# Office AMS: sharepoint proxy for singe page provider hosted apps

|  |  |
| --- | --- |
| Summary: | Applies to: |
| Demonstration of how to write WebAPI controller that acts as proxy for SPA apps to protect access tokens, yet still allow normal request construction to occur within the client. | * Office 365 Multi-Tenant (MT) * Office 365 Dedicated (D) * SharePoint 2013 on-premises |
| Solution: | Core.SharePointProxyForSpaApps, version 1.0 |
| Author: | Matt Mazzola (Microsoft) |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // THIS CODE IS PROVIDED \*AS IS\* WITHOUT WARRANTY OF  // ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING ANY  // IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR  // PURPOSE, MERCHANTABILITY, OR NON-INFRINGEMENT.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |

## Version log

Change log between different releases.

|  |  |
| --- | --- |
| Version | Notes |
| 1.0 | Initial |

# General comments

This sample shows how one construct requests from SPA and proxy through app server to add necessary access tokens. It also could help to familiarize people with how simple it is to get started using angularjs as a tool for more complex apps.

In order for provider hosted apps to request resources from sharepoint it must send an access token for authentication. Currently SharePoint doesn't support the OAuth 2.0 Implicit grant flow so the acces tokens we get have a longer expiration period and must be protected with more caution.

In other words, we can't expose the access tokens to the client and must make request through the server; however we still want to use the same programming model of having all the logic on the client. To have the best of both worlds we use a custom web api controller which acts as proxy add access token to requests and passing them through to sharepoint.

There was another article by Scot Hillier explaining a little more about this technique but it doesn't provide a full sample or utilize the proxy technique which in my opinion allows for more flexibility. With his article people might be tempted to write a controller for every service they might need from sharepoint which is obviously not scalable.

<http://www.itunity.com/article/managing-tokens-sharepoint-2013-singlepage-providerhosted-apps-445>

# Server side setup:

1. Update the Index Action to save the spContext to the Session
2. Follow steps from the article to add WebAPI to MVC project, and create a controller that can access SessionState from the current HttpContext.   
   See: <http://www.itunity.com/article/managing-tokens-sharepoint-2013-singlepage-providerhosted-apps-445>
3. Add Models folder and create Request object that allows deserialization of the body from the wrapper request which we will use to construct the actual request to sharepoint.
4. Update controller you created in step 1 to accept the model as an argument.
5. Update controller to fetch the spContext from the current session and cast it into the proper form.
6. Update controller to construct an httpRequestMessage from the Request model we added earlier
7. Update controller to get the access token from the spContext based on the httpRequestMessage url
8. Update controller to send the httpRequestMessage and return the response

At this point you should have the server side logic completed. When the app is first loaded and hits the index action the spContext is saved, then future requests to the custom controller retreive this context from the session and apply access tokens to the requests.

# CLIENT SIDE SETUP:

This particular example uses AngularJS as it seems to be popular with those haven’t heard of EmberJS. Ok that was a joke to make sure you are reading. Anyways yes this uses angularjs but it really doesn’t matter what you use as long as you follow the following steps:

1. Construct a request configuration object as if you were able to send it directly to the sharepoint site:

{

"url"**:** "https://<hostWebSiteUrl>/\_api/web/title",

"method"**:** "GET",

"headers"**:** {

"Accept"**:** "application/json;odata=verbose"

}

}

1. Wrap the request in another request setting the previous request object to the data attribute:

{

"url"**:** "api/sharepoint",

"method"**:** "POST",

"data"**:** {

"url"**:** "https://<hostWebSiteUrl>/\_api/web/title",

"method"**:** "GET",

"headers"**:** {

"Accept"**:** "application/json;odata=verbose"

}

}

}

1. This ‘api/sharepoint’ endpoint must be the relative path to our proxy controller we created in the server side setup. The Request model accepts the data object, converts to HttpRequestMessage, adds an Authorization header with access token and returns the response:

{

"d"**:** {

"Title"**:** "Site Title"

}

}