

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

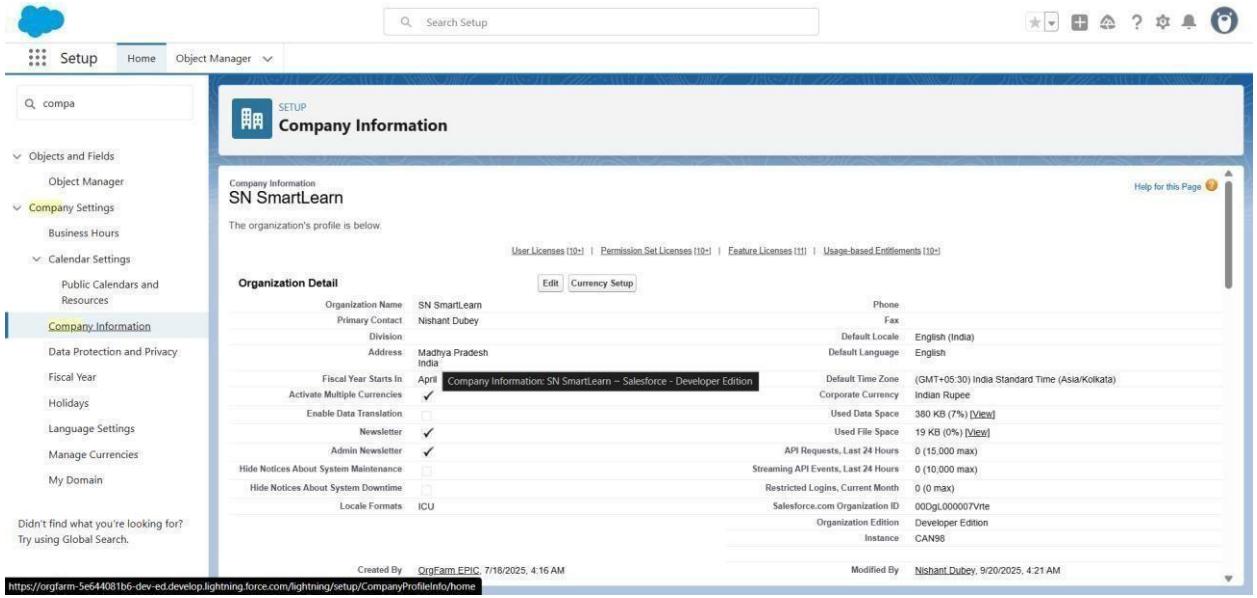
We used a **Salesforce Developer Edition Org** for this implementation. This edition was selected because it provides all the core CRM features required for our project, such as custom objects, roles, profiles, automation tools, and APIs. It also supports AppExchange integration, which we plan to explore 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 for 'Company Information'. The left sidebar has sections like Objects and Fields, Object Manager, Company Settings (selected), Business Hours, Calendar Settings, Public Calendars and Resources, Company Information (selected), Data Protection and Privacy, Fiscal Year, Holidays, Language Settings, Manage Currencies, and My Domain. The main content area is titled 'Company Information' and 'SN SmartLearn'. It shows organization details: Name (SN SmartLearn), Primary Contact (Nishant Dubey), Division (None), Address (Madhya Pradesh, India). Under 'Organization Detail', there are checkboxes for 'Fiscal Year Starts In April' (checked), 'Activate Multiple Currencies' (checked), 'Enable Data Translation' (unchecked), 'Newsletter' (checked), 'Admin Newsletter' (checked), 'Hide Notices About System Maintenance' (unchecked), 'Hide Notices About System Downtime' (unchecked), and 'Locale Formats' (ICU). On the right, there are sections for Phone, Fax, 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%)), Used File Space (19 KB (0%)), 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 (00DG1.000007VtE), Organization Edition (Developer Edition), and Instance (CAN98). At the bottom, it shows 'Created By' (OrgFarm EPIC) and 'Modified By' (Nishant Dubey).

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.

Business Hours Edit

Step 1. Business Hours Name

Business Hours Name: SN SmartLearn Hours

Active:

Step 2. Time Zone

Time Zone: (GMT+05:30) India Standard Time (Asia/Kolkata)

Step 3. Business Hours

Day	From	To	Notes
Sunday	9:00 AM	6:00 PM	<input type="checkbox"/> 24 hours
Monday	9:00 AM	6:00 PM	<input type="checkbox"/> 24 hours
Tuesday	9:00 AM	6:00 PM	<input type="checkbox"/> 24 hours
Wednesday	9:00 AM	6:00 PM	<input type="checkbox"/> 24 hours
Thursday	9:00 AM	6:00 PM	<input type="checkbox"/> 24 hours
Friday	9:00 AM	6:00 PM	<input type="checkbox"/> 24 hours
Saturday	9:00 AM	6:00 PM	<input type="checkbox"/> 24 hours

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.

Fiscal Year Information

Your organization can change the fiscal year start month, and specify whether the fiscal year name is set to the starting or ending year. For example, if your fiscal year starts in April 2025 and ends in March 2026, your Fiscal Year setting can be either 2025 or 2026.

Change Fiscal Year Period

Name: SN SmartLearn

Fiscal Year Start Month: April

Fiscal Year Is Based On:

- The ending month
- The starting month

Month	Period
January	2025-01-01 to 2025-03-31
February	2025-01-01 to 2025-03-31
March	2025-01-01 to 2025-03-31
April	2025-01-01 to 2025-03-31
May	2025-01-01 to 2025-03-31
June	2025-01-01 to 2025-03-31
July	2025-01-01 to 2025-03-31
August	2025-01-01 to 2025-03-31
September	2025-01-01 to 2025-03-31
October	2025-01-01 to 2025-03-31
November	2025-01-01 to 2025-03-31
December	2025-01-01 to 2025-03-31

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.

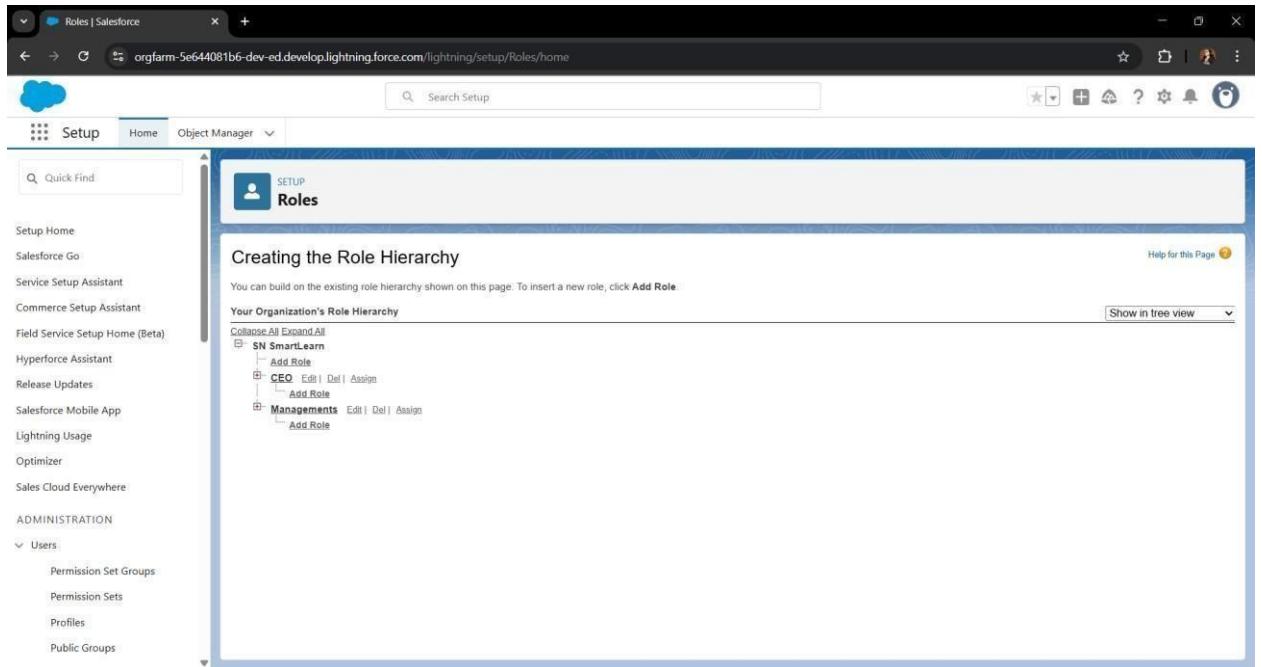
Profiles ensure each role has the exact level of access needed to perform their duties, reducing risks of unauthorized data exposure.

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.

The screenshot shows the Salesforce Setup interface with the 'Sharing Settings' page open. The left sidebar has 'Sharing Settings' selected under 'Security'. The main area contains six sections: 'Work Type Group Sharing Rules', 'Course Sharing Rules', 'Enrollment Sharing Rules', 'Mentor Sharing Rules', 'Student Sharing Rules', and 'Student Progress Sharing Rules'. Each section has 'New' and 'Recalculate' buttons. The 'Course Sharing Rules' section contains one rule: 'Edit | Del Course: Coarse Name EQUALS Intro to Salesforce' shared with 'All Internal Users' at 'Read Only' level. The 'Enrollment Sharing Rules' section contains one rule: 'Edit | Del Enrollment: Enrollment Number EQUALS Intro to Salesforce' shared with 'All Internal Users' at 'Read Only' level.

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.

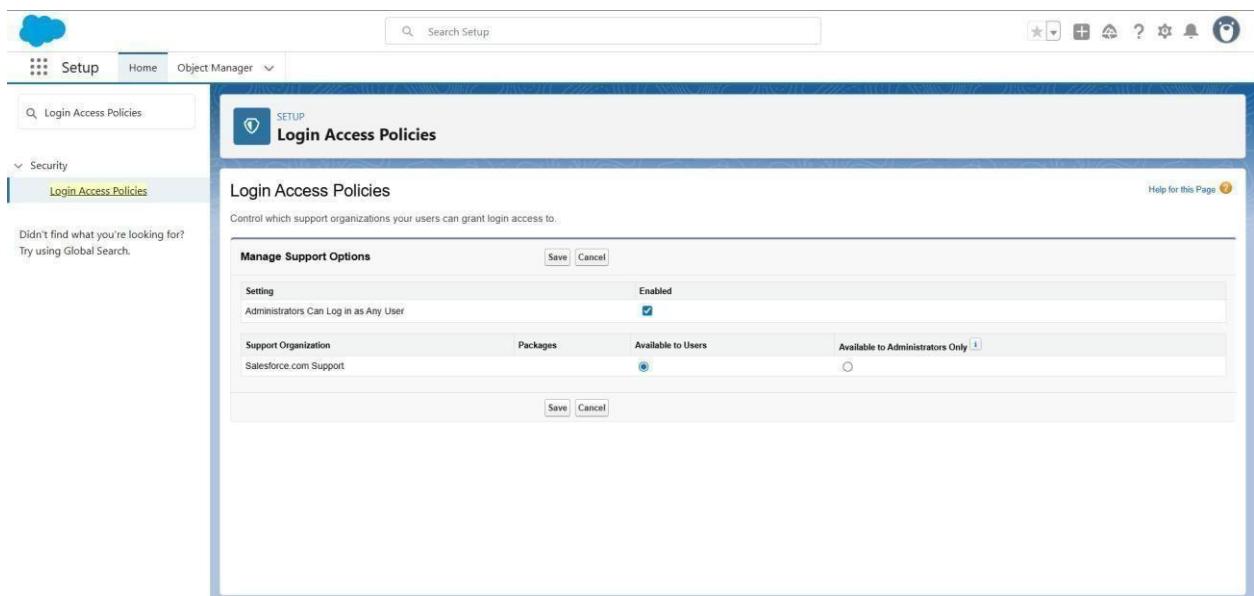
This prevents unnecessary exposure of sensitive data while enabling collaboration.

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

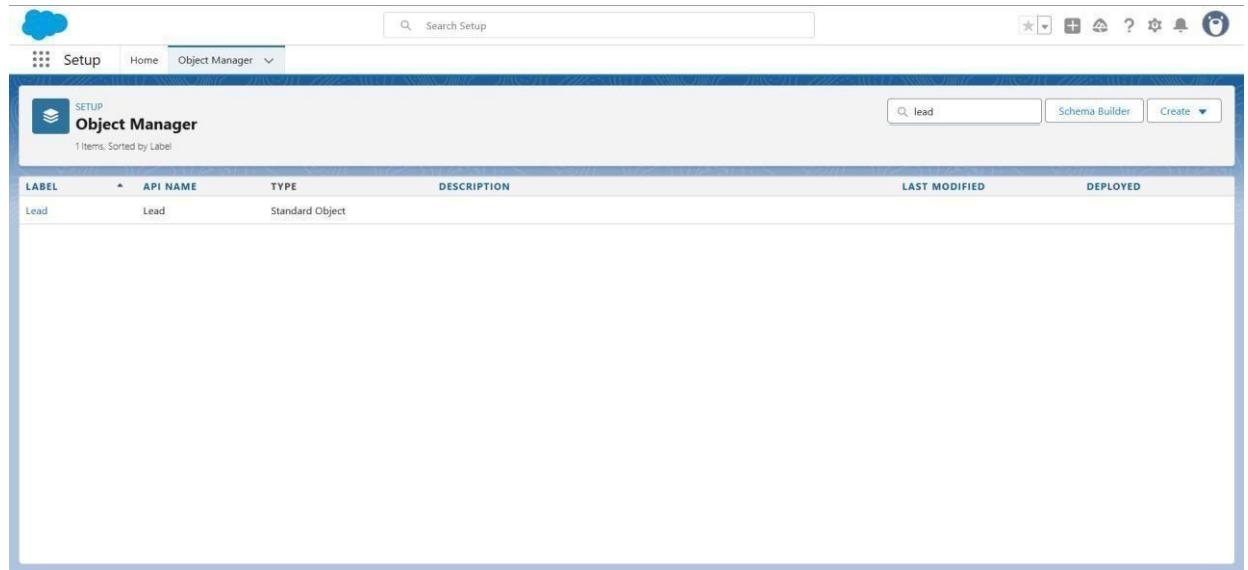
Phase 3 : Data Modeling & Relationships

1. Standard & Custom Objects

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

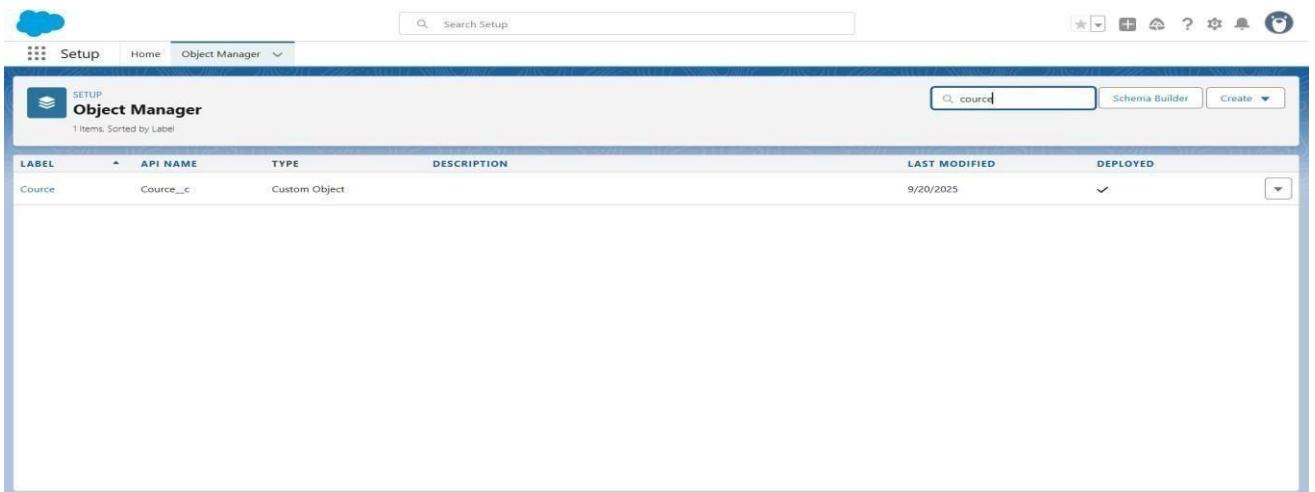
- **Standard Objects to Use:**

- **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:

- o **Course:** This is required to maintain a centralized inventory of all courses offered.
- o **Enrollment:** This custom object will be created upon Lead conversion to link a student to a specific course.
- o **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:
 - o **On the Course object:**
 - Course Code (Data Type: Text, **Unique**)
 - Instructor (Data Type: Lookup to **User**)
 - Status (Data Type: Picklist; Values: Active, Planned, Archived)
 - o **On the Enrollment object:**
 - Enrollment Date (Data Type: Date)
 - Status (Data Type: Picklist; Values: Applied, Enrolled, In Progress, Completed, Dropped)
 - o **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

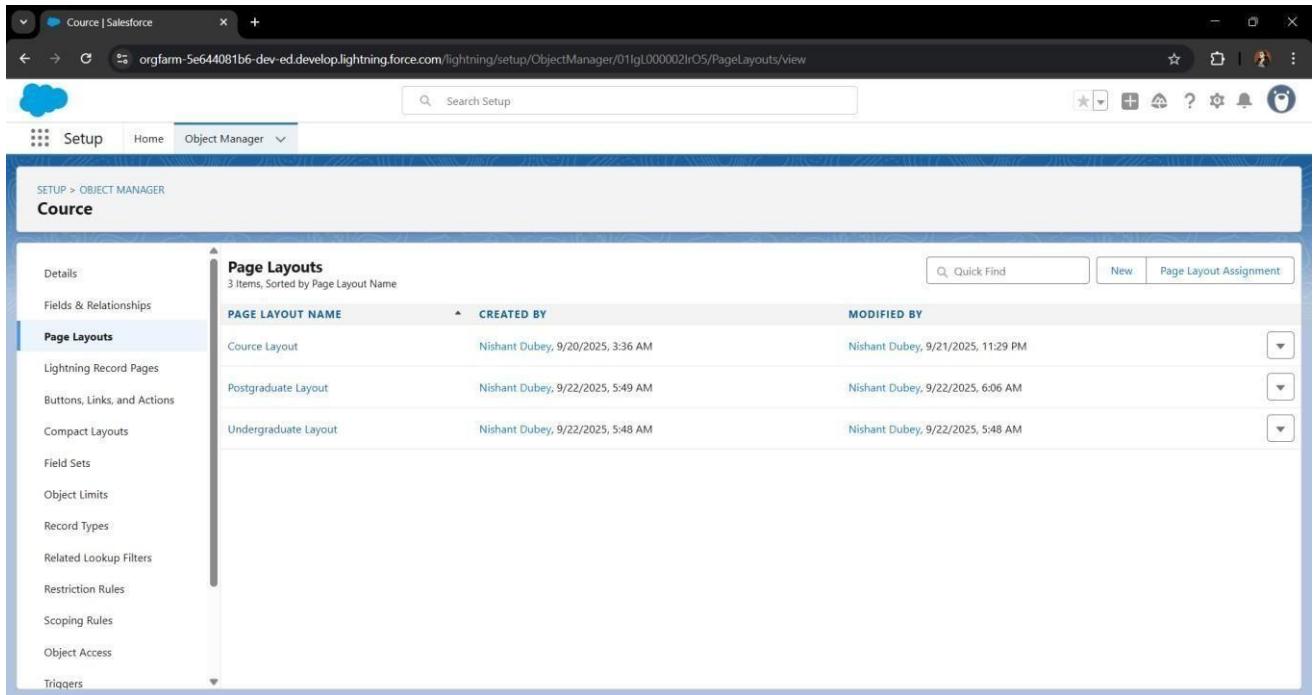
3. Record Types & Page Layouts

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 created 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 like Details, Fields & Relationships, Page Layouts, Lightning Record Pages, etc. The 'Page Layouts' section is currently selected. The main area displays a table titled 'Page Layouts' with three items: 'Course Layout', 'Postgraduate Layout', and 'Undergraduate Layout'. Each row includes columns for 'PAGE LAYOUT NAME', 'CREATED BY', and 'MODIFIED BY'. The 'Course Layout' was created by Nishant Dubey on 9/20/2025 at 3:36 AM and modified on 9/21/2025 at 11:29 PM. The 'Postgraduate Layout' was created by Nishant Dubey on 9/22/2025 at 5:49 AM and modified on 9/22/2025 at 6:06 AM. The 'Undergraduate Layout' was created by Nishant Dubey on 9/22/2025 at 5:48 AM and modified on 9/22/2025 at 5:48 AM. There are also 'Quick Find', 'New', and 'Page Layout Assignment' buttons at the top right of the table.

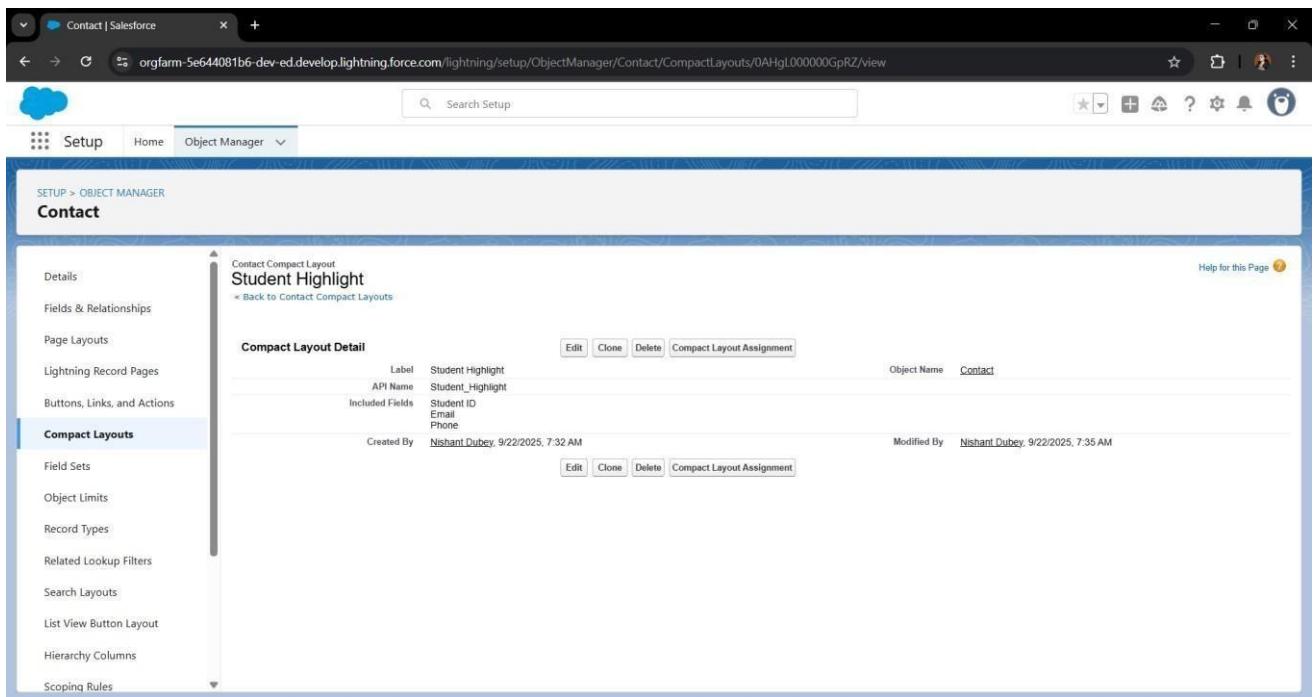
PAGE LAYOUT NAME	CREATED BY	MODIFIED BY
Course Layout	Nishant Dubey, 9/20/2025, 3:36 AM	Nishant Dubey, 9/21/2025, 11:29 PM
Postgraduate Layout	Nishant Dubey, 9/22/2025, 5:49 AM	Nishant Dubey, 9/22/2025, 6:06 AM
Undergraduate Layout	Nishant Dubey, 9/22/2025, 5:48 AM	Nishant Dubey, 9/22/2025, 5:48 AM

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.



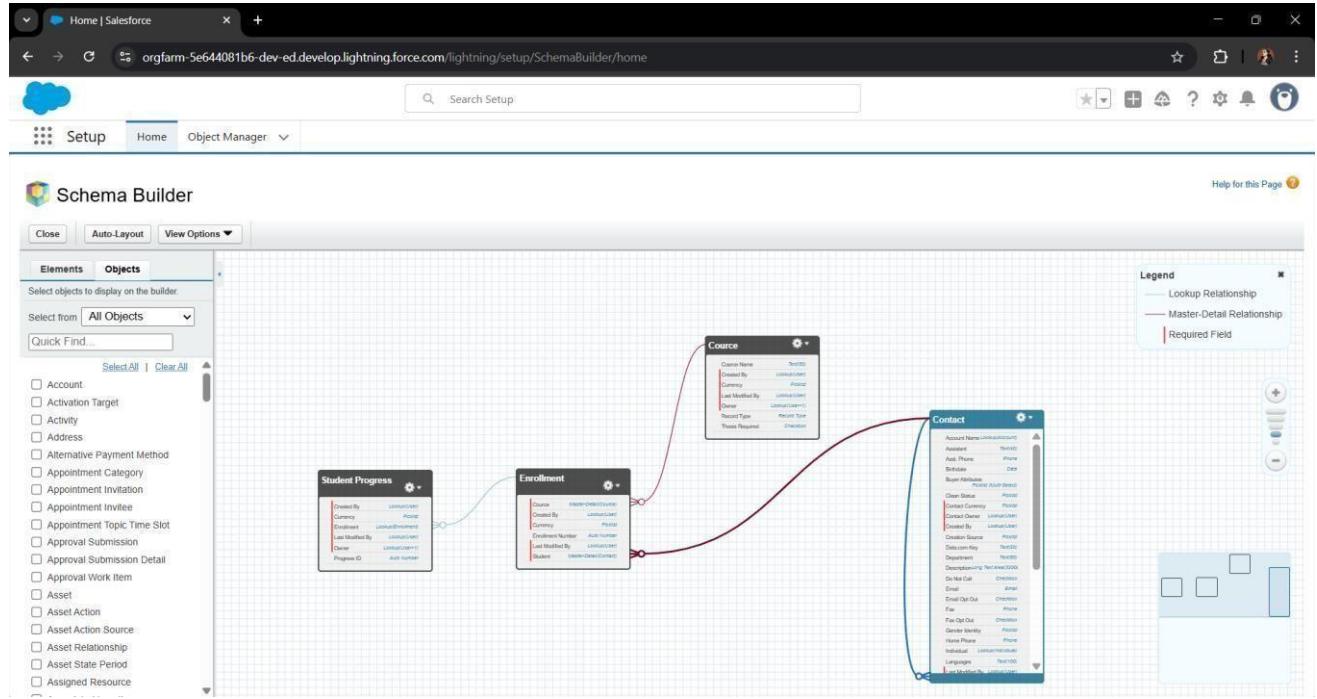
5. Schema Builder

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:
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.

The screenshot shows the Salesforce Setup interface for the 'Student Progress' object. The left sidebar lists various configuration options like Details, Fields & Relationships, Page Layouts, etc. The main content area displays the 'Validation Rule Detail' for a rule named 'Grade_Cannot_Exceed_100'. The rule's formula is 'Grade_c > 1', and its message is 'A grade cannot be more than 100%.' The rule is marked as Active. The right side of the screen shows standard Salesforce navigation and utility icons.

Validation Rule Detail	
Rule Name	Grade_Cannot_Exceed_100
Error Condition Formula	Grade_c > 1
Error Message	A grade cannot be more than 100%.
Description	
Created By	Nishant Dubey, 9/22/2025, 10:09 AM
Modified By	Nishant Dubey, 9/22/2025, 10:09 AM

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 following details:

- Workflow Rule Detail:**
 - Rule Name: Create Task for New Application
 - Active: ✓
 - Description: Follow up on new application
 - Rule Criteria: Lead: Created Date NOTEQUALTO null
 - Object: Lead
 - Evaluation Criteria: Evaluate the rule when a record is created
 - Created By: Nishant Dubey, 9/22/2025, 10:17 AM
 - Modified By: Nishant Dubey, 9/22/2025, 10:21 AM
- Workflow Actions:**
 - Immediate Workflow Actions:**
 - Type: Task
 - Description: Follow up on new application
 - Time-Dependent Workflow Actions:** See an example
- A warning message 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 View All Processes Clone View Properties Deactivate Read Only

Update Records

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 View All Processes Clone View Properties Deactivate Read Only

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 Approval Processes
- Left Sidebar:** Approval proc, Process Automation, Approval Processes (selected), Didn't find what you're looking for? Try using Global Search.
- Process Definition Detail:**
 - Process Name: Discount Approval
 - Unique Name: Discount_Approval
 - Description: Enrollment: Discount Percentage GREATER THAN 0.20
 - Entry Criteria: Enrollment: Discount Percentage GREATER THAN 0.20
 - Record Editability: Administrator ONLY
 - Active: checked
 - Next Automated Approver Determined By: Manager of Record Submitter
 - Allow Submitters to Recall Approval Requests: unchecked
 - 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: Record Lock
 - Description: 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: Record Lock
 - Description: Lock the record from being edited
- Final Rejection Actions:**
 - Add Existing | Add New

5. Flow Builder

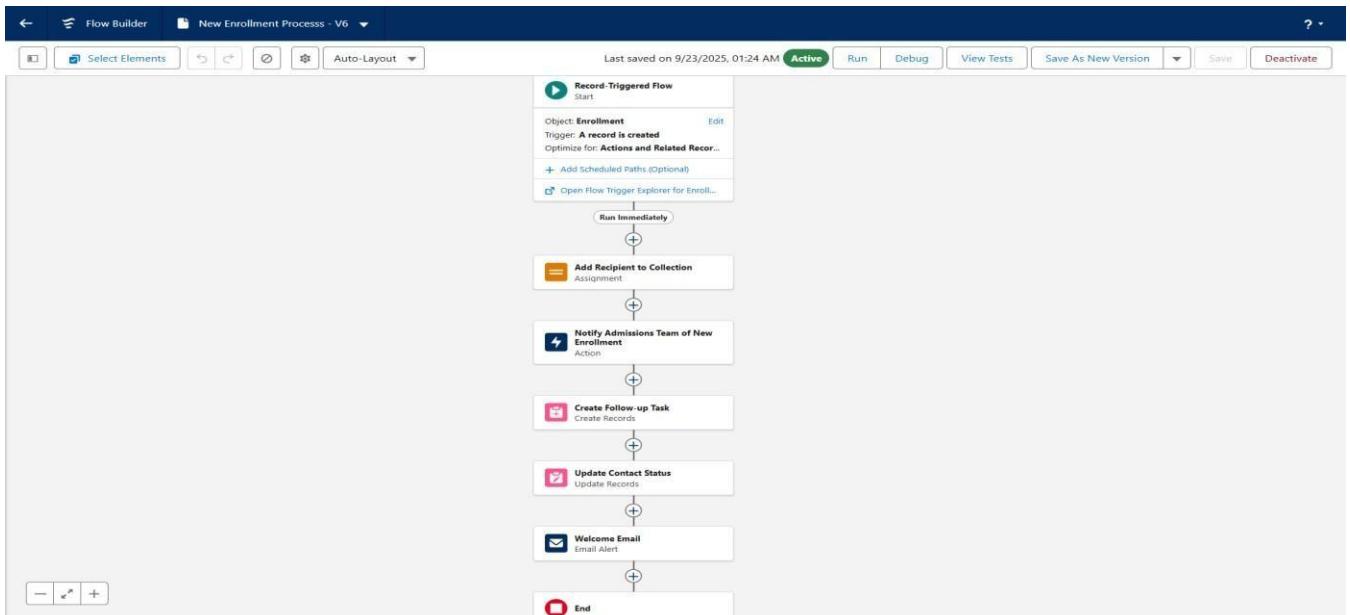
Flow Builder is the primary **automation tool** in Salesforce, replacing Workflow and Process Builder.

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



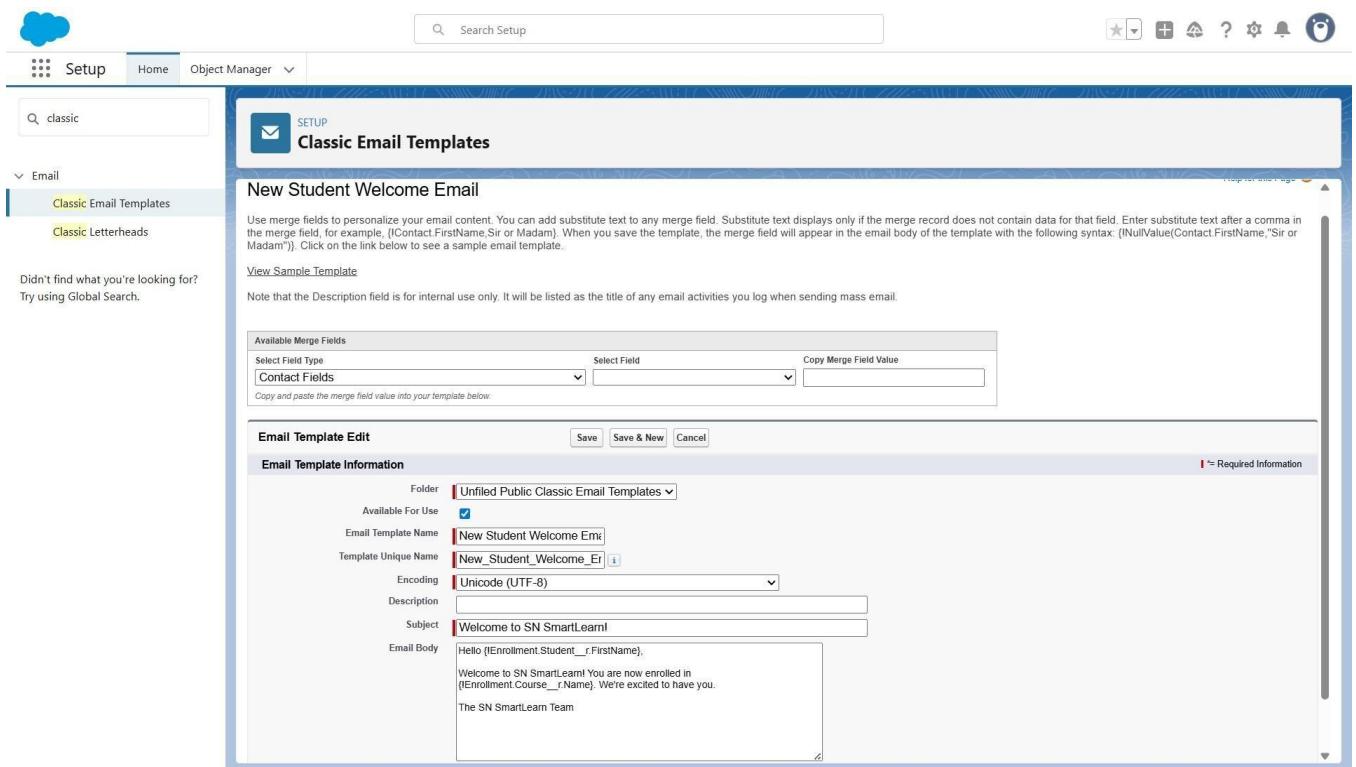
6. Email Alerts

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.



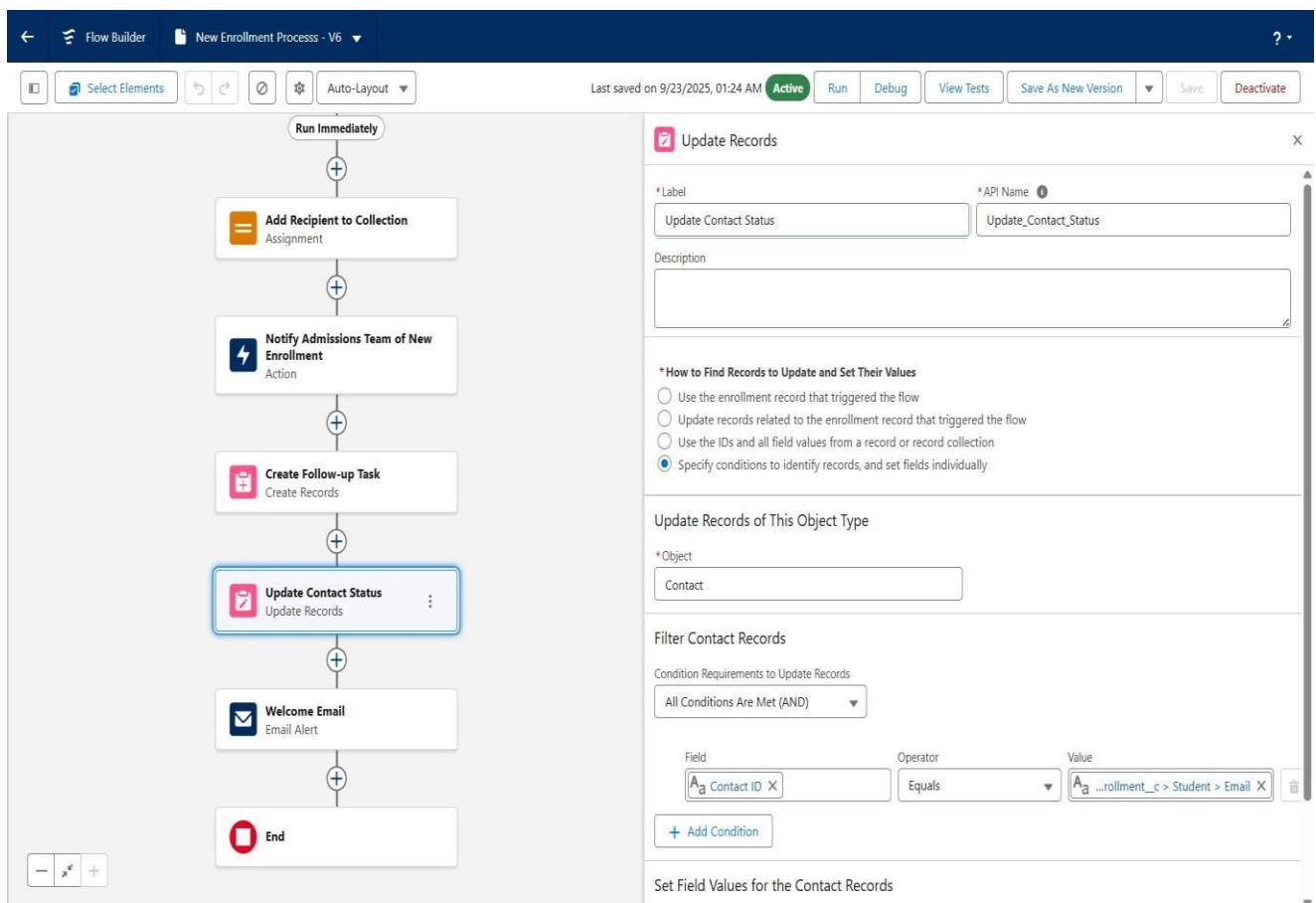
7. Field Updates

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.



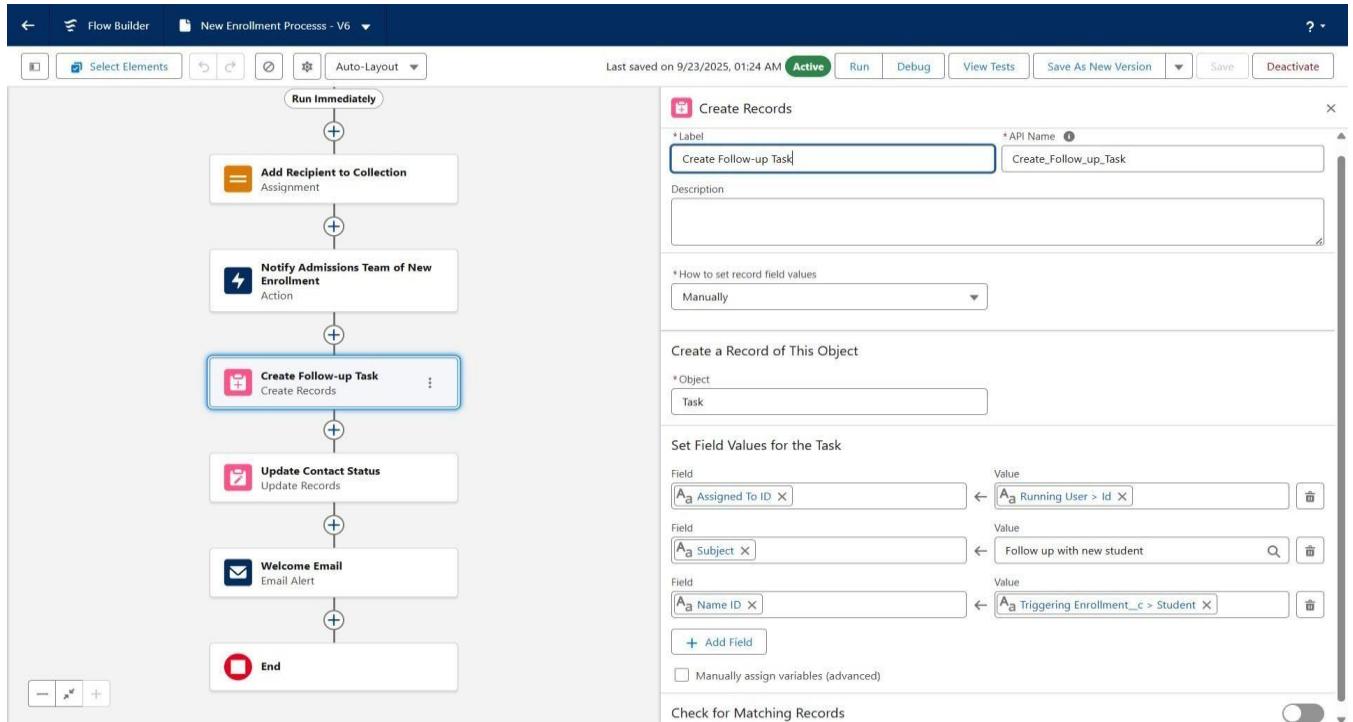
8. Tasks

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

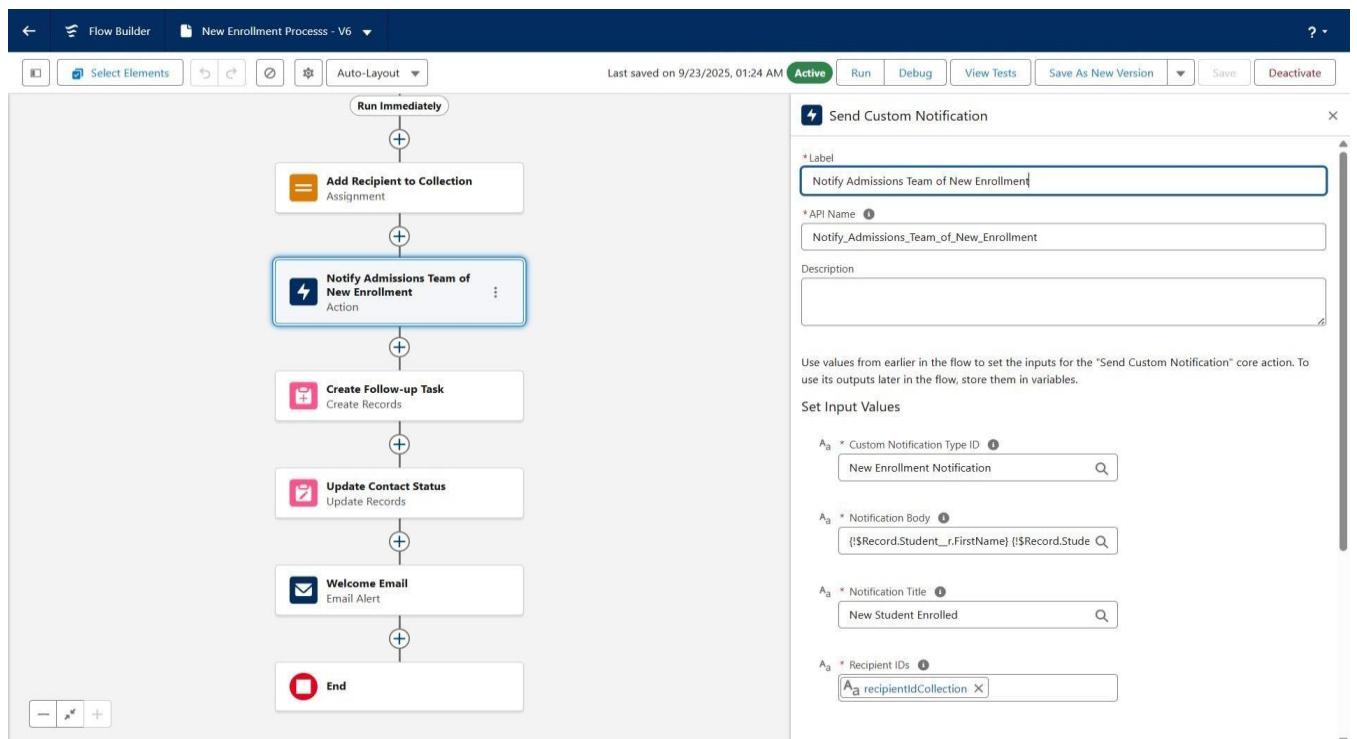
Custom Notifications send alerts to users in Salesforce (bell icon & mobile).

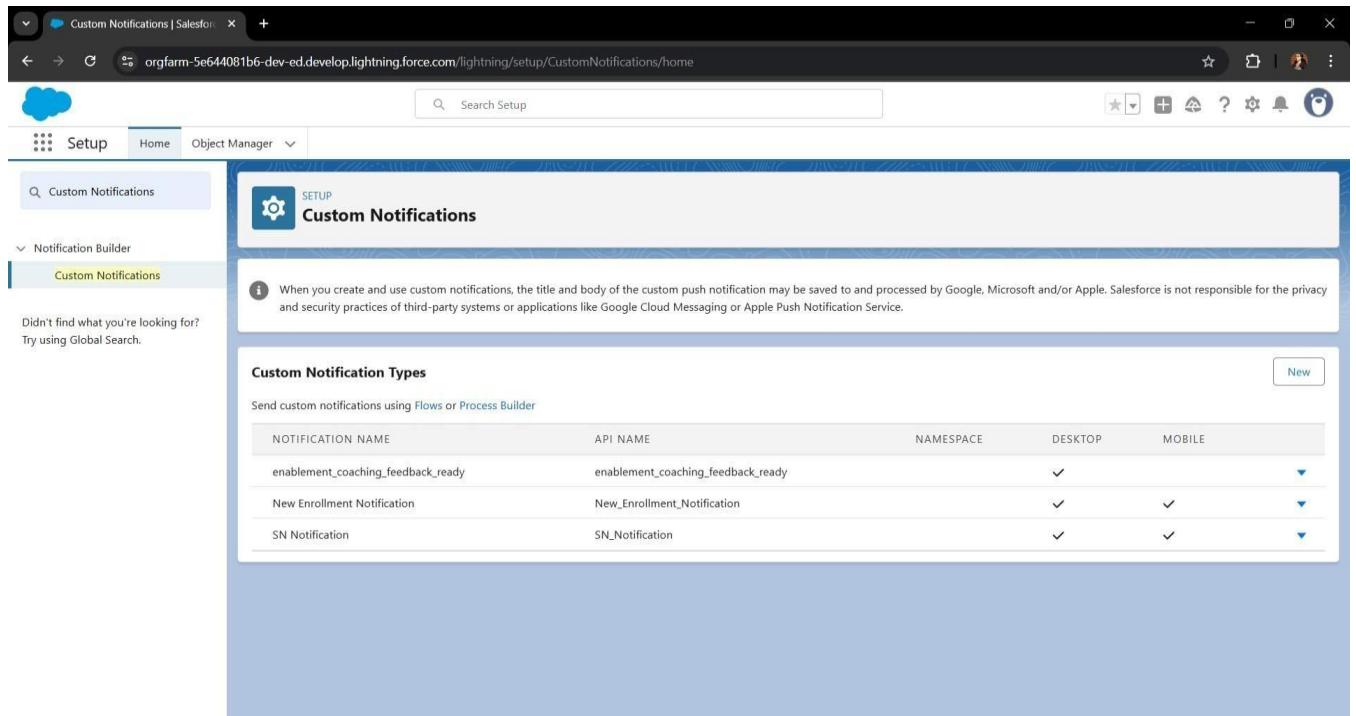
Use Case: Notify Admissions Team Lead when a new student enrolls.

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 left sidebar, under 'Custom Code', 'Apex Classes' is selected. The main content area displays the 'Apex Class Detail' for the 'EnrollmentService' class. The class name is 'EnrollmentService'. The code editor contains 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> agrs = [
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 : agrs) {
14            result.put((Id)ar.get('en'), (Decimal)ar.get('avgG'));
15        }
16        return result;
17    }
18
19    // Update enrollments with calculated averages
20    public static void updateEnrollmentsWithAverages(Map<Id, Decimal> averages) {
21        List<Enrollment__c> ups = new List<Enrollment__c>();
22        for (Id eld : averages.keySet()) {
23            ups.add(new Enrollment__c(id = eld, Average_Grade__c = averages.get(eld)));
24        }
25        if (ups.isEmpty()) update ups;
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 Setup interface with the following details:

- Header:** Search Setup, Home, Object Manager
- Breadcrumbs:** SETUP > OBJECT MANAGER > Student Progress
- Page Title:** Apex Trigger StudentProgressTrigger
- Apex Trigger Detail:**
 - Name: StudentProgressTrigger
 - Code Coverage: 0% (0/9)
 - Created By: Nishant Dubey, 9/24/2025, 9:03 AM
 - sObject Type: Student Progress
 - Status: Active
 - Last Modified By: Nishant Dubey, 9/24/2025, 9:06 AM
 - Namespace Prefix:
- Code Preview:**

```

1trigger 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}

```
- Sidebar (Triggers selected):**
 - 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** (selected)
 - Flow Triggers
 - Validation Rules
 - Conditional Field Formatting

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>();
```

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, 1:56:15 PM

Execution Log

Timestamp	Event	Details
13:56:15:001	USER_INFO	[EXTERNAL]005gL000005D77F(nishant.dubey.csbs22174@agentforce.com)(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)(GMT-07:00)
13:56:15:001	EXECUTION_ST...	[EXTERNAL]execute_anonymous_apex
13:56:15:001	CODE_UNIT_ST...	[EXTERNAL]execute_anonymous_apex
13:56:15:001	VARIABLE_SCO...	[2]ids[Set<Id>]true false
13:56:15:001	VARIABLE_SCO...	[3]mapEx[Map<Id, String>]true false
13:56:15:001	VARIABLE_SCO...	[1]names[List<String>]true false
13:56:15:001	HEAP_ALLOCATE	[95]Bytes:3
13:56:15:001	HEAP_ALLOCATE	[100]Bytes:152
13:56:15:001	HEAP_ALLOCATE	[417]Bytes:408
13:56:15:001	HEAP_ALLOCATE	[430]Bytes:408
13:56:15:001	HEAP_ALLOCATE	[317]Bytes:6
13:56:15:001	HEAP_ALLOCATE	[EXTERNAL]Bytes:5
13:56:15:001	STATEMENT_EX...	[1]
13:56:15:001	STATEMENT_EX...	[1]
13:56:15:001	HEAP_ALLOCATE	[1]Bytes:4
13:56:15:001	HEAP_ALLOCATE	[1]Bytes:1
13:56:15:001	HEAP_ALLOCATE	[1]Bytes:1
13:56:15:001	HEAP_ALLOCATE	[EXTERNAL]Bytes:12
13:56:15:001	VARIABLE_ASST...	[1]names[[A,"B"]] 0x46be6d55
13:56:15:001	STATEMENT_EX...	[2]
13:56:15:001	HEAP_ALLOCATE	[2]Bytes:4

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	/services/data/v64.0/tooling/executeA...	9/25/2025, 1:56:15 PM	Success		2.53 KB
Nishant Dubey	Browser	/setup/build/listApexClass.apexp	9/25/2025, 1:56:04 PM	Success	Unread	1.44 KB

Filter Click here to filter the log list

30°C Haze Search

ENG IN 13:57 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. On the left, there's a sidebar with navigation links like Home, Object Manager, and various Apex-related sections such as Apex Classes, Apex Settings, Apex Test Execution, Apex Test History, and Apex Triggers. The main content area is titled "Apex Classes" and shows the details for the class "RecalculateEnrollmentBatch". The class detail table includes fields for Name, Namespace Prefix, Created By, Status, and Active. Below the table is the class code in a code editor:

```

1 public class RecalculateEnrollmentBatch implements Database.Batchable<SObject> {
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 Developer Console in Google Chrome. At the top, it displays the URL "orgfarm-5e644081b6-dev-ed.develop.my.salesforce.com/ui/common/apex/debug/ApexCSIPage?action=selectExtent&extent=apexclass" and the timestamp "Log executeAnonymous @9/25/2025, 1:59:07 PM". The main area is the "Execution Log" which lists various system events and allocations during the execution of the apex class. Below the log is a table titled "Logs" showing a list of log entries with columns for User, Application, Operation, Time, Status, Read, and Size.

User	Application	Operation	Time	Status	Read	Size
Nishant Dubey	Unknown	Batch Apex	9/25/2025, 1:59:07 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 interface with the search bar set to "apex". The main pane displays the "Apex Classes" section under "Custom Code". A specific class, "EnrollmentAverageQueueable", is selected and shown in detail. The "Apex Class Detail" section includes fields for Name (EnrollmentAverageQueueable), Namespace Prefix, Created By (Nishant Dubey), and Last Modified By (Nishant Dubey). The "Status" is Active and "Code Coverage" is 0%. The "Class Body" tab is selected, showing 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 }
```

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 search bar set to "apex". The left sidebar has sections for Email, Custom Code, Apex Classes, Environments, and Jobs. The "Apex Classes" section is selected and expanded, showing sub-options like Apex Settings, Apex Test Execution, Apex Test History, and Apex Triggers. The main content area displays the "Apex Class Detail" for the "EnrollmentRecalcScheduler" class. The class name is "EnrollmentRecalcScheduler", and it implements the "Schedulable" interface. The code body contains the following Apex code:

```

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

```

The class is created by "Nishant Dubey" on 9/25/2025, 12:40 AM. It has 0% code coverage and was last modified by "Nishant Dubey" on 9/25/2025, 12:40 AM. The status is Active.

9. Future Methods

Used for lightweight async calls.

The screenshot shows the Salesforce Setup interface with the 'Apex Classes' section selected. The 'ExternalIntegration' class is displayed, which contains the following 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 'Apex Classes' page selected. The search bar at the top contains 'apex'. The left sidebar has sections for Email, Custom Code (with 'Apex Classes' highlighted), Environments, and Jobs. The main content area displays the 'ExceptionDemo' class details. The class name is 'ExceptionDemo' with a namespace prefix. It was created by 'Nishant.Dubey' on 9/25/2025, 1:34 AM. The status is Active and code coverage is 0% (0/4). The last modified by was also 'Nishant.Dubey' on the same date and time. Below the details, the 'Class Body' tab is selected, showing 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:

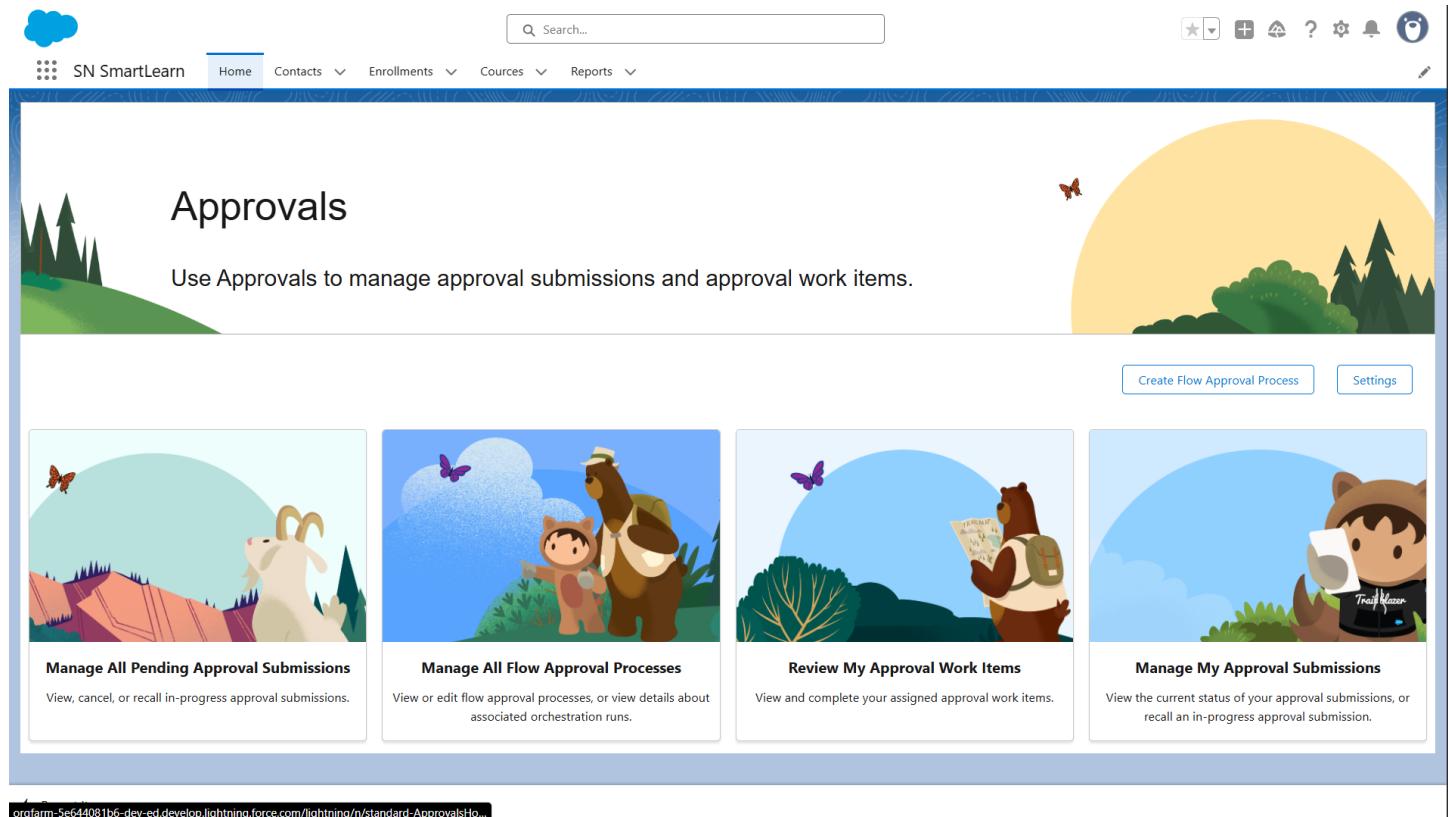
- Used Queueable for enrollment recalculations.
- Used Batch Apex for large dataset processing.
- Used Future for external calls.
- Used Scheduled Apex for nightly runs.

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.

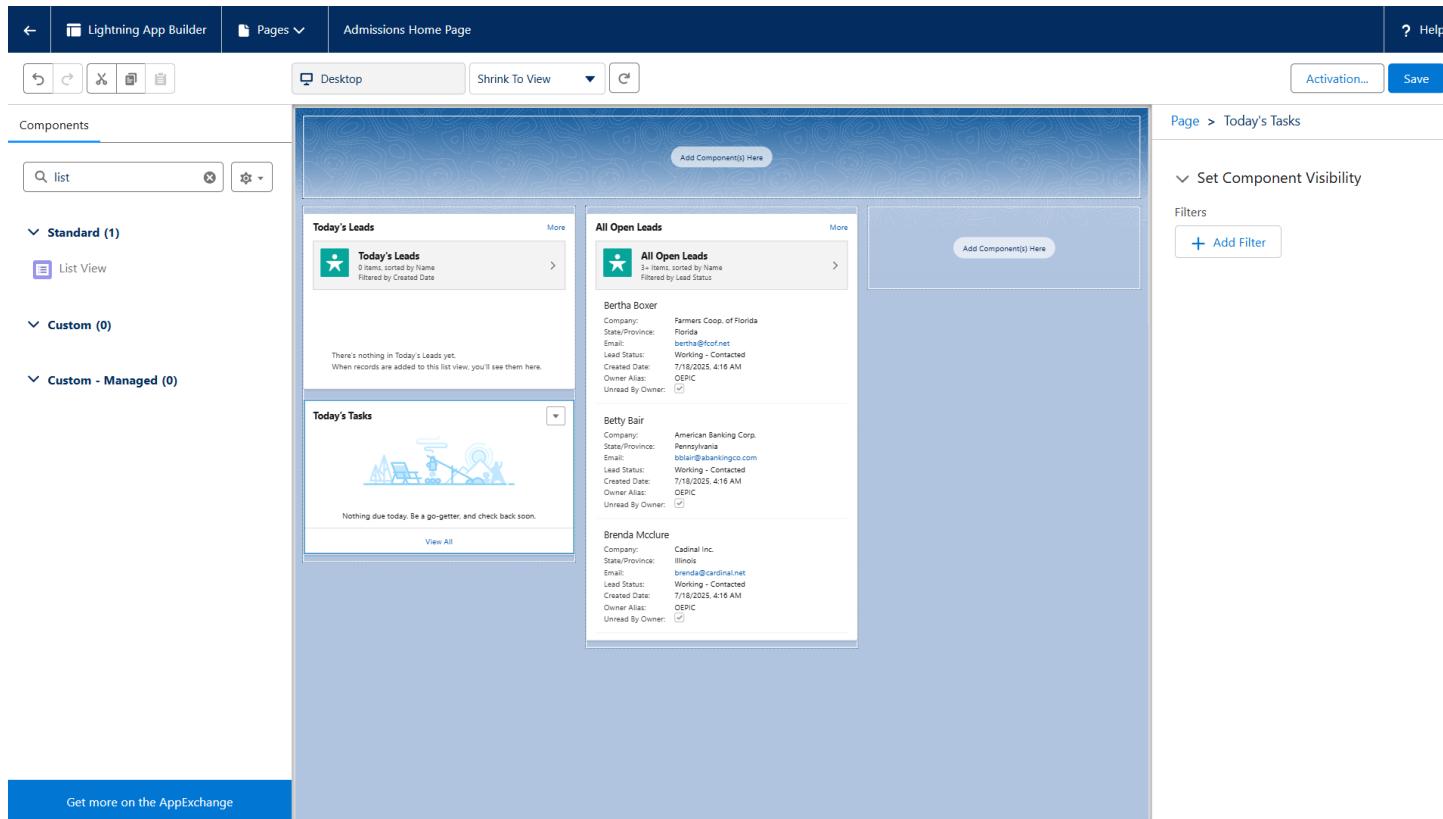
Use: This creates the "360-degree view" of the student, allowing users to see a student's details and all their related enrollments and progress in one clean, organized screen.

The screenshot shows the Lightning App Builder interface for creating a 'Contact Record Page'. The top navigation bar includes 'Lightning App Builder', 'Pages', 'Contact Record Page', 'Help', 'Analyze', 'Activation...', and 'Save' buttons. The left sidebar has tabs for 'Components' (selected) and 'Fields', with a search bar and filter icon. Below is a list of 'Standard (50)' components: Accordion, Action Launcher, Actions & Recommendations, Activities, Approval Trace, Assessment List, Chatter, Chatter Feed, Chatter Publisher, CRM Analytics Collection, CRM Analytics Dashboard, Data Cloud Profile Engagements, Data Cloud Profile Insights, Data Cloud Profile Related Records, Dynamic Related List - Single, Einstein Next Best Action, Flow, Flow Orchestration Work Guide, Highlights Panel, and Invoice Preview. A blue banner at the bottom says 'Get more on the AppExchange'. The main workspace displays two tabs: 'Related' and 'Details'. Under 'Related', sections include Opportunities (0), Cases (0), Campaign History (0), Notes & Attachments (0), and Enrollments (0). Under 'Details', it says 'We found no potential duplicates of this Contact.' and lists Opportunities (0), Cases (0), and Campaign History (0). To the right, a sidebar titled 'Page' contains fields for Label ('Contact Record Page'), API Name ('Contact_Record_Page'), Page Type ('Record Page'), Object ('Contact'), Template ('Header and Right Sidebar'), and Description. Buttons for 'Change' and 'Save' are at the bottom right.

Home Page Layouts

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

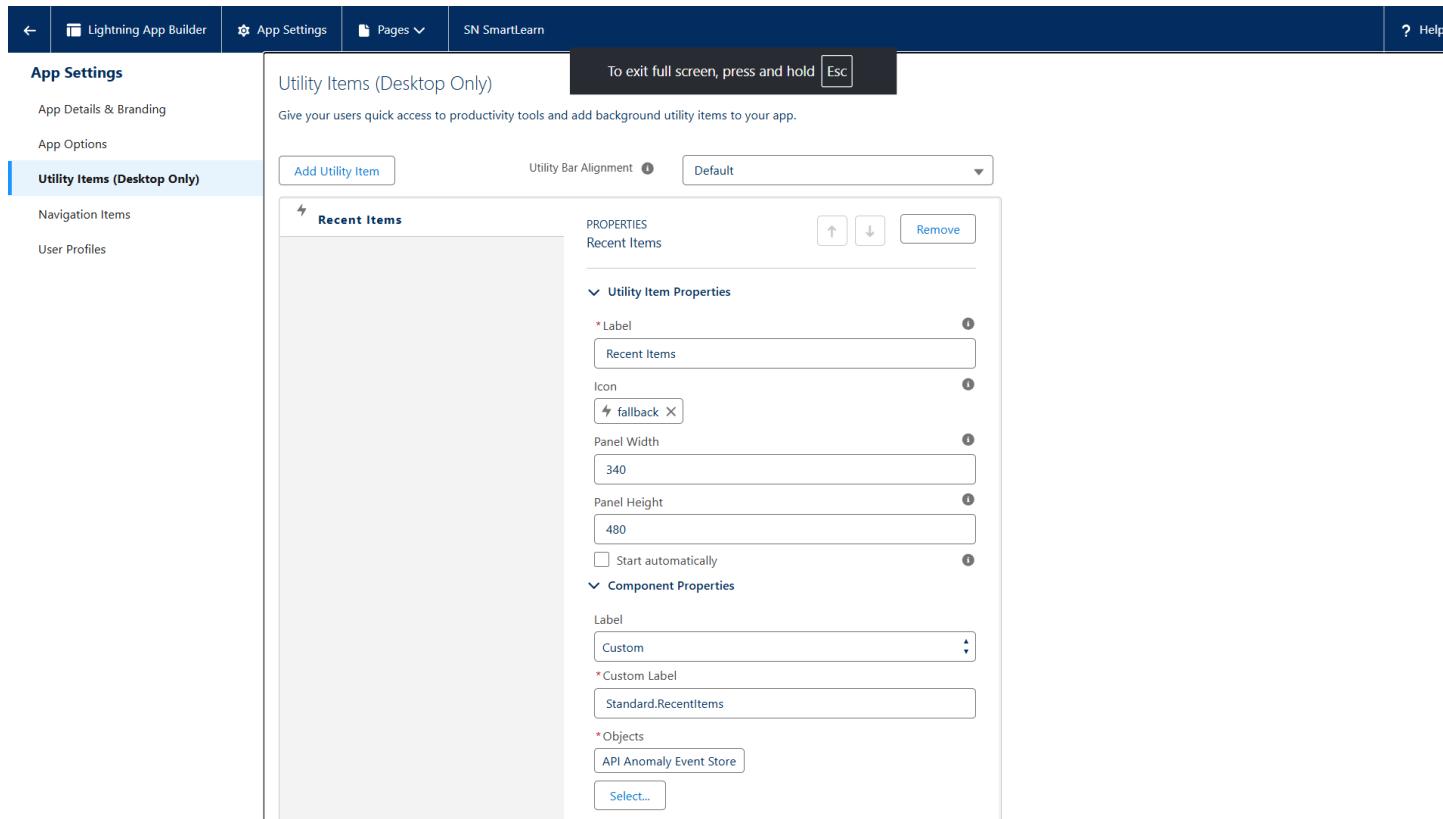
Use: This provides a dashboard-style landing page for the Admissions Team, showing them the most relevant information for their day, such as new applications (Leads) and their assigned tasks.



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.