

# Smart Student Enrollment & Progress Tracker

## Problem Statement

Currently, the student enrollment process is completely manual. Students have to physically submit forms or email their requests. They cannot track the status of their enrollment, leading to confusion and delays. Faculty members must manually review and approve requests, which is timeconsuming. There is also no proper reporting mechanism for administrators to monitor student progress or enrollment statistics.

## Phase 1 — Problem Understanding

The first phase focuses on clearly understanding the problem we are solving. We defined the issue, wrote user stories, and drew a simple process flow.

- Problem Statement: Enrollment is manual, students can't track progress, and there is no reporting for Admin.
- User Stories: 1) As a Student, I want to request enrollment in a course so I can participate. 2) As a Faculty member, I want to approve/reject enrollment requests so that only eligible students are enrolled. 3) As an Admin, I want to see reports of student enrollment and progress to monitor performance.
- Process Flow Diagram: Student → Enrollment Request → Faculty Approval → Enrollment Status Updated → Admin Reporting.
- **Requirement Gathering:**  
We began by understanding the current enrollment process at universities. The main issues identified were that enrollment was fully manual, students had no way to track the status of their requests, and faculty had to handle approvals through emails or offline communication. This created confusion, delays, and duplication of work. To resolve this, we gathered requirements from the perspective of all users. Finally, we decided Salesforce automation could simplify and centralize the process.
- **Stakeholder Analysis:**  
The project involves three main stakeholders. Students want an easy way to request enrollment in courses and view their progress. Faculty need a structured method to approve or reject requests without managing everything manually. Admins require access

to reports and dashboards to monitor student performance and enrollment trends. By analyzing these stakeholders, we defined clear roles and responsibilities. This ensured that our solution would be user-focused.

- **Business Process Mapping:**

We mapped out the entire enrollment process to visualize how information flows. The process starts when a Student submits an enrollment request. It moves to Faculty for approval or rejection. Once approved, the enrollment status is updated to Enrolled. Finally, Admins access this data in reports and dashboards. This clear mapping helped us identify the key objects required — Student, Course, and Enrollment. We used a flow diagram to illustrate this.

- **Industry-specific Use Case Analysis:**

We studied how real-world universities and institutions handle enrollments. Most rely on either manual paperwork or fragmented systems that do not give students transparency. Our analysis confirmed the need for a digital solution to improve efficiency, reduce errors, and provide visibility. With Salesforce's CRM and automation features, we could design a solution similar to what many modern universities are adopting. This validated our approach.

- **AppExchange Exploration:**

Before starting custom development, we explored Salesforce AppExchange for ready-made solutions. While there are education management apps available, they were either too complex or not aligned with our specific use case. Many had extra features we didn't need, which would add unnecessary complexity. After this exploration, we decided to build a custom solution from scratch. This ensured simplicity and full control over functionality.

## **Verification:**

We confirmed that we could explain the problem in two clear sentences and had a process diagram ready. This ensures the team fully understands the goal before starting development.

## Phase 2 — Org Setup & Security

The second phase involves preparing the Salesforce org to support our solution. This includes setting up organization details, business hours, holidays, roles, profiles, and security settings.

### Salesforce Editions:

Salesforce provides different editions like Essentials, Professional, Enterprise, and Developer. For this project, we selected the **Developer Edition** because it offers all customization features at no cost. This edition allows us to create custom objects, write Apex triggers, build Lightning Components, and configure automation flows.

- **Company Profile Setup** : Configured Company Information: Set organization name to ABC University, timezone to Asia/Kolkata, and currency to INR.

The screenshot shows the 'Company Information' setup page in Salesforce. The organization's profile is displayed, including the organization name 'Prof. Ram meghe institute of technology and research badnera', primary contact 'shravani kalimbe', address 'anjangao bari road Badnera 444701 Maharashtra India', phone '(902) 891-4659', fax '(902) 891-4659', default locale 'English (United States)', and default language 'English'. Other details include fiscal year starting in April, currency INR, and time zone (GMT+05:30) India Standard Time (Asia/Kolkata). The page also lists API requests, streaming API events, restricted logins, and salesforce.com organization ID.

**Business Hours & Holidays:** To ensure system automations respect working times, we configured Business Hours. We created “University Hours” as Monday to Friday, 9 AM–5 PM. Next, we added two holidays — Diwali and Christmas. These holidays were associated with University Hours so that approval processes and escalation rules would pause during non-working days. This ensures the system operates realistically according to the academic calendar.

**Roles:** We created sample users to represent real stakeholders. Prof. Rahul was added as a Faculty member with a Salesforce license, which grants full CRM access. Student One was created with a Salesforce Platform license, which provides limited access but is ideal for self-service students. Defining correct licenses ensures users only consume the necessary features. This step also allowed us to simulate real scenarios by logging in as different roles.

**Defined Roles:** Admin (top level), Faculty (reports to Admin), and Student (reports to Faculty).

**Profiles:** Created Profiles: Cloned Standard User to create Faculty and Student profiles with appropriate permissions.

- Configured Sharing Settings: Student\_\_c = Private **OWD (Org-Wide Defaults):** Org-Wide Defaults establish the baseline visibility for all objects. We configured:

- Student\_c = Private (a student can only see their own record).
- Course\_c = Public Read Only (everyone can view courses).
- Enrollment\_c = Controlled by Parent (visibility depends on the linked Student and Course). This ensures data privacy while maintaining transparency where required.

The screenshot shows the 'Sharing Settings' page in the Salesforce Setup. The left sidebar has a search bar and navigation links for Security, Users, and Objects. Under 'Sharing Settings', there's a note about global search. The main area displays a table of objects and their sharing settings:

Object	Sharing Method	Access Level	Controlled By
Shipping Carrier Method	Public Read Only	Private	
Shipping Configuration Set	Public Read Only	Private	
Streaming Channel	Public Read/Write	Private	
Tableau Host Mapping	Public Read Only	Private	
User Presence	Public Read Only	Private	
User Provisioning Request	Private	Private	
Waitlist	Private	Private	
Webs Cart Document	Private	Private	
Work Order	Private	Private	
Work Plan	Private	Private	
Work Plan Template	Private	Private	
Work Step Template	Private	Private	
Work Type	Private	Private	
Work Type Group	Public Read/Write	Private	
Course	Public Read Only	Private	
Enrollment	Private	Private	
mentor	Public Read/Write	Private	
Student	Private	Private	

**User Setup & Licenses:** Created Users: Prof. Rahul (Faculty profile) and Student One (Student profile) with respective roles and licenses.

The screenshot shows the 'Users' page in the Salesforce Setup. The left sidebar has a search bar and navigation links for Users, Permission Sets, Profiles, and Roles. Under 'User Management Settings', there's a link to 'User Management Settings'. The main area displays a table of users:

Action	Full Name	Alias	Username	Role	Active	Profile
Edit	Chatter Expert	Chatter	chatty_00dgk0000007ansnua1tqkzoylnhr6@chatter.salesforce.com		✓	Chatter Free User
Edit   Login	chaudhari_prof_anand	pchau	anandchaudhari@gmail.com	faculty	✓	Faculty
Edit   Login	EPIC_OrgFarm	QEPIQ	epic.1aa13ff481a5@orgfarm.salesforce.com		✓	System Administrator
Edit   Login	Kalambe_Shrawan	shr	shrawanikalambe28722@agenforce.com		✓	System Administrator
Edit   Login	one_student	sone	studentfirst@gmail.com	Student	✓	Standard Platform User
Edit	sara	sara	sara1234@gmail.com		✓	System Administrator
Edit   Login	Shravani_Kalambe_Shrawan_Kalambe	shri	shrawanikalambe28@gmail.com		✓	System Administrator
Edit	User_Integration	Integ	integration@000g00000007ansnua1.com		✓	Analytics Cloud Integration User
Edit	User_Security	sec	insightsecurity@00dgx0000007ansnua1.com		✓	Analytics Cloud Security User

- Created Custom Tabs: Student, Course, and Enrollment objects now have tabs to allow navigation in the Lightning app.

The screenshot shows the Salesforce Setup interface with the 'Custom Tabs' page selected. The page has a header with tabs for 'Setup', 'Home', and 'Object Manager'. A search bar at the top right says 'Search Setup'. Below the header, there's a sidebar with 'User Interface' and 'Rename Tabs and Labels' sections. The main content area is titled 'Custom Tabs' and contains a message about creating new custom tabs. It then lists four categories: 'Custom Object Tabs', 'Web Tabs', 'Visualforce Tabs', and 'Lightning Component Tabs'. Under 'Custom Object Tabs', there is a table with the following data:

Action	Label	Tab Style	Description
Edit   Del	Courses	Books	
Edit   Del	Enrollments	Cell phone	
Edit   Del	Students	Form	

- Added Tabs to Student Management App: Ensured Students, Courses, and Enrollments are visible to Admin and Faculty.

### Login Access Policies:

Salesforce allows admins to log in as other users to test profiles and permissions. We used this feature to verify the experience of both Faculty and Students. Logging in as Prof. Rahul confirmed he could view Courses and Enrollments but not other Students. Logging in as Student One confirmed they only had access to their own data. This final step ensured our security configuration was correct..

The screenshot shows the Student Management app in Salesforce Lightning Experience. The page title is 'Approvals'. The header includes tabs for 'Home', 'Enrollments', 'Students', and 'Courses'. The main content area has a heading 'Approvals' and a sub-instruction 'Use Approvals to manage approval submissions and approval work items.' Below the text is a cartoon illustration of a fish swimming near a hook and a boat.

### Verification:

Logged in as Faculty and confirmed they could see the Courses and Enrollments tabs but not other students' records. Logged in as Student and confirmed they could only see their own data. This ensured security settings worked as expected.

# Phase 3 — Data Modeling & Relationships

## 1. Standard & Custom Objects

**Information:** Objects are database tables.

**What We Did:** Created 3 custom objects: Student\_\_c, Course\_\_c, Enrollment\_\_c. **Verification:** Objects appear in Object Manager.

The image displays three separate screenshots of the Salesforce Object Manager interface, each showing a list of objects and their details. The first screenshot shows the 'Course' object, which is a custom object. The second screenshot shows the 'Enrollment' object, also a custom object. The third screenshot shows the 'Student' object, another custom object. Each screenshot includes a table with columns for Name, Type, Status, and Last Modified Date.

Name	Type	Status	Last Modified Date
Course	Course__c	Custom Object	9/19/2025
Credential Stuffing Event Store	CredentialStuffingEventStore	Standard Object	
Credit Memo	CreditMemo	Standard Object	
Credit Memo Invoice Application	CreditMemoInvApplication	Standard Object	
Credit Memo Line	CreditMemoLine	Standard Object	
Custom Library	DataMaskCustomValueLibrary	Standard Object	
Customer	Customer	Standard Object	
D&B Company	DandBCompany	Standard Object	
Data Kit Deployment Log	DataKitDeploymentLog	Standard Object	
Data Use Legal Basis	DataUseLegalBasis	Standard Object	
Data Use Purpose	DataUsePurpose	Standard Object	
Delivery Estimation Setup	DeliveryEstimationSetup	Standard Object	
Digital Wallet	DigitalWallet	Standard Object	

Name	Type	Status	Last Modified Date
Enrollment	Enrollment__c	Custom Object	9/19/2025
Entitlement	Entitlement	Standard Object	
Entitlement Contact	EntitlementContact	Standard Object	

Name	Type	Status	Last Modified Date
Skill Requirement	SkillRequirement	Standard Object	
Social Persona	SocialPersona	Standard Object	
Store	WebStore	Standard Object	
Store Buyer Group	WebStoreBuyerGroup	Standard Object	
Store Catalog	WebStoreCatalog	Standard Object	
Student	student__c	Custom Object	to add information of the student
Student	Student__c	Custom Object	

## 2. Fields

**Information:** Store attributes of each object.

- Student\_\_c → Student\_ID, DOB, Program, GPA, Email

**Fields & Relationships**  
10 items, Sorted by Field Label

Created By	CreatedById	Lookup(User)
DOB_c	DOB_c_c	Date
Email_c	Email_c_c	Email
Flag_For_Counseling_c	Flag_For_Counseling_c_c	Checkbox
GPA_c	GPA_c_c	Number(2, 2)
Last Modified By	LastModifiedById	Lookup(User)
Owner	OwnerId	Lookup(User,Group)
Program_c	Program_c_c	Picklist
Student Number	Name	Text(80)
Student_ID_c	Student_ID_c_c	Text(18)

- Course\_\_c → Course\_Code, Capacity, Mode.

**Fields & Relationships**  
8 items, Sorted by Field Label

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Capacity_c	Capacity_c_c	Number(18, 0)		
Course Name	Name	Text(80)		✓
Course_Code_c	Course_Code_c_c	Text(40)		
Created By	CreatedById	Lookup(User)		
Last Modified By	LastModifiedById	Lookup(User)		
Mode_c	Mode_c_c	Picklist		
Owner	OwnerId	Lookup(User,Group)		✓
Record Type	RecordTypeId	Record Type		✓

- Enrollment\_\_c → Status, Request Date, Student\_\_c (lookup), Course\_\_c (lookup).  
**Verification:** Fields visible on page layouts and records.

### 3. Record Types

**Information:** Different business processes per object.

For Course\_\_c → Record Types: Online, On-Campus.

**Verification:** When creating Course, Salesforce asks for record type → correct layout shown.

The screenshot shows the Salesforce Object Manager interface for the 'Course' object. The left sidebar has a 'Record Types' section selected. The main area displays a table titled 'Record Types' with two items: 'On-Campus' and 'Online'. The table includes columns for 'Record Type Label', 'Description', 'Active', and 'Modified By'. Both entries were modified by 'Shravani Kalambe' on 9/23/2025 at different times (11:49 AM and 11:47 AM).

## 4. Page Layouts

**Information:** Define UI for records.

- Online Course Layout → fields like Platform, Meeting Link.
- On-Campus Layout → fields like Classroom, Campus Location. **Verification:** Different layouts show based on record type.

The screenshot shows the 'Online Course Layout' configuration for the 'Course' object. The left sidebar has a 'Page Layouts' section selected. The main area shows the layout editor with various sections like 'Fields', 'Section', 'Blank Space', 'Created By', 'Owner', 'Record Type', 'Last Modified By', and 'Mode\_\_c'. Below the layout editor, there are panels for 'Highlights Panel', 'Quick Actions in the Salesforce Classic Publisher', and 'Salesforce Mobile and Lightning Experience Actions'.

## 5. Compact Layouts

**Information:** Show key fields in highlights panel.

Student Compact Layout = Name, Program, GPA.

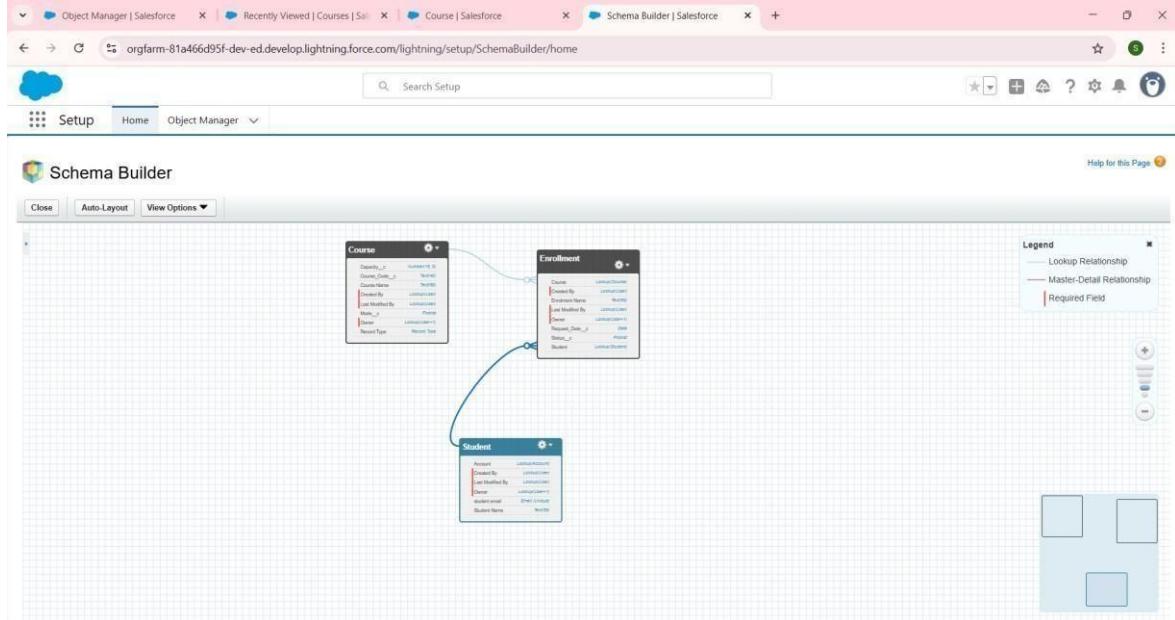
**Verification:** Compact layout visible on top of record page.

## 6. Schema Builder

**Information:** Visual tool to see object relationships.

**What We Did:** Used Schema Builder to check links between Student, Course, Enrollment.

**Verification:** Diagram shows Enrollment linked to Student & Course.



## 7. Lookup vs Master-Detail vs Hierarchical

**Information:** Relationship types.

Used **Lookup** relationships for Enrollment → Student and Enrollment → Course. **Verification:** Enrollment records display in related lists on Student & Course.

## 8. Junction Objects

**Information:** Object to connect 2 objects in many-to-many. Enrollment\_\_c acts as junction between Student & Course.

**Verification:** One Student can enroll in many Courses; One Course can have many Students.

## 9. External Objects

**Information:** Used for connecting external DBs. Not used in this project. **Verification:** N/A.

### End of Phase 3 Verification:

- Created Student record.
- Created Course record.
- Created Enrollment linking Student + Course. • Related lists working. • Record Types tested.

## Phase 4: Process Automation (Admin)

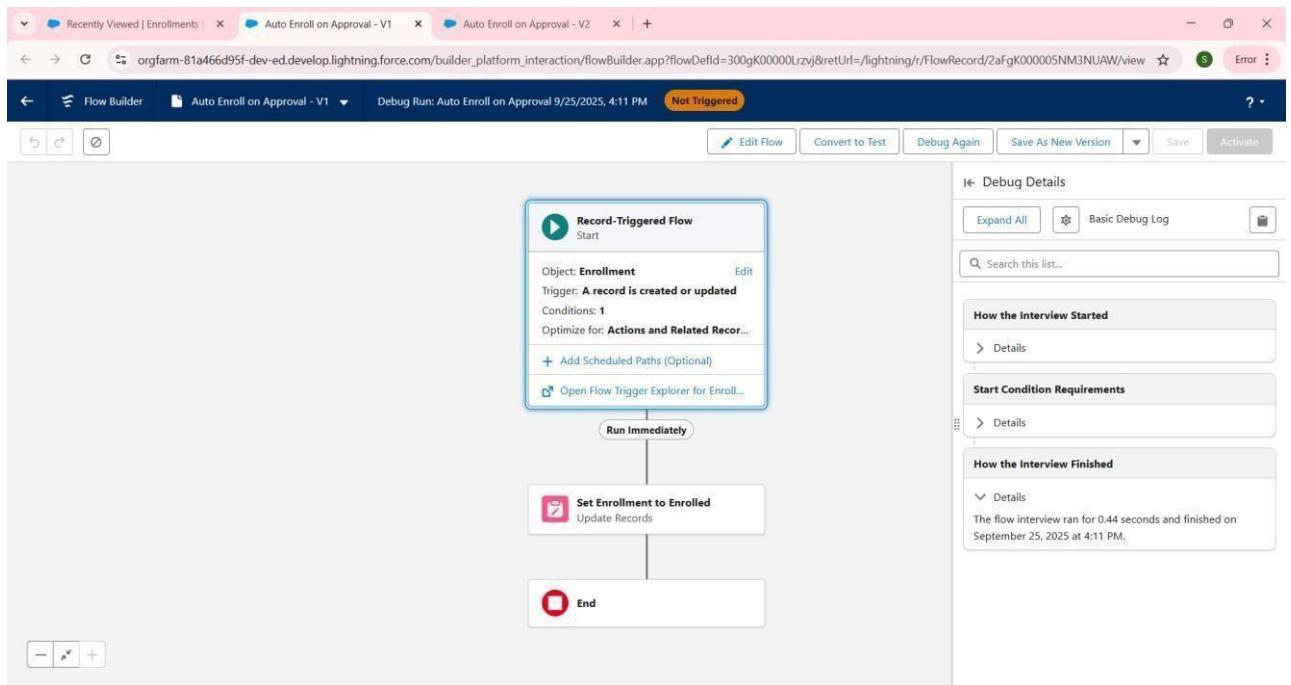
## 1. Validation Rules

We implemented a **Validation Rule** on the Student\_\_c object to ensure that the **Date of Birth (DOB\_\_c)** is mandatory. This helps maintain accurate student records and prevents saving incomplete data. The rule uses the formula ISBLANK(DOB\_\_c) and shows an error message if left empty. This runs automatically when a record is created or edited.

The screenshot shows the Salesforce interface for editing a student record. The main title is 'Edit student 2'. On the left, there's a sidebar with 'Related' and 'Details' tabs. Under 'Details', the 'Student Name' field contains 'student 2'. The 'Owner' is listed as 'Shravani Kalambe'. The 'student email' field contains 'student2@gmail.com'. The 'Account' field has a search bar. The 'DOB\_c' field is highlighted with a red border and contains the placeholder text 'Please enter Date of Birth.'. Below the form, there are buttons for 'Cancel', 'Save & New', and 'Save'. The status bar at the bottom right shows 'No activities to show, sending an email, scheduling a task, and more.'

## 2. Workflow Rules

Although **Workflow Rules** are legacy tools, we planned to use modern alternatives like **Flows**. Traditionally, workflow rules help automate actions like field updates, email alerts, or tasks based on record criteria. In our project, we replaced workflows with **Record-Triggered Flows** and **Approval Processes** for better flexibility. If needed, simple workflows can still be added for quick alerts or updates.



### 3. Process Builder

Salesforce is phasing out **Process Builder** in Favor of **Flows**, so we did not use it directly. Instead, all process automation (like auto-enrolment after approval) was implemented via **Flows**. Process Builder is useful for multi-step logic and was previously used for approval actions, field updates, and related record changes. In our solution, its functionality is fully replaced by Flow Builder.

### 4. Approval Process

An **Approval Process** was created for Enrollment\_\_c to manage enrollment requests. When the status is set to "Requested", it gets routed to a user in the **faculty role** for review. Upon approval, the status automatically updates to "Approved"; if rejected, it changes to "Rejected". This ensures formal authorization before enrolling a student into a course.

### 5. Flow Builder

We used a **Record-Triggered Flow** on Enrollment\_\_c to detect when a record is approved. It automatically updates the status to "**Enrolled**" if the previous status wasn't already approved. Flow Builder replaced older tools like Workflow and Process Builder. More Flow types (Screen, Auto-launched, Scheduled) can be added later for GPA recalculation or student re-evaluation.

### 6. Email Alerts

While not implemented directly yet, **Email Alerts** can be used to notify students or faculty. For example, an alert can be sent to a student when their enrollment is approved or rejected. This can be set up within an **Approval Process** or **Flow** using predefined templates and recipients. It enhances communication without manual effort.

## 7. Field Updates

We used **Field Updates** as part of the Approval Process to change the Status\_\_c field. On **final approval**, the status is updated to "Approved", and on **rejection**, it's set to "Rejected". This is a key part of automating the enrolment workflow and ensures the record reflects the latest state of the approval lifecycle.

## 8. Tasks

Although not implemented yet, **Tasks** can be auto-created using Flows to assign follow-up actions. For example, if a student's GPA drops below 2.0, a task could be assigned to a counsellor to intervene. Tasks help ensure accountability and follow-through on critical actions without manual assignment.

## 9. Custom Notifications

**Custom Notifications** can be configured via **Flows** to notify users inside the Salesforce UI or mobile app. We can notify students or faculty when key events occur — like successful enrollment or when GPA triggers a counselling flag. These are more modern than email and appear directly in the Salesforce bell icon.

# Phase 5: Apex Programming (Developer)

## 1. Classes & Objects

We created **Apex classes** like GpaBatch and GpaScheduler. These classes encapsulate logic in methods, and objects (like Student\_\_c, Enrolment\_\_c) are manipulated inside them. This shows how Salesforce data (custom objects) interact with code structures (Apex classes).

The screenshot shows the Salesforce Setup interface with the 'Apex Classes' tab selected. The page title is 'Apex Classes'. A search bar at the top right contains the text 'Search Setup'. The main content area displays the details for the 'GpaBatchScheduler' class. The class name is 'GpaBatchScheduler'. Below it, the code body contains the following Apex code:

```

1 global class GpaBatchScheduler implements Schedulable {
2     global void execute(SchedulableContext sc) {
3         Database.executeBatch(new GpaBatch(), 200);
4     }
5 }

```

At the bottom of the code editor, there are buttons for 'Edit', 'Delete', 'Download', 'Security', and 'Show Dependencies'.

## 2. Apex Triggers (before/after insert/update/delete)

We built **before insert** and **before update** triggers on Student\_\_c.

- Before insert → blocks duplicate emails.
- Before update → flags counseling and publishes a platform event if GPA < 2.0. This shows

The screenshot shows the Salesforce Setup interface with the 'Apex Triggers' tab selected. The page title is 'Apex Triggers'. A search bar at the top right contains the text 'Search Setup'. The main content area displays the details for the 'StudentTrigger' trigger. The trigger name is 'StudentTrigger'. Below it, the code body contains the following Apex code:

```

trigger StudentTrigger on Student__c (before insert, after update) {
    if (Trigger.isBefore && Trigger.isInsert) {
        StudentTriggerHandler.preventDuplicateEmails(Trigger.new);
    }
    if (Trigger.isAfter && Trigger.isUpdate) {
        StudentTriggerHandler.handlePostUpdate(Trigger.new, Trigger.oldMap);
    }
}

```

At the bottom of the code editor, there are buttons for 'Edit', 'Delete', 'Download', 'Security', and 'Show Dependencies'.

how triggers can enforce rules before saving data.

### 3. Trigger Design Pattern

We separated logic into **helper classes** instead of writing heavy logic directly in triggers. This is a best practice called the **Trigger Handler Pattern**, which makes code reusable, easier to test, and maintainable.

### 4. SOQL & SOSL

We used **SOQL queries** to fetch existing students (for duplicate email check) and to calculate GPA from Enrolment\_\_c. Aggregate queries with AVG() were applied in the batch. SOSL wasn't needed here but is useful for text searches across objects.

### 5. Platform Event (LowGPAAlert\_\_e)

The **LowGPAAlert\_\_e** event is a custom platform event used to broadcast low GPA cases. When the GPA trigger fires, this event is published with student details. Other Salesforce processes or external apps can **subscribe** to this event to take actions like sending notifications or assigning tasks.

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

- Platform Events** page.
- Action**: Publish Behavior is set to "Publish Immediately".
- Created By**: Shrawan Kalambe, 9/25/2025, 1:26 PM.
- Modified By**: Shrawan Kalambe, 9/25/2025, 1:26 PM.
- Standard Fields** table:

Action	Field Label	API Name	Data Type	Controlling Field	Indexed
Edit   Del	Created By	Createdby	Lookup(User)		
Edit   Del	Created Date	CreatedDate	Date/Time		
Edit   Del	Event UUID	EventUuid	Text(36)		
Edit   Del	Replay ID	ReplayId	External Lookup		
- Custom Fields & Relationships** table:

Action	Field Label	API Name	Data Type	Indexed	Controlling Field	Modified By
Edit   Del	GPA	GPA__c	Number(4, 2)			Shrawan Kalambe, 9/25/2025, 1:30 PM
Edit   Del	Message	Message__c	Long Text Area(256)			Shrawan Kalambe, 9/25/2025, 1:31 PM
Edit   Del	StudentId	StudentId__c	Text(18)			Shrawan Kalambe, 9/25/2025, 1:29 PM
- Triggers**: No triggers defined.
- Subscriptions**: Subscriber, Last Processed Id, Last Published Id, State, Batch Size, User.

### 6. Collections: List, Set, Map

- List**: stored students to update.
- Set**: used for unique student IDs in batch scope.
- Map**: stored StudentId → GPA pairs from aggregate queries.

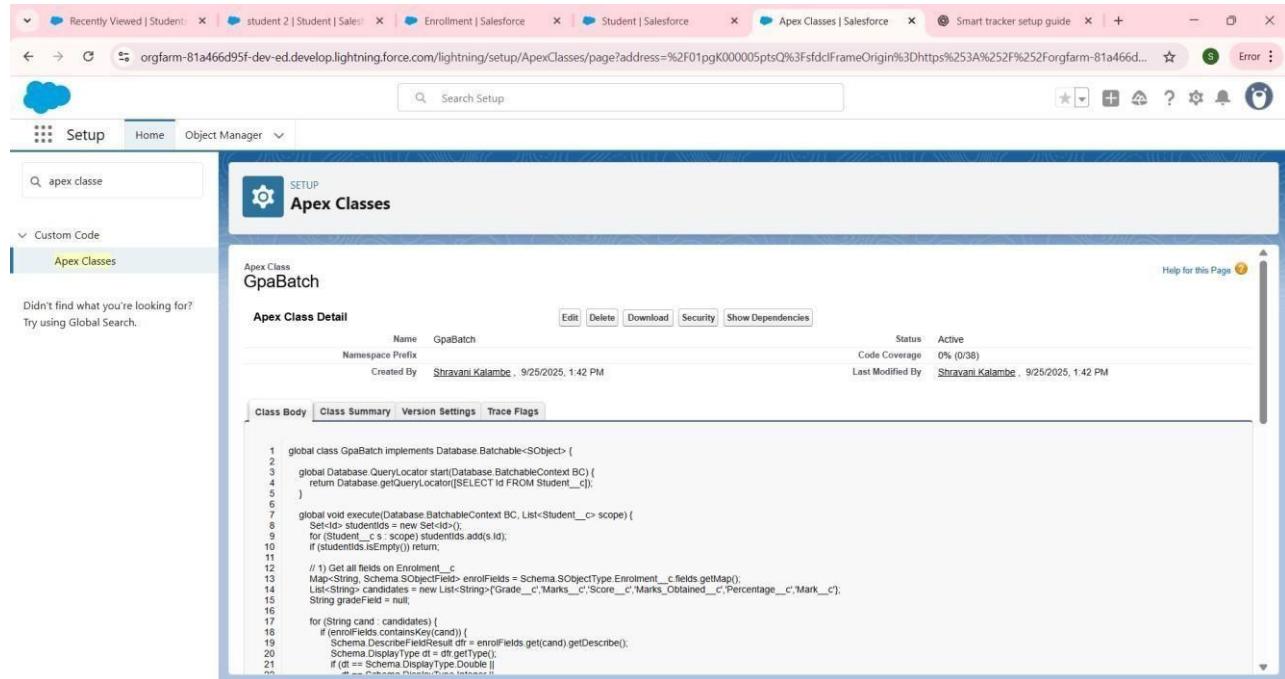
Collections improved performance and avoided duplicates in processing.

### 7. Control Statements

We used **if conditions and loops** to check duplicates, validate GPA, and loop through batch records. For example, if(avgMap.containsKey(s.Id)) ensures only students with enrolments are updated.

## 8. Batch Apex

We created GpaBatch to recalculate GPA in bulk. Batch Apex processes records in chunks, avoiding governor limits. The execute() method handled GPA recalculation and record updates efficiently.



The screenshot shows the Salesforce Setup Apex Classes page. The search bar at the top contains "apex classe". The results list shows "Apex Classes" under "Custom Code". The "Apex Classes" section is selected. A search bar below it shows "apex classe". The main content area displays the "Apex Class Detail" for "GpaBatch". The class has a Name of "GpaBatch", a Namespace Prefix of "", and was Created By "Shravani Kalambe" on 9/25/2025, 1:42 PM. The Status is "Active" and the Active status is "0% (0/38)". The "Class Body" tab is selected, showing the Apex code:

```
1 global class GpaBatch implements Database.Batchable<SOBJECT> {
2     global Database.QueryLocator start(Database.BatchableContext BC) {
3         return Database.getQueryLocator([SELECT Id FROM Student__c]);
4     }
5
6     global void execute(Database.BatchableContext BC, List<Student__c> scope) {
7         Set<Id> studentIds = new Set<Id>();
8         for (Student__c s : scope) studentIds.add(s.Id);
9         if (studentIds.isEmpty()) return;
10
11        // 1) Get all fields Enrollment__c
12        Map<String, Schema.SObjectField> enrolFields = Schema.SObjectType.Enrollment__c.fields.getMap();
13        List<String> candidates = new List<String>{'Grade__c','Marks__c','Score__c','Marks_Obtained__c','Percentage__c','Mark__c'};
14        String gradeField = null;
15
16        for (String cand : candidates) {
17            if (cand == Schema.DisplayType.Double) {
18                Schema.DescribeFieldResult dfr = enrolFields.get(cand).getDescribe();
19                Schema.DisplayType dt = dfr.getType();
20                if (dt == Schema.DisplayType.Double) {
21                    ...
22                }
23            }
24        }
25    }
26}
```

## 9. Queueable Apex

(Not yet built, but could be added.) Queueable Apex is a lightweight async process. Example: publishing Low GPA alerts to external systems using System.enqueueJob() would be a Queueable use case.

## 10. Scheduled Apex

We implemented GpaScheduler to run the GPA Batch every Sunday at 3 AM. Scheduled Apex uses CRON expressions and helps automate repetitive jobs. This ensures GPA data stays fresh without manual runs.

The All Scheduled Jobs page lists all of the jobs scheduled by your users. Multiple job types may display on this page. You can delete scheduled jobs if you have the permission to do so.

**Percentage of Scheduled Jobs Used: 1%**  
You have Currently used 1 scheduled Apex jobs out of an allowed organization limit of 100 active or scheduled jobs. To learn about how this limit is calculated and what contributes to it see the [Lightning Platform Apex Limits](#) topic.

Action	Job Name	Submitted By	Submitted	Started	Next Scheduled Run	Type	Cron Trigger ID
Del	Metalytics Data Loader Job for Org : 00DgK00007anSn	User_Integration	7/17/2025, 1:54 PM	9/25/2025, 11:32 AM	9/26/2025, 11:32 AM	Autonomous Data Loader Job	08egK00007cbsG
	Program Milestone Computation Cron Job	Process_Automated	7/17/2025, 1:54 PM	9/25/2025, 11:59 PM	9/25/2025, 4:59 PM	Program Milestone Computation Cron Job	08egK00007cbsE
	Program Status Update Cron Job	Process_Automated	7/17/2025, 1:54 PM	9/25/2025, 5:00 AM	9/25/2025, 8:00 PM	Program Status Update Cron Job	08egK00007cbsF
Manage   Del   Pause Job	Weekly GPA Recalc	Kalambe_Shrawan	9/25/2025, 1:52 PM		9/28/2025, 3:00 AM	Scheduled Apex	08egK00000CGBfTU

## 10. Future Methods

(Not yet built, but relevant.) Future methods are used for asynchronous operations like sending callouts or heavy processing. For example, emailing students when GPA is recalculated could be done in a @future method.

## 11. Exception Handling

We used **try-catch blocks** in the LWC Apex controller (StudentController) to handle errors and return messages. Exception handling ensures the system doesn't crash and users see friendly error messages.

# Phase 6: User Interface Development

## 1. Lightning App Builder

We used **Lightning App Builder** to edit the **Student record page**. We dragged our custom LWC (student Progress) onto the layout.

Activated the page for the **Student Management App** so it shows to Faculty/Admin. Result: Users see the custom component when opening any Student record.

## 2. Record Pages

The Student object's **Record Page** was customized.

Default tabs (Details, Related) were kept as is, and the custom component was placed on the page. This ensures the Student record view is tailored with additional functionality (progress bar + button). Result: Record page now has both standard fields and the custom LWC.

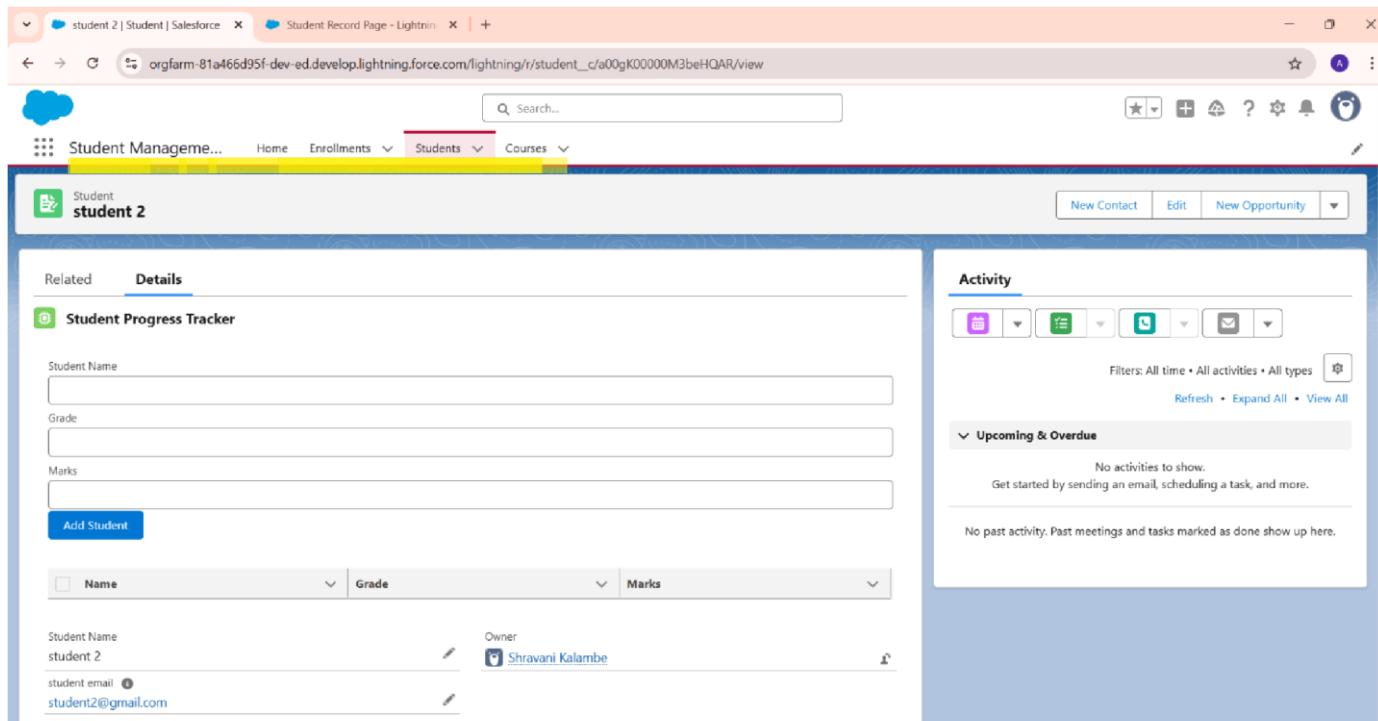
### 3. Tabs

In App Manager, we edited the Student Management app navigation. Added **Students, Courses, Enrollments** as navigation items.

The screenshot shows a Salesforce Student Record Page for a student named "student 2". The page includes a custom Lightning Web Component (LWC) titled "Student Progress Tracker" which contains fields for "Student Name", "Grade", and "Marks", along with an "Add Student" button. Below this, there is a table with columns for "Name", "Grade", and "Marks". Standard fields visible include "Owner" (Shravani Kalambe) and "Account". On the right side of the page, there is a "Activity" section with a "Upcoming & Overdue" summary and a "No past activity" message. The top navigation bar shows tabs for "Home", "Enrollments", "Students", and "Courses".

This gives Faculty/Admin direct access from the top navigation bar. Result: App navigation is organized and user-friendly.

## 4. Home Page Layouts



The screenshot shows the Salesforce Lightning Home Page for a student record. The top navigation bar includes links for 'Student Management', 'Home', 'Enrollments', 'Students' (which is the active tab), and 'Courses'. Below the navigation is a search bar and a toolbar with various icons. The main content area is titled 'Student Progress Tracker' and contains fields for 'Student Name', 'Grade', and 'Marks', along with a 'Add Student' button. To the right is an 'Activity' sidebar showing a list of activity types (Calendar, List, Task, Email) with a 'Filters' dropdown set to 'All time • All activities • All types'. Below this is a section titled 'Upcoming & Overdue' with a note: 'No activities to show. Get started by sending an email, scheduling a task, and more.' At the bottom of the sidebar, it says 'No past activity. Past meetings and tasks marked as done show up here.'

### X Not Done / Not Required for your project.

We didn't customize the Lightning Home Page layout (welcome screen/dashboard). Result: Home Page remains default.

## 5. Utility Bar

### X Not Done / Optional.

We did not add any **Utility Bar items** (like Notes, History, or custom components). Result: Utility Bar is not part of Phase 6 scope.

## 6. LWC (Lightning Web Components)

Created **studentProgress LWC**.

Includes GPA display, progress bar, and button for "Request Re-evaluation." Configured with an XML file so it's exposed only on **Student\_\_c Record Pages**.

Result: Fully working UI component embedded in Student record.

The screenshot shows the Salesforce Setup interface with the "Apex Classes" page open. The class "StudentProgressAPI" is displayed, showing its code:

```

1  @RestResource(urlMapping="/studentprogress")
2  global with sharing class StudentProgressAPI {
3      @HttpGet
4      global static Student__c getStudent() {
5          RestRequest req = RestContext.request;
6          String studentId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
7
8          return
9              SELECT Id, Name, GPA__c, DOB_c__c
10             FROM Student__c
11            WHERE Id = :studentId
12            LIMIT 1
13      }
14  }
15

```

The code editor also shows the deployment logs for the "studentProgress" component, indicating successful deployment to the org.

## 7. Apex with LWC

Built StudentController.cls Apex class.

Connected it to the LWC via **imperative Apex call** (button → Apex method).

Handles the re-evaluation request and returns success/error messages. Result:  
UI → Server → UI round-trip working properly.

## 8. Events in LWC

### ✗ Not Done / Not Required.

We didn't use custom events between components (e.g., child → parent communication).

Only **standard ShowToastEvent** was used to show notifications. Result: Events beyond toast handling not needed for now.

## 9. Wire Adapters

Used @wire(getRecord, { recordId: '\$recordId', fields: [GPA\_FIELD] }).

Automatically fetches the **GPA\_c** field for the Student record.

Displayed value in UI and linked it to progress bar dynamically. Result: Data loads reactively without manual calls.

## 10. Imperative Apex Calls

Button click calls requestReevaluation({ studentId: this.recordId }).

Shows toast message for success/failure.

Imperative Apex is triggered only on user action (button). Result: On-demand server interaction built successfully.

## 11. Navigation Service

### ✗ Not Done / Not Required.

We did not implement **NavigationMixin** (e.g., navigating to another record, page, or app). Result: No navigation actions configured in LWC.

### Summary

- **Done:** Lightning App Builder, Record Pages, Tabs, LWC, Apex with LWC, Wire Adapters, Imperative Apex Calls.
- **Not Applicable / Skipped:** Home Page Layouts, Utility Bar, Events in LWC (custom ones), Navigation Service.

# Phase 7 – Integration & Events

## 1. Named Credentials

We created a **Named Credential** called **CertVerifier** to securely store the base URL of an external API for certificate verification.

This lets Apex use the alias `callout:CertVerifier` instead of hard-coding full URLs or authentication details.

For testing, we used a public API (JSONPlaceholder) with **No Authentication**. This improves security and makes callouts easier to maintain.

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

- Tab Bar:** Includes 'Smart tracker setup guide', 'Lightning Experience | Salesforce', and 'Named Credentials | Salesforce'.
- Page Header:** Shows the URL 'orgfarm-81a466d95f-dev-ed.develop.lightning.force.com/lightning/setup/NamedCredential/0XAgKD0000009lQj/view'.
- Search Bar:** 'Search Setup'.
- Left Sidebar:** 'Setup' tab selected, with sections like 'Quick Find', 'Setup Home', 'Service Setup Assistant', 'Commerce Setup Assistant', etc., and 'ADMINISTRATION' and 'PLATFORM TOOLS' sections.
- Current View:** 'SETUP > NAMED CREDENTIALS' for 'CertVerifier'.
  - Label:** CertVerifier
  - Name:** CertVerifierEC
  - URL:** https://api.example.com
  - Enabled for Callouts:** Checked.
  - Authentication:** External Credential set to 'Cert Verifier'.
  - Callout Options:** 'Generate Authorization Header' is checked.

## 2. Remote Site Settings

As a fallback for environments without Named Credentials, we also created a **Remote Site Setting** with the same API base URL.

This step ensures Salesforce can connect to external endpoints when Named Credentials are not used. In our demo, we relied on the Named Credential, but the Remote Site is a backup to avoid callout-blocking errors.

## 3. Web Services (REST) & Callouts

We implemented a simple Apex class **CertVerifierService** to perform a REST callout to the external API using the Named Credential.

The class sends an HTTP GET request and returns the API's response.

We tested the callout in **Developer Console → Execute Anonymous** and confirmed we could retrieve live JSON data.

This proves that our Salesforce app can communicate with external services.

## 4. Platform Events

We created a **Platform Event** named **LowGPAAlert\_\_e** with fields for **StudentId**, **GPA**, and **Message**. Whenever a student's GPA drops below 2.0, our existing **Student Trigger** publishes this event. This allows downstream systems or even Salesforce triggers to react in near real-time to academic performance issues.

It decouples internal logic from external listeners.

The screenshot shows the Salesforce Platform Events setup page. The left sidebar has a search bar and navigation links for MuleSoft, Einstein, Custom Code, Integrations, Security, and Platform Encryption. The 'Platform Events' link under 'Integrations' is highlighted. The main content area has a title 'Platform Events' and a subtitle 'Use platform events to define the data to be delivered in custom notifications. Monitor the publishing and delivery usage for platform events and change events.' It includes two tables: 'Event Allocations' and 'Custom Events'.

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

Action	Label	Deployed	Description
Edit   Del	LowGPAAlert	✓	
Edit   Del	gpa.alert	✓	

## 5. Change Data Capture (CDC)

We enabled **Change Data Capture** for the **Student\_\_c** and **Enrollment\_\_c** objects. This feature automatically publishes record-change events whenever these objects are created, updated, or deleted. We tested it by subscribing to `/data/StudentChangeEvent` in **Workbench** and verified that updates in Salesforce trigger real-time events. CDC ensures external systems stay synchronized without manual exports.

### End Result:

Phase 7 equips our app with secure external integration via Named Credentials and REST callouts, plus real-time messaging through **Platform Events** and **CDC**, enabling smooth two-way communication with external systems and better monitoring of student progress.

## Phase 8: Data Management & Deployment

### 1) Prepare CSVs

Create three CSVs: **Students**, **Courses**, **Enrollments**.

Each should have an **External ID column** (unique text field).

Ensure clean data: no formulas, saved as UTF-8.

Verify headers and sample rows in a text editor before upload.

The screenshot shows a Microsoft Excel spreadsheet titled "Enrollment.csv". The data is organized into columns: Enrollment, StudentID, CourseID, Status, and RequestDate. The status column contains values like "Enrolled" and "Pending", while the RequestDate column shows several "# #####" entries.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Enrollment	StudentID	CourseID	Status	RequestDate																		
2	ENR001	STU001	CSE101	Enrolled	#####																		
3	ENR002	STU002	MTH201	Pending	#####																		
4	ENR003	STU003	PHY301	Enrolled	#####																		
5																							
6																							
7																							
8																							
9																							
10																							
11																							
12																							
13																							
14																							
15																							
16																							
17																							
18																							
19																							
20																							
21																							
22																							
23																							
24																							
25																							
26																							
27																							

## 2) Data Import Wizard

Use for **small/simple loads** (students, courses).

Setup → Data Import Wizard → choose object.

Upload CSV, map headers → Salesforce fields.

Run import, check email + sample records.

The screenshot shows the Salesforce Setup interface with the Bulk Data Load Jobs page open. The job ID is 750gK00000DrzSU. The job was submitted by Shravani Kalambe and completed successfully. It processed 301 records in 215 ms using API Active Processing Time. The job used Parallel Concurrency Mode and CSV Content Type. The progress was 100% complete with 3 records processed and 3 failed. The Apex Processing Time was 75 ms.

Job ID	750gK00000DrzSU	Submitted by	Shravani Kalambe	Job Type	Bulk V1	Status	Closed
Start Time	9/26/2025, 3:27 AM PST	Queued Batches	0	Operation	Upsert	Total	301
End Time	9/26/2025, 3:27 AM PST	In Progress Batches	0	API Active Processing Time (ms)		215	
Time to Complete (hh:mm:ss)	00:01	Completed Batches	1	Apex Processing Time (ms)		75	
Object	Student	Failed Batches	0				
External ID Field	Name	Progress	100%				
Content Type	CSV	Records Processed	3				
Concurrency Mode	Parallel	Records Failed	3				
API Version	64.0	Retries	0				

### 3) Data Loader

Best for **large or complex loads** (like enrollments with lookups).

Download & install from Setup → Data Loader.

Use **Insert/Update/Upinsert** with CSV and mapping.

Review success & error CSVs after run.

The screenshot shows the Salesforce Data Loader application. A CSV Viewer window is open, displaying a table of data with columns: Row Number, ID, CourseExternalId, Name, Course\_Code, Capacity, Mode, STAT, and Status. The status column shows several rows with a value of 'Success'. At the bottom of the CSV viewer, there are buttons for Hard Delete, Export, and Export All. A tooltip indicates that clicking the 'Open in external program' button will open the CSV in Microsoft Excel. The main Data Loader interface has a sidebar with options like Process, Import, and Export, and a central area with the Salesforce logo and the text 'data loader'.

## 4) Duplicate & Matching Rules

Prevent duplicate Students in the system.

Create **Matching Rule** (e.g., match Email or Name+DOB).

Create **Duplicate Rule** to block or alert on duplicates.

Test by inserting a duplicate → confirm block/alert works.

The screenshot shows the Salesforce Setup interface with the 'Matching Rules' page open. The URL in the browser is <https://orgfarm-81a466d95f-dev-ed.lightning.force.com/lightning/setup/MatchingRules/page?address=%2F0JDgK000004CzSR>. The page title is 'Matching Rules'. A matching rule named 'match email' is displayed, detailing its configuration: Object: Student, Rule Name: match\_email, Unique Name: match\_email, Matching Criteria: Student: student\_email exact MatchBlank = FALSE, Status: Activating, Created By: Shravani Kalambe, and Modified By: Shravani Kalambe. The page includes a sidebar for Data and Duplicate Management, and a search bar at the top.

## 5) Data Export & Backup

Always back up before big imports/changes.

Setup → Data Export → choose objects → Export Now.

Download and unzip CSV package securely.

Schedule weekly exports for safety.

## 6) Change Sets

Use to move metadata Sandbox → Production.

Create **Outbound Change Set** → add components (objects, fields, Apex, LWC).

Upload, then deploy in **Inbound Change Sets** in target org.

Validate first, then deploy and test.

## 7) VS Code & CLI

Best for source-driven development.

Install VS Code + Salesforce CLI, create a project.

Retrieve metadata (sf project retrieve) and deploy (sf project deploy). Use Git for version control, test before deploying.

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>59.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```

## 8) Unmanaged vs Managed Packages

**Unmanaged:** one-time copy of components (internal use).

**Managed:** with namespace + versioning (AppExchange).

For internal projects, mainly use Change Sets or CLI.

Packages are optional, not required.

# Phase 9: Reporting, Dashboards & Security Review

## Reports

Reports let you analyze Salesforce data in different formats: Tabular (simple lists), Summary (grouped rows), Matrix (rows & columns), and Joined (combine multiple blocks). You build them in the Report Builder by selecting fields, filters, and groupings, then save and share.

## Report Types

A Report Type defines which objects and fields are available when building a report. You select a Primary Object (like Students or Enrollments), then add related objects to expose more data. Custom report types let you tailor reporting to your org's data model.

**Custom Report Types**

**Enrollments with Students and Courses**

Below is the information for this custom report type. You can click the buttons on this to preview or update information for the custom report type.

**Details**

Display Label	Enrollments with Students and Courses
API Name	Enrollments_with_Students_and_Courses
Description	Use this report type to view and analyze Enrollment records along with related Student and Course information.
Created By	Shravani Kalambe, 9/26/25, 9:25 PM
Store in Category	other
Deployment Status	Deployed
Modified By	Shravani Kalambe, 9/26/25, 9:25 PM

**Object Relationships**

Enrollments (A) ... with at least one related record from Duplicate Record Items (B)

## Dashboards

Dashboards visualize reports with charts, metrics, gauges, and tables in one view. Each component is powered by a saved report and can be resized, arranged, and filtered. They give managers and users an at-a-glance view of key business indicators.

### Dynamic Dashboards

Dynamic Dashboards show data based on the logged-in user's security and sharing settings. This ensures each person only sees the records they are allowed to access. They are especially useful for role-based organizations with varying data visibility needs.

## Sharing Settings

Sharing Settings control record visibility across users. You set Org-Wide Defaults (OWD) first, then open up access with Roles, Public Groups, and Sharing Rules. This ensures data is kept private by default but shared with the right

Object	Sharing Rule Access	Record Type	Status
User Provisioning Request	Private	Private	✓
Waitlist	Private	Private	✓
Web Cart Document	Private	Private	✓
Work Order	Private	Private	✓
Work Plan	Private	Private	✓
Work Plan Template	Private	Private	✓
Work Step Template	Private	Private	✓
Work Type	Private	Private	✓
Work Type Group	Public Read/Write	Private	✓
Course	Public Read Only	Private	✓
Enrollment	Private	Private	✓
mentor	Public Read/Write	Private	✓
Student	Private	Private	✓
Student	Public Read/Write	Private	✓

people.

## Field-Level Security (FLS)

FLS restricts access to individual fields on objects. You can make fields visible, read-only, or hidden for specific profiles or permission sets. This protects sensitive information, like emails or personal IDs, from being exposed unnecessarily.

The screenshot shows the Salesforce Setup interface. The left sidebar lists various setup categories: Setup Home, Salesforce Go, Service Setup Assistant, Commerce Setup Assistant, Field Service Setup Home (Beta), Hyperforce Assistant, Release Updates, Salesforce Mobile App, Lightning Usage, Optimizer, Sales Cloud Everywhere, ADMINISTRATION (with sub-options for Users, Data, Email), and PLATFORM TOOLS (with sub-option for Subscription Management). The main content area is titled "Set Field-Level Security" and "Student External ID". It shows the field label "Student External ID" and data type "Text(50) (External ID) (Unique Case Insensitive)". Below this is a table titled "Field-Level Security for Profile" with columns for "Visible" (checkboxes) and "Read Only" (checkboxes). The table lists several profiles, all of which have the "Visible" checkbox checked and the "Read Only" checkbox unchecked. The profiles listed are: Analytics Cloud Integration User, Analytics Cloud Security User, Anypoint Integration, Contract Manager, Cross Org Data Proxy User, Custom: Marketing Profile, Custom: Sales Profile, Custom: Support Profile, Einstein Agent User, Faculty, and Force.com - Ann Subscription User.

Field-Level Security for Profile	Visible	Read Only
Analytics Cloud Integration User	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Analytics Cloud Security User	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Anypoint Integration	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Contract Manager	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Cross Org Data Proxy User	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Custom: Marketing Profile	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Custom: Sales Profile	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Custom: Support Profile	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Einstein Agent User	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Faculty	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Force.com - Ann Subscription User	<input checked="" type="checkbox"/>	<input type="checkbox"/>

## Session Settings

Session Settings define how long users stay logged in and how sessions are secured. You can set timeouts, lock sessions to IP addresses, and control persistent logins. These settings reduce the risk of unauthorized access if devices are left unattended.

The screenshot shows the Salesforce Setup interface with the 'Session Settings' tab selected. The page title is 'Session Settings'. It includes sections for 'Session Timeout' (set to 2 hours) and 'Session Settings' (with several checkboxes like 'Lock sessions to the domain in which they were first used' checked). A note at the bottom states: '\*\*EXTENDED USE OF IE11 WITH LIGHTNING EXPERIENCE HAS NOW ENDED\*\* AS OF DECEMBER 31, THE EXTENDED PERIOD HAS ENDED, AND USE OF INTERNET EXPLORER 11 (IE 11) WITH LIGHTNING EXPERIENCE IS NO LONGER SUPPORTED. ISSUES WITH PERFORMANCE OR FUNCTIONALITY THAT AFFECT ONLY IE 11 WILL NOT BE FIXED. PLEASE SWITCH TO A SUPPORTED BROWSER.

## Login IP Ranges

Login IP Ranges restrict where users can log in from. At the org level, trusted IPs bypass security tokens; at the profile level, strict ranges ensure logins only come from approved networks. This adds an extra layer of security, especially for remote access.

## Audit Trail

Audit Trail helps track system and data changes. Setup Audit Trail records configuration changes (who, when, what), while Field History Tracking logs changes to selected fields. This supports compliance, accountability, and troubleshooting by showing change history.