

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.

Auth. Provider Detail	
Auth. Provider ID	0SOgL000000srAb
Provider Type	Google
Name	Google
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
Icon URL	

2. Setup → Named Credentials → New Named Credential.

- o 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`.

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.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.

The screenshots show the Salesforce Platform Events setup interface. The top screenshot displays the 'Event Allocations' section, which lists two allocations: 'High Volume Platform Event Hourly Publishing Allocation' and 'High Volume Platform Event and Change Event Daily Delivery Allocation'. The bottom screenshot shows the 'Custom Events' section for 'FeePaid_Event', where you can view and edit event details such as Plural Label, Object Name, API Name, Event Type, Publish Behavior, and Standard 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.