N26

Case Study - Salesforce Engineer

By Ignacio Cobián

Table of Contents

1. Introduction

This document provides insight on the solution of the Case Study for the Salesforce Engineer position. The given case turns around displaying the product information related to a contact based on their Home Country and plan type selected. This information also needs to be displayed in an external system via an API.

2. Business Problem

We're new to Lightning, and starting to make the most of component's flexibility to show our customer service agents the most important information as, and when, they need it. This might be displaying the customer's other open issues as they're talking to them. For this task, we'd like you to display the customer's product information to the agent whilst they're interacting.

N26 also uses an external system, where the customer data is shown from Salesforce. The team would need a mechanism to connect to Salesforce and get the relevant information. The external system passes a unique identifier (UUID) that is mapped to every contact record in Salesforce. Design an API that would be able to return the data for a contact based on the tables provided.

3. Solution Design

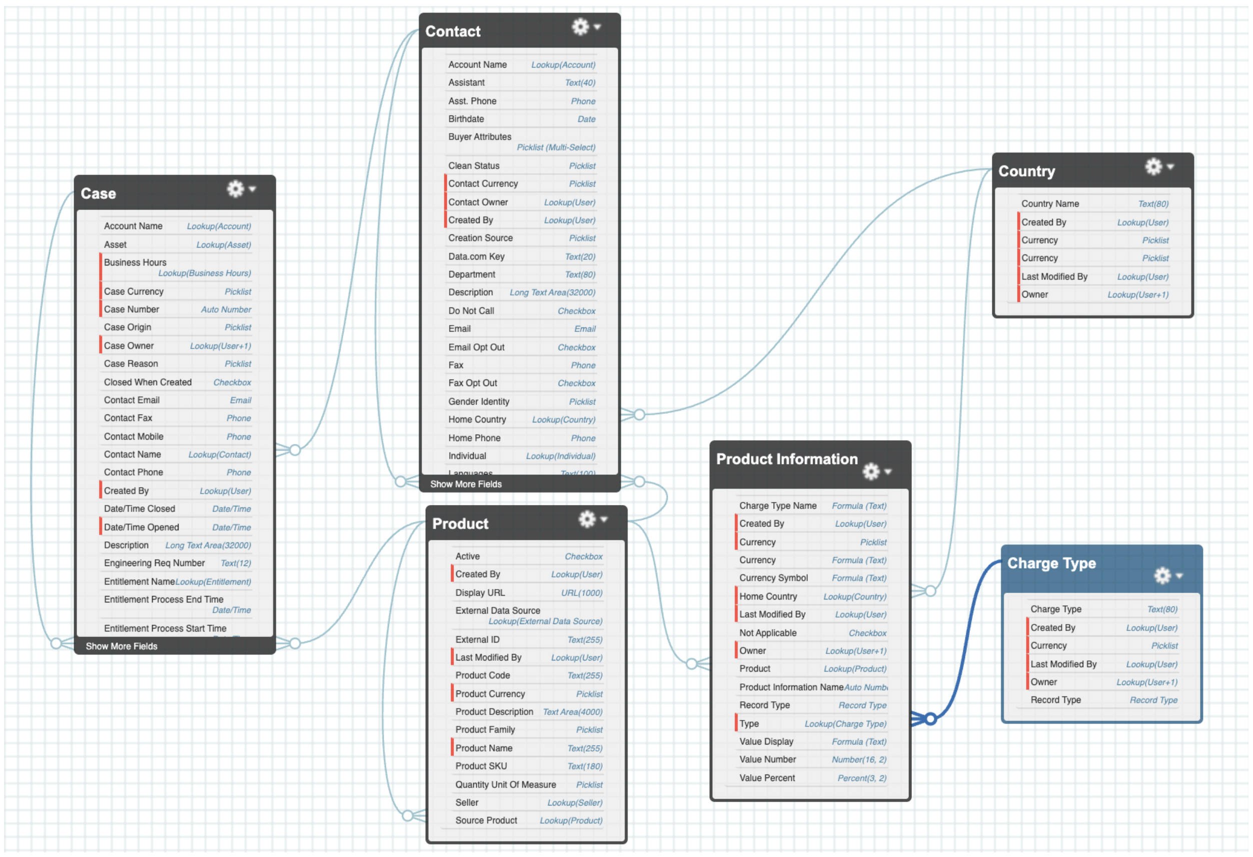
Our main goal with this solution has been to provide the owners of the cost / fee tables full independence to create, edit, add new plan types, countries, products, without needing any deployment or System administrator involved. Based on the principle of, whoever has to do the change, owns the table. Thats why for the solution we have opted to use 3 standard objects:

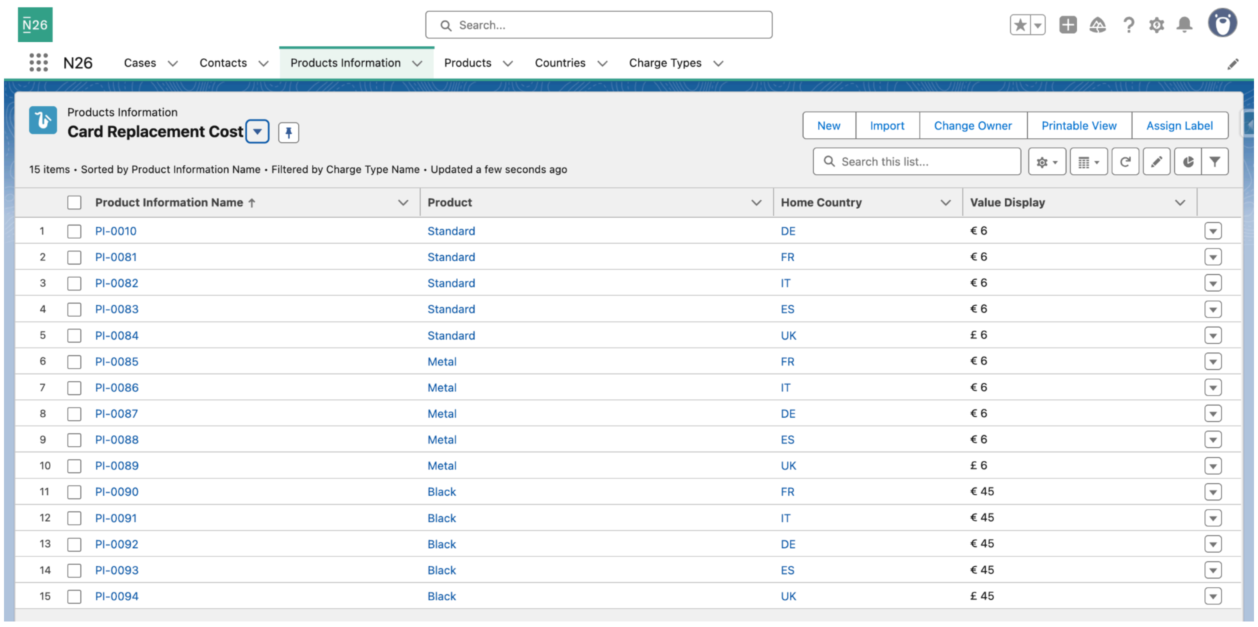
- Contact, edited two include three new custom fields (Home Country, Plan Type and UUID).
- Case, edited to include a new LWC in the layout.
- Product, non edited but used to add the plan type.

And three new custom objects:

- Country, it displays the country code and the currency of the country, could be scalable adding new fields to be used somewhere else.
- Charge type, used to specify the type of charge, in the case study we have Cost per Calendar Month, ATM Fee in Other Currencies and Card Replacement Cost.
- Product Information, a junction object of Product, Country and Charge type that can be easily edited by the charge table owners to edit or include new records.

The data model of the solution looks like:





Products Information
Card Replacement Cost

15 items • Sorted by Product Information Name • Filtered by Charge Type Name • Updated a few seconds ago

	Product Information Name ↑	Product	Home Country	Value Display
1	PI-0010	Standard	DE	€ 6
2	PI-0081	Standard	FR	€ 6
3	PI-0082	Standard	IT	€ 6
4	PI-0083	Standard	ES	€ 6
5	PI-0084	Standard	UK	£ 6
6	PI-0085	Metal	FR	€ 6
7	PI-0086	Metal	IT	€ 6
8	PI-0087	Metal	DE	€ 6
9	PI-0088	Metal	ES	€ 6
10	PI-0089	Metal	UK	£ 6
11	PI-0090	Black	FR	€ 45
12	PI-0091	Black	IT	€ 45
13	PI-0092	Black	DE	€ 45
14	PI-0093	Black	ES	€ 45
15	PI-0094	Black	UK	£ 45

For the first part of the solution we have opted to use a LWC to be included in the Case Lightning Page, it uses an ApexClass as a controller and in addition we have added an ApexTrigger with its handler to avoid creating Product Information duplicates. For the second part we have used an ApexClass to be consumed in the API.

Regarding security, we have created the following custom profiles:

- Customer Service, it can edit both the contacts and the cases as if customer services were allowed to and just has visibility (cannot edit, create or delete) to product, product information, country and charge type. Has access to the LWC controller apex class and the trigger handler.
- N26 API User, cant access any object but has the API enabled to be able to consume APIs.

In addition to those profiles we have included two permission sets:

- Edit - Product Information, has edit and delete access to the required objects to display a contact product information, can edit and create product information, country, product and charge type.
- API - Get Contact Information, has access to the API apex class and the required objects to display the information.

It is important to mention that any new type of charge, different from Cost and Fee, will need a deployment from the System Administrators (no need to edit code), as product information and charge type both objects have a Cost record type and a Fee record type that need to match. Also we have opted to create a global picklist value set for the currency as we are not aware of any integration in N26 that updates the SF standard currency rates. So any new currency needs to be added in the picklist global value set.

4. Implementation

4.1. Apex Classes

4.1.1 APIRestGetContactInfo

This class is used as the N26 external system would need a mechanism to connect to Salesforce and get the relevant information. The external system passes a unique identifier (UUID) that is mapped to every contact record in Salesforce. The REST API endpoint is (/api/n26/*). It uses the following method:

4.1.1.1 getContactInfo()

Based on the UUID, it validates that only the right username consumes the resource. Links the contact with the UUID and at last returns the relevant data of the contact.

4.1.2 DisplayContactProdInfCaseController

This is the controller class of the DisplayContactProdInfCase LWC. It returns the necessary data to be displayed in the LWC. It contains the following method:

4.1.2.1 getContactProdInfDetails (Id caseId)

Given a CaseId it fetches for the related contact to provide the product information that is related to that contact based on its plan type and country. It uses a wrapper class to return this information.

4.1.3 ProductInformationTriggerHandler

This is the Apex Class Handler of the trigger ProductInformationTrigger on Product_Information__c. It is used to ensure no duplicated records are created. It uses the following method:

4.1.3.1 handleBeforeInsert(List<Product_Information__c> newProdInf)

Receiving the list before insert of Product_Information__c ensures that there are no existing records with the same criteria, it also ensures that in the same insertion no duplicates are added.

4.2. Apex Triggers

4.2.1 ProductInformationTrigger

Trigger used before insert for the object Product_Information__c.

4.3. Lightning Web Components

4.3.1 DisplayContactProdInfCase

LWC used to display the relevant information obtained with a dynamic table in the DisplayContactProdInfCaseController apex class. In case of not receiving information it displays the error and doesn't display the table.

4.4. Validation Rules

4.4.1 Product Information - Check_Charge_Type

The charge type record type selected must be the same type as the record type of the product information.

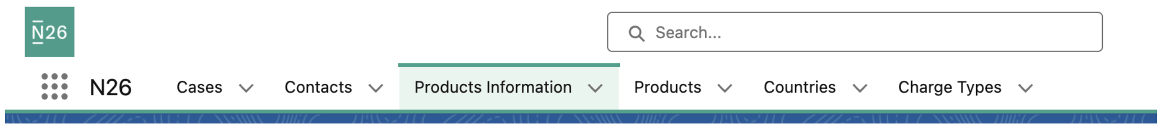
4.4.2 Product Information - Ensure_Mandatory_Fields_Selected

You must ensure the following fields are added: Home Country, Product and Type

4.5. Salesforce Apps

4.5.1 N26

Lightning App, used to display the relevant objects and tabs for the solution.

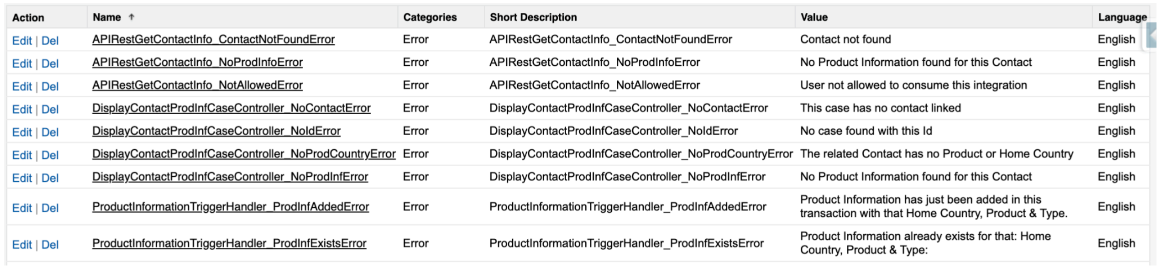


4.5.2 N26 API Integration

Connected App to allow the external system to connect to Salesforce and consume the API.

5. Error Handling and Validation

Custom error messages are returned when data is missing or invalid. Errors are displayed in the user interface or returned in the body of the response API. We have included the following custom labels to display the necessary errors, in the future we can include translations to these error labels.



Action	Name ↑	Categories	Short Description	Value	Language
Edit | Del	APIRestGetContactInfo_ContactNotFoundError	Error	APIRestGetContactInfo_ContactNotFoundError	Contact not found	English
Edit | Del	APIRestGetContactInfo_NoProdInfoError	Error	APIRestGetContactInfo_NoProdInfoError	No Product Information found for this Contact	English
Edit | Del	APIRestGetContactInfo_NotAllowedError	Error	APIRestGetContactInfo_NotAllowedError	User not allowed to consume this integration	English
Edit | Del	DisplayContactProdInfCaseController_NoContactError	Error	DisplayContactProdInfCaseController_NoContactError	This case has no contact linked	English
Edit | Del	DisplayContactProdInfCaseController_NoIdError	Error	DisplayContactProdInfCaseController_NoIdError	No case found with this Id	English
Edit | Del	DisplayContactProdInfCaseController_NoProdCountryError	Error	DisplayContactProdInfCaseController_NoProdCountryError	The related Contact has no Product or Home Country	English
Edit | Del	DisplayContactProdInfCaseController_NoProdInfError	Error	DisplayContactProdInfCaseController_NoProdInfError	No Product Information found for this Contact	English
Edit | Del	ProductInformationTriggerHandler_ProdInfAddedError	Error	ProductInformationTriggerHandler_ProdInfAddedError	Product Information has just been added in this transaction with that Home Country, Product & Type.	English
Edit | Del	ProductInformationTriggerHandler_ProdInfExistsError	Error	ProductInformationTriggerHandler_ProdInfExistsError	Product Information already exists for that: Home Country, Product & Type:	English

6. Security Considerations

The REST API accepts requests only from the predefined integration user. All data access follows Salesforce sharing and field-level security rules.

7. Testing

7.1. Functional testing

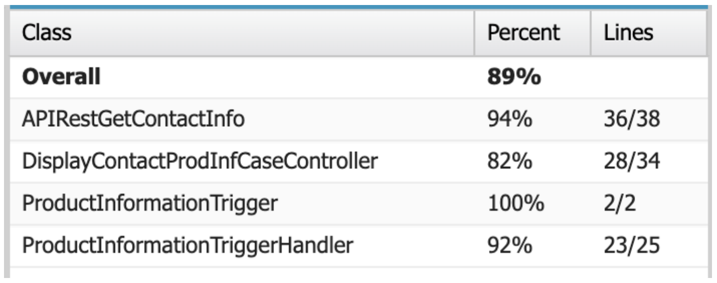
We have ensured with functional testing both in the layout and postman that both the happy path and the error path are met and displayed the necessary errors or success messages.

7.1. Technical testing

We have created the following apex Class tests:

- APIRestGetContactInfoTest, test class for APIRestGetContactInfo.
- DisplayContactProdInfCaseControllerTest, test class for DisplayContactProdInfCaseController.
- ProductInformationTriggerHandlerTest, test class for ProductInformationTriggerHandler.

Obtaining always a +75% test cover in every class:



Class	Percent	Lines
Overall	**89%**	
APIRestGetContactInfo	94%	36/38
DisplayContactProdInfCaseController	82%	28/34
ProductInformationTrigger	100%	2/2
ProductInformationTriggerHandler	92%	23/25

8. Conclusion

With this solution we provide as much freedom as the table owners will need to implement their own changes and additions to the relevant objects of the solution. Minimizing with error handling and avoiding for them to create duplicates in the most relevant table of the solution. Any changes in the solution given the robust built and dynamic code can be held entirely by system administrators, just needing to change a global picklist value set or adding new record types. The solution is dynamic and scalable and easily editable.

Future improvements can include an Integration with the Salesforce Org to be able to update daily the currency ratios to be able to use SF currency standard field. Also new translations to the error labels so different located customer service people can understand the error messages without them knowing english. Also future meetings with the stakeholders to improve the functionality, like if they want to be able to edit the records in the listview.