Project: EduLink Student & Course Management System

Phase 5: Apex Programming (Developer)

Goal of this Phase

The goal of this phase was to implement core backend logic using Apex to support the EduLink system's student and course management functionalities. This involved creating classes with business logic, automating processes with triggers, handling data operations safely, and ensuring code reliability with test classes.

Of course. Based on our entire conversation, the Apex classes you've created, and the reference document you provided, here is a summary document for your Phase 5.

This document accurately reflects the work you have completed so far.

Project: EduLink Student & Course Management System

Phase 5: Apex Programming (Developer)

Goal of this Phase

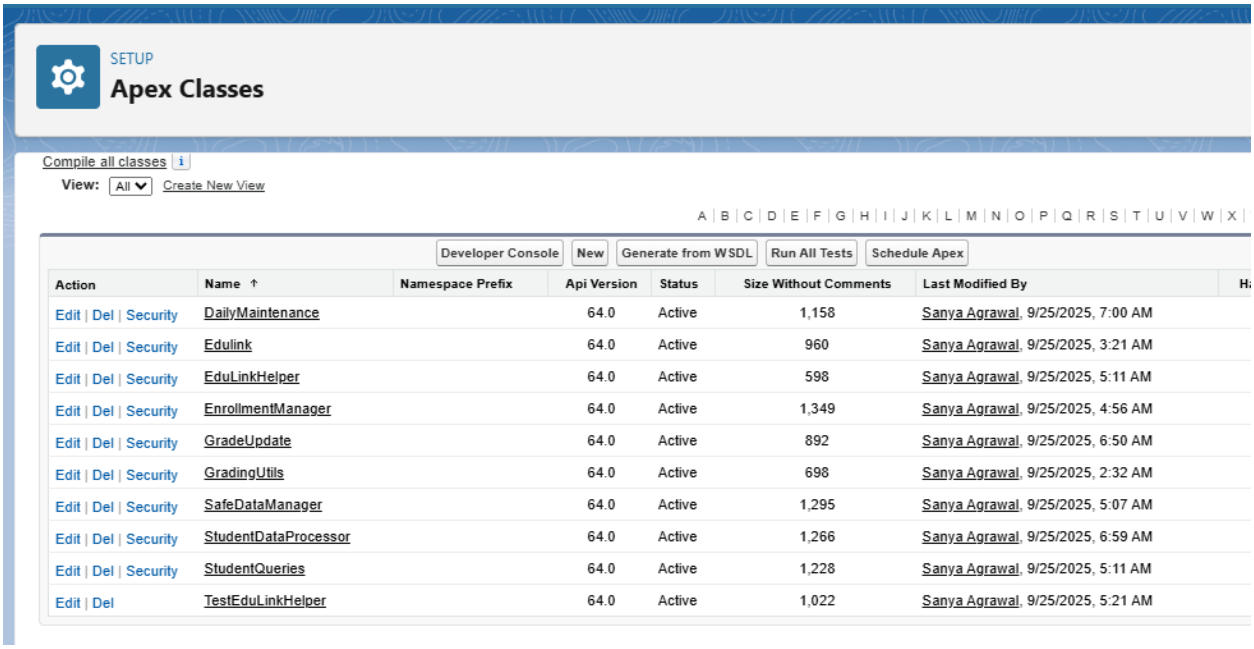
The goal of this phase was to implement core backend logic using Apex to support the EduLink system's student and course management functionalities. This involved creating classes with business logic, automating processes with triggers, handling data operations safely, and ensuring code reliability with test classes.

1. Classes & Objects

Several Apex classes were created to encapsulate business logic and manage data operations. Key classes include:

- **`StudentQueries`**: A utility class to centralize common SOQL queries for retrieving student and course data.

- **`EnrollmentManager`**: A service class to handle the logic for enrolling students in courses in bulk.
- **`SafeDataManager`**: A class designed to perform DML operations (inserts, updates) safely using exception handling.
- **`EduLinkHelper`**: A simple utility class with helper methods for calculations and grading logic.



Action	Name	Namespace Prefix	Api Version	Status	Size Without Comments	Last Modified By
Edit | Del | Security	DailyMaintenance		64.0	Active	1,158	Sanya Agrawal, 9/25/2025, 7:00 AM
Edit | Del | Security	Edulink		64.0	Active	960	Sanya Agrawal, 9/25/2025, 3:21 AM
Edit | Del | Security	EduLinkHelper		64.0	Active	598	Sanya Agrawal, 9/25/2025, 5:11 AM
Edit | Del | Security	EnrollmentManager		64.0	Active	1,349	Sanya Agrawal, 9/25/2025, 4:56 AM
Edit | Del | Security	GradeUpdate		64.0	Active	892	Sanya Agrawal, 9/25/2025, 6:50 AM
Edit | Del | Security	GradingUtils		64.0	Active	698	Sanya Agrawal, 9/25/2025, 2:32 AM
Edit | Del | Security	SafeDataManager		64.0	Active	1,295	Sanya Agrawal, 9/25/2025, 5:07 AM
Edit | Del | Security	StudentDataProcessor		64.0	Active	1,266	Sanya Agrawal, 9/25/2025, 6:59 AM
Edit | Del | Security	StudentQueries		64.0	Active	1,228	Sanya Agrawal, 9/25/2025, 5:11 AM
Edit | Del	TestEduLinkHelper		64.0	Active	1,022	Sanya Agrawal, 9/25/2025, 5:21 AM

Scenario 1: Testing the `calculateGPA` Method

For this an apex class named GradingUtils was created

Apex Class

GradingUtils

Apex Class Detail

Edit | Delete | Download | Security | Show Dependencies

Name	GradingUtils	Status	Active
Namespace Prefix		Code Coverage	0% (0/16)
Created By	Sanya Agrawal , 9/25/2025, 2:05 AM	Last Modified By	Sanya Agrawal , 9/25/2025, 2:32 AM

Class Body | Class Summary | Version Settings | Trace Flags

```
1  public class GradingUtils {
2      // Calculate simple average grade
3      public static Decimal calculateAverage(Decimal grade1, Decimal grade2, Decimal grade3) {
4          Decimal total = grade1 + grade2 + grade3;
5          Decimal average = total / 3;
6          return average;
7      }
8
9      // Convert percentage to letter grade
10     public static String getLetterGrade(Decimal percentage) {
11         if (percentage >= 90) {
12             return 'A';
13         } else if (percentage >= 80) {
14             return 'B';
15         } else if (percentage >= 70) {
16             return 'C';
17         } else if (percentage >= 60) {
18             return 'D';
19         } else {
20             return 'F';
21         }
22     }
23
24     // Check if student passed (simple version)
25     public static Boolean isStudentPassing(Decimal grade) {
26         return grade >= 60;
27     }
28 }
```

// 1. Find progress records for a specific student enrollment.

```
List<Progress__c> progressList = [SELECT Id, Points_Earned__c, Assignment__r.Max_Points__c
                    FROM Progress__c
                    WHERE Enrollment__r.Student__r.Name = Arjun Patel'
                    LIMIT 10];
if (!progressList.isEmpty()) {
   Decimal studentGPA = Edulink.calculateGPA(progressList);
   System.debug('The calculated GPA is: ' + studentGPA);
} else {
   System.debug('No progress records found for that student.');
}
```

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

Edulink.apxc | Log executeAnonymous @9/25/2025, 3:46:17 PM | StudentTrigger.apxt | StudentQueries.apxc | Log executeAn

Execution Log

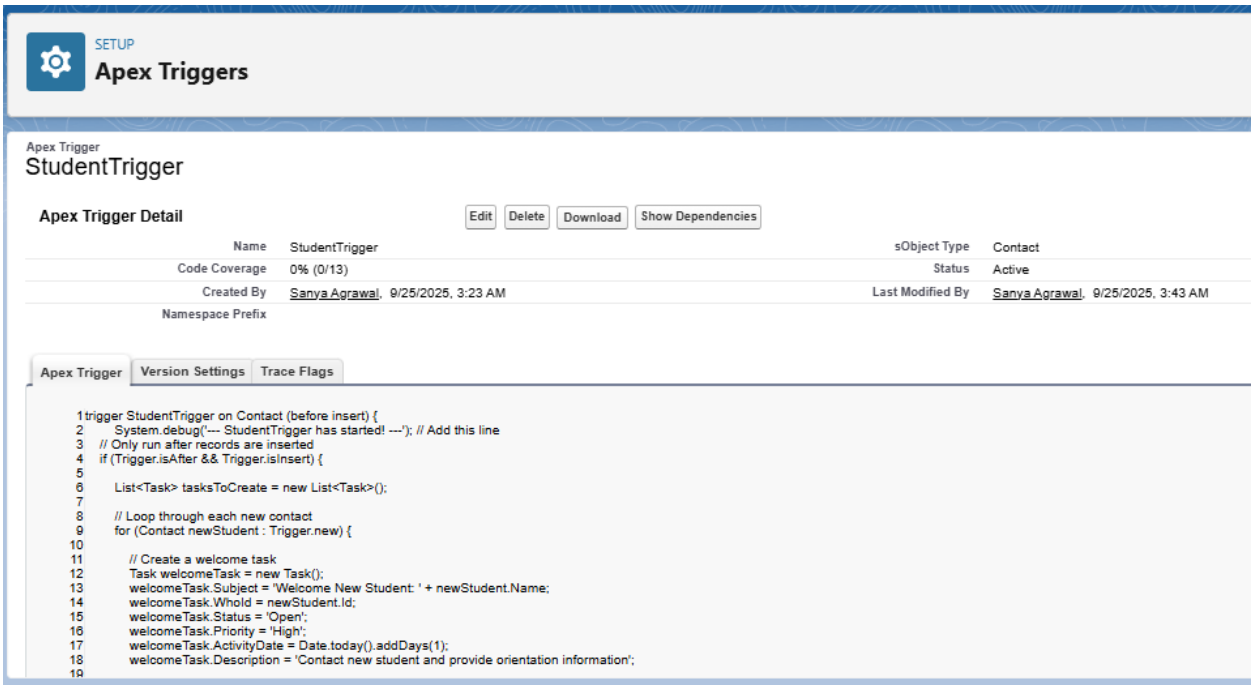
Timestamp	Event	Details
15:46:17:044	STATEMENT_EX...	[15]|
15:46:17:044	HEAP_ALLOCATE	[15]|Bytes:28
15:46:17:044	HEAP_ALLOCATE	[15]|Bytes:28
15:46:17:044	HEAP_ALLOCATE	[15]|Bytes:28
15:46:17:044	HEAP_ALLOCATE	[15]|Bytes:28
15:46:17:044	METHOD_EXIT	[10]|01pgL000005qLK1|Edulink.calculateGPA(List<Progress__c>)
15:46:17:044	SYSTEM_MODE...	false
15:46:17:044	VARIABLE_SCO...	[10]|studentGPA|Decimal|false|false
15:46:17:044	VARIABLE_ASSI...	[10]|studentGPA|3.60
15:46:17:044	STATEMENT_EX...	[13]|
15:46:17:044	HEAP_ALLOCATE	[13]|Bytes:39
15:46:17:044	HEAP_ALLOCATE	[13]|Bytes:4
15:46:17:044	HEAP_ALLOCATE	[13]|Bytes:43
15:46:17:044	USER_DEBUG	[13]|DEBUG|The calculated GPA for Arjun Patel is: 3.60
15:46:17:045	CUMULATIVE_L...	
15:46:17:045	LIMIT_USAGE_...	(default)|
15:46:17:000	LIMIT_USAGE_...	Number of SOQL queries: 1 out of 100
15:46:17:000	LIMIT_USAGE_...	Number of query rows: 1 out of 50000
15:46:17:000	LIMIT_USAGE_...	Number of SOSL queries: 0 out of 20
15:46:17:000	LIMIT_USAGE_...	Number of DML statements: 0 out of 150
15:46:17:000	LIMIT_USAGE_...	Number of Publish Immediate DML: 0 out of 150
15:46:17:000	LIMIT_USAGE_...	Number of DML rows: 0 out of 10000
15:46:17:000	LIMIT_USAGE_...	Maximum CPU time: 0 out of 10000
15:46:17:000	LIMIT_USAGE_...	Maximum heap size: 0 out of 6000000
15:46:17:000	LIMIT_USAGE_...	Number of callouts: 0 out of 100
15:46:17:000	LIMIT_USAGE_...	Number of Email Invocations: 0 out of 10
15:46:17:000	LIMIT_USAGE_...	Number of future calls: 0 out of 50
15:46:17:000	LIMIT_USAGE_...	Number of queueable jobs added to the queue: 0 out of 50

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

2. Apex Triggers (Before/After Insert/Update/Delete)

A trigger was implemented on the Contact object to automate tasks related to new students.

- **`StudentTrigger`**: This trigger fires `after insert` on Contact records. Its purpose is to automatically create a "Welcome New Student" Task and assign it to the new student's record owner, ensuring prompt follow-up.



3. Trigger Design Pattern

A formal trigger design pattern, such as a handler class, was not implemented in this phase. The logic for the `StudentTrigger` resides directly within the trigger file itself. This was suitable for the simple, single-purpose automation created.

4. SOQL

- **SOQL**: SOQL was used extensively across multiple classes (`StudentQueries`, `EnrollmentManager`, `GradeUpdateBatch`) to select, filter, and retrieve records from custom and standard objects.

cuteAnonymous @9/25/2025, 3:46:17 PM | StudentTrigger.apxt | StudentQueries.apxc | Log executeAnonymous @9/25/2025, 4:33:00 PM

Execution Log

Timestamp	Event	Details
19:53:50:026	STATEMENT_EX...	[9]|
19:53:50:026	METHOD_EXIT	[1]|01pgL000005qSF3|StudentQueries.getActiveStudents()
19:53:50:026	SYSTEM_MODE...	false
19:53:50:026	VARIABLE_ASSI...	[1]|students|[{"Id":"003gL00000CrG8AQAV","Email":"priya.sharma@student (12 more) ...","Phone":"+916657865421","Name":"Priya S
19:53:50:026	STATEMENT_EX...	[2]|
19:53:50:026	HEAP_ALLOCATE	[2]|Bytes:6
19:53:50:026	HEAP_ALLOCATE	[2]|Bytes:1
19:53:50:026	HEAP_ALLOCATE	[2]|Bytes:16
19:53:50:026	HEAP_ALLOCATE	[2]|Bytes:23
19:53:50:026	USER_DEBUG	[2]|DEBUG|Found 2 active students
19:53:50:026	STATEMENT_EX...	[4]|
19:53:50:026	SYSTEM_MODE...	false
19:53:50:026	HEAP_ALLOCATE	[4]|Bytes:5
19:53:50:026	METHOD_ENTRY	[4]|01pgL000005qSF3|StudentQueries.getTotalStudentCount()
19:53:50:026	STATEMENT_EX...	[27]|
19:53:50:026	STATEMENT_EX...	[30]|

Execution Log

Timestamp	Event	Details
19:53:50:035	SYSTEM_MODE...	false
19:53:50:035	HEAP_ALLOCATE	[4]|Bytes:4
19:53:50:035	VARIABLE_ASSI...	[4]|totalCount|3
19:53:50:035	STATEMENT_EX...	[5]|
19:53:50:035	HEAP_ALLOCATE	[5]|Bytes:16
19:53:50:035	HEAP_ALLOCATE	[5]|Bytes:1
19:53:50:035	HEAP_ALLOCATE	[5]|Bytes:17
19:53:50:035	USER_DEBUG	[5]|DEBUG|Total students: 3
19:53:50:035	CUMULATIVE_L...	
19:53:50:035	LIMIT_USAGE_...	(default)|
19:53:50:000	LIMIT_USAGE_...	Number of SOQL queries: 2 out of 100
19:53:50:000	LIMIT_USAGE_...	Number of query rows: 3 out of 50000
19:53:50:000	LIMIT_USAGE_...	Number of SOSL queries: 0 out of 20
19:53:50:000	LIMIT_USAGE_...	Number of DML statements: 0 out of 150
19:53:50:000	LIMIT_USAGE_...	Number of Publish Immediate DML: 0 out of 150

5. Collections (List, Set, Map)

Collections were used to handle records in a bulk-safe manner.

- **List**: `List<Contact>`, `List<Id>`, and `List<Enrollment__c>` were used to query, process, and insert multiple records at once.

- **Map**: A `Map<String, List<Contact>>` was used in `EnrollmentManager` to efficiently group students by their status field for reporting or summary purposes.

SETUP

Apex Classes

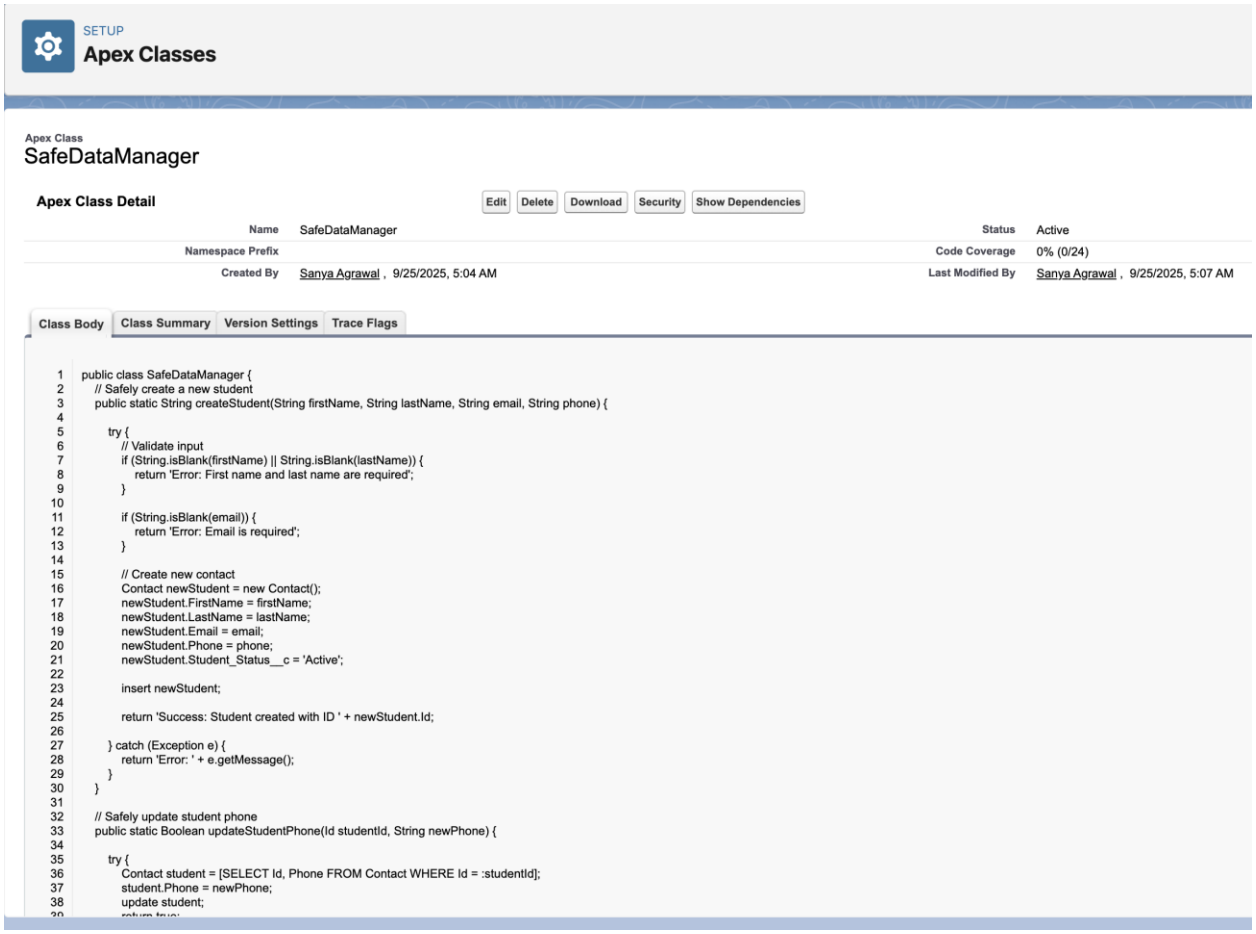
```
public class EnrollmentManager {    // Enroll multiple students in a course
    public static void enrollStudentsInCourse(List<Id> studentIds, Id courseId) {
        List<Enrollment__c> newEnrollments = new List<Enrollment__c>();
        // Loop through each student ID
        for (Id studentId : studentIds) {
            Enrollment__c enrollment = new Enrollment__c();
            enrollment.Student__c = studentId;
            enrollment.Course__c = courseId;
            enrollment.Enrollment_Date__c = Date.today();
            enrollment.Status__c = 'Enrolled';
            newEnrollments.add(enrollment);
        }
        // Insert all enrollments
        insert newEnrollments;
        System.debug('Enrolled ' + newEnrollments.size() + ' students');
    }

    // Get students by enrollment status
    public static Map<String, List<Contact>> getStudentsByStatus() {
        Map<String, List<Contact>> studentsByStatus = new Map<String, List<Contact>>();
        // This query now uses the correct 'Student_Status__c' field
        List<Contact> allStudents = [SELECT Id, Name, Student_Status__c
                                     FROM Contact
                                     WHERE Student_Status__c != null];
        for (Contact student : allStudents) {
            String status = student.Student_Status__c;
            if (!studentsByStatus.containsKey(status)) {
                studentsByStatus.put(status, new List<Contact>());
            }
            studentsByStatus.get(status).add(student);
        }
        return studentsByStatus;
```

6. Control Statements Batch Apex

Standard control statements were used to direct the flow of logic.

- **If-Else**: Used in `SafeDataManager` for input validation and in `EnrollmentManager` to check if collections were empty before processing.
- **For Loops**: Used to iterate over lists of records for processing in triggers and service classes



7. Batch Apex

A batch class was created to handle potentially large-scale data updates.

- **GradeUpdateBatch:** This class implements the Database.Batchable interface to find all "Active" enrollments with a blank final grade and update them to "In Progress". The job was executed and monitored via the **Apex Jobs** page in Setup.

SETUP
Apex Classes

Apex Class
GradeUpdate

Apex Class Detail

Edit | Delete | Download | Security | Show Dependencies

Name	GradeUpdate	Status	Active
Namespace Prefix		Code Coverage	0% (0/11)
Created By	Sanya Agrawal , 9/25/2025, 6:37 AM	Last Modified By	Sanya Agrawal , 9/25/2025, 6:50 AM

Class Body | Class Summary | Version Settings | Trace Flags

```
public class GradeUpdate implements Database.Batchable<sObject> {
    // Select records to process
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id, Final_Grade__c FROM Enrollment__c WHERE Status__c = \'Active\'');
    }

    // Process each batch of records
    public void execute(Database.BatchableContext bc, List<Enrollment__c> enrollments) {
        List<Enrollment__c> enrollmentsToUpdate = new List<Enrollment__c>();
        for (Enrollment__c enrollment : enrollments) {
            // Simple logic: if no grade assigned, give default grade
            if (String.isBlank(enrollment.Final_Grade__c)) {
                enrollment.Final_Grade__c = 'In Progress';
                enrollmentsToUpdate.add(enrollment);
            }
        }
        if (enrollmentsToUpdate.size() > 0) {
            update enrollmentsToUpdate;
        }
    }

    // Finish processing
    public void finish(Database.BatchableContext bc) {
        System.debug('Grade update completed');
    }
}
```

8. Exception Handling

Exception handling was implemented to make DML operations more robust.

- The **`SafeDataManager`** class uses `try-catch` blocks to gracefully handle potential DML exceptions (e.g., errors from validation rules) and return user-friendly error messages instead of crashing.

Apex Class

SafeDataManager

Apex Class Detail

Edit | Delete | Download | Security | Show Dependencies

Name	SafeDataManager	Status	Active
Namespace Prefix		Code Coverage	0% (0/24)
Created By	Sanya Agrawal , 9/25/2025, 5:04 AM	Last Modified By	Sanya Agrawal , 9/25/2025, 5:07 AM

Class Body | Class Summary | Version Settings | Trace Flags

```
public class SafeDataManager {
    // Safely create a new student
    public static String createStudent(String firstName, String lastName, String email, String phone) {

        try {
            // Validate input
            if (String.isBlank(firstName) || String.isBlank(lastName)) {
                return 'Error: First name and last name are required';
            }

            if (String.isBlank(email)) {
                return 'Error: Email is required';
            }

            // Create new contact
            Contact newStudent = new Contact();
            newStudent.FirstName = firstName;
            newStudent.LastName = lastName;
            newStudent.Email = email;
            newStudent.Phone = phone;
            newStudent.Student_Status__c = 'Active';

            insert newStudent;

            return 'Success: Student created with ID ' + newStudent.Id;

        } catch (Exception e) {
            return 'Error: ' + e.getMessage();
        }
    }

    // Safely update student phone
    public static Boolean updateStudentPhone(Id studentId, String newPhone) {
```

9. Test Classes

A test class was created to ensure code quality and verify business logic.

- **`TestEduLinkHelper`**: This class uses the `@isTest` annotation and `System.assertEquals()` methods to validate the outputs of the `EduLinkHelper` class, confirming its reliability and achieving code coverage.

10. Asynchronous Processing

- **Batch Apex** was the primary method of asynchronous processing implemented and tested in this phase.