Lab 4: Collect Input in Microsoft Teams with Task Modules

Contents

[Exercise 1 - Collecting user input with task modules 3](#_Toc33100537)

[Create Microsoft Teams app 4](#_Toc33100538)

[Test the personal tab 5](#_Toc33100539)

[Update the project to use the Stardust UI library 10](#_Toc33100540)

[Implement the personal tab's user interface 11](#_Toc33100541)

[Test the personal tab 14](#_Toc33100542)

[Add video player task module 14](#_Toc33100543)

[Test the video player task module 16](#_Toc33100544)

[Add video selector task module 17](#_Toc33100545)

[Create the task module's React app web page host 17](#_Toc33100546)

[Register the new page 18](#_Toc33100547)

[Implement the React app for the selector task module 19](#_Toc33100548)

[Test the video selector task module 21](#_Toc33100549)

[Summary 23](#_Toc33100550)

[Exercise 2- Using adaptive cards and deep links in task modules 23](#_Toc33100551)

[Create video selector as an adaptive card 23](#_Toc33100552)

[Create a new task module that uses the Adaptive Card 27](#_Toc33100553)

[Test the Adaptive Card task module 29](#_Toc33100554)

[Invoking task modules with deep links 29](#_Toc33100555)

[Summary 32](#_Toc33100556)

[Exercise 3- Using task modules with bots 32](#_Toc33100557)

[Register a new bot in Microsoft Azure 32](#_Toc33100558)

[Register the bot with Microsoft Azure's Bot Framework 32](#_Toc33100559)

[Enable the Microsoft Teams channel for the bot 36](#_Toc33100560)

[Retrieve the bot app id and password 37](#_Toc33100561)

[Create a client secret for the app 38](#_Toc33100562)

[Update the Microsoft Teams app project with the app details 39](#_Toc33100563)

[Add a bot to the project 39](#_Toc33100564)

[Code the bot 40](#_Toc33100565)

[Update the webserver to load the bot 42](#_Toc33100566)

[Test the bot in Microsoft Teams 43](#_Toc33100567)

[Add the video player task module 47](#_Toc33100568)

[Test the video player task module 49](#_Toc33100569)

[Add the video selector Adaptive Card task module 51](#_Toc33100570)

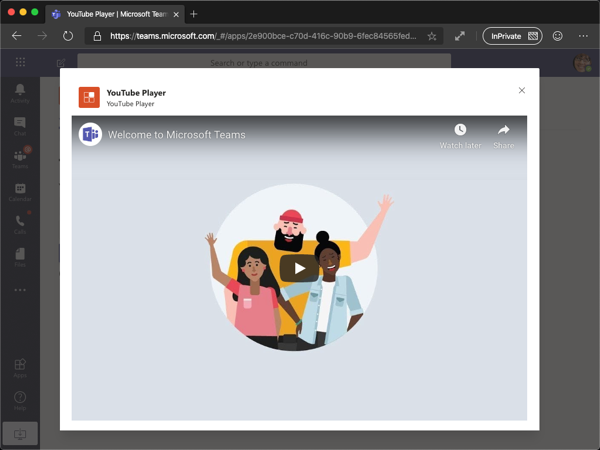
[Test the video selector Adaptive Card task module 53](#_Toc33100571)

[Summary 54](#_Toc33100572)

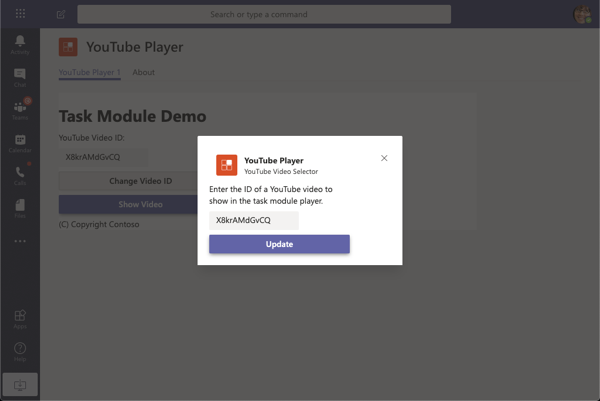
# Exercise 1 - Collecting user input with task modules

In this exercise, you’ll learn the basics of task modules in Microsoft Teams and how to collect input from users in a custom Teams tab. After creating a new Microsoft Teams personal tab, you'll add two task modules to it.

One is a standard HTML page that accepts the ID of a video on YouTube. When the task module is invoked, it will display the video using the YouTube embedded player. This task module will get the video ID from the query string, but it will not need to return any information back to the tab.



The other task module is implemented using React, the same way custom tabs are implemented using the Yeoman Generator for Microsoft Teams. This task module enables the user to specify the ID of the YouTube video to display. Once changed, when the user saves their changes, it will use the callback to close submit the new ID back to the tab.



## Create Microsoft Teams app

Open your command prompt, navigate to a directory where you want to save your work, create a new folder **learn-msteams-taskmodules**, and change directory into that folder.

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

yo teams



Yeoman will launch and ask you a series of questions. Answer the questions with the following values:

* **What is your solution name?**: YouTubePlayer
* **Where do you want to place the files?**: Use the current folder
* **Title of your Microsoft Teams App project?**: YouTube Player
* **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://youtubeplayer.azurewebsites.net](https://youtubeplayer.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)**: YouTube Player 1
* **Do you want to create a configurable or static tab?**: Static

**Note**

Most of the answers to these questions can be changed after creating 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 answering the generator's questions, the generator will create the scaffolding for the project and then execute npm install that downloads all the dependencies required by the project.

## Test the personal tab

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

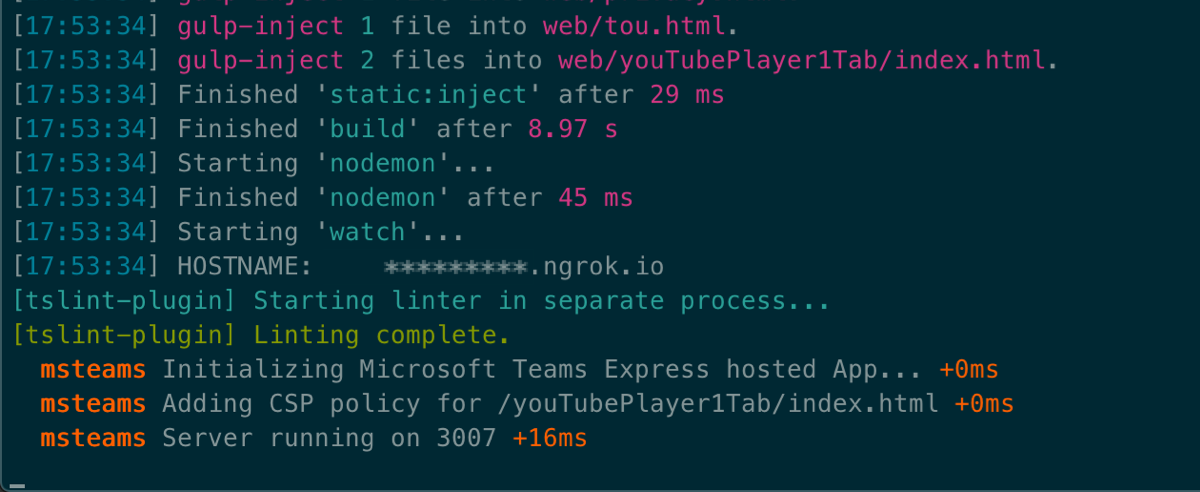
From the command line, navigate to the root folder for the project and execute the following command:

gulp ngrok-serve

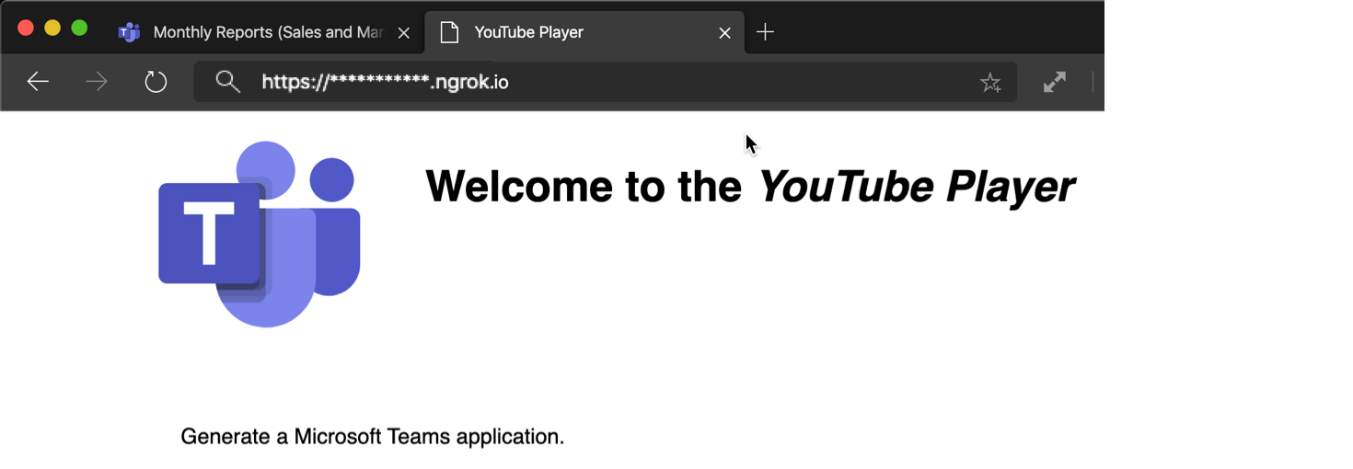
This gulp task will run many other tasks all displayed within the command-line console. The **ngrok-serve** task builds your project and starts a local web server (http://localhost:3007). It then starts ngrok with a random subdomain that creates a secure URL to your local webserver.

**Note**

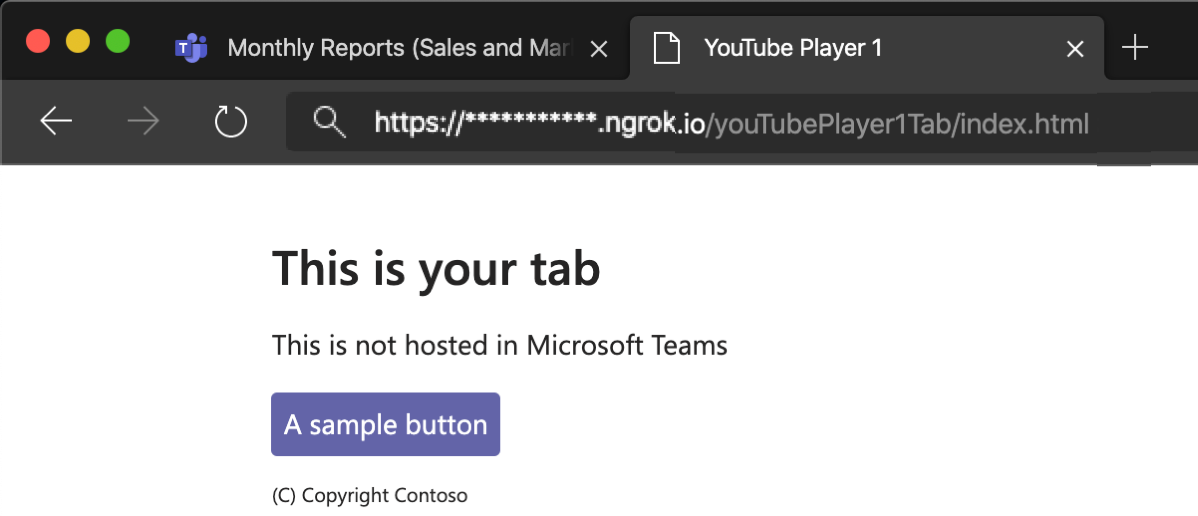
Microsoft Teams requires all content displayed within a tab be loaded from an HTTPS request. In development, can be done using the tool [**ngrok**](https://www.ngrok.com/) that creates a secure rotatable URL to your local HTTP webserver. Ngrok is included as a dependency within the project so there is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



Open a browser and navigate 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.

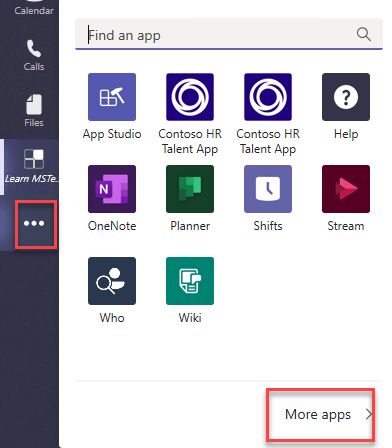


Now let's load the tab in Microsoft Teams. In the browser, navigate to [**https://teams.microsoft.com**](https://teams.microsoft.com/) and sign in with the credentials of a Work and School account.

**Note**

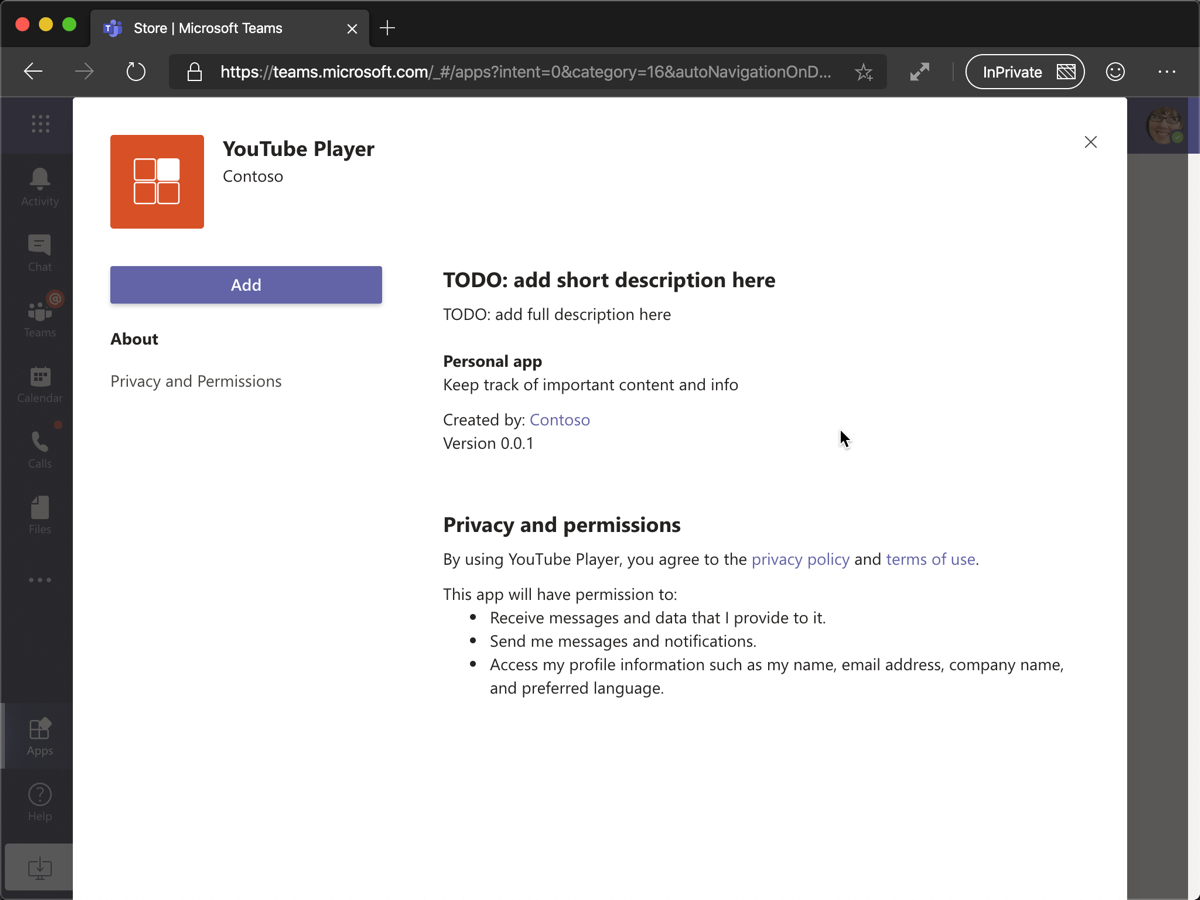
Microsoft Teams is available for use as a web client, desktop client and a mobile client. In this module, we will use the web client but any of the clients can be used.

Using the app bar navigation menu, select the **More added apps** button. Then select **More apps** followed by **Upload a custom app** > **Upload for me or my teams**.

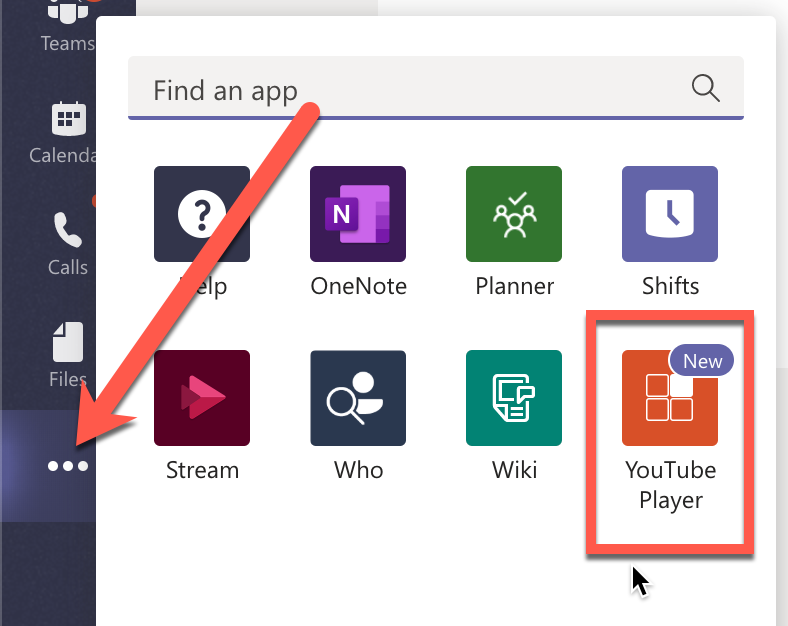


In the file dialog that appears, select the Microsoft Teams package in your project. This app package is a ZIP file that can be found in the project's **./package** folder.

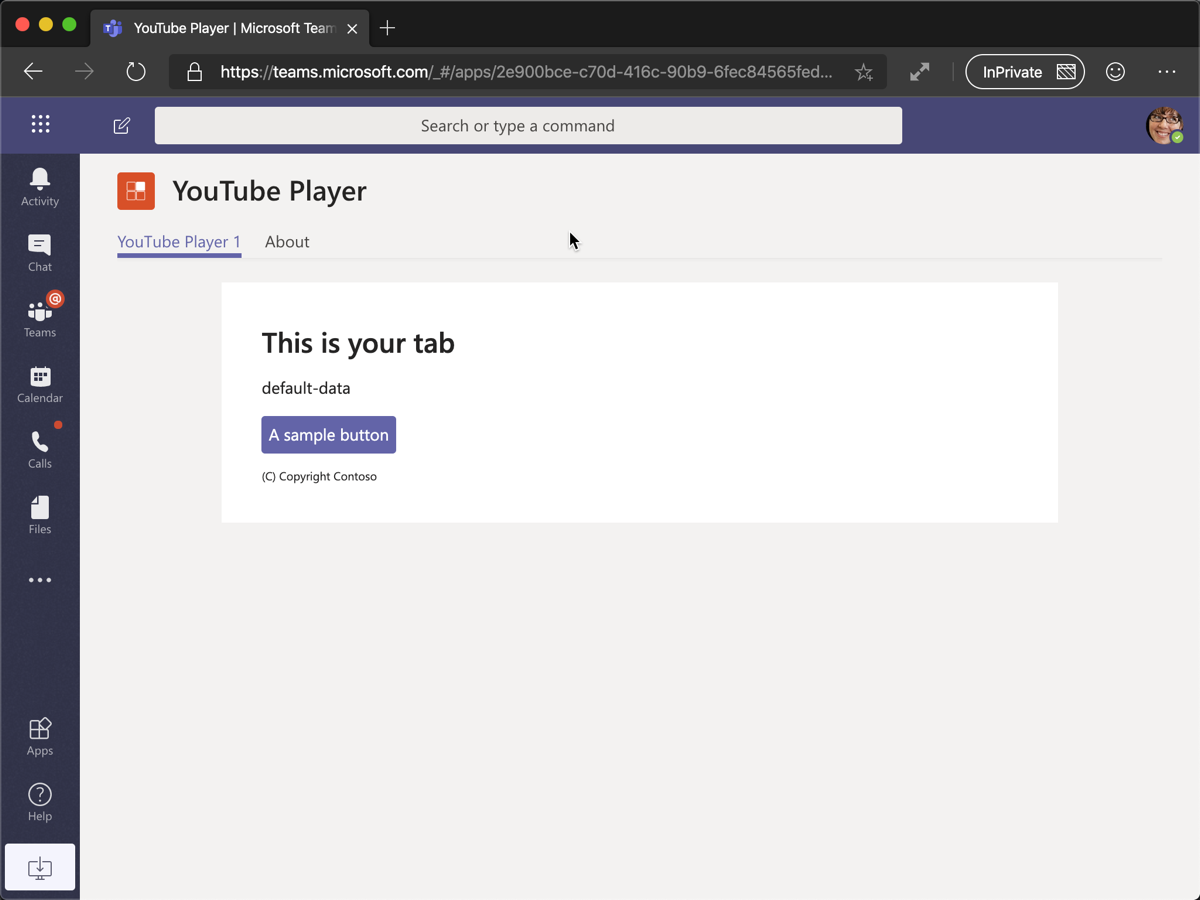
Once the package is uploaded, Microsoft Teams will display a summary of the app. Here you can see some "todo" items to address. *None of these "todo" items are important to this exercise, so you will leave them as is.*



Select the **Add** button to install the app, adding a new personal tab to your **More added apps** dialog:



Select the app to navigate to the new tab:



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

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

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

## Update the project to use the Stardust UI library

Microsoft Teams recommends 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. However, at the time of publication of this module, the default project uses the older Microsoft Teams control library that Stardust is replacing.

Therefore, the steps in this section may or may not be necessary as the Yeoman generator for Microsoft Teams may have been updated. Review each of the instructions in this section and compare the results with your project to determine if they are necessary.

The first step is to uninstall the existing control library and install the Stardust library. Execute 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/youTubePlayer1Tab/YouTubePlayer1Tab.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 above statement with the following import statement:

import {

Flex, Provider, themes, ThemePrepared,

Header, Button, Input, 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 YouTubePlayer1Tab class and delete it:

this.setState({

fontSize: this.pageFontSize()

});

Locate the following code in the render() method in the YouTubePlayer1Tab 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 YouTubePlayer1Tab 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 will have a basic interface.

First, update the state of the component to contain a list of items and a property for a new item. Locate the IYouTubePlayer1TabState interface in the **YouTubePlayer1Tab.tsx** file and add the following properties to it:

teamsTheme: ThemePrepared;

todoItems: string[];

newTodoValue: string;

youTubeVideoId?: string;

Add the following method to the YouTubePlayer1Tab 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"));

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 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 will now display the list of items in our state out with a brief copyright statement:

public render() {

return (

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

<Flex column gap="gap.smaller">

<Header>Task Module Demo</Header>

<Text>YouTube Video ID:</Text>

<Input value={this.state.youTubeVideoId} disabled></Input>

<Button content="Change Video ID" onClick={this.onChangeVideo}></Button>

<Button content="Show Video" primary onClick={this.onShowVideo}></Button>

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

</Flex>

</Provider>

);

}

The next step is to add some interactivity to the tab. Add the following methods to the YouTubePlayer1Tab class. These methods will handle updating the state when specific events happen on the form you'll add to the component:

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

}

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

}

Finally, initialize the state of the tab by setting it to a default video to display.

Within the componentWillMount() method, add the following statement to the top of the method:

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

youTubeVideoId: "X8krAMdGvCQ"

}));

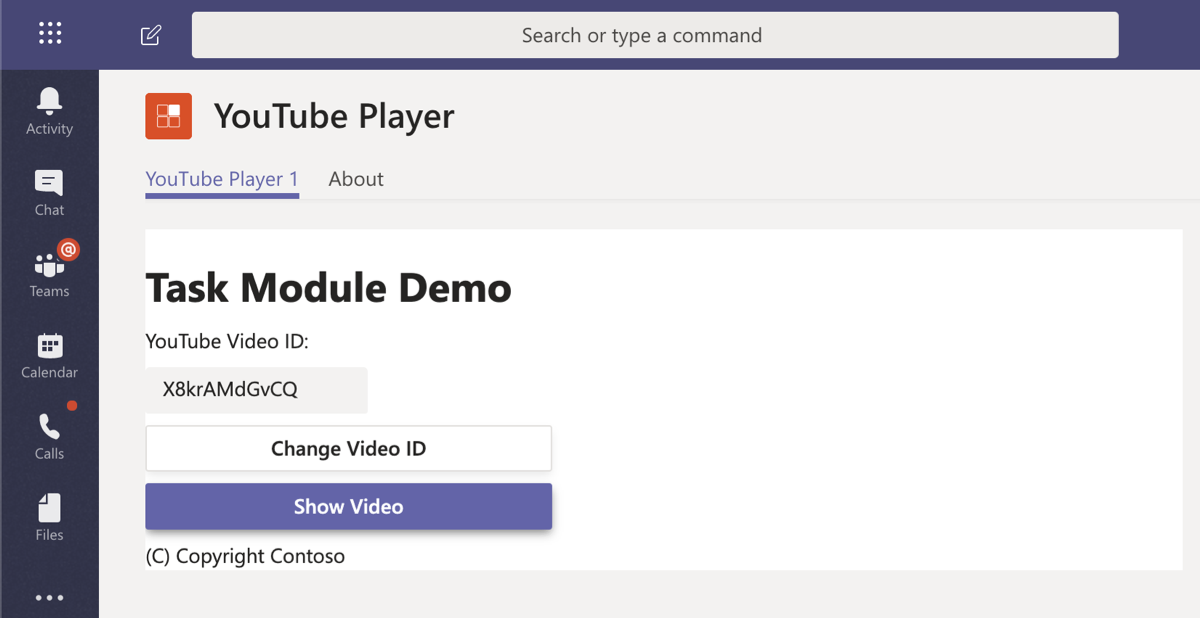
## Test the personal tab

At this point, our Microsoft Teams app, implemented as a custom person tab is set up and working. Verify this by starting ngrok again and refreshing the Microsoft Teams interface.

From the command line, navigate to the root folder for the project and execute the following command:

gulp ngrok-serve

Refresh the Microsoft Teams interface and notice the new UI you've implemented for the tab:



Now you can update the project and add task modules to the custom Microsoft Teams app.

Stop the local web server by pressing CTRL+C in the console to stop the running process.

## Add video player task module

A task module can be a web page implemented with HTML and JavaScript. Create the video player task module by creating a new file, **player.html** in the **./src/app/web/youTubePlayer1Tab** folder in your project.

Add the following HTML to the **player.html** file:

<!DOCTYPE html>

<html lang="en">

<head>

<title>YouTube Player Task Module</title>

<style>

#embed-container iframe {

position: absolute;

top: 0;

left: 0;

width: 95%;

height: 95%;

padding-left: 20px;

padding-right: 20px;

padding-top: 10px;

padding-bottom: 10px;

border-style: none;

}

</style>

</head>

<body>

<div id="embed-container"></div>

</body>

</html>

The video player task module will use the YouTube embedded player to show the specified video. The video will be defined in the query string when the **player.html** file is loaded.

Implement the <iframe> embedded video player by adding the following JavaScript before the closing </body> tag in the **player.html** file:

<script>

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, ' '));

};

var element = document.createElement("iframe");

element.src = "https://www.youtube.com/embed/" + getUrlParameter("vid");

element.width = "1000";

element.height = "700";

element.frameborder = "0";

element.allow = "autoplay; encrypted-media";

element.allowfullscreen = "";

document.getElementById("embed-container").appendChild(element);

</script>

Now, implement the task module in the personal tab.

Locate and open the **./src/app/scripts/youTubePlayer1Tab/YouTubePlayer1Tab.tsx** file.

First, add the following utility method to the YouTubePlayer1Tab class:

private appRoot(): string {

if (typeof window === "undefined") {

return "https://{{HOSTNAME}}";

} else {

return window.location.protocol + "//" + window.location.host;

}

}

Next, add the following code to the onShowVideo method:

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

const taskModuleInfo = {

title: "YouTube Player",

url: this.appRoot() + `/youTubePlayer1Tab/player.html?vid=${this.state.youTubeVideoId}`,

width: 1000,

height: 700

};

microsoftTeams.tasks.startTask(taskModuleInfo);

}

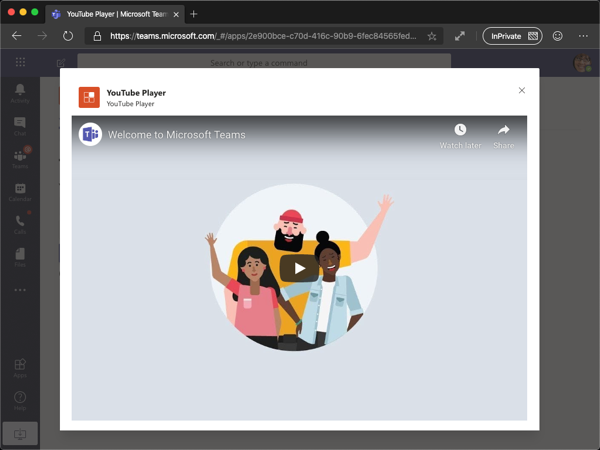
This code will create a new taskInfo object with the details of the task module. It will then launch the task module. This task module does nothing but display information, so we don't need to implement the callback.

## Test the video player task module

From the command line, navigate to the root folder for the project and execute the following command:

gulp ngrok-serve

Refresh the Microsoft Teams interface. Select the **Show video** button. Microsoft Teams will load the video player task module with the specified video loaded in the embedded player:



Stop the local web server by pressing CTRL+C in the console to stop the running process.

## Add video selector task module

Now let's update the project to include a task module that will enable the user to change the video loaded in the player task module. For this task module, you'll implement it similar to how the custom tab is implemented: as a React app.

## Create the task module's React app web page host

Create a new file, **selector.html**, in the **./src/app/web/youTubePlayer1Tab** folder.

Add the following HTML to it:

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>YouTube Video Selector</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<!-- inject:css -->

<!-- endinject -->

<style>

#app {

margin-left: 20px;

margin-right: 20px;

margin-top: 10px;

margin-bottom: 10px;

}

</style>

</head>

<body>

<div id='app'>

Loading...

</div>

<!-- inject:js -->

<!-- endinject -->

<script type='text/javascript'>

youTubePlayer.VideoSelectorTaskModule.render(document.getElementById('app'), {});

</script>

</body>

</html>

## Register the new page

Next, register the page you created in the last step with the project's hosting infrastructure. This will also add the necessary HTTP headers to the page's response to ensure it can be loaded within an IFRAME, but only within a Microsoft Teams client.

Create a new file, **VideoSelectorTaskModule.ts**, in the **./src/app/youTubePlayer1Tab** folder.

Add the following code to the file:

import { PreventIframe } from "express-msteams-host";

@PreventIframe("/youTubePlayer1Tab/selector.html")

export class VideoSelectorTaskModule { }

Now register the page by adding the following line to the end of the **./src/app/TeamsAppsComponents.ts** file:

export \* from "./youTubePlayer1Tab/VideoSelectorTaskModule";

## Implement the React app for the selector task module

With the selector's page created and registered, the next step is to implement the React app that is loaded in the page.

Create a new file, **VideoSelectorTaskModule.tsx**, in the folder **./src/app/scripts/youTubePlayer1Tab**.

Add the following code to the page. Most of this code mirrors what you would see if you created a new tab.

import \* as React from "react";

import {

Flex, Provider, themes, ThemePrepared,

Button, Input, Text

} from "@stardust-ui/react";

import TeamsBaseComponent, { ITeamsBaseComponentProps, ITeamsBaseComponentState } from "msteams-react-base-component";

import \* as microsoftTeams from "@microsoft/teams-js";

export interface IVideoSelectorTaskModuleState extends ITeamsBaseComponentState {

teamsTheme: ThemePrepared;

youTubeVideoId?: string;

}

export interface IVideoSelectorTaskModuleProps extends ITeamsBaseComponentProps {

}

export class VideoSelectorTaskModule extends TeamsBaseComponent<IVideoSelectorTaskModuleProps, IVideoSelectorTaskModuleState> {

public componentWillMount(): void {

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

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

youTubeVideoId: this.getQueryVariable("vid")

}));

if (this.inTeams()) {

microsoftTeams.initialize();

microsoftTeams.registerOnThemeChangeHandler(this.updateStardustTheme);

}

}

public render() {

return (

);

}

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

}));

}

}

Implement the user interface of the task module by adding the following code to the return statement of render() method:

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

<Flex column gap="gap.smaller">

<Text size="medium">

Enter the ID of a YouTube video to show in the task module player.

</Text>

<Input value={this.state.youTubeVideoId} onChange={this.handleOnChanged}></Input>

<Button content="Update" primary onClick={this.handleOnClick}></Button>

</Flex>

</Provider>

Next, implement the two handlers referenced in the render() method. Add these two functions to the VideoSelectorTaskModule class:

private handleOnChanged = (event): void => {

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

youTubeVideoId: event.target.value

}));

}

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

microsoftTeams.tasks.submitTask(this.state.youTubeVideoId, undefined);

}

The handleOnChanged() method updates the state with the value specified in the input control, while the handleOnClick() method uses the Microsoft Teams SDK to pass the ID of the video back to the personal tab.

Make this React class available to the rest of the application by adding the following line to the **./src/app/scripts/client.ts** file:

export \* from "./youTubePlayer1Tab/VideoSelectorTaskModule";

The last step is to wire this task module up to the tab. Within the **./src/app/scripts/youTubePlayer1Tab/YouTubePlayer1Tab.tsx** file, locate the method onChangeVideo(). Add the following code to the method:

const taskModuleInfo = {

title: "YouTube Video Selector",

url: this.appRoot() + `/youTubePlayer1Tab/selector.html?theme={theme}&vid=${this.state.youTubeVideoId}`,

width: 350,

height: 150

};

const submitHandler = (err: string, result: string): void => {

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

youTubeVideoId: result

}));

};

microsoftTeams.tasks.startTask(taskModuleInfo, submitHandler);

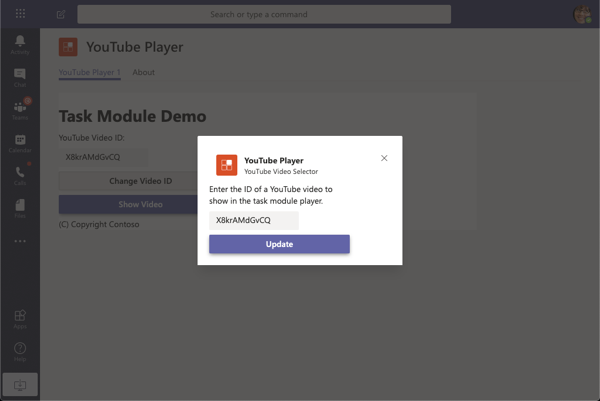
This code will first create the taskInfo object that defines the task module. It also creates a callback that will take the result from the task module and use it to update the state of the React app.

## Test the video selector task module

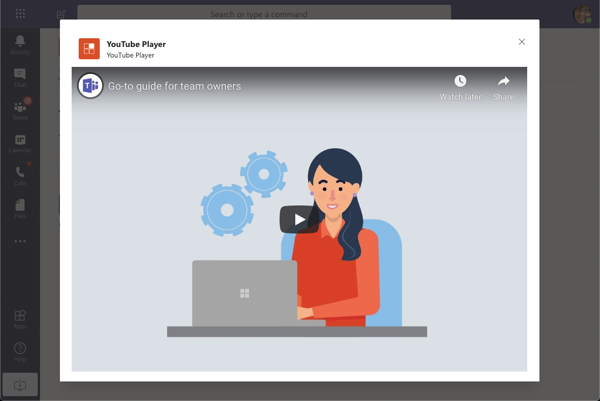
From the command line, navigate to the root folder for the project and execute the following command:

gulp ngrok-serve

Refresh the Microsoft Teams interface. Select the **Change video ID** button. Microsoft Teams will load the video selector task module with the ID of the currently selected video in the text control.



Enter the ID of another YouTube video and select the **Update** button. Notice the ID of the video has changed. Now when you select the **Show video** button, the player task module loads with the new video.



Stop the local web server by pressing CTRL+C in the console to stop the running process.

## Summary

In this exercise, you learned the basics of task modules in Microsoft Teams and how to collect input from users in a custom Teams tab. After creating a new Microsoft Teams personal tab, you added two task modules to it.

# Exercise 2- Using adaptive cards and deep links in task modules

In this exercise, you’ll learn how to use adaptive cards in a custom task module in a custom Microsoft Teams app. You’ll also learn how to invoke task modules from anywhere within Microsoft Teams using deep links.

**Important**

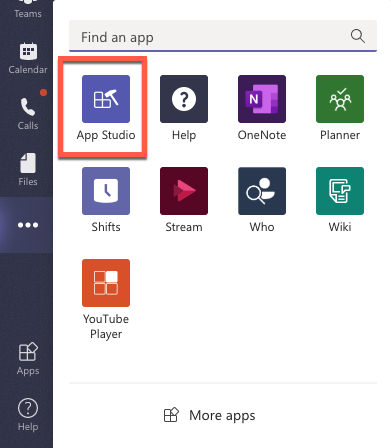
This exercise assumes you have created the Microsoft Teams app project with the Yeoman generator that contains a personal tab from the previous exercise in this module. You'll update the project to add a new task module that uses an Adaptive Card.

## Create video selector as an adaptive card

In this section, you'll create an Adaptive Card that mirrors the functionality of the video selector task module created in a previous exercise. Adaptive Cards are defined using JSON that can be written by hand or using the Microsoft Teams App Studio app.

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

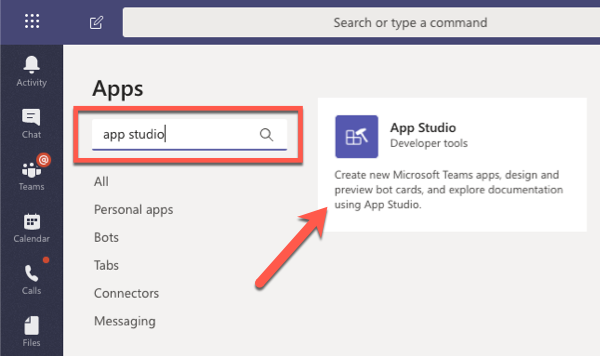
Select the **More added apps** menu item from the left-hand navigation and select the **App Studio** app:



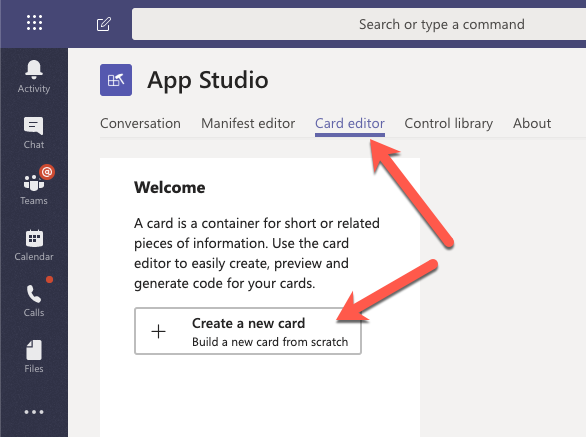
**Tip**

If App Studio is not listed in the **More added apps** dialog as shown above, select **More apps** at the bottom of the dialog.

Search for **App Studio** and select it from the search results to install it:



In App Studio, select the **Card editor** tab and then select the **Create a new card** button.



Select the **Create** button for an **Adaptive Card** in the list of card types you can create in the **Create a new card** dialog.

The App Studio Card editor will show a default card and its live preview. You can use this interface to design your card and see how it will be rendered.

From the **json** tab, replace the contents of the default card with the following JSON:

{

"$schema": "http://adaptivecards.io/schemas/adaptive-card.json",

"type": "AdaptiveCard",

"version": "1.0",

"body": [

{

"type": "Container",

"items": [

{

"type": "TextBlock",

"text": "YouTube Video Selector",

"weight": "bolder",

"size": "extraLarge"

}

]

},

{

"type": "Container",

"items": [

{

"type": "TextBlock",

"text": "Enter the ID of a YouTube video to show in the task module player.",

"wrap": true

},

{

"type": "Input.Text",

"id": "youTubeVideoId",

"value": ""

}

]

}

],

"actions": [

{

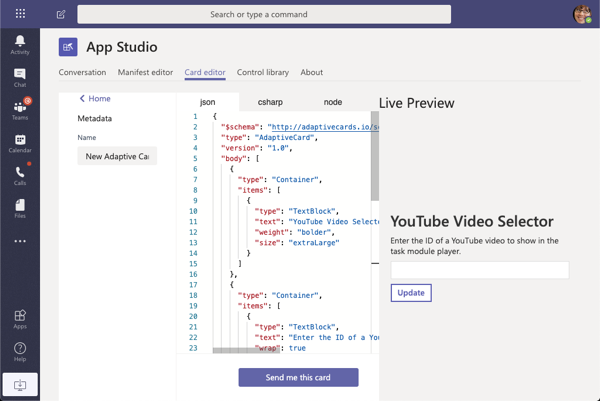
"type": "Action.Submit",

"title": "Update"

}

]

}



This JSON code instructs Microsoft Teams to render a textbox and button. When the button is selected, it will submit the card.

Copy and paste this JSON into a new file, **YouTubeSelectorCard.json**, into the folder in the existing project **./src/app/scripts/youTubePlayer1Tab/**.

## Create a new task module that uses the Adaptive Card

After creating the Adaptive Card, the next step is to create a task module that will display it and handle the submission action.

Within the existing Microsoft Teams app project, locate the file **./src/app/scripts/youTubePlayer1Tab/YouTubePlayer1Tab.tsx** that contains the custom personal tab.

Locate the render() method and add the following code to add a button after one of the existing buttons:

<Button content="Change Video ID (AdaptiveCard)" onClick={this.onChangeVideoAdaptiveCard}></Button>

Next, add the following method to the YouTubePlayer1Tab class:

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

const taskModuleInfo = {

title: "YouTube Video Selector",

width: 350,

height: 250

};

const submitHandler = (err: string, result: any): void => {

};

microsoftTeams.tasks.startTask(taskModuleInfo, submitHandler);

}

The first step is to load the Adaptive Card and set the value of the video ID to display when it loads. Do this by adding the following code to the top of the onChangeVideoAdaptiveCard() method:

// load adaptive card

const adaptiveCard: any = require("./YouTubeSelectorCard.json");

// update card with current video ID

adaptiveCard.body.forEach((container: any) => {

if (container.type === "Container") {

container.items.forEach((item: any) => {

if (item.id && item.id === "youTubeVideoId") {

item.value = this.state.youTubeVideoId;

}

});

}

});

Next, implement the callback. When the Adaptive Card executes the submit action, it will send an object back with all the input objects as properties. Add the following code ot the existing submitHandler() in the onChangeVideoAdaptiveCard() function. This code will update the state with the value of the video ID specified in the Adaptive Card:

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

youTubeVideoId: result.youTubeVideoId

}));

Lastly, add a new card property to the taskModuleInfo object, and set its value to the adaptive card. The resulting taskModuleInfo should look like the following code:

const taskModuleInfo = {

title: "YouTube Video Selector",

card: adaptiveCard,

width: 350,

height: 250

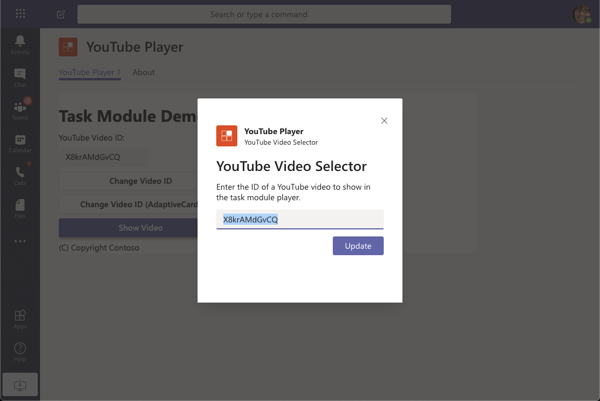
};

## Test the Adaptive Card task module

From the command line, navigate to the root folder for the project and execute the following command:

gulp ngrok-serve

Refresh the Microsoft Teams interface and select the new button **Change Video ID (AdaptiveCard)**. Microsoft Teams will open a task module with the rendered Adaptive Card:



Replace the video ID in the input box and select **Update**. Notice the video ID displayed in the tab is updated to reflect this new value.

## Invoking task modules with deep links

Task modules can be invoked by selecting a button in the Microsoft Teams experience, or using a deep link. Deep links allow you to trigger a task module invocation from outside of teams, or within teams from a conversation.

The format for a deep link is as follows:

https://teams.microsoft.com/l/task/<APP\_ID>?url=<TaskInfo.url>&height=<TaskInfo.height>&width=<TaskInfo.width>&title=<TaskInfo.title>

Consider if you wanted to open a task module from a conversation that would display the following video on an [**Overview of teams and channels**](https://www.youtube.com/watch?v=OjEgeHNKyt4).

As you learned in a previous exercise, the URL to display the video in the player task module would be the following:

https://{{REPLACE\_WITH\_YOUR\_NGROK\_URL}}/youTubePlayer1Tab/player.html?vid=OjEgeHNKyt4

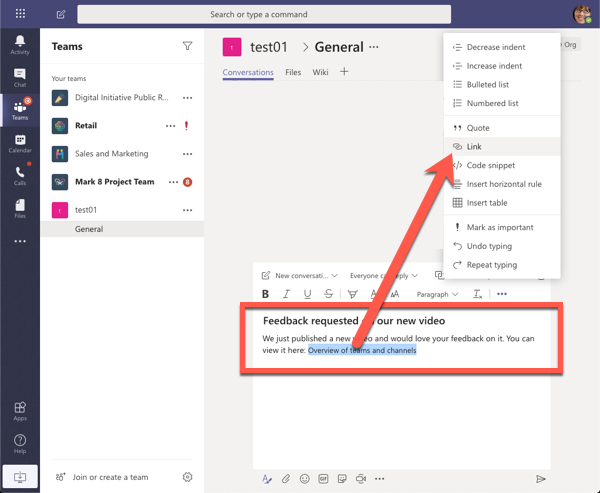
The deep link for to launch the video player task module would be the following (*assuming your custom Microsoft Teams app's ID is 3386faf0-109f-11ea-9799-77a28170bd5d*):

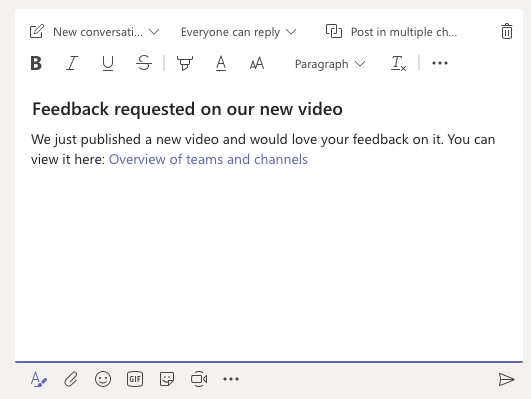
https://{{REPLACE\_WITH\_YOUR\_NGROK\_URL}}/l/task/3386faf0-109f-11ea-9799-77a28170bd5d?url=https://andrewconnell.ngrok.io/youTubePlayer1Tab/player.html?vid=OjEgeHNKyt4&height=700&width=1000&title=YouTube%20Player:%20Overview%20of%20teams%20and%20channels

In Microsoft teams, go to a channel, select the **Conversations** tab, and select the **Format** button in the message dialog:



Enter a message to post to the channel. Select some of the text and use the **Link** feature to add the deep link to the message:





Now, select the link to see the task module open without having to trigger it from the custom tab.

## Summary

In this exercise, you learned how to use adaptive cards in a custom task module in a custom Microsoft Teams app. You also learned how to invoke task modules from anywhere within Microsoft Teams using deep links.

# Exercise 3- Using task modules with bots

In this exercise, you’ll learn how to use task modules with bots in Microsoft Teams.

The first step will be to create a bot and add it to our existing Microsoft Teams app. You'll then extend the bot to support the existing task modules to display a video and change the selected video.

## Register a new bot in Microsoft Azure

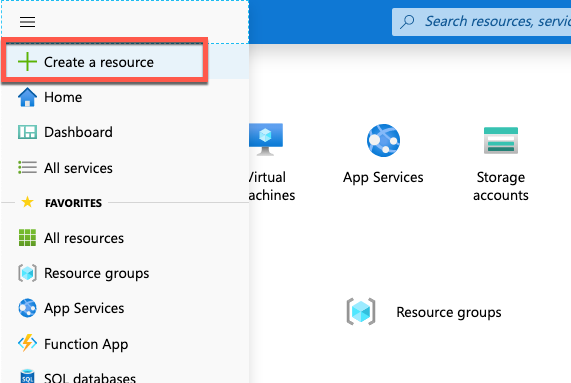
The first step is to create a new Microsoft Teams bot. Adding a bot to the Teams app involves two steps:

1. Register the bot with Microsoft Azure's Bot Framework
2. Add a bot to the project codebase

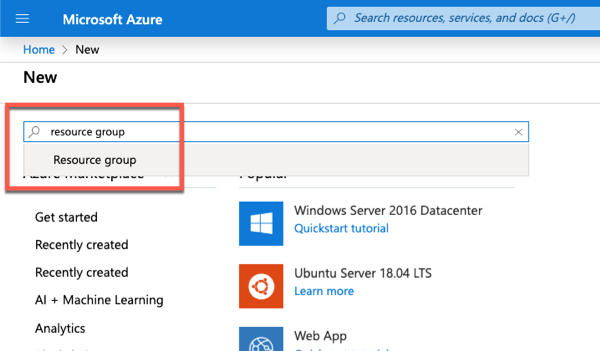
## Register the bot with Microsoft Azure's Bot Framework

Open a browser and navigate to the [Azure portal](https://portal.azure.com/). Sign in using a **Work or School Account** that has rights to create resources in your Azure subscription.

Select **Create a resource** in the left-hand navigation:

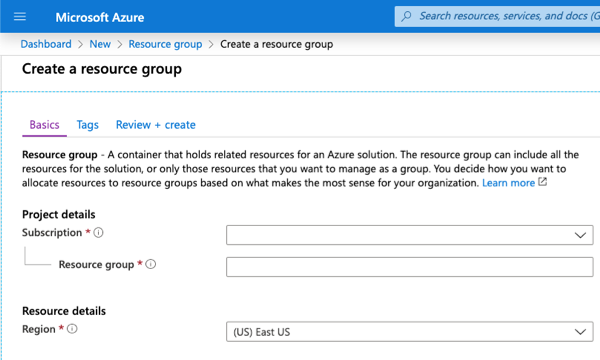


Enter **resource group** in the **Search the marketplace** input box, and select **Resource group**.



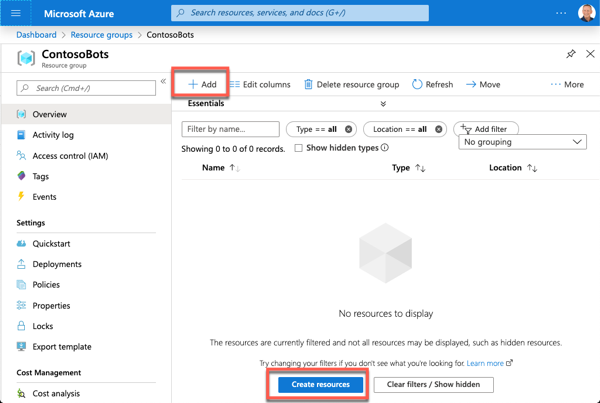
On the **Resource Group** page, select the **Create** button to create a new resource group.

Select a valid subscription, enter a name for the resource group, and select the wanted region. *None of these choices will impact the bot registration and are up to you.*

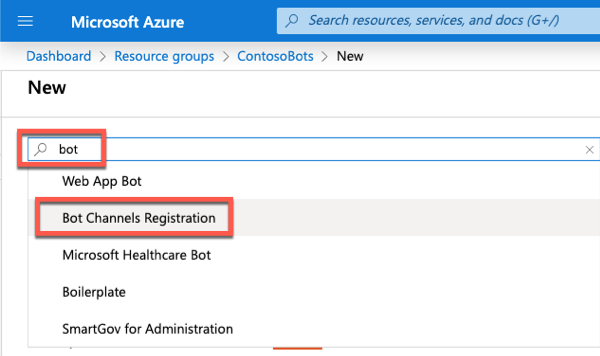


Complete the wizard to create the resource group. Once Azure has completed the resource group creation process, navigate to the resource group.

From the resource group, select the **Add** or **Create resources** button.



Enter **bot** in the **Search the marketplace** input box, and select **Bot Channels Registration** from the list of resources returned. Then select **Create** on the next page to start the process of registering a new bot resource:



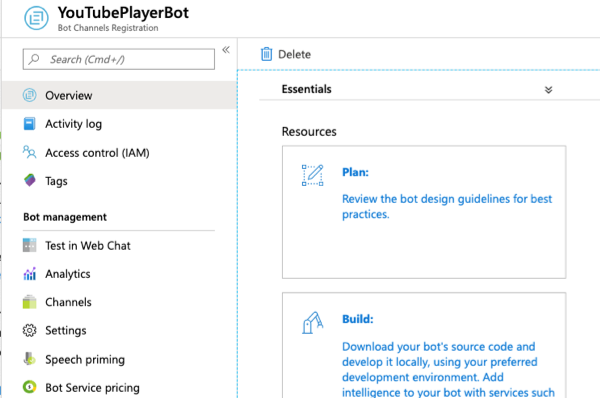
In the **Bot Channels Registration** blade, enter the following values and then select **Create**:

* **Bot handle**: *Enter a globally unique name for the bot*
* **Subscription**: *Select the subscription you selected previously when creating the resource group*
* **Resource group**: *Select the resource group you created previously*
* **Location**: *Select your preferred Azure region*
* **Pricing tier**: *Select a preferred pricing tier; the F0 tier is free*
* **Messaging endpoint**: [https://REPLACE\_THIS.ngrok.io/api/messages](https://replace_this.ngrok.io/api/messages)

The bot registration needs to know the endpoint of the web service where the bot is implemented. This will change each time you start the ngrok utility used in previous exercises.

* **Application Insights**: Off
* **Microsoft App ID and password**: Auto create App ID and password

Azure will start to provision the new resource. This will take a moment or two. Once its finished, navigate to the bot resource in the resource group.

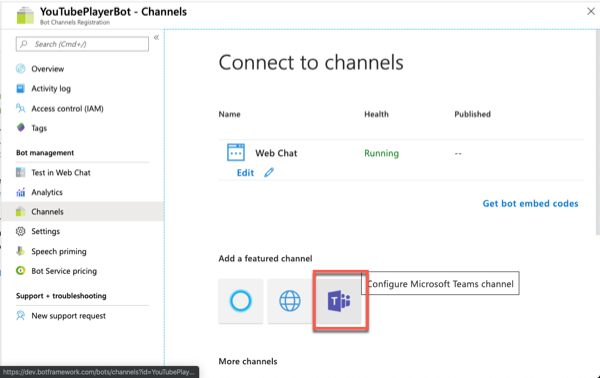


## Enable the Microsoft Teams channel for the bot

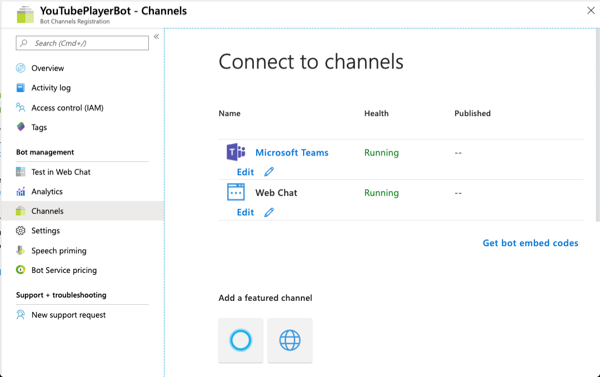
In order for the bot to interact with Microsoft Teams, you must enable the Teams channel.

From the bot resource in Azure, select **Channels** in the left-hand navigation.

On the **Connect to channels** pane, select the Microsoft Teams channel, then select **Save** to confirm the action.



Once this process is complete, you should see both the **Web Chat** and **Microsoft Teams** listed in your enabled channels:



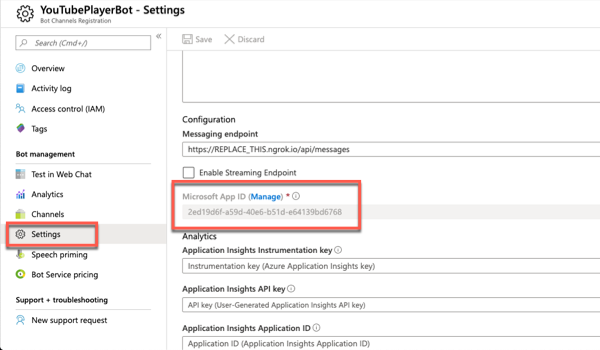
## Retrieve the bot app id and password

When Azure created the bot, it also registered a new Azure AD app for the bot. You need to generate this new bot app a secret and copy the app's credentials.

Select **Settings** from the left-hand navigation. Scroll down to the **Microsoft App ID** section.

Copy the ID of the bot as you'll need it later.

Select **Manage** to navigate to the Azure AD app blade:

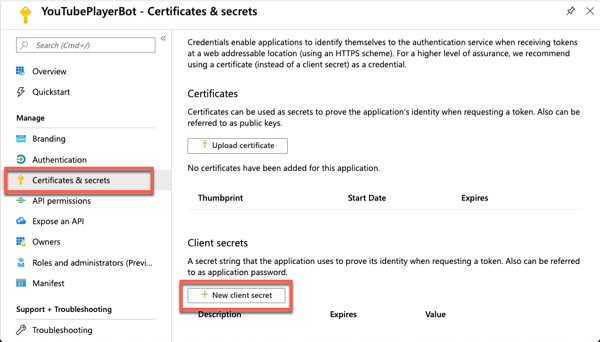


## Create a client secret for the app

In order for the daemon app to run without user involvement, it will sign in to Azure AD with an application ID and either a certificate or secret. In this exercise, you'll use a secret.

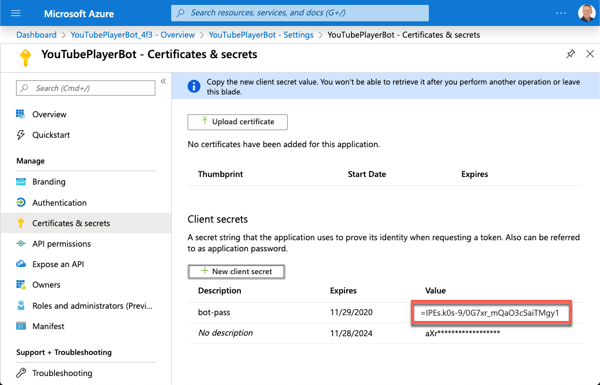
Select **Certificates & secrets** from the left-hand navigation panel.

Select the **New client secret** button:



When prompted, give the secret a description and select one of the expiration duration options provided and select **Add**. *What you enter and select doesn't matter for the exercise.*

The **Certificate & Secrets** page will display the new secret. It's important you copy this value as it's only shown this one time; if you leave the page and come back, it will only show as a masked value.



Copy the value of the secret as you'll need it later.

## Update the Microsoft Teams app project with the app details

Open the project you created in previous exercises in Visual Studio Code.

Locate the **./.env** file. This is an environment variable helper file used during development.

Locate the properties **MICROSOFT\_APP\_ID** and **MICROSOFT\_APP\_PASSWORD** and copy the values of the bot's application ID and secret to these values.

At this point, the bot has been registered with the Azure Bot service. The next step is to implement the bot and add it to Microsoft Teams.

## Add a bot to the project

The first step is to add the bot to the app's manifest file. Locate and open the file **./src/manifest/manifest.json**.

Locate the id property and change its value to the same ID as the Azure AD application created for the bot.

Locate the section "bots": [], in the **manifest.json** file. Replace this empty collection with the following. Ensure you use the same Azure AD app ID for the botId in the following code:

"bots": [

{

"botId": "618ddfa7-7990-4bba-ab95-fc6e76d2c019",

"scopes": [

"personal"

],

"supportsFiles": false,

"isNotificationOnly": false,

"commandLists": [

{

"scopes": [ "personal" ],

"commands": [

{

"title": "MentionMe",

"description": "Sends message with @mention of the sender"

}

]

}

]

}

],

This registers a new bot that only exists in the personal scope and has a single command **MentionMe**. You'll use this to ensure the bot is working.

The next step is to add the code to implement the bot.

## Code the bot

**Important**

At the time of publication of this module, there are plans to update the Yeoman generator for Microsoft Teams to use the Bot Framework v4 SDK. However, at the time of publication of this module, the default project uses an older SDK.

Therefore, the following steps that remove & upgrade the **botbuilder** package may or may not be necessary as the Yeoman generator for Microsoft Teams may have been updated. Review each of the instructions in this section and compare the results with your project to determine if they are necessary.

Remove the older bot SDK and related packages. Then, upgrade the bot builder project, execute the following two commands in the command line from the root folder of the project:

npm uninstall botbuilder-dialogs botbuilder-teams botframework-config

npm install botbuilder@4.6.2

Create a new folder **learningTeamsBot** in the **./src/app** folder.

Create a new file **teamsLearningBot.ts** in this new folder **./src/app/learningTeamsBot/teamsLearningBot.ts**

Add the following code to the **teamsLearningBot.ts** file:

import {

ActionTypes,

CardFactory, MessageFactory,

TeamsActivityHandler, TaskModuleTaskInfo,

TurnContext, TaskModuleRequest, TaskModuleResponse

} from "botbuilder";

import \* as Util from "util";

const TextEncoder = Util.TextEncoder;

export class TeamsLearningBot extends TeamsActivityHandler {

constructor() {

super();

// create handlers

this.onMessage(async (context, next) => {

switch (context.activity.text.trim().toLowerCase()) {

case "mentionme":

await this.mentionActivity(context);

break;

default:

const card = CardFactory.heroCard("Learn Microsoft Teams", undefined, [ ]);

await context.sendActivity({ attachments: [card] });

break;

}

await next();

});

}

private async mentionActivity(context: TurnContext) {

const mention = {

mentioned: context.activity.from,

text: `<at>${new TextEncoder().encode(context.activity.from.name)}</at>`,

type: 'mention'

};

const replyActivity = MessageFactory.text(`Hi ${mention.text}`);

replyActivity.entities = [mention];

await context.sendActivity(replyActivity);

}

}

## Update the webserver to load the bot

The next step is to get the bot working in the project.

Locate and open the file **./src/app/server.ts**.

Add the following import statements to the end of the existing import statements:

import { BotFrameworkAdapter } from "botbuilder";

import { TeamsLearningBot } from "./learningTeamsBot/teamsLearningBot";

Locate the following code at the end of the file:

http.createServer(express).listen(port, () => {

log(`Server running on ${port}`);

});

This code creates the server and starts listening on a port. You need access to the server to load the bot. Update the above code to separate the creation of the web server and listening on a specific port to the following:

const server = http.createServer(express);

server.listen(port, () => {

log(`Server running on ${port}`);

});

Add the following code to the end of the **server.ts** to create a bot framework adapter. This code uses the environment variables defined in the **./.env** file to set the Azure AD app ID and secret of the bot:

const botAdapter = new BotFrameworkAdapter({

appId: process.env.MICROSOFT\_APP\_ID,

appPassword: process.env.MICROSOFT\_APP\_PASSWORD

});

Add the following error handler to the end of the **server.ts** file.

botAdapter.onTurnError = async (context, error) => {

console.error(`\n [bot.onTurnError] unhandled error: ${error}`);

await context.sendTraceActivity("OnTurnError Trace", `${error}`, "https://www.botframework.com/schemas/error", "TurnError");

await context.sendActivity("bot error");

};

Load the bot and register a new endpoint, **/api/messages**. When HTTP POSTS are received on this endpoint, they're routed into the bot adapter:

const bot = new TeamsLearningBot();

express.post("/api/messages", (request, response) => {

botAdapter.processActivity(request, response, async (context) => {

await bot.run(context);

});

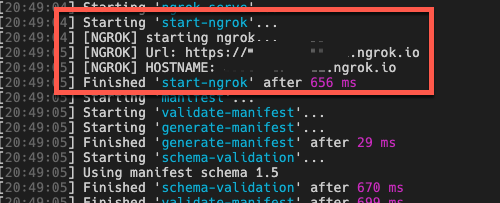
});

## Test the bot in Microsoft Teams

From the command line, navigate to the root folder for the project and execute the following command:

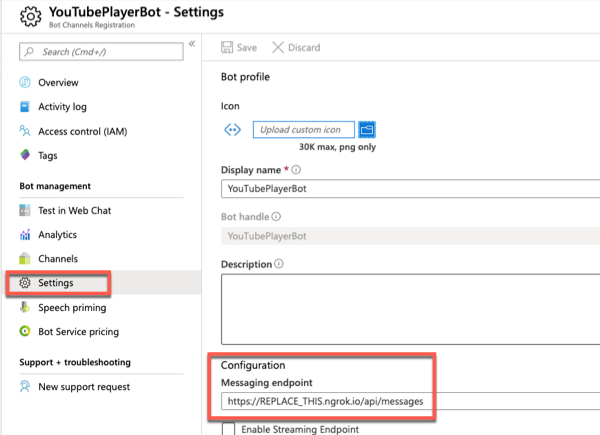
gulp ngrok-serve

In the console, locate the dynamic URL created by ngrok:



Go back to the bot registration in the Azure portal.

Select **Settings** from the left-hand navigation. Update the **Messaging endpoint** of the bot to match the ngrok URL. The resulting URL should be **https://{{REPLACE\_THIS}}.ngrok.io/api/messages**.



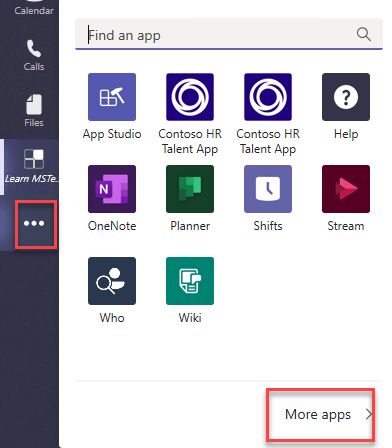
Locate the **./.env** file in the project. Update the property **HOSTNAME** to point to the same ngrok URL: **{{REPLACE\_THIS}}.ngrok.io**.

Now you're ready to add the bot to Microsoft Teams. In the browser, navigate to [**https://teams.microsoft.com**](https://teams.microsoft.com/) and sign in with the credentials of a Work and School account.

**Note**

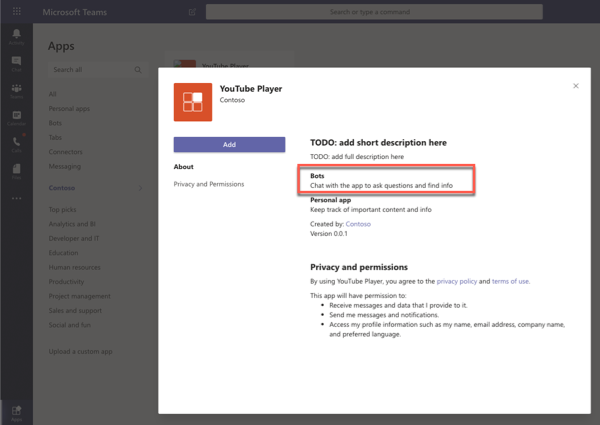
Microsoft Teams is available for use as a web client, desktop client and a mobile client. In this module, we will use the web client but any of the clients can be used.

Using the app br navigation menu, select the **More added apps** button. Then select **More apps** followed by **Upload a custom app** > **Upload for me or my teams**.



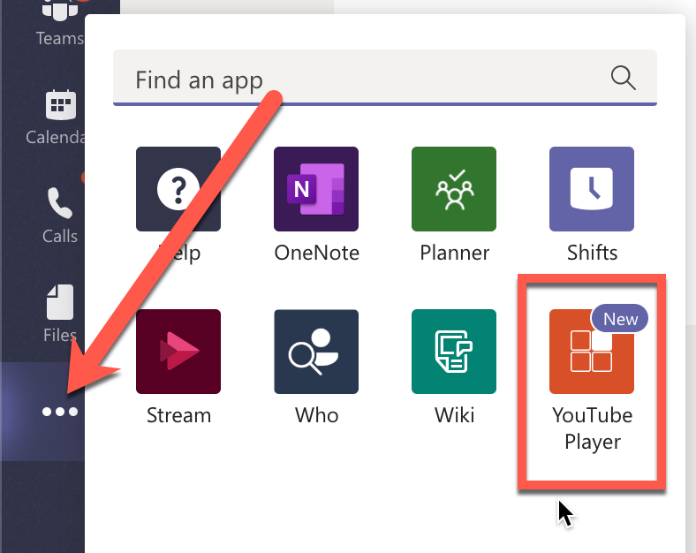
In the file dialog that appears, select the Microsoft Teams package in your project. This app package is a ZIP file that can be found in the project's **./package** folder.

Once the package is uploaded, Microsoft Teams will display a summary of the app. Here you can see some "todo" items to address. *None of these "todo" items are important to this exercise, so you'll leave them as is.*

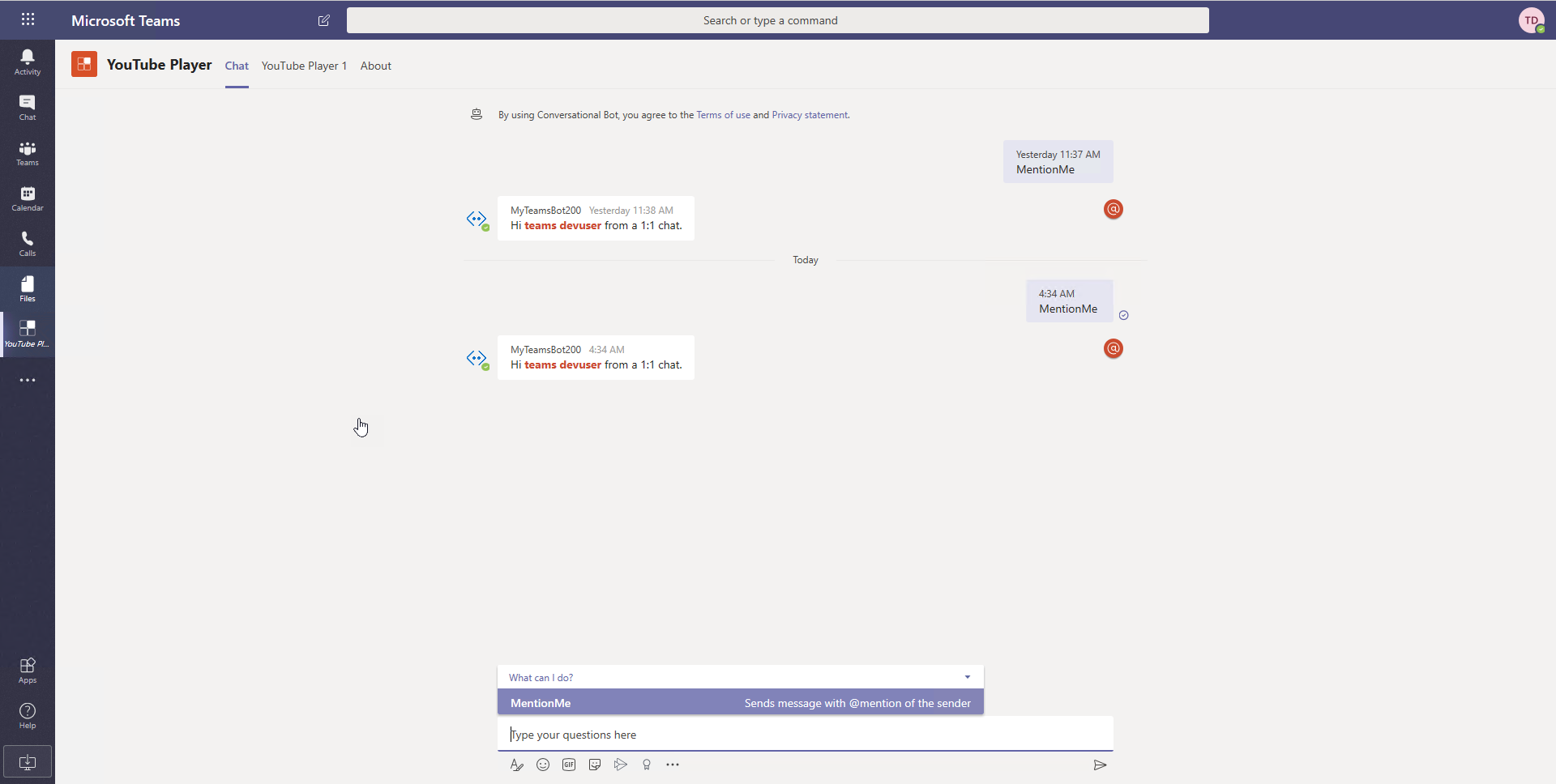


Notice Microsoft Teams recognizes there's a bot associated with the app.

Select the **Add** button to install the app, adding a new personal tab to your **More added apps** dialog:

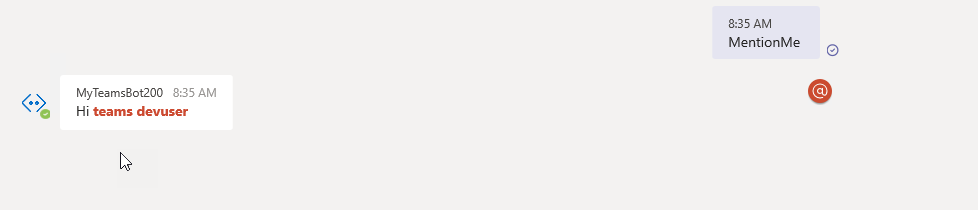


Select the app to navigate to the tab:

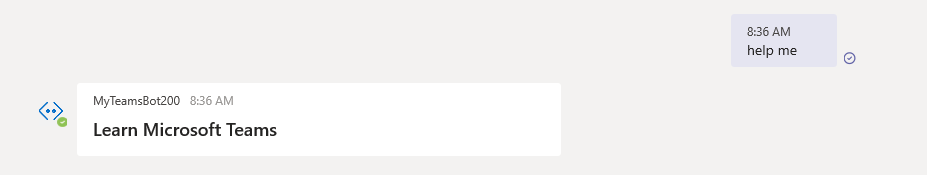


Notice there's a new **Chat** tab. Focus on the message box at the bottom of the user interface. Microsoft Teams is displaying the things this bot can do, including the **MentionMe** action. Let's try that out.

Enter MentionMe following in the message box and press enter.



Next, enter some random text. Because the bot is looking for specific text, it will display the default message:



At this point, the bot is working. Move onto the next section to add task module support.

## Add the video player task module

In this section, you'll add a task module to the bot. First, let's update the Hero card to add a button the user can select.

In the bot file, **./src/app/learningTeamsBot/teamsLearningBot.ts**, locate the class constructor. Locate Hero card statement in the switch statement's default path in the onMessage() handler:

const card = CardFactory.heroCard("Learn Microsoft Teams", undefined, [ ]);

Update this statement to add a new action to the card. The type of this action is special: invoke.

const card = CardFactory.heroCard("Learn Microsoft Teams", undefined, [

{

type: "invoke",

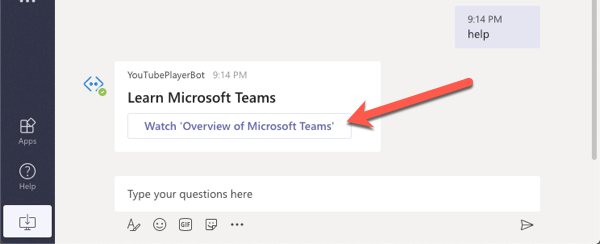
title: "Watch 'Overview of Microsoft Teams'",

value: { type: "task/fetch", taskModule: "player", videoId: "X8krAMdGvCQ" }

}

]);

The project's **ngrok-serve** task will detect the code change, rebuild & restart the web server. After a moment, enter another random string in the **Conversations** tab for the bot and notice the new card:



The bot framework is looking for messages of type invoke with their value.type property set to task/fetch. For each one it finds, it passes it into a handler handleTeamsTaskModuleFetch(). To handle what happens when you select the action, implement the method.

Add the following code to the TeamsLearningBot class:

protected handleTeamsTaskModuleFetch(context: TurnContext, request: TaskModuleRequest): Promise<TaskModuleResponse> {

let response: TaskModuleResponse;

switch (request.data.taskModule) {

case "player":

response = <TaskModuleResponse>{

task: {

type: "continue",

value: <TaskModuleTaskInfo>{

title: "YouTube Player",

url: `https://${process.env.HOSTNAME}/youTubePlayer1Tab/player.html?vid=${request.data.videoId}`,

width: 1000,

height: 700

}

}

};

break;

default:

response = <TaskModuleResponse>{

task: {

type: "continue",

value: <TaskModuleTaskInfo>{

title: "YouTube Player",

url: `https://${process.env.HOSTNAME}/youTubePlayer1Tab/player.html?vid=X8krAMdGvCQ&default=1`,

width: 1000,

height: 700

}

}

};

break;

};

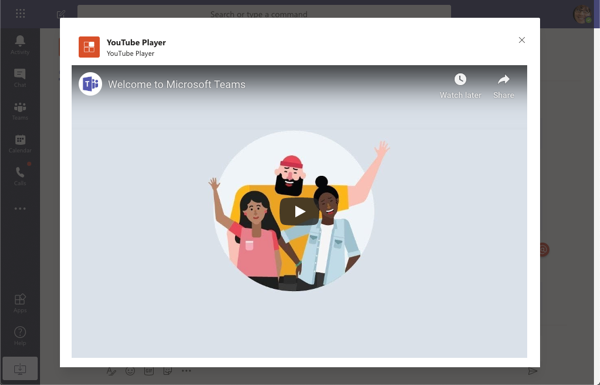
console.log("handleTeamsTaskModuleFetch() response", response);

return Promise.resolve(response);

}

## Test the video player task module

After saving the page, let the project rebuild itself and restart the web server. Once that's happened, select the **Watch Overview of Microsoft Teams** button in the Hero card. Notice the task module from the previous exercise is displayed:



Create a few more buttons to the Hero card that reference different videos by duplicating the existing button. Make sure to create one that has the value.taskModule property set to something other than player so the handler takes the default path:

const card = CardFactory.heroCard("Learn Microsoft Teams", undefined, [

{

type: "invoke",

title: "Watch 'Overview of Microsoft Teams'",

value: { type: "task/fetch", taskModule: "player", videoId: "X8krAMdGvCQ" }

},

{

type: "invoke",

title: "Watch 'Go-to guide for team owners'",

value: { type: "task/fetch", taskModule: "player", videoId: "7XcDSuw6NR4" }

},

{

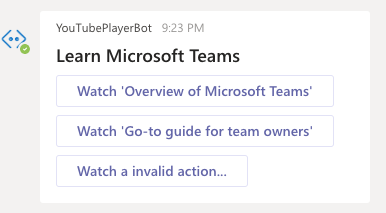
type: "invoke",

title: "Watch a invalid action...",

value: { type: "task/fetch", taskModule: "something", videoId: "helloworld" }

}

]);



The next step is to add a task module that submits data.

## Add the video selector Adaptive Card task module

In this section, you'll add an action to the Hero card that displays a task module using an Adaptive Card. This task module will submit data back to the bot that will use the value to display the specified video in the player task module.

In the bot file, **./src/app/learningTeamsBot/teamsLearningBot.ts**, locate the code in the teamsLearningBot class constructor that created the Hero card. Add another action to the end, but notice this action specifies a different value.taskModule property:

{

type: "invoke",

title: "Watch Specific Video",

value: { type: "task/fetch", taskModule: "selector", videoId: "QHPBw7F4OL4" }

}

You'll chain two task modules together. The first will display the Adaptive Card selector. Within the handleTaskModuleFetch() method, add the following case statement to the existing switch statement:

case "selector":

response = <TaskModuleResponse>{

task: {

type: "continue",

value: <TaskModuleTaskInfo>{

title: "YouTube Video Selector",

card: this.getSelectorAdaptiveCard(request.data.videoId),

width: 350,

height: 250

}

}

};

break;

Add the following method to the TeamsLearningBot class. This will create an Adaptive Card with an input control. This is the programmatic way of creating the same adaptive card from a previous exercise.

private getSelectorAdaptiveCard(defaultVideoId: string = "") {

return CardFactory.adaptiveCard({

type: "AdaptiveCard",

version: "1.0",

body: [

{

type: "Container",

items: [

{

type: "TextBlock",

text: "YouTube Video Selector",

weight: "bolder",

size: "extraLarge"

}

]

},

{

type: "Container",

items: [

{

type: "TextBlock",

text: "Enter the ID of a YouTube video to show in the task module player.",

wrap: true

},

{

type: "Input.Text",

id: "youTubeVideoId",

value: defaultVideoId

}

]

}

],

actions: [

{

type: "Action.Submit",

title: "Update"

}

]

});

}

Next, add the handler for the Adaptive Card handler method. Similar to the task/fetch, we need to handle a submit, or task/submit. Do this by adding the following method to the TeamsLearningBot class:

protected handleTeamsTaskModuleSubmit(context: TurnContext, request: TaskModuleRequest): Promise<TaskModuleResponse> {

const response: TaskModuleResponse = <TaskModuleResponse>{

task: {

type: "continue",

value: <TaskModuleTaskInfo>{

title: "YouTube Player",

url: `https://${process.env.HOSTNAME}/youTubePlayer1Tab/player.html?vid=${request.data.youTubeVideoId}`,

width: 1000,

height: 700

}

}

};

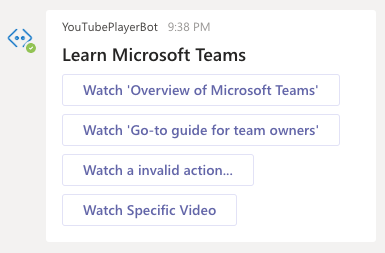
return Promise.resolve(response);

}

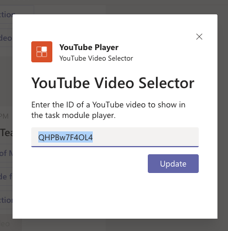
This handler will fetch the submitted video ID from the Adaptive Card and use it to present the player task module.

## Test the video selector Adaptive Card task module

After saving your changes and giving the project a moment to rebuild and restart the webserver, enter another message into the bot **Chat** tab:



Select the **Watch Specific Video** button. When the Adaptive Card task module loads, either accept the default value or enter the ID of a video on YouTube.



Select the **Update** button. Notice the Adaptive Card task module disappears and the player task module loads with the specified video.

Stop the local web server by pressing CTRL+C in the console to stop the running process.

## Summary

In this exercise, you learned how to use task modules with bots in Microsoft Teams. After creating and adding it to our existing Microsoft Teams app, then extend the bot to support the existing task modules to display a video and change the selected video.