I adjusted the system for four business cases, making changes to both the front and back ends. I used TypeScript resources and plugins, did the development in Visual Studio, and utilized tools like Plugin Registration, XRM Toolbox, and Fetch XML Builder. I also configured the system using build in methods.

Business Scenario 1

Initially, I created the "Skills" entity, incorporating fields such as Skill, Proficiency Rating (Choice), and User (Lookup). Later I created a Polymorphic Lookup on Opportunity entity, enabling simultaneous searches in both the "Skills" and "System User" entities. This facilitates the easy identification of users possessing a specific skill. The results are sorted from the highest to the lowest proficiency level. Notably, the lookup restricts the storage of an entity of type "Skill" and only allows "System User" since a System User may serve as the owner of an Opportunity. This control is implemented through TypeScript code.

TypeScript Code

```typescript
let Form: D365.Sdk.FormContext;
0 references
export async function OnLoad(executionContext: D365.Sdk.ExecutionContext) {
    Form = executionContext.getFormContext() as D365.Sdk.FormContext;
    TeamOnChange();
}

1 reference
export async function TeamOnChange() {
    let Team = Form.getControl("new_salesteamid").getAttribute().getValue();

    if (Team) {
        if (Team[0].entityType == "cr260_skill") {
            const confirmStrings = {
                text: "You can not select a skill for this field. Please select a record from Teams Entity!" ,
                title: `Warning!` ,
            };

            const confirmOptions = {
                height: 200,
                width: 400,
            };
            Helper.CRM.getXrm().Navigation.openAlertDialog(confirmStrings, confirmOptions);
```

To finalize the solution, I added a plugin ensuring synchronization between the custom field "Sales Agent" and the default "Owner" field, keeping them automatically updated. I specifically designed the plugin to update the "OwnerId" field during the pre-operation stage, minimizing unnecessary traffic. This approach assumes for other background processes triggered by the update of the default Owner field.

C# Code

```csharp
try
{
    if (Target.Contains("new_salesteamid"))
    {
        Target["ownerid"] = Target.GetAttributeValue<EntityReference>("new_salesteamid");
    }
}
```

Note: For better search, a new view on System User is to be created that filters only Users that are part of Sales Team. The view will be selected for both Lookups on Skills and Opportunity entities.

Business Scenario 2

- **Onboarding and NDA agreements can only be created once for any account.**

In this case, I created a plugin that uses a fetchXml with conditions for "Status" and "Account ID" to check if there are active records related to the account in the "Lookup" field (for the "Onboarding" and "NDA" OptionSet). In cases where the fetchXml returns more than 0 records (the plugin is in the PreOperation step), it displays an error to the user: "There already is an agreement of type Onboarding associated with this Account."

```
EntityReference AccountER = Target.GetAttributeValue<EntityReference>("cr260_account");
Entity Account = service.Retrieve(AccountER.LogicalName, AccountER.Id, new ColumnSet("primarycontactid"));
EntityReference Contact = Account.GetAttributeValue<EntityReference>("primarycontactid");
int AgreementType = Target.GetAttributeValue<OptionSetValue>("cr260_agreementtype").Value;

switch (AgreementType)
{
    case (int)agreementType.Onboarding:
        //Check if any Onboarding Agreements exists with selected Account.
        var fetchOnboarding = $@"<fetch version='1.0' mapping='logical' no-lock='false' distinct='true'>
            <entity name='cr260_agreement'>
            <attribute name='cr260_account'/>
            <attribute name='cr260_agreementtype'/>
             <filter type='and'>
             <condition attribute='statecode' operator='eq' value='{(int)stateCode.Active}'/>
             <condition attribute='cr260_account' operator='eq' value='{AccountER.Id}'/>
             <condition attribute='cr260_agreementtype' operator='eq' value='{(int)agreementType.Onboarding}'/>
           </filter>
           </entity>
           </fetch>";
        EntityCollection Onboardings = service.RetrieveMultiple(new FetchExpression(fetchOnboarding));

        if (Onboardings.Entities.Count > 0)
        {
            throw new InvalidPluginExecutionException("There already is an agreement of type Onboarding associated with this Account");
        }
        else if (Target.Contains("cr260_agreementstartdate") && Target.Contains("cr260_agreementenddate"))...
        break;
    case (int)agreementType.NDA:
        //Check if any NDAs Agreements exists with selected Account.
        var fetchNDA = $@"<fetch version='1.0' mapping='logical' no-lock='false' distinct='true'>
            <entity name='cr260_agreement'>
            <attribute name='cr260_account'/>
             <attribute name='cr260_agreementtype'/>
                 <filter type='and'>
                 <condition attribute='statecode' operator='eq' value='{(int)stateCode.Active}'/>
                 <condition attribute='cr260_account' operator='eq' value='{AccountER.Id}'/>
                 <condition attribute='cr260_agreementtype' operator='eq' value='{(int)agreementType.NDA}'/>
              </filter>
             </entity>
           </fetch>";
        EntityCollection NDA = service.RetrieveMultiple(new FetchExpression(fetchNDA));

        if (NDA.Entities.Count > 0)
        {
            throw new InvalidPluginExecutionException("There already is an agreement of type NDA associated with this Account");
```

- **We want to have this field set automatically to yes whenever the Onboarding agreement connected to the account related to the current opportunity contains values in the Agreement start date and Agreement end date.**

For this request, I create a QueryExpression for the "Opportunity" entity with conditions for "Status" and "ParentAccount," where the latter should match the Account for which the "Onboarding" type Agreement is being created. Using a foreach loop, I update the "T&C" field for each Opportunity. This field is hidden in the form and is visible the BPF Step because otherwise fields that are not present on form are not able to be accessed.

For the Update I have initialized a new entity and assigned the id of each opportunity that is to be updated and only the field that is needed. This ensures lightweight update operation.

```
else if (Target.Contains("cr260_agreementstartdate") && Target.Contains("cr260_agreementenddate"))
{
    //Create Query with filter Account Id and Update Field "cr260_tc" on Opportunities.
    QueryExpression query = new QueryExpression("opportunity");
    query.ColumnSet = new ColumnSet("parentaccountid", "statecode");

    FilterExpression filter = new FilterExpression(LogicalOperator.And);
    ConditionExpression stateCondition = new ConditionExpression("statecode", ConditionOperator.Equal, (int)stateCode.Active);
    ConditionExpression accountCondition = new ConditionExpression("parentaccountid", ConditionOperator.Equal, Account.Id);
    filter.AddCondition(stateCondition);
    filter.AddCondition(accountCondition);

    query.Criteria = filter;

    EntityCollection Opportunities = service.RetrieveMultiple(query);

    foreach(var Opportunity in Opportunities.Entities){

        Entity OpportunityEnt = new Entity(Opportunity.LogicalName, Opportunity.Id);
        bool tc = Opportunity.GetAttributeValue<bool>("cr260_tc");

        Opportunity["cr260_tc"] = true;
        service.Update(Opportunity);
        Console.WriteLine("Updated Records" + Opportunity.Id.ToString());
    }
}
```

Business Scenario 3

- **<u>Whenever creating a work order, I want to be able to stop the creator if they try to assign it to an agent that is not scheduled on that specific day of the week.</u>**

In this case, I've created a plugin that checks the "Is Scheduled" Boolean fields in the "Agent" entity. It creates a Boolean variable named "scheduledOnMatches" and sets it to "True." Using a switch method, we verify the selected day and check if the agent is available. If the agent is not available, it displays an error to the user: "Agent " + AgentName (using dynamic values to display the agent's name) + " isn't available on that day."

```
ColumnSet allColumns = new ColumnSet(true);
EntityReference AgentER = Target.GetAttributeValue<EntityReference>("new_assignedagent");
Entity AgentEnt = service.Retrieve(AgentER.LogicalName, AgentER.Id, allColumns);

string AgentName = AgentEnt.GetAttributeValue<string>("new_agentname");
bool WorksOnMonday = AgentEnt.GetAttributeValue<bool>("new_isscheduledmonday");
bool WorksOnTuesday = AgentEnt.GetAttributeValue<bool>("new_isscheduledtuesday");
bool WorksOnWednesday = AgentEnt.GetAttributeValue<bool>("new_isscheduledwednesday");
bool WorksOnThursday = AgentEnt.GetAttributeValue<bool>("new_isscheduledthursday");
bool WorksOnFriday = AgentEnt.GetAttributeValue<bool>("new_isscheduledfriday");
int scheduleDay = Target.GetAttributeValue<OptionSetValue>("new_scheduledon").Value;

bool scheduledOnMatches = true;

switch (scheduleDay)
{
    case (int)scheduledDay.Monday:
        scheduledOnMatches = WorksOnMonday;
        break;
    case (int)scheduledDay.Tuesday:
        scheduledOnMatches = WorksOnTuesday;
        break;
    case (int)scheduledDay.Wednesday:
        scheduledOnMatches = WorksOnWednesday;
        break;
    case (int)scheduledDay.Thursay:
        scheduledOnMatches = WorksOnThursday;
        break;
    case (int)scheduledDay.Friday:
        scheduledOnMatches = WorksOnFriday;
        break;
}
if (scheduledOnMatches != true)
{
    throw new InvalidPluginExecutionException("Agent " + AgentName + " isn't available on that day");
}
```

Business Scenario 4

- **Whenever I create a Lead, I want the topic to populate with the date of creation.**

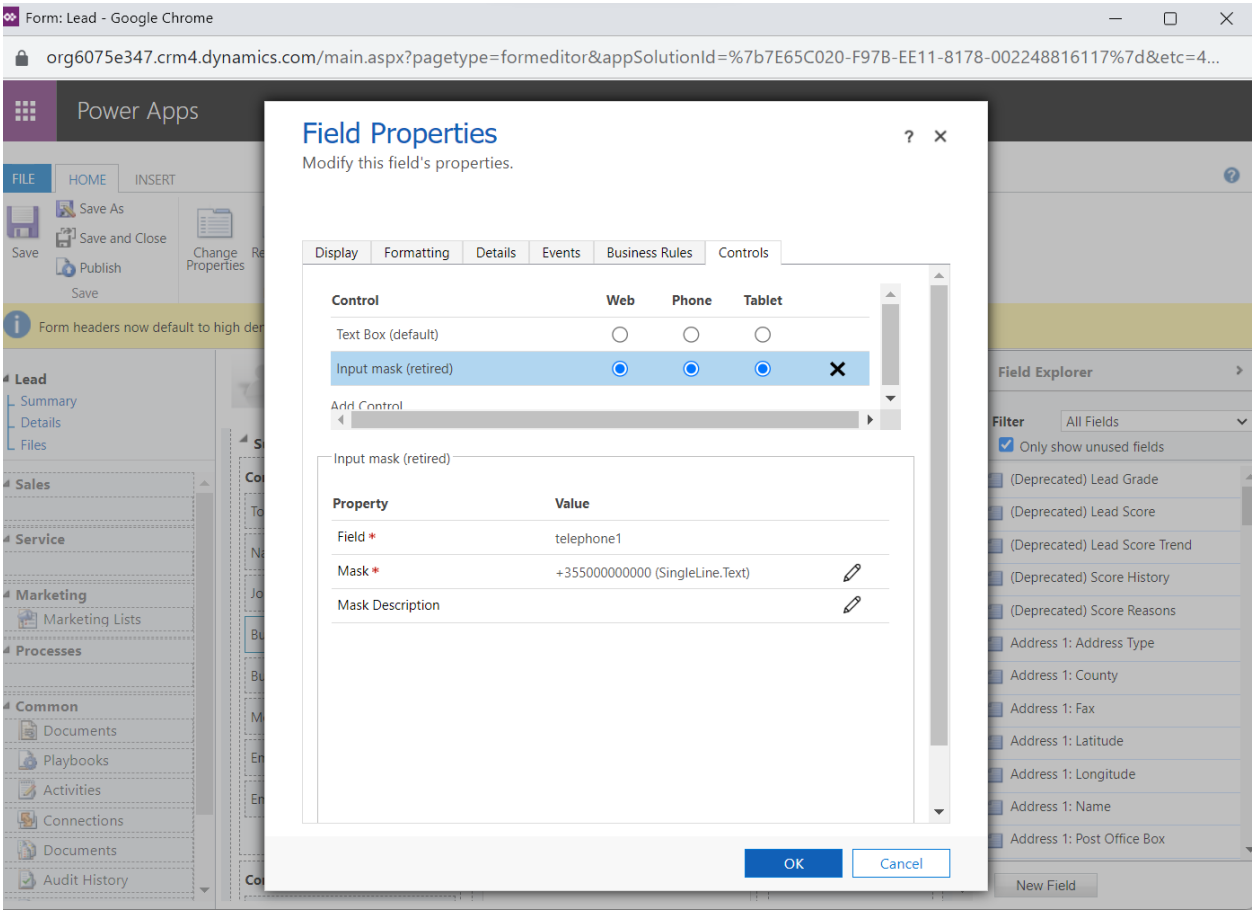
To solve this, I've created a plugin that runs on PreOperation of events Create/Update to acccess the value of the "Topic" field that the user has provided and append the creation date to the field.

```
string Topic = Target.GetAttributeValue<string>("subject");
Entity newTask = new Entity("task");

newTask["regardingobjectid"] = new EntityReference("lead", Target.Id);
newTask["subject"] = "Follow Up";
Target["subject"] = Topic +" "+ DateTime.UtcNow.ToString("dd/MM/yyyy");//Format to Input only the Date.
service.Update(Target);
//  service.Create(newTask);
```

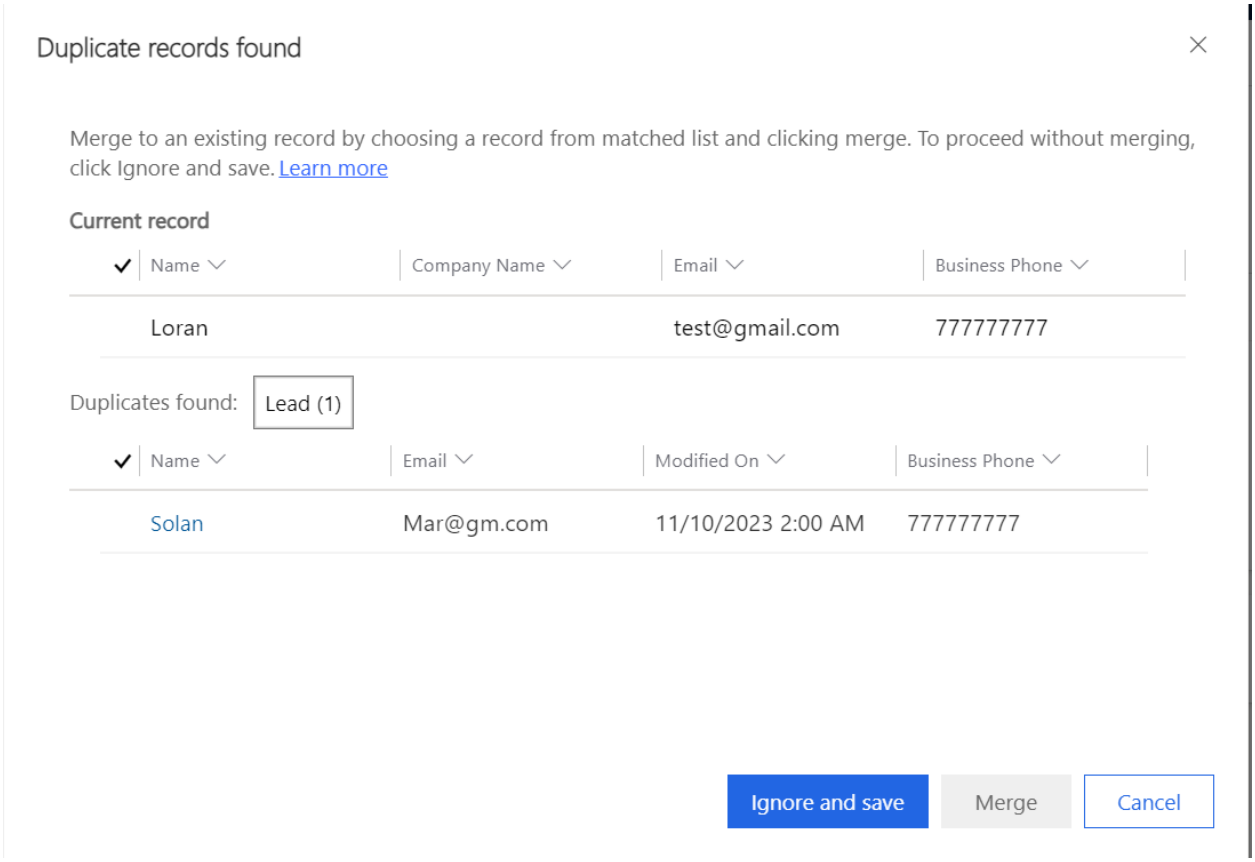
- **I want the phone number of the lead to be checked for format and validity issues, such as inserting letters and country codes that are different from +355. Also, the length of the numbers coming after +355 should not be longer than 9 numbers.**

I've managed this by configuring an "Input Mask" for the "Business Phone" field with the corresponding validations.



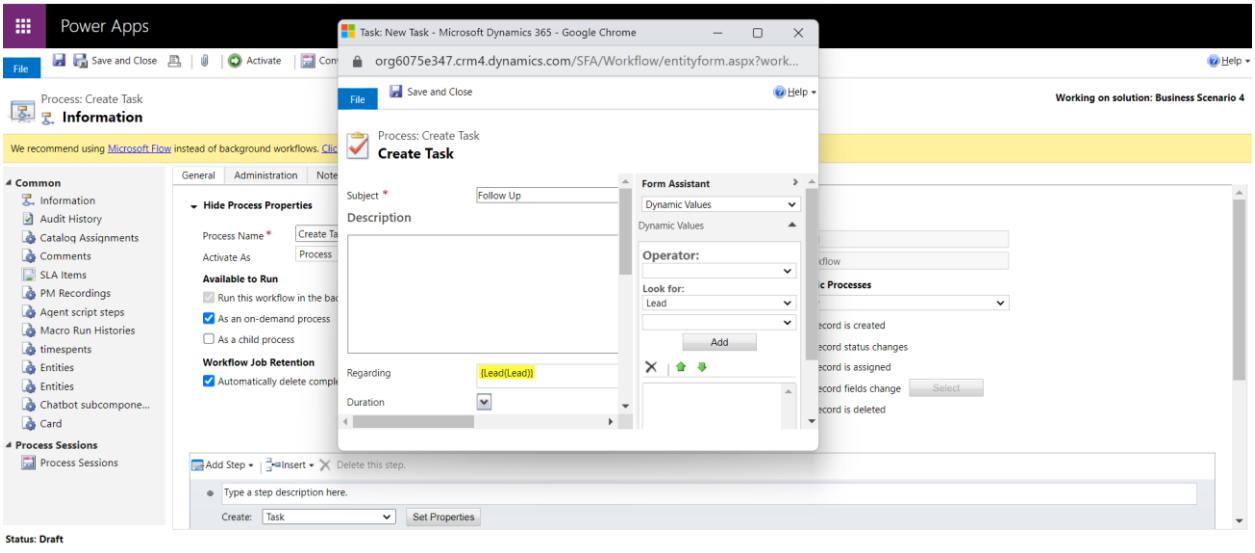
- **During the duplicate check process, I want to check for duplication based on the business phone column too.**

I've handled this by creating a Duplicate Detection Rule for the "Business Phone" field.



- **After the lead is created, I want a task to be created automatically and tied to the current lead, with the title 'Follow Up'.**

In this case, I've created an On-Demand Workflow Process that triggers upon the creation of a record in the "Lead" entity. The process has a step that creates a record in the "Task" entity with the subject "Follow Up," and the "Regarding" field (Lookup) is set to the Lead record that was just created.



The same operation can also be performed using a Plugin

```
string Topic = Target.GetAttributeValue<string>("subject");
Entity newTask = new Entity("task");

newTask["regardingobjectid"] = new EntityReference("lead", Target.Id);
newTask["subject"] = "Follow Up";
Target["subject"] = Topic +" "+ DateTime.UtcNow.ToString("dd/MM/yyyy");//Format to Input only the Date.
service.Update(Target);
//  service.Create(newTask);
```

What other types of customizations and extensions can be done in MSDyn365 and when would

you choose which?

Power Automate Process Automation:

- When to use:

- It is used when there is a need to automate various processes in Dynamics 365 or to integrate with other applications. Power Automate can be utilized to respond to different events, create automated workflows, and connect actions across multiple applications in a coordinated manner.

- Why to use:

- It allows users to create automation without the need for deep programming knowledge. Instead of writing code, users can use a simple interface to define workflows and automate actions.

- Enables integration with various services and applications outside of Dynamics 365, allowing organizations to create a connected ecosystem of applications.

- How to use:

- Users leverage the Power Automate interface to create workflows. Steps can include reacting to an event in Dynamics 365, sending an email, or executing an action in another application.

- Examples:

- When a record is created in Dynamics 365, a workflow can be triggered to notify users, send an email, or use the data in a third-party application.

- Integrating Dynamics 365 with a third-party service like Google Sheets for data synchronization between platforms.