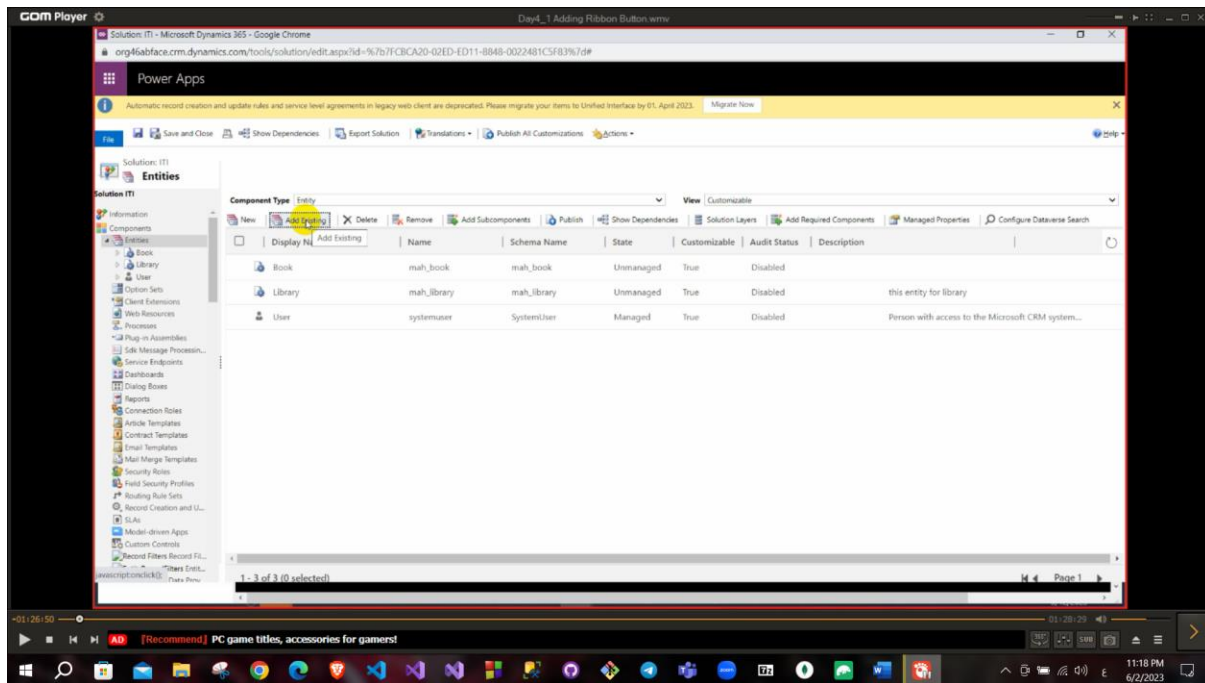
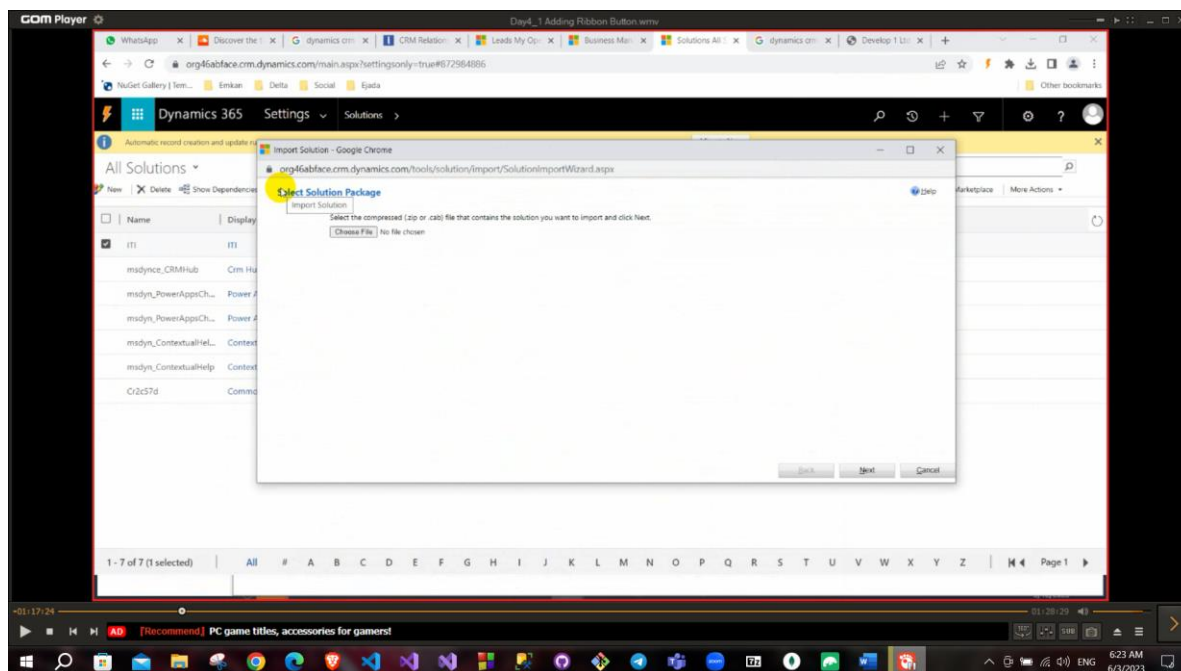


We can add existing entities like contact



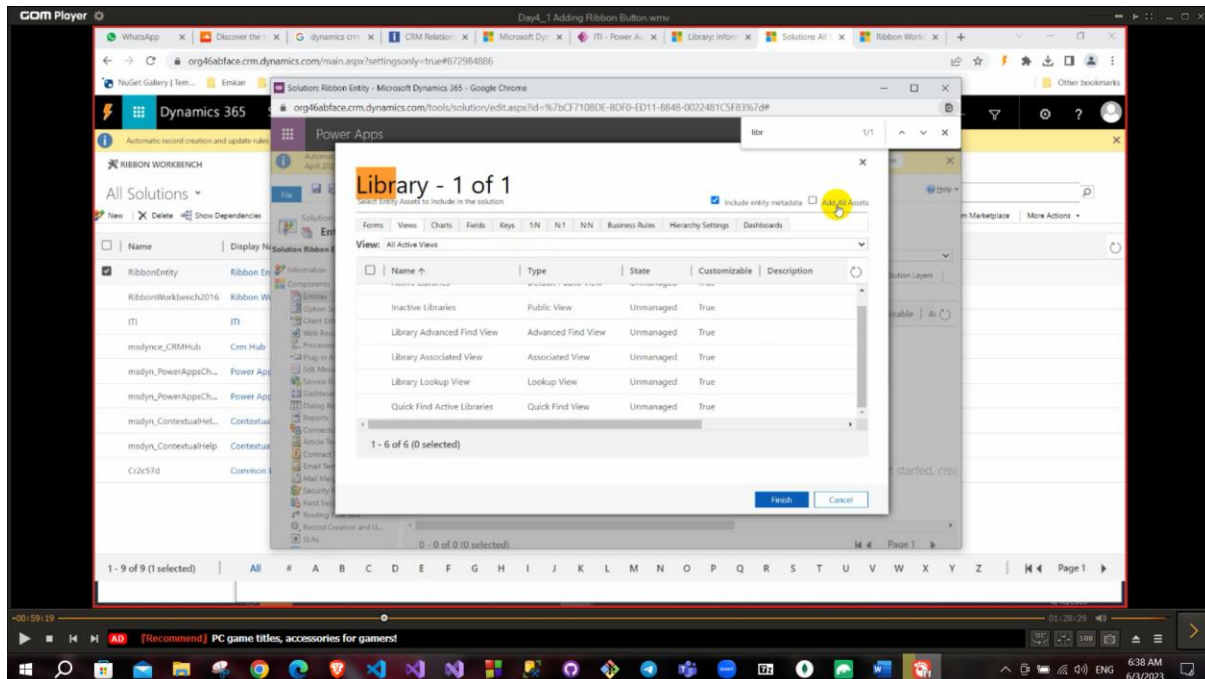
Client extension that to create or use a side menu or application ribbons. Also, you can add existing entity or edit on its fields. Buttons that added in ribbon will appear **Ribbon work bench** tool in customization. To install this tool import ZIP file in solution.



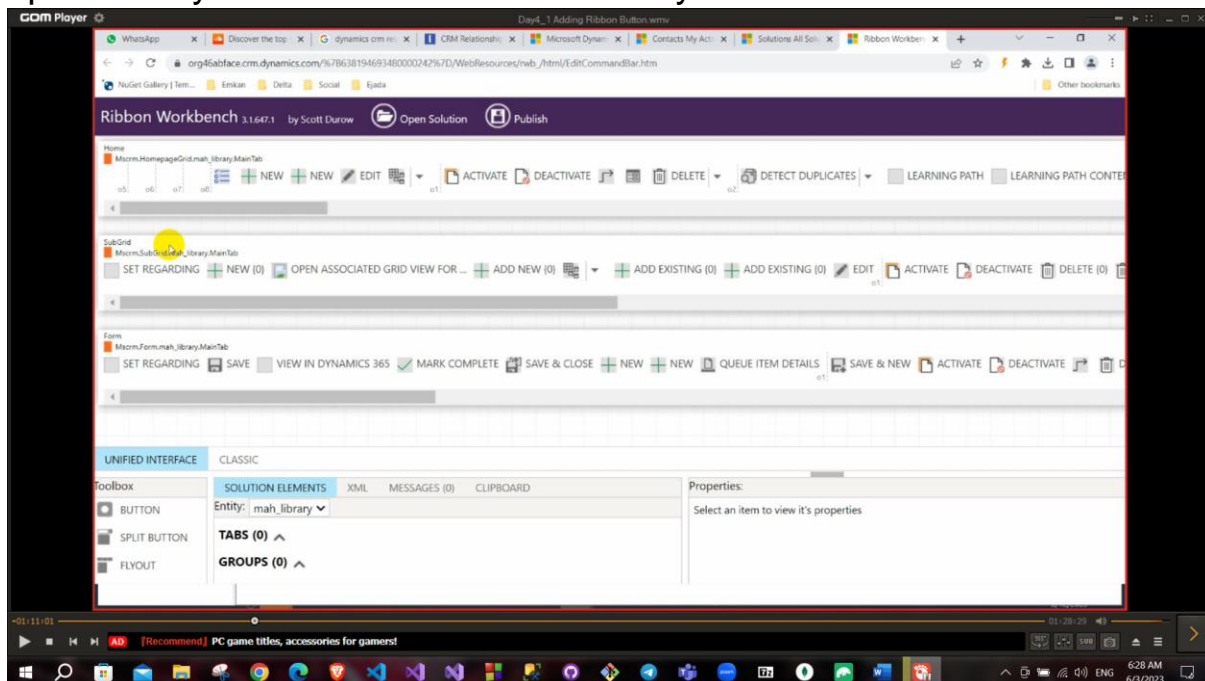
Managed Solution vs unmanaged solution.

To deal with this tool you will need to create a newsolution has a new entity.

To do that add an existing entity like library to the solution but uncheck add all assets.

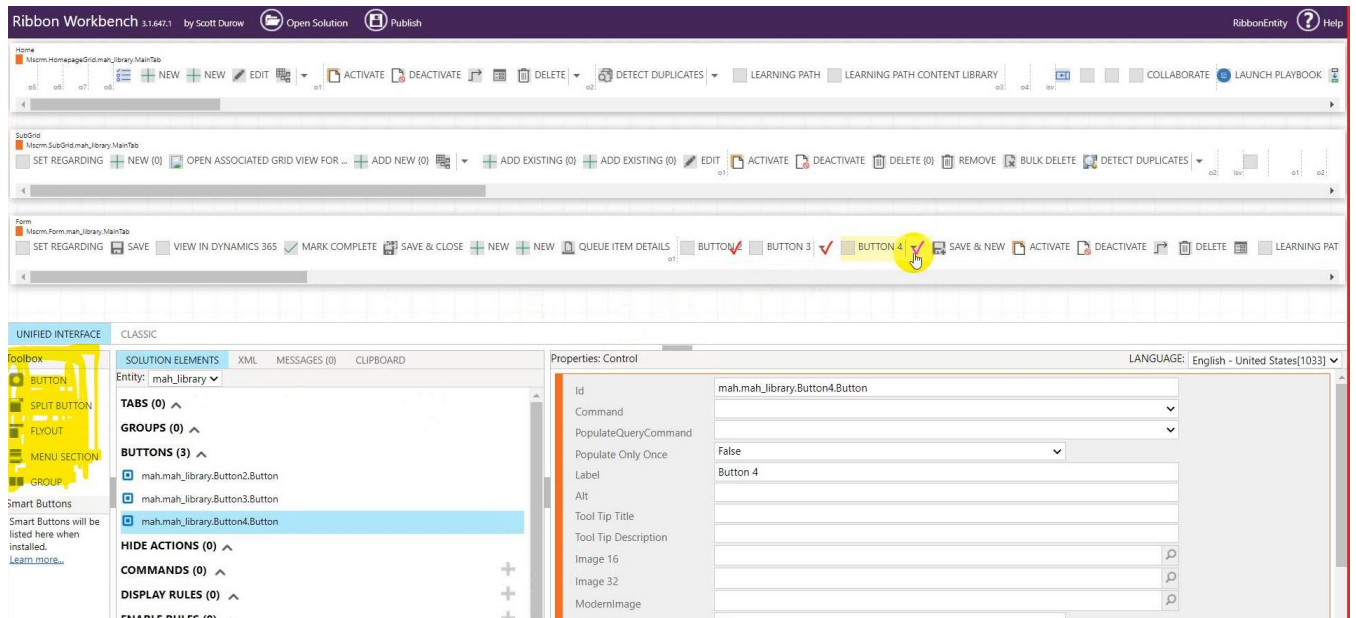


Now if you clicked on the ribbon workbench on solutions page this opens and you can choose the new entity



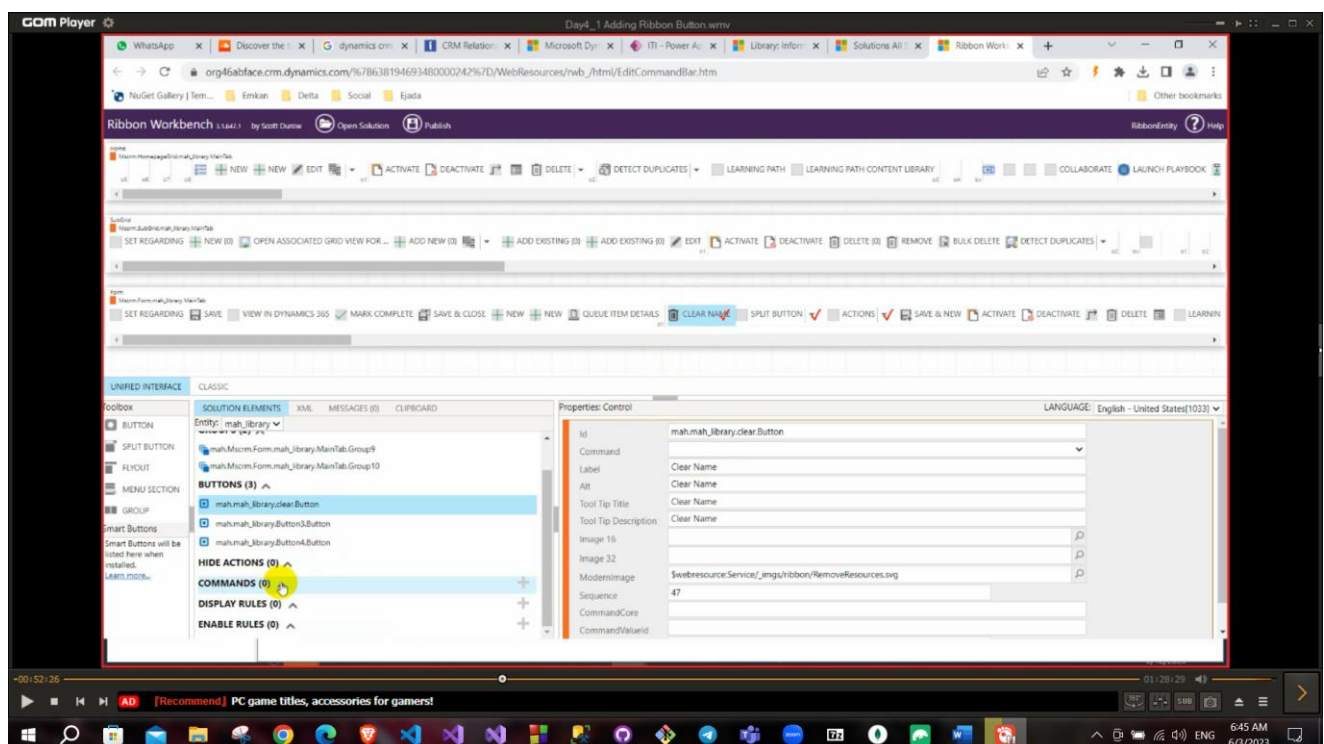
As you can see above Your button will appear on (view or Form or sub-grid)

Note: To add entity to area: right click on your app then click on **OPEN IN APP DESIGNER** button and add what you need.



From the below area you can choose the buttons from the left and add it to view or form or subgrid.

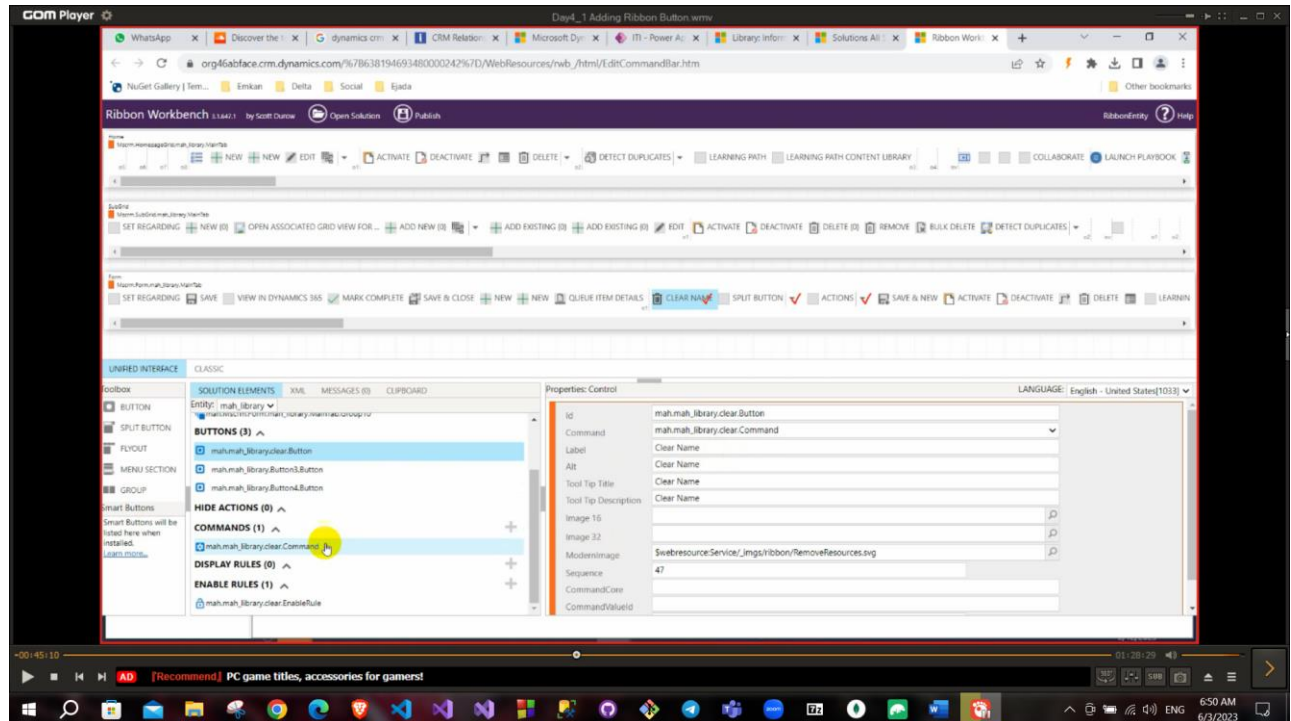
After adding the button you need to set command: command refer to action that this button will execute also, when this button will appear (all condition will be written in command) to create command click on command that existed in left part and click on add.



All command has:

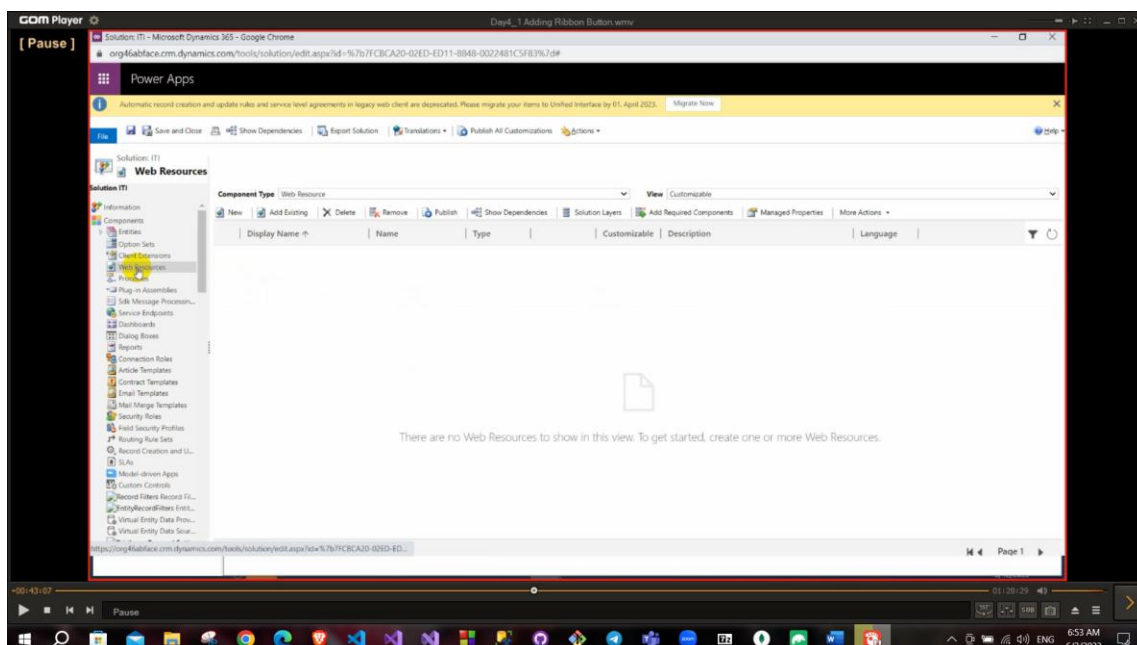
Action, display rule and enable rule. => م

For the button to work you should set the command field with the command created.

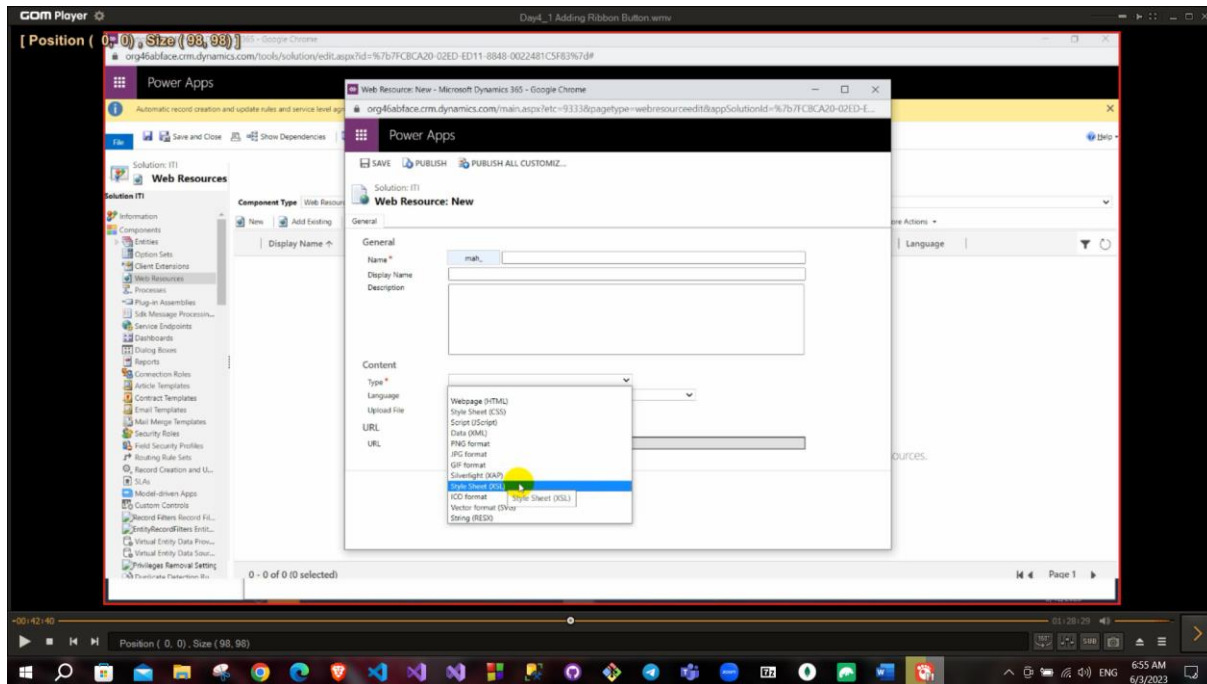


To link with a function with **enable rule** create a **JavaScript action**, And add to enable rules a **Custom rule**.

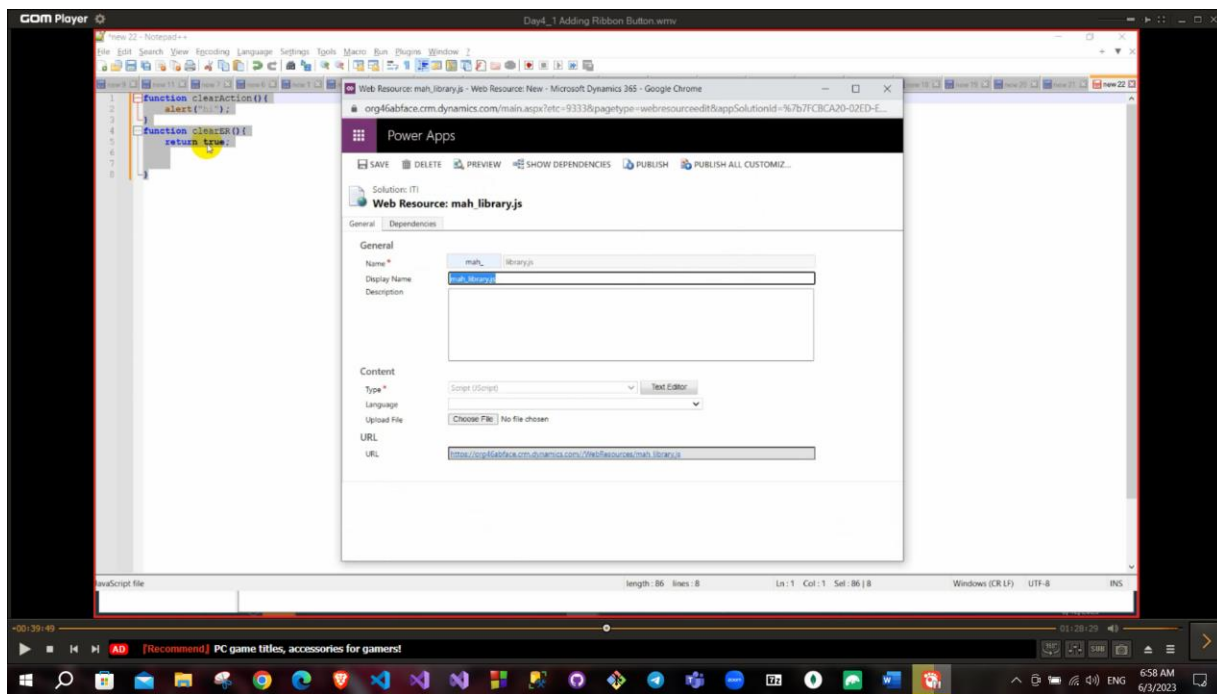
Web Resource: to add any file to your solution it like html, css, js, xml, svg, img,

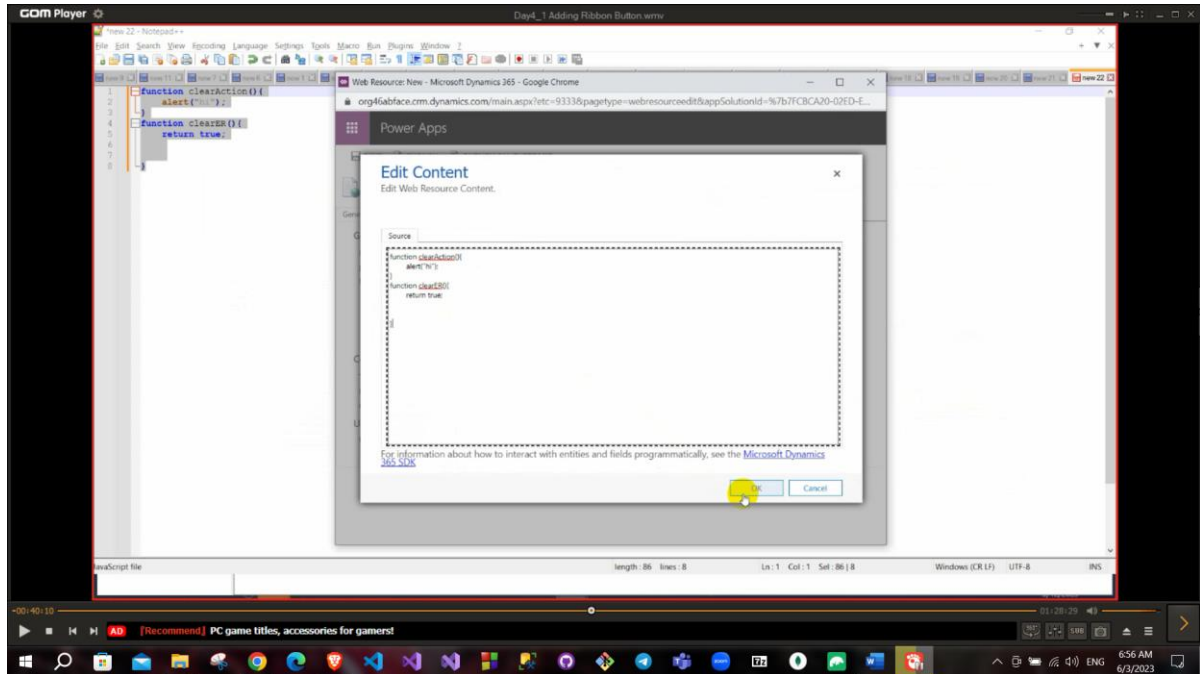


So, to write JS functions create a JS file in web resource.

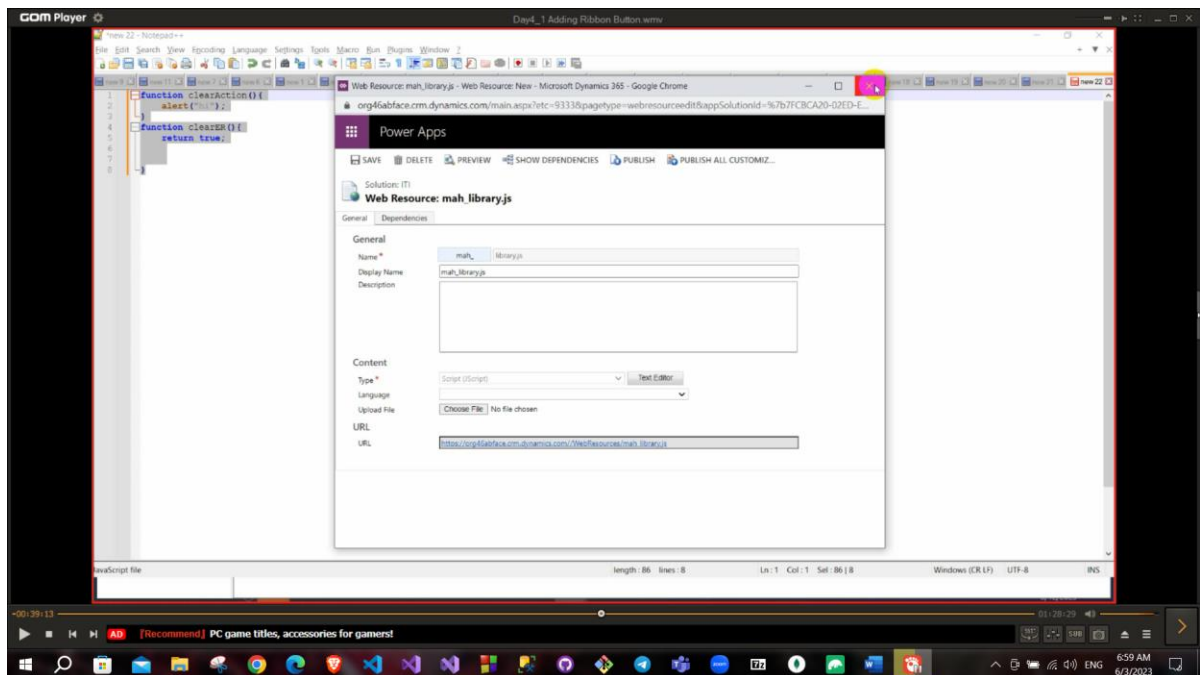


Choose js and then click on the text editor to add your js functions

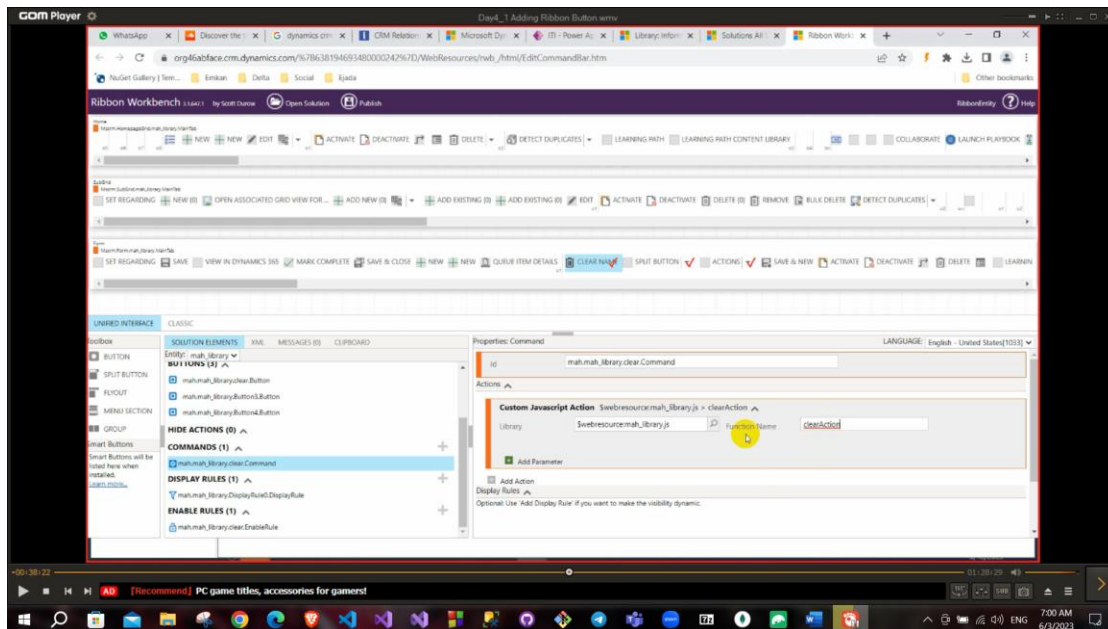




Then click on publish



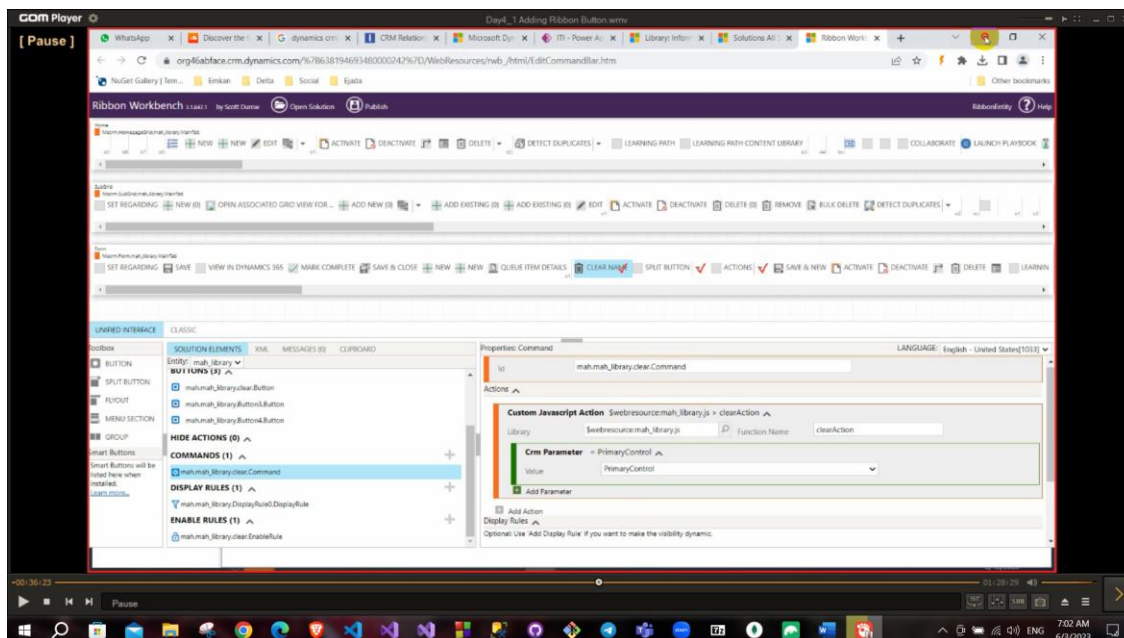
Then go to the button and add the function to the command,
Choose the library you added and the function name



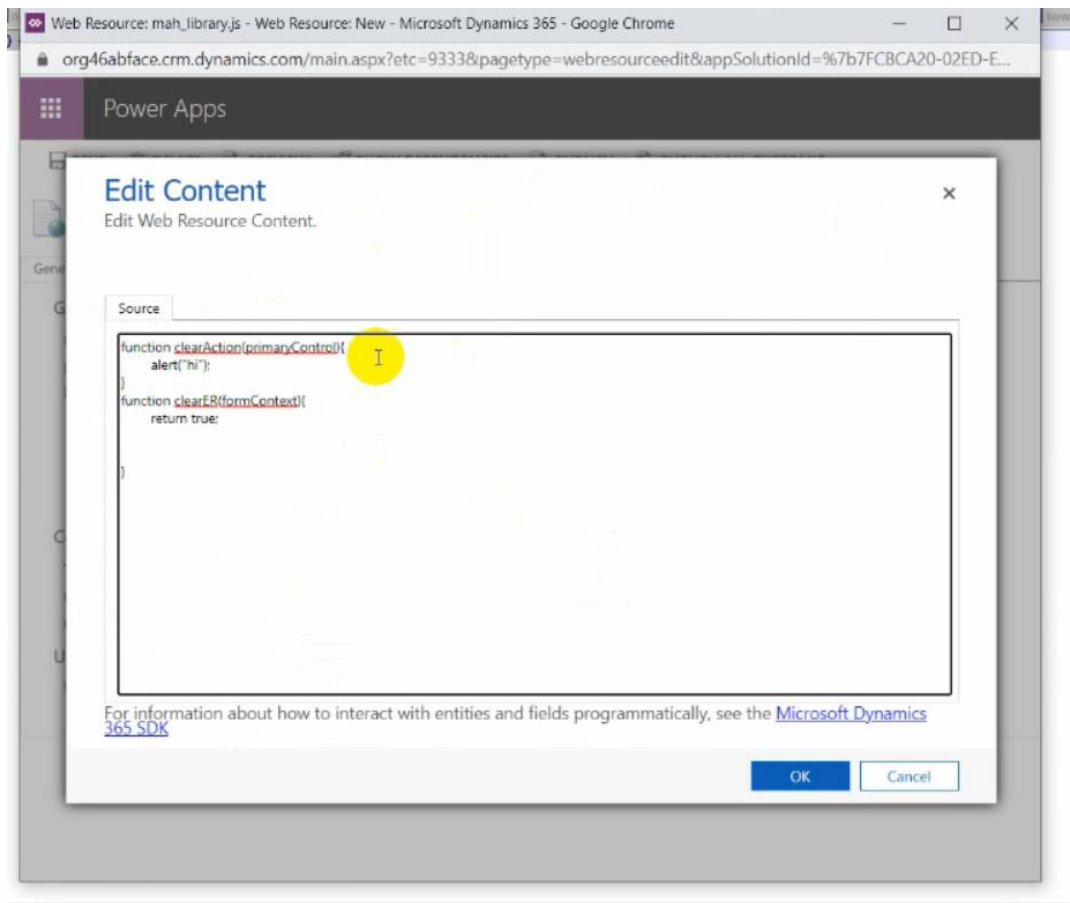
you can add parameter to functions and these are data types which you can to pass.

CRM parameter: to pass a related parameter to your entity (like passing entity name, or passing status...)

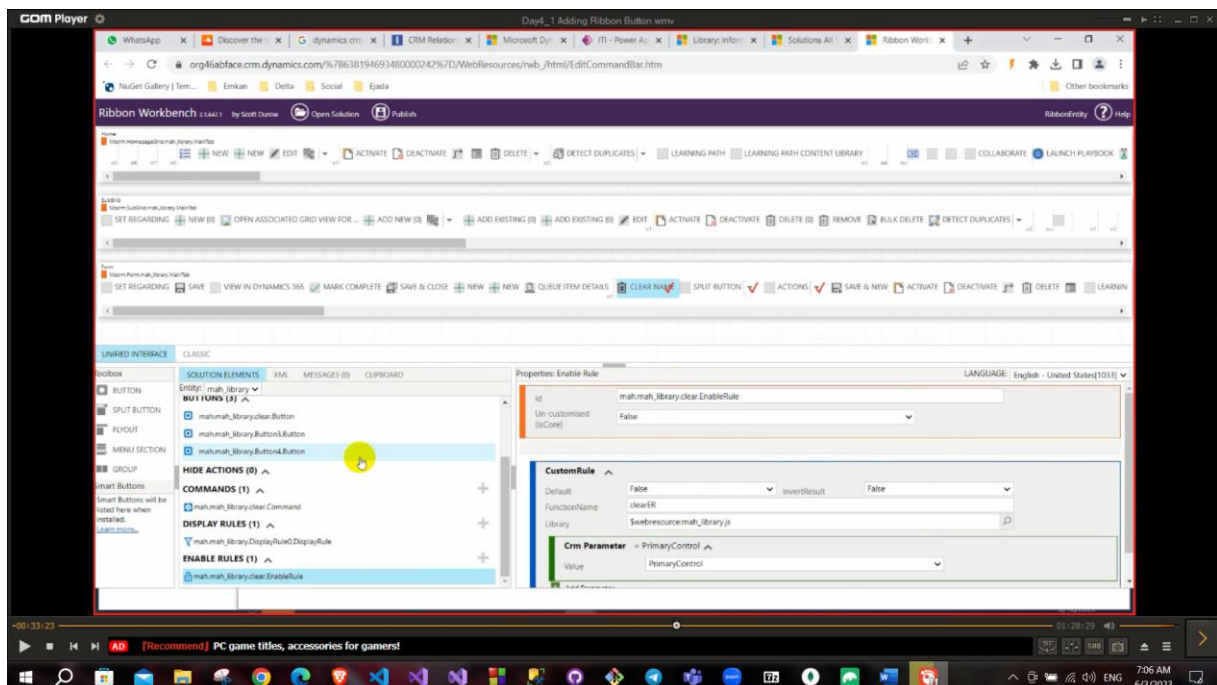
There is a very important parameter called “**formContext**” or “**primaryControl**” that like window object in JS (like context of all data).



And add these parameters to your functions



And this is the enable rule function



****You can customize any out of the box buttons.**

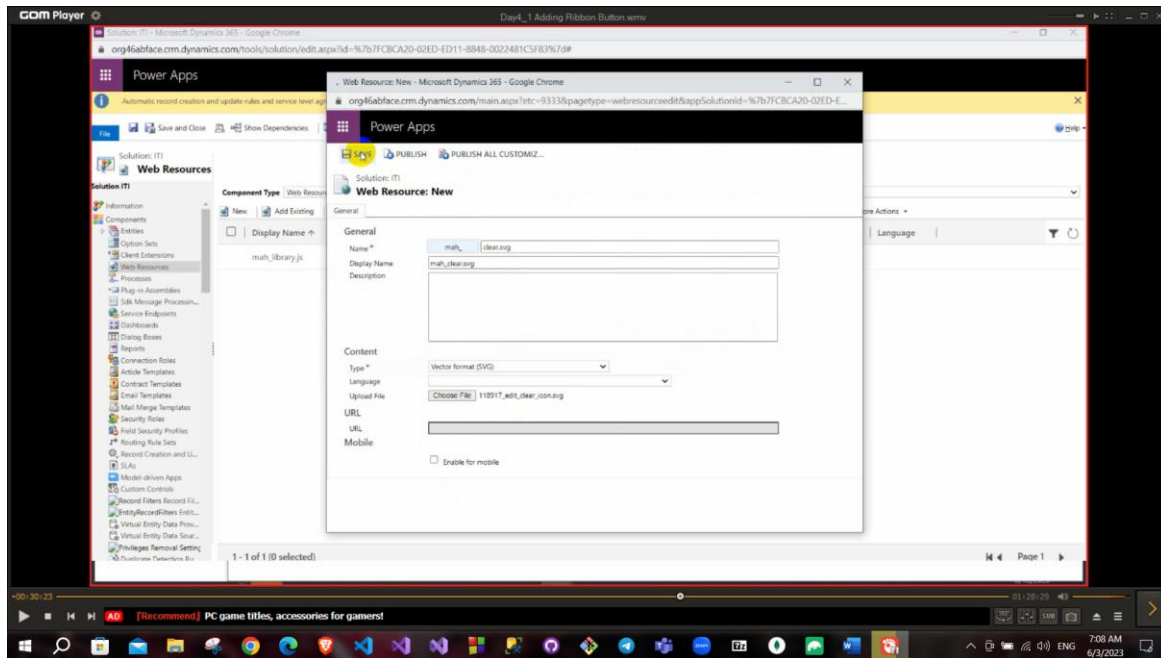
****Enable rule function return Boolean and you can negate it.**

Both enable rule and display rule control if the button will be showed or not. Display rule run at the first and enable rule run in client side.

You can add more than one action to button.

****Save, Active and publish if exist it will be important to activation.**

IconFinder is a website to download icons after downloading icon as SVGupload it on web resource.



****Any required came: Search in out of the box in CRM → Customization (add field or edit...) → Client-side script → Plugins or customs or workflows**

To get element value from primaryControl:

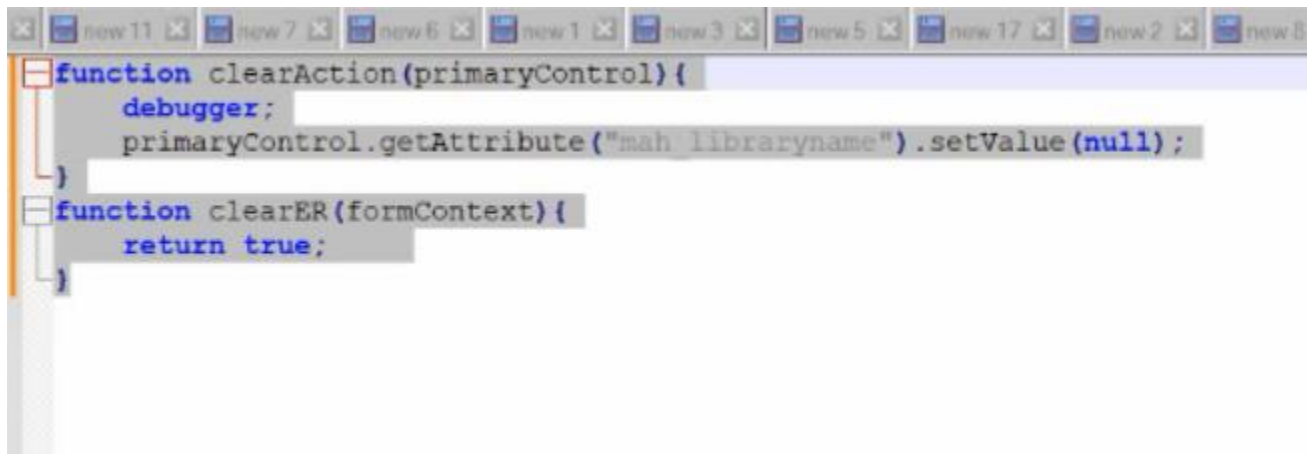
```
primaryControl.getAttribute("attributeName").getValue()
```

```
primaryControl.getAttribute("attributeName").setValue("newValue")
```

Attribute name: To get value like contact name there are two names of these field first one is display name that for display only and does not exist in HTML and another one called name this dealing in background (HTML)

To know Name attribute you can get it from inspect or from entity details or from external tool in google chrome called **Dynamics 365 power pane** or **Level up for dynamics 365/Power Apps**.

If I want the button to clear the library name on click



Remember to get any data "primaryControl" like context.

If you get lookup from context, it will return an array of objects.

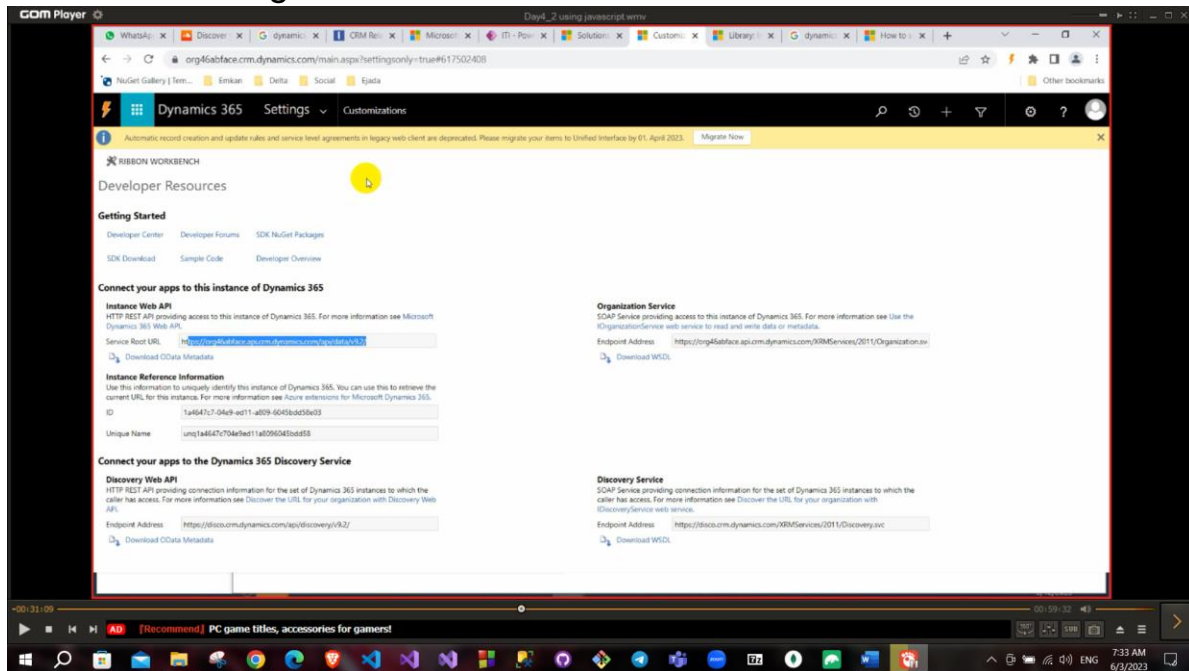
This object has {**entityType**, **id**, **name**} (lookup).

Here is sample code how to set the lookup value using JavaScript Dynamics CRM.

```
var lookupValue = new Array();
lookupValue[0] = new Object();
lookupValue[0].id = "{727504ed-64c5-4bc8-ac22-0a3071c427e3}"; // GUID of the lookup id
lookupValue[0].name = "Goutam Das"; // Name of the lookup
lookupValue[0].entityType = "contact"; //Entity Type of the lookup entity
Xrm.Page.getAttribute("FieldName").setValue(lookupValue); // You need to replace the lookup field Name..
```

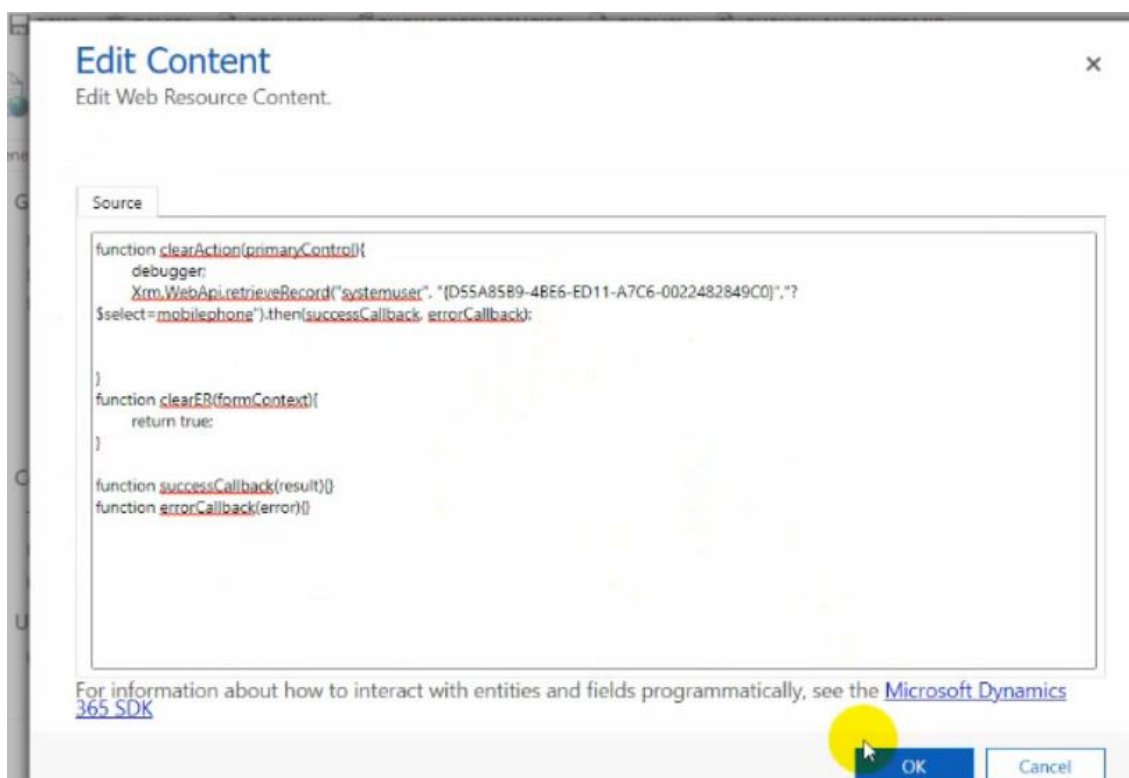
To retrieve data from CRM, like to retrieve mobile phone of a contact (in this case you need to call API because you will retrieve data from outside your page).

Use the api link given in the developer resources in customization in advanced settings



In previous case, you will consume API “Xrm.WebApi.retrieveRecord()”

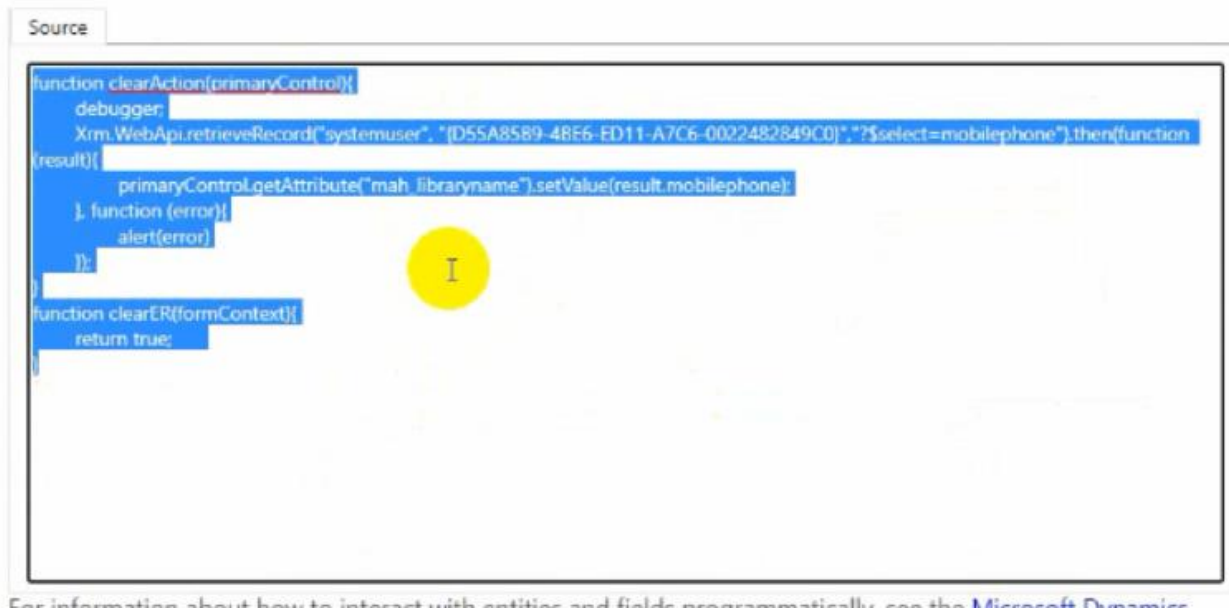
This function will return a promise so, you need to define success function and error function.



Using retrieveRecord

```
function clearAction(primaryControl) {  
    debugger;  
    Xrm.WebApi.retrieveRecord("systemuser", "{D55A85B9-4BE6-ED11-A7C6-0022482849C0}", "?$select=mobilephone").then(function (result) {  
        primaryControl.getAttribute("mah_libraryname").setValue(result.mobilephone);  
    }, function (error) {  
        alert(error);  
    });  
}  
function clearER(formContext) {  
    return true;  
}
```

To save your record “primaryControl.data.entity.save()”.



```
function clearAction(primaryControl) {  
    debugger;  
    Xrm.WebApi.retrieveRecord("systemuser", "{D55A85B9-4BE6-ED11-A7C6-0022482849C0}", "?$select=mobilephone").then(function (result) {  
        primaryControl.getAttribute("mah_libraryname").setValue(result.mobilephone);  
    }, function (error) {  
        alert(error);  
    });  
}  
function clearER(formContext) {  
    return true;  
}
```

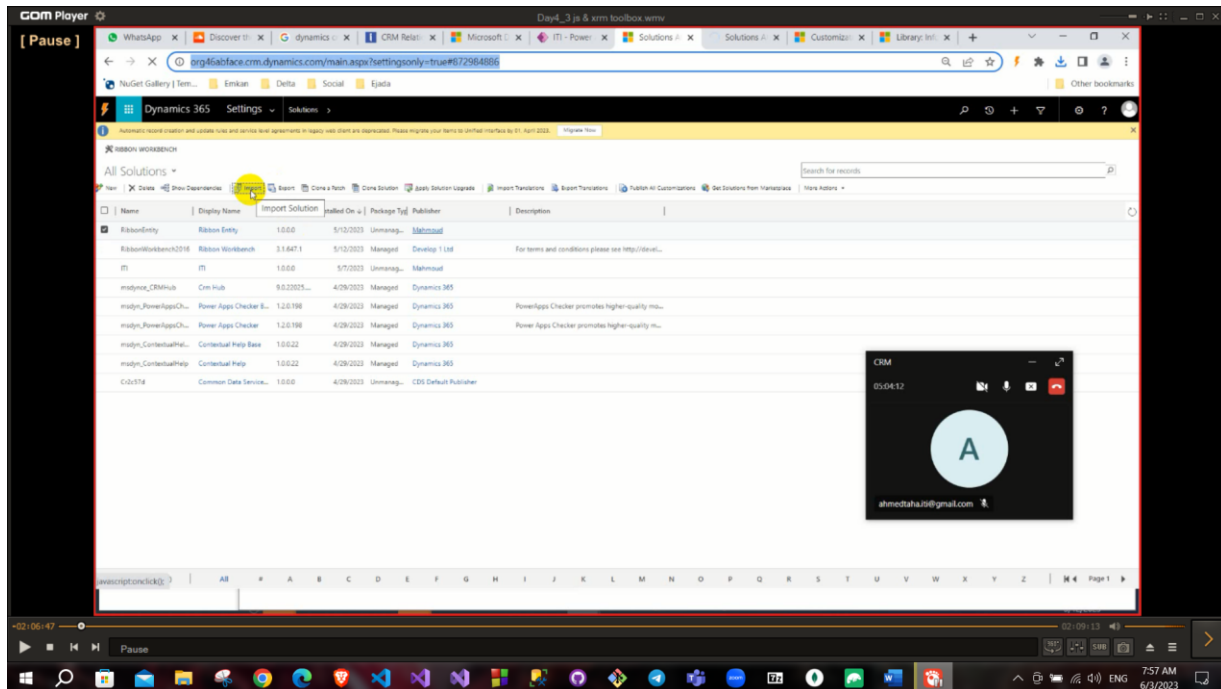
**To do anything in current record (primaryControl).

**To do anything outside this record call webApi through Xrm.

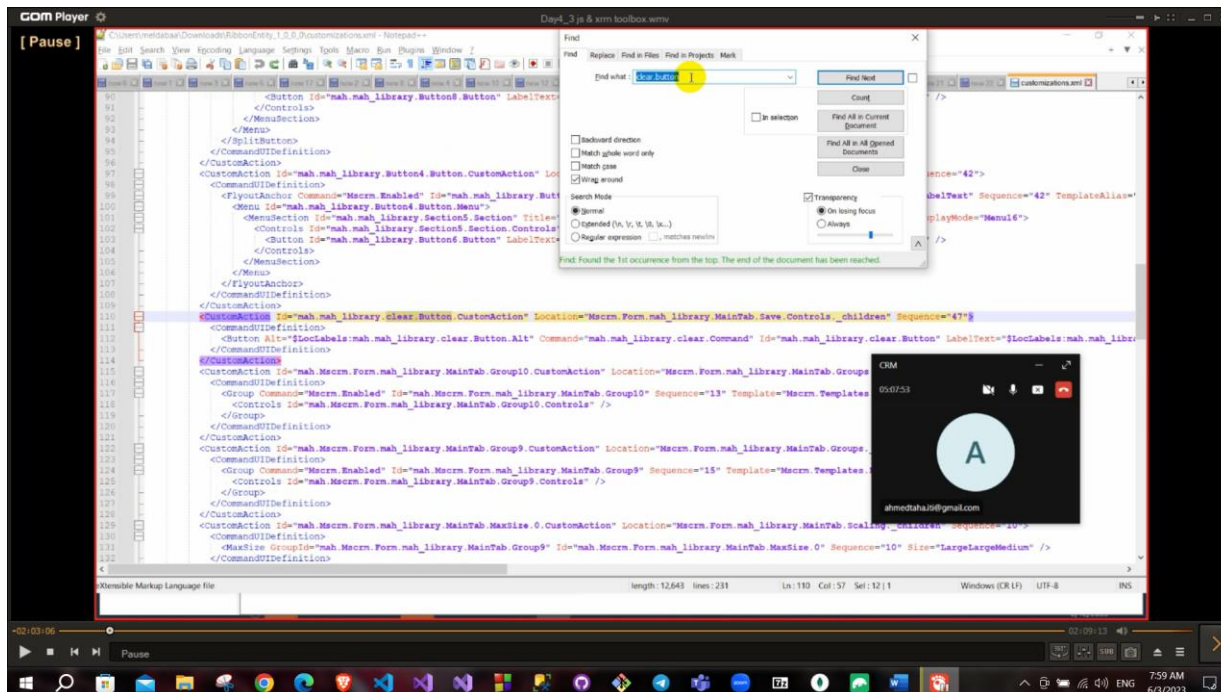
formContext.getFormType() if returned 1 Create, if 2 Update.

This previous line to check if this record is in creation time or modification time.

All edition in ribbon will transform to XML so you can export solution to XML and edit and customize as you can and import this file again but you need to compress this file .ZIP



And you can find the entities and ribbons in this solution



Here you can find the command we added on the button



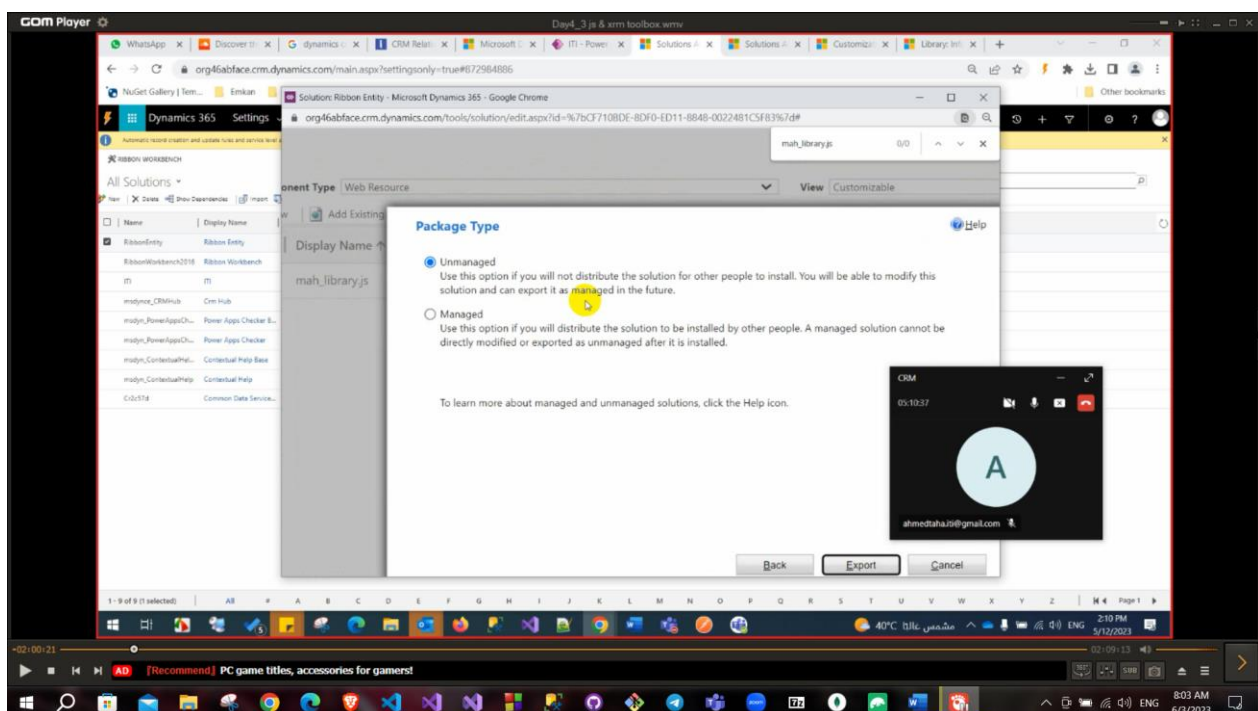
```
<Templates>
<CommandDefinitions>
<CommandDefinition Id="mah.mah_library.clear.Command">
  <EnableRules>
    <EnableRule Id="mah.mah_library.clear.EnableRule" />
  </EnableRules>
  <DisplayRules />
  <Actions>
    <JavaScriptFunction FunctionName="clearAction" Library="$webresource:mah_library.js">
      <CrmpParameter Value="PrimaryControl" />
    </JavaScriptFunction>
  </Actions>
</CommandDefinition>
</CommandDefinitions>
<RuleDefinitions>
<TabDisplayRules />
<DisplayRules />
<EnableRules>
  <EnableRule Id="mah.mah_library.clear.EnableRule">
    <CustomRule FunctionName="clearER" Library="$webresource:mah_library.js" Default="false" InvertResult="false">
      <CrmpParameter Value="PrimaryControl" />
    </CustomRule>
  </EnableRule>
</EnableRules>
</RuleDefinitions>
<LocLabels>
<LocLabel Id="mah.mah_library.Button3.Button.LabelText">
  <Titles>
    <Title description="Split Button" languagecode="1033" />
  </Titles>
</LocLabel>
```

Unmanaged vs managed solution

Unmanaged you can directly modify in this solution but you cannot in managed solution.

Also, in managed solution if you delete this managed solution will delete all entities and customization on it, on other hand in unmanaged solution if you delete this solution included entities will remain and do not be removed.

We use unmanaged in development time, managed for production time.



Using retrieveMultipleRecord

“Xrm.WebApi.retrieveMultipleRecord(entityLogicalName, options, maxPage size)”

```
JavaScript Copy  
  
Xrm.WebApi.retrieveMultipleRecords("account", "?$select=name&$top=3").then(  
    function success(result) {  
        for (var i = 0; i < result.entities.length; i++) {  
            console.log(result.entities[i]);  
        }  
        // perform additional operations on retrieved records  
    },  
    function (error) {  
        console.log(error.message);  
        // handle error conditions  
    }  
);
```

option is to customize your selection so can filter your result.

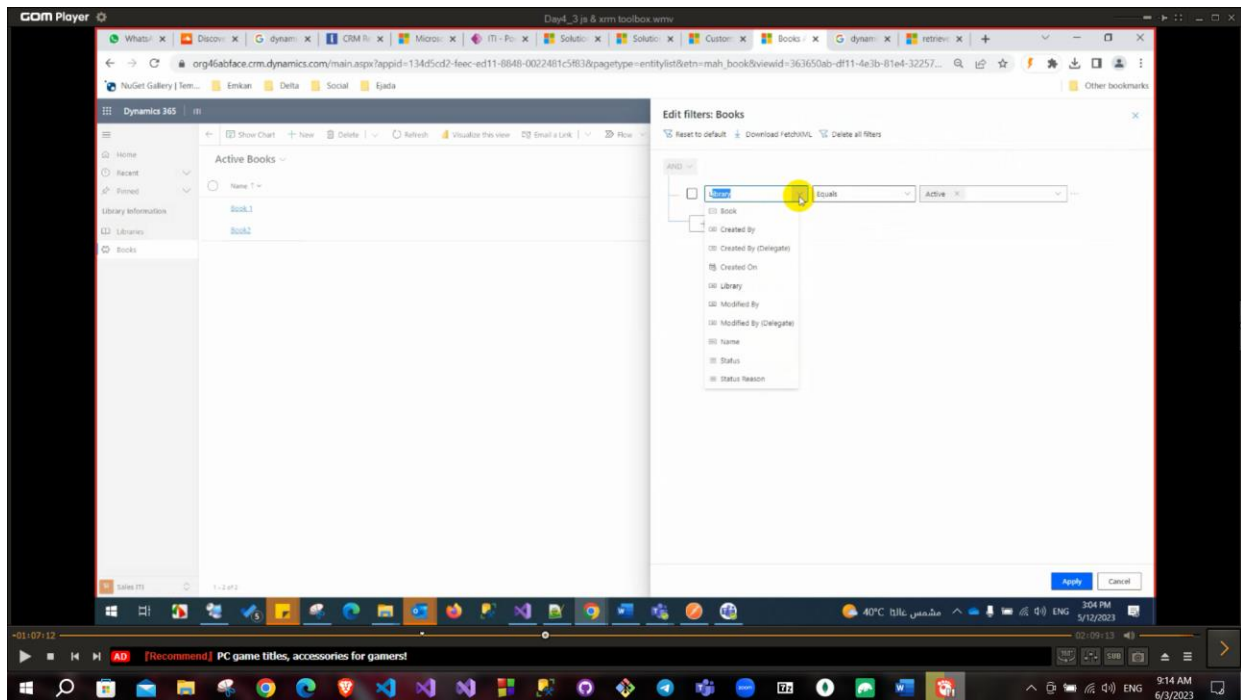
But this option is difficult to build so we use advanced filter and export it as XML and put this XML to your function.

Basic retrieve multiple with FetchXML

This example queries the `account` entity using fetchXML.

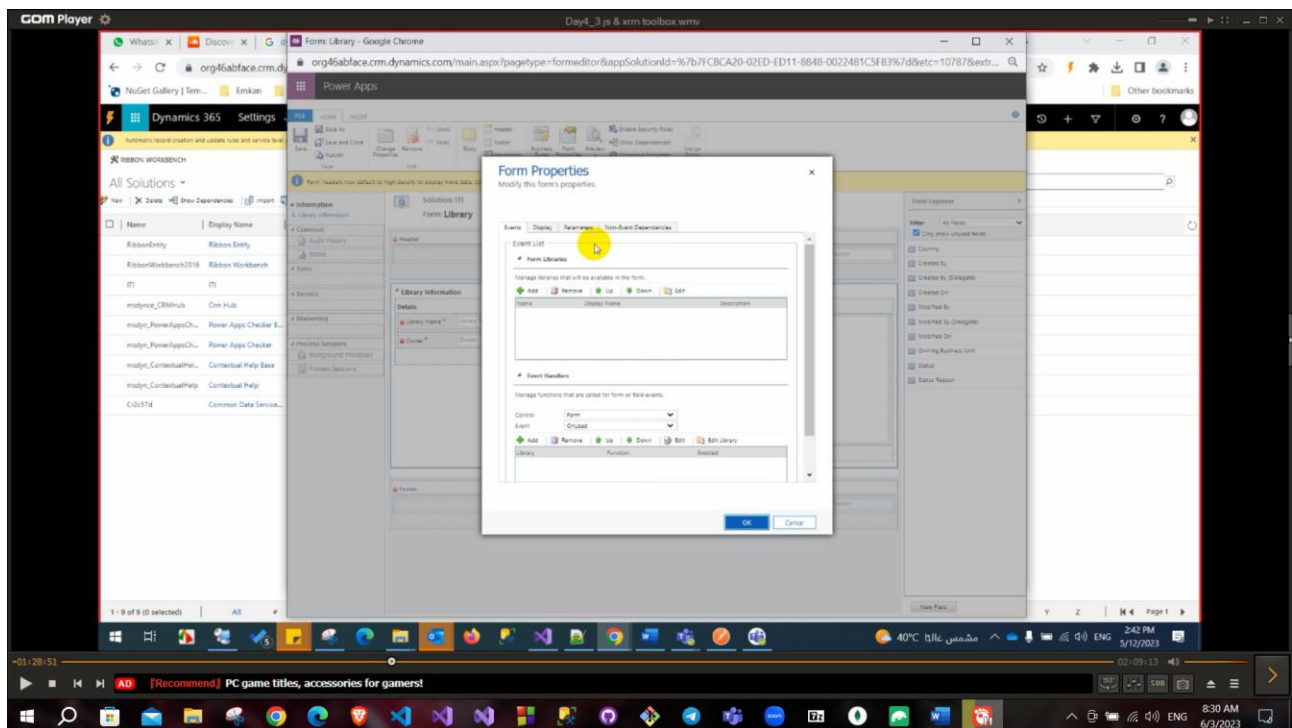
```
JavaScript Copy  
  
var fetchXml = "<fetch mapping='logical'><entity name='account'><attribute name='"  
  
Xrm.WebApi.retrieveMultipleRecords("account", fetchXml).then(  
    function success(result) {  
        for (var i = 0; i < result.entities.length; i++) {  
            console.log(result.entities[i]);  
        }  
        // perform additional operations on retrieved records  
    },  
    function (error) {  
        console.log(error.message);  
        // handle error conditions  
    }  
);
```

You can get the fetchXml from edit filter just choose the entity



There are two events on form (onSave, onLoad).

To access these events from open form from entity → Form properties → add your JS library name



In this event you need to pass “executionContext” as parameter.

executionContext.getForContext() will return formContext that have all data of this form.

In button action we can you primaryControl directly because his form has this object.

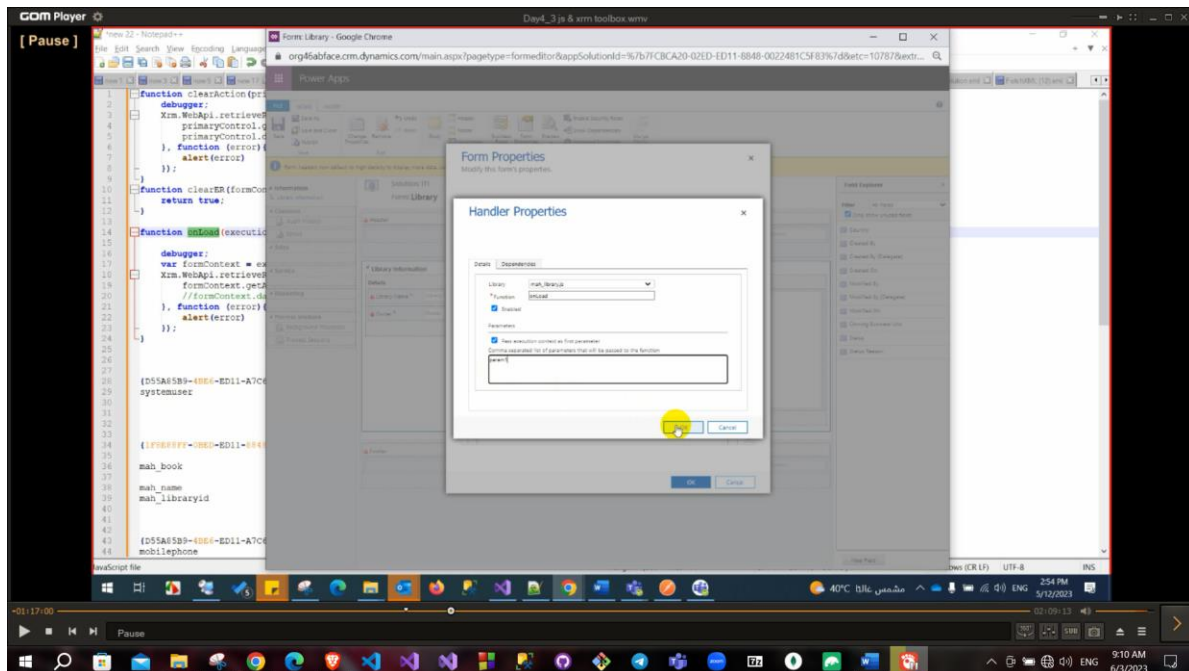
But in events we need to use executionContext.getForContext()

var formContext = executionContext.getForContext().

To get our context of this form.

```
function onLoad(executionContext){  
    debugger;  
    var formContext = executionContext.getFormContext();  
    Xrm.WebApi.retrieveRecord("systemuser", "{D55A05B9-4BE6-ED11-A7C6-0022482849C0}", "?$select=mobilephone").then(function (result){  
        formContext.getAttribute("mah_libraryname").setValue(result.mobilephone);  
        formContext.data.entity.save()  
    }, function (error){  
        alert(error)  
    });  
}
```

And you can pass parameters



We can also use retrieve multiple inside onload

```
function onLoad(executionContext){
    //alert();
    debugger;
    var formContext = executionContext.getFormContext();
    var recordId = formContext.data.entity.getId();
    //recordId = recordId.replace(/[\{\}]/g, "");
    var fetchXml = "<?fetch version='1.0' mapping='logical' no-lock='false' distinct='true'>\n
    <entity name='mah_book'><attribute name='mah_bookid'><attribute name='mah_name'>\n
    <attribute name='Createdon'><order attribute='mah_name' descending='false'>\n
    <filter type='and'><condition attribute='mah_libraryid' operator='eq' value='"+recordId+"' uitype='mah_library'>\n
    </filter>\n
    </entity>\n
    </fetch>";
    Xrm.WebApi.retrieveMultipleRecords("mah_book", fetchXml).then(
    function success(result) {
        for (var i = 0; i < result.entities.length; i++) {
            console.log(result.entities[i]);
        }
        // perform additional operations on retrieved records
    },
    function (error) {
        console.log(error.message);
        // handle error conditions
    }
    );
}
```

To create a new record Xrm.WebApi.createRecord(entityLogicalName, data)

This data will be as JSON file

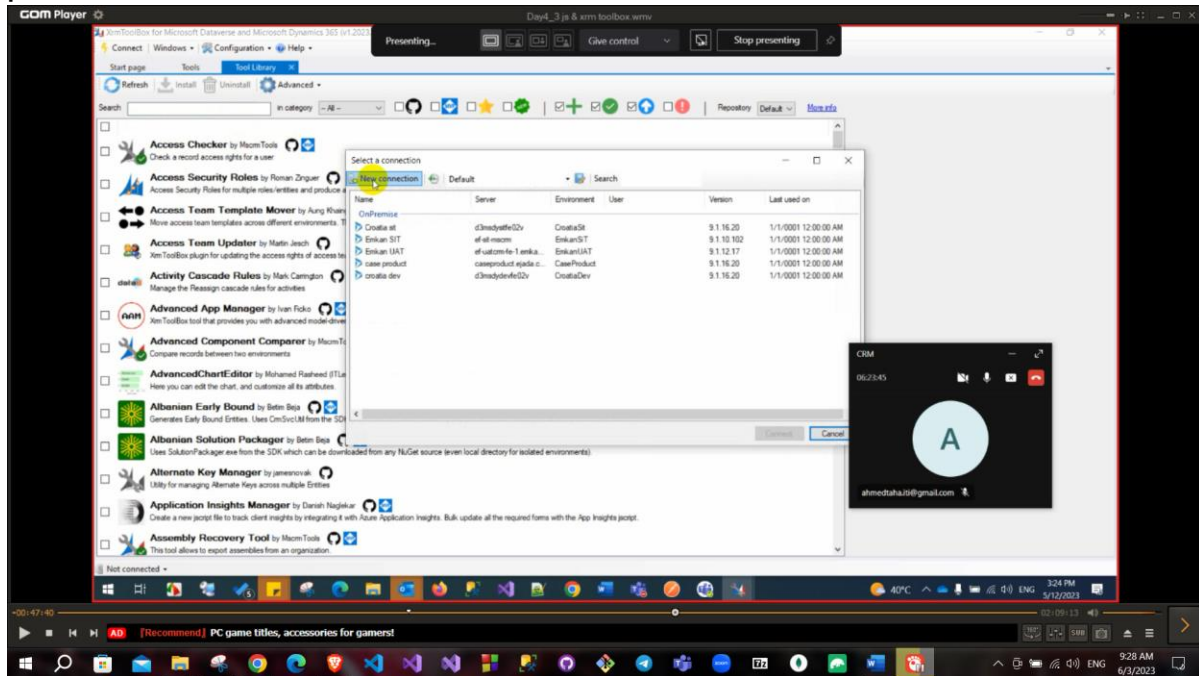
```
// define the data to create new account
var data =
{
    "name": "Sample Account",
    "credithold": false,
    "address1_latitude": 47.639583,
    "description": "This is the description of the sample account",
    "revenue": 5000000,
    "accountcategorycode": 1
}

// create account record
Xrm.WebApi.createRecord("account", data).then(
    function success(result) {
        console.log("Account created with ID: " + result.id);
        // perform operations on record creation
    },
    function (error) {
        console.log(error.message);
        // handle error conditions
    }
);
```

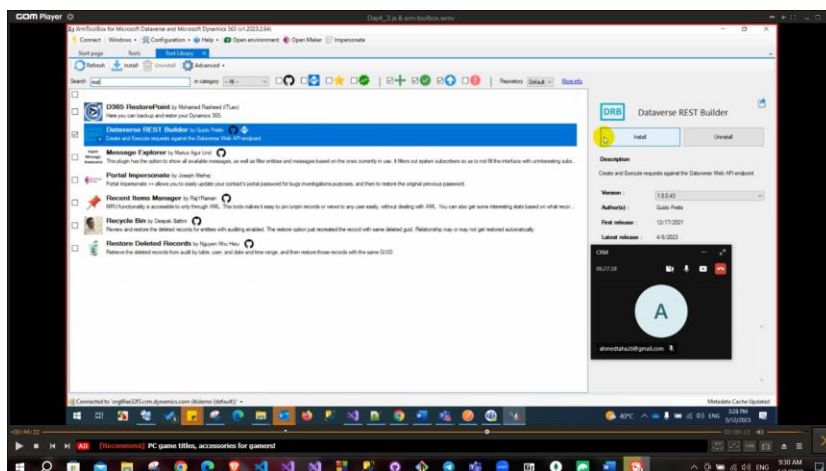
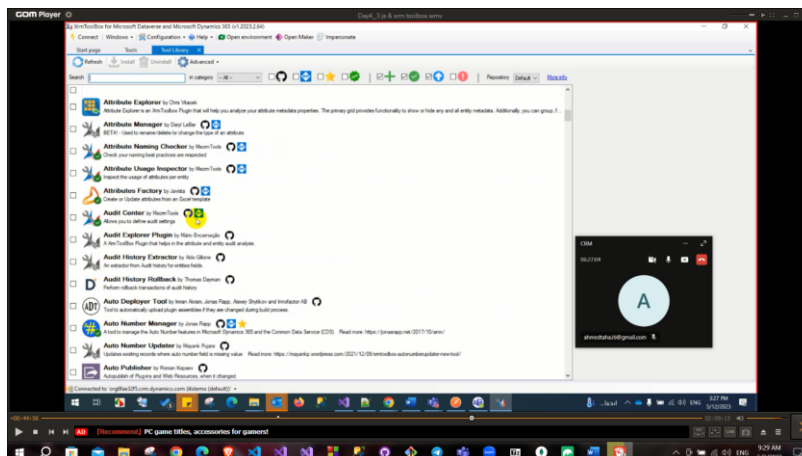
Update like create but need to pass id of this record too.

XrmToolBox is a very important tool

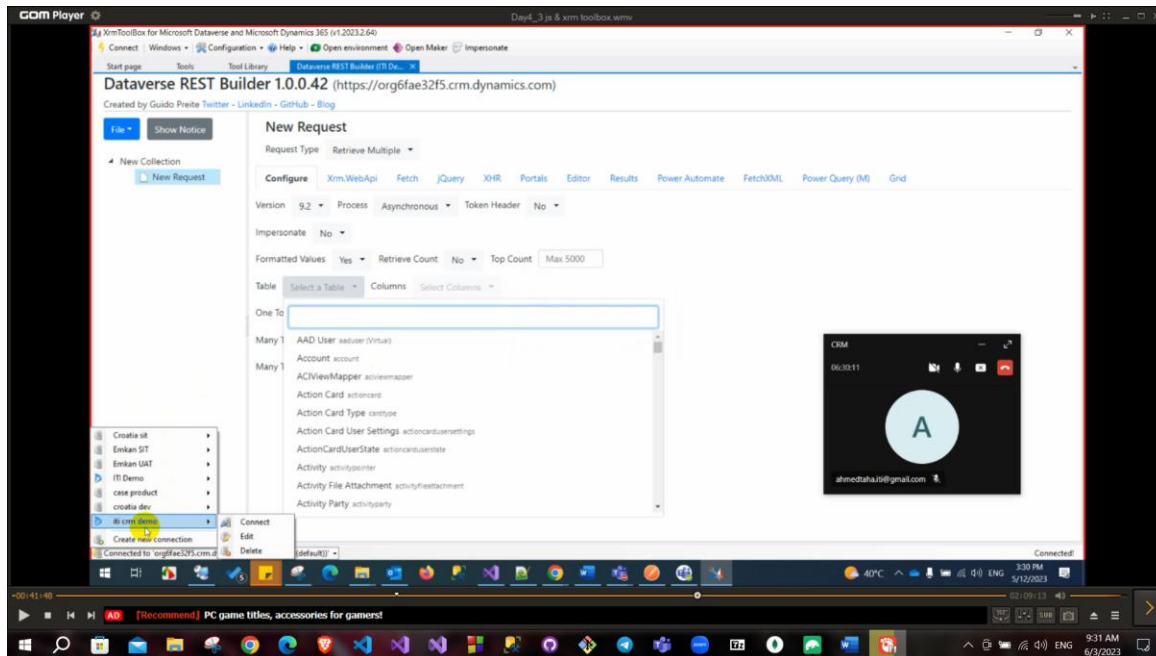
First step to connect on your environment with environment name and password.



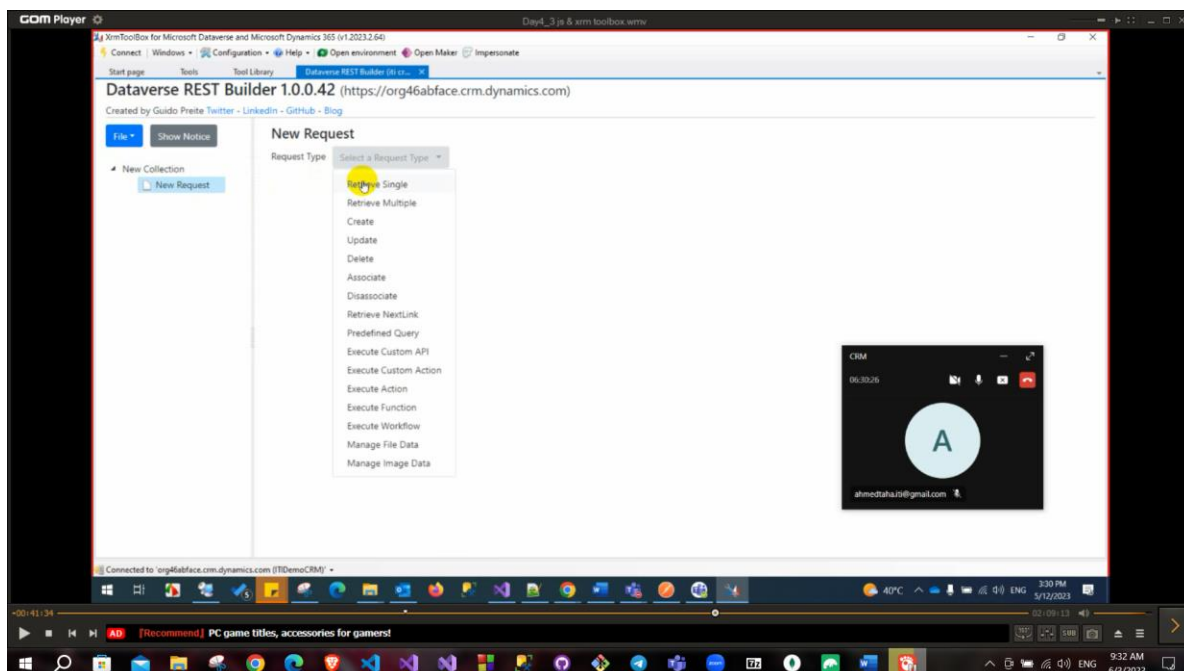
This tool has many libraries or plugins to use like Dataverse Rest Builder



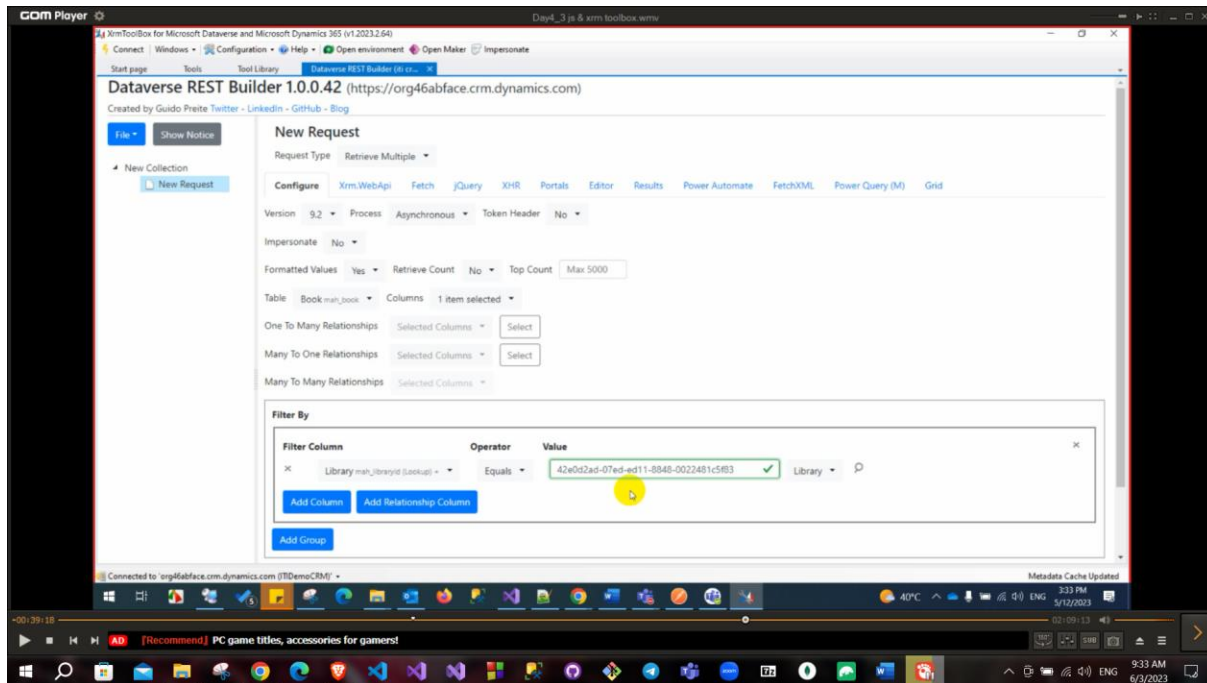
This library will build your needed request for webAPI and you can customize your result and filter is as you want also you can set it as synchronous or asynchronous



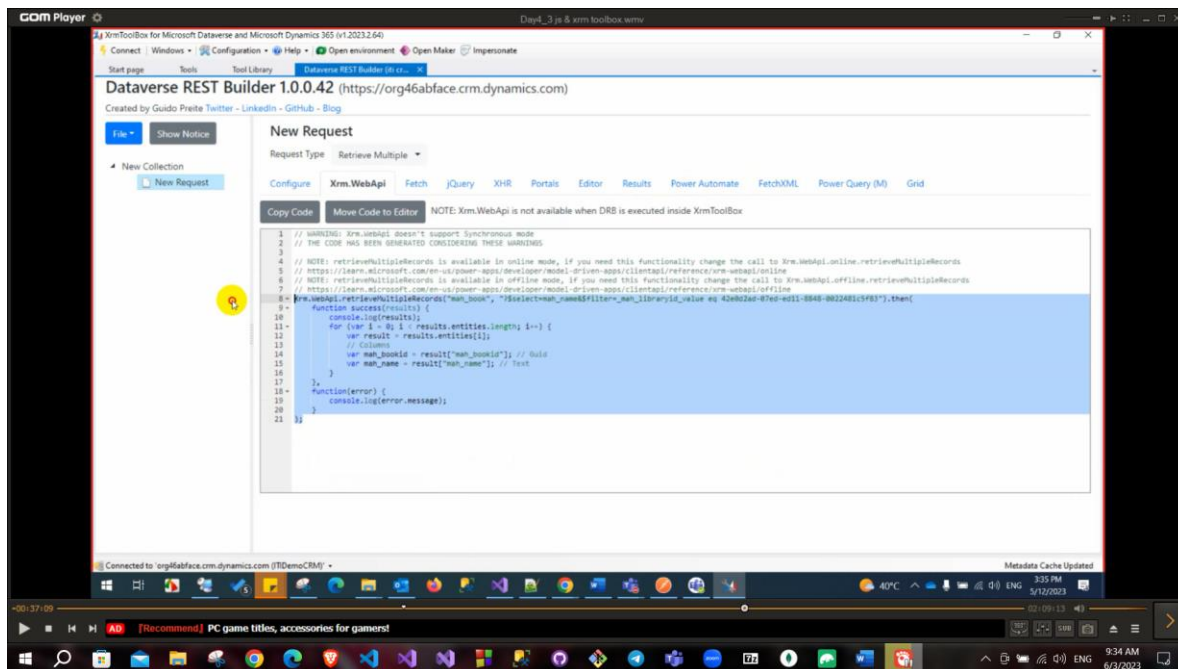
You can choose what you want



Choose the request and the entity and select what you want



And go to xrm web api tab and you will find the code



Very important question difference among relationship behavior

1. Parental (As parent any update or delete child will be affected and deleted or assign or share will **cascade in all cases**).
2. Referential (in this case child will not be affected if his parent has been deleted or updated **will not cascade anything**).
3. Referential restrict delete (in this case child will cascade for all actions of his parent except deletion will return error message will **cascade all cases except deletion**).
4. Configurable cascading (you will customize your actions in case of delete or assign or share).

There are many tools in Xrmtoolbox to ease your usage of dynamics

To create entity → advanced settings → customization → solutions →

Entity → add field → add this field to entity → add this field to view

To create JS function → add this function to web Resource → add this function name to your button.

Some question of interviews

- Access levels
- Relationship types (1-1, 1-N, N-1)
- Relationship Behavior (Parental, Referential, Referential restrict delete, Configurable cascading)
- To get data related to your current record (Context)
- To get data from CRM (webAPI)

Context usage: to update field, to lock field, get id of this record, clear field

CRM usage: All CRUD operation on data update, delete, retrieve, retrieveMultiple, create and execute

Button: has command (action (JS), enable rule (JS) and display rule).

Using ribbon workbench