**Task 1 - JavaScript Customization Task**

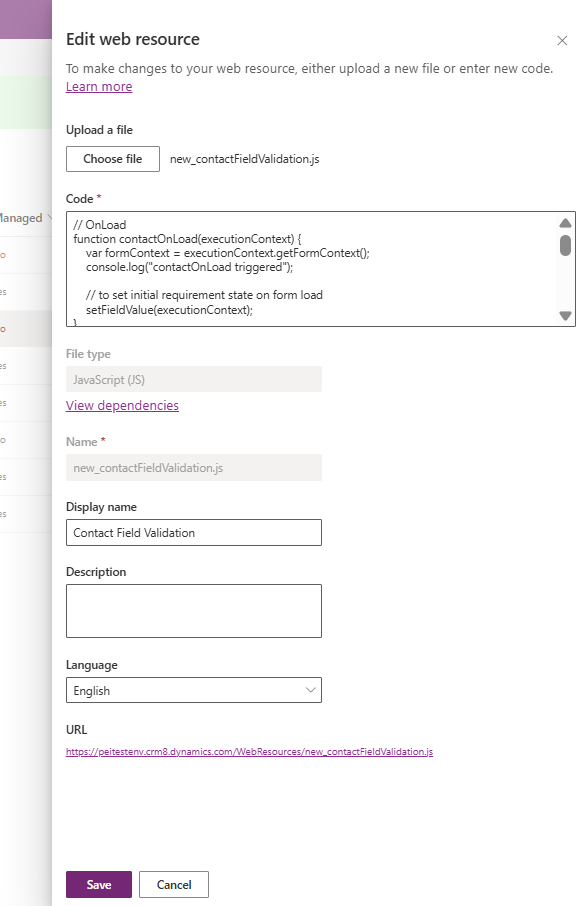
**Scenario 1**:

* Implement a client-side customization for the **Contact** entity form:
  + **Functionality**:
    - Make the "Email Address" field mandatory if the "Preferred Method of Contact" is set to "Email."
    - Make the "Phone Number" field mandatory if the "Preferred Method of Contact" is set to "Phone."
    - Ensure at least one of these fields is filled before saving the form.
  + Use the Dynamics 365 JavaScript API to achieve this.
  + Deploy and test the script on the Contact form.

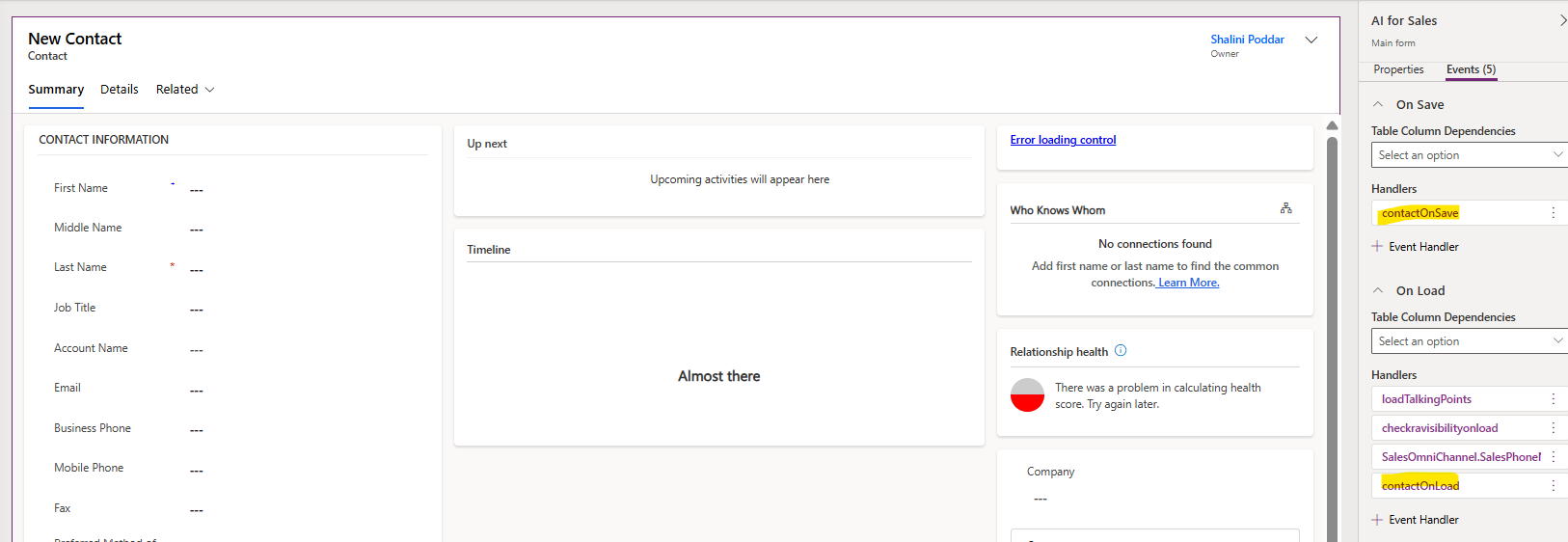
**JS Code**

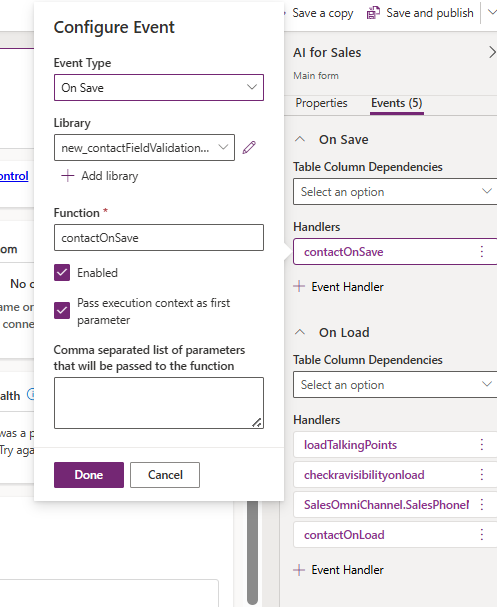
****

Added JS file new\_contactFieldValidation

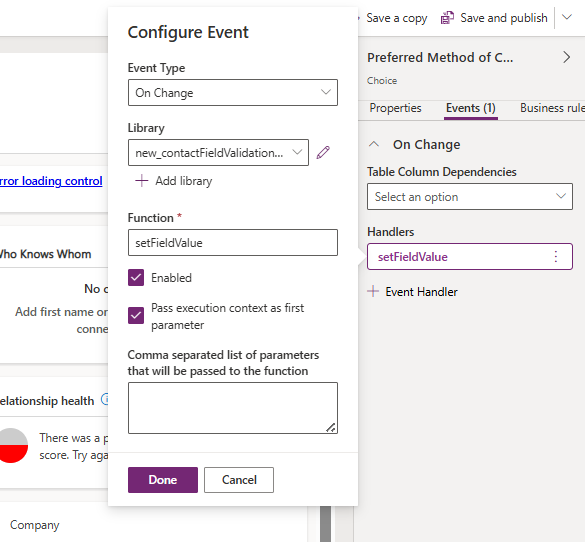
****

Added the following Events on Form properties

****

****

And On Change event on the field

****

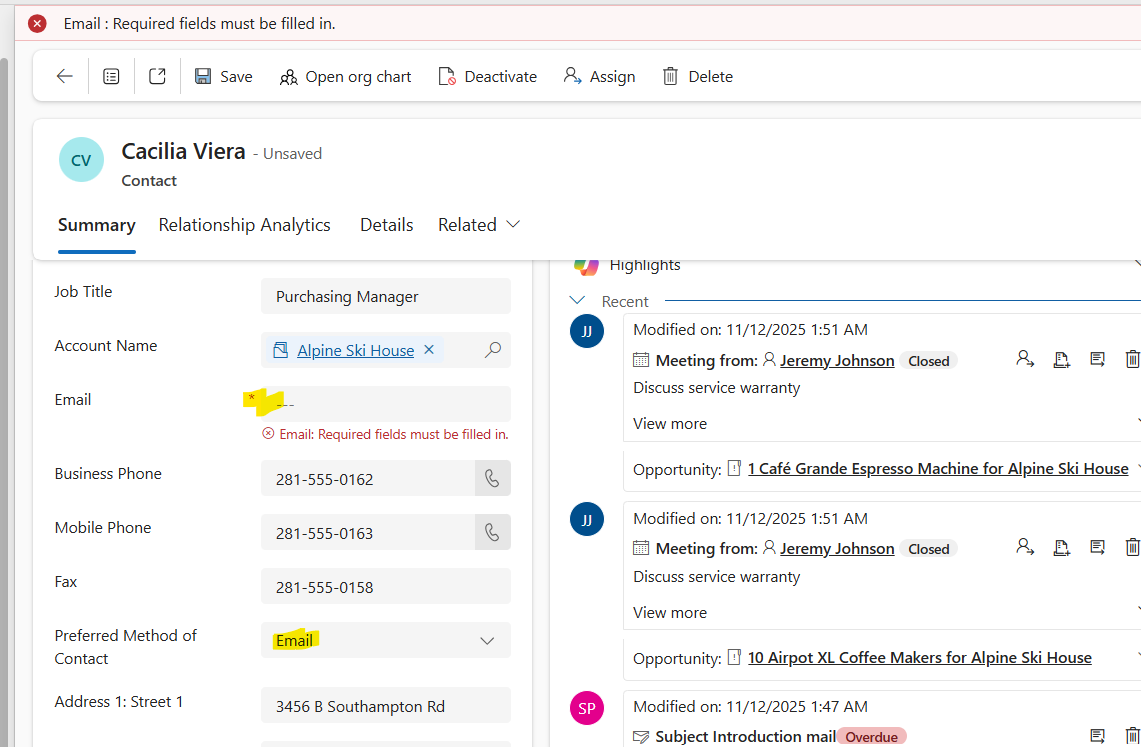
Added the events on Main Form of Contact and Published the changes.

**Working**

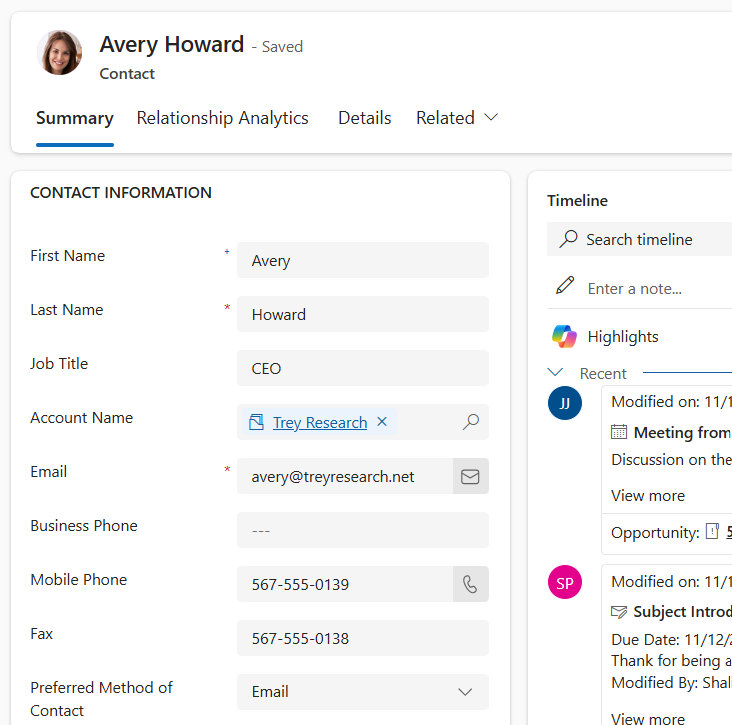
**TEST CASES:**

1. WHEN Preferred Method of Contact is set to Email  
   AND Email is empty  
   AND Phone may or may not have a value  
   THEN the system must prevent saving  
   AND show the alert:  
   “Email : Required fields must be filled in.”

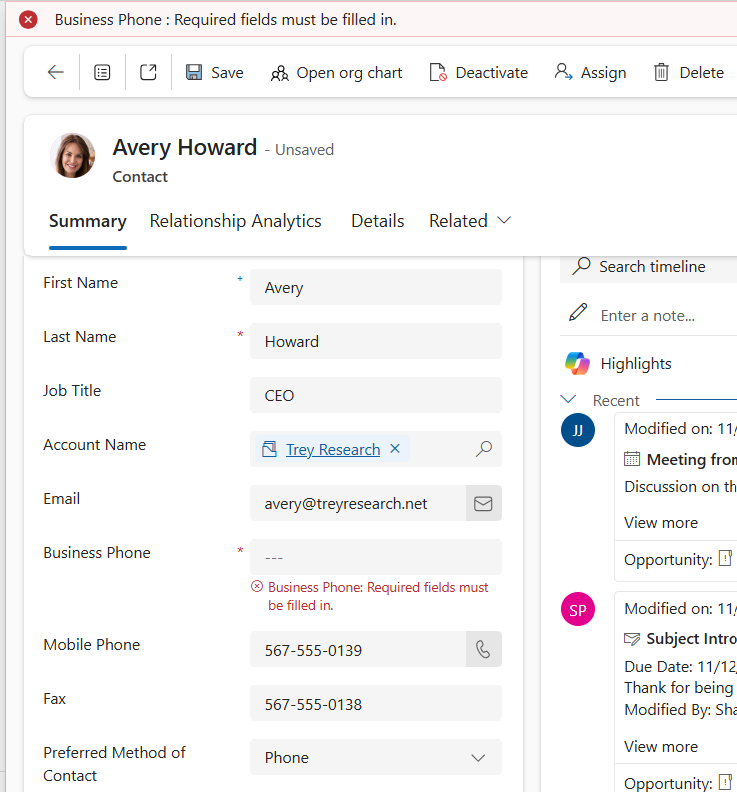
When Email is selected as Preferred Method of Contact, then it becomes a mandatory field and to save the record that value needs to be passed.



1. WHEN Preferred Method is Email  
   AND Email has a valid value  
   THEN the system must allow saving.

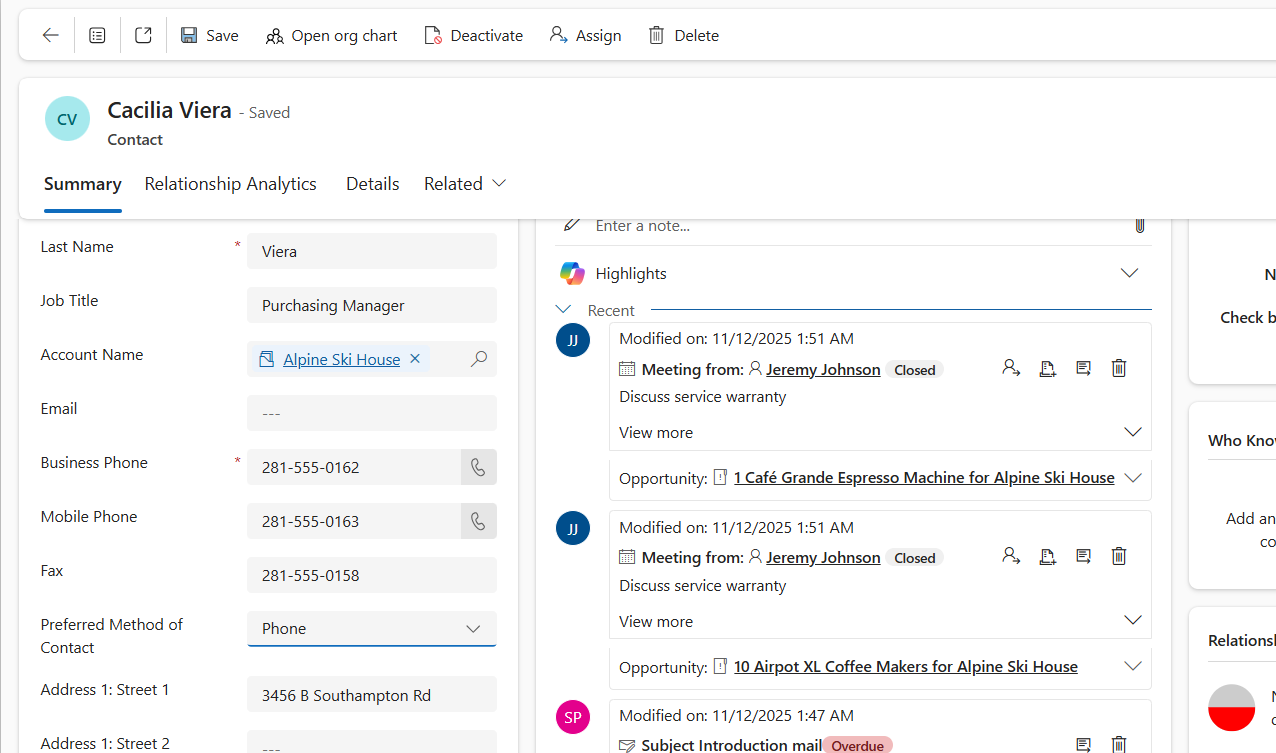


1. WHEN Preferred Method is Phone  
   AND Phone is empty  
   THEN the system must prevent saving  
   AND show:  
   “Business Phone : Required fields must be filled in.”

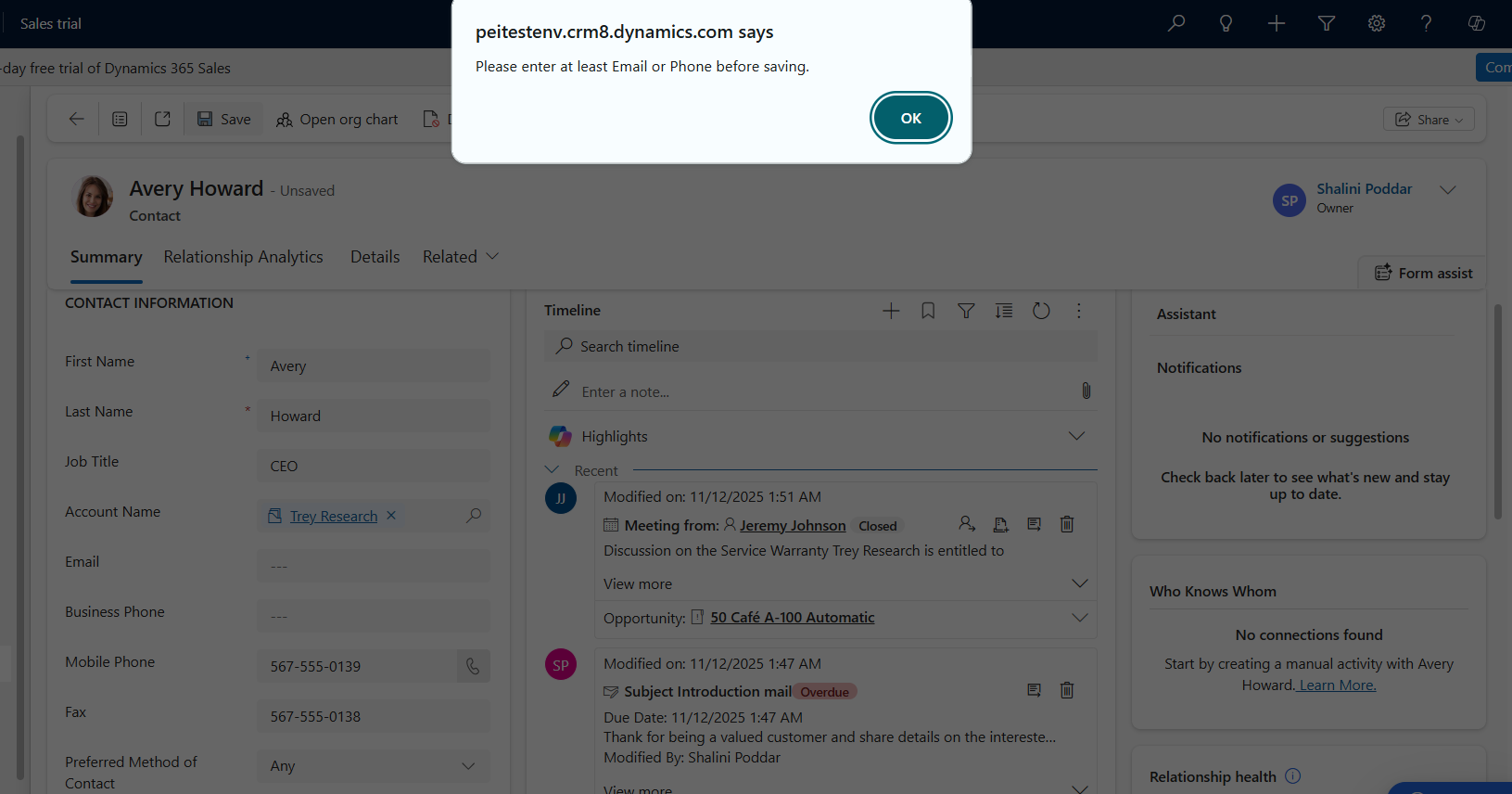


1. WHEN Preferred Method is Phone  
   AND Phone has a valid value  
   THEN the system must allow saving.

If Phone is selected, then that field becomes mandatory



1. WHEN Email is empty  
   AND Phone is empty  
   THEN the system must prevent saving  
   AND show:  
   “Please enter at least Email or Phone before saving.”

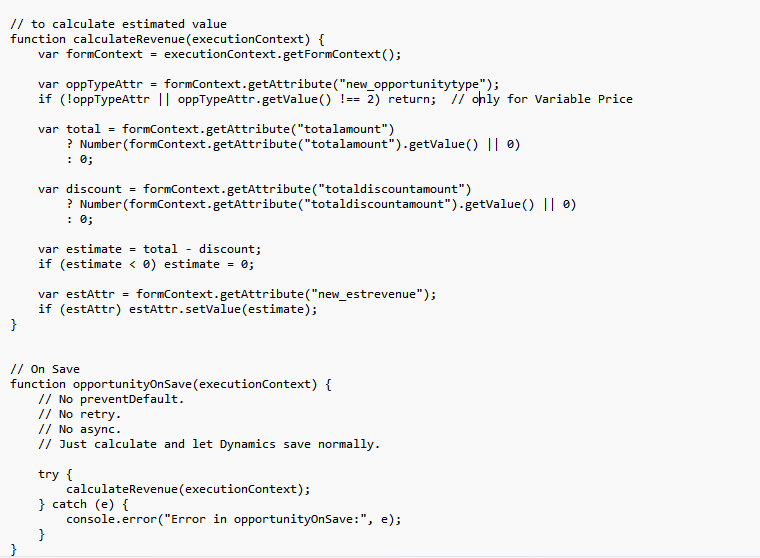
To ensure at least one of these fields is filled before saving the form

**Scenario 2**:

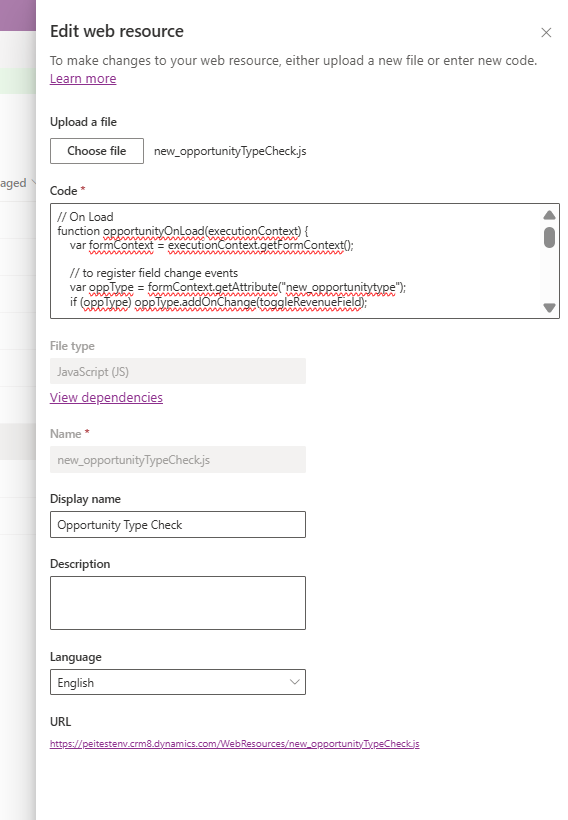
* Implement client-side customization for the **Opportunity** entity form:
  + **Functionality**:
    - Disable the "Estimated Revenue" field if the "Opportunity Type" is set to "Fixed Price."
    - Automatically calculate and populate "Estimated Revenue" if the "Opportunity Type" is set to "Variable Price" using the formula:
      * Estimated Revenue = (Total Units \* Unit Price) - Discount.
  + Use JavaScript to implement the logic and deploy it on the Opportunity form.

**JS Code**

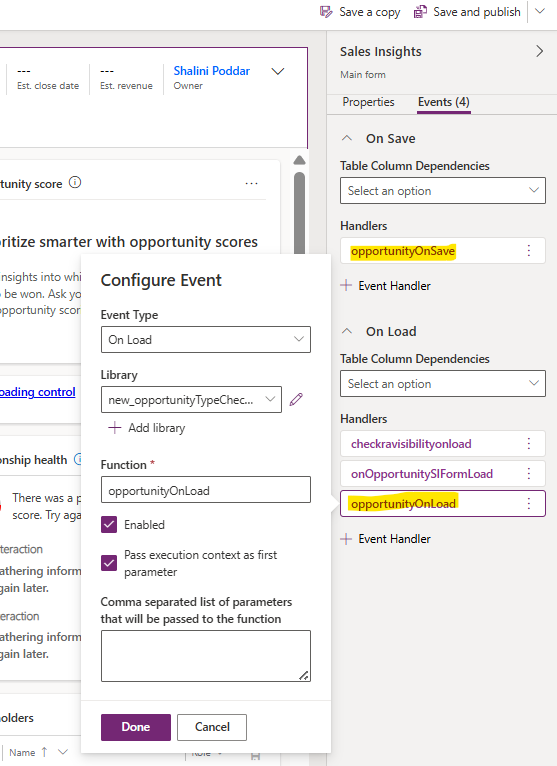
****

****

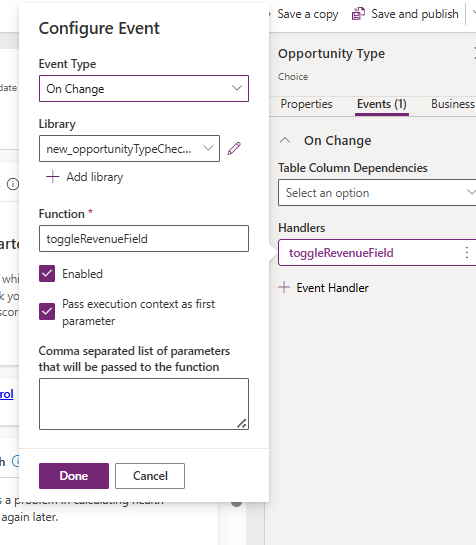
Added JS file new\_opportunityTypeCheck



Added the following Events on Form properties

****

And On Change event on the field

****

Added the events on Main Form of Opportunity and Published the changes.

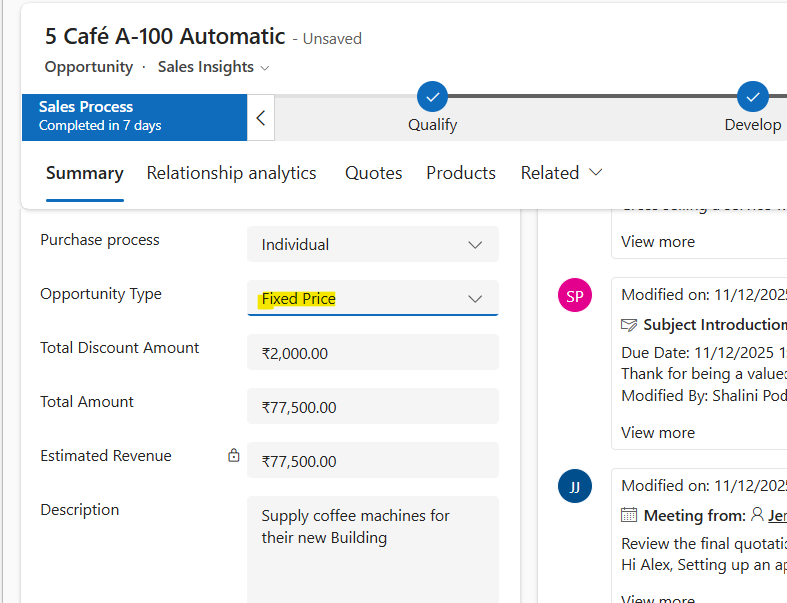
**Working**

**TEST CASES**

WHEN Opportunity Type = Fixed Price

With totalamount AND totaldiscountamount values

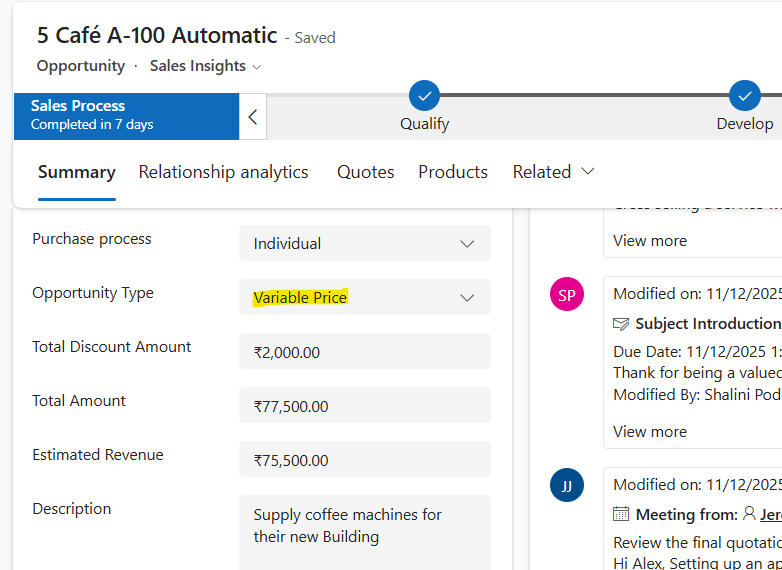
THEN Estimated Revenue should be set to equal to totalamount and the Estimated Revenue field should be disabled (read-only).

****

Opportunity Type = Variable Price

With totalamount AND totaldiscountamount values

THEN Estimated Revenue should be set to equal to (totalamount - totaldiscountamount) and the Estimated Revenue field should be enabled

****

**The issues with both the approaches**

For JS file to work properly, it’s required that correct logical names are used in the JS code.

The code file needs to be added in Form library

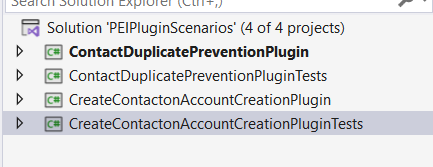
The form events need to be registered with correct function names

Hard coded values may not work in all scenarios

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**2. Plugin Development Task**

Solution, along with all Projects

****

**Scenario 1**:

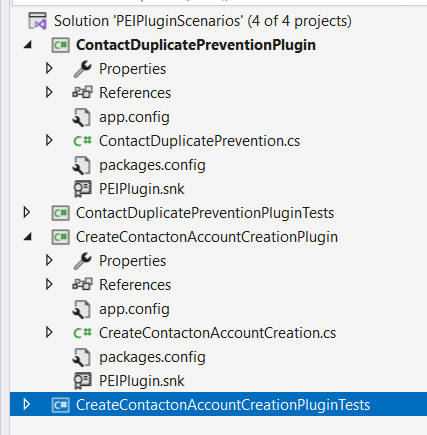
* Develop a plugin for the **Contact** entity:
  + Prevent the creation of duplicate Contacts with the same email address.
  + Display an error message: *"A contact with this email address already exists."*
  + Ensure the plugin is efficient and scalable for large datasets.

1. **Plugin Code**:

I have installedNuGet package -Microsoft.CrmSdk.CoreAssemblies

****

Then I have built the solution and then signed the assembly with strong key

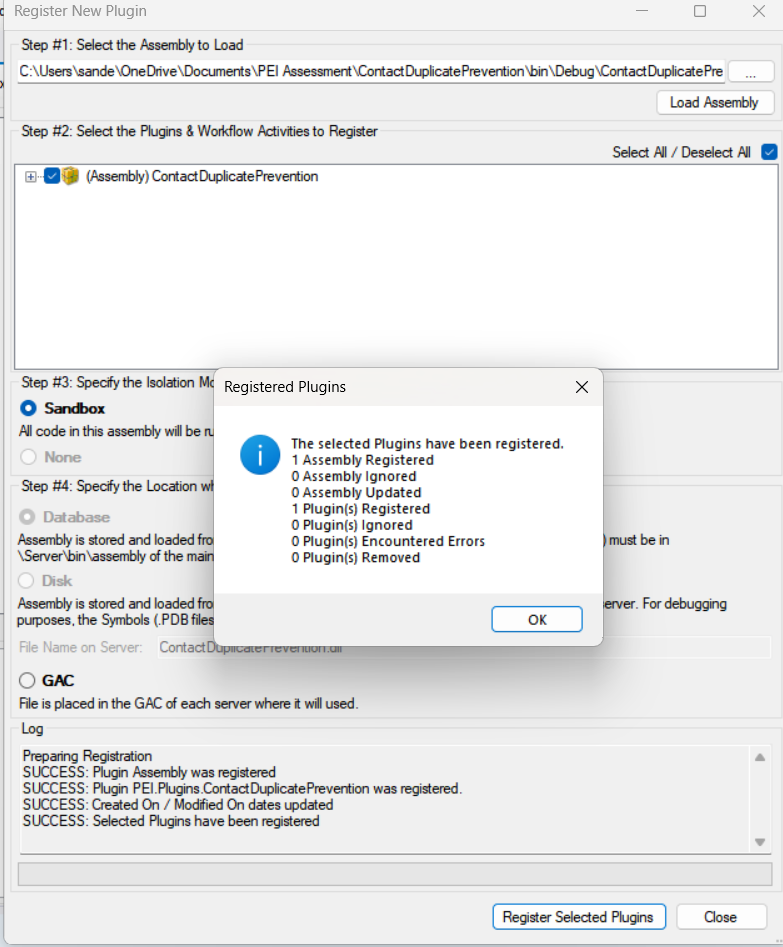


1. **Steps to Register Plugins**:

In Plugin Registration Tool,

Register -> Register Assembly

Then clicked on Load Assembly and selected the .dll file of the project and registered the assembly

****

After registering the plugin, next step is to register the step

Below are the details of the step -

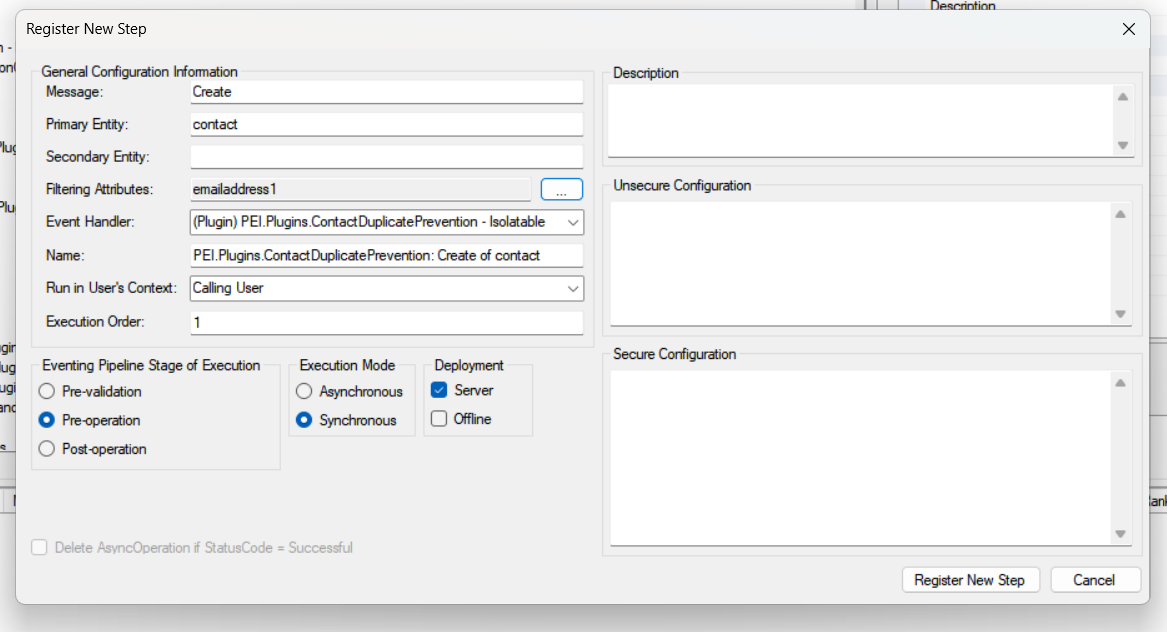
Message – Create

Entity – contact

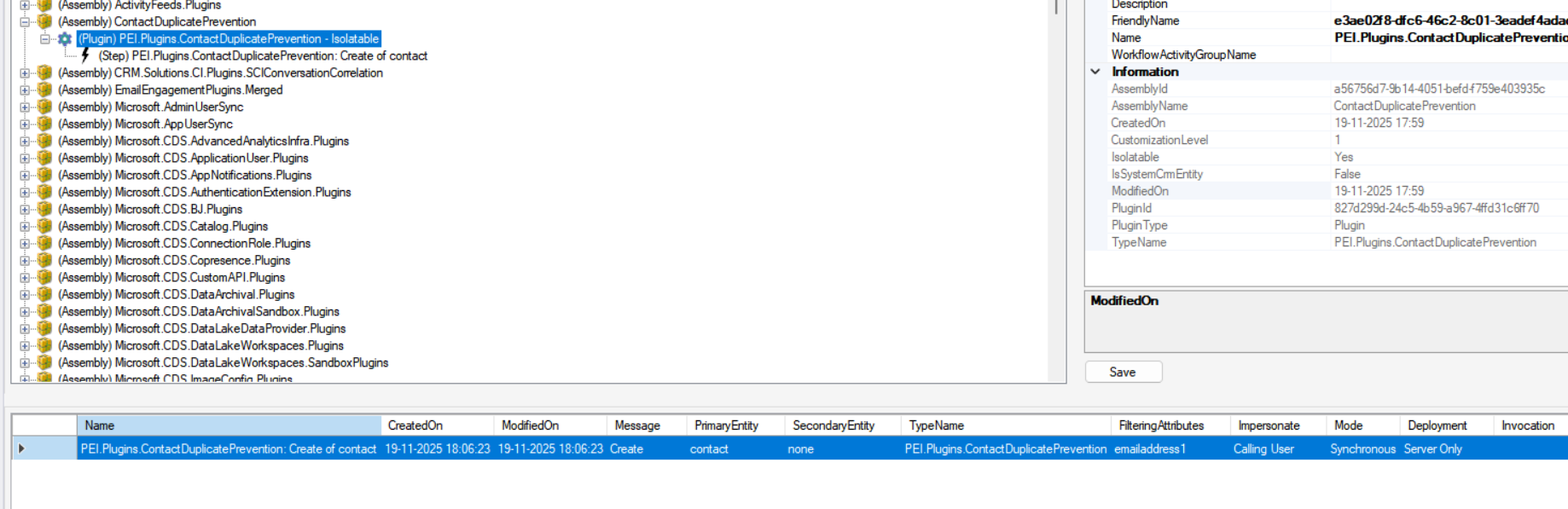
Stage – PreOperation

Mode – Synchronous

Filtering attributes – emailaddress1

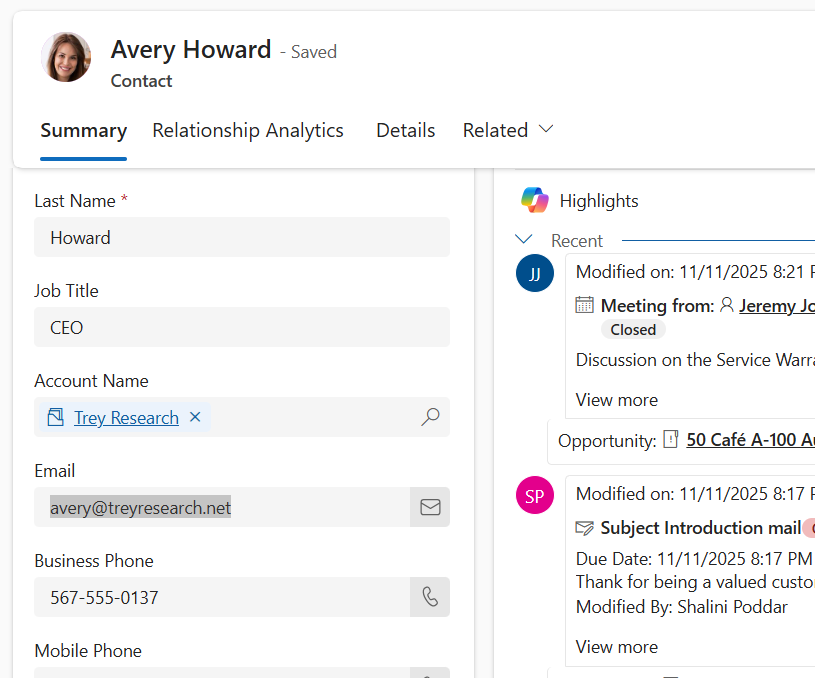
****

Now, it can be seen that step has been registered successfully

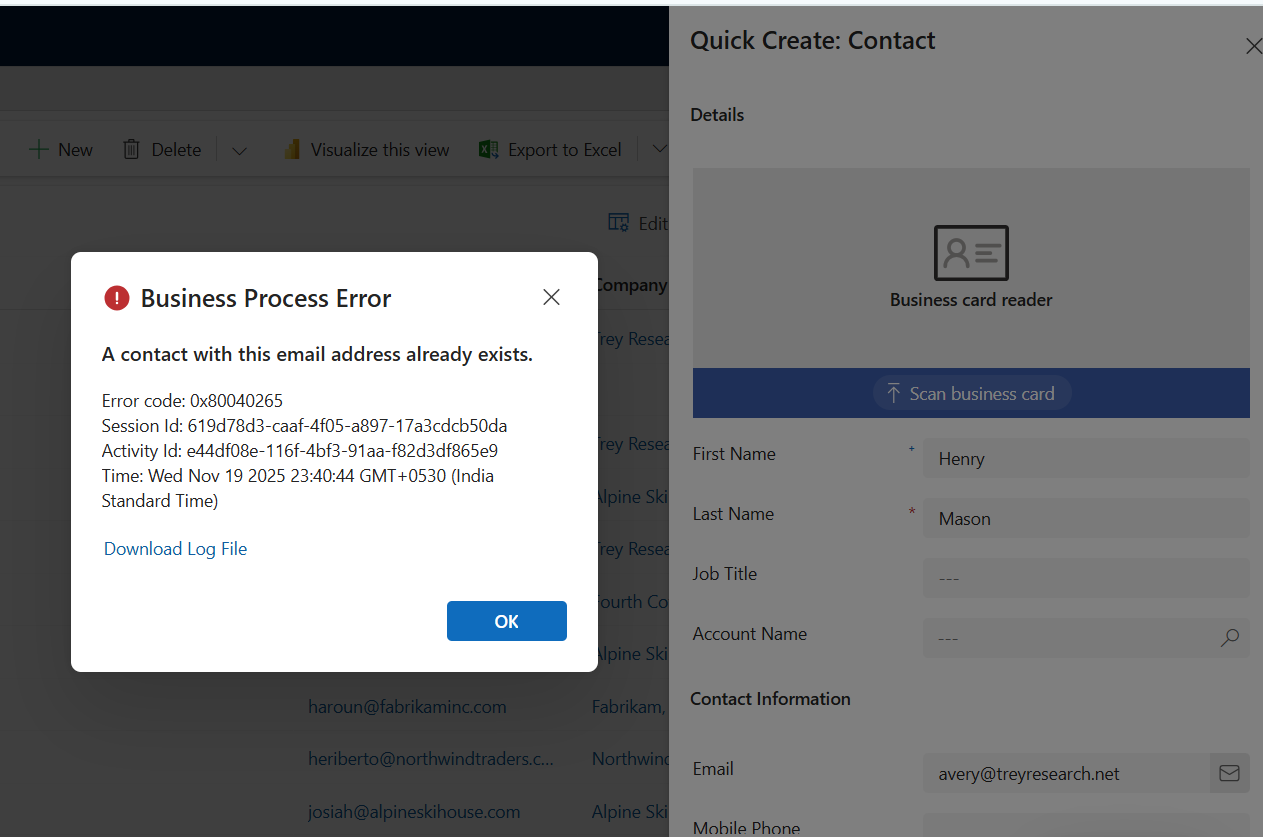
****

**Working**

From the below Contact record, I have taken the email address and copied it to create a new Contact record

****

But when I tried to save the new record, it gave me error that ‘A contact with this email address already exists.’ as per mentioned in plugin code

****

1. **Skeleton CI/CD Pipelines:**

To deploy the code through CI/CD pipeline, we need to first clone a repository and then create a feature branch for the fresh changes.

When code is ready, we have to commit and push the code on the feature branch and then create a pull request to merge the feature branch with the master branch.

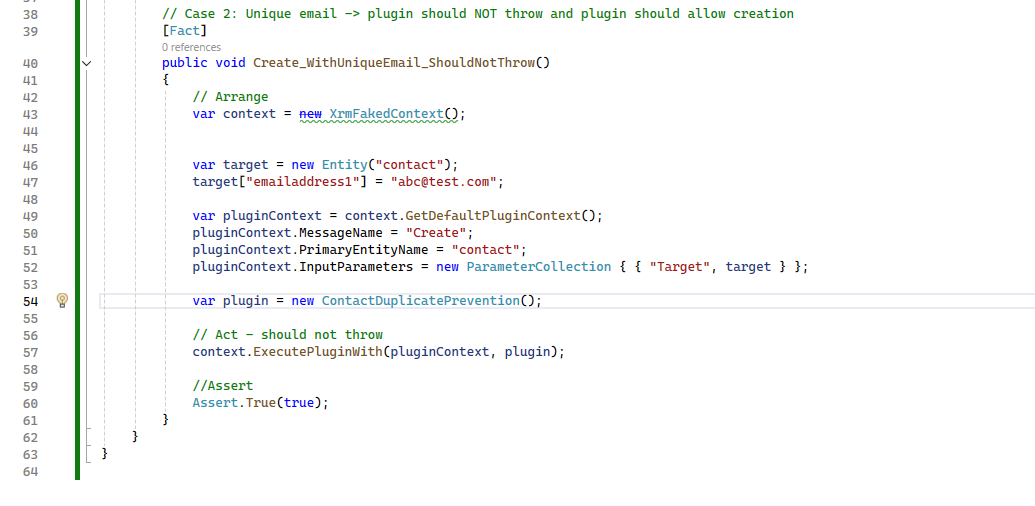
When code is merged, Build Pipeline (CI) is triggered which creates Deployment Artefact and will contain the changes.

Through Release Pipeline (CD), we can select the deployment artefact and deploy it to next environment.

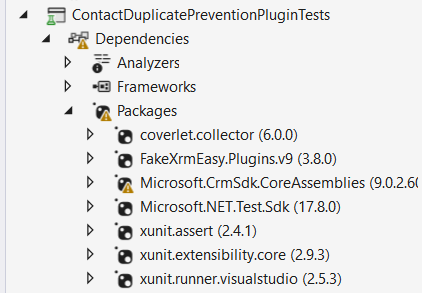
1. **Unit test cases**

Created a test project (xUnit)

****

****

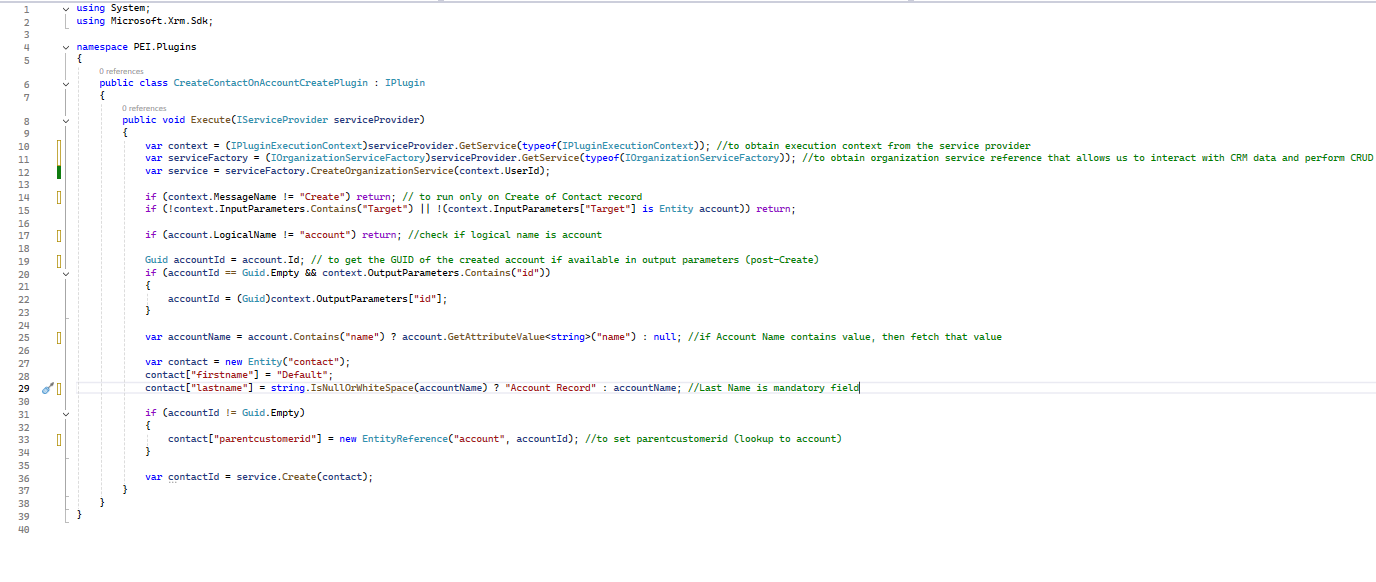
**NuGet packages**

****

**Scenario 2**:

1. Develop a plugin for the **Account** entity:
   1. On Account creation, automatically create a child Contact with:
      1. First Name: *Default*.
      2. Last Name: *Account Name*.
   2. Log a message in the plugin trace log after successful execution.
2. **Plugin Code:**

I have installedMicrosoft.CrmSdk.CoreAssemblies.

****

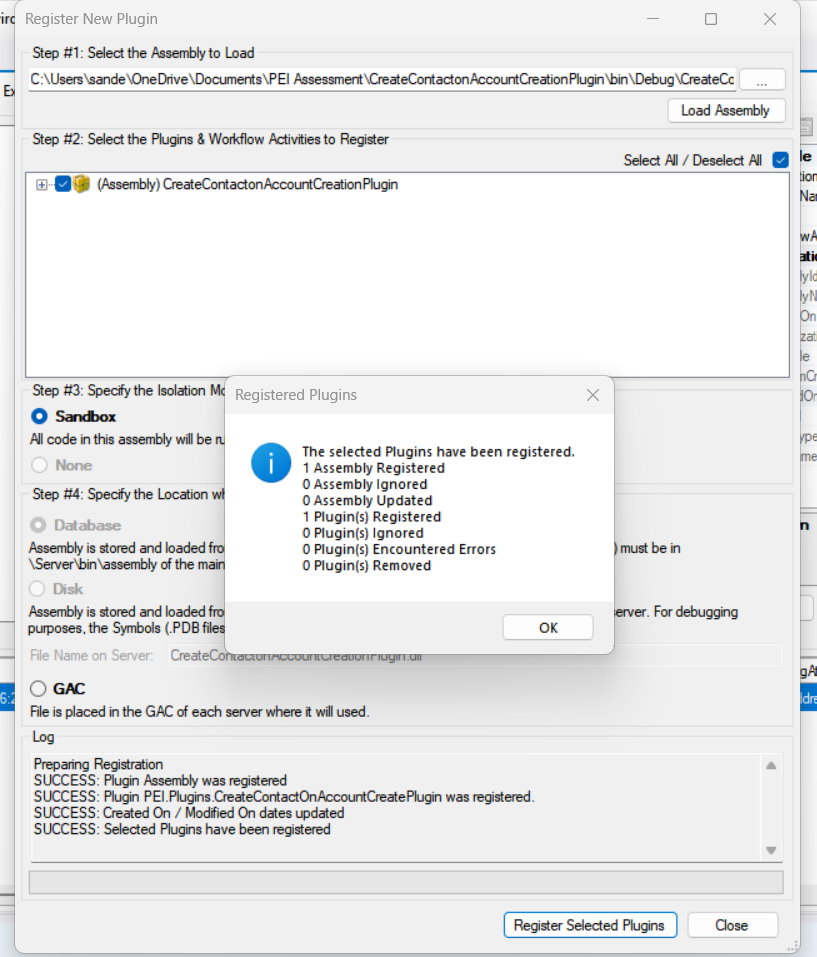
Then I have built the solution and then signed the assembly with strong key

1. **Steps to Register Plugin**:

In Plugin Registration Tool,

Register -> Register Assembly

Then clicked on Load Assembly and selected the .dll file of the project and registered the assembly

****

After registering the plugin, next step is to register the step

Below are the details of the step -

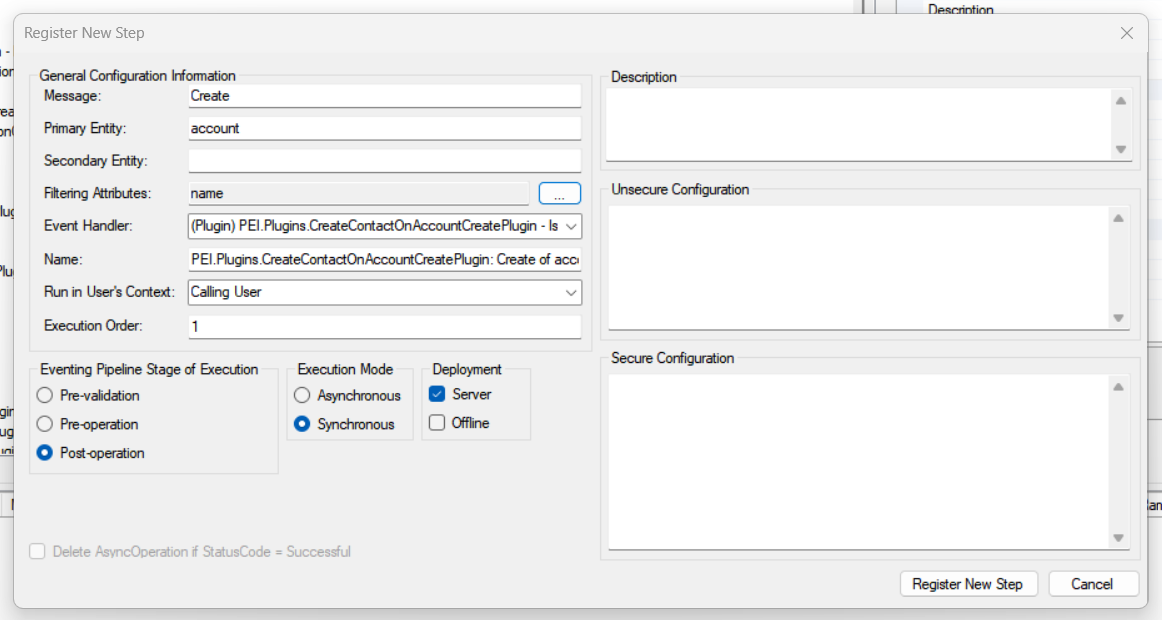
Message – Create

Entity – account

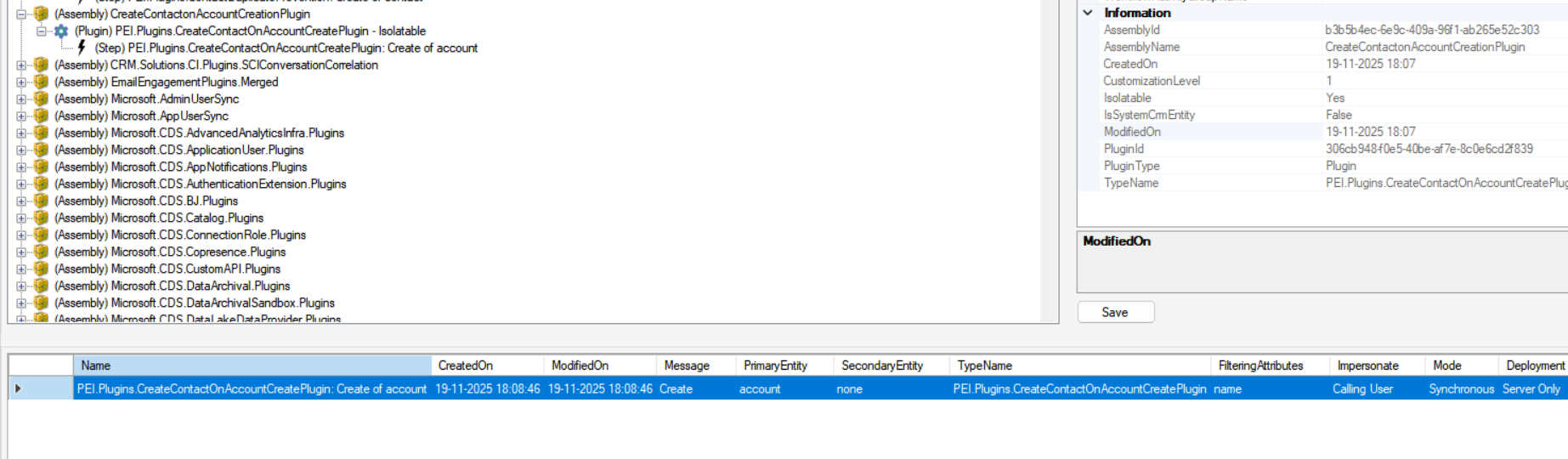
Stage – PostOperation

Mode – Synchronous

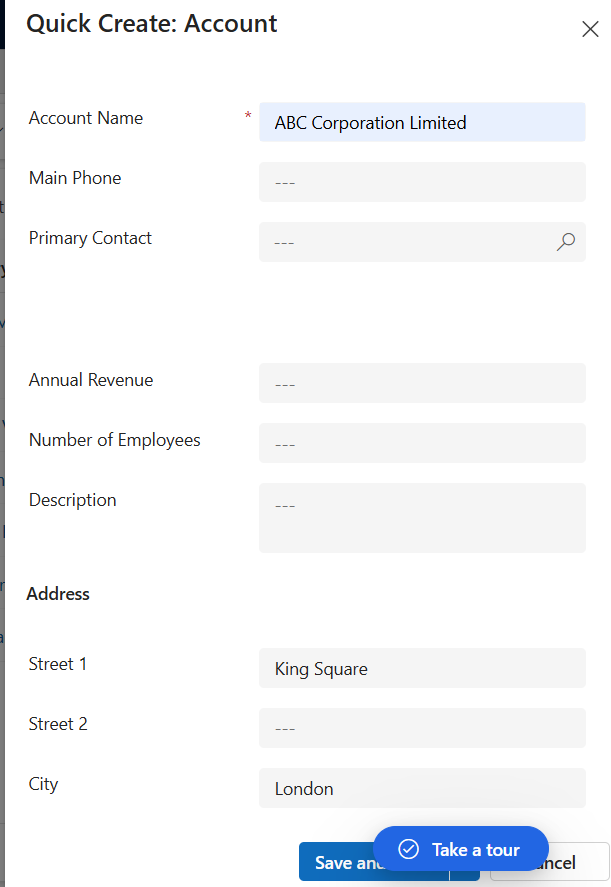
Filtering attributes – name

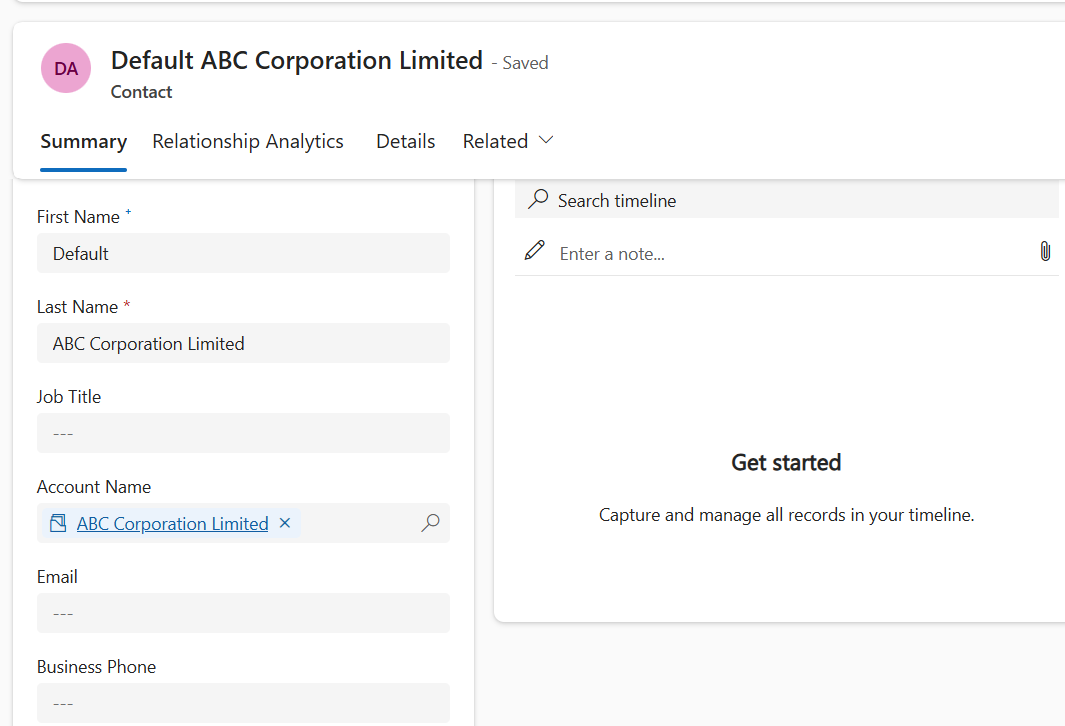
****

Now, it can be seen that step has been registered successfully

****

**Working**

Here, I have created an Account with Account Name as ABC Corporation Limited ****

After saving the record, I could see corresponding Contact record has been created with First Name as Default and Last Name as ABC Corporation Limited, as per plugin code****

1. **Skeleton CI/CD Pipelines:**

To deploy the code through CI/CD pipeline, we need to first clone a repository and then create a feature branch for the fresh changes

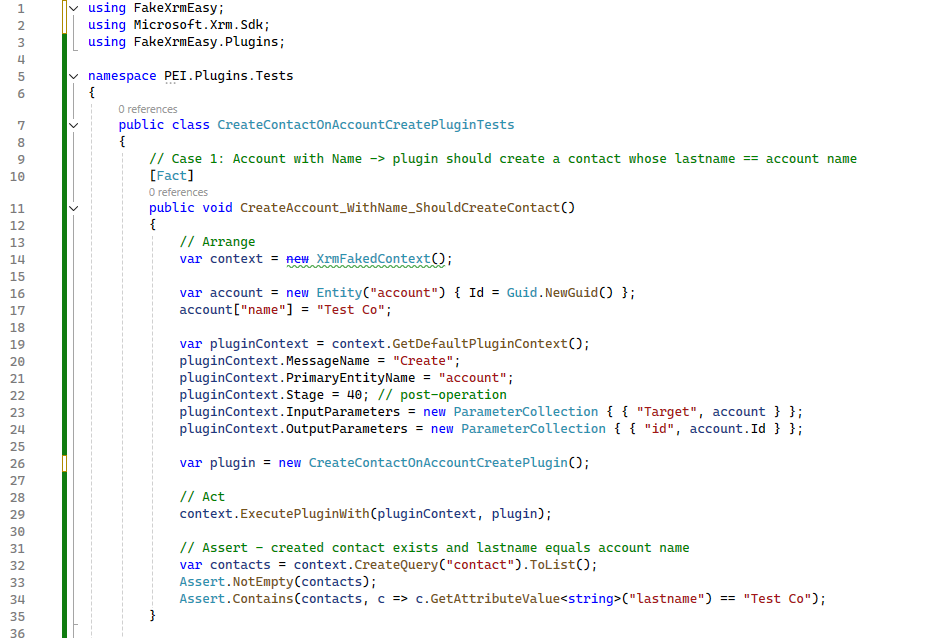
When code is ready, we have to commit and push the code on the feature branch and then create a pull request to merge the feature branch with the master branch.

When code is merged, Build Pipeline (CI) is triggered which creates Deployment Artefact and will contain the changes.

Through Release Pipeline (CD), we can select the deployment artefact and deploy it to next environment.

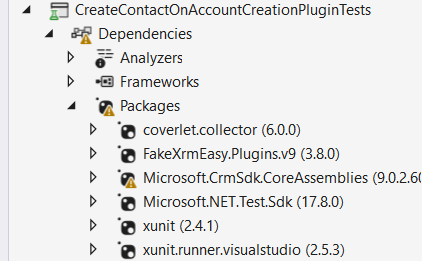
1. **Unit test cases**

Created a test project (xUnit)





**NuGet packages**

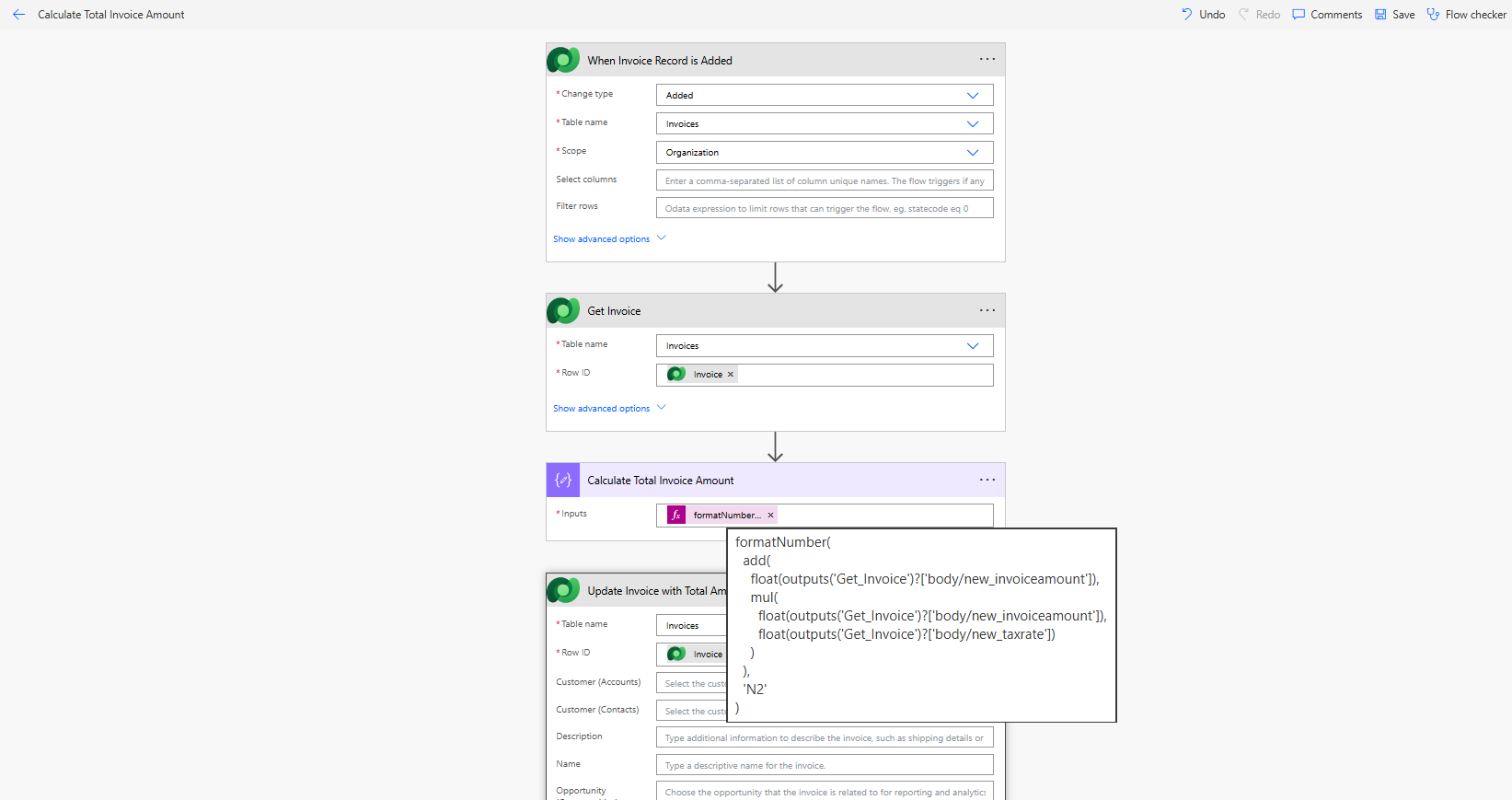
****

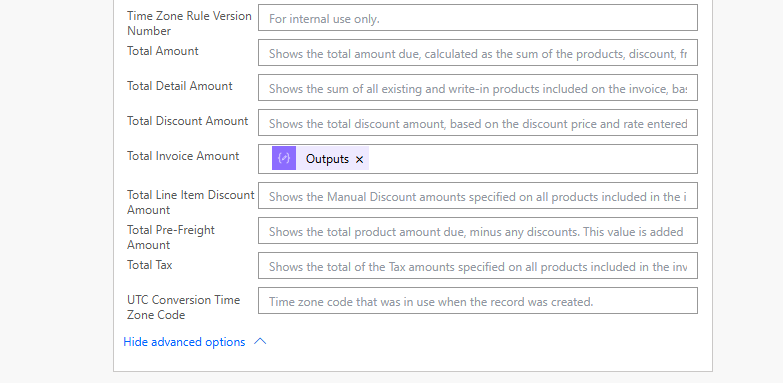
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**3. Use Workflow (or Power Automate)**

**Scenario**:

* Create a flow to calculate the total invoice amount, including tax which triggers when an invoice is created.
  + Inputs: InvoiceAmount (decimal), TaxRate (decimal).
  + Output: TotalAmount = InvoiceAmount + (InvoiceAmount \* TaxRate).

****

****

For this task, I have used Power Automate flow and named the flow as **Calculate Total Invoice Amount.**

Here, I have created 3 custom fields

**Invoice Amount (new\_invoiceamount)**

**Tax Rate (new\_taxrate)**

**Total Invoice Amount (new\_totalinvoiceamount)**

Trigger – when a row is added, modified or deleted

To trigger the flow when new Invoice record is created

Action – Get a row by ID

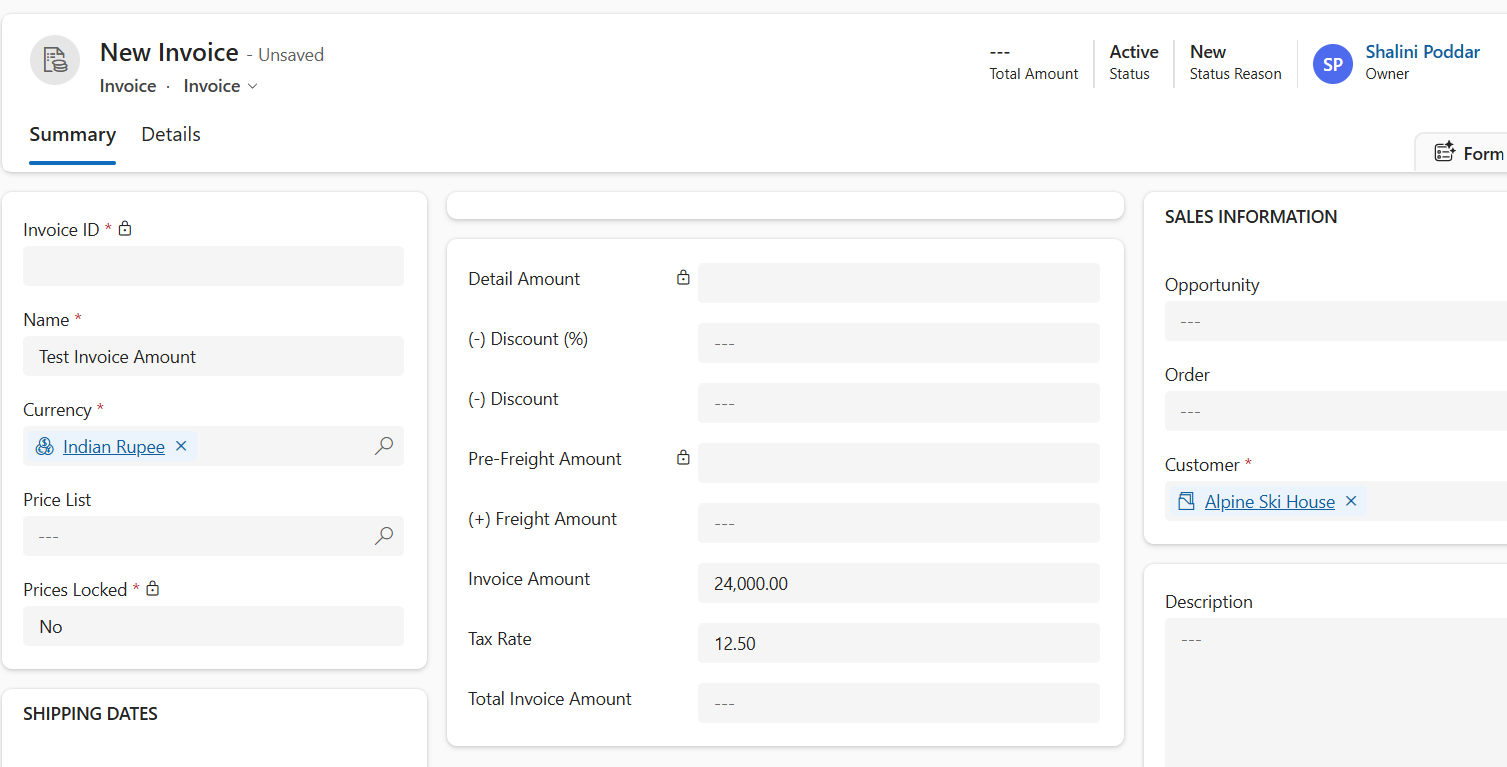
to get full Invoice data and access the fields for calculation

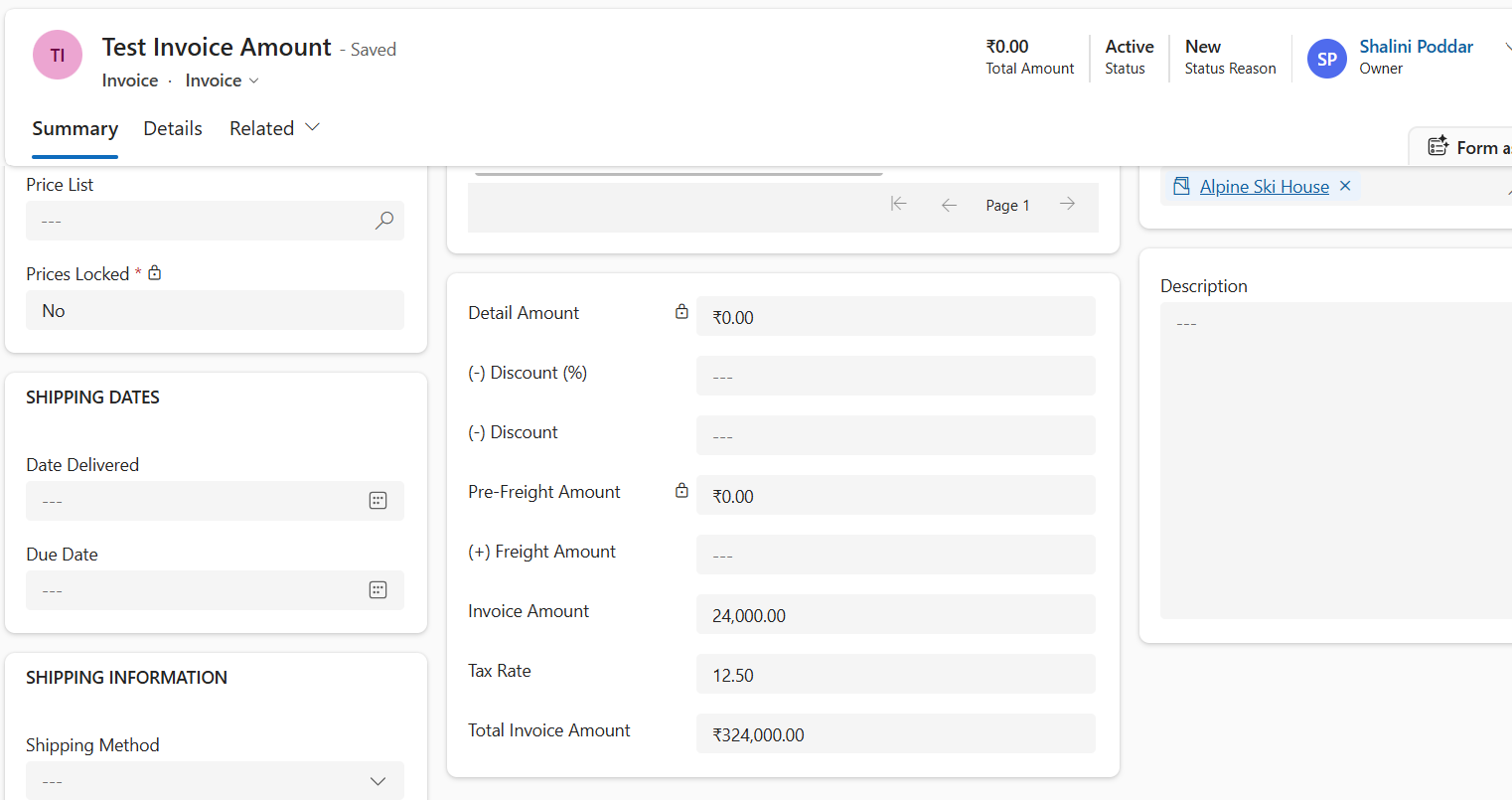
Compose – to calculate the Total Amount as per given criteria

Update a row – to update the total amount field with value fetched from Compose action

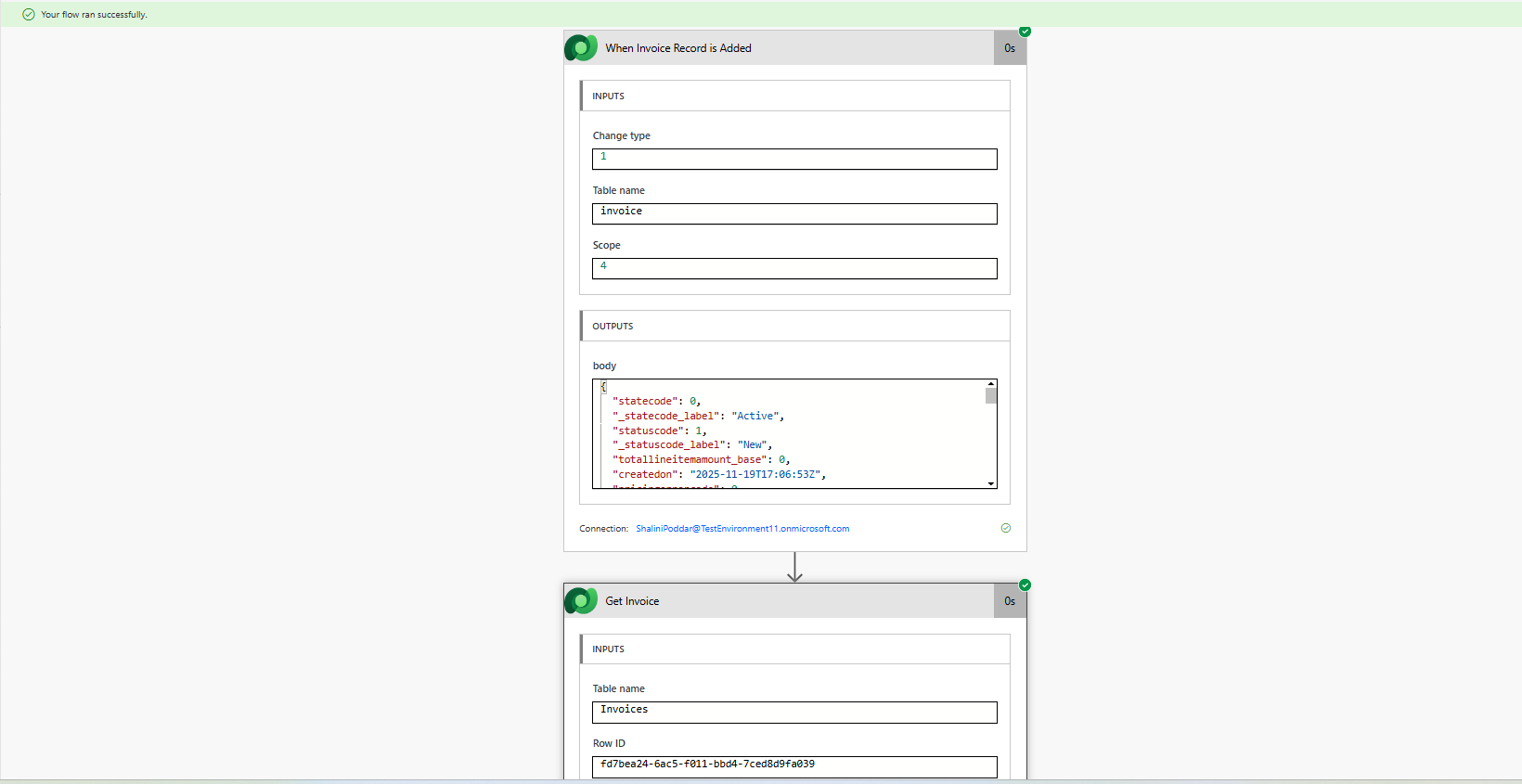
**Working of the flow**

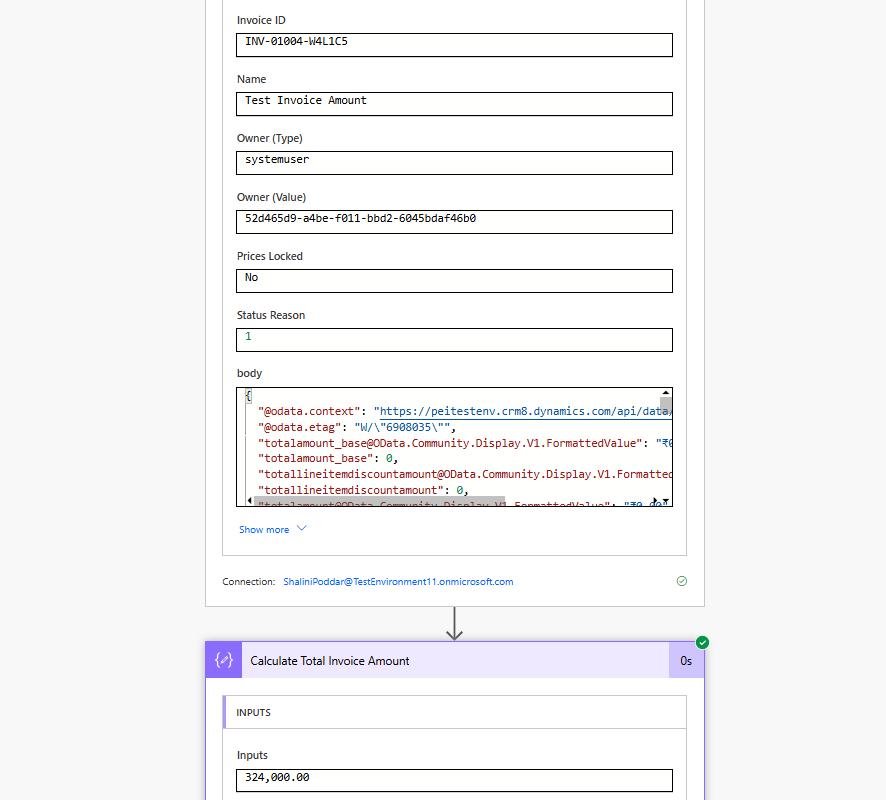
I have created the Invoice record with Invoice Amount and Tax Rate along with other required fields

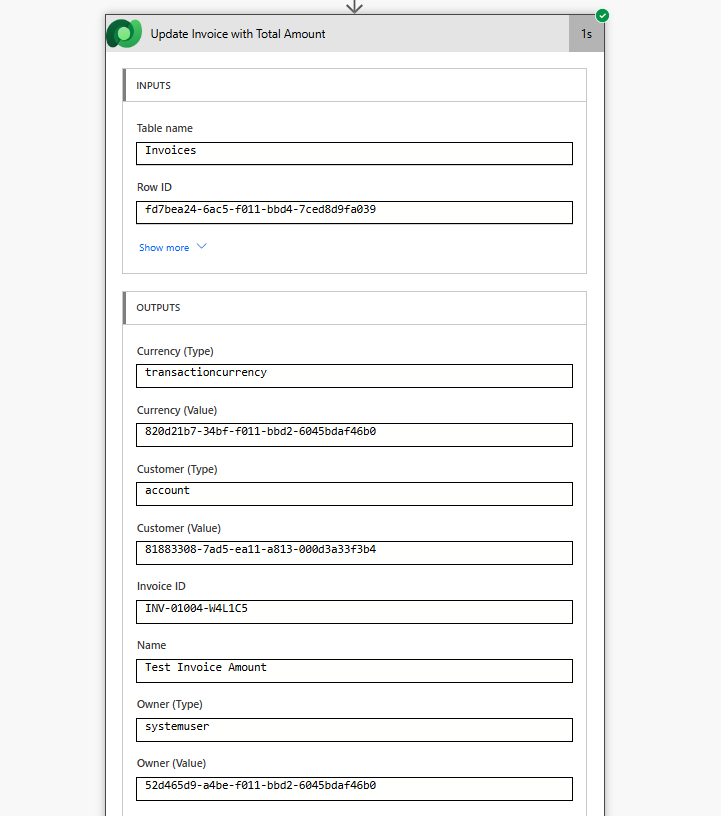
****

After Saving the record and refreshing, we get the value in **Total Invoice Amount** field****

In the background, Power Automate flow has run successfully which has calculated the Total Invoice Amount and populated the field with the value.

****

****

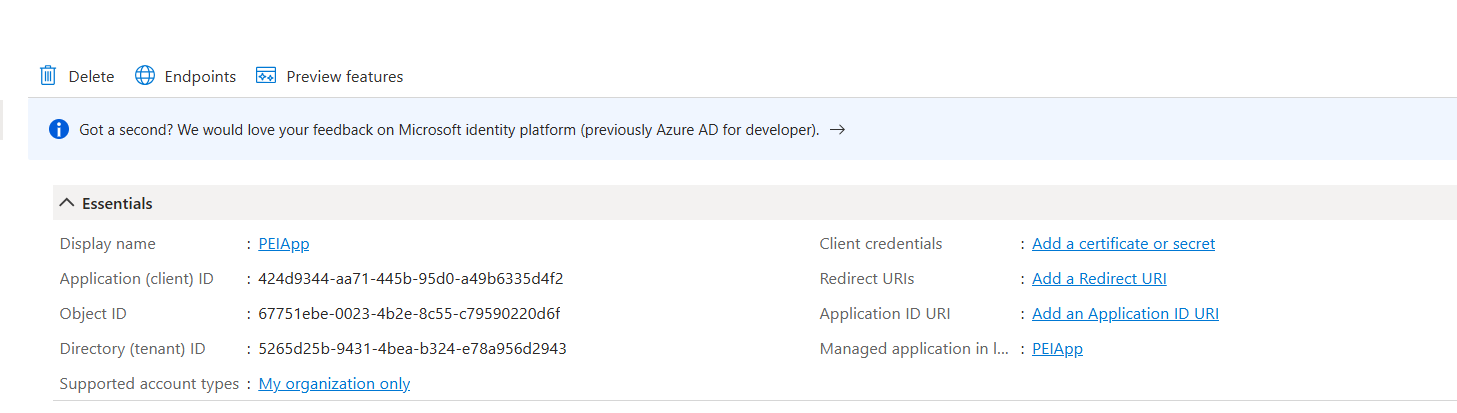
****

**Task 4 - Integration Task**

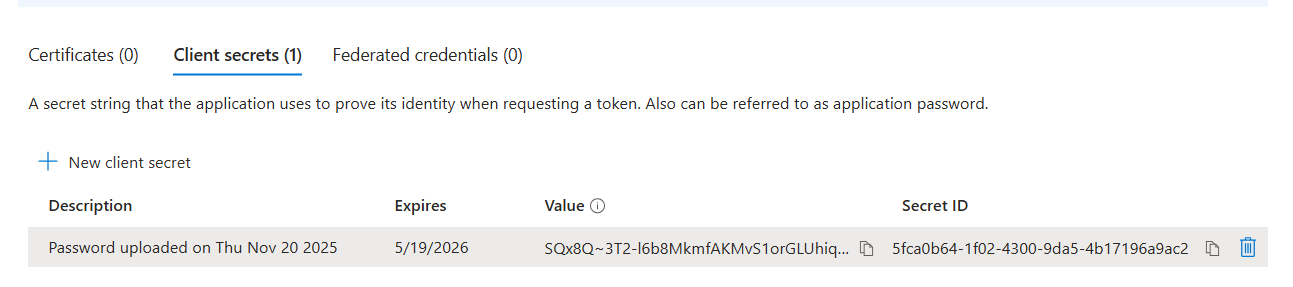
**Scenario**:

* Integrate Dynamics 365 CRM with a mock external API to send customer order details upon Order creation.
  + **Requirements**:
    - Develop a console application or Azure Function in C#.
    - Retrieve newly created Order records.
    - Send details (e.g., Customer Name, Order Total, Order Date) to a mock REST API.
    - Log success or failure responses.

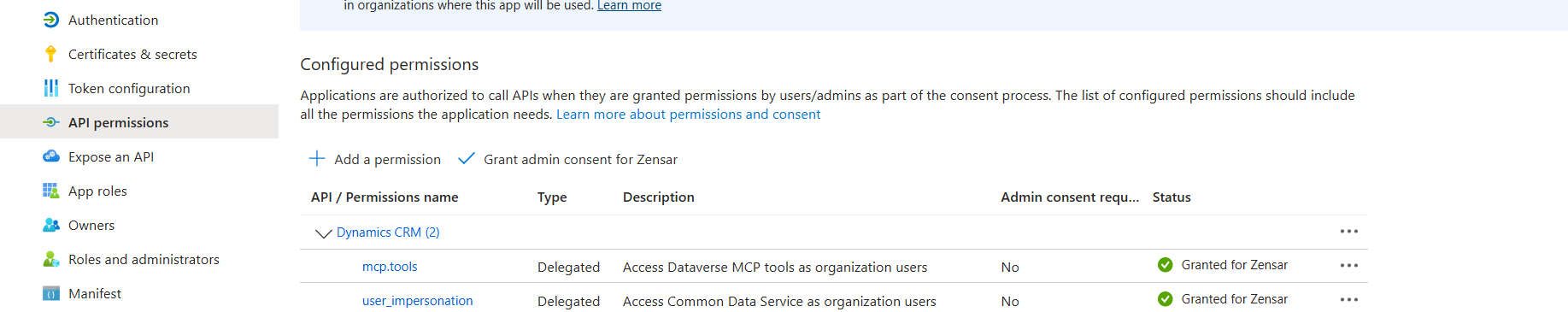
First of all, I went Azure portal (portal.azure.com) to register the app so that I can connect to the environment through the connection

****

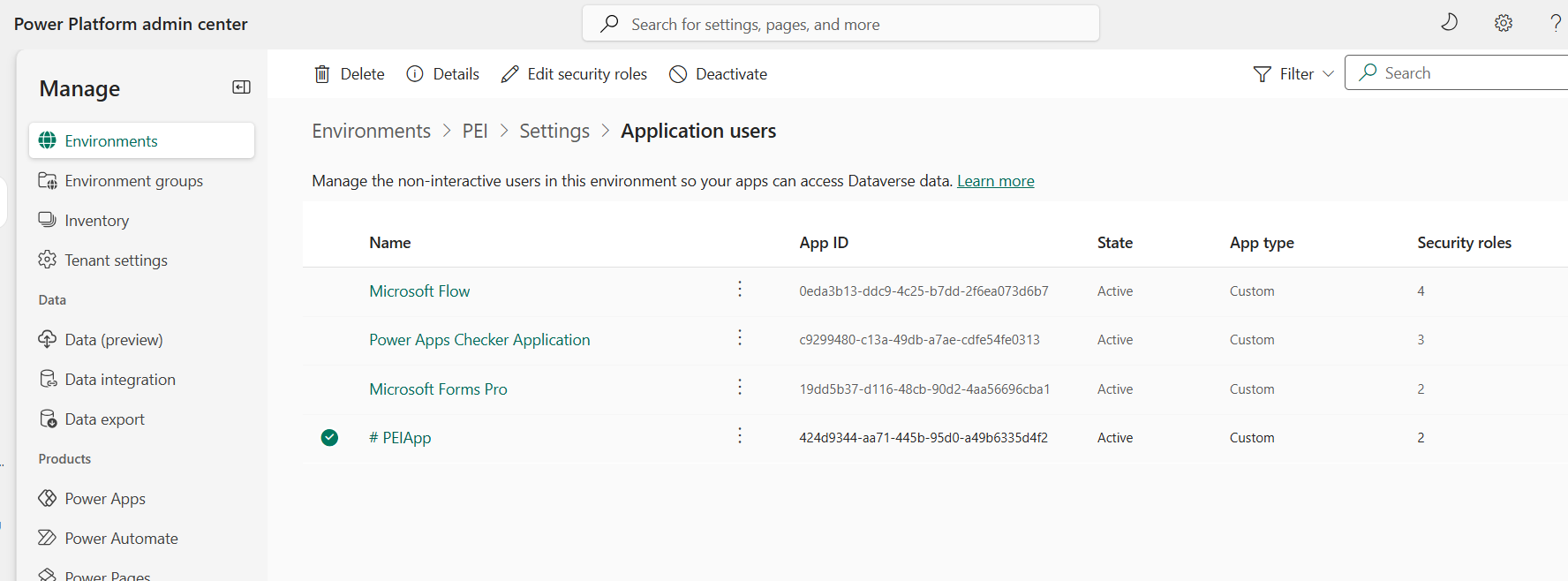
Created new client secret

****

Added API permissions

****

In Power Platform Admin Center -> Environments -> selected the required environment (PEI) -> Settings -> User+ Permissions -> Application Users -> New App User -> Added PEIApp

****

Now, the Console App can authenticate.

**Console App**

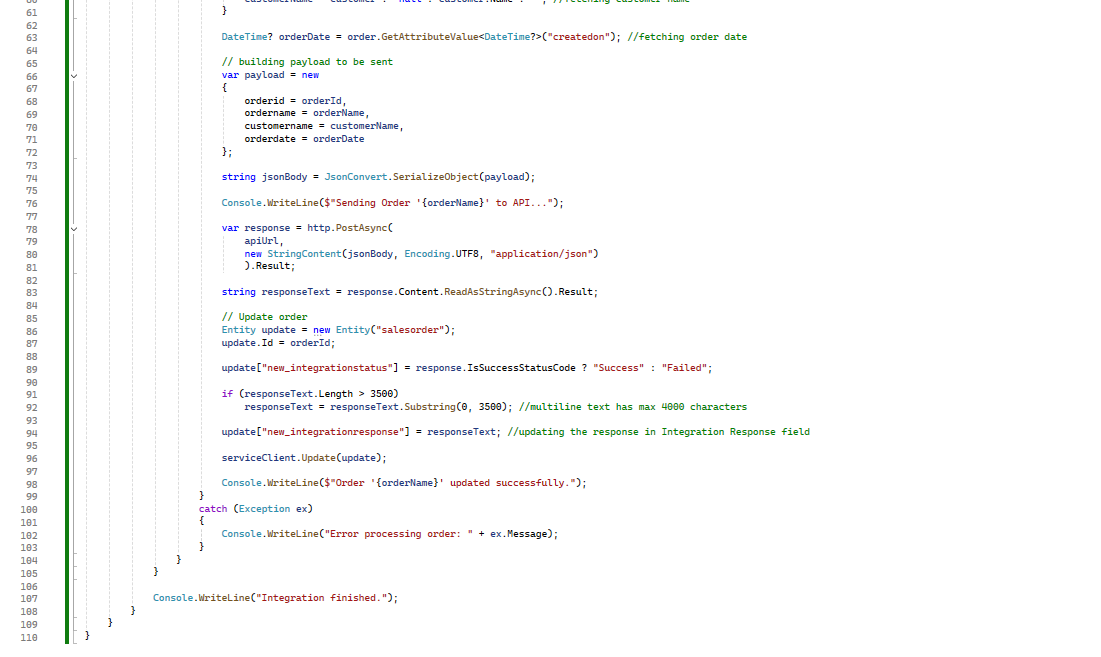
Installed the following NuGet packages:

* Microsoft.PowerPlatform.Dataverse.Client (Dataverse Service Client)

Brings in the Dataverse ServiceClient class so we can connect to Dataverse easily.

* Newtonsoft.Json (for JSON serialization)

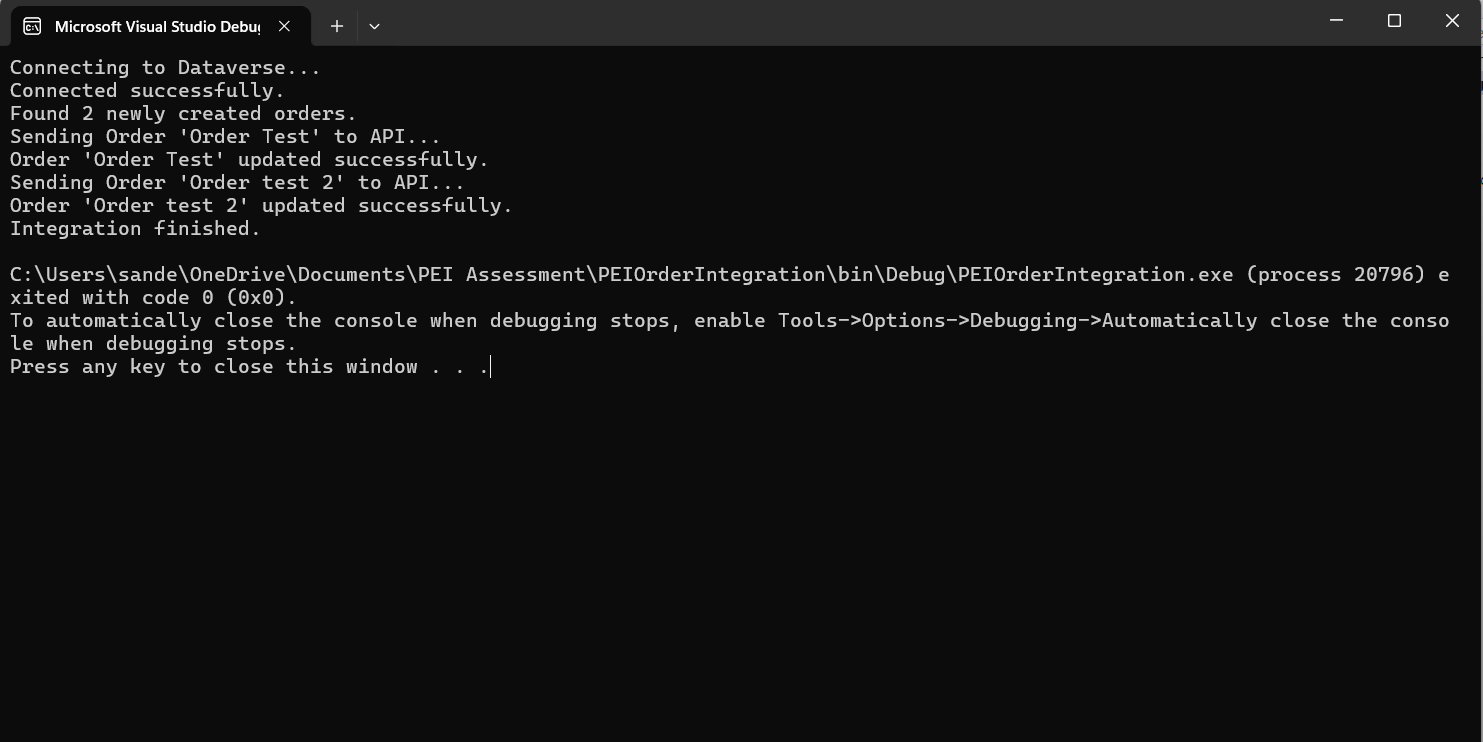
****

****

For Mock API, I have used webhook.site.

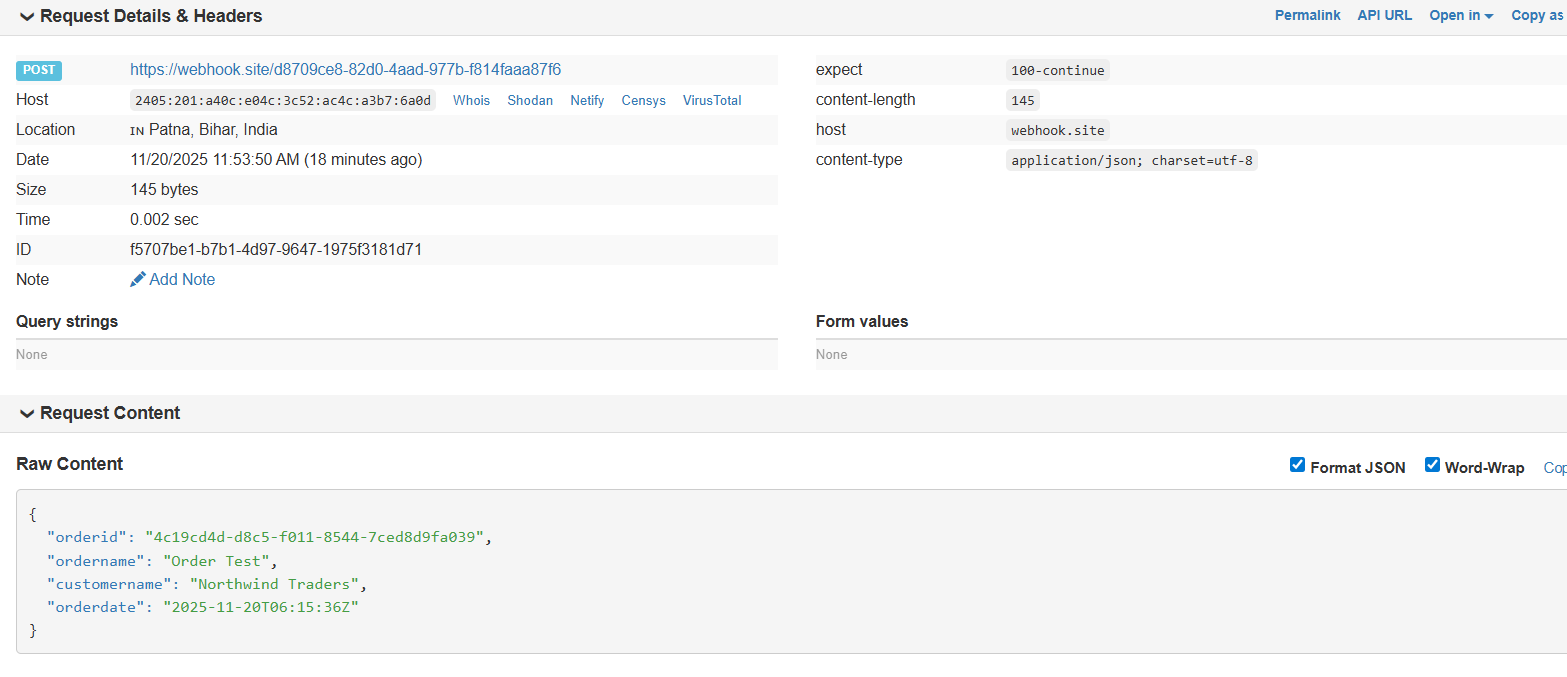
**Working**

**On CRM, I have created 2 new Order records, Order Test and Order test2**

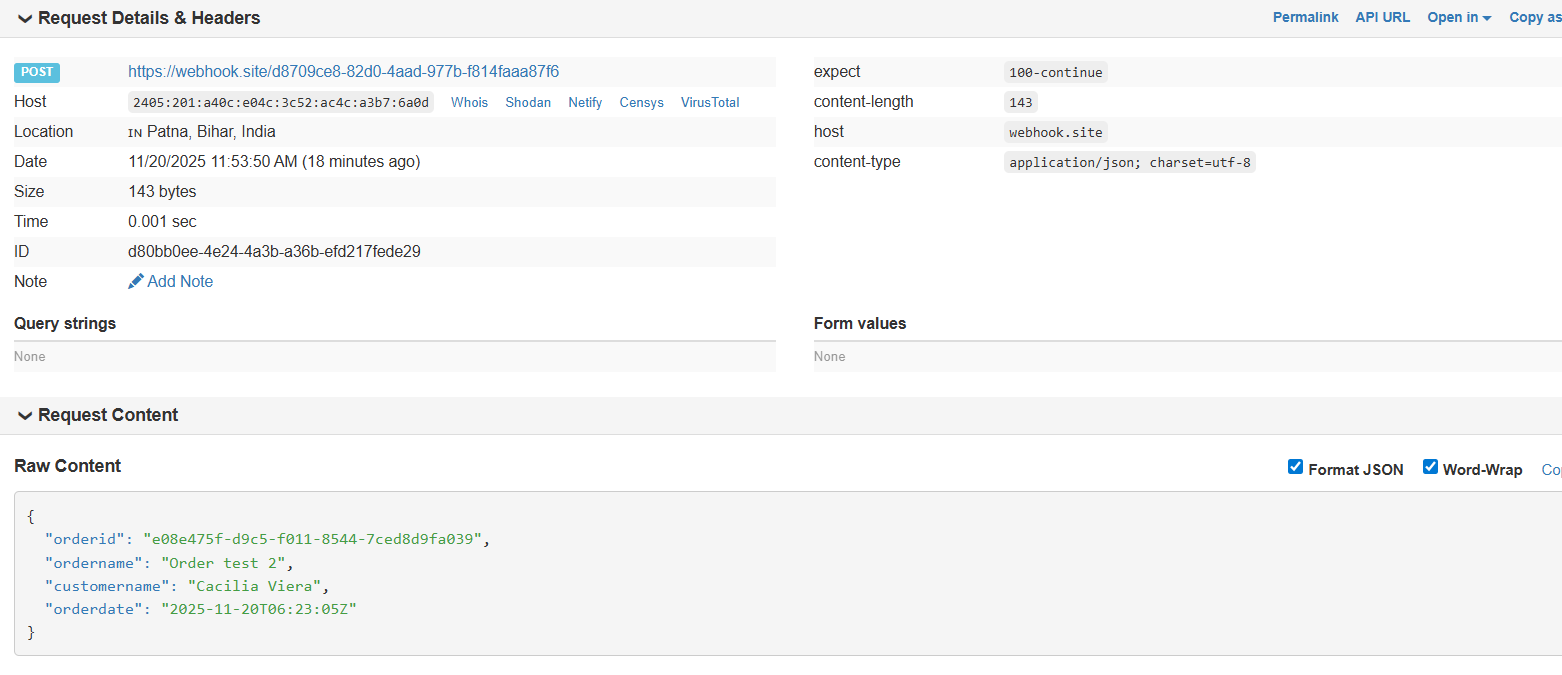
On Visual Studio, I have clicked on Start to run the Console App, the following Console screen popped up with the messages as per added in the code

**On Webhook.site**

**For Order Test**

****

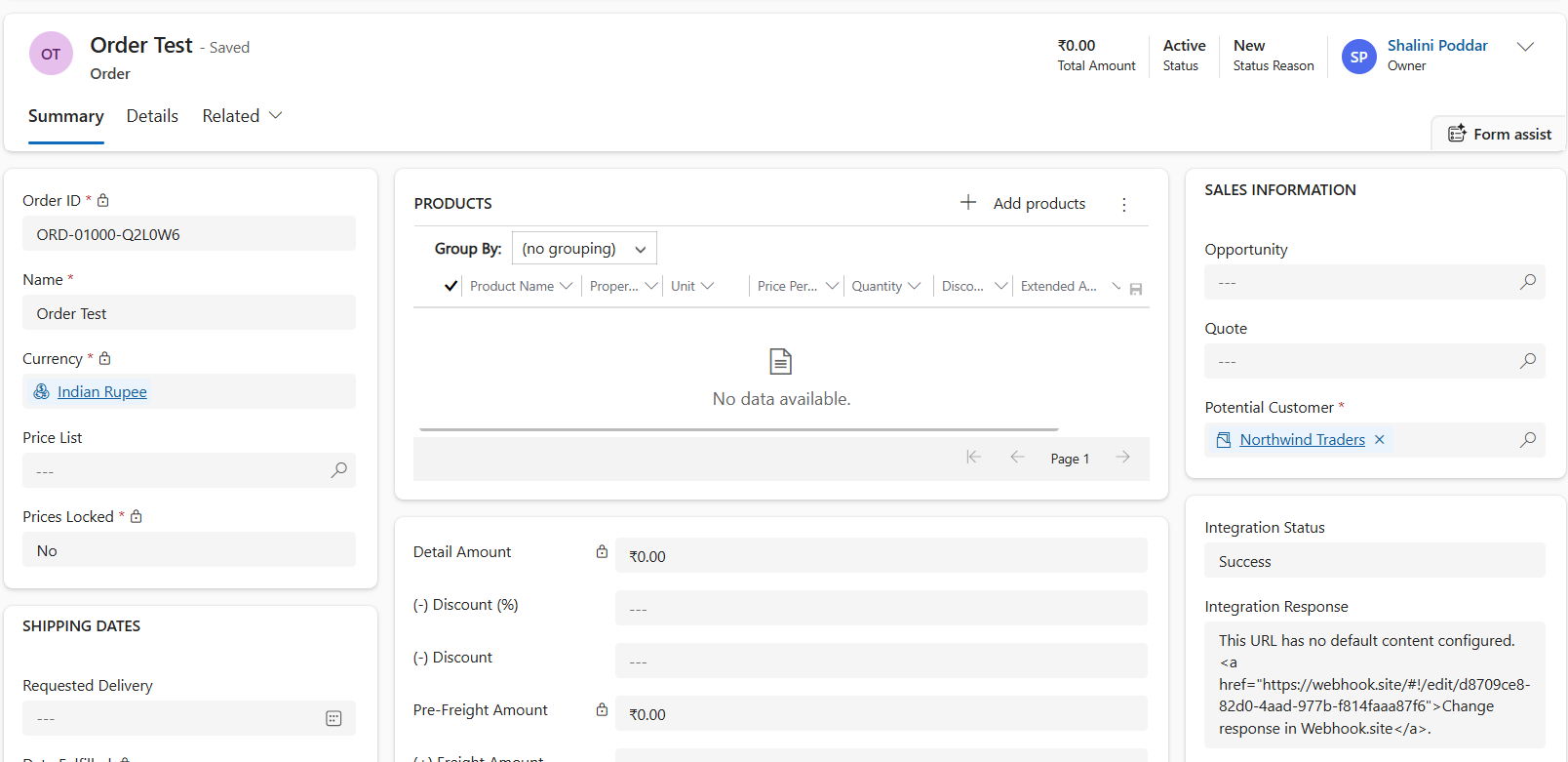
**For Order test 2**

****

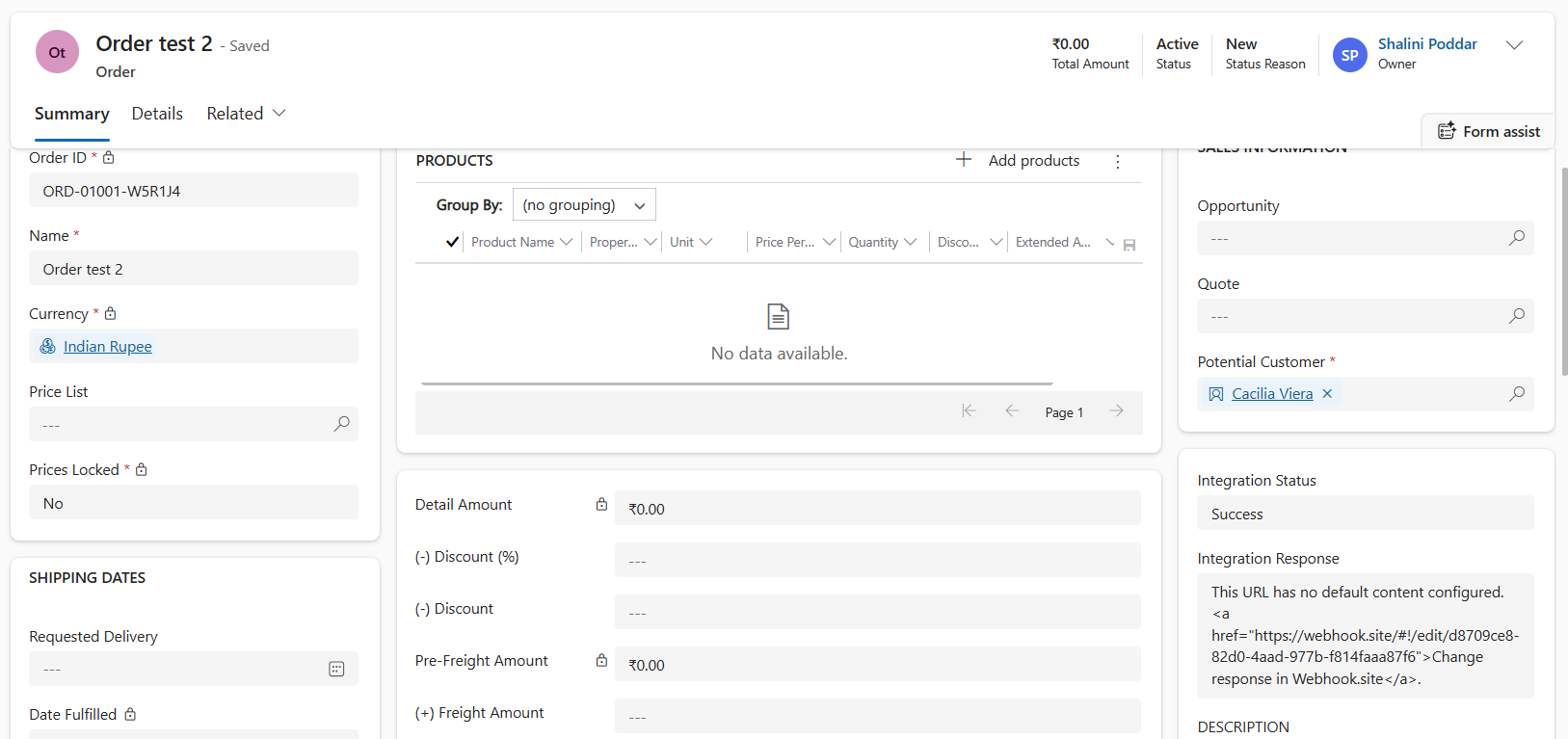
**On CRM**

I have added custom Integrations Status and Integration Response on the Order form, to record the response

**For Order Test**

****

**For Order test 2**

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Task 5 - XRM Toolbox Task**

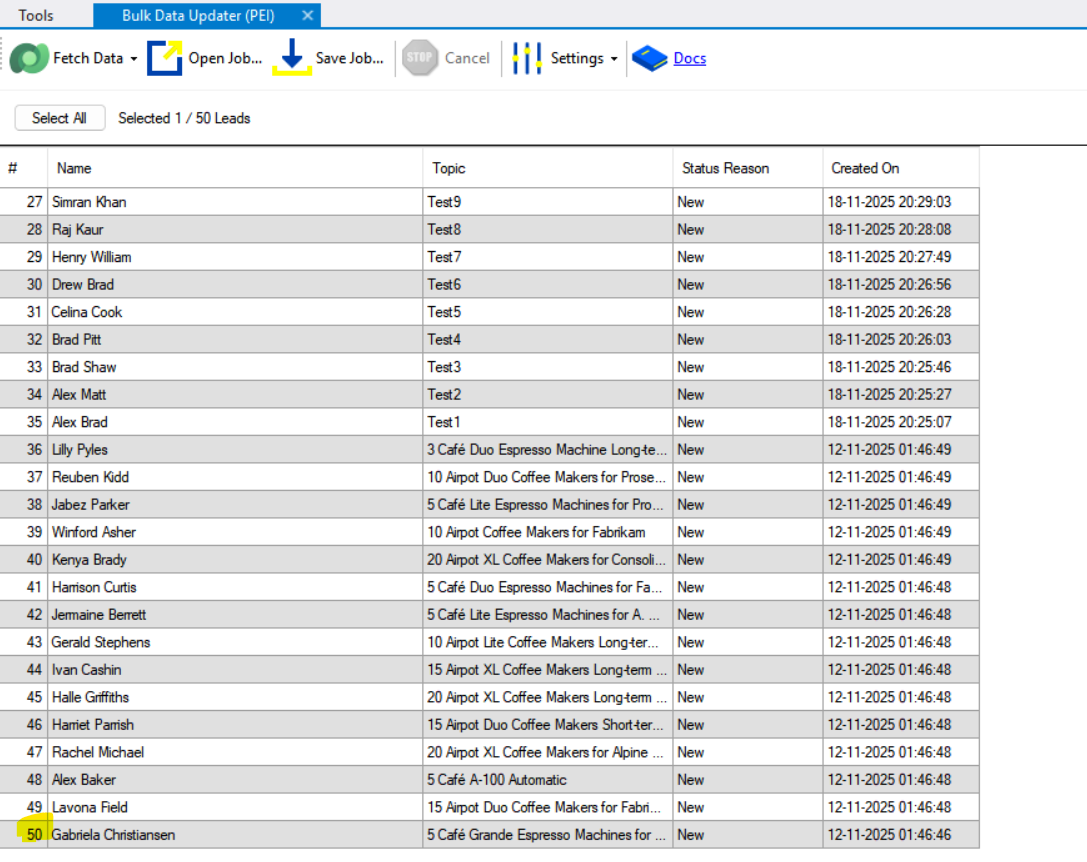
**Scenario**:

* Use XRM Toolbox for the following tasks:
  + **Bulk Update**: Update 50 Leads to set their Status to "Qualified."

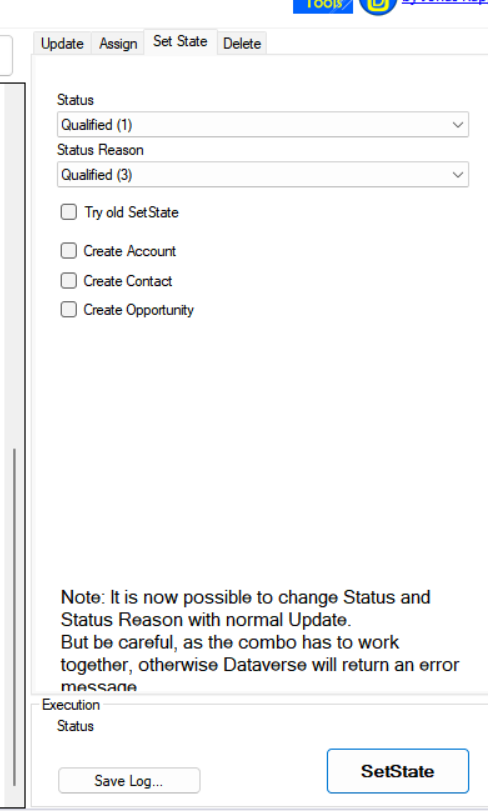
I have used **Bulk Data Updater** tool in XRM Toolbox for this.

Then in **Fetch Data** -> I have selected **Open View** -> Then selected **Lead** and View as **My Open Leads,** which gave me the below list

There is a total of 50 Lead records with Status as Open and Status Reason as New

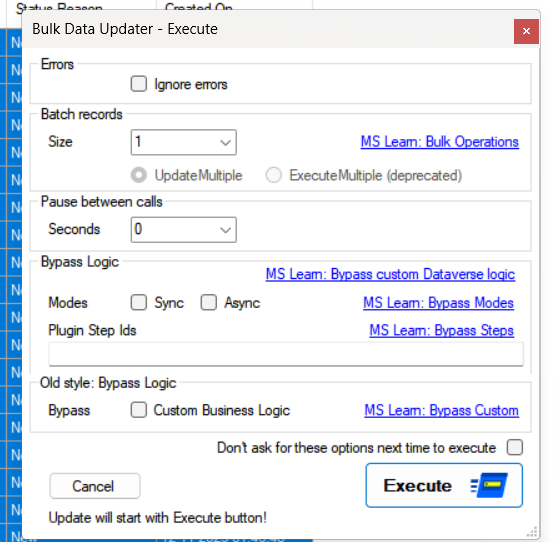


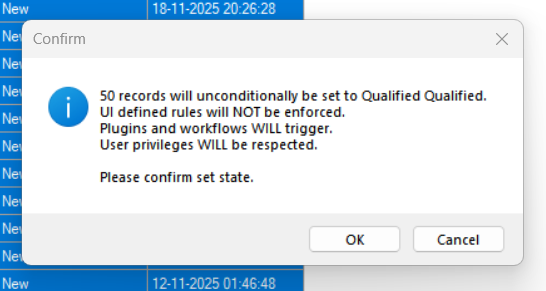
In Set State, I have set the value of Status as Qualified(1) and Status Reason as Qualified(3) and then clicked on **SetState** button

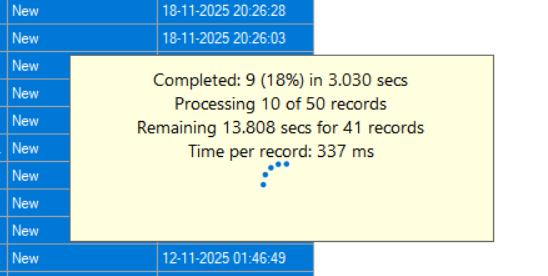


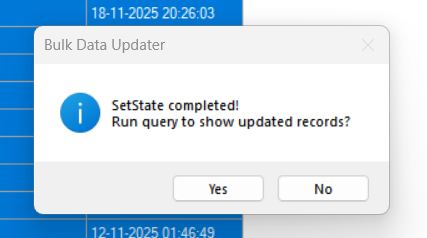
Which will give the below popup and from there, I have clicked on **Execute** button

I have also **Selected All** records, before performing the execution







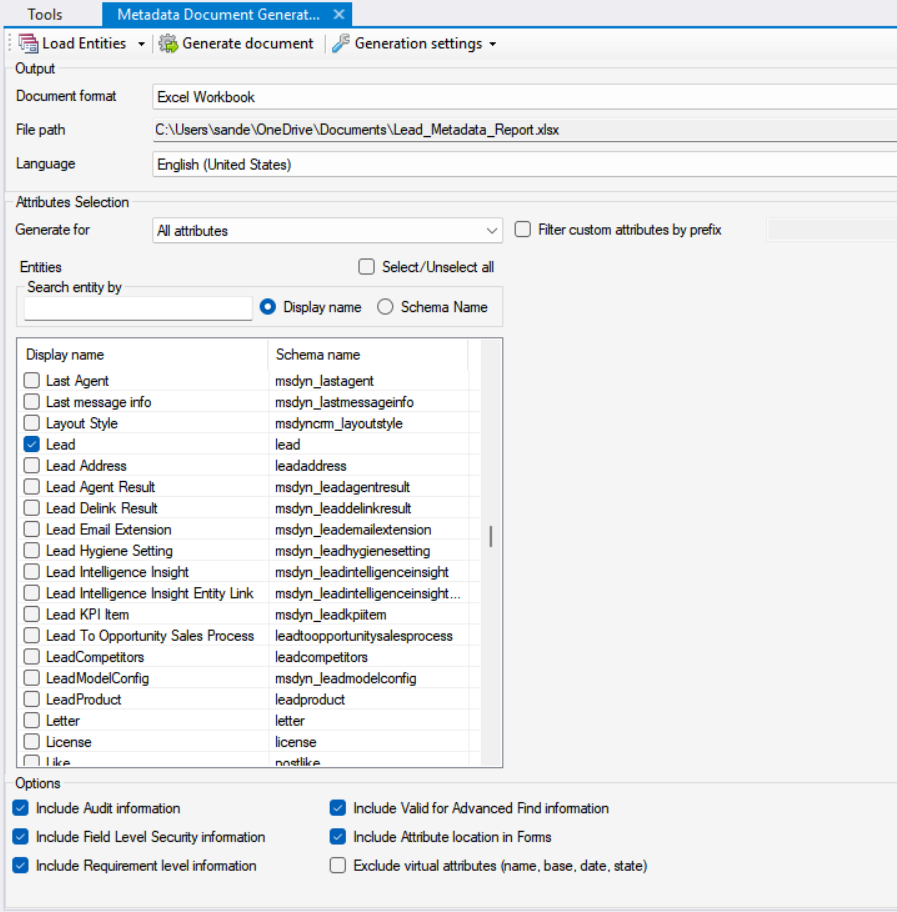


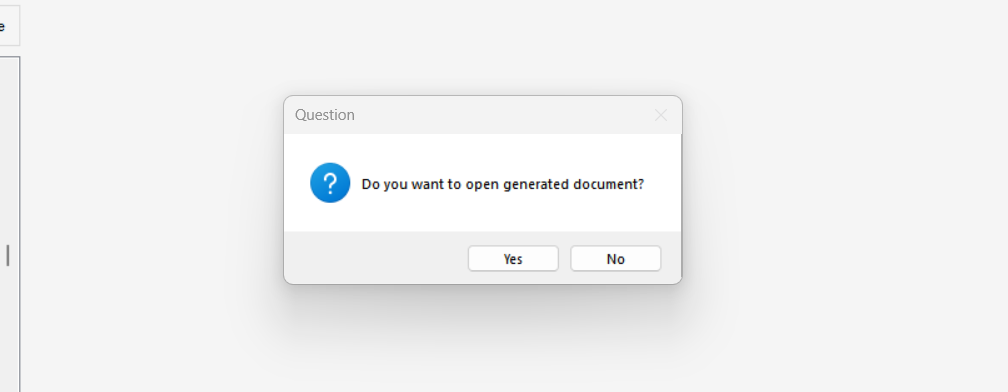
After Update, the Status Reason has been changed to ‘**Qualified’**



* + **Metadata Report**: Generate a report for the Leads entity listing all attributes and their properties.

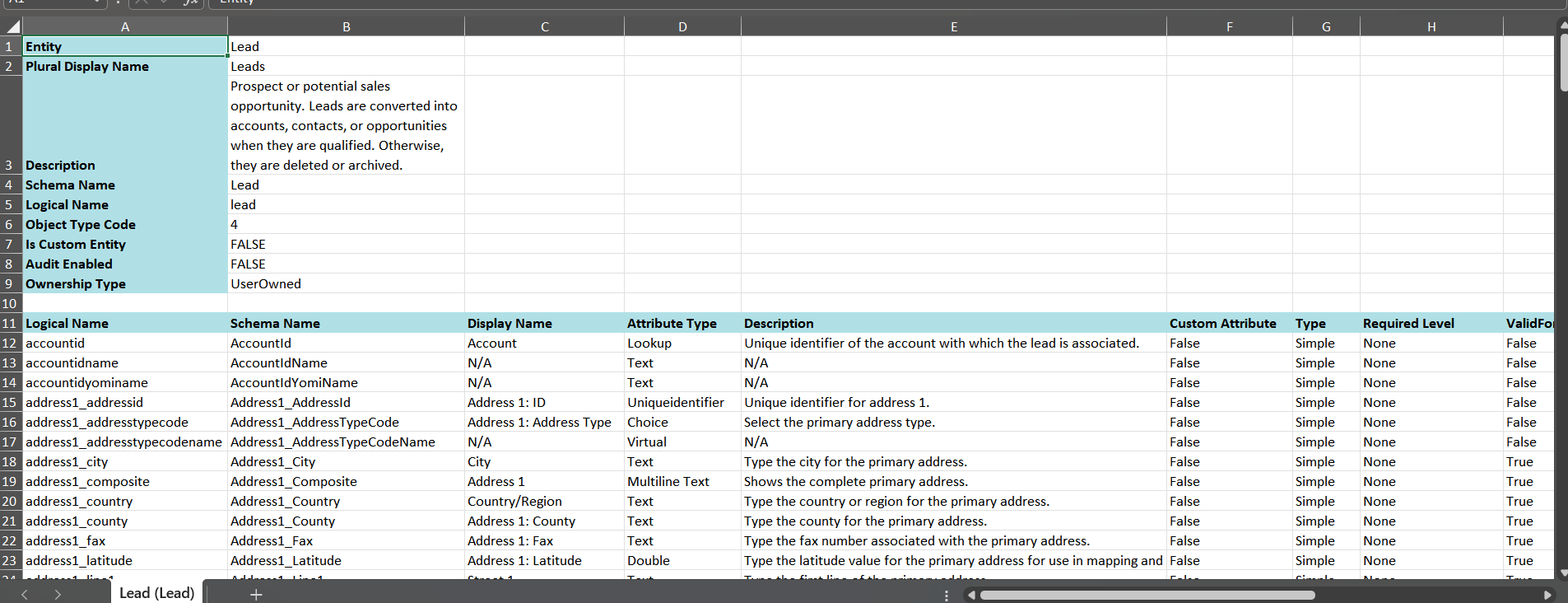
I have used the tool **Metadata Document Generator** in XRM Toolbox to generate the Metadata for All attributes of Lead Entity in Excel





This has generated Metadata in Excel with all Attributes of Lead Entity

There is a total of 234 attributes in Lead Entity



Attached **Lead\_Metadata\_Report.xlsx** along with this word document file.