Microsoft Office 365 Outlook - Web Add-in - Consulting Practice

Cognitive Convergence is Subject Matter Expert in Office 365, Dynamics 365, SharePoint, Project Server, Power Platform: Power Apps-Power BI-Power Automate-Power Virtual Agents.

Our Microsoft Office 365 Consulting, add-in Development, Customization, Integration services and solutions, can help companies maximize business performance, overcoming market challenges, achieving profitability and providing best customer service.

CONTENTS

OBJECTIVE

This case study is a brief introduction about the core concept of MS-Outlook Web Add-in, its fundamentals, building architecture & its development tracks.

MS OUTLOOK ADD-IN

Outlook add-ins are integrations built by third parties into Outlook by using our web-based platform. Outlook add-ins have three key aspects:

- The same add-in and business logic work across desktop (Outlook on Windows and Mac), web (Microsoft 365 and Outlook.com), and mobile.
- Outlook add-ins consist of a manifest, which describes how the add-in integrates into Outlook (for example, a button or a task pane), and JavaScript/HTML code, which makes up the UI and business logic of the add-in.
- Outlook add-ins can be acquired from AppSource or sideloaded by end-users or administrators.

Outlook add-ins are different from COM or VSTO add-ins, which are older integrations specific to Outlook running on Windows. Unlike COM add-ins, Outlook add-ins don't have any code physically installed on the user's device or Outlook client. For an Outlook add-in, Outlook reads the manifest and hooks up the specified controls in the UI and then loads the JavaScript and HTML. The web components all run in the context of a browser in a sandbox.

The Outlook items that support add-ins include email messages, meeting requests, responses and cancellations, and appointments. Each Outlook add-in defines the context in which it is available, including the types of items and if the user is reading or composing an item.

COMPONENTS OF AN OFFICE ADD-IN

An Office Add-in includes two basic components: an XML manifest file, and a user's web application. The manifest defines various settings, including how your add-in integrates with Office clients. User's web application needs to be hosted on a web server, or web hosting service, such as Microsoft Azure.

Manifest

The manifest is an XML file that specifies settings and capabilities of the add-in, such as:

- The add-in's display name, description, ID, version, and default locale.
- How the add-in integrates with Office.
- The permission level and data access requirements for the add-in.

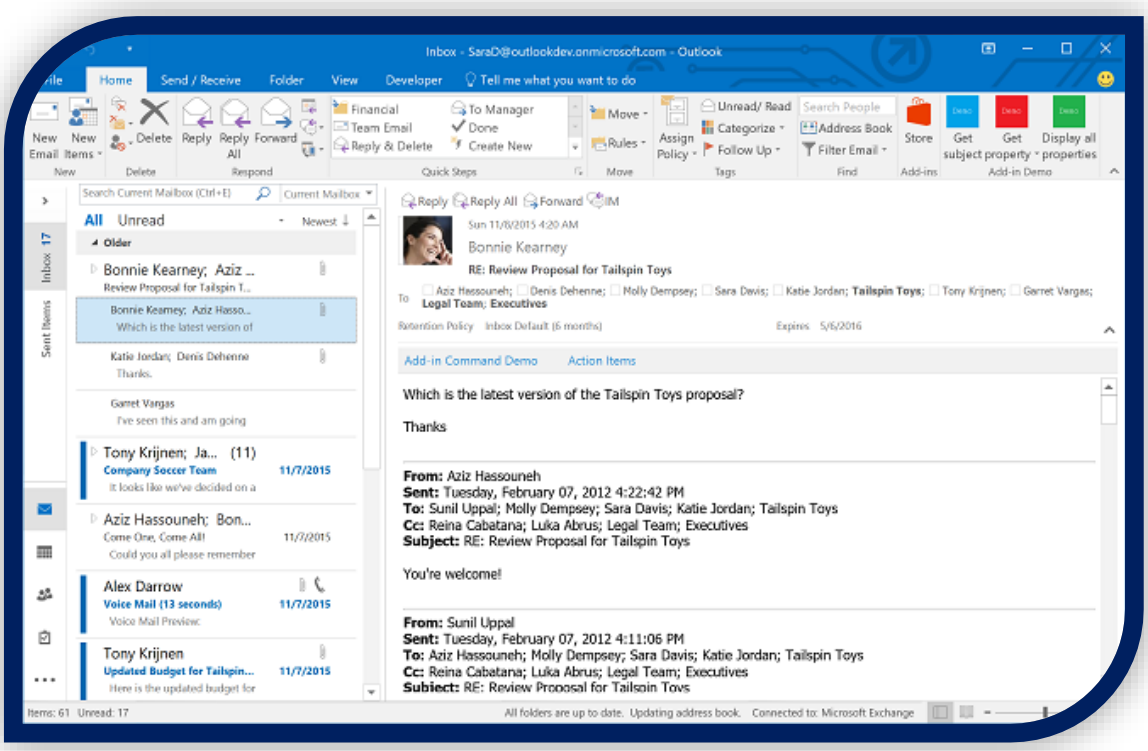
Web app

The most basic Office Add-in consists of a static HTML page that is displayed inside an Office application, but that doesn't interact with either the Office document or any other Internet resource. However, to create an experience that interacts with Office documents or allows the user to interact with online resources from an Office host application, the user can use any technologies, both client and server-side, that the hosting provider supports (such as ASP.NET, PHP, or Node.js). To interact with Office clients and documents, the user uses the Office.js JavaScript APIs.

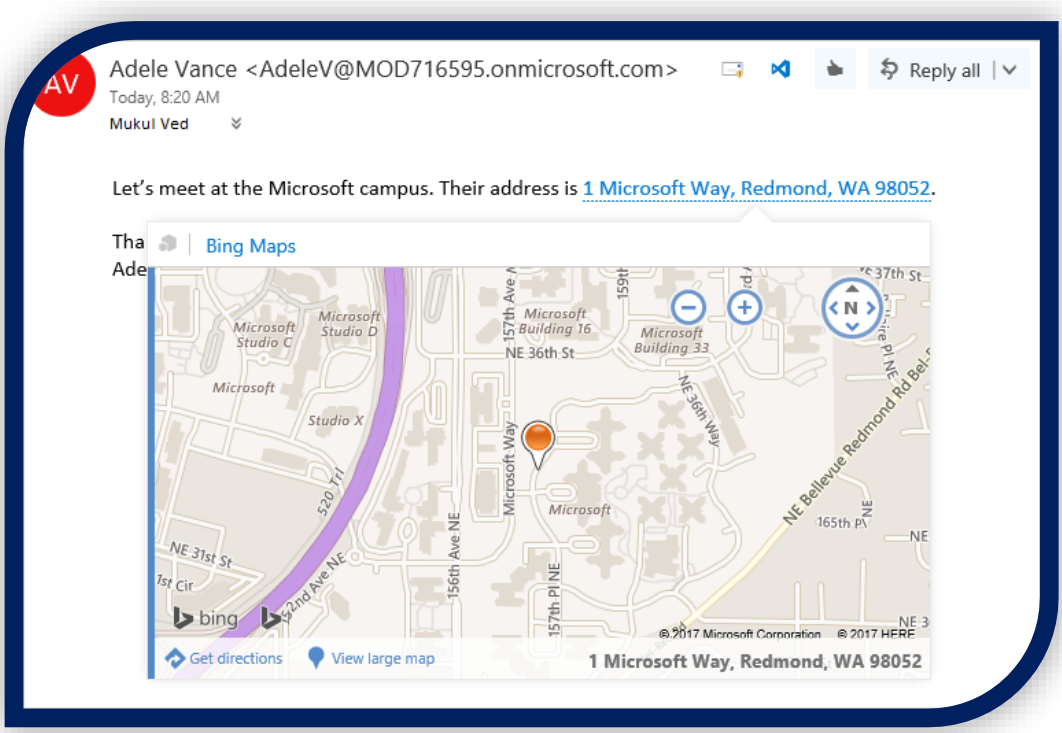
EXTENSION POINTS

Extension points are the ways that add-ins integrate with Outlook. The following are the ways this can be done:

- Add-ins can declare buttons that appear in command surfaces across messages and appointments.



- Add-ins can link off regular expression matches or detected entities in messages and appointments



MAILBOX ITEMS AVAILABLE TO ADD-INS

Outlook add-ins are available on messages or appointments while composing or reading, but not other item types. Outlook does not activate add-ins if the current message item, in a compose or read form, is one of the following:

Protected by Information Rights Management (IRM) or encrypted in other ways for protection. A digitally signed message is an example since digital signing relies on one of these mechanisms.

- A delivery report or notification that has the message class IPM.Report.*, including delivery and Non-Delivery Report (NDR) reports, and read, non-read, and delay notifications.
- A draft (does not have a sender assigned to it), or in the Outlook Drafts folder.
- A .msg or .eml file which is an attachment to another message.
- A .msg or .eml file opened from the file system.
- In a shared mailbox, in another user's mailbox, in an archive mailbox, or a public folder.
- Using a custom form.

In general, Outlook can activate add-ins in reading form for items in the Sent Items folder, except for add-ins that activate based on string matches of well-known entities.

SUPPORTED HOSTS

Outlook add-ins are supported in Outlook 2013 or later on Windows, Outlook 2016 or later on Mac, Outlook on the web for Exchange 2013 on-premises and the later versions, Outlook on iOS, Outlook on Android, and Outlook on the web and Outlook.com. Not all of the newest features are supported in all clients at the same time.

OUTLOOK WEB ADD-IN OBJECT MODEL - CODE LEVEL ARCHITECTURE

Office.Appointment	The subclass of Item dealing with appointments. **Important**: This is an internal Outlook object, not directly exposed through existing interfaces. You should treat this as a mode of Office.context.mailbox.item. Refer to the Object Model page for more information. Child interfaces: • AppointmentCompose • AppointmentRead
Office.AppointmentCompose	The appointment organizer mode of Office.context.mailbox.item. **Important**: This is an internal Outlook object, not directly exposed through existing interfaces. You should treat this as a mode of Office.context.mailbox.item. Refer to the Object Model page for more information. Parent interfaces: • ItemCompose • Appointment
Office.AppointmentForm	The AppointmentForm object is used to access the currently selected appointment.
Office.AppointmentRead	The appointment attendee mode of Office.context.mailbox.item. **Important**: This is an internal Outlook object, not directly exposed through existing interfaces. You should treat this as a mode of Office.context.mailbox.item. Refer to the Object Model page for more information. Parent interfaces: • ItemRead • Appointment
Office.AppointmentTimeChangedEventArgs	Provides the current dates and times of the appointment that raised the Office.EventType.AppointmentTimeChanged event.
Office.AttachmentContent	Represents the content of an attachment on a message or appointment item.
Office.AttachmentDetails	Represents an attachment on an item from the server. Read mode only. An array of AttachmentDetails objects is returned as the attachments property of an appointment or message item.
Office.AttachmentDetailsCompose	Represents an attachment on an item. Compose mode only. An array of AttachmentDetailsCompose objects is returned as the attachments property of an appointment or message item.
Office.AttachmentsChangedEventArgs	Provides information about the attachments that raised the Office.EventType.AttachmentsChanged event.

Office.Body	The body object provides methods for adding and updating the content of the message or appointment. It is returned in the body property of the selected item.
Office.Categories	Represents the categories on an item. In Outlook, a user can tag messages and appointments by using a category to color-code them. The user defines categories in a master list in their mailbox. They can then apply one or more categories to an item. **Important**: In Outlook on the web, you can't use the API to manage categories applied to a message in Compose mode.
Office.CategoryDetails	Represents a category's details like name and associated color.
Office.CoercionTypeOptions	Provides an option for the data format.
Office.Contact	Represents the details about a contact (similar to what's on a physical contact or business card) extracted from the item's body. Read mode only. The list of contacts extracted from the body of an email message or appointment is returned in the contacts property of the Entities object returned by the getEntities or getEntitiesByType method of the current item.
Office.CustomProperties	The CustomProperties object represents custom properties that are specific to a particular item and specific to a mail add-in for Outlook. For example, there might be a need for a mail add-in to save some data that is specific to the current email message that activated the add-in. If the user revisits the same message in the future and activates the mail add-in again, the add-in will be able to retrieve the data that had been saved as custom properties. **Important**: The maximum length of a CustomProperties JSON object is 2500 characters. Because Outlook on Mac doesn't cache custom properties, if the user's network goes down, mail add-ins cannot access their custom properties.
Office.Diagnostics	Provides diagnostic information to an Outlook add-in.
Office.EmailAddressDetails	Provides the email properties of the sender or specified recipients of an email message or appointment.
Office.EmailUser	Represents an email account on an Exchange Server.
Office.EnhancedLocation	Represents the set of locations on an appointment.
Office.EnhancedLocationsChangedEventArgs	Provides the current enhanced locations when the Office.EventType.EnhancedLocationsChanged event is raised.
Office.Entities	Represents a collection of entities found in an email message or appointment. Read mode only. The Entities object is a container for the entity arrays returned by the getEntities and getEntitiesByType methods when the item (either an email message or an appointment) contains one or more entities that have been found by the server. You can use these entities in your code to provide additional context information to the viewer, such as a map to an address found in the item, or to open a dialer for a phone number found in the item. If no entities of the type specified in the property are present in the item, the property associated with that entity is null. For example, if a message contains a street address and a phone number, the addresses property and phoneNumbers property would contain information, and the other properties would be null. To be recognized as an address, the string must contain a United States postal address that has at least a subset of the elements of a street number, street name, city, state, and zip code. To be recognized as a phone number, the string must contain a North American phone number format. Entity recognition relies on natural language recognition that is based on machine learning of large amounts of data. The recognition of an entity is non-

	deterministic and success sometimes relies on the particular context in the item. When the property arrays are returned by the getEntitiesByType method, only the property for the specified entity contains data; all other properties are null.
Office.From	Provides a method to get the from the value of a message in an Outlook add-in.
Office.InternetHeaders	The InternetHeaders object represents custom internet headers that are preserved after the message item leaves Exchange and is converted to a MIME message. These headers are stored as x-headers in the MIME message. Internet headers are stored as key/value pairs on a per-item basis. **Note**: This object is intended for the users to set and get the custom headers on a message item.
Office.IsAllDayEvent	Provides methods to get and set the all-day event status of a meeting in an Outlook add-in.
Office.Item	The item namespace is used to access the currently selected message, meeting request, or appointment. You can determine the type of the item by using the itemType property.
Office.ItemCompose	The compose mode of Office.context.mailbox.item. **Important**: This is an internal Outlook object, not directly exposed through existing interfaces. You should treat this as a mode of Office.context.mailbox.item Child interfaces: • AppointmentCompose • MessageCompose
Office.ItemRead	The read mode of Office.context.mailbox.item. **Important**: This is an internal Outlook object, not directly exposed through existing interfaces. You should treat this as a mode of Office.context.mailbox.item. Refer to the Object Model page for more information. Child interfaces: • AppointmentRead • MessageRead
Office.LocalClientTime	Represents a date and time in the local client's time zone. Read mode only.
Office.Location	Provides methods to get and set the location of a meeting in an Outlook add-in.
Office.LocationDetails	Represents a location. Read-only.
Office.LocationIdentifier	Represents the ID of a location.
Office.Mailbox	Provides access to the Microsoft Outlook add-in object model. Key properties: • diagnostics: Provides diagnostic information to an Outlook add-in. • item: Provides methods and properties for accessing a message or appointment in an Outlook add-in. • userProfile: Provides information about the user in an Outlook add-in.
Office.MasterCategories	Represents the categories master list on the mailbox. In Outlook, a user can tag messages and appointments by using a category to color-code them. The user defines categories in a master list in their mailbox.

	They can then apply one or more categories to an item. **Important**: In delegate or shared scenarios, the delegate can get the categories in the master list but can't add or remove categories.
Office.MeetingSuggestion	Represents a suggested meeting found in an item. Read mode only. The list of meetings suggested in an email message is returned in the meetingSuggestions property of the Entities object that is returned when the getEntities or getEntitiesByType method is called on the active item. The start and end values are string representations of a Date object that contains the date and time at which the suggested meeting is to begin and end. The values are in the default time zone specified for the current user.
Office.Message	A subclass of Item for messages. **Important**: This is an internal Outlook object, not directly exposed through existing interfaces. You should treat this as a mode of Office.context.mailbox.item.
Office.MessageCompose	The message composes mode of Office.context.mailbox.item. **Important**: This is an internal Outlook object, not directly exposed through existing interfaces. You should treat this as a mode of Office.context.mailbox.item.
Office.MessageRead	The message read mode of Office.context.mailbox.item. **Important**: This is an internal Outlook object, not directly exposed through existing interfaces. You should treat this as a mode of Office.context.mailbox.item.
Office.NotificationMessageAction	The definition of the action for a notification message.
Office.NotificationMessageDetails	An array of NotificationMessageDetails objects are returned by the NotificationMessages.getAllAsync method.
Office.NotificationMessages	The NotificationMessages object is returned as the notificationMessages property of an item.
Office.OfficeThemeChangedEventArgs	Provides the updated Office theme that raised the Office.EventType.OfficeThemeChanged event.
Office.Organizer	Represents the appointment organizer, even if an alias or a delegate was used to create the appointment. This object provides a method to get the organizer value of an appointment in an Outlook add-in.
Office.PhoneNumber	Represents a phone number identified in an item. Read mode only. An array of PhoneNumber objects containing the phone numbers found in an email message is returned in the phoneNumbers property of the Entities object that is returned when you call the getEntities method on the selected item.
Office.Recipients	Represents recipients of an item. Compose mode only.
Office.RecipientsChangedEventArgs	Provides change status of recipients fields when the Office.EventType.RecipientsChanged event is raised.
Office.RecipientsChangedFields	Represents RecipientsChangedEventArgs.changedRecipientFields object.
Office.Recurrence	The Recurrence object provides methods to get and set the recurrence pattern of appointments but only get the recurrence pattern of meeting requests. It will have a dictionary with the following keys: seriesTime, recurrenceType, recurrenceProperties, and recurrenceTimeZone (optional).
Office.RecurrenceChangedEventArgs	Provides updated recurrence object that raised the Office.EventType.RecurrenceChanged event.
Office.RecurrenceProperties	Represents the properties of the recurrence.
Office.RecurrenceTimeZone	Represents the time zone of the recurrence.
Office.ReplyFormAttachment	A file or item attachment. Used when displaying a reply form.
Office.ReplyFormData	A ReplyFormData object that contains body or attachment data and a callback

	function. Used when displaying a reply form.
Office.RoamingSettings	The settings created by using the methods of the RoamingSettings object are saved per add-in and user. That is, they are available only to the add-in that created them, and only from the user's mailbox in which they are saved. While the Outlook Add-in API limits access to these settings to only the add-in that created them, these settings should not be considered secure storage. They can be accessed by Exchange Web Services or Extended MAPI. They should not be used to store sensitive information such as user credentials or security tokens. The name of a setting is a String, while the value can be a String, Number, Boolean, null, Object, or Array. The RoamingSettings object is accessible via the roamingSettings property in the Office.context namespace. **Important**: The RoamingSettings object is initialized from the persisted storage only when the add-in is first loaded. For task panes, this means that it is only initialized when the task pane first opens. If the task pane navigates to another page or reloads the current page, the in-memory object is reset to its initial values, even if your add-in has persisted changes. The persisted changes will not be available until the task pane (or item in the case of UI-less add-ins) is closed and reopened.
Office.Sensitivity	Provides methods to get and set the appointment sensitivity of a meeting in an Outlook add-in.
Office.SeriesTime	The SeriesTime object provides methods to get and set the dates and times of appointments in a recurring series and get the dates and times of meeting requests in a recurring series.
Office.SessionData	Provides methods to manage an item's session data.
Office.SharedProperties	Represents the properties of an appointment or message in a shared folder, mailbox, or calendar.
Office.Subject	Provides methods to get and set the subject of an appointment or message in an Outlook add-in.
Office.TaskSuggestion	Represents a suggested task identified in an item. Read mode only. The list of tasks suggested in an email message is returned in the taskSuggestions property of the Entities object that is returned when the getEntities or getEntitiesByType method is called on the active item.
Office.Time	The Time object is returned as the start or end property of an appointment in compose mode.
Office.UserProfile	Information about the user associated with the mailbox. This includes their account type, display name, email address, and time zone.

CREATE THE ADD-IN

Users can create an Office Add-in by using the Yeoman generator for Office Add-ins or Visual Studio. The Yeoman generator creates a Node.js project that can be managed with Visual Studio Code or any other editor, whereas Visual Studio creates a Visual Studio solution.

METHOD 1 - OFFICE 365 WORD WEB ADDIN - YEOMAN GENERATOR - VISUAL CODE

Prerequisites

1. Node.js (the latest LTS version)
2. The latest version of Yeoman and the Yeoman generator for Office Add-ins. To install these tools globally, run the following command via the command prompt:

```
npm install -g yo generator-office
```

Create the add-in project

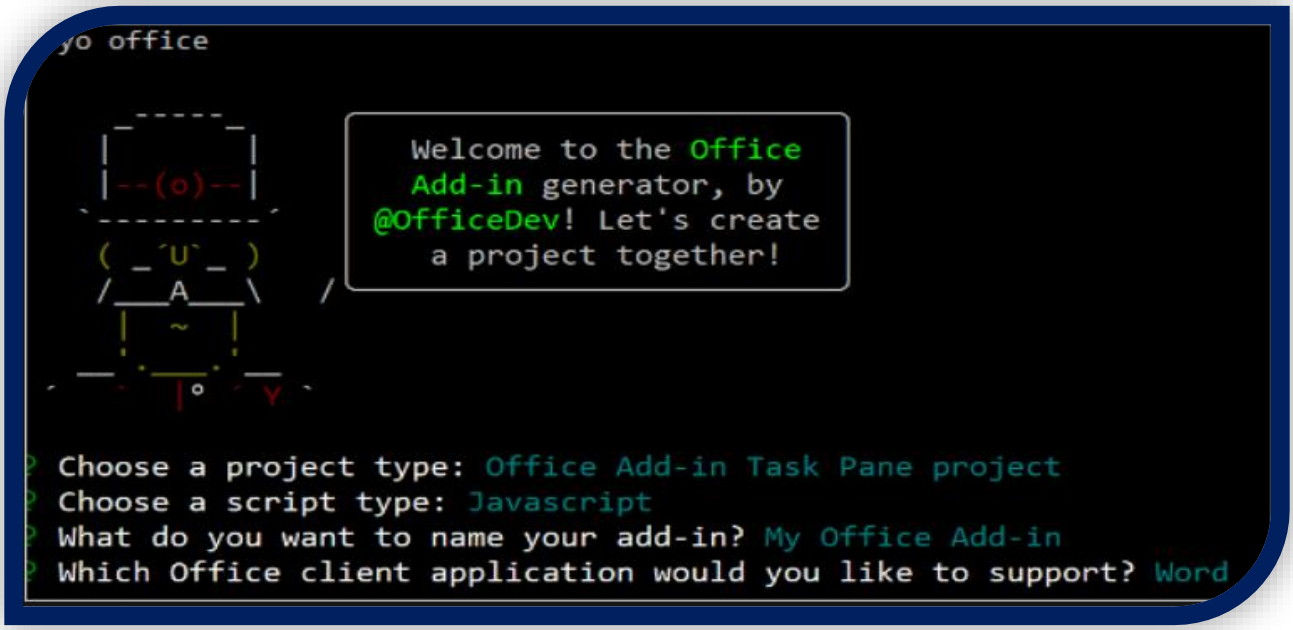
Run the following command to create an add-in project using the Yeoman generator:

```
yo office
```

When prompted, provide the following information to create your add-in project:

1. Choose a project type: Office Add-in Task Pane project
2. Choose a script type: JavaScript
3. What do you want to name your add-in? My Office Add-in

4. Which Office client application would you like to support? Outlook



After completing the wizard, the generator creates the project and installs supporting Node components.

Explore the project

The add-in project that the user has just created with the Yeoman generator contains sample code for a very basic task pane add-in. To explore the components of the add-in project, open the project in the code editor, and review the files listed below.

- The ./manifest.xml file in the root directory of the project defines the settings and capabilities of the add-in.
- The ./src/taskpane/taskpane.html file contains the HTML markup for the task pane.
- The ./src/taskpane/taskpane.css file contains the CSS that's applied to content in the task pane.
- The ./src/taskpane/taskpane.js file contains the Office JavaScript API code that facilitates interaction between the task pane and the Office host application.

Update the code

1. In the code editor, open the file **./src/taskpane/taskpane.html** and replace the entire <main> element (within the <body> element) with the following markup. This new markup adds a label where the script in **./src/taskpane/taskpane.js** will write data.

```
<main id="app-body" class="ms-welcome__main" style="display: none;">
    <h2 class="ms-font-xl"> Discover what Office Add-ins can do for you today! </h2>
    <p><label id="item-subject"></label></p>
    <div role="button" id="run" class="ms-welcome__action ms-Button ms-Button--hero ms-font-xl">
        <span class="ms-Button-label">Run</span>
    </div>
</main>
```

2. In the code editor, open the file **./src/taskpane/taskpane.js** and add the following code within the run function. This code uses the Office JavaScript API to get a reference to the current message and write its subject property value to the task pane.

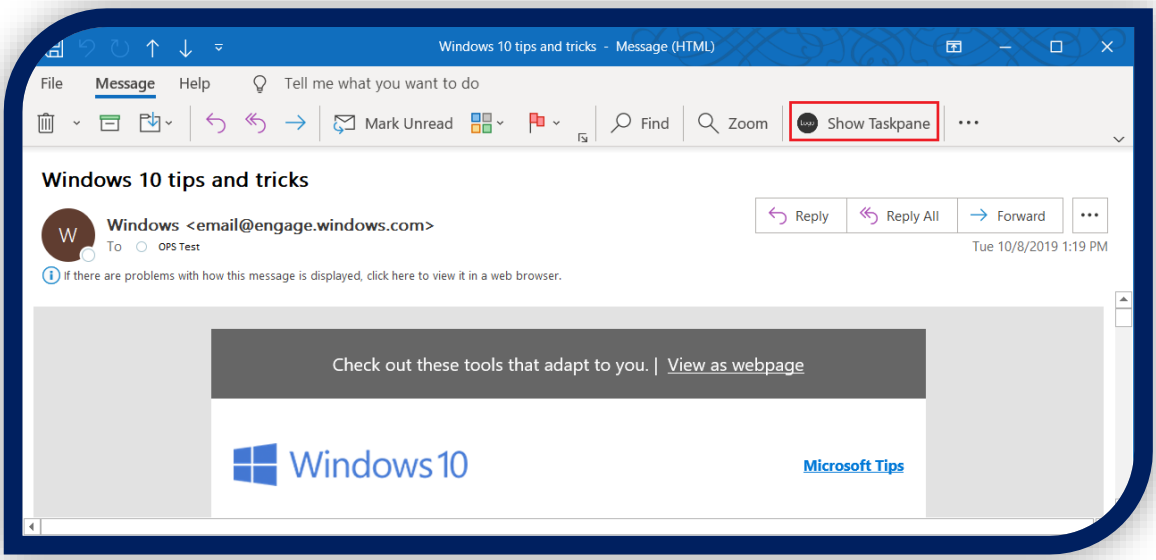
```javascript
// Get a reference to the current message
var item = Office.context.mailbox.item;

// Write message property value to the task pane
document.getElementById("item-subject").innerHTML = "<b>Subject:</b> <br/>" + item.subject;
```

Try it out

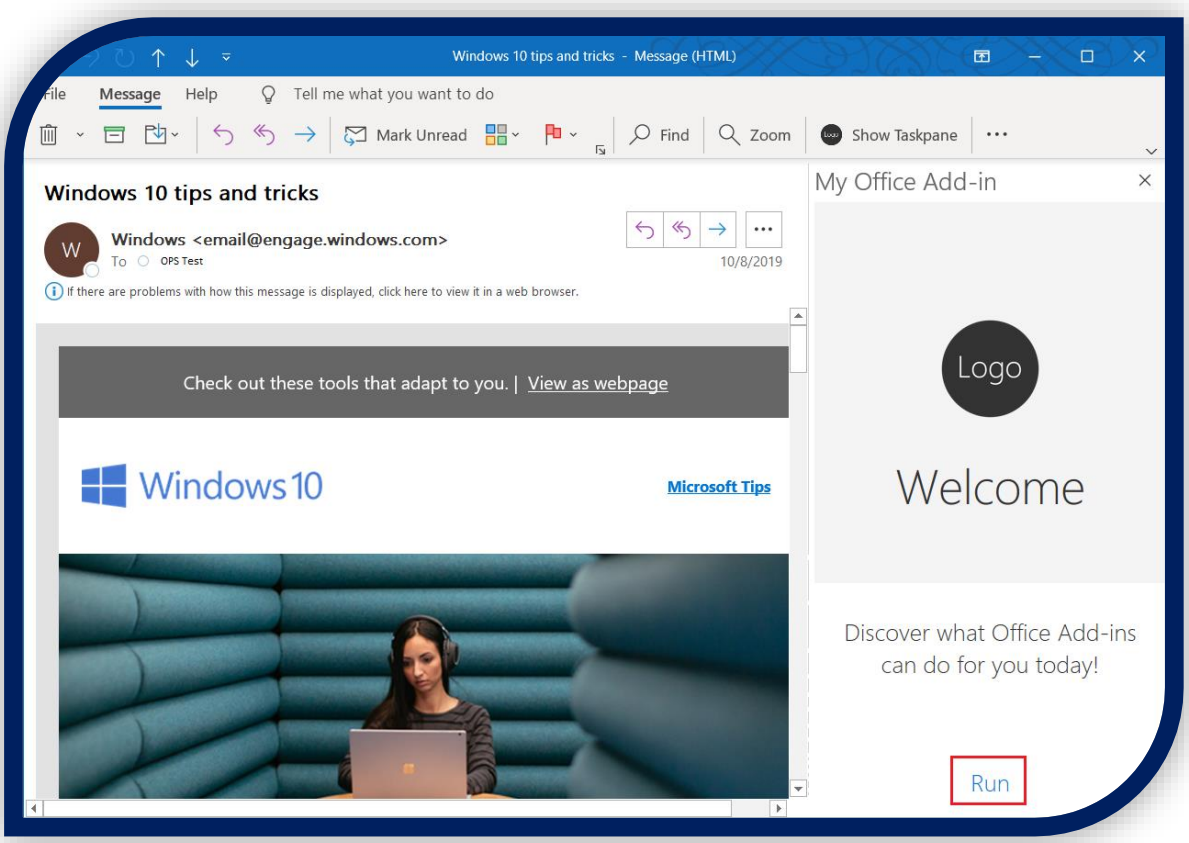
1. Run the following command in the root directory of your project. When you run this command, the local web server will start (if it's not already running).

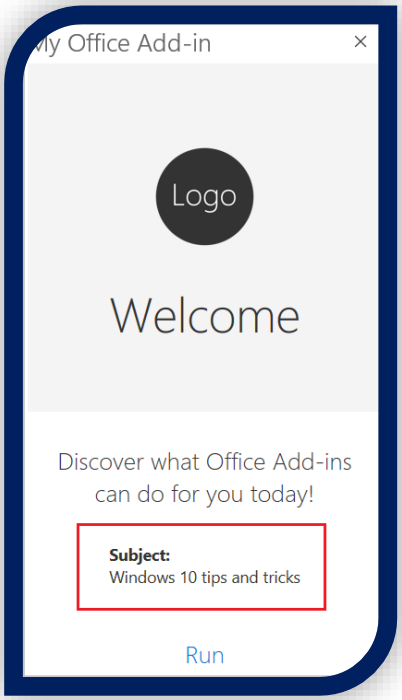
```
command line

npm run dev-server
```

2. In Outlook, select or **open a message**.
3. Choose the Home tab (or the Message tab if you opened the message in a new window), and then choose the **Show Taskpane** button in the ribbon to open the add-in task pane.



4. Scroll to the bottom of the task pane and choose the **Run** link to write the message subject to the task pane.





METHOD 2 - OFFICE 365 OUTLOOK WEB ADDIN - VISUAL STUDIO

Prerequisites

a. Visual Studio 2019 with the Office/SharePoint development workload installed
b. Office 2016 or later

Create the add-in project

1. In Visual Studio, choose **Create a new project**.
2. Using the search box, enter add-in. Choose **Outlook Web Add-in**, then select **Next**.
3. Name your project and select **Create**.
4. Visual Studio creates a solution and its two projects appear in **Solution Explorer**.
5. The **MessageRead.html** file opens in Visual Studio.

Explore the Visual Studio solution

After completing the wizard, Visual Studio creates a solution that contains two projects.

Project	Description
Add-in project	Contains only an XML manifest file, which contains all the settings that describe the add-in. These settings help the Office host determine when the add-in should be activated and where the add-in should appear. Visual Studio generates the contents of this file for you so that the user can run the project and use the add-in immediately. user change these settings any time by modifying the XML file.
Web application project	Contains the content pages of your add-in, including all the files and file references that are required to develop Office-aware HTML and JavaScript pages. While developing the add-in, Visual Studio hosts the web application on the user's local IIS server.

Update the code

1. **MessageRead.html** specifies the HTML that will be rendered in the add-in's task pane.
In **MessageRead.html**, replace the <body> element with the following markup and save the file.

```
body class="ms-font-m ms-welcome">
    <div class="ms-Fabric content-main">
        <h1 class="ms-font-xxl">Message properties</h1>
        <table class="ms-Table ms-Table--selectable">
            <thead>
                <tr>
                    <th>Property</th>
                    <th>Value</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td><strong>Id</strong></td>
                    <td class="prop-val"><code><label id="item-id"></label></code></td>
                </tr>
                <tr>
                    <td><strong>Subject</strong></td>
                    <td class="prop-val"><code><label id="item-subject"></label></code></td>
                </tr>
                <tr>
                    <td><strong>Message Id</strong></td>
                    <td class="prop-val"><code><label id="item-internetMessageId"></label></c
                </tr>
                <tr>
                    <td><strong>From</strong></td>
                    <td class="prop-val"><code><label id="item-from"></label></code></td>
                </tr>
            </tbody>
        </table>
    </div>
</body>
```

2. Open the file **MessageRead.js** at the root of the web application project. This file specifies the script for the add-in. Replace the entire contents with the following code and save the file.

```
    'use strict';
(function () {

Office.onReady(function () {
    // Office is ready
    $(document).ready(function () {
        // The document is ready
        loadItemProps(Office.context.mailbox.item);
    });
});

function loadItemProps(item) {
    // Write message property values to the task pane
    $('#item-id').text(item.itemId);
    $('#item-subject').text(item.subject);
    $('#item-internetMessageId').text(item.internetMessageId);
    $('#item-from').html(item.from.displayName + " &lt;" + item.from.emailAddress + "&gt;");
}
})();
```

3. Open the file **MessageRead.css** in the root of the web application project. This file specifies the custom styles for the add-in. Replace the entire contents with the following code and save the file.

```css
html,
body {
    width: 100%;
    height: 100%;
    margin: 0;
    padding: 0;
}

td.prop-val {
    word-break: break-all;
}

.content-main {
    margin: 10px;
}
```

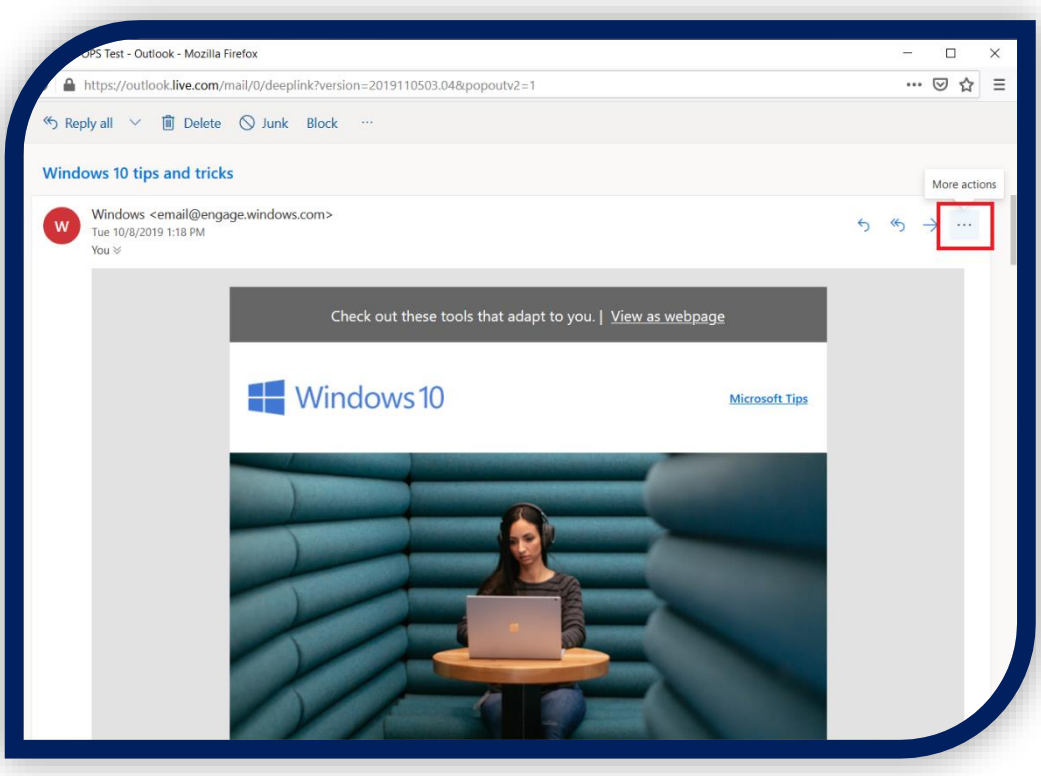
Update the manifest

1. Open the XML manifest file in the Add-in project. This file defines the add-in's settings and capabilities.
2. The ProviderName element has a placeholder value. Replace it with your name.
3. The DefaultValue attribute of the DisplayName element has a placeholder. Replace it with My Office Add-in.
4. The DefaultValue attribute of the Description element has a placeholder. Replace it with My First Outlook add-in.
5. Save the file.

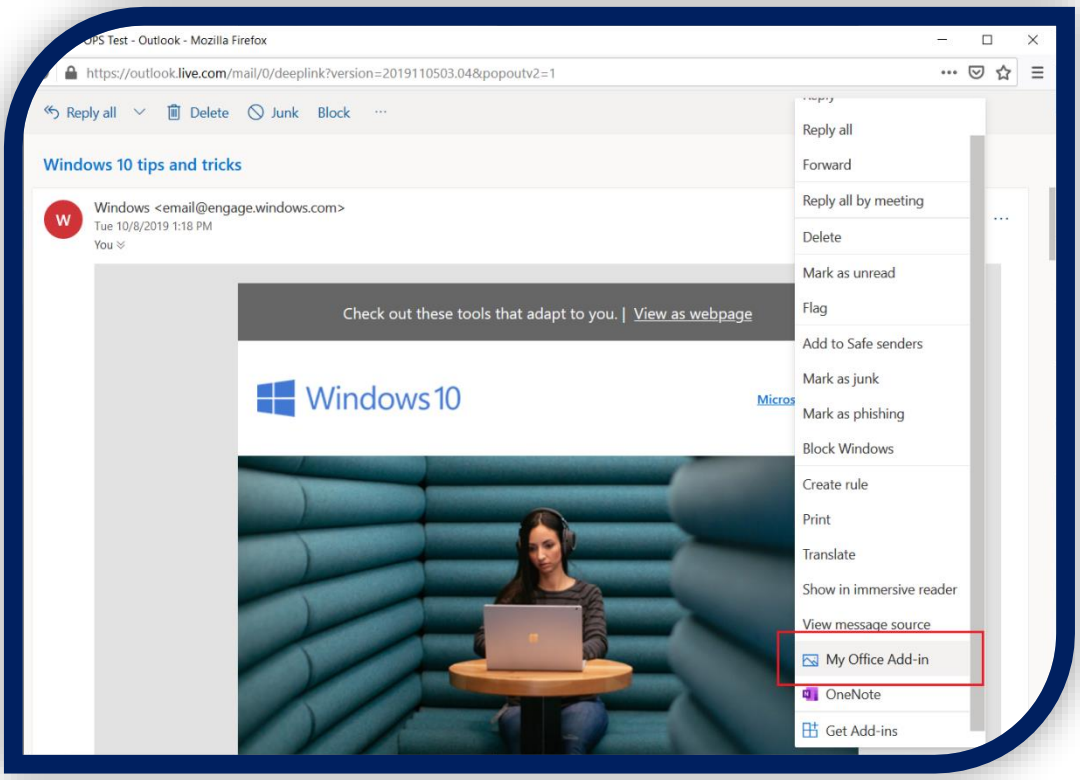
```
...
<ProviderName>John Doe</ProviderName>
<DefaultLocale>en-US</DefaultLocale>
<!-- The display name of your add-in. Used on the store and various places of the Office UI such as the add-ins dialog. -->
<DisplayName DefaultValue="My Office Add-in" />
<Description DefaultValue="My First Outlook add-in"/>
...
```

Try it out

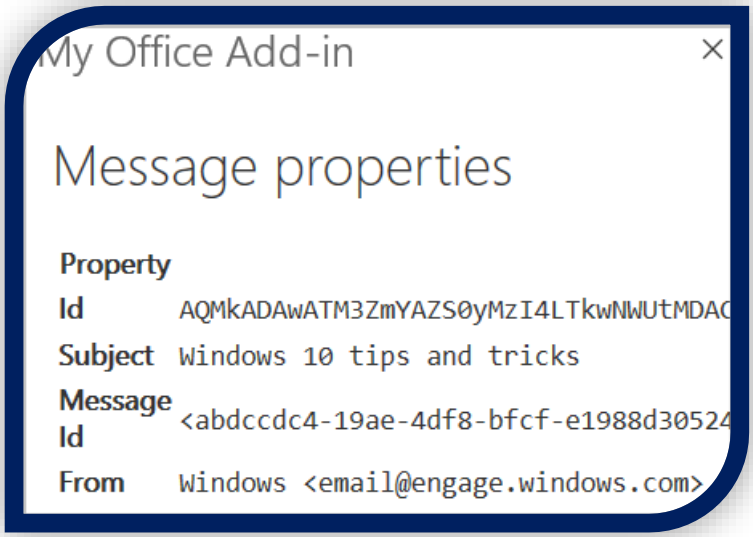
1. Using Visual Studio, test the newly created Outlook add-in by pressing F5 or choosing the **Start** button. The add-in will be hosted locally on IIS.
2. In the **Connect to Exchange email account** dialog box, enter the email address and password for the Microsoft account and then choose **Connect**. When the Outlook.com login page opens in a browser, sign in to your email account with the same credentials as you entered previously.
3. In Outlook on the web, select or open a message.
4. Within the message, locate the ellipsis for the overflow menu containing the add-in's button.



5. Within the overflow menu, locate the add-in's button.



6. Click the button to open the add-in's task pane.



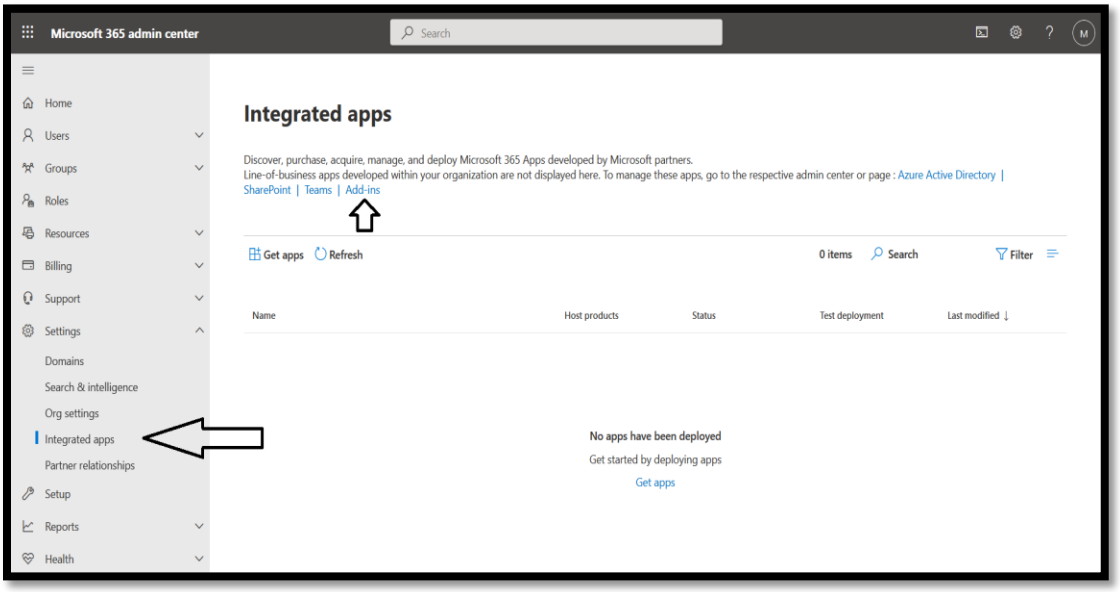
DEPLOYMENT OF ADDIN

There are multiple methods of deployment but we have followed the Centralized Deployment.

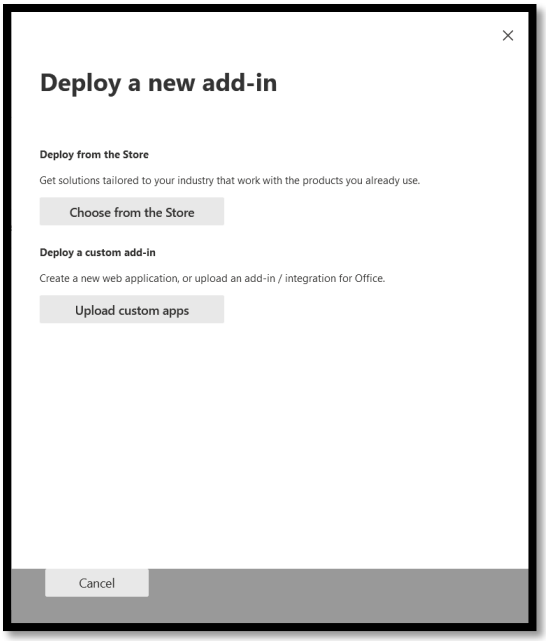
CENTRALIZED DEPLOYMENT

Follow steps below to publish an Office Add-in via Centralized Deployment:

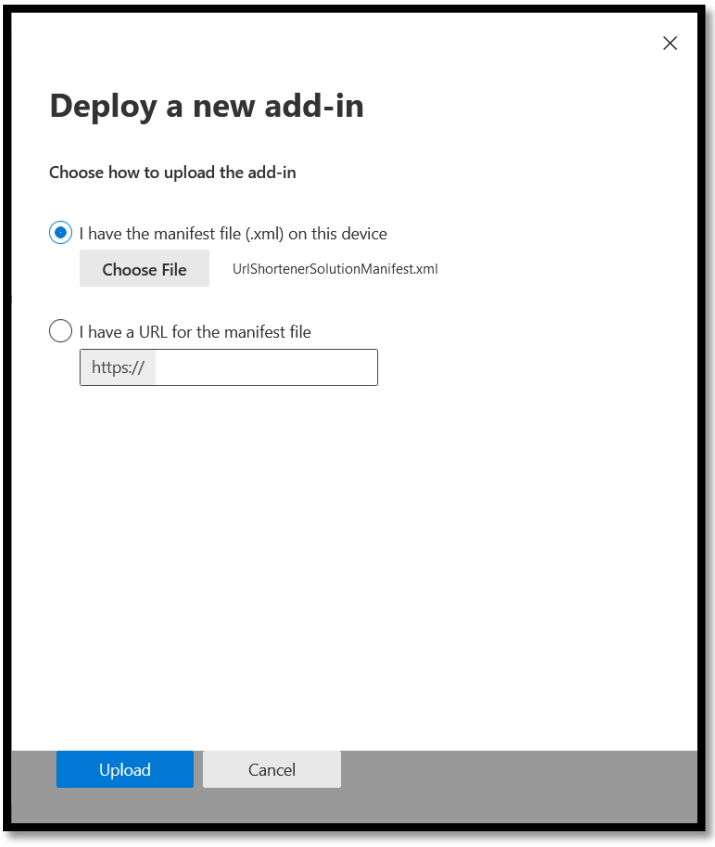
1. Sign in to Microsoft 365 with your work or education account.
2. Select the app launcher icon in the upper-left and choose **Admin**.
3. In the navigation menu, press **Show more**, then choose **Settings** > **Integrated apps**.
4. Choose **Add-ins** at the top of the page



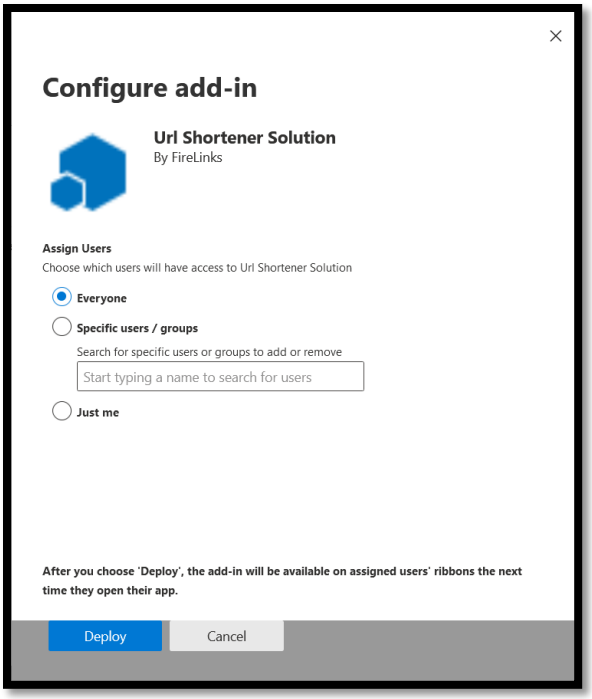
5. Choose **Deploy Add-In** at the top of the page.
6. Choose **Next** after reviewing the requirements.
7. Choose **Upload Custom apps** in the following page



8. Click on **Choose file** and select Add-in manifest file (.xml)
9. Choose **Upload** at the end of the task pane



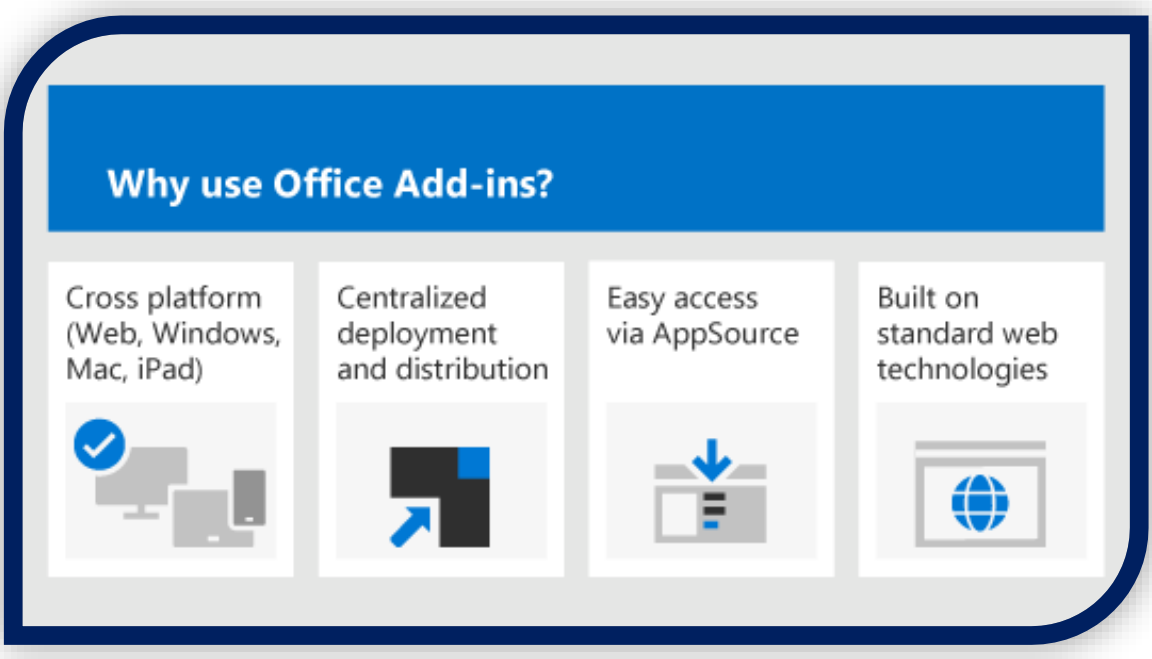
10. On the **Assign Users** page, choose **Everyone**, **Specific Users/Groups**, or **Only me**. Use the search box to find the users and groups to whom you want to deploy the add-in.



11. When finished, choose **Deploy**. This process may take up to three minutes. Then, finish the walkthrough by pressing **Next**.

HOW ARE OFFICE ADD-INS DIFFERENT FROM COM AND VSTO ADD-INS?

COM or VSTO add-ins are earlier Office integration solutions that run only on Office on Windows. Unlike COM add-ins, Office Add-ins don't involve code that runs on the user's device or in the Office client. For an Office Add-in, the host application, for example, Excel, reads the add-in manifest and hooks up the add-in's custom ribbon buttons and menu commands in the UI. When needed, it loads the add-in's JavaScript and HTML code, which



executes in the context of a browser in a sandbox.

Office Add-ins provide the following advantages over add-ins built using VBA, COM, or VSTO:

- **Cross-platform support**. Office Add-ins run in Office on the web, Windows, Mac, and iPad.
- **Centralized deployment and distribution**. Admins can deploy Office Add-ins centrally across an organization.
- **Easy access via AppSource**. You can make your solution available to a broad audience by submitting it to AppSource.
- **Based on standard web technology**. You can use any library you like to build Office Add-ins.

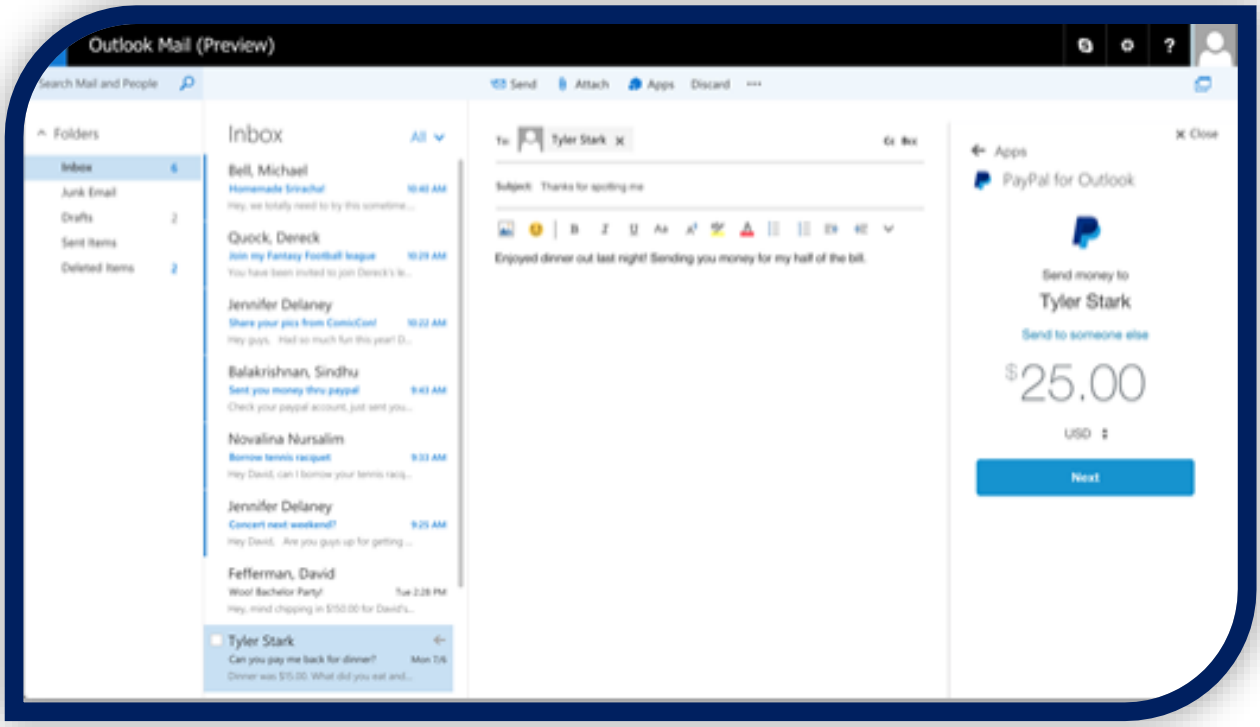
EXAMPLES OF OUTLOOK WEB ADD-IN IN APP SOURCE

PayPal for Outlook

Company: PayPal Inc

AppSource URL: link

Description: PayPal for Outlook provides the facility of sending money using the email address. When this add-in is used, it Can send data over the Internet. This add-in can access and modify personal information in the active message, such as the body, subject, sender, recipients, and attachment information. It may send this data to a third-party service. Other items in your mailbox can't be read or modified.



Translator for Outlook

Company: Microsoft Corporation

AppSource URL: link

Description: Translator provides a simple way to translate and read messages in the preferred language, across devices.

- Install once and use across devices. Enable Translator once, and use across all the devices, with Outlook for iOS, Mac, Windows, Office 365 & Outlook.com. (Coming soon to Android.)
- Translate seamlessly. Don't switch between apps to get a translation; see a fully translated view of the message with a tap of a button

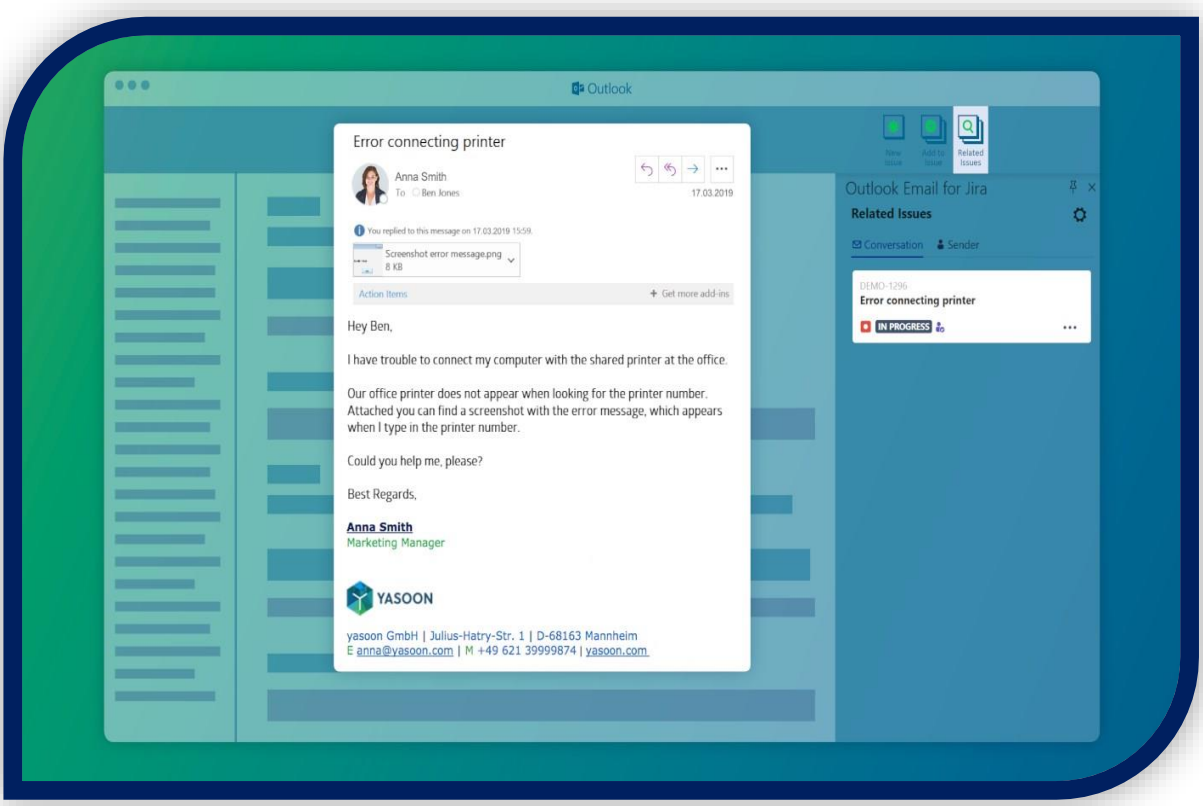
Jira for Outlook

Company: yasoon GmbH

AppSource URL: link.

Description: Jira for Outlook eases your workflow with many features:

- Create and update issues without leaving Outlook: Take over email attachments and text formatting like lists, colors and tables
- Get Jira context for emails: View related issues for the same conversation or sender in a separate sidebar
- Fully integrated with Jira Service Desk: Create tickets on behalf of existing Service Desk users or create a new user for the sender of the email
- Be efficient on all your devices: Use all advantages of Jira on the go (Runs on Outlook for Windows, iOS, Mac and Web)



CONCLUSION

In this case study, a brief introduction about the Microsoft Office 365 Outlook Web add-in, its architecture, development & deployment ways & a demo of the introductory level is discussed in detail.

Our Microsoft Office 365 Consulting, add-in Development, Customization, Integration services, and solutions, can help companies maximize business performance, overcoming market challenges, achieving profitability, and providing the best customer care service.

CONTACT US

Shahzad Sarwar

Cognitive Convergence Team