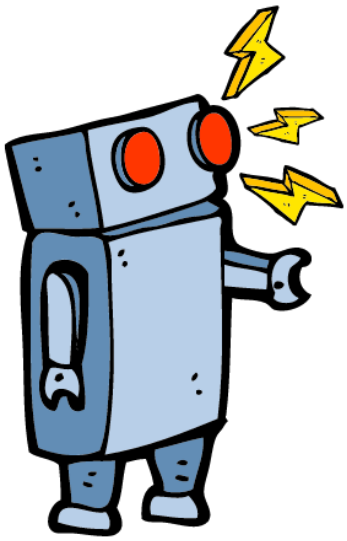


TEAMS APPS HANDS-ON LAB

Using the Teams Platform to surface a Talent Management App in Teams

ABSTRACT

This lab document contains a step-by-step guide to create a Teams App that allowed a recruitment team to manage the positions and candidates all within Teams.

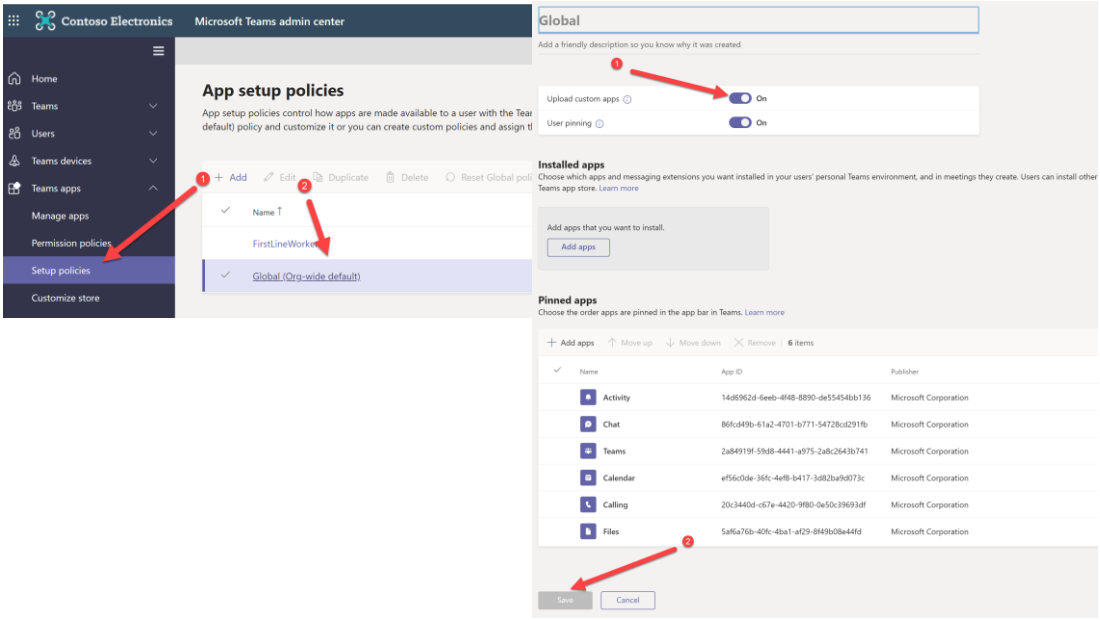
About

Written for the App Modernisation Series – a series of events for Microsoft Partners created by Jason Cabot, Mike Ormerod, Scott Perham, Luciana Blanchard and Jack Lewis. Credit to the original team behind this Teams app hands on lab, found here - https://github.com/OfficeDev/msteams-sample-contoso-hr-talent-app

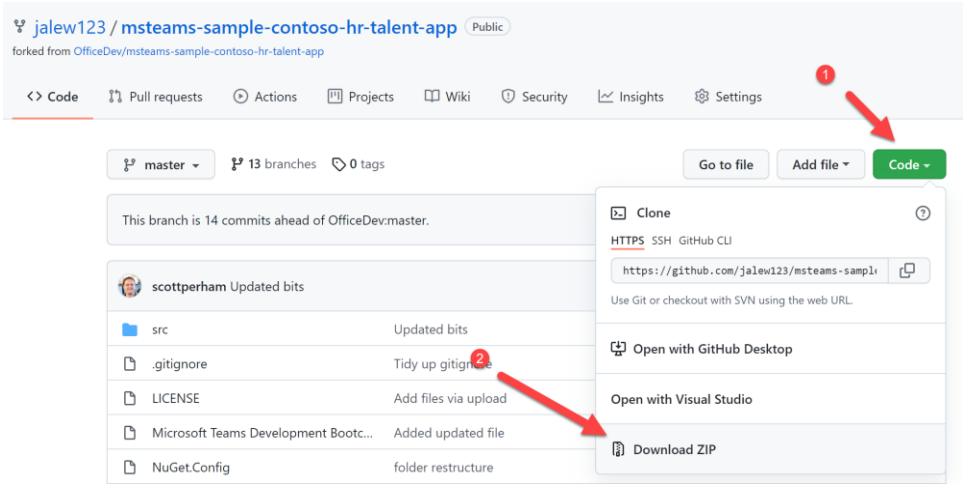
Getting Started

You will need the following to complete this lab:

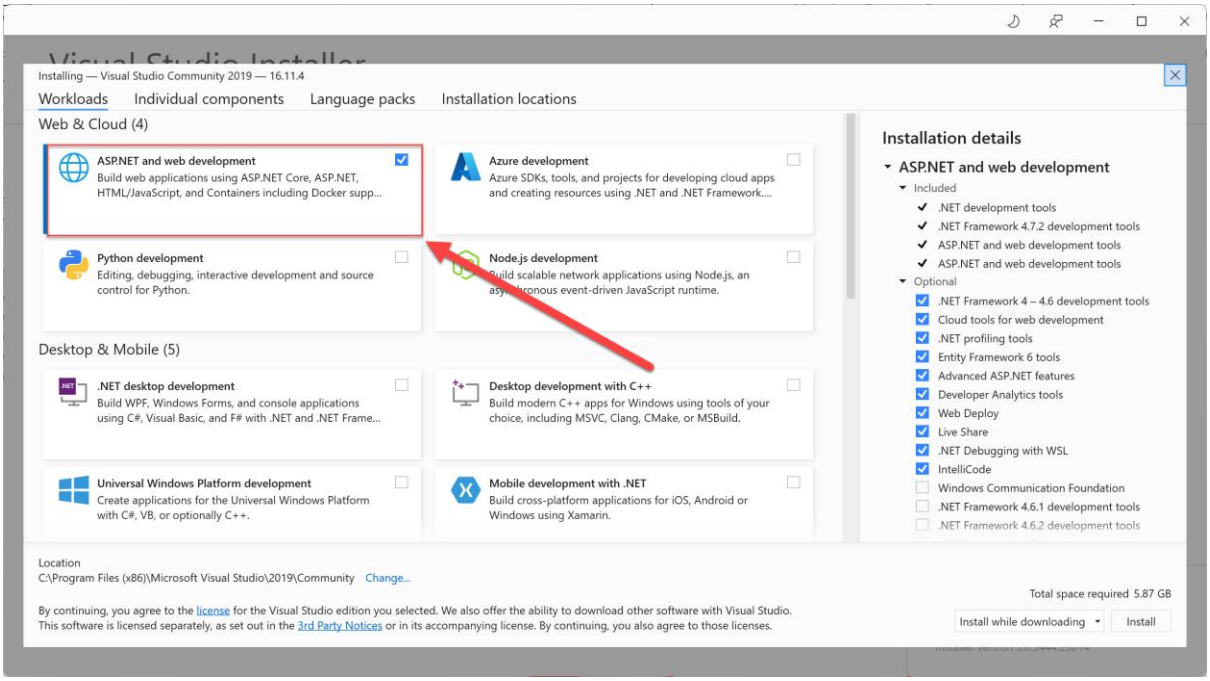
1. **A Microsoft 365 E5 Tenant**, which can be obtained from https://cdx.transform.microsoft.com/ (for Microsoft Partners) or by signing up to the Microsoft 365 Developer Program – Welcome to the Microsoft 365 Developer Program | Microsoft Docs . Alternatively, a Microsoft 365 E5 Trial will also provide the functionality you need. We do **not** recommend using your production/live Tenant for this lab. We will refer to this Tenant as the **Test/Dev Tenant** in this documentation.
2. **Enable Side-Loading in Teams**. To do this, go to https://admin.teams.microsoft.com/ and login as an administrator in your Test/Dev Tenant. Click **Setup policies** and select the **Global (Org-wide default)** policy. Ensure **Upload Custom Apps** is **On,** click **Save** if you have set this from Off to On. ***Note: if this is not enabled, it may take up to 8 hours for this policy to take effect.***



3. **The Talent Management App**, which can be downloaded from here, click **Code** and select **Download ZIP**. You'll then need to extract that ZIP to a location on your computer.



4. **Visual Studio**, installed onto your local machine. If you do not have Visual Studio, the Community edition can be downloaded here. When installing Visual Studio, make sure to select **ASP.NET and web development**.



5. **.Net 5 SDK**, which is required to run the Talent Management App, available here.
6. **NGROK,** a tunnelling tool, which will allow you to run your app within Teams, available here. You'll need to extract the NGROK ZIP to a location on your computer.
7. **Postman,** a tool that allows you to send HTTP requests from your computer, available here.
8. **Azure Subscription,** required to create the Azure Bot Service instance that will be used within your Teams app. This resource does not cost anything so if you want to use an existing subscription that is fine. Alternatively, sign up for an Azure trial here.

Talent Management App Lab

Overview

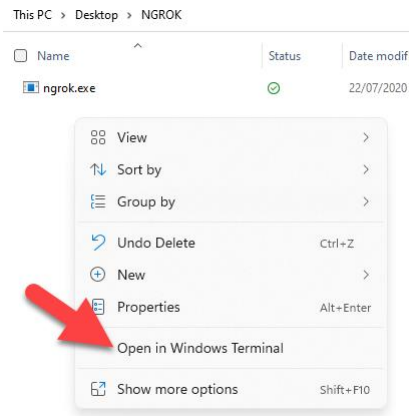
During this lab, you will learn how to build and deploy a Microsoft Teams app in Office 365 that will be used by the Human Resources department within their Microsoft Teams clients. The app will facilitate the department's hiring of new talent into the organization, provide immediate interview feedback, schedule interview loops, and improve the overall hiring process of new employees. The lab is divided into several exercises that will help you understand how to transform hiring and candidate management flow of new talent and make it more interactive and responsive for HR teams and interviewees.

Lab Steps

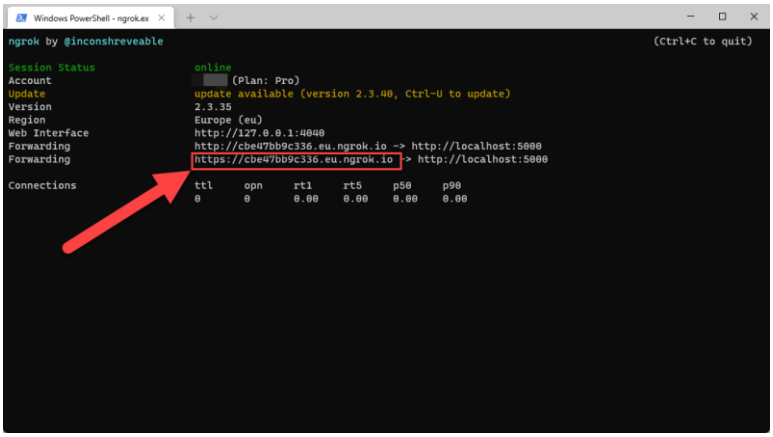
1) Setup Talent Management App

In this step we will setup the NGROK Tunnel and open and configure the Talent Management App in Visual Studio. Setup the Azure AD App Registration & Bot Service Resource in Azure. And then run (debug) the Talent Management App locally, within Visual Studio.

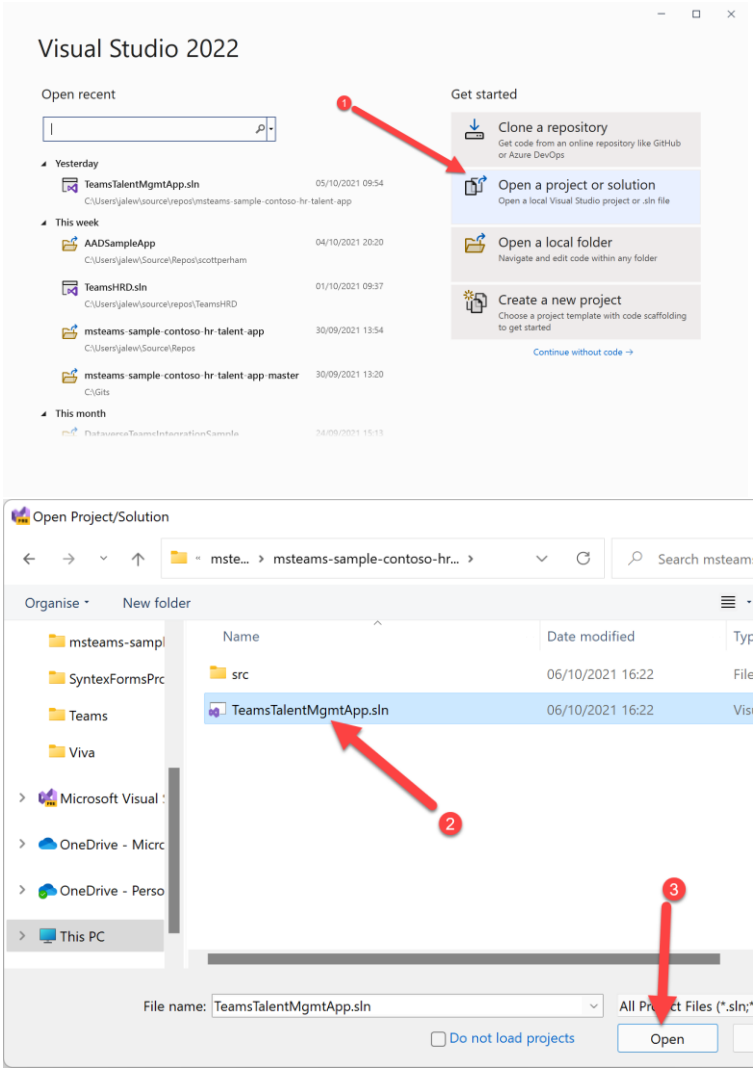
1. Open the folder where you extracted the NGROK ZIP to, and start a Command Prompt/Windows Terminal session here.



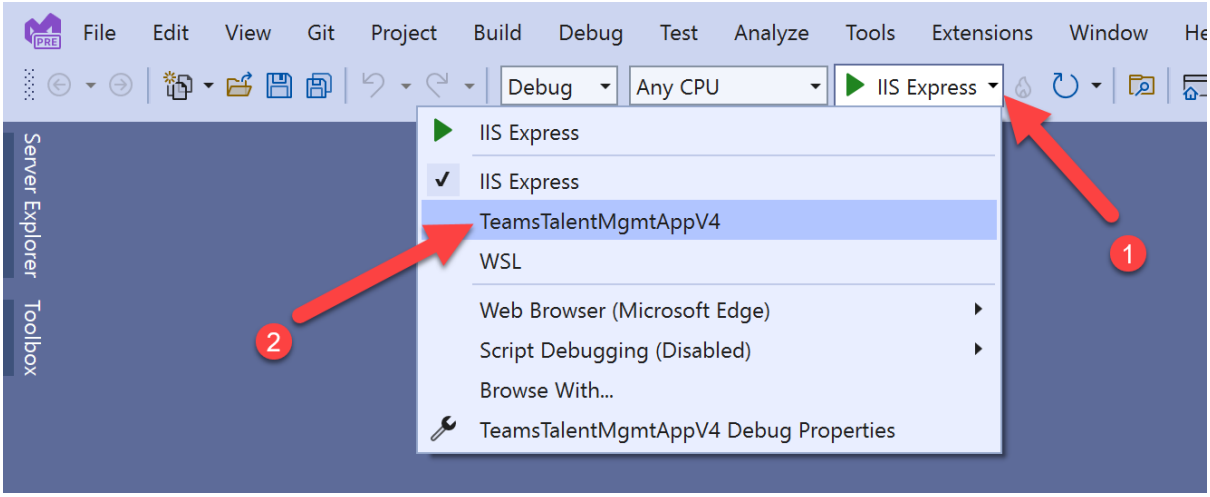
2. Enter the following in the Command Prompt/Windows Terminal session: **.\ngrok http 5000** This will open an NGROK tunnel and forward traffic destined for the forwarding URLs to localhost:5000 (where you will run the Talent Management app later in this lab).
Make note of the HTTPS forwarding URL. You will need this later when you configure the Teams app. Do not close the NGROK window. Make sure this window is left open.



3. Open Visual Studio and open the **TeamsTalentMgmtApp.sln** file found within the extracted Talent Management Repo you downloaded from GitHub.

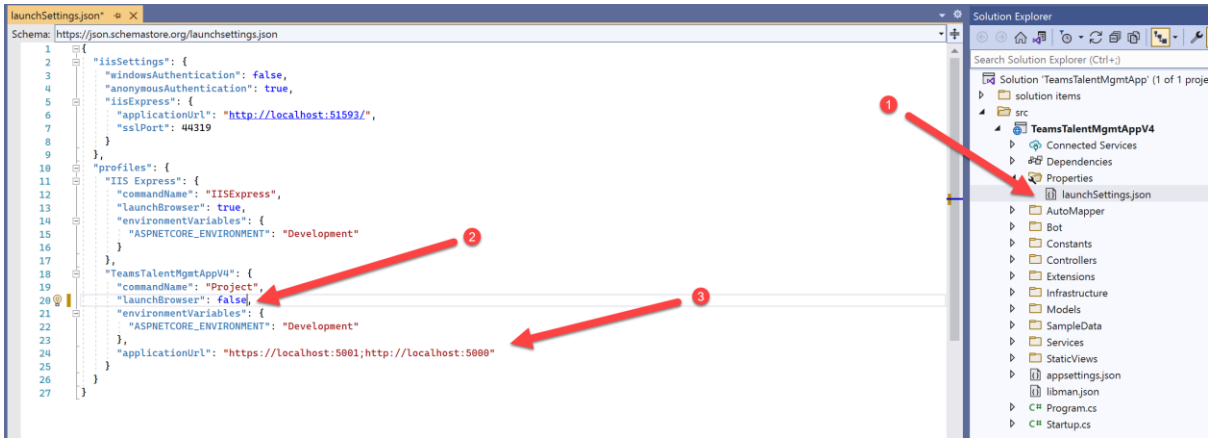


4. When Visual Studio has loaded the solution and associated projects, we need to set the debug properties correctly. In the Visual Studio toolbar, select the **IIS Express dropdown menu** and select **TeamsTalentMgmtAppV4**.

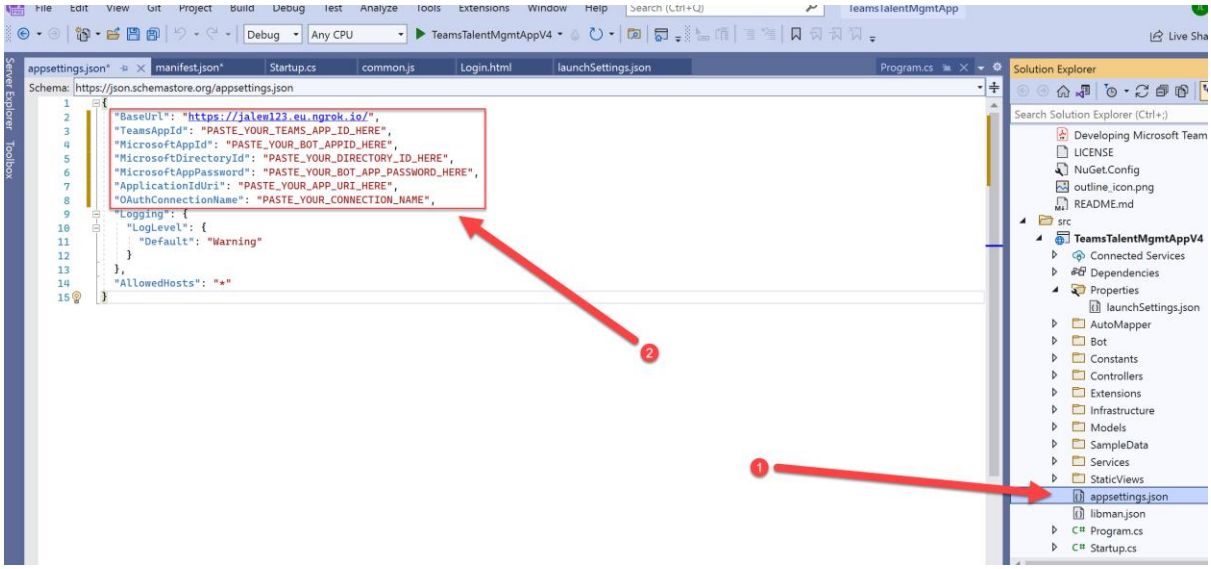


5. On the right-side of Visual Studio, within Solution Explorer, expand **src, TeamsTalentMgmtAppV4, Properties** and select **launchSettings.json.** This file contains the properties that are used when we run (debug) the Talent Management App locally, in Visual Studio. Change the value of **launchBrowser** (line 20) to **false**. Review the **applicationUrl**

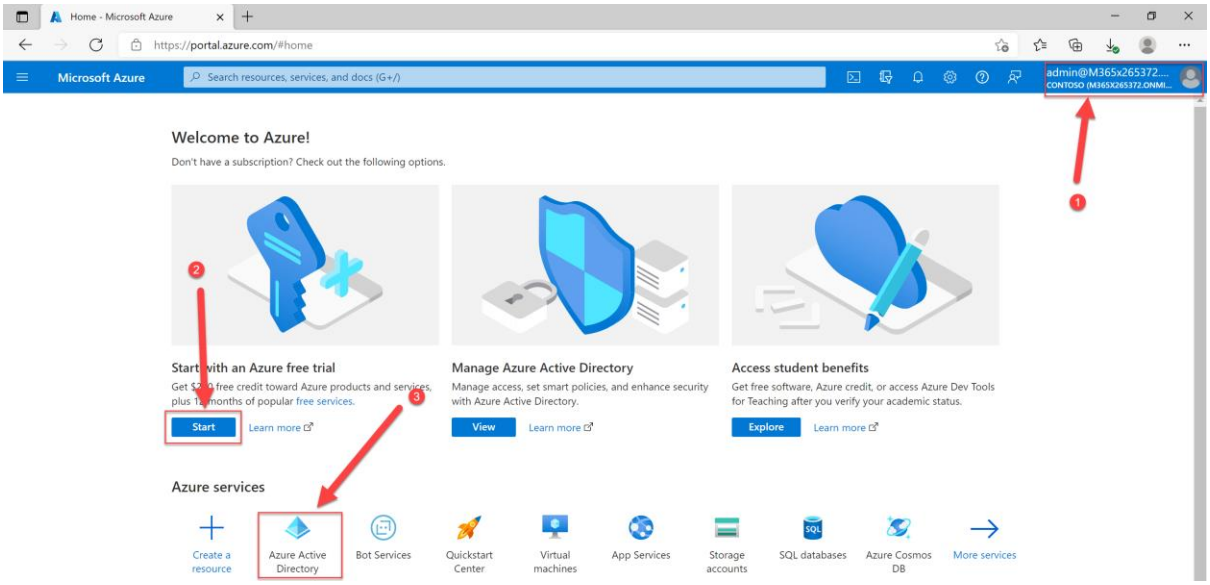
values (line 24) – these should use port 5001 and 5000 (for HTTPS and HTTP respectively). Hit **CTRL+S** to save or click the Save icon in the toolbar.



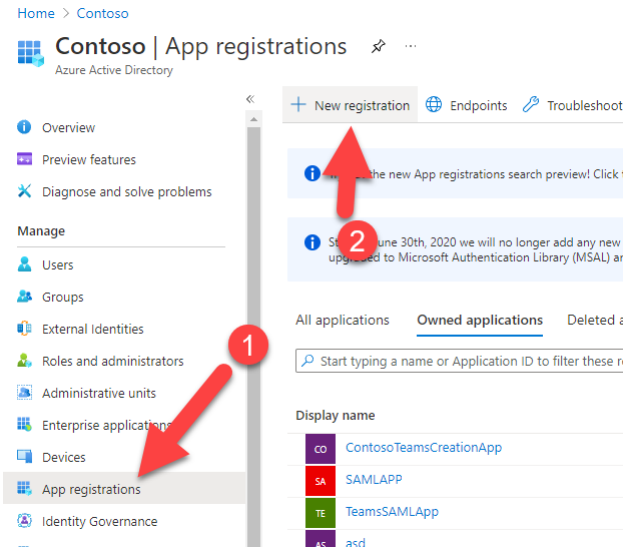
6. Select **appSettings.json**. This file contains placeholder values for **BaseUrl, TeamsAppId, MicrosoftAppId, MicrosoftDirectoryId, MicrosoftAppPassword, ApplicationIdUri & OAuthConnectionName**. Before we can run this app, we need to replace these with actual values. Currently, we only know the **BaseUrl** value - it is the NGROK URL. Replace the value for **BaseUrl** (line 2) with the **NGROK HTTPS URL Value** (gathered during step 2 of this section of the lab). Hit **CTRL+S** to save or click the Save icon in the toolbar.



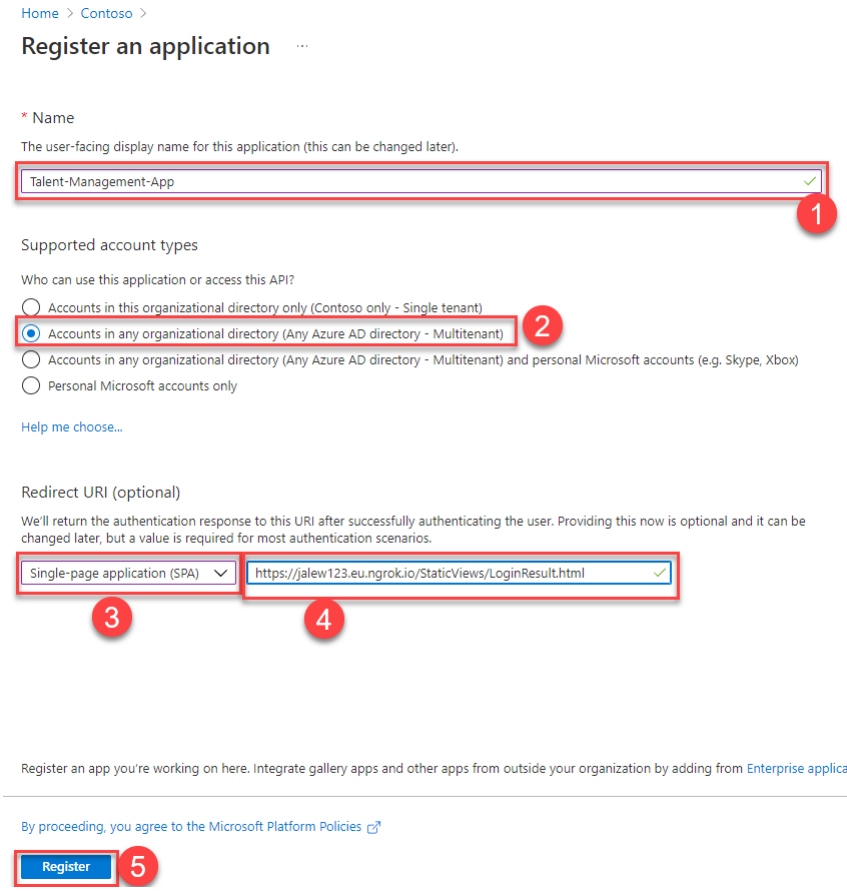
7. We now need to setup the Azure AD App Registration and Azure Bot Service resource in Azure. This will provide us with the remaining values for the appSettings.json file. Navigate to https://portal.azure.com/ and **sign in with the Administrator account, that has access to an Azure Subscription. If you do not have an Azure Subscription, sign in with your Test/Dev Tenant Administrator, and sign up for an Azure Free Trial**. Once you are signed in and have access to an Azure Subscription on that account, Select **Azure Active Directory**.



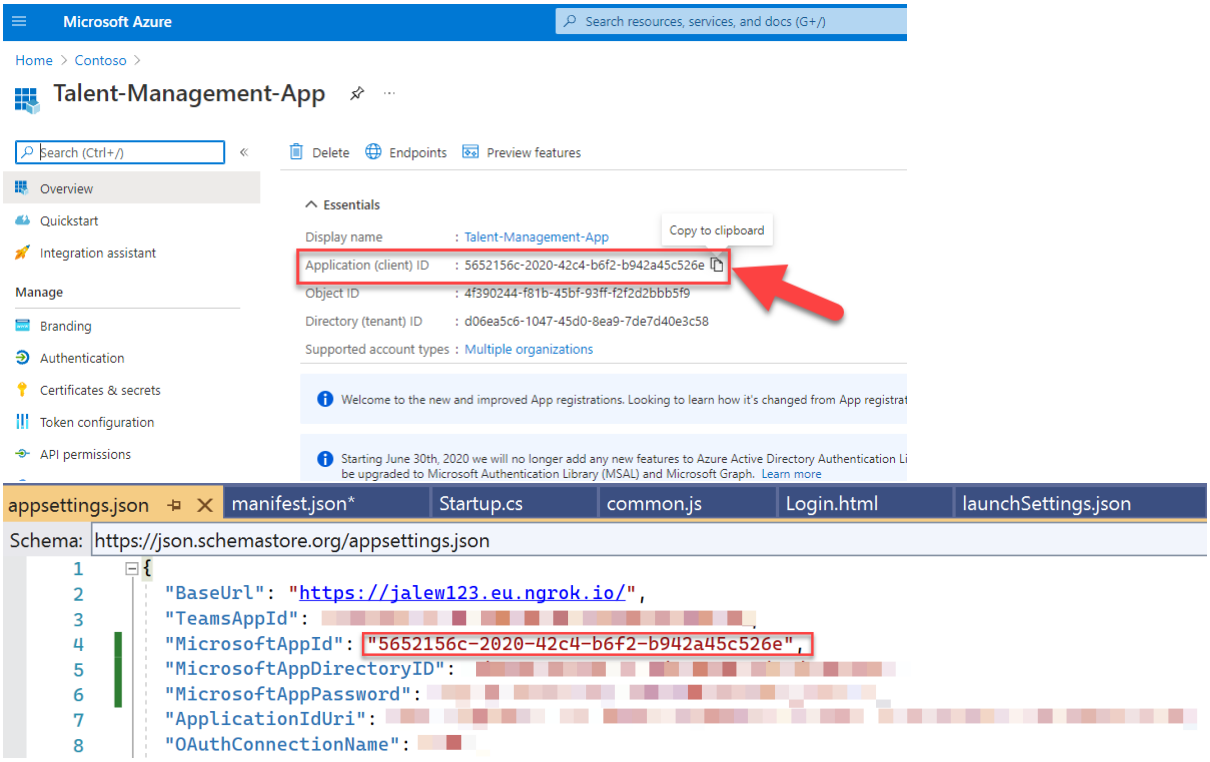
8. Select **App Registrations** and then click **New Registration**



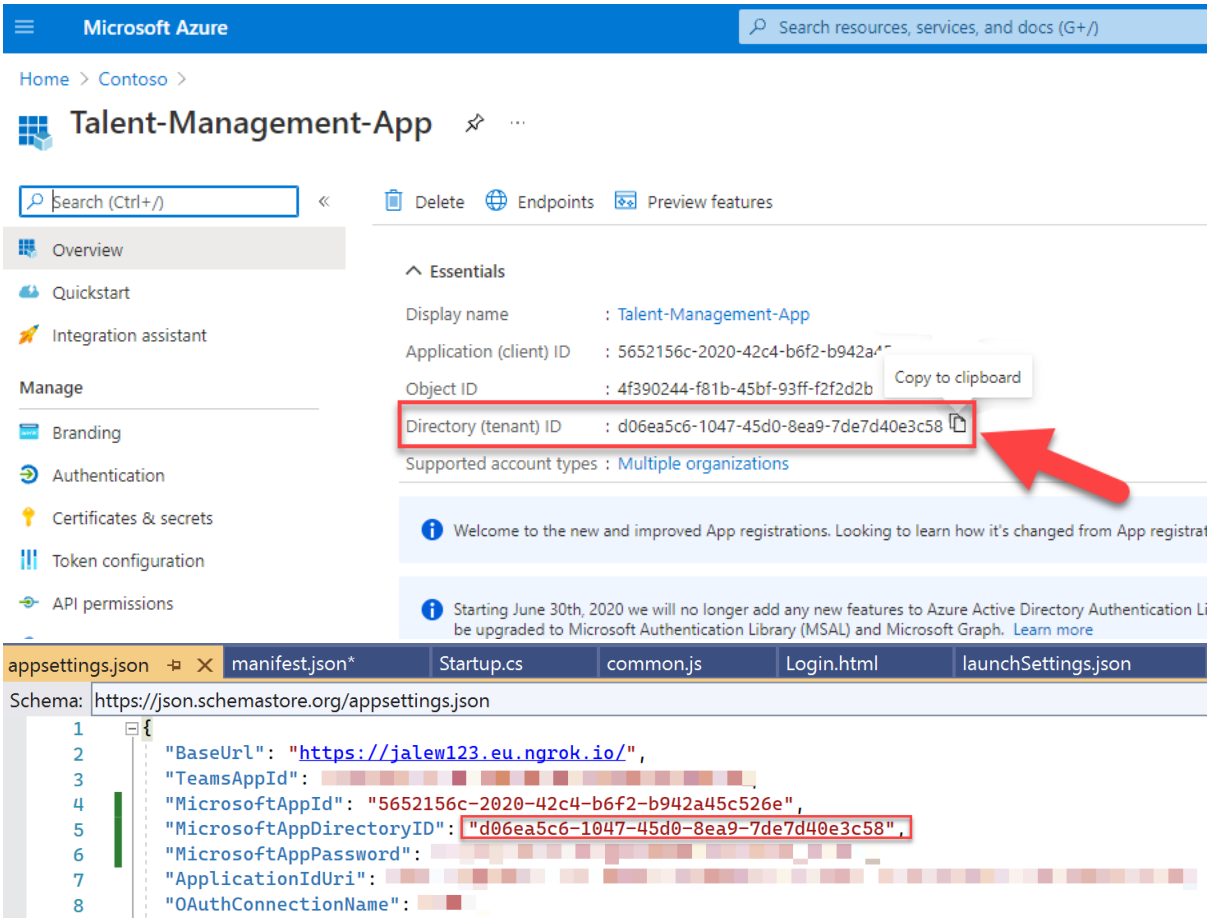
9. Enter a name for your App, such as **Talent-Management-App**, select **Accounts in any organizational directory (Any Azure AD directory – Multitenant)**, select **Single-page application (SPA)** from the dropdown menu and then enter your Redirect URI as NGROKURL/StaticViews/LoginResult.html – click **Register.**



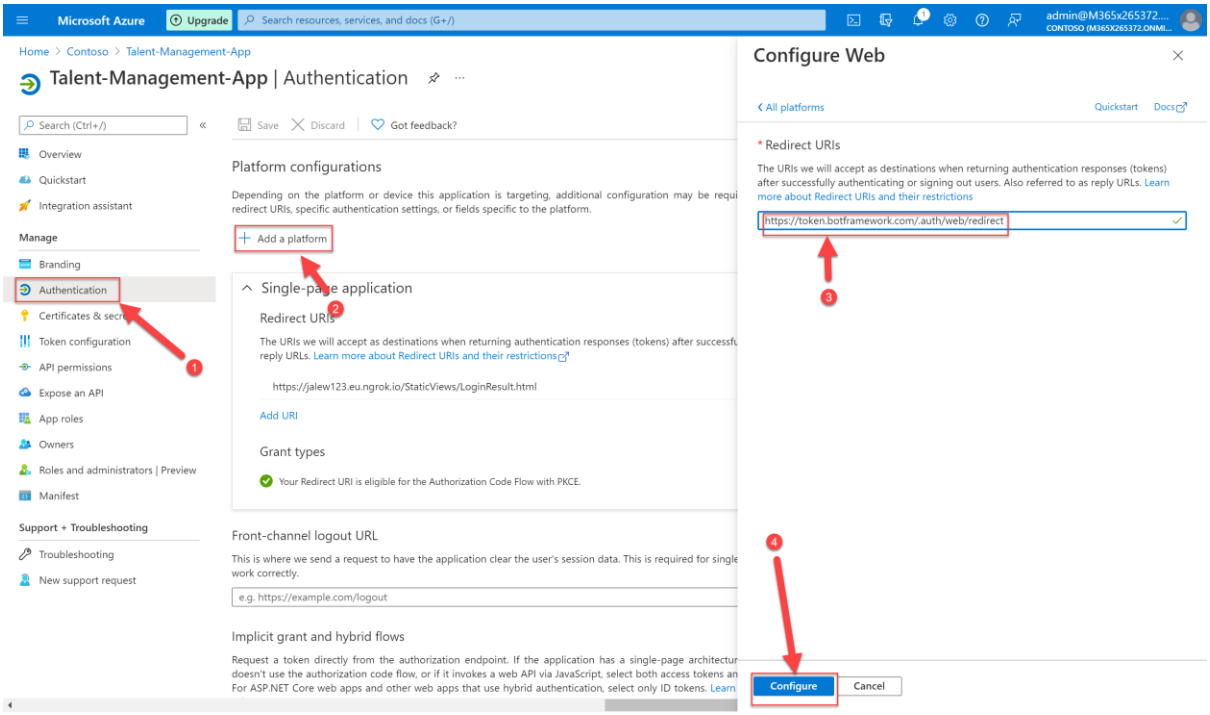
10. On the Overview page, copy the **Application (client) ID** and enter this as the **MicrosoftAppId** value in appsettings.json (line 4).



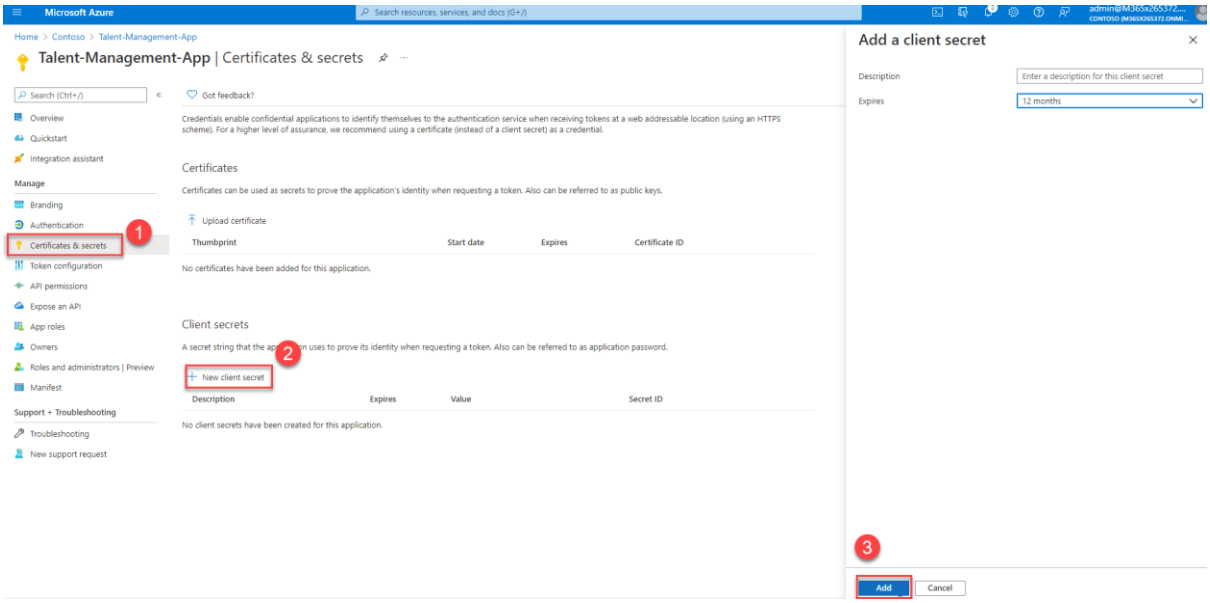
11. Copy the Directory (tenant) ID and enter this as the **MicrosoftDirectoryID** value in appsettings.json (line 5).



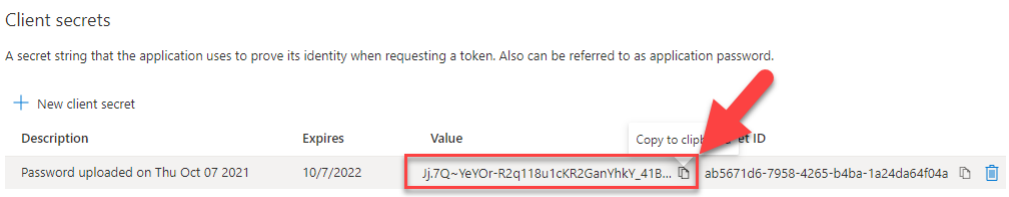
12. Select **Authentication**, click **Add a platform,** select **Web**, enter the Redirect URI of https://token.botframework.com/auth/web/redirect and click **Configure**. For context, these Redirect URIs are the allowed locations that Azure AD will redirect clients to when an access token has been received via a successful authentication. We need to configure the Bot Framework URI, as Bot SSO uses this address when obtaining authorisation/consent. The SPA URI is used to redirect browsers back to you LoginRedirect.html page, which is the page used to process the Azure AD access token, and then store it in Local Storage for use when calling the Talent Management APIs. Feel free to inspect the LoginRedirect.html & Login.html pages to see what they do, and how the use the latest MSAL Browser libraries to enable SSO support for this app.



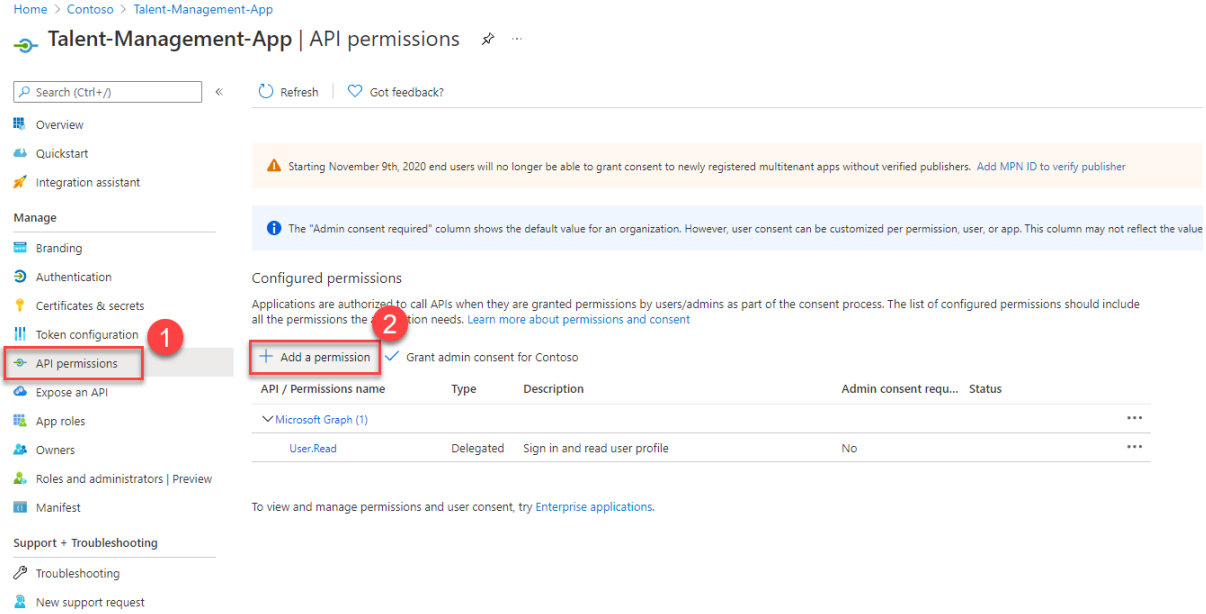
13. Select **Certificates & secrets,** select **New client secret** and click **Add**



14. Once generated, copy the **Value** of the Client Secret and enter this as the **MicrosoftAppPassword** value in appsettings.json (line 6).

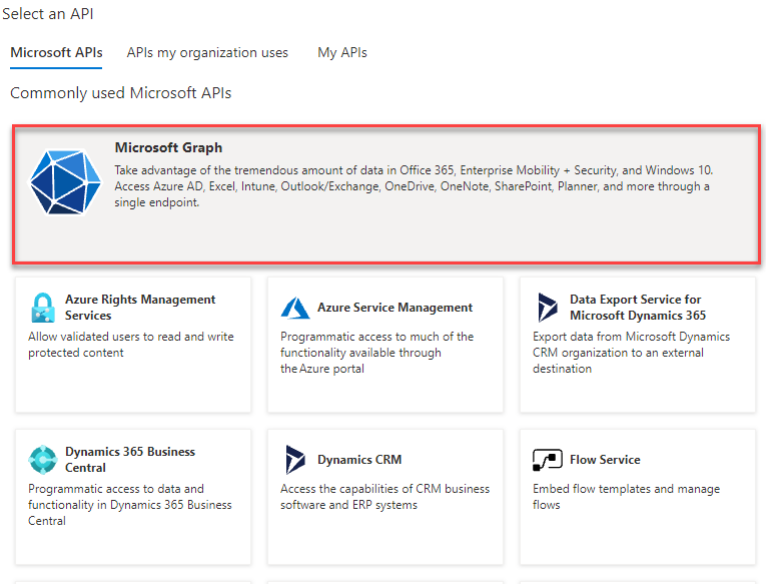


15. Select **API permission** and click **Add a permission**.

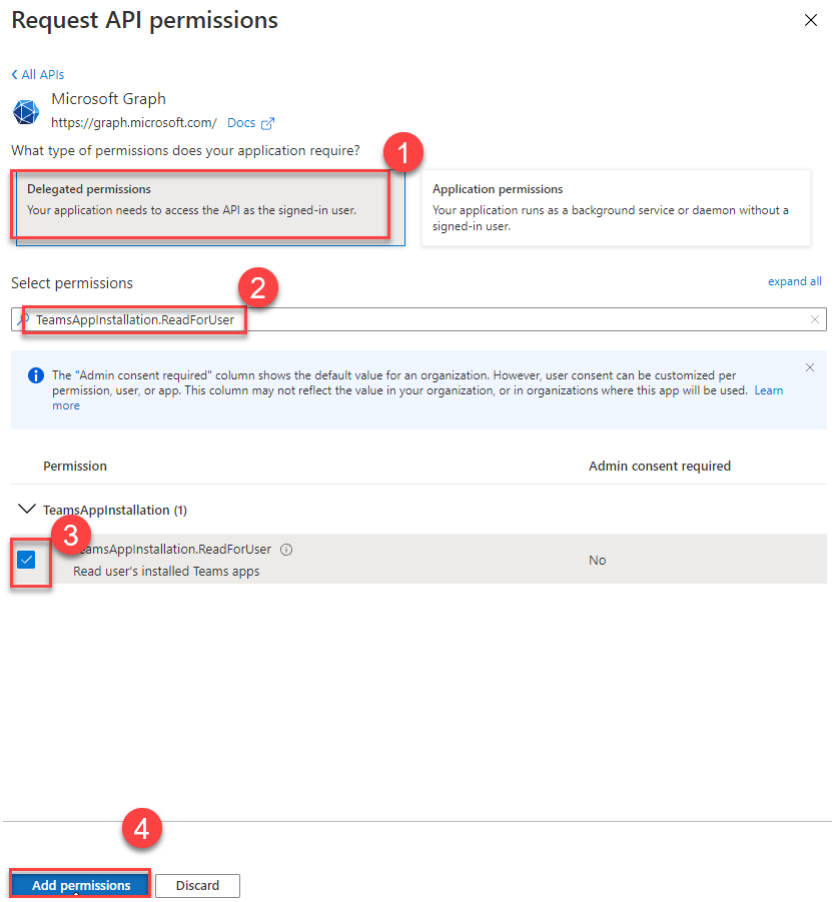


16. Select **Microsoft Graph**

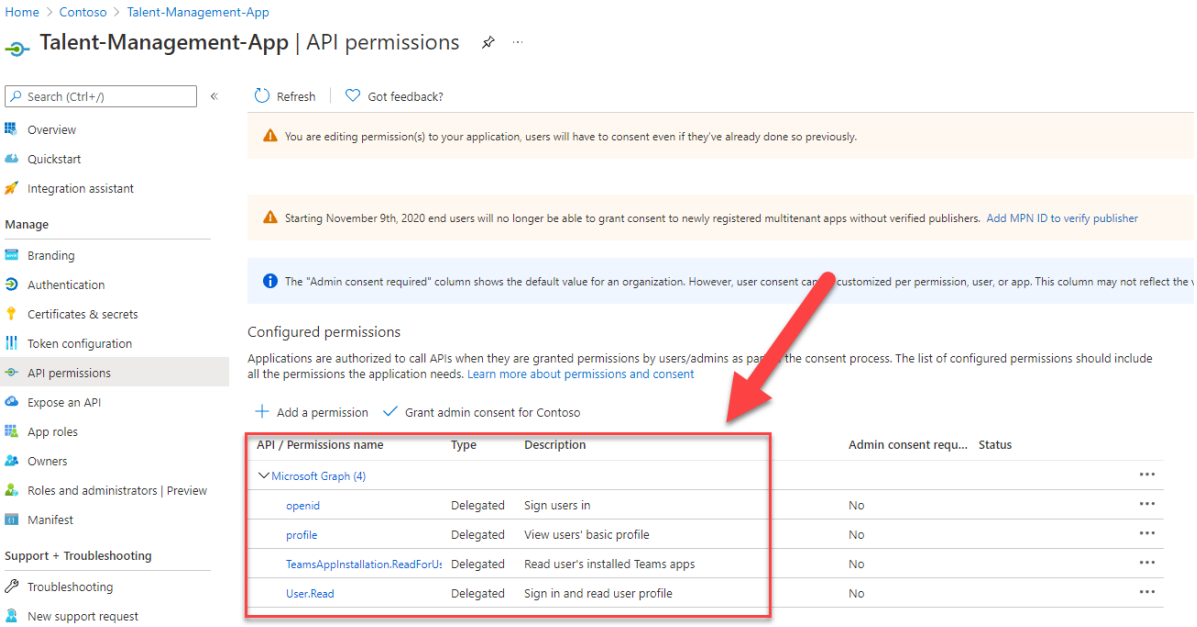
Request API permissions



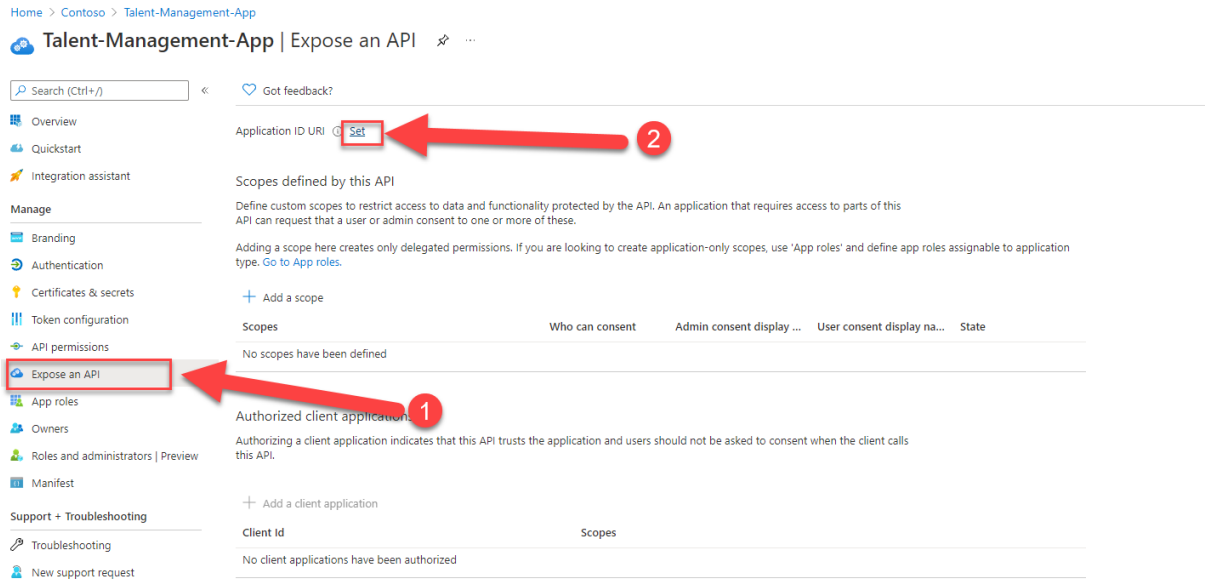
17. Select **Delegated permissions**, select **openid**, **profile** and then search for and select **TeamsAppsInstallation.ReadForUser**. Click **Add permissions.**



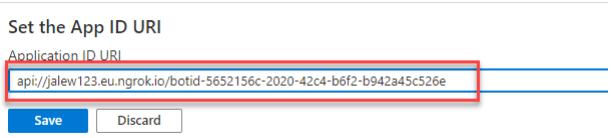
18. If successful, your API Permissions should look like the below:



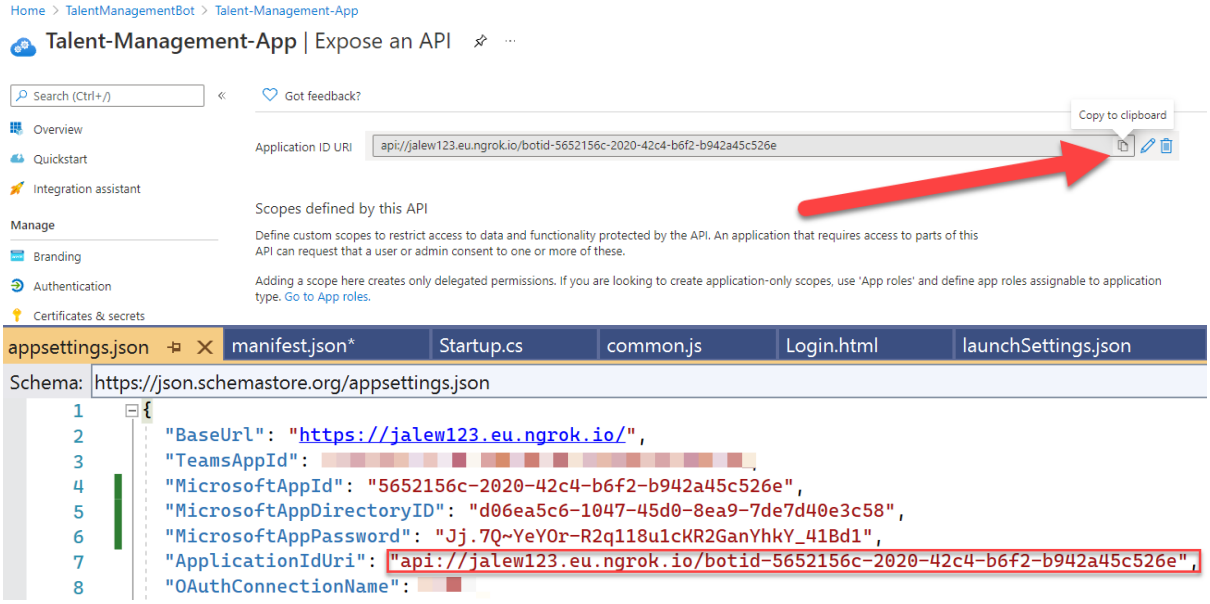
19. Select **Expose an API** and **Set** the Application ID URI.



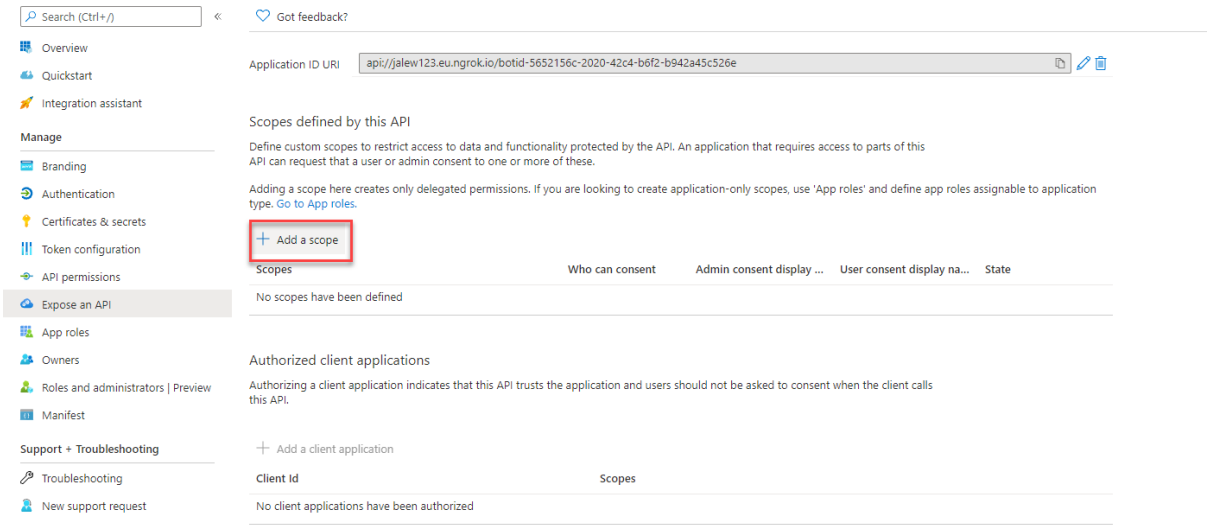
20. The Application ID URI needs to be configured in the following format: **api://NGROKURL/botid-MICROSOFTAPPID** – click **Save.**



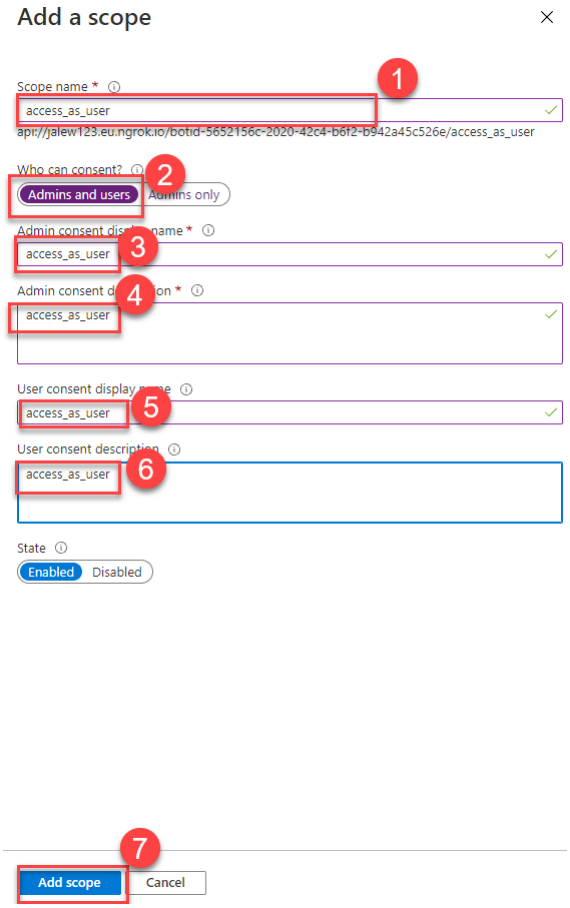
21. Once saved, copy the **Application ID URI** and enter this as the **ApplicationIdUri** value in appsettings.json (line 7).



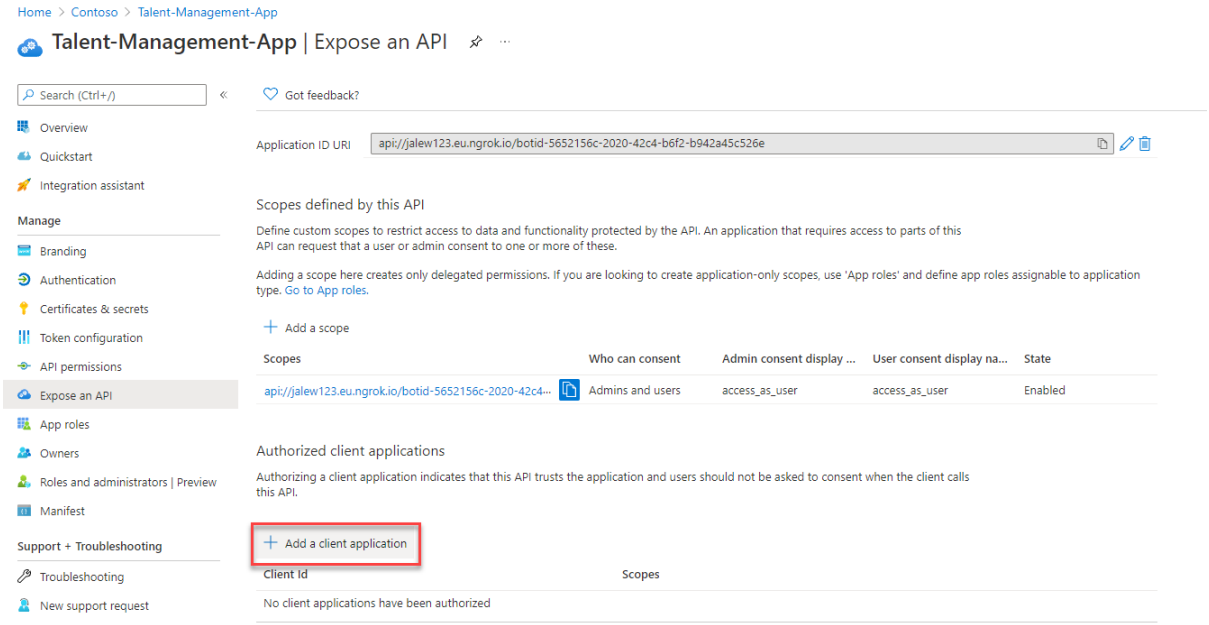
22. Select **Add a scope**



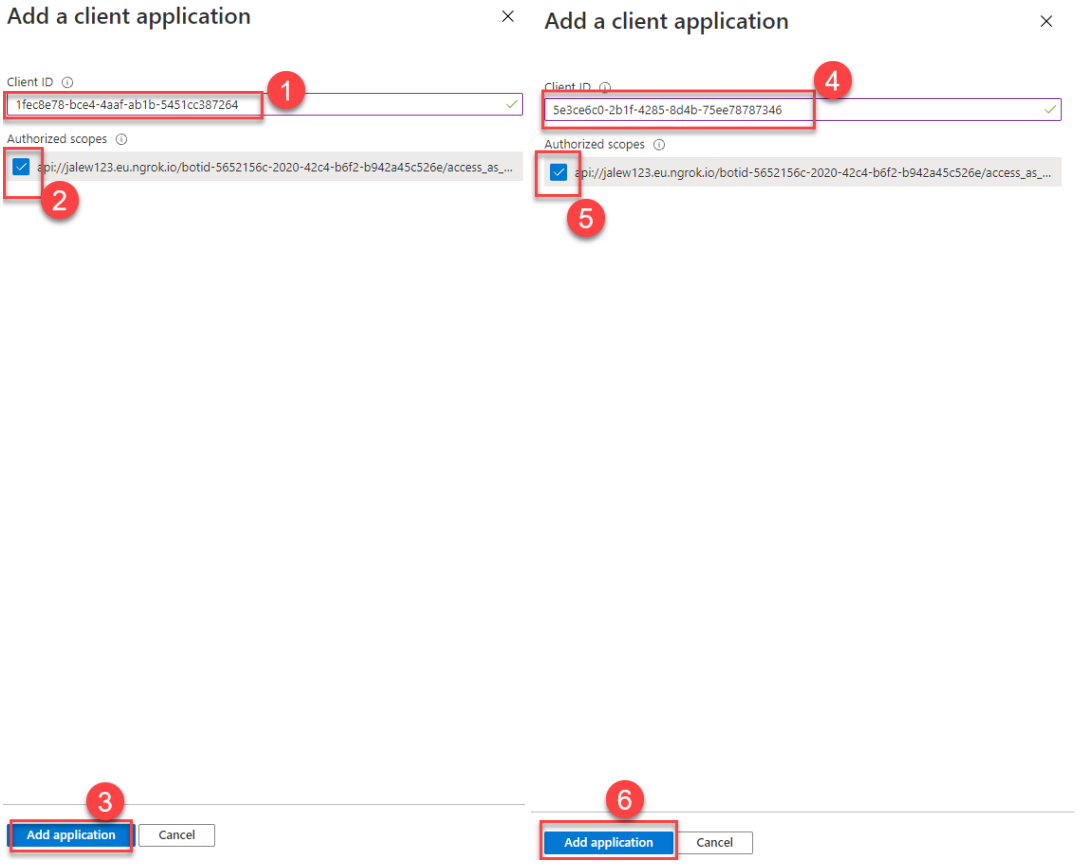
23. We now need to configure the API scope that will be used when SSO into your application has been completed both inside and outside of Microsoft Teams. Enter the Scope name **access_as_user** select **Admins and users**. For the display names and descriptions enter **access_as_user** (note: in a real world scenario you would populate these with more information). Select **Add scope**



24. We now need to enable the Teams Native/Mobile and Web applications to swap their access tokens for an access token for the Talent Management Application. Select **Add a client application.**

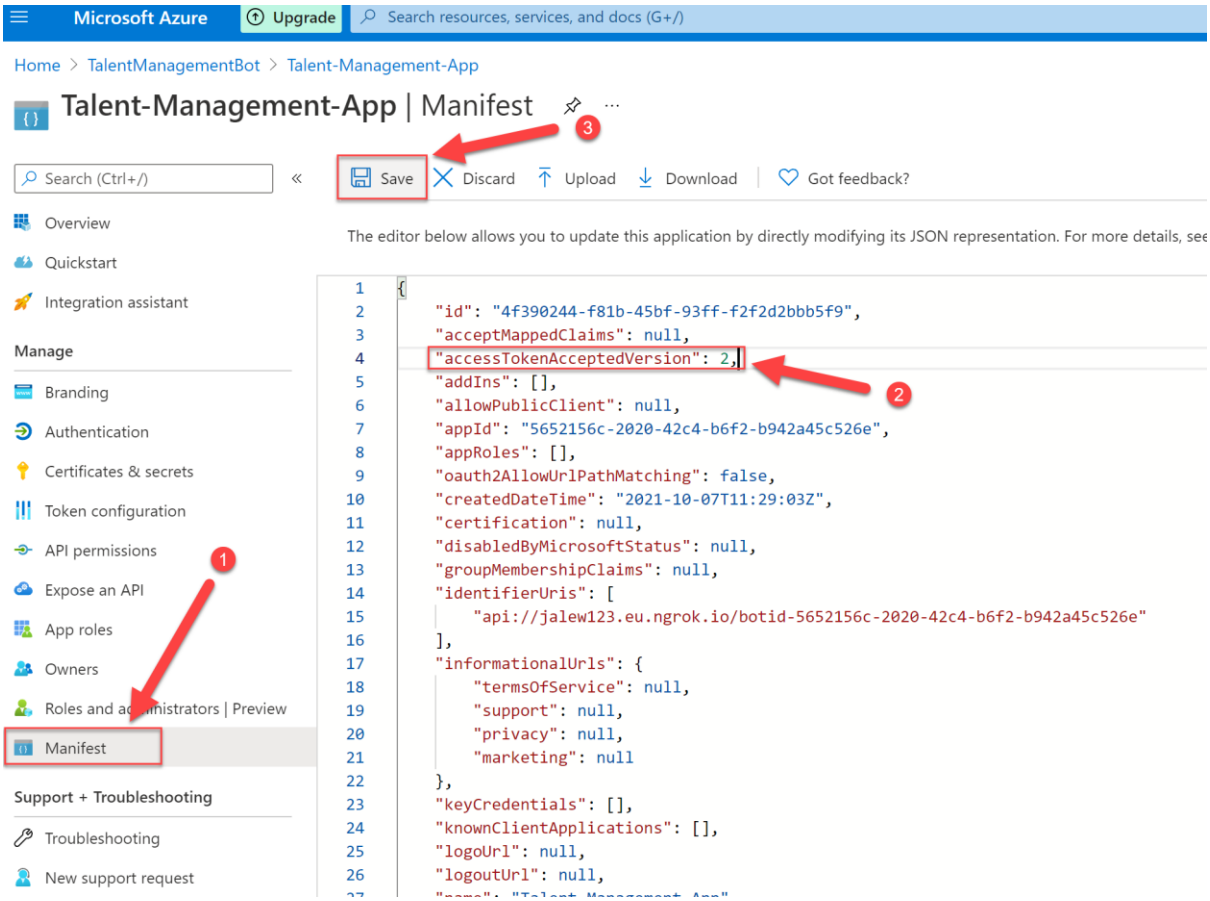


25. Teams is identified by 2 separate Client IDs, so we need to repeat this step twice. Enter **1fec8e78-bce4-4aaf-ab1b-5451cc387264** as the Client ID, select the **access_as_user** scope and click **Add Application.** Then click **Add a client application** again, enter **5e3ce6c0-2b1f-4285-8d4b-75ee78787346** as the Client ID, select the **access_as_user** scope and click **Add Application.**

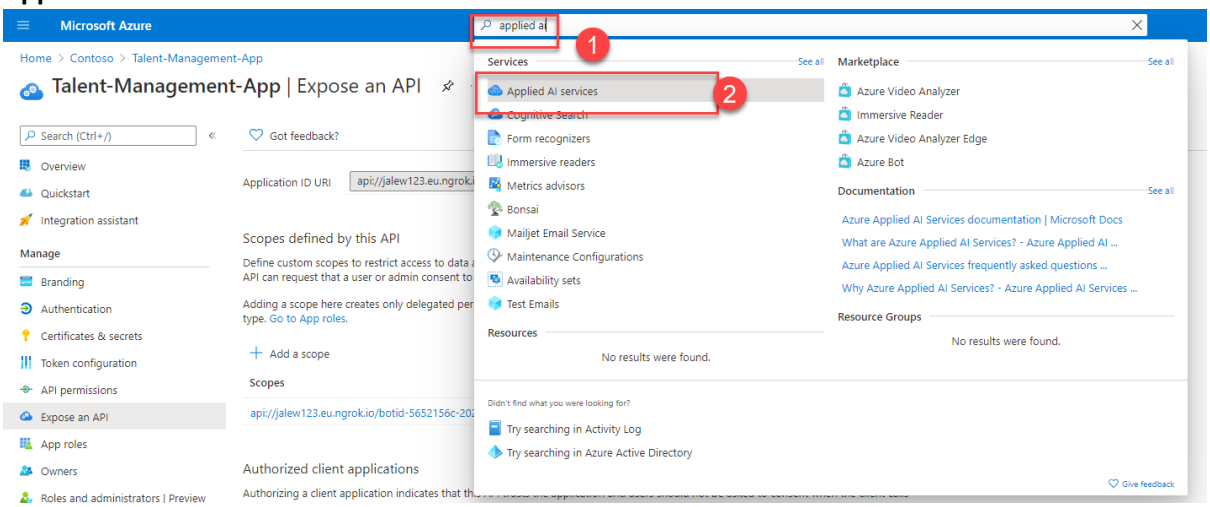


You have now authorised the Teams Apps to swap their access tokens for an access token for you application. More on this later in the lab!

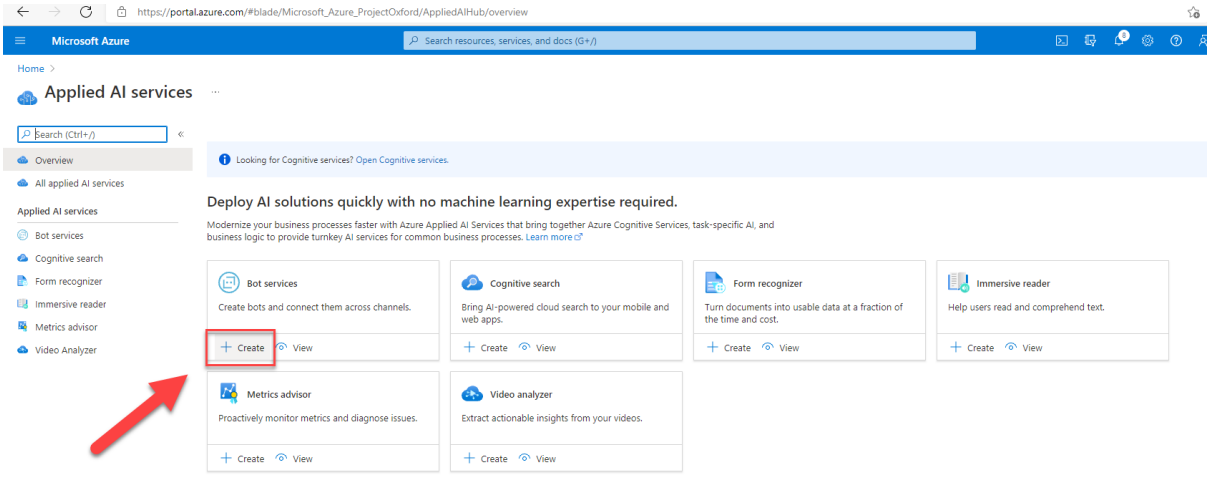
26. The final step we need to do, is to configure the App Registration manifest to only use Access Tokens v2. Select **Manifest**, change the value of **accessTokenAcceptedVersion** to **2** (line 4). Select **Save**.



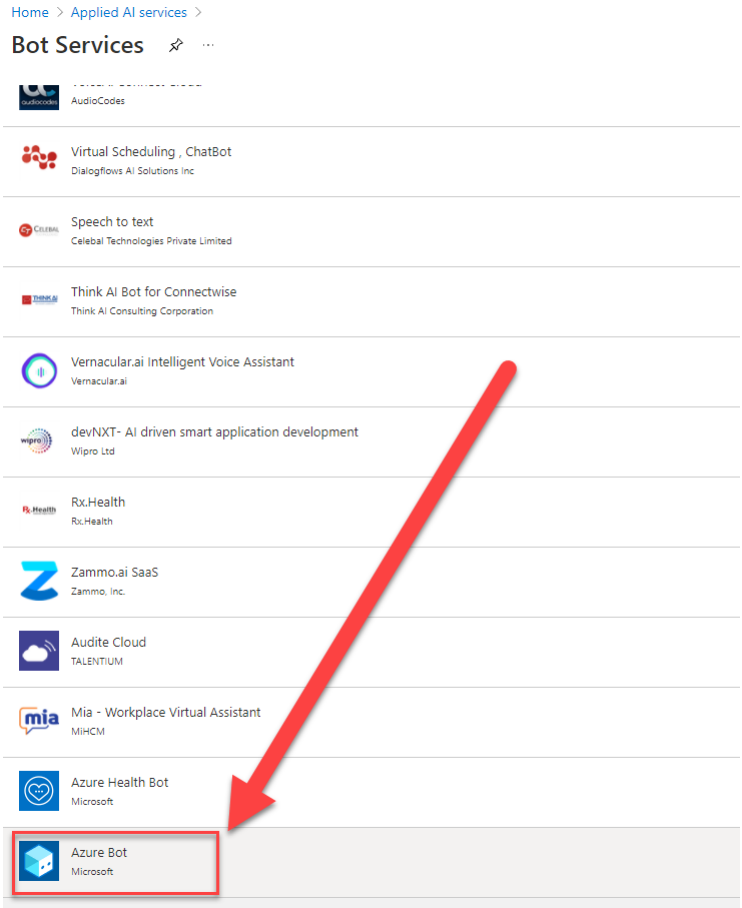
27. The Azure AD App Registration has now successfully been configured. We now need to configure the Azure Bot Services resource, that is linked to the Azure AD App Registration you have just created. In the Search bar at the top of the Azure Portal, search for and select **Applied AI services**.



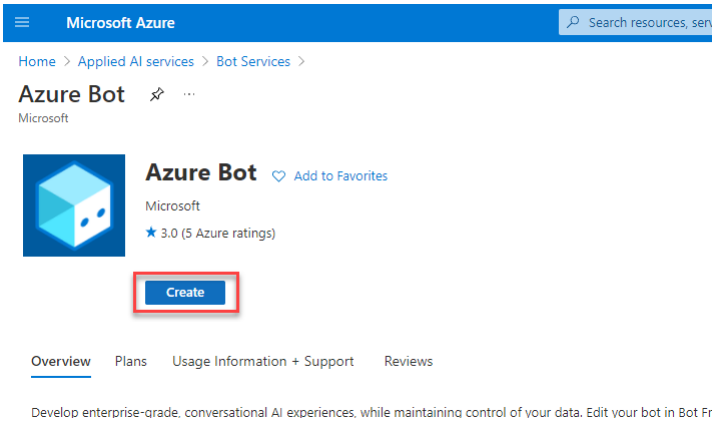
28. Find Bot Services and click **Create**.



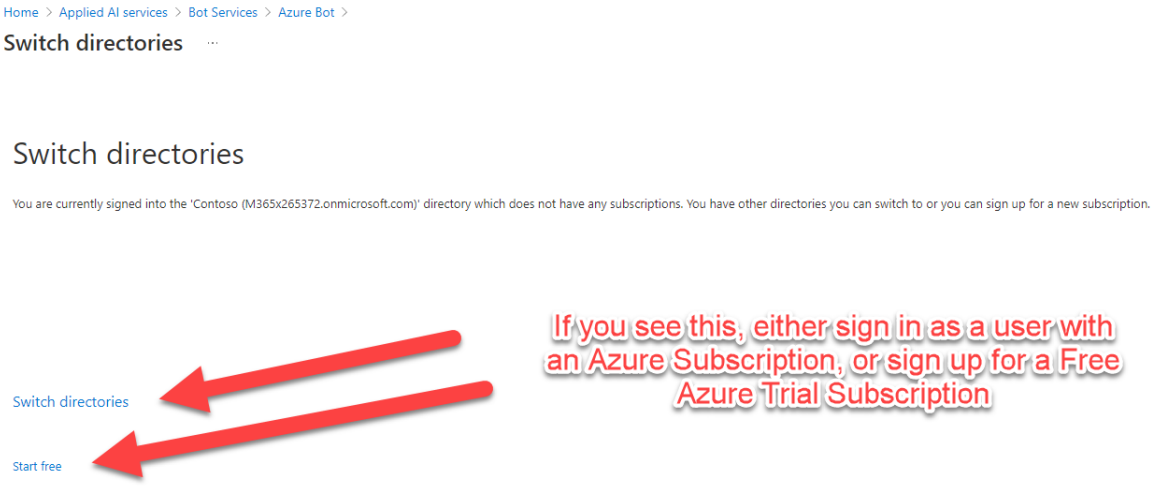
29. Scroll down and select **Azure Bot** (note: you may need to select **load more** at the bottom of the page to see Azure Bot!)



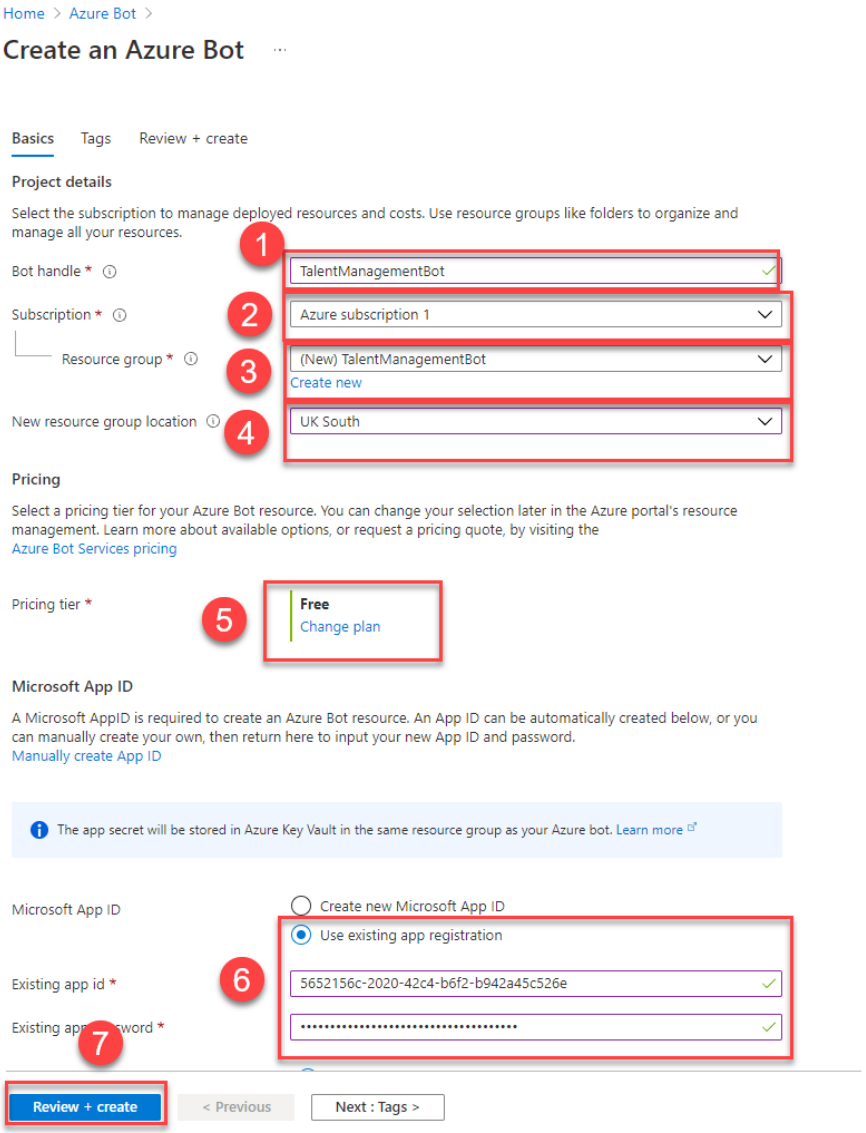
30. Select **Create**



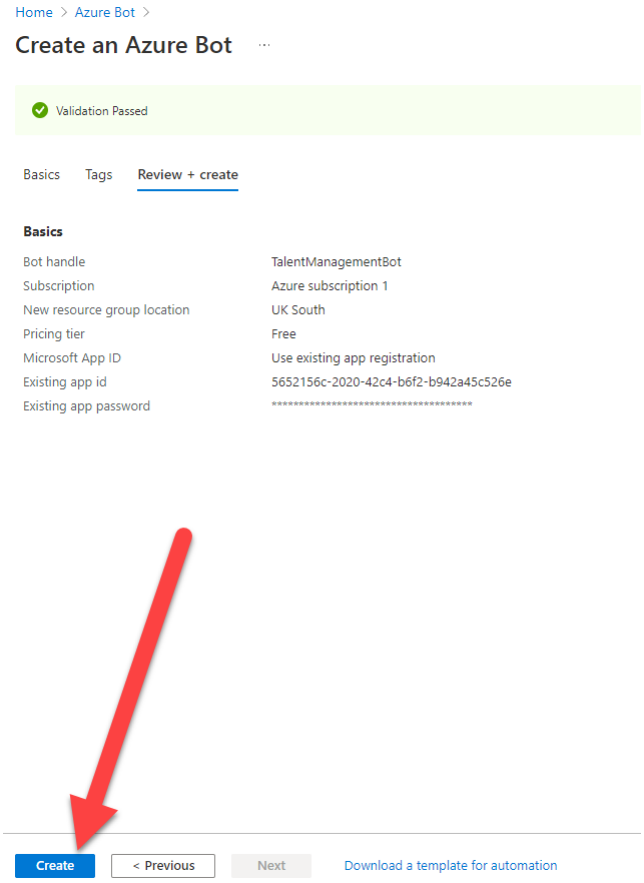
31. **At this point, you may see the following error page** – this is because the user you are logged in as has no access to an Azure Subscription with funding/credits. If you see this, either sign-in as another user with access to an Azure Subscription or sign up for an Azure Free trial!



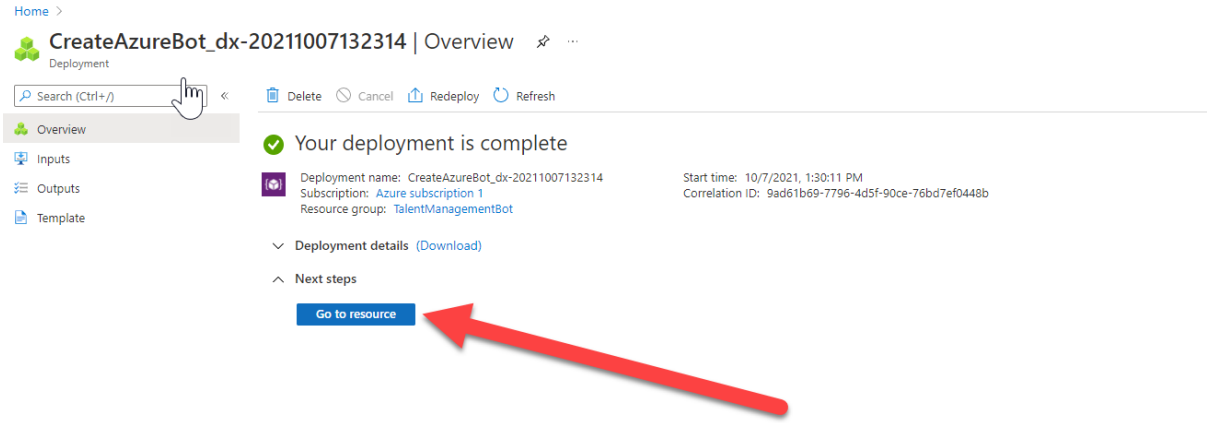
32. Assuming you have the Azure Subscription in place, you will see this interface. Enter a Bot handle name such as **TalentManagementBot** (note: these must be globally unique). Select an **Azure Subscription**, create or select an existing **Resource Group** for the Azure Bot to reside within. Select a **location** for the resource group if you are creating a new resource group. Change from Standard pricing to **Free**. For the Microsoft App ID, it's important that you select **Use an existing app registration**, enter your **Microsoft App ID** and **App Secret** that you generated earlier in this lab. Select **Review + Create**.



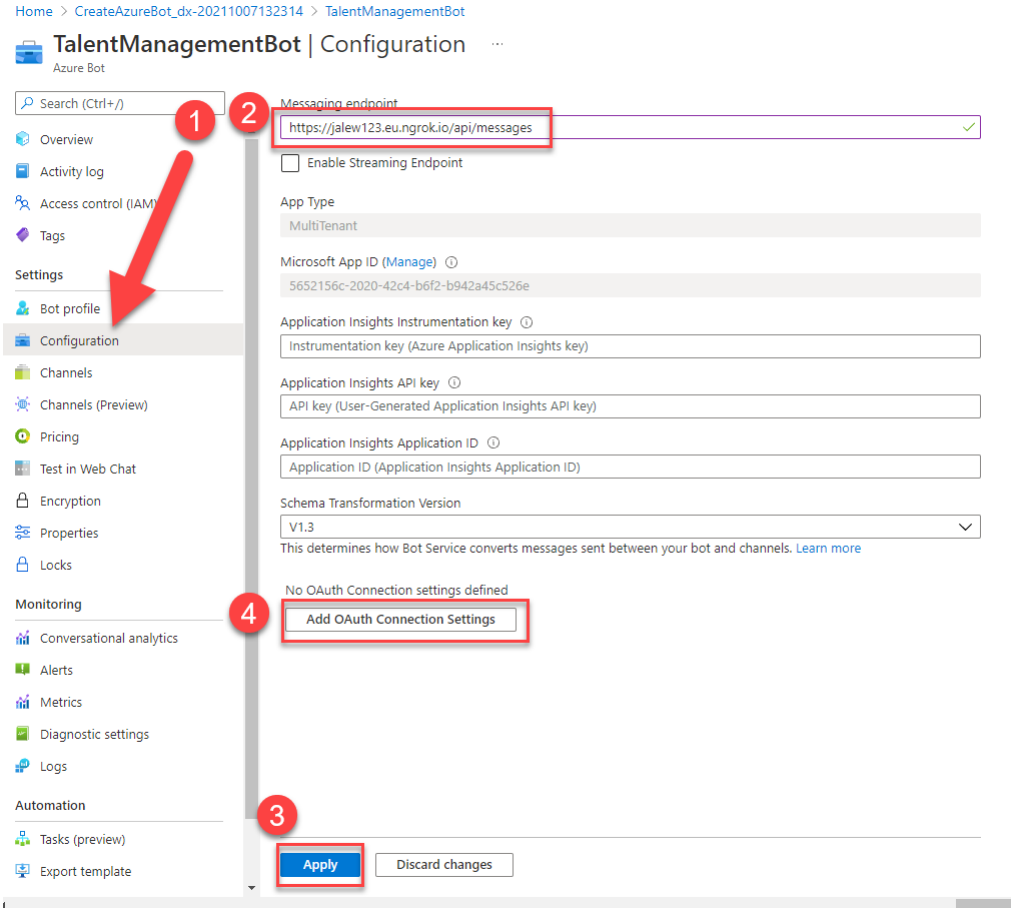
33. Once Validation has passed, select **Create**.



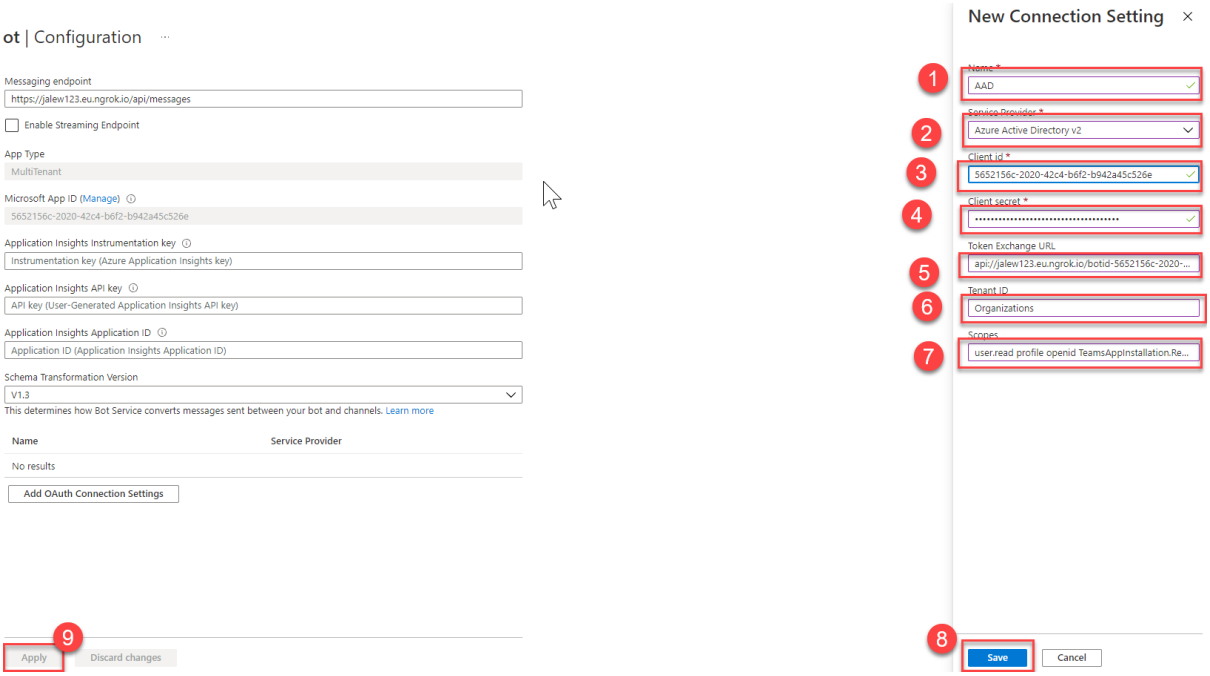
34. Select **Go to resource** once the resource has been deployed



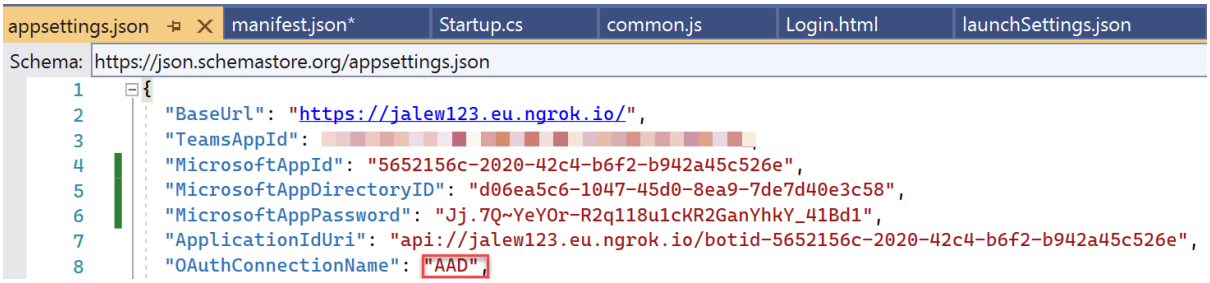
35. Select **Configuration**, enter the **Messaging endpoint** in the format of **NGROKURL/api/messages –** select **Apply** and then click **Add OAuth Connection Settings**.



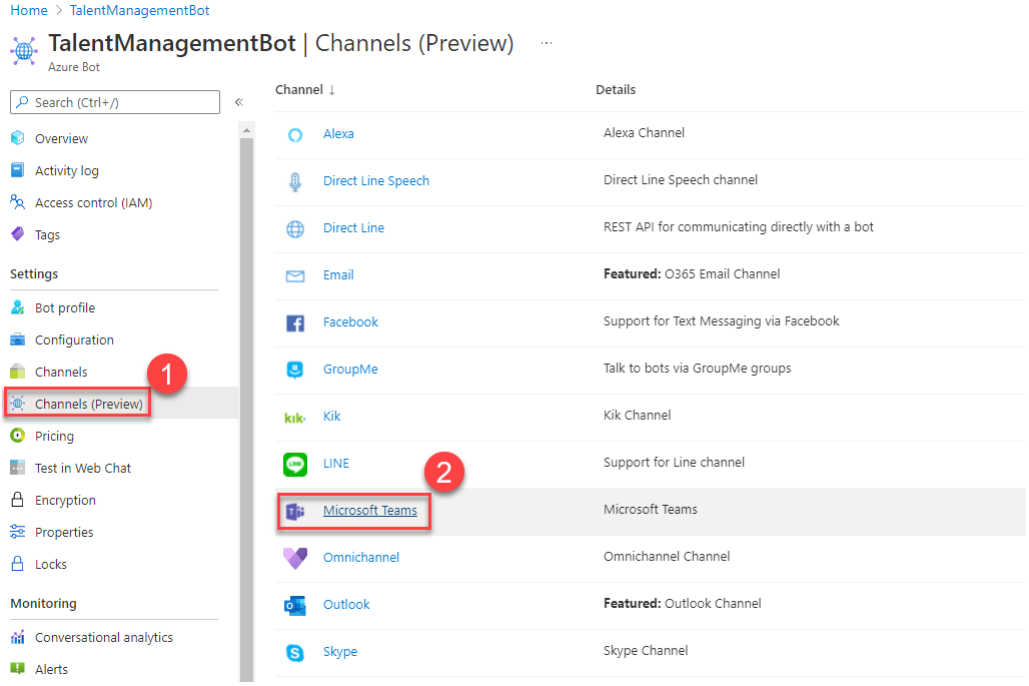
36. Enter the name as **AAD**, select **Azure Active Directory v2** as the Service Provider, the Client ID is your **MicrosoftAppID**, the Client secret is your **MicrosoftAppPassword**, the Token Exchange URL if your **ApplicationIDURI,** Tenant ID is **Organizations** and Scopes is **user.read profile openid TeamsAppInstallation.ReadForUser –** click **Save** and **Apply.**



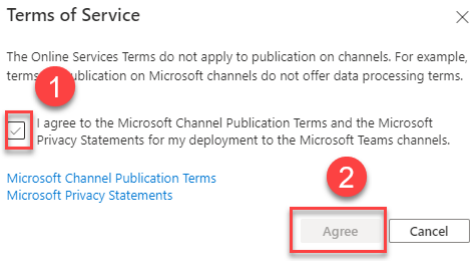
37. Back into Visual Studio, set the **OAuthConnectionName** value in appsettings.json (line 8) to **AAD –** click CTRL+S to save the appsettings file.



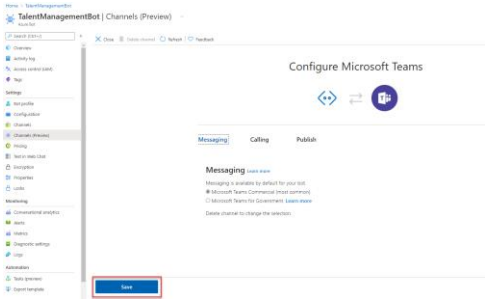
38. Now go back into the Azure Portal and configure the channels for the bot to support Teams. Select **Channels (Preview)** and select **Microsoft Teams.**



39. Tick the **checkbox** to agree to the terms of service and select **Agree**.

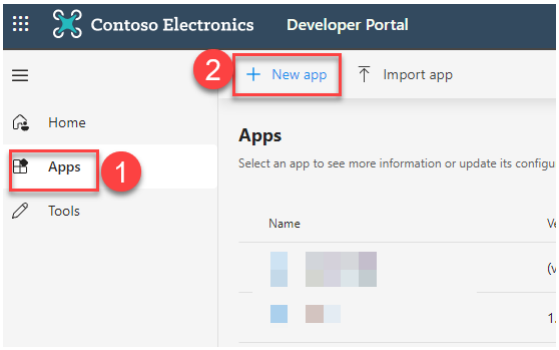


40. Click **Save**.



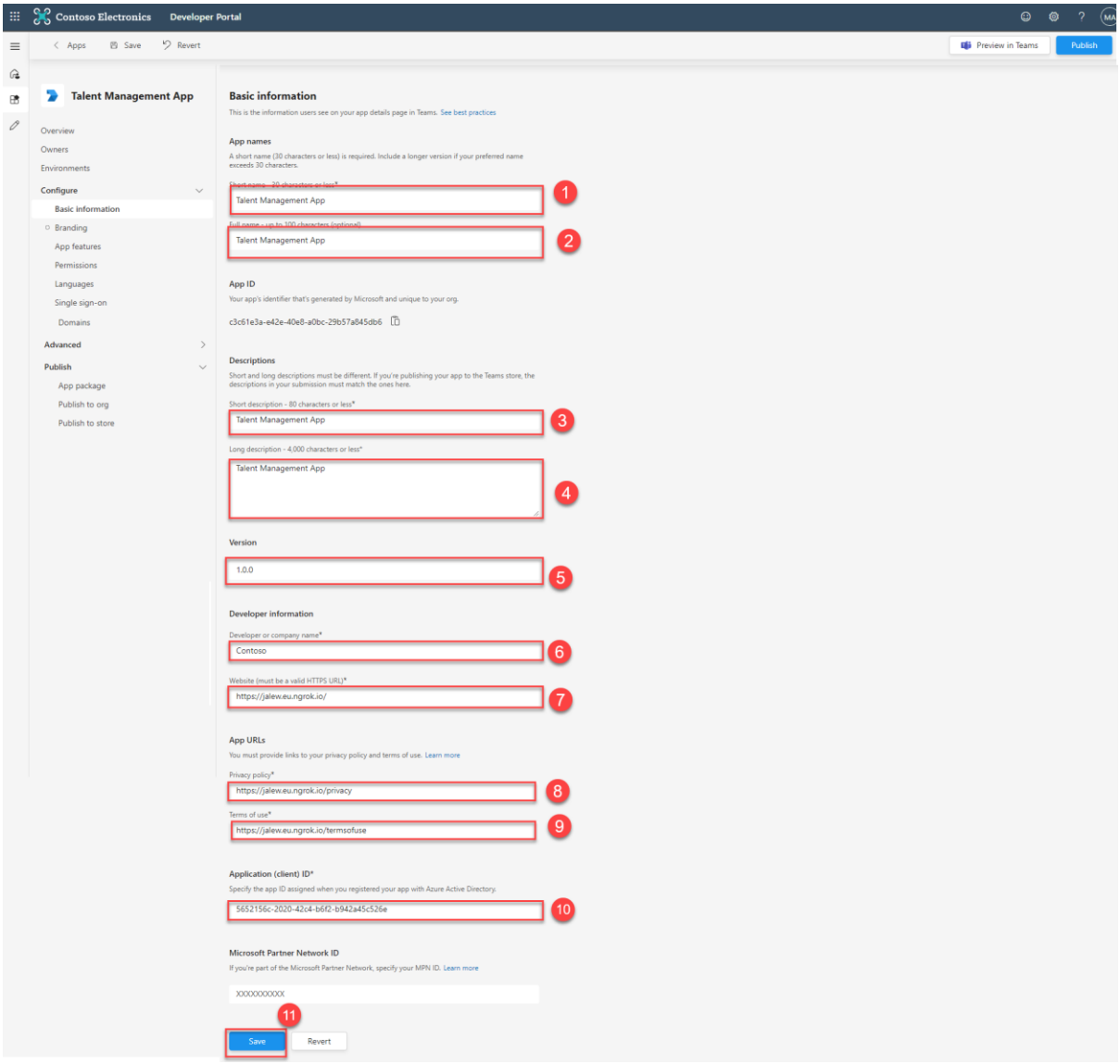
41. Your Bot is now configured to forward messages received onto your App (via the NGROK URL) and is configured to work in Teams. The final step, before running the app locally, is to create a Teams App Manifest and populate the final value (TeamsAppID) in the appsettings.json file. To do this go to the **Teams Developer Portal**

https://dev.teams.microsoft.com/ and sign in with the Test/Dev Tenant Administrator account. Select **Apps** and click **+ New App.**

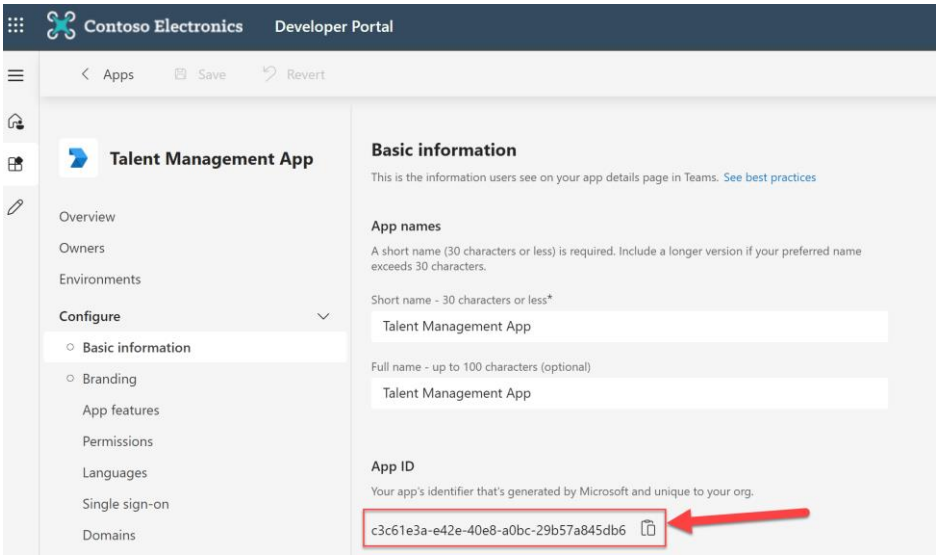


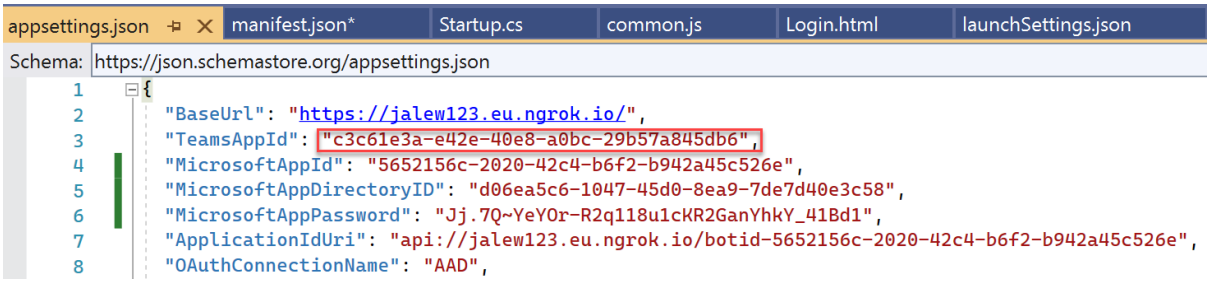
42. Enter the name **Talent Management App** and click **Add.**

43. We now need to enter some placeholder values into the Basic Information configuration of your Teams app. Start with copying the Short Name **Talent Management App**, and paste it into the **Full Name, Short Description and Long Description** fields. Set the version as **1.0.0**. Developer to **Contoso**. Website to your **NGROKURL.** Privacy Policy URL to **NGROKURL/privacy** and Terms of Use URL to **NGROK/termsofuse.** For Application (Client) ID, enter your **MicrosoftAppID**. Click **Save**.

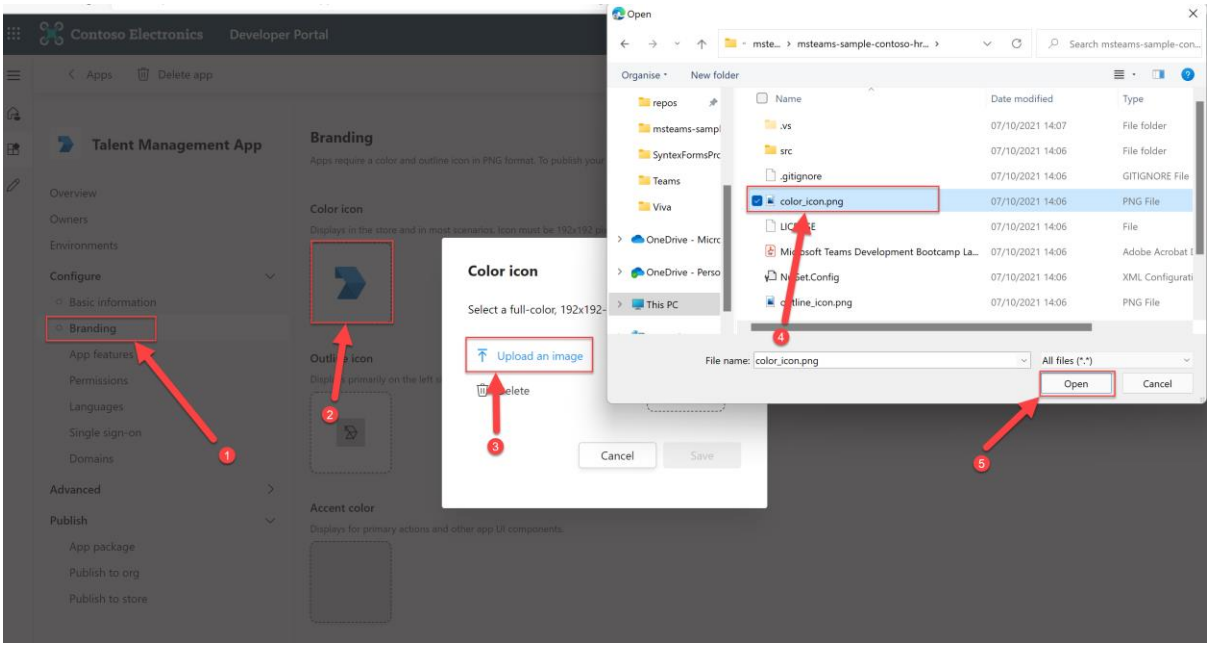


44. Copy the **App ID** and paste this into appsettings.json as the value of **MicrosoftAppID** (line 3). Save appsettings.json.

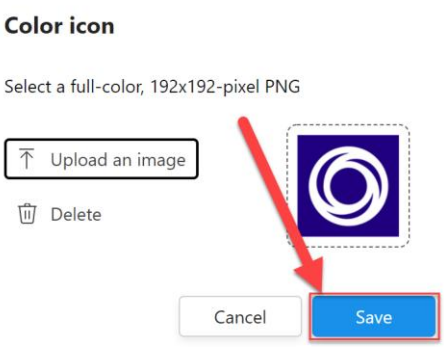




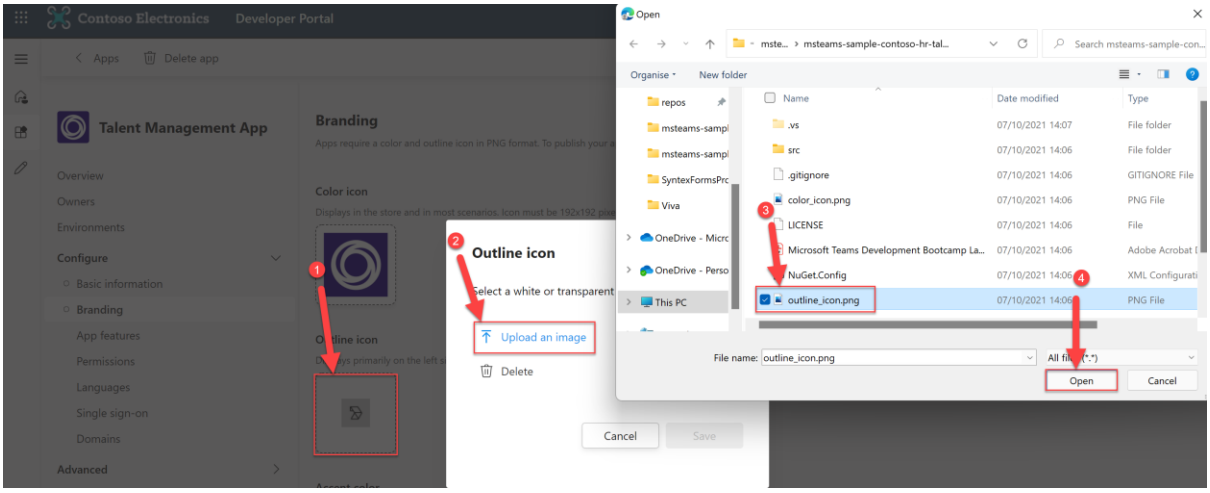
45. Go back into the Teams Developer Portal, and select **Branding,** click the **Colour icon**, select **Upload an image**, navigate to the folder that contains the application you downloaded from GitGub, select the **color_icon.png** file and click **Open**



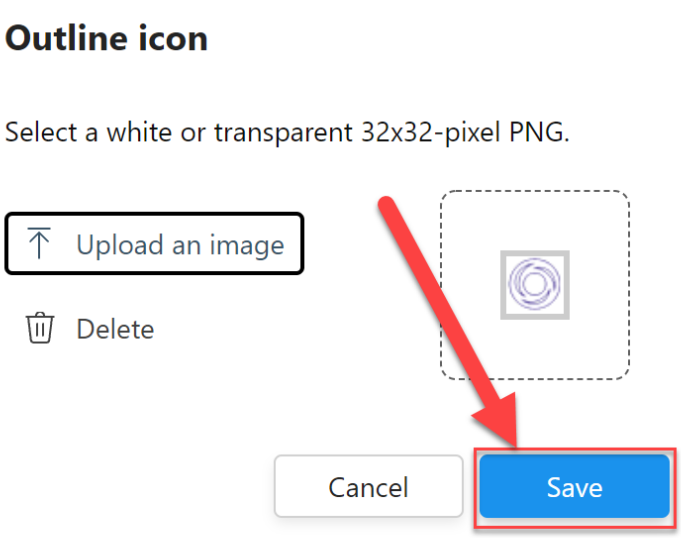
46. Click **Save**



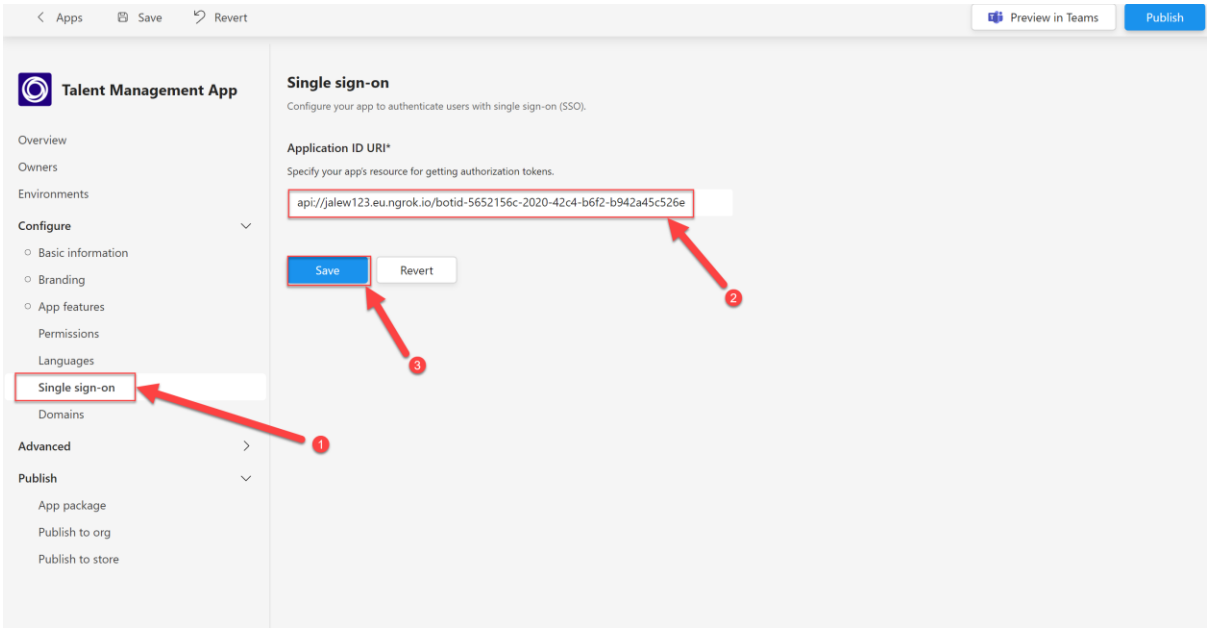
47. Click the **Outline icon,** select **Upload an image**, navigate to the folder that contains the application you downloaded from GitHub, select the **outline_icon.png** file and click **Open**



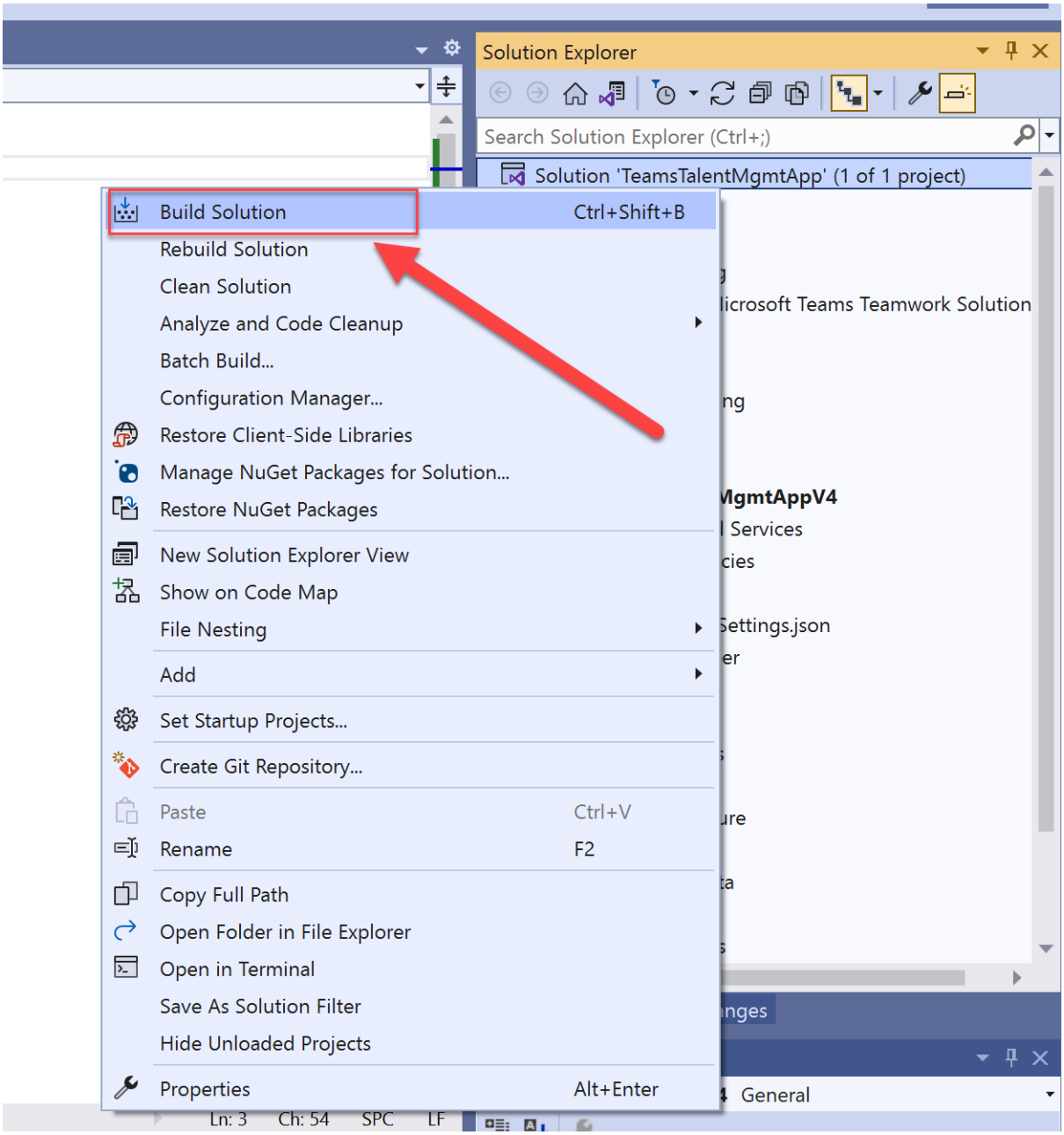
48. Click **Save**



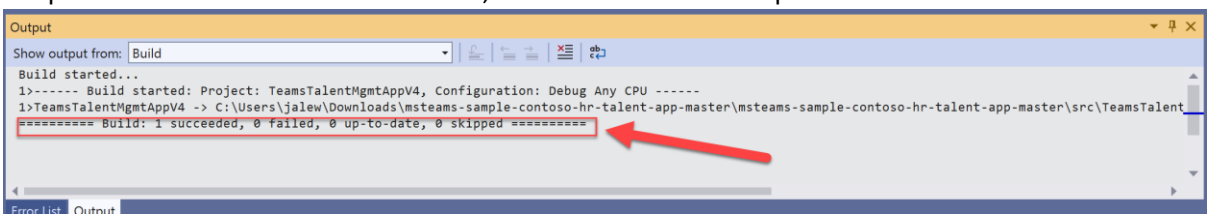
49. The final step to configure in the Manifest, during the initial setup, is the Single sign-on properties. Select **Single sign-on**, enter the **ApplicationIdURI** value into the textbox and click **Save**.



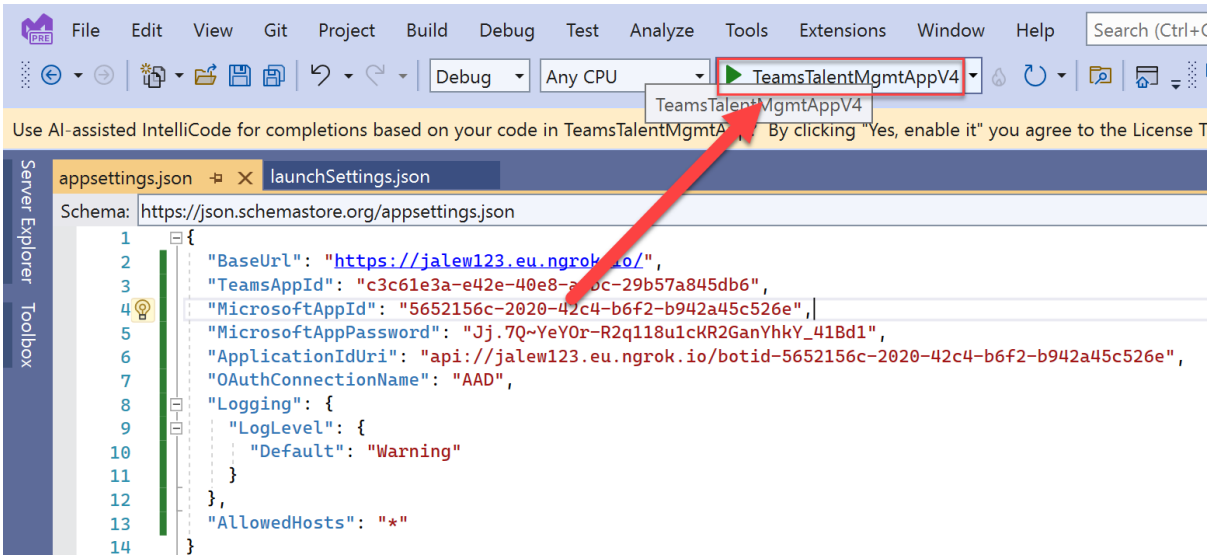
50. Now that the manifest has been configured and all 7 settings in appsettings.json have values associated to them, we are in a position to be able to build and run the app locally, using Visual Studio. **Right click the Solution** and click **Build Solution**.



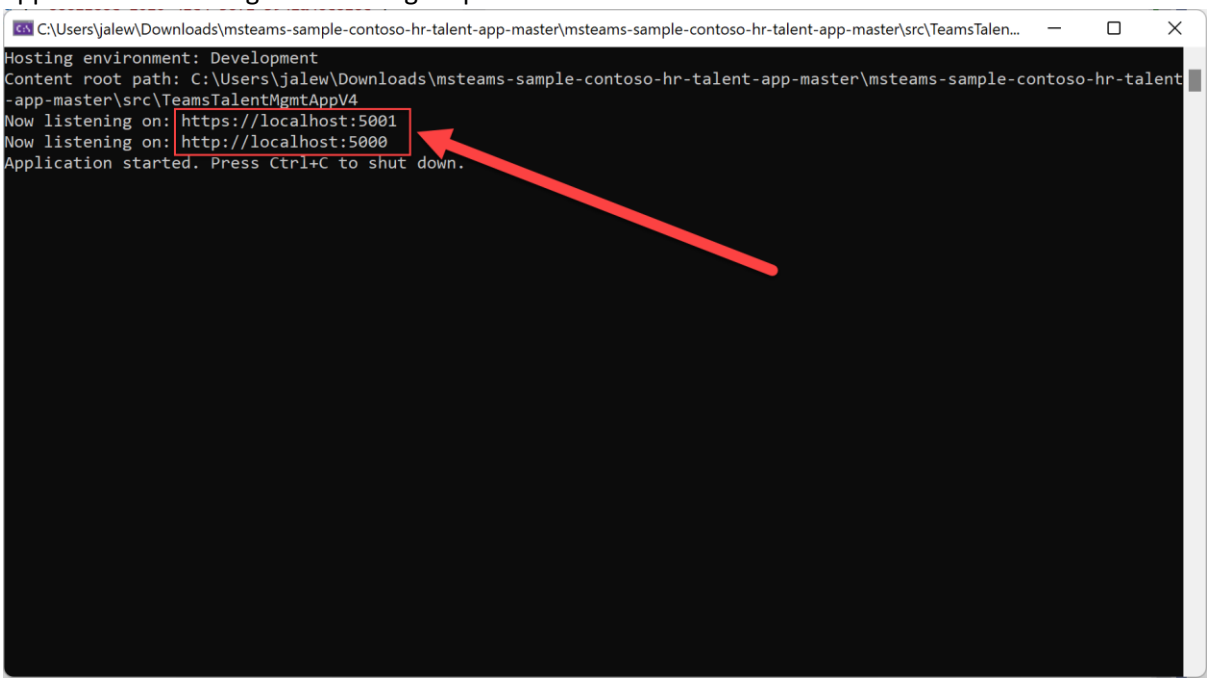
51. At the bottom of Visual Studio, the Output section will begin populating with text. Once the Output confirms that the build succeeds, move onto the next step.



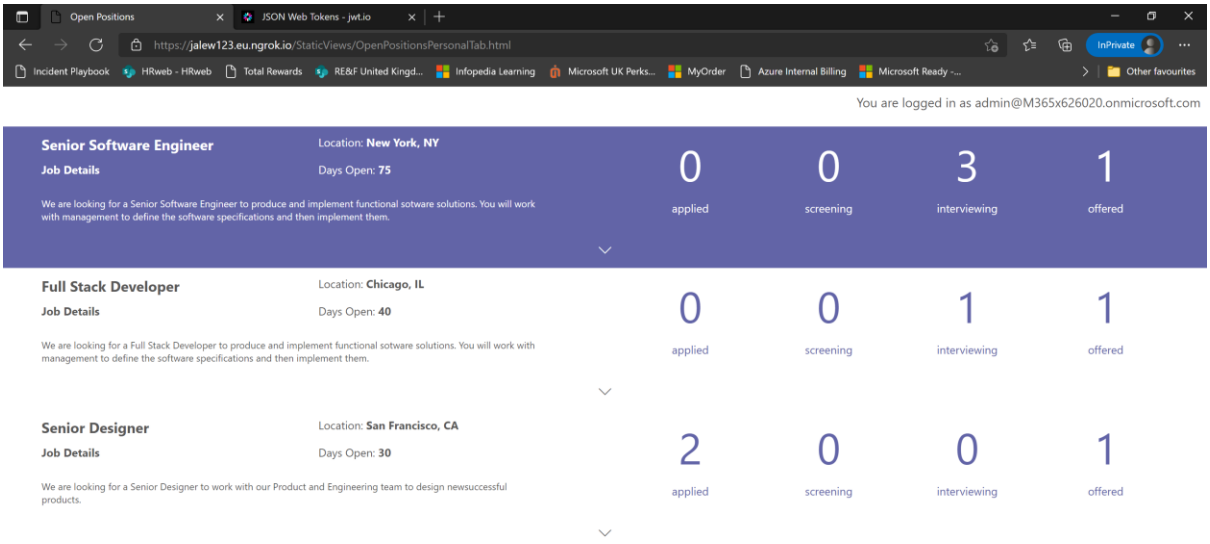
52. Click the **Play** button, to run (debug) the Talent Management App locally.



53. All being well, you should receive a message in a console window, confirming that the application is running and listening on ports 5000 and 5001.



54. The NGROK tunnel that we established earlier, should be forwarding traffic it receives on 443 (https) to localhost:5000. We can test this by going to the following URL in an InPrivate browser window: **NGROK/StaticViews/OpenPositionsPersonalTab.html** – you should then be sent through an interactive Login and Consent process via Azure AD, and presented with the following dashboard view:

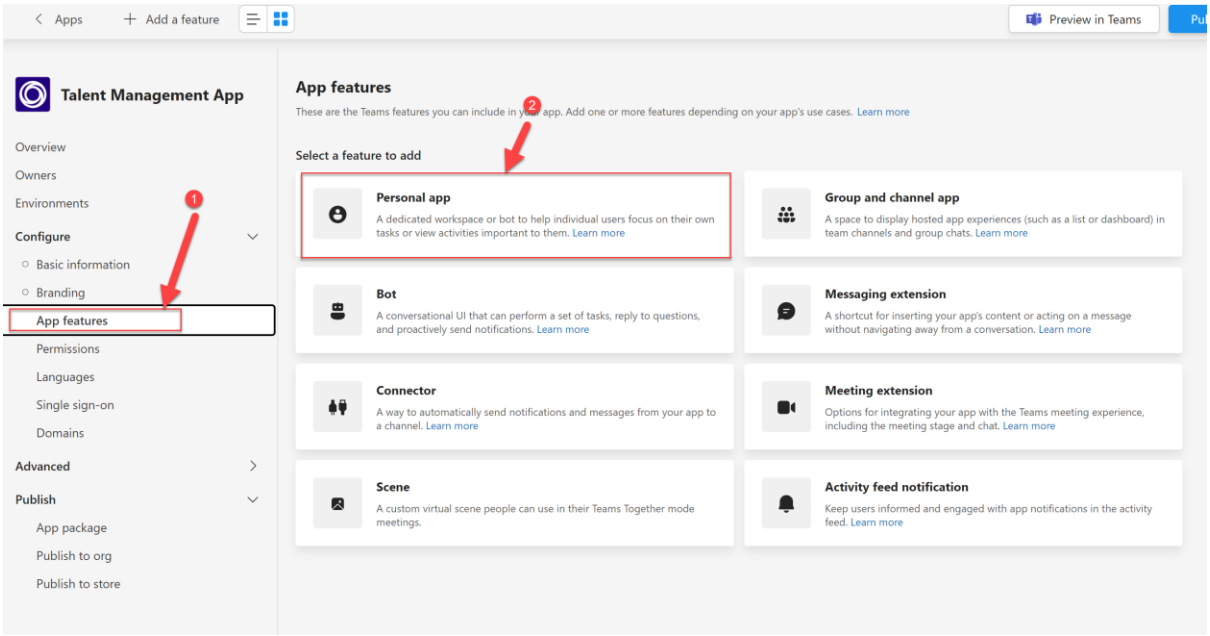


Nice job! You have setup the developer environment and Azure resources required to run this application locally. Now that the application is running and working outside of Teams. We can start to configure our Teams App Manifest to bring the apps functionality inside Teams. We'll start with this Open Positions dashboard view in the next lab!

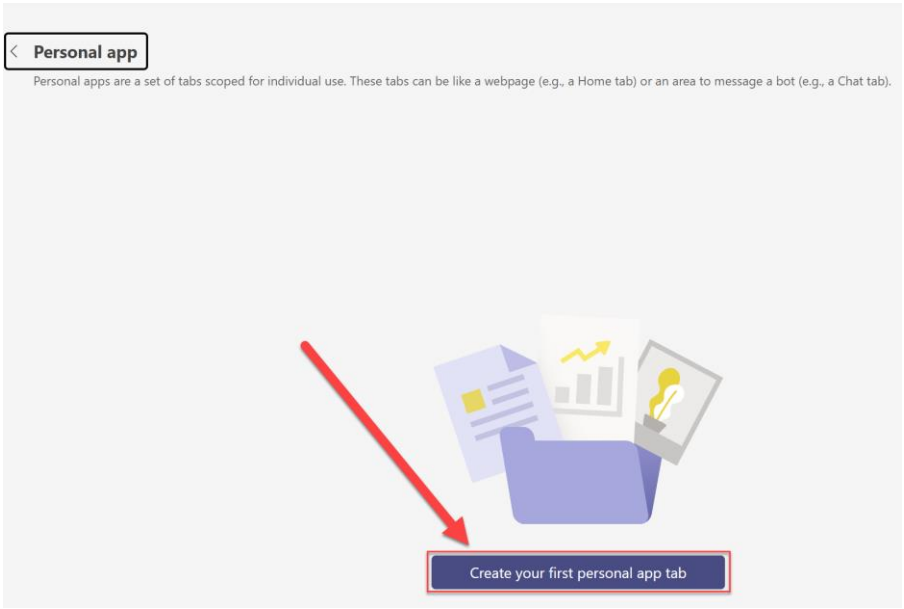
2) Implement a Tab inside a Personal App

In this step we will configure the Positions dashboard to work inside Teams as a Tab inside a Personal App. We will also see Teams SSO in action and review the code that allows this all to work!

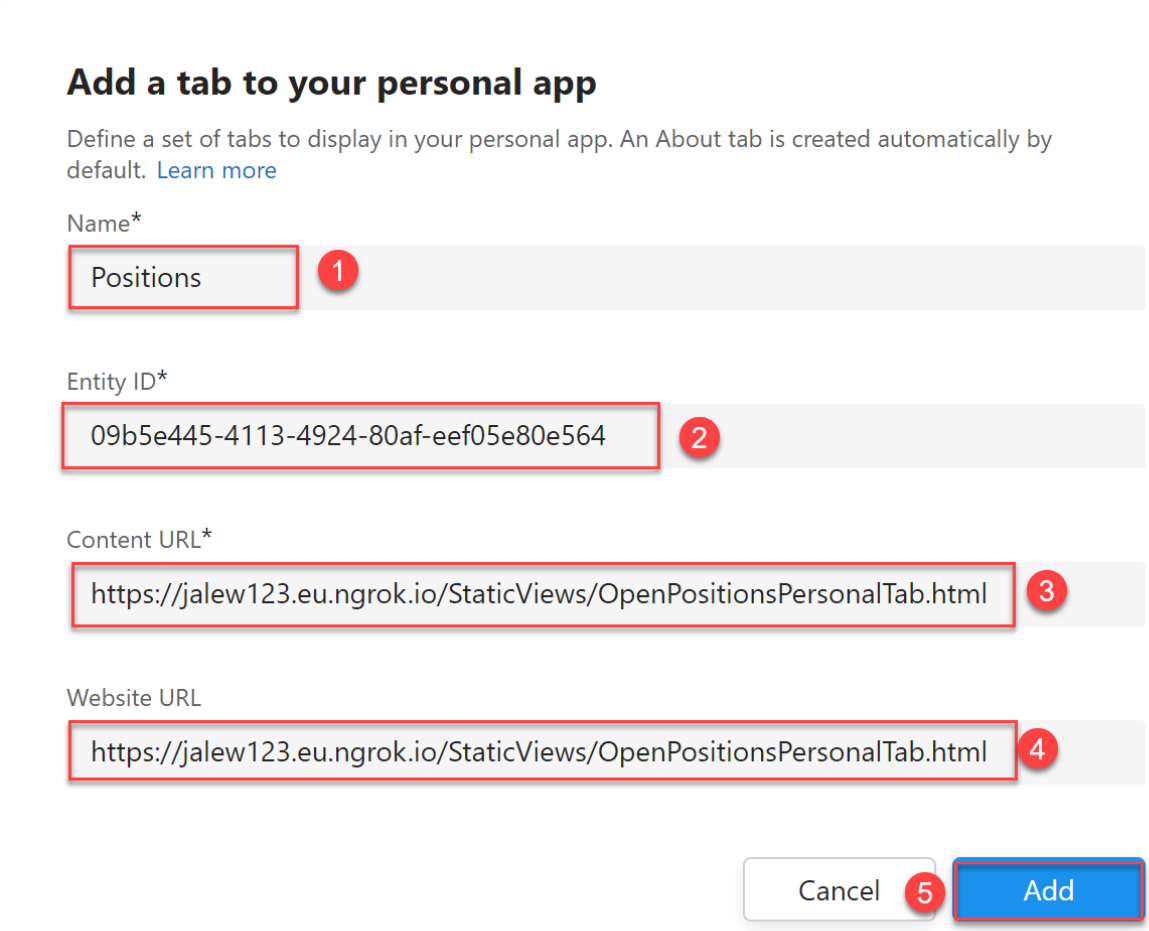
1. Go back to the Teams Developer Portal and open your Manifest. Select **App Features**, and then select **Personal app**.



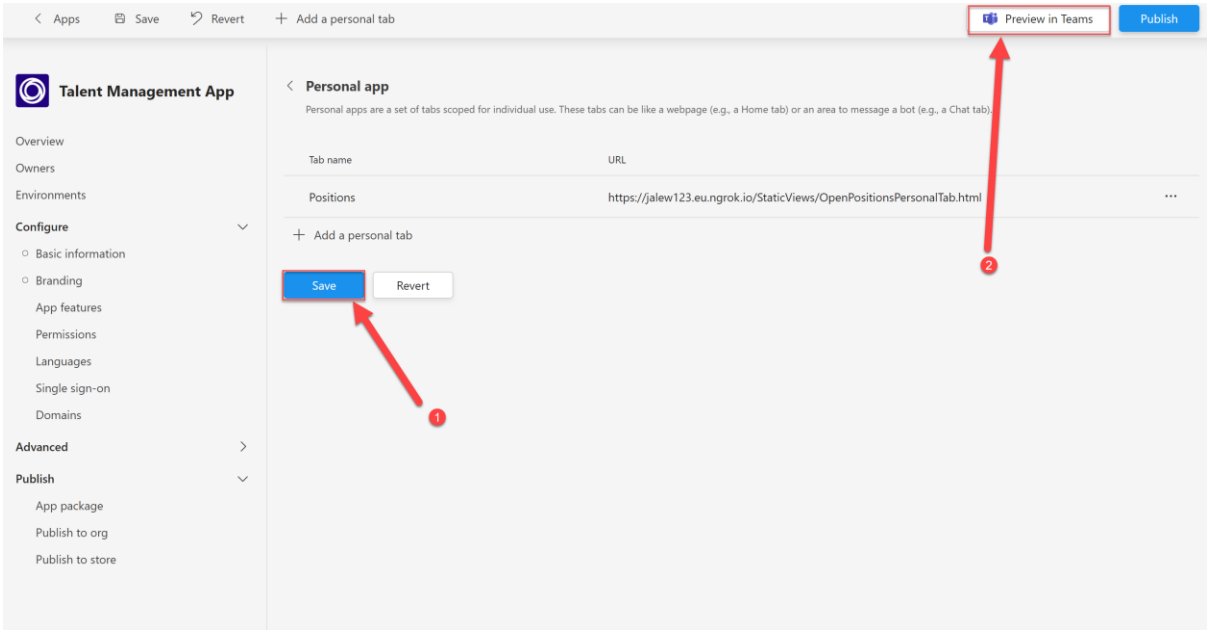
2. Select **Create your first personal app tab.**



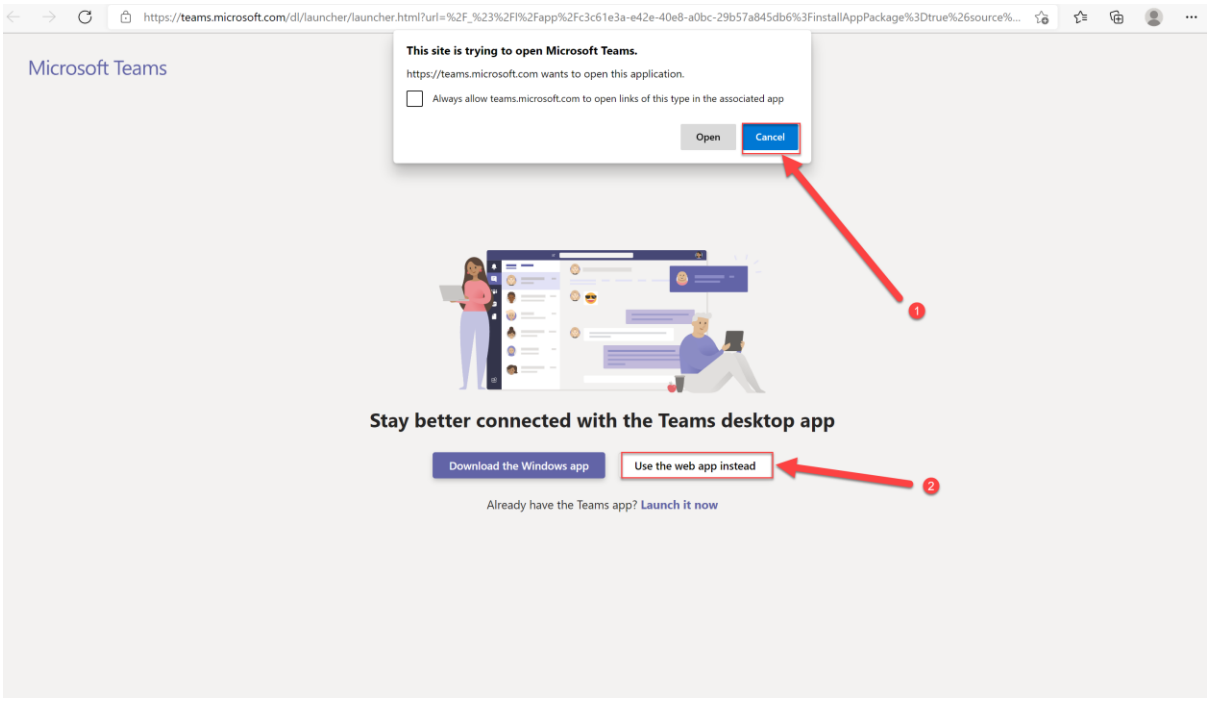
3. Enter the name **Positions**, leave the Entity ID as default, Content URL as **NGROK/StaticViews/OpenPositionsPersonalTab.html** and Website URL **NGROK/StaticViews/OpenPositionsPersonalTab.html** – The Website URL is what will be opened if the user wants to open the website outside of Teams. The Content URL is what will be used when the Tab is loaded inside Teams.



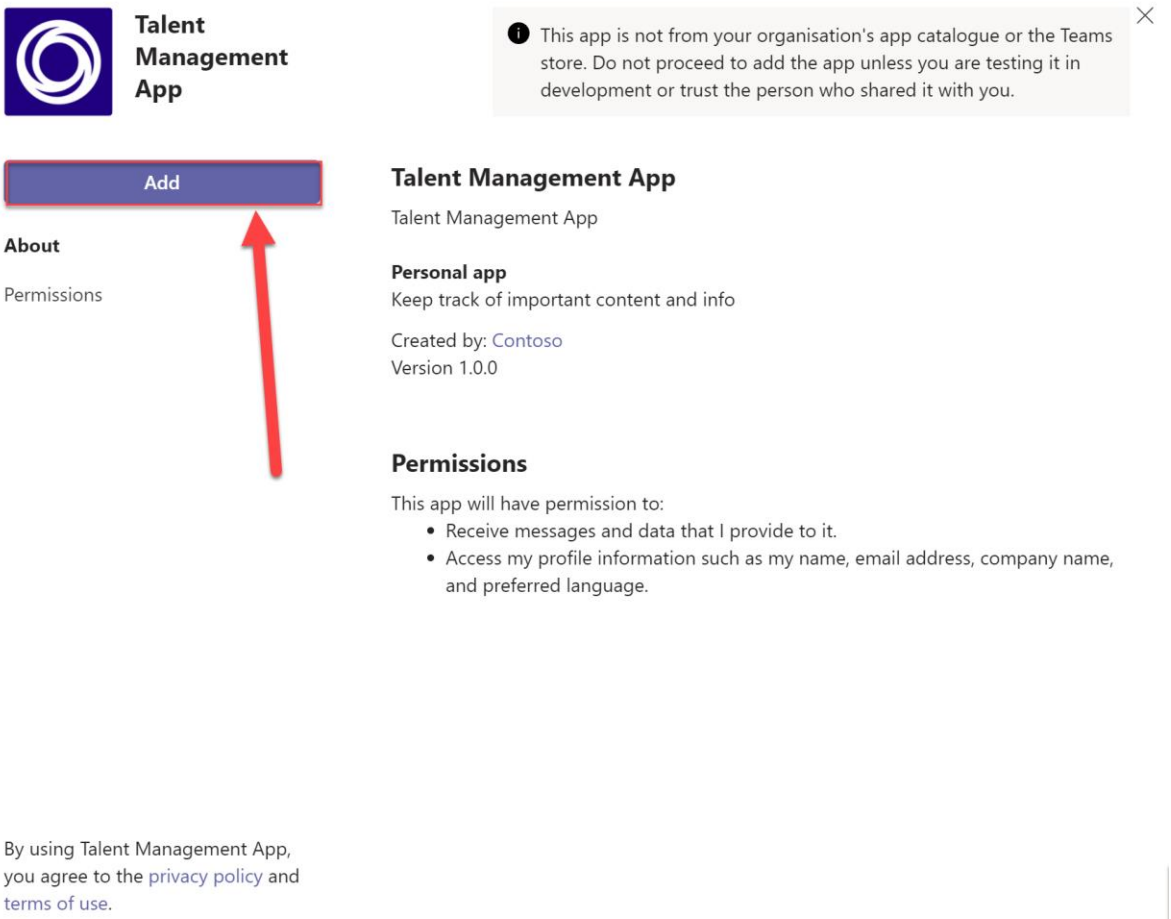
4. Click **Save** and select **Preview in Teams**



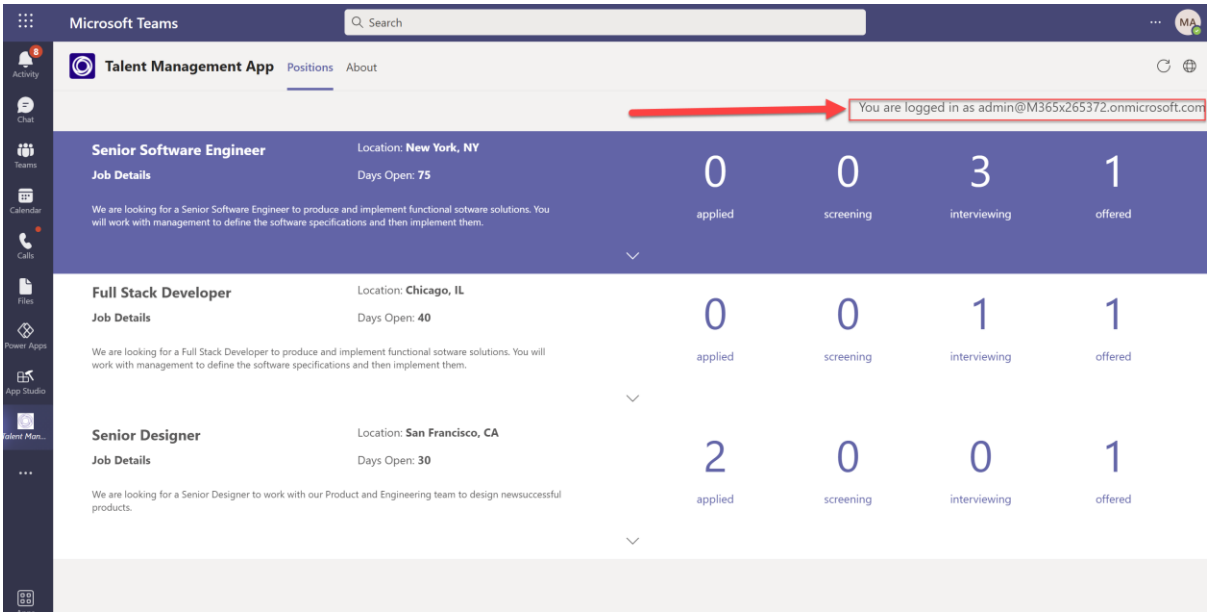
5. Select **Cancel** and click **Use the web app instead.**



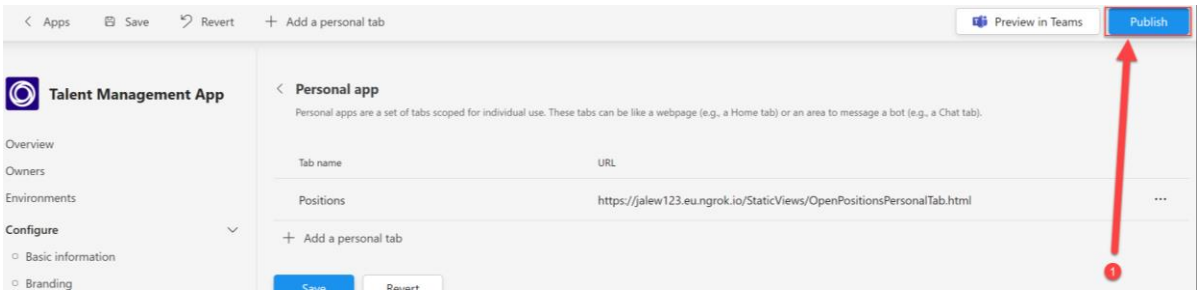
6. After a short wait, your app will appear in Teams. Select **Add.**



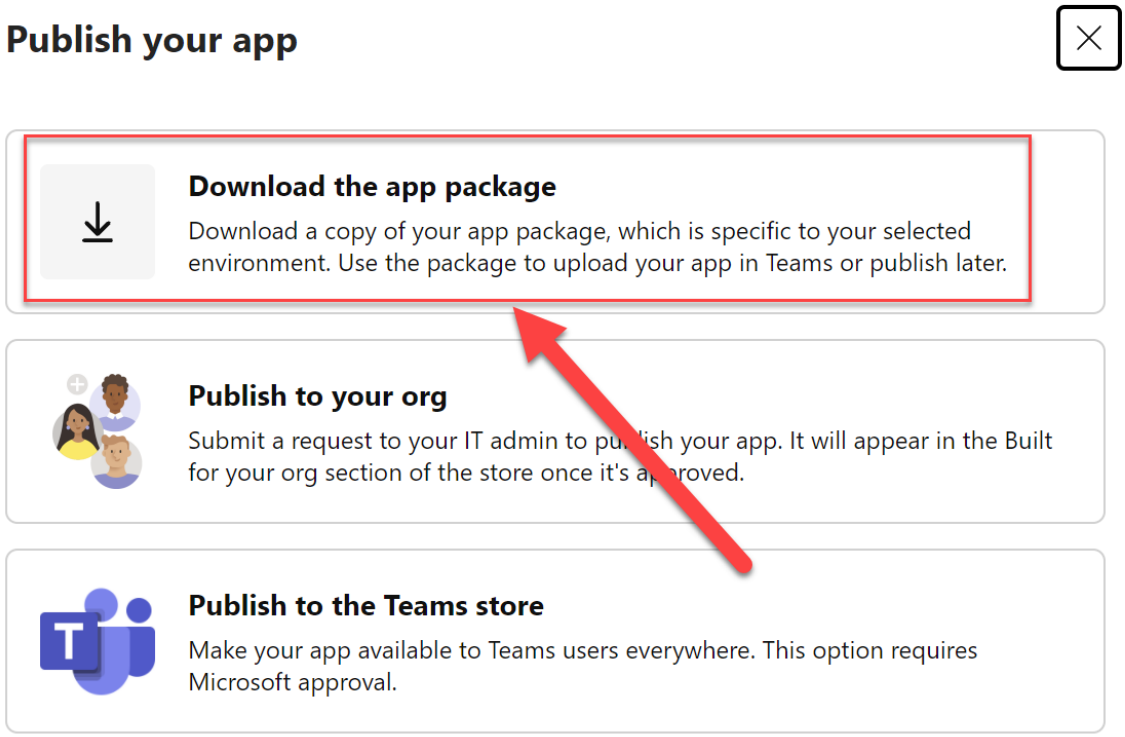
7. Once the Tab has successfully loaded, confirm that you are logged in as the user who is logged into Teams.



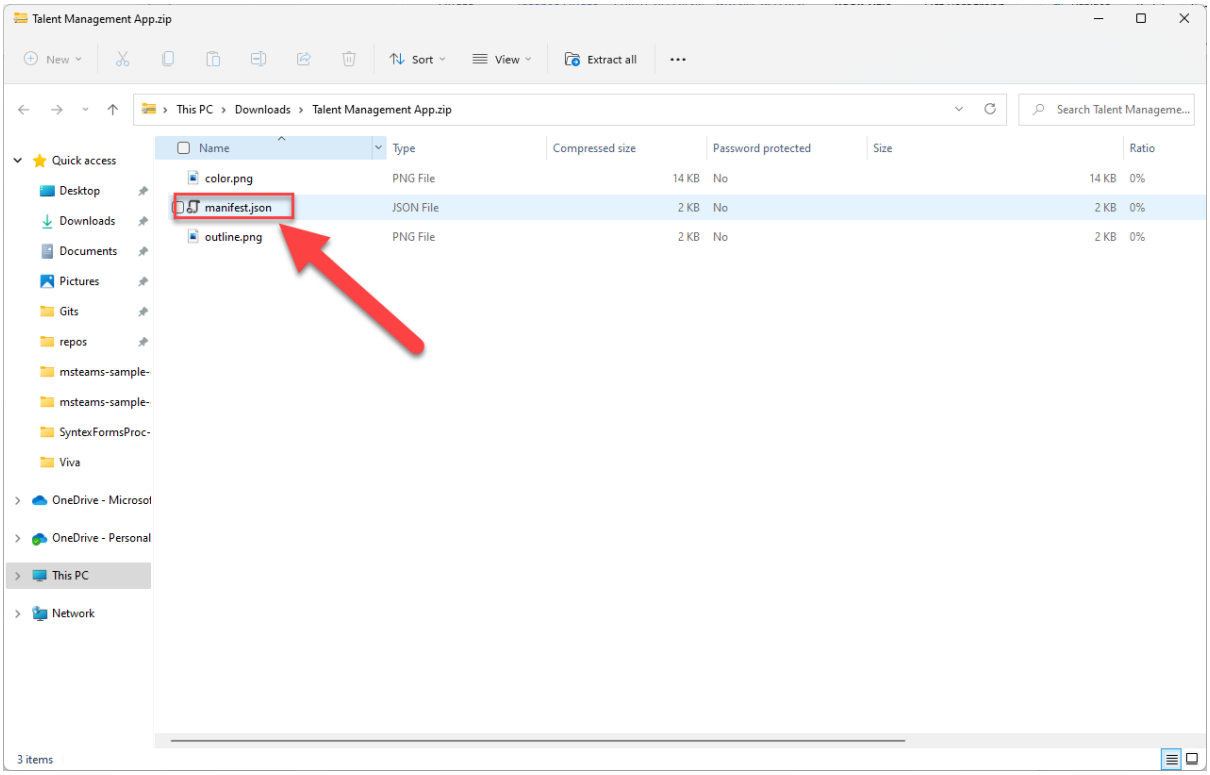
8. We will now review the App Manifest. Go back to the Teams Developer Portal, open your Manifest and select **Publish**.



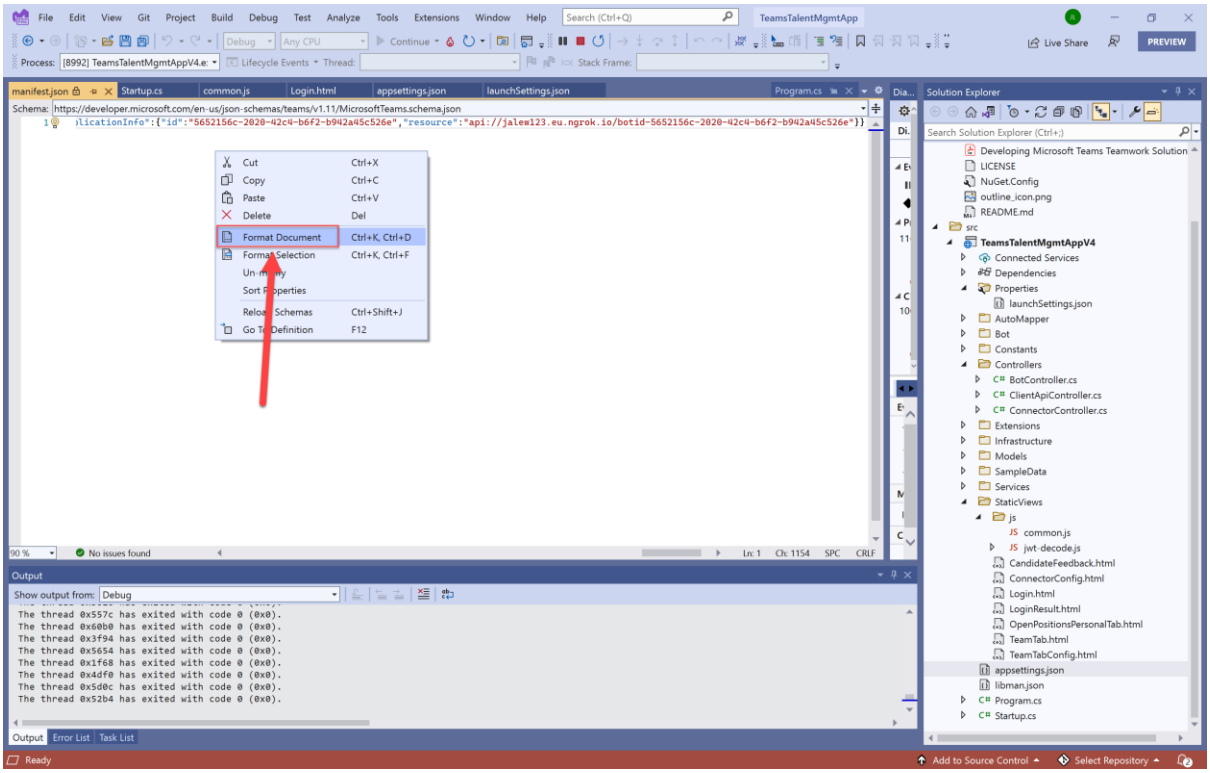
9. Click **Download the app package.**



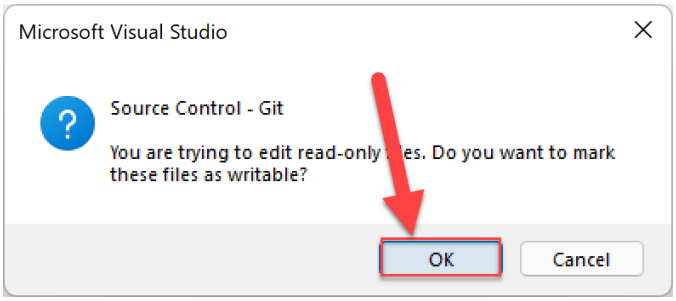
10. Open the ZIP that contains your manifest that has just been downloaded and open the **manifest.json** file in Visual Studio.



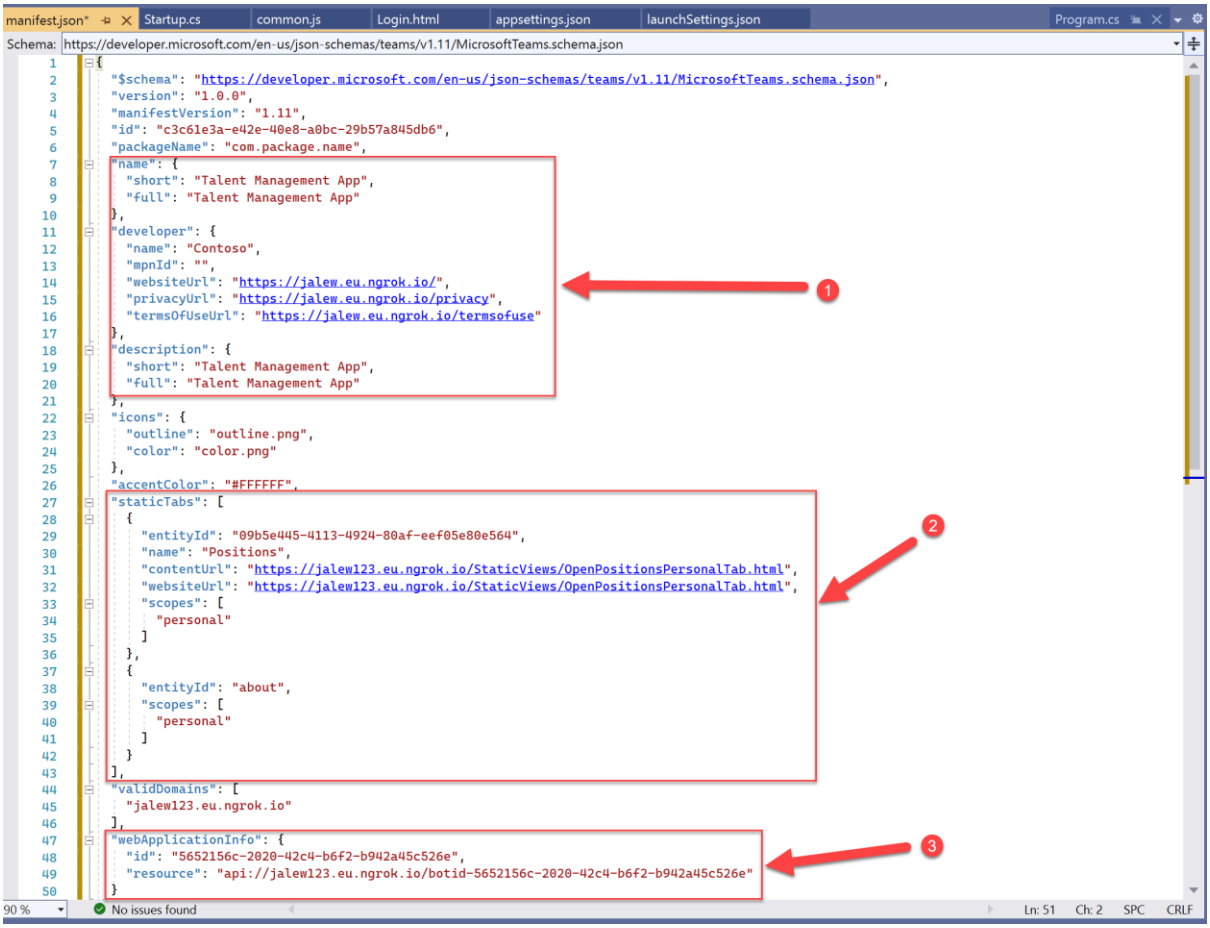
11. To improve readability, **right click the whitespace** in the manifest.json file and select **Format Document.**



12. If prompted, select **Ok.**

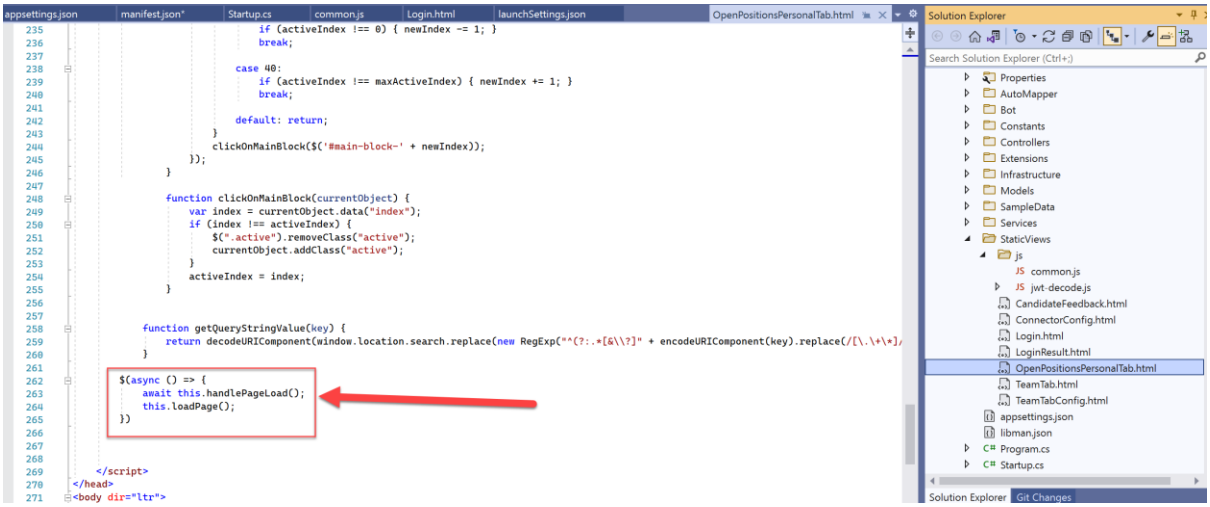


13. Review the settings and values in this manifest – the manifest is your Teams app! Firstly, see the name, developer and description settings and values. They contain the Basic Information you entered on the first page when you created your Teams app in the Teams Developer Portal. Secondly, review the Static Tabs array – this contains the contentUrl and websiteUrl settings and values. Finally, the webApplicationInfo settings and values contain your Azure AD and SSO information, that will be used when the page is loaded to perform Single Sign-On. This is explained in more detail in the next few steps.

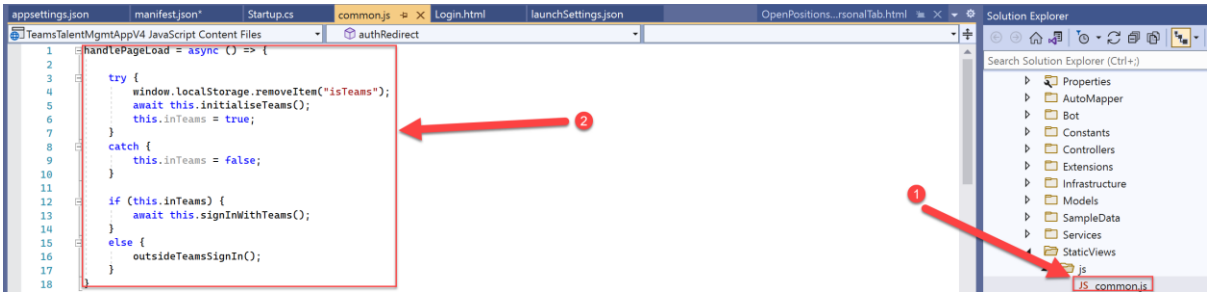


14. Go back into Visual Studio, expand the Static Views folder and select the **OpenPositionsPersonalTab.html** file. Review the scripts that are used when this page is loaded, importantly the Teams SDK (line 7) and the common.js (line 11) are used here. The Teams SDK is used in web applications to enable interoperability between the Teams client and the web-app. It can be used to pass context, perform SSO and many other things. common.js is where all the sign-in and Teams related JavaScript resides. We will review this later.

15. For this specific page, we are using jQuery to control when the JavaScript functions are called. This is detailed on line 262, where we call and await the completion of **handlePageLoad** (which is found in common.js and handles authentication both inside and outside of Teams), before we call **loadPage** (which is what builds the page and calls APIs to get data for Positions).



16. Expand the **js** folder, found inside StaticViews, and open the **common.js** file. Here you can see the **handlePageLoad** function. We are using this to call other functions that then determine if the page is being loaded inside or outside of Teams. If the page is loaded inside Teams, then the function **signInWithTeams** is called, which then attempts to do a Teams SSO. If the page is outside of Teams, the function **outsideTeamsSignIn** is called, which will redirect the user to Azure AD. It's important to note that Azure AD (and many other identity providers) do **not** support iFraming of their authentication pages. As website inside Teams are iFramed, this means that we need a custom way of handling SSO. Hence why the flow of SSO is different when the page is loaded in Teams.

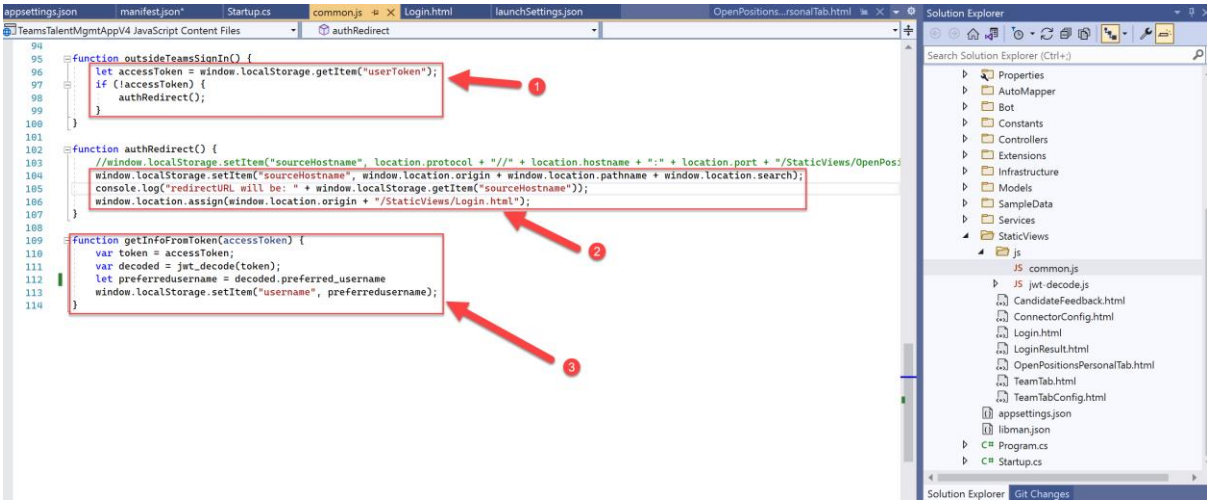


17. Scroll down and review the **getTeamsToken** function. This is the function that attempts to do a Teams SSO, by calling the **microsoftTeams.authentication.getAuthToken** function. This will take the Access Token that was used to sign-in to Teams, by the end user, and submit it to a back-end service that is operated by Microsoft. The back-end service will then try to swap the Teams Access Token, for an access token that has an Audience of your application. This should work based on the configured that you applied during the setup of the Azure AD App Registration, and will return an access token, with your MicrosoftAppId as the Audience in the claims of the access token, with a scope of access_as_user. **If successful, it will set the access token as an item in Local Storage for the Browser**, this can then be used by the page to call protected APIs to get Position and Candidate data. We will inspect this later on in the lab. The **teamsFallbackAuth** function is called if the SSO function fails, and will allow the user to perform an interactive sign-in to either provide Consent, or get an access token for use within the App.

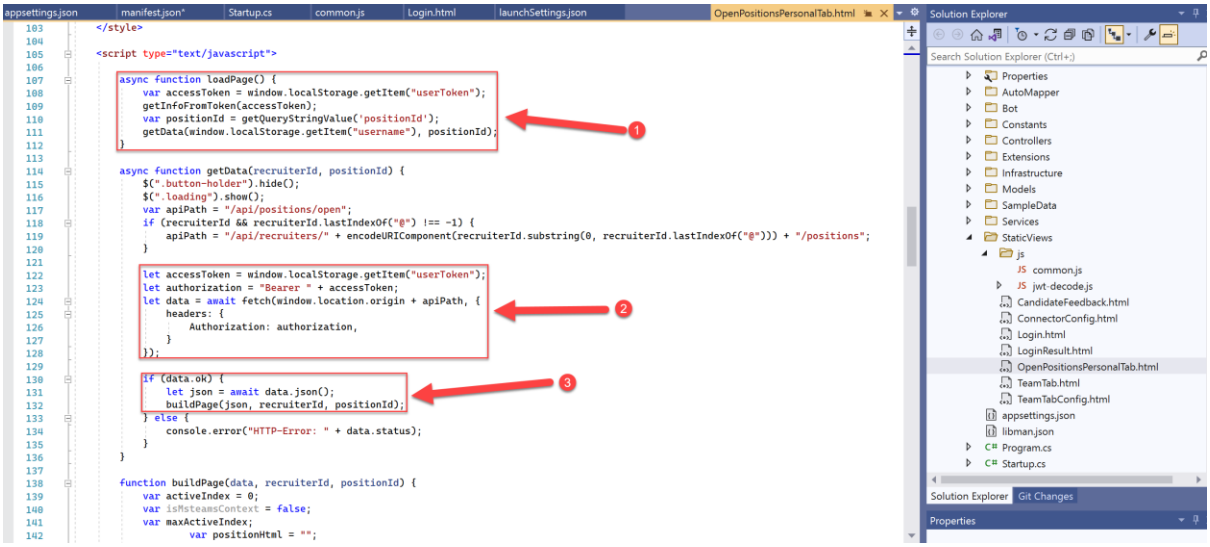
```
getTeamsToken = () => {
    window.localStorage.setItem("isTeams", "yes");
    return new Promise((resolve, reject) => {
        microsoftTeams.authentication.getAuthToken({
            successCallback: (token) => {
                window.localStorage.setItem("userToken", token);
                resolve(token);
            },
            failureCallback: (reason) => {
                reject(reason);
            }
        })
    });
}

function teamsFallbackAuth() {
    microsoftTeams.authentication.authenticate({
        url: window.location.origin + "/StaticViews/Login.html",
        width: 600,
        height: 535,
        successCallback: function (result) {
        },
        failureCallback: function (reason) {
            handleAuthError(reason);
        }
    });
}
```

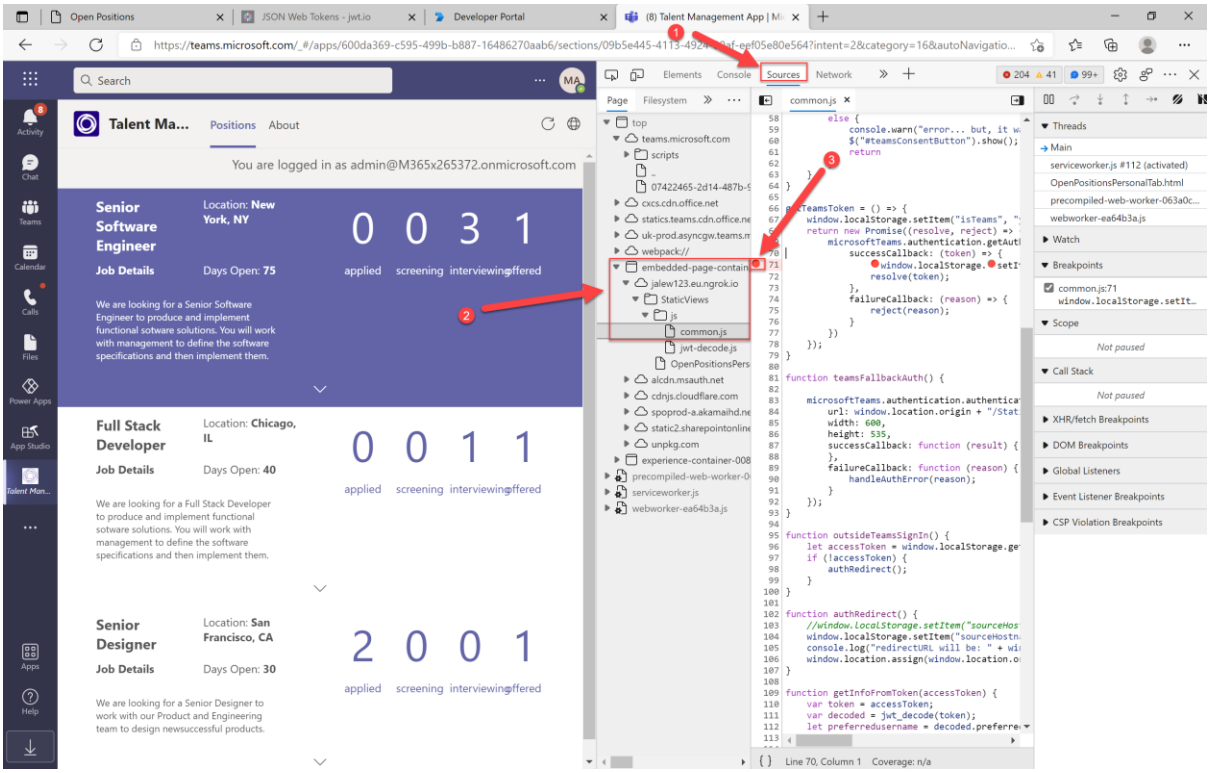
18. Scrolling to the bottom of the common.js file, you will see the **outsideTeamsSignIn** function. This checks for the existence of an access token in Local Storage, and if one doesn't exist, will call the **authRedirect** function. **authRedirect** will set some properties and then redirect the end-user to **Login.html**, which will then redirect the user to Azure AD for sign-in. Eventually, this will provide the user with an access token and they will return to the OpenPositionsPersonalTab.html page, but this time, the accessToken variable will contain an access token and the user will not be redirected the Azure AD. **getInfoFromToken** is called on the OpenPositionsPersonalTab.html page, and is used to pull out the claims (specifically the username attribute) for use when that page is loaded.



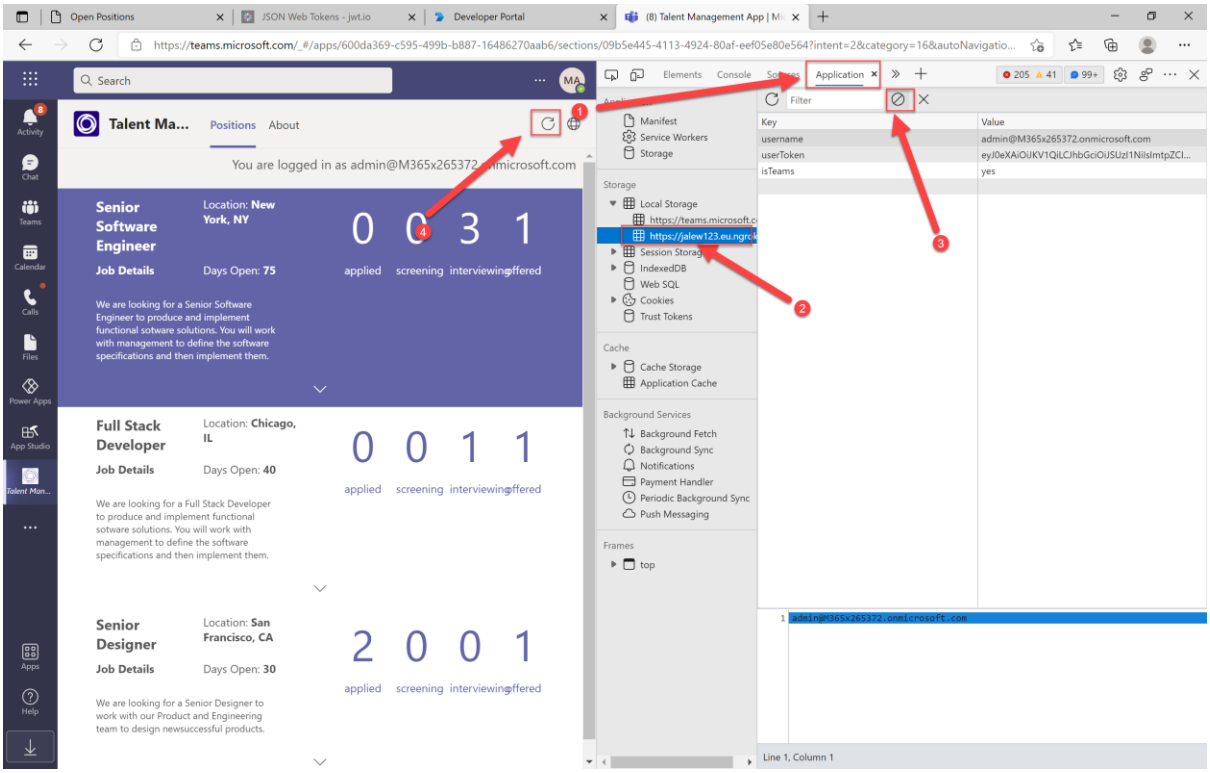
19. Go back to **OpenPositionsPersonalTab.html** and review the **loadPage** function. This will call the **getData** function. **getData** uses fetch within JavaScript to make a HTTP request, with the access token obtained from Azure AD in the header, to the apps Client APIs, this will return information about the available positions stored within the app. Once we have this information, and a 200 OK response is returned form the Client APIs, **buildPage** will be called, and the position data (stored in the **json** variable), recruiterId (which is your username) and positionId (which is obtained from the URL querystrings) is passed into this function – buildPage will then render the page as you see it!



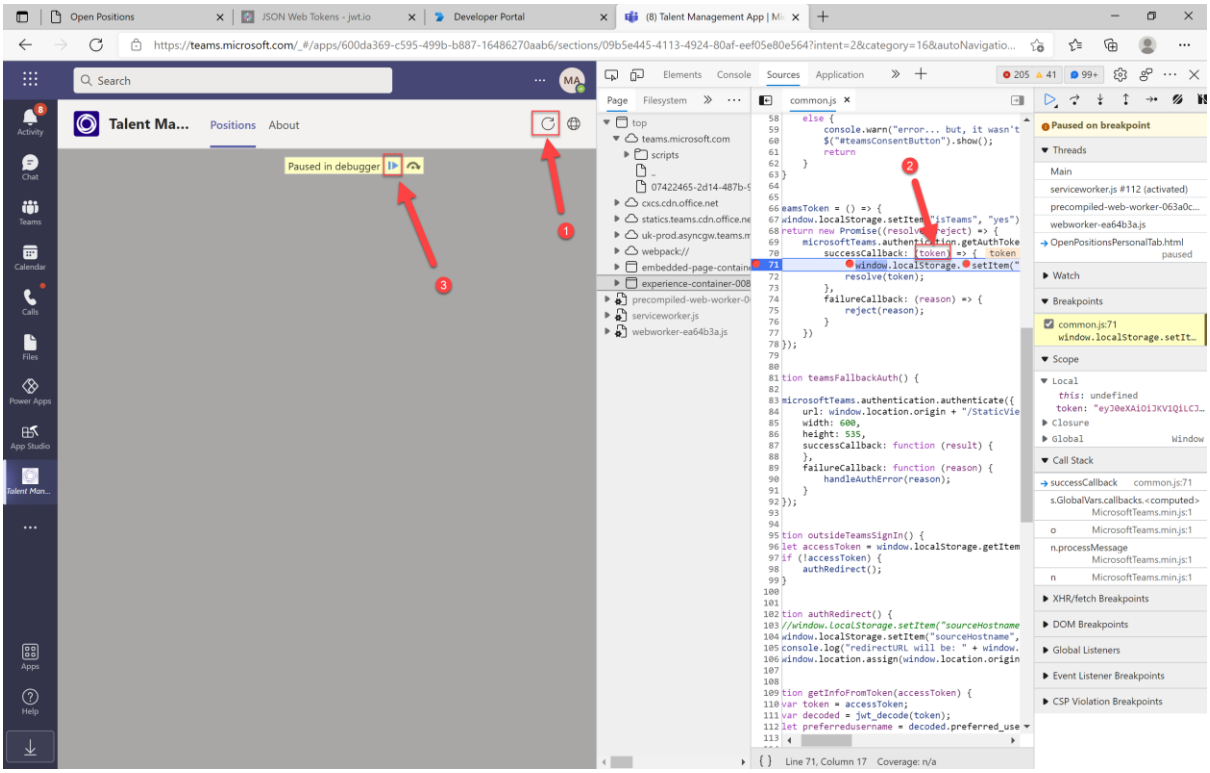
20. Let's see this in action! Go back into Teams and open the Talent Management App. Hit **F12** on your keyboard to open your browser's development tools. Select **Sources**, expand the folder hierarchy, until you find **common.js** (which is inside StaticViews/js folder, found within embedded-page/NGROK). Scroll down and click **to the left of line 71 to add a Breakpoint** (this will add a red dot to the left of the number).



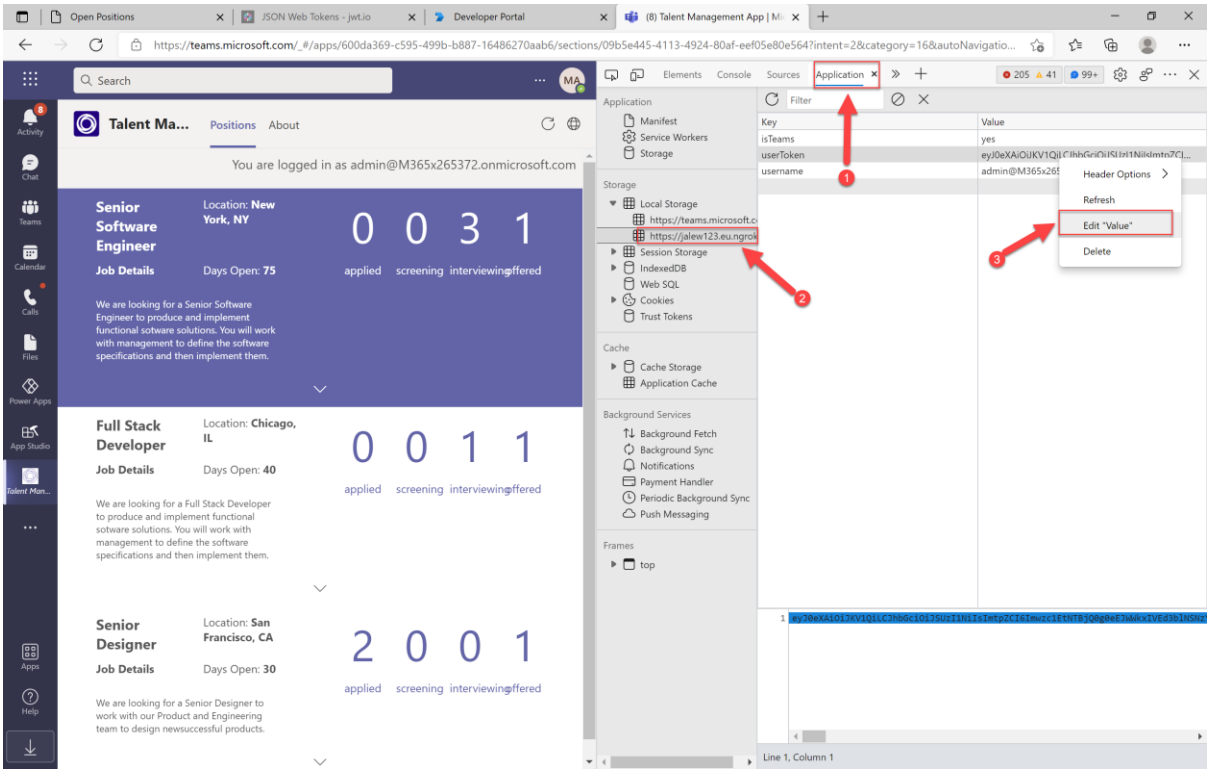
21. Before we re-load the page, it's a good idea to remove the items in Local Storage for this app. Click **Application**, expand **Local Storage** and select your **NGROK app**. Click the **clear button.**



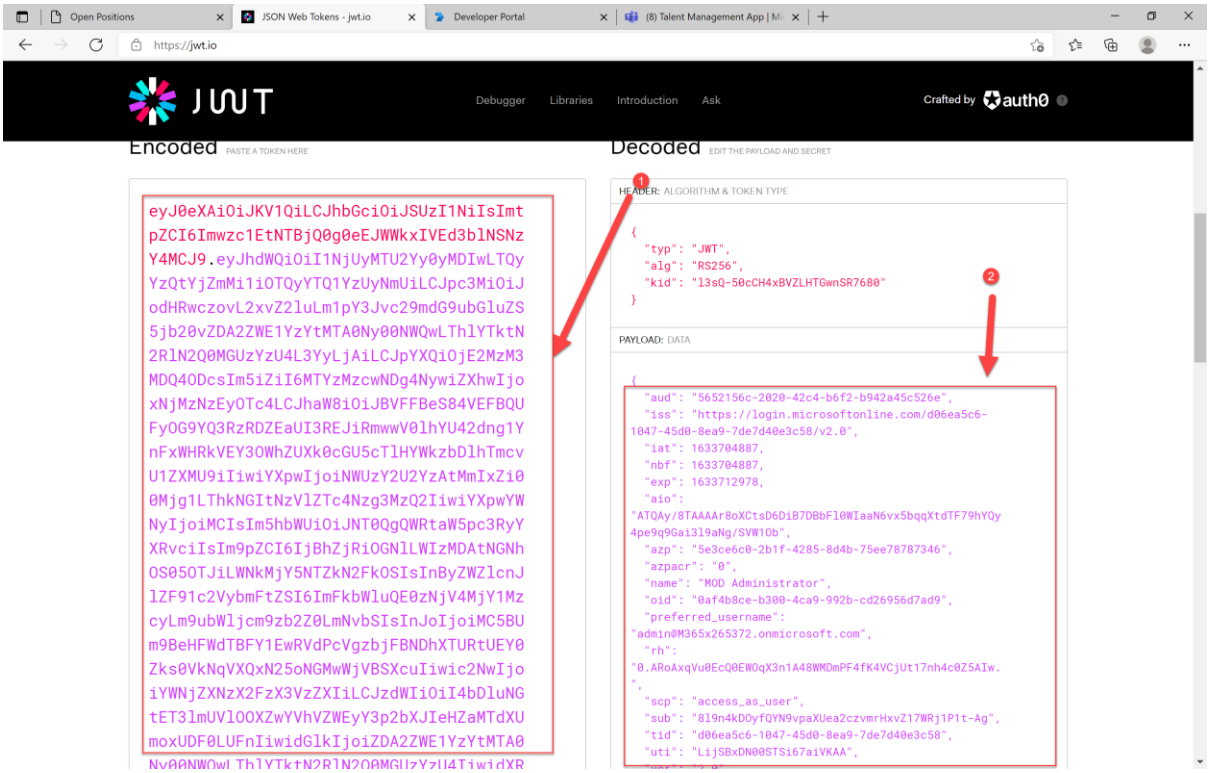
22. Hit the **Reload page** button. This will trigger the breakpoint. Mouse over the **Token** variable on line 70. This should then present you with an Access Token that has been retrieved via the Teams SSO method. Hit the **Continue** button to allow the page to complete loading.



23. Go back to the **Application** tab and select your **app** under **Local Storage.** Right click the **userToken** item, click **Edit** and **copy the value to your clipboard**.



24. Open a browser tab and go to https://jwt.io , **paste your access token** into the left input box and **review the claims within the token**. Importantly, you will see that it was Issued by Azure AD, the Audience is your MicrosoftAppId, the scope is access_as_user & the user is the current logged in Teams user.

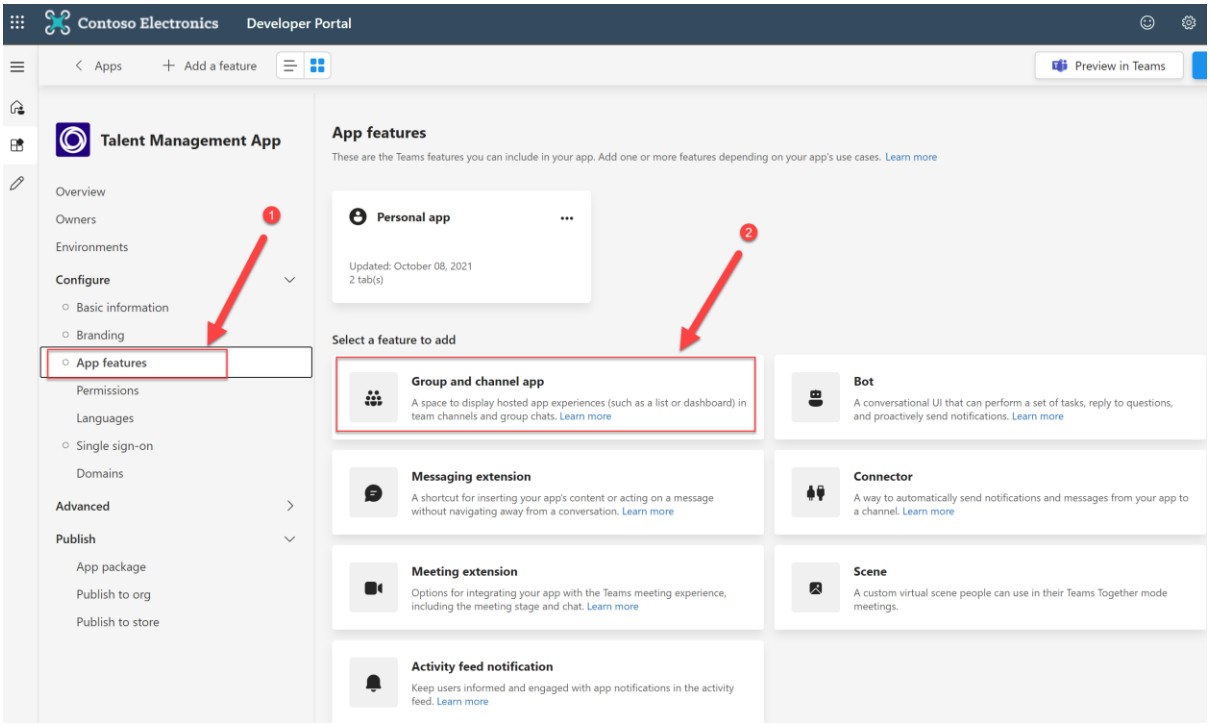


Great work! Now that the Personal Tab is working, and we have seen Teams SSO in action, let's move onto a Tab in a Team! These work slightly differently as they use a configuration page.

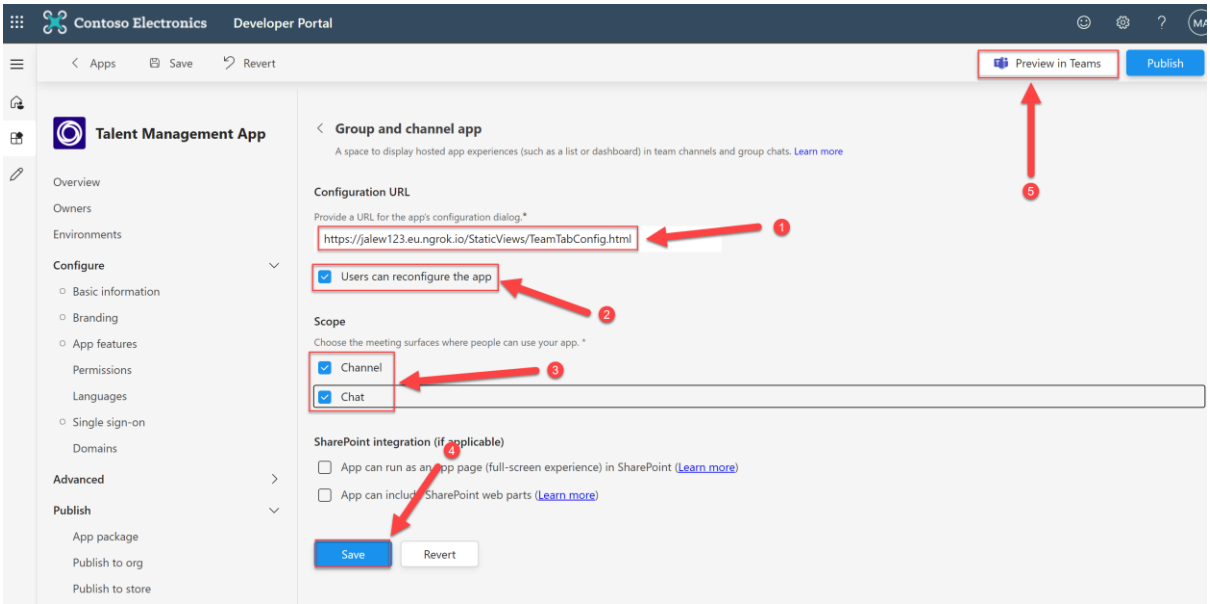
3) Implement a Tab inside a Team Channel

In this step we will implement a Team Channel Tab, which uses a special Configuration page, that we will inspect during this section of the lab. Channel Tabs are great for pinning specific information about specific objects that exist within your application and enable collaboration around that specific object. In this example, we'll be pinning a tab that contains details about an available position and the associated candidates that have applied for that position.

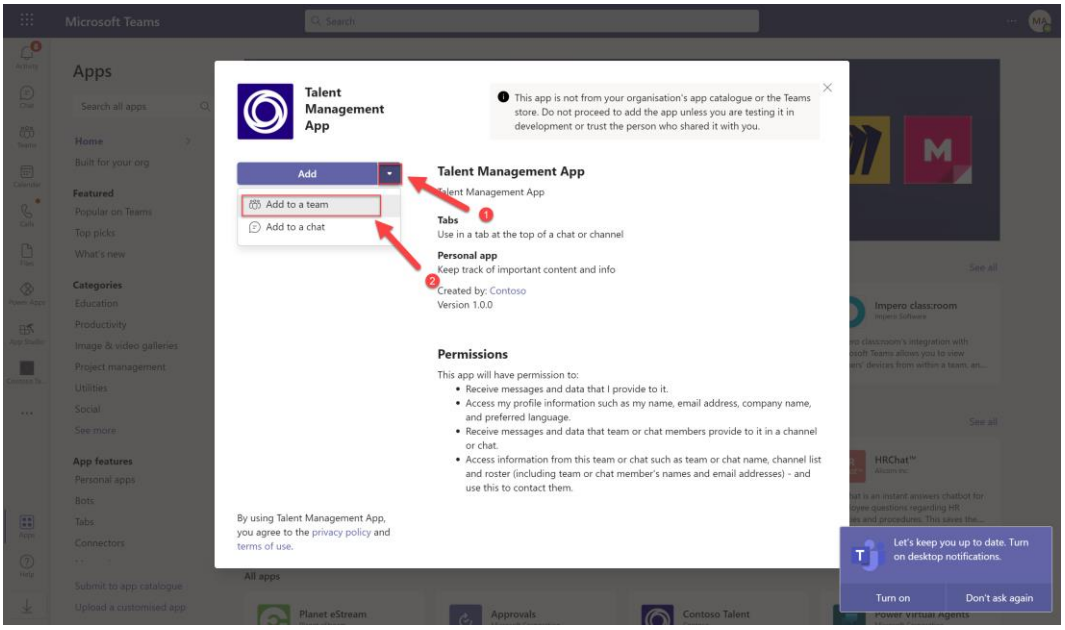
1. Go back to the Teams Developer Portal and open the Teams Talent Management App. Select **App Features** and click **Group and channel app**.



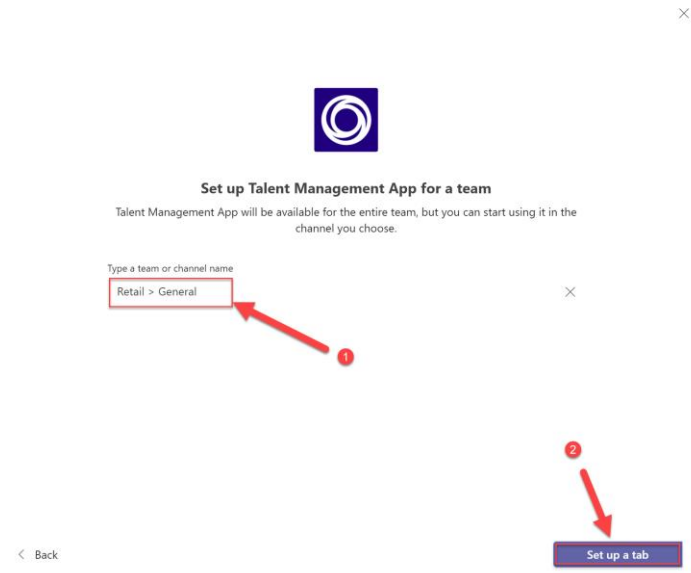
2. Enter the following into the Configuration URL **NGROK/StaticViews/TeamTabConfig.html** – select **Users can reconfigure the app** and select both **Channel and Chat** in the scopes. Click **Save.** For context, the Configuration URL is the URL that will open the Tab Configuration page, this is a special page that allows users to decide which Position you want to include in the Team Tab, we will inspect the JavaScript later in this lab that enables this. Click **Preview in Teams.**



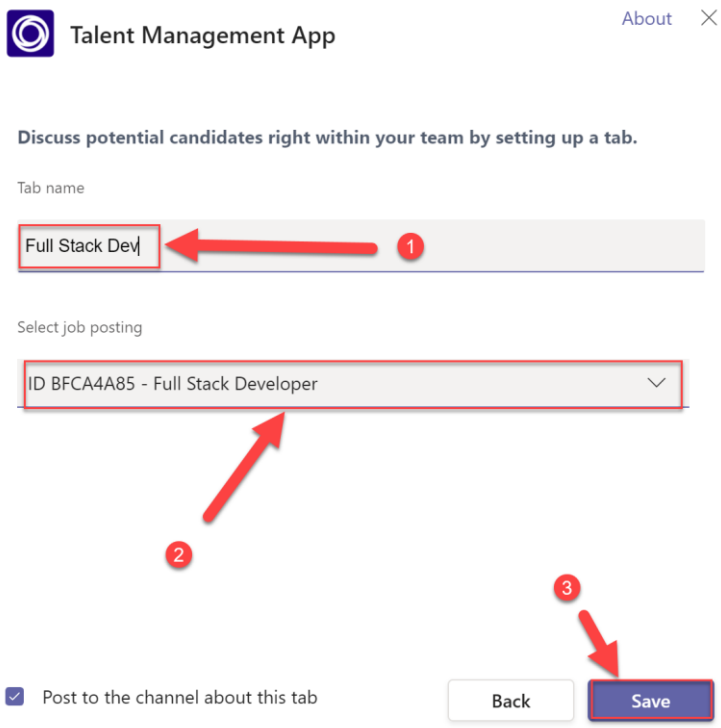
3. Select the **drop down button** and select **Add to a Team**.



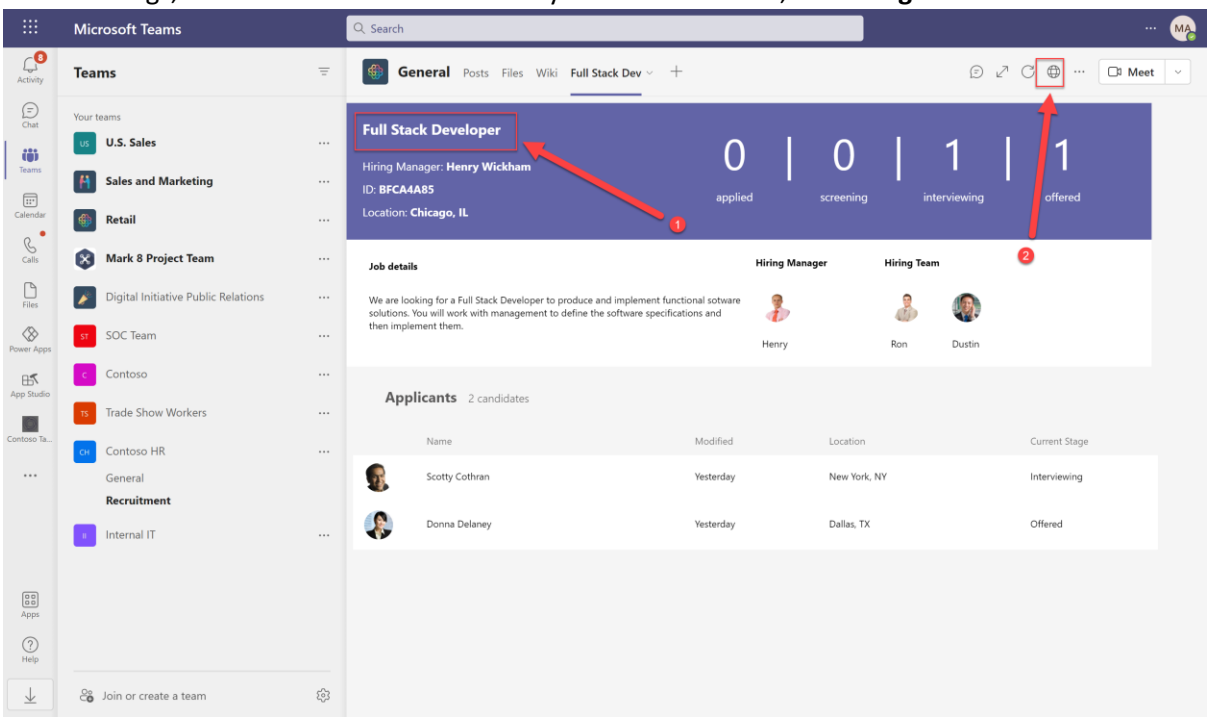
4. Select a Team and Channel, in this demo I selected **Retail -> General**, and click **Set up a tab**.



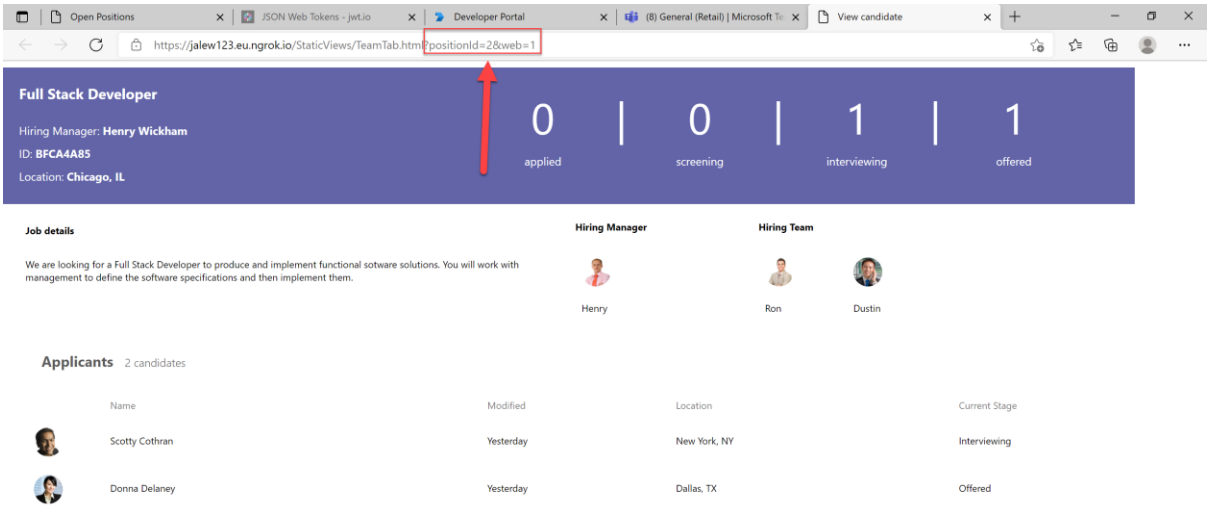
5. This will open the Tab Configuration page defined in the App Manifest. On this page you can enter a name for the Tab and select a job Position. Enter the tab name of **Full Stack Dev** and select **Full Stack Developer** in the dropdown. Click **Save.**



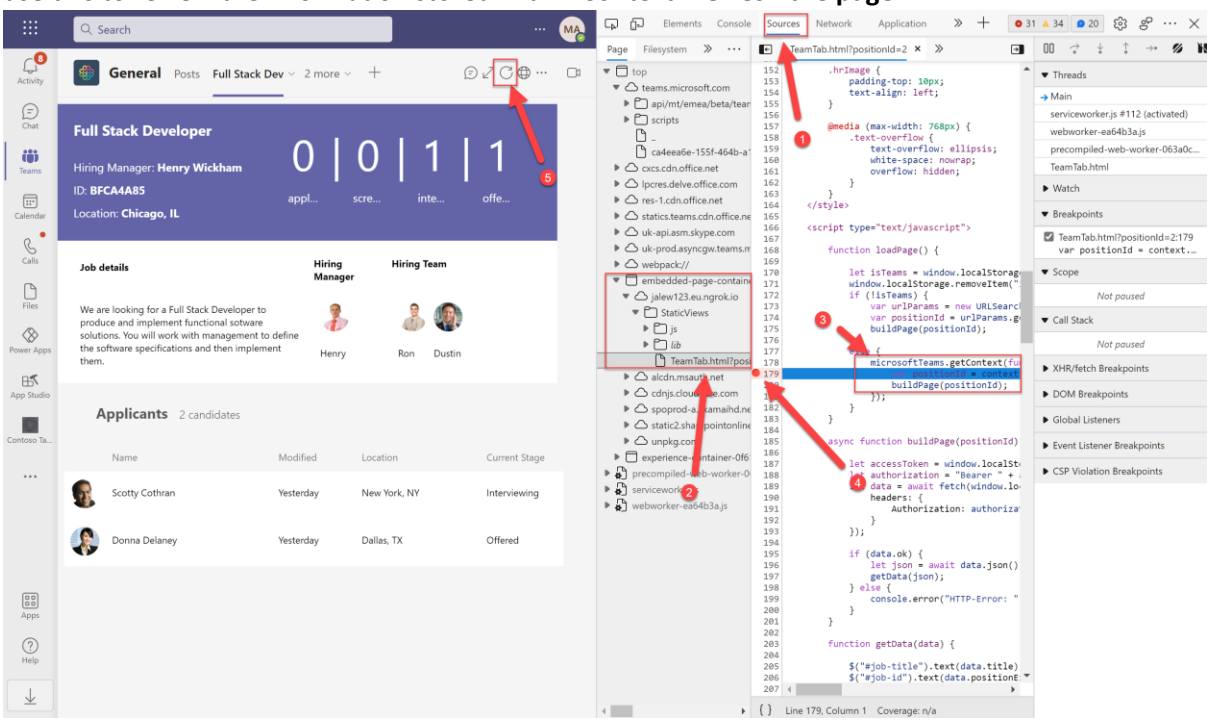
6. This should load the TeamTab.html page inside the Team Channel Tab, that contains information about the Full Stack Developer position. This is achieved by passing the Teams Tab EntityID to our web-application, via the microsoftTeams.getContext function in the Teams SDK. This page has also been configured to load outside of Teams, by using the URL Search Strings, instead of the Teams Tab EntityID. To confirm this, click the **globe** icon.



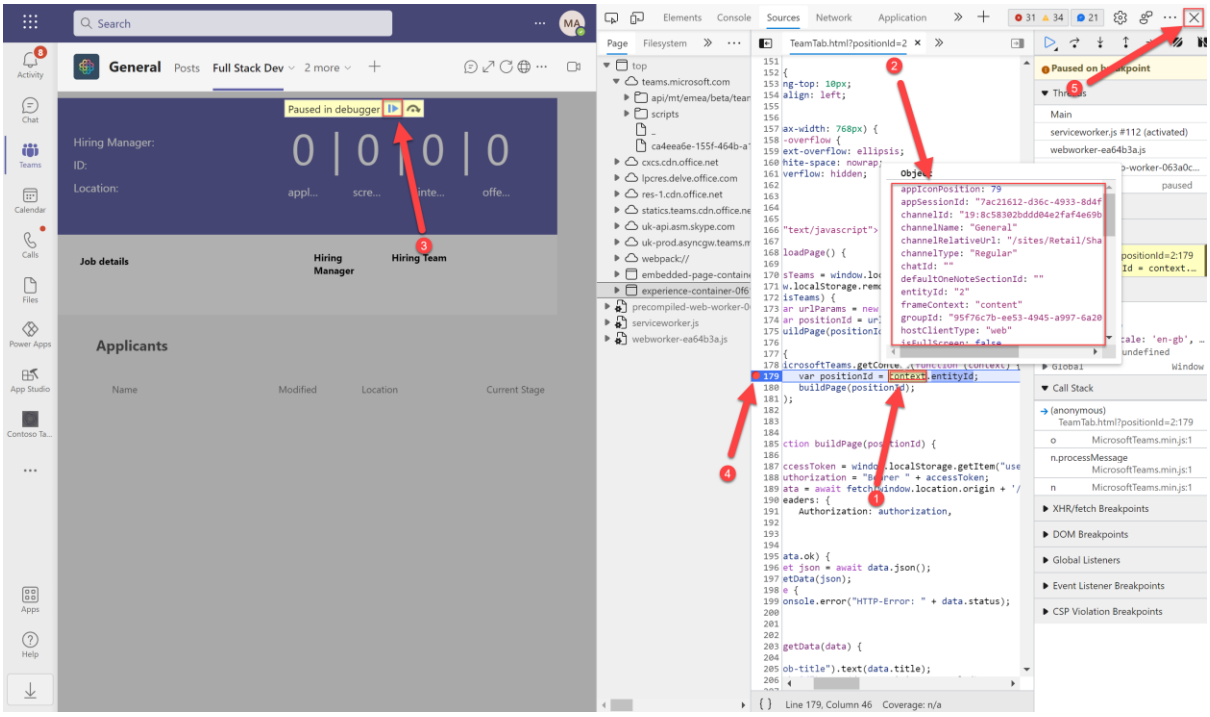
7. You will then see the page load successfully, and if you review the URL Search Strings, you will see **positionID & web** contain values.



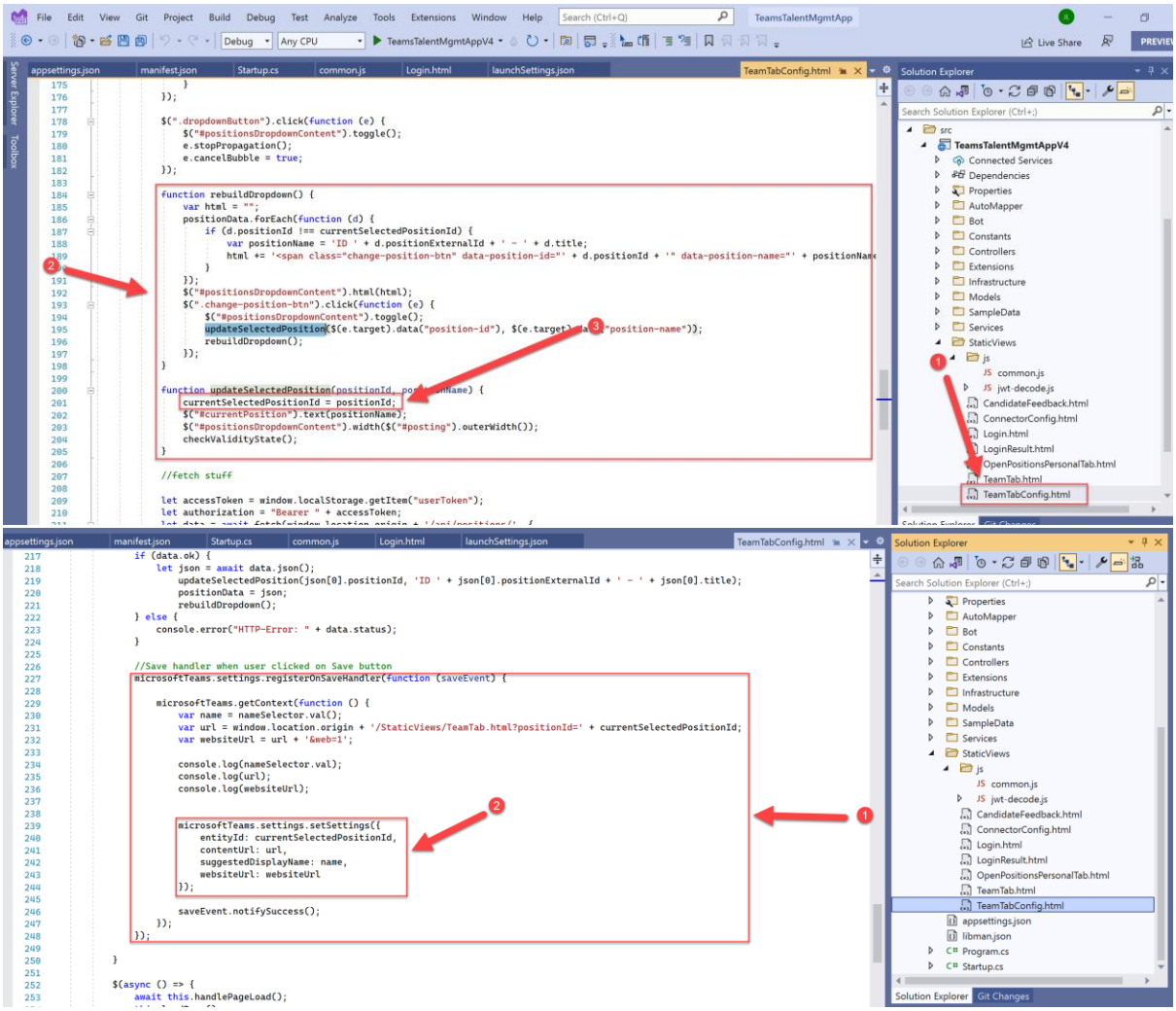
8. Let's review the code that enables this functionality. Go back into Teams, within the Channel Tab. Press **F12** to open the development tools. Select **Sources**, and expand **embedded-page, NGROK, StaticViews and select TeamTab.html**. Scroll down to line 177 in the code and review the **microsoftTeams.getContext** function, this is the capability within the Teams SDK that will pass the EntityID, and a lot of other context information through to the Talent Management Web App. We use the EntityID, stored within the Context variable, to build the page by calling **buildPage(positionID)** on line 180. **Add a breakpoint to line 179**, as we will use this to review the information stored within Context. **Refresh the page**.



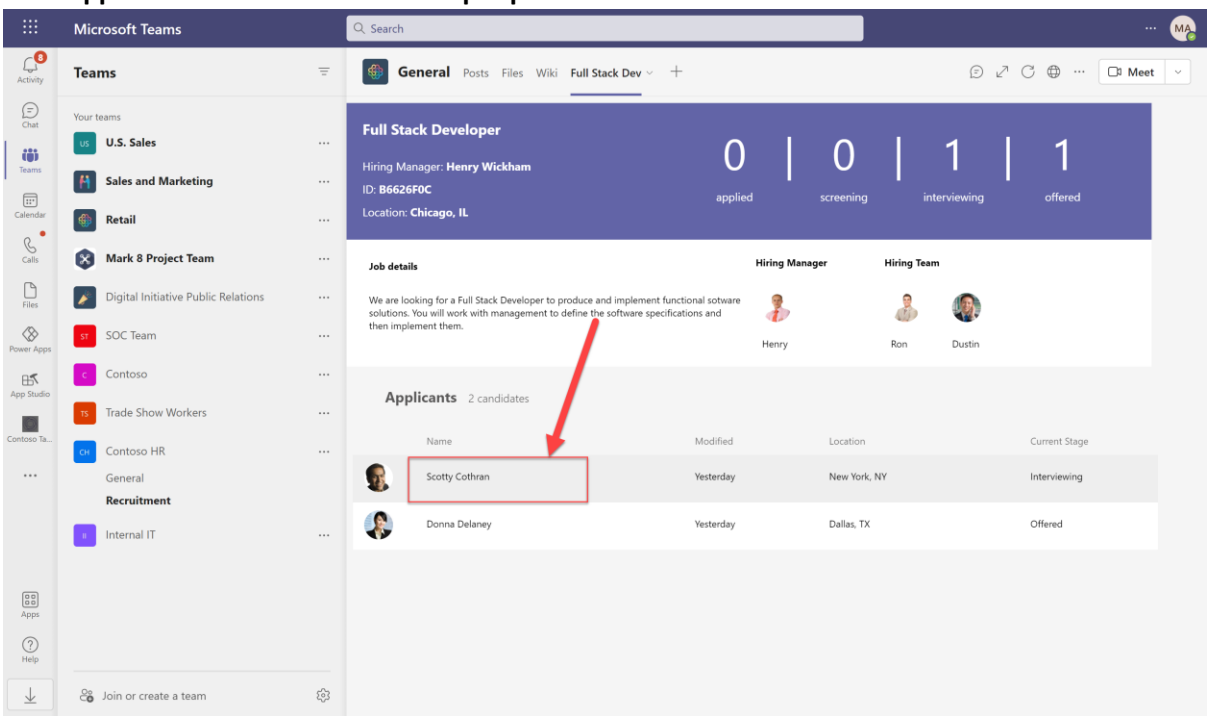
9. The breakpoint will be triggered. **Mouse over the word context** on line 179, this will display the values stored within the context variable, which are set when **microsoftTeams.getContext** is called. You can see here that alot of useful information is available to you, when you are building you web-app. In our case, we simply use **entityID** to decide which position we should show on the page, but this context can be used to discover channelName, themeSettings, and a load of other settings! Review this and consider what you could use in your application to provide a brilliant experience when your app is delivered inside Teams. Click the **continue** button to resume the debugger, remove the **breakpoint** by clicking red circle icon to the left of line 179, and **close the developer tools**. Let's now go back into visual studio and review the steps that are taken to pass the entityID from the TeamTabConfig.html page, into the TeamTab.html page.



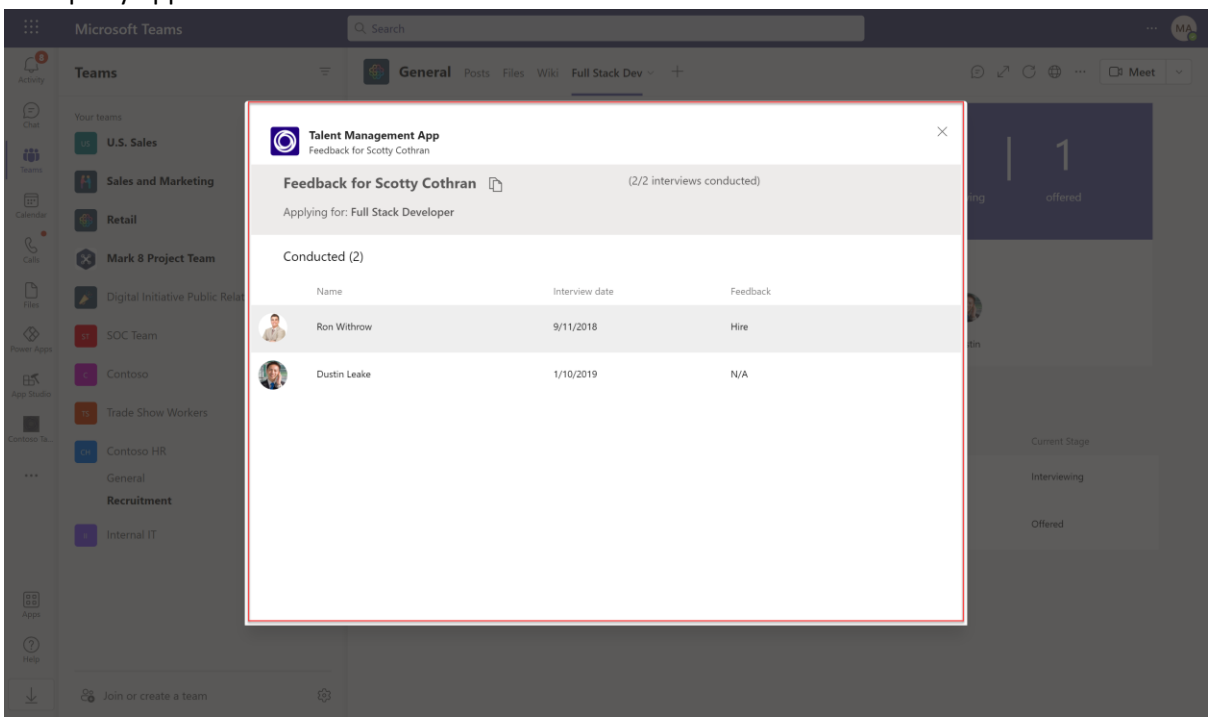
10. Open Visual Studio, and open the **TeamTabConfig.html** page, scroll down to line 184 and review the code contained in the **rebuildDropdown and updateSelectedPosition** functions. These work together to build the dropdown with available Positions, and when a position is selected, they set the **currentSelectedPositionId** variable with the ID of the selected position. Scrolling further down to line 227, review the **microsoftTeams.settings.registerOnSaveHandler** function – this uses the **currentSelectedPositionId** variable, to set the **entityId** that will be passed into the TeamTab.html page when it is loaded inside Teams.



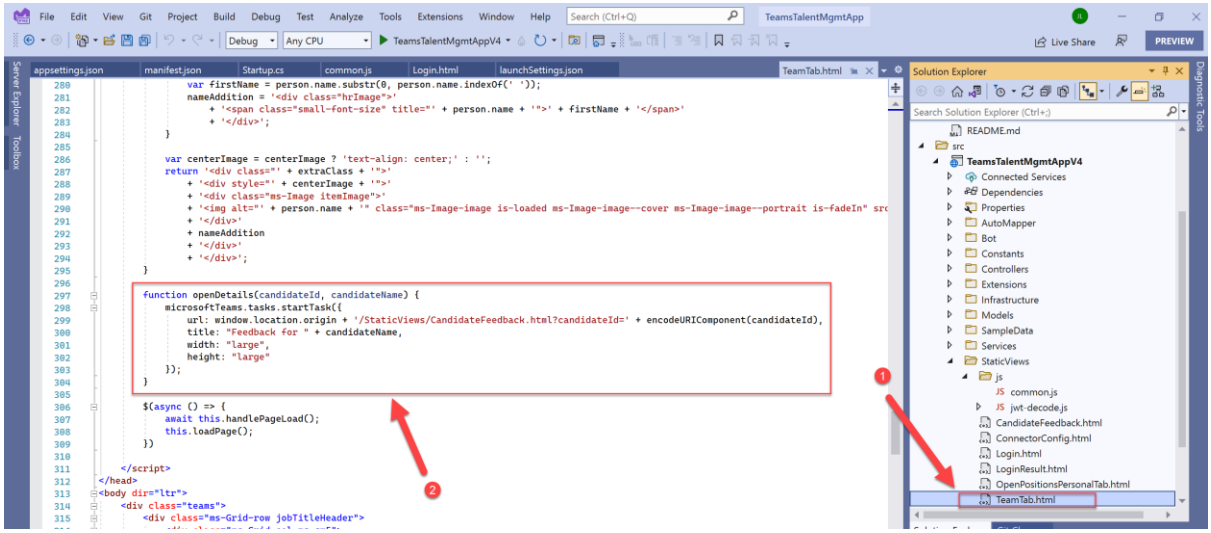
11. Go back into Teams and open the Team Channel Tab and select one of the **candidates that have applied for the Full Stack Developer position**.



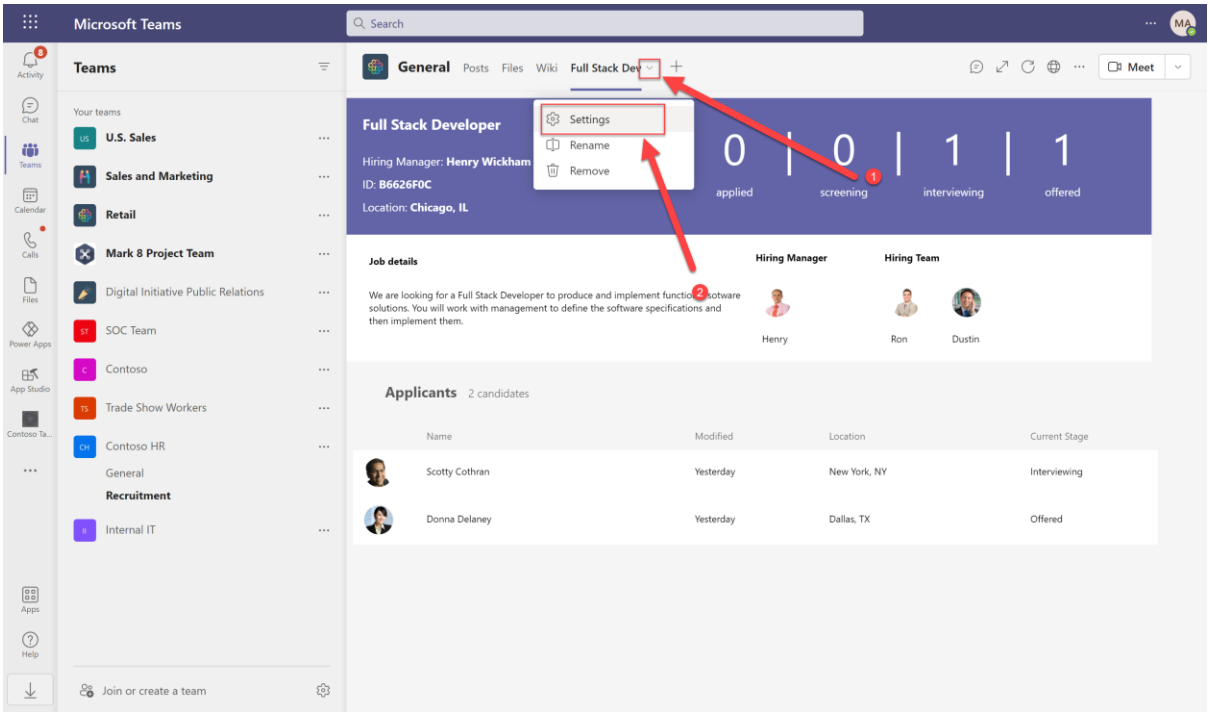
12. This will open a **Task Module**, which is a pop-up within Teams that iFrames a web-application (similar to how Tabs work!). These are great, as they allow you to stay in the context of what you are doing within Teams, while reviewing information that exists within a third party application. Let's review the code that initiated the task module…



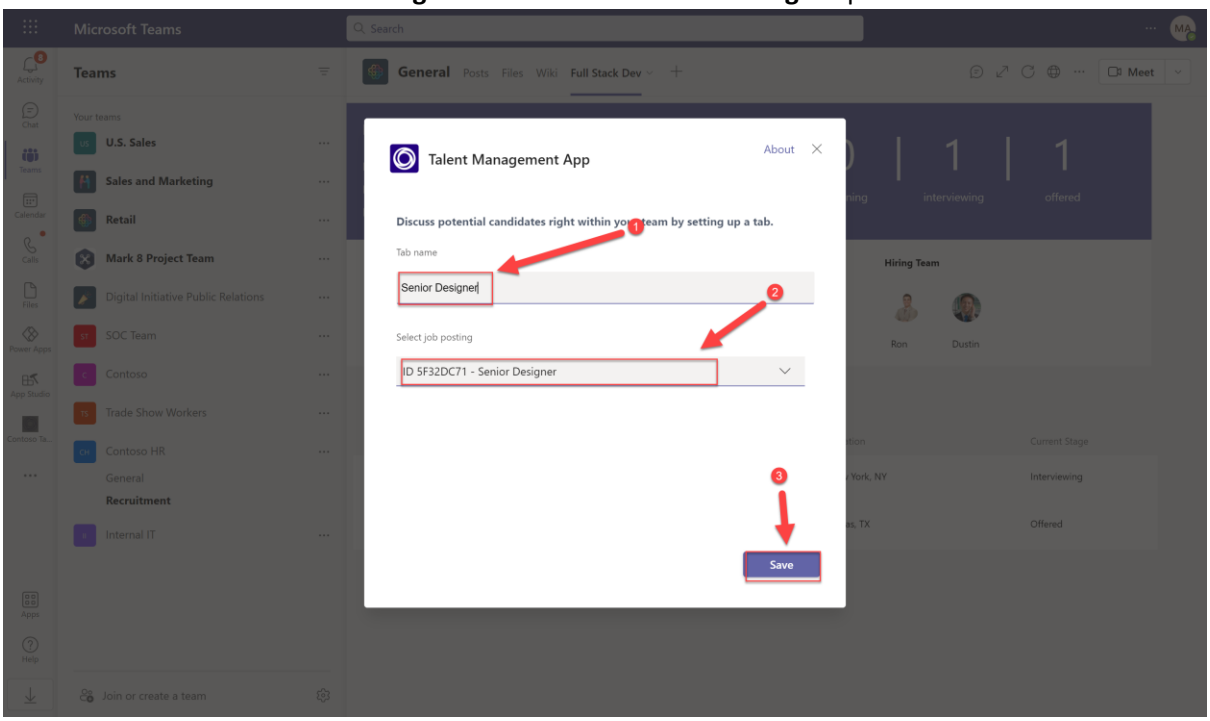
13. Head back into Visual Studio and open the **TeamTab.html** file. Scroll down to line 297 and review the **openDetails** function. This uses the **microsoftTeams.tasks.startTask** function, passing in the url, title, width and height of the task module.



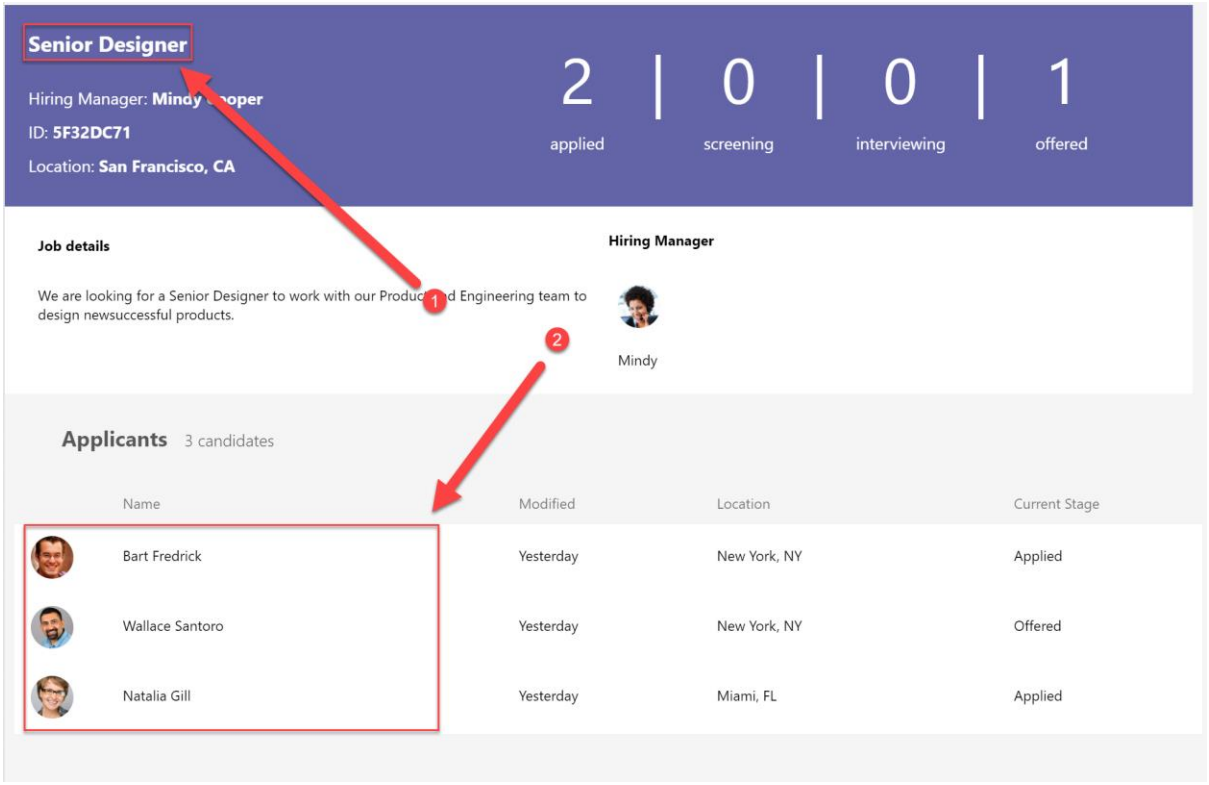
14. Head back into Teams and let's review how you can edit Team Channel Tabs, which will re-invoke the TeamTabConfig.html page. Select the **drop-down button** and click **Settings**.



15. Enter the Tab name of **Senior Designer** and select the **Senior Designer** position. Click **Save**



16. You will see now that the TeamTabConfig.html page has changed the entityId to match the positionId of the Senior Designer position. If you want to; review the context in the developer tools to confirm this.

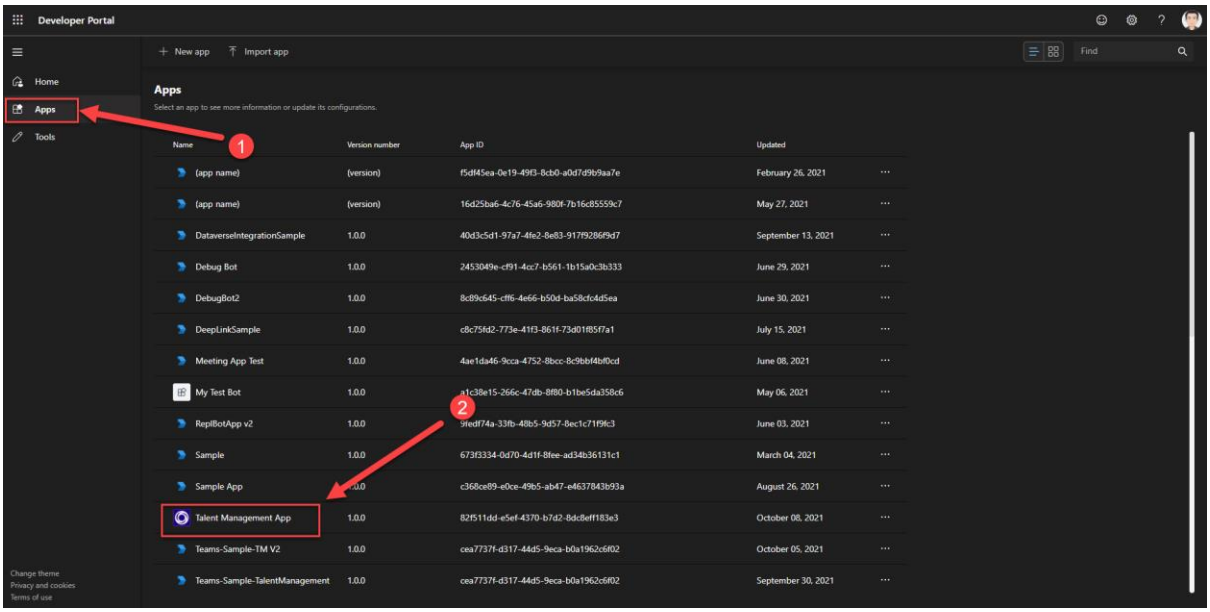


Now that the Channel Tab is working we will now move onto configuring the ChatBot. As we are now moving from HTML and JavaScript to .Net/C#, you will see the following screenshots in dark-mode - a favourite amongst back-end developers…

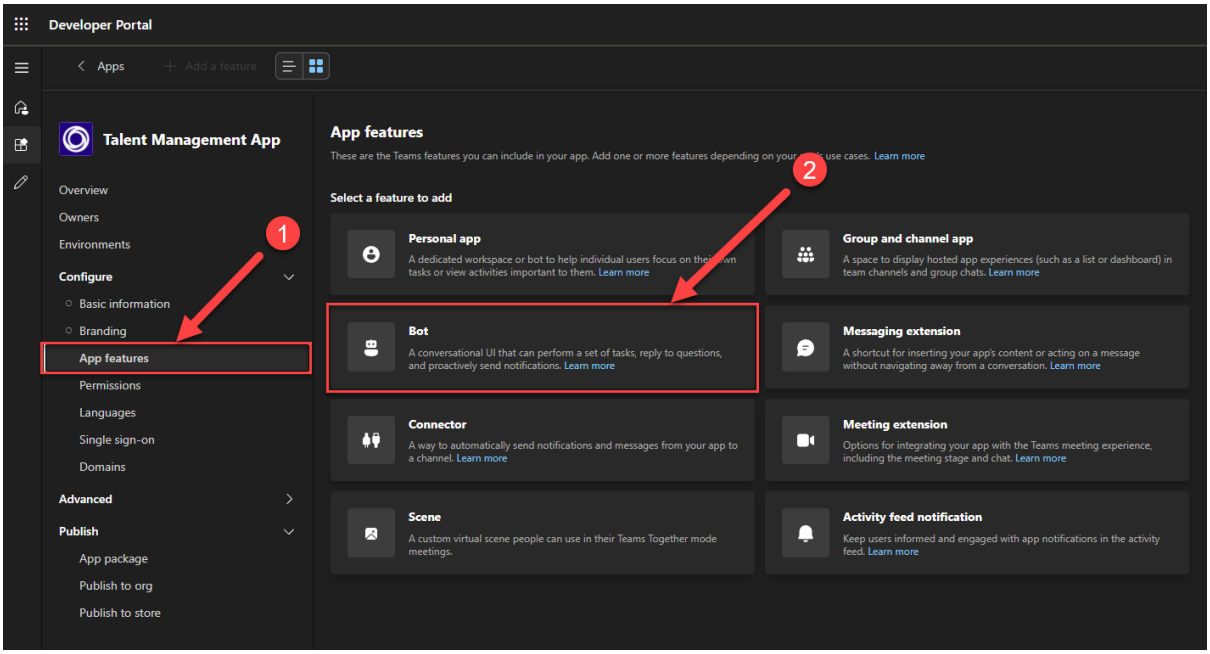
4) Implement a Personal ChatBot

In this step we will modify the application manifest to add our bot to the app.

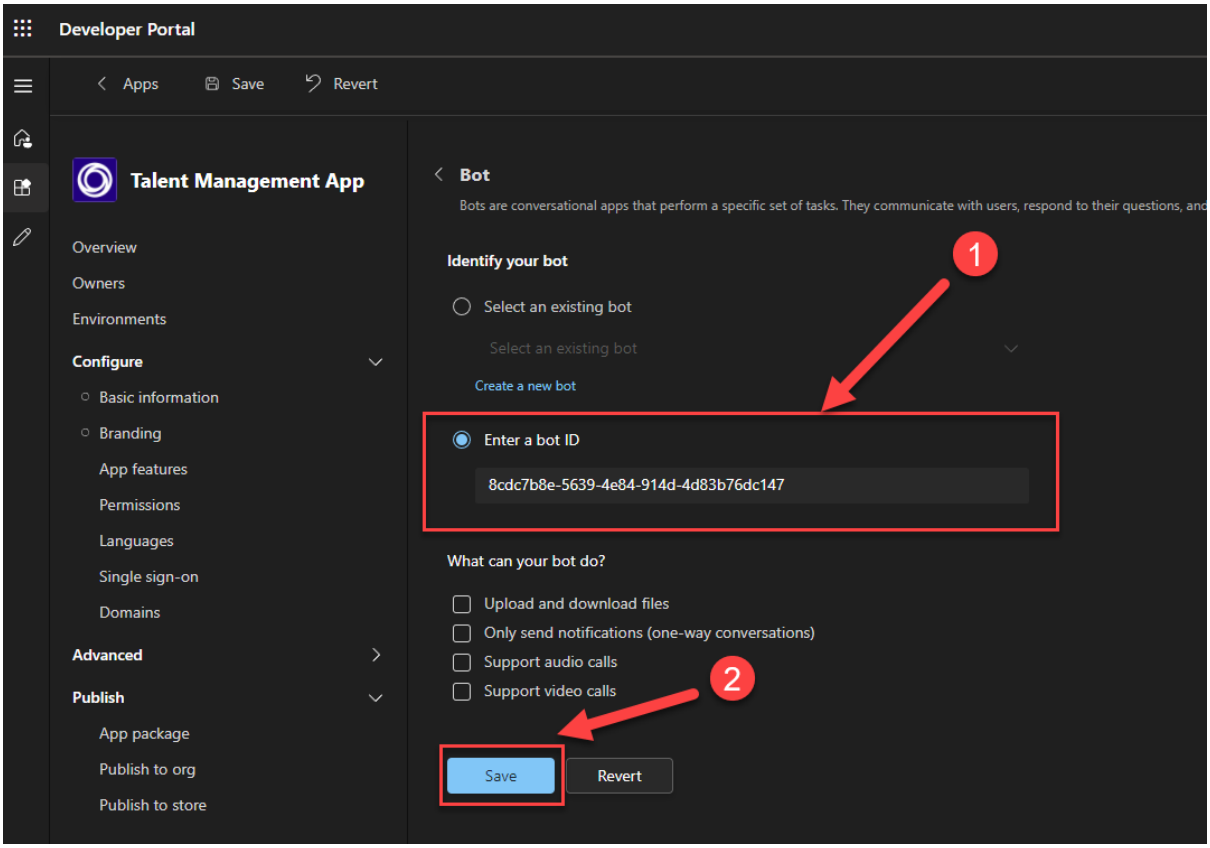
1. Open the Developer Portal by navigating to *https://dev.teams.microsoft.com*. Once there we will select **Apps** from the navigation menu and find the application we had previously created.



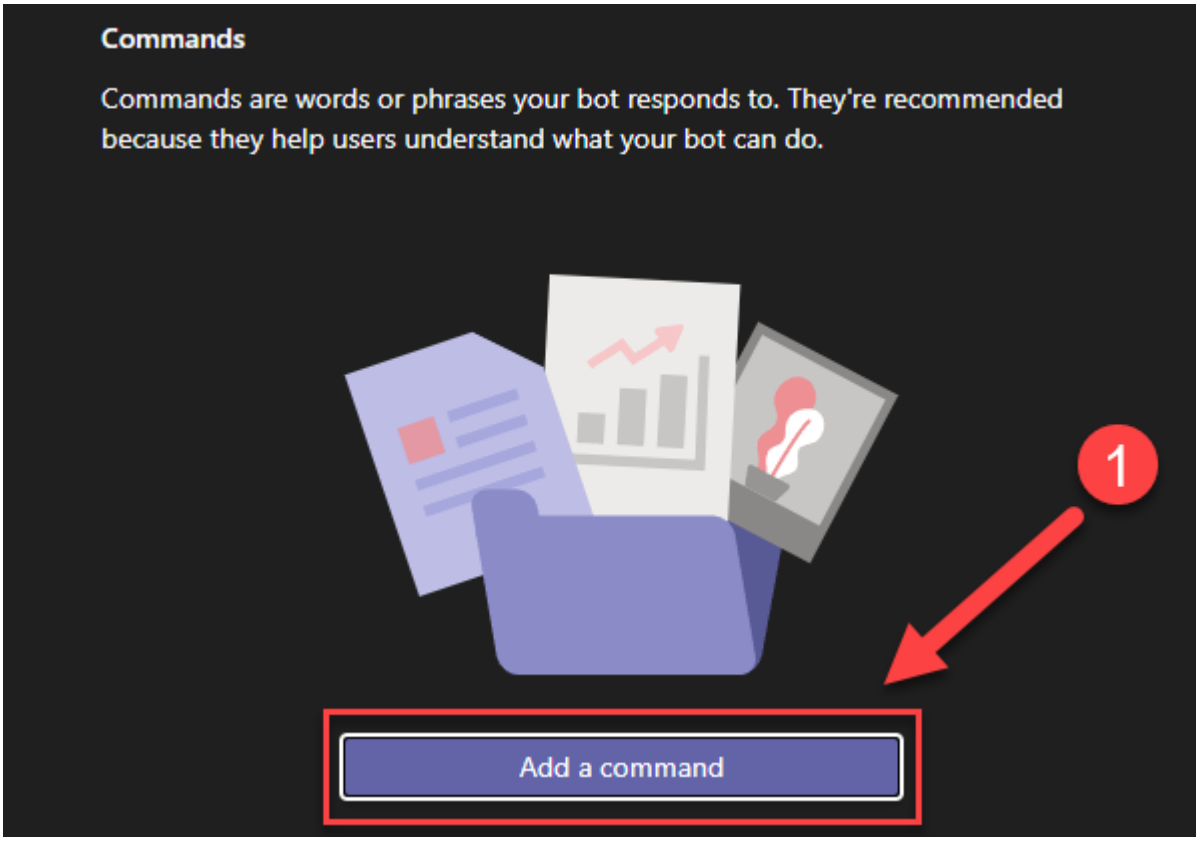
2. Once we are in the context of our application, we will add the bot. Select **App Features** from the left menu and choose **Bot** as the feature we would like to add.



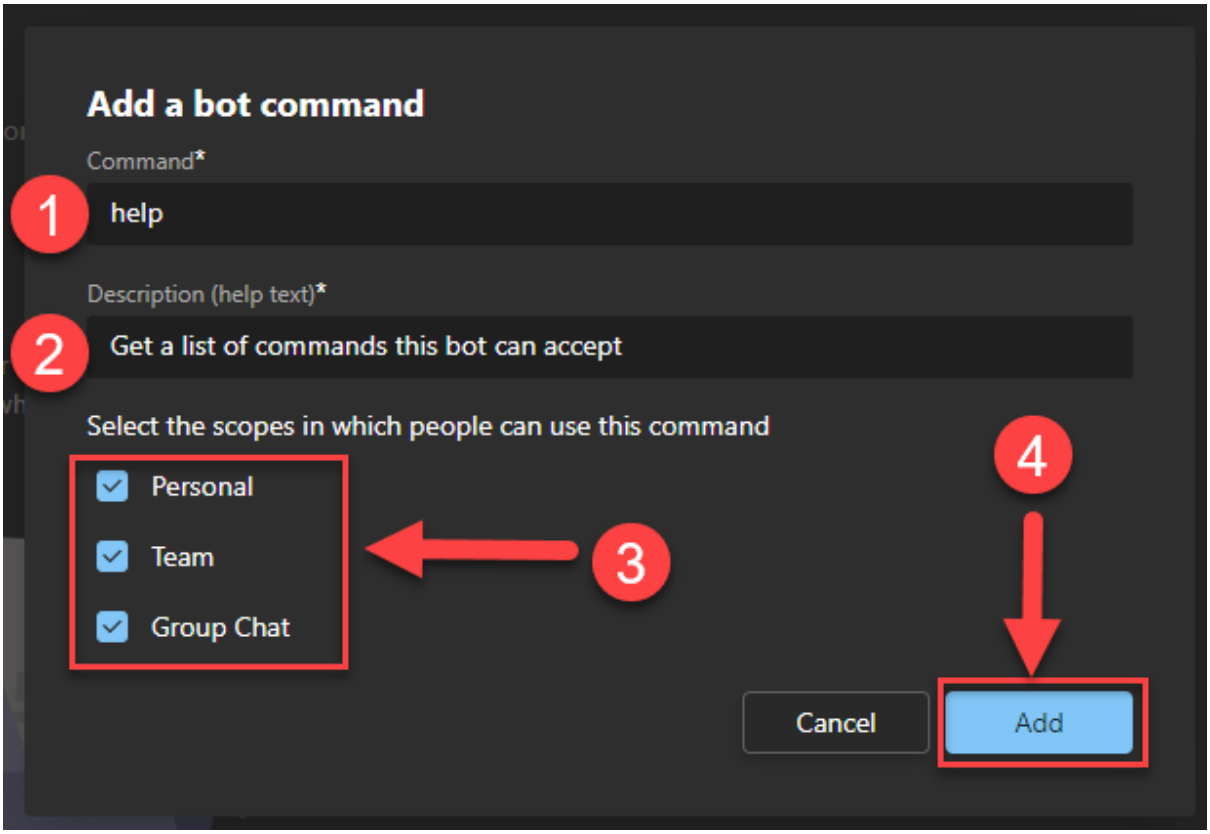
3. Here we will modify the configuration to use the bot service instance we created earlier. There is a bug in the Developer Portal here and adding an existing bot ID doesn't enable the Save button. In order to get around this issue, either **select an existing** bot first (if you have one, or **create a new bot**) Once the dropdown has a bot and the **Save** button has been enabled, click **Enter a bot ID** and copy in the **Client Id** from the App Registration created earlier… the Client Id and bot ID are the same! Finally, click **Save** to save the manifest.



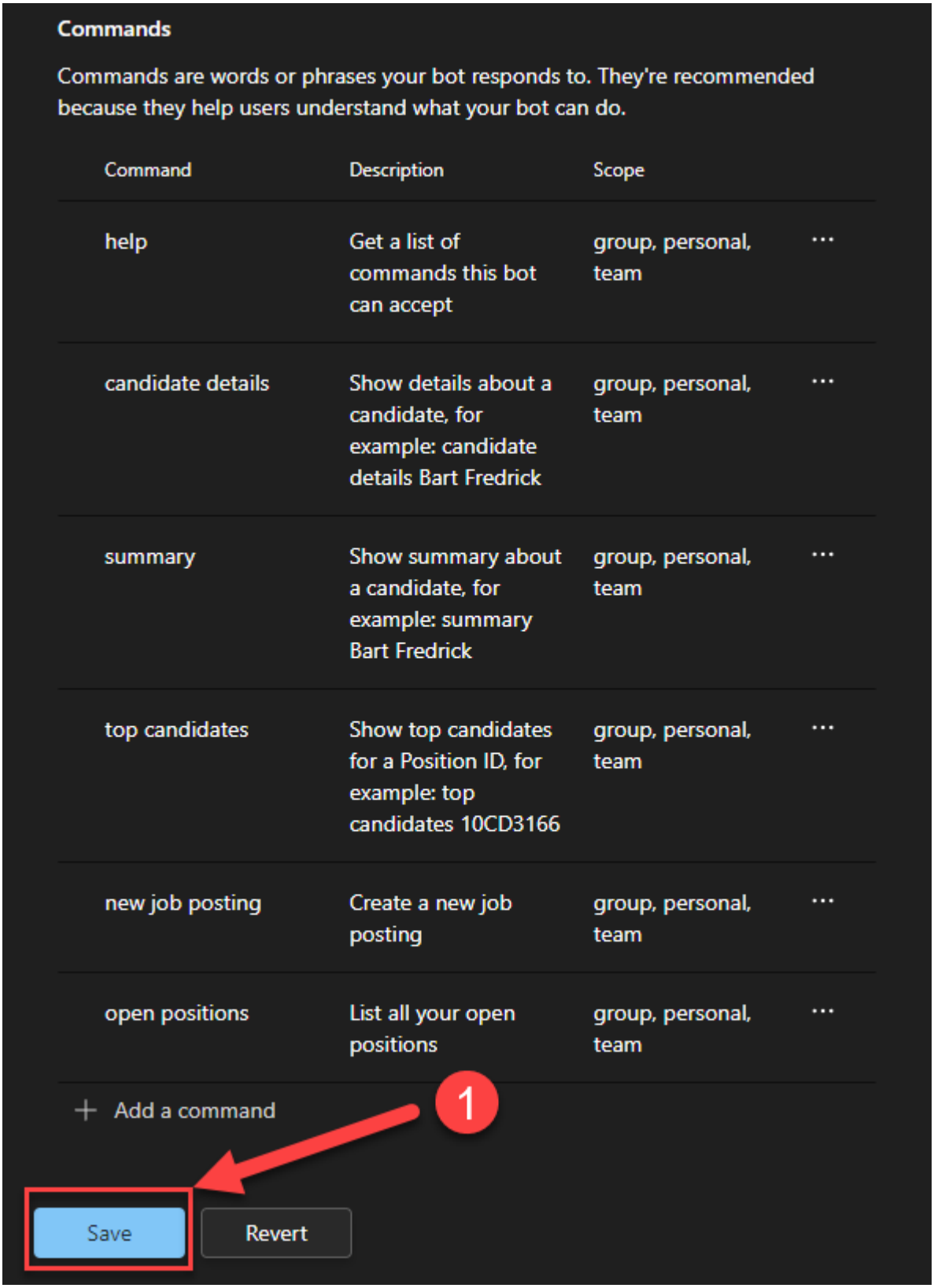
4. Now we want to add commands to the bot, a new Commands dialog should now be visible. Click the **Add a command** button.



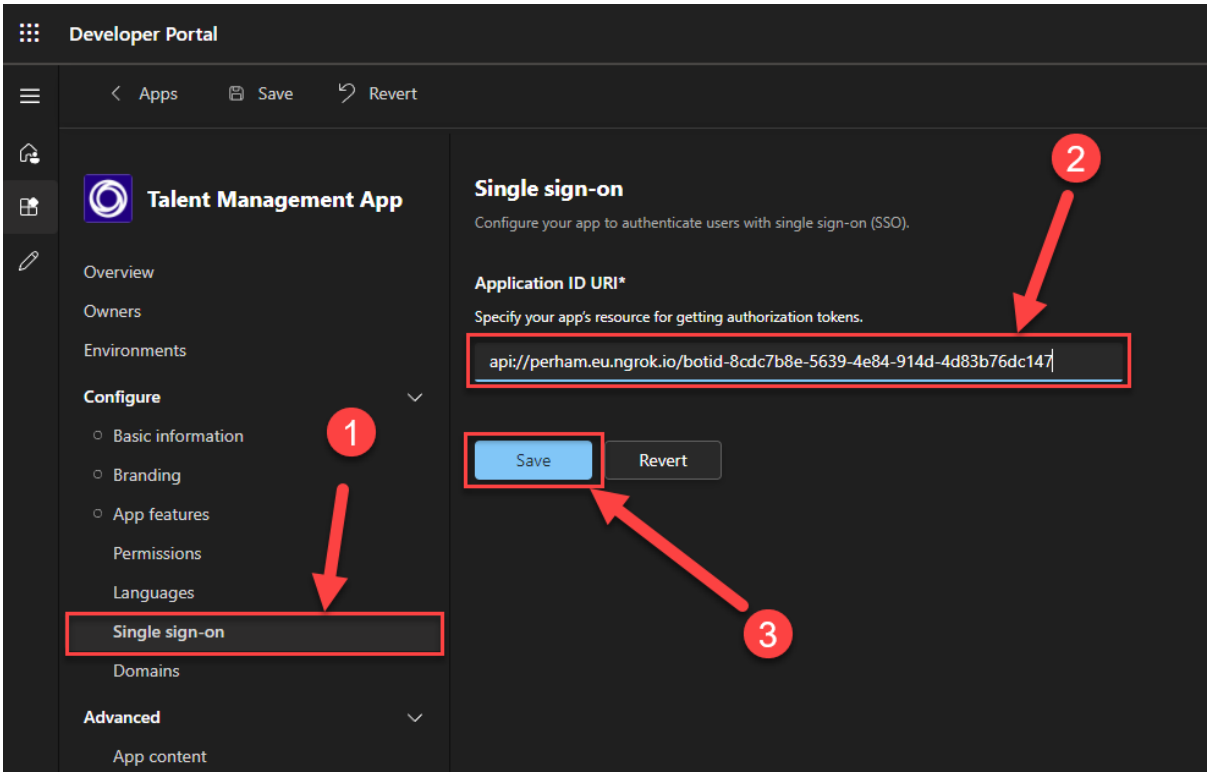
5. Now we will configure the command. Enter the details below and click **Add.**



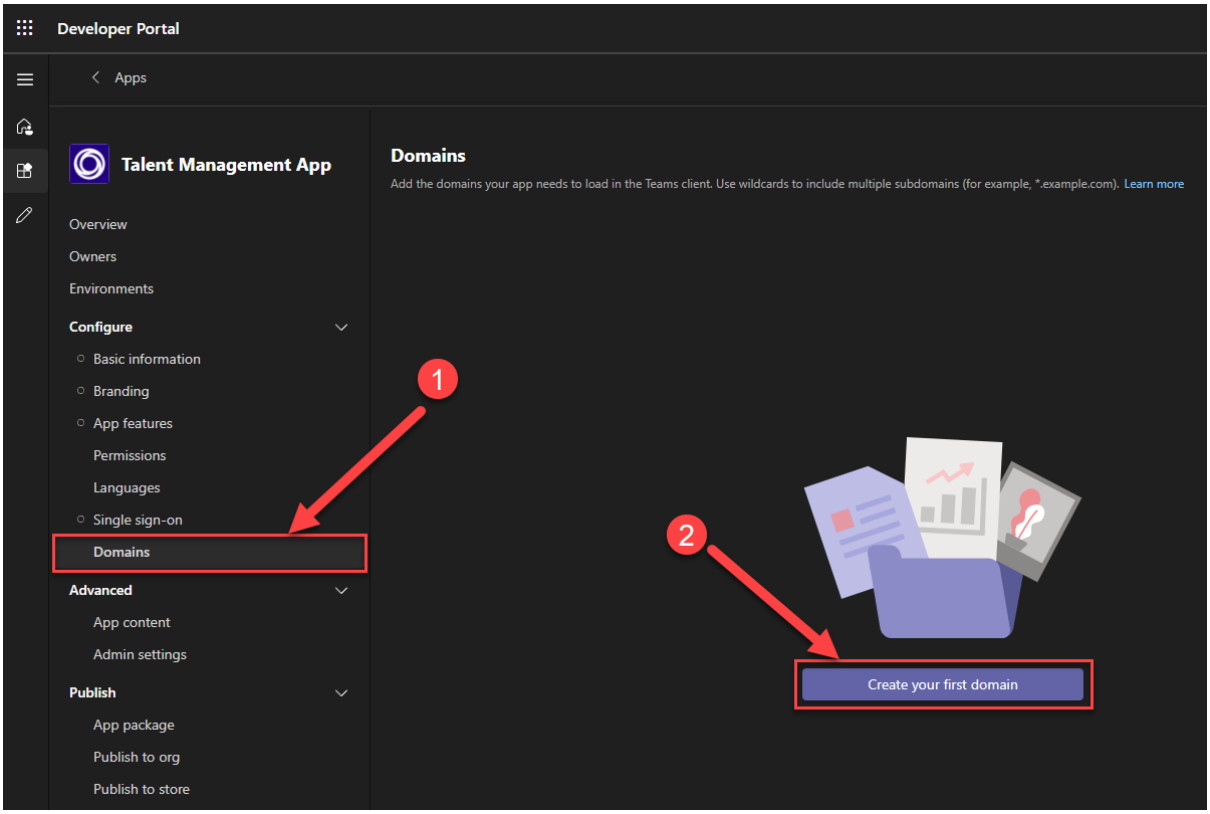
6. Repeat this process for all the commands the bot is able to perform as shown below.
NOTE: Ensure that you click **Save** after each command to ensure that you don't lose any configuration if the edit breaks!



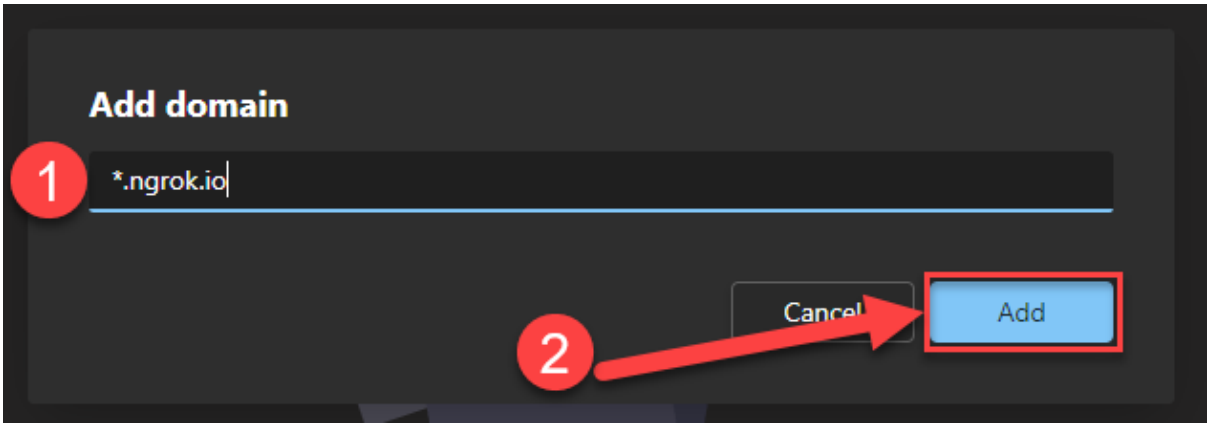
7. Next will configure Single sign-on so that your bot can work in the context of the current user. Select **Single sign-on** from the left menu. Enter your Application ID URI as defined in your App Registration earlier (this should be in the following format: *api://{domain}/botid-{client id}*). Finally, click **Save**.



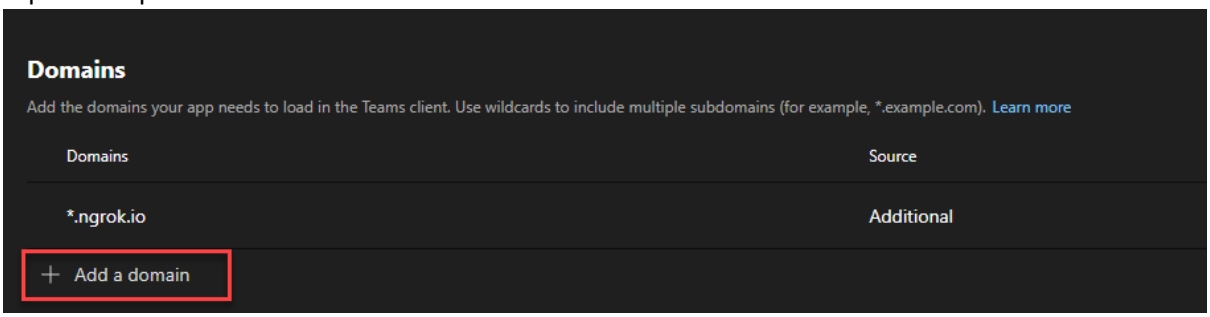
8. Now that the bot has been configured and SSO has been enabled, we need to allow the bot to talk to both the Azure bot service and also your backend API service. Select **Domains** from the left menu and then click **Create your first domain**.



9. There are two domains we will need to add, the first will be your API service. As we are using ngrok to tunnel, add the wildcard domain **.ngrok.io* and the click **Save**.

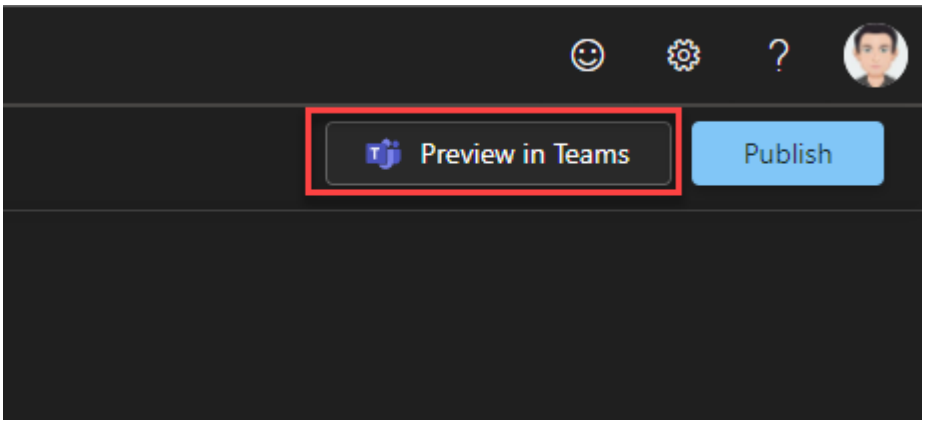


10. You will now see that your domain appears in the domains list. Click **Add a domain** and repeat this process.

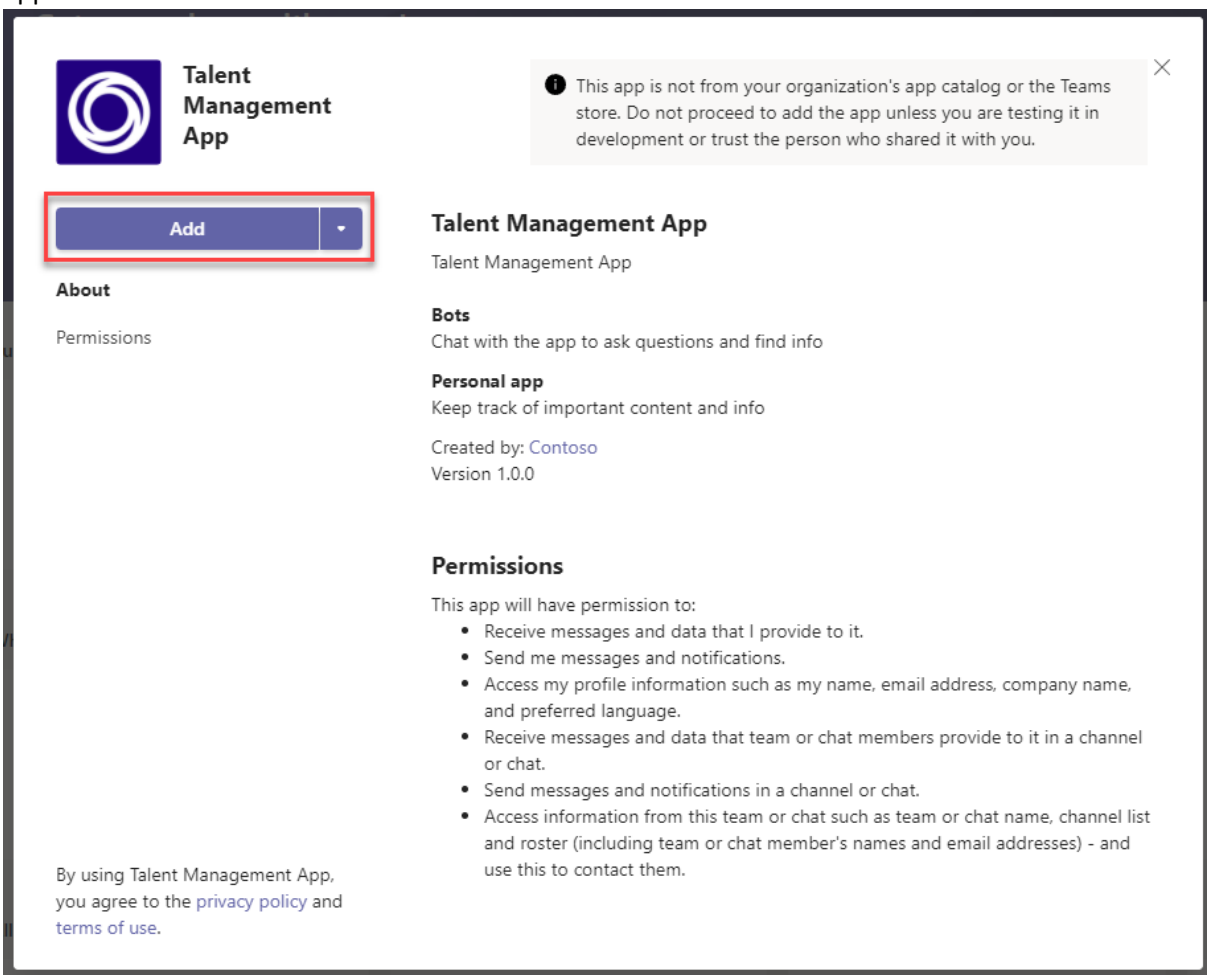


11. We will now add the domain that bot framework uses to community with our bot for SSO. Enter *token.botframework.com* and click **Add**.

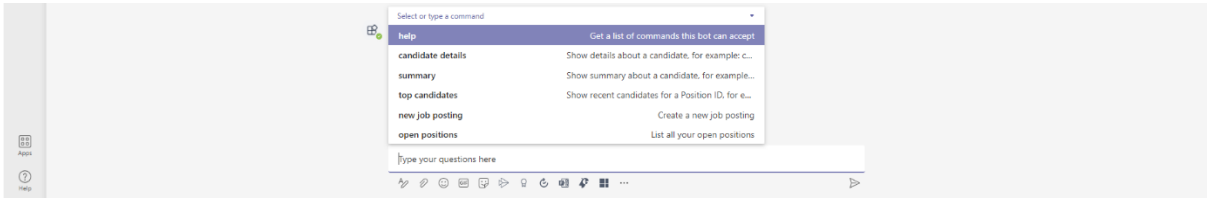
12. Now that the bot has been configured completely, let's test it out! Click **Preview in Teams** to load the manifest into a Teams instance.



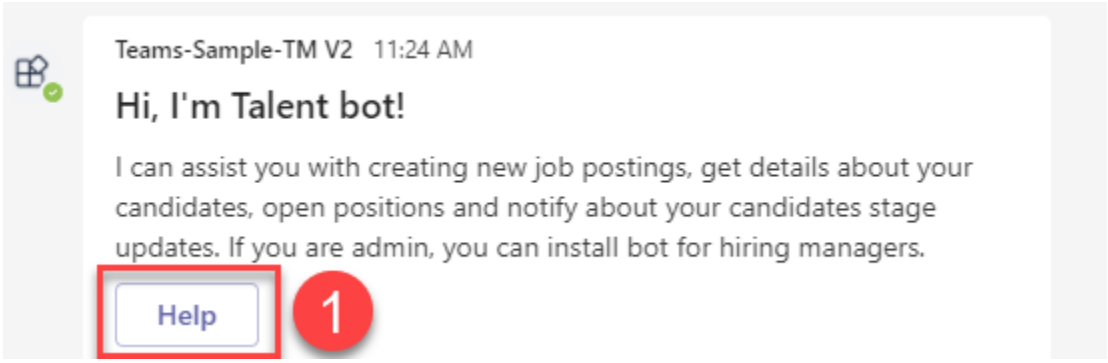
13. Once in Teams, you should be displayed with the following dialog asking you to add your application. This will actually overwrite the manifest that already exists and you will see that the permissions list has grown now that you've added your bot. Click **Add** to update your application and be taken to the channel.



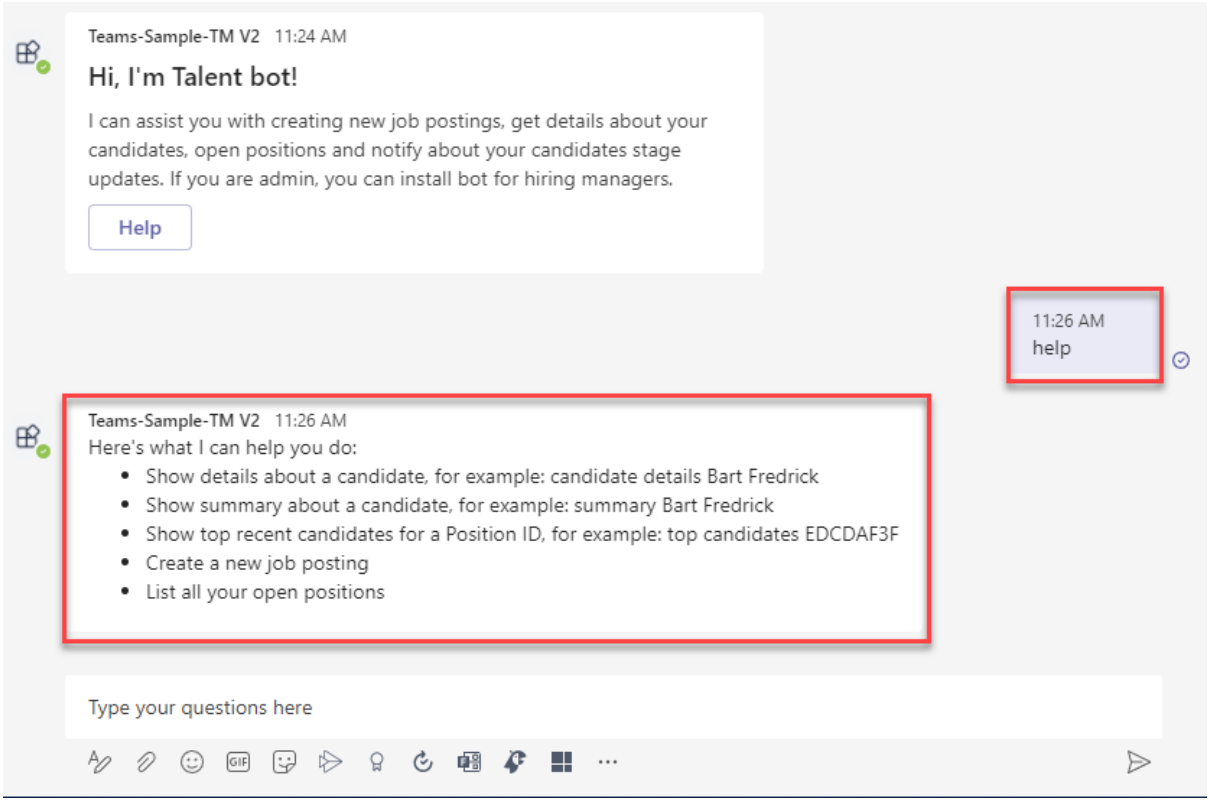
14. When you highlight the Compose box, you will now see that you are presented with a list of command available through the bot. You can either click a command or type the name. Note: that the bot will have also added a welcome message but depending on the size of your window it might be hidden by the bot command list. Once the list is dismissed you will see the welcome message that was sent by your bot.



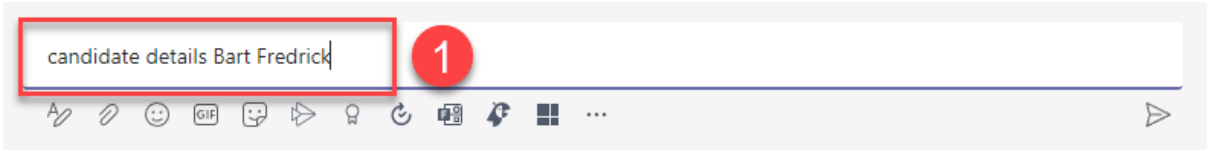
15. This message includes an action button which has been configured send the help command to the bot. Clicking Help will send the text "help" to the bot just as if you'd typed it yourself in the compose box.



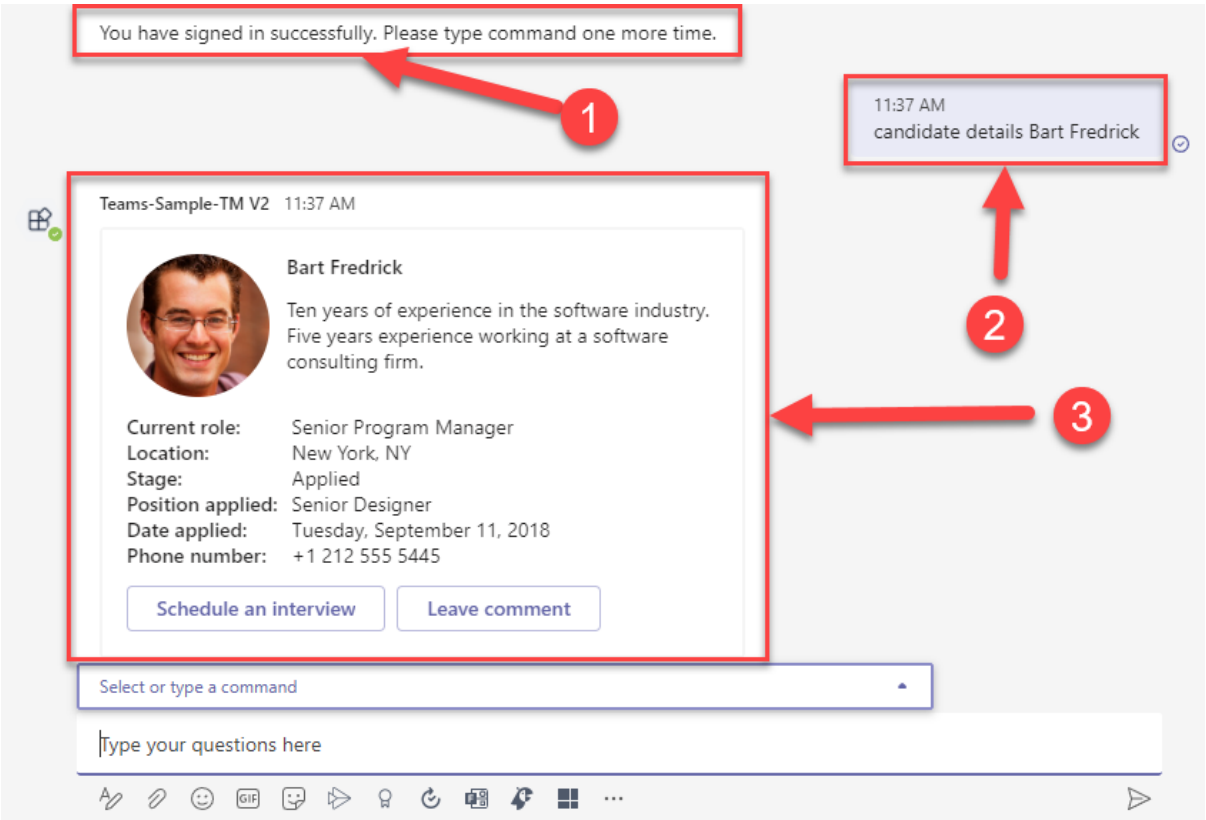
16. Once the command has been received by the bot it will respond with a friendly help message listing some of the common commands.



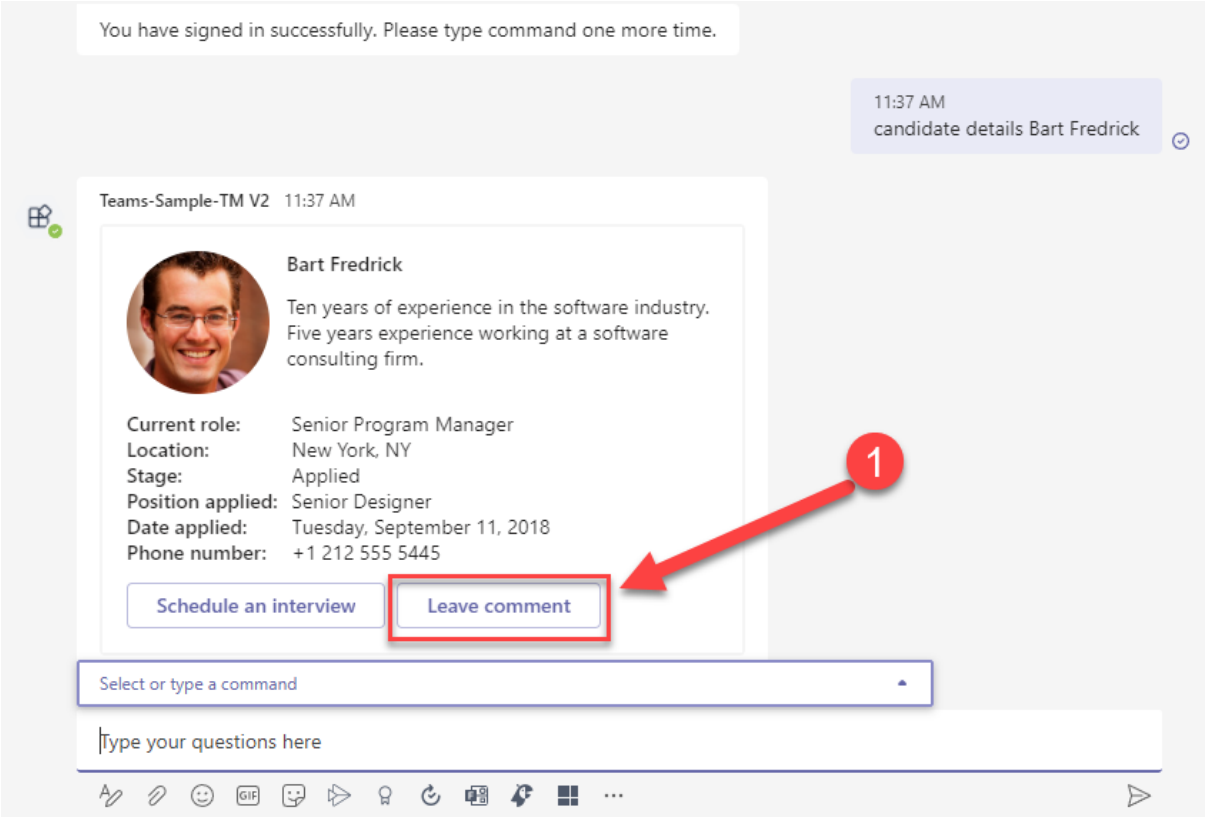
17. One such command is the candidate details command. Let's try this now… copy or type *candidate details Bart Fredrick* to see details about the candidate.



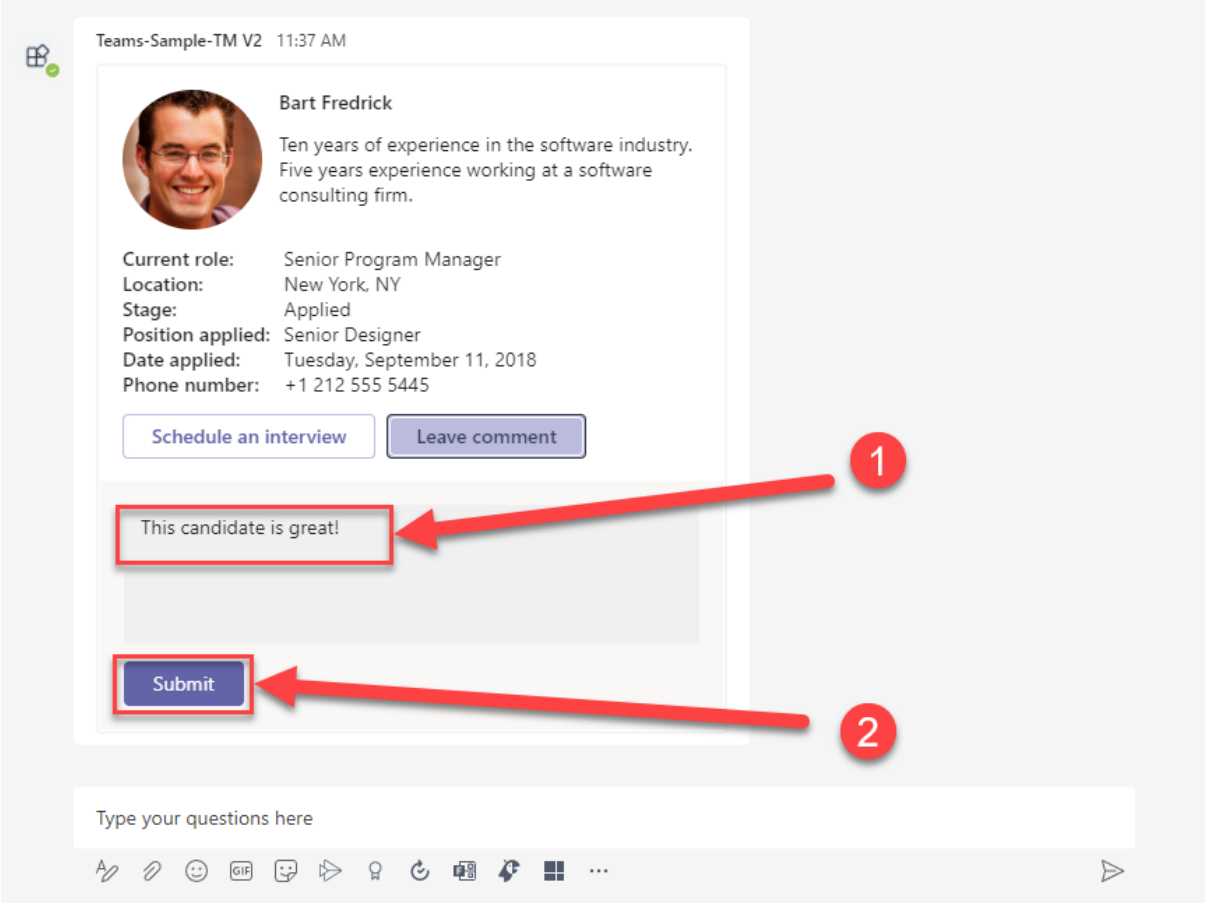
18. You will notice that before any candidate details are shown you are presented with a message saying that you have successfully been signed in. This is the single sign-on process working in the background. The message will also say that now you are signed in you must resend your command. Doing this will display an Adaptive Card displaying details about Bart Fredrick.



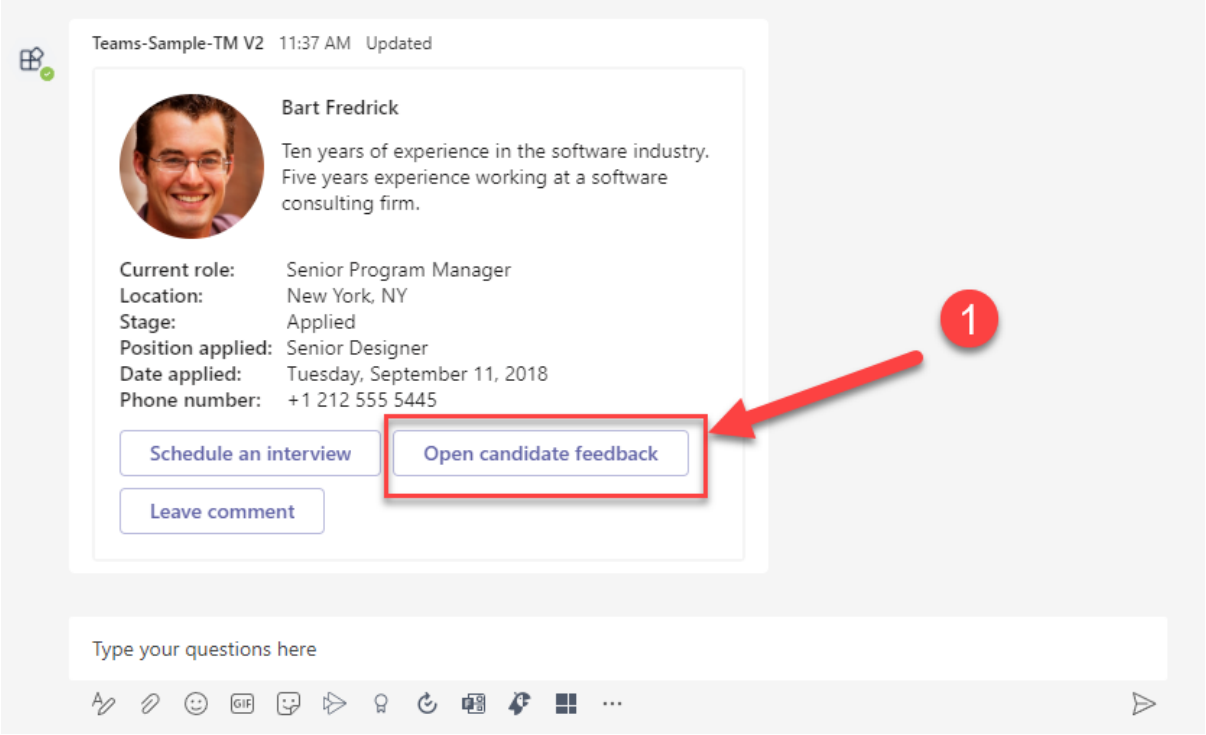
19. You will also notice that there are some buttons on the card. Clicking **Leave comment** will display a textbox giving you the ability to leave a comment about the candidate.



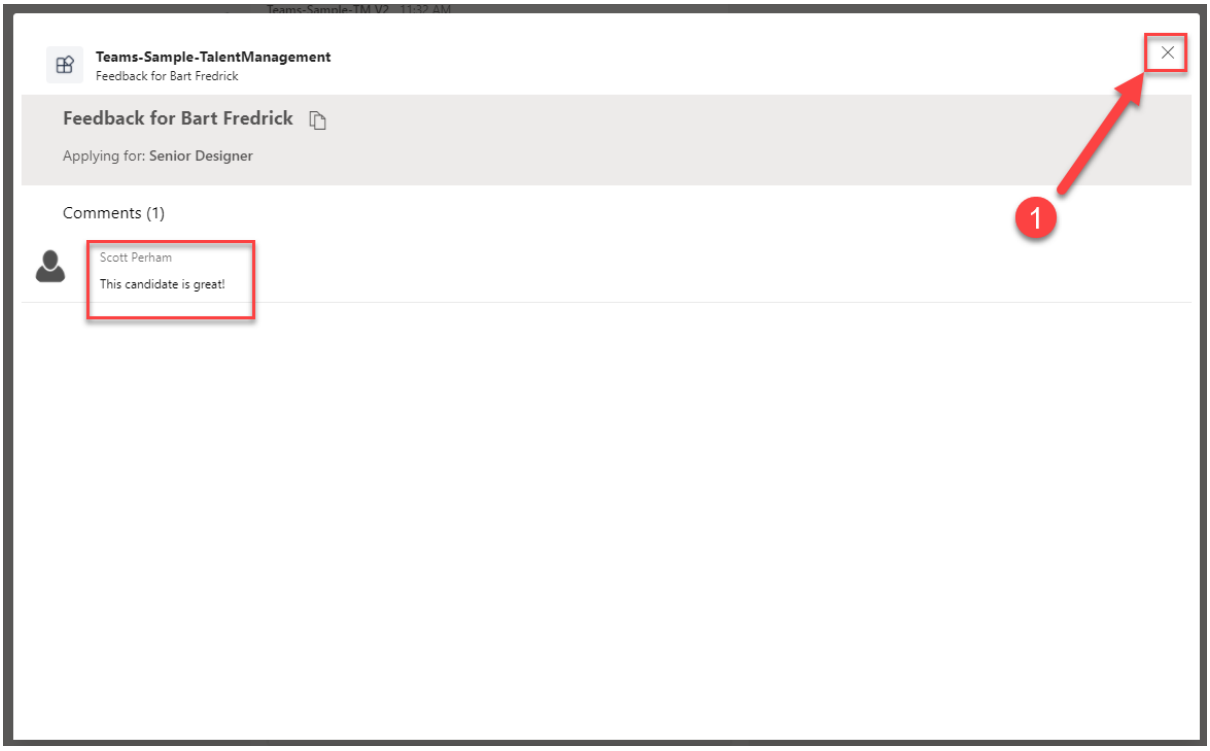
20. Enter a comment and click **Submit** to send the comment to the bot.



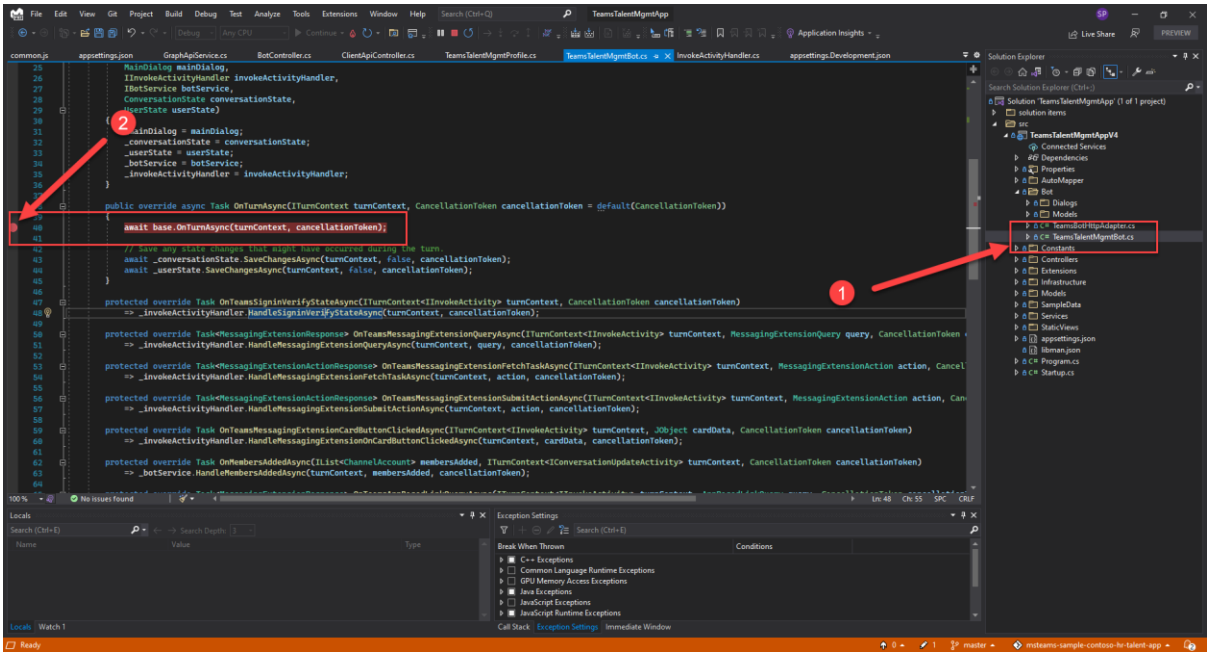
21. This will resend the candidate adaptive card to Teams and because we now have some feedback, and additional (conditional) button will be shown. Click **Open candidate feedback** to see a summary of all the feedback left for the candidate.



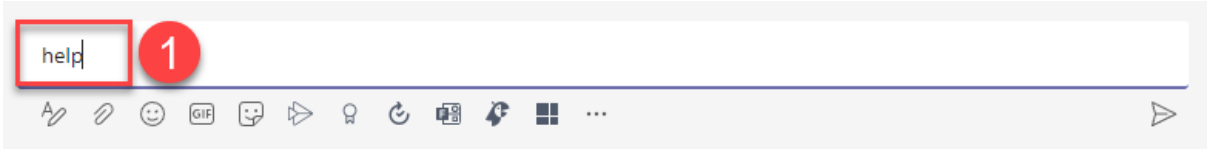
22. You will see your feedback in the list! Once you are done click the X in the corner to close the Task Module.



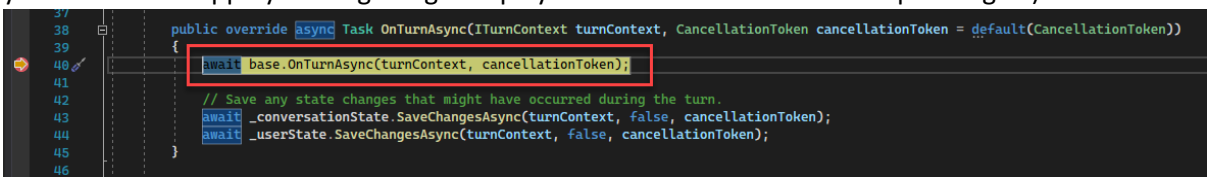
23. We will now have a look behind the scenes and see what happens when commands are sent from Teams to your bot. Open Visual Studio and navigate to the TeamsTalentMgmtBot.cs file. Then put a breakpoint on line 40. (Breakpoint can either be set by clicking the leftmost gutter on the appropriate line, or by clicking somewhere in the line and hitting F9). The line will turn red once the breakpoint has been set.



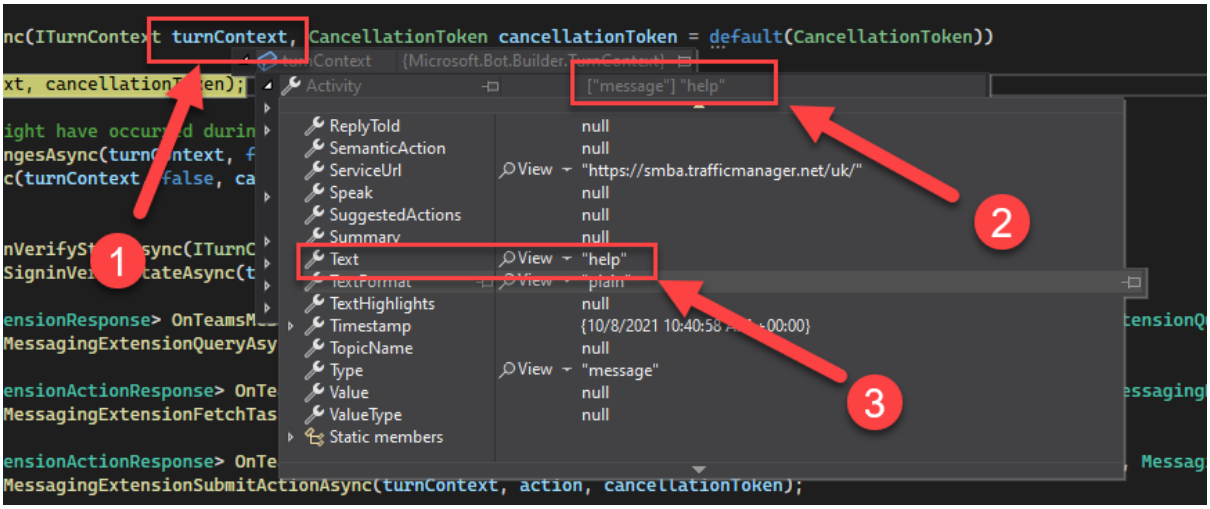
24. Now with your application running, go back to Teams, in your bot chat channel, type “help” and press *Enter* or the send button.



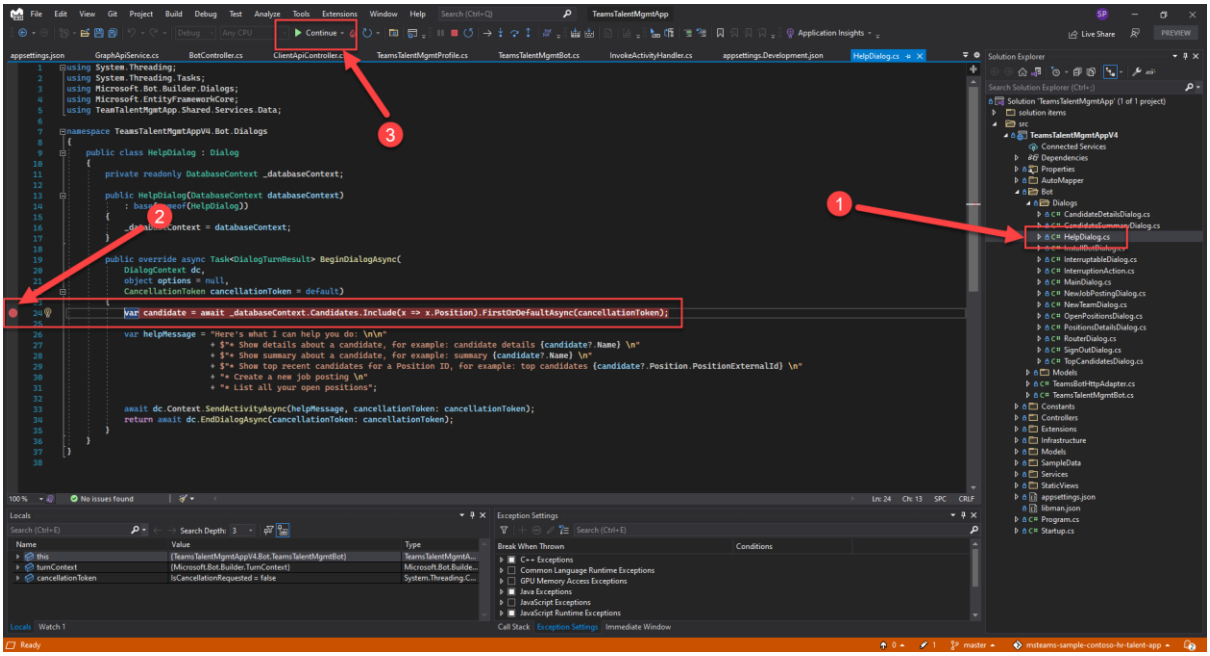
25. You should see that Visual Studio has halted execution and highlighted line 40 in yellow. This means that this is the current line of execution but the breakpoint has caused the application to pause. (Note that while you have halted execution the bot will not work until you resume the app by clicking the green play button from the toolbar - or pressing F5).



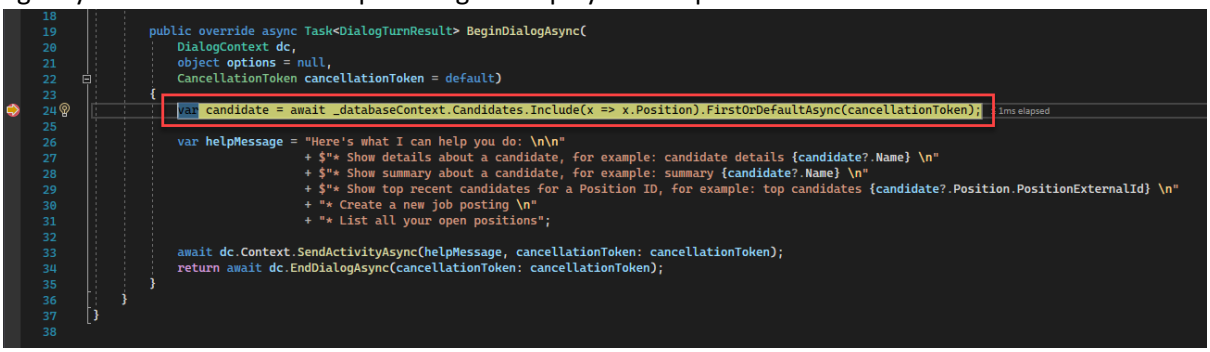
26. While we are paused Visual Studio allows us to inspect the value of any variable in the current scope. As we are in the scope of a method called **OnTurnContext**, we can inspect the parameters that were passed to it. To do this, hover your mouse over the parameter called *turnContext* then expand the various treeview items in the pop-out to see the values. In the case of the image below, we can see the text that was passed in the message.
Note: In bot framework, conversational concepts are used to describe the various parts of bot interaction. In the same way that two humans might take turns speaking in a conversation, a Turn in bot framework represents your bots chance to respond to an event.



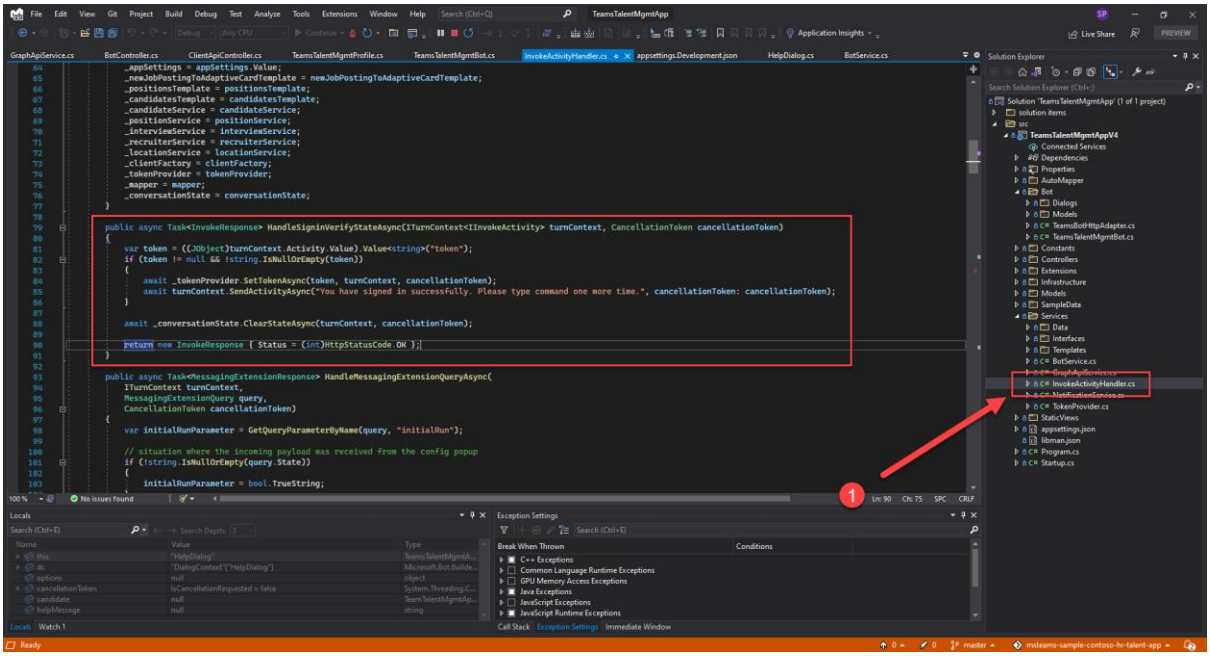
27. Another concept in bot framework is that of Dialogs. A dialog represents a part of a conversation. This could be a single text response or a conversation that spans multiple turns. It allows us to encapsulate the functionality of a particular conversation.
One such dialog is the HelpDialog. Open the HelpDialog.cs file and set a breakpoint on line 24. This click the green play button to resume the bots execution.



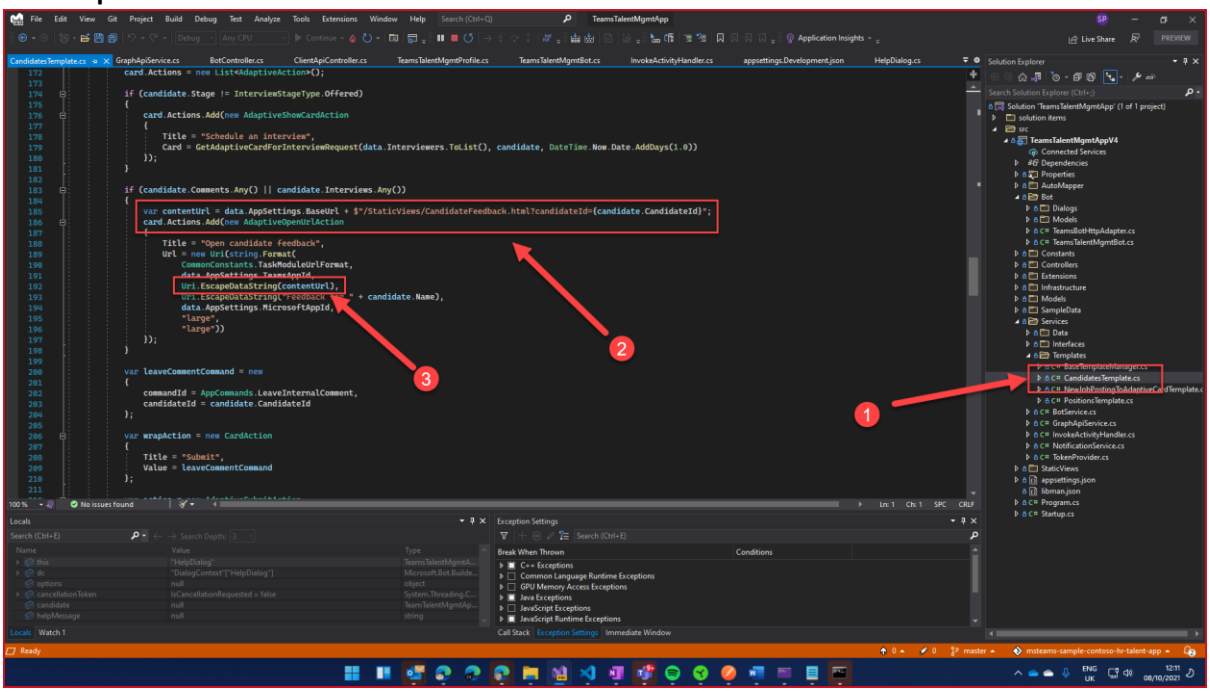
28. You will see that your breakpoint has been hit! As before you can inspect the various variable values if you wish.
    Finally, this method will send the **helpMessage** back to Teams.
    Note: If an execution takes more than a few seconds, Teams will ignore the response as it will assume that the bot has broken and is not responding. You may therefore find that when you resume execution from this point the help message may not actually appear in Teams. This is expected! If you remove both of the breakpoints and send the *help* command again you will see that the help message is displayed as expected.



29. There are two more areas of interest in the code that you may be interested to see. The first is the code for SSO. Most of this is handled for us in a dialog provided by the framework, but we still have to handle the users security token once it's been returned to us. This is particularly useful if you want to cache the token for use in the future, perhaps exchange it for a graph token to be able to call graph on the users behalf?
    To see how this sample has dealt with the token, open the **InvokeActivityHandler.cs** file and inspect the **HandleSignInVerifyStateAsync** method.



30. Another area of interest is how the bot responds with adaptive cards. In the CandiateTemplate.cs file is the code used to create a relatively complex card. The image below shows how the bot is configuring the candidate feedback task module to be displayed when **Open candidate feedback** is clicked.

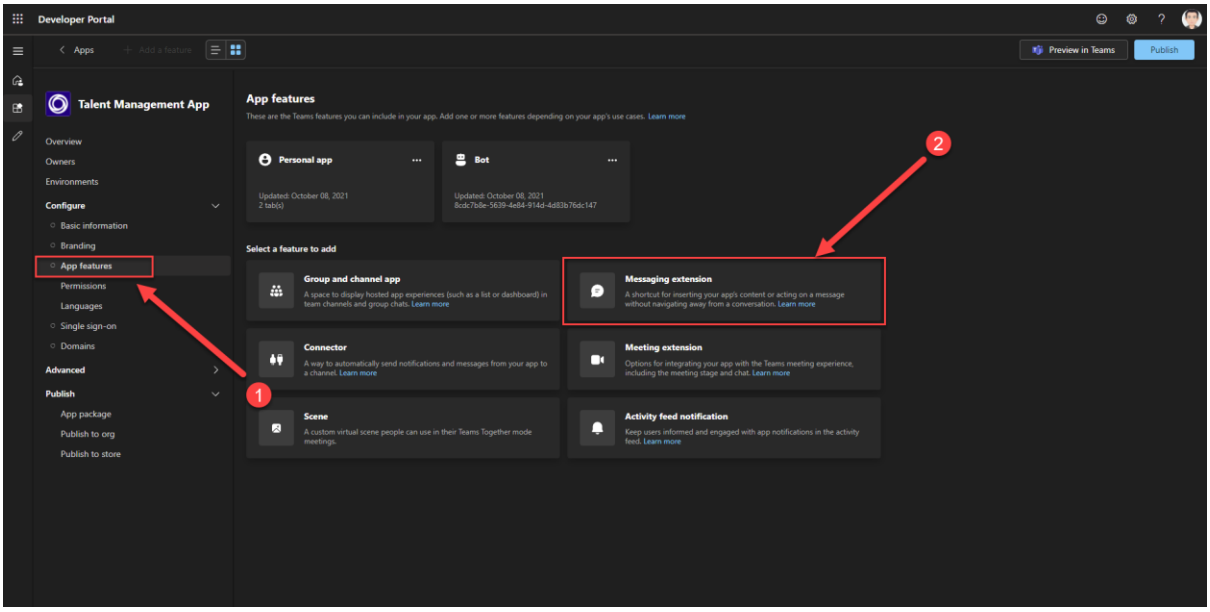


Now that the Tab is working…

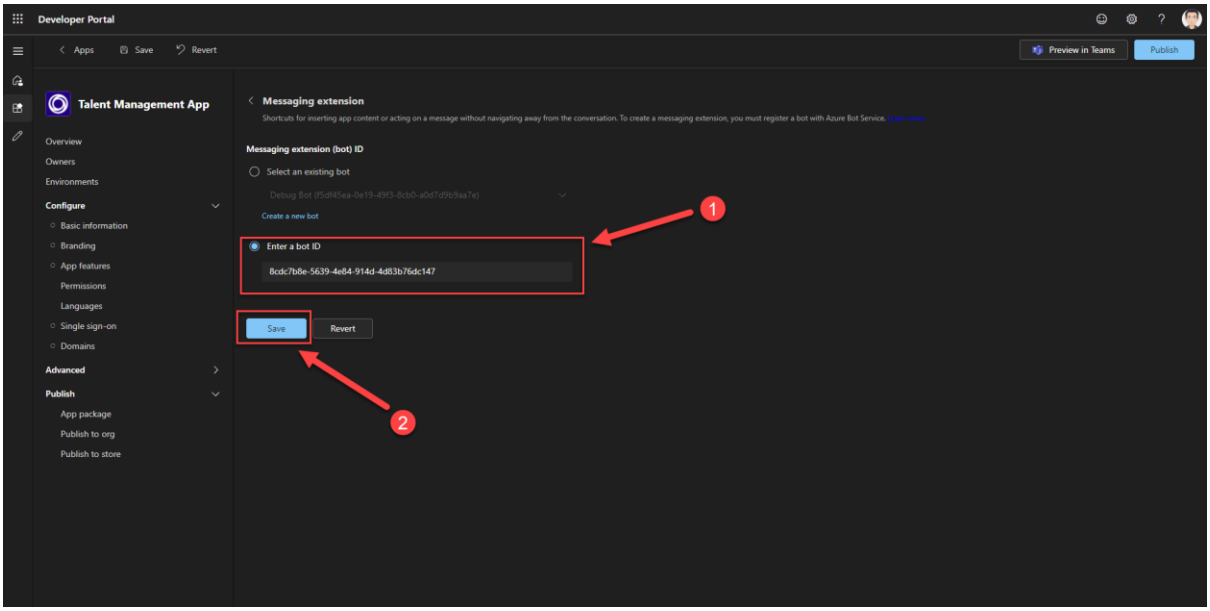
5) Implement a Message Extension Search

In this step we will extend the functionality of Teams to include a search function. Searching for candidates and positions using a message extension

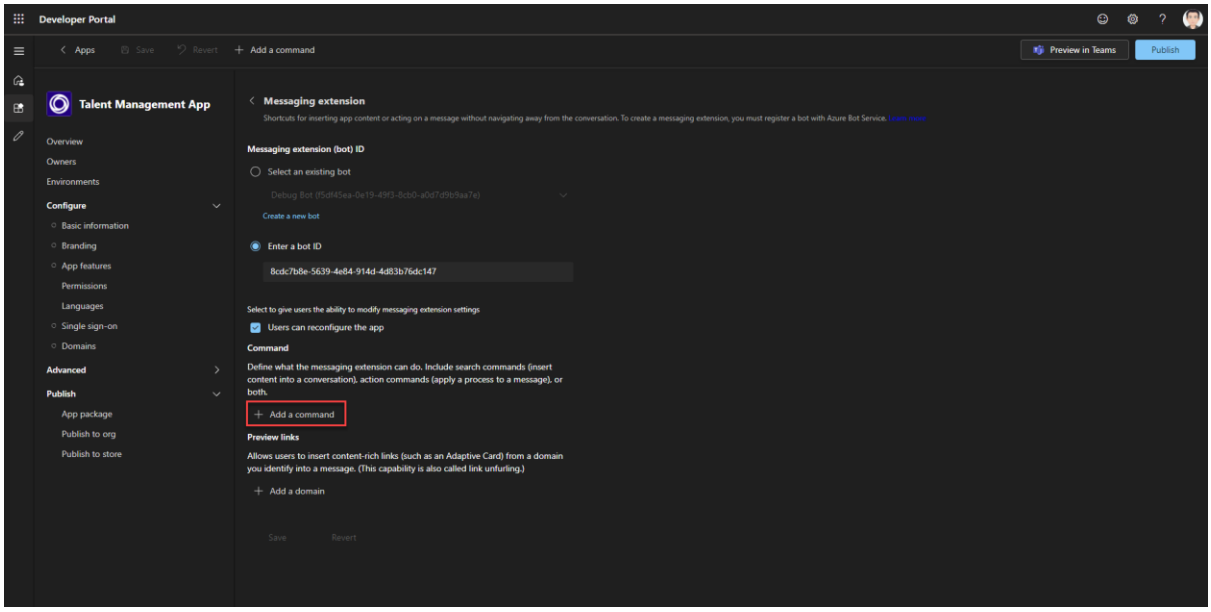
1. As with everything else in the manifest we need to go back to the Developer Portal to add a new feature. Select **App Features** from the left menu and click **Messaging extensions**.



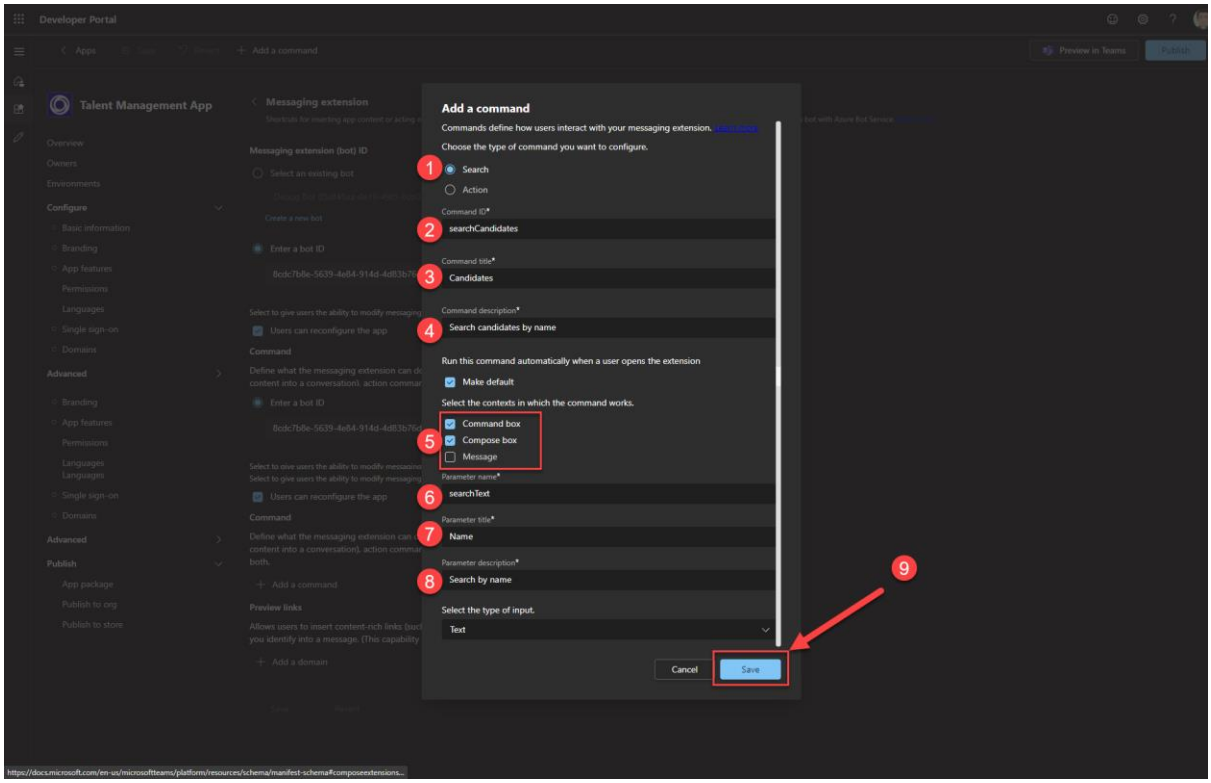
2. Select **Enter a bot id** and enter your App Registration *Client Id* into the textbox. (This is so that you can have a different bot handling your messaging extensions if you wish.) Once this is done, click **Save**.



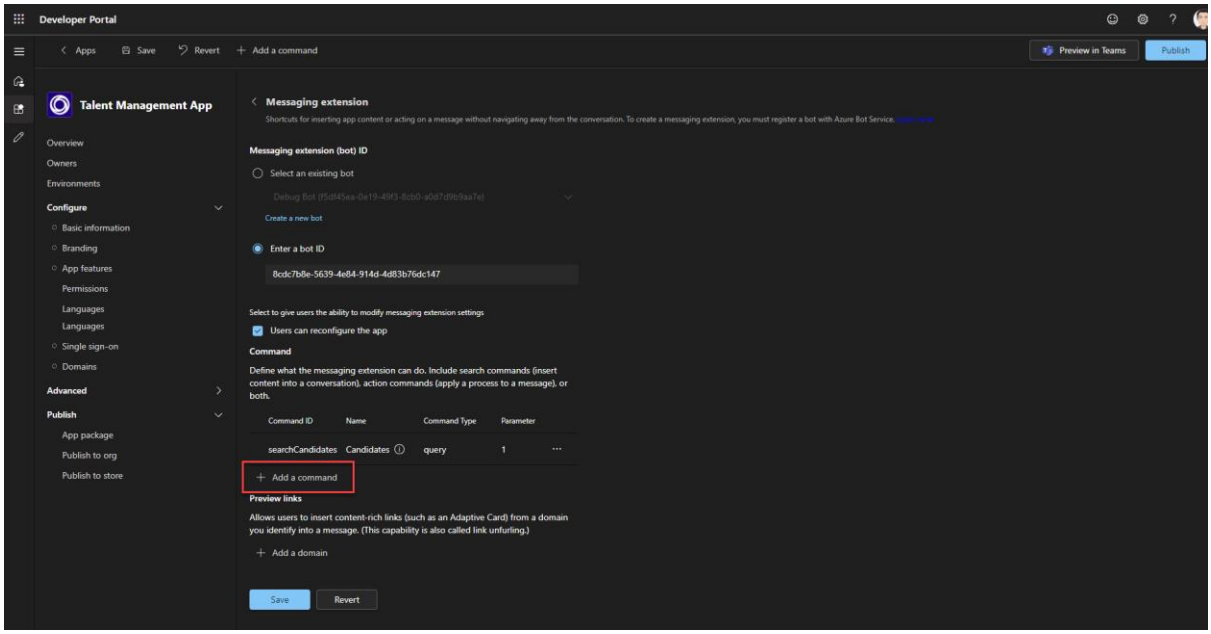
3. Now we need to add the commands. Select **Add a command**.



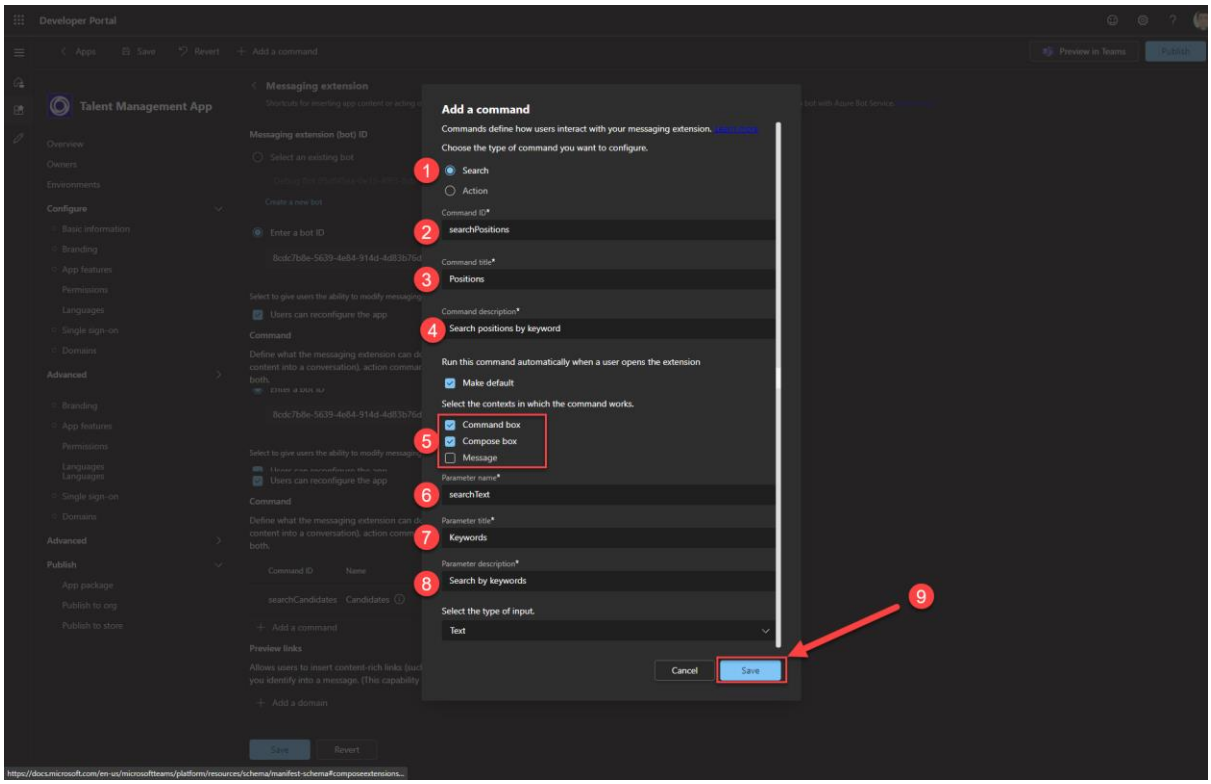
4. Enter the values below to add the search candidates search command and click **Save**.
Note: we are selecting Command box and Compose box as the areas within Teams that this function will be available. This is the large search box at the top of the Teams interface (Command box) and the area you type a message respectively.



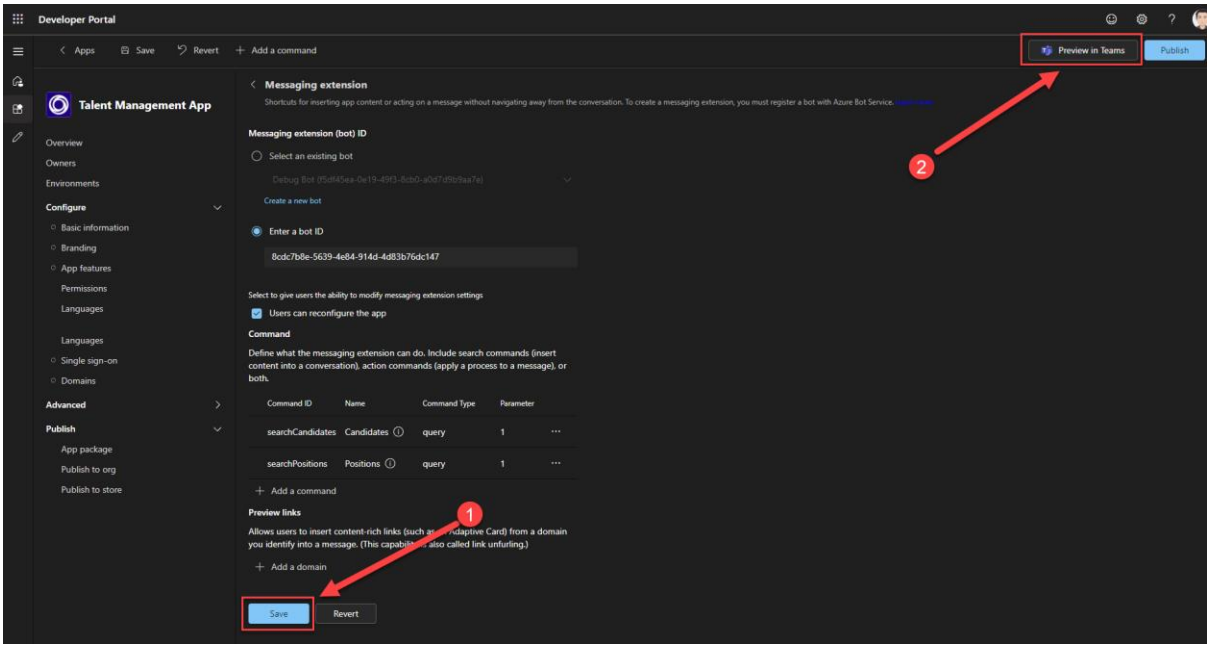
5. Once done, we will add a second search command, this one is for the position search function. Click **Add a command** again.



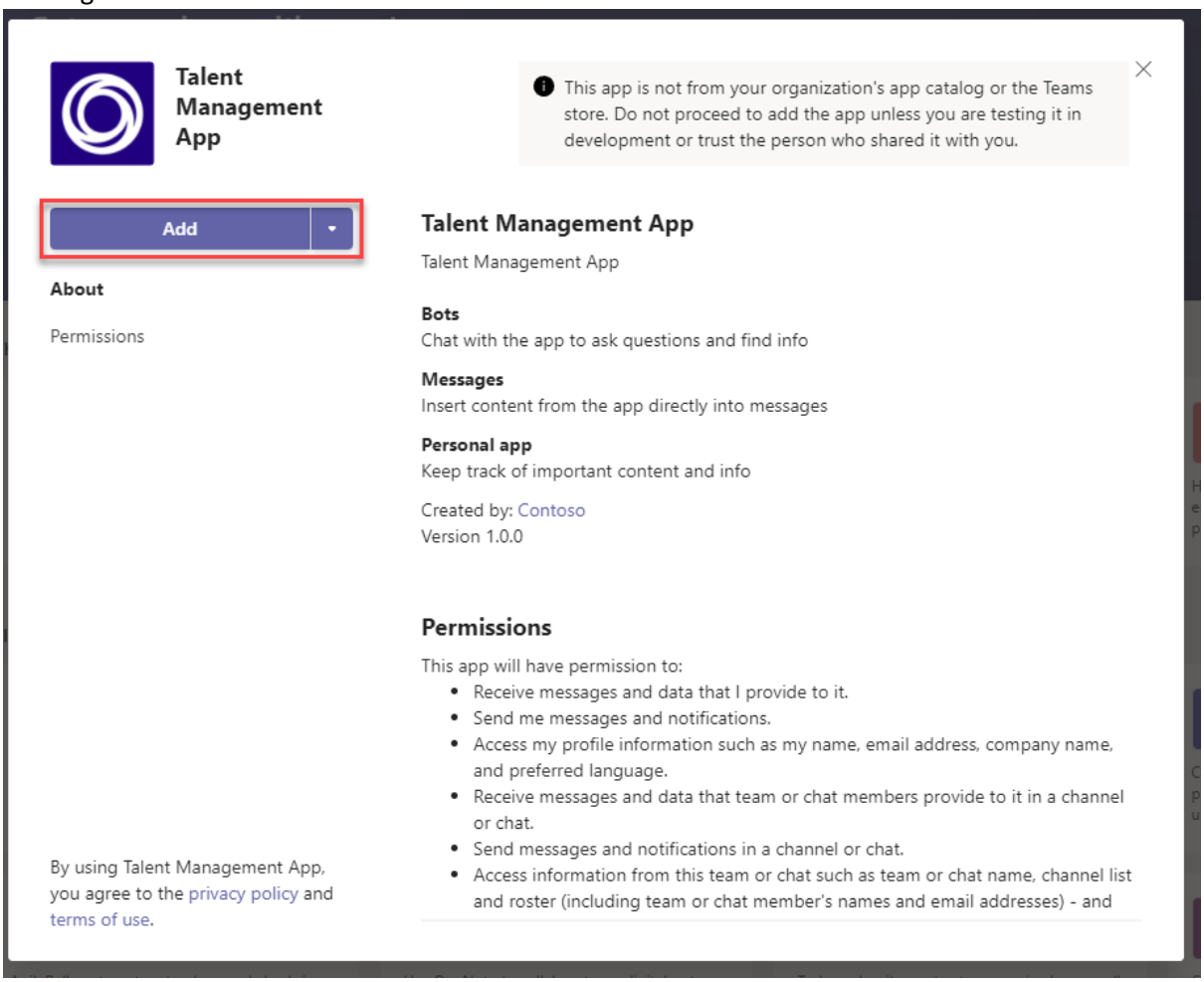
6. Enter the values below and click **Save**.



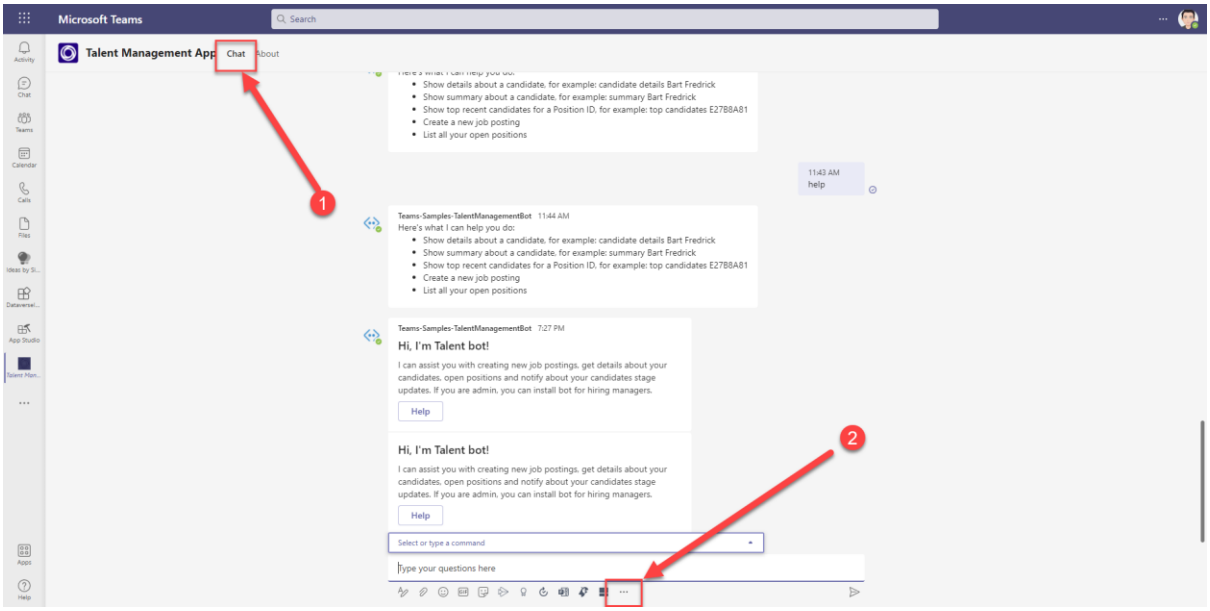
7. Now you can see the two commands have been added, click **Save** and then **Preview in Teams** to test it out.



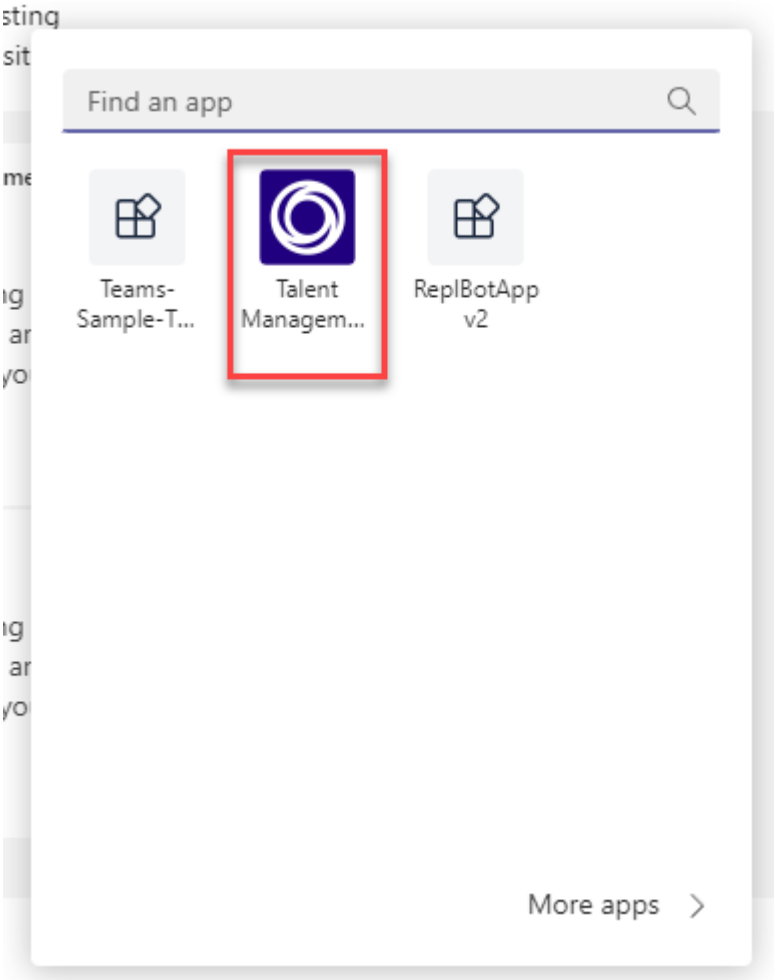
8. As before you will be prompted to “add” your app and as before this will overwrite the existing manifest inside Teams.



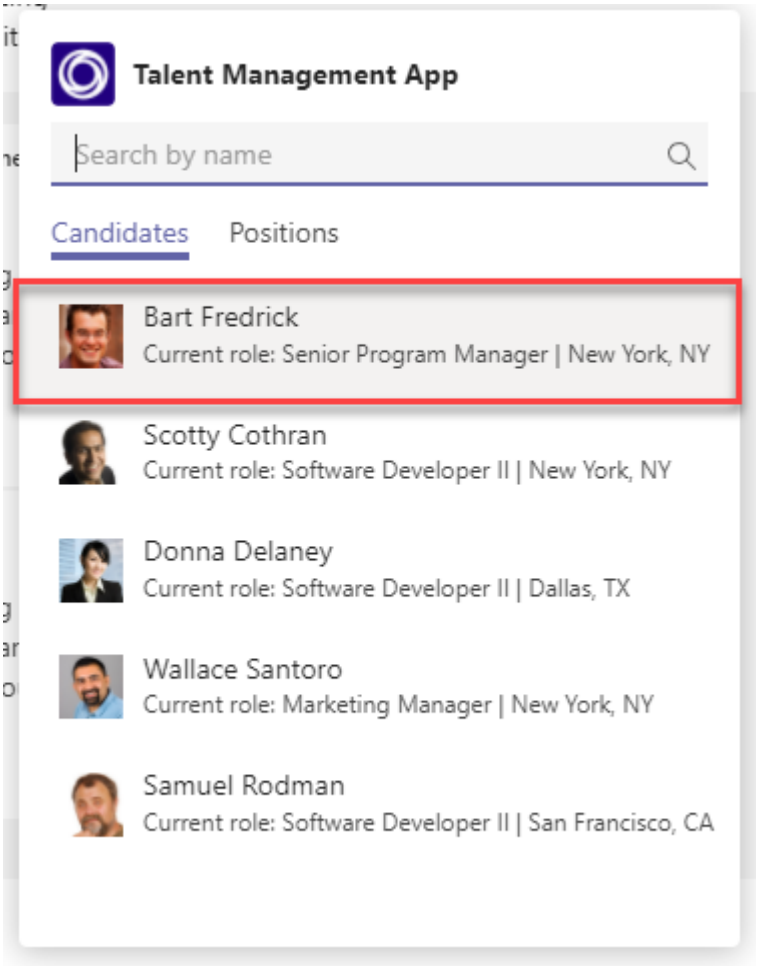
9. Ensure that you are in the Chat tab for your application. (If you have completed this document in order you will see other tabs here, but **Chat** is where we interact with our bot). Then click the ellipsis (…) next to the icons below the compose box.



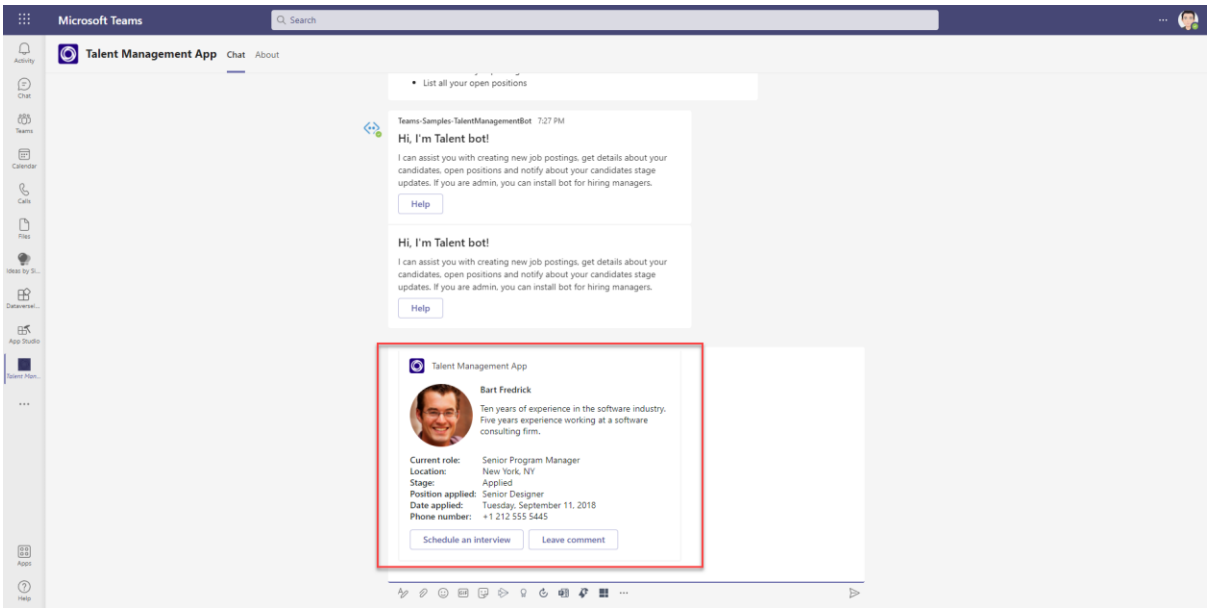
10. Within the dialog that's displayed you should see your application listed, if not you can use the **Find an app** textbox to search for it.



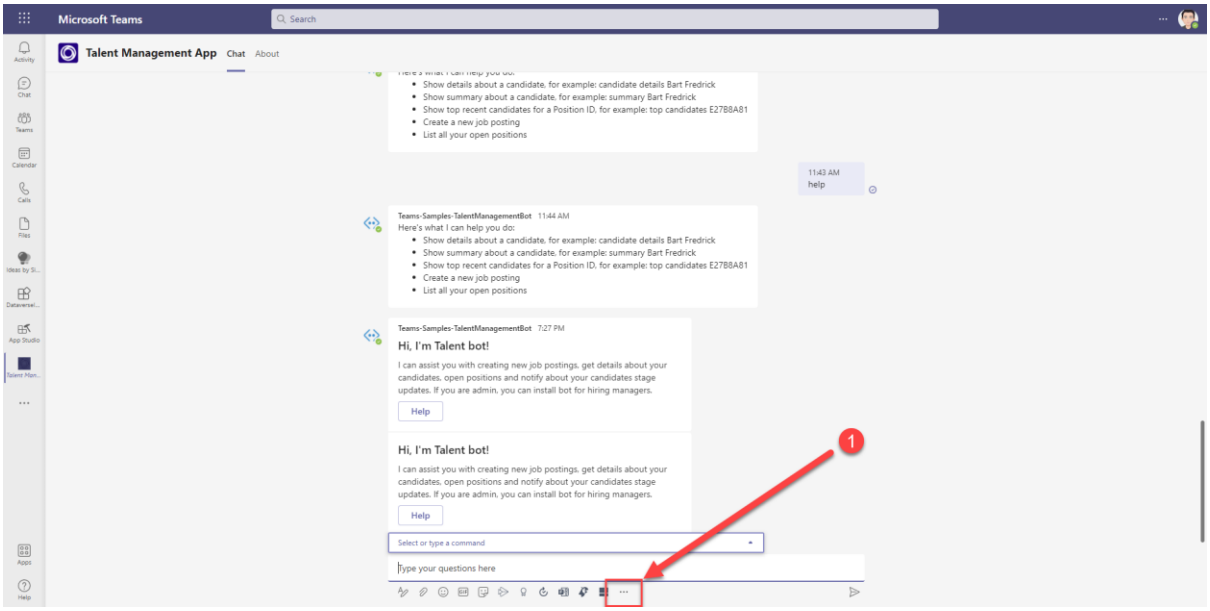
11. Selecting your application will display the following dialog. The things to note here are that there are two search functions **Candidates** and **Positions** shown as tabs below the **Search by name** textbox. Select one of the candidates.



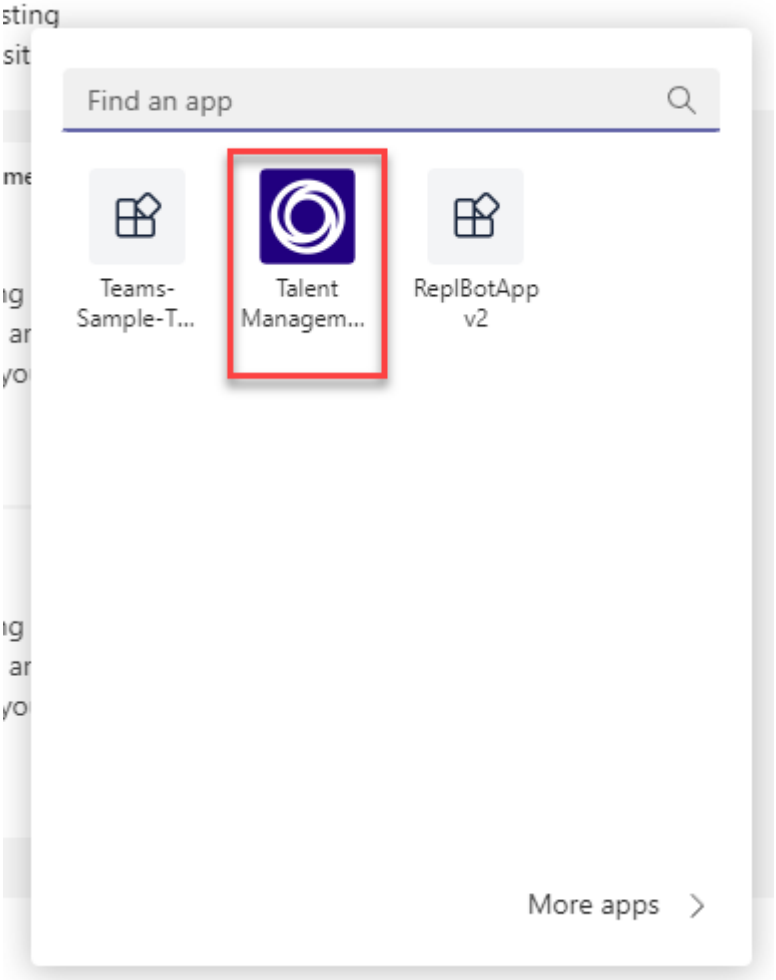
12. This will place the adaptive card we saw earlier in the compose box. It's important to note that message extensions don't post directly to the chat, instead they add their content to the compose box and allow you to include more text before choosing to send the content.



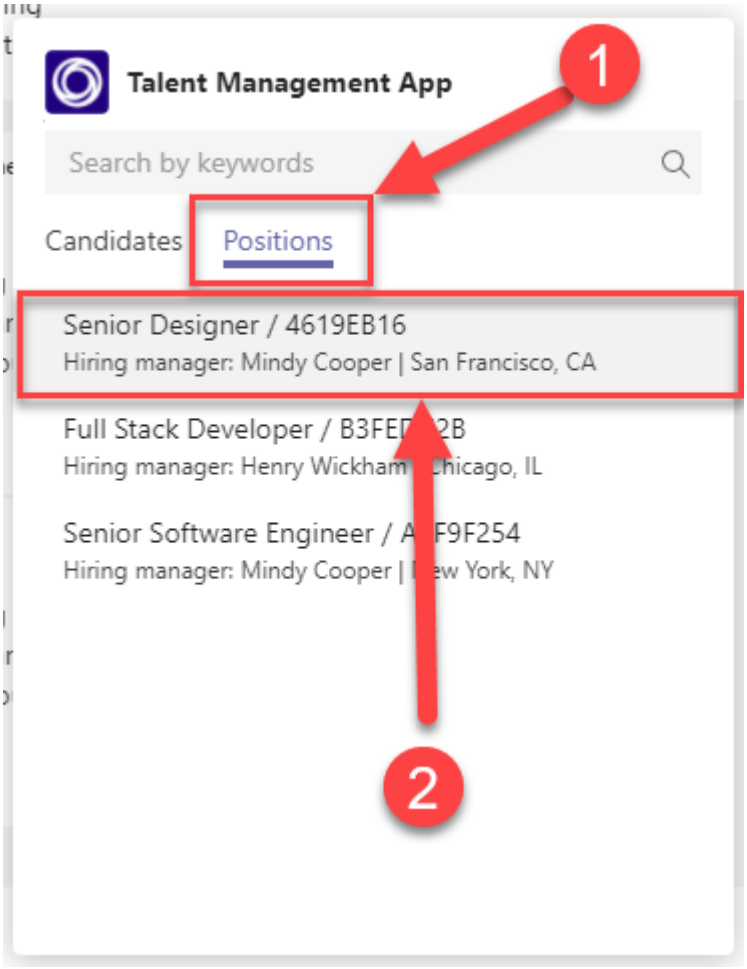
13. Repeat this process to see the positions adaptive card. Click the ellipsis again (…).



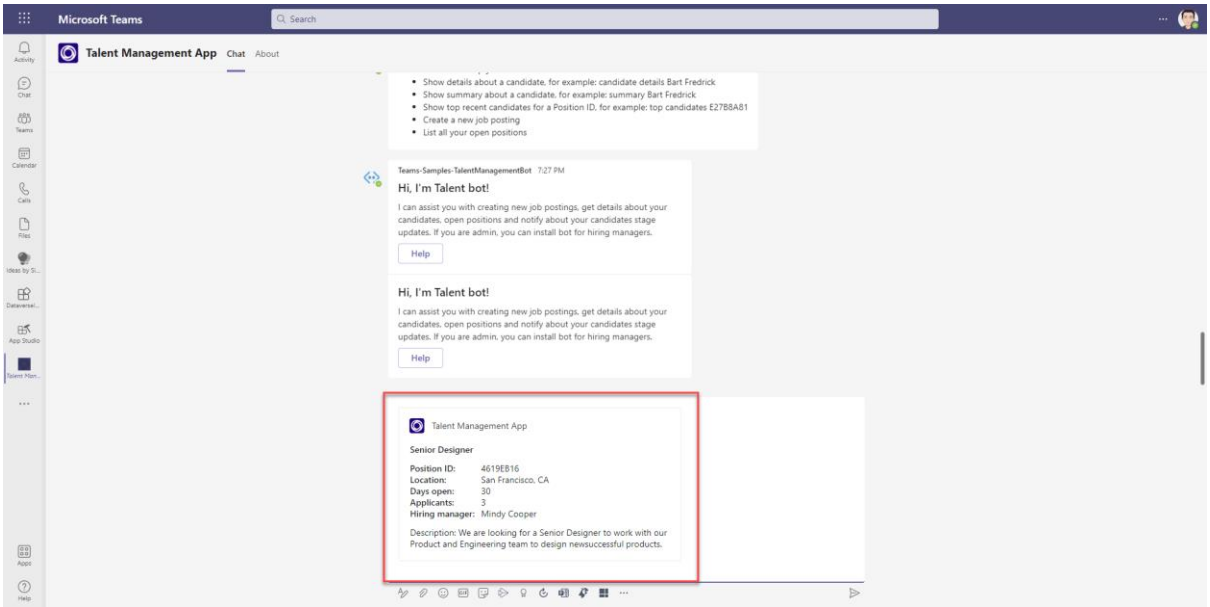
14. Find your application



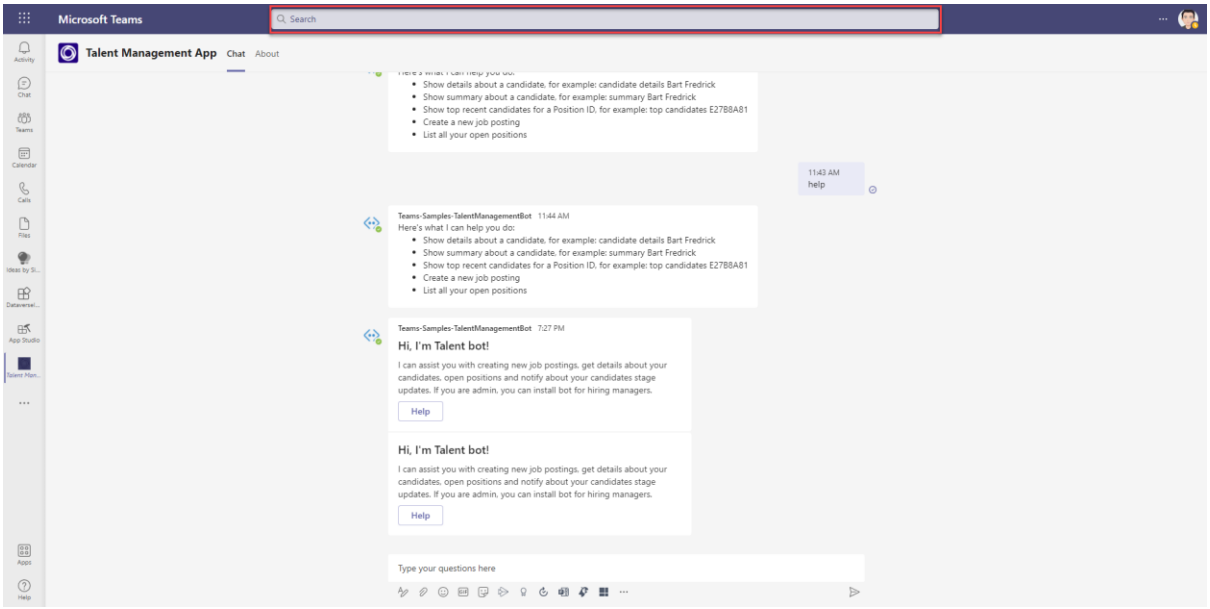
15. This time, select **Positions**. You will see a list of positions displayed. Now select one.



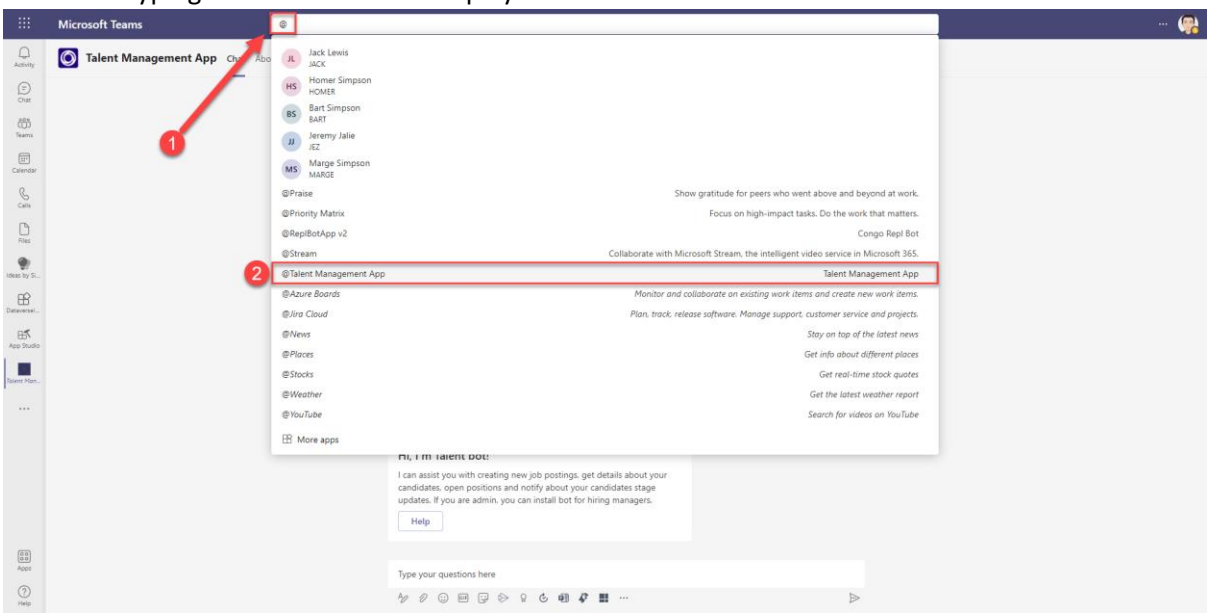
16. As before you will see that an adaptive card has been added to the compose box ready to be sent!



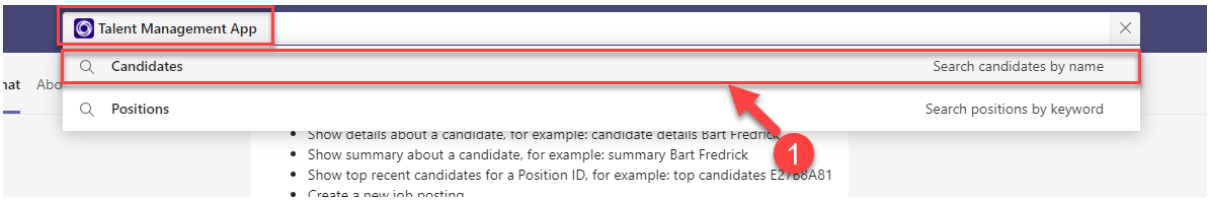
17. Another feature of a search command is that it can be made available to the global search bar (Command box) at the top of the Teams interface.



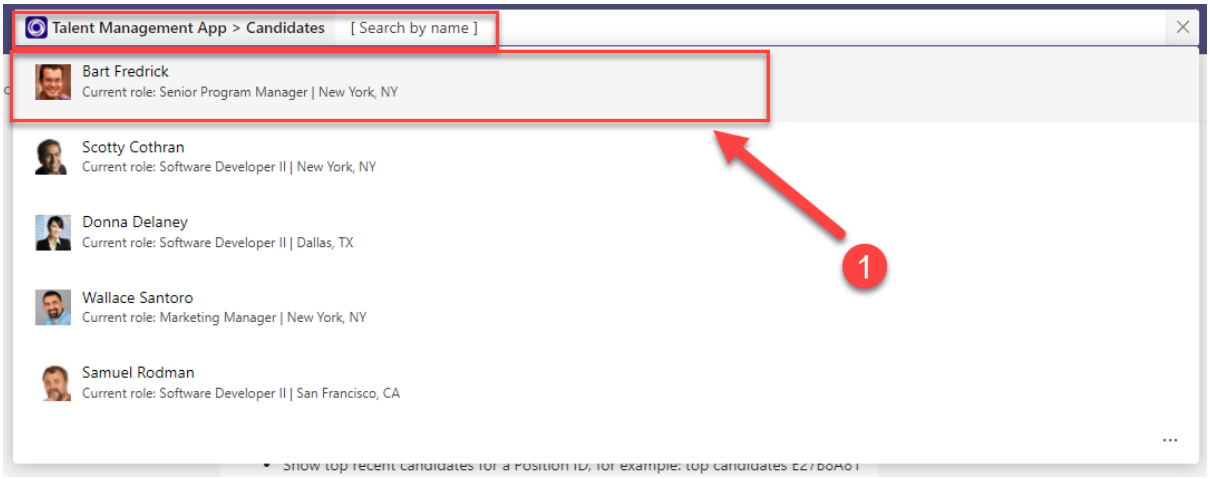
18. Typing an @ symbol will show a dropdown with a list of installed bots. If you can't see yours then start typing the name until it's displayed and then select it from the list.



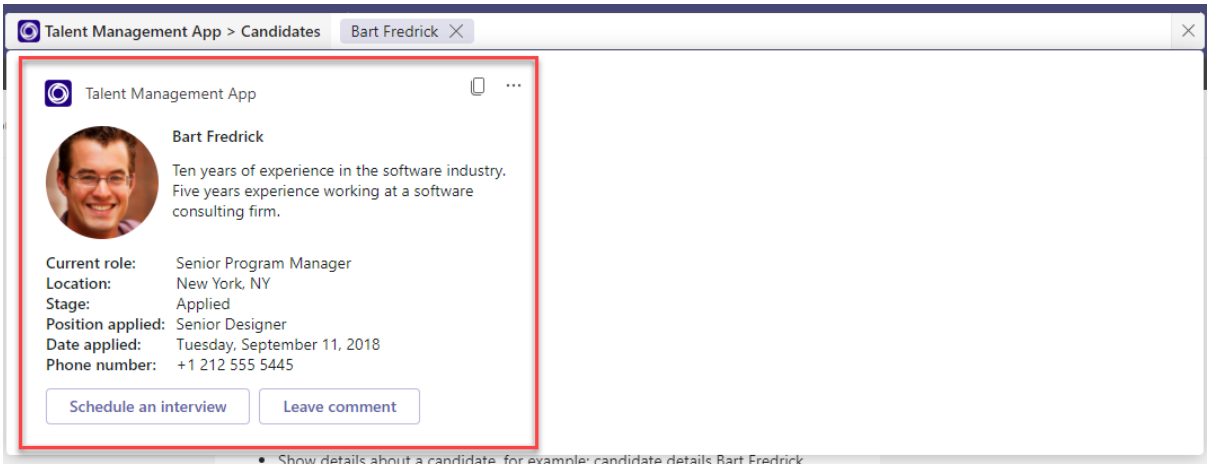
19. This will change the context of the search box to your bot only and once in that context you will be prompted with all the available search functionality exposed by our bot. In this case **Candidates** and **Positions**. Select candidates to see a list of candidates.



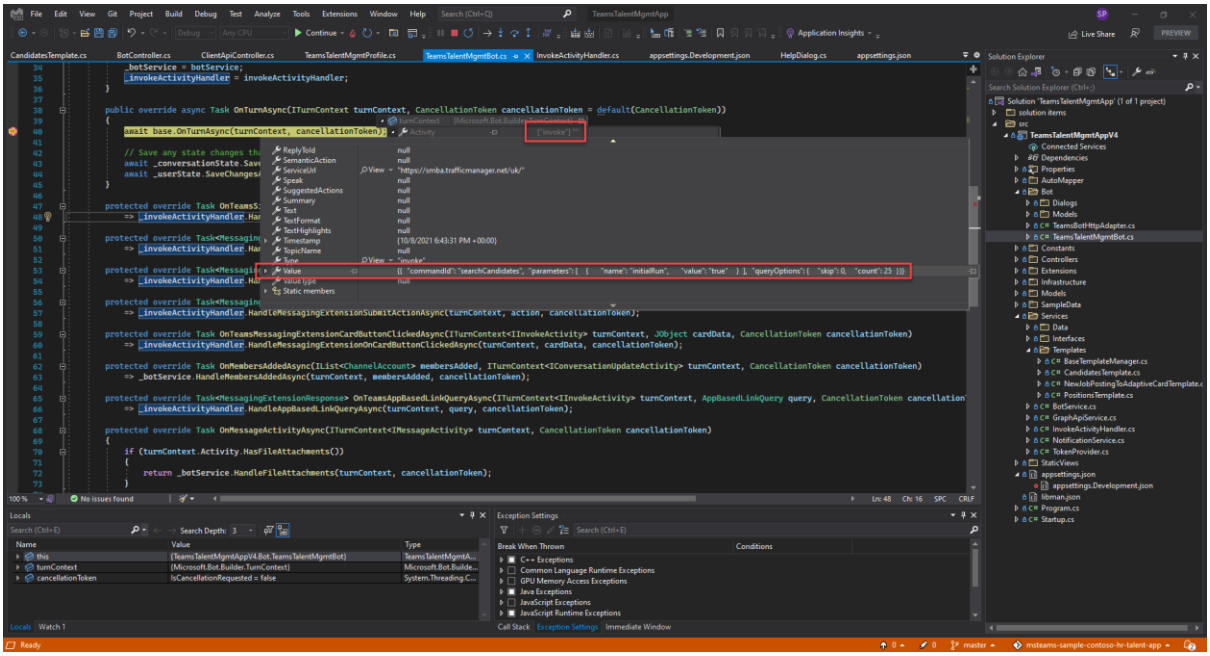
20. The search box has changed context again as is now only working within the Candidates search function. You can search for a candidate by name, or select one from the list.



21. Unlike the compose message extension, the adaptive card is displayed directly in the search box dropdown, but works like any other adaptive card.



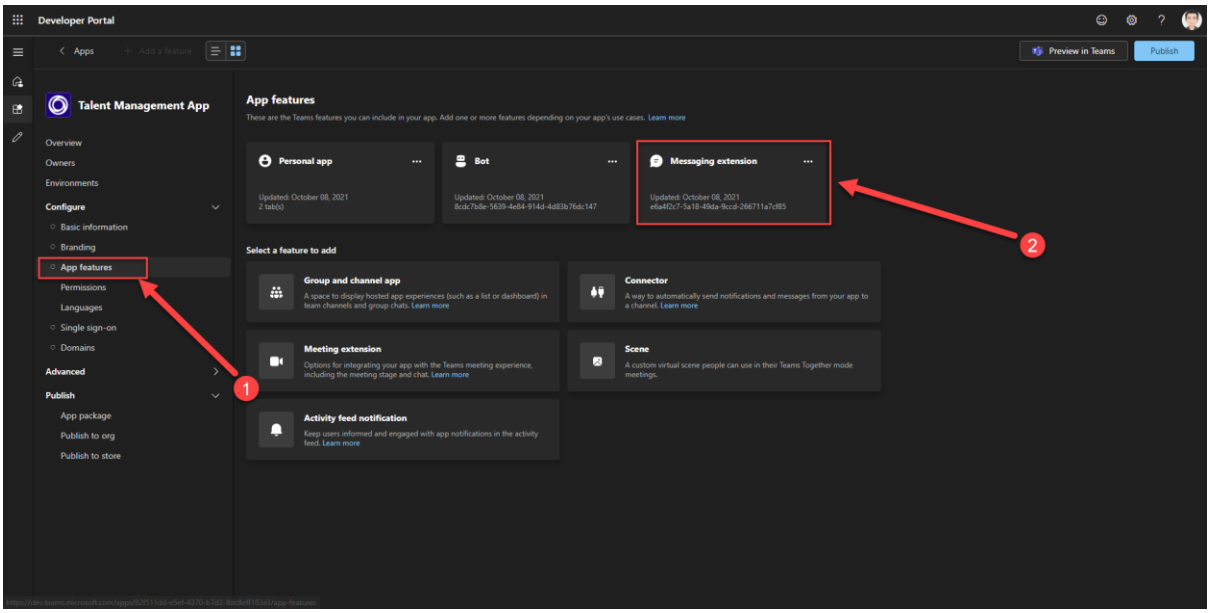
22. To give you an idea of how this works, let's open Visual Studio and inspect some variables! Open the TeamsTalentMgmtBot.cs file and set a breakpoint on line 40. Once you invoke the search by either using the compose message extension or the command box your breakpoint should be hit and you can inspect the values that make up the search command.



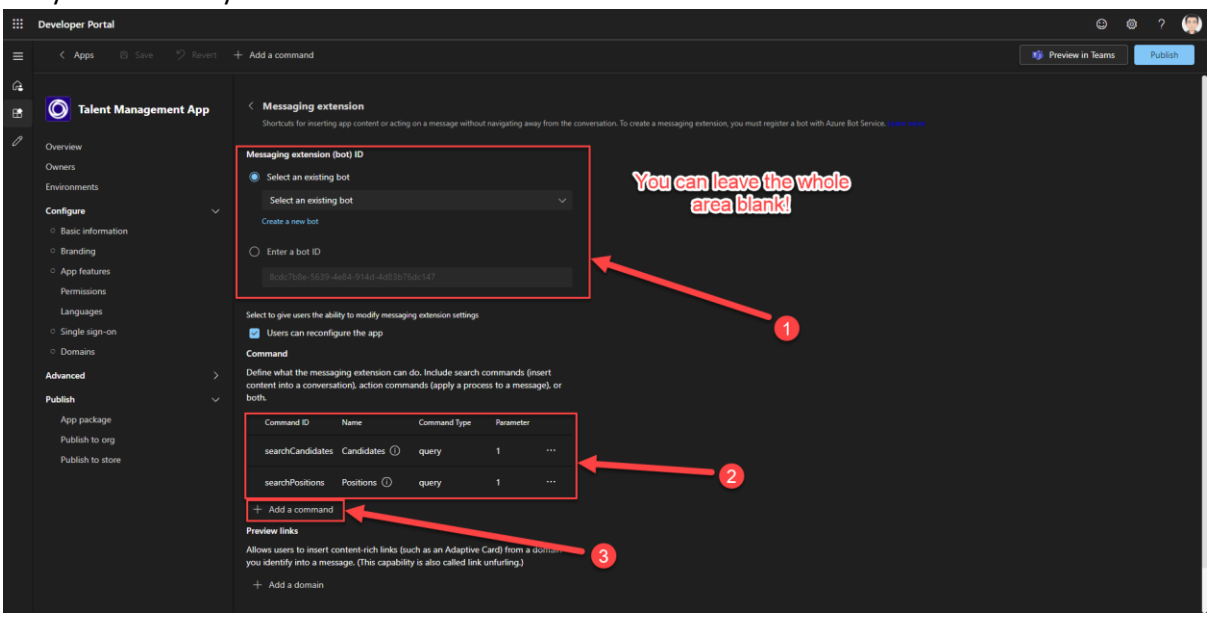
6) Implement a Message Extension Action

In this step we will add another message extension to perform and action instead of a search.

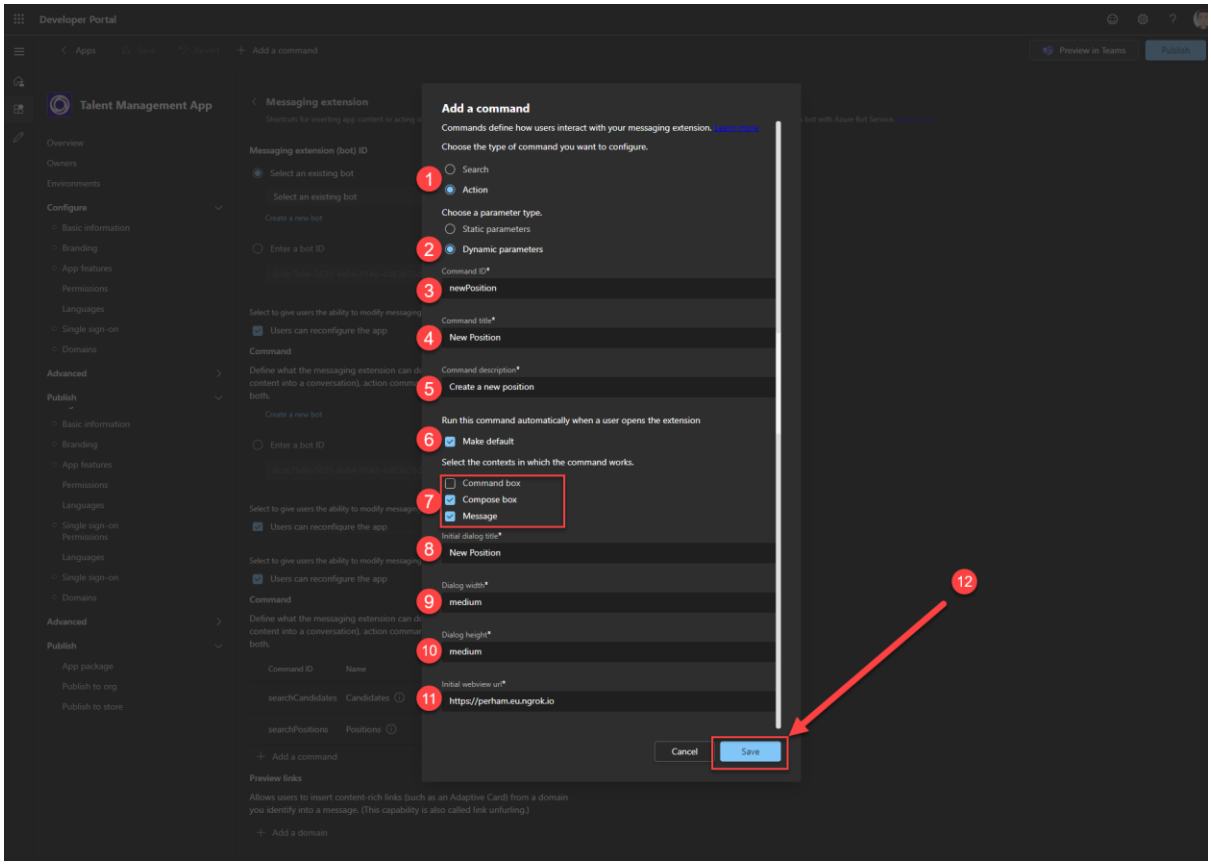
1. Open the Developer Portal, navigate to App Features and select Messaging extensions to edit the manifest.



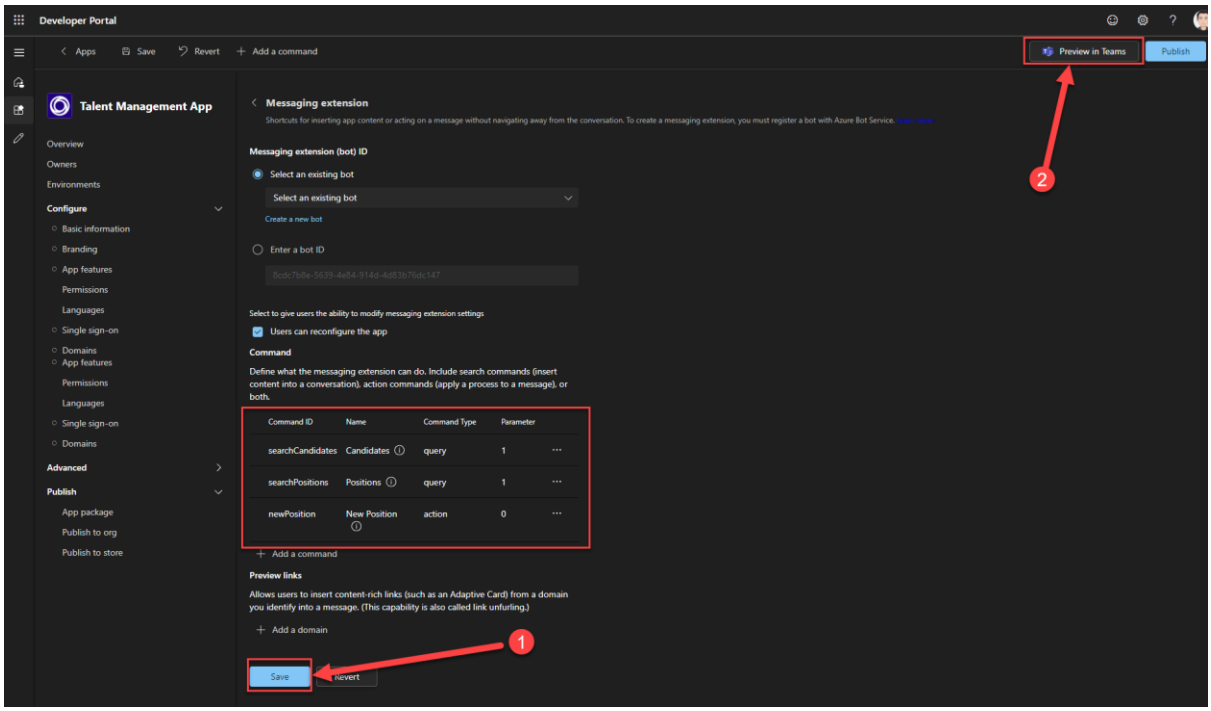
2. Here we will add a new command, if you can see the search commands listed then you are already in the context of your bot and don't need to select anything from the top section. If not you can enter your bot id as before. Then click **Add a command**.



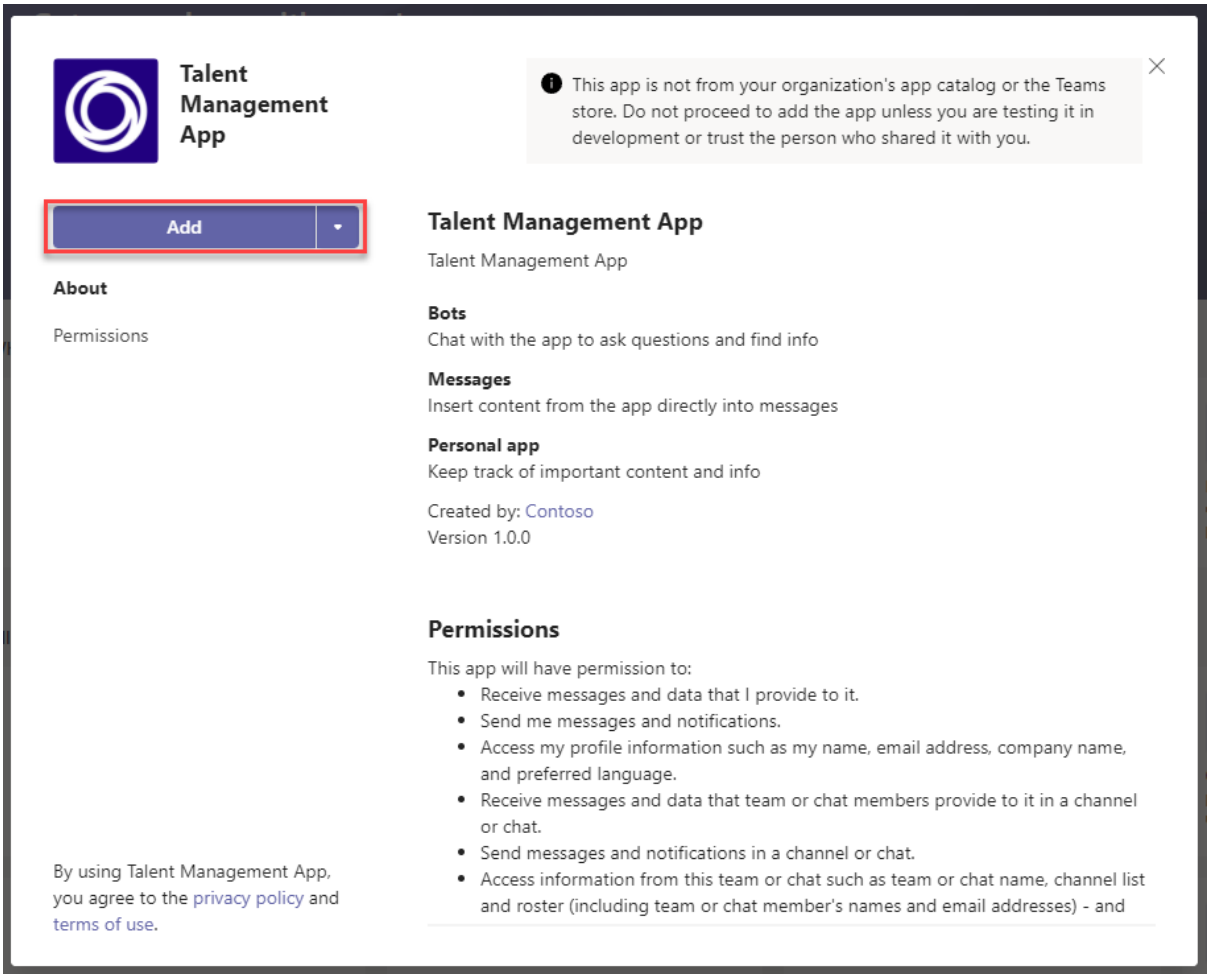
3. Complete the dialog as shown below, then click **Save**.



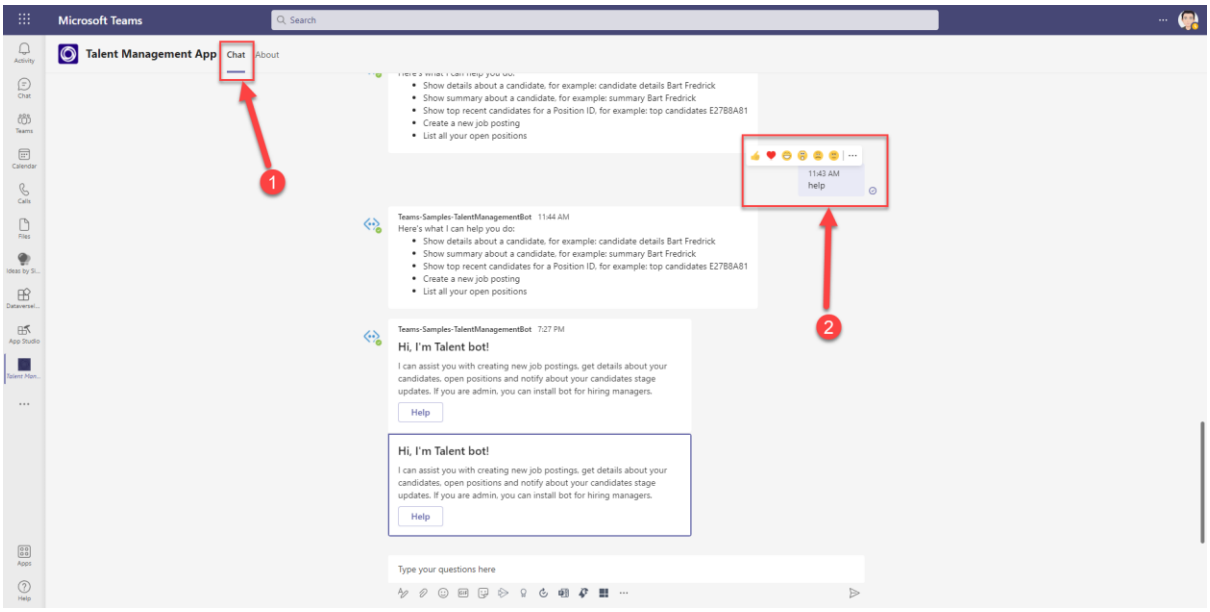
4. Ensure that you save the manifest by clicking **Save** and then we can test it. Click **Preview in Teams**.



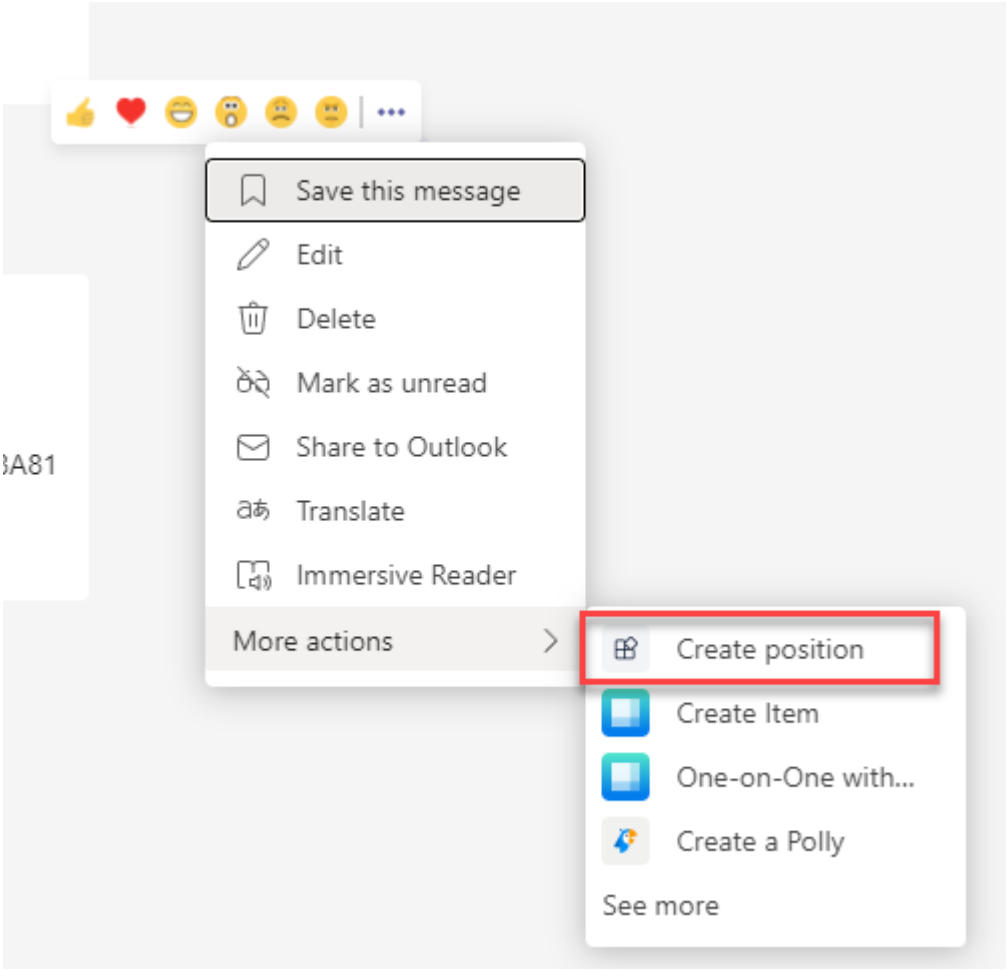
5. Again we will be prompted to add the app to Teams.



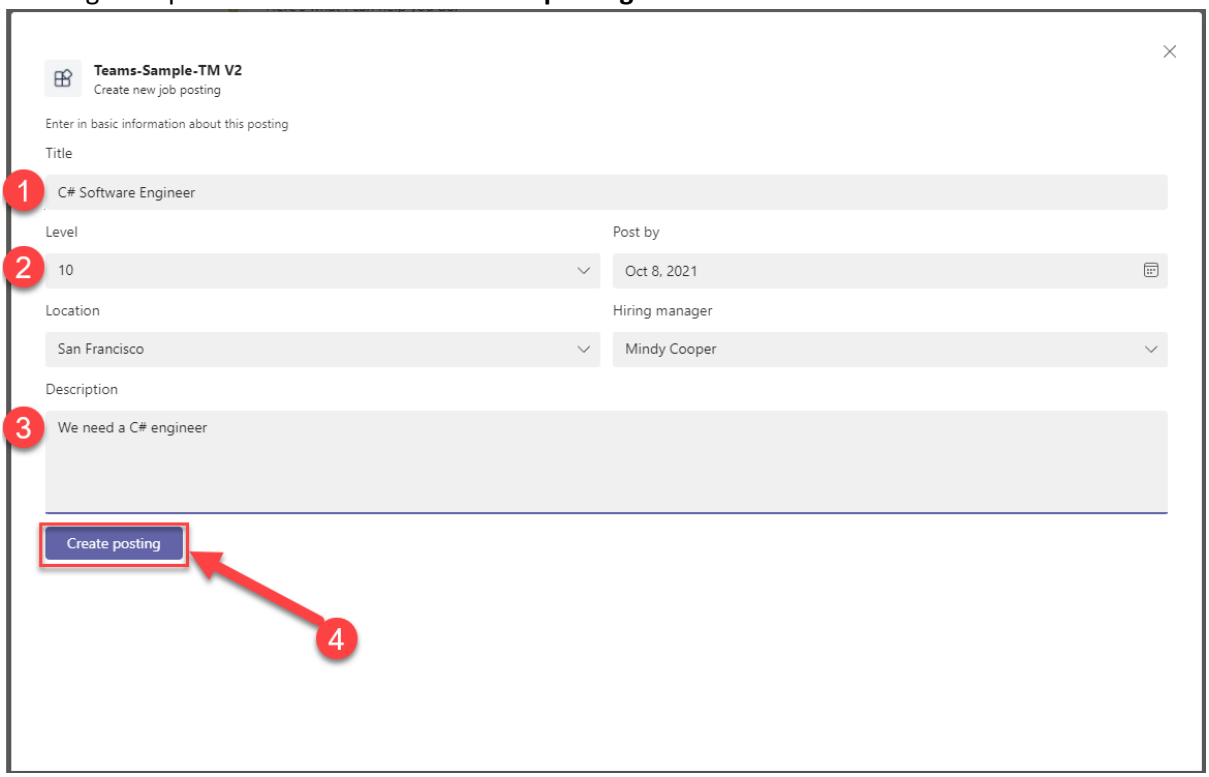
6. Again ensure that you are in the Chat tab of your application. We configured the action to work on messages so we can invoke it by hovering over an existing message, and clicking the ellipsis (…).



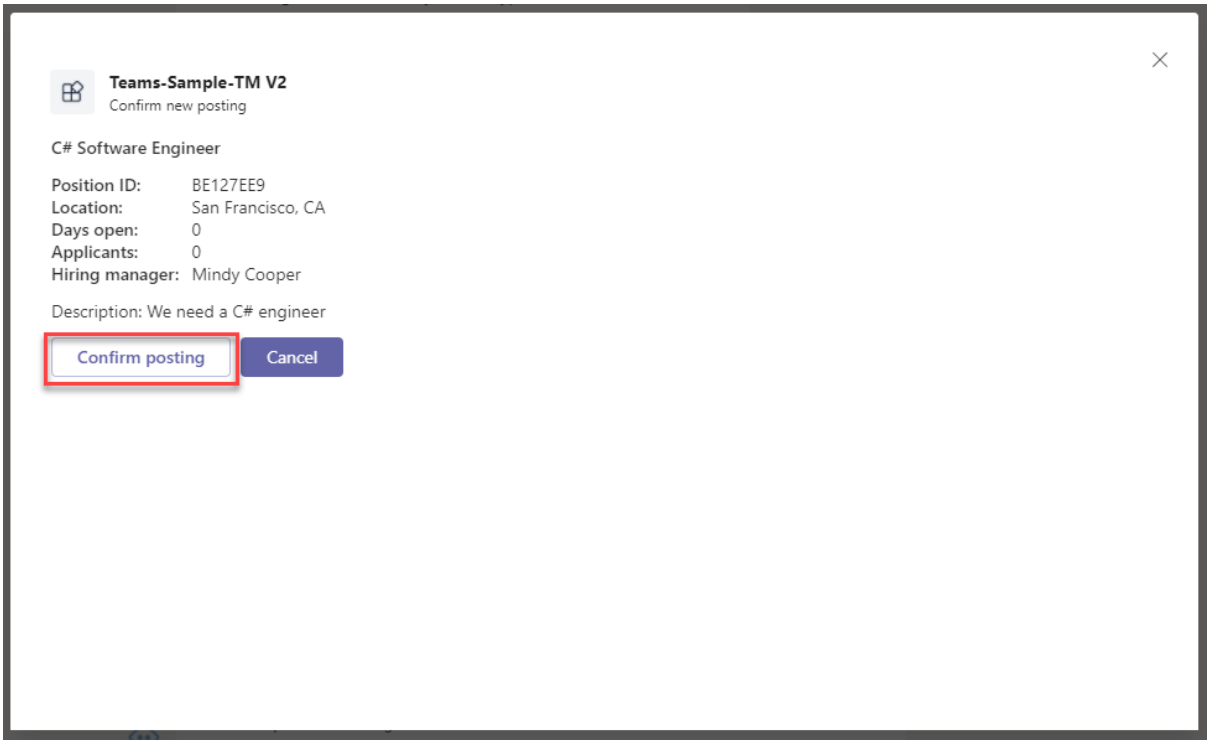
7. From the menu that's displayed, click **More actions** and then **Create position**.



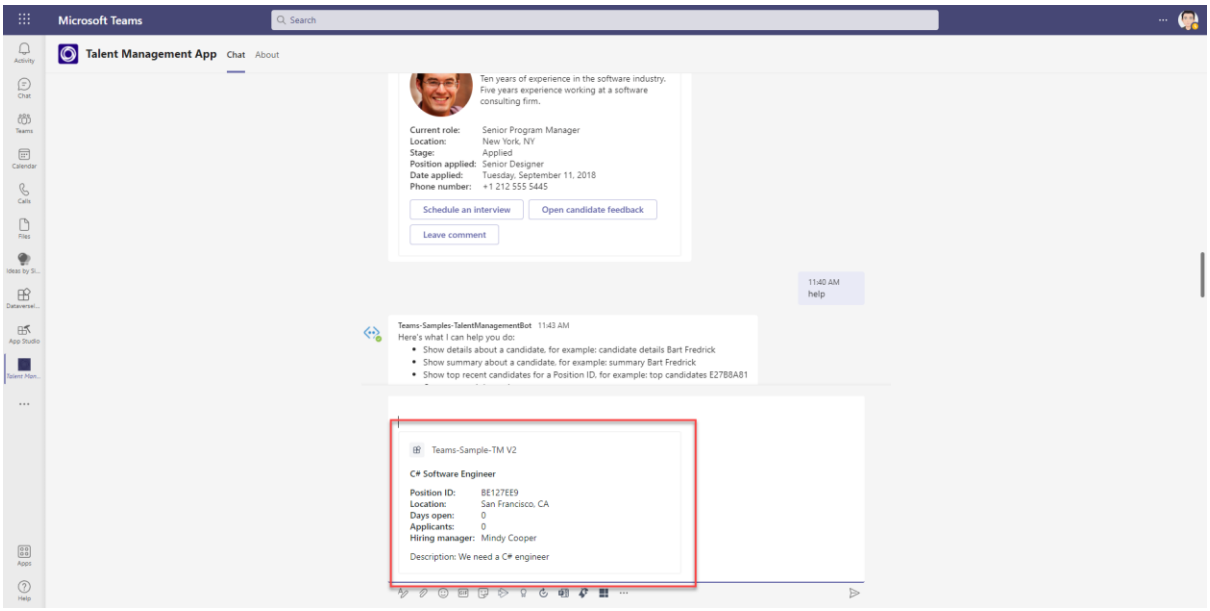
8. This will display a task module prompting for information pertaining to the position you are creating. Complete the form and click **Create posting**.



9. You will then be asked to confirm the details before clicking **Confirm posting**.



10. At this point the position has been saved and will be available through the tab or search functions shown earlier. It will also put an adaptive card in the compose box to allow you to send it to the chat if you wish!



Congratulations! You have now successfully completed this lab. We hope that you found this lab, and the associated lab materials useful. We look forward to seeing what Teams Apps you build as a result of attending this lab!

Lab Complete.