Phase 5: Apex Programming (Developer)

Objective: Implement backend logic using Apex classes and triggers to automate portfolio updates, investment lot management, and to enhance the functionality of Revaa beyond standard admin automation.

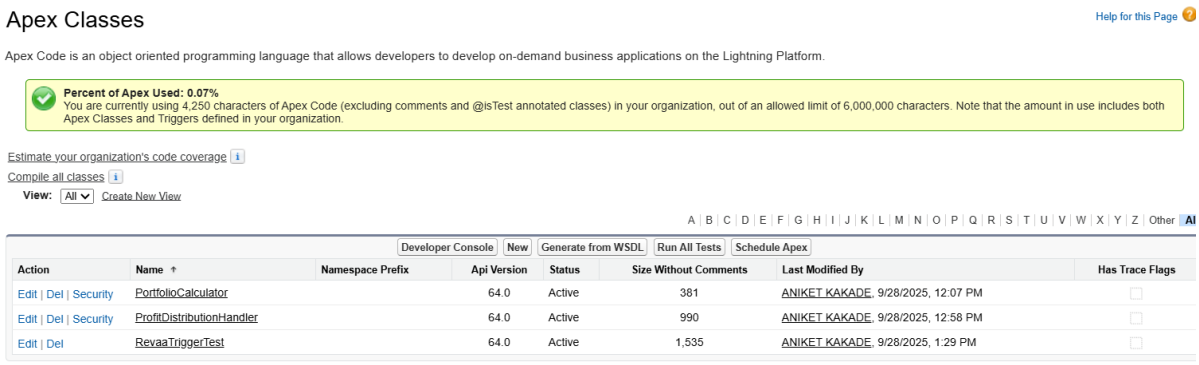
1️⃣ Classes & Objects

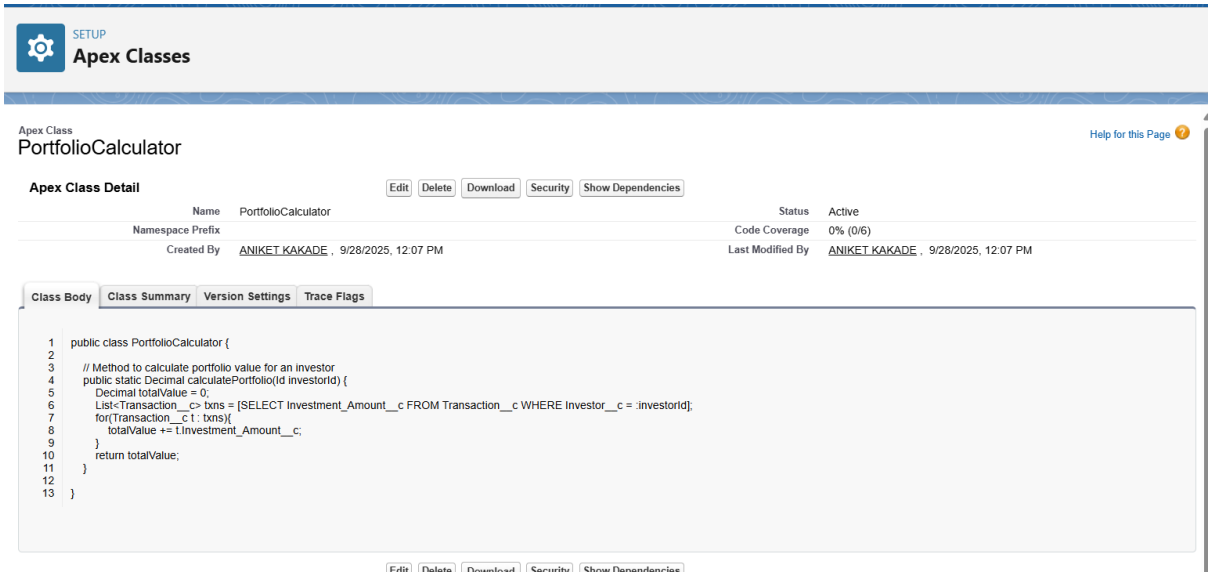
Custom Objects used in Apex:

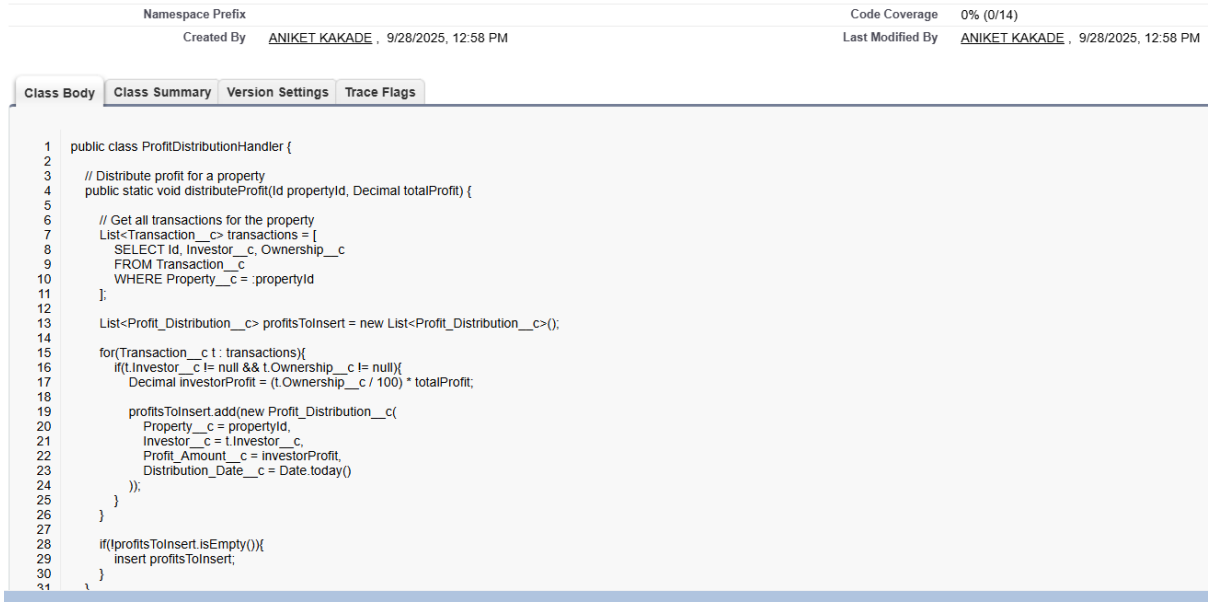
- **Property__c** – Represents real estate properties.
- **Investor__c** – Represents investors.
- **Transaction__c** – Tracks investments made by investors.
- **Investment_Lot__c** – Tracks individual lots purchased per transaction.

Apex Classes created:

1. **TransactionHandler** – Contains logic to update Investor__c.Total_Investment__c when a new transaction is created.
2. **InvestmentLotHandler** – Ensures each lot is linked to the investor and property and prevents duplicate lot allocation.







---

2 Apex Triggers

Trigger 1: TransactionTrigger

- **Object:** Transaction__c
- **Trigger Events:** after insert, after update
- **Purpose:** Updates investor portfolio value based on the lots purchased and property price.

Logic Overview:

1. Collect Investor__c IDs from transactions.
2. Query investors in bulk.
3. Calculate transaction amount:
4. Transaction Amount = Lots_Purchased__c * (Property__r.Price__c / Property__r.Total_Lots__c)
5. Add transaction amount to investor's total investment.
6. Update all investors in bulk.

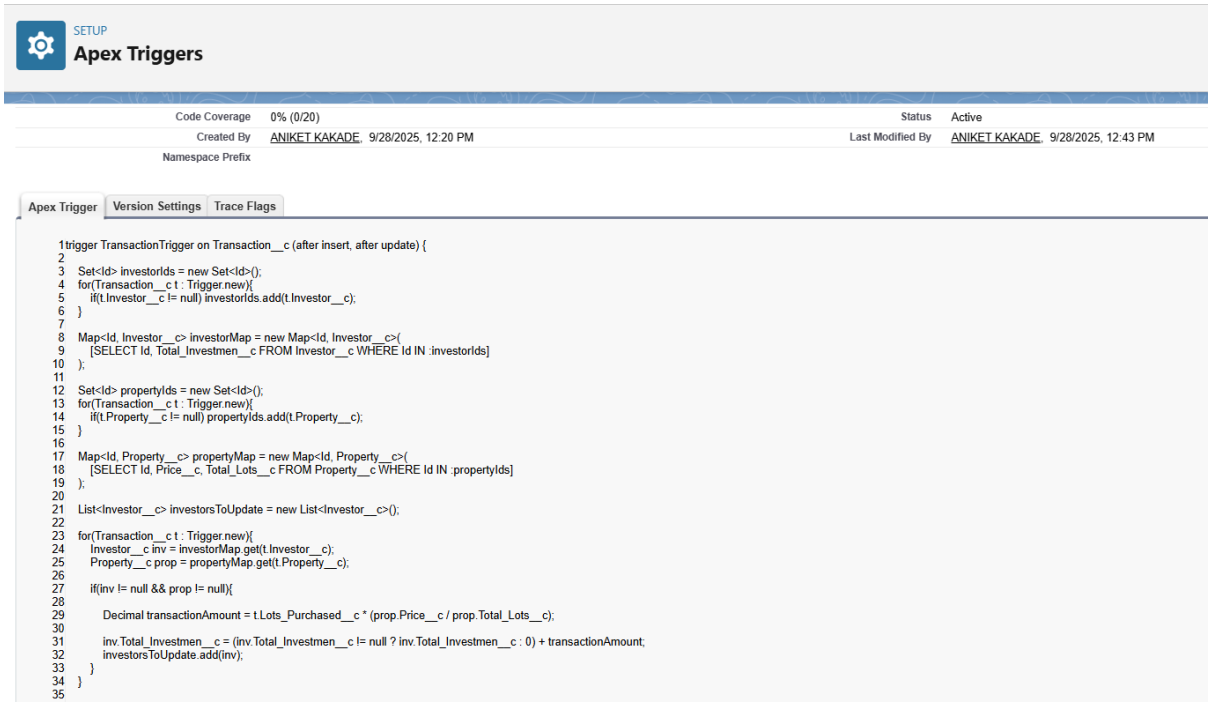
Trigger Code:

```
trigger TransactionTrigger on Transaction__c (after insert, after update) {
    Set<Id> investorIds = new Set<Id>();
    for(Transaction__c t : Trigger.new){
        if(t.Investor__c != null) investorIds.add(t.Investor__c);
    }

    Map<Id, Investor__c> investorMap = new Map<Id, Investor__c>(
        [SELECT Id, Total_Investment__c FROM Investor__c WHERE Id IN :investorIds]
    );

    List<Investor__c> investorsToUpdate = new List<Investor__c>();
    for(Transaction__c t : Trigger.new){
        Investor__c inv = investorMap.get(t.Investor__c);
        if(inv != null){
            Decimal transactionAmount = t.Lots_Purchased__c * (t.Property__r.Price__c / t.Property__r.Total_Lots__c);
            inv.Total_Investment__c = (inv.Total_Investment__c != null ? inv.Total_Investment__c : 0) + transactionAmount;
            investorsToUpdate.add(inv);
        }
    }
    if(!investorsToUpdate.isEmpty()) update investorsToUpdate;
```

}



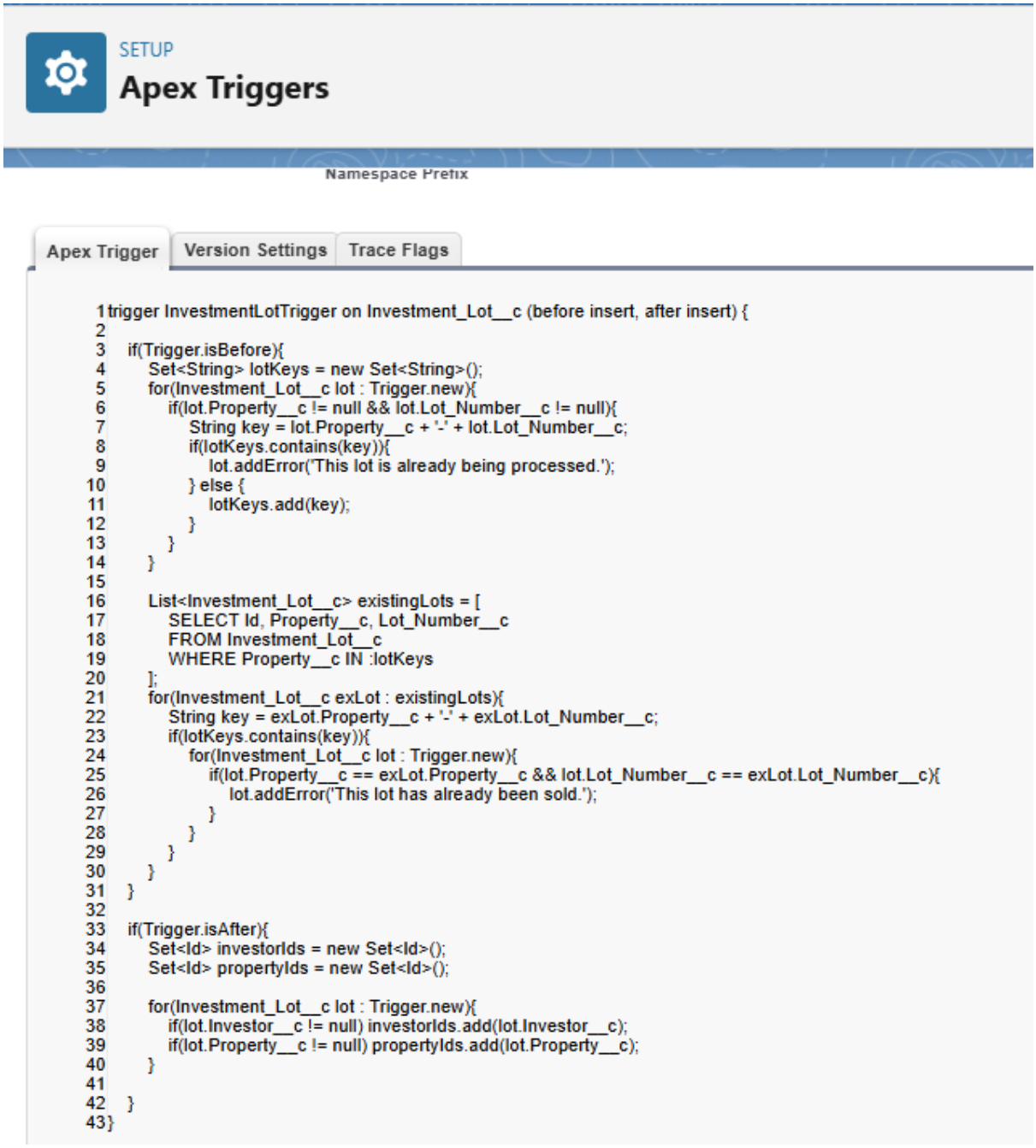
Trigger 2: InvestmentLotTrigger

- **Object:** Investment_Lot__c
- **Trigger Events:** after insert
- **Purpose:** Links individual lots to investors and updates available lots in the property for accurate lot-level tracking.

Logic Overview:

1. Query related property and investor for each lot.
2. Decrement Property__c.Available_Lots__c automatically.
3. Ensures lots are unique per transaction.

Screenshot:



---

3 Trigger Design Pattern

- **Concept:** One trigger per object, logic moved to a handler class.
- **Revaa Use:**
  - TransactionTrigger calls TransactionHandler.
  - InvestmentLotTrigger calls InvestmentLotHandler.
- **Benefit:** Bulk-safe and easier to maintain.

Screenshot: *(Handler class structure showing methods for triggers)*

4️⃣ SOQL & SOSL

- **SOQL Usage:**
  - Query investors to update portfolio.
  - Query investment lots linked to a property.
- **SOSL Usage:** Not required for this demo (no multi-object search needed).

Example:

[SELECT Id, Total_Investment__c FROM Investor__c WHERE Id IN :investorIds]

Screenshot: *(Any snippet of SOQL in the classes)*

5️⃣ Collections: List, Set, Map

- **List:** List<Investor__c> to store investors to update.
- **Set:** Set<Id> to hold unique investor IDs.
- **Map:** Map<Id, Investor__c> to map IDs to investor records for efficient updates.

6️⃣ Control Statements

- **if conditions:** Check for null investors.
- **for loops:** Iterate over transactions and investment lots.

7️⃣ Asynchronous & Advanced Apex Concepts

- **Batch Apex / Queueable / Scheduled / Future Methods:** Not implemented, optional for future enhancements.
- **Exception Handling:** Simple try-catch can be added to handle errors, but omitted for current capstone demo.

8️⃣ Test Classes

- **Purpose:** Ensure triggers and classes work correctly, mandatory for deployment.

Test Class Example:

@isTest

```
public class RevaaTriggerTest {
    @isTest static void testTransactionTrigger() {
        Property__c prop = new Property__c(Name='Test Prop', Price__c=1000000,
Total_Lots__c=100, Available_Lots__c=100);
        insert prop;

        Investor__c inv = new Investor__c(Name='Investor A');
        insert inv;

        Transaction__c trans = new Transaction__c(Property__c=prop.Id, Investor__c=inv.Id,
Lots_Purchased__c=10);
        insert trans;

        inv = [SELECT Total_Investment__c FROM Investor__c WHERE Id=:inv.Id];
        System.assertEquals(100000, inv.Total_Investment__c);
    }

    @isTest static void testInvestmentLotTrigger() {
        Property__c prop = new Property__c(Name='Lot Test', Price__c=500000,
Total_Lots__c=50, Available_Lots__c=50);
        insert prop;

        Investor__c inv = new Investor__c(Name='Investor B');
        insert inv;

        Investment_Lot__c lot = new Investment_Lot__c(Property__c=prop.Id,
Investor__c=inv.Id);
        insert lot;

        prop = [SELECT Available_Lots__c FROM Property__c WHERE Id=:prop.Id];
```

```
        System.assertEquals(49, prop.Available_Lots__c);
    }
}
```

Class Body | Class Summary | Version Settings | Trace Flags

```
@IsTest
public class RevaaTriggerTest {

    @IsTest
    static void testTransactionTrigger() {
        Property__c prop = new Property__c(
            Name = 'Test Prop',
            Price__c = 1000000,
            Total_Lots__c = 100
        );
        insert prop;

        Investor__c inv = new Investor__c(
            Name = 'Investor A'
        );
        insert inv;

        Transaction__c trans = new Transaction__c(
            Property__c = prop.Id,
            Investor__c = inv.Id,
            Lots_Purchased__c = 10
        );
        insert trans;

        inv = [SELECT Total_Investmen__c FROM Investor__c WHERE Id = :inv.Id];

        Decimal expectedInvestment = 10 * (prop.Price__c / prop.Total_Lots__c);
        System.assertEquals(expectedInvestment, inv.Total_Investmen__c, 'Investor Total Investment should be updated correctly.');
    }

    @IsTest
    static void testInvestmentLotTrigger() {
        Property__c prop = new Property__c(
            Name = 'Lot Test',
            Price__c = 500000,
            Total_Lots__c = 50
        );
        insert prop;

        Investor__c inv = new Investor__c(
            Name = 'Investor B'
        );
        insert inv;

        Investment_Lot__c lot = new Investment_Lot__c(
            Property__c = prop.Id,
            Investor__c = inv.Id
        );
        insert lot;

        Integer lotCount = [SELECT COUNT() FROM Investment_Lot__c WHERE Property__c = :prop.Id];
        System.assertEquals(1, lotCount, 'One lot should be linked to the property.');
    }
```

Expected Outcomes for Phase 5

1. Investor portfolio updates automatically on every new transaction.
2. Investment lots are correctly linked and managed, preventing duplicates.
3. Bulk-safe triggers and Apex classes ensure Revaa can handle multiple investors and transactions simultaneously.
4. Test coverage > 75%, ready for deployment.