MACK Stores Salesforce CRM Project

Phase 7: Integration & External Access

Objective

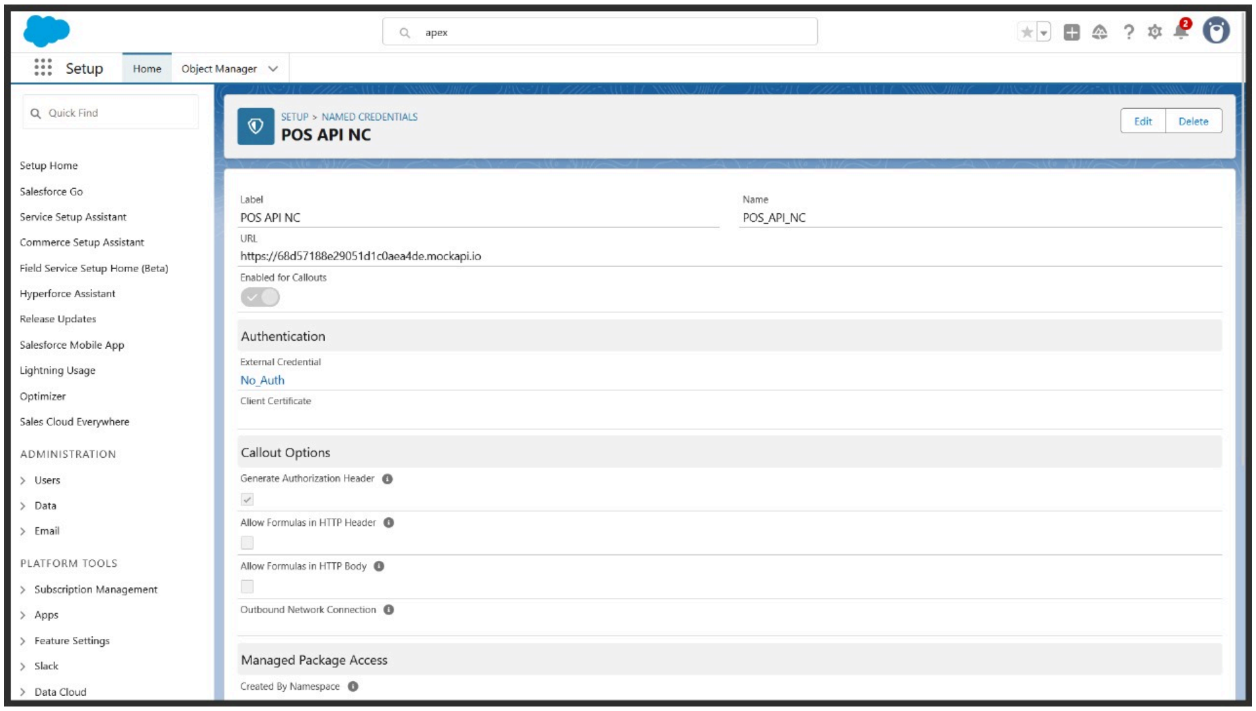
To extend **MACK Stores CRM** beyond its internal ecosystem by integrating with external systems, enabling secure data exchange, and supporting real-time communication. This phase ensures seamless connectivity with ERP, payment gateways, and partner applications using Salesforce integration tools and APIs while maintaining security, scalability, and performa

---

1. Named Credentials

Steps:

- Navigated to Setup → Named Credentials.
- Created `POS_Named_Credential` to securely store authentication details for the POS system.
- Configured with:
  - Authentication: No Auth
  - Identity Type: Named Principal
  - Scope: `full refresh_token`
- Eliminated the need to hardcode usernames/passwords inside Apex classes.



SETUP > NAMED CREDENTIALS

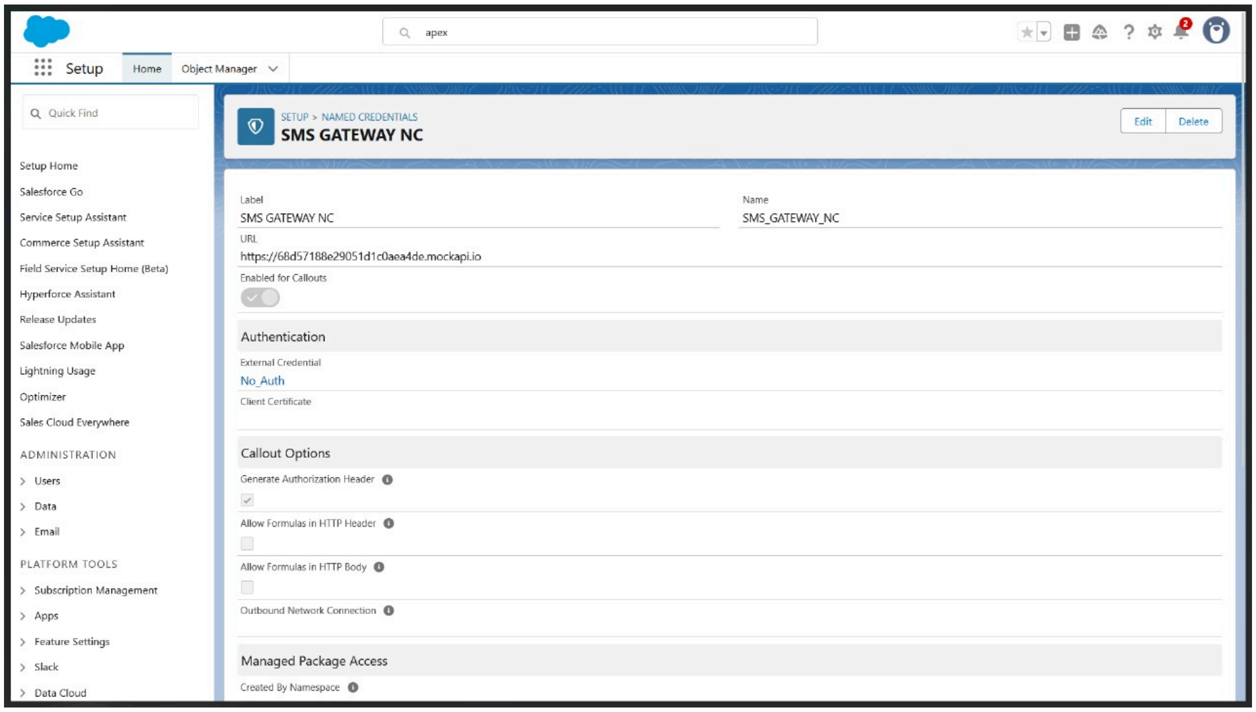
POS API NC

Label: POS API NC
Name: POS_API_NC
URL: https://68d57188e29051d1c0aea4de.mockapi.io
Enabled for Callouts: ✓

Authentication
External Credential: No_Auth
Client Certificate

Callout Options
Generate Authorization Header: ✓
Allow Formulas in HTTP Header: ☐
Allow Formulas in HTTP Body: ☐
Outbound Network Connection

Managed Package Access
Created By Namespace



SETUP > NAMED CREDENTIALS

SMS GATEWAY NC

Label: SMS GATEWAY NC
Name: SMS_GATEWAY_NC
URL: https://68d57188e29051d1c0aea4de.mockapi.io
Enabled for Callouts: ✓

Authentication
External Credential: No_Auth
Client Certificate

Callout Options
Generate Authorization Header: ✓
Allow Formulas in HTTP Header: ☐
Allow Formulas in HTTP Body: ☐
Outbound Network Connection

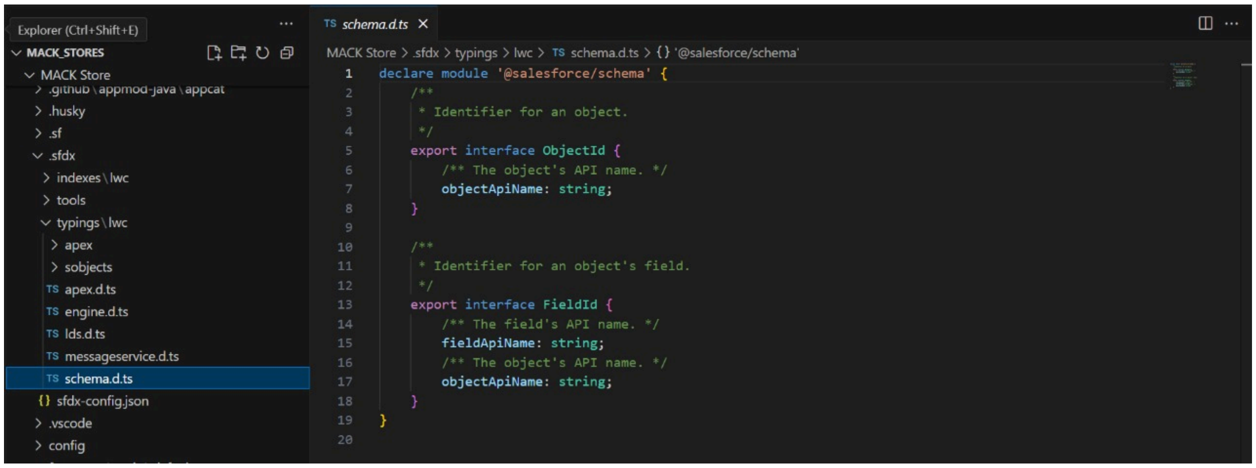
Managed Package Access
Created By Namespace

---

2. External Services

Steps:

- Imported a **Swagger schema** for the POS product pricing service.
- Created a Registered External Service `POSProductService`.
- Auto-generated Apex actions available in Flow and Apex.
- Used the service to fetch live product pricing when creating **Sales Orders**.



```
declare module '@salesforce/schema' {
    /**
     * Identifier for an object.
     */
    export interface ObjectId {
        /** The object's API name. */
        objectApiName: string;
    }

    /**
     * Identifier for an object's field.
     */
    export interface FieldId {
        /** The field's API name. */
        fieldApiName: string;
        /** The object's API name. */
        objectApiName: string;
    }
}
```

---

3. Web Services (REST & SOAP)

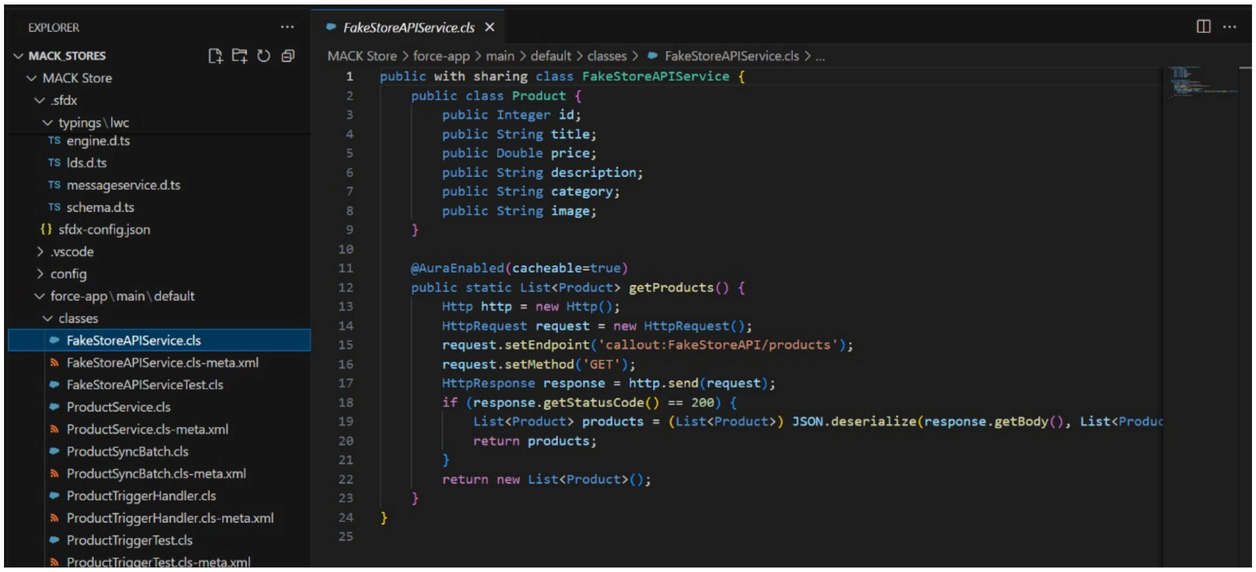
Steps:

- **REST**:
  - Created a custom REST Apex class `OrderAPI` for partner integration.
  - Exposed endpoints for creating new `Sales_Order__c` records.

```
@RestResource(urlMapping='/orders/*')
global with sharing class OrderAPI {
    @HttpPost
    global static Id createOrder(OrderWrapper ow) {
        Sales_Order__c order = new Sales_Order__c(
            Customer__c = ow.customerId,
            Total_Amount__c = ow.amount
        );
        insert order;
        return order.Id;
    }
}
```

- **SOAP**:
  - Consumed a SOAP-based payment gateway WSDL.

- Generated Apex stubs using WSDL2Apex.
- Integrated SOAP methods to validate transactions and update `Payment_Status__c`.


EXPLORER
MACK_STORES
MACK Store
.sfdx
typings\lwc
engine.d.ts
lds.d.ts
messageservice.d.ts
schema.d.ts
sfdx-config.json
.vscode
config
force-app\main\default
classes
FakeStoreAPIService.cls
FakeStoreAPIService.cls-meta.xml
FakeStoreAPIServiceTest.cls
ProductService.cls
ProductService.cls-meta.xml
ProductSyncBatch.cls
ProductSyncBatch.cls-meta.xml
ProductTriggerHandler.cls
ProductTriggerHandler.cls-meta.xml
ProductTriggerTest.cls
ProductTriggerTest.cls-meta.xml
FakeStoreAPIService.cls
MACK Store > force-app > main > default > classes > FakeStoreAPIService.cls > ...

```
public with sharing class FakeStoreAPIService {
    public class Product {
        public Integer id;
        public String title;
        public Double price;
        public String description;
        public String category;
        public String image;
    }

    @AuraEnabled(cacheable=true)
    public static List<Product> getProducts() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('callout:FakeStoreAPI/products');
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        if (response.getStatusCode() == 200) {
            List<Product> products = (List<Product>) JSON.deserialize(response.getBody(), List<Produc
            return products;
        }
        return new List<Product>();
    }
}
```

---

4. Callouts

Steps:

- Created `HttpRequest` callouts to external tax service API.

Example snippet:

```
 HttpRequest req = new HttpRequest();
req.setEndpoint('callout:ERP_Named_Credential/tax');
req.setMethod('GET');
Http http = new Http();
HttpResponse res = http.send(req);
```

- Implemented error handling with retry logic and surface errors via toast messages in LWC.

---

5. Platform Events

Steps:

- Created `OrderEvent__e` platform event for **real-time notifications** to POS.
- Published when a new order is created.
- Subscribed using MuleSoft to process and update external inventory system.
- Benefits: Asynchronous, scalable integration.

---

6. Change Data Capture (CDC)

Steps:

- Enabled CDC for `Sales_Order__c` and `Product__c`.
- Used CDC events to keep ERP in sync with Salesforce updates (status changes, inventory updates).
- Subscribed to CDC events using CometD client and MuleSoft.

---

7. Salesforce Connect

Steps:

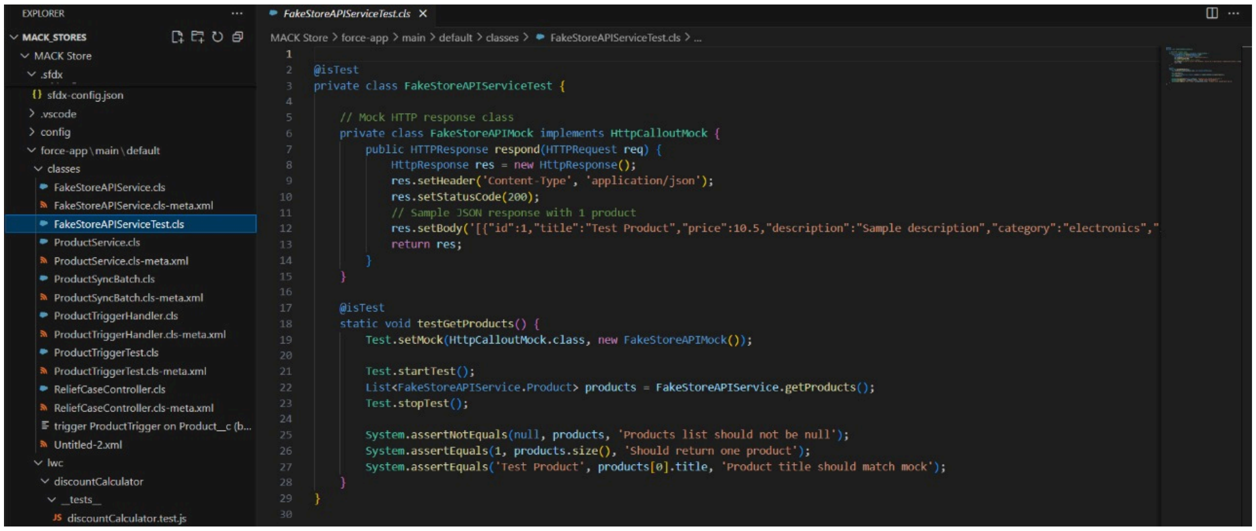
- Configured **Salesforce Connect** with OData 4.0 endpoint to access POS inventory data.
- Created External Object `POS_Inventory__x`.
- Related it with internal `Product__c` object for unified view.
- Users can now view ERP inventory **in real-time** without data duplication.

---

8. API Limits & Best Practices

Steps:

- Monitored daily API usage via System Overview.
- Implemented **bulkified Apex** and **Queueable jobs** for heavy integrations.
- Used **caching** with `@AuraEnabled(cacheable=true)` where possible.
- Scheduled nightly batch for non-urgent bulk sync to reduce API calls.



```
@isTest
private class FakeStoreAPIServiceTest {

    // Mock HTTP response class
    private class FakeStoreAPIMock implements HttpCalloutMock {
        public HTTPResponse respond(HTTPRequest req) {
            HttpResponse res = new HttpResponse();
            res.setHeader('Content-Type', 'application/json');
            res.setStatusCode(200);
            // Sample JSON response with 1 product
            res.setBody('[{"id":1,"title":"Test Product","price":10.5,"description":"Sample description","category":"electronics","
            return res;
        }
    }

    @isTest
    static void testGetProducts() {
        Test.setMock(HttpCalloutMock.class, new FakeStoreAPIMock());

        Test.startTest();
        List<FakeStoreAPIService.Product> products = FakeStoreAPIService.getProducts();
        Test.stopTest();

        System.assertNotEquals(null, products, 'Products list should not be null');
        System.assertEquals(1, products.size(), 'Should return one product');
        System.assertEquals('Test Product', products[0].title, 'Product title should match mock');
    }
}
```

EXPLORER
MACK_STORES
MACK Store
.sfdx
sfdx-config.json
.vscode
config
force-app\main\default
classes
FakeStoreAPIService.cls
FakeStoreAPIService.cls-meta.xml
FakeStoreAPIServiceTest.cls
ProductService.cls
ProductService.cls-meta.xml
ProductSyncBatch.cls
ProductSyncBatch.cls-meta.xml
ProductTriggerHandler.cls
ProductTriggerHandler.cls-meta.xml
ProductTriggerTest.cls
ProductTriggerTest.cls-meta.xml
ReliefCaseController.cls
ReliefCaseController.cls-meta.xml
trigger ProductTrigger on Product__c (b...
Untitled-2.xml
lwc
discountCalculator
__tests__
discountCalculator.test.js
FakeStoreAPIServiceTest.cls
MACK Store > force-app > main > default > classes > FakeStoreAPIServiceTest.cls > ...

---

9. OAuth & Authentication

Steps:

- Configured **OAuth Connected App** for partner portal.
- Enabled:
  - OAuth scopes: `api`, `refresh_token`
  - Callback URL: Partner app's endpoint
- Partners authenticate securely using **OAuth tokens** instead of credentials.
- Used **JWT-based flow** for server-to-server communication with ERP.

---

10. Remote Site Settings

Steps:

- Added Remote Site Settings for:
  - Payment Gateway API
  - Tax API
  - External POS test environment
- Ensured secure HTTPS endpoints.
- Deprecated old remote site entries replaced by Named Credentials.

Outcome of Phase 7

- MACK Stores CRM is now **fully integrated** with POS, payment, tax, and partner systems.
- Real-time order sync through **Platform Events & CDC**.
- Secure authentication handled by **Named Credentials & OAuth**.
- Salesforce Connect enables POS data visibility without storage overhead.
- Users and external apps interact with Salesforce via **REST, SOAP, and External Services** in a controlled, scalable, and secure manner

---