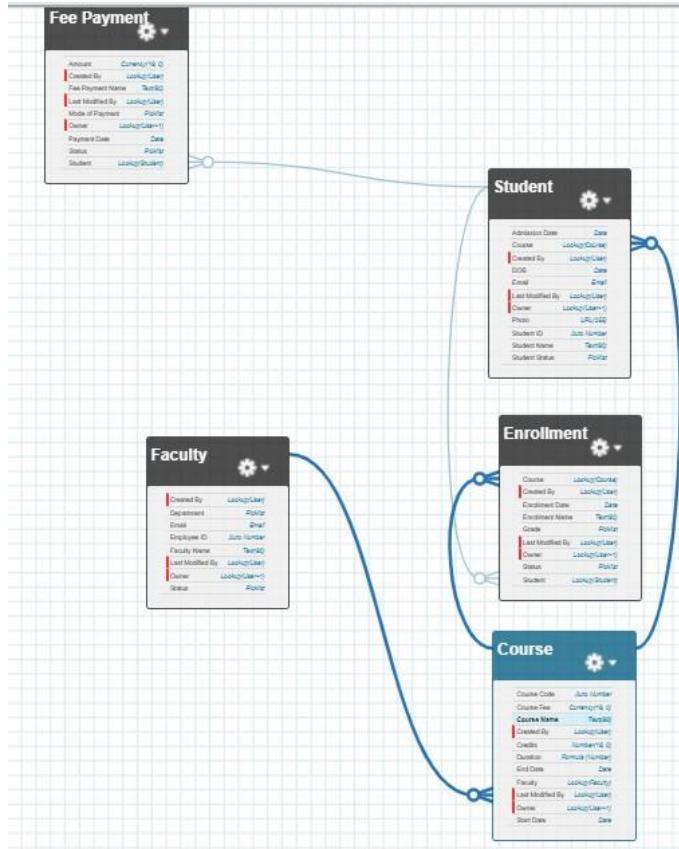


Fig 7 Schema map

Step 5: Test with Sample Records

- Create 1–2 Students.
- Create a Faculty record.
- Create a Course and assign to Faculty.
- Enroll Student in Course (via Enrollment object).

- Add a Fee Payment record for Student.
- Verify relationships in Related Lists of each record.

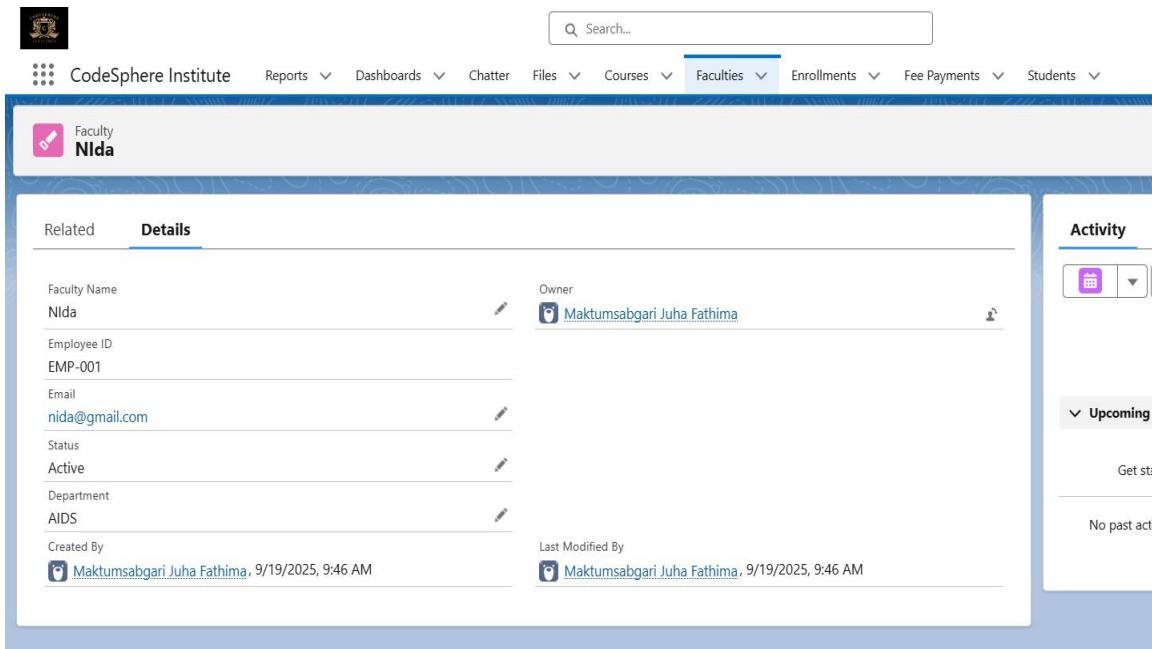


Fig 8: Verification

Phase 4

Business Logic & Automation

Step 1: Creating Tabs for Objects

In this step, I created **custom objects** in Salesforce to represent the key entities of the Codesphere Institute Of Technology. For each object, I also created a **tab** so that users can access and manage records easily from the Salesforce app navigation bar.

- Navigated to **Setup → Object Manager**.
- Created a **new custom object** for each entity (Student, Faculty, Course, Fee Payment, Enrollment).
- For every object, created a **tab** via:

- o Setup → Tabs → New Tab
- o Selected the object and an icon, then saved.

The screenshot shows the Salesforce Setup interface under the 'Tabs' section. At the top, there's a header with a gear icon labeled 'SETUP' and 'Tabs'. Below the header, the title 'Custom Tabs' is displayed. A descriptive text follows: 'You can create new custom tabs to extend Salesforce functionality or to build new application functionality. Custom Object tabs look and behave like the standard tabs provided with Salesforce. Web tabs allow you to embed external Visualforce pages. Lightning Component tabs allow you to add Lightning components to the navigation menu in Lightning Experience and the mobile app.'

Custom Object Tabs

Action	Label	Tab Style
Edit Del	Courses	Chess piece
Edit Del	Enrollments	Lightning
Edit Del	Faculty	People
Edit Del	Fee Payments	Books
Edit Del	Students	Bridge

Web Tabs

No Web Tabs have been defined.

Visualforce Tabs

Fig: Creation of Tabs

Step 2: App Creation :

Step 1: Create a New App

- Go to **Setup → App Manager**.
- Click on **New Lightning App**.
- Enter **App Name** (Codesphere Institute).
- Upload an **App Logo** to personalize the app.
- Click **Next** to continue.

App Details & Branding

Give your Lightning app a name and description. Upload an image and choose the highlight color for its navigation bar.

App Details	App Branding
*App Name <small>i</small> <input type="text" value="CodeSphere Institute"/>	Image <small>i</small>  Primary Color Hex Value <small>i</small> <input type="color" value="#0070D2"/> #0070D2
*Developer Name <small>i</small> <input type="text" value="CodeSphere_Institute"/>	<input type="checkbox"/> Use the app's image and color instead of the org's custom theme
Description <small>i</small> <input type="text" value="Custom app for managing students, courses, faculty, and fees."/>	Org Theme Options <input type="checkbox"/> Use the app's image and color instead of the org's custom theme
App Launcher Preview	
	CodeSphere Institute Custom app for managing students, courses, faculty, an...

Fig 2:App creation

Step 2: Configure Navigation

- Select **Navigation Style → Standard Navigation.**
- Choose whether the app should be available on **Desktop or Desktop & Mobile.**
- Click **Next.**

Step 3: Assign User Profiles

- Select the **profiles** that should access this app.
 - Example: System Administrator, Standard User.
- Click **Save and Finish.**

Step 4: Verify the App

- Open the **App Launcher** (grid icon in top-left).
- Search for your app (Codesphere Institute Of Technology).
- Open it and confirm that all required **tabs** (Students, Faculty, Courses, Fee Payment, Enrollments) are visible.
- Test by creating a sample record in each tab to ensure everything works properly.

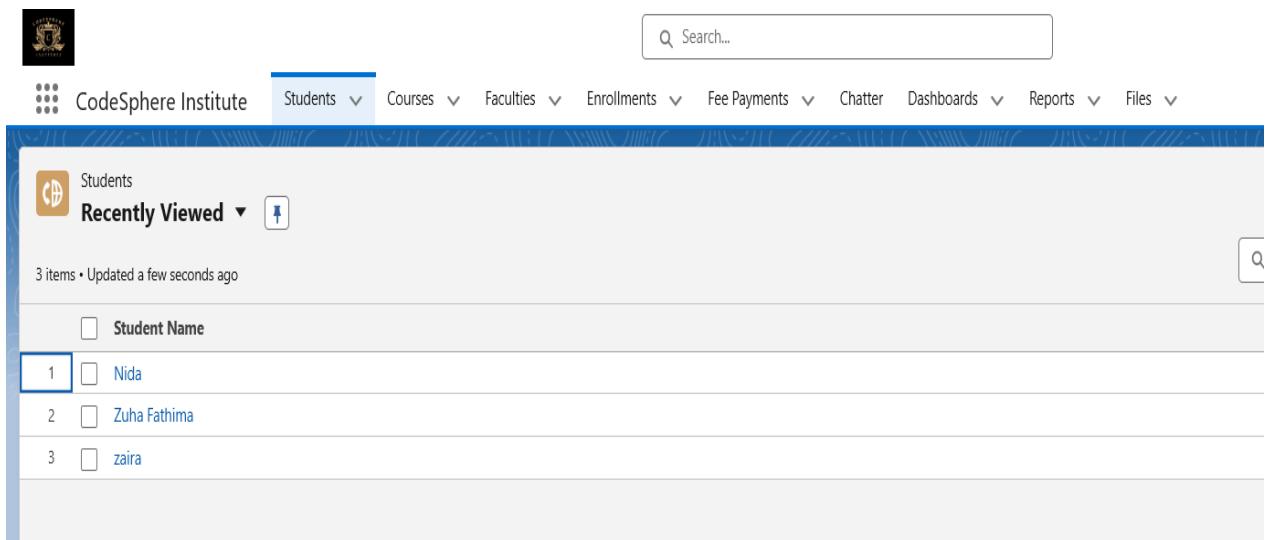


Fig 3:Codesphere app Image

- birth.

Step 3: Create Validation Rules

Validation Rules are designed to maintain data integrity by ensuring that only valid data is entered in Salesforce objects.

Student Object

- **Rule Name:** DOB_Check
- **Formula:** Date_of_Birth__c >= TODAY()
- **Error Message:** “Date of Birth must be in the past.”

The screenshot shows a Salesforce page for a 'Student' object named 'Zuhra Fathima'. The page includes fields for Student Name (Zuhra Fathima), Email (zuhaf8899@gmail.com), Admission Date (9/19/2025), Photo (empty), Student ID (STU-0001), Course (Electronics), and Student Status (Admitted). The 'DOB' field contains the value '11/14/2025'. A red validation message box at the bottom center states 'We hit a snag.' with the sub-instruction 'Review the errors on this page.' It lists one error: 'Date of Birth must be in the past.' There are 'Cancel' and 'Save' buttons at the bottom right, along with a 'By' field showing 'Maktumsabgari'.

Purpose: Prevents users from entering a future date as a student's date of

Fig 4:Checking Validation rule for DOB

Course Object

- **Rule Name:** StartDate_Check

- **Formula:**
- Start_Date__c < TODAY()
- **Error Message:** “Course Start Date cannot be in the past.”
- **Purpose:** Ensures course start dates are valid and in the future.

Fee Payment Object

- **Rule Name:** Amount_Check
- **Formula:**
- Paid_Amount__c <= 100
- **Error Message:** “Payment amount must be greater than hundred.”
- **Purpose:** Prevents entry of invalid payment amount

Fee Payment Validation Rule

[Back to Fee Payment](#)

Validation Rule Detail

[Edit](#) [Clone](#)

Rule Name	Amount_Check
Error Condition Formula	Amount__c <= 500
Error Message	Payment amount must be greater than Five hundred
Description	
Created By	Maktumsabgari Juha Fathima, 9/19/2025, 11:17 AM

[Edit](#) [Clone](#)

Fig 5: Validation rule Formula

Step 4: Flow 1 – Admission Process (Auto-create Enrollment + Fee Payment)

Purpose:

Automatically create Enrollment and Fee Payment records when a Student is admitted.

Configuration Steps:

- **Navigate to Flows:**
 - Setup → Flows → New Flow
- **Select Flow Type:**
 - Record-Triggered Flow
- **Object:**
 - Student__c
- **Trigger:**
 - When a record is **created**
- **Entry Condition:**
 - Status__c = "Admitted"
- **Add Elements:**

a. Create Enrollment Record

- Object: Enrollment__c
- Set Fields:
 1. Student__c = {!\$Record.Id}
 2. Course__c = {!\$Record.Course__c}

b. Create Fee Payment Record

- Object: Fee_Payment__c
- Set Fields:
 1. Student__c = {!\$Record.Id}
 2. Status__c = "Pending"
- **Click On Save □Label = Admission Process**
- **And Activate** the flow

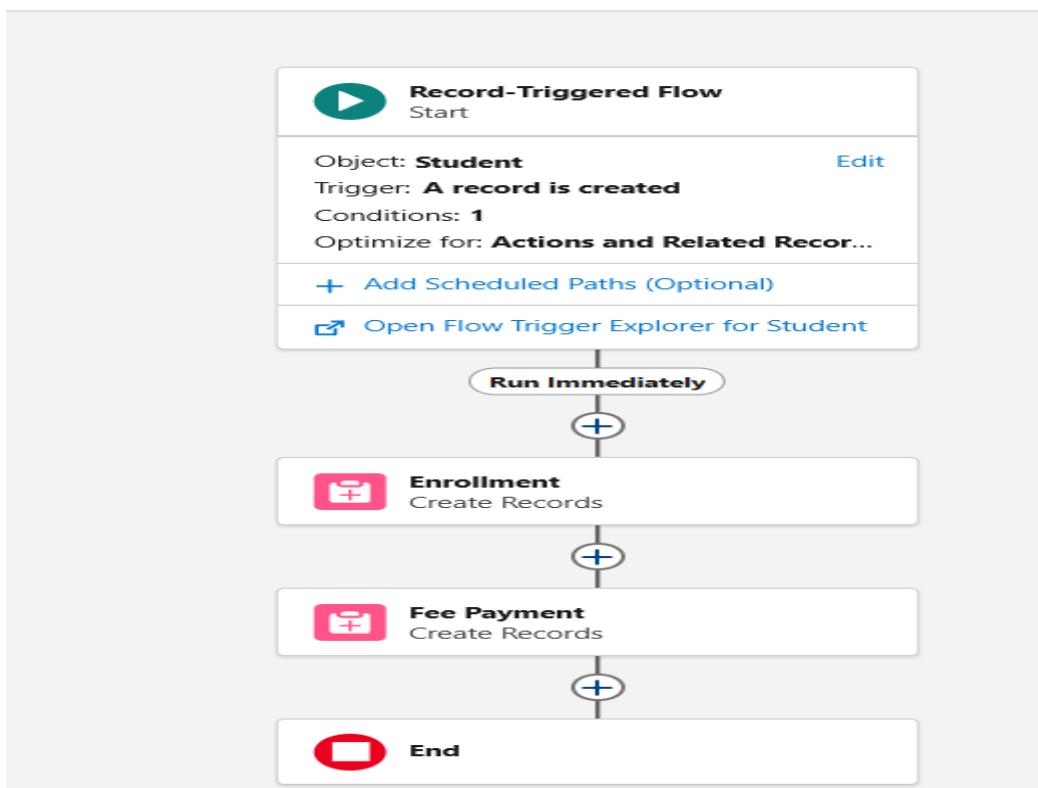


Fig 6: Flow of Admission Process

Outcome:

When a Student record is created with Enrollment_Status_c = "Admitted", corresponding Enrollment and Fee Payment records are automatically generated, reducing manual data entry and ensuring process consistency.

Section	Count	Record Details	Action
Enrollments	(1)	ENR-0105	New
Fee Payments	(1)	PAY-00005	New
Contacts	(0)		New

Fig 7:Output

Step 5: Scheduled Flow – Daily Fee Reminders

Purpose:

Send daily reminders to students for pending fee payments.

Configuration Steps:

- **Navigate to Flows:**
 - Setup → Flows → New Flow
- **Select Flow Type:**
 - Scheduled-Triggered Flow
- **Set Schedule:**
 - Frequency: Daily
 - Start Time: 4.30 PM
- **Object / Records to Process:**
 - Fee_Payment__c
 - Condition: Status__c = "Pending"
- **Add Elements:**
 - Action → **Send Email Alert**
 - Recipient: Student__c.Email
- **Save & Activate**
- **Click On Save □ Label = Fee remainder flow**
- **And Activate the flow**

Outcome:

All students with pending fees receive automated daily reminders, ensuring timely payments.

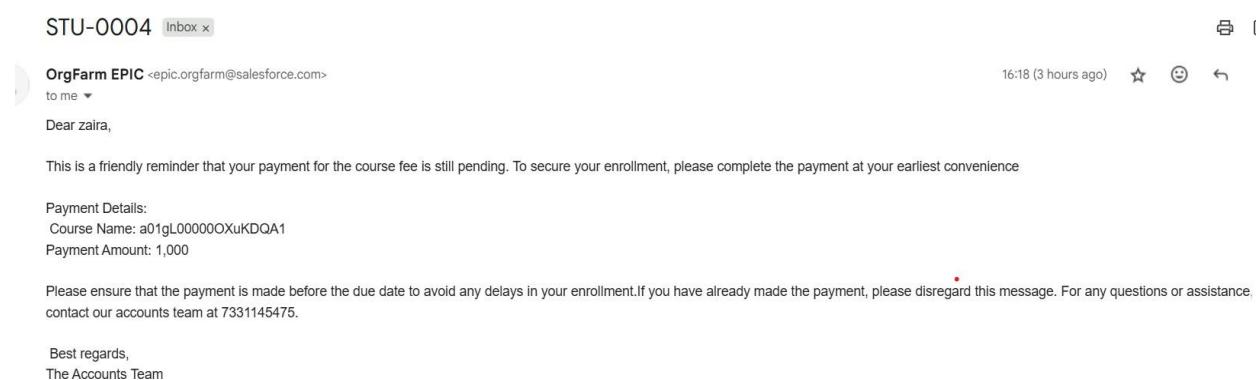


Fig 8:Email Alert

Step 6: Record-Triggered Flows

Enrollment Status Update When Grade Is Entered

Purpose: Automatically update Enrollment status to “Completed” once a grade is entered.

- **Configuration Steps:**
- Setup → Flow → New Flow → Record-Triggered Flow
- Object: Enrollment__c
- Trigger: When a record is **created or updated**
- Condition Requirements:
 - Grade__c **Is Changed**
 - Grade__c **Is Not Null**
- Optimize the Flow for: **Actions and Related Records**
- Add Element → **Update Records**
 - Update the same Enrollment record
 - Set Enrollment_Status__c = "Completed"
- **Click On Save □Label** = Status completed
- **And Activate** the flow

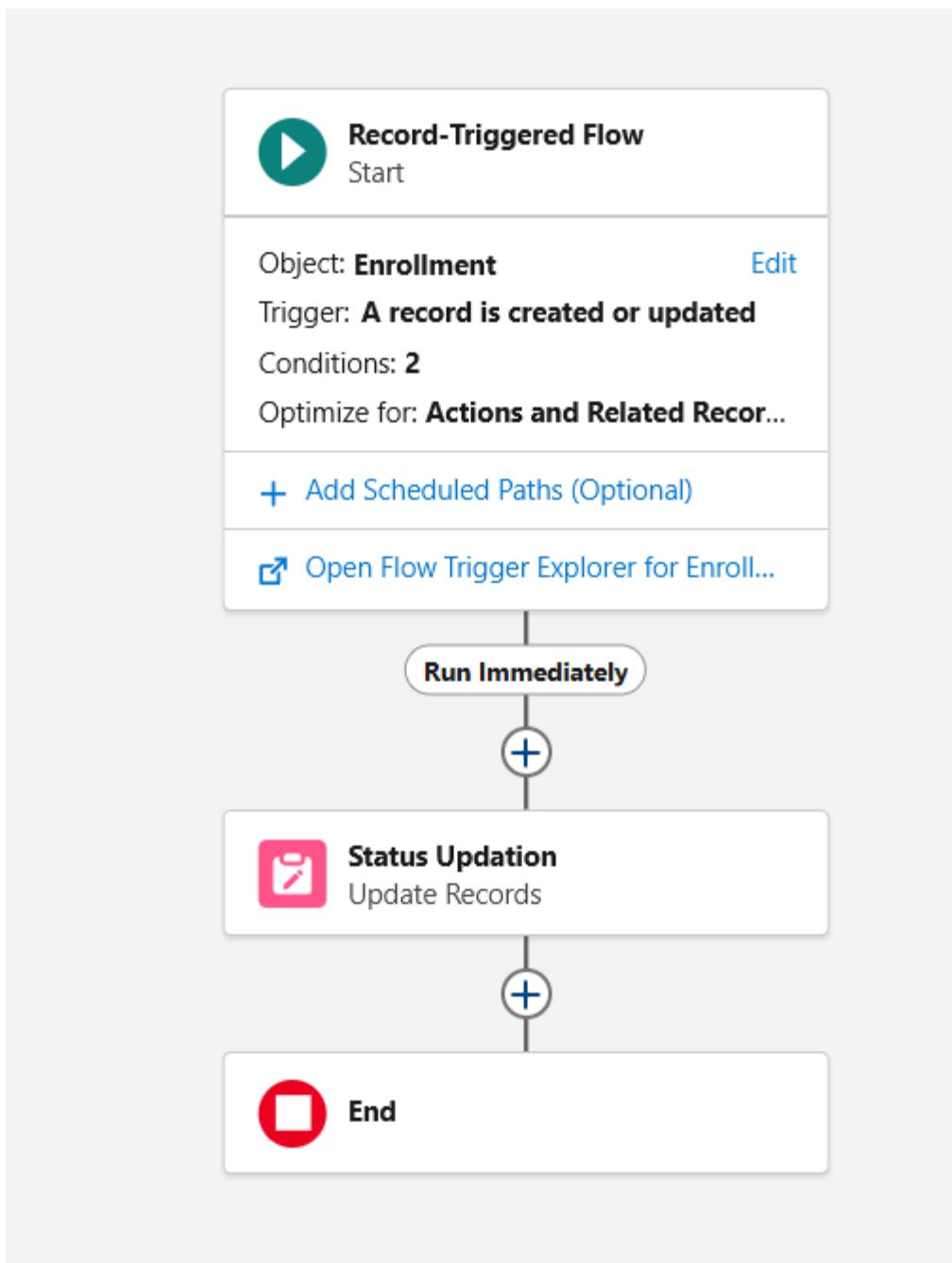


Fig 9:Flow construction

Outcome: Enrollment records are automatically updated to “Completed” when a grade is recorded.

Enrollment
ENR-0101

Related	Details
Enrollment Number	ENR-0101
Student	Zuha Fathima
Course	Electronics
Enrollment Date	9/20/2025
Status	Completed
Grade	C
Created By	Maktumsabgari Juha Fathima, 9/19/2025, 11:39 AM
Owner	Maktumsabgari Juha Fathima
Last Modified By	Maktumsabgari Juha Fathima, 9/22/2025, 4:13 AM

Fig 10:Status automatically changed

Fee Payment Status Update When Amount Received

Purpose: Automatically update Fee Payment status to “Paid” when a payment is received.

Configuration Steps:

- Setup → Flow → New Flow → Record-Triggered Flow
- Object: Fee_Payment__c
- Trigger: When a record is **created or updated**
- Condition Requirements:
 - Amount __c = 0
- Add Element → **Update Records**
 - Update the same Fee Payment record
 - Set Status__c = "Paid"
- Add **Email Alert** → Notify Student of payment received
- **Click On Save □Label = Fee status Updated**
- **And Activate** the flow

Outcome: Fee Payment records are automatically updated, and students are notified, ensuring accurate financial tracking.



Fig 11:Email for clearing of fees

Step 6: Email Alert:

Purpose:

Automatically notify students via email when their enrollment status becomes **Active**, ensuring timely communication about their successful enrollment in the institute.

Configuration Steps:

- Setup → Flow → New Flow → Record-Triggered Flow
- Object: Enrollment__c
- Trigger: When a record is **created or updated**
- Condition Requirements:
 - Status__c = "Active"
- Add **Email Alert** → Notify Student who are enrolled in institute
- **Click On Save □Label = Enroll students**
- **And Activate** the flow

Outcome:

When an enrollment record is created or updated with Status__c = "Active", the

student automatically receives an enrollment confirmation email, improving communication and reducing manual follow-ups.

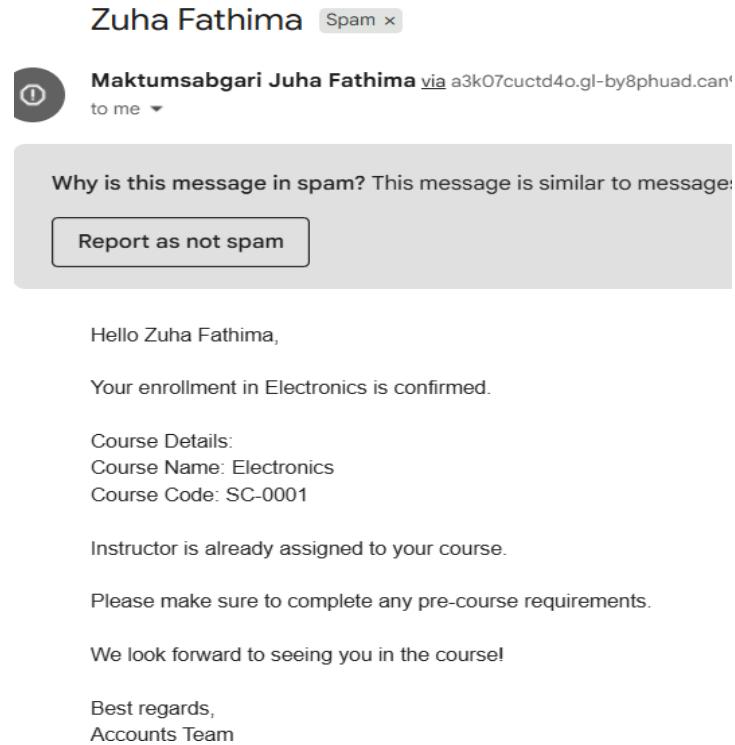


Fig 12: Enrollment Email

Phase 5
Apex Programming (Developer)

1. Objective

Implement Apex logic to:

- Prevent deletion of paid fee payments.
- Calculate outstanding fees for a student.
- Ensure code is tested and achieves $\geq 75\%$ coverage.

Deliverables:

- Apex Trigger
- Apex Classes
- Apex Test Class
- Screenshot of test results and coverage

2. Apex Trigger

Step1: Open Developer Console in a new tab

Step2: Click on new → Apex Trigger

Name: FeePaymentTrigger

Object: Fee_Payment__c

Step3: Click on “OK”

Purpose: Prevent deletion of Fee Payment if status = 'Paid'.

Trigger Code

```
trigger FeePaymentPreventDelete on Fee_Payment__c (before delete) {  
    for (Fee_Payment__c fp : Trigger.old) {  
        // Check if Status = 'Paid'  
        if (fp.Status__c != null &&  
            String.valueOf(fp.Status__c).trim().equalsIgnoreCase('Paid')) {  
                fpaddError('Paid Fee Payments cannot be deleted.');//  
            }  
    }  
}
```

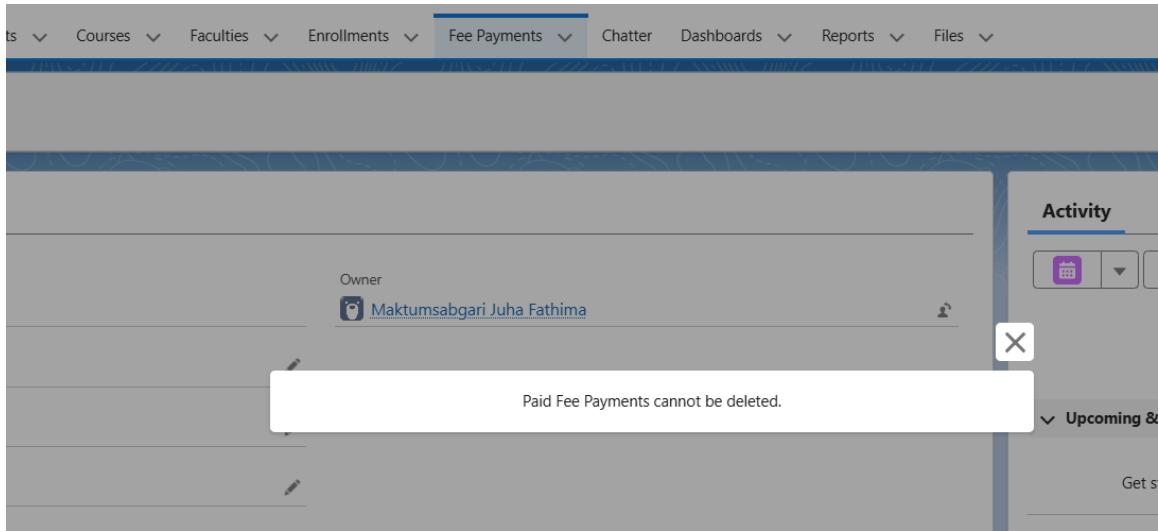


Fig1:Fee payment record detection

FeeCalculator

Step1: Open Developer Console in a new tab

Step2:Click on new→Apex Class

Name: FeeCalculator

Step3:Click on “OK”

Purpose: Calculate outstanding fees for a student.

FeeCalculator Code of Apex class:

```
public class FeeCalculator {  
  
    // Update balance and status for a Fee Payment  
    public static void updateFeePayment(Fee_Payment__c fp) {  
        if(fp.Course__c == null) {  
            throw new AuraHandledException('Course not linked for this Fee Payment.');//  
        }  
  
        // Fetch course fee  
        Course__c course = [SELECT Course_Fee__c FROM Course__c WHERE Id = :fp.Course__c LIMIT 1];  
  
        // Calculate balance  
        Decimal balance = course.Course_Fee__c - fp.Paid_Amount__c;  
  
        // Update Amount field (balance)  
        fp.Amount__c = balance;  
  
        // Update Status  
        if(balance == 0) {  
            fp.Status__c = 'Paid';  
        } else {  
            fp.Status__c = 'Pending');  
        }  
  
        update fp; // Save changes  
    }  
  
    // Method to get outstanding fees for a student  
    public static Decimal getOutstandingFees(Id studentId) {  
        Decimal totalOutstanding = 0;  
  
        // Fetch all fee payments of the student  
        List<Fee_Payment__c> payments = [  
            SELECT Amount__c  
            FROM Fee_Payment__c  
            WHERE Student__c = :studentId  
        ];  
  
        for(Fee_Payment__c fp : payments){
```

```

        totalOutstanding += fp.Amount__c; // Amount__c is balance
    }

    return totalOutstanding;
}
}

```

FeeCalculator Code of Apex Trigger:

Step1: Open Developer Console in a new tab

Step2: Click on new → Apex Trigger

Name: FeePaymentTrigger

Object: Fee_Payment__c

Step3: Click on “OK”

```

trigger FeePaymentTrigger on Fee_Payment__c (before insert, before update) {

    for (Fee_Payment__c fp : Trigger.new) {
        if (fp.Course__c != null) {

            // Fetch course fee
            Course__c course = [
                SELECT Course_Fee__c
                FROM Course__c
                WHERE Id = :fp.Course__c
            ]
        }
    }
}

```

```

LIMIT 1

];

// If Paid_Amount__c is provided
if (fp.Paid_Amount__c != null) {
    // Calculate remaining balance
    Decimal balance = course.Course_Fee__c - fp.Paid_Amount__c;
    fp.Amount__c = balance;

    // Update status
    if (balance == 0) {
        fp.Status__c = 'Paid';
    } else {
        fp.Status__c = 'Pending';
    }

    // Auto-fill today's date
    fp.Payment_Date__c = Date.today();

} else {
    // If no Paid Amount entered
    fp.Amount__c = course.Course_Fee__c;
    fp.Status__c = 'None';
    fp.Payment_Date__c = null;
}

}
}
}
}

```

Fig2:Calcuate amount by code

4. Test Class

Step1: Open Developer Console in a new tab

Step2:Click on new→Apex Class

Name: FeeCalculatorTest

Step3:Click on “OK”

Purpose:Validate trigger and class logic; achieve $\geq 75\%$ code coverage.

Test Class Code

```
@isTest
public class FeeModuleTest {

    @testSetup
    static void createData() {

        // Create Course
        Course__c course = new Course__c(
            Name = 'Test Course',
            ...
        );
    }
}
```

```

        Course_Fee__c = 1000
    );
    insert course;

    // Create Student
    Student__c stu = new Student__c(
        Name = 'Test Student'
    );
    insert stu;

    // Create Paid Fee Payment
    Fee_Payment__c fee1 = new Fee_Payment__c(
        Student__c = stu.Id,
        Course__c = course.Id,
        Paid_Amount__c = 400,
        Status__c = 'Paid'
    );
    insert fee1;

    // Create Pending Fee Payment
    Fee_Payment__c fee2 = new Fee_Payment__c(
        Student__c = stu.Id,
        Course__c = course.Id,
        Paid_Amount__c = 200,
        Status__c = 'Pending'
    );
    insert fee2;

    // Update Amount__c using FeeCalculator
    FeeCalculator.updateFeePayment(fee1);
    FeeCalculator.updateFeePayment(fee2);
}

// Test 1: Outstanding Fee Calculation
@isTest
static void testOutstandingFees() {
    Student__c stu = [SELECT Id FROM Student__c LIMIT 1];
    Decimal outstanding = FeeCalculator.getOutstandingFees(stu.Id);

    // Assuming each Fee Payment has its own balance stored in Amount__c
    // Fee1: 1000 - 400 = 600, Fee2: 1000 - 200 = 800 → total = 1400
    System.assertEquals(1400, outstanding, 'Outstanding Fee should be 1400');
}

// Test 2: Prevent Deletion of Paid Fee (Trigger)
@isTest
static void testTriggerPreventDeletePaid() {
    Fee_Payment__c paidFee = [SELECT Id FROM Fee_Payment__c WHERE Status__c = 'Paid'

```

```

LIMIT 1];

try {
    delete paidFee;
    System.assert(false, 'Deletion should not be allowed for Paid fee');
} catch (DmlException e) {
    System.assert(e.getMessage().contains('cannot be deleted'),
        'Expected error message not found');
}

// Test 3: UpdateFeePayment Method
@isTest
static void testUpdateFeePayment() {
    Fee_Payment__c pendingFee = [SELECT Id, Amount__c FROM Fee_Payment__c WHERE
Status__c = 'Pending' LIMIT 1];

    // Call update method again to ensure it updates Amount__c correctly
    FeeCalculator.updateFeePayment(pendingFee);

    Fee_Payment__c updatedFee = [SELECT Amount__c FROM Fee_Payment__c WHERE Id =
:pendingFee.Id];
    System.assertEquals(800, updatedFee.Amount__c, 'Pending Fee Amount__c should be
updated correctly');
}
}

```

```

72
73     // Call update method again to ensure it updates Amount__c correctly
74     FeeCalculator.updateFeePayment(pendingFee);

Logs Tests Checkpoints Query Editor View State Progress Problems

```

Status	Test Run	Enqueued Time	Duration	Failures	Total	Overall Code Coverage		
x	207gl.00000Eg7Gn	Wed Sep 24 2025 19:33:58 GM...		1	2	Class	Percent	Lines
x	TestRun @ 7:22:58 pm			2	2	Overall	80%	
x	TestRun @ 7:29:06 pm			1	2	FeeCalculator	88%	15/17
v	TestRun @ 7:33:31 pm			0	1	FeePaymentTrigger	71%	10/14
x	TestRun @ 7:43:15 pm			2	3			

Fig 3:Test class

5. Summary

Component	Purpose	
Apex Trigger	Prevent deletion of paid fee payments	
FeeHandler	Centralized logic for trigger	

FeeCalculator	Calculate outstanding fees for a student	
Test Class	Validate logic, test triggers, ensure $\geq 75\%$ coverage	

Deliverables:

- Apex Trigger code (FeePaymentTrigger)
- Apex Classes (FeeHandler, FeeCalculator)
- Apex Test Class (FeeCalculatorTest)
- Screenshot of test results showing code coverage

Phase 6

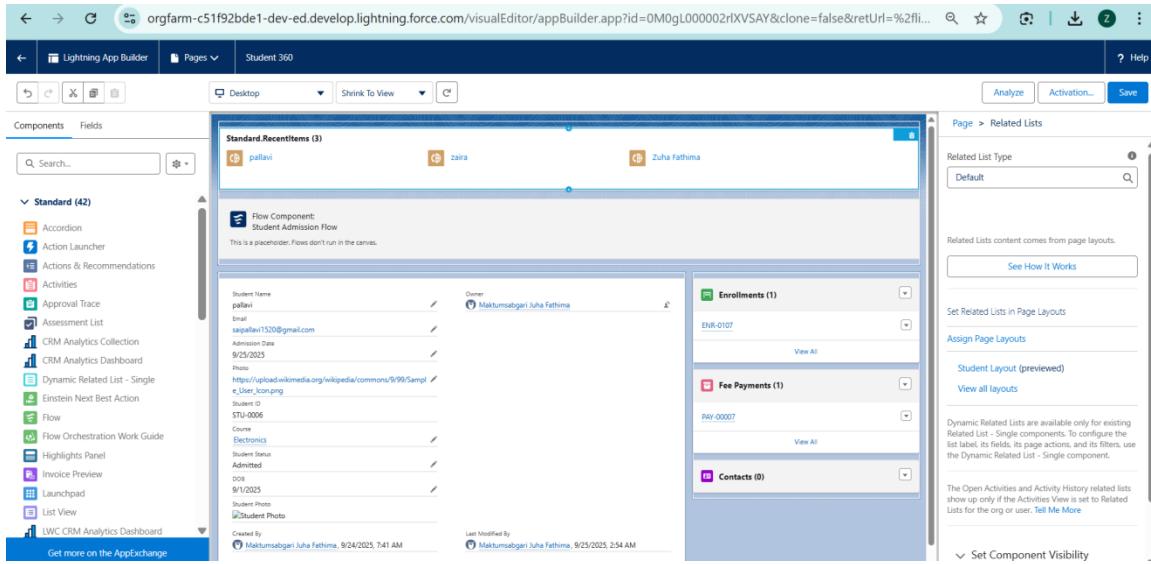
User Interface Development

Goal

Provide a user-friendly interface for managing students, enrollments, and fee payments in Salesforce.

Lightning Record Page for Student (Student 360 Page)

- Navigate to Setup → Object Manager → Student → Lightning Record Pages.
- Create a new App Page or Record Page.
- Add components:
 - Highlights Panel → Student's Name, Email, Status.
 - Related Lists: Enrollments, Fee Payments.
 - Tabs → Info | Related | Fee Calculator (if LWC is added).
- Deliverable → Screenshot of Student 360 Page showing student details + related lists.

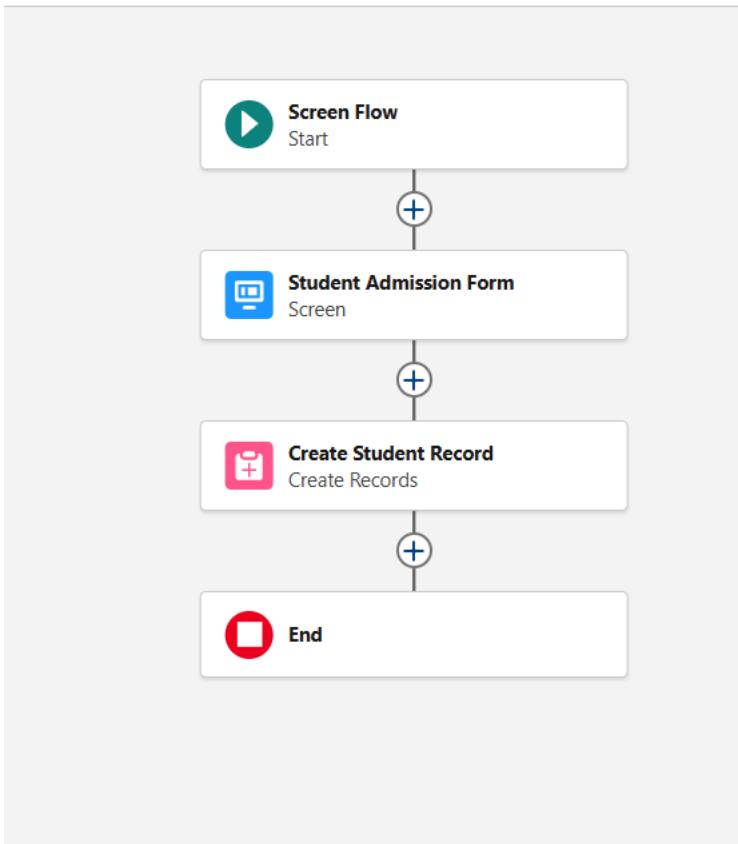


Screen Flow – Student Admission Form

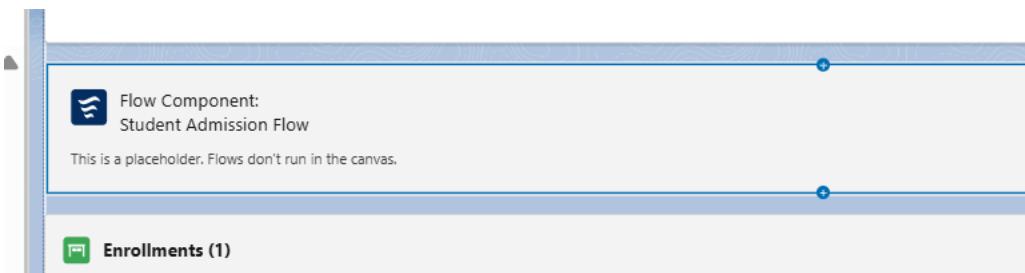
Purpose: Let admins create a new student easily via a guided form.

Steps:

1. Go to Setup → Flow → New Flow → Screen Flow.
2. Screen Element: Add fields for student details (Name, Email, Address, Course selection).
3. Create Records Element:
 - Object = Student
 - Map input fields to Student fields.
4. Connect elements → Save → Activate the Flow.



5. Add Flow to Student App Page using Flow Component.



Result: Admin can admit students using a simple UI form.

Lightning Web Component – Fee Calculator

Purpose: Allow quick calculation of fees (e.g., Course Fee – Paid Amount = Outstanding Fee).

LWC Development Steps:

1. Create Project (in VS Code Command Palette):

SFDX: Create Project with Manifest

2. Create LWC:

SFDX: Create Lightning Web Component

Name → feeCalculator

3. Edit Files:

- feeCalculator.html → Create form (inputs for Course Fee & Paid Fee).

Code:

```
<template>
  <lightning-card title="Fee Calculator">
    <div class="slds-m-around_medium">
      <lightning-input
        label="Course Fee"
        type="number"
        value={courseFee}
        onchange={handleCourseFeeChange}>
      </lightning-input>

      <lightning-input
        label="Paid Amount"
        type="number"
        value={paidAmount}
        onchange={handlePaidAmountChange}>
      </lightning-input>

      <lightning-button
        label="Calculate"
        onclick={calculateOutstanding}
        class="slds-m-top_small">
      </lightning-button>

      <template if:true={outstandingFee}>
        <p class="slds-m-top_medium">
          <b>Outstanding Fee:</b> {outstandingFee}
        </p>
      </template>
    </div>
  </lightning-card>
</template>
```

```

<?xml-version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>58.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>

```

Successfully authorized zuhaf8899720@agentforce.com with org ID 00DgL0000BY8phUAD
17:46:15.276 sf org:login:web --alias MyDevOrg --instance-url https://login.salesforce.com
--set-default
ended with exit code 0

17:47:40.112 Starting SFDX: Create Lightning Web Component
target dir =
c:\Users\mr\Desktop\salesforce>HelloWorldLightningWebComponent\MyLWCProject\force-app\main\default\lwc
create force-app\main\default\lwc\feeCalculator\feeCalculator.js

- feeCalculator.js → Logic to calculate outstanding fee.

Code:

```

import { LightningElement } from 'lwc';

export default class FeeCalculator extends LightningElement {
    courseFee = 0;
    paidAmount = 0;
    outstandingFee;

    handleCourseFeeChange(event) {
        this.courseFee = parseFloat(event.target.value) || 0;
    }

    handlePaidAmountChange(event) {
        this.paidAmount = parseFloat(event.target.value) || 0;
    }

    calculateOutstanding() {
        this.outstandingFee = this.courseFee - this.paidAmount;
    }
}

```

- feeCalculator.js-meta.xml → Expose component to Record Page.

Code:

```

<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
  <apiVersion>58.0</apiVersion>
  <isExposed>true</isExposed>
  <targets>
    <target>lightning__RecordPage</target>
    <target>lightning__AppPage</target>
    <target>lightning__HomePage</target>
  </targets>
</LightningComponentBundle>

```

4. Deploy Component:

```
sf project deploy start --metadata LightningComponentBundle:feeCalculator -o
```

```

17:47:40.212 Finished SFDX: Create Lightning Web Component
17:50:27.412 Starting SFDX: Deploy This Source to Org

== Deployed Source
STATE      FULL NAME          TYPE           PROJECT
PATH

Created feeCalculator LightningComponentBundle
force-app\main\default\lwc\feeCalculator\feeCalculator.html
Created feeCalculator LightningComponentBundle
force-app\main\default\lwc\feeCalculator\feeCalculator.js
Created feeCalculator LightningComponentBundle
force-app\main\default\lwc\feeCalculator\feeCalculator.js-meta.xml

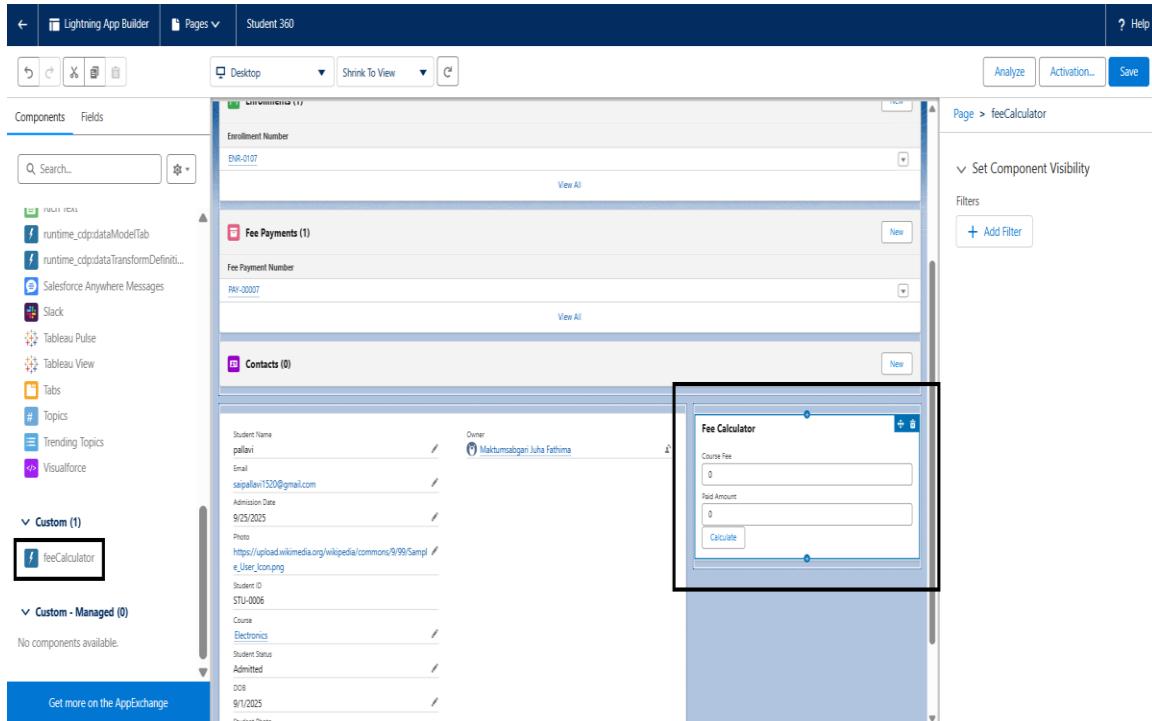
17:50:30.877 Ended SFDX: Deploy This Source to Org

```

MyDevOrg

5. Add to Lightning Page:

- Go to Lightning App Builder → Drag feeCalculator onto Student Page.



Result: Admins can instantly calculate and display outstanding fees inside the Student 360 page.

Results

- **Student 360 Page:** Displays student info and related records in a tabbed layout.
- **Screen Flow:** Enables guided student admission.
- **Fee Calculator LWC:** Interactive tool for fee management.
- **Overall Outcome:** Clean, intuitive, and functional UI for managing student-related data in Salesforce.

Phase 7

Integration and external access

Goal: Connect the Institute Management System with external systems (payment gateways, university systems, library DBs, analytics) in a secure, maintainable way. Below are step-by-step instructions, configuration clicks, and small code examples you can copy into your org.

1) Named Credential — store external API credentials securely

Why: avoid hard-coding endpoints/credentials; simplifies callouts and OAuth.

Steps

1. If using OAuth: create an Auth. Provider first
 - o Setup → Auth. Providers → New → choose provider (Google, OpenID Connect, etc.) → fill client id/secret → Save.

The screenshot shows the Salesforce Setup interface with the following details:

Auth. Provider Detail

Auth. Provider ID	0SOgL000000srAb	Edit
Provider Type	Google	Delete
Name	Google	Clone
URL Suffix	Google	
Consumer Key	1234	
Consumer Secret	Click to reveal	
Authorize Endpoint URL		
Token Endpoint URL		
User Info Endpoint URL		
Use Proof Key for Code Exchange (PKCE) Extension	<input checked="" type="checkbox"/> i	
Default Scopes		
Include Consumer Secret in SOAP API Responses	<input checked="" type="checkbox"/> i	
Custom Error URL		
Custom Logout URL		
Registration Handler Type	None	
	Portal	
Icon URL		

2. Setup → Named Credentials → New Named Credential.
 - o Label / Name: UniversityAPI_NC
 - o URL: <https://api.university.example> (base)
 - o Identity Type: *Named Principal* (or *Per User* if required)
 - o Authentication Protocol: *OAuth 2.0* (pick Auth. Provider) or *Password Authentication*
 - o Save.

Usage: Apex callouts use callout:UniversityAPI_NC/endpoint/path.

Named Credential Edit: UniversityAPI_NC

Specify the callout endpoint's URL and the authentication settings that are required for Salesforce to make callouts to the remote system

		Save	Cancel
Label	<input type="text" value="UniversityAPI_NC"/>		
Name	<input type="text" value="UniversityAPI_NC"/>		
URL	<input type="text" value="https://api.university.example"/>		
Authentication			
Certificate	<input type="text"/>		
Identity Type	<input type="text" value="Named Principal"/>		
Authentication Protocol	<input type="text" value="OAuth 2.0"/>		
Authentication Provider	<input type="text" value="Google"/>		
Scope	<input type="text"/>		
Authentication Status	Pending		
Start Authentication Flow on Save	<input checked="" type="checkbox"/>		

2) External Services — connect OpenAPI (no code flows)

Why: expose external REST endpoints as Actions in Flow/Process Builder without hand-coding.

Steps

1. Get an OpenAPI (Swagger) JSON/YAML for the external API.
2. Setup → External Services → New External Service.
 - Provide Named Credential (created above) and upload the OpenAPI spec.
3. Once registered, the External Service actions appear in Flow as Apex Actions — you can drag them into a Flow and call external operations.

3) Web Services (REST/SOAP) callouts from Apex

Why: verify insurance, call payment gateway, sync student data.

Steps

1. Create Named Credential (preferred) or Remote Site Setting.
2. Write Apex that uses HttpRequest + Http and points to callout:Named_Credential/....

Apex example (REST using Named Credential):

```
public with sharing class ExternalIntegration {  
    @future(callout=true)  
  
    public static void notifyEnrollment(Id enrollmentId) {  
  
        Enrollment__c e = [SELECT Id, Student__r.Email__c, Course__r.Course_Name__c,  
        Enrollment_Status__c  
  
        FROM Enrollment__c WHERE Id = :enrollmentId LIMIT 1];  
  
        HttpRequest req = new HttpRequest();  
  
        req.setEndpoint('callout:UniversityAPI_NC/v1/enrollments'); // Named Credential  
        req.setMethod('POST');  
  
        req.setHeader('Content-Type','application/json');  
  
        req.setBody(JSON.serialize(new Map<String, Object>{  
  
            'enrollmentId' => e.Id,  
  
            'studentEmail' => e.Student__r.Email__c,  
  
            'course' => e.Course__r.Course_Name__c,  
  
            'status' => e.Enrollment_Status__c  
  
        }));  
  
        Http http = new Http();  
  
        HttpResponse res = http.send(req);  
  
        // handle res.getStatusCode() / body  
    }  
}
```

Testing: implement HttpCalloutMock and use Test.setMock(...) in test classes.

4) Callouts triggered when records change

Why: notify external systems immediately when relevant IMS records change.

Steps

1. Create a trigger on the Salesforce object (e.g., Enrollment__c AFTER insert/after update).
2. In trigger handler, decide when to notify (e.g., status changed).
3. Enqueue an async job that performs callout (@future(callout=true) or Queueable implementing Database.AllowCallouts).

I've already implemented that in **Phase 3–4** with your **Fee Payment Trigger + Enrollment Trigger** calling the async Apex (@future) methods.

Why it's sufficient:

- The triggers detect **insert/update events**.
 - They only call the external system when the **status changes** (Enrollment_Status__c or Status__c = 'Paid').
 - The async Apex (@future(callout=true) or Queueable) handles the actual REST call.
- No extra steps are needed for this, because the logic is already **event-driven** and **bulk-safe**, fulfilling the requirement of callouts triggered by record changes.

5.Creation of Platform events

Step 1: Create the Platform Event

1. Go to **Setup → Platform Events → New Platform Event**.
2. Fill in the details:
 - **Label:** FeePaid_Event
 - **Plural Label:** FeePaid_Events
 - **API Name:** FeePaid_Event__e
 - **Publish Behavior:** Publish Immediately (so events are available as soon as saved)
3. Click **Save**.

Step 2: Add Fields to the Platform Event

Create fields to capture the information you want to send:

1. **EnrollmentId__c → Text (18)**
2. **StudentEmail__c → Email**
3. **FeeAmount__c → Number**
4. **Status__c → Text**

Click **Save** after adding all fields.

Platform Events

Event Allocations

Item	Usage	Allocation
High Volume Platform Event Hourly Publishing Allocation	0	50,000
High Volume Platform Event and Change Event Daily Delivery Allocation	0	10,000

Custom Events

Action	Label	Deployed	Description
Edit Del	FeePaid_Event	✓	

Platform Events

Standard Fields

Action	Field Label	Field Name	Data Type	Controlling Field
Created By	CreatedBy	Lookup(User)		
Created Date	CreatedDate	Date/Time		
Event UUID	EventUuid	Text(36)		
Replay ID	ReplayId	External Lookup		

Custom Fields & Relationships

Action	Field Label	API Name	Data Type	Indexed	Controlling Field	Modified By
Edit Del	EnrollmentId_c	EnrollmentId_c_c	Text(17)			Maksumsabgari_Juha Fathima, 9/26/2025, 2:25 AM
Edit Del	FeeAmount_c	FeeAmount_c_c	Number(18, 0)			Maksumsabgari_Juha Fathima, 9/26/2025, 2:26 AM
Edit Del	Status	Status_c	Text(18)			Maksumsabgari_Juha Fathima, 9/26/2025, 2:26 AM

Triggers

Action	Field Label	API Name	Data Type	Indexed	Controlling Field	Modified By
New						
No triggers defined						

6) Change Data Capture (CDC)

- **Reason:** Overhead unless you have **large-scale external data sync** (like nightly export of *every change* to a data warehouse).
- IMS is usually transactional → Platform Events are enough.

7) Salesforce Connect

- **Reason:** Only needed if **major data (like student master data or course catalog)** lives in an external DB (e.g., Oracle, SAP, university ERP).
- If IMS data is **all inside Salesforce**, ✗ no need.

Phase 8

Data Management & Deployment

Goal

To efficiently manage, migrate, and secure student data and metadata across Salesforce environments using native tools, packages, and modern deployment methods.

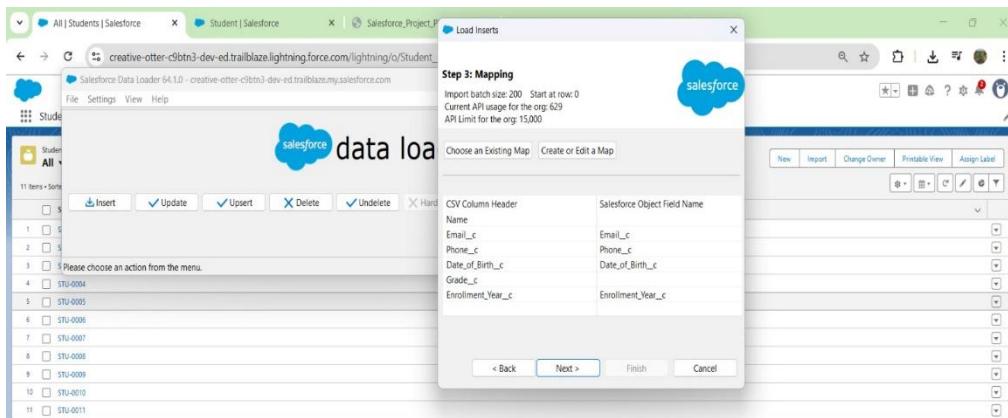
1. Data Import Wizard

- Go to Setup → Data Import Wizard in Salesforce.
 - Select the object (Student, Faculty, Course).
 - Upload the CSV file containing sample data.
 - Map CSV columns to Salesforce fields.
 - Click Start Import.
 - Verify import success via list views and reports.

	A	B	C	D	E
1	Id	Success	Created	Error	
2	a01gL0000	TRUE	TRUE		
3	a01gL0000	TRUE	TRUE		
4	a01gL0000	TRUE	TRUE		
5	a01gL0000	TRUE	TRUE		
6	a01gL0000	TRUE	TRUE		
7					
8					
9					
10					

2. Data Loader (Optional for Bulk Operations)

- Download and install Data Loader.
- Open Data Loader → Insert → Log in to Salesforce.
- Choose the object (Student, Faculty, Course).
- Browse and select the CSV file.
- Map CSV columns to Salesforce fields.
- Click Next → Finish → OK.
- Review success and error CSVs for audit.



3. Duplicate Rules

- Go to Setup → Duplicate Management → Matching Rules.
- Create rules to detect duplicate student records.
- Configure Duplicate Rules to block or alert duplicates.
- Test with sample data to ensure data quality.

Your Salesforce matching rule is now activated Inbox ×



Salesforce Duplicate Management (noreply@salesforce.com) <noreply@salesforce.com>

to me ▾

Hello Maktumsabgari Juha Fathima,

Your matching rule Duplicate prevention for identifying duplicate records has been activated and is now ready to use.

Salesforce Duplicate Management

You're registered as zuhaf8899@gmail.com in CodeSphere Institute Of Technology. Need help? Contact Salesforce Customer Support.

Immission Date

Photo

Student ID

Course

Search Courses

Student Status

--None--

⚠ Similar Records Exist ×

This record looks like an existing record. Make sure to check any potential duplicate records before saving.

[View Duplicates](#)

⚠ Cancel Save & New Save

4. Data Export & Backup

- Go to Setup → Data Export.
- Schedule weekly export of critical objects (Students, Courses, Enrollments).
- Download CSVs and store securely for rollback.

Schedule Data Export

Schedule Data Export

Save Cancel

Export File Encoding: ISO-8859-1 (General US & Western European, ISO-LATIN-1)

Include images, documents, and attachments:

Include Salesforce Files and Salesforce CRM Content document versions:

Replace carriage returns with spaces:

Schedule Data Export

Frequency: On day 1 of every month

Start: 9/25/2025 [9/25/2025]

End: 10/25/2025 [9/25/2025]

Preferred Start Time: 9:00 AM

Exact start time will depend on job queue activity.

Leads Accounts Contacts Opportunities Forecasts Contracts Orders Cases Solutions Products Reports Dashboards +

Help for this Page ?

Monthly Export Service

Next scheduled export: None

Export Now Schedule Export

Scheduled By: Maktumsabgari Juha Fathima

Schedule Date: 9/26/2025

Export File Encoding: ISO-8859-1 (General US & Western European, ISO-LATIN-1)

Action	File Name	File Size
download	WE_00DgL00000BY8phUAD_1.ZIP	5.3K

5. Deploy Metadata

Option A: VS Code & Salesforce CLI

- Move your project to a local folder (avoid OneDrive).
- Open VS Code and authorize org:

sf org login web --alias SourceOrg --set-default

- Retrieve metadata (Student, Course, Faculty objects):

sf project retrieve start --metadata CustomObject:Student__c --metadata

CustomObject:Course__c --metadata CustomObject:Faculty__c

- Verify retrieved files in force-app/main/default/objects/.

- Deploy to target org:

sf project deploy start --source-dir force-app/main/default

- Confirm success in terminal or VS Code output.

Option B: Change Sets

- In Source Org → Setup → Outbound Change Set → New.

- Add components (objects, fields, Apex classes).

- Upload to target org → Inbound Change Set → Deploy → Validate.

STATE	FULL NAME	TYPE	PROJECT PATH
Unchanged	Course__c	CustomObject	force-app/main/default/objects\Course__c\Course__c.object-meta.xml
Unchanged	Course_c.Course_Code__c	CustomField	force-app/main/default/objects\Course__c\fields\Course_Code__c.field-meta.xml
Unchanged	Course_c.Course_Fee__c	CustomField	force-app/main/default/objects\Course__c\fields\Course_Fee__c.field-meta.xml
Unchanged	Course_c.Credits__c	CustomField	force-app/main/default/objects\Course__c\fields\Credits__c.field-meta.xml
Unchanged	Course_c.Duration__c	CustomField	force-app/main/default/objects\Course__c\fields\Duration__c.field-meta.xml
Unchanged	Course_c.End_Date__c	CustomField	force-app/main/default/objects\Course__c\fields\End_Date__c.field-meta.xml
Unchanged	Course_c.Enrolled_Students__c	CustomField	force-app/main/default/objects\Course__c\fields\Enrolled_Students__c.field-meta.xml
Unchanged	Course_c.Faculty__c	CustomField	force-app/main/default/objects\Course__c\fields\Faculty__c.field-meta.xml
Unchanged	Course_c.Seats__c	CustomField	force-app/main/default/objects\Course__c\fields\Seats__c.field-meta.xml
Unchanged	Course_c.Start_Date__c	CustomField	force-app/main/default/objects\Course__c\fields\Start_Date__c.field-meta.xml
Unchanged	Course_c.All	ListView	force-app/main/default/objects\Course__c\listViews\All.listView-meta.xml
Unchanged	Course_c.StartDate_Check	ValidationRule	force-app/main/default/objects\Course__c\ValidationRules\StartDate_Check.validationRule-meta.xml
Unchanged	Faculty__c	CustomObject	force-app/main/default/objects\Faculty__c\Faculty__c.object-meta.xml
Unchanged	Faculty_c.Department__c	CustomField	force-app/main/default/objects\Faculty__c\fields\Department__c.field-meta.xml
Unchanged	Faculty_c.Email__c	CustomField	force-app/main/default/objects\Faculty__c\fields\Email__c.field-meta.xml
Unchanged	Faculty_c.Employee_ID__c	CustomField	force-app/main/default/objects\Faculty__c\fields\Employee_ID__c.field-meta.xml
Unchanged	Faculty_c.Status__c	CustomField	force-app/main/default/objects\Faculty__c\fields>Status__c.field-meta.xml
Unchanged	Faculty_c.All	ListView	force-app/main/default/objects\Faculty__c\listViews\All.listView-meta.xml
Unchanged	Student__c	CustomObject	force-app/main/default/objects\Student__c\Student__c.object-meta.xml
Unchanged	Student_c.Admission_Date__c	CustomField	force-app/main/default/objects\Student__c\fields\Admission_Date__c.field-meta.xml
Unchanged	Student_c.Course__c	CustomField	force-app/main/default/objects\Student__c\fields\Course__c.field-meta.xml
Unchanged	Student_c.DOB__c	CustomField	force-app/main/default/objects\Student__c\fields\DOB__c.field-meta.xml
Unchanged	Student_c.Email__c	CustomField	force-app/main/default/objects\Student__c\fields\Email__c.field-meta.xml
Unchanged	Student_c.Photo__c	CustomField	force-app/main/default/objects\Student__c\fields\Photo__c.field-meta.xml
Unchanged	Student_c.Student_ID__c	CustomField	force-app/main/default/objects\Student__c\fields\Student_ID__c.field-meta.xml
Unchanged	Student_c.Student_Photo__c	CustomField	force-app/main/default/objects\Student__c\fields\Student_Photo__c.field-meta.xml
Unchanged	Student_c.Student_Status__c	CustomField	force-app/main/default/objects\Student__c\fields\Student_Status__c.field-meta.xml
Unchanged	Student_c.All	ListView	force-app/main/default/objects\Student__c\listViews\All.listView-meta.xml
Unchanged	Student_c.DOB_Check	ValidationRule	force-app/main/default/objects\Student__c\ValidationRules\DOB_Check.validationRule-meta.xml

6. ANT Migration Tool

- Configure build.xml and package.xml.
- Retrieve custom object and Apex class metadata from source org.
- Deploy changes to target org.
- Validate success logs.

7. Notes & Best Practices

- Always retrieve metadata to a local folder, not OneDrive, to avoid empty folders.
- Verify all fields, list views, and validation rules are retrieved.
- Keep backups of CSV files and metadata.
- Use CLI commands with correct syntax in new Salesforce CLI (sf).

Phase 9

Reports & Dashboards

This document provides the step-by-step instructions to create Reports and Dashboards in Salesforce as part of Phase 9 implementation, including report types and dashboard types.

1. Reports

Salesforce offers different types of reports depending on the analysis needed.

Report Types

- Tabular Reports: Simple lists of data; ideal for exporting or quick views. Example: Student List, Attendance List.
- Summary Reports: Grouped data with subtotals; useful for summaries. Example: Class-wise Student Count, Attendance Percentage by Class.
- Matrix Reports: Data grouped by rows and columns; ideal for cross-tab analysis. Example: Attendance Matrix (Classes vs Months), Grades Matrix (Subjects vs Terms).
- Joined Reports: Combine multiple report types into one view; good for holistic performance. Example: Student + Attendance + Grades, Course Enrollment + Completion Rate.

Steps to Create Reports

- Go to the App Launcher (9 dots) and search for 'Reports'.
- Click on 'New Report'.
- Choose the appropriate report type (Tabular, Summary, Matrix, Joined) based on your data needs.
- Click 'Continue'.
- Add required filters (e.g., filter by 'Status = Active' for Students).
- Drag and drop fields into the report outline (such as Name, Program, Semester, Fee Paid).
- Group data if required (Summary/Matrix reports).
- Click 'Run' to preview the data.
- Save the report with a suitable name and folder (e.g., 'Student Reports').

Report: Enrollments with Course

New Enrollments with Course Report

Total Records
6

	Enrollment: Enrollment Number ▾	Course: Course Name ▾
1	ENR-0107	Electronics
2	ENR-0106	Electronics
3	ENR-0105	Electronics
4	ENR-0104	Electronics
5	ENR-0101	Electronics
6	ENR-0108	Python

Reports

Recent

6 items

Search recent reports...

REPORTS	Report Name	Description	Folder	Created By	Created On	Subsc
Recent	New Enrollments with Course Report		Private Reports	Maktumsabgari Juha Fathima	9/25/2025, 3:26 PM	
Created by Me	New Faculties Report		Private Reports	Maktumsabgari Juha Fathima	9/25/2025, 3:29 PM	
Private Reports	New Students with Course Report		Private Reports	Maktumsabgari Juha Fathima	9/25/2025, 3:21 PM	
Public Reports	Lead conversion rate	What percentage of leads have been converted?	Channel Sales	Automated Process	9/25/2025, 8:18 AM	
All Reports	Exercise Completion Status by Section	Analyze an exercise's completion status based on program sections.	Enablement Dashboard Reports Summer '24	Automated Process	9/12/2025, 10:59 PM	
FOLDERS	Sample Flow Report: Screen Flows	Which flows run, what's the status of each interview, and how long do users take to complete the screens?	Public Reports	Automated Process	9/12/2025, 10:59 PM	
All Folders						
Created by Me						

2. Dashboards

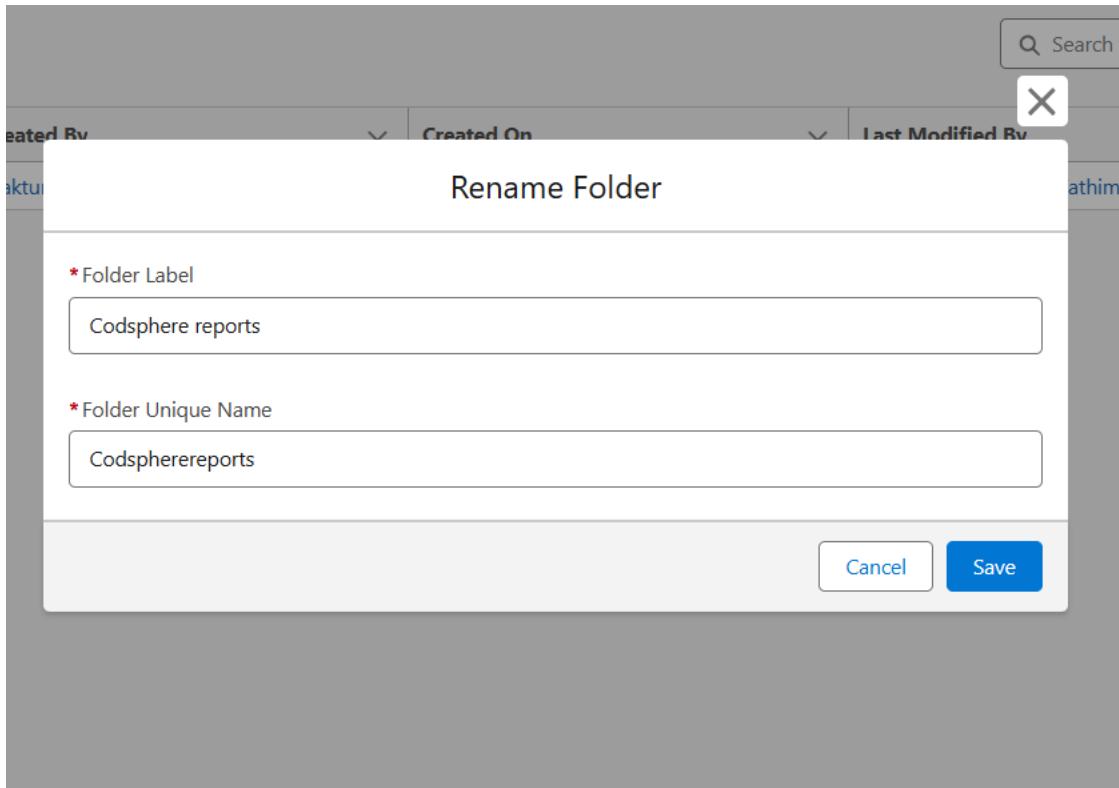
Dashboards allow visual representation of reports. Each dashboard can contain multiple components based on report charts.

Dashboard Types

- Chart Component: Visual charts like bar, line, pie, or donut charts.
- Gauge Component: Shows a single metric in a gauge style, useful for targets.
- Metric Component: Displays a single number or KPI metric.
- Table Component: Displays tabular data from reports.
- Visualforce Component: Embeds Visualforce pages or custom content in dashboard.

Steps to Create Dashboards

- Go to the App Launcher and search for 'Dashboards'.
- Click on 'New Dashboard'.
- Enter the Dashboard Name (e.g., Student & Fee Insights) and select a folder.
- Click 'Create'.
- Click '+ Component' to add a chart or table.
- Select the source report (e.g., 'Fee Payment by Semester').
- Choose the display type (bar chart, pie chart, line chart, gauge, metric, table, etc.).
- Adjust chart settings like labels, legends, and filters.
- Click 'Add' and repeat to add more components.
- Save the dashboard and click 'Done'.
- View the dashboard to monitor reports visually.



The screenshot shows the CodeSphere Institute dashboard with a navigation bar at the top: CodeSphere Institute, Students, Courses, Faculties, Enrollments, Fee Payments, Chatter, Dashboards (selected), Reports, Files. Below the navigation is a toolbar with 'Widget', 'Filter', 'Save', and 'Done' buttons. The main area displays three report cards:

- New Faculties Report**: Faculty: Faculty Name ↑

Basha
Nida
Reshma
Rija

[View Report \(New Faculties Report\)](#)
- New Enrollments with Course Report**: Enrollment: Enrollment Number ↑ Course: Course Name ↑

ENR-0101	Electronics
ENR-0104	Electronics
ENR-0105	Electronics
ENR-0106	Electronics
ENR-0107	Electronics
ENR-0108	Python

[View Report \(New Enrollments with Course Report\)](#)
- New Students with Course Report**: Student: Student Name ↑ Course: Course Name ↑

Nida	Electronics
paltavi	Electronics
zaira	Electronics
Zuha Fatima	Electronics
Zuhab	Python

[View Report \(New Students with Course Report\)](#)

Phase 10

Integration & Automation

1. Integration

- **Email Integration:**
 - Configure Salesforce Email-to-Case or Email Alerts for key events (e.g., admission confirmation, fee reminders).
 - **Calendar Integration:**
 - Sync academic schedules and exam timetables with Salesforce Calendar or Google Calendar integration.
 - **Payment Gateway Integration:**
 - Connect fee collection system with external payment services (e.g., Razorpay, PayPal, Stripe).
 - **External Systems:**
 - If the institute uses ERP/Library/Attendance software, integrate via **APIs or middleware (MuleSoft/REST API)**.
-

2. Automation

- **Process Builder / Flows:**
 - Automate student admission approvals.
 - Send automatic reminders for pending fees.
 - Auto-assign faculty to newly created course batches.
 - **Scheduled Jobs:**
 - Monthly fee due reports.
 - Automatic grade publishing notifications.
 - **Approval Processes:**
 - Multi-level approvals for scholarships, leave requests, or refunds.
-

3. Benefits

- Reduced manual workload.
- Timely communication with students and faculty.
- Centralized system connecting academics, finance, and administration.
- Higher efficiency and accuracy in institute operations.