Lab 2: Create embedded web experiences with tabs for Microsoft Teams

Contents

[Exercise 1 - Create a custom Microsoft Teams personal tab 3](#_Toc32999831)

[Create Microsoft Teams app 3](#_Toc32999832)

[Test the personal tab 4](#_Toc32999833)

[Update the project to use the Stardust UI library 8](#_Toc32999834)

[Implement the personal tab's user interface 10](#_Toc32999835)

[Use App Studio to update the Microsoft Teams app manifest 13](#_Toc32999836)

[Install and test the Microsoft Teams app 16](#_Toc32999837)

[Summary 17](#_Toc32999838)

[Exercise 2 - Create a custom Microsoft Teams channel or group tab 17](#_Toc32999839)

[Add a channel app to the Microsoft Teams app project 17](#_Toc32999840)

[Test the channel tab 18](#_Toc32999841)

[Update the configuration tab 21](#_Toc32999842)

[Update the configuration tab to use the Stardust UI library 21](#_Toc32999843)

[Implement the configuration page logic 23](#_Toc32999844)

[Implement the configuration page user interface 24](#_Toc32999845)

[Test the configuration page 24](#_Toc32999846)

[Implement the channel tab 25](#_Toc32999847)

[Update the channel tab to use the Stardust UI library 26](#_Toc32999848)

[Implement the channel page logic 28](#_Toc32999849)

[Implement the channel page user interface 29](#_Toc32999850)

[Test the channel tab page 30](#_Toc32999851)

[Summary 31](#_Toc32999852)

[Exercise 3 - Implement authentication in a custom tab 31](#_Toc32999853)

[Create an Azure AD application 31](#_Toc32999854)

[Grant Azure AD application permissions to Microsoft Graph 34](#_Toc32999855)

[Create Microsoft Teams app 36](#_Toc32999856)

[Update the project to use the Stardust UI library 37](#_Toc32999857)

[Implement the tab's logic and user interface 40](#_Toc32999858)

[Implement the authentication and Microsoft Graph request logic 42](#_Toc32999859)

[Install and test the Microsoft Teams app 47](#_Toc32999860)

[Summary 48](#_Toc32999861)

# Exercise 1 - Create a custom Microsoft Teams personal tab

In this exercise, you'll create a new Microsoft Teams personal tab by using the Microsoft Teams Yeoman generator, Visual Studio Code, and App Studio.

## Create Microsoft Teams app

Open your command prompt, and go to a directory where you want to save your work. Create a new folder named learn-msteams-tabs, and change the directory into that folder.

Run the Yeoman generator for Microsoft Teams by running the following command:

yo teams



Yeoman starts and asks you a series of questions. Answer the questions with the following values:

* **What is your solution name?**: Learn MSTeams Tabs
* **Where do you want to place the files?**: Use the current folder
* **Title of your Microsoft Teams App project**: Learn MSTeams Tabs
* **Your (company) name (max 32 characters)**: Contoso
* **Which manifest version would you like to use?**: 1.5
* **Enter your Microsoft Partner Id, if you have one**: (Leave blank to skip)
* **What features do you want to add to your project?**: A tab
* **The URL where you will host this solution**: [https://learnmsteamstabs.azurewebsites.net](https://learnmsteamstabs.azurewebsites.net/)
* **Would you like to include Test framework and initial tests?**: No
* **Would you like to use Azure Applications Insights for telemetry?**: No
* **Default Tab name (max 16 characters)**: LearnPersonalTab
* **Do you want to create a configurable or static tab?**: Static

**Note**

Most of the answers to these questions can be changed after you create the project. For example, the URL where the project will be hosted isn't important at the time of creating or testing the project.

After you answer the generator's questions, the generator creates the scaffolding for the project. The generator then runs npm install that downloads all the dependencies required by the project.

## Test the personal tab

Before you customize the tab, let's test the tab to see the initial developer experience for testing.

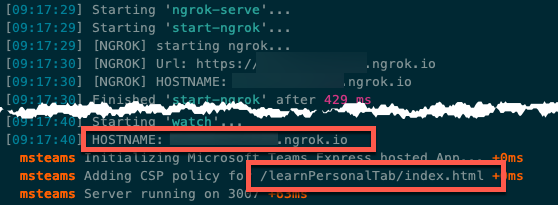
From the command line, go to the root folder for the project and run the following command:

gulp ngrok-serve

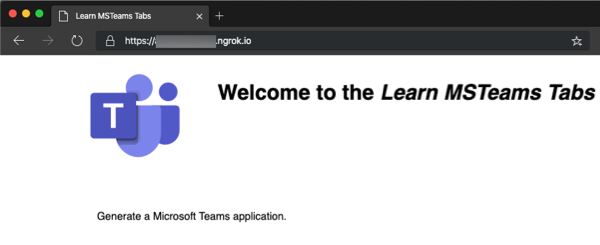
This gulp task runs many other tasks all displayed within the command-line console. The ngrok-serve task builds your project and starts a local web server (http://localhost:3007). It then starts ngrok with a random subdomain that creates a secure URL to your local web server.

**Note**

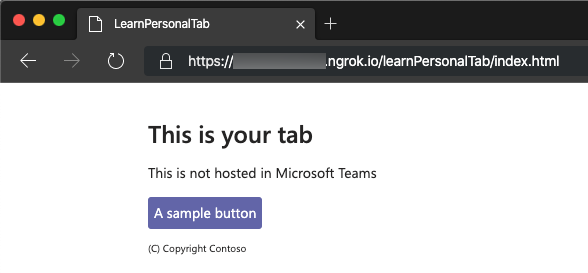
Microsoft Teams requires all content displayed within a tab to be loaded from an HTTPS request. In development, this can be done by using the tool **[ngrok](https://www.ngrok.com/)** that creates a secure rotatable URL to your local HTTP web server. Ngrok is included as a dependency within the project, so there's nothing to set up or configure.



Open a browser, and go to the ngrok URL displayed in the console.



Update the URL in the browser to load the tab created by the scaffolding process. Here you can see the page can determine that it isn't running within the Microsoft Teams client.

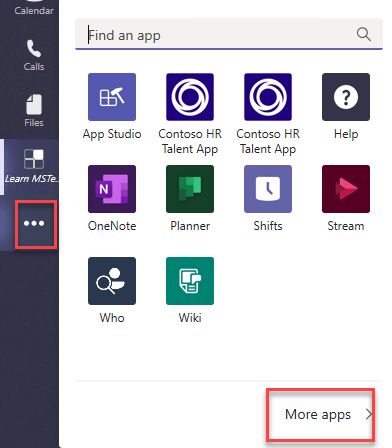


Now let's load the tab in Microsoft Teams. In the browser, go to [Microsoft Teams](https://teams.microsoft.com/). Sign in with the credentials of a Work and School account.

**Note**

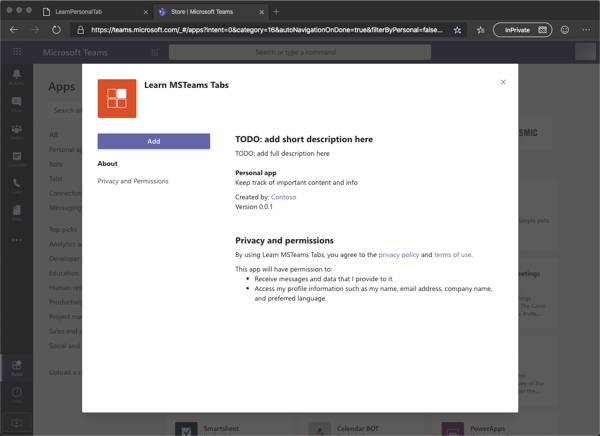
Microsoft Teams is available for use as a web client, a desktop client, and a mobile client. In this module, we use the web client, but any of the clients can be used.

In the app bar on the left, select the **More added apps** button. Then select **More Apps** > **Upload a custom app >** **Upload for me or my teams**.

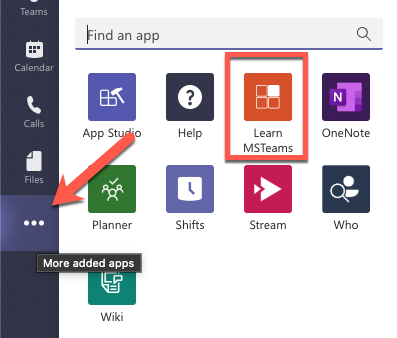


In the file dialog box that appears, select the Microsoft Teams package in your project. This app package is a zip file in the project's ./package folder.

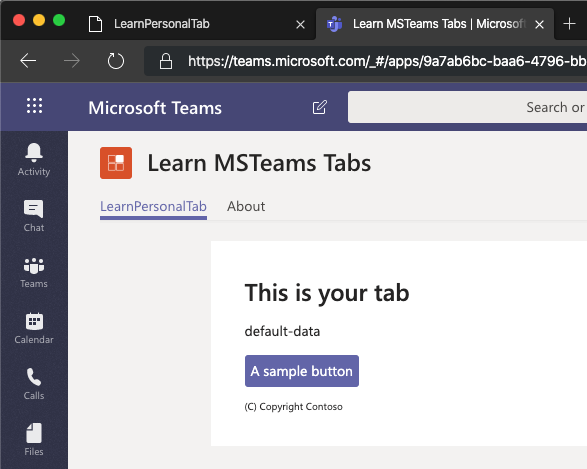
After the package is uploaded, Microsoft Teams displays a summary of the app. Here you can see some todo items to address. You'll update the todo items later in the exercise.



Select the **Add** button to install the app, which adds a new personal tab to your **More added apps** dialog box.



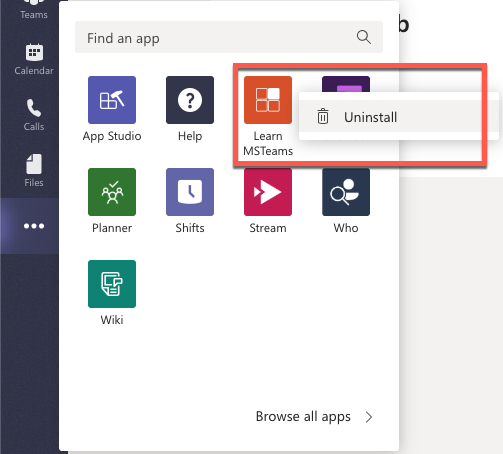
Select the app to go to the new tab.

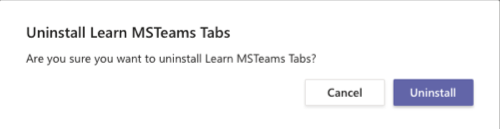


Notice that when the content page is loaded in a tab within the Microsoft teams client, it displays the entityId property of the tab, not the message "This isn't hosted in Microsoft Teams" as you saw when you viewed the content page in the browser. The tab can detect if it's loaded within the Microsoft Teams client by using the Microsoft Teams JavaScript SDK.

The next step is to make some changes to the project.

Uninstall the app by right-clicking the app in the **More added apps** dialog box and select **Uninstall**. Then select **Uninstall** in the confirmation dialog box that appears.





Next, stop the local web server by selecting Ctrl+C in the console to stop the running process.

## Update the project to use the Stardust UI library

Microsoft Teams recommends that your custom apps use React and the themable React component library [Stardust UI React](https://stardust-ui.github.io/react/). To use Stardust in the Microsoft Teams app, we need to make some changes to the project.

**Important**

At the time of publication of this module, there are plans to update the Yeoman generator for Microsoft Teams to include Stardust in the default project. At the time of publication of this module, the default project uses the older Microsoft Teams control library that Stardust is replacing.

The steps in this section might not be necessary because the Yeoman generator for Microsoft Teams might have been updated. Review each of the instructions in this section and compare the results with your project to determine if they're necessary.

The first step is to uninstall the existing control library and install the Stardust library. Run the following two commands in the command line from the root folder of the project:

npm uninstall msteams-ui-components-react

npm install @stardust-ui/react

Locate and open the file that contains the React component used in the project: ./src/app/scripts/learnPersonalTab/LearnPersonalTab.tsx.

Update the import statements in this file to replace the component library used. Find the following import statement that imports the legacy Microsoft Teams UI Components - React library:

import {

PrimaryButton,

TeamsThemeContext,

Panel,

PanelBody,

PanelHeader,

PanelFooter,

Surface,

getContext

} from "msteams-ui-components-react";

Replace the previous statement with the following import statement:

import {

Flex, Provider, themes, ThemePrepared,

Alert, Header,

Button, Icon, Input, Label, List, Text

} from "@stardust-ui/react";

The default project contains additional user interface style code that used the previous control library. This code is no longer necessary.

Locate the following code in the componentWillMount() method in the LearnPersonalTab class and delete it.

this.setState({

fontSize: this.pageFontSize()

});

Locate the following code in the render() method in the LearnPersonalTab class and delete it.

const context = getContext({

baseFontSize: this.state.fontSize,

style: this.state.theme

});

const { rem, font } = context;

const { sizes, weights } = font;

const styles = {

header: { ...sizes.title, ...weights.semibold },

section: { ...sizes.base, marginTop: rem(1.4), marginBottom: rem(1.4) },

footer: { ...sizes.xsmall }

};

Locate the return () statement in the render() method in the LearnPersonalTab class and delete the contents. This code used the UI library that you replaced with Stardust. At this point, the render() method should look like the following code:

public render() {

return (

);

}

## Implement the personal tab's user interface

Now you can implement the user interface for the tab. The simple tab has a basic interface. It presents a list of items, and users can add items to the list.

First, update the state of the component to contain a list of items and a property for a new item. Locate the ILearnPersonalTabState interface in the LearnPersonalTab.tsx file, and add the following properties to it:

teamsTheme: ThemePrepared;

todoItems: string[];

newTodoValue: string;

Add the following method to the LearnPersonalTab class that updates the component state to the Stardust theme that matches the currently selected Microsoft Teams client theme:

private updateStardustTheme = (teamsTheme: string = "default"): void => {

let stardustTheme: ThemePrepared;

switch (teamsTheme) {

case "default":

stardustTheme = themes.teams;

break;

case "dark":

stardustTheme = themes.teamsDark;

break;

case "contrast":

stardustTheme = themes.teamsHighContrast;

break;

default:

stardustTheme = themes.teams;

break;

}

// update the state

this.setState(Object.assign({}, this.state, {

teamsTheme: stardustTheme

}));

}

Initialize the current theme and state of the component. Locate the line this.updateTheme(this.getQueryVariable("theme")); and replace it with the following code in the componentWillMount() method:

this.updateStardustTheme(this.getQueryVariable("theme"));

this.setState(Object.assign({}, this.state, {

todoItems: ["Submit time sheet", "Submit expense report"],

newTodoValue: ""

}));

Within the componentWillMount() method, locate the following line:

microsoftTeams.registerOnThemeChangeHandler(this.updateTheme);

This code registers an event handler to update the component's theme to match the theme of the current Microsoft Teams client when this page is loaded as a tab. Update this line to call the new handler in the following line to register another handler to update the Stardust library theme:

microsoftTeams.registerOnThemeChangeHandler(this.updateStardustTheme);

With the theme management and state initialized, we can now implement the user interface.

Locate the render() method, and update the return statement to the following code. The render() method now displays the list of items in our state output with a brief copyright statement.

public render() {

return (

<Provider theme={ this.state.teamsTheme }>

<Flex column gap="gap.smaller">

<Header>This is your tab</Header>

<Alert icon="exclamation-triangle" content={ this.state.entityId } dismissible></Alert>

<Text content="These are your to-do items:" size="medium"></Text>

<List selectable>

{ this.state.todoItems.map(todoItem => (

<List.Item media={<Icon name="window-maximize outline"></Icon> }

content={ todoItem }>

</List.Item> ))

}

</List>

TODO: add new list item form here

<Text content="(C) Copyright Contoso" size="smallest"></Text>

</Flex>

</Provider>

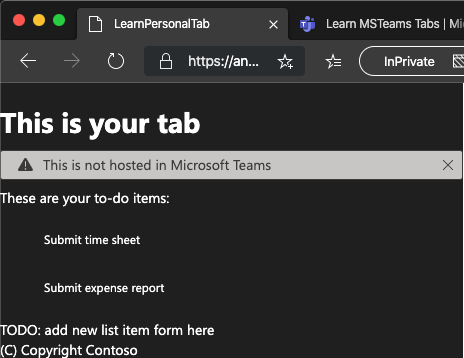
);

}

**Tip**

At this point, you can test your tab without loading it in Microsoft Teams. Run the command **gulp ngrok-serve** from the command line, and go to https://{your-ngrok-subdomain}.ngrok.io/learnPersonalTab/index.html in the browser.

Add the query string value ?theme=dark to the URL to see the theme change:



The next step is to add some interactivity to the tab. Add the following methods to the LearnPersonalTab class. These methods handle updating the state when specific events happen on the form that you'll add to the component.

private handleOnChanged = (event): void => {

this.setState(Object.assign({}, this.state, { newTodoValue: event.target.value }));

}

private handleOnClick = (event: React.MouseEvent<HTMLButtonElement>): void => {

const newTodoItems = this.state.todoItems;

newTodoItems.push(this.state.newTodoValue);

this.setState(Object.assign({}, this.state, {

todoItems: newTodoItems,

newTodoValue: ""

}));

}

Finally, locate the string TODO: add new list item form here in the render() method, and replace it with the following code. This code displays a form that the user can use to add items to the list.

<Flex gap="gap.medium">

<Flex.Item grow>

<Flex>

<Label icon="to-do-list"

styles={{

background: "darkgray",

height: "auto",

padding: "0 15px"

}}></Label>

<Flex.Item grow>

<Input placeholder="New todo item" fluid

value={this.state.newTodoValue}

onChange={this.handleOnChanged}></Input>

</Flex.Item>

</Flex>

</Flex.Item>

<Button content="Add Todo" primary

onClick={this.handleOnClick}></Button>

</Flex>

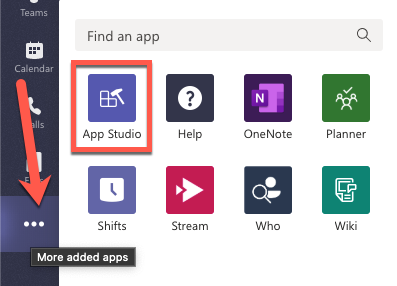
## Use App Studio to update the Microsoft Teams app manifest

At this point, the app is complete. Recall from our initial test that when the app was added to Microsoft Teams, it had a few todo strings for the description of the app. While you could change these values in the project's ./src/manifest/manifest.json file, you use App Studio to make these changes.

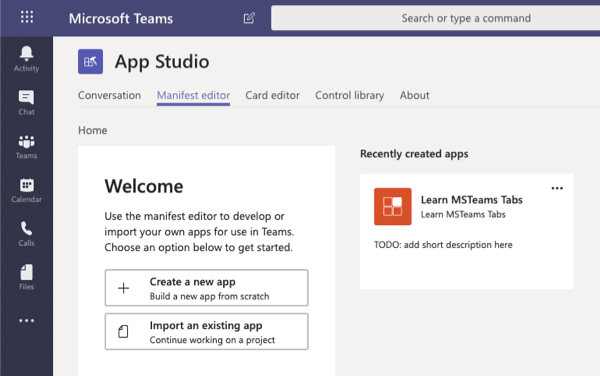
First, build and run the project by running the command **gulp ngrok-serve** in the command line like you did previously. This step also creates the Microsoft Teams app package.

In the browser, go to [Microsoft Teams](https://teams.microsoft.com/) and sign in with the credentials of a Work and School account.

Using the **More added apps** link in the app bar on the left, select **App Studio**.



Select the **Manifest editor** tab in App Studio, and then select **Import an existing app**. Locate the zip file that can be found in the project's ../package folder and open it.

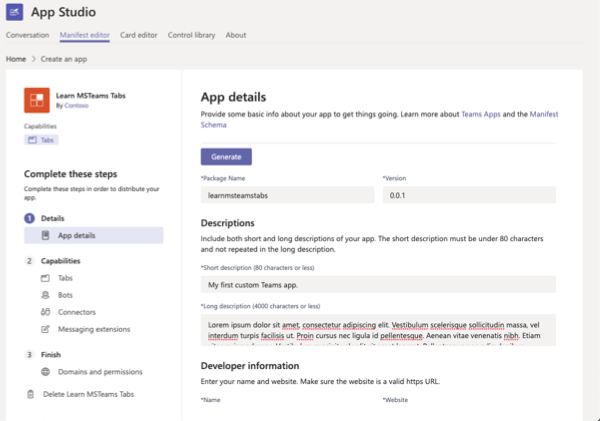


Edit the app by selecting its tile, or use the menu in the upper-right corner for more options and select **Edit**.

On the **App details** page, change the **Full name** of the app to **Learn Microsoft Teams Tabs**.

On the **App details** page, scroll down to the **Descriptions** section and enter the following values:

* **Short description**: My first custom Teams app
* **Full description**: *enter a long description*



Update the name of the tab by selecting **Capabilities** > **Tabs** in the left pane in App Studio.

Locate the only personal tab in the project. Select the menu for more options on the tab, and select **Edit**. Change the name of the tab to **My First Tab**. Add ?theme={theme} to the end of the **Content URL** property. Select **Save** to save your changes.

The changes made to the app within App Studio aren't saved to your project. If you want to update the project, download the app package from App Studio.

To download the project, select **Finish** > **Test and distribute** in the left pane in App Studio. Then select **Download**.

**Caution**

Be careful if you chose to update the manifest.json file in your project with the one in the package downloaded from App Studio.

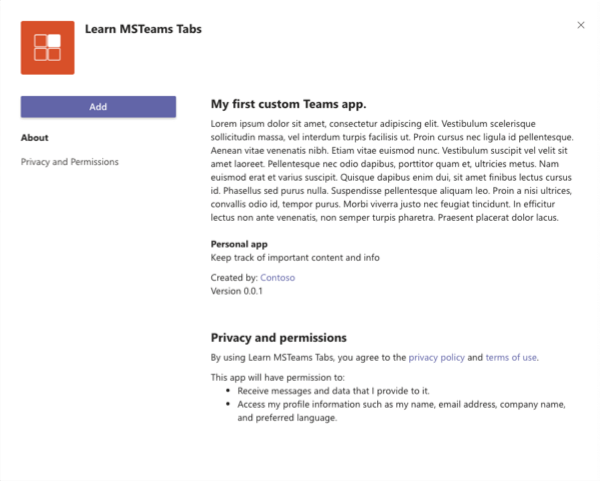
The manifest file in your project contains placeholder strings that are updated by the build and debugging process that's replaced when you test the project. using placeholder strings simplifies the development and debugging process.

For instance, the placeholder {{HOSTNAME}} is replaced with the hosting URL of the app each time the package is re-created.

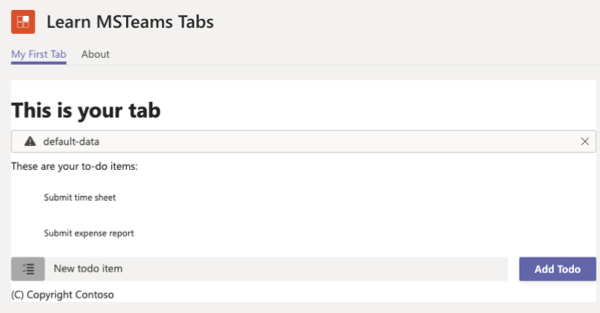
So you might not want to completely replace the existing manifest.json file with the file generated by App Studio.

## Install and test the Microsoft Teams app

In App Studio, select **Finish** > **Test and distribute** in the left pane in App Studio. Then select **Install**. Notice that the new names and descriptions are shown prior to installing the app.



Select **Add** to install the app. This action adds a new personal tab to your **More added apps** dialog box. Select the app to see the updated working version.



## Summary

In this exercise, you created a new Microsoft Teams personal tab by using the Microsoft Teams Yeoman generator, Visual Studio Code, and App Studio.

# Exercise 2 - Create a custom Microsoft Teams channel or group tab

In this exercise, you'll learn how to create a channel tab with a configuration page in a Microsoft Teams app.

**Important**

This exercise assumes that you created the Microsoft Teams app project with the Yeoman generator that contains a personal tab from the previous unit in this module. You'll update the project and add a channel tab in this exercise.

## Add a channel app to the Microsoft Teams app project

The Yeoman generator for Microsoft Teams can be used to add new components to an existing project. In this section, you'll add a channel tab to the existing project.

Execute the following command in the console from the root folder of the project:

yo teams

Yeoman starts and asks you a series of questions. Answer the questions with the following values:

* **You are running the generator on an already existing project... are you sure you want to continue?**: Yes
* **Do you want to change the current manifest version (1.5)?**: No
* **What features do you want to add to your project?**: learnMsTeamsTabs
* **Default tab name (max 16 characters)**: ConfigMathTab
* **Do you want to create a configurable or static tab?**: Configurable
* **What scopes do you intend to use for your tab?**: In a Team
* **Do you want this tab to be available in SharePoint Online?**: Yes
* **How do you want your tab to be available in SharePoint?**: As a full-page application, as a web part

After you answer the generator's questions, the generator adds the additional files for a new component. Then it runs npm install to ensure that any new dependencies are downloaded for the project.

## Test the channel tab

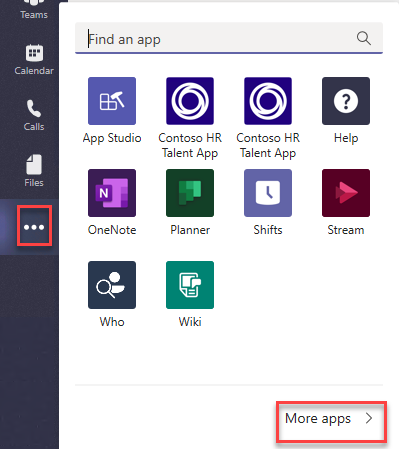
Before you customize the tab, let's test the tab to see the experience for testing.

From the command line, go to the root folder for the project and run the following command:

gulp ngrok-serve

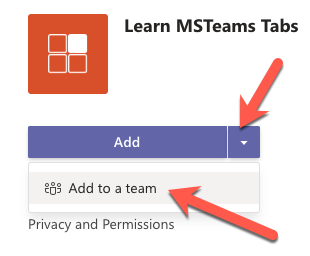
Open a browser, and go to [Microsoft Teams](https://teams.microsoft.com/). Sign in with the credentials of a Work and School account.

In the app bar on the left, select the **More added apps** button. Then select **More apps** > **Upload a custom app >Upload for me or my teams**.

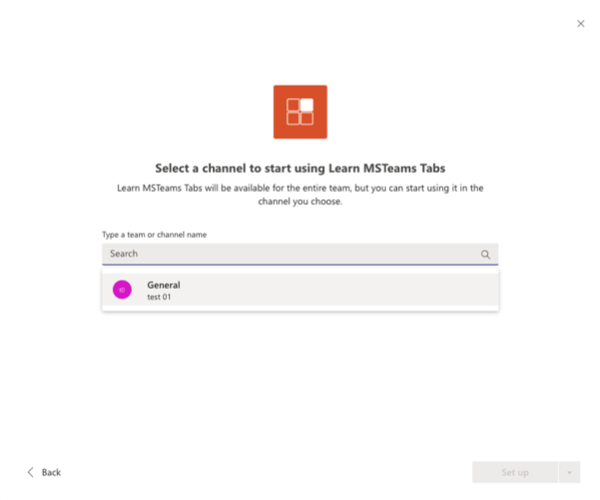


In the file dialog box that appears, select the Microsoft Teams package in your project. This app package is a zip file in the project's ./package folder.

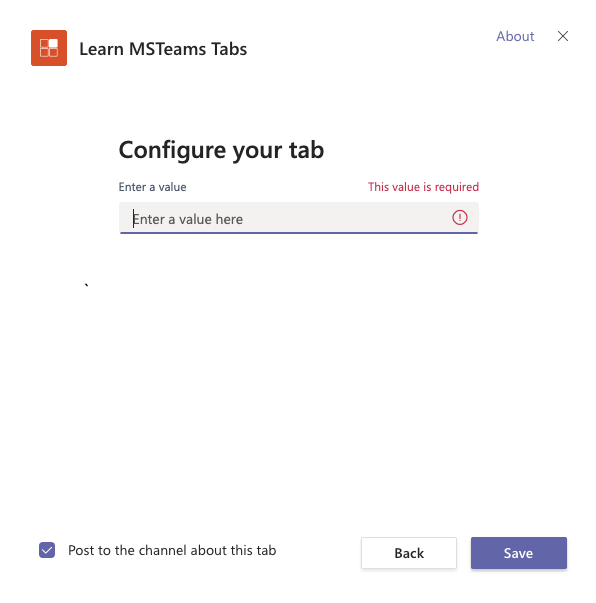
After the package is uploaded, Microsoft Teams displays a summary of the app. Select the arrow next to the **Add** button, and select **Add to a team** to install the app.



In the **Select a channel to start using** dialog box, select an existing team. Then select **Set up a tab**.



Before the tab is added to the team, Microsoft Teams displays the tab's configuration page.



Enter anything in the text box, and select **Save**.

Microsoft Teams adds the tab to the channel and displays it for you. You should see the text you entered on the configuration page displayed in the tab.

## Update the configuration tab

On the tab you create in this exercise, the user can select a math operation to do on the configuration page. This value is saved with the tab so that users of the tab can do this operation on two values to see the results.

The first step is to modify the configuration page.

Locate and open the file ./src/app/scripts/configMathTab/ConfigMathTabConfig.tsx.

## Update the configuration tab to use the Stardust UI library

Before you create the configuration tab, first update it to use the Stardust UI library.

Update the import statements in this file to replace the component library used. Find the following import statement that imports the legacy Microsoft Teams UI Components - React library:

import {

PrimaryButton,

TeamsThemeContext,

Panel,

PanelBody,

PanelHeader,

PanelFooter,

Surface,

getContext

} from "msteams-ui-components-react";

Replace the previous statement with the following import statement:

import {

Flex, Provider, themes, ThemePrepared,

Header,

Dropdown, DropdownProps, Text

} from "@stardust-ui/react";

Locate the IConfigMathTabConfigState interface, and replace its contents with the following two members:

teamsTheme: ThemePrepared;

mathOperator: string;

Locate the following code in the render() method in the ConfigMathTabConfig class and delete it:

const context = getContext({

baseFontSize: this.state.fontSize,

style: this.state.theme

});

const { rem, font } = context;

const { sizes, weights } = font;

const styles = {

header: { ...sizes.title, ...weights.semibold },

section: { ...sizes.base, marginTop: rem(1.4), marginBottom: rem(1.4) },

footer: { ...sizes.xsmall }

};

Locate the return () statement in the render() method in the ConfigMathTabConfig class, and delete the contents. This code used the UI library that you replaced with Stardust. At this point, the render() method should look like the following code:

public render() {

return (

);

}

Add the following method to the ConfigMathTabConfig class that will update the component state to the Stardust theme that matches the currently selected Microsoft Teams client theme:

private updateStardustTheme = (teamsTheme: string = "default"): void => {

let stardustTheme: ThemePrepared;

switch (teamsTheme) {

case "default":

stardustTheme = themes.teams;

break;

case "dark":

stardustTheme = themes.teamsDark;

break;

case "contrast":

stardustTheme = themes.teamsHighContrast;

break;

default:

stardustTheme = themes.teams;

break;

}

// update the state

this.setState(Object.assign({}, this.state, {

teamsTheme: stardustTheme

}));

}

Initialize the current theme and state of the component. Locate the line this.updateTheme(this.getQueryVariable("theme")); and replace it with the following code in the componentWillMount() method:

this.updateStardustTheme(this.getQueryVariable("theme"));

Locate the following code in the componentWillMount() method and delete it:

this.setState({

fontSize: this.pageFontSize()

});

## Implement the configuration page logic

The configuration page displays a drop-down list of four math operators to select from. After an operator is selected, it's saved to the tab's entityId property with the string **MathPage** appended to it. This value is used by the tab page to determine what operation to perform in the tab.

Locate the following line in the componentWillMount() method: microsoftTeams.getContext(). The function passed into this method sets the state of the React component. Replace the this.setState() method with the following code. This new code takes the value of the entityId property on the tab, removes the **MathPage** string, and leaves only the operator.

this.setState(Object.assign({}, this.state, {

mathOperator: context.entityId.replace("MathPage", "")

}));

Next, locate the following line in the componentWillMount() method: microsoftTeams.settings.registerOnSaveHandler(). This method lets you provide the function to execute when the user selects the **Save** button on the configuration page. This code should save any settings you need to save and notify Microsoft Teams that the configuration page saved the settings successfully.

Update this code to save the selected math operation and change the name of the tab.

microsoftTeams.settings.registerOnSaveHandler((saveEvent: microsoftTeams.settings.SaveEvent) => {

// Calculate host dynamically to enable local debugging

const host = "https://" + window.location.host;

microsoftTeams.settings.setSettings({

contentUrl: host + "/configMathTab/?data=",

suggestedDisplayName: "Config Math Tab",

removeUrl: host + "/configMathTab/remove.html",

entityId: `${this.state.mathOperator}MathPage`

});

saveEvent.notifySuccess();

});

Add the following event handler to the ConfigMathTabConfig class, which updates the component state to be the value of the selected operator:

private handleOnSelectedChange = (event, props: DropdownProps): void => {

this.setState(Object.assign({}, this.state, {

mathOperator: (props.value) ? props.value.toString() : "add"

}));

}

## Implement the configuration page user interface

Locate the render() method. Replace it with the following code, which adds a drop-down list for the user to select the operator they want to use:

public render() {

return (

<Provider theme={this.state.teamsTheme}>

<Flex gap="gap.smaller" style={{ height: "300px" }}>

<Dropdown placeholder="Select the math operator"

items={[

"add",

"subtract",

"multiply",

"divide"

]}

onSelectedChange={this.handleOnSelectedChange}></Dropdown>

</Flex>

</Provider>

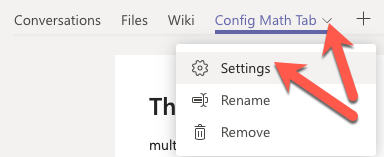
);

}

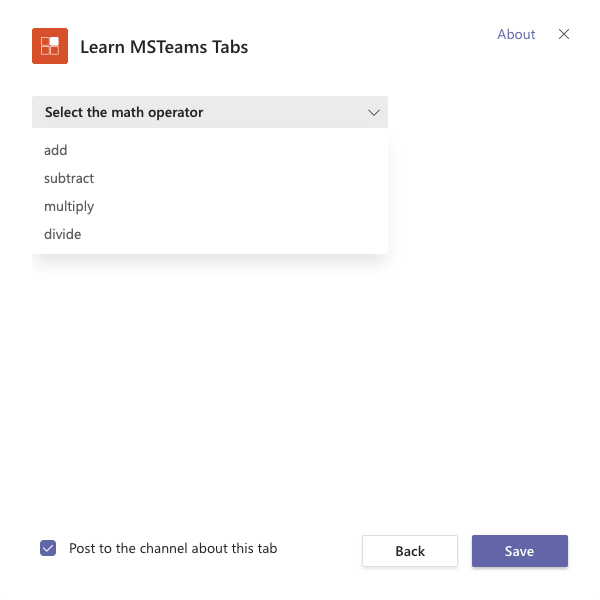
## Test the configuration page

At this point, the configuration page is complete. If you removed the tab in the last step, repeat the process to upload and add it again.

If you didn't remove the tab, select the menu from the tab and then select **Settings**.



The configuration page opens with the updated component.



Select one of the math operators, and save your changes by selecting **Save**. The tab should display the selected operator with the **MathPage** suffix.

## Implement the channel tab

The last step is to implement the channel tab.

Locate and open the file ./src/app/scripts/configMathTab/ConfigMathTab.tsx.

## Update the channel tab to use the Stardust UI library

Before you create the configuration tab, first update it to use the Stardust UI library.

Update the import statements in this file to replace the component library used. Find the following import statement that imports the legacy Microsoft Teams UI Components - React library:

import {

PrimaryButton,

TeamsThemeContext,

Panel,

PanelBody,

PanelHeader,

PanelFooter,

Surface,

getContext

} from "msteams-ui-components-react";

Replace the previous statement with the following import statement:

import {

Flex, Provider, themes, ThemePrepared,

Header,

Button, Input, Text

} from "@stardust-ui/react";

Locate the IConfigMathTabState interface, and replace its contents with the following two members:

teamsTheme: ThemePrepared;

mathOperator?: string;

operandA: number;

operandB: number;

result: string;

Locate the following code in the render() method in the ConfigMathTab class and delete it:

const context = getContext({

baseFontSize: this.state.fontSize,

style: this.state.theme

});

const { rem, font } = context;

const { sizes, weights } = font;

const styles = {

header: { ...sizes.title, ...weights.semibold },

section: { ...sizes.base, marginTop: rem(1.4), marginBottom: rem(1.4) },

footer: { ...sizes.xsmall }

};

Locate the return () statement in the render() method in the ConfigMathTab class, and delete the contents. This code used the UI library that you replaced with Stardust. At this point, the render() method should look like the following code:

public render() {

return (

);

}

Add the following method to the ConfigMathTab class that will update the component state to the Stardust theme that matches the currently selected Microsoft Teams client theme:

private updateStardustTheme = (teamsTheme: string = "default"): void => {

let stardustTheme: ThemePrepared;

switch (teamsTheme) {

case "default":

stardustTheme = themes.teams;

break;

case "dark":

stardustTheme = themes.teamsDark;

break;

case "contrast":

stardustTheme = themes.teamsHighContrast;

break;

default:

stardustTheme = themes.teams;

break;

}

// update the state

this.setState(Object.assign({}, this.state, {

teamsTheme: stardustTheme

}));

}

Initialize the current theme and state of the component. Locate the line this.updateTheme(this.getQueryVariable("theme")); and replace it with the following code in the componentWillMount() method:

this.updateStardustTheme(this.getQueryVariable("theme"));

Within the componentWillMount() method, locate the following line:

microsoftTeams.registerOnThemeChangeHandler(this.updateTheme);

This code registers an event handler to update the component's theme to match the theme of the current Microsoft Teams client when this page is loaded as a tab. Update this line to call the new handler in the following line to register another handler to update the Stardust library theme.

microsoftTeams.registerOnThemeChangeHandler(this.updateStardustTheme);

Locate the following code in the componentWillMount() method and delete it:

this.setState({

fontSize: this.pageFontSize()

});

## Implement the channel page logic

Locate the following line in the componentWillMount() method: microsoftTeams.getContext(). The function passed into this method sets the state of the React component. Replace the this.setState() method with the following code. This new code takes the value of the entityId property on the tab, removes the **MathPage** string, and leaves only the operator.

this.setState(Object.assign({}, this.state, {

mathOperator: context.entityId.replace("MathPage", "")

}));

Locate the following code in the componetWillMount() method:

this.setState({

entityId: "This is not hosted in Microsoft Teams"

});

Replace this code with the following code. This new code will cause the math operator to add two numbers by default in case this page is loaded outside of a Microsoft Teams client.

this.setState(Object.assign({}, this.state, {

mathOperator: "add"

}));

Add the following event handlers to the ConfigMathTab class. These event handlers will update the state with the values from the controls and perform the calculation of the two numbers by using the operator specified on the configuration page.

private handleOnChangedOperandA = (event): void => {

this.setState(Object.assign({}, this.state, { operandA: event.target.value }));

}

private handleOnChangedOperandB = (event): void => {

this.setState(Object.assign({}, this.state, { operandB: event.target.value }));

}

private handleOperandChange = (): void => {

let stringResult: string = "n/a";

if (!isNaN(Number(this.state.operandA)) && !isNaN(Number(this.state.operandB))) {

switch (this.state.mathOperator) {

case "add":

stringResult = (Number(this.state.operandA) + Number(this.state.operandB)).toString();

break;

case "subtract":

stringResult = (Number(this.state.operandA) - Number(this.state.operandB)).toString();

break;

case "multiply":

stringResult = (Number(this.state.operandA) \* Number(this.state.operandB)).toString();

break;

case "divide":

stringResult = (Number(this.state.operandA) / Number(this.state.operandB)).toString();

break;

default:

stringResult = "n/a";

break;

}

}

this.setState(Object.assign({}, this.state, {

result: stringResult

}));

}

## Implement the channel page user interface

Locate the render() method in the ConfigMathTab class. Replace the existing method implementation with the following code. This new code adds two input boxes and a button to the page. When the button is selected, it performs the math operation selected on the configuration page to the two values and displays the results.

public render() {

return (

<Provider theme={this.state.teamsTheme}>

<Flex column gap="gap.smaller">

<Header>This is your tab</Header>

<Text content="Enter the values to calculate" size="medium"></Text>

<Flex gap="gap.smaller">

<Flex.Item>

<Flex gap="gap.smaller">

<Flex.Item>

<Input autoFocus

value={this.state.operandA}

onChange={this.handleOnChangedOperandA}></Input>

</Flex.Item>

<Text content={this.state.mathOperator}></Text>

<Flex.Item>

<Input value={this.state.operandB}

onChange={this.handleOnChangedOperandB}></Input>

</Flex.Item>

</Flex>

</Flex.Item>

<Button content="Calculate" primary

onClick={this.handleOperandChange}></Button>

<Text content={this.state.result}></Text>

</Flex>

<Text content="(C) Copyright Contoso" size="smallest"></Text>

</Flex>

</Provider>

);

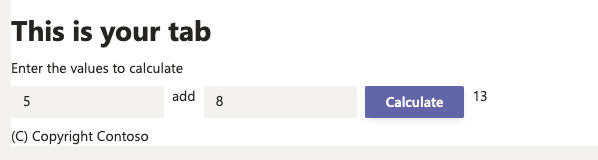
}

## Test the channel tab page

At this point, the channel tab page is complete. If the web server isn't still running, rebuild the project and start the web server by running **gulp ngrok-server**.

Open a browser, and go to [Microsoft Teams](https://teams.microsoft.com/). Sign in with the credentials of a Work and School account.

Go to the team where the tab is installed, and select the channel tab. Enter two values, and select the **Calculate** button. The results of the calculation appear next to the button.



Use the **Settings** link on the tab to open the configuration tab, and change the math operation.

Stop the local web server by selecting Ctrl+C in the console to stop the running process.

## Summary

In this exercise, you created a channel tab with a configuration page in a Microsoft Teams app.

# Exercise 3 - Implement authentication in a custom tab

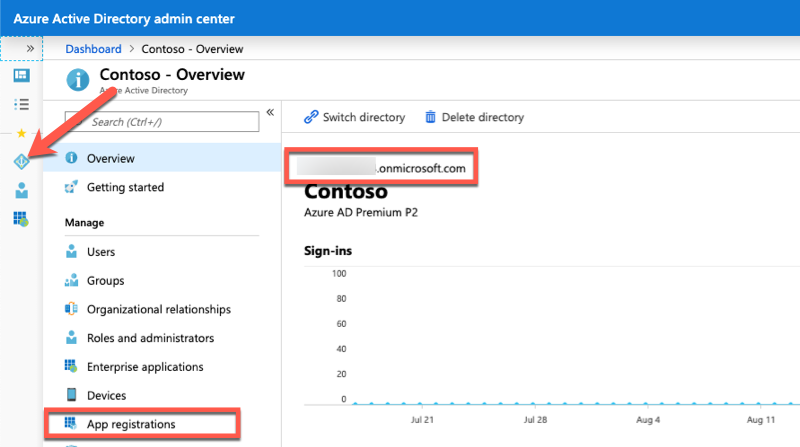
In this exercise, you'll create a custom channel tab that displays information about the current user, which was retrieved from Microsoft Graph.

## Create an Azure AD application

The tab created in this project submits a request to Microsoft Graph to retrieve email messages. All requests to Microsoft Graph must include an access token as proof of the user's identity and that they have the necessary permissions to call Microsoft Graph. To obtain an access token, you must create an Azure Active Directory (Azure AD) application that has the necessary Microsoft Graph permissions.

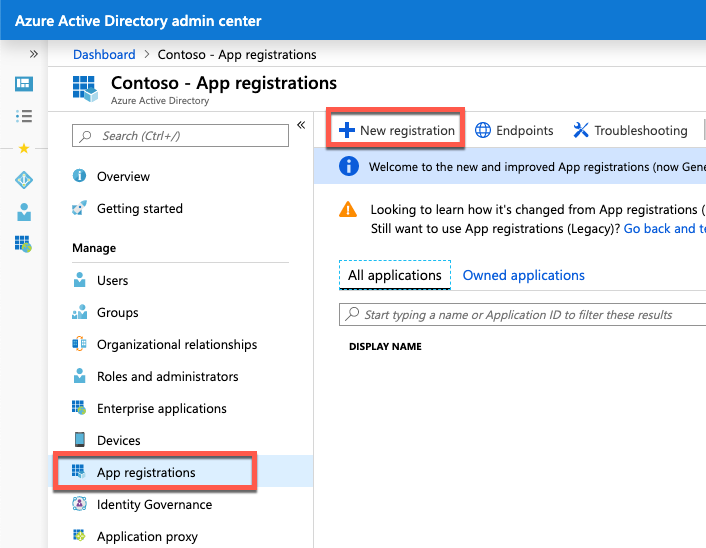
Open a browser, and go to the [Azure Active Directory admin center (https://aad.portal.azure.com)](https://aad.portal.azure.com/). Sign in by using a Work or School account that has global administrator rights to the tenancy.

Select **Azure Active Directory** in the left pane.



Select **Manage** > **App registrations** in the left pane.

On the **App registrations** page, select **New registration**.

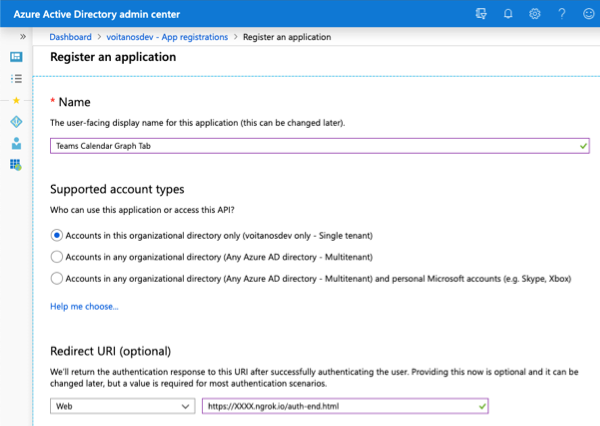


On the **Register an application** page, set the values as follows:

* **Name**: Teams Calendar Graph Tab
* **Supported account types**: Accounts in this organizational directory only (Contoso only - Single tenant)
* **Redirect URI**: Web = *https://XXXX.ngrok.io/auth-end.html*

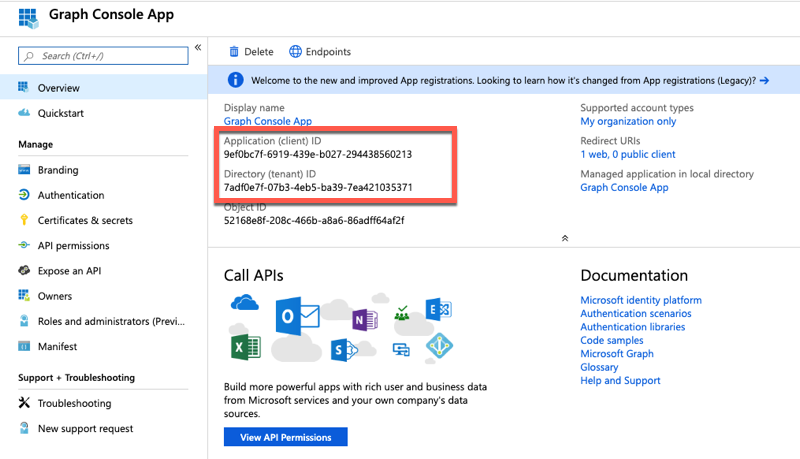
**Note**

Each time ngrok starts, it generates a new random subdomain. Azure AD requires that the redirect URI is specified in the app registration. You'll need to return to this Azure AD app registration to add or change the redirect URI after you start the ngrok utility.



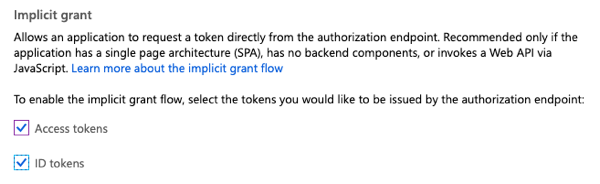
Select **Register**.

On the **Teams Calendar Graph Tab** page, copy the value of the **Application (client) ID**. You'll need it later in this exercise.



On the **Teams Calendar Graph Tab** page, select the **1 web, 0 public client** link under the **Redirect URIs**.

Locate the section **Implicit grant**, and select both **Access tokens** and **ID tokens**. This action tells Azure AD to return these tokens to the authenticated user if requested.

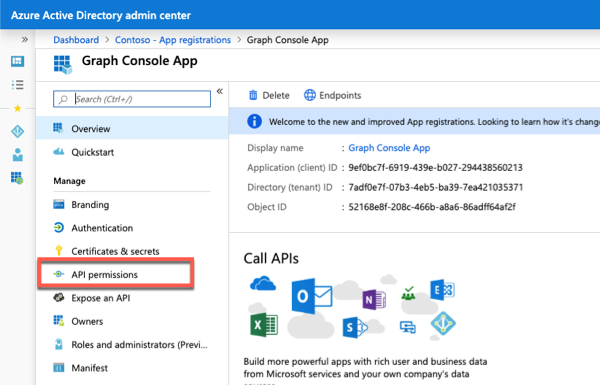


Save the settings by selecting **Save** in the upper-right corner.

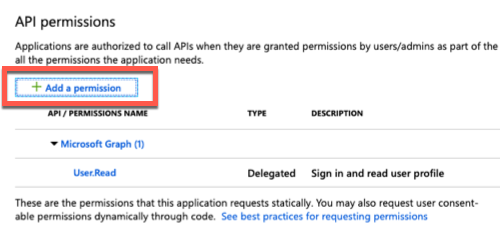
## Grant Azure AD application permissions to Microsoft Graph

After you create the application, you need to grant it the necessary permissions to Microsoft Graph.

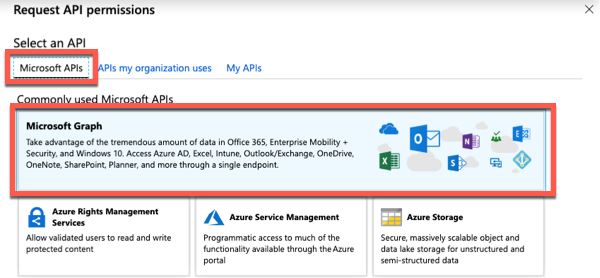
Select **API Permissions** in the left pane.



Select the **Add a permission** button.



In the **Request API permissions** panel that appears, select **Microsoft Graph** from the **Microsoft APIs** tab.



When you're prompted for the type of permission, select **Delegated permissions**.

Enter *Mail.R* in the **Select permissions** search box, and select the **Mail.Read** permission. Select the **Add permission** button at the bottom of the panel.

At the bottom of the **API Permissions** panel, select the **Grant admin consent for [tenant]** button. Select **Yes** to grant all users in your organization this permission.

## Create Microsoft Teams app

Open your command prompt, and go to a directory where you want to save your work. Create a new folder named auth-tab, and change the directory into that folder.

Run the Yeoman generator for Microsoft Teams by running the following command:

yo teams



Yeoman starts and asks you a series of questions. Answer the questions with the following values:

* **What is your solution name?**: Learn MSTeams Auth Tabs
* **Where do you want to place the files?**: Use the current folder
* **Title of your Microsoft Teams App project**: Learn MSTeams Auth Tabs
* **Your (company) name (max 32 characters)**: Contoso
* **Which manifest version would you like to use?**: 1.5
* **Enter your Microsoft Partner ID, if you have one**: (Leave blank to skip)
* **What features do you want to add to your project?**: A tab
* **The URL where you'll host this solution**: [https://larnteamsauthtabs.azurewebsites.net](https://larnteamsauthtabs.azurewebsites.net/)
* **Would you like to include Test framework and initial tests?**: No
* **Would you like to use Azure Applications Insights for telemetry?**: No
* **Default Tab name (max 16 characters)**: LearnAuthTab
* **Do you want to create a configurable or static tab?**: Configurable

After you answer the generator's questions, the generator creates the scaffolding for the project. The generator then runs npm install that downloads all the dependencies required by the project.

The tab you'll create in this exercise will get the latest emails from the current user's mailbox by using Microsoft Graph. Install the Microsoft Graph JavaScript SDK and associated TypeScript type declarations for Microsoft Graph in the project. To install these packages, run the following commands:

npm install @microsoft/microsoft-graph-client

npm install @types/microsoft-graph --save-dev

## Update the project to use the Stardust UI library

Microsoft Teams recommends that your custom apps use React and the themable React component library [Stardust UI React](https://stardust-ui.github.io/react/). To use Stardust in the Microsoft Teams app, we need to make some changes to the project.

**Important**

At the time of publication of this module, there are plans to update the Yeoman generator for Microsoft Teams to include Stardust in the default project. At the time of publication of this module, the default project uses the older Microsoft Teams control library that Stardust is replacing.

The steps in this section might not be necessary because the Yeoman generator for Microsoft Teams might have been updated. Review each of the instructions in this section and compare the results with your project to determine if they're necessary.

The first step is to uninstall the existing control library and install the Stardust library. Run the following two commands in the command line from the root folder of the project:

npm uninstall msteams-ui-components-react

npm install @stardust-ui/react

Locate and open the file that contains the React component used in the project: ./src/app/scripts/learnAuthTab/LearnAuthTab.tsx.

Update the import statements in this file to replace the component library used. Find the following import statement that imports the legacy Microsoft Teams UI Components - React library:

import {

PrimaryButton,

TeamsThemeContext,

Panel,

PanelBody,

PanelHeader,

PanelFooter,

Surface,

getContext

} from "msteams-ui-components-react";

Replace the previous statement with the following import statement:

import {

Flex, Provider, themes, ThemePrepared,

Header,

Button, Icon, List

} from "@stardust-ui/react";

The default project contains additional user interface style code that used the previous control library. This code is no longer necessary.

Locate the following code in the componentWillMount() method in the LearnAuthTab class and delete it:

this.setState({

fontSize: this.pageFontSize()

});

Locate the following code in the render() method in the LearnAuthTab class and delete it:

const context = getContext({

baseFontSize: this.state.fontSize,

style: this.state.theme

});

const { rem, font } = context;

const { sizes, weights } = font;

const styles = {

header: { ...sizes.title, ...weights.semibold },

section: { ...sizes.base, marginTop: rem(1.4), marginBottom: rem(1.4) },

footer: { ...sizes.xsmall }

};

Locate the return () statement in the render() method in the LearnAuthTab class and delete the contents. This code used the UI library that you replaced with Stardust. At this point, the render() method should look like the following code:

public render() {

return (

);

}

Update the state of the component to contain a property for the current Stardust theme. Locate the ILearnAuthTabState interface in the LearnAuthTab.tsx file, and add the following member to it:

teamsTheme: ThemePrepared;

Add the following method to the LearnAuthTab class that updates the component state to the Stardust theme that matches the currently selected Microsoft Teams client theme:

private updateStardustTheme = (teamsTheme: string = "default"): void => {

let stardustTheme: ThemePrepared;

switch (teamsTheme) {

case "default":

stardustTheme = themes.teams;

break;

case "dark":

stardustTheme = themes.teamsDark;

break;

case "contrast":

stardustTheme = themes.teamsHighContrast;

break;

default:

stardustTheme = themes.teams;

break;

}

// update the state

this.setState(Object.assign({}, this.state, {

teamsTheme: stardustTheme

}));

}

Initialize the current theme and state of the component. Locate the line this.updateTheme(this.getQueryVariable("theme")); and replace it with the following code in the componentWillMount() method:

this.updateStardustTheme(this.getQueryVariable("theme"));

Within the componentWillMount() method, locate the following line:

microsoftTeams.registerOnThemeChangeHandler(this.updateTheme);

This code registers an event handler to update the component's theme to match the theme of the current Microsoft Teams client when this page is loaded as a tab. Update this line to call the new handler in the following line to register another handler to update the Stardust library theme:

microsoftTeams.registerOnThemeChangeHandler(this.updateStardustTheme);

With the theme management and state initialized, we can now implement the user interface.

## Implement the tab's logic and user interface

Now you can implement the user interface for the tab. The simple tab has a basic interface. It presents a list of email messages for the current user and a button to initiate the request.

Add the following import statements after the existing import statements in the LearnAuthTab.tsx file. These statements include the Microsoft Graph JavaScript SDK and associated TypeScript type declarations into the file:

import \* as MicrosoftGraphClient from "@microsoft/microsoft-graph-client";

import \* as MicrosoftGraph from "microsoft-graph";

Locate the ILearnAuthTabState interface, and add the following members to it. These properties are used to store the OAuth access token used to authenticate with and the email messages returned from Microsoft Graph.

accessToken: string;

messages: MicrosoftGraph.Message[];

Add the following code to the top of the LearnAuthTab class. This action creates a new class-scoped member of the Microsoft Graph client and initializes the state of the component.

private msGraphClient: MicrosoftGraphClient.Client;

constructor(props: ILearnAuthTabProps, state: ILearnAuthTabState) {

super(props, state);

state.messages = [];

state.accessToken = "";

this.state = state;

}

Locate the render() method, and update the return statement to the following code. The render() method displays a button for the user to select to sign in and request their emails from Microsoft Graph. It then displays the email messages in a list.

public render() {

return (

<Provider theme={themes.teams}>

<Flex column gap="gap.small">

<Header>Recent messages in current user's mailbox</Header>

<Button primary

content="Get My Messages"

onClick={this.handleGetMyMessagesOnClick}></Button>

<List selectable>

{

this.state.messages.map(message => (

<List.Item media={<Icon name="email"></Icon>}

header={message.receivedDateTime}

content={message.subject}>

</List.Item>

))

}

</List>

</Flex>

</Provider>

);

}

Add the onclick event handler for the button to the LearnAuthTab class.

private handleGetMyMessagesOnClick = async (event): Promise<void> => {

await this.getMessages();

}

## Implement the authentication and Microsoft Graph request logic

At this point, the tab is ready to add the logic necessary to request the email messages for the current user. Before you request email messages from Microsoft Graph, you need the user to sign in and obtain an access token from Azure AD. There are multiple steps to perform to implement the authentication routine.

Start by adding the following code to the end of componentWillMount() to initialize the Microsoft Graph client:

// init the graph client

this.msGraphClient = MicrosoftGraphClient.Client.init({

authProvider: async (done) => {

if (!this.state.accessToken) {

const token = await this.getAccessToken();

this.setState({

accessToken: token

});

}

done(null, this.state.accessToken);

}

});

Next, add the following method to the LearnAuthTab class:

private async getMessages(promptConsent: boolean = false): Promise<void> {

if (promptConsent || this.state.accessToken === "") {

await this.signin(promptConsent);

}

this.msGraphClient

.api("me/messages")

.select(["receivedDateTime", "subject"])

.top(15)

.get(async (error: any, rawMessages: any, rawResponse?: any) => {

if (!error) {

this.setState(Object.assign({}, this.state, {

messages: rawMessages.value

}));

Promise.resolve();

} else {

console.error("graph error", error);

// re-sign in but this time force consent

await this.getMessages(true);

}

});

}

The getMessages() method first checks if the component has an access token. If so, it submits the request to Microsoft Graph for the top 15 email messages. Otherwise, if the component doesn't have an access token, it calls the signin() method.

Add the following code to implement the signin() method:

private async signin(promptConsent: boolean = false): Promise<void> {

const token = await this.getAccessToken(promptConsent);

this.setState({

accessToken: token

});

Promise.resolve();

}

This method calls the getAccessToken() method that uses the Microsoft Teams JavaScript SDK to initiate the authentication process. It opens a pop-up window that loads the **auth-start.html** page to start the authentication process with Azure AD. Ultimately, the authentication process ends in the pop-up window and results in either a successful or failed authentication process. In either case, the associated callback handlers are registered in the authenticate() method in the following code:

private async getAccessToken(promptConsent: boolean = false): Promise<string> {

return new Promise<string>((resolve, reject) => {

microsoftTeams.authentication.authenticate({

url: window.location.origin + "/auth-start.html",

width: 600,

height: 535,

successCallback: (accessToken: string) => {

resolve(accessToken);

},

failureCallback: (reason) => {

reject(reason);

}

});

});

}

Create the new file ./src/app/web/auth-start.html in the project, and add the following code to it. This file uses the Microsoft Teams JavaScript SDK and Azure Active Directory Authentication Library JavaScript (ADAL.js) libraries to configure ADAL for the Azure AD application created previously in this exercise. It then redirects the user to the Azure AD sign-in page and instructs the page to redirect the user back to **auth-end.html** on our site.

<!DOCTYPE html>

<html>

<body>

<script src="https://statics.teams.cdn.office.net/sdk/v1.5.2/js/MicrosoftTeams.min.js" crossorigin="anonymous"></script>

<script src="https://secure.aadcdn.microsoftonline-p.com/lib/1.0.17/js/adal.min.js" crossorigin="anonymous"></script>

<script type="text/javascript">

microsoftTeams.initialize();

microsoftTeams.getContext(function (msTeamsContext) {

// ADAL.js configuration

let config = {

clientId: "{{REPLACE\_AZUREAD\_APP\_ID}}",

redirectUri: window.location.origin + "/auth-end.html",

cacheLocation: "localStorage",

endpoints: {

"https://graph.microsoft.com": "https://graph.microsoft.com"

}

};

// add extra query parameters Azure AD login request

// include scope for OpenID connect and log-in hint by using the current Microsoft Teams logged-in user

config.extraQueryParameters = "scope=open+profile";

if (msTeamsContext.upn) {

config.extraQueryParameters += "&login-hint=" + encodeURIComponent(msTeamsContext.userProfileName);

}

// check if consent required for new permission

if (getUrlParameter('prompt') !== "") {

config.extraQueryParameters += "&prompt=" + getUrlParameter('prompt');

}

// override URL to Azure AD auth endpoint to include extra query parameters

config.displayCall = function (urlNavigate) {

if (urlNavigate) {

if (config.extraQueryParameters) {

urlNavigate += "&" + config.extraQueryParameters;

}

window.location.replace(urlNavigate);

}

}

// login

let authContext = new AuthenticationContext(config);

authContext.clearCache();

authContext.login();

});

function getUrlParameter(name) {

name = name.replace(/[\[]/, '\\[').replace(/[\]]/, '\\]');

var regex = new RegExp('[\\?&]' + name + '=([^&#]\*)');

var results = regex.exec(location.search);

return results === null ? '' : decodeURIComponent(results[1].replace(/\+/g, ' '));

};

</script>

</body>

</html>

Create the new file ./src/app/web/auth-end.html in the project, and add the following code to it. Like the auth-start.html file, this file uses the Microsoft Teams JavaScript SDK and ADAL.js libraries to configure ADAL for the Azure AD application created previously in this exercise. It parses the results received from Azure AD. If the user successfully authenticated, this page requests an access token for Microsoft Graph from Azure AD and then notifies Microsoft Teams that the authentication process succeeded or failed.

The notification process triggers Microsoft Teams to close the pop-up window and run the registered callback handlers in our tab:

<!DOCTYPE html>

<html>

<body>

<script src="https://statics.teams.cdn.office.net/sdk/v1.5.2/js/MicrosoftTeams.min.js" crossorigin="anonymous"></script>

<script src="https://secure.aadcdn.microsoftonline-p.com/lib/1.0.17/js/adal.min.js" crossorigin="anonymous"></script>

<script type="text/javascript">

microsoftTeams.initialize();

// ADAL.js configuration

let config = {

clientId: "{{REPLACE\_AZUREAD\_APP\_ID}}",

cacheLocation: "localStorage",

navigateToLoginRequestUrl: false,

endpoints: {

"https://graph.microsoft.com": "https://graph.microsoft.com"

}

};

let authContext = new AuthenticationContext(config);

// ensure page loaded via Azure AD callback

if (authContext.isCallback(window.location.hash)) {

authContext.handleWindowCallback(window.location.hash);

// Only call notifySuccess or notifyFailure if this page is in the authentication pop-up

if (window.opener) {

// if able to retrieve current user...

if (authContext.getCachedUser()) {

// get access token for Microsoft Graph

authContext.acquireToken("https://graph.microsoft.com", function (error, token) {

if (token) {

microsoftTeams.authentication.notifySuccess(token);

} else if (error) {

microsoftTeams.microsoftTeams.notifyFailure(error);

} else {

microsoftTeams.authentication.notifyFailure("UnexpectedFailure");

}

});

} else {

microsoftTeams.authentication.notifyFailure(authContext.getLoginError());

}

}

}

</script>

</body>

</html>

**Important**

Make sure to replace the string {{REPLACE\_AZUREAD\_APP\_ID}} in the previous two code listings with the ID of the Azure AD application you created previously in this exercise.

Open tsconfig-client.json file present in the project’s root directory. Locate the following statement:

"target": "es5",

Replace this statement with the following statement and save the file:

"target": "es2015",

## Install and test the Microsoft Teams app

From the command line, go to the root folder for the project and run the following command:

gulp ngrok-serve

Copy the ngrok URL displayed in the console. Go back to Azure AD, and add or update the redirect URI of the Azure AD application previously created in this lab. Otherwise, Azure AD won't redirect you back to the **auth-end.html** page. The URL should be in the form of https://{ngrok-subdomain}.ngrok.io/auth-end.html.

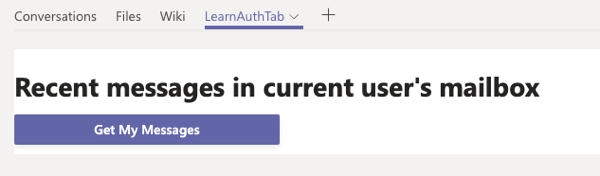
In the browser, go to [Microsoft Teams](https://teams.microsoft.com/), and sign in with the credentials of a Work and School account.

Using the app bar on the left, select the **More added apps** button. Then select **More apps** > **Upload a custom app** > **Upload for me or my teams**.

In the file dialog box that appears, select the Microsoft Teams package in your project. This app package is a zip file that can be found in the project's ./package folder.

After the package is uploaded, Microsoft Teams displays a summary of the app. Select the **Add** button to install the app. Select a team to add the channel to, and select **Save** on the configuration page.

Select the app to go to the new tab.

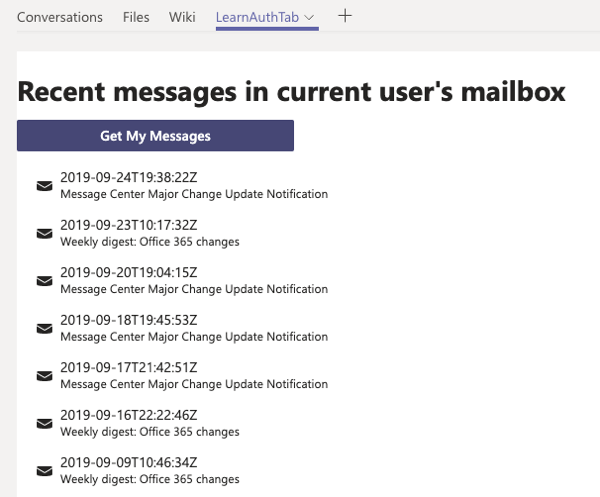


Select the **Get My Messages** button. Microsoft Teams opens the pop-up window that will be redirected to Azure AD for sign-in. If you're prompted to sign in, enter the credentials of your Work and School account.

**Note**

You might not be prompted to sign in because you're already signed in to Microsoft Teams. Azure AD won't require you to sign in again and redirects you to the specified redirect URI for the Azure AD application.

After you've successfully signed in to Azure AD, Microsoft Teams closes the pop-up window and displays the last 15 emails in your mailbox retrieved from Microsoft Graph.



## Summary

In this exercise, you created a custom channel tab that displays information about the current user, which is retrieved from Microsoft Graph.