

TEAMS APPS

HANDS-ON LAB

Using the Teams Platform to surface a Recruitment application in Teams

ABSTRACT

This lab document contains a step-by-step guide to create a Teams App that enables a recruitment team to manage the job positions and potential candidates who have applied for those job positions all within Teams.

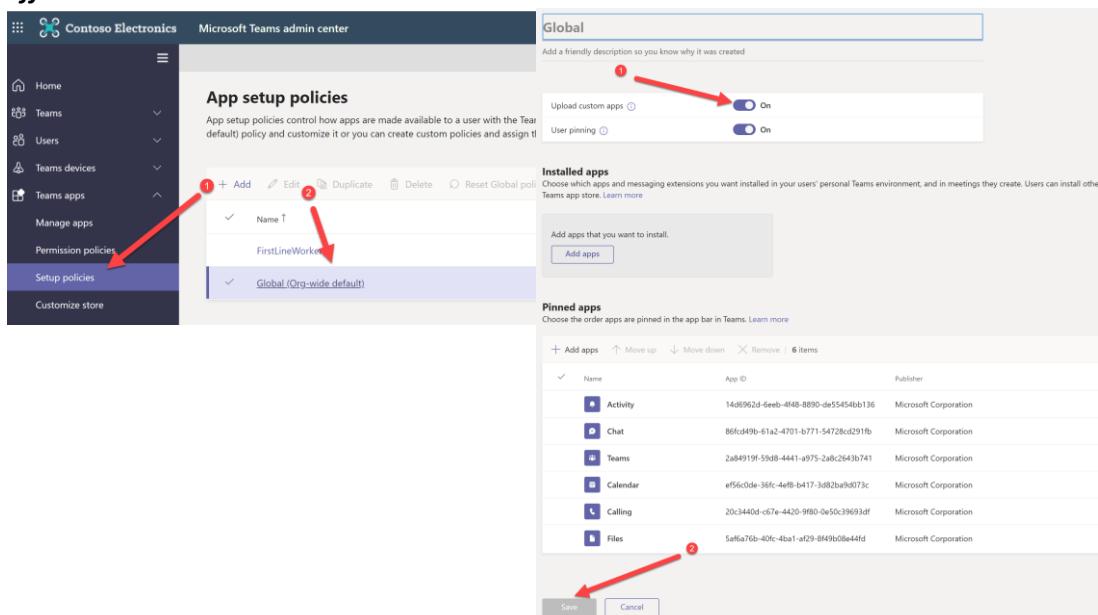
About

Lab guide v2 created by Jack Lewis and Scott Perham of Microsoft UK. Credit to the original team behind this Teams app hands on lab, found here - <https://github.com/OfficeDev/msteams-sample-contoso-hr-talent-app>

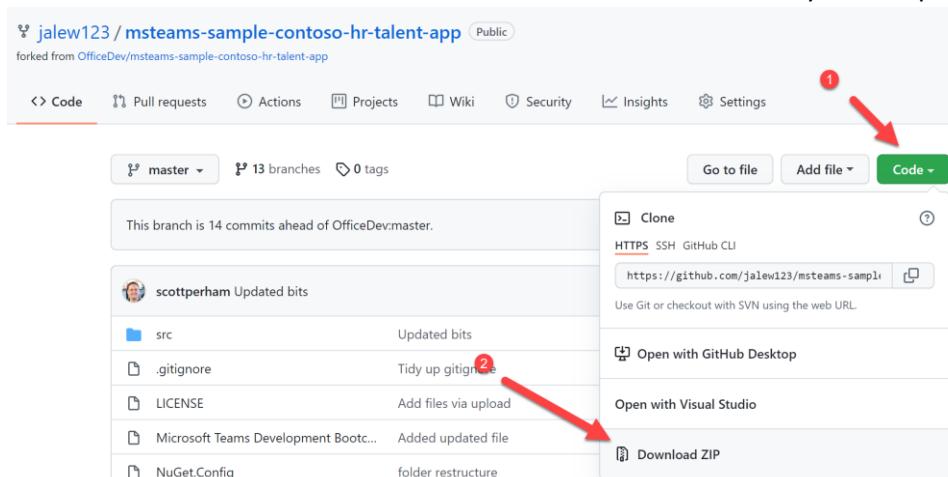
Getting Started

You will need the following to complete this lab:

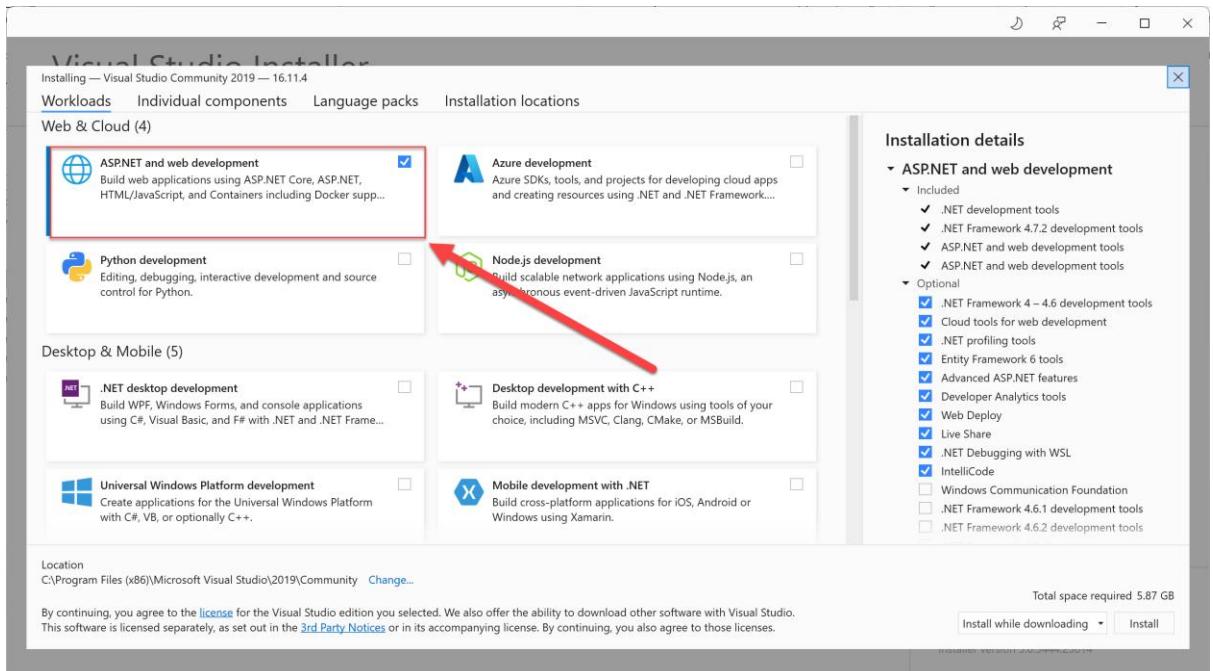
1. **A Microsoft 365 E5 Tenant**, which can be obtained from <https://cdx.transform.microsoft.com/> (for Microsoft Partners) or by signing up to the Microsoft 365 Developer Program – [Welcome to the Microsoft 365 Developer Program | Microsoft Docs](#). Alternatively, a Microsoft 365 E5 Trial will also 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 6 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. We highly recommend signing up for a free NGROK account, as without this, your tunnels will expire in 2 hours.
7. **Postman**, a tool that allows you to send HTTP requests from your computer, which will be used during the Microsoft Graph API section of this lab, and is also a useful troubleshooting tool, 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 (Mandatory reading before you begin!)

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 the hiring and candidate management flow of new talent and make it more interactive and responsive for HR teams and interviewees.

Before you attempt to begin and complete the lab in this document, please make sure all the pre-requisite steps have been completed and your machine is ready for the labs. These pre-requisites can be found in the 'Getting Started' section at the very beginning of this document.

It is strongly recommended that you complete each section of the lab, in the order of this document. While you may be able to jump around, it is likely that you will miss key concepts/configuration if you skip steps in this lab.

If you are unfamiliar with Teams Apps and the interfaces that are available for you to expose a third-party app's capabilities in Teams, I would highly recommend reviewing the following documentation, which will introduce you to the interfaces, at a high-level, that you'll be working with throughout this lab - [Build apps for the Microsoft Teams platform - Teams | Microsoft Docs](#) .

I'd also encourage everybody to visit the app store in Teams, and install some popular apps, such as Polly, and see how they work and allow you to interact with the third-party app, all inside the Teams interface. This will allow you to understand some of what is possible via the various interfaces, and for developers who work at our partners, it will allow you to get an initial understanding of how multi-tenant SAAS apps are offered as Teams app via the Store.

A key concept to understand is how auth works in Teams apps. From our experience, working with Partners in the UK, the most difficult/time-consuming work is typically aligned to getting SSO working for your Teams app. Remember that; all users of Teams have signed into Teams using an Azure Active Directory Identity, there is no other way to get access to Teams. For this reason, it makes sense to configure your application to work with Azure AD SSO, specifically OpenID and OAuth for Teams Apps (not SAML), as this allows you to do a silent, non-interactive, retrieval of the SSO token, via Azure AD, by using the Teams JavaScript SDK and allows you to work with the Microsoft Graph API. Another key concept to understand with Teams and SSO, is that because your app is being delivered inside an iFrame, the Teams SDK must be used to control when an authentication/login screen is being displayed. From experience, we have found that some SPA apps have had to go through a decent amount of re-configuration to hand off this process, when a protected page/resource is accessed. I am making you aware of this, not to put you off developing a Teams app, but to let you know that some work is likely going to be required to make sure the authentication/access process, into your app, is as smooth as possible. Once you've got the hang of working with the Teams SDK, I am confident you will see that it adds immense value, and results in a great user experience when they are accessing your app. Finally, we work a lot with partners, when building the SSO with them, around Identity matching. This is likely something most partners will need to do, to ensure that when customers access your app in Teams, and Azure AD SSO is performed via the Teams SDK, they gain access to the same resources, if they logged into your app, using a client and a username/password that is unique for your application. This sample app does not demonstrate how to do identity matching, but Scott has built a sample, available on his GitHub,

that does demonstrate how this **can** work in a production environment – [scottperham/AADLab \(github.com\)](https://github.com/scottperham/AADLab)

Additionally, while this is a .NET/C# sample app, in the real world, your app can use whatever code/framework you want, and we have SDKs that work across multiple languages, and samples for you to review. In the UK we have worked with partners who have built Teams apps using; React, node/JS, .NET, Ruby on Rails, Angular, JAVA, Python, Blazor, PROGRESS, and many others! See the following for a list of some of the Teams apps samples that are available - [OfficeDev/Microsoft-Teams-Samples: Welcome to the Microsoft Teams samples repository. Here you will find task-focused samples in C#, JavaScript and TypeScript to help you get started with the Microsoft Teams App! \(github.com\)](https://github.com/OfficeDev/Microsoft-Teams-Samples)

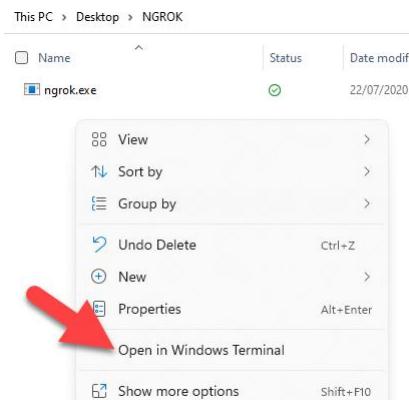
Finally, we want to say, good luck with the labs, and we hope you enjoy and learn a lot from them! It has taken a substantial amount of time for us to rework this app, get it updated; using the latest version of .NET, Teams JS SDK & Bot Framework SDK, and of course, to write this lab guide (which probably was the most time-consuming task out of it all!). Please do let us know if you have any feedback, by either logging an issue on the GitHub repo, or getting in touch with us, via Teams or email, at jack.lewis@microsoft.com & scperham@microsoft.com. Thank you for taking the time to work through these labs!

Lab Steps

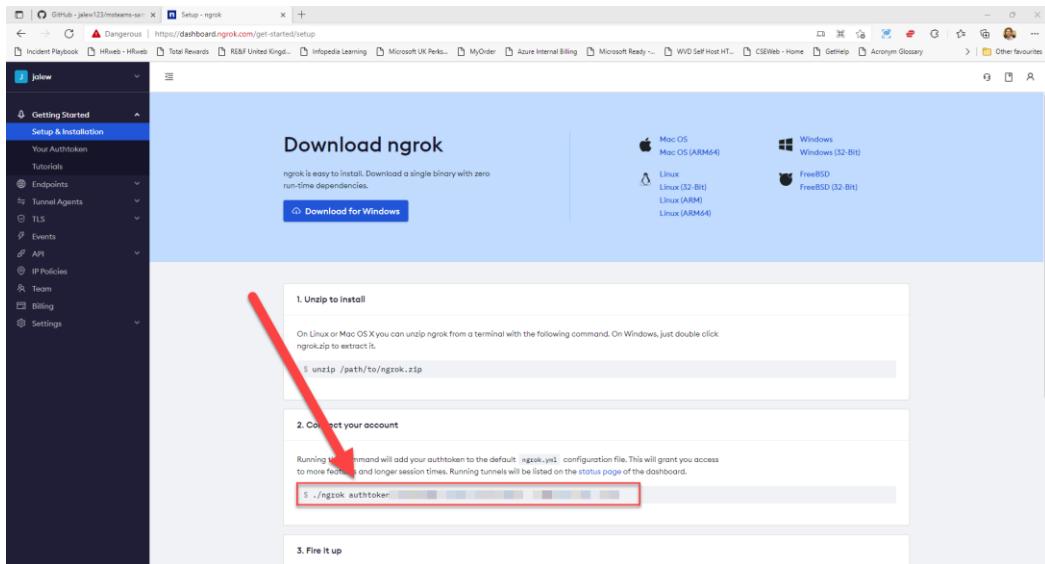
- 1) Setup Talent Management App

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. If you signed up for an NGROK free trial, you'll need to submit your access token to NGROK, to authenticate you with their service. To do this, enter the following in the Command Prompt/Windows Terminal session: `\ngrok authtoken ENTERAUTHTOKENHERE`. You can get your auth token from the NGROK account dashboard, found here <https://dashboard.ngrok.com/get-started/setup>



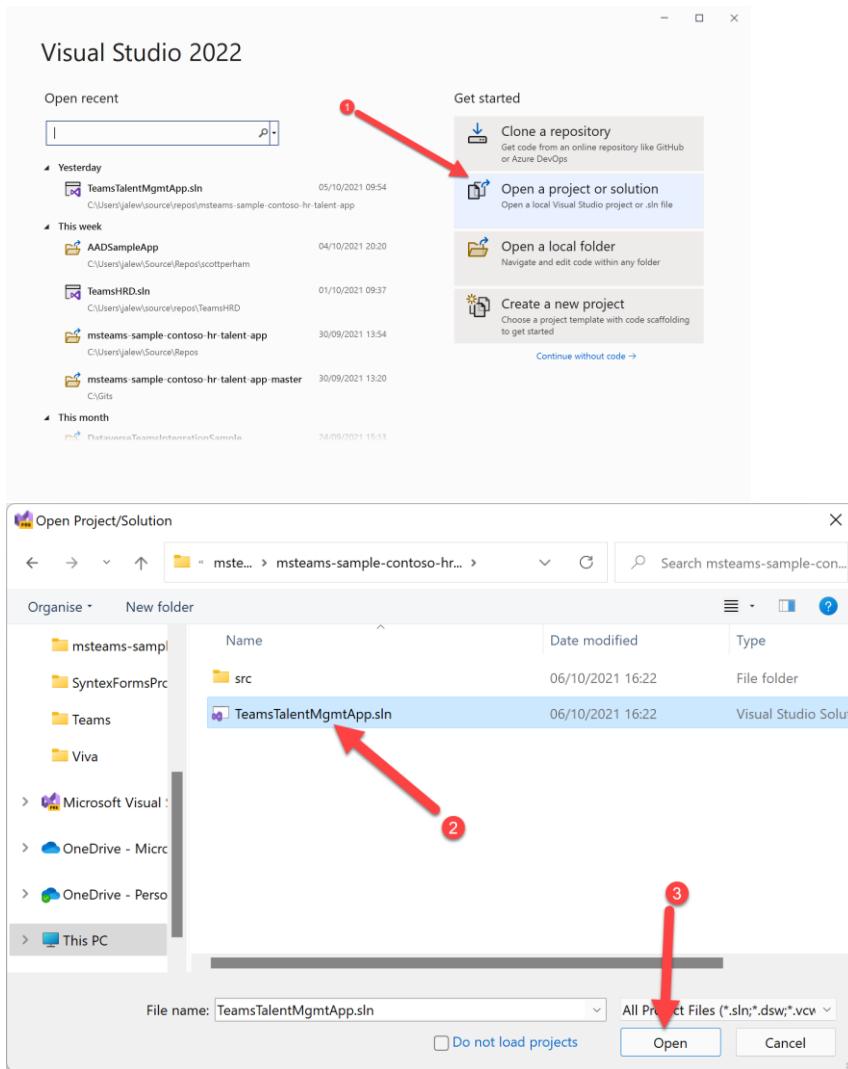
3. Enter the following in the Command Prompt/Windows Terminal session: `.\ngrok http 5000`
 This will open an NGROK tunnel and forward traffic destined for the forwarding URLs to localhost:5000 (where you will run the Talent Management app later in this lab).
Make note of the HTTPS forwarding URL. You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

```

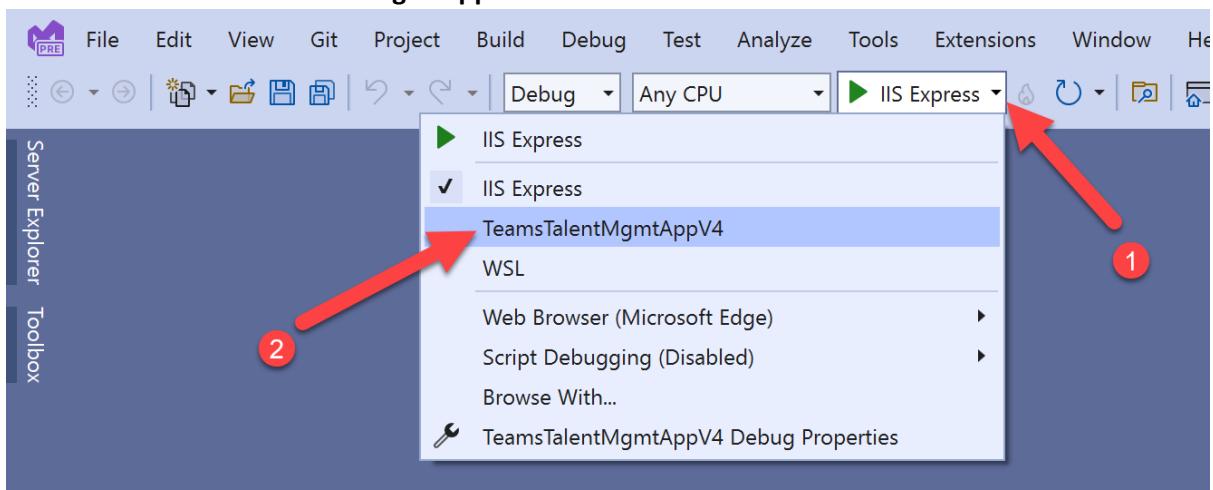
ngrok by @inconshreveable
Session Status          online (Plan: Pro)
Account
Update
Version
Region
Web Interface
Forwarding
Forwarding
Connections
      ttl     opn      rtt1     rt5      p50      p98
      0       0       0.00     0.00     0.00     0.00

```

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



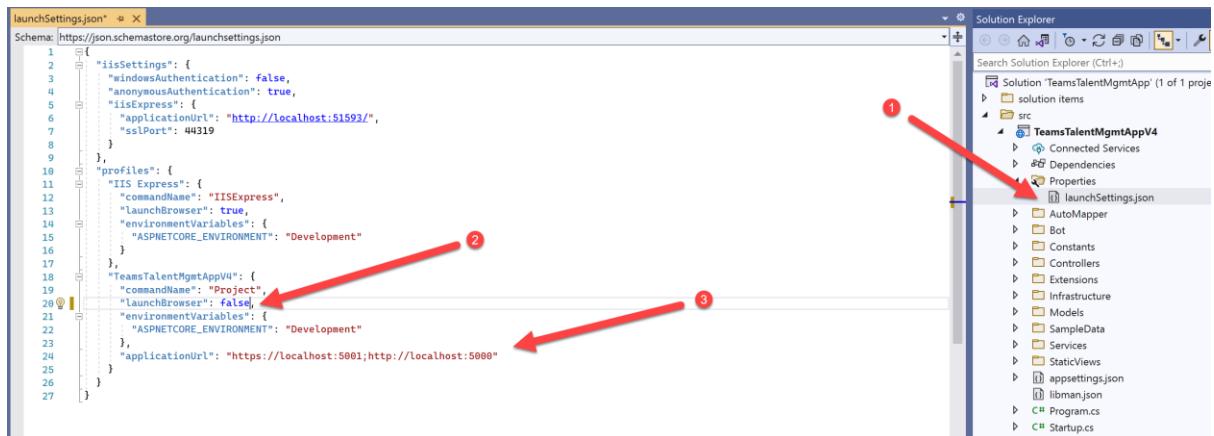
- When Visual Studio has loaded the solution and associated projects, we need to set the debug properties correctly. In the Visual Studio toolbar, select the **IIS Express dropdown menu** and select **TeamsTalentMgmtAppV6**.



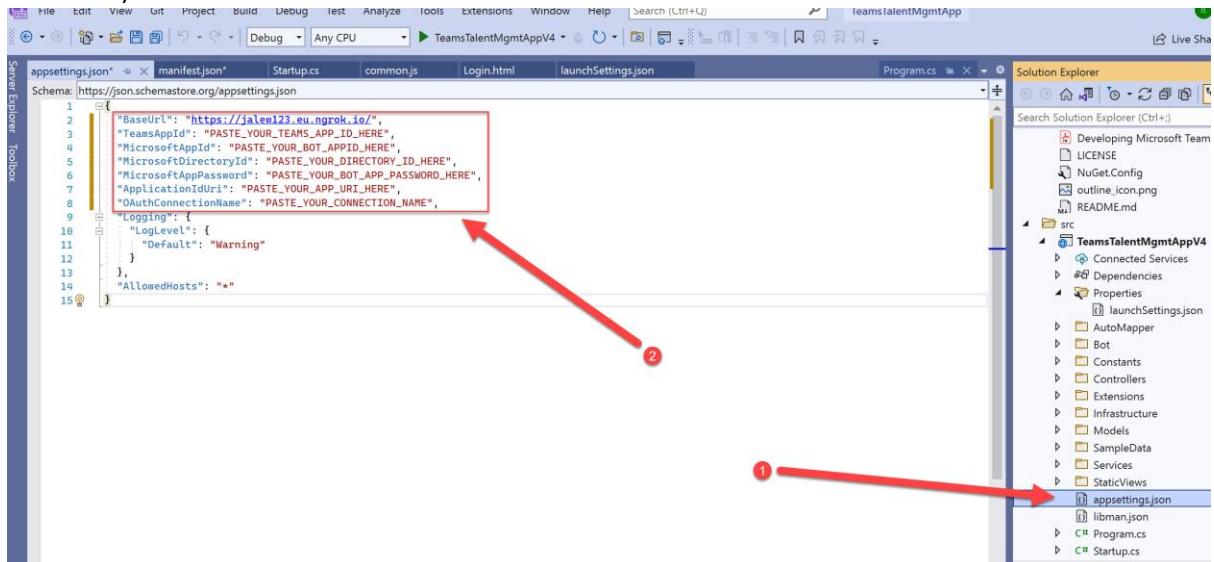
- On the right-side of Visual Studio, within Solution Explorer, expand **src**, **TeamsTalentMgmtAppV6**, **Properties** and select **launchSettings.json**. This file contains the properties that are used when we run (debug) the Talent Management App locally, in Visual Studio. Change the value of **launchBrowser** (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.



7. 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.



OPTIONAL: If you have cloned this application, from a forked GitHub repo, and want to ensure that your App Settings file is not synchronised back to GitHub when changes are pushed from Visual Studio to your GitHub repo, then create a file and name it **development.appsettings.json** and copy the contents of **appsettings.json** into the **development** file. You will then notice that a red icon (similar to a stop sign here in the UK!) appears next to the file, this is telling you that the file will be 'ignored' when you use GitHub. If you populate **development.appsettings.json** with your application's values, it will not synchronise back to GitHub, therefore keeping the values that you want to protect (such as the app secret) stored locally on your machine only:

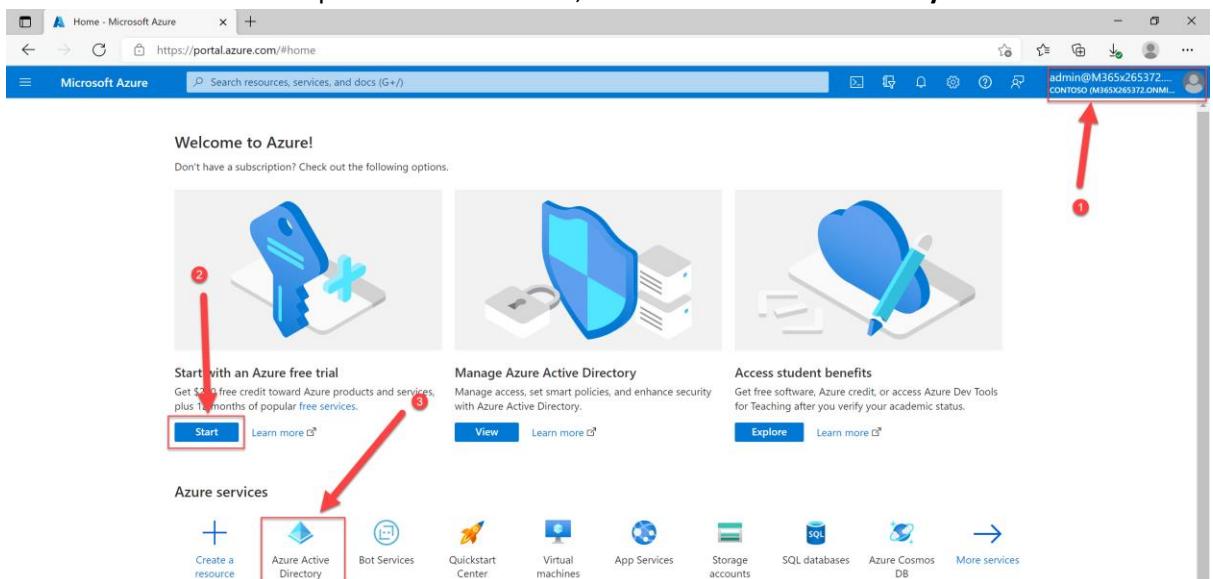
```

1  {
2    "BaseUrl": "https://jalew123.eu.ngrok.io/",
3    "TeamAppId": "",
4    "MicrosoftAppId": "",
5    "MicrosoftDirectoryId": "",
6    "MicrosoftAppPassword": "",
7    "ApplicationIdUri": "api:",
8    "OAuthConnectionName": "",
9    "Logging": {
10      "LogLevel": {
11        "Default": "Warning"
12      }
13    },
14    "AllowedHosts": "*"
15  }

```

Please Note: The ServiceUrl item, in the appsettings.json file, is pre-populated with the value "https://smba.trafficmanager.net/uk" – this should work for global implementations of this Teams app, although you can change the path from uk, to one of the following, if you'd prefer: amer, apac, in, emea or jp.

- 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.



- Select App Registrations and then click New Registration

The screenshot shows the Azure Active Directory - App registrations page. On the left, there's a sidebar with various management options like Overview, Preview features, and Diagnose and solve problems. Under the 'Manage' section, 'App registrations' is highlighted with a red circle labeled '1'. At the top right, there's a 'New registration' button with a red circle labeled '2'.

10. 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**.

The screenshot shows the 'Register an application' form. The 'Name' field (1) contains 'Talent-Management-App'. The 'Supported account types' dropdown (2) is set to 'Accounts in any organizational directory (Any Azure AD directory - Multitenant)'. The 'Single-page application (SPA)' dropdown (3) is selected. The 'Redirect URI (optional)' field (4) contains 'https://jalew123.eu.ngrok.io/StaticViews/LoginResult.html'. The 'Register' button (5) is highlighted with a red circle.

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

The screenshot shows the Microsoft Azure portal interface for managing app registrations. The left sidebar includes links for Home, Contoso, Overview, Quickstart, Integration assistant, Manage, Branding, Authentication, Certificates & secrets, Token configuration, and API permissions. The main content area is titled "Talent-Management-App". It features a search bar, a toolbar with Delete, Endpoints, and Preview features, and a collapsible "Essentials" section. The "Application (client) ID" field is highlighted with a red box and a red arrow pointing to a "Copy to clipboard" button. Below it, other fields show "Object ID" and "Directory (tenant) ID". A note at the bottom states: "Starting June 30th, 2020 we will no longer add any new features to Azure Active Directory Authentication Library. It will be upgraded to Microsoft Authentication Library (MSAL) and Microsoft Graph. Learn more".

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

The screenshot shows the Azure portal interface for managing app registrations. On the left, there's a sidebar with navigation links like Overview, Quickstart, Integration assistant, Manage, Branding, Authentication, Certificates & secrets, Token configuration, API permissions, and others. The main content area is titled 'Talent-Management-App'. It has tabs for Overview, Delete, Endpoints, and Preview features. Under the 'Manage' section, there's a list of items: Branding, Authentication, Certificates & secrets, Token configuration, and API permissions. The 'API permissions' item is currently selected. In the main content area, there's a section titled 'Essentials' with fields for Display name (Talent-Management-App), Application (client) ID (5652156c-2020-42c4-b6f2-b942a45c526e), Object ID (4f390244-f81b-45bf-93ff-f2f2d2b), and Directory (tenant) ID (d06ea5c6-1047-45d0-8ea9-7de7d40e3c58). A tooltip for the 'Copy to clipboard' button is visible over the Object ID. Another tooltip for the same button is visible over the Directory (tenant) ID. Below these fields, it says 'Supported account types : Multiple organizations'. At the bottom, there are two informational messages: one about the new App registrations experience and another about the deprecation of the Active Directory Authentication Library starting June 30, 2020.

13. 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.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with various management options like Overview, Quickstart, Integration assistant, Branding, Authentication (which is highlighted with a red box and numbered 1), Certificates & secrets, Token configuration, API permissions, Expose an API, App roles, Owners, Roles and administrators | Preview, and Manifest. The main content area is titled 'Talent-Management-App | Authentication'. It shows 'Platform configurations' with a note about redirect URIs and a 'Configure Web' dialog open. The 'Configure Web' dialog has tabs for 'All platforms', 'Quickstart', and 'Docs'. Under 'Redirect URIs', a URL 'https://token.botframework.com/auth/web/redirect' is listed. A red box highlights this URL, and a red arrow labeled 3 points from it to the 'Configure' button at the bottom right of the dialog. Another red arrow labeled 4 points from the 'Configure' button in the dialog to the 'Configure' button at the bottom right of the main configuration page. The main configuration page also has a 'Configure' button at the bottom right.

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

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with various management options like Overview, Quickstart, Integration assistant, Branding, Authentication (which is highlighted with a red box and numbered 1), Certificates & secrets (which is highlighted with a red box and numbered 2), Token configuration, API permissions, Expose an API, App roles, Owners, Roles and administrators | Preview, and Manifest. The main content area is titled 'Talent-Management-App | Certificates & secrets'. It shows 'Certificates' and 'Client secrets'. Under 'Client secrets', there's a table with columns: Description, Expires, Value, and Secret ID. A red box highlights the '+ New client secret' button. A red arrow labeled 3 points from the 'Add' button at the bottom right of the 'Add a client secret' dialog to the 'Add' button at the bottom right of the main configuration page.

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

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

+ New client secret

Description	Expires	Value	Copy to clipboard	Set ID
Password uploaded on Thu Oct 07 2021	10/7/2022	Jj.7Q~YeYOr-R2q118u1cKR2GanYhKY_41B...		ab5671d6-7958-4265-b4ba-1a24da64f04a

```
appsettings.json manifest.json* Startup.cs common.js Login.html launchSettings.json
Schema: https://json.schemastore.org/appsettings.json
1 "BaseUrl": "https://jalew123.eu.ngrok.io/",
2 "TeamsAppId": "████████████████████████████████████",
3 "MicrosoftAppId": "5652156c-2020-42c4-b6f2-b942a45c526e",
4 "MicrosoftAppDirectoryID": "d06ea5c6-1047-45d0-8ea9-7de7d40e3c58",
5 "MicrosoftAppPassword": "Jj.7Q~YeYOr-R2q118u1cKR2GanYhKY_41Bd1",
6 "ApplicationIdUri": "████████████████████████████████████",
7 "OAuthConnectionName": "████████████████████████████████████"
8
```

16. Select API permission and click Add a permission.

Home > Contoso > Talent-Management-App

Talent-Management-App | API permissions

Search (Ctrl+ /) Refresh Got feedback?

Overview Quickstart Integration assistant

Starting November 9th, 2020 end users will no longer be able to grant consent to newly registered multitenant apps without verified publishers. Add MPN ID to verify publisher

The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the value all the permissions the application needs. Learn more about permissions and consent

Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. Learn more about permissions and consent

+ Add a permission Grant admin consent for Contoso

API / Permissions name	Type	Description	Admin consent req...	Status
Microsoft Graph (1)	User.Read	Delegated Sign in and read user profile	No	...

To view and manage permissions and user consent, try Enterprise applications.

17. Select Microsoft Graph Request API permissions

Select an API

Microsoft APIs APIs my organization uses My APIs

Commonly used Microsoft APIs

Microsoft Graph

Take advantage of the tremendous amount of data in Office 365, Enterprise Mobility + Security, and Windows 10. Access Azure AD, Excel, Intune, Outlook/Exchange, OneDrive, OneNote, SharePoint, Planner, and more through a single endpoint.

Azure Rights Management Services
Allow validated users to read and write protected content

Azure Service Management
Programmatic access to much of the functionality available through the Azure portal

Data Export Service for Microsoft Dynamics 365
Export data from Microsoft Dynamics CRM organization to an external destination

Dynamics 365 Business Central
Programmatic access to data and functionality in Dynamics 365 Business Central

Dynamics CRM
Access the capabilities of CRM business software and ERP systems

Flow Service
Embed flow templates and manage flows

18. We'll start with the Delegated Permissions first, then we'll do the App Permissions. Select **Delegated permissions**, select **email, offline_access, openid & profile**.

Request API permissions

Microsoft Azure Search resources, services, and docs (G+/)

Delegated permissions

Your application needs to access the API as the signed-in user.

Application permissions

Your application runs as a background service or daemon without a signed-in user.

Select permissions

Start typing a permission to filter these results

The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the value in your organization, or in organizations where this app will be used. [Learn more](#)

Permission	Admin consent required
email ⓘ View users' email address	No
offline_access ⓘ Maintain access to data you have given it access to	No
openid ⓘ Sign users in	No
profile ⓘ View users' basic profile	No

Add permissions Discard

19. Then search for and select **Team.ReadBasic.All, Channel.ReadBasic.All, ChannelMessage.Send, ChannelMessage.Read.All, Chat.ReadBasic, ChatMessage.Read & ChatMessage.Send**. Select **Add permissions** once all these scopes have been selected.

Request API permissions

[All APIs](#)

 Microsoft Graph
<https://graph.microsoft.com/> [Docs](#)

What type of permissions does your application require?

Delegated permissions
Your application needs to access the API as the signed-in user.

Application permissions
Your application runs as a background service or daemon without a signed-in user.

Select permissions

expand all

i The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the value in your organization, or in organizations where this app will be used. [Learn more](#)

Permission	Admin consent required
▽ Team (1) <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Team.ReadBasic.All ⓘ Read the names and descriptions of teams 	No

Add permissions Discard

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

[Microsoft Azure](#)

Home > Contoso > Contoso Web App

 [Contoso Web App | API permissions](#)

i You are editing permission(s) to your application, users will have to consent even if they've already done so previously.
will be used. [Learn more](#)

Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. [Learn more about permissions and consent](#)

[+ Add a permission](#) [Grant admin consent for Contoso](#)

API / Permissions name	Type	Description	Admin consent requ...	Status
▽ Microsoft Graph (12) <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Channel.ReadBasic.All Delegated Read the names and descriptions of channels <input checked="" type="checkbox"/> ChannelMessage.Read.All Delegated Read user chat messages <input checked="" type="checkbox"/> ChannelMessage.Send Delegated Send channel messages <input checked="" type="checkbox"/> Chat.ReadBasic Delegated Read names and members of user chat threads <input checked="" type="checkbox"/> ChatMessage.Read Delegated Read user chat messages <input checked="" type="checkbox"/> ChatMessage.Send Delegated Send user chat messages <input checked="" type="checkbox"/> email Delegated View users' email address <input checked="" type="checkbox"/> offline_access Delegated Maintain access to data you have given it access to <input checked="" type="checkbox"/> openid Delegated Sign users in <input checked="" type="checkbox"/> profile Delegated View users' basic profile <input checked="" type="checkbox"/> Team.ReadBasic.All Delegated Read the names and descriptions of teams <input checked="" type="checkbox"/> User.Read Delegated Sign in and read user profile 			No	Not granted for Contoso

To view and manage permissions and user consent, try [Enterprise applications](#).

21. Now that the delegated permissions have been added, we can now add our Application Permissions. Click **Add a permission** again

Home > Contoso > Contoso Web App

Contoso Web App | API permissions

Search (Ctrl+ /) Refresh Got feedback?

Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the permissions the application needs. [Learn more about permissions and consent](#)

+ Add a permission Grant admin consent for Contoso

API / Permissions name	Type	Description
Microsoft Graph (12)		
Channel.ReadBasic.All	Delegated	Read the names and descriptions of channels
ChannelMessage.Read.All	Delegated	Read user channel messages
ChannelMessage.Send	Delegated	Send channel messages
Chat.ReadBasic	Delegated	Read names and members of user chat threads
ChatMessage.Read	Delegated	Read user chat messages
ChatMessage.Send	Delegated	Send user chat messages
email	Delegated	View users' email address
offline_access	Delegated	Maintain access to data you have given it access to
openid	Delegated	Sign users in
profile	Delegated	View users' basic profile
Team.ReadBasic.All	Delegated	Read the names and descriptions of teams
User.Read	Delegated	Sign in and read user profile

To view and manage permissions and user consent, try [Enterprise applications](#).

22. Select Microsoft Graph
Request API permissions

Select an API

Microsoft APIs APIs my organization uses My APIs

Commonly used Microsoft APIs

Microsoft Graph
Take advantage of the tremendous amount of data in Office 365, Enterprise Mobility + Security, and Windows 10. Access Azure AD, Excel, Intune, Outlook/Exchange, OneDrive, OneNote, SharePoint, Planner, and more through a single endpoint.

Azure Rights Management Services Allow validated users to read and write protected content	Azure Service Management Programmatic access to much of the functionality available through the Azure portal	Data Export Service for Microsoft Dynamics 365 Export data from Microsoft Dynamics CRM organization to an external destination
Dynamics 365 Business Central Programmatic access to data and functionality in Dynamics 365 Business Central	Dynamics CRM Access the capabilities of CRM business software and ERP systems	Flow Service Embed flow templates and manage flows

23. Select **Application permission**, then search for and select **AppCatalog.Read.All**, **TeamsAppInstallation.ReadWriteSelfForUser.All** & **User.Read.All**. Select **Add permissions** once all these scopes have been selected.

Request API permissions

Microsoft Graph

What type of permissions does your application require?

Delegated permissions

Your application needs to access the API as the signed-in user.

Application permissions

Your application runs as a background service or daemon without a signed-in user.

Select permissions

User.Read.All

Permission

Admin consent required

User (1)

User.Read.All ⓘ
Read all users' full profiles

Yes

Add permissions Discard

24. If successful, your API permissions should now look like the below, and include both the delegated and application permission type scopes:

Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. [Learn more about permissions and consent](#)

API / Permissions name	Type	Description	Admin consent req...	Status
Microsoft Graph (15)				...
AppCatalog.Read.All	Application	Read all app catalogs	Yes	⚠️ Not granted for Contoso
Channel.ReadBasic.All	Delegated	Read the names and descriptions of channels	No	...
ChannelMessage.Read.All	Delegated	Read user channel messages	Yes	⚠️ Not granted for Contoso
ChannelMessage.Send	Delegated	Send channel messages	No	...
Chat.ReadBasic	Delegated	Read names and members of user chat threads	No	...
ChatMessage.Read	Delegated	Read user chat messages	No	...
ChatMessage.Send	Delegated	Send user chat messages	No	...
email	Delegated	View users' email address	No	...
offline_access	Delegated	Maintain access to data you have given it access to	No	...
openid	Delegated	Sign users in	No	...
profile	Delegated	View users' basic profile	No	...
Team.ReadBasic.All	Delegated	Read the names and descriptions of teams	No	...
TeamsAppInstallation.ReadWrite	Application	Allow the app to manage itself for all users	Yes	⚠️ Not granted for Contoso
User.Read	Delegated	Sign in and read user profile	No	...
User.Read.All	Application	Read all users' full profiles	Yes	⚠️ Not granted for Contoso

To view and manage permissions and user consent, try [Enterprise applications](#).

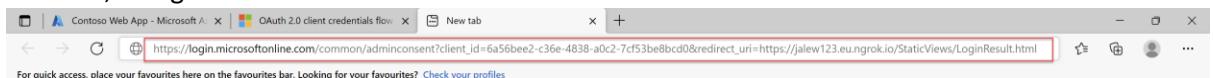
25. We now need to consent to these scopes in our application, as the Talent Management app utilises some of these scopes for proactive messaging, whilst the remaining scopes are used in section 7 of this lab guide, when you start working with Postman to interact with the Microsoft Graph API. To do this, we need to generate an Admin Consent URL, that will ask for all of the scopes you have just selected in the App Registration. In order to build your admin consent URL you will need to take the following template, and replace the values in **BOLD** with the values that are in the appsettings.json file in Visual Studio:

https://login.microsoftonline.com/common/adminconsent?client_id=MICROSOFTAPPID&redirect_uri=BASEURLStaticViews/LoginResult.html

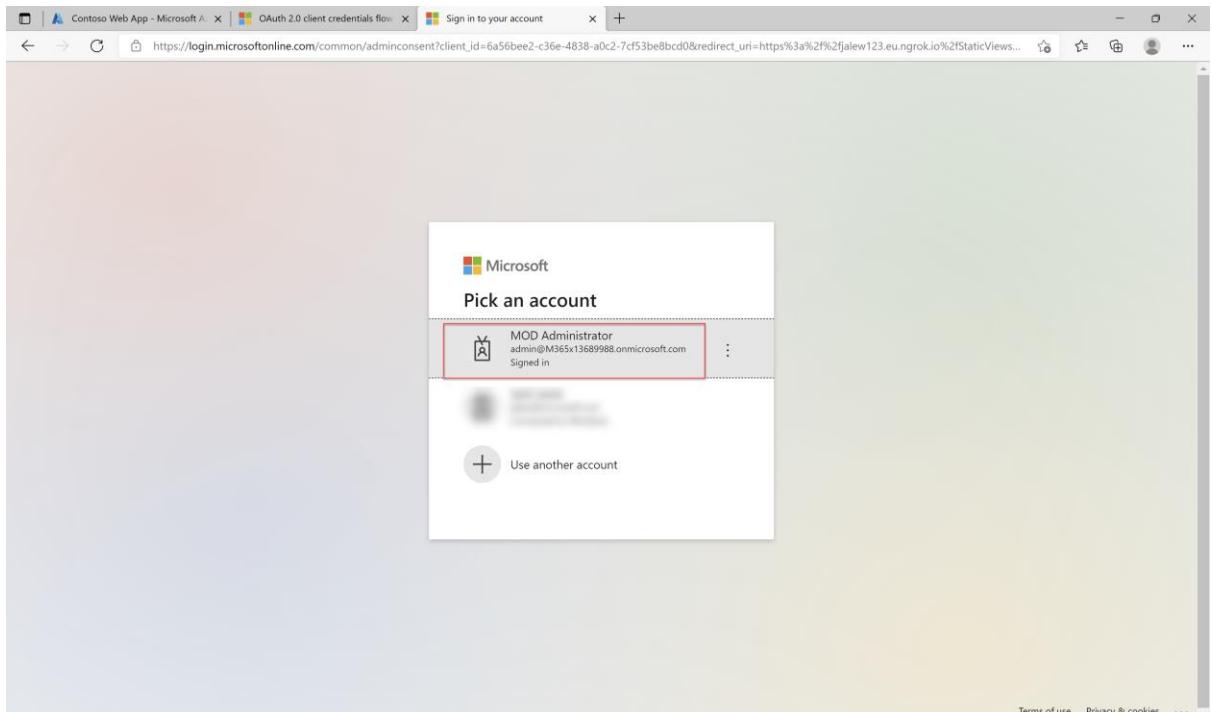
For example, my admin consent URL looks like the following:

https://login.microsoftonline.com/common/adminconsent?client_id=6a56bee2-c36e-4838-a0c2-7cf53be8bcd0&redirect_uri=https://jalew123.eu.ngrok.io/StaticViews/LoginResult.html

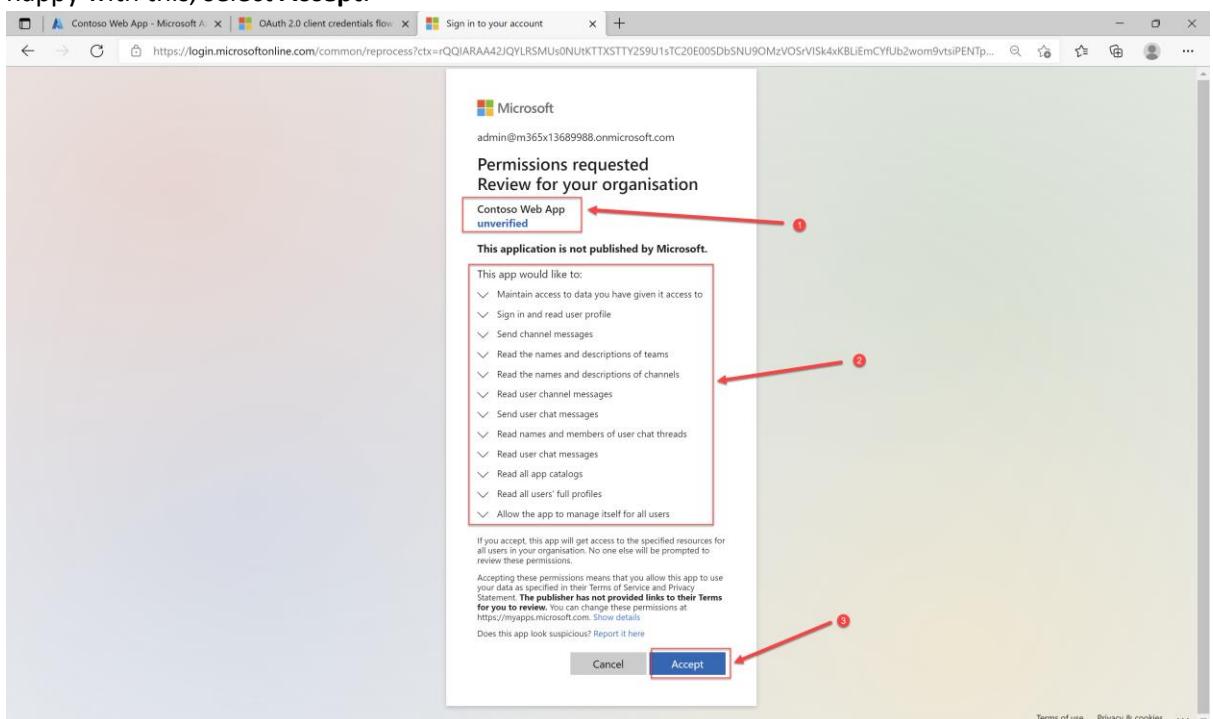
26. Once you have the admin consent URL created, open up a new tab in your preferred web browser, and go to that website.



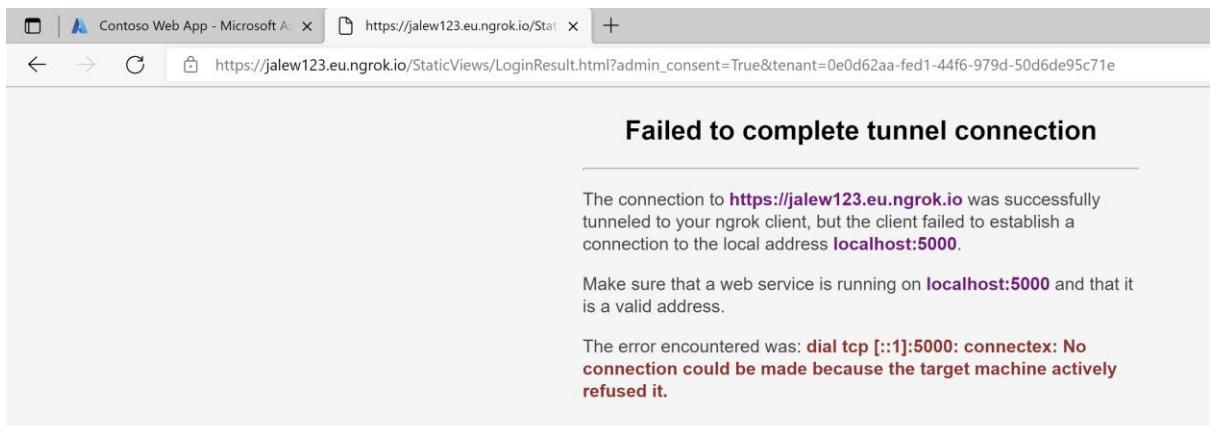
27. You will be prompted to authenticate, make sure you **sign in as a user, in your test/dev tenant who is a Global Administrator**.



28. Review the information on the consent screen. Firstly, **make sure that your App Registration's name is correct**, and then **review the scopes you are being asked to consent to**, they should match the scopes that you selected in your app registration, a few steps previously and will include both delegated and app permission type scopes. Once you are happy with this, select **Accept**.



29. You will then be redirected back to your application, but as the app isn't running, this will fail and is expected. In a real world scenario, you would be redirected back to a web-page, that is able to process the consent being granted (or not granted) and then do something with that consent (such as start to install apps, bots, send chat messages, etc...).



30. Now that we have consent in place, go back to the Azure Portal, and back into your App Registration. Select **Expose an API** and **Set** the Application ID URI.

Home > Contoso > Talent-Management-App

Talent-Management-App | Expose an API

Search (Ctrl+ /) < Got feedback?

Application ID URI Set (2)

Scopes defined by this API

Define custom scopes to restrict access to data and functionality protected by the API. An application that requires access to parts of this API can request that a user or admin consent to one or more of these.

Adding a scope here creates only delegated permissions. If you are looking to create application-only scopes, use 'App roles' and define app roles assignable to application type. [Go to App roles.](#)

+ Add a scope

Scopes	Who can consent	Admin consent display ...	User consent display na...	State
No scopes have been defined				

Authorized client applications (1)

Authorizing a client application indicates that this API trusts the application and users should not be asked to consent when the client calls this API.

+ Add a client application

Client Id	Scopes
No client applications have been authorized	

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

Set the App ID URI

Application ID URI

Save **Discard**

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

Home > TalentManagementBot > Talent-Management-App

Talent-Management-App | Expose an API

Search (Ctrl+ /) < Got feedback?

Application ID URI Copy to clipboard

Scopes defined by this API

Define custom scopes to restrict access to data and functionality protected by the API. An application that requires access to parts of this API can request that a user or admin consent to one or more of these.

Adding a scope here creates only delegated permissions. If you are looking to create application-only scopes, use 'App roles' and define app roles assignable to application type. [Go to App roles.](#)

```

1  "BaseUrl": "https://jalew123.eu.ngrok.io/",
2  "TeamsAppId": "████████████████████████████████████████",
3  "MicrosoftAppId": "5652156c-2020-42c4-b6f2-b942a45c526e",
4  "MicrosoftAppDirectoryID": "d06ea5c6-1047-45d0-8ea9-7de7d40e3c58",
5  "MicrosoftAppPassword": "Jj.7Q-YeYOr-R2q118u1CKR2GanYhkY_41Bd1",
6  "ApplicationIdUri": "api://jalew123.eu.ngrok.io/botid-5652156c-2020-42c4-b6f2-b942a45c526e",
7  "OAuthConnectionName": "████"

```

33. Select Add a scope

Scopes defined by this API

Define custom scopes to restrict access to data and functionality protected by the API. An application that requires access to parts of this API can request that a user or admin consent to one or more of these.

Adding a scope here creates only delegated permissions. If you are looking to create application-only scopes, use 'App roles' and define app roles assignable to application type. [Go to App roles.](#)

+ Add a scope

Scopes	Who can consent	Admin consent display ...	User consent display na...	State
No scopes have been defined				

Authorized client applications

Authorizing a client application indicates that this API trusts the application and users should not be asked to consent when the client calls this API.

+ Add a client application

Client Id	Scopes
No client applications have been authorized	

34. 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**

Add a scope

Scope name * ①
access_as_user

Who can consent? ②
Admins and users

Admin consent display name * ③
access_as_user

Admin consent display value * ④
access_as_user

User consent display name ⑤
access_as_user

User consent description ⑥
access_as_user

State ⑦
Enabled

35. 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**.

Home > Contoso > Talent-Management-App

Talent-Management-App | Expose an API

Search (Ctrl+ /) Got feedback?

Overview Quickstart Integration assistant

Application ID URI: api://jalew123.eu.ngrok.io/botid-5652156c-2020-42c4-b6f2-b942a45c526e

Scopes defined by this API

Define custom scopes to restrict access to data and functionality protected by the API. An application that requires access to parts of this API can request that a user or admin consent to one or more of these.

Adding a scope here creates only delegated permissions. If you are looking to create application-only scopes, use 'App roles' and define app roles assignable to application type. [Go to App roles](#).

+ Add a scope

Scopes	Who can consent	Admin consent display ...	User consent display na...	State
api://jalew123.eu.ngrok.io/botid-5652156c-2020-42c4... 	Admins and users	access_as_user	access_as_user	Enabled

Authorized client applications

Authorizing a client application indicates that this API trusts the application and users should not be asked to consent when the client calls this API.

+ Add a client application

Client Id

No client applications have been authorized

36. 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, below the Authorized scopes section, 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, below the Authorized scopes section, and click **Add Application**.

Add a client application

× Add a client application

×

The screenshot shows two separate configuration dialogs for client applications. Both dialogs have a 'Client ID' field (1) containing a unique GUID and an 'Authorized scopes' field (2) containing a specific URL with a checked checkbox. Red boxes and numbers highlight the 'Client ID' fields (1, 4) and the 'Authorized scopes' fields (2, 5) across both dialogs.

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!

37. 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**.

The editor below allows you to update this application by directly modifying its JSON representation. For more details, see the documentation.

```

1  {
2      "id": "4f390244-f81b-45bf-93ff-f2f2d2bbb5f9",
3      "acceptMappedClaims": null,
4      "accessTokenAcceptedVersion": 2, 2
5      "addIns": [],
6      "allowPublicClient": null,
7      "appId": "5652156c-2020-42c4-b6f2-b942a45c526e",
8      "appRoles": [],
9      "oauth2AllowUrlPathMatching": false,
10     "createdDateTime": "2021-10-07T11:29:03Z",
11     "certification": null,
12     "disabledByMicrosoftStatus": null,
13     "groupMembershipClaims": null,
14     "identifierUris": [
15         "api://jalew123.eu.ngrok.io/botid-5652156c-2020-42c4-b6f2-b942a45c526e"
16     ],
17     "informationalUrls": {
18         "termsOfService": null,
19         "support": null,
20         "privacy": null,
21         "marketing": null
22     },
23     "keyCredentials": [],
24     "knownClientApplications": [],
25     "logoUrl": null,
26     "logoutUrl": null,
27     "tags": [
28         "Talent Management App"
29     ]
30 }

```

38. 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**.

Services

- Applied AI services** 2
- Cognitive Search
- Form recognizers
- Immersive readers
- Metrics advisors
- Bonsai
- Mailjet Email Service
- Maintenance Configurations
- Availability sets
- Test Emails

Resources

No results were found.

Documentation

Azure Applied AI Services documentation | Microsoft Docs
What are Azure Applied AI Services? - Azure Applied AI ...
Azure Applied AI Services frequently asked questions ...
Why Azure Applied AI Services? - Azure Applied AI Services ...

Resource Groups

No results were found.

Didn't find what you were looking for?
Try searching in Activity Log
Try searching in Azure Active Directory

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

https://portal.azure.com/#blade/Microsoft_Azure_ProjectOxford/AppliedAIHub/overview

Applied AI services

Bot services

Metrics advisor

Video analyzer

Bot services

Cognitive search

Form recognizer

Immersive reader

Metrics advisor

Video analyzer

Bot services

+ Create

Metrics advisor

+ Create

Video analyzer

+ Create

Immersive reader

+ Create

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

Home > Applied AI services >

Bot Services

AudioCodes

Virtual Scheduling , ChatBot

Dialogflows AI Solutions Inc

Speech to text

Celebal Technologies Private Limited

Think AI Bot for Connectwise

Think AI Consulting Corporation

Vernacular.ai Intelligent Voice Assistant

Vernacular.ai

devNXT- AI driven smart application development

Wipro Ltd

Rx.Health

Rx.Health

Zammo.ai SaaS

Zammo. Inc.

Audite Cloud

TALENTUM

Mia - Workplace Virtual Assistant

MIHCM

Azure Health Bot

Microsoft

Azure Bot

Microsoft

41. Select **Create**

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with 'Microsoft Azure' and a search bar. Below it, the breadcrumb navigation shows 'Home > Applied AI services > Bot Services > Azure Bot'. The main content area displays a card for 'Azure Bot' by Microsoft, featuring a blue cube icon, the name 'Azure Bot', a 'Add to Favorites' link, a Microsoft logo, a rating of '★ 3.0 (5 Azure ratings)', and a prominent blue 'Create' button which is highlighted with a red rectangular box. Below the card, there are tabs for 'Overview', 'Plans', 'Usage Information + Support', and 'Reviews', with 'Overview' being the active tab. A note at the bottom of the page reads: 'Develop enterprise-grade, conversational AI experiences, while maintaining control of your data. Edit your bot in Bot Framework or LUIS.'

42. 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!

[Home](#) > [Applied AI services](#) > [Bot Services](#) > [Azure Bot](#) >
Switch directories ...

Switch directories

You are currently signed into the 'Contoso (M365x26537.onmicrosoft.com)' directory which does not have any subscriptions. You have other directories you can switch to or you can sign up for a new subscription.

Switch directories
Start free

If you see this, either sign in as a user with an Azure Subscription, or sign up for a Free Azure Trial Subscription

43. 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. The pricing tier can remain for **standard** as the Teams channel doesn't incur any costs (only premium channels are charged). In type of app, select **Multi Tenant**. For the Microsoft App ID, it's important that you select **Use an existing app registration**, enter your **MicrosoftAppID** and **MicrosoftAppPassword** that you generated earlier in this lab. Select **Review + Create**.

Microsoft Azure Search resources, services, and docs (G+ /)

Home > Applied AI services > Bot Services > Azure Bot >

Create an Azure Bot

[Basics](#) [Tags](#) [Review + create](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

1 Bot handle *	TalentManagementBot
2 Subscription *	Microsoft FTE - Visual Studio Enterprise
3 Resource group *	(New) TalentManagementBot Create new
New resource group location	West Europe

Pricing

Select a pricing tier for your Azure Bot resource. You can change your selection later in the Azure portal's resource management. Learn more about available options, or request a pricing quote, by visiting the [Azure Bot Services pricing](#)

4 Pricing tier *	Standard Change plan
------------------	-------------------------

Microsoft App ID

A Microsoft App ID is required to create an Azure Bot resource. If your bot app doesn't need to access resources outside of its home tenant and if your bot app will be hosted on an Azure resource that supports Managed Identities, then choose option User-Assigned Managed Identity so that Azure takes care of managing the App credentials for you. Otherwise, depending on whether your bot will be accessing resources only in its home tenant or not, choose either Single tenant or Multi tenant option respectively.

5 Type of App	Multi Tenant
---------------	--------------

An App ID can be automatically created below or you can manually create your own, then return to input your new App ID and secret in the open fields.

[Manually create App ID](#)

6 Creation type	<input checked="" type="radio"/> Create new Microsoft App ID <input type="radio"/> Use existing app registration
App ID *	██████████
App secret *	*****

[Review + create](#) [< Previous](#) [Next : Tags >](#)

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

Validation Passed

Basics Tags Review + create

Basics

Bot handle	TalentManagementBot
Subscription	Azure subscription 1
New resource group location	UK South
Pricing tier	Free
Microsoft App ID	Use existing app registration
Existing app id	5652156c-2020-42c4-b6f2-b942a45c526e
Existing app password	*****

Create < Previous Next Download a template for automation

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

Home > CreateAzureBot_dx-20211007132314 | Overview ✎ ...

Deployment

Search (Ctrl+ /) < Delete Cancel Redeploy Refresh

Overview

Your deployment is complete

Deployment name: CreateAzureBot_dx-20211007132314
Subscription: Azure subscription 1
Resource group: TalentManagementBot

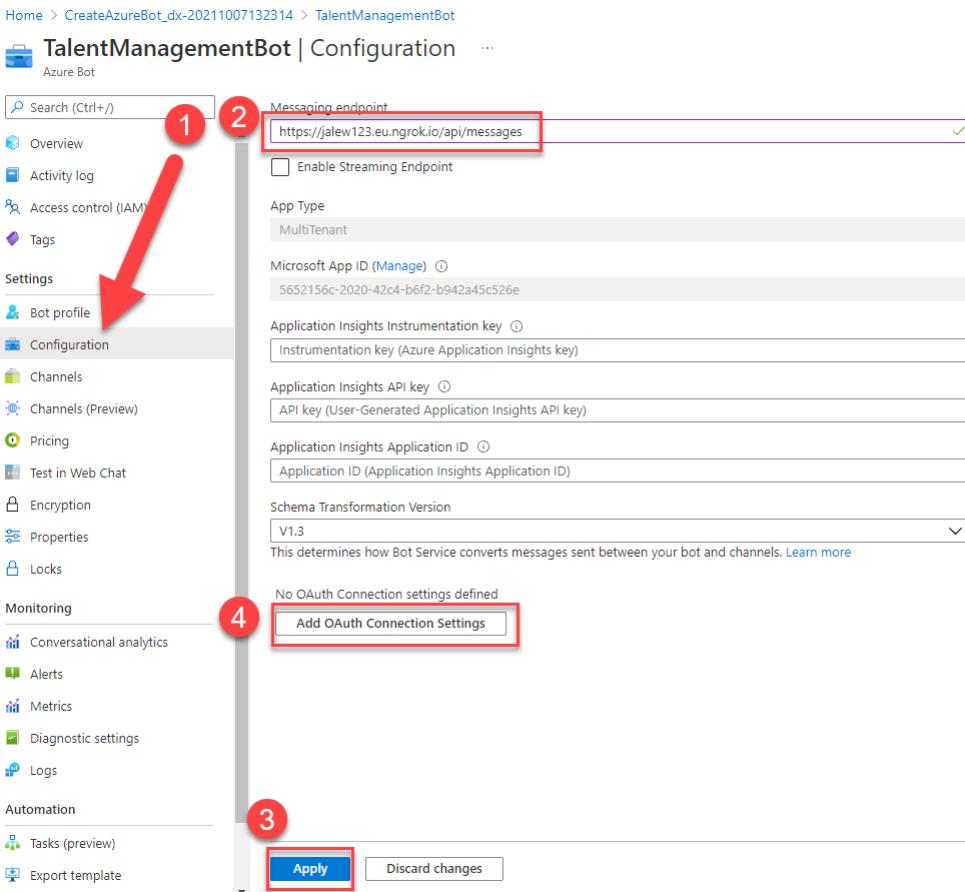
Start time: 10/7/2021, 1:30:11 PM
Correlation ID: 9ad61b69-7796-4d5f-90ce-76bd7ef0448b

Deployment details (Download)

Next steps

Go to resource

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



47. 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.

ot | Configuration ...

Messaging endpoint
https://jalew123.eu.ngrok.io/api/messages

Enable Streaming Endpoint

App Type
MultiTenant

Microsoft App ID (Manage) ⓘ
5652156c-2020-42c4-b6f2-b942a45c526e

Application Insights Instrumentation key ⓘ
Instrumentation key (Azure Application Insights key)

Application Insights API key ⓘ
API key (User-Generated Application Insights API key)

Application Insights Application ID ⓘ
Application ID (Application Insights Application ID)

Schema Transformation Version
V1.3

This determines how Bot Service converts messages sent between your bot and channels. [Learn more](#)

Name _____ Service Provider _____

No results

Add OAuth Connection Settings

New Connection Setting x

1 Name * AAD

2 Service Provider * Azure Active Directory v2

3 Client id * 5652156c-2020-42c4-b6f2-b942a45c526e

4 Client secret *

5 Token Exchange URL api://jalew123.eu.ngrok.io/botid-5652156c-2020-...

6 Tenant ID Organizations

7 Scopes user.read profile openid TeamsAppInstallation.Re...

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

9 Apply Discard changes

8 Save Cancel

```

appsettings.json manifest.json* Startup.cs common.js Login.html launchSettings.json
Schema: https://json.schemastore.org/appsettings.json
1 "BaseUrl": "https://jalew123.eu.ngrok.io/",
2 "TeamsAppId": "REDACTED",
3 "MicrosoftAppId": "5652156c-2020-42c4-b6f2-b942a45c526e",
4 "MicrosoftAppDirectoryID": "d06ea5c6-1047-45d0-8ea9-7de7d40e3c58",
5 "MicrosoftAppPassword": "Jj.7Q-YeYOr-R2q118u1CKR2GanYhkY_41Bd1",
6 "ApplicationIdUri": "api://jalew123.eu.ngrok.io/botid-5652156c-2020-42c4-b6f2-b942a45c526e",
7 "OAuthConnectionName": "AAD"

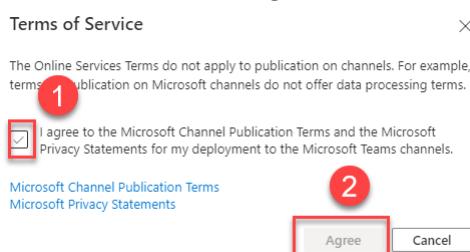
```

49. Now go back into the Azure Portal and configure the channels for the bot to support Teams.

Select **Channels** and select **Microsoft Teams**.

Channel Type	Description
Alexa	Alexa Channel
Direct Line Speech	Direct Line Speech channel
Direct Line	REST API for communicating directly with a bot
Email	Featured: O365 Email Channel
Facebook	Support for Text Messaging via Facebook
GroupMe	Talk to bots via GroupMe groups
Kik	Kik Channel
LINE	Support for Line channel
Microsoft Teams	Microsoft Teams
Omnichannel	Omnichannel Channel

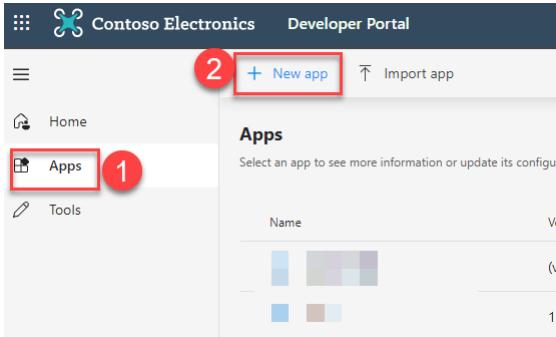
50. Tick the **checkbox** to agree to the terms of service and select **Agree**.



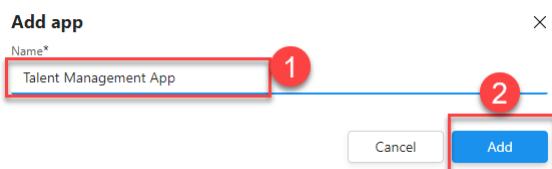
51. Click **Save**.

52. 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

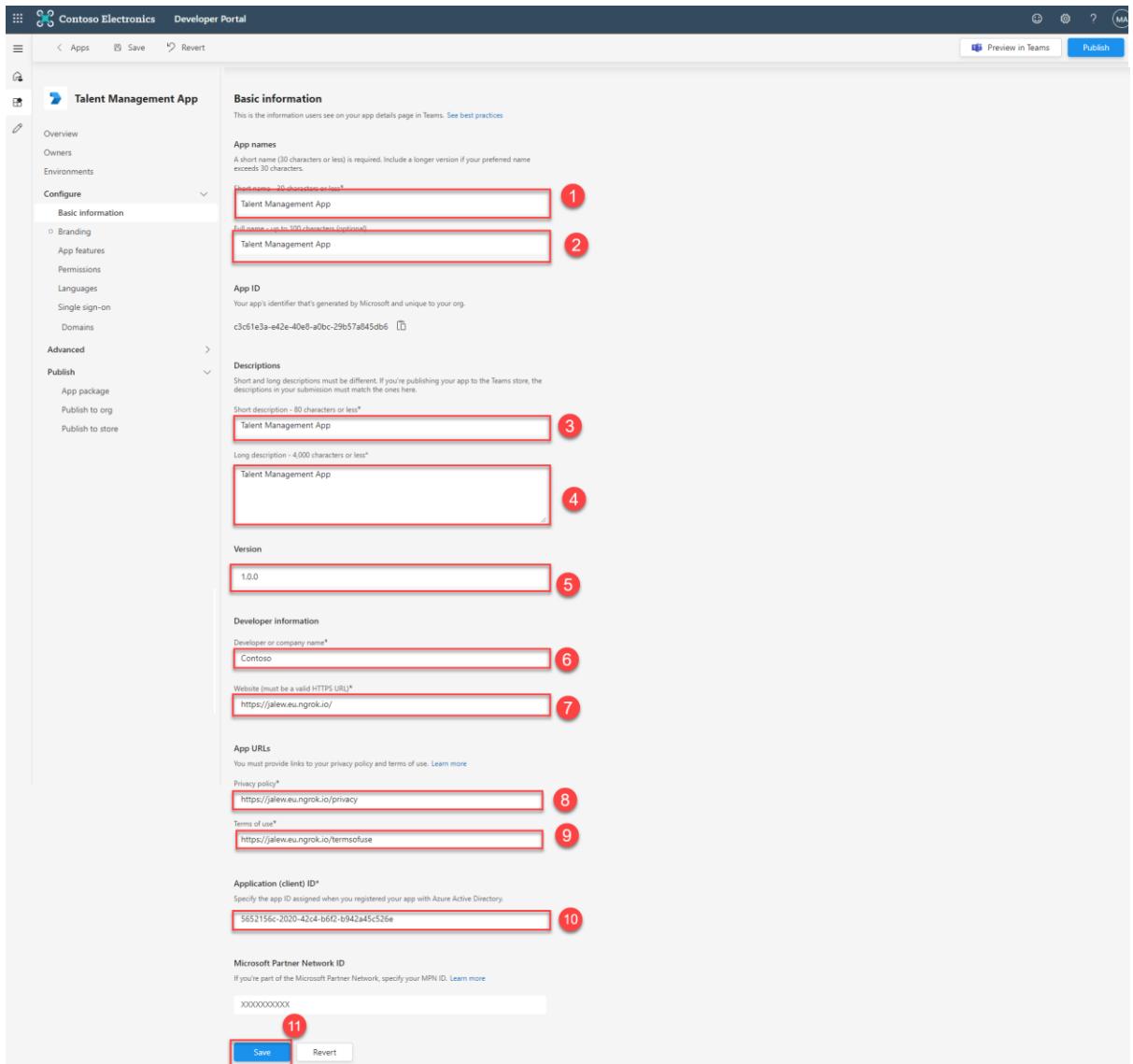
create a Teams App Manifest and populate the final value (TeamsAppID) in the appsettings.json file. To do this go to the **Teams Developer Portal** <https://dev.teams.microsoft.com/> and sign in with the Test/Dev Tenant Administrator account. Select **Apps** and click **+ New App**.



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

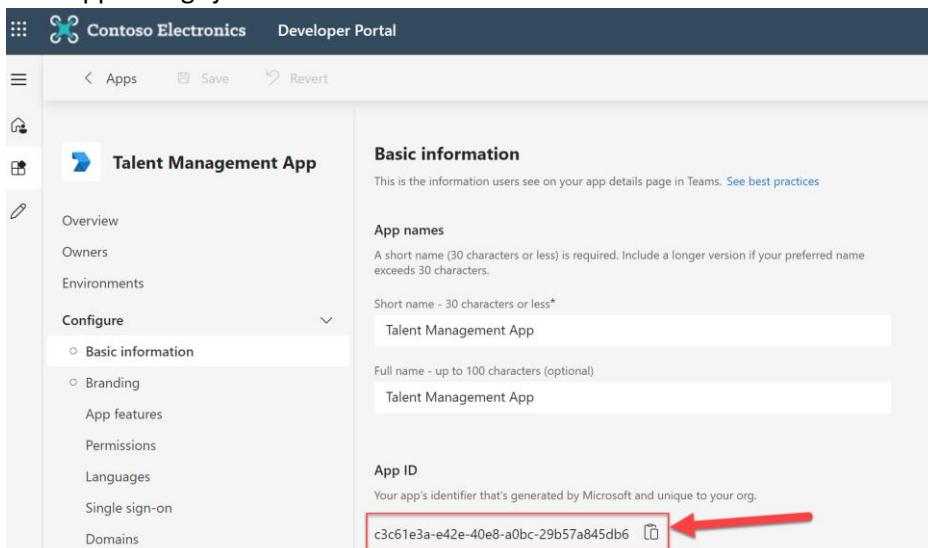


54. We now need to enter some placeholder values into the Basic Information configuration of your Teams app. Start with copying the Short Name **Talent Management App**, and paste it into the **Full Name**, **Short Description** and **Long Description** fields. Set the version as **1.0.0**. Developer 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**.



55. Copy the **App ID** and paste this into appsettings.json as the value of **TeamsAppID** (line 3).

Save appsettings.json.

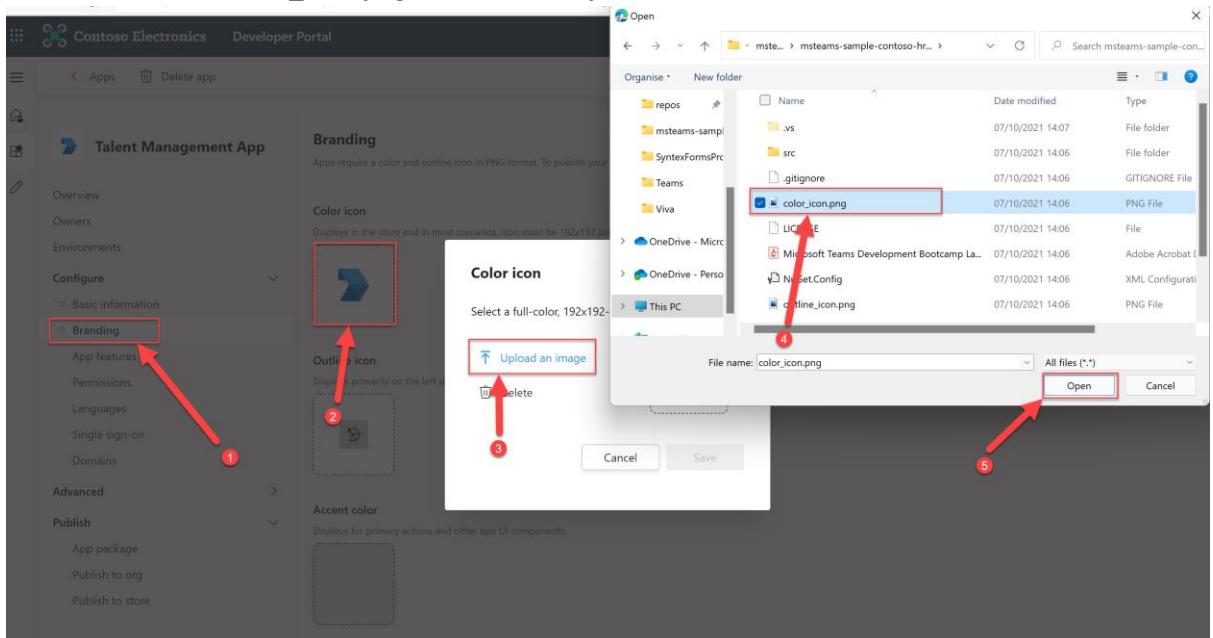


```

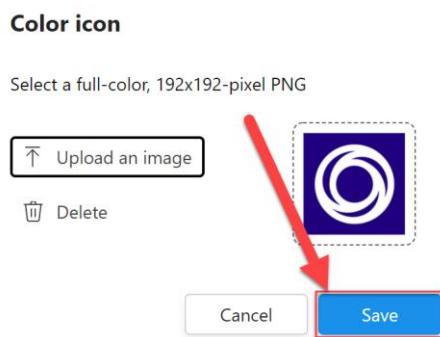
appsettings.json manifest.json* Startup.cs common.js Login.html launchSettings.json
Schema: https://json.schemastore.org/appsettings.json
1 "BaseUrl": "https://jalew123.eu.ngrok.io/",
2 "TeamsAppId": "c3c61e3a-e42e-40e8-a0bc-29b57a845db6",
3 "MicrosoftAppId": "5652156c-2020-42c4-b6f2-b942a45c526e",
4 "MicrosoftAppDirectoryID": "d06ea5c6-1047-45d0-8ea9-7de7d40e3c58",
5 "MicrosoftAppPassword": "Jj.7Q-YeYOr-R2q118u1CKR2GanYhkY_41Bd1",
6 "ApplicationIdUri": "api://jalew123.eu.ngrok.io/botid-5652156c-2020-42c4-b6f2-b942a45c526e",
7 "OAuthConnectionName": "AAD",
8

```

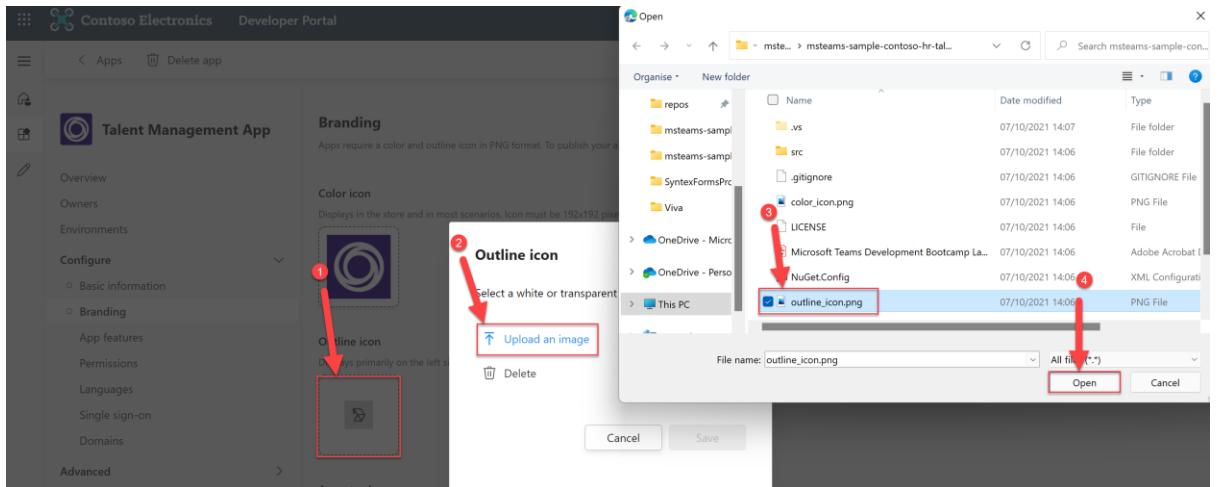
56. Go back into the Teams Developer Portal, and select **Branding**, click the **Colour icon**, select **Upload an image**, navigate to the folder that contains the application you downloaded from GitHub, select the **color_icon.png** file and click **Open**



57. Click **Save**



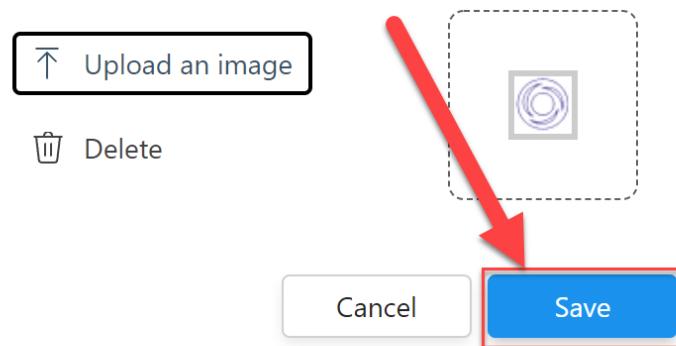
58. Click the **Outline icon**, select **Upload an image**, navigate to the folder that contains the application you downloaded from GitHub, select the **outline_icon.png** file and click **Open**



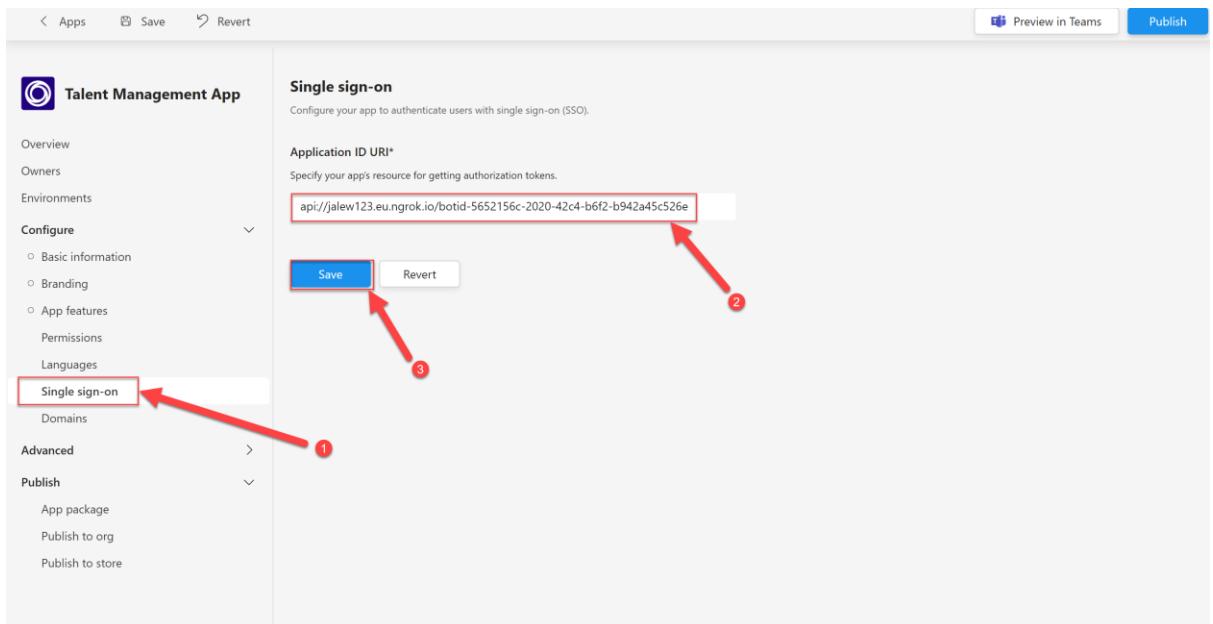
59. Click **Save**

Outline icon

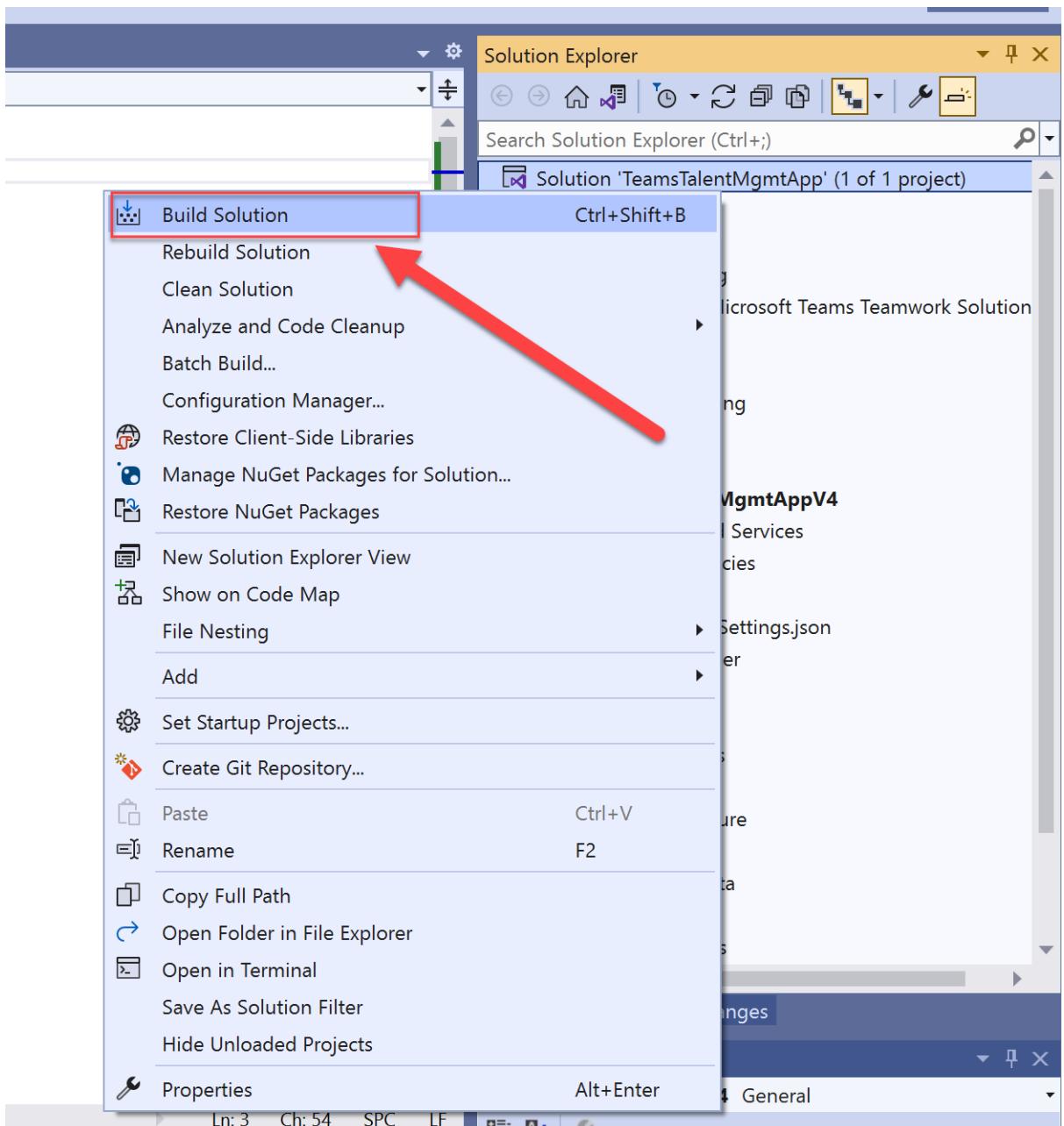
Select a white or transparent 32x32-pixel PNG.



60. 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**.



61. 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**.



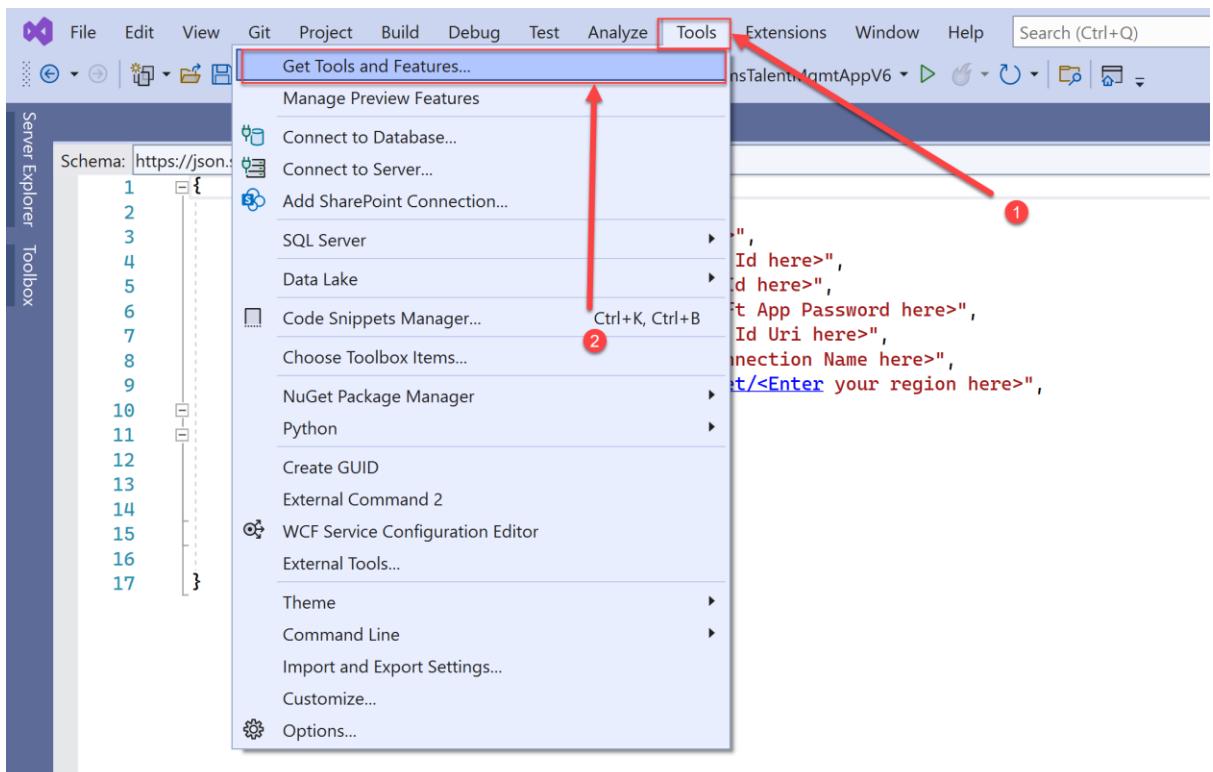
62. At the bottom of Visual Studio, the Output section will begin populating with text. Once the Output confirms that the build succeeds, move onto the next step.

```
Output
Show output from: Build
Build started...
1>----- Build started: Project: TeamsTalentMgmtAppV4, Configuration: Debug Any CPU -----
1>TeamsTalentMgmtAppV4 -> C:\users\jalew\Downloads\msteams-sample-contoso-hr-talent-app-master\msteams-sample-contoso-hr-talent-app-master\src\TeamsTalent
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

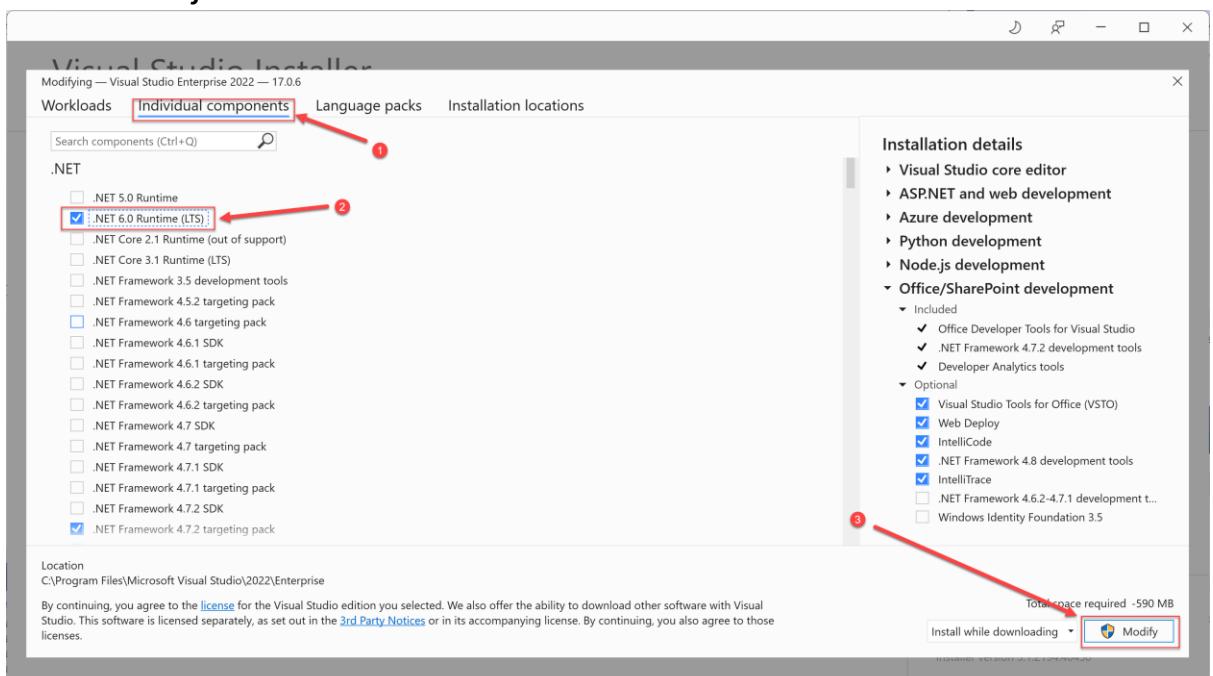
A screenshot of the Visual Studio 'Output' window. The window title is 'Output' and the dropdown says 'Show output from: Build'. The text area contains the build log: 'Build started...', '1>----- Build started: Project: TeamsTalentMgmtAppV4, Configuration: Debug Any CPU -----', '1>TeamsTalentMgmtAppV4 -> C:\users\jalew\Downloads\msteams-sample-contoso-hr-talent-app-master\msteams-sample-contoso-hr-talent-app-master\src\TeamsTalent', and '===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ====='. A red arrow points to the success message at the end of the log.

NOTE: If your build fails, it is likely caused by either; the .NET 6 Runtime not being available in Visual Studio, or the NuGet package manager settings not being setup correctly.

To resolve .Net Runtime issues, click Tools and select Get Tools and Features:

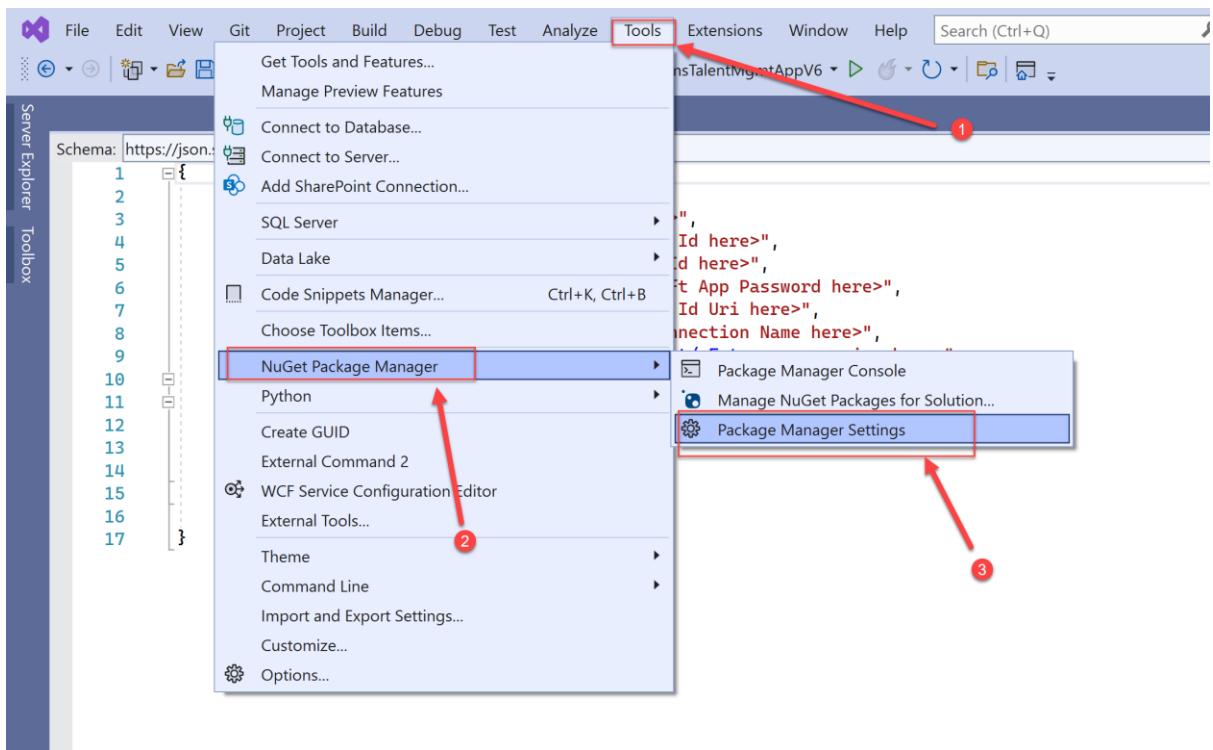


Select Individual components, click .NET 6.0 Runtime (LTS) and click Modify to install the .NET 6 runtime for use in Visual Studio.

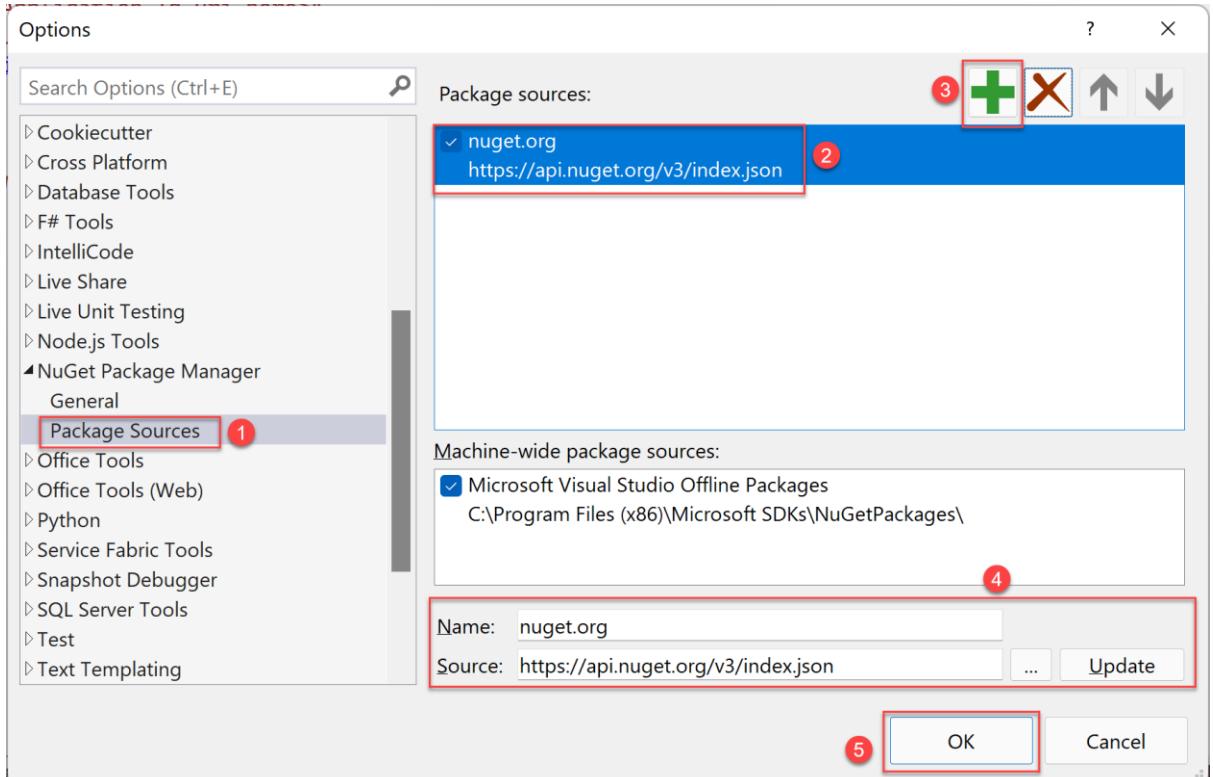


To resolve NuGet Package Manager restore issues:

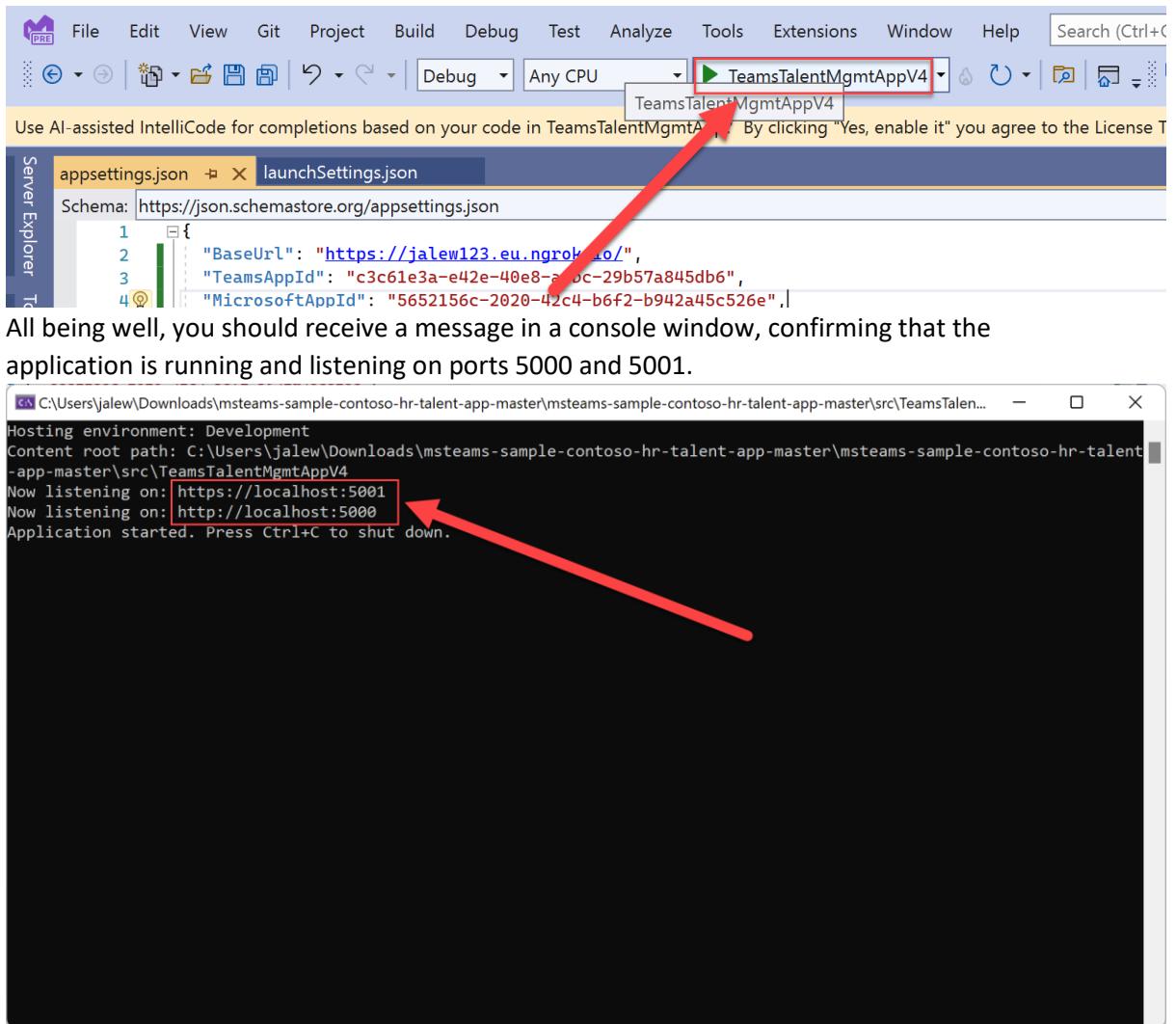
Select Tools, highlight NuGet Package Manager and select Package Manager Settings



Select Package Sources and review the Package Sources. If nuget.org appears as it does in the below screenshot, your NuGet Package Sources are configured correctly. If not, select +, and enter the Name of nuget.org and Source of <https://api.nuget.org/v3/index.json> select OK.



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



64. 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.
65. 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:

Job Title	Location	Days Open	Applied	Screening	Interviewing	Offered
Senior Software Engineer	New York, NY	75	0	0	3	1
Full Stack Developer	Chicago, IL	40	0	0	1	1
Senior Designer	San Francisco, CA	30	2	0	0	1

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**.

The screenshot shows the 'Talent Management App' configuration page. On the left, there's a sidebar with sections like Overview, Owners, Environments, Configure (with sub-options Basic information and Branding), App features (which is highlighted with a red box and arrow 1), Permissions, Languages, Single sign-on, Domains, Advanced, Publish (with sub-options App package, Publish to org, Publish to store), and a preview in Teams button.

The main area is titled 'App features' and contains a heading 'Select a feature to add'. It lists several options:

- Personal app**: A dedicated workspace or bot to help individual users focus on their own tasks or view activities important to them. (This option is highlighted with a red box and arrow 2.)
- Group and channel app**: A space to display hosted app experiences (such as a list or dashboard) in team channels and group chats.
- Bot**: A conversational UI that can perform a set of tasks, reply to questions, and proactively send notifications.
- Connector**: A way to automatically send notifications and messages from your app to a channel.
- Meeting extension**: Options for integrating your app with the Teams meeting experience, including the meeting stage and chat.
- Scene**: A custom virtual scene people can use in their Teams Together mode meetings.
- Activity feed notification**: Keeps users informed and engaged with app notifications in the activity feed.

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

This screenshot shows the 'Personal app' configuration page. At the top, it says 'Personal apps are a set of tabs scoped for individual use. These tabs can be like a webpage (e.g., a Home tab) or an area to message a bot (e.g., a Chat tab).'. Below this is a large blue button labeled 'Create your first personal app tab' with a red arrow pointing down to it. Above the button is an icon depicting a folder containing documents and a chart.

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.

Add a tab to your personal app

Define a set of tabs to display in your personal app. An About tab is created automatically by default. [Learn more](#)

Name*

Positions

1

Entity ID*

09b5e445-4113-4924-80af-eef05e80e564

2

Content URL*

<https://jalew123.eu.ngrok.io/StaticViews/OpenPositionsPersonalTab.html>

3

Website URL

<https://jalew123.eu.ngrok.io/StaticViews/OpenPositionsPersonalTab.html>

4

Cancel

5

Add

4. We are now in a place where we can deploy this application into Teams to review how the Tabbed interfaces work. There are a couple of options you can use to get the app installed into Teams, so that you can test it, but as we are only testing Tab interfaces in this initial app, we will deploy it as a sideloaded app for an individual user. When we introduce Bots into the app, in section 4, we will install the Teams app as a sideloaded app into the organisational Teams app store, to keep the Teams App IDs consistent across the organisation (more on this later, and why it is important when working with apps that contain bots!). To sideload the app for an individual Teams user click **Save** and select **Preview in Teams**

The screenshot shows the Microsoft Teams App Studio interface. On the left, the navigation pane includes sections like Overview, Owners, Environments, Configure (with sub-options like Basic information, Branding, App features, Permissions, Languages, Single sign-on, Domains), Advanced, Publish (with sub-options like App package, Publish to org, Publish to store), and a preview section for the 'Talent Management App'. The main content area is titled 'Personal app' and contains fields for 'Tab name' (set to 'Positions') and 'URL' (set to 'https://jalew123.eu.ngrok.io/StaticViews/OpenPositionsPersonalTab.html'). Below these fields is a button labeled '+ Add a personal tab'. At the bottom of the configuration area are two buttons: 'Save' (highlighted with a red box and arrow) and 'Revert'. In the top right corner, there is a 'Preview in Teams' button (also highlighted with a red box and arrow) and a 'Publish' button.

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

The screenshot shows a Microsoft Teams browser dialog box. It displays the message: 'This site is trying to open Microsoft Teams.' followed by the URL 'https://teams.microsoft.com wants to open this application.' There is a checkbox labeled 'Always allow teams.microsoft.com to open links of this type in the associated app' and two buttons: 'Open' and 'Cancel' (highlighted with a red box and arrow). A red arrow labeled '1' points from the 'Cancel' button to the 'Use the web app instead' button on the subsequent web page. The web page itself has a heading 'Stay better connected with the Teams desktop app' and two buttons: 'Download the Windows app' and 'Use the web app instead' (highlighted with a red box and arrow). Below these buttons is the text 'Already have the Teams app? Launch it now'.

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

Talent Management App

This app is not from your organisation's app catalogue or the Teams store. Do not proceed to add the app unless you are testing it in development or trust the person who shared it with you.

Add

About

Permissions

Talent Management App

Personal app
Keep track of important content and info

Created by: Contoso
Version 1.0.0

Permissions

This app will have permission to:

- Receive messages and data that I provide to it.
- Access my profile information such as my name, email address, company name, and preferred language.

By using Talent Management App, you agree to the [privacy policy](#) and [terms of use](#).

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

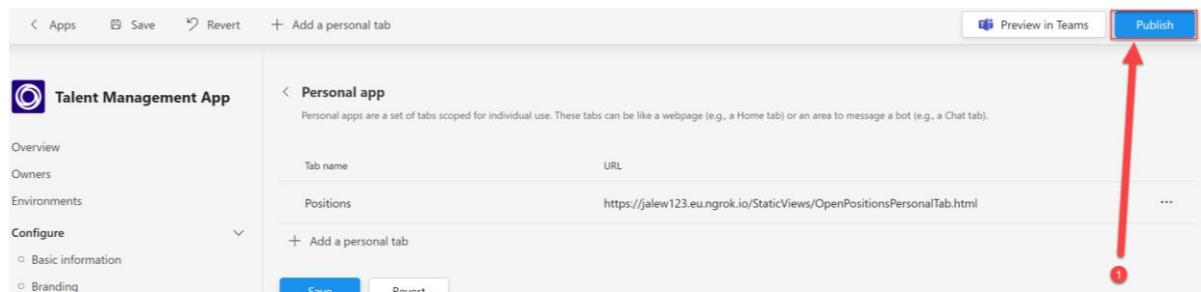
Microsoft Teams

Talent Management App Positions About

You are logged in as admin@M365x265372.onmicrosoft.com

Job Details	Location	Applied	Screening	Interviewing	Offered
Senior Software Engineer	New York, NY	0	0	3	1
Full Stack Developer	Chicago, IL	0	0	1	1
Senior Designer	San Francisco, CA	2	0	0	1

- 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**.

Publish your app



Download the app package

Download a copy of your app package, which is specific to your selected environment. Use the package to upload your app in Teams or publish later.

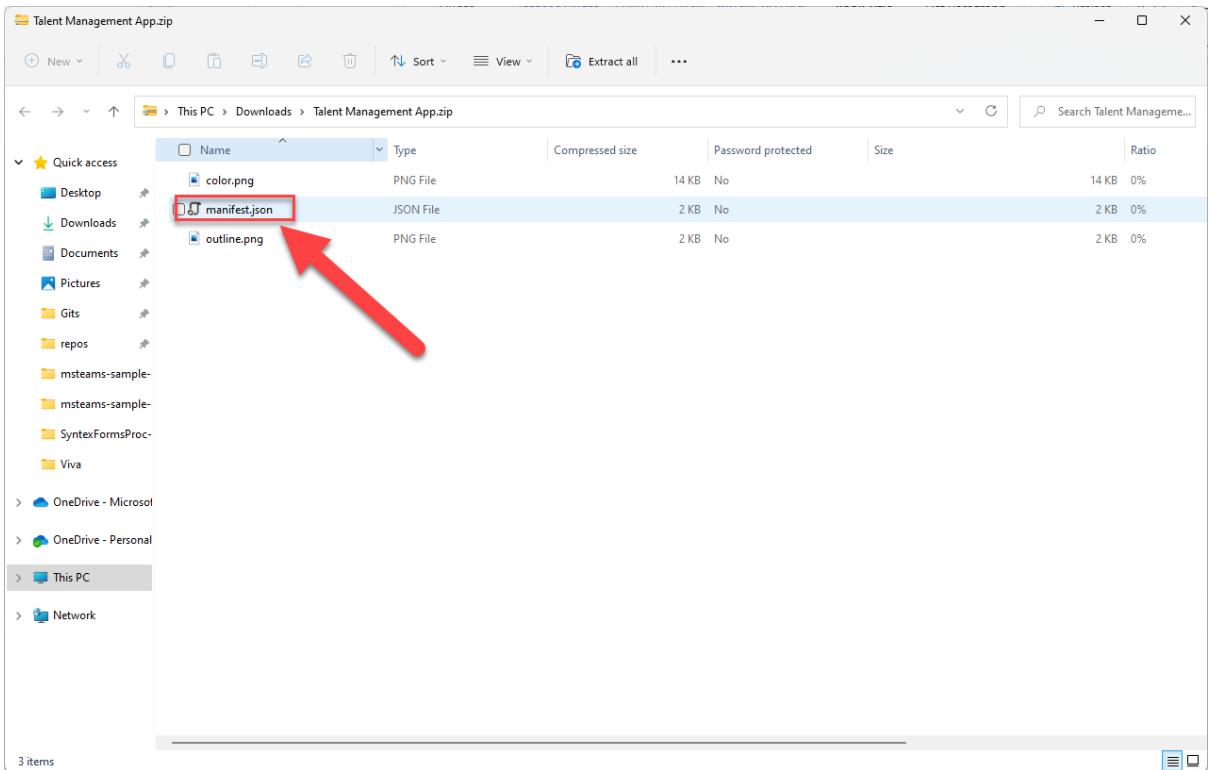
Publish to your org

Submit a request to your IT admin to publish your app. It will appear in the Built for your org section of the store once it's approved.

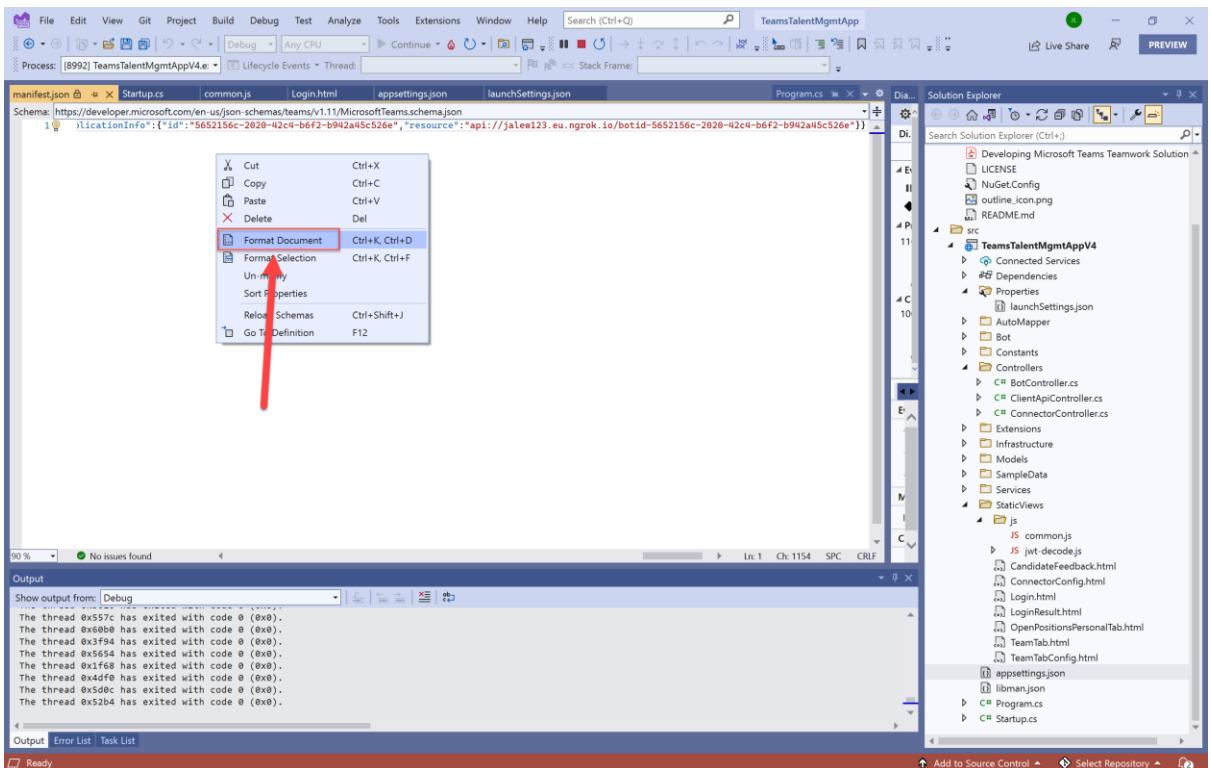
Publish to the Teams store

Make your app available to Teams users everywhere. This option requires Microsoft approval.

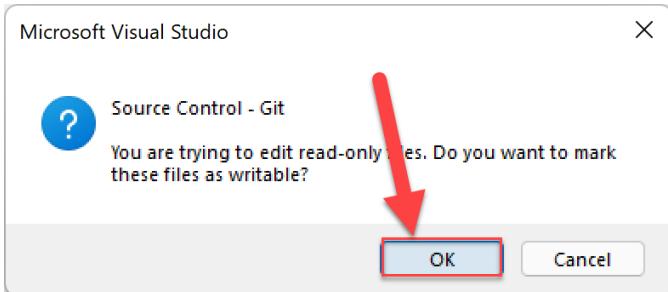
10. Open the ZIP that contains your manifest that has just been downloaded and open the **manifest.json** file in Visual Studio.



- 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.

```

manifest.json* ✘ Startup.cs common.js Login.html appsettings.json launchSettings.json Program.cs
Schema: https://developer.microsoft.com/en-us/json-schemas/teams/v1.11/MicrosoftTeams.schema.json
1  {
2    "$schema": "https://developer.microsoft.com/en-us/json-schemas/teams/v1.11/MicrosoftTeams.schema.json",
3    "version": "1.0.0",
4    "manifestVersion": "1.11",
5    "id": "c3c61e3a-e42e-a0bc-29b57a845db6",
6    "packageName": "com.package.name",
7    "name": {
8      "short": "Talent Management App",
9      "full": "Talent Management App"
10     },
11    "developer": {
12      "name": "Contoso",
13      "mpnId": "",
14      "websiteUrl": "https://jalew.eu.ngrok.io/",
15      "privacyUrl": "https://jalew.eu.ngrok.io/privacy",
16      "termsOfUseUrl": "https://jalew.eu.ngrok.io/termsofuse"
17    },
18    "description": {
19      "short": "Talent Management App",
20      "full": "Talent Management App"
21    },
22    "icons": {
23      "outline": "outline.png",
24      "color": "color.png"
25    },
26    "accentColor": "#FFFFFF",
27    "staticTabs": [
28      {
29        "entityId": "09b5e445-4113-4924-80af-eef05e80e564",
30        "name": "Positions",
31        "contentUrl": "https://jalew123.eu.ngrok.io/StaticViews/OpenPositionsPersonalTab.html",
32        "websiteUrl": "https://jalew123.eu.ngrok.io/StaticViews/OpenPositionsPersonalTab.html",
33        "scopes": [
34          "personal"
35        ],
36      },
37      {
38        "entityId": "about",
39        "scopes": [
40          "personal"
41        ]
42      },
43    ],
44    "validDomains": [
45      "jalew123.eu.ngrok.io"
46    ],
47    "webApplicationInfo": {
48      "id": "5652156c-2020-42c4-b6f2-b942a45c526e",
49      "resource": "api://jalew123.eu.ngrok.io/botid=5652156c-2020-42c4-b6f2-b942a45c526e"
50    }
51  }

```

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.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Open Positions</title>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
    <script src="https://unpkg.com/microsoft/teams-jss@1.5.0/dist/MicrosoftTeams.min.js" integrity="sha384-EK79wsdn78EAMQeoH50wmaPfH3aPa7YzI"></script>
    <script src="https://cdn.jsdelivr.net/npm/@microsoft/fabric/dist/fabric.min.js" integrity="sha256-FppCb/KJQLNFou91ta32o/NMzLxtRro8QtLmkPRd/></script>
    <script type="text/javascript" src="https://atcdn.msauth.net/browser/2.16.1/js/msal-browser.js"></script>
    <script type="text/javascript" src="~/StaticViews/js/jwt-decode.js"></script>
    <script type="text/javascript" src="~/js/common.js"></script>
    <LINK rel="stylesheet" href="https://static2.sharepointonline.com/files/fabric/office-ui-fabric-core/9.6.1/css/fabric.min.css" />
    <link rel="stylesheet" href="https://static2.sharepointonline.com/files/fabric/office-ui-fabric/1.4.0/css/fabric.min.css" />
<style type="text/css">
    .row {
        padding: 5px 20px;
    }

    .wideRow {
        padding: 20px;
    }

    .teams {
        color: #5B6775;
        font-family: "Segoe UI Web (West European)", "Segoe UI", -apple-system, BlinkMacSystemFont, Roboto, Helvetica Neue, sans-serif;
    }
</style>
```

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).

```
235         if (activeIndex != 0) { newIndex -= 1; }
236         break;
237 
238     case 40:
239         if (activeIndex != maxActiveIndex) { newIndex += 1; }
240         break;
241 
242     default: return;
243   }
244   clickOnMainBlock($("#main-block" + newIndex));
245 }
246 
247 function clickOnMainBlock(currentObject) {
248   var index = currentObject.data("index");
249   if (index != activeIndex) {
250     $(".active").removeClass("active");
251     currentObject.addClass("active");
252   }
253   activeIndex = index;
254 }
255 
256 
257 function getQueryStringValue(key) {
258   return decodeURIComponent(window.location.search.replace(new RegExp(`^.*${key}\\?` + encodeURIComponent(key).replace(/[\.,\+\\*]/g, `\\\$<1>`), "g")));
259 }
260 
261 $async((C) => {
262   await this.handlePageLoad();
263   this.loadPage();
264 })
265 
266 
267 </script>
268 </head>
269 <body dir="ltr">
```

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.

```
1 handlePageLoad = async () => {
2
3     try {
4         window.localStorage.removeItem("isTeams");
5         await this.initialiseTeams();
6         this.inTeams = true;
7     }
8     catch {
9         this.inTeams = false;
10    }
11
12    if (this.inTeams) {
13        await this.signInWithTeams();
14    }
15    else {
16        outsideTeamsSignIn();
17    }
18}
```

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 audience 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);
    }
});
}

```

The code is annotated with two red arrows pointing to specific sections:

- Section 1:** Points to the `microsoftTeams.authentication.getAuthToken` call within the `getTeamsToken` function. A red box surrounds the entire callback object.
- Section 2:** Points to the `microsoftTeams.authentication.authenticate` call within the `teamsFallbackAuth` function. A red box surrounds the entire callback object.

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 to 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.

```

94     function outsideTeamsSignIn() {
95       let accessToken = window.localStorage.getItem("userToken");
96       if (!accessToken) {
97         authRedirect();
98       }
99     }
100
101   function authRedirect() {
102     //window.localStorage.setItem("sourceHostname", location.protocol + "//" + location.hostname + ":" + location.port + "/StaticViews/OpenPos");
103     window.localStorage.setItem("sourceHostname", window.location.origin + window.location.pathname + window.location.search);
104     console.log("redirectURL will be: " + window.localStorage.getItem("sourceHostname"));
105     window.location.assign(window.location.origin + "/StaticViews/Login.html");
106   }
107
108   function getInfoFromToken(accessToken) {
109     var token = accessToken;
110     var decoded = jwt_decode(token);
111     let preferredUsername = decoded.preferred_username;
112     window.localStorage.setItem("username", preferredUsername);
113   }
114 }

```

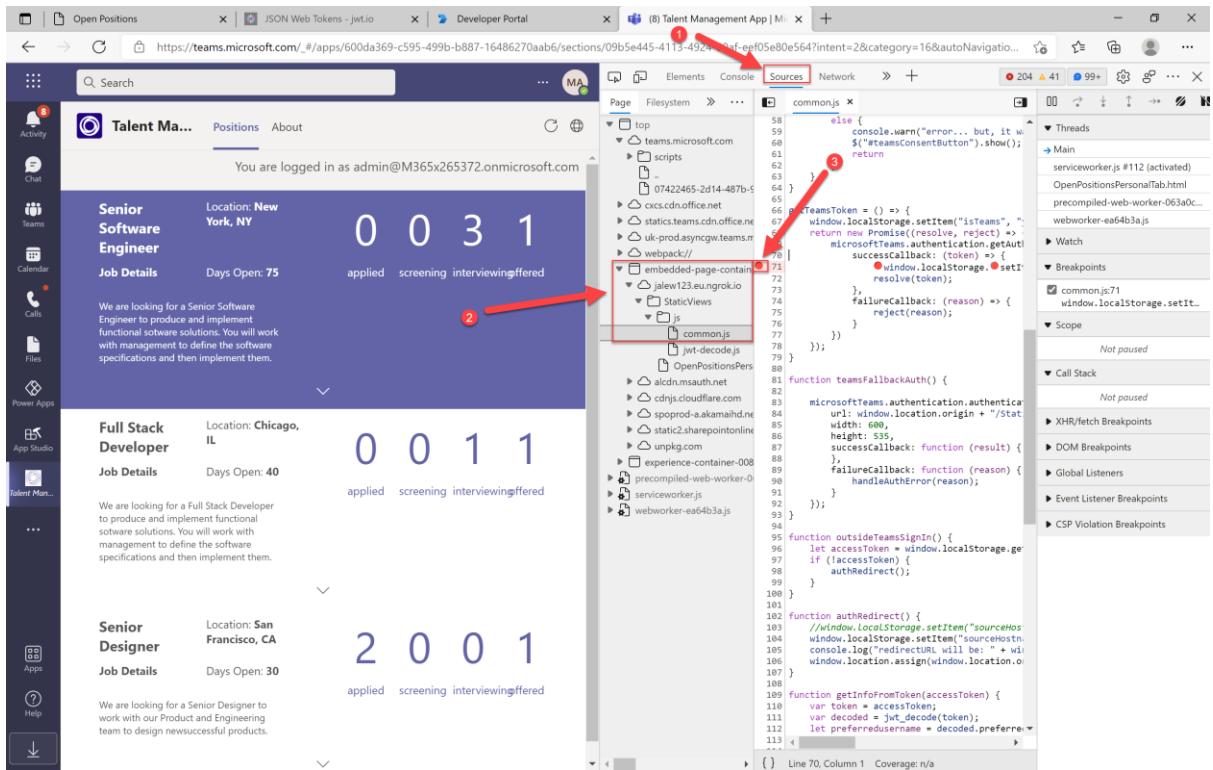
19. Go back to **OpenPositionsPersonalTab.html** and review the **loadPage** function. This will call the **getData** function. **getData** uses fetch within JavaScript to make a HTTP request, with the access token obtained from Azure AD in the header, to the apps Client APIs, this will return information 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!

```

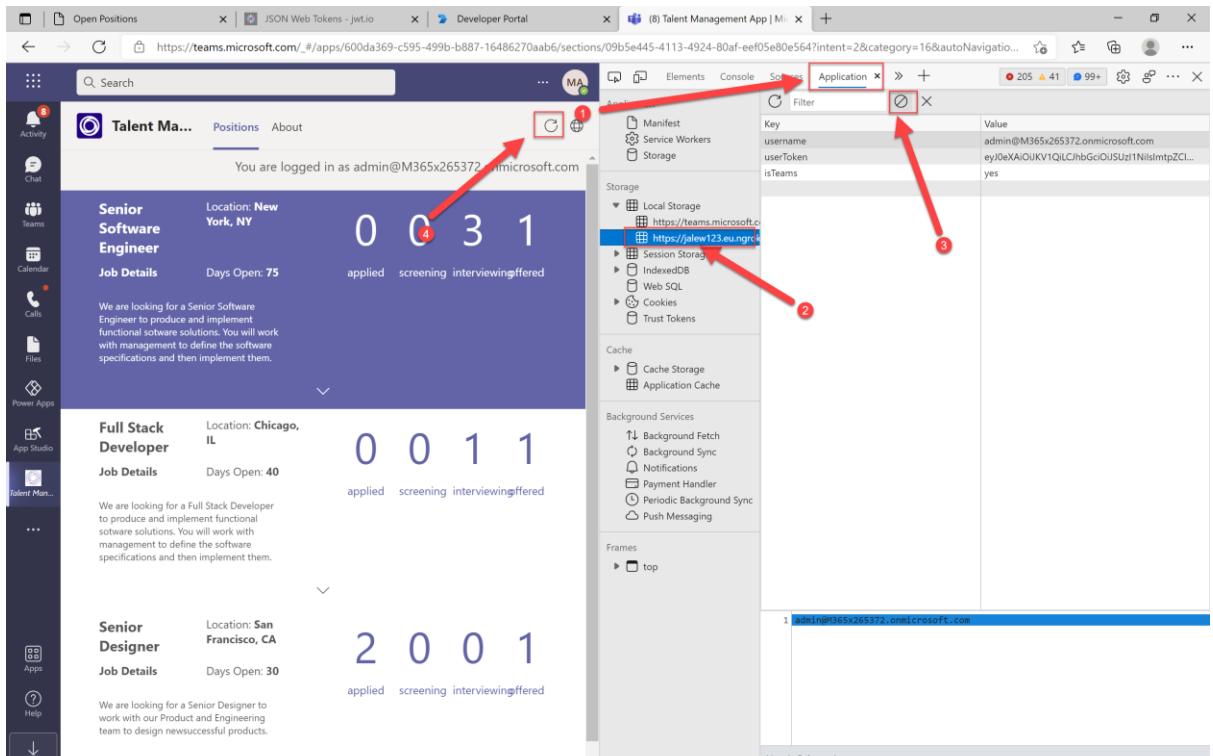
103 </style>
104
105 <script type="text/javascript">
106
107   async function loadPage() {
108     var accessToken = window.localStorage.getItem("userToken");
109     getInfoFromToken(accessToken);
110     var positionId = getQuerystringValue('positionId');
111     getData(window.localStorage.getItem("username"), positionId);
112   }
113
114
115   async function getData(recruiterId, positionId) {
116     $('#button-holder').hide();
117     $('#loading').show();
118     var apiPath = "/api/positions/open";
119     if (recruiterId && recruiterId.lastIndexOf("@") !== -1) {
120       apiPath = "/api/recruiters/" + encodeURIComponent(recruiterId.substring(0, recruiterId.lastIndexOf("@")) + "/positions";
121     }
122
123     let accessToken = window.localStorage.getItem("userToken");
124     let authorization = "Bearer " + accessToken;
125     let data = await fetch(window.location.origin + apiPath, {
126       headers: {
127         Authorization: authorization,
128       }
129     });
130
131     if (data.ok) {
132       let json = await data.json();
133       buildPage(json, recruiterId, positionId);
134     } else {
135       console.error("HTTP-Error: " + data.status);
136     }
137
138   function buildPage(data, recruiterId, positionId) {
139     var activeIndex = 0;
140     var isTeamsContext = false;
141     var maxActiveIndex;
142     var positionHtml = "";

```

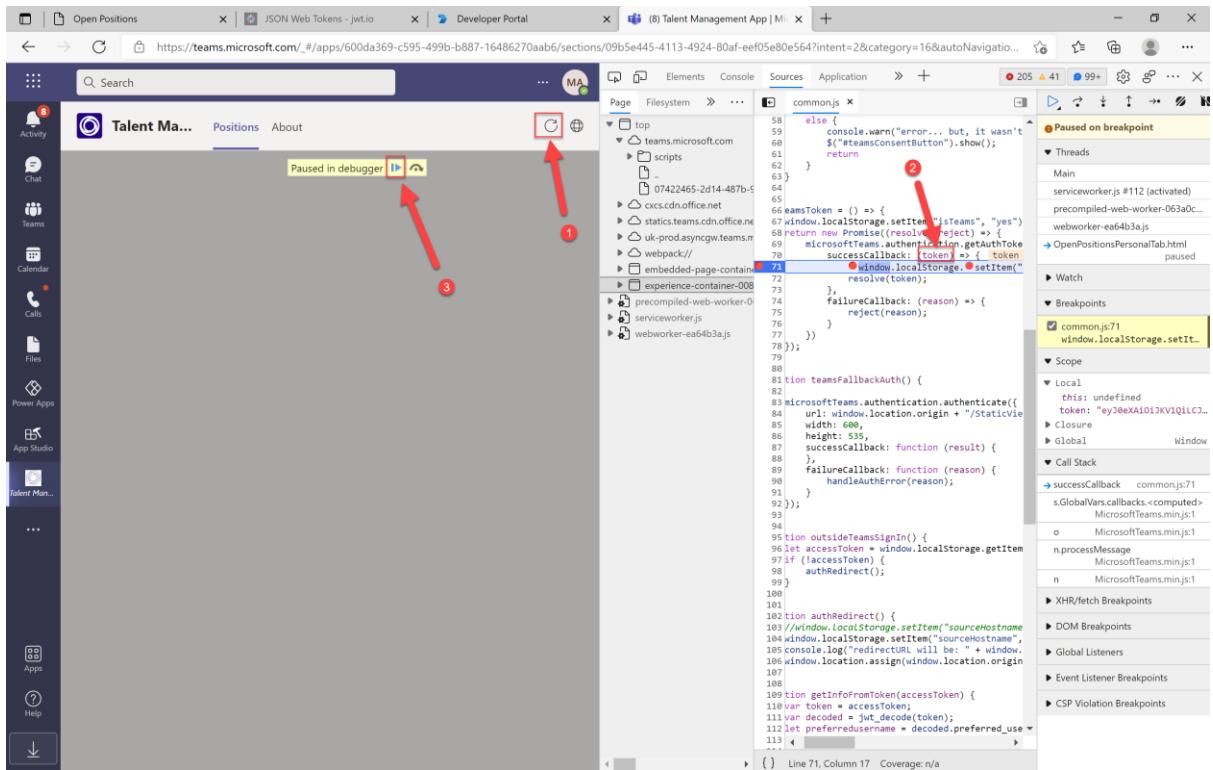
20. Let's see this in action! Go back into Teams and open the Talent Management App. Hit **F12** on your keyboard to open your browser's development tools. Select **Sources**, expand the folder hierarchy, until you find **common.js** (which is inside **StaticViews/js** folder, found within embedded-page/NGROK). 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).



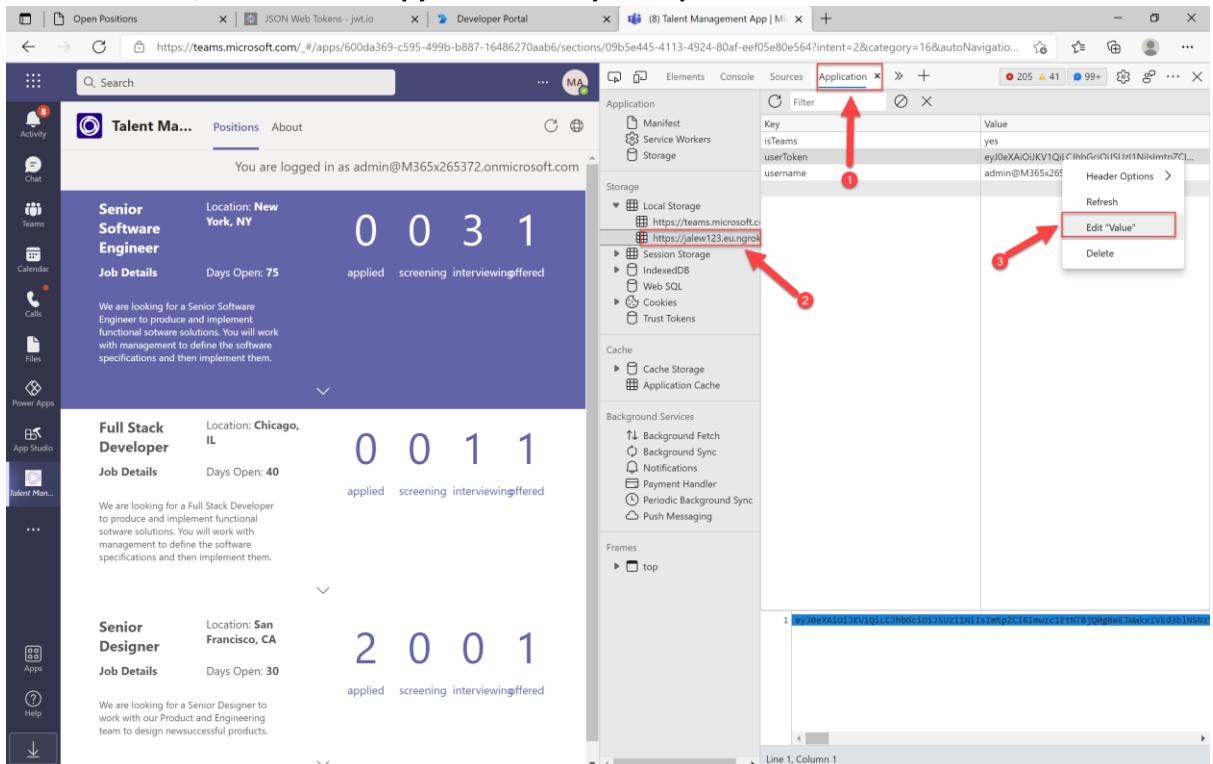
21. Before we re-load the page, it's a good idea to remove the items in Local Storage for this app. Click **Application**, expand **Local Storage** and select your NGROK app. Click the **clear** button.



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.

The screenshot shows a JWT token being analyzed on jwt.io. The token is pasted into the 'Encoded' field. The 'Decoded' field shows the structure of the JWT:

```

HEADER: ALGORITHM & TOKEN TYPE
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "13s0-58cCH4xBVZLHTGwN87680"
}

PAYLOAD: DATA
{
  "aud": "5652156c-2028-42c4-b6f2-b942a45c526e",
  "iss": "https://login.microsoftonline.com/d06ea5c6-1047-45d0-8ea9-7de7d40e3c58/v2.0",
  "iat": 1633704887,
  "nbf": 1633704887,
  "exp": 1633712978,
  "aio": "ATQAY/8TAAAAR8oXCsD6D1B7DBbf10WIaaN6vx5bqqXtdF79hYQy4pe9g96a1319aNg/SVW10b",
  "azp": "5e3ce6c0-2b1f-4285-8d4b-75ee78787346",
  "azpacr": "0",
  "name": "MOD Administrator",
  "oid": "0af4b8ce-b300-4ca9-992b-cd26956d7ad9",
  "preferred_username": "admin@M365x265372.onmicrosoft.com",
  "rh": "0.ARoAxqVu0EcQ0EW0qX3n1A48NM0mPF4fK4VCjUt17nh4c0Z5AIw",
  "scp": "access_as_user",
  "sub": "819n4kD0yfQ0Y0vpaXUea2czvmlrxvZ17WRj1P1t-Ag",
  "tid": "d06ea5c6-1047-45d0-8ea9-7de7d40e3c58",
  "uti": "LijSBxDN00STS167aiVKAA",
  ...
}

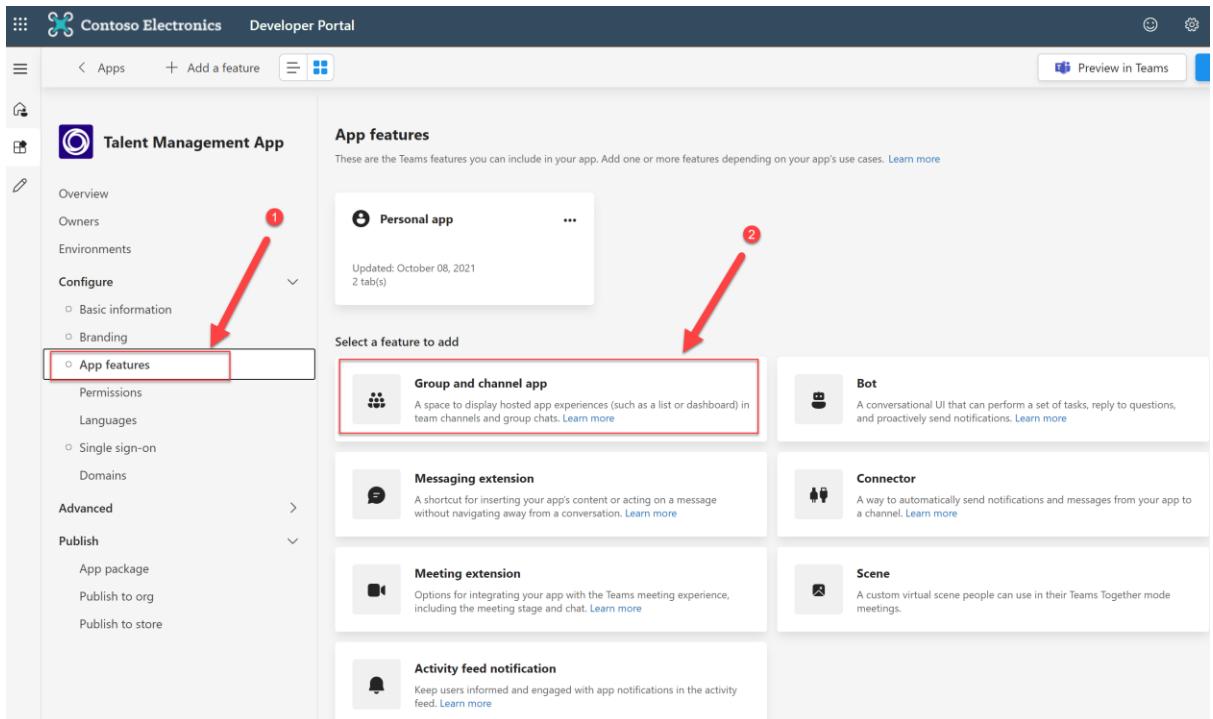
```

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.

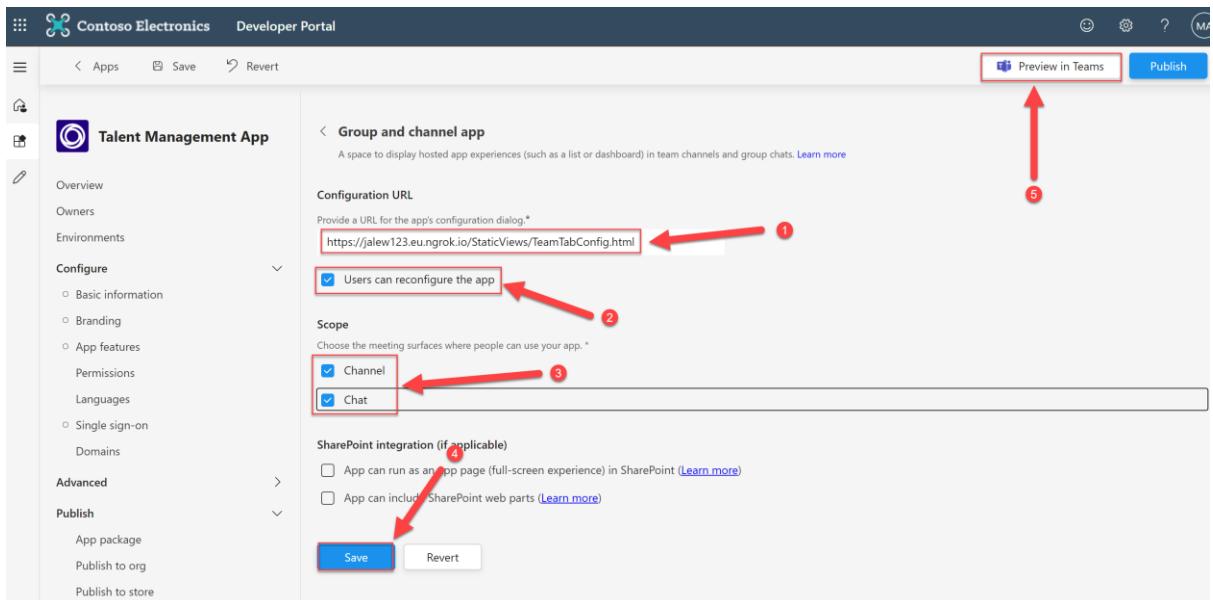
3) Implement a Tab inside a Team Channel

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.

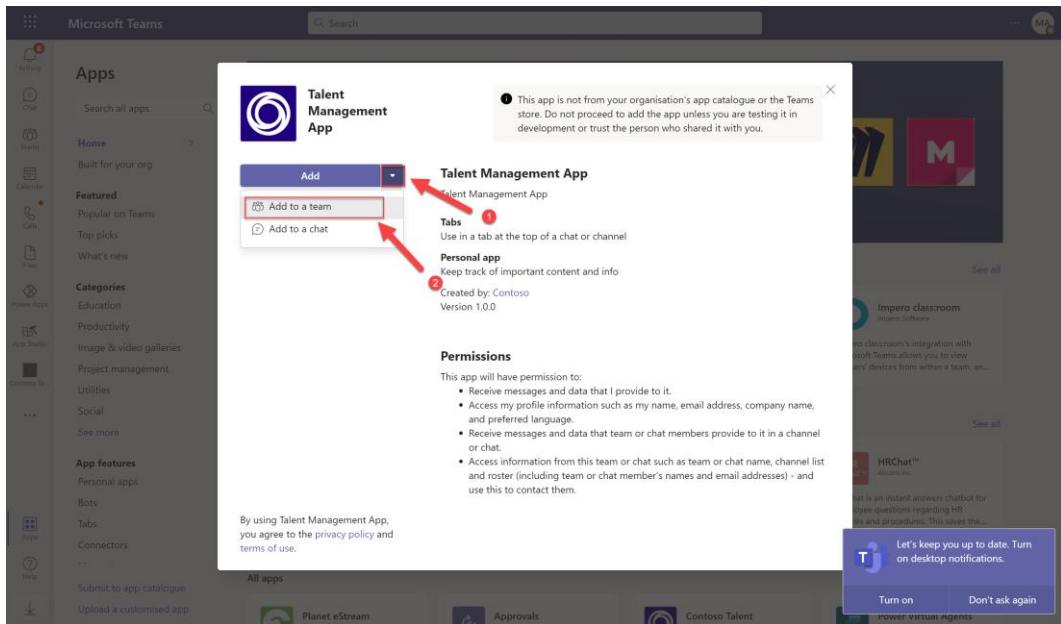
1. Go back to the Teams Developer Portal and open the Teams Talent Management App. Select App Features and click Group and channel app.



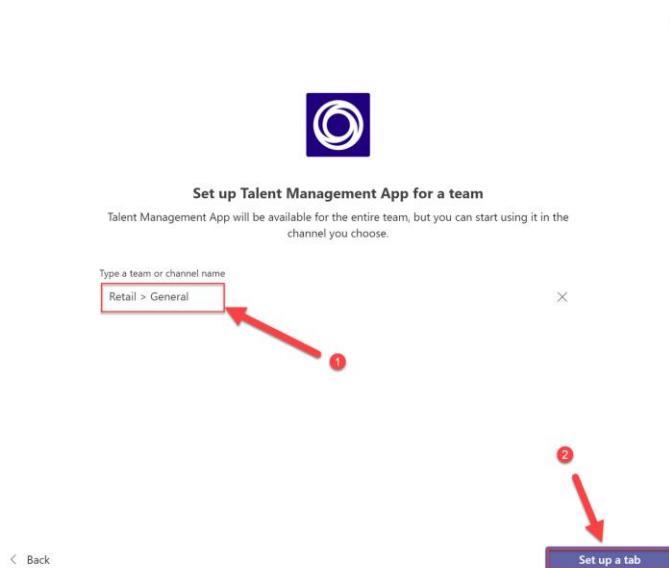
2. Enter the following into the Configuration URL **NGROK/StaticViews/TeamTabConfig.html** – select **Users can reconfigure the app** and select both **Channel** and **Chat** in the scopes. Click **Save**. For context, the Configuration URL is the URL that will open the Tab Configuration page, this is 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**.



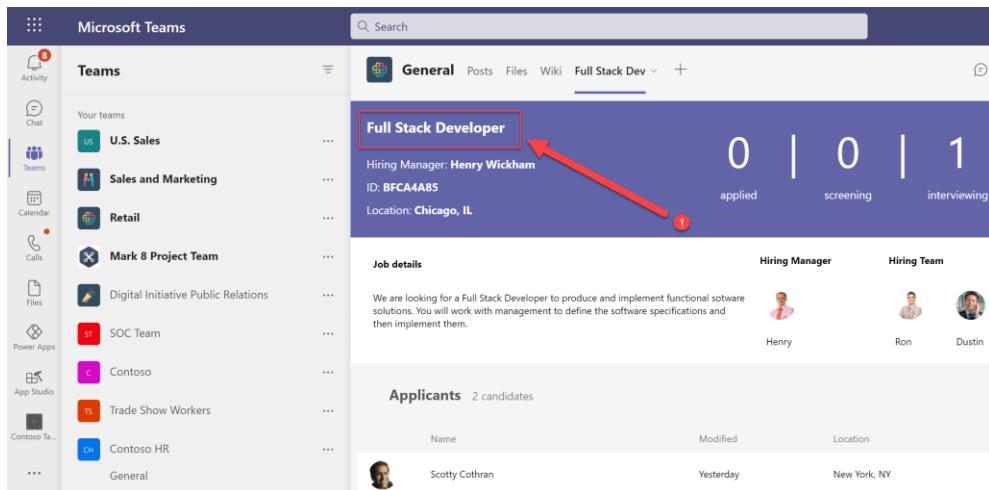
4. Select a Team and Channel, in this demo I selected **Retail -> General**, and click **Set up a tab**.



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**.



- 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.

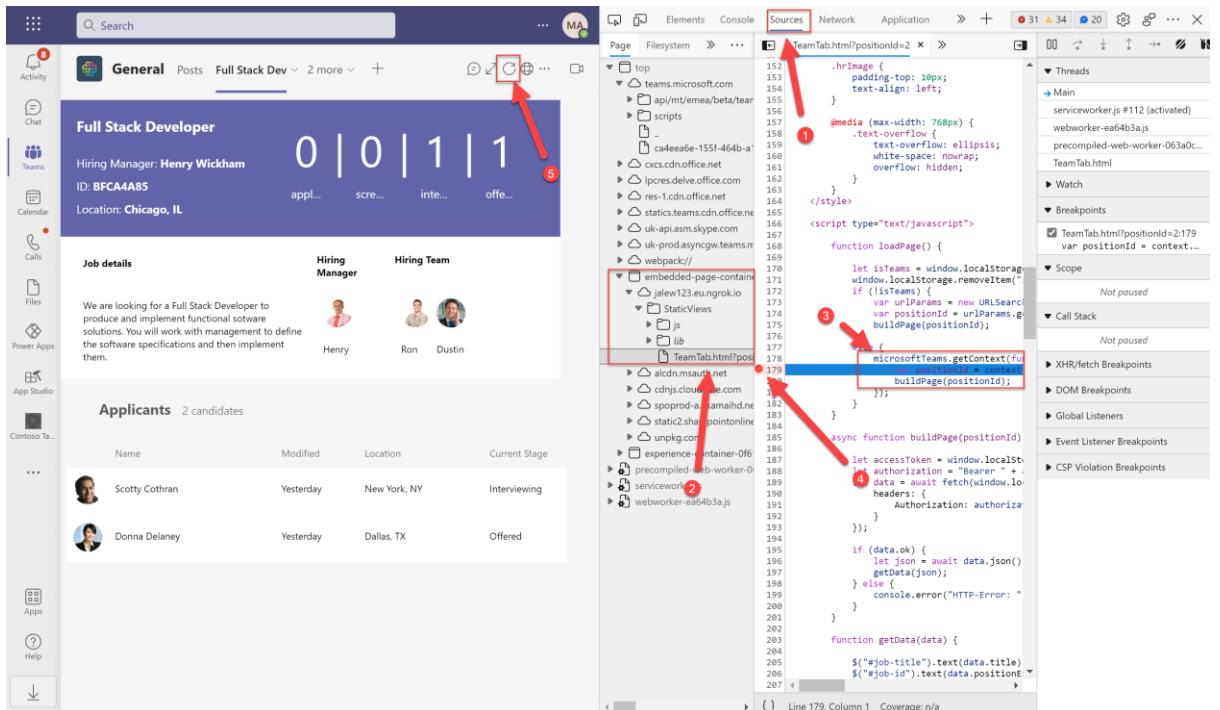


- This page has also been configured to load outside of Teams, by using the URL Search Strings, instead of the Teams Tab EntityID. To confirm this, click the **dropdown button**, and select **Go to website**.

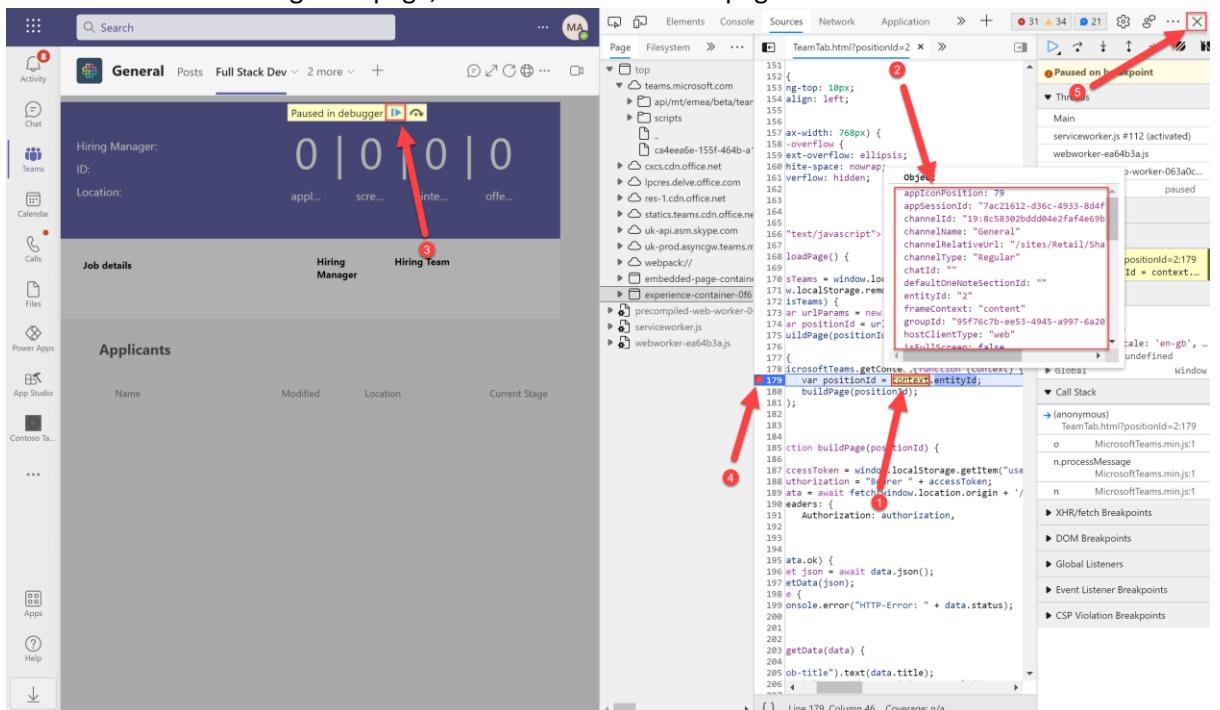
8. You will then see the page load successfully, and if you review the URL Search Strings, you will see **positionID & web** contain values.

Name	Modified	Location	Current Stage
Scotty Cochran	Yesterday	New York, NY	Interviewing
Donna Delaney	Yesterday	Dallas, TX	Offered

9. 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.



10. 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 a lot of useful information is available to you, when you are building your 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.



11. 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.

```

175     });
176   });
177 
178   $(".dropdownButton").click(function (e) {
179     $("#positionsDropdownContent").toggle();
180     e.stopPropagation();
181     e.cancelBubble = true;
182   });
183 
184   function rebuildDropdown() {
185     var html = "";
186     positionData.forEach(function (d) {
187       if (d.positionId === currentSelectedPositionId) {
188         var positionName = "ID " + d.positionExternalId + ' - ' + d.title;
189         html += '<span class="change-position-btn" data-position-id=' + d.positionId + ' data-position-name=' + positionName + '>';
190       }
191     });
192     $("#positionsDropdownContent").html(html);
193     $(".change-position-btn").click(function (e) {
194       $("#positionsDropdownContent").toggle();
195       updateSelectedPosition($(e.target).data("position-id"), $(e.target).data("position-name"));
196       rebuildDropdown();
197     });
198   }
199 
200   function updateSelectedPosition(positionId, positionName) {
201     currentSelectedPositionId = positionId;
202     $("#CurrentPosition").text(positionName);
203     $("#positionsDropdownContent").width($("#posting").outerWidth());
204     checkValidityState();
205   }
206 
207   //fetch stuff
208 
209   let accessToken = window.localStorage.getItem("userToken");
210   let authorization = "Bearer " + accessToken;
211   let url = "https://graph.microsoft.com/v1.0/me/jobs?$filter=positionExternalId eq '10000000000000000000000000000000' $select=id,externalId,externalName,externalUrl";
212 
213   if (data.ok) {
214     let json = await data.json();
215     updateSelectedPosition(json[0].positionId, 'ID ' + json[0].positionExternalId + ' - ' + json[0].title);
216     positionData = json;
217     rebuildDropdown();
218   } else {
219     console.error("HTTP-Error: " + data.status);
220   }
221 
222   //Save handler when user clicked on Save button
223   microsoftTeams.settings.registerOnSaveHandler(function (saveEvent) {
224 
225     microsoftTeams.getContext(function () {
226       var name = nameSelector.val();
227       var url = window.location.origin + '/StaticViews/TeamTab.html?positionId=' + currentSelectedPositionId;
228       var websiteUrl = url + '?web!1';
229 
230       console.log(nameSelector.val());
231       console.log(url);
232       console.log(websiteUrl);
233 
234       microsoftTeams.settings.setSettings({
235         entityId: currentSelectedPositionId,
236         contentUrl: url,
237         suggestedDisplayName: name,
238         websiteUrl: websiteUrl
239       });
240 
241       saveEvent.notifySuccess();
242     });
243   });
244 
245   $async () => {
246     await this.handlePageLoad();
247   };
248 
249   $();
250 
251   $();
252 
253   $();
254 
255   $();
256 
257   $();
258 
259   $();
260 
261   $();
262 
263   $();
264 
265   $();
266 
267   $();
268 
269   $();
270 
271   $();
272 
273   $();
274 
275   $();
276 
277   $();
278 
279   $();
280 
281   $();
282 
283   $();
284 
285   $();
286 
287   $();
288 
289   $();
290 
291   $();
292 
293   $();
294 
295   $();
296 
297   $();
298 
299   $();
299 
300   $();
301 
302   $();
303 
304   $();
305 
306   $();
307 
307   $();
308 
309   $();
309 
310   $();
311 
311   $();
312 
312   $();
313 
313   $();
314 
314   $();
315 
315   $();
316 
316   $();
317 
317   $();
318 
318   $();
319 
319   $();
320 
320   $();
321 
321   $();
322 
322   $();
323 
323   $();
324 
324   $();
325 
325   $();
326 
326   $();
327 
327   $();
328 
328   $();
329 
329   $();
330 
330   $();
331 
331   $();
332 
332   $();
333 
333   $();
334 
334   $();
335 
335   $();
336 
336   $();
337 
337   $();
338 
338   $();
339 
339   $();
340 
340   $();
341 
341   $();
342 
342   $();
343 
343   $();
344 
344   $();
345 
345   $();
346 
346   $();
347 
347   $();
348 
348   $();
349 
349   $();
350 
350   $();
351 
351   $();
352 
352   $();
353 
353   $();
354 
354   $();
355 
355   $();
356 
356   $();
357 
357   $();
358 
358   $();
359 
359   $();
360 
360   $();
361 
361   $();
362 
362   $();
363 
363   $();
364 
364   $();
365 
365   $();
366 
366   $();
367 
367   $();
368 
368   $();
369 
369   $();
370 
370   $();
371 
371   $();
372 
372   $();
373 
373   $();
374 
374   $();
375 
375   $();
376 
376   $();
377 
377   $();
378 
378   $();
379 
379   $();
380 
380   $();
381 
381   $();
382 
382   $();
383 
383   $();
384 
384   $();
385 
385   $();
386 
386   $();
387 
387   $();
388 
388   $();
389 
389   $();
390 
390   $();
391 
391   $();
392 
392   $();
393 
393   $();
394 
394   $();
395 
395   $();
396 
396   $();
397 
397   $();
398 
398   $();
399 
399   $();
400 
400   $();
401 
401   $();
402 
402   $();
403 
403   $();
404 
404   $();
405 
405   $();
406 
406   $();
407 
407   $();
408 
408   $();
409 
409   $();
410 
410   $();
411 
411   $();
412 
412   $();
413 
413   $();
414 
414   $();
415 
415   $();
416 
416   $();
417 
417   $();
418 
418   $();
419 
419   $();
420 
420   $();
421 
421   $();
422 
422   $();
423 
423   $();
424 
424   $();
425 
425   $();
426 
426   $();
427 
427   $();
428 
428   $();
429 
429   $();
430 
430   $();
431 
431   $();
432 
432   $();
433 
433   $();
434 
434   $();
435 
435   $();
436 
436   $();
437 
437   $();
438 
438   $();
439 
439   $();
440 
440   $();
441 
441   $();
442 
442   $();
443 
443   $();
444 
444   $();
445 
445   $();
446 
446   $();
447 
447   $();
448 
448   $();
449 
449   $();
450 
450   $();
451 
451   $();
452 
452   $();
453 
453   $();
454 
454   $();
455 
455   $();
456 
456   $();
457 
457   $();
458 
458   $();
459 
459   $();
460 
460   $();
461 
461   $();
462 
462   $();
463 
463   $();
464 
464   $();
465 
465   $();
466 
466   $();
467 
467   $();
468 
468   $();
469 
469   $();
470 
470   $();
471 
471   $();
472 
472   $();
473 
473   $();
474 
474   $();
475 
475   $();
476 
476   $();
477 
477   $();
478 
478   $();
479 
479   $();
480 
480   $();
481 
481   $();
482 
482   $();
483 
483   $();
484 
484   $();
485 
485   $();
486 
486   $();
487 
487   $();
488 
488   $();
489 
489   $();
490 
490   $();
491 
491   $();
492 
492   $();
493 
493   $();
494 
494   $();
495 
495   $();
496 
496   $();
497 
497   $();
498 
498   $();
499 
499   $();
500 
500   $();
501 
501   $();
502 
502   $();
503 
503   $();
504 
504   $();
505 
505   $();
506 
506   $();
507 
507   $();
508 
508   $();
509 
509   $();
510 
510   $();
511 
511   $();
512 
512   $();
513 
513   $();
514 
514   $();
515 
515   $();
516 
516   $();
517 
517   $();
518 
518   $();
519 
519   $();
520 
520   $();
521 
521   $();
522 
522   $();
523 
523   $();
524 
524   $();
525 
525   $();
526 
526   $();
527 
527   $();
528 
528   $();
529 
529   $();
530 
530   $();
531 
531   $();
532 
532   $();
533 
533   $();
534 
534   $();
535 
535   $();
536 
536   $();
537 
537   $();
538 
538   $();
539 
539   $();
540 
540   $();
541 
541   $();
542 
542   $();
543 
543   $();
544 
544   $();
545 
545   $();
546 
546   $();
547 
547   $();
548 
548   $();
549 
549   $();
550 
550   $();
551 
551   $();
552 
552   $();
553 
553   $();
554 
554   $();
555 
555   $();
556 
556   $();
557 
557   $();
558 
558   $();
559 
559   $();
560 
560   $();
561 
561   $();
562 
562   $();
563 
563   $();
564 
564   $();
565 
565   $();
566 
566   $();
567 
567   $();
568 
568   $();
569 
569   $();
570 
570   $();
571 
571   $();
572 
572   $();
573 
573   $();
574 
574   $();
575 
575   $();
576 
576   $();
577 
577   $();
578 
578   $();
579 
579   $();
580 
580   $();
581 
581   $();
582 
582   $();
583 
583   $();
584 
584   $();
585 
585   $();
586 
586   $();
587 
587   $();
588 
588   $();
589 
589   $();
590 
590   $();
591 
591   $();
592 
592   $();
593 
593   $();
594 
594   $();
595 
595   $();
596 
596   $();
597 
597   $();
598 
598   $();
599 
599   $();
599 
599   $();
600 
600   $();
601 
601   $();
602 
602   $();
603 
603   $();
604 
604   $();
605 
605   $();
606 
606   $();
607 
607   $();
608 
608   $();
609 
609   $();
610 
610   $();
611 
611   $();
612 
612   $();
613 
613   $();
614 
614   $();
615 
615   $();
616 
616   $();
617 
617   $();
618 
618   $();
619 
619   $();
620 
620   $();
621 
621   $();
622 
622   $();
623 
623   $();
624 
624   $();
625 
625   $();
626 
626   $();
627 
627   $();
628 
628   $();
629 
629   $();
630 
630   $();
631 
631   $();
632 
632   $();
633 
633   $();
634 
634   $();
635 
635   $();
636 
636   $();
637 
637   $();
638 
638   $();
639 
639   $();
640 
640   $();
641 
641   $();
642 
642   $();
643 
643   $();
644 
644   $();
645 
645   $();
646 
646   $();
647 
647   $();
648 
648   $();
649 
649   $();
650 
650   $();
651 
651   $();
652 
652   $();
653 
653   $();
654 
654   $();
655 
655   $();
656 
656   $();
657 
657   $();
658 
658   $();
659 
659   $();
660 
660   $();
661 
661   $();
662 
662   $();
663 
663   $();
664 
664   $();
665 
665   $();
666 
666   $();
667 
667   $();
668 
668   $();
669 
669   $();
670 
670   $();
671 
671   $();
672 
672   $();
673 
673   $();
674 
674   $();
675 
675   $();
676 
676   $();
677 
677   $();
678 
678   $();
679 
679   $();
680 
680   $();
681 
681   $();
682 
682   $();
683 
683   $();
684 
684   $();
685 
685   $();
686 
686   $();
687 
687   $();
688 
688   $();
689 
689   $();
690 
690   $();
691 
691   $();
692 
692   $();
693 
693   $();
694 
694   $();
695 
695   $();
696 
696   $();
697 
697   $();
698 
698   $();
699 
699   $();
700 
700   $();
701 
701   $();
702 
702   $();
703 
703   $();
704 
704   $();
705 
705   $();
706 
706   $();
707 
707   $();
708 
708   $();
709 
709   $();
710 
710   $();
711 
711   $();
712 
712   $();
713 
713   $();
714 
714   $();
715 
715   $();
716 
716   $();
717 
717   $();
718 
718   $();
719 
719   $();
720 
720   $();
721 
721   $();
722 
722   $();
723 
723   $();
724 
724   $();
725 
725   $();
726 
726   $();
727 
727   $();
728 
728   $();
729 
729   $();
730 
730   $();
731 
731   $();
732 
732   $();
733 
733   $();
734 
734   $();
735 
735   $();
736 
736   $();
737 
737   $();
738 
738   $();
739 
739   $();
740 
740   $();
741 
741   $();
742 
742   $();
743 
743   $();
744 
744   $();
745 
745   $();
746 
746   $();
747 
747   $();
748 
748   $();
749 
749   $();
750 
750   $();
751 
751   $();
752 
752   $();
753 
753   $();
754 
754   $();
755 
755   $();
756 
756   $();
757 
757   $();
758 
758   $();
759 
759   $();
760 
760   $();
761 
761   $();
762 
762   $();
763 
763   $();
764 
764   $();
765 
765   $();
766 
766   $();
767 
767   $();
768 
768   $();
769 
769   $();
770 
770   $();
771 
771   $();
772 
772   $();
773 
773   $();
774 
774   $();
775 
775   $();
776 
776   $();
777 
777   $();
778 
778   $();
779 
779   $();
780 
780   $();
781 
781   $();
782 
782   $();
783 
783   $();
784 
784   $();
785 
785   $();
786 
786   $();
787 
787   $();
788 
788   $();
789 
789   $();
790 
790   $();
791 
791   $();
792 
792   $();
793 
793   $();
794 
794   $();
795 
795   $();
796 
796   $();
797 
797   $();
798 
798   $();
799 
799   $();
800 
800   $();
801 
801   $();
802 
802   $();
803 
803   $();
804 
804   $();
805 
805   $();
806 
806   $();
807 
807   $();
808 
808   $();
809 
809   $();
810 
810   $();
811 
811   $();
812 
812   $();
813 
813   $();
814 
814   $();
815 
815   $();
816 
816   $();
817 
817   $();
818 
818   $();
819 
819   $();
820 
820   $();
821 
821   $();
822 
822   $();
823 
823   $();
824 
824   $();
825 
825   $();
826 
826   $();
827 
827   $();
828 
828   $();
829 
829   $();
830 
830   $();
831 
831   $();
832 
832   $();
833 
833   $();
834 
834   $();
835 
835   $();
836 
836   $();
837 
837   $();
838 
838   $();
839 
839   $();
840 
840   $();
841 
841   $();
842 
842   $();
843 
843   $();
844 
844   $();
845 
845   $();
846 
846   $();
847 
847   $();
848 
848   $();
849 
849   $();
850 
850   $();
851 
851   $();
852 
852   $();
853 
853   $();
854 
854   $();
855 
855   $();
856 
856   $();
857 
857   $();
858 
858   $();
859 
859   $();
860 
860   $();
861 
861   $();
862 
862   $();
863 
863   $();
864 
864   $();
865 
865   $();
866 
866   $();
867 
867   $();
868 
868   $();
869 
869   $();
870 
870   $();
871 
871   $();
872 
872   $();
873 
873   $();
874 
874   $();
875 
875   $();
876 
876   $();
877 
877   $();
878 
878   $();
879 
879   $();
880 
880   $();
881 
881   $();
882 
882   $();
883 
883   $();
884 
884   $();
885 
885   $();
886 
886   $();
887 
887   $();
888 
888   $();
889 
889   $();
890 
890   $();
891 
891   $();
892 
892   $();
893 
893   $();
894 
894   $();
895 
895   $();
896 
896   $();
897 
897   $();
898 
898   $();
899 
899   $();
900 
900   $();
901 
901   $();
902 
902   $();
903 
903   $();
904 
904   $();
905 
905   $();
906 
906   $();
907 
907   $();
908 
908   $();
909 
909   $();
910 
910   $();
911 
911   $();
912 
912   $();
913 
913   $();
914 
914   $();
915 
915   $();
916 
916   $();
917 
917   $();
918 
918   $();
919 
919   $();
920 
920   $();
921 
921   $();
922 
922   $();
923 
923   $();
924 
924   $();
925 
925   $();
926 
926   $();
927 
927   $();
928 
928   $();
929 
929   $();
930 
930   $();
931 
931   $();
932 
932   $();
933 
933   $();
934 
934   $();
935 
935   $();
936 
936   $();
937 
937   $();
938 
938   $();
939 
939   $();
940 
940   $();
941 
941   $();
942 
942   $();
943 
943   $();
944 
944   $();
945 
945   $();
946 
946   $();
947 
947   $();
948 
948   $();
949 
949   $();
950 
950   $();
951 
951   $();
952 
952   $();
953 
953   $();
954 
954   $();
955 
955   $();
956 
956   $();
957 
957   $();
958 
958   $();
959 
959   $();
960 
960   $();
961 
961   $();
962 
962   $();
963 
963   $();
964 
964   $();
965 
965   $();
966 
966   $();
967 
967   $();
968 
968   $();
969 
969   $();
970 
970   $();
971 
971   $();
972 
972   $();
973 
973   $();
974 
974   $();
975 
975   $();
976 
976   $();
977 
977   $();
978 
978   $();
979 
979   $();
980 
980   $();
981 
981   $();
982 
982   $();
983 
983   $();
984 
984   $();
985 
985   $();
986 
986   $();
987 
987   $();
988 
988   $();
989 
989   $();
990 
990   $();
991 
991   $();
992 
992   $();
993 
993   $();
994 
994   $();
995 
995   $();
996 
996   $();
997 
997   $();
998 
998   $();
999 
999   $();
1000 
1000   $();
1001 
1001   $();
1002 
1002   $();
1003 
1003   $();
1004 
1004   $();
1005 
1005   $();
1006 
1006   $();
1007 
1007   $();
1008 
1008   $();
1009 
1009   $();
1010 
1010   $();
1011 
1011   $();
1012 
1012   $();
1013 
1013   $();
1014 
1014   $();
1015 
1015   $();
1016 
1016   $();
1017 
1017   $();
1018 
1018   $();
1019 
1019   $();
1020 
1020   $();
1021 
1021   $();
1022 
1022   $();
1023 
1023   $();
1024 
1024   $();
1025 
1025   $();
1026 
1026   $();
1027 
1027   $();
1028 
1028   $();
1029 
1029   $();
1030 
1030   $();
1031 
1031   $();
1032 
1032   $();
1033 
1033   $();
1034 
1034   $();
1035 
1035   $();
1036 
1036   $();
1037 
1037   $();
1038 
1038   $();
1039 
1039   $();
1040 
1040   $();
1041 
1041   $();
1042 
1042   $();
1043 
1043   $();
1044 
1044   $();
1045 
1045   $();
1046 
1046   $();
1047 
1047   $();
1048 
1048   $();
1049 
1049   $();
1050 
1050   $();
1051 
1051   $();
1052 
1052   $();
1053 
1053   $();
1054 
1054   $();
1055 
1055   $();
1056 
1056   $();
1057 
1057   $();
1058 
1058   $();
1059 
1059   $();
1060 
1060   $();
1061 
1061   $();
1062 
1062   $();
1063 
1063   $();
1064 
1064   $();
1065 
1065   $();
1066 
1066   $();
1067 
1067   $();
1068 
1068   $();
1069 
1069   $();
1070 
1070   $();
1071 
1071   $();
1072 
1072   $();
1073 
1073   $();
1074 
1074   $();
1075 
1075   $();
1076 
1076   $();
1077 
1077   $();
1078 
1078   $();
1079 
1079   $();
1080 
1080   $();
1081 
1081   $();
1082 
1082   $();
1083 
1083   $();
1084 
1084   $();
1085 
1085   $();
1086 
1086   $();
1087 
1087   $();
1088 
1088   $();
1089 
1089   $();
1090 
1090   $();
1091 
1091   $();
1092 
1092   $();
1093 
1093   $();
1094 
1094   $();
1095 
1095   $();
1096 
1096   $();
1097 
1097   $();
1098 
1098   $();
1099 
1099   $();
1100 
1100   $();
1101 
1101   $();
1102 
1102   $();
1103 
1103   $();
1104 
1104   $();
1105 
1105   $();
1106 
1106   $();
1107 
1107   $();
1108 
1108   $();
1109 
1109   $();
1110 
1110   $();
1111 
1111   $();
1112 
1112   $();
1113 
1113   $();
1114 
1114   $();
1115 
1115   $();
1116 
1116   $();
1117 
1117   $();
1118 
1118   $();
1119 
1119   $();
1120 
1120   $();
1121 
1121   $();
1122 
1122   $();
1123 
1123   $();
1124 
1124   $();
1125 
1125   $();
1126 
1126   $();
1127 
1127   $();
1128 
1128   $();
1129 
1129   $();
1130 
1130   $();
1131 
1131   $();
1132 
1132   $();
1133 
1133   $();
1134 
1134   $();
1135 
1135   $();
1136 
1136   $();
1137 
1137   $();
1138 
1138   $();
1139 
1139   $();
1140 
1140   $();
1141 
1141   $();
1142 
1142   $();
1143 
1143   $();
1144 
1144   $();
1145 
1145   $();
1146 
1146   $();
1147 
1147   $();
1148 
1148   $();
1149 
1149   $();
1150 
1150   $();
1151 
1151   $();
1152 
1152   $();
1153 
1153   $();
1154 
1154   $();
1155 
1155   $();
1156 
1156   $();
1157 
1157   $();
1158 
1158   $();
1159 
1159   $();
1160 
1160   $();
1161 
1161   $();
1162 
1162   $();
1163 
1163   $();
1164 
1164   $();
1165 
1165   $();
1166 
1166   $();
1167 
1167   $();
1168 
1168   $();
1169 
1169   $();
1170 
1170   $();
1171 
1171   $();
1172 
1172   $();
1173 
1173   $();
1174 
1174   $();
1175 
1175   $();
1176 
1176   $();
1177 
1177   $();
1178 
1178   $();
1179 
1179   $();
1180 
1180   $();
1181 
1181   $();
1182 
1182   $();
1183 
1183   $();
1184 
1184   $();
1185 
1185   $();
1186 
1186   $();
1187 
1187   $();
1188 
1188   $();
1189 
1189   $();
1190 
1190   $();
1191 
1191   $();
1192 
1192   $();
1193 
1193   $();
1194 
1194   $();
1195 
1195   $();
1196 
1196   $();
1197 
1197   $();
1198 
1198   $();
1199 
1199   $();
1200 
1200   $();
1201 
1201   $();
1202 
1202   $();
1203 
1203   $();
1204 
1204   $();
1205 
1205   $();
1206 
1206   $();
1207 
1207   $();
1208 
1208   $();
1209 
1209   $();
1210 
1210   $();
1211 
1211   $();
1212 
1212   $();
1213 
1213   $();
1214 
1214   $();
1215 
1215   $();
1216 
1216   $();
1217 
1217   $();
1218 
1218   $();
1219 
1219   $();
1220 
1220   $();
1221 
1221   $();
1222 
1222   $();
1223 
1223   $();
1224 
1224   $();
1225 
1225   $();
1226 
1226   $();
1227 
1227   $();
1228 
1228   $();
1229 
1229   $();
1230 
1230   $();
1231 
1231   $();
1232 
1232   $();
1233 
1233   $();
1234 
1234   $();
1235 
1235   $();
1236 
1236   $();
1237 
1237   $();
1238 
1238   $();
1239 
1239
```

The screenshot shows the Microsoft Teams interface. On the left is the sidebar with various icons for Activity, Chat, Teams, Calendar, Calls, Files, Power Apps, App Studio, and Contoso Teams. The main area shows a team named "General" under "Your teams". A tab labeled "Full Stack Dev" is selected. Below it, a card for a "Full Stack Developer" position is displayed, listing the hiring manager as Henry Wickham, ID B6626FOC, and location as Chicago, IL. It shows status counts: applied (0), screening (0), interviewing (1), and offered (1). A red arrow points from the text "This will open a Task Module, which is a pop-up within Teams that iFrames a web-application (similar to how Tabs work!)" to the applicant list below. The applicants section shows two candidates: Scotty Cothran and Donna Delaney. Scotty Cothran is highlighted with a red box.

13. 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...

The screenshot shows the Microsoft Teams interface with the same sidebar and "General" team selected. A task module is open in the foreground, titled "Talent Management App - Feedback for Scotty Cothran". It displays feedback for "Scotty Cothran" applying for "Full Stack Developer". The module shows "Conducted (2)" interviews. A table lists the interviewees:

Name	Interview date	Feedback
Ron Withrow	9/11/2018	Hire
Dustin Leake	1/10/2019	N/A

14. 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.

```

280     var firstName = person.name.substring(0, person.name.indexOf(' '));
281     nameAddition = '<div class="hrImage">' +
282       + '<span class="small-font-size" title=' + person.name + '>' + firstName + '</span>' +
283       + '</div>';
284   }
285
286   var centerImage = centerImage ? 'text-align: center;' : '';
287   return [
288     + '<div class=""' + extraClass + '>' +
289       + '<div style="' + centerImage + '>' +
290         + '<img alt=' + person.name + '" class="ms-Image itemImage">' +
291           + '<img alt=' + person.name + '" class="ms-Image-image is-loaded ms-Image-image--cover ms-Image-image--portrait ms-Image-image--fadeIn" src=' +
292             + nameAddition +
293             + '</div>' +
294             + '</div>';
295   ];
296
297   function openDetails(candidateId, candidateName) {
298     microsoftTeams.tasks.startTask({
299       url: window.location.origin + '/StaticViews/CandidateFeedback.html?candidateId=' + encodeURIComponent(candidateId),
300       title: "Feedback for " + candidateName,
301       width: "large",
302       height: "large"
303     });
304
305   }
306
307   $async () => {
308     await this.handlePageLoad();
309     this.loadPage();
310   }
311
312   </script>
313   </body>
314   <div class="ms-Grid-row jobTitleHeader">
315     ...

```

15. 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**.

Microsoft Teams

Teams

Your teams

- U.S. Sales
- Sales and Marketing
- Retail
- Mark 8 Project Team
- Digital Initiative Public Relations
- SOC Team
- Contoso
- Trade Show Workers
- Contoso HR
- Internal IT

General Posts Files Wiki Full Stack Dev +

Full Stack Developer

Hiring Manager: Henry Wickham
ID: B6626FOC
Location: Chicago, IL

Job details

We are looking for a Full Stack Developer to produce and implement functional software solutions. You will work with management to define the software specifications and then implement them.

Hiring Manager Hiring Team

Henry Ron Dustin

Applicants 2 candidates

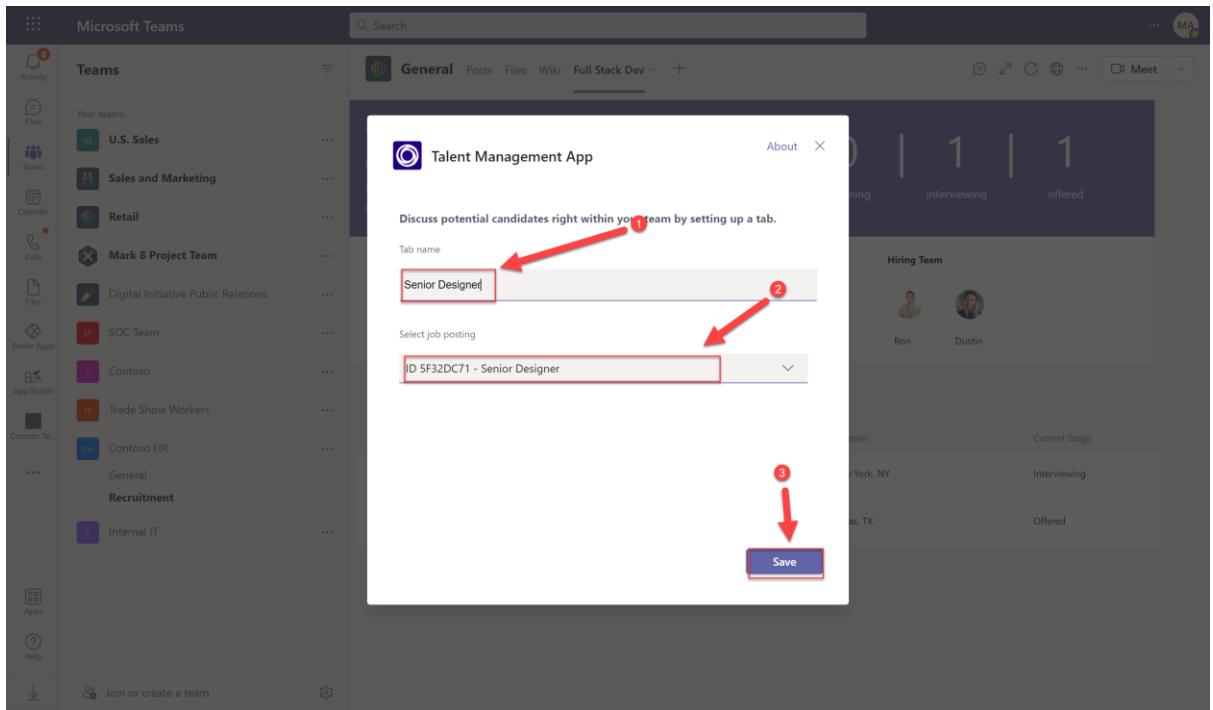
Name	Modified	Location	Current Stage
Scotty Cothran	Yesterday	New York, NY	Interviewing
Donna Delaney	Yesterday	Dallas, TX	Offered

Settings

1

2

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



17. 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.

Name	Modified	Location	Current Stage
Bart Fredrick	Yesterday	New York, NY	Applied
Wallace Santoro	Yesterday	New York, NY	Offered
Natalia Gill	Yesterday	Miami, FL	Applied

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 ChatBot that supports Personal, Group and Channel scopes

In this step we will modify the application manifest to add our bot to the app. Importantly, we need to sideload the app, that contains the bot, into the Organisational app store, or the Teams App ID is randomised, which will then cause issues when we want to initiate a Task Module from an adaptive card (the SSO won't work properly if the Teams app ID is randomised).

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.

Name	Version number	App ID	Updated
(app name)	(version)	f5d45ea-0e19-49f3-8cb0-a0d7d9b9aa7e	February 26, 2021
(app name)	(version)	16d25ba5-4c76-4546-980f-7b16c85559c7	May 27, 2021
DataverseIntegrationSample	1.0.0	40d3c5d1-97a7-4fe2-0e83-917f9286f9d7	September 13, 2021
Debug Bot	1.0.0	245394e-cf91-4cc7-b561-1b15a043b333	June 29, 2021
DebugBot2	1.0.0	8d89c545-cff6-4e66-b50d-ba5cfcd4d5ea	June 30, 2021
DeepLinkSample	1.0.0	c8c75fd2-773e-41f3-861f-73d0185f7a1	July 15, 2021
Meeting App Test	1.0.0	4ae1da46-9eca-4752-8bcc-8c9bb4bf0cd	June 08, 2021
My Test Bot	1.0.0	a1c8e15-26fc-47db-8f80-b1be5da358c6	May 06, 2021
ReplBotApp v2	1.0.0	94edf74a-33fb-40b5-9d57-8ec1c7119fc3	June 03, 2021
Sample	1.0.0	673f334-0d70-4df7-8ee-ad34b36131c1	March 04, 2021
Sample App	1.0.0	c360ca99-e0ce-49b5-ab47-e463743b93a	August 26, 2021
Talent Management App	1.0.0	82f511d-e5ef-4370-b7d2-8dc8eff183e3	October 08, 2021
Teams-Sample-TM V2	1.0.0	cea7737f-d317-44d5-9eca-b0a1962c6f02	October 05, 2021
Teams-Sample-TalentManagement	1.0.0	cea7737f-d317-44d5-9eca-b0a1962c6f02	September 30, 2021

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.

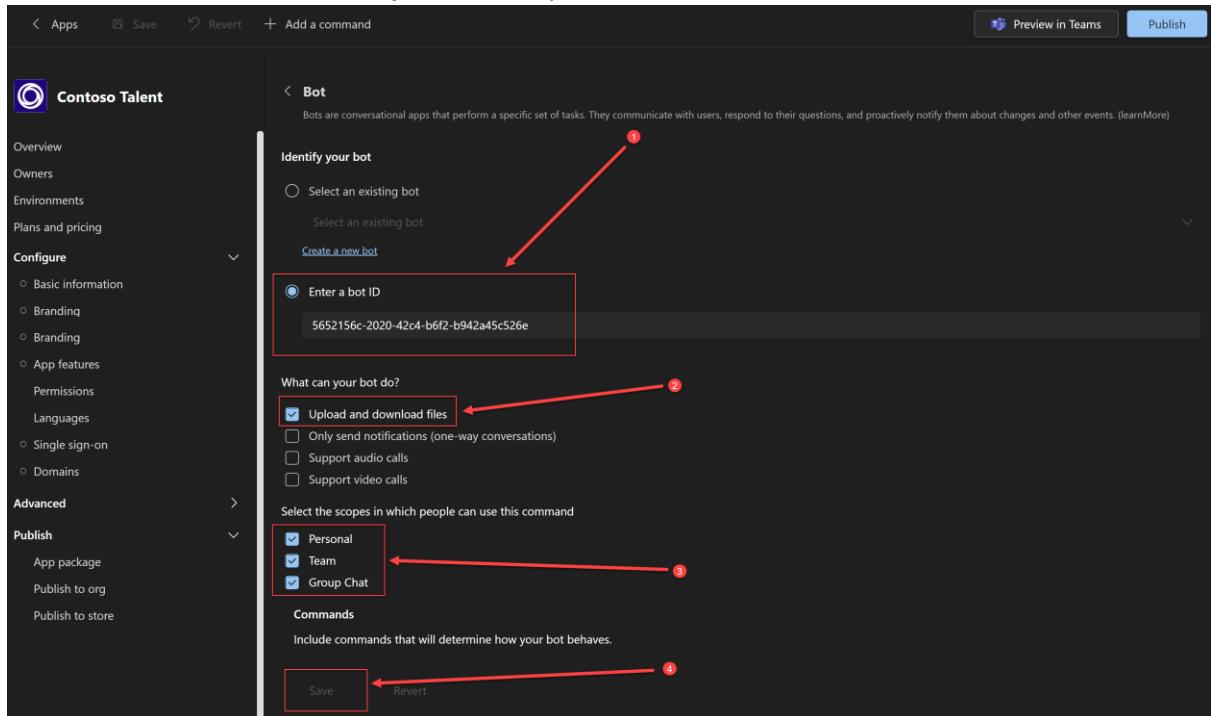
App features

These are the Teams features you can include in your app. Add one or more features depending on your app's use cases. [Learn more](#)

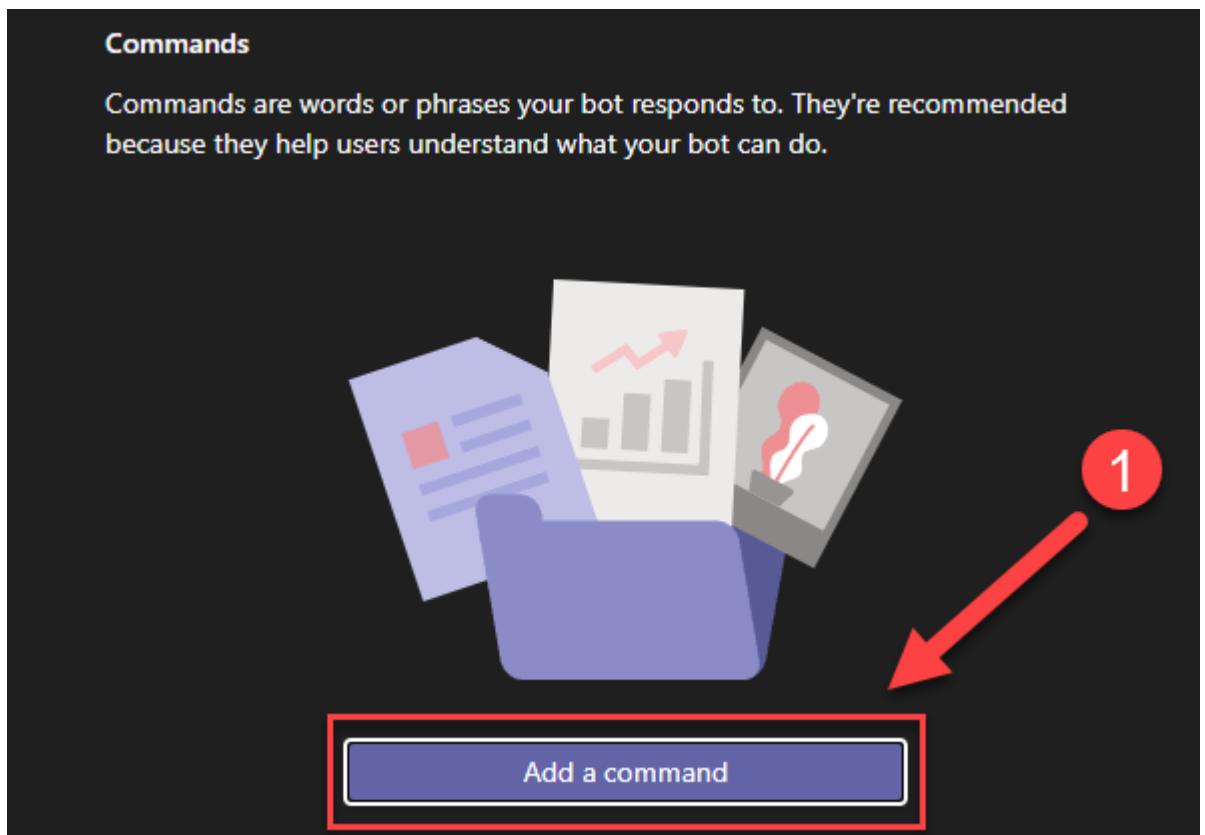
Select a feature to add
Personal app A dedicated workspace or bot to help individual users focus on their own tasks or view activities important to them. Learn more
Group and channel app A space to display hosted app experiences (such as a list or dashboard) in team channels and group chats. Learn more
Bot A conversational UI that can perform a set of tasks, reply to questions, and proactively send notifications. Learn more
Connector A way to automatically send notifications and messages from your app to a channel. Learn more
Messaging extension A shortcut for inserting your app's content or acting on a message without navigating away from a conversation. Learn more
Meeting extension Options for integrating your app with the Teams meeting experience, including the meeting stage and chat. Learn more
Scene A custom virtual scene people can use in their Teams Together mode meetings.
Activity feed notification Keep users informed and engaged with app notifications in the activity feed. Learn more

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 **MicrosoftAppId** from the App Settings in Visual

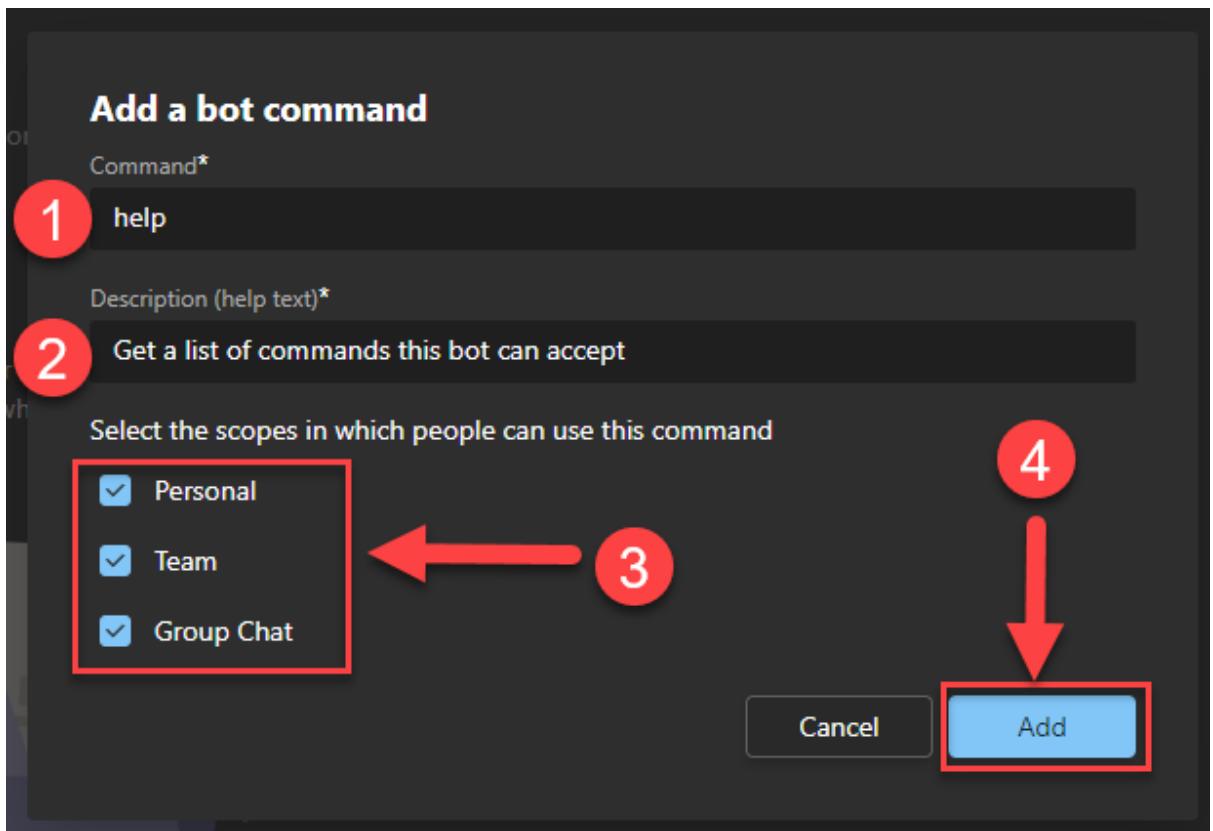
Studio. The Microsoft App Id (Client ID in Azure AD App Registration) and bot ID are the same, as there is a 1-2-1 mapping between Bots and Azure AD App Registrations. Select **Upload and download files** as this bot is able to work with files within Teams. Under scopes, select **Personal, Team and Group Chat**. Finally, click **Save** to save the manifest.



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



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



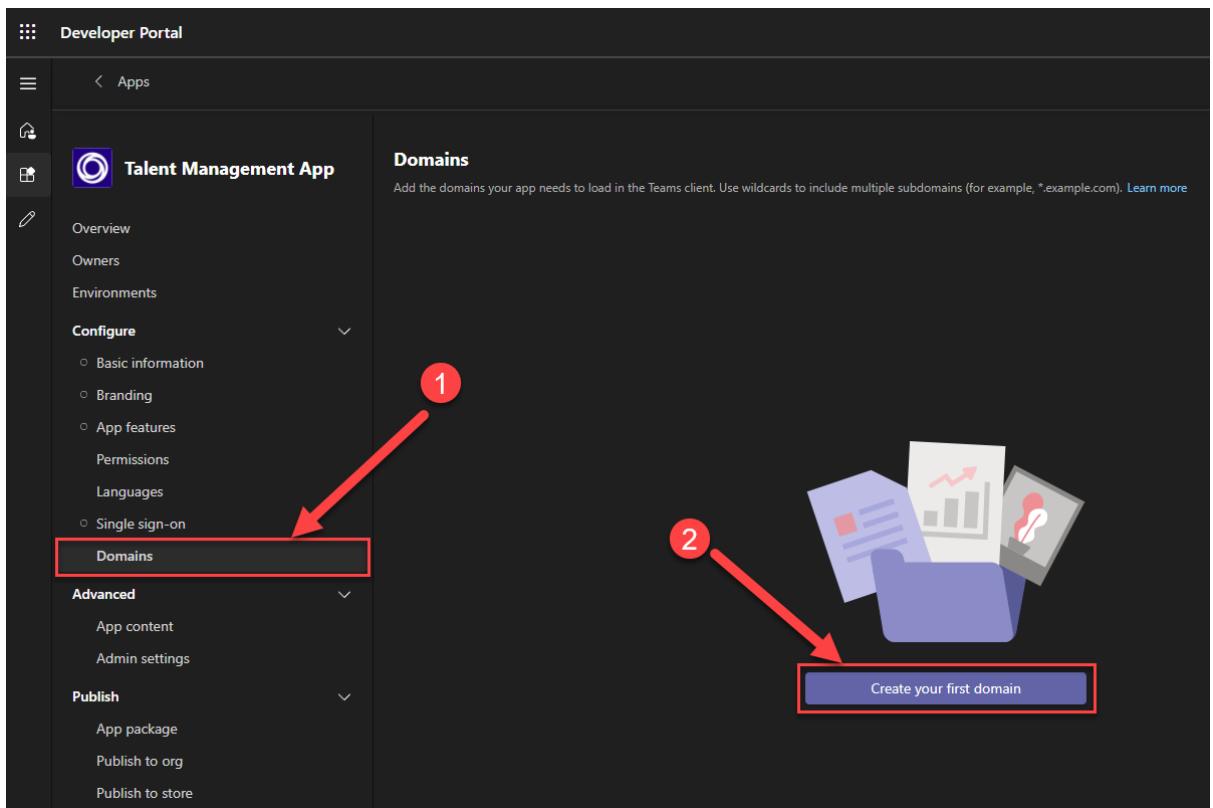
6. Repeat this process for all the commands the bot is able to perform as shown below.
NOTE: Ensure that you click **Save** after each command to ensure that you don't lose any configuration if the edit breaks!

Commands

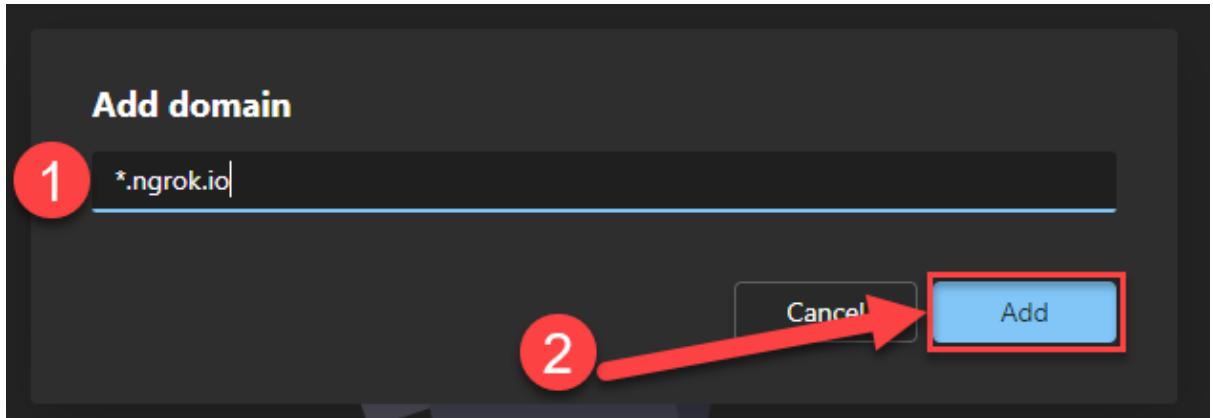
Commands are words or phrases your bot responds to. They're recommended because they help users understand what your bot can do.

Command	Description	Scope	...
help	Get a list of commands this bot can accept	group, personal, team	...
candidate details	Show details about a candidate, for example: candidate details Bart Fredrick	group, personal, team	...
summary	Show summary about a candidate, for example: summary Bart Fredrick	group, personal, team	...
top candidates	Show top candidates for a Position ID, for example: top candidates 10CD3166	group, personal, team	...
new job posting	Create a new job posting	group, personal, team	...
open positions	List all your open positions	group, personal, team	...
+ Add a command			
Save			
Revert			

7. Next we need to define which domains are allowed to be opened within the iFrame, this needs to include Bot Service domains (for fallback authentication) and the domain your local app instance is using (the NGROK domain). Select **Domains** from the left menu and then click **Create your first domain**.



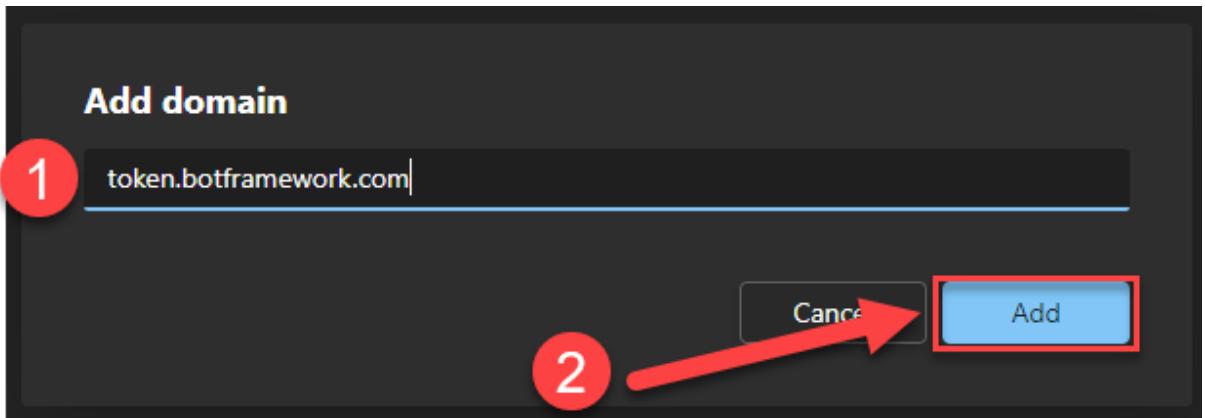
8. There are two domains we will need to add, the first will be your API service. As we are using ngrok to tunnel, add the wildcard domain `*.ngrok.io` and click **Save**.



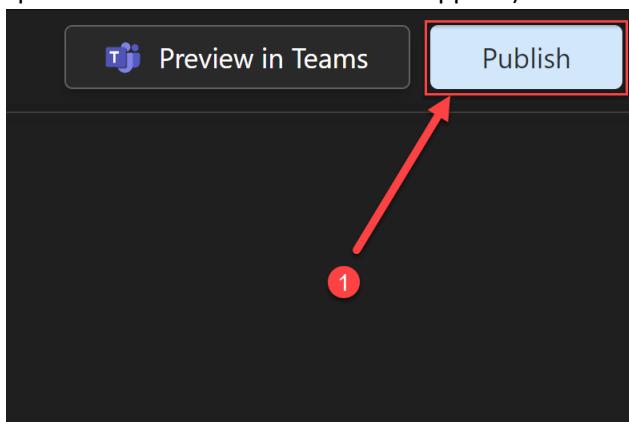
9. You will now see that your domain appears in the domains list. Click **Add a domain** and repeat this process.

Domains	
Add the domains your app needs to load in the Teams client. Use wildcards to include multiple subdomains (for example, *.example.com). Learn more	
Domains	Source
<code>*.ngrok.io</code>	Additional
+ Add a domain	

10. We will now add the domain that bot framework uses for fallback authentication, with our bot, if SSO fails. Enter `token.botframework.com` and click **Add**.



11. Now that the bot has been configured completely, let's test it out! Instead of sideloading the app for a specific user, this time we need to publish the app to the organisation, as this will ensure that the Teams App ID, defined in the manifest, does not change (sideloaded apps for specific users have random Teams App IDs). Click **Publish**.



12. Click **Download the app package**, this will save the app manifest ZIP into your Downloads folder.

Publish your app

 **Download the app package**
Download a copy of your app package, which is specific to your selected environment. Use the package to upload your app in Teams or publish later.

 **Publish to your org**
Submit a request to your IT admin to publish your app. It will appear in the Built for your org section of the store once it's approved.

 **Publish to the Teams store**
Make your app available to Teams users everywhere. This option requires Microsoft approval.

13. If the App package has errors, they will display here. Some errors must be fixed before you can install the app into Teams (note, in the screenshot below, all 3 errors can be safely ignored), **review and fix the errors**, click **Download**.

App package has errors

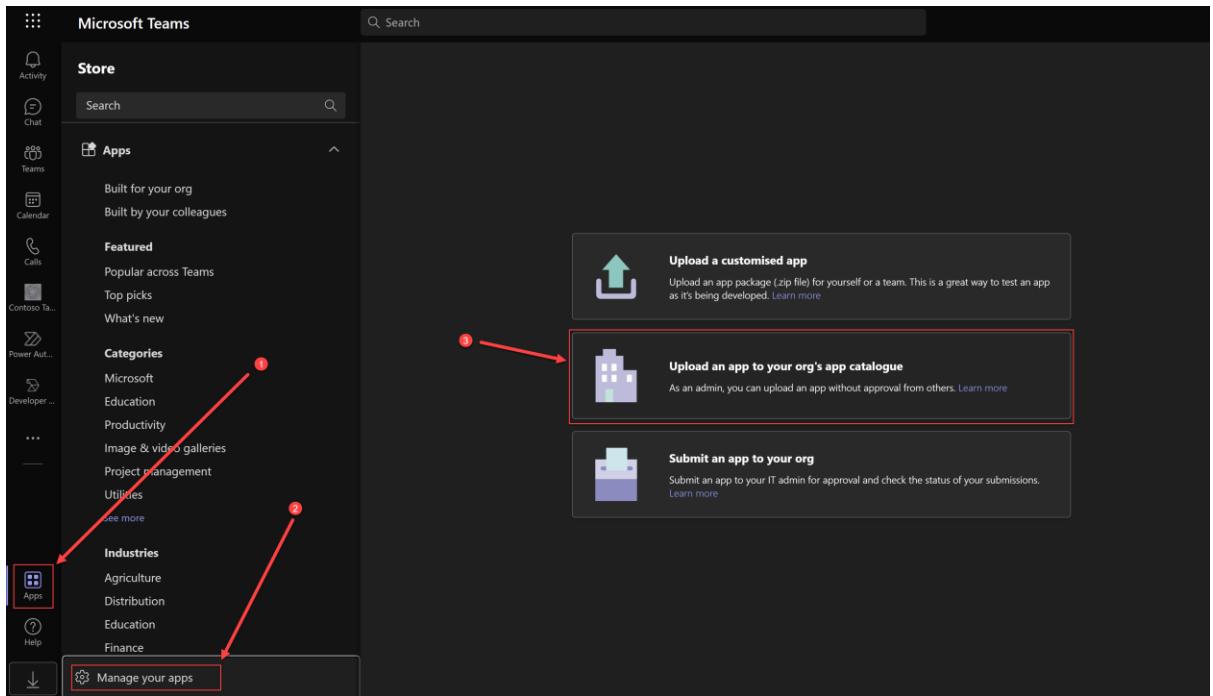
You can still download a copy of the app package, but you must resolve these errors before you can upload or publish the app to Teams.

 InvalidOutlineIconTransparency
Outline icon is not transparent. It's Alpha,R,G,B: 255,224,224,234

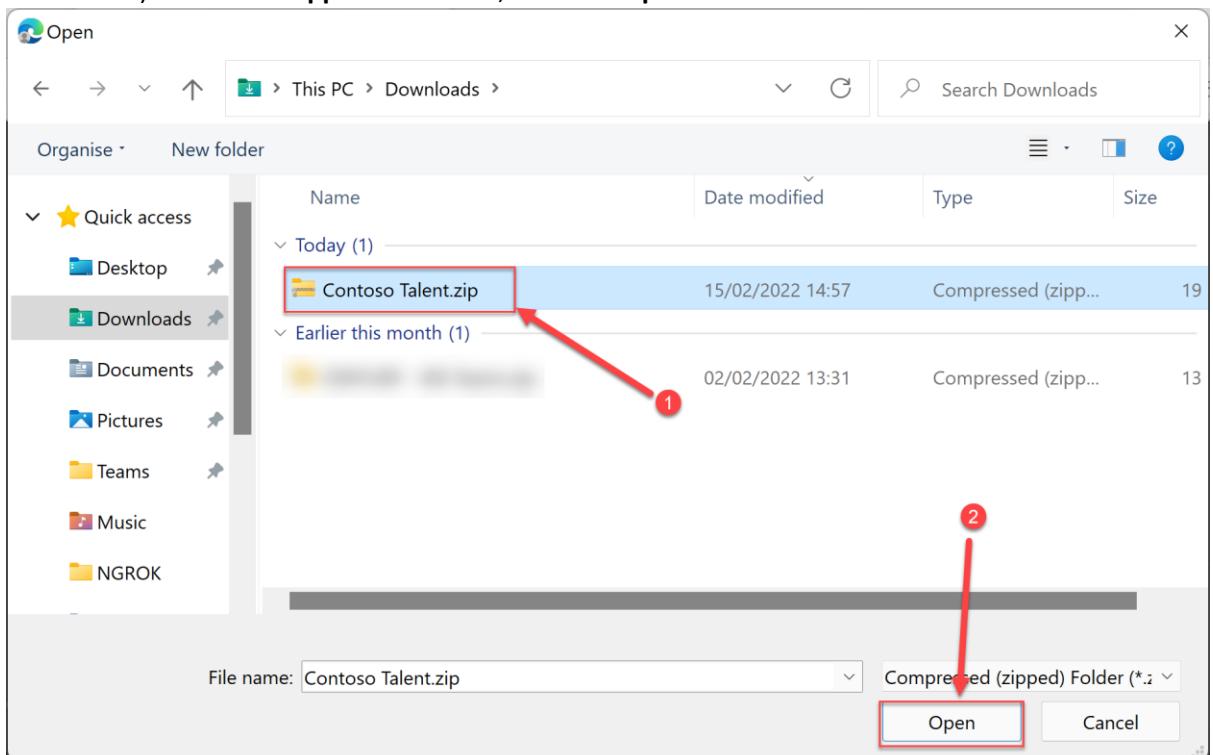
 ReservedStaticTabNameShouldBeNull
Reserved tab "Name" property should not be specified

 ReservedStaticTabNameShouldBeNull
Reserved tab "Name" property should not be specified

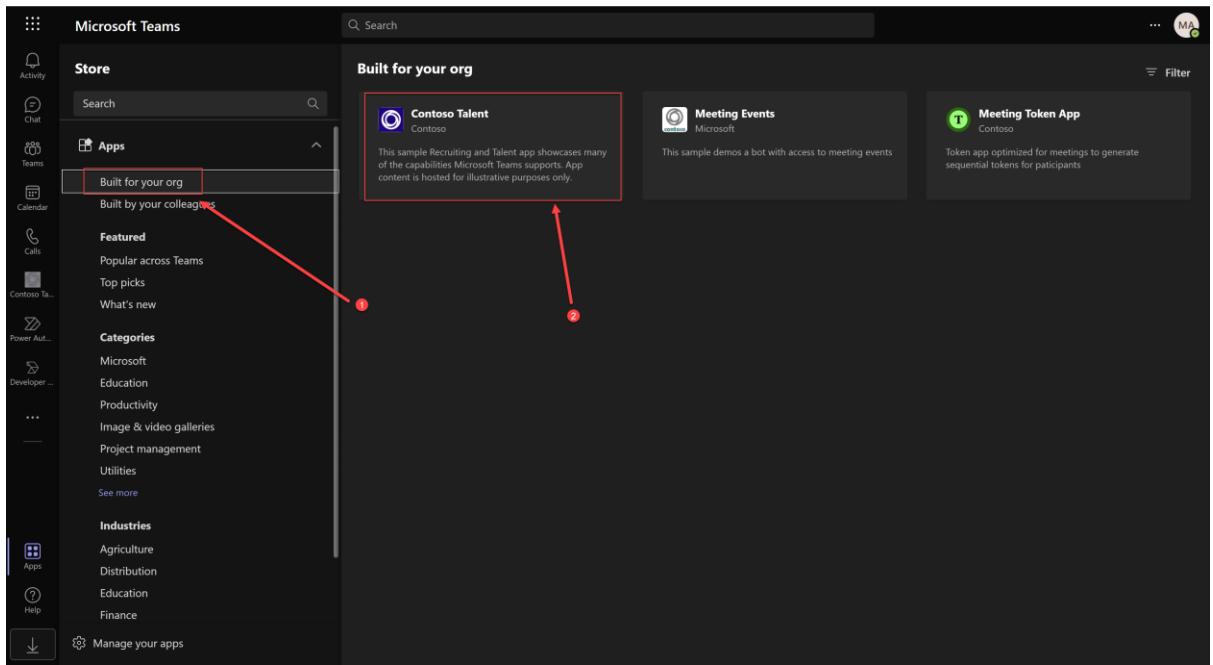
14. Go back into Teams, select **Apps**, then click **Manage your apps**, click **Upload an app to your org's app catalogue**. *Note: this option will only exist if you are logged into Teams as a Global Administrator, if you are not, log-out and log-in as a Global Administrator.*



15. Navigate to the directory (folder) where the Teams App Manifest was saved (probably Downloads). Select the **App Manifest ZIP**, and click **Open**.



16. Click **Built for your org**, and select the **Contoso Talent** Teams app.



17. Click **Add** to install the latest version of this application, which includes the Bot.

Talent Management App

This app is not from your organization's app catalog or the Teams store. Do not proceed to add the app unless you are testing it in development or trust the person who shared it with you.

Add

About

Permissions

Bots
Chat with the app to ask questions and find info

Personal app
Keep track of important content and info

Created by: [Contoso](#)
Version 1.0.0

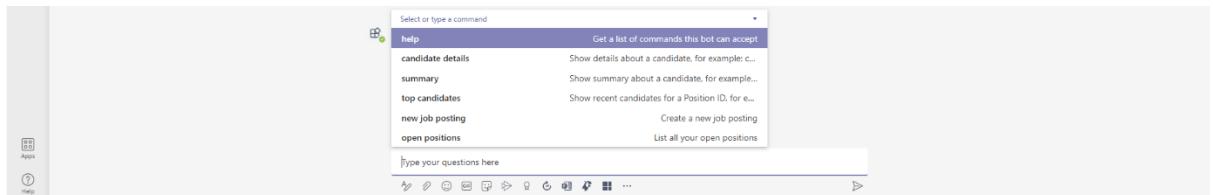
Permissions

This app will have permission to:

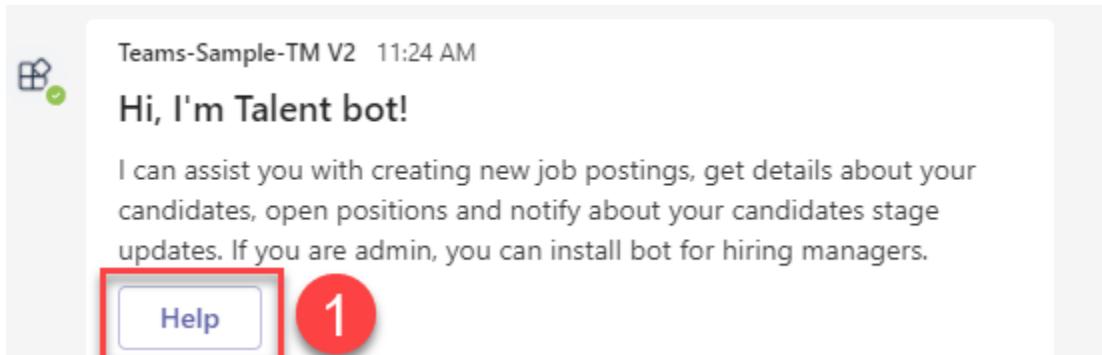
- Receive messages and data that I provide to it.
- Send me messages and notifications.
- Access my profile information such as my name, email address, company name, and preferred language.
- Receive messages and data that team or chat members provide to it in a channel or chat.
- Send messages and notifications in a channel or chat.
- Access information from this team or chat such as team or chat name, channel list and roster (including team or chat member's names and email addresses) - and use this to contact them.

By using Talent Management App, you agree to the [privacy policy](#) and [terms of use](#).

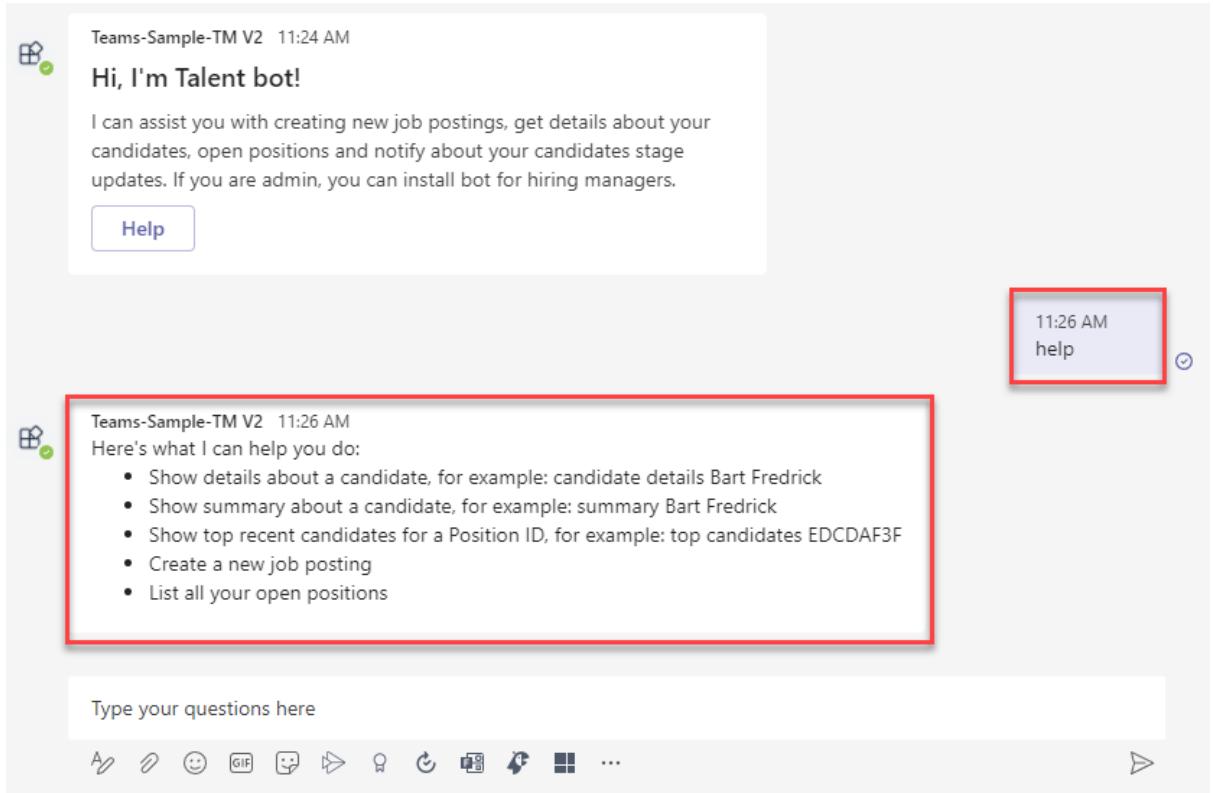
18. When you highlight the Compose box, you will now see that you are presented with a list of commands available through the bot. You can either click a command or type the name. Note: that the bot will have also added a welcome message but depending on the size of your window it might be hidden by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



- 19.** 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. **Note: This is using the ‘imBack’ (instant message back to bot) action type, from within the Adaptive Card, to send the ‘help’ message back to the bot when you click on the button.**



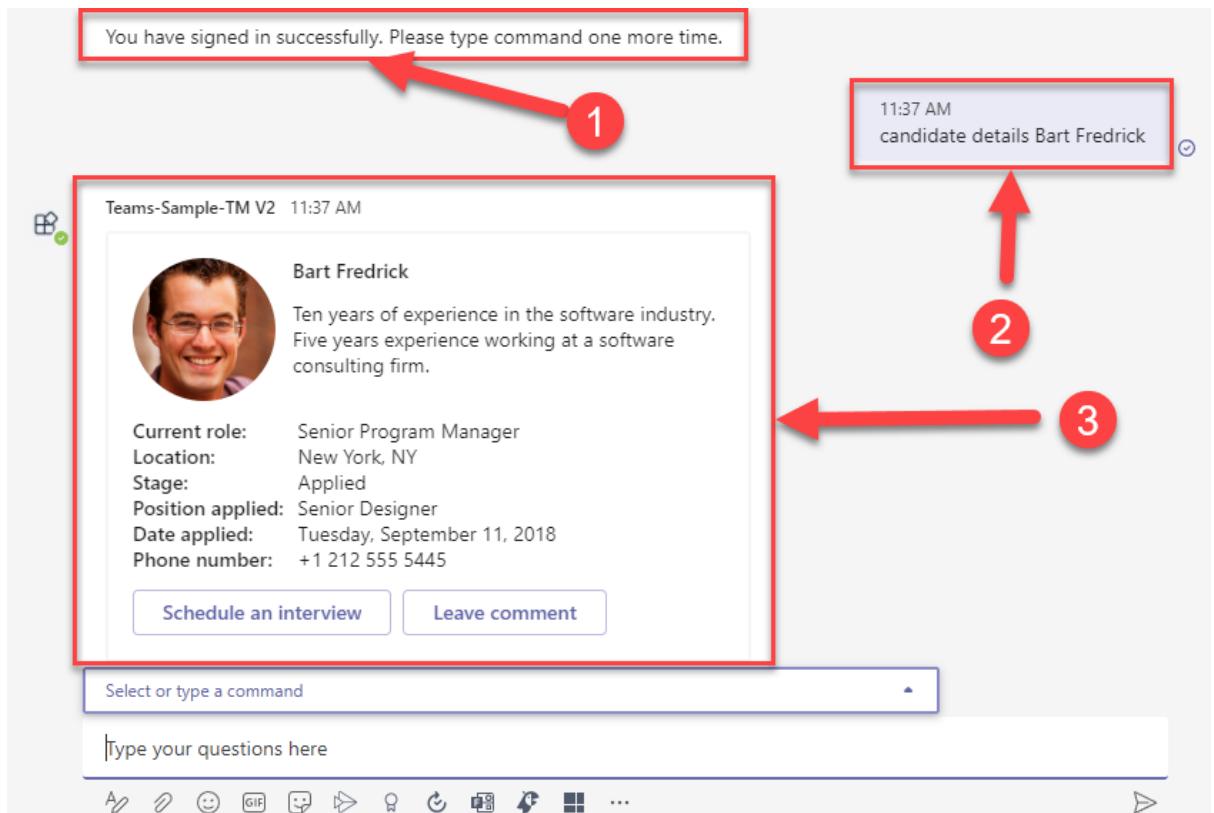
- 20.** Once the command has been received by the bot it will respond with a friendly help message listing some of the common commands.



- 21.** 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, and send it to the bot.



22. 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.



23. 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.

You have signed in successfully. Please type command one more time.

11:37 AM
candidate details Bart Fredrick

Teams-Sample-TM V2 11:37 AM



Bart Fredrick
Ten years of experience in the software industry.
Five years experience working at a software consulting firm.

Current role: Senior Program Manager
Location: New York, NY
Stage: Applied
Position applied: Senior Designer
Date applied: Tuesday, September 11, 2018
Phone number: +1 212 555 5445

Schedule an interview Leave comment

Select or type a command

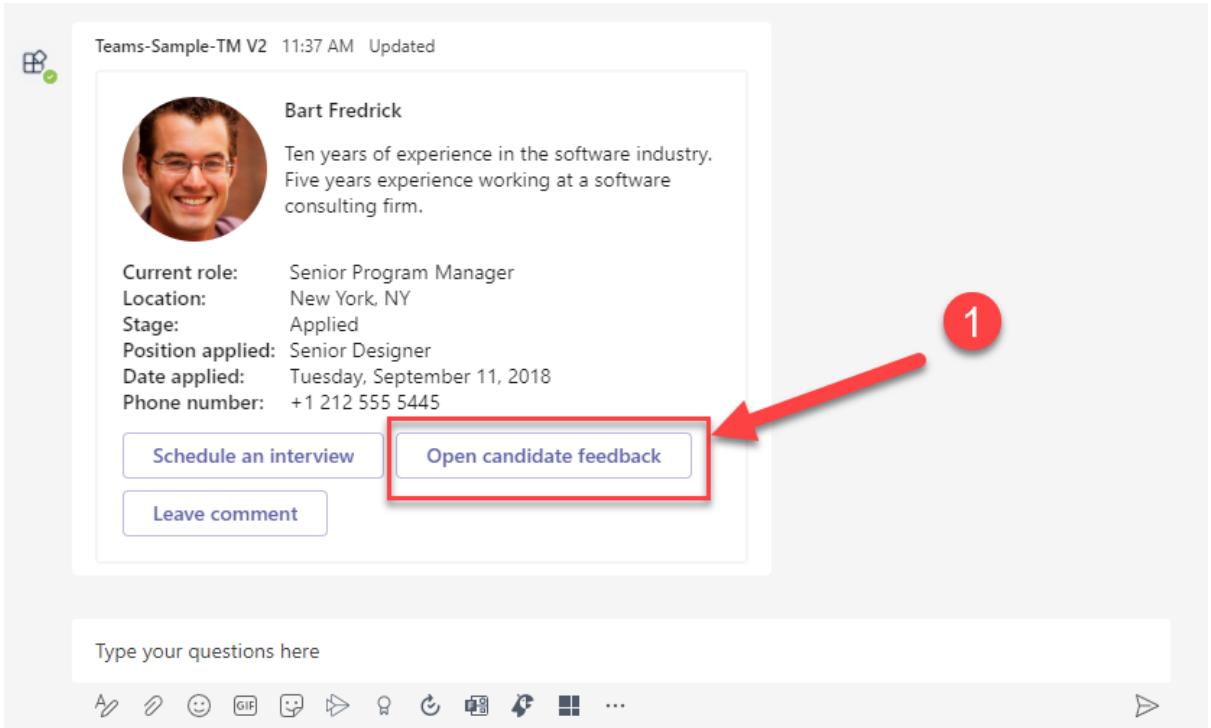
Type your questions here

Attachment icons: GIF, emoji, file, etc.

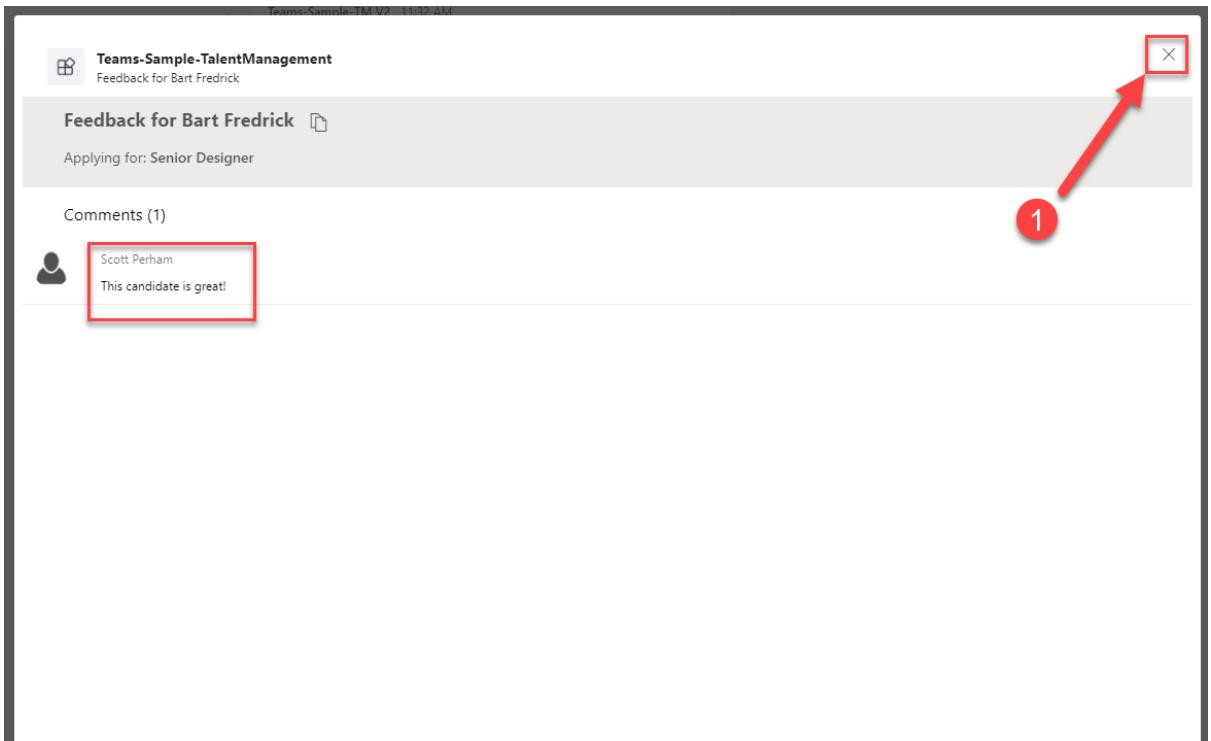
1

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

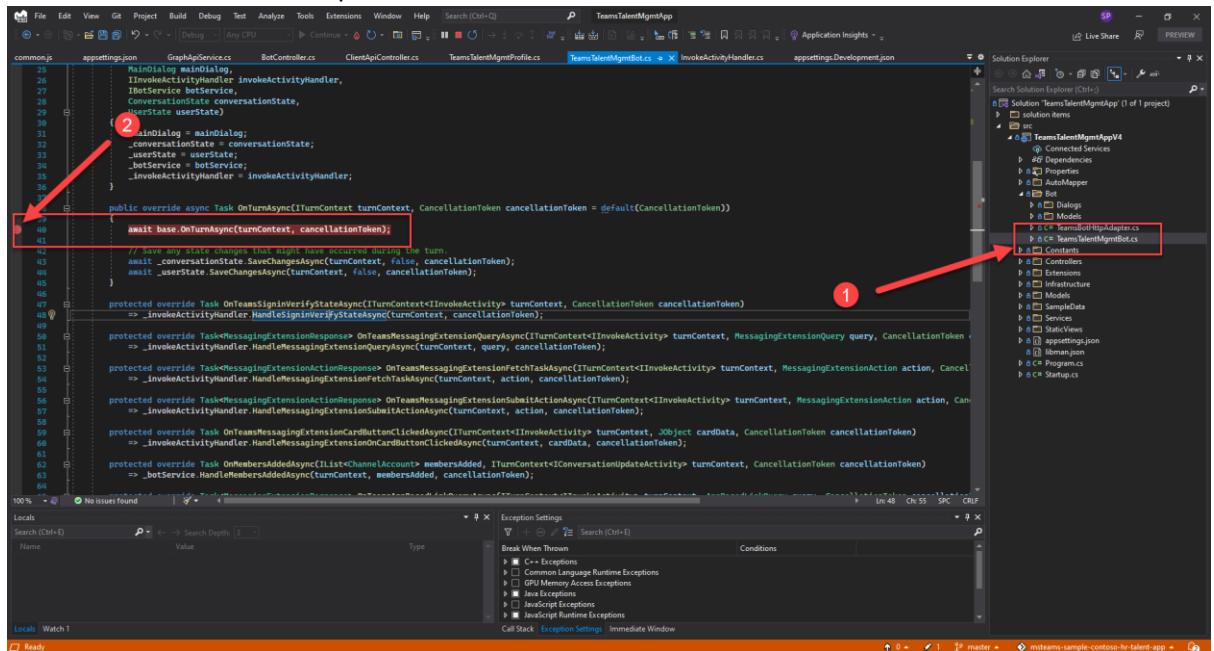
25. This will resend the candidate adaptive card to Teams and because we now have some feedback, and additional (conditional) button will be shown. Click **Open candidate feedback** to see a summary of all the feedback left for the candidate.



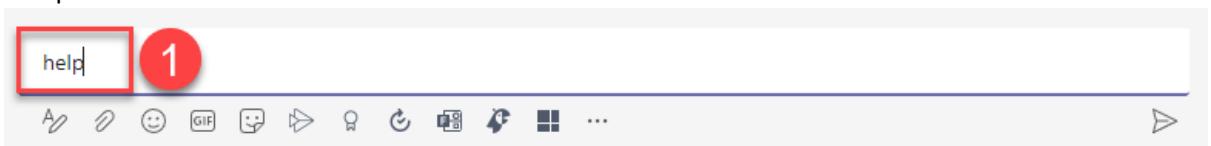
26. You will see your feedback in the list! **NOTE: If the feedback does not appear, this is likely because the Teams SSO within the Task Module has failed. In our experience this is usually because the Teams App ID (associated with the invoke task module button) that is being passed down in the Adaptive Card (from the code), does not match the Teams App ID in your Teams App manifest. If this is the case, you will see 401 unauthorised displayed in the network tab of the browsers developer tools, when the task module is initialised and it tries to pull data from the Client APIs. In most circumstances this mismatch is because the Teams App hasn't been installed into the Organisational App Store, and instead, has been sideloaded for a specific user. When Teams apps are sideloaded for a specific user, the Teams App ID is randomised. To fix this, go back to step 11 in this section and install the app into the Teams Organisation App Store, and try again.** Once you are done click the X in the corner to close the Task Module.



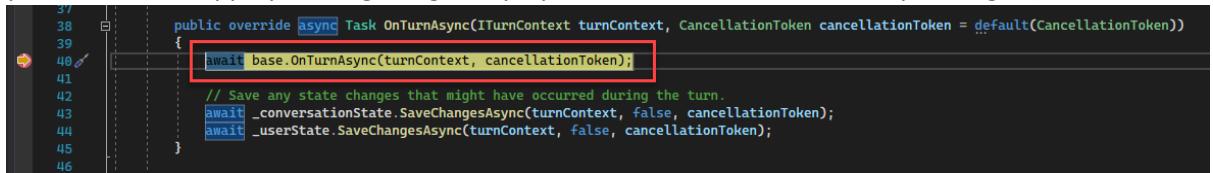
27. We will now have a look behind the scenes and see what happens when commands are sent from Teams to your bot. Open Visual Studio and navigate to the TeamsTalentMgmtBot.cs file. Then put a breakpoint on line 40. (Breakpoint can either be set by clicking the leftmost gutter on the appropriate line,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



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



29. 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).

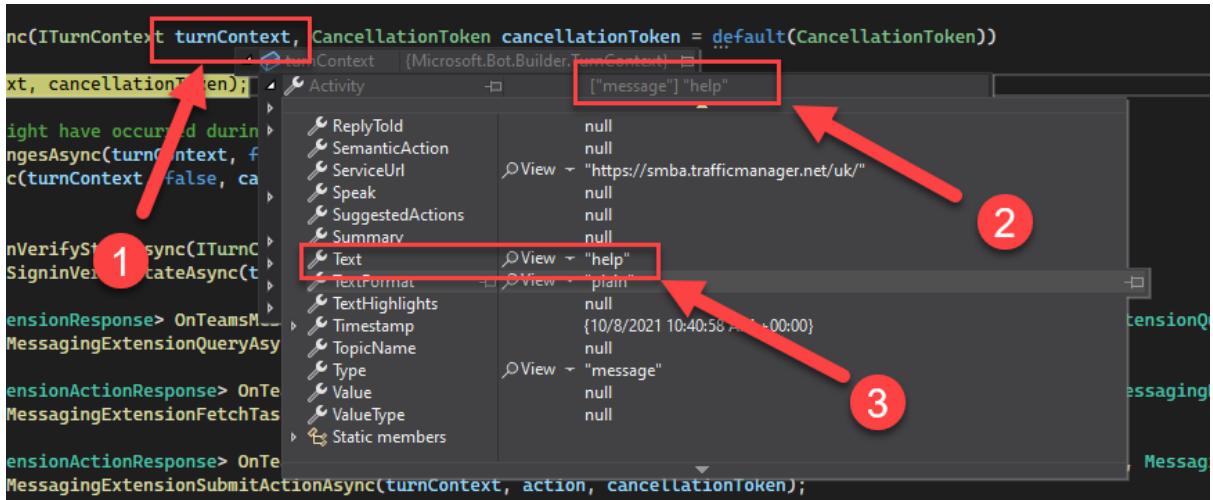


```

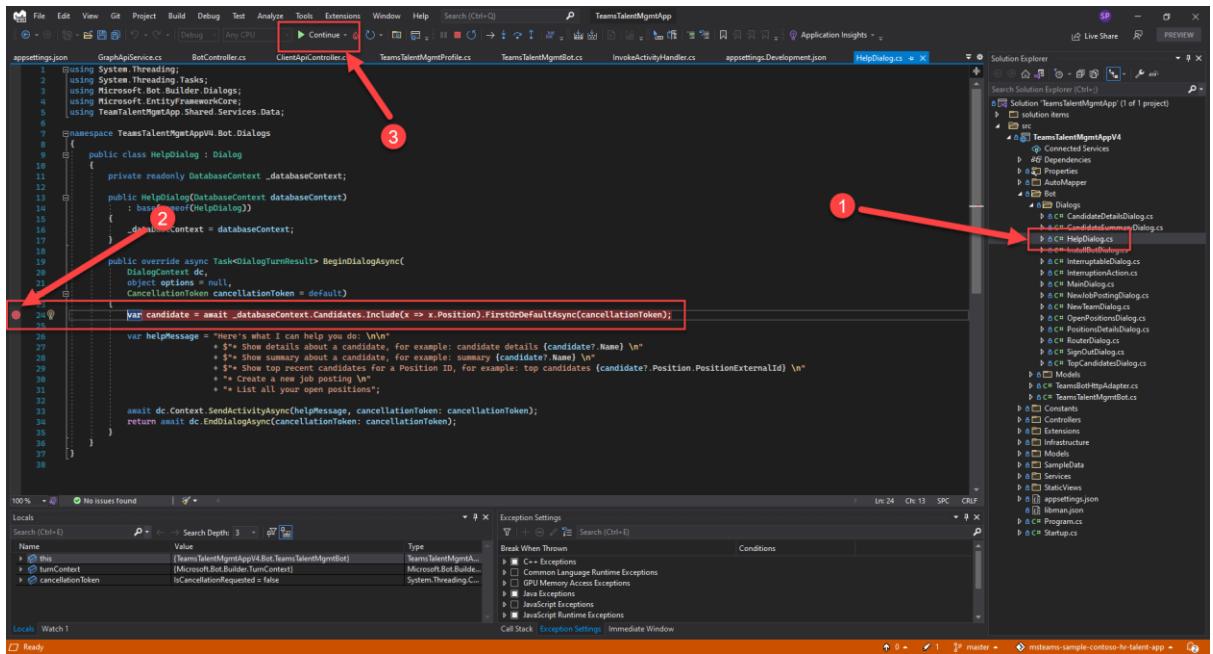
37
38     public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken = default(CancellationToken))
39 {
40     await base.OnTurnAsync(turnContext, cancellationToken);
41
42     // Save any state changes that might have occurred during the turn.
43     await _conversationState.SaveChangesAsync(turnContext, false, cancellationToken);
44     await _userState.SaveChangesAsync(turnContext, false, cancellationToken);
45 }
46

```

30. 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.



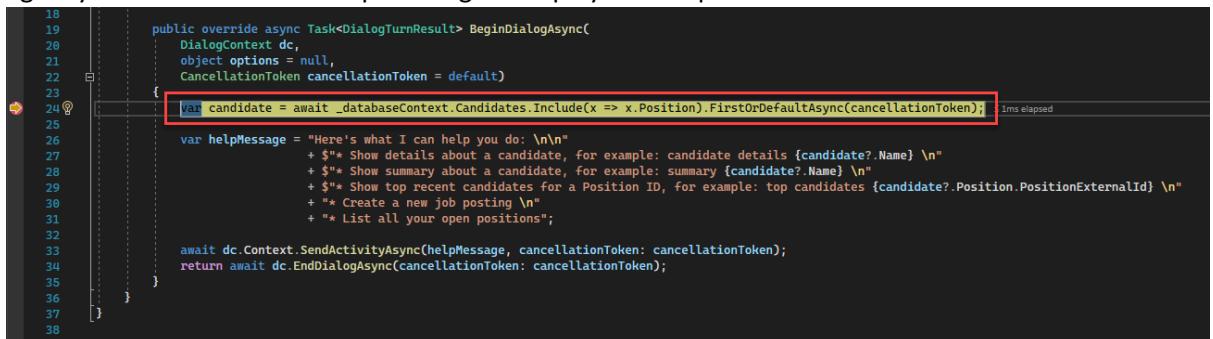
31. Another concept in bot framework is that of Dialogs. A dialog represents a part of a conversation. This could be a single text response or a conversation that spans multiple turns. It allows us to encapsulate the functionality of a particular conversation. One such dialog is the HelpDialog. Open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



32. 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.



33. 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.

```

public async Task<InvokeResponse> HandleSignInVerifyStateAsync(ITurnContext<ITurnContext<ActivityValue>> turnContext, CancellationToken cancellationToken)
{
    var token = ((TurnContext<ITurnContext<ActivityValue>>)turnContext).Activity.Value.ToString("token");
    if (token != null && !string.IsNullOrEmpty(token))
    {
        await _tokenProvider.SetTokenAsync(token, turnContext, cancellationToken);
        await turnContext.SendActivityAsync("You have signed in successfully. Please type command one more time.", cancellationToken);
    }
    await _conversationState.ClearStateAsync(turnContext, cancellationToken);
    return new InvokeResponse { Status = (int) HttpStatusCode.OK };
}

```

34. Another area of interest is how the bot responds with adaptive cards. In the CandidateTemplate.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.

```

card.Actions = new List<AdaptiveAction>();
if (candidate.Stage == InterviewStageType.Offered)
{
    card.Actions.Add(new AdaptiveShowCardAction
    {
        Title = "Schedule an interview",
        Card = GetAdaptiveCardForInterviewRequest(data.Interviewers.ToList(), candidate, DateTime.Now.Date.AddDays(1.0))
    });
}

if (candidate.Comments.Any() || candidate.Interviews.Any())
{
    var contentUrl = data.AppSettings.BaseUrl + $"/StaticViews/CandidateFeedback.html?candidateId={candidate.CandidateId}";
    card.Actions.Add(new AdaptiveOpenUrlAction
    {
        Title = "Open candidate feedback",
        Url = new Uri(string.Format(
            Constants.TenantTemplateFormat,
            data.AppSettings.MicrosoftAppId,
            Uri.EscapeDataString(ContentUrl),
            Uri.EscapeDataString(candidate.Name),
            data.AppSettings.MicrosoftAppId,
            "large",
            "large")));
    });
}

var leaveCommentCommand = new
{
    commandId = AppCommands.LeaveInternalComment,
    candidateId = candidate.CandidateId
};

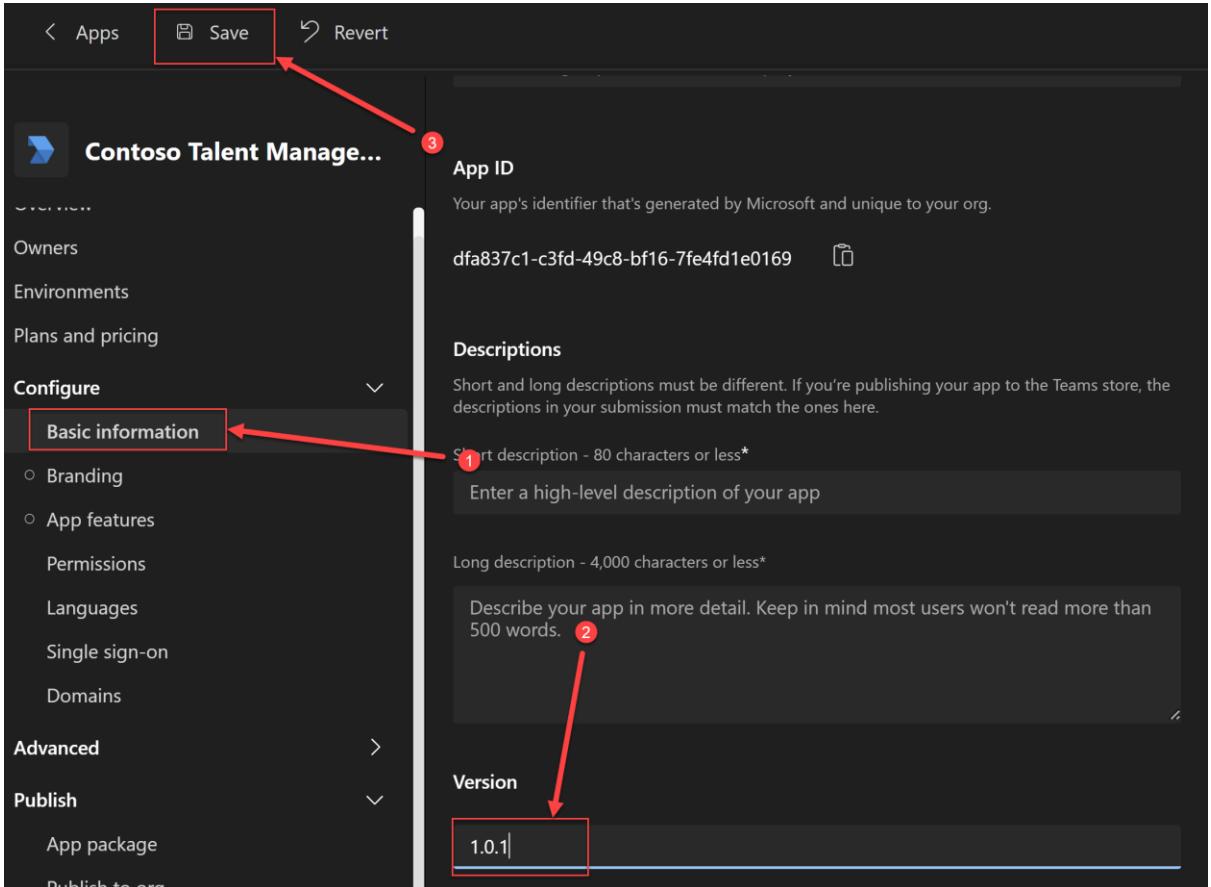
var wrapAction = new CardAction
{
    Title = "Submit",
    Value = leaveCommentCommand
};

```

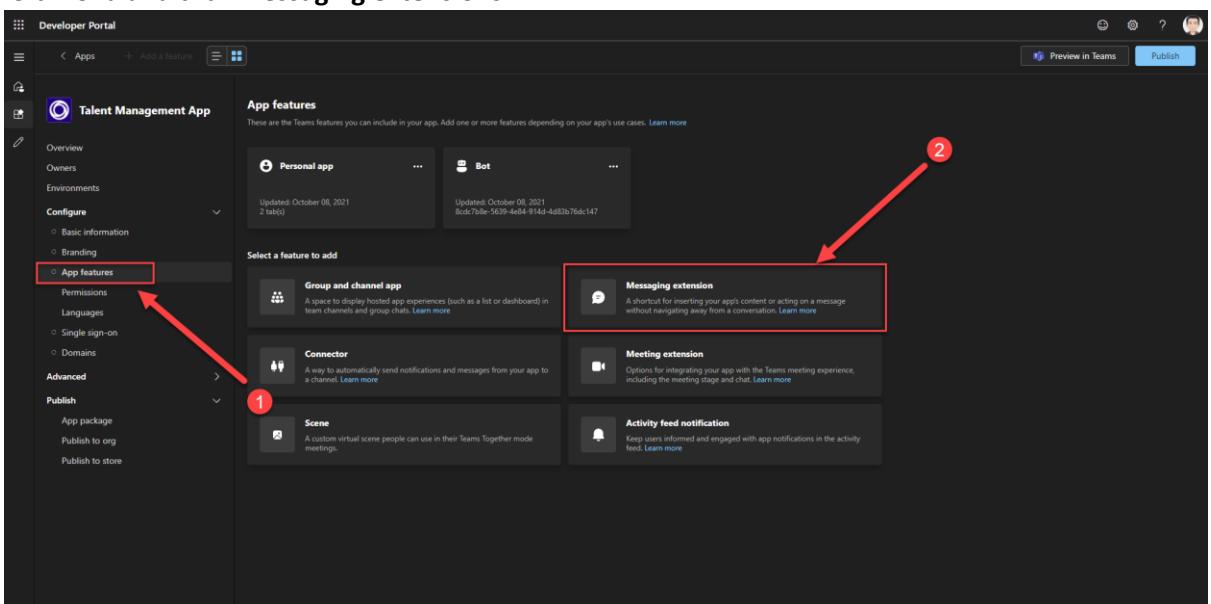
Now that the ChatBot (in a personal context) is working, we will now implement a more advanced Bot feature in Teams, known as a Messaging Extension. Messaging Extension search is where we will begin, and then we'll move onto Messaging Extension actions.

- 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

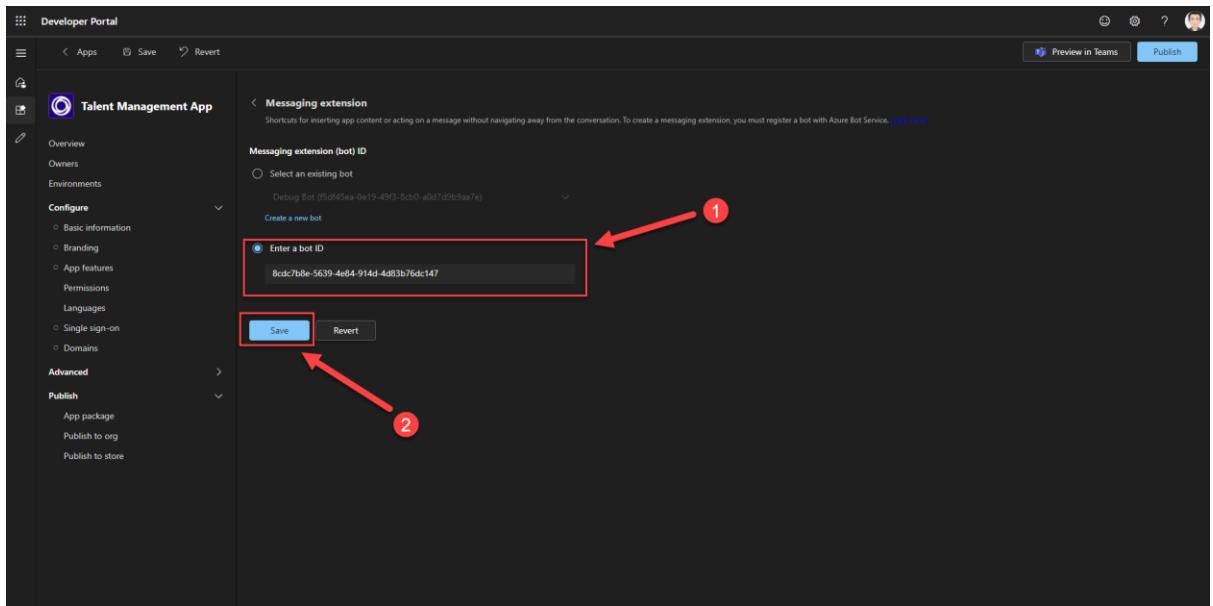
- As with everything else in the manifest we need to go back to the Developer Portal to add a new feature. Let's start with iterating the version number, click **Basic information**. Scroll down and iterate the number from 1.0.0 to **1.0.1** – click **Save**.



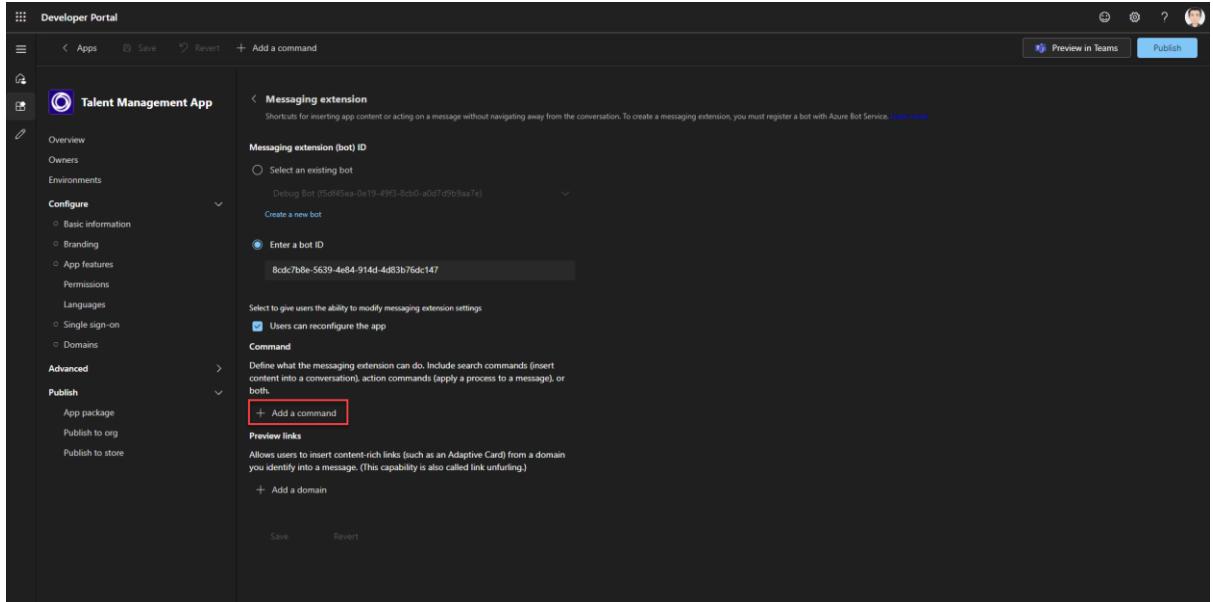
- Now we can configure the Messaging extension configuration. Select **App Features** from the left menu and click **Messaging extensions**.



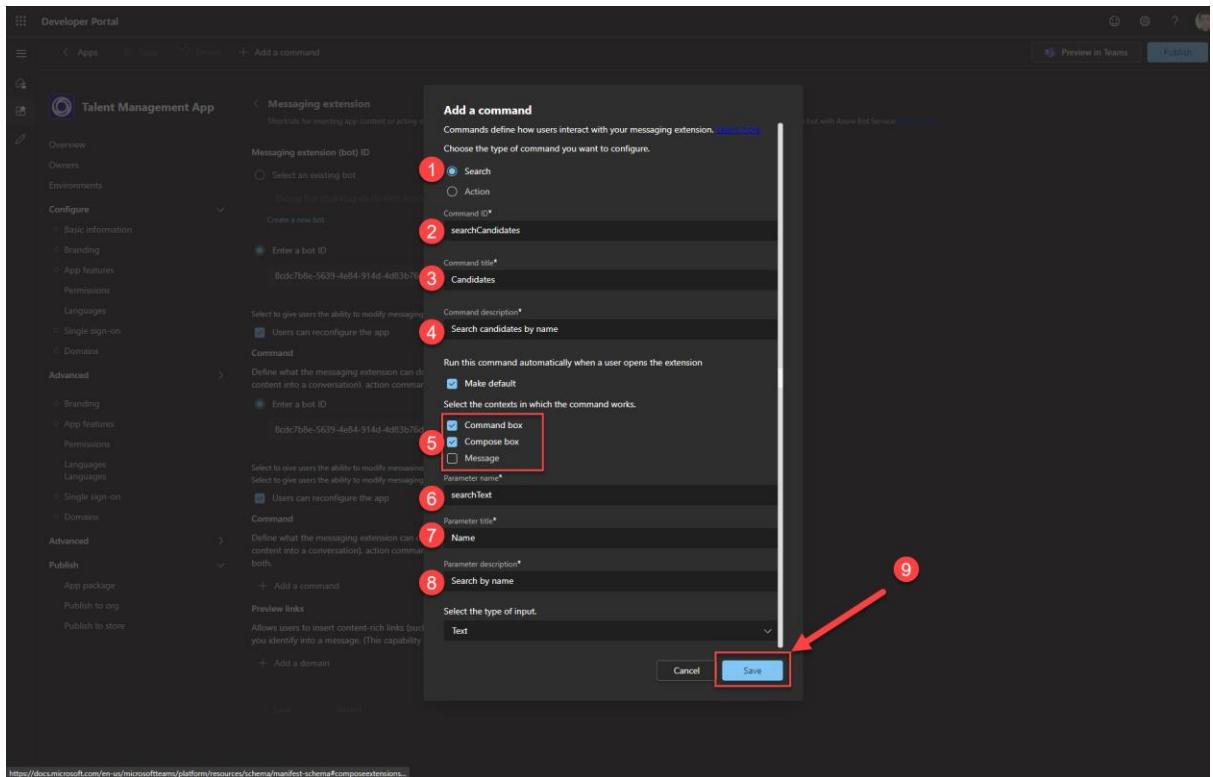
- Select **Enter a bot id** and enter your App Registration *Client Id* into the textbox. (This is so that you can have a different bot handling your messaging extensions if you wish.) Once this is done, click **Save**.



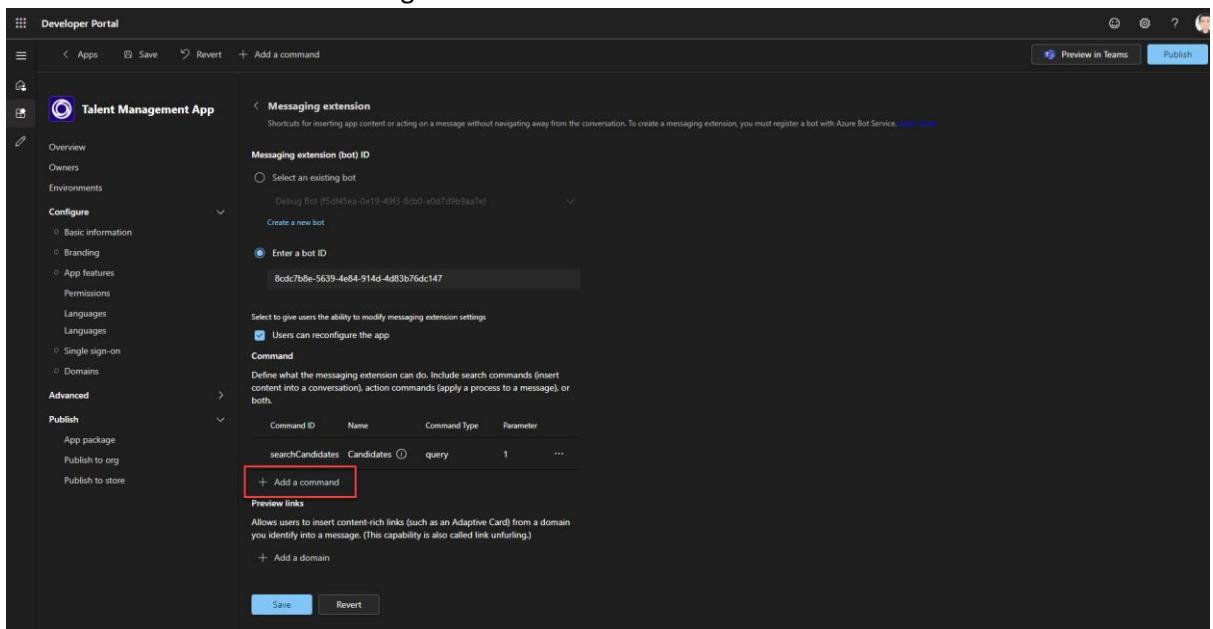
- Now we need to add the commands. Select **Add a command**.



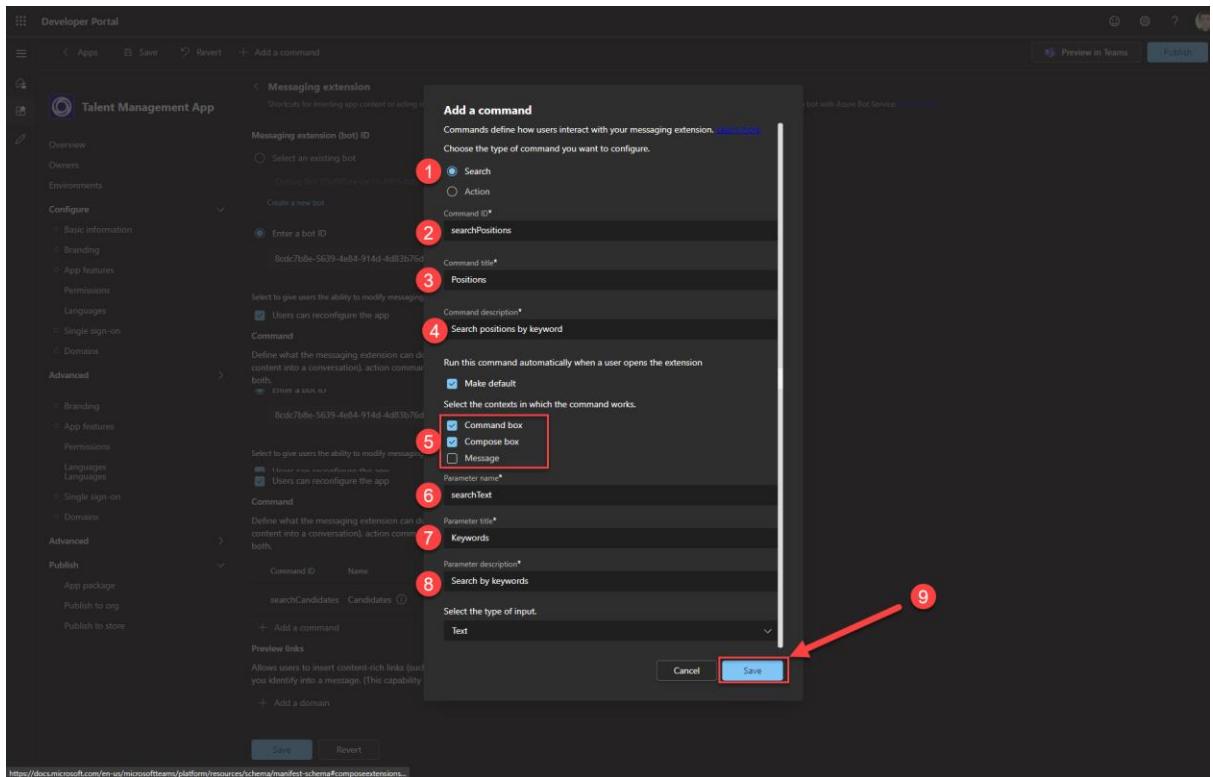
- Enter the values below to add the search candidates search command and click **Save**.
Note: we are selecting Command box and Compose box as the areas within Teams that this function will be available. This is the large search box at the top of the Teams interface (Command box) and the area you type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



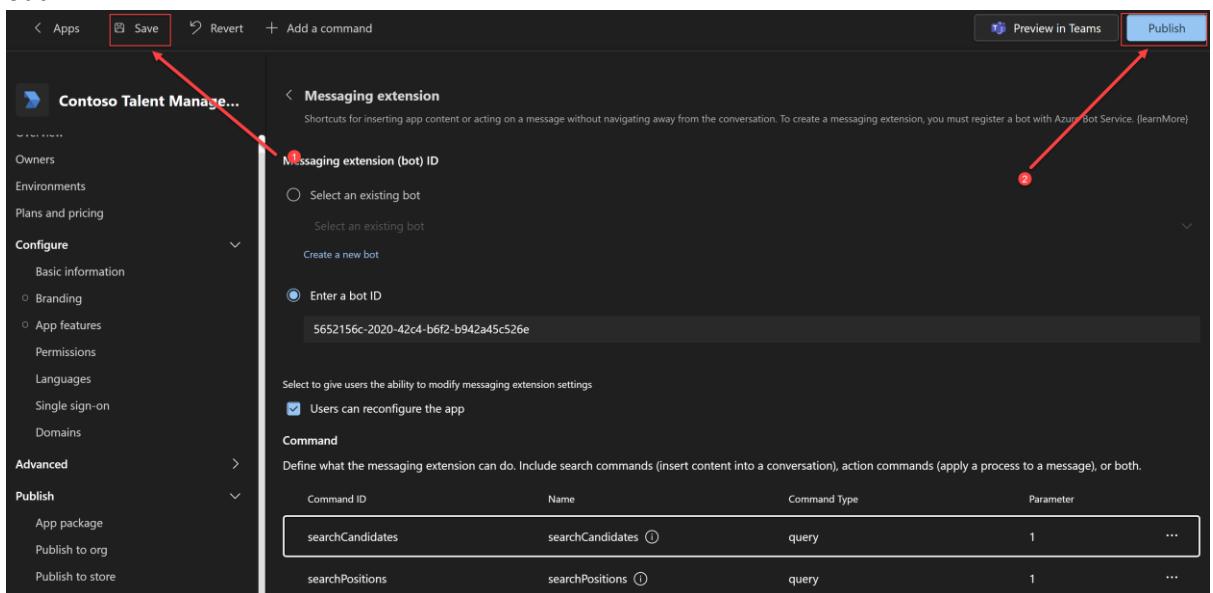
6. Once done, we will add a second search command, this one is for the position search function. Click **Add a command** again.



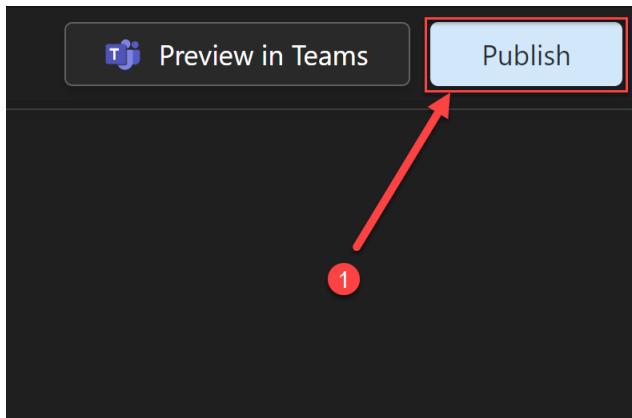
7. Enter the values below and click **Save**.



8. Now you can see the two commands have been added, click **Save** and then **Publish** to test it out.



9. Now that the Messaging Extension Search has been configured completely, let's test it out! Click **Publish**.



10. Click **Download the app package**, this will save the app manifest ZIP into your Downloads folder.

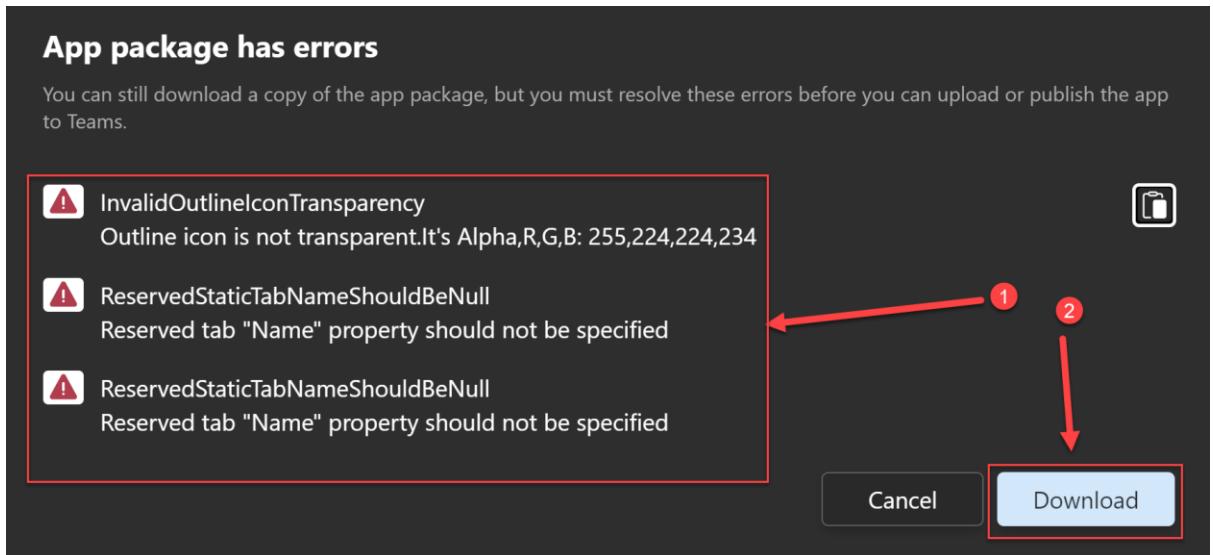
Publish your app

Download the app package
Download a copy of your app package, which is specific to your selected environment. Use the package to upload your app in Teams or publish later.

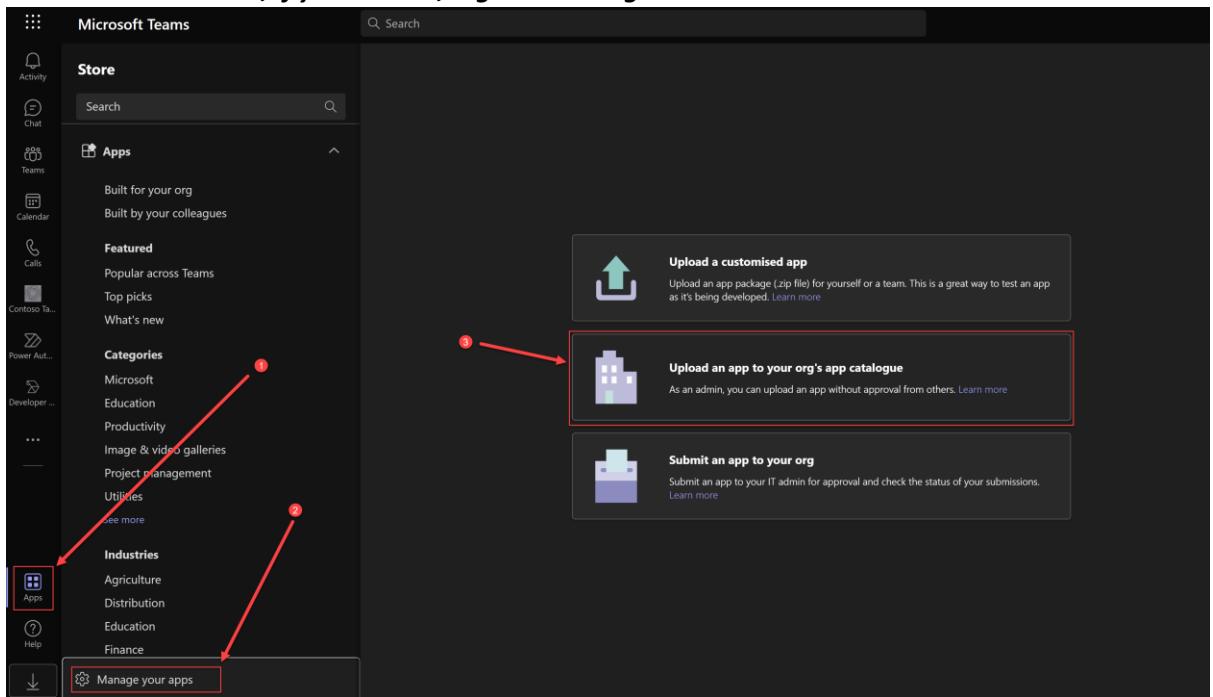
Publish to your org
Submit a request to your IT admin to publish your app. It will appear in the Built for your org section of the store once it's approved.

Publish to the Teams store
Make your app available to Teams users everywhere. This option requires Microsoft approval.

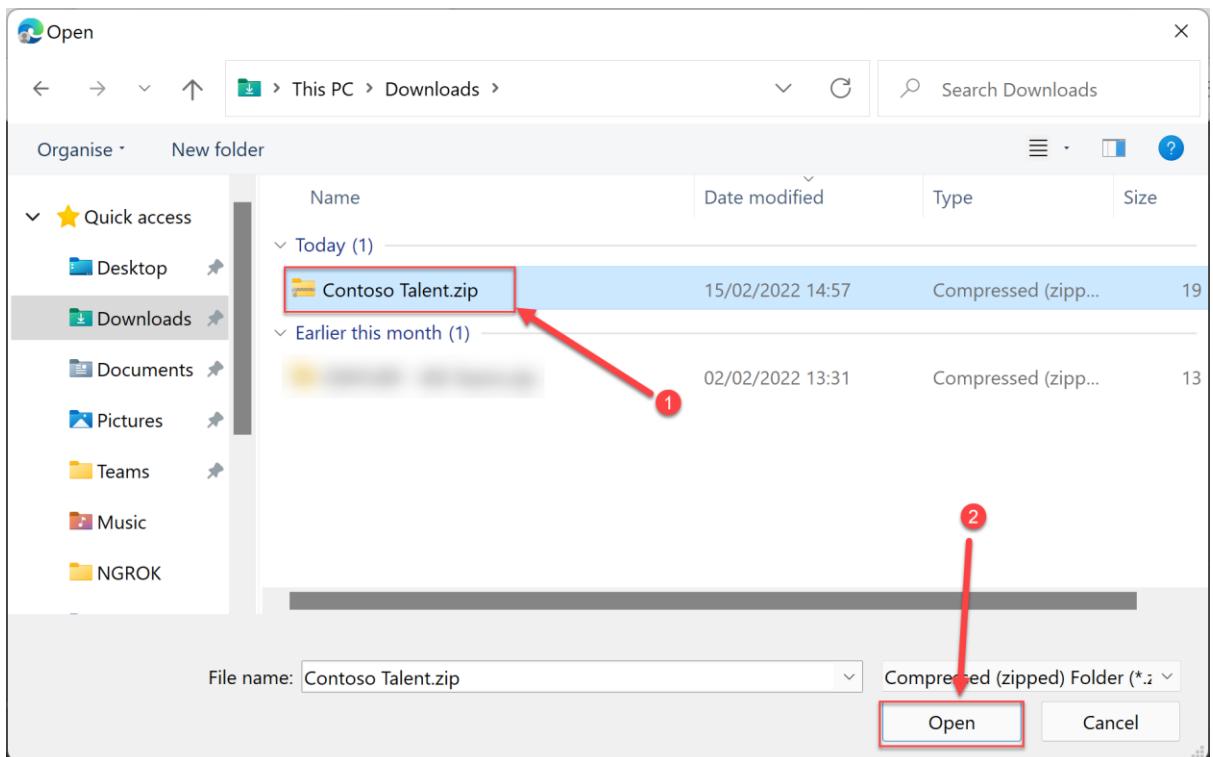
11. If the App package has errors, they will display here. Some errors must be fixed before you can install the app into Teams (note, in the screenshot below, all 3 errors can be safely ignored), **review and fix the errors**, click **Download**.



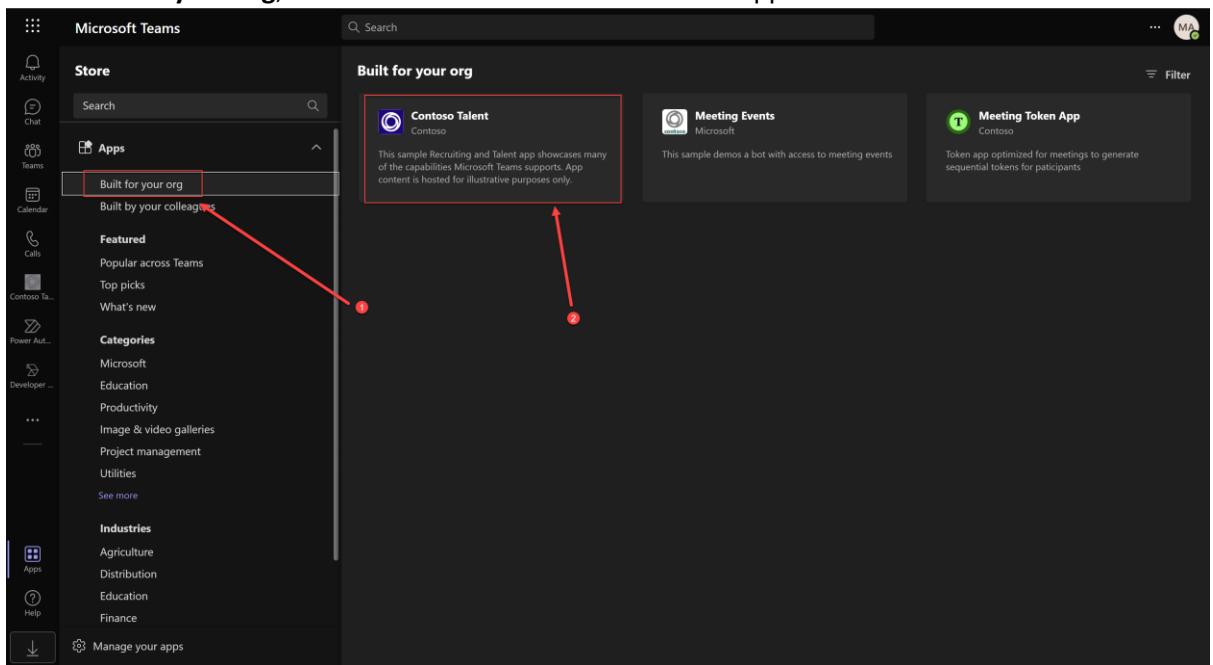
12. Go back into Teams, select **Apps**, then click **Manage your apps**, click **Upload an app to your org's app catalogue**. *Note: this option will only exist if you are logged into Teams as a Global Administrator, if you are not, log-out and log-in as a Global Administrator.*



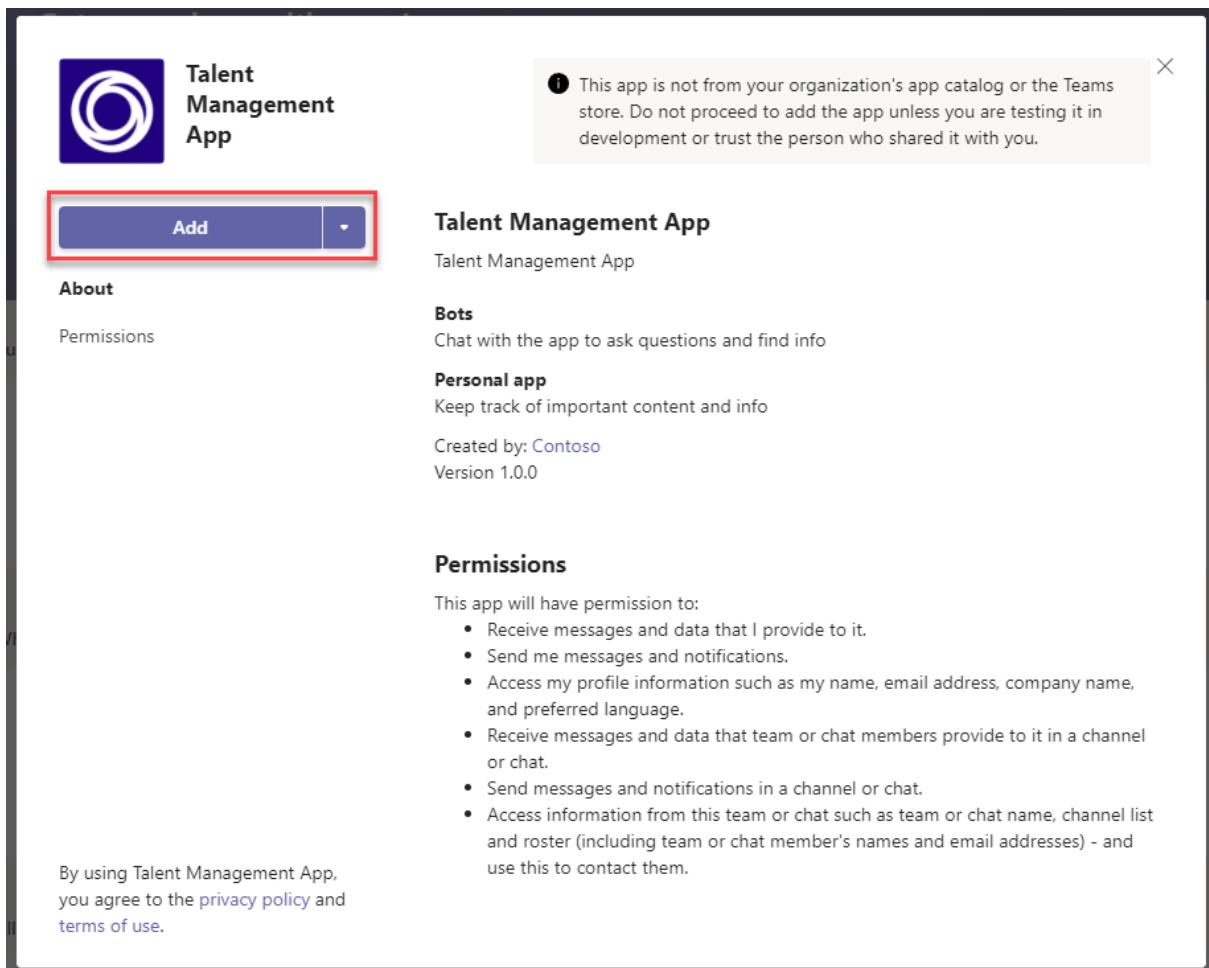
13. Navigate to the directory (folder) where the Teams App Manifest was saved (probably Downloads). Select the **App Manifest ZIP**, and click **Open**.



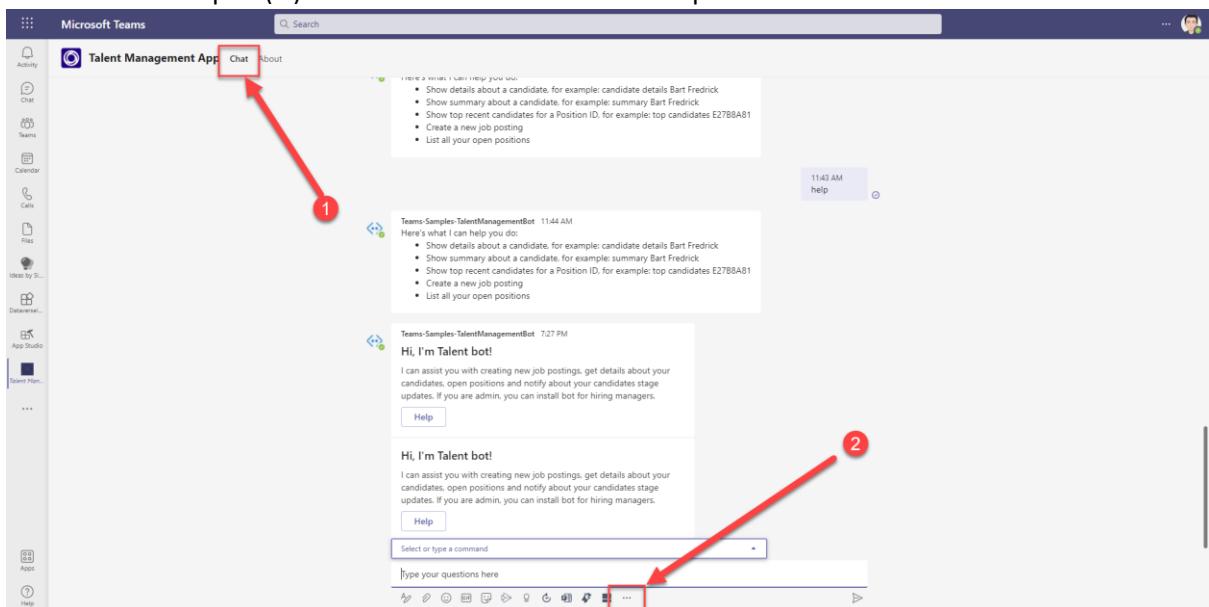
14. Click **Built for your org**, and select the **Contoso Talent** Teams app.



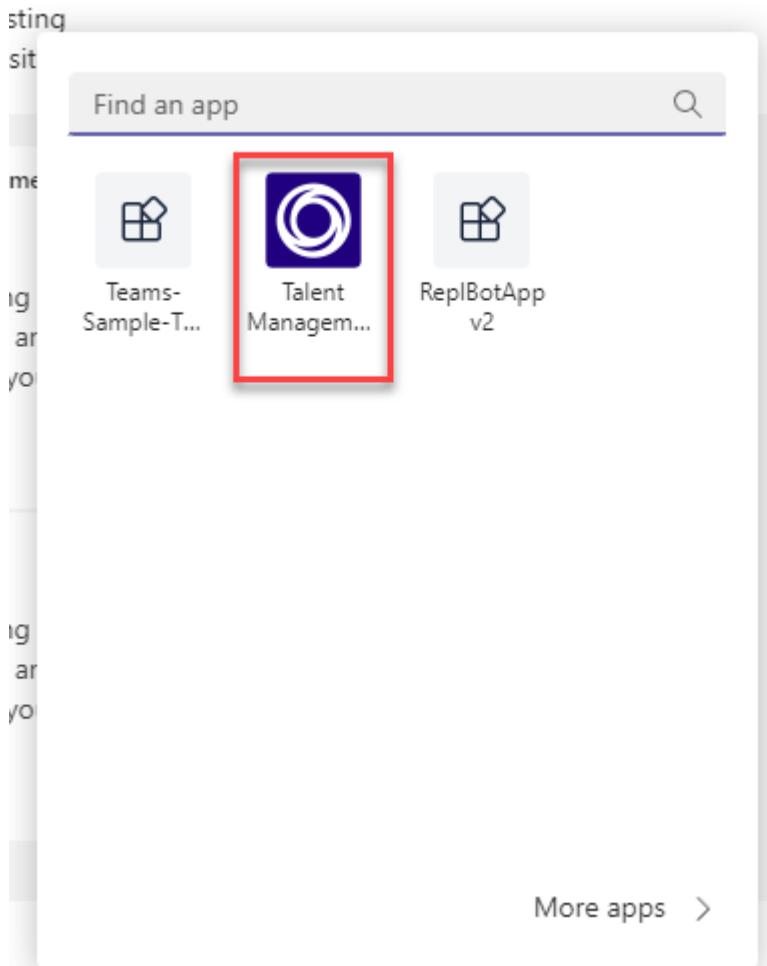
15. Click **Add** to install the latest version of this application, which includes the Messaging Extension Search.



16. 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.



17. 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.



18. 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.

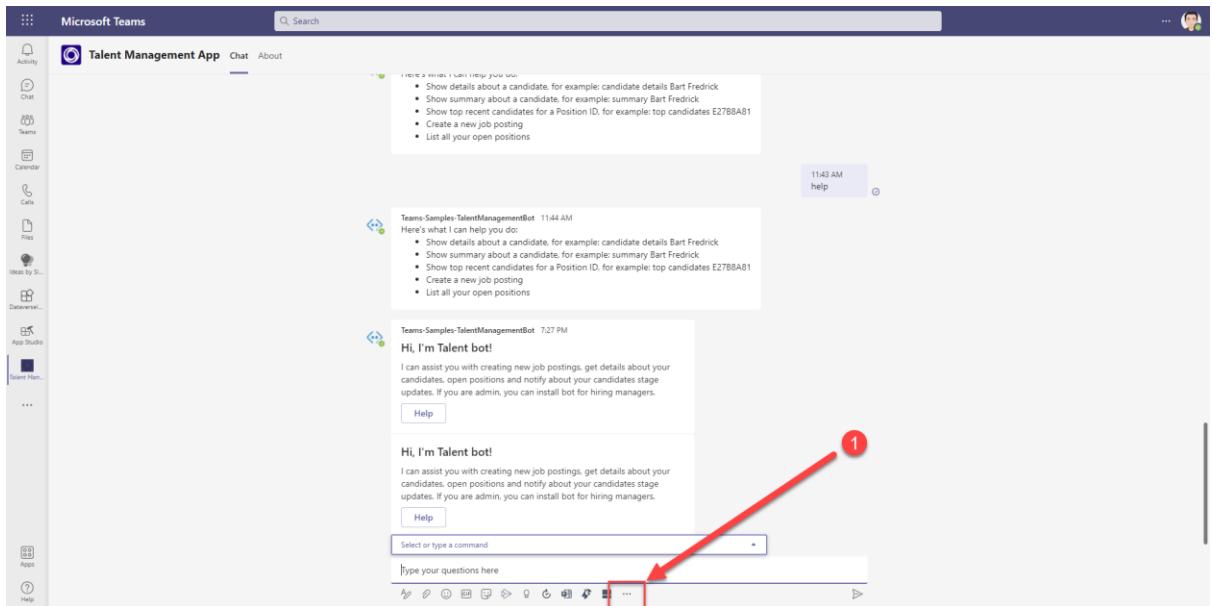
The screenshot shows a web-based application titled "Talent Management App". At the top, there is a search bar labeled "Search by name" with a magnifying glass icon. Below the search bar are two tabs: "Candidates" (which is underlined) and "Positions". A list of candidates is displayed below the tabs. The first candidate, "Bart Fredrick", is highlighted with a red box. His profile picture is on the left, followed by his name and title: "Current role: Senior Program Manager | New York, NY". The other candidates listed are Scotty Cothran, Donna Delaney, Wallace Santoro, and Samuel Rodman, each with their respective profile pictures and titles.

19. 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.

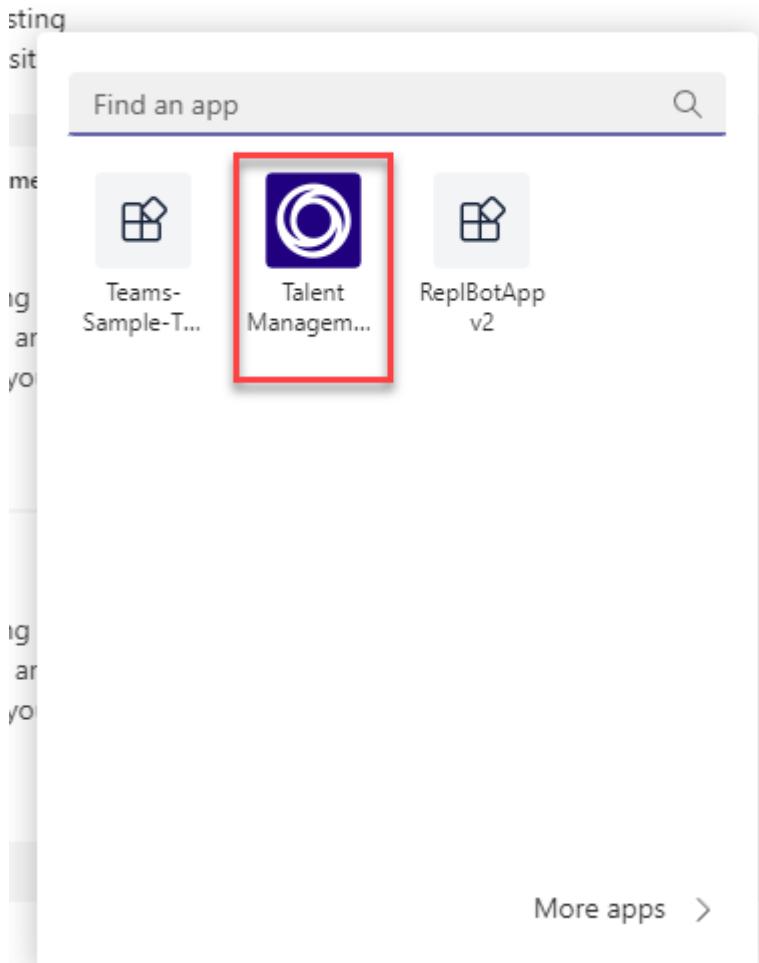
The screenshot shows a Microsoft Teams chat window. On the left, there is a sidebar with various icons for Activity, Chat, Teams, Calendar, Calls, Files, Idea by S... (disabled), Database, App Studio, and Talent M... (selected). The main area shows a conversation with a bot named "Hi, I'm Talent bot!". The bot has sent a message with a link to "List all your open positions". Below this, there is a message from "Teams-Samples/TalentManagementBot" at 7:27 PM. The message contains an adaptive card with the following details:

Talent Management App	
	Bart Fredrick
Ten years of experience in the software industry. Five years experience working at a software consulting firm.	
Current role:	Senior Program Manager
Location:	New York, NY
Stage:	Applied
Position applied:	Senior Designer
Date applied:	Tuesday, September 11, 2018
Phone number:	+1 212 555 5445
Schedule an interview Leave comment	

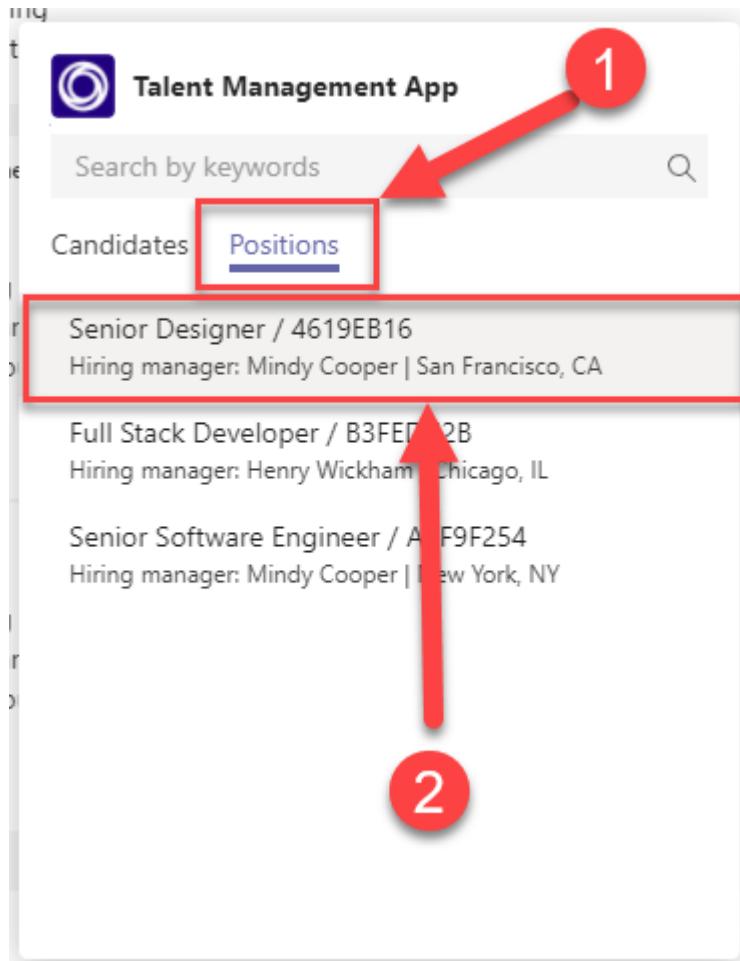
20. Repeat this process to see the positions adaptive card. Click the ellipsis again (...).



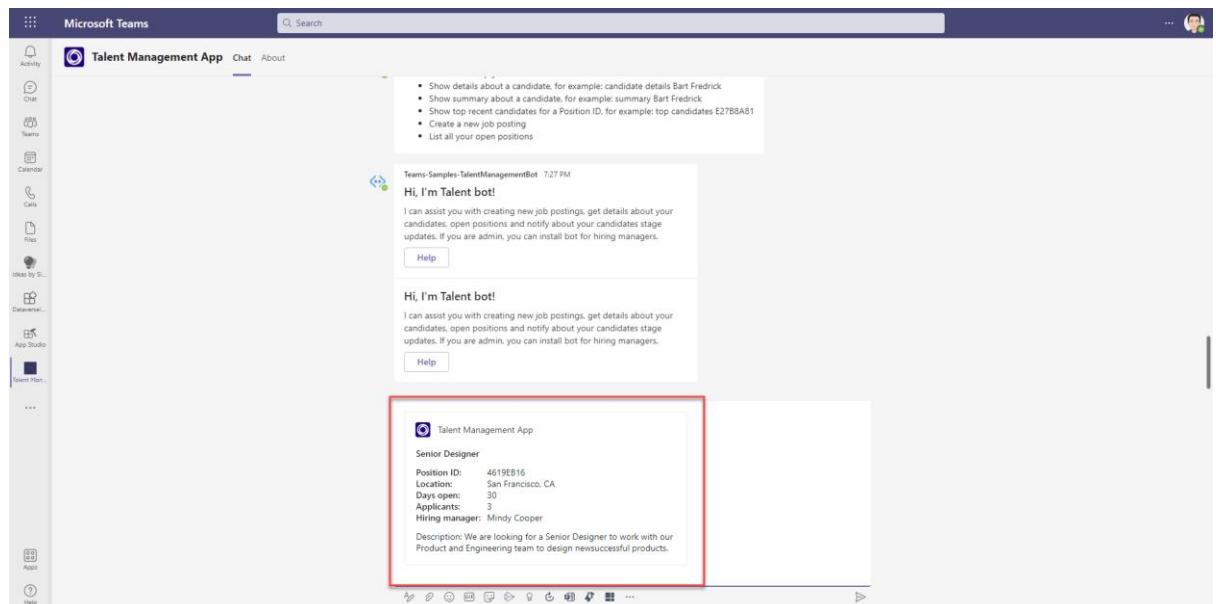
21. Find your application



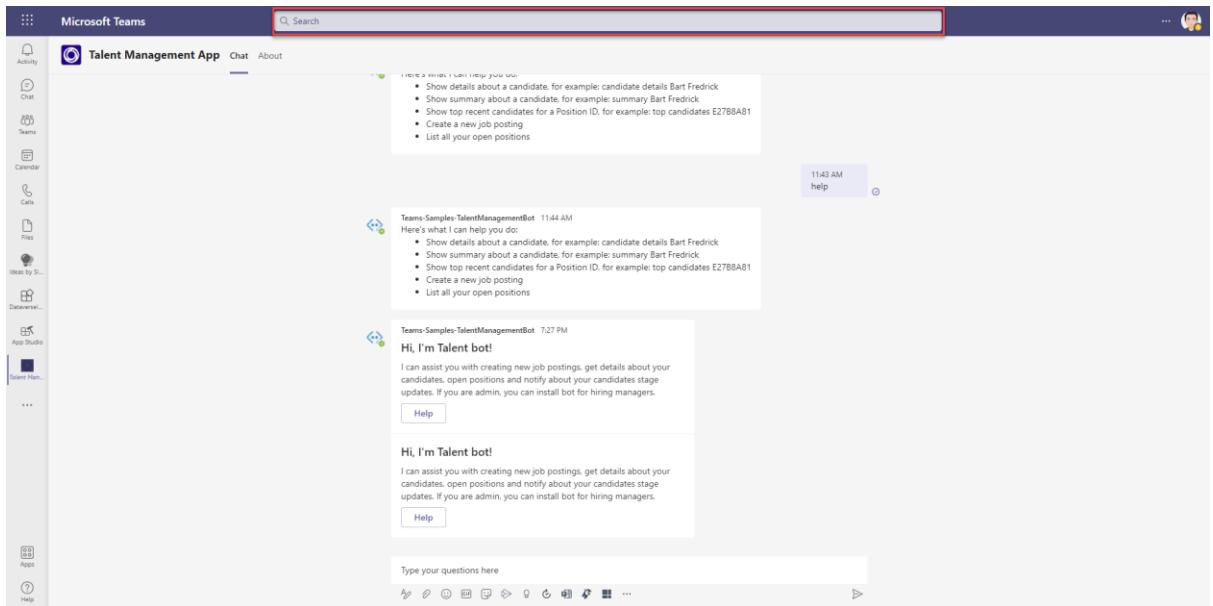
22. This time, select **Positions**. You will see a list of positions displayed. Now select one.



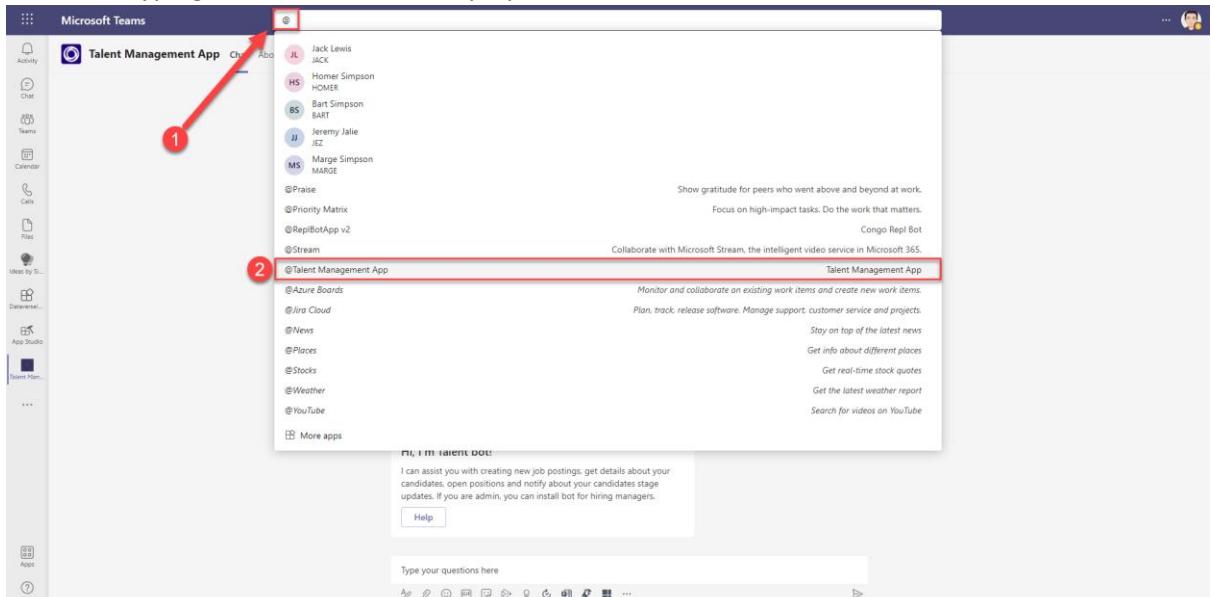
23. As before you will see that an adaptive card has been added to the compose box ready to be sent!



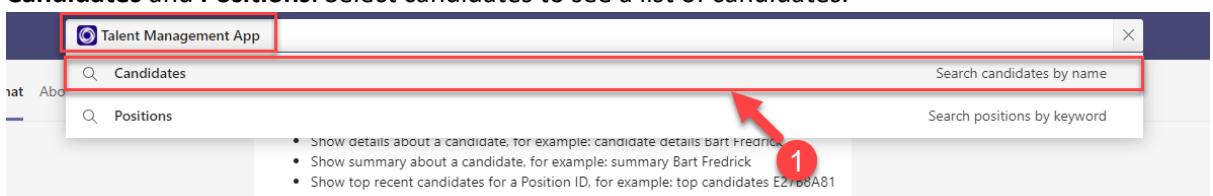
24. Another feature of a search command is that it can be made available to the global search bar (Command box) at the top of the Teams interface.



25. 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.



26. 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.



27. 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.

Talent Management App > Candidates		[Search by name]
	Bart Fredrick Current role: Senior Program Manager New York, NY	
	Scotty Cothran Current role: Software Developer II New York, NY	
	Donna Delaney Current role: Software Developer II Dallas, TX	
	Wallace Santoro Current role: Marketing Manager New York, NY	
	Samuel Rodman Current role: Software Developer II San Francisco, CA	

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

 Talent Management App > Candidates Bart Fredrick X

 Talent Management App

Bart Fredrick
Ten years of experience in the software industry.
Five years experience working at a software consulting firm.

Current role: Senior Program Manager
Location: New York, NY
Stage: Applied
Position applied: Senior Designer
Date applied: Tuesday, September 11, 2018
Phone number: +1 212 555 5445

Schedule an interview Leave comment

29. 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.

```
34     _botService = botService;
35     _invokeActivityHandler = invokeActivityHandler;
36 }
37
38 public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken) => default(CancellationToken)
39 {
40     await base.OnTurnAsync(turnContext, cancellationToken);
41
42     // Save any state changes before sending a message.
43     await _conversationsState.SaveChangesAsync(turnContext);
44     await _userState.SaveChangesAsync(turnContext);
45 }
46
47 protected override Task OnTeamsMessageReceivedAsync(ITurnContext turnContext, CancellationToken cancellationToken) => default(CancellationToken)
48 {
49     await _invokeActivityHandler.HandleMessageReceivedAsync(turnContext, cancellationToken);
50 }
51
52 protected override Task<Object> Value(IInvokeActivityHandler handler, IActivity activity, CancellationToken cancellationToken)
53 {
54     var value = await handler.HandleActivityAsync(activity, cancellationToken);
55
56     return value;
57 }
58
59 protected override Task<Object> Value(IInvokeActivityHandler handler, string commandId, string queryText, CancellationToken cancellationToken)
60 {
61     var value = await handler.HandleActivityAsync(commandId, queryText, cancellationToken);
62
63     return value;
64 }
65
66 protected override Task<Object> Value(IInvokeActivityHandler handler, string commandId, LinkQuery query, CancellationToken cancellationToken)
67 {
68     var value = await handler.HandleActivityAsync(commandId, query, cancellationToken);
69
70     return value;
71 }
72
73 protected override Task OnFileAttachmentAddedAsync(ITurnContext turnContext, IAttachment cardAttachment, CancellationToken cancellationToken)
74 {
75     if (turnContext.Activity.Attachments != null)
76     {
77         return _botService.HandleFileAttachments(turnContext, cancellationToken);
78     }
79
80     return Task.CompletedTask;
81 }
```

No issues found

Locals

Name	Type
turnContext	Microsoft.Bot.Builder.TurnContext
cancellationToken	System.Threading.CancellationToken

Exception Settings

Break When Throw	Conditions
b: Common Language Runtime Exceptions	
b: GPU Memory Access Exceptions	
b: Invalid Arguments	
b: JavaScript Exceptions	
b: JavaScript Runtime Exceptions	

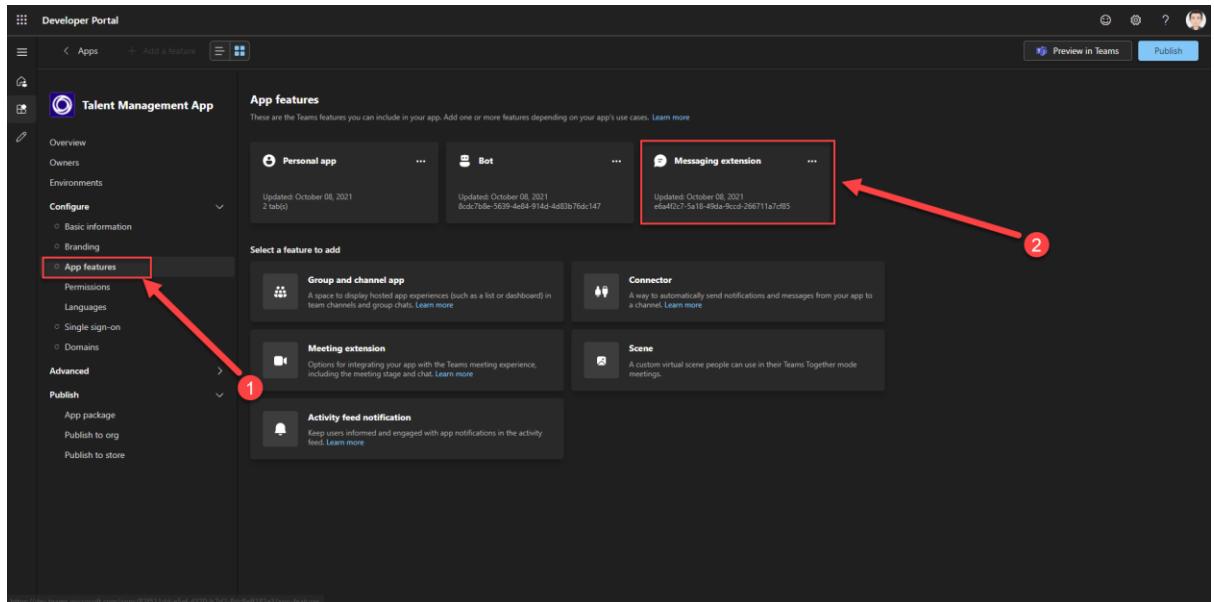
Call Stack

Now that we have implemented the Messaging Extension search feature, we will now move onto a Messaging Extension Action feature!

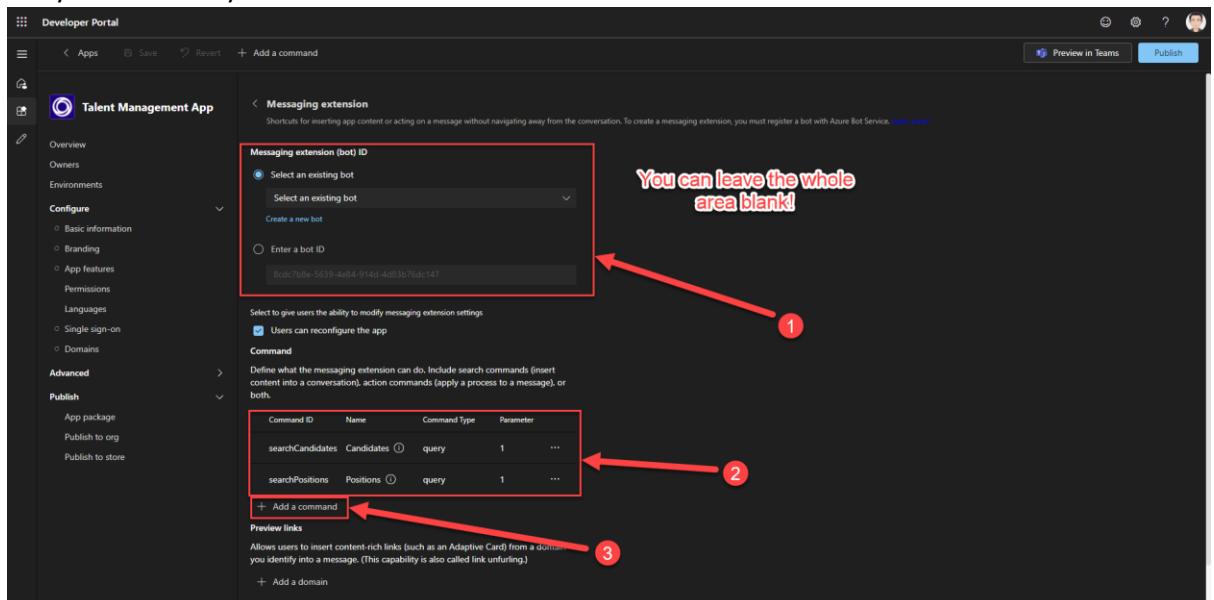
6) Implement a Message Extension Action

In this step we will add another message extension to perform an action instead of a search, this will allow us to create information (positions) in a third-party app, rather than just searching for objects/information that already exists.

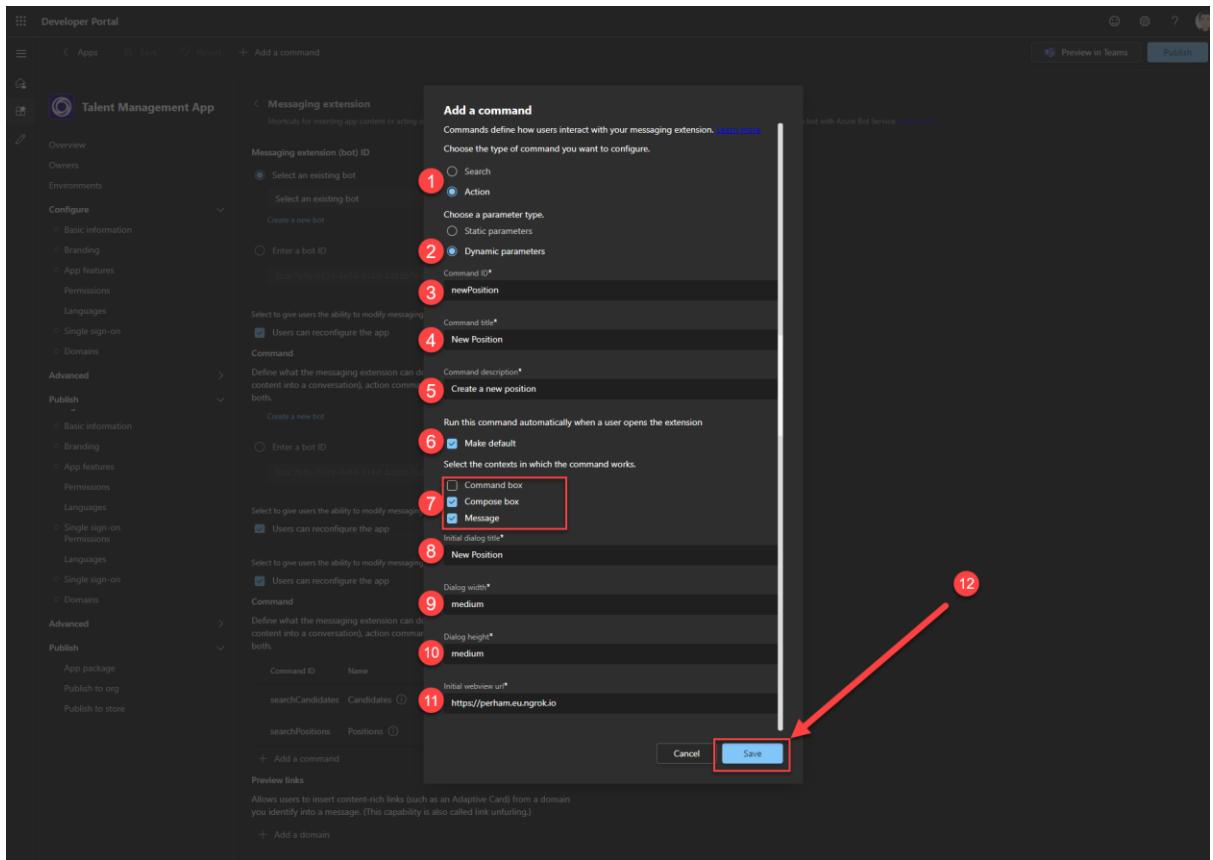
1. Open the Developer Portal. Let's start with iterating the version number, click **Basic information**. Scroll down and iterate the number from 1.0.1 to **1.0.2** – click **Save**.
2. Navigate to App Features and select Messaging extensions to edit the manifest.



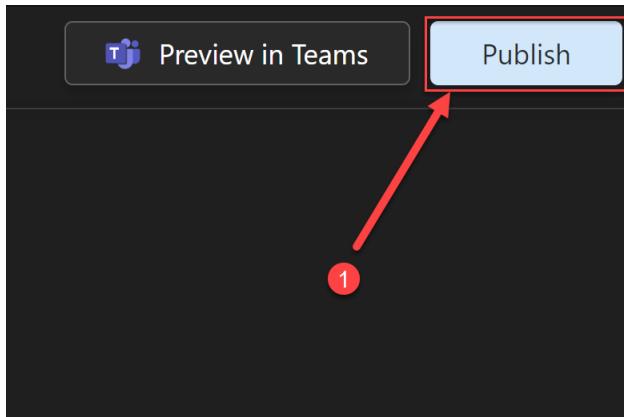
3. Here we will add a new command, if you can see the search commands listed then you are already in the context of your bot and don't need to select anything from the top section. If not you can enter your bot id as before. Then click **Add a command**.



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



- Now that the Messaging Extension Action has been configured completely, let's test it out!
- Click **Publish**.



- Click **Download the app package**, this will save the app manifest ZIP into your Downloads folder.

Publish your app

 **Download the app package**
Download a copy of your app package, which is specific to your selected environment. Use the package to upload your app in Teams or publish later.

 **Publish to your org**
Submit a request to your IT admin to publish your app. It will appear in the Built for your org section of the store once it's approved.

 **Publish to the Teams store**
Make your app available to Teams users everywhere. This option requires Microsoft approval.

7. If the App package has errors, they will display here. Some errors must be fixed before you can install the app into Teams (note, in the screenshot below, all 3 errors can be safely ignored), **review and fix the errors**, click **Download**.

App package has errors

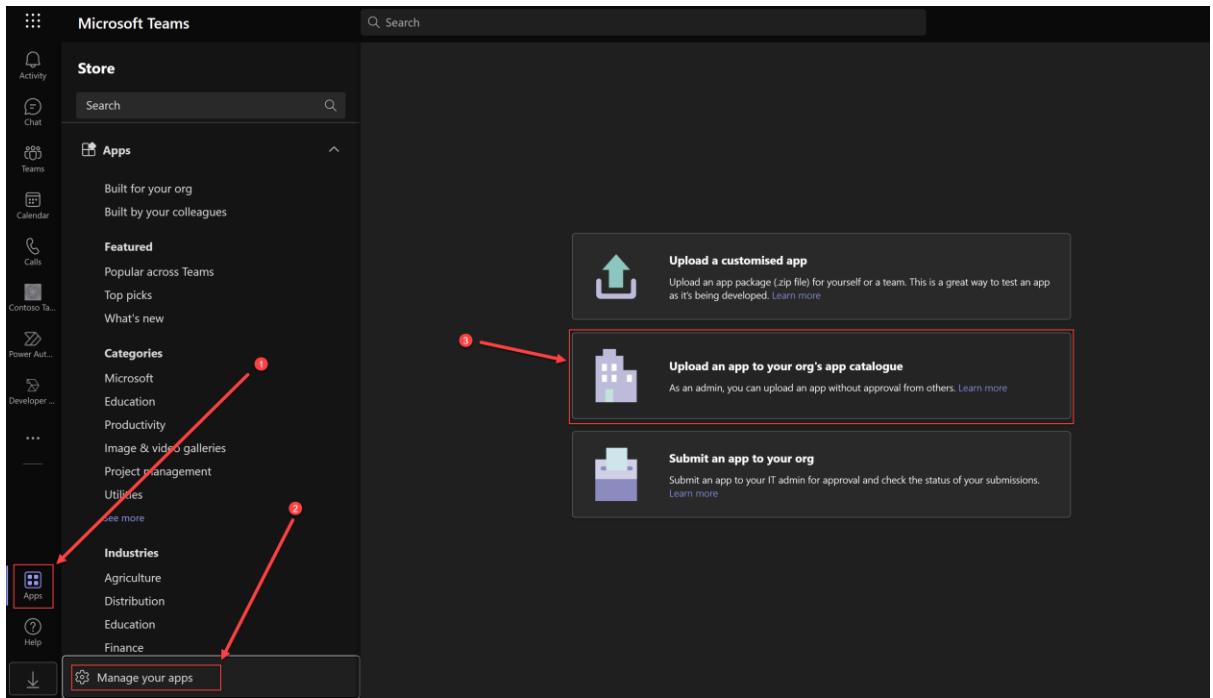
You can still download a copy of the app package, but you must resolve these errors before you can upload or publish the app to Teams.

 InvalidOutlineIconTransparency
Outline icon is not transparent. It's Alpha,R,G,B: 255,224,224,234

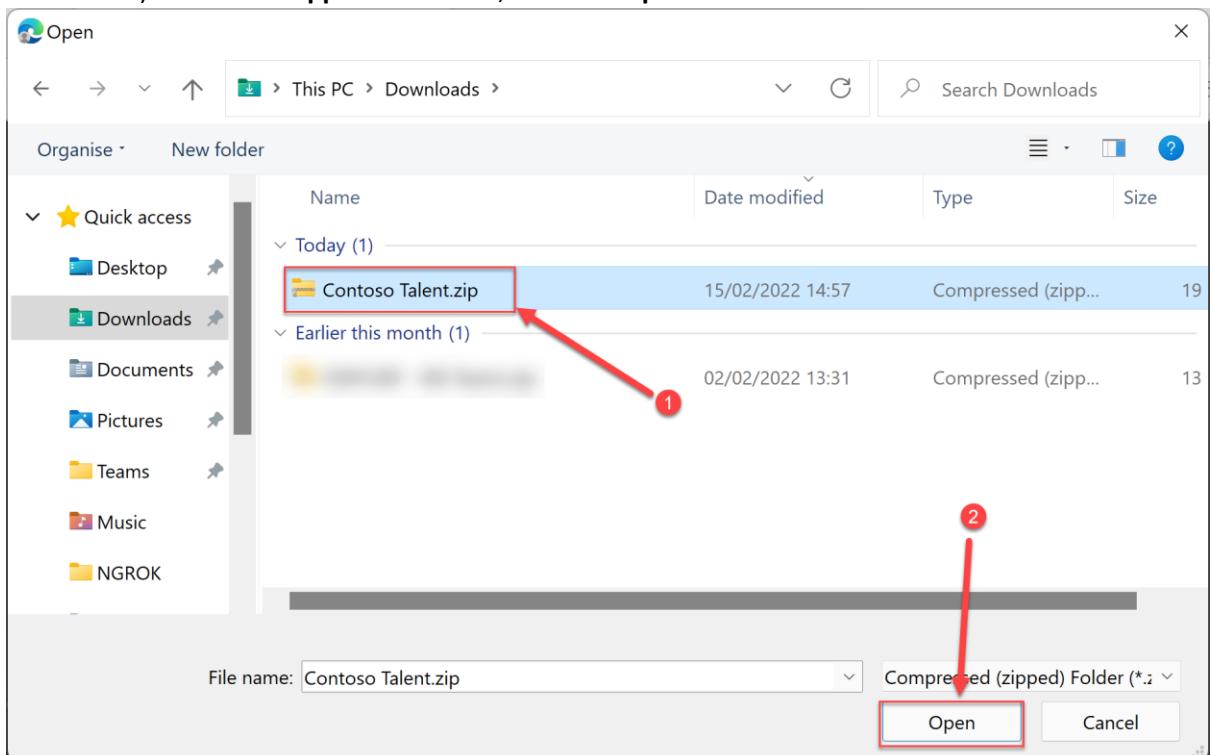
 ReservedStaticTabNameShouldBeNull
Reserved tab "Name" property should not be specified

 ReservedStaticTabNameShouldBeNull
Reserved tab "Name" property should not be specified

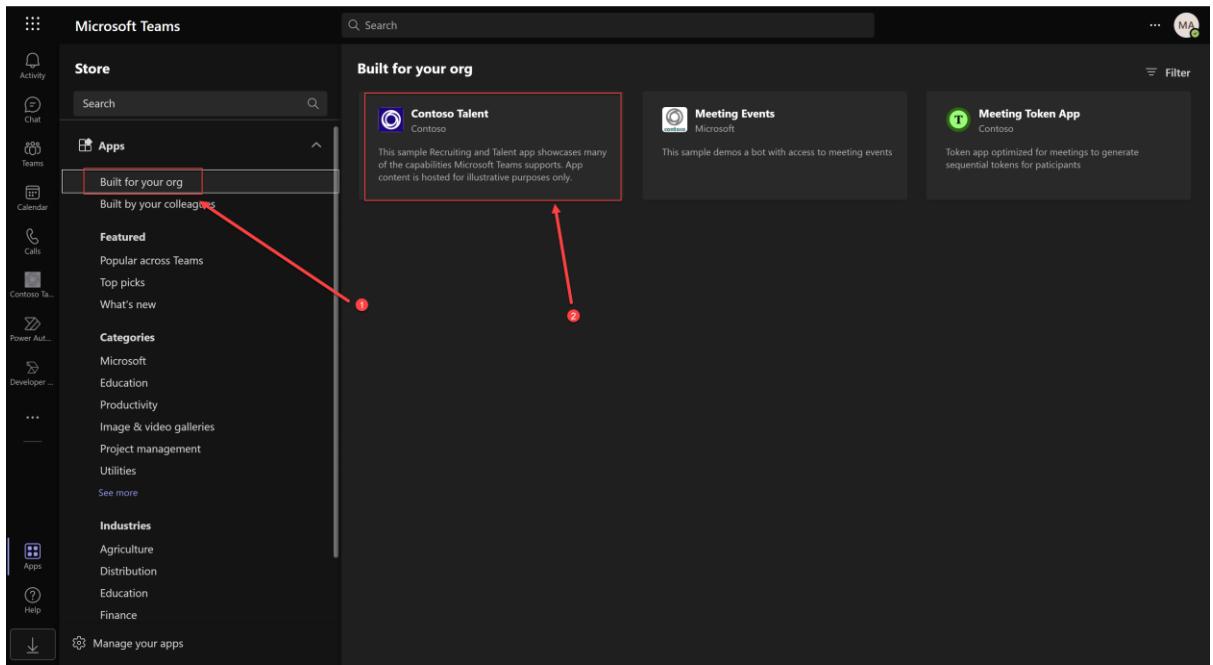
8. Go back into Teams, select **Apps**, then click **Manage your apps**, click **Upload an app to your org's app catalogue**. *Note: this option will only exist if you are logged into Teams as a Global Administrator, if you are not, log-out and log-in as a Global Administrator.*



9. Navigate to the directory (folder) where the Teams App Manifest was saved (probably Downloads). Select the **App Manifest ZIP**, and click **Open**.



10. Click **Built for your org**, and select the **Contoso Talent** Teams app.



11. Click **Add** to install the latest version of this application, which includes the Messaging Extension Action.

Talent Management App

This app is not from your organization's app catalog or the Teams store. Do not proceed to add the app unless you are testing it in development or trust the person who shared it with you.

Add

About

Permissions

Bots
Chat with the app to ask questions and find info

Personal app
Keep track of important content and info

Created by: Contoso
Version 1.0.0

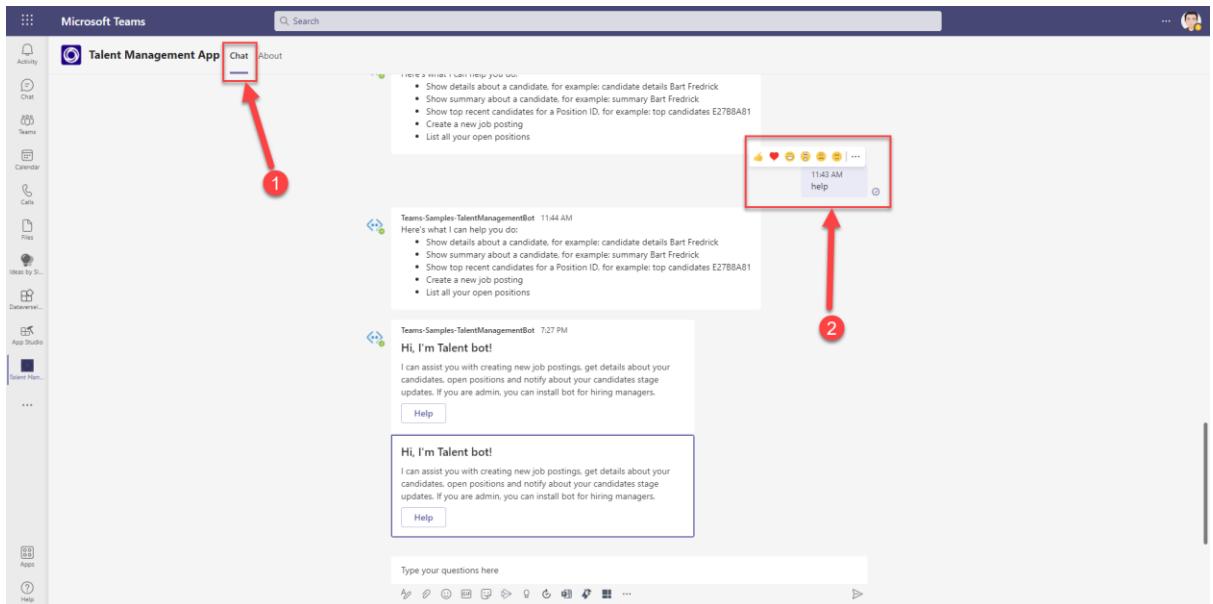
Permissions

This app will have permission to:

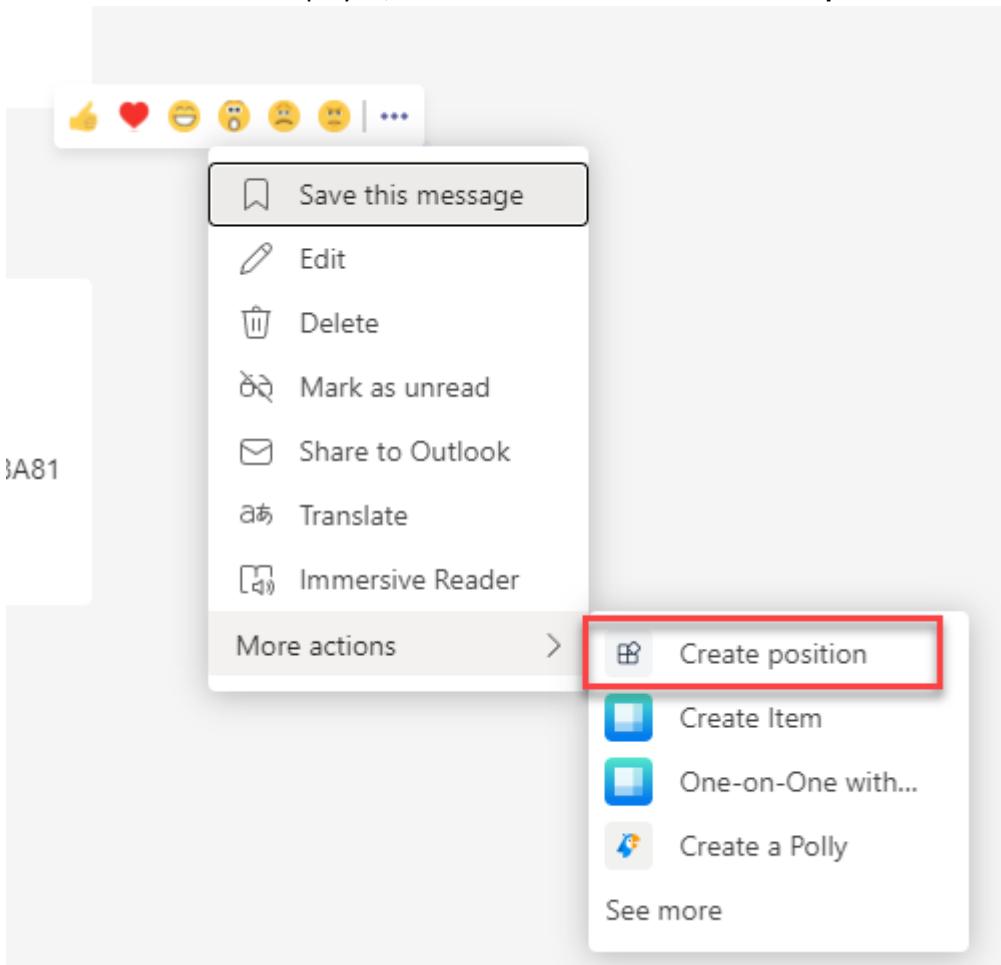
- Receive messages and data that I provide to it.
- Send me messages and notifications.
- Access my profile information such as my name, email address, company name, and preferred language.
- Receive messages and data that team or chat members provide to it in a channel or chat.
- Send messages and notifications in a channel or chat.
- Access information from this team or chat such as team or chat name, channel list and roster (including team or chat member's names and email addresses) - and use this to contact them.

By using Talent Management App, you agree to the [privacy policy](#) and [terms of use](#).

12. 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 (...).



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



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

Teams-Sample-TM V2
Create new job posting

Enter in basic information about this posting

Title: C# Software Engineer

Level: 10

Post by: Oct 8, 2021

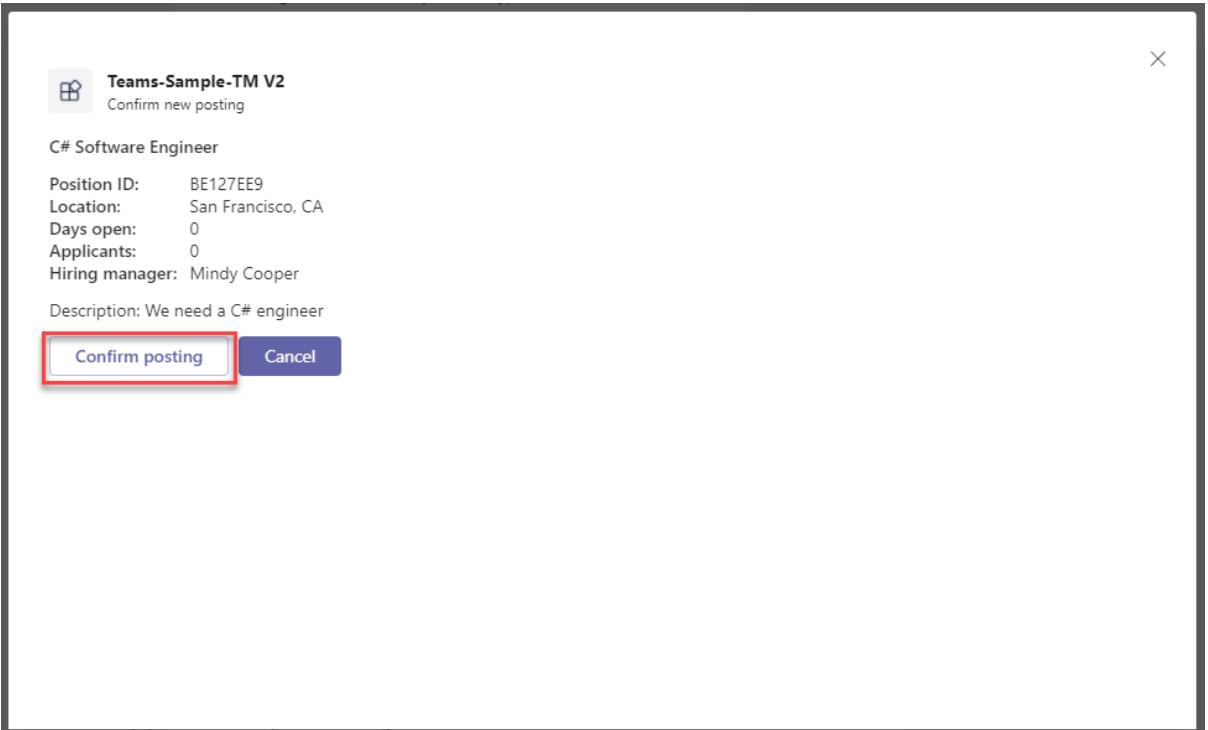
Location: San Francisco

Hiring manager: Mindy Cooper

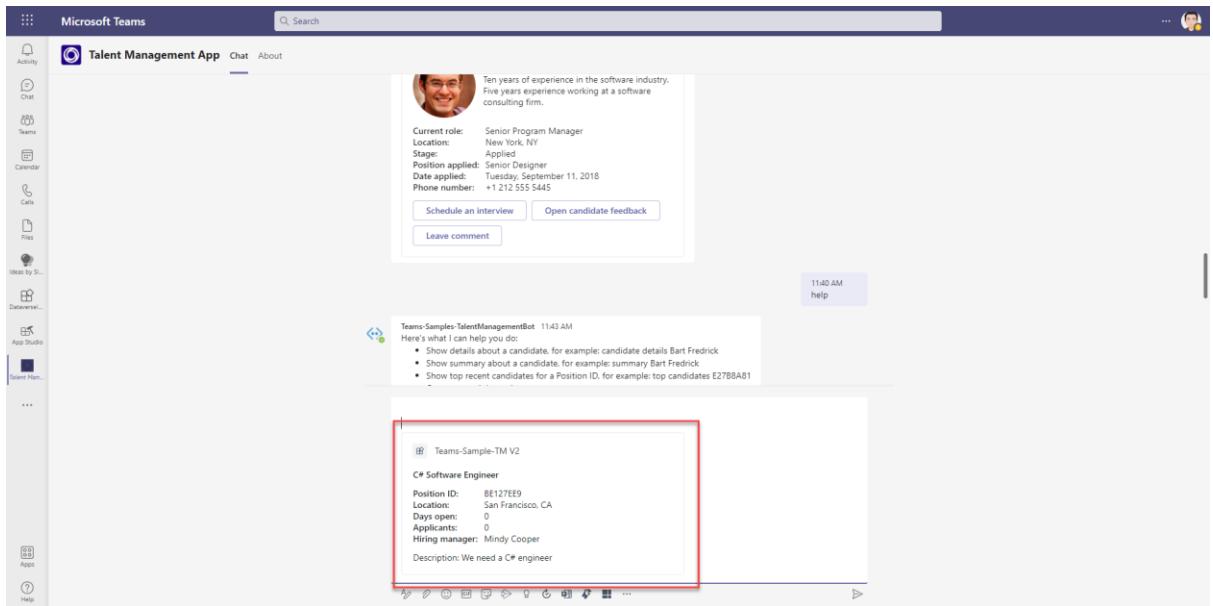
Description: We need a C# engineer

Create posting

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



16. 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!



Now that the Messaging Action is working, we will now spend some time working with the Microsoft Graph API, which will allow us to perform CRUD (create, read, update & delete) operations, via a REST API, to interact with Microsoft 365 resources.

7) Working with Microsoft Graph API

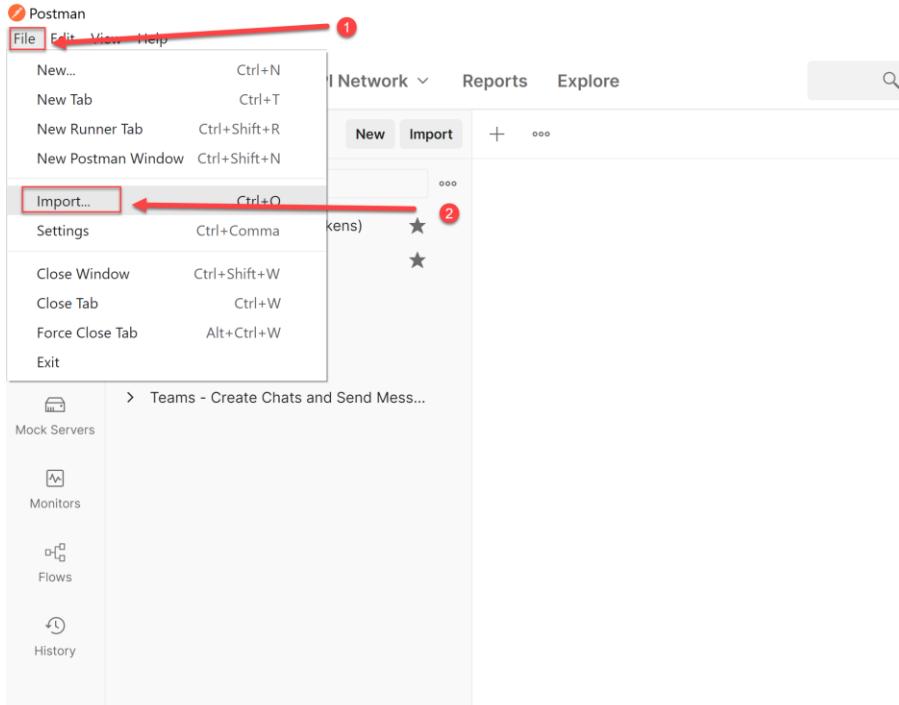
We will now move away from working with the Talent Management app for this section, to learn about the Microsoft Graph API, and how it allows you to work with Microsoft 365 data. We will begin by using the Microsoft Graph Explorer tool, a sandbox app that was created by Microsoft to introduce you to working with the Microsoft Graph API. We will then move onto crafting some custom API calls, using Postman, and will clearly define the differences between a Delegated Access Token and an Application Access Token. Finally, we will review some of the code that is used in the Talent Management App, to proactively install the application you have been working on, as this is required to enable Bot Proactive Messaging in Teams.

1. Microsoft 365 includes a suite of tools and technologies to enable productivity, this includes Teams, SharePoint, OneDrive, Exchange, Azure AD and many others. The unified API endpoint for working with resources that reside in Microsoft 365 is the Microsoft Graph API. It enables developers to perform CRUD (create, read, update & delete) operations, via a REST API, to interact with Microsoft 365 resources. To introduce you to these concepts, we are going to start with the Microsoft Graph Explorer sandbox. Open a web-browser and go to [Graph Explorer - Microsoft Graph](#)
2. Once you see the Graph Explorer, there are a number of features that you need to be made aware of: Firstly, you will see that you are signed in, by default, using a sample account. This is great for getting you working with the Microsoft Graph API quickly, although if you want to do more than just retrieve data, you will need to sign in, although, there is no need for you to do that yet! Looking at number 3 in the below picture, you will see some sample queries, that the Graph Explorer team have made available for you to use out of the box, if you click on the **GET my profile** sample request, you will see section 4 change and make a request, to Microsoft Graph API, to the /me resource, via the v1.0 Endpoint. In this simple first example, you will see that the request body does not contain any data, and if you check the access token, it will be for the sample account. Section 5 contains the response code, in this case it's 200, which is expected and means the request was successful, and section 6

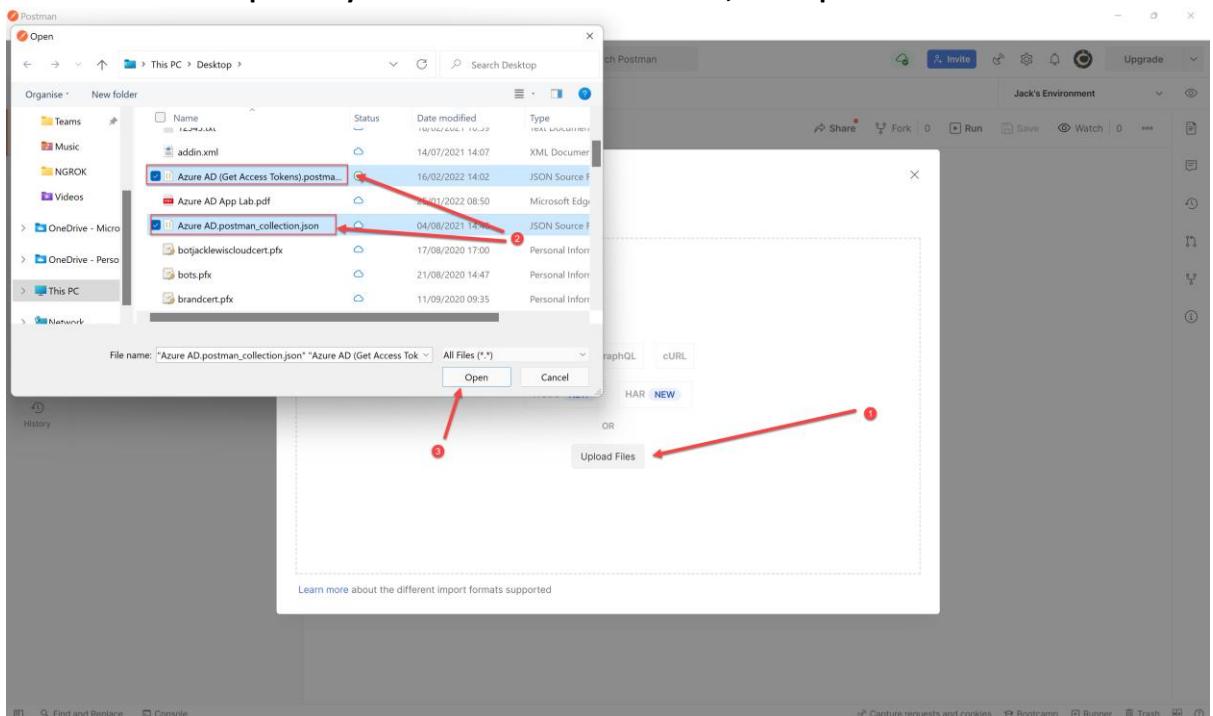
contains the response data. If you review the response, you will see that this details the information of the sample account, allowing you to discover more information about this user, such as their email address and job title. This is an example of a request that was made using a delegated permissions access token, which means it was made in the context, and on-behalf-of the logged in user (which is the sample account). If you Sign in to Graph Explorer, using your test/dev user, and make the same request, you will be asked to consent to the Graph Explorer app being granted specific scopes, and the response will detail information about your test/dev user, as the request was made, using an access token that contains delegated permission scopes. **Note: it is not possible to make a Graph API call, on behalf of an app, using Graph Explorer, only delegated permission scopes and access tokens on-behalf-of users can be used. We will make an API call to Microsoft Graph on behalf of an app, using Postman, in the next few steps, to allow you to explore the differences between app and delegated permissions.**

The screenshot shows the Microsoft Graph Explorer interface. On the left, under 'Sample queries', there is a list of endpoints. One endpoint, 'GET my profile', is highlighted with a red box and a red arrow pointing from the main text to the button. The 'Response preview' section on the right shows a JSON response for a successful 'GET https://graph.microsoft.com/v1.0/me/' request. The response includes fields like '@odata.context', 'businessPhones', 'displayName', 'givenName', 'jobTitle', 'mail', 'mobilePhone', 'officeLocation', 'preferredLanguage', and 'surname'. A green box highlights the status bar message 'OK - 200 - 1324ms'.

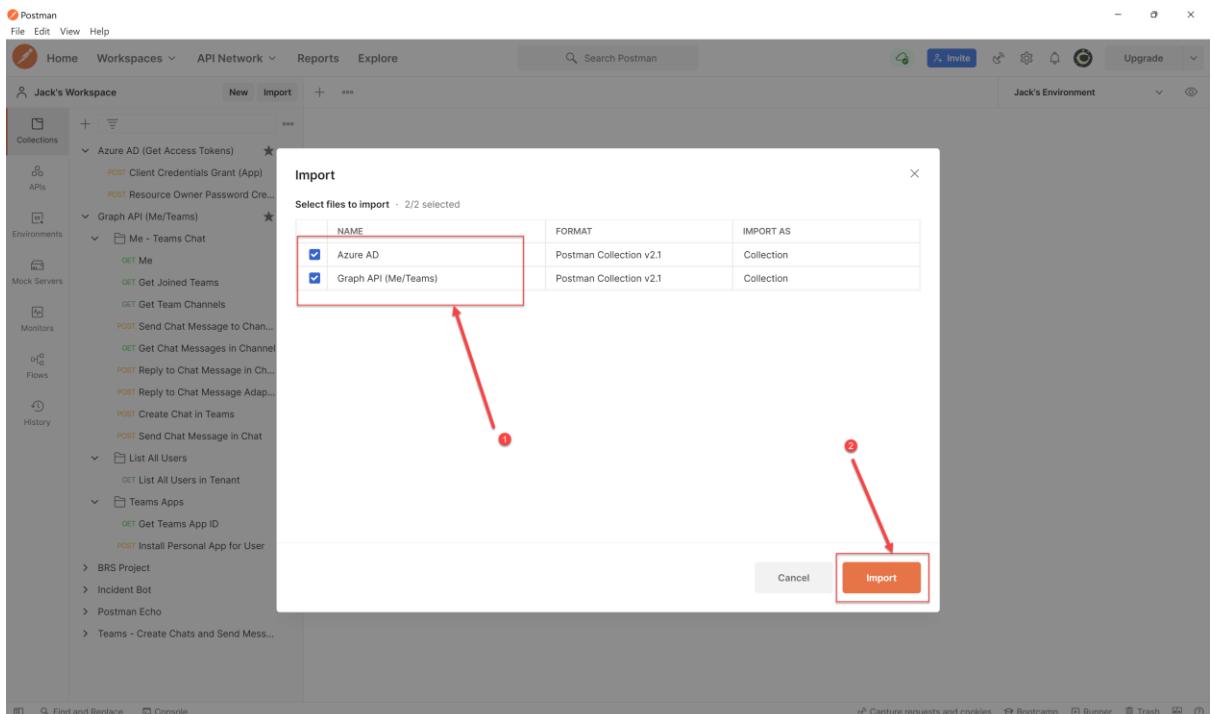
3. Continue to spend time with the Graph Explorer, using the sample queries, and when you feel comfortable with how the request/response model works for Microsoft Graph API, open Postman, which is the app you installed at the start of the labs, during the pre-reqs.
4. To import the downloaded collections into Postman, select **File**, and then click **Import...**



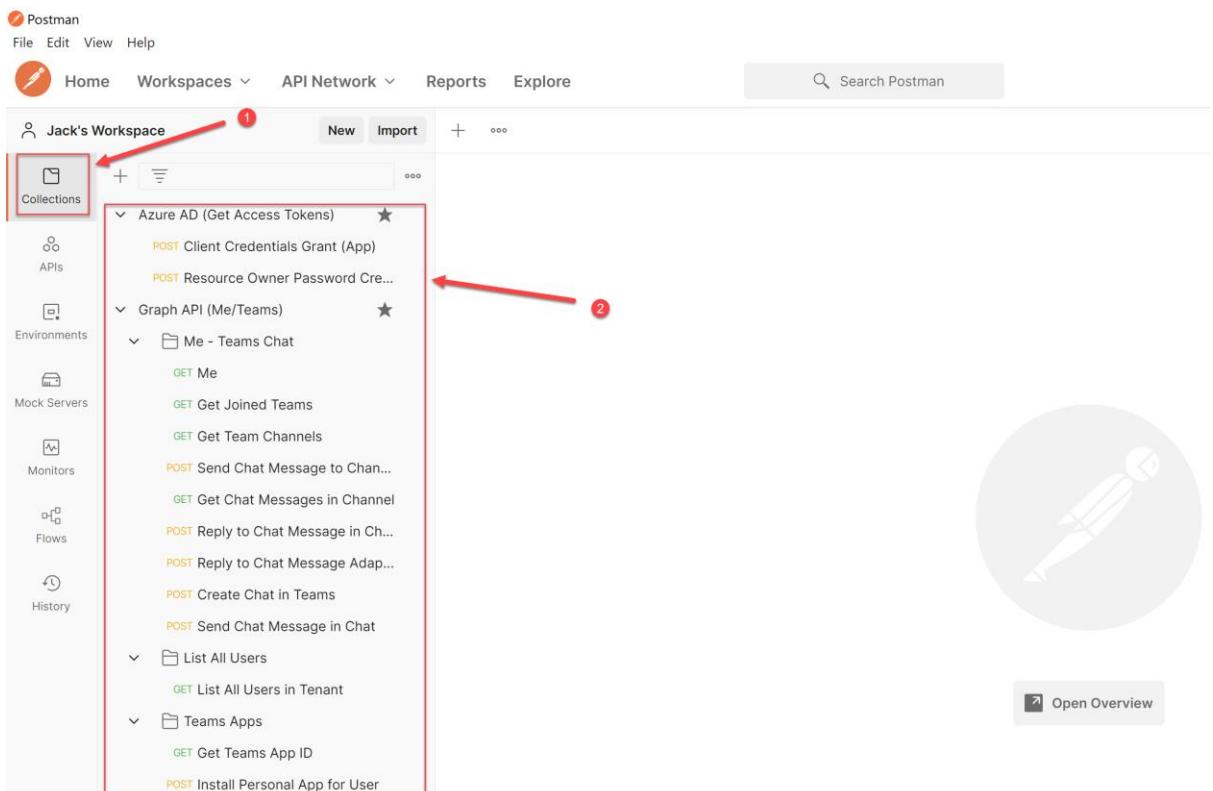
5. Select **Upload Files**, then select the 2 collections that are contained within the Postman folder within the Repo that you downloaded to run this lab, click **Open**



6. Review the names of the files that are being imported, there should be 2 collections **Azure AD & Graph API (Me/Teams)**, click **Import**



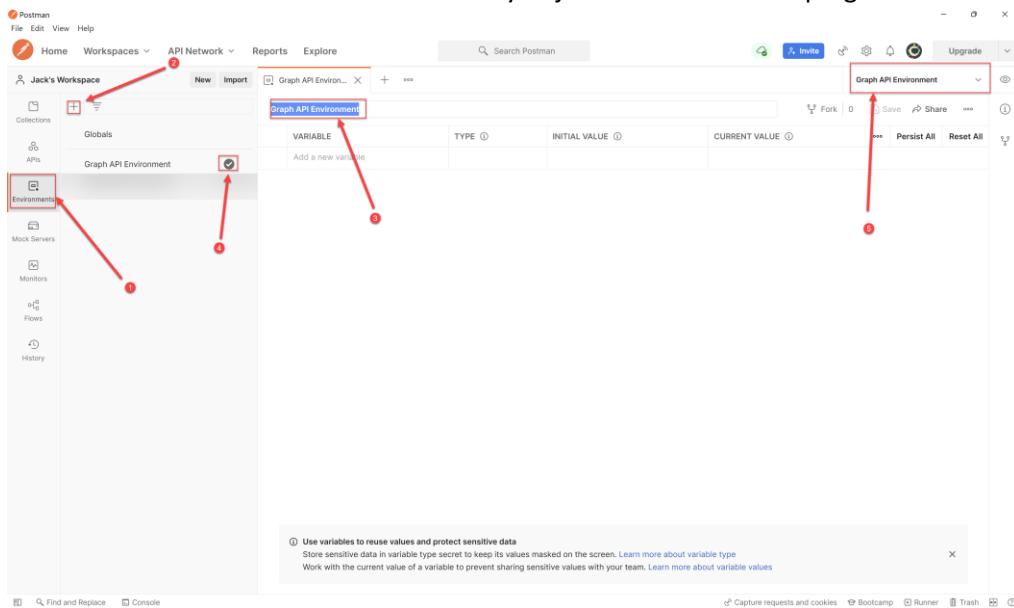
7. Ensure that **Collections** is selected in the left navigational menu. If so, you should now see the following Collections and Requests within those collections, once you have expanded the folder structure:



8. Similar to the Graph Explorer, the collections that you have just imported into Postman contain sample requests, that will allow you to interact with the Microsoft Graph API. The key differences here are that, when using Postman; we will need to get an Access Token manually, which are the 2 requests in the Azure AD (Get Access Tokens) folder, we can work with both App and Delegated Permission scope access tokens, and we can work with

variables, by provisioning an environment and populating it with variables and values. Postman also has an abundance of other features that are useful to professional developers, such as code-snippets, mock servers, flows, and many others. I would encourage all developers who want to work with the Microsoft Graph API to get familiar with this tool.

- Now that we have the collections imported into Postman, let's create an Environment and populate it with some variables we will need to use to make the requests function. Click **Environments**, select the + button and give the environment a name, such as **Graph API Environment**. Ensure that the **grey tick** is enabled to set this as the current environment. Review the environment is set to the one you just created in the top right of Postman.



- We now need to create the following variables, in this environment, and set the initial values, from the appsettings.json file, to the following:

Variable Name	Initial Value
clientID	<i>MicrosoftAppId</i>
clientSecret	<i>MicrosoftAppPassword</i>
delegateScopes	<i>email%20offline_access%20openid%20profile%20User.Read%20Team.ReadBasic.All%20Channel.ReadBasic.All%20ChannelMessage.Send%20ChannelMessage.Read.All%20Chat.ReadBasic%20ChatMessage.Read%20ChatMessage.Send</i>
tenantID	<i>MicrosoftDirectoryId</i>
delegateUserName	<i>admin@DOMAINNAME.onmicrosoft.com</i>
delegateUserPassword	<i>admin@DOMAINNAME.onmicrosoft.com's password</i>
delegateAccessTokenToken	DO NOT POPULATE WITH INITIAL VALUE (LEAVE BLANK)
appAccesssToken	DO NOT POPULATE WITH INITIAL VALUE (LEAVE BLANK)

Once complete, it should look something like this:

VARIABLE	TYPE	INITIAL VALUE	CURRENT VALUE
clientID	default	6a56bee2-c36e-4838-a0c2-7cf53be8...	
clientSecret	default	PfV	8c...
delegatedScopes	default	email%20offline_access%20openid%2...	
tenantID	default	0e0	95...
delegatedUsername	default	admin@M365x1.onmicrosoft...	
delegatedUserPassword	default	50E	P
delegatedAccessToken	default		
appAccessToken	default		

11. To commit the initial values of these variables to the current values of the variables, click **Reset All**, and you should see the **Current Value** column populate with data. Click **Save**

VARIABLE	TYPE	INITIAL VALUE	CURRENT VALUE
clientID	default	6a56bee2-c36e-4838-a0c2-7cf53be8...	6a56bee2-c36e-4838-a0c2-7cf53be8bcd0
clientSecret	default	PfV	P
delegatedScopes	default	email%20offline_access%20openid%2...	email%20offline_access%20openid%20profile%20U...
tenantID	default	0e0c	0e0d6
delegatedUsername	default	admin@l.onmicrosoft...	admin@t.onmicrosoft.com
delegatedUserPassword	default	50E	50
delegatedAccessToken	default		
appAccessToken	default		

12. Once the variables have been populated and committed to the current values, we now need to go and get an Access Token from Azure Active Directory, that will work with the Microsoft Graph API. We'll start first with getting an access token that contains App Permission scopes (roles). To get an App Permission scope access token, you need to use the Client Credentials Grant Flow with Azure AD, (more details on this grant flow can be found here - [OAuth 2.0 client credentials flow on the Microsoft identity platform | Microsoft Docs](#)). Essentially, this flow requires you to send a POST HTTP request, to a specific API Endpoint, with the TenantID of the tenant you want to receive an app token for, in the URL of the request. In the body of the request, details referring to your Azure AD App Registration needs to be submitted, such as the MicrosoftAppID (client_id), the scope and the MicrosoftAppPassword (client_secret). This should then return an access token, with app permission scopes (roles), assuming the correct consent is in place. Click **Collections**, then select the **Client Credentials Grant (App)** request. Review the **HTTP Request Type of POST and the URL**, this is using the tenantID variable you provisioned in the previous step as part of the URL. Review the **Body of the request**, this should contain **client_id, scope, client_secret and grant_type**. Variables are used for the client_id & client_secret values, but for scope, we have left it hard-coded in this request, and set it to <https://graph.microsoft.com/.default>. When working with app permissions, you must use the /.default scope, which essentially asks for the scopes that are

listed in your App Registration, which you configured in section 1 of this lab. Once you have reviewed these settings, hit **Send** to make the request. All being well, you should see a response status code of **200 OK**, and the response, will contain an access_token. We will now set the variable of appAccessToken to this access token, and review this access token to see what it contains.

The screenshot shows the Postman interface with the following details:

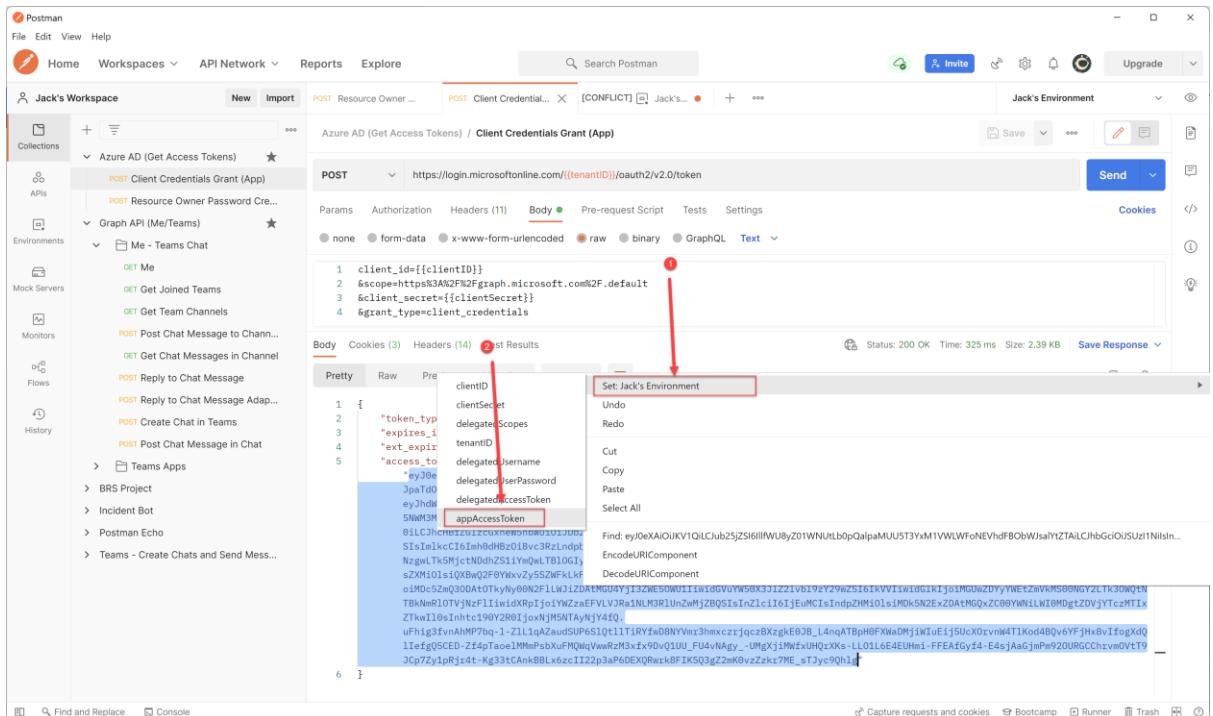
- Left Sidebar:** Shows "Jack's Workspace" with various collections and environments. A red box highlights the "Collections" section.
- Request Details:**
 - Method: POST
 - URL: https://login.microsoftonline.com/([tenantID])/oauth2/v2.0/token
 - Headers: Authorization, Content-Type
 - Body (raw JSON):

```
1 client_id=[clientID]
2 scope=https%3A%2F%2Fgraph.microsoft.com%2F.default
3 &client_secret=[clientSecret];
4 &grant_type=client_credentials
```
- Response:**
 - Status: 200 OK
 - Time: 325 ms
 - Size: 2.39 KB
 - Content (Pretty):

```
1 {
2   "token_type": "Bearer",
3   "expires_in": 3599,
4   "ext_expires_in": 3599,
5   "access_token":
6     "eyJ0eXAiOiJKV1QiLCJub25jZSI6I1lfWU8yZ01wNUtLb8pQalpaMU5T3YxM1VWLhfOEVhdFB0bW0salYtZTAiLCJhbGciOiJSUzI1NiIsIngldC16Ik1yNS1BVWlizK
7     JCs7Zy1pRjz4t-Kg33tCAnkBBLx6zcII2p3aP6DEXQRwzBFIK5Q3gZ2mK0vzzkz7ME_sTJyc9Qhlg"
```

NOTE: If you receive an error when making the request of 'unauthorized_client', this means that consent for the app permission scopes, and the creation of the Enterprise app in the tenant you are requesting the token for, has not been completed. Go back to step 25, in section 1, of this lab document to go through the consent process, and solve this issue. Once the consent has been granted, try sending the request again in Postman and you should receive an access token in the response.

- Select all of the value of the access_token in the response (everything in between the quotation marks) and right click, highlight the option to Set: **Graph API Environment** and select **appAccessToken**. This will set the variable value to this access token.



14. **Copy the value of the access_token, in the response, to your clipboard, and open <https://jwt.ms> in your preferred web browser client. Paste the value of the access token into the form.** You can now review the claims that are embedded within the decoded access token, the most important are:

Aud: This should be The Microsoft Graph API

App_displayname & appid: Should match the name and Client ID of your App Registration
Poles: These are the scopes that are included in the access token, that your customer has

Roles: These are the scopes that are included in the access token, that your customer has consented to. Roles signify that these are application permission type scopes (delegated scopes are included in the ‘scp’ claim)

Tid: The tenant ID that this token is allowed to be used within

jwt.ms

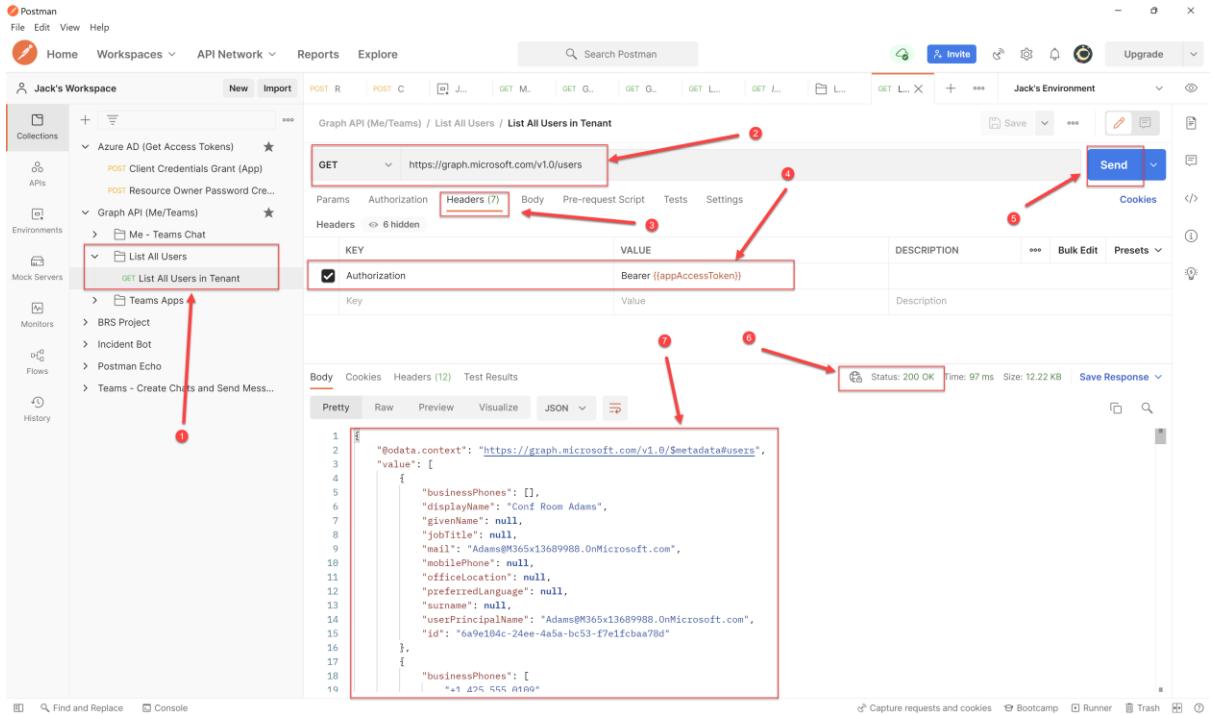
Enter token below (it never leaves your browser):

```
eyJ0eXAiOiJKV1QiLCJub25jZS1G1ll1fW8yZ01WNutlb0pQalpaMUU5T3YXm1VWLwFoNEVhdFB0bwJsalYtZtaILCjhGc1o1JSUzI1NiIsIngldCI6Ik1yN51BwI1zKjpa
d0ZDFq0mViYXhibi1hXMCisImtpzC161k1yN51BVwlzKjpaTd0ZDfqMviYXhibi1hXMCj9.eyJhdwQio1jodHRwczovL2dyYXBoLm1p33vc29mdc5jb201Lcpc3Mio1jodHR
wcZovl3N0cy53aw5kb3dLm51d8wZTBkWj3hYS1mZQxLQ0ZjvtOTc5ZC01MG02ZGUJNwM3MwUvIiwiabiF0joxNjQ1MDyENToxLcJuYmY1oJE2NDuMTI1NDEsImV4C16M
TV0NTaxhjQ0mSw1yV1jo1RTJaZ11Z3Mrdkh11zBhK3V0TgZjv0BXTJ40UN3Q01LcJhcHBFZGccZgxne0vhbwU1O1JD02503NV1Fd1y1BcHA1LcJhBpZC161jZHTNZ
izWMy1Nw1zmuNTDgZ0C1HMM0MltjdjZu1yM4yMhKNC1sImfWcG1kYMyNy1joiMSisIm1kC161mh0dHBz018v3R2LndpbmRv3Mu0mV0LzB1MQ2PmFLNLZDETNDRmN105N
z1kLTu2DZk2Tk1Y3wzZ58LLcPjZH86c16ImFwC1sIm9pZC161jA30wZKNgwLTk5MjctDdh2S11yQmLTBL06y1N2vH0T1n51sIn1o1j01MCSVTbCw17TRk0SC05e1n
Yb12EVzhwNmEH1z01BQF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0QUF0
VN1bg2Gb3Vc2Vlykfbc1s1lVzX1uUmhZC5BhGwiXSwic3V1i1j01Mdc5ZmQ300At0Tykyly00N2f1w1Z04TMGUAyj13ZWE50w11iwidGVuW50X33121vbl9zY29wZSI
61kV1i1widG1k1j01MuGwZDyYwEtZmVks0NGY2L1k30QntBkNmR107vJNzF11iwidXRp1j01yWzaEFVLVJra1NLm3R1u2mMjZBQ5Si1nLci16IjEuMCi1nDpZHM10
1s1MDk52E2xDATM0QzC00ywN1LwI0MdgtZDvYtCzMT1xZTkW1l0sInhtc190Y2R01joxNjMSNTAyNjY4F0.uHig3fvnAHP7bq-1-Z1LiqAzaudSUP651Qt11T1RYfW0BN
YVm3hmczrjzcBZxgkE03B_L4ngATBpH0FxWaDmj1n1u1j5uc0rvnW4T1Kod4Bqv6YFjHx8vIfogxd1IefgQ5CED-Zf4pTaee1MmPsbxUFM0NqVwWzRm3xf9DvQ1UU_
FU4vNagy_-UMgxjiMwFxUhQrxXs-LL01L6E4EUhmi-FFEAfGyF4-E4sJaaGjmPm920URGCChrv0Vt9Jcp7z1phjr4t-Kg33tCAnkBBLx6zcII22p3aP6DEXQRwrk8FIK5Q
g22mK0vzZkr7ME-sT3yc9qlg
```

This token was issued by Azure Active Directory.

Decoded Token	Claims
<pre>{ "typ": "JWT", "nonce": "Y_02gMVS5KKoJPjZZ1E90v13Uv-ah4EatPNmb1jv-e0", "alg": "RS256", "x5t": "Mm5-AUibFBii7Nd1jBebabx0X0", "kid": "Mm5-AUibFBii7Nd1jBebabx0X0" }.{ "aud": "https://graph.microsoft.com", "iss": "https://sts.windows.net/0e0d62aa-fed1-44f6-979d-50d6de95c71e", "iat": 1645012541, "nbf": 1645012541, "exp": 1645016441, "aio": "E2ZgYIgs+vHe/0a7utLfjTPwMrx9CwA=", "app_displayname": "Contoso Web App", "appid": "6a56bee2-c36e-4838-aec2-7cf53be8bcd0", "appidacr": "1", "idp": "https://sts.windows.net/0e0d62aa-fed1-44f6-979d-50d6de95c71e/", "idtyp": "app", "oid": "079fd780-9927-47ae-bbd0-0e8b27ea99e5", "rh": "0..AU8Aqm1NDTH-9kSXnVDW3pxHtgMAAAAAAAAAAAwAAAAAAAABPAAA..", "roles": ["AppCatalog.Read.All", "TeamsAppInstallation.ReadWriteSelfForUser.All", "User.Read.All"], "sub": "079fd780-9927-47ae-bbd0-0e8b27ea99e5", "tenant_region_scope": "EU", "tid": "0e0d62aa-fed1-44f6-979d-50d6de95c71e", "ut1": "afshAU-RQkSK3teRvp26AA", "ver": "1.0", "wids": ["0997a1d0-0d1d-4acb-b408-d5ca73121e90"], "xms_tcdt": 1639502668 }.[Signature]</pre>	<p>①</p> <p>②</p> <p>③</p> <p>④</p> <p>⑤</p>

15. Now that we have reviewed what's included in the decoded application permissions access token, we can use it to make requests to the Microsoft Graph API. Firstly though it's worth noting that application permission type scopes are usually highly privileged, and are often not scoped to subsets of resources (some exceptions to this rule do exist, such as app policies that can restrict access to resources in Exchange for example). For that reason, we need to be wary that it is, in most scenarios, preferable to use delegated access tokens where possible, as this will require less privileges to perform actions. There are scenarios though, where it doesn't make sense to use delegated, and where application access tokens must be used, such as if users never log into your application, or if you need organisation/tenant-wide permissions to perform actions. **Go back into Postman**, click on the **List All Users in Tenant** request under the List All Users directory. Review the URL and HTTP Request Type. Select **Headers**, and review the **Authorization header**, this is using the `appAccessToken` variable as the Bearer token for authentication with the Microsoft Graph API. Hit **Send**, and if successful, you will receive a **200 OK response code**. If you review the Body of the response, you will see that all of the users of the tenant have been sent back in the response. This is a useful request when working with Teams proactive notification bots (which we will in the next section) as we need to discover the users in a tenant, and then see if they have our Bot installed, to allow our app to send them a proactive message for notification purposes. We will review this code in Visual Studio later.



16. In the collection you downloaded and imported, there are a selection of sample requests that are designed to work with delegated permission type access tokens. In order to retrieve a delegated permission type access token, with specific scopes, we can use the Resource Owner Password Credentials Grant (more information on this here - [Sign in with resource owner password credentials grant - Microsoft identity platform | Microsoft Docs](#)) for test/dev purposes. In production scenarios, this grant flow type should **never** be used, as more secure alternatives exist. This grant type allows you to pass the username and the password of a user to Azure AD, using the Token Endpoint APIs, to retrieve an access token for that specific user, with delegated permission scopes in the access token. Select the **Resource Owner Password Credentials (Delegated)** request, review the HTTP Request Type of **POST** and the API endpoint URL, in this scenario we do not need to specify the TenantID, instead we can use the **organizations** endpoint, as the App Registration we created is multi-tenant, and the tenantId can be retrieved from the user, when the request is made to Azure AD. Review the **Body**, you will see that the following: client_id is the MicrosoftAppID, client_secret is the MicrosoftAppPassword, scopes contains a list of the scopes we want within the delegated permissions type access token, this is different to working with app permission type access tokens, as you can define what scopes you want in the request, not just in the app registration, although it makes sense to keep the scopes in the app reg and the request consistent in most scenarios (best practices state that the app registration should contain a superset of all the permissions your app will ever request, but in each request you can ask for a subset of those permissions). In addition to this, we are also passing the username and password of the user we are authenticating as (**note: this grant type does not work for users who have MFA enabled!**) and finally, the grant_type of password is defined. Once you have reviewed the body, hit **Send**, if successful you will see a status response code of **200 OK**. Review the body of the response, this should contain the **scopes** and the **access_token** that we can use to make calls to Graph API using delegated permission types, on behalf of a user, rather than an app.

17. Select all of the value of the `access_token` in the response (everything in between the quotation marks) and right click, highlight the option to Set: **Graph API Environment** and select **delegatedAccessToken**. This will set the variable value to this access token.

18. Copy the value of the `access_token`, in the response, to your clipboard, and open <https://jwt.ms> in your preferred web browser client. Paste the value of the access token into the form. You can now review the claims that are embedded within the decoded access token, the most important are:

Aud: This GUID is a well-known GUID and represents the Microsoft Graph API

App_displayname & appid: Should match the name and Client ID of your App Registration

Name & oid: This is the name of the user and the unique (globally) ID of the user. Most third-party applications will scope the OID of the user, when they do identity matching, as this is guaranteed to be unique across Azure AD, and will remain consistent, regardless of email changes, surname changes, etc...

Scp: These are the scopes that are included in the access token, that your customer has consented to. Scp signifies that these are delegation permission type scopes (application scopes are included in the 'roles' claim)

Tid: The tenant ID that this token is allowed to be used within

Unique_name and UPN: UPN is often used to match an Azure AD user, to an existing user in a third-party app, during the first authentication the user performs through Azure AD to login to the third-party app. After this initial identity match has been performed, it is recommended that OID and TID are used to match the user for subsequent authentications.

jwt.ms

Enter token below (it never leaves your browser)

This token was issued by Azure Active Directory

Decoded Token Claims

```
{  
  "typ": "JWT",  
  "nonce": "XoPYVvaKHHqGdZHodEe1xB_U3LaJ2X95r9t_9b5TY10w",  
  "alg": "RS256",  
  "x5t": "Mr5-AUiFBii7Nd1jBebaxbxW0",  
  "kid": "Mr5-AUiFBii7Nd1jBebaxbxW0"  
}.{  
  "aud": "00000003-0000-0000-0000-000000000000",  
  "iss": "https://sts.windows.net/000d62aa-fed1-44f6-979d-50d6de95c71e/",  
  "iat": 1645812756,  
  "nbf": 1645812756,  
  "exp": 1645816676,  
  "acct": 0,  
  "acr": "1",  
  "aio": "ASQAZ2/8TAAAQfqPKE3H0odwQMit9EbXxW0icxb9TovFzFQ+2FlrQlq4=",  
  "amr": [  
    "pwd"  
  ],  
  "app_displayname": "Contoso Web App",  
  "appid": "6a56bee2-c36e-4838-a0c2-7cf53be8bcd0",  
  "appidacr": "1",  
  "family_name": "Administrator",  
  "given_name": "MOD",  
  "idtyp": "user",  
  "ipaddr": "94.11.211.52",  
  "name": "MOD Administrator",  
  "oid": "2d5c91fa-4dde-4cd4-acbf-359ede2758c5",  
  "platf": "14",  
  "puid": "10032001C014AAB1",  
  "rh": "0.AUBAgmI9KSNvDh3pxHgMAAAAAAAAIAAAAAAAAAAAAPAOY.",  
  "scp": "Channel.ReadBasic.All ChannelMessage.Read.All ChannelMessage.Send Chat.ReadBasic ChatMessage.Read ChatMessage.Send email  
openid profile Team.ReadBasic.All User.Read",  
  "sub": "3c3e9Vks3jkhB32zrtb6h5j3NzBM3fdKwIxQuMd1",  
  "tenant_region_scope": "EU",  
  "tid": "0e0d62aa-fed1-44f6-979d-50d6de95c71e",  
  "unique_name": "admin@365x13689988.onmicrosoft.com",  
  "upn": "admin@365x13689988.onmicrosoft.com",  
  "utb": "322kSgMbVu6GnwgrCDCA",  
  "ver": "1.0",  
  "wids": [  
    "62e09394-69f5-4237-9190-012177145e10",  
    "79fbffad-3ef9-4689-8143-76b194e85509"  
  ],  
  "xms_st": {  
    "sub": "n80dyWZPGBuHf05HnfgaEtws3m6LTxsgmDw1zRVw0U"  
  },  
  "xms_tcdt": 1639502668  
}. [Signature]
```

<https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-token-and-claims>

19. Now, let's head back to Postman, and make a simple request to the ME resource in Graph API, using a delegated type access token for authentication. Open the **Me** request, found within the 'Graph API (Me/Teams)\Me – Teams Chat' directory. Review the HTTP Request type of **GET** and the URL of the API endpoint **v1.0/me**. As we are using a delegated access token, no further information is required in the body of the request, but we do need to authenticate, which is passing a Bearer Token in the headers, click **Headers** and review the **Authorization** header to see how this uses the {{delegatedAccessToken}} variable to

authenticate with Graph API. Once you are happy that the request is ready, hit **Send**. If the request is successful, you will receive a **200 OK** status response code, and information about your user will be displayed in the **response body**.

The screenshot shows the Postman interface with a successful API call. The request URL is `https://graph.microsoft.com/v1.0/me`. The Headers tab shows an `Authorization` header with the value `Bearer {{delegatedAccessToken}}`. The response body is a JSON object containing user details:

```

1  "@odata.context": "https://graph.microsoft.com/v1.0/$metadata#users/$entity",
2  "businessPhones": [
3    "425-555-0100"
4  ],
5  "displayName": "MOD Administrator",
6  "givenName": "MOD",
7  "jobTitle": null,
8  "mail": "admin@M365x1368988.OnMicrosoft.com",
9  "mobilePhone": "425-555-0101",
10  "officeLocation": null,
11  "preferredLanguage": "en-US",
12  "surname": "Administrator",
13  "userPrincipalName": "admin@M365x1368988.onmicrosoft.com",
14  "id": "2d5c91fa-4cd4-acbf-359ede2758c5"
15
16
  
```

20. Now that you have been introduced to these Graph API and Azure AD concepts, continue to work through the ‘Graph API (Me/Teams)\Me – Teams Chat’ directory, to get the Teams the user is a member of, discover the channels of a specific Team, and send a message into the conversation associated with that channel. You can then also continue to work with the requests in that directory to start a 1-2-1 or group chat in Teams, and then send messages, as that user, into those chats.

Nice one! Now that you are familiar with the key concepts associated with the Microsoft Graph API, we will now continue working on the Talent Management App, and review how the app uses the Graph API to install the App, that contains the bot, programmatically, retrieve chat/conversationIDs between the Bot and the Users, and then use this information to proactively message users with an adaptive card.

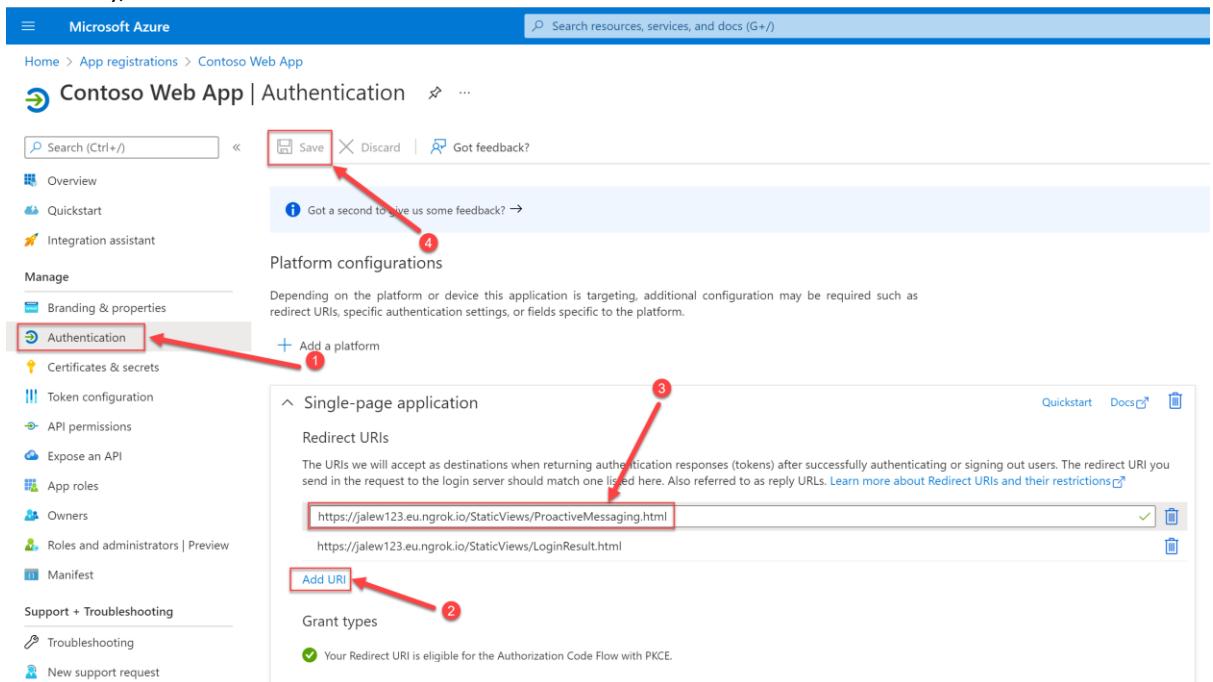
8) Proactive Notifications in Teams (via the Teams Bot)

One of the most popular use cases for apps in Teams is the ability for you to notify an individual in Teams when an event happens in your third-party app. To enable this, Teams supports Proactive Notifications/Messaging. Proactive Messaging in Teams, can be done in a number of different ways; Graph API, Connectors & Bots. There are pros and cons to each approach, but in our experience, Bots offer the most functionality/flexibility, as they support 2 way communications and actionable notifications, but the trade off is they require the most code/configuration to get them to work, and also require that your Teams app is installed into the place where they want to send that notification (for example, for a bot to send a message to a specific user, the app that contains the bot, must be installed for that user – or if the bot wants to message a Teams channel, the app that contains the bot, must be installed into that Team, before it can message the channel).

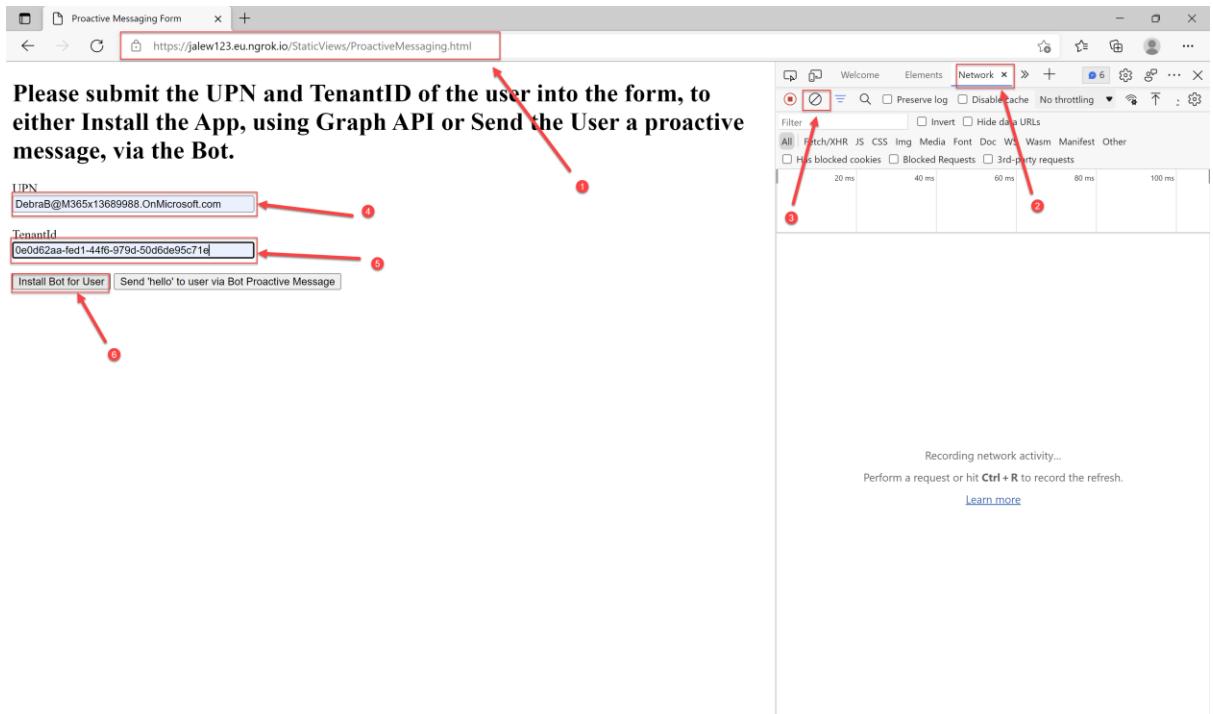
The other 2 options (Graph API and Connectors) can be used but lack the ability for you to add actionable buttons onto the adaptive cards that you send, as such, we define them as ‘one way notifications’. The trade off is that they are easier and quicker to get up and running.

As we want to demonstrate the preferred way of doing this, we are going to demonstrate how you can use bots, to proactively message individuals, which also requires some Graph API calls to be made, to discover if the app is installed in the scope you want it to message, and if it isn’t, install it via Graph API (meaning you are going to build on the Graph API knowledge you gained in the previous step)! Once all that is in place, it will be the Bot that sends the message/adaptive card to the user.

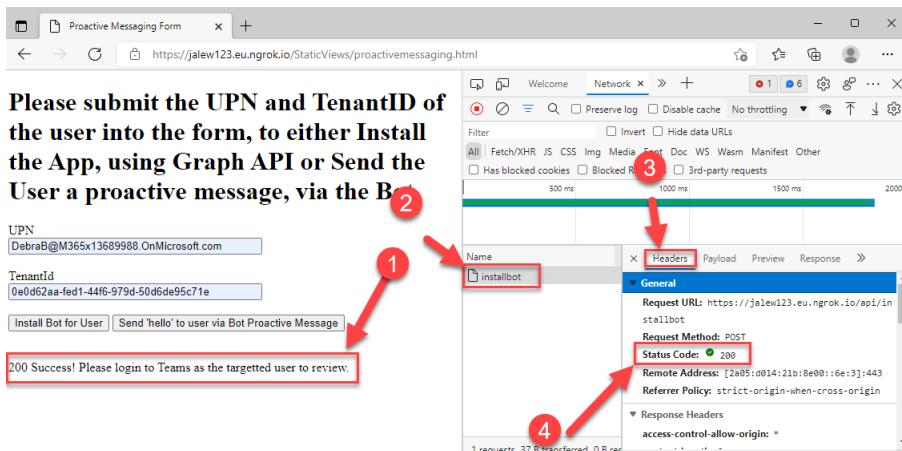
1. Before we begin with the proactive messaging, we need to add another Redirect URI to our Azure AD App Registration. Go to <https://portal.azure.com> and **navigate to your App Registration**. Select **Authentication**, then click **Add URI** and enter your URI as: <https://NGROKURL/StaticViews/ProactiveMessaging.html> (be careful, as this is case sensitive), and then click **Save**.



2. Then go to <https://NGROKURL/StaticViews/ProactiveMessaging.html> in a new browser tab. This is a simple page, that allows you to submit the UPN (or Alias) of a user, and their Tenant ID, to the app’s APIs, to get your app to proactively install your Teams app for that user, or proactively message that user, via the Bot, with a simple ‘hello’ message. Press **F12** to open the developer tools, in the browser, and then select the **Network** tools, hit the **Clear Network Log** button. Enter the **UPN of a user who does not yet have your app installed (for example DebraB@YOURTENANTDOMAIN.ONMICROSOFT.COM)** and enter the **TenantID** where that user resides. Click **Install Bot for User**.



3. A request will now be made to the appropriate API endpoint controller (in this case /api/installbot) with the UPN and TenantID submitted to that API endpoint in the body of the request. There are now a number of different things that could happen within the app, and you should receive an appropriate response to notify you of if the request was successful or not. **If the request isn't successful, follow the guidance that is displayed on the screen, under the button you clicked to Install the Bot for the User, and use the network developer tools to help troubleshoot the issue.** The reasons for failure vary from not having the correct consent, to your request being incorrect (maybe you sent the wrong UPN or TenantID), although you should be able to troubleshoot this by reviewing the response information, associated with the request that was made to the endpoint controller, from within the Network developer tools in the browser. **If the Install is successful, you will receive a 200 Success message, and can now move onto the next step in this section of the lab.** Here is an example of a successful request:



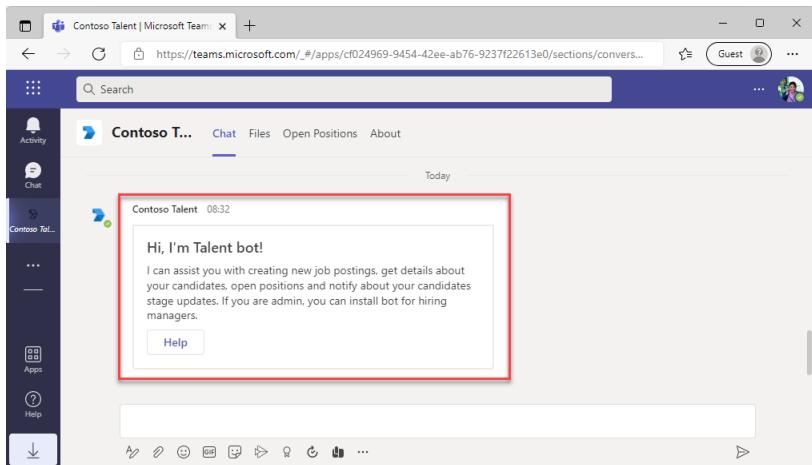
Here is an example screen of a failed request that I am troubleshooting: In this scenario, the issue is an easy fix, as you can just click the button to grant Admin Consent, so that your App has the right authorisation to install the bot for that user:

The screenshot shows a browser window with two tabs: "Proactive Messaging Form" and "Enterprise applications - Microsoft". The main content area displays a form with fields for UPN (DebraB@M365x13689988.OnMicrosoft.com) and TenantId (0e0d62aa-fed1-44f6-979d-50d6de95c71e). Below the form are two buttons: "Install Bot for User" and "Send 'hello' to user via Bot Proactive Message". A red box highlights the "Grant Consent" button in a tooltip message: "403 Forbidden error - Consent must be granted to allow Proactive Install/Messaging. Please click the below button to grant Consent and then try again." The Network tab in the developer tools shows a single request to "installbot" with a status of "Forbidden". Red arrows point from the "Grant Consent" button to the tooltip message, and another arrow points from the tooltip message to the "installbot" entry in the Network tab.

- If you received the 200 Success message response, you should now open a InPrivate/Incognito web browser session, and go to <https://teams.microsoft.com> and log in as the user who you targeted the bot install to.
 - Let's now go back to the Proactive Messaging HTML page, and click the **Send 'hello' to user via Bot Proactive Message**. You can use the same methods as described in step 3 of this section of the lab to troubleshoot, but assuming you receive a **200 Success message**, you can **Go back into Teams as the targeted user**, click the ellipsis ... and you should see the **Contoso Talent Management App** installed for the user, with a notification. Click on the **Contoso Talent Management app**.

The screenshot shows the Microsoft Teams interface. On the left, the sidebar displays pinned apps: Approvals, Contoso Talent (pinned with a red box and a red arrow labeled 2), OneNote, Shifts, Stream, Tasks by Planner, Viva Insights, and Viva Learning. Below these are three more pinned apps: WPS Office (with a red box and a red arrow labeled 1), Microsoft Edge, and Microsoft Word. A 'More apps' button is visible at the bottom of the pinned list. The main content area shows a pinned post from 'MOD Administrator' dated 13/01, 12:17, which is a job listing for an IT Admin. Another post from the same user dated 11/02, 13:51, is also visible. The bottom navigation bar includes 'New' and '3 replies from MOD'.

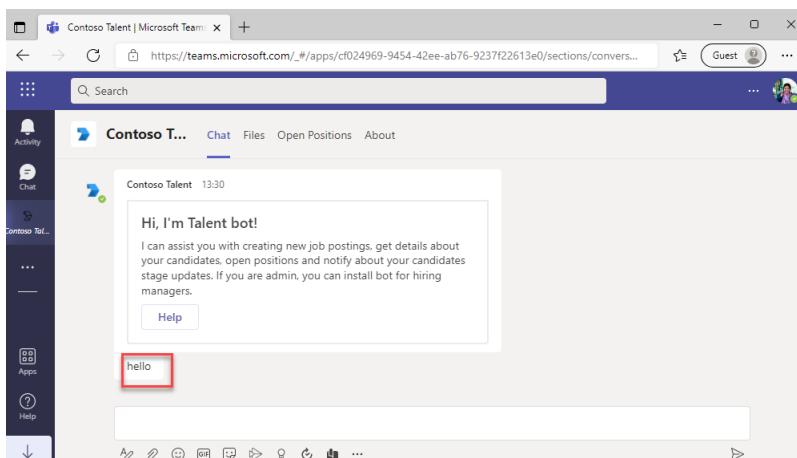
- ## 6. You should see a Welcome Message from the Bot.



7. Navigate back to the **ProactiveMessaging.html** page and click the **Send 'hello' to user via Bot Proactive Message**. If you review the **Network developer tools**, you will see a request being sent to the **notify** bot controller API. If the request is successful, you will receive a **202 Status** in the response to the request.

The screenshot shows a browser window with the URL <https://jalew123.eu.ngrok.io/StaticViews/proactivemessaging.html>. The page contains a form with fields for UPN (DebraB@M365x13689988 OnMicrosoft.com) and TenantId (0e0d62aa-fed1-44f6-979d-50d5de95c71e). Below the form are two buttons: 'Install Bot for User' and 'Send 'hello' to user via Bot Proactive Message'. A red box highlights the 'Send' button, labeled 1. To the right, the Network developer tools show a request to 'https://jalew123.eu.ngrok.io/api/notify' with a POST method and a status code of 202, labeled 2. A red box highlights the status code, labeled 3. A red arrow points from the 'Send' button to the status code. At the bottom of the browser window, a message says '202 Success! Please login to Teams as the targetted user to review.', labeled 4.

8. Navigate back into Teams, as the targeted User, and you should now see a message saying '**Hello**' from the Contoso Talent Bot.



9. Let's review some of the code in Visual Studio, to see what's happening within the API Controller, and what happens once the API controller receives a request, with the expected information it requires to perform a Bot Install and to send a Proactive Message to a User in Teams. Open **Visual Studio** and navigate to the **BotController.cs** file. You will notice there are two endpoints that handle the installation and notification requests named **InstallAsync** and **NotifyAsync** respectively.

```

[HttpPost]
[Route("api/installbot")]
public async Task<IActionResult> InstallAsync(
    [FromBody] UserTenantMessageRequest request,
    CancellationToken cancellationToken) ...

[HttpPost]
[Route("api/notify")]
public async Task<IActionResult> NotifyAsync(
    [FromBody] NotifyRequest request,
    CancellationToken cancellationToken) ...

```

10. Let's first look at **InstallAsync**. This method will attempt to install the bot for the user and then return the installation result in the response.
11. Installation is handled by **GraphAPI** and is defined in the **Graph ApiService.cs** file in a method called **InstallBotForUser**. This method goes through the following steps to install the bot for the specified user:

```

public async Task<InstallResult> InstallBotForUser(
    string aliasUpnOrOid,
    string tenantId,
    CancellationToken cancellationToken)

1 var token = await GetTokenForApp(tenantId);
var upn = await GetUpnFromAlias(token, aliasUpnOrOid, cancellationToken);

if (upn == null)
{
    return InstallResult.AliasNotFound;
}

var graphClient = GetGraphServiceClient(token);

2 var teamsApps = await graphClient.AppCatalogs.TeamsApps
    .Request()
    .Filter($"distributionMethod eq 'organization' and externalId eq '{_appSettings.TeamsAppId}'")
    .GetAsync(cancellationToken);

var teamApp = teamsApps.FirstOrDefault();
var success = false;
if (!string.IsNullOrEmpty(teamApp?.Id))
{
    try
    {
        3 var installBotRequest = new BaseRequest($"https://graph.microsoft.com/v1.0/users/{upn}/teamwork/installedApps", graphClient)
        {
            Method = HttpMethods.POST,
            ContentType = MediaTypeNames.Application.Json
        };

        await installBotRequest.SendAsync(
            new TeamsAppInstallation
            {
                AdditionalData = new Dictionary<string, object>
                {
                    { "teamsApp@odata.bind", $"https://graph.microsoft.com/v1.0/appCatalogs/teamsApps/{teamApp.Id}" }
                }
            }, cancellationToken);

        // Getting the chat id will confirm the installation was successful and send the welcome message
        await GetProactiveChatIdForUserInternal(token, aliasUpnOrOid, tenantId, cancellationToken);
        success = true;
    }
}

```

1. Gets an application token containing the application scopes defined in our App Registration created earlier. If an alias was passed in the request rather than a full UPN, it then attempts to lookup the UPN based on the alias.
2. Assuming this was successful, we then must find the Teams app installation ID for your application. (*Note: This is not the same as the Teams App ID that's contained in the manifest, each tenant will have its own identifier for your application stored against the Teams App ID*). We look this up using the [appCatalogs/teamsApps](#)

- GraphAPI endpoint and the ODATA filter: **distributionMethod eq 'organization' and externalId eq 'Your Teams App ID'**.
3. Once this is found, we then install the application using the [teamwork/installedApps](#) GraphAPI endpoint.
 4. Finally, we request the chat ID for conversation between the user and the bot to confirm the app was installed and to prompt Teams to send the user a welcome message if this was the first time the app was installed.
 12. Let's now look at **NotifyAsync** in **BotController.cs** and see how proactive notification is handled by the bot. Like with the installation process, we call off to another service to handle the actual notification logic and the rest of the controller action is handling the request and response.
 13. Open **NotificationService.cs** and find the **SendProactiveNotification** method. This is where the "magic" to find the conversation between the bot and the user exists before actually sending the notification message. The goal of this process is to get a **TurnContext** that represents the conversation between the bot and the user and is reasonably involved!

```

public async Task<NotificationResult> SendProactiveNotification(string aliasUpnOrOid, string tenantId, IActivity activity, CancellationToken cancellationToken)
{
    1 var (upn, chatId) = await _graph ApiService.GetProactiveChatIdForUser(aliasUpnOrOid, tenantId, cancellationToken);

    if (upn == null)
    {
        return NotificationResult.AliasNotFound;
    }

    if (chatId == null)
    {
        return NotificationResult.BotNotInstalled;
    }

    var credentials = new MicrosoftAppCredentials(_appSettings.MicrosoftAppId, _appSettings.MicrosoftAppPassword);
    2 var connectorClient = new ConnectorClient(new Uri(_appSettings.ServiceUrl), credentials);

    var members = await connectorClient.Conversations.GetConversationMembersAsync(chatId);

    var conversationParameters = new ConversationParameters
    {
        IsGroup = false,
        Bot = new ChannelAccount
        {
            Id = "28:" + credentials.MicrosoftAppId
        },
        Members = new ChannelAccount[] { members[0] },
        TenantId = tenantId,
    };

    3 await ((CloudAdapter)_adapter).CreateConversationAsync(credentials.MicrosoftAppId, null, _appSettings.ServiceUrl, credentials.OAuthScope, conversationParameters);
    {
        var conversationReference = t1.Activity.GetConversationReference();
        await ((CloudAdapter)_adapter).ContinueConversationAsync(credentials.MicrosoftAppId, conversationReference, async (t2, c2) =>
        {
            await t2.SendActivityAsync(activity, c2);
        }, cancellationToken);
    }, cancellationToken;

    return NotificationResult.Success;
}

```

1. First it gets the chat ID for the conversation between the bot and the user. This will be used to get the internal ID for the chat members, in our case this will be the user we want to send the message to.
2. Then it creates a **ConnectorClient** and **ConversationParameters** to tell the bot service who the message is to (using the internal id we found above) and from (our bot). **Note:** *By convention, bot service internal identifiers are usually a number followed by and encoded value. In the case of a bot id, it's the number 28 followed by our Microsoft App Id.*
3. Finally, we find the **ConversationReference** by asking the bot service to create a new conversation between the bot and the user. This is delivered to us via a callback method which then allows us to get access to a turn context that represents the *continuation of the conversation between the bot and the user*.

Now that the Proactive Notifications are working

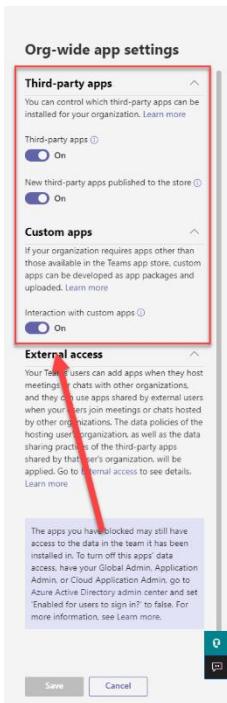
9) Teams Apps Management (via the Teams Administrator Portal)

In this step we will introduce you to the tools that Microsoft Teams Administrators have access to, to enable them to manage, install and pin third-party applications in Teams for the Tenants that they manage. This is an important concept to understand, as by working with the IT departments, and the global administrators, within a Tenant, you may be able to make accessing your application, from within Teams, easier for the users within that Tenant.

1. Navigate to <https://admin.teams.microsoft.com/> and log in as the Tenant Global Administrator. Expand **Teams apps** and select **Manage apps**. This allows administrators to manage the apps that are available throughout their organisation. Here the administrators can **manage pending app approvals, that have been submitted by non-admin users**, **Upload** new apps, in the screenshot below section 4 allows administrators to manage apps at the individual level. Finally click **org-wide app settings** to view the options.

Name	Certification	Publisher	Public status	Licenses	Custom app	Permissions
1-on-1 Hub	--	Appfluence Inc	Allowed	--	No	View details
10xGoals	Publisher attestation	xto10x Technologies	Allowed	--	No	View details
15Five	--	15Five, Inc.	Allowed	--	No	--
360 Recognition	--	Terryberry	Allowed	--	No	--
360 Tours	--	Apithings	Allowed	--	No	View details
365-QA	Publisher attestation	Advantive	Allowed	--	No	--
365Projects	Publisher attestation	365Apps	Allowed	--	No	View details
7Geese	--	7Geese	Allowed	--	No	--
8x8	Publisher attestation	8x8, Inc	Allowed	--	No	--

2. In org-wide app settings, administrators can enable or disable **Third party apps** in Teams, which would prevent the use of third-party apps if disabled. Here administrators can also enable or disable **custom apps**. Click **Cancel** once you've finished reviewing these settings.

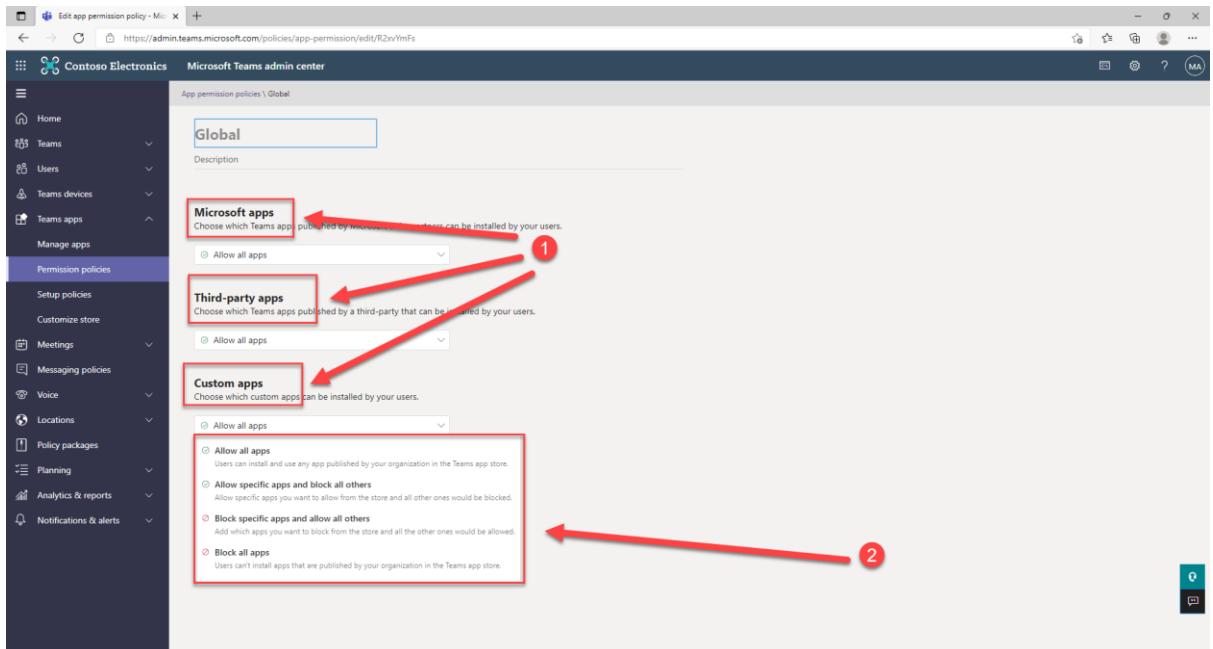


3. Now let's review the Permission Policies for Teams Apps. Select **Permission policies** and select the **Global (Org-wide default)**

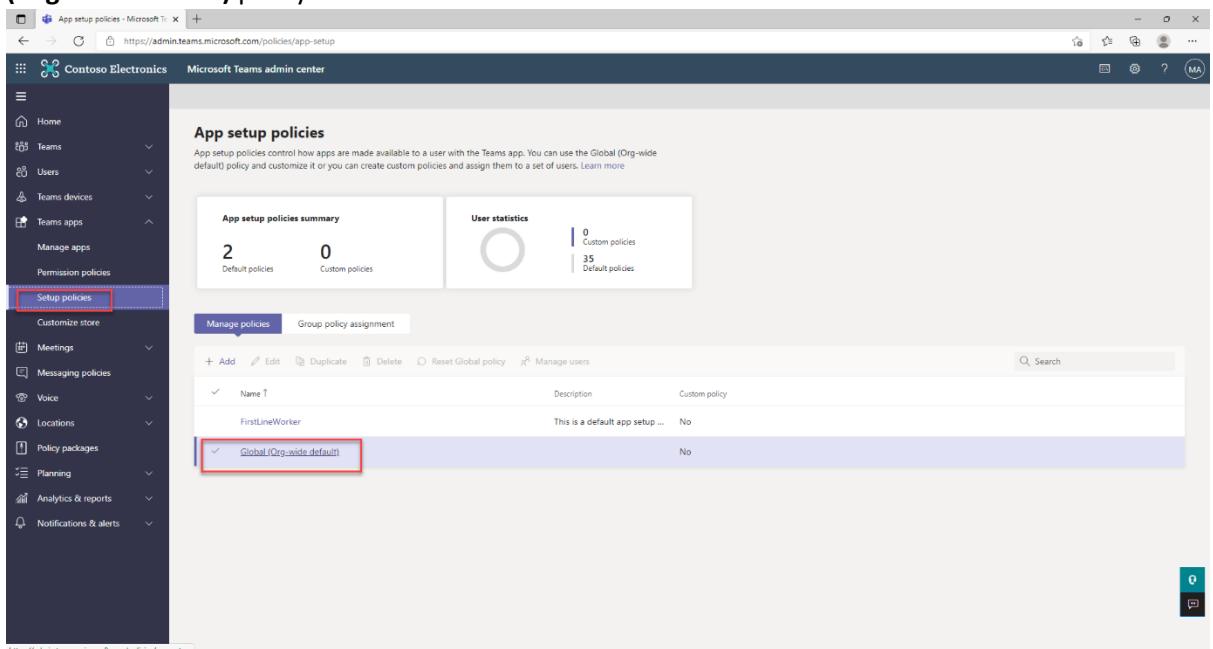
Name	Description	Custom policy
Global (Org-wide default)	No	

4. Permission policies enable administrators to be more specific on which apps are, or aren't, allowed to be used within Teams. Reviewing the options, you can see that administrators can **Allow all apps**, **Allow specific apps and block all others**, **Block specific apps and allow all others** and **Block all apps** for each type of app (**Microsoft apps (1st party)**, **Third-Party apps and custom apps**).

Administrators can create multiple policies and target them throughout their organisation accordingly, for example; an app permission policies could be targeted at Frontline Workers that enables access only to very specific Third-party apps, but blocks access to Custom apps, whilst the global policy, targeted at everybody else, enables access to all Third-party apps.



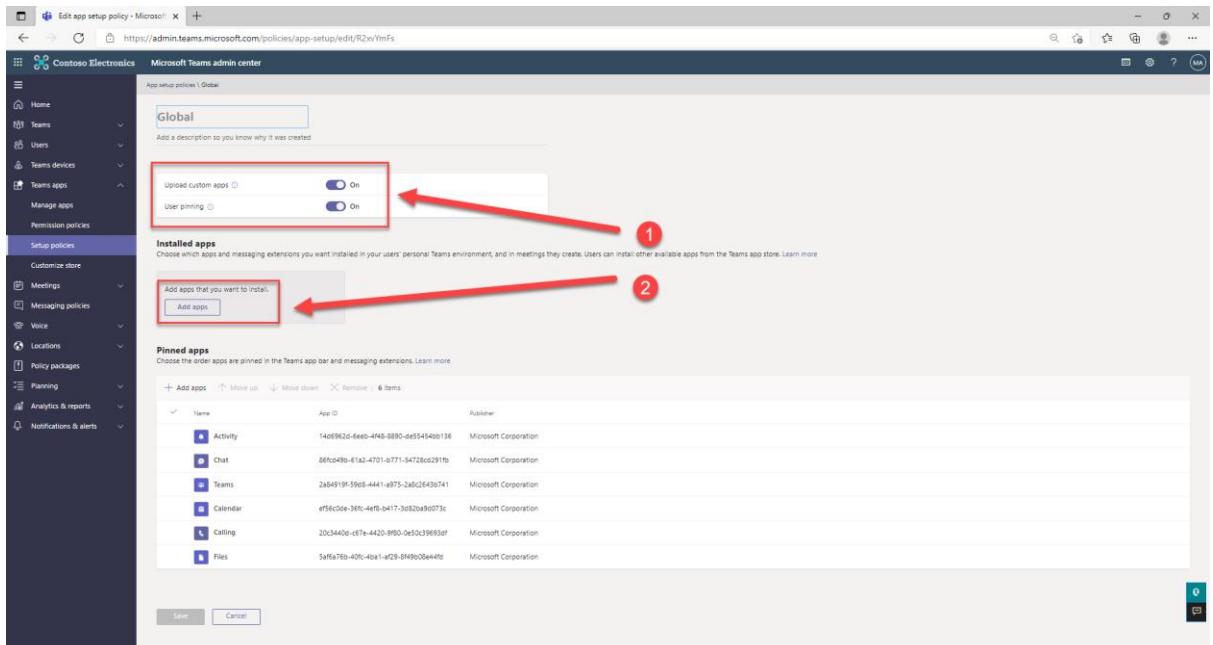
5. Now let's review the App Setup Policies, select **Setup policies** and then select the **Global (Org-wide default)** policy.



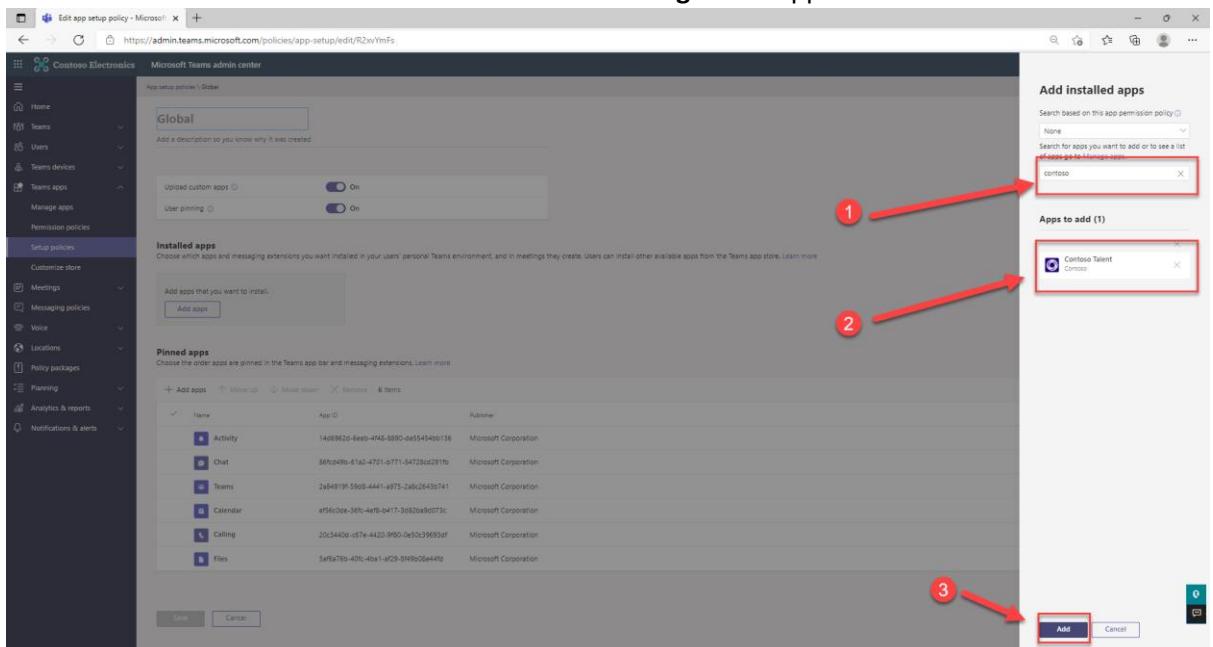
6. Setup policies are very interesting, as they allow administrators to define if the users that are in scope of these policies can **Upload custom apps and pin their own apps**, they allow administrators to **Install apps** for all users in scope of the policy, and perhaps most interestingly, **change the order of the apps that are pinned in the Teams app bar**.

NOTE: *Changes to these settings can take anything upto 24hrs to take effect, due to the globally distributed services that enable Teams to function.*

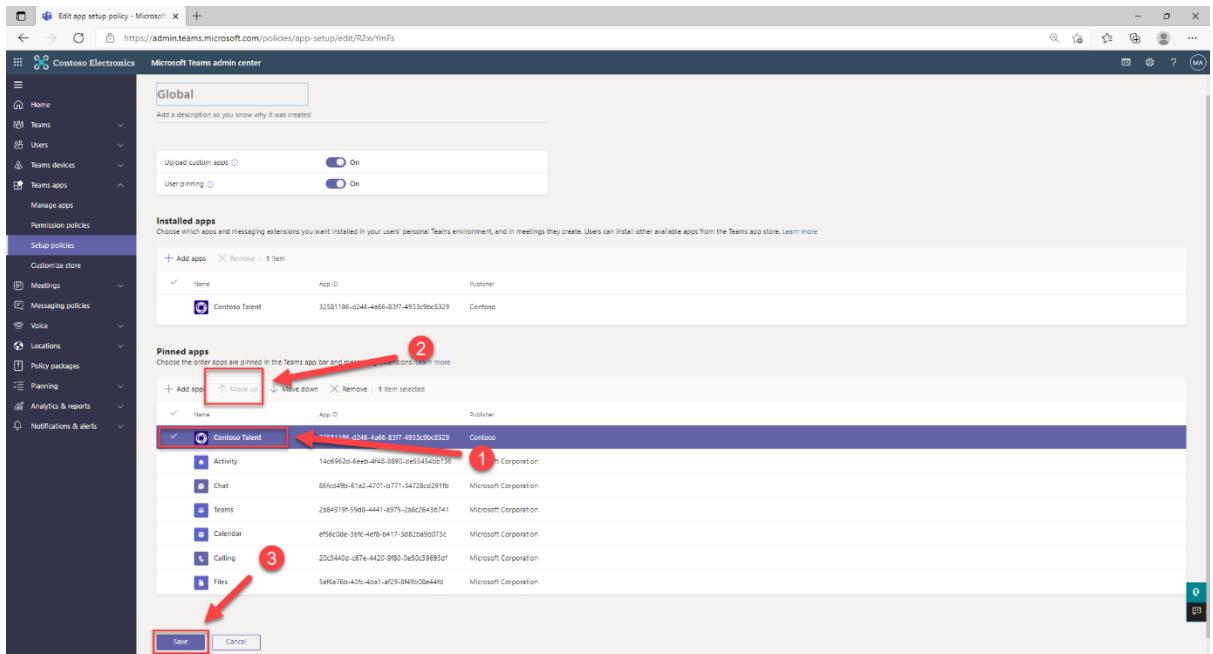
Within the Install apps section, select **Add apps**, and let's Install the Contoso Talent management app to all users in this tenant.



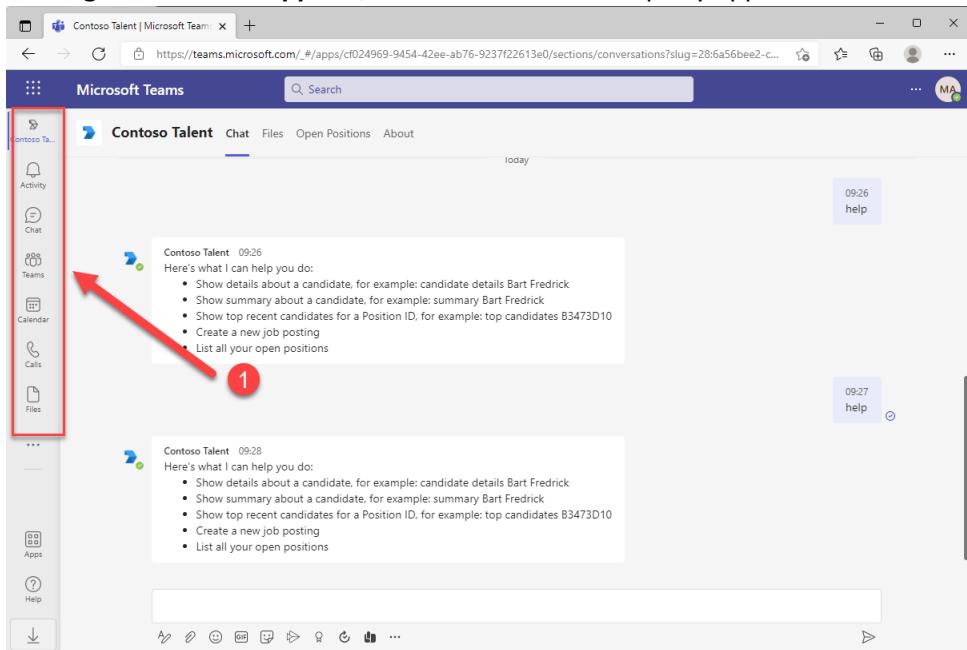
7. Search for **Contoso** and select the **Contoso Talent Management** app. Select **Add**.



8. Now that the app is scheduled to be installed for all users in this tenant, let's make it easier for them to access the Personal App in Teams. Within the **Pinned apps** section, select **Add apps**. Search for and select the **Contoso Talent Management** app, click **Add**.
9. Select the **Contoso Talent Management** app and click **Move up** until it is above the **Activity** app and is at the top of the list. That will make it easy for users to find the app, and will help drive usage of this app within Teams, therefore making the time spent developing this app worth the effort! Click **Save**!



10. As mentioned previously, changes to the App policies, in the Teams Admin portal, can take an extended amount of time to take effect. Once they have been applied though (try in about 4 hours and it may have been applied), if you log into Teams as a user that was in scope of the App Setup policy, you should see that the **Contoso Talent App is installed and Pinning in the Teams app bar**, above the default first party apps!



Great job! Hopefully this short section of the lab has helped to demystify how administrators can build policies to manage, install, deploy and secure Teams apps throughout their Microsoft 365 Tenants.

- 10) Universal Actions Adaptive Cards (delivered to both Teams and Outlook simultaneously) – Coming soon!

This section is coming soon! We are working on the code, and documentation steps, to enable this app to send proactive notifications with Adaptive Cards into both Teams and Outlook simultaneously and will push an update into the GitHub repo when it's ready.

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.