**Phase 5 Report: Apex Programming (Developer)**

**Executive Summary**

Phase 5 transitions the Phishing Awareness Training Tracker project from declarative automation to programmatic development using **Apex**. The goal was to implement a scalable backend that can handle complex business logic beyond the capabilities of declarative tools like Flow. This includes building service classes, applying the trigger design pattern, using SOQL/SOSL for efficient data retrieval, handling large datasets through asynchronous processing, and ensuring reliability through test classes. The outcome is a system that is both **intelligent** and **robust**, capable of running efficiently under heavy usage.

A screenshot of a computer

AI-generated content may be incorrect.

**Classes & Objects**

I created a central service class TrainingService to encapsulate reusable business logic for Assignments and Completions. This separation improves maintainability by keeping triggers lightweight.

**Code Example (TrainingService Class)**

public with sharing class TrainingService {

public static void calculateAndUpdateEmployeeScore(Set<Id> employeeIds) {

if (employeeIds == null || employeeIds.isEmpty()) return;

List<AggregateResult> aggs = [

SELECT Employee\_\_c emp, AVG(Score\_\_c) avgScore

FROM Completion\_\_c

WHERE Employee\_\_c IN :employeeIds

GROUP BY Employee\_\_c

];

Map<Id, Decimal> avgScoreByEmp = new Map<Id, Decimal>();

for (AggregateResult ar : aggs) {

avgScoreByEmp.put((Id)ar.get('emp'), (Decimal)ar.get('avgScore'));

}

List<Employee\_\_c> empsToUpdate = new List<Employee\_\_c>();

for (Id empId : employeeIds) {

Decimal avgScore = avgScoreByEmp.containsKey(empId) ? avgScoreByEmp.get(empId) : 0;

Employee\_\_c e = new Employee\_\_c(Id = empId);

e.Overall\_Training\_Score\_\_c = avgScore;

e.Trainin\_\_c = (avgScore >= 70) ? 'Passed' : (avgScore == 0 ? 'Pending' : 'Needs Retraining');

empsToUpdate.add(e);

}

if (!empsToUpdate.isEmpty()) update empsToUpdate;

}

}

A screenshot of a computer program

AI-generated content may be incorrect.

**Apex Triggers & Trigger Design Pattern**

To follow best practices, I applied a **“one trigger per object”** design pattern. Triggers are logic-less and only delegate work to their respective handler classes.

**Example Trigger: AssignmentTrigger**

trigger AssignmentTrigger on Assignment\_\_c (before insert, before update, after insert, after update) {

new AssignmentTriggerHandler().run();

}

**Example Handler: AssignmentTriggerHandler**

public class AssignmentTriggerHandler {

public void run() {

if (Trigger.isBefore && Trigger.isInsert) handleBeforeInsert();

if (Trigger.isAfter && Trigger.isInsert) handleAfterInsert();

}

private void handleBeforeInsert() {

for (Assignment\_\_c a : (List<Assignment\_\_c>) Trigger.new) {

if (a.Due\_Date\_\_c < a.Assigned\_Date\_\_c) {

a.addError('Due Date cannot be before Assigned Date.');

}

}

}

private void handleAfterInsert() {

Set<Id> empIds = new Set<Id>();

for (Assignment\_\_c a : (List<Assignment\_\_c>) Trigger.new) {

if (a.Employee\_\_c != null) empIds.add(a.Employee\_\_c);

}

if (!empIds.isEmpty()) {

System.enqueueJob(new CalculateEmployeeScoreQueueable(empIds));

}

}

}

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer screen

AI-generated content may be incorrect.

**SOQL & SOSL**

* **SOQL** was used to fetch pending Assignments and calculate scores.
* **SOSL** can be used for keyword-based searches across multiple objects (optional).

**SOQL Example**

List<Assignment\_\_c> pendingAssignments = [

SELECT Id, Name, Status\_\_c, Due\_Date\_\_c

FROM Assignment\_\_c

WHERE Status\_\_c != 'Completed'

ORDER BY Due\_Date\_\_c ASC

];

**Asynchronous Apex Processing**

To handle large volumes and time-dependent tasks, multiple async tools were implemented:

* **Queueable Apex**: For recalculating employee scores.
* **Batch Apex**: Archiving old completions.
* **Scheduled Apex**: Running batch nightly.
* **Future Methods**: Example callouts to external LMS systems.

**Example Queueable Class**

public class CalculateEmployeeScoreQueueable implements Queueable {

private Set<Id> employeeIds;

public CalculateEmployeeScoreQueueable(Set<Id> empIds) {

this.employeeIds = empIds;

}

public void execute(QueueableContext context) {

TrainingService.calculateAndUpdateEmployeeScore(employeeIds);

}

}

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer screen

AI-generated content may be incorrect.

**Exception Handling**

A utility logger (TrainingErrorLogger) was implemented to capture exceptions into a custom object Error\_Log\_\_c.

**Example**

public with sharing class TrainingErrorLogger {

public static void logException(String context, Exception ex) {

Error\_Log\_\_c logRec = new Error\_Log\_\_c(

Name = 'ApexError-' + Datetime.now().getTime(),

Context\_\_c = context,

Message\_\_c = ex.getMessage()

);

insert logRec;

}

}

A screenshot of a computer program

AI-generated content may be incorrect.

**Test Classes**

Unit testing ensured quality and deployment readiness. A test class was built to cover triggers, service classes, and async processes.

**Example Test Class**

@isTest

private class TrainingIntegrationTest {

@testSetup

static void setupData() {

Employee\_\_c emp = new Employee\_\_c();

insert emp;

Assignment\_\_c a = new Assignment\_\_c(

Name='Phishing Module 1',

Employee\_\_c=emp.Id,

Assigned\_Date\_\_c=Date.today(),

Due\_Date\_\_c=Date.today().addDays(5),

Status\_\_c='Assigned'

);

insert a;

}

@isTest

static void testAssignmentValidation() {

Assignment\_\_c bad = new Assignment\_\_c(

Name='Invalid',

Employee\_\_c=[SELECT Id FROM Employee\_\_c LIMIT 1].Id,

Assigned\_Date\_\_c=Date.today(),

Due\_Date\_\_c=Date.today().addDays(-1)

);

Database.SaveResult sr = Database.insert(bad, false);

System.assert(!sr.isSuccess(), 'Should not allow invalid due date.');

}

}

A screenshot of a computer program

AI-generated content may be incorrect.

**Conclusion**

Phase 5 successfully integrated Apex programming into the Phishing Awareness Training Tracker. By using best-practice patterns such as **logic-less triggers**, **service classes**, and **async processing**, the system became scalable and efficient. Comprehensive unit testing ensured high reliability and deployment readiness, achieving over **75% test coverage**.