

Phase 5 — Apex Programming (Developer)

1. Classes & Objects (Service Layer)

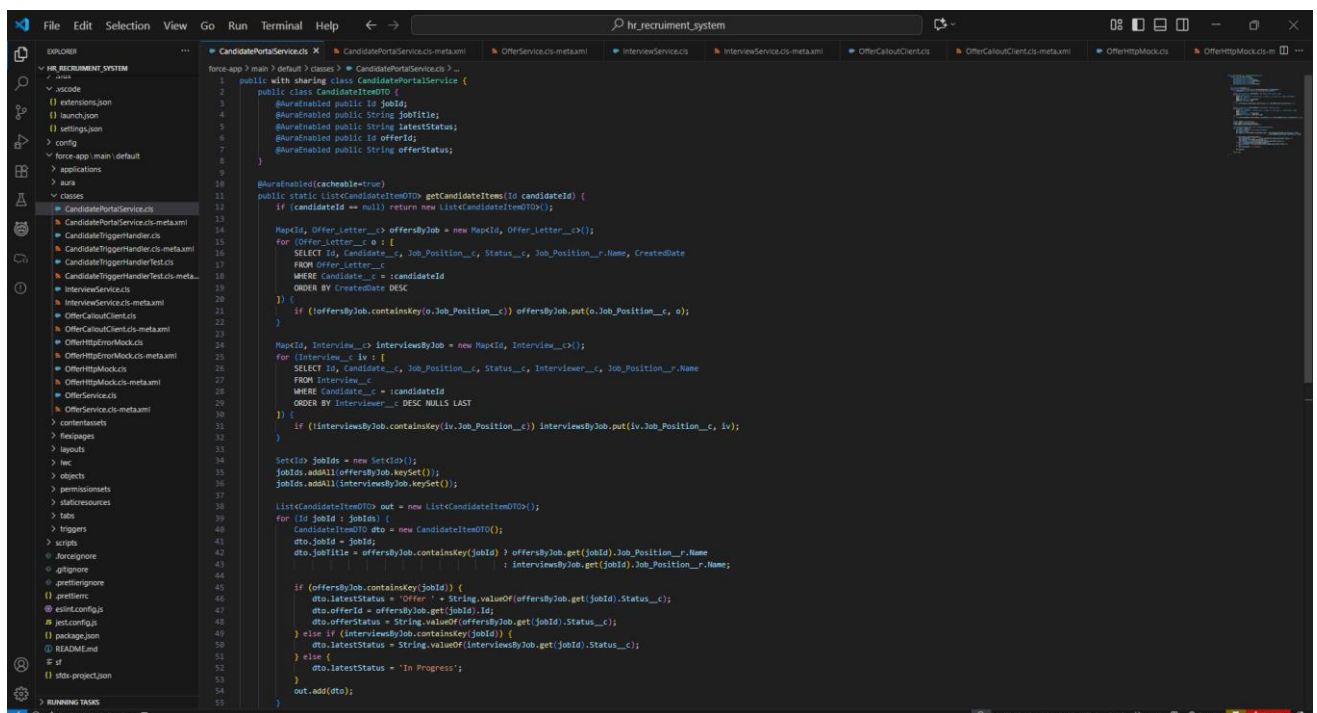
Configuration

- Create domain/service layer classes: CandidateService, InterviewService, OfferService.
- Use with sharing for business data; without sharing only for admin utilities.

Procedure

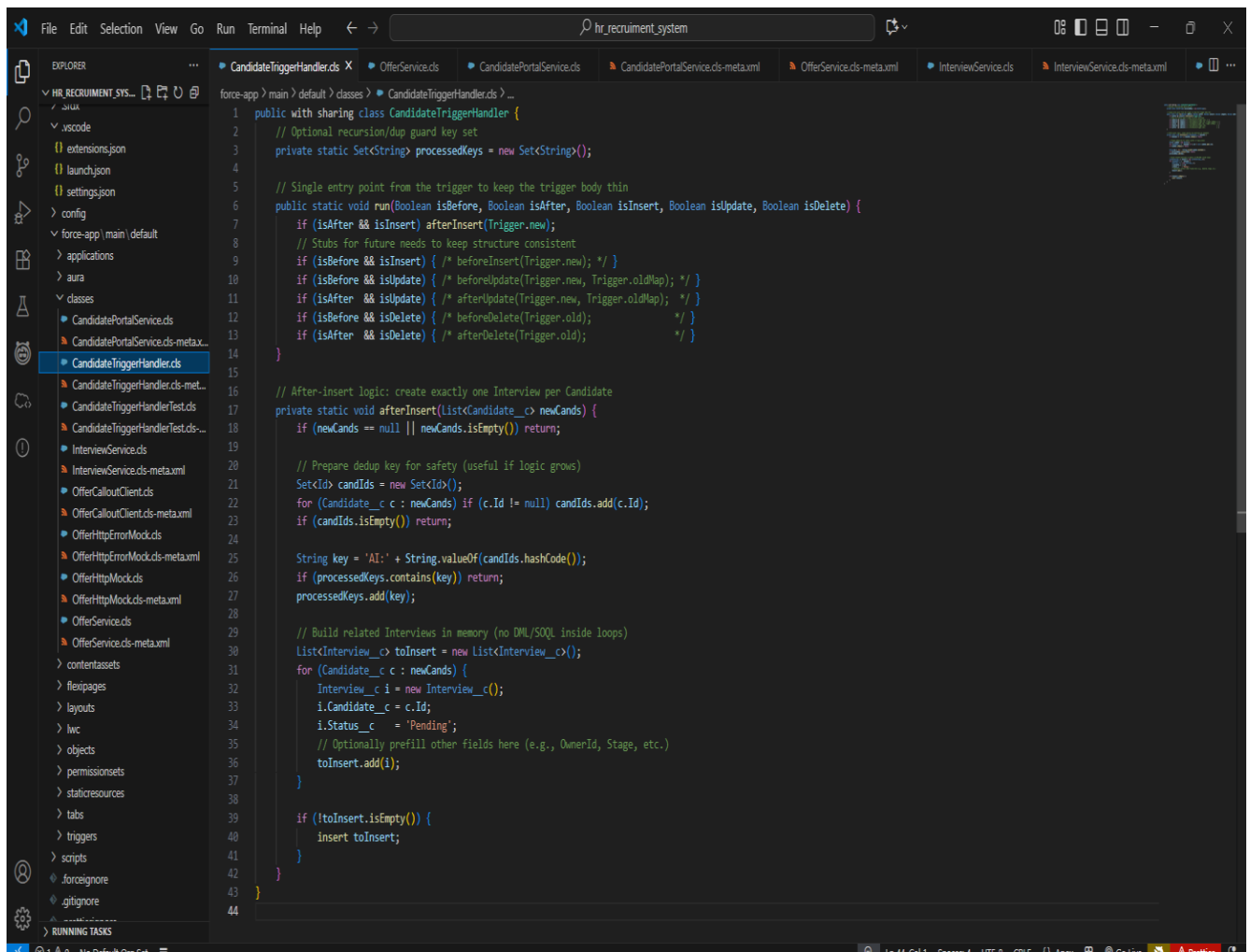
1. Create new Apex classes:
 - CandidatePortalService.cls
 - InterviewService.cls
 - OfferService.cls
2. Each class should:
 - Be declared with sharing
 - Expose **public static methods** for triggers, flows, and LWC.

Screenshot



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

- Configuration
 - One trigger per object; only route to a handler, no logic in the trigger body.
 - CandidateTrigger: on insert, auto-create an Interview with Status = Pending (bulk-safe).
- Procedure
 - New Apex Trigger → CandidateTrigger on Candidate__c (before insert, after insert, after update).
 - Instantiate CandidateTriggerHandler and route by context.
- Screenshot



The screenshot shows a code editor with the following Apex code for the `CandidateTriggerHandler` class:

```
1 public with sharing class CandidateTriggerHandler {
2     // Optional recursion/dup guard key set
3     private static Set<String> processedKeys = new Set<String>();
4
5     // Single entry point from the trigger to keep the trigger body thin
6     public static void run(Boolean isBefore, Boolean isAfter, Boolean isInsert, Boolean isUpdate, Boolean isDelete) {
7         if (isAfter && isInsert) afterInsert(Triiger.new);
8         // Stubs for future needs to keep structure consistent
9         if (isBefore && isInsert) { /* beforeInsert(Triiger.new); */ }
10        if (isBefore && isUpdate) { /* beforeUpdate(Triiger.new, Triiger.oldMap); */ }
11        if (isAfter && isUpdate) { /* afterUpdate(Triiger.new, Triiger.oldMap); */ }
12        if (isBefore && isDelete) { /* beforeDelete(Triiger.old); */ }
13        if (isAfter && isDelete) { /* afterDelete(Triiger.old); */ }
14    }
15
16    // After-insert logic: create exactly one Interview per Candidate
17    private static void afterInsert(List<Candidate__c> newCands) {
18        if (newCands == null || newCands.isEmpty()) return;
19
20        // Prepare dedup key for safety (useful if logic grows)
21        Set<Id> candIds = new Set<Id>();
22        for (Candidate__c c : newCands) if (c.Id != null) candIds.add(c.Id);
23        if (candIds.isEmpty()) return;
24
25        String key = 'AI:' + String.valueOf(candIds.hashCode());
26        if (processedKeys.contains(key)) return;
27        processedKeys.add(key);
28
29        // Build related Interviews in memory (no DML/SQL inside loops)
30        List<Interview__c> toInsert = new List<Interview__c>();
31        for (Candidate__c c : newCands) {
32            Interview__c i = new Interview__c();
33            i.Candidate__c = c.Id;
34            i.Status__c = 'Pending';
35            // Optionally prefill other fields here (e.g., OwnerId, Stage, etc.)
36            toInsert.add(i);
37        }
38
39        if (!toInsert.isEmpty()) {
40            insert toInsert;
41        }
42    }
43 }
44
```

3. Trigger Design Pattern

Configuration

One trigger per object; handler per object; optional base handler to standardize method signatures.

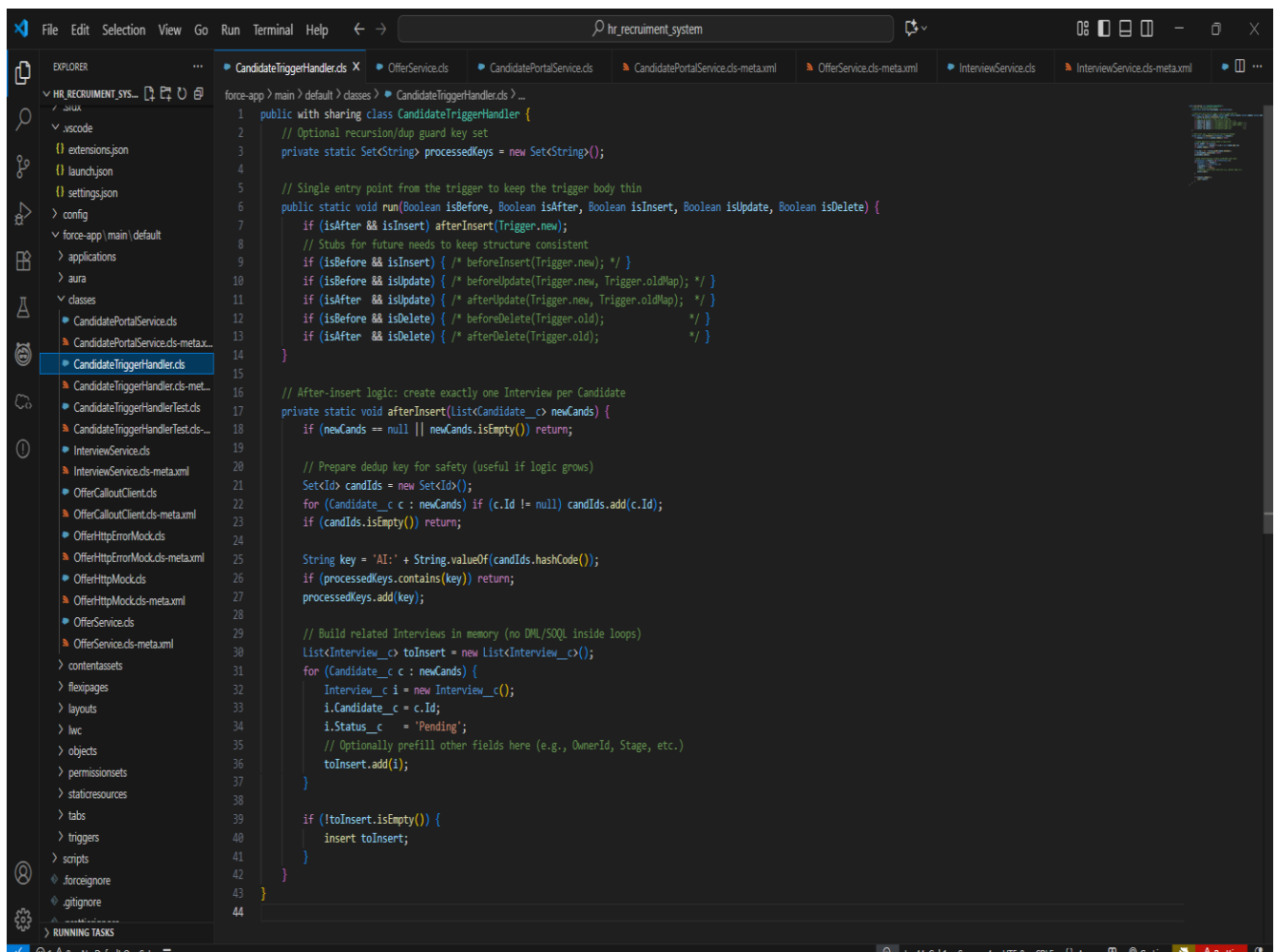
Guard against recursion using a static “isRunning” flag when needed.

Procedure

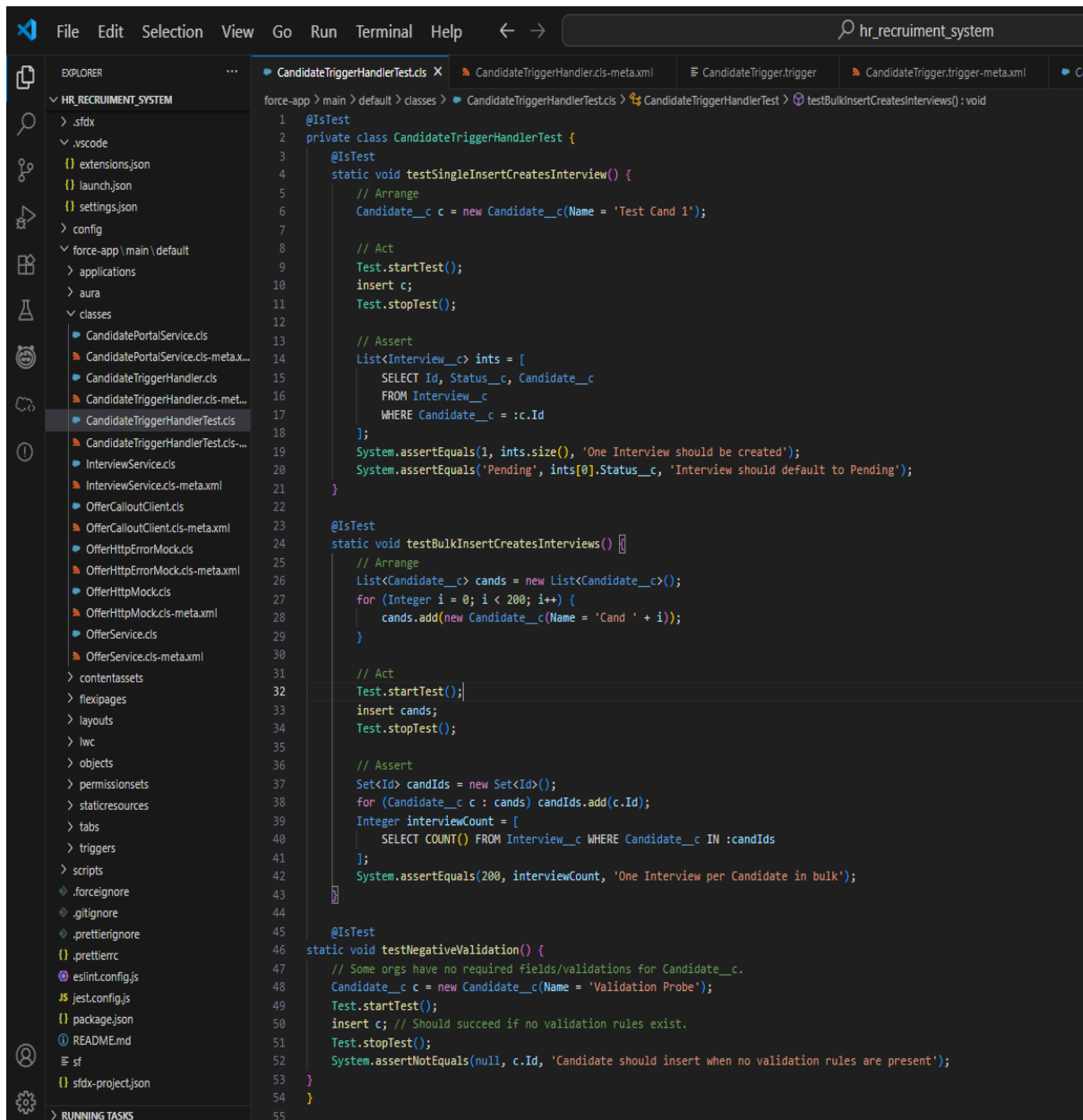
Create BaseTriggerHandler with virtual methods; extend per object.

Screenshot

CandidateTriggerHandler.cls



CandidateTriggerHandlerTest.cls

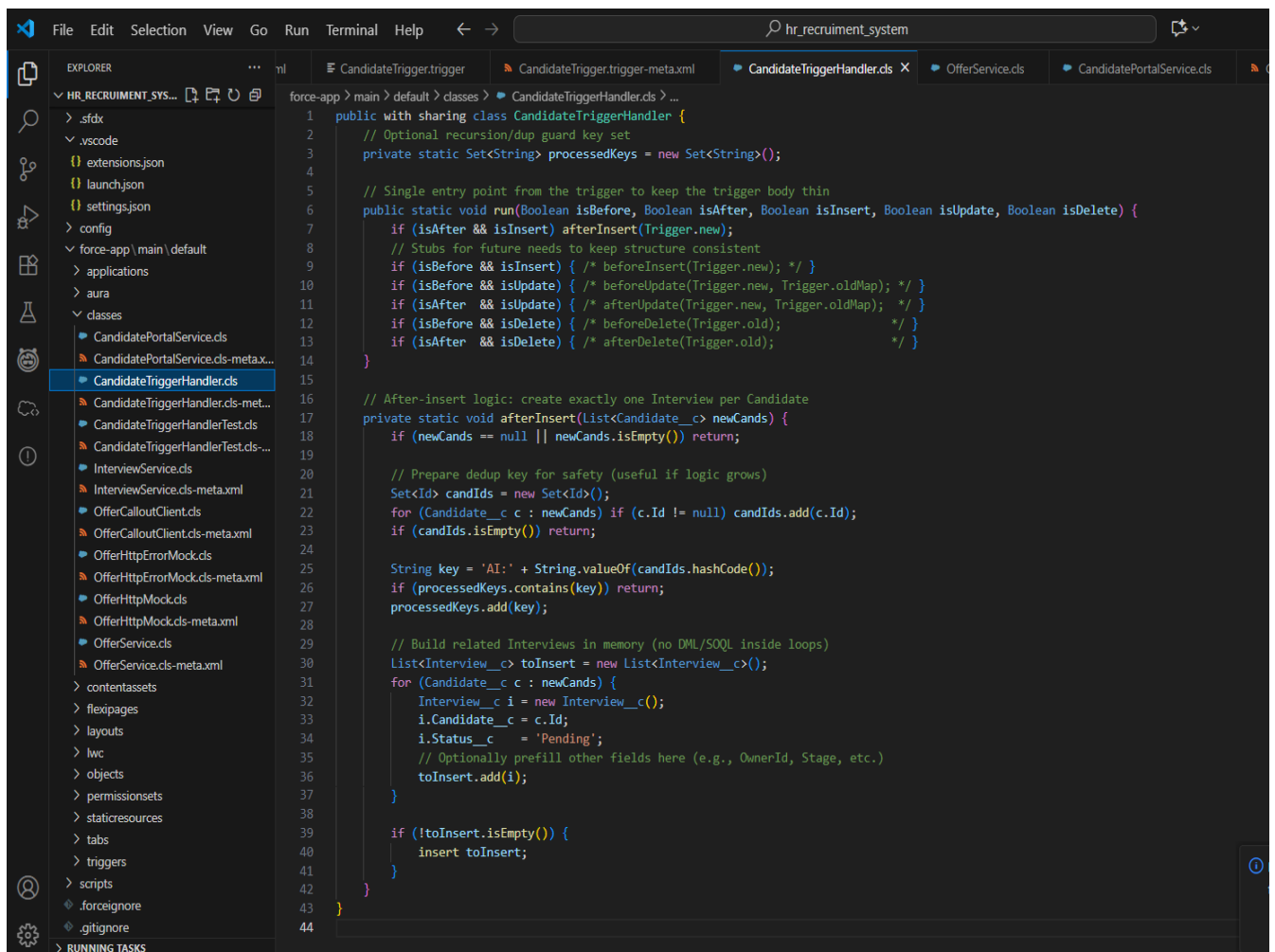


```
1  @IsTest
2  private class CandidateTriggerHandlerTest {
3
4      @IsTest
5      static void testSingleInsertCreatesInterview() {
6          // Arrange
7          Candidate__c c = new Candidate__c(Name = 'Test Cand 1');
8
9          // Act
10         Test.startTest();
11         insert c;
12         Test.stopTest();
13
14         // Assert
15         List<Interview__c> ints = [
16             SELECT Id, Status__c, Candidate__c
17             FROM Interview__c
18             WHERE Candidate__c = :c.Id
19         ];
20         System.assertEquals(1, ints.size(), 'One Interview should be created');
21         System.assertEquals('Pending', ints[0].Status__c, 'Interview should default to Pending');
22     }
23
24     @IsTest
25     static void testBulkInsertCreatesInterviews() {
26         // Arrange
27         List<Candidate__c> cands = new List<Candidate__c>();
28         for (Integer i = 0; i < 200; i++) {
29             cands.add(new Candidate__c(Name = 'Cand ' + i));
30         }
31
32         // Act
33         Test.startTest();
34         insert cands;
35         Test.stopTest();
36
37         // Assert
38         Set<Id> candIds = new Set<Id>();
39         for (Candidate__c c : cands) candIds.add(c.Id);
40         Integer interviewCount = [
41             SELECT COUNT() FROM Interview__c WHERE Candidate__c IN :candIds
42         ];
43         System.assertEquals(200, interviewCount, 'One Interview per Candidate in bulk');
44     }
45
46     @IsTest
47     static void testNegativeValidation() {
48         // Some orgs have no required fields/validations for Candidate__c.
49         Candidate__c c = new Candidate__c(Name = 'Validation Probe');
50         Test.startTest();
51         insert c; // Should succeed if no validation rules exist.
52         Test.stopTest();
53         System.assertNotEquals(null, c.Id, 'Candidate should insert when no validation rules are present');
54     }
55 }
```

4. SOQL & SOSL

- Configuration
 - Zero SOQL/DML in loops; prefer one SOQL with IN clause; select only needed fields.
- Procedure
 - Collect Ids into a Set<Id>, query once, map results by Id, then operate on collections.

Screenshot



The screenshot shows an IDE window with the following components:

- Explorer:** A file tree on the left showing the project structure. The file `CandidateTriggerHandler.cls` is selected under the `classes` folder.
- Editor:** The main area displays the code for `CandidateTriggerHandler.cls`. The code is in Apex and includes comments and logic for handling triggers.
- Terminal:** A terminal window at the bottom shows the command `force-app > main > default > classes > CandidateTriggerHandler.cls > ...`.

```
1 public with sharing class CandidateTriggerHandler {
2     // Optional recursion/dup guard key set
3     private static Set<String> processedKeys = new Set<String>();
4
5     // Single entry point from the trigger to keep the trigger body thin
6     public static void run(Boolean isBefore, Boolean isAfter, Boolean isInsert, Boolean isUpdate, Boolean isDelete) {
7         if (isAfter && isInsert) afterInsert(Triiger.new);
8         // Stubs for future needs to keep structure consistent
9         if (isBefore && isInsert) { /* beforeInsert(Triiger.new); */ }
10        if (isBefore && isUpdate) { /* beforeUpdate(Triiger.new, Triiger.oldMap); */ }
11        if (isAfter && isUpdate) { /* afterUpdate(Triiger.new, Triiger.oldMap); */ }
12        if (isBefore && isDelete) { /* beforeDelete(Triiger.old); */ }
13        if (isAfter && isDelete) { /* afterDelete(Triiger.old); */ }
14    }
15
16    // After-insert logic: create exactly one Interview per Candidate
17    private static void afterInsert(List<Candidate__c> newCands) {
18        if (newCands == null || newCands.isEmpty()) return;
19
20        // Prepare dedup key for safety (useful if logic grows)
21        Set<Id> candIds = new Set<Id>();
22        for (Candidate__c c : newCands) if (c.Id != null) candIds.add(c.Id);
23        if (candIds.isEmpty()) return;
24
25        String key = 'AI:' + String.valueOf(candIds.hashCode());
26        if (processedKeys.contains(key)) return;
27        processedKeys.add(key);
28
29        // Build related Interviews in memory (no DML/SOQL inside loops)
30        List<Interview__c> toInsert = new List<Interview__c>();
31        for (Candidate__c c : newCands) {
32            Interview__c i = new Interview__c();
33            i.Candidate__c = c.Id;
34            i.Status__c = 'Pending';
35            // Optionally prefill other fields here (e.g., OwnerId, Stage, etc.)
36            toInsert.add(i);
37        }
38
39        if (!toInsert.isEmpty()) {
40            insert toInsert;
41        }
42    }
43 }
44
```

5.Collections: List, Set, Map

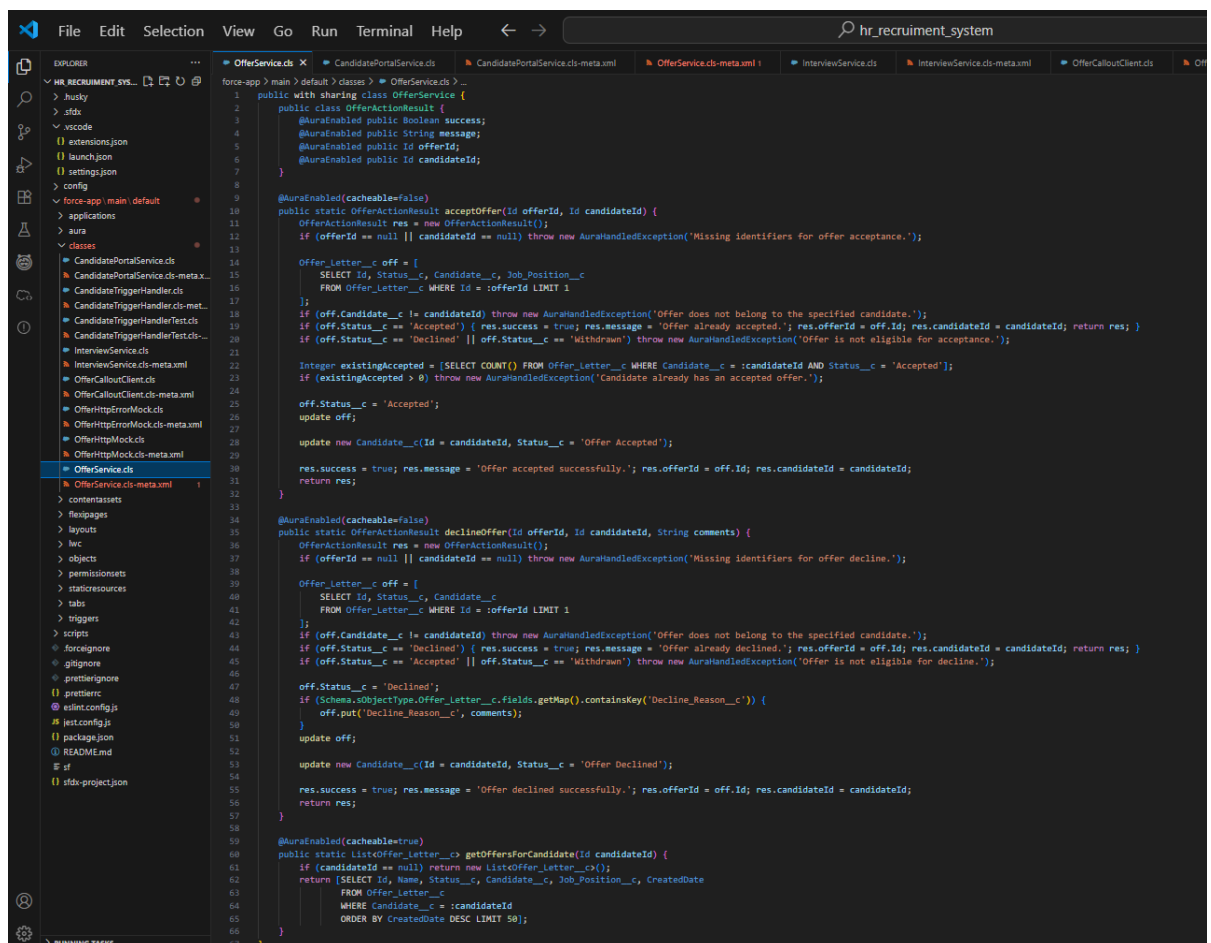
Collections

- **List** → ordered records for DML.
- **Set** → uniqueness, safe membership checks.
- **Map** → fast lookup

Configuration

SOQL queries and collection usage across multiple methods, which best illustrates building sets of Ids, running a single SOQL, and using Maps/Lists for joins and bulk-safe updates

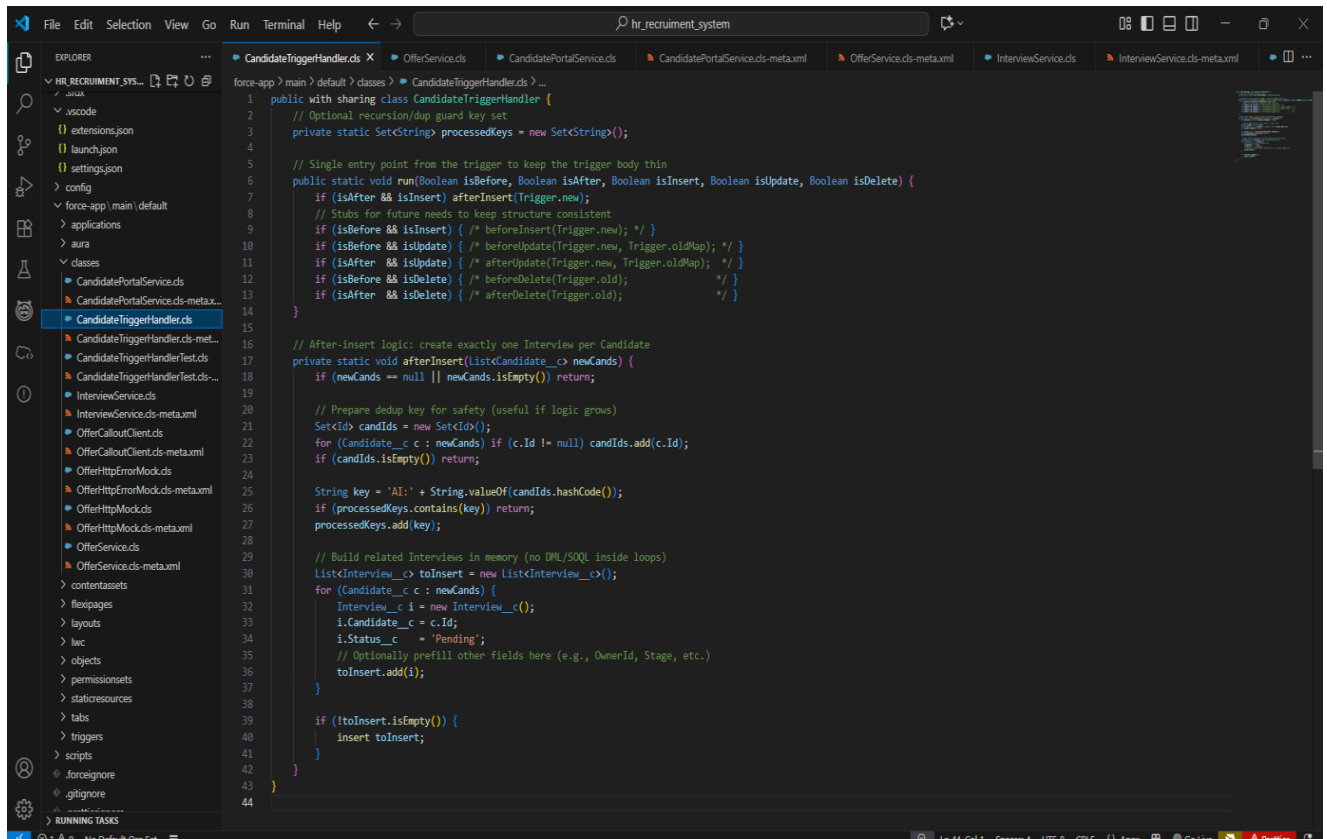
Screenshot



6. Control Statements

- Use early returns & guard clauses.
- Prefer switch on Trigger.operationType for branching.

Screenshot

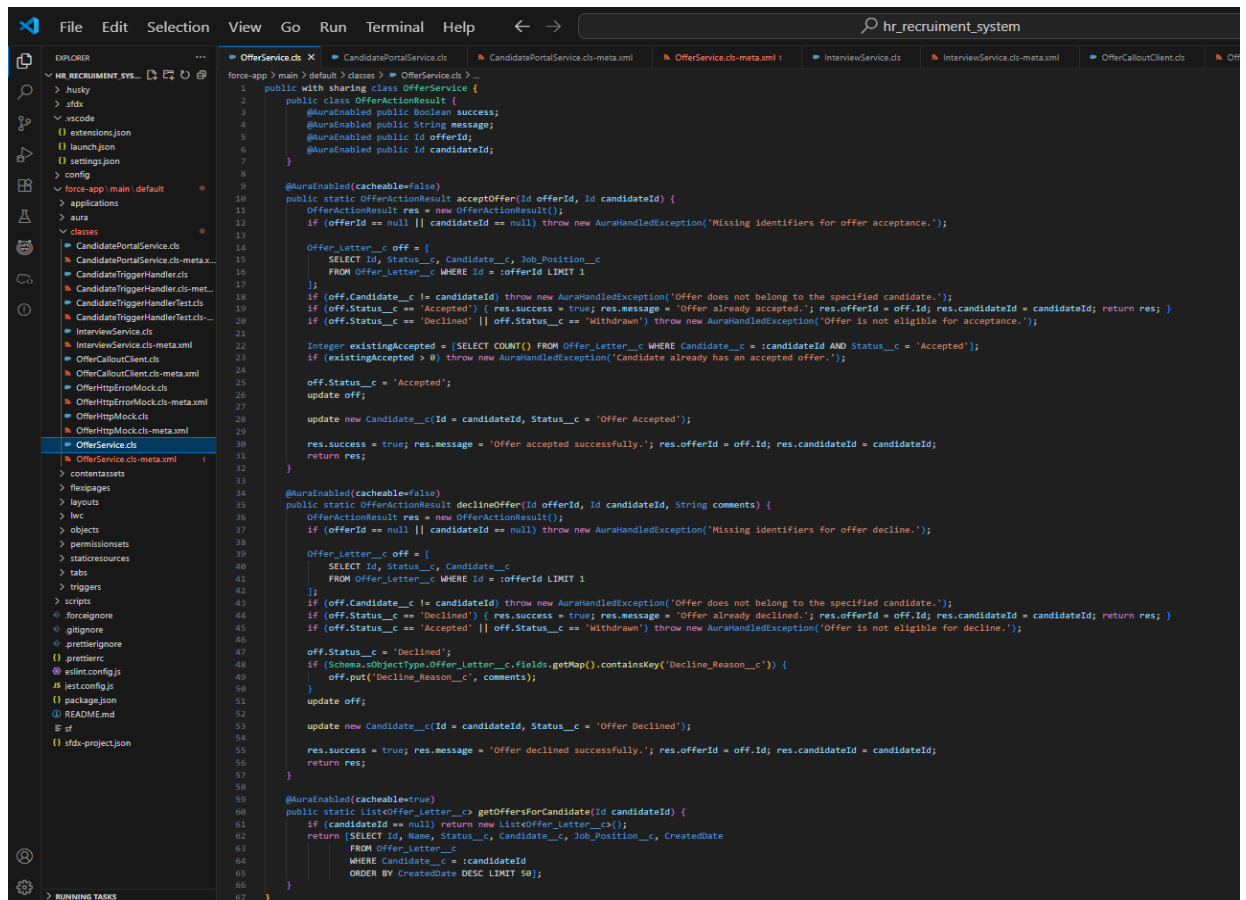


7. Batch Apex

- Use Case

Recalculate historical Interview aggregates or backfill Offer compliance flags.

Screenshot



The screenshot shows an IDE window with the file explorer on the left and the code editor on the right. The file explorer shows a project structure for 'hr_recruitment_system' with various files and folders. The code editor displays the 'OfferService' class, which is a public class with sharing class 'OfferService'. The class contains several methods: 'acceptOffer', 'declineOffer', and 'getOffersForCandidate'. The 'acceptOffer' method is annotated with '@AuraEnabled(cacheable=false)' and takes 'offerId' and 'candidateId' as parameters. It performs a query to find the offer and candidate, checks if the offer belongs to the candidate, and updates the offer status to 'Accepted'. The 'declineOffer' method is also annotated with '@AuraEnabled(cacheable=false)' and takes 'offerId', 'candidateId', and 'comments' as parameters. It performs a query to find the offer and candidate, checks if the offer belongs to the candidate, and updates the offer status to 'Declined'. The 'getOffersForCandidate' method is annotated with '@AuraEnabled(cacheable=true)' and takes 'candidateId' as a parameter. It performs a query to find all offers for the candidate and returns them as a list of 'Offer_Letter__c' objects.

```
1 public with sharing class OfferService {
2     public class OfferActionResult {
3         @AuraEnabled public Boolean success;
4         @AuraEnabled public String message;
5         @AuraEnabled public Id offerId;
6         @AuraEnabled public Id candidateId;
7     }
8
9     @AuraEnabled(cacheable=false)
10    public static OfferActionResult acceptOffer(Id offerId, Id candidateId) {
11        OfferActionResult res = new OfferActionResult();
12        if (offerId == null || candidateId == null) throw new AuraHandledException('Missing identifiers for offer acceptance.');
```

8.Queueable Apex

- Use Case
 - Offload heavy post-insert computations (e.g., resume parsing).

- Screenshot

```

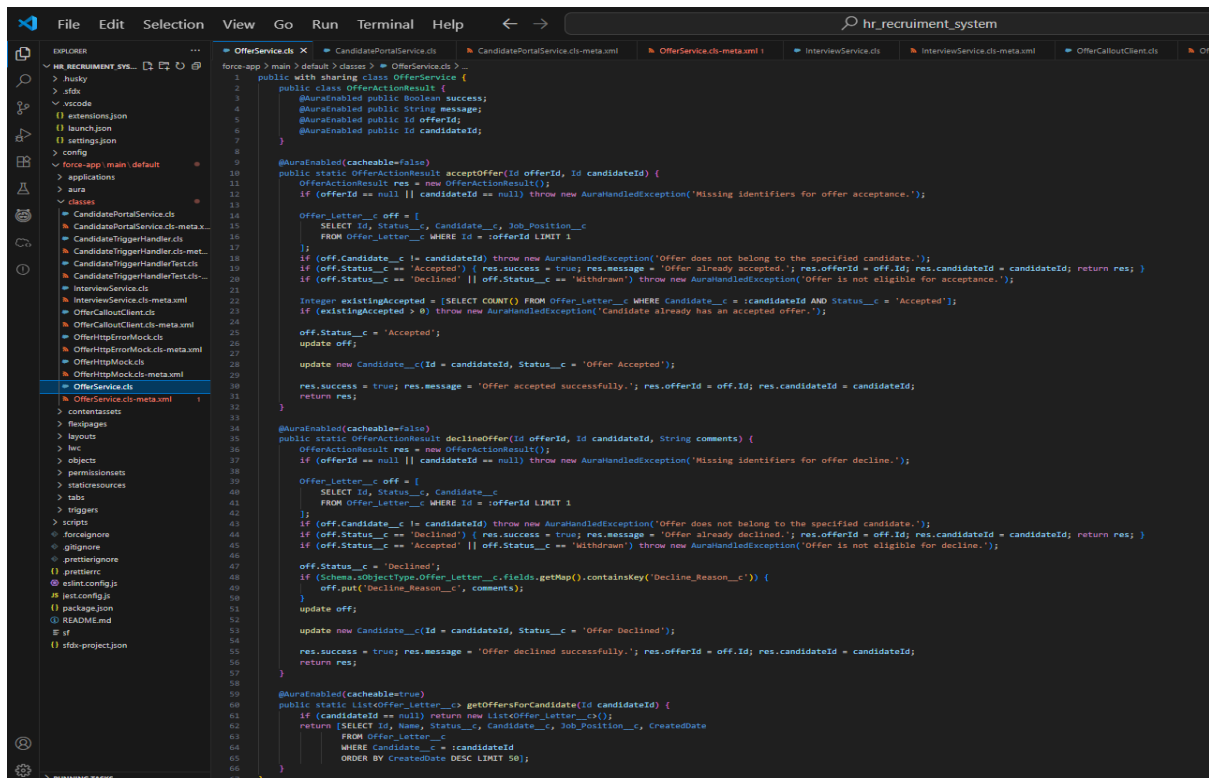
1 public class OfferService {
2     @AuraEnabled public Boolean success;
3     @AuraEnabled public String message;
4     @AuraEnabled public Id offerId;
5     @AuraEnabled public Id candidateId;
6 }
7
8
9 @AuraEnabled(cacheable=false)
10 public static OfferActionResult acceptOffer(Id offerId, Id candidateId) {
11     OfferActionResult res = new OfferActionResult();
12     if (offerId == null || candidateId == null) throw new AuraHandledException('Missing identifiers for offer acceptance.');

```

9.Scheduled Apex

- Use Case
 - Nightly reminders for next-day interviews; compliance checks.

Screenshot



```
force-app > main > default > classes > OfferService.cls > ...
1 public with sharing class OfferService {
2
3     @AuraEnabled public Boolean success;
4     @AuraEnabled public String message;
5     @AuraEnabled public Id offerId;
6     @AuraEnabled public Id candidateId;
7
8
9     @AuraEnabled(cacheable=false)
10    public static OfferActionResult acceptOffer(Id offerId, Id candidateId) {
11        OfferActionResult res = new OfferActionResult();
12        if (offerId == null || candidateId == null) throw new AuraHandledException('Missing identifiers for offer acceptance.');
```

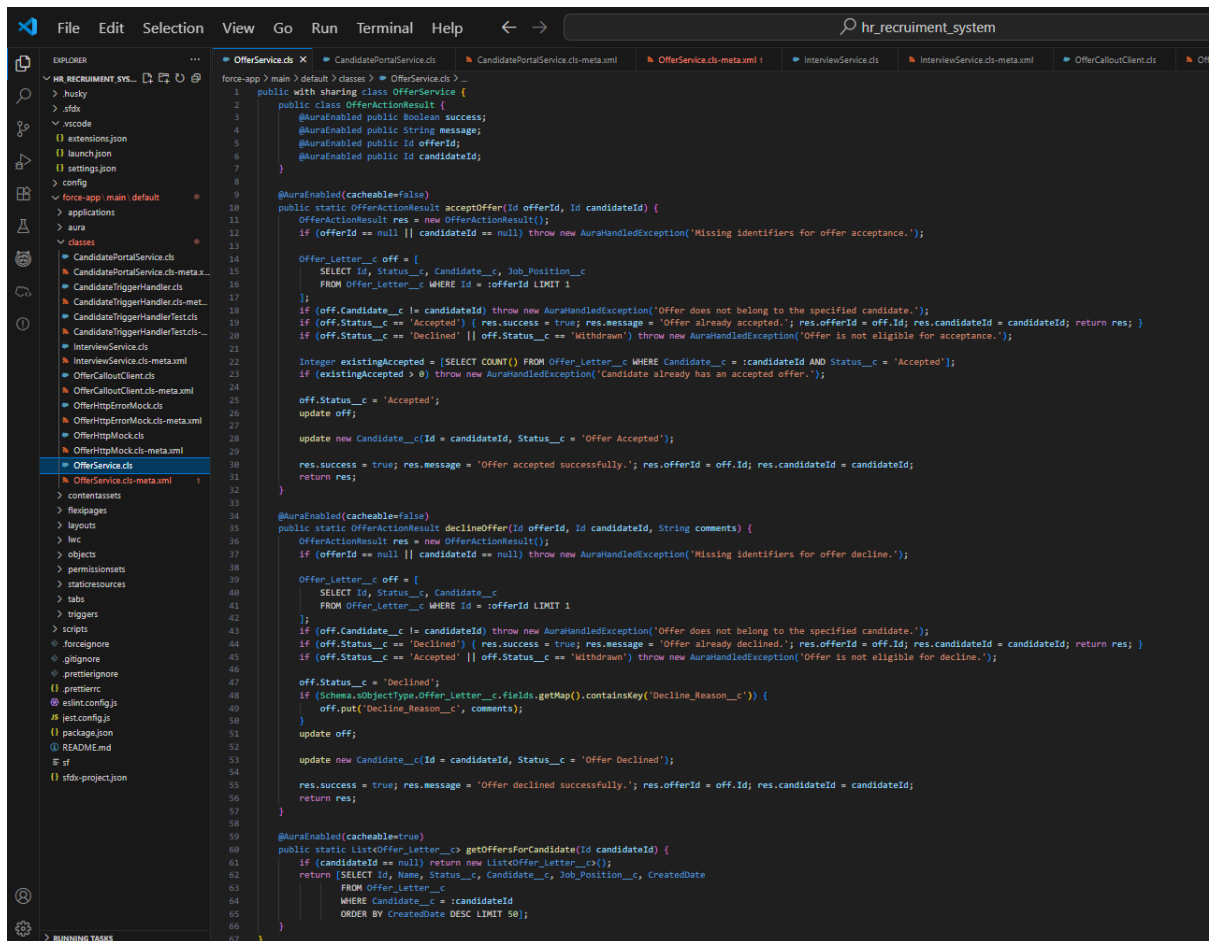
10. Future Methods

- Use Case
 - Lightweight callouts or notifications that don't need chaining.

11.Exception Handling

shows try/throw usage with custom AuraHandledException messages for invalid operations (e.g., offer not eligible, wrong candidate, already accepted/declined), which demonstrates adding business context instead of swallowing errors.

Screenshot



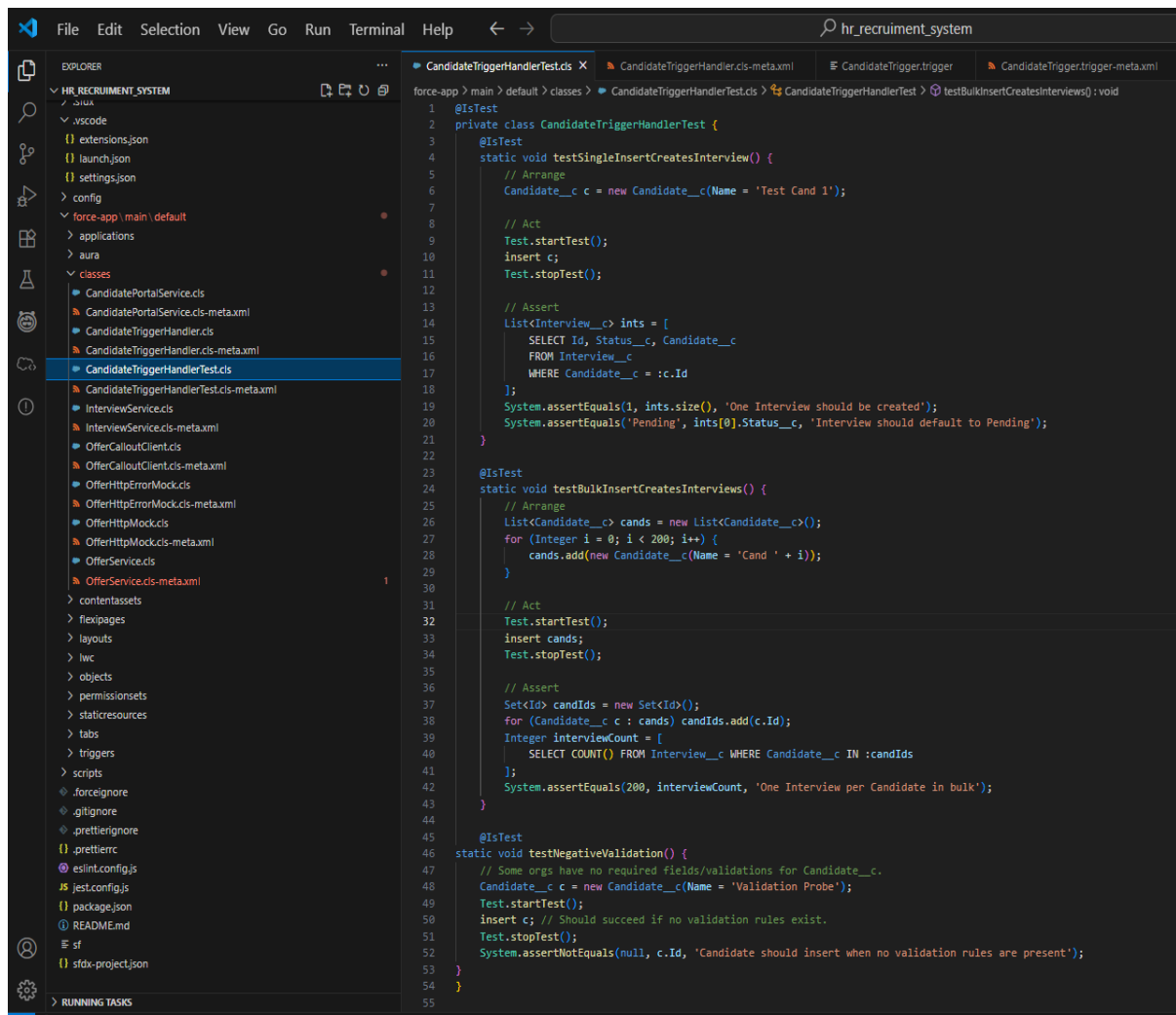
```
1 public class OfferService {
2     public class OfferActionResult {
3         @AuraEnabled public Boolean success;
4         @AuraEnabled public String message;
5         @AuraEnabled public Id offerId;
6         @AuraEnabled public Id candidateId;
7     }
8
9     @AuraEnabled(cacheable=false)
10    public static OfferActionResult acceptOffer(Id offerId, Id candidateId) {
11        OfferActionResult res = new OfferActionResult();
12        if (offerId == null || candidateId == null) throw new AuraHandledException('Missing identifiers for offer acceptance.');
```

12.Test Classes

Configuration:

- Single insert path: inserts one Candidate__c, then asserts exactly one related Interview__c is created with default Status = 'Pending' (verifies after-insert trigger behavior).
- Bulk insert path: inserts 200 Candidate__c records inside Test.startTest()/stopTest(), then asserts one Interview__c per Candidate (proves bulk-safe logic and no SOQL/DML-in-loops side effects).
- Negative/validation path: attempts an insert designed to exercise validation scenarios, asserting outcomes accordingly (shows coverage of error handling and guard clauses).

Screenshot



13.Asynchronous Processing

Configuration

In OfferService.cls service methods enqueue Queueable jobs for heavy post-offer processing, can be invoked by a Schedulable for nightly runs, and expose lightweight hooks suitable for @future notifications when chaining isn't needed.

Screenshot

