SharePoint 2013: Working with REST API using jQuery Ajax

Table of Contents

Introduction

This article describes how to work with SharePoint list items, basically performing CRUD operations, using the combination of REST API and jQuery Ajax. Working with REST API or JQuery comes handy while working with SharePoint Apps or creating Custom Forms which involves Client Side Code.

Scenario

Let's take a basic scenario, where we have a list "**Projects**" with columns - Name (Single Line of Text). Description (Multiple Lines of Text) and Start Date (Date and Time). We have to add new items to the list, update few items and delete the specific item.

Get List Items

In order to get all items from the list, below code can be used-

```
<script>
    $.ajax({
        var siteurl = _spPageContextInfo.webAbsoluteUrl;
        $.ajax({
                    url: siteurl
+ "/_api/web/lists/getbytitle('Projects')/items",
                    method: "GET",
                    headers: { "Accept": "application/json; odata=verbose" },
                    success: function (data) {
                         if (data.d.results.length > 0 ) {
                              //This section can be used to iterate through
data and show it on screen
                         }
                    },
                    error: function (data) {
                        alert("Error: "+ data);
                    }
          });
    });
</script>
```

Now lets see, whats happening in code -

1.) Setting up **siteurl** using **_spPageContextInfo**. This object provides many SharePoint page properties which are useful in client-side code model. A complete list of properties can be seen here .

2.) HTTP **'GET'** method is used to send a request for retrieving items

3.) headers - These are defined to return output in **JSON** format otherwise by default SharePoint returns XML data.

Get Specific List Item

In order to get a specific item from the list, we have to make a small change in URL and pass item Id. See below code -

```
<script>
    $.ajax({
        var siteurl =  spPageContextInfo.webAbsoluteUrl;
        $.ajax({
                    url: siteurl +
"/_api/web/lists/getbytitle('Projects')/items(3)",
                    method: "GET",
                    headers: { "Accept": "application/json; odata=verbose" },
                    success: function (data) {
                         if (data.d.results.length > 0 ) {
                              //This section can be used to iterate through
data and show it on screen
                         }
                   },
                   error: function (error) {
                       alert("Error: "+ JSON.stringify(error));
                  }
          });
    });
</script>
```

If you see Url, **3** is passed which is list Item Id

Delete Specific List Item

In order to delete a specific item from the list, we have to make few changes and do some addition to the code. See below code-

```
<script>
    $.ajax({
        var siteurl = _spPageContextInfo.webAbsoluteUrl;
        $.ajax({
                    url: siteurl +
"/_api/web/lists/getbytitle('Projects')/items(1)",
                    method: "DELETE",
                    headers: { "Accept": "application/json; odata=verbose",
```

```
                                        "X-RequestDigest":
$("#__REQUESTDIGEST").val(),
                                        "If-Match": "*"
                    },
                    success: function (data) {

                },
                error: function (error) {
                    alert("Error: "+ JSON.stringify(error));
                }
            });
    });
</script>
```

Here we have used HTTP "DELETE" method to send the request. In headers, we have added values -

1.) **"X-RequestDigest": $("#__REQUESTDIGEST").val()** - This is used while performing Create Update or Delete operations on entries, if request is not OAuth authorized.

2.) **"If-Match": "*"** - This helps to avoid concurrency problems while updating entries. But if concurrency is not an issue, then '*' can be passed as in above example. Otherwise, we need to get **etag** value by performing GET request to the entity.

Add New List Item

In order to add new list item, we have to create a data variable and assign values to the fields, then perform a '**POST**' request. See below code-

```
<script>
    var data = {
            __metadata: { 'type': 'SP.Data.ProjectsListItem' },
          Title: 'Please provide title here',
          Description: 'Please provide multi line text here',
          Start_x0020_Date: new Date().toISOString();
      };
    $.ajax({
        var siteurl = _spPageContextInfo.webAbsoluteUrl;
        $.ajax({
                    url: siteurl +
"/_api/web/lists/getbytitle('Projects')/items",
                    method: "POST",
                                data: JSON.stringify(data),
                    headers: { "Accept": "application/json; odata=verbose",
                                        "X-RequestDigest":
$("#__REQUESTDIGEST").val()
                    },
```

```
                    success: function (data) {
                                alert('Item added successfully');
                    },
                    error: function (error) {
                        alert("Error: "+ JSON.stringify(error));
                    }
            });
    });
</script>
```

Here data variable assigns values to list fields. The key thing to note here is field names should be internal names, so as in above example, "Start Date" is used as "Start_x0020_Date" (space replaced by _x0020).

Also **__metadata: { 'type': 'SP.Data.ProjectsListItem'** is used, whenever create or update operation is performed. This defines the list item. It is created as below-

__metadata: { 'type': 'SP.Data.' + **List Internal Name** + 'ListItem'

Update List Item

In order to update a list item, we have to pass Item Id in Url, create data variable and use one of the HTTP methods - MERGE, PATCH or PUT. Preferred or recommended methods are PATCH or MERGE a PUT requires all fields to be sent as an object.

This is similar to adding a new item with few minor changes.

```
<script>
    var data = {
            __metadata: { 'type': 'SP.Data.ProjectsListItem' },
          Title: 'Please provide title here',
          Description: 'Please provide multi line text here',
          Start_x0020_Date: new Date().toISOString();
      };
    $.ajax({
        var siteurl = _spPageContextInfo.webAbsoluteUrl;
        $.ajax({
                    url: siteurl +
"/_api/web/lists/getbytitle('Projects')/items(3)",
                    method: "PATCH",
                                data: JSON.stringify(data),
                    headers: { "Accept": "application/json; odata=verbose",
                                "X-RequestDigest":
$("#__REQUESTDIGEST").val(),
                                "If-Match":
"*"
                    },
                    success: function (data) {
```

```
                                alert('Item added successfully');
                    },
                    error: function (error) {
                        alert("Error: "+ JSON.stringify(error));
                    }
                });
    });
</script>
```

Key Notes

1.) If there is a lookup column in the list, say '**Project Status**', then REST returns this column name as "**Project_x0020_StatusId**" i.e. by appending **Id** to column name and value is integer i.e. Id of the referring column in another list. So while performing Create or Update operation, you have to pass Id, not the value.

2.) If there is a date column, the value is returned in ISO format e.x. "**2015-08-26T09:59:32Z**". So while performing Create or Update operation, you have to pass vale in ISO format.

3.) If there is a person field, this is also similar to lookup field, the value returned is Id. So while performing Create or Update operation, you have to pass Id, not the person name.

4.) If there is a Hyperlink field, say "**Project Url**', then it is returned as an array which contains Url and Description. So while performing Create or Update operation, you have to create an array as below and add to the data variable.

```
Project_x0020_Url: {
_metadata: {'type' : 'SP.FieldUrlValue'},
Url : "http://contoso.com",
Description: "Contoso Site"
}
```

Other Operations

There can be scenarios where you want to perform additional operations on data returned by REST API like filtering, sorting, selected fields only etc. For such operations while forming Url to Get data, we have to add some parameters as described below:-

1.) $filter - Syntax for this is **$filter=(Column Internal Name operator value).** See below examples-

- Filter by Title: /_api/web/lists/getbytitle('Projects')/items?$filter=Title eq 'Project 1'
- Filter by ID: /_api/web/lists/getbytitle('Projects')/items?$filter=ID eq 2
- Filter by Date: /_api/web/lists/getbytitle('Projects')/items?$filter=Start_x0020_Date le datetime'2015-08-26T09:59:32Z'
- Multiple Filters: /_api/web/lists/getbytitle('Projects')/items?$filter=(Start_x0020_Date le datetime'2015-08-26T09:59:32Z') and (ID eq 2)

Operators that can be used are - **eq** (equal), **ge** (greater than or equal to), **le** (less than or equal to), **gt** (greater than). **lt** (less than), **ne** (not equal)

2.) $orderby - Syntax for this is **$orderby=(Column Internal Name order).** See below examples-

- Ascending Order: /_api/web/lists/getbytitle('Projects')/items?$orderby=ID asc
- Descending Order: /_api/web/lists/getbytitle('Projects')/items?$orderby=ID desc

3.) $select - Syntax for this is **$select=Field1,Field2,Field3**

- /_api/web/lists/getbytitle('Projects')/items?$select=Title,Start_x0020_Date - By default all columns are returned

4.) $top - Syntax for this is $top Count. This returns top n records. See below example

- /_api/web/lists/getbytitle('Projects')/items?$top 100

5.) $skip - Syntax for this is $skip Count. This skip n records and return rest of the records See below example

- /_api/web/lists/getbytitle('Projects')/items?$skip 10

6.) $expand - Syntax for this is $expand=FieldName/Id. This is very useful when dealing with person or lookup fields where only Id is returned. Using this we can get corresponding value based on Id. Say there is 'Project Status' column in Projects list which is a lookup to Title column in Status List. See below example

- /_api/web/lists/getbytitle('Projects')/items?$select=Project_x0020_Status/Title&$expand=Project_x0020_Status/Id

References

- https://msdn.microsoft.com/en-us/library/office/jj164022.aspx
- https://msdn.microsoft.com/en-us/library/dd541276.aspx
- https://msdn.microsoft.com/en-us/library/office/fp142380.aspx
- https://msdn.microsoft.com/en-us/library/office/jj860569.aspx
- https://msdn.microsoft.com/en-us/magazine/dn198245.aspx

See Also

- http://social.technet.microsoft.com/wiki/contents/articles/31210.sharepoint-2013-get-user-details-from-person-or-group-field-using-rest-api.aspx