

MediConnect — Phase 5: Apex Programming

Author: Neha Doddi

Org Alias: MediConnectOrg

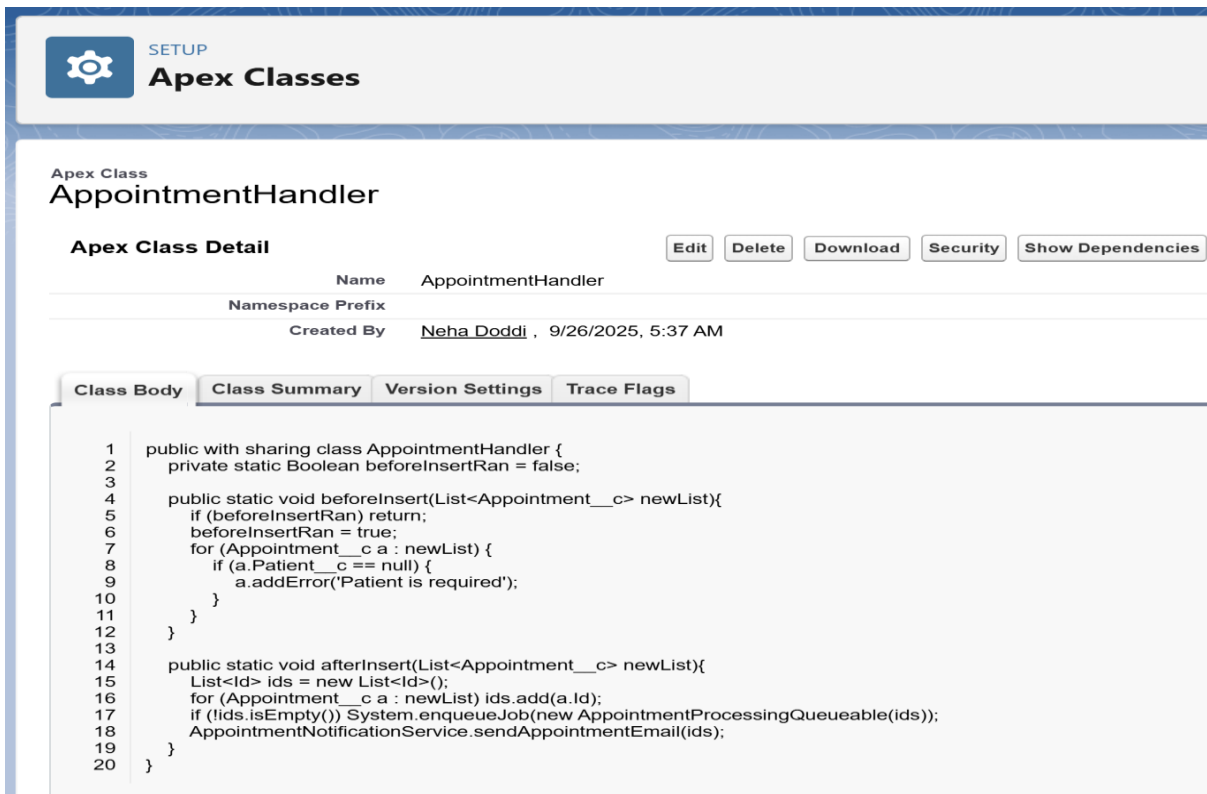
Date: 2025-09-26

- 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



The screenshot shows the Salesforce interface for the 'AppointmentHandler' Apex Class. At the top, there's a 'SETUP' button and the title 'Apex Classes'. Below this, the class name 'AppointmentHandler' is displayed. A table provides details about the class, including its name, namespace prefix, and creation information. The 'Class Body' tab is selected, showing the Apex code for the class. The code includes a 'beforeInsert' trigger that checks for patient information and an 'afterInsert' trigger that enqueues a job to send appointment emails.

Apex Class Detail

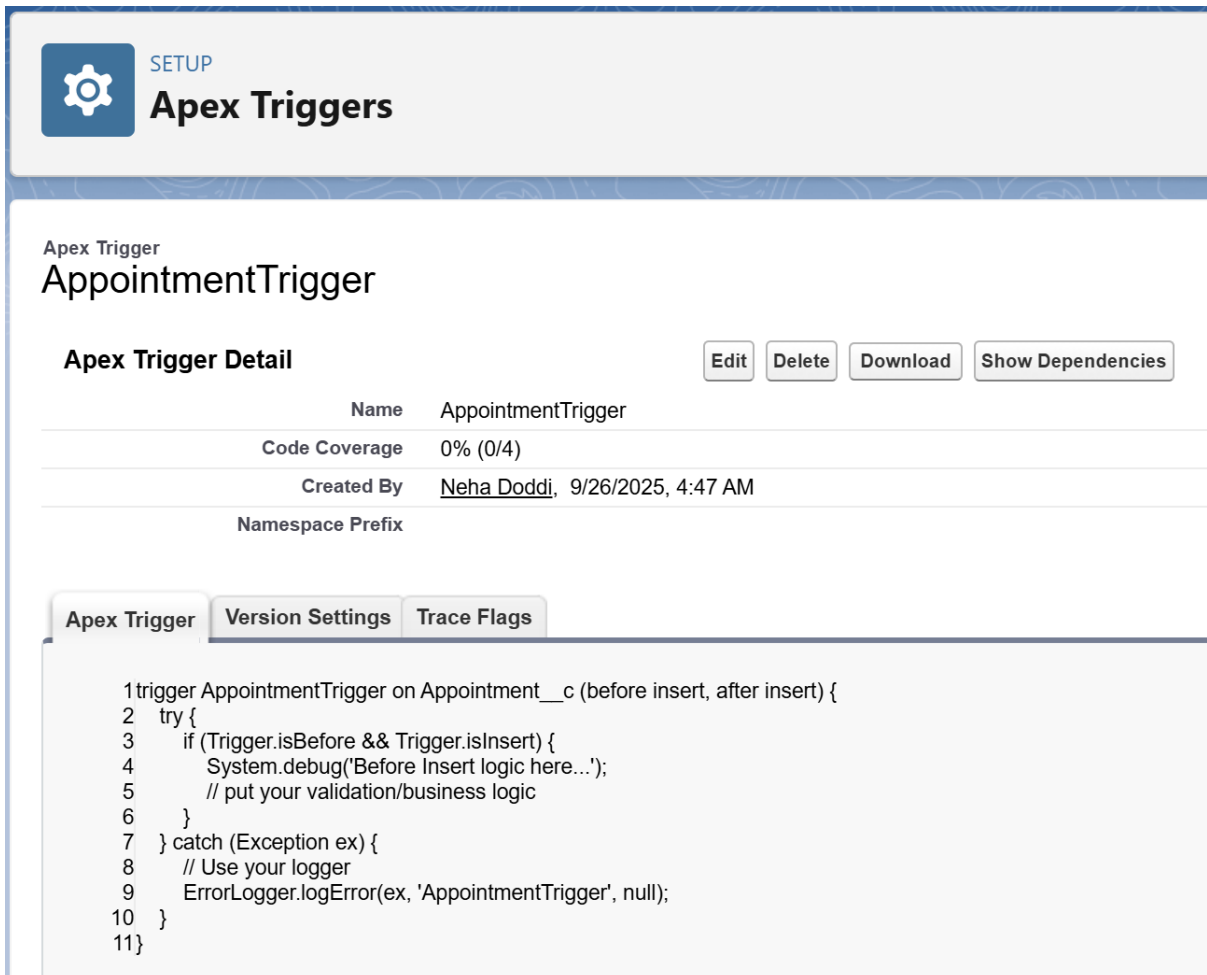
Name	AppointmentHandler
Namespace Prefix	
Created By	Neha Doddi , 9/26/2025, 5:37 AM

Class Body

```
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



The screenshot shows the Salesforce 'Apex Triggers' setup page. At the top, there's a 'SETUP' button and the title 'Apex Triggers'. Below this, the specific trigger 'AppointmentTrigger' is selected. A table provides details: Name is 'AppointmentTrigger', Code Coverage is '0% (0/4)', and Created By is 'Neha Doddi' on '9/26/2025, 4:47 AM'. The 'Namespace Prefix' is empty. Below the table are tabs for 'Apex Trigger', 'Version Settings', and 'Trace Flags'. The 'Apex Trigger' tab is active, displaying the following code:

```

1 trigger AppointmentTrigger on Appointment__c (before insert, after insert) {
2   try {
3     if (Trigger.isBefore && Trigger.isInsert) {
4       System.debug('Before Insert logic here...');
5       // put your validation/business logic
6     }
7   } catch (Exception ex) {
8     // Use your logger
9     ErrorLogger.logError(ex, 'AppointmentTrigger', null);
10  }
11}

```

- **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 Doddji](#) , 9/26/2025, 5:36 AM

[Class Body](#)[Class Summary](#)[Version Settings](#)[Trace Flags](#)

```
1 public class AppointmentProcessingQueueable implements Queueable {
2     private List<Id> appointmentIds;
3
4     public AppointmentProcessingQueueable(List<Id> ids){
5         appointmentIds = ids;
6     }
7
8     public void execute(QueueableContext qc){
9         // Bulk-safe query
10        List<Appointment__c> listA = [SELECT Id FROM Appointment__c WHERE Id IN :appointmentIds];
11        System.debug('Processing appointments: ' + appointmentIds);
12        // TODO: Add your background processing logic here
13    }
14 }
```

Asynchronous Processing

Future Method:

Apex Class

AppointmentNotificationService

Apex Class Detail

[Edit](#)[Delete](#)[Download](#)[Security](#)[Show Dependencies](#)

Name AppointmentNotificationService

Namespace Prefix

Created By [Neha Doddji](#) , 9/26/2025, 9:53 AM

[Class Body](#)[Class Summary](#)[Version Settings](#)[Trace Flags](#)

```
1 public class AppointmentNotificationService {
2
3     @future(callout=false)
4     public static void sendAppointmentEmail(List<Id> appointmentIds) {
5         try {
6             // Query the appointments and related patients
7             List<Appointment__c> appointments = [
8                 SELECT Id, Appointment_Date__c, Patient__r.Name, Patient__r.Patient_Email__c
9                 FROM Appointment__c
10                WHERE Id IN :appointmentIds
11            ];
12
13            List<Messaging.SingleEmailMessage> emails = new List<Messaging.SingleEmailMessage>();
14
15            for (Appointment__c a : appointments) {
16                if (a.Patient__r.Patient_Email__c != null) {
17                    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
18                    mail.setToAddresses(new String[] {a.Patient__r.Patient_Email__c});
19                    mail.setSubject('Appointment Confirmation');
20                    mail.setPlainTextBody(
21                        'Hello ' + a.Patient__r.Name + ',\n\n' +
22                        'Your appointment is scheduled on ' + String.valueOf(a.Appointment_Date__c) + '.\n\n' +
23                        'Thank you,\nMediConnect Team'
```

```

23         thank you, inMedConnect team
24     };
25     emails.add(mail);
26 }
27 }
28
29 if (!emails.isEmpty()) {
30     Messaging.sendEmail(emails);
31 }
32
33 } catch (Exception ex) {
34     ErrorLogger.log(ex, 'sendAppointmentEmail', null);
35 }
36 }
37 }

```

- 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

SETUP > OBJECT MANAGER

Error log

Details

Fields & Relationships

Page Layouts

Lightning Record Pages

Buttons, Links, and Actions

Compact Layouts

Field Sets

Object Limits

Record Types

Related Lookup Filters

Restriction Rules

Sharing Rules

Fields & Relationships

8 Items, Sorted by Field Label

Q Quick Find

FIELD LABEL	FIELD NAME	DATA TYPE
Context	Context__c	Text(255)
Created By	CreatedById	Lookup(User)
Error log Number	Name	Auto Number
Last Modified By	LastModifiedById	Lookup(User)
Message	Message__c	Long Text Area(32768)
Owner	OwnerId	Lookup(User,Group)
Related Record Id	Related_Record_Id__c	Text(255)
Stack Trace	Stack_Trace__c	Long Text Area(32768)

- 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

```
1  @isTest
2  private class TestPatientService {
3
4      @isTest
5      static void testUpdatePatientEmail() {
6          // Create test Patient with required fields
7          Patient__c patient = new Patient__c(
8              Name = 'Patient One',
9              Age__c = 40,
10             Patient_Email__c = 'oldemail@test.com'
11         );
12         insert patient;
13
14         Test.startTest();
15         // Call service method to update email
16         PatientService.updatePatientEmail(patient.Id, 'newemail@test.com');
17         Test.stopTest();
18
19         // Verify email is updated
20         Patient__c updatedPatient = [SELECT Patient_Email__c FROM Patient__c WHERE Id = :patient.Id];
21         System.assertEquals('newemail@test.com', updatedPatient.Patient_Email__c);
22     }
23
24     @isTest
25     static void testValidatePatient() {
26         // Create test Patient with valid data
27         Patient__c patient = new Patient__c(
28             Name = 'Valid Patient',
29             Age__c = 50,
30             Patient_Email__c = 'valid@test.com'
31         );
32         insert patient;
33
34         Test.startTest();
35         // Call service method to validate patient
36         Boolean isValid = PatientService.validatePatient(patient.Id);
37         Test.stopTest();
38
39         // Assert patient is valid
40         System.assert(isValid, 'Patient should be valid');
41     }
```

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

```
1  @isTest
2  private class TestBillingService {
3
4      @isTest static void testGenerateBilling() {
5          // Create related patient and appointment
6          Patient__c p = new Patient__c(Name='Billing Patient', Patient_Email__c='test@example.com');
7          insert p;
8
9          Appointment__c a = new Appointment__c(Patient__c = p.Id);
10         insert a;
11
12         Test.startTest();
13         BillingService.generateBilling(a);
14         Test.stopTest();
15
16         // Verify a Billing__c record was created
17         List<Billing__c> bills = [SELECT Id, Appointment__c, Patient__c, Amount__c
18                                FROM Billing__c WHERE Appointment__c = :a.Id];
19         System.assertEquals(1, bills.size());
20         System.assertEquals(a.Id, bills[0].Appointment__c);
21         System.assertEquals(p.Id, bills[0].Patient__c);
22     }
23
24     @isTest static void testGetBillsForPatient() {
25         Patient__c p = new Patient__c(Name='Patient 2', Patient_Email__c='p2@example.com');
26         insert p;
27
28         Appointment__c a1 = new Appointment__c(Patient__c = p.Id);
29         insert a1;
30
31         Appointment__c a2 = new Appointment__c(Patient__c = p.Id);
32         insert a2;
33
34         Billing__c b1 = new Billing__c(Appointment__c = a1.Id, Patient__c = p.Id, Amount__c = 100);
35         Billing__c b2 = new Billing__c(Appointment__c = a2.Id, Patient__c = p.Id, Amount__c = 200);
36         insert new List<Billing__c>{b1, b2};
37
38         List<Billing__c> bills = BillingService.getBillsForPatient(p.Id);
39         System.assertEquals(2, bills.size());
40     }
41 }
```

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

```
1  @isTest
2  private class TestAppointmentHandler {
3      @isTest
4      static void testBeforeInsert_and_afterInsert_enqueuesQueueable() {
5          Patient__c patient = new Patient__c(
6              Name = 'Test Patient',
7              Age__c = 25,
8              Patient_Email__c = 'patient@test.com'
9          );
10         insert patient;
11
12         Doctor__c doctor = new Doctor__c(
13             Name = 'Test Doctor',
14             Specialization__c = 'Cardiology'
15         );
16         insert doctor;
17
18         Appointment__c app = new Appointment__c(
19             Patient__c = patient.Id,
20             Doctor__c = doctor.Id,
21             Appointment_Date__c = Date.today()
22         );
23         insert app;
24
25         System.assertNotEquals(null, app.Id);
26     }
27
28     @isTest
29     static void testBeforeInsert_validation() {
30         Patient__c patient = new Patient__c(
31             Name = 'Invalid Patient',
32             Age__c = 0, // if validation requires positive age, this will fail
33             Patient_Email__c = 'invalid@test.com'
34         );
35         insert patient;
36
37         Test.startTest();
38         Boolean isValid = patient.Age__c > 0;
39         Test.stopTest();
40
41         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