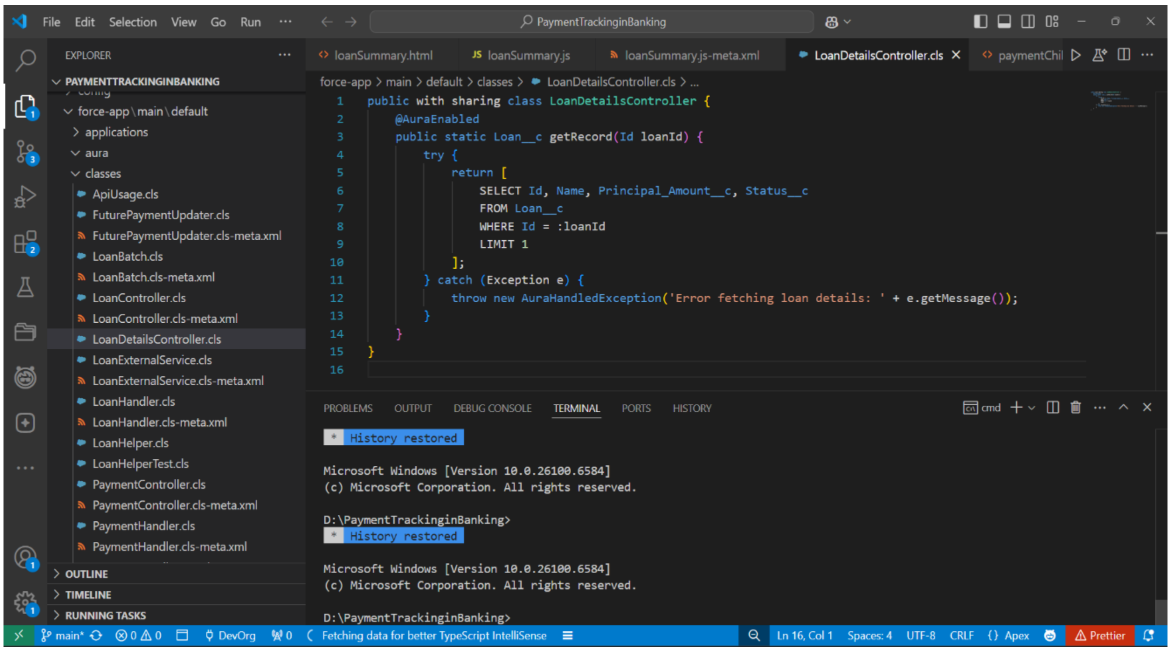
Phase 5: Apex Programming (Developer)

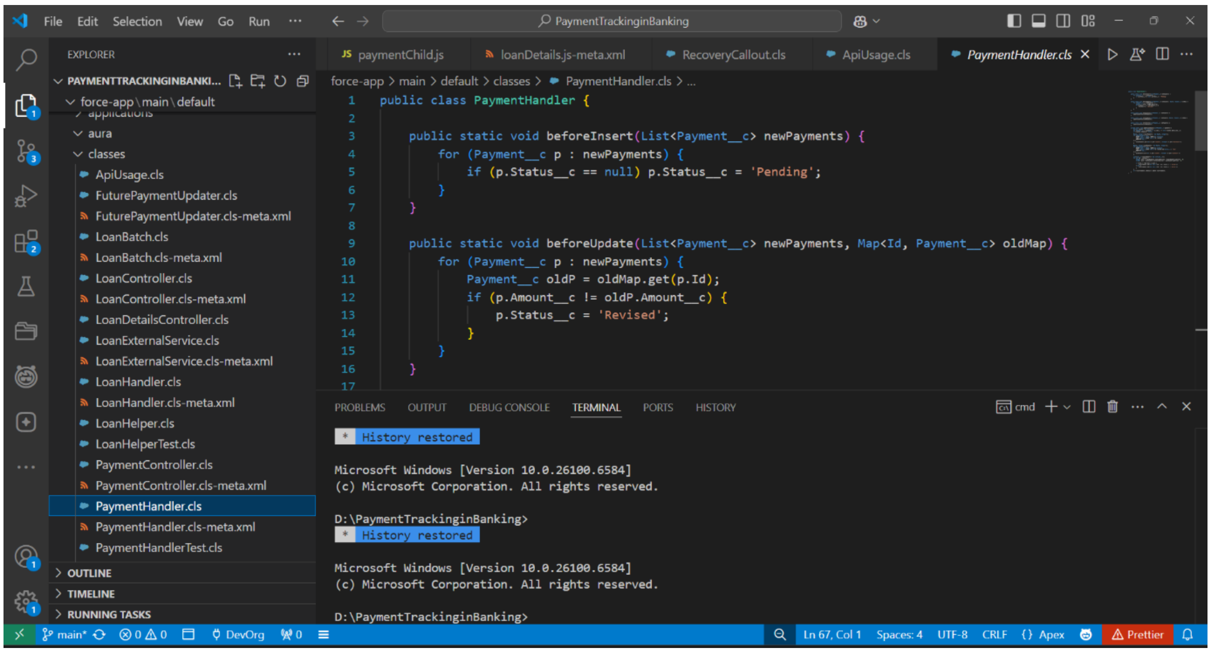
Classes & Objects

- Created Apex helper classes (LoanHandler, PaymentHandler, RecoveryHandler) to separate business logic from triggers.
- Used object-oriented approach for cleaner, reusable, and modular code.


File Edit Selection View Go Run
PaymentTrackingBanking
EXPLORER
PAYMENTTRACKINGINBANKING
force-app\main\default
applications
aura
classes
ApiUsage.cls
FuturePaymentUpdater.cls
FuturePaymentUpdater.cls-meta.xml
LoanBatch.cls
LoanBatch.cls-meta.xml
LoanController.cls
LoanController.cls-meta.xml
LoanDetailsController.cls
LoanExternalService.cls
LoanExternalService.cls-meta.xml
LoanHandler.cls
LoanHandler.cls-meta.xml
LoanHelper.cls
LoanHelperTest.cls
PaymentController.cls
PaymentController.cls-meta.xml
PaymentHandler.cls
PaymentHandler.cls-meta.xml
OUTLINE
TIMELINE
RUNNING TASKS
loanSummary.html
loanSummary.js
loanSummary.js-meta.xml
LoanDetailsController.cls
paymentChil
force-app > main > default > classes > LoanDetailsController.cls > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS HISTORY
cmd
main* ⊗0 ⚠0 DevOrg 0 Fetching data for better TypeScript IntelliSense
Ln 16, Col 1 Spaces: 4 UTF-8 CRLF {} Apex Prettier

```
public with sharing class LoanDetailsController {
    @AuraEnabled
    public static Loan__c getRecord(Id loanId) {
        try {
            return [
                SELECT Id, Name, Principal_Amount__c, Status__c
                FROM Loan__c
                WHERE Id = :loanId
                LIMIT 1
            ];
        } catch (Exception e) {
            throw new AuraHandledException('Error fetching loan details: ' + e.getMessage());
        }
    }
}
```

```
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
```

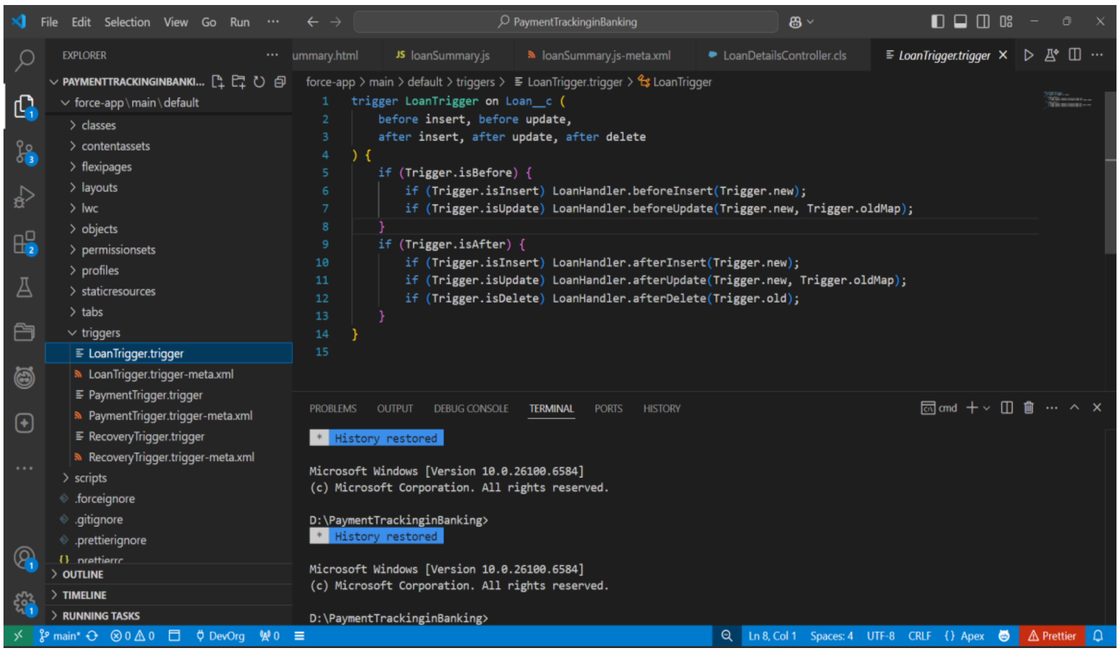


```
public class PaymentHandler {

    public static void beforeInsert(List<Payment__c> newPayments) {
        for (Payment__c p : newPayments) {
            if (p.Status__c == null) p.Status__c = 'Pending';
        }
    }

    public static void beforeUpdate(List<Payment__c> newPayments, Map<Id, Payment__c> oldMap) {
        for (Payment__c p : newPayments) {
            Payment__c oldP = oldMap.get(p.Id);
            if (p.Amount__c != oldP.Amount__c) {
                p.Status__c = 'Revised';
            }
        }
    }
```

```
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
```

Apex Triggers

- Implemented triggers for Loan, Payment, and Recovery to handle automation like updating statuses and roll-ups.
- Used before/after events for validation and DML updates while avoiding recursion.

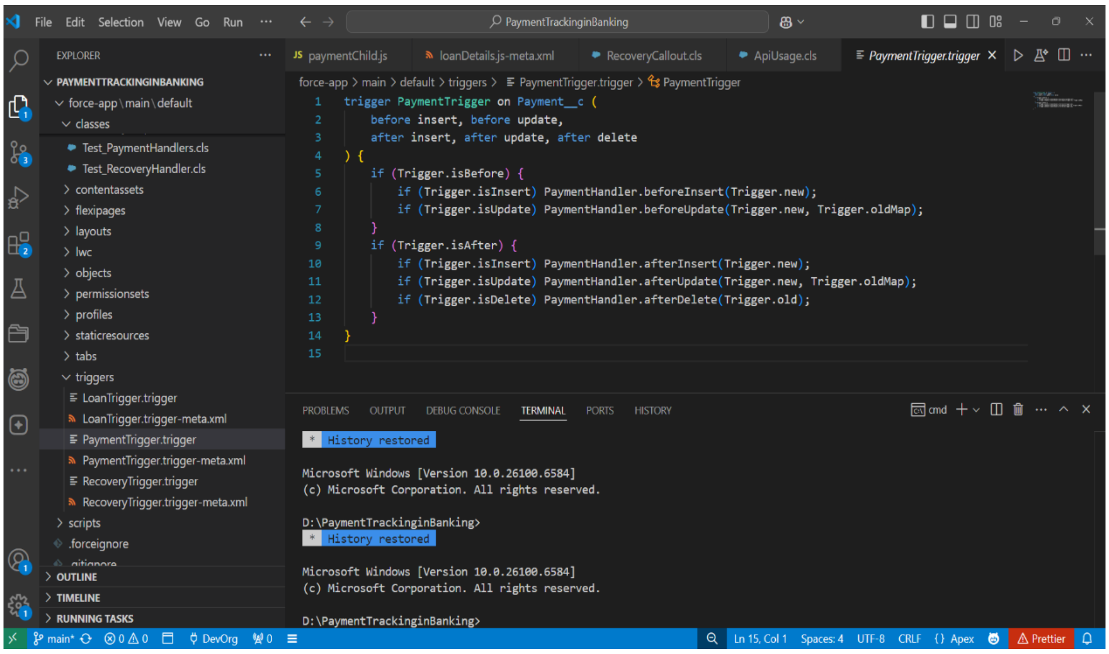


```
trigger LoanTrigger on Loan__c (
    before insert, before update,
    after insert, after update, after delete
) {
    if (Trigger.isBefore) {
        if (Trigger.isInsert) LoanHandler.beforeInsert(Trigger.new);
        if (Trigger.isUpdate) LoanHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
    }
    if (Trigger.isAfter) {
        if (Trigger.isInsert) LoanHandler.afterInsert(Trigger.new);
        if (Trigger.isUpdate) LoanHandler.afterUpdate(Trigger.new, Trigger.oldMap);
        if (Trigger.isDelete) LoanHandler.afterDelete(Trigger.old);
    }
}
```

```
 * History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
 * History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
```

Trigger Design Pattern

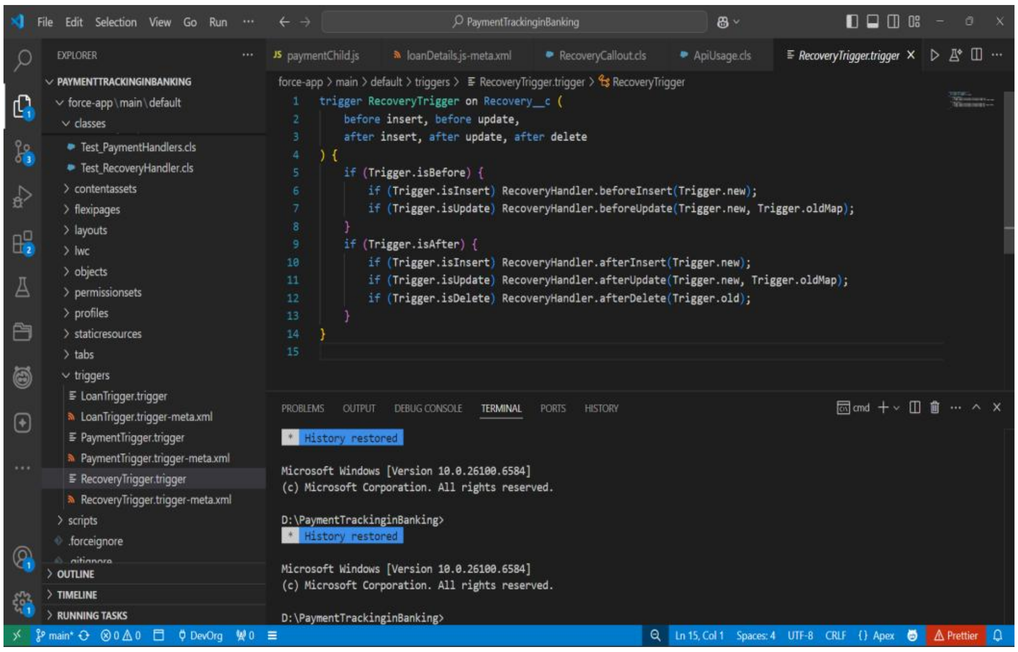
- Applied Trigger Handler Pattern to keep trigger code clean and delegate logic to handler classes.



```
trigger PaymentTrigger on Payment__c (
    before insert, before update,
    after insert, after update, after delete
) {
    if (Trigger.isBefore) {
        if (Trigger.isInsert) PaymentHandler.beforeInsert(Trigger.new);
        if (Trigger.isUpdate) PaymentHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
    }
    if (Trigger.isAfter) {
        if (Trigger.isInsert) PaymentHandler.afterInsert(Trigger.new);
        if (Trigger.isUpdate) PaymentHandler.afterUpdate(Trigger.new, Trigger.oldMap);
        if (Trigger.isDelete) PaymentHandler.afterDelete(Trigger.old);
    }
}
```

```
 * History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
 * History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
```

- Ensured only one trigger per object with proper event handling.

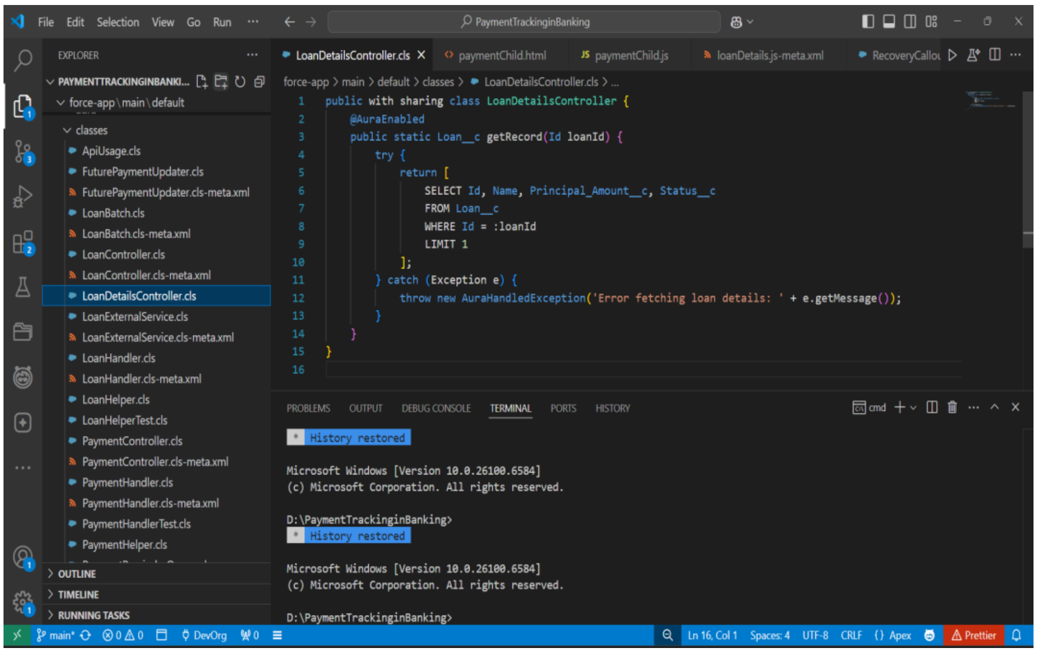


```
trigger RecoveryTrigger on Recovery__c (
    before insert, before update,
    after insert, after update, after delete
) {
    if (Trigger.isBefore) {
        if (Trigger.isInsert) RecoveryHandler.beforeInsert(Trigger.new);
        if (Trigger.isUpdate) RecoveryHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
    }
    if (Trigger.isAfter) {
        if (Trigger.isInsert) RecoveryHandler.afterInsert(Trigger.new);
        if (Trigger.isUpdate) RecoveryHandler.afterUpdate(Trigger.new, Trigger.oldMap);
        if (Trigger.isDelete) RecoveryHandler.afterDelete(Trigger.old);
    }
}
```

```
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
```

SOQL & SOSL

- Used SOQL in classes to query Loans, Payments, and Recoveries.
- Implemented simple search functionality to fetch customer records.



```
public with sharing class LoanDetailsController {
    @AuraEnabled
    public static Loan__c getRecord(Id loanId) {
        try {
            return [
                SELECT Id, Name, Principal_Amount__c, Status__c
                FROM Loan__c
                WHERE Id = :loanId
                LIMIT 1
            ];
        } catch (Exception e) {
            throw new AuraHandledException('Error fetching loan details: ' + e.getMessage());
        }
    }
}
```

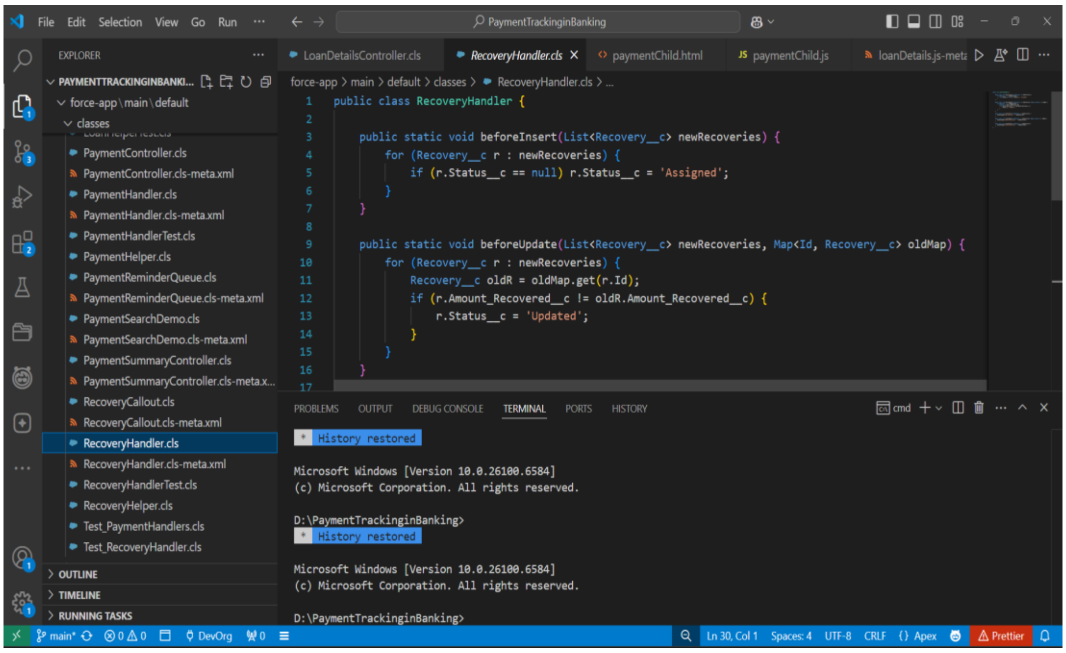
```
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
```

Collections: List, Set, Map

- Applied Lists to hold multiple Payments for bulk processing.
- Used Maps for Loan → Payments mapping to handle roll-ups efficiently.

Control Statements

- Implemented IF conditions for validation

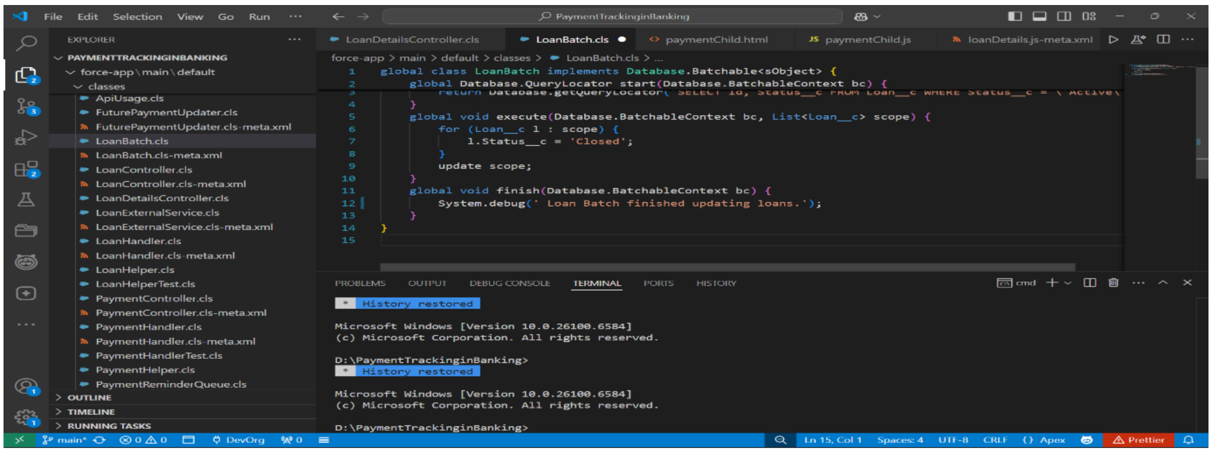


```
public class RecoveryHandler {

    public static void beforeInsert(List<Recovery__c> newRecoveries) {
        for (Recovery__c r : newRecoveries) {
            if (r.Status__c == null) r.Status__c = 'Assigned';
        }
    }

    public static void beforeUpdate(List<Recovery__c> newRecoveries, Map<Id, Recovery__c> oldMap) {
        for (Recovery__c r : newRecoveries) {
            Recovery__c oldR = oldMap.get(r.Id);
            if (r.Amount_Recovered__c != oldR.Amount_Recovered__c) {
                r.Status__c = 'Updated';
            }
        }
    }
```

```
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
```

Batch Apex

- Created simple batch class for recalculating loan balances.
- Enabled processing of large datasets asynchronously.

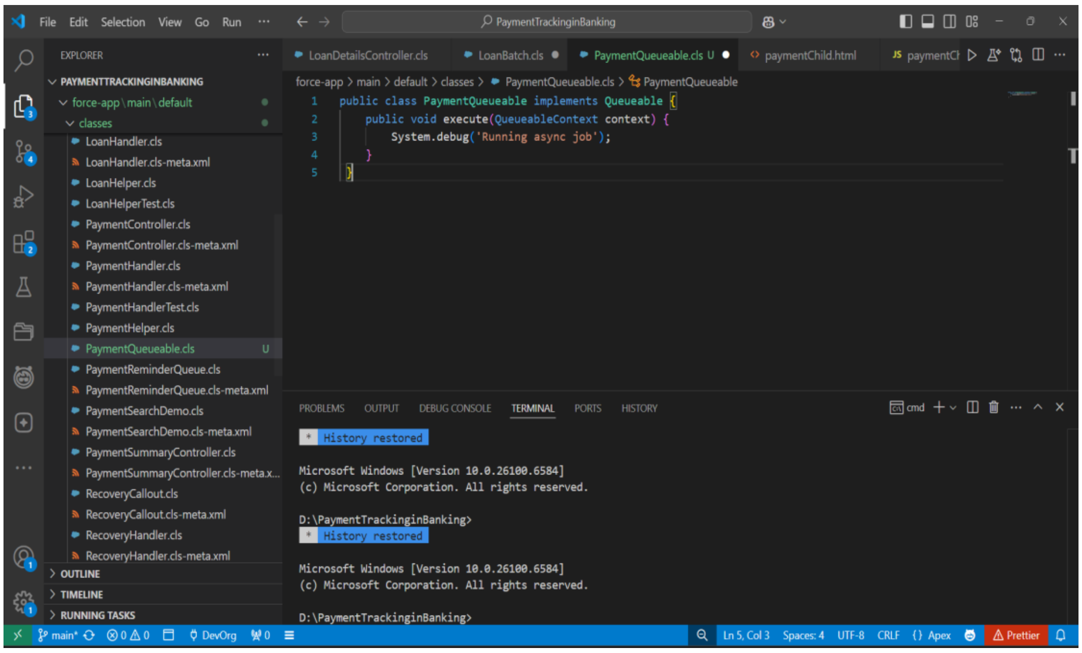


```
global class LoanBatch implements Database.Batchable<sObject> {
    global Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id, Status__c FROM Loan__c WHERE Status__c = \'Active\
    }
    global void execute(Database.BatchableContext bc, List<Loan__c> scope) {
        for (Loan__c l : scope) {
            l.Status__c = 'Closed';
        }
        update scope;
    }
    global void finish(Database.BatchableContext bc) {
        System.debug(' Loan Batch finished updating loans.');
    }
}
```

```
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
* History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
```

Queueable Apex

- Used Queueable Apex for lightweight async operations like notifications.
- Provides better chaining support compared to future methods.

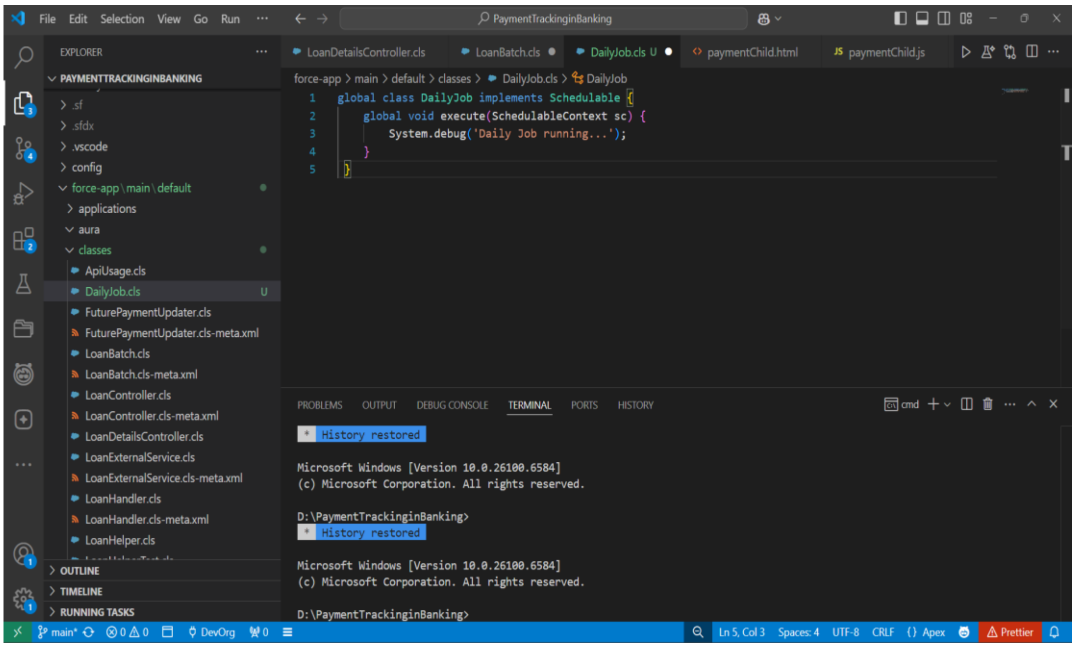


```
public class PaymentQueueable implements Queueable {
    public void execute(QueueableContext context) {
        System.debug('Running async job');
    }
}
```

```
History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
```

Scheduled Apex

- Built a scheduler to run daily overdue payment checks.
- Automated reminders and recovery creation without manual intervention.



```
global class DailyJob implements Schedulable {
    global void execute(SchedulableContext sc) {
        System.debug('Daily Job running...');
    }
}
```

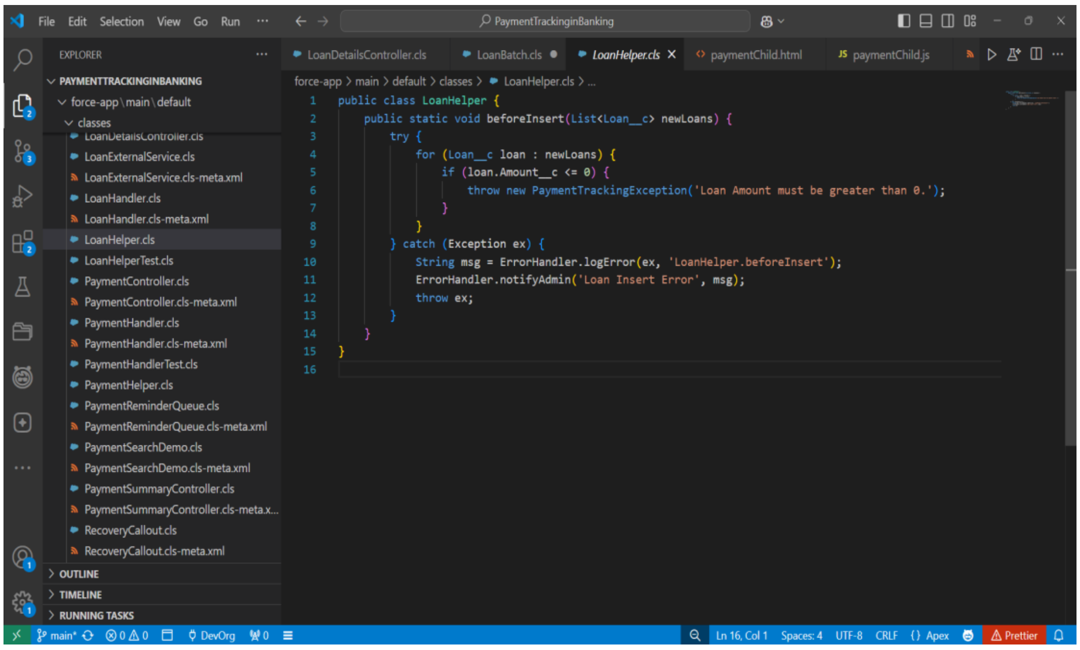
```
History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
History restored

Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

D:\PaymentTrackinginBanking>
```

Future Methods

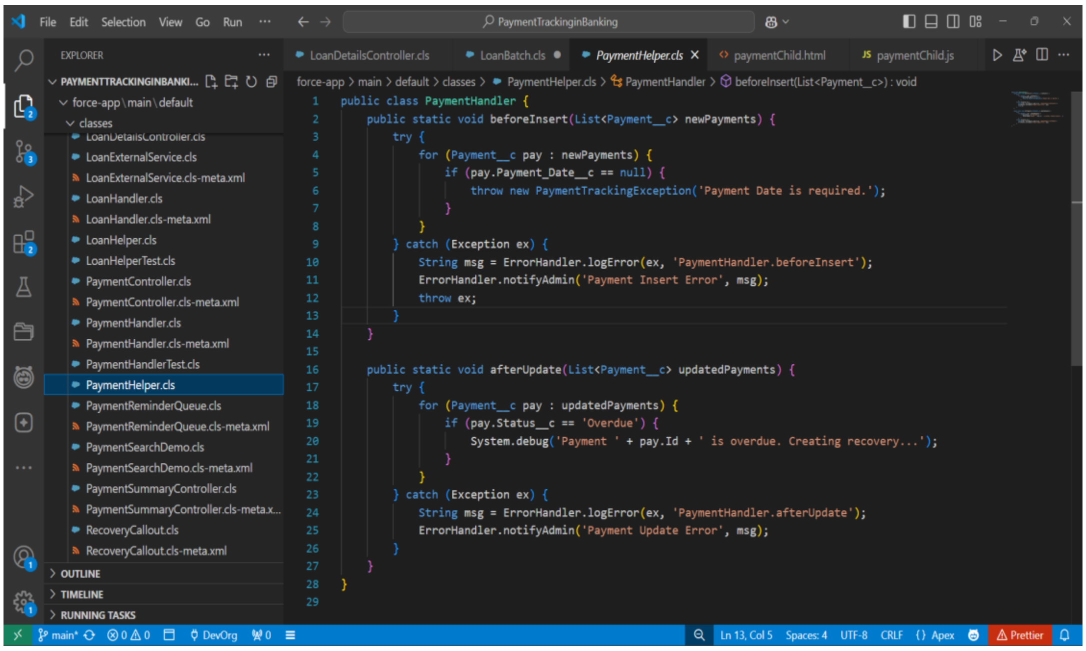
- Implemented future call for sending async email notifications.

Exception Handling

- Created a centralized ErrorHandler class to log and handle exceptions.
- Ensured smooth error reporting without stopping automation.



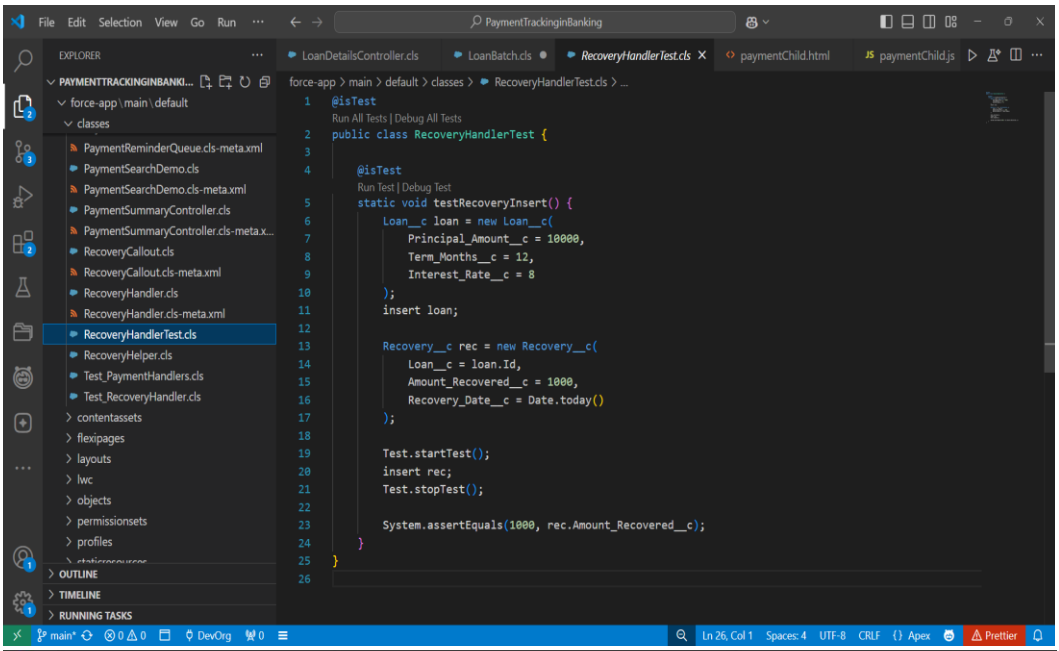
```
public class LoanHelper {
    public static void beforeInsert(List<Loan__c> newLoans) {
        try {
            for (Loan__c loan : newLoans) {
                if (loan.Amount__c <= 0) {
                    throw new PaymentTrackingException('Loan Amount must be greater than 0.');
                }
            }
        } catch (Exception ex) {
            String msg = ErrorHandler.logError(ex, 'LoanHelper.beforeInsert');
            ErrorHandler.notifyAdmin('Loan Insert Error', msg);
            throw ex;
        }
    }
}
```



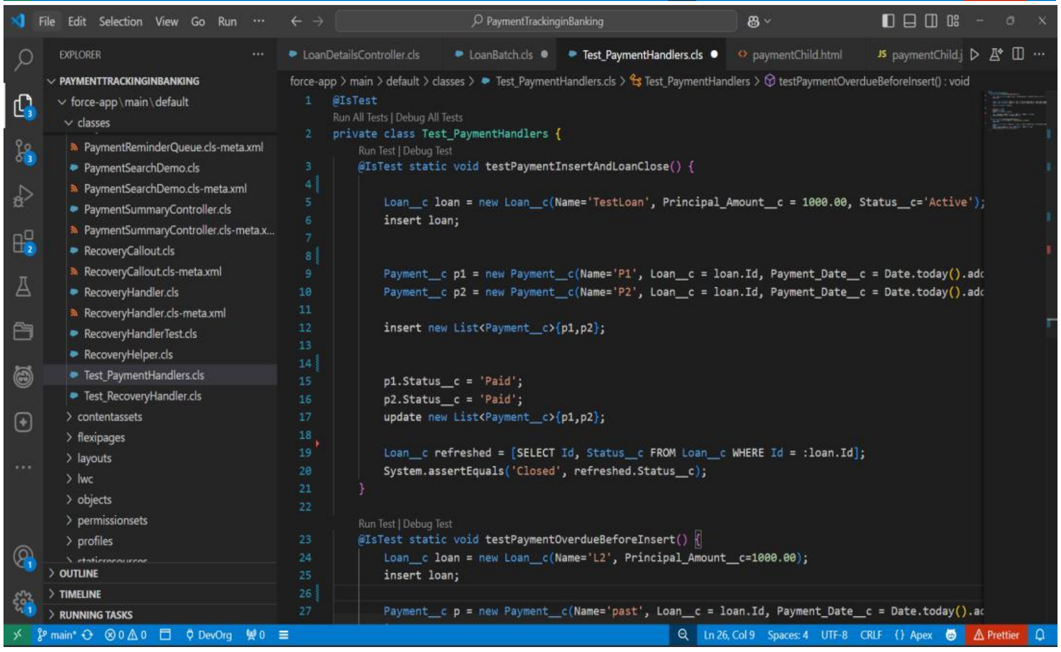
```
public class PaymentHandler {
    public static void beforeInsert(List<Payment__c> newPayments) {
        try {
            for (Payment__c pay : newPayments) {
                if (pay.Payment_Date__c == null) {
                    throw new PaymentTrackingException('Payment Date is required.');
                }
            }
        } catch (Exception ex) {
            String msg = ErrorHandler.logError(ex, 'PaymentHandler.beforeInsert');
            ErrorHandler.notifyAdmin('Payment Insert Error', msg);
            throw ex;
        }
    }

    public static void afterUpdate(List<Payment__c> updatedPayments) {
        try {
            for (Payment__c pay : updatedPayments) {
                if (pay.Status__c == 'Overdue') {
                    System.debug('Payment ' + pay.Id + ' is overdue. Creating recovery...');
                }
            }
        } catch (Exception ex) {
            String msg = ErrorHandler.logError(ex, 'PaymentHandler.afterUpdate');
            ErrorHandler.notifyAdmin('Payment Update Error', msg);
        }
    }
}
```

Test Classes

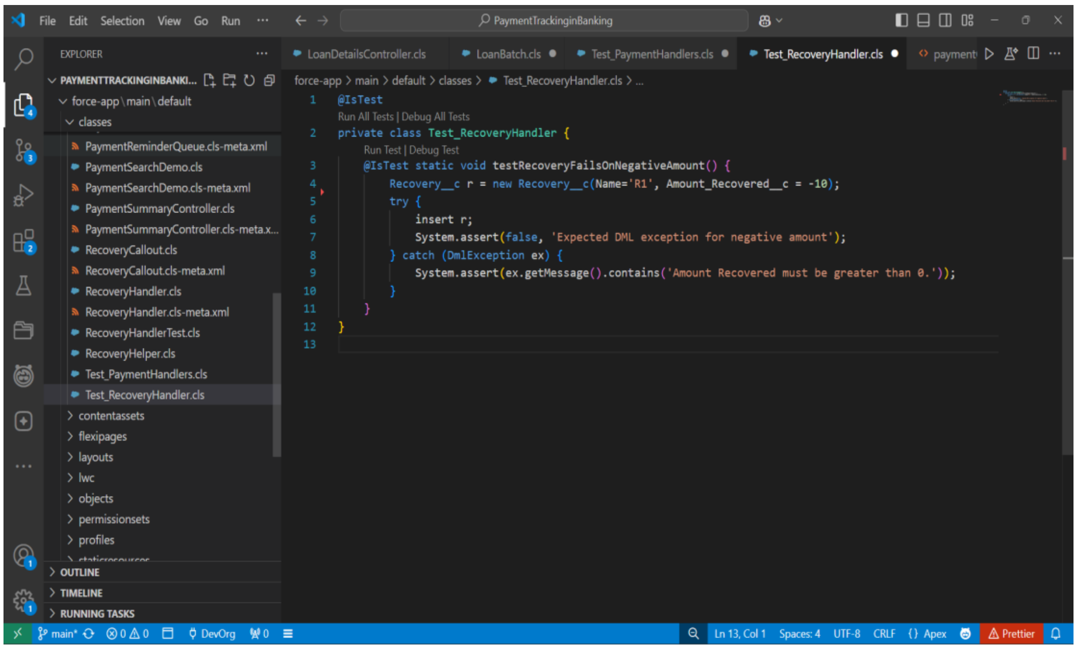
- Wrote test classes for each handler & trigger to validate functionality.


force-app > main > default > classes > RecoveryHandlerTest.cls > ...

```apex
@isTest
public class RecoveryHandlerTest {

    @isTest
    static void testRecoveryInsert() {
        Loan__c loan = new Loan__c(
            Principal_Amount__c = 10000,
            Term_Months__c = 12,
            Interest_Rate__c = 8
        );
        insert loan;

        Recovery__c rec = new Recovery__c(
            Loan__c = loan.Id,
            Amount_Recovered__c = 1000,
            Recovery_Date__c = Date.today()
        );

        Test.startTest();
        insert rec;
        Test.stopTest();

        System.assertEquals(1000, rec.Amount_Recovered__c);
    }
}
```


force-app > main > default > classes > Test_PaymentHandlers.cls > Test_PaymentHandlers > testPaymentOverdueBeforeInsert() : void

```apex
@IsTest
private class Test_PaymentHandlers {
    @IsTest static void testPaymentInsertAndLoanClose() {

        Loan__c loan = new Loan__c(Name='TestLoan', Principal_Amount__c = 1000.00, Status__c='Active');
        insert loan;


        Payment__c p1 = new Payment__c(Name='P1', Loan__c = loan.Id, Payment_Date__c = Date.today().adc
        Payment__c p2 = new Payment__c(Name='P2', Loan__c = loan.Id, Payment_Date__c = Date.today().adc

        insert new List<Payment__c>{p1,p2};


        p1.Status__c = 'Paid';
        p2.Status__c = 'Paid';
        update new List<Payment__c>{p1,p2};

        Loan__c refreshed = [SELECT Id, Status__c FROM Loan__c WHERE Id = :loan.Id];
        System.assertEquals('Closed', refreshed.Status__c);
    }

    @IsTest static void testPaymentOverdueBeforeInsert() {
        Loan__c loan = new Loan__c(Name='L2', Principal_Amount__c=1000.00);
        insert loan;

        Payment__c p = new Payment__c(Name='past', Loan__c = loan.Id, Payment_Date__c = Date.today().ac
```



```apex
@IsTest
private class Test_RecoveryHandler {
    @IsTest static void testRecoveryFailsOnNegativeAmount() {
        Recovery__c r = new Recovery__c(Name='R1', Amount_Recovered__c = -10);
        try {
            insert r;
            System.assert(false, 'Expected DML exception for negative amount');
        } catch (DmlException ex) {
            System.assert(ex.getMessage().contains('Amount Recovered must be greater than 0.'));
        }
    }
}
```

Asynchronous Processing

- Combined Batch, Queueable, Scheduled, and Future methods for async operations.
- Ensured the system scales for high-volume data in banking use cases.