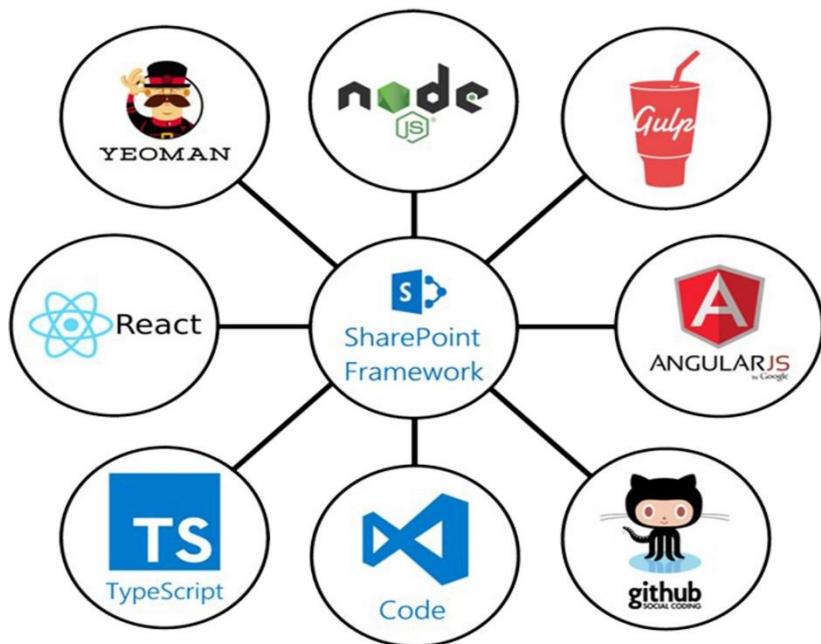


SharePoint Framework (SPFx)

A Developers Guide



Nanddeep Nachan

About the Author

Nanddeep Nachan is results-oriented Technology Architect with over 13 years' experience in Microsoft Technologies especially with .NET, MS Azure and SharePoint. He has been working with SharePoint since last 10 years and has exposure to SharePoint versions starting from SharePoint 2007 (MOSS).

He is CSM (Certified Scrum Master), as well as Microsoft Certified Professional with certifications in SharePoint, MS Azure, Office 365, and .Net. He is a C# Corner MVP, Author, and Speaker.

He is also a creative, technically sound photographer with experience in custom and specialized photography.

Active contributor to Office 365 Dev Patterns and Practices (<https://github.com/OfficeDev/PnP>) and C# Corner (<http://www.c-sharpcorner.com>)

What you will learn

This book is targeted for beginners to intermediate SharePoint developers who want to build SharePoint Framework (SPFx) solutions from scratch and get their hands dirty with practical examples. I have tried to author this book in an easy to follow way and help develop SharePoint Framework knowledge. As you go along and until such time you finished following the book, you will learn:

- Overview of SharePoint Framework (SPFx)
- Using JavaScript frameworks with SharePoint Framework
- Deploying SharePoint Framework
- SharePoint operations with SharePoint Framework



[/members/nanddeep-nachan](#)



[@NanddeepNachan](#)



[/in/nanddeepnachan](#)



[nanddeepnachan](#)



Nanddeep Nachan

CSM, MSP, C# Corner MVP

Author, Speaker

Table of Contents

Chapter 1: Overview of SharePoint Framework (SPFx)	10
Overview	10
Script Editor WebParts vs App Parts vs SPFx WebParts.....	10
Light weight Components / Tools Used.....	11
Flow of Client Side Web Part	12
Setup Developer Environment for SPFx.....	13
Summary	14
Chapter 2: Develop First Client Side Web Part	15
Overview	15
Create SPFx Solution	15
Open Solution in Code Editor.....	18
Run WebPart on local server	19
SharePoint Local Workbench.....	21
Summary	21
Chapter 3: Understand the Solution Structure	22
Overview	22
Solution Structure.....	22
Major Elements of Solution	23
Summary	26
Chapter 4: Avoid NPM Package Dependencies	27
Overview	27
Semantic Versioning	27
Lock Down the Package version.....	27
Freeze the entire tree of dependencies.....	28
Updating npm-shrinkwrap.json file	28
Summary	29
Chapter 5: Introduction to Web Part Property Pane	30
Overview	30
Property Pane Metadata.....	30
Create SPFx Solution	31
Property Pane Code	32
Add Our Properties	33

Test the Property Pane	37
Summary	38
Chapter 6: Deploy SPFx WebParts to Azure CDN	39
Overview	39
Configure MS Azure Storage Account.....	39
Configure BLOB Container	41
Enable Azure CDN for Storage Account	42
Configure SPFx Solution for Azure CDN	43
Deploy Package to SharePoint	45
Test the web part.....	45
Summary	46
Chapter 7: Deploy SPFx WebParts to Office 365 Public CDN.....	47
Overview	47
Configure CDN in Office 365 Tenant.....	47
Setup New Office 365 CDN in Tenant.....	49
Configure SPFx Solution for Azure CDN	50
Test the web part.....	51
Summary	52
Chapter 8: Deploy SPFx WebParts to SharePoint Library	53
Overview	53
Deployment to CDN	53
Create SharePoint Library as CDN.....	54
Configure SPFx Solution for SharePoint Library.....	54
Test the web part.....	56
Summary	57
Chapter 9: Integrating JQuery with SPFx WebParts	58
Overview	58
Create SPFx Solution	58
Configured Required Packages and Dependencies	59
Solution Changes.....	60
Test the web part.....	63
Summary	64
Chapter 10: CRUD operations using No Framework.....	65

Overview	65
Create SPFx Solution	65
Configure Property for List Name	66
Model for List Item.....	69
Add Controls to WebPart.....	69
Implement Create Operation.....	72
Implement Read Operation	73
Implement Update Operation	74
Implement Delete Operation.....	75
Test the WebPart	76
Troubleshooting.....	79
Summary.....	79
Chapter 11: CRUD operations using React JS	80
Overview	80
Brief about React JS	80
Create SPFx Solution	80
Configure Property for List Name	82
Model for List Item.....	86
Add Controls to WebPart.....	88
Implement Create Operation.....	91
Implement Read Operation	92
Implement Update Operation	93
Implement Delete Operation.....	94
Test the WebPart	96
Summary.....	98
Chapter 12: CRUD operations using Angular JS	99
Overview	99
Brief about Angular JS.....	99
Create SPFx Solution	99
Configure Angular JS	101
Configure Property for List Name	103
Build an Angular application	106
Add Controls to WebPart.....	114

Test the WebPart	115
Troubleshooting	118
Summary	118
Chapter 13: CRUD operations using Knockout JS	119
Overview	119
Brief about Knockout JS	119
Create SPFx Solution	119
Configure Property for List Name	121
Configure ViewModel	128
Add Controls to Knockout template	134
Test the WebPart	136
Troubleshooting	138
Summary	139
Chapter 14: CRUD operations using SP PnP JS	140
Overview	140
Brief about SP-PnP-JS	140
Create SPFx Solution	140
Configure sp-pnp-js	142
Configure Property for List Name	142
Model for List Item	144
Add Controls to WebPart	144
Import sp-pnp-js	146
Implement Create Operation	147
Implement Read Operation	147
Implement Update Operation	148
Implement Delete Operation	149
Test the WebPart	150
Summary	152
Chapter 15: Integrating Office UI Fabric	153
Overview	153
The UI Challenges	153
Brief information about Office UI Fabric	153
Office UI Fabric for SharePoint Framework	153

Create SPFx Solution	154
Office UI Fabric Components	155
How to use Office UI Fabric Components in SPFx WebPart	157
Implement Greet Message WebPart using Office UI Fabric	159
Test the WebPart	160
Summary	161
Chapter 16: Provision SharePoint Assets	162
Overview	162
SharePoint Assets.....	162
Create SPFx Solution	163
Add SharePoint Assets to Solution	165
Custom Schema	167
Package Assets as part of Solution	168
Deploy and Test	169
Summary	171
Chapter 17: Consume Microsoft Graph API using MSGraphClient	172
Overview	172
Brief about Microsoft Graph	172
Create SPFx Solution	172
Access MS Graph.....	174
Typings for MS Graph	174
Permissions	174
Configure Props.....	175
Configure State	176
Get User Details	176
Enable Targeted Release on your Tenant	177
Test the WebPart	178
API Management	178
Troubleshooting	179
Summary	179
Chapter 18: Consume Microsoft Graph API using AadHttpClient	180
Overview	180
Create SPFx Solution	180

Access MS Graph using AadHttpClient	181
Permissions	182
Configure Props.....	182
Configure State	183
Get User Details	183
Enable Targeted Release on your Tenant	184
Test the WebPart	185
API Management	186
Summary.....	186
Chapter 19: Logging	187
Overview	187
Logging Overview.....	187
Create SPFx Solution	187
Working with Logging API	189
Summary	190
Chapter 20: SPFx Extensions Overview.....	191
Overview	191
How will SharePoint Framework Extensions help?	191
Application Customizers	191
Field Customizers.....	191
Command Sets	191
Update Generators for SharePoint Framework Extensions.....	191
Useful commands	192
Create an SharePoint Framework Extensions Project	193
Visual Studio Extension for SharePoint Framework	193
How does Visual Studio extension work?	194
Summary	194
Chapter 21: Application Customizer Overview.....	195
Overview	195
Brief about Application Customizer	195
Page Placeholders.....	195
Create SPFx Solution	195
Solution Structure	197

Implement Application Customizer	198
Test the extension.....	201
Summary.....	203
Chapter 22: Field Customizer Overview	204
Overview	204
Brief about Field Customizer.....	204
Create SPFx Solution	204
Solution Structure.....	206
Implement Field Customizer.....	207
Debug Field Customizer	207
Update the Solution for Field changes	210
Apply Field Customization	212
Summary.....	213
Chapter 23: ListView Command Set Overview	214
Overview	214
Brief about Field Customizer.....	214
Create SPFx Solution	214
Solution Structure.....	216
Implement Field Customizer.....	217
Debug Field Customizer	218
Update the Solution for Field changes	221
Apply List View Command Set Customization	224
Summary.....	225
Recommendations	226
Resources	226
The Final Word.....	226

Chapter 1: Overview of SharePoint Framework (SPFx)

Overview

The customization in modern sites is supported using SharePoint Framework (SPFx). SPFx is an open and connected platform. SPFx is page and web part model. It can be entirely developed using client-side languages and open source tooling. SPFx provides easy integration with SharePoint data.

Key Features of SharePoint Framework

1. No iframes, runs within the context of the user browser and connection in the browser
2. Faster rendering on the browser as all controls are rendered in normal DOM
3. Controls are responsive
4. Runs in the context of current user
5. Gives controls to access the lifecycle of the SharePoint Framework web part (component) (Init, render, load, serialize, deserialize, configuration changes and many more)
6. No dependency on underlying Framework. You can use any framework like React, Angular, Knockout and more
7. Open source Development tools are used (npm, TypeScript, Yeoman, webpack and Gulp)
8. Can be added on both classic pages and modern pages
9. Safe and Secure, need tenant access to deploy/make changes to the SPFx webpart
10. Controlled visibility, we can decide who can view this webpart in the App Catalog of the site contents
11. You can leverage your earlier knowledge of CSOM, as the data models are not changed and is completely transferable
12. SPFx web parts can be used with classic or modern sites in SharePoint
13. Supports mobile views of SharePoint Online sites

Script Editor Web Parts vs App Parts vs SPFx Web Parts

Script Editor Web Parts

- The obvious choice of developers for customizing DOM on classic SharePoint sites.
- Script can be edited by any users easily
- Cannot be added to “NoScript” sites

App Parts

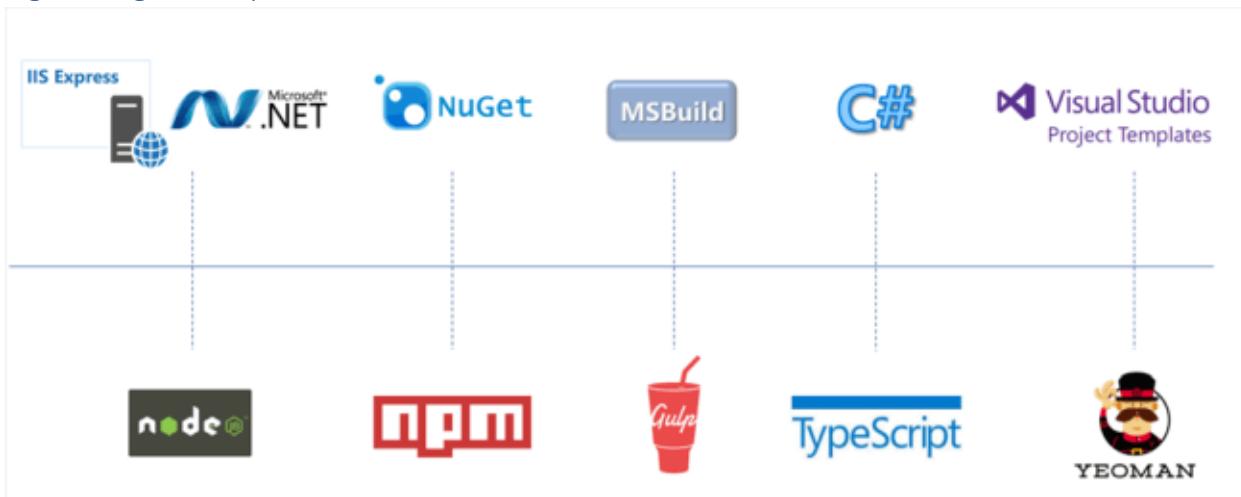
- Developed using Add-in model
- Uses iframe
- Cannot access DOM of SharePoint page
- Development and deployment is bit complicated

SPFx Web Parts

- Client side web parts, leverages modern JavaScript frameworks

- Can be used with classic SharePoint pages
- Provides modern experience, responsiveness out of the box
- Free, open source tool chain

Light-weight Components / Tools Used



Node.js

- Open source JavaScript runtime
- Used to build and run the applications which is equivalent to the .Net
- SPFx supports latest LTS (Long Term Support) version

NPM Packages

- Stands for Node Package Manager
- Installs modules and its dependencies (equivalent to NuGet Package)
- Packages can be installed globally (-g switch) or locally
- Installed packages go inside node_modules folder

Gulp

- Automates SPFx development and deployment tasks
- Bundle and minify JavaScript and CSS files
- Run tools to call the bundling and minification tasks before each build
- Compiles LESS or SASS files to CSS
- Compiles TypeScript files to JavaScript
- Equivalent to MSBuild in Microsoft World
- Compiles, bundle and copies files to deployment folder for packaging

Yeoman

- Relies on NPM and Gulp
- Scaffolding tool for Modern web apps
- Used as SPFx solution generator and builds required project structure
- 'yo' is the command line utility for creation of projects

TypeScript

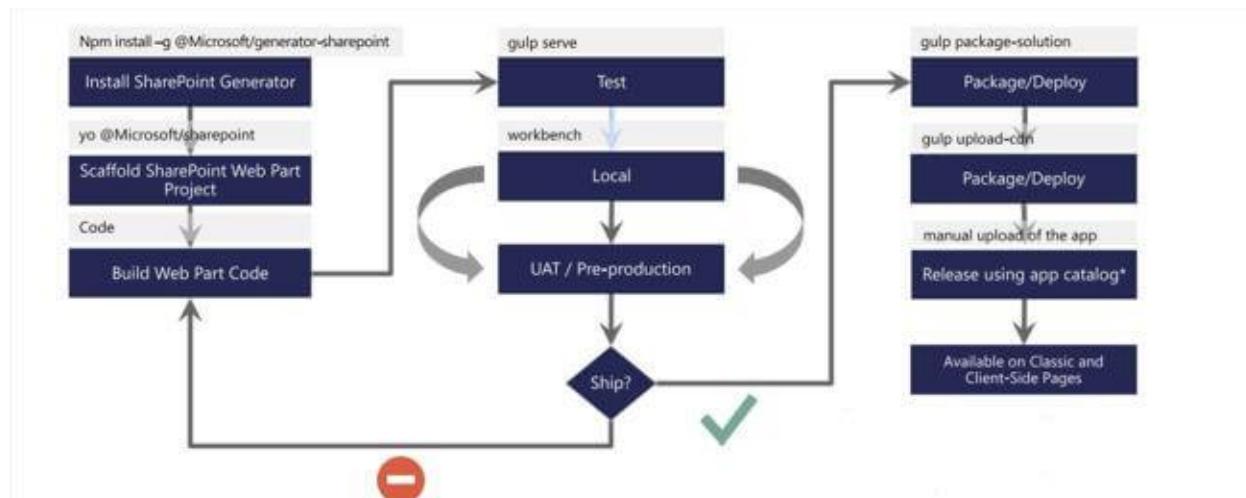
- Strongly typed language
- Adds compile time syntax and type checking for JavaScript
- Help to build the Applications which will be then compiled to clean JS code

Visual Studio Code

- Interface for working with SPFx solutions
- Fast and light weight IDE
- Can work on Windows, Mac OS, and Linux

Flow of Client Side Web Part

The below diagram depicts the flow of client side web part from installation, development, and deployment.



Setup Developer Environment for SPFx

Follow the below set of commands to get your developer environment ready for SPFx.

Install Node JS

- Install latest LTS version from <https://nodejs.org>
- If you already have NodeJS installed, check the version

```
node -v
```

Install Yeoman and gulp

Run the below command (to install globally)

```
npm install -g yo gulp
```

Install Yeoman SharePoint Generator

Run the below command

```
npm install -g @microsoft/generator-sharepoint
```

Note:

You can create batch files to install all packages. Please find the attached one for reference.

Install Code Editor

Install any of below code editors:

1. Visual Studio Code (<https://code.visualstudio.com>)
2. Atom (<https://atom.io>)
3. Webstorm (<https://www.jetbrains.com/webstorm>)
4. Visual Studio SPFx Project Template
(<https://marketplace.visualstudio.com/items?itemName=SharePointPnP.SPFxProjectTemplate>)

Updating NPM packages

Yo, Gulp, Yeoman SharePoint Generator gets installed as NPM packages. Use the below commands to check and update them.

To update NPM,

```
npm i -g npm
```

To check outdated packages,

```
npm outdated --global
```

This command will report packages that need updates. Use the below command to update the reported packages.

```
npm update -g <package-name>
```

Summary

In this chapter, we had a walkthrough on high level understanding of SharePoint Framework. We also discussed key features of SharePoint Framework. We also discussed the tool comparison which will be used in SharePoint Framework development and along with that we had a discussion on high level flow of Client Side Web Parts development.

Chapter 2: Develop First Client Side Web Part

Overview

Client Side Web Parts developed using SharePoint Framework (SPFx) are the future of Modern SharePoint. Client Side Web Parts are developed using modern UI standards, modern JavaScript tools and libraries. They run inside context of SharePoint page and are responsive in nature.

Features of SPFx Client Side Web Parts:

- Lightweight - Developed using JavaScript libraries and HTML
- Responsive - Renders on any device
- Environments - Work on both SharePoint Online and OnPremise (SharePoint 2016 onwards)

In this chapter, we will learn practically how to build a simple SPFx based Client Side Web Part. Follow the instructions from previous chapter to setup your development environment first.

Create SPFx Solution

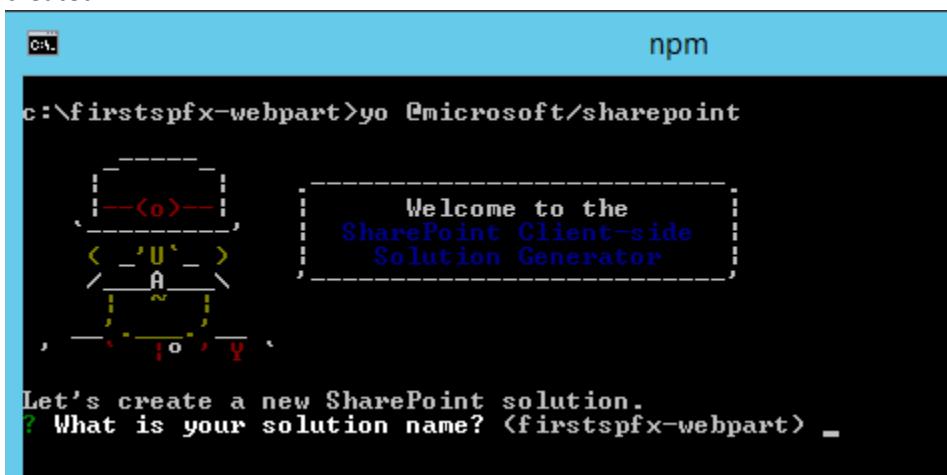
Follow below steps to create first SPFx solution.

1. Create a directory for SPFx solution

```
md firstspfx-webpart
```
2. Navigate to above created directory

```
Cd firstspfx-webpart
```
3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```
4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



The screenshot shows a terminal window with the following text:

```
c:\firstspfx-webpart>yo @microsoft/sharepoint
Welcome to the
SharePoint Client-side
Solution Generator
Let's create a new SharePoint solution.
? What is your solution name? (firstspfx-webpart) _
```

Solution Name: Hit enter to have default name (firstspfx-webpart in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client web part, i.e., SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side web part or an extension. Choose web part option.

Selected choice: Web Part

Web part name: Hit enter to select the default name or type in any other name.

Selected choice: Hit enter (default name)

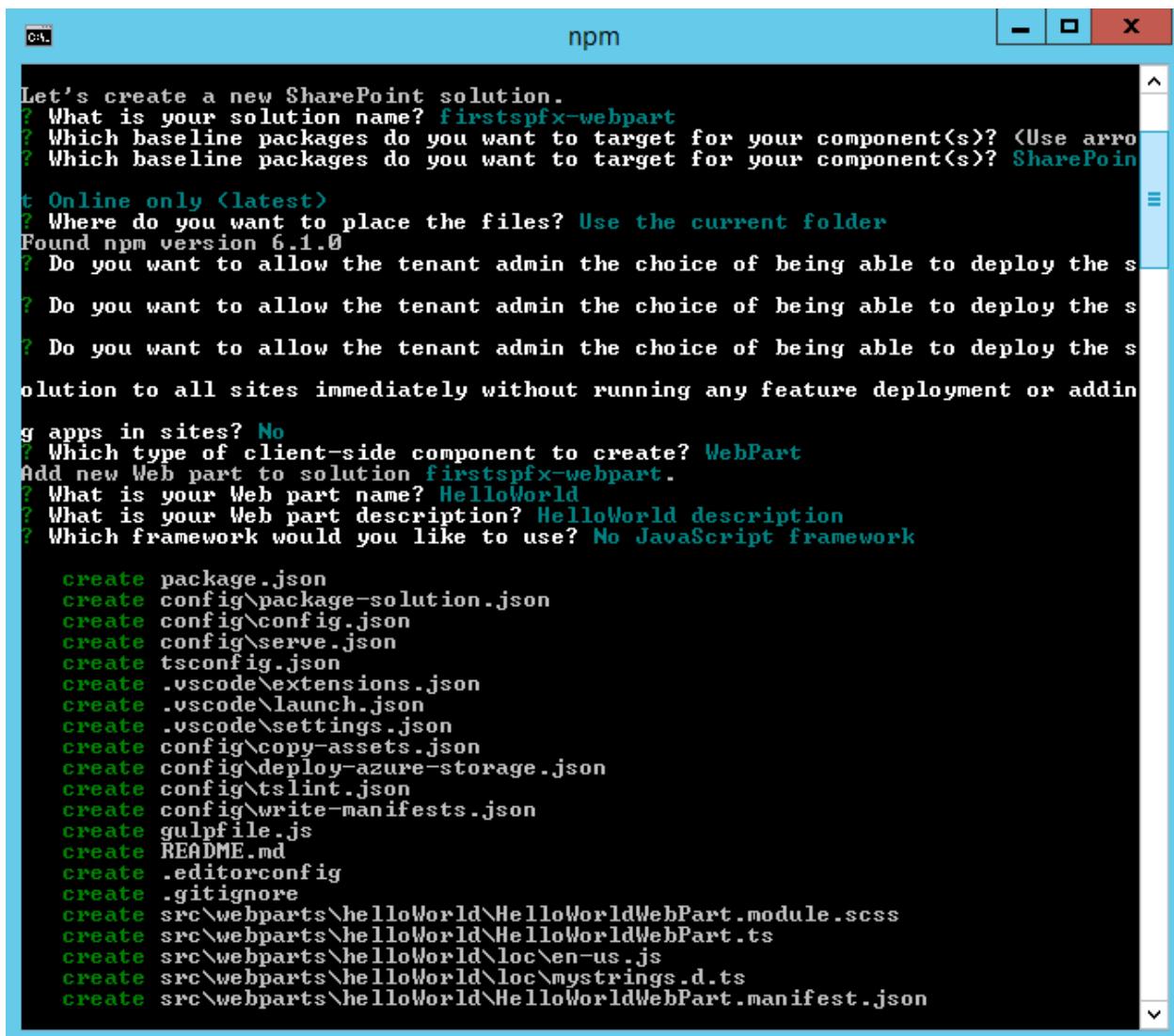
Web part description: Hit enter to select the default description or type in any other value.

Selected choice: Hit enter (default description)

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)

Selected choice: No JavaScript Framework

5. Yeoman generator will start the scaffolding process to create the solution and will install required dependencies. This process will take significant amount of time.



```
Let's create a new SharePoint solution.
? What is your solution name? firstspfx-webpart
? Which baseline packages do you want to target for your component(s)? <Use arrow keys to select>
? Which baseline packages do you want to target for your component(s)? SharePoint Online only <latest>
? Where do you want to place the files? Use the current folder
Found npm version 6.1.0
? Do you want to allow the tenant admin the choice of being able to deploy the solution to all sites immediately without running any feature deployment or adding apps in sites? No
? Which type of client-side component to create? WebPart
Add new Web part to solution firstspfx-webpart.
? What is your Web part name? HelloWorld
? What is your Web part description? HelloWorld description
? Which framework would you like to use? No JavaScript framework

create package.json
create config\package-solution.json
create config\config.json
create config\serve.json
create tsconfig.json
create .vscode\extensions.json
create .vscode\launch.json
create .vscode\settings.json
create config\copy-assets.json
create config\deploy-azure-storage.json
create config\tslint.json
create config\write-manifests.json
create gulpfile.js
create README.md
create .editorconfig
create .gitignore
create src\webparts\helloWorld\HelloWorldWebPart.module.scss
create src\webparts\helloWorld\HelloWorldWebPart.ts
create src\webparts\helloWorld\loc\en-us.js
create src\webparts\helloWorld\loc\mystrings.d.ts
create src\webparts\helloWorld\HelloWorldWebPart.manifest.json
```

6. Once scaffolding is completed, Yeoman will show the below message

Administrator: Command Prompt

```
> spawn-sync@1.0.15 postinstall c:\firstspfx-webpart\node_modules\spawn-sync
> node postinstall

> uglifyjs-webpack-plugin@0.4.6 postinstall c:\firstspfx-webpart\node_modules\uglifyjs-webpack-plugin
> node lib/post_install.js

> node-sass@4.9.0 postinstall c:\firstspfx-webpart\node_modules\node-sass
> node scripts/build.js

Binary found at c:\firstspfx-webpart\node_modules\node-sass\vendor\win32-x64-57\
binding.node
Testing binary
Binary is fine
npm notice created a lockfile as package-lock.json. You should commit this file.

npm WARN ajv-keywords@3.2.0 requires a peer of ajv@^6.0.0 but none is installed.
You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
)

added 1835 packages from 1747 contributors and audited 752645 packages in 511.36
7s
  run 'npm audit fix' to fix them, or 'npm audit' for details

          _=+#####!
 ####/   <##|<@>           Congratulations!
 ####/   #####|  > | Solution firstspfx-webpart is created.
 ####/   /###|  <@> | Run gulp serve to play with it!
 #######|  #
 ####/   /##|<@>
 #######|  *
 ***=+#####!
```

c:\firstspfx-webpart>

Open Solution in Code Editor

Any of below code editor can be used to open the SPFx client side solution.

- Visual Studio Code (<https://code.visualstudio.com/>)
- Atom (<https://atom.io/>)
- Webstorm (<https://www.jetbrains.com/webstorm>)

Install any of above code editor. Being in the solution folder in command prompt, type below command to open the solution in code editor.

```
code .
```

Run the Web Part on a local server

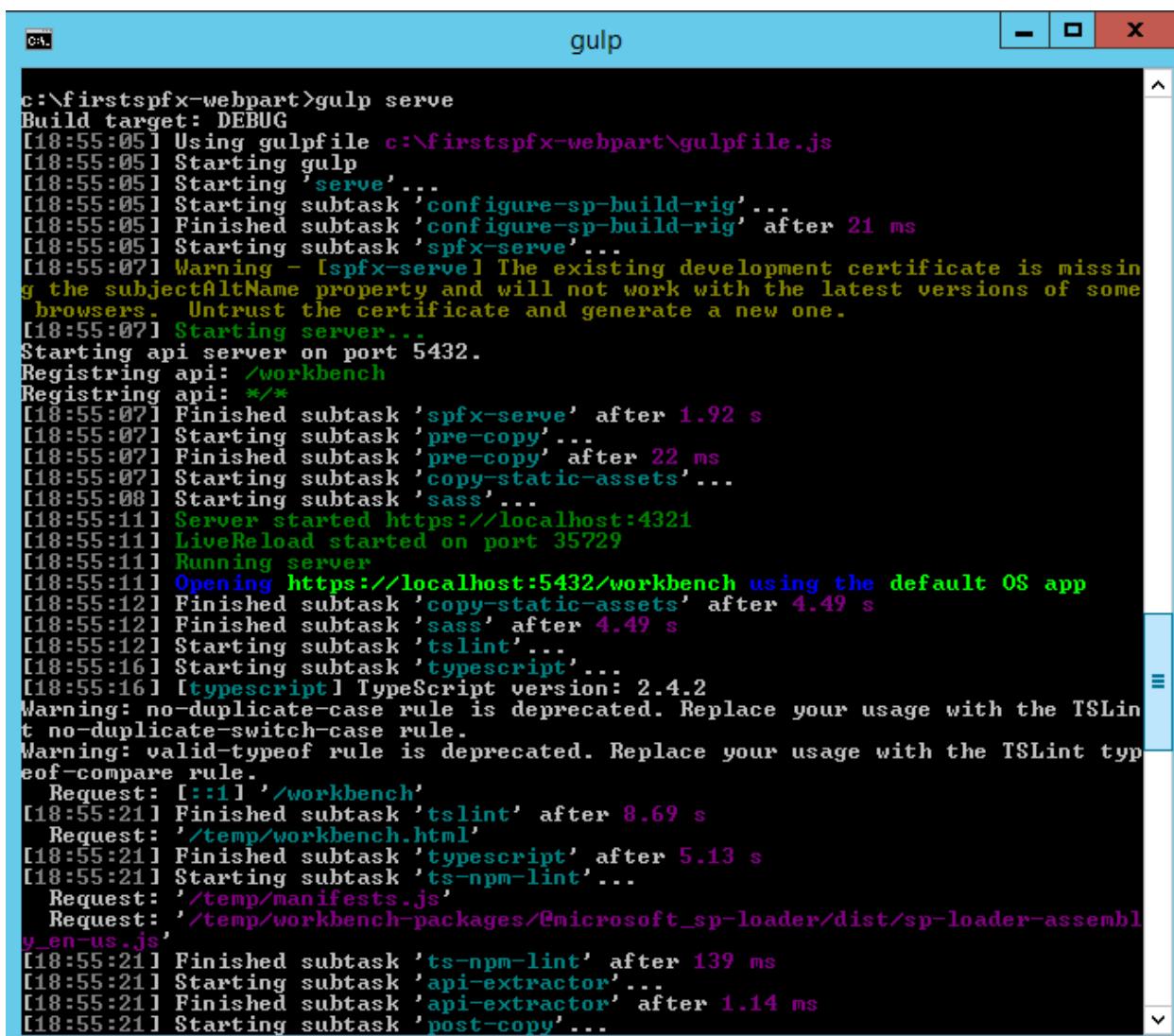
As client side toolchain uses HTTPS endpoint, we need to install the developer certificate that comes with SPFx toolchain.

To install developer certificate, run below command from the command prompt

```
gulp trust-dev-cert
```

Run below command to preview the web part

```
gulp serve
```

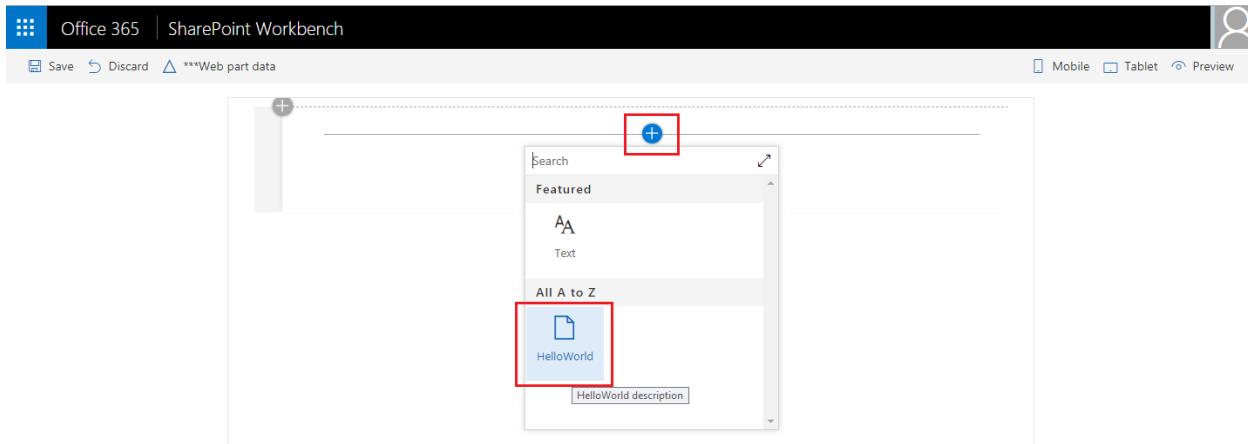


```
c:\firstspfx-webpart>gulp serve
Build target: DEBUG
[18:55:05] Using gulpfile c:\firstspfx-webpart\gulpfile.js
[18:55:05] Starting gulp
[18:55:05] Starting 'serve'...
[18:55:05] Starting subtask 'configure-sp-build-rig'...
[18:55:05] Finished subtask 'configure-sp-build-rig' after 21 ms
[18:55:05] Starting subtask 'spfx-serve'...
[18:55:07] Warning - [spfx-serve] The existing development certificate is missing the subjectAltName property and will not work with the latest versions of some browsers. Untrust the certificate and generate a new one.
[18:55:07] Starting server...
Starting api server on port 5432.
Registering api: /workbench
Registering api: /**
[18:55:07] Finished subtask 'spfx-serve' after 1.92 s
[18:55:07] Starting subtask 'pre-copy'...
[18:55:07] Finished subtask 'pre-copy' after 22 ms
[18:55:07] Starting subtask 'copy-static-assets'...
[18:55:08] Starting subtask 'sass'...
[18:55:11] Server started https://localhost:4321
[18:55:11] LiveReload started on port 35729
[18:55:11] Running server
[18:55:11] Opening https://localhost:5432/workbench using the default OS app
[18:55:12] Finished subtask 'copy-static-assets' after 4.49 s
[18:55:12] Finished subtask 'sass' after 4.49 s
[18:55:12] Starting subtask 'tslint'...
[18:55:16] Starting subtask 'typescript'...
[18:55:16] [typescript] TypeScript version: 2.4.2
Warning: no-duplicate-case rule is deprecated. Replace your usage with the TSLint no-duplicate-switch-case rule.
Warning: valid-typeof rule is deprecated. Replace your usage with the TSLint typeof-compare rule.
  Request: [::1] '/workbench'
[18:55:21] Finished subtask 'tslint' after 8.69 s
  Request: '/temp/workbench.html'
[18:55:21] Finished subtask 'typescript' after 5.13 s
[18:55:21] Starting subtask 'ts-npm-lint'...
  Request: '/temp/manifests.js'
  Request: '/temp/workbench-packages/@microsoft_sp-loader/dist/sp-loader-assembly_en-us.js'
[18:55:21] Finished subtask 'ts-npm-lint' after 139 ms
[18:55:21] Starting subtask 'api-extractor'...
[18:55:21] Finished subtask 'api-extractor' after 1.14 ms
[18:55:21] Starting subtask 'post-copy'...
```

This command runs series of gulp tasks that performs below:

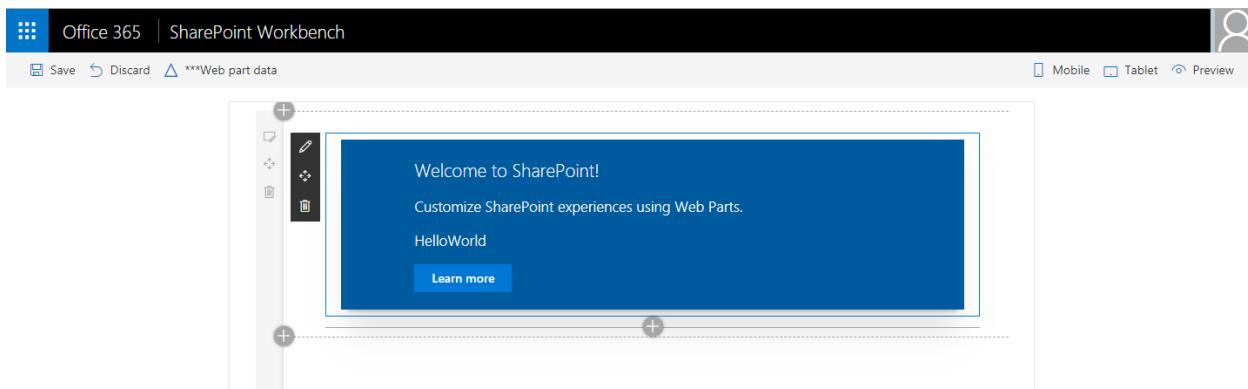
- Minify and bundle JavaScript and CSS files
- Compiles SASS (Syntactically Awesome Style Sheets) to CSS
- Compiles TypeScript to JavaScript

Upon successful compilation and running the gulp tasks, it will open up SharePoint local workbench.



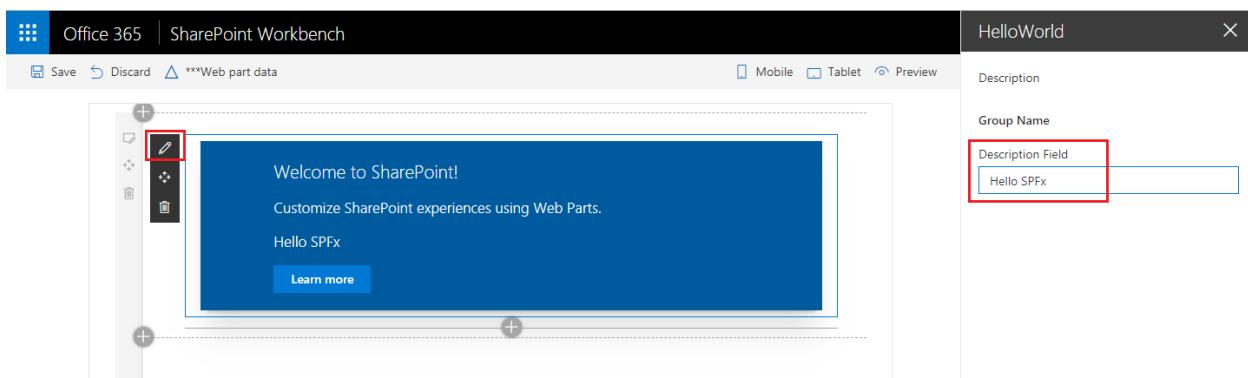
Click Add (+) icon

Select the web part to add it on page.



Congratulations! The first client side web part is ready.

Click Edit icon, modify description from properties pane and it will be reflected on client side web part.



SharePoint Local Workbench

SharePoint Local Workbench is HTML page which helps to preview and test client side web parts without deploying to SharePoint. It gets served locally usually on the URL -

<https://localhost:4321/temp/workbench.html>

Summary

SPFx client side web parts are based on modern JavaScript libraries, which can be created using Yeoman generator. We can choose the JavaScript library (No JavaScript Framework, React, and Knockout) to build the client side web part. They can be tested locally on SharePoint Workbench.

Chapter 3: Understand the Solution Structure

Overview

SharePoint Framework (SPFx) client side web parts are lightweight in nature. They can be developed using open source tools such as Node.JS, NPM, Yeoman generators and can be opened in code editors of our choice (Visual Studio Code, Atom, Webstorm). Node Package Manager (NPM) helps to install modules and its dependencies. Yeoman generator carries out the scaffolding and builds the required project structure.

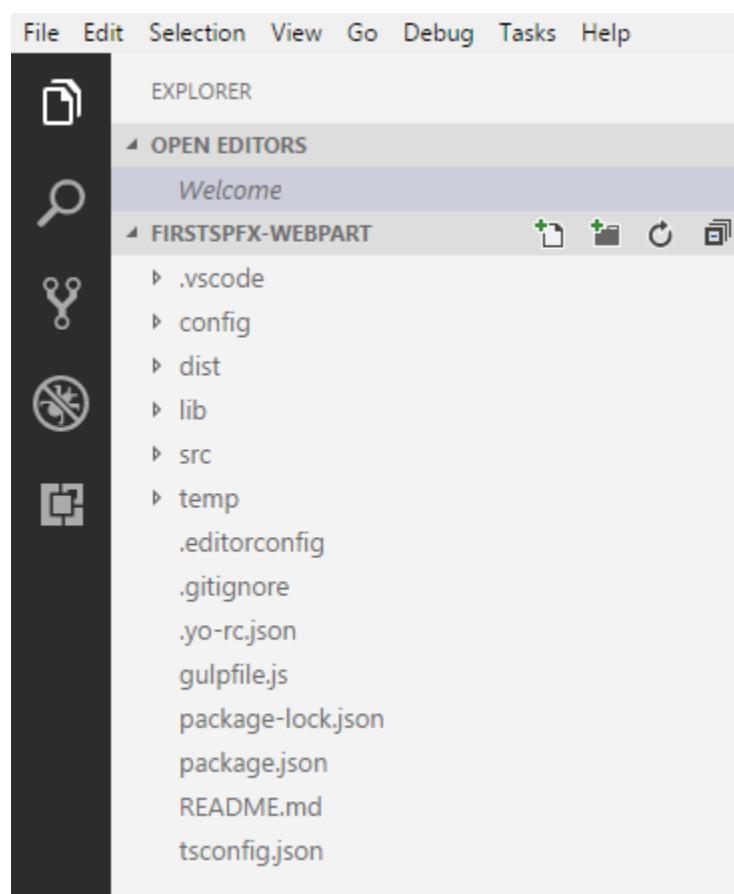
In this chapter, we will have a close look at the solution structure of SPFx client side web part and understand major elements of it.

Solution Structure

In the previous chapter, we developed our first SPFx client side web part.

In the command prompt, run below command to open the solution in code editor of your choice.

```
code .
```



The solution structure consists of various folders and few configuration files at the root. The primary language used is TypeScript (which is a superset of JavaScript). The code mostly uses classes, modules and interfaces written in TypeScript.

Major Elements of Solution

Web Part Class (**HelloWorldWebPart.ts**)

This defines the entry point of the web part. In our solution, we have only one web part class - **HelloWorldWebPart.ts** located at **src\webparts\helloWorld** folder. The web part class extends **BaseClientSideWebPart**. Each client side web part should extend from **BaseClientSideWebPart**, which provides basic functioning for web part.

Property Type Interface (**IHelloWorldWebPartProps**)

The interface resides in the same web part file. The web part class accepts the property type of interface. This interface is used to define our own custom properties for web part. We can add any of below type of property.

- Button
- Checkbox
- Choice group
- Dropdown
- Horizontal rule
- Label
- Link
- Slider
- Textbox
- Multi-line Textbox
- Toggle
- Custom

The properties can be used in web part as below

```
<p class="${ styles.description }">${ escape(this.properties.description) }</p>
```

WebPart Manifest

HelloWorldWebPart.manifest.json holds the metadata of web part such as display name, description, icon, version, id.

```
{  
  "$schema": "https://developer.microsoft.com/json-schemas/spfx/client-side-  
  web-part-manifest.schema.json",  
  "id": "a925e124-f0f8-4a26-a797-d384046cf5a7",  
  "alias": "HelloWorldWebPart",  
  "componentType": "WebPart",  
  
  // The "*" signifies that the version should be taken from the package.json  
  "version": "*",
  "manifestVersion": 2,  
  
  // If true, the component can only be installed on sites where Custom Script  
  // is allowed.  
  // Components that allow authors to embed arbitrary script code should set  
  // this to true.  
  // https://support.office.com/en-us/article/Turn-scripting-capabilities-on-  
  // or-off-1f2c515f-5d7e-448a-9fd7-835da935584f  
  "requiresCustomScript": false,  
  
  "preconfiguredEntries": [  
    {  
      "groupId": "5c03119e-3074-46fd-976b-c60198311f70", // Other  
      "group": { "default": "Other" },  
      "title": { "default": "HelloWorld" },  
      "description": { "default": "HelloWorld description" },  
      "officeFabricIconFontName": "Page",  
      "properties": {  
        "description": "HelloWorld"  
      }  
    }  
  ]  
}
```

Config.json

The configuration file contains the bundling information, the components used in the solution and entry point of the solution.

This file also records the external references (for e.g. jQuery) and its dependencies. Also, references to localization resources are stored in this file.

```
{
  "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/config.2.0.schema.json",
  "version": "2.0",
  "bundles": {
    "hello-world-web-part": {
      "components": [
        {
          "entrypoint": "./lib/webparts/helloWorld/HelloWorldWebPart.js",
          "manifest": "./src/webparts/helloWorld/HelloWorldWebPart.manifest.json"
        }
      ]
    },
    "externals": {},
    "localizedResources": {
      "HelloWorldWebPartStrings": "lib/webparts/helloWorld/loc/{locale}.js"
    }
}
```

deploy-azure-storage.json

This file is used while deploying the client-side web part to Azure CDN. This file contains Azure storage account details.

```
{
  "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/deploy-azure-storage.schema.json",
  "workingDir": "./temp/deploy/",
  "account": "<!-- STORAGE ACCOUNT NAME -->",
  "container": "spfxcontainer",
  "accessKey": "<!--ACCESS KEY -->"
}
```

package-solution.json

This file contains solution path configurations.

```
{  
  "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/package-  
  solution.schema.json",  
  "solution": {  
    "name": "spfx-helloworld-client-side-solution",  
    "id": "41c6825d-8ef2-45e1-b544-5f0d64a480e6",  
    "version": "1.0.0.0",  
    "includeClientSideAssets": false,  
    "skipFeatureDeployment": true  
  },  
  "paths": {  
    "zippedPackage": "solution/spfx-helloworld.sppkg"  
  }  
}
```

gulpfile.js

Defines gulp tasks to run.

package.json

Defines JavaScript library dependencies.

tsconfig.json

Defines settings for TypeScript compilation

Summary

Yeoman generators help to generate SPFx client-side web part solution. The solution has a predefined structure. Each folder and file has its own significance. It is important to understand the role of each file for better development.

Chapter 4: Avoid NPM Package Dependencies

Overview

NPM (Node Package Manager) is a package manager for JavaScript as like Nuget for managed code of .Net framework. NPM is used to consume third party libraries in SharePoint Framework (SPFx) solutions. All of the required packages like jQuery, React, Angular, Office Fabric UI, etc. can be added to the solution using NPM. The packages are dependent on each other. Change of semantic versioning (semver) of one package can break the dependency build toolchain.

In this chapter, we will understand the package dependencies and how it can be avoided.

Semantic Versioning

Each package has a version associated with it which is denoted by 3 numbers (major.minor.patch)

- **Major** represents substantial changes to package that may break the existing code.
- **Minor** represents non-breaking changes.
- **Patch** represents no interface changes.

The dependencies in SPFx solutions are recorded in package.json in following ways:

Caret dependency: Any version up to (but not including) is accepted

Example: ^2.3.1 means greater than or equal to 2.3.1 but less than 3.0.0

Tilde dependency: It is similar to Caret dependency, but the range is narrower

Example: ~2.3.1 means greater than or equal to 2.3.1 but less than 2.4.0

Exact dependency: Constrained to a specific version

Example: 2.3.1 means 2.3.1 only

Lock Down the Package version

When a package is added to SPFx solution using “npm install” command, the package is downloaded and installed to the solution. It can be found under “node_modules” folder and entry of it is recorded in package.json file. Adding --save flag to “npm install” command ensure that the dependency is recorded to package.json. This means, whenever any developer runs “npm install” or “npm update” on the SPFx solution, NPM will download package of appropriate version. It is always recommended to specify --save or --save-exact while running “npm install” command to record and track the dependencies.

Failing to specify --save flag will not record the entry in package.json and can cause code to fail because of missing packages. TypeScript compiler might throw an error “Cannot find module”.

“node_modules” folder contains tree of dependencies for packages. First level of dependencies are stored in child “node_modules” folders. Because of this reason, “node_modules” folder can grow bigger in size and deep in hierarchy.

“`--save-exact`” will avoid caret or tilde dependencies only at first level, which makes difficult to get the exact build unless we check-in entire “`node_modules`” folder to source control (which is not recommended and feasible solution).

Freeze the entire tree of dependencies

“`npm shrinkwrap`” command will help you to freeze the entire tree of dependencies. It creates a JSON file called “`npm-shrinkwrap.json`” which records exact version of each package used in the solution. It is important in the production environment that same versions of packages get installed as that used during development.

```
npm-shrinkwrap.json ✘
1
2   "name": "angular-1-hello-world",
3   "version": "0.0.1",
4   "lockfileVersion": 1,
5   "requires": true,
6   "dependencies": {
7     "@microsoft/api-extractor": {
8       "version": "4.2.6",
9       "resolved": "https://registry.npmjs.org/@microsoft/api-extractor/-/api-extractor-4.2.6.tgz",
10      "integrity": "sha1-LEQcgMR1AO/LG49ICH+gNkAGC5U=",
11      "dev": true,
12      "requires": {
13        "@microsoft/node-core-library": "0.3.16",
14        "@microsoft/ts-command-line": "2.2.4",
15        "@types/fs-extra": "0.0.37",
16        "@types/node": "6.0.88",
17        "@types/z-schema": "3.16.31",
18        "colors": "1.1.2",
19        "fs-extra": "0.26.7",
20        "jju": "1.3.0",
21        "lodash": "4.15.0",
22        "typescript": "2.4.2",
23        "z-schema": "3.18.4"
24      },
25      "dependencies": {
26        "@types/node": {
27          "version": "6.0.88",
28          "resolved": "https://registry.npmjs.org/@types/node/-/node-6.0.88.tgz",
29          "integrity": "sha512-bYDPZTX0/s1aihdjLuAgogUAT5M+Tp0WChEMea2p0yOcfn5bu3k6cJb9cp6nw268XeSNIIGGr+4+/8V5K6BGzLQ==",
30          "dev": true
31        }
32      }
33    }
34  }
35 }
```

Because of using “`npm shrinkwrap`”, if “`npm install`” or “`npm update`” commands are used against the solution containing “`npm-shrinkwrap.json`” will restore the exact version of packages and avoids code break issues for other developers.

Updating `npm-shrinkwrap.json` file

`npm-shrinkwrap.json` file freezes the version of package. However, we might have to update any single version of the package in future.

To update single package version run below command:

```
npm update <package_name>
```

This will only update a single package. Once done re-run “npm shrinkwrap” command to update npm-shrinkwrap.json file.

To find out the outdated packages in the solution, run below command:

```
npm outdated
```

Summary

It is recommended to run “npm shrinkwrap” every time against the SPFx solution before you release the version of your solution. It will record exact version of package dependencies. Also ensure to check-in the “npm-shrinkwrap.json” file to source control. This will allow to rebuild your solution on any machine even though newer versions of used packages are released in the future.

Chapter 5: Introduction to Web Part Property Pane

Overview

In classic SharePoint, web parts can be configured using the properties, generally referred as Web Part properties. In the SPFx world they are referred as property panes. The property panes allows to control behavior and appearance of web part. In this chapter we will deep dive into configuring the property panes.

Property Pane Metadata

Property pane has below metadata:

Pages

Allows to separate complex interactions and divide them across one or multiple pages. Pages has sub-metadata as Header and Groups.

Header

Defines title of the property pane.

Groups

Groups together various property fields. Below is the list of out of box supported property fields.

#	Property field	SPFx Typed Object
1	Label	PropertyPaneLabel
2	Textbox	PropertyPaneTextField
3	Multiple lines of text	PropertyPaneTextField
4	Link	PropertyPaneLink
5	Dropdown	PropertyPaneDropdown
6	Checkbox	PropertyPaneCheckbox
7	Choice	PropertyPaneChoiceGroup
8	Toggle	PropertyPaneToggle
9	Slider	PropertyPaneSlider
10	Button	PropertyPaneButton
11	Horizontal Rule	PropertyPaneHorizontalRule
12	Custom	Our own implementation using combination of above

Create SPFx Solution

1. Create a directory for SPFx solution.

```
md spfx-ooopropertypanes
```

2. Navigate to the above-created directory.

```
cd spfx-ooopropertypanes
```

3. Run Yeoman SharePoint Generator to create the solution.

```
yo @microsoft/sharepoint
```

4. Provide below values in the Yeoman generator wizard

Solution Name: Hit enter to have default name (spfx-ooopropertypanes in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client web part, i.e., SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension. Choose web part option.

Selected choice: Web Part

Web part name: Hit enter to select the default name or type in any other name.

Selected choice: SPFXPropertyPane

Web part description: Hit enter to select the default description or type in any other value.

Selected choice: Hit enter (default description)

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)

Selected choice: No JavaScript Framework

5. Once Yeoman finishes generating the solution, on command prompt type “code .” to open the solution in code editor.

Property Pane Code

Open file SpfxPropertyPaneWebPart.ts located at \src\webparts\spfxPropertyPane

The method **getPropertyPaneConfiguration()** contains the configuration to build the property pane.

```
protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
    return {
        pages: [
            {
                header: {
                    description: strings.PropertyPaneDescription
                },
                groups: [
                    {
                        groupName: strings.BasicGroupName,
                        groupFields: [
                            PropertyPaneTextField('description', {
                                label: strings.DescriptionFieldLabel
                            })
                        ]
                    }
                ]
            }
        ];
    };
}
```

The default class “SpfxPropertyPaneWebPart” accepts property type of interface “ISpfxPropertyPaneWebPartProps”, which by default has description property of type string. The property can be used in render() method as below.

```
 ${escape(this.properties.description)}
```

```

export interface ISpfxPropertyPaneWebPartProps {
  description: string;
}

export default class SpfxPropertyPaneWebPart extends BaseClientSideWebPart<ISpfxPropertyPaneWebPartProps> {

  public render(): void {
    this.domElement.innerHTML = `
      <div class="${ styles.spfxPropertyPane }">
        <div class="${ styles.container }">
          <div class="${ styles.row }">
            <div class="${ styles.column }">
              <span class="${ styles.title }">Welcome to SharePoint!</span>
              <p class="${ styles.subTitle }">Customize SharePoint experiences using Web Parts.</p>
              <p class="${ styles.description }">${escape(this.properties.description)}</p>
              <a href="https://aka.ms/spfx" class="${ styles.button }">
                <span class="${ styles.label }">Learn more</span>
              </a>
            </div>
          </div>
        </div>
      </div>`;
  }
}

```

The import section has the property types defined.

```

import {
  BaseClientSideWebPart,
  IPropertyPaneConfiguration,
  PropertyPaneTextField
} from '@microsoft/sp-webpart-base';

```

Add Our Properties

To add new properties

1. Import corresponding typed objects first

```

import {
  BaseClientSideWebPart,
  IPropertyPaneConfiguration,
  PropertyPaneTextField, // Textbox
  PropertyPaneCheckbox, // Checkbox
  PropertyPaneLabel, // Label
  PropertyPaneLink, // Link
  PropertyPaneSlider, // Slider
  PropertyPaneToggle, // Toggle
  PropertyPaneDropdown // Dropdown
} from '@microsoft/sp-webpart-base';

```

2. Add new properties. Map fields to typed objects.

```
export interface ISpfxPropertyPaneWebPartProps {
    name: string;
    description: string;
    Slider:string;
    Toggle:string;
    dropdownm:string;
    checkbox:string;
    URL:string;
    textbox:string;
}
```

3. Add new property pane fields and map to respective typed objects in getPropertyPaneConfiguration method.

```
protected textBoxValidationMethod(value: string): string {
    if (value.length < 10) { return "Name should be at least 10 characters!"; }
    else { return ""; }
}

protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
    return {
        pages: [
            { //Page 1
                header: {
                    description: "Page 1 - Name and Description"
                },
                groups: [
                    {
                        groupName: "Group one",
                        groupFields: [
                            PropertyPaneTextField('name', {
                                label: "Name",
                                multiline: false,
                                resizable: false,
                                onGetErrorMessage: this.textBoxValidationMethod,
                                errorMessage: "This is the error message",
                                deferredValidationTime: 5000,
                                placeholder: "Please enter name", "description": "Name property field"
                            }),
                            PropertyPaneTextField('description', {
                                label: "Description",
                                multiline: true,
                                resizable: true,
                                placeholder: "Please enter description", "description": "Description property field"
                            })
                        ]
                    }
                ]
            }
        ];
    };
}
```

4. We can add multiple pages as below.

```
{ //Page 2
  header: {
    description: "Page 2 - Slider and Dropdown"
  },
  groups: [
    {
      groupName: "Group one",
      groupFields: [
        PropertyPaneSlider('Slider', {
          label:'Slider',min:1,max:10
        }),
        PropertyPaneToggle('Toggle', {
          label: 'Slider'
        })
      ]
    },
    {
      groupName: "Group Two",
      groupFields: [
        PropertyPaneDropdown('dropdown', {
          label:'Drop Down',
          options: [
            { key: 'Item1', text: 'Item 1' },
            { key: 'Item2', text: 'Item 2' },
            { key: 'Item3', text: 'Item 3' }
          ]
        }),
        PropertyPaneCheckbox('checkbox',
          { text: 'Yes/No'})
      ]
    }
  ],
}
```

5. Setup one more page for remaining fields.

```
[ { //Page 3
  header: {
    description: "Page 3 - URL and Label"
  },
  groups: [
    {
      groupName: "Group One",
      groupFields: [
        PropertyPaneLink('URL',
        { text:"Microsoft", href:'http://www.microsoft.com',target:'_blank'}),
        PropertyPaneLabel('label',
        { text:'Please enter designation',required:true}),
        PropertyPaneTextField('textbox',{})
      ]
    }
  ]
}
```

6. Modify render method to show selected field values.

```
export default class SpfxPropertyPaneWebPart extends BaseClientSideWebPart<ISpfxPropertyPaneWebPartProps> {

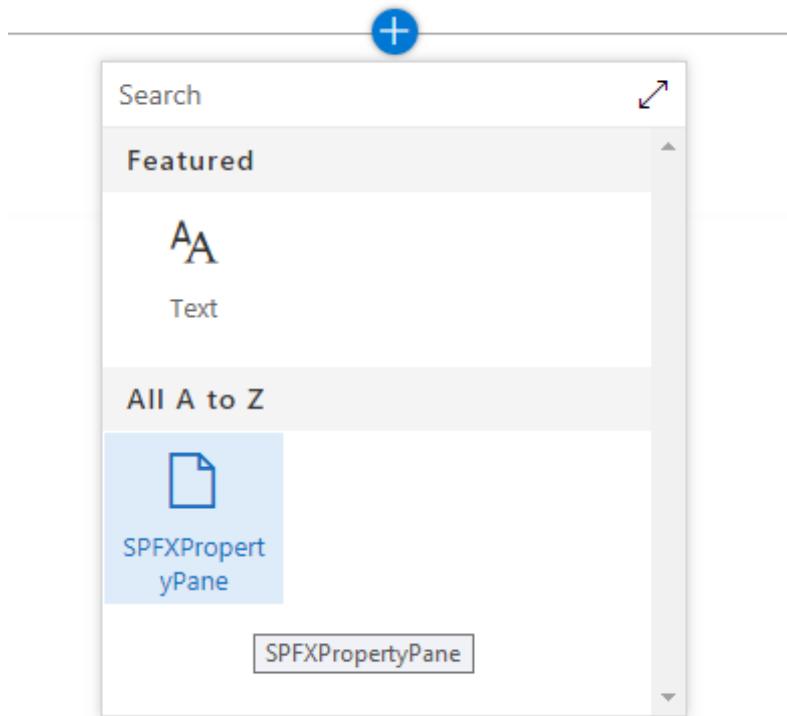
  public render(): void {
    this.domElement.innerHTML = `
      <div class="${ styles.spfxPropertyPane }">
        <div class="${ styles.container }">
          <div class="${ styles.row }">
            <div class="${ styles.column }">
              <span class="${ styles.title }">Welcome to SharePoint!</span>
              <p class="${ styles.subTitle }">Customize SharePoint experiences using Web Parts.</p>
              <p class="${ styles.description }">Name: ${escape(this.properties.name)}</p>
              <p class="${ styles.description }">Description: ${escape(this.properties.description)}</p>

              <p class="${ styles.description }">Slider: ${escape(this.properties.Slider)}</p>
              <p class="${ styles.description }">Toggle: ${escape(this.properties.Toggle)}</p>
              <p class="${ styles.description }">dropdown: ${escape(this.properties.dropdown)}</p>
              <p class="${ styles.description }">checkbox: ${escape(this.properties.checkbox)}</p>

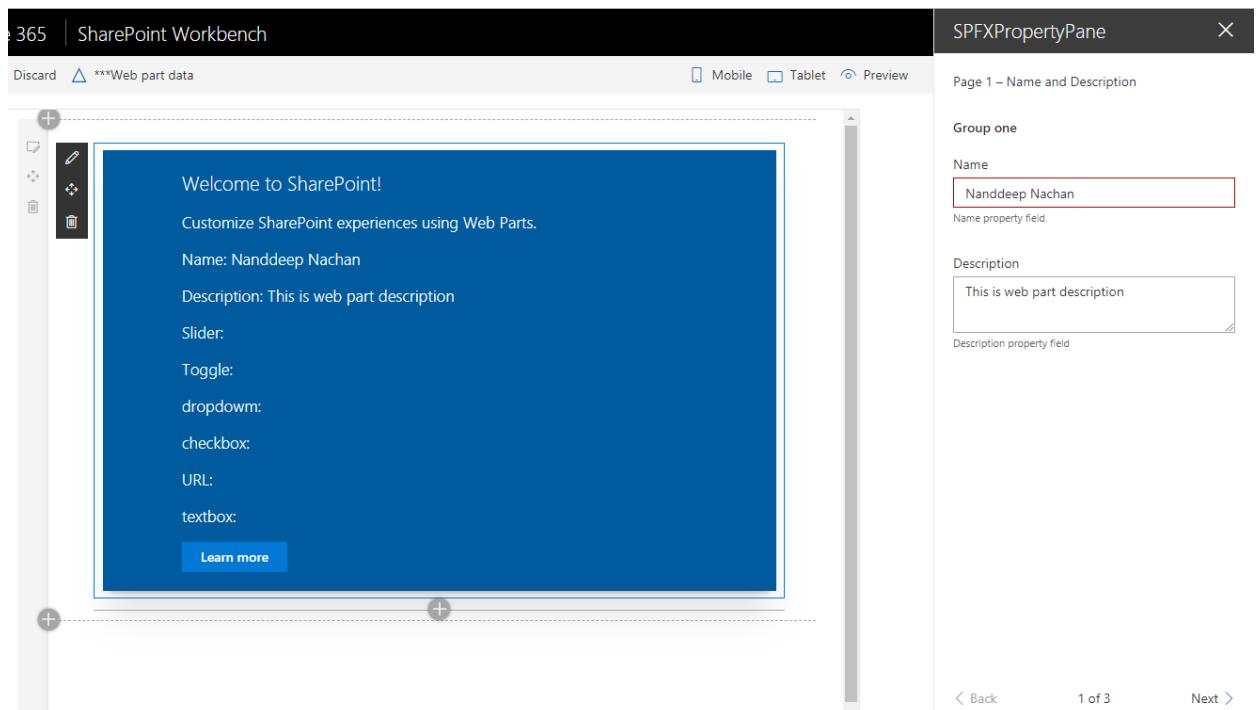
              <p class="${ styles.description }">URL: ${escape(this.properties.URL)}</p>
              <p class="${ styles.description }">textbox: ${escape(this.properties.textbox)}</p>
              <a href="https://aka.ms/spfx" class="${ styles.button }">
                <span class="${ styles.label }">Learn more</span>
              </a>
            </div>
          </div>
        </div>
      `;
  }
}
```

Test the Property Pane

1. On the command prompt type “gulp serve”.
2. In SharePoint workbench, add your web part.

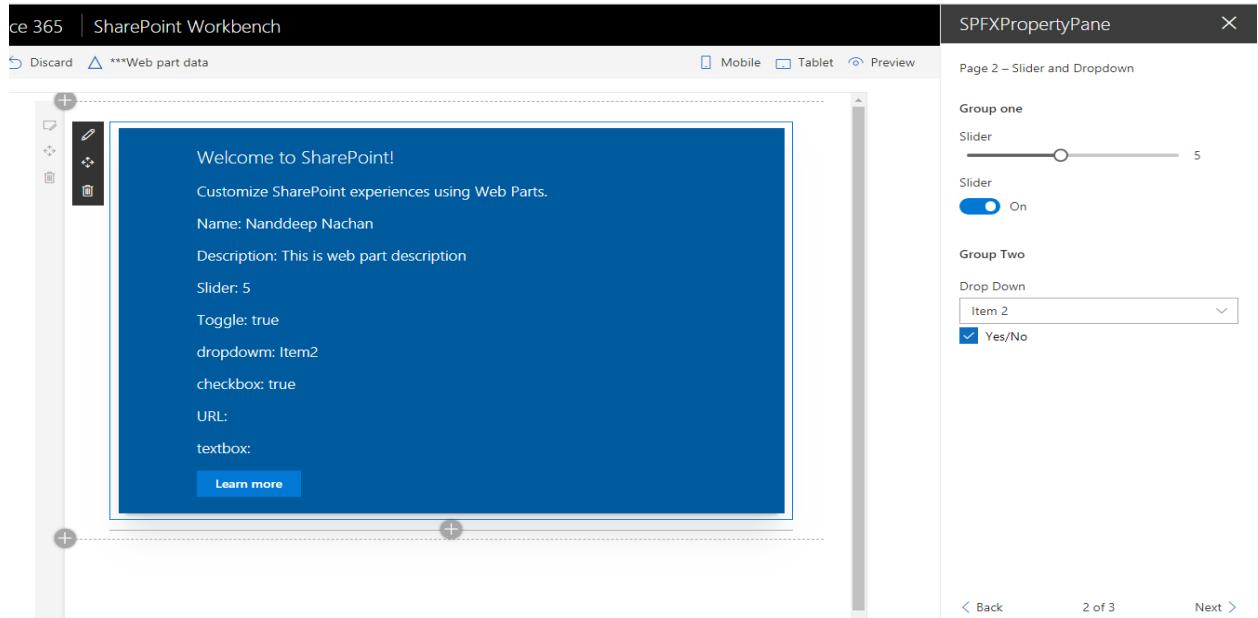


3. Edit web part to modify property fields.



The screenshot shows the SharePoint Workbench with the 'SPFXPropertyPane' web part selected. The left pane displays the web part's content, which includes a welcome message and various input fields like Name, Description, Slider, Toggle, dropdown, checkbox, URL, and textbox. The right pane shows the 'Name' field set to 'Nanddeep Nachan' and the 'Description' field set to 'This is web part description'.

4. Click Next to see next page of fields



Welcome to SharePoint!

Customize SharePoint experiences using Web Parts.

Name: Nanddeep Nachan

Description: This is web part description

Slider: 5

Toggle: true

dropdown: Item2

checkbox: true

URL:

textbox:

[Learn more](#)

Group one

Slider

Slider: 5

Slider: On

Group Two

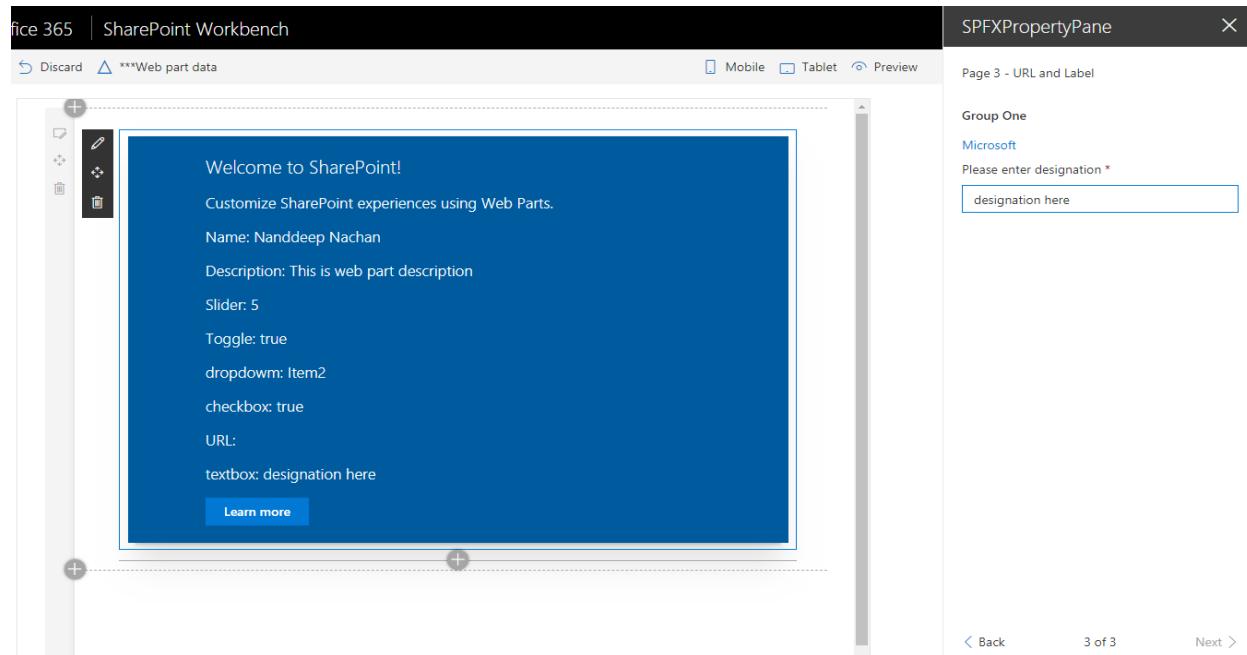
Drop Down

Item 2

✓ Yes/No

Back 2 of 3 Next >

5. Click Next to see next page of fields



Welcome to SharePoint!

Customize SharePoint experiences using Web Parts.

Name: Nanddeep Nachan

Description: This is web part description

Slider: 5

Toggle: true

dropdown: Item2

checkbox: true

URL:

textbox: designation here

[Learn more](#)

Group One

Microsoft

Please enter designation *

designation here

Back 3 of 3 Next >

Summary

Property panes allows easy configuration of SPFx client side web part. We can use predefined typed objects to create properties for web part. Properties can be grouped by pages.

Chapter 6: Deploy SPFx Web Parts to Azure CDN

Overview

In the chapters so far, we developed SharePoint Framework client side webparts and tested it on local SharePoint workbench. The local SharePoint workbench allows to quickly test the SPFx web parts. The Node.js provides the hosting platform to run on local environment. However in the real life scenarios we will have to deploy these SPFx web parts to some hosting environments from where it can be served to SharePoint.

Majorly the SPFx web parts are deployed on either of below CDN (Content Delivery Network) options:

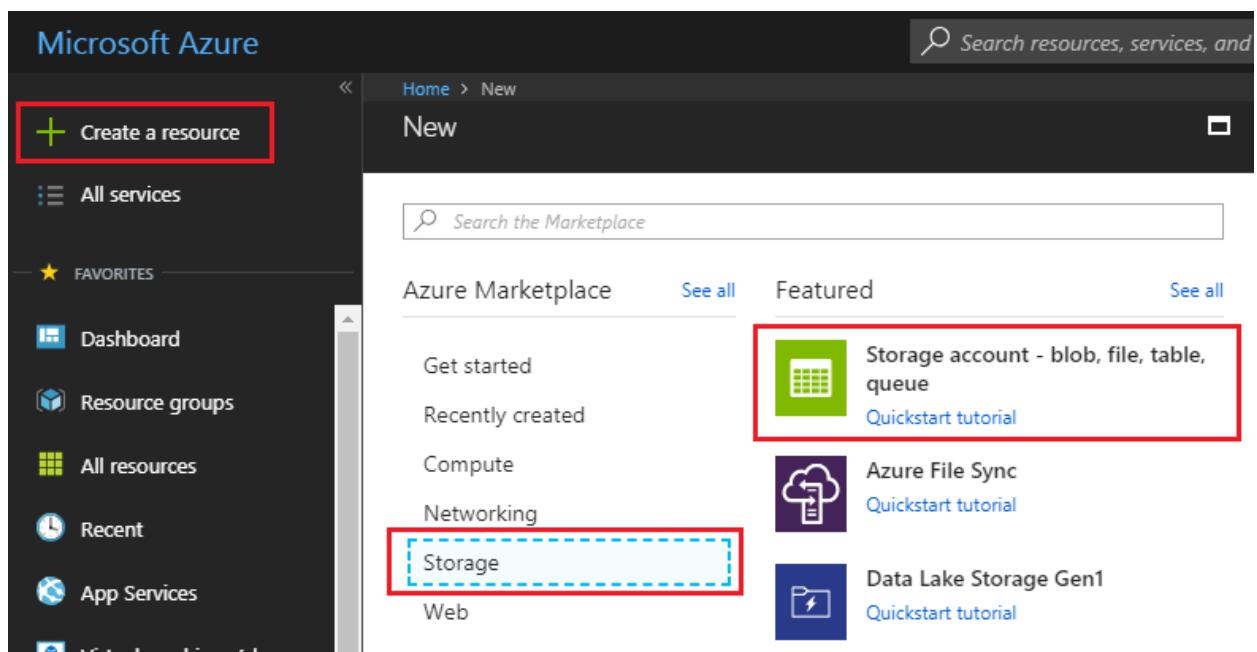
1. Azure CDN
2. Office 365 Public CDN
3. SharePoint Library in your tenant

In this chapter, we will have a look at how SPFx web parts can be deployed to Azure CDN.

Configure MS Azure Storage Account

Follow below steps to configure MS Azure storage account and integrate it with CDN

1. Login to MS Azure Portal (<https://portal.azure.com>)
2. Click “Create a resource”
3. Click “Storage account - blob, file, table, queue”.



4. Fill out the form

Create storage account □

The cost of your storage account depends on the usage and the options you choose below.
[Learn more](#)

* Name i
spfxdeploy ✓
.core.windows.net

Deployment model i
Resource manager Classic

Account kind i
Storage (general purpose v1) ▼

* Location
West India ▼

Replication i
Read-access geo-redundant storage (R... ▼

Performance i
Standard Premium

* Secure transfer required i
Disabled Enabled

* Subscription
Visual Studio Enterprise ▼

* Resource group
 Create new Use existing
SPFx ▼

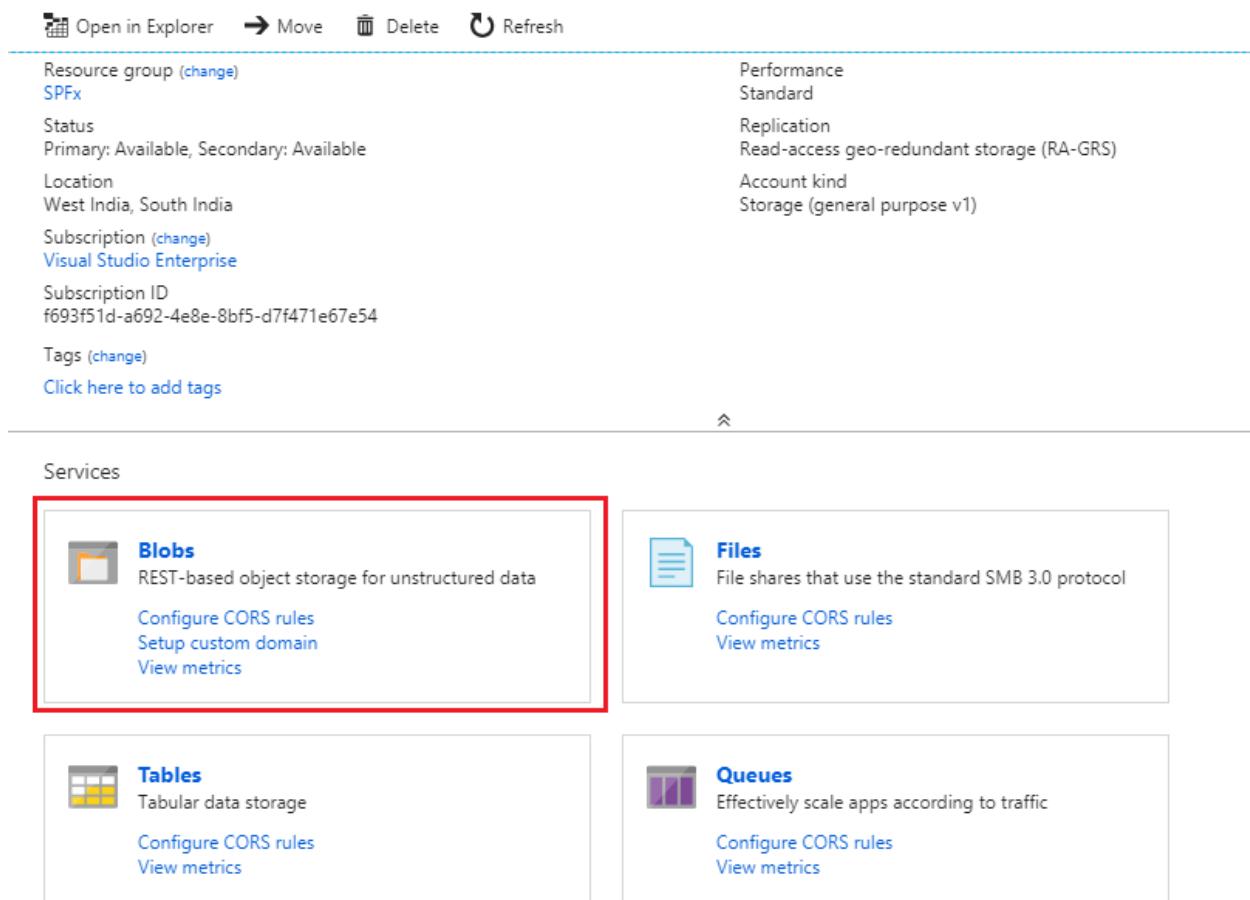
Create [Automation options](#)

5. Click **Create**

6. The storage account endpoint can be referred by URI –
<http://<StorageAccountName>.blob.core.windows.net>

Configure BLOB Container

1. Select storage account from dashboard
2. Click “Blobs”



The screenshot shows the Azure Storage Account dashboard for a resource group named 'SPFx'. The 'Blobs' service is highlighted with a red box. Other services like 'Files', 'Tables', and 'Queues' are also listed.

Resource group (change)
SPFx

Status
Primary: Available, Secondary: Available

Location
West India, South India

Subscription (change)
Visual Studio Enterprise

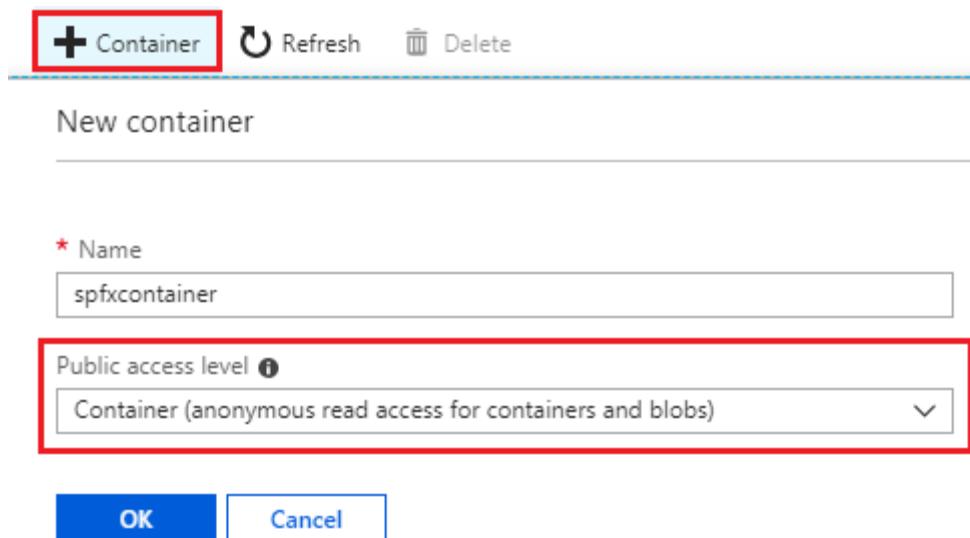
Subscription ID
f693f51d-a692-4e8e-8bf5-d7f471e67e54

Tags (change)
[Click here to add tags](#)

Services

- Blobs**
REST-based object storage for unstructured data
[Configure CORS rules](#)
[Setup custom domain](#)
[View metrics](#)
- Files**
File shares that use the standard SMB 3.0 protocol
[Configure CORS rules](#)
[View metrics](#)
- Tables**
Tabular data storage
[Configure CORS rules](#)
[View metrics](#)
- Queues**
Effectively scale apps according to traffic
[Configure CORS rules](#)
[View metrics](#)

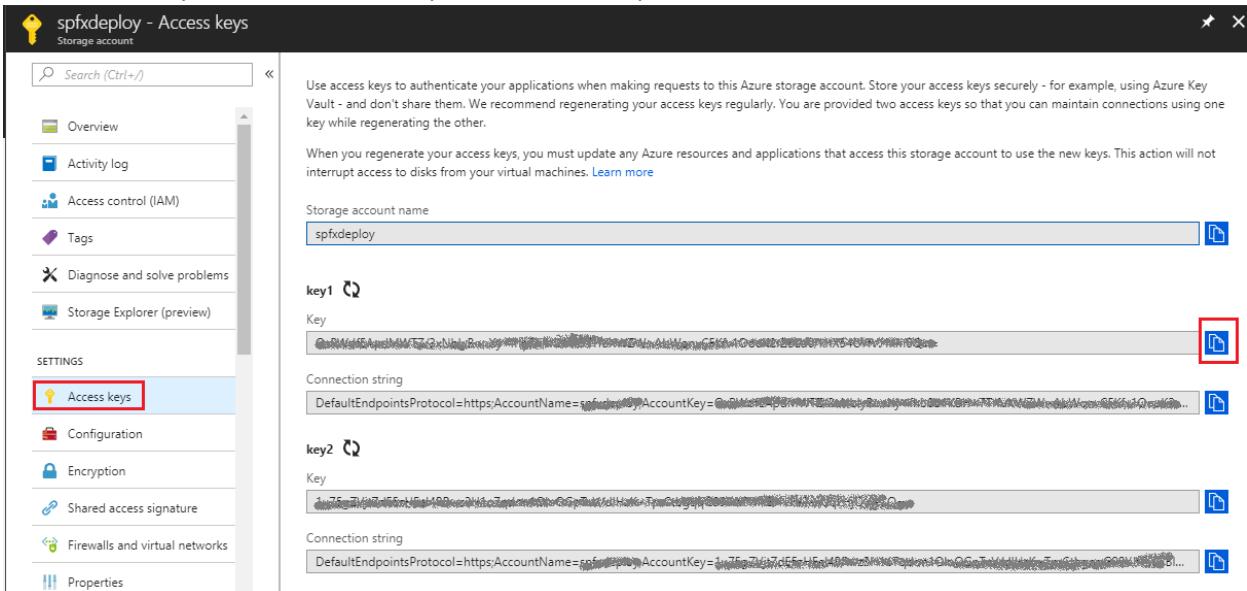
3. Click “+ Container”



The dialog box for creating a new container has several fields and buttons:

- + Container** (button, highlighted with a red box)
- Refresh** (button)
- Delete** (button)
- New container** (label)
- * Name** (label)
- Public access level** (label)
- OK** (button)
- Cancel** (button)

4. Click Access keys and note down any of the access key

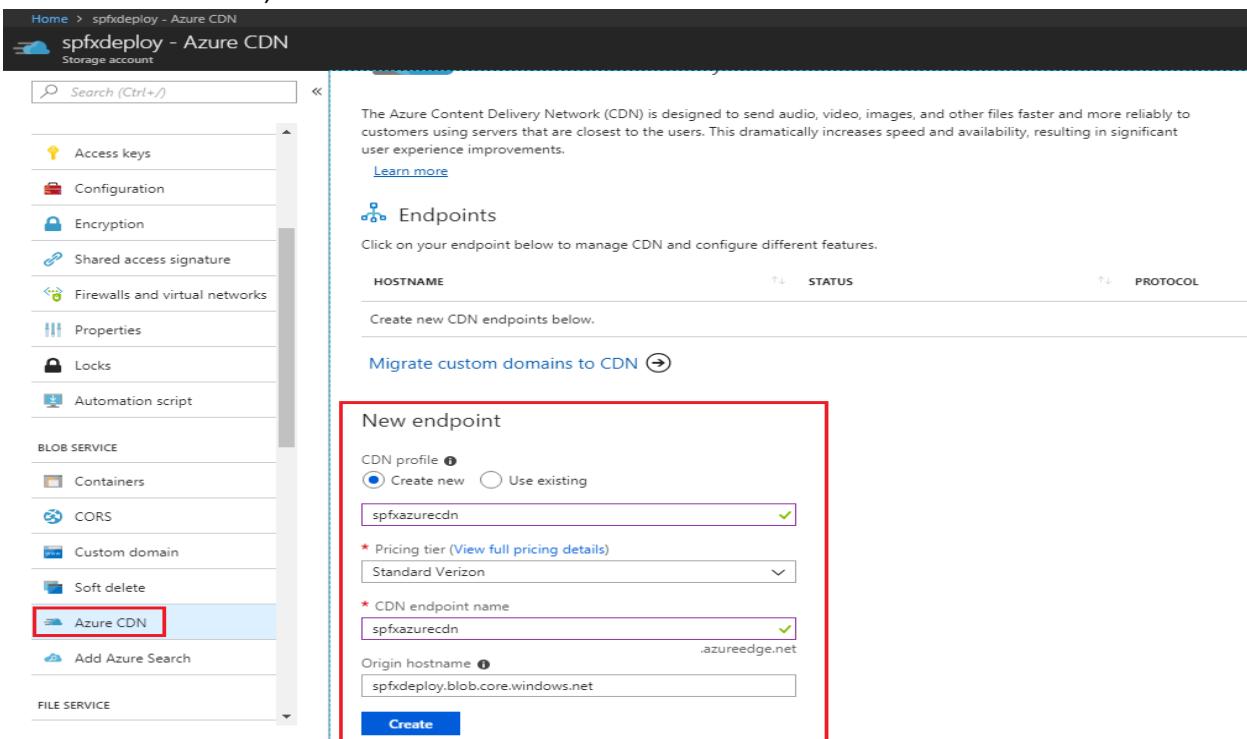


The screenshot shows the 'Access keys' section of the Azure Storage account 'spfxdeploy'. The left sidebar lists various settings like Overview, Activity log, and Diagnose and solve problems. The 'Access keys' tab is selected and highlighted with a red box. It displays two sets of keys: 'key1' and 'key2'. The 'key1' value is heavily redacted with black text, while the 'key2' value is partially visible. Below each key is a 'Connection string' field.

Enable Azure CDN for Storage Account

We will enable CDN for the above created storage account.

1. Select storage account from dashboard
2. Under “BLOB Service”, select “Azure CDN”



The screenshot shows the 'Azure CDN' configuration page for the 'spfxdeploy' storage account. The left sidebar has sections for Home, Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Storage Explorer (preview), Configuration, Encryption, Shared access signature, Firewalls and virtual networks, Properties, Locks, Automation script, BLOB SERVICE (Containers, CORS, Custom domain, Soft delete, Azure CDN), FILE SERVICE (Add Azure Search), and Metrics. The 'Azure CDN' option under 'BLOB SERVICE' is highlighted with a red box. On the right, there's a summary of the Azure Content Delivery Network (CDN) and an 'Endpoints' section. A 'New endpoint' modal window is open, also highlighted with a red box. It contains fields for 'CDN profile' (radio buttons for 'Create new' and 'Use existing', with 'spfxazurecdn' selected), 'Pricing tier' (Standard Verizon), 'CDN endpoint name' (spfxazurecdn.azureedge.net), and 'Origin hostname' (spfxdeploy.blob.core.windows.net). A 'Create' button is at the bottom of the modal.

3. Click **Create**
4. The endpoint will appear in the endpoint list

Azure Content Delivery Network

The Azure Content Delivery Network (CDN) is designed to send audio, video, images, and other files faster and more reliably to customers using servers that are closest to the users. This dramatically increases speed and availability, resulting in significant user experience improvements.

[Learn more](#)

Endpoints

Click on your endpoint below to manage CDN and configure different features.

HOSTNAME	STATUS	PROTOCOL
spfxazurecdn.azureedge.net	Running	HTTP, HTTPS

[Migrate custom domains to CDN](#) 

5. The BLOB container can be accessed from url:
<http://<EndpointName>.azureedge.net/<myPublicContainer>/<BlobName>>

Configure SPFx Solution for Azure CDN

Update package details

1. Open command prompt
2. In the command prompt, navigate to SPFx solution folder
3. Type “code .” to open the solution in code editor of your choice
4. Open **package-solution.json** file from **config** folder. This file takes care of solution packaging.
5. Set **includeClientSideAsserts** value as **false**. The client side assets will not be packaged inside final package (sppkg file) because these will be hosted on external CDN.



```

EXPLORER                                package-solution.json x
-----+-----+
  OPEN EDITORS      package-solution.json config
  SPFX-HELLOWORLD
    .vscode
      config
        config.json
        copy-assets.json
        deploy-azure-storage.json
        package-solution.json
        serve.json
        tslint.json
        write-manifests.json
-----+-----+
1  {
2    "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/package-solution.schema.json",
3    "solution": {
4      "name": "spfx-helloworld-client-side-solution",
5      "id": "41c6825d-8ef2-45e1-b544-5f0d64a480e6",
6      "version": "1.0.0.0",
7      "includeClientSideAssets": false,
8      "skipFeatureDeployment": true
9    },
10   "paths": {
11     "zippedPackage": "solution/spfx-helloworld.sppkg"
12   }
13 }
14

```

Update Azure storage account details to Solution

1. Open **deploy-azure-storage.json** from **config** folder to specify the Azure storage account details
2. Update values as below
 - account: storage account name
 - container: BLOB container
 - accessKey: storage account access key (primary or secondary)



```

EXPLORER
OPEN EDITORS
deploy-azure-storage.json con...
SPFX-HELLOWORLD
  .vscode
    config
      config.json
      copy-assets.json
      deploy-azure-storage.json
        <-- Red Box
      package-solution.json
      serve.json
      tslint.json
      write-manifests.json

```

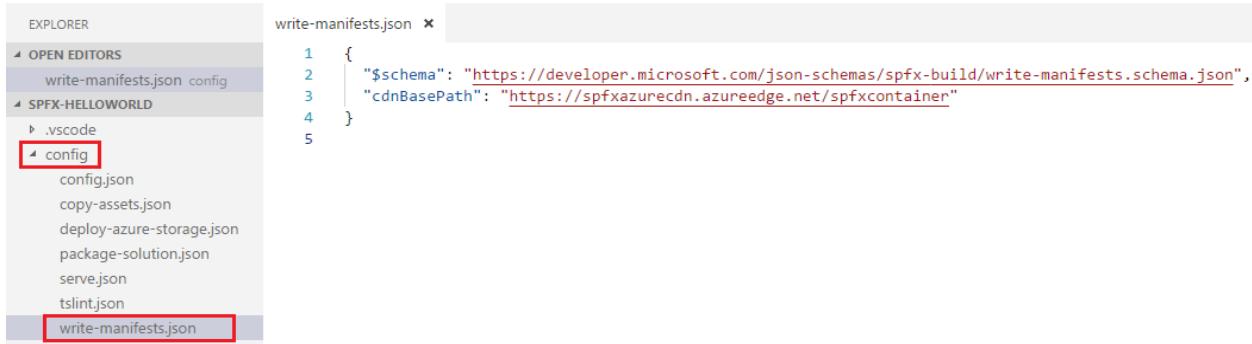
```

1 {
2   "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/deploy-azure-storage.schema.json",
3   "workingDir": "./temp/deploy/",
4   "account": "spfxdeploy",
5   "container": "spfxcontainer",
6   "accessKey": "QsPwdfEApdMWTZ/3xNbIyBonXy+fhb8o1KBH+7TrfuXWZW+AbWonvCEKfv10ouK2rZBLd6/khX54CYfVJ1xm0Q=="
7 }
8

```

Update CDN Path

1. Open **write-manifests.json** from **config** folder
2. Update CDN base path as BLOB container end point



```

EXPLORER
OPEN EDITORS
write-manifests.json config
SPFX-HELLOWORLD
  .vscode
    config
      config.json
      copy-assets.json
      deploy-azure-storage.json
      package-solution.json
      serve.json
      tslint.json
      write-manifests.json
        <-- Red Box

```

```

1 {
2   "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/write-manifests.schema.json",
3   "cdnBasePath": "https://spfxazurecdn.azureedge.net/spfxcontainer"
4 }
5

```

Prepare the package

In the command prompt, type below command

```
gulp bundle --ship
```

This will minify the required assets to upload to CDN. The ship switch denotes distribution. The minified assets are located at “temp\deploy” folder.

Deploy assets to Azure Storage

In the command prompt, type below command

```
gulp deploy-azure-storage
```

This will deploy the assets (JavaScript, CSS files) to Azure CDN.

Deploy Package to SharePoint

The next step is to deploy the app package (sppkg) to SharePoint App catalog.

In the command prompt, type below command

```
gulp package-solution --ship
```

This will create the solution package (sppkg) in sharepoint\solution folder

Upload package to app catalog

1. Open the SharePoint app catalog site
2. Upload the solution package (sppkg) from sharepoint\solution folder to app catalog
3. Make sure the url is pointing to Azure CDN

Do you trust spfx-helloworld-client-side-solution?

The client-side solution you are about to deploy contains full trust client side code. The components in the solution can, and usually do, run in full trust, and no resource usage restrictions are placed on them.

This client side solution will get content from the following domains:

<https://spfxazurecdn.azureedge.net/spfxcontainer/>



spfx-helloworld-client-side-solution

Make this solution available to all sites in the organization

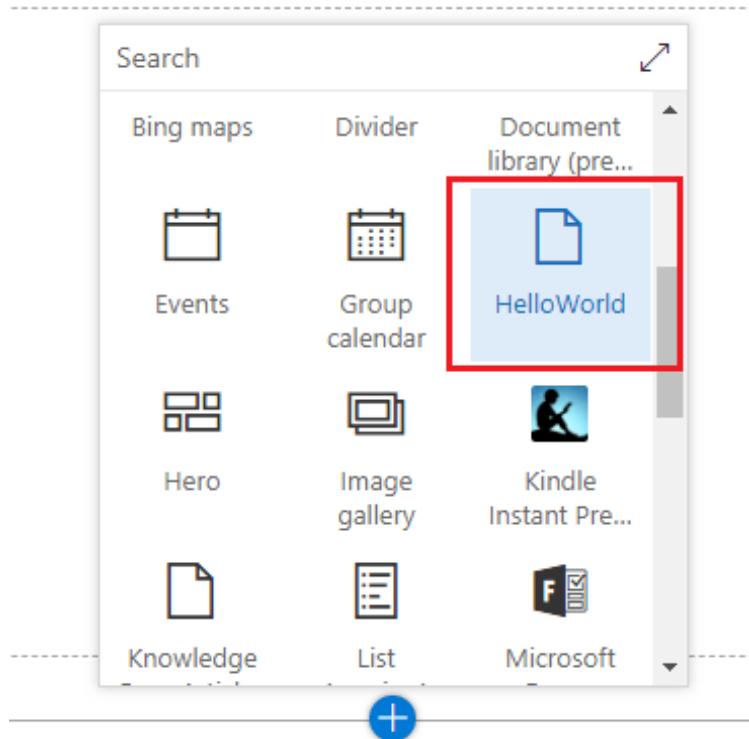
Deploy

Cancel

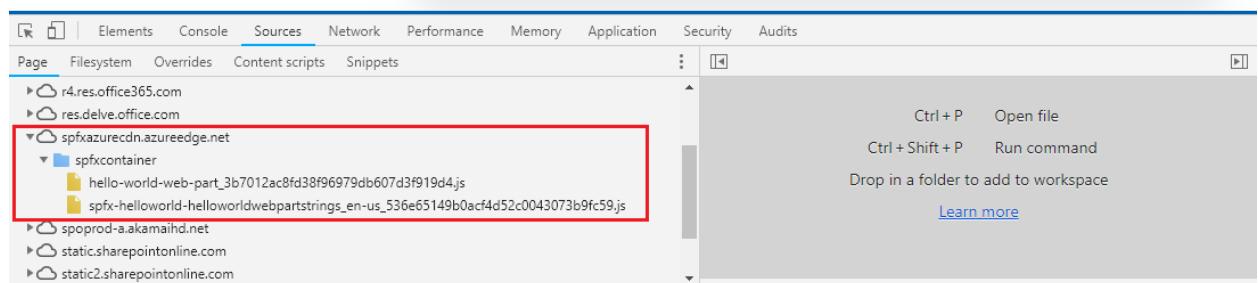
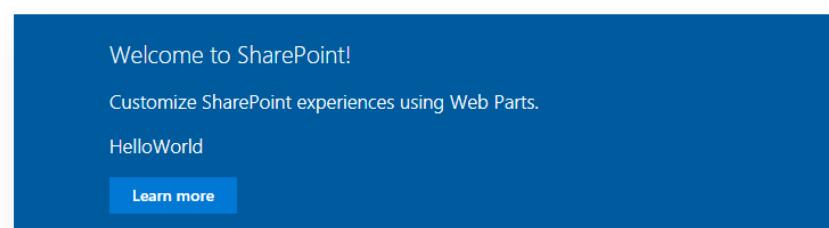
4. Click Deploy

Test the web part

1. Open any SharePoint site in your tenant
2. Add the App to your site from “Add an App” menu
3. Edit any page and add the webpart



- Click F12 to open developer toolbar. Confirm that it is served from Azure CDN



Summary

Azure CDN is one of the preferred option to deploy the SPFx client side webpart. SharePoint framework provides out of box support to deploy assets to Azure CDN. You can update the CDN path later in SPFx solution and redeploy it.

Chapter 7: Deploy SPFx WebParts to Office 365 Public CDN

Overview

In the real life scenarios we will have to deploy these SPFx webparts to some hosting environments from where it can be served to SharePoint.

Below are the more common followed deployment approaches

1. Azure CDN
2. Office 365 Public CDN
3. SharePoint Library in your tenant

In this chapter, we will explore how SPFx webparts can be deployed to office 365 CDN.

Configure CDN in Office 365 Tenant

Any of the document library in SharePoint Online tenant can be promoted as a CDN, which will help to serve the JS files for SPFx client webparts hosted in SharePoint. This CDN location being public can be accessed easily.

To configure the CDN in Office 365 follow below steps:

1. Download and install latest version of SharePoint Online Management Shell from
<https://www.microsoft.com/en-us/download/details.aspx?id=35588>
2. Open the SharePoint Online Management Shell
3. Connect to SharePoint Online tenant using below cmdlet

```
Connect-SPOService -Url https://[tenant]-admin.sharepoint.com
```

 Replace [tenant] with your actual tenant name
4. Run below set of commands to review the existing Office 365 public CDN settings on your tenant

```
Get-SPOTenantCdnEnabled -CdnType Public
```

This command will return the status of CDN. True if CDN is enabled, false otherwise.

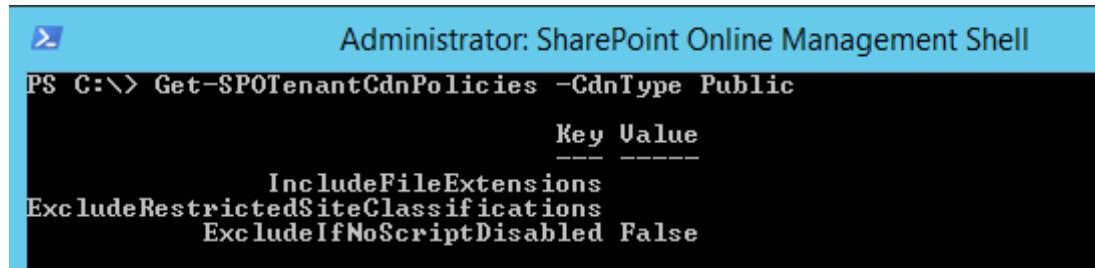
```
> Administrator: SharePoint Online Management Shell
PS C:\> Get-SPOTenantCdnEnabled -CdnType Public
Value
-----
False
```

```
Get-SPOTenantCdnOrigins -CdnType Public
```

```
> Administrator: SharePoint Online Management Shell
PS C:\> Get-SPOTenantCdnOrigins -CdnType Public
PS C:\>
```

This command will return the location of already configured CDN origins. The urls are relative. On first instance, it will not return any values as CDN is not yet setup.

```
Get-SPOTenantCdnPolicies -CdnType Public
```

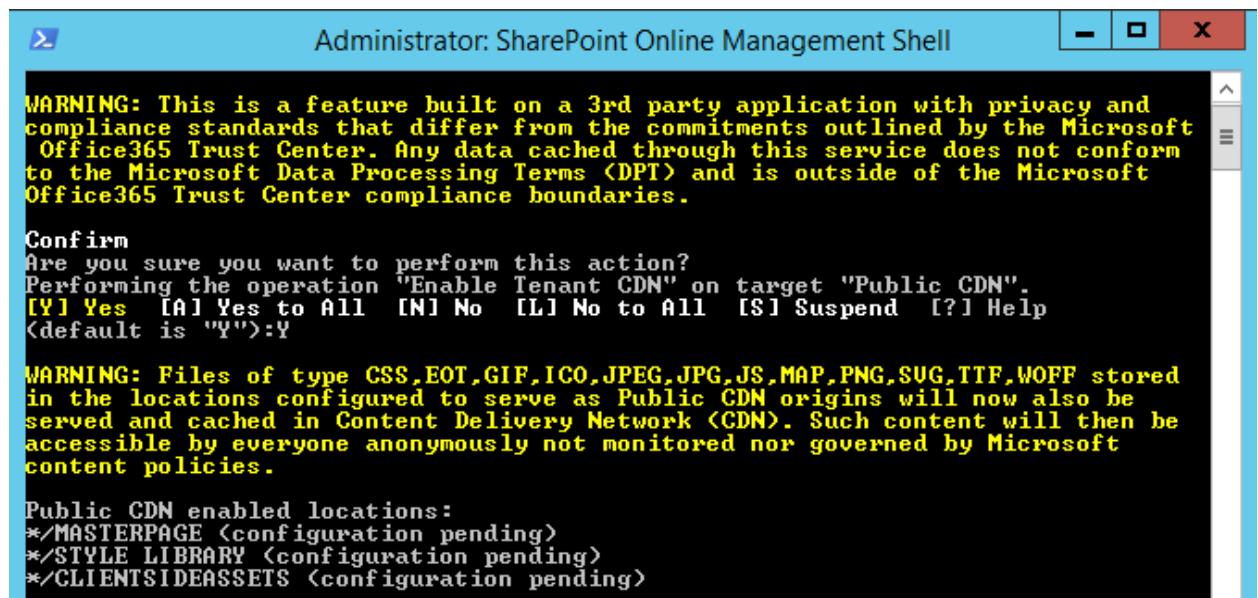


Key	Value
IncludeFileExtensions	
ExcludeRestrictedSiteClassifications	
ExcludeIfNoScriptDisabled	False

This command will return the policy settings for CDN

- Run below command to enable the CDN

```
Set-SPOTenantCdnEnabled -CdnType Public
```



WARNING: This is a feature built on a 3rd party application with privacy and compliance standards that differ from the commitments outlined by the Microsoft Office365 Trust Center. Any data cached through this service does not conform to the Microsoft Data Processing Terms (DPT) and is outside of the Microsoft Office365 Trust Center compliance boundaries.

Confirm
Are you sure you want to perform this action?
Performing the operation "Enable Tenant CDN" on target "Public CDN".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help
<default is "Y">:Y

WARNING: Files of type CSS,EOT,GIF,ICO,JPEG,JPG,JS,MAP,PNG,SVG,TTF,WOFF stored in the locations configured to serve as Public CDN origins will now also be served and cached in Content Delivery Network (CDN). Such content will then be accessible by everyone anonymously not monitored nor governed by Microsoft content policies.

Public CDN enabled locations:
*/MASTERPAGE <configuration pending>
*/STYLE LIBRARY <configuration pending>
*/CLIENTSIDEASSETS <configuration pending>

After enabling the CDN, */CLIENTSIDEASSETS origin is by default added as valid origin. By default allowed file extensions are: CSS, EOT, GIF, ICO, JPEG, JPG, JS, MAP, PNG, SVG, TTF, and WOFF. The configuration takes up to 15 to 20 minutes.

- To check the current status of CDN end points, run below command

```
Get-SPOTenantCdnOrigins -CdnType Public
```

Administrator: SharePoint Online Management Shell

```
PS C:\> Get-SPOTenantCdnOrigins -CdnType Public
*/MASTERPAGE <configuration pending>
*/STYLE LIBRARY <configuration pending>
*/CLIENTSIDEASSETS <configuration pending>
PS C:\> _
```

The origin will be listed without (configuration pending) status when it is ready. Please be patient for 15 to 20 minutes until it is ready.

Once CDN origin is ready, the output will be as below.

Administrator: SharePoint Online Management Shell

```
PS C:\> Get-SPOTenantCdnOrigins -CdnType Public
*/MASTERPAGE
*/STYLE LIBRARY
*/CLIENTSIDEASSETS
PS C:\> _
```

Setup New Office 365 CDN in Tenant

Follow below steps to setup new CDN location

1. Open SharePoint site (e.g. [https://\[tenant\].sharepoint.com/sites/\[site-collection-name\]](https://[tenant].sharepoint.com/sites/[site-collection-name]))
2. Create a document library (e.g. CDN)
3. Run below command in SharePoint Online Management Shell

```
Add-SPOTenantCdnOrigin -CdnType Public -OriginUrl sites/[site-collection-name]/ [document-library]
```

The OriginUrl is a relative url. New CDN origin configuration will again take 15 to 20 minutes.

4. Run Get-SPOTenantCdnOrigins to check the status

Administrator: SharePoint Online Management Shell

```
PS C:\Windows\system32> Get-SPOTenantCdnOrigins -CdnType Public
*/MASTERPAGE
*/STYLE LIBRARY
*/CLIENTSIDEASSETS
SITES/MODERNCOMMUNICATION/CDN
PS C:\Windows\system32> _
```

5. Create a folder in document library preferably with name of SPFx web part (e.g. O365CDNDeploy)
6. To get the path of CDN type below url in browser

```
https://\[tenant\].sharepoint.com/\_vti\_bin/publiccdn.ashx?url?itemurl=https://\[tenant\].sharepoint.com/sites/\[site-collection-name\]/\[document-library\]/\[folder\]
```

Configure SPFx Solution for Azure CDN

Update package details

1. Open command prompt
2. In the command prompt, navigate to SPFx solution folder
3. Type “code .” to open the solution in code editor of your choice
4. Open package-solution.json file from config folder. This file takes care of solution packaging.
5. Set includeClientSideAssets value as false. The client side assets will not be packaged inside final package (sppkg file) because these will be hosted on external Office 365 public CDN.



```

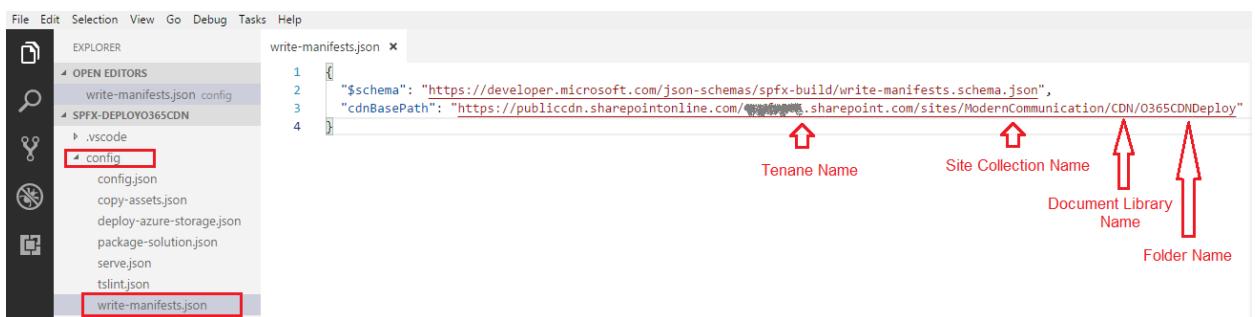
EXPLORER package-solution.json x
OPEN EDITORS
SPFx-DEPLOYO365CDN
  .vscode
    config
      config.json
      copy-assets.json
      deploy-azure-storage.json
      package-solution.json
      serve.json
  ...
  package-solution.json

1 {
2   "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/package-solution.schema.json",
3   "solution": {
4     "name": "spfx-deployo-365-cdn-client-side-solution",
5     "id": "da7d753b-b447-46f9-aac0-d49697f08599",
6     "version": "1.0.0.0",
7     "includeClientSideAssets": false
8   },
9   "paths": {
10     "zippedPackage": "solution/spfx-deployo-365-cdn.sppkg"
11   }
12 }

```

Update CDN Path

1. Open write-manifests.json from config folder
2. Update CDN base path as Office 365 CDN end point
[https://publiccdn.sharepointonline.com/\[tenant\]/sites/\[site-collection-name\]/\[document-library\]/\[folder\]](https://publiccdn.sharepointonline.com/[tenant]/sites/[site-collection-name]/[document-library]/[folder])



```

File Edit Selection View Go Debug Tasks Help
EXPLORER write-manifests.json x
OPEN EDITORS
SPFx-DEPLOYO365CDN
  .vscode
    config
      config.json
      copy-assets.json
      deploy-azure-storage.json
      package-solution.json
      serve.json
      tslint.json
      write-manifests.json

```

```

1 {
2   "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/write-manifests.schema.json",
3   "cdnBasePath": "https://publiccdn.sharepointonline.com/[tenant].sharepoint.com/sites/ModernCommunication/CDN/O365CDNDeploy"
4 }

```

↑ ↑ ↑ ↑
 Tenant Name Site Collection Name Document Library Name Folder Name

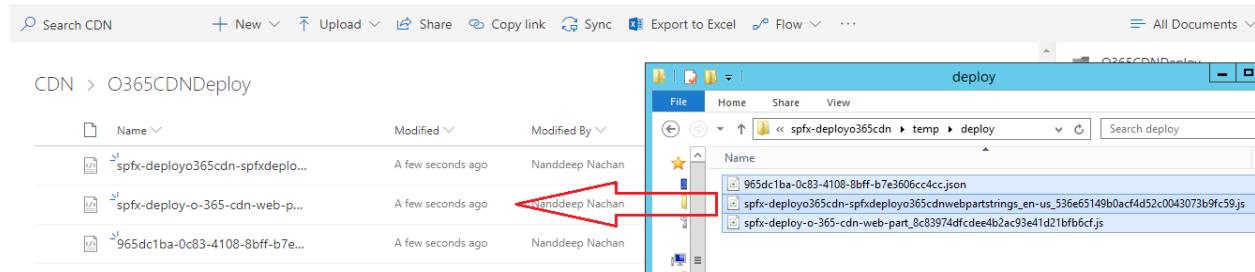
Prepare the package

In the command prompt, type the below command

```
gulp bundle --ship
```

This will minify the required assets to upload to CDN. The ship switch denotes distribution. The minified assets are located at “temp\deploy” folder.

Upload the files from “temp\deploy” folder to CDN location (SharePoint document library setup as CDN)



The screenshot shows a SharePoint document library named 'O365CDNDeploy'. Inside, there's a folder named 'deploy'. The 'deploy' folder contains three files:

- spfx-deployo365cdn-spfxdepl...
- spfx-deploy-o-365-cdn-web-p...
- 965dc1ba-0c83-4108-8bff-b7e3606cc4cc.json

Deploy Package to SharePoint

In the command prompt, type the below command

```
gulp package-solution --ship
```

This will create the solution package (sppkg) in sharepoint\solution folder.

Upload package to app catalog

1. Open the SharePoint app catalog site
2. Upload the solution package (sppkg) from sharepoint\solution folder to app catalog
3. Make sure the url is pointing to Office 365 CDN

Do you trust spfx-deployo-365-cdn-client-side-solution?

The client-side solution you are about to deploy contains full trust client side code. The components in the solution can, and usually do, run in full trust, and no resource usage restrictions are placed on them.

This client side solution will get content from the following domains:

<https://publiccdn.sharepointonline.com/> [REDACTED].sharepoint.com/sites/ModernCommunication/CDN/O365CDNDeploy/



spfx-deployo-365-cdn-client-side-solution

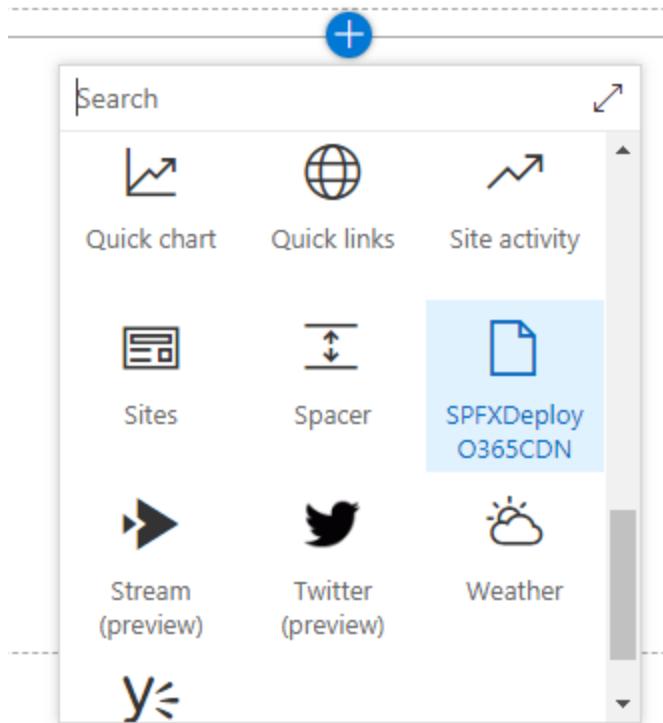
Deploy

Cancel

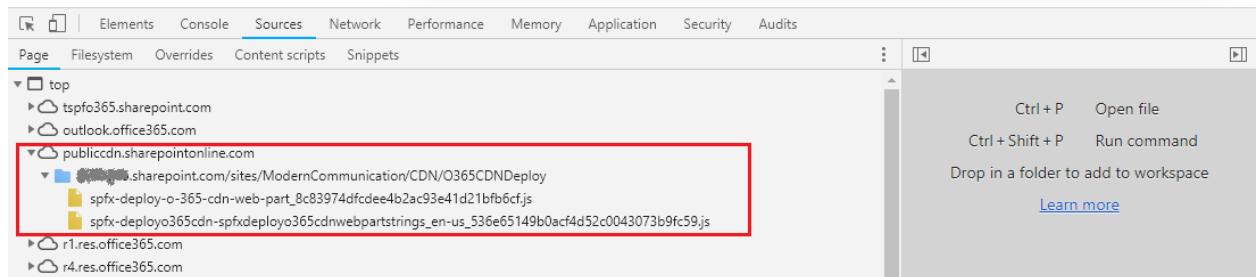
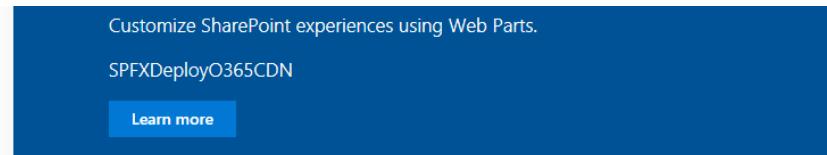
4. Click Deploy

Test the web part

1. Open any SharePoint site in your tenant
2. Add the App to your site from “Add an App” menu
3. Edit any page and add the webpart



4. Click F12 to open developer toolbar. Confirm that it is served from Office 365 Public CDN



Summary

Office 365 CDN is one of the feasible option to deploy the SPFx client side webpart. It requires some manual steps to copy the assets to CDN path. You can update the Office 365 CDN path later in SPFx solution and redeploy it.

Chapter 8: Deploy SPFx WebParts to SharePoint Library

Overview

Please refer chapters to get started with developing SPFx client side web parts and test it on a local SharePoint workbench.

In the production ready scenario, the webparts can be deployed to any of below

1. Azure CDN
2. Office 365 Public CDN
3. SharePoint Library in your tenant

In this chapter, we will explore the option of deploying the SPFx webparts using SharePoint library.

Deployment to CDN

The build output of SPFx client side webpart consists of solution package (.sppkg file) and various scripts, css and other assets. The sppkg file is deployed to SharePoint which includes manifest with url pointing to location where script files are deployed. The script files should exists at the location specified in manifest file, otherwise the web part will not work in SharePoint.

These files usually gets deployed to a CDN location. CDN location could be Azure BLOB storage or Office 365 public CDN. We can use any of this CDN location, however using CDN is not must. SharePoint Framework does not impose any restrictions that scripts should be deployed to any CDN. You can upload the scripts anywhere from where it can be accessed. That means, you can even upload the scripts to a SharePoint library inside your tenant to which everyone has an access. SharePoint Framework client webpart can access the scripts from SharePoint library.

In short, scripts can be deployed to any location including Azure BLOB storage, Office 365 CDN, any document library in SharePoint or even a physical web server from where it can be accessed. However each of these approaches has its own pros and cons with regards to cost, performance, and scalability.

If the organization has its users worldwide across different geo locations, having a CDN will help to serve as a central repository.

In package-solution.json, specifying includeClientSideAssets as true will include the assets in .sppkg file. After deploying the sppkg solution to app catalog, SharePoint will unpack the assets to predefined location in tenant.

Create SharePoint Library as CDN

1. Open SharePoint Site
2. Click Settings (gear icon) > Add an App
3. Click Document Library tile
4. Give it a name - SPFxDeploy
5. Click (gear icon) > Library settings
6. Under “Permissions and Management”, click “Permissions for this document library”
7. Give read permission to all users (use everyone user)

Configure SPFx Solution for SharePoint Library

Update package details

1. Open command prompt
2. In the command prompt, navigate to SPFx solution folder
3. Type “code .” to open the solution in code editor of your choice
4. Open package-solution.json file from config folder. This file takes care of solution packaging.
5. Set includeClientSideAssets value as false. The client side assets will not be packaged inside final package (sppkg file) because these will be hosted in SharePoint library.



```

EXPLORER package-solution.json x
OPEN EDITORS
  package-solution.json config
SPEX-DEPLOYTODOCLIB
  .vscode
    config
      config.json
      copy-assets.json
      deploy-azure-storage.json
    package-solution.json
      serve.json
      tslint.json
      write-manifests.json
  package-solution.json
  serve.json
  tslint.json
  write-manifests.json

package-solution.json
1  {
2    "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/package-solution.schema.json",
3    "solution": {
4      "name": "spfx-deploytodoilib-client-side-solution",
5      "id": "c56a47ad-a0e5-40a6-805d-423190246196",
6      "version": "1.0.0.0",
7      "includeClientSideAssets": false
8    },
9    "paths": {
10      "zippedPackage": "solution/spfx-deploytodoilib.sppkg"
11    }
12  }
13

```

Update CDN Path

3. Open write-manifests.json from config folder
4. Update CDN base path as SharePoint library url



```

EXPLORER write-manifests.json x
OPEN EDITORS
  write-manifests.json config
SPEX-DEPLOYTODOCLIB
  .vscode
    config
      config.json
      copy-assets.json
      deploy-azure-storage.json
      package-solution.json
      serve.json
      tslint.json
    write-manifests.json
  write-manifests.json
  serve.json
  tslint.json
  write-manifests.json

write-manifests.json
1  {
2    "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/write-manifests.schema.json",
3    "cdnBasePath": "https://myo365.sharepoint.com/sites/ModernCommunication/SPFxDeploy/"
4  }
5

```

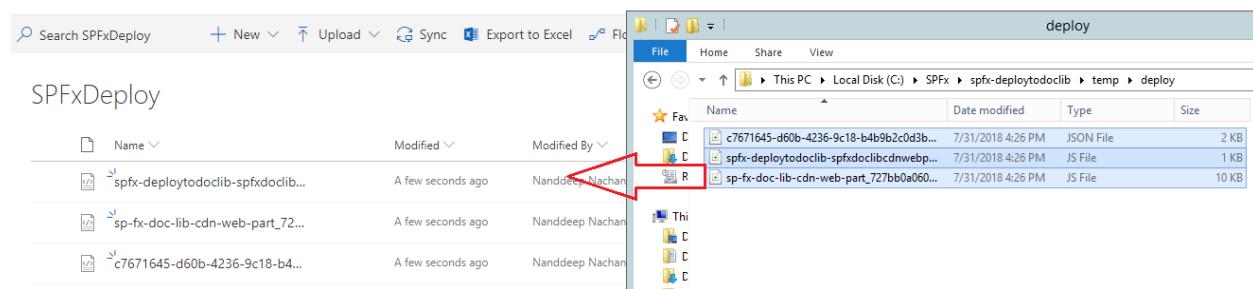
Prepare the package

In the command prompt, type the below command

```
gulp bundle --ship
```

This will minify the required assets to upload to CDN. The ship switch denotes distribution. The minified assets are located at “temp\deploy” folder.

Upload the files from “temp\deploy” folder to SharePoint library



Deploy Package to SharePoint

In the command prompt, type the below command

```
gulp package-solution --ship
```

This will create the solution package (sppkg) in sharepoint\solution folder.

Upload package to app catalog

1. Open the SharePoint app catalog site
2. Upload the solution package (sppkg) from sharepoint\solution folder to app catalog
3. Make sure the url is pointing to SharePoint library

Do you trust spfx-deploytodoclib-client-side-solution? X

The client-side solution you are about to deploy contains full trust client side code. The components in the solution can, and usually do, run in full trust, and no resource usage restrictions are placed on them.



spfx-deploytodoclib-client-side-solution

This client side solution will get content from the following domains:

[https://\[REDACTED\].sharepoint.com/sites/ModernCommunication/SPFxDeploy/](https://[REDACTED].sharepoint.com/sites/ModernCommunication/SPFxDeploy/)

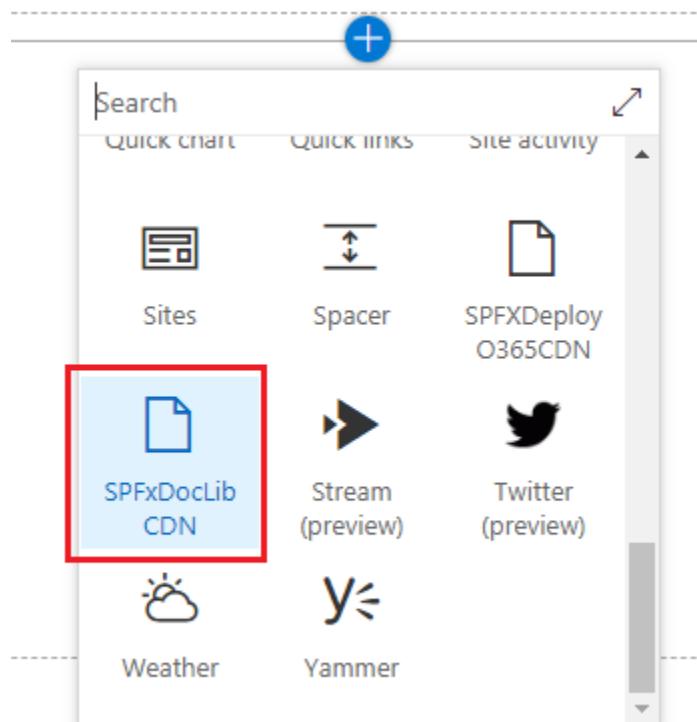
Deploy

Cancel

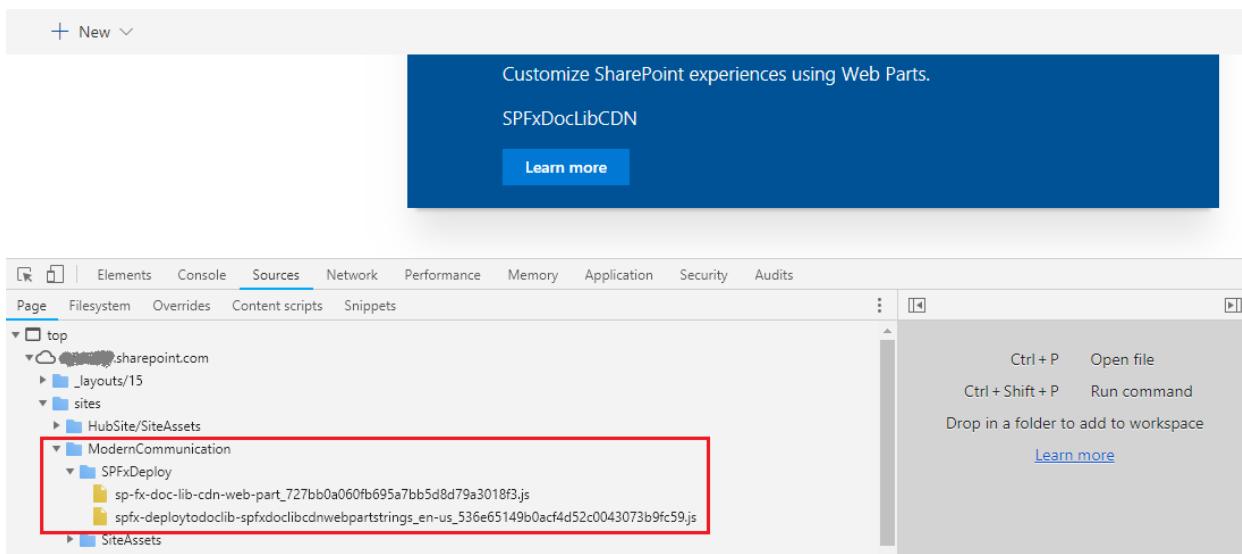
4. Click Deploy

Test the web part

1. Open any SharePoint site in your tenant
2. Add the App to your site from “Add an App” menu
3. Edit any page and add the webpart



4. Click F12 to open developer toolbar. Confirm that it is served from SharePoint library



Summary

There are multiple options to deploy the SPFx client side webparts ranging from Azure BLOB storage, Office 365 public CDN, or even regular SharePoint library. Considering the pros and cons of each option, choose the one that suits you better.

Chapter 9: Integrating JQuery with SPFx Web Parts

Overview

SharePoint Framework Client side webparts are developed using JavaScript framework. JQuery is one of the largely used JavaScript framework.

In this chapter, we will explore how to integrate JQuery with SPFx web parts.

Create SPFx Solution

1. Create a directory for SPFx solution

```
md spfx-jqueryintegration
```

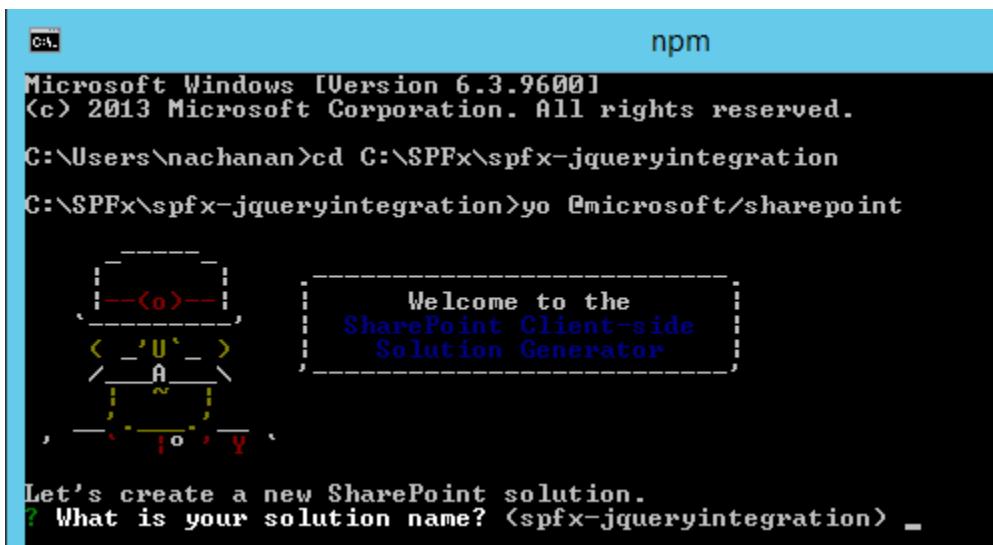
2. Navigate to above created directory

```
cd spfx-jqueryintegration
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



```
npm
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\nachanan>cd C:\SPFx\spfx-jqueryintegration
C:\SPFx\spfx-jqueryintegration>yo @microsoft/sharepoint

Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? <spfx-jqueryintegration> _
```

Solution Name: Hit enter to have default name (spfx-jqueryintegration in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension. Choose webpart option.

Selected choice: WebPart

Web part name: Hit enter to select the default name or type in any other name.

Selected choice: SPFxJqueryIntegration

Web part description: Hit enter to select the default description or type in any other value.

Selected choice: Hit enter (default description)

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)

Selected choice: No JavaScript Framework. We will integrate JQuery to this solution.

Configured Required Packages and Dependencies

Include Packages

We will now include the JQuery packages. Type in below commands on command prompt.

```
npm i jquery jqueryui combokeys --save
```

The --save option enables NPM to include the packages to dependencies section of package.json file.

Include typings

Typings will help for auto complete while writing the code in code editor. Type below command.

```
tsd install jquery jqueryui combokeys --save
```

Lock down the package dependencies

Type in below command to lock down the package dependencies

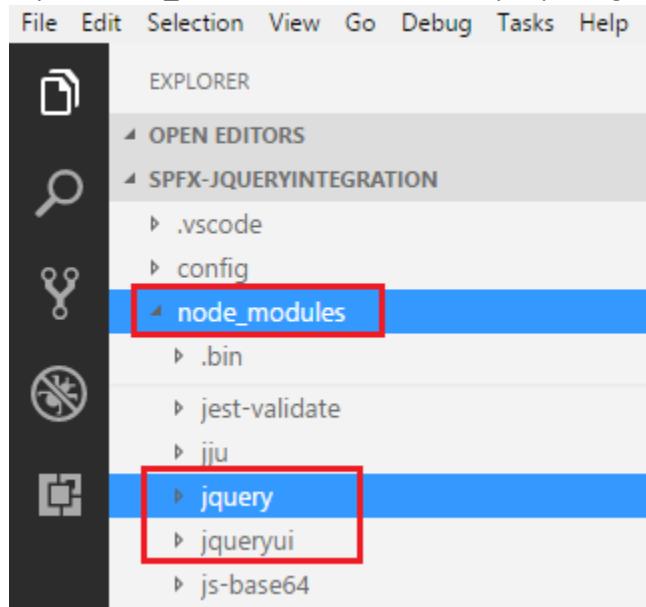
```
npm shrinkwrap
```

Solution Changes

1. In the command prompt, type below command to open the solution in code editor of your choice.

```
code .
```

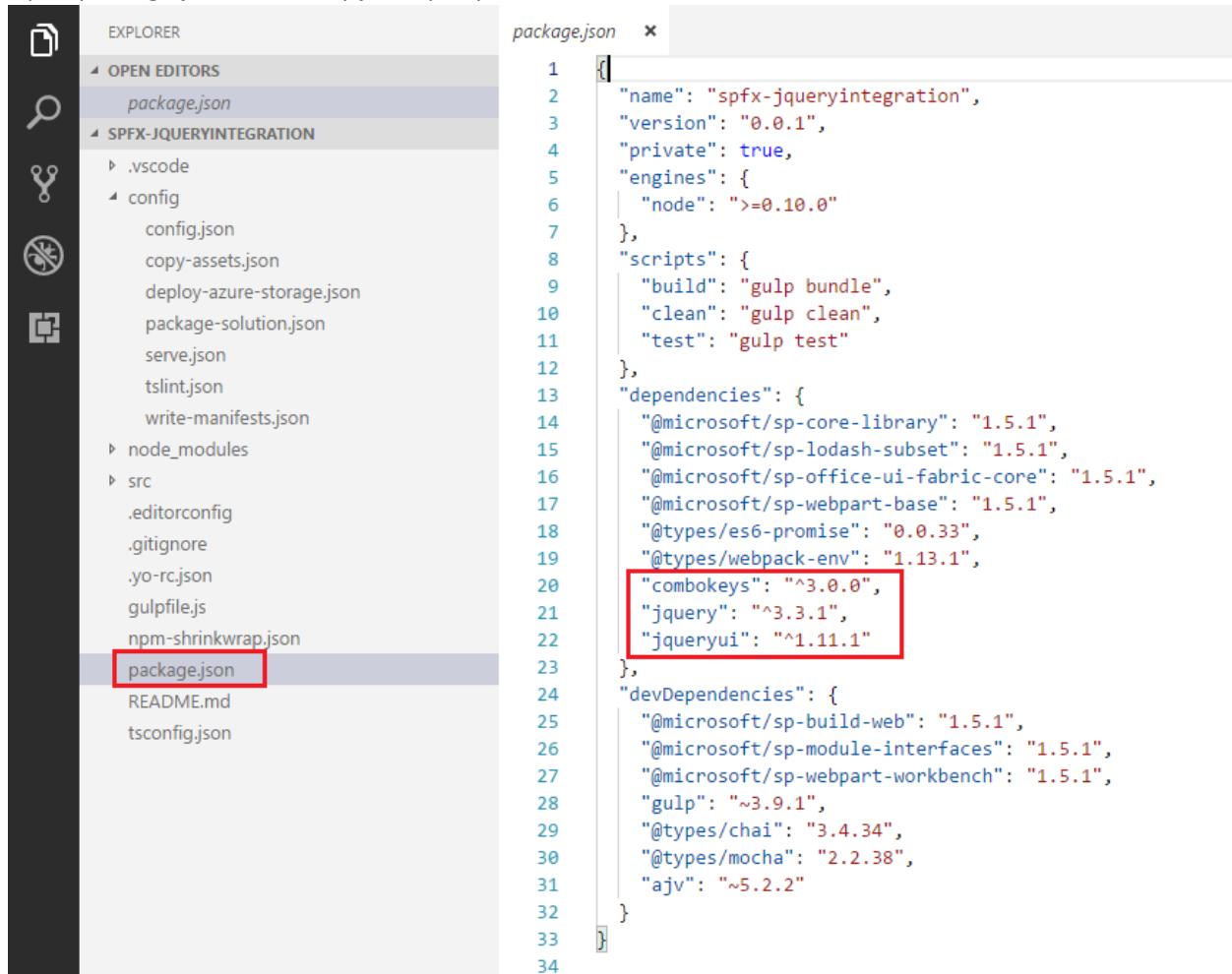
2. Expand node_module folder to see the npm packages being added for jQuery



3. Open config.json under config folder
4. Under externals node, add jQuery references



5. Open package.json and verify jQuery dependencies are listed



```

EXPLORER package.json
OPEN EDITORS package.json
SPFX-JQUERYINTEGRATION
  .vscode
  config
    config.json
    copy-assets.json
    deploy-azure-storage.json
    package-solution.json
    serve.json
    tslint.json
    write-manifests.json
  node_modules
  src
    .editorconfig
    .gitignore
    .yo-rc.json
    gulpfile.js
    npm-shrinkwrap.json
  package.json
  README.md
  tsconfig.json
  
```

```

1  {
2    "name": "spfx-jqueryintegration",
3    "version": "0.0.1",
4    "private": true,
5    "engines": {
6      "node": ">=0.10.0"
7    },
8    "scripts": {
9      "build": "gulp bundle",
10     "clean": "gulp clean",
11     "test": "gulp test"
12   },
13   "dependencies": {
14     "@microsoft/sp-core-library": "1.5.1",
15     "@microsoft/sp-lodash-subset": "1.5.1",
16     "@microsoft/sp-office-ui-fabric-core": "1.5.1",
17     "@microsoft/sp-webpart-base": "1.5.1",
18     "@types/es6-promise": "0.0.33",
19     "@types/webpack-env": "1.13.1",
20     "combokeys": "^3.0.0",
21     "jquery": "^3.3.1",
22     "jqueryui": "^1.11.1"
23   },
24   "devDependencies": {
25     "@microsoft/sp-build-web": "1.5.1",
26     "@microsoft/sp-module-interfaces": "1.5.1",
27     "@microsoft/sp-webpart-workbench": "1.5.1",
28     "gulp": "~3.9.1",
29     "@types/chai": "3.4.34",
30     "@types/mocha": "2.2.38",
31     "ajv": "~5.2.2"
32   }
33 }
34
  
```

6. Open webpart file SpFxJQueryIntegrationWebPart.ts and import jQuery



```

EXPLORER SpFxJQueryIntegrationWebPart.ts src\w... 1
SPFX-JQUERYINTEGRATION
  .vscode
  config
  node_modules
  src
    webparts
      spFxJQueryIntegration
        loc
          SpFxJQueryIntegrationWebPart.manif...
          SpFxJQueryIntegrationWebPart.modu...
        SpFxJQueryIntegrationWebPart.ts 1
  
```

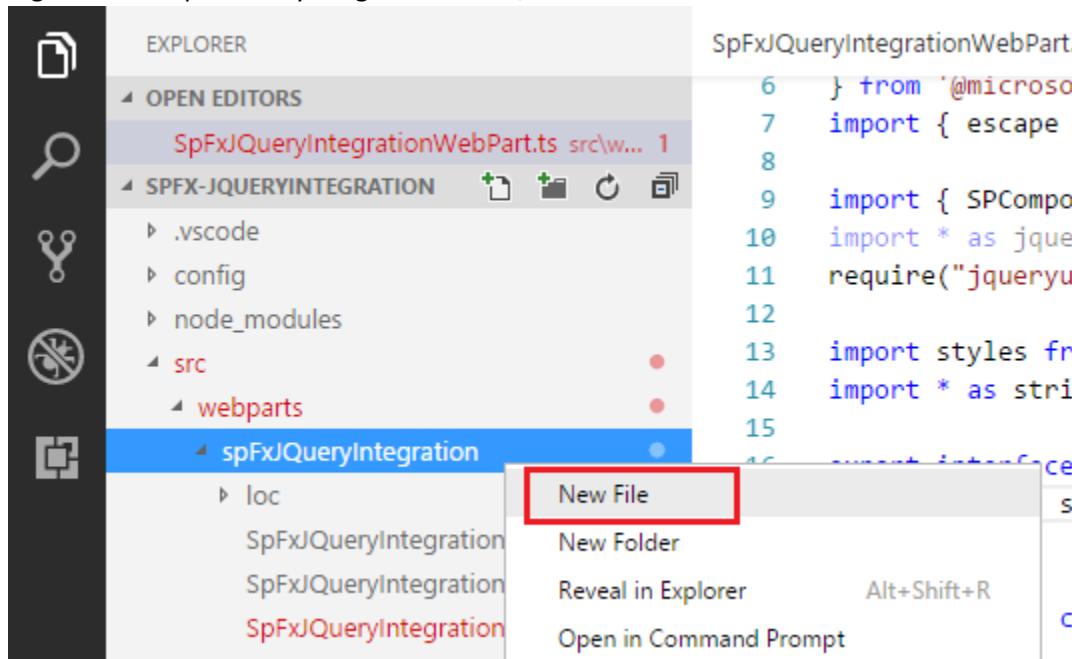
```

1  import { Version } from '@microsoft/sp-core-library';
2  import {
3    BaseClientSideWebPart,
4    IPropertyPaneConfiguration,
5    PropertyPaneTextField
6  } from '@microsoft/sp-webpart-base';
7  import { escape } from '@microsoft/sp-lodash-subset';
8
9  import { SPComponentLoader } from '@microsoft/sp-loader';
10 import * as jquery from "jquery";
11 require("jqueryui");
12
13 import styles from './SpFxJQueryIntegrationWebPart.module.scss';
14 import * as strings from 'SpFxJQueryIntegrationWebPartStrings';
15
  
```

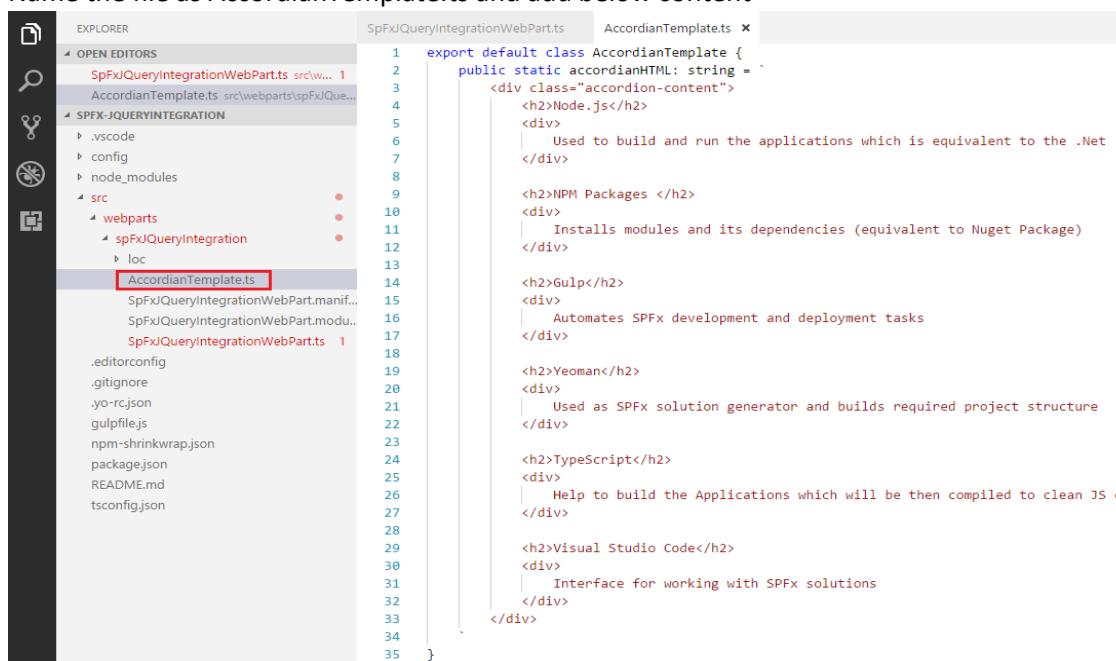
7. Define a constructor and load external jQuery UI css from it.

```
export default class SpFxJQueryIntegrationWebPart extends BaseClientSideWebPart<ISpFxJQueryIntegrationWebPartProps> {
    public constructor() {
        super();
        SPComponentLoader.loadCss("//code.jquery.com/ui/1.12.0/themes/smoothness/jquery-ui.css");
    }
}
```

8. Right click on spFxJQueryIntegration folder, click New file



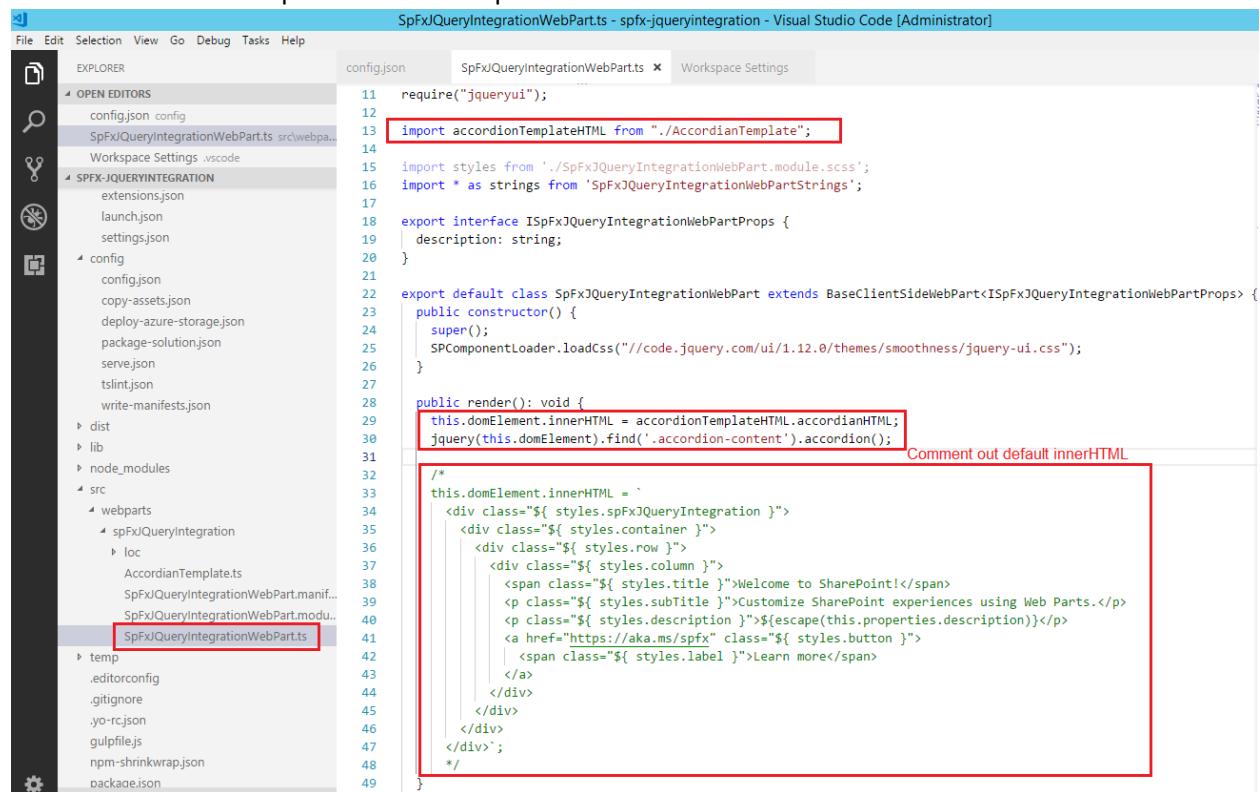
9. Name the file as AccordionTemplate.ts and add below content



The screenshot shows the VS Code editor with the 'AccordionTemplate.ts' file open. The code defines a class 'AccordionTemplate' with a static property 'accordionHTML' containing a string of HTML code. The code is annotated with comments explaining various build tools and their equivalents in .Net, NPM, Gulp, Yeoman, and TypeScript.

```
1  export default class AccordionTemplate {
2      public static accordionHTML: string =
3          <div class="accordion-content">
4              <h2>Node.js</h2>
5              <div>
6                  | Used to build and run the applications which is equivalent to the .Net
7                  | NPM Packages
8                  | Installs modules and its dependencies (equivalent to Nuget Package)
9              </div>
10             <h2>Gulp</h2>
11             <div>
12                 | Automates SPFx development and deployment tasks
13             </div>
14             <h2>Yeoman</h2>
15             <div>
16                 | Used as SPFx solution generator and builds required project structure
17             </div>
18             <h2>TypeScript</h2>
19             <div>
20                 | Help to build the Applications which will be then compiled to clean JS code
21             </div>
22             <h2>Visual Studio Code</h2>
23             <div>
24                 | Interface for working with SPFx solutions
25             </div>
26         </div>
27     }
28 }
29 }
30 }
31 }
32 }
33 }
34 }
```

10. Use the accordion template in main webpart class



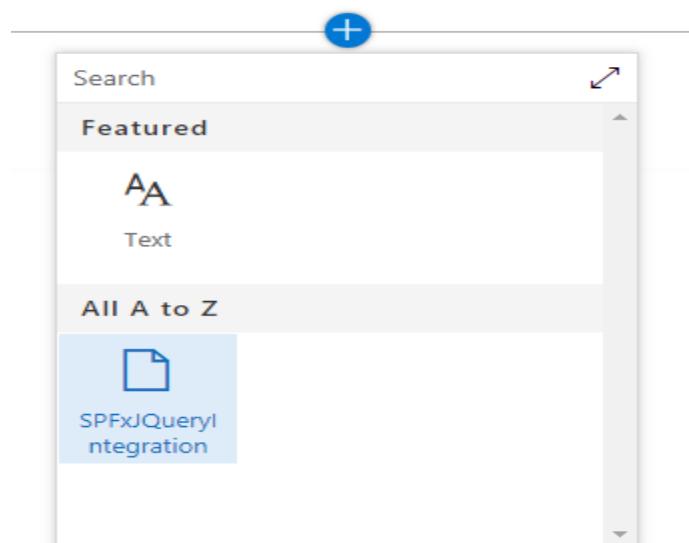
```

File Edit Selection View Go Debug Tasks Help
EXPLORER config.json SpFxJQueryIntegrationWebPart.ts ...
OPEN EDITORS config.json config ...
SPFx-JQUERYINTEGRATION SpFxJQueryIntegrationWebPart.ts src\webpa...
config.json extensions.json launch.json workspace settings.json
config.json copy-assets.json deploy-azure-storage.json package-solution.json serve.json tslint.json write-manifests.json
dist lib node_modules
src webparts spFxJQueryIntegration loc AccordionTemplate.ts SpFxJQueryIntegrationWebPart.manif... SpFxJQueryIntegrationWebPart.modu... SpFxJQueryIntegrationWebParts
temp editorconfig .gitignore yo-rc.json gulpfile.js npm-shrinkwrap.json package.json
11 require("jqueryui");
12 import accordionTemplateHTML from "./AccordionTemplate";
13
14 import styles from './SpFxJQueryIntegrationWebPart.module.scss';
15 import * as strings from 'SpFxJQueryIntegrationWebPartStrings';
16
17 export interface ISpFxJQueryIntegrationWebPartProps {
18   description: string;
19 }
20
21 export default class SpFxJQueryIntegrationWebPart extends BaseClientSideWebPart<ISpFxJQueryIntegrationWebPartProps> {
22   public constructor() {
23     super();
24     SPComponentLoader.loadCss("//code.jquery.com/ui/1.12.0/themes/smoothness/jquery-ui.css");
25   }
26
27   public render(): void {
28     this.domElement.innerHTML = accordionTemplateHTML.accordionHTML;
29     jquery(this.domElement).find('.accordion-content').accordion();
30   }
31
32   /*
33   this.domElement.innerHTML =
34     <div class="${ styles.spFxJQueryIntegration }">
35       <div class="${ styles.container }">
36         <div class="${ styles.row }">
37           <div class="${ styles.column }">
38             <span class="${ styles.title }>Welcome to SharePoint!</span>
39             <p class="${ styles.subtitle }>Customize SharePoint experiences using Web Parts.</p>
40             <p class="${ styles.description }>${escape(this.properties.description)}</p>
41             <a href="https://aka.ms/spfx" class="${ styles.button }">
42               <span class="${ styles.label }>Learn more</span>
43             </a>
44           </div>
45         </div>
46       </div>
47     </div>;
48   */
49 }

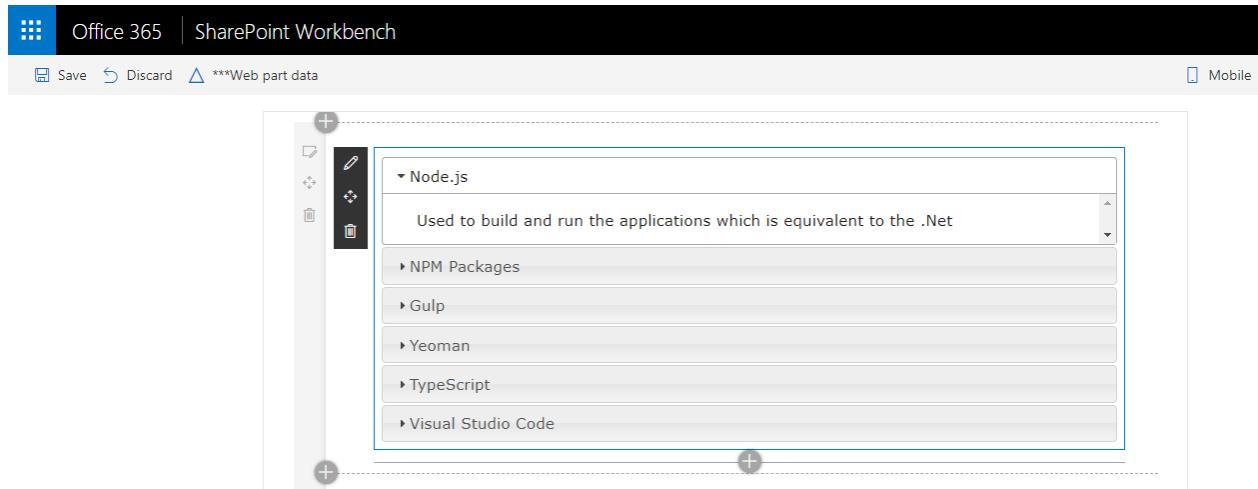
```

Test the web part

1. On command prompt, type `gulp serve`
2. Open any SharePoint site in your tenant or use SharePoint local workbench
3. Add the App to your site from “Add an App” menu
4. Edit any page and add the webpart



5. See the webpart working



Summary

JavaScript frameworks like JQuery can be easily integrated with SPFx client web parts. They also supports typings which helps for intellisense while developing the code in editors.

Chapter 10: CRUD operations using No Framework

Overview

In the chapters so far, we have explored on developing basic SharePoint client web part which can run independently without any interaction with SharePoint.

In this chapter, we will explore to interact with the SharePoint list for CRUD (Create, Read, Update, and Delete) operations using No framework option. The CRUS operations will be performed using REST APIs.

Create SPFx Solution

1. Open command prompt. Create a directory for SPFx solution

```
md spfx-crud-no-framework
```

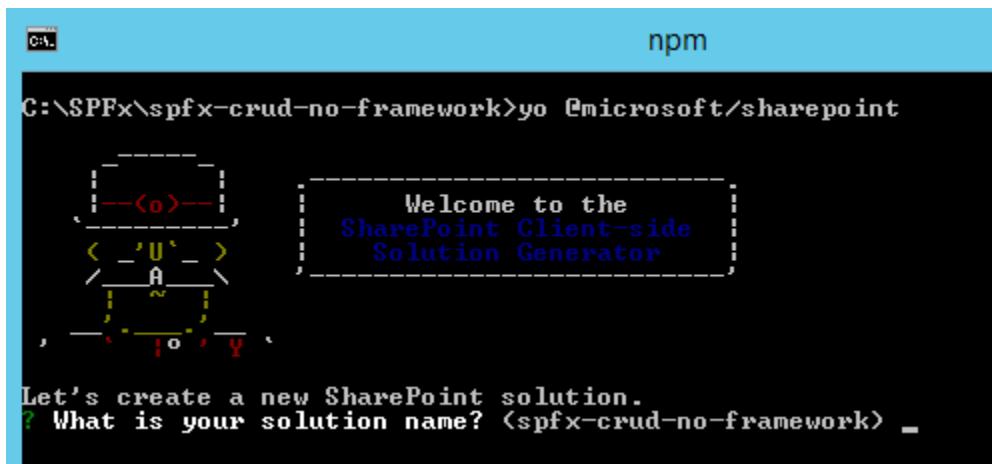
2. Navigate to above created directory

```
cd spfx-crud-no-framework
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



The screenshot shows a terminal window titled 'npm'. The command 'C:\SPFx\spfx-crud-no-framework>yo @microsoft/sharepoint' is being typed. A blue dashed box highlights the text 'Welcome to the SharePoint Client-side Solution Generator'. Below it, another blue dashed box highlights the question 'Let's create a new SharePoint solution.' followed by the prompt '? What is your solution name? <spfx-crud-no-framework>'. The terminal has a light blue header bar.

Solution Name: Hit enter to have default name (spfx-crud-no-framework in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension. Choose webpart option.

Selected choice: WebPart

Web part name: Hit enter to select the default name or type in any other name.

Selected choice: NoFrameworkCRUD

Web part description: Hit enter to select the default description or type in any other value.

Selected choice: CRUD operations with no framework

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)

Selected choice: No JavaScript Framework

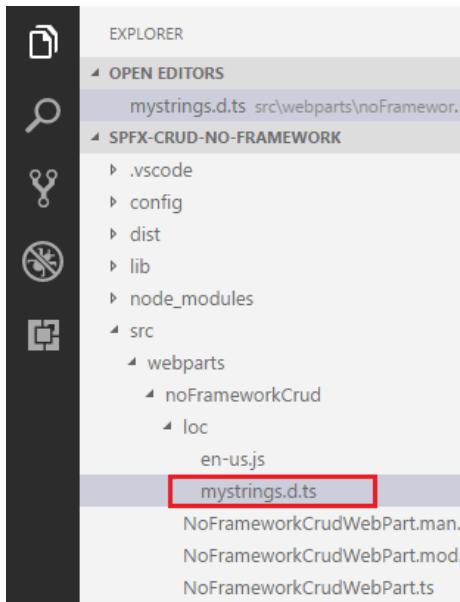
5. Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take significant amount of time.
6. Once the scaffolding process is completed, in the command prompt type below command to open the solution in code editor of your choice.

code .

Configure Property for List Name

SPFx solution by default have description property created. Let us change the property to list name. We will use this property to configure the list name on which the CRUD operation is to perform.

1. Open mystrings.d.ts under \src\webparts\noFrameworkCrud\loc\ folder
2. Rename DescriptionFieldLabel to ListNameFieldLabel



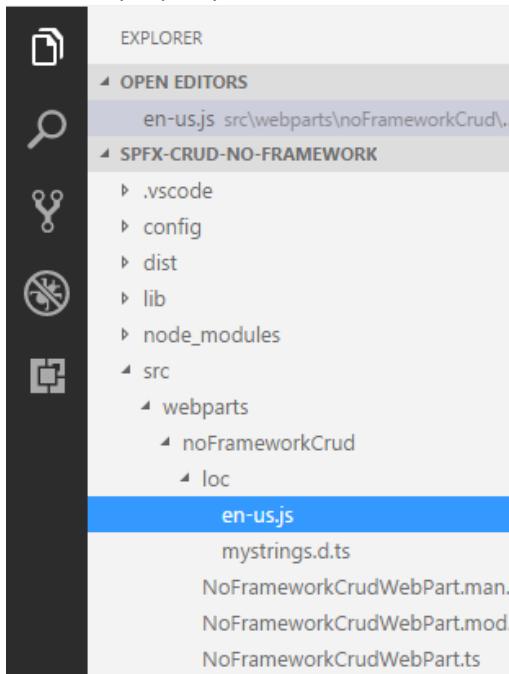
The screenshot shows the VS Code interface with the Explorer sidebar on the left. Under the 'OPEN EDITORS' section, 'mystrings.d.ts' is listed. In the main editor area, the code for 'mystrings.d.ts' is shown:

```

1 declare interface INoFrameworkCrudWebPartStrings {
2   PropertyPaneDescription: string;
3   BasicGroupName: string;
4   listNameFieldLabel: string;
5 }
6
7 declare module 'NoFrameworkCrudWebPartStrings' {
8   const strings: INoFrameworkCrudWebPartStrings;
9   export = strings;
10 }
11

```

3. In en-us.js file under \src\webparts\noFrameworkCrud\loc\ folder set the display name for listName property



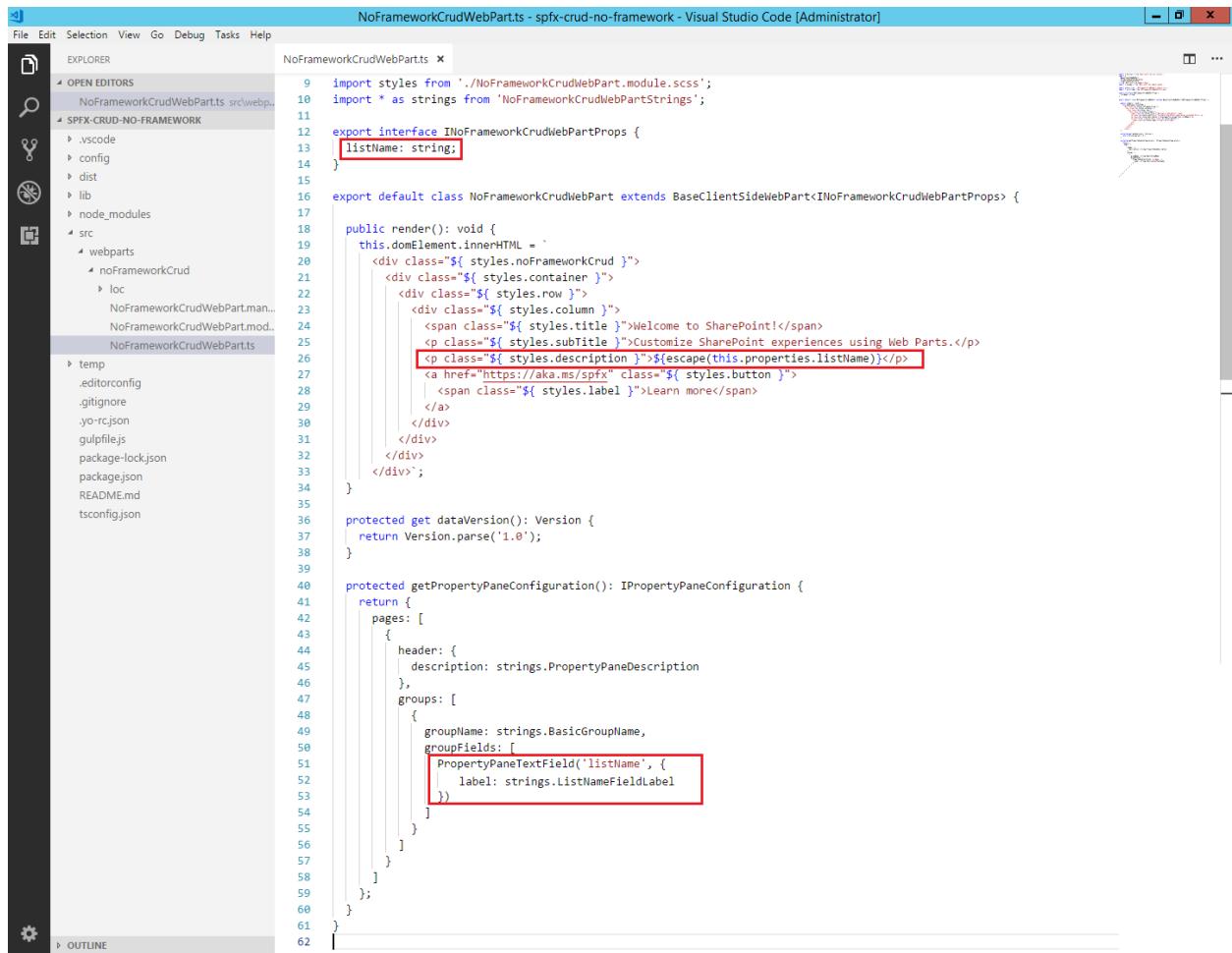
The screenshot shows the VS Code interface with the Explorer sidebar on the left. Under the 'OPEN EDITORS' section, 'en-us.js' is listed. In the main editor area, the code for 'en-us.js' is shown:

```

1 define([], function() {
2   return {
3     "PropertyPaneDescription": "Description",
4     "BasicGroupName": "Group Name",
5     "listNameFieldLabel": "List Name"
6   }
7 });

```

4. Open main webpart file (NoFrameworkCrudWebPart.ts) under \src\webparts\noFrameworkCrud folder.
5. Rename description property pane field to listName



```

NoFrameworkCrudWebPart.ts - spfx-crud-no-framework - Visual Studio Code [Administrator]

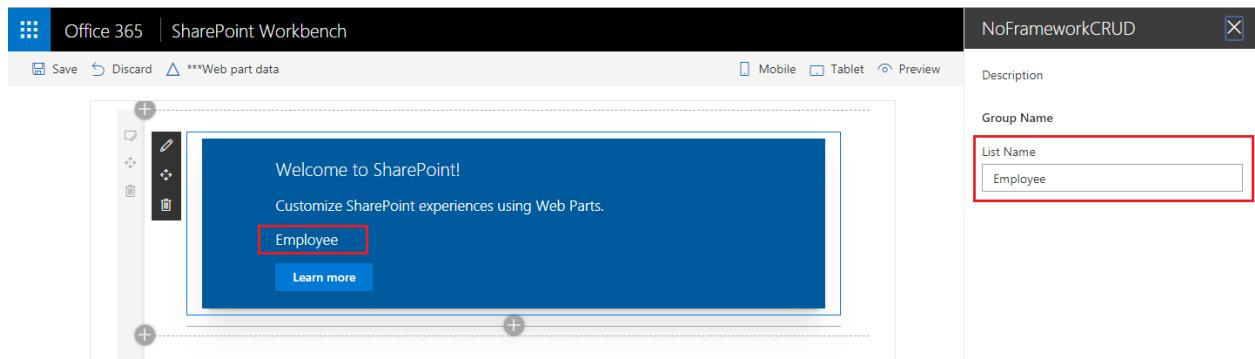
File Edit Selection View Go Debug Tasks Help

EXPLORER
OPEN EDITORS
SPFX-CRUD-NO-FRAMEWORK
  .vscode
  config
  dist
  lib
  node_modules
  src
    webparts
      noFrameworkCrud
        loc
        NoFrameworkCrudWebPart.man...
        NoFrameworkCrudWebPart.mod...
        NoFrameworkCrudWebParts.ts
      temp
.editorconfig
.gitignore
.yo-rc.json
gulpfile.js
package-lock.json
package.json
README.md
tsconfig.json

NoFrameworkCrudWebPart.ts
  9 import styles from './NoFrameworkCrudWebPart.module.scss';
10 import * as strings from 'NoFrameworkCrudWebPartStrings';
11
12 export interface INoFrameworkCrudWebPartProps {
13   listName: string;
14 }
15
16 export default class NoFrameworkCrudWebPart extends BaseClientSideWebPart<INoFrameworkCrudWebPartProps> {
17
18   public render(): void {
19     this.domElement.innerHTML = `
20       <div class="${ styles.noFrameworkCrud }">
21         <div class="${ styles.container }">
22           <div class="${ styles.row }">
23             <div class="${ styles.column }">
24               <span class="${ styles.title }">Welcome to SharePoint!</span>
25               <p class="${ styles.subtitle }">Customize SharePoint experiences using Web Parts.</p>
26               <p class="${ styles.description }">${escape(this.properties.listName)}</p>
27               <a href="https://aka.ms/spfx" class="${ styles.button }">
28                 <span class="${ styles.label }">Learn more</span>
29               </a>
30             </div>
31           </div>
32         </div>
33       </div>;
34     `;
35   }
36
37   protected get dataVersion(): Version {
38     return Version.parse('1.0');
39   }
40
41   protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
42     return {
43       pages: [
44         {
45           header: {
46             description: strings.PropertyPaneDescription
47           },
48           groups: [
49             {
50               groupName: strings.BasicGroupName,
51               groupFields: [
52                 PropertyPaneTextField('listName', {
53                   label: strings.ListNameFieldLabel
54                 })
55               ]
56             }
57           ]
58         };
59       }
60     }
61   }
62 }

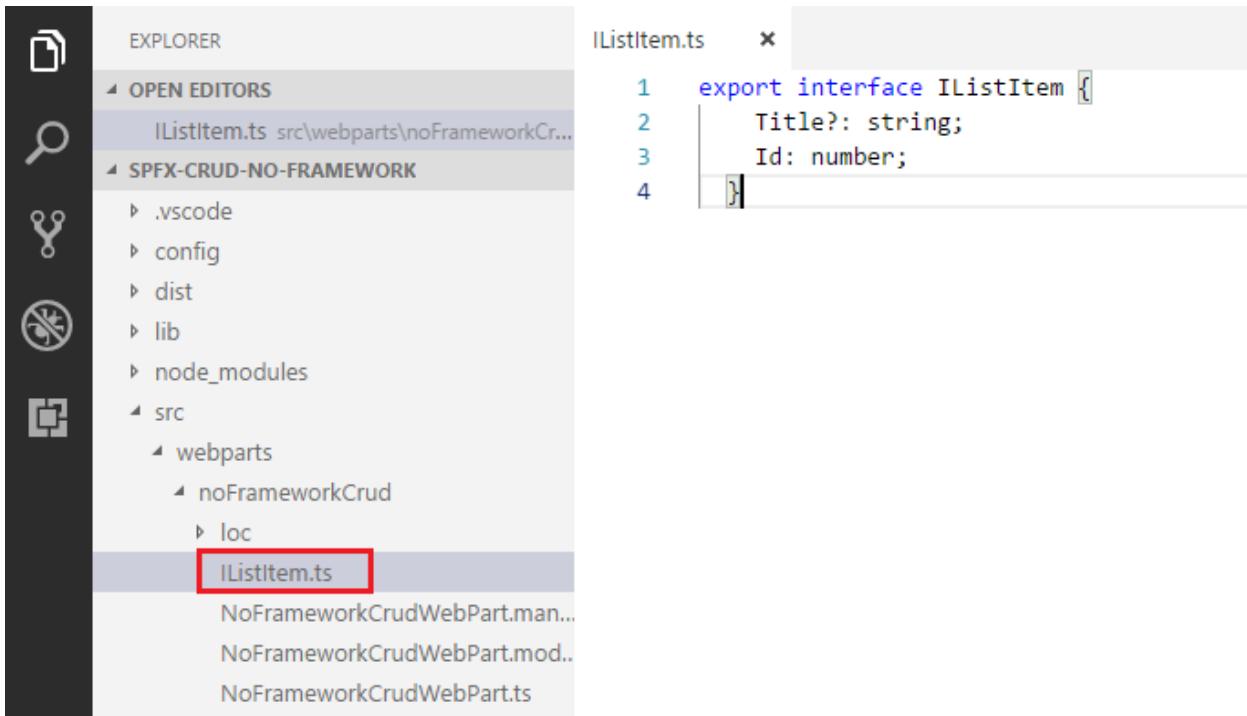
```

6. In the command prompt, type “gulp serve”
7. In the SharePoint local workbench page, add the web part.
8. Edit the web part to ensure the listName property pane field is getting reflected.



Model for List Item

Let us add a class (IListItem.ts) representing the list item.



The screenshot shows the VS Code interface with the Explorer sidebar on the left. The file structure is as follows:

- OPEN EDITORS
- IListItem.ts (highlighted with a red box)
- SFX-CRUD-NO-FRAMEWORK
- .vscode
- config
- dist
- lib
- node_modules
- src
 - webparts
 - noFrameworkCrud
 - loc
 - IListItem.ts (highlighted with a red box)
 - NoFrameworkCrudWebPart.man...
 - NoFrameworkCrudWebPart.mod..
 - NoFrameworkCrudWebPart.ts

The code editor window shows the IListItem.ts file with the following content:

```
1 export interface IListItem {  
2   Title?: string;  
3   Id: number;  
4 }
```

Add Controls to WebPart

1. Open main webpart file (NoFrameworkCrudWebPart.ts) under \src\webparts\noFrameworkCrud folder.
2. Modify Render method to include buttons for CRUD operations and add event handlers to each of the button

NoFrameworkCrudWebPart.ts - spfx-crud-no-framework - Visual Studio Code [Administrator]

```

File Edit Selection View Go Debug Tasks Help
EXPLORER NoFrameworkCrudWebPart.ts x
OPEN EDITORS
SPFX-CRUD-NO-FRAMEWORK
.vscode
config
dist
lib
node_modules
src
webparts
noFrameworkCrud
loc
IListItem.ts
NoFrameworkCrudWebPart.man...
NoFrameworkCrudWebPart.mod...
NoFrameworkCrudWebPart.ts
temp
.editorconfig
.gitignore
.yo-rc.json
gulpfile.js
package-lock.json
package.json
README.md
tsconfig.json

```

Buttons

```

19  export default class NoFrameworkCrudWebPart extends BaseClientSideWebPart<INoFrameworkCrudWebPartProps> {
20
21  public render(): void {
22    this.domElement.innerHTML = `
23      <div class="${ styles.noFrameworkCrud }">
24        <div class="${ styles.container }">
25          <div class="${ styles.row }">
26            <div class="${ styles.column }">
27              <span class="${ styles.title }">CRUD operations</span>
28              <p class="${ styles.subtitle }">No Framework</p>

```

Button Event Handlers

```

29
30
31      <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
32        <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
33          <button class="${styles.button} create-Button">
34            <span class="${styles.label}">Create item</span>
35          </button>
36          <button class="${styles.button} read-Button">
37            <span class="${styles.label}">Read item</span>
38          </button>
39        </div>
40      </div>
41
42      <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
43        <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
44          <button class="${styles.button} update-Button">
45            <span class="${styles.label}">Update item</span>
46          </button>
47          <button class="${styles.button} delete-Button">
48            <span class="${styles.label}">Delete item</span>
49          </button>
50        </div>
51      </div>
52    </div>
53  </div>
54  </div>
55  </div>;
56
57  this.setButtonsEventHandlers();
58}

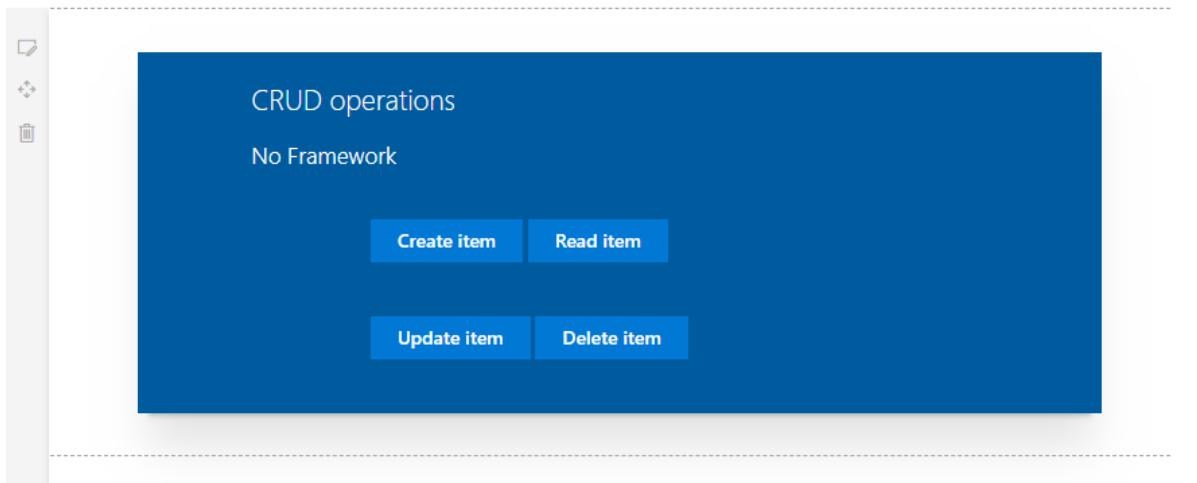
```

```

59
60 private setButtonsEventHandlers(): void {
61   const webPart: NoFrameworkCrudWebPart = this;
62   this.domElement.querySelector('button.create-Button').addEventListener('click', () => { webPart.createItem(); });
63   this.domElement.querySelector('button.read-Button').addEventListener('click', () => { webPart.readItem(); });
64   this.domElement.querySelector('button.update-Button').addEventListener('click', () => { webPart.updateItem(); });
65   this.domElement.querySelector('button.delete-Button').addEventListener('click', () => { webPart.deleteItem(); });
66 }
67
68 private createItem(): void {
69 }
70
71 private readItem(): void {
72 }
73
74 private updateItem(): void {
75 }
76
77 private deleteItem(): void {
78 }

```

3. In the command prompt type “gulp serve” to see the buttons on webpart.



4. We will perform read, update, and delete operations on latest item in SharePoint list. Let us implement generic method which will return the id of latest item from given list. We will use the REST API to query the list.

```
private getLatestItemId(): Promise<number> {
  return new Promise<number>((resolve: (itemId: number) => void, reject: (error: any) => void): void => {

    this.context.spHttpClient.get(` ${this.context.pageContext.web.absoluteUrl}
1}/_api/web/lists/getbytitle('${this.properties.listName}')/items?$order
by=Id desc&$top=1&$select=id`,
      SPHttpClient.configurations.v1,
    {
      headers: {
        'Accept': 'application/json;odata=nometadata',
        'odata-version': ''
      }
    })
    .then((response: SPHttpClientResponse): Promise<{ value: { Id: number }[] }> => {
      return response.json();
    }, (error: any): void => {
      reject(error);
    })
    .then((response: { value: { Id: number }[] }): void => {
      if (response.value.length === 0) {
        resolve(-1);
      }
    })
  });
}
```

```
        }
    }  
    else {
        resolve(response.value[0].Id);
    }
});  
});  
}
```

Implement Create Operation

First start by implementing create method, which will add an item to SharePoint list.

```
private createItem(): void {
  const body: string = JSON.stringify({
    'Title': `Item ${new Date()}`}
  });

this.context.spHttpClient.post(`.${this.context.pageContext.web.absoluteUrl}/_api
/web/lists/getbytitle('${this.properties.listName}')/items`,
SPHttpClient.configurations.v1,
{
  headers: {
    'Accept': 'application/json;odata=nometadata',
    'Content-type': 'application/json;odata=nometadata',
    'odata-version': ''
  },
  body: body
})
.then((response: SPHttpClientResponse): Promise<IListItem> => {
  return response.json();
})
.then((item: IListItem): void => {
  this.updateStatus(`Item '${item.Title}' (ID: ${item.Id}) successfully
created`);
}, (error: any): void => {
  this.updateStatus('Error while creating the item: ' + error);
});
}
```

Implement Read Operation

We will use REST API to read the latest item

```
private readItem(): void {
    this.getLatestItemId()
        .then((itemId: number): Promise<SPHttpClientResponse> => {
            if (itemId === -1) {
                throw new Error('No items found in the list');
            }

            this.updateStatus(`Loading information about item ID: ${itemId}...`);

            return
        this.context.spHttpClient.get(`${this.context.pageContext.web.absoluteUrl}/_api/web/lists/getbytitle('${this.properties.listName}')/items(${itemId})?$select=Title,Id`,
            SPHttpClient.configurations.v1,
            {
                headers: {
                    'Accept': 'application/json;odata=nometadata',
                    'odata-version': ''
                }
            });
        })
        .then((response: SPHttpClientResponse): Promise<IListItem> => {
            return response.json();
        })
        .then((item: IListItem): void => {
            this.updateStatus(`Item ID: ${item.Id}, Title: ${item.Title}`);
        }, (error: any): void => {
            this.updateStatus('Loading latest item failed with error: ' + error);
        });
}
```

Implement Update Operation

Firstly we will get the latest item and update it.

```

private updateItem(): void {
  let latestItemId: number = undefined;
  this.updateStatus('Loading latest item...');

  this.getLatestItemId()
    .then((itemId: number): Promise<SPHttpClientResponse> => {
      if (itemId === -1) {
        throw new Error('No items found in the list');
      }

      latestItemId = itemId;
      this.updateStatus(`Loading information about item ID: ${itemId}...`);

      return
    })
    .then(() => this.context.spHttpClient.get(`${this.context.pageContext.web.absoluteUrl}/_api/web/lists/getbytitle('${this.properties.listName}')/items(${latestItemId})?select=Title,Id`, SPHttpClient.configurations.v1,
    {
      headers: {
        'Accept': 'application/json;odata=nometadata',
        'odata-version': ''
      }
    }));
}

.then((response: SPHttpClientResponse): Promise<IListItem> => {
  return response.json();
})

.then((item: IListItem): void => {
  this.updateStatus(`Item ID1: ${item.Id}, Title: ${item.Title}`);

  const body: string = JSON.stringify({
    'Title': `Updated Item ${new Date()}`
  });

  this.context.spHttpClient.post(`${this.context.pageContext.web.absoluteUrl}/_api/web/lists/getbytitle('${this.properties.listName}')/items(${item.Id})`, SPHttpClient.configurations.v1,
  {
    headers: {
      'Accept': 'application/json;odata=nometadata',
    }
  });
})

```

```

        'Content-type': 'application/json;odata=nometadata',
        'odata-version': '',
        'IF-MATCH': '*',
        'X-HTTP-Method': 'MERGE'
    },
    body: body
})
.then((response: SPHttpClientResponse): void => {
    this.updateStatus(`Item with ID: ${latestItemId} successfully
updated`);
}, (error: any): void => {
    this.updateStatus(`Error updating item: ${error}`);
});
});
}
}

```

Implement Delete Operation

REST APIs are used to find and delete the latest item.

```

private deleteItem(): void {
    if (!window.confirm('Are you sure you want to delete the latest item?')) {
        return;
    }

    this.updateStatus('Loading latest items...');
    let latestItemId: number = undefined;
    let etag: string = undefined;
    this.getLatestItemId()
        .then((itemId: number): Promise<SPHttpClientResponse> => {
            if (itemId === -1) {
                throw new Error('No items found in the list');
            }

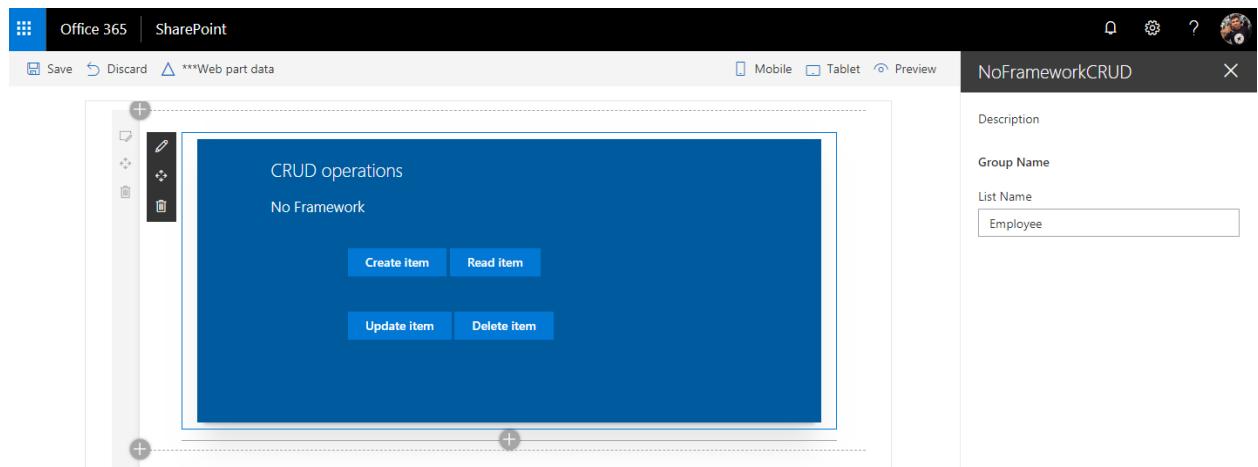
            latestItemId = itemId;
            this.updateStatus(`Loading information about item ID:
${latestItemId}...`);
            return
        })
        .then(() => {
            this.context.spHttpClient.get(`${this.context.pageContext.web.absoluteUrl}/_api
/web/lists/getbytitle('${this.properties.listName}')/items(${latestItemId})?$se
lect=Id`,
            SPHttpClient.configurations.v1,
            {
                headers: {

```

```
'Accept': 'application/json;odata=nometadata',
'odata-version': ''
})
});
})
.then((response: SPHttpClientResponse): Promise<IListItem> => {
  etag = response.headers.get('ETag');
  return response.json();
})
.then((item: IListItem): Promise<SPHttpClientResponse> => {
  this.updateStatus(`Deleting item with ID: ${latestItemId}...`);
  return
this.context.spHttpClient.post(`${this.context.pageContext.web.absoluteUrl}/_api/web/lists/getbytitle('${this.properties.listName}')/items(${item.Id})`,
  SPHttpClient.configurations.v1,
{
  headers: {
    'Accept': 'application/json;odata=nometadata',
    'Content-type': 'application/json;odata=verbose',
    'odata-version': '',
    'IF-MATCH': etag,
    'X-HTTP-Method': 'DELETE'
  }
});
})
.then((response: SPHttpClientResponse): void => {
  this.updateStatus(`Item with ID: ${latestItemId} successfully deleted`);
}, (error: any): void => {
  this.updateStatus(`Error deleting item: ${error}`);
});
});
```

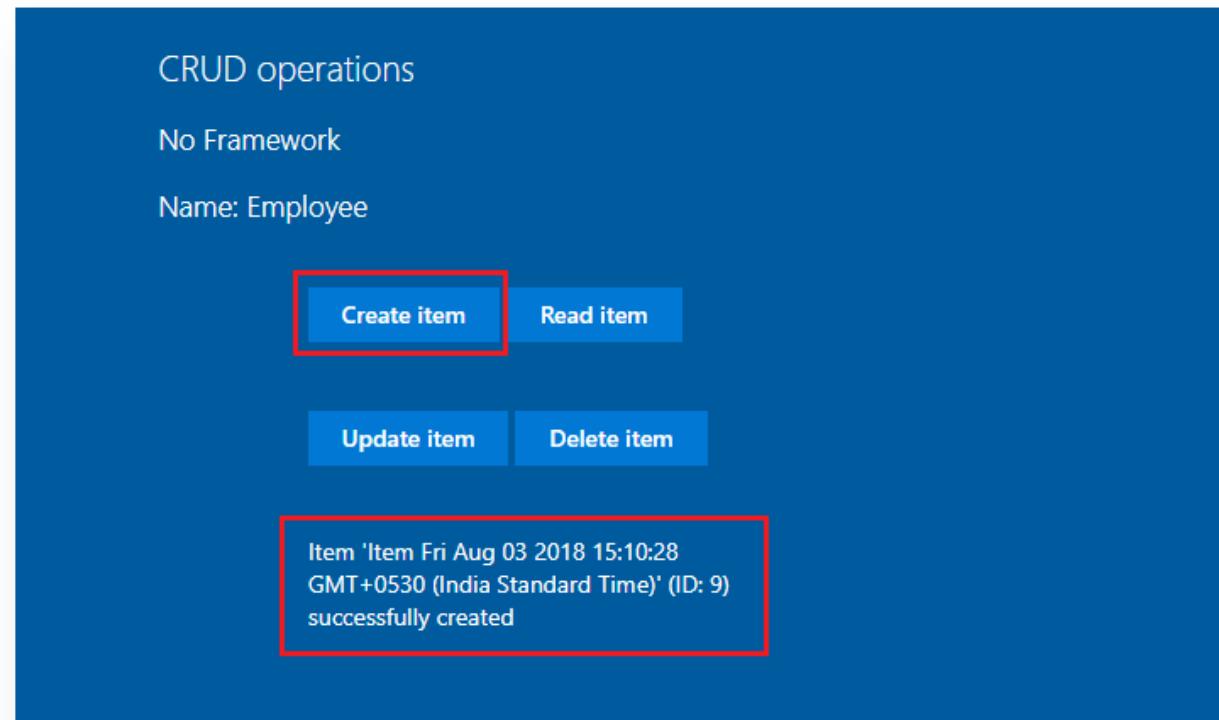
Test the WebPart

1. On the command prompt, type “gulp serve”
 2. Open SharePoint site
 3. Navigate to /_layouts/15/workbench.aspx
 4. Add the webpart to page.
 5. Edit webpart, in the properties pane type the list name

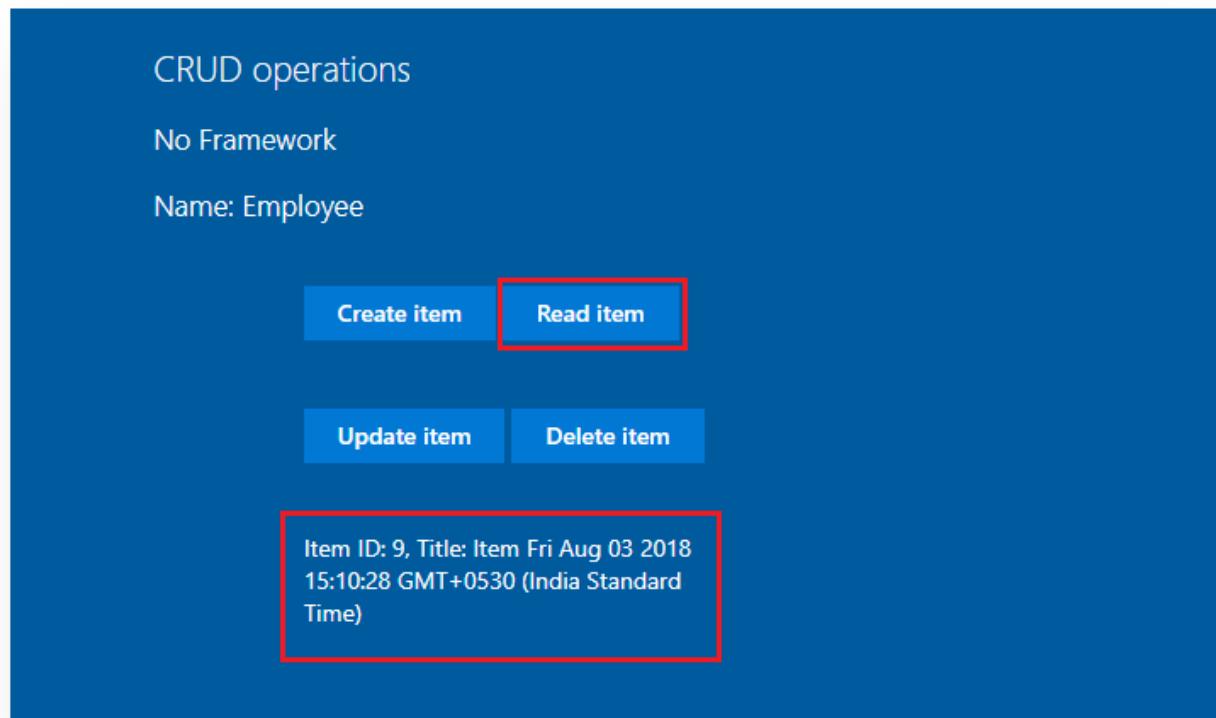


6. Click the buttons (Create Item, Read Item, Update Item, and Delete Item) one by one to test the webpart
7. Verify the operations are taking place in SharePoint list.

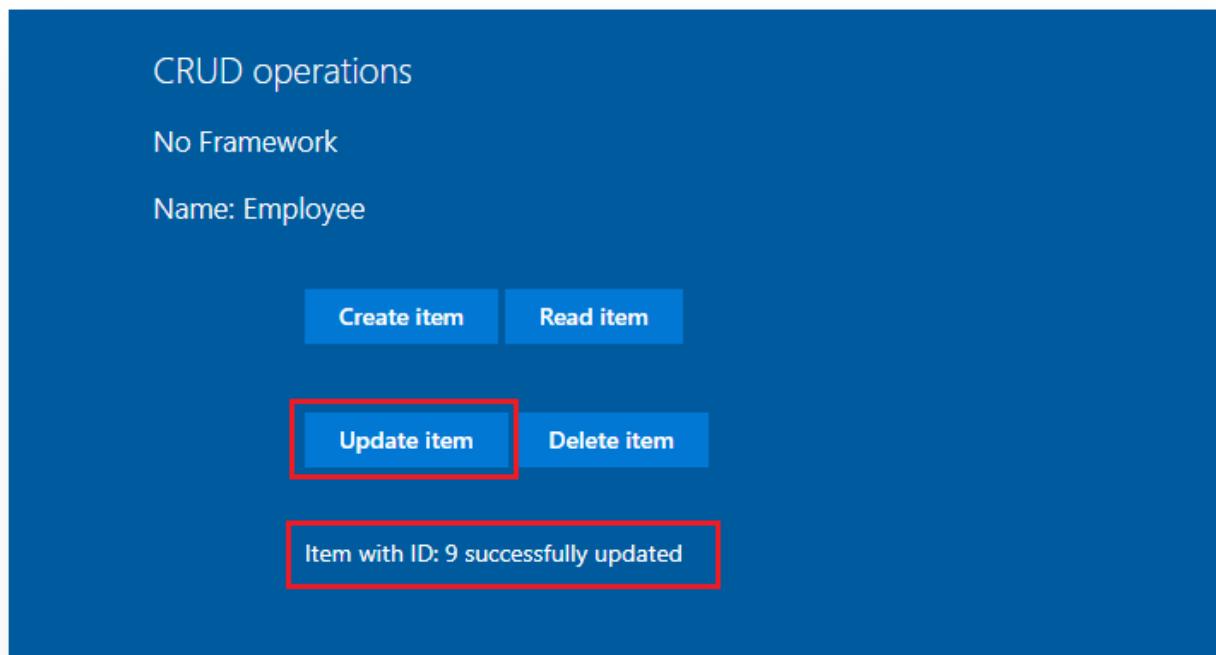
Create Operation



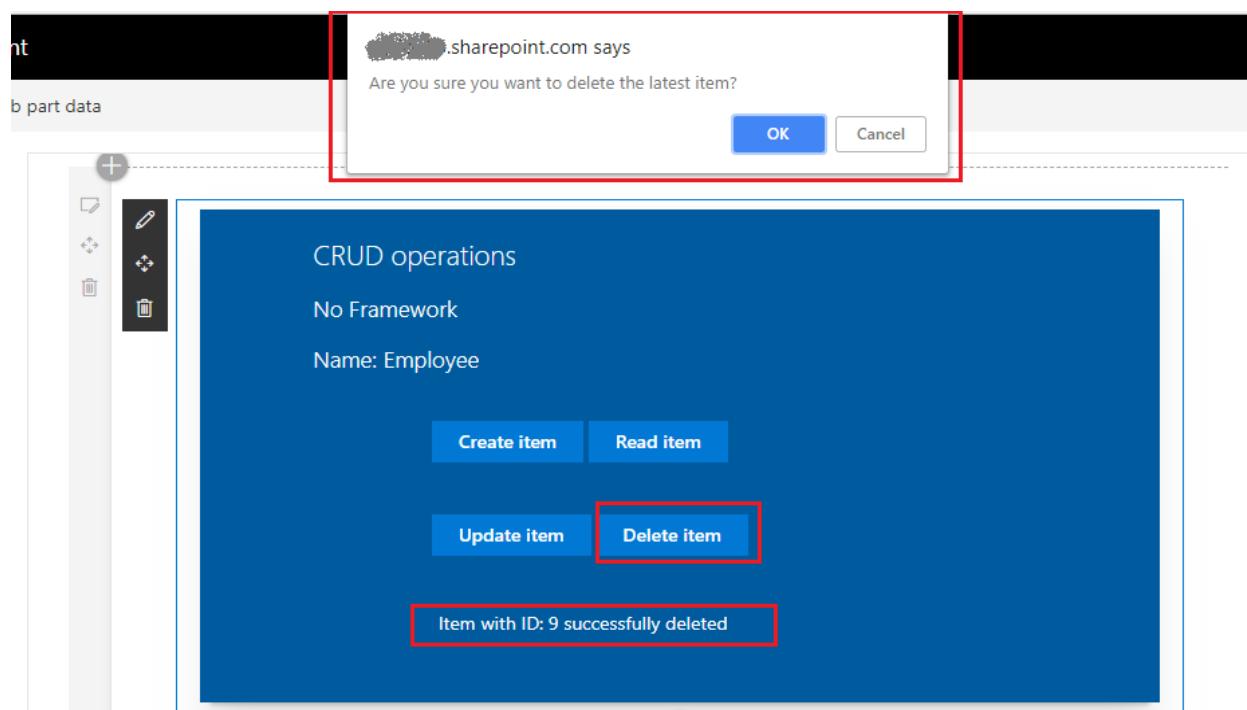
Read Operation



Update Operation

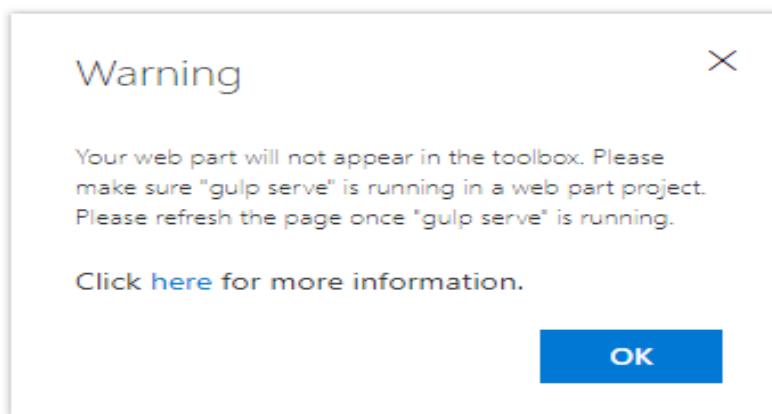


Delete Operation



Troubleshooting

In some cases SharePoint workbench ([https://\[tenant\].sharepoint.com/_layouts/15/workbench.aspx](https://[tenant].sharepoint.com/_layouts/15/workbench.aspx)) shows below error although “gulp serve” is running.



Open below url (<https://localhost:4321/temp/manifests.js>) in the next tab of browser. Accept the warning message.

Summary

SharePoint framework client web parts can be developed with No JavaScript options. REST APIs can be used to perform the CRUD operations on SharePoint list.

Chapter 11: CRUD operations using React JS

Overview

In the chapters so far, we have explored on developing basic SharePoint client web part which can run independently without any interaction with SharePoint.

In this chapter, we will explore to interact with the SharePoint list for CRUD (Create, Read, Update, and Delete) operations using React JS. React JS is natively supported by SharePoint Framework.

Brief about React JS

React is JavaScript library developed and licensed by Facebook. It represents V (View) in MVC (Model View Controller). React JS can react to changes in application state. SharePoint Framework itself is built using React JS. Read more about React JS at <https://reactjs.org/>

Create SPFx Solution

1. Open command prompt. Create a directory for SPFx solution

```
md spfx-crud-reactjs
```

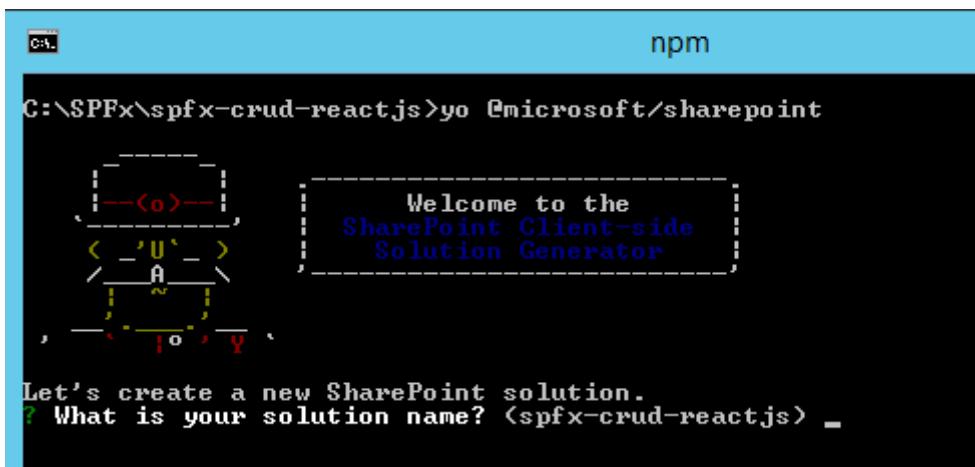
2. Navigate to above created directory

```
cd spfx-crud-reactjs
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



The screenshot shows a terminal window titled 'npm' with the command 'C:\SPFx\spfx-crud-reactjs>yo @microsoft/sharepoint' entered. A wizard interface is displayed, asking 'Let's create a new SharePoint solution.' followed by the question '? What is your solution name? <spfx-crud-reactjs> _'. The user has partially typed '(spfx-crud-reactjs)' into the input field.

Solution Name: Hit enter to have default name (spfx-crud-reactjs in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension. Choose webpart option.

Selected choice: WebPart

Web part name: Hit enter to select the default name or type in any other name.

Selected choice: ReactCRUD

Web part description: Hit enter to select the default description or type in any other value.

Selected choice: CRUD operations with React JS

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)

Selected choice: React

5. Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take significant amount of time.

6. Once the scaffolding process is completed, lock down the version of project dependencies by running below command

```
npm shrinkwrap
```

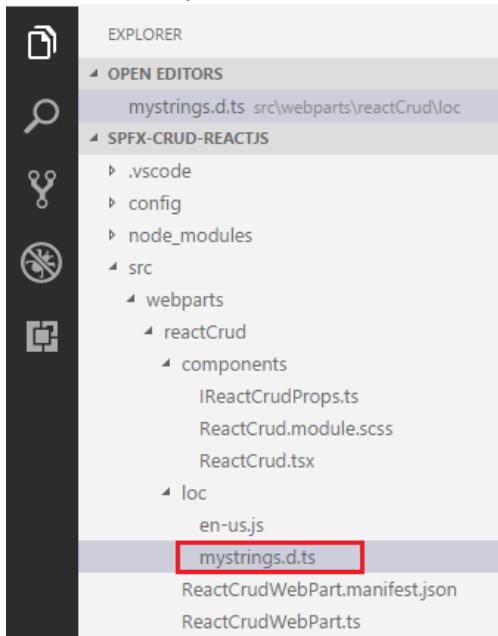
7. In the command prompt type below command to open the solution in code editor of your choice.

```
code .
```

Configure Property for List Name

SPFx solution by default have description property created. Let us change the property to list name. We will use this property to configure the list name on which the CRUD operation is to perform.

1. Open mystrings.d.ts under \src\webparts\reactCrud\loc\ folder
2. Rename DescriptionFieldLabel to ListNameFieldLabel

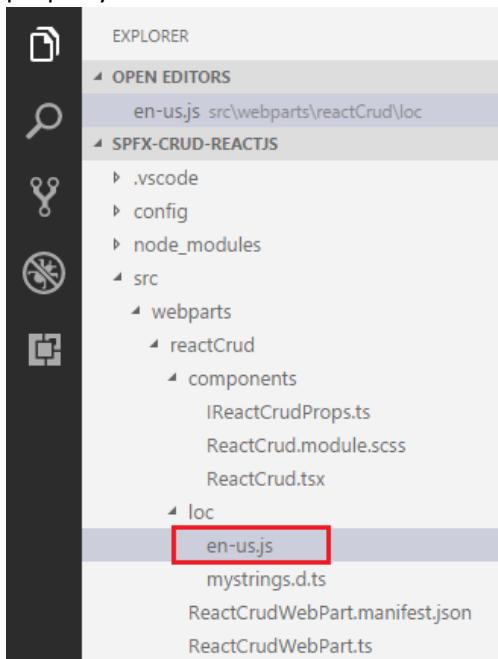


```
EXPLORER
OPEN EDITORS
mystrings.d.ts src\webparts\reactCrud\loc
SPFX-CRUD-REACTJS
.vscode
config
node_modules
src
webparts
reactCrud
components
IReactCrudProps.ts
ReactCrud.module.scss
ReactCrud.tsx
loc
en-us.js
mystrings.d.ts
ReactCrudWebPart.manifest.json
ReactCrudWebPart.ts
```

mystrings.d.ts

```
1 declare interface IReactCrudWebPartStrings {
2   PropertyPaneDescription: string;
3   BasicGroupName: string;
4   ListNameFieldLabel: string;
5 }
6
7 declare module 'ReactCrudWebPartStrings' {
8   const strings: IReactCrudWebPartStrings;
9   export = strings;
10 }
11
```

3. In en-us.js file under \src\webparts\reactCrud\loc\ folder set the display name for listName property

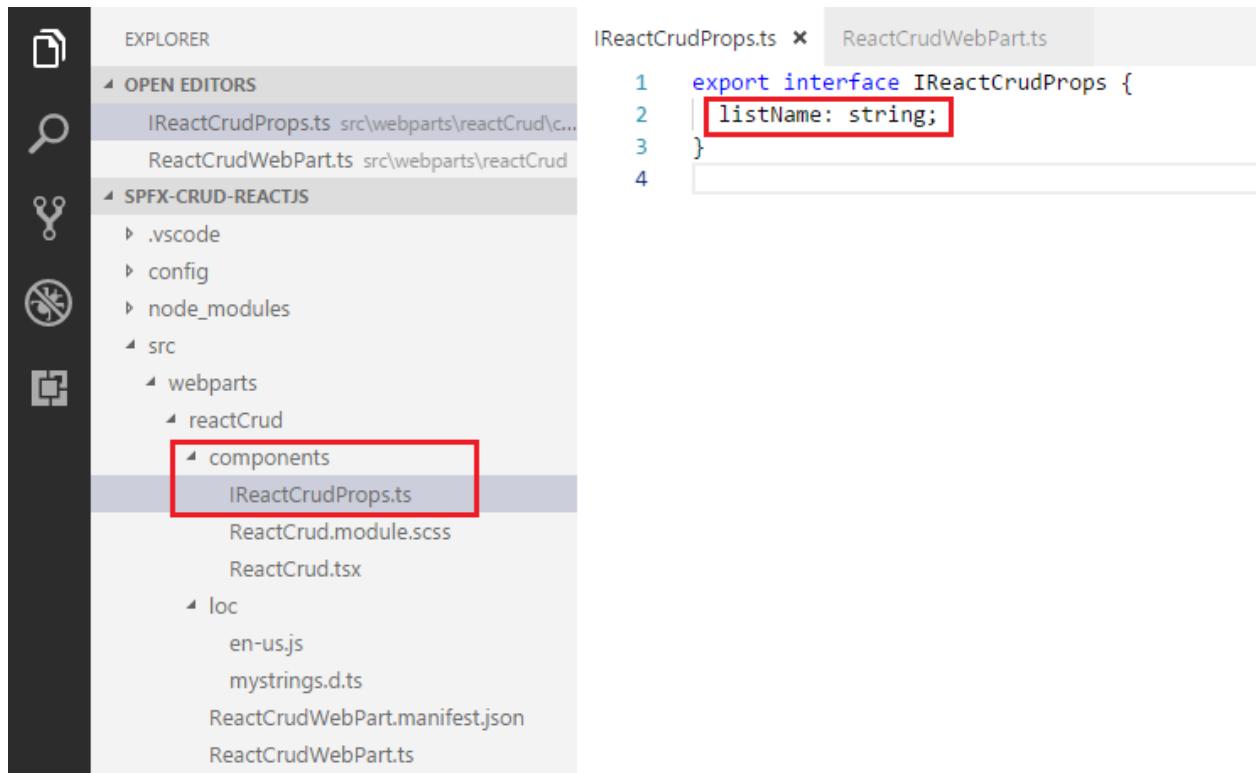


```
EXPLORER
OPEN EDITORS
en-us.js src\webparts\reactCrud\loc
SPFX-CRUD-REACTJS
.vscode
config
node_modules
src
webparts
reactCrud
components
IReactCrudProps.ts
ReactCrud.module.scss
ReactCrud.tsx
loc
en-us.js
mystrings.d.ts
ReactCrudWebPart.manifest.json
ReactCrudWebPart.ts
```

en-us.js

```
1 define([], function() {
2   return {
3     "PropertyPaneDescription": "Description",
4     "BasicGroupName": "Group Name",
5     "ListNameFieldLabel": "List Name"
6   };
7 });
8
```

4. In the interface IReactCrudProps.ts under \src\webparts\reactCrud\components\ set the member name to listName

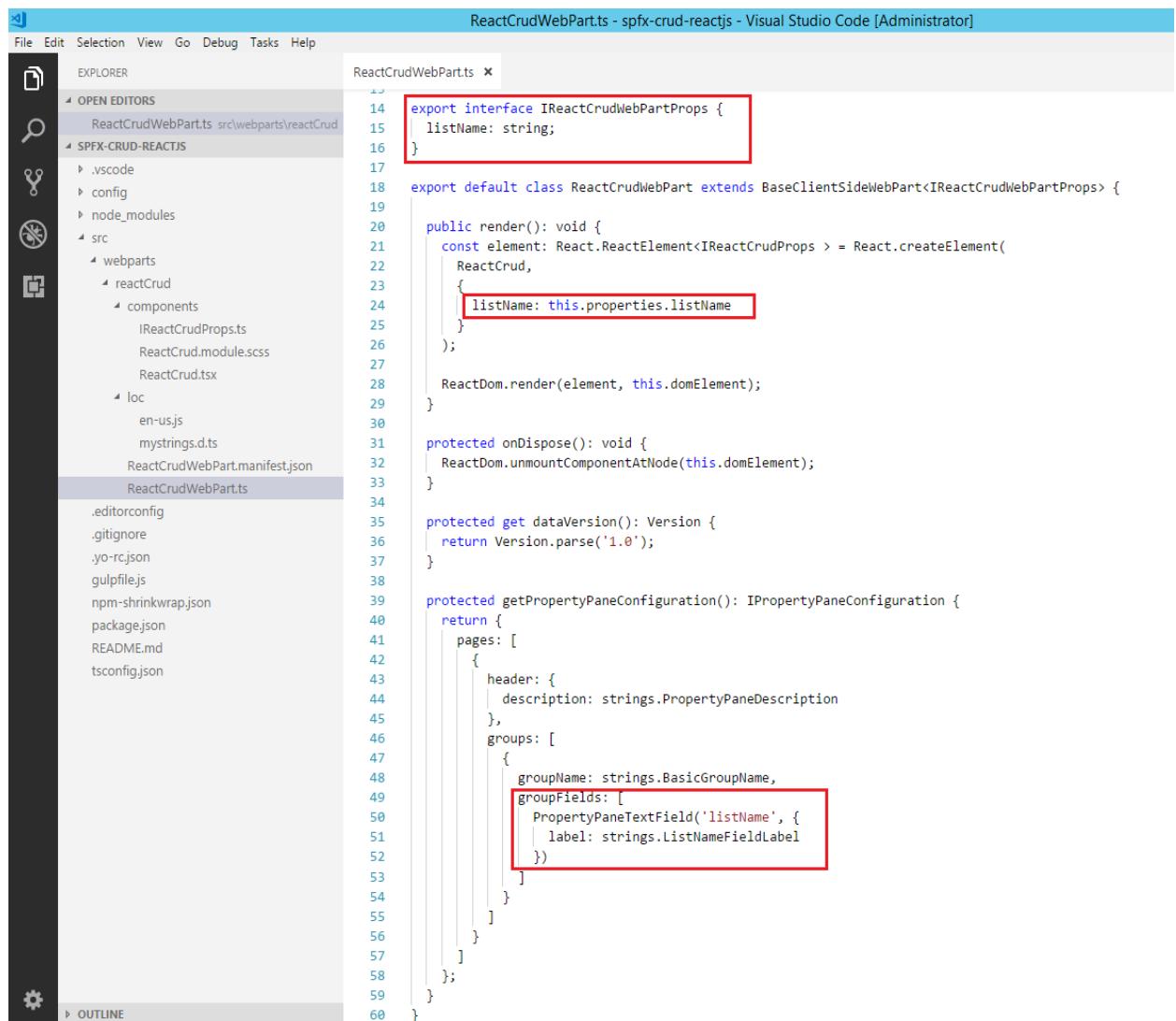


The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with a tree view of the project structure. The file `IReactCrudProps.ts` is selected in the tree. The main area is the code editor with the following content:

```
1  export interface IReactCrudProps {  
2      listName: string;  
3  }  
4
```

A red box highlights the line `listName: string;` in the code editor.

5. Open main webpart file (ReactCrudWebPart.ts) under \src\webparts\reactCrud folder.
6. Rename description property pane field to listName



```

File Edit Selection View Go Debug Tasks Help
RENDERER
OPEN EDITORS
SPFX-CRUD-REACTJS
  .vscode
  config
  node_modules
  src
    webparts
      reactCrud
        components
          IReactCrudProps.ts
          ReactCrud.module.scss
          ReactCrud.tsx
        loc
          en-us.js
          mystrings.d.ts
          ReactCrudWebPart.manifest.json
          ReactCrudWebPart.ts
.editorconfig
.gitignore
.yo-rc.json
gulpfile.js
npm-shrinkwrap.json
package.json
README.md
tsconfig.json
OUTLINE

```

```

export interface IReactCrudWebPartProps {
  listName: string;
}

export default class ReactCrudWebPart extends BaseClientSideWebPart<IReactCrudWebPartProps> {
  public render(): void {
    const element: React.ReactNode = React.createElement(
      ReactCrud,
      {
        listName: this.properties.listName
      }
    );
    ReactDOM.render(element, this.domElement);
  }

  protected onDispose(): void {
    ReactDOM.unmountComponentAtNode(this.domElement);
  }

  protected getDataVersion(): Version {
    return Version.parse('1.0');
  }

  protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
    return {
      pages: [
        {
          header: {
            description: strings.PropertyPaneDescription
          },
          groups: [
            {
              groupName: strings.BasicGroupName,
              groupFields: [
                PropertyPaneTextField('listName', {
                  label: strings.ListNameFieldLabel
                })
              ]
            }
          ]
        };
      }
    };
  }
}

```

- The UI in React gets served from component ReactCrud.tsx under \src\webparts\reactCrud\components\ReactCrud.tsx. Make the changes for listName property in the component.

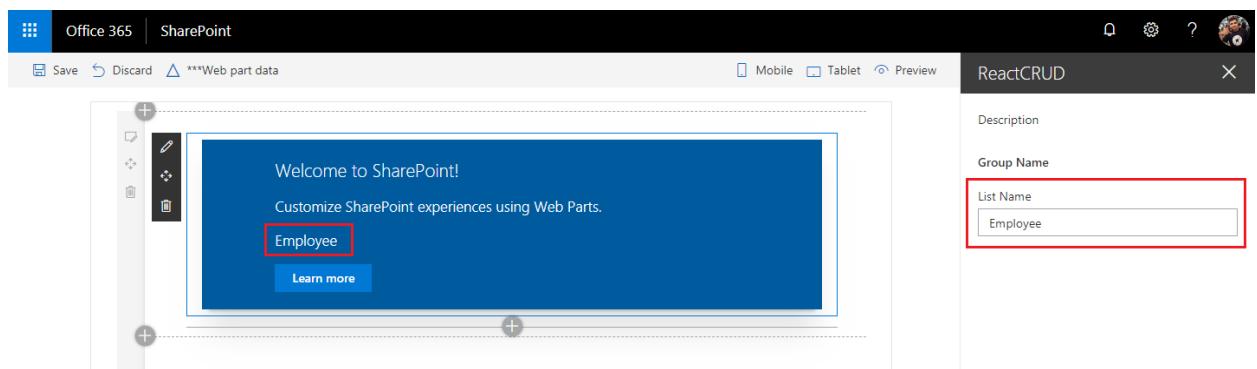


```

1 import * as React from 'react';
2 import styles from './ReactCrud.module.scss';
3 import { IReactCrudProps } from './IReactCrudProps';
4 import { escape } from '@microsoft/sp-lodash-subset';
5
6 export default class ReactCrud extends React.Component<IReactCrudProps, {}> {
7   public render(): React.ReactNode<IReactCrudProps> {
8     return (
9       <div className={ styles.reactCrud }>
10         <div className={ styles.container }>
11           <div className={ styles.row }>
12             <div className={ styles.column }>
13               <span className={ styles.title }>Welcome to SharePoint!</span>
14               <p className={ styles.subTitle }>Customize SharePoint experiences using Web Parts.</p>
15               <p className={ styles.description }>{escape(this.props.listName)}</p>
16               <a href="https://aka.ms/spfx" className={ styles.button }>
17                 <span className={ styles.label }>Learn more</span>
18               </a>
19             </div>
20           </div>
21         </div>
22       );
23     }
24   }
25 }

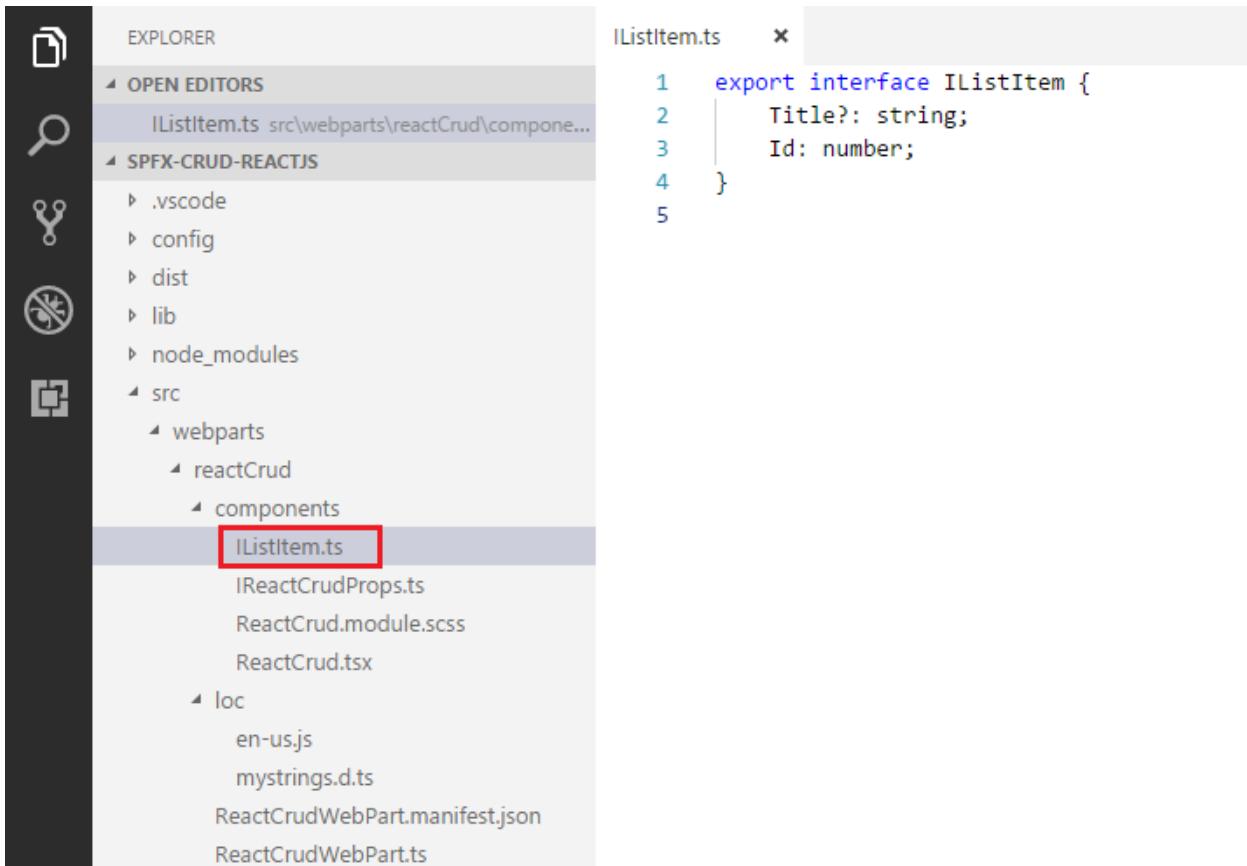
```

8. In the command prompt, type “gulp serve”
9. In the SharePoint local workbench page, add the web part.
10. Edit the web part to ensure the listName property pane field is getting reflected.



Model for List Item

1. Add a class (`IListItem.ts`) representing the list item.



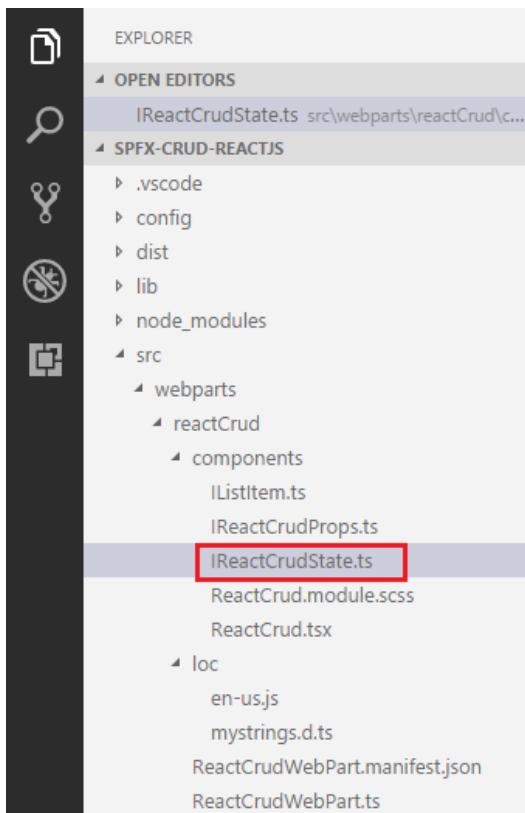
The screenshot shows the VS Code interface. On the left is the Explorer sidebar with a tree view of the project structure:

- OPEN EDITORS
- SPFX-CRUD-REACTJS
 - .vscode
 - config
 - dist
 - lib
 - node_modules
 - src
 - webparts
 - reactCrud
 - IListItem.ts
 - IReactCrudProps.ts
 - ReactCrud.module.scss
 - ReactCrud.tsx
 - components
 - loc
 - en-us.js
 - mystrings.d.ts
 - ReactCrudWebPart.manifest.json
 - ReactCrudWebPart.ts

The file `IListItem.ts` is selected and highlighted with a red box in the Explorer view. The main editor window on the right contains the following TypeScript code:

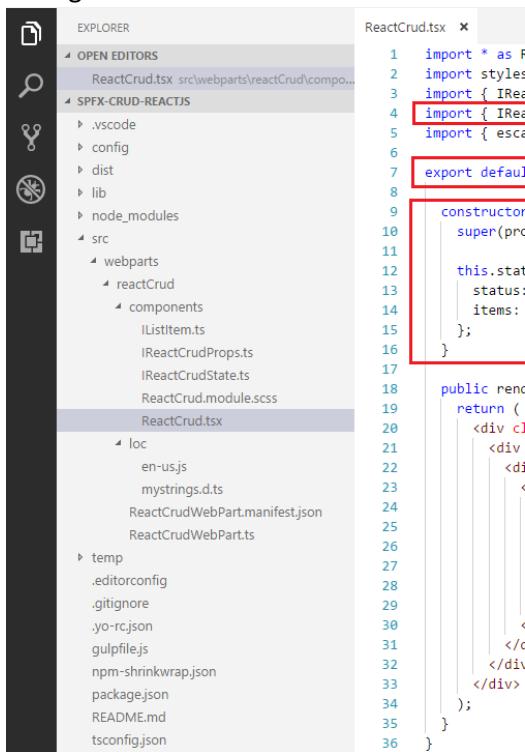
```
1 export interface IListItem {  
2     Title?: string;  
3     Id: number;  
4 }  
5
```

2. React JS acts on the state change. Let us add state to our solution.



```
IReactCrudState.ts ✘
1 import { IListItem } from './IListItem';
2
3 export interface IReactCrudState {
4   status: string;
5   items: IListItem[];
6 }
7
```

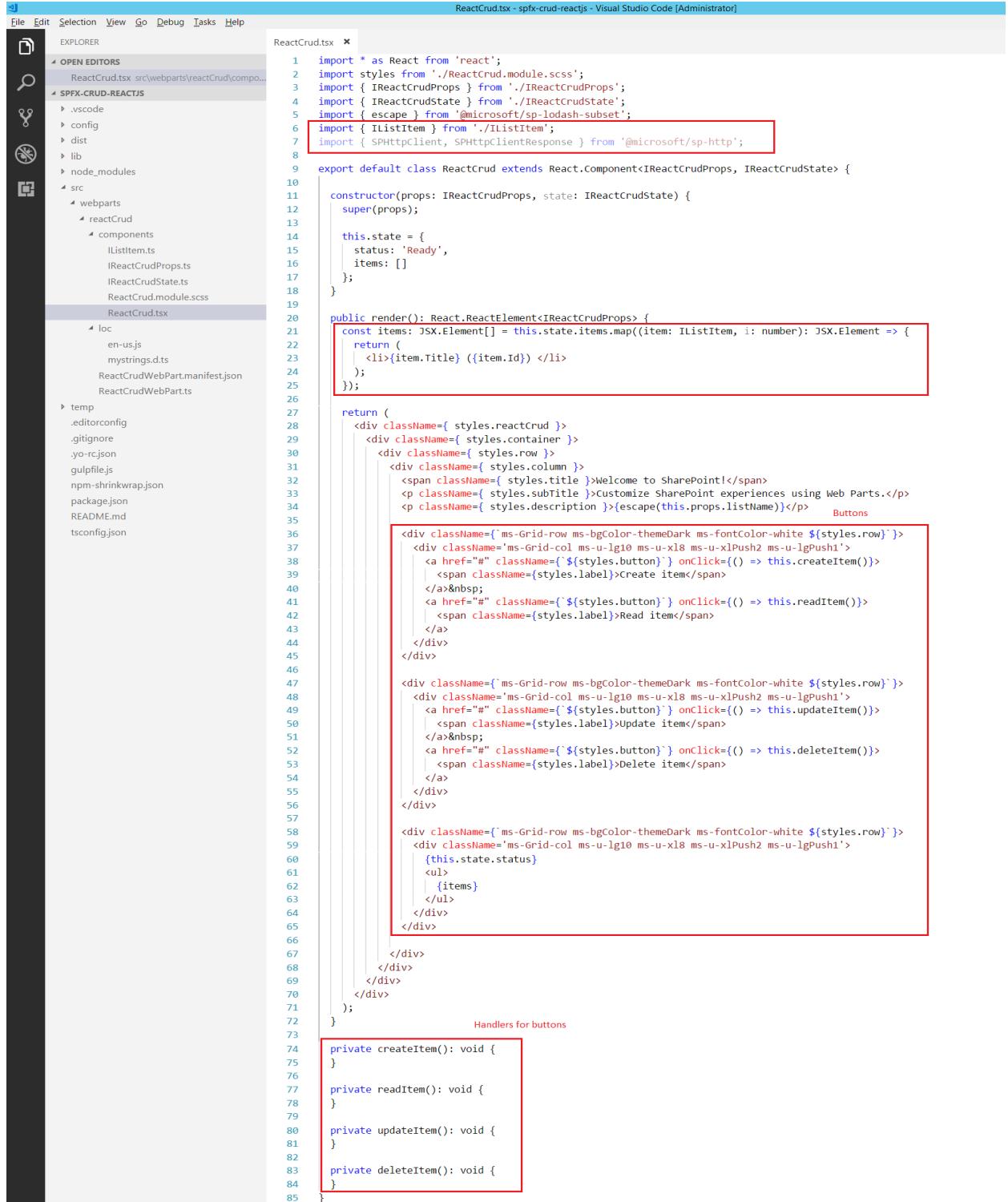
3. Configure ReactCrud.tsx for this state



```
ReactCrud.tsx ✘
1 import * as React from 'react';
2 import styles from './ReactCrud.module.scss';
3 import { IReactCrudProps } from './IReactCrudProps';
4 import { IReactCrudState } from './IReactCrudState';
5 import { escape } from '@microsoft/sp-lodash-subset';
6
7 export default class ReactCrud extends React.Component<IReactCrudProps, IReactCrudState> {
8
9   constructor(props: IReactCrudProps, state: IReactCrudState) {
10     super(props);
11
12     this.state = {
13       status: 'Ready',
14       items: []
15     };
16   }
17
18   public render(): React.ReactElement<IReactCrudProps> {
19     return (
20       <div className={ styles.reactCrud }>
21         <div className={ styles.container }>
22           <div className={ styles.row }>
23             <div className={ styles.column }>
24               <span className={ styles.title }>Welcome to SharePoint!</span>
25               <p className={ styles.subtitle }>Customize SharePoint experiences using Web Parts.</p>
26               <p className={ styles.description }>{escape(this.props.listName)}</p>
27               <a href="https://aka.ms/spfx" className={ styles.button }>
28                 <span className={ styles.label }>Learn more</span>
29               </a>
30             </div>
31           </div>
32         </div>
33       );
34     }
35   }
36 }
```

Add Controls to WebPart

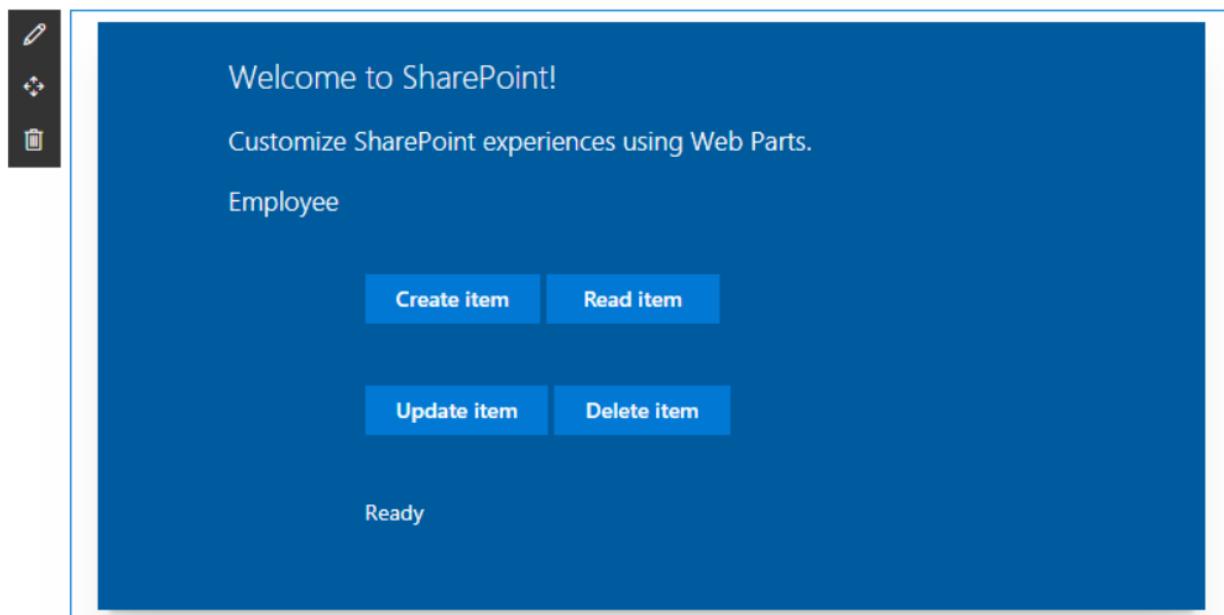
1. Open ReactCrud.tsx under \src\webparts\reactCrud\components\ReactCrud.tsxfolder.
2. Modify Render method to include buttons for CRUD operations and add event handlers to each of the button



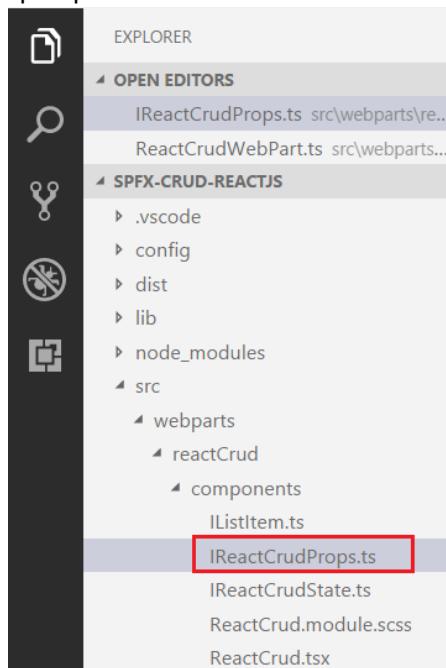
```

File Edit Selection View Go Debug Tasks Help
REACTCRUD-REACTJS
  .vscode
  config
  dist
  lib
  node_modules
  src
    webparts
      reactCrud
        components
          IListItem.ts
          IReactCrudProps.ts
          IReactCrudState.ts
          ReactCrud.module.scss
        ReactCrud.tsx
    loc
      en-us.js
      mystrings.dts
      ReactCrudWebPart.manifest.json
      ReactCrudWebPart.ts
    temp
      .editorconfig
      .gitignore
      yo-rc.json
      gulpfile.js
      npm-shrinkwrap.json
      package.json
      README.md
      tsconfig.json
  ReactCrud.tsx x
  1 import * as React from 'react';
  2 import styles from './ReactCrud.module.scss';
  3 import { IReactCrudProps } from './IReactcrudProps';
  4 import { IReactCrudState } from './IReactcrudstate';
  5 import { escape } from '@microsoft/sp-lodash-subset';
  6 import { IListItem } from './IListItem';
  7 import { SPHttpClient, SPHttpClientResponse } from '@microsoft/sp-http';
  8
  9 export default class ReactCrud extends React.Component<IReactCrudProps, IReactCrudState> {
 10
 11   constructor(props: IReactCrudProps, state: IReactCrudState) {
 12     super(props);
 13
 14     this.state = {
 15       status: 'Ready',
 16       items: []
 17     };
 18   }
 19
 20   public render(): React.ReactNode<IReactCrudProps> {
 21     const items: JSX.Element[] = this.state.items.map((item: IListItem, i: number): JSX.Element => {
 22       return (
 23         <li>{item.Title} ({item.Id}) </li>
 24       );
 25     });
 26
 27     return (
 28       <div className={ styles.reactCrud }>
 29         <div className={ styles.container }>
 30           <div className={ styles.row }>
 31             <div className={ styles.column }>
 32               <span className={ styles.title }>Welcome to SharePoint!</span>
 33               <p className={ styles.subtitle }>Customize SharePoint experiences using Web Parts.</p>
 34               <p className={ styles.description }>{escape(this.props.listName)}</p>
 35             </div>
 36             <div className="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}>
 37               <div className="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
 38                 <a href="#" className={`${styles.button}`} onClick={() => this.createItem()}>
 39                   <span className={styles.label}>Create item</span>
 40                 </a>&nbsp;
 41                 <a href="#" className={`${styles.button}`} onClick={() => this.readItem()}>
 42                   <span className={styles.label}>Read item</span>
 43                 </a>
 44               </div>
 45             </div>
 46             <div className="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}>
 47               <div className="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
 48                 <a href="#" className={`${styles.button}`} onClick={() => this.updateItem()}>
 49                   <span className={styles.label}>Update item</span>
 50                 </a>&nbsp;
 51                 <a href="#" className={`${styles.button}`} onClick={() => this.deleteItem()}>
 52                   <span className={styles.label}>Delete item</span>
 53                 </a>
 54               </div>
 55             </div>
 56             <div className="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}>
 57               <div className="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
 58                 {this.state.status}
 59                 <ul>
 60                   {items}
 61                 </ul>
 62               </div>
 63             </div>
 64             <div>
 65               <div>
 66                 <div>
 67                   <div>
 68                     <div>
 69                       <div>
 70                         <div>
 71                           <div>
 72                             <div>
 73                               <div>
 74                                 <private createItem(): void {>
 75                                 </private>
 76                                 <private readItem(): void {>
 77                                 </private>
 78                                 <private updateItem(): void {>
 79                                 </private>
 80                                 <private deleteItem(): void {>
 81                                 </private>
 82                               </div>
 83                             </div>
 84                           </div>
 85                         </div>
 86                       </div>
 87                     </div>
 88                   </div>
 89                 </div>
 90               </div>
 91             </div>
 92           </div>
 93         </div>
 94       </div>
 95     );
 96   }
 97
 98   Handlers for buttons
 99
 100  private createItem(): void {
 101  }
 102
 103  private readItem(): void {
 104  }
 105
 106  private updateItem(): void {
 107  }
 108
 109  private deleteItem(): void {
 110  }
 111}
  
```

3. In the command prompt type “gulp serve” to see the buttons on webpart.



4. The read, update and delete operations will be performed on latest item. Update interface IReactCrudProps.ts at \src\webparts\reactCrud\components\ to include site url and spHttpClient



```
IReactCrudProps.ts x ReactCrudWebPart.ts
1 import { SPHttpClient } from '@microsoft/sp-http';
2
3 export interface IReactCrudProps {
4   listName: string;
5   spHttpClient: SPHttpClient;
6   siteUrl: string;
7 }
8
```

5. Update \src\webparts\reactCrud\ReactCrudWebPart.ts to initiate site url and spHttpClient



```

17
18  export default class ReactCrudWebPart extends BaseClientSideWebPart<IReactCrudWebPartProps> {
19
20    public render(): void {
21      const element: React.ReactElement<IReactCrudProps> = React.createElement(
22        ReactCrud,
23        {
24          listName: this.properties.listName,
25          spHttpClient: this.context.spHttpClient,
26          siteUrl: this.context.pageContext.web.absoluteUrl
27        }
28      );
29
30      ReactDOM.render(element, this.domElement);
31    }
32
33    protected onDispose(): void {
34      ReactDOM.unmountComponentAtNode(this.domElement);
35    }
}

```

6. Let us implement generic method which will return the id of latest item from given list.

```

private getLatestItemId(): Promise<number> {
  return new Promise<number>((resolve: (itemId: number) => void, reject: (error: any) => void) => {

    this.props.spHttpClient.get(`${this.props.siteUrl}/_api/web/lists/getbytitle('${this.props.listName}')/items?$orderby=Id
desc&$top=1&$select=id`,
      SPHttpClient.configurations.v1,
    {
      headers: {
        'Accept': 'application/json;odata=nometadata',
        'odata-version': ''
      }
    })
    .then((response: SPHttpClientResponse): Promise<{ value: { Id: number }[] }> => {
      return response.json();
    }, (error: any): void => {
      reject(error);
    })
    .then((response: { value: { Id: number }[] }): void => {
      if (response.value.length === 0) {
        resolve(-1);
      }
      else {
        resolve(response.value[0].Id);
      }
    })
  })
}

```

```

        });
    });
}

```

Implement Create Operation

We will use the REST API to add the item to list.

```

private createItem(): void {
  this.setState({
    status: 'Creating item...',
    items: []
  });

  const body: string = JSON.stringify({
    'Title': `Item ${new Date()}`
  });

  this.props.spHttpClient.post(`${this.props.siteUrl}/_api/web/lists/getbytitle('${this.props.listName}')/items`,
    SPHttpClient.configurations.v1,
  {
    headers: {
      'Accept': 'application/json;odata=nometadata',
      'Content-type': 'application/json;odata=nometadata',
      'odata-version': ''
    },
    body: body
  })
  .then((response: SPHttpClientResponse): Promise<IListItem> => {
    return response.json();
  })
  .then((item: IListItem): void => {
    this.setState({
      status: `Item '${item.Title}' (ID: ${item.Id}) successfully created`,
      items: []
    });
  }, (error: any): void => {
    this.setState({
      status: 'Error while creating the item: ' + error,
      items: []
    });
  });
}

```

Implement Read Operation

We will use REST API to read the latest item.

```

private readItem(): void {
    this.setState({
        status: 'Loading latest items...',
        items: []
    });

    this.getLatestItemId()
        .then((itemId: number): Promise<SPHttpClientResponse> => {
            if (itemId === -1) {
                throw new Error('No items found in the list');
            }

            this.setState({
                status: `Loading information about item ID: ${itemId}...`,
                items: []
            });
            return
        })
        .then(() => {
            this.props.spHttpClient.get(`${this.props.siteUrl}/_api/web/lists/getbytitle('${this.props.listName}')/items(${itemId})?$select=Title,Id`,
                SPHttpClient.configurations.v1,
            {
                headers: {
                    'Accept': 'application/json;odata=nometadata',
                    'odata-version': ''
                }
            });
        })
        .then((response: SPHttpClientResponse): Promise<IListItem> => {
            return response.json();
        })
        .then((item: IListItem): void => {
            this.setState({
                status: `Item ID: ${item.Id}, Title: ${item.Title}`,
                items: []
            });
        }, (error: any): void => {
            this.setState({
                status: 'Loading latest item failed with error: ' + error,
                items: []
            });
        });
}

```

Implement Update Operation

We will use REST API to update the latest item.

```

private updateItem(): void {
  this.setState({
    status: 'Loading latest items...',
    items: []
  });

  let latestItemId: number = undefined;

  this.getLatestItemId()
    .then((itemId: number): Promise<SPHttpClientResponse> => {
      if (itemId === -1) {
        throw new Error('No items found in the list');
      }

      latestItemId = itemId;
      this.setState({
        status: `Loading information about item ID: ${latestItemId}...`,
        items: []
      });
    });

  return
  this.props.spHttpClient.get(`${this.props.siteUrl}/_api/web/lists/getbytitle('${this.props.listName}')/items(${latestItemId})?$select=Title,Id`,
    SPHttpClient.configurations.v1,
    {
      headers: {
        'Accept': 'application/json;odata=nometadata',
        'odata-version': ''
      }
    });
}.then((response: SPHttpClientResponse): Promise<IListItem> => {
  return response.json();
})
.then((item: IListItem): void => {
  this.setState({
    status: 'Loading latest items...',
    items: []
  });

  const body: string = JSON.stringify({
    'Title': `Updated Item ${new Date()}`

}

```

```

});
```

```

this.props.spHttpClient.post(` ${this.props.siteUrl}/_api/web/lists/getbytitle('
${this.props.listName}')/items(${item.Id})` ,
    SPHttpClient.configurations.v1,
{
    headers: {
        'Accept': 'application/json;odata=nometadata',
        'Content-type': 'application/json;odata=nometadata',
        'odata-version': '',
        'IF-MATCH': '*',
        'X-HTTP-Method': 'MERGE'
    },
    body: body
})
.then((response: SPHttpClientResponse): void => {
    this.setState({
        status: `Item with ID: ${latestItemId} successfully updated`,
        items: []
    });
}, (error: any): void => {
    this.setState({
        status: `Error updating item: ${error}`,
        items: []
    });
});
});
```

}

Implement Delete Operation

We will use REST API to delete the latest item.

```

private deleteItem(): void {
    if (!window.confirm('Are you sure you want to delete the latest item?')) {
        return;
    }

    this.setState({
        status: 'Loading latest items...',
        items: []
    });

    let latestItemId: number = undefined;
```

```

let etag: string = undefined;
this.getLatestItemId()
.then((itemId: number): Promise<SPHttpClientResponse> => {
  if (itemId === -1) {
    throw new Error('No items found in the list');
  }

  latestItemId = itemId;
  this.setState({
    status: `Loading information about item ID: ${latestItemId}...`,
    items: []
  });

  return
this.props.spHttpClient.get(`${this.props.siteUrl}/_api/web/lists/getbytitle('${this.props.listName}')/items(${latestItemId})?$select=Id`,
  SPHttpClient.configurations.v1,
{
  headers: {
    'Accept': 'application/json;odata=nometadata',
    'odata-version': ''
  }
});
.then((response: SPHttpClientResponse): Promise<IListItem> => {
  etag = response.headers.get('ETag');
  return response.json();
})
.then((item: IListItem): Promise<SPHttpClientResponse> => {
  this.setState({
    status: `Deleting item with ID: ${latestItemId}...`,
    items: []
  });

  return
this.props.spHttpClient.post(`${this.props.siteUrl}/_api/web/lists/getbytitle('${this.props.listName}')/items(${item.Id})`,
  SPHttpClient.configurations.v1,
{
  headers: {
    'Accept': 'application/json;odata=nometadata',
    'Content-type': 'application/json;odata=verbose',
    'odata-version': '',
    'IF-MATCH': etag,
    'X-HTTP-Method': 'DELETE'
  }
});

```

```

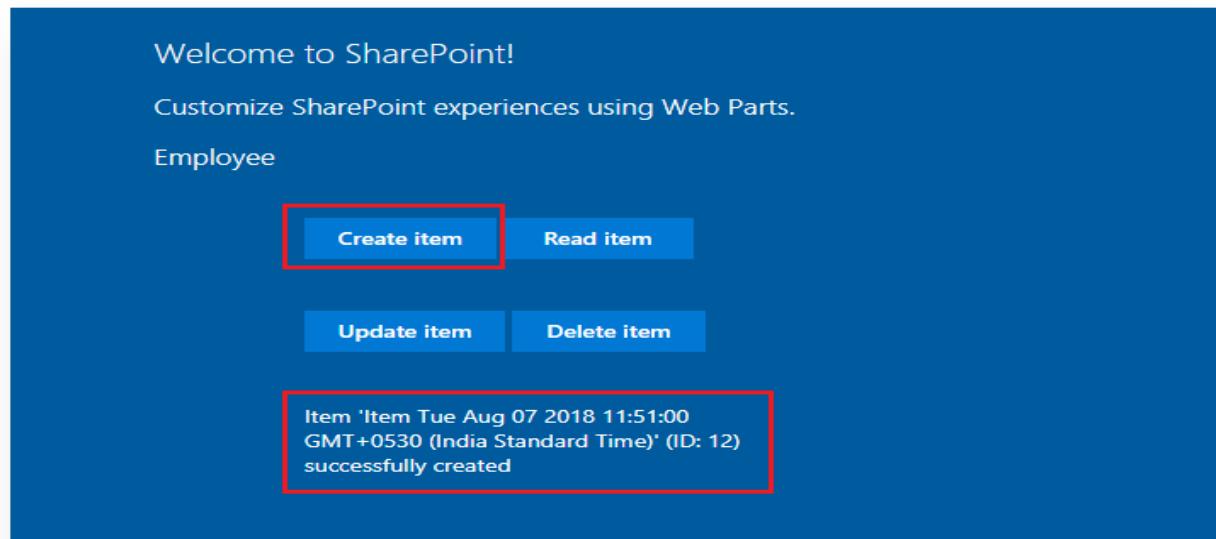
        }
    });
})
.then((response: SPHttpClientResponse): void => {
    this.setState({
        status: `Item with ID: ${latestItemId} successfully deleted`,
        items: []
    });
}, (error: any): void => {
    this.setState({
        status: `Error deleting item: ${error}`,
        items: []
    });
});
}
}

```

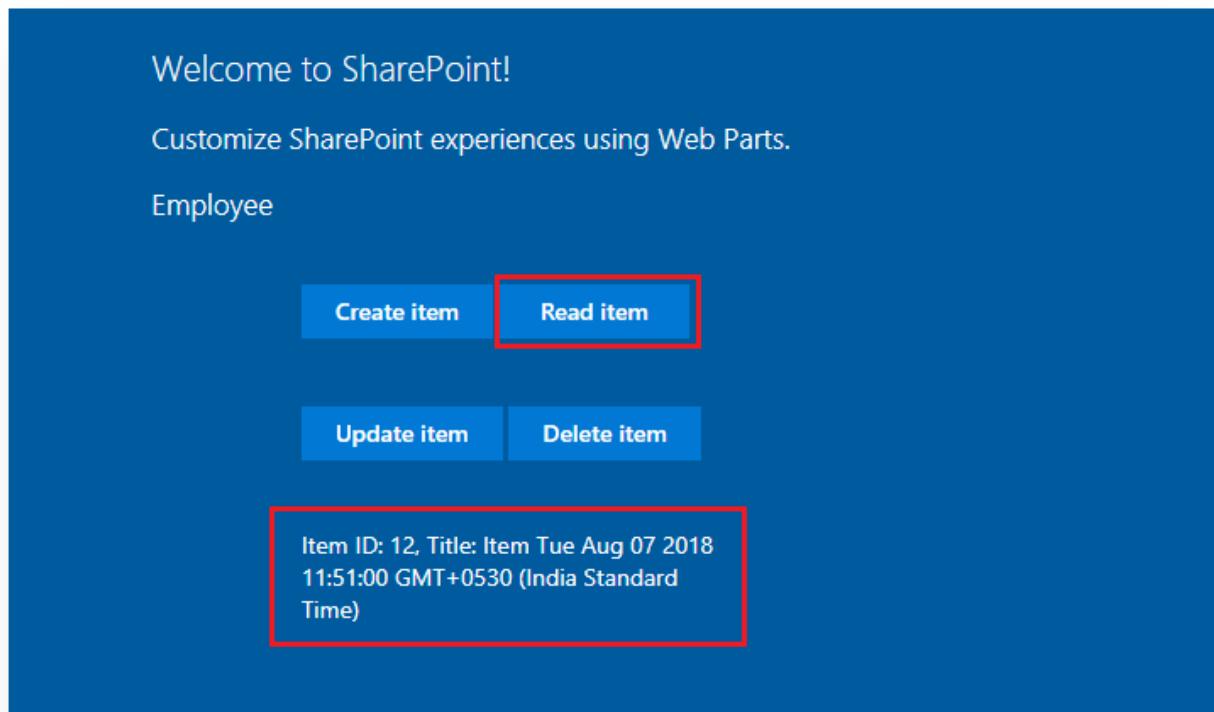
Test the WebPart

1. On the command prompt, type “gulp serve”
2. Open SharePoint site
3. Navigate to /_layouts/15/workbench.aspx
4. Add the webpart to page.
5. Edit webpart, in the properties pane type the list name
6. Click the buttons (Create Item, Read Item, Update Item, and Delete Item) one by one to test the webpart
7. Verify the operations are taking place in SharePoint list.

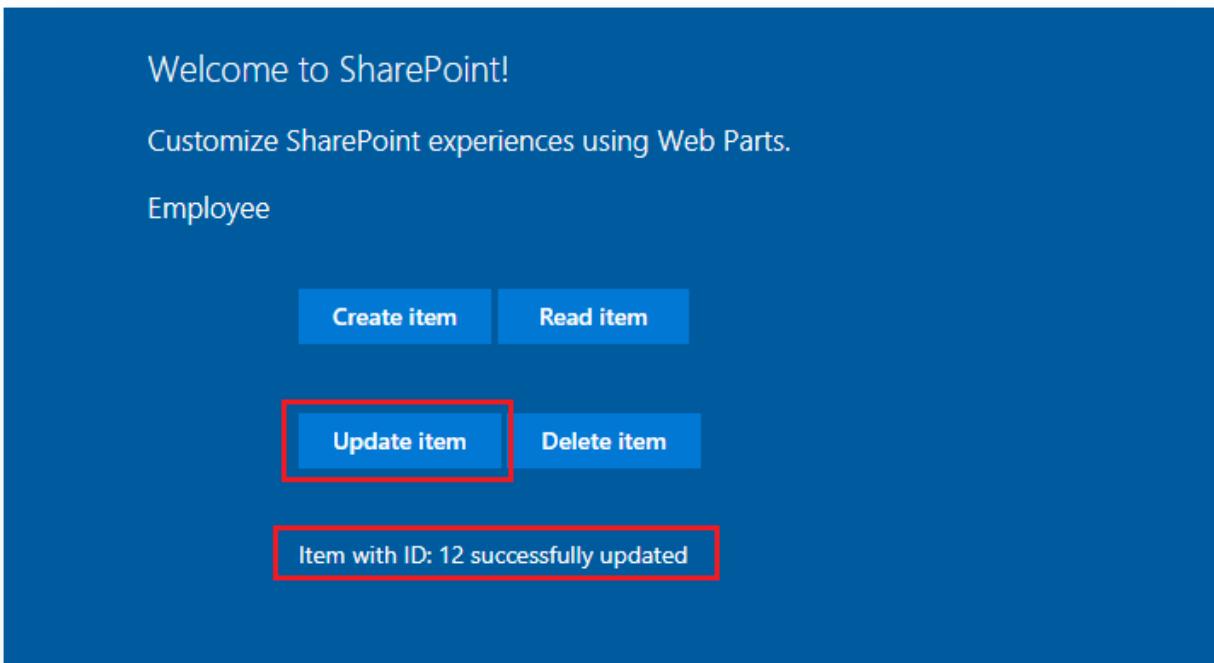
Create Operation



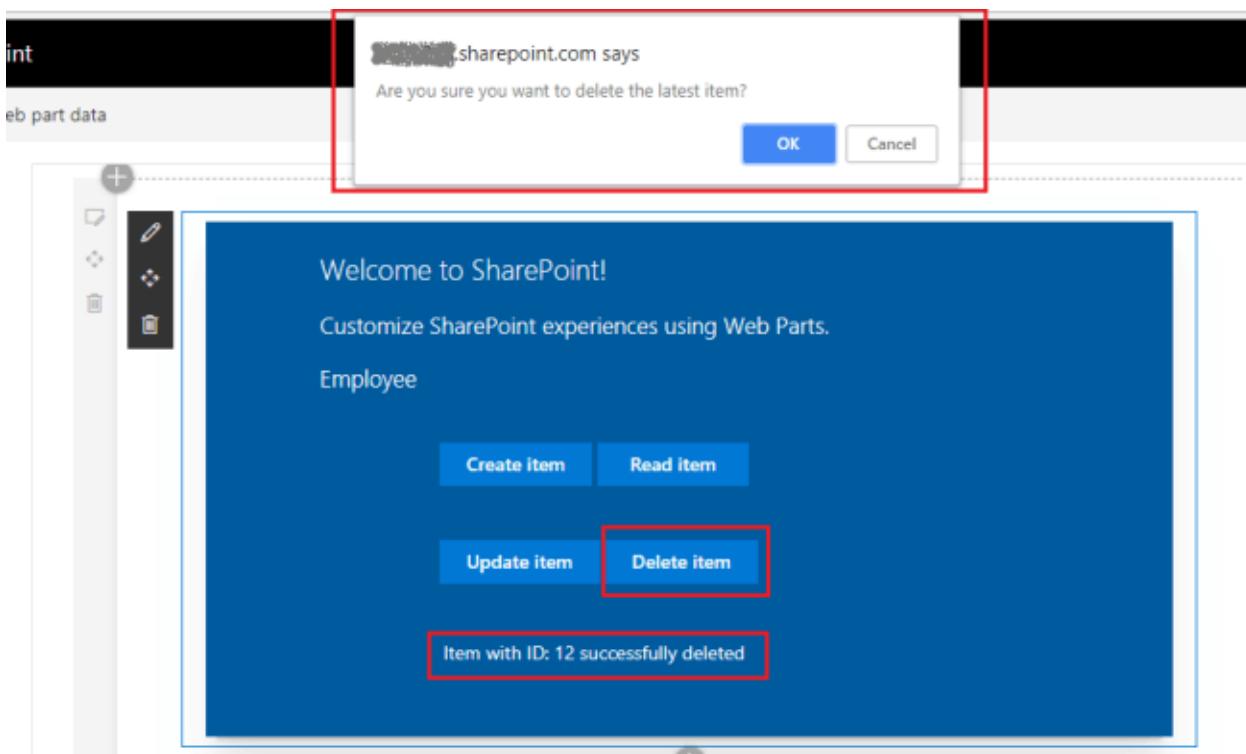
Read Operation



Update Operation



Delete Operation



Summary

React JS is natively supported by SharePoint framework. SPFx generates all needed React components for you to get started with the development. React JS supports the application to be broken down into small React components, which simplifies development and maintenance. React JS focuses more on UI development, in contrast to Angular which is more famous for creating SPA (Single Page Applications).

Chapter 12: CRUD operations using Angular JS

Overview

In the chapters so far, we have explored on developing basic SharePoint client web part which can run independently without any interaction with SharePoint.

In this chapter, we will explore to interact with the SharePoint list for CRUD (Create, Read, Update, and Delete) operations using Angular JS. React JS is not natively supported by SharePoint Framework.

Brief about Angular JS

AngularJS is a JavaScript framework which extends HTML with new attributes. It is popular for creating SPA (Single Page Application). Read more about Angular JS at <https://angularjs.org/>

Create SPFx Solution

1. Open command prompt. Create a directory for SPFx solution

```
md spfx-crud-angularjs
```

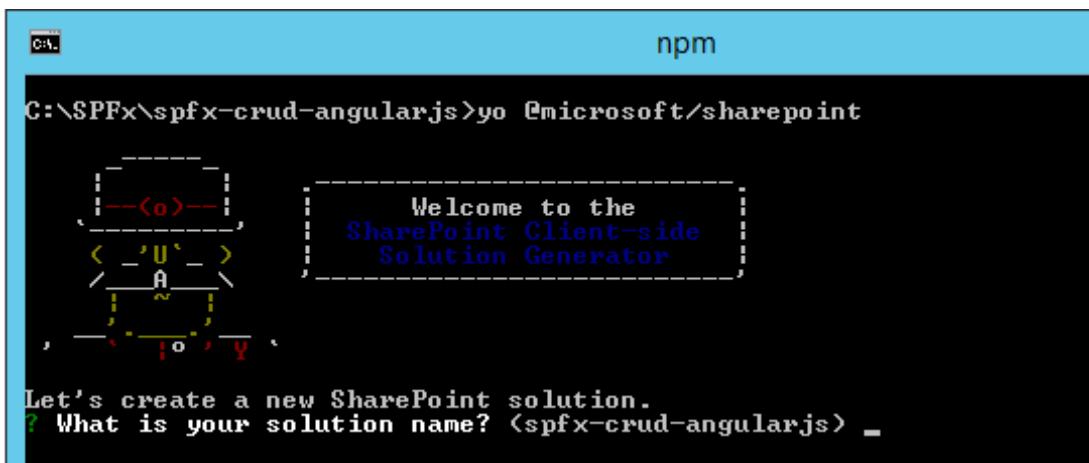
2. Navigate to above created directory

```
cd spfx-crud-angularjs
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



The screenshot shows a terminal window titled 'npm' with the command 'yo @microsoft/sharepoint' entered. The output from the Yeoman generator is displayed, starting with 'Welcome to the SharePoint Client-side Solution Generator'. Below this, it says 'Let's create a new SharePoint solution.' and then asks the user for the solution name, with '(spfx-crud-angularjs)' followed by a blank line for input.

Solution Name: Hit enter to have default name (spfx-crud-angularjs in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension. Choose webpart option.

Selected choice: WebPart

Web part name: Hit enter to select the default name or type in any other name.

Selected choice: AngularCRUD

Web part description: Hit enter to select the default description or type in any other value.

Selected choice: CRUD operations with Angular JS

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)

Selected choice: React

5. Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take significant amount of time.

6. Once the scaffolding process is completed, lock down the version of project dependencies by running below command

```
npm shrinkwrap
```

7. In the command prompt type below command to open the solution in code editor of your choice.

```
code .
```

Configure Angular JS

As Angular JS is not natively supported in SharePoint framework, we have to add a reference of it to our solution using NPM package. Type in the below commands on the command prompt.

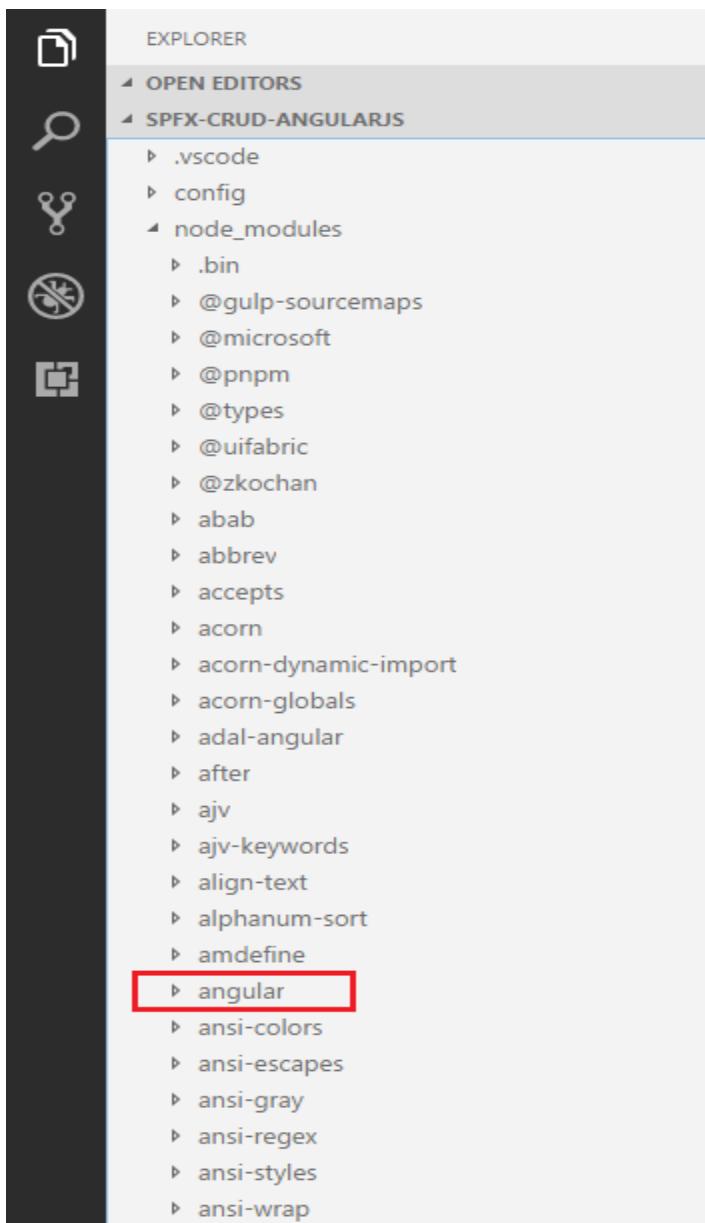
```
npm i angular ng-office-ui-fabric --save
```

The --save option enables NPM to include the packages to dependencies section of the package.json file.

Install Angular typings for TypeScript. Typings will help for auto complete while writing the code in the code editor.

```
tsd install @types/angular --save
```

Expand node_module folder to see the npm packages being added for Angular.



Open **config.json** under config folder. Under externals node, add Angular reference.



```

EXPLORER          config.json ×
OPEN EDITORS      config.json config
SPFX-CRUD-ANGULARJS
  .vscode
    config
      config.json
      copy-assets.json
      deploy-azure-storage.json
      package-solution.json
      serve.json
      tslint.json
      write-manifests.json
  node_modules
  src
    .editorconfig
    .gitignore
    .yo-rc.json
    gulpfile.js
    npm-shrinkwrap.json
    package.json
    README.md
    tsconfig.json

```

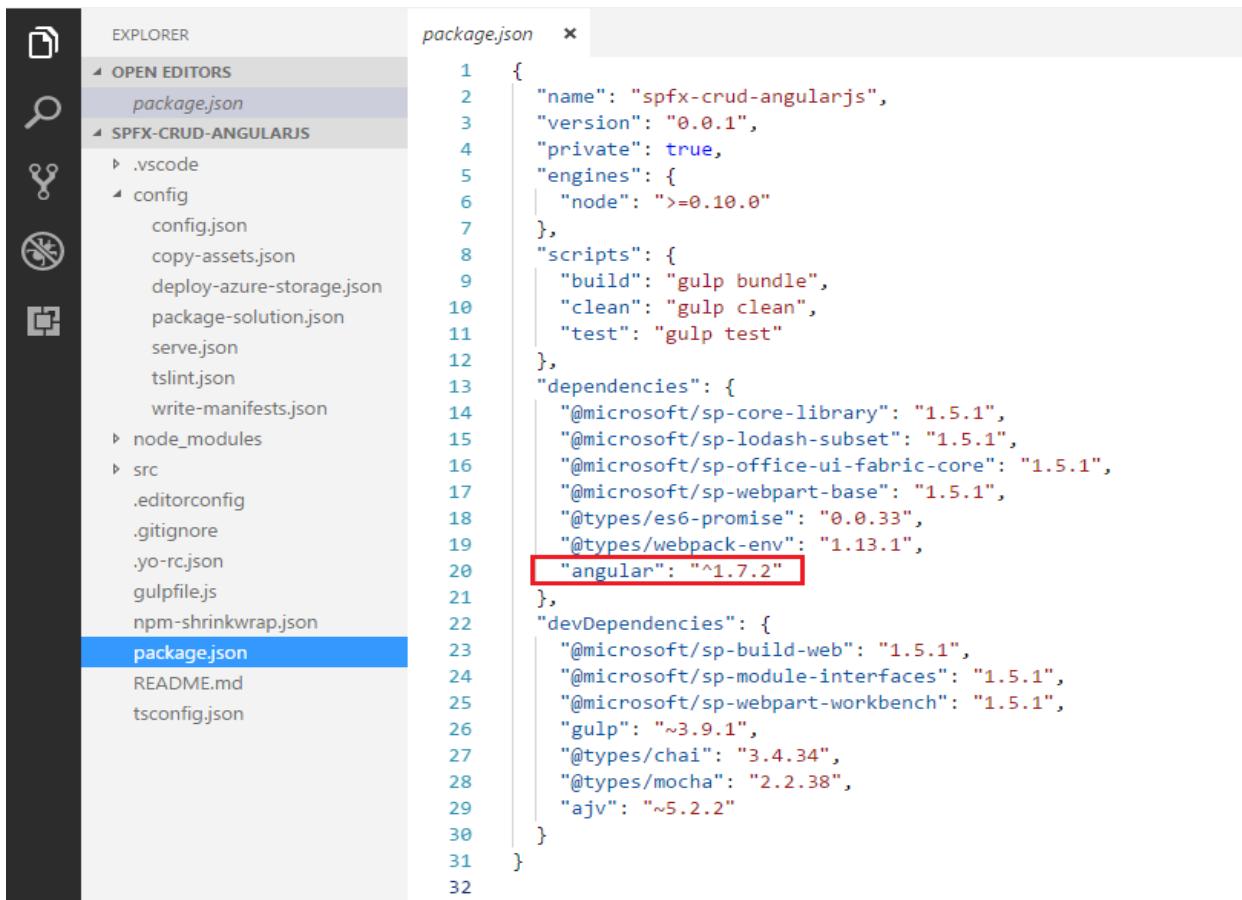
```

1  {
2   "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/config.2.0.schema.json",
3   "version": "2.0",
4   "bundles": [
5     "angular-crud-web-part": {
6       "components": [
7         {
8           "entrypoint": "./lib/webparts/angularCrud/AngularCrudWebPart.js",
9           "manifest": "./src/webparts/angularCrud/AngularCrudWebPart.manifest.json"
10        }
11      ]
12    }
13  },
14  "externals": {
15    "angular": {
16      "path": "node_modules/angular/angular.min.js",
17      "globalName": "angular"
18    }
19  },
20  "localizedResources": {
21    "AngularCrudWebPartStrings": "lib/webparts/angularCrud/loc/{locale}.js"
22  }
23}
24

```

You may also add path from external CDN.

Open **package.json** and verify Angular dependency is listed.



```

EXPLORER          package.json ×
OPEN EDITORS      package.json
SPFX-CRUD-ANGULARJS
  .vscode
    config
      config.json
      copy-assets.json
      deploy-azure-storage.json
      package-solution.json
      serve.json
      tslint.json
      write-manifests.json
  node_modules
  src
    .editorconfig
    .gitignore
    .yo-rc.json
    gulpfile.js
    npm-shrinkwrap.json
    package.json
    README.md
    tsconfig.json

```

```

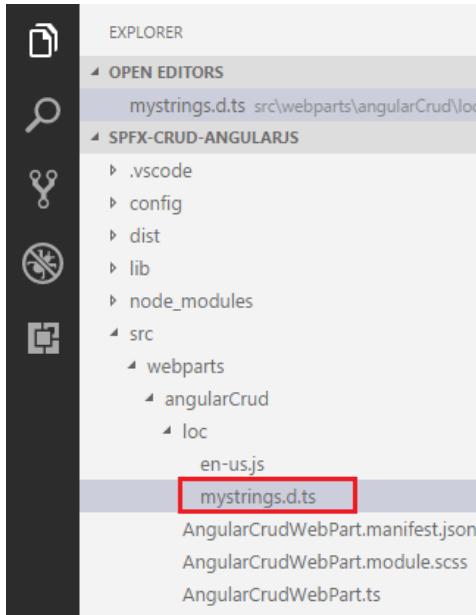
1  {
2   "name": "spfx-crud-angularjs",
3   "version": "0.0.1",
4   "private": true,
5   "engines": {
6     "node": ">=0.10.0"
7   },
8   "scripts": {
9     "build": "gulp bundle",
10    "clean": "gulp clean",
11    "test": "gulp test"
12  },
13  "dependencies": {
14    "@microsoft/sp-core-library": "1.5.1",
15    "@microsoft/sp-lodash-subset": "1.5.1",
16    "@microsoft/sp-office-ui-fabric-core": "1.5.1",
17    "@microsoft/sp-webpart-base": "1.5.1",
18    "@types/es6-promise": "0.0.33",
19    "@types/webpack-env": "1.13.1",
20    "angular": "^1.7.2"
21  },
22  "devDependencies": {
23    "@microsoft/sp-build-web": "1.5.1",
24    "@microsoft/sp-module-interfaces": "1.5.1",
25    "@microsoft/sp-webpart-workbench": "1.5.1",
26    "gulp": "~3.9.1",
27    "@types/chai": "3.4.34",
28    "@types/mocha": "2.2.38",
29    "ajv": "~5.2.2"
30  }
31}
32

```

Configure Property for List Name

SPFx solution by default have description property created. Let us change the property to list name. We will use this property to configure the list name on which the CRUD operation is to perform.

1. Open mystrings.d.ts under \src\webparts\angularCrud\loc\ folder
2. Rename DescriptionFieldLabel to ListNameFieldLabel



EXPLORER

- OPEN EDITORS
- mystrings.d.ts src\webparts\angularCrud\loc
- SPFx-CRUD-ANGULARJS
 - .vscode
 - config
 - dist
 - lib
 - node_modules
 - src
 - webparts
 - angularCrud
 - loc
 - en-us.js
 - mystrings.d.ts**
- AngularCrudWebPart.manifest.json
- AngularCrudWebPart.module.scss
- AngularCrudWebPart.ts

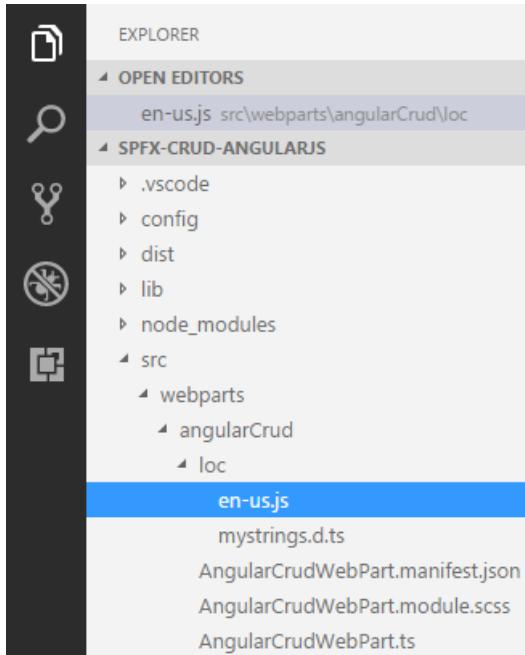
mystrings.d.ts

```

1 declare interface IAngularCrudWebPartStrings {
2   PropertyPaneDescription: string;
3   BasicGroupName: string;
4   ListNameFieldLabel: string;
5 }
6
7 declare module 'AngularCrudWebPartStrings' {
8   const strings: IAngularCrudWebPartStrings;
9   export = strings;
10}
11

```

3. In en-us.js file under \src\webparts\angularCrud\loc\ folder set the display name for listName property



EXPLORER

- OPEN EDITORS
- en-us.js src\webparts\angularCrud\loc
- SPFx-CRUD-ANGULARJS
 - .vscode
 - config
 - dist
 - lib
 - node_modules
 - src
 - webparts
 - angularCrud
 - loc
 - en-us.js**
 - mystrings.d.ts
 - AngularCrudWebPart.manifest.json
 - AngularCrudWebPart.module.scss
 - AngularCrudWebPart.ts

en-us.js

```

1 define([], function() {
2   return {
3     "PropertyPaneDescription": "Description",
4     "BasicGroupName": "Group Name",
5     "ListNameFieldLabel": "List Name"
6   }
7 });

```

4. Open main webpart file (AngularCrudWebPart.ts) under \src\webparts\angularCrud folder.

5. Rename description property pane field to listName

```

export interface IAngularCrudWebPartProps {
    listName: string;
}

export default class AngularCrudWebPart extends BaseClientSideWebPart<IAngularCrudWebPartProps> {

    public render(): void {
        this.domElement.innerHTML = `
            <div class="${ styles.angularCrud }">
                <div class="${ styles.container }">
                    <div class="${ styles.row }">
                        <div class="${ styles.column }">
                            <span class="${ styles.title }">Welcome to SharePoint!</span>
                            <p class="${ styles.subTitle }">Customize SharePoint experiences using Web Parts.</p>
                            <p class="${ styles.description }">${escape(this.properties.listName)}</p>
                            <a href="https://aka.ms/spfx" class="${ styles.button }">
                                <span class="${ styles.label }">Learn more</span>
                            </a>
                        </div>
                    </div>
                </div>
            </div>`;
    }

    protected get dataVersion(): Version {
        return Version.parse('1.0');
    }

    protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
        return {
            pages: [
                {
                    header: {
                        description: strings.PropertyPaneDescription
                    },
                    groups: [
                        {
                            groupName: strings.BasicGroupName,
                            groupFields: [
                                PropertyPaneTextField('listName', {
                                    label: strings.ListNameFieldLabel
                                })
                            ]
                        }
                    ]
                }
            ];
        };
    }
}

```

```

export interface IAngularCrudWebPartProps {
    listName: string;
}

export default class AngularCrudWebPart extends
BaseClientSideWebPart<IAngularCrudWebPartProps> {

    public render(): void {
        this.domElement.innerHTML = `
            <div class="${ styles.angularCrud }">
                <div class="${ styles.container }">
                    <div class="${ styles.row }">
                        <div class="${ styles.column }">

```

```

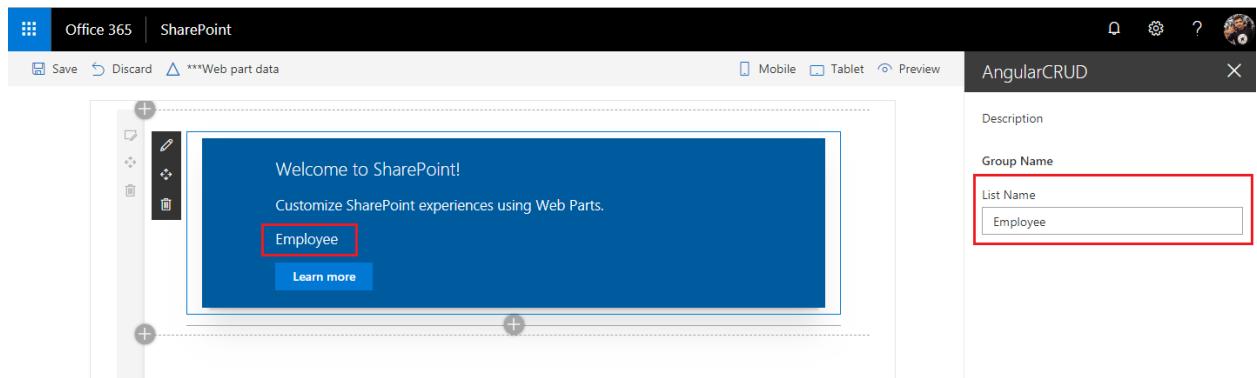
        <span class="${ styles.title }">Welcome to
SharePoint!</span>
            <p class="${ styles.subTitle }">Customize SharePoint
experiences using Web Parts.</p>
            <p class="${ styles.description
}">${escape(this.properties.listName)}</p>
            <a href="https://aka.ms/spfx" class="${ styles.button }">
                <span class="${ styles.label }">Learn more</span>
            </a>
        </div>
    </div>
</div>`;
}

protected get dataVersion(): Version {
    return Version.parse('1.0');
}

protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
    return {
        pages: [
            {
                header: {
                    description: strings.PropertyPaneDescription
                },
                groups: [
                    {
                        groupName: strings.BasicGroupName,
                        groupFields: [
                            PropertyPaneTextField('listName', {
                                label: strings.ListNameFieldLabel
                            })
                        ]
                    }
                ]
            }
        ];
    };
}
}

```

6. In the command prompt, type “gulp serve”
7. In the SharePoint local workbench page, add the web part.
8. Edit the web part to ensure the listName property pane field is getting reflected.



Build an Angular application

Add folder named “app” as starting point for creating Angular app.

Configure Data Service

Add a file DataService.ts under app folder and define the data services we are going to perform.

```
export interface IListItem {
  Id: number;
  Title?: string;
}

export interface IDataService {
  createItem(title: string, webUrl: string, listName: string):
  angular.IPromise<IListItem>;
  readItem(itemId: number, webUrl: string, listName: string):
  angular.IPromise<IListItem>;
  getLatestItemId(webUrl: string, listName: string):
  angular.IPromise<number>;
  updateItem(item: IListItem, webUrl: string, listName: string):
  angular.IPromise<{}>;
  deleteItem(item: IListItem, webUrl: string, listName: string):
  angular.IPromise<{}>;
}

export default class DataService implements IDataService {
  public static $inject: string[] = ['$q', '$http'];

  constructor(private $q: angular.IQService, private $http:
  angular.IHttpService) {
  }

  public createItem(title: string, webUrl: string, listName: string):
  angular.IPromise<IListItem> {
```

```

    const deferred: angular.IDeferred<IListItem> = this.$q.defer();
    return deferred.promise;
}

public readItem(itemId: number, webUrl: string, listName: string):
angular.IPromise<IListItem> {
    const deferred: angular.IDeferred<IListItem> = this.$q.defer();
    return deferred.promise;
}

public getLatestItemId(webUrl: string, listName: string):
angular.IPromise<number> {
    const deferred: angular.IDeferred<number> = this.$q.defer();
    return deferred.promise;
}

public updateItem(item: IListItem, webUrl: string, listName: string):
angular.IPromise<{}> {
    const deferred: angular.IDeferred<IListItem> = this.$q.defer();
    return deferred.promise;
}

public deleteItem(item: IListItem, webUrl: string, listName: string):
angular.IPromise<{}> {
    const deferred: angular.IDeferred<IListItem> = this.$q.defer();
    return deferred.promise;
}
}

```

Implement Create Operation

We will use the REST API to add the item to list.

```

public createItem(title: string, webUrl: string, listName: string):
angular.IPromise<IListItem> {
    const deferred: angular.IDeferred<IListItem> = this.$q.defer();
    const body: string = JSON.stringify({
        'Title': `Item ${new Date()}`
    });

    this.getRequestDigest(webUrl)
        .then((requestDigest: string):
angular.IPromise<angular.IHttpPromiseCallbackArg<IListItem>> => {
        return this.$http({

```

```

        url: `${webUrl}/_api/web/lists/getbytitle('${listName}')/items`,
        method: 'POST',
        headers: {
            "Accept": "application/json;odata=nometadata",
            "Content-type": "application/json;odata=verbose",
            "X-RequestDigest": requestDigest
        }
    });
})
.then((response: angular.IHttpPromiseCallbackArg<IListItem>): void => {
    deferred.resolve(response.data);
}, (error: any): void => {
    deferred.reject(error);
});

return deferred.promise;
}

```

Implement Read Operation

We will use REST API to read the latest item.

```

public readItem(itemId: number, webUrl: string, listName: string):
angular.IPromise<IListItem> {
    const deferred: angular.IDeferred<IListItem> = this.$q.defer();

    this.$http({
        url:
`${webUrl}/_api/web/lists/getbytitle('${listName}')/items(${itemId})`,
        method: 'GET',
        headers: {
            'Accept': 'application/json;odata=nometadata'
        }
    })
    .then((response: angular.IHttpPromiseCallbackArg<IListItem>): void => {
        const item: IListItem = response.data;
        item.ETag = response.headers('ETag');
        deferred.resolve(item);
    }, (error: any): void => {
        deferred.reject(error);
    });

    return deferred.promise;
}

```

Implement Update Operation

We will use REST API to update the latest item.

```
public updateItem(item: IListItem, webUrl: string, listName: string):
angular.IPromise<{}> {
  const deferred: angular.IDeferred<IListItem> = this.$q.defer();

  const body: string = JSON.stringify({
    'Title': `Item ${new Date()}`)
  });

  this.$http({
    url:
`${webUrl}/_api/web/lists/getbytitle('${listName}')/items(${item.Id})`,
    method: 'POST',
    headers: {
      'Accept': 'application/json;odata=nometadata',
      'Content-type': 'application/json;odata=nometadata',
      'odata-version': '',
      'IF-MATCH': '*',
      'X-HTTP-Method': 'MERGE'
    },
    data: body
  })
  .then((result: {}): void => {
    deferred.resolve();
  }, (error: any): void => {
    deferred.reject(error);
  });
}

return deferred.promise;
}
```

Implement Delete Operation

We will use REST API to delete the latest item.

```
public deleteItem(item: IListItem, webUrl: string, listName: string):
angular.IPromise<{}> {
  const deferred: angular.IDeferred<IListItem> = this.$q.defer();

  this.$http({
    url:
`${webUrl}/_api/web/lists/getbytitle('${listName}')/items(${item.Id})`,
    method: 'POST',
```

```

        headers: {
            'Accept': 'application/json;odata=nometadata',
            'Content-type': 'application/json;odata=verbose',
            'odata-version': '',
            'IF-MATCH': item.ETag,
            'X-HTTP-Method': 'DELETE'
        }
    })
    .then((result: {}): void => {
        deferred.resolve();
    }, (error: any): void => {
        deferred.reject(error);
    });
}

return deferred.promise;
}

```

Configure Controller

Add a file HomeController.ts under app folder

```

import { IDataService, IListItem } from './DataService';

interface IConfigurationChangedEventArgs {
    webUrl: string;
    listName: string;
}

export default class HomeController {
    public static $inject: string[] = ['DataService', '$window', '$rootScope',
    '$scope'];

    public status: string = undefined;
    public items: IListItem[] = [];

    private webUrl: string = undefined;
    private listName: string = undefined;

    constructor(private DataService: IDataService, private $window:
angular.IWindowService, private $rootScope: angular.IRootScopeService, private
$scope: angular.IScope) {
        const vm: HomeController = this;
        this.init(undefined, undefined, undefined);

        $rootScope.$on('configurationChanged',

```

```

        (event: angular.IAngularEvent, args: IConfigurationChangedEventArgs): void =>
{
    vm.init(args.webUrl, args.listName, vm.$scope);
});

}

private init(webUrl: string, listName: string, $scope: angular.IScope): void
{
    if (webUrl !== undefined && webUrl.length > 0 &&
        listName !== undefined && listName.length > 0) {
        this.webUrl = webUrl;
        this.listName = listName;
    }

    if ($scope) {
        $scope.$digest();
    }
}

public createItem(): void {
    const itemTitle: string = `Item ${new Date()}`;
    this.status = 'Creating item...';
    this.items.length = 0;
    this.dataService.createItem(itemTitle, this.webUrl, this.listName)
        .then((item: IListItem): void => {
            this.status = `Item '${item.Title}' (ID: ${item.Id}) successfully
created`;
        }, (error: any): void => {
            this.status = 'Error while creating the item: ' + error;
        });
}

public readItem(): void {
    this.status = 'Loading latest items...';
    this.items.length = 0;
    this.dataService.getLatestItemId(this.webUrl, this.listName)
        .then((itemId: number): angular.IPromise<IListItem> => {
            if (itemId === -1) {
                throw new Error('No items found in the list');
            }

            this.status = `Loading information about item ID: ${itemId}...`;
            return this.dataService.readItem(itemId, this.webUrl, this.listName);
        })
        .then((item: IListItem): void => {
}

```

```
        this.status = `Item ID: ${item.Id}, Title: ${item.Title}`;
    }, (error: any): void => {
    this.status = 'Loading latest item failed with error: ' + error;
});
}

public updateItem(): void {
    this.status = 'Loading latest items...';
    this.items.length = 0;
    let latestItemId: number = undefined;
    this.dataService.getLatestItemId(this.webUrl, this.listName)
        .then((itemId: number): angular.IPromise<IL listItem> => {
            if (itemId === -1) {
                throw new Error('No items found in the list');
            }

            latestItemId = itemId;
            this.status = `Loading information about item ID: ${latestItemId}...`;

            return this.dataService.readItem(latestItemId, this.webUrl,
this.listName);
        })
        .then((latestItem: IL listItem): angular.IPromise<{}> => {
            this.status = `Updating item with ID: ${latestItemId}...`;
            latestItem.Title = `Item ${new Date()}`;
            return this.dataService.updateItem(latestItem, this.webUrl,
this.listName);
        })
        .then((result: {}): void => {
            this.status = `Item with ID: ${latestItemId} successfully updated`;
        }, (error: any): void => {
            this.status = `Error updating item: ${error}`;
        });
}
}

public deleteItem(): void {
    if (!this.$window.confirm('Are you sure you want to delete this todo
item?')) {
        return;
    }

    this.status = 'Loading latest items...';
    this.items.length = 0;
    let latestItemId: number = undefined;
    this.dataService.getLatestItemId(this.webUrl, this.listName)
```

```

    .then((itemId: number): angular.IPromise<IListItem> => {
      if (itemId === -1) {
        throw new Error('No items found in the list');
      }

      latestItemId = itemId;
      this.status = `Loading information about item ID: ${latestItemId}...`;

      return this.dataService.readItem(latestItemId, this.webUrl,
this.listName);
    })
    .then((latestItem: IListItem): angular.IPromise<{}> => {
      this.status = `Deleting item with ID: ${latestItemId}...`;
      return this.dataService.deleteItem(latestItem, this.webUrl,
this.listName);
    })
    .then((result: {}): void => {
      this.status = `Item with ID: ${latestItemId} successfully deleted`;
    }, (error: any): void => {
      this.status = `Error deleting item: ${error}`;
    });
  }
}

```

Configure Module

Add a file app-module.ts under app folder

```

import * as angular from 'angular';
import HomeController from './HomeController';
import DataService from './DataService';

import 'ng-office-ui-fabric';

const crudapp: ng.IModule = angular.module('crudapp', [
  'officeuifabric.core',
  'officeuifabric.components'
]);

crudapp
  .controller('HomeController', HomeController)
  .service('DataService', DataService);

```

Add Controls to WebPart

1. Open AngularCrudWebPart.ts under “\src\webparts\angularCrud\” folder.
2. Import Angular module

```
import * as angular from 'angular';
import './app/app-module';
```

3. Modify Render method to include buttons for CRUD operations and add event handlers to each of the button

```
export default class AngularCrudWebPart extends
BaseClientSideWebPart<IAngularCrudWebPartProps> {
    private $injector: angular.auto.IInjectorService;

    public render(): void {
        if (this.renderedOnce === false) {
            this.domElement.innerHTML =
                <div class="${styles.angularCrud}" data-ng-
controller="HomeController as vm">
                    <div class="${styles.container}">
                        <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-
white ${styles.row}">
                            <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2
ms-u-lgPush1">
                                <span class="ms-font-xl ms-fontColor-white">
                                    Sample SharePoint CRUD operations in Angular
                                </span>
                            </div>
                        </div>
                        <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-
white ${styles.row}">
                            <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2
ms-u-lgPush1">
                                <uif-button ng-click="vm.createItem()" ng-
disabled="vm.listNotConfigured">Create item</uif-button>
                                <uif-button ng-click="vm.readItem()" ng-
disabled="vm.listNotConfigured">Read item</uif-button>
                            </div>
                        </div>
                        <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-
white ${styles.row}">
                            <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2
ms-u-lgPush1">
                                <uif-button ng-click="vm.readItems()" ng-
disabled="vm.listNotConfigured">Read all items</uif-button>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        }
    }
}
```

```

        </div>
        <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
            <div class="ms-Grid-col ms-u-lg10 ms-u-xl18 ms-u-xlPush2 ms-u-lgPush1">
                <uif-button ng-click="vm.updateItem()" ng-disabled="vm.listNotConfigured">Update item</uif-button>
                <uif-button ng-click="vm.deleteItem()" ng-disabled="vm.listNotConfigured">Delete item</uif-button>
            </div>
        </div>
        <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
            <div class="ms-Grid-col ms-u-lg10 ms-u-xl18 ms-u-xlPush2 ms-u-lgPush1">
                <div>{{vm.status}}</div>
                <ul>
                    <li ng-repeat="item in vm.items">{{item.Title}}<br/>{{item.Id}}</li>
                    <ul>
                </div>
            </div>
        </div>
    </div>`;

    this.$injector = angular.bootstrap(this.documentElement, ['crudapp']);
}

this.$injector.get('$rootScope').$broadcast('configurationChanged',
{
    webUrl: this.context.pageContext.web.absoluteUrl,
    listName: this.properties.listName
});
}
}

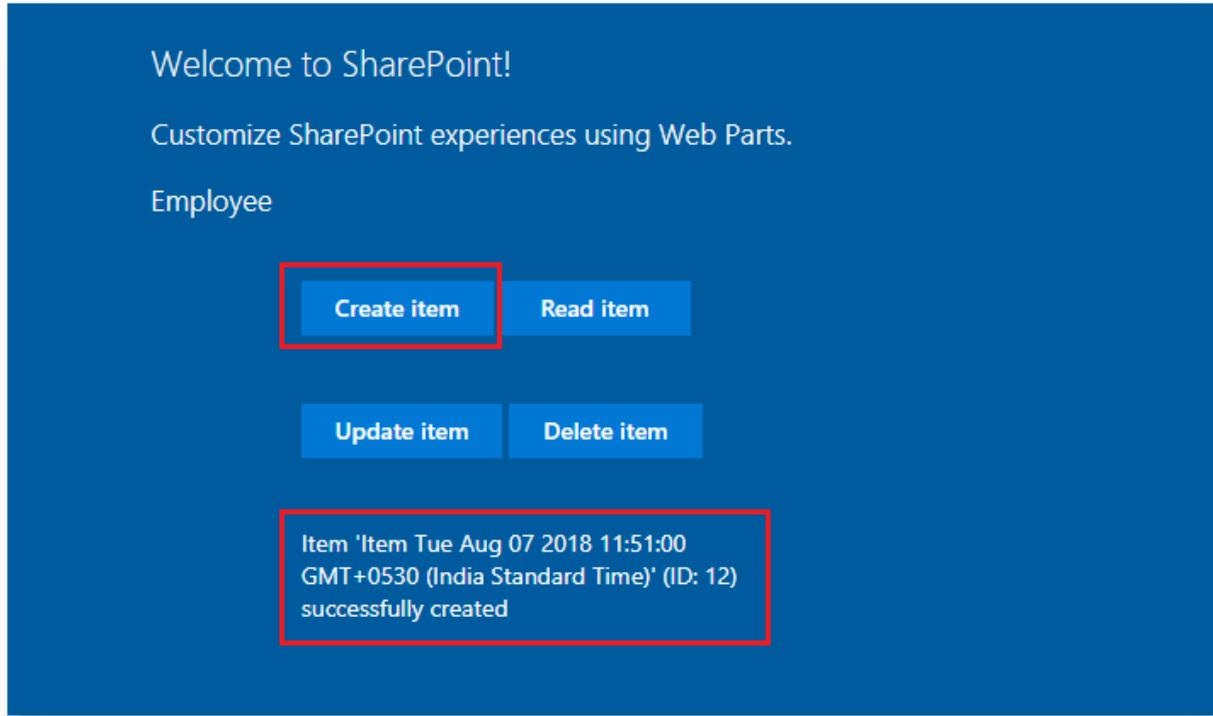
```

Test the WebPart

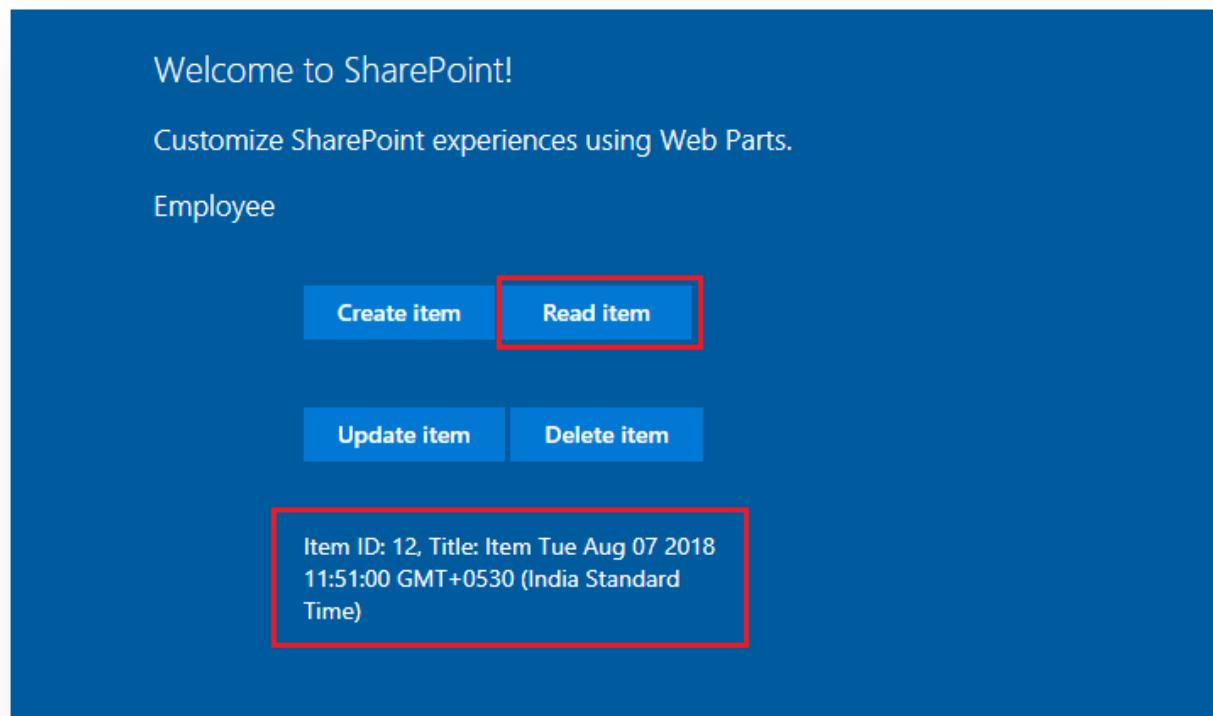
1. On the command prompt, type “gulp serve”
 2. Open SharePoint site
 3. Navigate to /_layouts/15/workbench.aspx
 4. Add the webpart to page.
 5. Edit webpart, in the properties pane type the list name
 6. Click the buttons (Create Item, Read Item, Update Item, and Delete Item) one by one to test the webpart

7. Verify the operations are taking place in SharePoint list.

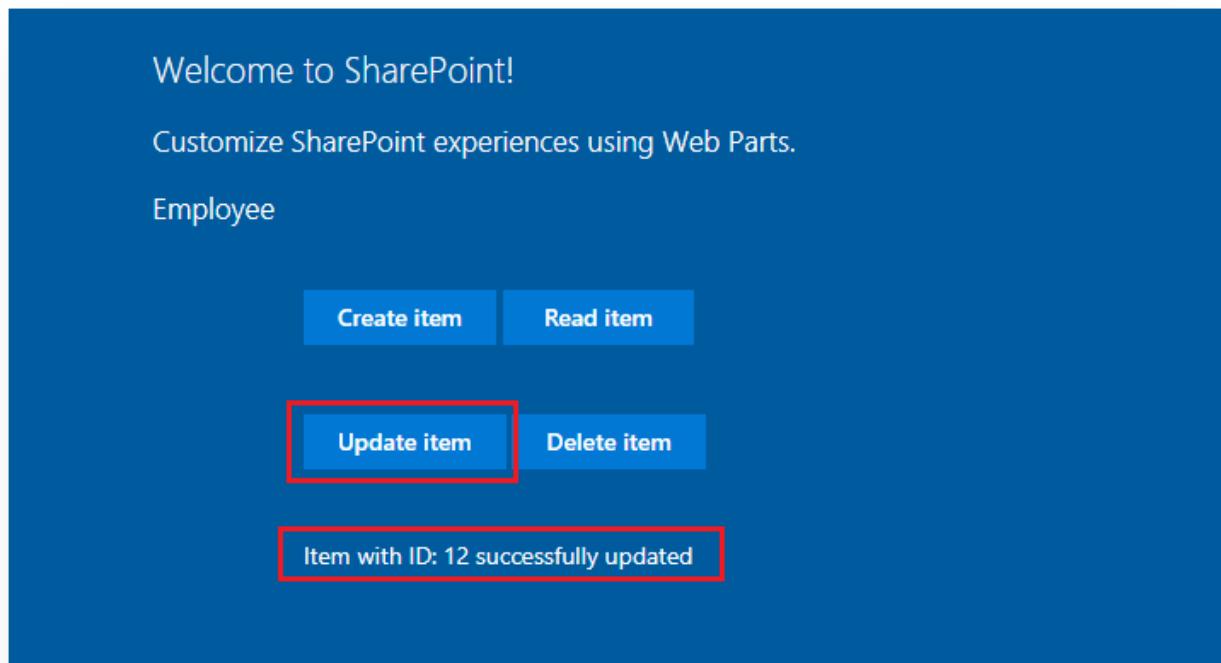
Create Operation



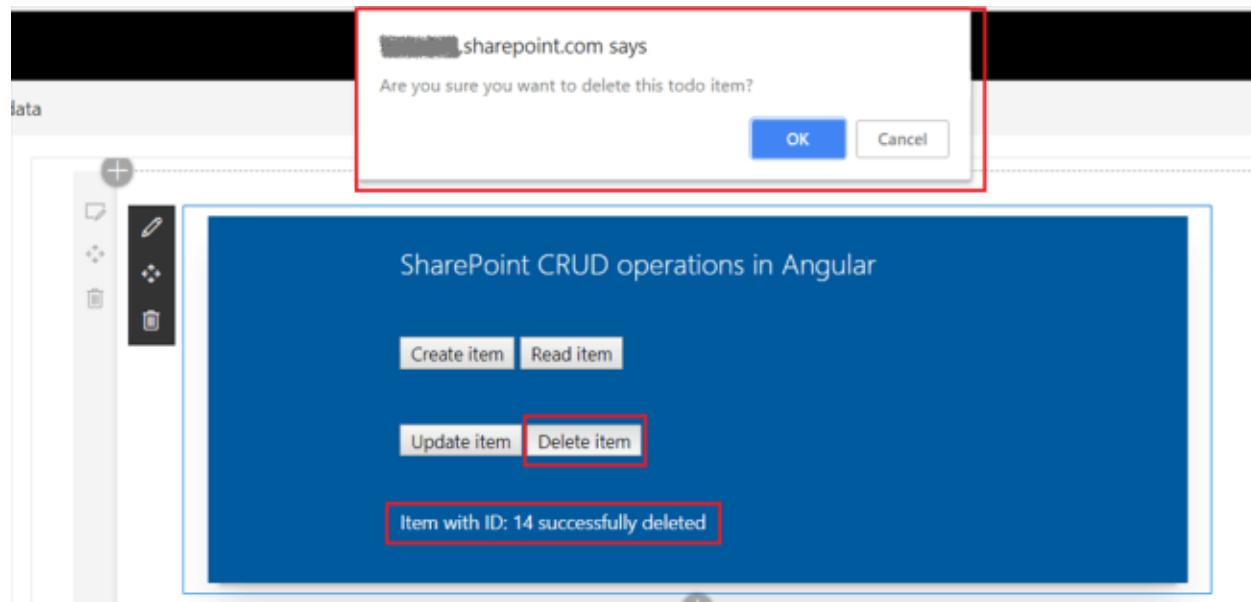
Read Operation



Update Operation

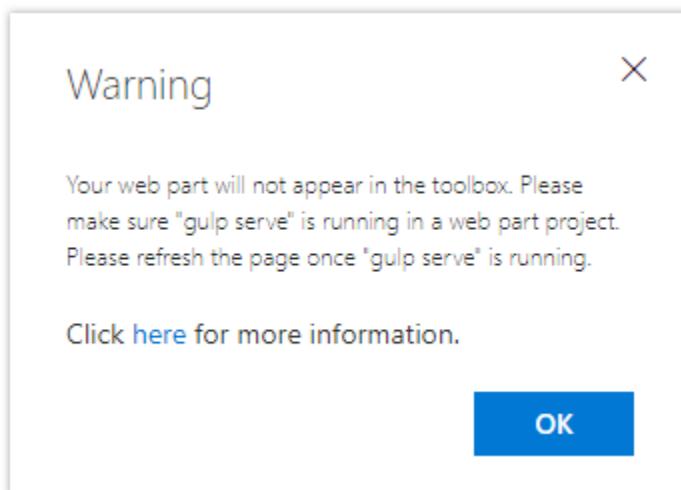


Delete Operation



Troubleshooting

In some cases SharePoint workbench ([https://\[tenant\].sharepoint.com/_layouts/15/workbench.aspx](https://[tenant].sharepoint.com/_layouts/15/workbench.aspx)) shows below error although “gulp serve” is running.



Open below url in the next tab of browser. Accept the warning message.

<https://localhost:4321/temp/manifests.js>

Summary

React JS is natively supported by SharePoint framework. SPFx generates all needed React components for you to get started with the development. React JS supports the application to be broken down into small React components, which simplifies development and maintenance. React JS focuses more on UI development, in contrast to Angular which is more famous for creating SPA (Single Page Applications).

Chapter 13: CRUD operations using Knockout JS

Overview

In the chapters so far, we have explored on developing basic SharePoint client web part which can run independently without any interaction with SharePoint.

In this chapter, we will explore to interact with the SharePoint list for CRUD (Create, Read, Update, and Delete) operations using Knockout JS. Knockout JS is not natively supported by SharePoint Framework.

Brief about Knockout JS

Knockout JS was developed and maintained as an open source project by Steve Sanderson, a Microsoft employee. Knockout JS follows JavaScript implementation of the Model-View-ViewModel pattern with templates. Read more about Knockout JS at <http://knockoutjs.com/>

Create SPFx Solution

1. Open command prompt. Create a directory for SPFx solution

```
md spfx-crud-knockoutjs
```

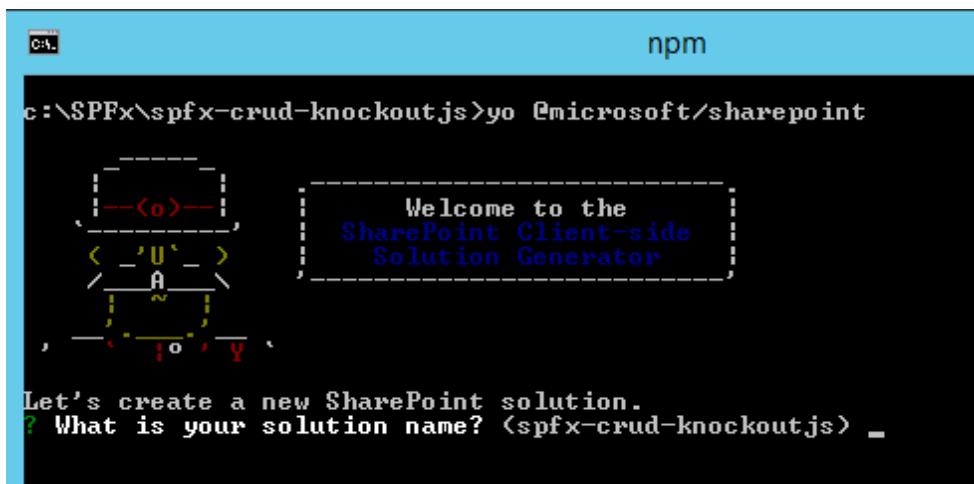
2. Navigate to above created directory

```
cd spfx-crud-knockoutjs
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



The screenshot shows a terminal window titled 'npm' with the command 'yo @microsoft/sharepoint' entered. A wizard interface is displayed, asking 'What is your solution name?'. The placeholder '(spfx-crud-knockoutjs)' is shown in the input field.

Solution Name: Hit enter to have default name (spfx-crud-knockoutjs in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension. Choose webpart option.

Selected choice: WebPart

Web part name: Hit enter to select the default name or type in any other name.

Selected choice: KnockoutCRUD

Web part description: Hit enter to select the default description or type in any other value.

Selected choice: CRUD operations with Knockout JS

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)

Selected choice: Knockout

5. Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take significant amount of time.

6. Once the scaffolding process is completed, lock down the version of project dependencies by running below command

```
npm shrinkwrap
```

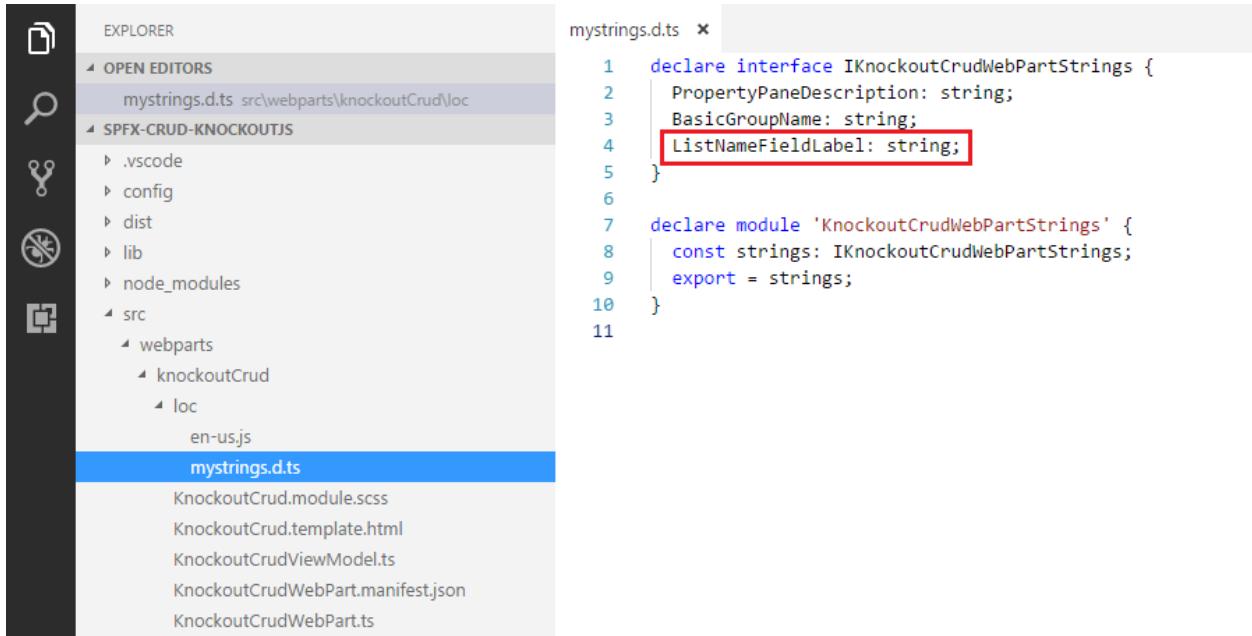
7. In the command prompt type below command to open the solution in code editor of your choice.

```
code .
```

Configure Property for List Name

SPFx solution by default have description property created. Let us change the property to list name. We will use this property to configure the list name on which the CRUD operation is to perform.

1. Open mystrings.d.ts under \src\webparts\knockoutCrud\loc\ folder
2. Rename DescriptionFieldLabel to ListNameFieldLabel



```

EXPLORER
OPEN EDITORS
mystrings.d.ts src\webparts\knockoutCrud\loc
SPFX-CRUD-KNOCKOUTJS
  .vscode
  config
  dist
  lib
  node_modules
  src
    webparts
      knockoutCrud
        loc
        en-us.js
        mystrings.d.ts
      KnockoutCrud.module.scss
      KnockoutCrud.template.html
      KnockoutCrudViewModel.ts
      KnockoutCrudWebPart.manifest.json
      KnockoutCrudWebPart.ts

```

```

mystrings.d.ts x
1 declare interface IKnockoutCrudWebPartStrings {
2   PropertyPaneDescription: string;
3   BasicGroupName: string;
4   ListNameFieldLabel: string;
5 }
6
7 declare module 'KnockoutCrudWebPartStrings' {
8   const strings: IKnockoutCrudWebPartStrings;
9   export = strings;
10 }
11

```

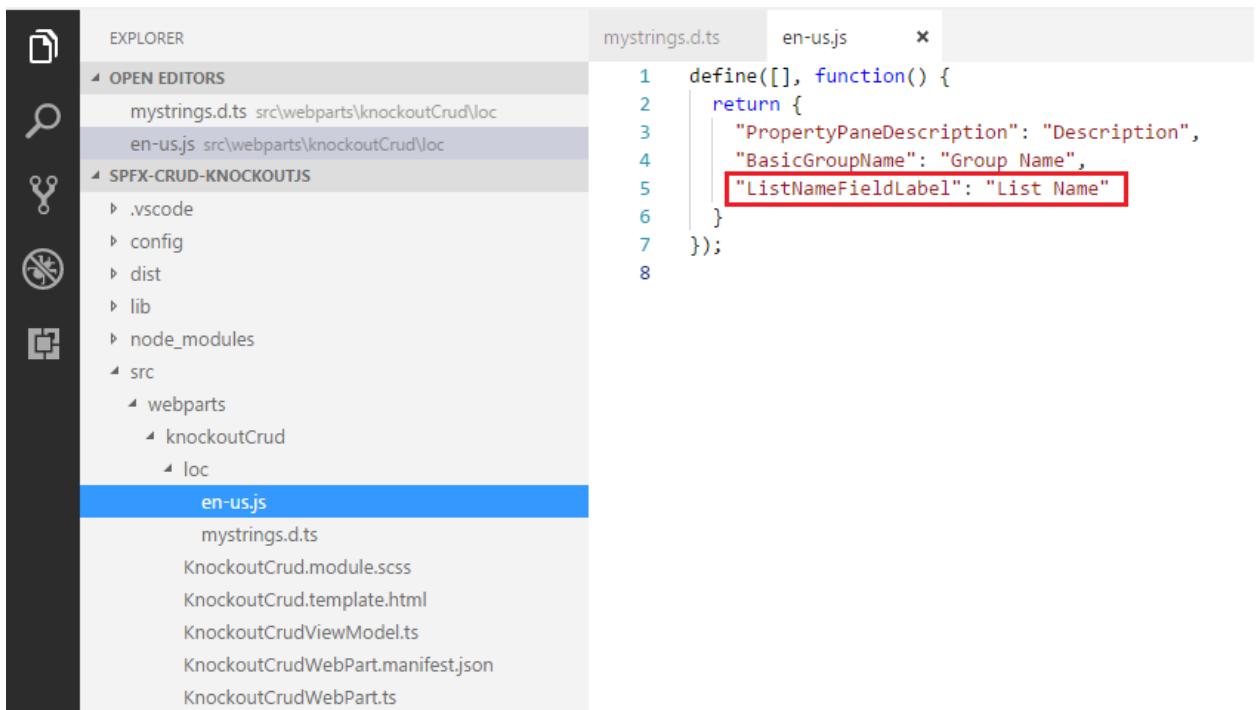
```

declare interface IKnockoutCrudWebPartStrings {
  PropertyPaneDescription: string;
  BasicGroupName: string;
  ListNameFieldLabel: string;
}

declare module 'KnockoutCrudWebPartStrings' {
  const strings: IKnockoutCrudWebPartStrings;
  export = strings;
}

```

3. In en-us.js file under \src\webparts\knockoutCrud\loc\ folder set the display name for listName property



The screenshot shows the SharePoint Framework Explorer in VS Code. The left sidebar lists project files under 'SPFX-CRUD-KNOCKOUTJS'. The 'en-us.js' file is selected in the Explorer, and its content is displayed in the main editor area. The code defines a string object with properties: 'PropertyPaneDescription', 'BasicGroupName', and 'ListNameFieldLabel'. The 'ListNameFieldLabel' property is highlighted with a red box.

```
1 define([], function() {
2   return {
3     "PropertyPaneDescription": "Description",
4     "BasicGroupName": "Group Name",
5     "ListNameFieldLabel": "List Name"
6   }
7 });
8
```

```
define([], function() {
  return {
    "PropertyPaneDescription": "Description",
    "BasicGroupName": "Group Name",
    "ListNameFieldLabel": "List Name"
  }
});
```

4. Open main webpart file (KnockoutCrudWebPart.ts) under \src\webparts\knockoutCrud folder.
5. Rename description property pane field to listName

KnockoutCrudWebPart.ts - spfx-crud-knockoutjs - Visual Studio Code [Administrator]

```

File Edit Selection View Go Debug Tasks Help
OPEN EDITORS KnockoutCrudWebPart.ts src\webparts\knockoutCrud
VS CODE .vscode
SPFX-CRUD-KNOCKOUTJS config
dist lib node_modules
src
  webparts
    knockoutCrud
      loc
        en-us.js
        mystrings.scss
      KnockoutCrud.module.scss
      KnockoutCrud.template.html
      KnockoutCrudViewModel.ts
      KnockoutCrudWebPart.manifest.json
      KnockoutCrudWebPart.ts
    temp
    .editorconfig
    .gitignore
    .yo-rc.json
    gulpfile.js
    npm-shrinkwrap.json
    package.json
    README.md
    tsconfig.json

KnockoutCrudWebPart.ts

15
14 export interface IKnockoutCrudWebPartProps {
15   |   listName: string;
16 }
17
18 export default class KnockoutCrudWebPart extends BaseClientSideWebPart<IKnockoutCrudWebPartProps> {
19   private _id: number;
20   private _componentElement: HTMLElement;
21   private _koDescription: KnockoutObservable<string> = ko.observable('');
22
23 /**
24  * Shouter is used to communicate between web part and view model.
25  */
26 private _shouter: KnockoutSubscribable<{}> = new ko.subscribable();
27
28 /**
29  * Initialize the web part.
30  */
31 protected OnInit(): Promise<void> {
32   this._id = _instance++;
33
34   const tagName: string = `ComponentElement-${this._id}`;
35   this._componentElement = this._createComponentElement(tagName);
36   this._registerComponent(tagName);
37
38   // When web part description is changed, notify view model to update.
39   this._koDescription.subscribe((newValue: string) => {
40     this._shouter.notifySubscribers(newValue, 'description');
41   });
42
43   const bindings: IKnockoutCrudBindingContext = {
44     |   listName: this.properties.listName,
45     |   shouter: this._shouter
46   };
47
48   ko.applyBindings(bindings, this._componentElement);
49
50   return super.OnInit();
51 }
52
53 public render(): void {
54   if (!this.renderedOnce) {
55     this.domElement.appendChild(this._componentElement);
56   }
57
58   this._koDescription(this.properties.listName);
59 }
60
61 private _createComponentElement(tagName: string): HTMLElement {
62   const componentElement: HTMLElement = document.createElement('div');
63   componentElement.setAttribute('data-bind', `component: { name: "${tagName}" }`);
64   return componentElement;
65 }
66
67 private _registerComponent(tagName: string): void {
68   ko.components.register(
69     tagName,
70     {
71       viewModel: KnockoutCrudViewModel,
72       template: require('./KnockoutCrud.template.html'),
73       synchronous: false
74     }
75   );
76 }
77
78 protected get dataVersion(): Version {
79   return Version.parse('1.0');
80 }
81
82 protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
83   return {
84     pages: [
85       {
86         header: {
87           description: strings.PropertyPaneDescription
88         },
89         groups: [
90           {
91             groupName: strings.BasicGroupName,
92             groupFields: [
93               PropertyPaneTextField('listName', {
94                 |   label: strings.ListNameFieldLabel
95               })
96             ]
97           }
98         ]
99       }
100     }
101   }
102 }
103 }

```

```
import * as ko from 'knockout';
import { Version } from '@microsoft/sp-core-library';
import {
  BaseClientSideWebPart,
  IPropertyPaneConfiguration,
  PropertyPaneTextField
} from '@microsoft/sp-webpart-base';

import * as strings from 'KnockoutCrudWebPartStrings';
import KnockoutCrudViewModel, { IKnockoutCrudBindingContext } from
'./KnockoutCrudViewModel';

let _instance: number = 0;

export interface IKnockoutCrudWebPartProps {
  listName: string;
}

export default class KnockoutCrudWebPart extends
BaseClientSideWebPart<IKnockoutCrudWebPartProps> {
  private _id: number;
  private _componentElement: HTMLElement;
  private _koDescription: KnockoutObservable<string> =
ko.observable('');

  /**
   * Shouter is used to communicate between web part and view model.
   */
  private _shouter: KnockoutSubscribable<{}> = new ko.subscribable();

  /**
   * Initialize the web part.
   */
  protected onInit(): Promise<void> {
    this._id = _instance++;

    const tagName: string = `ComponentElement-${this._id}`;
    this._componentElement = this._createComponentElement(tagName);
    this._registerComponent(tagName);

    // When web part description is changed, notify view model to
    // update.
    this._koDescription.subscribe((newValue: string) => {
      this._shouter.notifySubscribers(newValue, 'description');
    });
  }
}
```

```
const bindings: IKnockoutCrudBindingContext = {
  listName: this.properties.listName,
  shouter: this._shouter
};

ko.applyBindings(bindings, this._componentElement);

return super.onInit();
}

public render(): void {
  if (!this.renderedOnce) {
    this.domElement.appendChild(this._componentElement);
  }

  this._koDescription(this.properties.listName);
}

private _createComponentElement(tagName: string): HTMLElement {
  const componentElement: HTMLElement = document.createElement('div');
  componentElement.setAttribute('data-bind', `component: { name:
"${tagName}", params: $data }`);
  return componentElement;
}

private _registerComponent(tagName: string): void {
  ko.components.register(
    tagName,
    {
      viewModel: KnockoutCrudViewModel,
      template: require('./KnockoutCrud.template.html'),
      synchronous: false
    }
  );
}

protected get dataVersion(): Version {
  return Version.parse('1.0');
}

protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
  return {
    pages: [
      {
        title: 'General',
        description: 'Configure general settings for the component'
      }
    ]
  };
}
```

```

        header: {
            description: strings.PropertyPaneDescription
        },
        groups: [
            {
                groupName: strings.BasicGroupName,
                groupFields: [
                    PropertyPaneTextField('listName', {
                        label: strings.ListNameFieldLabel
                    })
                ]
            }
        ]
    };
}
}
}

```

6. Update the ViewModel inside KnockoutCrudViewModel.ts to reflect listName property

```

import * as ko from 'knockout';
import styles from './KnockoutCrud.module.scss';
import { IKnockoutCrudWebPartProps } from './KnockoutCrudWebPart';

export interface IKnockoutCrudBindingContext extends
IKnockoutCrudWebPartProps {
    shouter: KnockoutSubscribable<{}>;
}

export default class KnockoutCrudViewModel {
    public listName: KnockoutObservable<string> = ko.observable('');

    public knockoutCrudClass: string = styles.knockoutCrud;
    public containerClass: string = styles.container;
    public rowClass: string = styles.row;
    public columnClass: string = styles.column;
    public titleClass: string = styles.title;
    public subTitleClass: string = styles.subTitle;
    public descriptionClass: string = styles.description;
    public buttonClass: string = styles.button;
    public labelClass: string = styles.label;

    constructor(bindings: IKnockoutCrudBindingContext) {
        this.listName(bindings.listName);
    }
}

```

```
// When web part description is updated, change this view model's
// description.
bindings.shouter.subscribe((value: string) => {
  this.listName(value);
}, this, 'listName');
}
```

7. In the template (KnockoutCrud.template.html) reflect the listName property



The screenshot shows the SharePoint Framework developer tools interface. On the left, there is an 'EXPLORER' sidebar with a tree view of project files. The 'KnockoutCrud.template.html' file is highlighted in blue at the bottom of the list. To the right of the sidebar, there are three tabs: 'KnockoutCrudWebPart.ts', 'KnockoutCrudViewModel.ts', and 'KnockoutCrud.template.html'. The 'KnockoutCrud.template.html' tab is active, displaying the HTML code for the web part template.

```

1 <div data-bind="attr: { class:knockoutCrudClass }">
2   <div data-bind="attr: { class:containerClass }">
3     <div data-bind="attr: { class:rowClass }">
4       <div data-bind="attr: { class:columnClass }">
5         <span data-bind="attr: { class:titleClass }>Welcome to SharePoint!</span>
6         <p data-bind="attr: { class:subTitleClass }">Customize SharePoint experiences using Web Parts.</p>
7         <p data-bind="attr: { class:descriptionClass }, text:listName"></p>
8         <a href="https://aka.ms/spfx" data-bind="attr: { class:buttonClass }">
9           <span data-bind="attr: { class:labelClass }">Learn more</span>
10        </a>
11      </div>
12    </div>
13  </div>
14 </div>

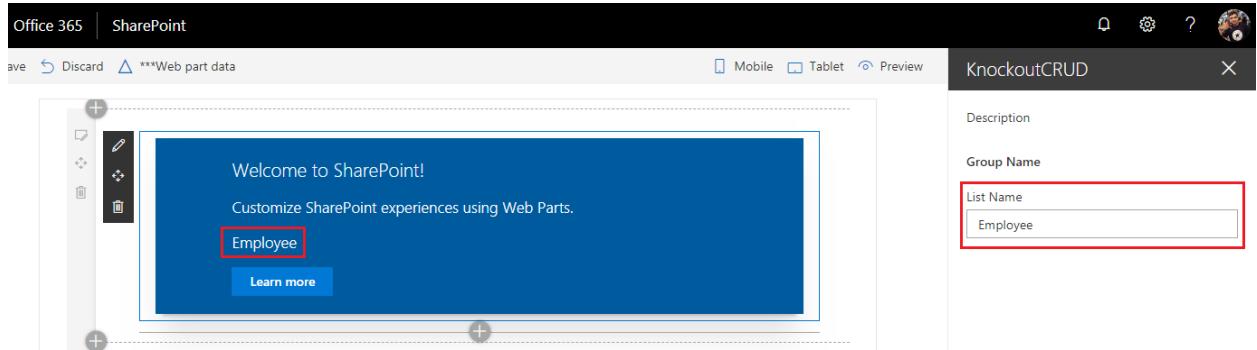
```

```

<div data-bind="attr: { class:knockoutCrudClass }">
  <div data-bind="attr: { class:containerClass }">
    <div data-bind="attr: { class:rowClass }">
      <div data-bind="attr: { class:columnClass }">
        <span data-bind="attr: { class:titleClass }>Welcome to
SharePoint!</span>
        <p data-bind="attr: { class:subTitleClass }">Customize
SharePoint experiences using Web Parts.</p>
        <p data-bind="attr: { class:descriptionClass },
text:listName"></p>
        <a href="https://aka.ms/spfx" data-bind="attr: {
class:buttonClass }">
          <span data-bind="attr: { class:labelClass }">Learn
more</span>
        </a>
      </div>
    </div>
  </div>
</div>

```

8. In the command prompt, type “gulp serve”
9. In the SharePoint local workbench page, add the web part.
10. Edit the web part to ensure the listName property pane field is getting reflected.



Configure ViewModel

1. Open KnockoutCrudViewModel.ts, add below import statements

```
import { IWebPartContext } from '@microsoft/sp-webpart-base';
import { SPHttpClient, SPHttpClientResponse } from '@microsoft/sp-http';
```

```
KnockoutCrudViewModel.ts ✘

1  import * as ko from 'knockout';
2  import styles from './KnockoutCrud.module.scss';
3  import { IKnockoutCrudWebPartProps } from './KnockoutCrudWebPart';
4  import { IWebPartContext } from '@microsoft/sp-webpart-base';
5  import { SPHttpClient, SPHttpClientResponse } from '@microsoft/sp-http';
```

2. Add context to interface IKnockoutCrudBindingContext

```
export interface IKnockoutCrudBindingContext extends
IKnockoutCrudWebPartProps {
  shouter: KnockoutSubscribable<{}>;
  context: IWebPartContext;
}
```

3. Add an interface for representing SharePoint list item

```
export interface IListItem {
  Id: number;
  Title: string;
}
```

```
KnockoutCrudViewModel.ts ✘

1 import * as ko from 'knockout';
2 import styles from './KnockoutCrud.module.scss';
3 import { IKnockoutCrudWebPartProps } from './KnockoutCrudWebPart';
4 import { IWebPartContext } from '@microsoft/sp-webpart-base';
5 import { SPHttpClient, SPHttpClientResponse } from '@microsoft/sp-http';
6
7 export interface IKnockoutCrudBindingContext extends IKnockoutCrudWebPartProps {
8   shouter: KnockoutSubscribable<{}>;
9   context: IWebPartContext;
10 }
11
12 export interface IListItem {
13   Id: number;
14   Title: string;
15 }
```

4. Add below properties to bind to UI

```
public message: KnockoutObservable<string> = ko.observable('');
```

5. Implement generic method to get latest item id

```
private getLatestItemId(): Promise<number> {
  return
this._context.spHttpClient.get(this._context.pageContext["web"]["absolut
eUrl"]
  +
`/_api/web/lists/GetByTitle('${this.listName}')/items?$orderby=Id
desc&$top=1&$select=id`, SPHttpClient.configurations.v1)
  .then((response: SPHttpClientResponse): Promise<any> => {
    return response.json();
  })
  .then((data: any): number => {
    this.message("Load succeeded");
    return data;
  },
  (error: any) => {
    this.message("Load failed");
  }) as Promise<number>;
}
```

Implement Create Operation

We will use the REST API to add the item to list.

```
private createItem(): void {
  const body: string = JSON.stringify({
    'Title': `Item ${new Date()}`)
  });

  this._context.spHttpClient.post(`.${this._context.pageContext["web"]}[{"absoluteUrl"}]/_api/web/lists/getbytitle('${this._listName}')/items`,
    SPHttpClient.configurations.v1,
  {
    headers: {
      'Accept': 'application/json;odata=nometadata',
      'Content-type': 'application/json;odata=nometadata',
      'odata-version': ''
    },
    body: body
  })
  .then((response: SPHttpClientResponse): Promise<IListItem> => {
    return response.json();
  })
  .then((item: IListItem): void => {
    this.message(`Item '${item.Title}' (ID: ${item.Id}) successfully created`);
  }, (error: any): void => {
    this.message('Error while creating the item: ' + error);
  });
}
```

Implement Read Operation

We will use REST API to read the latest item.

```
private readItem(): void {
  this.getLatestItemId()
  .then((itemId: number): Promise<SPHttpClientResponse> => {
    if (itemId === -1) {
      throw new Error('No items found in the list');
    }

    this.message(`Loading information about item ID: ${itemId}...`);

    return
  this._context.spHttpClient.get(`.${this._context.pageContext["web"]}[{"absoluteUrl"}]`)
```

```

        "]} /_api/web/lists/getbytitle('${this._listName}')/items(${itemId})?$select=Title,Id`,
        SPHttpClient.configurations.v1,
    {
        headers: {
            'Accept': 'application/json;odata=nometadata',
            'odata-version': ''
        }
    });
})
.then((response: SPHttpClientResponse): Promise<IListItem> => {
    return response.json();
})
.then((item: IListItem): void => {
    this.message(`Item ID: ${item.Id}, Title: ${item.Title}`);
}, (error: any): void => {
    this.message('Loading latest item failed with error: ' + error);
});
}
}

```

Implement Update Operation

We will use REST API to update the latest item.

```

private updateItem(): void {
    let latestItemId: number = undefined;
    this.message('Loading latest item...');

    this.getLatestItemId()
        .then((itemId: number): Promise<SPHttpClientResponse> => {
            if (itemId === -1) {
                throw new Error('No items found in the list');
            }

            latestItemId = itemId;
            this.message(`Loading information about item ID: ${itemId}...`);

            return
        })
        .then(() => {
            this._context.spHttpClient.get(`${this._context.pageContext["web"]]["absoluteUrl"]
                }/_api/web/lists/getbytitle('${this._listName}')/items(${latestItemId})?$sele
                ct=Title,Id`,
                SPHttpClient.configurations.v1,
            {
                headers: {
                    'Accept': 'application/json;odata=nometadata',

```

```

        'odata-version': ''
    }
});
})
.then((response: SPHttpClientResponse): Promise<IListItem> => {
    return response.json();
})
.then((item: IListItem): void => {
    this.message(`Item ID: ${item.Id}, Title: ${item.Title}`);
}

const body: string = JSON.stringify({
    'Title': `Updated Item ${new Date()}`});
});

this._context.spHttpClient.post(`${this._context.pageContext["web"]["absoluteUrl"]}/_api/web/lists/getbytitle('${this._listName}')/items(${item.Id})`,
    SPHttpClient.configurations.v1,
{
    headers: {
        'Accept': 'application/json;odata=nometadata',
        'Content-type': 'application/json;odata=nometadata',
        'odata-version': '',
        'IF-MATCH': '*',
        'X-HTTP-Method': 'MERGE'
    },
    body: body
})
.then((response: SPHttpClientResponse): void => {
    this.message(`Item with ID: ${latestItemId} successfully updated`);
}, (error: any): void => {
    this.message(`Error updating item: ${error}`);
});
});
});
}
}

```

Implement Delete Operation

We will use REST API to delete the latest item.

```

private deleteItem(): void {
    if (!window.confirm('Are you sure you want to delete the latest item?')) {
        return;
    }
}

```

```

this.message('Loading latest items...');
let latestItemId: number = undefined;
let etag: string = undefined;
this.getLatestItemId()
.then((itemId: number): Promise<SPHttpClientResponse> => {
  if (itemId === -1) {
    throw new Error('No items found in the list');
  }

  latestItemId = itemId;
  this.message(`Loading information about item ID: ${latestItemId}...`);
  return
this._context.spHttpClient.get(`${this._context.pageContext["web"]]["absoluteUrl"]
"]/_api/web/lists/getbytitle('${this._listName}')/items(${latestItemId})?$sele
ct=Id`,
  SPHttpClient.configurations.v1,
{
  headers: {
    'Accept': 'application/json;odata=nometadata',
    'odata-version': ''
  }
});
})
.then((response: SPHttpClientResponse): Promise<IListItem> => {
  etag = response.headers.get('ETag');
  return response.json();
})
.then((item: IListItem): Promise<SPHttpClientResponse> => {
  this.message(`Deleting item with ID: ${latestItemId}...`);
  return
this._context.spHttpClient.post(`${this._context.pageContext["web"]]["absoluteUr
l"]}/_api/web/lists/getbytitle('${this._listName}')/items(${item.Id})`,
  SPHttpClient.configurations.v1,
{
  headers: {
    'Accept': 'application/json;odata=nometadata',
    'Content-type': 'application/json;odata=verbose',
    'odata-version': '',
    'IF-MATCH': etag,
    'X-HTTP-Method': 'DELETE'
  }
});
})
.then((response: SPHttpClientResponse): void => {
  this.message(`Item with ID: ${latestItemId} successfully deleted`);
}
)

```

```

    }, (error: any): void => {
      this.message(`Error deleting item: ${error}`);
    });
}

```

6. In KnockoutCrudWebPart.ts file update the bindings in OnInit method

```

const bindings: IKnockoutCrudBindingContext = {
  listName: this.properties.listName,
  context: this.context,
  shouter: this._shouter
};

```

Add Controls to Knockout template

1. Open KnockoutCrud.template.html under “\src\webparts\knockoutCrud\” folder.
2. Import Angular module

```

import * as angular from 'angular';
import './app/app-module';

```

3. Modify Render method to include buttons for CRUD operations and add event handlers to each of the button

```

export default class AngularCrudWebPart extends
BaseClientSideWebPart<IAngularCrudWebPartProps> {
  private $injector: angular.auto.IInjectorService;

  public render(): void {
    if (this.renderedOnce === false) {
      this.domElement.innerHTML =
        <div class="${styles.angularCrud}" data-ng-
controller="HomeController as vm">
          <div class="${styles.container}">
            <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-
white ${styles.row}">
              <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2
ms-u-lgPush1">
                <span class="ms-font-xl ms-fontColor-white">
                  Sample SharePoint CRUD operations in Angular
                </span>
              </div>
            </div>
          <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-
white ${styles.row}">

```

```

        <div class="ms-Grid-col ms-u-lg10 ms-u-xl18 ms-u-xlPush2
ms-u-lgPush1">
            <uif-button ng-click="vm.createItem()" ng-
disabled="vm.listNotConfigured">Create item</uif-button>
            <uif-button ng-click="vm.readItem()" ng-
disabled="vm.listNotConfigured">Read item</uif-button>
        </div>
    </div>
    <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-
white ${styles.row}">
        <div class="ms-Grid-col ms-u-lg10 ms-u-xl18 ms-u-xlPush2
ms-u-lgPush1">
            <uif-button ng-click="vm.readItems()" ng-
disabled="vm.listNotConfigured">Read all items</uif-button>
        </div>
        <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-
white ${styles.row}">
            <div class="ms-Grid-col ms-u-lg10 ms-u-xl18 ms-u-xlPush2
ms-u-lgPush1">
                <uif-button ng-click="vm.updateItem()" ng-
disabled="vm.listNotConfigured">Update item</uif-button>
                <uif-button ng-click="vm.deleteItem()" ng-
disabled="vm.listNotConfigured">Delete item</uif-button>
            </div>
        </div>
        <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-
white ${styles.row}">
            <div class="ms-Grid-col ms-u-lg10 ms-u-xl18 ms-u-xlPush2
ms-u-lgPush1">
                <div>{{vm.status}}</div>
                <ul>
                    <li ng-repeat="item in vm.items">{{item.Title}}
({{item.Id}})</li>
                <ul>
                </div>
                </div>
            </div>
        </div>;
    
```

```

        this.$injector = angular.bootstrap(this.domElement, ['crudapp']);
    }

    this.$injector.get('$rootScope').$broadcast('configurationChanged',
{

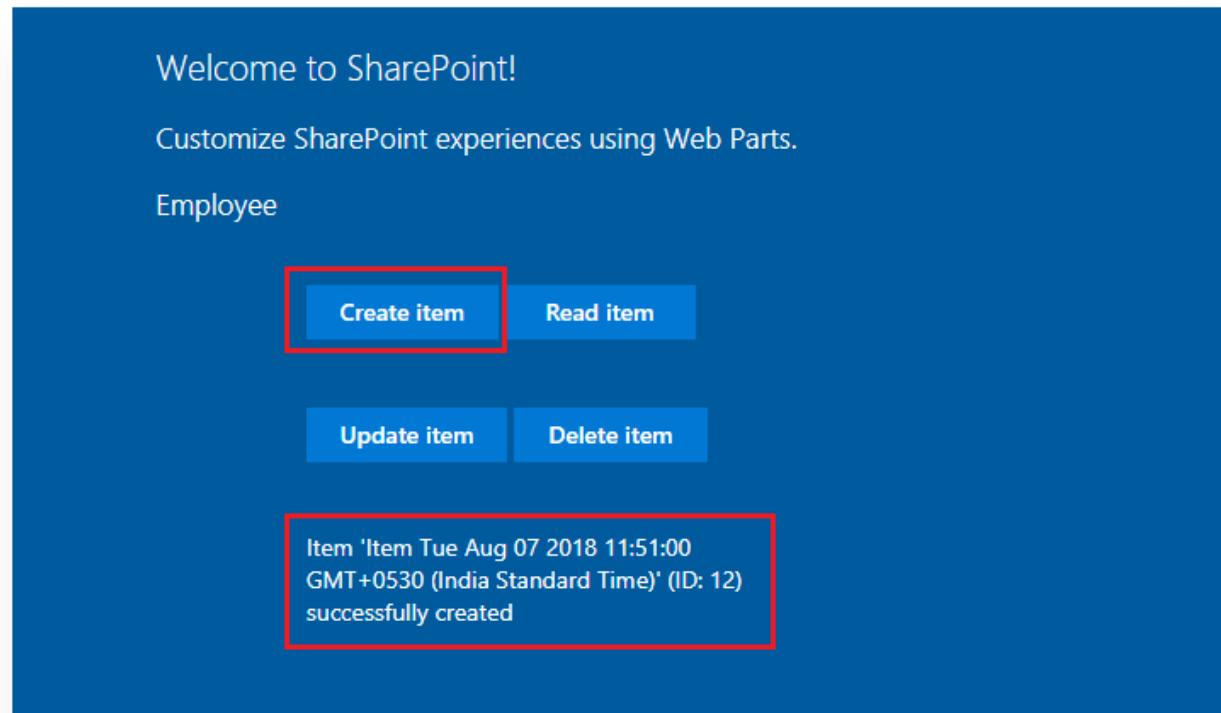
```

```
    webUrl: this.context.pageContext.web.absoluteUrl,  
    listName: this.properties.listName  
});  
}  
}
```

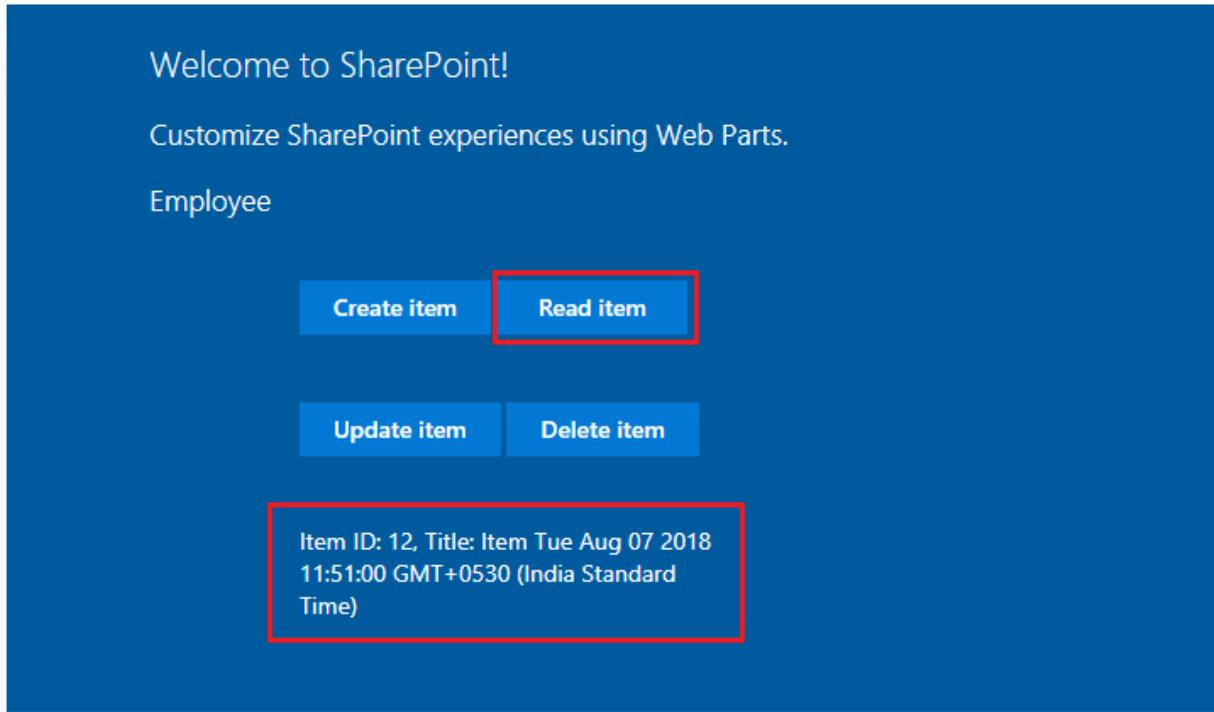
Test the WebPart

1. On the command prompt, type “gulp serve”
2. Open SharePoint site
3. Navigate to /_layouts/15/workbench.aspx
4. Add the webpart to page.
5. Edit webpart, in the properties pane type the list name
6. Click the buttons (Create Item, Read Item, Update Item, and Delete Item) one by one to test the webpart
7. Verify the operations are taking place in SharePoint list.

Create Operation



Read Operation



Welcome to SharePoint!

Customize SharePoint experiences using Web Parts.

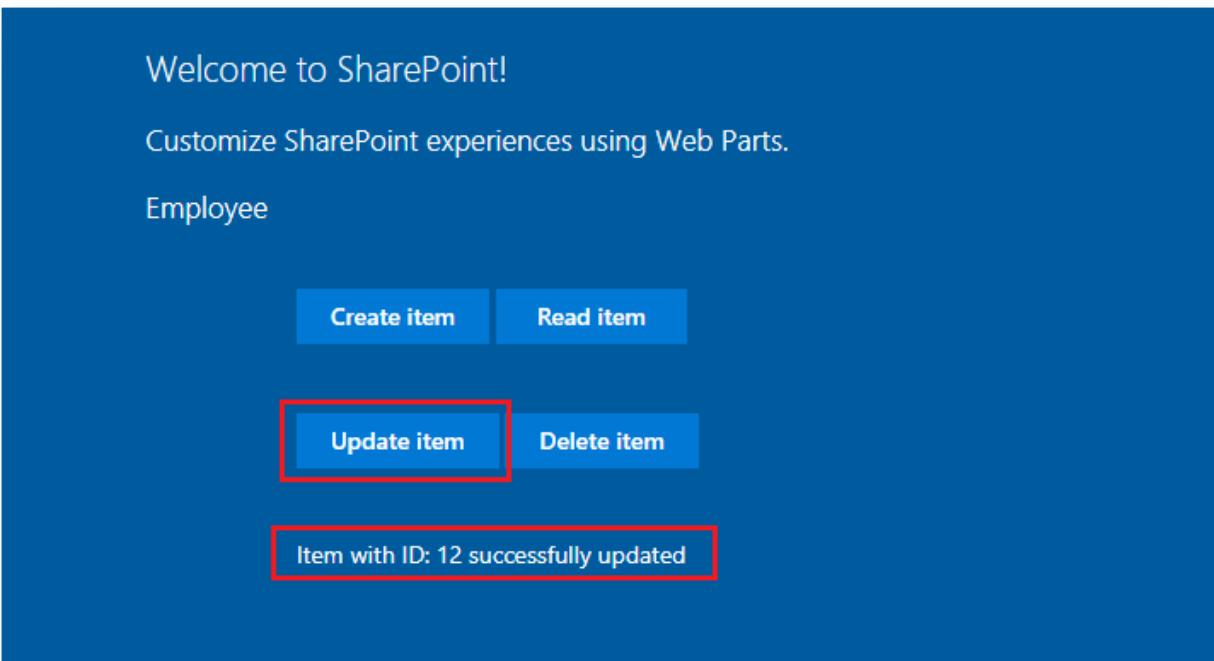
Employee

Create item **Read item**

Update item Delete item

Item ID: 12, Title: Item Tue Aug 07 2018
11:51:00 GMT+0530 (India Standard Time)

Update Operation



Welcome to SharePoint!

Customize SharePoint experiences using Web Parts.

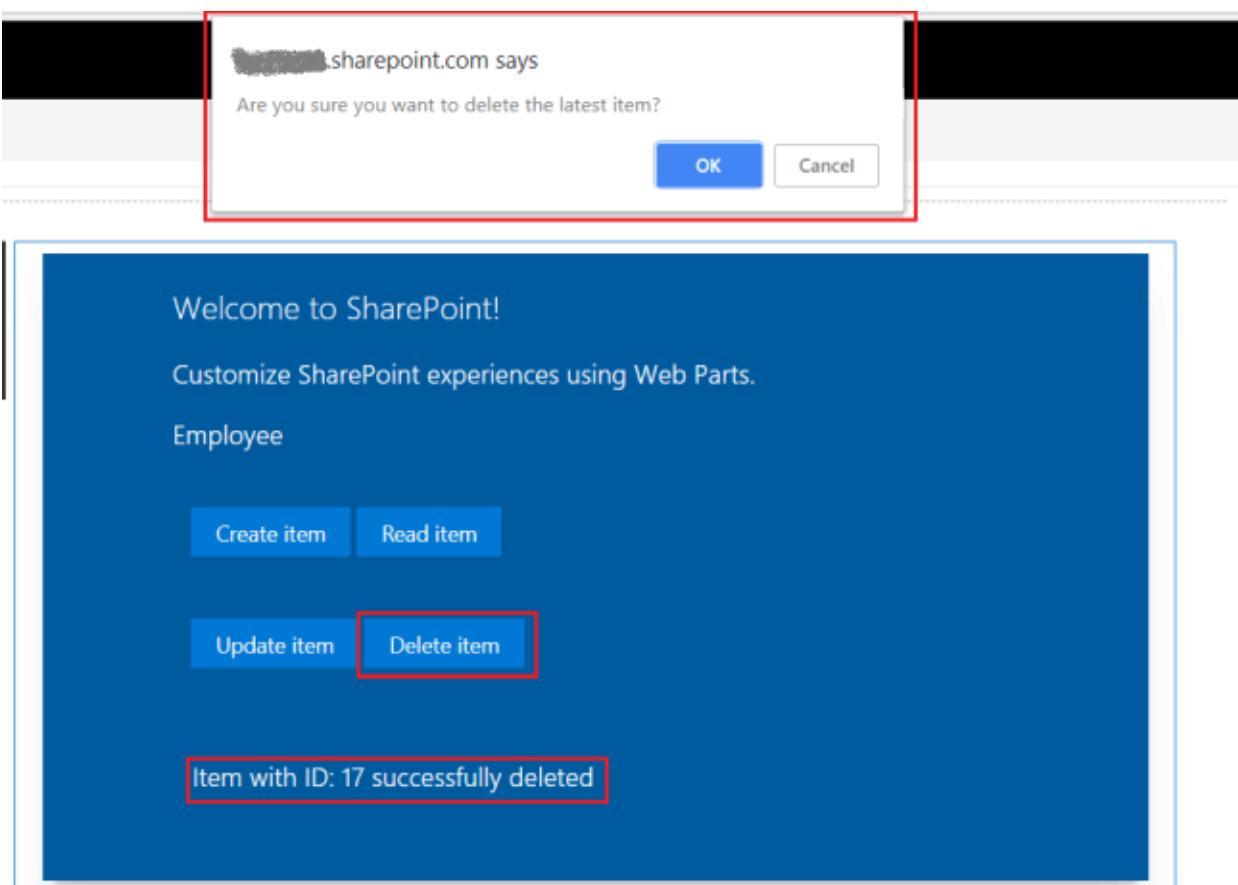
Employee

Create item Read item

Update item Delete item

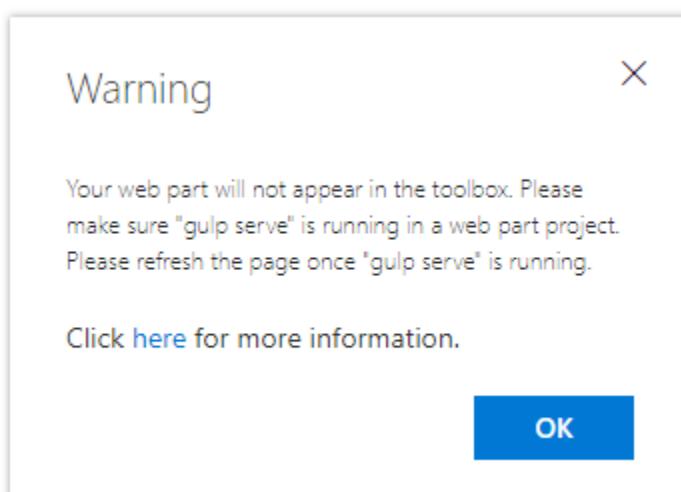
Item with ID: 12 successfully updated

Delete Operation



Troubleshooting

In some cases SharePoint workbench ([https://\[tenant\].sharepoint.com/_layouts/15/workbench.aspx](https://[tenant].sharepoint.com/_layouts/15/workbench.aspx)) shows below error although “gulp serve” is running.



Open below url in the next tab of browser. Accept the warning message.

<https://localhost:4321/temp/manifests.js>

Summary

React JS is natively supported by SharePoint framework. SPFx generates all needed React components for you to get started with the development. React JS supports the application to be broken down into small React components, which simplifies development and maintenance. React JS focuses more on UI development, in contrast to Angular which is more famous for creating SPA (Single Page Applications).

Chapter 14: CRUD operations using SP PNP JS

Overview

In the chapters so far, we have explored on developing basic SharePoint client web part which can run independently without any interaction with SharePoint.

In this chapter, we will explore to interact with the SharePoint list for CRUD (Create, Read, Update, and Delete) operations using external JavaScript library called SP PNP JS. The CRUS operations will be performed using REST APIs.

Brief about SP-PnP-JS

SP PnP JS is pattern and practices core JavaScript library offers simplified common operations with SharePoint to help developers concentrate on business logic without worrying much about the underlying technical implementation. It contains fluent APIs to work with SharePoint REST APIs. Read more about it at - <https://www.npmjs.com/package/sp-pnp-js>

Create SPFx Solution

1. Open command prompt. Create a directory for SPFx solution

```
md spfx-crud-sppnpjs
```

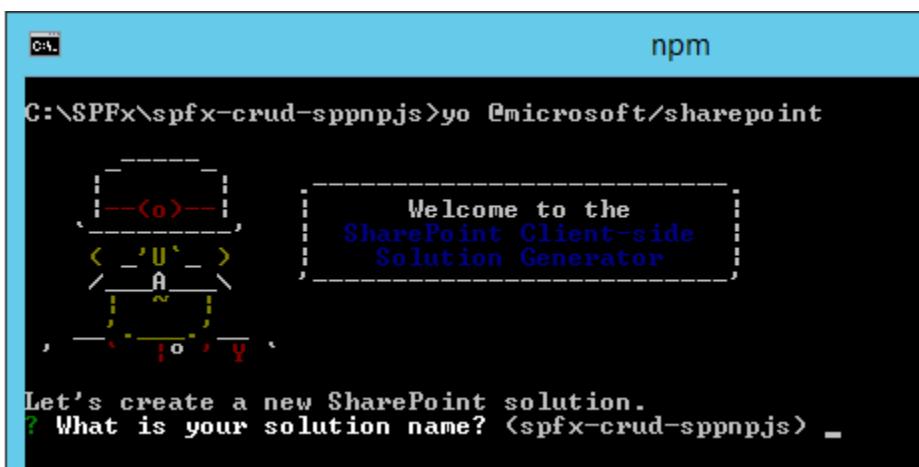
2. Navigate to above created directory

```
cd spfx-crud-sppnpjs
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



The screenshot shows a terminal window with the npm logo at the top. The command 'yo @microsoft/sharepoint' is being typed. A blue dashed box highlights the text 'Welcome to the SharePoint Client-side Solution Generator'. Below the command, the text 'Let's create a new SharePoint solution.' is displayed, followed by a question mark and the placeholder text '<spfx-crud-sppnpjs>'.

Solution Name: Hit enter to have default name (spfx-crud-no-framework in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension. Choose webpart option.

Selected choice: WebPart

Web part name: Hit enter to select the default name or type in any other name.

Selected choice: SPPnPJSCRUD

Web part description: Hit enter to select the default description or type in any other value.

Selected choice: CRUD operations with SP PnP JS

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)

Selected choice: No JavaScript Framework

5. Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take significant amount of time.

6. Once the scaffolding process is completed, lock down the version of project dependencies by running below command

```
npm shrinkwrap
```

7. In the command prompt type below command to open the solution in code editor of your choice.

```
code .
```

Configure sp-pnp-js

In the command prompt, run below command to install sp-pnp-js

```
npm install sp-pnp-js --save
```

Configure Property for List Name

SPFx solution by default have description property created. Let us change the property to list name. We will use this property to configure the list name on which the CRUD operation is to perform.

1. Open mystrings.d.ts under \src\webparts\spPnPjscrud\loc\ folder
2. Rename DescriptionFieldLabel to ListNameFieldLabel



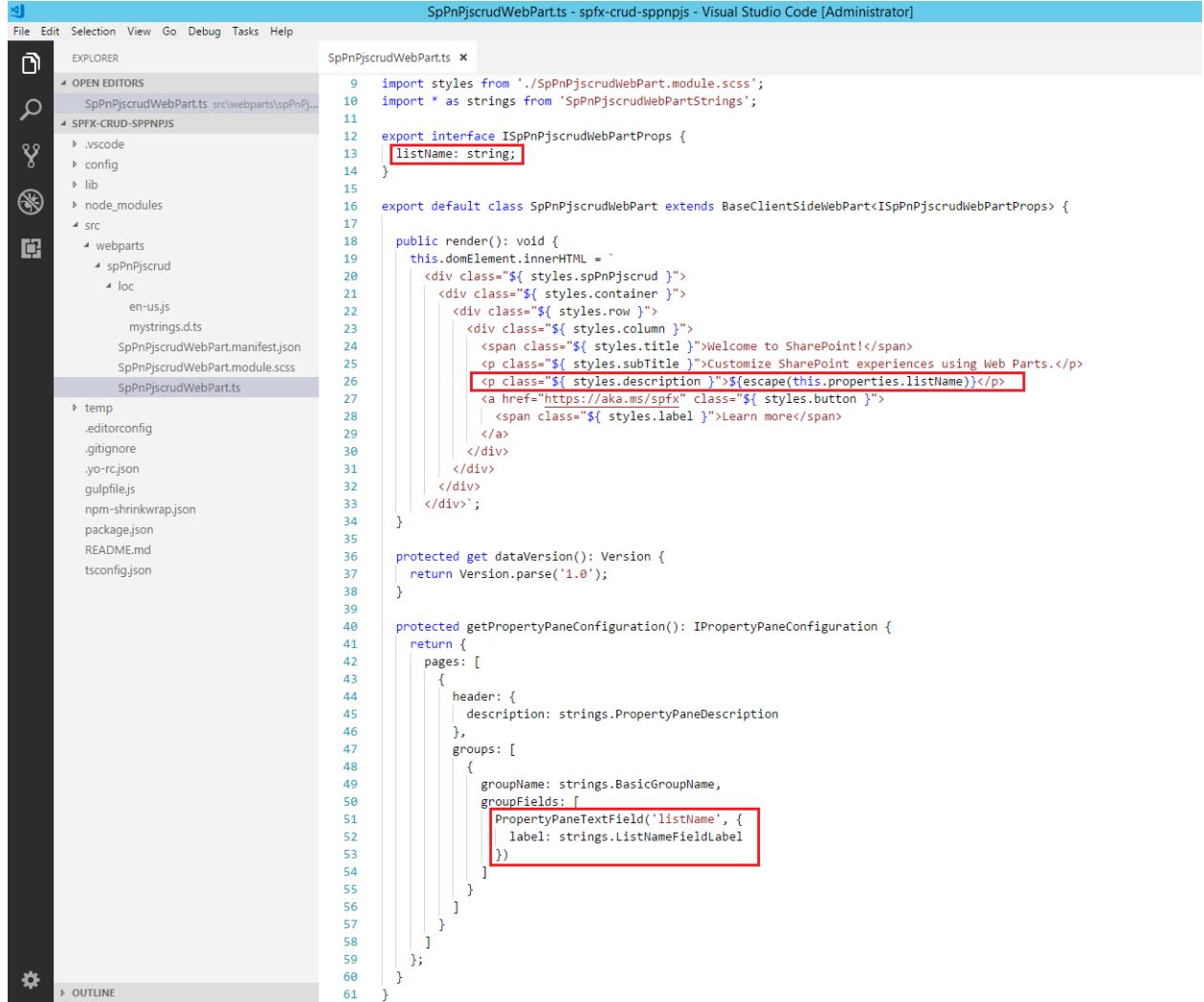
```
mystrings.d.ts x
1 declare interface ISpPnPjscrudWebPartStrings {
2   PropertyPaneDescription: string;
3   BasicGroupName: string;
4   ListNameFieldLabel: string;
5 }
6
7 declare module 'SpPnPjscrudWebPartStrings' {
8   const strings: ISpPnPjscrudWebPartStrings;
9   export = strings;
10 }
11
```

3. In en-us.js file under \src\webparts\spPnPjscrud\loc\ folder set the display name for listName property



```
en-us.js x
1 define([], function() {
2   return {
3     "PropertyPaneDescription": "Description",
4     "BasicGroupName": "Group Name",
5     "ListNameFieldLabel": "List Name"
6   }
7});
```

4. Open main webpart file (SpPnPjscrudWebPart.ts) under \src\webparts\spPnPjscrud folder.
5. Rename description property pane field to `listName`



```

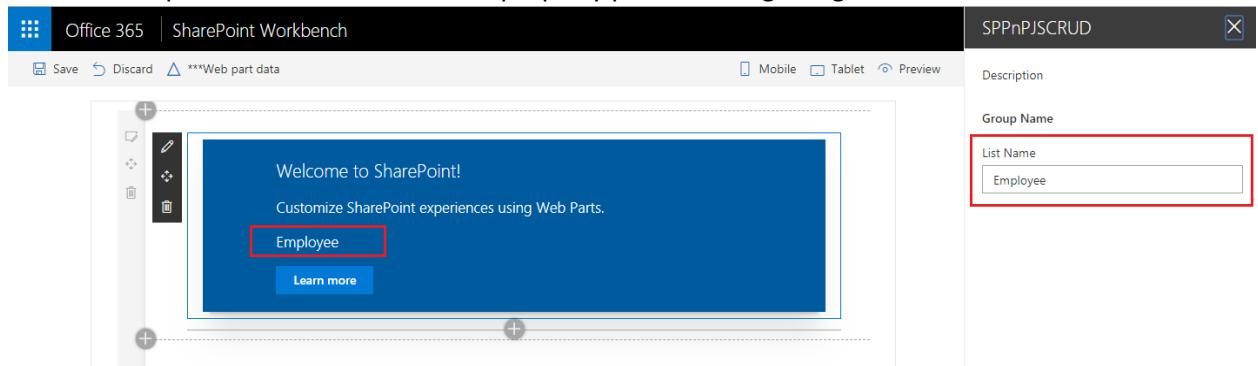
File Edit Selection View Go Debug Tasks Help
OPEN EDITORS
  SpPnPjscrudWebPart.ts src\webparts\spPnPj...
  SPFX-CRUD-SPPNPJS
  .vscode
  config
  lib
  node_modules
  src
    webparts
      spPnPjscrud
        loc
          en-us.js
          mystrings.d.ts
        SpPnPjscrudWebPart.manifest.json
        SpPnPjscrudWebPart.module.scss
        SpPnPjscrudWebPart.ts
  temp
  .editorconfig
  .gitignore
  .yo-rc.json
  gulpfile.js
  npm-shrinkwrap.json
  package.json
  README.md
  tsconfig.json

  OUTLINE

SpPnPjscrudWebPart.ts x
9 import styles from './SpPnPjscrudWebPart.module.scss';
10 import * as strings from 'SpPnPjscrudWebPartStrings';
11
12 export interface ISpPnPjscrudWebPartProps {
13   listName: string;
14 }
15
16 export default class SpPnPjscrudWebPart extends BaseClientSideWebPart<ISpPnPjscrudWebPartProps> {
17
18   public render(): void {
19     this.domElement.innerHTML = `
20       <div class="${ styles.spPnPjscrud }">
21         <div class="${ styles.container }">
22           <div class="${ styles.row }">
23             <div class="${ styles.column }">
24               <span class="${ styles.title }">Welcome to SharePoint!</span>
25               <p class="${ styles.subtitle }">Customize SharePoint experiences using Web Parts.</p>
26               <p class="${ styles.description }">${ escape(this.properties.listName) }</p>
27               <a href="https://aka.ms/spfx" class="${ styles.button }">
28                 <span class="${ styles.label }">Learn more</span>
29               </a>
30             </div>
31           </div>
32         </div>
33       </div>;
34     `;
35   }
36
37   protected getDataVersion(): Version {
38     return Version.parse('1.0');
39   }
40
41   protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
42     return {
43       pages: [
44         {
45           header: {
46             description: strings.PropertyPaneDescription
47           },
48           groups: [
49             {
50               groupName: strings.BasicGroupName,
51               groupFields: [
52                 PropertyPaneTextField('listName', {
53                   label: strings.ListNameFieldLabel
54                 })
55               ]
56             }
57           ]
58         };
59       }
60     };
61   }
}

```

6. In the command prompt, type “gulp serve”
7. In the SharePoint local workbench page, add the web part.
8. Edit the web part to ensure the `listName` property pane field is getting reflected.



Model for List Item

Add a class (`IListItem.ts`) representing the list item.



The screenshot shows the VS Code interface with the following details:

- EXPLORER** view: Shows the project structure:
 - OPEN EDITORS**: SpPnPjscrudWebPart.ts, IListItem.ts
 - SPFX-CRUD-SPPNPJS**: .vscode, config, dist, lib, node_modules, src (with webparts, spPnPjscrud, loc), IListItem.ts (highlighted with a red box)
 - SpPnPjscrudWebPart.manifest.json, SpPnPjscrudWebPart.module.scss, SpPnPjscrudWebPart.ts
- IListItem.ts** editor view: Contains the following TypeScript code:

```
1 export interface IListItem {  
2   Title?: string;  
3   Id: number;  
4 }  
5
```

Add Controls to WebPart

1. Open main webpart file (`SpPnPjscrudWebPart.ts`) under `\src\webparts\spPnPjscrud` folder.
2. Modify `Render` method to include buttons for CRUD operations and add event handlers to each of the button

File Edit Selection View Go Debug Tasks Help

EXPLORER

```
SpPnPjscrudWebPart.ts - spfx-crud-sppnpjs - Visual Studio Code [Administrator]
```

```

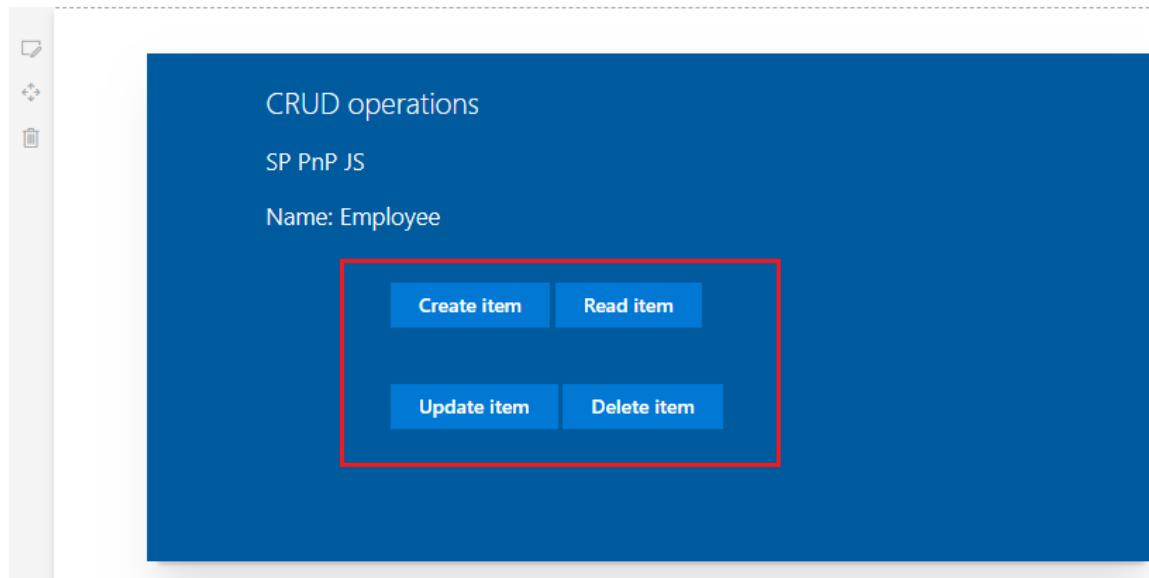
16  export default class SpPnPjscrudWebPart extends BaseClientSideWebPart<ISpPnPjscrudWebPartProps> {
17
18    public render(): void {
19      this.domElement.innerHTML =
20        <div class="${ styles.spPnPjscrud }">
21          <div class="${ styles.container }">
22              <div class="${ styles.row }">
23                  <div class="${ styles.column }">
24                      <span class="${ styles.title }">CRUD operations</span>
25                      <p class="${ styles.subtitle }">SP PnP JS</p>
26                      <p class="${ styles.description }">Name: ${escape(this.properties.listName)}</p>
27
28                  <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
29                      <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
30                          <button class="${styles.button} create-Button">
31                              | <span class="${styles.label}">Create item</span>
32                          </button>
33                          <button class="${styles.button} read-Button">
34                              | <span class="${styles.label}">Read item</span>
35                          </button>
36                      </div>
37                  </div>
38
39                  <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
40                      <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
41                          <button class="${styles.button} update-Button">
42                              | <span class="${styles.label}">Update item</span>
43                          </button>
44                          <button class="${styles.button} delete-Button">
45                              | <span class="${styles.label}">Delete item</span>
46                          </button>
47                      </div>
48                  </div>
49
50                  <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
51                      <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
52                          <div class="status"></div>
53                          <ul class="items"><ul>
54                              </div>
55                      </div>
56
57                      <div>
58                          <div>
59                          </div>
60                      </div>
61                  </div>
62
63                  this.setButtonsEventHandlers();
64
65
66      private setButtonsEventHandlers(): void {
67          const webPart: SpPnPjscrudWebPart = this;
68          this.domElement.querySelector('button.create-Button').addEventListener('click', () => { webPart.createItem(); });
69          this.domElement.querySelector('button.read-Button').addEventListener('click', () => { webPart.readItem(); });
70          this.domElement.querySelector('button.update-Button').addEventListener('click', () => { webPart.updateItem(); });
71          this.domElement.querySelector('button.delete-Button').addEventListener('click', () => { webPart.deleteItem(); });
72
73
74      private createItem(): void {
75
76      private readItem(): void {
77
78      private updateItem(): void {
79
80      private deleteItem(): void {
81
82
83
84

```

Buttons

Button Event Handlers

- In the command prompt type “gulp serve” to see the buttons on webpart.



4. We will use the sp-pnp-js APIs to perform CRUD operations. Let us implement generic method which will return the id of latest item from given list using sp-pnp-js APIs.

```
private getLatestItemId(): Promise<number> {
  return new Promise<number>((resolve: (itemId: number) => void, reject: (error: any) => void): void => {
    sp.web.lists.getByTitle(this.properties.listName)
      .items.orderBy('Id', false).top(1).select('Id').get()
      .then((items: { Id: number }[]): void => {
        if (items.length === 0) {
          resolve(-1);
        }
        else {
          resolve(items[0].Id);
        }
      }, (error: any): void => {
        reject(error);
      });
  });
}
```

Import sp-pnp-js

Add below import statements to main web part.

```
SpPnPscrudWebPart.ts ✘

11
12  import { IListItem } from './IListItem';
13  import pnp, { sp, Item, ItemAddResult, ItemUpdateResult } from "sp-pnp-js";
14
```

Implement Create Operation

We will use the sp-pnp-js API of items.add to add the item to list

```
private createItem(): void {
  this.updateStatus('Creating item...');

  sp.web.lists.getByTitle(this.properties.listName).items.add({
    'Title': `Item ${new Date()}`
  }).then((result: ItemAddResult): void => {
    const item: IListItem = result.data as IListItem;
    this.updateStatus(`Item '${item.Title}' (ID: ${item.Id}) successfully created`);
  }, (error: any): void => {
    this.updateStatus('Error while creating the item: ' + error);
  });
}
```

Implement Read Operation

We will use sp-pnp-js API - getById to read the item.

```
private readItem(): void {
  this.updateStatus('Reading latest items...');

  this.getLatestItemId()
    .then((itemId: number): Promise<IListItem> => {
      if (itemId === -1) {
        throw new Error('No items found in the list');
      }

      this.updateStatus(`Loading information about item ID: ${itemId}...`);
      return sp.web.lists.getByTitle(this.properties.listName)
        .items.getById(itemId).select('Title', 'Id').get();
    })
    .then((item: IListItem): void => {
      this.updateStatus(`Item ID: ${item.Id}, Title: ${item.Title}`);
    }, (error: any): void => {
      this.updateStatus('Loading latest item failed with error: ' + error);
    });
}
```

Implement Update Operation

We will use sp-pnp-js API - update

```
private updateItem(): void {
    this.updateStatus('Loading latest items...');
    let latestItemId: number = undefined;
    let etag: string = undefined;

    this.getLatestItemId()
        .then((itemId: number): Promise<Item> => {
            if (itemId === -1) {
                throw new Error('No items found in the list');
            }

            latestItemId = itemId;
            this.updateStatus(`Loading information about item ID: ${itemId}...`);
            return sp.web.lists.getByTitle(this.properties.listName)
                .items.getById(itemId).get(undefined, {
                    headers: {
                        'Accept': 'application/json;odata=minimalmetadata'
                    }
                });
        })
        .then((item: Item): Promise<IListItem> => {
            etag = item["odata.etag"];
            return Promise.resolve((item as any) as IListItem);
        })
        .then((item: IListItem): Promise<ItemUpdateResult> => {
            return sp.web.lists.getByTitle(this.properties.listName)
                .items.getById(item.Id).update({
                    'Title': `Updated Item ${new Date()}`
                }, etag);
        })
        .then((result: ItemUpdateResult): void => {
            this.updateStatus(`Item with ID: ${latestItemId} successfully updated`);
        }, (error: any): void => {
            this.updateStatus(`Loading latest item failed with error: ' + error);
        });
}
```

Implement Delete Operation

We will use sp-pnp-js API - delete

```

private deleteItem(): void {
  if (!window.confirm('Are you sure you want to delete the latest item?')) {
    return;
  }

  this.updateStatus('Loading latest items...');
  let latestItemId: number = undefined;
  let etag: string = undefined;
  this.getLatestItemId()
    .then((itemId: number): Promise<Item> => {
      if (itemId === -1) {
        throw new Error('No items found in the list');
      }

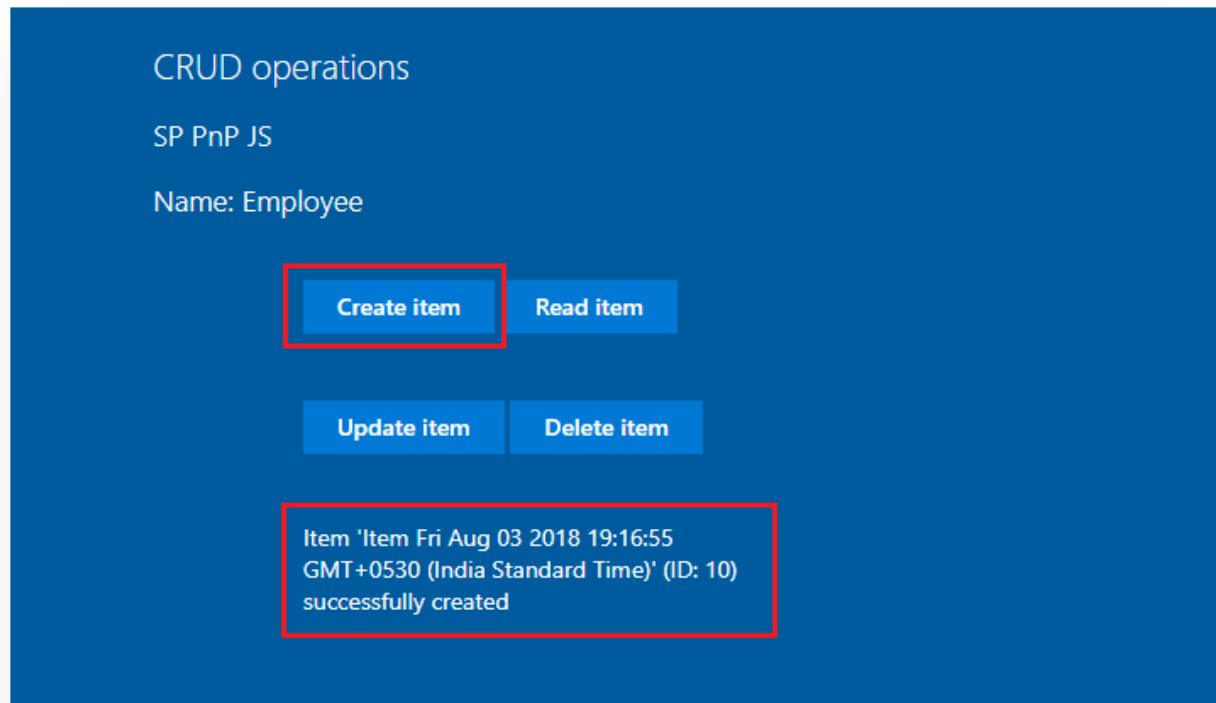
      latestItemId = itemId;
      this.updateStatus(`Loading information about item ID: ${latestItemId}...`);
      return sp.web.lists.getByTitle(this.properties.listName)
        .items.getById(latestItemId).select('Id').get(undefined, {
          headers: {
            'Accept': 'application/json;odata=minimalmetadata'
          }
        });
    })
    .then((item: Item): Promise<IListItem> => {
      etag = item["odata.etag"];
      return Promise.resolve((item as any) as IListItem);
    })
    .then((item: IListItem): Promise<void> => {
      this.updateStatus(`Deleting item with ID: ${latestItemId}...`);
      return sp.web.lists.getByTitle(this.properties.listName)
        .items.getById(item.Id).delete(etag);
    })
    .then((): void => {
      this.updateStatus(`Item with ID: ${latestItemId} successfully deleted`);
    }, (error: any): void => {
      this.updateStatus(`Error deleting item: ${error}`);
    });
}
}

```

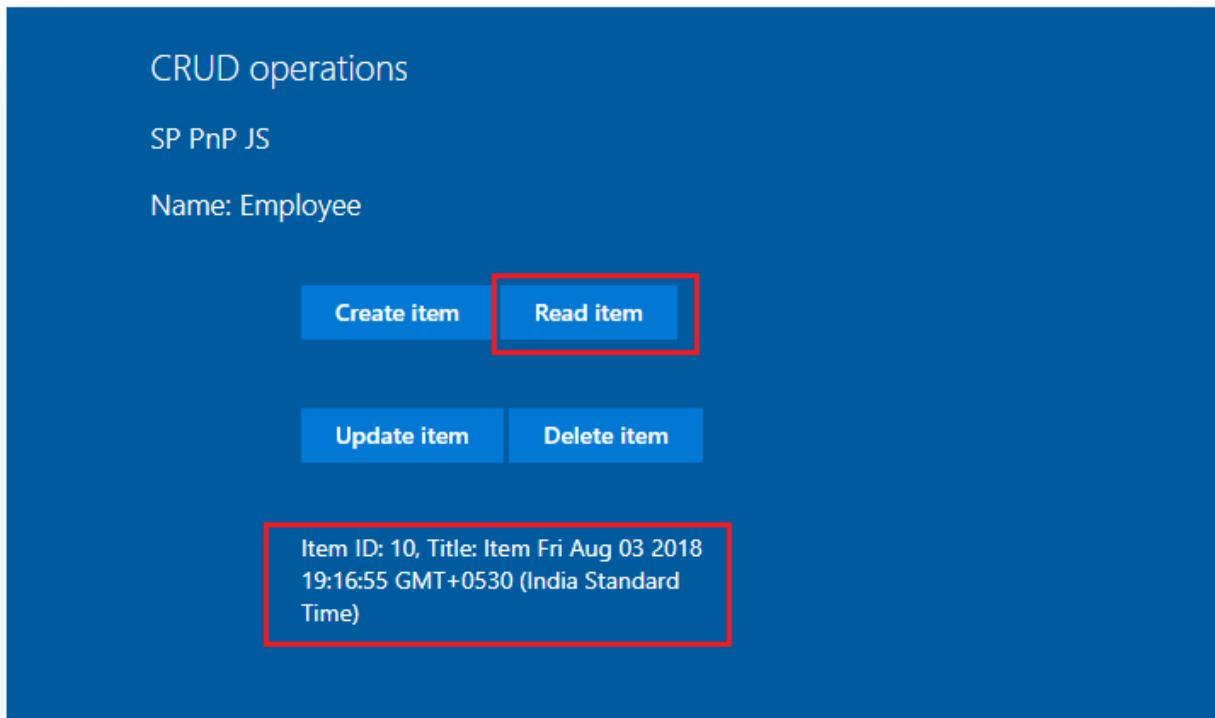
Test the WebPart

1. On the command prompt, type “gulp serve”
2. Open SharePoint site
3. Navigate to /_layouts/15/workbench.aspx
4. Add the webpart to page.
5. Edit webpart, in the properties pane type the list name
6. Click the buttons (Create Item, Read Item, Update Item, and Delete Item) one by one to test the webpart
7. Verify the operations are taking place in SharePoint list.

Create Operation

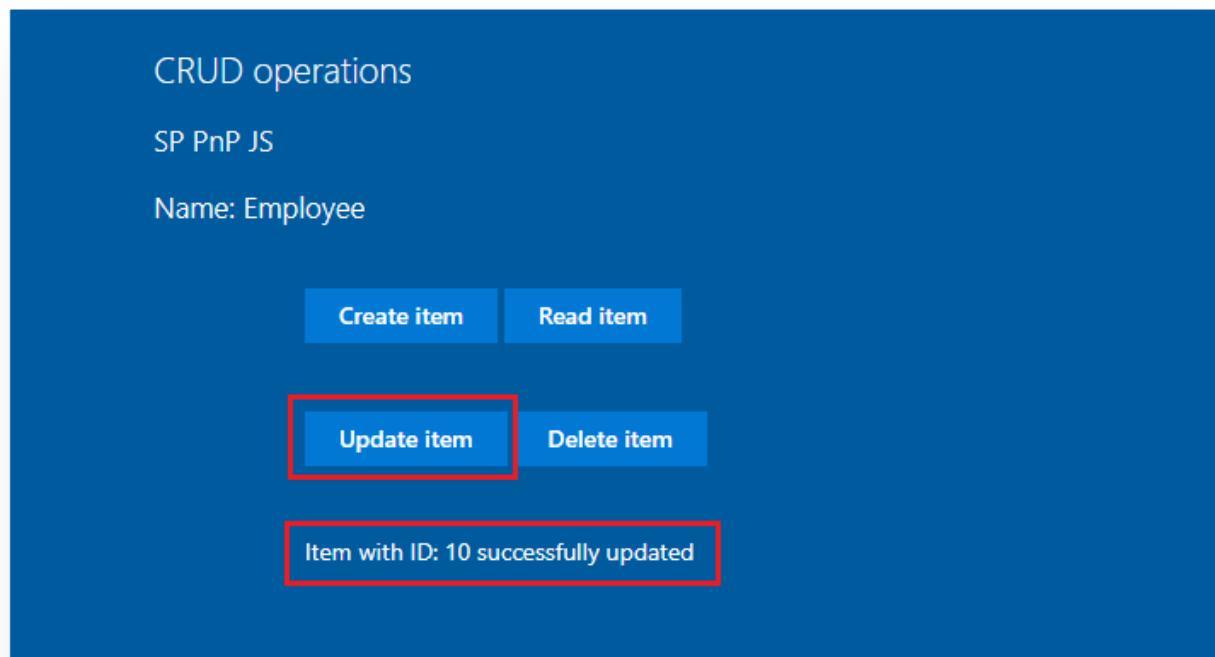


Read Operation



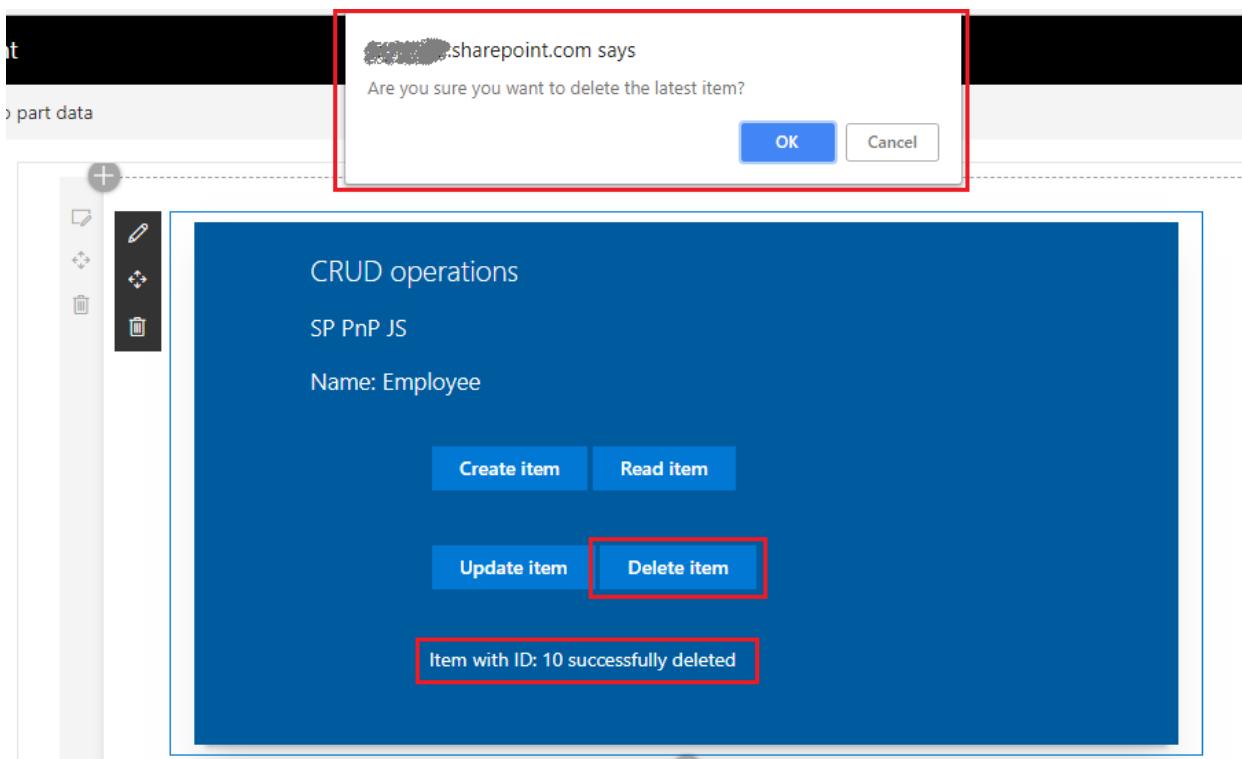
The screenshot shows a blue-themed user interface for 'CRUD operations'. At the top, it says 'SP PnP JS' and 'Name: Employee'. Below are four buttons: 'Create item', 'Read item' (which is highlighted with a red border), 'Update item', and 'Delete item'. A red-bordered callout box contains the text: 'Item ID: 10, Title: Item Fri Aug 03 2018 19:16:55 GMT+0530 (India Standard Time)'.

Update Operation



The screenshot shows a blue-themed user interface for 'CRUD operations'. At the top, it says 'SP PnP JS' and 'Name: Employee'. Below are four buttons: 'Create item', 'Read item', 'Update item' (which is highlighted with a red border), and 'Delete item'. A red-bordered callout box contains the text: 'Item with ID: 10 successfully updated'.

Delete Operation



Summary

sp-pnp-js APIs helps to perform common operations (like CRUD) with SharePoint easily. It makes the code lesser to maintain. Developer can concentrate on business logic rather than worrying about the identifying and using various REST APIs to use in the code.

Chapter 15: Integrating Office UI Fabric

Overview

SharePoint Framework (SPFx) offers lightweight, responsive client side web part development for classic as well modern SharePoint. In this chapter, we will explore how we can integrate the Office UI Fabric in to SharePoint Framework.

The UI Challenges

In the SharePoint journey so far, we have been developing custom web parts. Developing the visual interface for each of the custom web part had involved significant efforts from SharePoint developers as well UX designers and UI developers to make the web part look like an integral part of the SharePoint portal. Huge efforts were invested in designing the CSS classes and placing the required controls (like labels, textboxes, buttons, etc.) on the web part. This also had its own challenge to rewrite while re-branding the SharePoint portals.

Office UI Fabric handles this challenges gracefully for SharePoint developers and designers so that they can build the client side web parts which can resemble their appearance to Office 365 and provide responsiveness.

Brief information about Office UI Fabric

Office UI Fabric is official front end framework to build the experiences that fit seamlessly in Office 365. The Office UI Fabric components are built with React JS. Microsoft uses Fabric Core and Fabric React which offers numerous components and styles. Read more about Office UI Fabric at <https://developer.microsoft.com/en-us/fabric>

Office UI Fabric for SharePoint Framework

SharePoint Framework Fabric Core npm package (@microsoft/sp-office-ui-fabric-core) is a subset of Fabric core styles that can be easily integrated with SharePoint Framework. Yeoman generator for SharePoint Framework (from version 1.3.4 onwards) supports Office UI Fabric by default.

To install Fabric Core package in your existing project, use below command

```
npm install @microsoft/sp-office-ui-fabric-core --save
```

To use Fabric Core styles, use below declaration

```
@import '~@microsoft/sp-office-ui-fabric-core/dist/sass/SPFabricCore.scss';
```

However, it is recommended to use Office UI Fabric React package with SharePoint Framework.

Create SPFx Solution

1. Open command prompt. Create a directory for SPFx solution

```
md spfx-officeuifabric
```

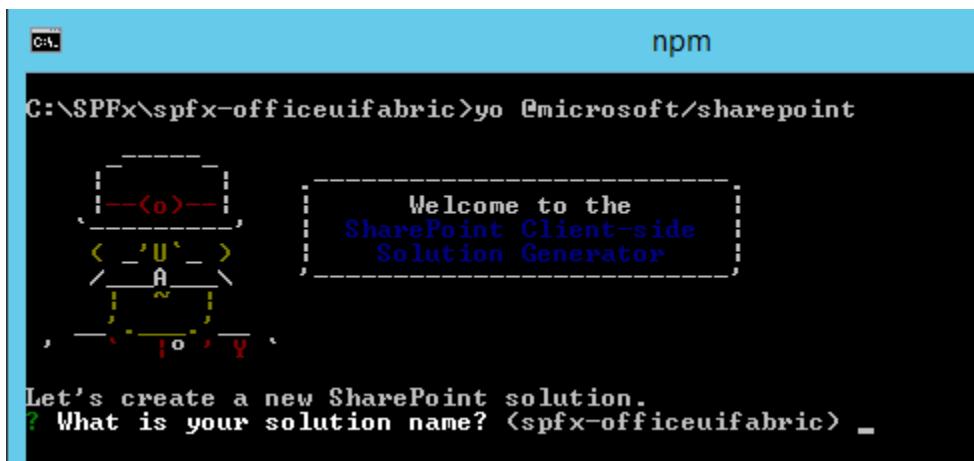
2. Navigate to above created directory

```
cd spfx-officeuifabric
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



The screenshot shows a terminal window titled 'npm'. The command 'C:\SPFx\spfx-officeuifabric>yo @microsoft/sharepoint' is being run. The output shows the Yeoman SharePoint Client-side Solution Generator wizard. It displays a logo consisting of various colored letters (O, U, A, T, I, O, Y) and asks 'Welcome to the SharePoint Client-side Solution Generator'. Below it, it says 'Let's create a new SharePoint solution.' followed by a question mark '?'.

Solution Name: Hit enter to have default name (spfx-officeuifabric in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension. Choose webpart option.

Selected choice: WebPart

Web part name: Hit enter to select the default name or type in any other name.

Selected choice: OfficeUIFabricExamples

Web part description: Hit enter to select the default description or type in any other value.

Selected choice: Office UI Fabric Integration with SPFx

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)

Selected choice: React

5. Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take significant amount of time.

6. Once the scaffolding process is completed, lock down the version of project dependencies by running below command

```
npm shrinkwrap
```

7. In the command prompt type below command to open the solution in code editor of your choice.

```
code .
```

Office UI Fabric Components

Office UI Fabric majorly have below categorized components:

Basic Inputs

- Button
- Checkbox
- ChoiceGroup
- ComboBox
- ContextualMenu
- Dropdown
- Label
- Link
- Rating
- Slider
- SpinButton
- TextField
- Toggle

Navigation

- Breadcrumb
- CommandBar
- Nav
- OverflowSet
- Pivot
- SearchBox

Content

- ActivityItem
- Calendar
- DetailsList
- Facepile
- GroupedList
- Icon
- Image
- List
- Persona

Pickers

- Pickers
- ColorPicker
- DatePicker
- PeoplePicker
- SwatchColorPicker

Progress & Validation

- MessageBar
- ProgressIndicator
- Shimmer
- Spinner

Surfaces

- Callout
- Dialog

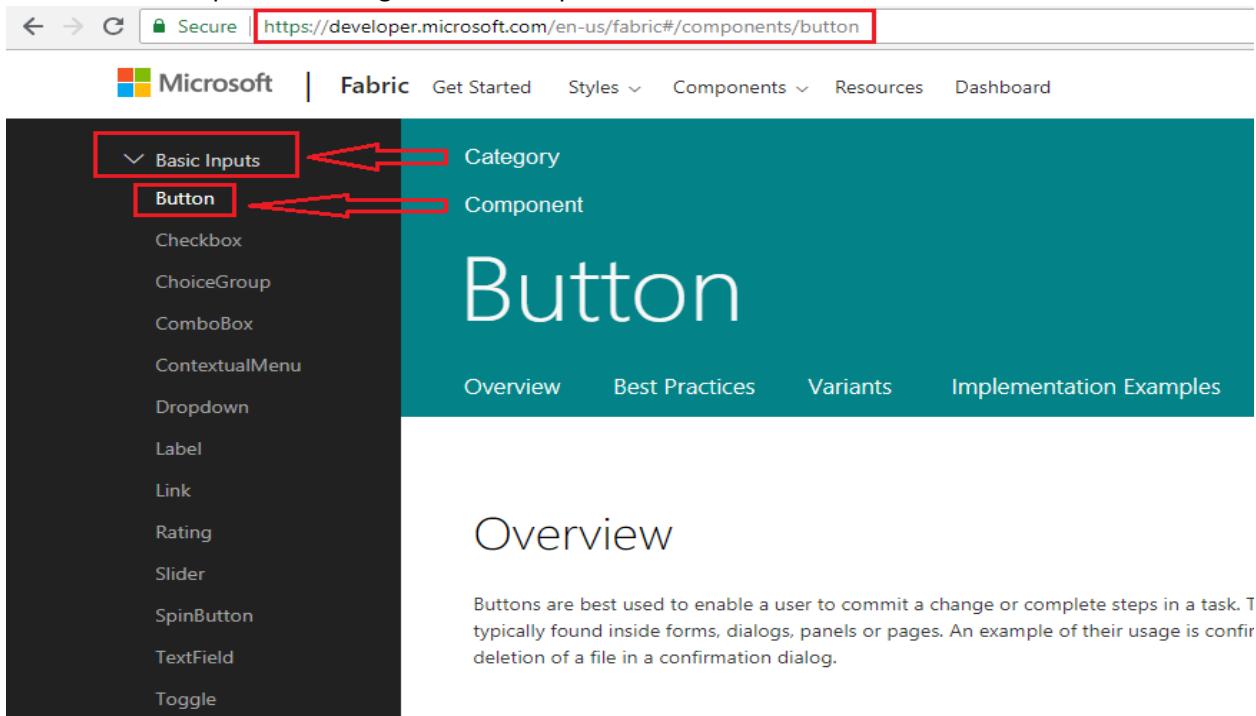
- DocumentCard
- HoverCard
- Layer
- Modal
- Overlay
- Panel
- ScrollablePane
- TeachingBubble
- Tooltip
- Coachmark

Utilities

- FocusTrapZone
- FocusZone
- MarqueeSelection
- ResizeGroup
- Selection
- Themes

How to use Office UI Fabric Components in SPFx WebPart

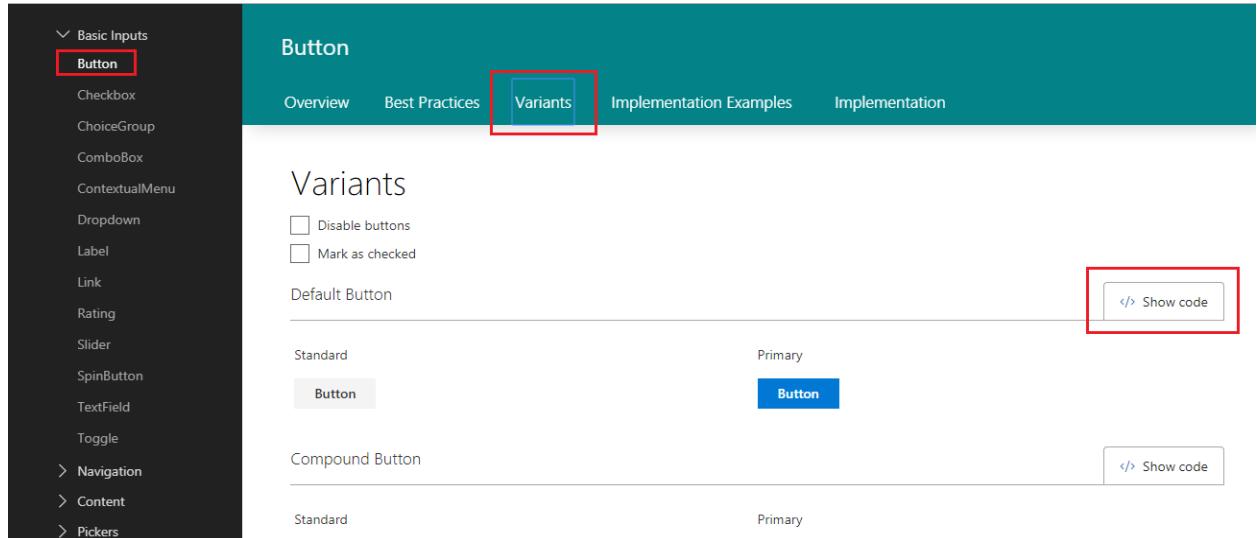
1. Navigate to Office UI Fabric Components site at <https://developer.microsoft.com/en-us/fabric#/components>
2. Select and expand the Category of your component
3. Click the component node
4. Below is an example of selecting a button component



The screenshot shows the Microsoft Fabric Components website. The URL in the address bar is <https://developer.microsoft.com/en-us/fabric#/components/button>. The page title is "Button". On the left, there is a sidebar with a tree view of components. The "Basic Inputs" category is expanded, and the "Button" component is selected. Red arrows point from the text labels "Category" and "Component" to the respective parts of the sidebar. The main content area has a large "Button" heading and a "Overview" section with the following text:

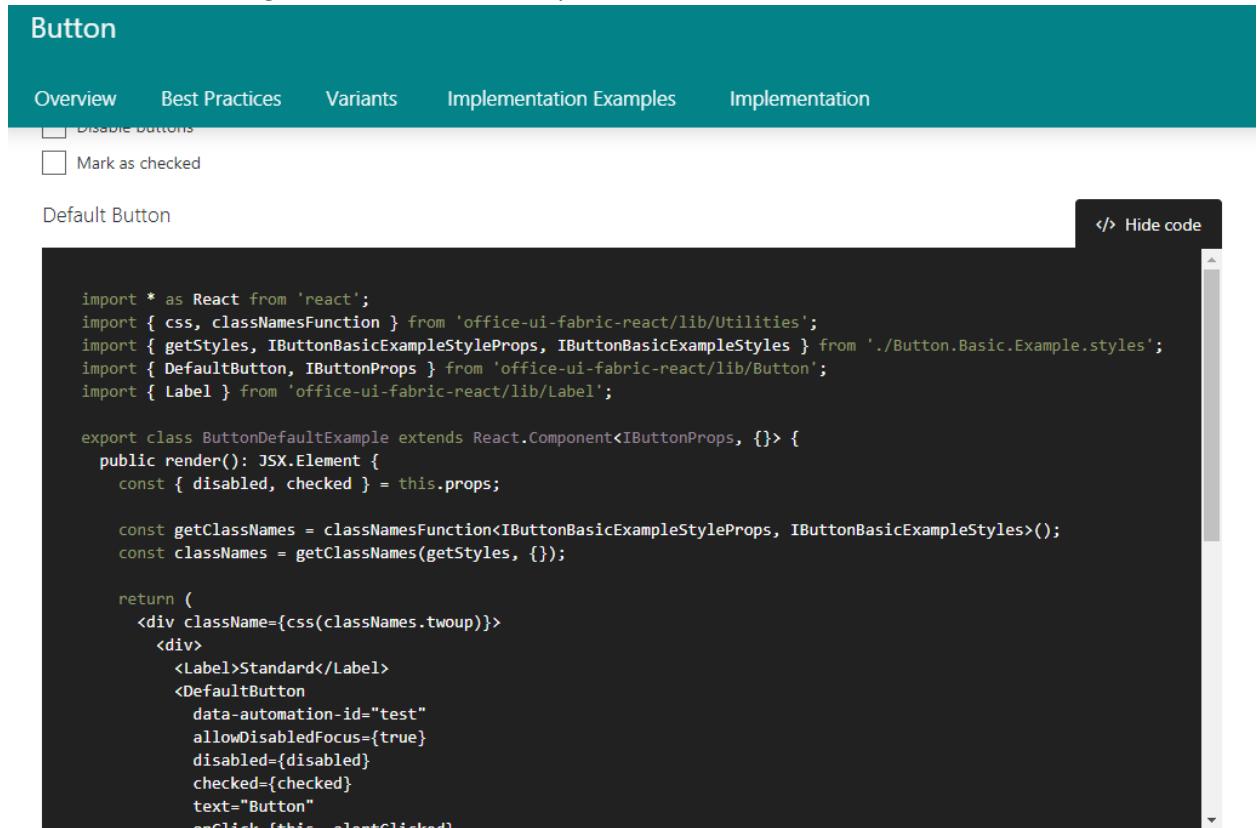
Buttons are best used to enable a user to commit a change or complete steps in a task. They are typically found inside forms, dialogs, panels or pages. An example of their usage is confirming the deletion of a file in a confirmation dialog.

5. Alternatively, search your component from search box
6. Click **Variants** tab to see the variations for use of the component



The screenshot shows the 'Button' component page in the SharePoint Framework UI. The left sidebar lists various components under 'Basic Inputs'. The 'Variants' tab is selected and highlighted with a red box. On the right, the 'Variants' section displays two examples: 'Default Button' and 'Compound Button', each with 'Standard' and 'Primary' variants. A 'Show code' button is located at the bottom right of the variants section, also highlighted with a red box.

7. Click "Show code" to get the code for the component



The screenshot shows the 'Show code' panel for the 'Button' component. The panel contains the following code:

```

import * as React from 'react';
import { css, classNamesFunction } from 'office-ui-fabric-react/lib/Utilities';
import { getStyles, IButtonBasicExampleStyleProps, IButtonBasicExampleStyles } from './Button.Basic.Example.styles';
import { DefaultButton, IButtonProps } from 'office-ui-fabric-react/lib/Button';
import { Label } from 'office-ui-fabric-react/lib/Label';

export class ButtonDefaultExample extends React.Component<IButtonProps, {}> {
  public render(): JSX.Element {
    const { disabled, checked } = this.props;

    const getClassNames = classNamesFunction<IButtonBasicExampleStyleProps, IButtonBasicExampleStyles>();
    const classNames = getClassNames(getStyles, {});

    return (
      <div className={css(classNames.twoup)}>
        <div>
          <Label>Standard</Label>
          <DefaultButton
            data-automation-id="test"
            allowDisabledFocus={true}
            disabled={disabled}
            checked={checked}
            text="Button"
            onClick={this._onClick}>
        </DefaultButton>
      </div>
    );
  }
}

class ButtonDefaultExampleTest extends React.Component<{}, {}> {
  _onClick() {
    alert('button clicked');
  }
}

```

8. Copy entire code or part of it to use in your SPFx solution

Implement Greet Message WebPart using Office UI Fabric

Using the Office UI Fabric components, we will implement simple webpart, having below components

Textbox – Accepts user name

Button – with text as “Greet” on click of which will show an alert greeting the text typed in textbox

1. In the solution, add file IComponentState.ts, which will represent the state of entered user name

```
export interface IComponentState {
  userName: string;
}
```

2. open OfficeUiFabricExamples.tsx under “\src\webparts\officeUiFabricExamples\components\” folder
3. Import Textfield and button components

```
// Import Textfield component
import { TextField } from 'office-ui-fabric-react/lib/TextField';

// Import Button component
import { IButtonProps, DefaultButton } from 'office-ui-fabric-
react/lib/Button';
```

4. Add components to Render method

```
export default class OfficeUiFabricExamples extends
React.Component<IOfficeUiFabricExamplesProps, IComponentState> {
  constructor(props: IOfficeUiFabricExamplesProps, state:
IComponentState) {
  super(props);

  this.state = ({
    userName: ''
  });

  this._greetClicked = this._greetClicked.bind(this);
}

public render(): React.ReactElement<IOfficeUiFabricExamplesProps> {
  return (
    <div className={ styles.officeUiFabricExamples }>
      <div className={ styles.container }>
        <div className={ styles.row }>
          <div className={ styles.column }>
```

```
<div className="docs-TextFieldExample">
  <TextField
    required={true}
    name="txtUserName"
    placeholder="Your name please!"
    value={this.state.userName}
    onChanged={e => this.setState({ userName: e })}>
  />

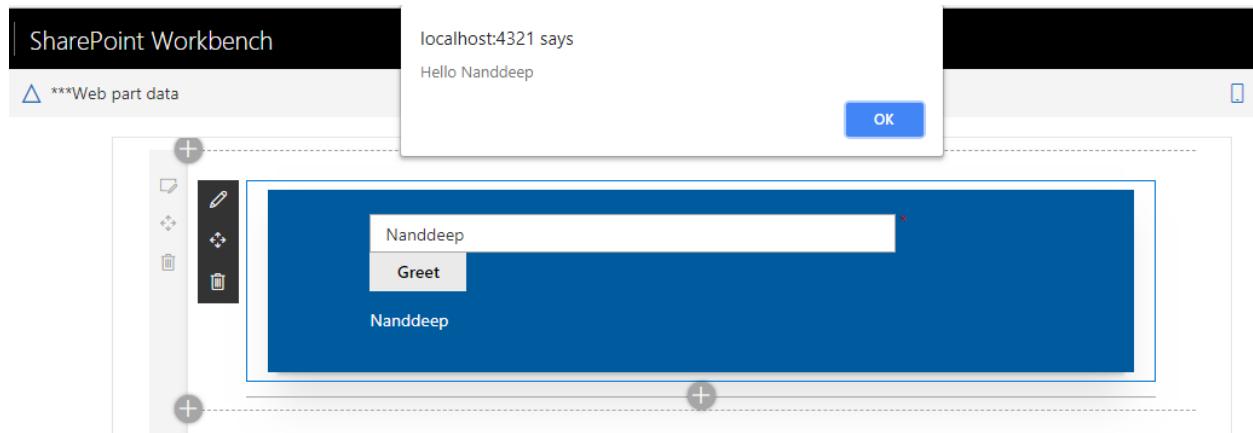
  <DefaultButton
    data-automation-id="greet"
    target="_blank"
    title="Greet the user!"
    onClick={this._greetClicked}>
  >
    Greet
  </DefaultButton>

  <p>{this.state.userName}</p>
</div>
</div>
</div>
</div>
</div>
);
}

private _greetClicked(): void {
  alert('Hello ' + this.state.userName);
}
}
```

Test the WebPart

1. On the command prompt, type “gulp serve”
2. SharePoint local workbench will open
3. Add the webpart to page
4. Enter text in TextBox component, click Greet button. An alert should appear greeting the value entered in textbox



Summary

Office UI Fabric React components helps to develop robust and consistent design across SharePoint modern experiences. Office UI Fabric is available out of the box for developers in SharePoint Framework.

Chapter 16: Provision SharePoint Assets

Overview

SharePoint components like web parts normally interact with the SharePoint assets like lists, libraries, content types, etc. In the full trust solutions, we deploy these SharePoint assets along with web part in a single deployable package (.wsp).

Similarly, SharePoint Framework (SPFx) client side web parts do interact with the underlying lists and libraries in SharePoint site. These SharePoint assets needs to be provisioned along with the client side solution package. SharePoint framework toolchain allows to package and deploy SharePoint assets with client side solution package.

In this chapter, we will explore how we can package together the SharePoint assets needed for SPFx web parts.

SharePoint Assets

SharePoint assets is a generic term referred to basic building blocks of SharePoint that constitutes fields, content types, list instances.

Fields (Site Columns):

A field is metadata that describes property or attribute of object we want to represent. For example, an Employee can be represented with ID, Name, Department, etc. Each field has a specific type such as text, number, Boolean, etc.

Below is an example of field representing number

```
<Field ID="{060E50AC-E9C1-4D3C-B1F9-DE0BCAC300F6}"
      Name="SPFxAmount"
      DisplayName="Amount"
      Type="Currency"
      Decimals="2"
      Min="0"
      Required="FALSE"
      Group="SPFx Columns" />
```

Content types

Content type is reusable collection of site columns.

Below is an example of content type that uses site column.

```
<ContentType ID="0x010042D0C1C200A14B6887742B6344675C8B"
             Name="Cost Center"
             Group="SPFx Content Types"
             Description="Content types from SPFx">
    <FieldRefs>
```

```
<FieldRef ID="{060E50AC-E9C1-4D3C-B1F9-DE0BCAC300F6}" />
</FieldRefs>
</ContentType>
```

List instance

List instance is pre-defined SharePoint list with well-known identifier. We can add, update, and delete items from list.

```
<ListInstance
  FeatureId="00bfea71-de22-43b2-a848-c05709900100"
  Title="SPFx List"
  Description="SPFx List"
  TemplateType="100"
  Url="Lists/SPFxList">
</ListInstance>
```

List instance with custom schema

We can create our own schema to define fields, content types, and views for list instance. Use CustomSchema attribute in ListInstance element to reference custom schema.

```
<ListInstance
  CustomSchema="schema.xml"
  FeatureId="00bfea71-de22-43b2-a848-c05709900100"
  Title="SPFx List"
  Description="SPFx List"
  TemplateType="100"
  Url="Lists/SPFxList">
</ListInstance>
```

Create SPFx Solution

1. Open command prompt. Create a directory for SPFx solution

```
md spfx-provisionspassets
```

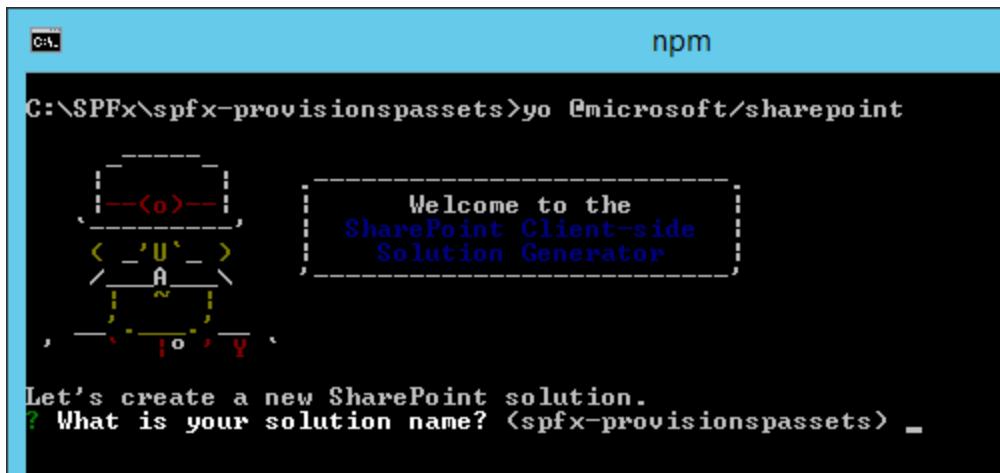
2. Navigate to above created directory

```
cd spfx-provisionspassets
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



```
C:\> npm
C:\> C:\SPFx\spfx-provisionspassets>yo @microsoft/sharepoint
Welcome to the
SharePoint Client-side
Solution Generator
Let's create a new SharePoint solution.
? What is your solution name? <spfx-provisionspassets> _
```

Solution Name: Hit enter to have default name (spfx-provisionspassets in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension. Choose webpart option.

Selected choice: WebPart

Web part name: Hit enter to select the default name or type in any other name.

Selected choice: ProvisionSPAssets

Web part description: Hit enter to select the default description or type in any other value.

Selected choice: Provision SharePoint Assets with SPFx

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)

Selected choice: No JavaScript Framework

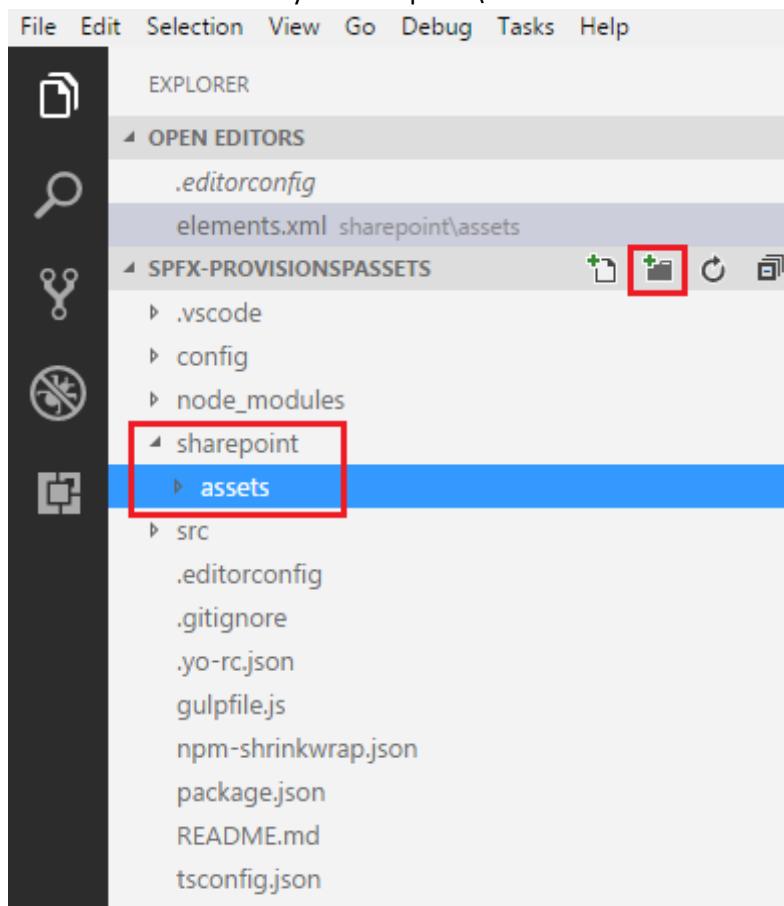
5. Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take significant amount of time.
6. Once the scaffolding process is completed, lock down the version of project dependencies by running below command

npm shrinkwrap
7. In the command prompt type below command to open the solution in code editor of your choice.

code .

Add SharePoint Assets to Solution

1. Create a folder hierarchy as sharepoint\assets



2. Add a file elements.xml under sharepoint\assets folder

```

<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

<Field ID="{060E50AC-E9C1-4D3C-B1F9-DE0BCAC300F6}">
```

```
Name="SPFxAmount"
    DisplayName="Amount"
    Type="Currency"
    Decimals="2"
    Min="0"
    Required="FALSE"
    Group="SPFx Columns" />

<Field ID="{943E7530-5E2B-4C02-8259-CCD93A9ECB18}"
    Name="SPFxCostCenter"
    DisplayName="Cost Center"
    Type="Choice"
    Required="FALSE"
    Group="SPFx Columns">
    <CHOICES>
        <CHOICE>Administration</CHOICE>
        <CHOICE>Information</CHOICE>
        <CHOICE>Facilities</CHOICE>
        <CHOICE>Operations</CHOICE>
        <CHOICE>Sales</CHOICE>
        <CHOICE>Marketing</CHOICE>
    </CHOICES>
</Field>

<ContentType ID="0x010042D0C1C200A14B6887742B6344675C8B"
    Name="Cost Center"
    Group="SPFx Content Types"
    Description="Sample content types from web part solution">
    <FieldRefs>
        <FieldRef ID="{060E50AC-E9C1-4D3C-B1F9-DE0BCAC300F6}" />
        <FieldRef ID="{943E7530-5E2B-4C02-8259-CCD93A9ECB18}" />
    </FieldRefs>
</ContentType>

<ListInstance
    CustomSchema="schema.xml"
    FeatureId="00bfea71-de22-43b2-a848-c05709900100"
    Title="SPFx List"
    Description="SPFx List"
    TemplateType="100"
    Url="Lists/SPFxList">
</ListInstance>

</Elements>
```

We are provisioning 2 fields, content type and list instance with custom schema. FeatureId in the ListInstance represents the ID of feature which contains list definition. The featureId mentioned in the xml represents the ID for custom list definition.

Custom Schema

Add schema.xml file as our custom schema

```

<List xmlns:ows="Microsoft SharePoint" Title="Basic List"
EnableContentTypes="TRUE" FolderCreation="FALSE"
Direction="$Resources:Direction;" Url="Lists/Basic List" BaseType="0"
xmlns="http://schemas.microsoft.com/sharepoint/">
<MetaData>
  <ContentTypes>
    <ContentTypeRef ID="0x010042D0C1C200A14B6887742B6344675C8B" />
  </ContentTypes>
  <Fields></Fields>
  <Views>
    <View BaseViewID="1" Type="HTML" WebPartZoneID="Main"
DisplayName="$Resources:core,objectiv_schema_mwsidcamlidC24;" DefaultView="TRUE" MobileView="TRUE" MobileDefaultView="TRUE"
SetupPath="pages\viewpage.aspx" ImageUrl="/_layouts/images/generic.png"
Url="AllItems.aspx">
      <XslLink Default="TRUE">main.xsl</XslLink>
      <JSLink>clienttemplates.js</JSLink>
      <RowLimit Paged="TRUE">30</RowLimit>
      <Toolbar Type="Standard" />
      <ViewFields>
        <FieldRef Name="LinkTitle"></FieldRef>
        <FieldRef Name="SPFxAmount"></FieldRef>
        <FieldRef Name="SPFxCostCenter"></FieldRef>
      </ViewFields>
      <Query>
        <OrderBy>
          <FieldRef Name="ID" />
        </OrderBy>
      </Query>
    </View>
  </Views>
  <Forms>
    <Form Type="DisplayForm" Url="DispForm.aspx"
SetupPath="pages\form.aspx" WebPartZoneID="Main" />
    <Form Type="EditForm" Url="EditForm.aspx"
SetupPath="pages\form.aspx" WebPartZoneID="Main" />
  </Forms>
</List>
```

```

<Form Type="NewForm" Url="NewForm.aspx" SetupPath="pages\form.aspx"
WebPartZoneID="Main" />
</Forms>
</MetaData>
</List>

```

Package Assets as part of Solution

We have created the assets and custom schema. We need to package these files as part of solution.

1. Open package-solution.json file under config folder
2. Include feature framework definition for solution package

```
{
  "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/package-solution.schema.json",
  "solution": {
    "name": "spfx-provisionspassets-client-side-solution",
    "id": "ea551656-9f74-4107-b049-e296e475932d",
    "version": "1.0.0.0",
    "includeClientSideAssets": true,
    "features": [
      {
        "title": "asset-deployment-webpart-client-side-solution",
        "description": "asset-deployment-webpart-client-side-solution",
        "id": "523fe887-ced5-4036-b564-8dad5c6c6e24", // Specify unique GUID
        "version": "1.0.0.0",
        "assets": {
          "elementManifests": [
            "elements.xml"
          ],
          "elementFiles": [
            "schema.xml"
          ]
        }
      }
    ],
    "paths": {
      "zippedPackage": "solution/spfx-provisionspassets.sppkg"
    }
  }
}
```

Specify unique GUID for feature ID.

Deploy and Test

1. Package your client-side solution by running below command

```
gulp bundle
```

2. Create solution package by running below command

```
gulp package-solution
```

This command will create package (.sppkg) inside sharepoint/solution folder.

3. Deploy the package to app catalog

Do you trust spfx-provisionspassets-client-side-solution? X

The client-side solution you are about to deploy contains full trust client side code. The components in the solution can, and usually do, run in full trust, and no resource usage restrictions are placed on them.



spfx-provisionspassets-client-side-solution

This client side solution will get content from the following domains:

<https://localhost:4321/>

[Deploy](#)

[Cancel](#)

4. Click Deploy
5. Open SharePoint site, click “Add an app”
6. Install the app

 EDIT LINKS

Site contents ▶ Your Apps

Your Apps

provision ×

Apps You Can Add

- From Your Organization
- Manage Licenses
- Your Requests

SharePoint Store

1 app matches your search Newest Name



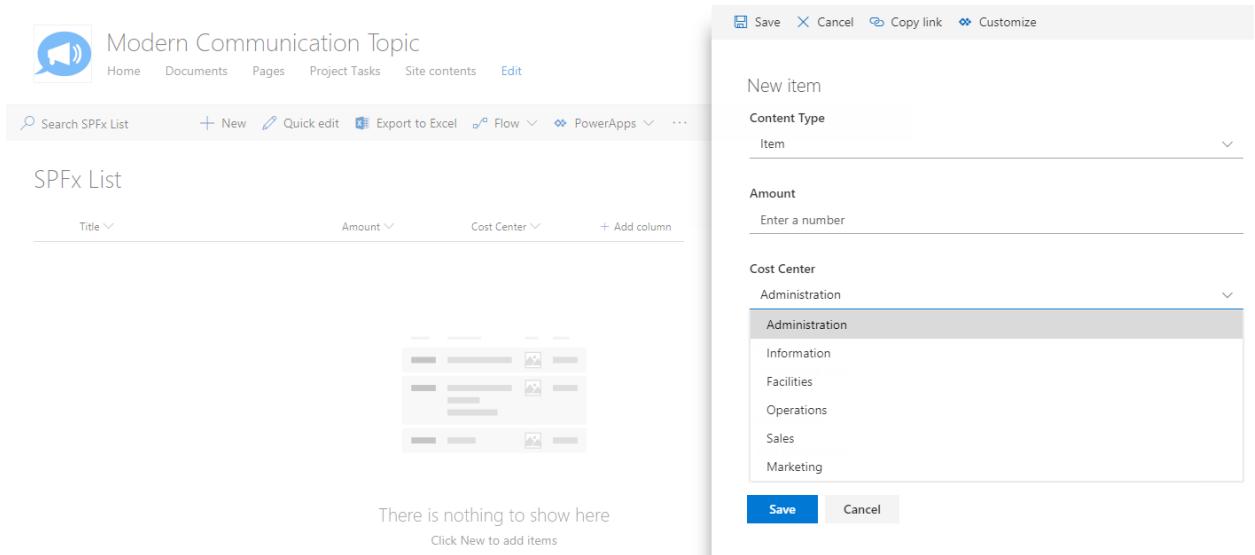
spfx-provisionspassets-client-side-solution

[App Details](#)

7. When installation finishes, refresh the page. The site should have “SPFx List” provisioned.

Contents		Subsites		
	Name	Type	Items	Modified
	Documents	Document library	4	5/8/2018 6:31 AM
	Form Templates	Document library	0	10/25/2017 6:01 AM
	Site Assets	Document library	1	6/3/2018 10:19 AM
	Style Library	Document library	0	10/19/2017 3:03 PM
	SPFx List	List	0	8/20/2018 4:37 AM
	Site Pages	Page library	3	7/27/2018 11:42 PM
	spfx-provisionspassets-clie...	App		8/20/2018 4:36 AM

8. Open “SPFx List”, it should have our content type with site columns inside it.



The screenshot shows a SharePoint Modern Communication Site. On the left, there's a list titled "SPFx List" with columns: Title, Amount, and Cost Center. The "Cost Center" column has a dropdown menu open, showing categories like Administration, Information, Facilities, Operations, Sales, and Marketing. The "Administration" category is selected. On the right, a "New item" dialog is open for a "Content Type" item. It has fields for "Amount" (with placeholder "Enter a number") and "Cost Center" (with "Administration" selected). At the bottom of the dialog are "Save" and "Cancel" buttons.

Summary

SharePoint assets can be provisioned using SPFx. We can define the needed structure to provision on SharePoint site, which can be utilized by SPFx web parts.

Chapter 17: Consume Microsoft Graph API using MSGraphClient

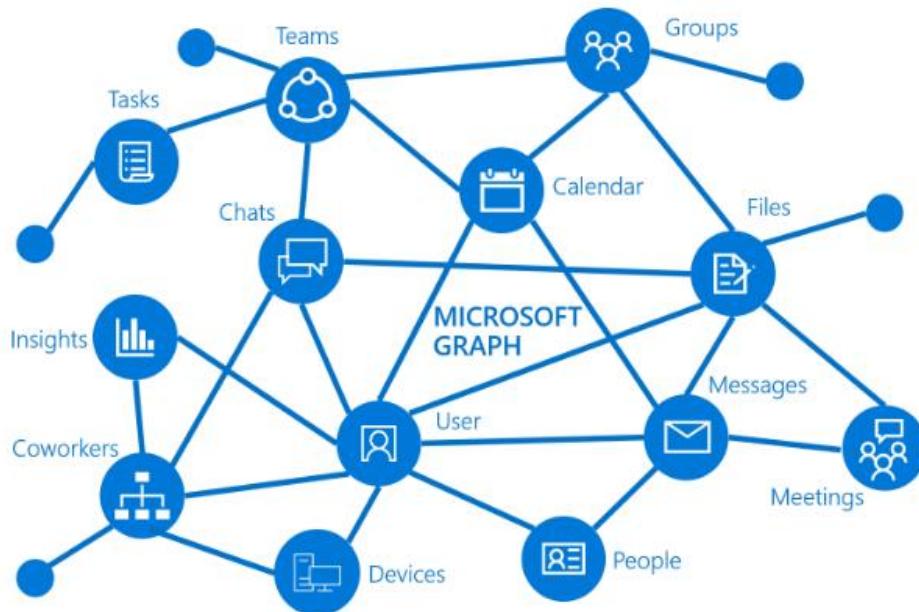
Overview

In the chapters so far, we have explored on developing basic SharePoint client web part which can run independently without any interaction with SharePoint.

In this chapter, we will explore to consume the Microsoft Graph in SharePoint Framework client side web parts.

Brief about Microsoft Graph

MS Graph is a rich and fast growing set of REST APIs provided by Microsoft to access content and services provided by Office 365. For example using Microsoft graph we can access mailbox, calendar, and one drive for business (OD4B) of a user. As well as we can access Site collection, sites, and lists in SharePoint Online. Also, we can access Office 365 Groups, Teams using MS Graph.



Read more about MS Graph at <https://developer.microsoft.com/en-us/graph/docs/concepts/overview>

Create SPFx Solution

1. Open command prompt. Create a directory for SPFx solution

```
md spfx-consume-msgraph
```

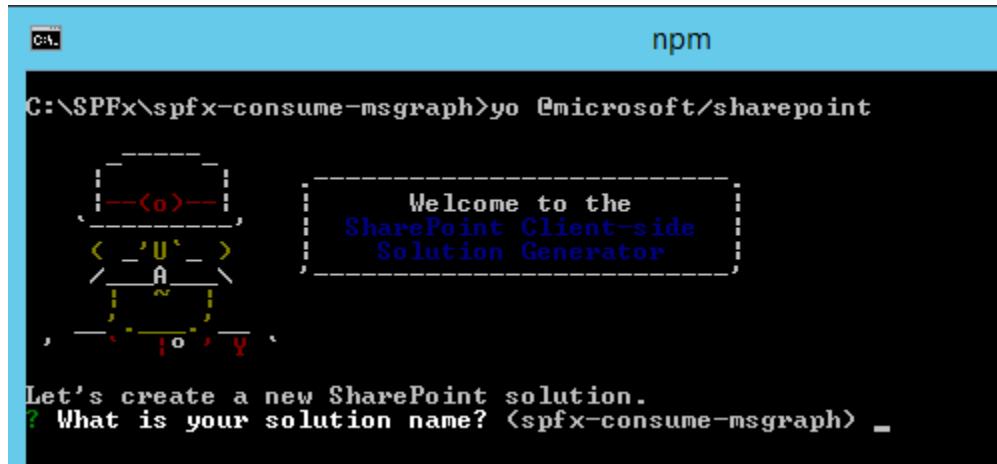
2. Navigate to above created directory

```
cd spfx-consume-msgraph
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



The screenshot shows a terminal window with a blue header bar labeled 'npm'. Below it, the command 'C:\SPFx\spfx-consume-msgraph>yo @microsoft/sharepoint' is typed. A decorative logo consisting of various colored brackets and symbols is displayed. To the right, a box contains the text 'Welcome to the SharePoint Client-side Solution Generator'. At the bottom, the message 'Let's create a new SharePoint solution.' is followed by the question 'What is your solution name? <spfx-consume-msgraph>'. The cursor is positioned at the end of the question.

Solution Name: Hit enter to have default name (spfx-consume-msgraph in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension. Choose webpart option.

Selected choice: WebPart

Web part name: Hit enter to select the default name or type in any other name.

Selected choice: ConsumeMSGraph

Web part description: Hit enter to select the default description or type in any other value.

Selected choice: Consume MS Graph with SPFx

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)

Selected choice: React

5. Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take significant amount of time.
6. Once the scaffolding process is completed, lock down the version of project dependencies by running below command

```
npm shrinkwrap
```

7. In the command prompt type below command to open the solution in code editor of your choice.

```
code .
```

Access MS Graph

Microsoft Graph can be accessed by either native graph client (MSGraphClient) or low-level type used to access Azure AD secured REST API (AadHttpClient)

In the ConsumeMsGraph.tsx file under “\src\webparts\consumeMsGraph\components\” folder, add below import statement

```
import { MSGraphClient } from '@microsoft/sp-client-preview';
```

Typings for MS Graph

Microsoft Graph TypeScript Types enables intellisense on Microsoft Graph objects including users, messages, and groups.

On the command prompt, run below command to include typings

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

This command will install types and save in package.json as a development dependency.

In the ConsumeMsGraph.tsx file under “\src\webparts\consumeMsGraph\components\” folder, add below import statement

```
import * as MicrosoftGraph from '@microsoft/microsoft-graph-types';
```

Permissions

In order to consume MS Graph or any third party REST APIs, we need to explicitly specify the permissions in manifest of solution.

In the package-solution.json file under “config” folder, configure webApiPermissionRequests property to specify User.ReadBasic.All permission.

```
{
  "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/package-solution.schema.json",
  "solution": {
    "name": "spfx-consume-msgraph-client-side-solution",
    "id": "f95e77e7-842b-4ac9-b3f7-f5c421efdb0d",
    "version": "1.0.0.0",
    "includeClientSideAssets": true,
    "webApiPermissionRequests": [
      {
        "resource": "Microsoft Graph",
        "scope": "User.ReadBasic.All"
      }
    ],
    "paths": {
      "zippedPackage": "solution/spfx-consume-msgraph.sppkg"
    }
  }
}
```

webApiPermissionRequests is an array of webApiPermissionRequest items where each item is defined as below:

- resource: name or the ObjectId (in Azure AD). E.g. Microsoft Graph
- scope: name or unique ID of the permission

Please refer the permission API documentation at - https://developer.microsoft.com/en-us/graph/docs/concepts/permissions_reference

Configure Props

Define the context property in IConsumeMsGraphProps.ts under
"\src\webparts\consumeMsGraph\components\"

```
import { WebPartContext } from '@microsoft/sp-webpart-base';

export interface IConsumeMsGraphProps {
  description: string;
  context: WebPartContext;
}
```

Configure State

We will define the state for our React component.

To define interface to represent user, add file IUserItem.ts under “\src\webparts\consumeMsGraph\components\”

```
export interface IUserItem {
    displayName: string;
    mail: string;
    userPrincipalName: string;
}
```

Add file IConsumeMsGraphState.ts under “\src\webparts\consumeMsGraph\components\”

```
import { IUserItem } from './IUserItem';

export interface IConsumeMsGraphState {
    users: Array<IUserItem>;
}
```

Get User Details

Implement below method to get the user details from your tenant.

```
private getUserDetails(): void {

    const graphClient: MSGraphClient = this.props.context.serviceScope.consume(
        MSGraphClient.serviceKey
    );

    graphClient
        .api("users")
        .version("v1.0")
        .select("displayName,mail,userPrincipalName")
        .get((err, res) => {

            if (err) {
                console.error(err);
                return;
            }

            // Prepare the output array
            var users: Array<IUserItem> = new Array<IUserItem>();

            // Map the JSON response to the output array
            res.value.map((item: any) => {
```

```
        users.push( {
            displayName: item.displayName,
            mail: item.mail,
            userPrincipalName: item.userPrincipalName,
        });
    });

// Update the component state accordingly to the result
this.setState(
{
    users: users,
}
);
});
```

Enable Targeted Release on your Tenant

The MS Graph operation is part of an experimental feature and is only available in Targeted release (first release) tenants

1. Open Office 365 admin center
 2. Click Settings > Organization profile
 3. Click Edit against “Release preferences”
 4. Select the preference

The screenshot shows the Microsoft 365 admin center interface. The left sidebar has a dark theme with white icons and labels. The 'Organization profile' item is highlighted with a red box. The main content area shows the 'Organization profile' page with the title 'Home > Organization profile'. It displays details for a user named 'Nachan', including fields for Name, Address, Phone, Technical contact, Preferred language (set to English), and Release preferences. The 'Edit' button for the organization profile and the 'Edit' button for release preferences are both highlighted with red boxes.

Microsoft 365 admin center

Home

Users

Groups

Resources

Billing

Support

Settings

Services & add-ins

Security & privacy

Organization profile

Partner relationships

Home > Organization profile

Nachan

Name

Address

Phone

Technical contact

Preferred language English

Release preferences

Choose how your organization gets new features and service updates from Office 365.

Release track Targeted release for everyone

Release track users All

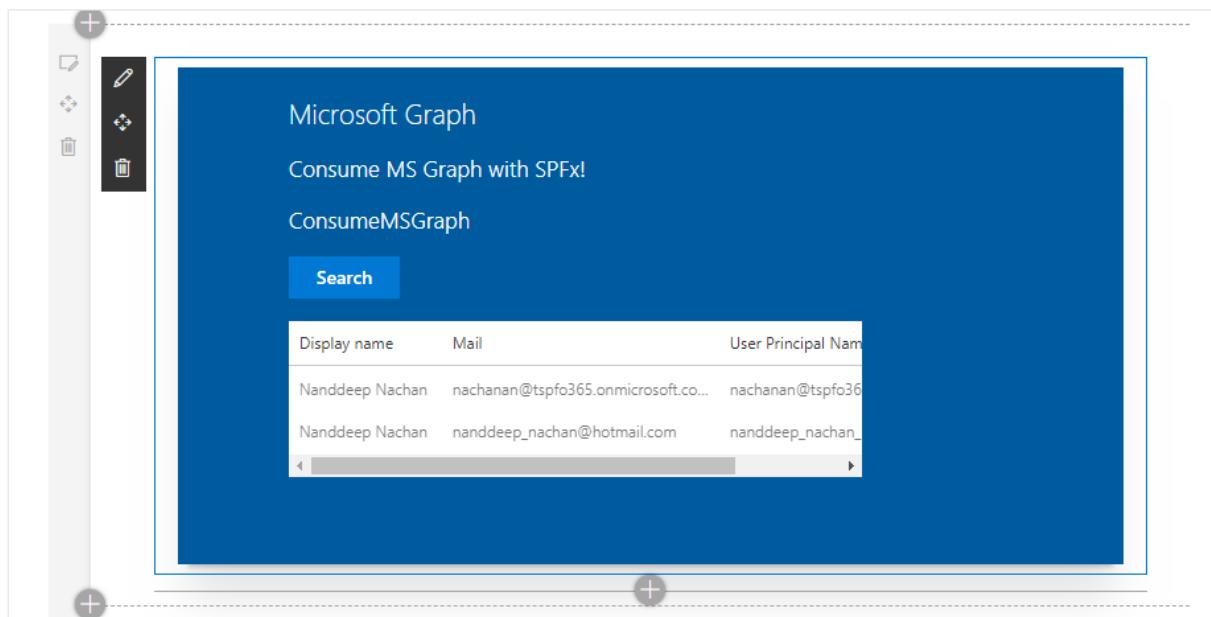
Edit

Edit

Please refer article here (https://support.office.com/en-us/article/set-up-the-standard-or-targeted-release-options-in-office-365-3b3adfa4-1777-4ff0-b606-fb8732101f47#bkmk_setup) to setup your tenant for targeted release.

Test the WebPart

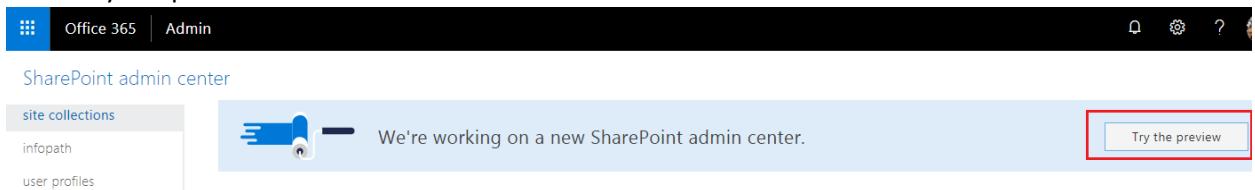
1. On the command prompt, type “gulp serve”
2. Open SharePoint site
3. Navigate to /_layouts/15/workbench.aspx
4. Add the webpart to page.



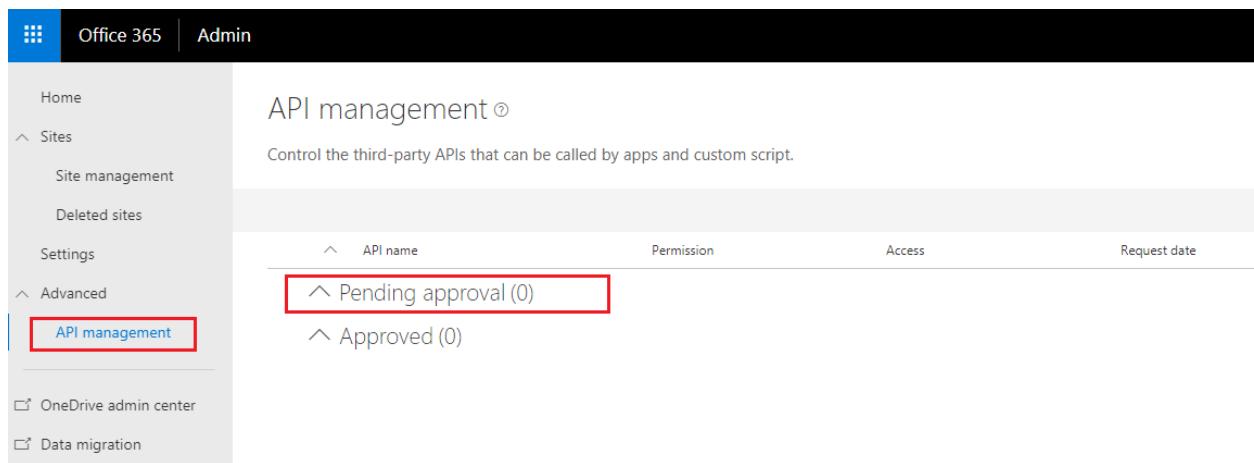
API Management

In the Production environment, after deploying the web part follow below steps to approve API requests.

1. Open SharePoint Admin Center ([https://\[tenant\]-admin.sharepoint.com](https://[tenant]-admin.sharepoint.com))
2. Click “Try the preview”



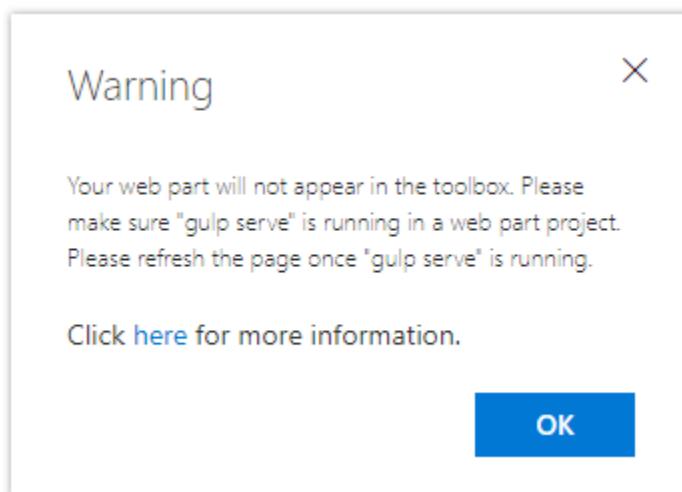
3. From left navigation, Click “API Management”
4. Approve pending requests



The screenshot shows the SharePoint Admin Center interface. In the left sidebar, under 'Advanced', the 'API management' link is selected and highlighted with a red box. The main content area is titled 'API management' and describes controlling third-party APIs. It shows two sections: 'Pending approval (0)' (highlighted with a red box) and 'Approved (0)'. A table header with columns 'API name', 'Permission', 'Access', and 'Request date' is visible.

Troubleshooting

In some cases SharePoint workbench ([https://\[tenant\].sharepoint.com/_layouts/15/workbench.aspx](https://[tenant].sharepoint.com/_layouts/15/workbench.aspx)) shows below error although “gulp serve” is running.



Open below url in the next tab of browser. Accept the warning message.

<https://localhost:4321/temp/manifests.js>

Summary

Microsoft Graph offers wide range of REST APIs to access content and services provided by Office 365. The MS Graph operation is part of an experimental feature and is only available in Targeted release (first release) tenants only.

Chapter 18: Consume Microsoft Graph API using AadHttpClient

Overview

In the chapters so far, we have explored on developing basic SharePoint client web part which can run independently without any interaction with SharePoint.

In this chapter, we will explore to consume the Microsoft Graph in SharePoint Framework client side web parts using AadHttpClient.

Create SPFx Solution

1. Open command prompt. Create a directory for SPFx solution

```
md spfx-msgraph-aadhttpclient
```

2. Navigate to above created directory

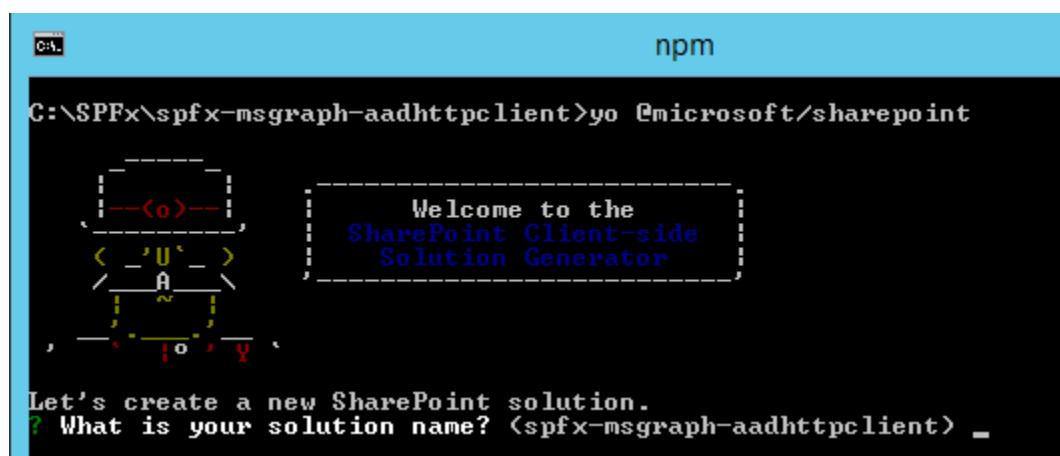
```
cd spfx-msgraph-aadhttpclient
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint --plusbeta
```

Since AadHttpClient is in beta, use --plusbeta with Yeoman generator.

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



The screenshot shows a terminal window with the npm logo at the top. The command entered is 'yo @microsoft/sharepoint'. A red box highlights the text 'Welcome to the SharePoint Client-side Solution Generator'. Another red box highlights the text 'Let's create a new SharePoint solution.' followed by the question 'What is your solution name? <spfx-msgraph-aadhttpclient> _'.

Solution Name: Hit enter to have default name (spfx-msgraph-aadhttpclient in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension. Choose webpart option.

Selected choice: WebPart

Web part name: Hit enter to select the default name or type in any other name.

Selected choice: MSGraphAADHttpClient

Web part description: Hit enter to select the default description or type in any other value.

Selected choice: Consume MS Graph with SPFx using AADHttpClient

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)

Selected choice: React

5. Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take significant amount of time.
6. Once the scaffolding process is completed, lock down the version of project dependencies by running below command

```
npm shrinkwrap
```

7. In the command prompt type below command to open the solution in code editor of your choice.

```
code .
```

Access MS Graph using AadHttpClient

Microsoft Graph can be accessed by low-level type used to access Azure AD secured REST API (AadHttpClient). AadHttpClient client object can be used to consume any REST API, whereas MSGraphClient client object can only consume Microsoft Graph.

In the MsGraphAadHttpClient.tsx file under “\src\webparts\msGraphAadHttpClient\components\” folder, add below import statement

```
import { AadHttpClient } from '@microsoft/sp-http';
```

Permissions

In order to consume MS Graph or any third party REST APIs, we need to explicitly specify the permissions in manifest of solution.

In the package-solution.json file under “config” folder, configure webApiPermissionRequests property to specify User.ReadBasic.All permission.

```
{
  "$schema": "https://developer.microsoft.com/json-schemas/spfx-build/package-solution.schema.json",
  "solution": {
    "name": "spfx-msgraph-aadhttpclient-client-side-solution",
    "id": "f4213803-dc4b-42d9-b6c4-96e895ec02fe",
    "version": "1.0.0.0",
    "includeClientSideAssets": true,
    "webApiPermissionRequests": [
      {
        "resource": "Microsoft Graph",
        "scope": "User.ReadBasic.All"
      }
    ],
    "paths": {
      "zippedPackage": "solution/spfx-msgraph-aadhttpclient.sppkg"
    }
}
```

webApiPermissionRequests is an array of webApiPermissionRequest items where each item is defined as below:

- resource: name or the ObjectId (in Azure AD). E.g. Microsoft Graph
- scope: name or unique ID of the permission

Please refer the permission API documentation at - https://developer.microsoft.com/en-us/graph/docs/concepts/permissions_reference

Configure Props

Define the context property in IMsGraphAadHttpClientProps.ts under “\src\webparts\msGraphAadHttpClient\components\”

```
import { WebPartContext } from '@microsoft/sp-webpart-base';
```

```
export interface IMsGraphAadHttpClientProps {
  description: string;
  context: WebPartContext;
}
```

Configure State

We will define the state for our React component.

To define interface to represent user, add file IUserItem.ts under “\src\webparts\msGraphAadHttpClient\components”

```
export interface IUserItem {
  displayName: string;
  mail: string;
  userPrincipalName: string;
}
```

Add file IMsGraphAadHttpClientState.ts under “\src\webparts\msGraphAadHttpClient\components”

```
import { IUserItem } from './IUserItem';

export interface IMsGraphAadHttpClientState {
  users: Array<IUserItem>;
}
```

Get User Details

Implement below method to get the user details from your tenant.

```
private getUserDetails(): void {
  const aadClient: AadHttpClient = new AadHttpClient(
    this.props.context.serviceScope,
    "https://graph.microsoft.com"
  );

  // Get users with givenName, surname, or displayName
  aadClient
    .get(
      `https://graph.microsoft.com/v1.0/users?$select=displayName,mail,userPrincipalName`,
      AadHttpClient.configurations.v1
    )
    .then(response => {
      return response.json();
    })
}
```

```
})
.then(json => {
    // Prepare the output array
    var users: Array<IUserItem> = new Array<IUserItem>();

    // Log the result in the console for testing purposes
    console.log(json);

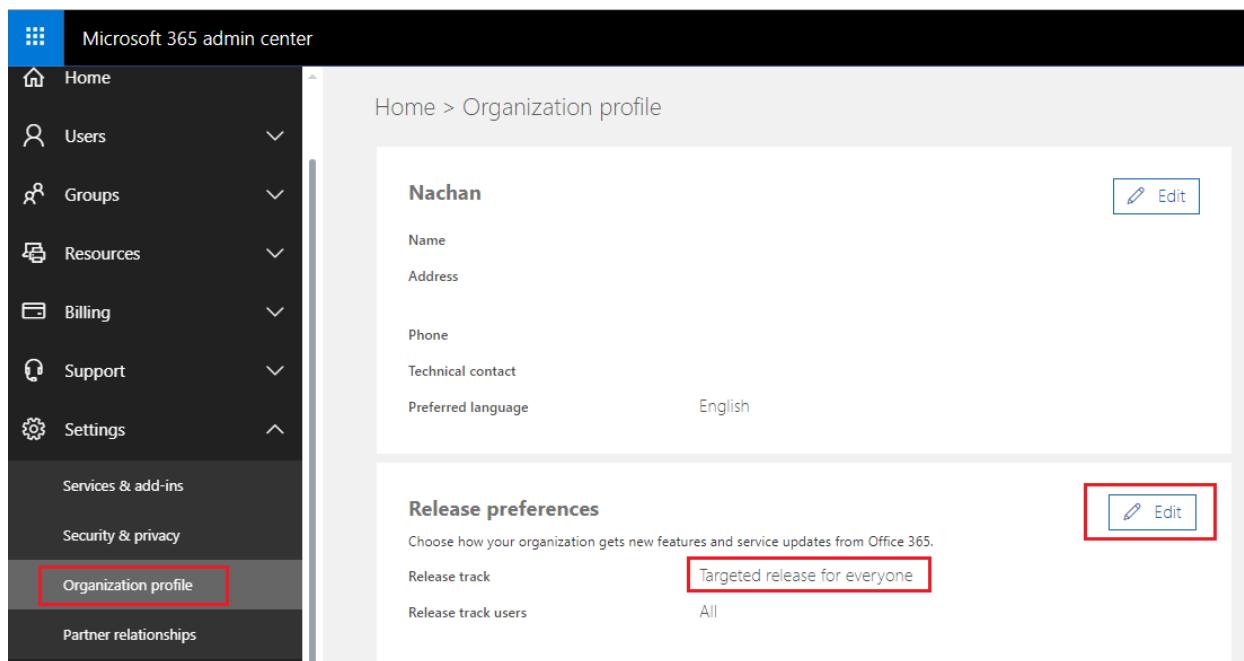
    // Map the JSON response to the output array
    json.value.map((item: any) => {
        users.push( {
            displayName: item.displayName,
            mail: item.mail,
            userPrincipalName: item.userPrincipalName,
        });
    });

    // Update the component state accordingly to the result
    this.setState(
        {
            users: users,
        }
    );
})
.catch(error => {
    console.error(error);
});
}
```

Enable Targeted Release on your Tenant

The MS Graph operation is part of an experimental feature and is only available in Targeted release (first release) tenants

1. Open Office 365 admin center
2. Click Settings > Organization profile
3. Click Edit against “Release preferences”
4. Select the preference

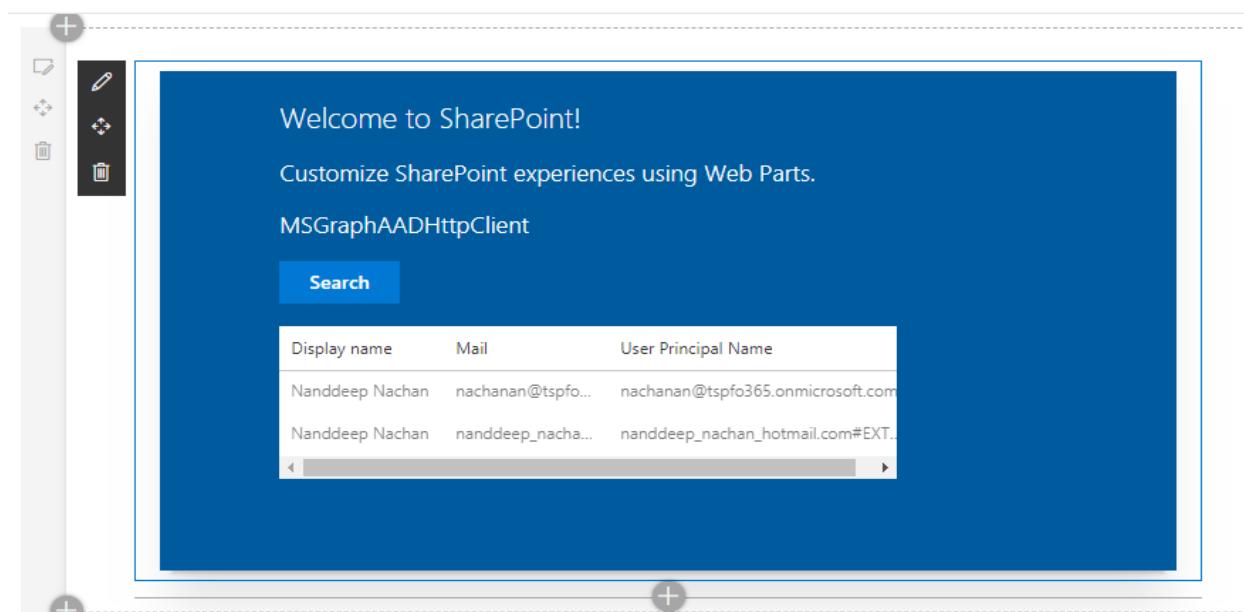


The screenshot shows the Microsoft 365 admin center interface. On the left, there's a navigation sidebar with various categories like Home, Users, Groups, Resources, Billing, Support, Settings, Services & add-ins, Security & privacy, and Organization profile (which is selected and highlighted with a red box). The main content area shows the 'Organization profile' page for 'Nachan'. It includes fields for Name, Address, Phone, Technical contact, Preferred language (set to English), and a 'Release preferences' section. In the 'Release preferences' section, the 'Release track' field is set to 'Targeted release for everyone' (also highlighted with a red box), and the 'Edit' button is also highlighted with a red box.

Please refer article here (https://support.office.com/en-us/article/set-up-the-standard-or-targeted-release-options-in-office-365-3b3adfa4-1777-4ff0-b606-fb8732101f47#bkmk_setup) to setup your tenant for targeted release.

Test the WebPart

1. On the command prompt, type “gulp serve”
2. Open SharePoint site
3. Navigate to /_layouts/15/workbench.aspx
4. Add the webpart to page.



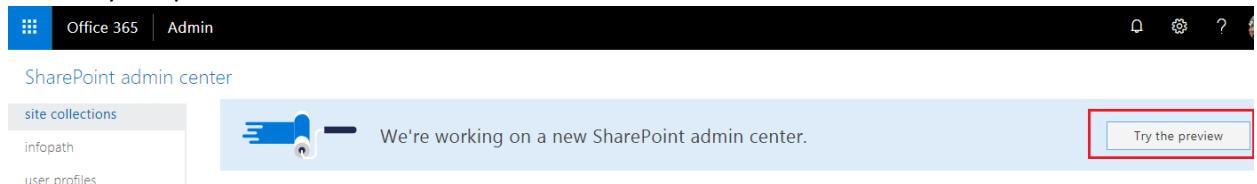
The screenshot shows the SharePoint Workbench page. On the left, there's a ribbon with icons for adding, deleting, and modifying items. The main content area contains a web part titled 'MSGraphAADHttpClient'. The web part displays a table with user information:

Display name	Mail	User Principal Name
Nanddeep Nachan	nachanan@tspfo...	nachanan@tspfo365.onmicrosoft.com
Nanddeep Nachan	nanddeep_nacha...	nanddeep_nachan_hotmail.com#EXT...

API Management

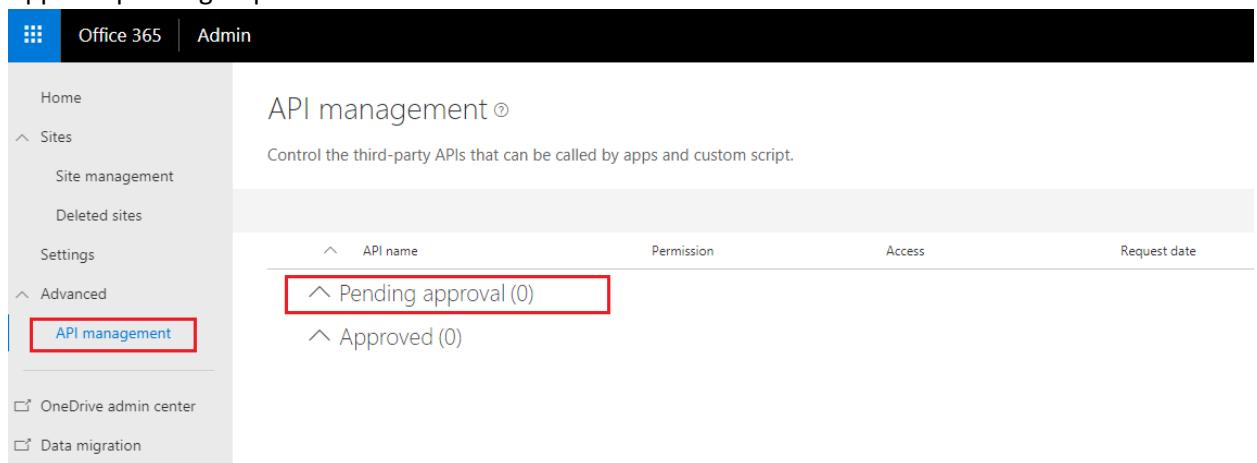
In the Production environment, after deploying the web part follow below steps to approve API requests.

1. Open SharePoint Admin Center ([https://\[tenant\]-admin.sharepoint.com](https://[tenant]-admin.sharepoint.com))
2. Click "Try the preview"



The screenshot shows the SharePoint Admin Center interface. At the top, there's a navigation bar with 'Office 365' and 'Admin'. Below it, a banner says 'We're working on a new SharePoint admin center.' On the right, there's a blue button labeled 'Try the preview' which is highlighted with a red box.

3. From left navigation, Click "API Management"
4. Approve pending requests



The screenshot shows the 'API management' page in the SharePoint Admin Center. The left sidebar has 'API management' selected and highlighted with a red box. The main area shows a table with columns: API name, Permission, Access, and Request date. There are two sections: 'Pending approval (0)' and 'Approved (0)', both of which are highlighted with red boxes.

Summary

Microsoft Graph offers wide range of REST APIs to access content and services provided by Office 365. The MS Graph operation is part of an experimental feature and is only available in Targeted release (first release) tenants only. AadHttpClient is in preview mode, avoid using it in Production.

Chapter 19: Logging

Overview

SharePoint web parts follows their own life cycle while executing the user requests. Everything looks normal till the time all requests goes well and web part performs the user interactions (without any error). The awkward moment comes when any of the user request fails and web part starts displaying error message.

SharePoint by default supports logging. If the verbose logging is enabled, SharePoint tracks each of the activity and obviously it creates loads of logs. In these situations, going through huge logs and finding related information about our error is cumbersome. It is always advisable to have our own logging mechanism implemented.

SharePoint Framework (SPFx) also has no exception to it and supports logging APIs which can be used during the lifecycle of the web part.

Logging Overview

The logging levels are predefined.

Logging Level	Definition
Verbose	Corresponds to lengthy events. Logs almost everything.
Information	Do not require any attention. However they provides valuable data for monitoring state of solution
Warning	Indicates potential problem or issue that might needs attention. We should monitor the pattern over the time. If ignored warning may result in sever error.
Error	Requires urgent attention. All error events should be investigated.
Critical	Indicates serious error that has caused major failure in the solution
None	No logging occurs

Verbose is least important followed by information, warning and error.

Create SPFx Solution

1. Open command prompt. Create a directory for SPFx solution

```
md spfx-logging
```

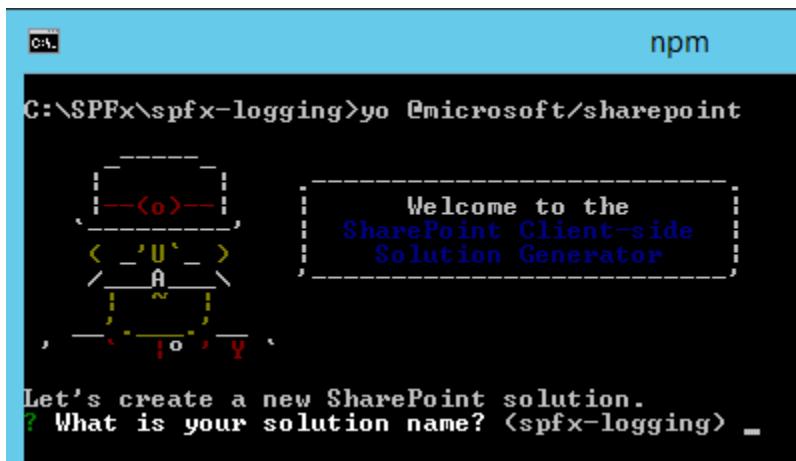
2. Navigate to above created directory

```
cd spfx-logging
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



The screenshot shows a terminal window titled 'npm' with the command 'C:\SPFx\spfx-logging>yo @microsoft/sharepoint'. The terminal displays a decorative logo consisting of various symbols like arrows and letters. To the right of the logo, the text 'Welcome to the SharePoint Client-side Solution Generator' is displayed. Below the logo, the message 'Let's create a new SharePoint solution.' is shown. A question mark prompt '? What is your solution name? <spfx-logging> _' is at the bottom, followed by a cursor.

Solution Name: Hit enter to have default name (spfx-logging in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension. Choose webpart option.

Selected choice: WebPart

Web part name: Hit enter to select the default name or type in any other name.

Selected choice: SPFxLogger

Web part description: Hit enter to select the default description or type in any other value.

Selected choice: Logging with SPFx

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React, and Knockout)

Selected choice: No JavaScript Framework

5. Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take significant amount of time.
6. Once the scaffolding process is completed, lock down the version of project dependencies by running below command

```
npm shrinkwrap
```

7. In the command prompt type below command to open the solution in code editor of your choice.

```
code .
```

Working with Logging API

SharePoint Framework supports logging out of the box. Use below import for logs.

```
import { Log } from '@microsoft/sp-core-library';
```

Define our log source

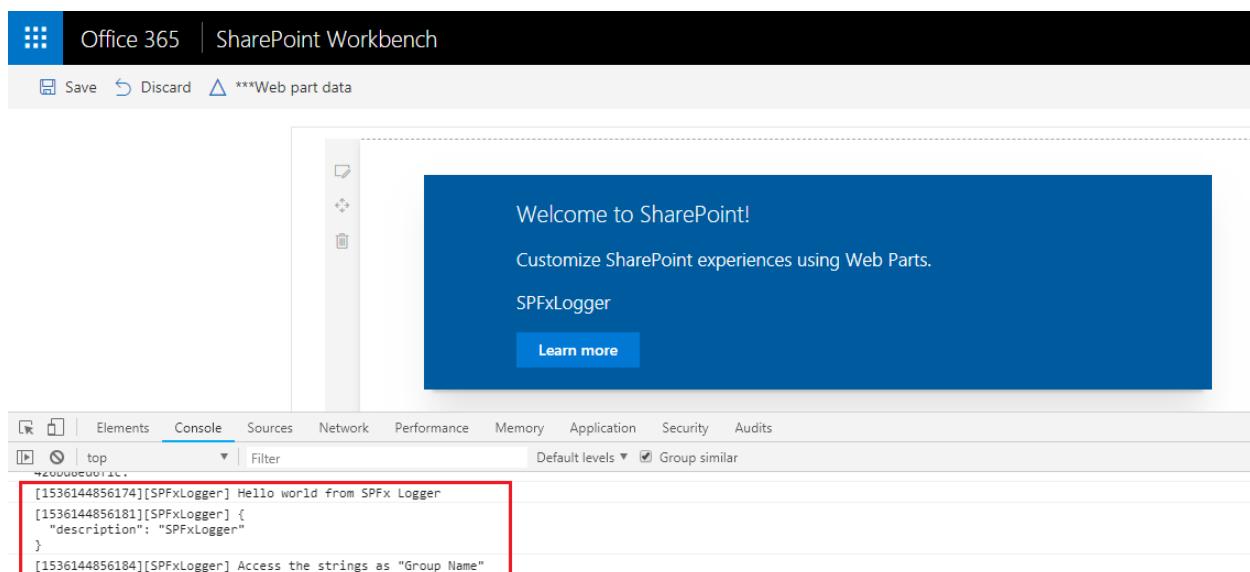
```
const LOG_SOURCE: string = 'SPFxLogger';
```

Log the messages from web part

```
@override
public onInit(): Promise<void> {
  Log.info(LOG_SOURCE, 'Hello world from SPFx Logger');
  Log.info(LOG_SOURCE, JSON.stringify(this.properties, undefined, 2));
  Log.info(LOG_SOURCE, `Access the strings as "${strings.BasicGroupName}"`);
  return Promise.resolve<void>();
}
```

The Log class offers 4 methods to log the information each of which represents corresponding log level – verbose, info, warn, and error. The first parameter is logging location. By default it is name of the web part. But can be overridden. The next parameter is the message to be logged. Optionally we can specify the service scope.

While debugging the SPFx web part, the log message should appear in the console window of browser.



Summary

SharePoint Framework out of the box provides an API to log information in your solutions. Use the logs effectively to get benefited out of it at the time of investigating the cause of error.

Chapter 20: SPFx Extensions Overview

Overview

SharePoint Framework (SPFx) was launched in February 2017 which changed the perspectives of SharePoint developers. Developers then started to build and deploy modern client side web parts using SharePoint Framework across their Office 365 tenants. We will also see this available soon in SharePoint OnPremise version of SP2016.

Earlier, development scenarios included adding Script Editor web parts all over the place on SharePoint site. Troubleshooting the issues to find out which script on page does the trick was very tricky.

SharePoint Framework brought some governance to developers' lives to address these issues.

How will SharePoint Framework Extensions help?

SharePoint Framework Extensions has an ability to expand further SharePoint modern UI which includes site, list, and command extensions, and Graph HttpClient support.

SharePoint Framework includes three new extension types:

Application Customizers

- Allows predefined HTML element placeholders and extend them with custom renderings.
- Used to embed visible or hidden JavaScript on the SharePoint site
- It can be added to Site collection, site or at list scope

Field Customizers

- Allows us to modify views to data for fields within a list.
- Can be used to override field presentation in the list
- Can be used with site columns or directly on the field in list.

Command Sets

- Allows developers to extend command surfaces of SharePoint to add new actions, along with client side code that can be used to implement behaviors.
- Can be used to provide action buttons to list
- Supports toolbar and context menu

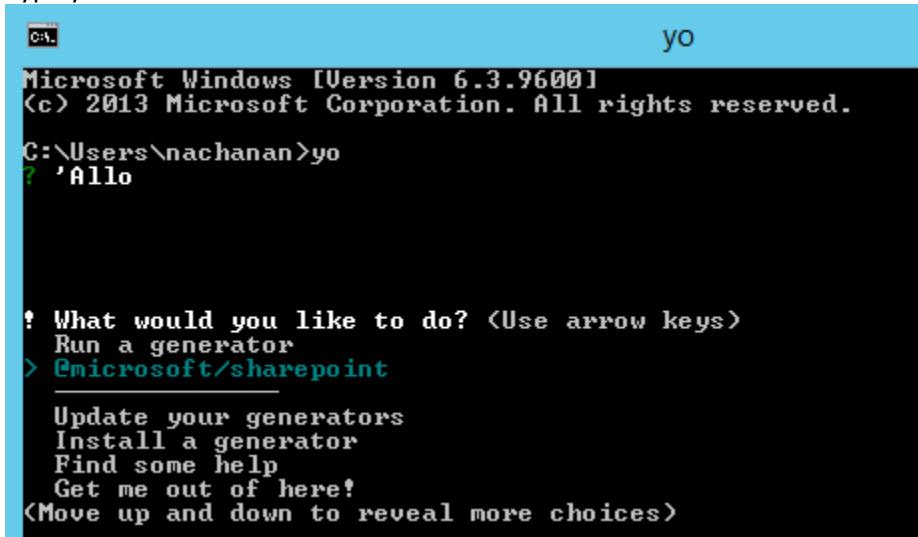
Update Generators for SharePoint Framework Extensions

SharePoint Framework Yeoman Generator gets frequent updates to support the SharePoint Framework Extensions.

If you already have the Yeoman generator installed, you will have to update it by following below instructions:

1. Open command prompt.

2. Type yo.



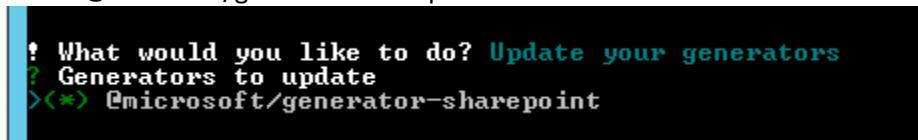
```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\nachanan>yo
? 'Allo

! What would you like to do? <Use arrow keys>
Run a generator
> @microsoft/sharepoint

Update your generators
Install a generator
Find some help
Get me out of here!
<Move up and down to reveal more choices>
```

3. Select option “Update your generators”.
 4. Select @microsoft/generator-sharepoint.



```
! What would you like to do? Update your generators
? Generators to update
><*> @microsoft/generator-sharepoint
```

5. Hit Enter to update the microsoft/generator-sharepoint

Useful commands

1. Update your SharePoint Framework Yeoman generator.

npm update -g @microsoft/generator-sharepoint@latest

You may run this command as you want, if there is no update available – the command will simply return.

2. Check the SharePoint Framework Yeoman generator version

npm view @microsoft/generator-sharepoint version

3. npm list

npm list -g @microsoft/generator-sharepoint

Displays information of installed package on your workstation. Optionally you may use -g parameter to get information on the globally installed package.

```
C:\>npm view @microsoft/generator-sharepoint version
1.1.1

C:\>npm list -g @microsoft/generator-sharepoint
C:\Users\nachanan\AppData\Roaming\npm
`-- @microsoft/generator-sharepoint@1.1.1
```

Create an SharePoint Framework Extensions Project

1. Open Command Prompt
2. Run the below command

`yo @microsoft/sharepoint`

3. When prompted

1. Enter the solution name
2. Choose Extension (Preview) as the client-side component type to create
3. Choose any of below customizer to create
 1. Application Customizer
 2. Field Customizer
 3. ListView Command Set

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

          _-----_
         |       o |
         |   _< U >_ |
         | /   \ A   \ |
         | \   / ~   \ |
          \ /  \ o   / |
           \   \ y   / |

[ Welcome to the
  SharePoint Client-side
  Solution Generator ]
```

Let's create a new SharePoint solution.

? What is your solution name? `HelloWordExtension`

? Which type of client-side component to create? `Extension (Preview)`

? Which type of client-side extension to create? `<Use arrow keys>`

> `Application Customizer (Preview)`

`Field Customizer (Preview)`

`ListView Command Set (Preview)`

4. Enter name of your extension and press Enter
5. Enter description of your extension and press Enter

Yeoman will install the required dependencies and start scaffolding the solution files. This might take a few minutes to generate the project.

Visual Studio Extension for SharePoint Framework

Visual Studio Code is one of the editors for developing SharePoint Framework web parts and SharePoint Framework Extensions.

If you are a Visual Studio enthusiast, SharePoint Patterns and Practices group recently have released a Visual Studio extension.

How does Visual Studio extension work?

The Visual Studio extension is a wrapper around the cmdlets of the Microsoft Yeoman generator (`yo @microsoft/sharepoint`) and presents it in the windows forms experience.

The extension generator project executes the scaffolding behind the scene to create all necessary files.

Once the project creation is successful, developers then can launch the local Workbench by pressing F5.

Summary

SharePoint Framework extensions helps to expand further SharePoint modern UI. Application Customizers, Field Customizers, and Command Sets are the available extension types to help extend the SharePoint site functionality.

Chapter 21: Application Customizer Overview

Overview

SharePoint Framework (SPFx) Extensions allows to extend the SharePoint user experience. Using SharePoint Framework Extensions, we can customize overall SharePoint user experience involving notification area, list views, and toolbars. In this chapter, we will explore the Application Customizer part of SharePoint extensions.

Brief about Application Customizer

Application customizer provides access to predefined locations on SharePoint page and allows to customize them. In this chapter, we will add header and footer to SharePoint site using application customizer.

In the classic SharePoint, we used to extend predefined HTML elements or placeholders from master page to display the custom content. In the modern SharePoint as JavaScript cannot be used on page, the application customizer helps to implement these scenarios.

Page Placeholders

The application customizer helps to get access to below zones on Modern SharePoint page

- Top: Header section of page
- Bottom: Footer section of page

Create SPFx Solution

1. Open command prompt. Create a directory for SPFx solution

```
md spfx-extensions-applicationcustomizer
```

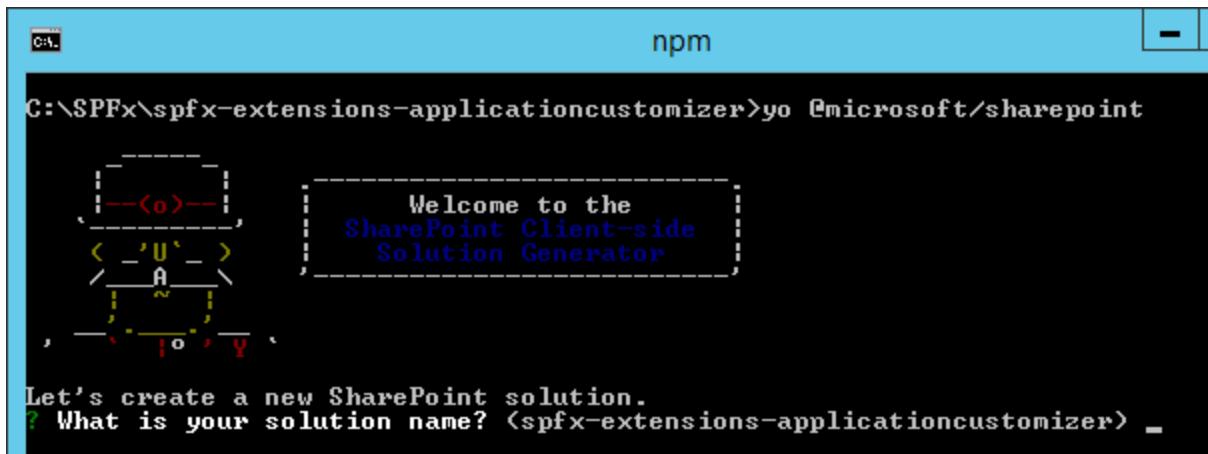
2. Navigate to above created directory

```
cd spfx-extensions-applicationcustomizer
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



```
C:\> npm
C:\> yo @microsoft/sharepoint
Welcome to the
SharePoint Client-side
Solution Generator
Let's create a new SharePoint solution.
? What is your solution name? <spfx-extensions-applicationcustomizer> _
```

Solution Name: Hit enter to have default name (spfx-logging in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension.

Selected choice: Extension

Type of client-side extension to create: We can choose to create Application customizer, Field customizer, or ListView Command Set.

Selected choice: Application customizer

Application customizer name: Hit enter to select the default name or type in any other name.

Selected choice: CustomHeaderFooter

Application customizer description: Hit enter to select the default description or type in any other value.

Selected choice: Adds custom header and footer to SharePoint site

5. Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take significant amount of time.

6. Once the scaffolding process is completed, lock down the version of project dependencies by running below command

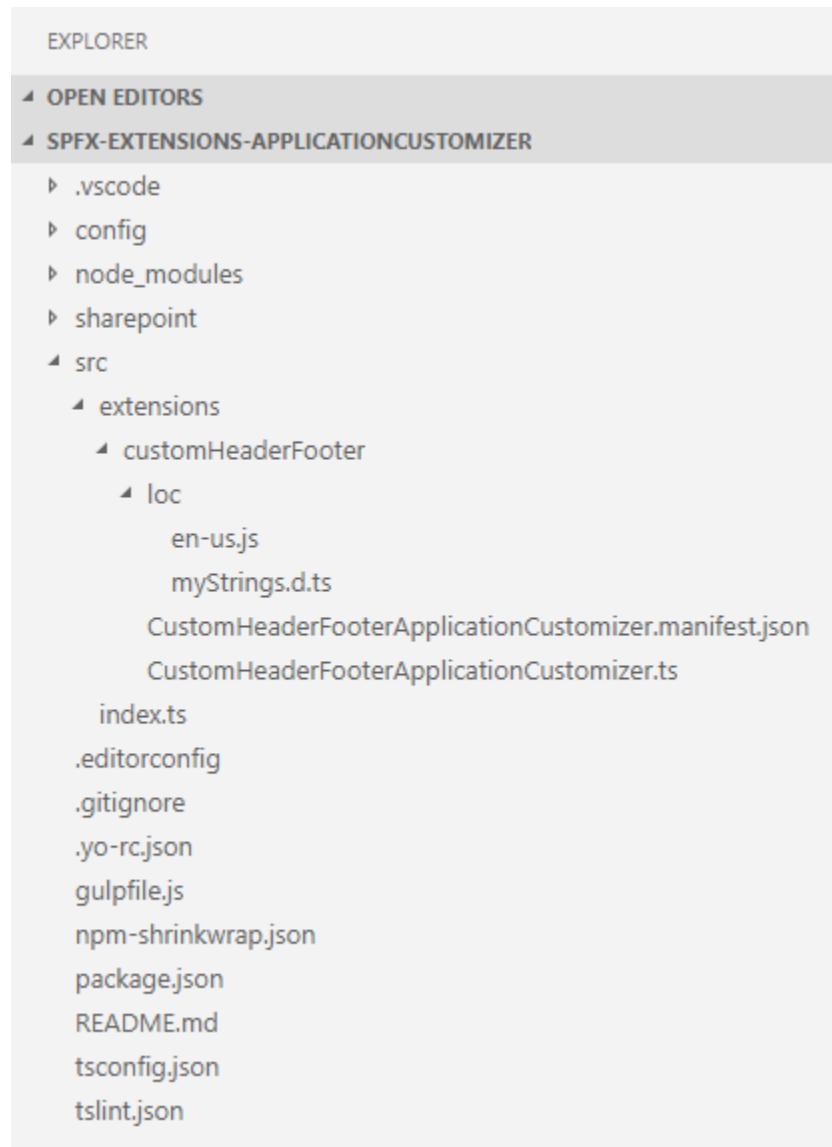
```
npm shrinkwrap
```

7. In the command prompt type below command to open the solution in code editor of your choice.

```
code .
```

Solution Structure

The solution structure is similar to client-side web parts with similar configuration options.



The file `CustomHeaderFooterApplicationCustomizer.manifest.json` inside the folder `"\src\extensions\customHeaderFooter\"` defines the extension type and unique identifier for the solution. Please note down the id. We will need it later for debugging purpose.

```
{
  "$schema": "https://developer.microsoft.com/json-schemas/spfx/client-side-extension-manifest.schema.json",

  "id": "f58a0902-c9d8-4cce-bd5e-4da887e21bed",
  "alias": "CustomHeaderFooterApplicationCustomizer",
  "componentType": "Extension",
  "extensionType": "ApplicationCustomizer",

  // The "*" signifies that the version should be taken from the package.json
  "version": "*",
  "manifestVersion": 2,

  // If true, the component can only be installed on sites where Custom Script
  // is allowed.
  // Components that allow authors to embed arbitrary script code should set
  // this to true.
  // https://support.office.com/en-us/article/Turn-scripting-capabilities-on-
  // or-off-1f2c515f-5d7e-448a-9fd7-835da935584f
  "requiresCustomScript": false
}
```

Implement Application Customizer

Open `CustomHeaderFooterApplicationCustomizer.ts` under `"\src\extensions\customHeaderFooter\"` folder.

To get access to placeholders on the page, use below imports

```
import {
  BaseApplicationCustomizer,
  PlaceholderContent,
  PlaceholderName
} from '@microsoft/sp-application-base';
```

Update the interface `IAlertApplicationCustomizerProperties` to include `Top` and `Bottom` properties

```
export interface ICustomHeaderFooterApplicationCustomizerProperties {
  Top: string;
  Bottom: string;
}
```

Implement OnInit method

```
public OnInit(): Promise<void> {
    Log.info(LOG_SOURCE, `Initialized ${strings.Title}`);

    // Added to handle possible changes on the existence of placeholders.
    this.context.placeholderProvider.changedEvent.add(this,
    this._renderPlaceHolders);

    // Call render method for generating the HTML elements.
    this._renderPlaceHolders();

    return Promise.resolve();
}
```

Implement _renderPlaceHolders method

```
private _renderPlaceHolders(): void {
    console.log('HelloWorldApplicationCustomizer._renderPlaceHolders()');
    console.log('Available placeholders: ',
    this.context.placeholderProvider.placeholderNames.map(name =>
PlaceholderName[name]).join(', '));

    // Handling the top placeholder
    if (!this._topPlaceholder) {
        this._topPlaceholder =
            this.context.placeholderProvider.tryCreateContent(
                PlaceholderName.Top,
                { onDispose: this._onDispose });

        // The extension should not assume that the expected placeholder is
available.
        if (!this._topPlaceholder) {
            console.error('The expected placeholder (Top) was not found.');
            return;
        }

        if (this.properties) {
            let topString: string = this.properties.Top;
            if (!topString) {
                topString = '(Top property was not defined.)';
            }

            if (this._topPlaceholder.domElement) {
                this._topPlaceholder.domElement.innerHTML = `

```

```
<div class="${styles.app}">
    <div class="ms-bgColor-themeDark ms-fontColor-white
${styles.top}">
        <i class="ms-Icon ms-Icon--Info" aria-hidden="true"></i>
${escape(topString)}
        </div>
    </div>`;
}
}

// Handling the bottom placeholder
if (!this._bottomPlaceholder) {
    this._bottomPlaceholder =
        this.context.placeholderProvider.tryCreateContent(
            PlaceholderName.Bottom,
            { onDispose: this._onDispose });

    // The extension should not assume that the expected placeholder is
    // available.
    if (!this._bottomPlaceholder) {
        console.error('The expected placeholder (Bottom) was not found.');
        return;
    }

    if (this.properties) {
        let bottomString: string = this.properties.Bottom;
        if (!bottomString) {
            bottomString = '(Bottom property was not defined.)';
        }

        if (this._bottomPlaceholder.domElement) {
            this._bottomPlaceholder.domElement.innerHTML =
                <div class="${styles.app}">
                    <div class="ms-bgColor-themeDark ms-fontColor-white
${styles.bottom}">
                        <i class="ms-Icon ms-Icon--Info" aria-hidden="true"></i>
${escape(bottomString)}
                        </div>
                    </div>`;
        }
    }
}
```

Use this.context.placeholderProvider.tryCreateContent to get access the placeholder, without assuming that the placeholder will exists

Custom Styles

Add a file CustomHeaderFooterApplicationCustomizer.module.scss under “\src\extensions\customHeaderFooter” folder

```
.app {
  .top {
    height:60px;
    text-align:center;
    line-height:2.5;
    font-weight:bold;
    display: flex;
    align-items: center;
    justify-content: center;
  }

  .bottom {
    height:40px;
    text-align:center;
    line-height:2.5;
    font-weight:bold;
    display: flex;
    align-items: center;
    justify-content: center;
  }
}
```

Include the styles in extension CustomHeaderFooterApplicationCustomizer.ts

```
import styles from './CustomHeaderFooterApplicationCustomizer.module.scss';
```

Test the extension

1. Open serve.json under config folder.
2. Update the properties section to include Top and Bottom messages.

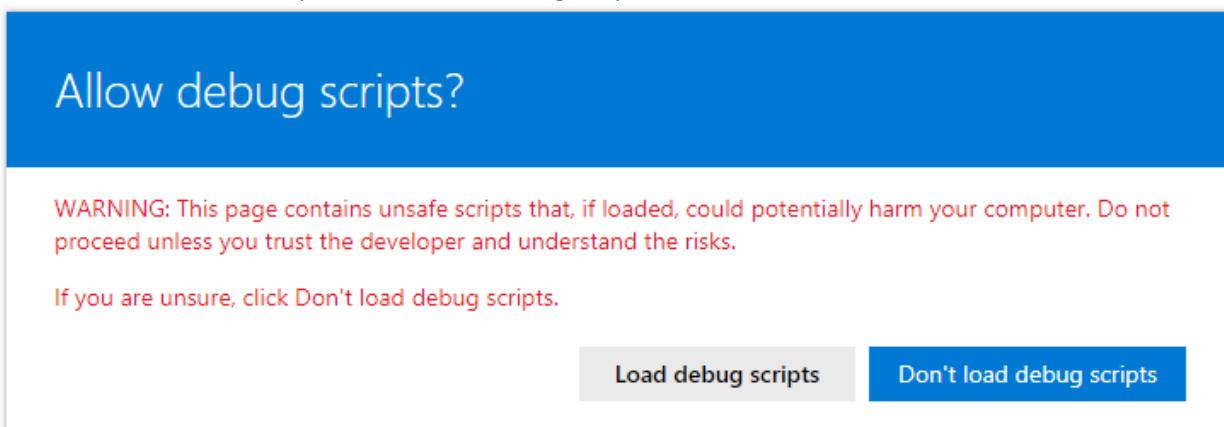
```
{
  "$schema": "https://developer.microsoft.com/json-schemas/core-
build/serve.schema.json",
  "port": 4321,
  "https": true,
  "serveConfigurations": {
    "default": {
```

```
"pageUrl":  
"https://contoso.sharepoint.com/sites/mySite/SitePages/myPage.aspx",  
    "customActions": {  
        "f58a0902-c9d8-4cce-bd5e-4da887e21bed": {  
            "location": "ClientSideExtension.ApplicationCustomizer",  
            "properties": {  
                "Top": "Header of the page",  
                "Bottom": "Footer of the page"  
            }  
        }  
    }  
},  
    "customHeaderFooter": {  
        "pageUrl":  
"https://contoso.sharepoint.com/sites/mySite/SitePages/myPage.aspx",  
        "customActions": {  
            "f58a0902-c9d8-4cce-bd5e-4da887e21bed": {  
                "location": "ClientSideExtension.ApplicationCustomizer",  
                "properties": {  
                    "Top": "Header of the page",  
                    "Bottom": "Footer of the page"  
                }  
            }  
        }  
    }  
}  
}
```

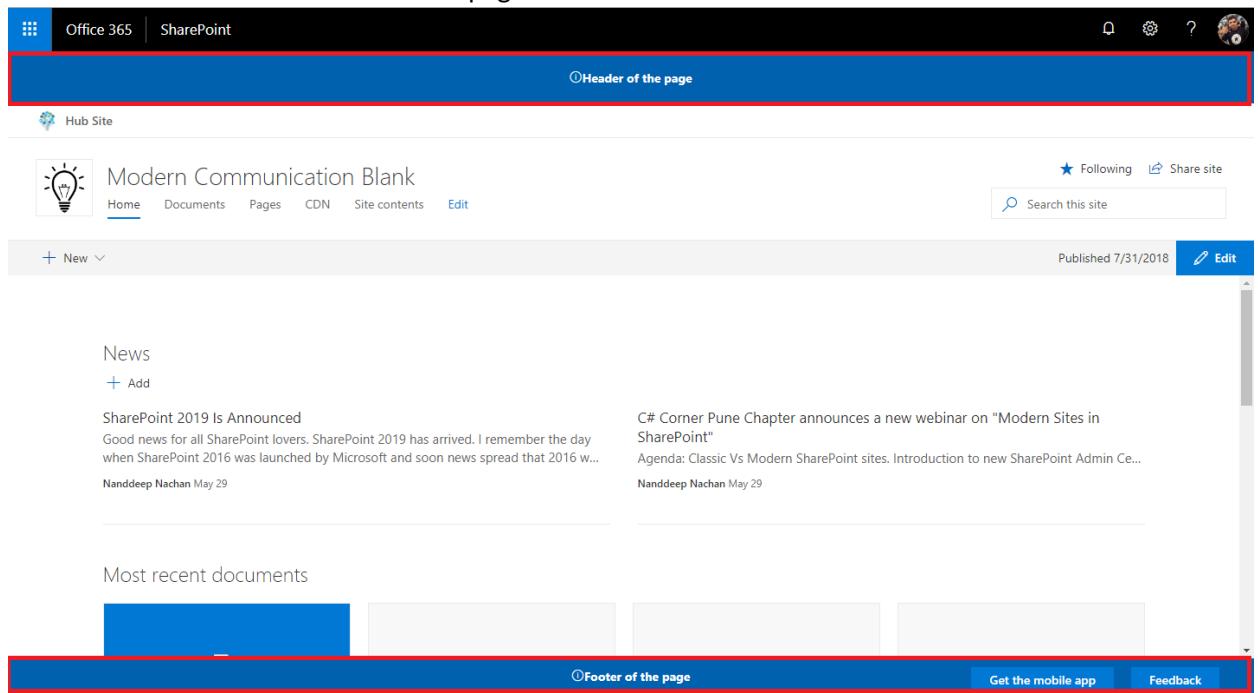
3. In the command prompt, type below command

gulp serve

4. The SharePoint site will open, click “Load debug scripts”



5. Observe the header and footer on the page.



The screenshot shows a SharePoint Modern Communication site. At the top, there's a dark header bar with the Office 365 and SharePoint logos. Below it is a blue header bar labeled "Header of the page". The main content area has a title "Modern Communication Blank" with a lightbulb icon. It includes navigation links for Home, Documents, Pages, CDN, Site contents, and Edit. On the right, there are "Following", "Share site", and a search bar. Below the title, there's a "New" button and a "Published 7/31/2018" timestamp. The main content displays news items and recent documents. At the bottom, there's a blue footer bar labeled "Footer of the page" which contains links for "Get the mobile app" and "Feedback".

Summary

Application customizer SharePoint Framework extension helps to extend the predefined placeholders on Modern SharePoint page. These extensions can be deployed tenant wide.

Chapter 22: Field Customizer Overview

Overview

SharePoint Framework (SPFx) Extensions are client side components which allows extending the SharePoint user experience. In this chapter, we will explore the Field Customizer part of SharePoint extensions.

Brief about Field Customizer

Field customizer allows to modify views for the field in a list view. It can be used to override the field representation in the list. Field customizer can be used with site columns or directly to the field inside a list.

In the classic SharePoint, we used to modify the representation of SharePoint list fields using JSLink property of the web part. In the modern SharePoint as JavaScript cannot be used on page, the field customizer helps to implement these scenarios.

Create SPFx Solution

1. Open command prompt. Create a directory for SPFx solution

```
md spfx-extensions-fieldcustomizer
```

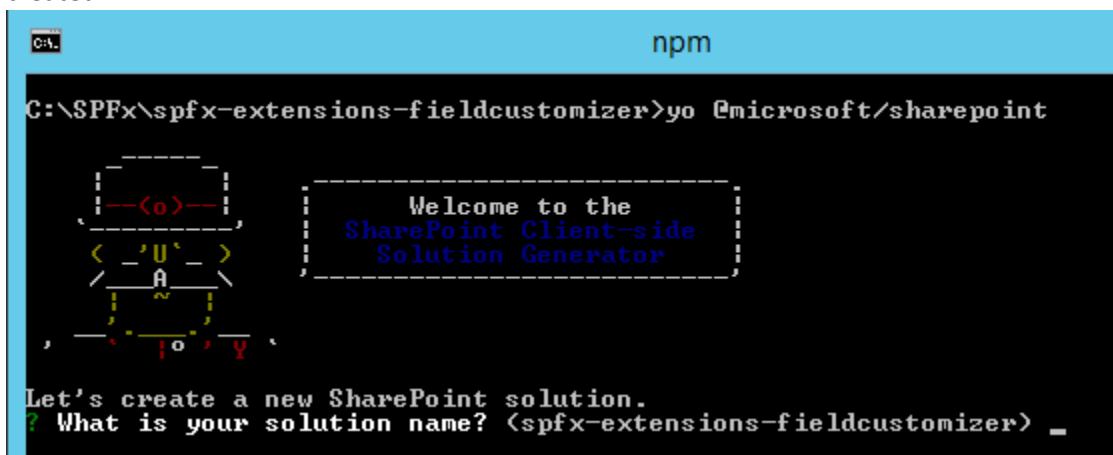
2. Navigate to above created directory

```
cd spfx-extensions-fieldcustomizer
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



```
C:\> npm
C:\> yo @microsoft/sharepoint
Welcome to the
SharePoint Client-side
Solution Generator
Let's create a new SharePoint solution.
? What is your solution name? <spfx-extensions-fieldcustomizer>
```

Solution Name: Hit enter to have default name (spfx-extensions-fieldcustomizer in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension.

Selected choice: Extension

Type of client-side extension to create: We can choose to create Application customizer, Field customizer, or ListView Command Set.

Selected choice: Field customizer

Field customizer name: Hit enter to select the default name or type in any other name.

Selected choice: PercentFieldCustomizer

Field customizer description: Hit enter to select the default description or type in any other value.

Selected choice: Displays field in percentage

Framework to use: Select any JavaScript framework to develop the component. Available choices are (No JavaScript Framework, React)

Selected choice - No JavaScript Framework

5. Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take significant amount of time.

6. Once the scaffolding process is completed, lock down the version of project dependencies by running below command

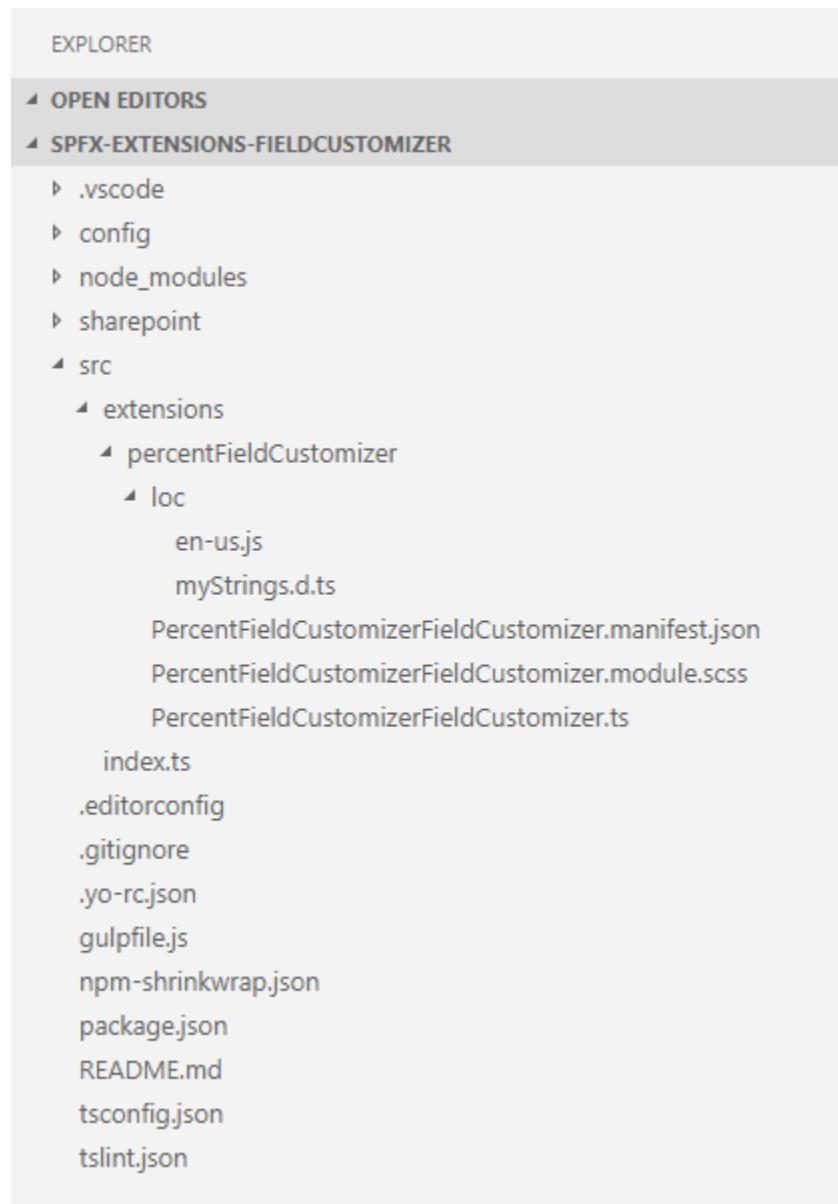
```
npm shrinkwrap
```

7. In the command prompt type below command to open the solution in code editor of your choice.

```
code .
```

Solution Structure

The solution structure is similar to client-side web parts with similar configuration options.



The file PercentFieldCustomizerFieldCustomizer.manifest.json inside the folder "\src\extensions\percentFieldCustomizer" defines the extension type and unique identifier for the solution. Please note down the id. We will need it later for debugging purpose.

```
{
  "$schema": "https://developer.microsoft.com/json-schemas/spfx/client-side-extension-manifest.schema.json",
  "id": "8db272d2-a083-48e7-b6da-dbf14fc98bbf",
  "alias": "PercentFieldCustomizerFieldCustomizer",
```

```

"componentType": "Extension",
"extensionType": "FieldCustomizer",

// The "*" signifies that the version should be taken from the package.json
"version": "*",
"manifestVersion": 2,

// If true, the component can only be installed on sites where Custom Script
is allowed.
// Components that allow authors to embed arbitrary script code should set
this to true.
// https://support.office.com/en-us/article/Turn-scripting-capabilities-on-
or-off-1f2c515f-5d7e-448a-9fd7-835da935584f
"requiresCustomScript": false
}

```

Implement Field Customizer

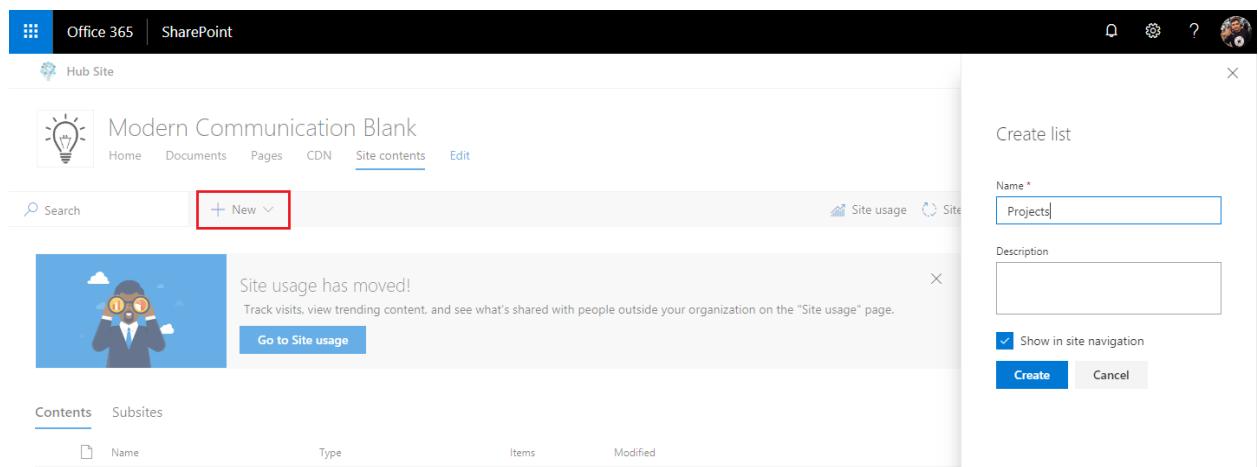
Open PercentFieldCustomizerFieldCustomizer.ts under “\src\extensions\percentFieldCustomizer\” folder. The class has below important methods:

- **onInit()** event occurs before the page DOM is ready. It returns promise to perform asynchronous operations. onRenderCell() is not invoked until this promise is resolved.
- **onRenderCell()** event occurs when each cell is rendered. It provides event.domElement to customize the representation of field.
- **onDisposeCell()** can be used to free up any used resources that were allocated during rendering of the field to avoid any resource leak.

Debug Field Customizer

The SharePoint local workbench now cannot be used to test the field customizer as we will need an actual SharePoint site to create needed list and fields.

1. Open SharePoint site
2. From site contents create new list named “Projects”

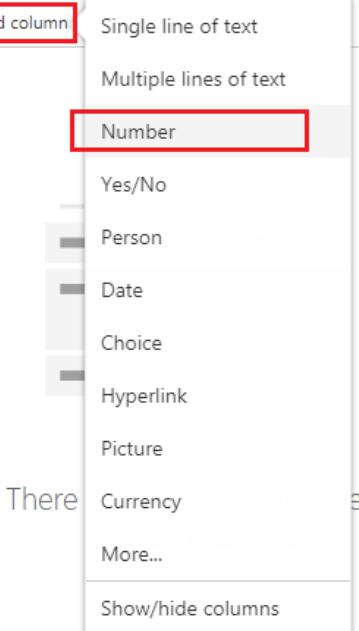


3. Click “Add column” to add a number field

Projects

Title ▾

+ Add column



4. Name it as **Completion**

Create a column

[Learn more about column creation.](#)

Name *

Description

Type

Number

Number of decimal places

Automatic

Show as percentage

Default value

Enter a number

Use calculated value ⓘ

[More options](#)

Save Cancel

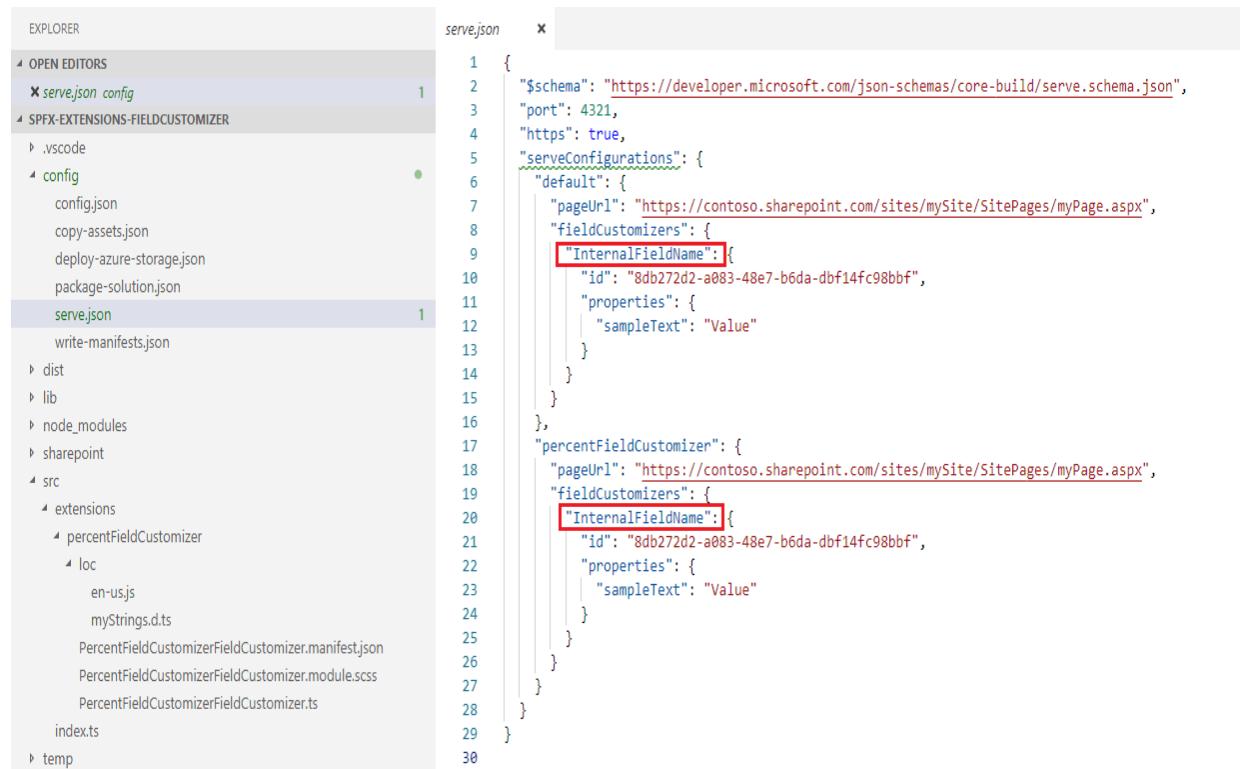
5. Click **Save**
6. Add some test data to the list

Projects

Title	Completion
Cold Fusion	20
Honeycomb	60
Liberation	75
Phoenix	45
Topaz	100
Whistler	56

Update the Solution for Field changes

1. Open serve.json file under config folder. Below is the default content.



```

EXPLORER
OPEN EDITORS
serve.json config
SPFX-EXTENSIONS-FIELDCUSTOMIZER
vscode
config
config.json
copy-assets.json
deploy-azure-storage.json
package-solution.json
serve.json
write-manifests.json
dist
lib
node_modules
sharepoint
src
extensions
percentFieldCustomizer
loc
en-us.js
myStrings.d.ts
PercentFieldCustomizerFieldCustomizer.manifest.json
PercentFieldCustomizerFieldCustomizer.module.scss
PercentFieldCustomizerFieldCustomizer.ts
index.ts
temp

serve.json
  1 {
  2   "$schema": "https://developer.microsoft.com/json-schemas/core-build/serve.schema.json",
  3   "port": 4321,
  4   "https": true,
  5   "serveConfigurations": {
  6     "default": {
  7       "pageUrl": "https://contoso.sharepoint.com/sites/mySite/SitePages/myPage.aspx",
  8       "fieldCustomizers": [
  9         {
  10           "InternalFieldName": {
  11             "id": "8db272d2-a083-48e7-b6da-dbf14fc98bbf",
  12             "properties": {
  13               "sampleText": "Value"
  14             }
  15           }
  16         },
  17         "percentFieldCustomizer": {
  18           "pageUrl": "https://contoso.sharepoint.com/sites/mySite/SitePages/myPage.aspx",
  19           "fieldCustomizers": [
  20             {
  21               "InternalFieldName": {
  22                 "id": "8db272d2-a083-48e7-b6da-dbf14fc98bbf",
  23                 "properties": {
  24                   "sampleText": "Value"
  25                 }
  26               }
  27             }
  28           }
  29         }
  30       }
  
```

2. Update the InternalFieldName attribute with the name of our field, i.e., Completion.

```
{  
  "$schema": "https://developer.microsoft.com/json-schemas/core-  
  build/serve.schema.json",  
  "port": 4321,  
  "https": true,  
  "serveConfigurations": {  
    "default": {  
      "pageUrl":  
        "https://contoso.sharepoint.com/sites/mySite/SitePages/myPage.aspx",  
      "fieldCustomizers": {  
        "Completion": {  
          "id": "8db272d2-a083-48e7-b6da-dbf14fc98bbf",  
          "properties": {  
            "sampleText": "Value"  
          }  
        }  
      }  
    },  
    "percentFieldCustomizer": {  
      "pageUrl":  
        "https://contoso.sharepoint.com/sites/mySite/SitePages/myPage.aspx",  
      "fieldCustomizers": {  
        "Completion": {  
          "id": "8db272d2-a083-48e7-b6da-dbf14fc98bbf",  
          "properties": {  
            "sampleText": "Value"  
          }  
        }  
      }  
    }  
  }  
}
```

3. In the command prompt, type below command to run the extension.

```
gulp serve
```

4. Accept loading of debug manifests by clicking “Load debug scripts”.

Allow debug scripts?

WARNING: This page contains unsafe scripts that, if loaded, could potentially harm your computer. Do not proceed unless you trust the developer and understand the risks.

If you are unsure, click Don't load debug scripts.

[Load debug scripts](#)

[Don't load debug scripts](#)

- The list view should display the formatted Completion column as below.

Projects

Title	Completion
Cold Fusion	Value: 20
Honeycomb	Value: 60
Liberation	Value: 75
Phoenix	Value: 45
Topaz	Value: 100
Whistler	Value: 56

Apply Field Customization

- Navigate back to the solution.
- Open PercentFieldCustomizer.module.scss file from “\src\extensions\percentFieldCustomizer” folder and update the css.

```
.PercentFieldCustomizer {
  .cell {
    background-color: "[theme:themePrimary, default:#e5e5e5]";
    display: 'inline-block';
  }
  .full {
    background-color: #e6e6e6;
    width: 100px;
  }
}
```

```

    }
}

```

3. Open PercentFieldCustomizerFieldCustomizer.ts file from “\src\extensions\percentFieldCustomizer\” folder.
4. Update the onRenderCell method.

```

public onRenderCell(event: IFieldCustomizerCellEventArgs): void {
    // Use this method to perform your custom cell rendering.
    event.domElement.classList.add(styles.cell);
    event.domElement.innerHTML = `
        <div class='${styles.PercentFieldCustomizer}'>
            <div class='${styles.full}'>
                <div style='width: ${event.fieldValue}px;
background:#0094ff; color:#ffffff'>
                    &nbsp; ${event.fieldValue}
                </div>
            </div>
        </div>`;
}

```

5. Make sure the gulp serve is running. Refresh the SharePoint list – Projects.

Projects

Title	Completion
Cold Fusion	20
Honeycomb	60
Liberation	75
Phoenix	45
Topaz	100
Whistler	56

The Completion field is now customized with the progress completion using field customizer.

Summary

The Field customizer SharePoint Framework extension helps to modify the representation of the field in SharePoint list. This is an alternative to JS Link in the modern SharePoint sites.

Chapter 23: ListView Command Set Overview

Overview

SharePoint Framework (SPFx) Extensions are client side components which allows extending the SharePoint user experience. In this chapter, we will explore the ListView Command Set of SharePoint extensions.

Brief about Field Customizer

ListView Command Set allows to extend command surfaces of SharePoint to add new actions. It supports toolbar and context menu.

In the classic SharePoint, we used to add the new actions to SharePoint list by implementing custom actions. In the modern SharePoint, the ListView Command Set helps to implement these scenarios.

Create SPFx Solution

1. Open command prompt. Create a directory for SPFx solution

```
md spfx-extensions-listviewcommandset
```

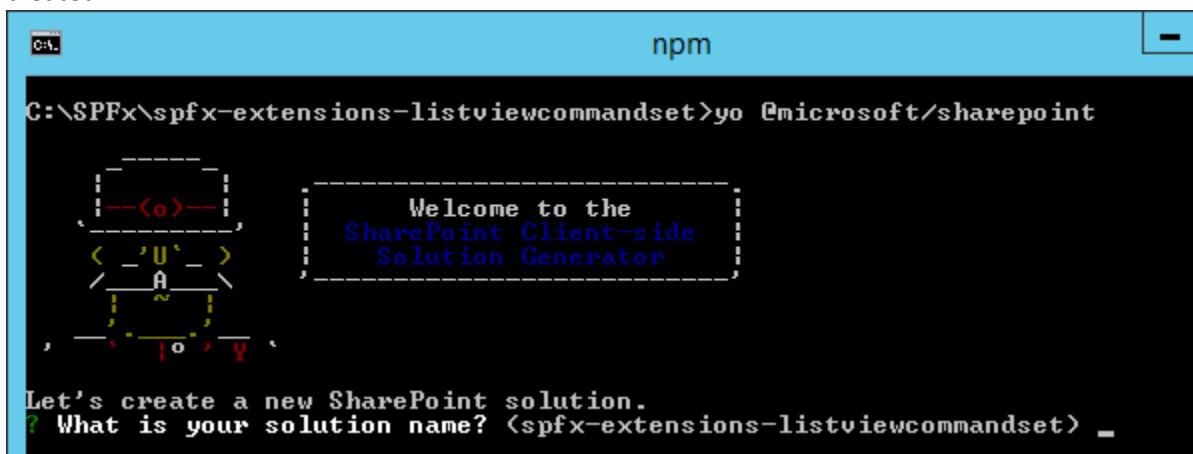
2. Navigate to above created directory

```
cd spfx-extensions-listviewcommandset
```

3. Run Yeoman SharePoint Generator to create the solution

```
yo @microsoft/sharepoint
```

4. Yeoman generator will present you with the wizard by asking questions about the solution to be created.



```
C:\> npm
C:\> yo @microsoft/sharepoint
Welcome to the
SharePoint Client-side
Solution Generator
Let's create a new SharePoint solution.
? What is your solution name? <spfx-extensions-listviewcommandset>
```

Solution Name: Hit enter to have default name (spfx-extensions-listviewcommandset in this case) or type in any other name for your solution.

Selected choice: Hit enter

Target for component: Here we can select the target environment where we are planning to deploy the client webpart i.e. SharePoint Online or SharePoint OnPremise (SharePoint 2016 onwards).

Selected choice: SharePoint Online only (latest)

Place of files: We may choose to use the same folder or create a subfolder for our solution.

Selected choice: Same folder

Deployment option: Selecting Y will allow the app to deployed instantly to all sites and will be accessible everywhere.

Selected choice: N (install on each site explicitly)

Type of client-side component to create: We can choose to create client side webpart or an extension.

Selected choice: Extension

Type of client-side extension to create: We can choose to create Application customizer, Field customizer, or ListView Command Set.

Selected choice: ListView Command Set

Command Set name: Hit enter to select the default name or type in any other name.

Selected choice: ModalDialog

Command Set description: Hit enter to select the default description or type in any other value.

Selected choice: Displays modal dialog using listview command set extension

5. Yeoman generator will perform scaffolding process to generate the solution. The scaffolding process will take significant amount of time.

6. Once the scaffolding process is completed, lock down the version of project dependencies by running below command

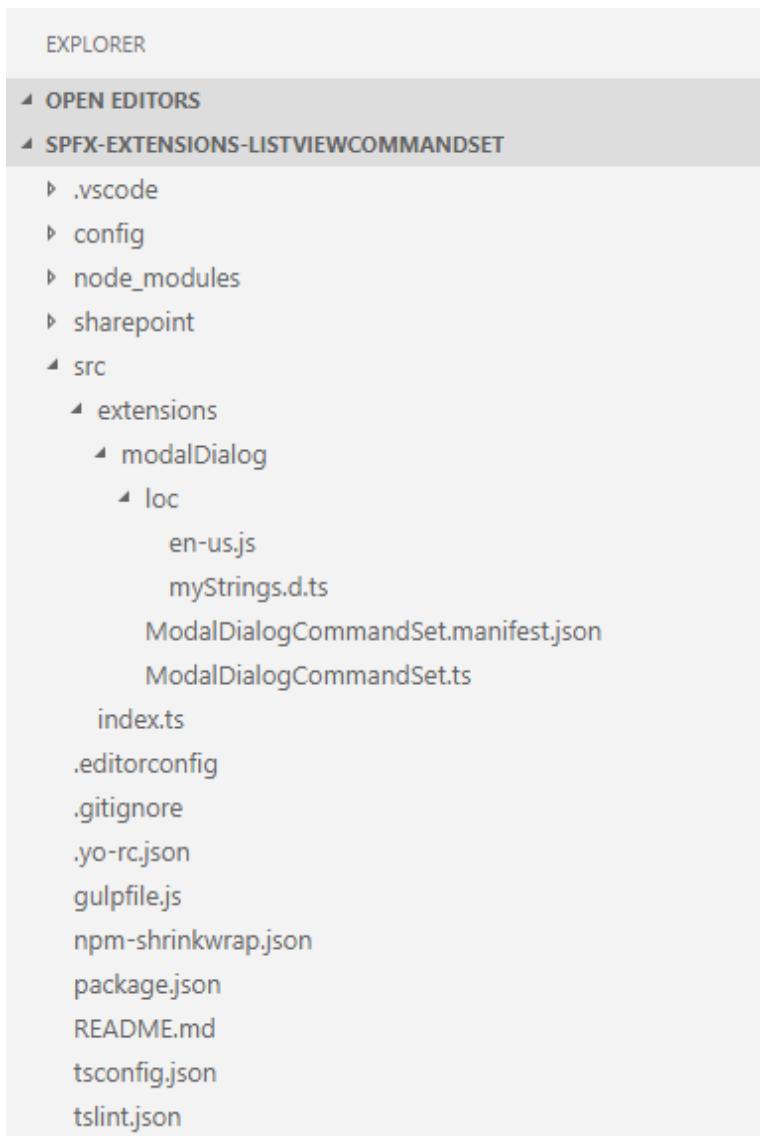
```
npm shrinkwrap
```

7. In the command prompt type below command to open the solution in code editor of your choice.

```
code .
```

Solution Structure

The solution structure is similar to client-side web parts with similar configuration options.



The file `ModalDialogCommandSet.manifest.json` inside the folder “`\src\extensions\modalDialog\`” defines the extension type and unique identifier for the solution. Please note down the id. We will need it later for debugging purpose.

```
{
  "$schema": "https://developer.microsoft.com/json-schemas/spfx/command-set-extension-manifest.schema.json",
  "id": "78c7b88e-c176-4694-b95d-1b371279b6e8",
  "alias": "ModalDialogCommandSet",
```

```

"componentType": "Extension",
"extensionType": "ListViewCommandSet",

// The "*" signifies that the version should be taken from the package.json
"version": "*",
"manifestVersion": 2,

// If true, the component can only be installed on sites where Custom Script
is allowed.
// Components that allow authors to embed arbitrary script code should set
this to true.
// https://support.office.com/en-us/article/Turn-scripting-capabilities-on-
or-off-1f2c515f-5d7e-448a-9fd7-835da935584f
"requiresCustomScript": false,

"items": {
  "COMMAND_1": {
    "title": { "default": "Command One" },
    "iconImageUrl": "icons/request.png",
    "type": "command"
  },
  "COMMAND_2": {
    "title": { "default": "Command Two" },
    "iconImageUrl": "icons/cancel.png",
    "type": "command"
  }
}
}

```

Implement Field Customizer

Open ModalDialogCommandSet.ts under “\src\extensions\modalDialog\” folder. The class has below important methods:

- **onInit()** event occurs before the page DOM is ready. It returns promise to perform asynchronous operations. onListViewUpdated() is not invoked until this promise is resolved.
- **onListViewUpdated()** event occurs separately for each command. The default implementation is as below

```

public onListViewUpdated(event:
  IListViewCommandSetListViewUpdatedParameters): void {
  const compareOneCommand: Command = this.tryGetCommand('COMMAND_1');
  if (compareOneCommand) {

```

```

    // This command should be hidden unless exactly one row is
    selected.
    compareOneCommand.visible = event.selectedRows.length === 1;
}
}
}

```

tryGetCommand helps to get the Command object, which is a representation of the command in UI.

- **onExecute()** method defines what happens when a command is executed. The default implementation is as below

```

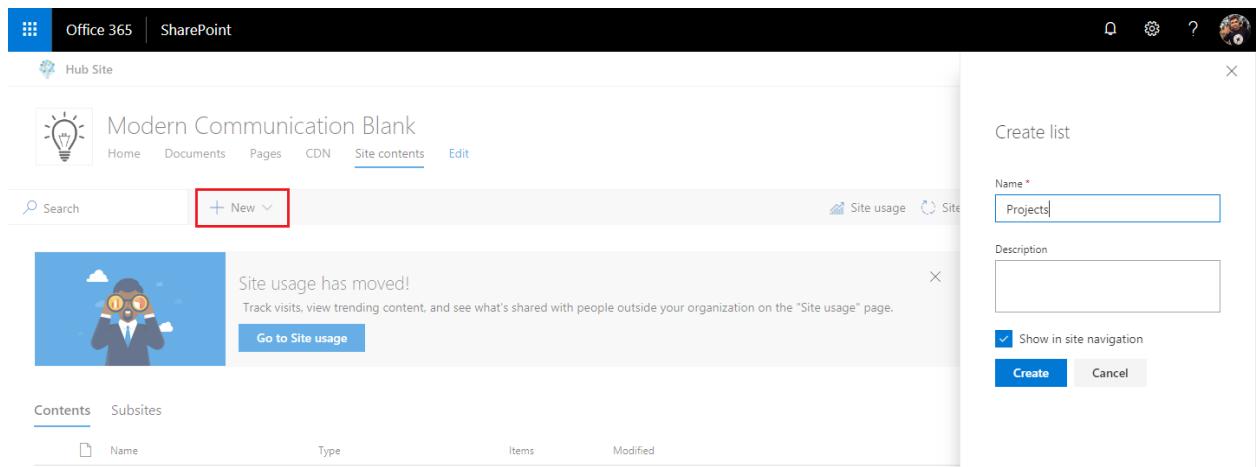
public onExecute(event: IListViewCommandSetExecuteEventArgs): void
{
    switch (event.itemId) {
        case 'COMMAND_1':
            Dialog.alert(` ${this.properties.sampleTextOne} `);
            break;
        case 'COMMAND_2':
            Dialog.alert(` ${this.properties.sampleTextTwo} `);
            break;
        default:
            throw new Error('Unknown command');
    }
}

```

Debug Field Customizer

The SharePoint local workbench now cannot be used to test the field customizer as we will need an actual SharePoint site to create needed list and fields.

1. Open SharePoint site.
2. From site contents create new list named “Projects”.



The screenshot shows a SharePoint Modern Communication Blank site. At the top, there's a navigation bar with 'Office 365' and 'SharePoint'. Below it, a 'Hub Site' header includes 'Home', 'Documents', 'Pages', 'CDN', 'Site contents' (which is underlined), and 'Edit'. A search bar and a 'New' button are also present. A prominent notification box says 'Site usage has moved!' with a link to 'Go to Site usage'. On the right, a 'Create list' dialog box is open, asking for a 'Name' (set to 'Projects') and a 'Description'. It also has a checked 'Show in site navigation' option and 'Create' and 'Cancel' buttons.

3. Click “Add column” to add a number field.

Projects

Title 

 + Add column

Single line of text

Multiple lines of text

 Number

Yes/No

Person

Date

Choice

Hyperlink

Picture

Currency

More...

Show/hide columns

4. Name it as **Completion**.

Create a column

[Learn more about column creation.](#)

Name *

Completion

Description

Project completion in percentage

Type

Number

Number of decimal places

Automatic

 Show as percentage

Default value

Enter a number

 Use calculated value (i)[More options](#)[Save](#)[Cancel](#)

5. Click **Save**.
6. Add some test data to the list.

Projects

Title	Completion
Cold Fusion	20
Honeycomb	60
Liberation	75
Phoenix	45
Topaz	100
Whistler	56

Update the Solution for Field changes

1. Open serve.json file under config folder. Update PageUrl to the url of list we created in the previous step.

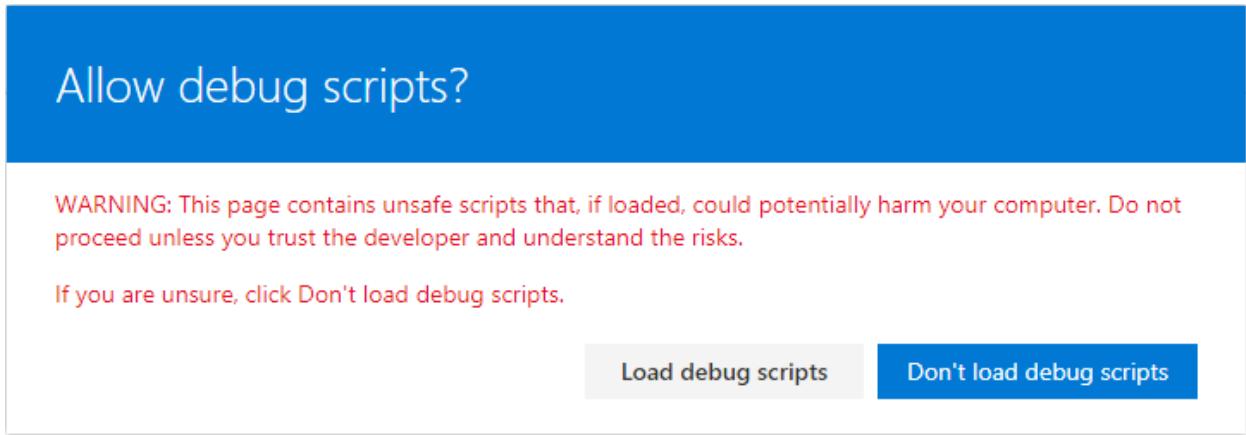
```
{
  "$schema": "https://developer.microsoft.com/json-schemas/core-build/serve.schema.json",
  "port": 4321,
  "https": true,
  "serveConfigurations": {
    "default": {
      "pageUrl": "https://contoso.sharepoint.com/sites/ModernCommunicationBlank/Lists/Projects/A11Items.aspx",
      "customActions": {
        "78c7b88e-c176-4694-b95d-1b371279b6e8": {
          "location": "ClientSideExtension.ListViewCommandSet.CommandBar",
          "properties": {
            "sampleTextOne": "One item is selected in the list",
            "sampleTextTwo": "This command is always visible."
          }
        }
      }
    }
  }
},
```

```
"modalDialog": {
  "pageUrl": "https://contoso.sharepoint.com/sites/ModernCommunicationBlank/Lists/Projects/A11Items.aspx",
  "customActions": {
    "78c7b88e-c176-4694-b95d-1b371279b6e8": {
      "location": "ClientSideExtension.ListViewCommandSet.CommandBar",
      "properties": {
        "sampleTextOne": "One item is selected in the list",
        "sampleTextTwo": "This command is always visible."
      }
    }
  }
}
```

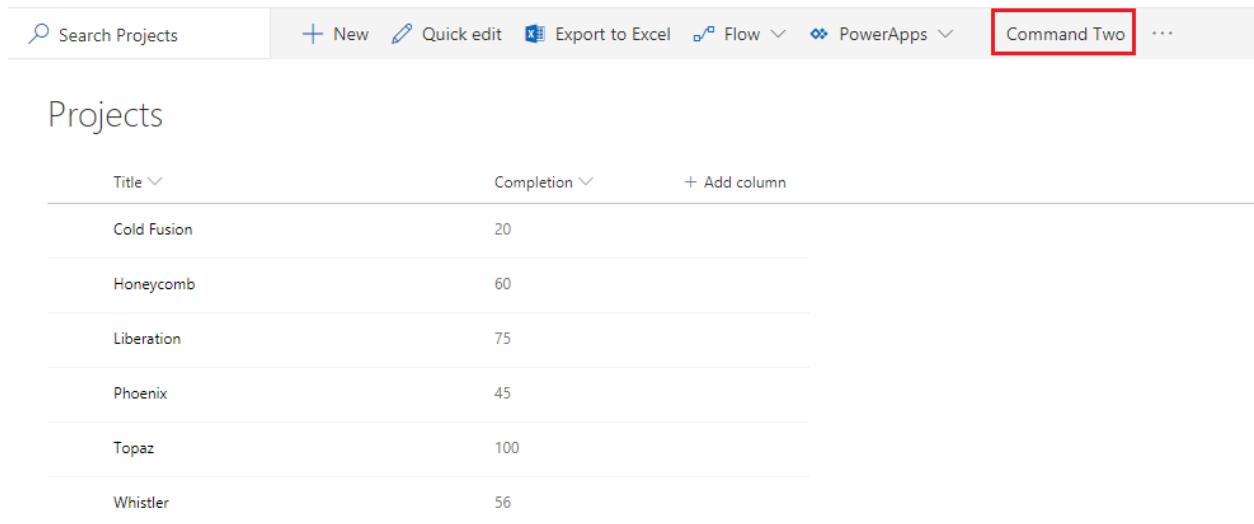
2. In the command prompt, type below command to run the extension.

```
gulp serve
```

3. Accept loading of debug manifests by clicking “Load debug scripts”.

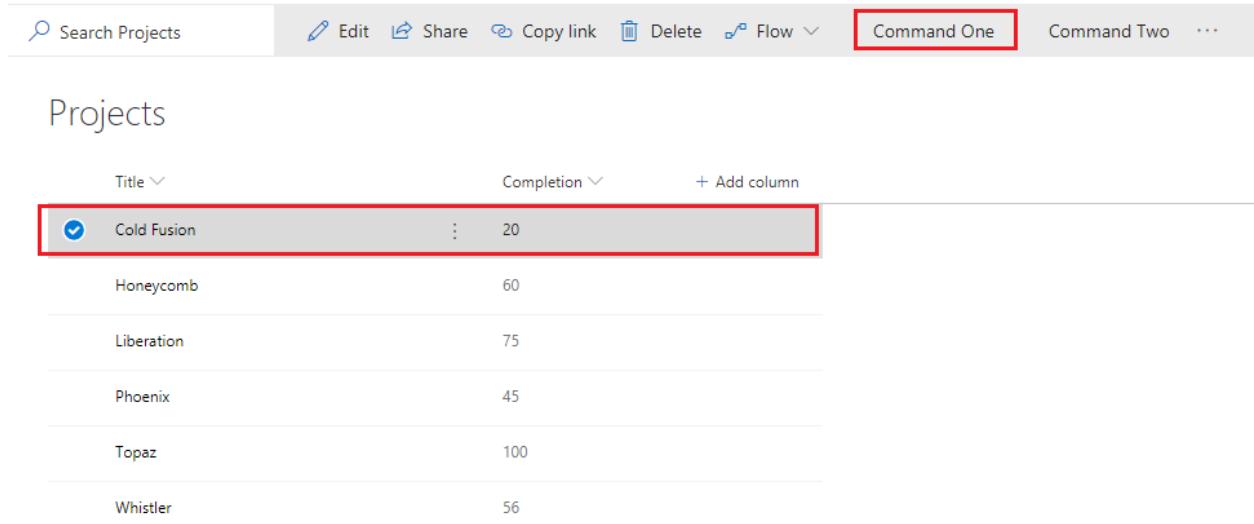


4. The Command Two button should appear in the toolbar.



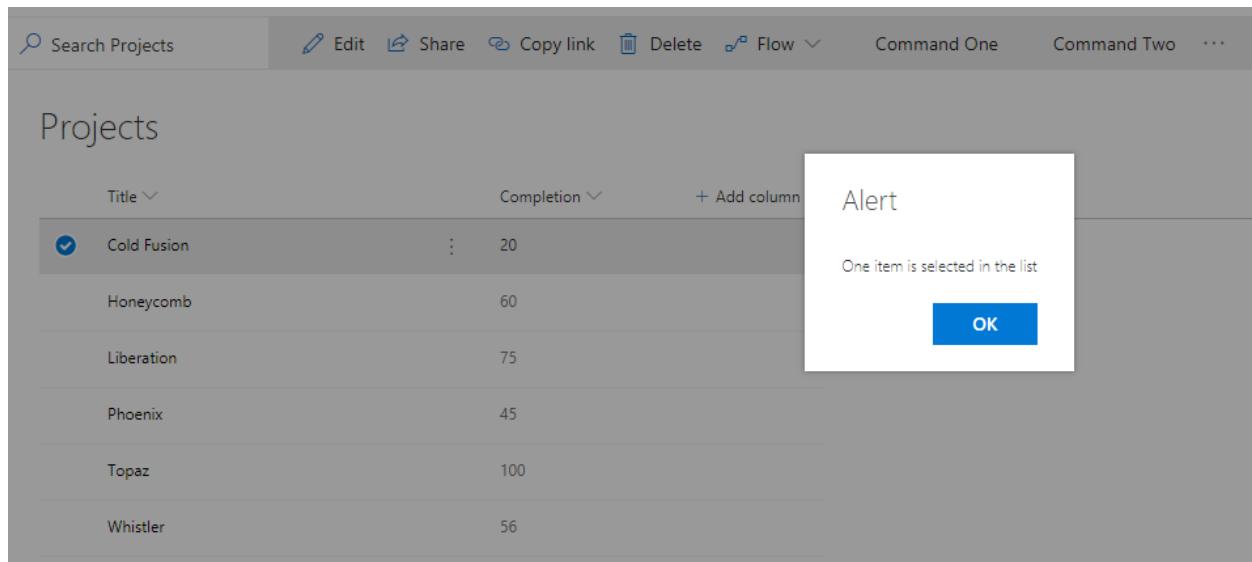
Title	Completion
Cold Fusion	20
Honeycomb	60
Liberation	75
Phoenix	45
Topaz	100
Whistler	56

5. “Command One” button is not visible until you select any list item.



Title	Completion
<input checked="" type="checkbox"/> Cold Fusion	20
Honeycomb	60
Liberation	75
Phoenix	45
Topaz	100
Whistler	56

6. Click “Command One” button, it will display an alert.

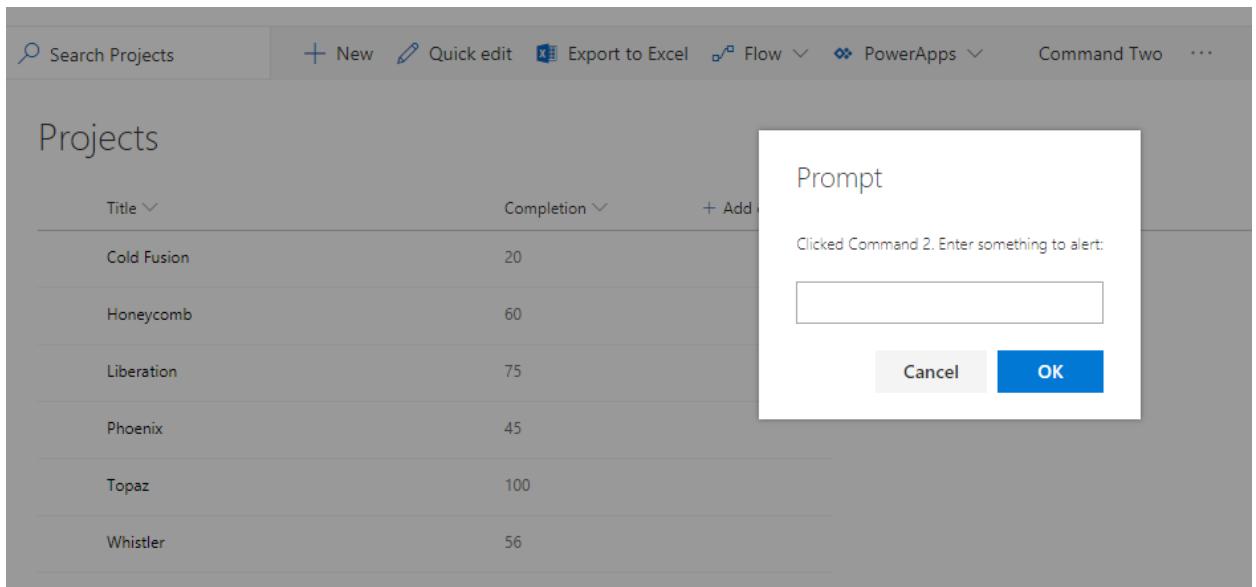


Apply List View Command Set Customization

1. Open ModalDialogCommandSet.ts file from “\src\extensions\modalDialog\” folder.
2. Update onExecute method.

```
public onExecute(event: IListViewCommandSetExecuteEventArgs): void {
    switch (event.itemId) {
        case 'COMMAND_1':
            Dialog.alert(`Clicked ${this.properties.sampleTextOne}`);
            break;
        case 'COMMAND_2':
            Dialog.prompt(`Clicked ${strings.Command2}. Enter something to
alert:`).then((value: string) => {
                Dialog.alert(value);
            });
            break;
        default:
            throw new Error('Unknown command');
    }
}
```

3. Make sure the gulp serve is running. Refresh the SharePoint list – Projects.



The screenshot shows a SharePoint list titled "Projects". The columns are "Title" and "Completion". The data is as follows:

Title	Completion
Cold Fusion	20
Honeycomb	60
Liberation	75
Phoenix	45
Topaz	100
Whistler	56

A modal dialog box titled "Prompt" is displayed over the list. It contains the text "Clicked Command 2. Enter something to alert:" followed by an input field. At the bottom are "Cancel" and "OK" buttons.

The ListView Command Set is now customized to show the prompt.

Summary

ListView Command Set SharePoint Framework extension helps to extend the toolbar of SharePoint list. This is an alternative to custom actions in the modern SharePoint sites.

Recommendations

- Learn TypeScript!
- Use SPHttpClient to connect to SharePoint
- HttpClient for other API's
- Use frameworks and libraries that already has typings
- Office UI Fabric available for consistent styling

Resources

- SPFx Overview
<https://dev.office.com/sharepoint/docs/sharepoint-framework-overview>
- SPFx Tutorials
<http://aka.ms/spfx-tutorials>
- Build your first webpart
<https://dev.office.com/sharepoint/docs/spfx/web-parts/get-started/build-a-hello-world-web-part>
- SharePoint Framework API
<https://dev.office.com/sharepoint/reference/spfx/sharepoint-framework-reference-overview>
- Get an introduction to the SharePoint Framework
<https://channel9.msdn.com/Events/Ignite/2016/BRK2114-TS>
- SharePoint PnP Community
<http://aka.ms/sppnp-community>
<http://aka.ms/spdev-docs>

The Final Word

SharePoint Framework is the future of SharePoint. The directions from Microsoft is to use the SPFx. I hope this book has helped you to get started your journey with SPFx. Thanks for sticking with me on this journey, and hopefully we'll cross paths again in the future.