

Introduction to the SharePoint Framework

Rob Windsor
rob@robwindsor.com
@robwindsor

About Me



Rob Windsor

.NET/Microsoft 365 developer, trainer, author
Microsoft MVP | Microsoft 365 Development



Twitter: <https://twitter.com/robwindsor>

LinkedIn: <https://www.linkedin.com/in/rwindsor>

Blog: <https://robwindsor.hashnode.dev>

YouTube: <https://www.youtube.com/@RobWindsor>

GitHub: <https://github.com/rob-windsor>

What is the SharePoint Framework?

The SharePoint Framework (SPFx) is a page and web part model that provides full support for client-side SharePoint development, easy integration with SharePoint data, and extending Microsoft Teams and Microsoft Viva.

Objectives:

- Reliable and predictable extensibility model for Microsoft 365
- Use of standard web stack development tools
- Automatic single sign-on with Azure Active Directory
- Simplify custom solution hosting

What can you build with the SharePoint Framework?

- Client-side web parts
 - Components that can be used in SharePoint pages
 - Can also be used as a tab in Teams
 - Can also be used as personal apps in Teams, Outlook, and or the Microsoft 365 app
- SharePoint Framework extensions
 - Extend SharePoint page experiences
- Adaptive Card extensions
 - Components that can be used in SharePoint pages or Viva Connections
- Library components
 - Package code for use with multiple SharePoint Framework projects

Tooling

Build Process & Tooling

- Yeoman Templates
- Gulp (task runner)
- Node.js
- NPM (package manager)
- SystemJS
- Webpack
- SASS
- TypeScript



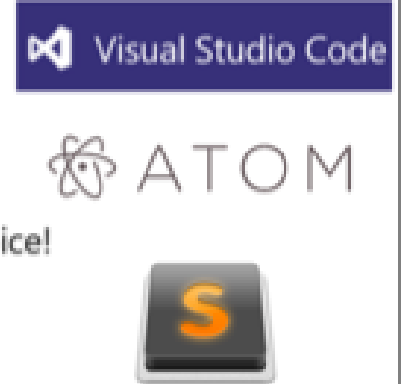
Web Frameworks

- React
- Angular
- Knockout
- jQuery
- ... and more... your choice!



Code Editors

- Visual Studio Code
- Atom
- Sublime
- Webstorm
- ... and more... your choice!



Versions

- Current GA version is 1.20.0
- Each version of SPFx is compatible with different versions of tools and libraries
- Developers will generally want to have multiple versions of Node.js installed on their machine so they can work with projects that use different versions of SPFx
 - Node Version Manager (nvm)

SPFx	Node.js (LTS)	TypeScript	React
1.20.0	v18	v4.5, v4.7	v17.0.1
1.19.0	v18	v4.5, v4.7	v17.0.1
1.18.2	v16, v18	v4.5, v4.7	v17.0.1
1.18.1	v16, v18	v4.5, v4.7	v17.0.1
1.18	v16, v18	v4.5, v4.7	v17.0.1
1.17.4	v16.13+	v4.5	v17.0.1
1.17.3	v16.13+	v4.5	v17.0.1
1.17.2	v16.13+	v4.5	v17.0.1
1.17.1	v16.13+	v4.5	v17.0.1
1.17.0	v16.13+	v4.5	v17.0.1
1.16.1	v16.13+	v4.5	v17.0.1
1.16.0	v16.13+	v4.5	v17.0.1
1.15.2	v12, v14, v16	v4.5	v16.13.1
1.15.0	v12, v14, v16	v4.5	v16.13.1

<https://learn.microsoft.com/en-us/sharepoint/dev/spfx/compatibility>

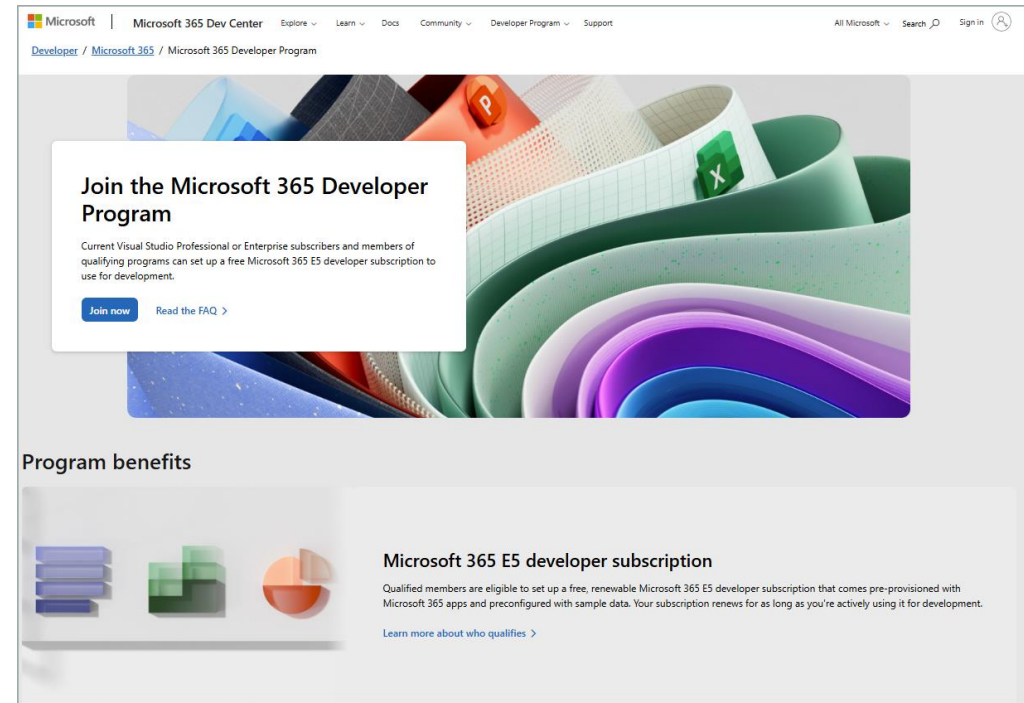
Setting Up a Development Environment

1. Install a code editor like Visual Studio Code
2. Install Node Version Manager (nvm)
3. Use nvm to install version of Node.js compatible with version of SPFx you will use
4. Tell nvm to use the version of Node.js you just installed
5. Install Gulp-CLI, Yeoman, and the Yeoman SharePoint generator

Repeat steps 3 to 5 above for any additional versions of SPFx you may use.

Microsoft 365 Developer Program

- Free 25-user development tenant
- Available for three months
 - Tenant auto-renews if being used for development
- **Due to bad actors using developer tenants for nefarious purposes, the program is currently only available to Visual Studio subscribers**
 - **Not certain if/when it will reopen to everyone**



<https://developer.microsoft.com/en-us/microsoft-365/dev-program>

Creating a New Solution (Project)

- Also known as scaffolding a new solution
- Navigate to a folder where you want the solution to be created
- Run Yeoman SharePoint generator by executing **yo @microsoft/sharepoint**
- Generator will ask questions about the name of the solution, the type of solution you wish to create, and the web framework you wish to use
- The solution files will be scaffolded and **npm install** will be run after you've answered the questions

```
C:\Dev\HelloWorld>yo @microsoft/sharepoint
```



```
Welcome to the Microsoft  
365 SPFx Yeoman  
Generator@1.18.2
```

```
See https://aka.ms/spfx-yeoman-info for more information on  
how to use this generator.
```

```
Let's create a new Microsoft 365 solution.
```

```
? What is your solution name? HelloWorld
```

```
? Which type of client-side component to create? WebPart
```

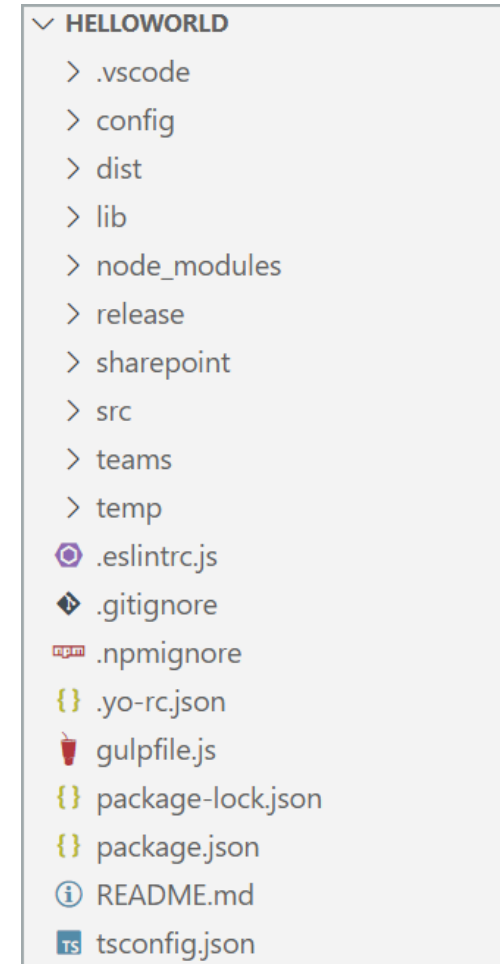
```
Add new Web part to solution hello-world.
```

```
? What is your Web part name? HelloWorld
```

```
? Which template would you like to use? React
```

Solution Structure

- **config:** configuration files used by the project's various tasks.
- **dist:** files generated when you bundle your project, regardless of which switch you use. Unminified JavaScript files and source maps contained in this folder are used when you run in debug mode.
- **lib:** temporary files generated from the compilation and transpilation of TypeScript to JavaScript and SCSS to CSS files.
- **release:** contains a subfolder named assets that contains the files generated when you use the ship switch when you bundle. These files are deployed to the CDN. Folder also contains two more subfolders that contain manifest files.
- **src:** the source code for your project.
- **temp:** files used by the local development web server.



Gulp Tasks

Gulp is a task runner used when developing SharePoint Framework solutions.

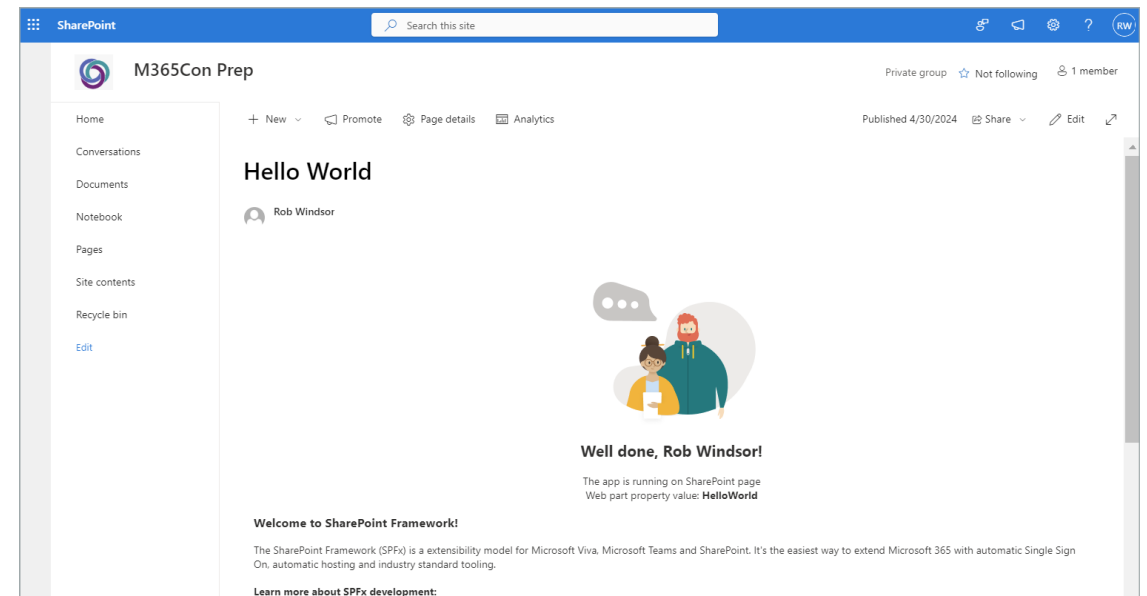
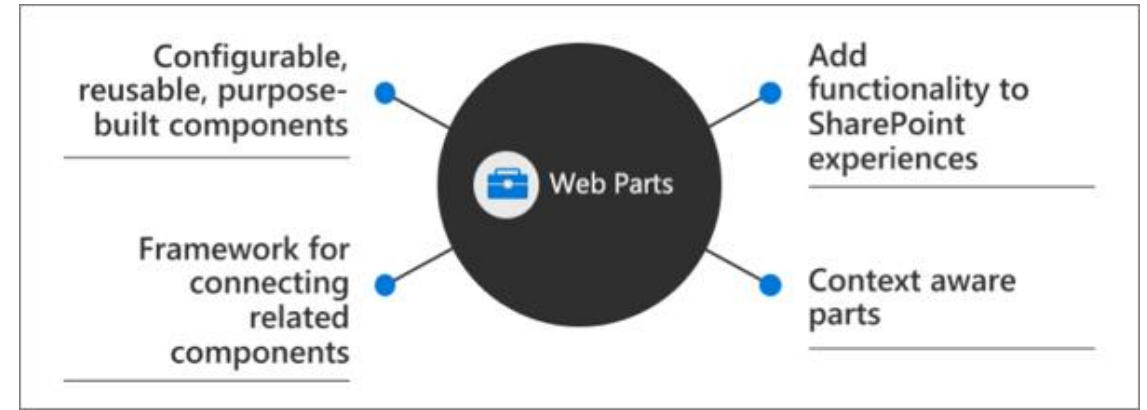
Commonly used Gulp tasks:

- **clean**: delete temporary files generated during builds or debugging sessions
- **serve**: start a debugging session
- **bundle**: bundle JavaScript and static files
- **package-solution**: create a solution package for deployment
- **trust-dev-cert**: trust the SPFx developer certificate so that the local web server can host pages from a HTTPS endpoint

```
C:\Dev\HelloWorld>gulp --tasks
Tasks for C:\Dev\HelloWorld\gulpfile.js
— clean
— build
— default
— bundle
— deploy-azure-storage
— package-solution
— test
— serve-deprecated
— trust-dev-cert
— untrust-dev-cert
— test-only
— serve
```

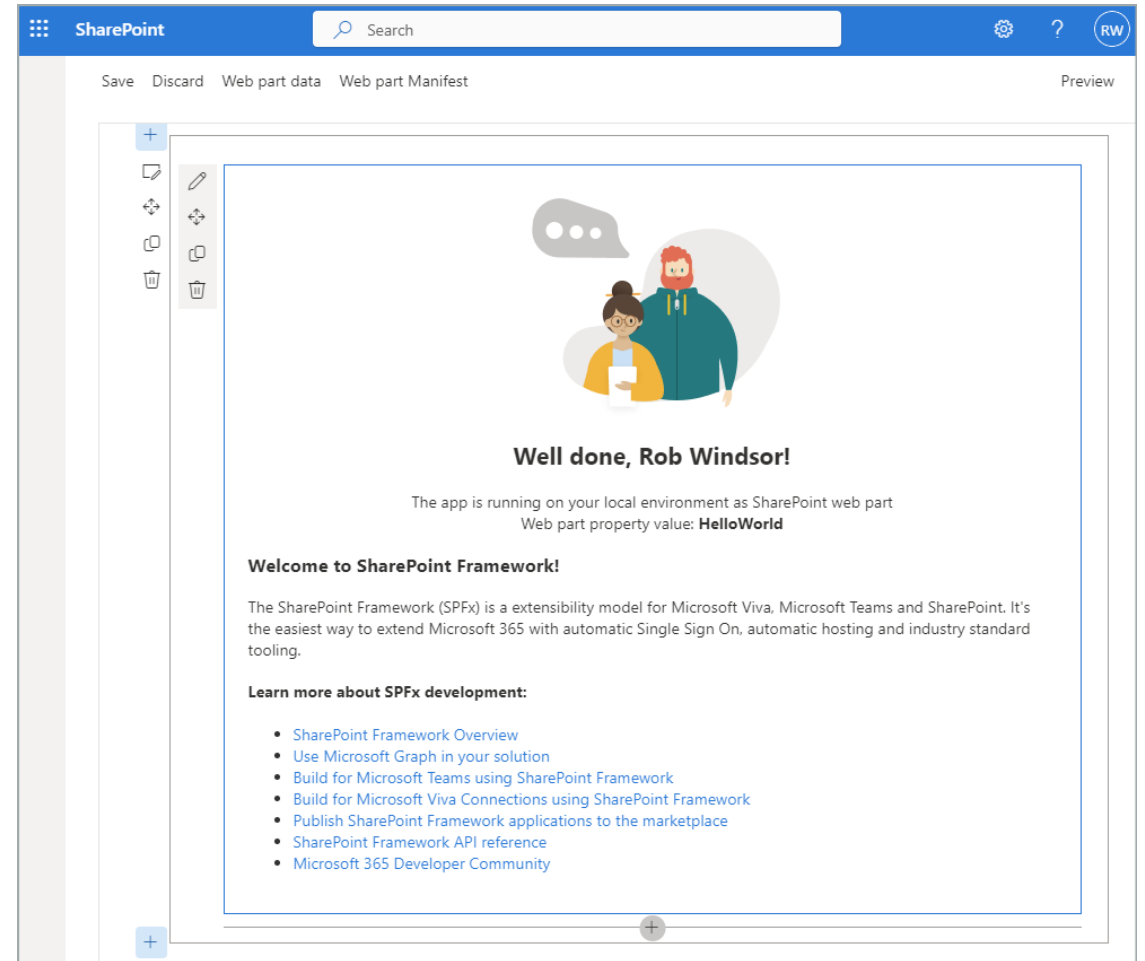
Client-side Web Parts

- Reusable components that can be used in SharePoint pages
 - Can also be used in Teams tabs
 - Can also be used as personal apps in Teams, Outlook, and or the Microsoft 365 app
- Execute client-side in the browser
- Context aware: can get information about the current user and current host application (SharePoint, Teams, Outlook, or Microsoft 365 app)
- Configurable via the Web Part Property Pane



SharePoint Framework Workbench

- Each SharePoint site contains a workbench page
 - **`https://{site-url}/_layouts/workbench.aspx`**
- The workbench is a special page that contains a single canvas that can be used to test web parts that have been deployed and/or web parts running locally

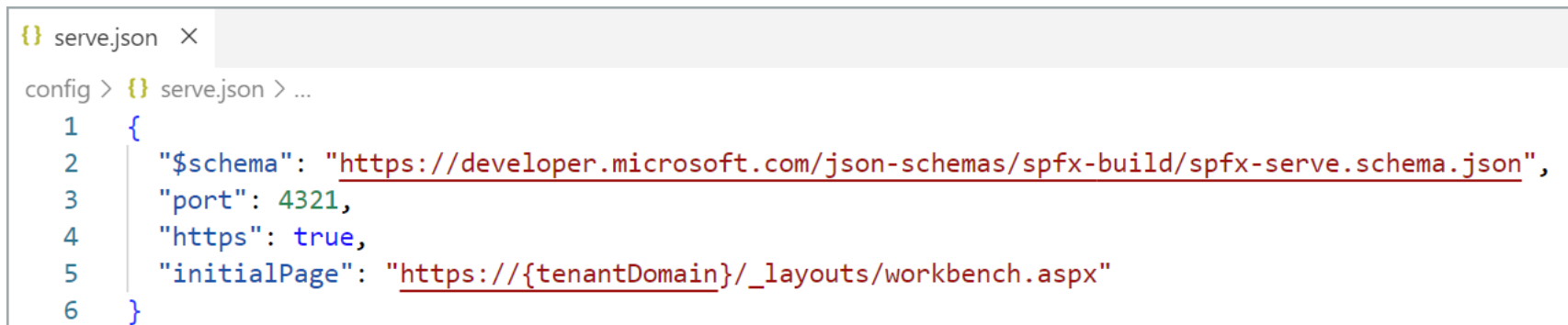


Testing a Web Part in the Workbench

Use the **gulp serve** task to start a debugging session. This will run the **build** and **bundle** tasks, start the local web server, launch the default browser, and load the workbench page at the URL configured in the **initialPage** property of the object stored in **config/serve.json**.

The default value of the **initialPage** property (shown in the image below) contains a **tenantDomain** token that will be replaced by the site URL contained in the **SPFX_SERVE_TENANT_DOMAIN** environment variable which you would need to add on your development machine. This is a good option when you commonly want to test different projects in the context of the same site.

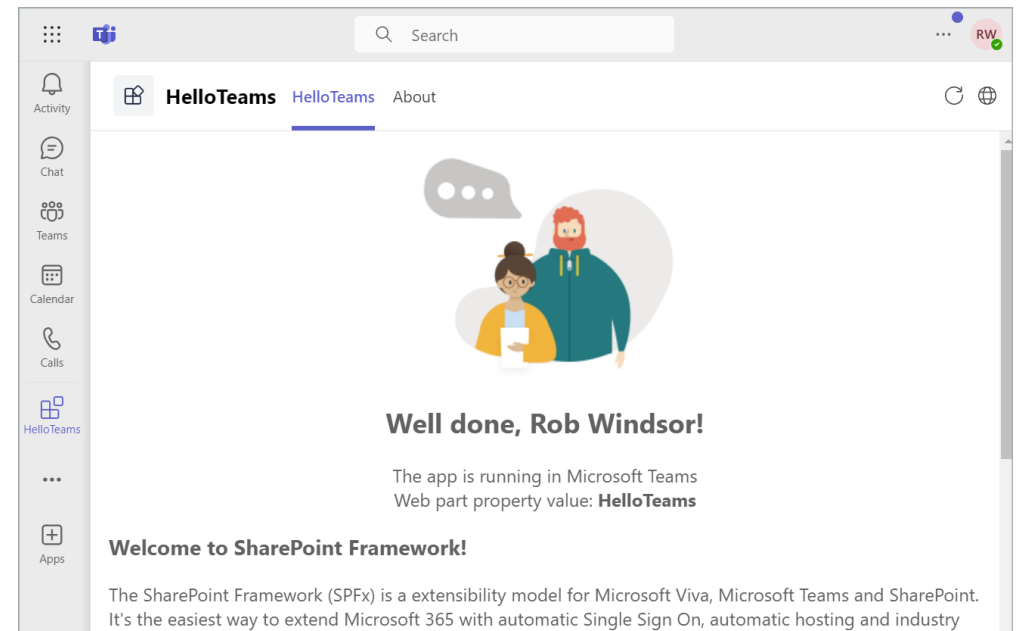
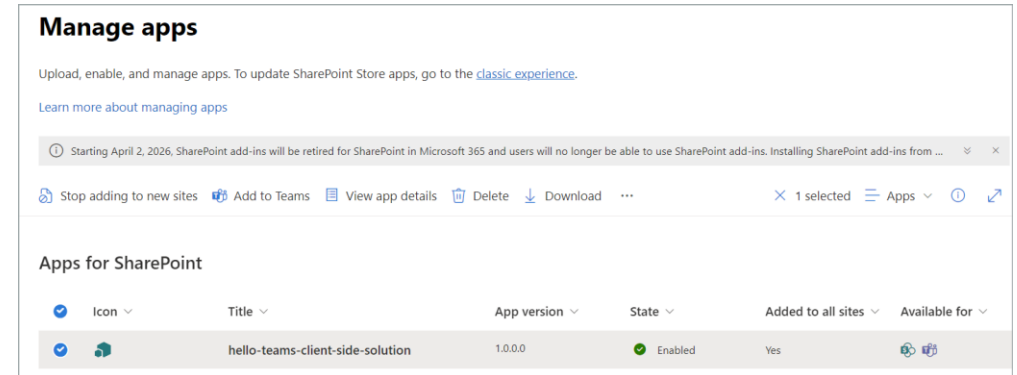
The alternative is to set the value of the **initialPage** property to the URL of the workbench page in a specific site. This is a good option when you commonly want to test different projects in the context of different sites.



```
{ } serve.json X
config > { } serve.json > ...
1  {
2    "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/spfx-serve.schema.json",
3    "port": 4321,
4    "https": true,
5    "initialPage": "https://{tenantDomain}/_layouts/workbench.aspx"
6  }
```

Using Client-side Web Parts in Teams

- Ensure the **supportedHosts** property in the web part manifest includes **TeamsTab** and/or **TeamsPersonalApp**
- Deploy the SharePoint package to the tenant app catalog and configure it for tenant wide deployment
- Select the package in the **Manage Apps** page and then select the **Add to Teams** button in the command bar.



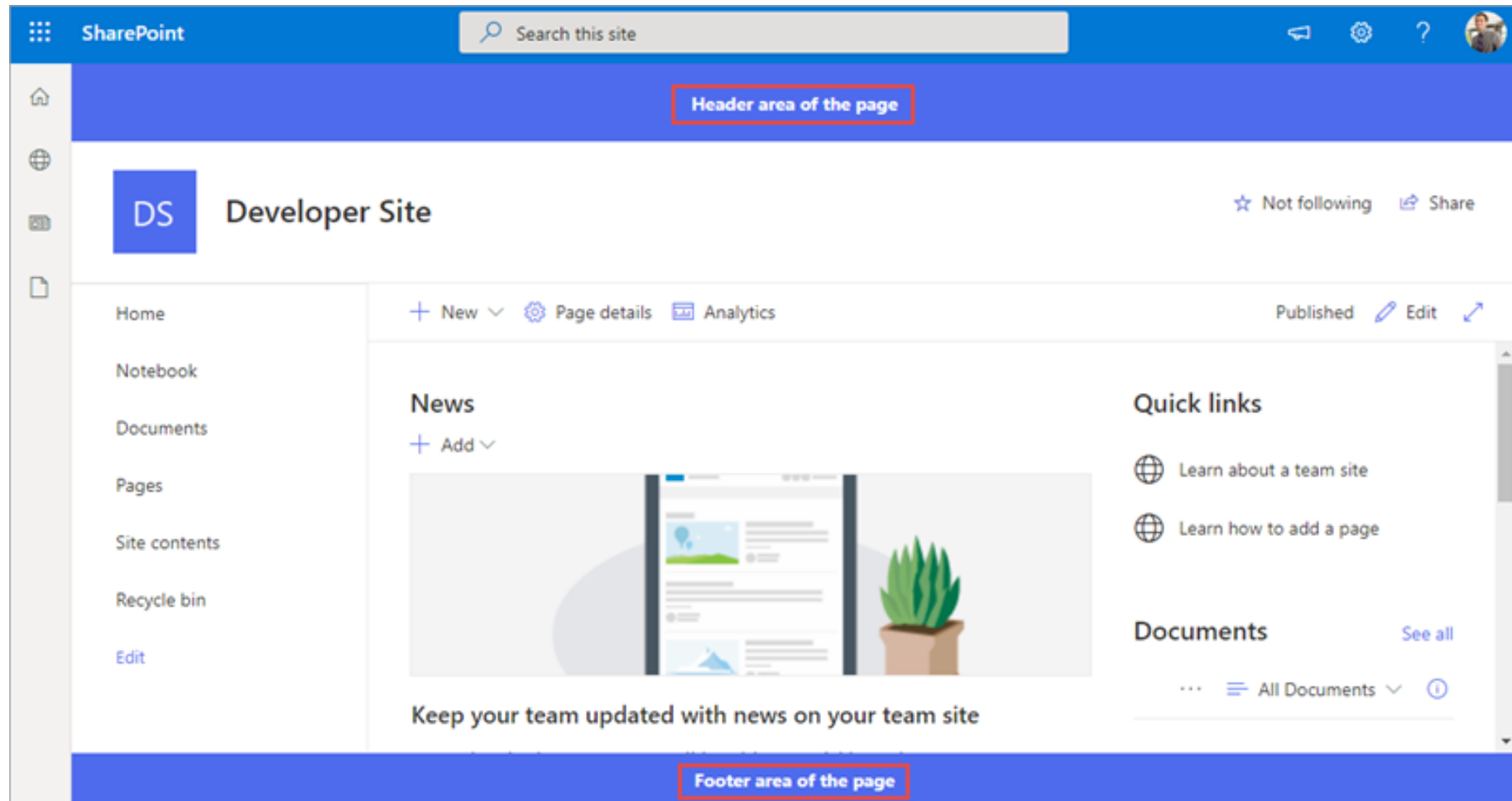
SharePoint Framework Extensions

SharePoint Framework extensions enable developers to customize and extend the SharePoint user experience.

The types of extensions available in the SharePoint Framework are:

- Application customizer
 - Add custom header and/or footer
- Field customizer
 - Change how a field is rendered in a list view
- ListView Command set
 - Add menu options to command bar or list item context menu in a list view
- Form customizer
 - Override form rendering for items in a list or library
- Search query modifier
 - Modify search queries executed using the search experience

Application Customizer





Field Customizer


Work Status ☆	
Title ▾	PercentComplete ▾ + Add column
🚀 Work item A	95% completed
🚀 Work item B	80% completed
🚀 Work item C	50% completed
🚀 Work item D	no progress

ListView Command Set Customizer

+ New


 Edit


 Edit in grid view


 Delete

Two Items Selected













Always On

Work Status 

Title 

PercentComplete 




+ Add column

	 Work item A				95
	 Work item B				80
	 Work item C				50
	 Work item D				

Form Customizer

SharePoint

Search



D

Demo

Private group Not following 4 members

Home

Conversations

Documents

Notebook

Pages

Site contents

Recycle bin

Custom List Form

T

Title

123

PercentComplete

Save

Cancel

Testing Extensions

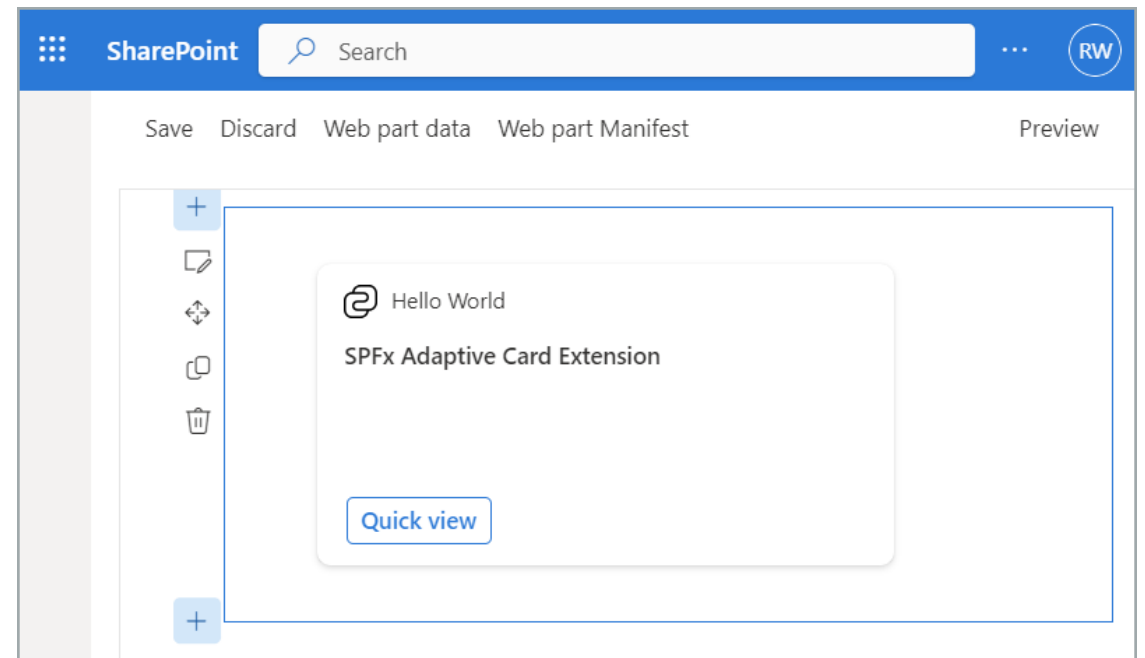
- Extensions are tested using live SharePoint pages
- The page URL and the extension property values used for debugging are configured in the **config/serve.json** file

```
{ serve.json } x
config > {} serve.json > ...
1  {
2    "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/spfx-serve.schema.json",
3    "port": 4321,
4    "https": true,
5    "serveConfigurations": {
6      "default": {
7        "pageUrl": "https://robwindsortest980.sharepoint.com/sites/M365ConPrep/SitePages/CollabHome.aspx",
8        "customActions": {
9          "45727358-fd45-4a8a-a11b-37dd756ac0b7": {
10            "location": "ClientSideExtension.ApplicationCustomizer",
11            "properties": {
12              "header": "Header area of the page",
13              "footer": "Footer area of the page"
14            }
15          }
16        }
17      },
18      "helloAppCustomizer": {
19        "pageUrl": "https://robwindsortest980.sharepoint.com/sites/M365ConPrep/SitePages/CollabHome.aspx",
20        "customActions": {
21          "45727358-fd45-4a8a-a11b-37dd756ac0b7": {
22            "location": "ClientSideExtension.ApplicationCustomizer",
23            "properties": {
24              "header": "Header area of the page",
25              "footer": "Footer area of the page"
26            }
27          }
28        }
29      }
30    }
31  }
```

Adaptive Card Extensions (ACEs)

Adaptive Cards are platform-agnostic snippets of user interface, authored in JSON, that apps and services can openly exchange. When delivered to a specific app, the JSON is transformed into native UI that automatically adapts to its surroundings.

Adaptive Card Extensions (ACEs) are a SharePoint Framework component type that use Adaptive Cards for the interface so that you can focus on the component's business logic.



Accessing Data

The SharePoint Framework contains the following types you can use to access data. Objects of these types, or factory objects that will create objects of these types, are available as properties of the native context object.

- HttpClient
 - Used when making requests to anonymous APIs
- SPHttpClient
 - Used when making requests to the SharePoint REST API
- AadHttpClient
 - Used when making requests to APIs secured by Entra ID (Azure AD)
- MSGraphClient
 - Used when making requests specifically to Microsoft Graph

Accessing Data – SPHttpClient Example

```
this.context.spHttpClient
    .get(`${this.context.pageContext.web.absoluteUrl}/_api/web?$select=Title`,
        SPHttpClient.configurations.v1)
    .then((res: SPHttpClientResponse): Promise<{ Title: string; }> => {
        return res.json();
    })
    .then((web: {Title: string}): void => {
        console.log(web.Title);
    });
```


Deployment – SharePoint Packages

Deployment of SharePoint Framework assets is done via SharePoint packages. A SharePoint package is a ZIP file with a SPPKG extension. The steps to create a SharePoint package for production deployment are:

- Run **gulp bundle --ship** to build the solution and create a minified bundle. The generated files are placed in both the **dist** folder and the **release/assets** folder.
- Run **gulp package-solution --ship** to generate/collect the files necessary for deployment. This includes the app manifest, feature definitions, element manifests, the bundle, the component manifest(s) and string localization files. All these files are placed into the SharePoint package. The SharePoint package is placed in the **sharepoint/solution** folder. The files contained in the SharePoint package can also be found in the **sharepoint/solution/debug** folder.

After creating a SharePoint package, you need to upload it to an app catalog.

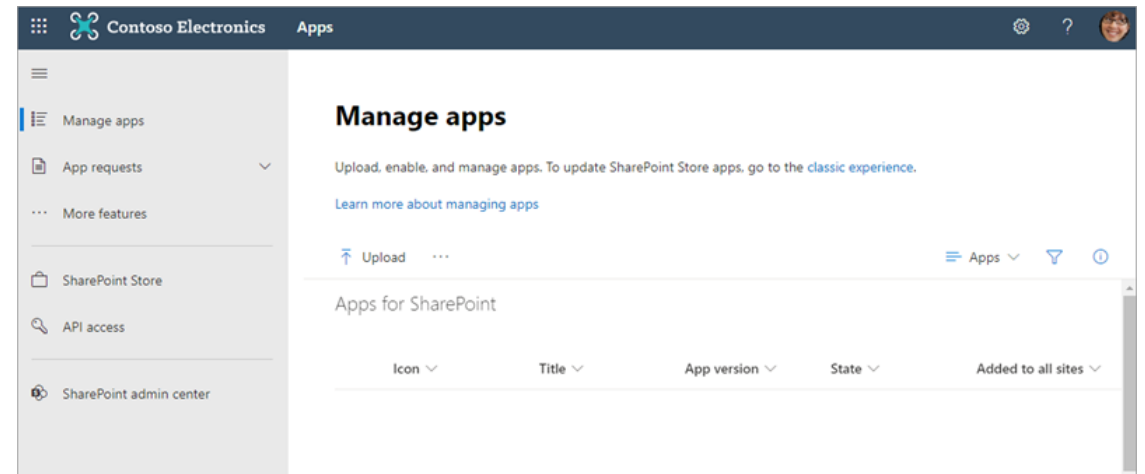
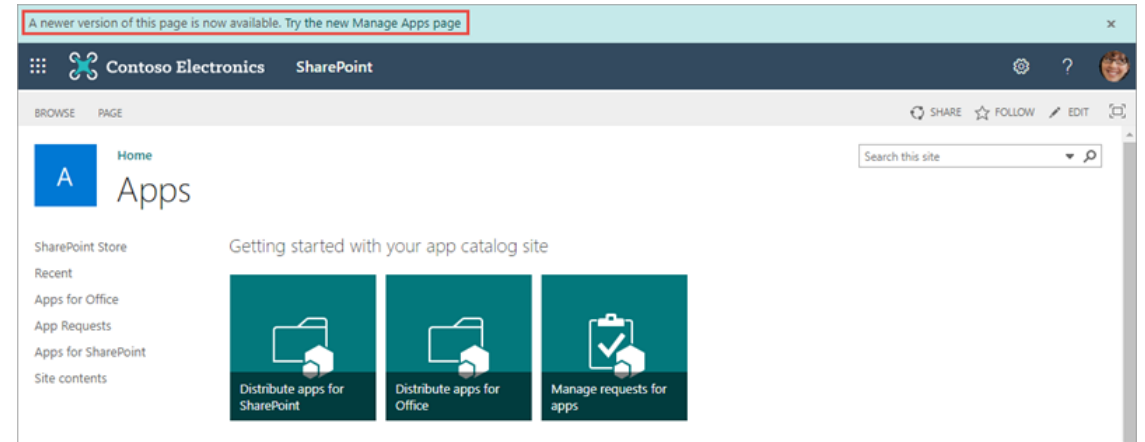
Deployment – Tenant App Catalog

Each tenant can have exactly one tenant app catalog. This is a site collection that is automatically created (if it doesn't already exist) when you go to **More features | Apps** in the SharePoint Admin Center.

Microsoft is in the process of transitioning from the classic user experience to a modern user experience for the tenant app catalog.

Navigate to **<https://{your-tenant-domain}.sharepoint.com/sites/appcatalog>** for the classic app catalog user experience.

Navigate to **https://{your-tenant-domain}.sharepoint.com/sites/appcatalog/_layouts/15/tenantAppCatalog.aspx** for the modern app catalog user experience.



Deployment – Site Collection App Catalogs

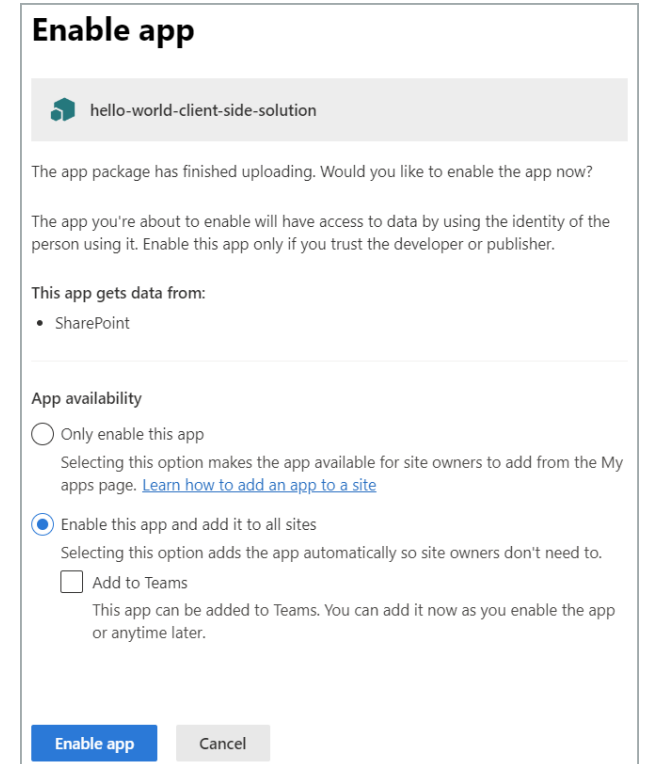
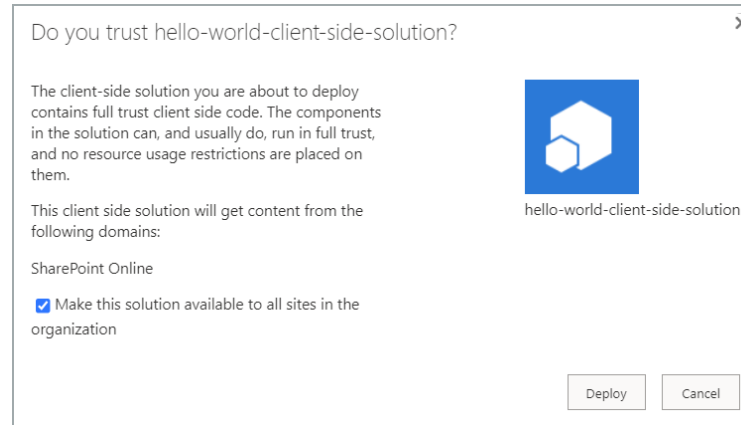
Administrators can add a site collection app catalog to any site collection.

- Use the **SharePoint Online Management Shell**, **PnP Powershell**, or **CLI for Microsoft 365** to add app catalog
 - <https://learn.microsoft.com/en-us/sharepoint/dev/general-development/site-collection-app-catalog>
- Adds a list named **Apps for SharePoint** to root site
- Can limit scope of deployment to site collection rather than the entire tenant
- Site collection administrators can deploy SharePoint packages to the site collection app catalog
- Site collection app catalogs use the classic user experience

Deployment – Trust Dialog

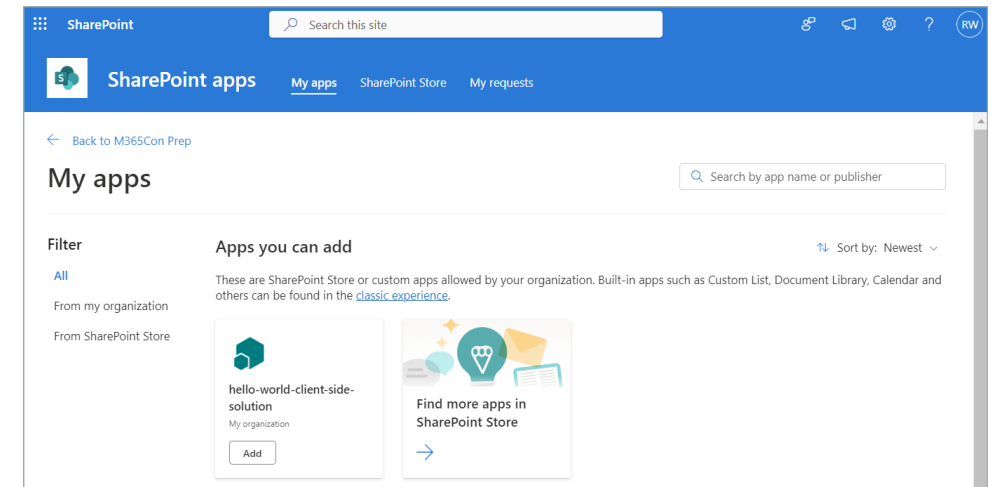
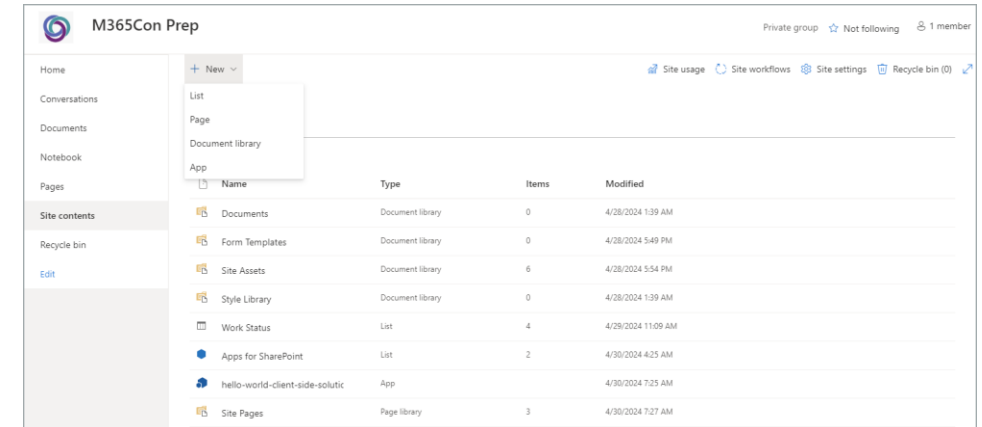
After uploading a SharePoint package to an app catalog, you will be shown a trust dialog or side panel.

- Select **Deploy** or **Enable app** to indicate you trust the package you just uploaded. This will complete the deployment process.



Installing SharePoint Framework Components

- Navigate to Site contents in the site where you wish to install the component(s)
- From the **New** menu, select **App**
- In the **My Apps** page, select the SharePoint package containing the component(s) you wish to install



Resources

SharePoint Framework Documentation

<https://learn.microsoft.com/en-us/sharepoint/dev/spfx/sharepoint-framework-overview>

Microsoft Learn Training - Extend Microsoft SharePoint

<https://learn.microsoft.com/en-us/training/paths/m365-sharepoint-associate/>

Microsoft 365 Platform Community (PnP)

<https://learn.microsoft.com/en-us/sharepoint/dev/community/community>