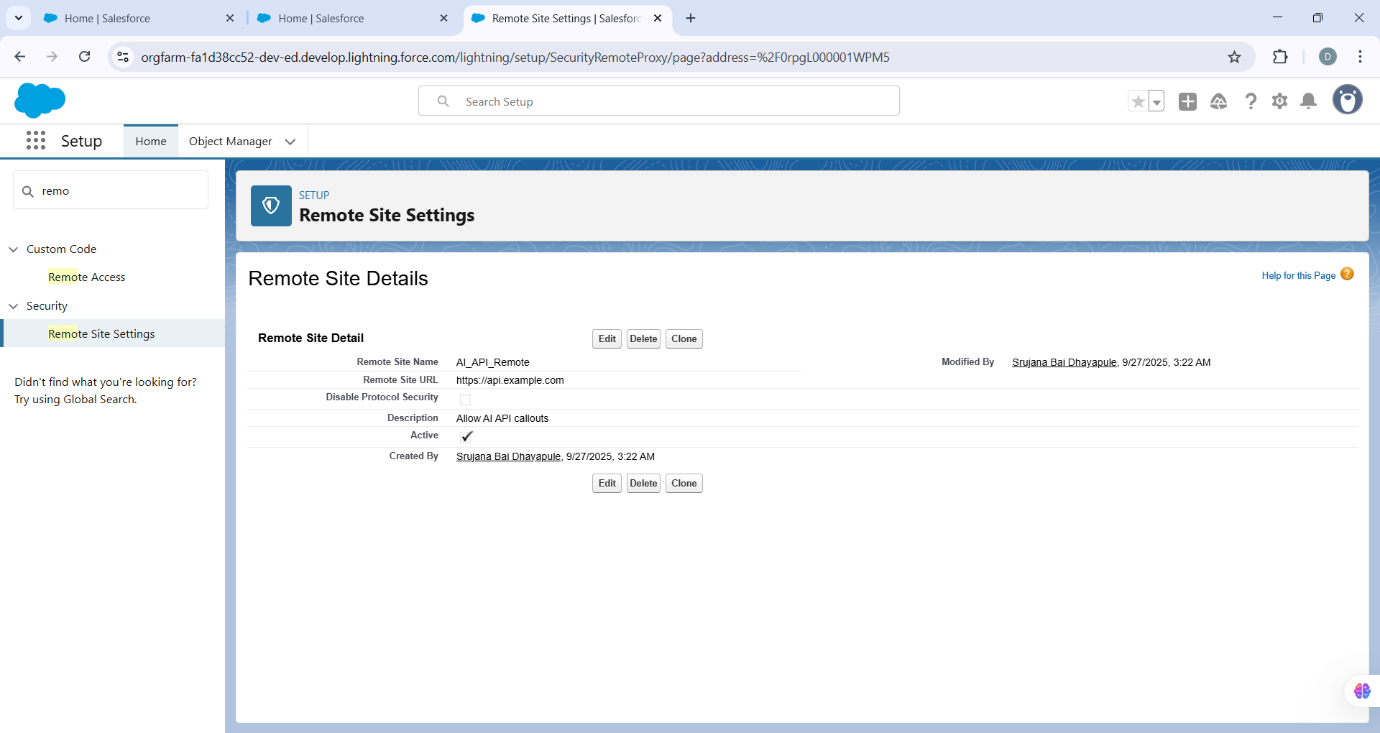
Phase 7 – Integration & External Access

**Configure Salesforce Remote Site Setting**

* Navigate to **Setup → Remote Site Settings → New Remote Site**
* **Remote Site Name:** AIServiceRemoteCallout
* **Remote Site URL:** https://api.example.com
* **Description:** Allows Salesforce to call the external AI service.
* Check **Active** and save.

**Note:** Salesforce requires Remote Site Settings to authorize callouts to external domains.  
  
  


**3. Write Apex Callout Class**

Create a class that sends requests and processes responses with robust error handling.

public class AIServiceRemoteCallout {

public class Suggestion {

public String text;

public Integer confidence;

}

public class ResponseWrapper {

public List<Suggestion> suggestions;

}

public static ResponseWrapper fetchSuggestions(String prompt) {

HttpRequest req = new HttpRequest();

req.setEndpoint('https://api.example.com/v1/suggestions');

req.setMethod('POST');

req.setHeader('Authorization', 'Bearer YOUR\_API\_KEY');

req.setHeader('Content-Type', 'application/json');

req.setBody(JSON.serialize(new Map<String, String>{'prompt' => prompt}));

Http http = new Http();

try {

HttpResponse res = http.send(req);

if(res.getStatusCode() == 200){

return (ResponseWrapper)JSON.deserialize(res.getBody(), ResponseWrapper.class);

} else {

System.debug('Error response: ' + res.getBody());

return null;

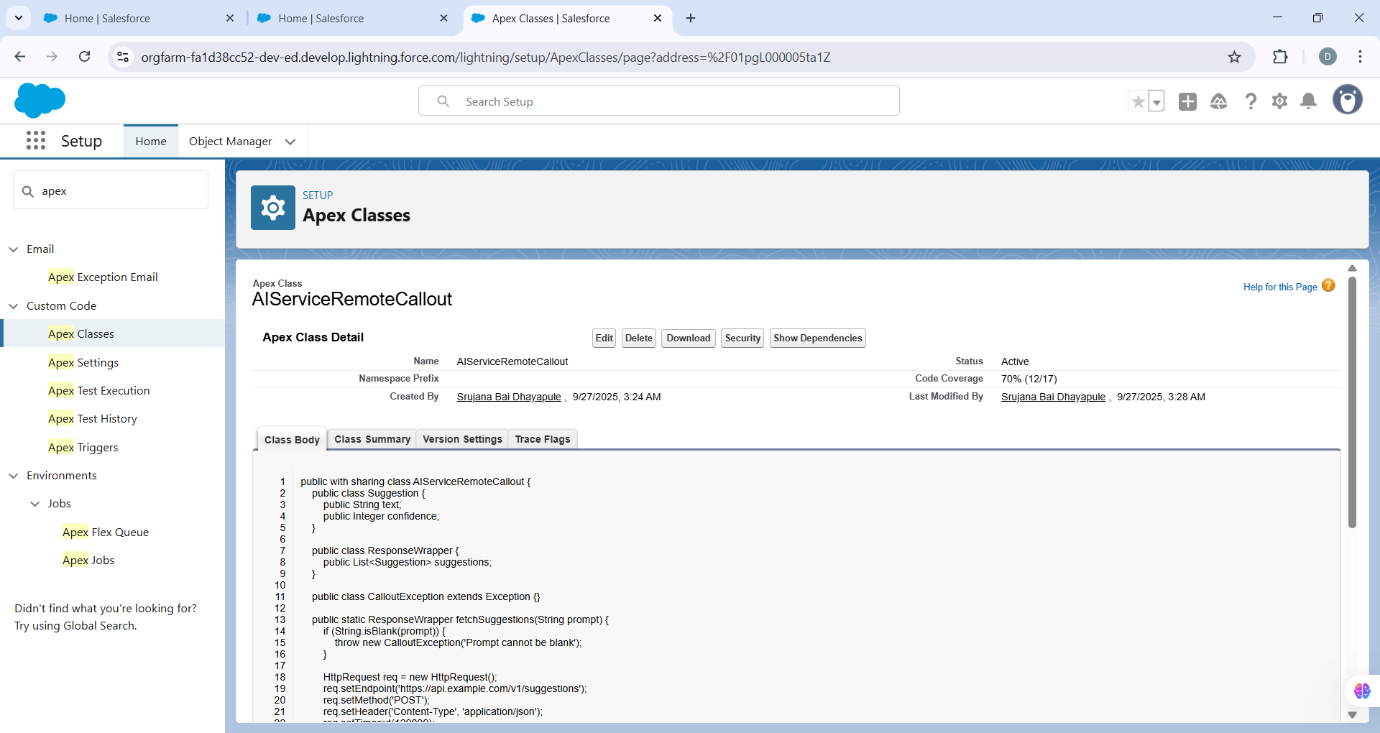
}

} catch(Exception e){

System.debug('Callout failed: ' + e.getMessage());

return null;

}

}  


}

**4. Add Async Wrapper (Queueable + Database.AllowsCallouts)**

To handle multiple records and avoid hitting limits, a Queueable class processes tasks asynchronously.

public class AIIntegrationQueueable\_Save\_Custom implements Queueable, Database.AllowsCallouts {

private List<Id> recordIds;

public AIIntegrationQueueable\_Save\_Custom(List<Id> recordIds){

this.recordIds = recordIds;

}

public void execute(QueueableContext ctx){

if(recordIds == null || recordIds.isEmpty()) return;

List<Task\_\_c> tasks = [SELECT Id, Name, Description\_\_c FROM Task\_\_c WHERE Id IN :recordIds];

List<AI\_Suggestion\_\_c> toInsert = new List<AI\_Suggestion\_\_c>();

for(Task\_\_c t : tasks){

String prompt = t.Name;

if(String.isBlank(prompt)) prompt = t.Description\_\_c;

if(String.isBlank(prompt)) continue;

try{

AIServiceRemoteCallout.ResponseWrapper resp = AIServiceRemoteCallout.fetchSuggestions(prompt);

if(resp != null && resp.suggestions != null){

for(AIServiceRemoteCallout.Suggestion s : resp.suggestions){

AI\_Suggestion\_\_c rec = new AI\_Suggestion\_\_c();

rec.Related\_Task\_\_c = t.Id;

rec.Suggested\_Text\_\_c = s.text;

rec.Confidence\_Score\_\_c = s.confidence;

rec.Status\_\_c = 'Draft';

toInsert.add(rec);

}

}

} catch(Exception e){

AI\_Suggestion\_\_c err = new AI\_Suggestion\_\_c();

err.Related\_Task\_\_c = t.Id;

err.Suggested\_Text\_\_c = 'ERROR: ' + e.getMessage();

err.Confidence\_Score\_\_c = 0;

err.Status\_\_c = 'Draft';

toInsert.add(err);

}

}

if(!toInsert.isEmpty()){

try{

insert toInsert;

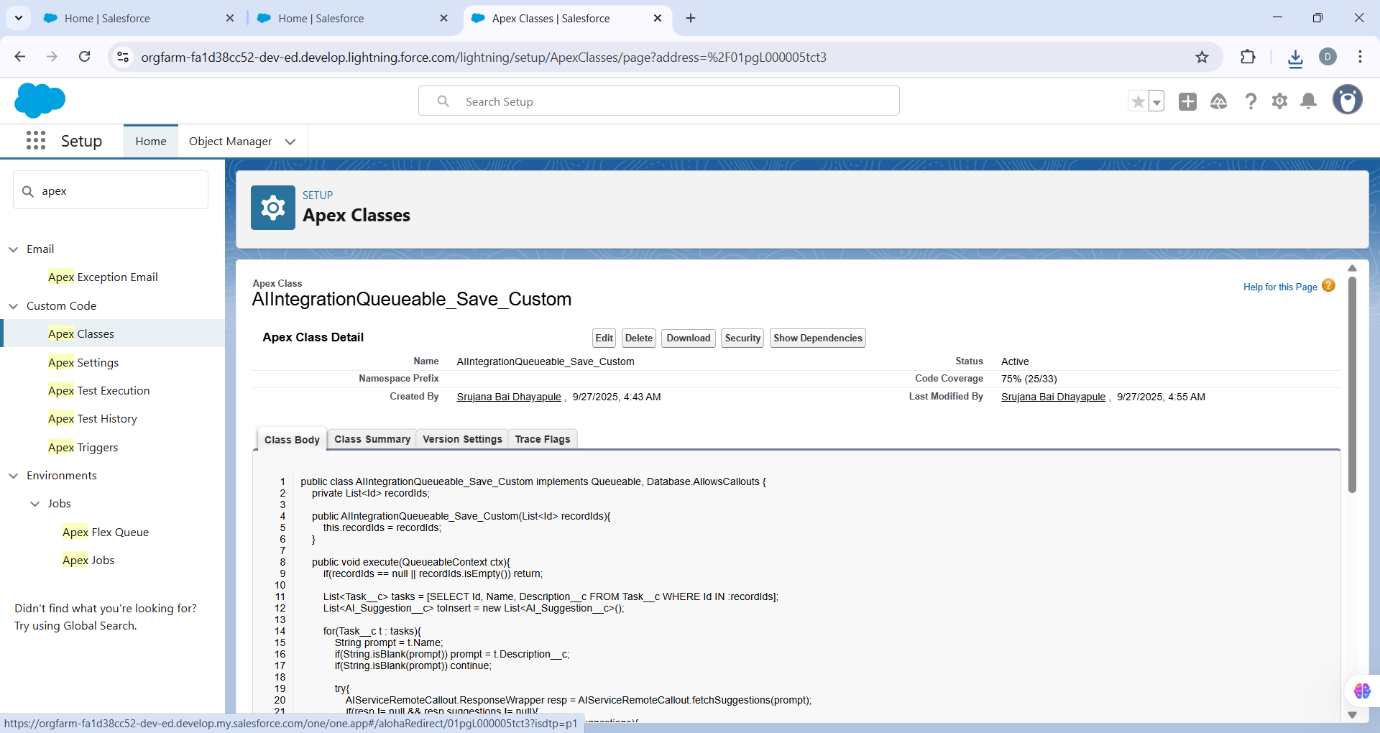
} catch(Exception e){

System.debug('Failed to insert AI\_Suggestion\_\_c records: ' + e.getMessage());

}

}

}

}  
  
  


**5. Add Trigger (Bulk-Safe)**

trigger TaskAfterInsert on Task\_\_c (after insert) {

List<Id> ids = new List<Id>();

for(Task\_\_c t : Trigger.new){

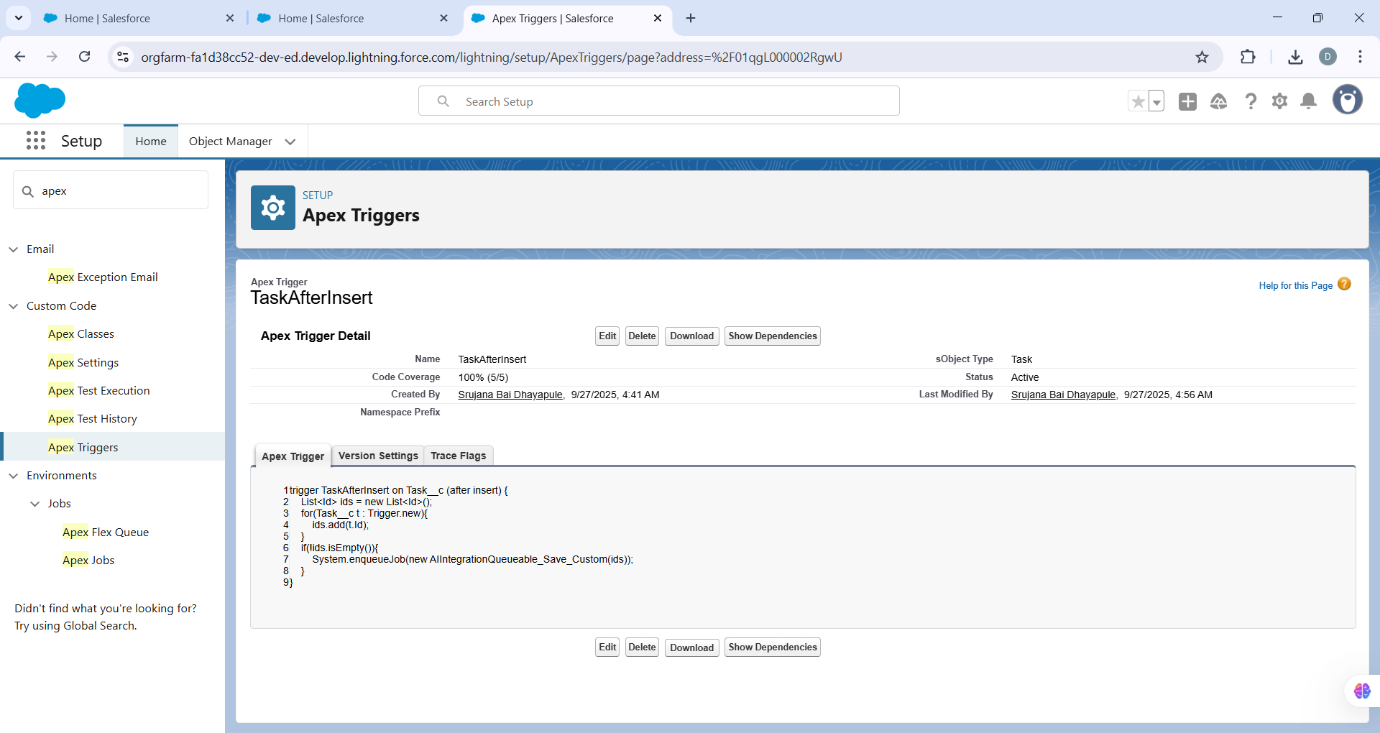
ids.add(t.Id);

}

if(!ids.isEmpty()){

System.enqueueJob(new AIIntegrationQueueable\_Save\_Custom(ids));

}

}  
  


**6. Test Class Using HttpCalloutMock**

@isTest

public class AIServiceRemoteCallout\_Test {

private class MockAIService implements HttpCalloutMock {

public HTTPResponse respond(HTTPRequest req){

HttpResponse res = new HttpResponse();

res.setHeader('Content-Type','application/json');

res.setBody('{"suggestions":[{"text":"Test Suggestion 1","confidence":90},{"text":"Test Suggestion 2","confidence":80}]}');

res.setStatusCode(200);

return res;

}

}

@isTest

static void testQueueableExecution() {

Test.setMock(HttpCalloutMock.class, new MockAIService());

List<Task\_\_c> tasks = new List<Task\_\_c>();

tasks.add(new Task\_\_c(Name = 'Test Task 1'));

tasks.add(new Task\_\_c(Name = 'Test Task 2'));

insert tasks;

List<Id> taskIds = new List<Id>();

for(Task\_\_c t : tasks){

taskIds.add(t.Id);

}

Test.startTest();

System.enqueueJob(new AIIntegrationQueueable\_Save\_Custom(taskIds));

Test.stopTest();

List<AI\_Suggestion\_\_c> suggestions = [SELECT Related\_Task\_\_c, Suggested\_Text\_\_c, Confidence\_Score\_\_c, Status\_\_c

FROM AI\_Suggestion\_\_c

WHERE Related\_Task\_\_c IN :taskIds];

System.assertEquals(4, suggestions.size()); // 2 suggestions per task

for(AI\_Suggestion\_\_c s : suggestions){

System.assert(s.Suggested\_Text\_\_c.contains('Test Suggestion'));

}

}

}

