# Phase 5 – Apex Programming (Developer) Purpose

Ensure data quality and enforce business rules using automation tools such as triggers, validation logic, and declarative automation. The key goal of this phase is to make sure that duplicate or invalid records cannot be created in the system, thereby improving data integrity and user trust.

## Objectives

• Prevent creation of duplicate task records in the custom object Task\_\_c.

• Ensure the system provides real-time feedback to the user when attempting to save invalid data.

• Implement the solution using Apex Trigger and cover it with a Test Class for deployment readiness.

## 1. Classes & Objects

What: Apex classes hold reusable logic (handlers, services). Use classes to encapsulate business logic, enable testability, and keep triggers thin. Create classes for services such as Task creation, AI processing, notification delivery, and utilities.

Steps:

a) In Setup → Apex Classes → New, create a class file. Prefer small, focused classes.

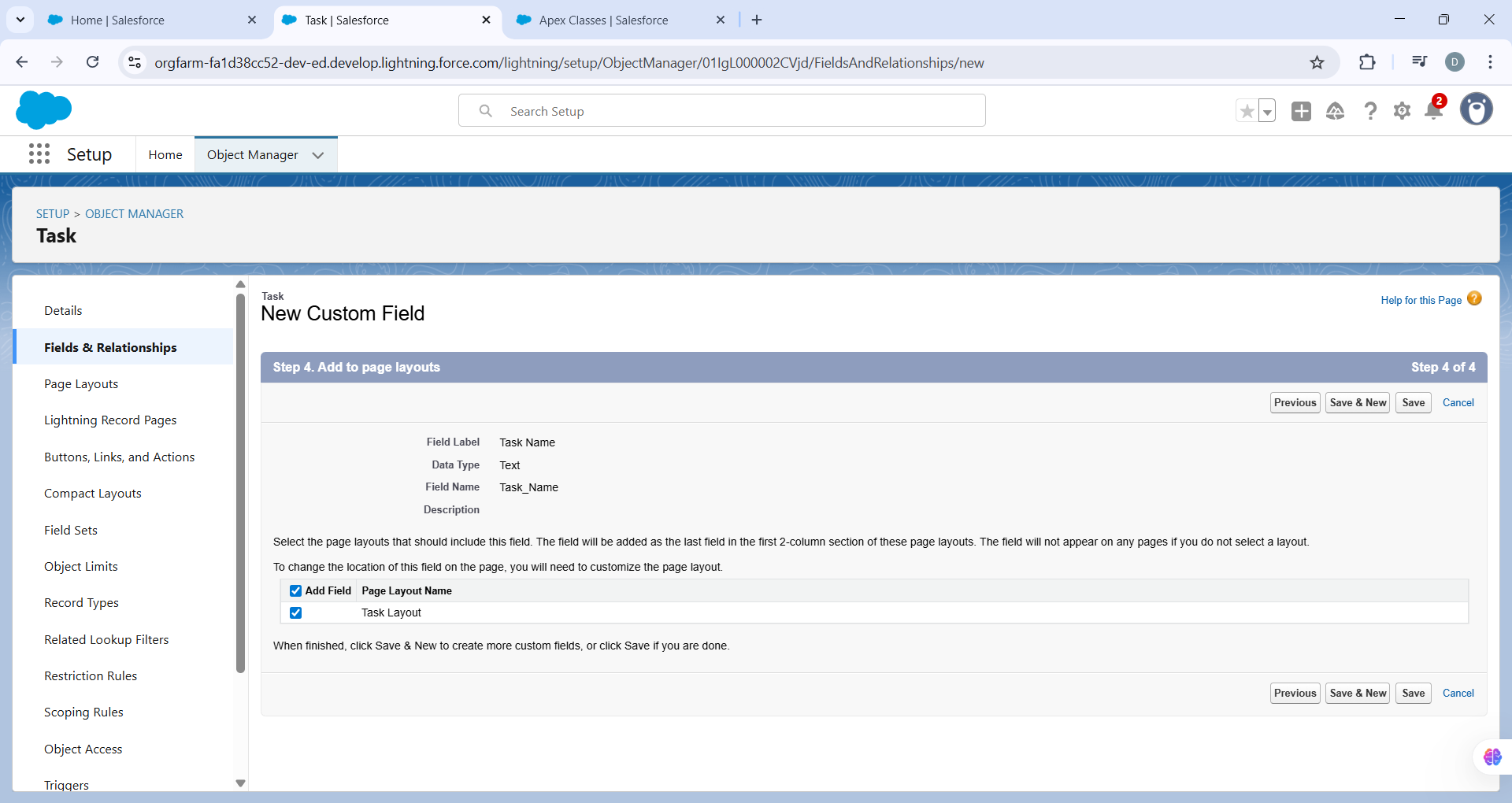
b) Keep methods public or global only when needed. Use 'with sharing' or 'without sharing' intentionally.

Best practice: Put DML and SOQL in service classes (not in triggers) and write unit tests for each class.

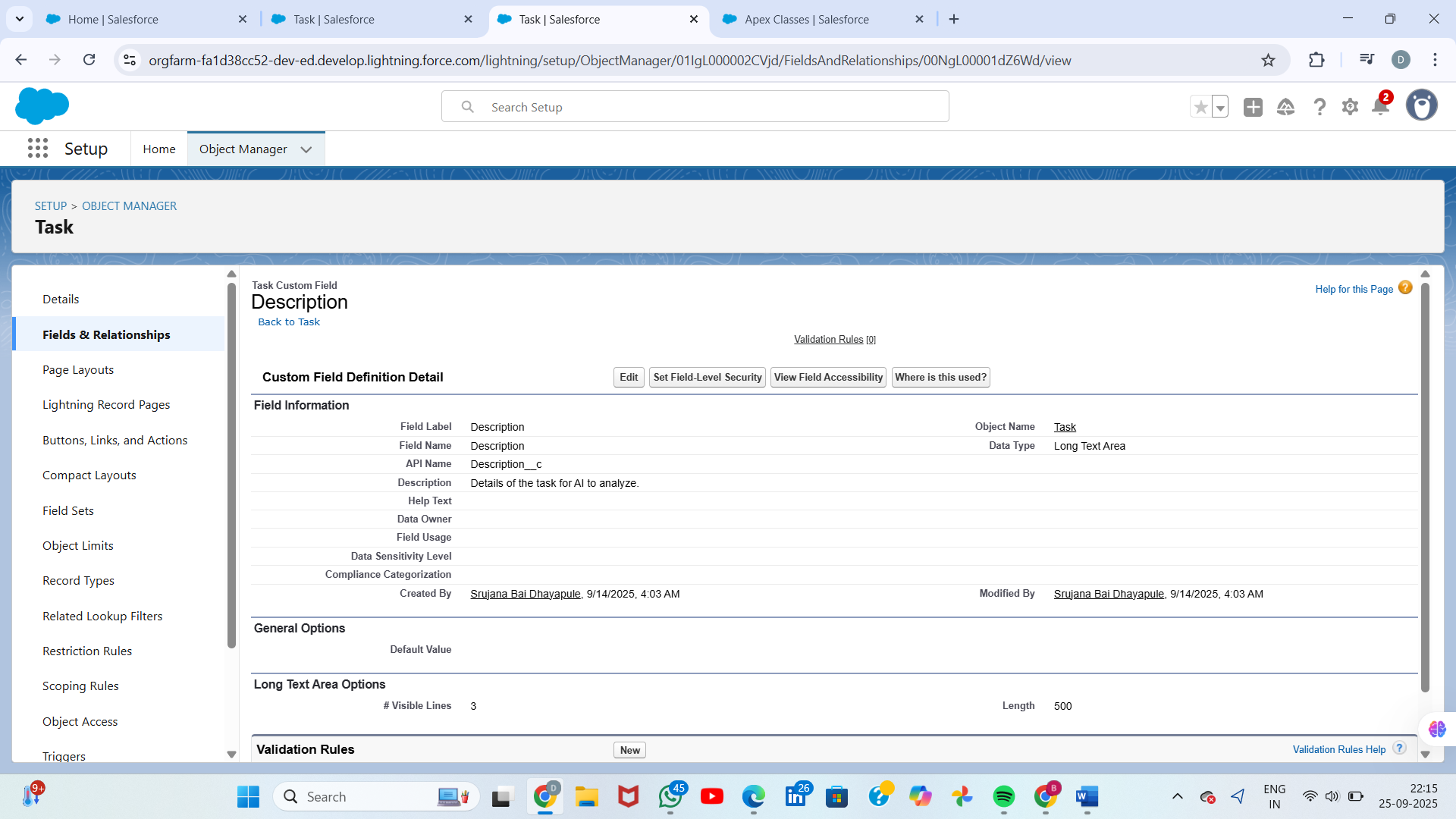
Example: TaskCreatorService (builds Task\_\_c records)

public with sharing class TaskCreatorService {  
 public static Task\_\_c buildReviewTask(Id aiSuggestionId, Id feedbackId, Decimal confidence, Id assignedUserId) {  
 Task\_\_c t = new Task\_\_c();  
 t.Task\_Name\_\_c = 'Review Approved Feedback';  
 t.Description\_\_c = 'Feedback ' + (feedbackId==null? '' : feedbackId) + ' approved. Suggestion: ' + String.valueOf(aiSuggestionId) + '. Confidence: ' + String.valueOf(confidence) + '%';  
 t.Status\_\_c = 'New';  
 t.Assigned\_User\_\_c = assignedUserId;  
 t.Related\_AI\_Suggestion\_\_c = aiSuggestionId;  
 return t;  
 }  
  
 public static void insertTasks(List<Task\_\_c> tasks) {  
 if (tasks == null || tasks.isEmpty()) return;  
 try {  
 insert tasks;  
 } catch (DmlException e) {  
 // handle or log error  
 System.debug('Error inserting tasks: ' + e.getMessage());  
 throw e; // rethrow or implement custom handling  
 }  
 }  
}  
**Task\_Name\_\_c:**

* **Data Type:** This should be a **Text** field.
* **Field Label:** Task Name
* **Field Name:** Task\_Name (the system will automatically add \_\_c)

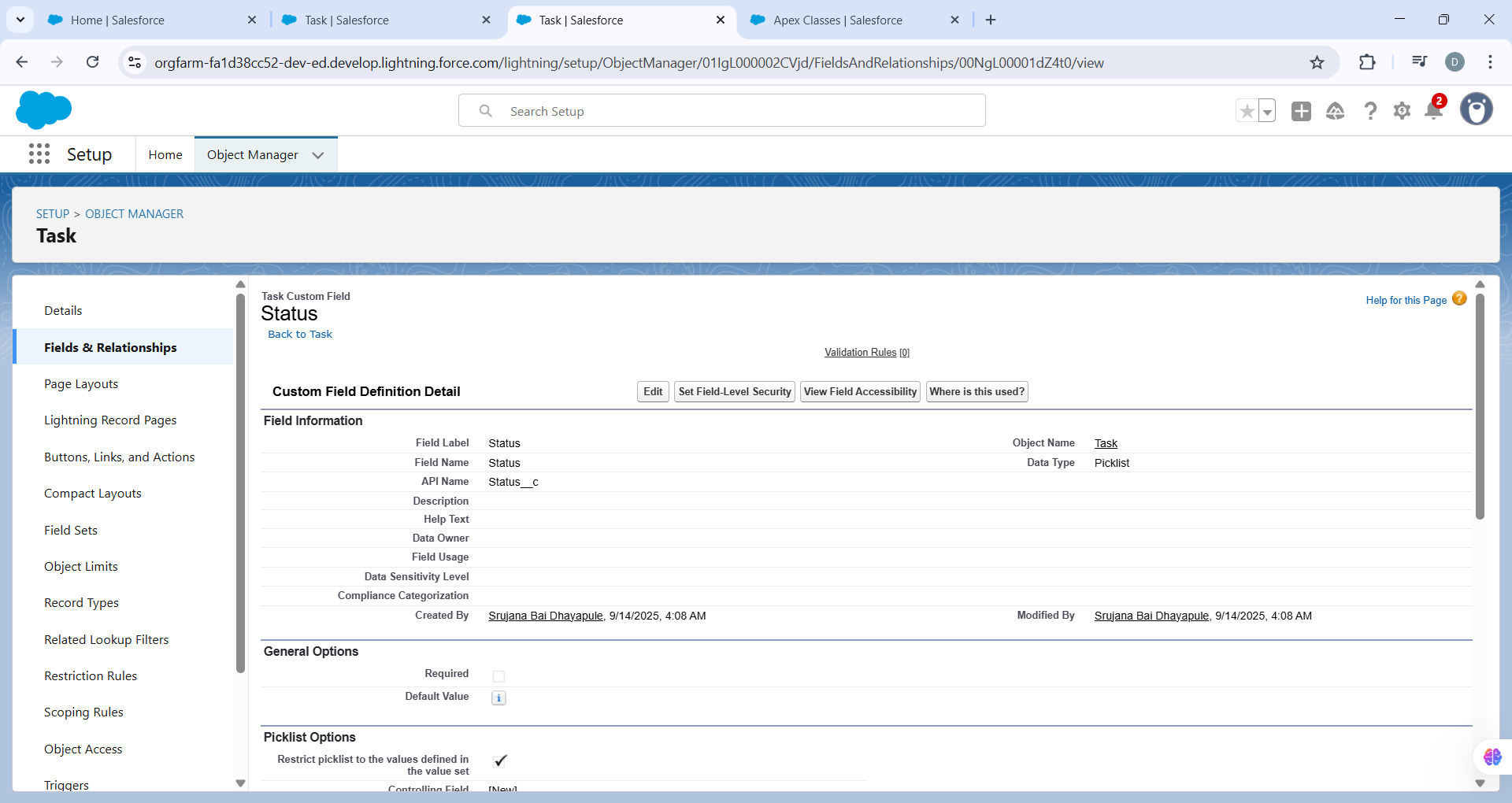
  
**Description\_\_c:**

* **Data Type:** This should be a **Text Area (Long)** field to hold more text.
* **Field Label:** Description
* **Field Name:** Description



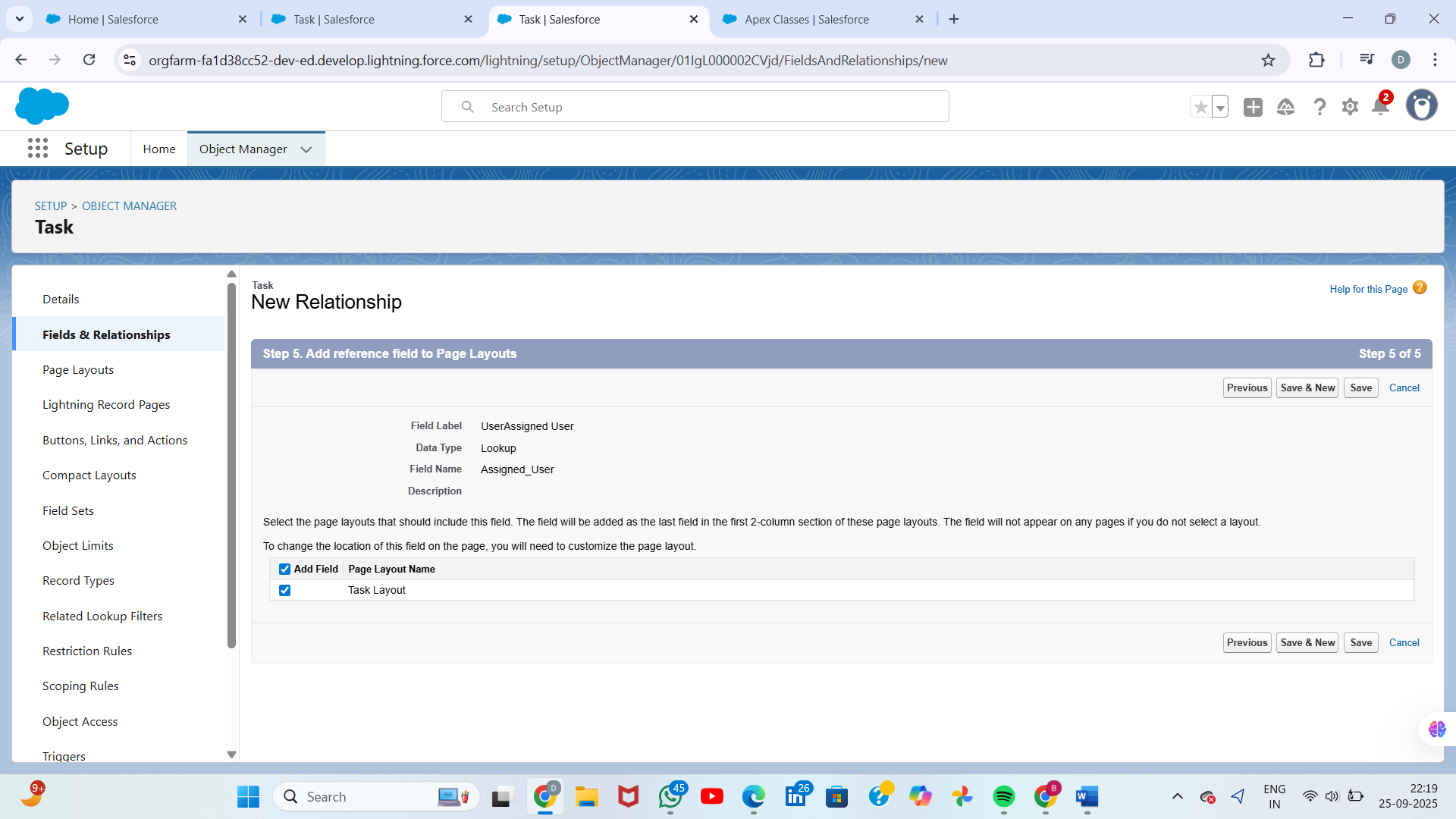
**Status\_\_c:**

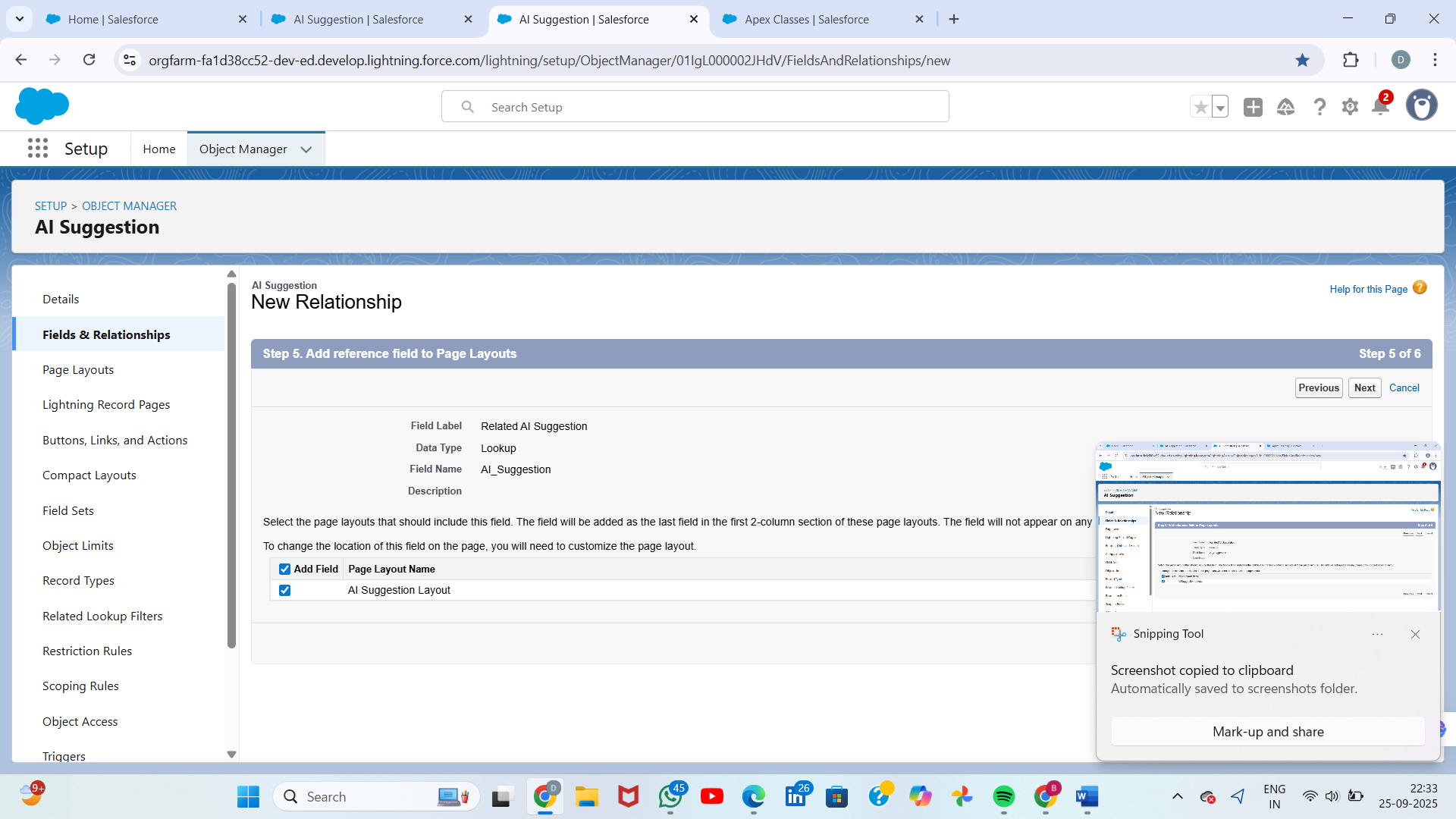
* **Data Type:** This is likely a **Picklist** field with values like 'New', 'In Progress', 'Completed', etc.
* **Field Label:** Status
* **Field Name:** Status



**Assigned\_User\_\_c:**

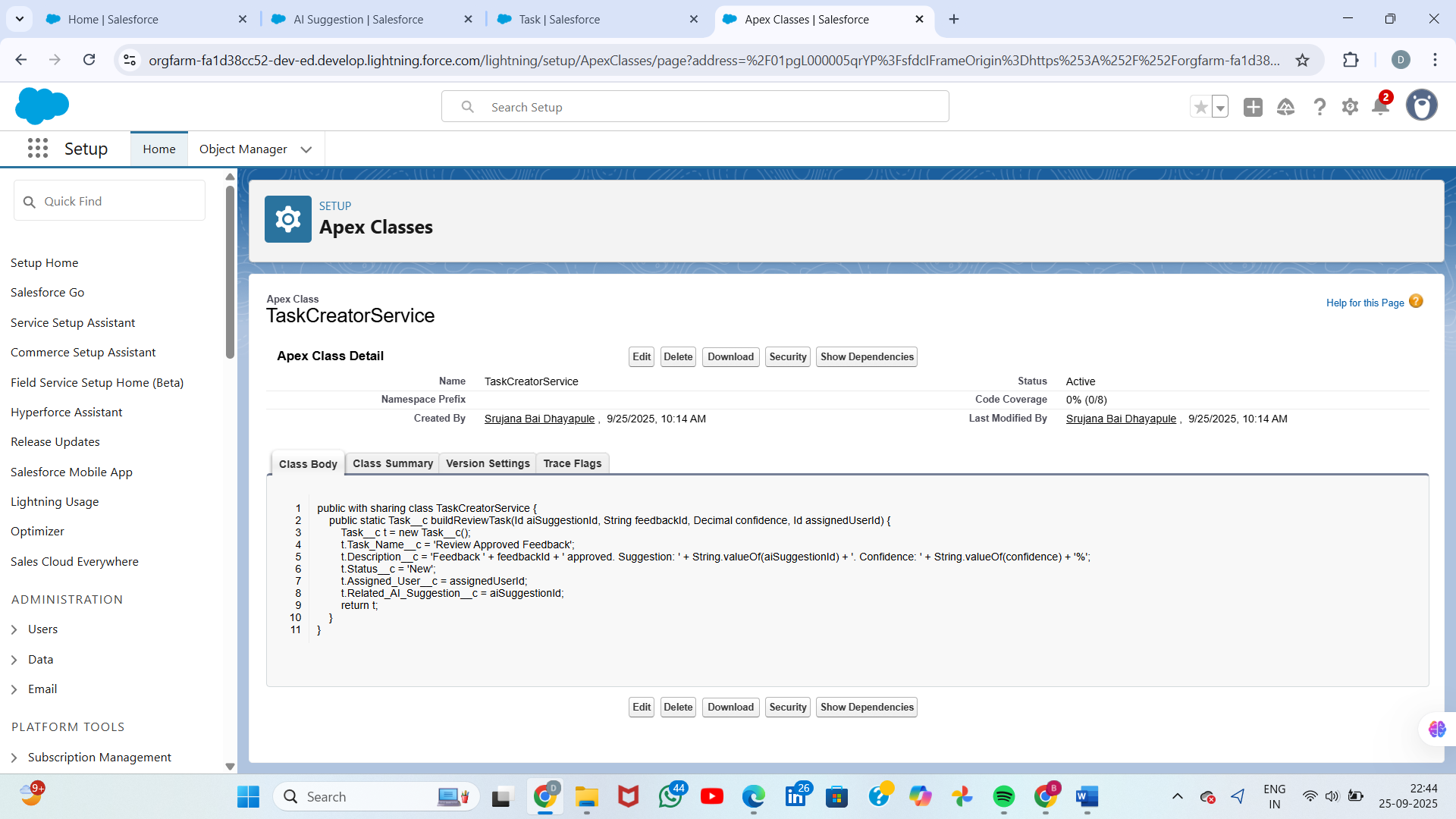
* **Data Type:** This should be a **Lookup Relationship** to the **User** object.
* **Related to:** User
* **Field Label:** Assigned User
* **Field Name:** Assigned\_User

  
**Related\_AI\_Suggestion\_\_c:**

* **Data Type:** This should be a **Lookup Relationship** to the AI\_Suggestion\_\_c object (or whatever your suggestion object is named).
* **Related to:** AI Suggestion (or the name of your object)
* **Field Label:** Related AI Suggestion
* **Field Name:** Related\_AI\_Suggestionc 

### Steps

1. Setup → Apex Classes → New.
2. Create service/handler class file names: e.g., AIProcessingService, TaskCreatorService, FeedbackHandler.
3. Keep methods public static for trigger handlers or instance methods for complex services.



## Step 2 : Apex Programming

## Implementation Steps

### Step 1: Identify the Requirement

Business rule: A Task\_\_c record must not have the same Name as an existing record.  
Example: If a task named 'Test' already exists, the system should prevent another 'Test' task from being saved.

### Step 2: Build the Trigger

We implemented a before insert, before update trigger on Task\_\_c to block duplicates:

trigger PreventDuplicateTaskName\_TaskCustom on Task\_\_c (before insert, before update) {  
 Set<String> normNames = new Set<String>();  
 Map<String, List<Task\_\_c>> triggerNameMap = new Map<String, List<Task\_\_c>>();  
  
 for (Task\_\_c t : Trigger.new) {  
 if (t.Name != null) {  
 String n = t.Name.trim().toLowerCase();  
 normNames.add(n);  
 if (!triggerNameMap.containsKey(n)) {  
 triggerNameMap.put(n, new List<Task\_\_c>());  
 }  
 triggerNameMap.get(n).add(t);  
 }  
 }  
  
 if (normNames.isEmpty()) {  
 return;  
 }  
  
 Set<Id> incomingIds = new Set<Id>();  
 if (Trigger.isUpdate) {  
 for (Task\_\_c t : Trigger.new) {  
 if (t.Id != null) incomingIds.add(t.Id);  
 }  
 }  
  
 Map<String, List<Task\_\_c>> existingMap = new Map<String, List<Task\_\_c>>();  
 for (Task\_\_c ex : [  
 SELECT Id, Name  
 FROM Task\_\_c  
 WHERE Name IN :normNames  
 AND Id NOT IN :incomingIds  
 ]) {  
 String exNorm = ex.Name == null ? '' : ex.Name.trim().toLowerCase();  
 if (!existingMap.containsKey(exNorm)) {  
 existingMap.put(exNorm, new List<Task\_\_c>());  
 }  
 existingMap.get(exNorm).add(ex);  
 }  
  
 for (String n : triggerNameMap.keySet()) {  
 List<Task\_\_c> incomingList = triggerNameMap.get(n);  
 Boolean duplicateFound = (incomingList.size() > 1) || existingMap.containsKey(n);  
  
 if (duplicateFound) {  
 for (Task\_\_c t : incomingList) {  
 t.addError('A Task with the name "' + t.Name + '" already exists. Please choose a different name.');  
 }  
 }  
 }  
}

### Step 3: Build the Test Class

To ensure deployment and validate the logic, a test class was written:

@isTest  
private class PreventDuplicateTaskName\_TaskCustom\_Test {  
 @isTest static void testSingleDuplicateBlocked() {  
 Task\_\_c t1 = new Task\_\_c(Name = 'Test Duplicate');  
 insert t1;  
  
 Task\_\_c t2 = new Task\_\_c(Name = 'Test Duplicate');  
 Test.startTest();  
 try {  
 insert t2;  
 System.assert(false, 'Expected DMLException due to duplicate name, but insert succeeded.');  
 } catch (DmlException e) {  
 System.assert(e.getMessage().contains('A Task with the name') , 'Expected duplicate error message.');  
 }  
 Test.stopTest();  
 }  
  
 @isTest static void testBulkDuplicatesInTriggerBlocked() {  
 Task\_\_c a = new Task\_\_c(Name = 'BulkDup');  
 Task\_\_c b = new Task\_\_c(Name = 'BulkDup');  
 List<Task\_\_c> lst = new List<Task\_\_c> { a, b };  
  
 Test.startTest();  
 try {  
 insert lst;  
 System.assert(false, 'Expected DMLException due to duplicate names in bulk insert.');  
 } catch (DmlException e) {  
 System.assert(e.getMessage().contains('A Task with the name'), 'Expected duplicate error message.');  
 }  
 Test.stopTest();  
 }  
  
 @isTest static void testUpdateToDuplicateBlocked() {  
 Task\_\_c t1 = new Task\_\_c(Name = 'OriginalOne');  
 insert t1;  
  
 Task\_\_c t2 = new Task\_\_c(Name = 'UniqueTwo');  
 insert t2;  
  
 t2.Name = 'OriginalOne';  
 Test.startTest();  
 try {  
 update t2;  
 System.assert(false, 'Expected DMLException due to update creating duplicate.');  
 } catch (DmlException e) {  
 System.assert(e.getMessage().contains('A Task with the name'), 'Expected duplicate error message.');  
 }  
 Test.stopTest();  
 }  
}

### Step 4: Deploy & Test

• Deploy trigger and test class using VS Code + SFDX or Developer Console.

• Run all tests — they should pass successfully.

• Try creating duplicate tasks in Salesforce UI to confirm that the system now blocks duplicates with an error message.

## Key Outcomes

• No duplicate Task\_\_c records are allowed.

• Users receive a clear error message when trying to save duplicates.

• Code is bulk-safe and passes Salesforce test coverage requirements.