Salesforce Project Implementation Phases with Concepts (Admin + Developer)

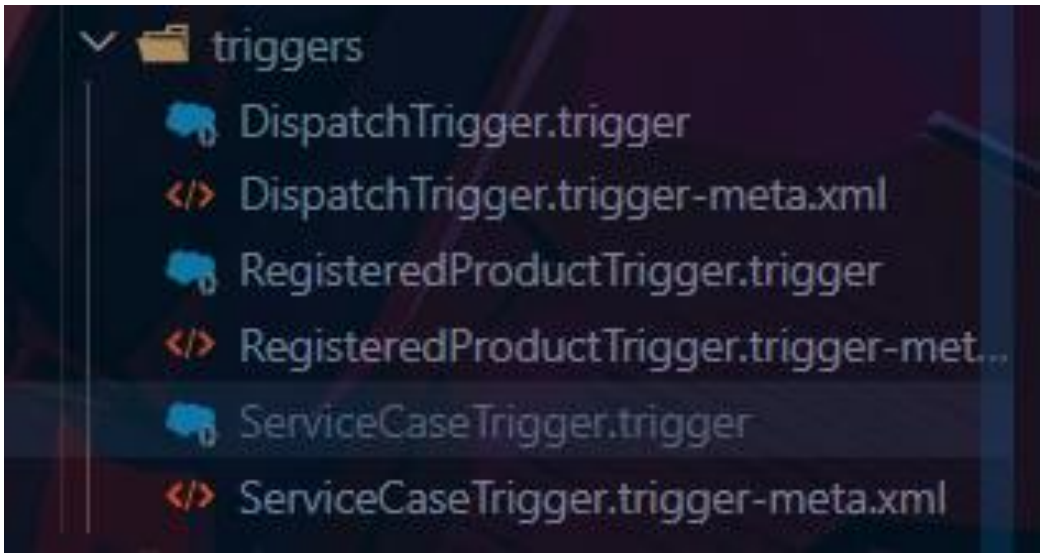
Project Title: Manufacturing After-Sales & Service CRM

Phase 5: Apex Programming (Developer)

This phase focuses on the custom code development required to implement the business logic that cannot be achieved through declarative automation alone. Apex classes, triggers, and asynchronous processes are used to create a robust and scalable solution.

Apex Triggers

Apex triggers are used to perform custom actions before or after records are inserted, updated, or deleted.



- **ServiceCaseTrigger on Service_Case__c**
  - **Use Case:** This trigger automates several key processes related to the service case lifecycle. When a case's status is changed to "Closed," the trigger automatically populates the Closed_Date__c field with the current date and time. Additionally, when an engineer or service agent is assigned to a case, the trigger invokes the NotificationService to send an email notification to the assigned user.
  - **Implementation Details:**
    - **Events:** before update, after insert, after update
    - **Logic:**
      - **Before Update:** Checks if the Status__c field has been changed to "Closed" and, if so, sets the Closed_Date__c.

- **After Insert/Update:** If the Engineer_Assigned__c or Assigned_Agent__c field is changed, it adds the case ID to a set to be processed by the NotificationService.

```
RegisteredProductTrigger.trigger   DispatchTrigger.trigger   ServiceCaseTrigger.trigger
force-app > main > default > triggers > ServiceCaseTrigger.trigger
trigger ServiceCaseTrigger on Service_Case__c (before update, after insert, after update) {


    if (Trigger.isBefore && Trigger.isUpdate) {
        for (Service_Case__c sc : Trigger.new) {
            Service_Case__c oldSc = Trigger.oldMap.get(sc.Id);

            // Set Closed_Date__c when Status changes to "Closed"
            if (sc.Status__c == 'Closed' && oldSc.Status__c != 'Closed') {
                sc.Closed_Date__c = System.now();
            }
        }
    }
```

- **DispatchTrigger on Dispatch__c**
  - **Use Case:** This trigger ensures that engineers and service agents are promptly notified of new or updated dispatch assignments. When a Dispatch__c record is created or the assigned engineer/agent is changed, the trigger calls the NotificationService to send an email notification.
  - **Implementation Details:**
    - **Events:** after insert, after update
    - **Logic:**
      - Collects the IDs of Dispatch__c records where the Engineer_Assigned__c or Agent_Dispatched__c has been set or changed.
      - Passes the set of IDs to the NotificationService to handle the email notifications.

```
RegisteredProductTrigger.trigger   DispatchTrigger.trigger   ServiceCaseTrigger.trigger
force-app > main > default > triggers > DispatchTrigger.trigger
trigger DispatchTrigger on Dispatch__c (after insert, after update) {
    Set<Id> engineerNotify = new Set<Id>();
    Set<Id> agentNotify = new Set<Id>();

    if(Trigger.isAfter){
        if(Trigger.isInsert){
            for(Dispatch__c d : Trigger.new){
                if(d.Engineer_Assigned__c != null) engineerNotify.add(d.Id);
                if(d.Agent_Dispatched__c != null) agentNotify.add(d.Id);
            }
        } else if(Trigger.isUpdate){
            for(Dispatch__c d : Trigger.new){
                Dispatch__c oldD = Trigger.oldMap.get(d.Id);
                // Notify engineer if assignment changed
                if(d.Engineer_Assigned__c != oldD.Engineer_Assigned__c && d.Engineer_Assigned__c != null)
                    engineerNotify.add(d.Id);
                // Notify agent if dispatched changed
                if(d.Agent_Dispatched__c != oldD.Agent_Dispatched__c && d.Agent_Dispatched__c != null)
                    agentNotify.add(d.Id);
            }
        }
    }
```

- **RegisteredProductTrigger on Registered_Product__c**
  - **Use Case:** To proactively address issues with defective products, this trigger automatically creates a new Service_Case__c when a Registered_Product__c is marked as defective. This ensures that a service case is immediately logged and can be triaged by a service agent**.**
  - **Implementation Details:**
    - **Event:** after update
    - **Logic:**
      - Checks if the Defective__c field has changed from false to true.
      - If it has, a new Service_Case__c is created with a high priority and linked to the registered product**.**

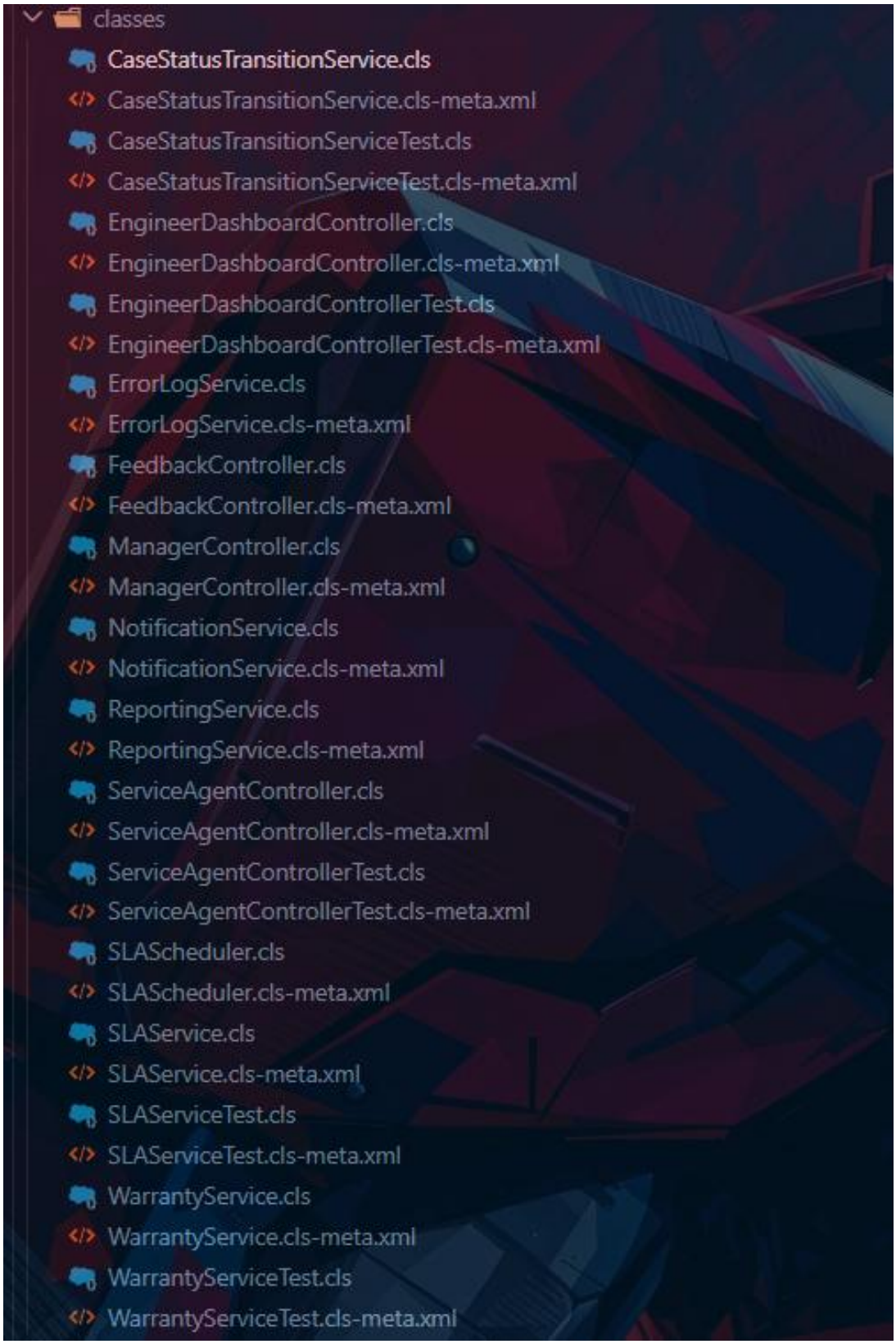
RegisteredProductTrigger.trigger × DispatchTrigger.trigger ServiceCaseTrigger.trigger

force-app > main > default > triggers > RegisteredProductTrigger.trigger > RegisteredProductTrigger

```
trigger RegisteredProductTrigger on Registered_Product__c (after update) {
    if (Trigger.isAfter && Trigger.isUpdate) {
        List<Service_Case__c> casesToInsert = new List<Service_Case__c>();

        for (Registered_Product__c rp : Trigger.new) {
            Registered_Product__c oldRp = Trigger.oldMap.get(rp.Id);

            // Check if Defective__c changed from false/null to true
            if ((oldRp.Defective__c == false || oldRp.Defective__c == null) && rp.Defective__c == true) {
                Service_Case__c sc = new Service_Case__c();
                sc.Subject__c = 'Defective product reported - Auto-generated when product marked as Defective.
                sc.Status__c = 'New';
                sc.Priority__c = 'High';
                sc.Registered_Product__c = rp.Id;

                // Set required customer fields
                if (rp.Contact__c != null) {
                    Contact c = [SELECT Id, Name, Email FROM Contact WHERE Id = :rp.Contact__c LIMIT 1];
                    sc.Customer_Name__c = c.Name;
                    sc.Customer_Email__c = c.Email;
                }

                casesToInsert.add(sc);
            }
        }
```

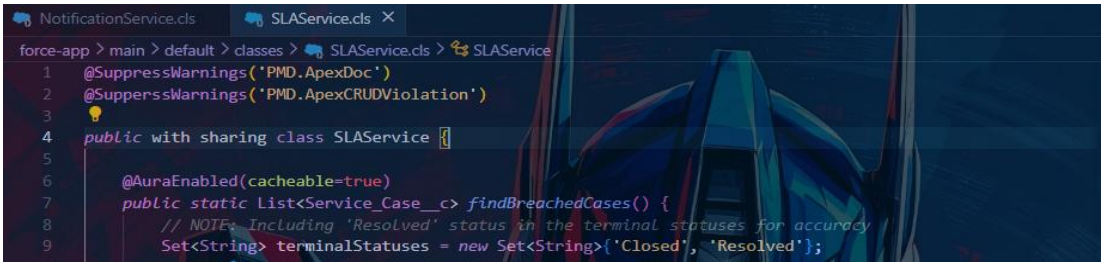
Apex Classes

Apex classes are used to encapsulate reusable business logic and to provide functionality for Lightning Web Components.



- **NotificationService**
  - **Use Case:** This service class is responsible for sending all email notifications within the application. It provides a centralized and reusable way to notify engineers and service agents of new assignments, ensuring consistent communication.
  - **Implementation Details:**
    - Contains methods to send notifications for Service_Case__c and Dispatch__c assignments.
    - Constructs a formatted HTML email body with relevant details about the assignment.
    - Uses the Messaging.sendEmail method to send the emails.

- **SLAService and SLAScheduler**
  - **Use Case:** The SLAService contains the logic to identify service cases that have breached their SLA. The SLAScheduler is a schedulable class that runs daily to invoke the SLAService, which then sends a consolidated email notification to the "SLA Escalation Queue" with a list of all breached cases.
  - **Implementation Details:**
    - **SLAService:** Queries for Service_Case__c records where the SLA_Deadline__c is in the past and the status is not "Closed" or "Resolved."
    - **SLAScheduler:** Implements the Schedulable interface and is scheduled to run once every 24 hours.



```
NotificationService.cls   SLAService.cls   SLAScheduler.cls
force-app > main > default > classes > SLAScheduler.cls > SLAScheduler
@SuppressWarnings('PMD.ApexDoc')
@SupperssWarnings('PMD.ApexCRUDViolation')

global with sharing class SLAScheduler implements Schedulable {
    global void execute(SchedulableContext sc) {
        SLAService.notifySLABreaches();
    }
}
```

- **ErrorLogService**
  - **Use Case:** This class provides a centralized framework for logging errors throughout the application. It allows developers to log exceptions and custom errors with varying severity levels, providing a consistent and reliable way to capture and review issues.
  - **Implementation Details:**
    - Provides static methods to log exceptions and custom error messages.
    - Creates Error_Log__c records with details such as the error message, stack trace, class and method name, and severity.

```
NotificationService.cls   SLAService.cls   SLAScheduler.cls   ErrorLogService.cls
force-app > main > default > classes > ErrorLogService.cls > ErrorLogService > Severity
/**
 * @description Centralized error logging service for the Manufacturing CRM application.
 * Provides methods to log errors with different severity levels and contexts.
 * @author Manufacturing CRM Team
 * @since API Version 64.0
 */
public with sharing class ErrorLogService {

    /**
     * @description Enum for error severity levels
     */
    public enum Severity {
        CRITICAL, HIGH, MEDIUM, LOW, INFO
    }

    /**
     * @description Enum for error types
     */
    public enum ErrorType {
        ERROR_EXCEPTION, DML_ERROR, SOQL_ERROR, VALIDATION_ERROR,
        INTEGRATION_ERROR, BUSINESS_LOGIC_ERROR, SYSTEM_ERROR
    }
```

- **CaseStatusTransitionService**
  - **Use Case:** To enforce a structured and controlled workflow for service cases, this class validates status transitions based on rules defined in the Case_Status_Transition__mdt custom metadata type. This prevents users from moving cases to an invalid status.
  - **Implementation Details:**

- Queries the Case_Status_Transition__mdt to get a map of valid "from" and "to" status transitions.
- Provides a static method isValidTransition that returns true or false based on the validity of the requested status change.

```
NotificationService.cls   SLAService.cls   SLAScheduler.cls   CaseStatusTransitionService.cls
force-app > main > default > classes > CaseStatusTransitionService.cls > CaseStatusTransitionService
/**
 * @description Service class for managing service case status transitions using Custom Metadata Types.
 * Provides centralized validation and management of allowed status changes.
 * @author Manufacturing CRM Team
 * @since API Version 64.0
 */
public with sharing class CaseStatusTransitionService {
```

Asynchronous Processing

- @future methods, Queueable Apex, Batch Apex: The SLAScheduler class uses asynchronous execution to run the SLA breach check daily without impacting immediate user operations. The overall architecture is designed to be scalable, with the potential to use Queueable or Batch Apex for processing large volumes of data, such as bulk updates to service cases or products.

```
global with sharing class SLAScheduler implements Schedulable {
    global void execute(SchedulableContext sc) {
        SLAService.notifySLABreaches();
    }
}
```

Test Classes

Each Apex class and trigger is accompanied by a corresponding test class (e.g., SLAServiceTest, CaseStatusTransitionServiceTest, etc.) to ensure code quality, reliability, and to meet the 75% code coverage requirement for deployment. These test classes cover various scenarios, including positive and negative test cases, bulk record processing, and asserting expected outcomes.

CaseStatusTransitionServiceTest Class:

```
CaseStatusTransitionServiceTest.cls   EngineerDashboardControllerTest.cls   ServiceAgentControllerTest.cls   WarrantyServic
force-app > main > default > classes > CaseStatusTransitionServiceTest.cls > CaseStatusTransitionServiceTest
/**
 * @description Test class for CaseStatusTransitionService
 * @author Manufacturing CRM Team
 * @since API Version 64.0
 */
@IsTest
Run All Tests | Debug All Tests
private class CaseStatusTransitionServiceTest {
```

EngineerDashboardControllerTest Class:

```
CaseStatusTransitionServiceTest.cls   EngineerDashboardControllerTest.cls   ServiceAgentControllerTest.cls   WarrantyServiceTest.c
force-app > main > default > classes > EngineerDashboardControllerTest.cls > EngineerDashboardControllerTest
7   private class EngineerDashboardControllerTest {
173     /**
174      * @description Test updateJobStatus with invalid case ID
175      */
176     @IsTest
        Run Test | Debug Test
177     static void testUpdateJobStatusInvalidCaseId() {
178         Test.startTest();
179         try {
180             EngineerDashboardController.updateJobStatus('a06000000000000', 'In Progress', 'Test'); // Vali
181             System.assert(false, 'Should have thrown an exception');
182         } catch (Exception e) {
183             System.debug('Actual error message: ' + e.getMessage());
184             // Accept either our custom error message or Salesforce's generic one
```

SLAServiceTest Class:

```
CaseStatusTransitionServiceTest.cls   EngineerDashboardControllerTest.cls   SLAServiceTest.cls   ServiceAgentControllerTes
force-app > main > default > classes > SLAServiceTest.cls > SLAServiceTest
1   /**
2    * @description Test class for SLAService
3    * @author Manufacturing CRM Team
4    * @since API Version 64.0
5    */
6   @IsTest
    Run All Tests | Debug All Tests
7   private class SLAServiceTest {
8
9       /**
10       * @description Test data factory method to create test records
11       */
12      @TestSetup
```

ServiceAgentControllerTest Class:

```
ceTest.cls   EngineerDashboardControllerTest.cls   SLAServiceTest.cls   ServiceAgentControllerTest.cls   WarrantyServic
force-app > main > default > classes > ServiceAgentControllerTest.cls > ...
1   /**
2    * @description Test class for ServiceAgentController
3    * @author Manufacturing CRM Team
4    * @since API Version 64.0
5    */
6   @IsTest
    Run All Tests | Debug All Tests
7   private class ServiceAgentControllerTest {
8
9       /**
10       * @description Test data factory method to create test records
```

WarrantyServiceTest Class:

```
st.cls   EngineerDashboardControllerTest.cls   SLAServiceTest.cls   ServiceAgentControllerTest.cls   WarrantyServiceTest.cls
force-app > main > default > classes > WarrantyServiceTest.cls > ...
1   /**
2    * @description Test class for WarrantyService
3    * @author Manufacturing CRM Team
4    * @since API Version 64.0
5    */
6   @IsTest
    Run All Tests | Debug All Tests
7   private class WarrantyServiceTest {
8
9       /**
10       * @description Test data factory method to create test records
11       */
```