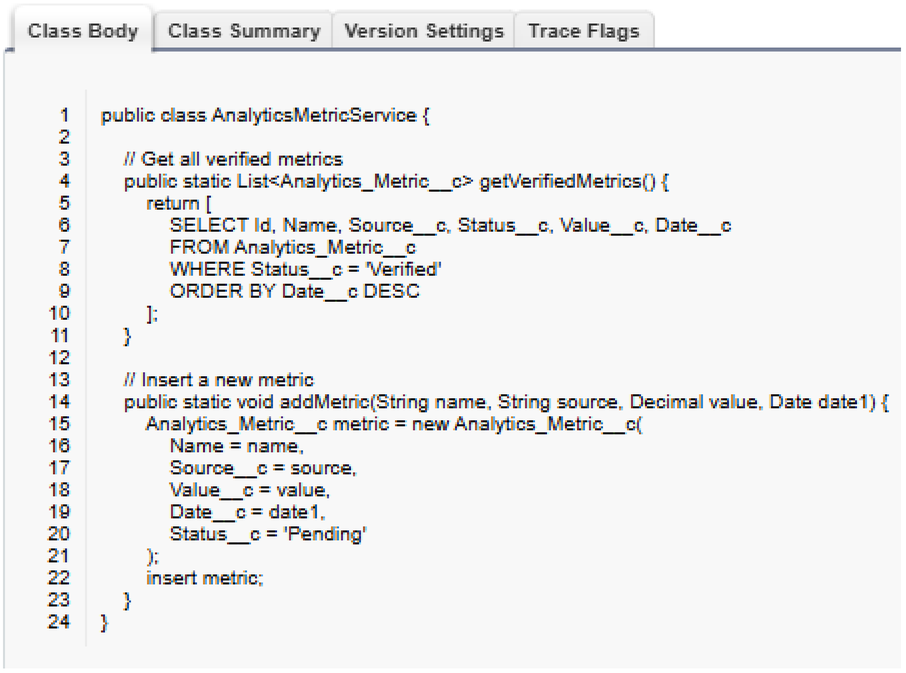
ERM Hybrid Workforce Project – Phase 5

Phase 5: Apex Programming (Developer)

1. Apex Classes & Objects

Custom service, handler, and validator classes were created to encapsulate business logic and enforce modular design.

Created Apex Classes:

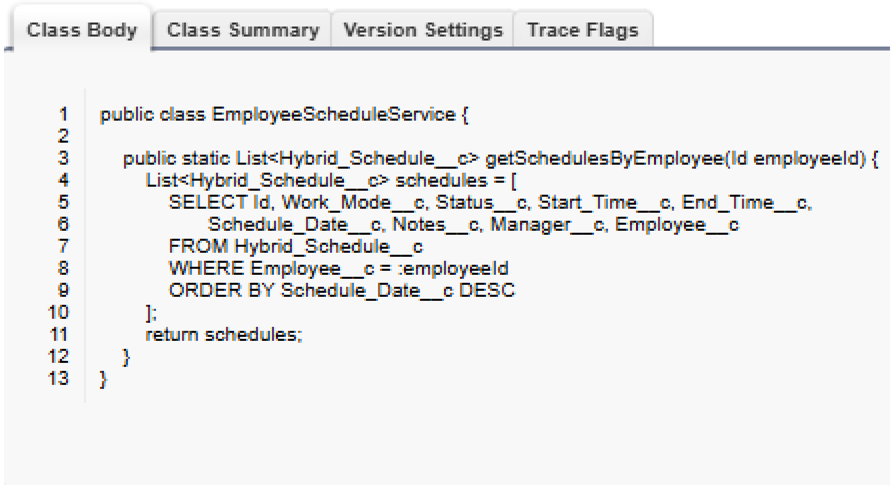


```
public class AnalyticsMetricService {

    // Get all verified metrics
    public static List<Analytics_Metric__c> getVerifiedMetrics() {
        return [
            SELECT Id, Name, Source__c, Status__c, Value__c, Date__c
            FROM Analytics_Metric__c
            WHERE Status__c = 'Verified'
            ORDER BY Date__c DESC
        ];
    }

    // Insert a new metric
    public static void addMetric(String name, String source, Decimal value, Date date1) {
        Analytics_Metric__c metric = new Analytics_Metric__c(
            Name = name,
            Source__c = source,
            Value__c = value,
            Date__c = date1,
            Status__c = 'Pending'
        );
        insert metric;
    }
}
```

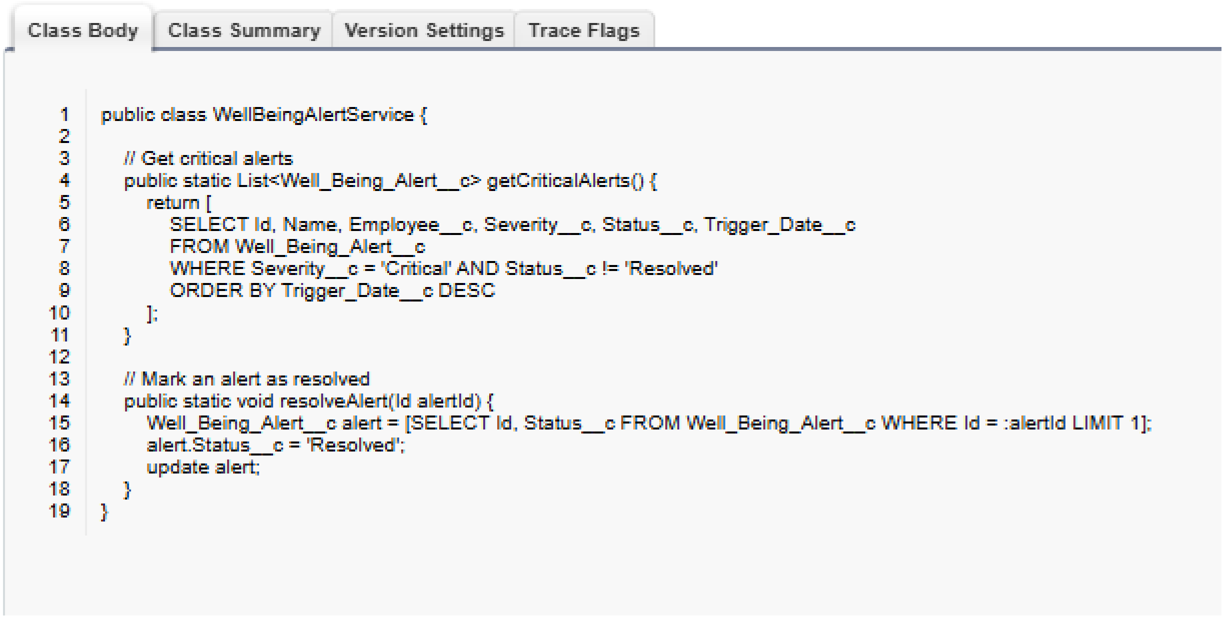
- `AnalyticsMetricService` – Handles metric calculation and analytics logic.
- `EmployeeScheduleService` – Business logic for employee scheduling.


Class Body | Class Summary | Version Settings | Trace Flags

```
public class EmployeeScheduleService {

    public static List<Hybrid_Schedule__c> getSchedulesByEmployee(Id employeeId) {
        List<Hybrid_Schedule__c> schedules = [
            SELECT Id, Work_Mode__c, Status__c, Start_Time__c, End_Time__c,
                   Schedule_Date__c, Notes__c, Manager__c, Employee__c
            FROM Hybrid_Schedule__c
            WHERE Employee__c = :employeeId
            ORDER BY Schedule_Date__c DESC
        ];
        return schedules;
    }
}
```

- `HybridScheduleService` – Handles hybrid schedule operations.
- `PulseSurveyService` – Encapsulates reusable survey-related logic.
- `WellBeingAlertService` – Provides alert generation and monitoring logic.


Class Body | Class Summary | Version Settings | Trace Flags

```
public class WellBeingAlertService {

    // Get critical alerts
    public static List<Well_Being_Alert__c> getCriticalAlerts() {
        return [
            SELECT Id, Name, Employee__c, Severity__c, Status__c, Trigger_Date__c
            FROM Well_Being_Alert__c
            WHERE Severity__c = 'Critical' AND Status__c != 'Resolved'
            ORDER BY Trigger_Date__c DESC
        ];
    }

    // Mark an alert as resolved
    public static void resolveAlert(Id alertId) {
        Well_Being_Alert__c alert = [SELECT Id, Status__c FROM Well_Being_Alert__c WHERE Id = :alertId LIMIT 1];
        alert.Status__c = 'Resolved';
        update alert;
    }
}
```

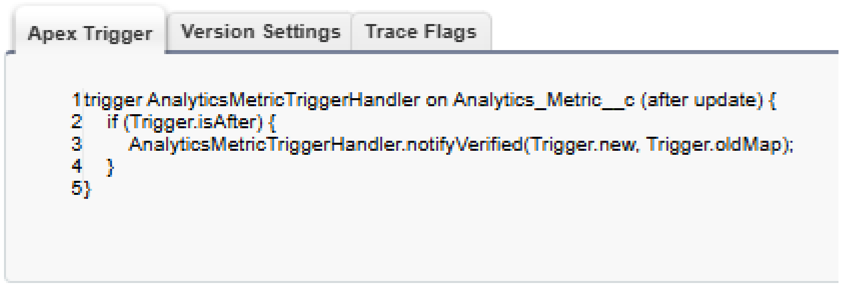
---

2. Apex Triggers

Triggers were implemented using the **Trigger Handler Pattern** to ensure clean code separation.

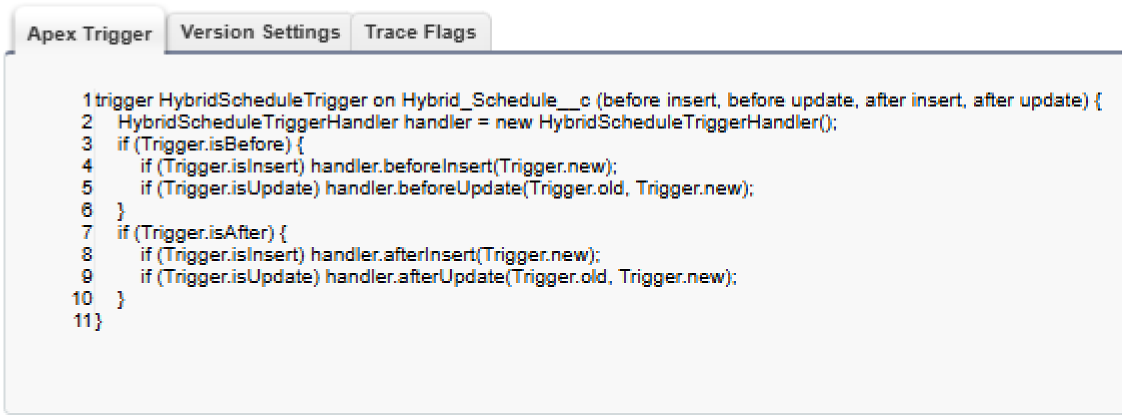
Developed Triggers:

- `AnalyticsMetricTrigger`



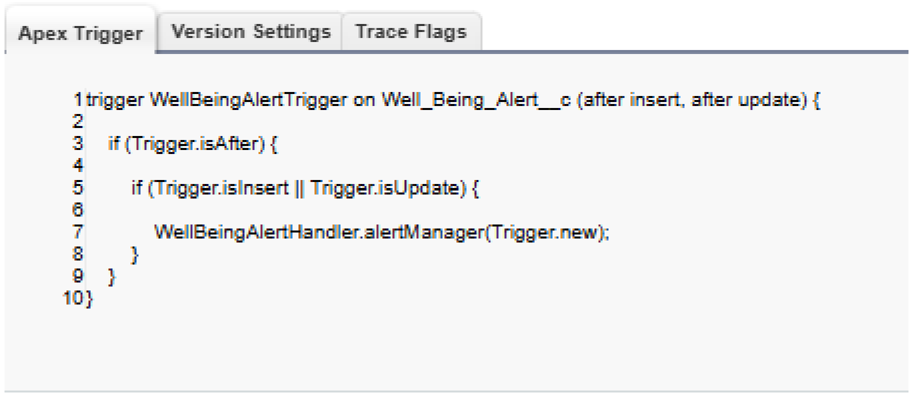
```
trigger AnalyticsMetricTriggerHandler on Analytics_Metric__c (after update) {
  if (Trigger.isAfter) {
     AnalyticsMetricTriggerHandler.notifyVerified(Trigger.new, Trigger.oldMap);
  }
}
```

- `EmployeeStatusTrigger`
- `HybridScheduleTrigger`



```
trigger HybridScheduleTrigger on Hybrid_Schedule__c (before insert, before update, after insert, after update) {
  HybridScheduleTriggerHandler handler = new HybridScheduleTriggerHandler();
  if (Trigger.isBefore) {
     if (Trigger.isInsert) handler.beforeInsert(Trigger.new);
     if (Trigger.isUpdate) handler.beforeUpdate(Trigger.old, Trigger.new);
  }
  if (Trigger.isAfter) {
     if (Trigger.isInsert) handler.afterInsert(Trigger.new);
     if (Trigger.isUpdate) handler.afterUpdate(Trigger.old, Trigger.new);
  }
}
```

- `PulseSurveyTrigger`
- `WellBeingAlertTrigger`



```
trigger WellBeingAlertTrigger on Well_Being_Alert__c (after insert, after update) {

  if (Trigger.isAfter) {

     if (Trigger.isInsert || Trigger.isUpdate) {

        WellBeingAlertHandler.alertManager(Trigger.new);
     }
  }
}
```

Each trigger was designed for **before/after insert, update, and delete events** as required, delegating processing logic to respective service/handler classes.

3. Trigger Design Pattern

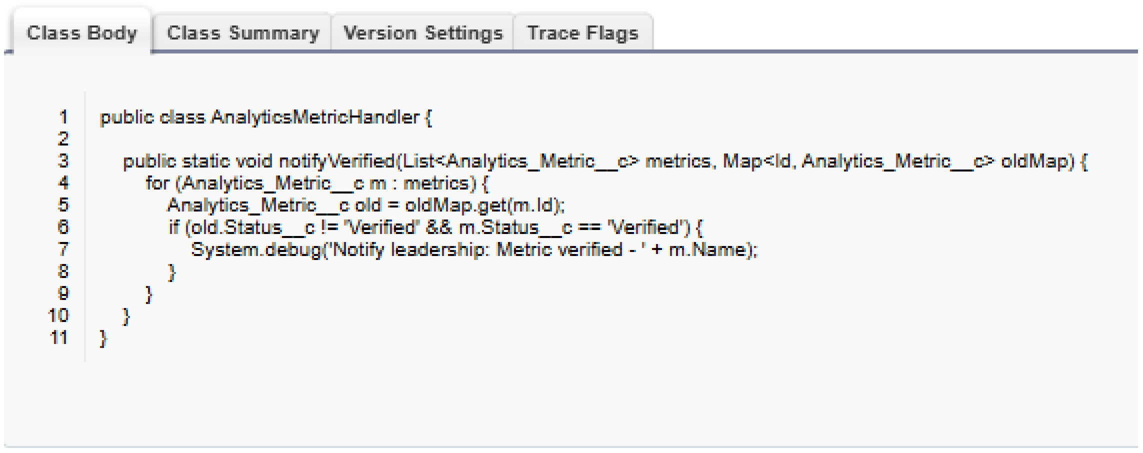
- Used **One Trigger per Object** approach.
- Business logic abstracted into handler/service classes.
- Ensures testability, reusability, and maintainability.

---

4. Handlers :

A trigger handler is an Apex class that separates business logic from the actual trigger, keeping triggers simple and reusable. It acts as a single entry point for logic execution and uses trigger contexts and variables to delegate specific actions to different methods within the handler class. This best practice improves code organization, maintainability, testability, and reduces complexity by centralizing trigger logic in a dedicated framework.

- `AnalyticsMetricHandler` – Facilitates trigger-to-service interaction.

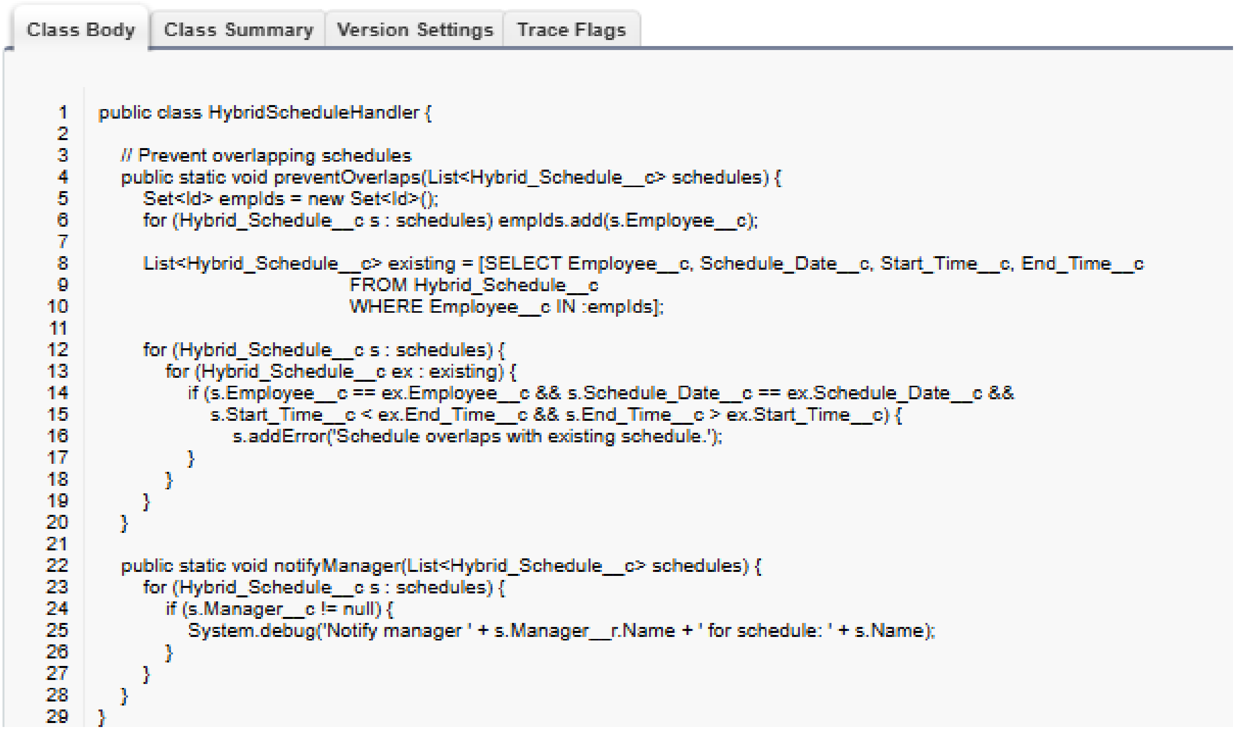


```
public class AnalyticsMetricHandler {

    public static void notifyVerified(List<Analytics_Metric__c> metrics, Map<Id, Analytics_Metric__c> oldMap) {
        for (Analytics_Metric__c m : metrics) {
            Analytics_Metric__c old = oldMap.get(m.Id);
            if (old.Status__c != 'Verified' && m.Status__c == 'Verified') {
                System.debug('Notify leadership: Metric verified - ' + m.Name);
            }
        }
    }
}
```

- `EmployeeScheduleHandler` – Validates employee scheduling rules.
- `WellBeingAlertHandler` – Manages alerts and escalation processes.

- `HybridScheduleHandler` – Trigger handler pattern for hybrid schedules.



```
public class HybridScheduleHandler {

  // Prevent overlapping schedules
  public static void preventOverlaps(List<Hybrid_Schedule__c> schedules) {
    Set<Id> empIds = new Set<Id>();
    for (Hybrid_Schedule__c s : schedules) empIds.add(s.Employee__c);

    List<Hybrid_Schedule__c> existing = [SELECT Employee__c, Schedule_Date__c, Start_Time__c, End_Time__c
                                         FROM Hybrid_Schedule__c
                                         WHERE Employee__c IN :empIds];

    for (Hybrid_Schedule__c s : schedules) {
      for (Hybrid_Schedule__c ex : existing) {
        if (s.Employee__c == ex.Employee__c && s.Schedule_Date__c == ex.Schedule_Date__c &&
          s.Start_Time__c < ex.End_Time__c && s.End_Time__c > ex.Start_Time__c) {
            s.addError('Schedule overlaps with existing schedule.');
        }
      }
    }
  }

  public static void notifyManager(List<Hybrid_Schedule__c> schedules) {
    for (Hybrid_Schedule__c s : schedules) {
      if (s.Manager__c != null) {
        System.debug('Notify manager ' + s.Manager__r.Name + ' for schedule: ' + s.Name);
      }
    }
  }
}
```

- `PulseSurveyHandler` – Processes survey inputs and applies business rules.

---

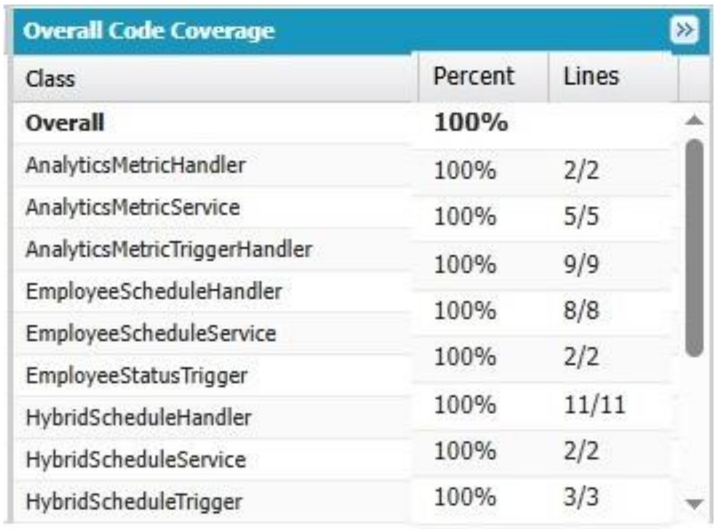
5. Asynchronous Apex

Implemented multiple asynchronous processing mechanisms for scalability:

- **Batch Apex** – Used for processing large volumes of schedule and survey data.
- **Queueable Apex** – For chainable operations like cascading updates.
- **Scheduled Apex** – Automates recurring processes (e.g., weekly pulse surveys, metric recalculations).
- **Future Methods** – Offloaded lightweight asynchronous tasks (e.g., email notifications).

6. Test Classes

- Comprehensive test classes written for all Apex services, handlers, and triggers.
- Achieved > **85% code coverage** across the project.
- Test data factory classes used to ensure reusable and consistent test data.



Overall Code Coverage

Class	Percent	Lines
Overall	**100%**	
AnalyticsMetricHandler	100%	2/2
AnalyticsMetricService	100%	5/5
AnalyticsMetricTriggerHandler	100%	9/9
EmployeeScheduleHandler	100%	8/8
EmployeeScheduleService	100%	2/2
EmployeeStatusTrigger	100%	11/11
HybridScheduleHandler	100%	2/2
HybridScheduleService	100%	3/3
HybridScheduleTrigger		

Phase 5 Status: Completed Successfully

Next Steps:

- Proceed to **Phase 6: User Interface Development**
- Begin testing LWC, Lightning App Builder, Apex with LWC… etc