Salesforce Property Management App

By Marthand Bhargav

Phase 5: Apex Programming (Developer)

Rental Management System - Payment Processing Module

---

1. Introduction

Importance of Apex Programming in Salesforce Development

Apex is Salesforce's proprietary, strongly-typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Salesforce platform. In this phase, we implement the business logic that powers our Rental Management System's payment processing capabilities.

Key Benefits:

- **Server-Side Execution**: All Apex code runs on Salesforce servers, ensuring data security
- **Automatic Governor Limit Enforcement**: Prevents resource monopolization in multi-tenant architecture
- **Integration Ready**: Seamlessly integrates with external payment gateways like Authorize.Net
- **Trigger-Based Automation**: Automatically calculates and updates financial rollups

Project Context: Our payment processing module handles credit card transactions, maintains financial rollups, and ensures data integrity across Statement, Payment, and Rental Agreement objects.

---

2. Concept Explanations

2.1 Classes & Objects

Definition: Classes are blueprints that define the structure and behavior of objects. Objects are instances of classes that hold actual data.

Class Body | Class Summary | Version Settings | Trace Flags

```
public class API_AuthorizeDotNet {
  //variables to hold our login credentials
  public static string APILOGIN;
  public static string APITRANSKEY;

    public static void getAuthNetCreds(){
    //get api login setting value
    //test condition to generate test key during test methods
    Authorize_net_Setting__c apiloginsetting =   Authorize_net_Setting__c.getInstance('API Login');
    //get transaction key seting value
    //test condition to generate test key during test methods
    Authorize_net_Setting__c apitranskeysetting =   Authorize_net_Setting__c.getInstance('TransKey');
    APILOGIN = apiloginsetting.value__c;
    APITRANSKEY = apitranskeysetting.value__c;
```

Example Explanation:

- API_AuthorizeDotNet is the main class handling payment gateway integration
- authnetReq_Wrapper is a nested inner class that encapsulates payment request data
- get;set; creates automatic getter and setter methods for properties

2.2 Triggers (Before/After Insert/Update/Delete)

Definition: Triggers are Apex code that executes before or after specific DML operations on Salesforce objects.

Trigger Context:

- **Before Triggers**: Used to validate or modify records before saving to database
- **After Triggers**: Used to access field values set by system (like Record ID) and affect changes in related records

Our Implementation:

```
trigger Payment_Trigger on Payment__c (after insert,after update) {

      if(trigger.isAfter && trigger.isInsert || trigger.isUpdate)
      {
          Rollup_Methods.rollupPaymentsToRentalAgreement(trigger.newMap);
      }
}
```

Why After Insert/Update?

- We need the Payment record ID after it's saved
- We're updating related Rental Agreement records (separate DML operation)
- Prevents recursion issues

Code Coverage: None API Version: 64

```
1  trigger Payment_Trigger on Payment__c (after insert,after update) {
2
3        if(trigger.isAfter && trigger.isInsert || trigger.isUpdate)
4        {
5            Rollup_Methods.rollupPaymentsToRentalAgreement(trigger.newMap);
6        }
7  }
```

2.3 Trigger Design Pattern

Best Practice: Handler Pattern

Our project follows a **delegated trigger pattern** where triggers delegate business logic to handler classes.

Benefits:

1. **Separation of Concerns**: Trigger contains only routing logic
2. **Reusability**: Handler methods can be called from other contexts
3. **Testability**: Easier to write unit tests for handler classes
4. **Maintainability**: Business logic centralized in one place

Pattern Structure:

Trigger → Handler Class → Business Logic Methods

Our Implementation:

- Payment_Trigger → Rollup_Methods.rollupPaymentsToRentalAgreement()
- Statement_Trigger → Rollup_Methods.rollupStatementsToRentalAgreement()

2.4 SOQL (Salesforce Object Query Language)

Definition: SOQL is used to query Salesforce database and retrieve records.

Key Features:

- Similar to SQL but designed for Salesforce's data model
- Supports relationship queries (parent-child, child-parent)
- Governor limited (50,000 records per transaction)

Examples from Our Code:

Basic Query with Relationship

```
Contact renter = [
    SELECT id, firstname, lastname, mailingstreet,
           mailingcity, mailingstate, mailingpostalcode
    FROM Contact
    WHERE id = :thisStatement.Rental_Agreement__r.renter__c
];
```

Aggregate Query (SUM with GROUP BY)

```
for(AggregateResult ar : [
    SELECT sum(Amount__c) totalPaid,
           Statement__r.Rental_Agreement__c ra
    FROM Payment__c
    WHERE Statement__r.Rental_Agreement__c IN :rentalAgreementSet
    GROUP BY Statement__r.Rental_Agreement__c
]) {
    // Process results
}
```

Best Practices in Our Code:

- Use WHERE clause with collections (:rentalAgreementSet)
- Aggregate queries to minimize DML operations
- Relationship queries to avoid multiple SOQL calls

2.5 Collections

Types Used in Our Project:

Set<Id>

```
Set<Id> rentalAgreementSet = new Set<Id>();
for(Payment__c p : [SELECT id, statement__r.Rental_Agreement__c
                    FROM Payment__c
                    WHERE Id IN :newMap.keySet()]) {
    rentalAgreementSet.add(p.statement__r.Rental_Agreement__c);
}
```

Purpose: Ensures unique Rental Agreement IDs, prevents duplicate processing

Map<String, String>

```
Map<string,string> messagestring = new map<String,String>();
messagestring.put('x_login', APILOGIN);
messagestring.put('x_tran_key', APITRANSKEY);
```

Purpose: Ensures unique Rental Agreement IDs, prevents duplicate processing

List<Rental_Agreement__c>

```
List<Rental_Agreement__c> rentalUpdates = new List<Rental_Agreement__c>
// ... populate list
if(!rentalUpdates.isEmpty()) {
    update rentalUpdates;
}
```

Purpose: Bulk DML operations (governor limit optimization)

2.6 Control Statements

If-Else Statements

```
if (trigger.isAfter && trigger.isInsert || trigger.isUpdate) {
    // Execute rollup logic
}
```

For Loop:

```
// Enhanced for loop (preferred for collections)
for(Statement__c s : newList) {
    rentalAgreementSet.add(s.Rental_agreement__c);
}

// Traditional for loop (used for building year options)
for (integer i=0; i<5; i++) {
    string y = ''+system.today().addyears(i).year();
    temp.add(new selectoption(y,y));
}
```

Ternary Operator:

```
req.firstname = (thisPayment.Billing_Name__c.contains(' '))
    ? thisPayment.Billing_Name__c.substringbefore(' ')
    : thisPayment.Billing_Name__c;
```

2.7 Exception Handling

Try-Catch Block in Email Sending:

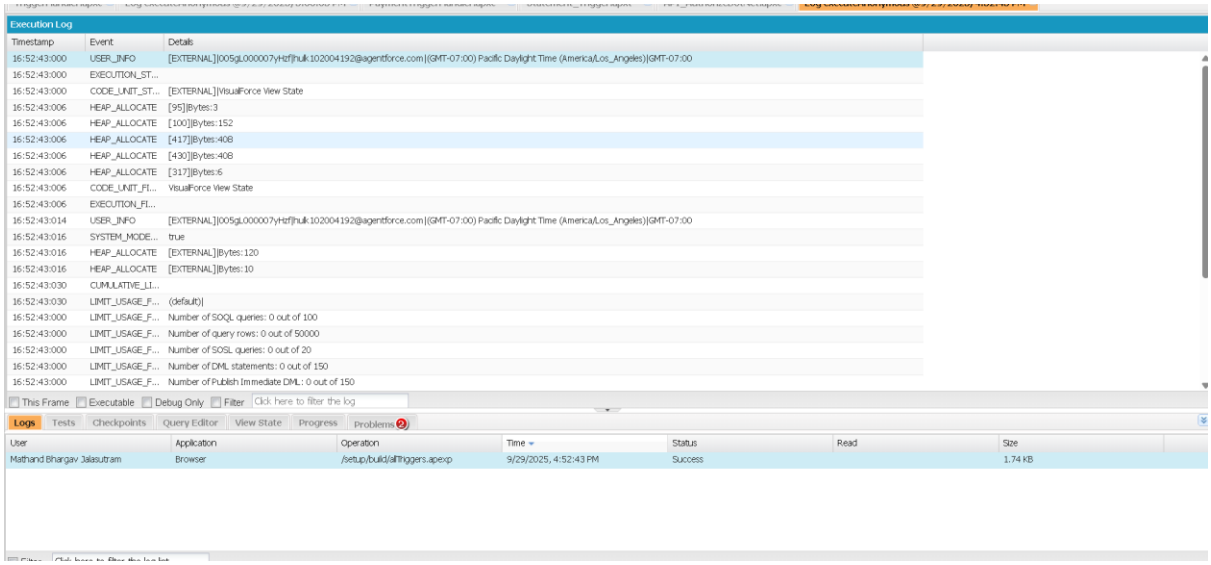
```
try {
    if (thisstatement ≠ null &&
        thisstatement.Rental_Agreement__r.renter__r.email ≠ null) {
        Messaging.SingleEmailMessage msg = new Messaging.SingleEmailMes
        // ... email configuration
        Messaging.sendEmail(new list<Messaging.SingleEmailMessage>{msg}
    }
}
catch(exception e) {
    // Silent failure - email is non-critical
}
```

2.8 HTTP Callouts (Integration)

Making External API Calls:

```
//Construct the request
HttpRequest req = new HttpRequest();
//for testing use the test endpoint
//otherwise use
//https://secure.authorize.net/gateway/transact.dll
   req.setEndpoint('https://test.authorize.net/gateway/transact.dll');
   req.setMethod('POST');

//build message
Map<string,string> messagestring = new map<String,String>();
```



Key Components:

1. HttpRequest: Configures endpoint, method, headers, body
2. EncodingUtil.urlEncode(): Prevents injection attacks
3. Http.send(): Executes callout (counts against governor limits)

3. Implementation Steps

Step 1: Create Custom Objects and Fields

Objects Required:

1. **Rental_Agreement__c**
    - Total_Invoiced__c (Currency)
    - Total_Payments__c (Currency)
    - Renter__c (Lookup to Contact)
2. **Statement__c**
    - Rental_Agreement__c (Master-Detail to Rental_Agreement__c)
    - Total_Amount__c (Currency)
    - Balance__c (Currency)
    - Statement_Date__c (Date)
3. **Payment__c**
    - Statement__c (Master-Detail to Statement__c)
    - Amount__c (Currency)
    - Status__c (Picklist: Paid, Failed, Pending)
    - Payment_Date__c (DateTime)
    - Billing_Name__c (Text)
    - Billing_Street__c (Text)
    - Billing_City__c (Text)
    - Billing_State__c (Text)
    - Billing_Postal_Code__c (Text)
    - Credit_Card_Number__c (Text, Encrypted)
    - Credit_Card_Expiration_Month__c (Picklist)
    - Expiration_Year__c (Text)
    - Credit_Card_Security_Card__c (Text, Encrypted)
    - Check_Account_Number__c (Text, Encrypted)
    - Check_Routing_Number__c (Text, Encrypted)

- Authorize_net_Transaction_ID__c (Text)
- Authorize_net_Authorization_Code__c (Text)
- Authorize_net_Response__c (Text)

[Insert Screenshot: Payment Object Field Layout] [Insert Screenshot: Rental Agreement Object Schema]

Step 2: Create Custom Settings for API Credentials

1. Navigate to **Setup** → **Custom Settings**
2. Click **New**
3. Create Authorize_net_Setting__c (Hierarchy type)
4. Add field: Value__c (Text, 255 characters)
5. Click **Manage** and add two records:
    - Name: "API Login", Value: [Your API Login ID]
    - Name: "TransKey", Value: [Your Transaction Key]

[Insert Screenshot: Custom Settings Definition] [Insert Screenshot: Authorize.Net Credentials Entry]

Step 3: Configure Remote Site Settings

1. Go to **Setup** → **Remote Site Settings**
2. Click **New Remote Site**
3. Configure:
    - Remote Site Name: AuthorizeNetTest
    - Remote Site URL: https://test.authorize.net
    - Active: ✓ Checked
4. Repeat for production: https://secure.authorize.net

[Insert Screenshot: Remote Site Settings Page]

Step 4: Create Record Types for Payment Methods

1. Navigate to **Setup** → **Object Manager** → **Payment__c**
2. Click **Record Types** → **New**
3. Create two record types:

    - **Credit Card**: For card payments
    - **Check**: For check payments
4. Assign different page layouts if needed

[Insert Screenshot: Payment Record Types Configuration]

Step 5: Deploy Apex Classes

5a. Create API_AuthorizeDotNet Class

1. Open **Developer Console** (Setup → Developer Console)
2. Click **File** → **New** → **Apex Class**
3. Name: API_AuthorizeDotNet
4. Paste the provided code from document
5. Click **File** → **Save**

[Insert Screenshot: Developer Console with API_AuthorizeDotNet Class]

5b. Create Rollup_Methods Class

1. Repeat steps above with name: Rollup_Methods
2. This class contains aggregation logic

[Insert Screenshot: Rollup_Methods Class in Developer Console]

Step 6: Deploy Triggers

6a. Create Payment_Trigger

1. In Developer Console: **File** → **New** → **Apex Trigger**
2. Object: Payment__c
3. Name: Payment_Trigger
4. Paste trigger code
5. Save

6b. Create Statement_Trigger

1. Repeat for Statement__c object
2. Name: Statement_Trigger

[Insert Screenshot: Trigger Deployment Screen] [Insert Screenshot: All Triggers List View]

Step 7: Test Data Setup

1. Create a Contact record (Renter)
2. Create a Rental Agreement linked to the Contact
3. Create a Statement linked to the Rental Agreement
4. Use Visualforce page to process a test payment

Sequence Flow :

```
1. User submits payment form (Visualforce Page)
   ↓
2. payments_Extension.savePayment() validates fields
   ↓
3. Creates authNetReq_Wrapper with payment details
   ↓
4. Calls API_AuthorizeDotNet.authdotnetCharge()
   ↓
5. HTTP callout to Authorize.Net API
   ↓
6. Response parsed into authNetResp_Wrapper
   ↓
7. Payment__c record inserted with transaction details
   ↓
8. Payment_Trigger fires (after insert)
   ↓
9. Rollup_Methods updates Rental Agreement totals
```

```
// Step 1: Create request wrapper
api_AuthorizeDotNet.authNetReq_Wrapper req =
    new api_AuthorizeDotNet.authNetReq_Wrapper();

// Step 2: Populate with form data
req.amt = string.valueof(thisPayment.Amount__c);
req.ccnum = thisPayment.Credit_Card_Number__c;
req.ccexp = monthmap.get(thisPayment.Credit_Card_Expiration_Month__c)
            + thisPayment.Expiration_Year__c;

// Step 3: Call payment gateway
api_AuthorizeDotNet.authNetResp_Wrapper res =
    api_AuthorizeDotNet.authdotnetCharge(req);

// Step 4: Check response
if (res.responseCode != '1') {
    thisPayment.Status__c = 'Failed';
    ApexPages.addMessage(new ApexPages.message(
        ApexPages.severity.error,
        'Payment Failed'
    ));
}
```

Example 2: Rollup Calculation Logic

Aggregate Query Pattern:

```
// Collect affected Rental Agreements
Set<Id> rentalAgreementSet = new Set<Id>();
for(Payment__c p : [SELECT id, statement__r.Rental_Agreement__c
                    FROM Payment__c
                    WHERE Id IN :newMap.keySet()]) {
    rentalAgreementSet.add(p.statement__r.Rental_Agreement__c);
}

// Perform aggregation
List<Rental_Agreement__c> rentalUpdates = new List<Rental_Agreement
for(AggregateResult ar : [
    SELECT sum(Amount__c) totalPaid,
           Statement__r.Rental_Agreement__c ra
    FROM Payment__c
    WHERE Statement__r.Rental_Agreement__c IN :rentalAgreementSet
    GROUP BY Statement__r.Rental_Agreement__c
]) {
    Rental_Agreement__c r = new Rental_Agreement__c(
        id = string.valueOf(ar.get('ra'))
    );
    r.Total_Payments__c = double.valueOf(ar.get('totalPaid'));
    rentalUpdates.add(r);
}

// Bulk update
if(!rentalUpdates.isEmpty()) {
    update rentalUpdates;
}
```

Why This Pattern Works:

- Minimizes SOQL queries (1 query for all affected records)
- Uses aggregate functions (server-side calculation)
- Bulk DML update (governor limit compliant)

5. Testing & Validation

```
@isTest
private class API_AuthorizeDotNet_Test {

    @testSetup
    static void setupTestData() {
        // Create test Custom Settings
        Authorize_net_Setting__c apiLogin = new Authorize_net_Setting__c(
            Name = 'API Login',
            Value__c = 'TEST_LOGIN'
        );
        insert apiLogin;

        Authorize_net_Setting__c transKey = new Authorize_net_Setting__c(
            Name = 'TransKey',
            Value__c = 'TEST_KEY'
        );
        insert transKey;

        // Create test data hierarchy
        Contact testRenter = new Contact(
            FirstName = 'John',
            LastName = 'Doe',
            Email = 'test@example.com'
        );
        insert testRenter;

        Rental_Agreement__c testAgreement = new Rental_Agreement__c(
            Renter__c = testRenter.Id
        );
        insert testAgreement;
```

```
    Statement__c testStatement = new Statement__c(
        Rental_Agreement__c = testAgreement.Id,
        Balance__c = 100.00
    );
    insert testStatement;
}

@isTest
static void testSuccessfulPayment() {
    // Set mock callout
    Test.setMock(HttpCalloutMock.class, new MockAuthorizeNetSuccess());

    Statement__c stmt = [SELECT Id FROM Statement__c LIMIT 1];

    // Create payment request
    API_AuthorizeDotNet.authnetReq_Wrapper req =
        new API_AuthorizeDotNet.authnetReq_Wrapper();
    req.ccnum = '4111111111111111'; // Test card
    req.ccexp = '1225'; // Dec 2025
    req.ccsec = '123';
    req.amt = '100.00';
    req.firstname = 'John';
    req.lastname = 'Doe';

    Test.startTest();
    API_AuthorizeDotNet.authNetResp_Wrapper resp =
        API_AuthorizeDotNet.authdotnetCharge(req);
    Test.stopTest();

    // Assertions
    System.assertEquals('1', resp.responseCode,
        'Payment should be approved');
    System.assertEquals('This transaction has been approved.',
```

```
        resp.ResponseReasonText);
    }

    @isTest
    static void testFailedPayment() {
        Test.setMock(HttpCalloutMock.class, new MockAuthorizeNetFailure());

        API_AuthorizeDotNet.authnetReq_Wrapper req =
            new API_AuthorizeDotNet.authnetReq_Wrapper();
        req.ccnum = '4111111111111111';
        req.ccexp = '1225';
        req.ccsec = '123';
        req.amt = '100.00';

        Test.startTest();
        API_AuthorizeDotNet.authNetResp_Wrapper resp =
            API_AuthorizeDotNet.authdotnetCharge(req);
        Test.stopTest();

        System.assertNotEquals('1', resp.responseCode,
            'Payment should fail');
    }

    @isTest
    static void testBulkPaymentRollup() {
        // Test bulk trigger processing (200 records)
        Statement__c stmt = [SELECT Id, Rental_Agreement__c
                             FROM Statement__c LIMIT 1];

        List<Payment__c> payments = new List<Payment__c>();
        for(Integer i = 0; i < 200; i++) {
            payments.add(new Payment__c(
                Statement__c = stmt.Id,
                Amount__c = 10.00,
```

```
        Status__c = 'Paid',
        Payment_Date__c = System.now()
    ));
  }

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

  // Verify rollup calculation
  Rental_Agreement__c ra = [SELECT Total_Payments__c
                    FROM Rental_Agreement__c
                    WHERE Id = :stmt.Rental_Agreement__c];
  System.assertEquals(2000.00, ra.Total_Payments__c,
    'Total should be 200 * 10.00');
}
```

```
}
```

5.2 Mock Callout Classes

```
@isTest
global class MockAuthorizeNetSuccess implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest req) {
        HttpResponse res = new HttpResponse();
        res.setHeader('Content-Type', 'text/plain');
        res.setBody('1;1;1;This transaction has been approved.;AUTH123;Y;'
                    '123456789;INV001;Payment;100.00;CC;AUTH_CAPTURE;;;;' +
                    'John;Doe;;123 Main St;City;ST;12345;US;;;;;;;;;;;;' +
                    '0.00;0.00;0.00;FALSE;;HASH123;M;;XXXX1111;Visa;;;');
        res.setStatusCode(200);
        return res;
    }
}

@isTest
global class MockAuthorizeNetFailure implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest req) {
        HttpResponse res = new HttpResponse();
        res.setBody('2;1;2;This transaction has been declined.;;;;;' +
                    ';;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;');
        res.setStatusCode(200);
        return res;
    }
}
```

6.3 Running Tests

Via Developer Console:

1. Open Developer Console
2. Click Test → New Run
3. Select test classes
4. Click Run
5. View results in Tests tab

Via Command Line (SFDX):

sfdx force:apex:test:run -n API_AuthorizeDotNet_Test -r human -c

Estimate your organization's code coverage

Compile all classes

View: All Create New View

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Other | All

Developer Console | New | Generate from WSDL | Run All Tests | Schedule Apex

Action	Name	Namespace Prefix	Api Version	Status	Size Without Comments	Last Modified By	Has Trace Flags
Edit | Del | Security	API_AuthorizeDotNet		64.0	Active	6,747	Mathand Bhargav Jalasutram, 9/25/2025, 7:14 AM	
Edit | Del | Security	payments_Extension		64.0	Active	36	Mathand Bhargav Jalasutram, 9/25/2025, 10:49 PM	
Edit | Del | Security	PaymentTriggerHandler		64.0	Active	39	Mathand Bhargav Jalasutram, 9/29/2025, 2:45 AM	
Edit | Del | Security	Rollup_Methods		64.0	Active	1,874	Mathand Bhargav Jalasutram, 9/29/2025, 5:13 AM	
Edit | Del | Security	TriggerHandler		64.0	Active	2,042	Mathand Bhargav Jalasutram, 9/29/2025, 2:57 AM	

Dynamic Apex Classes

Dynamic Apex extends your programming reach by interacting with Lightning Platform components.

View: All Create New View

Class Name	Namespace Prefix	Api Version	Created By	Last Modified By
No records to display.				

Percent of Apex Used: 0.18%
You are currently using 10,858 characters of Apex Code (excluding comments and @isTest annotated classes) in your organization, out of an allowed limit of 6,000,000 characters. Note that the amount in use includes both Apex Classes and Triggers defined in your organization.

Compile all triggers

View: All Create New View

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Other | All

Developer Console

Action	Name	Namespace Prefix	sObject Type	Api Version	Status	Size Without Comments	Last Modified By	Has Trace Flags
Edit | Del	Payment_Trigger		Payment	64.0	Active	58	Mathand Bhargav Jalasutram, 9/29/2025, 4:18 AM	
Edit | Del	Statement_Trigger		Statement	64.0	Active	62	Mathand Bhargav Jalasutram, 9/29/2025, 4:25 AM	