

SPFx

Getting Started with SharePoint Framework Development using TypeScript, PnP JS and React JS



Contents

Set up SharePoint Framework Development Environment.....	5
Install Node JS.....	5
Install Yeoman and Gulp	9
Install Yeoman SharePoint Generator	10
Code Editor	11
Additional Tools for Development and Debugging	14
Fiddler.....	14
Postman	16
Getting Started with SharePoint Framework Development using TypeScript.....	17
Create the First Hello World Client Web part	17
Create the Web part project	17
Test the web part.....	20
SharePoint Workbench.....	21
Edit the web part	23
Add the web part to SharePoint	25
Create SharePoint Framework Client Web Part to Retrieve and Display List Items.....	27
Create the Web part Project	28
Test the Web part locally.....	30
Edit the web part	32
Define List Model.....	33
Create Mock HttpClient to test data locally	33
Retrieve SharePoint List Items	35
Render the SharePoint List Items from Employee List	36
TS File Contents.....	37
Mock HTTP Client Content	40
Test the Web part in local SharePoint Workbench.....	40
Test the Web part in SharePoint Online.....	41
Provision Custom SharePoint List	43

Create the Web part Project	43
Edit the web part	45
Package and Deploy the Solution	47
Provision SharePoint List with custom Site Columns and Content Type.....	52
Edit the web part	54
Add the Default data to SharePoint List.....	56
Elements.XML.....	56
Schema.XML.....	59
Update Package-Solution.json.....	61
Package and Deploy the Solution	63
Upgrade the Solution	69
Upgrade the solution and add a new list.....	70
Package and Deploy the Solution	74
Resolve Package Errors.....	75
Getting Started with PnP JS development using SharePoint Framework.....	77
Retrieve SharePoint List Items using PnP JS and SharePoint Framework	77
Edit Webpart	79
Define List Model.....	80
Create Mock HttpClient to test data locally	81
Retrieve SharePoint List Items	83
Retrieve the SharePoint List Items From Employee List.....	83
TS File Contents to retrieve list data using PnP	85
Package and Deploy the Solution	87
Test the Web part in local SharePoint Workbench.....	88
Test the Web part in SharePoint Online.....	89
SharePoint List Creation using PnP and SPFx	90
Edit the web part	92
Install PnP JS Module.....	93
Create List using PnP method	93
TS File code for Creating the List.....	94
Test the Web part in SharePoint Online.....	96
Retrieve User Profile Properties using SPFx and PnP JS	98
Create the Web part Project	98
Edit the web part	99

Retrieve User Profile data	101
TS File content to retrieve User Profile Data.....	102
Test the Web part in SharePoint Online.....	103
Retrieve SharePoint Search Results using SPFx webpart.....	105
Create the Web part Project	105
Retrieve Search Results	107
TS File contents for displaying the retrieved search results	108
Test the Web part in SharePoint Online.....	110
Implement SharePoint List item CRUD using SPFx and PnP JS.....	111
Create the Web part Project	111
Edit the web part	112
Implement CRUD using PnP JS.....	113
TS File contents for implementing CRUD using PnP.....	114
Test the Web part in SharePoint Online.....	117
Add Item	119
Update Item	120
Delete item	122
Getting Started with REACT JS in SharePoint	123
Retrieve SharePoint List data using REST API and display using Content Editor Webpart .	123
REACT and REST API script to display SharePoint data as Grid	125
Create SPFx Webpart to retrieve SharePoint List Items using REACT & REST API.....	128
Edit the web part	129
Exploring the File Structure	130
ReactGetItemsWebPart.ts	131
IReactgetItemsProps.TS.....	131
ReactGetItems/modue.scss	131
ReactGetItems.tsx	133
Test the Web part in SharePoint Online.....	135
TSX File contents for retrieving list items using REST API and REACT.....	136
Summary	137

SharePoint Framework is the new development model in which lots of work had been going on in the past year. It went [to General Availability](#) on Feb 23rd 2017. It is a page and web part model that provides full support for client-side SharePoint development, easy integration with SharePoint data and support for open source tooling. With the SharePoint Framework, you can use modern web technologies and tools in your preferred development environment to build productive experiences and apps in SharePoint.

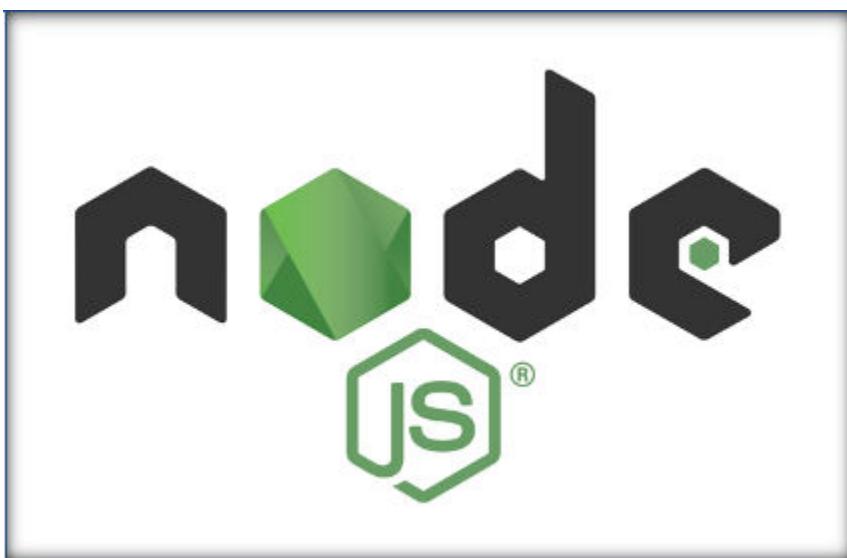
In this book, we will see how to set up the environment for getting started with development using SharePoint Framework and create client web parts using the new development model. We will make use of Typescript, PnP JS and React JS to create the webparts as we progress. This book would serve as a script cookbook to help you get started with the different frameworks used with SPFx.

Set up SharePoint Framework Development Environment

Let us see how to set up the development environment so that we can kick start with SharePoint Framework development. Below are the required components that we will have to install in the environment.

- Node JS
- Yeoman and Gulp
- Yeoman SharePoint Generator
- Code Editor(Visual Studio Code/Webstorm)
- Postman and Fiddler(optional)

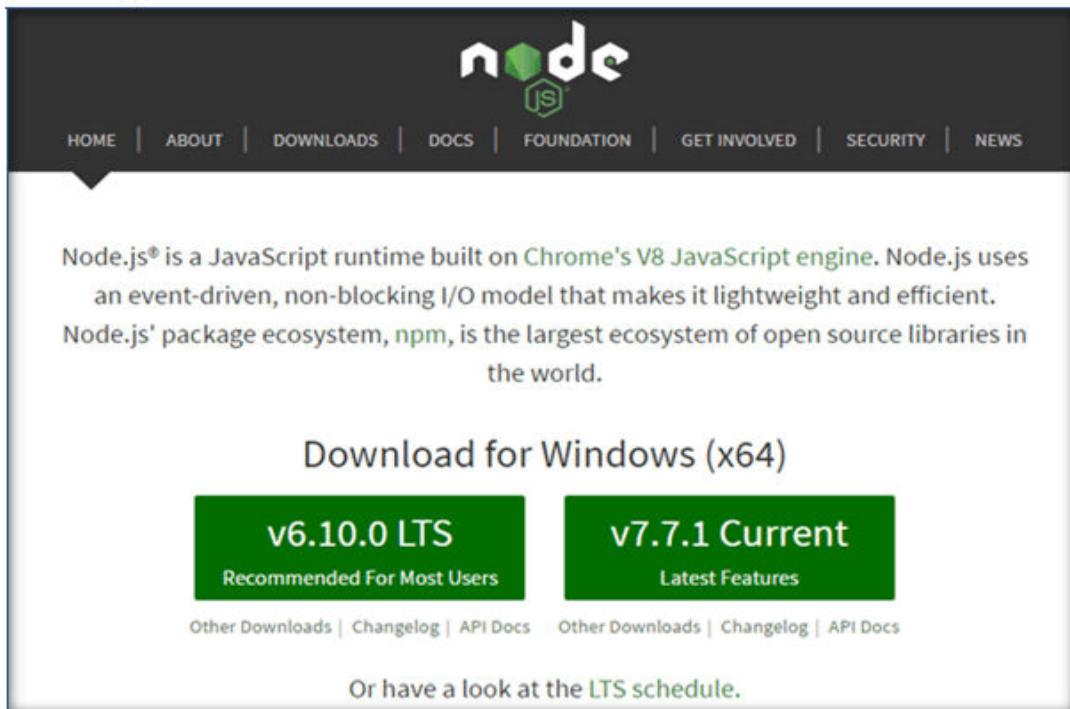
Install Node JS



Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the

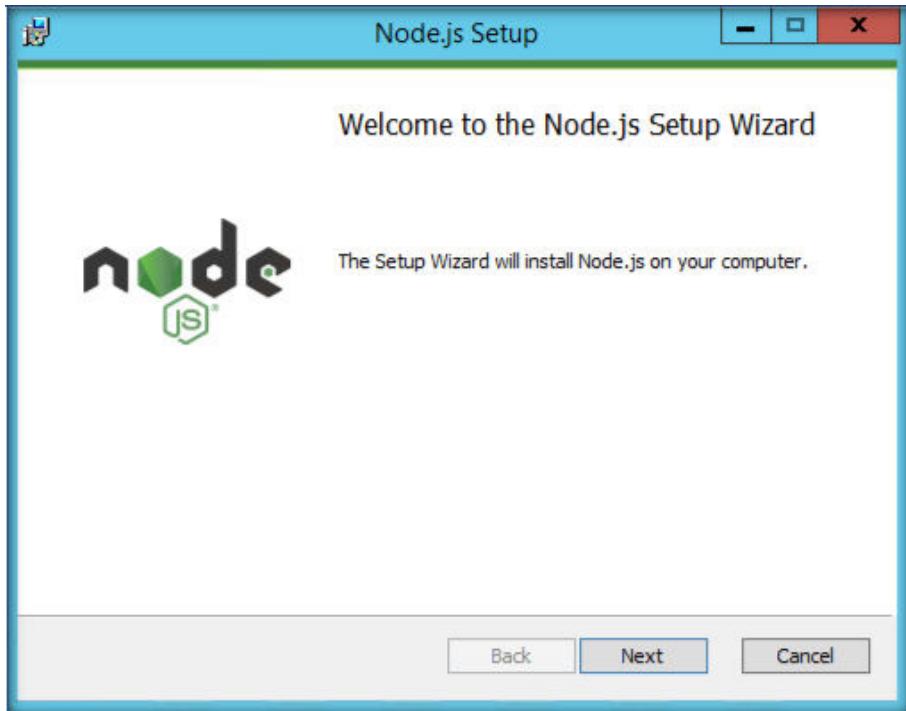
largest ecosystem of open source libraries in the world. We will be making use of npm along with Yeoman and Gulp to package and deploy modules.

As the first step we will install NodeJS Long Term Support Version (LTS). We can install Node JS from this [link](#).

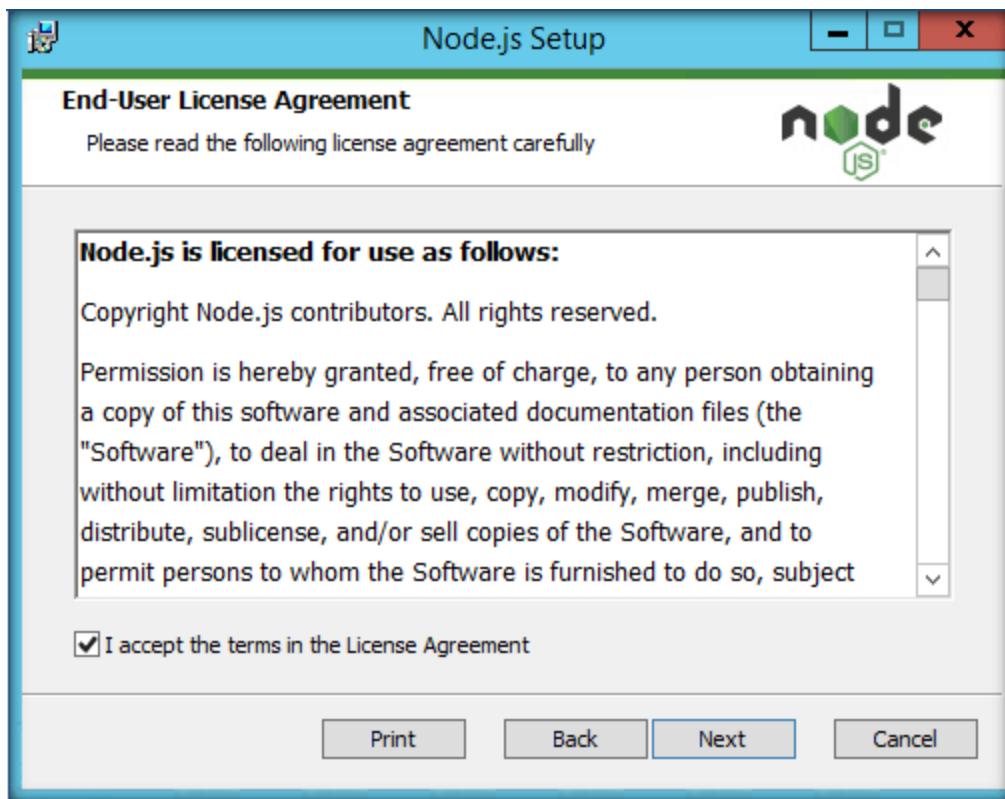


The screenshot shows the official Node.js website. At the top, there's a navigation bar with links for HOME, ABOUT, DOWNLOADS, DOCS, FOUNDATION, GET INVOLVED, SECURITY, and NEWS. The main content area features a large green header with the Node.js logo. Below it, a paragraph explains what Node.js is: "Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient." It also mentions that Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world. Two prominent download buttons are shown: one for "v6.10.0 LTS" (labeled "Recommended For Most Users") and another for "v7.7.1 Current" (labeled "Latest Features"). At the bottom of the page, there are links for "Other Downloads", "Changelog", and "API Docs" for both the LTS and Current versions. A note at the bottom encourages users to look at the LTS schedule.

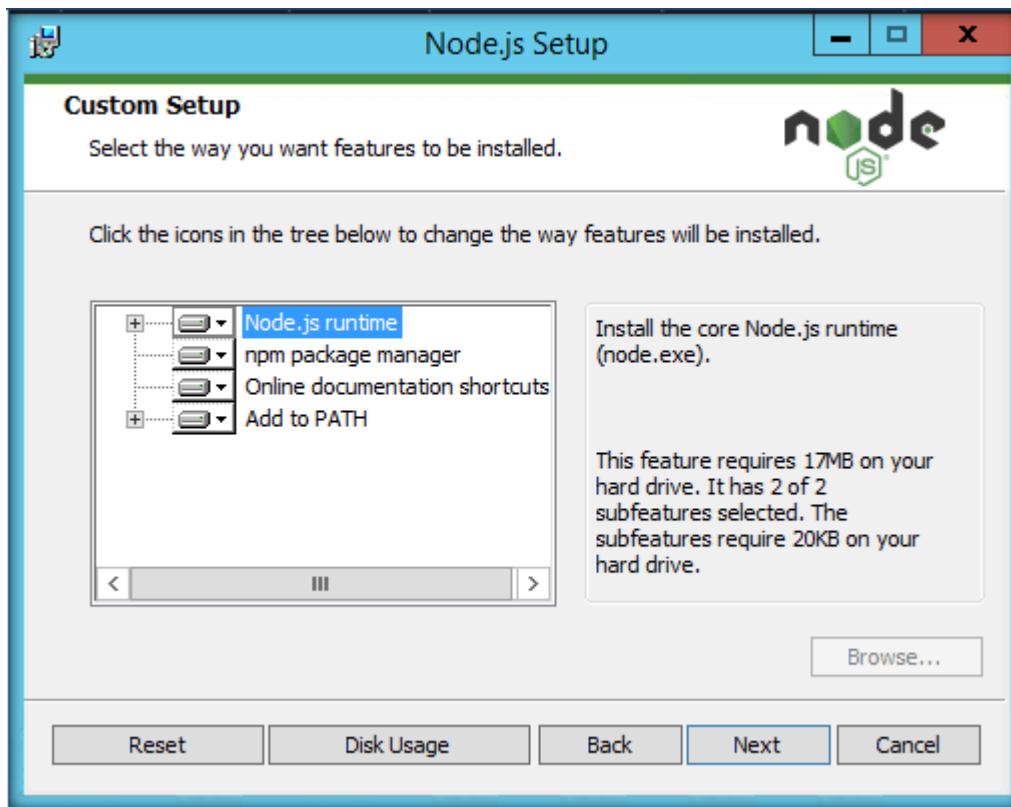
Once we have downloaded the LTS version, run the executable file and proceed.



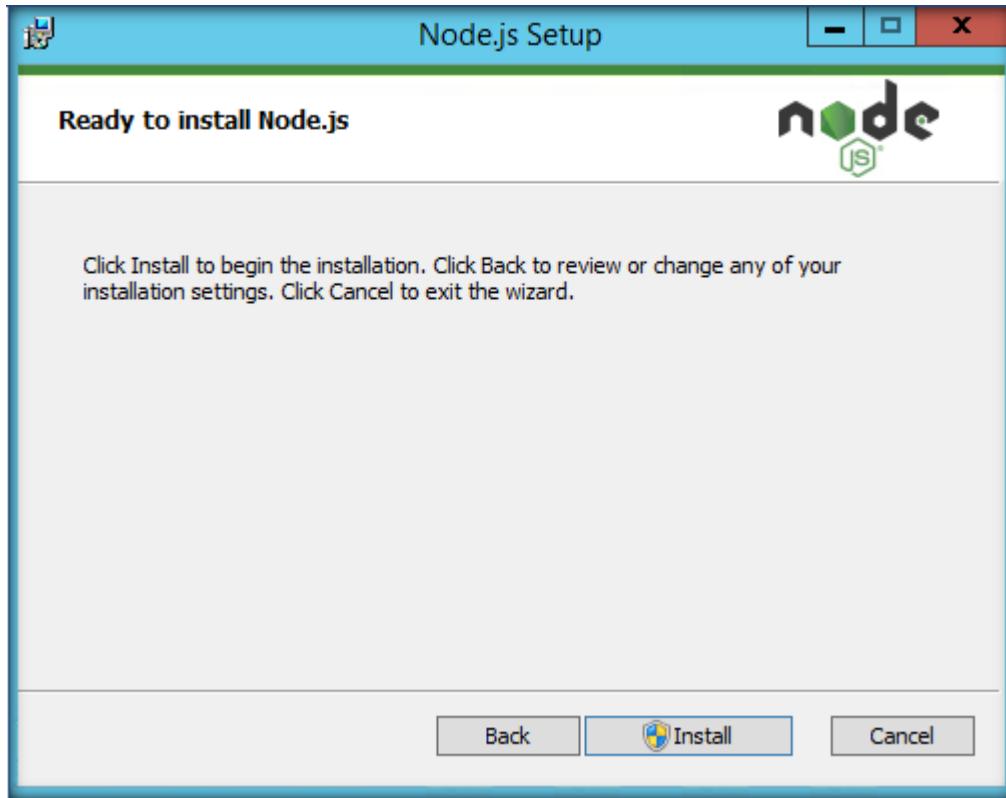
Accept the license agreement and click on Next.



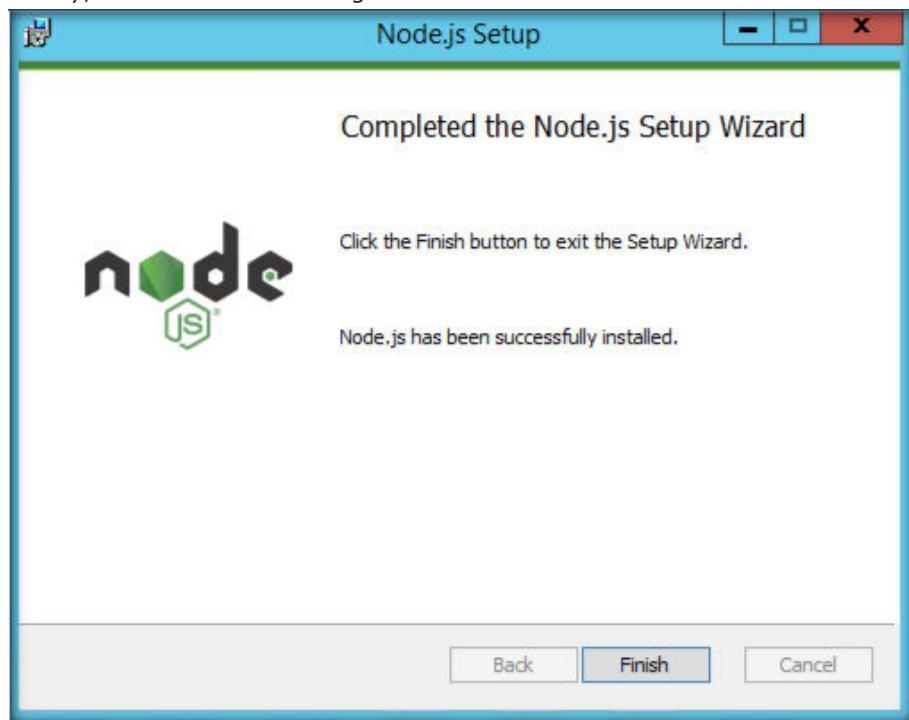
We will select Node.js run time installation.



Click on Install to start the installation procedure.

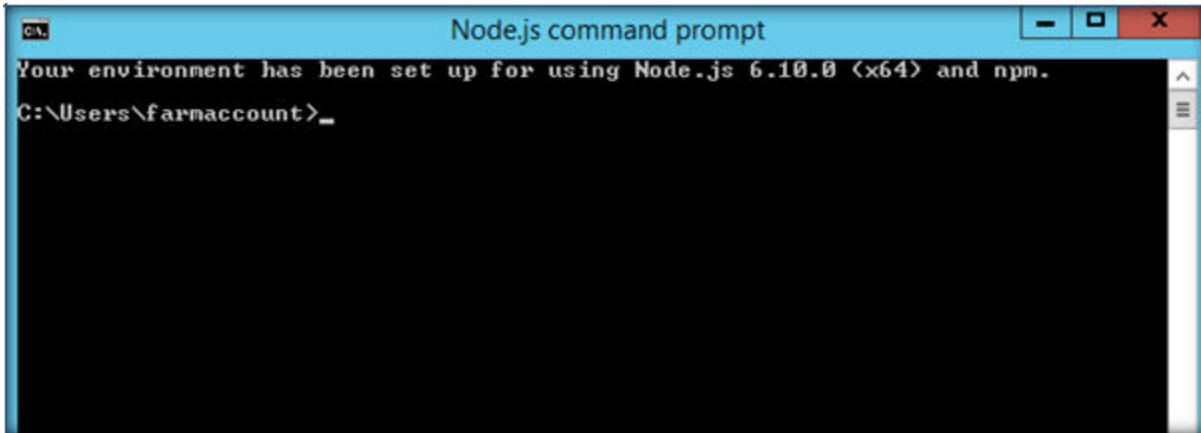


Finally, we are done installing NodeJS.



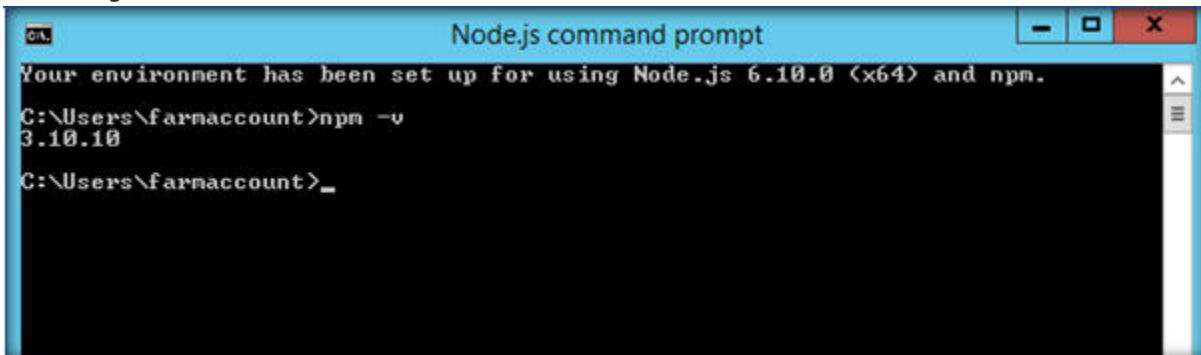
Click on Finish button and restart your computer. You won't be able to run Node.js until you restart your computer.

If we run the NodeJS command prompt, we will get the message as shown below. Thus, the Node JS has been successfully installed in the local machine.



A screenshot of a Windows Command Prompt window titled "Node.js command prompt". The window shows the text: "Your environment has been set up for using Node.js 6.10.0 <x64> and npm." followed by the command prompt "C:\Users\farmaccount>".

Now, let's see the version of Node Package Manager (npm) by running the command **npm -v**. It is running V3 version.



A screenshot of a Windows Command Prompt window titled "Node.js command prompt". The window shows the text: "Your environment has been set up for using Node.js 6.10.0 <x64> and npm.", followed by the command "npm -v", and the output "3.10.10". Below this is the command prompt "C:\Users\farmaccount>".

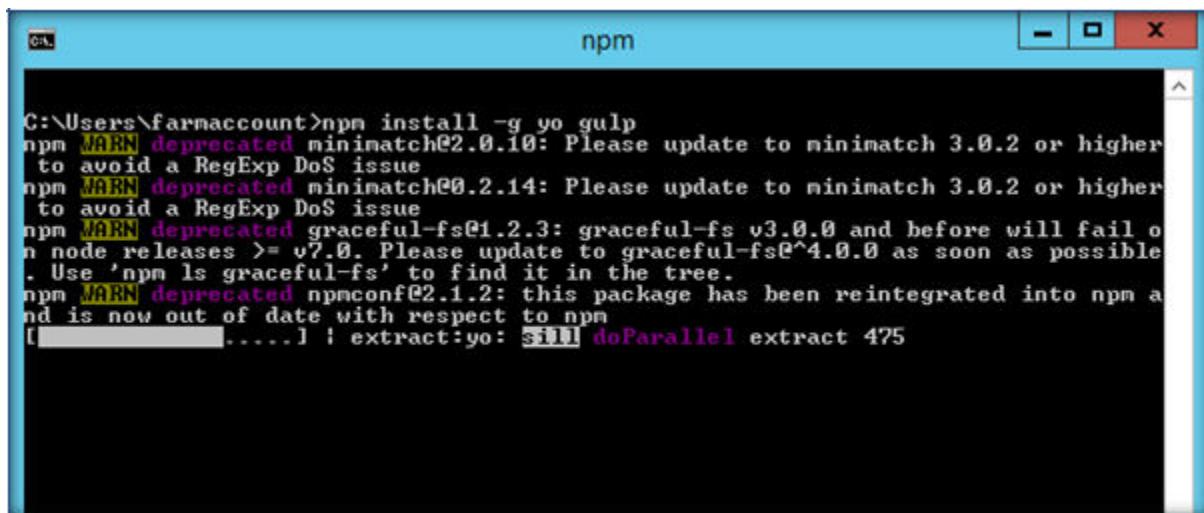
Install Yeoman and Gulp



[Yeoman](#) is a scaffolding tool for modern web apps. It helps you to kick-start new projects, prescribing best practices and tools to help you stay productive. Often called Yo, it scaffolds out a new

application, writing your build configuration (e.g Gulpfile) and pulling in relevant build tasks and package manager dependencies (e.g npm) that you might need for your build. Gulp is a JavaScript task runner that helps us automate common tasks like refreshing your browser when you save a file, Bundling and minifying libraries and CSS, Copying modified files to an output directory etc. We will be using Yo and Gulp together for creating SharePoint Client Webparts. Now, let's install Yeoman and Gulp simultaneously by running the below command:

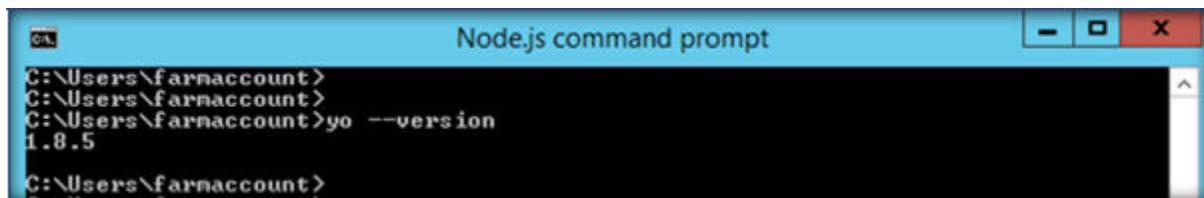
```
npm install -g yo gulp
```



```
C:\Users\farmaccount>npm install -g yo gulp
npm [WARN] deprecated minimatch@2.0.10: Please update to minimatch 3.0.2 or higher
to avoid a RegExp DoS issue
npm [WARN] deprecated minimatch@0.2.14: Please update to minimatch 3.0.2 or higher
to avoid a RegExp DoS issue
npm [WARN] deprecated graceful-fs@1.2.3: graceful-fs v3.0.0 and before will fail o
n node releases >= v7.0. Please update to graceful-fs@^4.0.0 as soon as possible
- Use 'npm ls graceful-fs' to find it in the tree.
npm [WARN] deprecated npmconf@2.1.2: this package has been reintegrated into npm a
nd is now out of date with respect to npm
[.....] : extract:yo: still doParallel extract 475
```

We can get the version of Yeoman using the command:

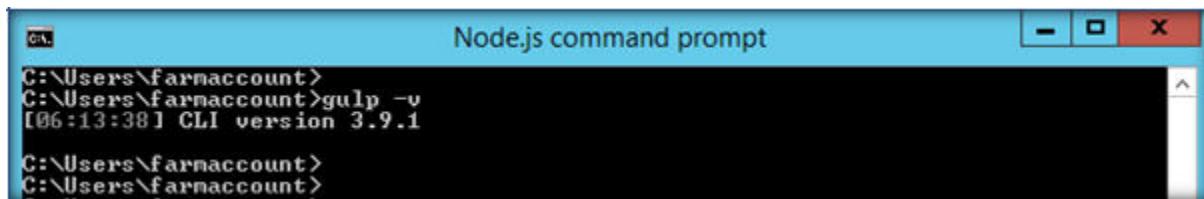
```
yo --version
```



```
C:\Users\farmaccount>
C:\Users\farmaccount>
C:\Users\farmaccount>yo --version
1.8.5
C:\Users\farmaccount>
```

We can get the Gulp Version using the command:

```
gulp -v
```



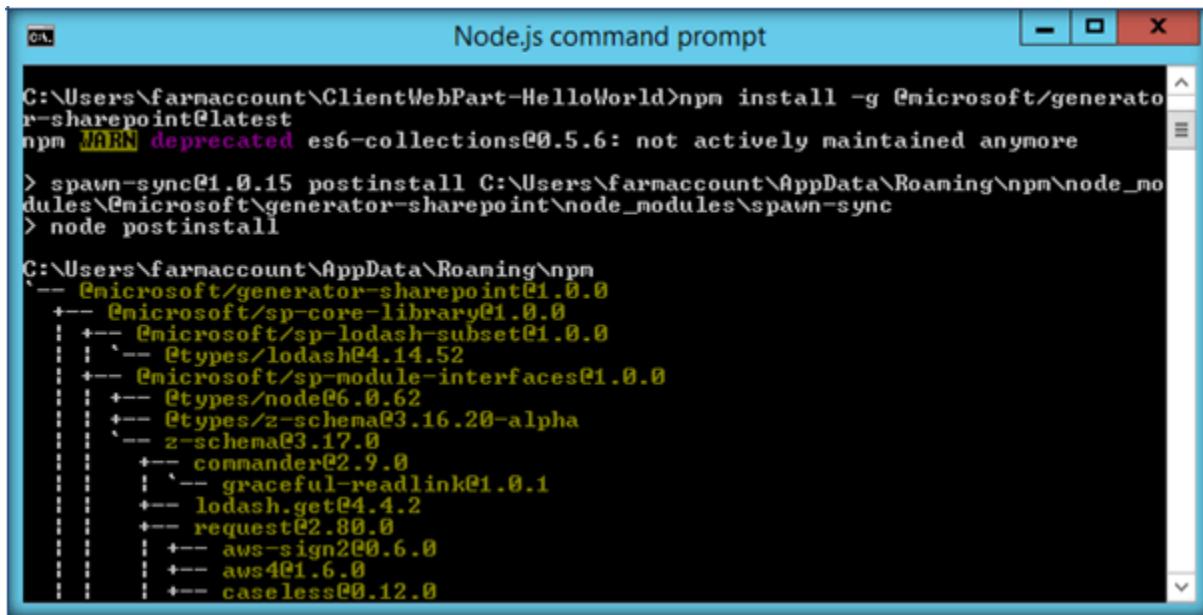
```
C:\Users\farmaccount>
C:\Users\farmaccount>
C:\Users\farmaccount>gulp -v
[06:13:38] CLI version 3.9.1
C:\Users\farmaccount>
C:\Users\farmaccount>
```

Install Yeoman SharePoint Generator

The Yeoman SharePoint web part generator helps you to quickly create a SharePoint client-side solution project with the right tool chain and project structure.

Yeoman SharePoint Generator can be installed using the below command:

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

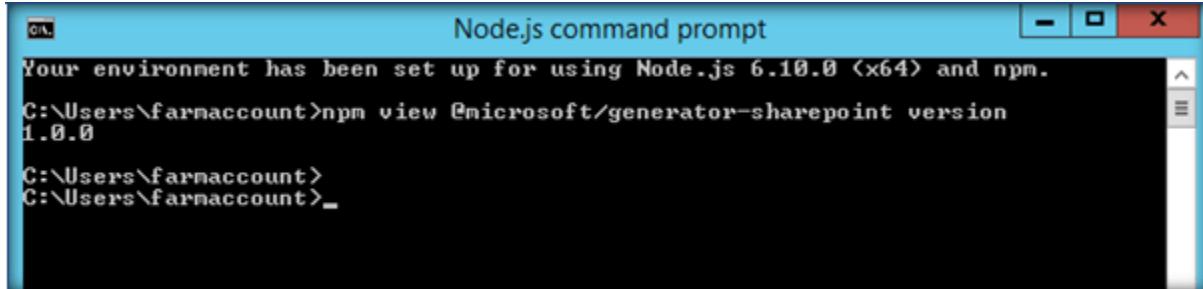


```
Node.js command prompt
C:\Users\farmaccount\ClientWebPart-HelloWorld>npm install -g @microsoft/generator-sharepoint@latest
npm WARN deprecated es6-collections@0.5.6: not actively maintained anymore
> spawn-sync@1.0.15 postinstall C:\Users\farmaccount\AppData\Roaming\npm\node_modules\@microsoft\generator-sharepoint\node_modules\spawn-sync
> node postinstall

C:\Users\farmaccount\AppData\Roaming\npm
`-- @microsoft/generator-sharepoint@1.0.0
  +-- @microsoft/sp-core-library@1.0.0
  |  +-- @types/lodash@4.14.52
  |  +-- @microsoft/sp-module-interfaces@1.0.0
  |  +-- @types/node@6.0.62
  |  +-- @types/z-schema@3.16.20-alpha
  |  +-- z-schema@3.17.0
  |  |  +-- commander@2.9.0
  |  |  |  +-- graceful-readlink@1.0.1
  |  |  +-- lodash.get@4.4.2
  |  |  +-- request@2.80.0
  |  |  |  +-- aws-sign2@0.6.0
  |  |  |  +-- aws4@1.6.0
  |  |  +-- caseless@0.12.0
```

We can get the version of Yeoman Generator by running the below command. As we can see 1.0.0 indicates General Availability version.

```
npm view @microsoft/generator-sharepoint version
```



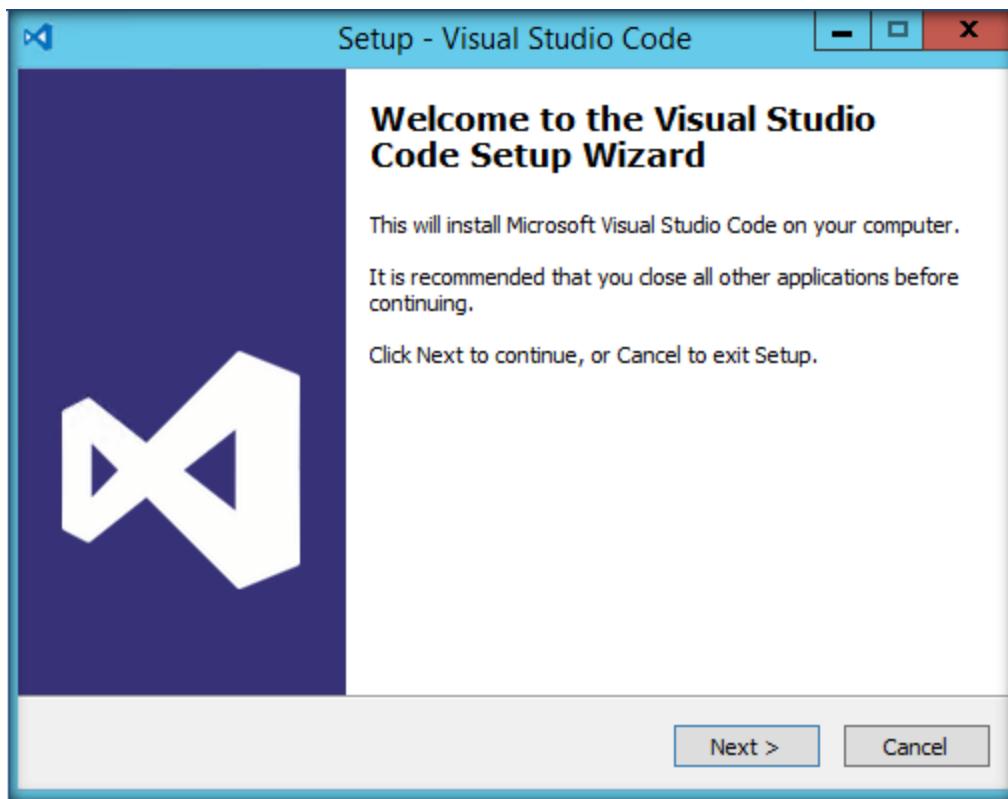
```
Node.js command prompt
Your environment has been set up for using Node.js 6.10.0 (x64) and npm.
C:\Users\farmaccount>npm view @microsoft/generator-sharepoint version
1.0.0
C:\Users\farmaccount>
C:\Users\farmaccount>
```

Code Editor

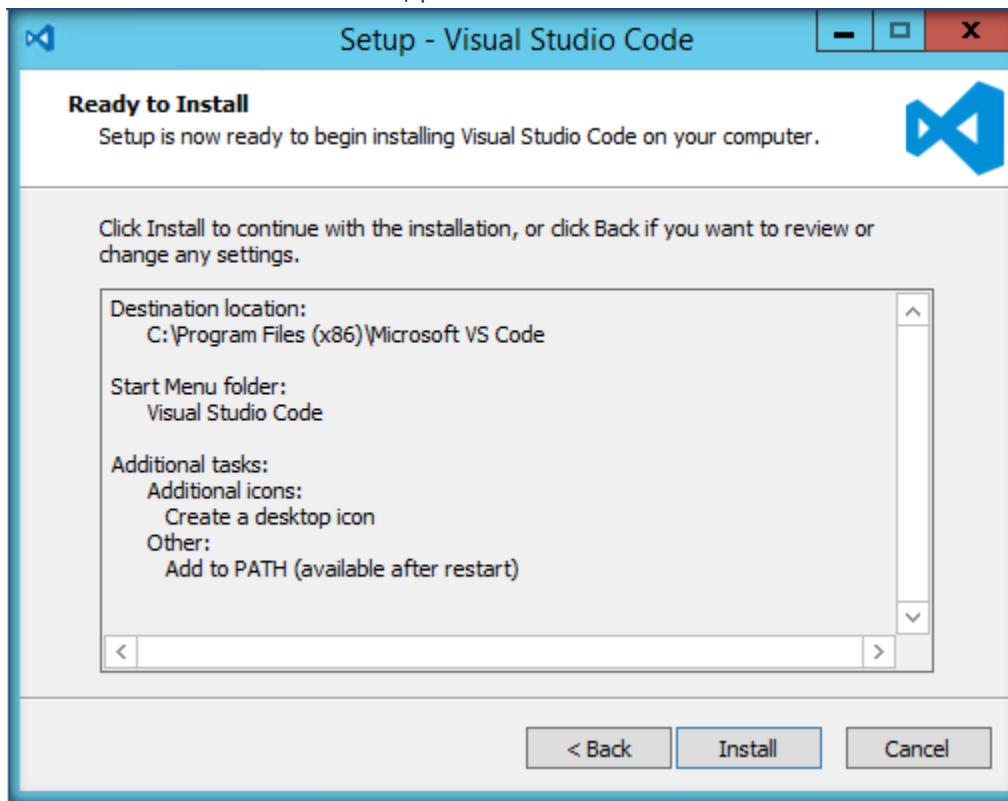
Next, we need a code editor that will help us with code editing. We can use any code editor or IDE that supports client-side development to build our web part, such as:

- Visual Studio Code
- Atom
- Webstorm

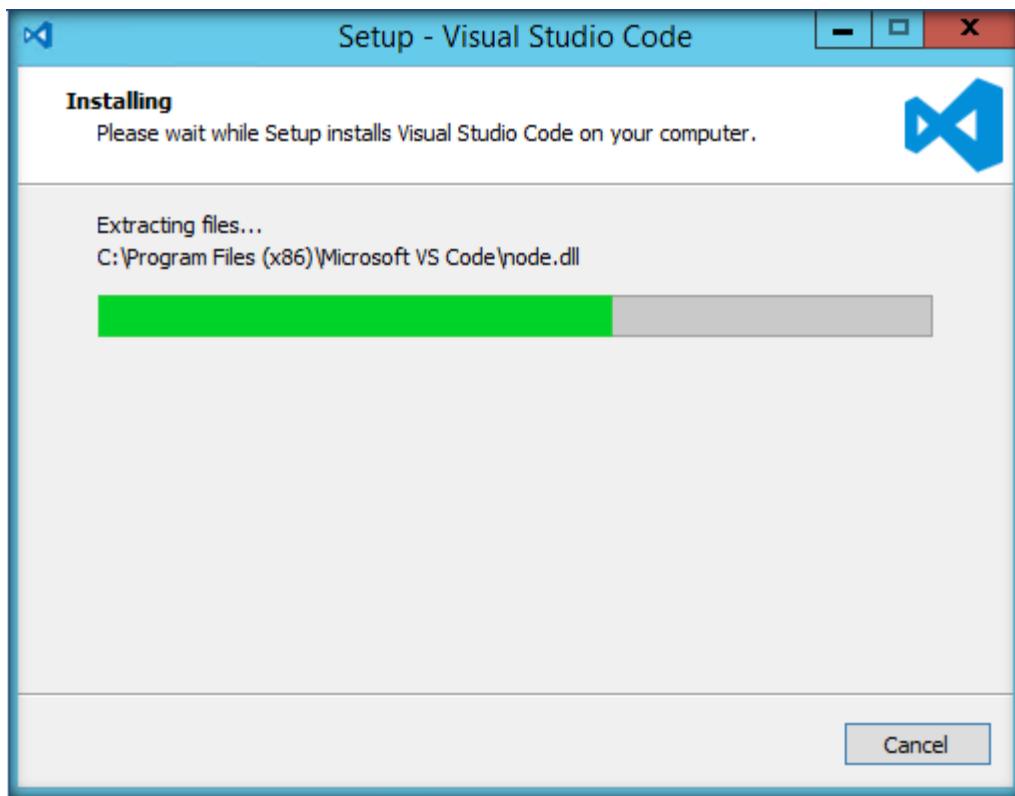
We will use Visual Studio Code in this walkthrough. You can get it from [here](#).



Once we have downloaded the exe, proceed with the installation.



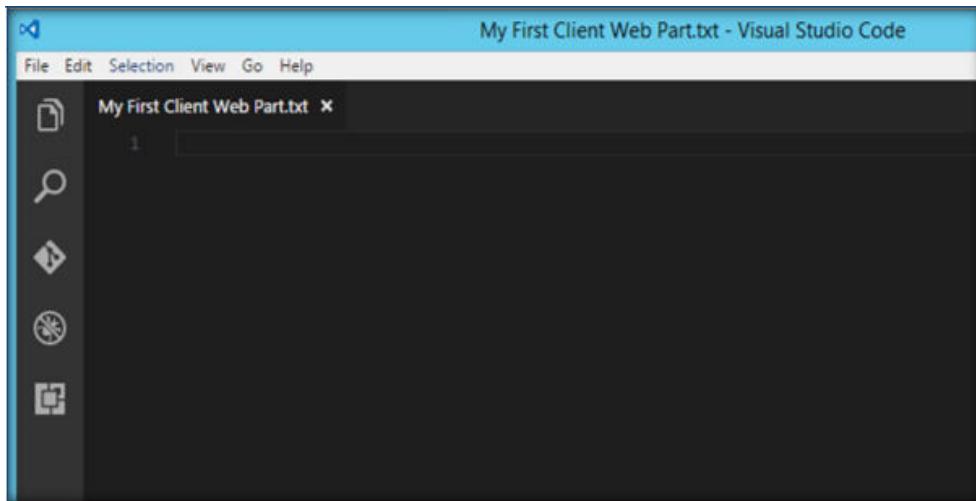
Click on Install to start the installation procedure.



Finally, we have completed installation of the Visual Studio Code editor.



Sample Screen Shot,



Additional Tools for Development and Debugging

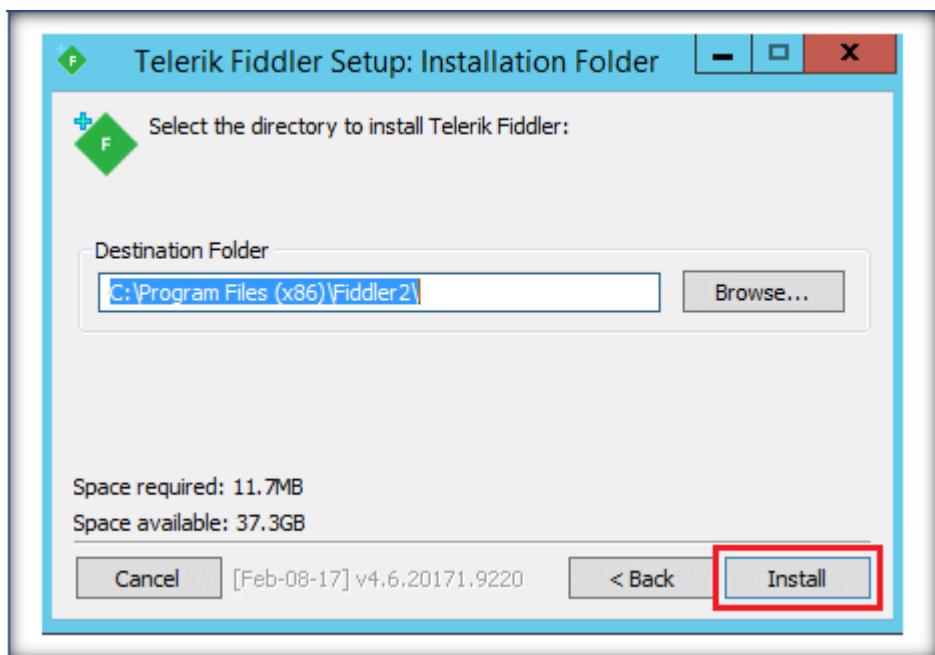
Once we start the development, we must debug or test the application. Fiddler and Postman can help us in this task.

Fiddler

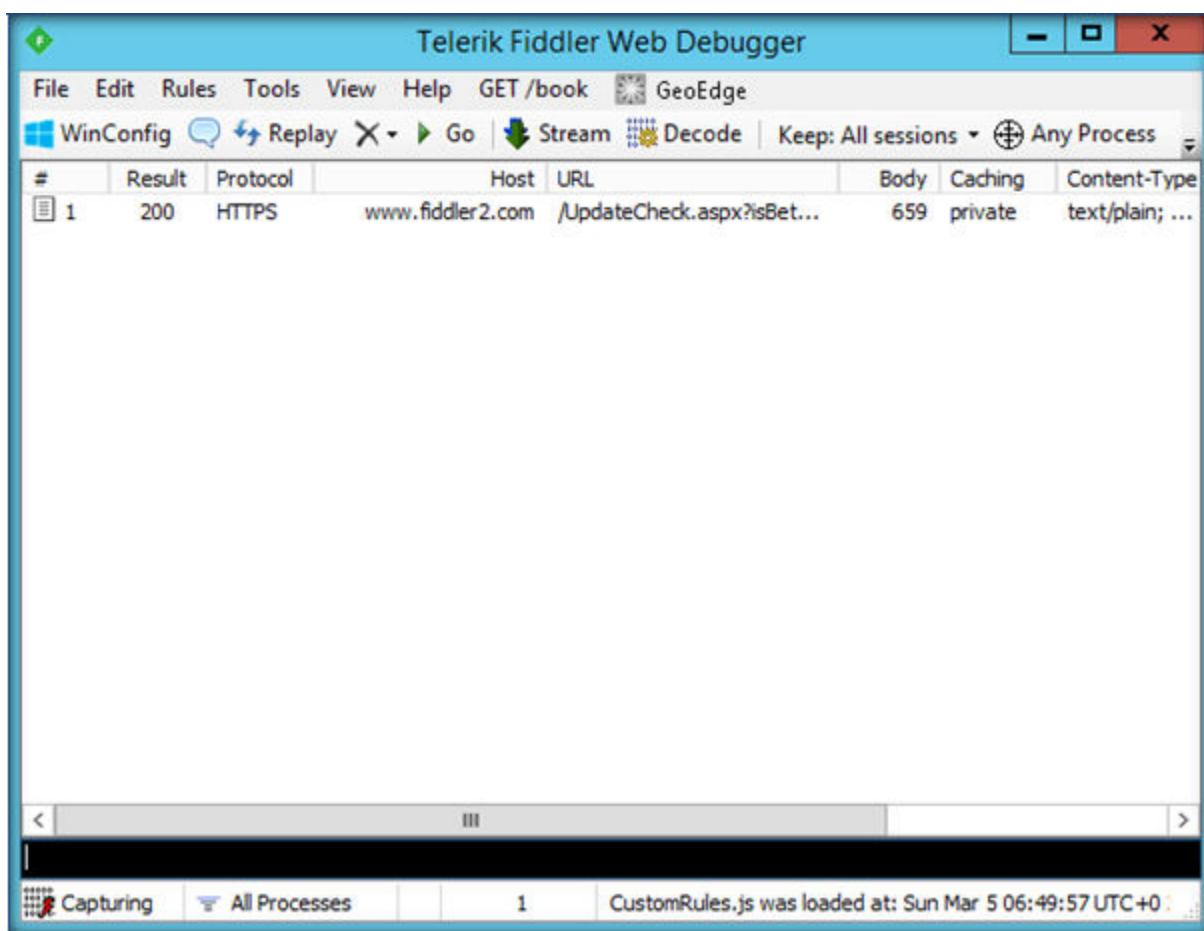
Fiddler is an HTTP debugging proxy server application. It captures HTTP and HTTPS traffic and logs it for the user to review. You can get fiddler from [here](#).



Once the executable has been downloaded. Click on Install to set up Fiddler in your local machine.

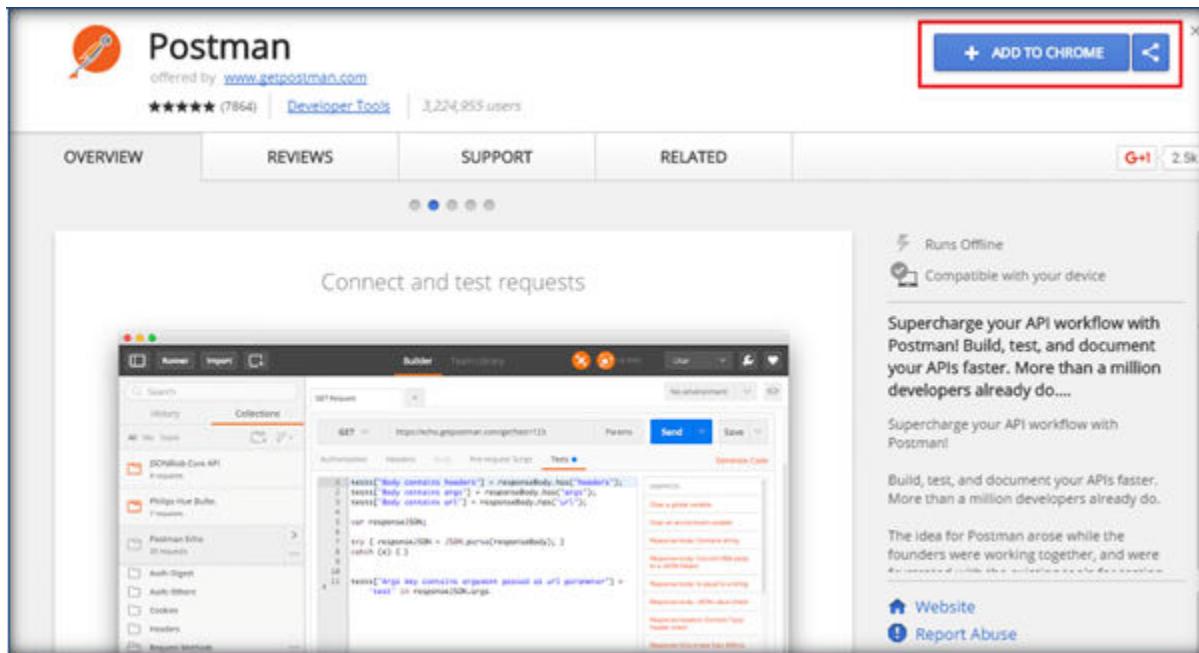


Using fiddler, we can examine the traffic as it is being sent or received.

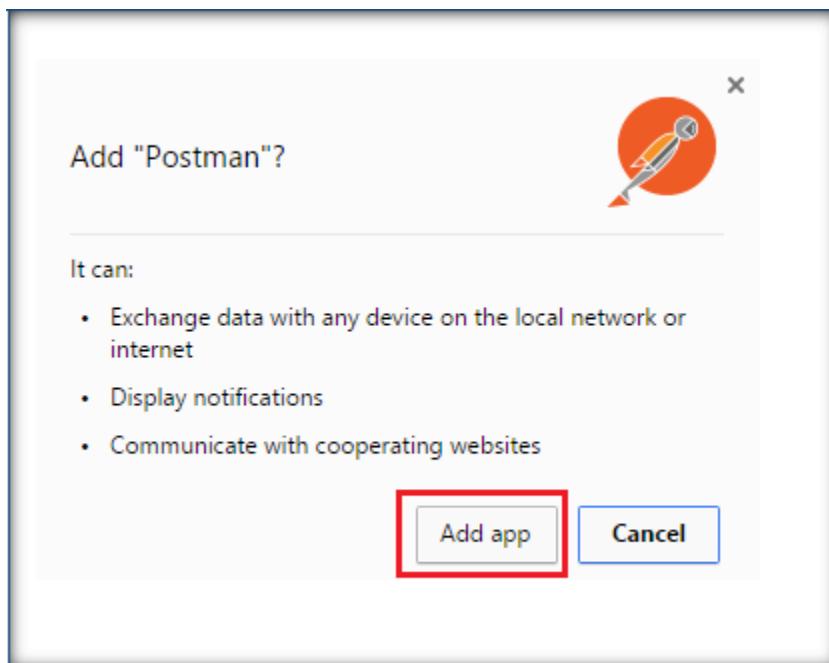


Postman

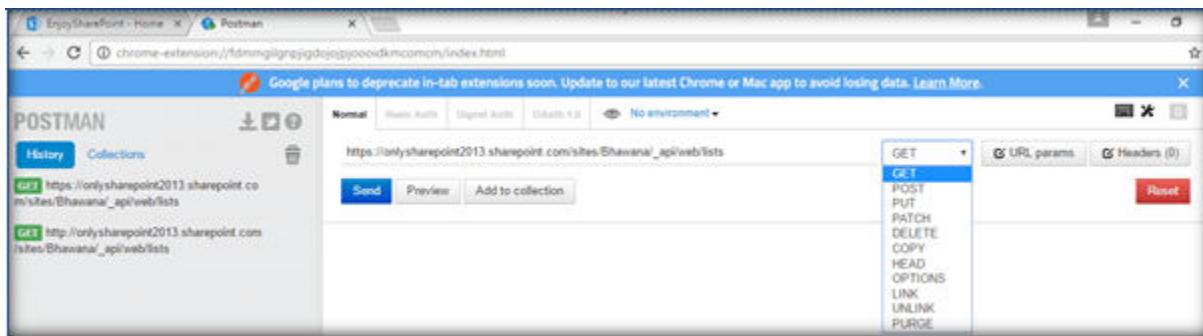
Postman can be used to test SharePoint's REST service endpoints and verify the returned data and request headers. We can get Postman from [here](#).



Postman can be added to Chrome as an app.



The REST URL can be entered in the Request URL field and we can click on Send to get the SharePoint data.



Thus, we saw how to set up the environment and now we are ready to get started with the new SharePoint Framework development model.

Getting Started with SharePoint Framework Development using TypeScript

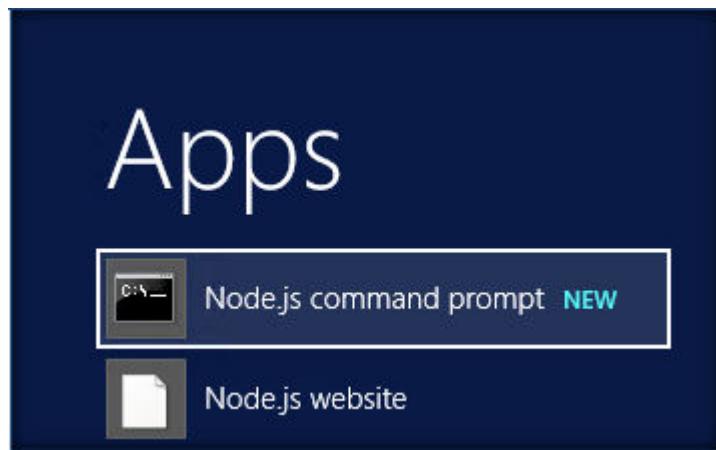
In the first section of the book we will be using TypeScript to build SharePoint Framework client webparts. As we proceed further, we will see how to use React JS and PnP JS to build the SPFx webparts

Create the First Hello World Client Web part

In this section, we will see how to create and deploy the first client web part using SharePoint Framework. We will be creating a Hello World client web part using TypeScript to understand the project structure and testing procedure.

Create the Web part project

Before moving forward, ensure that the SharePoint Framework development environment is ready. Spin up Node.js command prompt using which we will be creating the web part project structure.



We can create the directory where we would be adding the solution using the below command:
md ClientWebPart-HelloWorld

Let's move to the newly created working directory using the command:
cd ClientWebPart-HelloWorld

A screenshot of a "Node.js command prompt" window. The title bar says "Node.js command prompt". The window shows the following command-line session:

```
Your environment has been set up for using Node.js 6.10.0 (x64) and npm.  
C:\Users\farmaccount>npm -v  
4.3.0  
C:\Users\farmaccount>md ClientWebPart-HelloWorld  
C:\Users\farmaccount>cd ClientWebPart-HelloWorld  
C:\Users\farmaccount\ClientWebPart-HelloWorld>_
```

We will then create the client web part by running the Yeoman SharePoint Generator:
yo @microsoft/sharepoint

A screenshot of a command prompt window titled "yo". The title bar says "yo". The window shows the following command-line session:

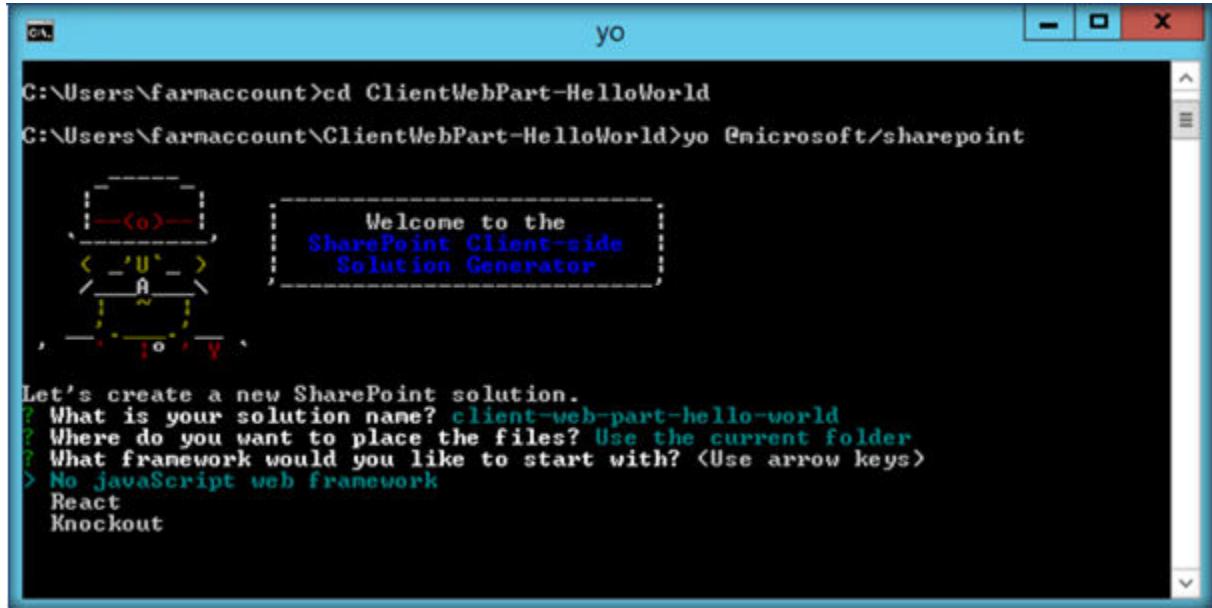
```
Your environment has been set up for using Node.js 6.10.0 (x64) and npm.  
C:\Users\farmaccount>npm -v  
4.3.0  
C:\Users\farmaccount>md ClientWebPart-HelloWorld  
C:\Users\farmaccount>cd ClientWebPart-HelloWorld  
C:\Users\farmaccount\ClientWebPart-HelloWorld>yo @microsoft/sharepoint
```

Below the command line, there is a graphical user interface for the Yeoman generator. It features a small terminal-like interface on the left with arrows for navigation, and a larger text area on the right with the title "Welcome to the SharePoint Client-side Solution Generator".

At the bottom of the window, the text "Let's create a new SharePoint solution." is displayed, followed by the question "What is your solution name? <client-web-part-hello-world> _".

This will display the prompt which we will have to fill up so as to proceed with project creation,

- What is your solution name? : Accept the default client-web-part-hello-world as your solution name and choose Enter.
- Where do you want to place your files : Use Current Folder
- What framework would you like to start with : Select “No javaScript web framework” for the time being as this is a sample web part



```
yo
C:\Users\farmaccount>cd ClientWebPart-HelloWorld
C:\Users\farmaccount\ClientWebPart-HelloWorld>yo @microsoft/sharepoint

Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? client-web-part-hello-world
? Where do you want to place the files? Use the current folder
? What framework would you like to start with? <Use arrow keys>
> No javascript web framework
  React
  Knockout
```

- What is your webpart name: Go on and press enter to accept the default Web part name as HelloWorld
- Go on and press enter to accept the default Web part description as HelloWorld description

Yeoman has started working on the creation of the project. It will install the required dependencies and scaffold the solution files for the HelloWorld web part which will take some time to complete. Once completed, we will get a Congratulations message.



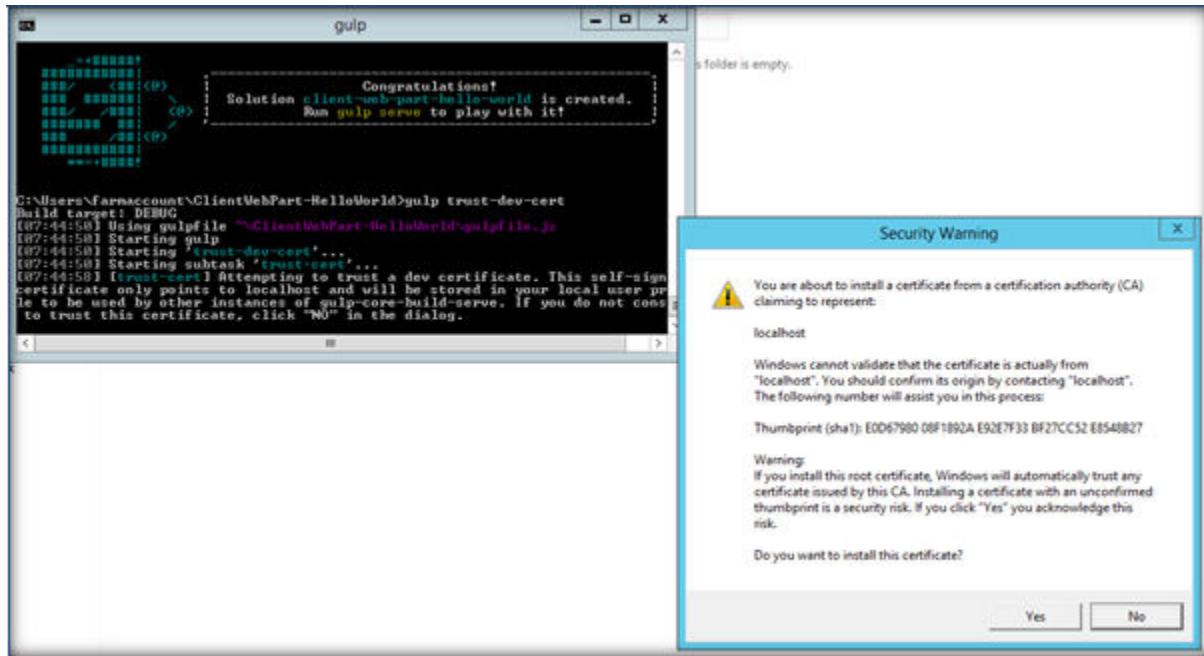
```
Node.js command prompt
`-- clone@0.2.0
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 <node_modules\chokidar\node_modules\fsevents>:
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} <current: {"os":"win32","arch":"x64"}>
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@* <node_modules\@microsoft\gulp-core-build-webpack\node_modules\fsevents>:
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} <current: {"os":"win32","arch":"x64"}>

          =+#+#+#+#
        #####(8) |   Congratulations!
      ##### /###(8) \ |   Solution client-web-part-hello-world is created.
      ##### /###(8) ; |   Run gulp serve to play with it!
      ##### /###(8)
      ######+#+#+#
      *=-+#+#+#+#
```

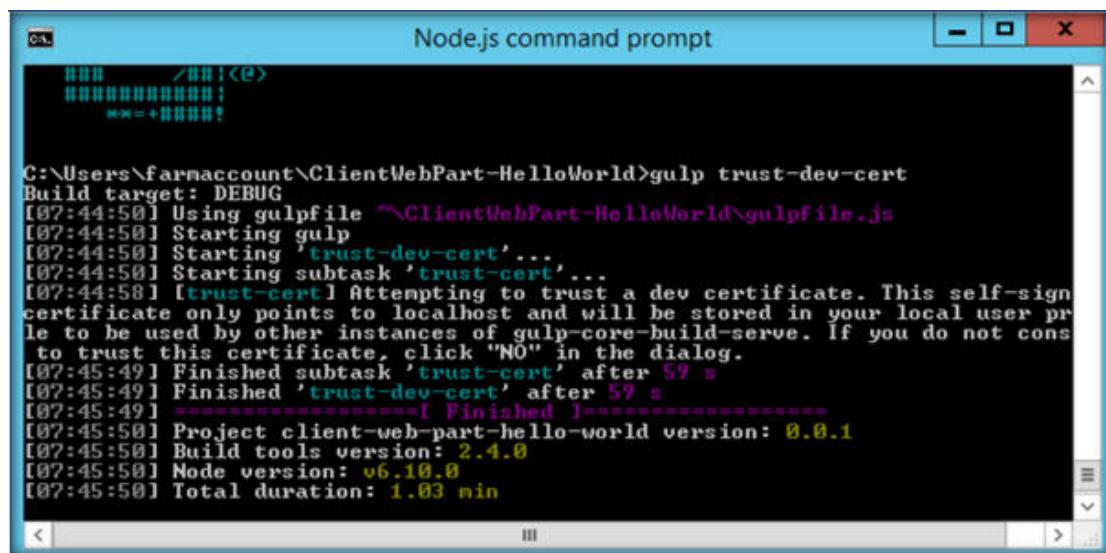
Test the web part

To test the client web part, we can build and run it on the local web server where we are developing the web part. SharePoint Framework development uses HTTPS endpoint by default. Since a default certificate is not configured for the local development environment, our browser will report a certificate error. The SharePoint Framework tool chain comes with a developer certificate that we can install for testing client web parts locally. From the current web part directory, run the below command:

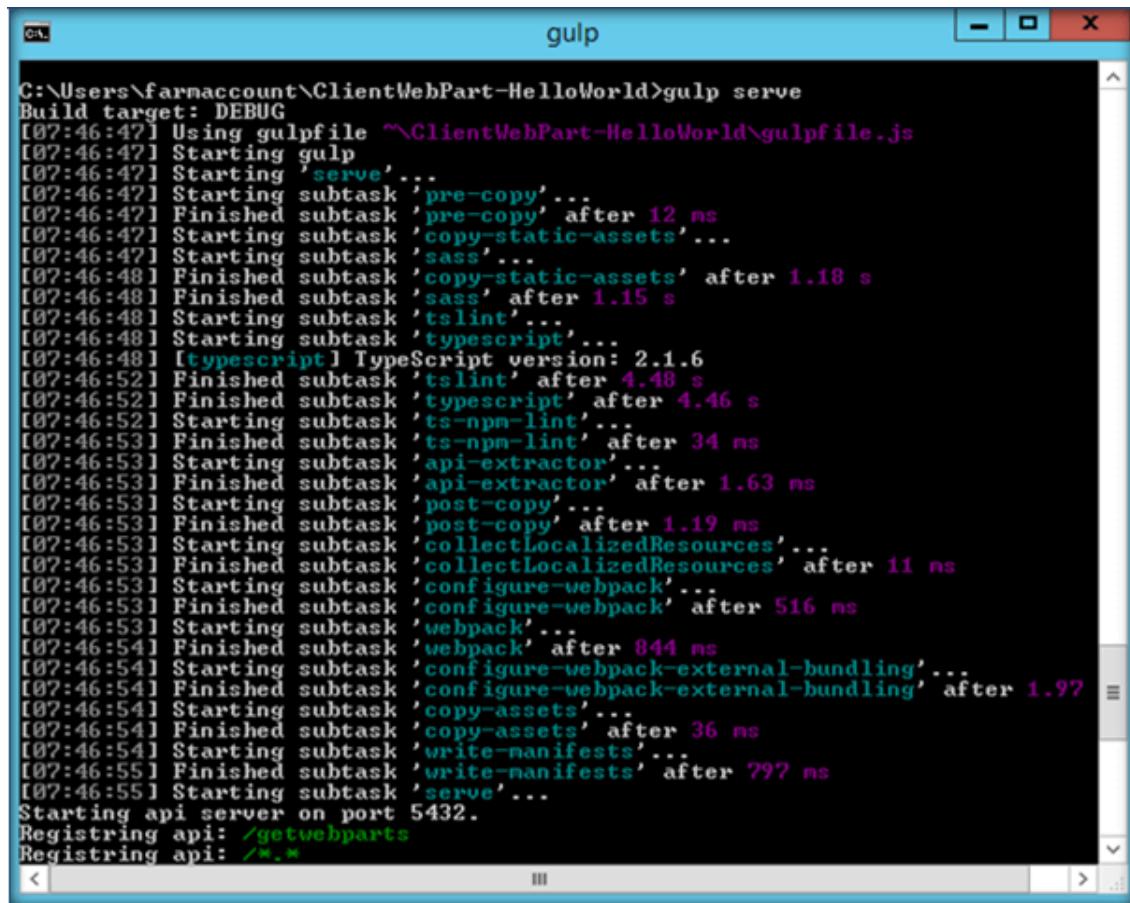
```
gulp trust-dev-cert
```



Click on Yes to install the certificate.



Now, let's preview the web part by running the `gulp serve` command. This command will execute a series of gulp tasks and will create a Node-based HTTPS server at 'localhost:4321'. It will then open the browser and display the client web part.

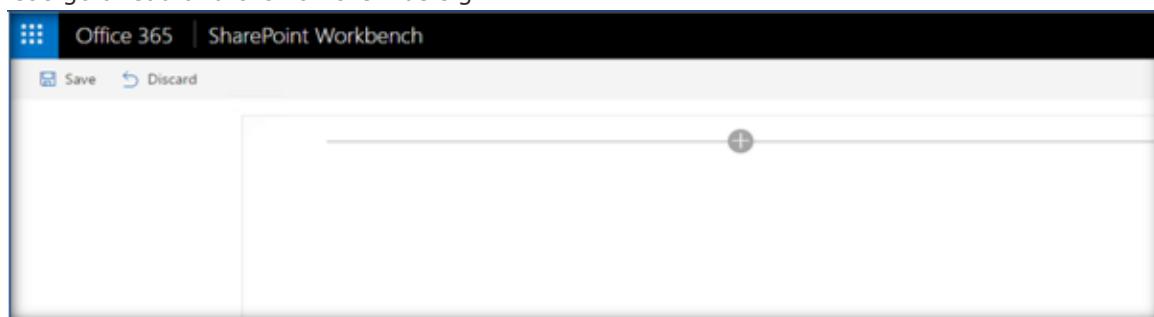


```
gulp
C:\Users\farmaccount\ClientWebPart-HelloWorld>gulp serve
Build target: DEBUG
[07:46:47] Using gulpfile ~\ClientWebPart-HelloWorld\gulpfile.js
[07:46:47] Starting 'gulp'
[07:46:47] Starting 'serve'...
[07:46:47] Starting subtask 'pre-copy'...
[07:46:47] Finished subtask 'pre-copy' after 12 ms
[07:46:47] Starting subtask 'copy-static-assets'...
[07:46:47] Starting subtask 'sass'...
[07:46:48] Finished subtask 'copy-static-assets' after 1.18 s
[07:46:48] Finished subtask 'sass' after 1.15 s
[07:46:48] Starting subtask 'tslint'...
[07:46:48] Starting subtask 'typescript'...
[07:46:48] [typescript] TypeScript version: 2.1.6
[07:46:52] Finished subtask 'tslint' after 4.48 s
[07:46:52] Finished subtask 'typescript' after 4.46 s
[07:46:52] Starting subtask 'ts-npm-lint'...
[07:46:53] Finished subtask 'ts-npm-lint' after 34 ms
[07:46:53] Starting subtask 'api-extractor'...
[07:46:53] Finished subtask 'api-extractor' after 1.63 ms
[07:46:53] Starting subtask 'post-copy'...
[07:46:53] Finished subtask 'post-copy' after 1.19 ms
[07:46:53] Starting subtask 'collectLocalizedResources'...
[07:46:53] Finished subtask 'collectLocalizedResources' after 11 ms
[07:46:53] Starting subtask 'configure-webpack'...
[07:46:53] Finished subtask 'configure-webpack' after 516 ms
[07:46:53] Starting subtask 'webpack'...
[07:46:54] Finished subtask 'webpack' after 844 ms
[07:46:54] Starting subtask 'configure-webpack-external-bundling'...
[07:46:54] Finished subtask 'configure-webpack-external-bundling' after 1.97 ms
[07:46:54] Starting subtask 'copy-assets'...
[07:46:54] Finished subtask 'copy-assets' after 36 ms
[07:46:54] Starting subtask 'write-manifests'...
[07:46:55] Finished subtask 'write-manifests' after 797 ms
[07:46:55] Starting subtask 'serve'...
Starting api server on port 5432.
Registering api: /getwebparts
Registering api: /*.*
```

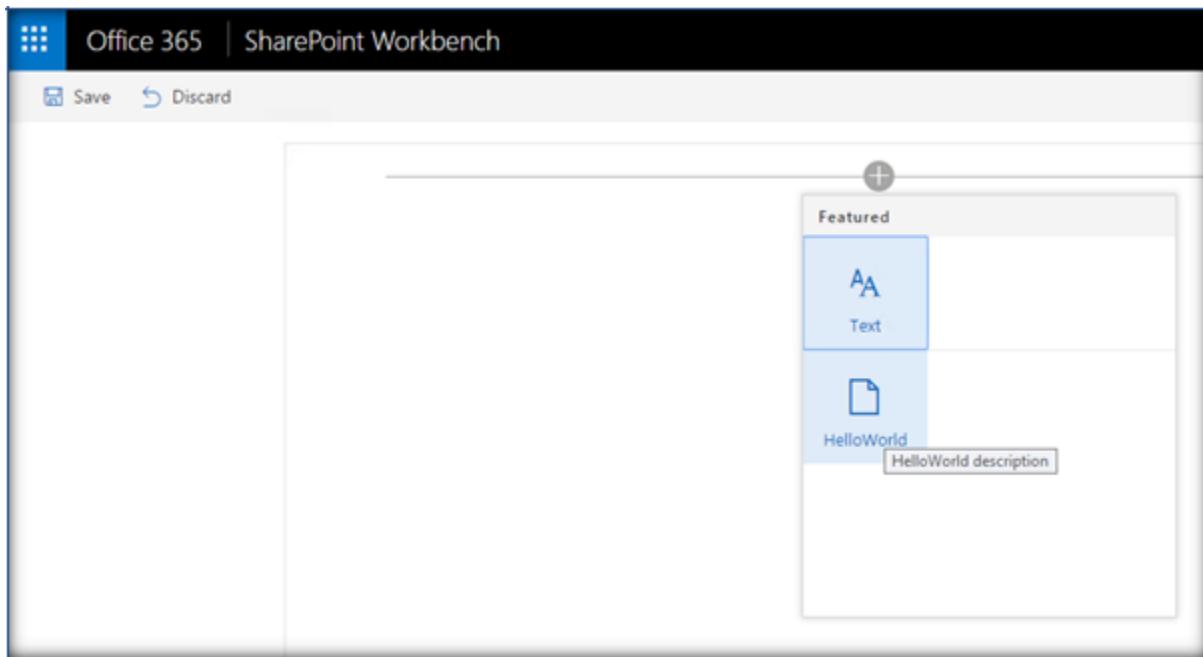
SharePoint Workbench

SharePoint Workbench is a developer design surface that enables us to test the developed client web parts without deploying them directly to SharePoint. It provides a client-side page to which we can add the created web parts.

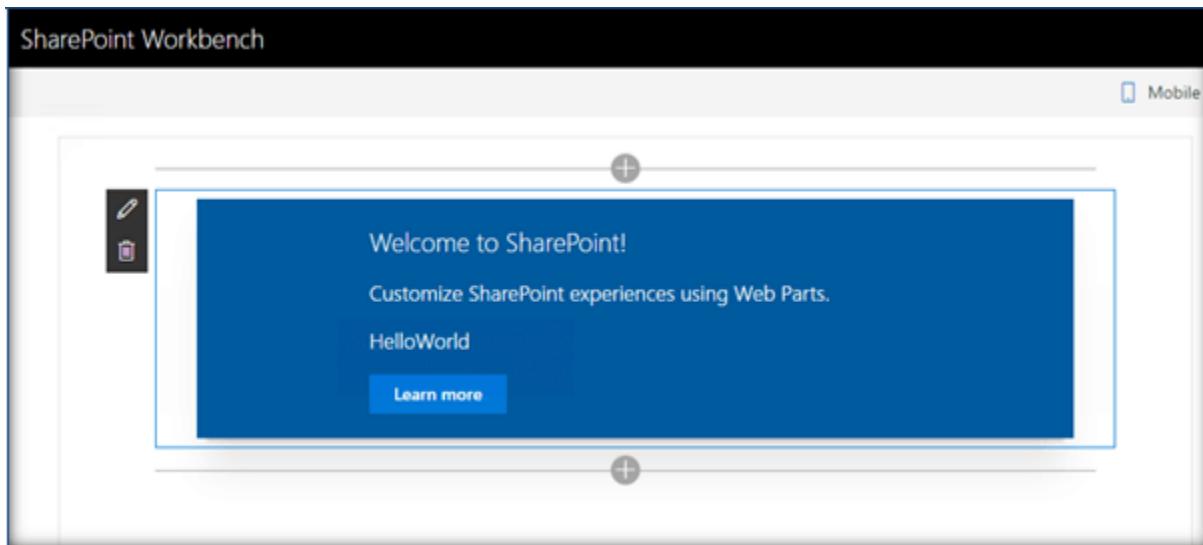
Thus, the SharePoint Workbench has opened in the browser but there are no visible web parts. So, let's go ahead and click on the Plus sign.



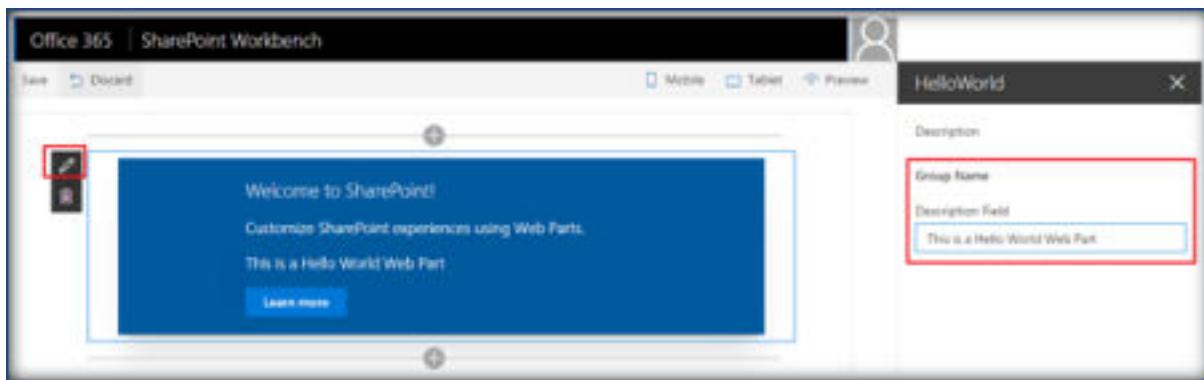
It will give us the option to add the Hello World web part that we have created recently.



On clicking it, the web part will be added to the page. The web part contains few custom messages.



We can edit the description property directly from the UI as shown below. However, if we want to edit this web part to add more details and functionality, we must go back and terminate the gulp server command.



To stop Gulp from listening to the process we can press 'Control + C'. This will terminate the Gulp Serve command and stop the server.

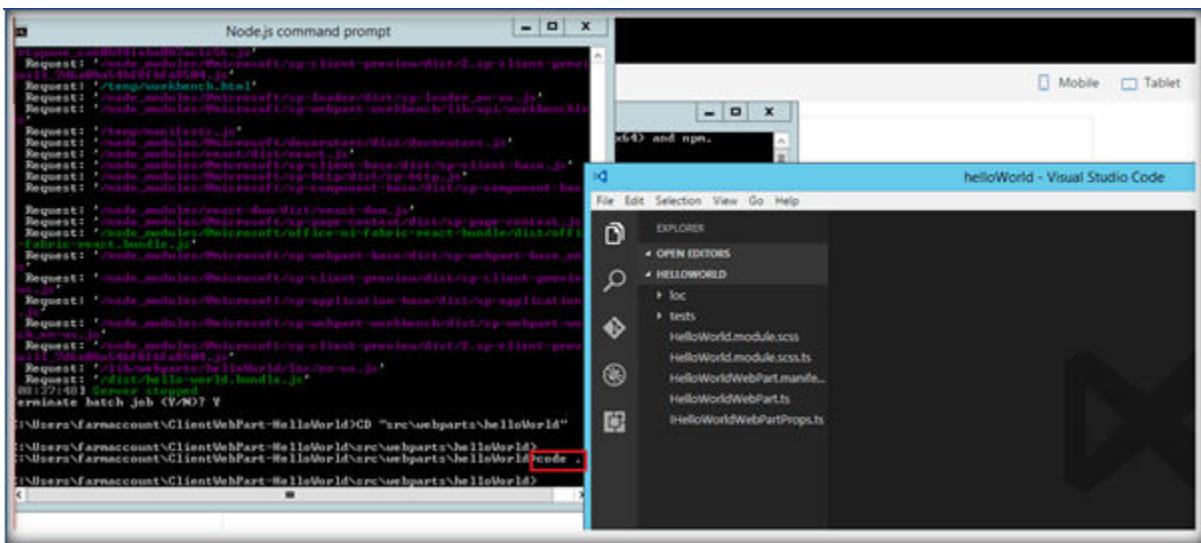
```
request: '/node_modules/@microsoft/sp-fabric-react/dist/fabric-react.bundle.js'
Request: '/node_modules/@microsoft/sp-webpart-base/dist/sp-webpart-base_en-us.js'
Request: '/node_modules/@microsoft/sp-client-preview/dist/sp-client-preview_en-us.js'
Request: '/node_modules/@microsoft/sp-application-base/dist/sp-application-base.js'
Request: '/node_modules/@microsoft/sp-webpart-workbench/dist/sp-webpart-workbench_en-us.js'
Request: '/node_modules/@microsoft/sp-client-preview/dist/2.sp-client-preview-quill_7d6a0ba54bf8fbfa8504.js'
Request: '/lib/webparts/helloWorld/loc/en-us.js'
Request: '/dist/hello-world.bundle.js'
[08:37:48] Server stopped
Terminate batch job <Y/N>? Y
C:\Users\farmaccount\ClientWebPart-HelloWorld>
```

Edit the web part

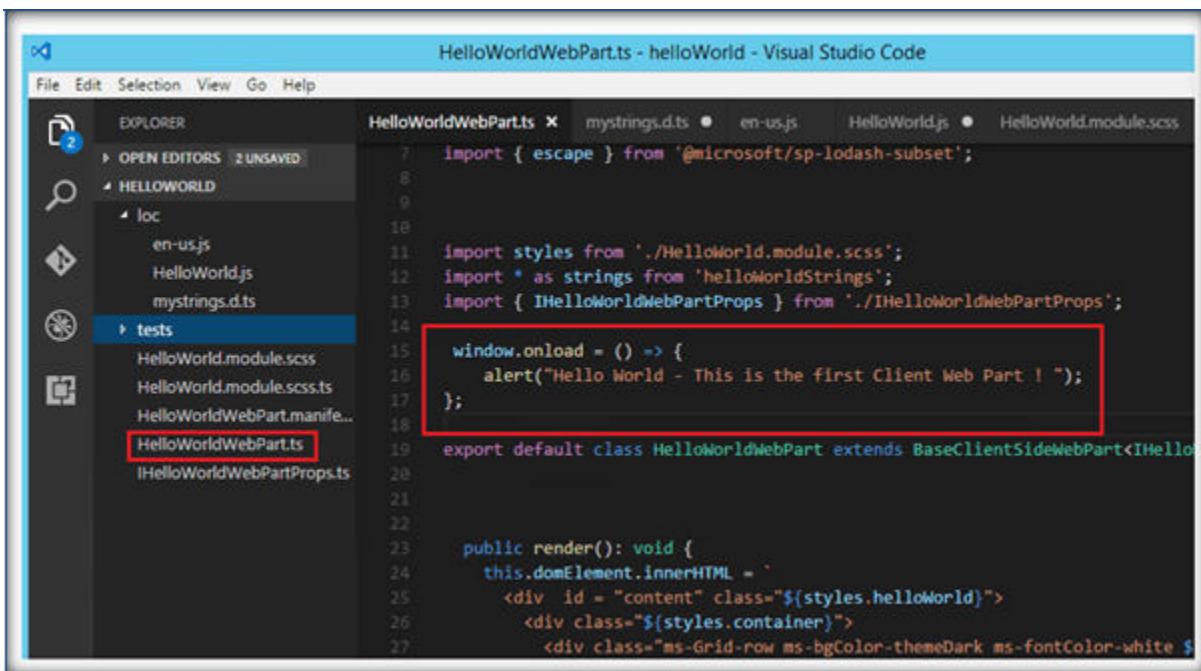
Now, let's try to edit the web part and add more functionality to it. To do that, navigate to 'src\webparts\helloWorld' location.

```
Request: '/node_modules/@microsoft/sp-client-preview/dist/2.sp-client-preview-quill_7d6a0ba54bf8fbfa8504.js'
Request: '/lib/webparts/helloWorld/loc/en-us.js'
Request: '/dist/hello-world.bundle.js'
[08:37:48] Server stopped
Terminate batch job <Y/N>? Y
C:\Users\farmaccount\ClientWebPart-HelloWorld>CD "src\webparts\helloWorld"
C:\Users\farmaccount\ClientWebPart-HelloWorld>
C:\Users\farmaccount\ClientWebPart-HelloWorld>
```

Run 'Code .' in the console which will open up the Visual Studio Code editor window.



In the left pane of Visual Studio Code, we can see the project structure. The bulk of the logic resides within the *HelloWorldWebPart.ts* file. Let's add JavaScript code to alert a message within this typescript file.



On clicking save Gulp will rebuild the web part project as shown below.

The terminal window shows the output of the 'gulp serve' command, detailing the execution of various subtasks including 'pre-copy', 'copy-static-assets', 'sass', 'tslint', 'typescript', 'post-copy', 'collectLocalizedResources', 'configure-webpack', and 'write-manifests'. The browser window displays the SharePoint Workbench interface with a modal dialog showing the message 'Hello World - This is the first Client Web Part!'.

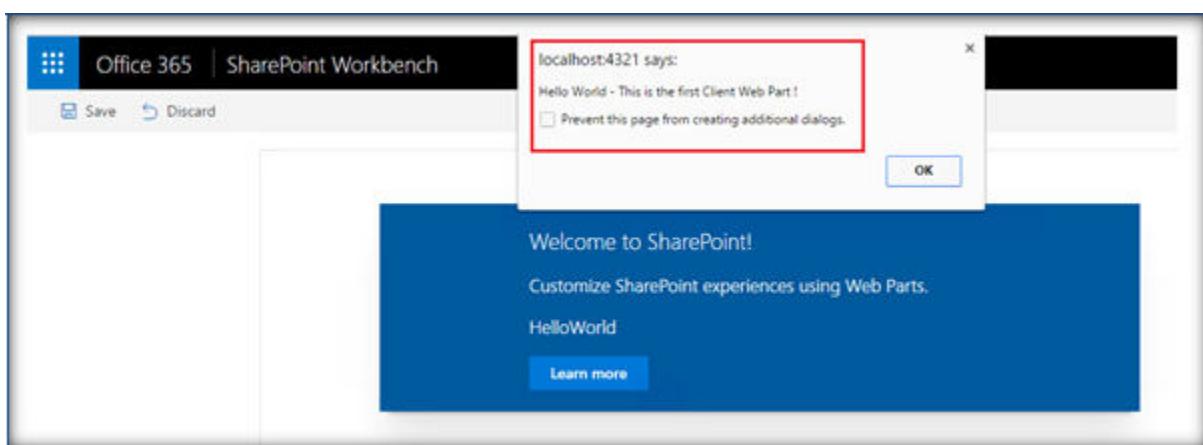
```

Request: '/dist/hello-world.bundle.js'
Request: '/node_modules/@microsoft/sp-application-base/dist/sp-application-base.js'
Request: '/node_modules/@microsoft/sp-webpart-workbench/dist/sp-webpart-workbench.js'
Request: '/lib/webparts/helloworld/icon/en-us.js'
Request: '/dist/hello-world.bundle.js'

[11:23:06] Starting subtask 'pre-copy'
[11:23:06] Finished subtask 'pre-copy' after 4.3 ms
[11:23:06] Starting subtask 'copy-static-assets'...
[11:23:06] Finished subtask 'sass'...
[11:23:06] Finished subtask 'sass' after 25 ms
[11:23:06] Starting subtask 'tslint'...
[11:23:06] Starting subtask 'typescript'...
[11:23:06] [typescript] TypeScript version: 2.1.6
[11:23:06] Finished subtask 'copy-static-assets' after 61 ms
[11:23:06] Finished subtask 'tslint' after 3.52 s
[11:23:06] Finished subtask 'typescript' after 3.51 s
[11:23:06] Starting subtask 'ts-npm-lint'...
[11:23:06] Finished subtask 'ts-npm-lint' after 15 ms
[11:23:06] Starting subtask 'api-extractor'...
[11:23:06] Finished subtask 'api-extractor' after 1.99 ms
[11:23:06] Starting subtask 'post-copy'...
[11:23:06] Finished subtask 'post-copy' after 894 ms
[11:23:06] Starting subtask 'collectLocalizedResources'...
[11:23:06] Finished subtask 'collectLocalizedResources' after 3.6 ms
[11:23:06] Starting subtask 'configure-webpack'...
[11:23:06] Finished subtask 'configure-webpack' after 2.76 ms
[11:23:06] Starting subtask 'webpack'...
[11:23:06] Finished subtask 'webpack' after 382 ms
[11:23:06] Starting subtask 'configure-webpack-external-bundling'...
[11:23:06] Finished subtask 'copy-assets'...
[11:23:06] Finished subtask 'copy-assets' after 10 ms
[11:23:06] Starting subtask 'write-manifests'...
[11:23:06] Finished subtask 'write-manifests' after 883 ms
[11:23:06] Starting subtask 'reload'...
[11:23:06] Finished subtask 'reload' after 4.06 ms

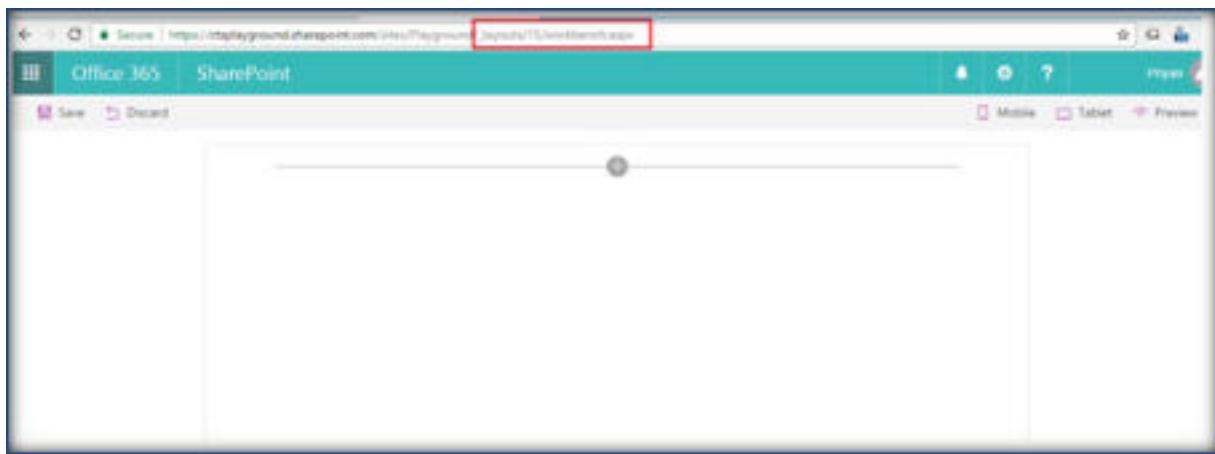
```

Again, running 'gulp serve' will display the updated web part in the browser. This time it will display the alert message as well.

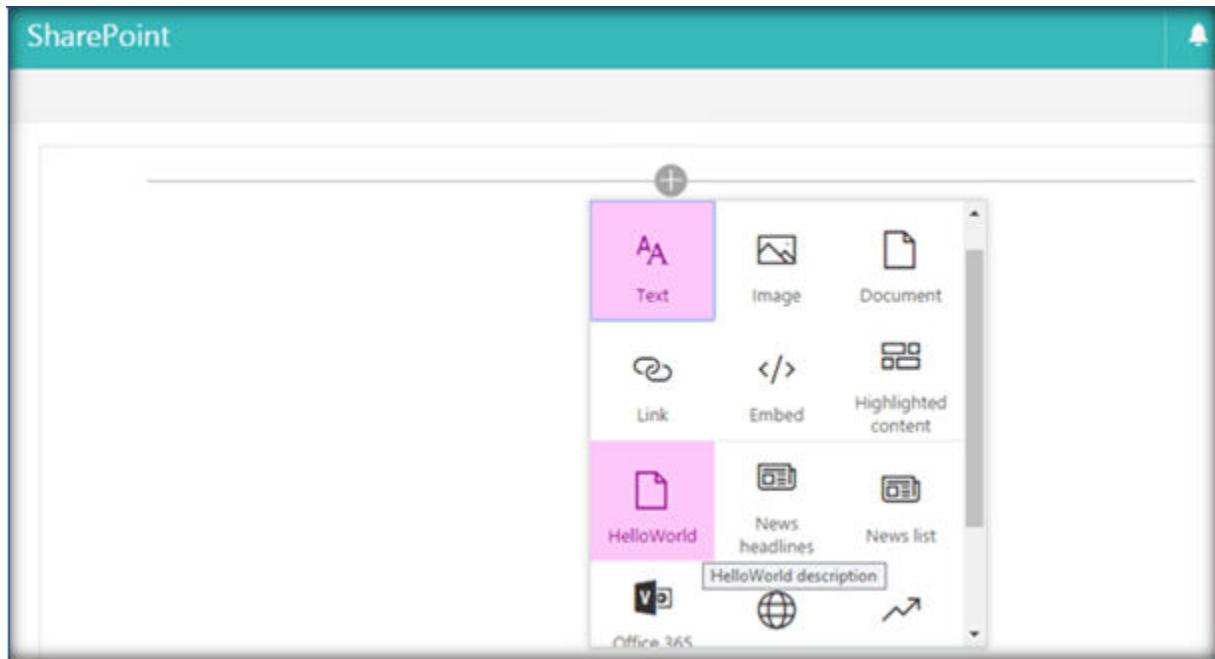


Add the web part to SharePoint

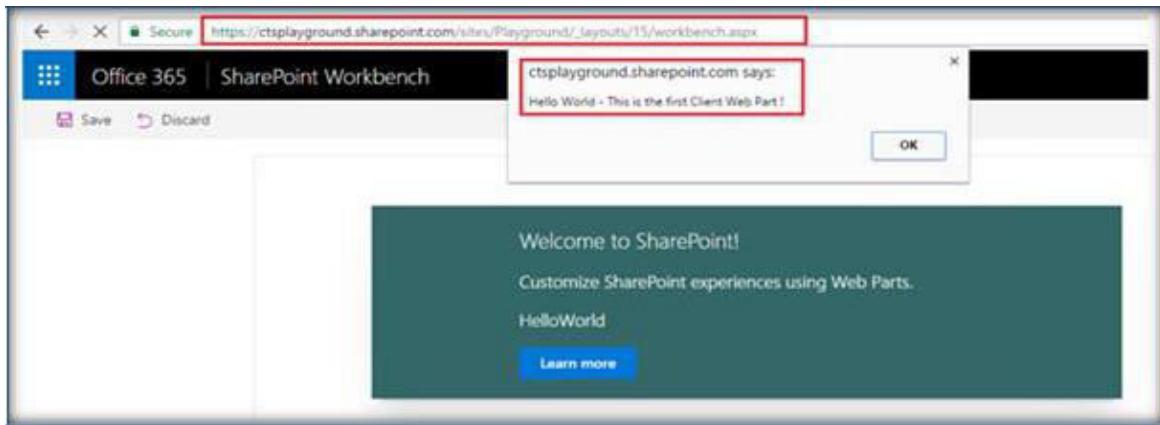
So far we were testing the web part in SharePoint Workbench locally, now let's try to test it within the SharePoint Context. SharePoint Workbench is also hosted in SharePoint Online to preview the web part. It can be accessed by adding ' _layouts/15/workbench.aspx' to the SharePoint Online URL.



Expand the Plus sign and add the Hello World web part.



The web part has triggered the alert message in the page indicating successful hosting of the web part within SharePoint.



Thus, we saw how to create a client web part using SharePoint Framework and test it within SharePoint Online.

Create SharePoint Framework Client Web Part to Retrieve and Display List Items

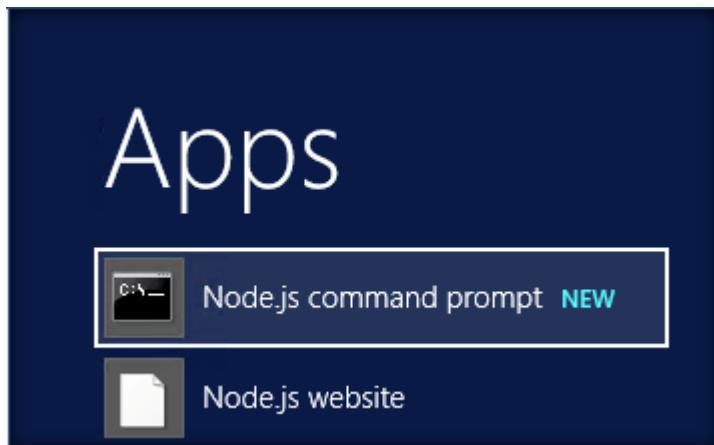
In this section, we will be creating a client Web part using TypeScript, which will be retrieving the list items from SharePoint List (EmployeeList) and display it in the tabular form in the client Web part, as shown below.

A screenshot of a SharePoint Framework client web part. The header bar is teal with the word "SharePoint". The main content area has a dark green header with the text "Welcome to SharePoint Framework Development" and "Demo : Retrieve Employee Data from SharePoint List". Below this is a table with a black header row labeled "Employee Details". The table contains the following data:

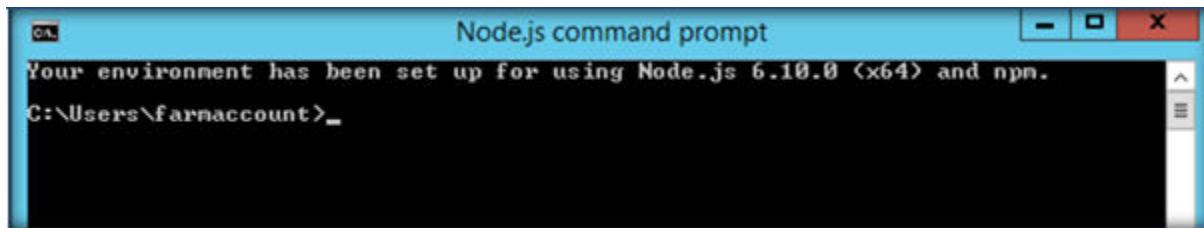
The Solutions files used in this section is zipped and uploaded to the Microsoft [TechNet gallery](#). Feel free to download it.

Create the Web part Project

Spin up Node.js command prompt, using which we will be creating the Web part project structure.

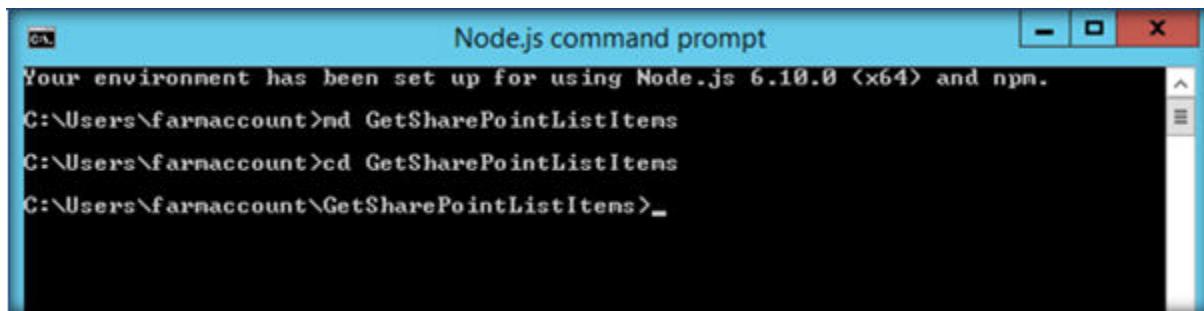


This will open the console where we can create the SharePoint Framework project structure.

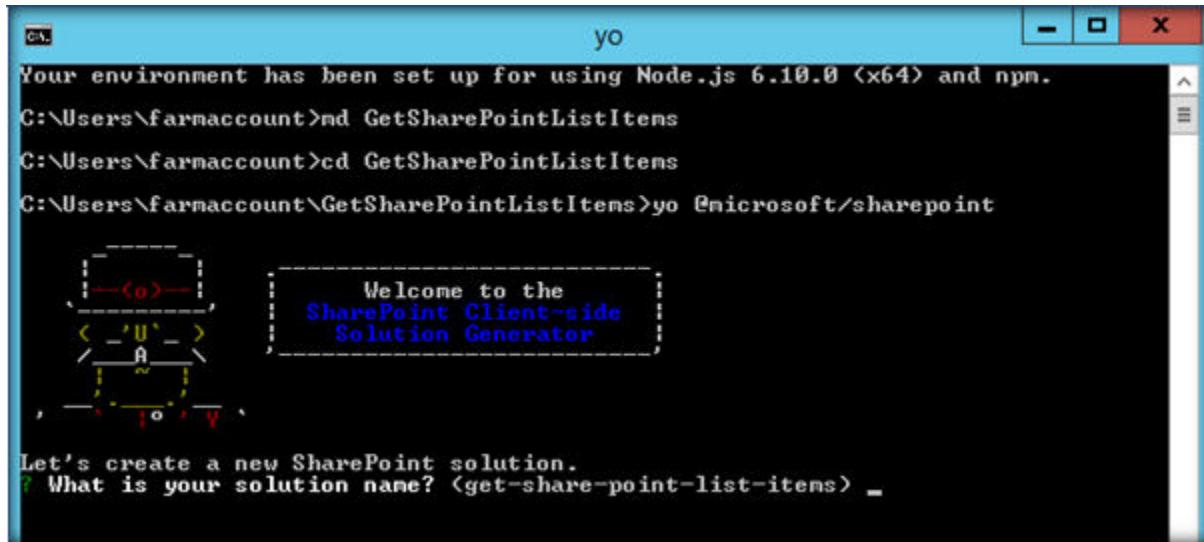


We can create the directory, where we will be adding the solution, using the command given below.
`md GetSharePointListItems`

Let's move to the newly created working directory, using the command.
`cd GetSharePointListItems`



We will then create the client Web part by running the Yeoman SharePoint Generator. `yo @microsoft/sharepoint`



```
yo
Your environment has been set up for using Node.js 6.10.0 (x64) and npm.
C:\Users\farmaccount>md GetSharePointListItems
C:\Users\farmaccount>cd GetSharePointListItems
C:\Users\farmaccount\GetSharePointListItems>yo @microsoft/sharepoint

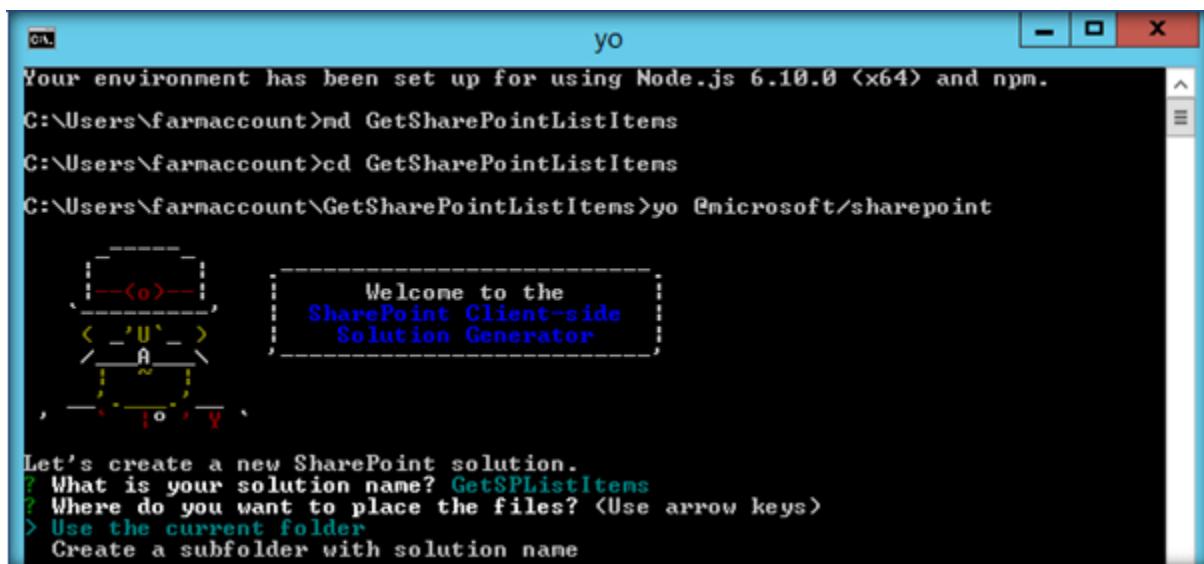
Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? <get-share-point-list-items> _
```

This will display the prompt, which we must fill up, to proceed with the project creation.

- What is your solution name? : Set it to 'GetSPListItems'.

On pressing enter, we will be asked to chose the working folder for the project.



```
yo
Your environment has been set up for using Node.js 6.10.0 (x64) and npm.
C:\Users\farmaccount>md GetSharePointListItems
C:\Users\farmaccount>cd GetSharePointListItems
C:\Users\farmaccount\GetSharePointListItems>yo @microsoft/sharepoint

Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? GetSPListItems
? Where do you want to place the files? <Use arrow keys>
> Use the current folder
Create a subfolder with solution name
```

- Where do you want to place your files- Use current folder.
- What framework would you like to start with- Select "No javaScript web framework" for the time being, as this is a sample Web part.
- What is your Webpart name- We will specify it as 'GetSPListItems' and press Enter
- What is your Webpart description- We will specify it as this Webpart will retrieve the list items from SharePoint list and display in a table

```
C:\Users\farnaccount>md GetSharePointListItems
C:\Users\farnaccount>cd GetSharePointListItems
C:\Users\farnaccount\GetSharePointListItems>yo @microsoft/sharepoint

Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? GetSPListItems
? Where do you want to place the files? Use the current folder
? What framework would you like to start with? No JavaScript web framework
A folder with solution name GetSPListItems will be created for you.
? What is your webpart name? GetSPListItems
? What is your webpart description? <GetSPListItems description> This WebPart w
ill retrieve list items from SharePoint List and display in a table
```

Yeoman has started working on the scaffolding of the project. It will install the required dependencies and scaffold the solution files for the 'GetListItems' Web part, which will take some time to complete. Once completed, we will get a congratulations message.

```
-- clone@0.2.0
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 <node_modules\chokidar\node_modules\fsevents>:
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} <current: {"os":"win32","arch":"x64"}>
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@* <node_modules\@microsoft\gulp-core-build-webpack\node_modules\fsevents>:
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} <current: {"os":"win32","arch":"x64"}>

=+=====
##<@>
## ##<@>
## / ##<@>
## / ##<@>
## =+=====

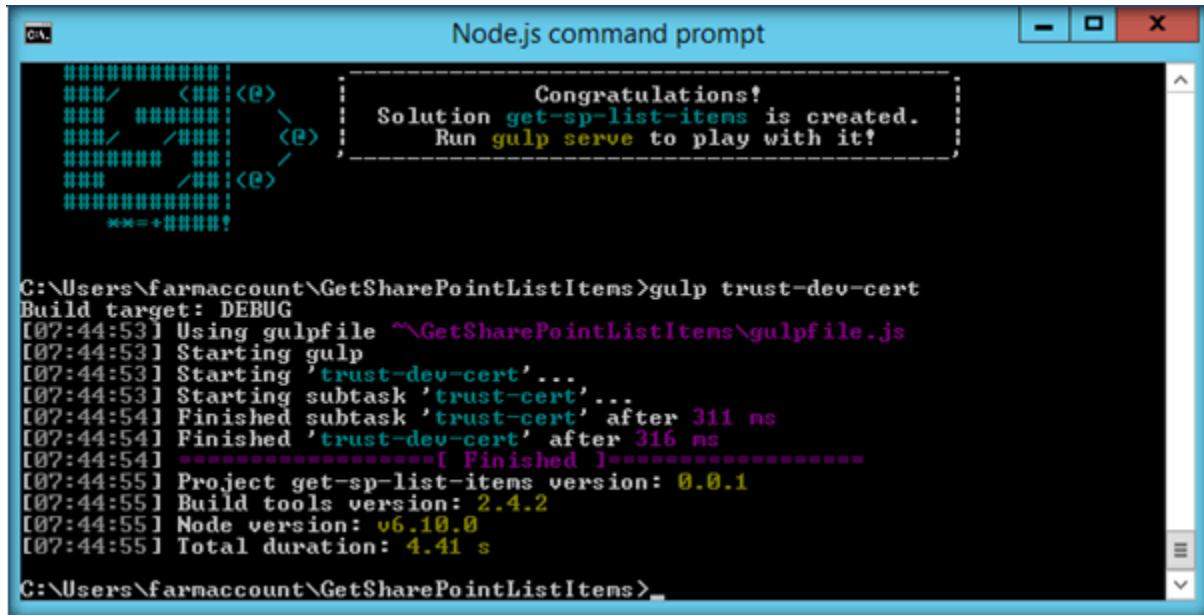
Congratulations!
Solution get-sp-list-items is created.
Run gulp serve to play with it!
```

Test the Web part locally

To test the client Web part, we can build and run it on the local Web Server, where we are developing the Web part. SharePoint Framework development uses HTTPS endpoint by default. Since a default certificate is not configured for the local development environment, our Browser will report a

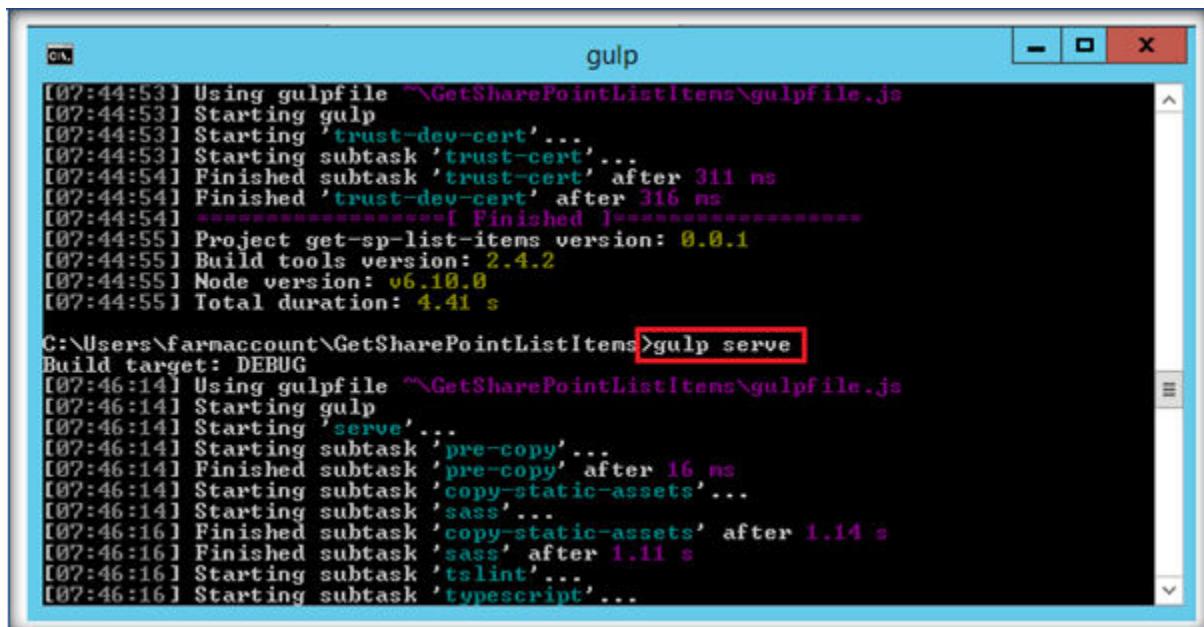
certificate error. SharePoint Framework tool chain comes with a developer certificate, which we can install for testing the client Web parts locally. From the current Web part directory, run the command given below.

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



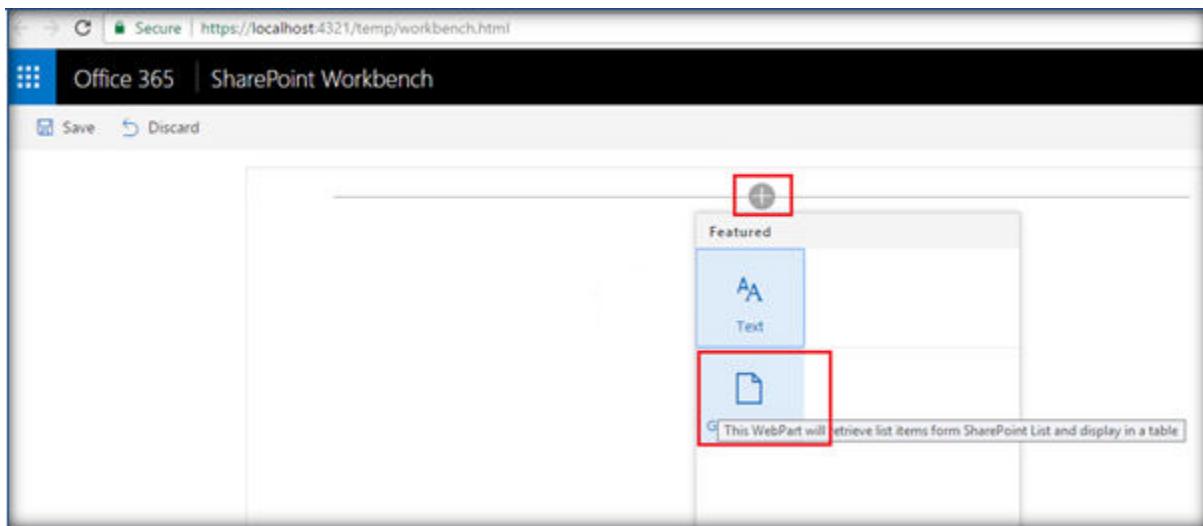
```
Node.js command prompt
C:\Users\farmaccount\GetSharePointListItems>gulp trust-dev-cert
Build target: DEBUG
[07:44:53] Using gulpfile ~\GetSharePointListItems\gulpfile.js
[07:44:53] Starting gulp
[07:44:53] Starting 'trust-dev-cert'...
[07:44:53] Starting subtask 'trust-cert'...
[07:44:54] Finished subtask 'trust-cert' after 311 ms
[07:44:54] Finished 'trust-dev-cert' after 316 ms
[07:44:54] [ Finished ]
[07:44:55] Project get-sp-list-items version: 0.0.1
[07:44:55] Build tools version: 2.4.2
[07:44:55] Node version: v6.10.0
[07:44:55] Total duration: 4.41 s
C:\Users\farmaccount\GetSharePointListItems>
```

Now, let's preview the Web part by running the *gulp serve* command.

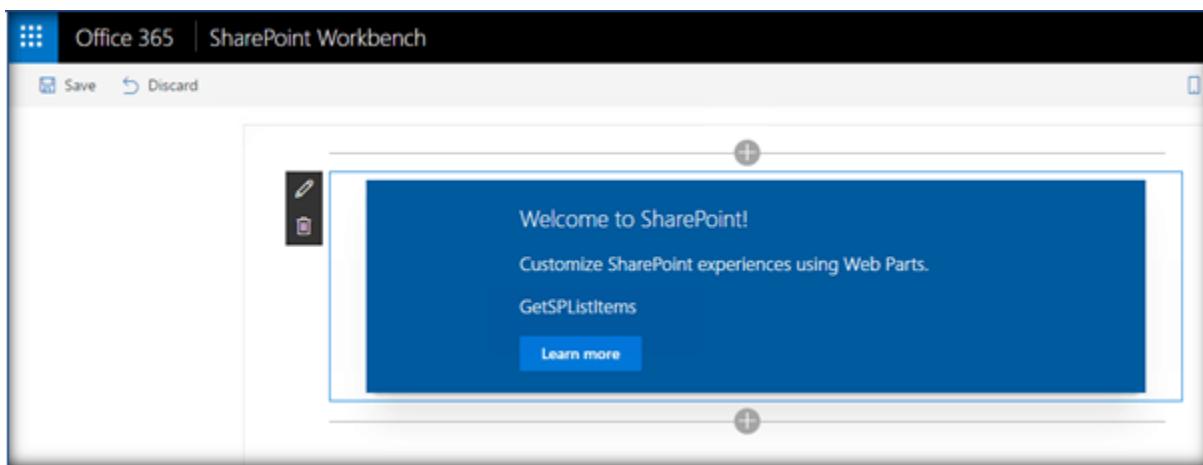


```
Node.js command prompt
C:\Users\farmaccount\GetSharePointListItems>gulp serve
Build target: DEBUG
[07:46:14] Using gulpfile ~\GetSharePointListItems\gulpfile.js
[07:46:14] Starting gulp
[07:46:14] Starting 'serve'...
[07:46:14] Starting subtask 'pre-copy'...
[07:46:14] Finished subtask 'pre-copy' after 16 ms
[07:46:14] Starting subtask 'copy-static-assets'...
[07:46:14] Starting subtask 'sass'...
[07:46:16] Finished subtask 'sass' after 1.14 s
[07:46:16] Starting subtask 'tslint'...
[07:46:16] Starting subtask 'typescript'...
```

This command will execute a series of gulp tasks and will create a Node-based HTTPS Server at 'localhost:4321'. It will then open the Browser and display the client Web part.



This indicates that the project structure is set up correctly. We will now open the solution in Visual Studio Code to add the logic to retrieve the list items from SharePoint and display it as a table in this page.



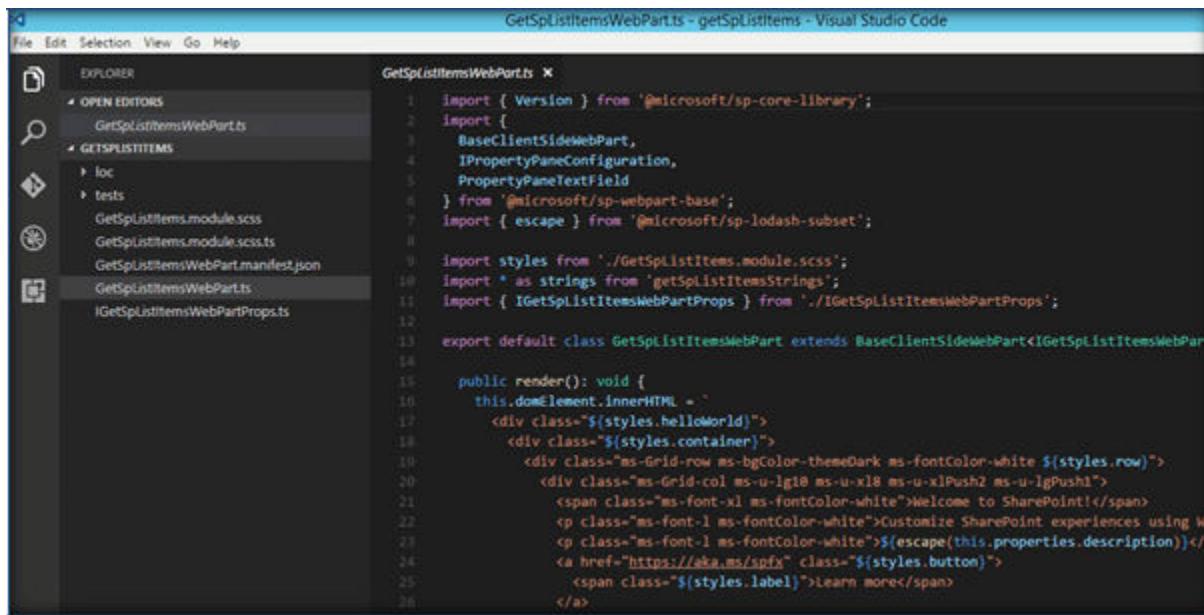
To stop Gulp from listening to the process, we can press '*Control + C*'. This will terminate the Gulp Serve command and stop the Server.

Edit the web part

Now let's try to edit the web part and add more functionality to it. To do that navigate to "src\webparts\getSpListItems" location.

```
C:\Users\farmaccount\GetSharePointListItems>CD "src\webparts\getSpListItems"
C:\Users\farmaccount\GetSharePointListItems\src\webparts\getSpListItems>
C:\Users\farmaccount\GetSharePointListItems\src\webparts\getSpListItems>code .
C:\Users\farmaccount\GetSharePointListItems\src\webparts\getSpListItems>_
```

In the left pane of Visual Studio Code, we can see the project structure. The bulk of the logic resides within the `GetSPListItemsWebPart.ts` file. Let's add the code to retrieve SharePoint list items from the Employee List within this TypeScript file.



The screenshot shows the Visual Studio Code interface with the title bar "GetSPListItemsWebPart.ts - getSPListItems - Visual Studio Code". The Explorer sidebar on the left shows the project structure with files like `GetSPListItemsWebPart.ts`, `GetSPListItems.module.scss`, and `GetSPListItemsWebPart.manifest.json`. The main editor area displays the `GetSPListItemsWebPart.ts` file content, which includes imports for SharePoint libraries and a render method that outputs a specific HTML template.

```
GetSPListItemsWebPart.ts
File Edit Selection View Go Help
EXPLORER
OPEN EDITORS
GetSPListItemsWebPart.ts
GETSPLISTITEMS
  loc
  tests
  GetSPListItems.module.scss
  GetSPListItems.module.scss.ts
  GetSPListItemsWebPart.manifest.json
  GetSPListItemsWebPart.ts
  IGetSPListItemsWebPartProps.ts
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 styles from './GetSPListItems.module.scss';
10 import * as strings from 'getSPListItemsStrings';
11 import { IGetSPListItemsWebPartProps } from './IGetSPListItemsWebPartProps';
12
13 export default class GetSPListItemsWebPart extends BaseClientSideWebPart<IGetSPListItemsWebPartProps> {
14
15   public render(): void {
16     this.domElement.innerHTML =
17       <div class="${styles.helloWorld}>
18         <div class="${styles.container}>
19           <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}>
20             <div class="ms-Grid-col ms-u-lg10 ms-u-xlg8 ms-u-xlpush2 ms-u-lgPush1">
21               <span class="ms-font-xl ms-fontColor-white">Welcome to SharePoint!</span>
22               <p class="ms-font-l ms-fontColor-white">Customize SharePoint experiences using Web Parts.</p>
23               <p class="ms-font-l ms-fontColor-white">${escape(this.properties.description)}</p>
24               <a href="https://aka.ms/spfy" class="${styles.button}">
25                 <span class="${styles.label}">Learn more</span>
26               </a>
27           </div>
28         </div>
29       </div>;
30   }
31 }
```

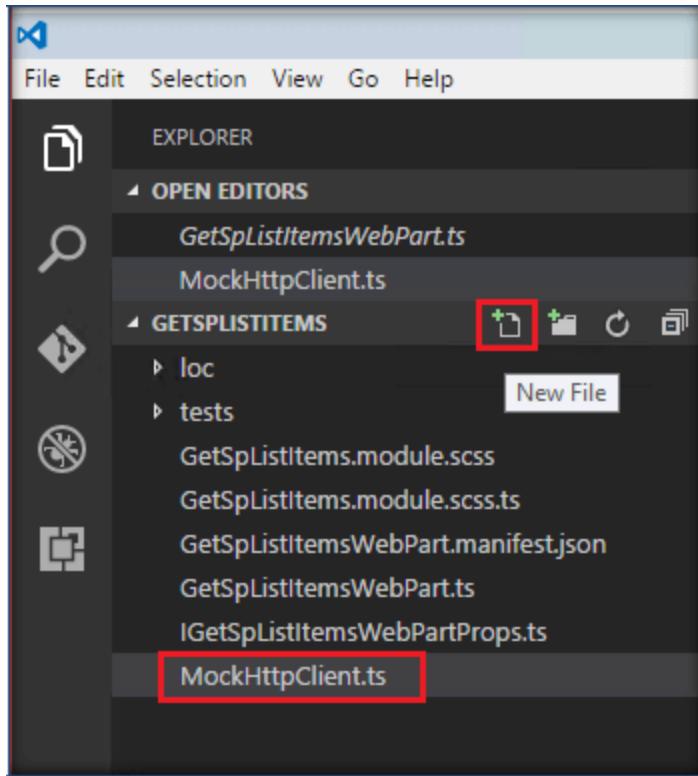
Define List Model

Since we want to retrieve an Employee list items data, we will be creating list model with SharePoint list fields in the `GetSPListItemsWebPart.TS` file, as shown below. Place it above the '`GetSPListItemsWebPart`' class.

```
1. export interface ISPLists {
2.   value: ISPList[];
3. }
4. export interface ISPList {
5.   EmployeeId: string;
6.   EmployeeName: string;
7.   Experience: string;
8.   Location: string;
9. }
```

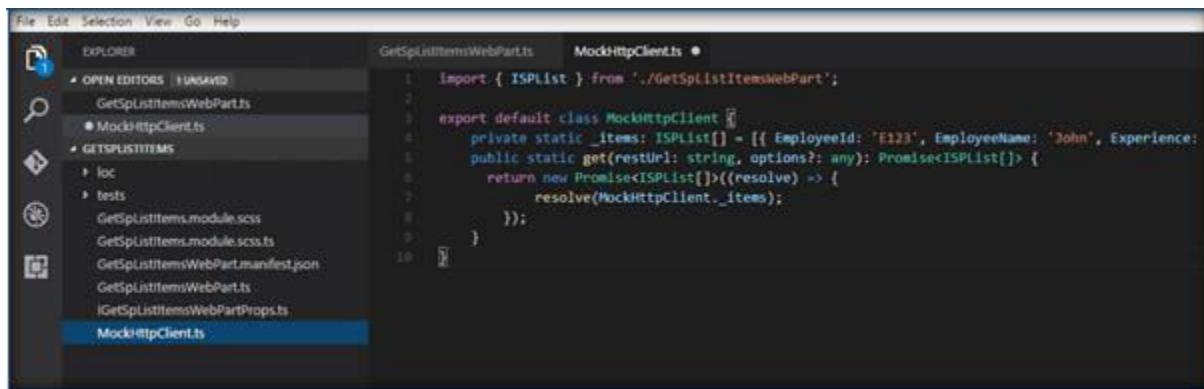
Create Mock HttpClient to test data locally

To test the list item retrieval in the local workbench, we will create a mock store, which returns mock Employee list data. We will create a new file inside 'src\webparts\ getSPListItems' folder named `MockHttpClient.ts`, as shown below.



We will then copy the code given below into MockHttpClient.ts, as shown below.

```
1. import { ISPList } from './GetSpListItemsWebPart';
2.
3. export default class MockHttpClient {
4.     private static _items: ISPList[] = [{ EmployeeId: 'E123', EmployeeName: 'John',
5.     Experience: 'SharePoint', Location: 'India' }];
6.     public static get(restUrl: string, options?: any): Promise<ISPList[]> {
7.         return new Promise<ISPList[]>((resolve) => {
8.             resolve(MockHttpClient._items);
9.         });
10.    }
```



We can now use the MockHttpClient class in the 'GetSPListItems' class. Let's import the 'MockHttpClient' module by going to the GetSpLitItemsWebPart.ts and pasting the line given below just after "*import { IGetSPListItemsWebPartProps } from './IGetSPListItemsWebPartProps';*"

```
1. import MockHttpClient from './MockHttpClient';
```

We will also add the mock list item retrieval method within the 'GetSPListItemsWebPart' class.

```
1. private _getMockListData(): Promise<ISPLists> {
2.   return MockHttpClient.get(this.context.pageContext.web.absoluteUrl).then(() => {
3.     const listData: ISPLists = {
4.       value:
5.       [
6.         { EmployeeId: 'E123', EmployeeName: 'John', Experience:
7.           'SharePoint', Location: 'India' },
8.           { EmployeeId: 'E567', EmployeeName: 'Martin', Experience:
9.             '.NET', Location: 'Qatar' },
10.             { EmployeeId: 'E367', EmployeeName: 'Luke', Experience:
11.               'JAVA', Location: 'UK' }
12.         ];
13.       };
14.     return listData;
15.   }) as Promise<ISPLists>;
16. }
```

Retrieve SharePoint List Items

SharePoint Framework has the helper class spHttpClient, which can be utilized to call REST API requests against SharePoint. We will use REST API: "*"/_api/web/lists/GetByTitle('EmployeeList')/Items*" to get the list items from SharePoint List.

To use 'spHttpClient', we will first have to import it from the '@microsoft/sp-http' module. We will import this module by placing the line given below after the mockHttpClient import code - "*import MockHttpClient from './MockHttpClient';*"

```
import {
  SPHttpClient
} from '@microsoft/sp-http';
```

We will be then adding the method given below to get SharePoint list items, using REST API within the 'GetSPListItemsWebPart' class.

```
1. private _getListData(): Promise<ISPLists> {
2.   return this.context.spHttpClient.get(this.context.pageContext.web.absoluteUrl +
  `/api/web/lists/GetByTitle('EmployeeList')/Items`, SPHttpClient.configurations.v1)
3.     .then((response: Response) => {
4.       debugger;
5.       return response.json();
6.     });
7. }
```

Render the SharePoint List Items from Employee List

Once we run the `gulp serve` command, we can test the Web part in SharePoint Workbench in the local environment or using SharePoint Online Context. SharePoint Framework uses '`EnvironmentType`' module to identify the environment, where the Web part is executed. In order to implement this, we will import '`Environment`' and the '`EnvironmentType`' modules from the `@microsoft/sp-core-library` bundle by placing it at the top of the `GetSpListItemsWebpart.ts` file.

```
import {
  Environment,
  EnvironmentType
} from '@microsoft/sp-core-library';
```

We will then check `Environment.type` value and if it is equal to `Environment.Local`, the `MockHttpClient` method, which returns dummy data which will be called else the method that calls REST API which can retrieve SharePoint list items will be called.

```
1. private _renderListAsync(): void {
2.
3.   if (Environment.type === EnvironmentType.Local) {
4.     this._getMockListData().then((response) => {
5.       this._renderList(response.value);
6.     });
7.   } else {
8.     this._getListData()
9.       .then((response) => {
10.         this._renderList(response.value);
11.       });
12.   }
13. }
14. }
```

Finally, we will add the method given below, which will create HTML table out of the retrieved SharePoint list items.

```
1. private _renderList(items: ISPList[]): void {
2. let html: string = '<table class="TFtable" border=1 width=100% style="border-
  collapse: collapse;">';
3. html +=
  `<th>EmployeeId</th><th>EmployeeName</th><th>Experience</th><th>Location</th>`;
4. items.forEach((item: ISPList) => {
5.   html += `
6.     <tr>
7.       <td>${item.EmployeeId}</td>
8.       <td>${item.EmployeeName}</td>
9.       <td>${item.Experience}</td>
10.      <td>${item.Location}</td>
11.      </tr>
12.    `;
13. });
14. html += `</table>`;
15. const listContainer: Element = this.domElement.querySelector('#spListContainer');
16. listContainer.innerHTML = html;
```

```
17. }
```

To enable rendering of the list items given above, we will replace Render method in the 'GetSPListsWebPart' class with the code given below.

```
1. public render(): void {
2.     this.domElement.innerHTML = `
3.     <div class="${styles.helloWorld}">
4.       <div class="${styles.container}">
5.         <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
6.           <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
7.             <span class="ms-font-xl ms-fontColor-white" style="font-size:28px">Welcome to
8.               SharePoint Framework Development</span>
9.         <p class="ms-font-l ms-fontColor-white" style="text-align: center">Demo :
10.            Retrieve Employee Data from SharePoint List</p>
11.        </div>
12.      </div>
13.      <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
14.        <div style="background-color:Black;color:white;text-align: center;font-weight:
15.          bold;font-size:18px;">Employee Details</div>
16.        <br>
17.      </div>
18.    </div>;
19.    this._renderListAsync();
20.  }
```

TS File Contents

The code contents used in the TS file to retrieve and display list items are given below:

```
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 {
10.   Environment,
11.   EnvironmentType
12. } from '@microsoft/sp-core-library';
13.
14.
15. import styles from './GetSPLists.module.scss';
16. import * as strings from 'getSPListsStrings';
17. import { IGetSPListsWebPartProps } from './IGetSPListsWebPartProps';
18. import MockHttpClient from './MockHttpClient';
19.
20. import {
21.   SPHttpClient
22. } from '@microsoft/sp-http';
23.
24.
25. export interface ISPLists {
```

```

26.         value: ISPList[];
27.     }
28.     export interface ISPList {
29.         EmployeeId: string;
30.         EmployeeName: string;
31.         Experience: string;
32.         Location: string;
33.     }
34.
35. export default class GetSpListItemsWebPart extends
36.     BaseClientSideWebPart<IGetSpListItemsWebPartProps> {
37.
38.     private _getListData(): Promise<ISPLists> {
39.         return this.context.spHttpClient.get(this.context.pageContext.web.absoluteUrl +
40.             `/api/web/lists/GetByTitle('EmployeeList')/Items`, SPHttpClient.configurations.v1)
41.             .then((response: Response) => {
42.                 debugger;
43.                 return response.json();
44.             });
45.
46.     private _renderListAsync(): void {
47.         if (Environment.type === EnvironmentType.Local) {
48.             this._getMockListData().then((response) => {
49.                 this._renderList(response.value);
50.             });
51.         } else {
52.             this._getListData()
53.                 .then((response) => {
54.                     this._renderList(response.value);
55.                 });
56.         }
57.     }
58. }
59.
60. private _renderList(items: ISPList[]): void {
61. let html: string = `<table class="TFtable" border=1 width=100% style="border-
62. collapse: collapse;">';
63.     items.forEach((item: ISPList) => {
64.         html += `
65.             <tr>
66.                 <td>${item.EmployeeId}</td>
67.                 <td>${item.EmployeeName}</td>
68.                 <td>${item.Experience}</td>
69.                 <td>${item.Location}</td>
70.             </tr>
71.             `;
72.     });
73.     html += `</table>`;
74.     const listContainer: Element = this.domElement.querySelector('#spListContainer');
75.     listContainer.innerHTML = html;
76. }
77.
78. public render(): void {
79.     this.domElement.innerHTML =
80.         <div class="${styles.helloWorld}">
81.             <div class="${styles.container}">
82.                 <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">

```

```
83.      <div class="ms-Grid-col ms-u-lg10 ms-u-xl18 ms-u-xlPush2 ms-u-lgPush1">
84.          <span class="ms-font-xl ms-fontColor-white" style="font-size:28px">Welcome to
SharePoint Framework Development</span>
85.
86.      <p class="ms-font-l ms-fontColor-white" style="text-align: center">Demo :
Retrieve Employee Data from SharePoint List</p>
87.      </div>
88.  </div>
89.  <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
90. <div style="background-color:Black;color:white;text-align: center;font-weight:
bold;font-size:18px;">Employee Details</div>
91.  <br>
92. <div id="spListContainer" />
93.  </div>
94. </div>
95. </div>';
96. this._renderListAsync();
97. }
98.
99. private _getMockListData(): Promise<ISPLists> {
100.         return MockHttpClient.get(this.context.pageContext.web.absoluteUrl).then(() => {
101.             const listData: ISPLists = {
102.                 value:
103.                     [
104.                         { EmployeeId: 'E123', EmployeeName: 'John', Experience:
'SharePoint', Location: 'India' },
105.                         { EmployeeId: 'E567', EmployeeName: 'Martin', Experience:
'.NET', Location: 'Qatar' },
106.                         { EmployeeId: 'E367', EmployeeName: 'Luke', Experience:
'JAVA', Location: 'UK' }
107.                     ]
108.                 };
109.             return listData;
110.         }) as Promise<ISPLists>;
111.     }
112.
113.     protected getDataVersion(): Version {
114.         return Version.parse('1.0');
115.     }
116.
117.     protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
118.         return {
119.             pages: [
120.                 {
121.                     header: {
122.                         description: strings.PropertyPaneDescription
123.                     },
124.                     groups: [
125.                         {
126.                             groupName: strings.BasicGroupName,
127.                             groupFields: [
128.                                 PropertyPaneTextField('description', {
129.                                     label: strings.DescriptionFieldLabel
130.                                 })
131.                             ]
132.                         }
133.                     ]
134.                 }
135.             ]
136.         };

```

```
137.      }
138. }
```

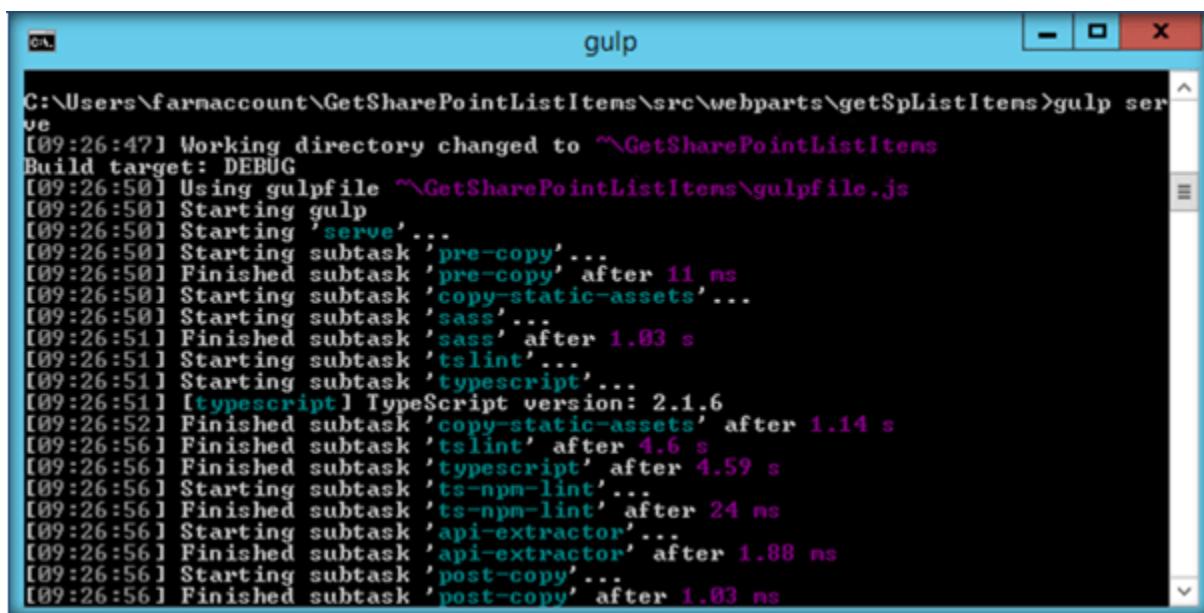
Mock HTTP Client Content

The mock http client content used to test in the local workbench is as follows:

```
1. import { ISPList } from './GetSpListItemsWebPart';
2.
3. export default class MockHttpClient {
4.   private static _items: ISPList[] = [{ EmployeeId: 'E123', EmployeeName: 'John',
5.   Experience: 'SharePoint', Location: 'India' },];
6.   public static get(restUrl: string, options?: any): Promise<ISPList[]> {
7.     return new Promise<ISPList[]>((resolve) => {
8.       resolve(MockHttpClient._items);
9.     });
10. }
```

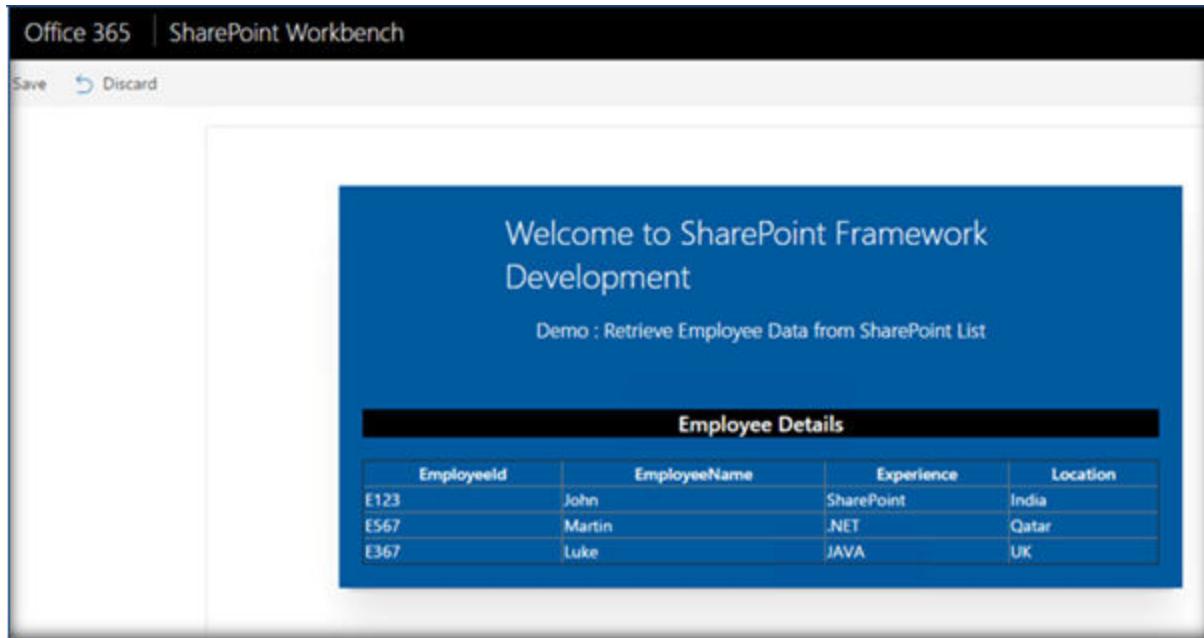
Test the Web part in local SharePoint Workbench

Now, we can see the output generated in the local SharePoint Workbench by running *gulp serve* command.



```
C:\Users\farmaccount\GetSharePointListItems\src\webparts\getSpListItems>gulp serve
[09:26:47] Working directory changed to '^\GetSharePointListItems'
Build target: DEBUG
[09:26:50] Using gulpfile ^\GetSharePointListItems\gulpfile.js
[09:26:50] Starting gulp
[09:26:50] Starting 'serve'...
[09:26:50] Starting subtask 'pre-copy'...
[09:26:50] Finished subtask 'pre-copy' after 11 ms
[09:26:50] Starting subtask 'copy-static-assets'...
[09:26:50] Starting subtask 'sass'...
[09:26:51] Finished subtask 'sass' after 1.03 s
[09:26:51] Starting subtask 'tslint'...
[09:26:51] Starting subtask 'typescript'...
[09:26:51] [typescript] TypeScript version: 2.1.6
[09:26:52] Finished subtask 'copy-static-assets' after 1.14 s
[09:26:56] Finished subtask 'tslint' after 4.6 s
[09:26:56] Finished subtask 'typescript' after 4.59 s
[09:26:56] Starting subtask 'ts-npm-lint'...
[09:26:56] Finished subtask 'ts-npm-lint' after 24 ms
[09:26:56] Starting subtask 'api-extractor'...
[09:26:56] Finished subtask 'api-extractor' after 1.88 ms
[09:26:56] Starting subtask 'post-copy'...
[09:26:56] Finished subtask 'post-copy' after 1.03 ms
```

Since the environment is local, the mock data has been used to generate the table, as shown below.



Thus, we have successfully tested the client Web part locally.

Test the Web part in SharePoint Online

Now, let's test the Web part in SharePoint Workbench available in SharePoint Online. This time, the 'EnvironmentType' check will evaluate to SharePoint and REST API endpoint method will be called to retrieve the list items from SharePoint list - EmployeesList , to which we are trying to connect, using REST API is given below.

A screenshot of a SharePoint Online page titled "Play Ground". The page shows a list titled "EmployeeList" with the following data:

Title	EmployeeName	EmployeeId	Experience	Location
1	PriyaranjanKS	E5678	SharePoint	India
2	Nimmy	E901	Java	Qatar
3	Jinesh	E782	Mobility	Sweden
4	Rajesh	E123	.NET	Bahrain
5	John	E781	SharePoint	UK
6	Vijai	E678	SharePoint	USA

Once we have login in to SharePoint Online, we can invoke the workbench by appending the text ‘_layouts/15/workbench.aspx’ to SharePoint Online URL. As we can see below, the items have been successfully retrieved, using REST API and the data has been built into HTML table in the client Web part.

Welcome to SharePoint Framework Development

Demo : Retrieve Employee Data from SharePoint List

EmployeeId	EmployeeName	Experience	Location
E5678	PriyaranjankS	SharePoint	India
E901	Nimmy	Java	Qatar
E782	Jinesh	Mobility	Sweden
E123	Rajesh	.NET	Bahrain
E781	John	SharePoint	UK
E678	Vijai	SharePoint	USA

We can further modify the CSS by making changes in the ‘GetSpListItems.module.scss’ file.

```
OPEN EDITORS
GetSpListItemsWebPart.ts
GetSpListItems.module.scss
GetListItemsWebParts C:\Users\Varma\account...
MockHttpClient.ts

GETSPLISTITEMS
+ loc
+ tests
GetSpListItems.module.scss
GetSpListItems.module.scss.ts
GetSpListItemsWebPart.manifest.json
GetSpListItemsWebPart.ts
IGetSpListItemsWebPartProps.ts
MockHttpClient.ts

.helloworld {
  .container {
    max-width: 700px;
    margin: 0px auto;
    box-shadow: 0 2px 4px 0 rgba(0, 0, 0, 0.2), 0 25px 50px 0 rgba(0, 0, 0, 0.1);
  }

  .row {
    padding: 20px;
  }

  .listItem {
    max-width: 715px;
    margin: 5px auto 5px auto;
    box-shadow: 0 0 4px 0 rgba(0, 0, 0, 0.2), 0 25px 50px 0 rgba(0, 0, 0, 0.1);
  }

  .button {
    // our button
    text-decoration: none;
    height: 32px;
  }
}
```

The Type Script solution file has been zipped and uploaded [here](#). Feel free to work on it.

Provision Custom SharePoint List

Next we will see how to provision SharePoint assets using SharePoint Framework and TypeScript. The main solution files used in this section are uploaded in Microsoft [TechNet gallery](#). Feel free to download it.

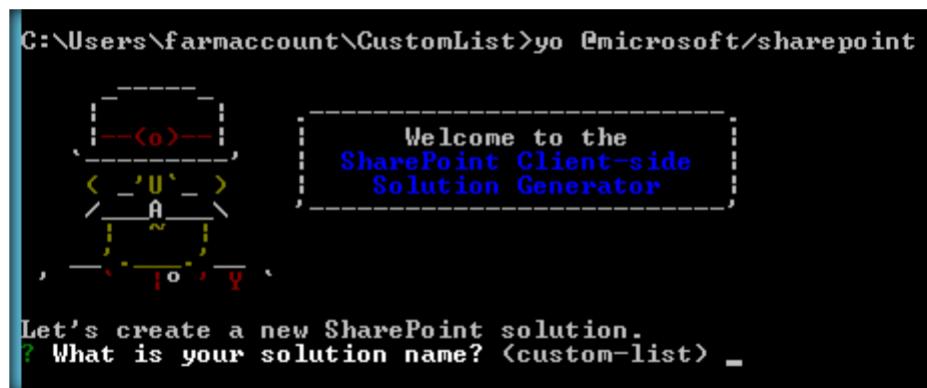
Create the Web part Project

We can create the directory, where we will be adding the solution, using the command given below.
md CustomList

Let's move to the newly created working directory, using the command.
cd CustomList

```
C:\Users\farmaccount>md CustomList  
C:\Users\farmaccount>cd CustomList
```

We will then create the client Web part by running the Yeoman SharePoint Generator. *yo @microsoft/sharepoint*



This will display the prompt, which we will have to fill up, so as to proceed with the project creation.

- What is your solution name? : Set it to 'CustomList'.

On pressing enter, we will be asked to chose the working folder for the project.

- Where do you want to place your files- Use current folder.
- What framework would you like to start with- Select "No javaScript web framework" for the time being, as this is a sample Web part.
- What is your Webpart name- We will specify it as 'CustomList' and press Enter
- What is your Webpart description- We will specify it as 'Custom List Created using SharePoint Framework'

```

Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? custom-list
? Where do you want to place the files? Use the current folder
? What framework would you like to start with? No JavaScript web framework
A folder with solution name custom-list will be created for you.
? What is your webpart name? CustomList
? What is your webpart description? <CustomList description> Custom List Create
d Using SharePoint Framework

```

Yeoman has started working on the scaffolding of the project. It will install the required dependencies and scaffold the solution files for the 'CustomList' Web part, which will take some time to complete. Once completed, we will get a congratulations message.

```

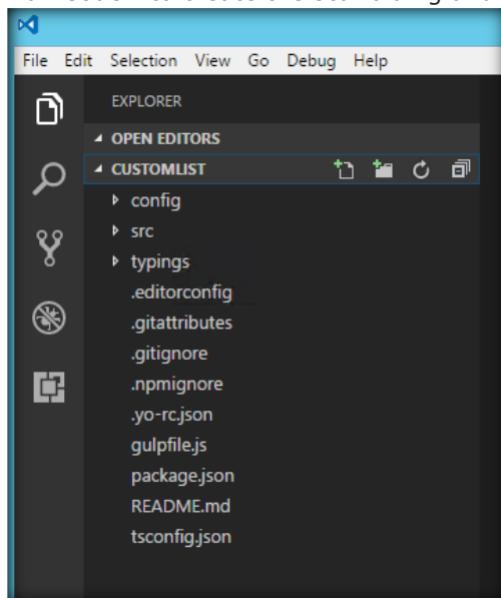
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@* (node_modules\@microsoft\gulp-core-build-webpack\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
>

    =+#####
    #####<@>          Congratulations!
    #####>@<e>          Solution custom-list is created.
    #####@><@>          Run gulp serve to play with it!
    #####@>
    #####>@<@>
    ***=+#####!

C:\Users\farmaccount\CustomList>_

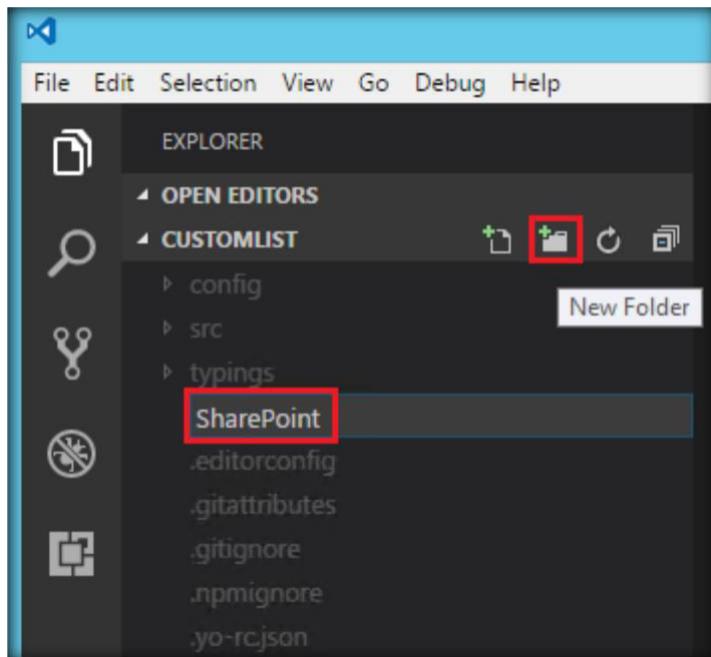
```

Run `Code .` to create the scaffolding and open the project in Visual Studio Code

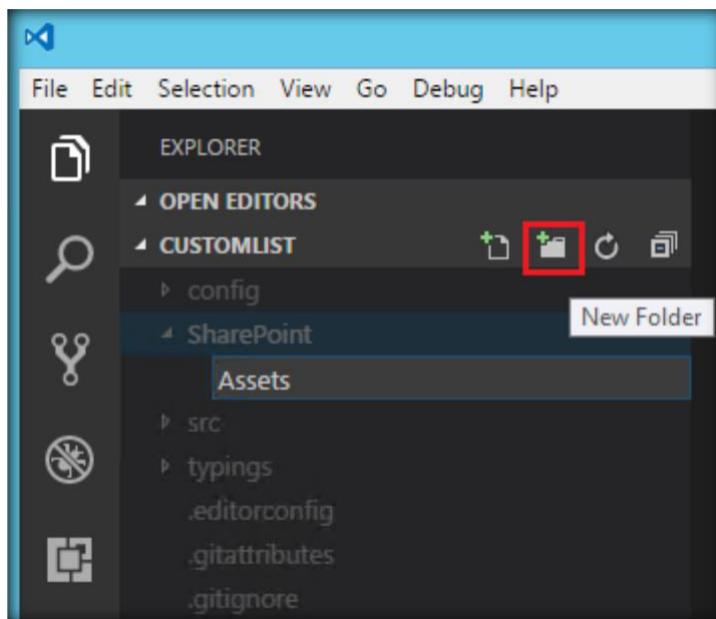


Edit the web part

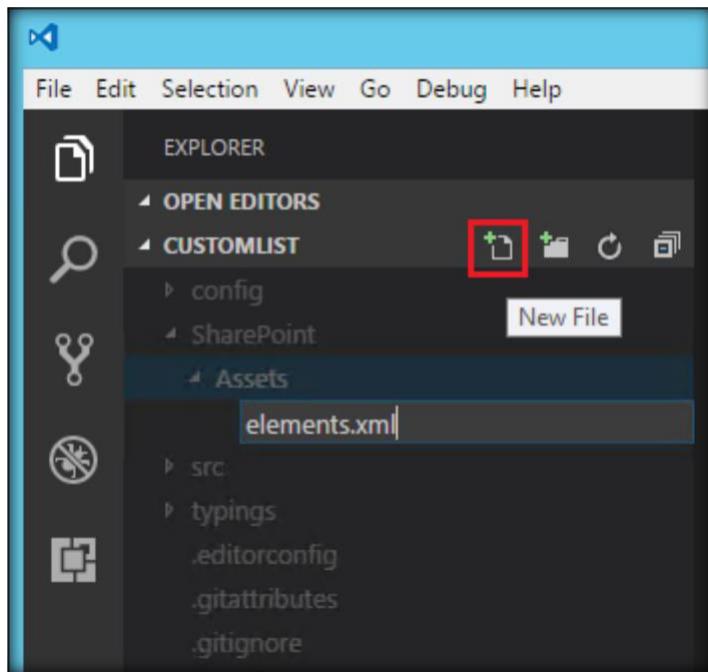
Now let's add the folder named 'SharePoint' to maintain the SharePoint files that will be deployed as a package.



Within the SharePoint folder lets add another sub folder named Assets.

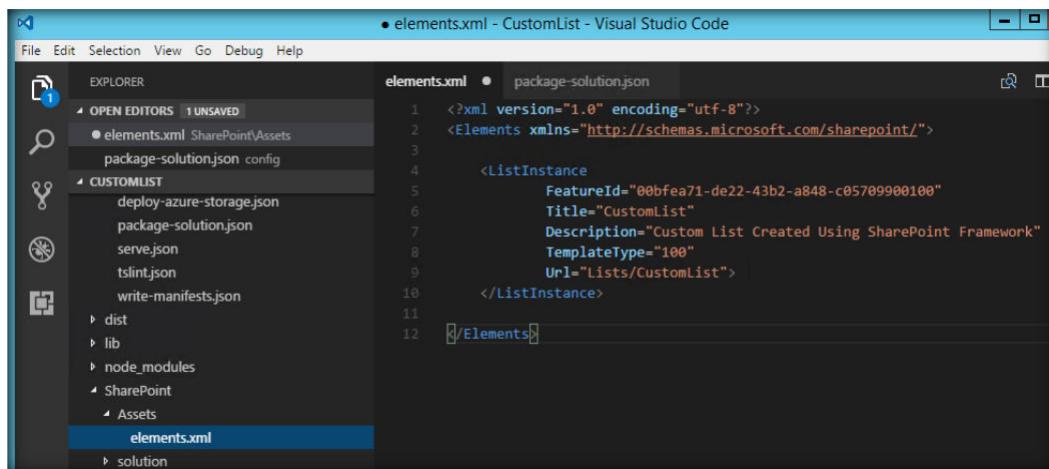


We will be creating an xml file - elements.xml which will hold the information required to provision the list. Let's create the first supporting xml file elements.xml.



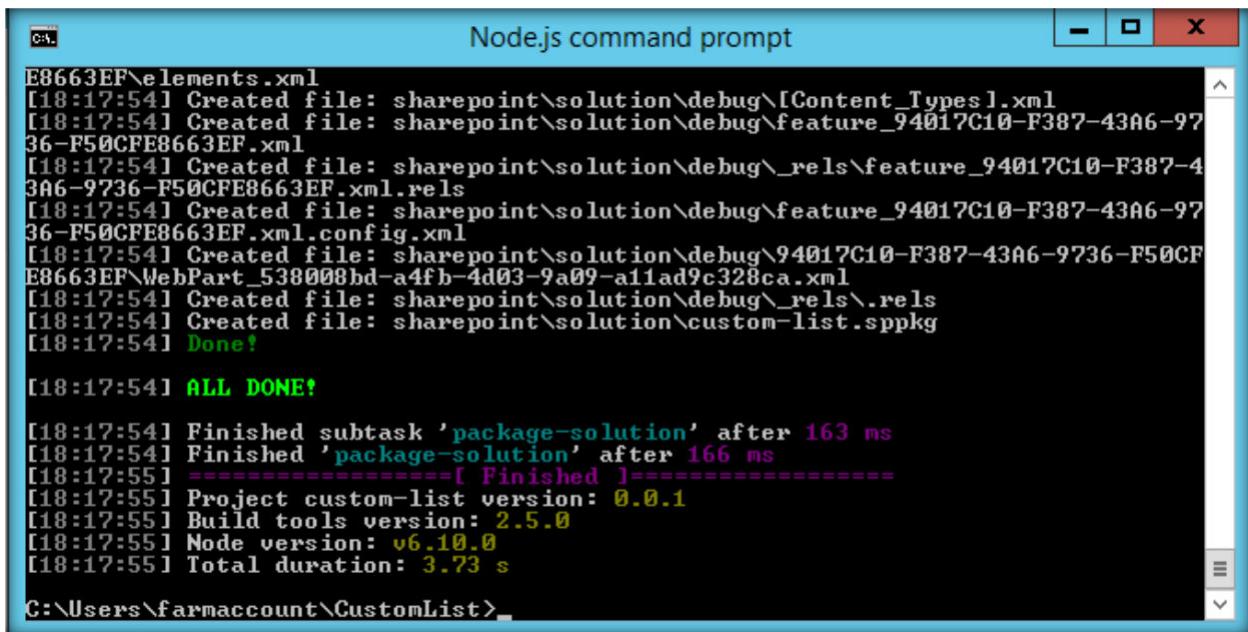
Add the below list information to the elements.xml file which contains the list name and type. The feature Id '00bfea71-de22-43b2-a848-c05709900100' refers to custom list.

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <Elements xmlns="http://schemas.microsoft.com/sharepoint/">
3.
4.     <ListInstance
5.         FeatureId="00bfea71-de22-43b2-a848-c05709900100"
6.         Title="CustomList"
7.         Description="Custom List Created Using SharePoint Framework"
8.         TemplateType="100"
9.         Url="Lists/CustomList">
10.    </ListInstance>
11.
12. </Elements>
```



Package and Deploy the Solution

Now let's create the deployment package by running `gulp serve` command from the Node.js command prompt.



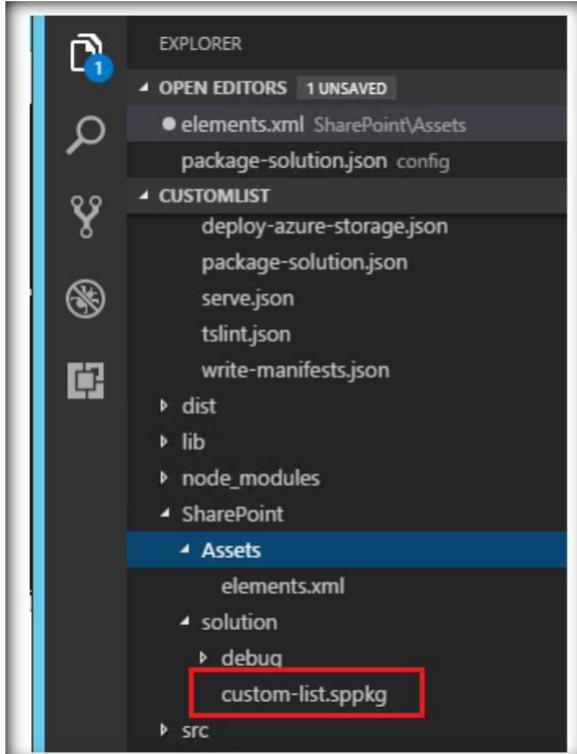
```
E8663EF\elements.xml
[18:17:54] Created file: sharepoint\solution\debug\[Content_Types].xml
[18:17:54] Created file: sharepoint\solution\debug\feature_94017C10-F387-43A6-97
36-F50CFE8663EF.xml
[18:17:54] Created file: sharepoint\solution\debug\_rels\feature_94017C10-F387-4
3A6-9736-F50CFE8663EF.xml.rels
[18:17:54] Created file: sharepoint\solution\debug\feature_94017C10-F387-43A6-97
36-F50CFE8663EF.xml.config.xml
[18:17:54] Created file: sharepoint\solution\debug\94017C10-F387-43A6-9736-F50CF
E8663EF\WebPart_538008bd-a4fb-4d03-9a09-a11ad9c328ca.xml
[18:17:54] Created file: sharepoint\solution\debug\_rels\.rels
[18:17:54] Created file: sharepoint\solution\custom-list.sppkg
[18:17:54] Done!

[18:17:54] ALL DONE!

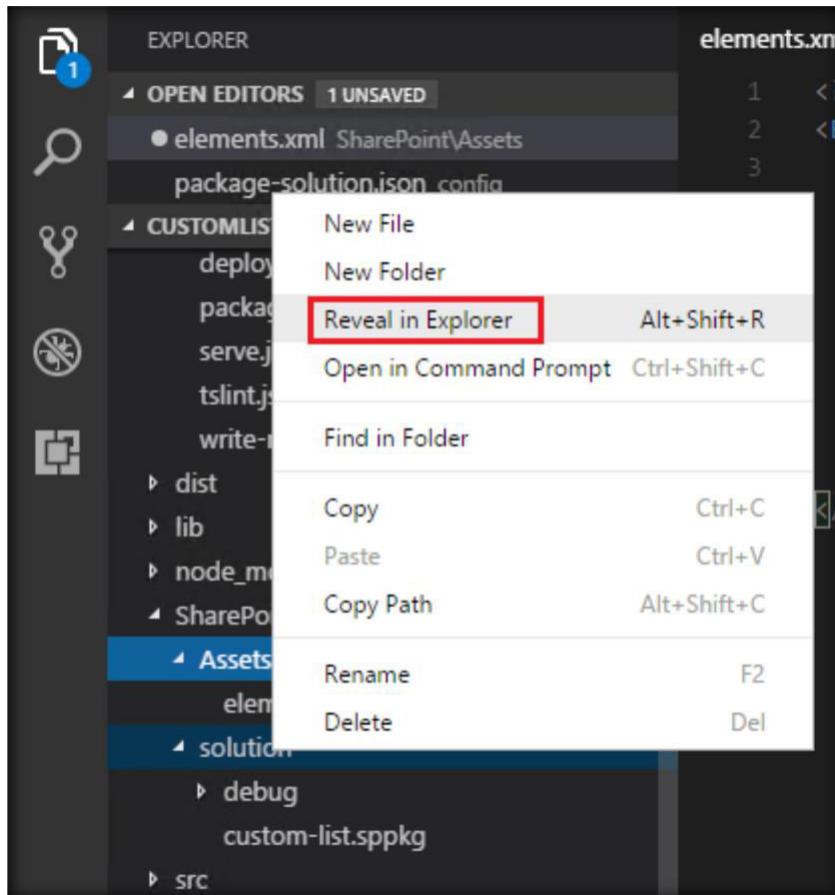
[18:17:54] Finished subtask 'package-solution' after 163 ms
[18:17:54] Finished 'package-solution' after 166 ms
[18:17:55] ======[ Finished ]=====[18:17:55] Project custom-list version: 0.0.1
[18:17:55] Build tools version: 2.5.0
[18:17:55] Node version: v6.10.0
[18:17:55] Total duration: 3.73 s

C:\Users\farmaccount\CustomList>
```

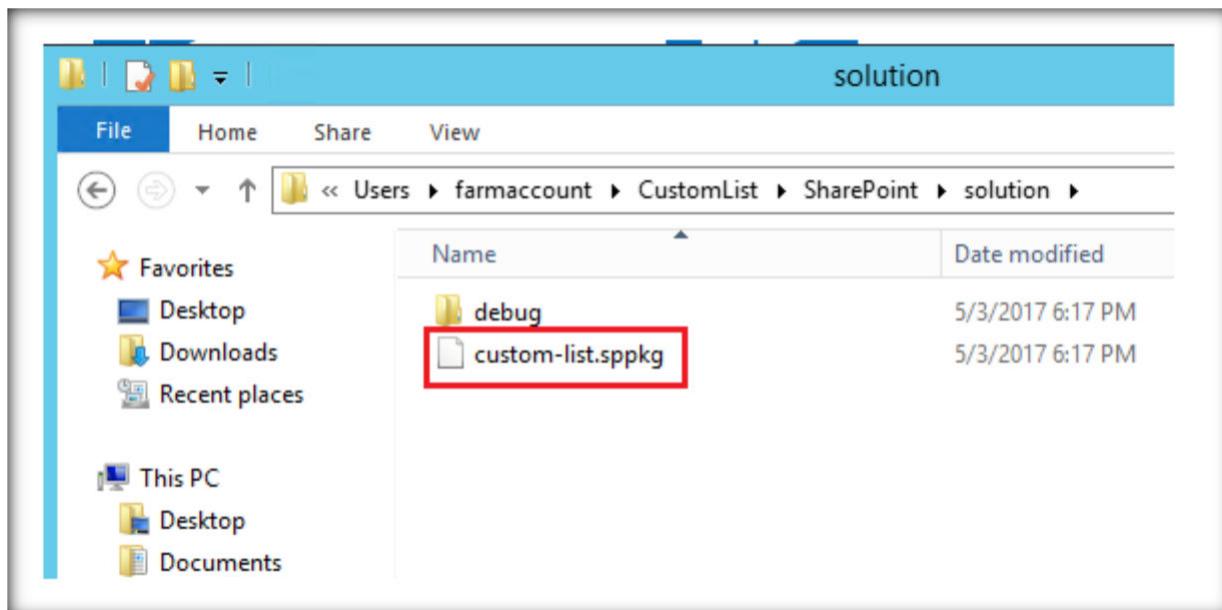
This will create the sppkg package in the solutions folder as shown below.



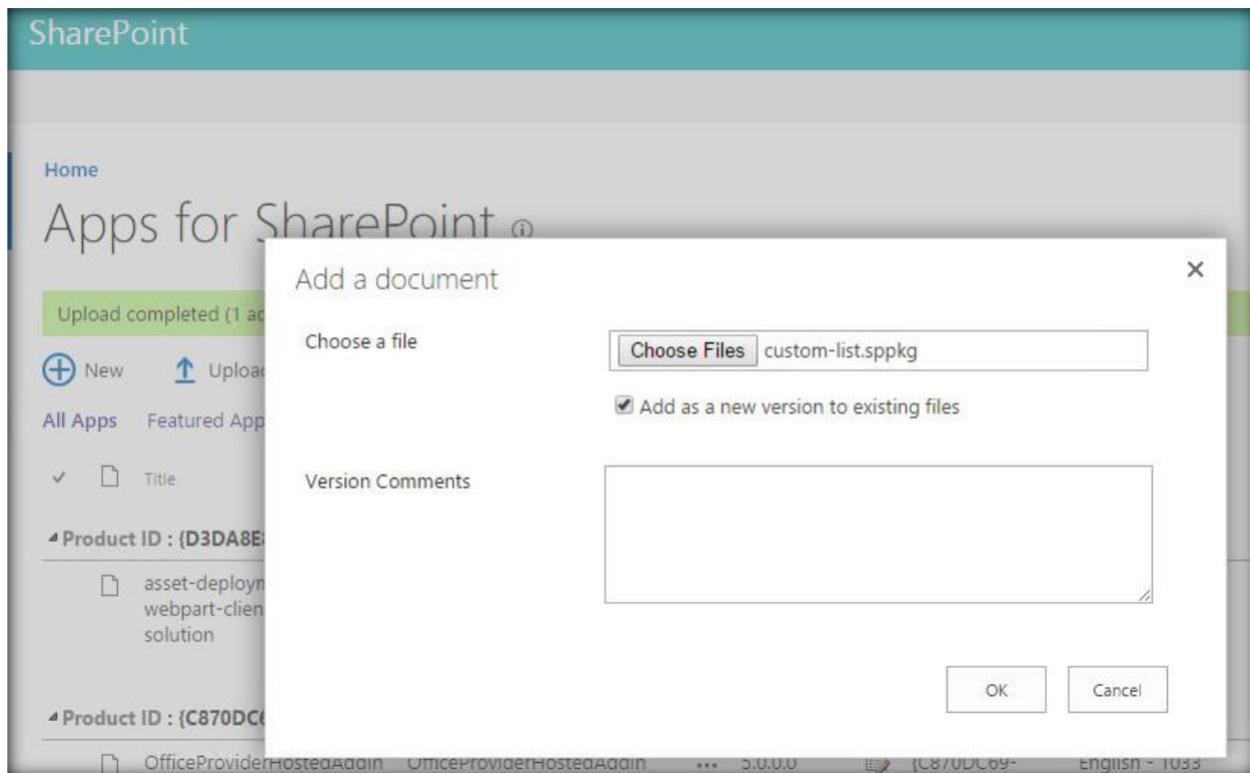
Take a note of the sppkg file url by opening it in File Explorer.



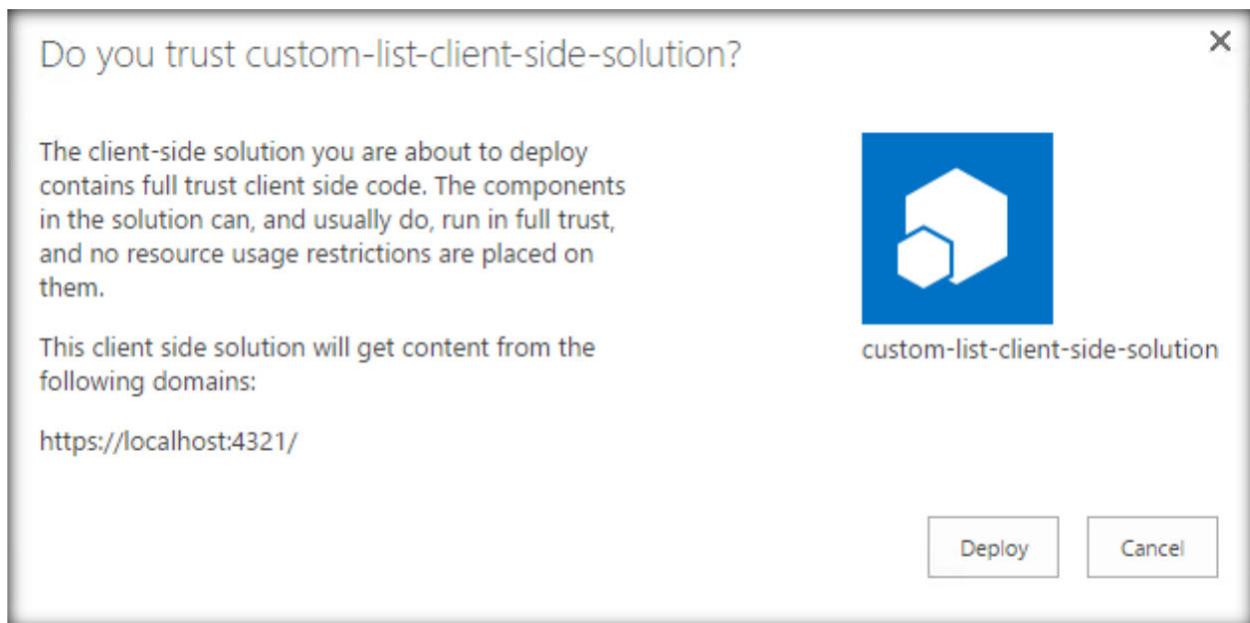
We will be uploading this package to the App Catalog in the next step.



Head over to the App Catalog and upload the package.



After upload it will ask if we trust the package. Click on Deploy to add the solution.



If we refresh the App Catalog page we can see the uploaded solution.

Apps for SharePoint

The screenshot shows the 'Apps for SharePoint' page with a green header bar indicating 'Upload completed (1 added) Refresh'. Below the header are standard navigation links: New, Upload, Sync, Share, More, All Apps, Featured Apps, Unavailable Apps, and a search bar. A table lists the uploaded app: 'custom-list-client-side-solution' (Title), 'custom-list' (Name), '1.0.0.0' (App Version), '(205BD8C4-356E-40E5-A072-A08B6D23762D)' (Product ID), 'English - 1033' (Metadata Language), and 'Yes' (Default Metadata Language). The entire row for the uploaded app is highlighted.

Ensure that there are no errors for the uploaded package by checking the below columns. In case there are some errors, we won't be able to add the solution to the site.

The screenshot shows a table with four columns: Enabled, Valid App Package, Deployed, and App Package Error Message. The 'Enabled' and 'Valid App Package' columns both have 'Yes' selected. The 'Deployed' column has 'Yes' selected. The 'App Package Error Message' column contains the text 'No errors.'

Now if we head over to the site, we can see the newly uploaded solution in the site contents.

The screenshot shows the SharePoint site contents page. At the top, there's a teal header bar with the SharePoint logo. Below it is a search bar labeled 'Find an app' with a magnifying glass icon. The main area is divided into sections: 'Noteworthy' (Document Library and Custom List tiles) and 'Apps you can add' (a grid of tiles). One tile in the 'Apps you can add' section, labeled 'custom-list-client-side-solution', is highlighted with a red border. Other tiles in this section include 'K2 blackpearl for SharePoint'.

Click on it to add the solution to the site.

The screenshot shows the SharePoint 'Site contents' page. At the top, there are links for 'EDIT LINKS' and 'SEARCH'. Below the title 'Site contents', there is a section titled 'Lists, Libraries, and other Apps'. It contains several items:

- 'add an app' (with a plus icon)
- 'Content and Structure Reports' (7 items, modified 14 months ago)
- 'CustomList' (new!, 0 items, modified 1 minute ago)
- 'custom-list-client-side-solution' (new!, with a blue cube icon)
- 'Documents' (0 items, modified 8 months ago)
- 'Employee' (3 items, modified 25 hours ago)

This will add the solution to the site contents. At the same time, it will provision whatever site assets were deployed as part of it. In our case, it is a custom list with the name 'Custom List'. We can see it from the Site contents as shown below.

The screenshot shows the 'CustomList' page under 'Site contents'. The title is 'CustomList'. Below it, there is a button for '+ new item or edit this list'. A search bar at the top right includes 'Find an item' and a magnifying glass icon. Below the search bar, there is a checkbox for 'Title'. A message at the bottom states: 'There are no items to show in this view of the "CustomList" list.'

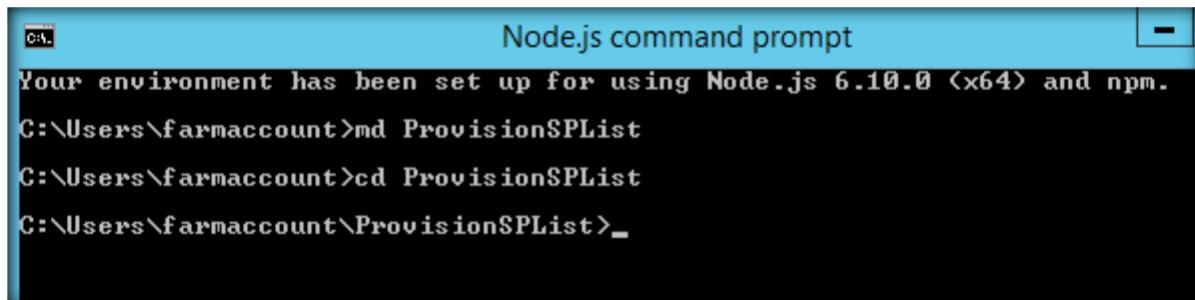
The main solution files used in this section are uploaded in Microsoft [TechNet gallery](#). Feel free to download it.

Provision SharePoint List with custom Site Columns and Content Type

In this section, we will see how to provision custom site columns and content type and how to use them while provisioning a custom list. The main solution files used in this section are uploaded in Microsoft [TechNet gallery](#). Feel free to download it.

We can create the directory, where we will be adding the solution, using the command given below.
md ProvisionSPList

Let's move to the newly created working directory, using the command.
cd ProvisionSPList



```
Node.js command prompt
Your environment has been set up for using Node.js 6.10.0 <x64> and npm.
C:\Users\farmaccount>md ProvisionSPList
C:\Users\farmaccount>cd ProvisionSPList
C:\Users\farmaccount\ProvisionSPList>_
```

We will then create the client Web part by running the Yeoman SharePoint Generator. *yo @microsoft/sharepoint*

This will display the prompt, which we will have to fill up, so as to proceed with the project creation.

- What is your solution name? : Set it to 'ProvisionSPList'.

On pressing enter, we will be asked to chose the working folder for the project.

```
yo
Your environment has been set up for using Node.js 6.10.0 <x64> and npm.
C:\Users\farmaccount>md ProvisionSPList
C:\Users\farmaccount>cd ProvisionSPList
C:\Users\farmaccount\ProvisionSPList>yo @microsoft/sharepoint

Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? (provision-sp-list) _
```

- Where do you want to place your files- Use current folder.
- What framework would you like to start with- Select “No javaScript web framework” for the time being, as this is a sample Web part.
- What is your Webpart name- We will specify it as ‘ProvisionSPList’ and press Enter
- What is your Webpart description- We will specify it as this Webpart will provision SharePoint List

```
yo
C:\Users\farmaccount\ProvisionSPList>yo @microsoft/sharepoint

Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? provision-sp-list
? Where do you want to place the files? Use the current folder
? What framework would you like to start with? No javaScript web framework
A folder with solution name provision-sp-list will be created for you.
? What is your webpart name? (HelloWorld) ProvisionSPList_
```

Yeoman has started working on the scaffolding of the project. It will install the required dependencies and scaffold the solution files for the ‘ProvisionSPList’ Web part, which will take some time to complete. Once completed, we will get a congratulations message.

```

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.0.0 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@* (node_modules\micromatch\gulp-core-build-webpack\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
>

=+#####!
#####
##/ <##:(e)
## ##:#:#: \
##/ /##: | <(e)
##### #:#:
## /##:(e)
#####
**=+#####!

Congratulations!
Solution provision-sp-list is created.
Run gulp serve to play with it!

```

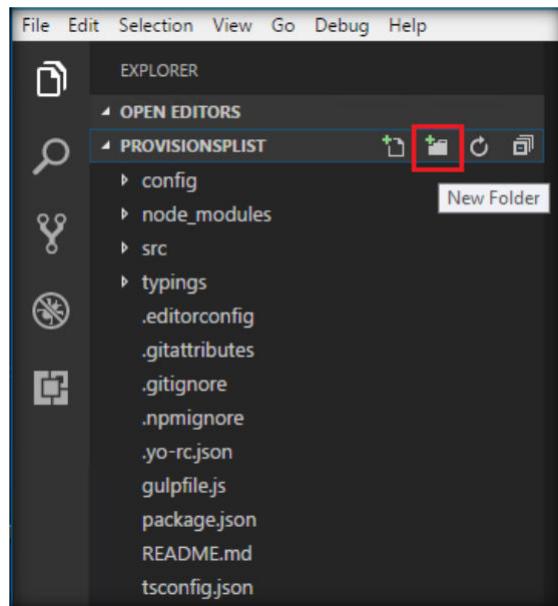
C:\Users\farmaccount\ProvisionSPList>_

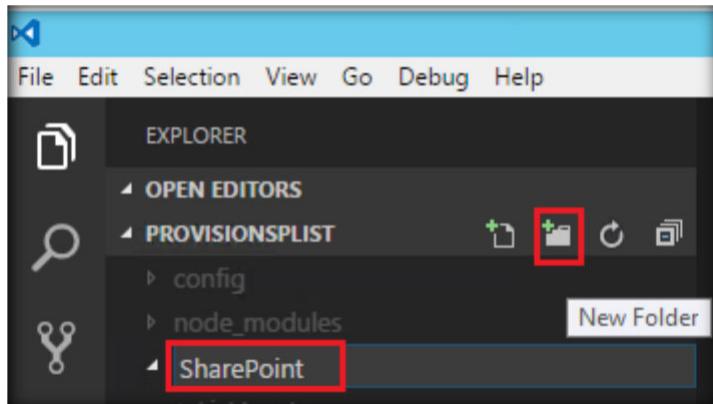
Edit the web part

Run the `Code .` to scaffold and open the project in Visual Studio Code.

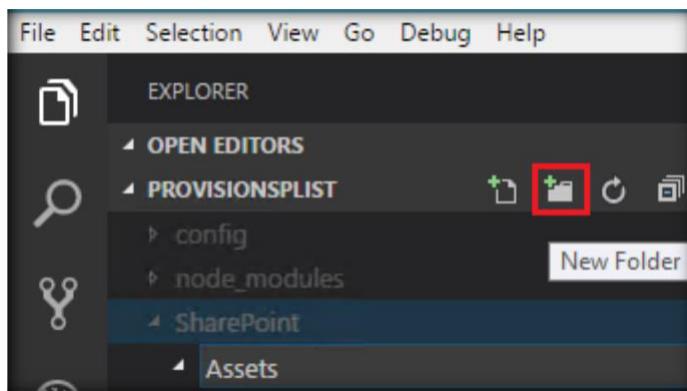
```
C:\Users\farmaccount\ProvisionSPList>code .
```

Now let's add the folder named 'Sharepoint' to maintain the SharePoint files that will be deployed as a package.

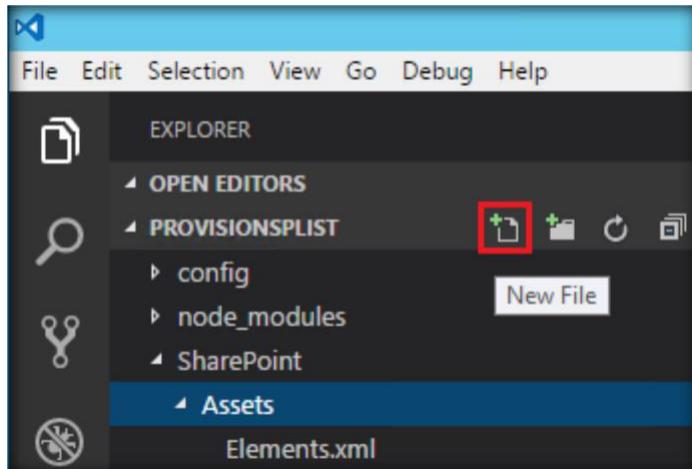




Within the SharePoint folder let's add another sub folder named Assets.



We will be creating two xml files - elements.xml and schema.xml which will hold the information required to provision the site columns, content type and then use them to create the list. Let's create the first supporting xml file elements.xml.



Elements.xml file will contain the list information that will be used to provision the list. At first we will be defining the site columns using the 'Field' tag and then the content type that will be deployed to the site. We will also be defining the default data that will be provisioned along with the list.

Add the Default data to SharePoint List

We will be adding the default data within the Rows tag as shown below.

```
<ListInstance
    CustomSchema="schema.xml"
    FeatureId="00bfea71-de22-43b2-a848-c05709900100"
    Title="Employee"
    Description="Employee List created using SharePoint Framework"
    TemplateType="100"
    Url="Lists/Employee">
<Data>
    <Rows>
        <Row>
            <Field Name="EmployeeName">Priyaranjan</Field>
            <Field Name="PreviousCompany">Cognizant</Field>
            <Field Name="JoiningDate">10/08/2010</Field>
            <Field Name="Expertise">SharePoint</Field>
            <Field Name="Experience">7</Field>
        </Row>
        <Row>
            <Field Name="EmployeeName">Nimmy</Field>
            <Field Name="PreviousCompany">SunTech</Field>
            <Field Name="JoiningDate">11/04/2012</Field>
            <Field Name="Expertise">Java</Field>
            <Field Name="Experience">4</Field>
        </Row>
        <Row>
            <Field Name="EmployeeName">Jinesh</Field>
            <Field Name="PreviousCompany">IBM</Field>
            <Field Name="JoiningDate">12/03/2006</Field>
            <Field Name="Expertise">.NET</Field>
            <Field Name="Experience">11</Field>
        </Row>
    </Rows>
</Data>
</ListInstance>
```

Elements.XML

The complete elements.xml that is used with the project is given below:

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <Elements xmlns="http://schemas.microsoft.com/sharepoint/">
3.
4.      <Field ID="{11ED4026-1C15-4636-80EF-C27C41DB90E0}"
5.              Name="EmployeeName"
6.              DisplayName="Employee Name"
7.              Type="Text"
8.              Required="FALSE"
```

```

9.           Group="Employee" />
10.      <Field ID="{1DA0BA30-F87A-4D1B-9303-729AA02BEE25}"
11.          Name="PreviousCompany"
12.          DisplayName="Previous Company"
13.          Type="Text"
14.          Required="FALSE"
15.          Group="Employee" />
16.      <Field ID="{145B5D00-E3AE-48EB-BB75-9699922AF8D8}"
17.          Name="JoiningDate"
18.          DisplayName="JoiningDate"
19.          Type="DateTime"
20.          Format="DateOnly"
21.          Required="FALSE"
22.          Group="Employee" />
23.      <Field ID="{197F8587-C417-458D-885E-4FBC28D1F612}"
24.          Name="Expertise"
25.          DisplayName="Expertise"
26.          Type="Choice"
27.          Required="FALSE"
28.          Group="Employee">
29.          <CHOICES>
30.              <CHOICE>SharePoint</CHOICE>
31.              <CHOICE>Java</CHOICE>
32.              <CHOICE>.NET</CHOICE>
33.              <CHOICE>Python</CHOICE>
34.              <CHOICE>C++</CHOICE>
35.              <CHOICE>Web Designer</CHOICE>
36.          </CHOICES>
37.      </Field>
38.  </ListInstance>
39. </List>
40. </List>
41. <Field ID="{10E72105-7577-4E9E-A758-BBBE8FF4E9BA}"
42.     Name="Experience"
43.     DisplayName="Experience"
44.     Group="Employee"
45.     Type="Number"
46.     Required="False"
47.     Min="0"
48.     Max="30"
49.     Percentage="FALSE">
50. </Field>
51. <ContentType ID="0x010100FA0963FA69A646AA916D2E41284FC9D1"
52.     Name="EmployeeContentType"
53.     Group="Employee Content Types"
54.     Description="This is the Content Type for Employee Onboarding">
55.     <FieldRefs>
56.         <FieldRef ID="{11ED4026-1C15-4636-80EF-C27C41DB90E0}" />
57.         <FieldRef ID="{1DA0BA30-F87A-4D1B-9303-729AA02BEE25}" />
58.         <FieldRef ID="{145B5D00-E3AE-48EB-BB75-9699922AF8D8}" />
59.         <FieldRef ID="{197F8587-C417-458D-885E-4FBC28D1F612}" />
60.         <FieldRef ID="{10E72105-7577-4E9E-A758-BBBE8FF4E9BA}" />
61.     </FieldRefs>
62. </ContentType>
63. <ListInstance
64.     CustomSchema="schema.xml"
65.     FeatureId="00bfea71-de22-43b2-a848-c05709900100"
66.     Title="Employee">
67. </ListInstance>
68. </List>
69. </List>

```

```

70.          Description="Employee List created using SharePoint Framework"
71.          TemplateType="100"
72.          Url="Lists/Employee">
73.      <Data>
74.          <Rows>
75.              <Row>
76.                  <Field Name="EmployeeName">Priyaranjan</Field>
77.                  <Field Name="PreviousCompany">Cognizant</Field>
78.                  <Field Name="JoiningDate">10/08/2010</Field>
79.                  <Field Name="Expertise">SharePoint</Field>
80.                  <Field Name="Experience">7</Field>
81.              </Row>
82.              <Row>
83.                  <Field Name="EmployeeName">Nimmy</Field>
84.                  <Field Name="PreviousCompany">SunTech</Field>
85.                  <Field Name="JoiningDate">11/04/2012</Field>
86.                  <Field Name="Expertise">Java</Field>
87.                  <Field Name="Experience">4</Field>
88.              </Row>
89.              <Row>
90.                  <Field Name="EmployeeName">Jinesh</Field>
91.                  <Field Name="PreviousCompany">IBM</Field>
92.                  <Field Name="JoiningDate">12/03/2006</Field>
93.                  <Field Name="Expertise">.NET</Field>
94.                  <Field Name="Experience">11</Field>
95.              </Row>
96.          </Rows>
97.      </Data>
98.  </ListInstance>
99.
100. </Elements>

```

```

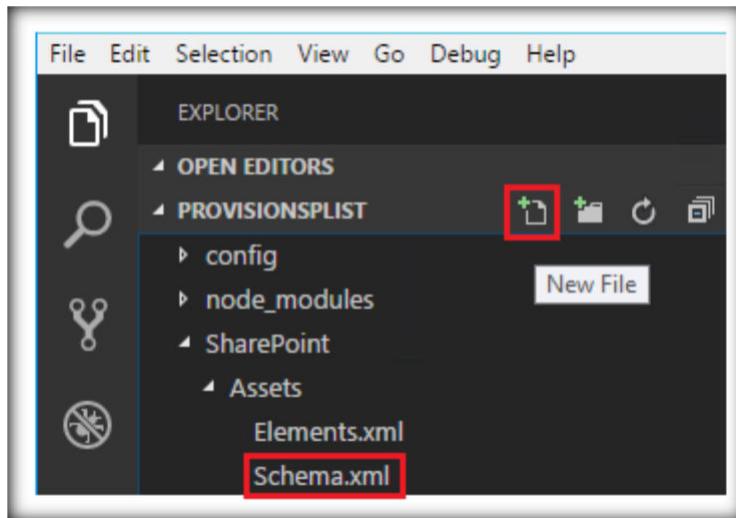
1  <?xml version="1.0" encoding="utf-8"?>
2  <Elements xmlns="http://schemas.microsoft.com/sharepoint/">
3
4      <Field ID="{21ED4026-1C15-4636-80EF-C27C41DB90E0}"
5          Name="EmployeeName"
6          DisplayName="Employee Name"
7          Type="Text"
8          Required="FALSE"
9          Group="Employee Columns" />
10
11     <Field ID="{BDA0BA30-F87A-4D1B-9303-729AA02BEE25}"
12         Name="PreviousCompany"
13         DisplayName="Previous Company"
14         Type="Text"
15         Required="FALSE"
16         Group="Employee Columns" />
17
18     <Field ID="{745B5D00-E3AE-48EB-BB75-9699922AF8D8}"
19         Name="JoiningDate"
20         DisplayName="JoiningDate"
21         Type="DateTime"
22         Format="DateOnly"
23         Required="FALSE"
24         Group="Employee Columns" />
25
26     <Field ID="{897F8587-C417-458D-885E-4FBC28D1F612}"
27         Name="Expertise"
28         DisplayName="Expertise"
29         Type="Choice"
30         Required="FALSE"
31         Group="Employee Columns" />
32

```

Schema.XML

Finally, we will be creating the schema.xml file which will contains the list xml. Here, we will be adding the Content Type that we have declared in the elements.xml as below

```
1. <ContentTypes>
2.   <ContentTypeRef ID="0x010100FA0963FA69A646AA916D2E41284FC9D9" />
3. </ContentTypes>
```



The complete schema.xml will look like below:

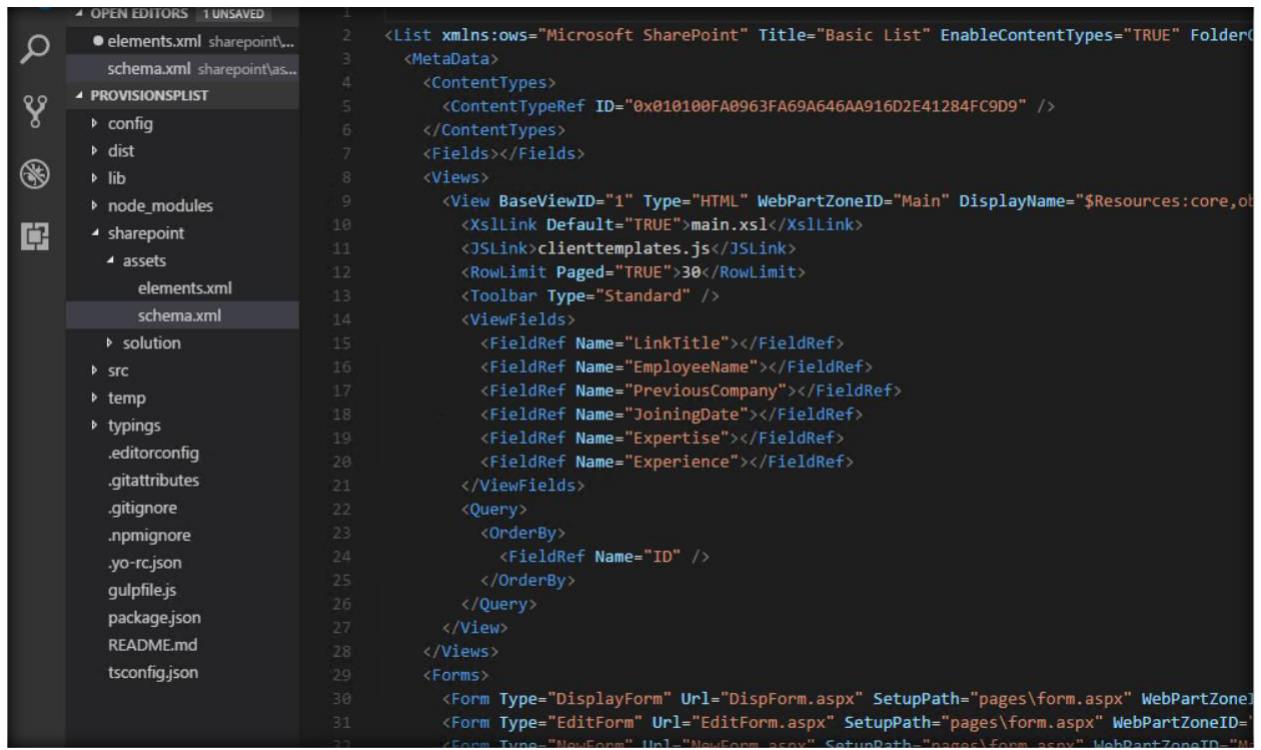
```
1. <List xmlns:ows="Microsoft SharePoint" Title="Basic List" EnableContentTypes="TRUE"
   FolderCreation="FALSE" Direction="$Resources:Direction;" Url="Lists/Basic List"
   BaseType="0" xmlns="http://schemas.microsoft.com/sharepoint/">
2.   <MetaData>
3.     <ContentTypes>
4.       <ContentTypeRef ID="0x010100FA0963FA69A646AA916D2E41284FC9D9" />
5.     </ContentTypes>
6.     <Fields></Fields>
7.     <Views>
8.       <View BaseViewID="1" Type="HTML" WebPartZoneID="Main">

9.         <XslLink Default="TRUE">main.xsl</XslLink>
10.        <JSLink>clienttemplates.js</JSLink>
11.        <RowLimit Paged="TRUE">30</RowLimit>
12.        <Toolbar Type="Standard" />
13.        <ViewFields>
14.          <FieldRef Name="LinkTitle"></FieldRef>
15.          <FieldRef Name="EmployeeName"></FieldRef>
16.          <FieldRef Name="PreviousCompany"></FieldRef>
17.          <FieldRef Name="JoiningDate"></FieldRef>
18.          <FieldRef Name="Expertise"></FieldRef>
19.          <FieldRef Name="Experience"></FieldRef>
20.        </ViewFields>
21.        <Query>
```

```

22.      <OrderBy>
23.          <FieldRef Name="ID" />
24.      </OrderBy>
25.      </Query>
26.  </View>
27. </Views>
28. <Forms>
29. <Form Type="DisplayForm" Url="DispForm.aspx" SetupPath="pages\form.aspx"
   WebPartZoneID="Main" />
30. <Form Type="EditForm" Url="EditForm.aspx" SetupPath="pages\form.aspx"
   WebPartZoneID="Main" />
31. <Form Type="NewForm" Url="NewForm.aspx" SetupPath="pages\form.aspx"
   WebPartZoneID="Main" />
32. </Forms>
33. </MetaData>
34. </List>

```



The screenshot shows a code editor interface with two tabs open: 'elements.xml' and 'schema.xml'. The 'schema.xml' tab is currently active, displaying the XML code for a SharePoint list definition. The code includes sections for MetaData, ContentTypes, Fields, Views, ViewFields, Query, OrderBy, and Forms, defining fields like LinkTitle, EmployeeName, PreviousCompany, JoiningDate, Expertise, and Experience, and specifying various list settings such as Default XSL link, RowLimit, and toolbar type.

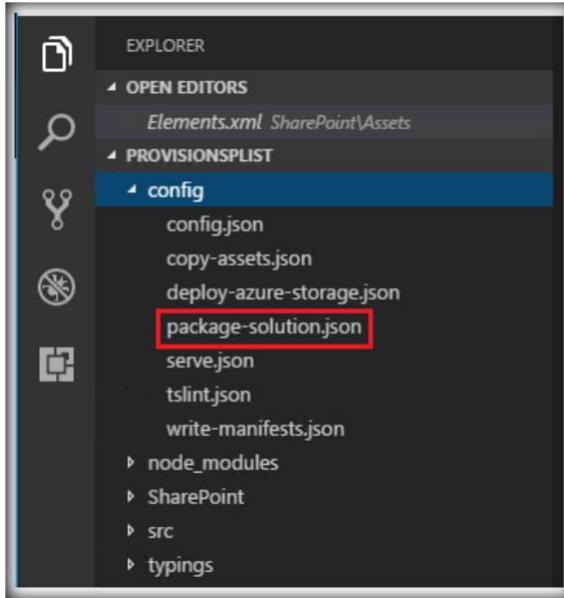
```

1  <List xmlns:ows="Microsoft SharePoint" Title="Basic List" EnableContentTypes="TRUE" FolderCreation="FALSE" Version="1" 
2  <MetaData>
3      <ContentTypes>
4          <ContentTypeRef ID="0x010100FA0963FA69A646AA916D2E41284FC9D9" />
5      </ContentTypes>
6      <Fields></Fields>
7      <Views>
8          <View BaseViewID="1" Type="HTML" WebPartZoneID="Main" DisplayName="$Resources:core,of...
9              <XslLink Default="TRUE">main.xsl</XslLink>
10             <JSLink>clienttemplates.js</JSLink>
11             <RowLimit Paged="TRUE">30</RowLimit>
12             <Toolbar Type="Standard" />
13             <ViewFields>
14                 <FieldRef Name="LinkTitle"></FieldRef>
15                 <FieldRef Name="EmployeeName"></FieldRef>
16                 <FieldRef Name="PreviousCompany"></FieldRef>
17                 <FieldRef Name="JoiningDate"></FieldRef>
18                 <FieldRef Name="Expertise"></FieldRef>
19                 <FieldRef Name="Experience"></FieldRef>
20             </ViewFields>
21             <Query>
22                 <OrderBy>
23                     <FieldRef Name="ID" />
24                 </OrderBy>
25             </Query>
26             <ViewFields>
27                 <FieldRef Name="LinkTitle"></FieldRef>
28             </ViewFields>
29             <Forms>
30                 <Form Type="DisplayForm" Url="DispForm.aspx" SetupPath="pages\form.aspx" WebPartZoneID="Main" />
31                 <Form Type="EditForm" Url="EditForm.aspx" SetupPath="pages\form.aspx" WebPartZoneID="Main" />
32                 <Form Type="NewForm" Url="NewForm.aspx" SetupPath="pages\form.aspx" WebPartZoneID="Main" />

```

Before we can deploy the package we have to update the feature in the package-solution.json file

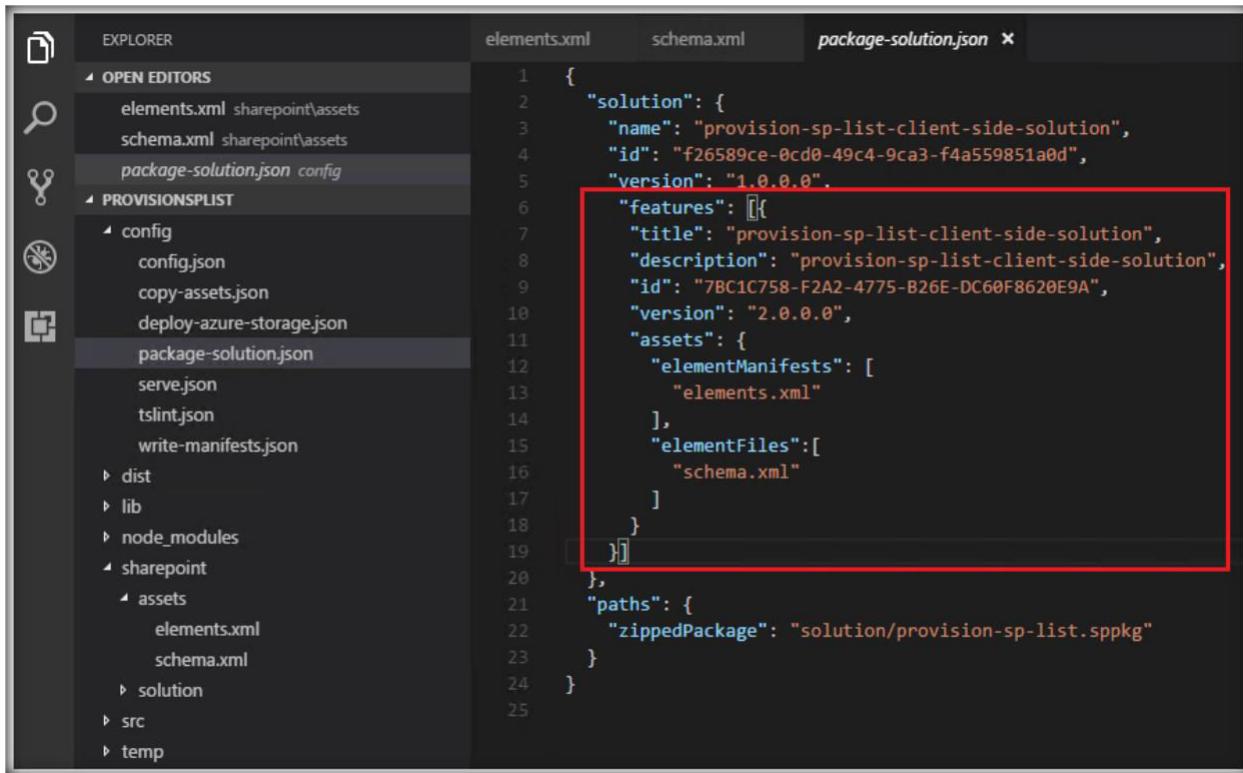
Update Package-Solution.json



Initially the file contents will contain only the solution name. We must add the feature node as well to this file.

```
1  {
2    "solution": {
3      "name": "provision-sp-list-client-side-solution",
4      "id": "f26589ce-0cd0-49c4-9ca3-f4a559851a0d",
5      "version": "1.0.0.0"
6    },
7    "paths": {
8      "zippedPackage": "solution/provision-sp-list.sppkg"
9    }
10 }
11 }
```

Add the below content after the version tag. Here the id is a Visual studio created GUID that identifies a unique feature.



```
1  {
2    "solution": {
3      "name": "provision-sp-list-client-side-solution",
4      "id": "f26589ce-0cd0-49c4-9ca3-f4a559851a0d",
5      "version": "1.0.0.0",
6      "features": [
7        {
8          "title": "provision-sp-list-client-side-solution",
9          "description": "provision-sp-list-client-side-solution",
10         "id": "7BC1C758-F2A2-4775-B26E-DC60F8620E9A",
11         "version": "2.0.0.0",
12         "assets": {
13           "elementManifests": [
14             "elements.xml"
15           ],
16           "elementFiles": [
17             "schema.xml"
18           ]
19         }
20       },
21       "paths": {
22         "zippedPackage": "solution/provision-sp-list.sppkg"
23       }
24     }
25 }
```

The contents of the package-solution.json will look like below :

```
1.  {
2.    "solution": {
3.      "name": "provision-sp-list-client-side-solution",
4.      "id": "f26589ce-0cd0-49c4-9ca3-f4a559851a0d",
5.      "version": "1.0.0.0",
6.      "features": [
7.        {
8.          "title": "provision-sp-list-client-side-solution",
9.          "description": "provision-sp-list-client-side-solution",
10.         "id": "7BC1C758-F2A2-4775-B26E-DC60F8620E9A",
11.         "version": "2.0.0.0",
12.         "assets": {
13.           "elementManifests": [
14.             "elements.xml"
15.           ],
16.           "elementFiles": [
17.             "schema.xml"
18.           ]
19.         }
20.       },
21.       "paths": {
22.         "zippedPackage": "solution/provision-sp-list.sppkg"
23.       }
24.     }
```

Now let's package the solution using gulp bundle

Package and Deploy the Solution

Now we must package and bundle the solution

using *gulp bundle*

```
C:\Users\farmaccount\ProvisionSPList>
C:\Users\farmaccount\ProvisionSPList>gulp bundle
```

```
[10:17:45] Finished subtask 'tslint' after 3.79 s
[10:17:45] Finished subtask 'typescript' after 3.78 s
[10:17:45] Starting subtask 'ts-npm-lint'...
[10:17:45] Finished subtask 'ts-npm-lint' after 29 ms
[10:17:45] Starting subtask 'api-extractor'...
[10:17:45] Finished subtask 'api-extractor' after 1.44 ms
[10:17:45] Starting subtask 'post-copy'...
[10:17:45] Finished subtask 'post-copy' after 18 ms
[10:17:45] Starting subtask 'collectLocalizedResources'...
[10:17:45] Finished subtask 'collectLocalizedResources' after 12 ms
[10:17:45] Starting subtask 'configure-webpack'...
[10:17:46] Finished subtask 'configure-webpack' after 605 ms
[10:17:46] Starting subtask 'webpack'...
[10:17:47] Finished subtask 'webpack' after 849 ms
[10:17:47] Starting subtask 'configure-webpack-external-bundling'...
[10:17:47] Finished subtask 'configure-webpack-external-bundling' after 1.6 ms
[10:17:47] Starting subtask 'copy-assets'...
[10:17:47] Finished subtask 'copy-assets' after 23 ms
[10:17:47] Starting subtask 'write-manifests'...
[10:17:47] Finished subtask 'write-manifests' after 632 ms
[10:17:47] Finished 'bundle' after 7.05 s
[10:17:48] ======[ Finished ]=====
[10:17:48] Project provision-sp-list version: 0.0.1
[10:17:48] Build tools version: 2.5.0
[10:17:48] Node version: v6.10.0
[10:17:48] Total duration: 11 s
C:\Users\farmaccount\ProvisionSPList>
```

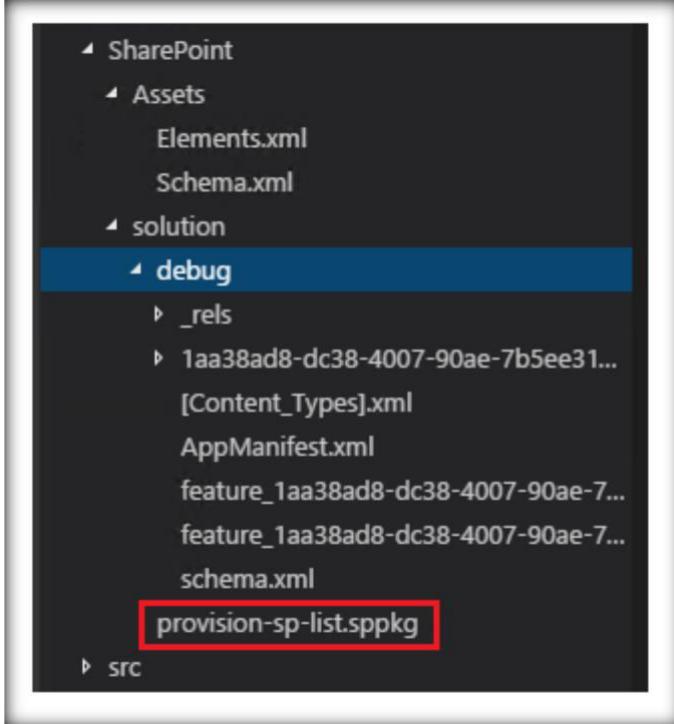
gulp package-solution

```
C:\Users\farmaccount\ProvisionSPList>
C:\Users\farmaccount\ProvisionSPList>gulp package-solution
```

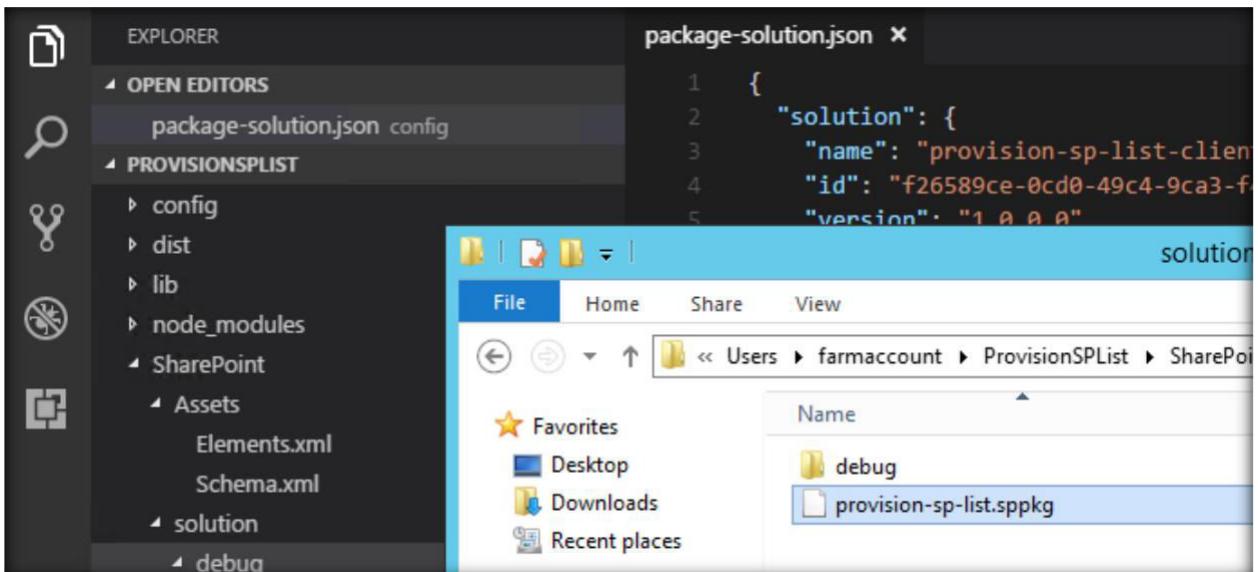
Thus, we are done with the packaging of the solution.

```
[10:19:07] Cleaned sharepoint\solution\debug
[10:19:07] Created file: sharepoint\solution\debug\_rels\AppManifest.xml.rels
[10:19:07] Created file: sharepoint\solution\debug\AppManifest.xml
[10:19:07] Created file: sharepoint\solution\debug\1aa38ad8-dc38-4007-90ae-7b5ee
3144e61\elements.xml
[10:19:07] Created file: sharepoint\solution\debug\_rels\.rels
[10:19:07] Created file: sharepoint\solution\debug\[Content_Types].xml
[10:19:07] Created file: sharepoint\solution\debug\feature_1aa38ad8-dc38-4007-90
ae-7b5ee3144e61.xml
[10:19:07] Created file: sharepoint\solution\debug\feature_1aa38ad8-dc38-4007-90
ae-7b5ee3144e61.xml.config.xml
[10:19:07] Created file: sharepoint\solution\debug\1aa38ad8-dc38-4007-90ae-7b5ee
3144e61\WebPart_eicca05f-1247-4d10-b43c-06d6859eb4f8.xml
[10:19:07] Created file: sharepoint\solution\debug\schema.xml
[10:19:07] Created file: sharepoint\solution\debug\_rels\feature_1aa38ad8-dc38-4
007-90ae-7b5ee3144e61.xml.rels
[10:19:08] Created file: sharepoint\solution\provision-sp-list.sppkg
[10:19:08] Done!
[10:19:08] ALL DONE!
[10:19:08] Finished subtask 'package-solution' after 168 ms
[10:19:08] Finished 'package-solution' after 170 ms
[10:19:08] ======[ Finished ]=====
[10:19:09] Project provision-sp-list version: 0.0.1
[10:19:09] Build tools version: 2.5.0
[10:19:09] Node version: v6.10.0
[10:19:09] Total duration: 3.8 s
C:\Users\farmaccount\ProvisionSPList>
```

If we head over to the solutions folder, we can see the 'provision-sp-list package' which we will be uploading to SharePoint.



Make a note of the solution URL in the local computer as we will need it to upload to SharePoint.



Let's head over to the SharePoint App Catalog site to where we will be uploading the solution.

term store
records management
search
secure store
apps
sharing

Search by URL...

URL

<https://ctsplayground.sharepoint.com>
<https://ctsplayground.sharepoint.com/portals/hub>
<https://ctsplayground.sharepoint.com/search>
 <https://ctsplayground.sharepoint.com/sites/AppCatalog>
<https://ctsplayground.sharepoint.com/sites/eDiscoverySite>

Click on Upload to add the solution file to the site.

Office 365 | SharePoint

BROWSE FILES LIBRARY

Home Apps for SharePoint ①

Recent More ▾

[New](#) Upload [Sync](#) [Share](#)

[All Apps](#) [Featured Apps](#) [Unavailable Apps](#) Find a file

Apps for SharePoint ...

Apps for Office

App Requests

✓ Title Name

Click on OK to complete the upload.

Home

Apps for SharePoint ①

Add a document

New Upload

All Apps Featured Apps

✓ Title

Product ID : {C870DC6A-0000-0000-0000-000000000000}

OfficeProvider

Product ID : {50F491D0-0000-0000-0000-000000000000}

SharePointHost

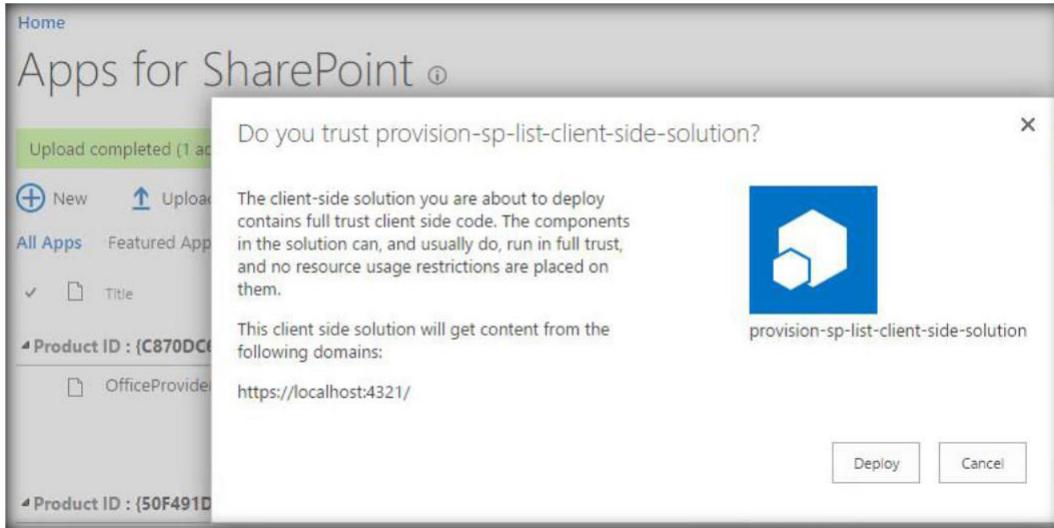
Choose a file provision-sp-list.sppkg

Add as a new version to existing files

Version Comments

OK Cancel

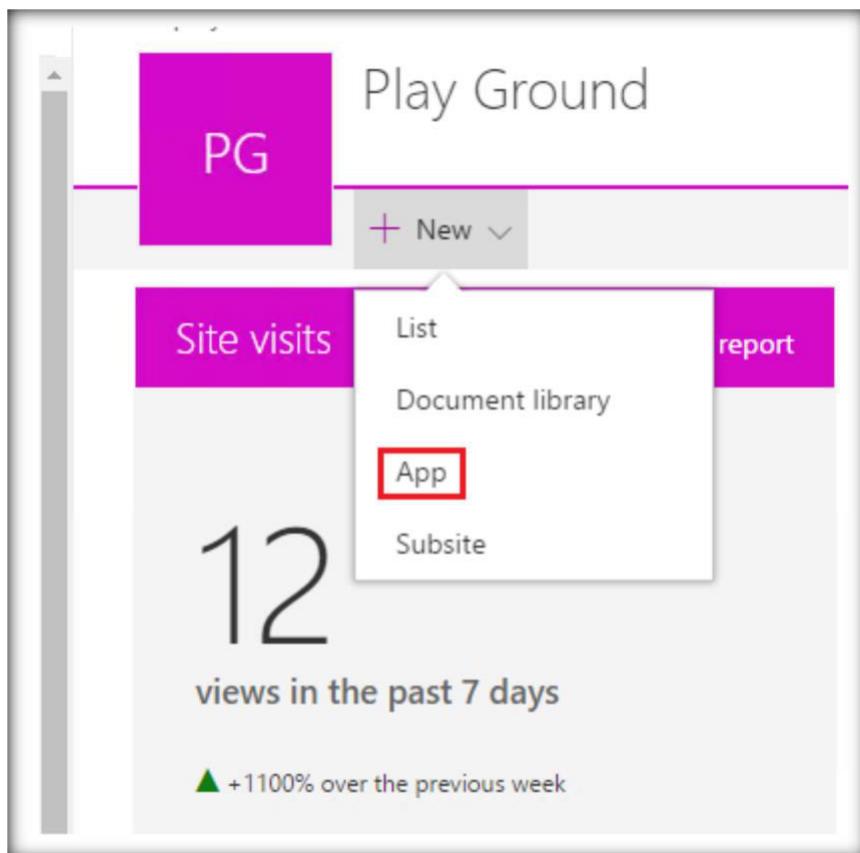
It will ask to trust and Deploy the solution to SharePoint.



We can see the uploaded solution in the App Catalog.

The screenshot shows the 'App Catalog' page with a green banner at the top indicating 'Upload completed (1 added) Refresh'. The page lists one item:
Product ID : {F26589CE-0CD0-49C4-9CA3-F4A559851A0D} (1)
provision-sp-list-client-side-solution
Name: provision-sp-list
Version: 1.0.0.0
Last modified: (F26589CE-0CD0-49C4-9CA3-F4A559851A0D)
Language: English - 1033
Default language: Yes

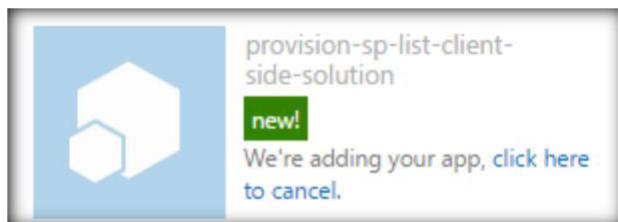
Now let's head over to the site contents and add the solution to the site.



On searching for the deployed app, it will list out the recently added solution.

The image shows the "Your Apps" section of a SharePoint site. The URL in the address bar is "provision-sp-list". The search results show one app matching the search term. The app is titled "provision-sp-list-client-side-solution" and has a blue icon with a white hexagonal shape. A red box highlights this icon and the app title. Below the app details, there is a link "App Details".

Click on it to add the solution to the site.



After few seconds, we can see the newly created custom site.

A screenshot of a SharePoint "Site contents" page. At the top, there is an "EDIT LINKS" button. The main title is "Site contents". Below it, a section titled "Lists, Libraries, and other Apps" contains two items: "add an app" (represented by a blue hexagonal icon with a plus sign) and "Employee" (represented by a blue square icon with four vertical bars). The "Employee" item includes a green "new!" button, the text "3 items", and the timestamp "Modified 3 minutes ago".

Going inside it we can see the default data that we had added to the list.

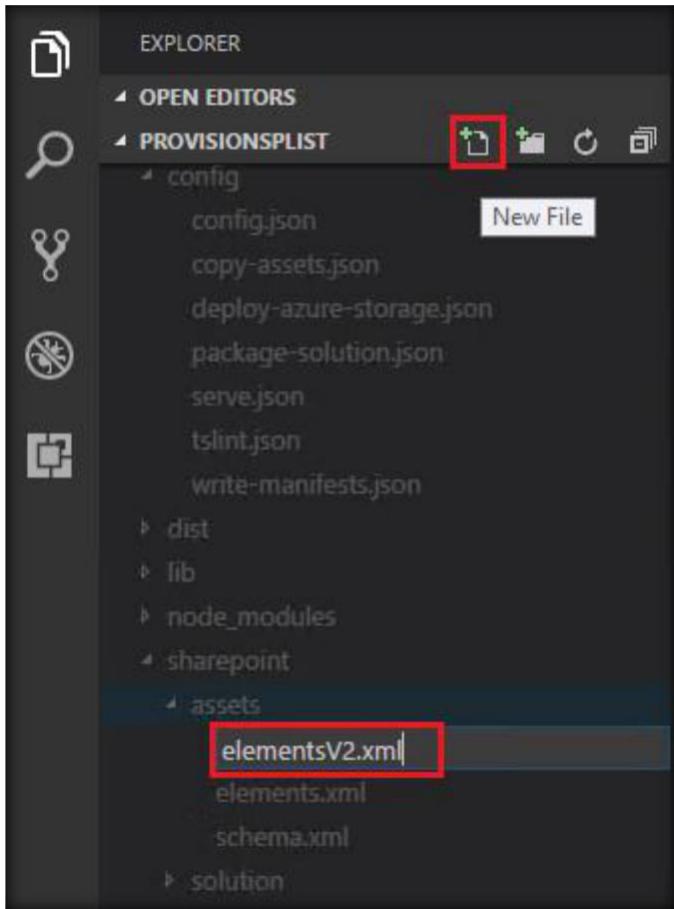
The screenshot shows a SharePoint list titled "Employee". The header bar is teal with the word "SharePoint". Below it, the page has a light gray header with a blue vertical bar on the left and the word "Home" in blue. The main content area has a white background. At the top of the list, there is a button labeled "+ new item or edit this list". Below the button, there is a search bar with the placeholder "Find an item" and a magnifying glass icon. To the left of the search bar are two buttons: "All Items" and "...". The list itself has a header row with columns: "Employee Name", "Previous Company", "JoiningDate", "Expertise", and "Experience". There are three data rows below the header:

Employee Name	Previous Company	JoiningDate	Expertise	Experience
Priyaranjan	Cognizant	10/8/2010	SharePoint	7
Nimmy	SunTech	11/4/2012	Java	4
Jinesh	IBM	12/3/2006	.NET	11

The main solution files used in this section are uploaded in Microsoft [TechNet gallery](#). Feel free to download it.

Upgrade the Solution

Once we have deployed the solution if we must make some changes at a later point, upgrade actions are available. Deploying without an upgrade will install a fresh copy of the solution. To retain the existing data, we will go with the upgrade option. However, to do the upgrade, we should add a new element file say elementsV2.xml.



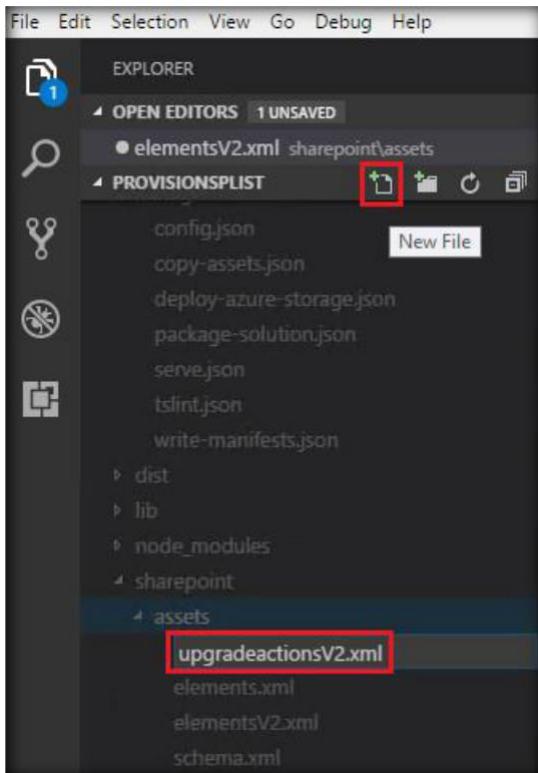
Upgrade the solution and add a new list

In our case, we are trying to add a new list along with the previously deployed solution. In the new elements file, we will specify the list instance declaration for a custom list.

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <Elements xmlns="http://schemas.microsoft.com/sharepoint/">
3.
4.   <ListInstance
5.     FeatureId="00bfea71-de22-43b2-a848-c05709900100"
6.     Title="ListAddedViaUpgrade"
7.     Description="List added through Upgrade Action"
8.     TemplateType="100"
9.     Url="Lists>ListAddedViaUpgrade">
10.    </ListInstance>
11.
12. </Elements>
```

```
elementsV2.xml •  
1  
2 <?xml version="1.0" encoding="utf-8"?>  
3 <Elements xmlns="http://schemas.microsoft.com/sharepoint/">  
4  
5     <ListInstance  
6         FeatureId="00bfea71-de22-43b2-a848-c05709900100"  
7         Title="ListAddedViaUpgrade"  
8         Description="List added through Upgrade Action"  
9         TemplateType="100"  
10        Url="Lists/ListAddedViaUpgrade">  
11    </ListInstance>  
12  
13 </Elements>
```

We will also add an upgrade action which will contain the information about the newly added elements.xml file.

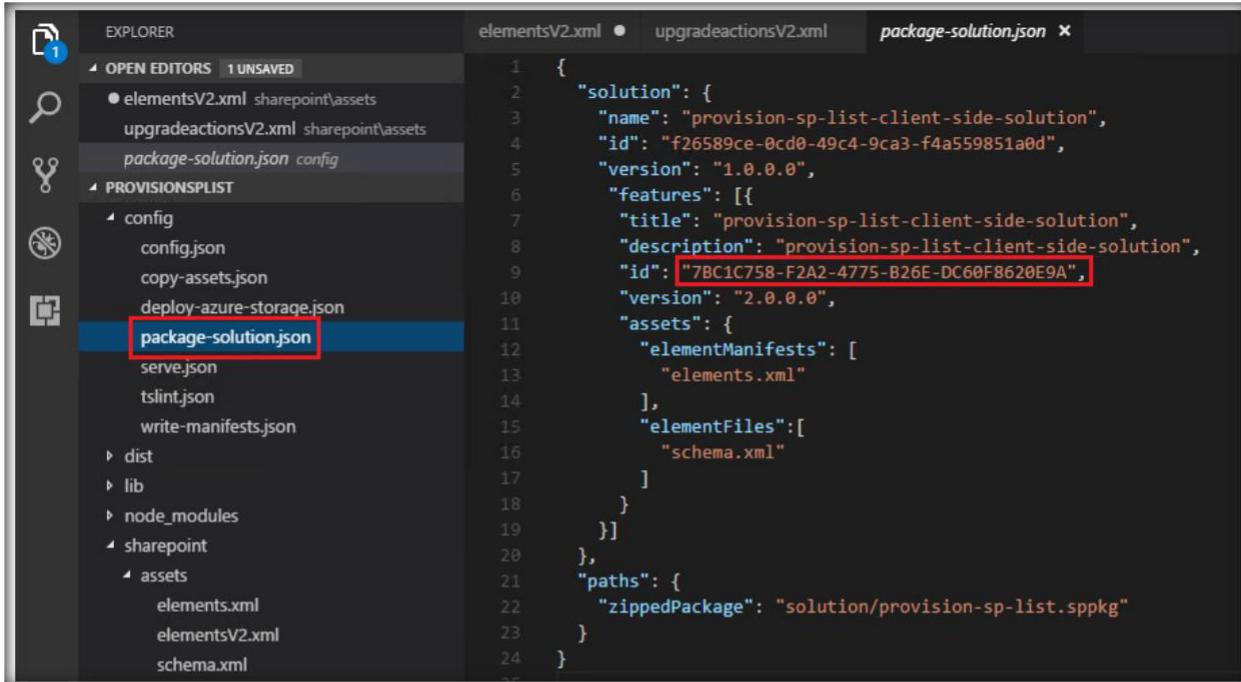


```

1. <ApplyElementManifests>
2.   <ElementManifest Location="7BC1C758-F2A2-4775-B26E-DC60F8620E9A\elementsV2.xml"
/>
3. </ApplyElementManifests>

```

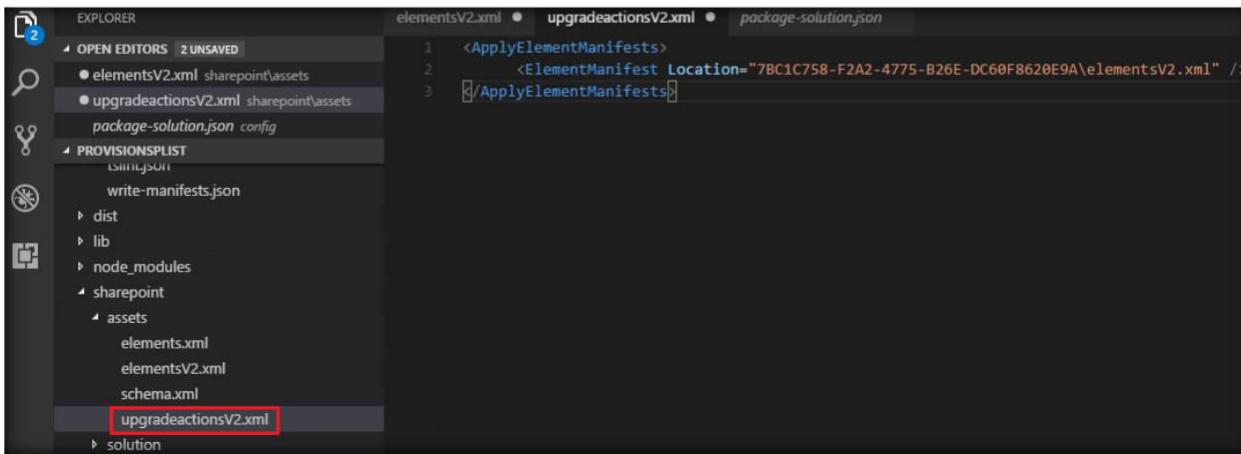
As the last step we have to update the *package-solution.json* file by adding *elementsV2.xml* to the *elementsmanifest* tag and we will also add the upgrade actions file reference.



```

1 { "solution": {
2   "name": "provision-sp-list-client-side-solution",
3   "id": "f26589ce-0cd0-49c4-9ca3-f4a559851a0d",
4   "version": "1.0.0.0",
5   "features": [{
6     "title": "provision-sp-list-client-side-solution",
7     "description": "provision-sp-list-client-side-solution",
8     "id": "7BC1C758-F2A2-4775-B26E-DC60F8620E9A",
9     "version": "2.0.0.0",
10    "assets": {
11      "elementManifests": [
12        "elements.xml"
13      ],
14      "elementFiles": [
15        "schema.xml"
16      ]
17    }
18  }],
19  "paths": {
20    "zippedPackage": "solution/provision-sp-list.sppkg"
21  }
22 }

```



```

1 <ApplyElementManifests>
2   <ElementManifest Location="7BC1C758-F2A2-4775-B26E-DC60F8620E9A\elementsV2.xml" />
3 </ApplyElementManifests>

```

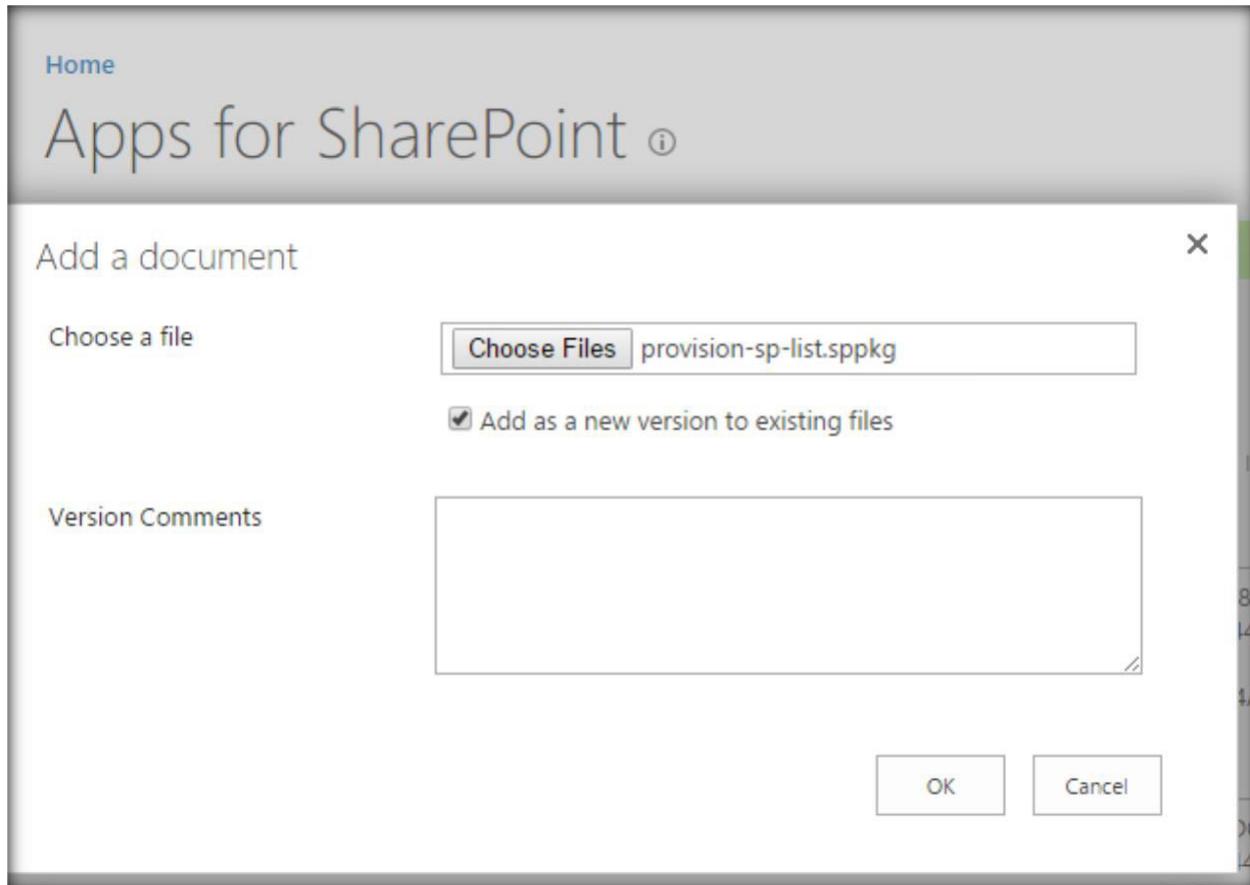
The screenshot shows the Visual Studio Code interface with the title "package-solution.json - ProvisionSPList - Visual Studio Code". The Explorer sidebar on the left lists various files and folders related to a SharePoint provisioning solution. The "package-solution.json" file is selected and highlighted with a red border in the Explorer. In the main editor area, the JSON content of the file is displayed, with several sections highlighted with red boxes:

```
{
  "solution": {
    "name": "provision-sp-list-client-side-solution",
    "id": "f26589ce-0cd0-49c4-9ca3-f4a559851a0d",
    "version": "2.0.0.0",
    "features": [
      {
        "title": "provision-sp-list-client-side-solution",
        "description": "provision-sp-list-client-side-solution",
        "id": "7BC1C758-F2A2-4775-B26E-DC60F8620E9A",
        "version": "2.0.0.0",
        "assets": {
          "elementManifests": [
            "elements.xml",
            "elementsV2.xml"
          ],
          "elementFiles": [
            "schema.xml"
          ],
          "upgradeActions": [
            "upgradeactionsV2.xml"
          ]
        }
      },
      "paths": {
        "zippedPackage": "solution/provision-sp-list.sppkg"
      }
    }
}
```

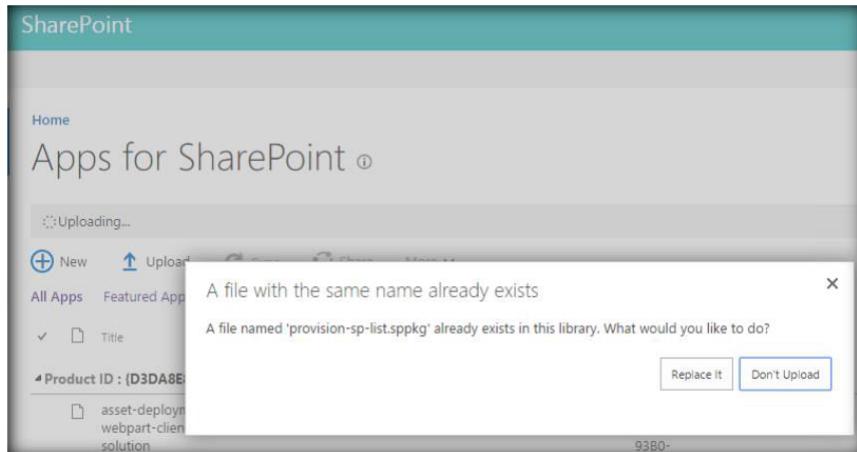
```
1.  {
2.    "solution": {
3.      "name": "provision-sp-list-client-side-solution",
4.      "id": "f26589ce-0cd0-49c4-9ca3-f4a559851a0d",
5.      "version": "2.0.0.0",
6.      "features": [
7.        {
8.          "title": "provision-sp-list-client-side-solution",
9.          "description": "provision-sp-list-client-side-solution",
10.         "id": "7BC1C758-F2A2-4775-B26E-DC60F8620E9A",
11.         "version": "2.0.0.0",
12.         "assets": {
13.           "elementManifests": [
14.             "elements.xml",
15.             "elementsV2.xml"
16.           ],
17.           "elementFiles": [
18.             "schema.xml"
19.           ],
20.           "upgradeActions": [
21.             "upgradeactionsV2.xml"
22.           ]
23.         }
24.       },
25.       "paths": {
26.         "zippedPackage": "solution/provision-sp-list.sppkg"
27.       }
28.     }
```

Package and Deploy the Solution

Save the files and run `gulp serve` to package the solution file. Now let's go ahead and upload the sppkg solution file to SharePoint App Catalog.



Upon clicking on OK it will give a warning whether to replace it with the new solution. Click on Replace it so that the new version is added.



We can see that the version has upgraded from 1.0.0.0 to 2.0.0.0



Heading over to the site contents we can see that the new list 'ListAddedViaUpgrade' has been provisioned in addition to the existing Employee list.

A screenshot of the Site Contents page. It displays several items:

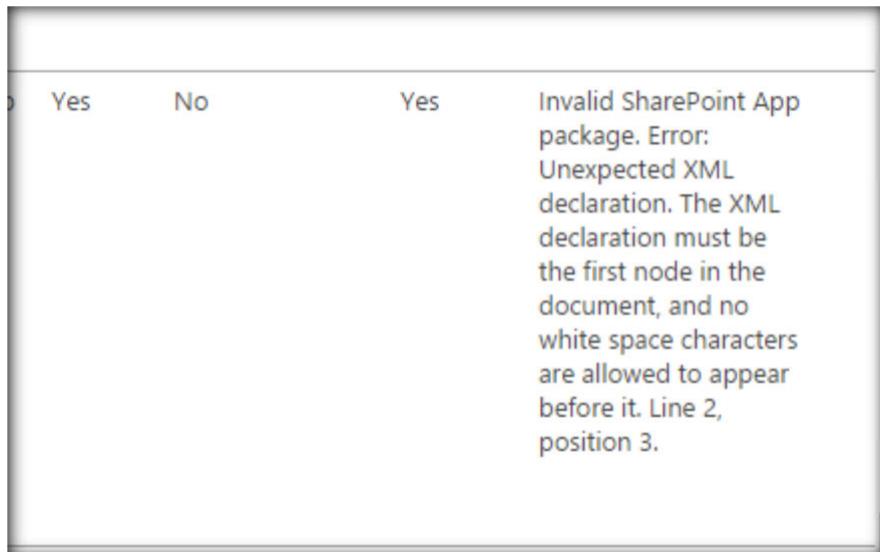
- Documents: 0 items, Modified 20 hours ago
- ListAddedViaUpgrade**: 0 items, Modified 1 minute ago (highlighted with a red box)
- Employee: 6 items, Modified 50 minutes ago
- MicroFeed: 2 items, Modified 10 hours ago
- Site Assets: 1 item, Modified 20 hours ago
- Site Pages: 2 items, Modified 20 hours ago

Resolve Package Errors

Once the solution has been uploaded to SharePoint, ensure that there are no errors mentioned in the below columns.

Valid App Package	Deployed	App Package Error Message
Yes	Yes	No errors.

At times if we have some error in the package, it will be displayed as shown below. Ensure that we resolve any such errors by analysing the error message. The below error was thrown because of a white space at the beginning of the Elements.XML file.



Getting Started with PnP JS development using SharePoint Framework

We can seamlessly integrate PnP JS file with SharePoint. The new Patterns and Practices JavaScript Core Library was created to help developers by simplifying common operations within SharePoint. Currently it contains a fluent API for working with the full SharePoint REST API as well as utility and helper functions. We can use them with SharePoint Framework to work with SharePoint with minimal effort. You can get detailed information and documentation about PnP JS from [here](#).

Retrieve SharePoint List Items using PnP JS and SharePoint Framework

As part of this article we will see how to create a SharePoint Framework webpart that retrieves List Items using PnP JS. The major project files used in this solution has been zipped and uploaded at Microsoft [TechNet Gallery](#). Feel free to download it.

We can create the directory, where we will be adding the solution, using the command given below.
`md PnPSPFrameworkGetItems`

Let's move to the newly created working directory, using the command.
`cd PnPSPFrameworkGetItems`

```
Node.js command prompt - yo @microsoft/sharepoint
Your environment has been set up for using Node.js 6.10.0 (x64) and npm.
C:\Users\farmaccount>md PnPSPFrameworkGetItems
C:\Users\farmaccount>cd PnPSPFrameworkGetItems
```

We will then create the client Web part by running the Yeoman SharePoint Generator. `yo @microsoft/sharepoint`

```
C:\Users\farmaccount\PnPSPFrameworkGetItems>yo @microsoft/sharepoint

Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? <pnp-psp-framework-get-items> _
```

This will display the prompt, which we will have to fill up, so as to proceed with the project creation.

- What is your solution name? : Set it to 'PnPSPFrameworkGetItems'. On pressing enter, we will be asked to chose the working folder for the project.
- Where do you want to place your files- Use current folder.
- What framework would you like to start with- Select "No javaScript web framework" for the time being, as this is a sample Web part.
- What is your Webpart name- We will specify it as 'PnPSPFrameworkGetItems' and press Enter
- What is your Webpart description- We will specify it as 'Get SP Lit items using PnP'

```
Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? PnPSPFrameworkGetItems
? Where do you want to place the files? Use the current folder
? What framework would you like to start with? No javaScript web framework
A folder with solution name PnPSPFrameworkGetItems will be created for you.
? What is your webpart name? PnPSPFrameworkGetItems
? What is your webpart description? <PnPSPFrameworkGetItems description> Get SP
List items using PnP_
```

Yeoman has started working on the scaffolding of the project. It will install the required dependencies and scaffold the solution files for the 'PnPSPFrameworkGetItems' Web part, which will take some time to complete. Once completed, we will get a congratulations message.

```
-- clone@0.2.0

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
>
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@* (node_modules\@microsoft\gulp-core-build-webpack\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
>

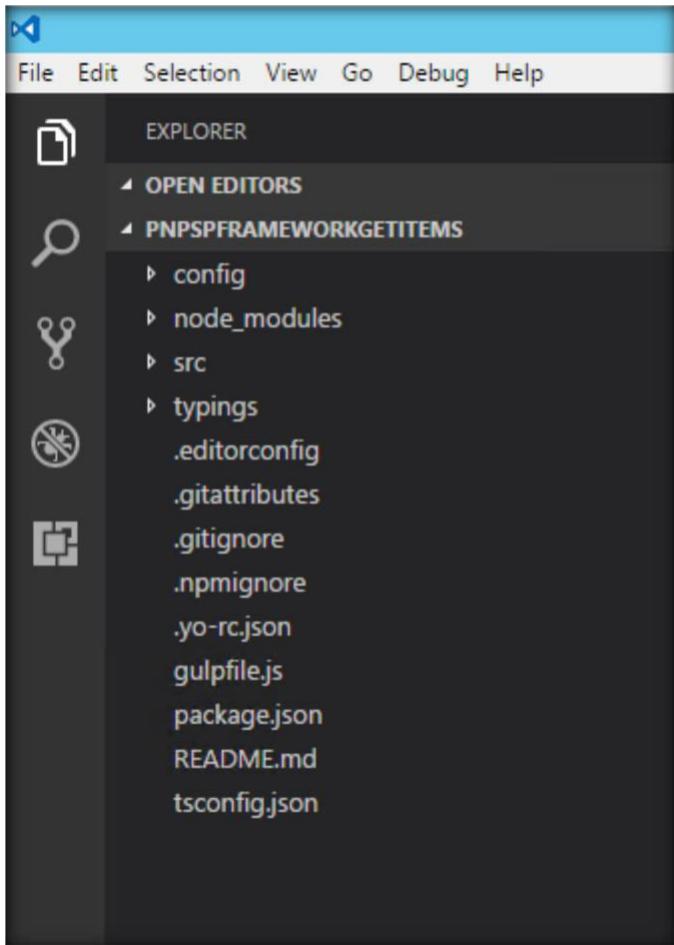
    =+#####
    #####:
    ####/ <##:<@>
    ##/  #####:
    ##/   /##:  <@> ,----- Congratulations!
    #####/  ##: | Solution pn-psp-framework-get-items is created.
    ##/   /##:<@>   Run gulp serve to play with it!
    ##########
    **=+#####!
```

C:\Users\farmaccount\PnPSPFrameworkGetItems>

Edit Webpart

Run Code . to create the scaffolding and open the project in Visual Studio Code

```
C:\Users\farmaccount\PnPSPFrameworkGetItems>
C:\Users\farmaccount\PnPSPFrameworkGetItems>code .
```



Now we have to load PnP JS file which we will use within the project to create list.We will be using npm to add PnP JS file as shown below.

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

```
C:\Users\farmaccount\PNPSPFrameworkGetItems>
C:\Users\farmaccount\PNPSPFrameworkGetItems>npm install sp-pnp-js --save
pn-pnp-framework-get-items@0.0.1 C:\Users\farmaccount\PNPSPFrameworkGetItems
  -- sp-pnp-js@2.0.4
    -- @types/sharepoint@2013.1.6
    -- @types/microsoft-ajax@0.0.32

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 <node_modules\chokidar\node_modules\fsevents>:
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} <current: {"os":"win32","arch":"x64"}>
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@* <node_modules\@micros\
```

Define List Model

Since we want to retrieve an Employee list items data, we will be creating list model with SharePoint list fields in the PnPspFrameworkGetItems.TS file, as shown below. Place it above the 'PnPspFrameworkGetItems' class.

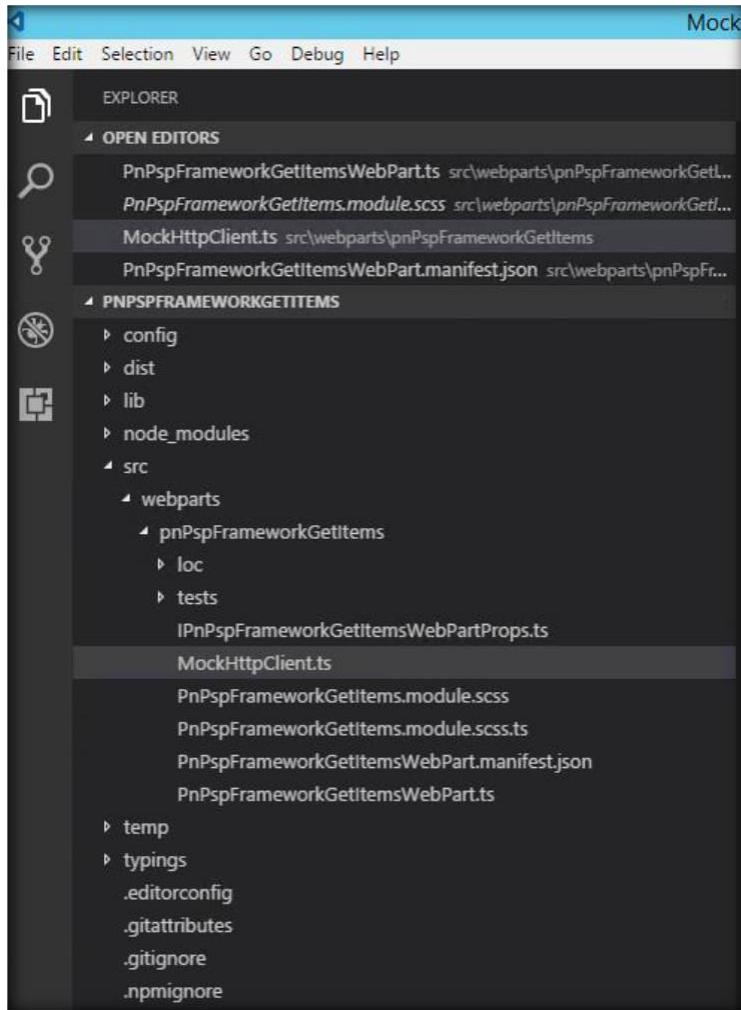
```
1. export interface ISPList {  
2.     EmployeeId: string;  
3.     EmployeeName: string;  
4.     Experience: string;  
5.     Location: string;  
6. }
```

Create Mock HttpClient to test data locally

In order to test the list item retrieval in the local workbench, we will create a mock store, which returns mock Employee list data. We will create a new file inside 'src\webparts PnPspFrameworkGetItemsWebPart' folder named MockHttpClient.ts, as shown below.

We will then copy the code given below into MockHttpClient.ts, as shown below.

```
1. import { ISPList } from './PnPspFrameworkGetItemsWebPart';  
2.  
3. export default class MockHttpClient {  
4.     private static _items: ISPList[] = [{ EmployeeId: 'E123', EmployeeName: 'John',  
      Experience: 'SharePoint', Location: 'India' },];  
5.     public static get(restUrl: string, options?: any): Promise<ISPList[]> {  
6.         return new Promise<ISPList[]>((resolve) => {  
7.             resolve(MockHttpClient._items);  
8.         });  
9.     }  
10. }
```



We can now use the `MockHttpClient` class in the '`PnPspFrameworkGetItems`' class. Let's import the '`MockHttpClient`' module by going to the `PnPspFrameworkGetItemsWebPart.ts` and pasting the line given below just after "`import { IPnPspFrameworkGetItemsWebPartProps } from './IPnPspFrameworkGetItemsWebPartProps';`"

```
1. import MockHttpClient from './MockHttpClient';
```

We will also add the mock list item retrieval method within the '`PnPspFrameworkGetItemsWebPart`' class.

```
1. private _getMockListData(): Promise<ISPList[]> {
2.   return MockHttpClient.get(this.context.pageContext.web.absoluteUrl).then(() => {
3.     const listData: ISPList[] = [
4.       { EmployeeId: 'E123', EmployeeName: 'John', Experience: 'SharePoint', Location: 'India' },
5.       { EmployeeId: 'E567', EmployeeName: 'Martin', Experience: '.NET', Location: 'Qatar' },
6.       { EmployeeId: 'E367', EmployeeName: 'Luke', Experience: 'JAVA', Location: 'UK' }
7.     ]
8.   })
9. }
```

```
7.           ];
8.           return listData;
9.       }) as Promise<ISPList[]>;
10. }
```

Retrieve SharePoint List Items

We will be making use of PnP library to get the list items as shown below:

```
1. private _getListData(): Promise<ISPList[]> {
2.     return pnp.sp.web.lists.getByTitle("EmployeeList").items.get().then((response) => {
3.
4.         return response;
5.     });
6.
7. }
```

Retrieve the SharePoint List Items From Employee List

Once we run the gulp serve command, we can test the Web part in SharePoint Workbench in the local environment or using SharePoint Online Context. SharePoint Framework uses 'EnvironmentType' module to identify the environment, where the Web part is executed.

To implement this, we will import 'Environment' and the 'EnvironmentType' modules from the @microsoft/sp-core-library bundle by placing it at the top of the PnPspFrameworkGetItemsWebpart.ts file.

```
import {
    Environment,
    EnvironmentType
} from '@microsoft/sp-core-library';
```

We will then check Environment.type value and if it is equal to Environment.Local, the MockHttpClient method, which returns dummy data will be called else the method that calls REST API that will retrieve SharePoint list items will be called.

```
1. private _renderListAsync(): void {
2.
3.     if (Environment.type === EnvironmentType.Local) {
4.         this._getMockListData().then((response) => {
5.             this._renderList(response.value);
6.         });
7.     }
8.     else {
9.         this._getListData()
10.            .then((response) => {
11.                this._renderList(response.value);
12.            });
13.    }
14. }
```

Finally, we will add the method given below, which will create HTML table out of the retrieved SharePoint list items.

```
1. private _renderList(items: ISPList[]): void {
2. let html: string = '<table class="TFtable" border=1 width=100% style="border-collapse: collapse;">';
3. html += `<th>EmployeeId</th><th>EmployeeName</th><th>Experience</th><th>Location</th>`;
4. items.forEach((item: ISPList) => {
5.     html += `
6.       <tr>
7.         <td>${item.EmployeeId}</td>
8.         <td>${item.EmployeeName}</td>
9.         <td>${item.Experience}</td>
10.        <td>${item.Location}</td>
11.      </tr>
12.    `;
13. });
14. html += `</table>`;
15. const listContainer: Element = this.domElement.querySelector('#spListContainer');
16. listContainer.innerHTML = html;
17. }
```

To enable rendering of the list items given above, we will replace Render method in the 'PnPspFrameworkGetItemsWebPart' class with the code given below.

```
1. public render(): void {
2.     this.domElement.innerHTML = `
3.       <div class="${styles.helloWorld}">
4.         <div class="${styles.container}">
5.           <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
6.             <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
7.               <span class="ms-font-xl ms-fontColor-white" style="font-size:28px">Welcome to
SharePoint Framework Development using PnP JS Library</span>
8.
9.             <p class="ms-font-l ms-fontColor-white" style="text-align: left">Demo : Retrieve
Employee Data from SharePoint List</p>
10.            </div>
11.          </div>
12.          <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
13.            <div style="background-color:Black;color:white;text-align: center;font-weight:
bold;font-size:18px;">Employee Details</div>
14.            <br>
15.            <div id="spListContainer" />
16.          </div>
17.        </div>
18.      </div>`;
19.     this._renderListAsync();
20.   }
```

TS File Contents to retrieve list data using PnP

The entire TS file contents that was used to implement data retrieval in the SPFx webpart is given below:

```
1. import pnp from 'sp-pnp-js';
2. import { Version } from '@microsoft/sp-core-library';
3.
4. import {
5.   BaseClientSideWebPart,
6.   IPropertyPaneConfiguration,
7.   PropertyPaneTextField
8. } from '@microsoft/sp-webpart-base';
9. import { escape } from '@microsoft/sp-lodash-subset';
10.
11. import {
12.   Environment,
13.   EnvironmentType
14. } from '@microsoft/sp-core-library';
15.
16. import styles from './PnPspFrameworkGetItems.module.scss';
17. import * as strings from 'pnpPspFrameworkGetItemsStrings';
18. import { IPnPspFrameworkGetItemsWebPartProps }
19.   from './IPnPspFrameworkGetItemsWebPartProps';
20. import MockHttpClient from './MockHttpClient';
21.
22. import {
23.   SPHttpClient
24. } from '@microsoft/sp-http';
25.
26. export interface ISPList {
27.   EmployeeId: string;
28.   EmployeeName: string;
29.   Experience: string;
30.   Location: string;
31. }
32.
33. export default class PnPspFrameworkGetItemsWebPart extends
34.   BaseClientSideWebPart<IPnPspFrameworkGetItemsWebPartProps> {
35.
36.   private _getListData(): Promise<ISPList[]> {
37.     return pnp.sp.web.lists.getByTitle("EmployeeList").items.get().then((response) => {
38.
39.       return response;
40.     });
41.
42.   }
43.
44.   private _renderListAsync(): void {
45.
46.     if (Environment.type === EnvironmentType.Local) {
47.       this._getMockListData().then((response) => {
48.         this._renderList(response);
49.       });
50.     }
51.     else {
52.       this._getListData()
53.         .then((response) => {
```

```

54.         this._renderList(response);
55.     });
56.   }
57. }
58.
59. private _renderList(items: ISPList[]): void {
60. let html: string = '<table class="TFtable" border=1 width=100% style="border-collapse: collapse;">';
61. html +=
62.   `<th>EmployeeId</th><th>EmployeeName</th><th>Experience</th><th>Location</th>`;
63. items.forEach((item: ISPList) => {
64.   html += `
65.     <tr>
66.       <td>${item.EmployeeId}</td>
67.       <td>${item.EmployeeName}</td>
68.       <td>${item.Experience}</td>
69.       <td>${item.Location}</td>
70.     </tr>
71.   `;
72. });
73. html += `</table>`;
74. const listContainer: Element = this.domElement.querySelector('#spListContainer');
75. listContainer.innerHTML = html;
76.
77. public render(): void {
78.   this.domElement.innerHTML =
79.     `<div class="${styles.helloWorld}">
80.      <div class="${styles.container}">
81.        <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
82.          <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
83.            <span class="ms-font-xl ms-fontColor-white" style="font-size:28px">Welcome to
84.              SharePoint Framework Development using PnP JS Library</span>
85.        <p class="ms-font-1 ms-fontColor-white" style="text-align: left">Demo : Retrieve
86.          Employee Data from SharePoint List</p>
87.        </div>
88.      </div>
89.      <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
90.        <div style="background-color:Black;color:white;text-align: center;font-weight:
91.          bold;font-size:18px;">Employee Details</div>
92.        <br>
93.        <div id="spListContainer" />
94.      </div>
95.    </div>`;
96.   this._renderListAsync();
97. }
98. private _getMockListData(): Promise<ISPList[]> {
99.   return MockHttpClient.get(this.context.pageContext.web.absoluteUrl).then(() => {
100.     const listData: ISPList[] = [
101.       { EmployeeId: 'E123', EmployeeName: 'John', Experience:
102.         'SharePoint', Location: 'India' },
103.       { EmployeeId: 'E567', EmployeeName: 'Martin', Experience:
104.         '.NET', Location: 'Qatar' },
105.       { EmployeeId: 'E367', EmployeeName: 'Luke', Experience:
106.         'JAVA', Location: 'UK' }
107.     ];
108.     return listData;
109.   }) as Promise<ISPList[]>;

```

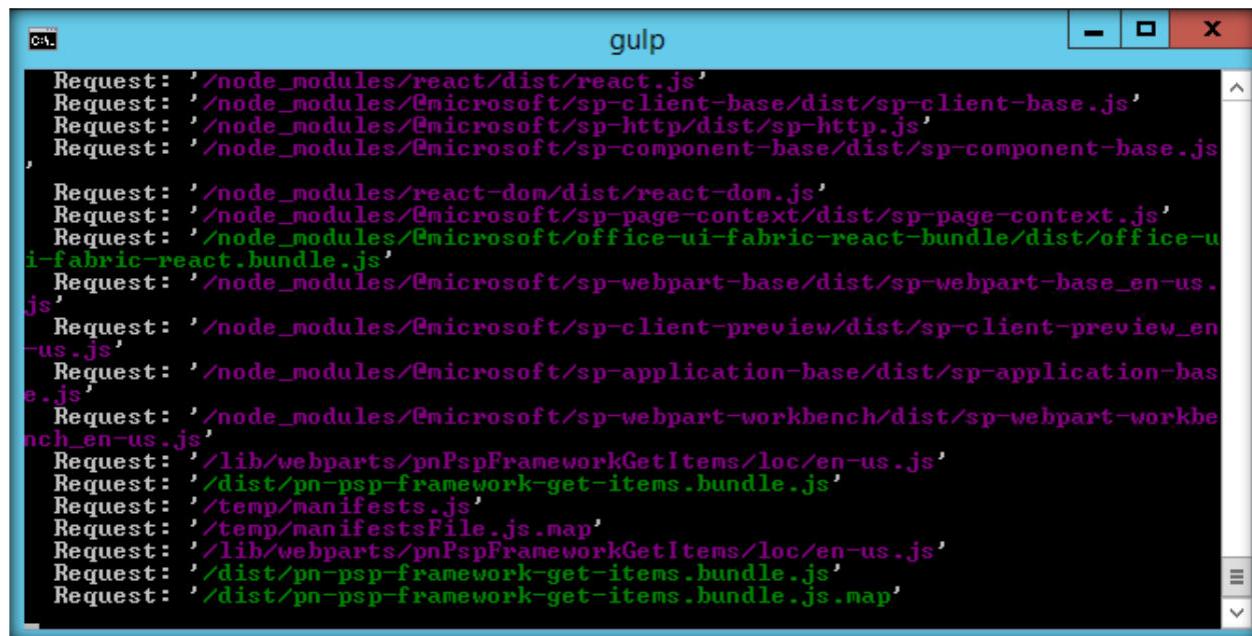
```

107.      }
108.      protected getDataVersion(): Version {
109.        return Version.parse('1.0');
110.      }
111.
112.
113.      protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
114.        return {
115.          pages: [
116.            {
117.              header: {
118.                description: strings.PropertyPaneDescription
119.              },
120.              groups: [
121.                {
122.                  groupName: strings.BasicGroupName,
123.                  groupFields: [
124.                    PropertyPaneTextField('description', {
125.                      label: strings.DescriptionFieldLabel
126.                    })
127.                  ]
128.                }
129.              ]
130.            }
131.          ],
132.        };
133.      }
134.    }

```

Package and Deploy the Solution

Let's run *Gulp serve* to package the solution.

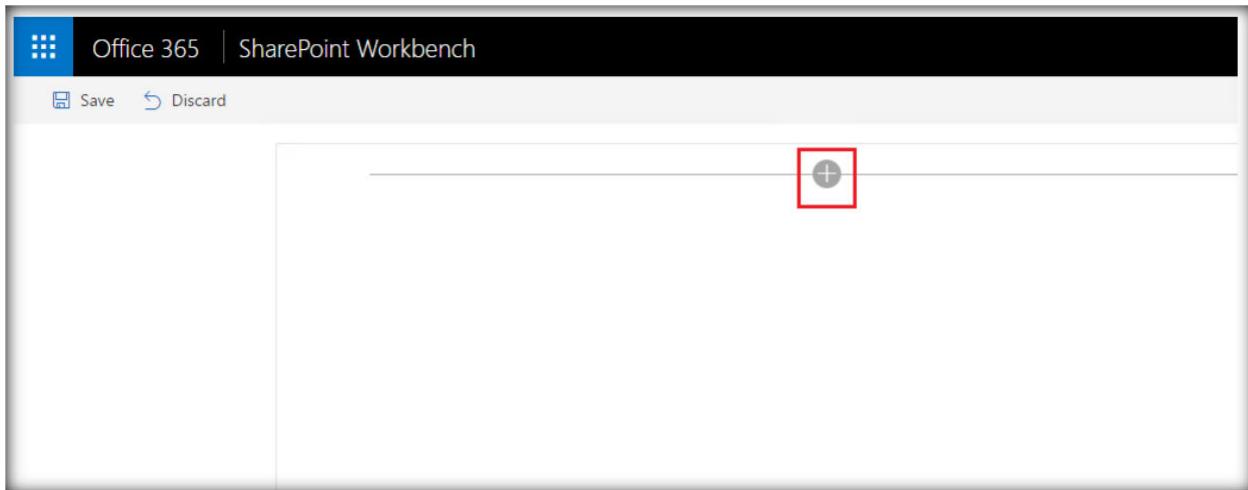


The screenshot shows a terminal window with the title bar 'gulp'. The window displays a list of network requests, likely from a browser developer tools Network tab or a similar monitoring tool. The requests listed are:

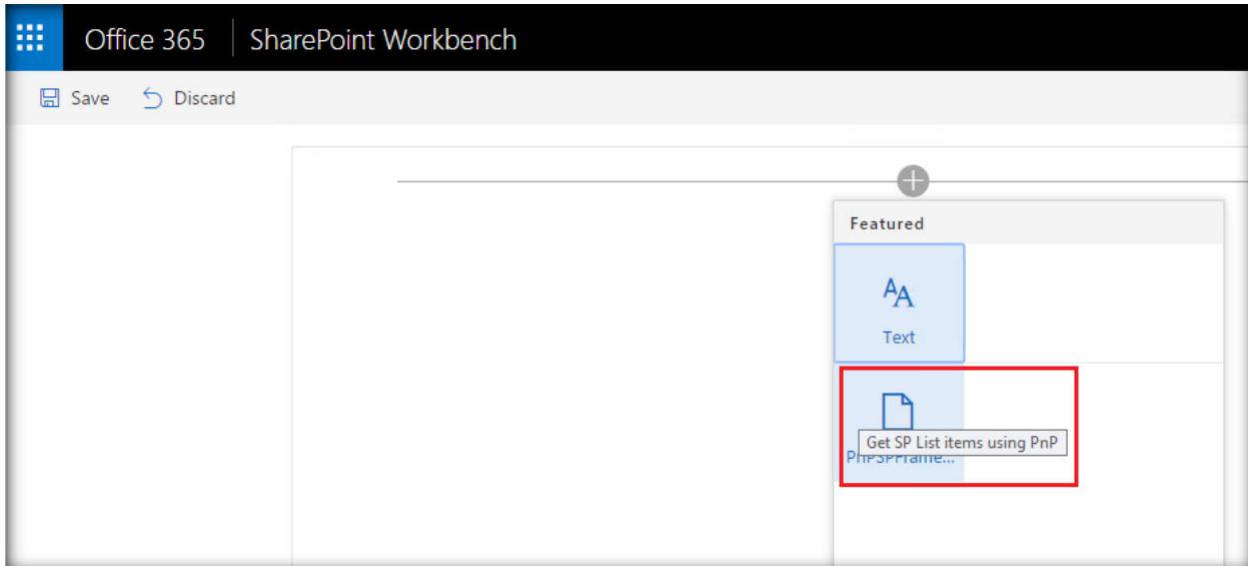
- Request: '/node_modules/react/dist/react.js'
- Request: '/node_modules/@microsoft/sp-client-base/dist/sp-client-base.js'
- Request: '/node_modules/@microsoft/sp-http/dist/sp-http.js'
- Request: '/node_modules/@microsoft/sp-component-base/dist/sp-component-base.js'
- Request: '/node_modules/react-dom/dist/react-dom.js'
- Request: '/node_modules/@microsoft/sp-page-context/dist/sp-page-context.js'
- Request: '/node_modules/@microsoft/office-ui-fabric-react-bundle/dist/office-uifabric-react.bundle.js'
- Request: '/node_modules/@microsoft/sp-webpart-base/dist/sp-webpart-base_en-us.js'
- Request: '/node_modules/@microsoft/sp-client-preview/dist/sp-client-preview_en-us.js'
- Request: '/node_modules/@microsoft/sp-application-base/dist/sp-application-base.js'
- Request: '/node_modules/@microsoft/sp-webpart-workbench/dist/sp-webpart-workbench_en-us.js'
- Request: '/lib/webparts/pnPspFrameworkGetItems/loc/en-us.js'
- Request: '/dist/pn-psp-framework-get-items.bundle.js'
- Request: '/temp/manifests.js'
- Request: '/temp/manifestsFile.js.map'
- Request: '/lib/webparts/pnPspFrameworkGetItems/loc/en-us.js'
- Request: '/dist/pn-psp-framework-get-items.bundle.js'
- Request: '/dist/pn-psp-framework-get-items.bundle.js.map'

Test the Web part in local SharePoint Workbench

Now, we can see the output generated in the local SharePoint Workbench.



Now let's add the web part by clicking on the Plus sign.



Since the environment is local, the mock data has been used to generate the table, as shown below.

The screenshot shows the SharePoint Workbench interface. At the top, there are 'Save' and 'Discard' buttons. Below the header, a blue web part displays the following content:

Welcome to SharePoint Framework
Development using PnP JS Library

Demo : Retrieve Employee Data from SharePoint List

Employee Details

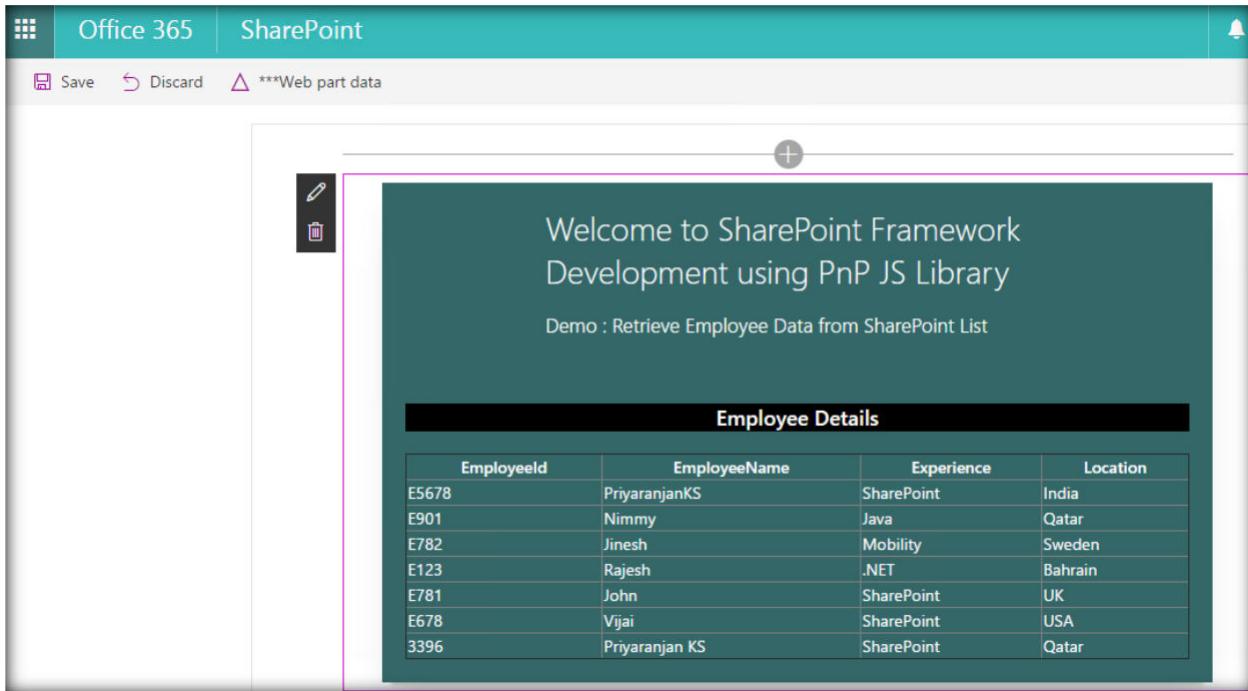
EmployeeId	EmployeeName	Experience	Location
E123	John	SharePoint	India
E567	Martin	.NET	Qatar
E367	Luke	JAVA	UK

Test the Web part in SharePoint Online

Now, let's test the Web part in SharePoint Workbench available in SharePoint Online. This time, the 'EnvironmentType' check will evaluate to SharePoint and the PnP method will be called to retrieve the list items from SharePoint list. Once we have login in to SharePoint Online, we can invoke the workbench by appending the text '_layouts/15/workbench.aspx' to SharePoint Online URL.

The screenshot shows the SharePoint Workbench interface with a teal header. At the top, there are 'Save' and 'Discard' buttons, and a note '***Web part data'. Below the header, a modal window displays a grid of web part icons. One icon, 'PnPSPFrame...', is highlighted with a pink background. A tooltip for this icon reads 'Get SP List items using PnP'. Other icons include 'Image gallery', 'K2 Form Viewer', 'K2 Worklist', 'News headlines', 'News list', 'Office 365 video', 'Power BI (preview)', 'Quick chart', 'Quick links', 'Site activity', and 'Yammer feed'.

It has picked up the list items from the list and displayed it as a table.



EmployeeId	EmployeeName	Experience	Location
E5678	PriyaranjanKS	SharePoint	India
E901	Nimmy	Java	Qatar
E782	Jinesh	Mobility	Sweden
E123	Rajesh	.NET	Bahrain
E781	John	SharePoint	UK
E678	Vijai	SharePoint	USA
3396	Priyaranjan KS	SharePoint	Qatar

The major project files used in this solution has been zipped and uploaded at Microsoft [TechNet Gallery](#). Feel free to download it.

SharePoint List Creation using PnP and SPFx

In this section, we will see how to use PnP JS in SPFx to create and provision a custom list. The major project files used in this solution has been zipped and uploaded at Microsoft [TechNet Gallery](#). Feel free to download it.

We can create the directory, where we will be adding the solution, using the command given below.
`md CreatePnPList`

Let's move to the newly created working directory, using the command.
`cd CreatePnPList`

```
Ch. yo
Your environment has been set up for using Node.js 6.10.0 <x64> and npm.
C:\Users\farmaccount>md createPnPList
C:\Users\farmaccount>cd createPnPList
```

We will then create the client Web part by running the Yeoman SharePoint Generator. `yo @microsoft/sharepoint`

```
C:\Users\farmaccount\createPnPList>yo @microsoft/sharepoint

Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? CreatePnPList
? Where do you want to place the files? Use the current folder
? What framework would you like to start with? No javaScript web framework
A folder with solution name CreatePnPList will be created for you.
? What is your webpart name? CreatePnPList
? What is your webpart description? (CreatePnPList description) List Created using PnP JS and SharePoint Framework
```

This will display the prompt, which we will have to fill up, so as to proceed with the project creation.

- What is your solution name? : Set it to 'CreatePnPList'.

On pressing enter, we will be asked to chose the working folder for the project.

- Where do you want to place your files- Use current folder.
- What framework would you like to start with- Select "No javaScript web framework" for the time being, as this is a sample Web part.
- What is your Webpart name- We will specify it as 'CreatePnPList' and press Enter
- What is your Webpart description- We will specify it as 'List created using PnP JS and SharePoint Framework'.

Node.js command prompt

```
-- clone@0.2.0
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@* (node_modules\@microsof\gulp-core-build-webpack\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
>

=#####
#####
##/ <##:(@)
##/ ##:
##/ /##: <(e) #####
##/ ##:
##/ /##:(@)
#####
**=#####

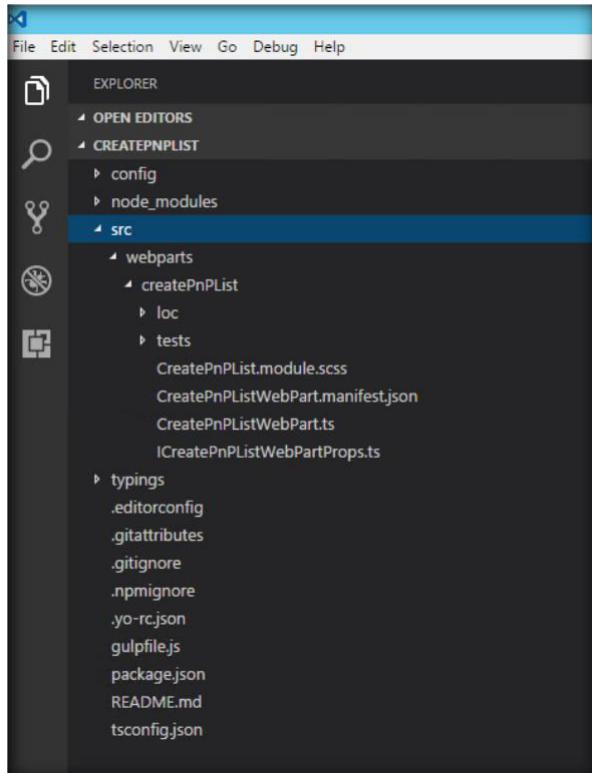
Congratulations!
Solution create-pn-p-list is created.
Run gulp serve to play with it!
```

C:\Users\farmaccount\createPnPList>

Yeoman has started working on the scaffolding of the project. It will install the required dependencies and scaffold the solution files for the 'CreatePnPList' Web part, which will take some time to complete. Once completed, we will get a congratulations message.

Edit the web part

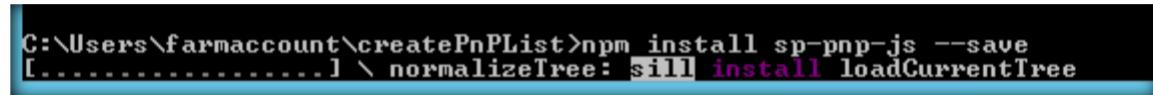
Run 'Code .' to open the project in Visual Studio Code.



Install PnP JS Module

Now we have to load PnP JS file which we will use within the project to create list. We will be using npm to add PnP JS file.

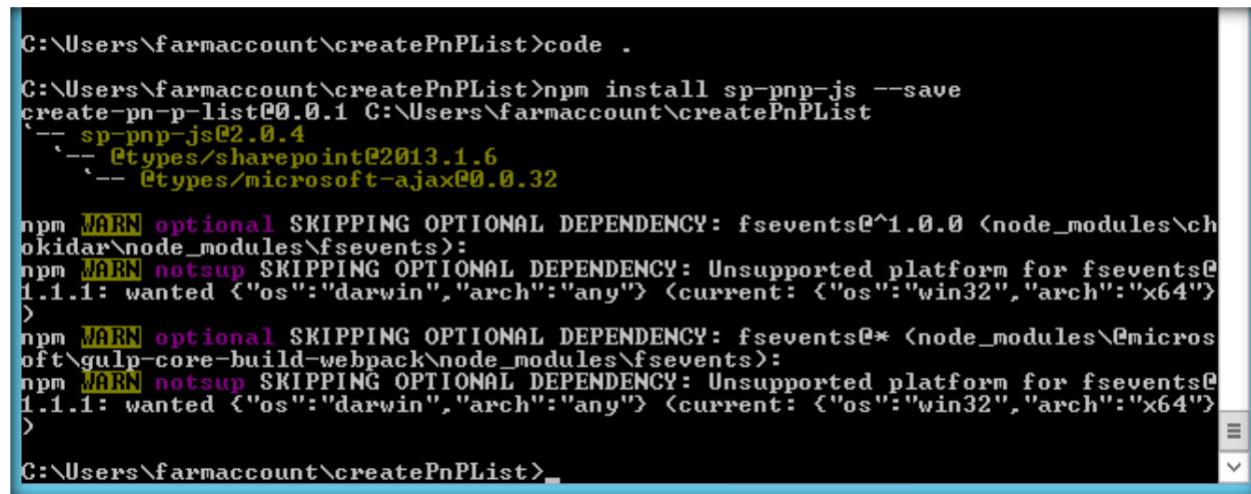
```
npm install sp-pnp-js --save
```



```
C:\Users\farmaccount\createPnPList>npm install sp-pnp-js --save
[.....] \ normalizeTree: sill install loadCurrentTree
```

Thus PnP js has been loaded to the project. We can refer it in the project by using

```
import { Web } from "sp-pnp-js";
```



```
C:\Users\farmaccount\createPnPList>code .

C:\Users\farmaccount\createPnPList>npm install sp-pnp-js --save
create-pn-p-list@0.0.1 C:\Users\farmaccount\createPnPList
  '-- sp-pnp-js@2.0.4
    '-- @types/sharepoint@2013.1.6
      '-- @types/microsoft-ajax@0.0.32

npm MARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 <(node_modules\chokidar\node_modules\fsevents>):
npm MARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} <(current: {"os":"win32","arch":"x64"})>
npm MARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@* <(node_modules\@microsof\ft\gulp-core-build-webpack\node_modules\fsevents)>
npm MARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} <(current: {"os":"win32","arch":"x64"})>

C:\Users\farmaccount\createPnPList>_
```

Create List using PnP method

We can create the list using PnP js method - `spWeb.lists.add` as shown below. We will be creating a custom list named `SPFxPnPList` which has the template id : 100.

```
1. private CreateList(): void {
2.
3.   let spWeb = new Web(this.context.pageContext.web.absoluteUrl);
4.   let spListTitle = "SPFxPnPList";
5.   let spListDescription = "SPFxPnP List";
6.   let spListTemplateId = 100;
7.   let spEnableCT = false;
8.   spWeb.lists.add(spListTitle, spListDescription, spListTemplateId,
9.     spEnableCT).then(function(splist){
10.   document.getElementById("ListCreationStatus").innerHTML += `New List `+
11.     spListTitle+ ` Created`;
```

Once the web part has been created, the status will be updated in the div.
<div id="ListCreationStatus" />

TS File code for Creating the List

The entire code for the TS file is shown below. 'this.CreateList()' method in the render method will call the PnP list creation method and create the SharePoint list.

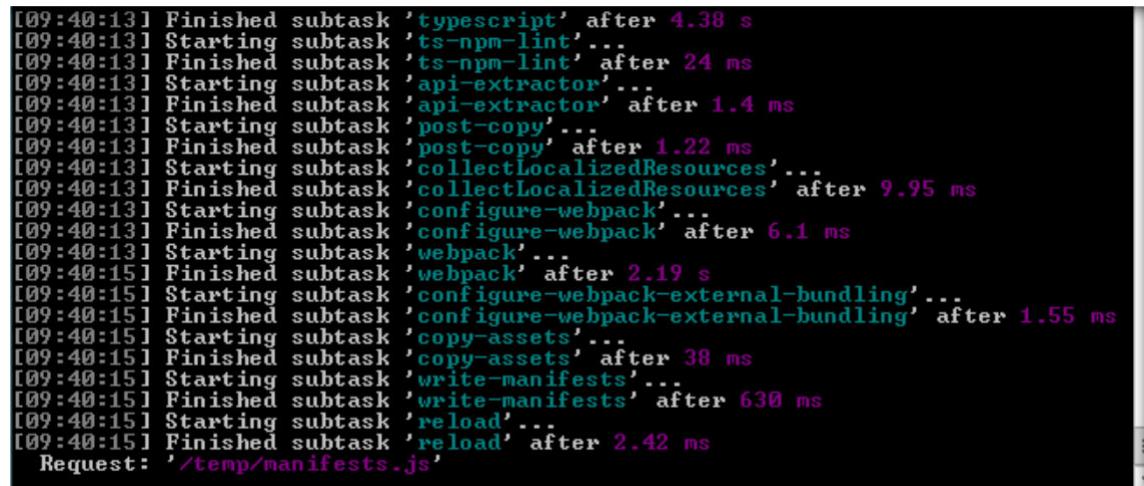
```
1. import { Web } from "sp-pnp-js";
2.
3.
4.
5. import { Version } from '@microsoft/sp-core-library';
6.
7. import {
8.   BaseClientSideWebPart,
9.   IPropertyPaneConfiguration,
10.  PropertyPaneTextField
11. } from '@microsoft/sp-webpart-base';
12. import { escape } from '@microsoft/sp-lodash-subset';
13.
14.
15. import styles from './CreatePnPList.module.scss';
16. import * as strings from 'createPnPListStrings';
17. import { ICreatePnPListWebPartProps } from './ICreatePnPListWebPartProps';
18.
19. export default class CreatePnPListWebPart extends
  BaseClientSideWebPart<ICreatePnPListWebPartProps> {
20.
21. private CreateList(): void {
22.
23.   let spWeb = new Web(this.context.pageContext.web.absoluteUrl);
24.   let spListTitle = "SPFxPnPList";
25.   let spListDescription = "SPFxPnP List";
26.   let spListTemplateId = 100;
27.   let spEnableCT = false;
28.   spWeb.lists.add(spListTitle, spListDescription, spListTemplateId,
    spEnableCT).then(function(splist){
29.     document.getElementById("ListCreationStatus").innerHTML += `New List `+
      spListTitle+ ` Created`;
30.   });
31. }
32.
33. public render(): void {
34.   this.domElement.innerHTML = `
35.     <div class="${styles.helloWorld}>
36.       <div class="${styles.container}>
37.         <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}>
38.           <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
39.             <span class="ms-font-xl ms-fontColor-white" style="font-size:28px">Welcome to
              SharePoint Framework Development using PnP JS Library</span>
40.
41.             <p class="ms-font-1 ms-fontColor-white" style="text-align: left">Demo : Create
              SharePoint List</p>
42.           </div>
43.         </div>
```

```

44.     <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
45. <div style="background-color:Black;color:white;text-align: center;font-weight:
  bold;font-size:18px;">Employee Details</div>
46.     <br>
47. <div id="ListCreationStatus" />
48.   </div>
49. </div>
50. </div>`;
51. this.CreateList();
52. }
53.
54. protected get dataVersion(): Version {
55.   return Version.parse('1.0');
56. }
57.
58. protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
59.   return {
60.     pages: [
61.       {
62.         header: {
63.           description: strings.PropertyPaneDescription
64.         },
65.         groups: [
66.           {
67.             groupName: strings.BasicGroupName,
68.             groupFields: [
69.               PropertyPaneTextField('description', {
70.                 label: strings.DescriptionFieldLabel
71.               })
72.             ]
73.           }
74.         ]
75.       }
76.     ];
77.   };
78. }
79. }

```

Run gulp serve to package the solution.



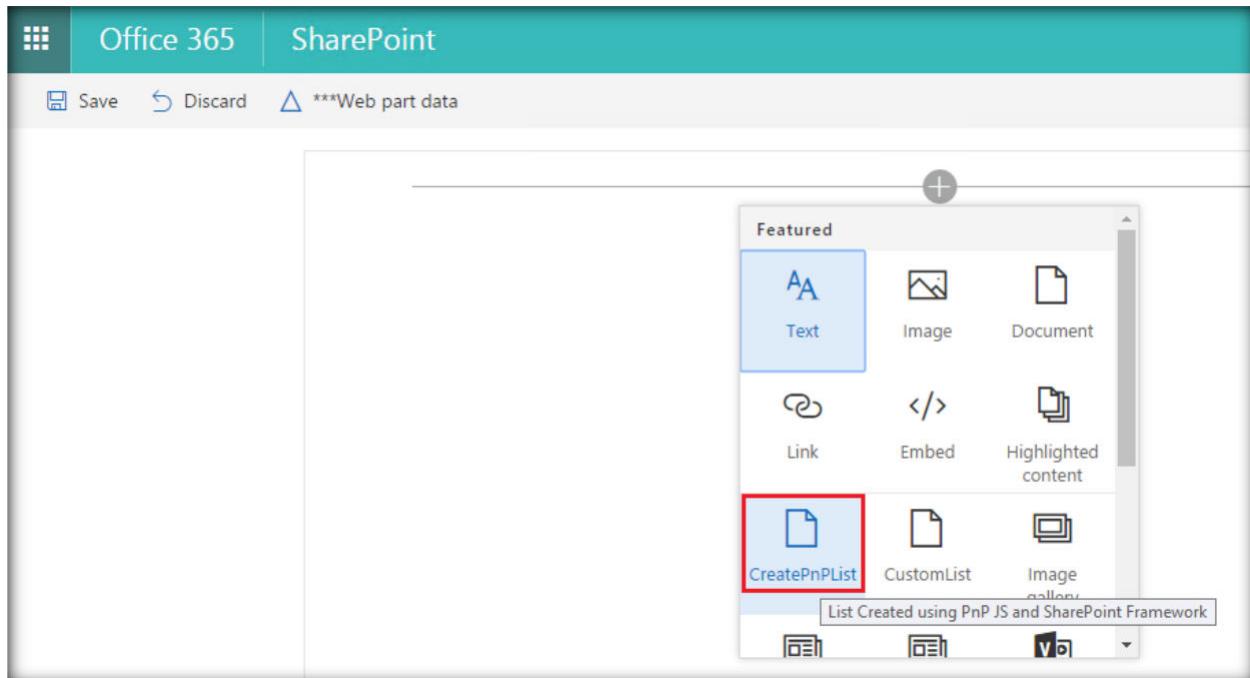
```

[09:40:13] Finished subtask 'typescript' after 4.38 s
[09:40:13] Starting subtask 'ts-npm-lint'...
[09:40:13] Finished subtask 'ts-npm-lint' after 24 ms
[09:40:13] Starting subtask 'api-extractor'...
[09:40:13] Finished subtask 'api-extractor' after 1.4 ms
[09:40:13] Starting subtask 'post-copy'...
[09:40:13] Finished subtask 'post-copy' after 1.22 ms
[09:40:13] Starting subtask 'collectLocalizedResources'...
[09:40:13] Finished subtask 'collectLocalizedResources' after 9.95 ms
[09:40:13] Starting subtask 'configure-webpack'...
[09:40:13] Finished subtask 'configure-webpack' after 6.1 ms
[09:40:13] Starting subtask 'webpack'...
[09:40:15] Finished subtask 'webpack' after 2.19 s
[09:40:15] Starting subtask 'configure-webpack-external-bundling'...
[09:40:15] Finished subtask 'configure-webpack-external-bundling' after 1.55 ms
[09:40:15] Starting subtask 'copy-assets'...
[09:40:15] Finished subtask 'copy-assets' after 38 ms
[09:40:15] Starting subtask 'write-manifests'...
[09:40:15] Finished subtask 'write-manifests' after 630 ms
[09:40:15] Starting subtask 'reload'...
[09:40:15] Finished subtask 'reload' after 2.42 ms
  Request: '/temp/manifests.js'

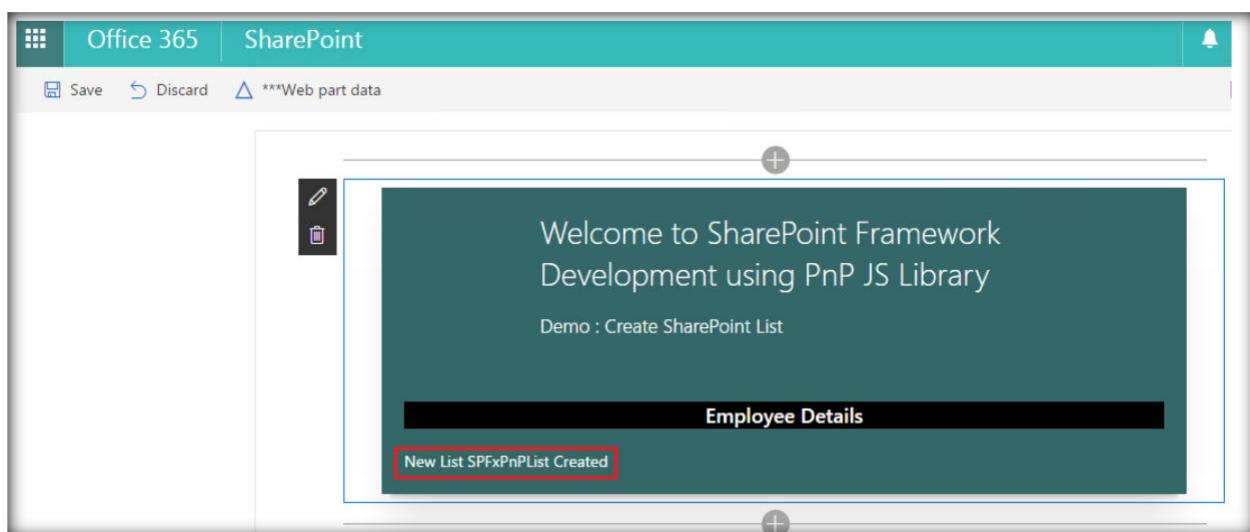
```

Test the Web part in SharePoint Online

Now, let's test the Web part in SharePoint Workbench available in SharePoint Online. Once we have logged in to SharePoint Online, we can invoke the workbench by appending the text '_layouts/15/workbench.aspx' to SharePoint Online URL. Add the webpart to the page by selecting CreatePnPList icon.



This will deploy the list to SharePoint and will show a success message in the UI as shown below.



Heading over to site contents we can see the newly created list.

The screenshot shows the SharePoint Site Contents page. At the top, there is a list of existing lists: "Reusable Content" (3 items, modified 14 months ago) and "SPFxPnPList" (0 items, modified 1 minute ago). The "SPFxPnPList" item is highlighted with a red border. Below this, the main content area displays the "SPFxPnPList" list details. The list title is "SPFxPnPList". There is a blue ribbon icon with a white 'S' and a 'PnP' symbol. To the right of the icon are "EDIT LINKS" and a "new!" button. Below the title, there is a link to "new item or edit this list". The "All Items" button is highlighted with a blue border. A search bar contains the placeholder "Find an item" and a magnifying glass icon. A note below the search bar states, "There are no items to show in this view of the 'SPFxPnPList' list." On the left side of the main content area, there is a sidebar with "EDIT LINKS" and a "Site contents" link.

The major project files used in this solution has been zipped and uploaded at Microsoft [TechNet Gallery](#). Feel free to download it.

Retrieve User Profile Properties using SPFx and PnP JS

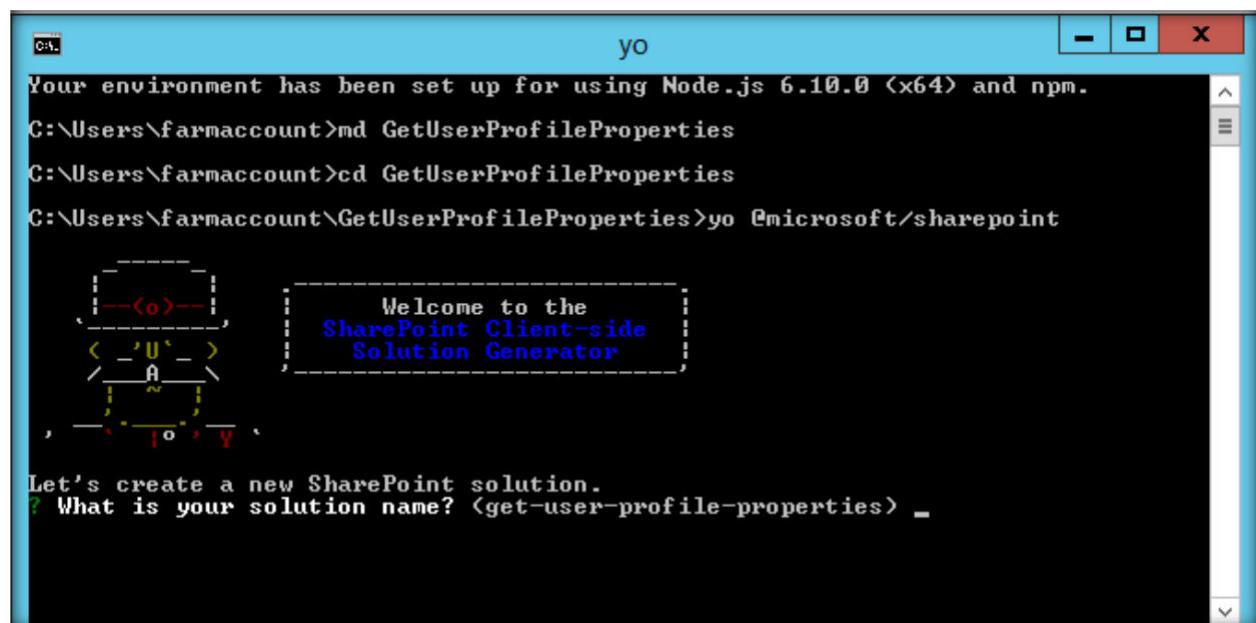
In this section, we will see another example of SPFx and PnP in action, here we will be using this combo to retrieve the user profile properties and display them in the web part. The major project files used in this solution has been zipped and uploaded at Microsoft [TechNet Gallery](#). Feel free to download it.

Create the Web part Project

Spin up Node.js command prompt, using which we will be creating the Web part project structure. We can create the directory, where we will be adding the solution, using the command given below.
md GetUserProfileProperties

Let's move to the newly created working directory, using the command.

cd GetUserProfileProperties



```
yo
Your environment has been set up for using Node.js 6.10.0 (x64) and npm.
C:\Users\farmaccount>md GetUserProfileProperties
C:\Users\farmaccount>cd GetUserProfileProperties
C:\Users\farmaccount\GetUserProfileProperties>yo @microsoft/sharepoint

Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? <get-user-profile-properties> -
```

We will then create the client Web part by running the Yeoman SharePoint Generator. *yo @microsoft/sharepoint*

```

yo
Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? GetUserProfileProperties
? Where do you want to place the files? Use the current folder
? What framework would you like to start with? No JavaScript web framework
A folder with solution name GetUserProfileProperties will be created for you.
? What is your webpart name? GetUserProfileProperties
? What is your webpart description? <GetUserProfileProperties description> Retrieve User Properties using SharePoint Framework

```

This will display the prompt, which we will have to fill up, so as to proceed with the project creation.

- What is your solution name? : Set it to 'GetUserProfileProperties'.

On pressing enter, we will be asked to chose the working folder for the project.

- Where do you want to place your files- Use current folder.
- What framework would you like to start with- Select "No JavaScript web framework" for the time being, as this is a sample Web part.
- What is your Webpart name- We will specify it as 'GetUserProfileProperties' and press Enter
- What is your Webpart description- We will specify it as 'Retrieve User Properties using SharePoint Framework'.

```

Node.js command prompt
-- clone@0.2.0

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@* (node_modules\@microsoft\gulp-core-build-webpack\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

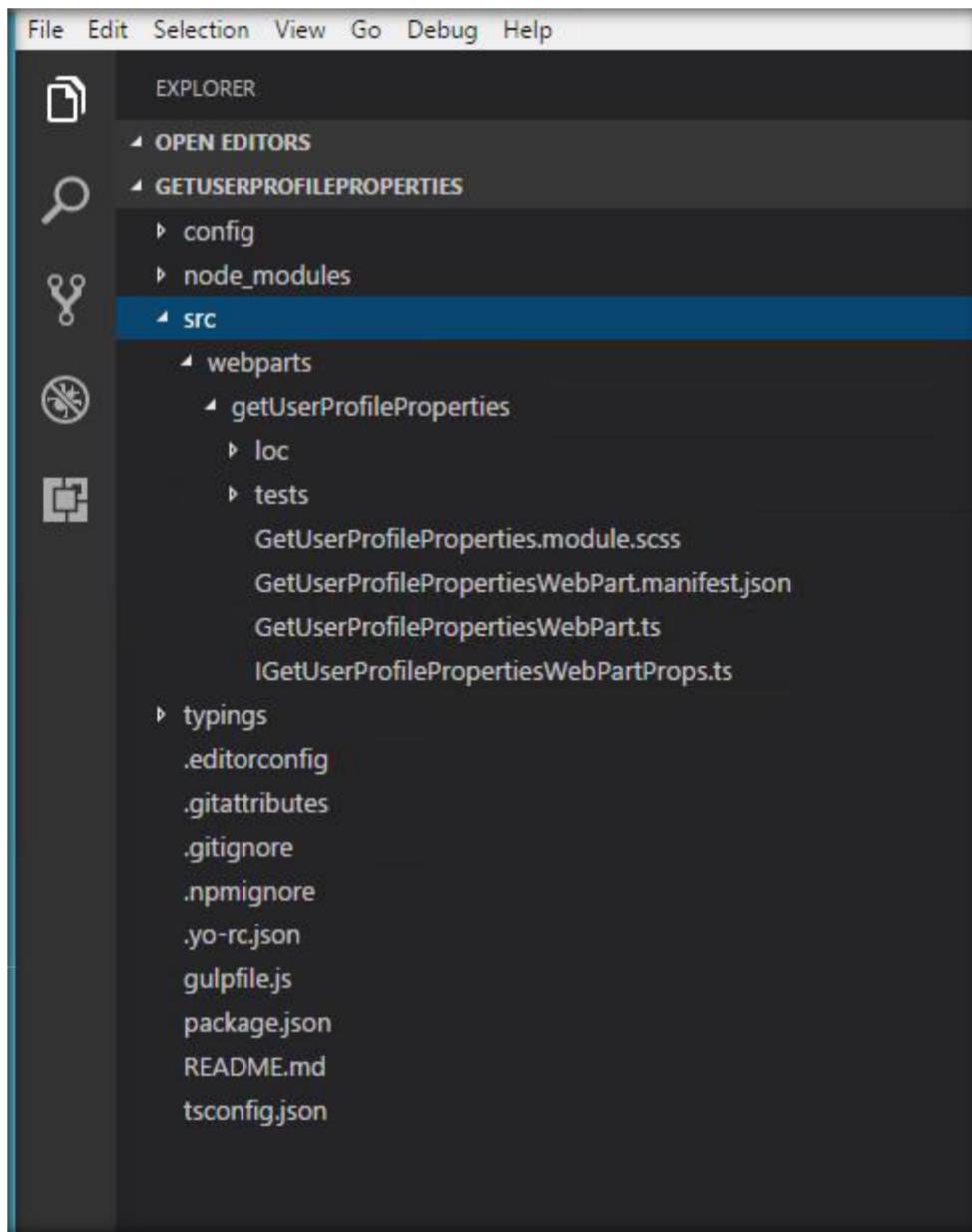
=+#####
# ##<@>
# ##/ <@>
# ## /<@>
# ####<@>
# ## /<@>
#####
Congratulation!
Solution get-user-profile-properties is created.
Run gulp serve to play with it!

```

C:\Users\farmaccount\GetUserProfileProperties>

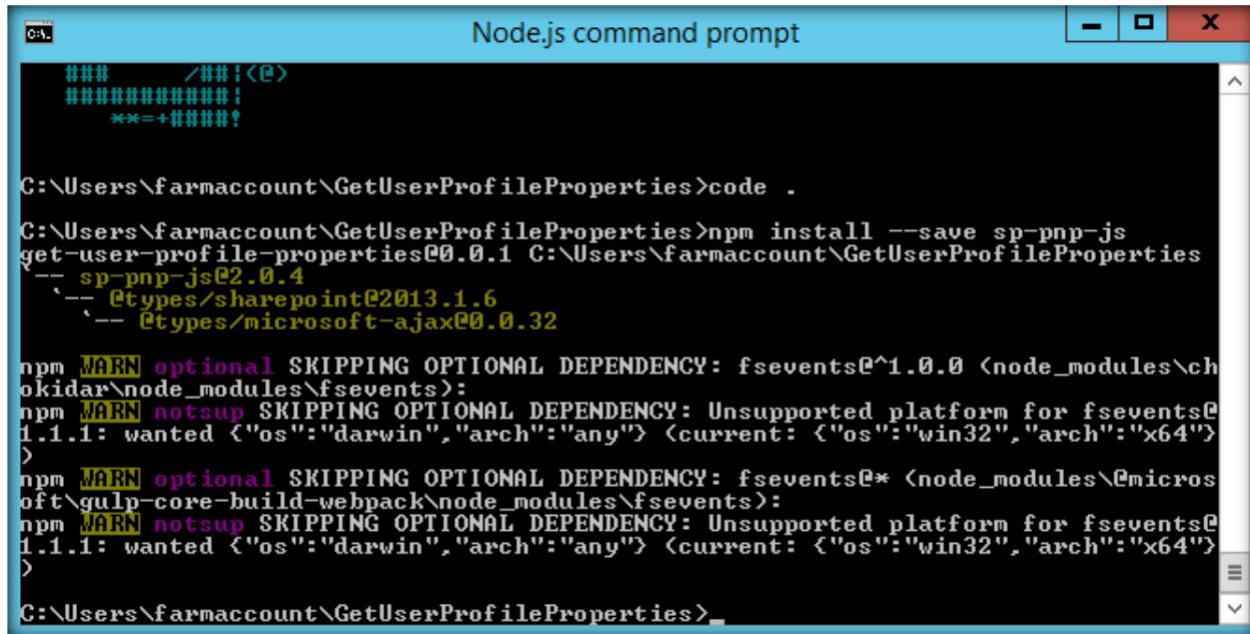
Edit the web part

Run Code . to open the project in Visual Studio Code



Now we have to load PnP JS file which we will use within the project to create list. We will be using npm to add PnP JS file as shown below:

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



```
Node.js command prompt
C:\Users\farmaccount\GetUserProfileProperties>code .
C:\Users\farmaccount\GetUserProfileProperties>npm install --save sp-pnp-js
get-user-profile-properties@0.0.1 C:\Users\farmaccount\GetUserProfileProperties
`-- sp-pnp-js@2.0.4
  `-- @types/sharepoint@2013.1.6
    `-- @types/microsoft-ajax@0.0.32

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@* (node_modules\@microsoft\gulp-core-build-webpack\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
C:\Users\farmaccount\GetUserProfileProperties>
```

Retrieve User Profile data

In order to use PnP methods, we can refer the PnP file in the project as below:

```
import * as pnp from 'sp-pnp-js';
```

We will then make use of the below function to fetch the user profile properties of the user and display it within the web part. `pnp.sp.profiles.myProperties.get()` will return the current user's profile properties which can be iterated to return the required information.

```
1. private GetUserProperties(): void {

  pnp.sp.profiles.myProperties.get().then(function(result)
    { var userProperties = result.UserProfileProperties;
      var userPropertyValues = "";
      userProperties.forEach(function(property) {
        userPropertyValues += property.Key + " - " + property.Value +
        "<br/>"; });
      document.getElementById("spUserProfileProperties").innerHTML = userPropertyValues;
    }).catch(function(error) {
      console.log("Error: " +
      error); });

}
```

TS File content to retrieve User Profile Data

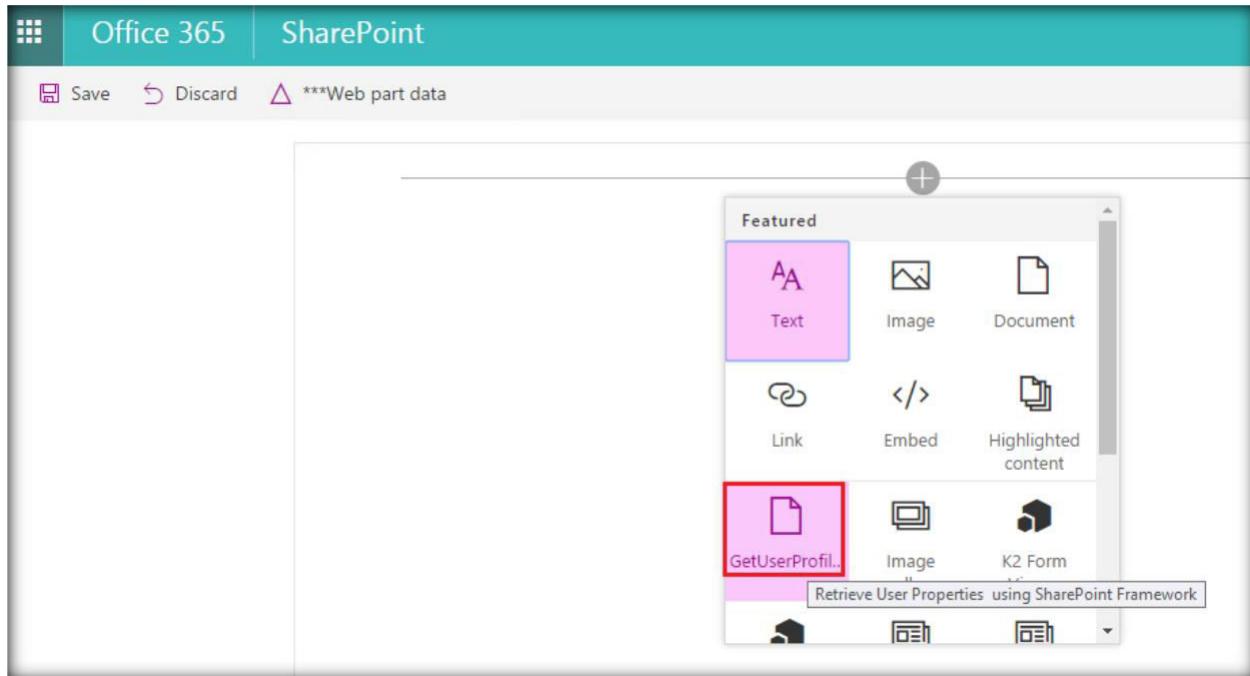
The entire TS file contents is as shown below. this.GetUserProperties() in the render method will call the function that will call the function that gets the User Profile properties of the user. It will be then displayed within the div element declared in the render method.

```
1. import * as pnp from 'sp-pnp-js';
2.
3. import { Version } from '@microsoft/sp-core-library';
4. import {
5.   BaseClientSideWebPart,
6.   IPropertyPaneConfiguration,
7.   PropertyPaneTextField
8. } from '@microsoft/sp-webpart-base';
9. import { escape } from '@microsoft/sp-lodash-subset';
10.
11. import styles from './GetUserProfileProperties.module.scss';
12. import * as strings from 'getUserProfilePropertiesStrings';
13. import { I GetUserProfilePropertiesWebPartProps }
14.   from './I GetUserProfilePropertiesWebPartProps';
14.
15. export default class GetUserProfilePropertiesWebPart extends
16.   BaseClientSideWebPart<IGetUserProfilePropertiesWebPartProps> {
16.
17. private GetUserProperties(): void {
18.
19.   pnp.sp.profiles.myProperties.get().then(function(result) {
20.     var userProperties = result.UserProfileProperties;
21.     var userPropertyValues = "";
22.     userProperties.forEach(function(property) {
23.       userPropertyValues += property.Key + " - " + property.Value + "<br/>";
24.     });
25.     document.getElementById("spUserProfileProperties").innerHTML = userPropertyValues;
26.   }).catch(function(error) {
27.     console.log("Error: " + error);
28.   });
29.
30. }
31.
32. public render(): void {
33.
34.   this.domElement.innerHTML =
35.     `<div class="${styles.helloWorld}">
36.       <div class="${styles.container}">
37.         <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
38.           <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
39.             <span class="ms-font-xl ms-fontColor-white" style="font-size:28px">Welcome to
40.               SharePoint Framework Development using PnP JS Library</span>
40.
41. <p class="ms-font-1 ms-fontColor-white" style="text-align: left">Demo : Retrieve User
42.   Profile Properties</p>
42.   </div>
43.   </div>
44.   <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
45.     <div style="background-color:Black;color:white;text-align: center;font-weight:
46.       bold;font-size:18px;">User Profile Details</div>
46.     <br>
47.   <div id="spUserProfileProperties" />
```

```
48.      </div>
49.    </div>
50.  </div>`;
51. this.GetUserProperties();
52. }
53.
54. protected get dataVersion(): Version {
55.   return Version.parse('1.0');
56. }
57.
58. protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
59.   return {
60.     pages: [
61.       {
62.         header: {
63.           description: strings.PropertyPaneDescription
64.         },
65.         groups: [
66.           {
67.             groupName: strings.BasicGroupName,
68.             groupFields: [
69.               PropertyPaneTextField('description', {
70.                 label: strings.DescriptionFieldLabel
71.               })
72.             ]
73.           }
74.         ]
75.       }
76.     ]
77.   };
78. }
79. }
```

Test the Web part in SharePoint Online

Now, let's test the Web part in SharePoint Workbench available in SharePoint Online. Once we have login in to SharePoint Online, we can invoke the workbench by appending the text '`_layouts/15/workbench.aspx`' to SharePoint Online URL. Add the webpart to the page by selecting GetUserProfile icon.



This will add the web part to the page and it will fetch the user profile details of the user and display it.

A screenshot of a SharePoint page. The title bar shows 'Office 365' and 'SharePoint'. Below the title bar, there are 'Save' and 'Discard' buttons, and a message '***Web part data'. The main content area displays a welcome message: 'Welcome to SharePoint Framework Development using PnP JS Library' and a subtitle 'Demo : Retrieve User Profile Properties'. Below this, there is a section titled 'User Profile Details' containing a list of user properties. The properties listed are: UserProfile_GUID - 01367498-b06b-40e2-b706-d5bb2496af64, SID - i:0\h|membership|1003bffd968bc779@live.com, ADGuid - System.Byte[], AccountName - i:0#.\|membership|priyanjan@sharepointchronicle.com, FirstName - Priyanjan, SPS-PhoneticFirstName - , LastName - KS, SPS-PhoneticLastName - , PreferredName - Priyan, SPS-PhoneticDisplayName - , WorkPhone - 8281671563, Department - SP-DEP, Title - , SPS-Department - , Manager - , AboutMe - SharePoint Consultant, PersonalSpace - /personal/priyanjan_sharepointchronicle_com/

The major project files used in this solution has been zipped and uploaded at Microsoft [TechNet Gallery](#). Feel free to download it.

Retrieve SharePoint Search Results using SPFx webpart

In this section, we will try to work with SharePoint Search and retrieve the search results based on a keyword using SPFx and PnP JS

Create the Web part Project

Spin up Node.js command prompt, using which we will be creating the Web part project structure. We can create the directory, where we will be adding the solution, using the command given below.

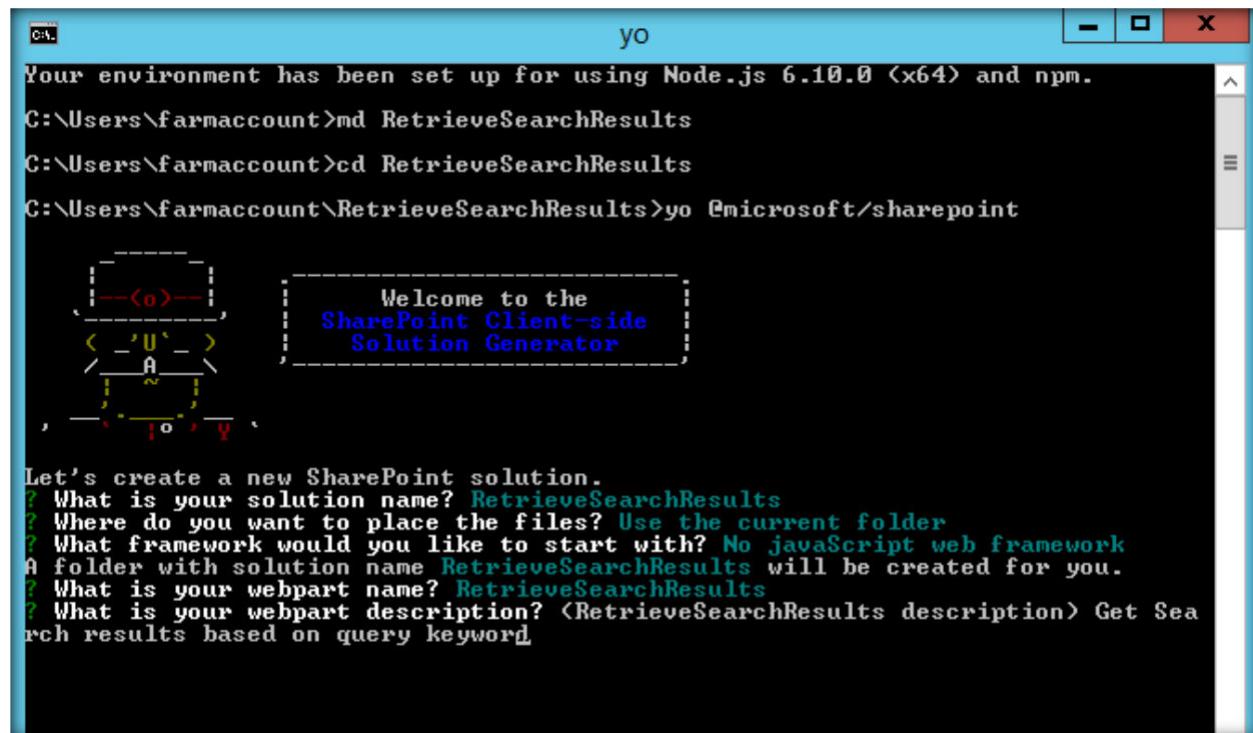
```
md RetrieveSearchResults
```

Let's move to the newly created working directory, using the command.

```
cd RetrieveSearchResults
```

We will then create the client Web part by running the Yeoman SharePoint

```
Generator. yo @microsoft/sharepoint
```



The screenshot shows a Windows Command Prompt window titled "yo". The command history and output are as follows:

```
Your environment has been set up for using Node.js 6.10.0 (x64) and npm.  
C:\Users\farmaccount>md RetrieveSearchResults  
C:\Users\farmaccount>cd RetrieveSearchResults  
C:\Users\farmaccount\RetrieveSearchResults>yo @microsoft/sharepoint
```

A decorative logo consisting of a stylized tree or branching structure is displayed between the command history and the generator welcome message.

>Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? RetrieveSearchResults
? Where do you want to place the files? Use the current folder
? What framework would you like to start with? No javaScript web framework
A folder with solution name RetrieveSearchResults will be created for you.
? What is your webpart name? RetrieveSearchResults
? What is your webpart description? <RetrieveSearchResults description> Get Sea
rch results based on query keyword

Node.js command prompt

```

|   '-- readable-stream@1.0.34
|     '-- unique-stream@1.0.0
+-- glob-watcher@0.0.6
  '-- gaze@0.5.2
    '-- globule@0.1.0
      '-- glob@3.1.21
        '-- graceful-fs@1.2.3
          '-- inherits@1.0.2
        '-- lodash@1.0.2
          '-- minimatch@0.2.14
+-- graceful-fs@3.0.11
  '-- natives@1.1.0
+-- strip-bom@1.0.0
  '-- first-chunk-stream@1.0.0
  '-- is-utf8@0.2.1
+-- through2@0.6.5
  '-- readable-stream@1.0.34
    '-- isarray@0.0.1
+-- vinyl@0.4.6
  '-- clone@0.2.0

npm [WARN optional] SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 <node_modules\chokidar\node_modules\fsevents>:
npm [WARN notsup] SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} <current: {"os":"win32","arch":"x64"}>
npm [WARN optional] SKIPPING OPTIONAL DEPENDENCY: fsevents@* <node_modules\@microsoft\gulp-core-build-webpack\node_modules\fsevents>:
npm [WARN notsup] SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} <current: {"os":"win32","arch":"x64"}>

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

C:\Users\farmaccount\RetrieveSearchResults>

Run Code . to create scaffolding of the project structure.

```

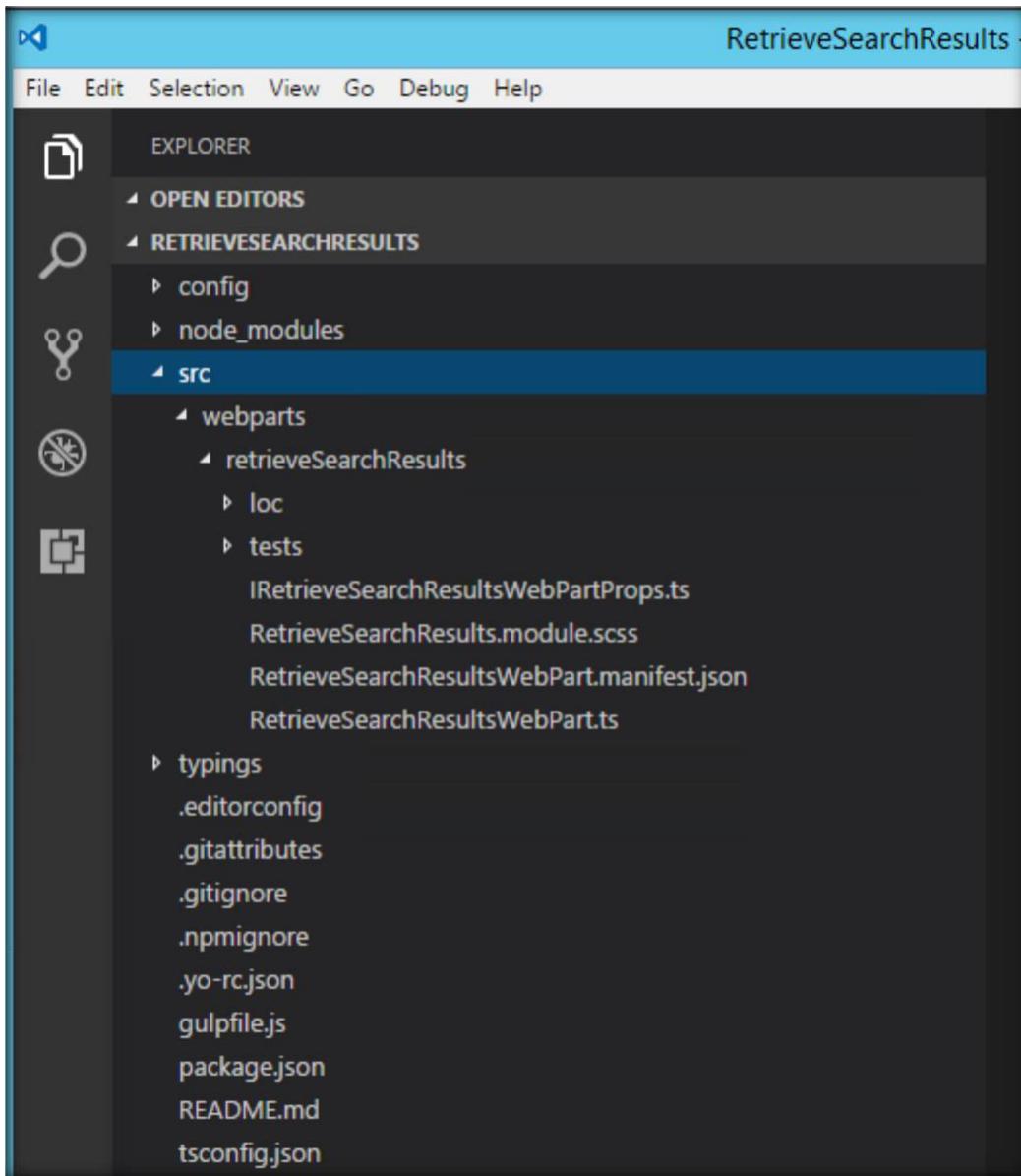
C:\Users\farmaccount\RetrieveSearchResults>code .

C:\Users\farmaccount\RetrieveSearchResults>npm install --save sp-pnp-js
retrieve-search-results@0.0.1 C:\Users\farmaccount\RetrieveSearchResults
  '-- sp-pnp-js@2.0.4
    '-- @types/sharepoint@2013.1.6
      '-- @types/microsoft-ajax@0.0.32

npm [WARN optional] SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 <node_modules\chokidar\node_modules\fsevents>:
npm [WARN notsup] SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} <current: {"os":"win32","arch":"x64"}>
npm [WARN optional] SKIPPING OPTIONAL DEPENDENCY: fsevents@* <node_modules\@microsoft\gulp-core-build-webpack\node_modules\fsevents>:
npm [WARN notsup] SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} <current: {"os":"win32","arch":"x64"}>

C:\Users\farmaccount\RetrieveSearchResults>
```

The generated project structure will look like below



Retrieve Search Results

Now we have to load PnP JS file which we will use within the project to retrieve the search results. We will be using npm to add PnP JS file as shown below.

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

we can then reference the PnP js file in the project by including the below line to the top of the TS file

```
import * as pnp from 'sp-pnp-js';
import { SearchQuery, SearchResults } from "sp-pnp-js";
```

We can get the search results using the below function. Here we will be using PnP method, 'pnp.sp.search' which accepts the search keyword as the parameter. It will return the search results which we will display as a table.

```
1. private GetSearchresults(): void {
2.
3.   pnp.sp.search("SharePoint").then((result : SearchResults) => {
4.     var props = result.PrimarySearchResults;
5.     debugger;
6.
7.     var propValue = "";
8.     var counter = 1;
9.     props.forEach(function(object) {
10.       propValue += counter++ +'. Title - ' +object.Title +"  
"+"Rank - " + object.Rank
11.       +"<br/>"+"File Type - " + object.FileType+"<br/>"+"Original Path - "
12.       +object.OriginalPath +"  
"+ "Summary - "+ object.HitHighlightedSummary +
13.       "<br/>"+"<br/>";
14.     });
15.     document.getElementById("spSearchresults").innerHTML = propValue;
16.   }).catch(function(err) {
17.     console.log("Error: " + err);
18.   });
19. }
```

TS File contents for displaying the retrieved search results

The entire TS file contents is as shown below, 'this.GetSearchresults()' in the render method will call the function that will retrieve the search results based on the keyword. It will be then displayed within the div element declared in the render method.

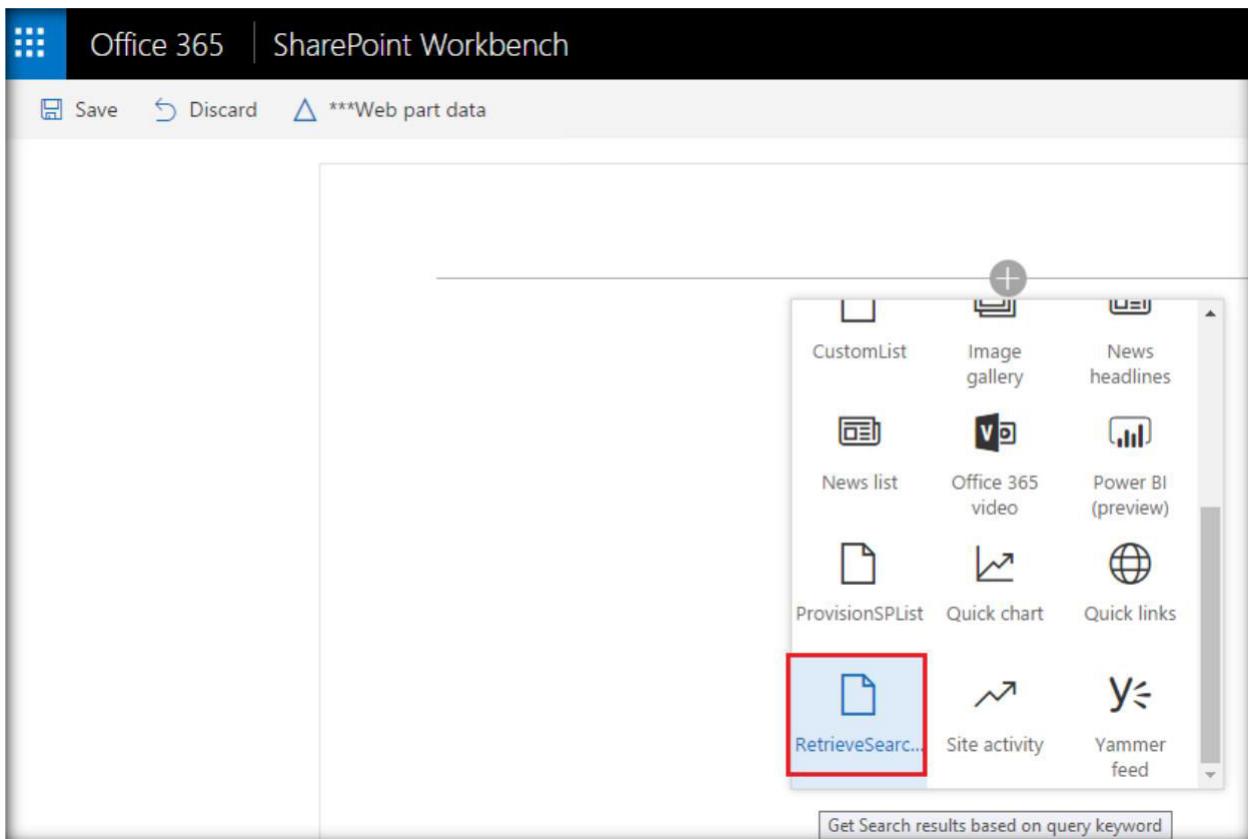
```
1. import * as pnp from 'sp-pnp-js';
2. import { SearchQuery, SearchResults } from "sp-pnp-js";
3.
4. import { Version } from '@microsoft/sp-core-library';
5. import {
6.   BaseClientSideWebPart,
7.   IPropertyPaneConfiguration,
8.   PropertyPaneTextField
9. } from '@microsoft/sp-webpart-base';
10.
11. import { escape } from '@microsoft/sp-lodash-subset';
12.
13. import styles from './RetrieveSearchResults.module.scss';
14. import * as strings from 'retrieveSearchResultsStrings';
15. import { IRetrieveSearchResultsWebPartProps } from
16.   './IRetrieveSearchResultsWebPartProps';
17. export default class RetrieveSearchResultsWebPart extends
18.   BaseClientSideWebPart<IRetrieveSearchResultsWebPartProps> {
19.   private GetSearchresults(): void {
```

```
20. pnp.sp.search("SharePoint").then((result : SearchResults) => {
21.   var props = result.PrimarySearchResults;
22.   debugger;
23.
24.   var propValue = "";
25.   var counter = 1;
26.   props.forEach(function(object) {
27.     propValue += counter++ +'. Title - ' +object.Title +"<br/>"+"Rank - " + object.Rank
    +"<br/>"+"File Type - " + object.FileType+"<br/>"+ "Original Path - "
    +object.OriginalPath +"<br/>"+ "Summary - "+ object.HitHighlightedSummary +
    "<br/>"+<br/>";
28.   });
29.   document.getElementById("spSearchresults").innerHTML = propValue;
30. }).catch(function(err) {
31.   console.log("Error: " + err);
32. });
33.
34. }
35.
36.
37. public render(): void {
38.
39.   this.domElement.innerHTML = `
40.     <div class="${styles.helloWorld}">
41.       <div class="${styles.container}">
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.             <span class="ms-font-xl ms-fontColor-white" style="font-size:28px">Welcome to
    SharePoint Framework Development using PnP JS Library</span>
45.
46.           <p class="ms-font-l ms-fontColor-white" style="text-align: left">Demo : Retrieve
    SharePoint Search Result</p>
47.         </div>
48.       </div>
49.       <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
50.         <div style="background-color:Black;color:white;text-align: center;font-weight:
    bold;font-size:18px;">Search Results </div>
51.         <br>
52.       <div id="spSearchresults" />
53.     </div>
54.   </div>
55. `;
56.   this.GetSearchresults();
57. }
58.
59. protected getDataVersion(): Version {
60.   return Version.parse('1.0');
61. }
62.
63. protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
64.   return {
65.     pages: [
66.       {
67.         header: {
68.           description: strings.PropertyPaneDescription
69.         },
70.         groups: [
71.           {
72.             groupName: strings.BasicGroupName,
73.             groupFields: [
74.               PropertyPaneTextField('description', {
```

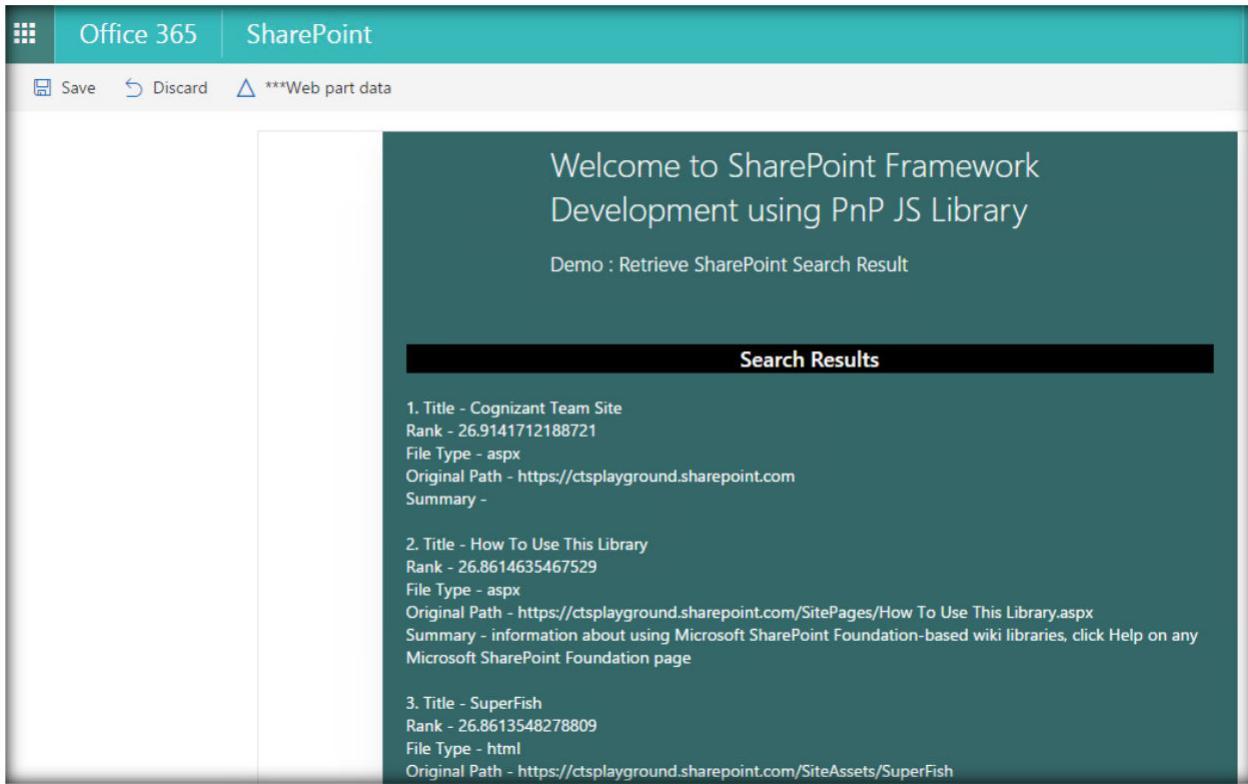
```
75.           label: strings.DescriptionFieldLabel
76.       })
77.   }
78. }
79. ]
80. ]
81. ];
82. }
83. }
84. }
```

Test the Web part in SharePoint Online

Now, let's test the Web part in SharePoint Workbench available in SharePoint Online. Once we have login in to SharePoint Online, we can invoke the workbench by appending the text '_layouts/15/workbench.aspx' to SharePoint Online URL. Add the webpart to the page by selecting 'RetrieveSearchResults' icon.



Thus, the search results have been retrieved and it has been listed in the web part as shown below.



The major project files used in this solution has been zipped and uploaded at Microsoft [TechNet Gallery](#). Feel free to download it.

Implement SharePoint List item CRUD using SPFx and PnP JS

In this section, we will see how to create a webpart that does Create/Read/Updated/Delete operations against SharePoint List items using SPFx and PnP JS. The major project files used in this solution has been zipped and uploaded at Microsoft [TechNet Gallery](#). Feel free to download it.

Create the Web part Project

Spin up Node.js command prompt, using which we will be creating the Web part project structure. We can create the directory, where we will be adding the solution, using the command given below.

```
md PnPSPCRUD
```

Let's move to the newly created working directory, using the command.
`cd PnPSPCRUD`

We will then create the client Web part by running the Yeoman SharePoint Generator. `yo @microsoft/sharepoint`

This will display the prompt, which we will have to fill up, so as to proceed with the project creation.

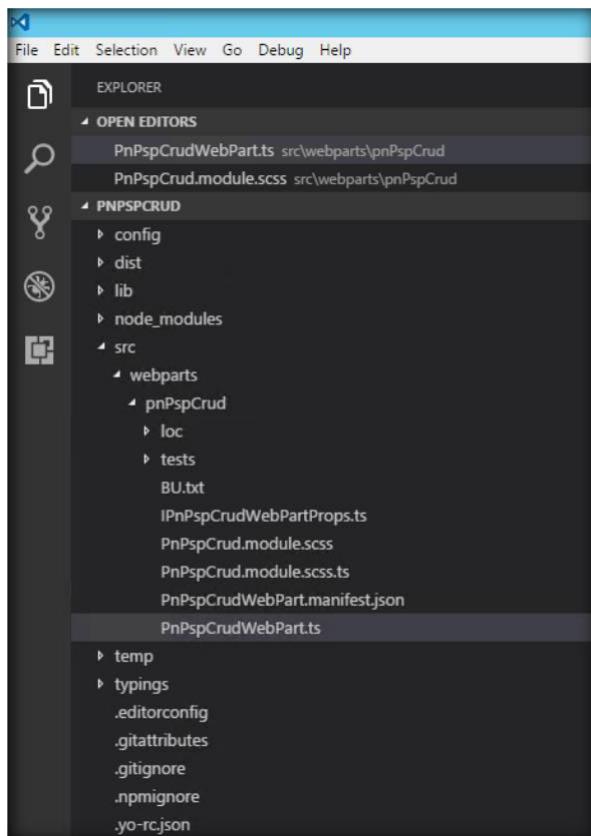
- What is your solution name? : Set it to 'PnPSPCRUD'.

On pressing enter, we will be asked to chose the working folder for the project.

- Where do you want to place your files- Use current folder.
- What framework would you like to start with- Select "No javaScript web framework" for the time being, as this is a sample Web part.
- What is your Webpart name- We will specify it as 'PnPSPCRUD' and press Enter
- What is your Webpart description- We will specify it as this Webpart will perform CRUD operations using PnP and SPFx

Edit the web part

Run `Code .` to create the scaffolding and open the project in Visual Studio Code



Now we have to load PnP JS file which we will use within the project to create list. We will be using npm to add PnP JS file.

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

Implement CRUD using PnP JS

In order to use PnP methods, we can refer the PnP file in the project as below :

```
import * as pnp from 'sp-pnp-js';
```

We will then add CRUD buttons in the render method so that the UI looks as below.

The screenshot shows a SharePoint page titled "Add SharePoint List Items". At the top, there are three input fields: "EmployeeName", "Experience", and "Location", followed by a "Add" button. Below this, a section titled "Update/Delete SharePoint List Items" contains a single input field "EmployeeId" and two buttons: "Update" and "Delete". At the bottom, there is a table with four columns: EmployeeId, EmployeeName, Experience, and Location. The table has three rows with data: Row 4 (EmployeeId 4, EmployeeName Rajesh, Experience .NET, Location Bahrain), Row 5 (EmployeeId 5, EmployeeName John, Experience SharePoint, Location UK), and Row 6 (EmployeeId 6, EmployeeName Vijai, Experience SharePoint, Location USA).

EmployeeId	EmployeeName	Experience	Location
4	Rajesh	.NET	Bahrain
5	John	SharePoint	UK
6	Vijai	SharePoint	USA

Each button will have an event listener which will be invoked on button click.

1. <button id="AddItem" type="submit" >Add</button>
2. <button id="UpdateItem" type="submit" >Update</button>
3. <button id="DeleteItem" type="submit" >Delete</button>

The event listeners will be added as :

1. private AddEventListeners() : void{
2. document.getElementById('AddItem').addEventListener('click', ()=>this.AddItem());
3. document.getElementById('UpdateItem').addEventListener('click', ()=>this.UpdateItem());
4. document.getElementById('DeleteItem').addEventListener('click', ()=>this.DeleteItem());
5. }

Each of these methods will use PnP to implement the CRUD operations as:

```

1. AddItem()
2. {
3.
4.     pnp.sp.web.lists.getByTitle('EmployeeList').items.add({
5.         EmployeeName : document.getElementById('EmployeeName')["value"],
6.         Experience : document.getElementById('Experience')["value"],
7.         Location:document.getElementById('Location')["value"]
8.     });
9.     alert("Record with Employee Name : "+
10.           document.getElementById('EmployeeName')["value"] + " Added !");
11. }
12.
13. UpdateItem()
14. {
15.
16.     var id = document.getElementById('EmployeeId')["value"];
17.     pnp.sp.web.lists.getByTitle("EmployeeList").items.getById(id).update({
18.         EmployeeName : document.getElementById('EmployeeName')["value"],
19.         Experience : document.getElementById('Experience')["value"],
20.         Location:document.getElementById('Location')["value"]
21.     });
22.     alert("Record with Employee Name : "+ document.getElementById('EmployeeName')["value"]
+ " Updated !");
23. }
24.
25. DeleteItem()
26. {
27.
28.     pnp.sp.web.lists.getByTitle("EmployeeList").items.getById(document.getElementById('EmployeeId')["value"]).delete();
29.     alert("Record with Employee ID : "+ document.getElementById('EmployeeId')["value"] + " Deleted !");
}

```

TS File contents for implementing CRUD using PnP

The entire TS file contents is as shown below, this.getListData(); this.AddEventListeners(); in the render method will first call the function that will retrieve the list items and display within the div element declared in the render method. AddEventListeners will bind the button events to respective functions which gets called upon the button clicks.

```

1. import pnp from 'sp-pnp-js';
2.
3. import { Version } from '@microsoft/sp-core-library';
4. import {
5.     BaseClientSideWebPart,
6.     IPropertyPaneConfiguration,
7.     PropertyPaneTextField
8. } from '@microsoft/sp-webpart-base';
9. import { escape } from '@microsoft/sp-lodash-subset';
10.
11. import styles from './PnPspCrud.module.scss';
12. import * as strings from 'pnPspCrudStrings';
13. import { IPnPspCrudWebPartProps } from './IPnPspCrudWebPartProps';
14.

```

```

15.         export interface ISPList {
16.             ID: string;
17.             EmployeeName: string;
18.             Experience: string;
19.             Location: string;
20.         }
21.
22.     export default class PnPspCrudWebPart extends
23.         BaseClientSideWebPart<IPnPspCrudWebPartProps> {
24.
25.         private AddEventListeners(): void{
26.             document.getElementById('AddItem').addEventListener('click',()=>this.AddItem());
27.             document.getElementById('UpdateItem').addEventListener('click',()=>this.UpdateItem());
28.             document.getElementById('DeleteItem').addEventListener('click',()=>this.DeleteItem());
29.         }
30.
31.         private _getListData(): Promise<ISPList[]> {
32.             return pnp.sp.web.lists.getByTitle("EmployeeList").items.get().then((response) => {
33.
34.                 return response;
35.             });
36.
37.         }
38.
39.         private getListData(): void {
40.
41.             this._getListData()
42.                 .then((response) => {
43.                     this._renderList(response);
44.                 });
45.         }
46.
47.         private _renderList(items: ISPList[]): void {
48.             let html: string = '<table class="TFtable" border=1 width=100% style="border-
49.             collapse: collapse;">';
50.             items.forEach((item: ISPList) => {
51.                 html += `
52.                     <tr>
53.                         <td>${item.ID}</td>
54.                         <td>${item.EmployeeName}</td>
55.                         <td>${item.Experience}</td>
56.                         <td>${item.Location}</td>
57.                     </tr>
58.                 `;
59.             });
60.             html += `</table>`;
61.             const listContainer: Element = this.domElement.querySelector('#spGetListItems');
62.             listContainer.innerHTML = html;
63.         }
64.
65.
66.
67.         public render(): void {
68.             this.domElement.innerHTML =
69.
70.                 <div class="parentContainer" style="background-color: lightgrey">
71.                     <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
72.                         <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">

```

```

73. <span class="ms-font-xl ms-fontColor-white" style="font-size:28px">Welcome to
    SharePoint Framework Development using PnP JS Library</span>
74. <p class="ms-font-l ms-fontColor-white" style="text-align: left">Demo :
    SharePoint List CRUD using PnP JS and SPFx</p>
75.   </div>
76. </div>
77. <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
78.   <div style="background-color:Black;color:white;text-align: center;font-weight:
        bold;font-size:18px;">Employee Details</div>
79.
80. </div>
81. <div style="background-color: lightgrey" >
82.   <form >
83.     <br>
84.     <div data-role="header">
85.       <h3>Add SharePoint List Items</h3>
86.     </div>
87.     <div data-role="main" class="ui-content">
88.       <div >
89.         <input id="EmployeeName" placeholder="EmployeeName"/>
90.         <input id="Experience" placeholder="Experience" />
91.         <input id="Location" placeholder="Location"/>
92.       </div>
93.       <div></br></div>
94.       <div >
95.         <button id="AddItem" type="submit" >Add</button>
96.       </div>
97.     </div>
98.     <div data-role="header">
99.       <h3>Update/Delete SharePoint List Items</h3>
100.      </div>
101.      <div data-role="main" class="ui-content">
102.        <div >
103.          <input id="EmployeeId" placeholder="EmployeeId" />
104.        </div>
105.        <div></br></div>
106.        <div >
107.          <button id="UpdateItem" type="submit" >Update</button>
108.          <button id="DeleteItem" type="submit" >Delete</button>
109.        </div>
110.      </div>
111.    </form>
112.  </div>
113.  <br>
114.  <div style="background-color: lightgrey" id="spGetListItems" />
115. </div>
116.
117.  `;
118.  this.getListData();
119.  this.AddEventListeners();
120.  }
121.
122.  AddItem()
123.  {
124.
125.    pnp.sp.web.lists.getByTitle('EmployeeList').items.add({
126.      EmployeeName : document.getElementById('EmployeeName')["value"],
127.      Experience : document.getElementById('Experience')["value"],
128.      Location:document.getElementById('Location')["value"]
129.    });

```

```

130.         alert("Record with Employee Name : "+  

131.             document.getElementById('EmployeeName')["value"] + " Added !");  

132.     }  

133.  

134.     UpdateItem()  

135.     {  

136.  

137.         var id = document.getElementById('EmployeeId')["value"];  

138.         pnp.sp.web.lists.getByTitle("EmployeeList").items.getById(id).update({  

139.             EmployeeName : document.getElementById('EmployeeName')["value"],  

140.             Experience : document.getElementById('Experience')["value"],  

141.             Location:document.getElementById('Location')["value"]  

142.         });  

143.         alert("Record with Employee Name : "+  

144.             document.getElementById('EmployeeName')["value"] + " Updated !");  

145.  

146.     DeleteItem()  

147.     {  

148.  

149.         pnp.sp.web.lists.getByTitle("EmployeeList").items.getById(document.getElementById('Empl  

oyeeId')["value"]).delete();  

150.         alert("Record with Employee ID : "+  

151.             document.getElementById('EmployeeId')["value"] + " Deleted !");  

152.     protected getDataVersion(): Version {  

153.         return Version.parse('1.0');  

154.     }  

155.  

156.     protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {  

157.         return {  

158.             pages: [  

159.                 {  

160.                     header: {  

161.                         description: strings.PropertyPaneDescription  

162.                     },  

163.                     groups: [  

164.                         {  

165.                             groupName: strings.BasicGroupName,  

166.                             groupFields: [  

167.                                 PropertyPaneTextField('description', {  

168.                                     label: strings.DescriptionFieldLabel  

169.                                 })  

170.                             ]  

171.                         }  

172.                     ]  

173.                 }  

174.             ]  

175.         };  

176.     }  

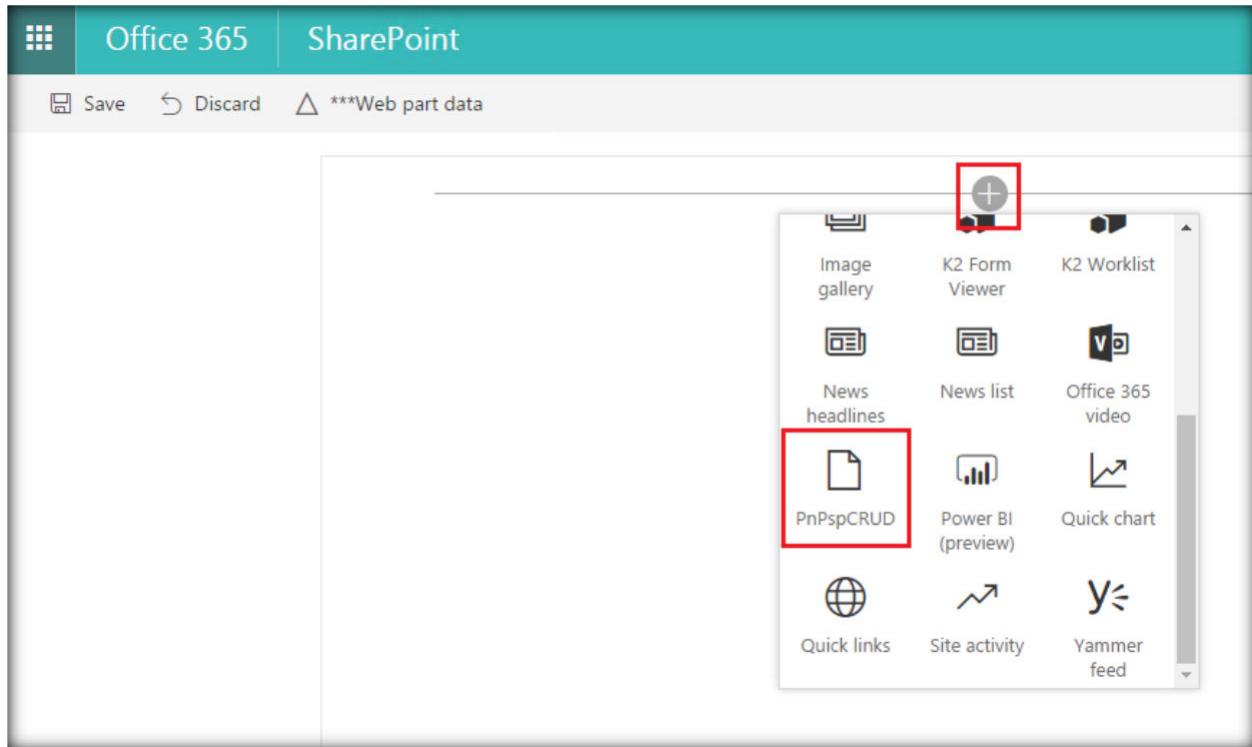
177. }

```

Test the Web part in SharePoint Online

Now, let's test the Web part in SharePoint Workbench available in SharePoint Online. Run *Gulp Serve* in the Node Command line and head over to SharePoint Online Site. Once we login to SharePoint

Online, we can invoke the workbench by appending the text '_layouts/15/workbench.aspx' to SharePoint Online URL. Add the webpart to the page by selecting 'PnPSPCRUD' icon.



The UI will look like below:

A screenshot of the SharePoint Framework development interface. The title bar says 'SharePoint'. Below it, there's a message 'Welcome to SharePoint Framework Development using PnP JS Library' and a subtitle 'Demo : SharePoint List CRUD using PnP JS and SPFx'. Underneath, there's a section titled 'Employee Details' with a sub-section 'Add SharePoint List Items' containing three input fields for 'EmployeeName', 'Experience', and 'Location', and an 'Add' button. Below that is another sub-section 'Update/Delete SharePoint List Items' with an input field for 'EmployeeId' and buttons for 'Update' and 'Delete'. At the bottom, there's a table with columns 'EmployeeId', 'EmployeeName', 'Experience', and 'Location', containing six rows of data. The table looks like this:

EmployeeId	EmployeeName	Experience	Location
2	Nimmy	Java	Qatar
3	Jinesh	Mobility	Sweden
4	Rajesh	.NET	Bahrain
5	John	SharePoint	UK
6	Vijai	SharePoint	USA

Add Item

We can input the details and save it to the list by clicking on Add which will create a new list item using the PnP Add method

Welcome to SharePoint Framework
Development using PnP JS Library

Demo : SharePoint List CRUD using PnP JS and SPFx

Employee Details

Add SharePoint List Items

Priyaranjan KS	SharePoint	India
----------------	------------	-------

Add

Update/Delete SharePoint List Items

EmployeeId	EmployeeName	Experience	Location
2	Nimmy	Java	Qatar
3	Jinesh	Mobility	Sweden

EmployeeId EmployeeName Experience Location

2 Nimmy Java Qatar

3 Jinesh Mobility Sweden

On Clicking on Add, the new item has been added to the SharePoint List.

SharePoint

ctsplayground.sharepoint.com says:
Record with Employee Name : Priyaranjan KS Added !

OK

Welcome to SharePoint Framework
Development using PnP JS Library

Demo : SharePoint List CRUD using PnP JS and SPFx

Employee Details

Add SharePoint List Items

Priyaranjan KS	SharePoint	India
----------------	------------	-------

Add

Update/Delete SharePoint List Items

EmployeeId	EmployeeName	Experience	Location
2	Nimmy	Java	Qatar
3	Jinesh	Mobility	Sweden

EmployeeId EmployeeName Experience Location

2 Nimmy Java Qatar

3 Jinesh Mobility Sweden

The table has listed the new item at the bottom of the list.

Demo : SharePoint List CRUD using PnP JS and SPFx

Employee Details

Add SharePoint List Items

EmployeeName Experience Location

Add

Update/Delete SharePoint List Items

EmployeeId

Update Delete

EmployeeId	EmployeeName	Experience	Location
2	Nimmy	Java	Qatar
3	Jinesh	Mobility	Sweden
4	Rajesh	.NET	Bahrain
5	John	SharePoint	UK
6	Vijai	SharePoint	USA
10	Kurien	Php	UK
11	Mathew Bhaskar	.NET Core	Bahrain
15	Priyanjan KS	SharePoint	India

Update Item

Similarly, we can update the existing list item by adding the data and providing the list item id which will be used to pick the item from the list and update it.

Demo : SharePoint List CRUD using PnP JS and SPFx

Employee Details

Add SharePoint List Items

Priyanjan KS SharePoint Qatar

Add

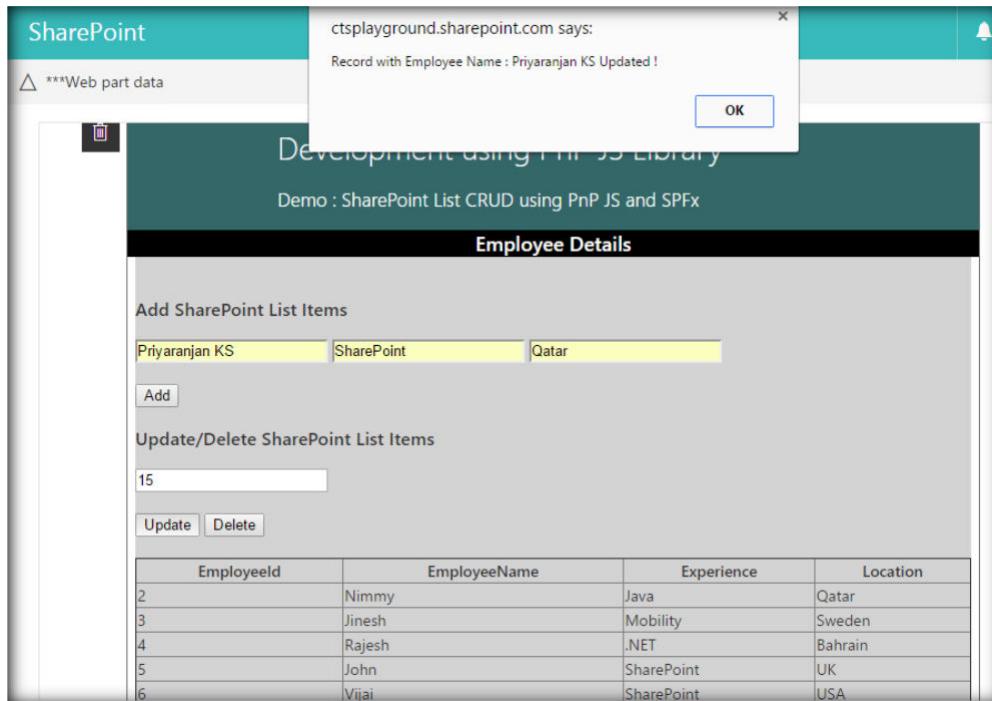
Update/Delete SharePoint List Items

15

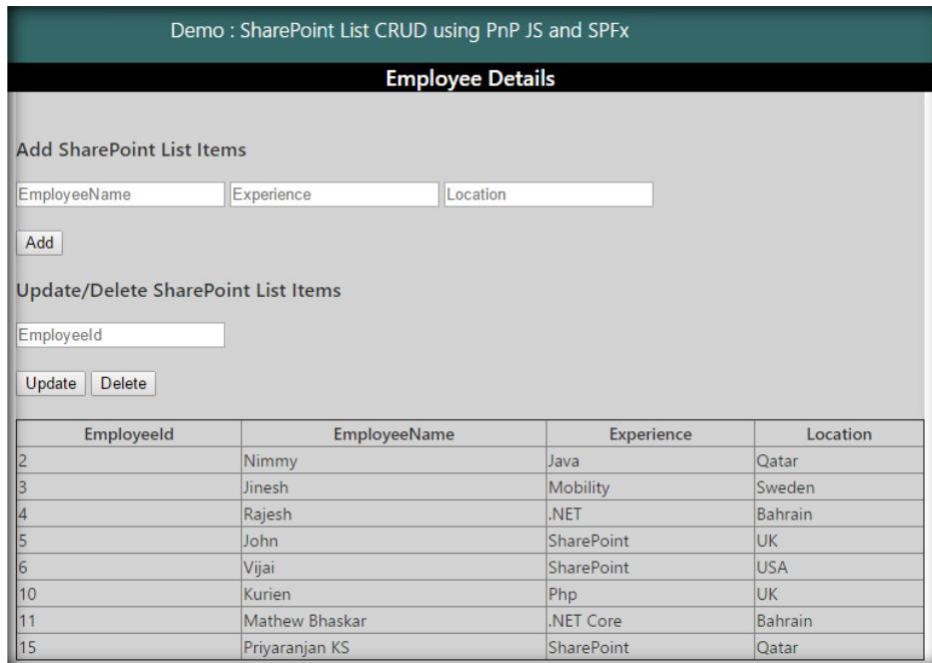
Update Delete

EmployeeId	EmployeeName	Experience	Location
2	Nimmy	Java	Qatar
3	Jinesh	Mobility	Sweden
4	Rajesh	.NET	Bahrain
5	John	SharePoint	UK
6	Vijai	SharePoint	USA
10	Kurien	Php	UK
11	Mathew Bhaskar	.NET Core	Bahrain
15	Priyanjan KS	SharePoint	India

On Clicking on Update, the list item has been updated.

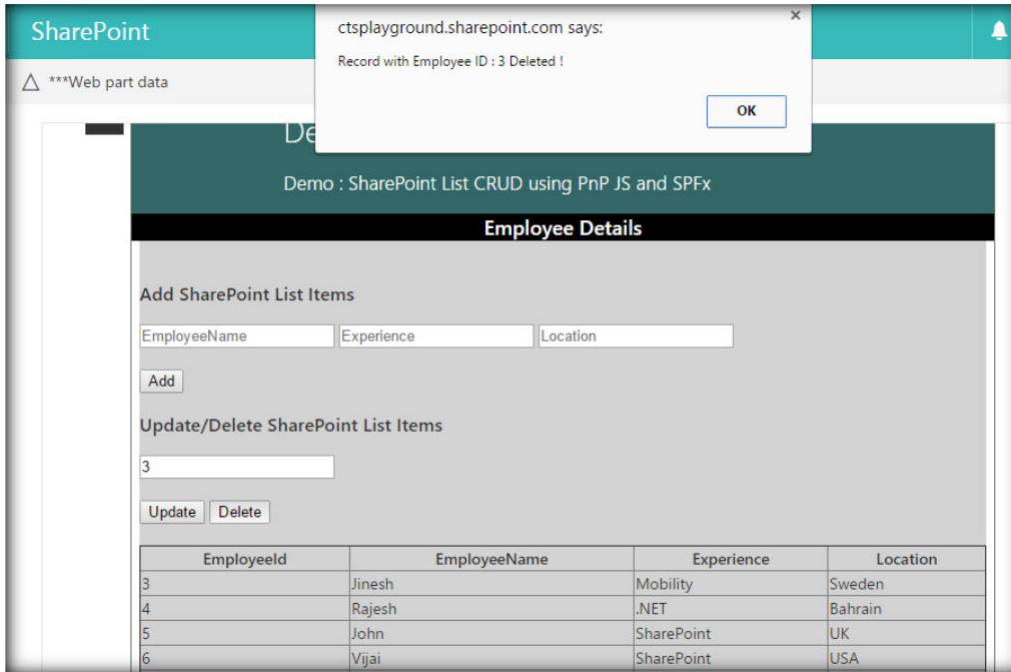


The location column value of the list item has been changed and reflected in the table as shown below.



Delete item

We can make use of the PnP delete method to delete the item from the list by providing the item id as shown below.



The major project files used in this solution has been zipped and uploaded at Microsoft [TechNet Gallery](#). Feel free to download it.

Getting Started with REACT JS in SharePoint

React makes it easy to create interactive UIs. It helps us to create simple views for each state in our application, efficiently update and render the right components when our data changes. In this section, we will make use of React and REST api to retrieve list items from SharePoint and display them using SPFx web part.

Retrieve SharePoint List data using REST API and display using Content Editor Webpart

Elements are the smallest building blocks of React apps. It describes what we want to see on the UI. Say :

```
const element = <h1>Display me at root node</h1>;
```

reactdom.render is the starting point of the React component. Let's say we have a <div> somewhere in our HTML file:

```
<div id="root"></div>
```

so as to render our React element into the above root DOM node, we will pass both to ReactDOM.render() as :

```
const element = <h1>Display me at root node</h1>;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

This will display the message at root div. In our case we will be displaying the data retrieved from 'ProductSales' list in the div named 'CarSalesData'.

```
ReactDOM.render(<ReactSP />, document.getElementById('CarSalesData'));
```

ReactSP represents the component that will be rendered at the CarSalesData div. ReactSP which is the component name is defined as plain javascript class which extends React.Component abstract class. We will also define at least one render method within it and will be defining the UI within this render method.

Within the class we will also have a constructor which is the right place to initialize state. We will update this state with SharePoint data at a later point in the react lifecycle. If we don't have to initialize state and we are not binding any methods, we don't need to implement a constructor for our React component.

Each component has several "lifecycle methods" that we can override to run code at a particular time of the process. Methods that are prefixed with 'will' are called just before some event happens, and methods prefixed with 'did' are called just after an event happens. We will be making use of the 'componentDidMount' method to fetch data from SharePoint List and we will update the state with this data. In the render method, we will then read the state data and display it in the UI.

Quarterly Car Sales Data				
Car Name	Quarter 1	Quarter 2	Quarter 3	Quarter 4
Maruti Baleno	6000	10500	7900	3600
Hyundai i20	13000	7000	8000	9900
Nissan Terrano	9000	11400	9100	8000
Ford Figo	13000	9100	9900	8000
Honda Accord	9100	7000	9100	9900
Tata Tiago	7000	9100	4500	7600
BMW X6	2500	1300	3400	3500
Nissan Terrano	5600	6500	7500	8000
Pajero	4500	2400	4600	6600
Land Cruiser	4600	5400	6500	5600

REACT and REST API script to display SharePoint data as Grid

The code can be saved as a text file and added to the Content Editor Webpart to display the grid webpart.

```
1. <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/react.js"></script>
2. <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/react-
   dom.min.js"></script>
3. <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
   standalone/6.24.0/babel.min.js"></script>
4. <div id="CarSalesData"></div>
5. <script type="text/babel">
6.   var tableStyle = {
7.     display: "table",
8.     marginLeft : "40px"
9.   }
10.  var panelStyle = {
11.    background: "#91A4A7"
12.  }
13.
14.  var divStyle = {
15.    background: "#eee",
16.    padding: "20px",
17.    margin: "20px"
18.  };
19.
20.  var headerCaptionStyle = {
21.    background: "#4B6978",
22.    display: "table-cell",
23.    border: "solid",
24.    textAlign : "center",
25.    width : "500px",
26.    height : "30px",
27.    paddingTop : "3px",
28.    color : "white",
29.    marginLeft : "80px",
30.    display : "block"
31.  };
32.
33.  var headerStyle = {
34.    background: "#4B6978",
35.    display: "table-cell",
36.    border: "solid",
37.    textAlign : "center",
38.    width : "100px",
39.    height : "30px",
40.    paddingTop : "10px",
41.    color : "white"
42.  };
43.
44.  var tableCaptionStyle = {
45.    background: "#c6e2ff",
46.    display: "block",
47.    fontSize : "20px",
48.    fontWeight: "bold",
49.    border: "solid",
50.    textAlign : "center",
```

```
51.          width : "650px",
52.          height : "30px",
53.          paddingTop : "3px",
54.          borderRadius: "25px",
55.          marginLeft : "30px",
56.          marginTop : "20px"
57.      }
58.
59.
60.      var rowCaptionStyle = {
61.          width : "600px",
62.          display : "table-caption",
63.          background: "#4B6978",
64.          textAlign : "center",
65.          padding: "20px",
66.          fontSize : "20px",
67.          fontWeight :"bold",
68.          color : "white"
69.      };
70.
71.      var rowStyle = {
72.          display : "table-row",
73.          background: "#eee",
74.          padding: "20px",
75.          margin: "20px",
76.          fontWeight :"bold"
77.      };
78.
79.      var CellStyle = {
80.          display: "table-cell",
81.          border: "solid",
82.          borderColor : "white",
83.          textAlign : "center",
84.          width : "100px",
85.          height : "30px",
86.          paddingTop : "10px"
87.
88.      }
89.
90.      class ReactSP extends React.Component{
91.          debugger;
92.          constructor(){
93.              super();
94.              this.state = {
95.                  items: [
96.                      {
97.                          "CarName": "",
98.                          "Quarter1": "",
99.                          "Quarter2":"",
100.                         "Quarter3": "",
101.                         "Quarter4":""
102.                     }
103.                 ]
104.             };
105.
106.         }
107.
108.         componentDidMount() {
109.             debugger;
110.             this.RetrieveSPData();
111.         }
112.
```

```

112.
113.     RetrieveSPData(){
114.         var reactHandler = this;
115.
116.         var spRequest = new XMLHttpRequest();
117.         spRequest.open('GET',
118.             "/sites/playground/_api/web/lists/getbytitle('ProductSales')/items",true);
119.         spRequest.setRequestHeader("Accept","application/json");
120.
121.         spRequest.onreadystatechange = function(){
122.
123.             if (spRequest.readyState === 4 && spRequest.status === 200){
124.                 var result = JSON.parse(spRequest.responseText);
125.
126.                 reactHandler.setState({
127.                     items: result.value
128.                 });
129.             } else if (spRequest.readyState === 4 && spRequest.status !==
200){
130.                 console.log('Error Occured !');
131.             }
132.         };
133.         spRequest.send();
134.     }
135.
136.     render(){
137.         debugger;
138.         return (
139.             <div style={panelStyle}>
140.                 <br></br>
141.
142.                 <br></br>
143.                 <div style={tableCaptionStyle} > Demo : Retrieve SharePoint List Items
using React JS </div>
144.                 <br></br>
145.
146.                 <div style={tableStyle} >
147.                     <div style={rowCaptionStyle} > Quarterly Car Sales Data </div>
148.                     <div style={rowStyle} >
149.                         <div style={headerStyle}>Car Name</div>
150.                         <div style={headerStyle}>Quarter 1 </div>
151.                         <div style={headerStyle}>Quarter 2</div>
152.                             <div style={headerStyle}>Quarter 3</div>
153.                             <div style={headerStyle}>Quarter 4</div>
154.                     </div>
155.
156.                     {this.state.items.map(function(item,key){
157.
158.                         return (<div style={rowStyle} key={key}>
159.                             <div style={CellStyle}>{item.CarName}</div>
160.                             <div style={CellStyle}>{item.Quarter1}</div>
161.                             <div style={CellStyle}>{item.Quarter2}</div>
162.                             <div style={CellStyle}>{item.Quarter3}</div>
163.                             <div style={CellStyle}>{item.Quarter4}</div>
164.                         </div>);
165.                     })}
166.
167.                 </div>
168.             </div>
169.

```

```

170.           );
171.       }
172.
173.       }
174.   }
175.
176.   ReactDOM.render(<ReactSP />, document.getElementById('CarSalesData'));
177. </script>

```

Create SPFx Webpart to retrieve SharePoint List Items using REACT & REST API

In this section, we will see how to create a client webpart using SharePoint Framework and React JS that displays SharePoint List data retrieved using REST API. The major project files used in this solution has been zipped and uploaded at Microsoft [TechNet Gallery](#). Feel free to download it.

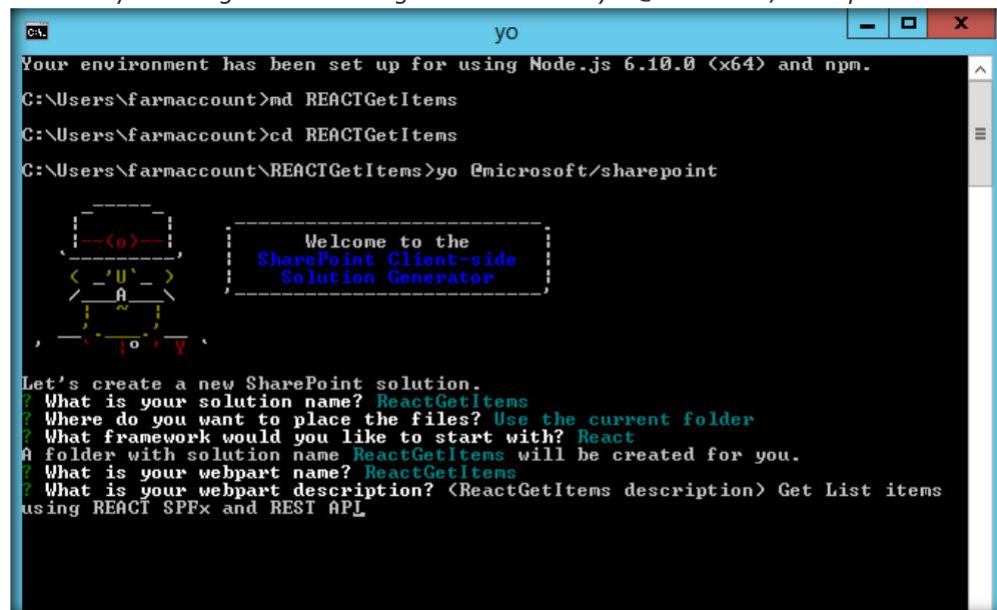
Let's get started with the creation of the project by creating a directory.

```

md REACTGetItems
cd REACTGetItems

```

Run the yeoman generator using the command 'yo @microsoft/sharepoint'



The screenshot shows a terminal window titled 'cmd' with the command 'yo' entered. The output shows the setup for Node.js 6.10.0 (x64) and npm, followed by the yeoman generator command: 'C:\Users\farmaccount>yo @microsoft/sharepoint'. A graphical interface for the 'SharePoint Client-side Solution Generator' appears, prompting for a new SharePoint solution. The terminal then lists several questions and their answers:

- What is your solution name? `ReactGetItems`
- Where do you want to place the files? `Use the current folder`
- What framework would you like to start with? `React`
- A folder with solution name `ReactGetItems` will be created for you.
- What is your webpart name? `ReactGetItems`
- What is your webpart description? `<ReactGetItems description> Get List items using REACT SPFx and REST API`

```
Node.js command prompt
+-- glob2base@0.0.12
|   '-- find-index@0.1.1
+-- minimatch@2.0.10
+-- ordered-read-streams@0.1.0
+-- through2@0.6.5
|   '-- readable-stream@1.0.34
+-- unique-stream@1.0.0
+-- glob-watcher@0.0.6
+-- gaze@0.5.2
|   '-- globule@0.1.0
|       '-- glob@3.1.21
|           '-- graceful-fs@1.2.3
|               '-- inherits@1.0.2
|       '-- lodash@1.0.2
|           '-- minimatch@0.2.14
+-- graceful-fs@3.0.11
|   '-- natives@1.1.0
+-- strip-bom@1.0.0
+-- first-chunk-stream@1.0.0
+-- is-utf8@0.2.1
+-- through2@0.6.5
|   '-- readable-stream@1.0.34
|       '-- isarray@0.0.1
+-- vinyl@0.4.6
+-- clone@0.2.0

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
>

=+#####
#####
##<##<(e)
## ####>\ <(e)      Congratulations!
## /##> <(e)      Solution react-get-items is created.
## ##> ##>      Run gulp serve to play with it!
## /##> <(e)
#####
**=+#####

C:\Users\farmaccount\REACTGetItems>
```

Edit the web part

Run Code . to create the scaffolding and open the project in Visual Studio Code. We will be required jQuery to make ajax rest api calls. So let's install jQuery using NPM as shown below.

```
npm install --save jquery
npm install --save @types/jquery
```

Later we will import them to the solution in the ReactGetItems.tsx file using:
import * as jquery from 'jquery';

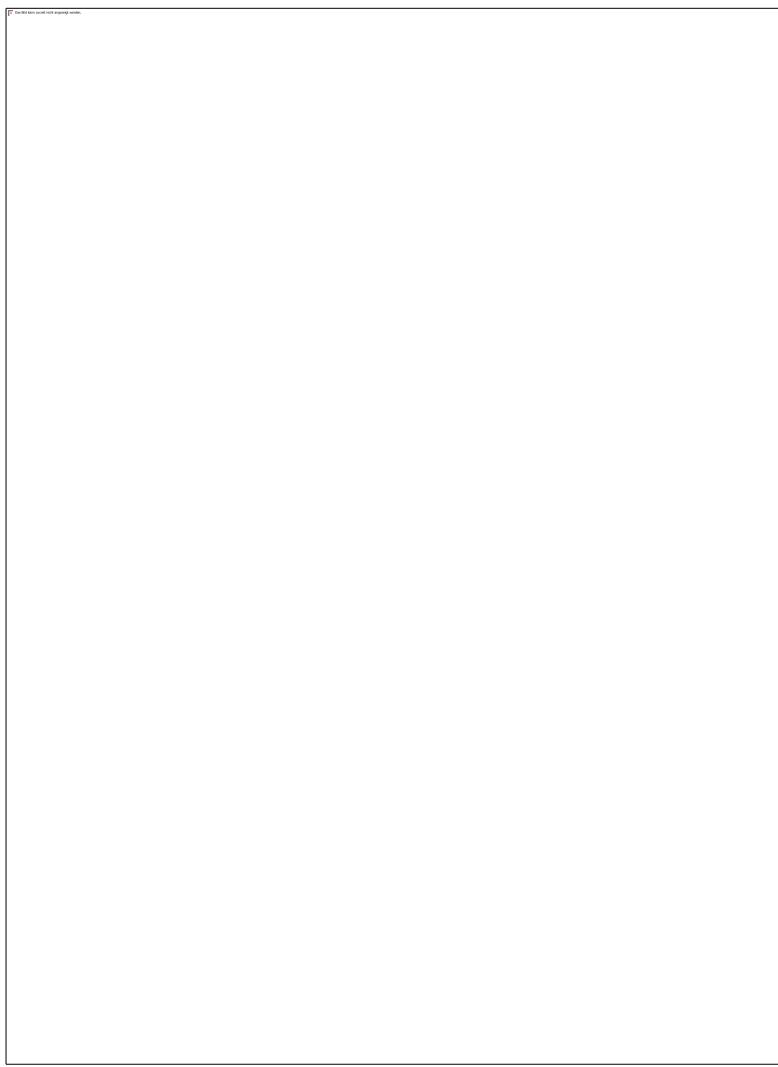
```
C:\Users\farmaccount\REACTGetItems>npm install --save jquery
react-get-items@0.0.1 C:\Users\farmaccount\REACTGetItems
`-- jquery@3.2.1

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
>

C:\Users\farmaccount\REACTGetItems>npm install --save @types/jquery
react-get-items@0.0.1 C:\Users\farmaccount\REACTGetItems
`-- @types/jquery@2.0.43

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
```

Exploring the File Structure



We will be making use of the above marked files to implement the solution using React.

IReactgetItemsProps.TS : This will hold the properties that will be accessed by other files in the solution. By default, there is a description property defined, we will add another property 'siteURL' as well using the interface.

ReactGetItems/modue.scss : This will contain the css styles that are used within the TSX file.

ReactGetItems.tsx : This file serves as the major location where the UI and Logics are defined.

ReactGetItemsWebPart.ts : This acts as the starting point of the control flow and data rendering phase is initiated from this file using the ReactDOM.render method.

ReactGetItemsWebPart.ts

The rendering of the web part is initiated from the ReactGetItemsWebPart TS file . Here ReactDOM.render method takes the first parameter as the element that should be rendered (after processing some logic) at the second parameter. The element is defined by the class 'ReactGetItems'. ReactGetItems extends React.Component in the ReactGetItems.tsx file that contains the major logic processing and UI.

```
1. export default class ReactGetItemsWebPart extends
   BaseClientSideWebPart<IReactGetItemsWebPartProps> {
2.
3.   public render(): void {
4.     const element: React.ReactElement<IReactGetItemsProps > = React.createElement(
5.       ReactGetItems,
6.       {
7.         description: this.properties.description,
8.         siteurl: this.context.pageContext.web.absoluteUrl
9.       }
10.    );
11.
12.    ReactDOM.render(element, this.domElement);
13.  }
```

IReactgetItemsProps.TS

This file contains the properties that will be accessed across the files and are declared using an Interface as shown below:

```
1. export interface IReactGetItemsProps {
2.   description: string;
3.   siteurl: string;
4. }
```

ReactGetItems/modue.scss

The css style used by the web part is defined within this file The CSS used for our web part is given below:

```
1. .tableStyle{
2.         display: table ;
3.         margin-left : 100px ;
4.     }
5. .panelStyle{
6.         background: lightblue ;
7.     }
8.
9. .divStyle{
10.    background: #eee ;
11.    padding: 20px ;
12.    margin: 20px ;
13. }
14.
15. .headerCaptionStyle{
16.         background: #4B6978 ;
17.         display: table-row ;
18.         border: solid ;
19.         text-align : center ;
20.         width : 420px ;
21.         height : 30px ;
22.         padding-top : 3px ;
23.         color : white ;
24.         margin-left : 100px ;
25.         display : block ;
26. }
27.
28. .headerStyle{
29.         background: #4B6978 ;
30.         display: table-row ;
31.         border: solid ;
32.         text-align : center ;
33.         width : 100px ;
34.         height : 30px ;
35.         padding-top : 10px ;
36.         color : white ;
37. }
38.
39. .tableCaptionStyle{
40.         background:#4B6978 ;
41.         display: block ;
42.         font-size : 20px ;
43.         font-weight: bold ;
44.         border: solid ;
45.         text-align : center ;
46.         width : 650px ;
47.         height : 30px ;
48.         padding-top : 3px ;
49.         border-radius: 25px ;
50.         margin-left : 30px ;
51.         margin-top : 20px ;
52.         color:white;
53. }
54.
55.
56. .rowCaptionStyle{
57.         width : 600px ;
58.         display : table-caption ;
```

```

59.     background: #4B6978 ;
60.     text-align : center ;
61.     padding: 20px ;
62.     font-size : 20px ;
63.     font-weight : bold ;
64.     color : white ;
65.   }
66.
67. .rowStyle{
68.   display : table-row ;
69.   background: #eee ;
70.   padding: 20px ;
71.   margin: 20px ;
72.   font-weight : bold ;
73. }
74.
75. .CellStyle{
76.   display: table-cell ;
77.   border: solid ;
78.   border-color : white ;
79.   text-align : center ;
80.   width : 100px ;
81.   height : 30px ;
82.   padding-top : 10px ;
83. }
```

ReactGetItems.tsx

This is the primary file where the logic and UI is written. ReactDOM.render method in the ReactGetItemsWebPart file passes the control over to this file. class ReactGetItems extends React.Component and implements a constructor where the state objects are initialized. The state object contains the list columns that will be populated using REST API calls.

```

1. public constructor(props: IReactGetItemsProps, state: IReactGetItemsState){
2.   super(props);
3.   this.state = {
4.     items: [
5.       {
6.         "EmployeeName": "",
7.         "EmployeeId": "",
8.         "Experience":"",
9.         "Location":""
10.      }
11.    ]
12.  };
13. }
```

The class also contains componentDidMount method implementation which will be called after mounting of the component. We can also make use of componentWillMount which is synchronous in nature. We will make the REST API call within this method to retrieve the list items and add it to the state object.

```

1. public componentDidMount(){
2.     var reactHandler = this;
3.     jquery.ajax({
4.         url: `${this.props.siteurl}/_api/web/lists/getbytitle('EmployeeList')/items`,
5.         type: "GET",
6.         headers:{'Accept': 'application/json; odata=verbose;'},
7.         success: function(resultData) {
8.             reactHandler.setState({
9.                 items: resultData.d.results
10.            });
11.        },
12.        error : function(jqXHR, textStatus, errorThrown) {
13.        }
14.    });
15. }

```

Finally, the render method will read the state object and build the UI

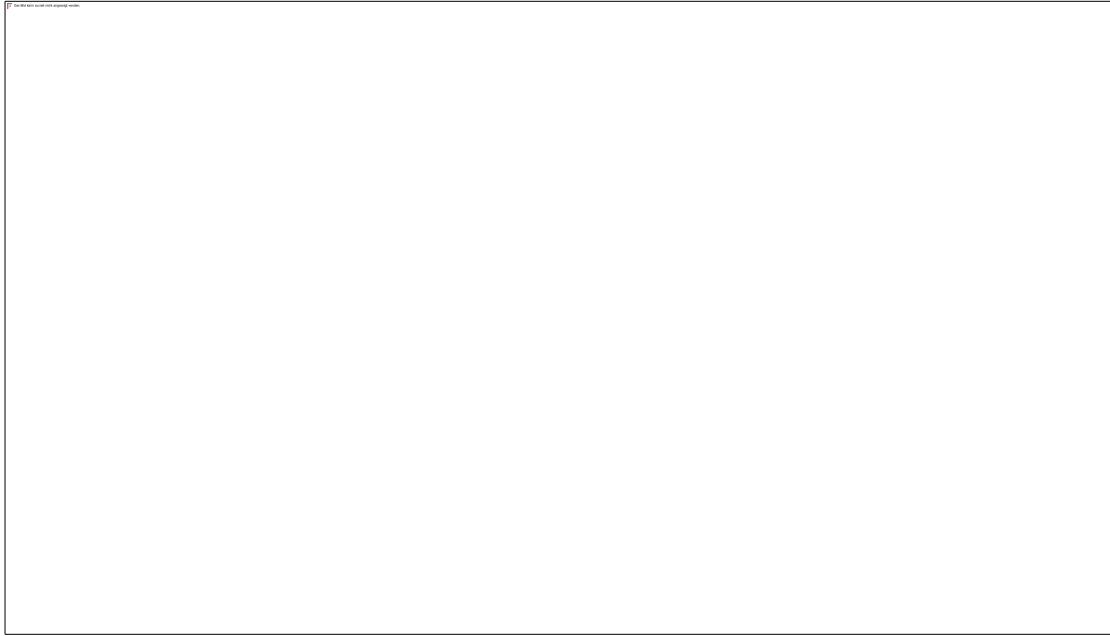
```

1. public render(): React.ReactElement<IReactGetItemsProps> {
2.     return (
3.
4.         <div className={styles.panelStyle} >
5.             <br></br>
6.
7.             <br></br>
8.             <div className={styles.tableCaptionStyle} > Demo : Retrieve SharePoint List
Items using SPFx , REST API & React JS </div>
9.             <br></br>
10.            <div className={styles.headerCaptionStyle} > Employee Details</div>
11.            <div className={styles.tableStyle} >
12.
13.                <div className={styles.headerStyle} >
14.                    <div className={styles.CellStyle}>Employee Name</div>
15.                    <div className={styles.CellStyle}>Employee Id </div>
16.                    <div className={styles.CellStyle}>Experience</div>
17.                    <div className={styles.CellStyle}>Location</div>
18.                </div>
19.
20.                {this.state.items.map(function(item,key){
21.
22.                    return (<div className={styles.rowStyle} key={key}>
23.                        <div className={styles.CellStyle}>{item.EmployeeName}</div>
24.                        <div className={styles.CellStyle}>{item.EmployeeId}</div>
25.                        <div className={styles.CellStyle}>{item.Experience}</div>
26.                        <div className={styles.CellStyle}>{item.Location}</div>
27.
28.                    </div>);
29.                })}
30.            </div>
31.        </div>
32.    );
33. }

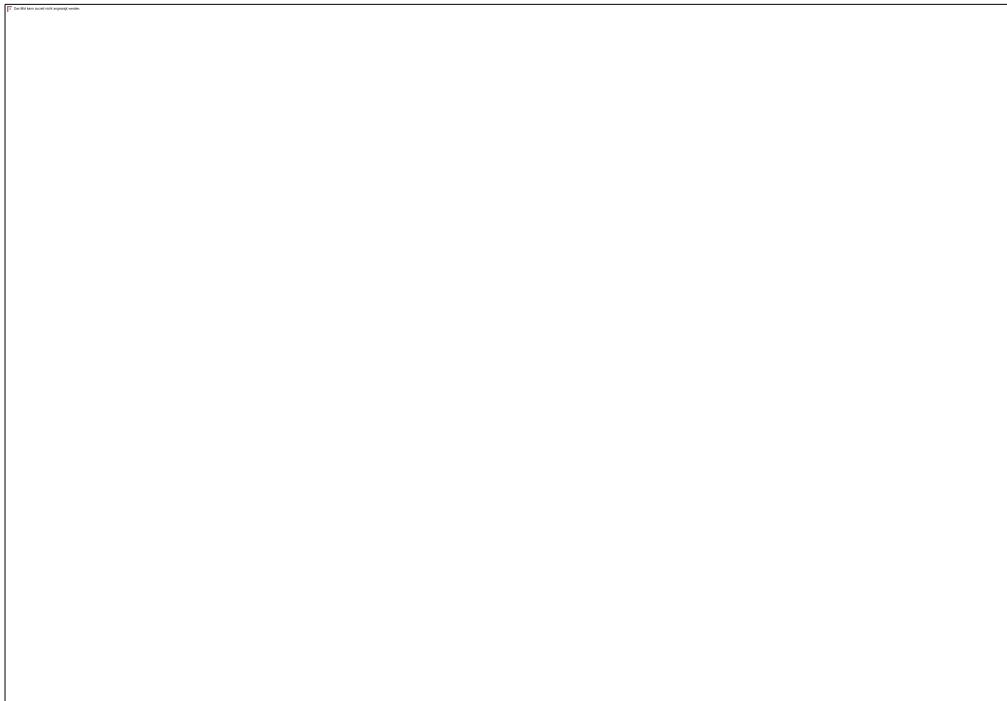
```

Test the Web part in SharePoint Online

Now let's test the solution in SharePoint Online Workbench. Run *Gulp Serve* in the node command and head over to the SharePoint Online Workbench URL by appending '_layouts/15/workbench.aspx' to the URL.



The retrieved data has been displayed as Grid as shown below:



TSX File contents for retrieving list items using REST API and REACT

The tsx file contents used to retrieve the list items via rest api is given below :

```
1. import * as React from 'react';
2. import styles from './ReactGetItems.module.scss';
3. import { IReactGetItemsProps } from './IReactGetItemsProps';
4. import { escape } from '@microsoft/sp-lodash-subset';
5. import * as jquery from 'jquery';
6.
7. export interface IReactGetItemsState{
8.     items:[
9.         {
10.             "EmployeeName": "",
11.             "EmployeeId": "",
12.             "Experience":"",
13.             "Location":""
14.         }]
15. }
16.
17. export default class ReactGetItems extends
    React.Component<IReactGetItemsProps, IReactGetItemsState> {
18.
19.     public constructor(props: IReactGetItemsProps, state: IReactGetItemsState){
20.         super(props);
21.         this.state = {
22.             items: [
23.                 {
24.                     "EmployeeName": "",
25.                     "EmployeeId": "",
26.                     "Experience":"",
27.                     "Location":""
28.                 }
29.             ]
30.         };
31.     }
32.
33.     public componentDidMount(){
34.         var reactHandler = this;
35.         jquery.ajax({
36.             url: `${this.props.siteurl}/_api/web/lists/getbytitle('EmployeeList')/items`,
37.             type: "GET",
38.             headers:{'Accept': 'application/json; odata=verbose;'},
39.             success: function(resultData) {
40.                 reactHandler.setState({
41.                     items: resultData.d.results
42.                 });
43.             },
44.             error : function(jqXHR, textStatus, errorThrown) {
45.             }
46.         });
47.     }
48.
49.
50.     public render(): React.ReactElement<IReactGetItemsProps> {
51.         return (
52.
53.             <div className={styles.panelStyle} >
```

```

54.          <br></br>
55.
56.          <br></br>
57.          <div className={styles.tableCaptionStyle} > Demo : Retrieve SharePoint List
58.            Items using SPFx , REST API & React JS </div>
59.          <br></br>
60.          <div className={styles.headerCaptionStyle} > Employee Details</div>
61.          <div className={styles.tableStyle} >
62.            <div className={styles.headerStyle} >
63.              <div className={styles.CellStyle}>Employee Name</div>
64.              <div className={styles.CellStyle}>Employee Id </div>
65.              <div className={styles.CellStyle}>Experience</div>
66.              <div className={styles.CellStyle}>Location</div>
67.
68.            </div>
69.
70.            {this.state.items.map(function(item,key){
71.
72.              return (<div className={styles.rowStyle} key={key}>
73.                  <div className={styles.CellStyle}>{item.EmployeeName}</div>
74.                  <div className={styles.CellStyle}>{item.EmployeeId}</div>
75.                  <div className={styles.CellStyle}>{item.Experience}</div>
76.                  <div className={styles.CellStyle}>{item.Location}</div>
77.
78.                </div>);
79.            })
80.
81.          </div>
82.        </div>
83.      );
84.    }
85.  }

```

The major project files used in this solution has been zipped and uploaded at Microsoft [TechNet Gallery](#). Feel free to download it.

Summary

This book serves as a script cookbook to give you a head start with SharePoint Framework development using TypeScript,PnP JS and React JS. More examples and scenario based solutions will be added to the upcoming versions as SharePoint Framework evolves with respect to functionality. All the major solution files used in this book has been uploaded to Microsoft TechNet gallery.