Problem Statement:

Institute Management System

Educational institutes handle a large number of inquiries through their website, direct walk-ins, and social media channels. Despite this, they face several issues:

- Lead follow-ups are irregular and often delayed
- Student information is managed manually in spreadsheets
- Faculty assignments lack workload balance
- Fee collection and monitoring are fragmented
- Management does not have access to real-time analytics for admissions, revenue, or faculty performance

👉 To address these pain points, the institute plans to roll out Salesforce CRM (IMS) to:

- Automate inquiry capture and admission qualification
- Centralize student records, faculty details, courses, and batch management
- Simplify fee tracking with reminders and receipts
- Deliver real-time dashboards for admissions, finance, and faculty utilization
- Strengthen communication with students and parents via SMS/Email notification.

Core Use Cases:

1. Lead Management:

- Capture leads automatically from website forms, walk-ins, and social channels
- Route leads to admission officers based on course or geography
- Evaluate and score leads by interest levels

2. Student Management:

- Maintain detailed student profiles including history, enrolled programs, and status
- Convert qualified leads into enrolled student records

3. Faculty Management:

- Maintain faculty profiles with expertise, skills, and availability
- Assign faculty to batches/courses while balancing workloads

4. Course & Batch Scheduling:

- Plan lectures, exams, and institutional events
- Trigger SMS/Email reminders for students and faculty

5. Fee Management:

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

6. Reporting & Analytics:

- Dashboards for admission funnel, revenue insights, and faculty workload
- Reports on enrolment patterns, counsellor performance, and fee collection status

INSTITUTE MANAGEMENT SYSTEM

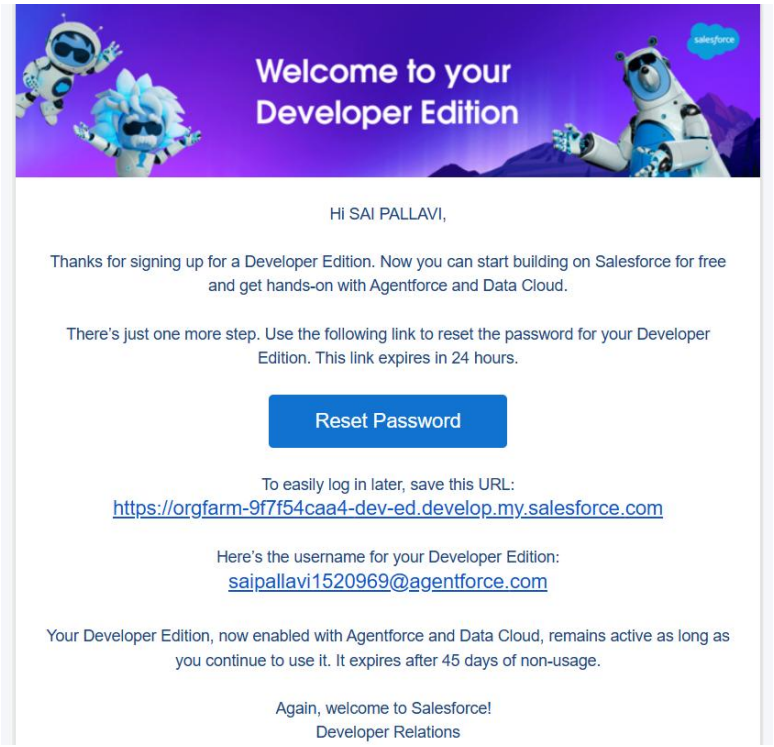
Phase 2: Org Setup & Configuration

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.

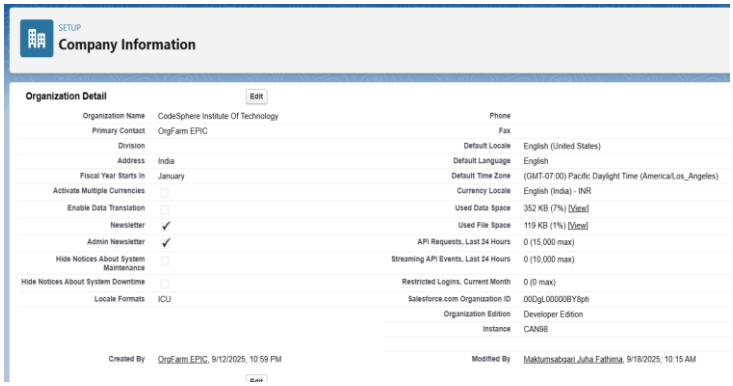


Step 2: Setup Company Profile

1. Navigate: Setup → Company Information.

2. Update:

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



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.

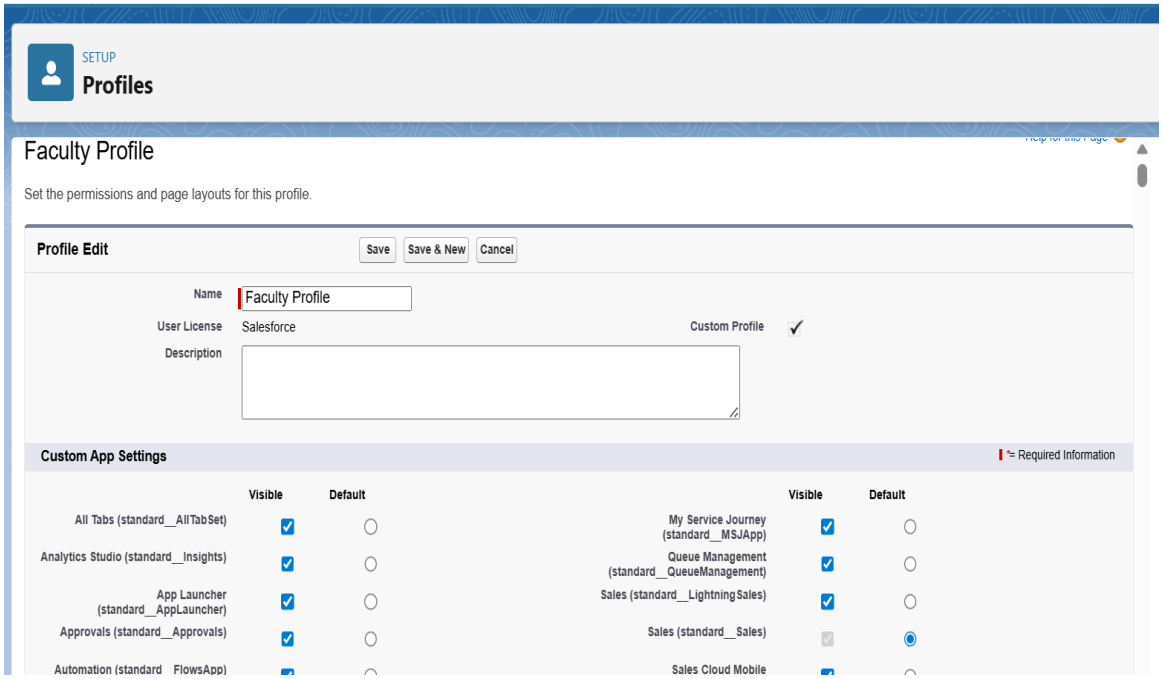
- Accounts Profile → for Accountant.

- Student Profile → for Students

- Faculty → Manage Courses, Student.

- Accounts → Manage Fee Payments, Reports.

- Students → Read-only access on their own record.

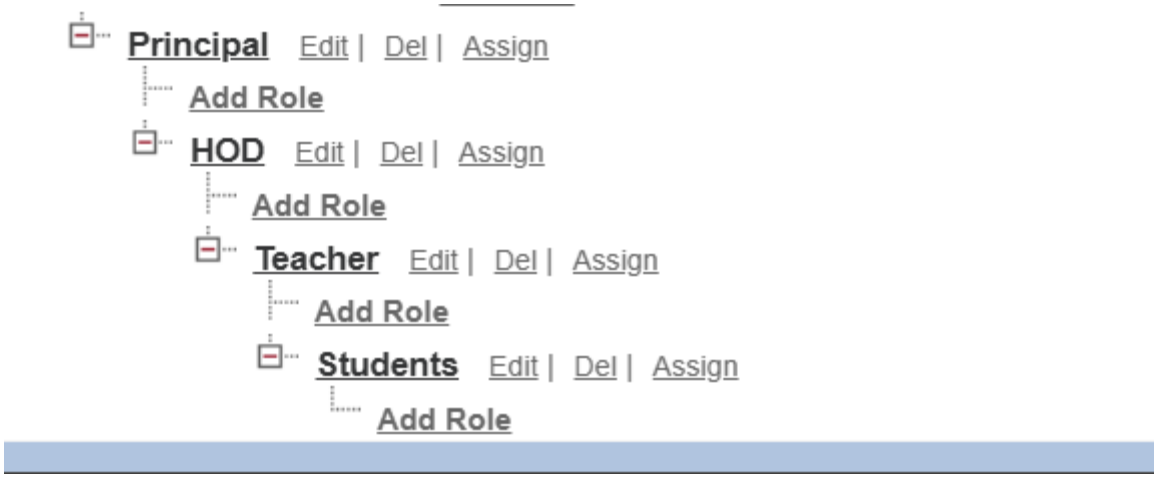


Step 4: Define Roles

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).



Step 5: Create Users

1. Navigate: Setup → Users → New User.

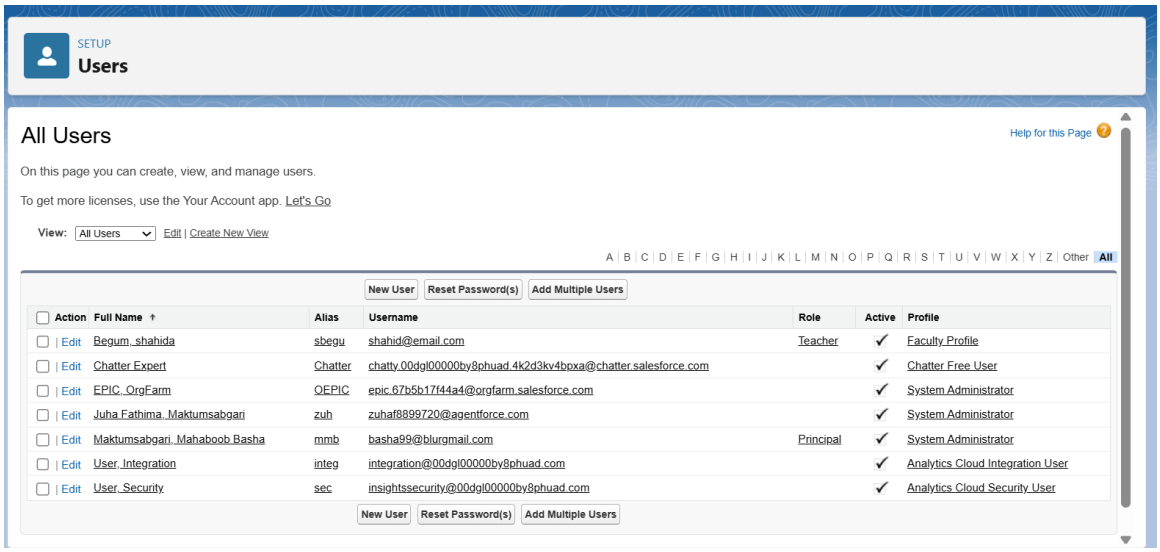
2. Assign details:

   - User License = Salesforce (for staff).

   - Profile = Faculty, Accounts, System Admin, etc.

   - Role = Principal, Teacher, Accountant, Student.

3. Create sample test users: Principal, Teacher.



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.

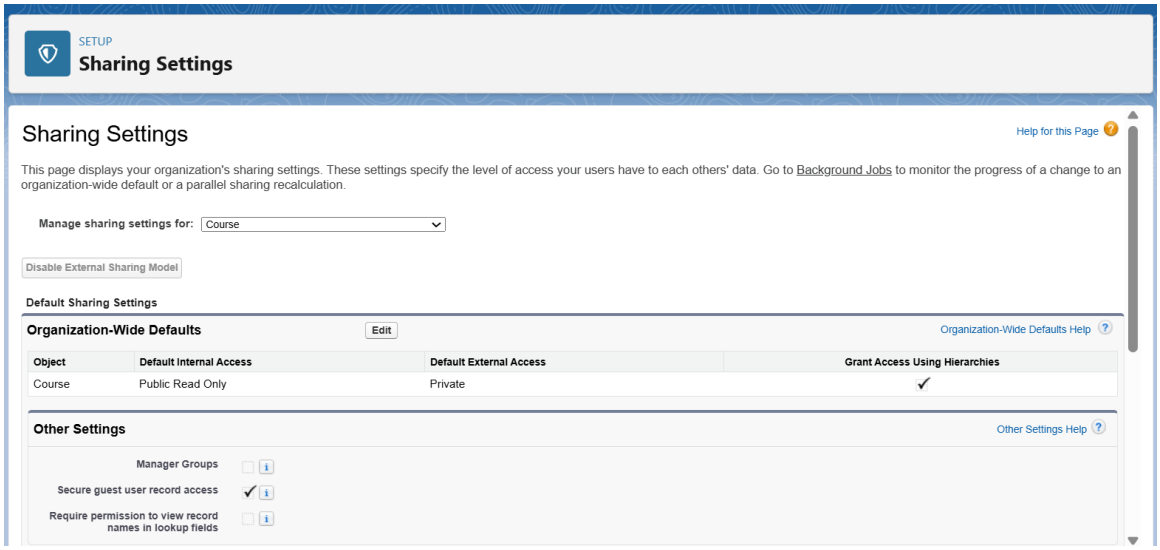
With this setup, my Salesforce Educational Org will have:

- 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.


SETUP
Sharing Settings
Sharing Settings
Help for this Page
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.
Manage sharing settings for: Course
Disable External Sharing Model
Default Sharing Settings
Organization-Wide Defaults
Edit
Organization-Wide Defaults Help
Object
Default Internal Access
Default External Access
Grant Access Using Hierarchies
Course
Public Read Only
Private
Other Settings
Other Settings Help
Manager Groups
Secure guest user record access
Require permission to view record names in lookup fields

Institute Management System

PHASE 3: Data Modeling & Relationships

1. Standard & Custom Objects:

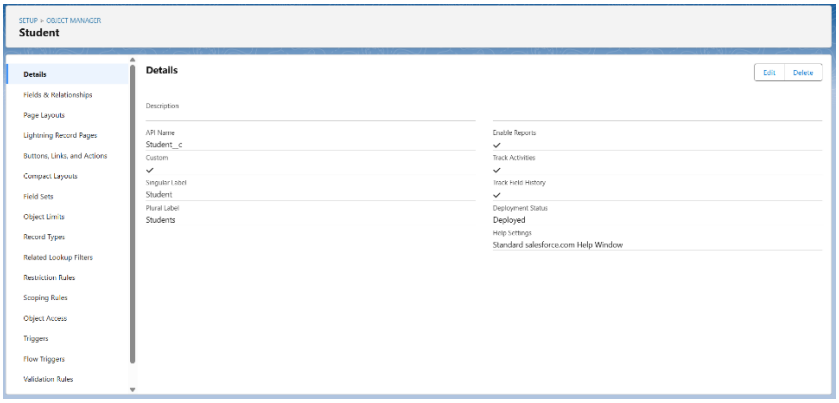
- Contact is used as the standard object.

Creation of the custom objects:

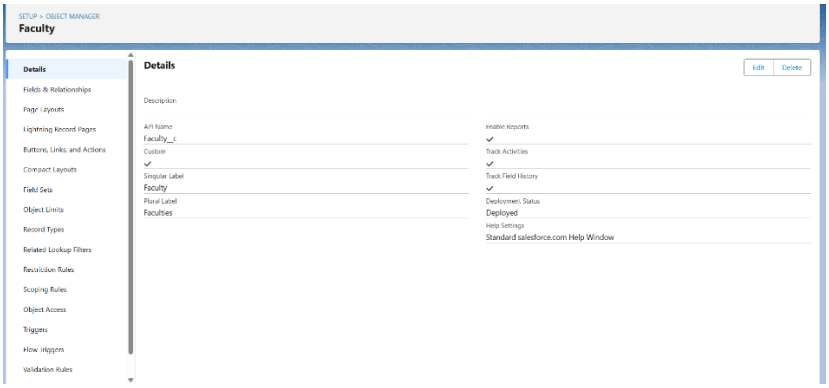
(Setup → Object Manager → **Create** → **Custom Object**)

Create these objects with these settings (recommended API names shown):

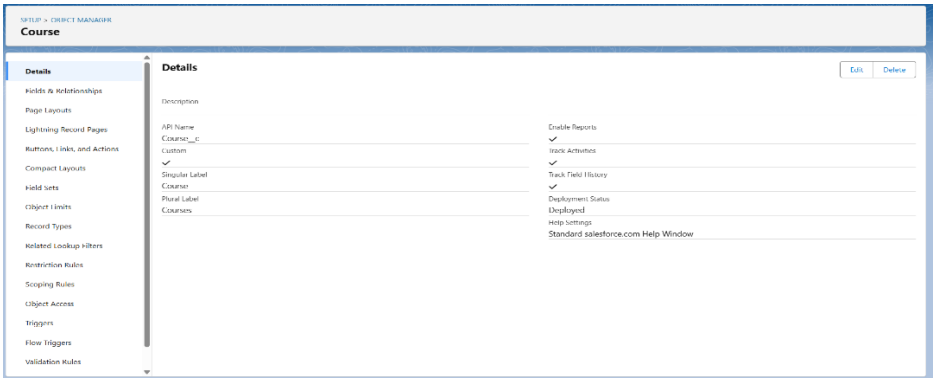
1. **Student** (Student__c)
    - Record Name: *Student ID* — **Auto Number** → Display format STU-{0000}; Starting Number 1001.
    - Check: Allow Reports, Allow Activities, Track Field History (optional).



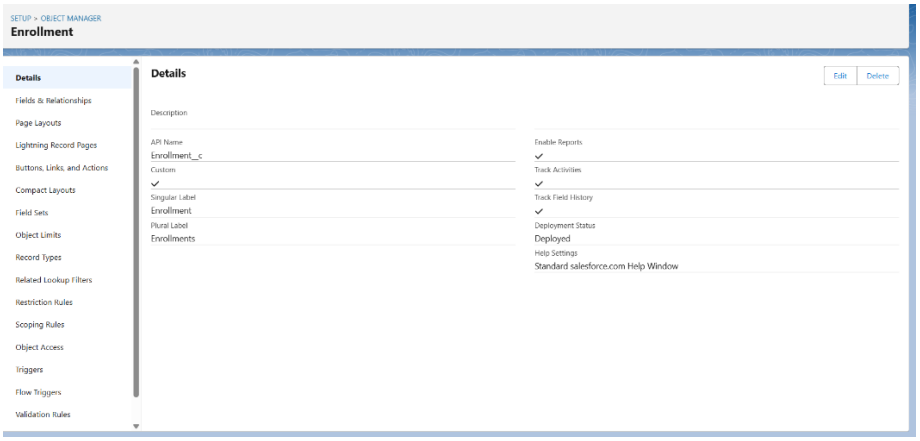
2. **Faculty** (Faculty__c)
    - Record Name: *Faculty ID* — Auto Number FAC-{0000}, start 1001.



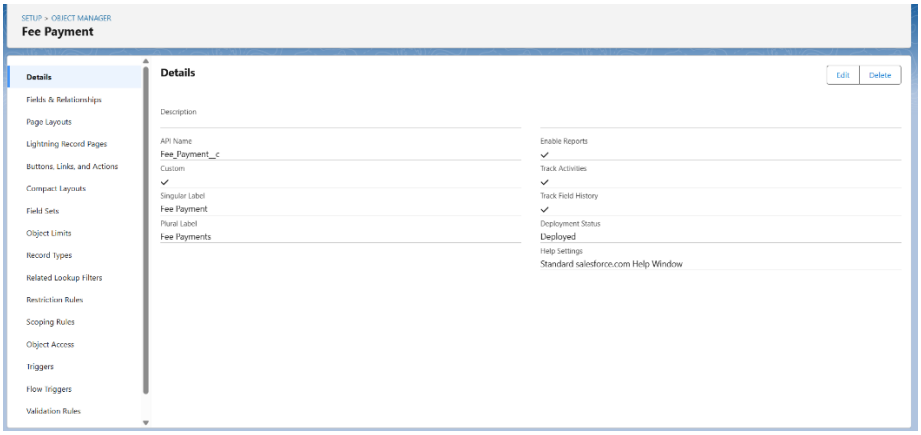
3. **Course** (Course__c)
    - Record Name: *Course Code* — Text (e.g., CSE101).



4. **Enrollment** (Enrollment__c) — **Junction object** between Student & Course
    - Record Name: *Enrollment Number* — Auto Number ENR-{0000}.



5. **Fee Payment** (Fee_Payment__c)
    - Record Name: *Payment Number* — Auto Number PAY-{0000}.



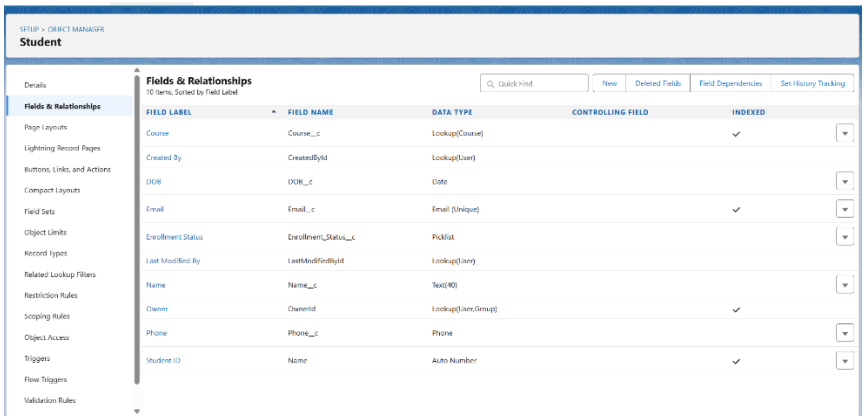
After creating each object, **create a custom tab** for it (Setup → Tabs → New → Custom Object Tab).

2 — Add custom fields (Object Manager → [Object] → Fields & Relationships → New)

Create these fields (pick types exactly):

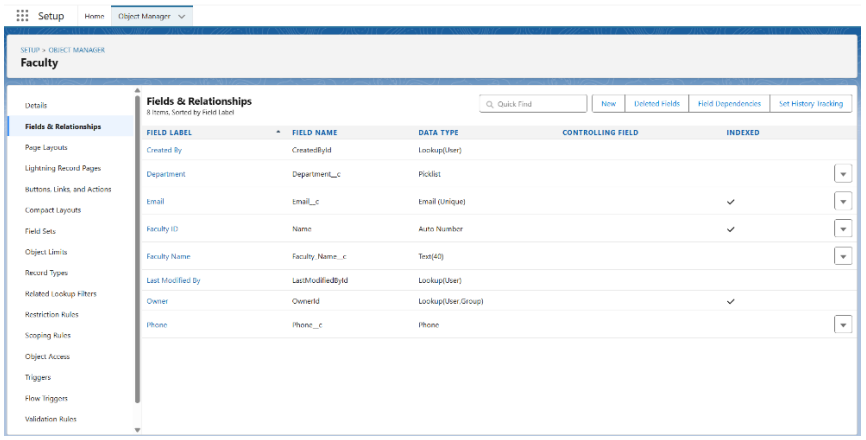
Student (Student__c)

- Full_Name__c — Text (255) or use standard Name field.
- Email__c — Email.
- Phone__c — Phone.
- DOB__c — Date.
- Enrollment_Status__c — Picklist: *Admitted, Active, Completed, Dropped*.
- Primary_Course__c — Lookup (Course__c) — optional quick link.



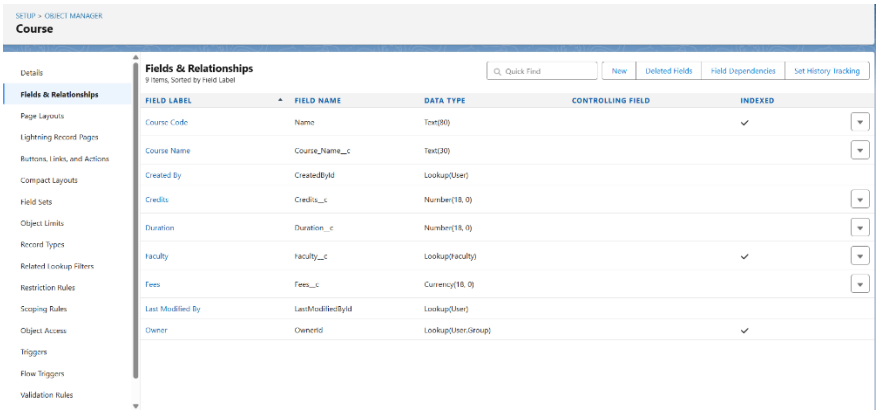
Faculty (Faculty__c)

- Full_Name__c — Text (or standard Name).
- Email__c — Email.
- Phone__c — Phone.
- Department__c — Picklist: *Science, Commerce, Arts, Tech, Other*.



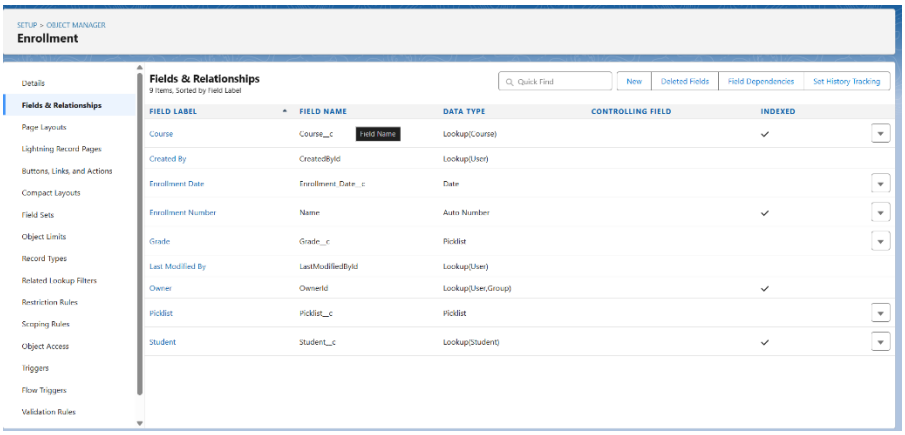
Course (Course__c)

- Course_Name__c — Text.
- Duration_Months__c — Number (0, no decimals).
- Credits__c — Number.
- Fees__c — Currency.
- Faculty__c — Lookup (Faculty__c).



Enrollment (Enrollment__c) — junction fields

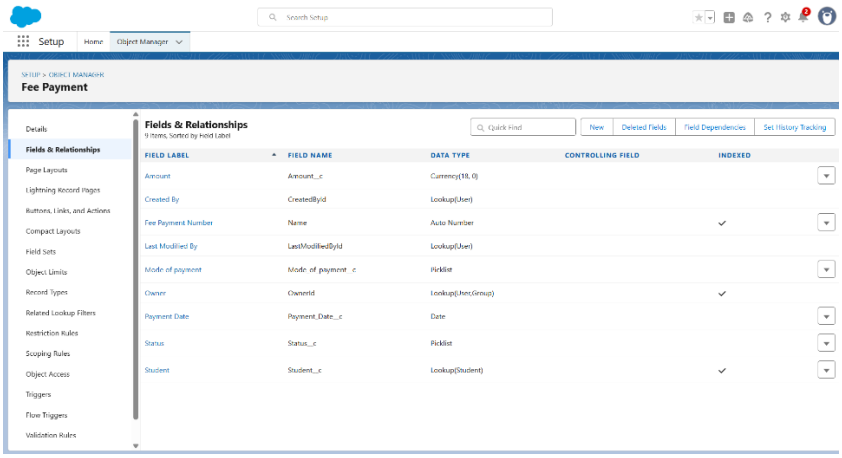
- Student__c — **Master-Detail (Student__c)** or Lookup (see Relationship section).
- Course__c — **Master-Detail (Course__c)** or Lookup.
- Enrollment_Date__c — Date.
- Status__c — Picklist: *Active, Completed, Dropped*.
- Grade__c — Picklist: *A, B, C, D, F*.



Fee Payment (Fee_Payment__c)

- Student__c — Lookup (Student__c).
- Payment_Date__c — Date.

- Amount__c — Currency.
- Payment_Mode__c — Picklist: *Cash, Card, Online*.
- Status__c — Picklist: *Paid, Pending*.

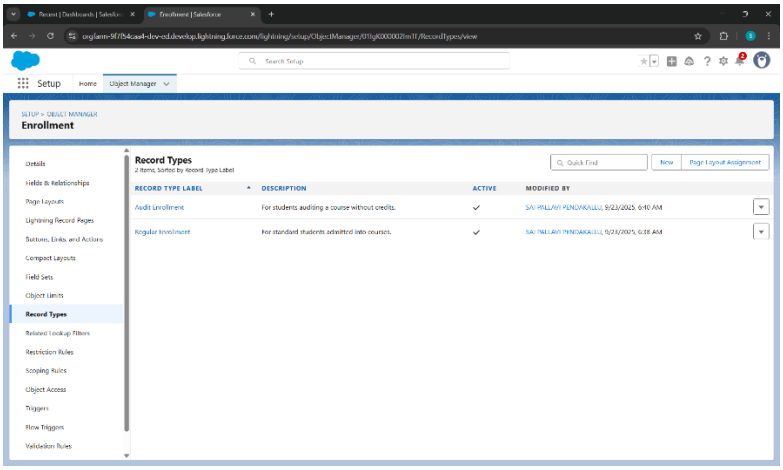


---

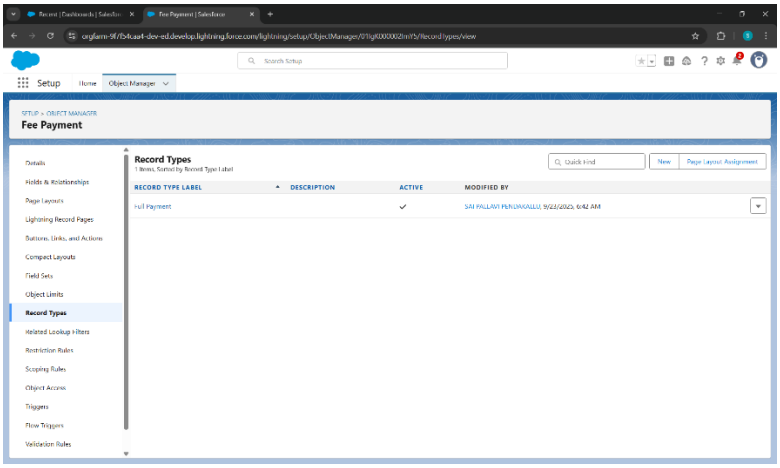
3 — Create Record Types (Setup → Object Manager → [Object] → Record Types → New)

Create record types where behavior differs:

- **Enrollment**:
  - Regular Enrollment — default page layout.
  - Audit Enrollment — different required fields or layouts.



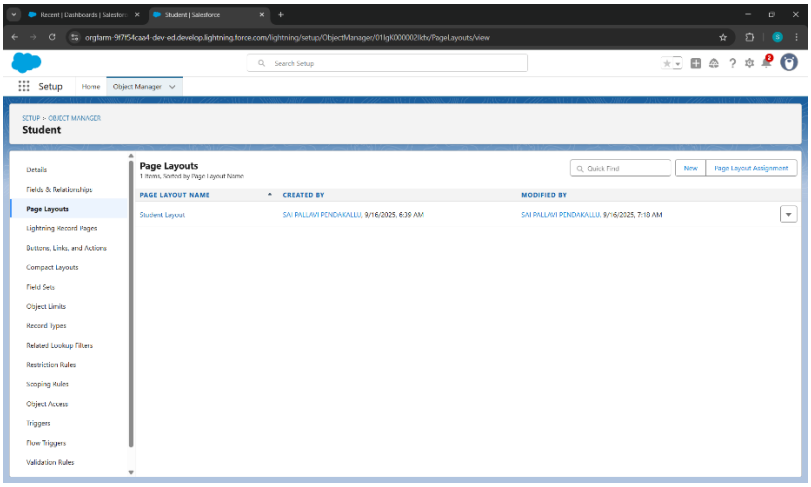
- **Fee Payment**:
    - Full Payment



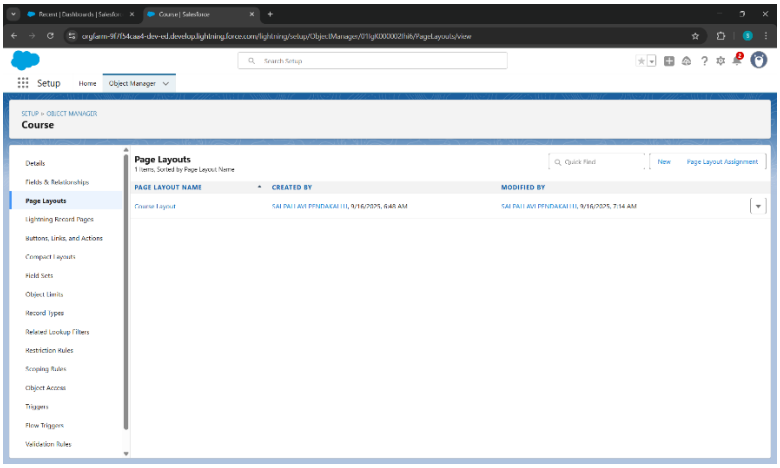
4— Page Layouts & Related Lists (Object Manager → Page Layouts)

Design UX for each object:

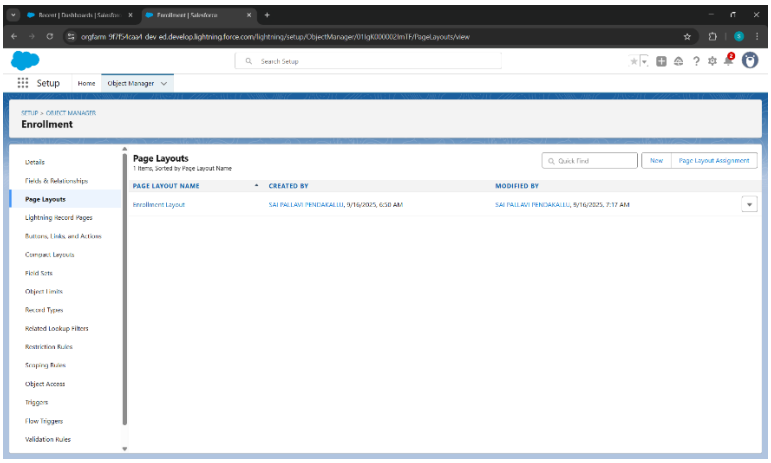
- **Student Page Layout**: put key fields top (Full Name, Email, DOB, Enrollment Status). Add related lists: **Enrollments**, **Fee Payments**. Place Student 360 components (if Lightning) — related records, quick actions.



- **Course Page Layout**: Course fields + related list **Enrollments** (show Students). Show Lookup to Faculty.



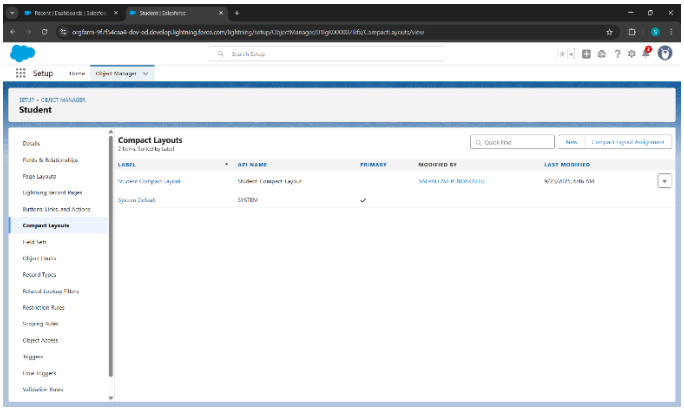
- **Enrollment Page Layout**: show Student & Course in top; Status, Enrollment Date, Grade.



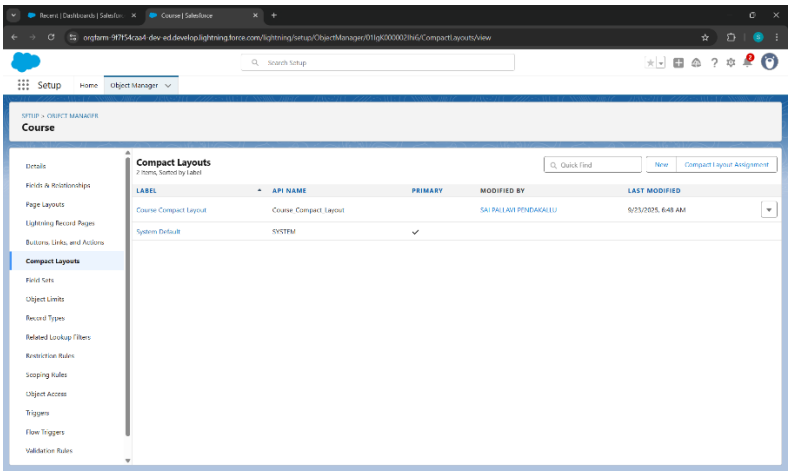
5— Compact Layouts (for mobile) (Object Manager → Compact Layouts)

Create small summaries for mobile:

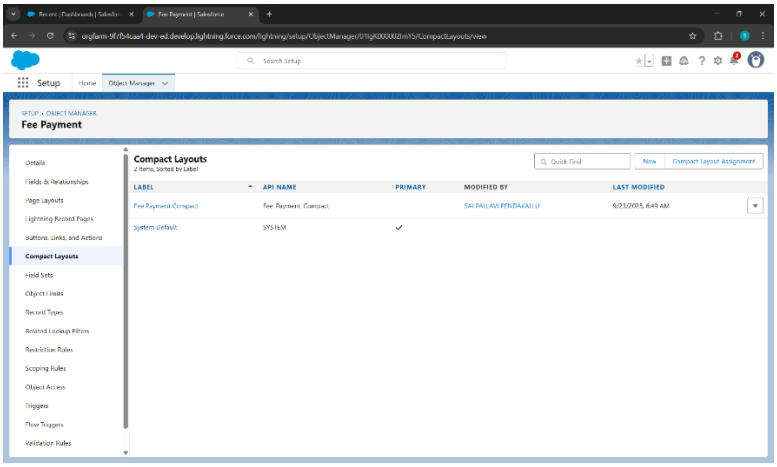
- **Student Compact Layout**: Full_Name__c, Email__c, Enrollment_Status__c.



- **Course Compact Layout**: Course_Name__c, Duration_Months__c, Fees__c.

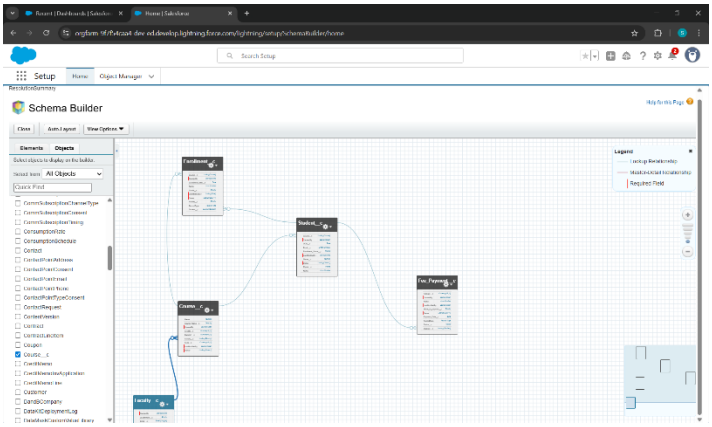


- **Fee Payment Compact**: Payment_Date__c, Amount__c, Status__c.



6— Schema Builder (Setup → Schema Builder)

1. Open **Schema Builder**.
2. From left pane, tick your custom objects to show them on canvas.
3. Drag fields or relationships if needed, or simply **visualize** Student ↔ Enrollment ↔ Course and Course ↔ Faculty, Student ↔ Fee Payment.
4. You can create new fields/relationships here by dragging, but I usually use Object Manager for precise control.

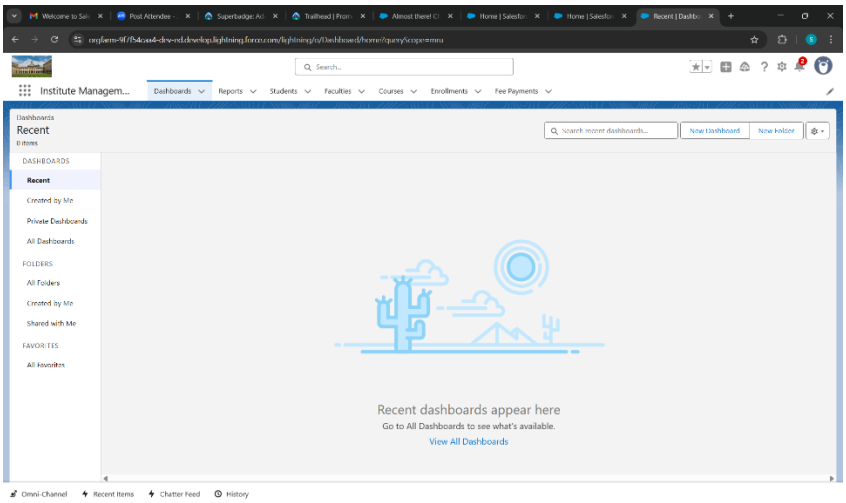


7— Tabs & Lightning App (Setup → App Manager → New Lightning App)

Tabs: (Setup → Tabs) create custom tabs for Students, Faculty, Courses, Enrollments, Fee Payments.

Lightning App:

1. Setup → App Manager → New Lightning App.
2. Name: *Institute Management System*.
3. Branding: choose color/logo.
4. Navigation Items: add the tabs you created + Reports, Dashboards.
5. Utility Bar: add **Notes**, **History**, **Chatter**, **Recent Items** (useful for IMS).
6. Assign to Profiles (Admin, Faculty, Accounts, Student). Save.



INSTITUTE MANAGEMENT SYSTEM

Phase 4: Process Automation

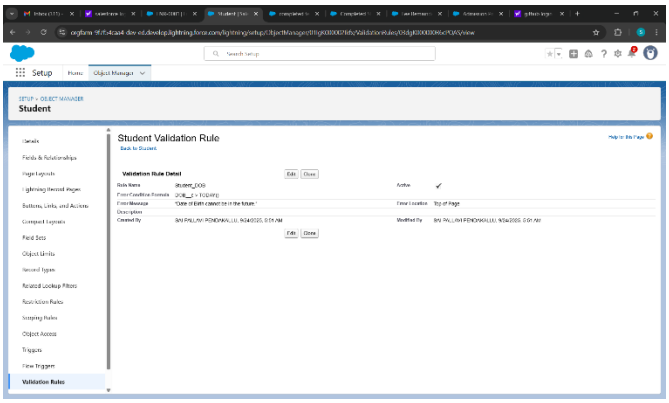
Goal: Automate repetitive or critical tasks for students, enrollments, courses, and fee payments.

Step 1: Create Validation Rules

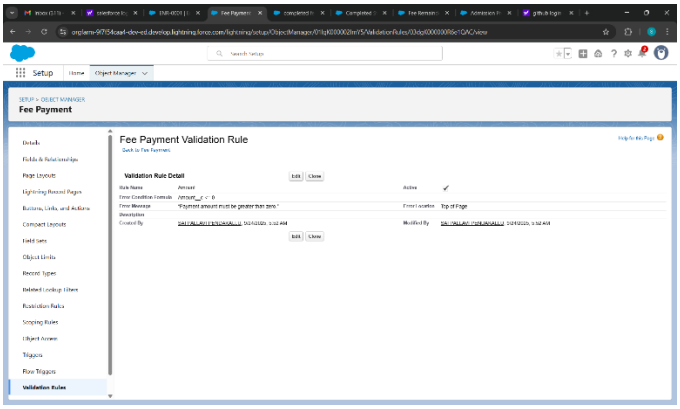
Validation rules ensure data integrity.

Examples for IMS:

- **Student DOB cannot be in the future:**
  - Formula: DOB__c > TODAY()
  - Error Message: *"Date of Birth cannot be in the future."*



- **Fee Payment Amount > 0:**
  - Formula: Amount__c <= 0
  - Error: *"Payment amount must be greater than zero."*

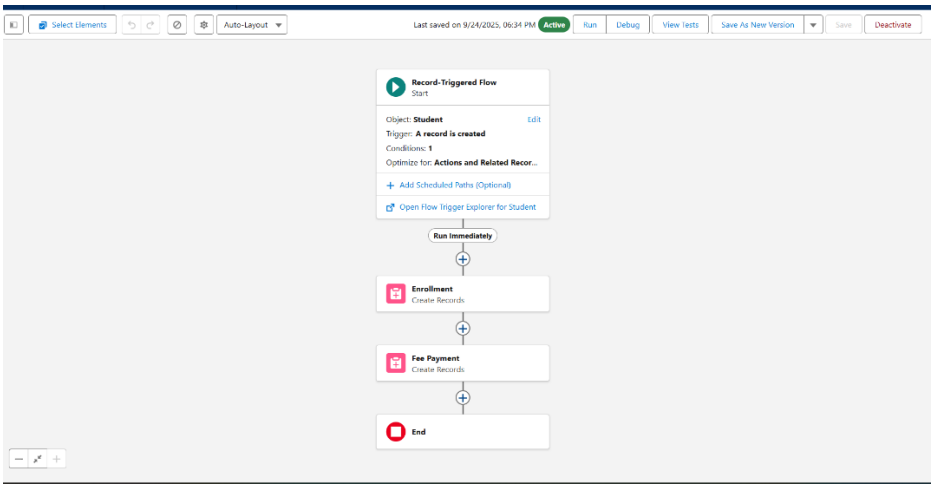


Step 2: Create Record-Triggered Flow

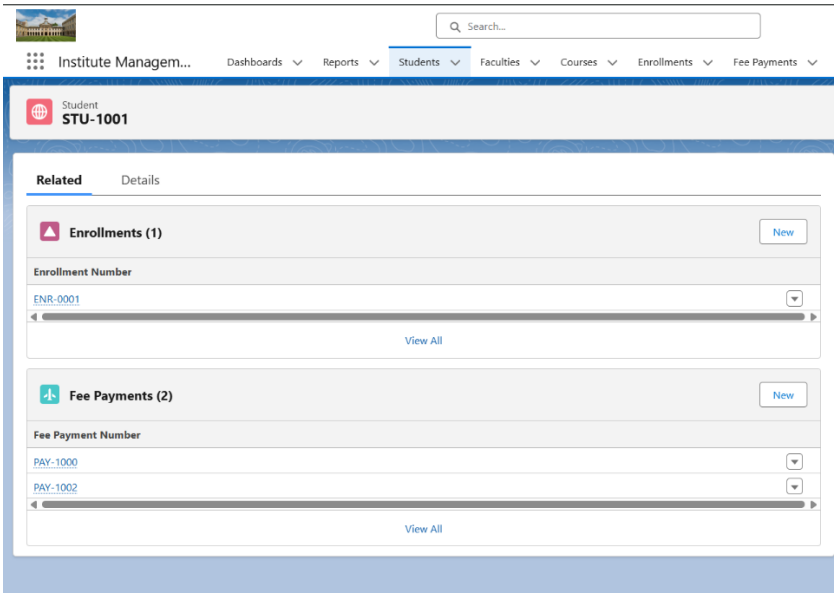
Goal: Automate Enrollment and Fee Payment creation when a Student is admitted.

1. Go to **Setup → Flow → New Flow → Record-Triggered Flow**.
2. **Select Object:** Student
3. **Trigger:** When a record is created or updated
4. **Entry Condition:** Enrollment_Status__c = 'Admitted'
5. **Actions inside the Flow:**
    - **Create Enrollment Record:**
        - Link to Student (Student__c)
        - Assign Course__c = Primary_Course__c
        - Set Enrollment_Date__c = TODAY()
        - Set Status__c = Active
    - **Create Fee Payment Record:**
        - Link to Student (Student__c)
        - Set Payment_Date__c = TODAY()
        - Set Amount__c = Course__c.Fees__c
        - Set Status__c = Pending
6. Save and activate the flow.

✅ Now when a student is admitted, Enrollment and Fee Payment are automatically created.



Outcome for the above flow is shown as below,

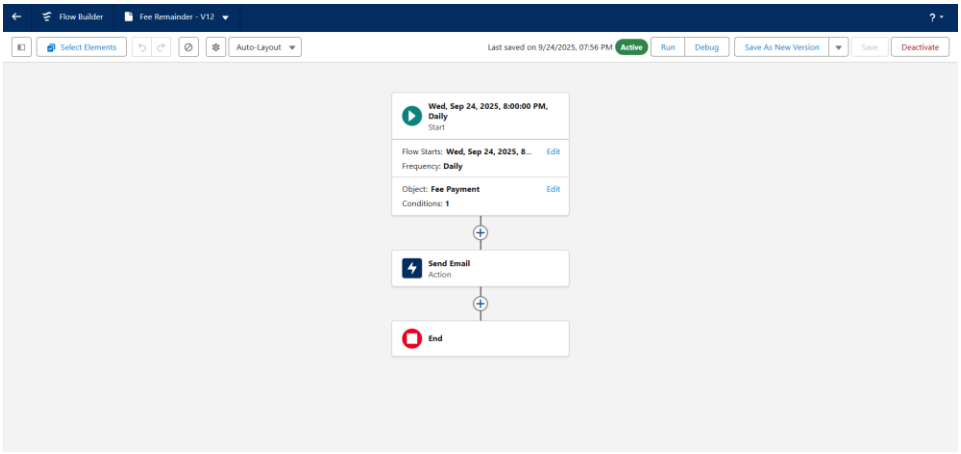


Step 3: Scheduled Flow – Daily Fee Reminders

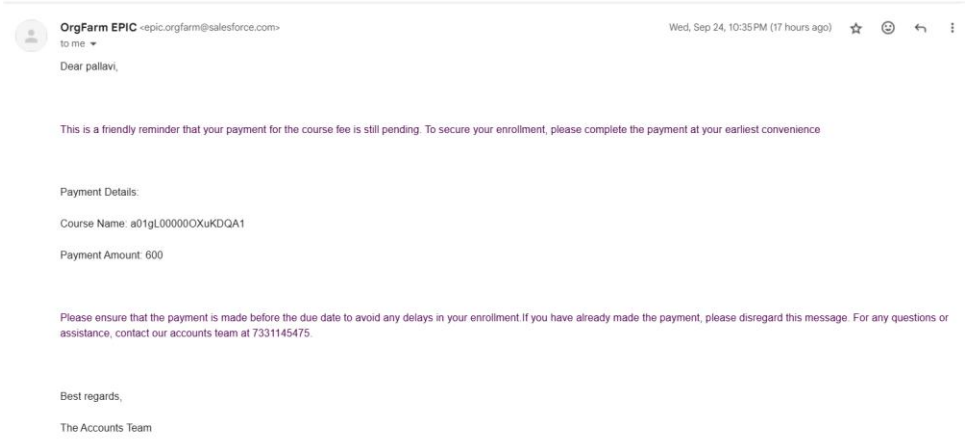
Objective:
Ensure students are reminded each day about any outstanding fee payments.

Setup Instructions:

- Go to **Setup → Flows → New Flow**.
- Choose **Scheduled-Triggered Flow** as the flow type.
- Define the schedule:
    - Run **every day**.
    - Set the start time to **4:30 PM**.
- Select the **Fee_Payment__c** object as the record source.
- Apply a filter where **Status__c = "Pending"**.
- Add an action element:
    - Type: **Send Email Alert**.
    - Recipient: **Student__c.Email**.
- Save the flow with the label **Fee Reminder Flow**.
- **Activate** the flow.



Result:
Students who still have fees marked as pending will automatically receive a daily reminder email, helping to promote timely payments.



Step 4: Record-Triggered Flow – Enrollment Status Update

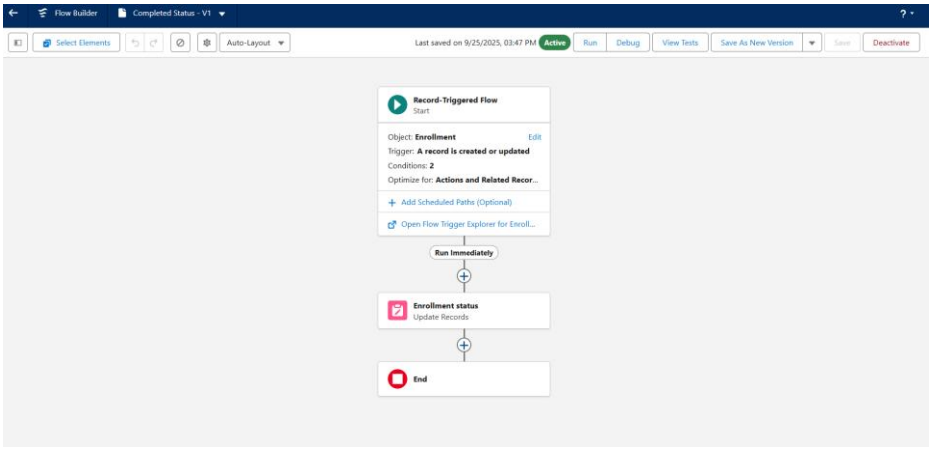
Objective:
Automatically change the enrollment status to **"Completed"** once a grade is provided for a student.

Setup Instructions:

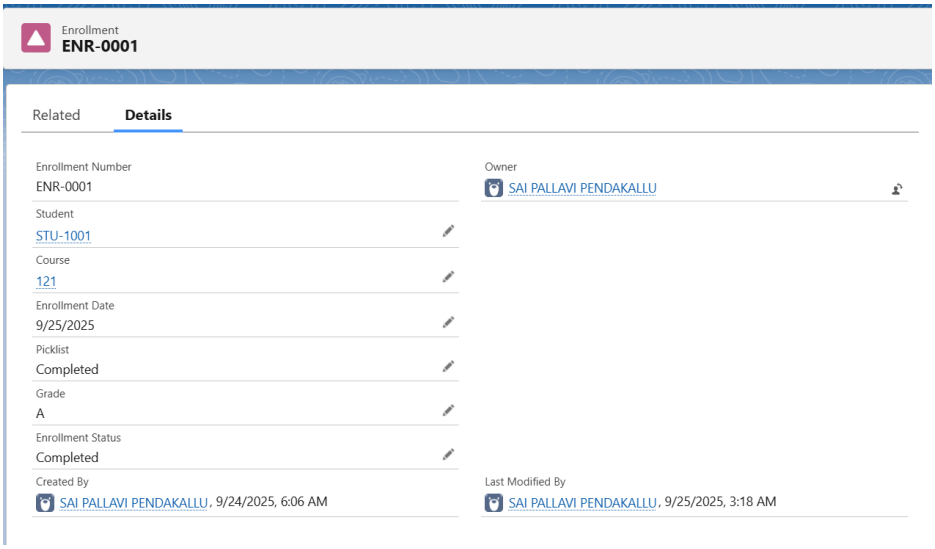
- Go to **Setup → Flows → New Flow**.
- Select **Record-Triggered Flow**.
- Choose **Enrollment__c** as the object.
- Set the trigger to run **when a record is created or updated**.
- Define entry conditions:
    - Grade__c has been modified.

- Grade__c is not blank.

- Optimize the flow for **Actions and Related Records**.
- Add an **Update Records** element:
  - Update the same **Enrollment__c** record.
  - Assign Enrollment_Status__c = "Completed".
- Save the flow with the label **Status Completed**.
- **Activate** the flow.



Result:
Whenever a grade is entered or updated for an enrollment, the system will automatically mark that enrollment as **Completed**, ensuring statuses stay accurate without manual updates.

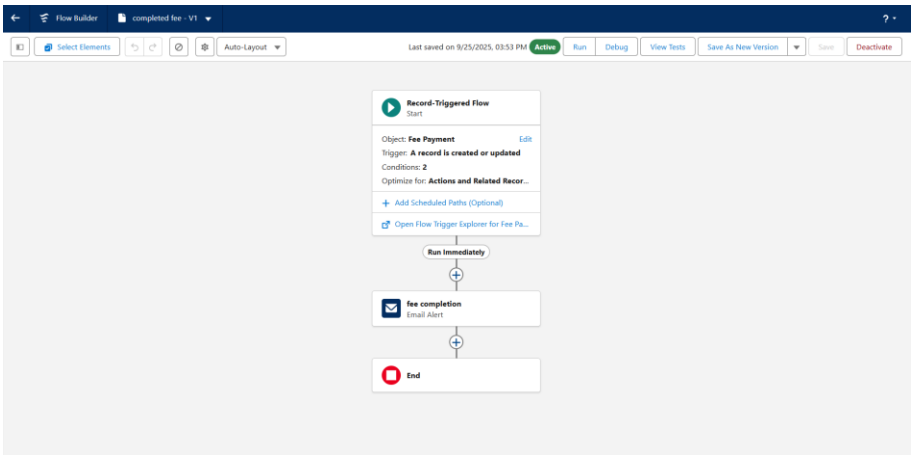


Step 5: Record-Triggered Flow – Fee Payment Status Update

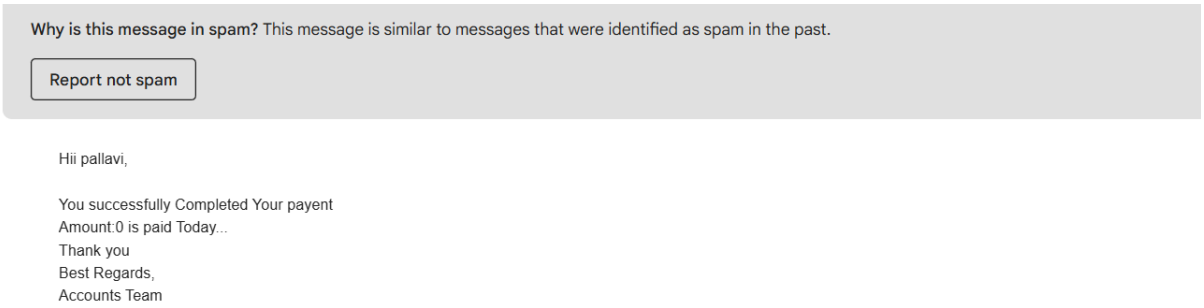
Objective:
Ensure that whenever a payment is made, the fee status is automatically updated to **"Paid"** and the student is notified.

Setup Instructions:

- Navigate to **Setup → Flows → New Flow**.
- Select **Record-Triggered Flow**.
- Pick **Fee_Payment__c** as the object.
- Configure the trigger to run **when a record is created or updated**.
- Define conditions:
    - Amount__c is greater than 0.
    - Status__c is not equal to "Paid".
- Add an **Update Records** element:
    - Update the same **Fee_Payment__c** record.
    - Set Status__c = "Paid".
- Add an **Email Alert** action:
    - Send an email to the student confirming payment received.
- Save the flow with the label **Fee Status Updated**.
- **Activate** the flow.



Result:
Whenever a payment amount is entered, the fee record automatically changes to **Paid**, and the student receives an email confirmation. This keeps financial records accurate and ensures prompt communication.



Step 6: Record-Triggered Flow – Enrollment Email Notification

Objective:
Automatically send an email to students when their enrollment status is set to **"Active"**, ensuring they are informed promptly about their successful enrollment.

Setup Instructions:

- Navigate to **Setup → Flows → New Flow**.
- Select **Record-Triggered Flow**.
- Choose **Enrollment__c** as the object.
- Set the trigger to run **when a record is created or updated**.
- Define entry conditions:
    - Status__c = "Active".
- Add an **Email Alert** element:
    - Send a notification to the student confirming their enrollment.
- Save the flow with the label **Enroll Students**.
- **Activate** the flow.

Result:
Whenever an enrollment record is created or updated with the status **Active**, the student automatically receives a confirmation email. This automation improves communication efficiency and eliminates the need for manual follow-ups.

Why is this message in spam? This message is similar to messages that were identified as spam in the past.

Report not spam

Hello pallavi,

Your enrollment in Electronics is confirmed.

Course Details:
Course Name: Electronics
Course Code: SC-0001

Instructor is already assigned to your course.

Please make sure to complete any pre-course requirements.

We look forward to seeing you in the course!

Best regards,
Accounts Team

INSTITUTE MANAGEMENT SYSTEM

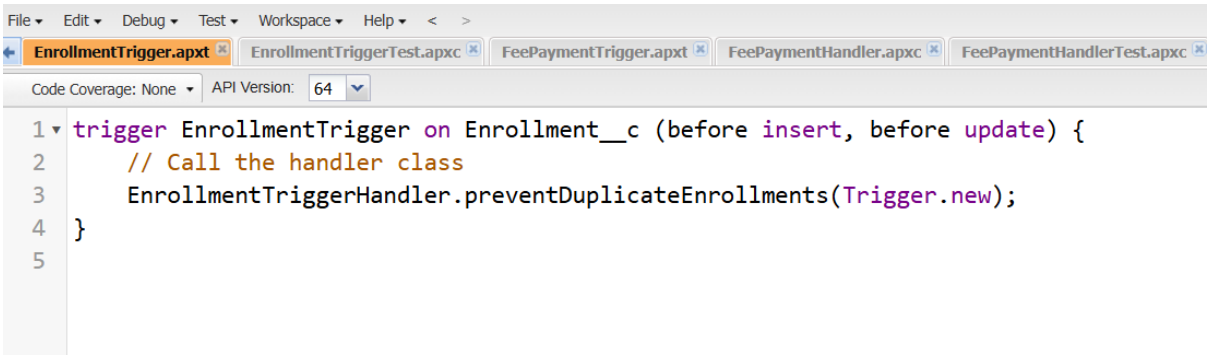
Phase 5: Apex Programming

Apex classes and Triggers:

📌 Enrollment Management

1. EnrollmentTrigger (Trigger)

Purpose:
Fires whenever a student enrollment record (Enrollment__c) is created or updated. Its only job is to call the EnrollmentTriggerHandler so the logic is not written directly in the trigger (best practice).



2. EnrollmentTriggerHandler (Handler Class)

Purpose:
Ensures that a student cannot be enrolled in the same course twice.

- Collects Student–Course combinations.
- Checks database for existing enrollments.
- Throws an error (addError) if a duplicate is found.

```
public class EnrollmentTriggerHandler {

    public static void preventDuplicateEnrollments(List<Enrollment__c> enrollments) {
        // Set to store new Student-Course combinations
        Set<String> newEnrollmentsSet = new Set<String>();

        // Sets to collect unique Student and Course IDs from the incoming records
        Set<Id> studentIds = new Set<Id>();
        Set<Id> courseIds = new Set<Id>();

        for (Enrollment__c e : enrollments) {
            if(e.Student__c != null && e.Course__c != null){
                studentIds.add(e.Student__c);
                courseIds.add(e.Course__c);

                // Also track combination of student-course in new batch
                newEnrollmentsSet.add(e.Student__c + '-' + e.Course__c);
            }
        }

        // Query existing enrollments in database
        Set<String> existingEnrollmentsSet = new Set<String>();
        for (Enrollment__c e : [
            SELECT Id, Student__c, Course__c
            FROM Enrollment__c
            WHERE Student__c IN :studentIds
              AND Course__c IN :courseIds
        ]) {
            existingEnrollmentsSet.add(e.Student__c + '-' + e.Course__c);
        }
```

```
        // Compare and throw error if duplicate found
        for (Enrollment__c e : enrollments) {
            if(e.Student__c != null && e.Course__c != null){
                String key = e.Student__c + '-' + e.Course__c;
                if(existingEnrollmentsSet.contains(key)){
                    e.addError('This student is already enrolled in the selected course.');
                }
            }
        }
    }
}
```

---

3. EnrollmentTriggerTest (Test Class)

Purpose:

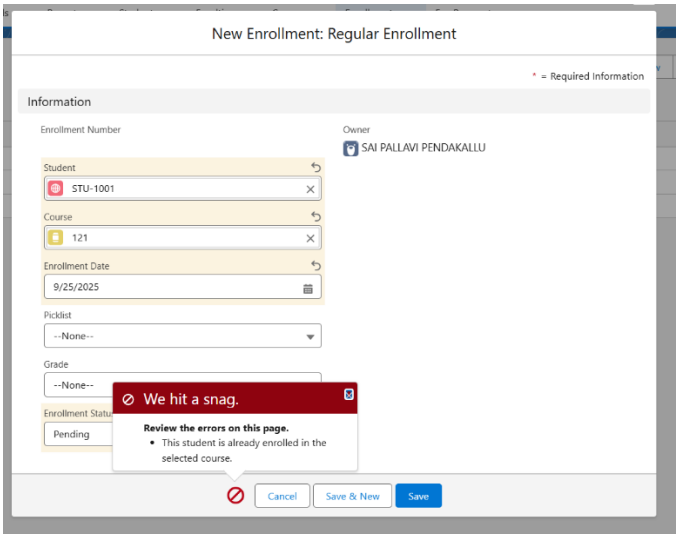
Verifies that the duplicate enrollment prevention logic works correctly.

- Inserts a valid enrollment (success).
- Tries inserting a duplicate (fails with error).
- Confirms the trigger behavior with assertions.

```apex
@IsTest
public class EnrollmentTriggerTest {
    @IsTest
    static void testPreventDuplicateEnrollments() {
        // Create sample student and course
        Student__c student = new Student__c(
            Name__c = 'Test Student',
            Email__c = 'test@student.com'
        );
        insert student;
        Course__c course = new Course__c(
            Course_Name__c = 'Test Course',
            Duration__c = 6
        );
        insert course;
        // Insert first enrollment (should succeed)
        Enrollment__c e1 = new Enrollment__c(
            Student__c = student.Id,
            Course__c = course.Id
        );
        insert e1;
        // Attempt duplicate enrollment (should fail)
        Enrollment__c e2 = new Enrollment__c(
            Student__c = student.Id,
            Course__c = course.Id
        );
        try {
            insert e2;
            System.assert(false, 'Expected duplicate enrollment error.');
        } catch (DmlException ex) {
            System.assert(ex.getMessage().contains('This student is already enrolled'));
```

OUTCOME:

- **Prevents Duplicate Enrollments**
  - A student cannot be enrolled in the same course more than once.
  - If attempted, Salesforce shows an error message: *"This student is already enrolled in the selected course."*

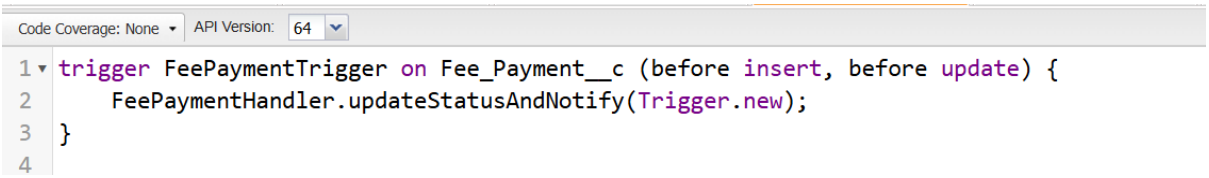


📌 Fee Payment Management

1. FeePaymentTrigger (Trigger)

Purpose:
Runs before insert or update of a Fee_Payment__c record.

- Calls FeePaymentHandler to update fee status and notify students.



---

2. FeePaymentHandler (Handler Class)

Purpose:
Automatically updates the fee payment status and sends confirmation.

- If Amount > 0 and Status != "Paid", update Status → "Paid".
- Sends an email to the student confirming payment.
- Bulk-safe for multiple payments at once.

---

3. FeePaymentHandlerTest (Test Class)

Purpose:
Confirms fee payments are marked as "Paid" correctly.

- Inserts a fee record with Pending status.
- Checks if trigger updates it to Paid.

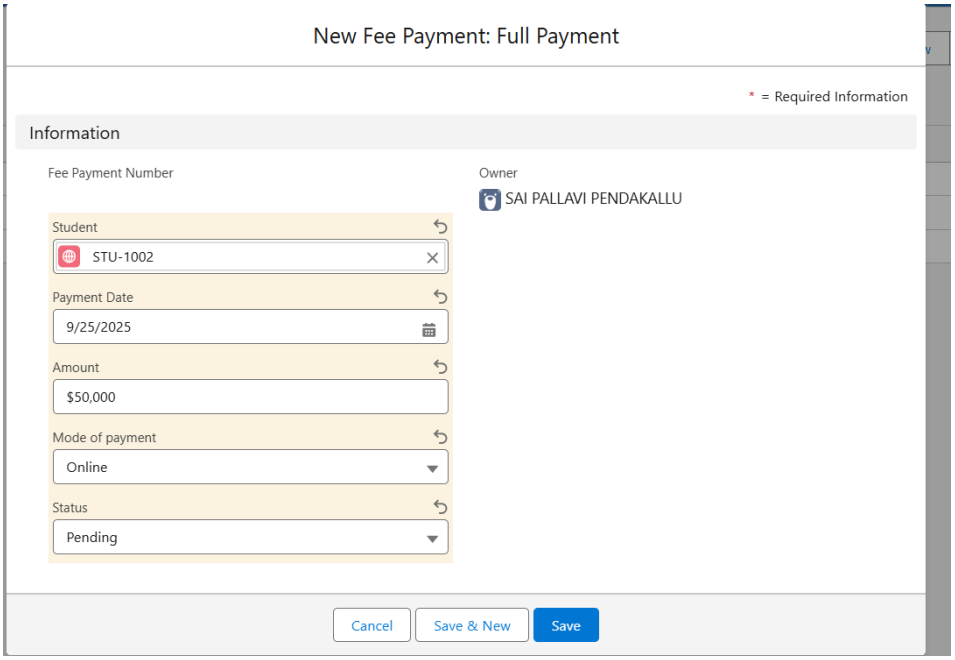
- Covers email notification logic in a safe way (emails not actually sent in test).

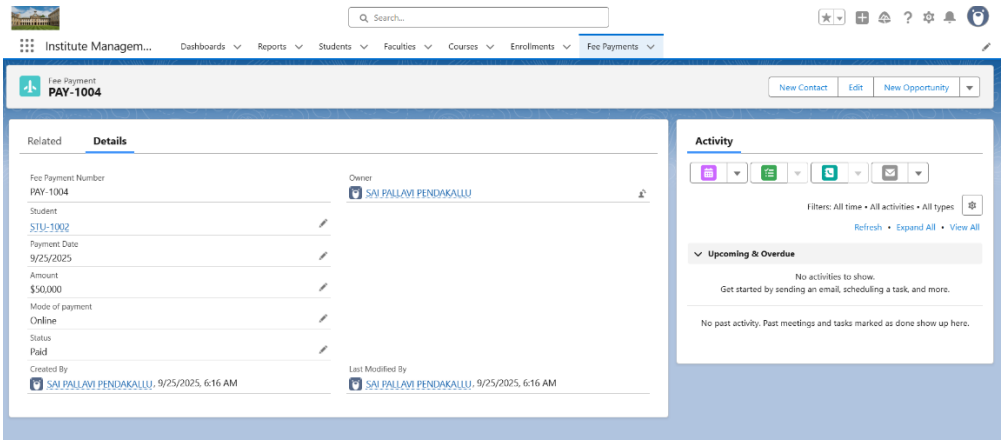
```
@IsTest
public class FeePaymentHandlerTest {
    @IsTest
    static void testFeePaymentStatusUpdate() {
        // Create sample student
        Student__c stu = new Student__c(Name__c='Test Student', Email__c='test@student.com');
        insert stu;

        // Insert payment
        Fee_Payment__c payment = new Fee_Payment__c(
            Student__c = stu.Id,
            Amount__c = 500,
            Status__c = 'Pending'
        );
        insert payment;

        // Verify status updated
        Fee_Payment__c fp = [SELECT Status__c FROM Fee_Payment__c WHERE Id = :payment.Id];
        System.assertEquals('Paid', fp.Status__c);
    }
}
```

Outcome:

□ Tracks Student Payments Accurately

- Each payment is linked to a student record.
- Captures amount, status (Pending, Paid, Overdue), and payment date.





 Prevents Inconsistencies

- Triggers and validation rules ensure no negative amounts or invalid payment entries.
- Stops duplicate or incomplete payment records.

📌 Enrollment Status Update

1. EnrollmentStatusTrigger (Trigger)

Purpose:
Fires when enrollment records are inserted or updated.

- Calls EnrollmentStatusHandler to check if a grade is entered.

```
trigger EnrollmentStatusTrigger on Enrollment__c (before insert, before update) {
    EnrollmentStatusHandler.updateStatusOnGrade(Trigger.new, Trigger.oldMap);
}
```

---

2. EnrollmentStatusHandler (Handler Class)

Purpose:
Automatically updates enrollment status based on grades.

- If Grade__c is filled in, update Enrollment_Status__c → "Completed".
- Ensures enrollments are closed out automatically when results are entered.

```
public class EnrollmentStatusHandler {
    public static void updateStatusOnGrade(List<Enrollment__c> newEnrollments, Map<Id, Enrollment__c> oldMap) {
        for(Enrollment__c e : newEnrollments) {
            // Check if grade is entered or updated
            if(e.Grade__c != null && (oldMap == null || e.Grade__c != oldMap.get(e.Id).Grade__c)) {
                e.Enrollment_Status__c = 'Completed';
            }
        }
    }
}
```

3. EnrollmentStatusHandlerTest (Test Class)

Purpose:
Tests that enrollment status auto-updates.

- Creates an Active enrollment.
- Updates with a grade.
- **Asserts status becomes "Completed".**

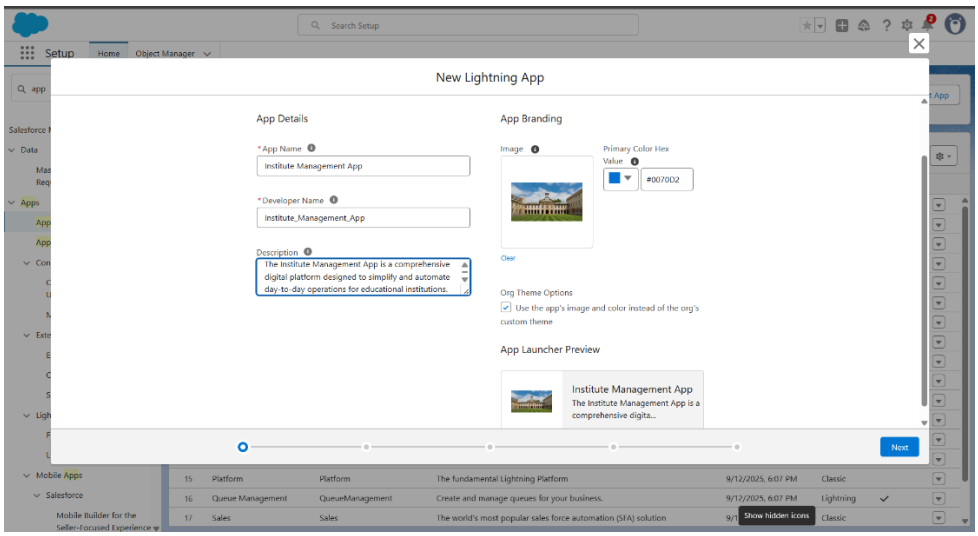
```
@IsTest
public class EnrollmentStatusHandlerTest {
    @IsTest
    static void testStatusUpdate() {
        // Create student and course
        Student__c stu = new Student__c(Name__c='Test Student', Email__c='test@student.com');
        insert stu;
        Course__c course = new Course__c(Course_Name__c='Test Course');
        insert course;

        Enrollment__c enrollment = new Enrollment__c(
            Student__c = stu.Id,
            Course__c = course.Id,
            Enrollment_Status__c = 'Active'
        );
        insert enrollment;

        // Update grade
        enrollment.Grade__c = 'A';
        update enrollment;

        Enrollment__c updated = [SELECT Enrollment_Status__c FROM Enrollment__c WHERE Id = :enrollment.Id];
        System.assertEquals('Completed', updated.Enrollment_Status__c);
    }
}
```

INSTITUTE MANAGEMENT SYSTEM

Phase 6: User Interface Development

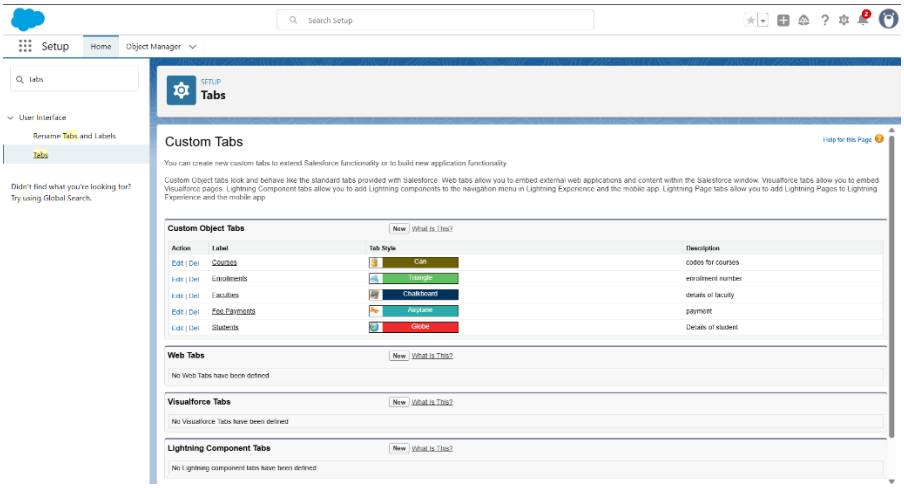
A. Create the Lightning App

1. Setup → App Manager → New Lightning App.
2. Name: Institute Management. Add branding (logo/color).
3. Navigation Items: add custom object tabs (Students, Courses, Enrollments, Fee Payments) + Reports + Dashboards.
4. Utility Bar: click Add, include actions like New Enrollment, New Fee Payment, Recent Items, Notes.
5. Assign the app to Profiles (Admin, Faculty, Accountant). Save.



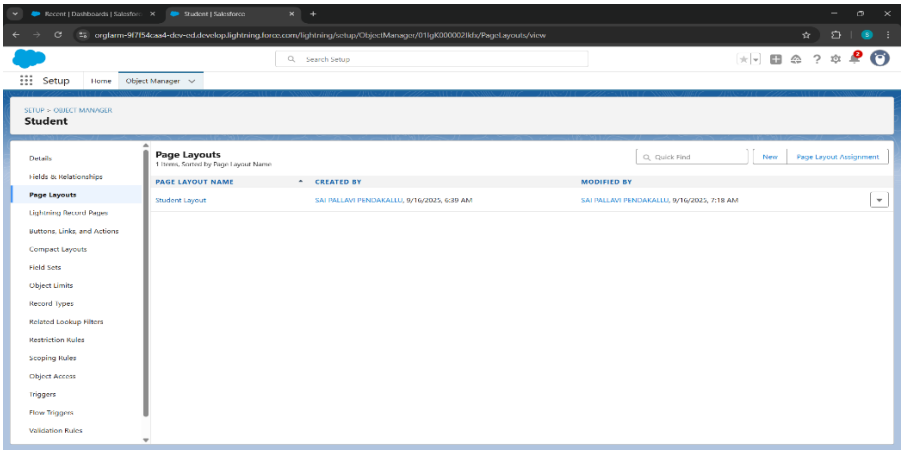
B. Create Tabs for Objects and LWC

1. Setup → Tabs → New → Custom Object Tab: create tabs for Student__c, Course__c, Enrollment__c, Fee_Payment__c.
2. For LWC components you want as standalone tabs: Setup → Tabs → Lightning Component Tabs → New → pick your LWC and label it (done after deploying LWC).

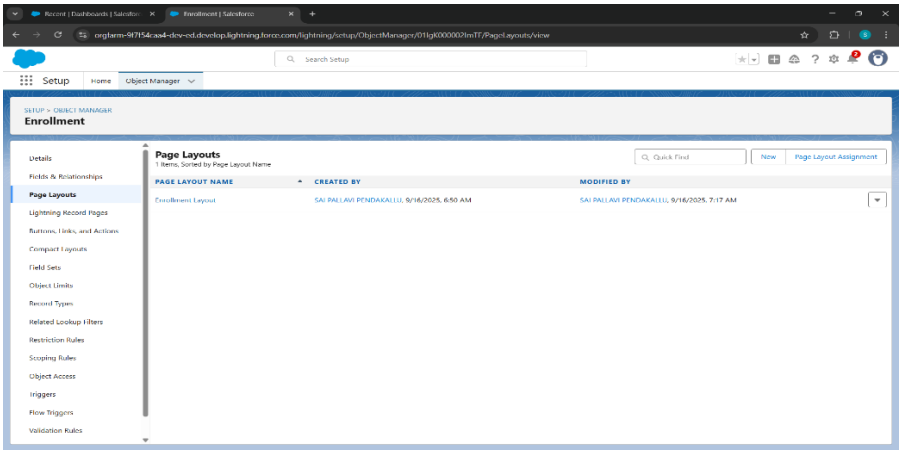


C. Page Layouts & Quick Actions

1. Setup → Object Manager → open Student__c → Page Layouts → Edit layout → drag Related Lists (Enrollments, Fee Payments). Save.

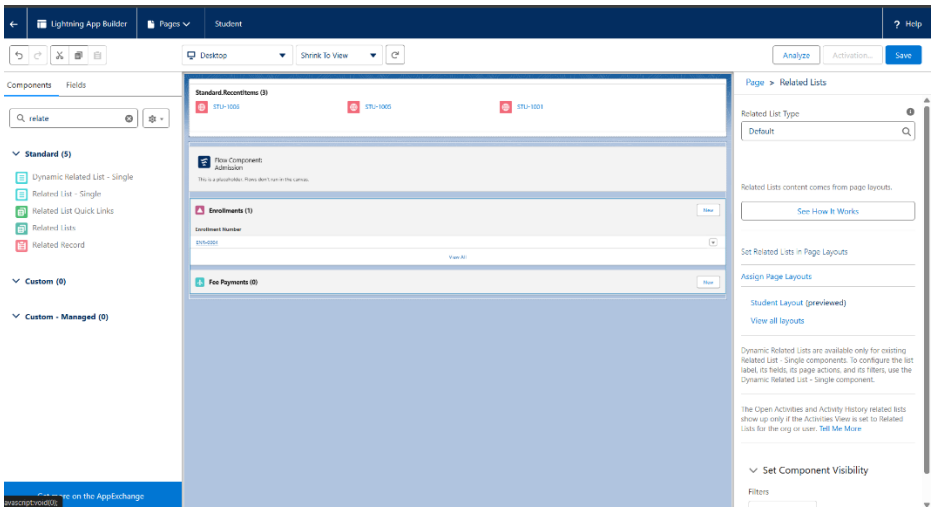


2. Setup → Object Manager → Enrollment__c → Page Layouts → add Submit for Approval or custom quick actions if needed.



D. Build Record Pages (Lightning App Builder)

1. App Launcher → open App Builder: Setup → Lightning App Builder → New → select Record Page.
2. Choose object (e.g., Student) → choose template (Header & Right Sidebar) → name (Student Record Page).
3. Drag standard components and your custom LWC (once deployed) onto the page: Related lists, Highlights, Quick Actions.
4. Click Activation → make it the org default for the object or assign by app/profile/record type. Save & Activate.

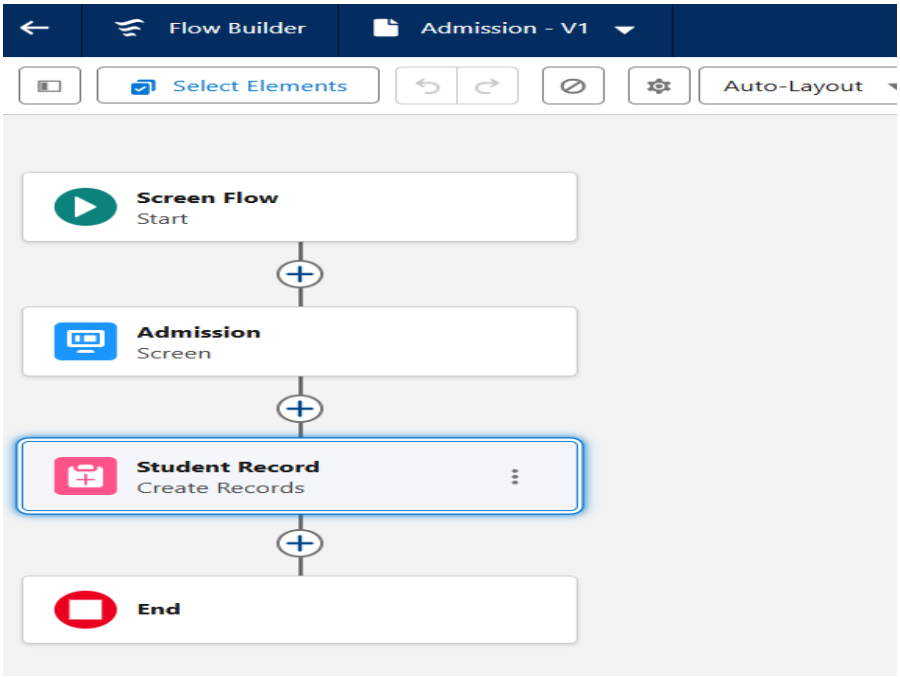


E. Screen Flow – Student Admission Form

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.

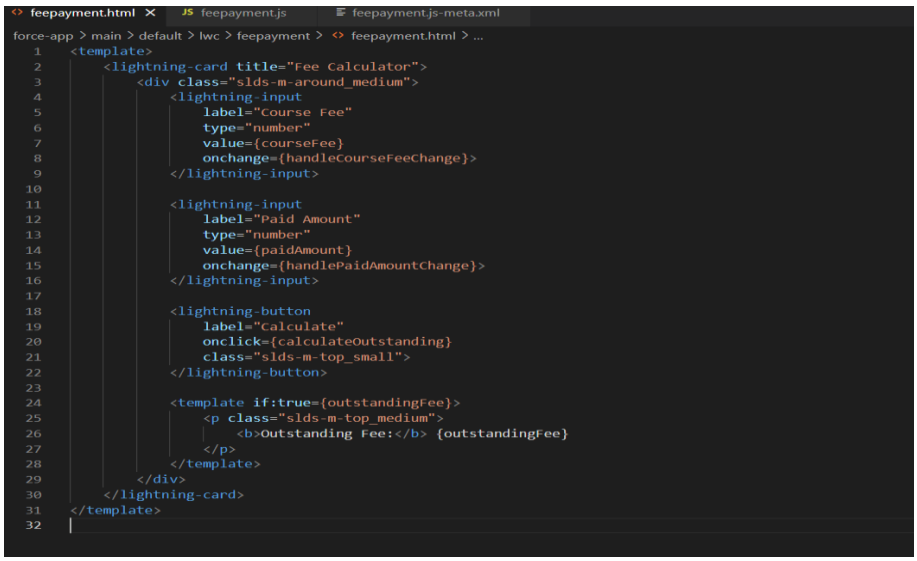


Lightning Web Component – Fee Calculator

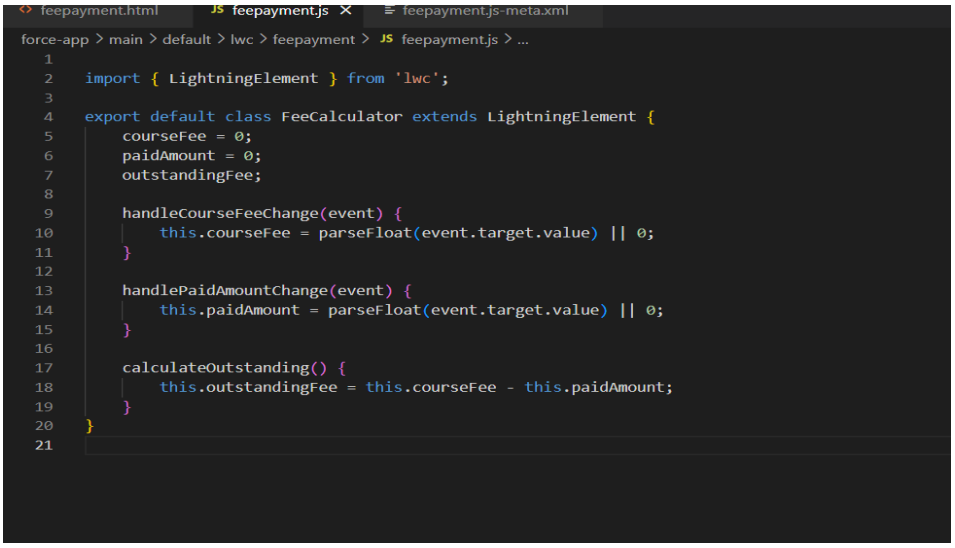
Purpose: Allow quick calculation of fees (e.g., Course Fee – Paid Amount = Outstanding Fee).

LWC Development Steps:

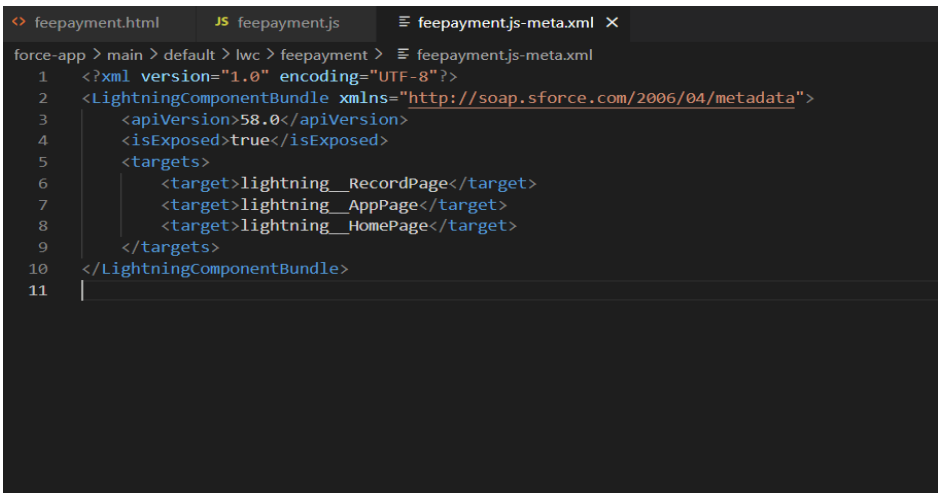
1. Create Project (in VS Code Command Palette):
   SFDX: Create Project with Manifest
2. Create LWC:
   SFDX: Create Lightning Web Component
   Name → feepayment
3. Edit Files:
   - feepayment.html → Create form (inputs for Course Fee & Paid Fee).



feepayment.js → Logic to calculate outstanding fee.

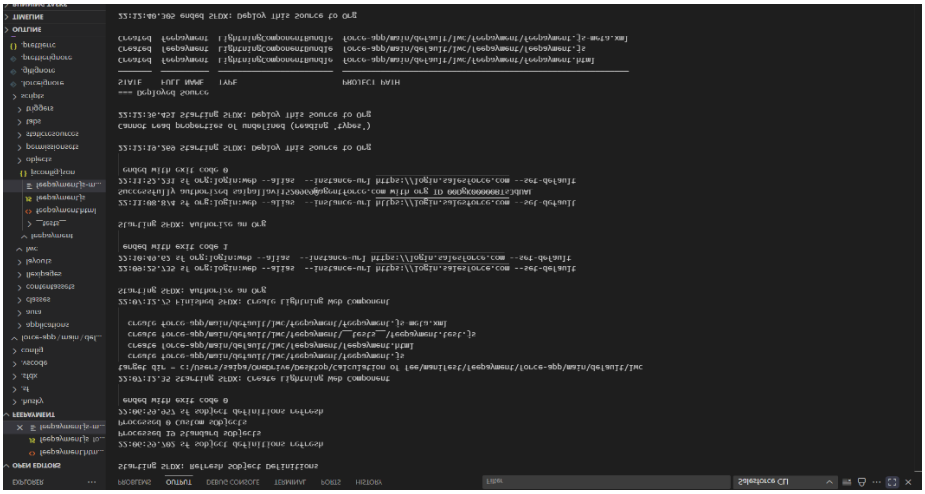


feepayment.js-meta.xml → Expose component to Record Page.

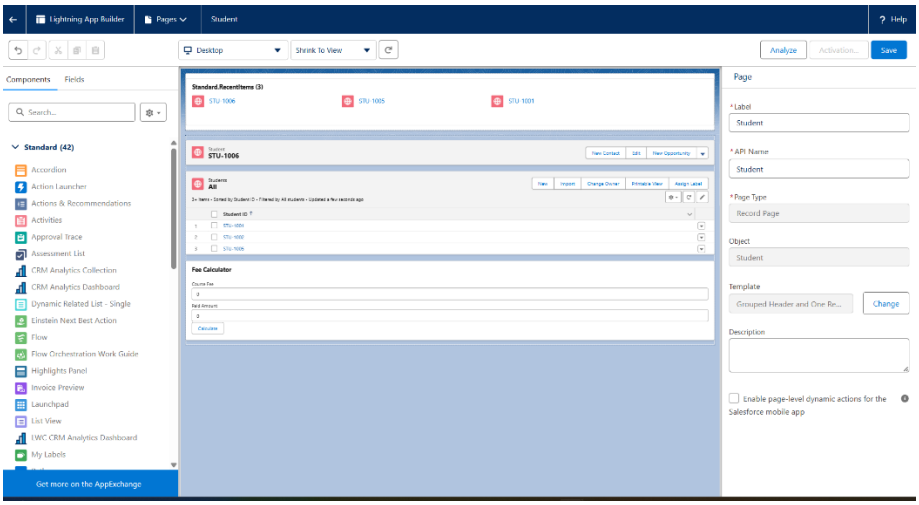


Deploy Component:

sf project deploy start --metadata LightningComponentBundle:feepayment -o



Add to Lightning Page:
- Go to Lightning App Builder → Drag feepayment onto Student Page.



Results:

- Student 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.

INSTITUTE MANAGEMENT SYSTEM

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.

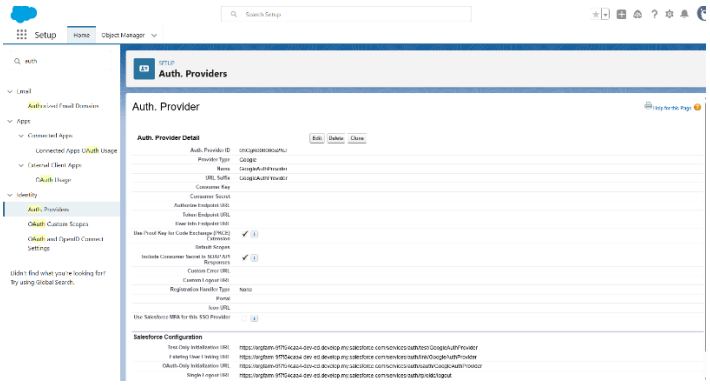
---

1) Named Credential — store external API credentials securely

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

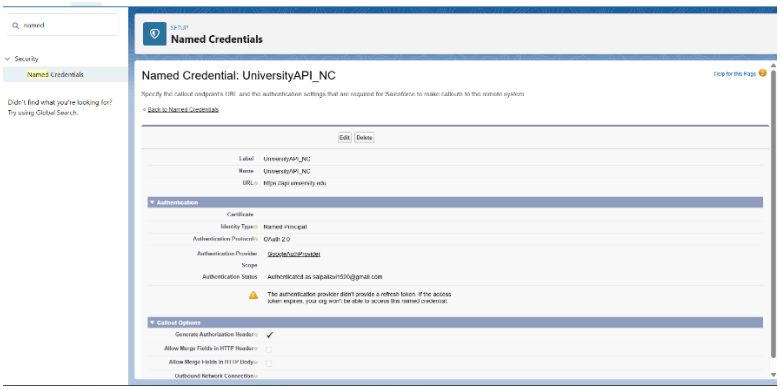
Steps

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



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

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



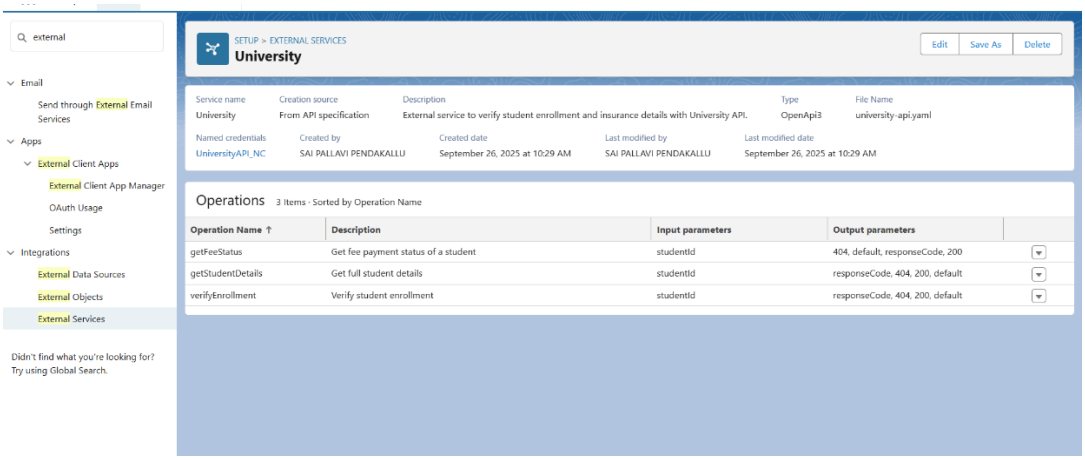
---

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.AllowsCallouts).

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

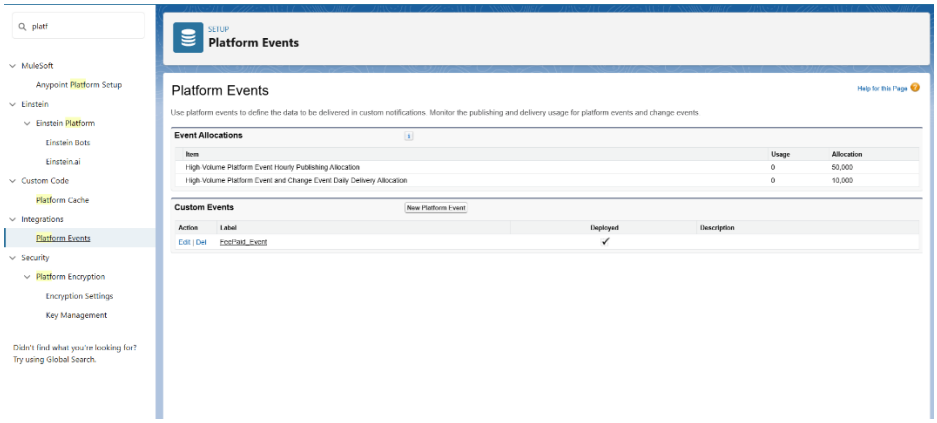
1. Go to **Setup → Platform Events → New Platform Event**.
2. Fill in the details:
    - **Label**: FeePaid_Event
    - **Plural Label**: FeePaid_Events
    - **API Name**: FeePaid_Event__e
    - **Publish Behavior**: Publish Immediately (so events are available as soon as saved)
3. Click **Save**.

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.



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.

INSTITUTE MANAGEMENT SYSTEM

Phase 8: Data Management & Deployment

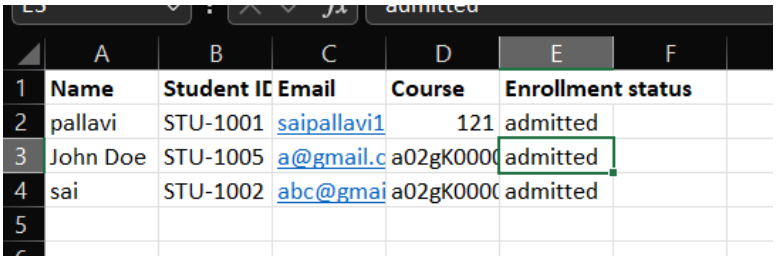
1 Data Import Wizard – Import Students, Courses, and Enrollments

Purpose: Quickly load small datasets (students, courses, enrollments) using Salesforce UI.

Step-by-Step:

1. Go to Setup → Data Import Wizard.
2. Click Launch Wizard.
3. Choose object to import:
    - Student__c → student records
    - Course__c → course catalog
    - Faculty__c → faculty records
4. Upload your CSV file for each object.

    Student csv file:



	A	B	C	D	E	F
1	**Name**	**Student ID**	**Email**	**Course**	**Enrollment status**	
2	pallavi	STU-1001	saipallavi1	121	admitted	
3	John Doe	STU-1005	a@gmail.c	a02gK000(	admitted	
4	sai	STU-1002	abc@gmai	a02gK000(	admitted	
5						

    Faculty csv file:

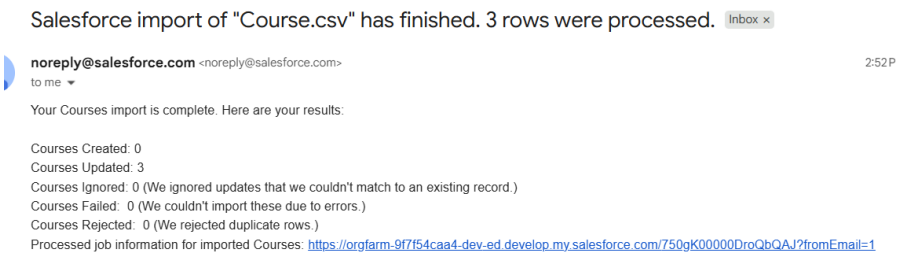
Faculty Na	Faculty ID	email	departmer	Phone
shiva	FAC-1001	shiva@gm	CSE	1.23E+08
raj	FAC-1002	raj@gmail.	ECE	(029)928-3839
sita	FAC-1003	sita@gmai	CE	(029)928-3527

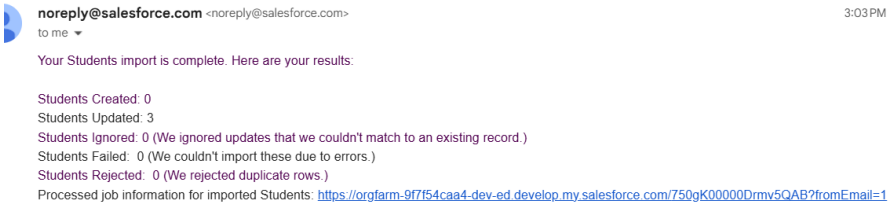
    Course csv file:

Course Co	Course na	Duration	Credits	Fees	Faculty
a02gK000(	CSE101	12	7	35000	FAC-1001
a02gK000(	ECE102	12	8	38000	FAC-1002
121	CE	12	6	100000	FAC-1001

5. Map CSV columns to Salesforce fields. Example:
    - Full_Name → Full_Name__c
    - Email → Email__c
    - Course Name → Course__c
6. Check checkboxes for:
    - “Trigger workflow rules, processes, and flows” if you want automation to run.
7. Click Start Import.
8. Review the success/error report. Correct any errors if necessary.

Outcome after importing csv files:



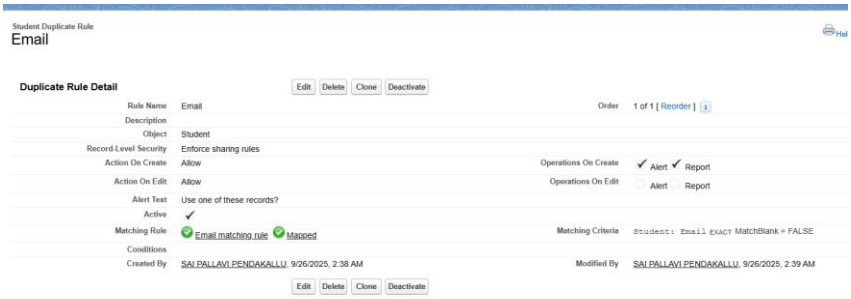


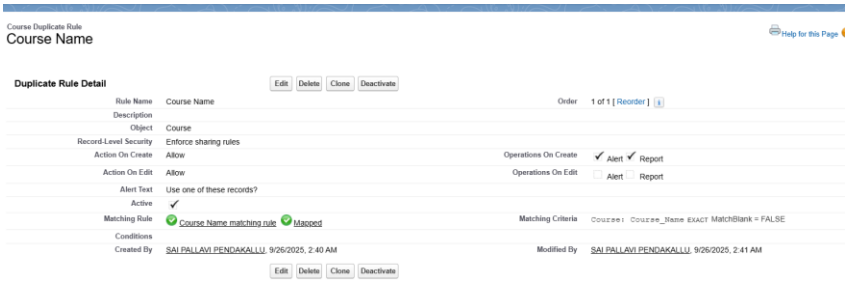
2. Duplicate Rules – Prevent Duplicate Students or Courses

Purpose: Ensure data integrity by preventing duplicate entries.

Step-by-Step:

1. Setup → **Duplicate Rules** → **New**.
2. Select **Object**:
    - **Student__c** (use Email as unique identifier)
    - **Course__c** (use Course Name as unique identifier)
3. Define **Matching Rules**:
    - Exact match on Email__c for students
    - Exact match on Course_Name__c for courses
4. Choose **Actions**:
    - **Block** → prevent duplicates completely
    - **Alert** → notify user but allow save
5. Save & Activate the rule.

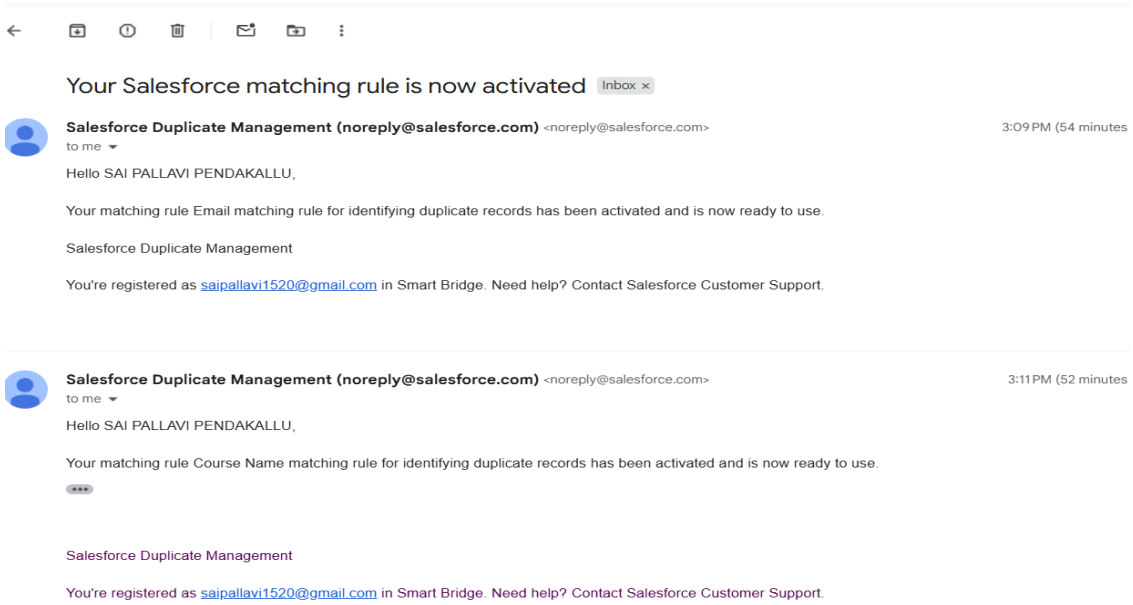




Reason for IMS: Prevents multiple records for the same student or course. Essential for accurate reporting and fee management.

Outcome:

After activation of these rules, some emails are generated. The generated emails are as follows,



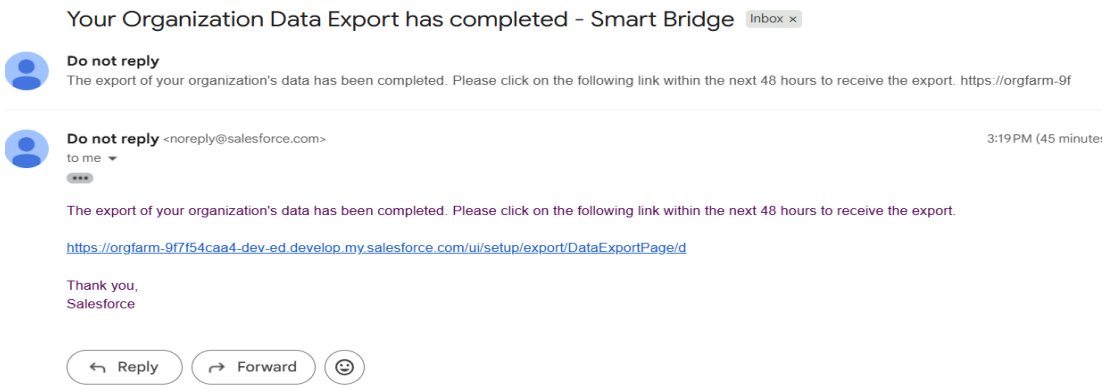
3.Data Export & Backup – Optional but Recommended

Purpose: Maintain regular backups for safety and disaster recovery.

Step-by-Step:

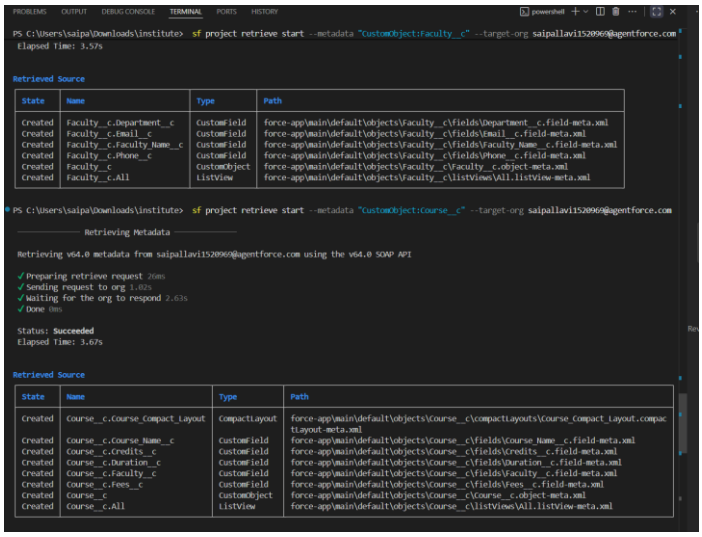
1. Setup → **Data Export**.
2. Click **Schedule Export**.
3. Select objects to export:
    - Student__c, Course__c, Enrollment__c, Fee_Payment__c
4. Choose **frequency**: Weekly or Monthly
5. Salesforce generates **CSV ZIP files**. Download and store securely.

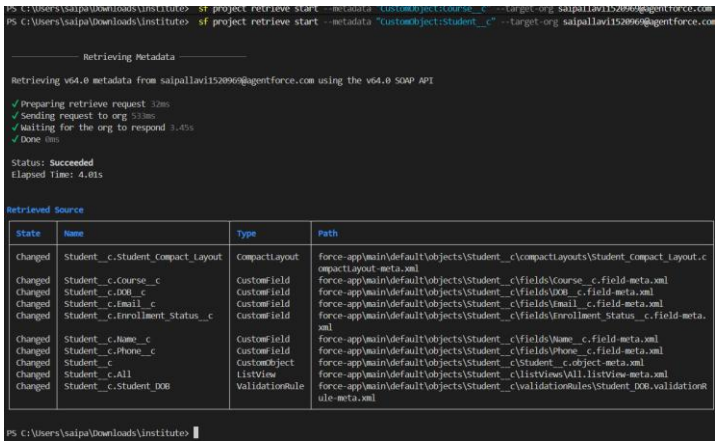
Reason for IMS: Not mandatory but recommended to protect data and comply with policies.



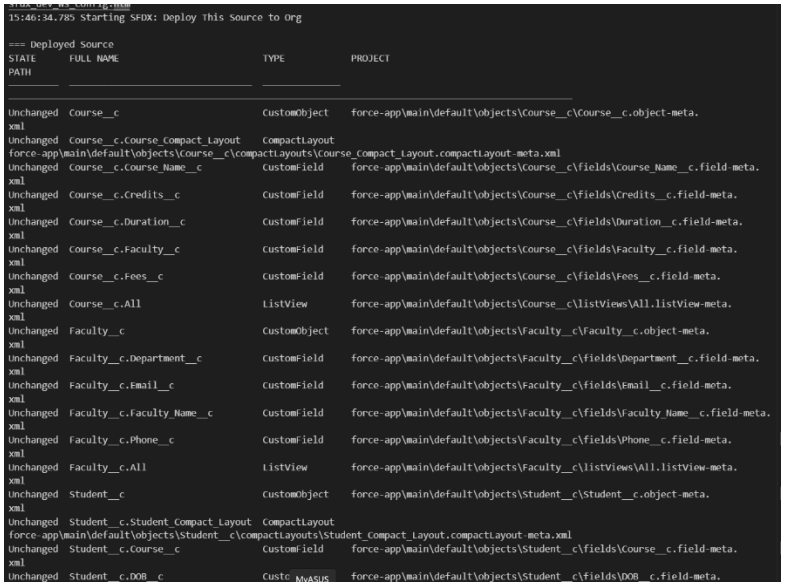
4. Deploy Metadata

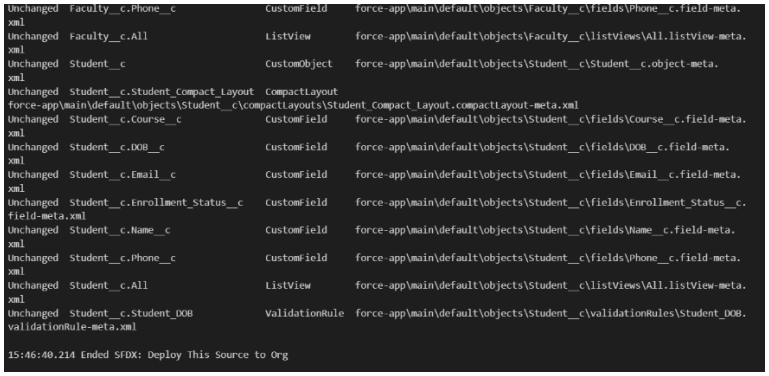
Option A: VS Code & Salesforce CLI
- Move your project to a local folder (avoid OneDrive).
- Open VS Code and authorize org:
  sf org login web --alias SourceOrg --set-default
- Retrieve metadata (Student, Course, Faculty objects):
  sf project retrieve start --metadata CustomObject:Student__c --metadata 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.

5. 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.

6. 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).

INSTITUTE MANAGEMENT SYSTEM

Phase 9: Reporting, Dashboards and Security Review

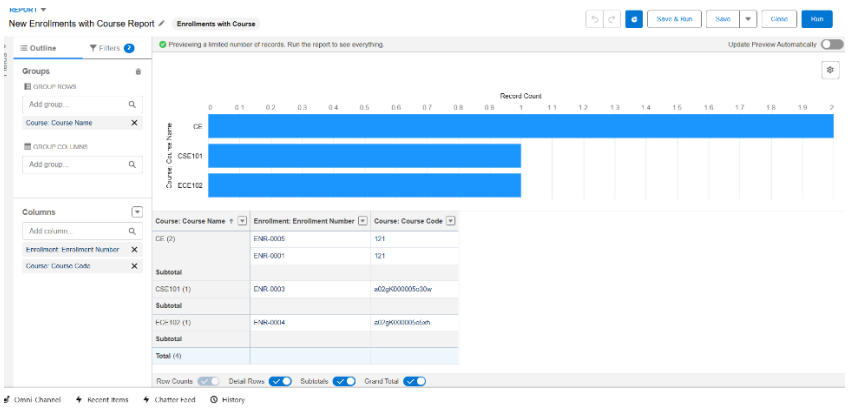
1 — Reports (create useful IMS reports)

Goal: build operational reports admins, finance and faculty will use.

A. Student Enrollment by Course (summary)

1. App Launcher → Reports → New Report.
2. In the Report Type selector choose Students with Enrollments (or use Enrollment__c report type if you created a custom one).
3. Click Start Report.
4. Remove unwanted columns; add these columns:
    - Course__c (or Enrollment → Course Name), Student__r.Full_Name__c, Enrollment_Status__c, Enrollment_Date__c.
5. Group Rows by Course__c:
    - Drag Course__c into the Group drop zone.
6. Add summary: count of records (this is automatic: “Count of Rows”).
7. Add filters:
    - Date Range (Enrollment_Date__c) → All Time or specific semester.
    - Enrollment_Status__c → (optional) Active.
8. Add a chart (click Add Chart) → Chart Type: Bar or Donut (Course vs Count).
9. Save → Name: Student Enrollment by Course → choose folder (e.g., “IMS Reports”) → Save & Run.

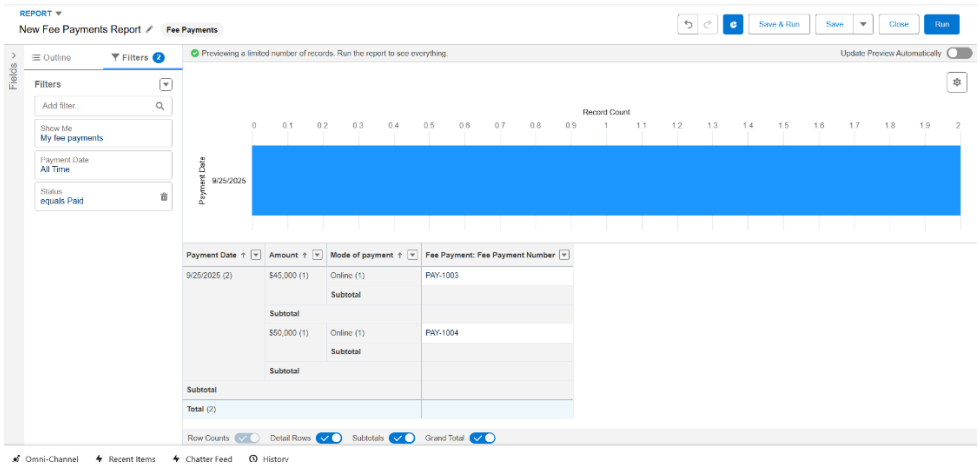
Verify: Run, ensure groups show correct student counts per course.



B. Fee Collection by Month (sum of payments)

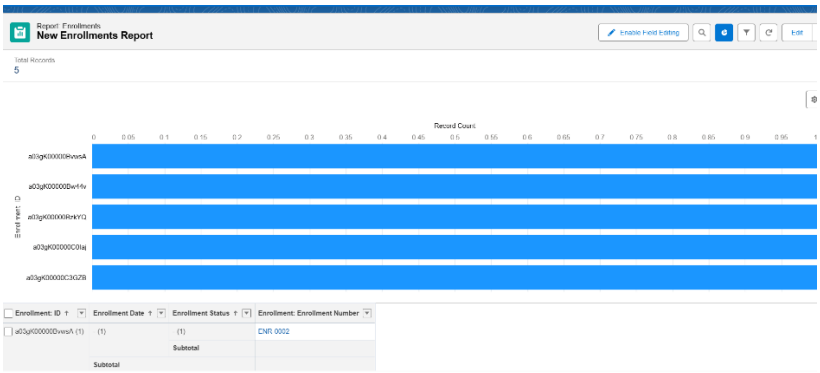
1. Reports → New Report → Select Fee Payments (or Fee_Payment__c) report type.
2. Start Report. Add columns:
    - Payment_Date__c, Amount__c, Student__r.Full_Name__c, Status__c.
3. Group Rows by Payment_Date__c → Group by Calendar Month (set grouping to Month).
4. Summarize Amount__c → click column dropdown → Summarize → choose Sum.
5. Add filters:
    - Status__c = Paid
    - Date range = This Year (or as needed).
6. Chart: Column chart (Month vs Amount).
7. Save → Name: Fee Collection by Month.

Verify: Values match finance totals for the selected timeframe.



C. Course Completion Report (who completed)

1. New Report → Choose Enrollments report type.
2. Filter: Enrollment_Status__c = Completed.
3. Columns: Student, Course, Grade__c, Completion Date (if available).
4. Group by Course or Faculty (if Course → Faculty lookup present).
5. Save as Course Completion Report.



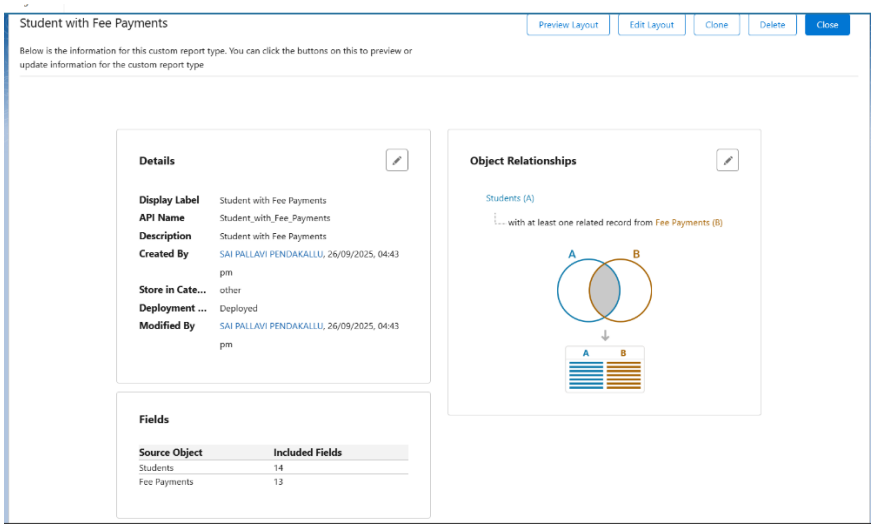
2 — Report Types (create custom ones for cross-object reporting)

Goal: enable reporting on combinations like Student + Fee Payment or Course + Enrollment.

Create Student + Fee Payment custom report type

1. Setup → Report Types → New Custom Report Type.
2. Primary Object: Student__c.
    - Label: Student with Fee Payments.
    - Plural Label: Students with Fee Payments.
    - Description: Students and their fee payments.
    - Store in category: IMS (choose or create).
    - Deployment Status: Deployed.
3. Click Next → Add Related Object:
    - Related Object: Fee_Payment__c
    - Relationship: Each "A" record must have at least one related "B" record OR A records may or may not have related B records — choose based on need.
4. Save.

Verify: New report type appears when creating new report; build a Student + Fee report



3 — Dashboards (visualize KPIs)

Goal: create a management dashboard and a finance dashboard.

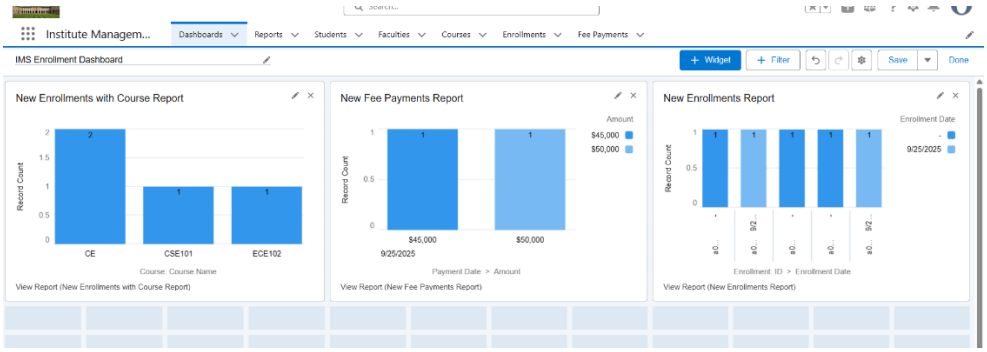
A. Create Enrollment Dashboard

1. App Launcher → Dashboards → New Dashboard.
2. Name: IMS Enrollment Dashboard. Folder: IMS Dashboards. Click Create.
3. Click + Component → Choose Report: Student Enrollment by Course → Choose component type: Bar Chart.
4. Configure display: Title, subtitle, ranges, and aggregate values. Click Add.
5. Add component: Fee Collection by Month (report) → Component type: Line/Column.
6. Add a Table component for Course Completion Report.
7. Save.

Verify: Dashboard loads; data refresh shows current metrics.

B. Dashboard Filters & Subscriptions

- Add Dashboard Filter (top-right → Add Filter) to allow toggling by Course or Faculty.
- Subscribe users: Open Dashboard → Subscribe → Add schedule & email recipients.



4 — Dynamic Dashboards (view as logged-in user)

Goal: show different data to each user without creating multiple dashboards.

Create/Modify Dashboard to be dynamic

1. Open dashboard in Edit mode → Edit Properties (gear icon).
2. Under View Dashboard As select "The dashboard viewer" (or “Logged-in user”).
    - This makes the dashboard dynamic: each viewer sees data they have access to (row-level security).
3. Save & Finish.

5 — Sharing Settings (OWD, Role Hierarchy, Sharing Rules)

Goal: set secure, appropriate record visibility.

A. Design (recommended for IMS)

- Student__c → Private (sensitive).
- Enrollment__c → Private (faculty see their students).
- Fee_Payment__c → Private (finance + admin).
- Course__c → Public Read Only (courses visible).

B. Set Organization-Wide Defaults (OWD)

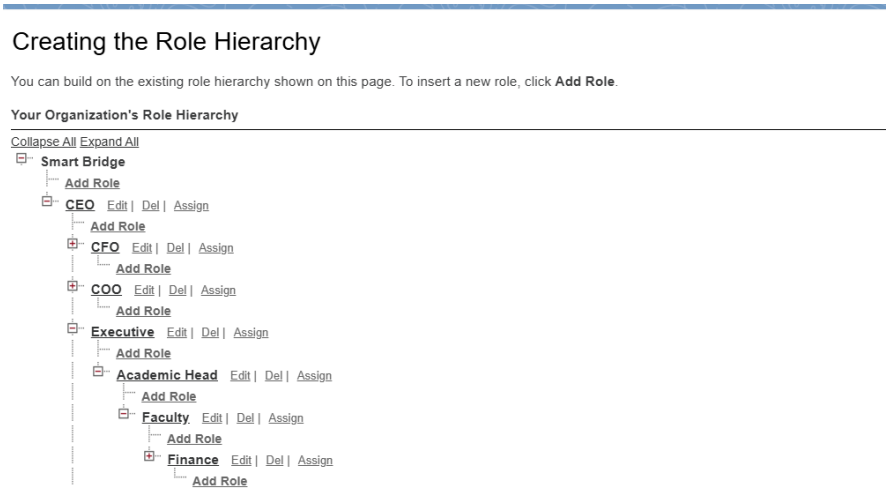
1. Setup → Sharing Settings.
2. Click Edit. For each object set Default Internal Access:
    - Student__c → Private
    - Enrollment__c → Private
    - Fee_Payment__c → Private
    - Course__c → Public Read Only
3. Save.

Work Type Group	Public Read/Write	Private	
Course	Public Read Only	Private	☑
Enrollment	Private	Private	☑
Faculty	Public Read/Write	Private	☑
Fee Payment	Private	Private	☑
Student	Private	Private	☑

C. Role Hierarchy

1. Setup → Roles → Set Up Roles.
2. Create roles: Executive, Academic Head, Faculty, Finance, Student Portal User etc.
3. Arrange hierarchy so supervisors inherit access.

Verify: Users see records owned by users below them in role hierarchy.



D. Sharing Rules (give broader access where needed)

1. Setup → Sharing Settings → scroll to object (e.g., Enrollment__c) → Click New under Sharing Rules.
2. Create rule:
    - Name: Share Enrollments to Faculty by Course (example).
    - Rule Type: Based on record owner or Based on criteria.
    - If criteria-based: Course__c = [specific course] (or use a custom field indicating faculty owner).
    - Share with: Roles (Faculty) or Public Groups (Faculty_Group).
    - Access Level: Read Only or Read/Write.
3. Save and Recalculate.

6 — Field-Level Security (FLS) & Page Layouts

Goal: hide sensitive fields for certain profiles.

A. Hide a field (example: Student ID Proof)

1. Setup → Object Manager → Student__c → Fields & Relationships.
2. Click the field you want to restrict (e.g., ID_Proof__c).
3. Click Set Field-Level Security.
4. Uncheck visibility for profiles you want to restrict (e.g., Faculty, Standard User) and keep checked for Admin and Finance only.
5. Save.

B. Remove field from Page Layouts (optional)

1. Still in Object Manager → Page Layouts → Edit the layout(s).
2. Remove the sensitive field from the layout for profiles you don't want seeing it. Save.

Verify: Login as test user with restricted profile and confirm field is not visible on record or in reports.

---

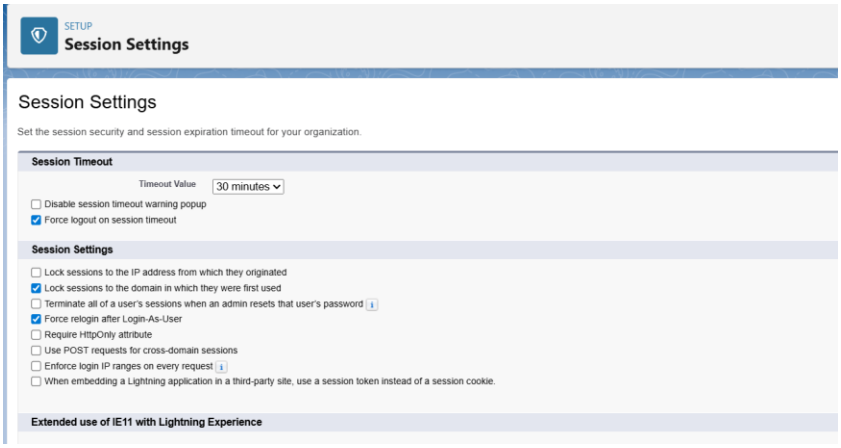
7 — Session Settings (timeout, security)

Goal: set session timeout to 30 minutes and enforce logout.

Steps

1. Setup → Session Settings.
2. Set Timeout Value to 30 minutes.
3. Optionally check Force logout on session timeout (if available) to block persistent sessions.
4. Save.

Verify: Log in as a test user, remain idle for 30 minutes (or use forced shorter value during testing), and confirm session times out.



---

8 — Login IP Ranges & Login Hours

Goal: restrict admin/faculty access to campus IPs while allowing students public portal access.

A. Profile-level Login IP Ranges (restrict)

1. Setup → Profiles → Select profile to restrict (e.g., Faculty Profile).
2. Scroll to Login IP Ranges → New.
    - Start IP Address = campus start IP
    - End IP Address = campus end IP
3. Save.

Note: Be careful—if you set IP ranges for the Admin profile, you can lock yourself out. Always test with a secondary admin account.

B. Network Access (org-wide allowed IP ranges; optional)

1. Setup → Network Access → New.
2. Add trusted IP ranges that bypass login verification (useful for offices).

Verify: Attempt login from outside allowed IP and confirm access is denied for that profile.

---

9 — Audit Trail & Field History Tracking

Goal: record who changed what and when.

A. View Setup Audit Trail (tracks setup changes)

1. Setup → View Setup Audit Trail.

2. Download last 180 days of changes to see metadata and admin changes (who changed profiles, permission sets, etc.).

B. Enable Field History Tracking (object-level)

1. Setup → Object Manager → Student__c → Fields & Relationships → click Set History Tracking (button near top).
2. Check Enable Track Field History.
3. Check the fields you want to track (e.g., Enrollment_Status__c, Course__c, Phone__c).
4. Save.
5. Edit Student page layout → Add the Student History related list to the layout so users can see history on the record page.

---

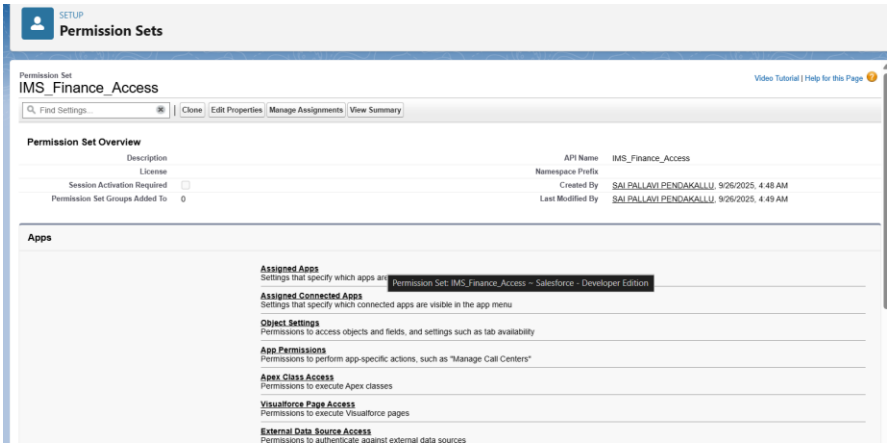
10 — Permission Sets & Profile Review (granular access)

Goal: give additional permissions without changing profiles.

Create a Permission Set

1. Setup → Permission Sets → New.
2. Label: IMS_Finance_Access. API Name auto-fills. Save.
3. In Permission Set → Object Settings → select Fee_Payment__c → click Edit → check Read, Create, Edit, Delete as appropriate. Save.
4. Assign the permission set to specific users: Permission Set → Manage Assignments → Add Assignments → select users.

Verify: Login as a user with the permission set and confirm they can access fee payment records as intended.



11 — Testing & Validation (end-to-end)

After implementing above:

1. Test reports & dashboards
    - Run Student Enrollment by Course and compare counts to source data.
    - Schedule a dashboard refresh and subscription to confirm email delivery.
2. Test security
    - Create test users with different roles/profiles (Faculty, Finance, Student).
    - Confirm View As behavior for dynamic dashboard (logged-in user sees correct data).
    - Confirm restricted profile cannot see sensitive fields and is blocked outside IP range.
3. Test history
    - Change a tracked field and confirm history entries appear.
4. Document everything
    - Record setup choices, folders, report names, dashboard names, sharing rules, and permission sets for future audits.

- Enable Field History Tracking for key fields; add History related lists.
- Create Permission Sets for Finance and Faculty; assign to users.
- Run test validation with test users.