Phase 7: Integration & External Access

AI-Enabled Hospital & Pharmacy Management System

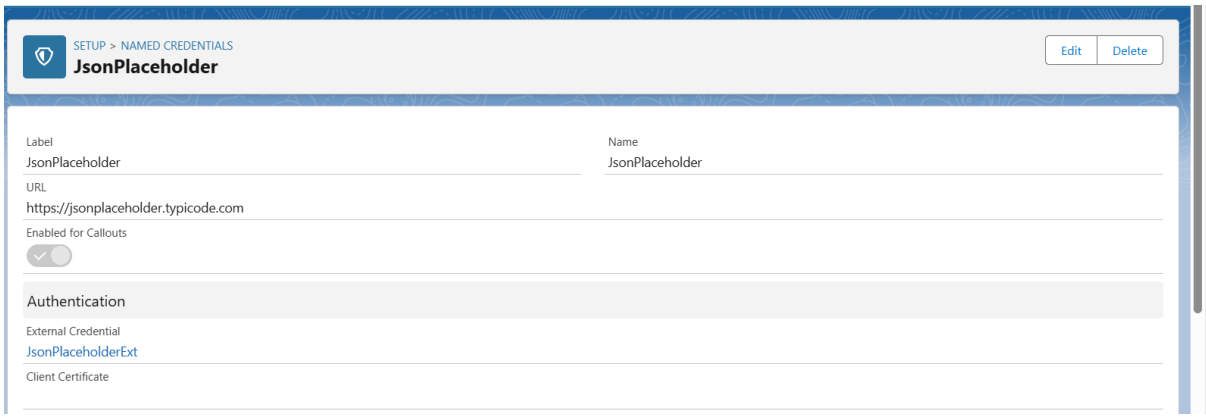
Goal: The goal of Phase 7 was to integrate Salesforce with external systems and services by enabling secure communication and extending Salesforce functionality beyond the platform. This included implementing Named Credentials, External Services, REST/SOAP Web Services, Callouts, Platform Events, Change Data Capture, Salesforce Connect, API Limits, OAuth & Authentication, and Remote Site Settings. The objective was to demonstrate how Salesforce can consume external APIs, expose its own services, and synchronize data in real-time, thereby building a more connected and scalable ecosystem.

Tasks in Phase 7:

- Named Credentials
- External Services
- Web Services (REST/SOAP)
- Callouts
- Platform Events
- Change Data Capture
- Salesforce Connect
- API Limits
- OAuth & Authentication
- Remote Site Setting

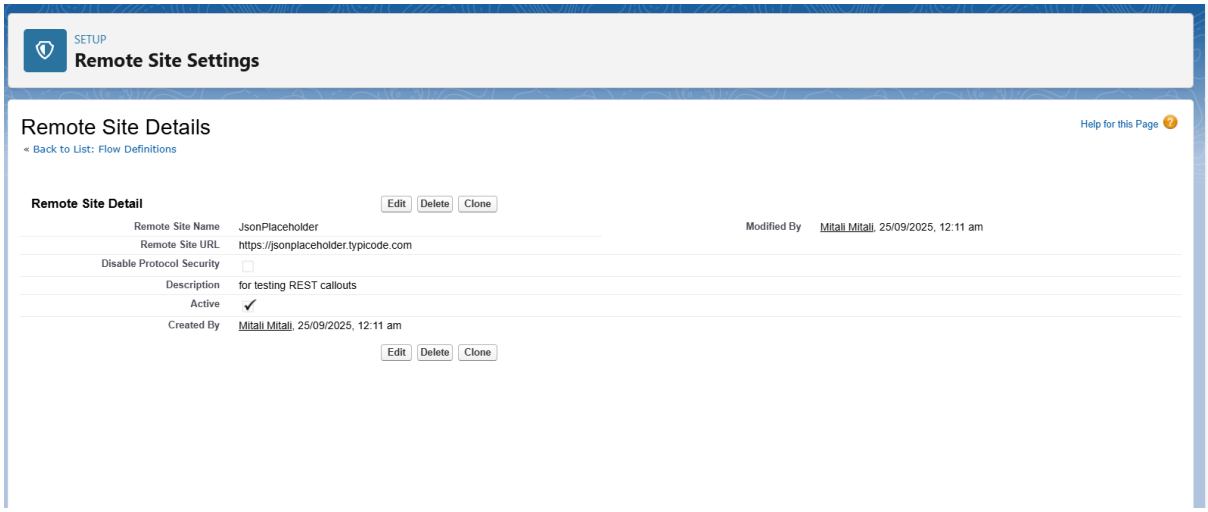
Named Credentials

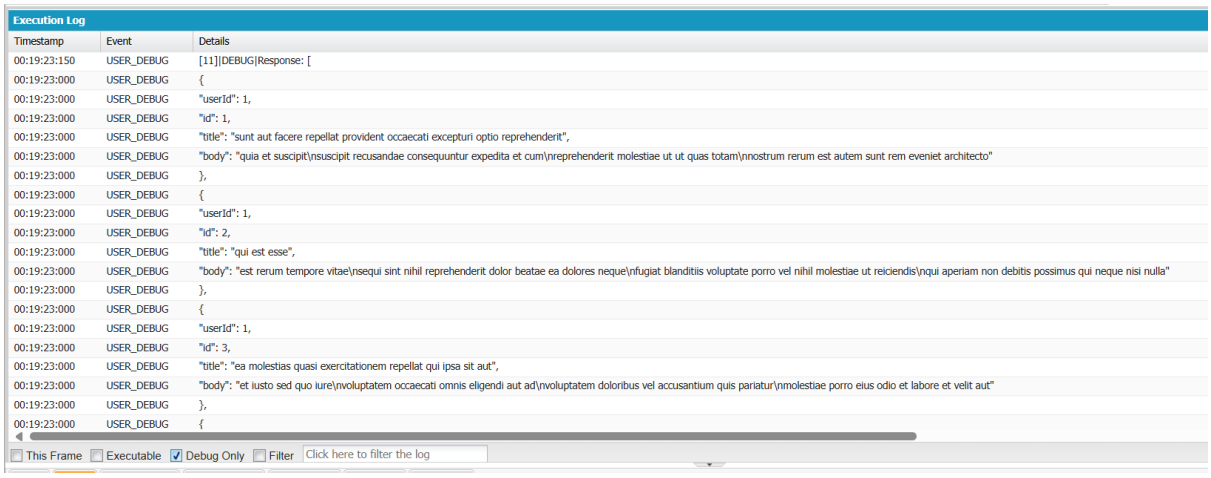
I created a Named Credential (JsonPlaceholder) to securely store the external API URL (https://jsonplaceholder.typicode.com). This allowed me to make callouts without hardcoding URLs in Apex. Named Credentials simplify authentication and improve security.



Remote Site Setting and Callouts

In this step, I implemented integration with an external REST API using Named Credentials and Apex callouts. I created a Named Credential (JsonPlaceholder) pointing to the external API endpoint and then built an Apex class (JsonPlaceholderCallout) to perform a GET request. Using the Developer Console's Execute Anonymous window, I invoked the class method, and Salesforce successfully returned the JSON response (list of posts). This demonstrates secure external API integration in Salesforce.





Execution Log

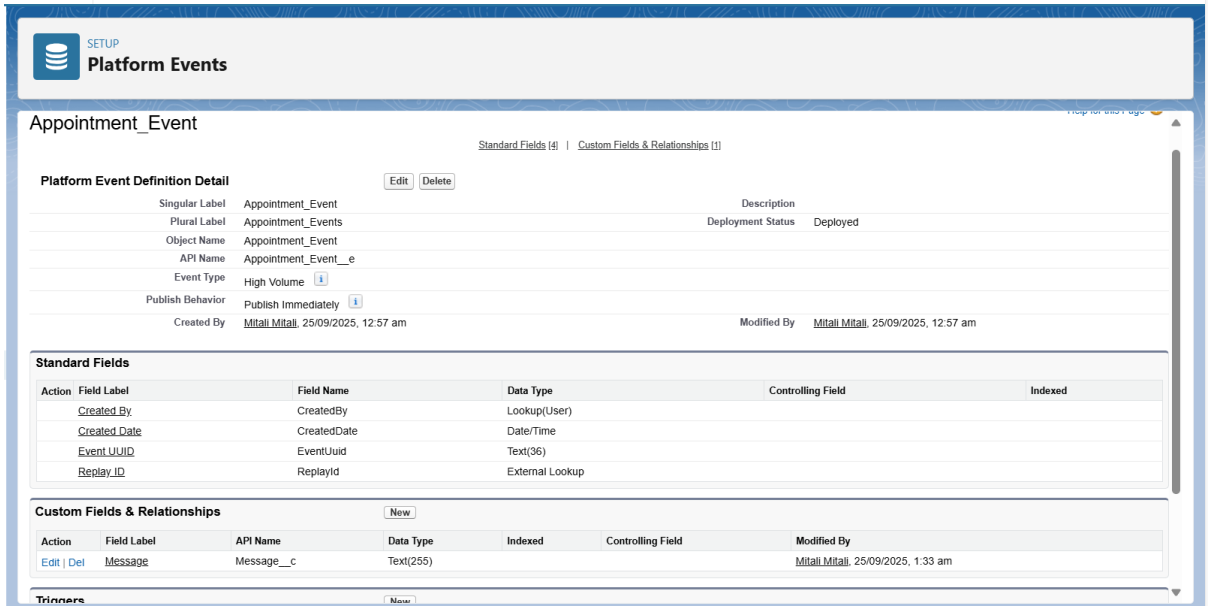
Timestamp	Event	Details
00:19:23:150	USER_DEBUG	[11]|DEBUG|Response: [
00:19:23:000	USER_DEBUG	{
00:19:23:000	USER_DEBUG	"userId": 1,
00:19:23:000	USER_DEBUG	"id": 1,
00:19:23:000	USER_DEBUG	"title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
00:19:23:000	USER_DEBUG	"body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
00:19:23:000	USER_DEBUG	},
00:19:23:000	USER_DEBUG	{
00:19:23:000	USER_DEBUG	"userId": 1,
00:19:23:000	USER_DEBUG	"id": 2,
00:19:23:000	USER_DEBUG	"title": "qui est esse",
00:19:23:000	USER_DEBUG	"body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"
00:19:23:000	USER_DEBUG	},
00:19:23:000	USER_DEBUG	{
00:19:23:000	USER_DEBUG	"userId": 1,
00:19:23:000	USER_DEBUG	"id": 3,
00:19:23:000	USER_DEBUG	"title": "ea molestias quasi exercitationem repellat qui ipsa sit aut",
00:19:23:000	USER_DEBUG	"body": "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut"
00:19:23:000	USER_DEBUG	},
00:19:23:000	USER_DEBUG	{

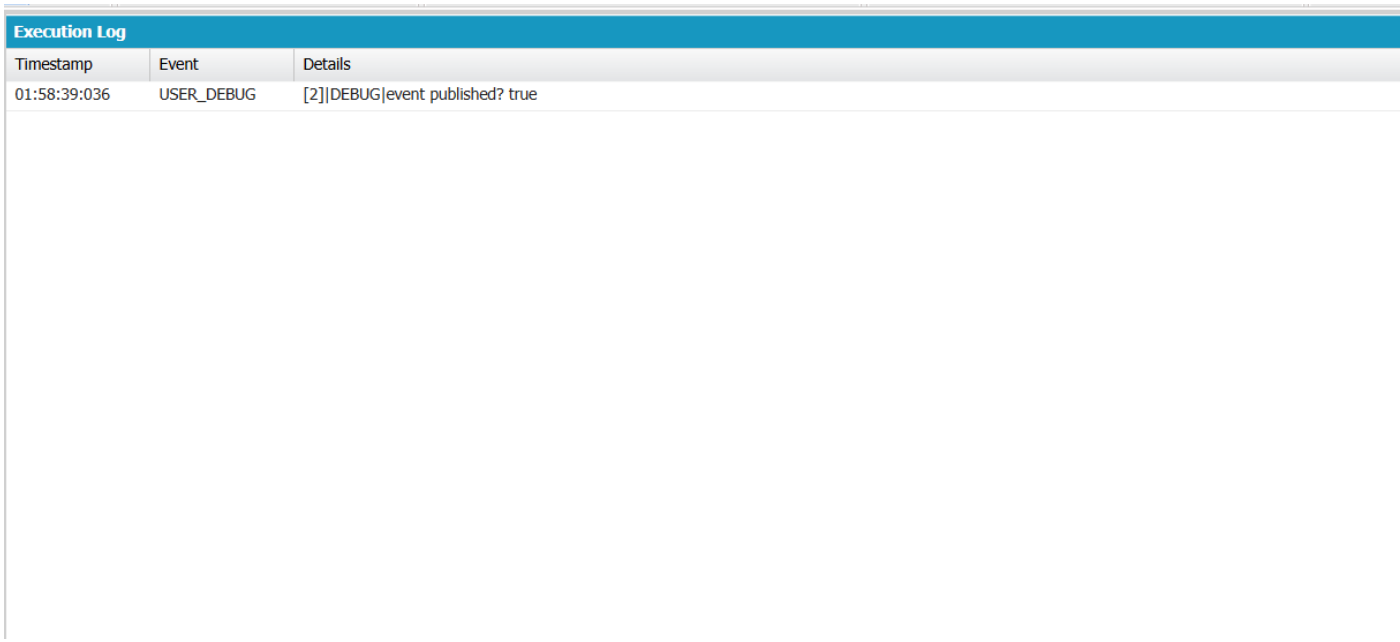
This Frame | Executable | Debug Only | Filter | Click here to filter the log

---

Platform Events

I created a Platform Event Appointment_Event__e to broadcast appointment updates. I published the event from Apex using EventBus.publish() and subscribed using an Apex Trigger. This demonstrated real-time, event-driven communication within Salesforce, suitable for notifying external systems or updating related records.

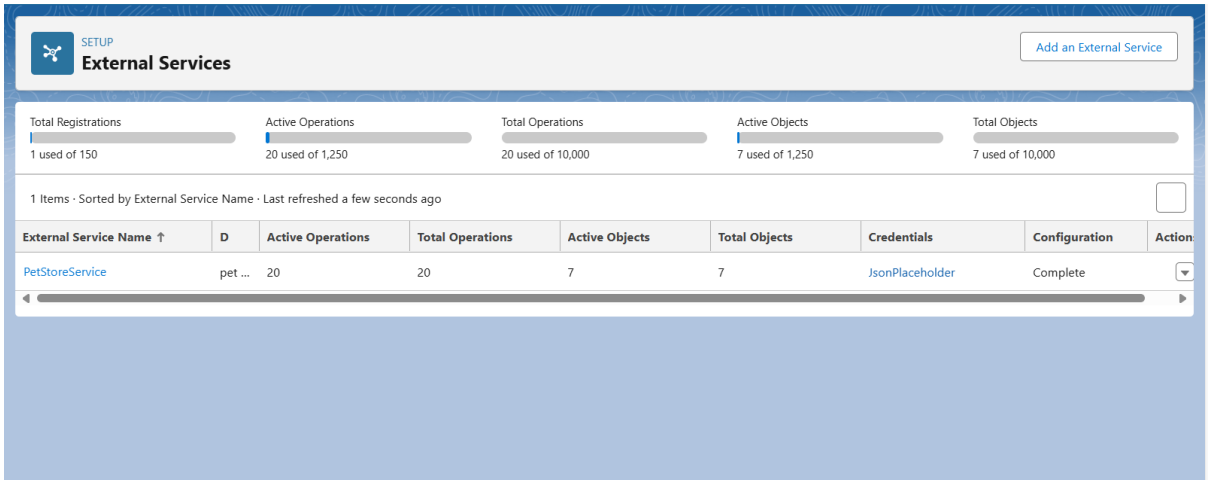




External Services

I registered an External Service using the Swagger schema (https://petstore.swagger.io/v2/swagger.json).

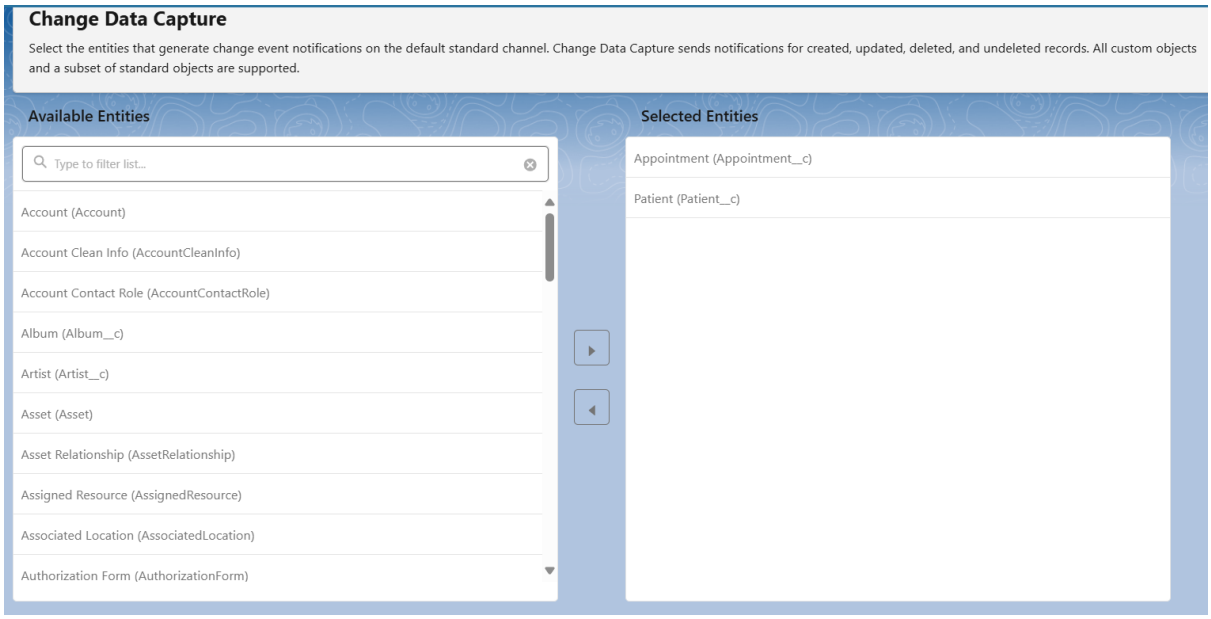
- Salesforce automatically generated actions that can be used in Flows.
- Unsupported media types (like multipart/form-data) were mapped to application/json. This proved how external APIs can be directly consumed inside Salesforce declaratively.



Change Data Capture (CDC)

I enabled Change Data Capture for key objects (Patient__c and Appointment__c).

- This allows Salesforce to automatically fire events whenever records are created/updated/deleted.
- Developers and Flows can subscribe to these events for real-time updates.



Web Services (REST/SOAP)

We created a SOAP web service class PatientSOAPService in Apex. Salesforce generated a WSDL file (shown in screenshot), which can be consumed by external systems. This exposes methods to create a patient and fetch patient records

Edit | Del | WSDL | Security	PatientSOAPService	64.0	Active	417	Mitali

https://caretrack-dev-ed.develop.my.salesforce.com/services/wsdl/class/PatientSOAPService

This XML file does not appear to have any style information associated with it. The document tree is shown below.


```
<!--
 Web Services API : PatientSOAPService
 -->
▼<definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://soap.sforce.com/schemas/class/PatientSOAPService" targetNamespace="http://soap.sforce.com/schemas/class/PatientSOAPService">
 ▼<types>
  ▼<xsd:schema elementFormDefault="qualified" targetNamespace="http://soap.sforce.com/schemas/class/PatientSOAPService">
   ▼<xsd:element name="AllowFieldTruncationHeader">
    ▼<xsd:complexType>
     ▼<xsd:sequence>
        <xsd:element name="allowFieldTruncation" type="xsd:boolean"/>
      </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   ▼<xsd:element name="CallOptions">
    ▼<xsd:complexType>
     ▼<xsd:sequence>
        <xsd:element name="client" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   ▼<xsd:element name="DebuggingHeader">
    ▼<xsd:complexType>
     ▼<xsd:sequence>
        <xsd:element name="categories" minOccurs="0" maxOccurs="unbounded" type="tns:LogInfo"/>
        <xsd:element name="debugLevel" type="tns:LogType"/>
      </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   ▼<xsd:complexType name="LogInfo">
    ▼<xsd:sequence>
        <xsd:element name="category" type="tns:LogCategory"/>
        <xsd:element name="level" type="tns:LogCategoryLevel"/>
    </xsd:sequence>
   </xsd:complexType>
   ▼<xsd:simpleType name="LogCategory">
    ▼<xsd:restriction base="xsd:string">
        <xsd:enumeration value="Db"/>
        <xsd:enumeration value="Workflow"/>
        <xsd:enumeration value="Validation"/>
        <xsd:enumeration value="Callout"/>
        <xsd:enumeration value="Apex_code"/>
        <xsd:enumeration value="Apex_profiling"/>
        <xsd:enumeration value="Visualforce"/>
        <xsd:enumeration value="System"/>
```


The screenshot is the WSDL (Web Services Description Language) file that Salesforce generated for your PatientSOAPService class.

That means:

- Your SOAP web service class compiled successfully.
- Salesforce exposed the methods you wrote (createPatient, getPatients) as SOAP operations.
- Any SOAP client (like Postman, SOAP UI, Java, or .NET apps) can now use this WSDL to integrate with your Salesforce org.

Created a REST Resource Apex Class – We developed an Apex class annotated with @RestResource to expose a custom endpoint /patientAPI.

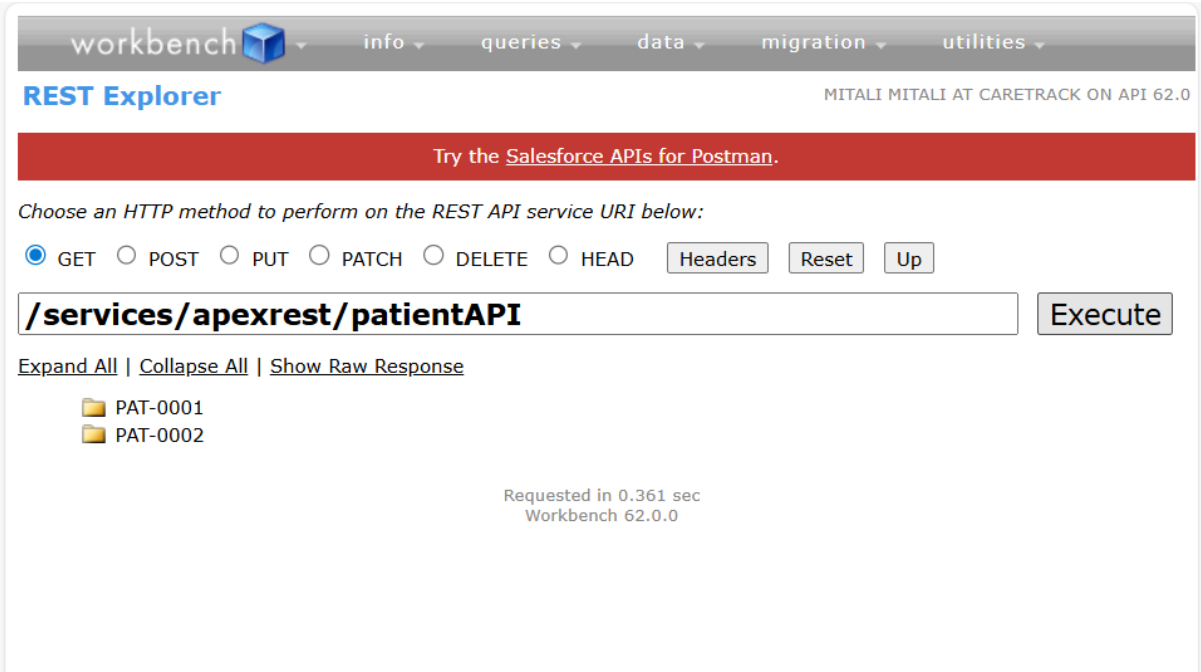
- Implemented @HttpGet to retrieve patient records.
- Implemented @HttpPost to insert a new patient record by accepting JSON input.

Tested using Workbench –

- Used the REST Explorer in Salesforce Workbench.
- Executed GET requests to fetch patient records.
- Executed POST requests by providing JSON request bodies (example: { "name": "Test Patient", "age": 25, "disease": "Fever" }).

Validated Responses – Verified that Workbench displayed the correct JSON responses or inserted the records into Salesforce.

This approach demonstrates how Salesforce can be easily integrated with external systems through REST APIs.



Salesforce Connect

For Salesforce Connect, I configured an External Data Source named Pharmacy_Supplier_Data.

Type: OData 2.0.

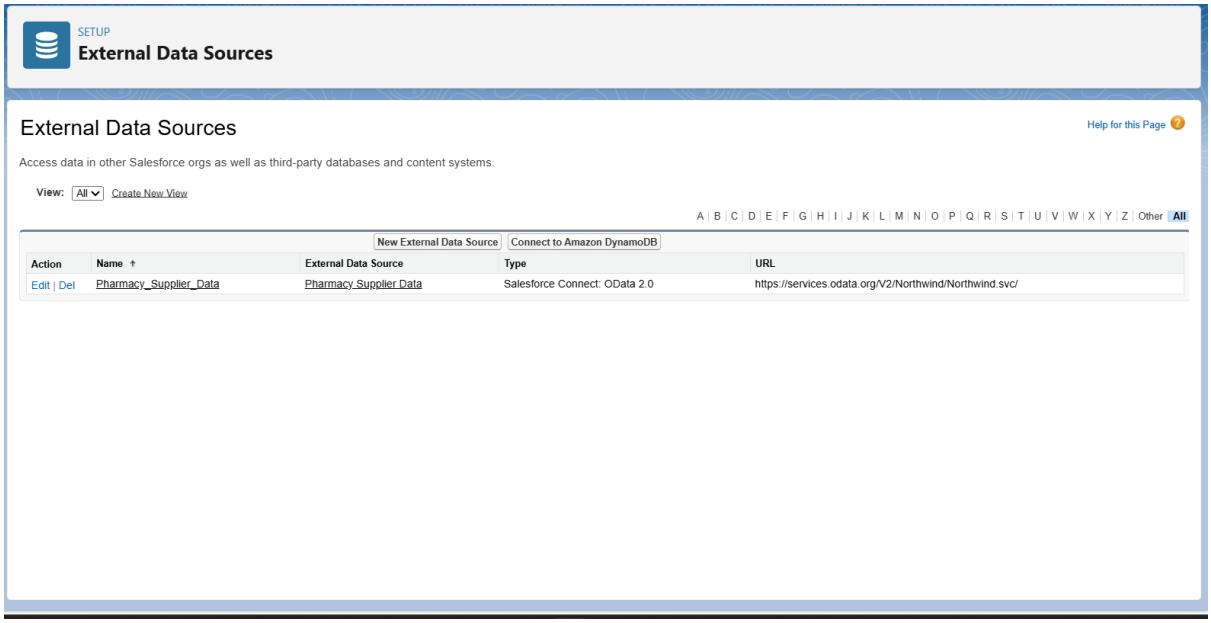
Connected it to the public Northwind OData service:

https://services.odata.org/V2/Northwind/Northwind.svc/

After saving, I used Validate and Sync to pull in external tables (e.g., Products, Orders).

These appeared as External Objects in Salesforce, such as Products__x.

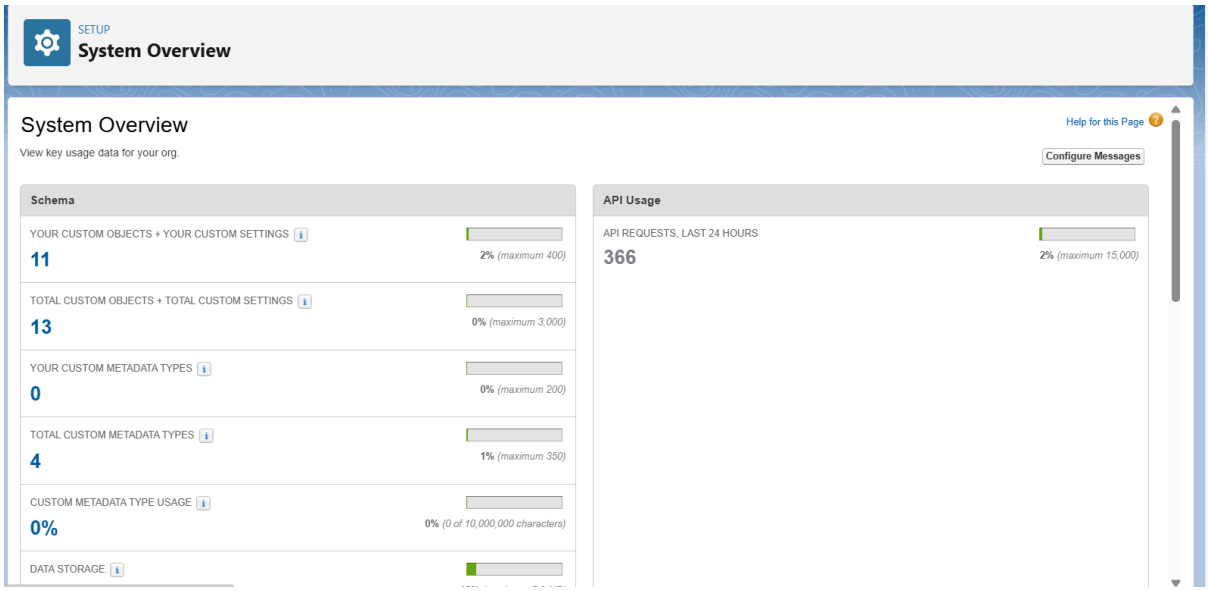
This integration shows how Salesforce can access real-time external data without importing or duplicating it into Salesforce storage.



API Limits

I reviewed API Limits in Setup → System Overview.

- Salesforce enforces daily limits on API calls.
- Monitoring these ensures integrations stay within allowed usage.
  This highlighted governance and scalability considerations in integrations.



OAuth & Authentication

I configured a Connected App with OAuth.

Conclusion

In Phase 7, I implemented different Salesforce integration techniques:

- Named Credentials for secure API access.
- External Services for declarative API consumption.
- REST Callouts for programmatic integration.
- Platform Events and Change Data Capture for real-time communication.
- Salesforce Connect for external object access.
- API Limits, OAuth, and Remote Site Settings to ensure secure and governed integration.

This phase successfully demonstrated Salesforce's ability to integrate with external systems in multiple ways while maintaining security, scalability, and real-time data exchange.