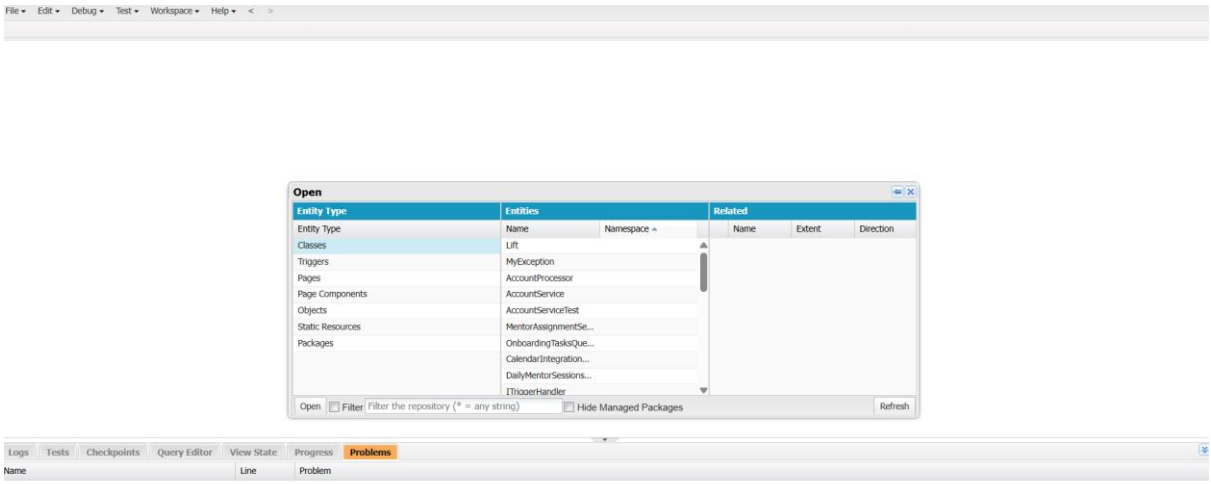
Phase 5 – Apex Programming (Developer)

Goal: The goal of this phase is to add advanced logic and automation using Apex classes, triggers, and asynchronous processing. This ensures robust handling of validations and external integrations within the system.

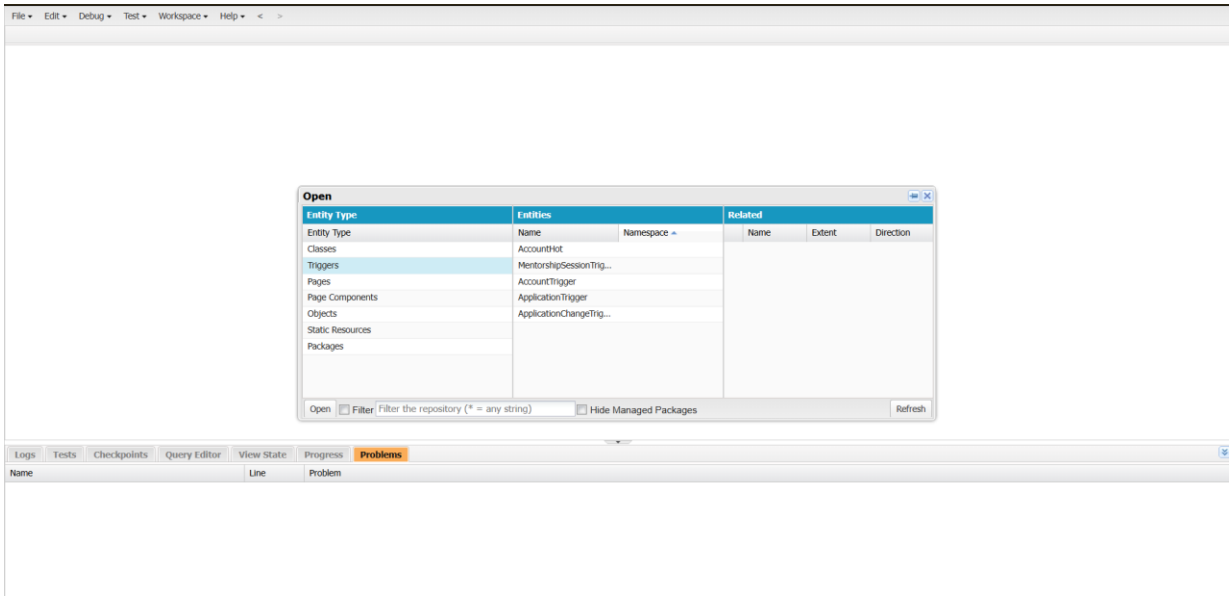
1. Classes & Objects

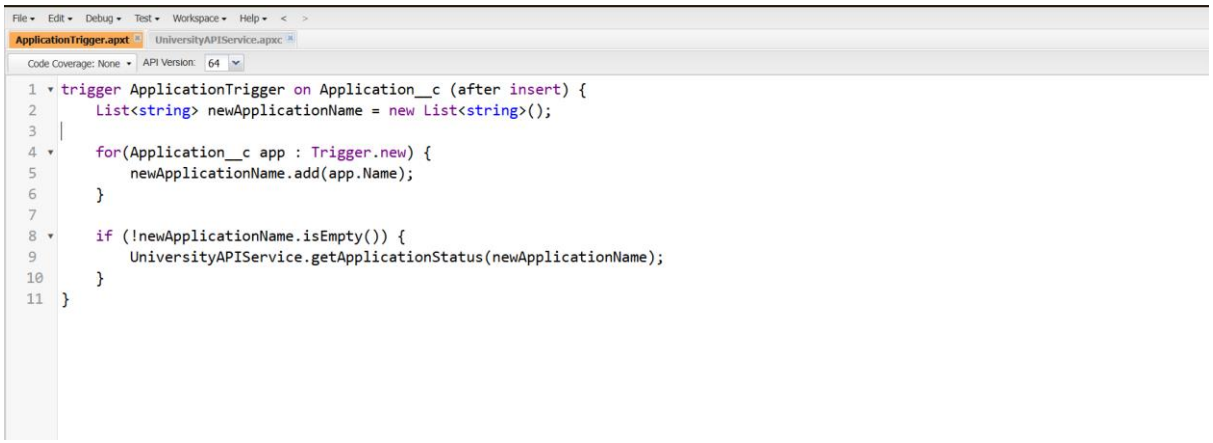
- Created a MentorshipService class that contains reusable logic to validate mentorship sessions.
- This ensures mentors are not double-booked for overlapping sessions.



2. Apex Triggers & Trigger Design Pattern

- Implemented a trigger on Mentorship_Session__c object.
- On insert or update, → system checks if the mentor already has a session scheduled at the same time.
- If overlap is found → block the record and show an error message.
- Used a Trigger Handler class instead of writing logic directly in the trigger.
- This improves the readability, reusability, and maintainability of code.





3. SOQL & SOSL Queries

- Used SOQL queries to fetch all sessions for mentors to detect scheduling conflicts.
- Example: Query all sessions for a mentor on the same date to check time overlaps.

```java
public with sharing class MentorAssignmentService {

    @InvocableMethod(label='Find Best Available Mentor' description='Finds the mentor with the fewest active students.')
    public static List<Id> findBestMentor(List<String> recordTypes) {
        AggregateResult[] mentorWorkload = [
            SELECT Assigned_Mentor__c, COUNT(Id) studentCount
            FROM Contact
            WHERE RecordType.Name = 'Student' AND Assigned_Mentor__c != NULL
            GROUP BY Assigned_Mentor__c
            ORDER BY COUNT(Id) ASC
            LIMIT 1
        ];

        if (!mentorWorkload.isEmpty()) {
            Id bestMentorId = (Id)mentorWorkload[0].get('Mentor__c');
            return new List<Id>{bestMentorId};
        }
        return new List<Id>{null};
    }
}
```

4. Control Statements

- Implemented IF conditions to check whether Start_Time__c and End_Time__c overlap with existing mentor sessions.
- If overlap is detected → throw an error.

```java


        for (SObject sObj : scope) {
            Mentorship_Session__c s = (Mentorship_Session__c) sObj;
            // Mark as missed if session end time is in the past and status is 'Scheduled'
            if (s.End__c != null && s.Status__c == 'Scheduled') {
                s.Status__c = 'Missed';
                toUpdate.add(s);
            }
        }

        if (!toUpdate.isEmpty()) {
            update toUpdate;
        }
    }

    global void finish(Database.BatchableContext bc) {
        // optional: notify admins or log
    }
}
```

5. Collections (List, Set, Map)

- The set was used to collect unique Mentor IDs from new sessions.
- The list was used to store potential conflicting sessions.
- The map can later be used to optimise mentor-student assignments.

```java
public with sharing class MentorAssignmentService {

    @InvocableMethod(label='Find Best Available Mentor' description='Finds the mentor with the fewest active students.')
    public static List<Id> findBestMentor(List<String> recordTypes) {
        AggregateResult[] mentorWorkload = [
            SELECT Assigned_Mentor__c, COUNT(Id) studentCount
            FROM Contact
            WHERE RecordType.Name = 'Student' AND Assigned_Mentor__c != NULL
            GROUP BY Assigned_Mentor__c
            ORDER BY COUNT(Id) ASC
            LIMIT 1
        ];

        if (!mentorWorkload.isEmpty()) {
            Id bestMentorId = (Id)mentorWorkload[0].get('Mentor__c');
            return new List<Id>{bestMentorId};
        }
        return new List<Id>{null};
    }
```

6. Batch Apex

- Designed a batch job that runs nightly.
- It marks overdue mentorship sessions (past sessions not marked as Completed).
- Helps administrators track pending follow-ups.

```java
global with sharing class MarkMissedSessionsBatch implements Database.Batchable<SObject>, Database.Stateful {
    global Database.QueryLocator start(Database.BatchableContext bc) {
        // Select scheduled sessions
        String q = 'SELECT Id, End__c, Status__c FROM Mentorship_Session__c WHERE Status__c = \'Scheduled\'';
        return Database.getQueryLocator(q);
    }

    global void execute(Database.BatchableContext bc, List<SObject> scope) {
        List<Mentorship_Session__c> toUpdate = new List<Mentorship_Session__c>();


        for (SObject sObj : scope) {
            Mentorship_Session__c s = (Mentorship_Session__c) sObj;
            // Mark as missed if session end time is in the past and status is 'Scheduled'
            if (s.End__c != null && s.Status__c == 'Scheduled') {
                s.Status__c = 'Missed';
                toUpdate.add(s);
            }
        }
```

7. Queueable Apex

- Implemented logic for bulk discount calculation (adapted for our project as bulk scholarship recommendations).
- Runs asynchronously to avoid hitting governor limits.

File Edit Debug Test Workspace Help < >

MentorMetricsQueueable.apxc *

Code Coverage: None API Version: 64

```
ublic with sharing class MentorMetricsQueueable implements Queueable {

    private List<Id> mentorIds;

    public MentorMetricsQueueable(List<Id> mentorIds) {
        this.mentorIds = mentorIds;
    }

    public void execute(QueueableContext context) {
        if (mentorIds == null || mentorIds.isEmpty()) {
            return;
        }

        List<AggregateResult> aggregateResults = [
            SELECT Mentor__c mentorId, SUM(Duration__c) totalMinutes
            FROM Mentorship_Session__c
            WHERE Mentor__c IN :mentorIds AND Status__c = 'Completed'
            GROUP BY Mentor__c
        ];
```

8. Scheduled Apex

- Created a scheduled class that runs every morning.
- Emails Program Administrators with a list of all mentorship sessions scheduled for today.
- Ensures admins always have visibility.

File Edit Debug Test Workspace Help < >

DailyMentorSessionsEmail.apxc

Code Coverage: None API Version: 64

```
global with sharing class DailyMentorSessionsEmail implements Schedulable {
    global void execute(SchedulableContext sc) {
        // Query all sessions with today's Date (assume you have a Session_Date__c field)
        List<Mentorship_Session__c> today = [
            SELECT Id, Name, Mentor__c, Student__c, Start__c, End__c
            FROM Mentorship_Session__c
            WHERE Date__c = :Datetime.now().Date()
        ];

        String body = 'Today\'s Mentorship Sessions: ' + today.size() + '\n\n';
        for (Mentorship_Session__c s : today) {
            body += (s.Name != null ? s.Name : s.Id)
                + ' | Mentor: ' + String.valueOf(s.Mentor__c)
                + ' | Start: ' + String.valueOf(s.Start__c) // show time safely
                + '\n';
        }

        Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
        mail.setToAddresses(new List<String>{'program-coordinator@example.com'}); // change
        mail.setSubject('Today\'s Mentorship Sessions - ' + Date.today());
```

9. Future Methods

- Not implemented in the project.
- Implemented a future method to call an external API (e.g., external education/insurance service for students).
- Runs asynchronously so that the user doesn't wait for API response.

10. Exception Handling

- Used try-catch blocks to handle errors gracefully.
- Example: If a session overlaps, the user sees a clear error message instead of a system crash.

File Edit Debug Test Workspace Help

ApplicationController.apxc

Code Coverage: None API Version: 64

```
    @AuraEnabled
    public static Id createApplication(Application__c app) {
        if (app == null) {
            throw new AuraHandledException('Application data required');
        }
        try {
            insert app;
            return app.Id;
        } catch (Exception e) {
            throw new AuraHandledException(e.getMessage());
        }
    }

    @AuraEnabled
```

11. Test Classes

- Created MentorshipService_Test class.
- Inserts test data (students, mentors, sessions).
- Validates that:
- Non-conflicting sessions save successfully.
- Conflicting sessions throw the correct error.
- Ensures more than 75% code coverage for deployment.

File Edit Debug Test Workspace Help

MentorMetricsQueueable.apxc | MentorshipService_Test.apxc

Code Coverage: None API Version: 64

```
@isTest
private class MentorshipService_Test {
    @isTest
    static void testSchedulingConflict() {
        // Test data
        Mentor__c mentor = new Mentor__c(Name = 'Test Mentor');
        insert mentor;

        Student__c student = new Student__c(Name = 'Test Student');
        insert student;

        // Create an initial session from 10:00 to 11:00
        Mentorship_Session__c existingSession = new Mentorship_Session__c(
            Mentor__c = mentor.Id,
            Student__c = student.Id,
            Date__c = Date.today(),
            Start__c = 10.0,
            End__c = 11.0,
            Status__c = 'Scheduled'
        );
```

12. Asynchronous Processing

- Combined multiple asynchronous approaches:
- Batch Apex → Nightly updates (overdue sessions).
- Queueable Apex → Bulk discount/scholarship calculations.
- Future Methods → External API calls.
- This makes the application scalable and efficient.