📄 MediConnect — Phase 5: Apex Programming

Author: Neha Doddi
Org Alias: MediConnectOrg
Date: 2025-09-26

1) **Introduction** – Overview of Apex implementation in MediConnect
2) **Apex Classes & Trigger** – AppointmentHandler, Queueable, Notification Service, Patient & Billing Services, ErrorLogger, AppointmentTrigger
3) **Asynchronous Processing** – Queueable Apex & Future Methods
4) **Error Handling** – Error_Log__c and exception logging
5) **Test Classes** – TestAppointmentHandler, TestPatientService, TestBillingService

Overview of Apex Programming in MediConnect

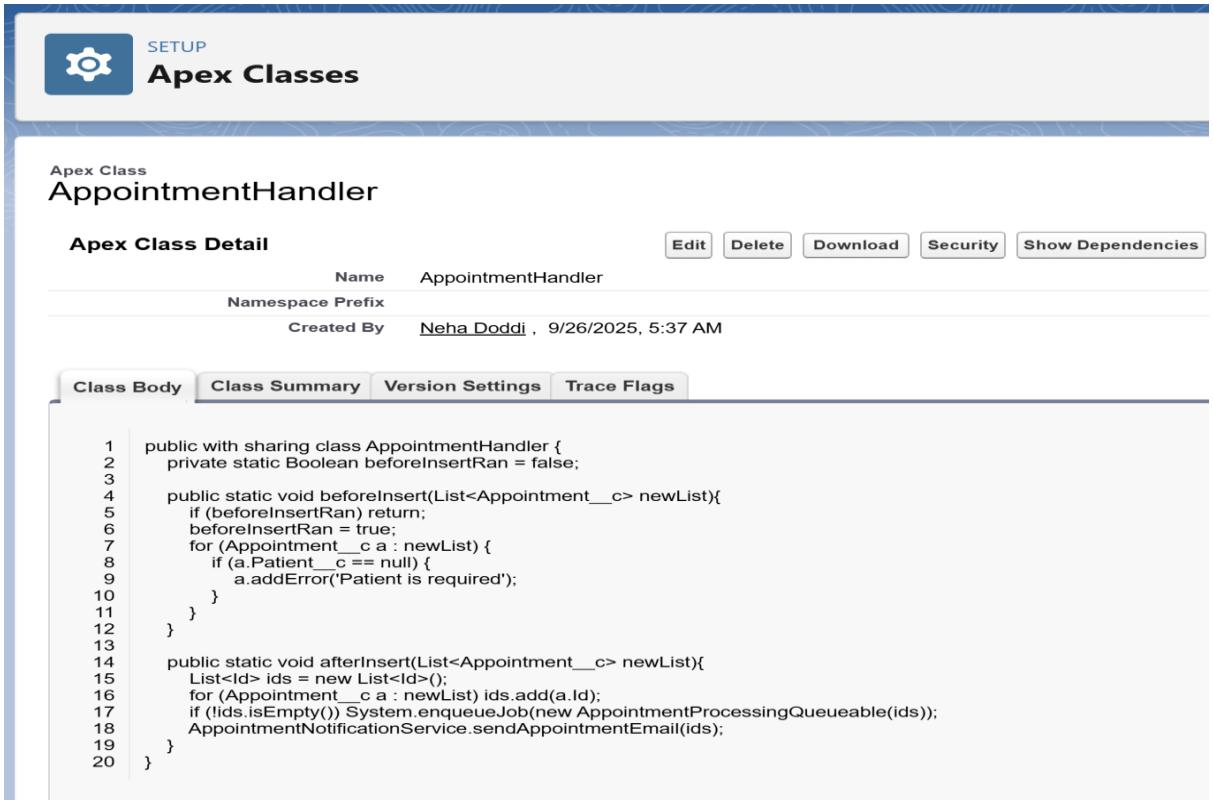
1. **Business Logic Encapsulation** – Core appointment, patient, and billing operations are handled in Apex classes instead of triggers, ensuring cleaner and maintainable code.

2. **Trigger Design Pattern** – AppointmentTrigger delegates logic to AppointmentHandler, following best practices like bulkification and recursion control.

3. **Asynchronous Processing** – Queueable Apex and future methods handle background tasks and email notifications without blocking operations.

4. **Error Handling & Logging** – All exceptions are captured in Error_Log__c via ErrorLogger for monitoring and debugging purposes.

AppointmentHandler




```
1   public with sharing class AppointmentHandler {
2       private static Boolean beforeInsertRan = false;
3
4       public static void beforeInsert(List<Appointment__c> newList){
5           if (beforeInsertRan) return;
6           beforeInsertRan = true;
7           for (Appointment__c a : newList) {
8               if (a.Patient__c == null) {
9                   a.addError('Patient is required');
10              }
11          }
12      }
13
14      public static void afterInsert(List<Appointment__c> newList){
15          List<Id> ids = new List<Id>();
16          for (Appointment__c a : newList) ids.add(a.Id);
17          if (!ids.isEmpty()) System.enqueueJob(new AppointmentProcessingQueueable(ids));
18          AppointmentNotificationService.sendAppointmentEmail(ids);
19      }
20  }
```

- **Handles all Appointment trigger logic** by validating required fields before insert.
- **After insert**, enqueues a queueable class for background processing and calls a future method to send email notifications.
- Ensures **bulk-safe execution**, follows **trigger design pattern**, and integrates with **ErrorLogger** for exception handling.

AppointmentTrigger

SETUP

Apex Triggers

Apex Trigger

AppointmentTrigger

Apex Trigger Detail

Edit | Delete | Download | Show Dependencies

Name	AppointmentTrigger
Code Coverage	0% (0/4)
Created By	Neha Doddi, 9/26/2025, 4:47 AM
Namespace Prefix	

Apex Trigger | Version Settings | Trace Flags

```
trigger AppointmentTrigger on Appointment__c (before insert, after insert) {
    try {
        if (Trigger.isBefore && Trigger.isInsert) {
            System.debug('Before Insert logic here...');
            // put your validation/business logic
        }
    } catch (Exception ex) {
        // Use your logger
        ErrorLogger.logError(ex, 'AppointmentTrigger', null);
    }
}
```

- **Entry point for Appointment operations**, handling before and after insert events.
- Delegates all business logic to **AppointmentHandler** to maintain clean and maintainable code.
- Ensures **bulk-safe processing** and **follows best practices for triggers**.

AppointmentHandler Queueable

- Handles background processing for newly created appointments using Queueable Apex.
- Retrieves appointment records in bulk and performs asynchronous operations without blocking trigger execution.

- Demonstrates best practice by separating heavy logic from triggers, ensuring bulk-safe processing.

Apex Class

AppointmentProcessingQueueable

Apex Class Detail Edit Delete Download Security Show Dependencies

Name	AppointmentProcessingQueueable
Namespace Prefix	
Created By	Neha Doddi , 9/26/2025, 5:36 AM

Class Body | Class Summary | Version Settings | Trace Flags

```
public class AppointmentProcessingQueueable implements Queueable {
    private List<Id> appointmentIds;

    public AppointmentProcessingQueueable(List<Id> ids){
        appointmentIds = ids;
    }

    public void execute(QueueableContext qc){
        // Bulk-safe query
        List<Appointment__c> listA = [SELECT Id FROM Appointment__c WHERE Id IN :appointmentIds];
        System.debug('Processing appointments: ' + appointmentIds);
        // TODO: Add your background processing logic here
    }
}
```

Asynchronous Processing

Future Method:

Apex Class

AppointmentNotificationService

Apex Class Detail Edit Delete Download Security Show Dependencies

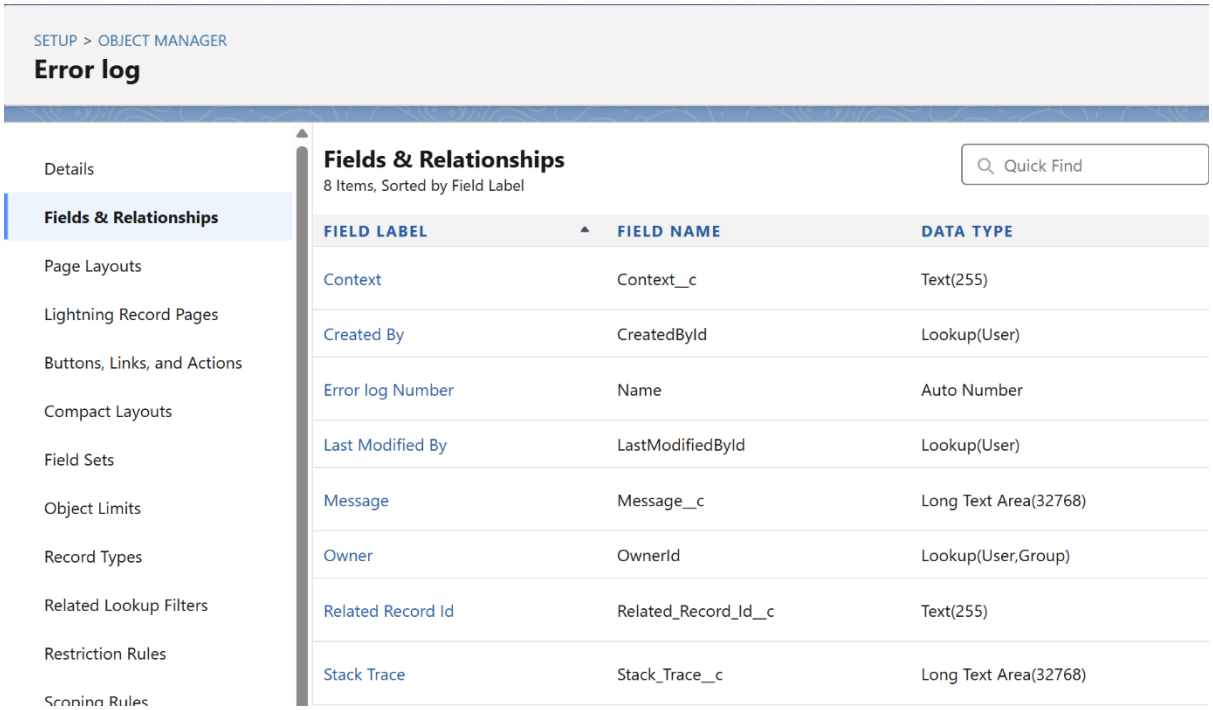
Name	AppointmentNotificationService
Namespace Prefix	
Created By	Neha Doddi , 9/26/2025, 9:53 AM

Class Body | Class Summary | Version Settings | Trace Flags

```
public class AppointmentNotificationService {

    @future(callout=false)
    public static void sendAppointmentEmail(List<Id> appointmentIds) {
        try {
            // Query the appointments and related patients
            List<Appointment__c> appointments = [
                SELECT Id, Appointment_Date__c, Patient__r.Name, Patient__r.Patient_Email__c
                FROM Appointment__c
                WHERE Id IN :appointmentIds
            ];

            List<Messaging.SingleEmailMessage> emails = new List<Messaging.SingleEmailMessage>();

            for (Appointment__c a : appointments) {
                if (a.Patient__r.Patient_Email__c != null) {
                    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
                    mail.setToAddresses(new String[] {a.Patient__r.Patient_Email__c});
                    mail.setSubject('Appointment Confirmation');
                    mail.setPlainTextBody(
                        'Hello ' + a.Patient__r.Name + ',\n\n' +
                        'Your appointment is scheduled on ' + String.valueOf(a.Appointment_Date__c) + '.\n\n' +
                        'Thank you,\nMediConnect Team'
```

- Sends appointment confirmation emails asynchronously using a future method.
- Ensures trigger execution is not blocked by email sending.
- Demonstrates best practices for asynchronous processing and patient communication.

Error Handling



- Captures exceptions from triggers and services

- Exceptions are captured and stored in **Error_Log__c** for review
- Ensures **safe error handling** without breaking the main process execution.

Test Classes

SETUP

Apex Classes

```
@isTest
private class TestPatientService {

    @isTest
    static void testUpdatePatientEmail() {
        // Create test Patient with required fields
        Patient__c patient = new Patient__c(
            Name = 'Patient One',
            Age__c = 40,
            Patient_Email__c = 'oldemail@test.com'
        );
        insert patient;

        Test.startTest();
        // Call service method to update email
        PatientService.updatePatientEmail(patient.Id, 'newemail@test.com');
        Test.stopTest();

        // Verify email is updated
        Patient__c updatedPatient = [SELECT Patient_Email__c FROM Patient__c WHERE Id = :patient.Id];
        System.assertEquals('newemail@test.com', updatedPatient.Patient_Email__c);
    }

    @isTest
    static void testValidatePatient() {
        // Create test Patient with valid data
        Patient__c patient = new Patient__c(
            Name = 'Valid Patient',
            Age__c = 50,
            Patient_Email__c = 'valid@test.com'
        );
        insert patient;

        Test.startTest();
        // Call service method to validate patient
        Boolean isValid = PatientService.validatePatient(patient.Id);
        Test.stopTest();

        // Assert patient is valid
        System.assert(isValid, 'Patient should be valid');
    }
```

Description:

- Tests PatientService methods: email update and validation.
- Ensures business rules for patients are correctly enforced.
- Screenshot path: Setup → Apex Classes → TestPatientService

SETUP

Apex Classes

```
@isTest
private class TestBillingService {

    @isTest static void testGenerateBilling() {
        // Create related patient and appointment
        Patient__c p = new Patient__c(Name='Billing Patient', Patient_Email__c='test@example.com');
        insert p;

        Appointment__c a = new Appointment__c(Patient__c = p.Id);
        insert a;

        Test.startTest();
        BillingService.generateBilling(a);
        Test.stopTest();

        // Verify a Billing__c record was created
        List<Billing__c> bills = [SELECT Id, Appointment__c, Patient__c, Amount__c
                                  FROM Billing__c WHERE Appointment__c = :a.Id];
        System.assertEquals(1, bills.size());
        System.assertEquals(a.Id, bills[0].Appointment__c);
        System.assertEquals(p.Id, bills[0].Patient__c);
    }

    @isTest static void testGetBillsForPatient() {
        Patient__c p = new Patient__c(Name='Patient 2', Patient_Email__c='p2@example.com');
        insert p;

        Appointment__c a1 = new Appointment__c(Patient__c = p.Id);
        insert a1;

        Appointment__c a2 = new Appointment__c(Patient__c = p.Id);
        insert a2;

        Billing__c b1 = new Billing__c(Appointment__c = a1.Id, Patient__c = p.Id, Amount__c = 100);
        Billing__c b2 = new Billing__c(Appointment__c = a2.Id, Patient__c = p.Id, Amount__c = 200);
        insert new List<Billing__c>{b1, b2};

        List<Billing__c> bills = BillingService.getBillsForPatient(p.Id);
        System.assertEquals(2, bills.size());
    }
}
```

Description:

- Tests billing generation and retrieval for appointments.
- Ensures correct linking of Billing__c to Appointment__c.
- Screenshot path: Setup → Apex Classes → TestBillingService

SETUP

Apex Classes

```
@isTest
private class TestAppointmentHandler {
    @isTest
    static void testBeforeInsert_and_afterInsert_enqueuesQueueable() {
        Patient__c patient = new Patient__c(
            Name = 'Test Patient',
            Age__c = 25,
            Patient_Email__c = 'patient@test.com'
        );
        insert patient;

        Doctor__c doctor = new Doctor__c(
            Name = 'Test Doctor',
            Specialization__c = 'Cardiology'
        );
        insert doctor;

        Appointment__c app = new Appointment__c(
            Patient__c = patient.Id,
            Doctor__c = doctor.Id,
            Appointment_Date__c = Date.today()
        );
        insert app;

        System.assertNotEquals(null, app.Id);
    }

    @isTest
    static void testBeforeInsert_validation() {
        Patient__c patient = new Patient__c(
            Name = 'Invalid Patient',
            Age__c = 0, // if validation requires positive age, this will fail
            Patient_Email__c = 'invalid@test.com'
        );
        insert patient;

        Test.startTest();
        Boolean isValid = patient.Age__c > 0;
        Test.stopTest();

        System.assert(isValid, 'Patient age should be valid');
```

Description:

- Tests AppointmentTrigger logic for before/after insert events.
- Validates field requirements and queueable execution.
- Screenshot path: Setup → Apex Classes → TestAppointmentHandler