Phase 5:Apex Programming(Developer)

We will implement the "Gamification" feature for our app. Our goal is to write a piece of Apex code that automatically awards Engagement Points to a student whenever they complete a meaningful action like Support Interaction or Well being Check-in.

5.1. Classes & Objects

- **StudentTriggerHandler (Class):** A public Apex class was created to act as a trigger handler. This class encapsulates all business logic, following best practices to keep the triggers themselves lean. It contains a public static method, calculateEngagementPoints, which can be called from multiple triggers, ensuring the code is reusable and centralized.
- **Objects Involved:** The logic interacts with four key objects:
    - Contact (Student): The parent object where the final Engagement_Points__c are stored.
    - Support_Interaction__c: One of the child objects that initiates the point calculation.
    - Well_being_Check_in__c: The second child object that also initiates a point calculation.
    - RecordType: This object was queried to dynamically find the ID for our "Academic Support" process, avoiding hardcoded IDs.

Apex Class Code:

```
public class StudentTriggerHandler {

   public static void calculateEngagementPoints(Set<Id> studentIdsToUpdate) {

      Id academicRecordTypeId;

      try {

         academicRecordTypeId = [SELECT Id FROM RecordType WHERE SobjectType =
'Support_Interaction__c' AND Name = 'Academic Support' LIMIT 1].Id;

      } catch (QueryException e) {

         System.debug('Error: Could not find "Academic Support" record type. ' +
e.getMessage());

         return;

      }
```

```
List<Support_Interaction__c> interactions = [SELECT Id, Student__c FROM Support_Interaction__c
                                        WHERE Student__c IN :studentIdsToUpdate
                                        AND RecordTypeId = :academicRecordTypeId
                                        AND Support_Type__c IN ('Current Subject Tutoring', 'Backlog Guidance')];


List<Well_being_Check_in__c> checkins = [SELECT Id, Student__c FROM Well_being_Check_in__c
                                    WHERE Student__c IN :studentIdsToUpdate];


Map<Id, Integer> studentPointsMap = new Map<Id, Integer>();


for (Support_Interaction__c si : interactions) {
    if (!studentPointsMap.containsKey(si.Student__c)) {
        studentPointsMap.put(si.Student__c, 0);
    }
    studentPointsMap.put(si.Student__c, studentPointsMap.get(si.Student__c) + 10);
}


for (Well_being_Check_in__c wb : checkins) {
    if (!studentPointsMap.containsKey(wb.Student__c)) {
        studentPointsMap.put(wb.Student__c, 0);
    }
    studentPointsMap.put(wb.Student__c, studentPointsMap.get(wb.Student__c) + 5);
}
List<Contact> studentsToUpdate = new List<Contact>();
for (Id studentId : studentPointsMap.keySet()) {
```

```
            Contact student = new Contact(
                Id = studentId,
                Engagement_Points__c = studentPointsMap.get(studentId)
            );
            studentsToUpdate.add(student);
        }
        if (!studentsToUpdate.isEmpty()) {
            update studentsToUpdate;
        }
    }
}
```

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

- **SupportInteractionTrigger:** An after insert trigger was created on the Support_Interaction__c object. When a new interaction record is saved, this trigger fires, collects the Student__c ID from the new record(s), and passes this ID to the StudentTriggerHandler class.

  Apex Trigger Code:

```
trigger SupportInteractionTrigger on Support_Interaction__c (after insert) {
    Set<Id> studentIds = new Set<Id>();
    for (Support_Interaction__c si : Trigger.new) {
        studentIds.add(si.Student__c);
    }

    if (!studentIds.isEmpty()) {
        StudentTriggerHandler.calculateEngagementPoints(studentIds);
    }
}
```

- **WellBeingCheckinTrigger:** A second after insert trigger was created on the Well_being_Check_in__c object. It performs the exact same function: collecting the Student__c ID(s) and calling the same centralized handler method.

  Apex Trigger Code:

```
trigger WellBeingCheckinTrigger on Well_being_Check_in__c (after insert) {
    Set<Id> studentIds = new Set<Id>();
    for (Well_being_Check_in__c wb : Trigger.new) {
        studentIds.add(wb.Student__c);
    }

    if (!studentIds.isEmpty()) {
        StudentTriggerHandler.calculateEngagementPoints(studentIds);
    }
}
```

- **StudentTrigger (Decommissioned):** An initial trigger on the Contact object was considered but was decommissioned. The correct design pattern for this use case is to place the triggers on the child objects that initiate the event, not the parent object being updated.

5.3. Trigger Design Pattern

- A **Trigger Handler Pattern** was implemented. This is a best-practice architecture where the trigger files (.apxt) contain virtually no logic. Their sole responsibility is to delegate the execution to a specific method within a handler class (.apxc).
- **Benefits:**
  - **Reusability:** The calculateEngagementPoints logic is written once in the handler and is called by two different triggers.
  - **Bulkification:** The triggers collect all affected Student IDs into a Set<Id> before making a single call to the handler, ensuring the logic is processed efficiently in bulk.
  - **Testability:** Logic inside a class is much easier to test than logic inside a trigger file.

5.4. SOQL & SOSL

SOQL (Salesforce Object Query Language) was used extensively within the handler class to fetch the necessary data.

Query 1 (Dynamic Record Type ID): . This query makes the code portable by fetching the Record Type ID dynamically.

```
SELECT Id FROM RecordType
WHERE SobjectType = 'Support_Interaction__c'
AND Name = 'Academic Support'
```

Query 2 (Filtered Interactions): A query on Support_Interaction__c was used with a WHERE clause to fetch only the records that matched our specific criteria (correct Record Type ID and the Support_Type__c being in our list of values).

```
SELECT Id, Student__c FROM Support_Interaction__c
WHERE Student__c IN :studentIdsToUpdate
AND RecordTypeId = :academicRecordTypeId
AND Support_Type__c IN ('Current Subject Tutoring', 'Backlog Guidance')
```

Query 3 (All Check-ins): A simpler query on Well_being_Check_in__c was used to fetch all check-ins for the affected students.

```
SELECT Id, Student__c FROM Well_being_Check_in__c
WHERE Student__c IN :studentIdsToUpdate];
```

5.5. Collections: List, Set, Map

Collections were used to handle data in a bulk-safe and efficient manner.

- **Set<Id>:** Used in the triggers to collect a unique list of Student IDs. A Set is used because if multiple interactions for the same student are created in one transaction, we only need their ID once.
- **List<SObject>:** The results of all SOQL queries were stored in Lists (e.g., List<Support_Interaction__c>).
- **Map<Id, Integer>:** A Map was the crucial collection used for the final calculation. It acted as a "scorecard," using the Student's Id as the key and their calculated Engagement Points as the value. This is far more efficient than querying for each student's current points.

5.6. Exception Handling

- A **try/catch block** was implemented for the RecordType SOQL query. This is a critical piece of exception handling. If the "Academic Support" Record Type is ever deleted or renamed, the query will throw a QueryException. Our catch block prevents this error from crashing the entire trigger and halting the user's save. Instead, it logs an error using System.debug() and gracefully exits the method, allowing the main record to save.

  Code:

```
try {
 academicRecordTypeId = [SELECT Id FROM RecordType WHERE SobjectType = 'Support_Interaction__c' AND Name = 'Academic Support' LIMIT 1].Id;
    } catch (QueryException e) {
        System.debug('Error: Could not find "Academic Support" record type. ' + e.getMessage());
        return;
    }
```

5.7. Test Classes

- A dedicated test class, **StudentTriggerHandler_Test**, was created with the @isTest annotation.
- **Data Isolation:** The test method creates all its own test data from scratch (Account, Contact, Support Interaction, etc.), ensuring it can run in any org without depending on existing data.
- **Required Field Handling:** The test was updated to satisfy all validation rules and required fields on the Contact and Support_Interaction__c objects, proving how tests validate the entire system, not just the code.
- **Assertions (System.assertEquals)**: After the test performs its actions, it uses System.assertEquals(15, updatedStudent.Engagement_Points__c, ...) to programmatically verify that the final calculated value is correct. A "green" test run is definitive proof that the logic works as designed.

Apex Class Code:

```
@isTest
private class StudentTriggerHandler_Test {
```

```
    @isTest
    static void testPointsAwardedForInteractions() {
        Account uniAccount = new Account(Name='Test University');
        insert uniAccount;

        Contact testStudent = new Contact(
            LastName = 'TestStudent3',
            AccountId = uniAccount.Id,
            Student_ID__c = 'AXh56',
            Overall_CGPA__c = 6.9,
            Expected_Graduation_Date__c = Date.today().addYears(4),
            Attendence_Percentage__c = 69,
            Backlogs__c = 0,
            Enrolled_Date__c = Date.today().addYears(-1)
        );
        insert testStudent;

        Id academicRTId = [SELECT Id FROM RecordType WHERE SobjectType =
'Support_Interaction__c' AND Name = 'Academic Support' LIMIT 1].Id;

        Test.startTest();

        // Action 1: Create a Tutoring Session with the required date
        Support_Interaction__c tutoring = new Support_Interaction__c(
            Student__c = testStudent.Id,
            RecordTypeId = academicRTId,
            Support_Type__c = 'Current Subject Tutoring',
```

```
        Interaction_Date__c = Date.today()
    );
    insert tutoring;

    // Action 2: Create a Well-being Check-in with the required date
    Well_being_Check_in__c checkin = new Well_being_Check_in__c(
        Student__c = testStudent.Id,
        Date_of_Check__c = Date.today()
    );
    insert checkin;

    Test.stopTest();

    Contact updatedStudent = [SELECT Id, Engagement_Points__c FROM Contact WHERE Id
= :testStudent.Id];

    // Expected points: 10 (tutoring) + 5 (check-in) = 15
    System.debug('FINAL ENGAGEMENT POINTS:updatedStudent.Engagement_Points__c);
    System.assertEquals(15, updatedStudent.Engagement_Points__c, 'The engagement points
should be 15.');
  }
}
```

Result:

I created a new student record with a Academic Support and Well Being Check in

And the Engagement points are updated to 15 points

Name Mr. Alex Fakir	Phone
Account Name Southern University	Home Phone
Account Owner Sreeja Nayani	Mobile
Birthdate 	Email
Student ID AXg45	Attendence Percentage 89.00%
Engagement Points 15	Overall CGPA 7.90
Enrolled Date 6/19/2024	Backlogs 2
Expected Graduation Date 7/27/2028	Risk Score 50.00
Risk Tier Medium	Student Owner Sreenidhi Muthyala
Mailing Address 	Other Address