

Phase 1. Project Title

SN SmartLearn - Student & Course Management System

2. Problem Statement

An online education platform is currently managing student applications, course enrollments, and communications through a fragmented system of spreadsheets and emails. This manual process is inefficient, prone to error, and lacks a centralized view of student data. As the platform grows, this approach is unsustainable, making it difficult to provide a quality student experience, track enrollment trends, and scale operations effectively.

The company requires a robust Salesforce CRM solution to overcome these challenges.

3. Objectives

The primary goals of this Salesforce implementation are to:

- **Automate** the student application and enrollment process to minimize manual errors.
- **Centralize** all student, course, and progress data into a single source of truth.
- **Track** student progress, course history, and assessment results effectively.
- **Streamline** communications with students, instructors, and the admissions team.
- **Enable** real-time dashboards and reports for management to monitor key metrics like enrollment and retention.

4. Stakeholder Analysis

The key stakeholders and their primary needs are identified as follows:

- **Admissions Team:** Needs an efficient system for tracking applications and reducing manual data entry.
- **Course Instructors:** Require easy access to student enrollment lists and progress data.
- **Students:** Expect a smooth, transparent enrollment process and timely, relevant communication.
- **Management:** Wants clear visibility into the admissions funnel, course popularity, and student retention rates for strategic decision-making.
- **IT/Admin:** Responsible for ensuring system stability, data integrity, and security.

2. Business Process Mapping

A comparison of the current and proposed business processes highlights the intended improvements.

Current Process (Before Salesforce)

1. A prospective student submits an application via a web form.
2. An administrator manually enters the application data into a spreadsheet.
3. The admissions team reviews applications from the shared spreadsheet.
4. All communication (updates, requests) is handled via individual emails, which are difficult to track.
5. Course enrollment and progress are logged in separate, disconnected documents.

Proposed Process (After Salesforce Implementation)

1. A student's application from the web form is **automatically captured** as a Lead record in Salesforce.
2. An automated workflow assigns the application, creates follow-up tasks, and updates its status.
3. Once approved, the Lead is converted into Contact (Student), Account (if applicable), and custom Enrollment records.
4. Automated welcome emails and deadline reminders are sent to students via email alerts.
5. All student data, course history, and progress are tracked in a unified, 360degree view.

3. Industry-Specific Use Case Analysis

The EdTech industry has unique requirements that this project will address:

- **Student Enrollment:** Automatically capture applications from web forms and track the status from submission to enrollment.
- **Course Management:** Maintain a centralized inventory of all courses, including details on modules and assigned instructors.
- **Student Progress Tracking:** Utilize custom objects to log student progress, assignment completion, and grades.
- **Cohort Management:** Group students by program or start date for targeted communication and specialized reporting.
- **Alumni Relations:** Build a foundation to manage relationships with graduates for future engagement and networking opportunities.

4. AppExchange Exploration

To enhance functionality, we will explore solutions on the Salesforce AppExchange:

- **Form Integration Apps (e.g., FormAssembly, Formstack):** To build complex web forms that map directly to Salesforce objects for seamless data capture.
- **Document Generation (e.g., Conga, Docusign):** For automatically generating and sending enrollment agreements or completion certificates.
- **Enhanced Notification Apps (e.g., Twilio):** To implement SMS/WhatsApp notifications for critical reminders and updates.

5. Conclusion

This initial analysis confirms that a Salesforce CRM implementation is the ideal solution to address SN-SmartLearn's challenges. The project will automate manual processes, create a centralized data system, and provide the analytical tools needed to scale operations and enhance the overall student experience.

Phase 2 - Org Setup & Configuration

1. Salesforce Editions

We used a **Salesforce Developer Edition Org** for this implementation. This edition was selected because it provides all the core CRM features required for our project, such as custom objects, roles, profiles, automation tools, and APIs. It also supports AppExchange integration, which we plan to explore in later phases.

2. Company Profile Setup

- Setup → **Company Information**
- Updated **Organization Name** to SN SmartLearn.
- Set **Default Currency** as INR (Indian Rupees).
- Configured **Locale** as English (India) to ensure formatting of numbers, currency, and dates as per Indian standards.
- Set **Time Zone** to (GMT+5:30) Asia/Kolkata.

This ensures consistency across all student and instructor records, communications, and reports.

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

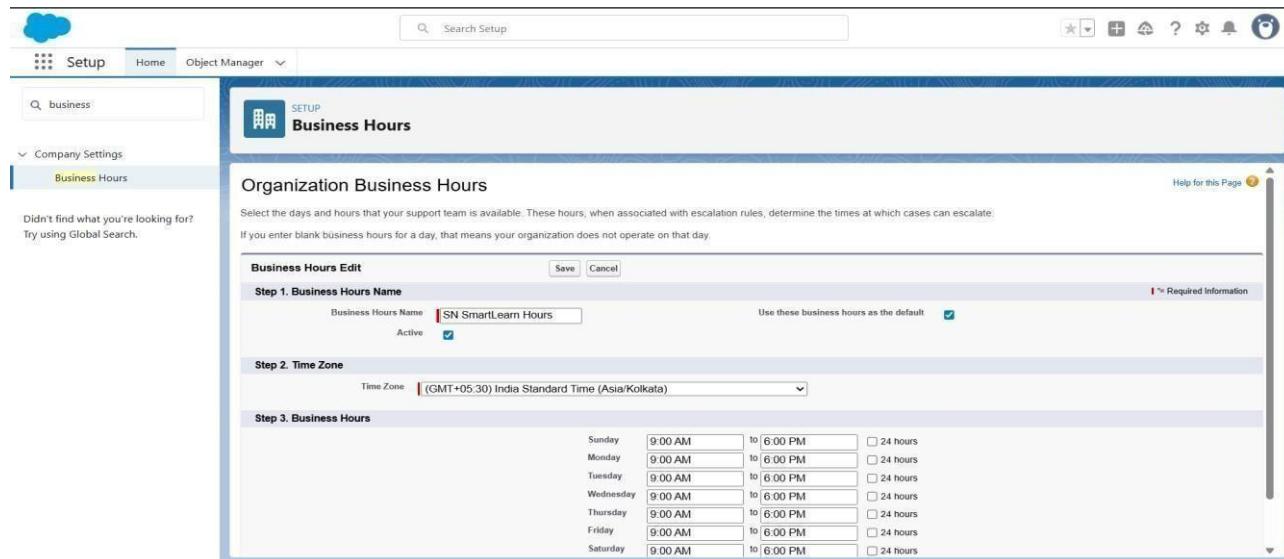
- Setup** tab selected in the top navigation bar.
- Company Information** page displayed.
- Organization Detail** section:
 - Organization Name: SN SmartLearn
 - Primary Contact: Nishant Dubey
 - Division: Madhya Pradesh
 - Address: India
 - Fiscal Year Starts In: April
 - Activate Multiple Currencies:
 - Enable Data Translation:
 - Newsletter:
 - Admin Newsletter:
 - Hide Notices About System Maintenance:
 - Hide Notices About System Downtime:
 - Locale Formats: ICU
- Phone** and **Fax** fields are empty.
- Default Locale**: English (India)
- Default Language**: English
- Default Time Zone**: (GMT+05:30) India Standard Time (Asia/Kolkata)
- Corporate Currency**: Indian Rupee
- Used Data Space**: 380 KB (7%) ([View](#))
- Used File Space**: 19 KB (0%) ([View](#))
- API Requests, Last 24 Hours**: 0 (15,000 max)
- Streaming API Events, Last 24 Hours**: 0 (10,000 max)
- Restricted Logins, Current Month**: 0 (0 max)
- Salesforce.com Organization ID**: 00Dg.000007VtE
- Organization Edition**: Developer Edition
- Instance**: CAN98

Page footer: Created By: OrgFarm EPIC, 7/18/2025, 4:16 AM Modified By: Nishant Dubey, 9/20/2025, 4:21 AM

3. Business Hours & Holidays

- Setup → **Business Hours** → Created SN SmartLearn Hours as **9:00 AM to 6:00 PM (Mon–Fri)**.
- Setup → **Holidays** → Added major holidays like **Diwali, Republic Day, Independence Day, and New Year**.

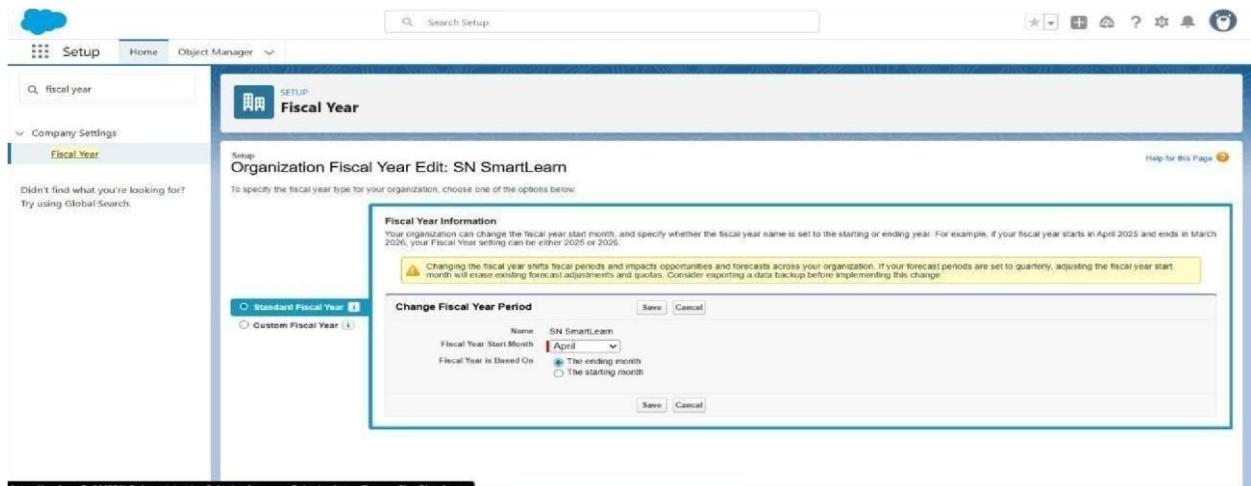
These settings ensure that automation processes like case escalations, reminders, and email alerts respect the organization's working schedule.



4. Fiscal Year Settings

- Setup → Fiscal Year
- Configured Standard Fiscal Year (April–March) to align with the Indian academic and financial cycle.

This setup ensures reporting and dashboards for admissions, enrollments, and revenue match the organization's fiscal planning.



5. User Setup & Licenses

We created different users to represent key stakeholders of the system:

- **Admissions Officer** – Responsible for managing student applications and enrollment.
- **Course Instructor** – Access to course records, enrolled student lists, and progress data.
- **Student (Test User)** – Limited access to check the student experience.

Each user was assigned appropriate licenses (Salesforce / Salesforce Platform) depending on their responsibilities.

6. Profiles

We created custom profiles by cloning the **Standard User Profile** and tailoring objectlevel permissions:

- **Admissions Profile** – Full access to Leads, Contacts, and Enrollment objects.
- **Instructor Profile** – Access to Course and Student Progress objects.

- **Student Profile** – Read-only access to their own course and progress records.

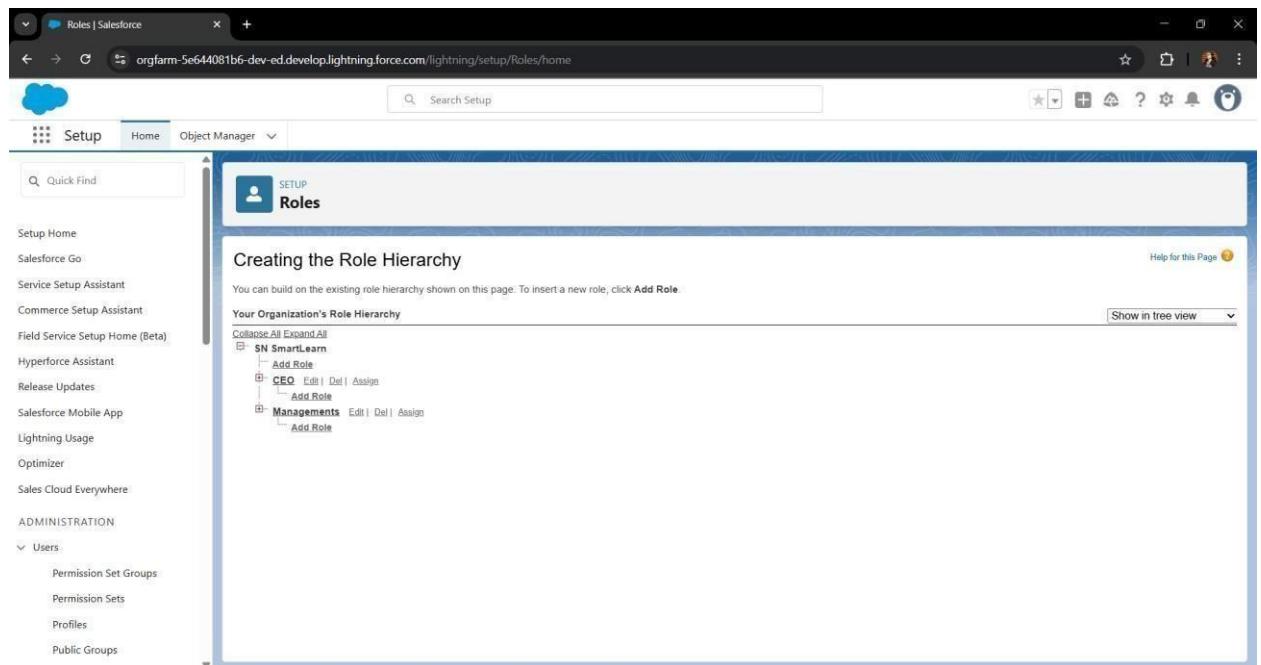
Profiles ensure each role has the exact level of access needed to perform their duties, reducing risks of unauthorized data exposure.

7. Roles

Setup → **Roles** → Created a hierarchy to control data visibility:

- **Management (Top)** ○ **Admissions Head** ○ **Course Instructor** ○ **Students**

This hierarchy ensures managers and admissions heads can view all related data, while instructors and students see only what is relevant to them.



8. Permission Sets

To provide additional, flexible access without altering profiles, we created:

- **Progress Tracking Access** → For instructors to log and monitor student progress.

- **Report Viewer** → For management to access analytical dashboards.

Permission Sets give fine-grained control and can be assigned on a need basis.

9. Org-Wide Defaults (OWD)

Setup → **Sharing Settings** → Configured the following:

- **Students** → Private (students can only view their own records).
- **Courses** → Public Read/Write (so instructors and admins can update them).
- **Enrollments** → Controlled by Parent (data visibility depends on related student/course record).

This enforces data security and ensures confidentiality of student records.

10. Sharing Rules

We implemented sharing rules for controlled data access:

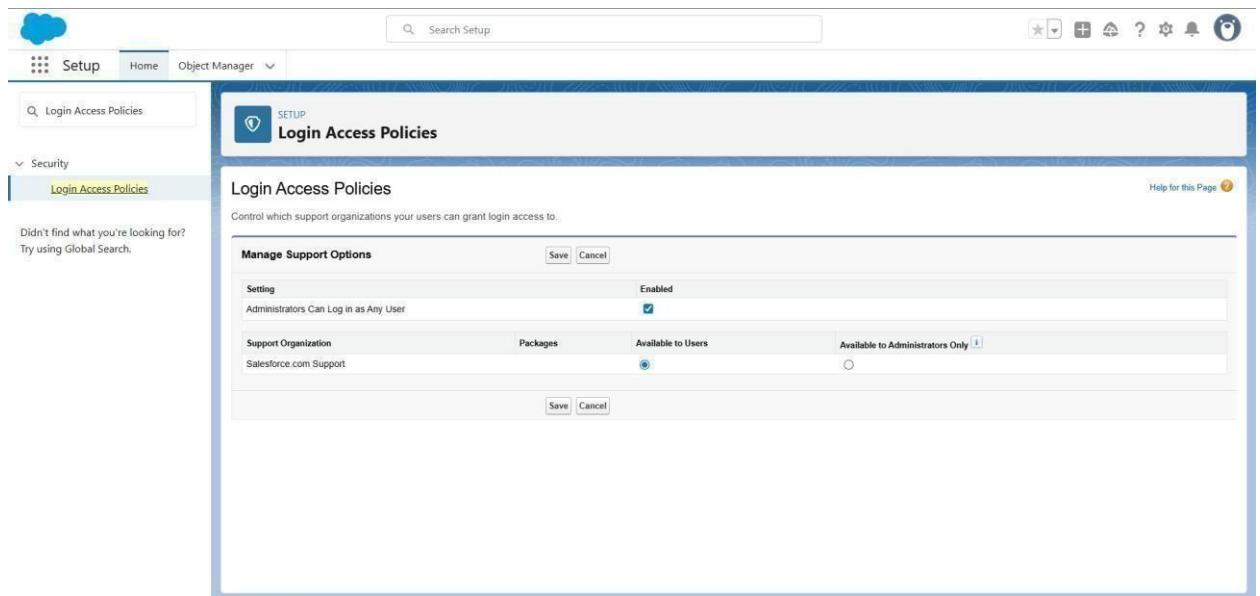
- Admissions users can access all student records to process applications.
- Instructors only see records of students enrolled in their assigned courses.

This prevents unnecessary exposure of sensitive data while enabling collaboration.

11. Login Access Policies

- Setup → **Login Access Policies**
- Enabled **Administrators Can Log in as Any User** to simplify troubleshooting and support. For example, the admin can log in as a student to check if course enrollment processes are working correctly.
- Enabled **Salesforce.com Support Login Access** to allow Salesforce support teams to securely access the org in case of technical issues.

This ensures quick issue resolution and strong governance during system operations.



12. Developer Org Setup

- Create Salesforce Developer Edition account.
- Configure Company Profile, Users, Roles, Profiles, Business Hours, and Security settings.
- Enable required features: custom objects, automation, reports.
- Integrate with GitHub/Salesforce CLI for version control.

13. Sandbox Usage

- Use Developer Sandbox for building and testing changes safely.
- Optionally, use Full Sandbox for testing production-level scenarios.
- Always test major changes in a sandbox before deploying to production.

14. Deployment Basics

- Change Sets: Simple point-and-click deployment between orgs.
- Salesforce CLI (SFDX): Advanced deployment with version control and automation.
- GitHub Integration: Track changes, collaborate, and maintain version control.
- Always document deployment steps and maintain backups

Phase 3 : Data Modeling & Relationships

1. Standard & Custom Objects

This is the foundation of your system. You will use a combination of standard and custom objects.

- **Standard Objects to Use:**

- **Lead:** This will be used to automatically capture a student's application from your web form.
- **Contact:** This will represent the student record after their application is approved and the Lead is converted.
- **Account:** This can be used to represent the student's household or a sponsoring organization, created during Lead conversion.

The screenshot shows the Salesforce Object Manager interface. At the top, there are tabs for Setup, Home, and Object Manager. A search bar is at the top right. Below the header, the title 'Object Manager' is displayed with a note '1 Items. Sorted by Label'. A search bar contains the text 'lead'. The main area is a table with columns: LABEL, API NAME, TYPE, DESCRIPTION, LAST MODIFIED, and DEPLOYED. One item is listed: 'Lead' (API Name: Lead, Type: Standard Object). The table has a light blue header row and white data rows.

Custom Objects to Create:

- **Course:** This is required to maintain a centralized inventory of all courses offered.

- **Enrollment:** This custom object will be created upon Lead conversion to link a student to a specific course.
- **Student Progress:** This object is necessary to log assignment completion and grades, enabling effective progress tracking.

The screenshot shows the Salesforce Object Manager interface. At the top, there's a navigation bar with 'Setup', 'Home', and 'Object Manager'. A search bar says 'Search Setup' and a filter icon is present. Below the header, the title 'Object Manager' is displayed with a subtitle '1 Items. Sorted by Label'. A search bar at the top right contains the text 'course'. On the right side of the header, there are buttons for 'Schema Builder' and 'Create'. The main area is a table with the following columns: 'LABEL', 'API NAME', 'TYPE', 'DESCRIPTION', 'LAST MODIFIED', and 'DEPLOYED'. There is one row for the 'Course' object, which has the API name 'Course__c', is a 'Custom Object', and was last modified on '9/20/2025'.

LABEL	API NAME	TYPE	DESCRIPTION	LAST MODIFIED	DEPLOYED
Course	Course__c	Custom Object		9/20/2025	✓

2. Fields

These are the specific data points you will track on each object.

Action Steps:

1. Navigate to **Setup > Object Manager**.
2. Select each custom object (Course, Enrollment, Student Progress) and use the **Fields & Relationships** section to add the following fields:
 - **On the Course object:**
 - Course Code (Data Type: Text, **Unique**)
 - Instructor (Data Type: Lookup to **User**)
 - Status (Data Type: Picklist; Values: Active, Planned, Archived)
 - **On the Enrollment object:**
 - Enrollment Date (Data Type: Date)

- Status (Data Type: Picklist; Values: Applied, Enrolled, In Progress, Completed, Dropped)
- **On the Student Progress object:**
 - Assessment Type (Data Type: Picklist; Values: Quiz, Assignment, Final Exam)
 - Grade (Data Type: Percent)
 - Submission Date (Data Type: Date)

Student Progress | Salesforce

orgfarm-5e644081b6-dev-ed.lightning.force.com/lightning/setup/ObjectManager/01lgL000002lrPh/FieldsAndRelationships/view

Setup Home Object Manager

SETUP > OBJECT MANAGER
Student Progress

Details

Fields & Relationships
6 items, Sorted by Field Label

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedById	Lookup(User)		
Currency	CurrencyIsoCode	Picklist		
Enrollment	Enrollment__c	Lookup(Enrollment)		
Last Modified By	LastModifiedById	Lookup(User)		
Owner	OwnerId	Lookup(User,Group)		
Progress ID	Name	Auto Number		

Page Layouts
Lightning Record Pages
Buttons, Links, and Actions
Compact Layouts
Field Sets
Object Limits
Record Types
Related Lookup Filters
Restriction Rules
Scoping Rules
Object Access
Triggers

Enrollment | Salesforce

orgfarm-5e644081b6-dev-ed.lightning.force.com/lightning/setup/ObjectManager/01lgL000002lrMT/FieldsAndRelationships/view

Setup Home Object Manager

SETUP > OBJECT MANAGER
Enrollment

Details

Fields & Relationships
6 items, Sorted by Field Label

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Course	Course__c	Master-Detail(Course)		
Created By	CreatedById	Lookup(User)		
Currency	CurrencyIsoCode	Picklist		
Enrollment Number	Name	Auto Number		
Last Modified By	LastModifiedById	Lookup(User)		
Student	Student__c	Master-Detail(Contact)		

Page Layouts
Lightning Record Pages
Buttons, Links, and Actions
Compact Layouts
Field Sets
Object Limits
Record Types
Related Lookup Filters
Restriction Rules
Scoping Rules
Object Access
Triggers

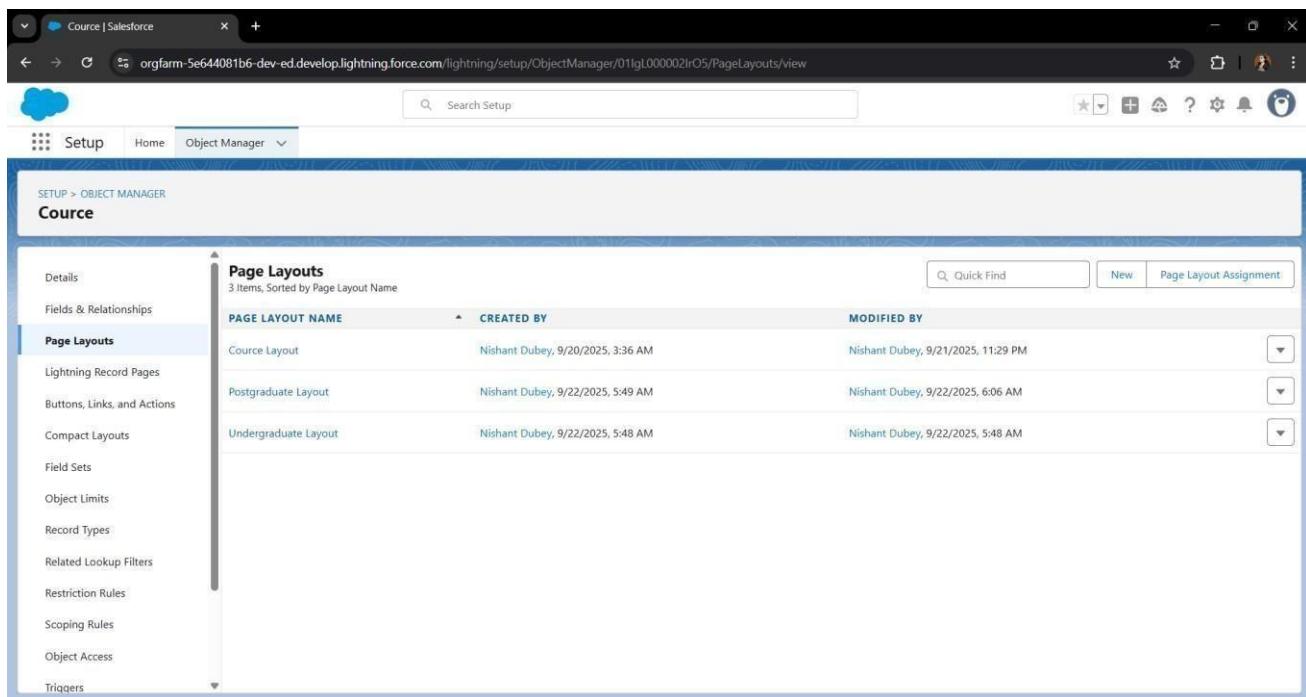
3. Record Types & Page Layouts

These allow you to customize the user experience for different processes. For example, you can create different page layouts on the

Contact object for a prospective student vs. an enrolled student to support the needs of the Admissions Team and Instructors.

Action Steps:

1. Navigate to the **Contact** object and create two **Page Layouts**: one named "Undergraduate Layout" and another named "Postgraduate Layout".
2. On the **Contact** object, go to **Record Types** and create a new record type: "Thesis Required", assigning the corresponding layout.



The screenshot shows the Salesforce Setup interface for the 'Course' object. The left sidebar lists various configuration options under 'Page Layouts'. The main area displays a table titled 'Page Layouts' with three items. The columns are 'PAGE LAYOUT NAME', 'CREATED BY', and 'MODIFIED BY'. The data is as follows:

PAGE LAYOUT NAME	CREATED BY	MODIFIED BY
Course Layout	Nishant Dubey, 9/20/2025, 3:36 AM	Nishant Dubey, 9/21/2025, 11:29 PM
Postgraduate Layout	Nishant Dubey, 9/22/2025, 5:49 AM	Nishant Dubey, 9/22/2025, 6:06 AM
Undergraduate Layout	Nishant Dubey, 9/22/2025, 5:48 AM	Nishant Dubey, 9/22/2025, 5:48 AM

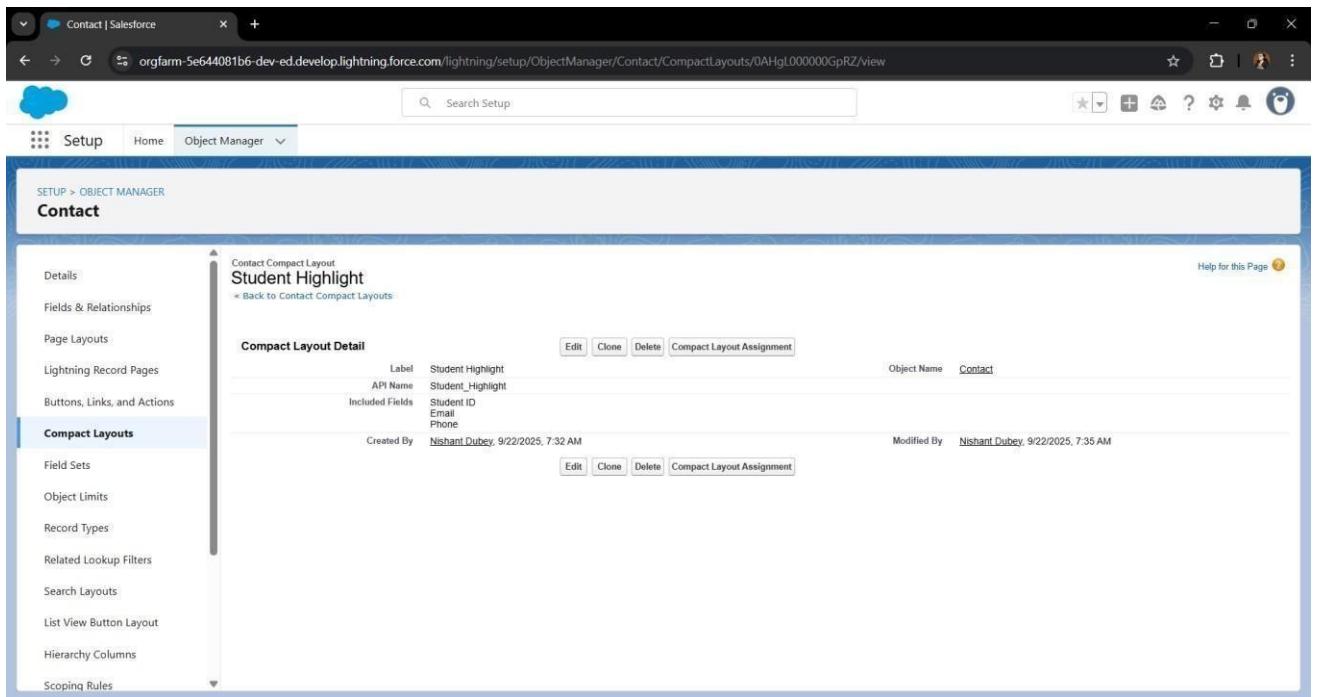
4. Compact Layouts

This controls the highlights panel at the top of a record.

Action Steps:

1. Navigate to the **Contact** object and go to **Compact Layouts**.

2. Create a new layout named "Student View".
3. Add key fields like **Name**, **Email**, and **Phone**.
4. Use **Compact Layout Assignment** to make this the primary layout.

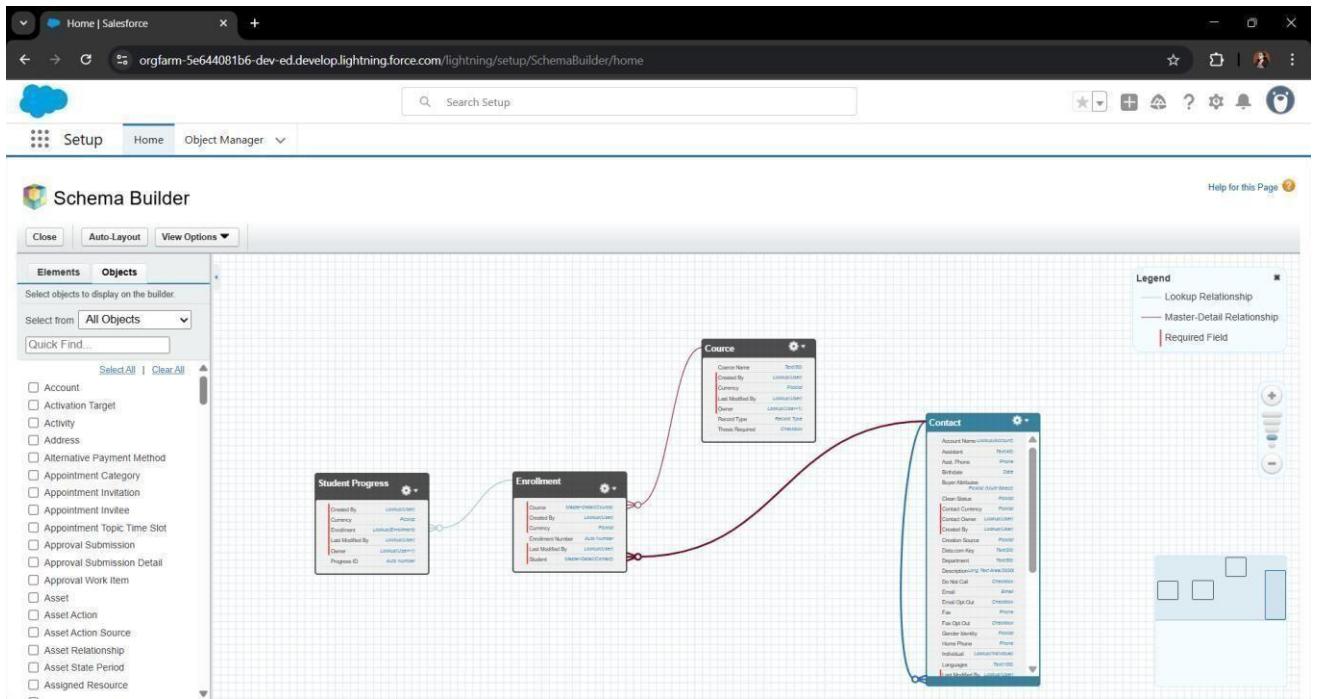


5. Schema Builder

Use this tool to visualize your completed data model.

Action Steps:

1. Go to **Setup > Schema Builder**.
2. Select your objects: **Lead**, **Contact**, **Account**, **Course**, **Enrollment**, and **Student Progress**.
3. Review the diagram to visually confirm the relationships you built.



6. Relationships & Junction Objects

These relationships will connect your objects to create the 360-degree view of the student mentioned in your project plan.

- **Junction Object:** Your **Enrollment** object is the junction object. It connects Students (Contacts) and Courses, creating a many-to-many relationship.

Action Steps:

1. On the **Enrollment** object, create two **Master-Detail Relationship** fields:
 - One that links to the **Contact** object (label it **Student**). ◦ A second one that links to the **Course** object (label it **Course**).
2. On the **Student Progress** object, create a required **Lookup Relationship** that links to the **Enrollment** object.

7. External Objects

This is a conceptual topic for this project. External Objects allow you to view data from other systems. For example, if your platform used an external library management system, you could create an External Object to display a student's checked-out books within Salesforce without actually storing that data.

Phase 4 : Process Automation (Admin)

In this phase, I implemented Salesforce automation tools to streamline business processes for SN SmartLearn. Each tool was configured with clear use cases to reduce manual work, ensure data accuracy, and improve the student/admissions experience.

1. Validation Rules

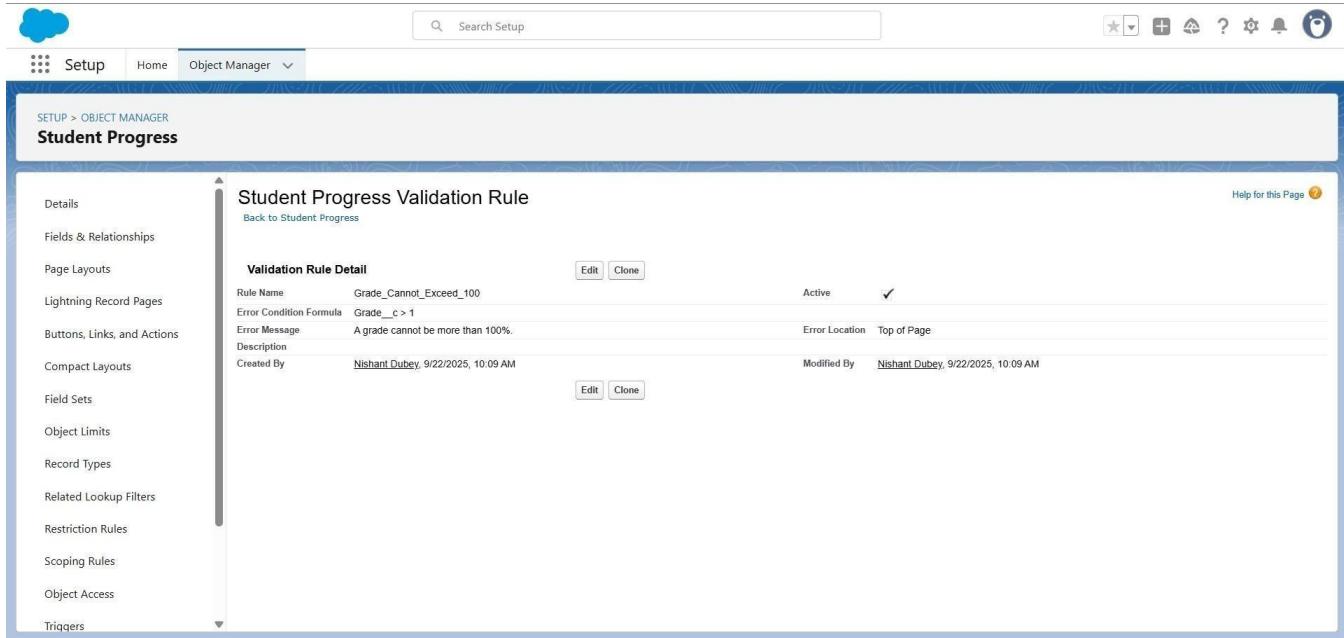
Validation Rules enforce data integrity by preventing users from saving records with invalid values.

Use Case: Prevent a Grade on a *Student Progress* record from being greater than 100%. **Steps Implemented:**

1. Setup → Object Manager → Student Progress.
2. Open **Validation Rules** → **New Rule**.
3. Rule Name: Grade_Cannot_Exceed_100.
4. Error Condition Formula:
 $Grade_c > 100$
5. Error Message: “A grade cannot be greater than 100%.”

7. Save and Activate.

This ensures grading remains within the correct range and prevents data entry errors.



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

Setup > OBJECT MANAGER

Student Progress

Validation Rule Detail

Field	Value	Action
Rule Name	Grade_Cannot_Exceed_100	Active
Error Condition Formula	Grade__c > 1	
Error Message	A grade cannot be more than 100%.	Error Location
Description		Top of Page
Created By	Nishant Dubey, 9/22/2025, 10:09 AM	Modified By
		Nishant Dubey, 9/22/2025, 10:09 AM

Left sidebar (Details)

- Details
- Fields & Relationships
- Page Layouts
- Lightning Record Pages
- Buttons, Links, and Actions
- Compact Layouts
- Field Sets
- Object Limits
- Record Types
- Related Lookup Filters
- Restriction Rules
- Scoping Rules
- Object Access
- Triggers

2. Workflow Rules (Legacy Tool)

Workflow Rules are an older automation tool, now replaced by Flow Builder, but documented here for completeness.

Use Case: Automatically create a follow-up Task for Admissions when a new *Student Application (Lead)* is created.

Steps Implemented:

1. Setup → Workflow Rules → New Rule.
2. Object: **Lead**.
3. Rule Name: *Create Task for New Application*.

4. Evaluation Criteria: *Created*.
5. Rule Criteria: None (runs for every new Lead).
6. Immediate Workflow Action → New Task:
 - Assigned To: Admissions Team User.
 - Subject: *Follow up on new application*.
 - Due Date: Lead Created Date + 7 days.
7. Save → Done → Activate Rule.

The screenshot shows the Salesforce Setup interface with the 'Workflow Rules' page open. The left sidebar shows navigation options like Process Automation, Workflow Actions, and Workflow Rules. The main area displays a 'Workflow Rule Detail' for a rule named 'Create Task for New Application'. The rule is active and triggered by the 'Lead: Created Date NOTEQUALTO null' criteria. It creates a 'Task' with the subject 'Follow up on new application'. A note at the bottom states: 'You cannot add new time triggers to an active rule. Deactivate This Rule'.

3. Process Builder (Legacy Tool)

Process Builder was used to automate record updates. It is now deprecated, but included here for legacy documentation.

Use Case: When *Enrollment* status changes to “Completed,” update the related *Contact (Student)* record’s status.

Steps Implemented:

1. Setup → Process Builder → New.
2. Process Name: *Update Student Status on Course Completion*.
3. Start Process: *When a record changes*.
4. Object: **Enrollment**. Trigger: Created or Edited.
5. Add Criteria:
 - Criteria Name: *Course Completed*.
 - Conditions:
 - Status__c Is Changed = True.
 - Status__c Equals = "Completed".
6. Immediate Action → Update Records:
 - Record Type: Student (related Contact).
 - New Value: Enrollment_Status__c = "Completed".
7. Save → Activate.

Setup Home Object Manager

Go with the flow! With Flow Builder, the future of low-code automation, you can do everything you do with Process Builder—and more! Salesforce plans to retire Process Builder and recommends building automation in Flow Builder.

Process Builder - Update Student Status on Course Completion

Expand All Collapse All

Action Name *

Record *

Criteria for Updating Records *
 Updated records meet all conditions
 No criteria—just update the records!

Set new field values for the records you update

Field *	Type *	Value *
Enrollment Status	Picklist	Completed

[Save](#) [Cancel](#)

Setup Home Object Manager

Go with the flow! With Flow Builder, the future of low-code automation, you can do everything you do with Process Builder—and more! Salesforce plans to retire Process Builder and recommends building automation in Flow Builder.

Process Builder - Update Student Status on Course Completion

Expand All Collapse All

Define Criteria for this Action Group

Criteria Name *

Criteria for Executing Actions *
 Conditions are met
 Formula evaluates to true
 No criteria—just execute the actions!

Set Conditions

Field *	Operator *	Type *	Value *
1 [Enrollment__c]...	Is changed	Boolean	True
2 [Enrollment__c]...	Equals	Picklist	Completed

Conditions *
 All of the conditions are met (AND)
 Any of the conditions are met (OR)
[Customize the logic](#)

[Save](#) [Cancel](#)

4. Approval Process

Approval Processes enable formal sign-off flows for records.

Use Case: Require management approval for enrollments with discounts above 20%.

Steps Implemented:

1. Setup → Approval Processes → New Approval Process.
2. Object: **Enrollment**.
3. Process Name: *Discount Approval*.
4. Entry Criteria: *Discount_Percentage_c > 20*.
5. Next Approver: User in *Management Role*.
6. Final Approval Action: Field Update → Enrollment Status = “Approved”.
7. Activate Process.

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

- Search Bar:** Search Setup
- Header:** Setup, Home, Object Manager
- Left Sidebar:** Process Automation, Approval Processes (selected), Global Search bar (containing "approval proc").
- Page Title:** Approval Processes
- Page Content:**
 - Process Definition Detail:**

Process Name	Discount Approval	Active	✓
Unique Name	Discount_Approval	Next Automated Approver Determined By	Manager of Record Submitter
Description			
Entry Criteria	Enrollment: Discount Percentage GREATER THAN 0.20		
Record Editability	Administrator ONLY	Allow Submitters to Recall Approval Requests	<input type="checkbox"/>
Approval Assignment Email Template			
Initial Submitters	Contact Owner, Role: Admissions Team		
Created By	Nishant Dubey, 9/22/2025, 11:22 AM	Modified By	Nishant Dubey, 9/22/2025, 11:25 AM
 - Initial Submission Actions:**

Action Type	Description
Record Lock	Lock the record from being edited
 - Approval Steps:**

Action	Step Number	Name	Description	Criteria	Assigned Approver	Reject Behavior
Show Actions Edit	1	Manager Approval	Manager reviews discount requests over 20%.		Manager	Final Rejection
 - Final Approval Actions:**

Action Type	Description
Edit Record Lock	Lock the record from being edited
 - Final Rejection Actions:**

Action Type	Description
Edit Record Lock	Lock the record from being edited

5. Flow Builder

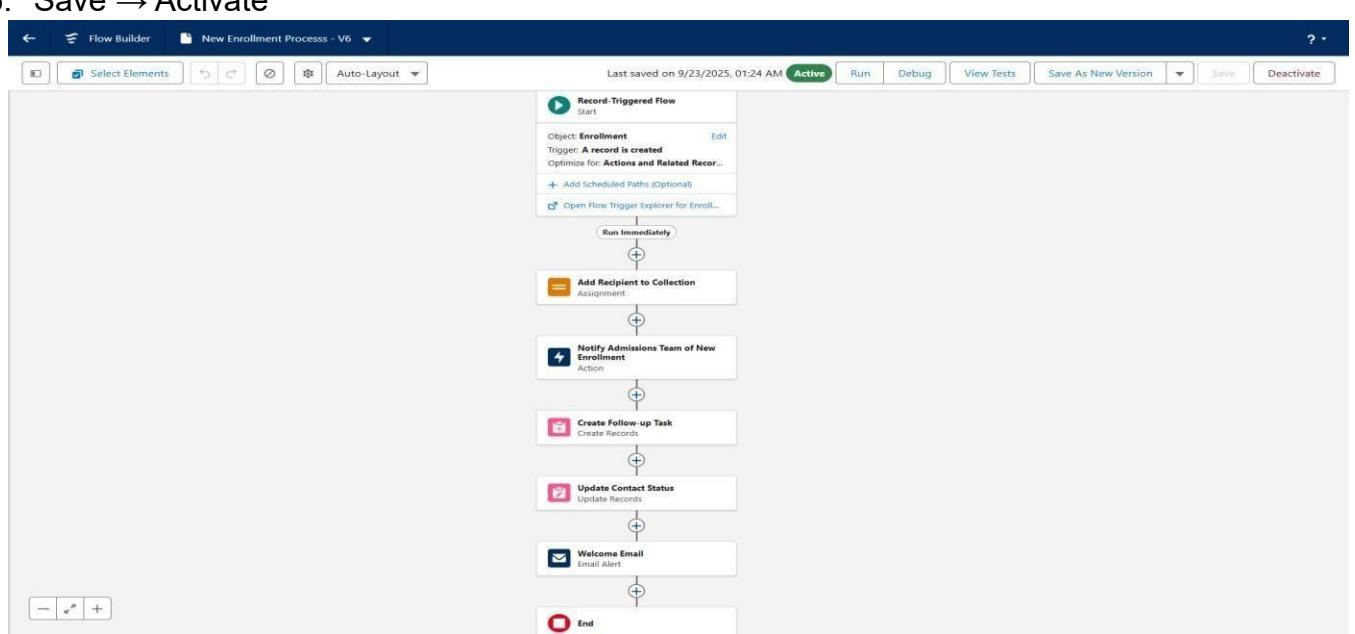
Flow Builder is the **primary automation tool** in Salesforce, replacing Workflow and Process Builder.

Use Case: Automate the student welcome process after enrollment.

Steps Implemented:

1. Setup → Flows → New Flow.
2. Flow Type: Record-Triggered Flow.
3. Object: **Enrollment**. Trigger: *Record Created*.
4. Optimize For: Actions & Related Records.
5. Added actions:
 - Send Welcome Email (Email Alert).
 - Create Admissions Follow-up Task.
 - Update Student Record fields.

6. Save → Activate



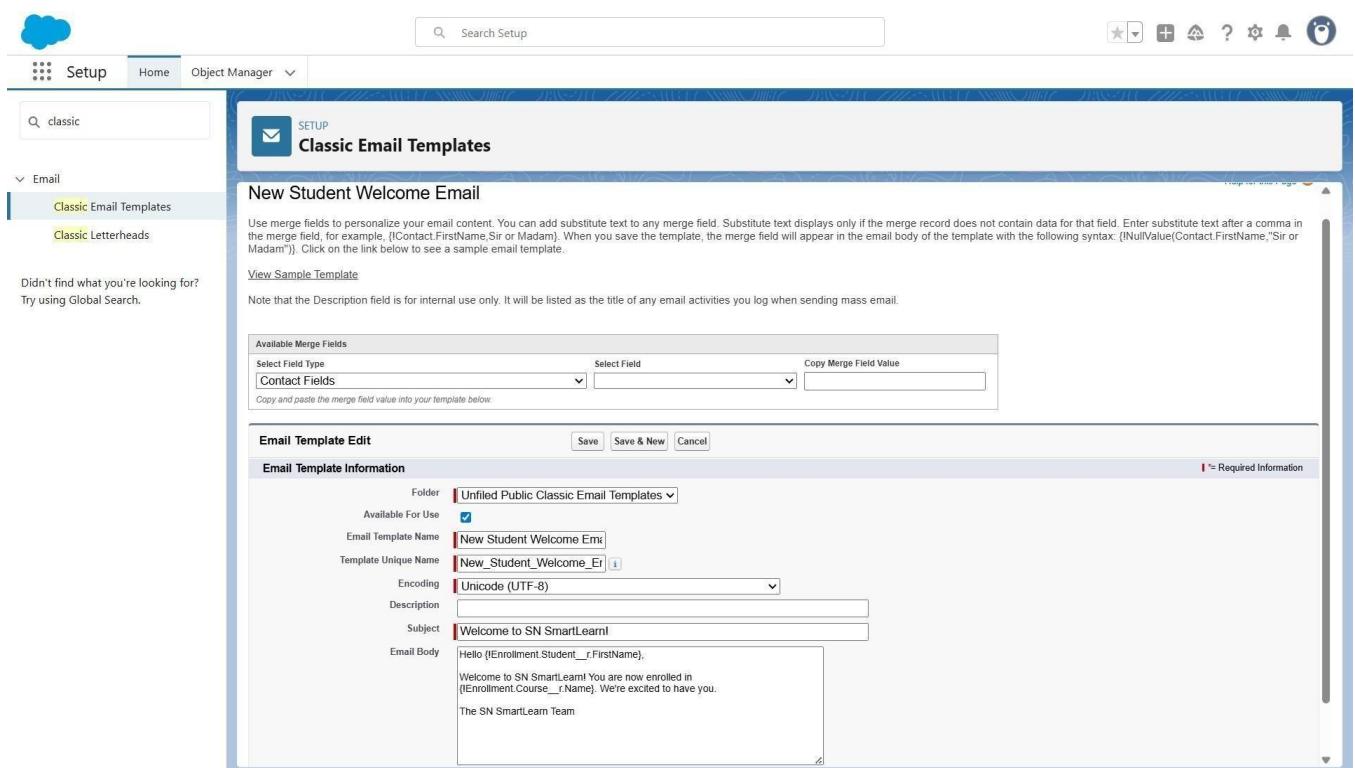
6. Email Alerts

Email Alerts send predefined email templates to specific recipients.

Use Case: Send a welcome email to newly enrolled students.

Steps Implemented:

1. Setup → Classic Email Templates → Create Template.
2. Setup → Email Alerts → New Alert.
 - Description: *Welcome Email to Student*.
 - Object: Enrollment. ○ Email Template: Select Welcome Template.
 - Recipient: Related Contact → Student.
3. Save.



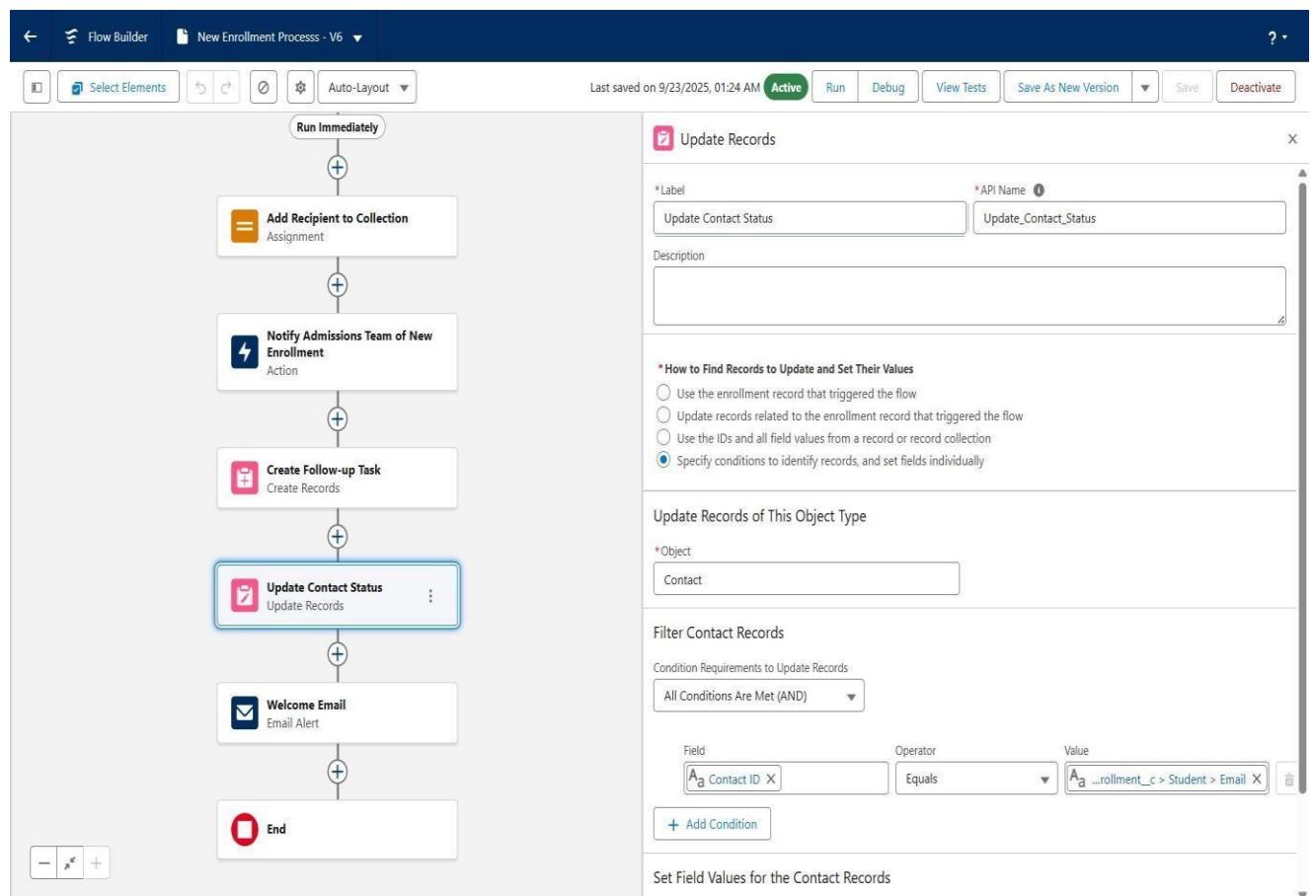
7. Field Updates

Field Updates automatically change field values when triggered by automation.

Use Case: Update Student's Contact record when Enrollment is created.

Steps Implemented (via Flow):

1. Flow Builder → Add *Update Records* element.
2. Object: **Contact**.
3. Condition: `Id = {!$Record.Student__c}`.
4. Field Value: `Enrollment_Status__c = "Enrolled"`.
5. Save → Connect → Activate.



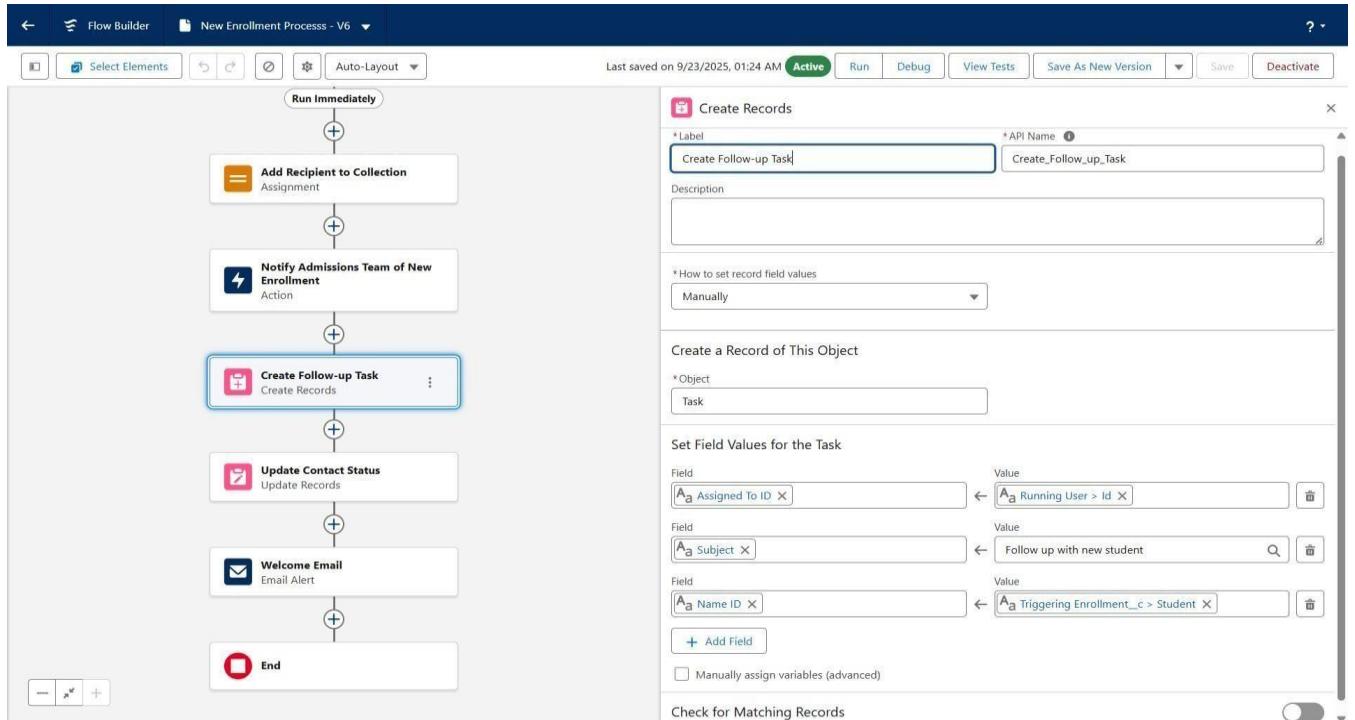
8. Tasks

Tasks create actionable to-dos for users inside Salesforce.

Use Case: Assign follow-up task for Admissions Officer when a new application is submitted.

Steps Implemented (via Flow):

1. Flow Builder → Add *Create Records* element.
2. Object: **Task**.
3. Field Values:
 - Subject: *Follow up with new student*. ○ Whold = Student (\$Record.Student__c).
 - OwnerId = Record Owner (\$Record.OwnerId).
4. Save → Activate.



9. Custom Notifications

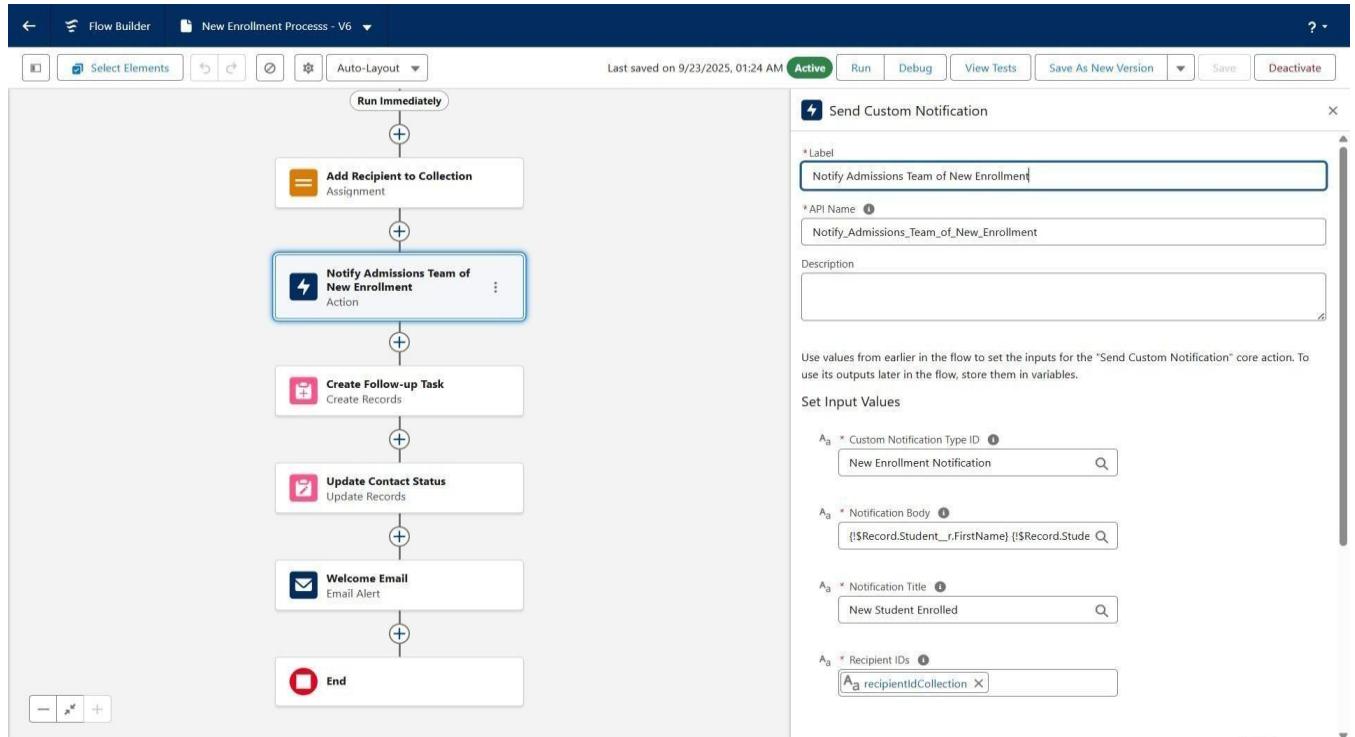
Custom Notifications send alerts to users in Salesforce (bell icon & mobile).

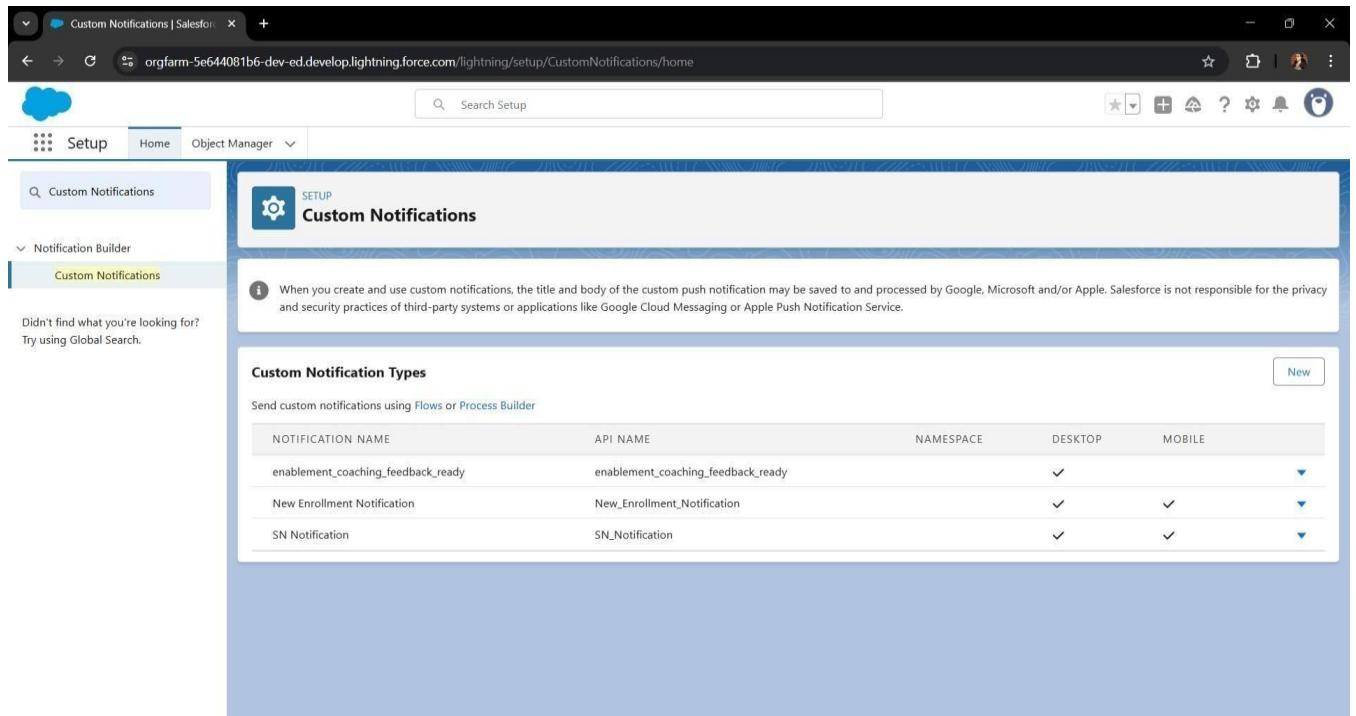
Use Case: Notify Admissions Team Lead when a new student enrolls.

Steps Implemented:

1. Setup → Custom Notifications → Create Notification Type.
2. Flow Builder → Add Action → *Send Custom Notification*.
3. Configure:
 - Notification Type: *Enrollment Notification*. ○ Title: *New Student Enrollment*.
 - Body: *A new student has enrolled. Please review.*
 - Recipient: Admissions Team Lead.

Save → Activate.





Phase 5 : Apex Programming (Developer Side)

1. Classes & Objects

Apex classes were created to encapsulate business logic and ensure reusability.

Steps Implemented:

1. Setup → Apex Classes → New.
2. Created utility classes such as EnrollmentService to calculate average grades and update enrollments.
3. Each class included methods (objects in Apex) that defined reusable operations.

The screenshot shows the Salesforce Setup interface. In the top navigation bar, there are icons for Home, Object Manager, and a search bar labeled "Search Setup". Below the navigation, a sidebar on the left contains a search field with "apex" typed in. The sidebar has several sections: Email (Apex Exception Email), Custom Code (Apex Classes, highlighted in blue), Apex Settings, Apex Test Execution, Apex Test History, Apex Triggers, Environments, Jobs (Apex Flex Queue, Apex Jobs), and a "Did you find what you're looking for? Try using Global Search." section.

The main content area is titled "SETUP Apex Classes" and shows the "EnrollmentService" Apex Class Detail. The class is defined under the "Apex Class" category. The "Name" is "EnrollmentService". The "Status" is "Active" and "Code Coverage" is "0% (0/13)". The "Last Modified By" is "Nishant Dubey" at "9/24/2025, 8:58 AM".

The code editor displays the following Apex code:

```

1 public with sharing class EnrollmentService {
2     // Calculate average grades for multiple enrollments
3     public static Map<Id, Decimal> calculateAverageGrades(Set<Id> enrollmentIds) {
4         Map<Id, Decimal> result = new Map<Id, Decimal>();
5         if (enrollmentIds.isEmpty()) return result;
6
7         List<AggregateResult> agg = [
8             SELECT Enrollment__c en, AVG(Grade__c) avgG
9             FROM Student_Progress__c
10            WHERE Enrollment__c IN :enrollmentIds
11            GROUP BY Enrollment__c
12        ];
13        for (AggregateResult ar : agg) {
14            result.put(ar.get('en'), (Decimal)ar.get('avgG'));
15        }
16    }
17
18    // Update enrollments with calculated averages
19    public static void updateEnrollmentsWithAverages(Map<Id, Decimal> averages) {
20        List<Enrollment__c> ups = new List<Enrollment__c>();
21        for (Id eld : averages.keySet()) {
22            ups.add(new Enrollment__c{Id = eld, Average_Grade__c = averages.get(eld)});
23        }
24        if (ups.isEmpty()) update ups;
25    }
26
27 }

```

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

Triggers were implemented to automate backend logic when records are created, updated, or deleted.

Use Case: When a Student Progress record is added/updated, recalculate the student's average grade.

Steps Implemented:

1. Setup → Object Manager → Student Progress → Triggers → New.
2. Defined before insert and after update logic.
3. Trigger calls the service class instead of writing logic directly.

The screenshot shows the Salesforce Object Manager interface. The top navigation bar includes 'Setup', 'Home', 'Object Manager', and various global icons. The main title is 'Student Progress' under 'SETUP > OBJECT MANAGER'. On the left, a sidebar lists various object settings like Details, Fields & Relationships, Page Layouts, etc., with 'Triggers' selected. The main content area is titled 'Apex Trigger StudentProgressTrigger'. It shows basic details: Name (StudentProgressTrigger), Code Coverage (0% (0/9)), Created By (Nishant Dubey, 9/24/2025, 9:03 AM), and Namespace Prefix. A large code editor window displays the Apex trigger code:

```

1 trigger StudentProgressTrigger on Student_Progress__c (
2     after insert, after update, after delete
3 ) {
4     Set<Id> enrollmentIds = new Set<Id>();
5 
6     if (Trigger.isInsert || Trigger.isUpdate) {
7         for (Student_Progress__c sp : Trigger.new) {
8             if (sp.Enrollment__c != null) enrollmentIds.add(sp.Enrollment__c);
9         }
10    }
11    if (Trigger.isDelete) {
12        for (Student_Progress__c sp : Trigger.old) {
13            if (sp.Enrollment__c != null) enrollmentIds.add(sp.Enrollment__c);
14        }
15    }
16 
17    if (enrollmentIds.isEmpty()) {
18        System.enqueueJob(new EnrollmentAverageQueueable(enrollmentIds));
19    }
20}

```

Below the code are standard edit, delete, download, and show dependencies buttons.

3. Trigger Design Pattern

To keep code clean and bulk-safe, a **Handler Class** was introduced.

Steps Implemented:

1. Created StudentProgressTriggerHandler class.
2. Trigger simply delegates logic to handler.
3. Improves maintainability.

4. SOQL & SOSL

Both query languages were used:

- **SOQL** (SELECT) for structured queries.

```
sql
```

```
SELECT Id, Name, Average_Grade__c FROM Enrollment__c LIMIT 5
```

```
List<List<SObject>> results = [FIND 'Math' IN ALL FIELDS RETURNING Course__c(Name)];
```

- **SOSL** (FIND) for text-based searches across objects

5. Collections: List, Set, Map

Collections were used to handle bulk data efficiently.

Steps Implemented:

- **List**: Store multiple records.
- **Set**: Avoid duplicates.
- **Map**: Key-value pairs for quick lookup.

```

List<String> names = new List<String>{'A','B'};
Set<Id> ids = new Set<Id>();
Map<Id, String> mapEx = new Map<Id, String>();

```

The screenshot shows the Salesforce Developer Console interface. At the top, there's a header bar with tabs for File, Edit, Debug, Test, Workspace, Help, and a search bar. Below the header is a title bar indicating the URL is orgfarm-5e644081b6-dev-ed.develop.my.salesforce.com and the specific page is /ui/common/apex/debug/ApexCSIPage?action=selectExtent&extent=apexclass. A timestamp at the top right shows 9/25/2025, 1:56:15 PM.

The main area is titled "Log executeAnonymous @9/25/2025, 1:56:15 PM". It contains an "Execution Log" table with columns for Timestamp, Event, and Details. The log entries show the execution of the provided Apex code, including variable declarations, list creation, and heap allocations for strings and sets.

Below the log is a toolbar with buttons for This Frame, Executable, Debug Only, Filter, and a link to Click here to filter the log. There are also tabs for Logs, Tests, Checkpoints, Query Editor, View State, Progress, and Problems. The Logs tab is currently selected.

At the bottom of the interface is a taskbar with various icons for system tools like File Explorer, Task Manager, and browser windows. On the far right of the taskbar, there are system status indicators for battery level, signal strength, and the date/time (25-09-2025).

6. Batch Apex

Used for large-scale recalculations and reporting.

Steps Implemented:

1. Setup → Apex Classes → New → Batch Class.
2. Implemented Database.Batchable.

3. Executed using Database.executeBatch()

The screenshot shows the Salesforce Setup interface under the Apex Classes section. The class `RecalculateEnrollmentBatch` is selected. The code implements the `Database.Batchable<SOObject>` interface. It starts by querying enrollment records, then iterates through them to calculate average grades, and finally updates the enrollment records with these averages.

```

1 public class RecalculateEnrollmentBatch implements Database.Batchable<SOObject> {
2     public Database.QueryLocator start(Database.BatchableContext bc) {
3         return Database.getQueryLocator("SELECT Id FROM Enrollment__c");
4     }
5     public void execute(Database.BatchableContext bc, List<Enrollment__c> scope) {
6         Set<Id> ids = new Set<Id>();
7         for(Enrollment__c e : scope) ids.add(e.Id);
8         Map<Id, Decimal> averages = EnrollmentService.calculateAverageGrades(ids);
9         EnrollmentService.updateEnrollmentsWithAverages(averages);
10    }
11    public void finish(Database.BatchableContext bc) {
12        System.debug("Batch job completed");
13    }
14 }

```

The screenshot shows the Salesforce Developer Console log for the anonymous user. The log details the execution of the `RecalculateEnrollmentBatch` class, including memory allocations and method calls. The log ends with a success message indicating the batch job was completed.

```

Log executeAnonymous @9/25/2025, 1:59:07 PM
[EXTERNAL]005g000005D77F|nishant.dubey.cs22174@agentforce.com|(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)|GMT-07:00
13:59:07.001 USER_INFO [EXTERNAL]005g000005D77F|nishant.dubey.cs22174@agentforce.com|(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)|GMT-07:00
13:59:07.001 EXECUTION_STARTED
13:59:07.001 CODE_UNIT_STARTED [EXTERNAL]execute_anonymous_apex
13:59:07.002 HEAP_ALLOCATE [95]Bytes:3
13:59:07.002 HEAP_ALLOCATE [100]Bytes:152
13:59:07.002 HEAP_ALLOCATE [417]Bytes:408
13:59:07.002 HEAP_ALLOCATE [430]Bytes:408
13:59:07.002 HEAP_ALLOCATE [317]Bytes:6
13:59:07.002 HEAP_ALLOCATE [EXTERNAL]Bytes:1
13:59:07.002 STATEMENT_EXECUTE [1]
13:59:07.002 STATEMENT_EXECUTE [1]
13:59:07.004 HEAP_ALLOCATE [1]Bytes:57
13:59:07.004 HEAP_ALLOCATE [1]Bytes:6
13:59:07.004 HEAP_ALLOCATE [1]Bytes:2
13:59:07.004 METHOD_ENTRY [1]01pgl000005q83R|RecalculateEnrollmentBatch.RecalculateEnrollmentBatch()
13:59:07.004 STATEMENT_EXECUTE [1]
13:59:07.004 STATEMENT_EXECUTE [1]
13:59:07.004 STATEMENT_EXECUTE [1]
13:59:07.004 METHOD_EXIT [1]RecalculateEnrollmentBatch
13:59:07.004 HEAP_ALLOCATE [1]Bytes:4
13:59:07.004 HEAP_ALLOCATE [68]Bytes:5
13:59:07.004 HEAP_ALLOCATE [74]Bytes:5

```

Logs

User	Application	Operation	Time	Status	Read	Size
Nishant Dubey	Unknown	Batch Apex	9/25/2025, 1:59:10 PM	Success	Unread	3.49 KB
Nishant Dubey	Unknown	Batch Apex	9/25/2025, 1:59:08 PM	Success	Unread	3.58 KB
Nishant Dubey	Unknown	/services/data/v64.0/tooling/executeA...	9/25/2025, 1:59:07 PM	Success		2.8 KB
Nishant Dubey	Unknown	Batch Apex	9/25/2025, 1:58:52 PM	Success		3.5 KB
Nishant Dubey	Unknown	Batch Apex	9/25/2025, 1:58:51 PM	Success	Unread	3.58 KB
Nishant Dubey	Unknown	/services/data/v64.0/tooling/executeA...	9/25/2025, 1:58:49 PM	Success	Unread	2.83 KB

7. Queueable Apex

Used for asynchronous jobs with more flexibility than Future methods.

Steps Implemented:

- Created EnrollmentAverageQueueable class implementing Queueable.
- Enqueued from trigger using System.enqueueJob().

The screenshot shows the Salesforce Setup Apex Classes page. The search bar at the top contains "apex". The main area displays the "Apex Class Detail" for "EnrollmentAverageQueueable". The class body contains the following Apex code:

```
1 public class EnrollmentAverageQueueable implements Queueable {
2     private Set<Id> enrollmentIds;
3     ...
4     // Constructor to accept Enrollment Ids
5     public EnrollmentAverageQueueable(Set<Id> ids) {
6         this.enrollmentIds = ids;
7     }
8     ...
9     public void execute(QueueableContext context) {
10        // Call service class to calculate averages
11        Map<Id, Decimal> averages = EnrollmentService.calculateAverageGrades(enrollmentIds);
12        EnrollmentService.updateEnrollmentsWithAverages(averages);
13    }
14 }
```

The page includes standard Salesforce navigation elements like "Edit", "Delete", "Download", "Security", and "Show Dependencies".

8. Scheduled Apex

Automated background jobs on a schedule.

Steps Implemented:

1. Created a scheduler class implementing Schedulable.
2. Setup → Apex Classes → Schedule Apex → Defined cron expression.

The screenshot shows the Salesforce Setup interface with the 'Apex Classes' page open. The left sidebar shows navigation options like 'Email', 'Custom Code', 'Environments', and 'Jobs'. The main area shows the details for the 'EnrollmentRecalcScheduler' class, which is active and has 0% code coverage. The code body is as follows:

```

1  public class EnrollmentRecalcScheduler implements Schedulable {
2      public void execute(SchedulableContext sc) {
3          Database.executeBatch(new RecalculateEnrollmentBatch(), 200);
4      }
5  }

```

9. Future Methods

Used for lightweight async calls.

The screenshot shows the Salesforce Setup interface with the 'Apex Classes' page selected. The left sidebar has a search bar and categories like Email, Custom Code, Environments, and Jobs. The main area shows the 'ExternalIntegration' class details. The class name is 'ExternalIntegration', namespace prefix is 'ExternalIntegration', created by 'Nishant Dubey' on 9/25/2025, and status is 'Active'. The code body contains the following Apex code:

```
1 public class ExternalIntegration {
2     @future
3     public static void notifySystem(Set<id> enrollmentIds) {
4         System.debug('Notify external system: ' + enrollmentIds);
5     }
6 }
```

10. Exception Handling

Added try-catch-finally blocks for error handling.

The screenshot shows the Salesforce Setup interface with the search bar set to 'apex'. The left sidebar lists various setup categories like Email, Custom Code, Environments, and Jobs. The 'Apex Classes' section is selected and expanded, showing sub-options: Apex Settings, Apex Test Execution, Apex Test History, and Apex Triggers. The main content area is titled 'Apex Classes' and shows a specific class named 'ExceptionDemo'. The 'Apex Class Detail' table includes fields for Name (ExceptionDemo), Namespace Prefix, Created By (Nishant Dubey), Status (Active), and Code Coverage (0%). Below the table, tabs for 'Class Body', 'Class Summary', 'Version Settings', and 'Trace Flags' are visible. The 'Class Body' tab displays the following Apex code:

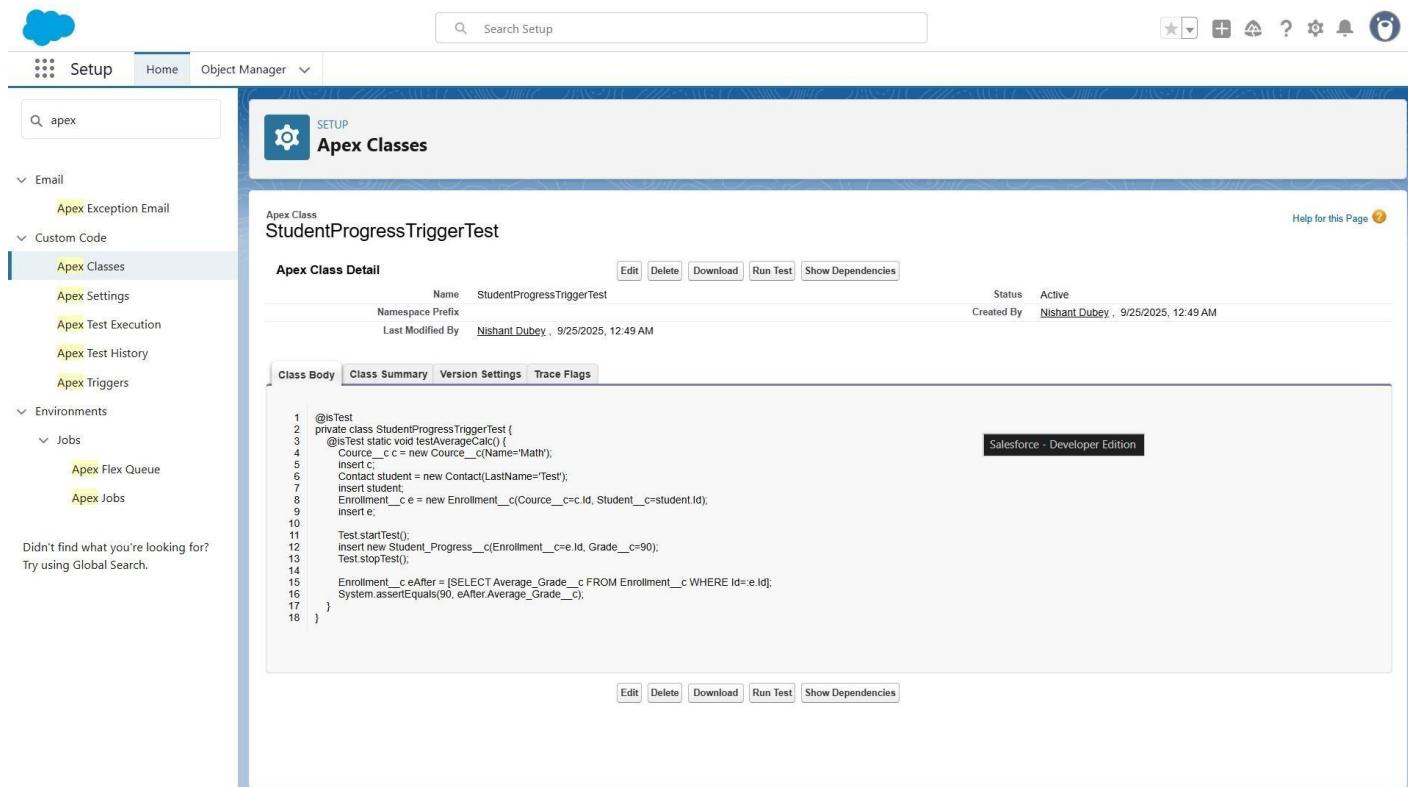
```
1 public class ExceptionDemo {  
2     public static void updateEnrollments(List<Enrollment__c> enrollments) {  
3         try {  
4             update enrollments;  
5         } catch (DmlException e){  
6             System.debug('Error: ' + e.getMessage());  
7         }  
8     }  
9 }
```

11. Test Classes

Created unit tests to validate Apex logic and increase code coverage.

Steps Implemented:

1. Setup → Apex Classes → New.
2. Used @isTest annotation.
3. Inserted test data → called methods → asserted results.



12. Asynchronous Processing

Covered all async types: Batch, Queueable, Future, Scheduled Apex.

Steps Implemented:

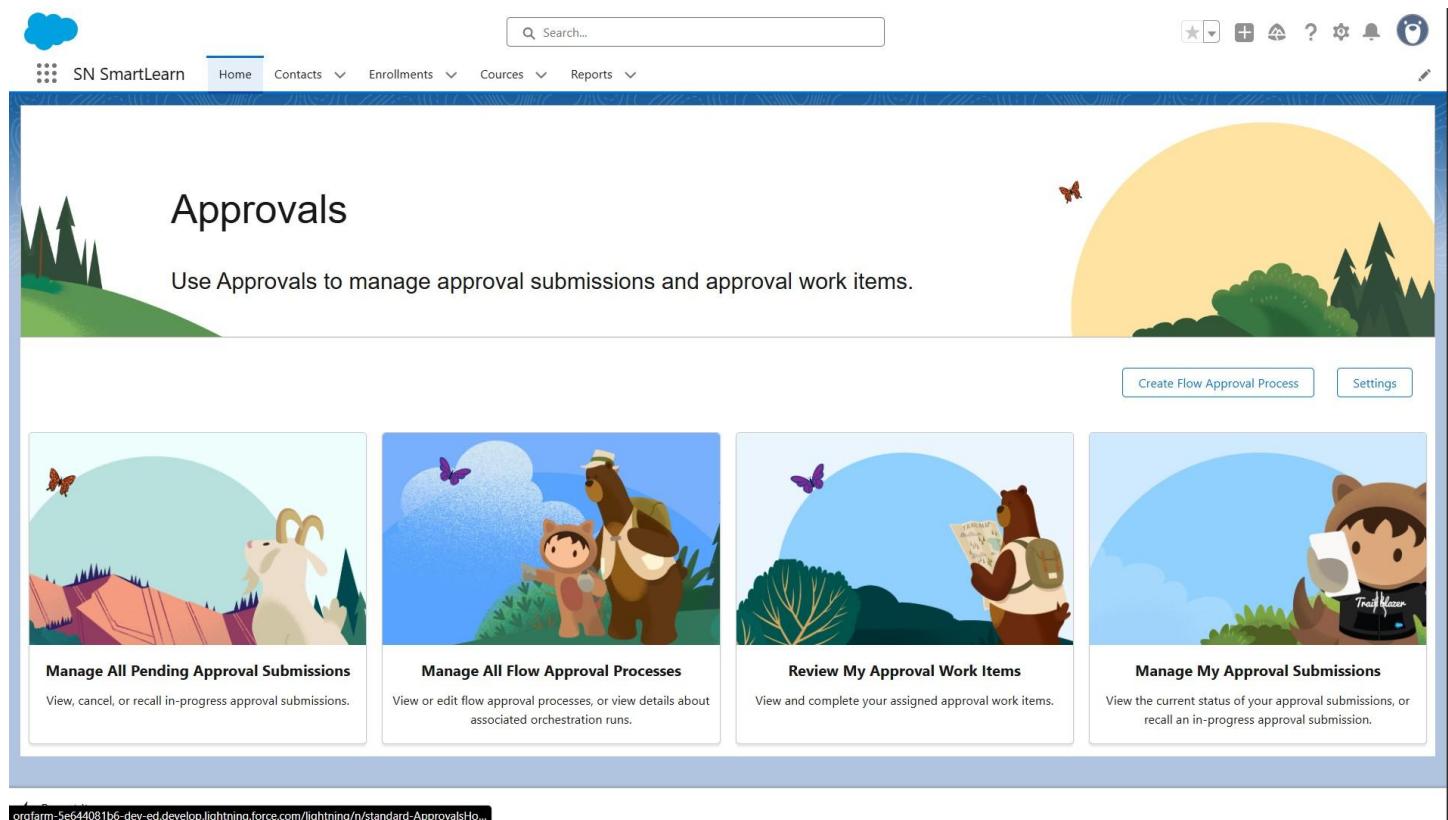
- Used Queueable for enrollment recalculations.
- Used Batch Apex for large dataset processing.
- Used Future for external calls.
- Used Scheduled Apex for nightly runs.

Phase 6 : User Interface Development

Lightning App & Tabs

Action: You created the **SN SmartLearn** Lightning App.

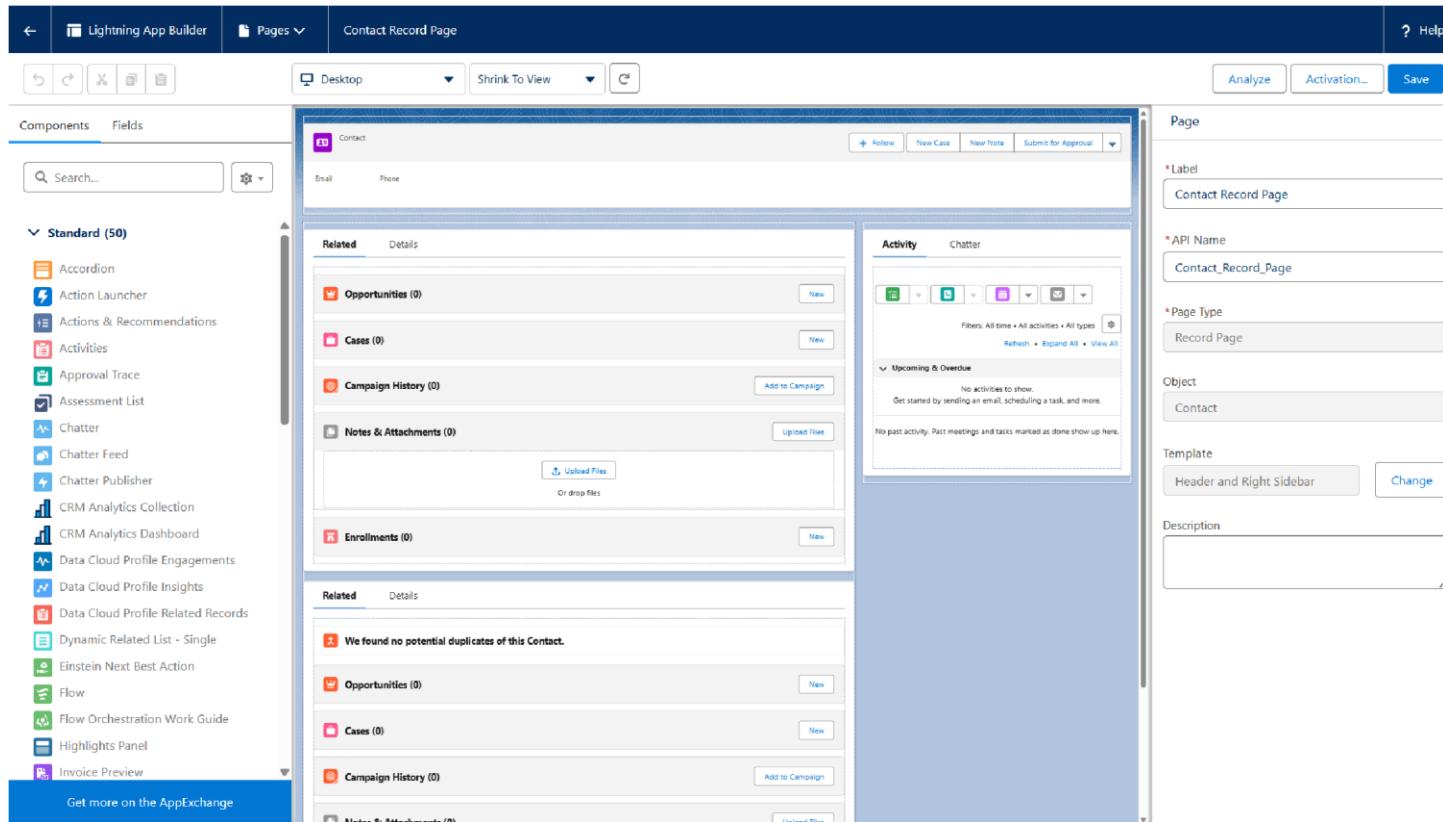
Use: This provides a dedicated, branded workspace for your Admissions Team and instructor separating their work from other functions in Salesforce. The **Tabs** (Home, Students, Courses, etc.) provide easy, one-click navigation to the most important objects.



Record Pages

Action: You customized the **Contact** (Student) record page using the Lightning App Builder, adding a Tabs component to organize details and related lists.

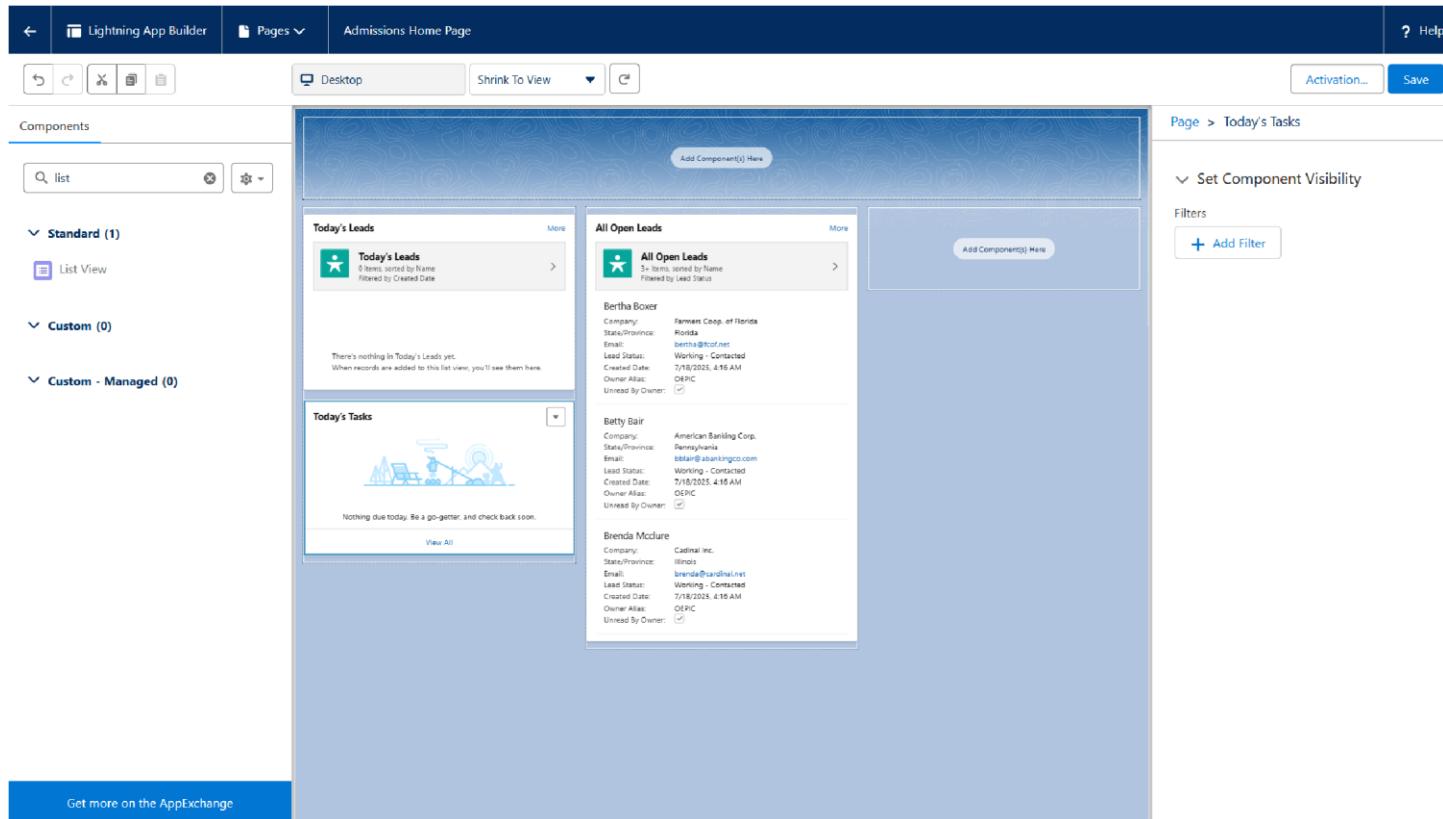
Use: This creates the "360-degree view" of the student, allowing users to see a student's details and all their related enrollments and progress in one clean, organized screen.



Home Page Layouts

Action: You designed a custom **Home Page** named "Admissions Home Page" and assigned it to the Admissions profile.

Use: This provides a dashboard-style landing page for the Admissions Team, showing them the most relevant information for their day, such as new applications (Leads) and their assigned tasks.

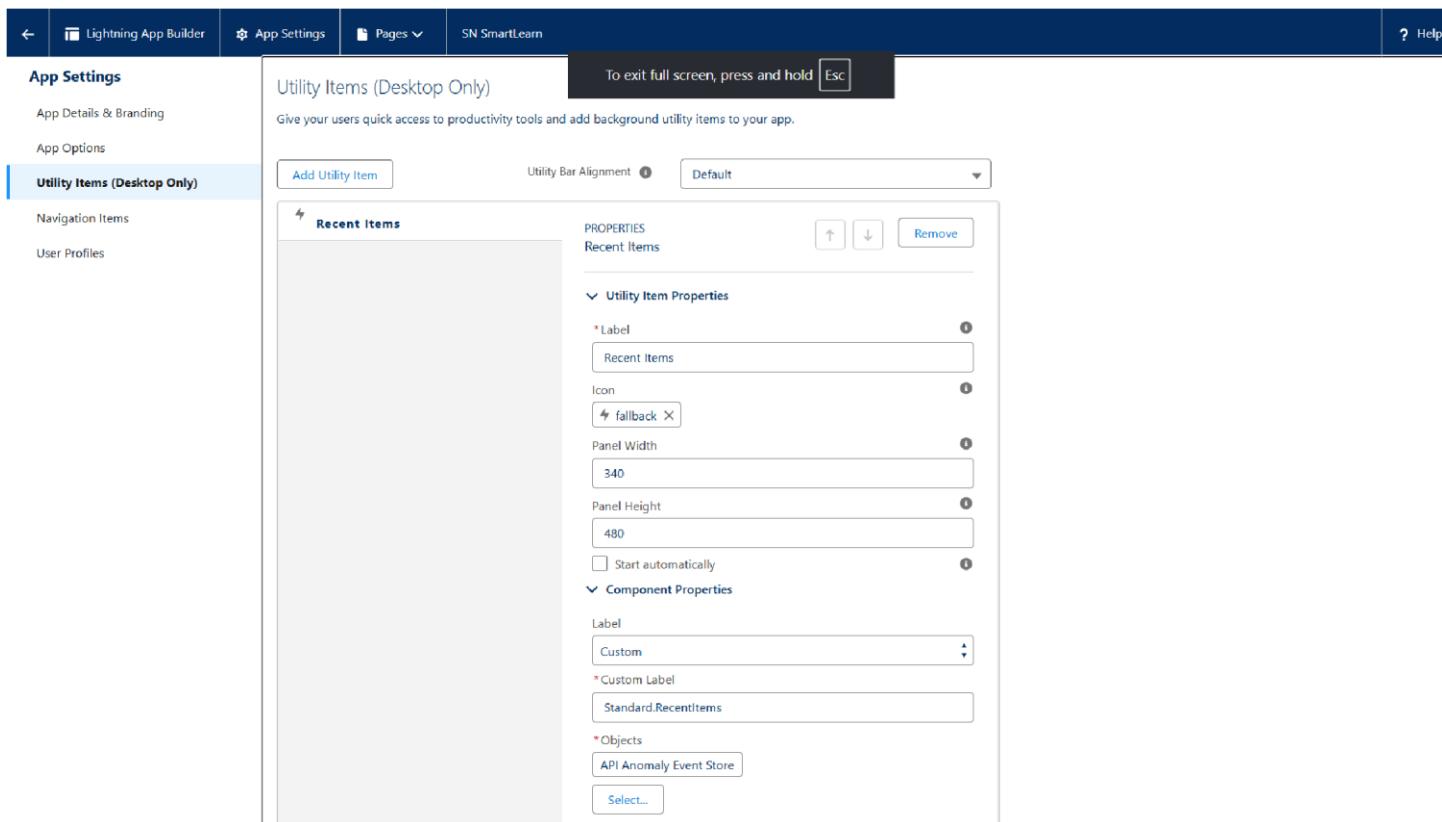


Utility Bar

Action: You added the Recent Items component to the app's **Utility Bar**.

Use: This gives users a persistent, quick-access menu at the bottom of the screen to easily find and navigate to the records they were recently working on, improving efficiency.

Phase 7 : Integration & External Access



Named Credentials

- **This is a future step.**
- **What it is:** A secure way to store the login details (like a username and API key) needed to connect to another system.
- **What you would do:** You would create a **Named Credential** to securely store the login information for an external student verification service. This is a best practice that keeps passwords out of your code.

External Services

- **This is a future step.**
- **What it is:** A point-and-click tool that lets you connect to an external system's API without writing code.

- **What you would do:** If a student verification service had a compatible API, you could use **External Services** to create a "Verify Student" action that could be used in a Flow by your admissions team.

Web Services (REST/SOAP) & Callouts

- **This is a future step.**
- **What they are:** **Web Services** are the languages different systems use to talk to each other over the internet. A **Callout** is the action of Salesforce sending a message to a web service.
- **What you would do:** A developer would write an Apex **Callout** using the modern **REST** format to send a student's ID to an external service and get a response back.

Platform Events

- **This is a future step.**
- **What it is:** A way for Salesforce to broadcast a message (an "event") that other systems can listen for, like a radio signal.
- **What you would do:** When an Enrollment status is updated to "Completed," your system could publish a `Student_Completed_Course__e` **Platform Event**. An external certificate-printing service could be "listening" for this signal and automatically print a certificate. This helps to streamline communications and automate the student journey.

Change Data Capture

- **This is a future step.**
- **What it is:** A feature that sends a notification to other systems whenever data changes in Salesforce.
- **What you would do:** You could enable **Change Data Capture** on the Contact object. If an admissions officer updates a student's address, a notification would be sent to an external student portal to keep the data in sync.

Salesforce Connect

- **This is a future step.**
- **What it is:** A tool to view data from an external database in real-time by creating "External Objects."
- **What you would do:** If your university's library had a separate database, you could use **Salesforce Connect** to create an External Object to display a list of a student's checked-out books on their Contact record, contributing to a unified, 360-degree view.

API Limits

- **This is a future step.**
- **What it is:** A limit on how many integration calls your Salesforce org can make in a 24-hour period to ensure system stability.
- **What you would do:** As an administrator, you would periodically monitor your API usage in **Setup > Company Information** to ensure your integrations are running efficiently.

OAuth & Authentication

- **This is a future step.**
- **What it is:** A secure way for systems to grant access to each other without sharing passwords (e.g., "Log in with Google").
- **What you would do:** You could set up an **OAuth** flow to allow students to log into a separate Student Portal using their main university login, providing a seamless experience.

Remote Site Settings

- **This is a future step.**
- **What it is:** A basic security setting where you must register the web address (URL) of any external system you want your code to call out to.
- **What you would do:** Before a developer could write an Apex Callout, an administrator would first go to **Setup > Remote Site Settings** to add the external service's URL (e.g., <https://api.studentverification.com>) to a trusted list.

Phase 8. Data Management & Deployment

1. Data Import Wizard

- **Use Case:** To perform the initial bulk import of Course records from a spreadsheet, which is a key part of centralizing all course and progress data into a single source of truth.
- **Steps Implemented:**
 1. A spreadsheet was prepared with Course Name and Course Code columns and saved as a CSV file.

2. The **Data Import Wizard** was launched from Setup.
3. The Course custom object was selected for import.
4. The CSV file was uploaded, and its columns were mapped to the corresponding Salesforce fields.
5. The import process was initiated to create the initial course records in the system.

Edit Field Mapping: Courses
Your file has been auto-mapped to existing Salesforce fields, but you can edit the mappings if you wish. Unmapped fields will not be imported.

Edit	Mapped Salesforce Object	CSV Header	Example	Example	Example
Change	Coarse Name	Course Name	Introduction to Si	Apex Programm	Business Administration
Change	Course Code	Course Code	SF-101	APX-201	BA-301

Cancel Previous Next

Review & Start Import
Review your import information and click Start Import.

Your selections:	Your import will include:	Your import will not include:
Courses ✓ Add new records ✓ sample.csv ✓	Mapped fields 2	Unmapped fields 0

Cancel Previous Start Import

Bulk Data Load Jobs

750gL00000E9g9h

View the details of a bulk data load job.

< Back to List: Bulk Data Load Jobs

Bulk Data Load Job Detail		Reload			
Job ID	750gL00000E9g9h	Job Type	Bulk V1	Status	Closed
Submitted By	Nishant Dubey	Operation	Insert	Total Processing Time (ms)	174
Start Time	9/26/2025, 3:26 AM PST	Queued Batches	0	API Active Processing Time (ms)	73
End Time	9/26/2025, 3:26 AM PST	In Progress Batches	0	Apex Processing Time (ms)	0
Time to Complete ([hh]:mm:ss)	00:01	Completed Batches	1		
Object	Course	Failed Batches	0		
External ID Field		Progress	100 %		
Content Type	CSV	Records Processed	3		
Concurrency Mode	Parallel	Records Failed	3		
API Version	64.0	Retries	0		

Reload

Batches														
View Request	View Result	Batch ID	Start Time	End Time	Total Processing Time (ms)	API Active Processing Time (ms)	Apex Processing Time (ms)	Records Processed	Records Failed	Retry Count	State Message	Status		
View_Request	View_Result	751gL00000B1Byv	9/26/2025, 3:26 AM	9/26/2025, 3:26 AM	174	73	0	3	3	0	Completed			

2. Data Loader

- **Use Case:** Data Loader was reviewed as an advanced tool for future, large-scale data migrations, such as importing millions of historical student records from a legacy system. For this project's initial setup, it was not required.

3. Duplicate Rules

- **Use Case:** To maintain data integrity and reduce manual errors by preventing the creation of duplicate student records.
- **Steps Implemented:**
 1. A **Matching Rule** was created for the Contact object to identify potential duplicates based on an exact match of the Email field and a fuzzy match of the First Name.

- A **Duplicate Rule** was created that uses this matching rule and is configured to **Block** a user from saving a record that is identified as a duplicate.

Bulk Data Load Jobs

750gL00000E9g9h

View the details of a bulk data load job.

< Back to List: Bulk Data Load Jobs

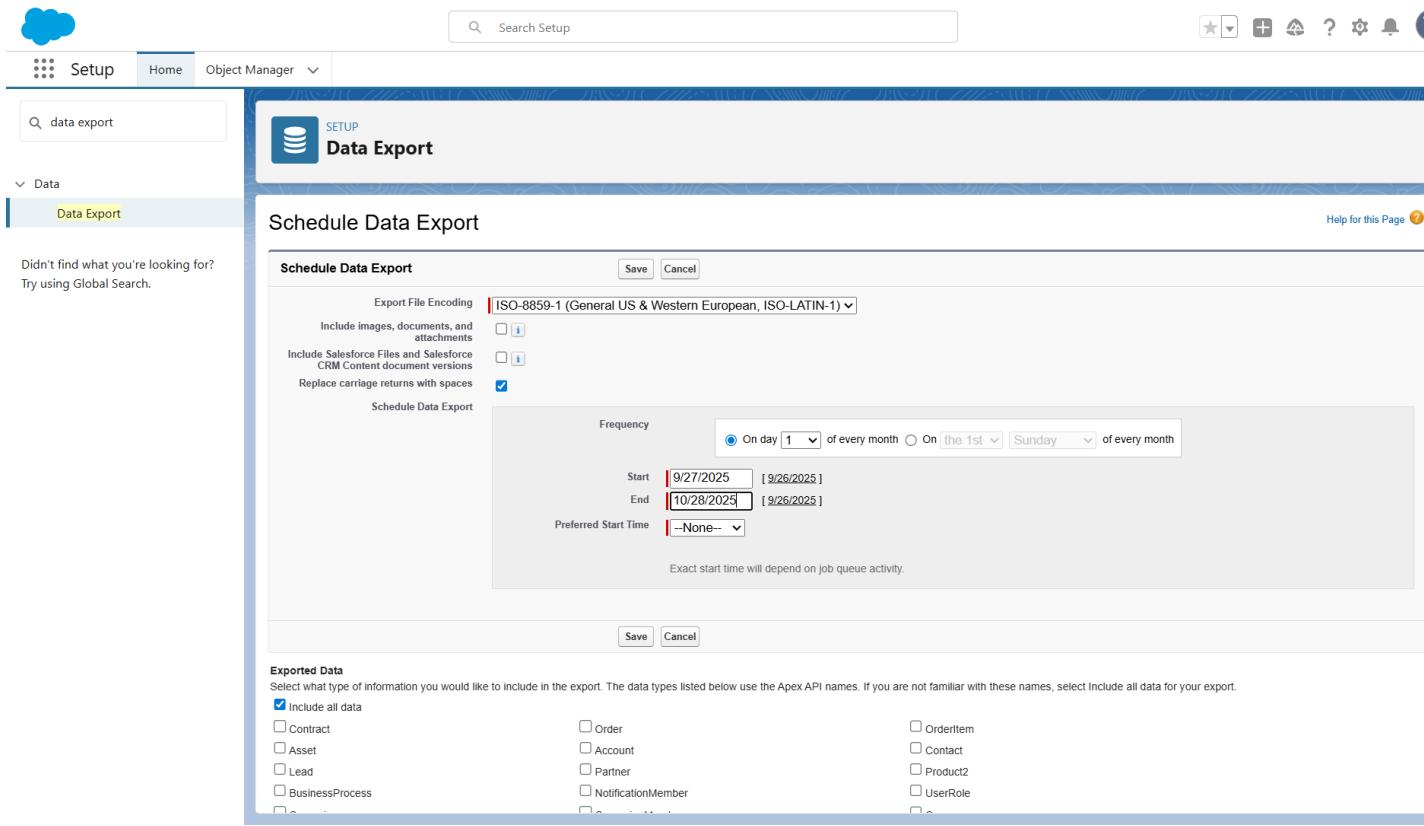
Bulk Data Load Job Detail		Reload			
Job ID	750gL00000E9g9h	Job Type	Bulk V1	Status	Closed
Submitted By	Nishant Dubey	Operation	Insert	Total Processing Time (ms)	174
Start Time	9/26/2025, 3:26 AM PST	Queued Batches	0	API Active Processing Time (ms)	73
End Time	9/26/2025, 3:26 AM PST	In Progress Batches	0	Apex Processing Time (ms)	0
Time to Complete ([hh]:mm:ss)	00:01	Completed Batches	1		
Object	Course	Failed Batches	0		
External ID Field		Progress	100 %		
Content Type	CSV	Records Processed	3		
Concurrency Mode	Parallel	Records Failed	3		
API Version	64.0	Retries	0		

Reload

Batches													
View Request	View Result	Batch ID	Start Time	End Time	Total Processing Time (ms)	API Active Processing Time (ms)	Apex Processing Time (ms)	Records Processed	Records Failed	Retry Count	State Message	Status	
View Request	View Result	751gL00000BiByv	9/26/2025, 3:26 AM	9/26/2025, 3:26 AM	174	73	0	3	3	0	Completed		

4. Data Export & Backup

- Use Case:** To ensure a secure backup of all critical student and course data for disaster recovery, supporting the IT/Admin's responsibility for data integrity.
- Steps Implemented:** The process of scheduling a **Weekly Export** was reviewed. This involves navigating to the **Data Export** service in Setup, selecting an interval, choosing to include all data, and saving the schedule.



5. Deployment Basics (Conceptual)

The following tools were reviewed conceptually to understand how a project is moved from a development environment to a live production org.

- **Change Sets:** Explained as the standard, point-and-click tool for deploying metadata (like custom objects, fields, and Flows) between a Sandbox and a Production org.
- **Unmanaged vs Managed Packages:** Reviewed as methods for bundling an application's components for distribution.
- **ANT Migration Tool & VS Code & SFDX:** Discussed as advanced, developer-focused tools for deployment that integrate with version control systems like **GitHub**.

Phase 9. Reporting, Dashboards & Security Review

1. Report Types

- **Use Case:** To create the necessary template for building reports that combine data from Students (Contacts) and their Enrollments, enabling effective progress tracking.
- **Steps Implemented:**
 1. A new Custom Report Type named Students with Enrollments was created.
 2. The Contact object was set as the primary object, and the Enrollment object was added as a related object.

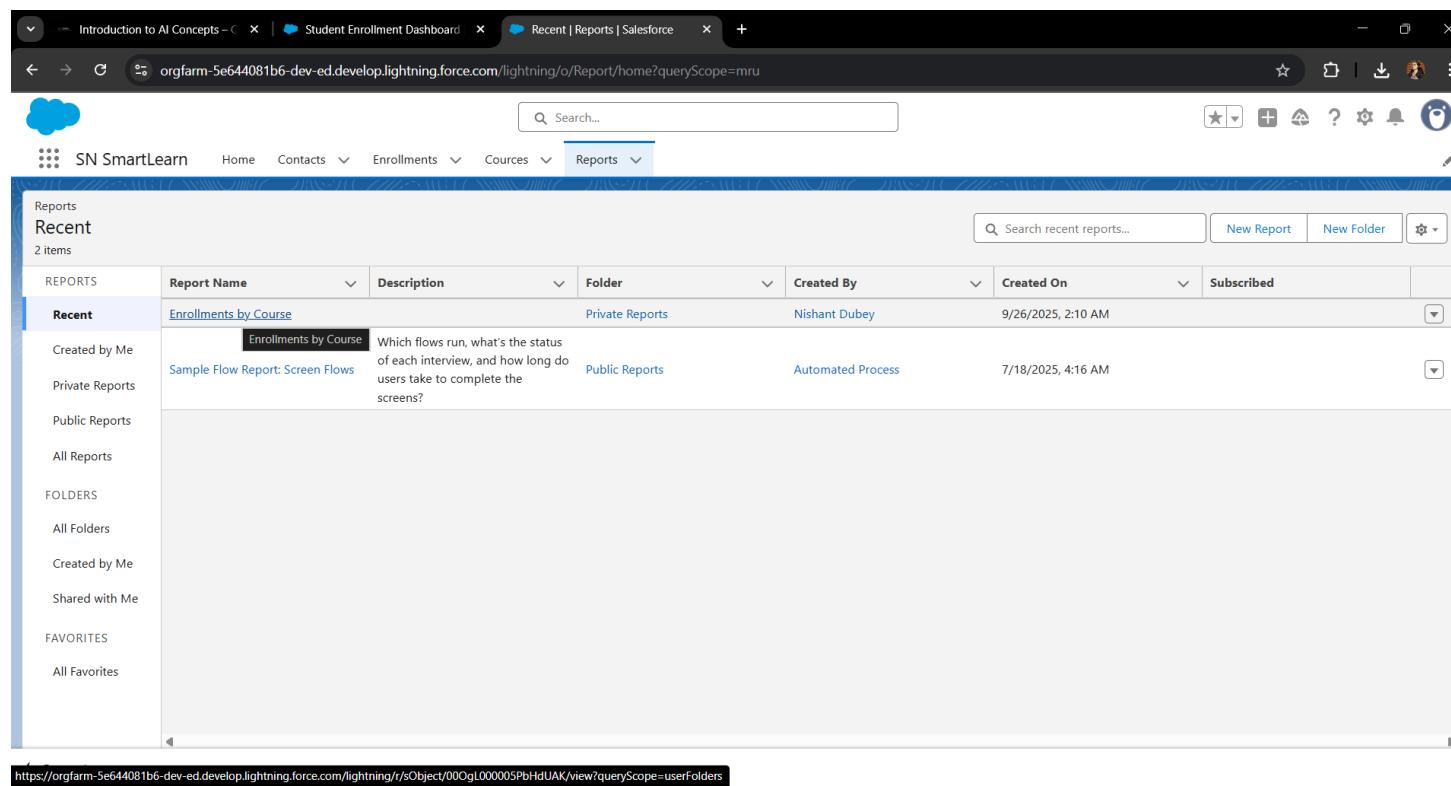
The screenshot shows the Salesforce Setup interface with the following details:

- Left Navigation:** Shows 'Setup' selected, followed by 'Object Manager' and 'Report Types'.
- Search Bar:** Displays 'report types'.
- Header:** 'Custom Report Types' with a 'SETUP' button.
- Section: Student with Enrollments**
 - Description: 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.
 - Buttons:** Preview Layout, Edit Layout, Clone, Delete, Close.
- Details Panel:** Contains fields:
 - Display Label:** Student with Enrollments
 - API Name:** Student_with_Enrollments
 - Description:** Report on students and the courses they are enrolled in
 - Created By:** Nishant Dubey, 9/26/25, 2:22 PM
 - Store in Category:** accounts
 - Deployment Status:** Deployed
 - Modified By:** Nishant Dubey, 9/26/25, 2:22 PM
- Object Relationships Panel:** Shows a Venn diagram where 'A' (Contacts) overlaps with 'B' (Enrollments). It states: "... with at least one related record from Enrollments (B)".
- Fields Panel:** Shows included fields from 'Contacts' and 'Enrollments'.

Source Object	Included Fields
Contacts	68
Enrollments	11

2. Reports

- **Use Case:** To provide Management with actionable insights into course popularity and enrollment numbers.
- **Steps Implemented:**
 1. A Summary Report named Enrollments by Course was built using the new custom report type.
 2. The report was configured to group records by the Course Name to show a count of students in each course.



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

- Header:** Introduction to AI Concepts - C, Student Enrollment Dashboard, Recent | Reports | Salesforce.
- Breadcrumbs:** orgfarm-5e644081b6-dev-ed.develop.lightning.force.com/lightning/o/Report/home?queryScope=mru
- Page Title:** SN SmartLearn
- Top Navigation:** Home, Contacts, Enrollments, Courses, Reports (selected).
- Search Bar:** Search...
- Recent Reports List:**

REPORTS	Report Name	Description	Folder	Created By	Created On	Subscribed
Recent	Enrollments by Course	Which flows run, what's the status of each interview, and how long do users take to complete the screens?	Private Reports	Nishant Dubey	9/26/2025, 2:10 AM	
	Sample Flow Report: Screen Flows		Public Reports	Automated Process	7/18/2025, 4:16 AM	
- Left Sidebar:**
 - Reports
 - Recent (2 items)
 - Created by Me
 - Private Reports
 - Public Reports
 - All Reports
 - FOLDERS
 - All Folders
 - Created by Me
 - Shared with Me
 - FAVORITES
 - All Favorites
- Page URL:** https://orgfarm-5e644081b6-dev-ed.develop.lightning.force.com/lightning/r/sObject/00OgL000005PbHdUAK/view?queryScope=userFolders

The screenshot shows the SN SmartLearn Report Builder interface. At the top, there are three tabs: 'Introduction to AI Concepts - C', 'Student Enrollment Dashboard', and 'Report Builder | Salesforce'. The 'Report Builder | Salesforce' tab is active. Below the tabs, the URL is orgfarm-5e644081b6-dev-ed.lightning.force.com/one/one.app#eyJb21wb25lbnREZWYiOjyZXBvcnRzOnJlcG9ydEJ1aWxkZXIiLCJhdHRyaWJ1dGVzIjp7InJY29yZEIkjoiwibmV3Um... The page title is 'SN SmartLearn'.

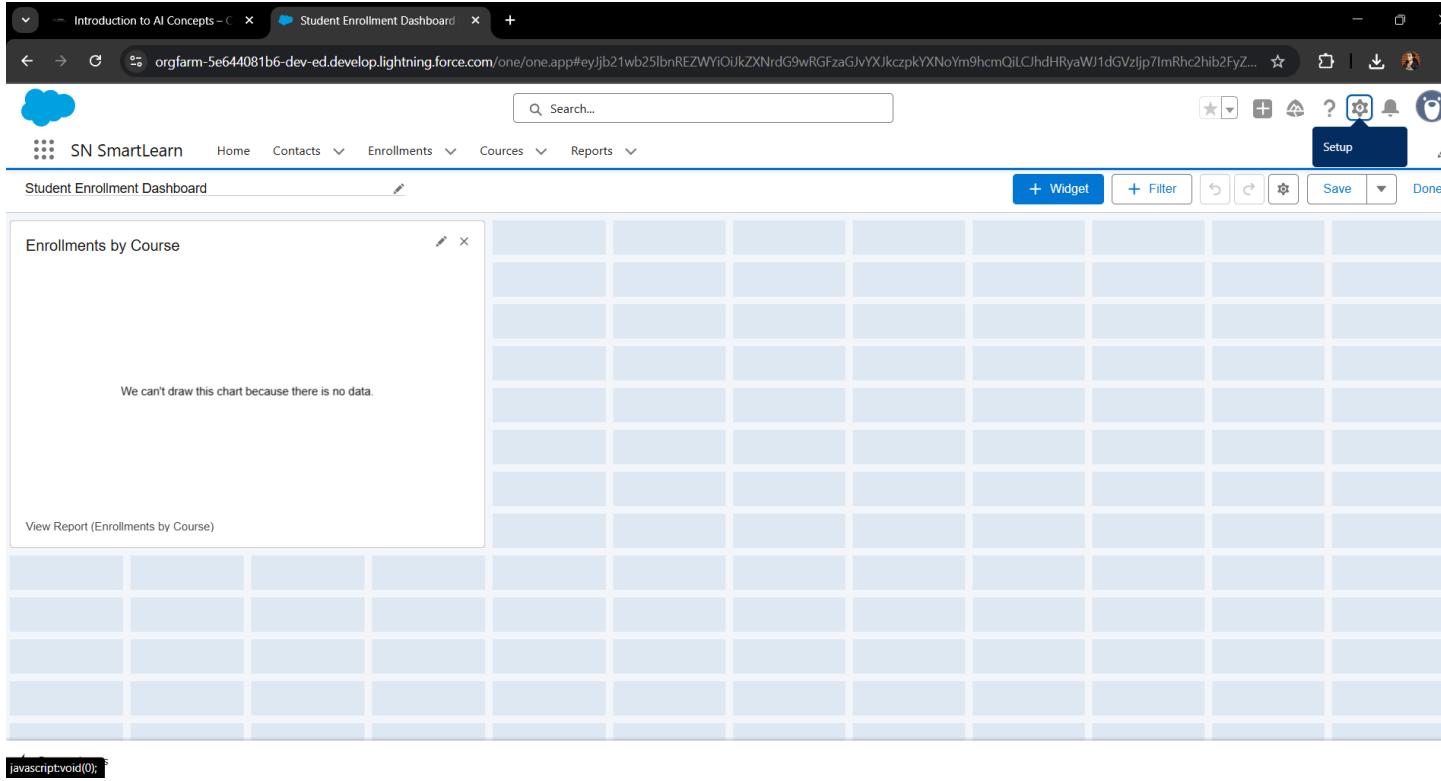
The main area is titled 'REPORT' with a dropdown menu. The current report is 'New Student with Enrollments Report' under the 'Student with Enrollments' section. On the right, there are buttons for 'Add Chart', 'Save & Run', 'Save', 'Close', and 'Run'.

The left sidebar is labeled 'Fields' and contains sections for 'Outline' (selected), 'Groups', and 'Columns'. Under 'Outline', there are 'GROUP ROWS' and 'GROUP COLUMNS' sections. Under 'Groups', there is a search bar for 'Source: Coarse Name' with a clear button. Under 'Columns', there are three items: 'Full Name', 'Enrollment Number', and 'Enrollment Status', each with a clear button. A note at the top says 'Previewing a limited number of records. Run the report to see everything.' Below the columns, there are checkboxes for 'Row Counts', 'Detail Rows', 'Subtotals', and 'Grand Total'.

At the bottom right, it says 'Currency: INR'.

3. Dashboards

- **Use Case:** To give Management a quick, visual overview of key enrollment metrics for strategic decision-making.
- **Steps Implemented:**
 1. A **Dashboard** named Student Enrollment Dashboard was created.
 2. A **Bar Chart** component was added to the dashboard, using the Enrollments by Course report as its source.



4. Dynamic Dashboards

- **Use Case:** To allow the same dashboard to show personalized data to different users (e.g., a manager sees all data, while an instructor sees only data for their own courses).
- **Steps Implemented:** The Student Enrollment Dashboard was edited, and its "View As" property was changed to "**The dashboard viewer**", making it a dynamic dashboard.

5. Sharing Settings (Review)

- **Use Case:** To confirm the foundational security model is secure and protects student data.

- **Steps Implemented:** The **Org-Wide Defaults (OWD)** were reviewed in **Sharing Settings**, confirming that the Contact and Enrollment objects were set to **Private** to ensure confidentiality.

Default Sharing Settings

Organization-Wide Defaults		Edit
Object	Default Internal Access	Default External Access
Lead	Private	Private
Account and Contract	Private	Private
Contact	Private	Private

6. Field-Level Security (FLS)

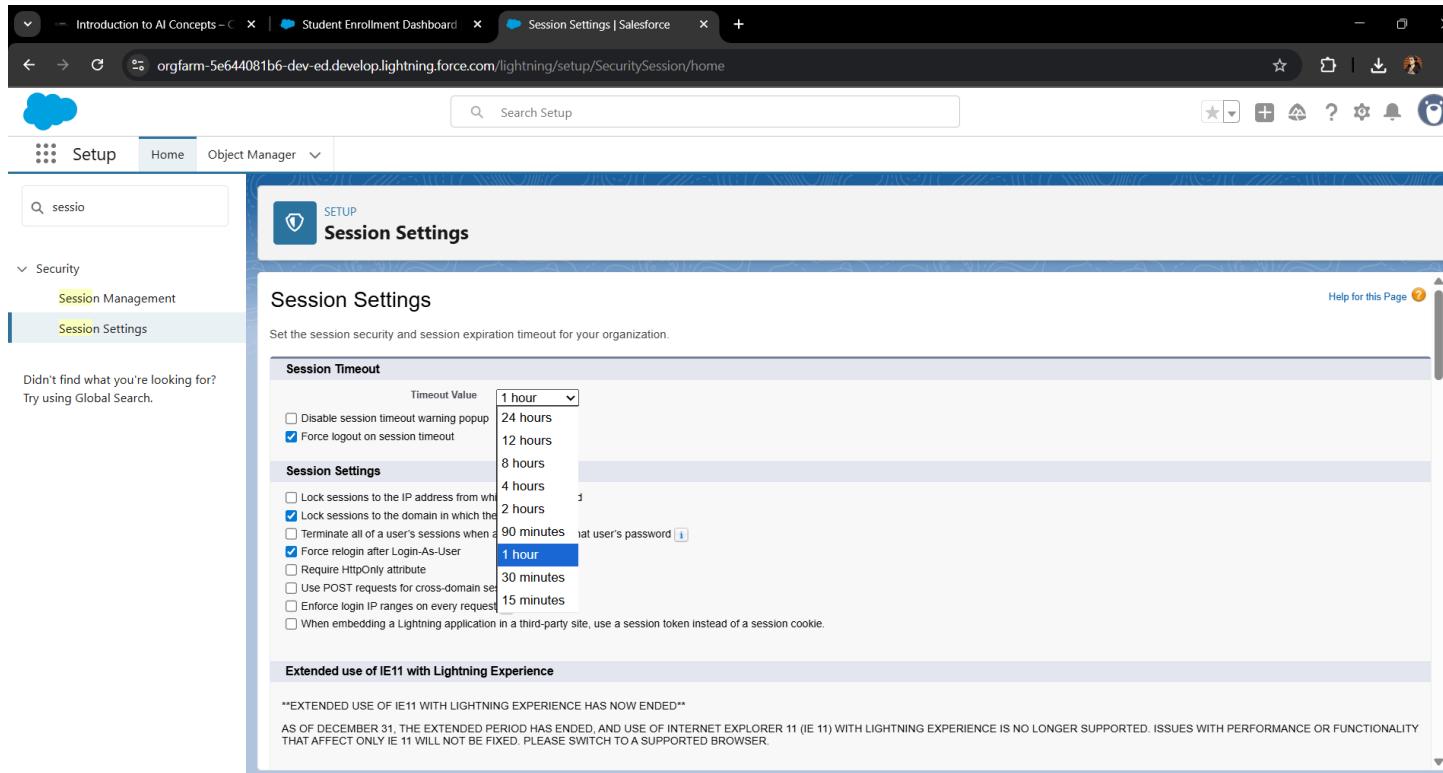
- **Use Case:** To protect sensitive data by hiding specific fields from profiles that do not need to see them.
- **Steps Implemented:** Field-Level Security was configured for a sensitive field (e.g., Discount Percentage) to be hidden from the Instructor profile while remaining visible to others.

The screenshot shows the Salesforce Setup interface. On the left, there's a sidebar with a search bar containing 'shar' and a 'Security' section. Under 'Sharing Settings', it lists 'Guest User Sharing Rule Access Report' and 'Sharing Settings'. A note says 'Didn't find what you're looking for? Try using Global Search.' The main content area is titled 'SETUP' and shows a list of user profiles. The 'Instructor Profile' is selected, indicated by a blue highlight. In the 'Visible' column for the 'Discount Percentage' field, there is a checked checkbox, which is highlighted with a black rectangle. Other profiles like 'Marketing User' and 'Standard User' have their checkboxes unchecked in this column.

User Profile	Visible
Guest User Sharing Rule Access Report	<input type="checkbox"/>
Sharing Settings	<input type="checkbox"/>
Instructor Profile	<input checked="" type="checkbox"/> Instructor Profile, Visible
Marketing User	<input type="checkbox"/>
Standard User	<input type="checkbox"/>

7. Session Settings

- **Use Case:** To enhance security by automatically logging out inactive users.
- **Steps Implemented:** The **Session Settings** in Setup were modified to reduce the session **Timeout Value** to 1 Hour.



The screenshot shows the Salesforce Setup interface for Session Settings. The left sidebar has 'Session Management' expanded, with 'Session Settings' selected. The main content area is titled 'Session Settings' and describes setting session security and expiration timeout. Under 'Session Timeout', the 'Timeout Value' dropdown is set to '1 hour'. Other options like 'Disable session timeout warning popups' and 'Force logout on session timeout' are checked. Below this, there's a section for 'Session Settings' with various checkboxes for locking sessions to IP, domain, or login-as-user, and requirements for POST requests and login ranges. At the bottom, a note about the extended use of IE11 is displayed.

8. Login IP Ranges

- **Use Case:** To restrict system access to trusted networks, such as the institution's physical campus or office.
- **Steps Implemented:** **Login IP Ranges** were configured on the Admissions profile to limit where users with that profile can log in from, preventing unauthorized access from unknown locations.

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

- Header:** Introduction to AI Concepts - C, Student Enrollment Dashboard, Profiles | Salesforce.
- Search Bar:** Search Setup
- Left Navigation:** Home, Object Manager, Setup (selected), and a search bar for "profile".
- Section:** Profiles
- Content:** A list of configuration items:
 - Login Hours:** No login hours specified.
 - Login IP Ranges:** A table with one entry: Action: IP Start Address (Edit | Del) 202.129.248.1, IP End Address: 202.129.248.255, Description: (empty).
 - Enabled Apex Class Access:** No Apex Classes enabled.
 - Enabled Visualforce Page Access:** No Visualforce Pages enabled.
 - Enabled External Data Source Access:** No External Data Sources enabled.
 - Enabled Named Credential Access:** No Named Credential enabled.
 - Enabled External Credential Principal Access:** (empty)

9. Audit Trail

- Use Case:** To maintain a log of all administrative changes for security and troubleshooting purposes.
- Steps Implemented:** The **Setup Audit Trail** was reviewed to understand how to track all setup changes and download the historical log.

[Download setup audit trail for last six months \(Excel .csv file\)](#)