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

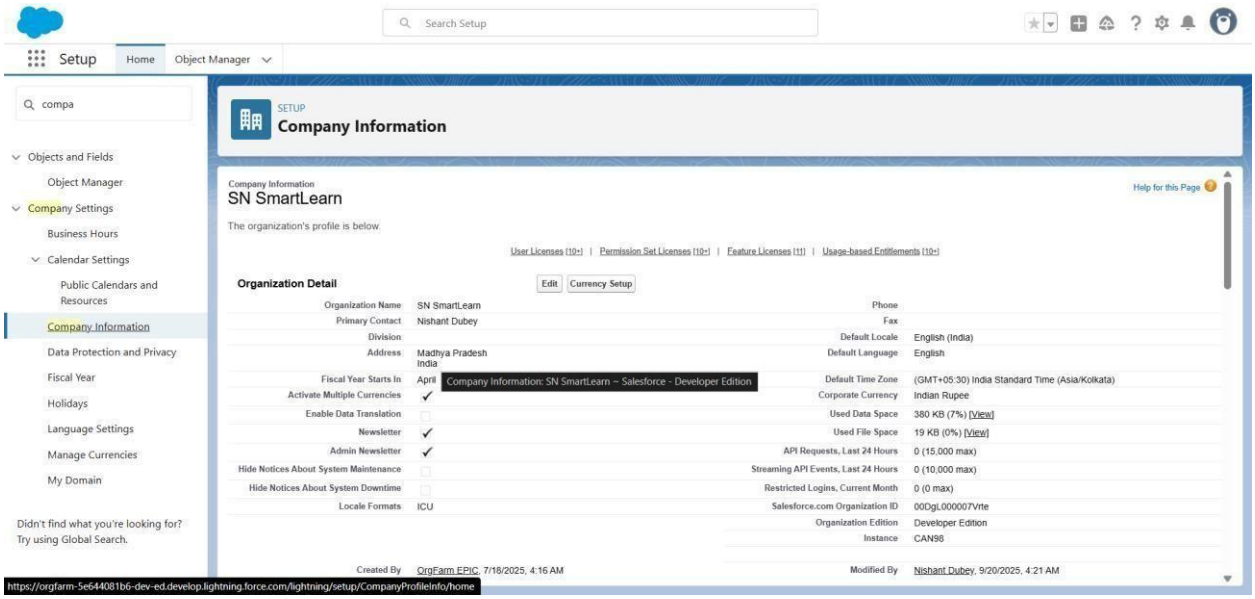
We used a **Salesforce Developer Edition Org** for this implementation. This edition was selected because it provides all the core CRM features required for our project, such as custom objects, roles, profiles, automation tools, and APIs. It also supports AppExchange integration, which we plan to explore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

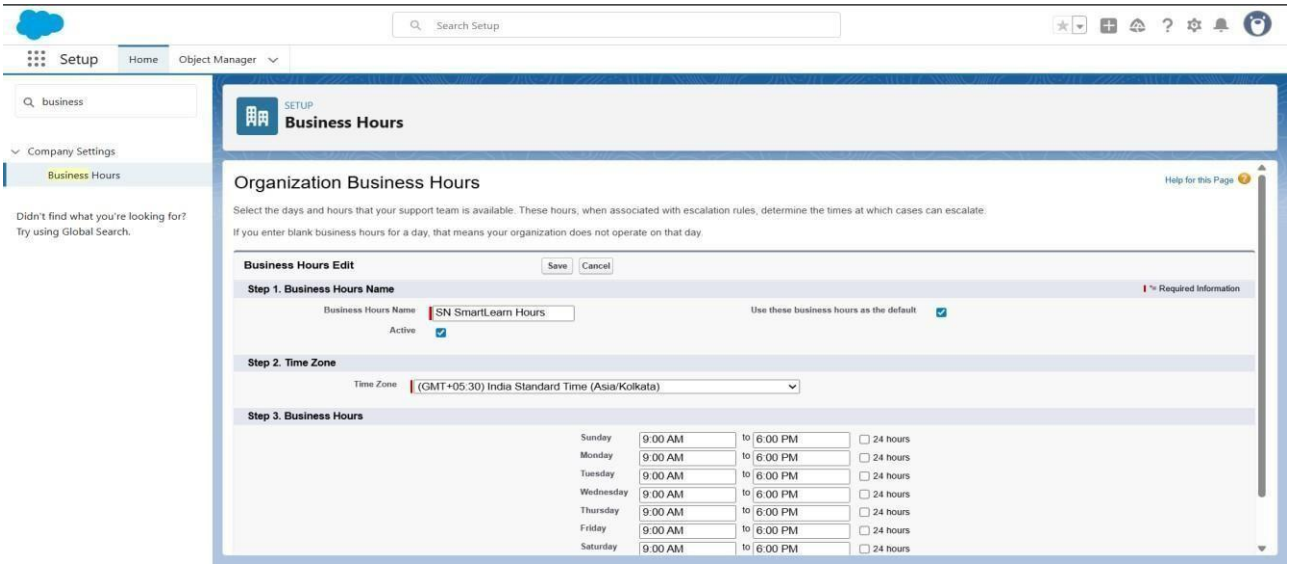
This ensures consistency across all student and instructor records, communications, and reports.



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**.

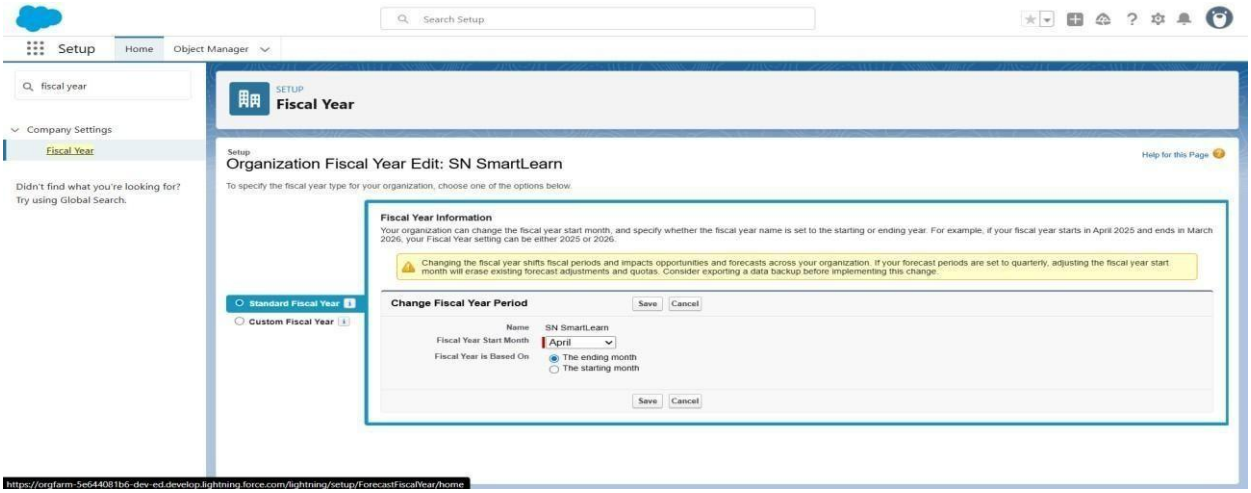
These settings ensure that automation processes like case escalations, reminders, and email alerts respect the organization's working schedule.



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.

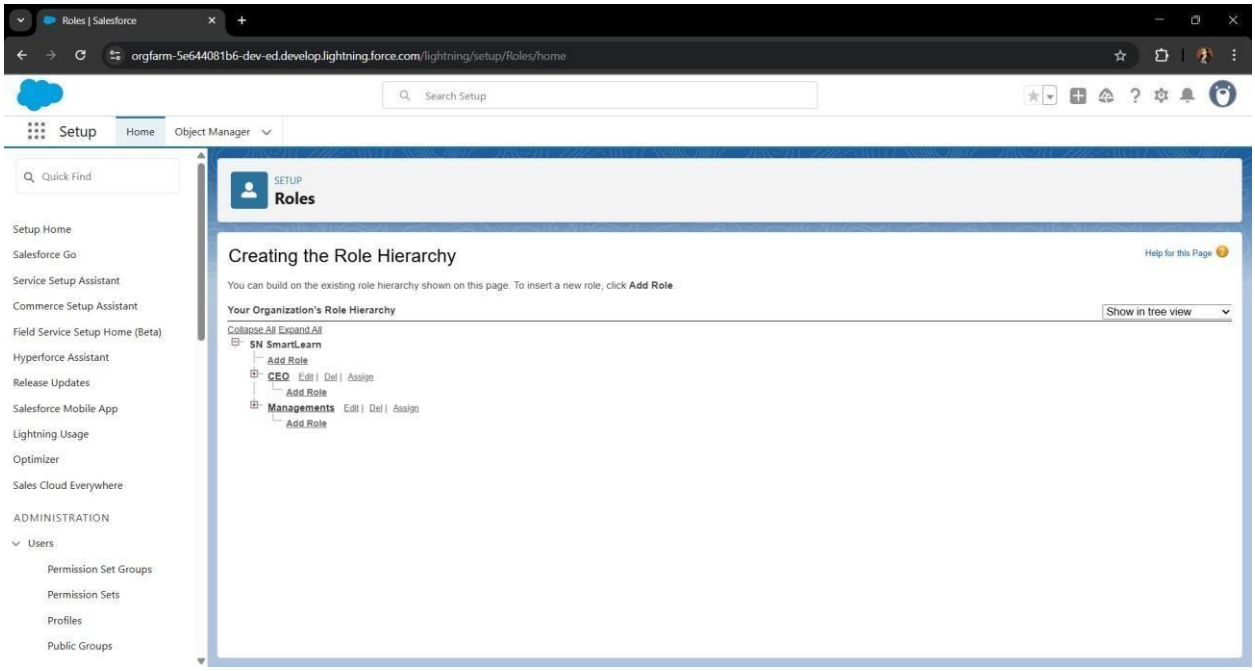
Profiles ensure each role has the exact level of access needed to perform their duties, reducing risks of unauthorized data exposure.

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.

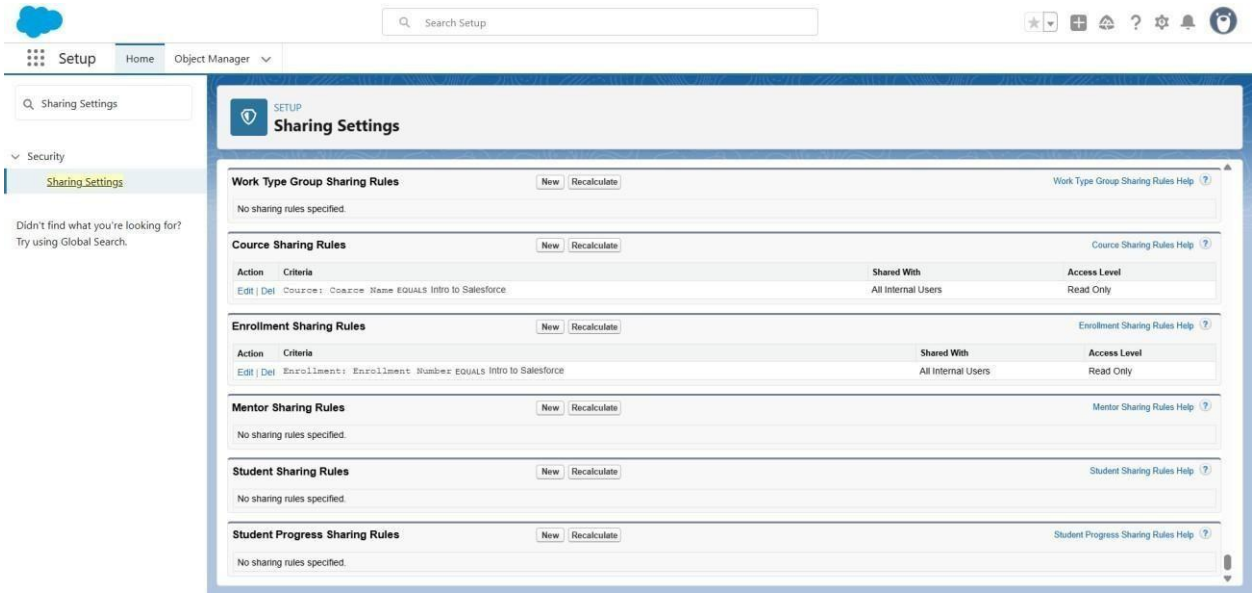
9. Org-Wide Defaults (OWD)

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.

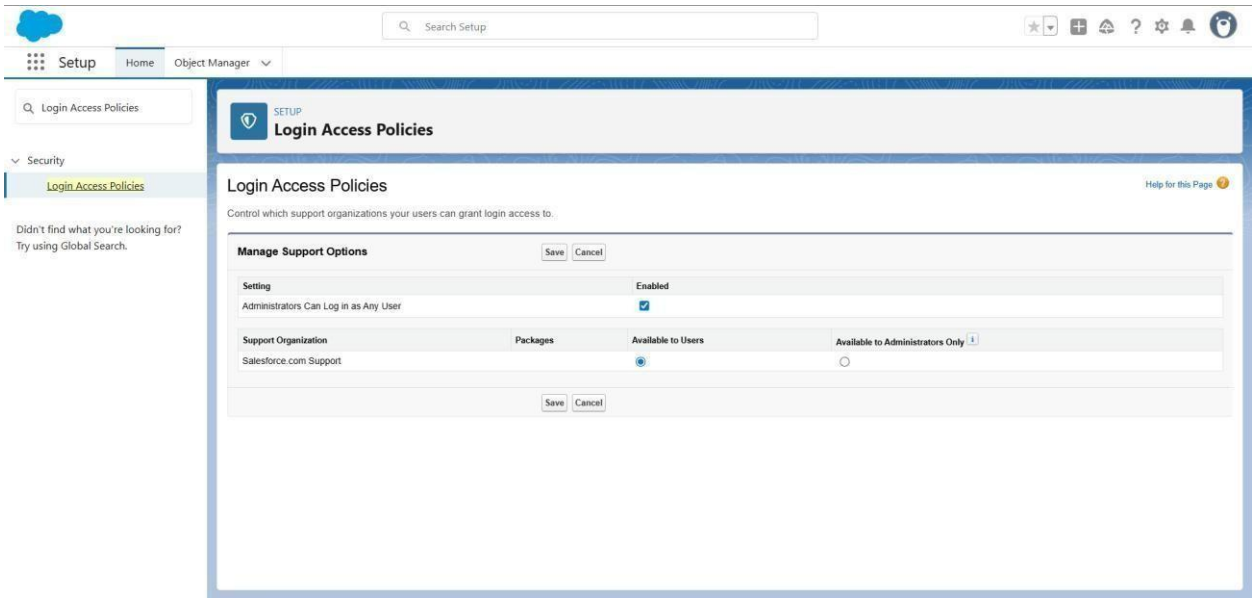
11. Login Access Policies

- 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

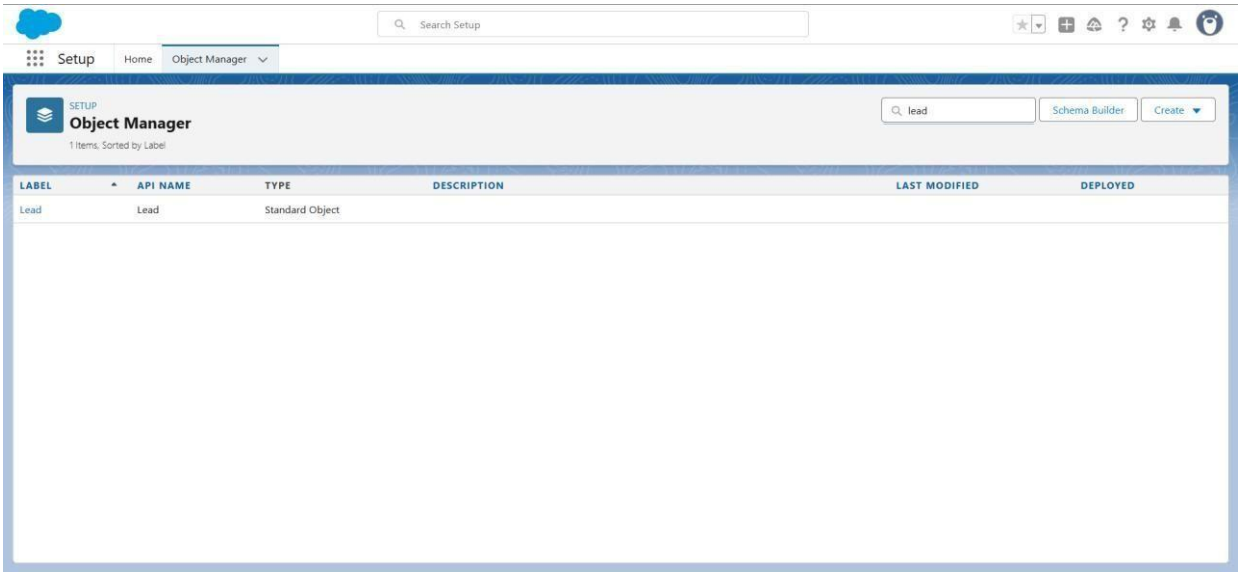
- Change Sets: Simple point-and-click deployment between orgs.
- Salesforce CLI (SFDX): Advanced deployment with version control and automation.
- GitHub Integration: Track changes, collaborate, and maintain version control.
- Always document deployment steps and maintain backups

Phase 3 : Data Modeling & Relationships

1. Standard & Custom Objects

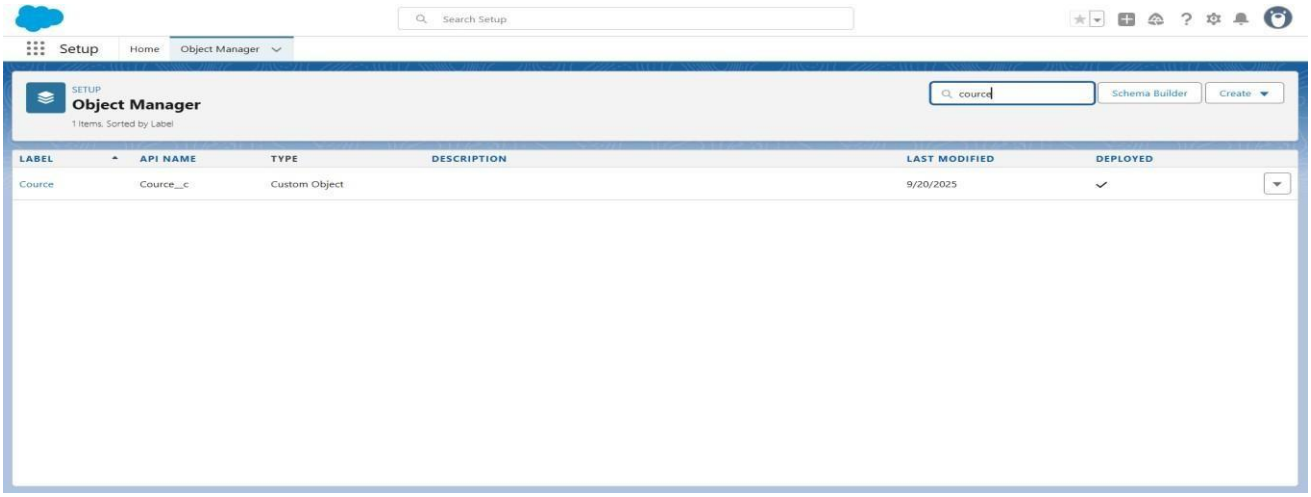
This is the foundation of your system. You will use a combination of standard and custom objects.

- **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.



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.

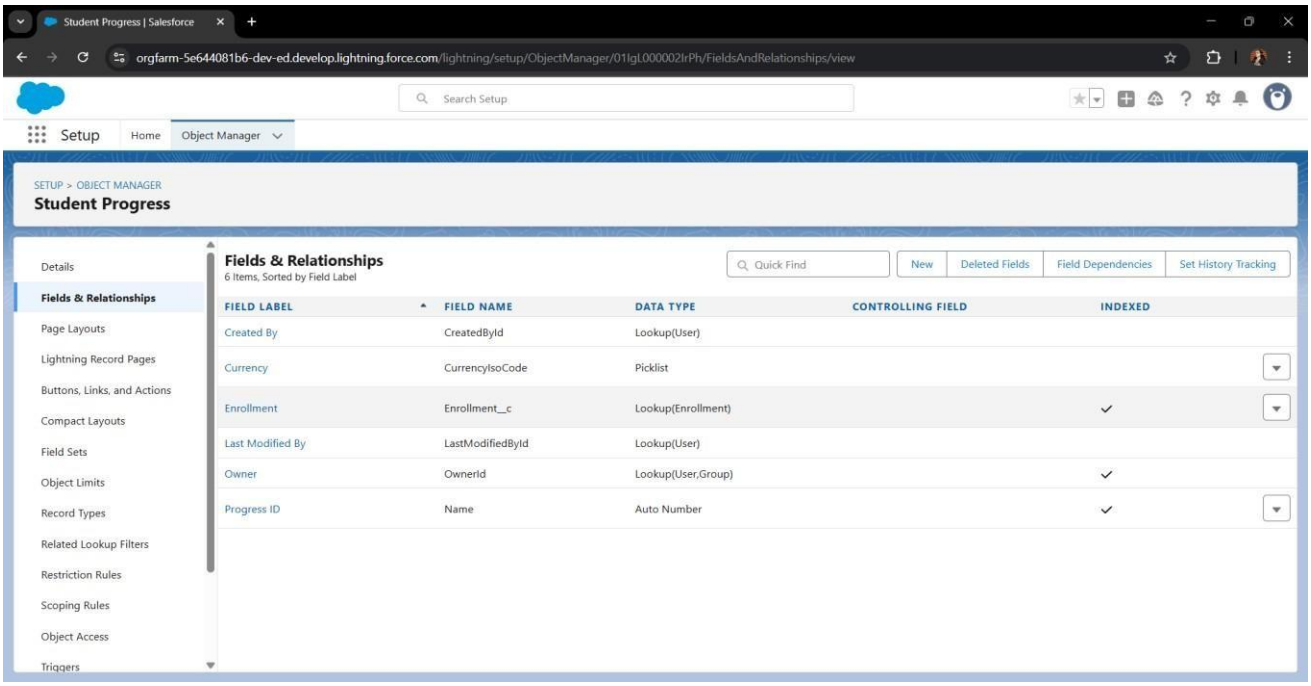


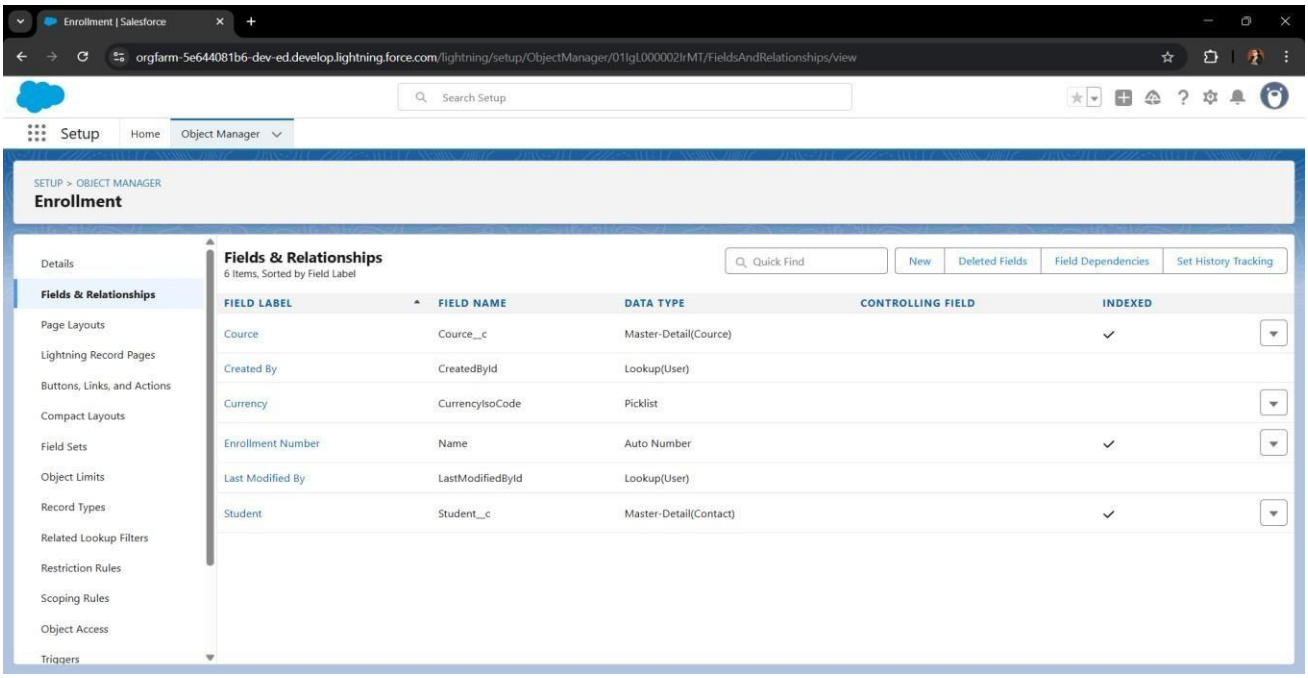
2. Fields

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.develop.lightning.force.com/lightning/setup/ObjectManager/01IgL000002IrPh/FieldsAndRelationships/view
Search Setup
Setup
Home
Object Manager
SETUP > OBJECT MANAGER
Student Progress
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
Fields & Relationships
6 Items, Sorted by Field Label
Quick Find
New
Deleted Fields
Field Dependencies
Set History Tracking
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


Enrollment | Salesforce
orgfarm-5e644081b6-dev-ed.develop.lightning.force.com/lightning/setup/ObjectManager/01IgL000002IrMT/FieldsAndRelationships/view
Search Setup
Setup
Home
Object Manager
SETUP > OBJECT MANAGER
Enrollment
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
Fields & Relationships
6 Items, Sorted by Field Label
Quick Find
New
Deleted Fields
Field Dependencies
Set History Tracking
FIELD LABEL
FIELD NAME
DATA TYPE
CONTROLLING FIELD
INDEXED
Cource
Cource__c
Master-Detail(Cource)
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)

3. Record Types & Page Layouts

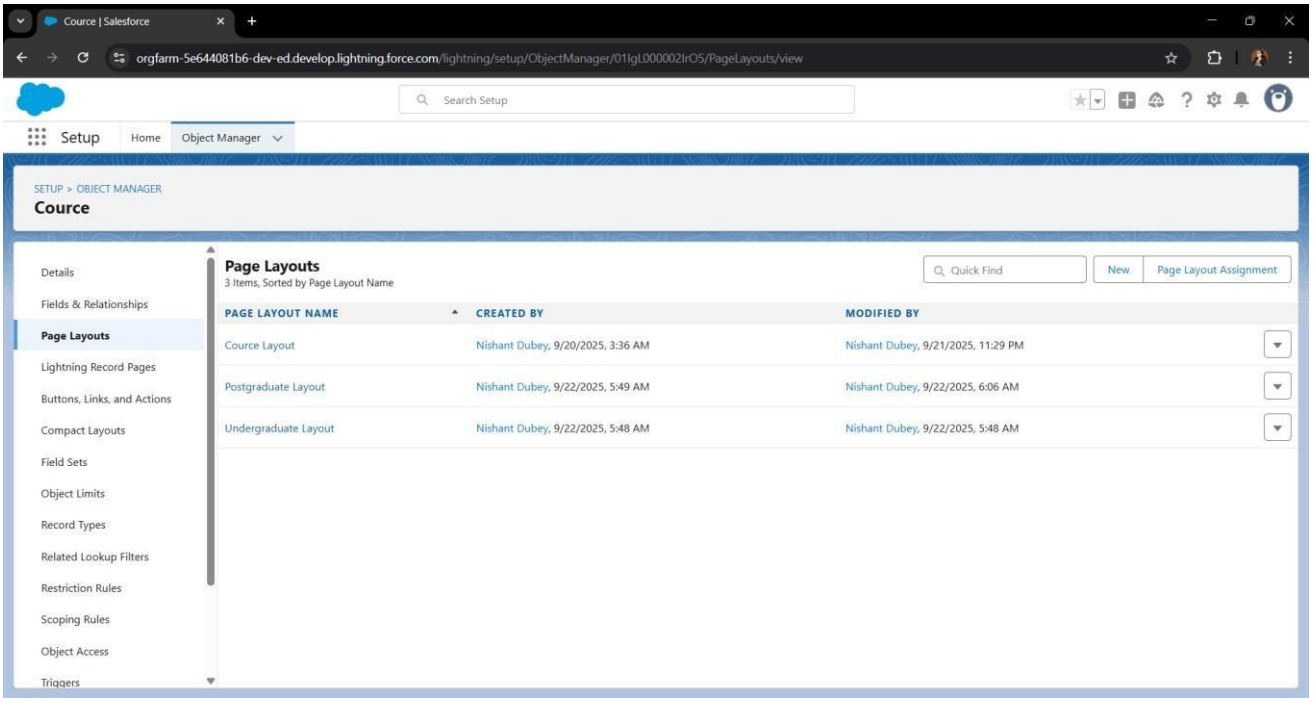
These allow you to customize the user experience for different processes. For example, you can create different page layouts on the

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 created new record type: "Thesis Required", assigning the corresponding layout.

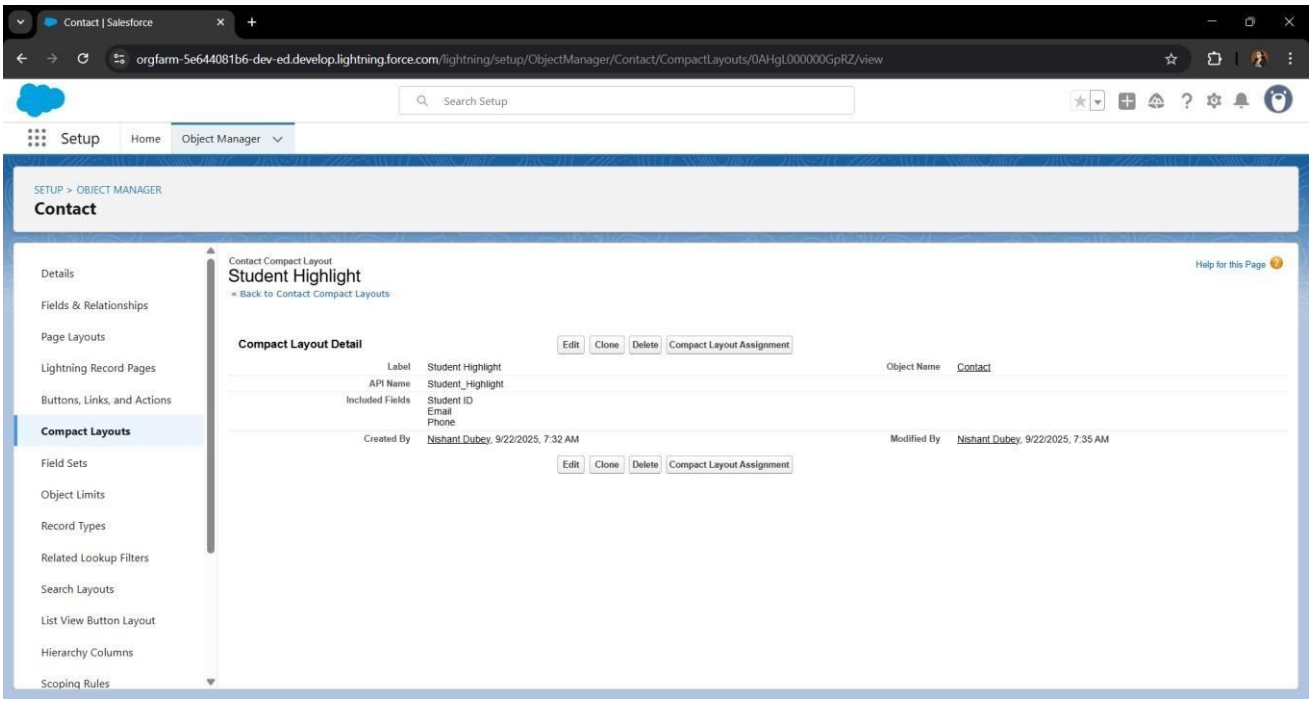


4. Compact Layouts

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

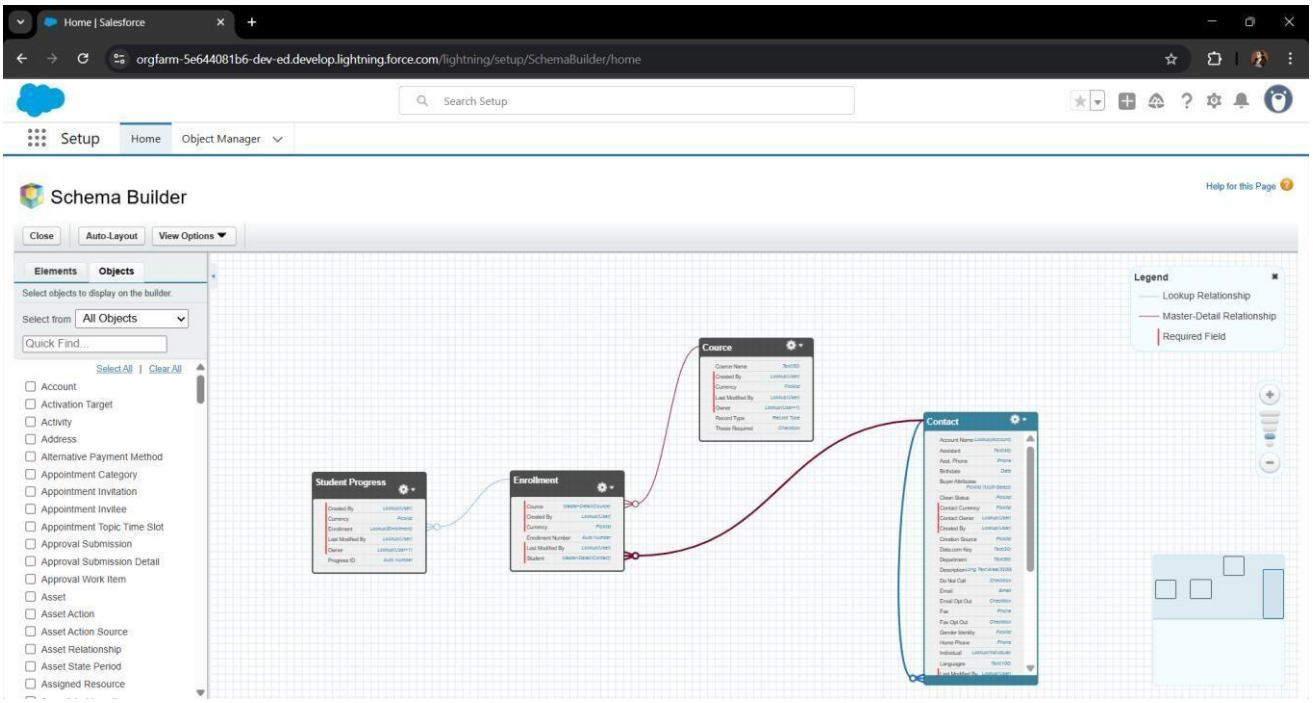
Use this tool to visualize your completed data model.

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

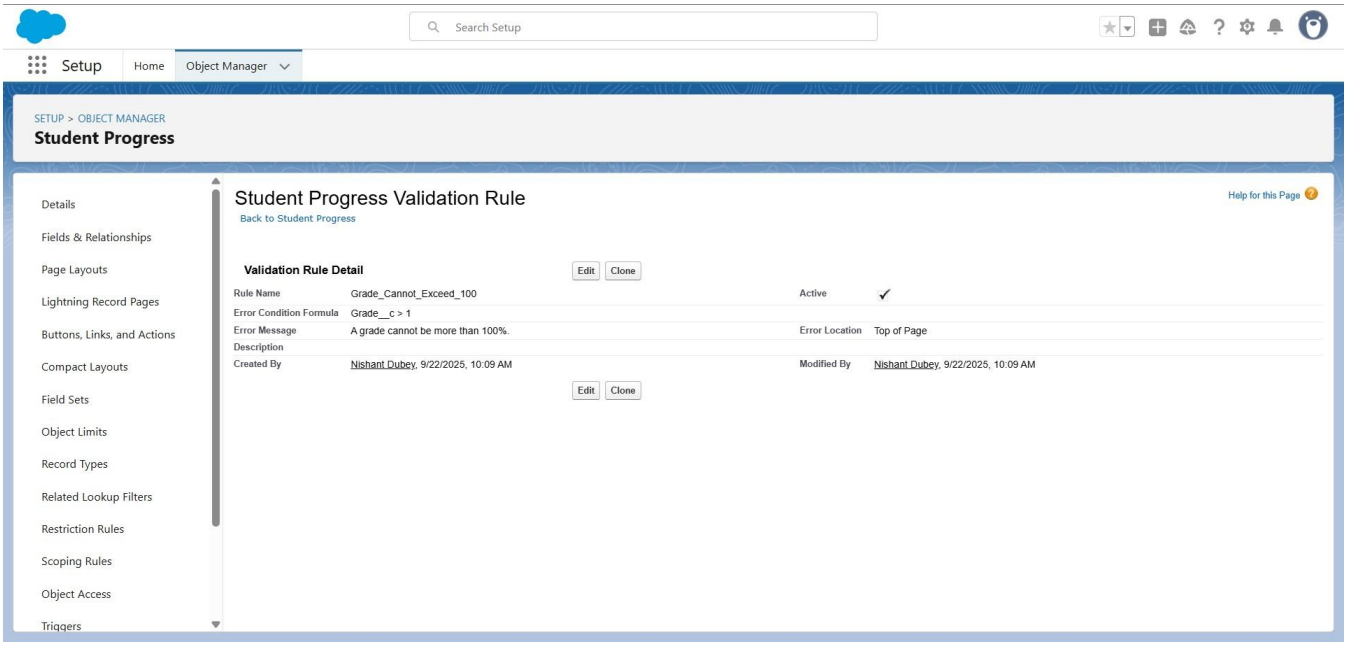
Validation Rules enforce data integrity by preventing users from saving records with invalid values.

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:
5. Grade__c > 100
6. 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.



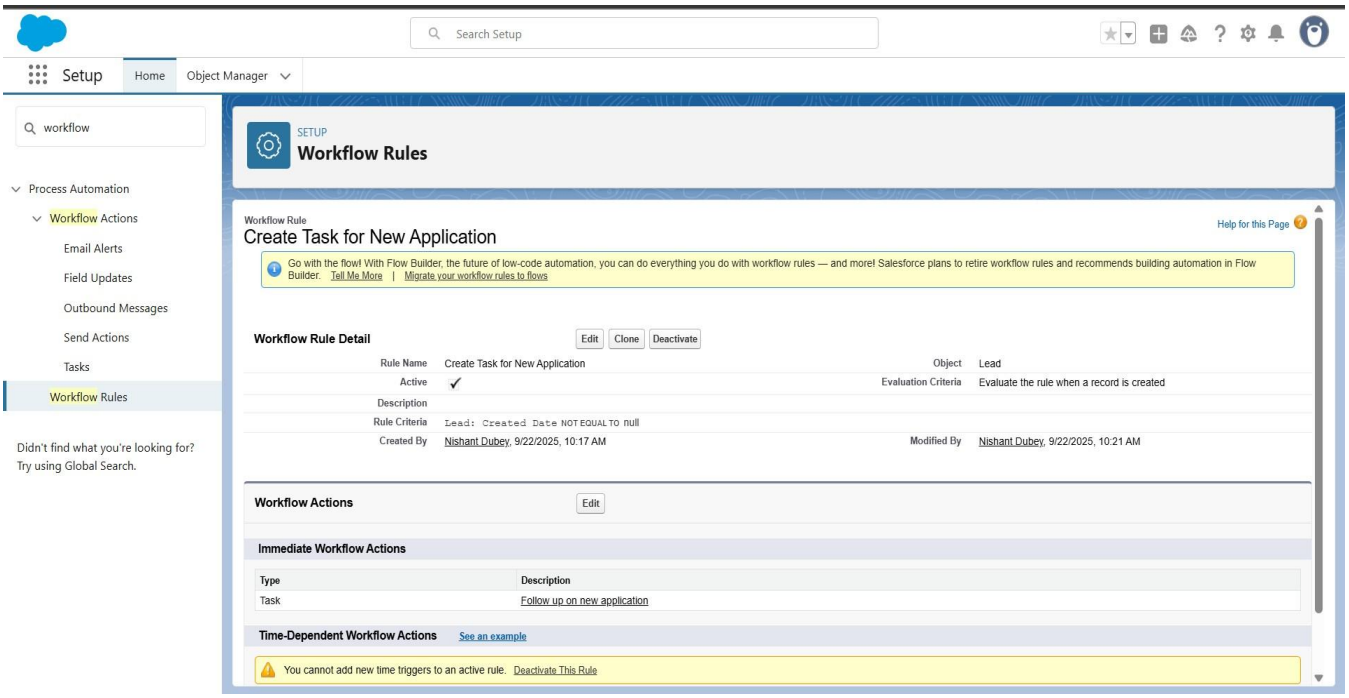
2. Workflow Rules (Legacy Tool)

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.



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.

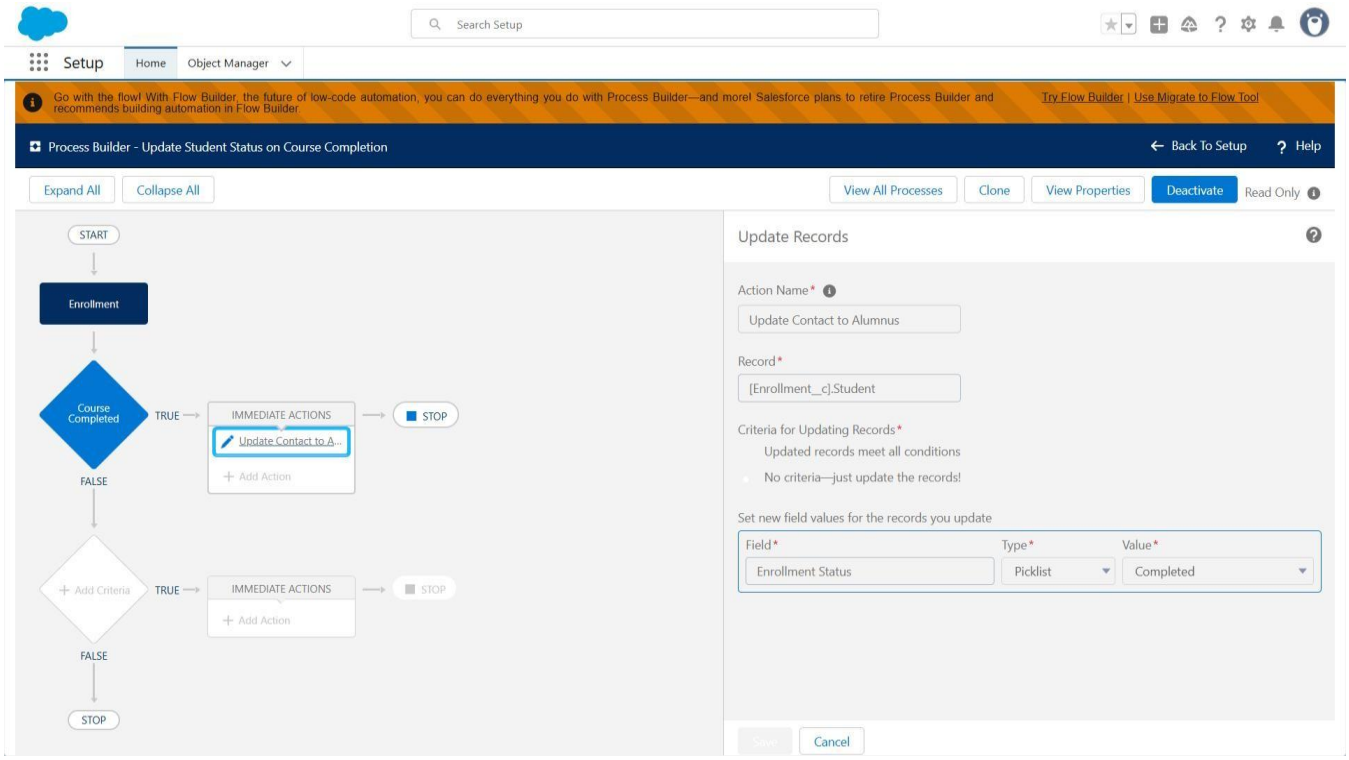
5. Add Criteria:

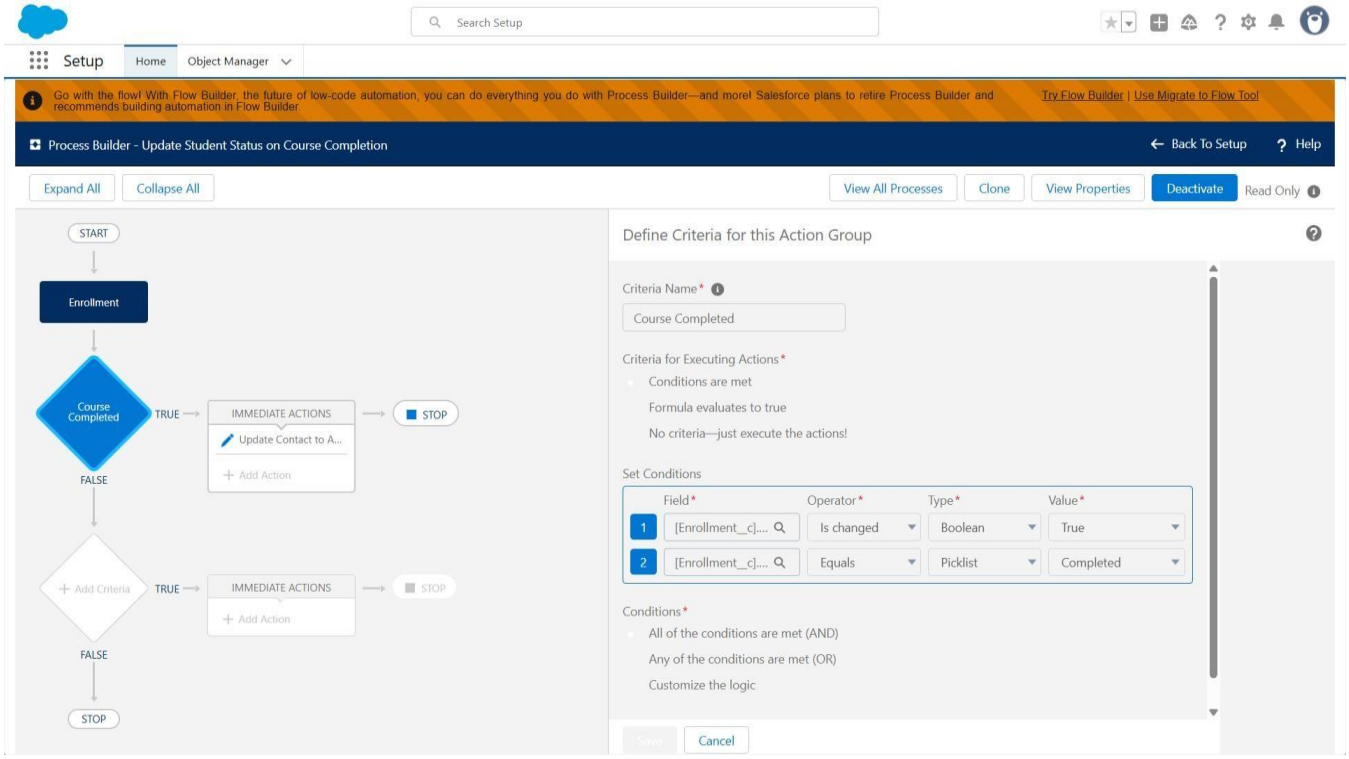
    - Criteria Name: *Course Completed*.
    - Conditions:
        - Status__c Is Changed = True.
        - Status__c Equals = “Completed”.

6. Immediate Action → Update Records:

    - Record Type: Student (related Contact).
    - New Value: Enrollment_Status__c = "Completed".

7. Save → Activate.


Search Setup
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.
Try Flow Builder | Use Migrate to Flow Tool
Process Builder - Update Student Status on Course Completion
Back To Setup
Help
Expand All
Collapse All
View All Processes
Clone
View Properties
Deactivate
Read Only
START
Enrollment
Course Completed
TRUE
IMMEDIATE ACTIONS
Update Contact to A...
Add Action
STOP
FALSE
Add Criteria
TRUE
IMMEDIATE ACTIONS
Add Action
STOP
FALSE
STOP
Update Records
Action Name*
Update Contact to Alumnus
Record*
[Enrollment__c].Student
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


Search Setup
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.
Try Flow Builder | Use Migrate to Flow Tool
Process Builder - Update Student Status on Course Completion
Back To Setup
Help
Expand All
Collapse All
View All Processes
Clone
View Properties
Deactivate
Read Only
START
Enrollment
Course Completed
TRUE
IMMEDIATE ACTIONS
Update Contact to A...
Add Action
STOP
FALSE
Add Criteria
TRUE
IMMEDIATE ACTIONS
Add Action
STOP
FALSE
STOP
Define Criteria for this Action Group
Criteria Name*
Course Completed
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

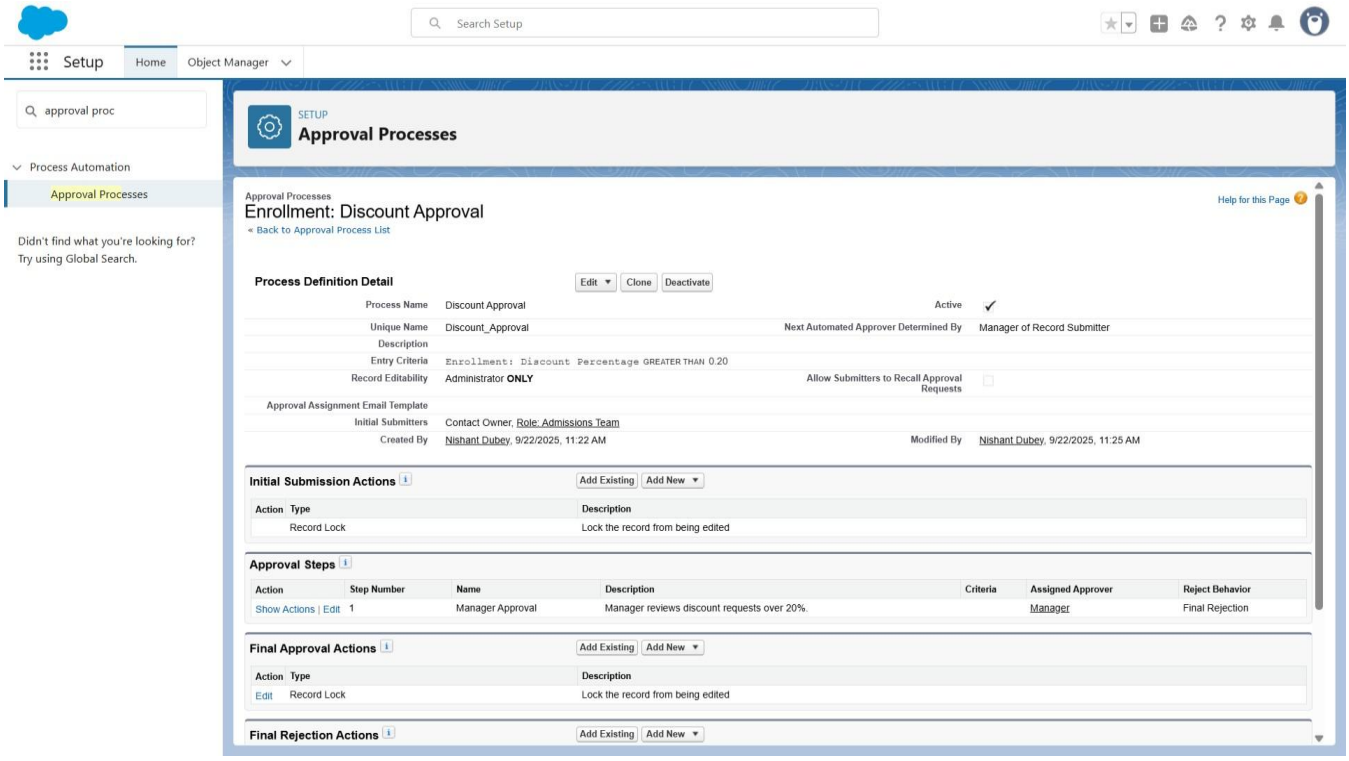
4. Approval Process

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.



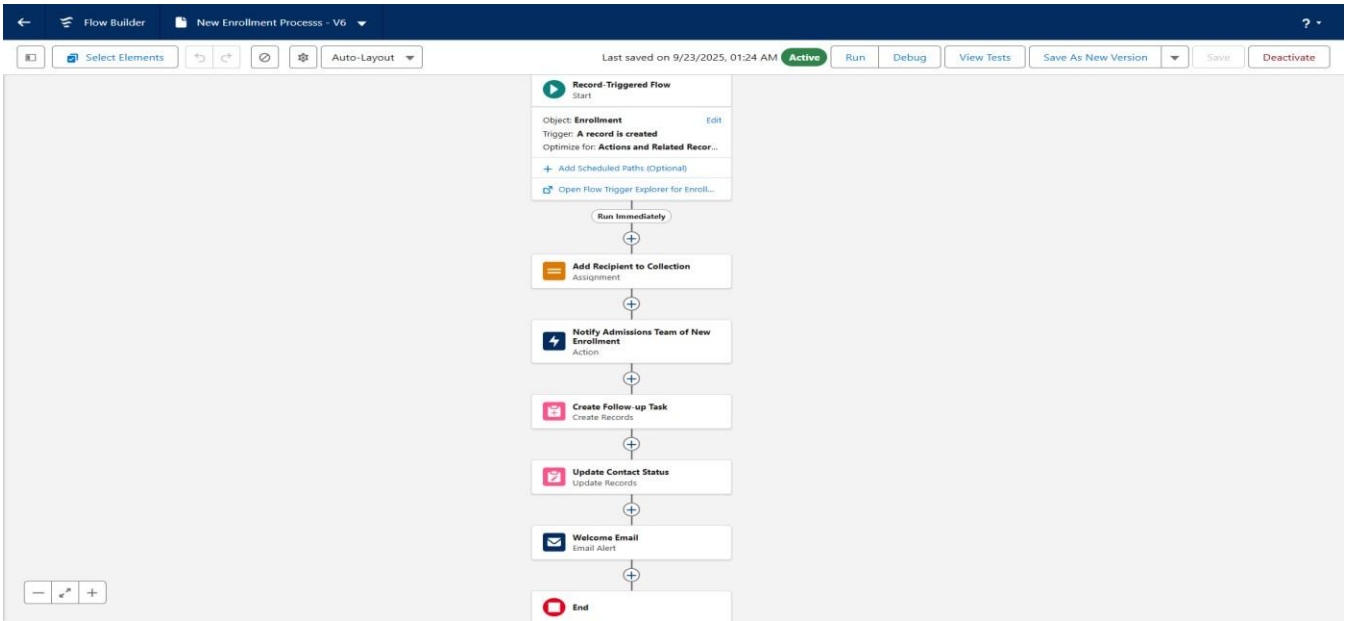
5. Flow Builder

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



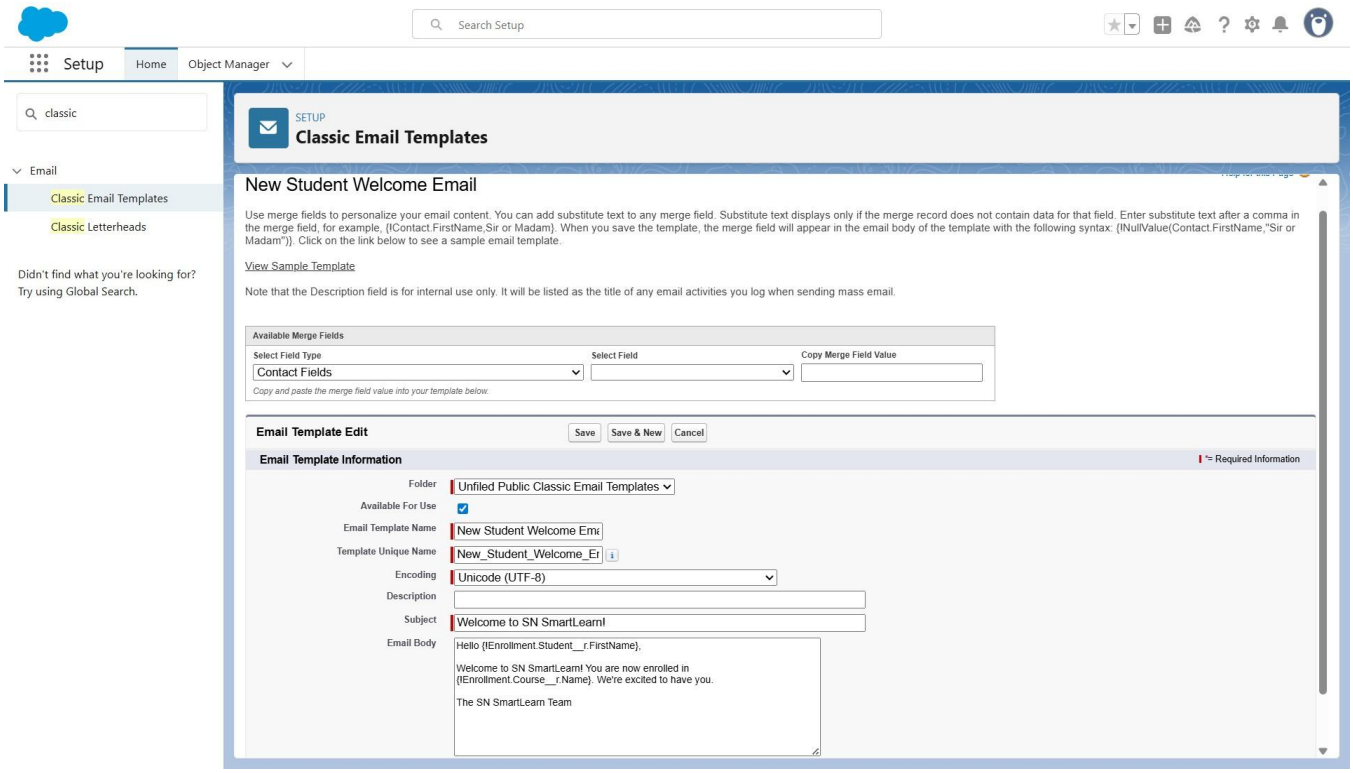
6. Email Alerts

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.



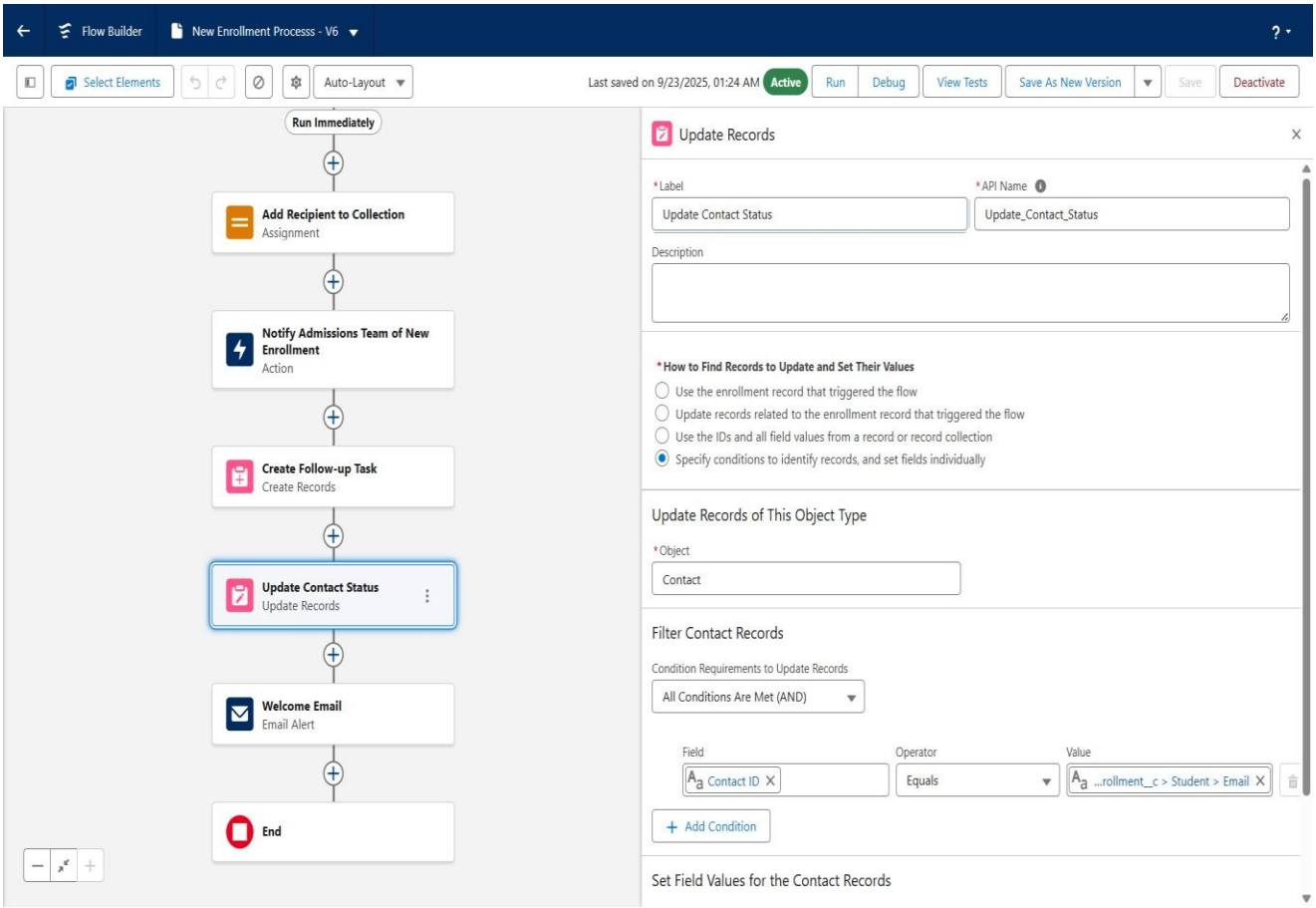
7. Field Updates

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.



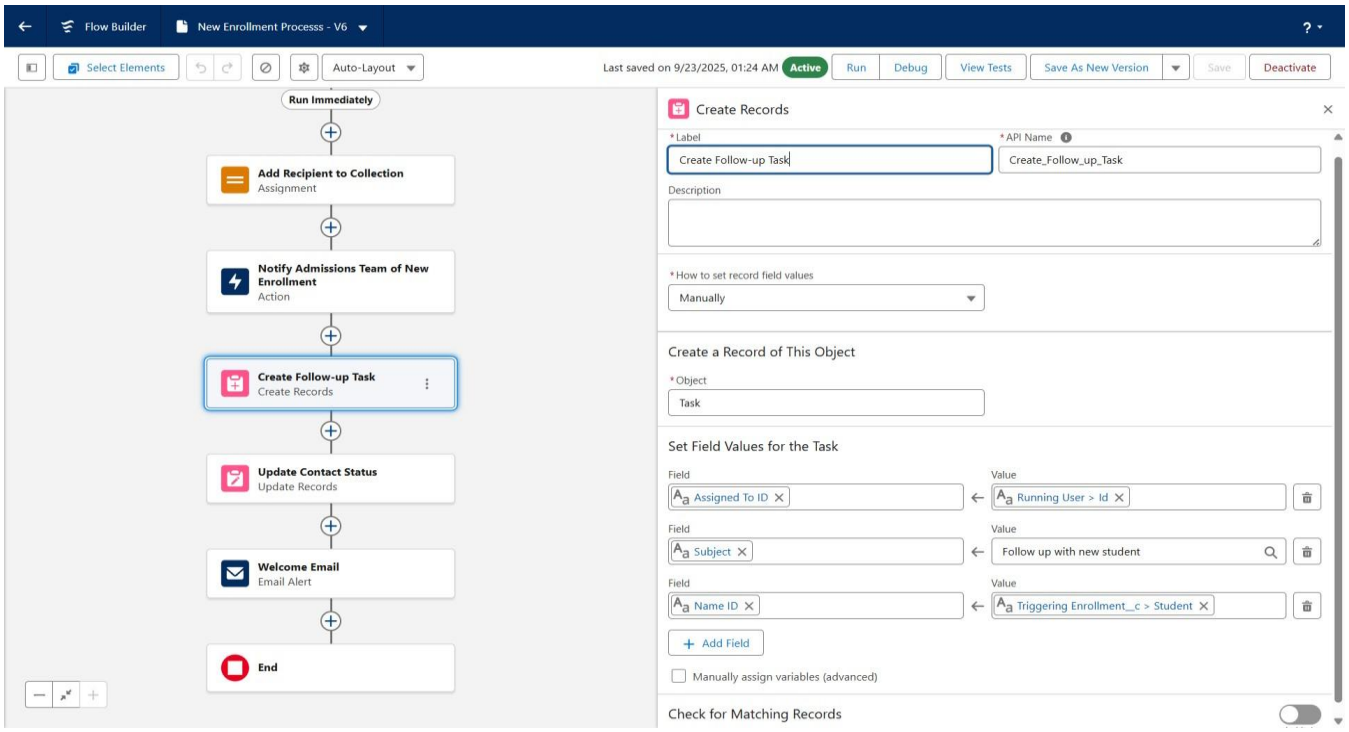
8. Tasks

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*. ○ WhoId = Student ($Record.Student__c).
   - OwnerId = Record Owner ($Record.OwnerId).
4. Save → Activate.



9. Custom Notifications

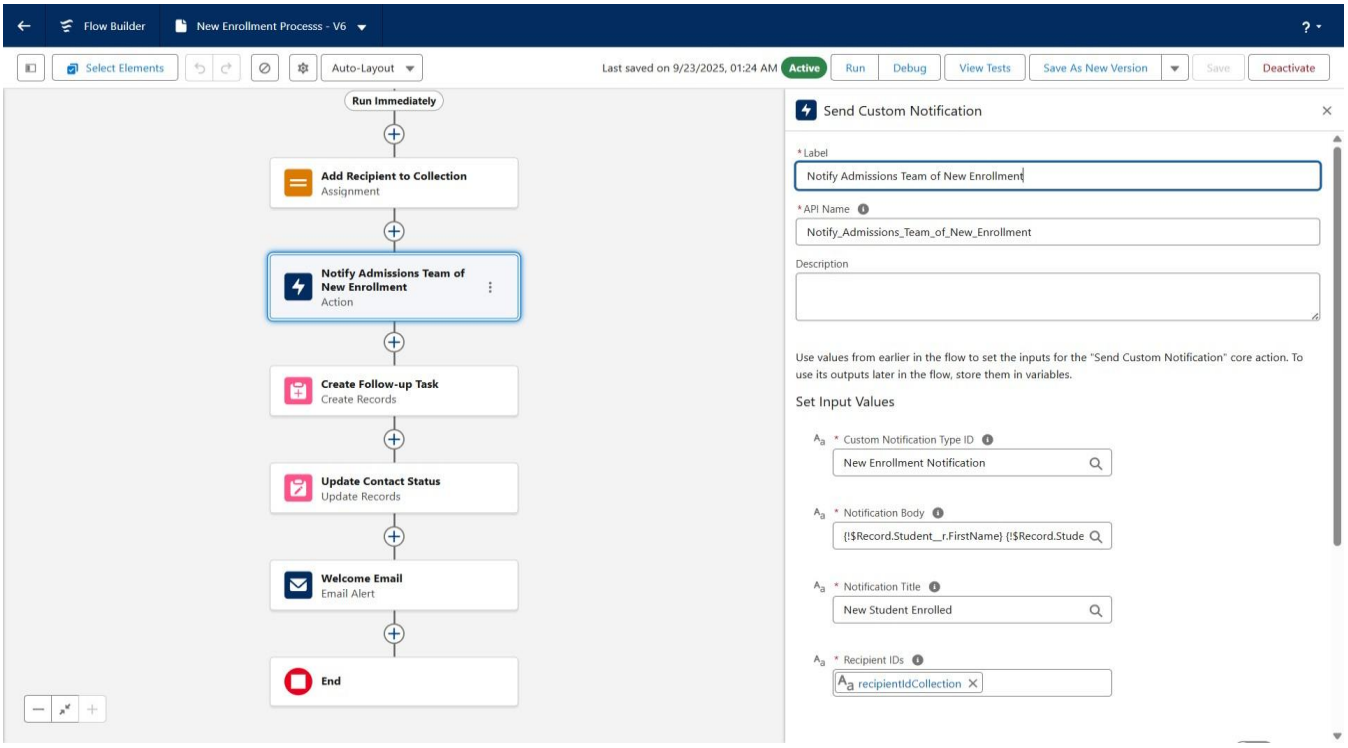
Custom Notifications send alerts to users in Salesforce (bell icon & mobile).

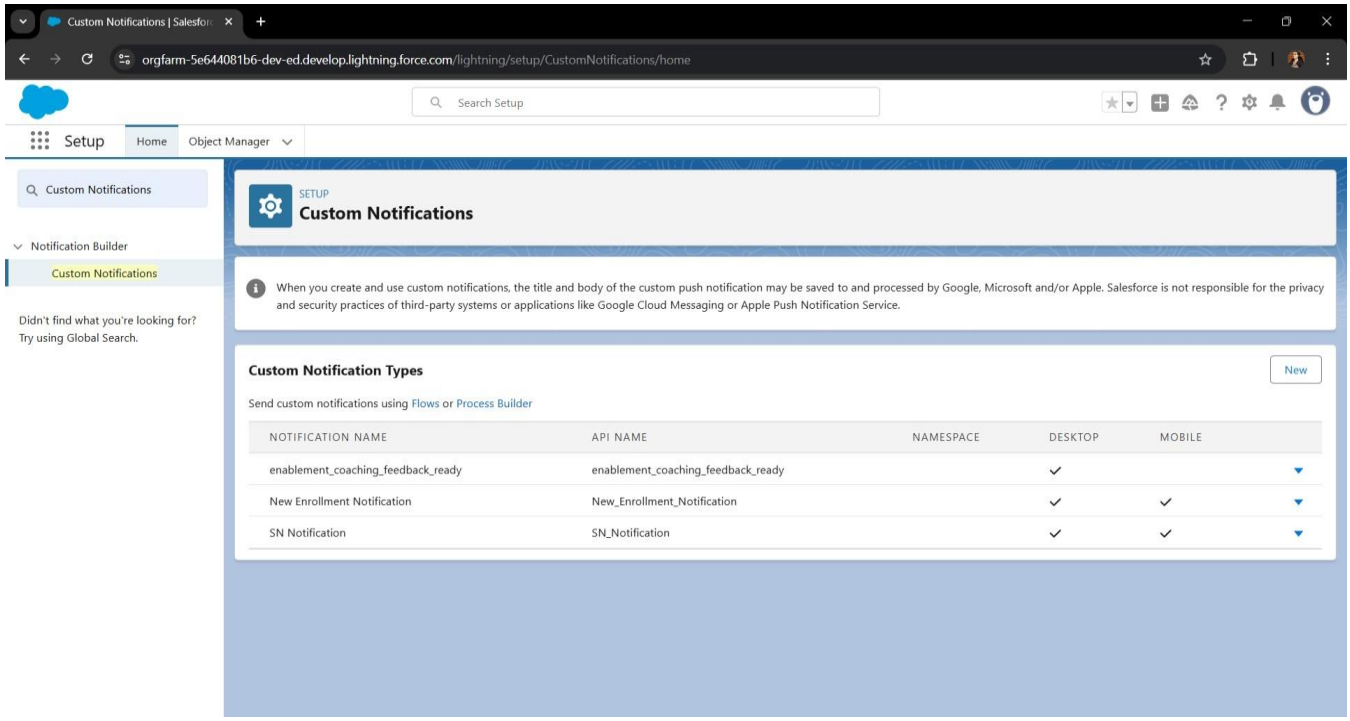
Use Case: Notify Admissions Team Lead when a new student enrolls.

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.





.

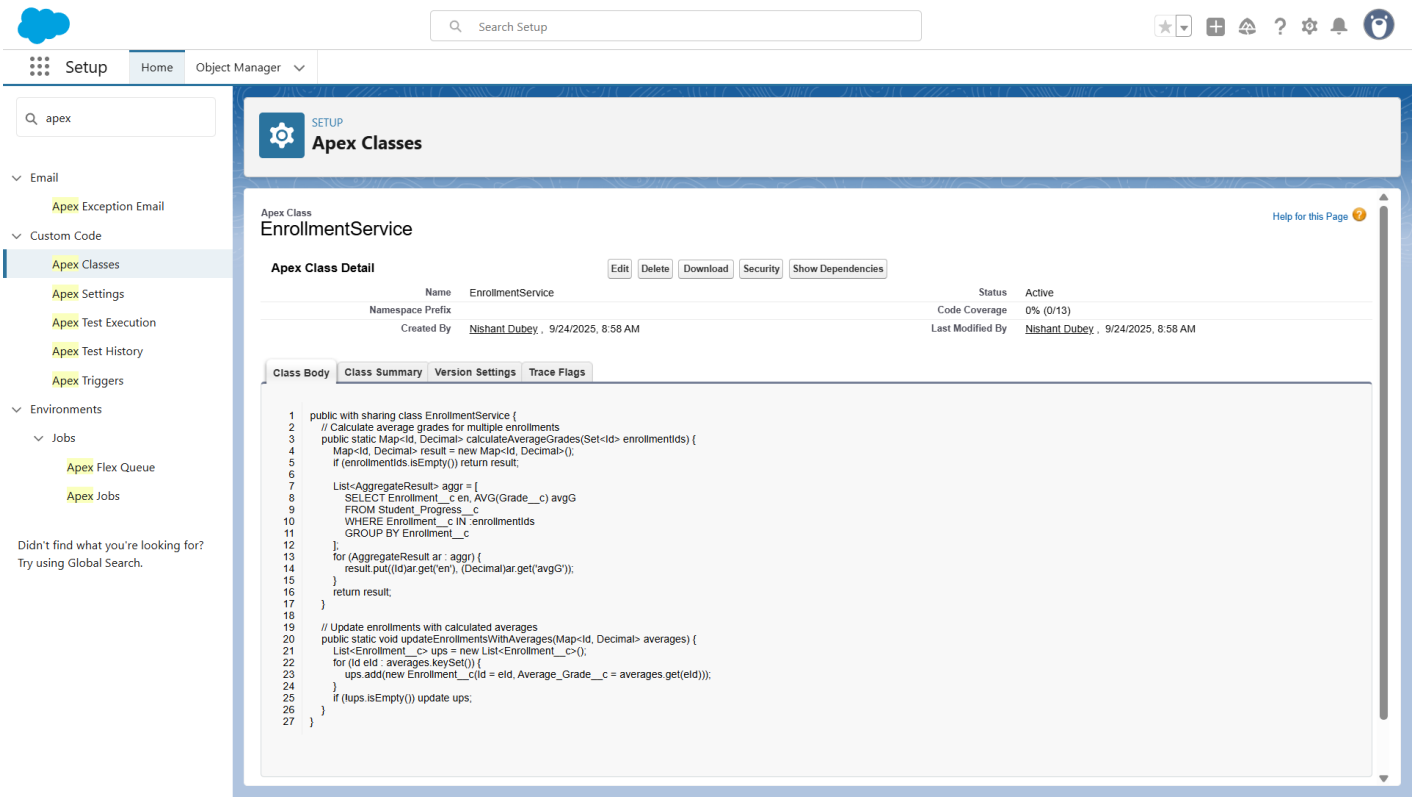
Phase 5 : Apex Programming (Developer Side)

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.



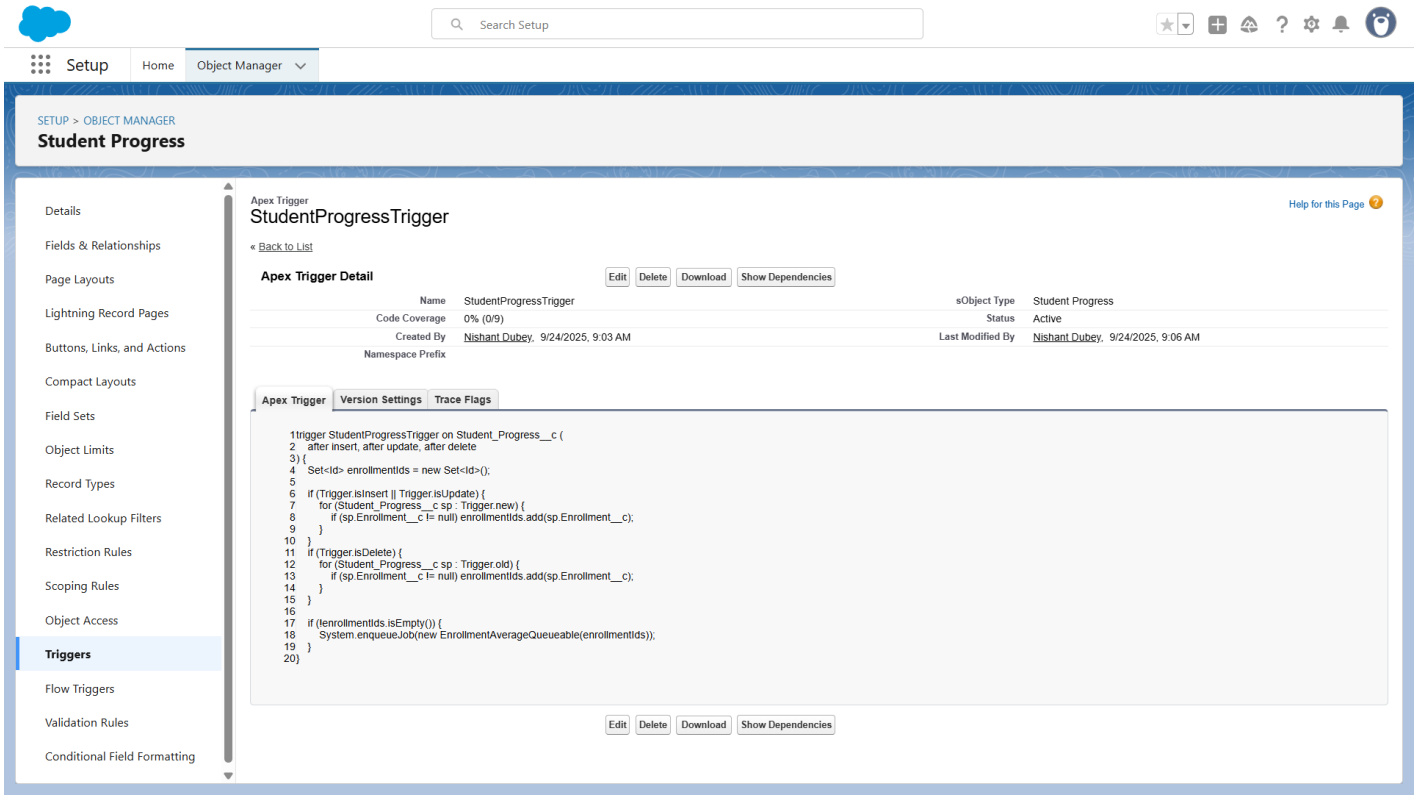
2. Apex Triggers (before/after insert/update/delete)

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.



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
```

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

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

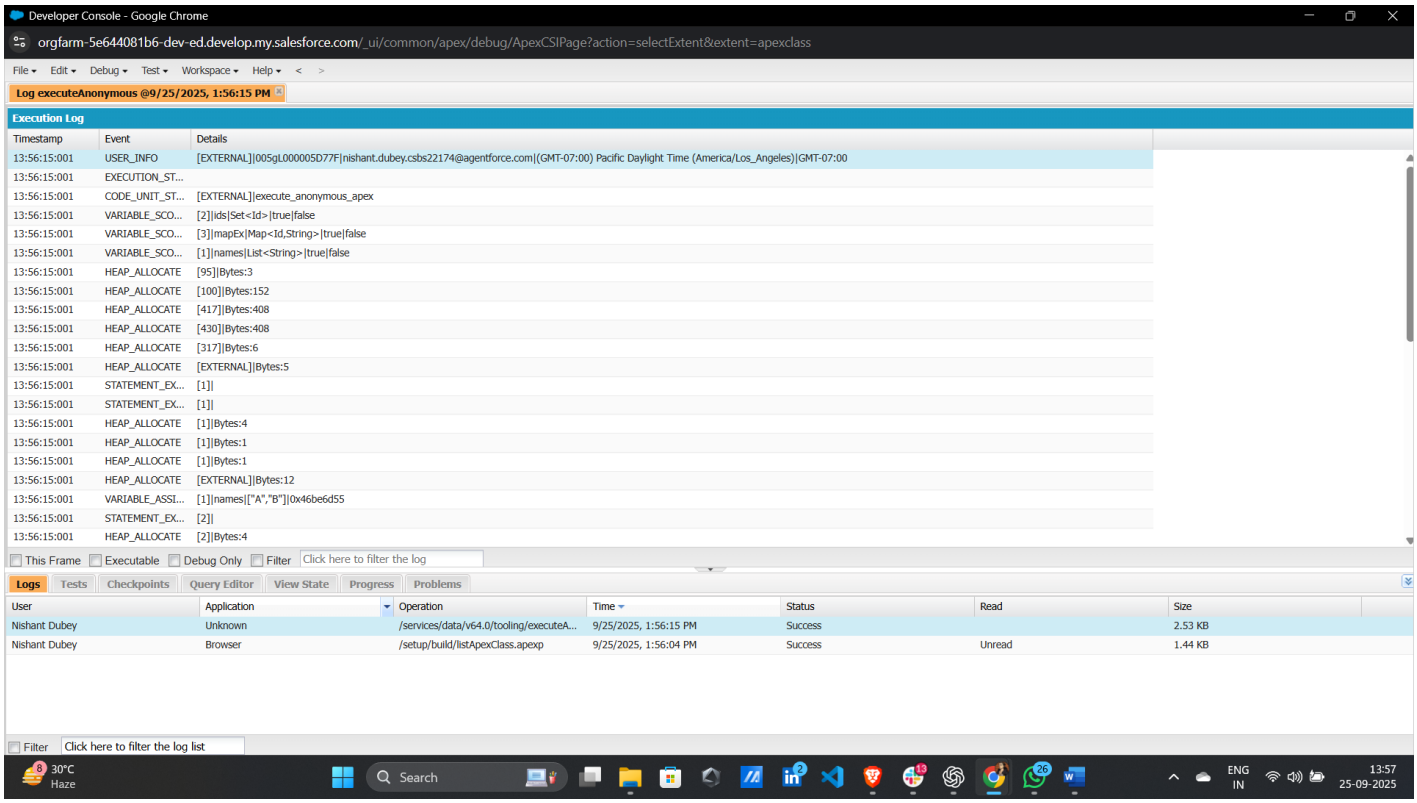
5.Collections: List, Set, Map

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>();
```



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().

Search Setup

Setup | Home | Object Manager

apex

- Email
  - Apex Exception Email
- Custom Code
  - Apex Classes
  - Apex Settings
  - Apex Test Execution
  - Apex Test History
  - Apex Triggers
- Environments
  - Jobs
    - Apex Flex Queue
    - Apex Jobs

Didn't find what you're looking for?
Try using Global Search.

SETUP
Apex Classes

Apex Class
RecalculateEnrollmentBatch

Help for this Page

Apex Class Detail Edit | Delete | Download | Security | Show Dependencies

Name	RecalculateEnrollmentBatch	Status	Active
Namespace Prefix		Code Coverage	0% (0/8)
Created By	Nishant Dubey , 9/25/2025, 12:34 AM	Last Modified By	Nishant Dubey , 9/25/2025, 12:34 AM

Class Body | Class Summary | Version Settings | Trace Flags

```
public class RecalculateEnrollmentBatch implements Database.Batchable<SObject> {
  public Database.QueryLocator start(Database.BatchableContext bc) {
    return Database.getQueryLocator('SELECT Id FROM Enrollment__c');
  }
  public void execute(Database.BatchableContext bc, List<Enrollment__c> scope) {
    Set<Id> ids = new Set<Id>();
    for (Enrollment__c e : scope) ids.add(e.Id);
    Map<Id, Decimal> averages = EnrollmentService.calculateAverageGrades(ids);
    EnrollmentService.updateEnrollmentsWithAverages(averages);
  }
  public void finish(Database.BatchableContext bc) {
    System.debug('Batch job completed');
  }
}
```

Edit | Delete | Download | Security | Show Dependencies

Developer Console - Google Chrome

orgfarm-5e644081b6-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage?action=selectExtent&extent=apexclass

File | Edit | Debug | Test | Workspace | Help

Log executeAnonymous @9/25/2025, 1:59:07 PM

Execution Log

Timestamp	Event	Details
13:59:07:001	USER_INFO	[EXTERNAL]|005gL000005D77F|nishant.dubey.csbs22174@agentforce.com|(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)|GMT-07:00
13:59:07:001	EXECUTION_ST...	
13:59:07:001	CODE_UNIT_ST...	[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_EX...	[1]|
13:59:07:002	STATEMENT_EX...	[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_EX...	[1]|
13:59:07:004	STATEMENT_EX...	[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

This Frame | Executable | Debug Only | Filter | Click here to filter the log

Logs | Tests | Checkpoints | Query Editor | View State | Progress | Problems

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

Filter | Click here to filter the log list

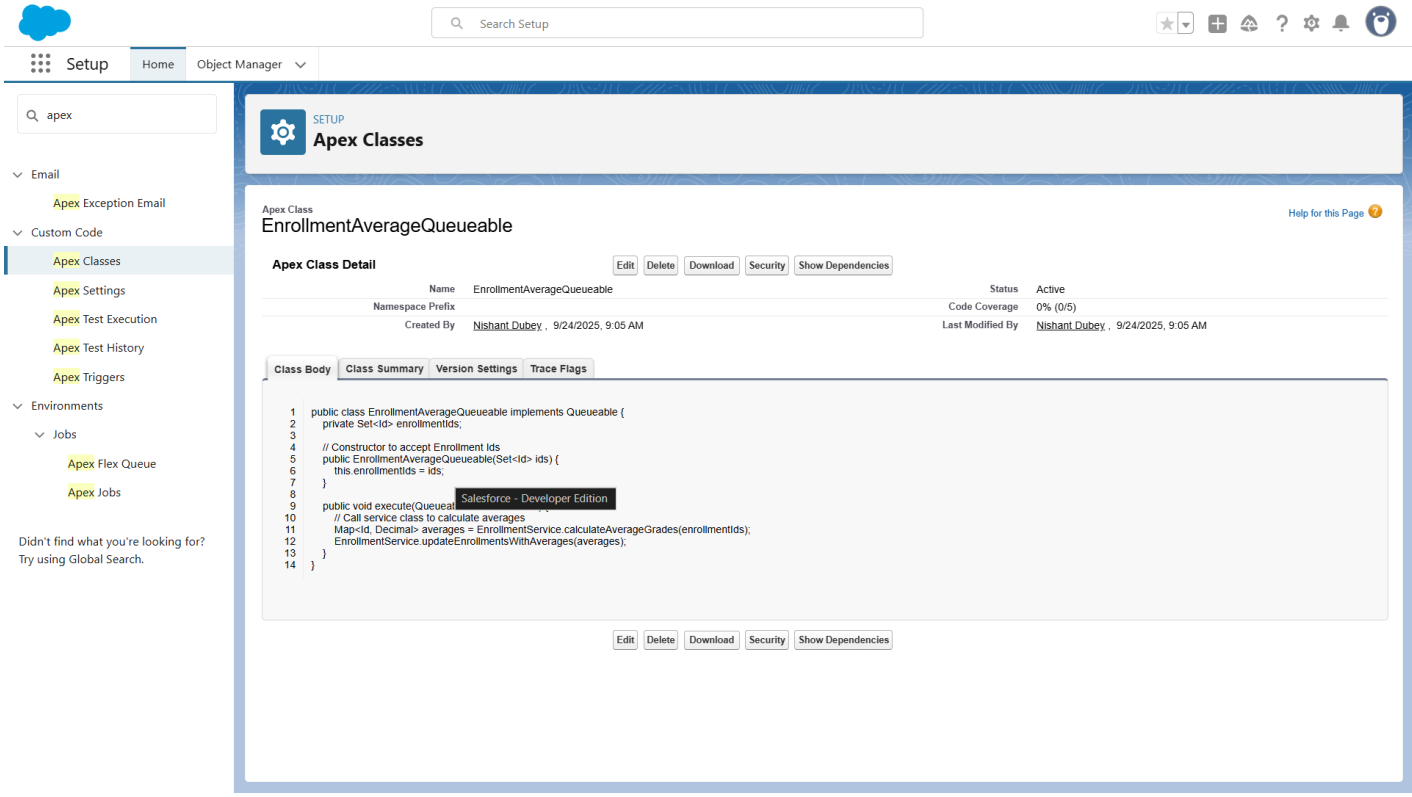
30°C Haze | Search | ENG IN | 13:59 25-09-2025

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().



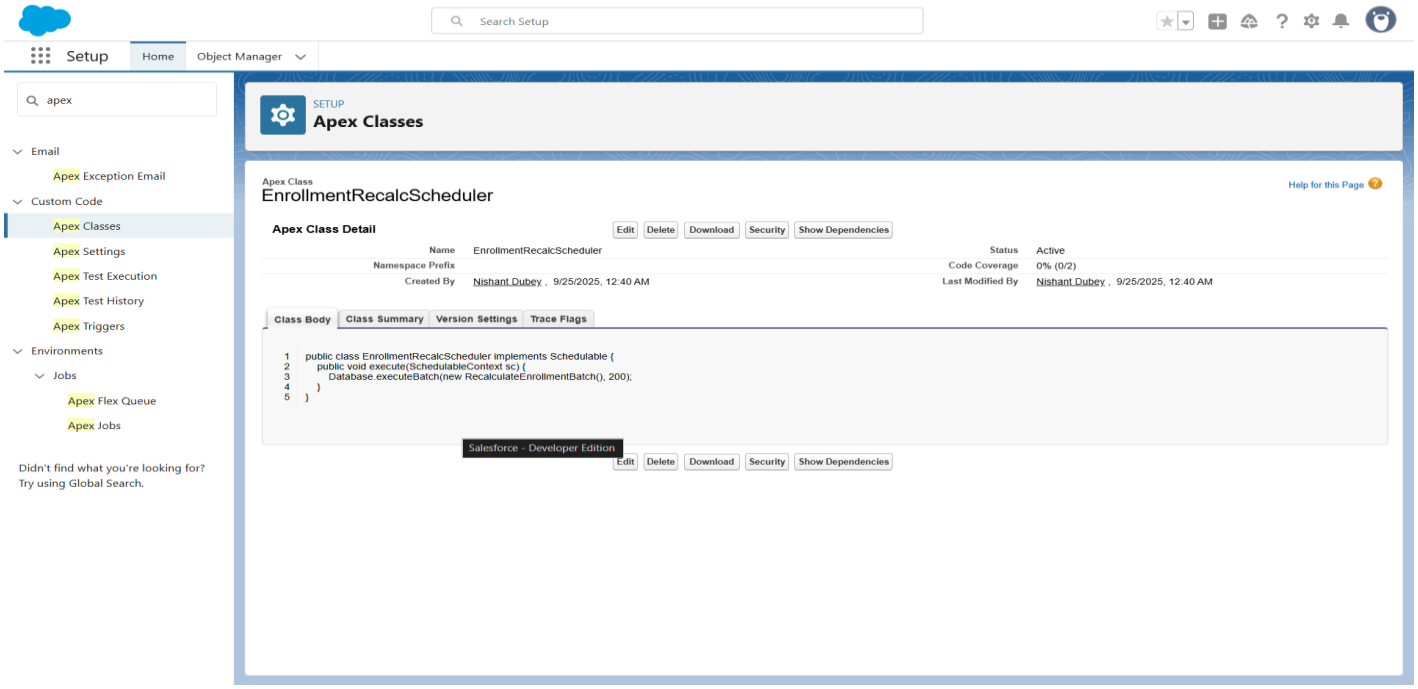
8. Scheduled Apex

Automated background jobs on a schedule.

Steps Implemented:

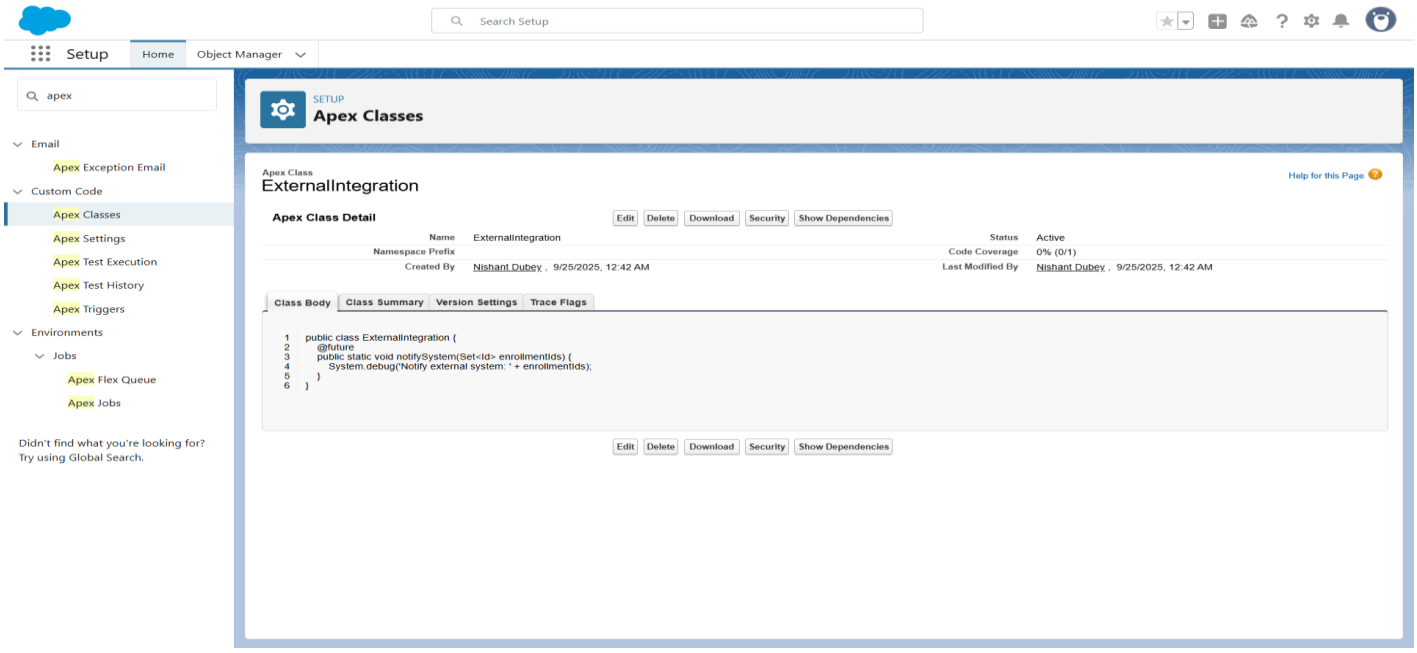
1. Created a scheduler class implementing Schedulable.

2. Setup → Apex Classes → Schedule Apex → Defined cron expression.



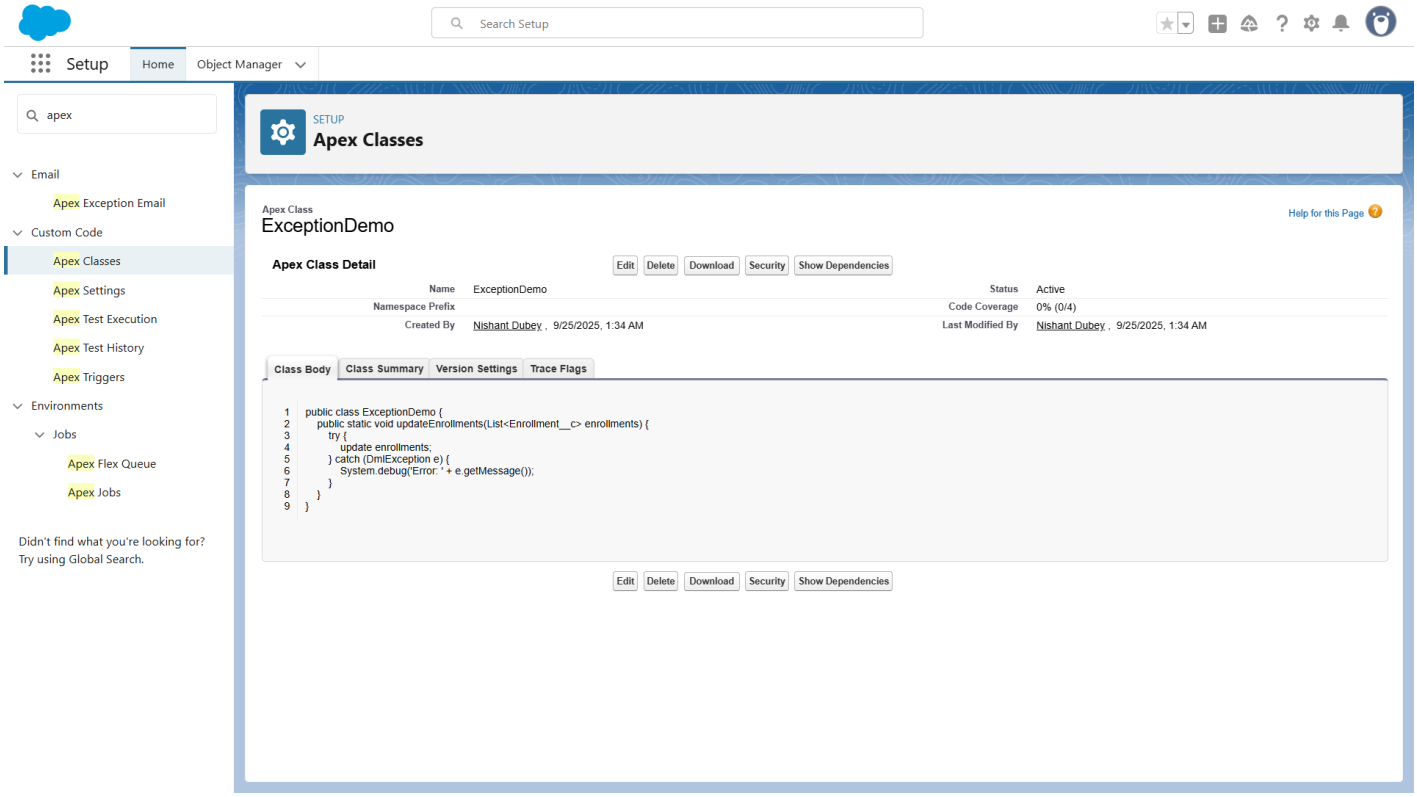
9. Future Methods

Used for lightweight async calls.



10. Exception Handling

Added try-catch-finally blocks for error handling.



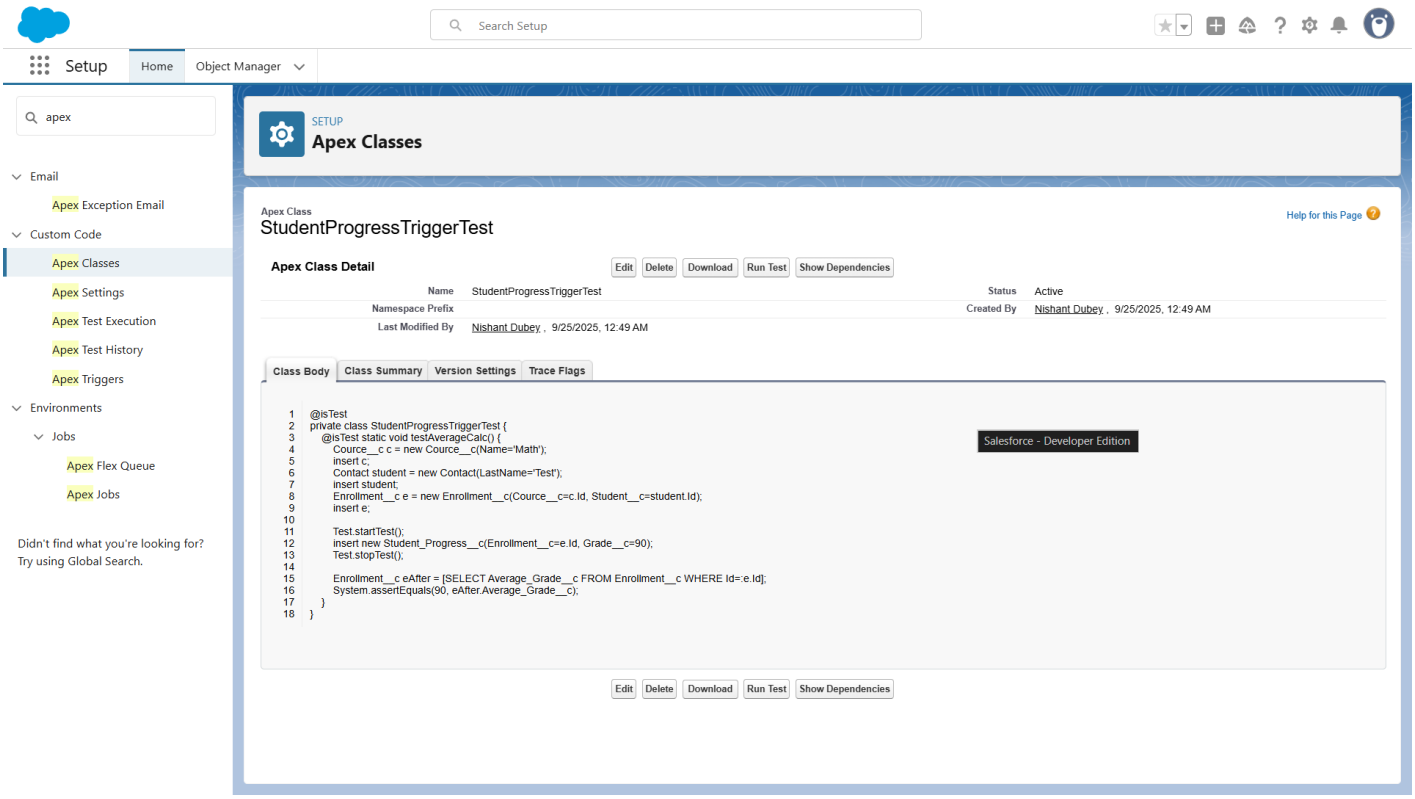
11. Test Classes

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.