Project Title: Smart Pharmacy Inventory Tracker

Phase 7: Integration & External Access

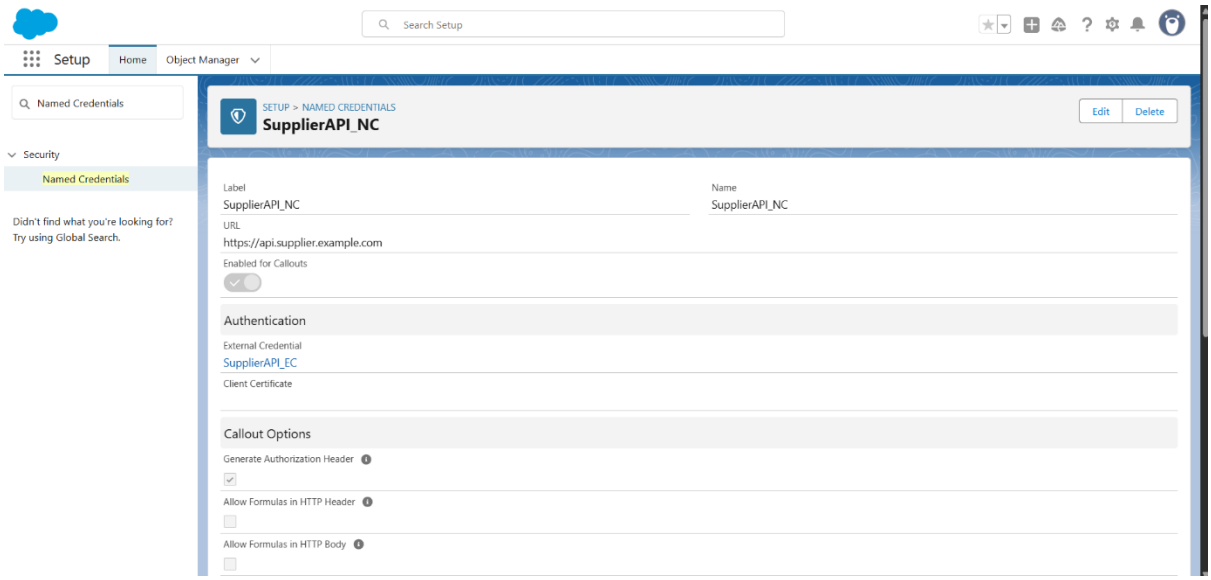
Executive Summary

Phase 7 focused on enabling the Smart Pharmacy application to connect with external systems, exchange data securely, and provide near real-time updates. The objective was to extend the project from an isolated system into a connected solution for suppliers, inventory checks, and managers. This was achieved using Named Credentials, Apex Callouts, Platform Events, Change Data Capture, Custom REST/SOAP APIs, and OAuth Authentication. Together, these integrations allow the pharmacy to stay updated with supplier APIs, notify stakeholders in real time, and ensure security while optimizing API usage.

Named Credentials:

□ **Purpose/Rationale:** To securely store external API endpoints and authentication information, avoiding hard-coded URLs or passwords in Apex.

□ **Detailed Implementation:** A Named Credential SupplierAPI_NC was created pointing to https://api.supplier.example.com. This was used in Apex callouts for supplier stock and pricing updates. By referencing it in code (callout:SupplierAPI_NC), sensitive information was abstracted away and the system followed Salesforce security best practices.



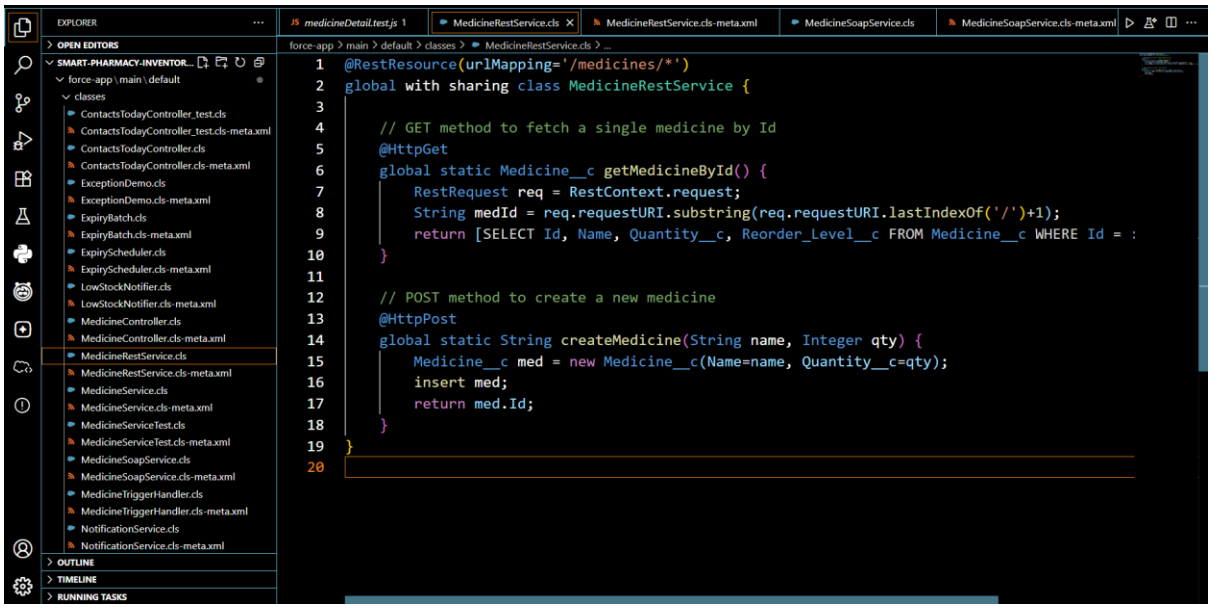
External Services:

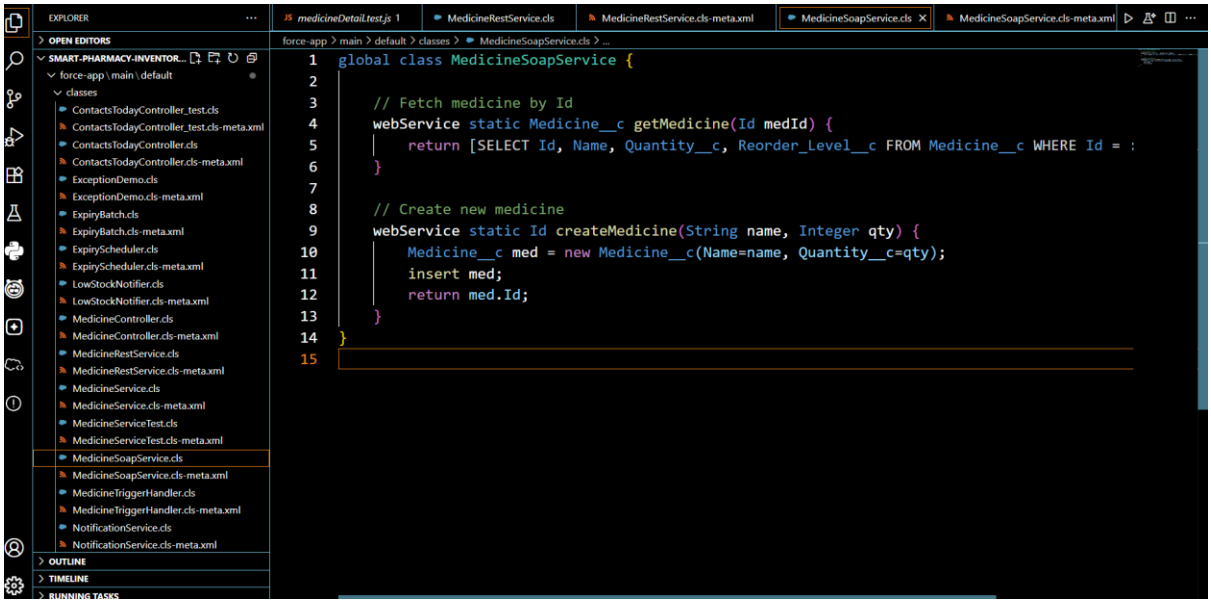
 Purpose/Rationale: External Services provide a declarative way to connect Salesforce to REST APIs that have OpenAPI specifications.

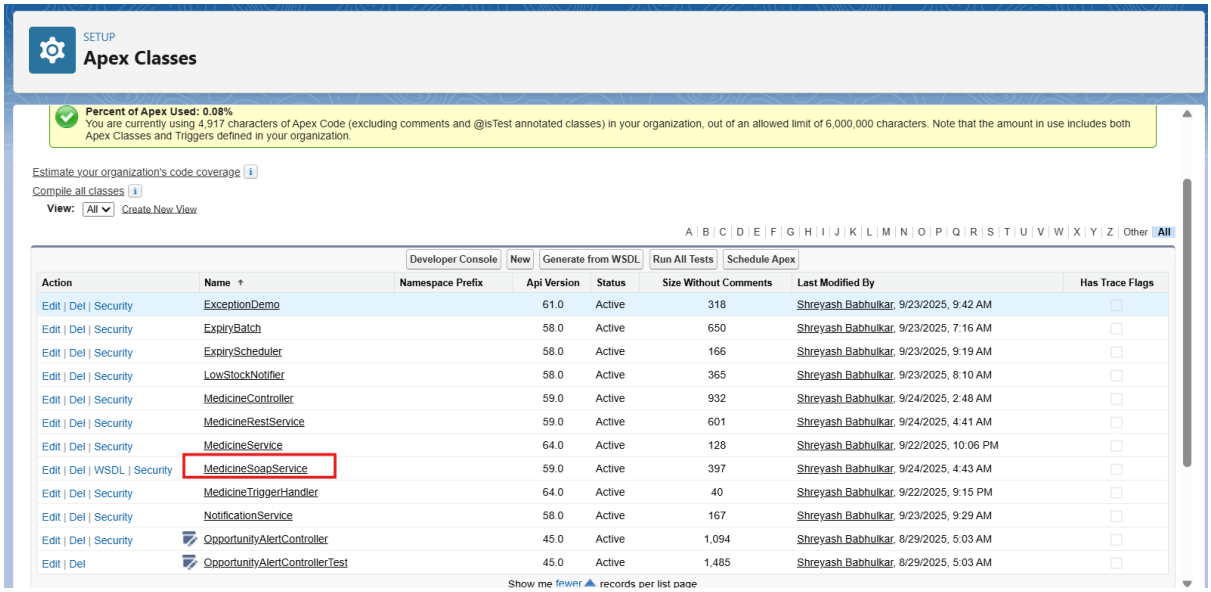
 Detailed Implementation: This feature was evaluated for future use. For example, if a supplier exposes an OpenAPI spec for automatic restocking, Smart Pharmacy can declaratively create an Invocable Action. This would allow a Flow to trigger supplier restocking directly, without complex Apex code.

Web Services (REST/SOAP):

- Purpose/Rationale: To expose Smart Pharmacy data (medicines, orders, suppliers) to external systems.
- Detailed Implementation:
    - A REST Service (MedicineRestService.cls) was built to return medicine details as JSON when queried by external systems.
    - A SOAP Service (MedicineSoapService.cls) was created with webservice methods to allow partners to create or query medicines using WSDL.







Callouts:

□ Purpose/Rationale: Apex callouts were implemented to fetch supplier data, synchronize stock, and check expiry notifications.

□ Detailed Implementation: A callout class SupplierApiService.cls was created to call SupplierAPI_NC using HTTP GET. Mock classes were written for test execution. Callouts were marked @future(callout=true) or wrapped in Queueable Apex for asynchronous execution.

```apex
public with sharing class SupplierApiService {
    // Inner class to deserialize JSON response
    public class SupplierResponse {
        public String id;
        public String name;
        public String status;
    }

    // Method to fetch suppliers using Named Credential
    public static List<SupplierResponse> fetchSuppliers() {
        Http h = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('callout:SupplierAPI_NC/v1/suppliers'); // use your Named Credential
        req.setMethod('GET');
        HttpResponse res = h.send(req);

        if (res.getStatusCode() != 200) {
            throw new CalloutException('Failed: ' + res.getStatus());
        }

        // Deserialize JSON into list of SupplierResponse
        return (List<SupplierResponse>) JSON.deserialize(res.getBody(), List<SupplierRespons
    }
}
```

```apex
@isTest
Run All Tests | Debug All Tests
global class SupplierApiMock implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest req) {
        HttpResponse res = new HttpResponse();
        res.setStatusCode(200);
        res.setHeader('Content-Type','application/json');
        res.setBody('[{"id":"S1","name":"ACME Supplies","status":"active"}]');
        return res;
    }
}

@isTest
private class SupplierApiServiceTest {
    @isTest static void testFetchSuppliers() {
        // Set mock
        Test.setMock(HttpCalloutMock.class, new SupplierApiMock());

        Test.startTest();
        List<SupplierApiService.SupplierResponse> list = SupplierApiService.fetchSuppliers();
        Test.stopTest();

        // Verify result
        System.assertEquals(1, list.size());
        System.assertEquals('ACME Supplies', list[0].name);
    }
```

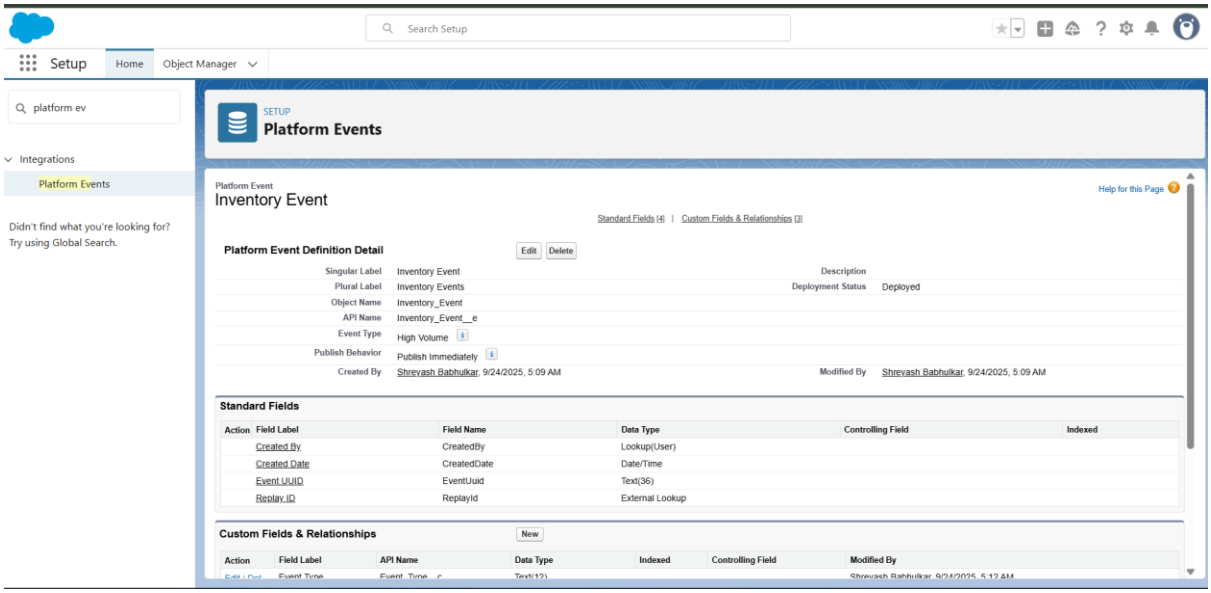
Platform Events:

□ Purpose/Rationale: Used for real-time communication inside Salesforce and with external systems.

□ Detailed Implementation: A Platform Event Inventory_Event__e was created with fields: MedicineId__c, Quantity__c, and EventType__c.

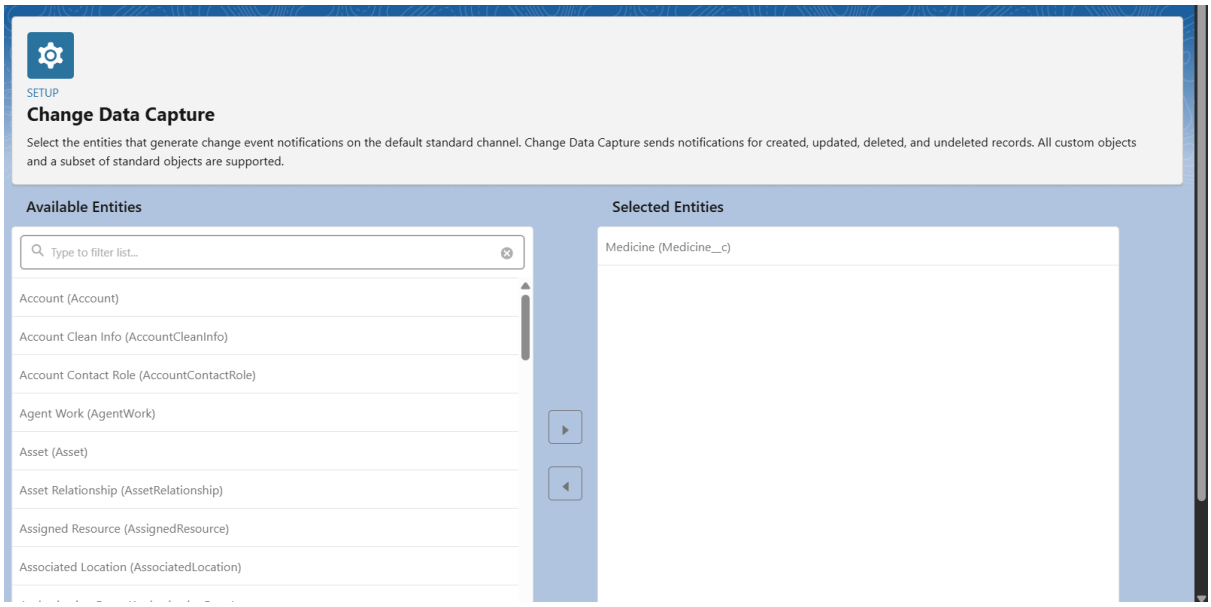
- Whenever a medicine is updated, Apex publishes an event.

- Subscribers (triggers, LWCs, or external CometD clients) receive updates instantly, e.g., when stock is updated or medicine expires.



Change Data Capture (CDC):

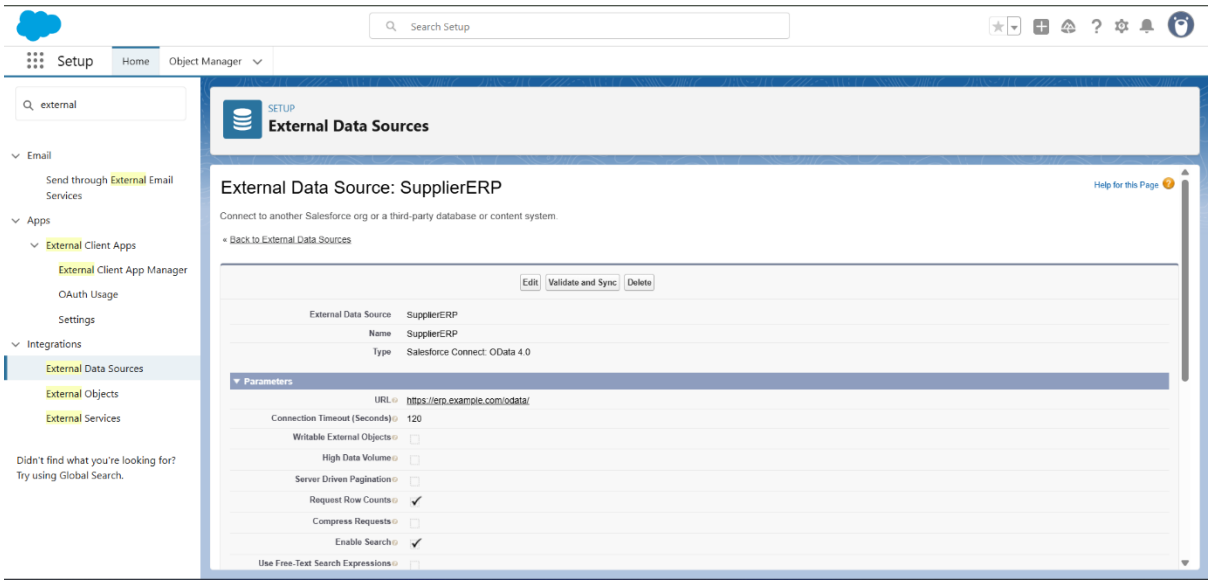
- Purpose/Rationale: To stream near real-time changes of Salesforce records (medicine stock/expiry) without polling.
- Detailed Implementation**:** CDC was enabled for Medicine__c. A Lightning Web Component (cdcSubscriber) subscribed to /data/Medicine__ChangeEvent. When a medicine's quantity or expiry changes, the UI updates live. External systems can also subscribe through CometD.



Salesforce Connect:

 Purpose/Rationale: To access external supplier databases directly without storing them in Salesforce.

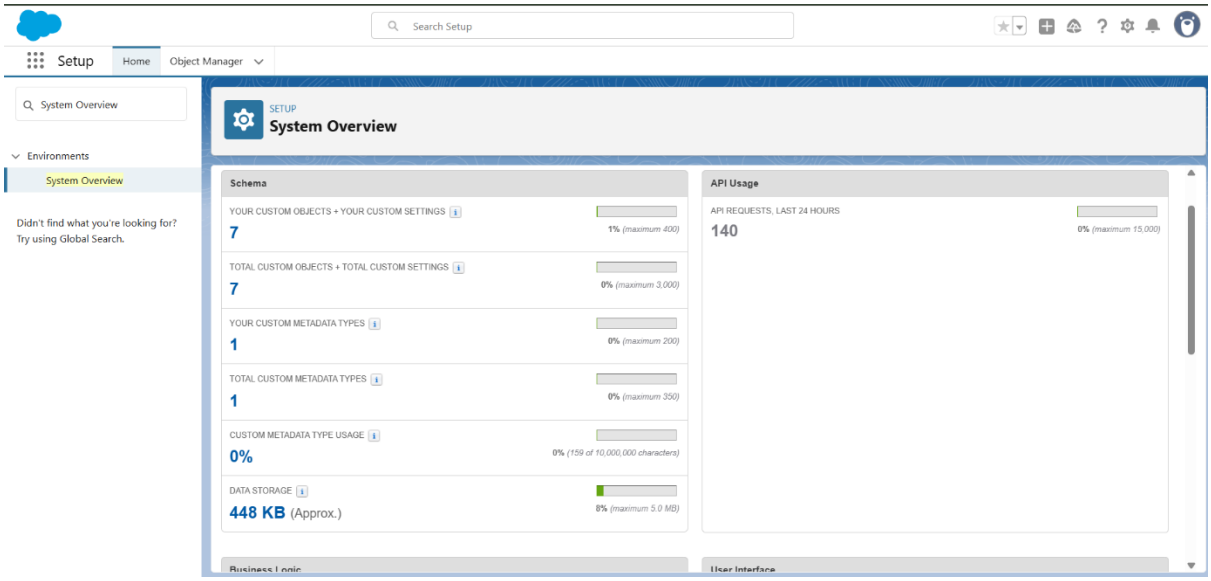
 Detailed Implementation: An external object Supplier_Master__x was evaluated using OData. This allows Smart Pharmacy users to view live supplier stock from an external SQL system as if it were native Salesforce data, without duplication.



API Limits:

 Purpose/Rationale: Salesforce enforces daily API call limits. Efficient integration ensures the pharmacy app does not exceed them.

 Detailed Implementation: Instead of polling, CDC and Platform Events were used. Bulk APIs and Composite APIs were planned for large updates. API usage was monitored via System Overview and /services/data/v59.0/limits.



Remote Site Settings:

- Purpose/Rationale: Required when making callouts to external domains without Named Credentials.
- Detailed Implementation: Since Smart Pharmacy primarily used Named Credentials, Remote Site Settings were not required. However, if a direct endpoint was needed (legacy approach), the supplier API URL would be added under Remote Site Settings for callouts.