TITLE: “Insurance Claim Fraud Detection – AI-Powered Monitoring & Alert System”

Phase 5: Apex Programming (Developer)

Goal: Implement server-side logic to support complex/frequent operations that cannot be handled (or are inefficient) with clicks alone — bulk-safe, test-covered, and maintainable code for fraud detection and claim processing.

1. Classes & Objects:

Purpose

Contain business logic in reusable, testable units (Apex classes) and keep triggers thin.

Implementation

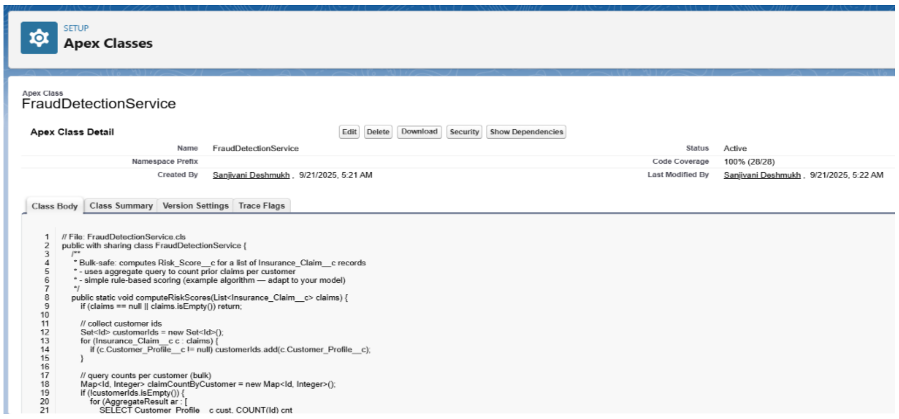
- **FraudDetectionService** — core business logic to calculate **Risk_Score__c** for **Insurance_Claim__c** records in bulk.
- Small utility classes where needed (e.g., email helpers, constants).

Why

- **Encapsulation:** single place to change scoring logic.
- **Reusability:** service can be called from triggers, batches, schedulers, tests.
- **Testability:** easy to unit test pure logic.

Implementation steps

1. Create a new Apex Class in Developer Console (File → New → Apex Class) named **FraudDetectionService.**
2. Implement a bulk-safe public static method like **computeRiskScores**(**List<Insurance_Claim__c>** claims) that updates **Risk_Score__c** in-memory.
3. Keep no DML inside loops; use aggregate queries to fetch supporting data.



Name FraudDetectionService
Namespace Prefix
Created By Sanjivani Deshmukh , 9/21/2025, 5:21 AM
Status Active
Code Coverage 100% (28/28)
Last Modified By Sanjivani Deshmukh , 9/21/2025, 5:22 AM

```
// File: FraudDetectionService.cls
public with sharing class FraudDetectionService {
    /**
     * Bulk-safe: computes Risk_Score__c for a list of Insurance_Claim__c records
     * - uses aggregate query to count prior claims per customer
     * - simple rule-based scoring (example algorithm — adapt to your model)
     */
    public static void computeRiskScores(List<Insurance_Claim__c> claims) {
        if (claims == null || claims.isEmpty()) return;

        // collect customer ids
        Set<Id> customerIds = new Set<Id>();
        for (Insurance_Claim__c c : claims) {
            if (c.Customer_Profile__c != null) customerIds.add(c.Customer_Profile__c);
        }

        // query counts per customer (bulk)
        Map<Id, Integer> claimCountByCustomer = new Map<Id, Integer>();
        if (!customerIds.isEmpty()) {
            for (AggregateResult ar : [
                SELECT Customer_Profile__c cust, COUNT(Id) cnt
```

2. Apex Triggers (before/after insert/update/delete)

Purpose

React to DML events to run server logic before or after records are saved.

Implementation

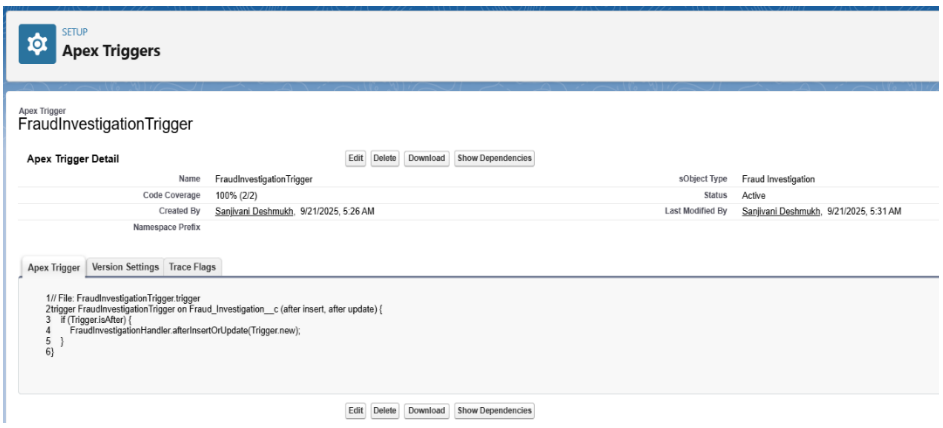
- **InsuranceClaimTrigger** (before insert, before update, after insert, after update) — delegates to handler.
    - Before: call **FraudDetectionService.computeRiskScores()** to set **Risk_Score__c**.
    - After insert/update: create **Fraud_Investigation__c** if claim is flagged/high-risk.
- **FraudInvestigationTrigger** (after insert, after update) — delegates to handler to update related claim(s) status and enqueue notifications.

Why

- before triggers update fields on the same record prior to DML (efficient).
- after triggers create related records (need Ids) and perform cross-object updates.

Implementation steps

1. Create trigger file (Developer Console → New → Apex Trigger). only delegate to a handler.
2. Create a handler class **InsuranceClaimTriggerHandler** and **FraudInvestigationHandler** with public static methods for each event.
3. Ensure bulk-safe coding: pass List<...> into handler methods and process collections.

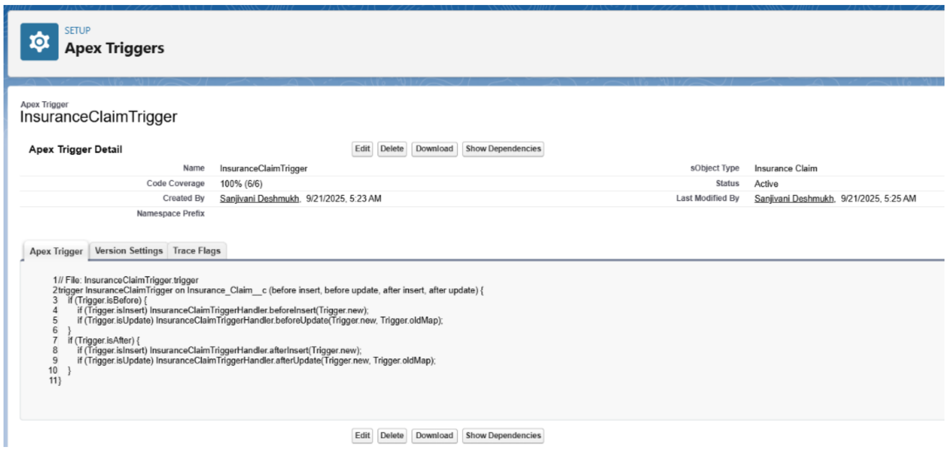

SETUP
Apex Triggers

Apex Trigger
FraudInvestigationTrigger

Apex Trigger Detail

Name	FraudInvestigationTrigger	sObject Type	Fraud Investigation
Code Coverage	100% (2/2)	Status	Active
Created By	Sanjivani Deshmukh, 9/21/2025, 5:26 AM	Last Modified By	Sanjivani Deshmukh, 9/21/2025, 5:31 AM
Namespace Prefix			

```
// File: FraudInvestigationTrigger.trigger
trigger FraudInvestigationTrigger on Fraud_Investigation__c (after insert, after update) {
  if (Trigger.isAfter) {
    FraudInvestigationHandler.afterInsertOrUpdate(Trigger.new);
  }
}
```

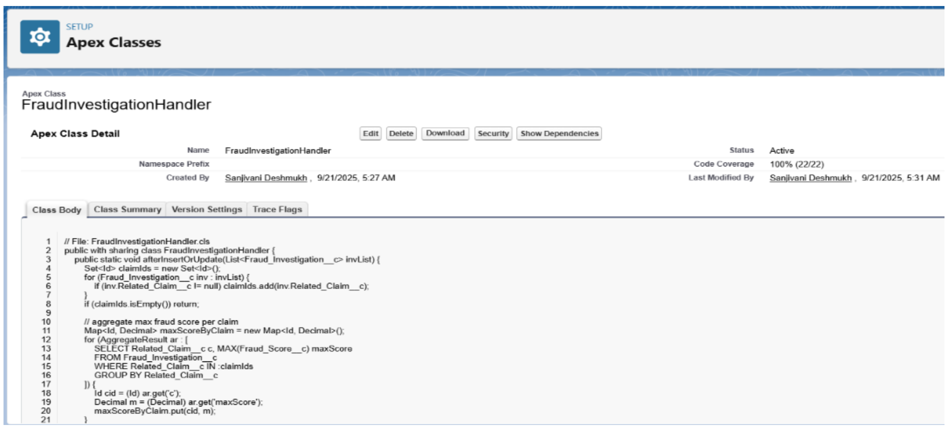

SETUP
Apex Triggers

Apex Trigger
InsuranceClaimTrigger

Apex Trigger Detail

Name	InsuranceClaimTrigger	sObject Type	Insurance Claim
Code Coverage	100% (6/6)	Status	Active
Created By	Sanjivani Deshmukh, 9/21/2025, 5:23 AM	Last Modified By	Sanjivani Deshmukh, 9/21/2025, 5:25 AM
Namespace Prefix			

```
// File: InsuranceClaimTrigger.trigger
trigger InsuranceClaimTrigger on Insurance_Claim__c (before insert, before update, after insert, after update) {
  if (Trigger.isBefore) {
    if (Trigger.isInsert) InsuranceClaimTriggerHandler.beforeInsert(Trigger.new);
    if (Trigger.isUpdate) InsuranceClaimTriggerHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
  }
  if (Trigger.isAfter) {
    if (Trigger.isInsert) InsuranceClaimTriggerHandler.afterInsert(Trigger.new);
    if (Trigger.isUpdate) InsuranceClaimTriggerHandler.afterUpdate(Trigger.new, Trigger.oldMap);
  }
}
```


SETUP
Apex Classes

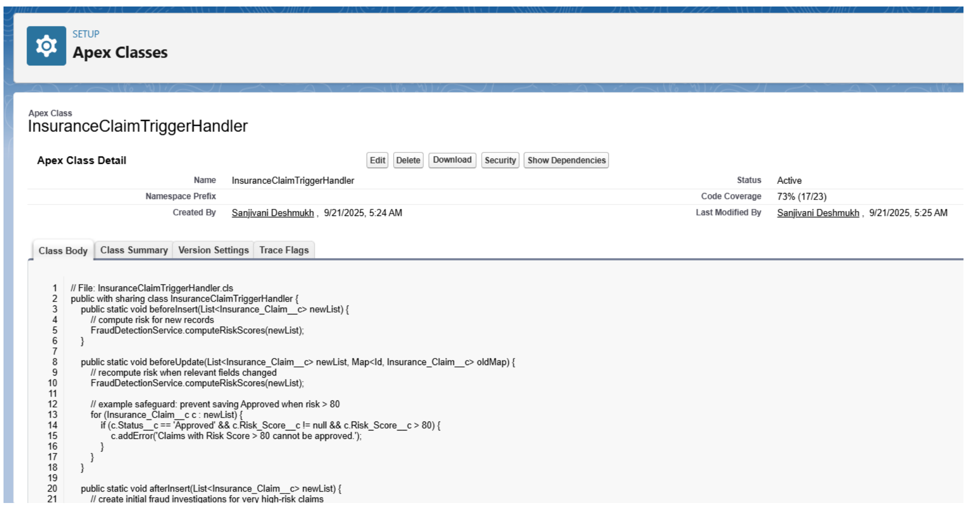
Apex Class
FraudInvestigationHandler

Apex Class Detail

Name	FraudInvestigationHandler	Status	Active
Namespace Prefix		Code Coverage	100% (22/22)
Created By	Sanjivani Deshmukh , 9/21/2025, 5:27 AM	Last Modified By	Sanjivani Deshmukh , 9/21/2025, 5:31 AM

```
// File: FraudInvestigationHandler.cls
public with sharing class FraudInvestigationHandler {
  public static void afterInsertOrUpdate(List<Fraud_Investigation__c> invList) {
    Set<Id> claimIds = new Set<Id>();
    for (Fraud_Investigation__c inv : invList) {
      if (inv.Related_Claim__c != null) claimIds.add(inv.Related_Claim__c);
    }
    if (claimIds.isEmpty()) return;

    // aggregate max fraud score per claim
    Map<Id, Decimal> maxScoreByClaim = new Map<Id, Decimal>();
    for (AggregateResult ar : [
      SELECT Related_Claim__c c, MAX(Fraud_Score__c) maxScore
      FROM Fraud_Investigation__c
      WHERE Related_Claim__c IN :claimIds
      GROUP BY Related_Claim__c
    ]) {
      Id cid = (Id) ar.get('c');
      Decimal m = (Decimal) ar.get('maxScore');
      maxScoreByClaim.put(cid, m);
    }
```


SETUP
Apex Classes

Apex Class
InsuranceClaimTriggerHandler

Apex Class Detail

Name	InsuranceClaimTriggerHandler	Status	Active
Namespace Prefix		Code Coverage	73% (17/23)
Created By	Sanjivani Deshmukh , 9/21/2025, 5:24 AM	Last Modified By	Sanjivani Deshmukh , 9/21/2025, 5:25 AM

```
// File: InsuranceClaimTriggerHandler.cls
public with sharing class InsuranceClaimTriggerHandler {
  public static void beforeInsert(List<Insurance_Claim__c> newList) {
    // compute risk for new records
    FraudDetectionService.computeRiskScores(newList);
  }

  public static void beforeUpdate(List<Insurance_Claim__c> newList, Map<Id, Insurance_Claim__c> oldMap) {
    // recompute risk when relevant fields changed
    FraudDetectionService.computeRiskScores(newList);

    // example safeguard: prevent saving Approved when risk > 80
    for (Insurance_Claim__c c : newList) {
      if (c.Status__c == 'Approved' && c.Risk_Score__c != null && c.Risk_Score__c > 80) {
        c.addError('Claims with Risk Score > 80 cannot be approved.');
      }
    }
  }

  public static void afterInsert(List<Insurance_Claim__c> newList) {
    // create initial fraud investigations for very high-risk claims
```

3. Trigger Design Pattern:

Purpose

Organize trigger logic to be maintainable, testable and avoid duplication.

Implementation

- Thin trigger files delegating to a ***Handler** class. Handler methods are organized by context (**beforeInsert, beforeUpdate, afterInsert, afterUpdate**).

Why

- Keeps triggers small and readable.
- Easier to unit test handler methods.
- Supports reuse and avoids mixed responsibilities.

Implementation notes

- Handler methods accept collections (e.g., **List<Insurance_Claim__c> newList**), and when needed **Map<Id, SObject> oldMap**.
- Add a central Handler class that calls the **FraudDetectionService** or other service classes.

4. SOQL & SOSL:

Purpose

Query data from Salesforce; SOSL for text search across objects.

Implementation

- SOQL aggregated query to count prior claims per customer (used in scoring).
- SOQL to fetch Users (e.g., managers) when sending notifications.
- No heavy SOSL needed in current scope; mention available for future search use-cases.

Why

- Required to compute counts and to find related records (**Fraud_Investigation__c** aggregates or find managers).

Implementation steps

- Use aggregate queries for counts:

- List<AggregateResult> arList = [SELECT Customer_Profile__c c, COUNT(Id) cnt

  FROM Insurance_Claim__c WHERE Customer_Profile__c IN :customerIds GROUP BY Customer_Profile__c];

  - Always limit fields and records returned; use **WHERE** clauses to reduce data.

5. Collections: List, Set, Map:

Purpose

Efficiently store and manipulate data in bulk operations.

Used

- **Set<Id>** to collect unique **Customer_Profile__c ids**.
- **Map<Id, Integer>** to store counts per customer.
- **List<Insurance_Claim__c>** for DML operations.

Why

- Prevent duplicates (Set), fast lookup (Map), ordered DML (List).

Implementation notes

- Build sets from trigger records, then query using IN :set.
- Map aggregate results to a map for O(1) lookup.

6. Control Statements:

Purpose

Implement decision logic (if/else, for, while, switch).

Used

- if-else to apply score buckets and to decide when to create investigations.
- for loops to iterate over collections.

Why

- Core to any logic; used for scoring rules and record transformations.

7. Batch Apex:

Purpose

Process large volumes of records asynchronously within governor limits.

Implementation

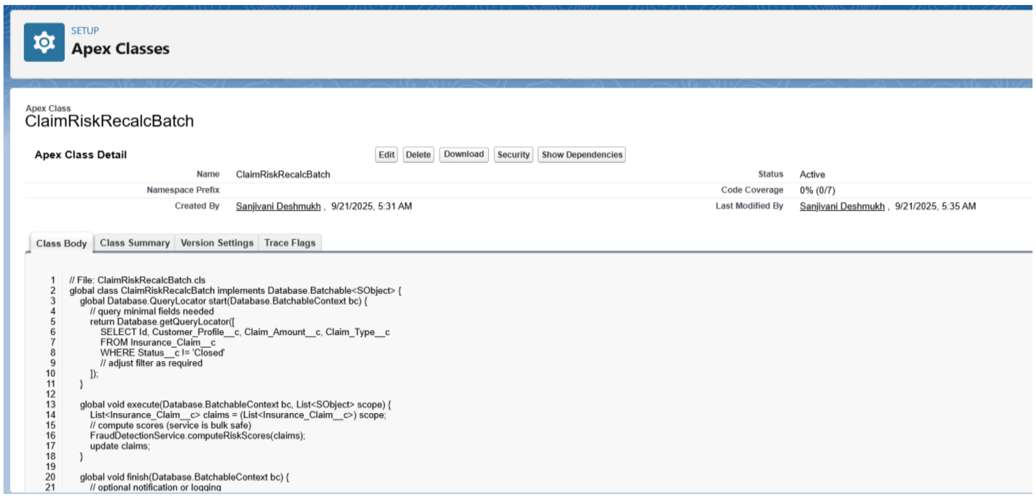
- **ClaimRiskRecalcBatch** — a **Database.Batchable<SObject>** implementation that re-calculates risk scores for claims in batches (chunk size e.g., 200).

Why

- If you have hundreds/thousands of claims, batch jobs let you recompute scores without hitting limits.

Implementation steps

1. Create batch class implementing start, execute, finish.
2. start returns a **Database.getQueryLocator()** with the claims to process.
3. execute calls **FraudDetectionService.computeRiskScores()** and updates chunk.
4. Schedule the batch via **System.schedule()** or a scheduled Apex class.



Apex Class
ClaimRiskRecalcBatch

Apex Class Detail

Name	ClaimRiskRecalcBatch	Status	Active
Namespace Prefix		Code Coverage	0% (0/7)
Created By	Sanjivani Deshmukh , 9/21/2025, 5:31 AM	Last Modified By	Sanjivani Deshmukh , 9/21/2025, 5:35 AM

```
// File: ClaimRiskRecalcBatch.cls
global class ClaimRiskRecalcBatch implements Database.Batchable<SObject> {
    global Database.QueryLocator start(Database.BatchableContext bc) {
        // query minimal fields needed
        return Database.getQueryLocator([
            SELECT Id, Customer_Profile__c, Claim_Amount__c, Claim_Type__c
            FROM Insurance_Claim__c
            WHERE Status__c != 'Closed'
            // adjust filter as required
        ]);
    }

    global void execute(Database.BatchableContext bc, List<SObject> scope) {
        List<Insurance_Claim__c> claims = (List<Insurance_Claim__c>) scope;
        // compute scores (service is bulk safe)
        FraudDetectionService.computeRiskScores(claims);
        update claims;
    }

    global void finish(Database.BatchableContext bc) {
        // optional notification or logging
```

8. Queueable Apex:

Purpose

Asynchronous job for medium complexity tasks (chaining allowed) — lighter than Batch for smaller jobs.

Implementation

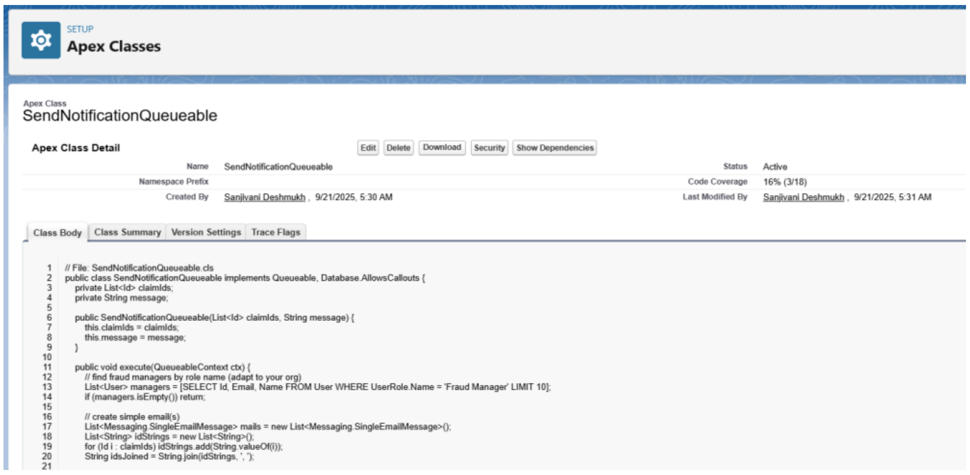
- **SendNotificationQueueable** to send emails or callouts asynchronously after fraud threshold reached.

Why

- Offloads email sending or callouts from trigger context to avoid long-running operations and limits.

Implementation notes

- Implement Queueable and call **System.enqueueJob(new SendNotificationQueueable(ids, message))**.
- If callouts are needed in queueable, implement **Database.AllowsCallouts.**



Apex Class
SendNotificationQueueable

Apex Class Detail

Name	SendNotificationQueueable	Status	Active
Namespace Prefix		Code Coverage	16% (3/18)
Created By	Sanjivani Deshmukh , 9/21/2025, 5:30 AM	Last Modified By	Sanjivani Deshmukh , 9/21/2025, 5:31 AM

```
// File: SendNotificationQueueable.cls
public class SendNotificationQueueable implements Queueable, Database.AllowsCallouts {
  private List<Id> claimIds;
  private String message;

  public SendNotificationQueueable(List<Id> claimIds, String message) {
    this.claimIds = claimIds;
    this.message = message;
  }

  public void execute(QueueableContext ctx) {
    // find fraud managers by role name (adapt to your org)
    List<User> managers = [SELECT Id, Email, Name FROM User WHERE UserRole.Name = 'Fraud Manager' LIMIT 10];
    if (managers.isEmpty()) return;

    // create simple email(s)
    List<Messaging.SingleEmailMessage> mails = new List<Messaging.SingleEmailMessage>();
    List<String> idStrings = new List<String>();
    for (Id i : claimIds) idStrings.add(String.valueOf(i));
    String idsJoined = String.join(idStrings, ', ');

```

9. Scheduled Apex

Purpose

Run jobs on a schedule (daily/nightly).

Implementation

- **ClaimRiskRecalcScheduler** that executes **ClaimRiskRecalcBatch** nightly.

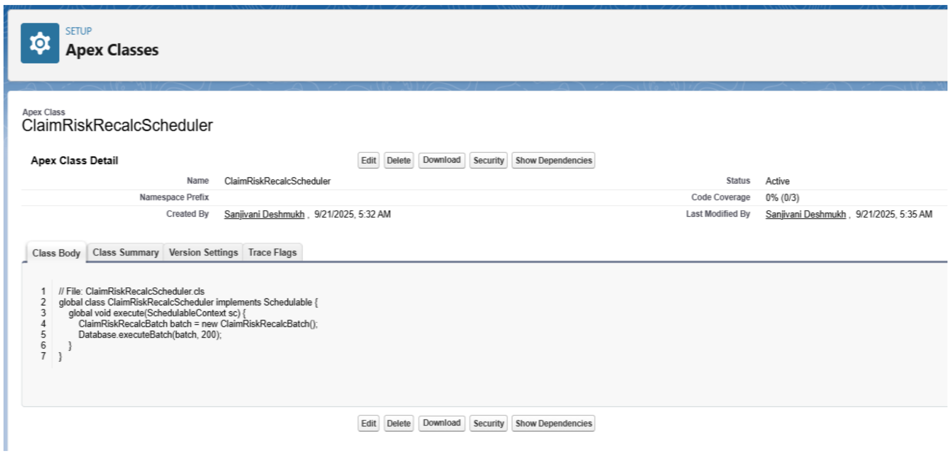
Why

- Regularly keep risk scores up-to-date and detect emerging fraud patterns.

How to schedule

- From Setup → Apex Classes → Schedule Apex, or use:

  System.schedule('Nightly Claim Risk', '0 0 2 * * ?', new ClaimRiskRecalcScheduler());



Apex Class
ClaimRiskRecalcScheduler

Apex Class Detail

Name	ClaimRiskRecalcScheduler	Status	Active
Namespace Prefix		Code Coverage	0% (0/3)
Created By	Sanjivani Deshmukh , 9/21/2025, 5:32 AM	Last Modified By	Sanjivani Deshmukh , 9/21/2025, 5:35 AM

```
// File: ClaimRiskRecalcScheduler.cls
global class ClaimRiskRecalcScheduler implements Schedulable {
    global void execute(SchedulableContext sc) {
        ClaimRiskRecalcBatch batch = new ClaimRiskRecalcBatch();
        Database.executeBatch(batch, 200);
    }
}
```

10. Future Methods

Purpose

Asynchronous execution for callouts that cannot be done synchronously in triggers.

Implementation

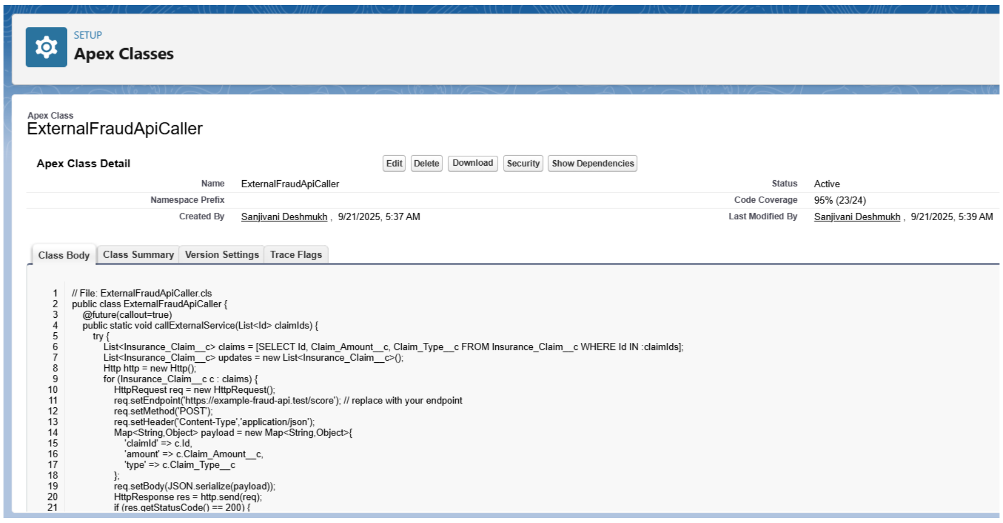
- **ExternalFraudApiCaller.callExternalService(List<Id> claimIds)** with **@future(callout=true)** as a wrapper to call an external fraud API and update scores.

Why

- Trigger context cannot perform callouts; **@future or Queueable** allows callouts asynchronously.

Important

- In tests, use **Test.setMock(HttpCalloutMock.class, ...)** to simulate external responses.



Apex Class
ExternalFraudApiCaller

Apex Class Detail

Name	ExternalFraudApiCaller	Status	Active
Namespace Prefix		Code Coverage	95% (23/24)
Created By	Sanjivani Deshmukh , 9/21/2025, 5:37 AM	Last Modified By	Sanjivani Deshmukh , 9/21/2025, 5:39 AM

```
// File: ExternalFraudApiCaller.cls
public class ExternalFraudApiCaller {
    @future(callout=true)
    public static void callExternalService(List<Id> claimIds) {
        try {
            List<Insurance_Claim__c> claims = [SELECT Id, Claim_Amount__c, Claim_Type__c FROM Insurance_Claim__c WHERE Id IN :claimIds];
            List<Insurance_Claim__c> updates = new List<Insurance_Claim__c>();
            Http http = new Http();
            for (Insurance_Claim__c c : claims) {
                HttpRequest req = new HttpRequest();
                req.setEndpoint('https://example-fraud-api.test/score'); // replace with your endpoint
                req.setMethod('POST');
                req.setHeader('Content-Type','application/json');
                Map<String,Object> payload = new Map<String,Object>{
                    'claimId' => c.Id,
                    'amount' => c.Claim_Amount__c,
                    'type' => c.Claim_Type__c
                };
                req.setBody(JSON.serialize(payload));
                HttpResponse res = http.send(req);
                if (res.getStatusCode() == 200) {
```

11. Exception Handling

Purpose
Catch and handle runtime errors without breaking user transactions unnecessarily.

Implementation

- try-catch blocks in callout and queueable classes; in triggers added **addError()** to block invalid saves when appropriate.

Why

- Provide graceful fallback (e.g., set default score if API fails) and log errors for troubleshooting.

12. Test Classes

Purpose
Unit tests validate logic, ensure >= 75% coverage and allow safe deployment.

Implementation

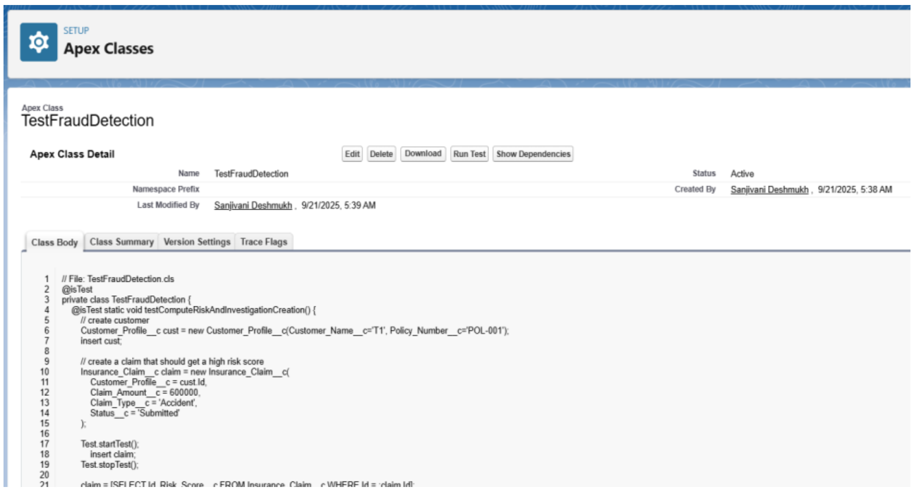
- **TestFraudDetection** test class with multiple test methods:
    - Insert customer & claim → assert **Risk_Score__c** computed and investigation created.
    - Insert **Fraud_Investigation__c** with **high Fraud_Score__c** → assert related claim status becomes In Review.
    - Mock HTTP callout and test **@future** call to external fraud API.

Why

- Required for deployment. Ensures code works across scenarios and handles async jobs.

Implementation notes

- Use **@isTest** annotation.
- Use **Test.startTest() / Test.stopTest()** around async operations to force execution within tests.
- Use **Test.setMock(HttpCalloutMock.class, new MockHttpResponseGenerator())** for callouts.



Apex Class
TestFraudDetection

Apex Class Detail

Name	TestFraudDetection	Status	Active
Namespace Prefix		Created By	Sanjivani Deshmukh , 9/21/2025, 5:38 AM
Last Modified By	Sanjivani Deshmukh , 9/21/2025, 5:39 AM		

```
// File: TestFraudDetection.cls
@isTest
private class TestFraudDetection {
  @isTest static void testComputeRiskAndInvestigationCreation() {
    // create customer
    Customer_Profile__c cust = new Customer_Profile__c(Customer_Name__c='T1', Policy_Number__c='POL-001');
    insert cust;

    // create a claim that should get a high risk score
    Insurance_Claim__c claim = new Insurance_Claim__c(
      Customer_Profile__c = cust.Id,
      Claim_Amount__c = 600000,
      Claim_Type__c = 'Accident',
      Status__c = 'Submitted'
    );

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

    claim = [SELECT Id, Risk_Score__c FROM Insurance_Claim__c WHERE Id = :claim.Id];
```

13. Asynchronous Processing (summary)

Purpose

Use Queueable, Batch, Scheduled Apex and Future to move heavy/remote/blocking work out of immediate transaction.

Used

- Queueable for notifications, Batch for nightly recalculation, Scheduled for running batch, Future for external API callouts.

Why

- Protects user experience, respects governor limits, supports scale.