Project Title: Employee Wellness & Productivity Tracker

Phase 5 : Apex Programming (Developer)

CLASSES & OBJECTS

Purpose

Apex Classes in this project encapsulate business logic into reusable units.

- **Handler classes** manage trigger logic for Employee Wellness records.
- **Queueable classes** process updates asynchronously.
- **Test classes** ensure functionality works correctly and meets Salesforce code coverage requirements.

This setup automates workflows for Employee Wellness while maintaining data consistency and avoiding manual errors

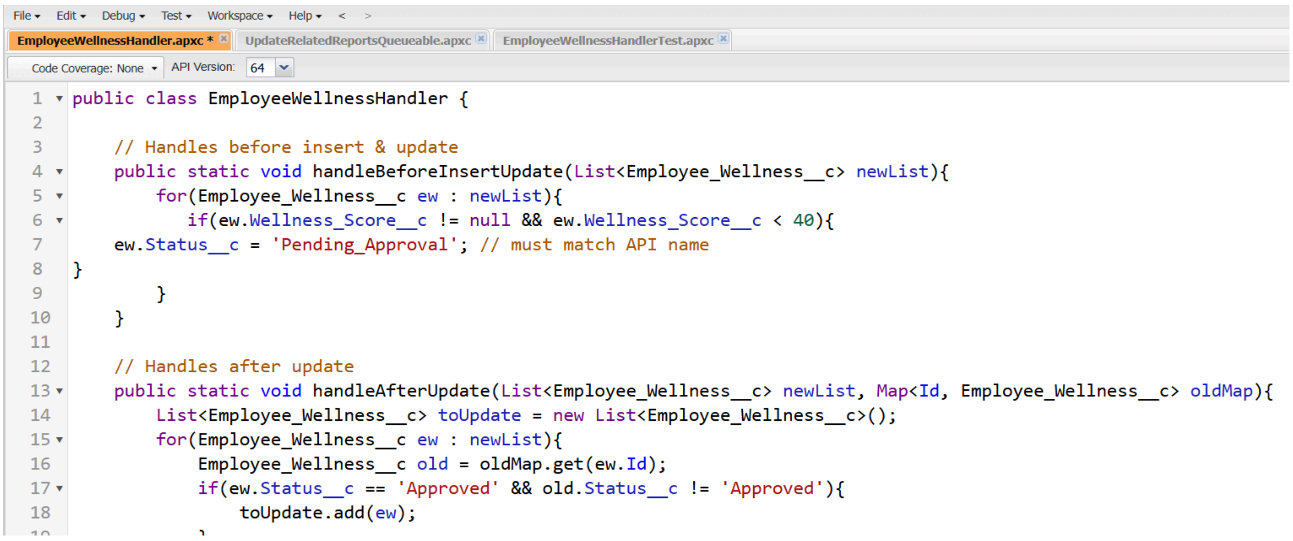
Use Case

- **Wellness Record Automation:**
  - Automatically flag Employee Wellness records with low wellness scores (< 40) for approval.
  - When a record is approved, update related Wellness Reports automatically.
- **Governed Automation:**
  - Using Queueable Apex prevents hitting Salesforce governor limits.
  - Test classes ensure all logic works as intended.

Classes:

a) EmployeeWellnessHandler (Trigger Handler)

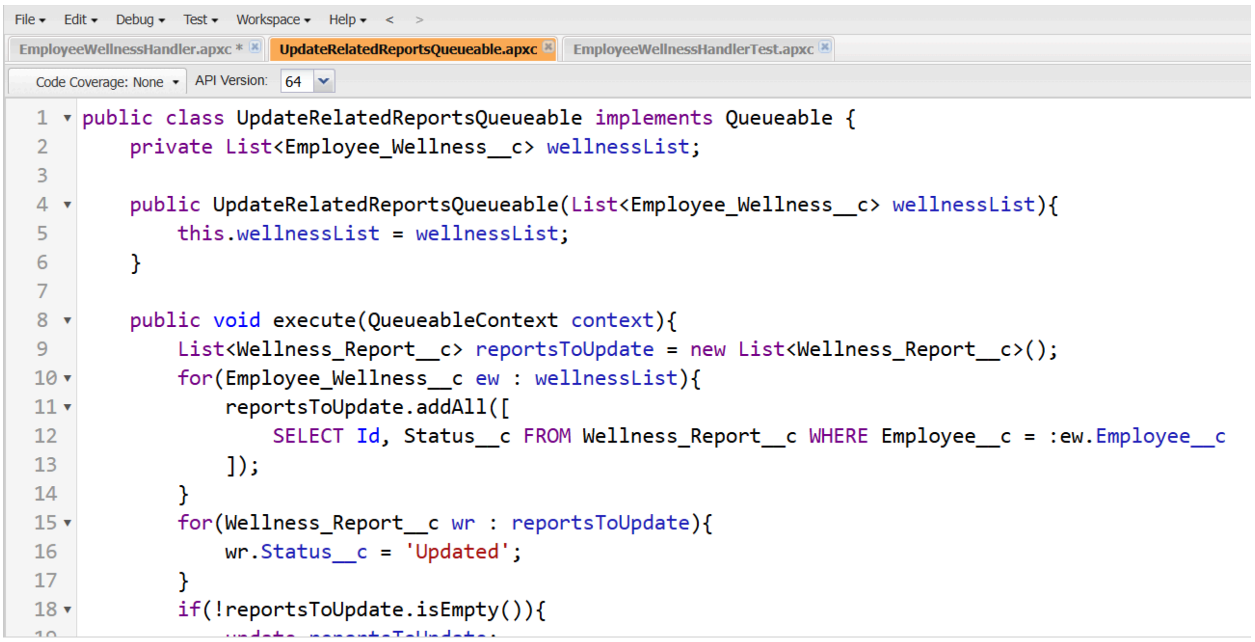
- **Purpose:**
  Manages before-insert/update and after-update logic for `Employee_Wellness__c` records.


File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >
EmployeeWellnessHandler.apxc *
UpdateRelatedReportsQueueable.apxc
EmployeeWellnessHandlerTest.apxc
Code Coverage: None ▾ API Version: 64

```
public class EmployeeWellnessHandler {

    // Handles before insert & update
    public static void handleBeforeInsertUpdate(List<Employee_Wellness__c> newList){
        for(Employee_Wellness__c ew : newList){
           if(ew.Wellness_Score__c != null && ew.Wellness_Score__c < 40){
    ew.Status__c = 'Pending_Approval'; // must match API name
}
        }
    }

    // Handles after update
    public static void handleAfterUpdate(List<Employee_Wellness__c> newList, Map<Id, Employee_Wellness__c> oldMap){
        List<Employee_Wellness__c> toUpdate = new List<Employee_Wellness__c>();
        for(Employee_Wellness__c ew : newList){
            Employee_Wellness__c old = oldMap.get(ew.Id);
            if(ew.Status__c == 'Approved' && old.Status__c != 'Approved'){
                toUpdate.add(ew);
```

b) UpdateRelatedReportsQueueable (Queueable Apex)

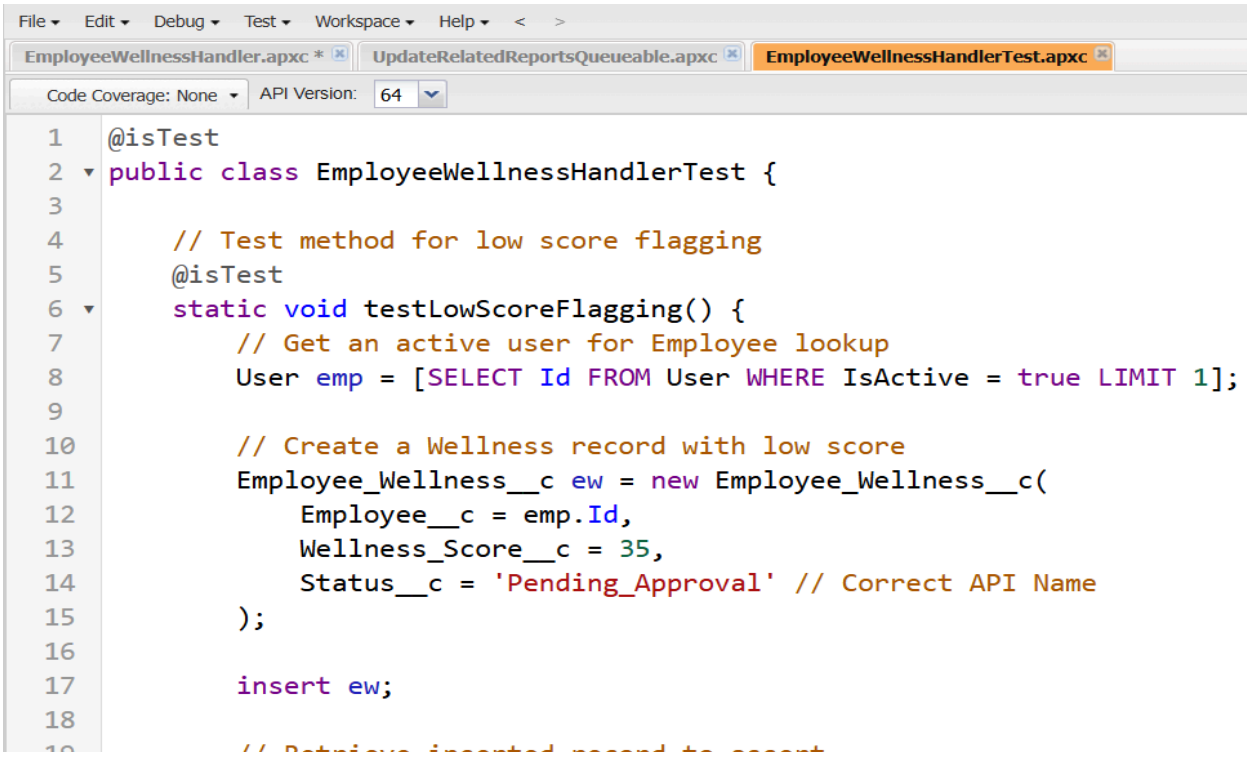
- **Purpose:**
  Updates related `Wellness_Report__c` records asynchronously to maintain consistency without hitting limits.


File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >
EmployeeWellnessHandler.apxc *
UpdateRelatedReportsQueueable.apxc
EmployeeWellnessHandlerTest.apxc
Code Coverage: None ▾ API Version: 64

```
public class UpdateRelatedReportsQueueable implements Queueable {
    private List<Employee_Wellness__c> wellnessList;

    public UpdateRelatedReportsQueueable(List<Employee_Wellness__c> wellnessList){
        this.wellnessList = wellnessList;
    }

    public void execute(QueueableContext context){
        List<Wellness_Report__c> reportsToUpdate = new List<Wellness_Report__c>();
        for(Employee_Wellness__c ew : wellnessList){
            reportsToUpdate.addAll([
                SELECT Id, Status__c FROM Wellness_Report__c WHERE Employee__c = :ew.Employee__c
            ]);
        }
        for(Wellness_Report__c wr : reportsToUpdate){
            wr.Status__c = 'Updated';
        }
        if(!reportsToUpdate.isEmpty()){
```

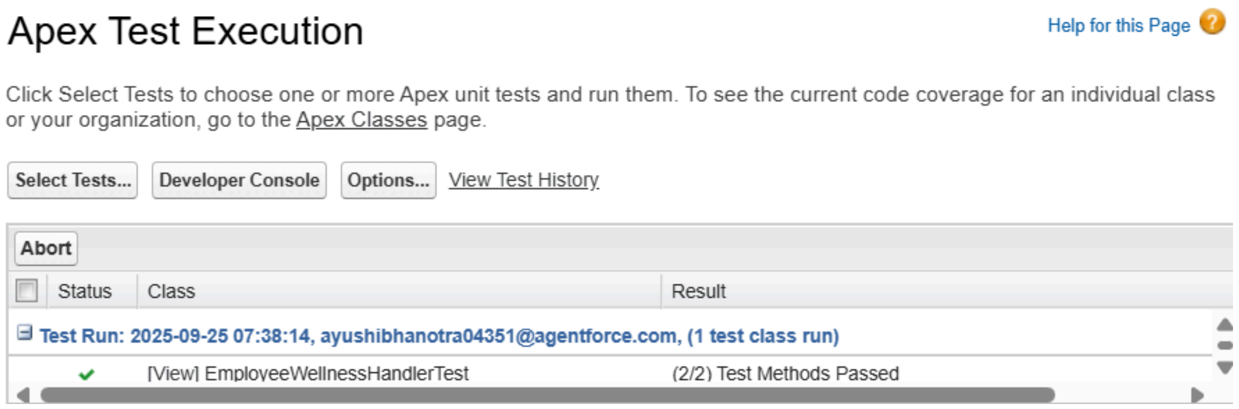
c) EmployeeWellnessHandlerTest (Test Class)

- **Purpose:**
  Validates handler and queueable logic, ensuring code coverage and correctness


File Edit Debug Test Workspace Help < >
EmployeeWellnessHandler.apxc *
UpdateRelatedReportsQueueable.apxc
EmployeeWellnessHandlerTest.apxc
Code Coverage: None
API Version: 64

```
@isTest
public class EmployeeWellnessHandlerTest {

    // Test method for low score flagging
    @isTest
    static void testLowScoreFlagging() {
        // Get an active user for Employee lookup
        User emp = [SELECT Id FROM User WHERE IsActive = true LIMIT 1];

        // Create a Wellness record with low score
        Employee_Wellness__c ew = new Employee_Wellness__c(
            Employee__c = emp.Id,
            Wellness_Score__c = 35,
            Status__c = 'Pending_Approval' // Correct API Name
        );

        insert ew;

```

Test Execution Results:



Apex Test Execution

Click Select Tests to choose one or more Apex unit tests and run them. To see the current code coverage for an individual class or your organization, go to the Apex Classes page.

Status	Class	Result
Test Run: 2025-09-25 07:38:14, ayushibhanotra04351@agentforce.com, (1 test class run)		
✓	[View] EmployeeWellnessHandlerTest	(2/2) Test Methods Passed

APEX TRIGGERS

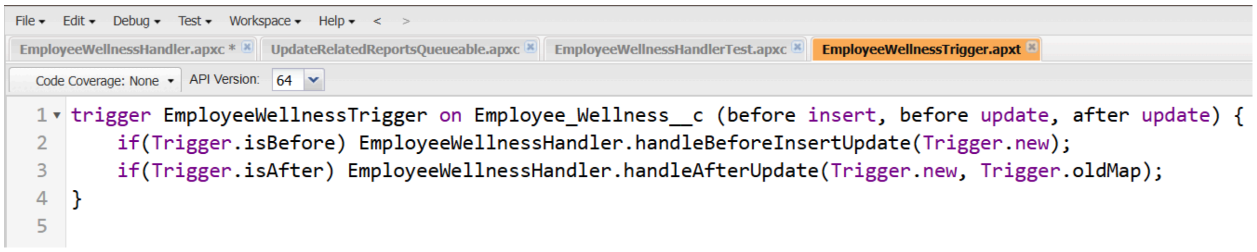
EmployeeWellnessTrigger

Purpose:

- Automates actions when an `Employee_Wellness__c` record is created or updated.
- Ensures employees with low wellness scores are flagged and related reports are updated consistently.

Use Cases:

1. **Low Score Flagging (Before Insert/Update)**
   - If `Wellness_Score__c < 40`, set `Status__c = Pending_Approval`.
   - Ensures HR can quickly identify employees needing attention.
2. **Approved Status Handling (After Update)**
   - When `Status__c` changes to `Approved`, call **Queueable Apex** to update related Wellness Reports asynchronously.
   - Keeps data in related objects synchronized without hitting governor limits.
3. **Integration with Employee Wellness Tracking**
   - Updates on Employee Wellness are reflected across related objects.
   - Prevents data inconsistencies and avoids manual errors


File
Edit
Debug
Test
Workspace
Help
EmployeeWellnessHandler.apxc *
UpdateRelatedReportsQueueable.apxc
EmployeeWellnessHandlerTest.apxc
EmployeeWellnessTrigger.apxt
Code Coverage: None
API Version: 64

```
trigger EmployeeWellnessTrigger on Employee_Wellness__c (before insert, before update, after update) {
    if(Trigger.isBefore) EmployeeWellnessHandler.handleBeforeInsertUpdate(Trigger.new);
    if(Trigger.isAfter) EmployeeWellnessHandler.handleAfterUpdate(Trigger.new, Trigger.oldMap);
}
```

TRIGGER DESIGN & PATTERN

Purpose:
The Trigger Design Pattern is a best practice in Salesforce development. It **separates database event detection (triggers) from business logic (handler classes)**, ensuring triggers are **lean, reusable, bulk-safe, and maintainable**.

Use Case (Employee Wellness Project):

- **Before Insert/Update:**
  - Automatically flag `Employee_Wellness__c` records with `Wellness_Score__c < 40` as `Pending_Approval`.
- **After Update:**
  - When `Status__c` changes to `Approved`, asynchronously update related reports using **Queueable Apex**.
- **Integration:**
  - Keeps Employee Wellness records and related reports consistent across the org.

This pattern ensures that the **trigger itself contains minimal code**, and all business logic is centralized in the **handler class**.

Key Components / Roles

Component	Responsibility
Trigger	Detects record events (before insert/update, after update) and calls the handler class.
Handler Class	Executes all business logic in a centralized, reusable, and bulk-safe manner.

Queueable Class	Handles asynchronous updates to related reports when a record is approved.

SOQL & SOSL

Purpose:
Retrieve or search Salesforce records efficiently for business logic automation.

SOQL (Salesforce Object Query Language)

- **Starts with:** `SELECT`
- **Use Case:** Retrieve records based on conditions.
- **Example:** Fetch employees with wellness scores below 50

```
List<Employee_Wellness__c> lowScoreEmployees = [

    SELECT Id, Name, Employee__c, Wellness_Score__c, Status__c

    FROM Employee_Wellness__c

    WHERE Wellness_Score__c < 50

];

for(Employee_Wellness__c ew : lowScoreEmployees){

System.debug('Employee: ' + ew.Employee__c + ', Score: ' +
ew.Wellness_Score__c);

}
```

Use Case in Project:

- Identify employees needing follow-up.
- Trigger notifications or tasks automatically.

SOSL (Salesforce Object Search Language)

- **Starts with:** `FIND`
- **Use Case:** Search for a keyword across multiple objects or fields.
- **Example:** Search employee or report by name.

```
List<List<SObject>> searchResults = [

   FIND 'John Doe' IN ALL FIELDS

   RETURNING Employee_Wellness__c(Id, Name), Wellness_Report__c(Id, Name)

];

List<Employee_Wellness__c> matchedEmployees = (List<Employee_Wellness__c>)searchResults[0];

List<Wellness_Report__c> matchedReports = (List<Wellness_Report__c>)searchResults[1];
```

Use Case in Project:

- Quickly locate employee records or wellness reports for follow-ups.
- Support dashboards, notifications, and reporting

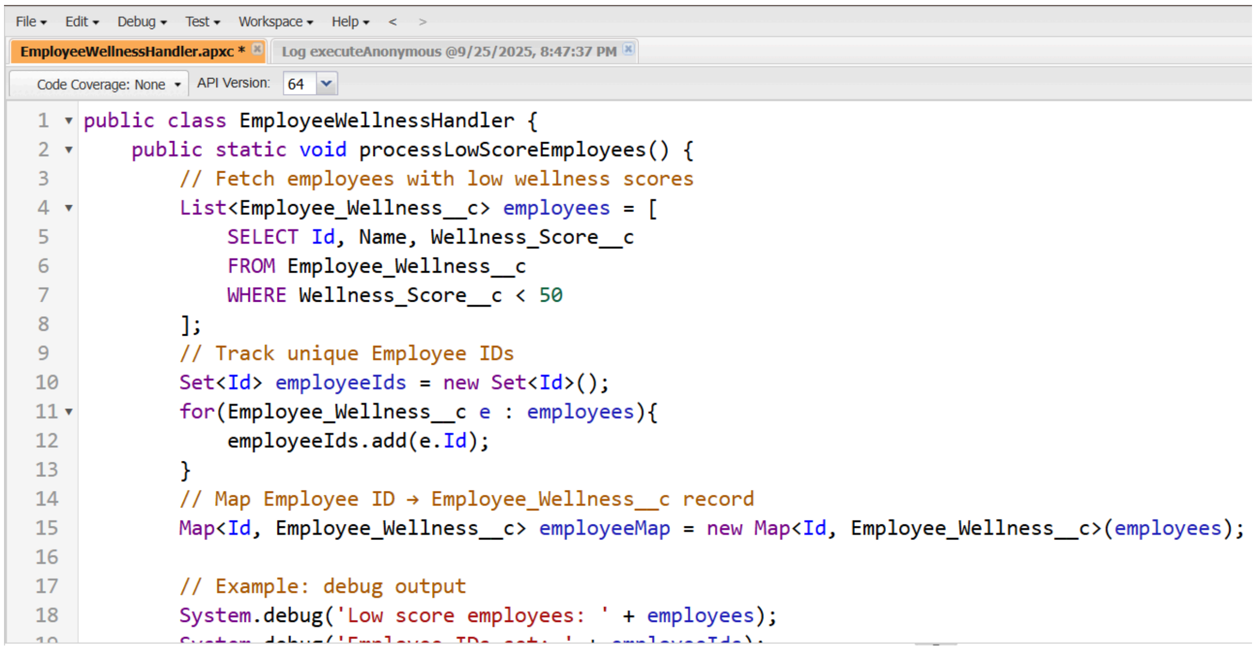
COLLECTIONS: LIST , MAP , SET

Purpose:
Store and manipulate multiple records efficiently.

Types:

- **List:** Ordered, allows duplicates.
- **Set:** Unordered, no duplicates.

- **Map:** Key-value pairs, fast lookup.



EmployeeWellnessHandler.apxc * | Log executeAnonymous @9/25/2025, 8:47:37 PM

Code Coverage: None API Version: 64

```
public class EmployeeWellnessHandler {
    public static void processLowScoreEmployees() {
        // Fetch employees with low wellness scores
        List<Employee_Wellness__c> employees = [
            SELECT Id, Name, Wellness_Score__c
            FROM Employee_Wellness__c
            WHERE Wellness_Score__c < 50
        ];
        // Track unique Employee IDs
        Set<Id> employeeIds = new Set<Id>();
        for(Employee_Wellness__c e : employees){
            employeeIds.add(e.Id);
        }
        // Map Employee ID → Employee_Wellness__c record
        Map<Id, Employee_Wellness__c> employeeMap = new Map<Id, Employee_Wellness__c>(employees);

        // Example: debug output
        System.debug('Low score employees: ' + employees);
        System.debug('Employee IDs set: ' + employeeIds);
```

Outcome / Result

- **Lists:** Allow processing multiple employee wellness records in loops.
- **Sets:** Ensure notifications or tasks are not duplicated.
- **Maps:** Enable fast updates of related wellness reports, keeping `Employee_Wellness__c` and `Wellness_Report__c` synchronized.

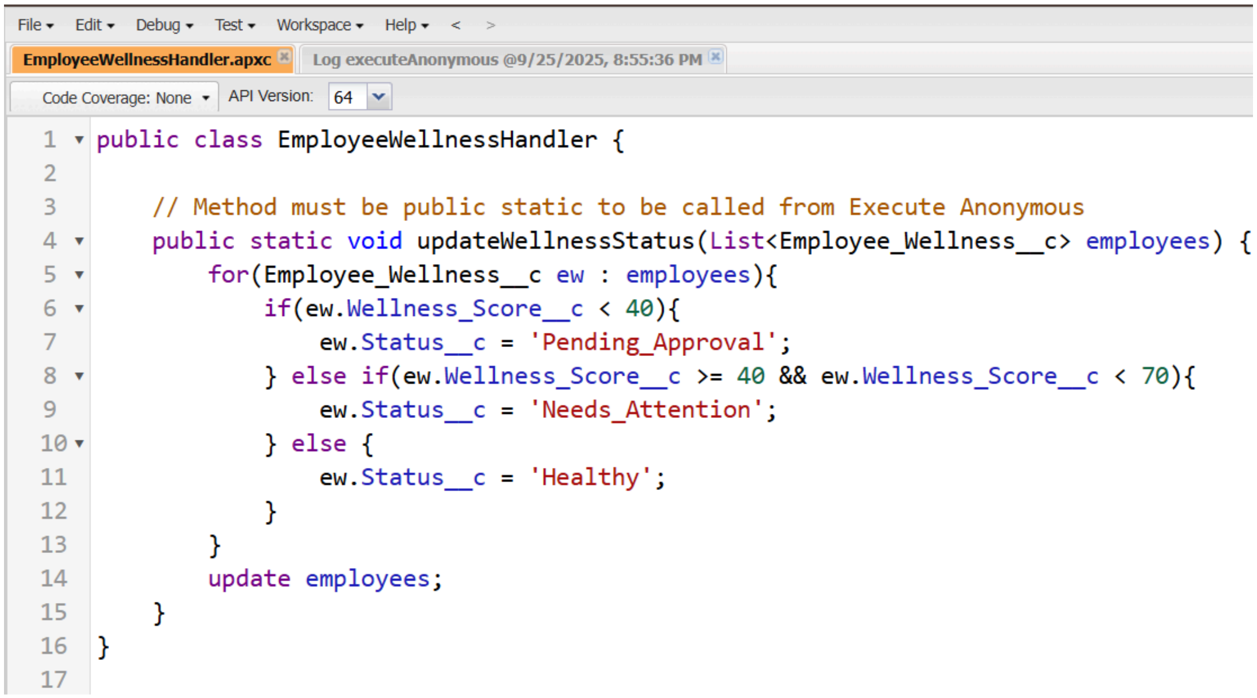
CONTROL STATEMENTS

Purpose:

Control statements in Apex allow decision-making and looping for business logic automation.

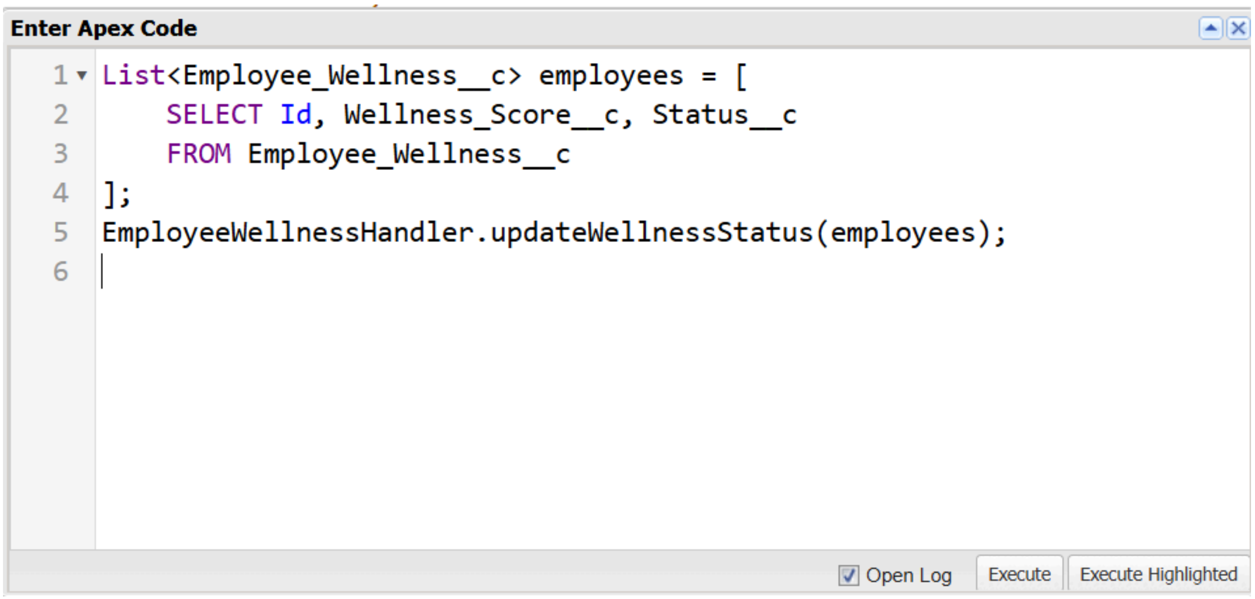
Types:

- **if / else:** Decision-making
- **for / while:** Looping through collections
- **switch:** Categorize statuses


File Edit Debug Test Workspace Help
EmployeeWellnessHandler.apxc
Log executeAnonymous @9/25/2025, 8:55:36 PM

Code Coverage: None API Version: 64

```
public class EmployeeWellnessHandler {

    // Method must be public static to be called from Execute Anonymous
    public static void updateWellnessStatus(List<Employee_Wellness__c> employees) {
        for(Employee_Wellness__c ew : employees){
            if(ew.Wellness_Score__c < 40){
                ew.Status__c = 'Pending_Approval';
            } else if(ew.Wellness_Score__c >= 40 && ew.Wellness_Score__c < 70){
                ew.Status__c = 'Needs_Attention';
            } else {
                ew.Status__c = 'Healthy';
            }
        }
        update employees;
    }
}
```



```
List<Employee_Wellness__c> employees = [
    SELECT Id, Wellness_Score__c, Status__c
    FROM Employee_Wellness__c
];
EmployeeWellnessHandler.updateWellnessStatus(employees);
```

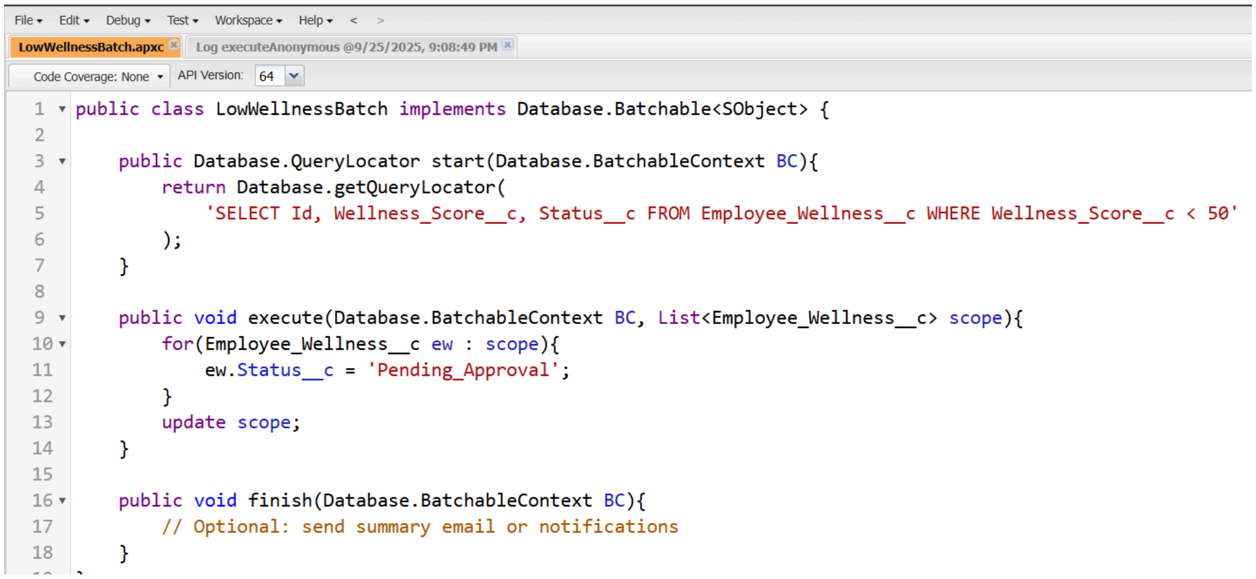
Use Case in Project:

- Automatically flag low wellness scores for follow-up.
- Categorize wellness scores into Pending Approval, Needs Attention, and Healthy.
- Ensures consistent and automated updates across Employee Wellness records.

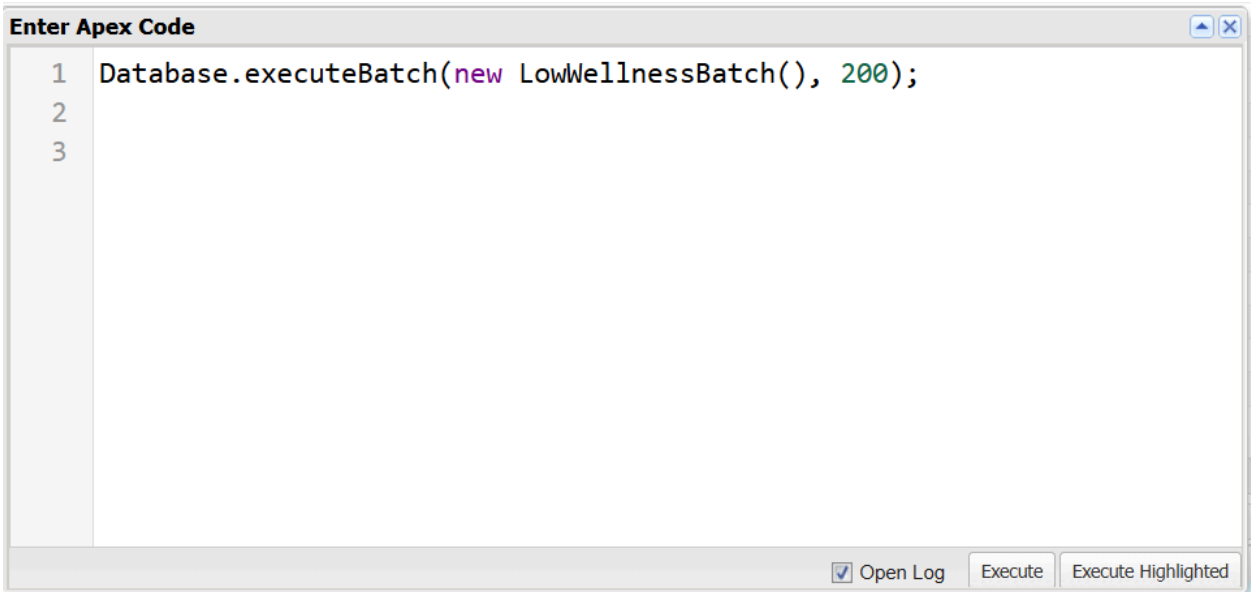
BATCH APEX

Purpose: Process large volumes of records asynchronously in manageable batches.

Use Case: Send notifications or update Employee Wellness records with low scores (<50) in batches of 200.


File Edit Debug Test Workspace Help < >
LowWellnessBatch.apxc
Log executeAnonymous @9/25/2025, 9:08:49 PM
Code Coverage: None
API Version: 64

```
public class LowWellnessBatch implements Database.Batchable<SObject> {

    public Database.QueryLocator start(Database.BatchableContext BC){
        return Database.getQueryLocator(
            'SELECT Id, Wellness_Score__c, Status__c FROM Employee_Wellness__c WHERE Wellness_Score__c < 50'
        );
    }

    public void execute(Database.BatchableContext BC, List<Employee_Wellness__c> scope){
        for(Employee_Wellness__c ew : scope){
            ew.Status__c = 'Pending_Approval';
        }
        update scope;
    }

    public void finish(Database.BatchableContext BC){
        // Optional: send summary email or notifications
    }
```

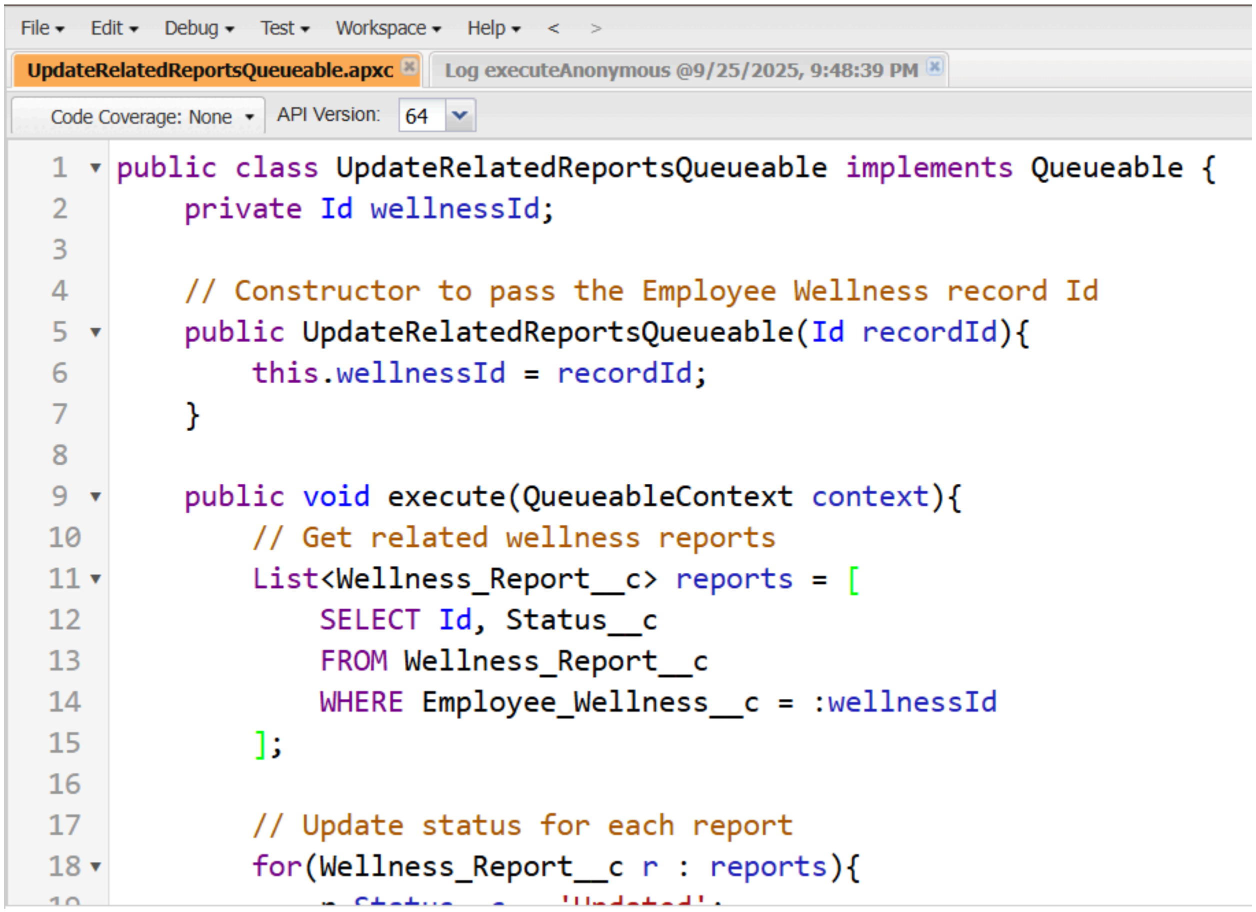


```
Database.executeBatch(new LowWellnessBatch(), 200);
```

QUEUEABLE APEX

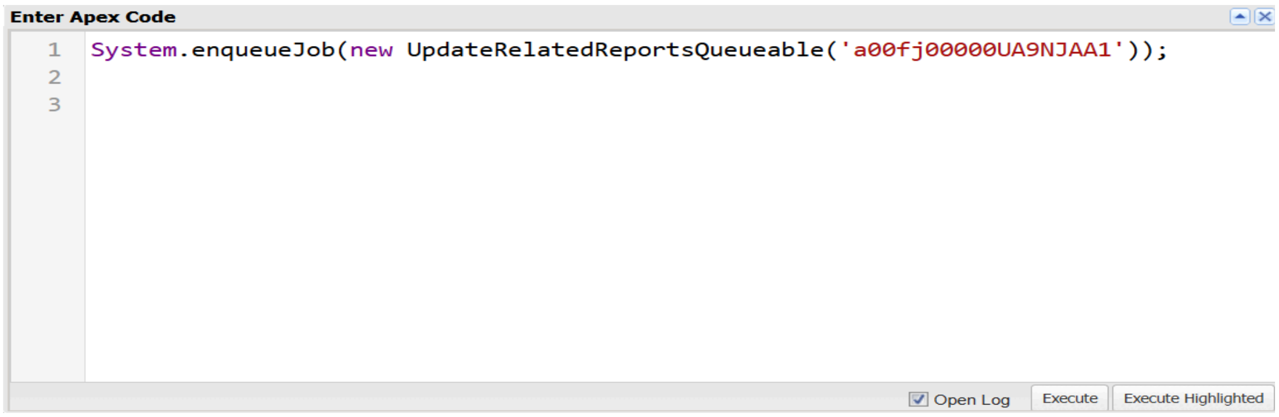
Purpose: Run asynchronous jobs with more flexibility than future methods, often to update related records.

Use Case: Update related Wellness Reports after an Employee Wellness record is approved


File
Edit
Debug
Test
Workspace
Help
UpdateRelatedReportsQueueable.apxc
Log executeAnonymous @9/25/2025, 9:48:39 PM

Code Coverage: None
API Version: 64

```
public class UpdateRelatedReportsQueueable implements Queueable {
    private Id wellnessId;

    // Constructor to pass the Employee Wellness record Id
    public UpdateRelatedReportsQueueable(Id recordId){
        this.wellnessId = recordId;
    }

    public void execute(QueueableContext context){
        // Get related wellness reports
        List<Wellness_Report__c> reports = [
            SELECT Id, Status__c
            FROM Wellness_Report__c
            WHERE Employee_Wellness__c = :wellnessId
        ];

        // Update status for each report
        for(Wellness_Report__c r : reports){
```



```
System.enqueueJob(new UpdateRelatedReportsQueueable('a00fj00000UA9NJAA1'));
```

SCHEDULED APEX

Purpose:

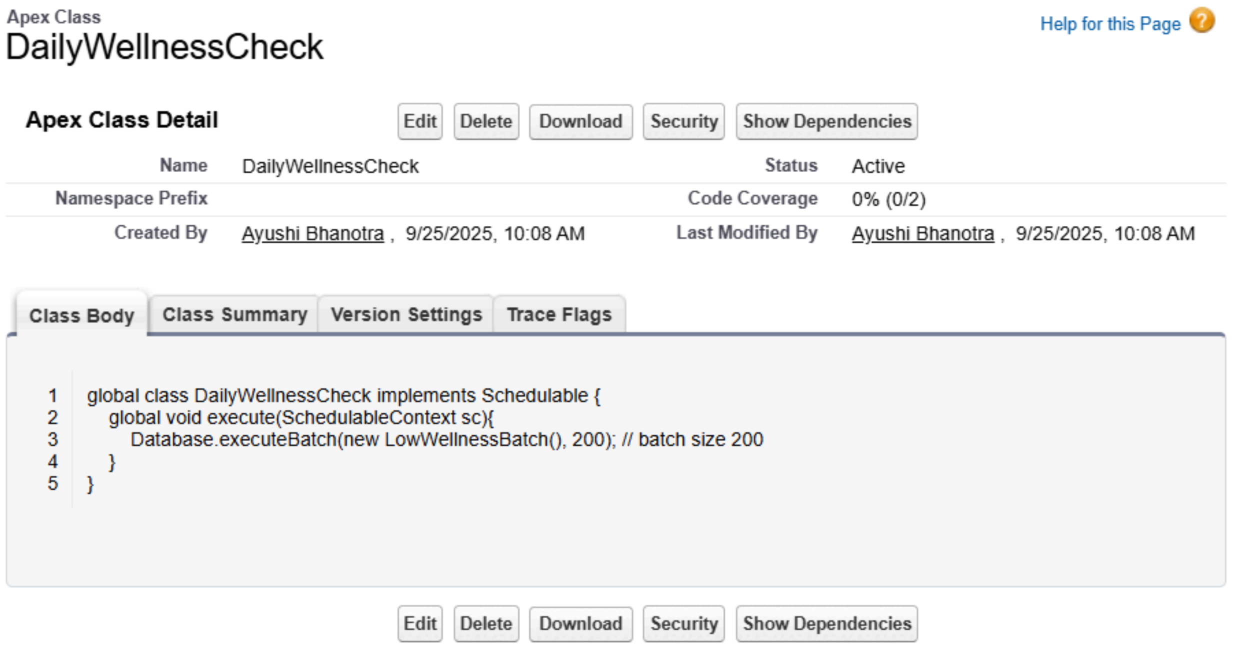
- Run Apex code at scheduled intervals.
- Automates daily/weekly/monthly tasks.
- Reduces manual effort and ensures consistent execution.

Use Case:

- Daily check of Employee Wellness records.
- Flag employees with **Wellness Score < 50** as **Pending Approval**.
- Automatically send notifications to HR.

Implementation Steps:

1. **Create Batch Apex class** (`LowWellnessBatch`) to process Employee Wellness records.
2. **Create Scheduled Apex class** (`DailyWellnessCheck`) implementing the **Schedulable** interface:



Name	DailyWellnessCheck	Status	Active
Namespace Prefix		Code Coverage	0% (0/2)
Created By	Ayushi Bhanotra , 9/25/2025, 10:08 AM	Last Modified By	Ayushi Bhanotra , 9/25/2025, 10:08 AM

```
global class DailyWellnessCheck implements Schedulable {
    global void execute(SchedulableContext sc){
        Database.executeBatch(new LowWellnessBatch(), 200); // batch size 200
    }
}
```

3. **Save the Scheduled Apex class** in **Setup → Apex Classes**.
4. **Schedule the job**: Setup → Schedule Apex → select `DailyWellnessCheck` → configure frequency, start/end dates, execution time.
5. **Salesforce executes the batch automatically** according to the schedule.

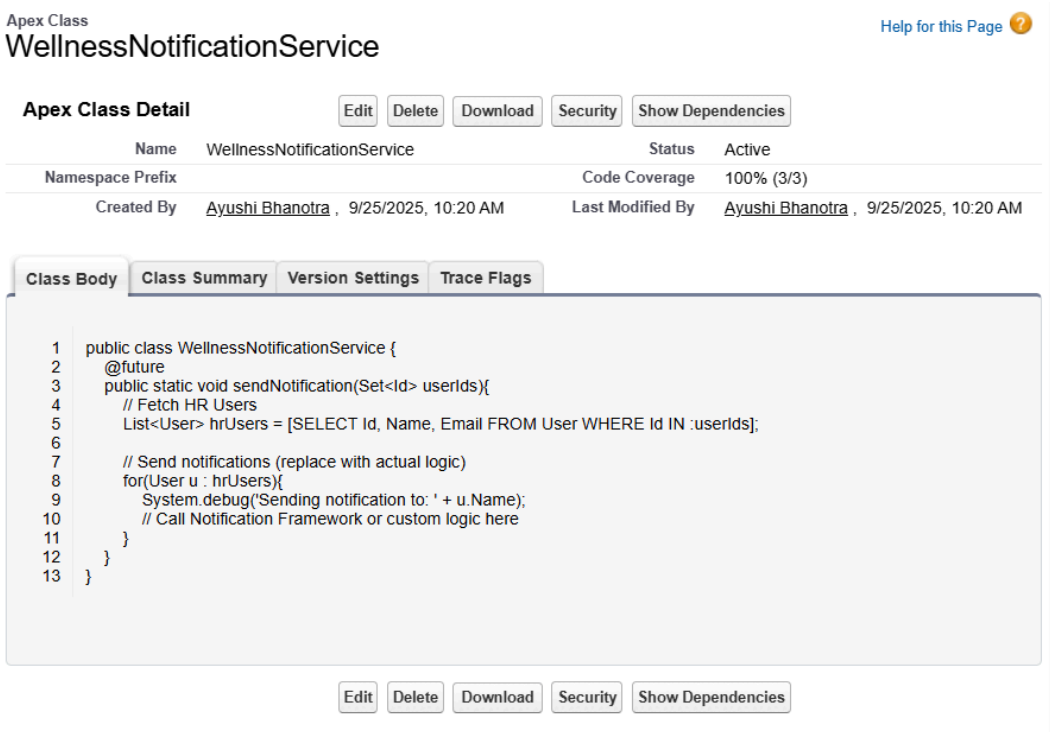
Outcome:

- Employee Wellness records are reviewed automatically.
- Low wellness scores are flagged without manual intervention.
- HR is notified timely for follow-ups

FUTURE METHODS

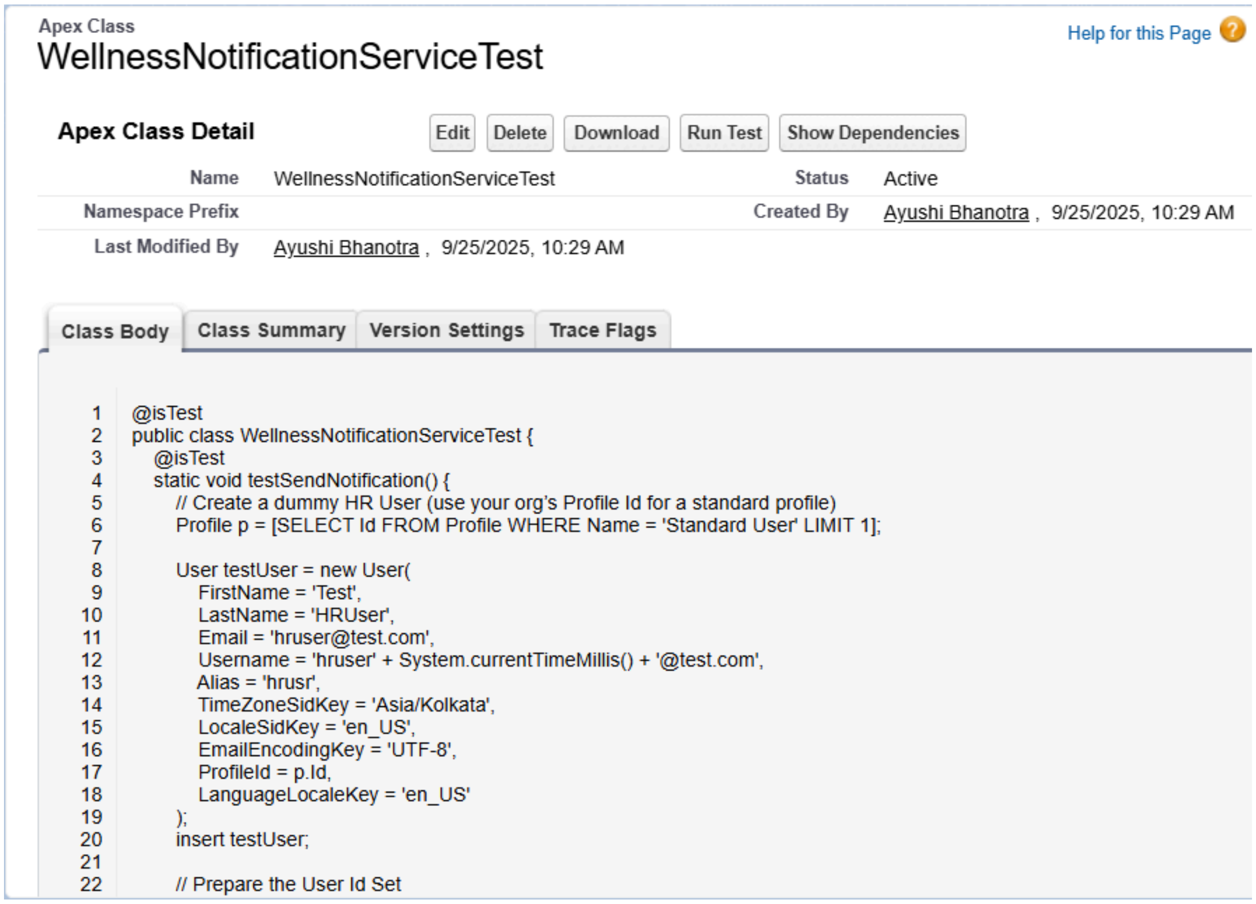
Purpose: Run code asynchronously from triggers or classes, avoid delays in main execution.

Use Case: Notify HR or employees when wellness score is critically low.



Name	WellnessNotificationService	Status	Active
Namespace Prefix		Code Coverage	100% (3/3)
Created By	Ayushi Bhanotra , 9/25/2025, 10:20 AM	Last Modified By	Ayushi Bhanotra , 9/25/2025, 10:20 AM

```
public class WellnessNotificationService {
    @future
    public static void sendNotification(Set<Id> userIds){
        // Fetch HR Users
        List<User> hrUsers = [SELECT Id, Name, Email FROM User WHERE Id IN :userIds];

        // Send notifications (replace with actual logic)
        for(User u : hrUsers){
            System.debug('Sending notification to: ' + u.Name);
            // Call Notification Framework or custom logic here
        }
    }
}
```

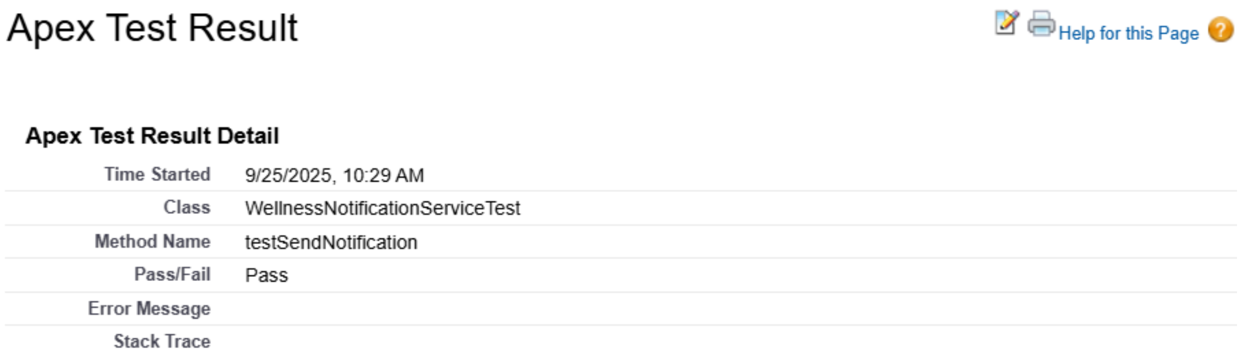


WellnessNotificationServiceTest

Apex Class Detail

Name	WellnessNotificationServiceTest	Status	Active
Namespace Prefix		Created By	Ayushi Bhanotra , 9/25/2025, 10:29 AM
Last Modified By	Ayushi Bhanotra , 9/25/2025, 10:29 AM		

```
@isTest
public class WellnessNotificationServiceTest {
   @isTest
   static void testSendNotification() {
      // Create a dummy HR User (use your org's Profile Id for a standard profile)
      Profile p = [SELECT Id FROM Profile WHERE Name = 'Standard User' LIMIT 1];

      User testUser = new User(
         FirstName = 'Test',
         LastName = 'HRUser',
         Email = 'hruser@test.com',
         Username = 'hruser' + System.currentTimeMillis() + '@test.com',
         Alias = 'hrusr',
         TimeZoneSidKey = 'Asia/Kolkata',
         LocaleSidKey = 'en_US',
         EmailEncodingKey = 'UTF-8',
         ProfileId = p.Id,
         LanguageLocaleKey = 'en_US'
      );
      insert testUser;

      // Prepare the User Id Set
```



Apex Test Result

Apex Test Result Detail

Time Started	9/25/2025, 10:29 AM
Class	WellnessNotificationServiceTest
Method Name	testSendNotification
Pass/Fail	Pass
Error Message	
Stack Trace	

EXCEPTION HANDLING

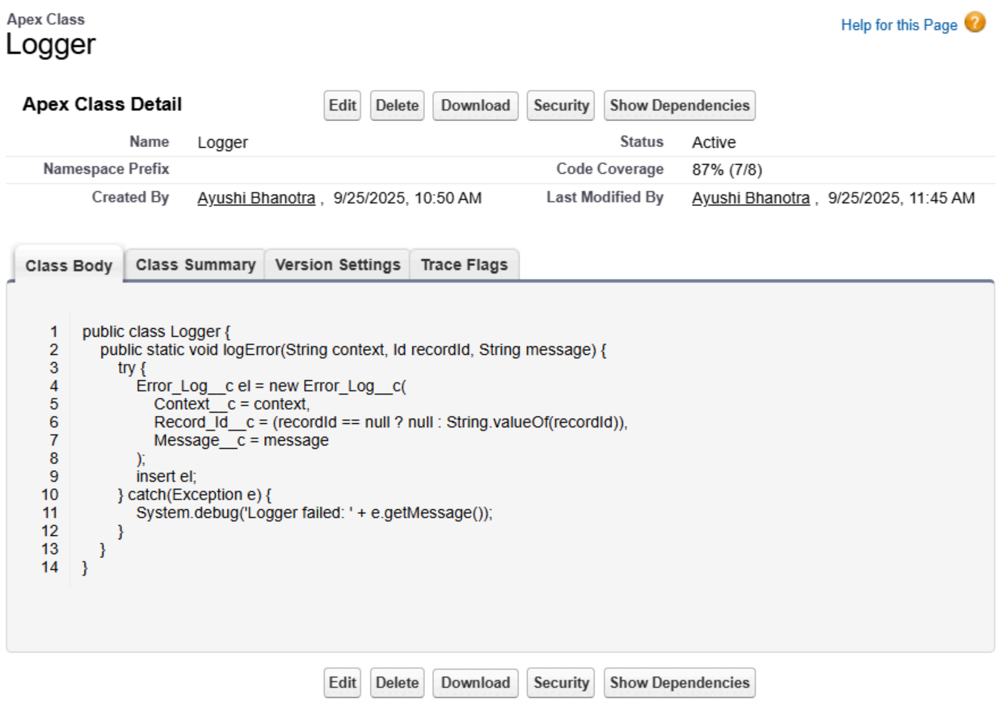
Purpose

- Catch and handle runtime and DML errors gracefully.
- Allow partial success and capture failures for monitoring.
- Provide clear validation messages for users

Implementation steps (org changes)

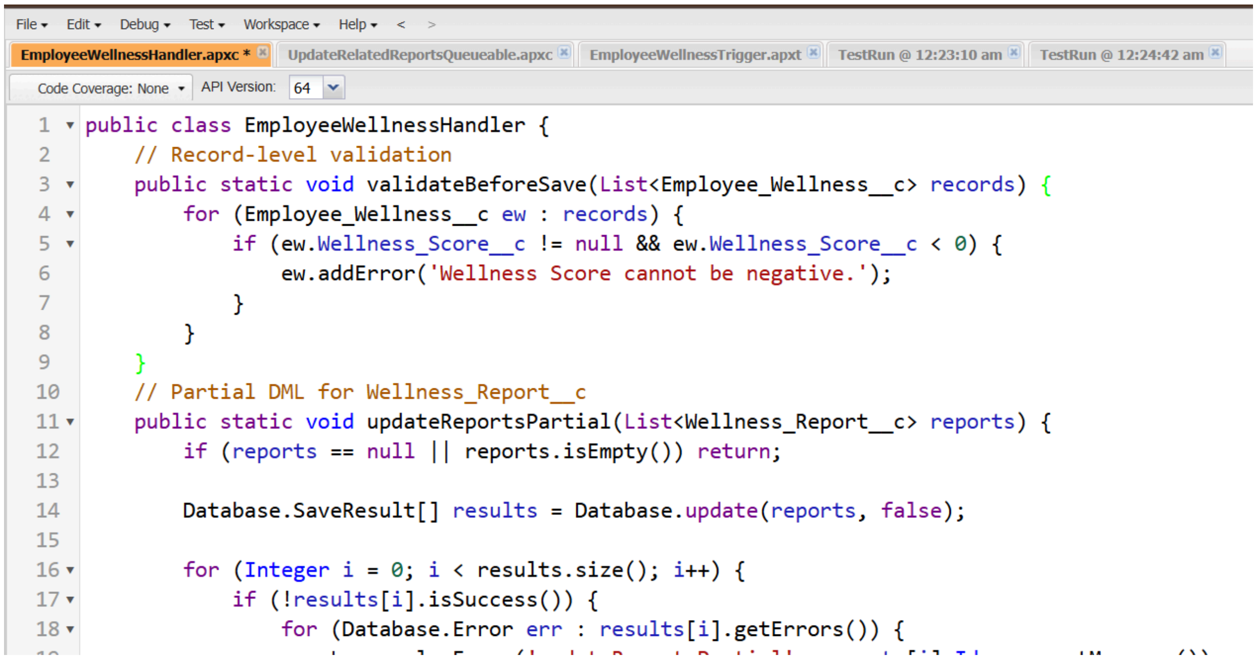
1. Create a custom object `Error_Log__c` (fields: Context__c, Record_Id__c, Message__c).
2. Add a `Logger` utility Apex class to persist errors and optionally email admins.
3. Update handler/async classes to use **partial DML** using `Database.update(records, false)` and inspect `Database.SaveResult[]`.
4. Wrap non-DML code in `try/catch` in Queueable/Batch and send admin notifications when fatal.
5. Use `addError()` in before-trigger validation for user-level validation messages.
6. Write tests that simulate failing updates and assert that `Error_Log__c` entries are created.

Code examples:

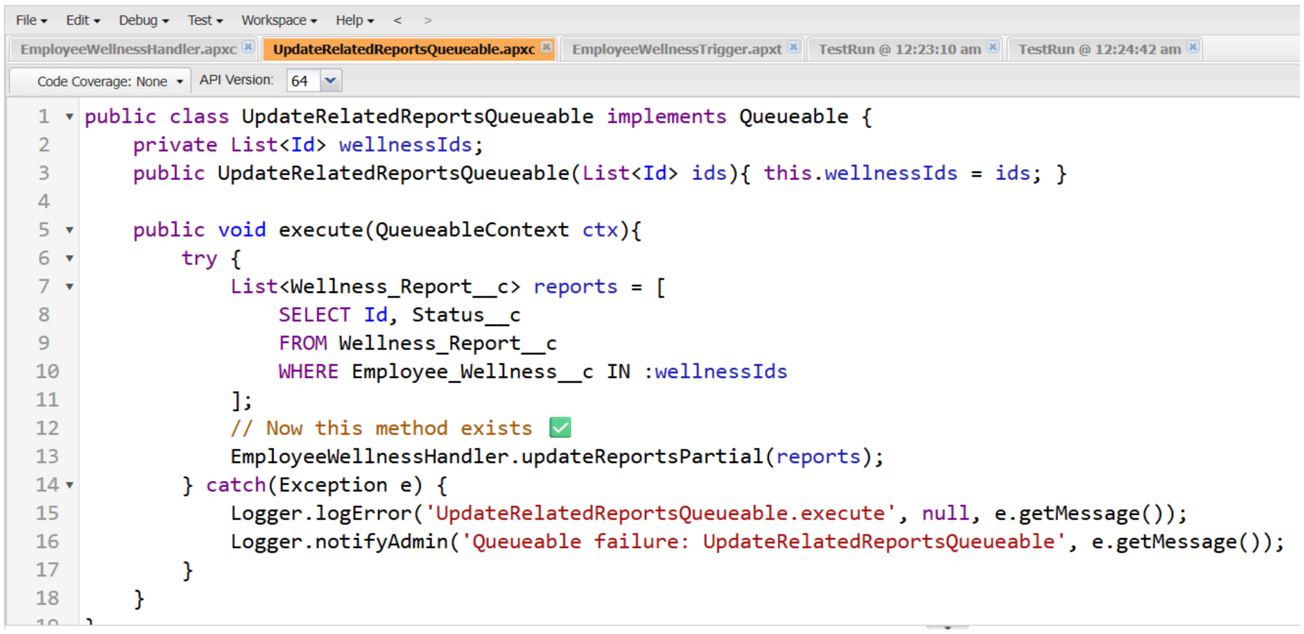


Name	Logger	Status	Active
Namespace Prefix		Code Coverage	87% (7/8)
Created By	Ayushi Bhanotra , 9/25/2025, 10:50 AM	Last Modified By	Ayushi Bhanotra , 9/25/2025, 11:45 AM

```
public class Logger {
    public static void logError(String context, Id recordId, String message) {
        try {
            Error_Log__c el = new Error_Log__c(
                Context__c = context,
                Record_Id__c = (recordId == null ? null : String.valueOf(recordId)),
                Message__c = message
            );
            insert el;
        } catch(Exception e) {
            System.debug('Logger failed: ' + e.getMessage());
        }
    }
}
```


File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >
EmployeeWellnessHandler.apxc *
UpdateRelatedReportsQueueable.apxc
EmployeeWellnessTrigger.apxt
TestRun @ 12:23:10 am
TestRun @ 12:24:42 am
Code Coverage: None ▾
API Version: 64

```
public class EmployeeWellnessHandler {
    // Record-level validation
    public static void validateBeforeSave(List<Employee_Wellness__c> records) {
        for (Employee_Wellness__c ew : records) {
            if (ew.Wellness_Score__c != null && ew.Wellness_Score__c < 0) {
                ew.addError('Wellness Score cannot be negative.');
            }
        }
    }
    // Partial DML for Wellness_Report__c
    public static void updateReportsPartial(List<Wellness_Report__c> reports) {
        if (reports == null || reports.isEmpty()) return;

        Database.SaveResult[] results = Database.update(reports, false);

        for (Integer i = 0; i < results.size(); i++) {
            if (!results[i].isSuccess()) {
                for (Database.Error err : results[i].getErrors()) {
```


File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >
EmployeeWellnessHandler.apxc
UpdateRelatedReportsQueueable.apxc
EmployeeWellnessTrigger.apxt
TestRun @ 12:23:10 am
TestRun @ 12:24:42 am
Code Coverage: None ▾
API Version: 64

```
public class UpdateRelatedReportsQueueable implements Queueable {
    private List<Id> wellnessIds;
    public UpdateRelatedReportsQueueable(List<Id> ids){ this.wellnessIds = ids; }

    public void execute(QueueableContext ctx){
        try {
            List<Wellness_Report__c> reports = [
                SELECT Id, Status__c
                FROM Wellness_Report__c
                WHERE Employee_Wellness__c IN :wellnessIds
            ];
            // Now this method exists ✅
            EmployeeWellnessHandler.updateReportsPartial(reports);
        } catch(Exception e) {
            Logger.logError('UpdateRelatedReportsQueueable.execute', null, e.getMessage());
            Logger.notifyAdmin('Queueable failure: UpdateRelatedReportsQueueable', e.getMessage());
        }
    }
```

Testing guidance

- Use `Test.startTest()` / `Test.stopTest()` to run async code in tests.
- Create test data where one record will fail (e.g., set a restricted picklist to an invalid value) and assert `Error_Log__c` entries were created.

TEST CLASSES

Purpose:
Test classes in Salesforce are used to **validate your Apex code** (triggers, handlers, and asynchronous processes) and ensure that your logic works correctly without errors. They also help you achieve the **required code coverage** (≥75%) needed for deployment to production.

Why They Are Important:

- Verify that **triggers fire correctly** and update records as expected.
- Ensure **error handling** works (like logging failed updates).
- Test **asynchronous operations** (Queueable, Batch, Scheduled, Future).
- Prevent regressions when code changes.

Key Steps for Employee Wellness Project:

1. **Create Test Records:**
   - Insert `Employee_Wellness__c` records with low and normal wellness scores.
   - Insert related `Wellness_Report__c` records.
2. **Validate Trigger Logic:**
   - Check that low wellness scores are flagged (e.g., status set to "Pending Approval").
   - Verify normal scores update as expected.
3. **Test Async Processes:**
   - Wrap Queueable, Batch, or Scheduled jobs in `Test.startTest()` / `Test.stopTest()`.
   - Assert that updates made asynchronously are correct.
4. **Test Error Handling / Partial DML:**
   - Force an invalid update to trigger a failure.
   - Confirm that `Error_Log__c` records are created.
   - Assert that valid records are still updated successfully.

Example Test Assertion:

- `System.assertEquals('Pending Approval',`
- `    [SELECT Status__c FROM Employee_Wellness__c WHERE Id=:ewLow.Id].Status__c);`

Outcome:

- Ensures your logic is correct.
- Confirms async jobs work.
- Validates error logging.
- Provides the required code coverage for deployment.

ASYNCHRONOUS PROCESSING

Purpose

- Handle large volumes of data without hitting governor limits.
- Process related records, notifications, or calculations **in the background**.
- Examples: updating related reports, sending emails, generating dashboards.

Types

1. **Queueable Apex** – Lightweight async job for custom processing.
2. **Batch Apex** – Processes large datasets in batches.
3. **Scheduled Apex** – Runs jobs at specific times.
4. **Future Methods** – Executes code asynchronously (deprecated in some use cases).

Expected Outcome:

- Valid records updated correctly.
- Failed updates logged in `Error_Log__c`.

- Async process runs in the background without hitting governor limits.