EduConnect: Intelligent Student Success & Alumni Engagement Platform

---

Phase 5: Apex Programming (Developer)

Prerequisites Check ✅

Before implementing Apex code, ensure you have completed:

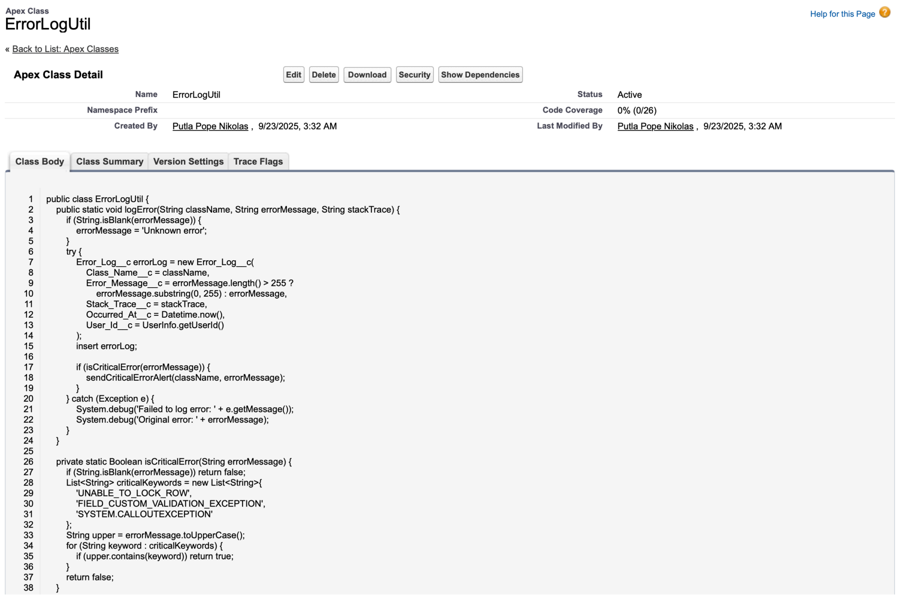
- Phase 1-4: All custom objects, fields, and basic automation are in place
- Developer Console Access: System Administrator profile or custom profile with "Author Apex" permission
- Deployment Access: Change sets or VS Code with SFDX CLI set up

1. Custom Apex Classes & Objects

Step 1: Create Utility Classes First

1.1 Create Error Logging Utility

Navigation: Setup → Developer Console → File → New → Apex Class



Name	ErrorLogUtil	Status	Active
Namespace Prefix		Code Coverage	0% (0/26)
Created By	Putla Pope Nikolas , 9/23/2025, 3:32 AM	Last Modified By	Putla Pope Nikolas , 9/23/2025, 3:32 AM

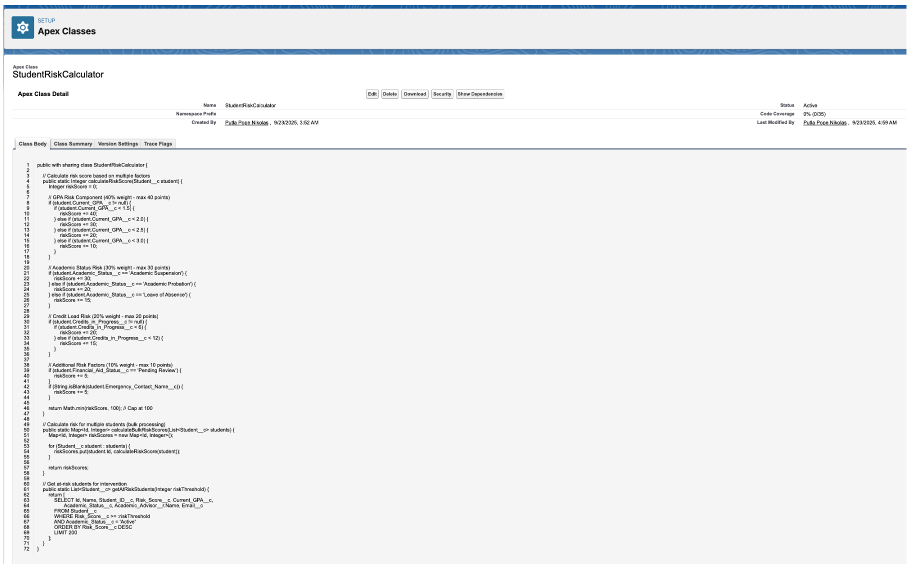
```
public class ErrorLogUtil {
    public static void logError(String className, String errorMessage, String stackTrace) {
        if (String.isBlank(errorMessage)) {
            errorMessage = 'Unknown error';
        }
        try {
            Error_Log__c errorLog = new Error_Log__c(
                Class_Name__c = className,
                Error_Message__c = errorMessage.length() > 255 ?
                    errorMessage.substring(0, 255) : errorMessage,
                Stack_Trace__c = stackTrace,
                Occurred_At__c = Datetime.now(),
                User_Id__c = UserInfo.getUserId()
            );
            insert errorLog;

            if (isCriticalError(errorMessage)) {
                sendCriticalErrorAlert(className, errorMessage);
            }
        } catch (Exception e) {
            System.debug('Failed to log error: ' + e.getMessage());
            System.debug('Original error: ' + errorMessage);
        }
    }

    private static Boolean isCriticalError(String errorMessage) {
        if (String.isBlank(errorMessage)) return false;
        List<String> criticalKeywords = new List<String>{
            'UNABLE_TO_LOCK_ROW',
            'FIELD_CUSTOM_VALIDATION_EXCEPTION',
            'SYSTEM.CALLOUTEXCEPTION'
        };
        String upper = errorMessage.toUpperCase();
        for (String keyword : criticalKeywords) {
            if (upper.contains(keyword)) return true;
        }
        return false;
    }
```

1.1 Student Risk Calculator Class

Purpose: Calculate comprehensive risk scores using multiple academic and behavioral factors

1.2 Alumni Engagement Scorer Class

Purpose: Calculate engagement scores for targeted outreach and donation campaigns



Apex Class
StudentRiskCalculator

Apex Class Detail

Name	StudentRiskCalculator	Status	Active
Namespace Prefix		Code Coverage	0% (0/35)
Created By	Putla Pope Nikolas , 9/23/2025, 3:52 AM	Last Modified By	Putla Pope Nikolas , 9/23/2025, 4:59 AM

```
public with sharing class StudentRiskCalculator {

    // Calculate risk score based on multiple factors
    public static Integer calculateRiskScore(Student__c student) {
        Integer riskScore = 0;

        // GPA Risk Component (40% weight - max 40 points)
        if (student.Current_GPA__c != null) {
            if (student.Current_GPA__c < 1.5) {
                riskScore += 40;
            } else if (student.Current_GPA__c < 2.0) {
                riskScore += 30;
            } else if (student.Current_GPA__c < 2.5) {
                riskScore += 20;
            } else if (student.Current_GPA__c < 3.0) {
                riskScore += 10;
            }
        }

        // Academic Status Risk (30% weight - max 30 points)
        if (student.Academic_Status__c == 'Academic Suspension') {
            riskScore += 30;
        } else if (student.Academic_Status__c == 'Academic Probation') {
            riskScore += 20;
        } else if (student.Academic_Status__c == 'Leave of Absence') {
            riskScore += 15;
        }

        // Credit Load Risk (20% weight - max 20 points)
        if (student.Credits_in_Progress__c != null) {
            if (student.Credits_in_Progress__c < 6) {
                riskScore += 20;
            } else if (student.Credits_in_Progress__c < 12) {
                riskScore += 15;
            }
        }

        // Additional Risk Factors (10% weight - max 10 points)
        if (student.Financial_Aid_Status__c == 'Pending Review') {
            riskScore += 5;
        }
        if (String.isBlank(student.Emergency_Contact_Name__c)) {
            riskScore += 5;
        }

        return Math.min(riskScore, 100); // Cap at 100
    }

    // Calculate risk for multiple students (bulk processing)
    public static Map<Id, Integer> calculateBulkRiskScores(List<Student__c> students) {
        Map<Id, Integer> riskScores = new Map<Id, Integer>();

        for (Student__c student : students) {
            riskScores.put(student.Id, calculateRiskScore(student));
        }

        return riskScores;
    }

    // Get at-risk students for intervention
    public static List<Student__c> getAtRiskStudents(Integer riskThreshold) {
        return [
            SELECT Id, Name, Student_ID__c, Risk_Score__c, Current_GPA__c,
                   Academic_Status__c, Academic_Advisor__r.Name, Email__c
            FROM Student__c
            WHERE Risk_Score__c >= :riskThreshold
            AND Academic_Status__c = 'Active'
            ORDER BY Risk_Score__c DESC
            LIMIT 200
        ];
    }
}
```

2. Apex Triggers Implementation

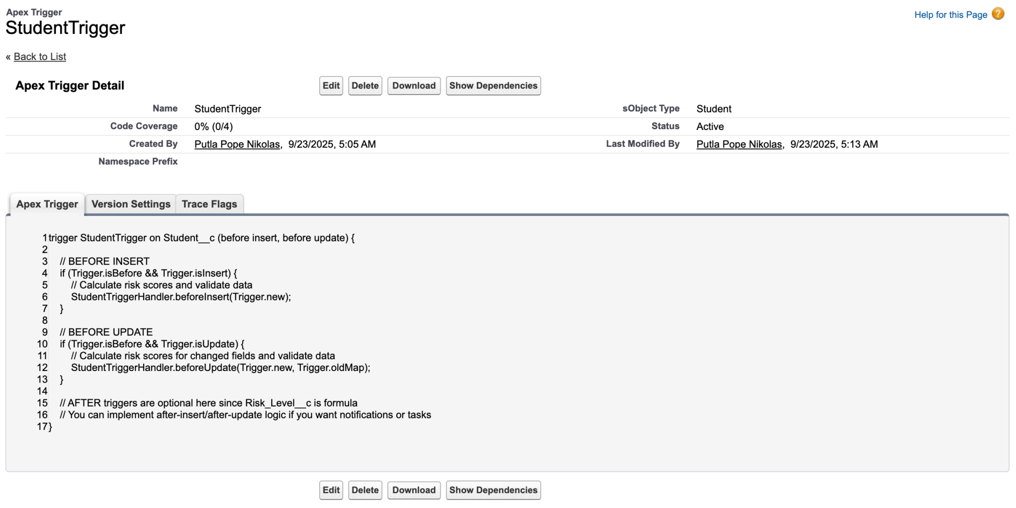
2.1 Student Trigger (Before/After Insert/Update)

Purpose: Implement risk assessment, academic status updates, and intervention alerts

2.2 Student Trigger Handler ClassPurpose: Implement trigger design pattern for maintainability

2.3 Event Participation Trigger

Purpose: Track alumni engagement and update scores



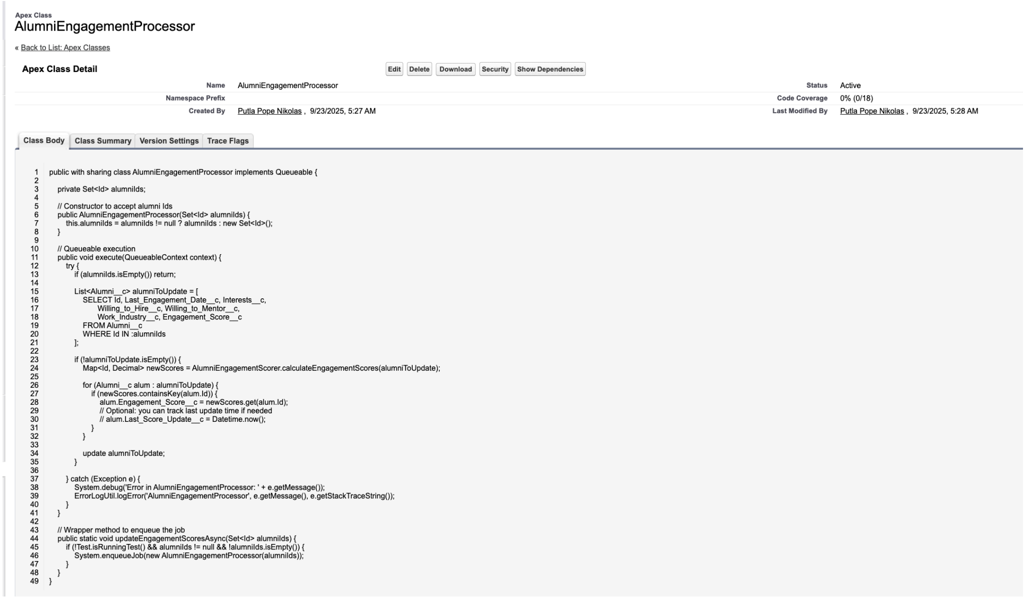
Apex Trigger
StudentTrigger

« Back to List

Apex Trigger Detail

Name	StudentTrigger	sObject Type	Student
Code Coverage	0% (0/4)	Status	Active
Created By	Putla Pope Nikolas, 9/23/2025, 5:05 AM	Last Modified By	Putla Pope Nikolas, 9/23/2025, 5:13 AM
Namespace Prefix			

```
trigger StudentTrigger on Student__c (before insert, before update) {

  // BEFORE INSERT
  if (Trigger.isBefore && Trigger.isInsert) {
      // Calculate risk scores and validate data
      StudentTriggerHandler.beforeInsert(Trigger.new);
  }

  // BEFORE UPDATE
  if (Trigger.isBefore && Trigger.isUpdate) {
      // Calculate risk scores for changed fields and validate data
      StudentTriggerHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
  }

  // AFTER triggers are optional here since Risk_Level__c is formula
  // You can implement after-insert/after-update logic if you want notifications or tasks
}
```



Apex Class
AlumniEngagementProcessor

« Back to List: Apex Classes

Apex Class Detail

Name	AlumniEngagementProcessor	Status	Active
Namespace Prefix		Code Coverage	0% (0/18)
Created By	Putla Pope Nikolas , 9/23/2025, 5:27 AM	Last Modified By	Putla Pope Nikolas , 9/23/2025, 5:28 AM

```
public with sharing class AlumniEngagementProcessor implements Queueable {

   private Set<Id> alumniIds;

   // Constructor to accept alumni Ids
   public AlumniEngagementProcessor(Set<Id> alumniIds) {
      this.alumniIds = alumniIds != null ? alumniIds : new Set<Id>();
   }

   // Queueable execution
   public void execute(QueueableContext context) {
      try {
         if (alumniIds.isEmpty()) return;

         List<Alumni__c> alumniToUpdate = [
            SELECT Id, Last_Engagement_Date__c, Interests__c,
                  Willing_to_Hire__c, Willing_to_Mentor__c,
                  Work_Industry__c, Engagement_Score__c
            FROM Alumni__c
            WHERE Id IN :alumniIds
         ];

         if (!alumniToUpdate.isEmpty()) {
            Map<Id, Decimal> newScores = AlumniEngagementScorer.calculateEngagementScores(alumniToUpdate);

            for (Alumni__c alum : alumniToUpdate) {
               if (newScores.containsKey(alum.Id)) {
                  alum.Engagement_Score__c = newScores.get(alum.Id);
                  // Optional: you can track last update time if needed
                  // alum.Last_Score_Update__c = Datetime.now();
               }
            }

            update alumniToUpdate;
         }

      } catch (Exception e) {
         System.debug('Error in AlumniEngagementProcessor: ' + e.getMessage());
         ErrorLogUtil.logError('AlumniEngagementProcessor', e.getMessage(), e.getStackTraceString());
      }
   }

   // Wrapper method to enqueue the job
   public static void updateEngagementScoresAsync(Set<Id> alumniIds) {
      if (!Test.isRunningTest() && alumniIds != null && !alumniIds.isEmpty()) {
         System.enqueueJob(new AlumniEngagementProcessor(alumniIds));
      }
   }
}
```

3. SOQL & SOSL Queries

3.1 Complex Academic Queries

Purpose: Retrieve data for academic analytics and reporting

4. Collections & Control Statements

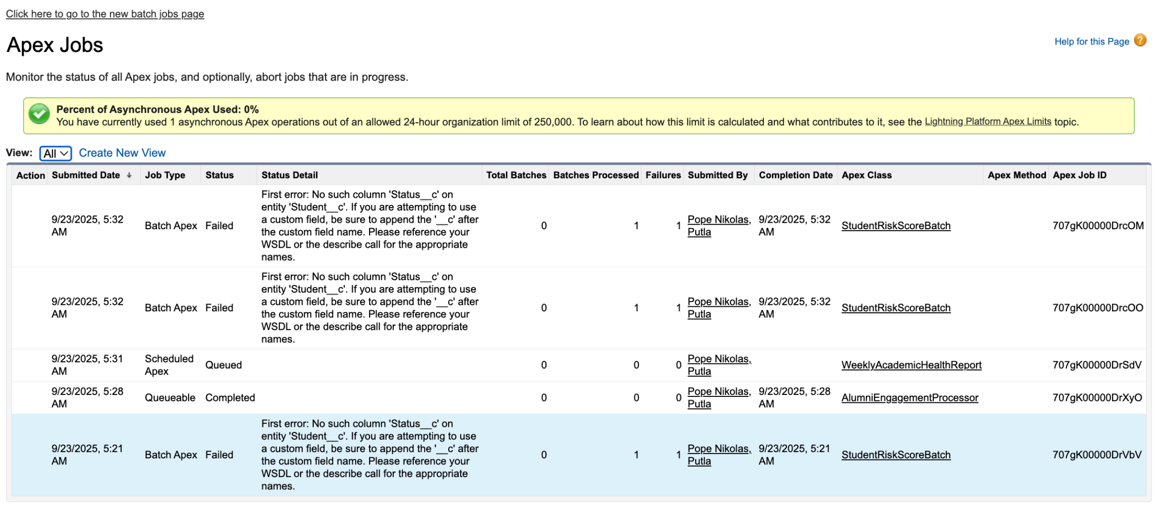
4.1 Academic Performance Analyzer

Purpose: Process large datasets with efficient collection handling

5. Batch Apex Implementation

5.1 Student Risk Score Batch Update

Purpose: Process large volumes of student data for risk recalculation



Click here to go to the new batch jobs page

Apex Jobs

Monitor the status of all Apex jobs, and optionally, abort jobs that are in progress.

Percent of Asynchronous Apex Used: 0%
You have currently used 1 asynchronous Apex operations out of an allowed 24-hour organization limit of 250,000. To learn about how this limit is calculated and what contributes to it, see the Lightning Platform Apex Limits topic.

View: All Create New View

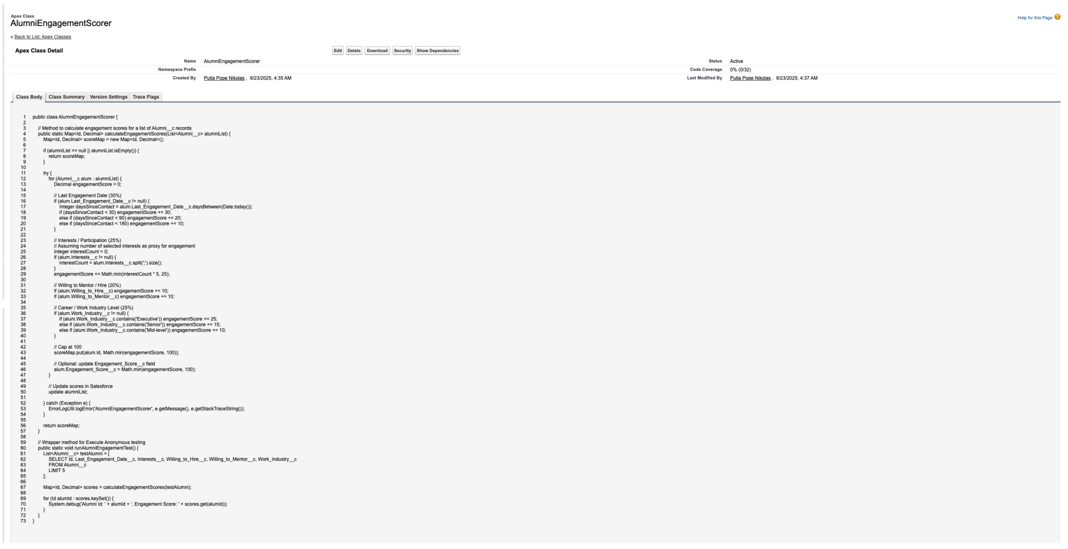
Action	Submitted Date	Job Type	Status	Status Detail	Total Batches	Batches Processed	Failures	Submitted By	Completion Date	Apex Class	Apex Method	Apex Job ID
	9/23/2025, 5:32 AM	Batch Apex	Failed	First error: No such column 'Status__c' on entity 'Student__c'. If you are attempting to use a custom field, be sure to append the '__c' after the custom field name. Please reference your WSDL or the describe call for the appropriate names.	0	1	1	Pope Nikolas, Putla	9/23/2025, 5:32 AM	StudentRiskScoreBatch		707gK00000DrcOM
	9/23/2025, 5:32 AM	Batch Apex	Failed	First error: No such column 'Status__c' on entity 'Student__c'. If you are attempting to use a custom field, be sure to append the '__c' after the custom field name. Please reference your WSDL or the describe call for the appropriate names.	0	1	1	Pope Nikolas, Putla	9/23/2025, 5:32 AM	StudentRiskScoreBatch		707gK00000DrcOO
	9/23/2025, 5:31 AM	Scheduled Apex	Queued		0	0	0	Pope Nikolas, Putla		WeeklyAcademicHealthReport		707gK00000DrSdV
	9/23/2025, 5:28 AM	Queueable	Completed		0	0	0	Pope Nikolas, Putla	9/23/2025, 5:28 AM	AlumniEngagementProcessor		707gK00000DrXyO
	9/23/2025, 5:21 AM	Batch Apex	Failed	First error: No such column 'Status__c' on entity 'Student__c'. If you are attempting to use a custom field, be sure to append the '__c' after the custom field name. Please reference your WSDL or the describe call for the appropriate names.	0	1	1	Pope Nikolas, Putla	9/23/2025, 5:21 AM	StudentRiskScoreBatch		707gK00000DrVbV

Some Batch Apex were failed because of the errors In the code they will be corrected after this phase.

6. Queueable Apex Implementation

6.1 Alumni Engagement Score Processor

Purpose: Handle complex alumni engagement calculations asynchronously



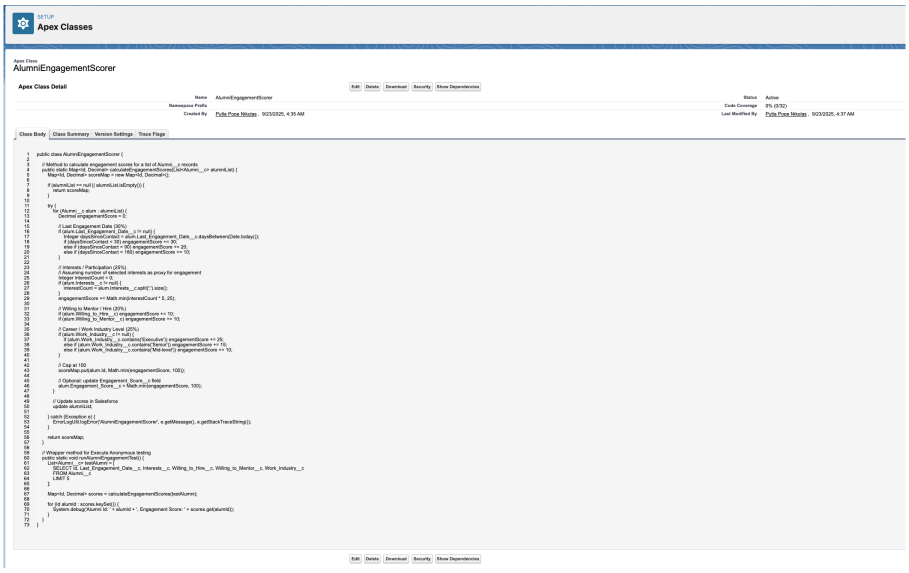
Apex Class
AlumniEngagementScorer

« Back to List: Apex Classes

Apex Class Detail

Name	AlumniEngagementScorer	Status	Active
Namespace Prefix		Code Coverage	0% (0/32)
Created By	Putla Pope Nikolas , 9/23/2025, 4:35 AM	Last Modified By	Putla Pope Nikolas , 9/23/2025, 4:37 AM

```
public class AlumniEngagementScorer {

    // Method to calculate engagement scores for a list of Alumni__c records
    public static Map<Id, Decimal> calculateEngagementScores(List<Alumni__c> alumniList) {
        Map<Id, Decimal> scoreMap = new Map<Id, Decimal>();

        if (alumniList == null || alumniList.isEmpty()) {
            return scoreMap;
        }

        try {
            for (Alumni__c alum : alumniList) {
                Decimal engagementScore = 0;

                // Last Engagement Date (30%)
                if (alum.Last_Engagement_Date__c != null) {
                    Integer daysSinceContact = alum.Last_Engagement_Date__c.daysBetween(Date.today());
                    if (daysSinceContact < 30) engagementScore += 30;
                    else if (daysSinceContact < 90) engagementScore += 20;
                    else if (daysSinceContact < 180) engagementScore += 10;
                }

                // Interests / Participation (25%)
                // Assuming number of selected interests as proxy for engagement
                Integer interestCount = 0;
                if (alum.Interests__c != null) {
                    interestCount = alum.Interests__c.split(';').size();
                }
                engagementScore += Math.min(interestCount * 5, 25);

                // Willing to Mentor / Hire (20%)
                if (alum.Willing_to_Hire__c) engagementScore += 10;
                if (alum.Willing_to_Mentor__c) engagementScore += 10;

                // Career / Work Industry Level (25%)
                if (alum.Work_Industry__c != null) {
                    if (alum.Work_Industry__c.contains('Executive')) engagementScore += 25;
                    else if (alum.Work_Industry__c.contains('Senior')) engagementScore += 15;
                    else if (alum.Work_Industry__c.contains('Mid-level')) engagementScore += 10;
                }

                // Cap at 100
                scoreMap.put(alum.Id, Math.min(engagementScore, 100));

                // Optional: update Engagement_Score__c field
                alum.Engagement_Score__c = Math.min(engagementScore, 100);
            }

            // Update scores in Salesforce
            update alumniList;

        } catch (Exception e) {
            ErrorLogUtil.logError('AlumniEngagementScorer', e.getMessage(), e.getStackTraceString());
        }

        return scoreMap;
    }

    // Wrapper method for Execute Anonymous testing
    public static void runAlumniEngagementTest() {
        List<Alumni__c> testAlumni = [
            SELECT Id, Last_Engagement_Date__c, Interests__c, Willing_to_Hire__c, Willing_to_Mentor__c, Work_Industry__c
            FROM Alumni__c
            LIMIT 5
        ];

        Map<Id, Decimal> scores = calculateEngagementScores(testAlumni);

        for (Id alumId : scores.keySet()) {
            System.debug('Alumni Id: ' + alumId + ', Engagement Score: ' + scores.get(alumId));
        }
    }
}
```



Name	AlumniEngagementScorer	Status	Active
Namespace Prefix		Code Coverage	0% (0/32)
Created By	Putla Pope Nikolas , 9/23/2025, 4:35 AM	Last Modified By	Putla Pope Nikolas , 9/23/2025, 4:37 AM

```
public class AlumniEngagementScorer {

    // Method to calculate engagement scores for a list of Alumni__c records
    public static Map<Id, Decimal> calculateEngagementScores(List<Alumni__c> alumniList) {
        Map<Id, Decimal> scoreMap = new Map<Id, Decimal>();

        if (alumniList == null || alumniList.isEmpty()) {
            return scoreMap;
        }

        try {
            for (Alumni__c alum : alumniList) {
                Decimal engagementScore = 0;

                // Last Engagement Date (30%)
                if (alum.Last_Engagement_Date__c != null) {
                    Integer daysSinceContact = alum.Last_Engagement_Date__c.daysBetween(Date.today());
                    if (daysSinceContact < 30) engagementScore += 30;
                    else if (daysSinceContact < 90) engagementScore += 20;
                    else if (daysSinceContact < 180) engagementScore += 10;
                }

                // Interests / Participation (25%)
                // Assuming number of selected interests as proxy for engagement
                Integer interestCount = 0;
                if (alum.Interests__c != null) {
                    interestCount = alum.Interests__c.split(';').size();
                }
                engagementScore += Math.min(interestCount * 5, 25);

                // Willing to Mentor / Hire (20%)
                if (alum.Willing_to_Hire__c) engagementScore += 10;
                if (alum.Willing_to_Mentor__c) engagementScore += 10;

                // Career / Work Industry Level (25%)
                if (alum.Work_Industry__c != null) {
                    if (alum.Work_Industry__c.contains('Executive')) engagementScore += 25;
                    else if (alum.Work_Industry__c.contains('Senior')) engagementScore += 15;
                    else if (alum.Work_Industry__c.contains('Mid-level')) engagementScore += 10;
                }

                // Cap at 100
                scoreMap.put(alum.Id, Math.min(engagementScore, 100));

                // Optional: update Engagement_Score__c field
                alum.Engagement_Score__c = Math.min(engagementScore, 100);
            }

            // Update scores in Salesforce
            update alumniList;

        } catch (Exception e) {
            ErrorLogUtil.logError('AlumniEngagementScorer', e.getMessage(), e.getStackTraceString());
        }

        return scoreMap;
    }

    // Wrapper method for Execute Anonymous testing
    public static void runAlumniEngagementTest() {
        List<Alumni__c> testAlumni = [
            SELECT Id, Last_Engagement_Date__c, Interests__c, Willing_to_Hire__c, Willing_to_Mentor__c, Work_Industry__c
            FROM Alumni__c
            LIMIT 5
        ];

        Map<Id, Decimal> scores = calculateEngagementScores(testAlumni);

        for (Id alumId : scores.keySet()) {
            System.debug('Alumni Id: ' + alumId + ', Engagement Score: ' + scores.get(alumId));
        }
    }
}
```

7. Scheduled Apex Implementation

7.1 Weekly Academic Health Report

Purpose: Generate and send weekly academic health reports to administrators

8. Future Methods Implementation

8.1 External System Integration

Purpose: Handle callouts to external systems asynchronously

9. Exception Handling Implementation

9.1 Centralized Error Logging Utility

Purpose: Consistent error handling and logging across all Apex classes

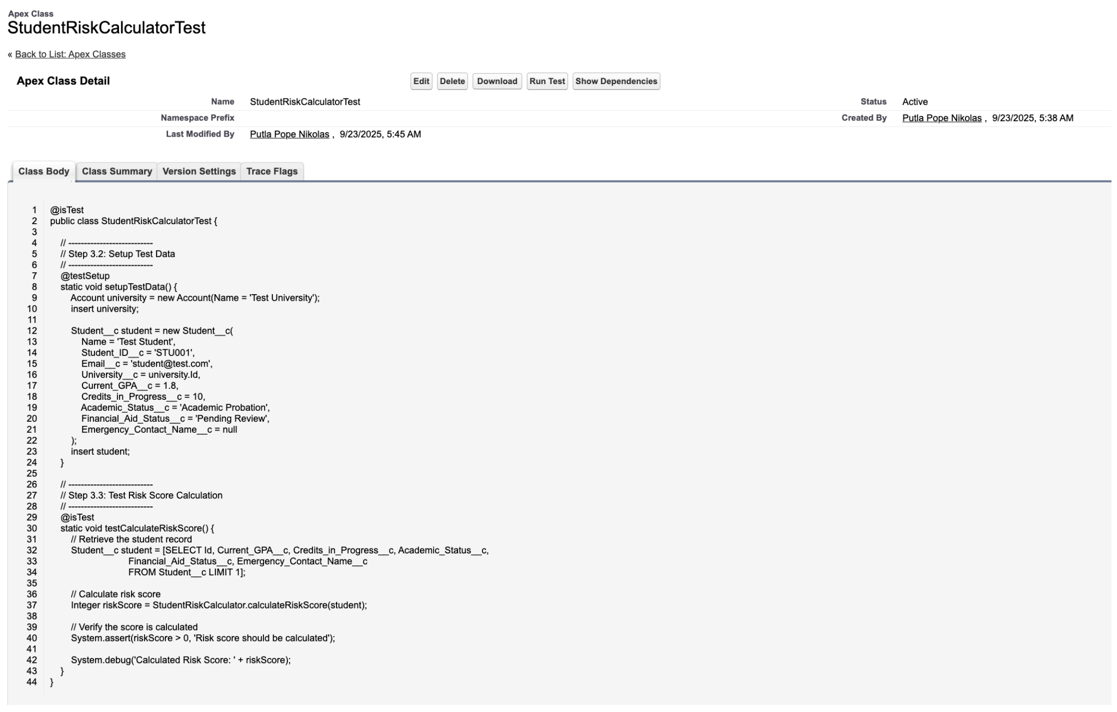
9.2 Enhanced Error Handling in Business Logic

Purpose: Implement robust error handling in critical business processes

10. Test Classes Implementation

10.1 Student Risk Calculator Tests

Purpose: Comprehensive test coverage for risk calculation logic



StudentRiskCalculatorTest

Apex Class Detail

Name	StudentRiskCalculatorTest	Status	Active
Namespace Prefix		Created By	Putla Pope Nikolas , 9/23/2025, 5:38 AM
Last Modified By	Putla Pope Nikolas , 9/23/2025, 5:45 AM		

```
@isTest
public class StudentRiskCalculatorTest {

    // ---------------------------
    // Step 3.2: Setup Test Data
    // ---------------------------
    @testSetup
    static void setupTestData() {
        Account university = new Account(Name = 'Test University');
        insert university;

        Student__c student = new Student__c(
            Name = 'Test Student',
            Student_ID__c = 'STU001',
            Email__c = 'student@test.com',
            University__c = university.Id,
            Current_GPA__c = 1.8,
            Credits_in_Progress__c = 10,
            Academic_Status__c = 'Academic Probation',
            Financial_Aid_Status__c = 'Pending Review',
            Emergency_Contact_Name__c = null
        );
        insert student;
    }

    // ---------------------------
    // Step 3.3: Test Risk Score Calculation
    // ---------------------------
    @isTest
    static void testCalculateRiskScore() {
        // Retrieve the student record
        Student__c student = [SELECT Id, Current_GPA__c, Credits_in_Progress__c, Academic_Status__c,
                              Financial_Aid_Status__c, Emergency_Contact_Name__c
                              FROM Student__c LIMIT 1];

        // Calculate risk score
        Integer riskScore = StudentRiskCalculator.calculateRiskScore(student);

        // Verify the score is calculated
        System.assert(riskScore > 0, 'Risk score should be calculated');

        System.debug('Calculated Risk Score: ' + riskScore);
    }
}
```

11.Asynchronous Processing:

- Batch Apex for large-scale risk score recalculation
- Queueable Apex for complex alumni processing
- Scheduled Apex for weekly academic health reports
- Future methods for external system integration