SPFx

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

Contents

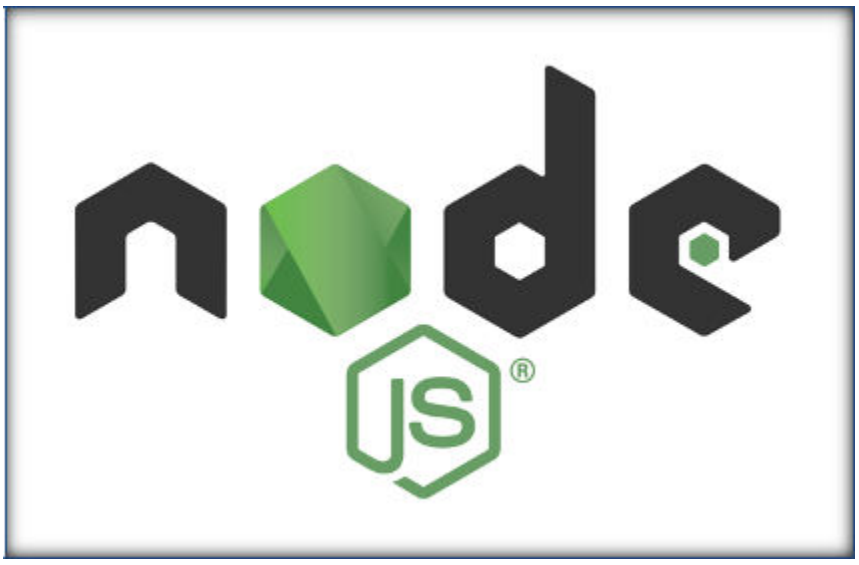
SharePoint Framework is the new development model in which lots of work had been going on in the past year. It went to General Availability on Feb 23rd 2017. It is a page and web part model that provides full support for client-side SharePoint development, easy integration with SharePoint data and support 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.

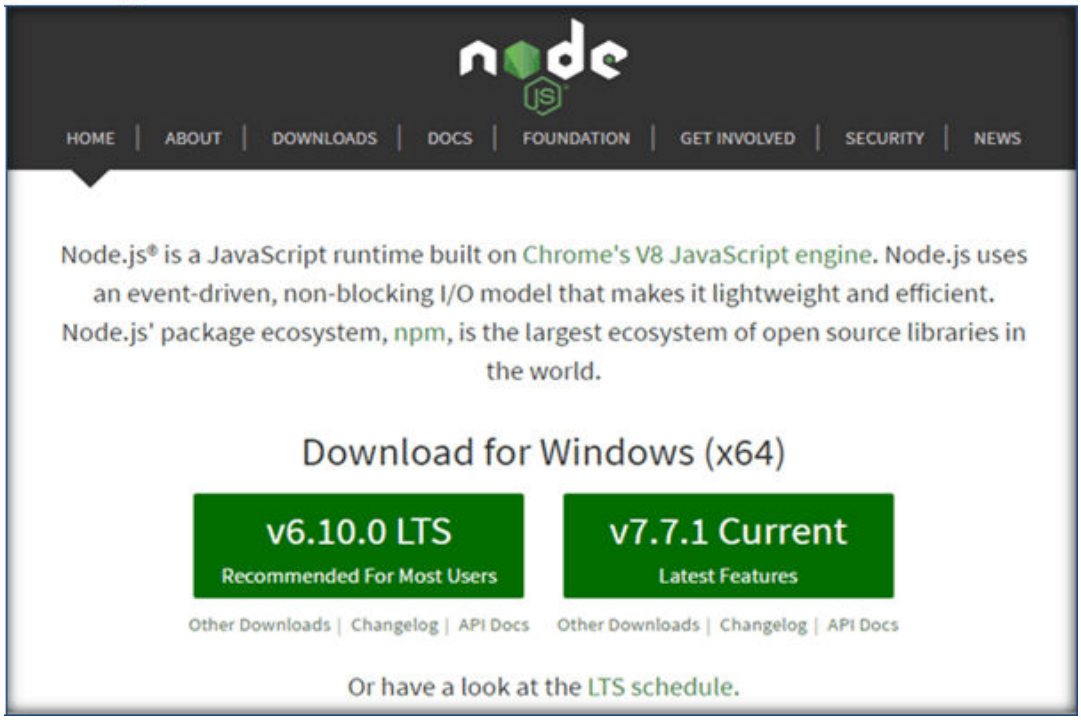
- Node JS
- Yeoman and Gulp
- Yeoman SharePoint Generator
- Code Editor(Visual Studio Code/Webstorm)
- Postman and Fiddler(optional)

Install Node JS

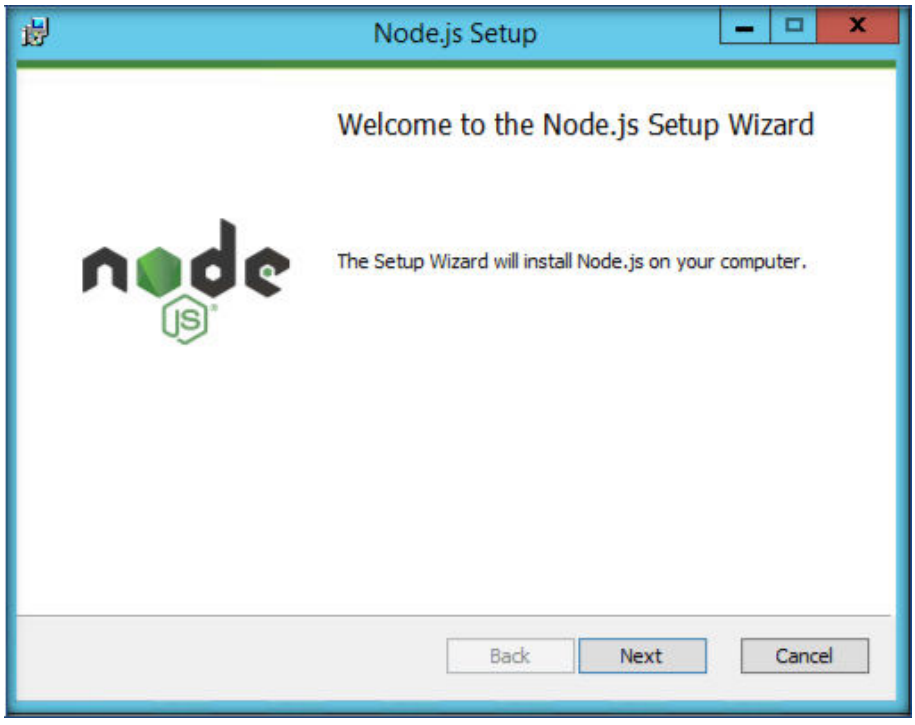


Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the

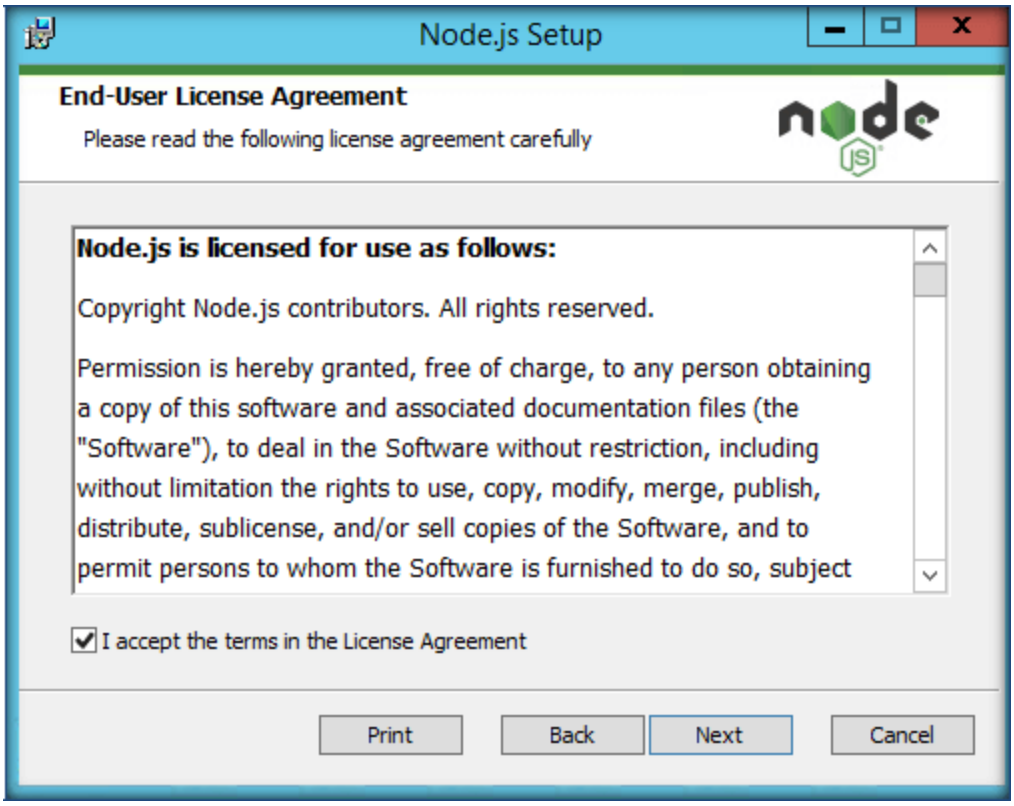
largest ecosystem of open source libraries in the world. We will be making use of npm along with Yeoman and Gulp to package and deploy modules.
As the first step we will install NodeJS Long Term Support Version (LTS). We can install Node JS from this link .



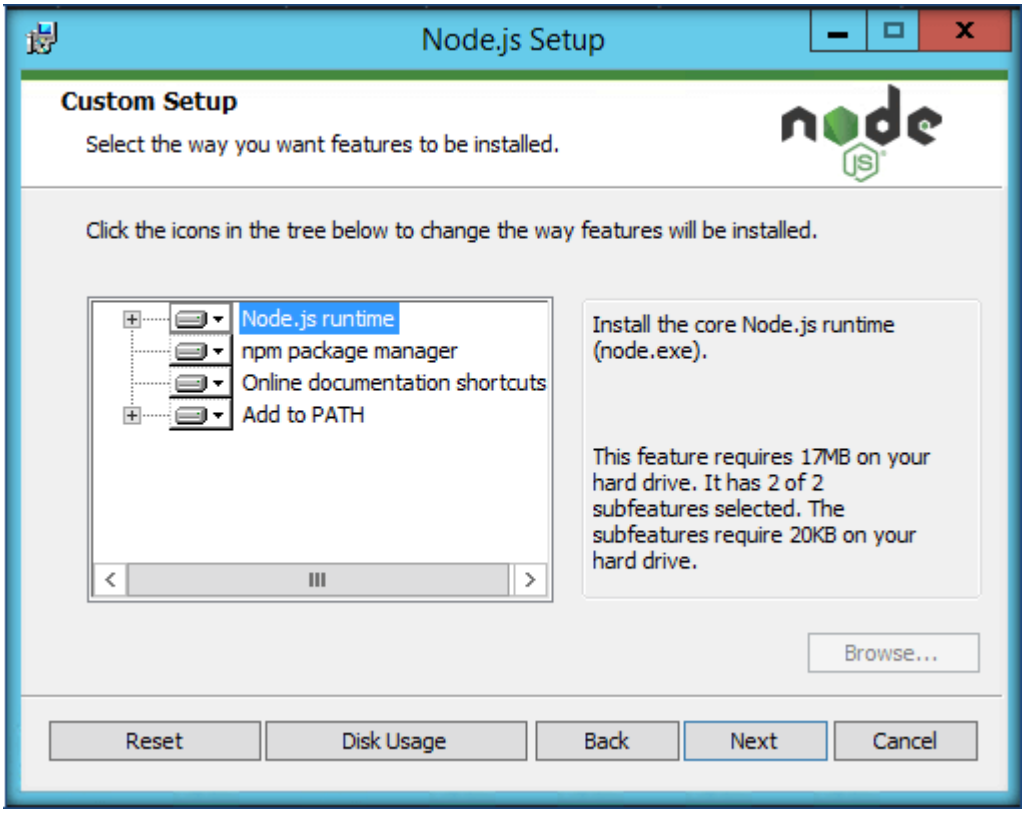
Once we have downloaded the LTS version, run the executable file and proceed.



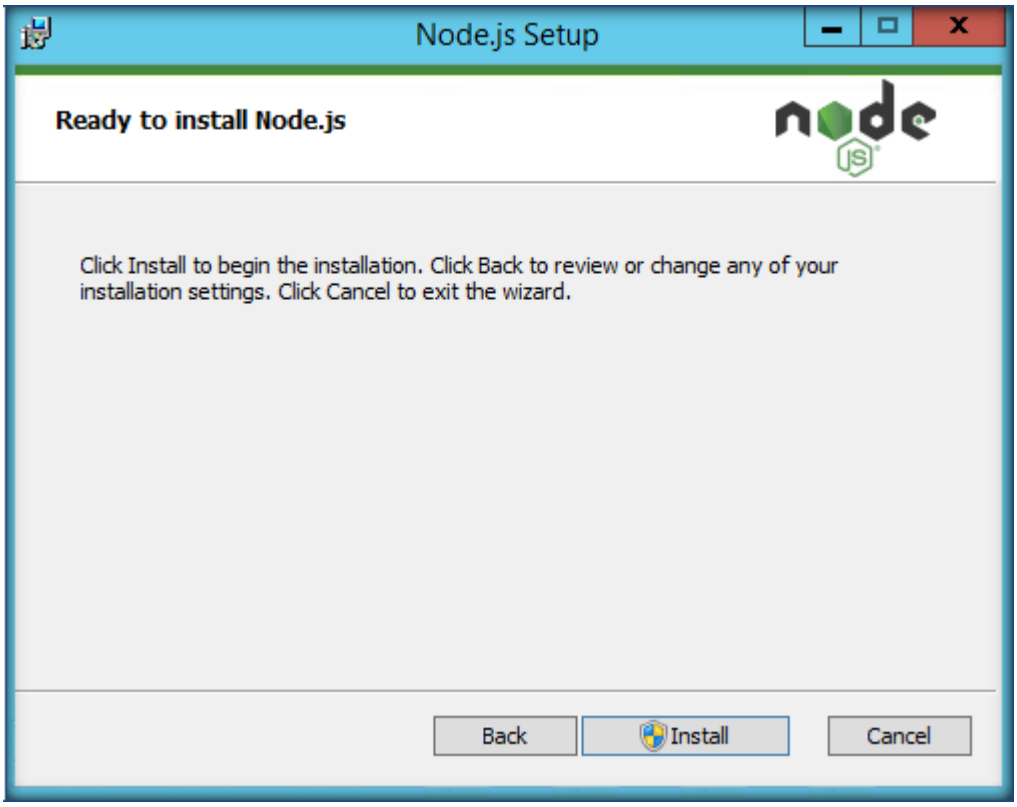
Accept the license agreement and click on Next.



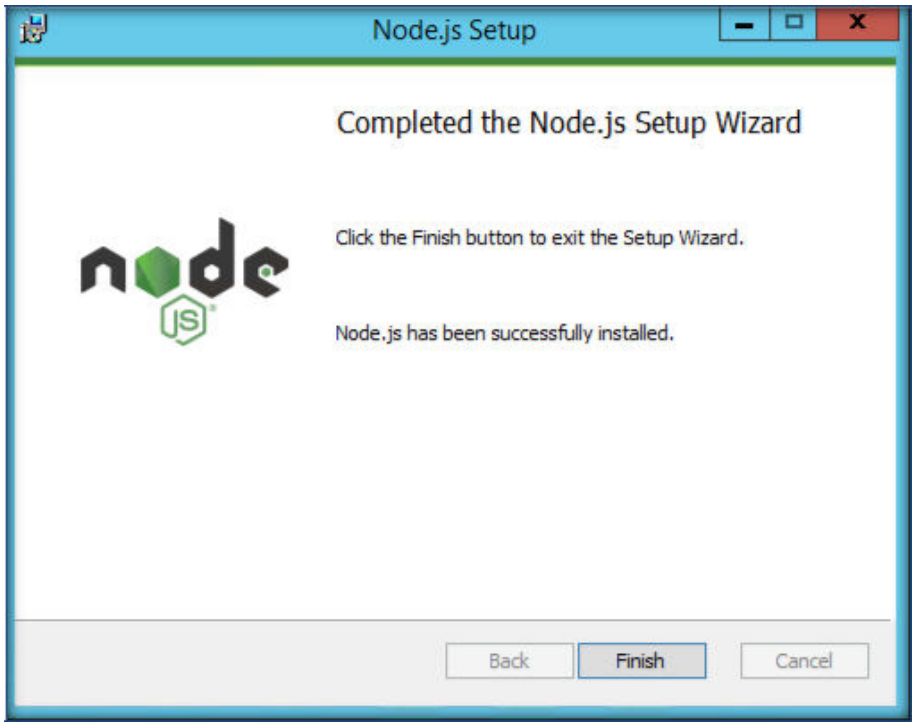
We will select Node.js run time installation.



Click on Install to start the installation procedure.

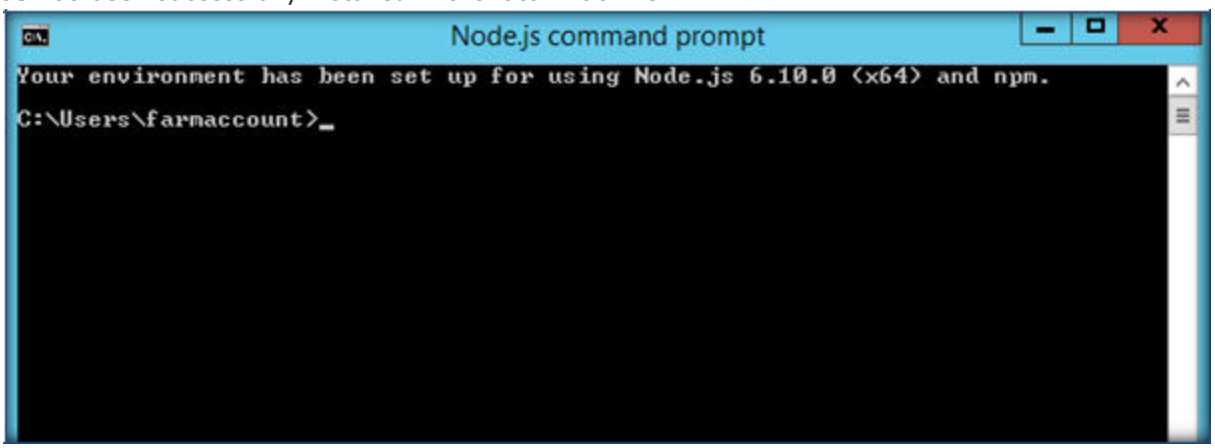


Finally, we are done installing NodeJS.

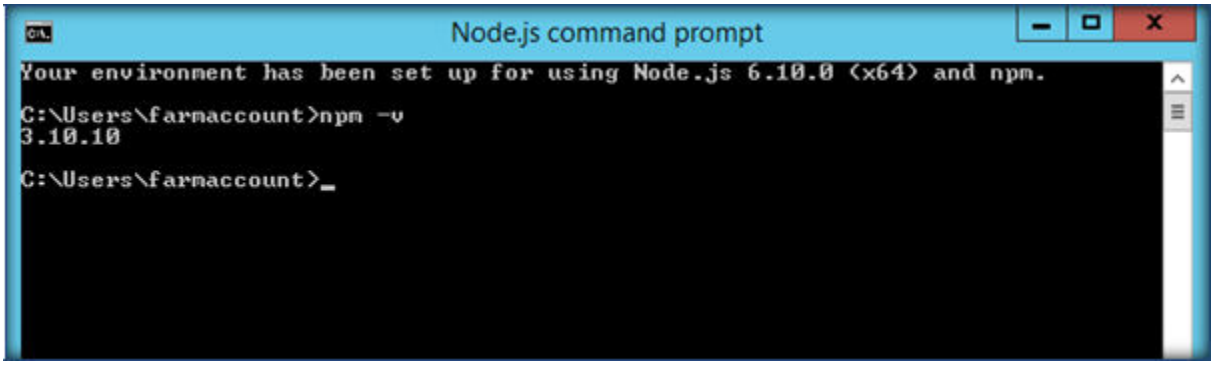


Click on Finish button and restart your computer. You won't be able to run Node.js until you restart your computer.

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.



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

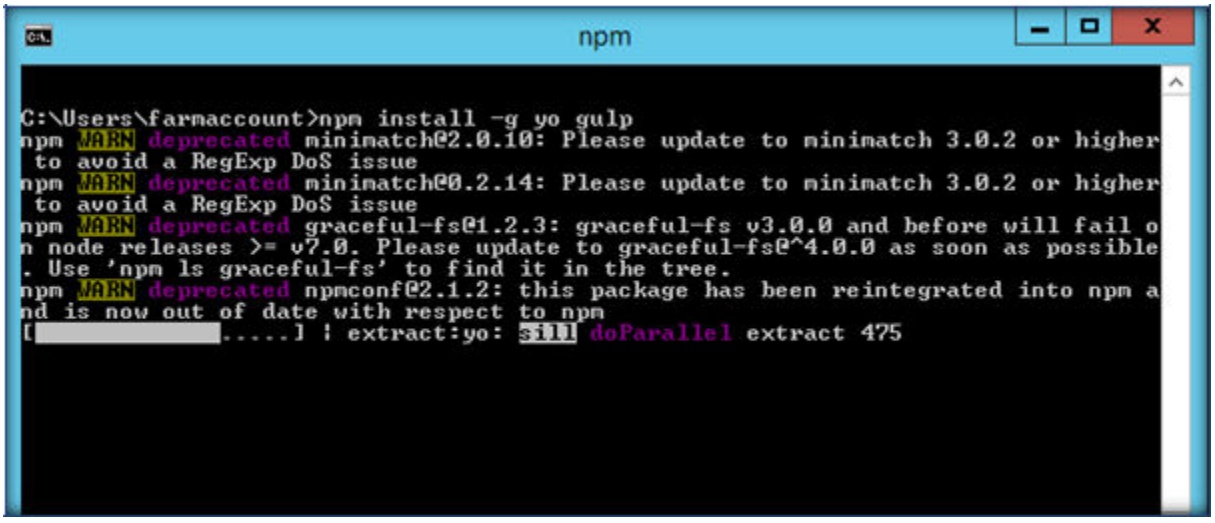


Install Yeoman and Gulp

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



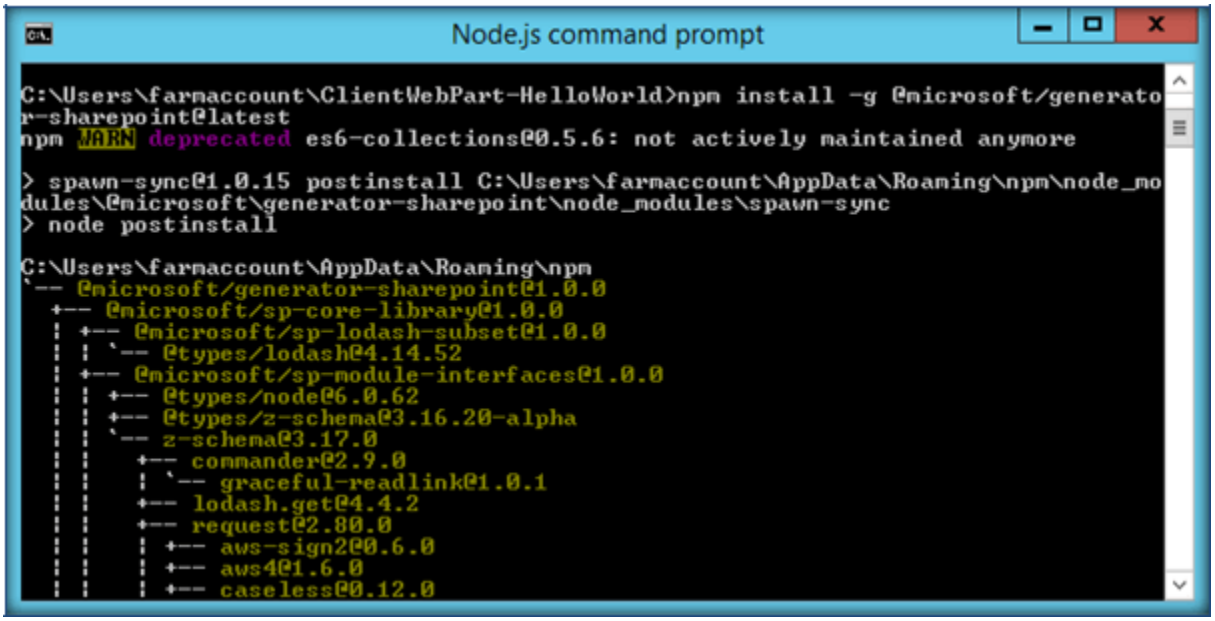
We can get the version of Yeoman using the command:
yo --version

We can get the Gulp Version using the command:
gulp –v

Install Yeoman SharePoint Generator

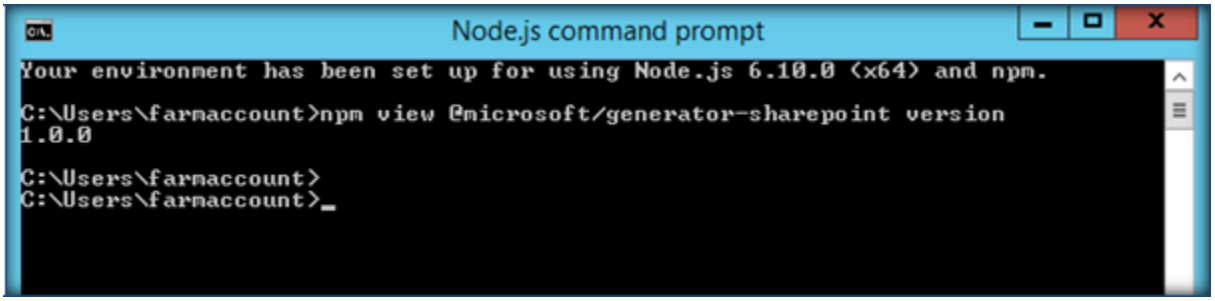
The Yeoman SharePoint web part generator helps you to quickly create a SharePoint client-side solution project with the right tool chain and project structure.
Yeoman SharePoint Generator can be installed using the below command:

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



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

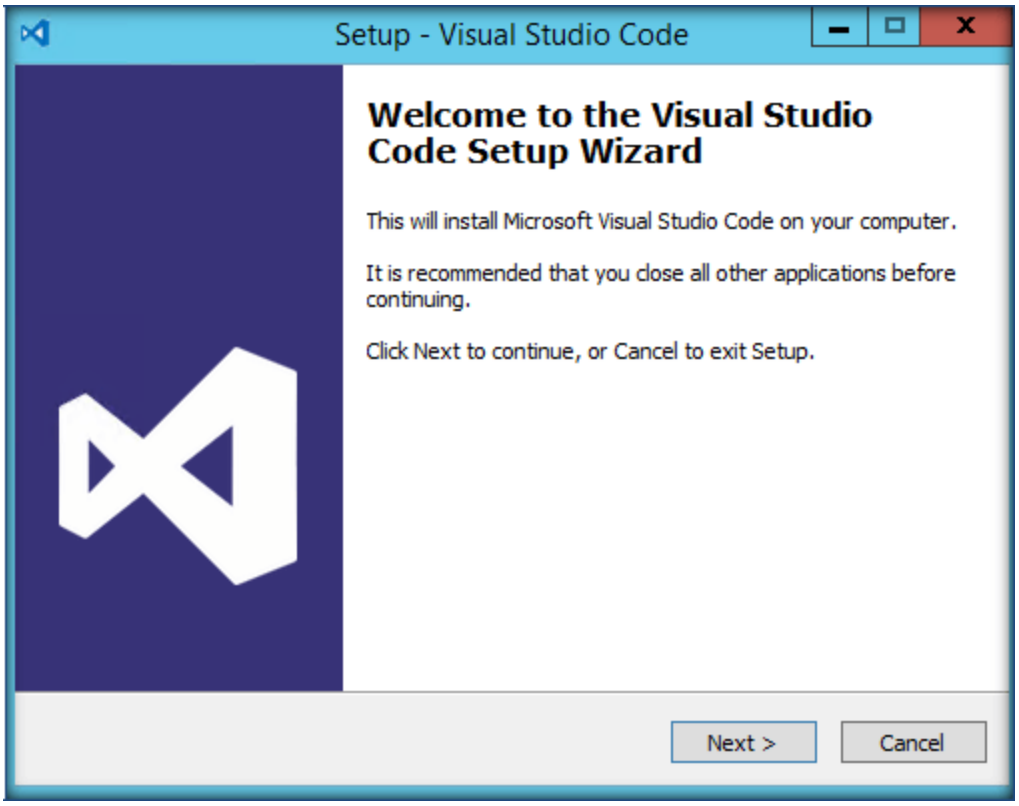


Code Editor

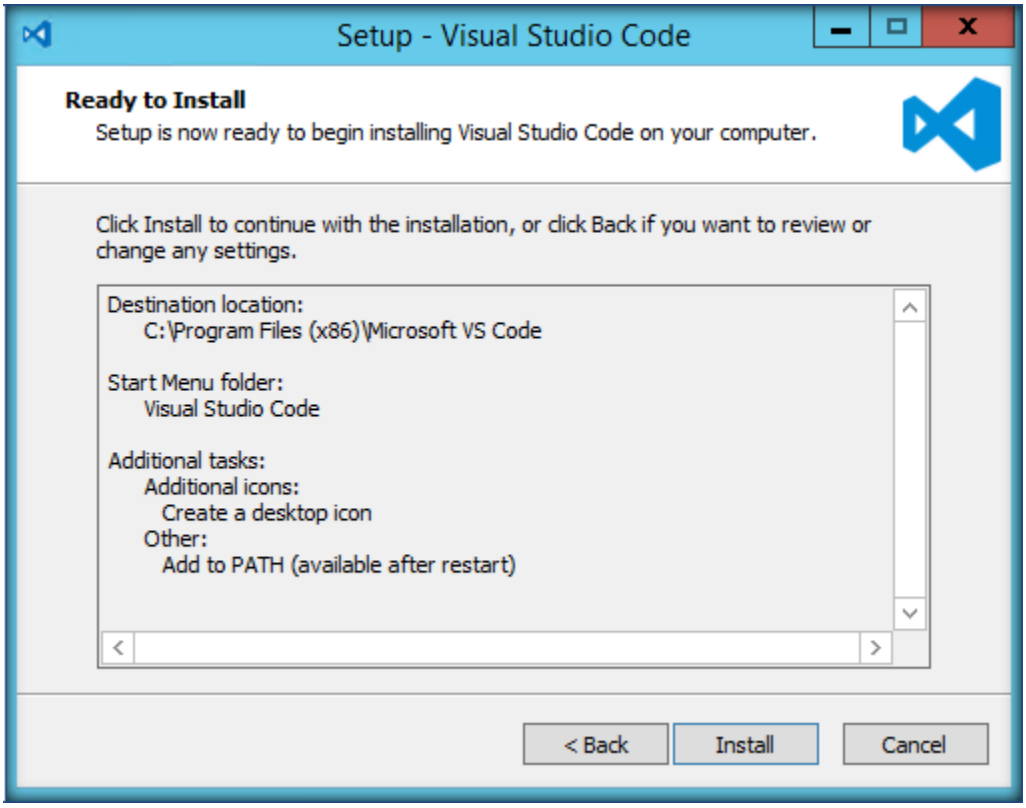
Next, we need a code editor that will help us with code editing. We can use any code editor or IDE that supports client-side development to build our web part, such as:

- Visual Studio Code
- Atom
- Webstorm

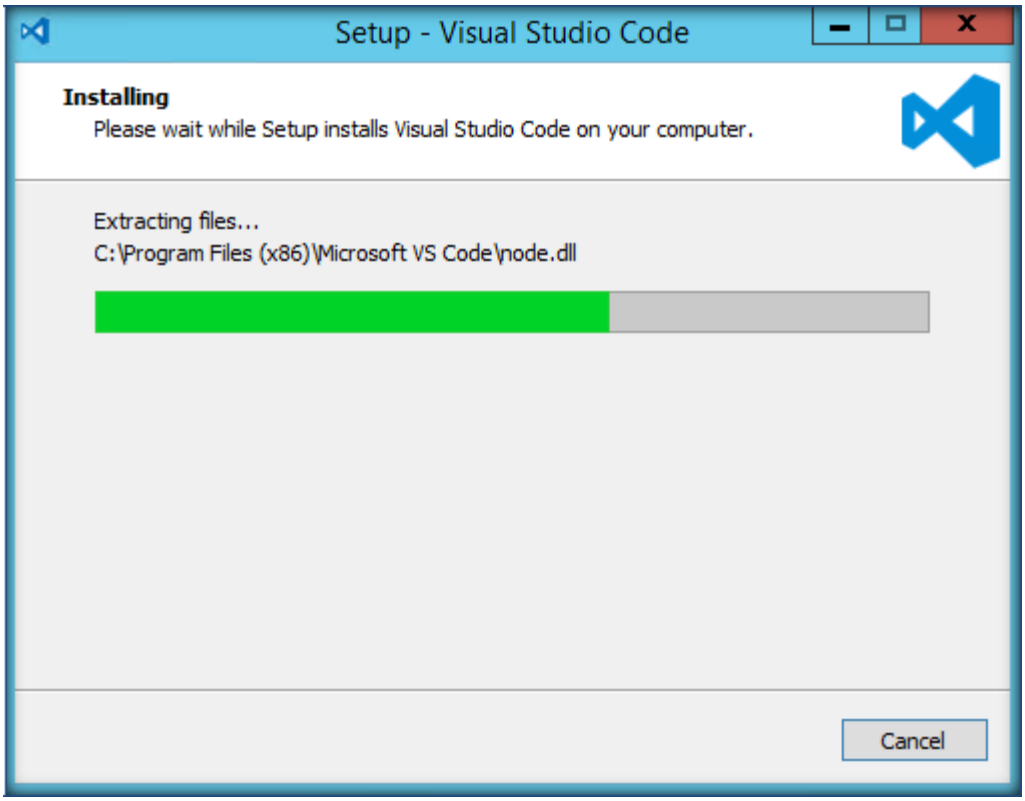
We will use Visual Studio Code in this walkthrough. You can get it from here.



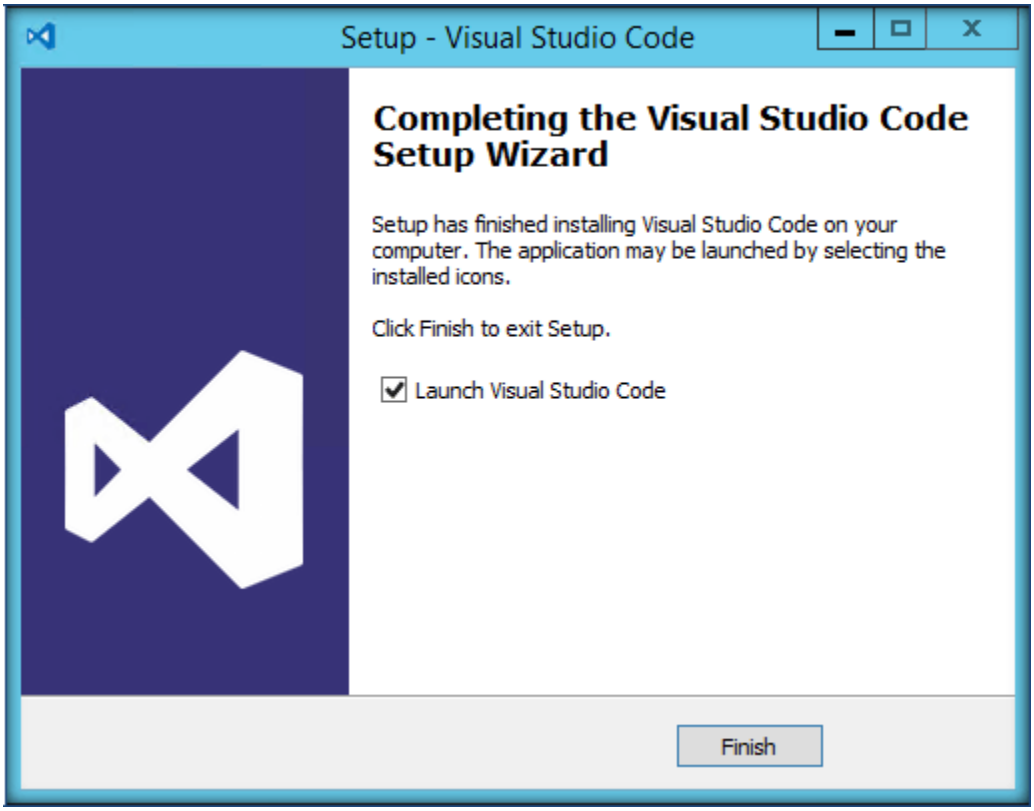
Once we have downloaded the exe, proceed with the installation.



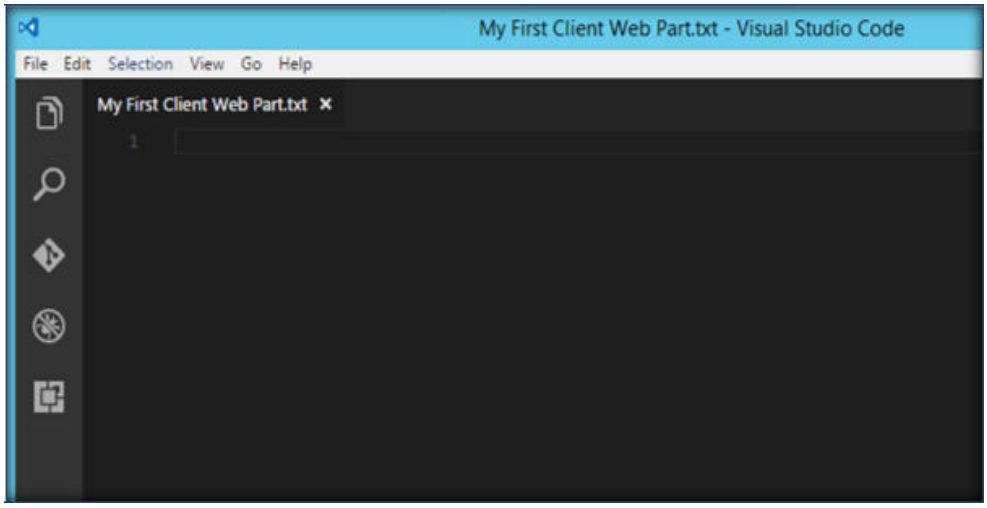
Click on Install to start the installation procedure.



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



Sample Screen Shot,



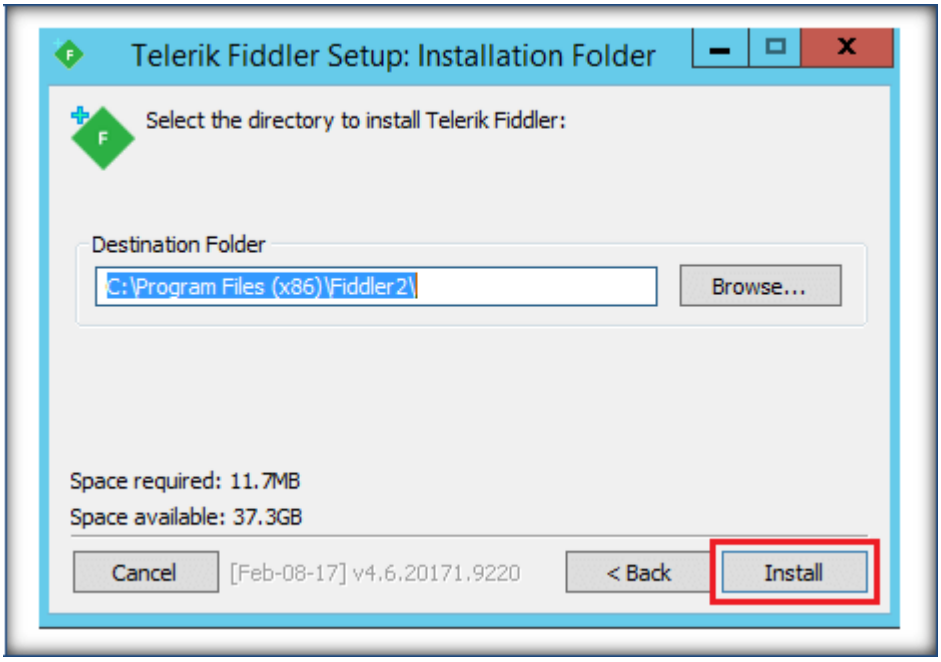
Additional Tools for Development and Debugging

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

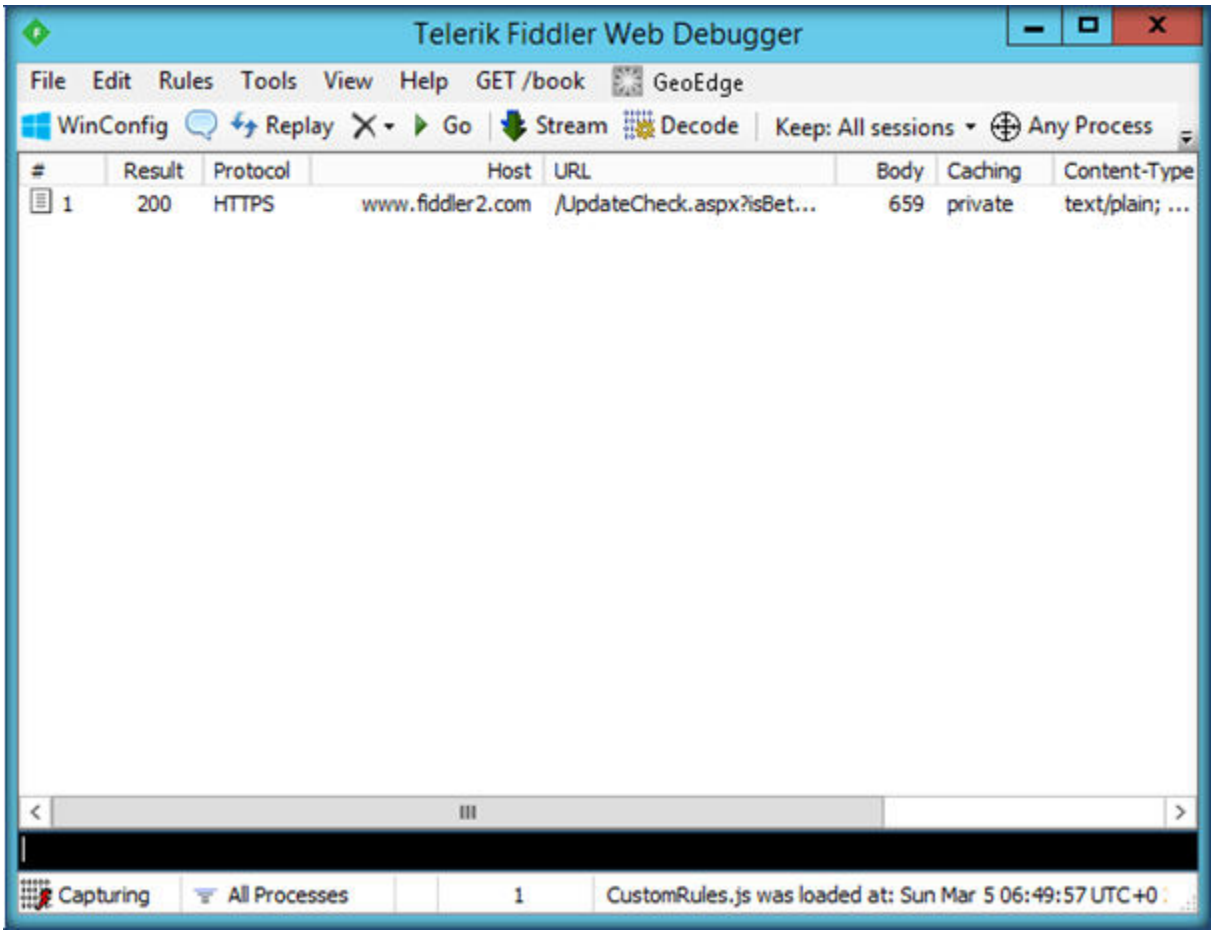
Fiddler

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.

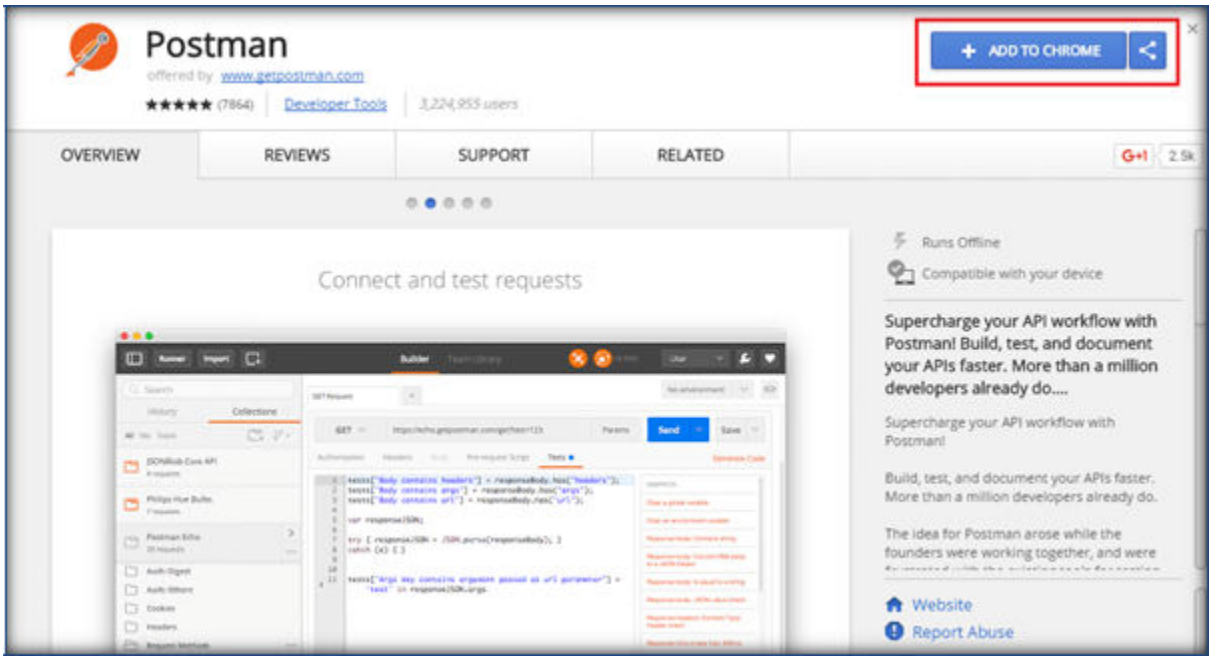


Using fiddler, we can examine the traffic as it is being sent or received.

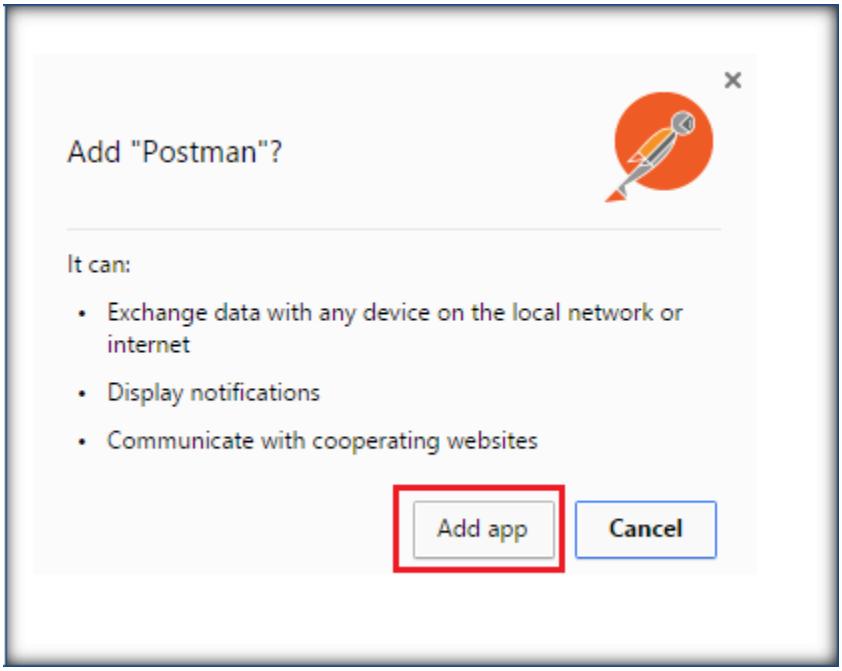


Postman

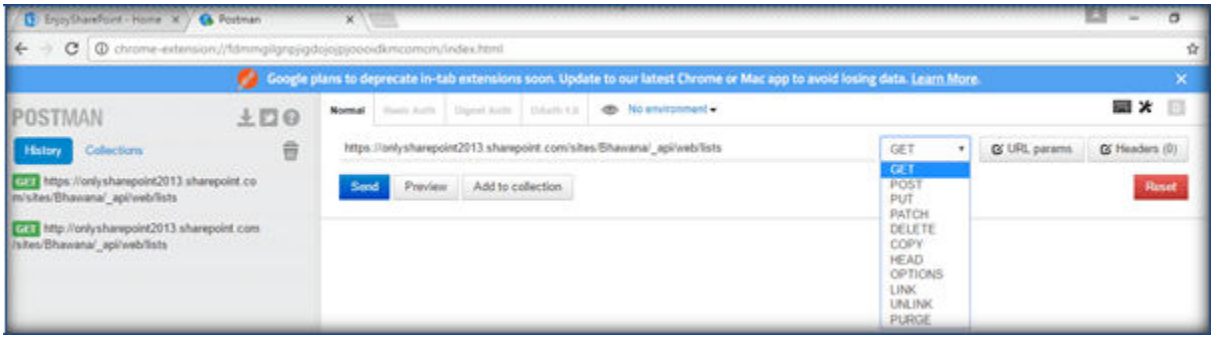
Postman can be used to test SharePoint's REST service endpoints and verify the returned data and request headers. We can get Postman from here



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

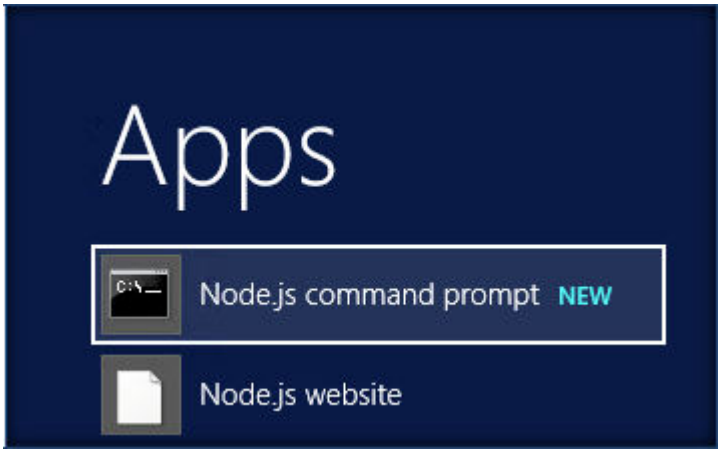
In the first section of the book we will be using TypeScript to build SharePoint Framework client webparts. As we proceed further, we will see how to use React JS and PnP JS to build the SPFx webparts

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

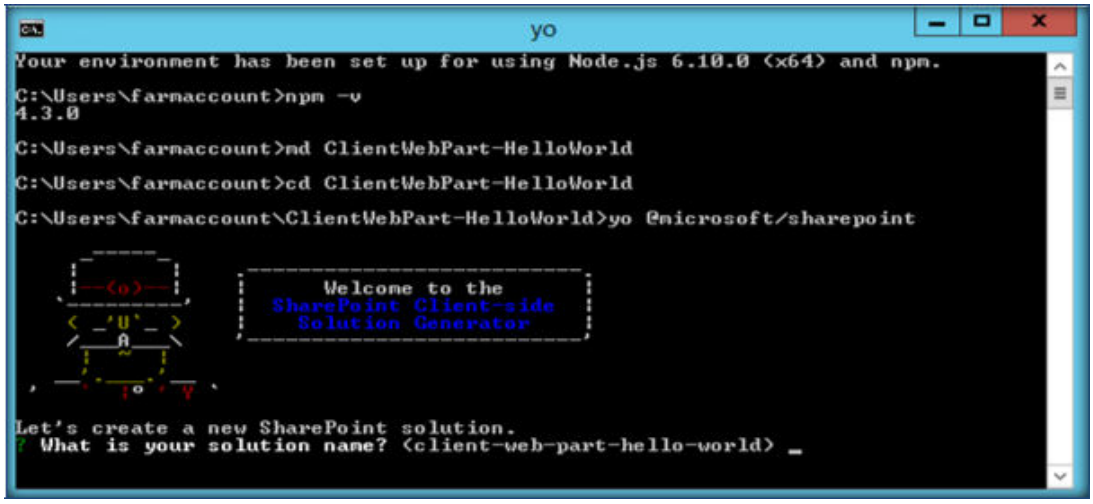
Before moving forward, ensure that the SharePoint Framework development environment is ready. Spin up Node.js command prompt using which we will be creating the web part project structure.



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

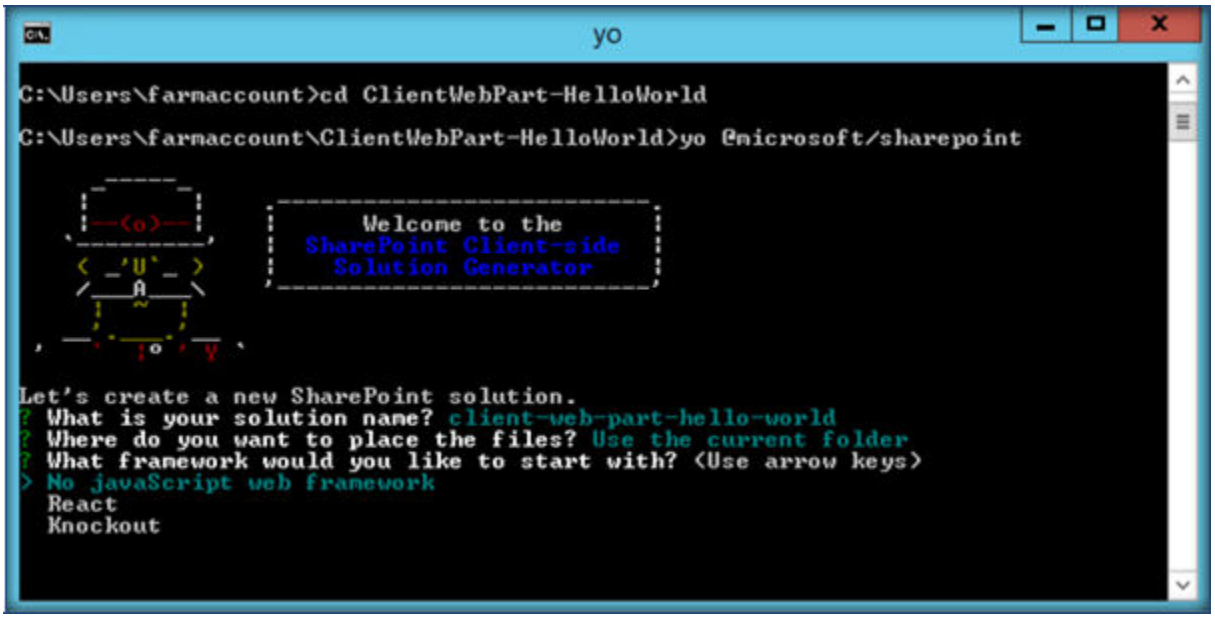
Let's move to the newly created working directory using the command:
cd ClientWebPart-HelloWorld

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



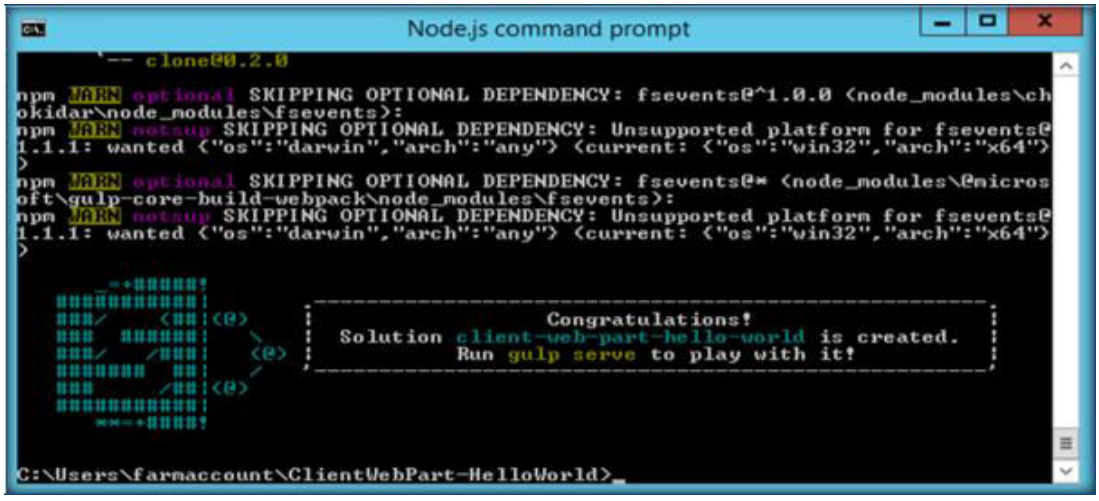
This will display the prompt which we will have to fill up so as to proceed with project creation,

- 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



- 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

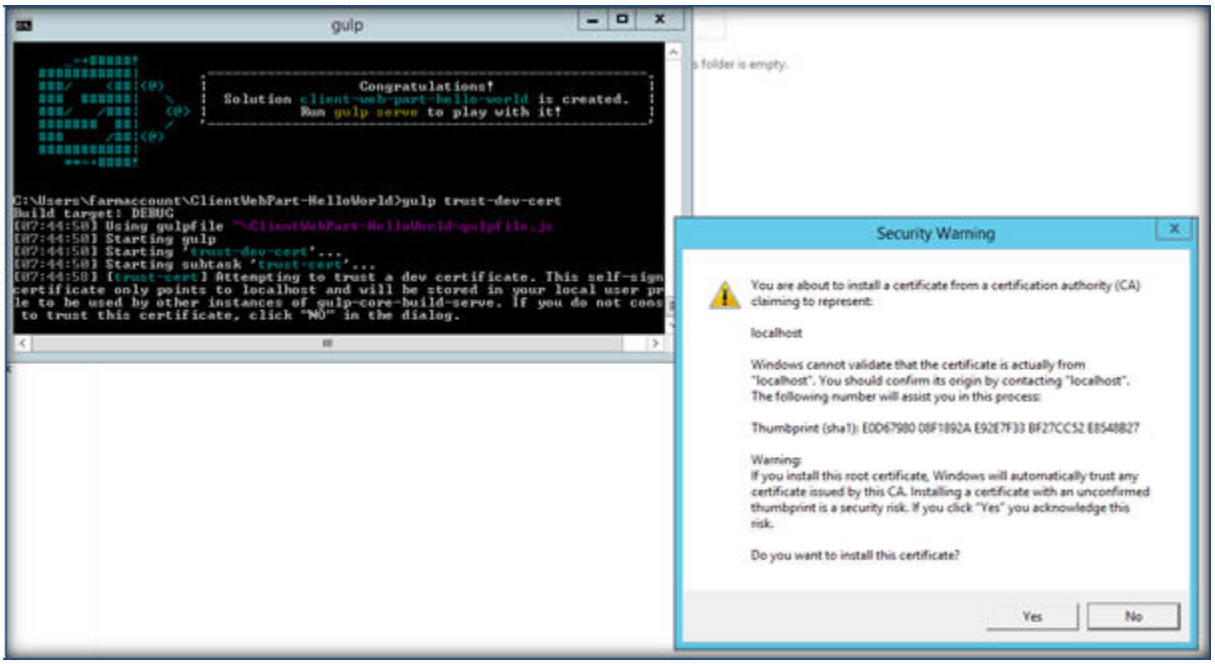
Yeoman has started working on the creation of the project. It will install the required dependencies and scaffold the solution files for the HelloWorld web part which will take some time to complete. Once completed, we will get a Congratulations message.



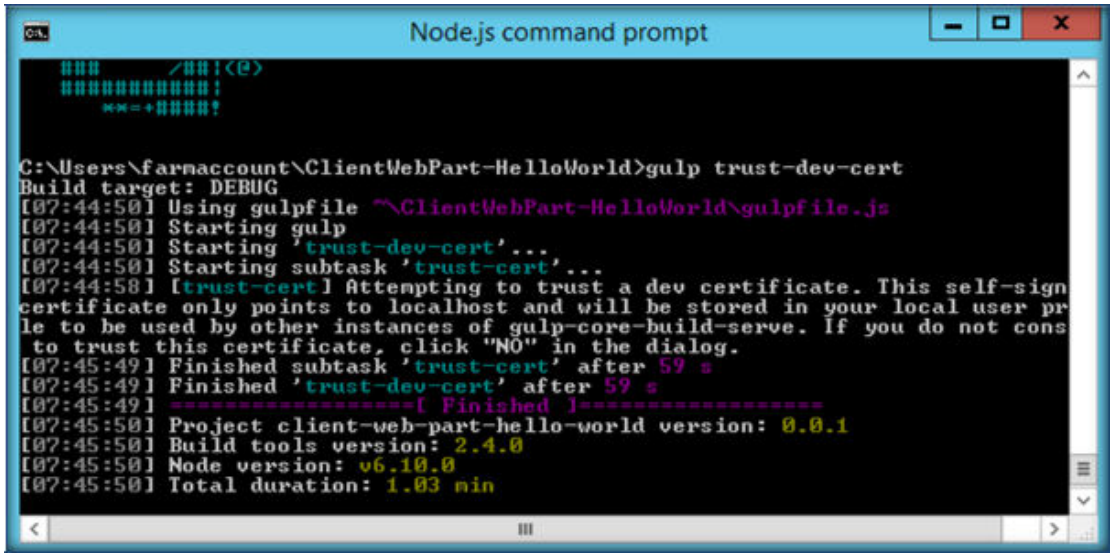
Test the web part

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:

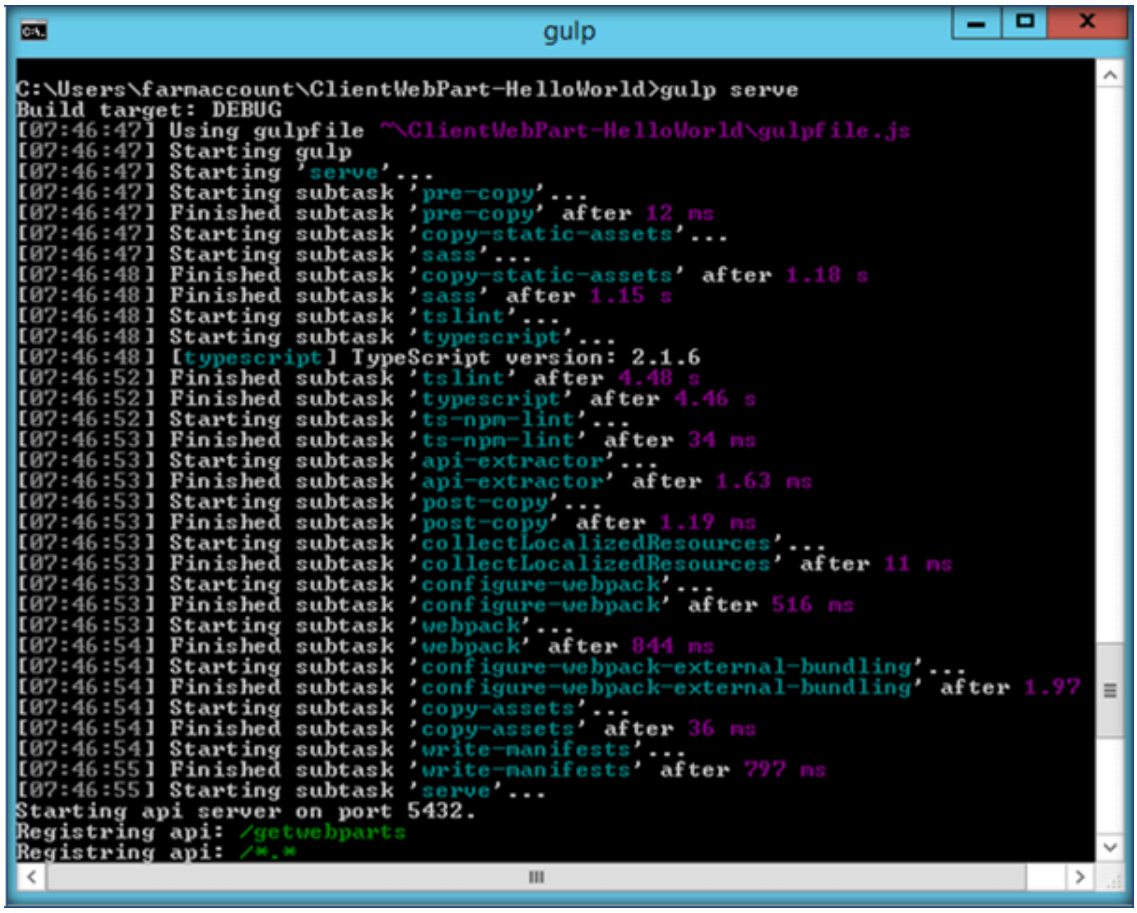
gulp trust-dev-cert



Click on Yes to install the certificate.



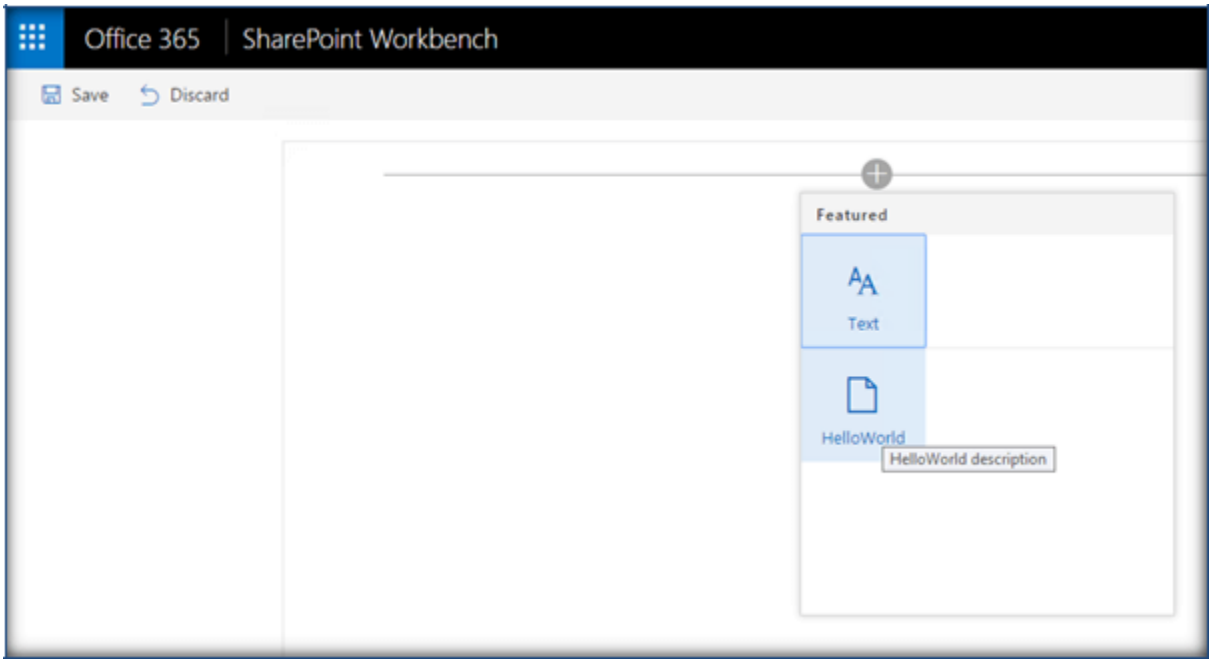
Now, let’s preview the web part by running the *gulp serve* command. This command will execute a series of gulp tasks and will create a Node-based HTTPS server at 'localhost:4321'. It will then open the browser and display the client web part.



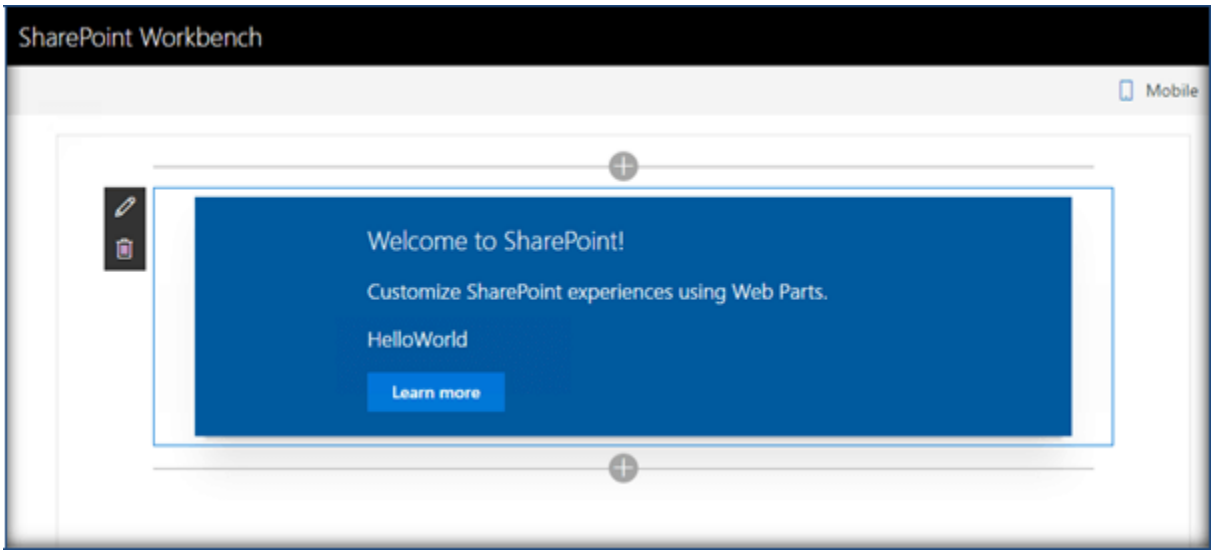
SharePoint Workbench

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.

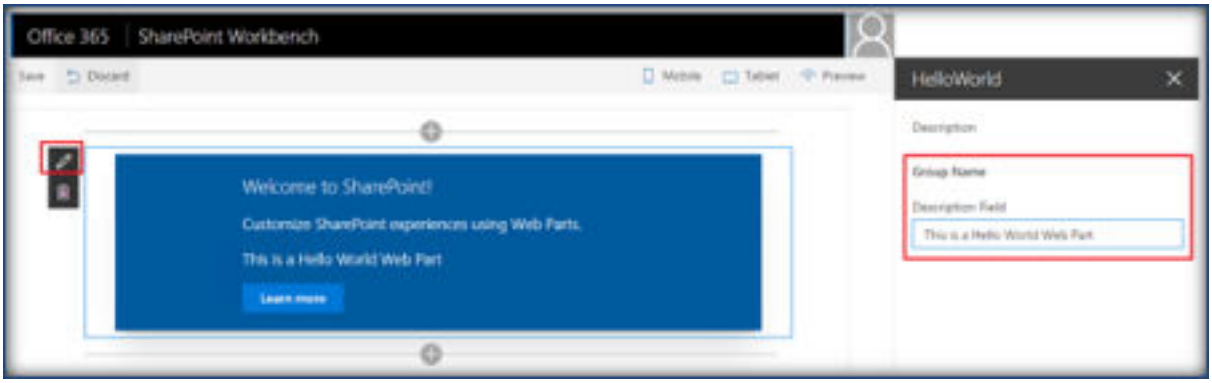
It will give us the option to add the Hello World web part that we have created recently.



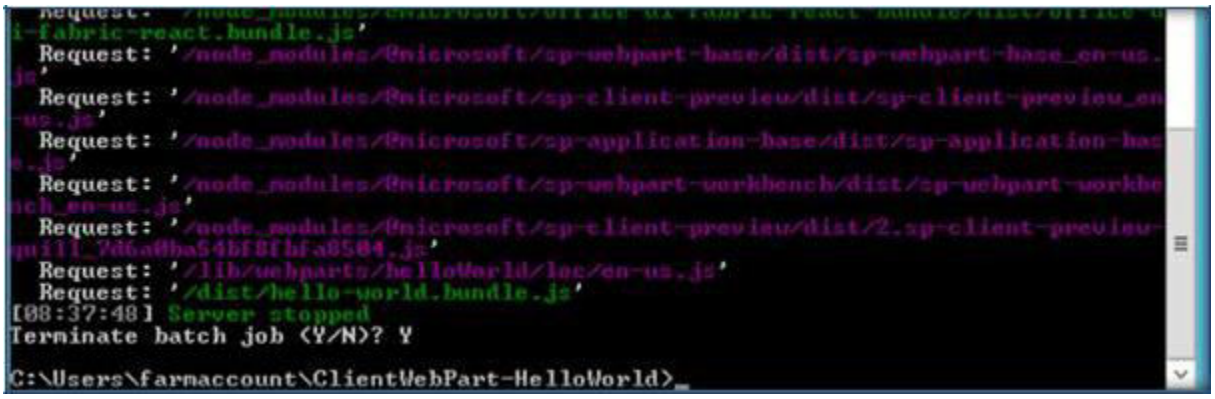
On clicking it, the web part will be added to the page. The web part contains few custom messages.



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.

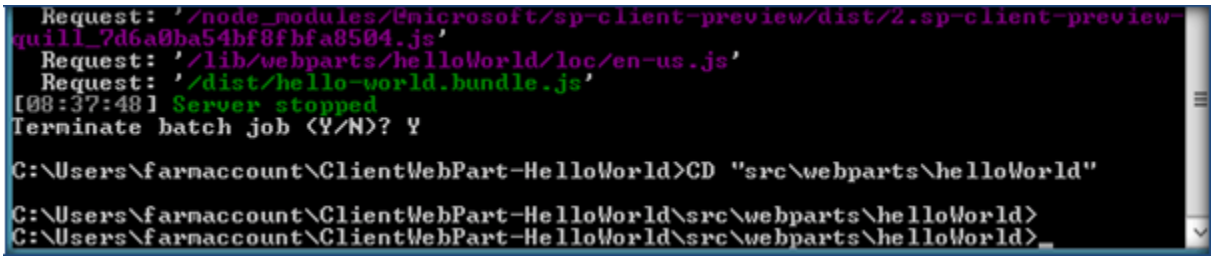


To stop Gulp from listening to the process we can press 'Control + C'. This will terminate the Gulp Serve command and stop the server.

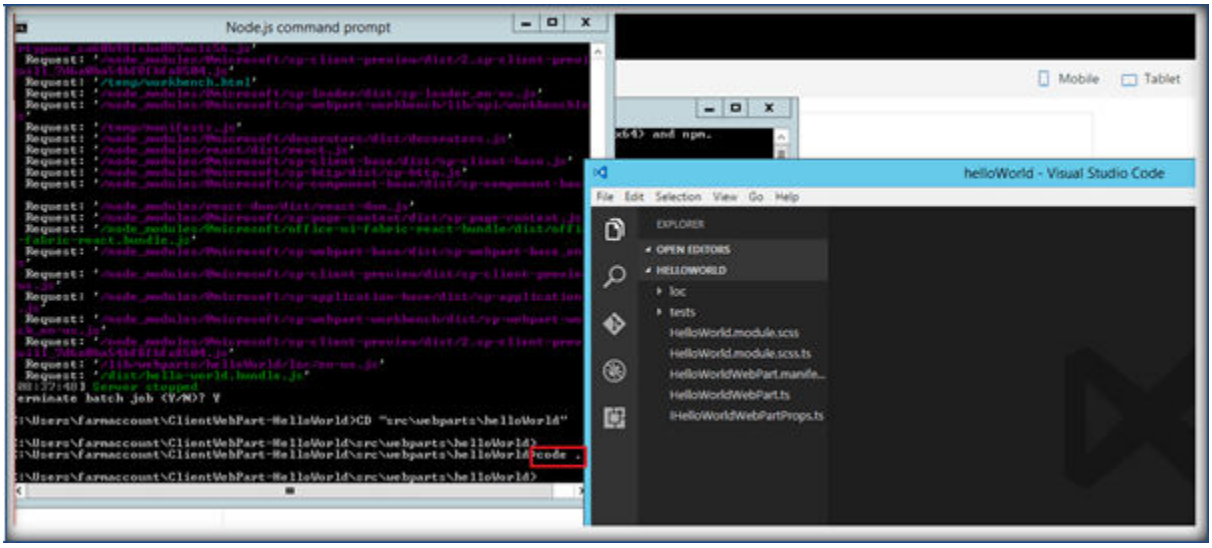


Edit the web part

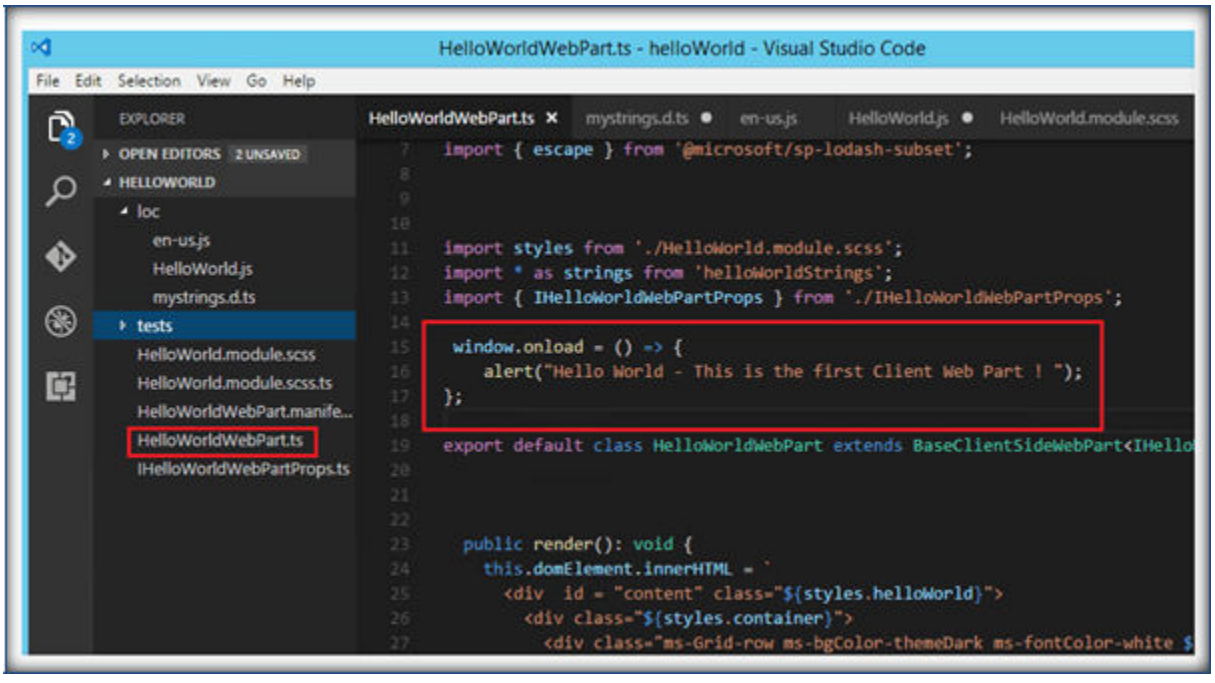
Now, let's try to edit the web part and add more functionality to it. To do that, navigate to 'src\webparts\helloWorld' location.



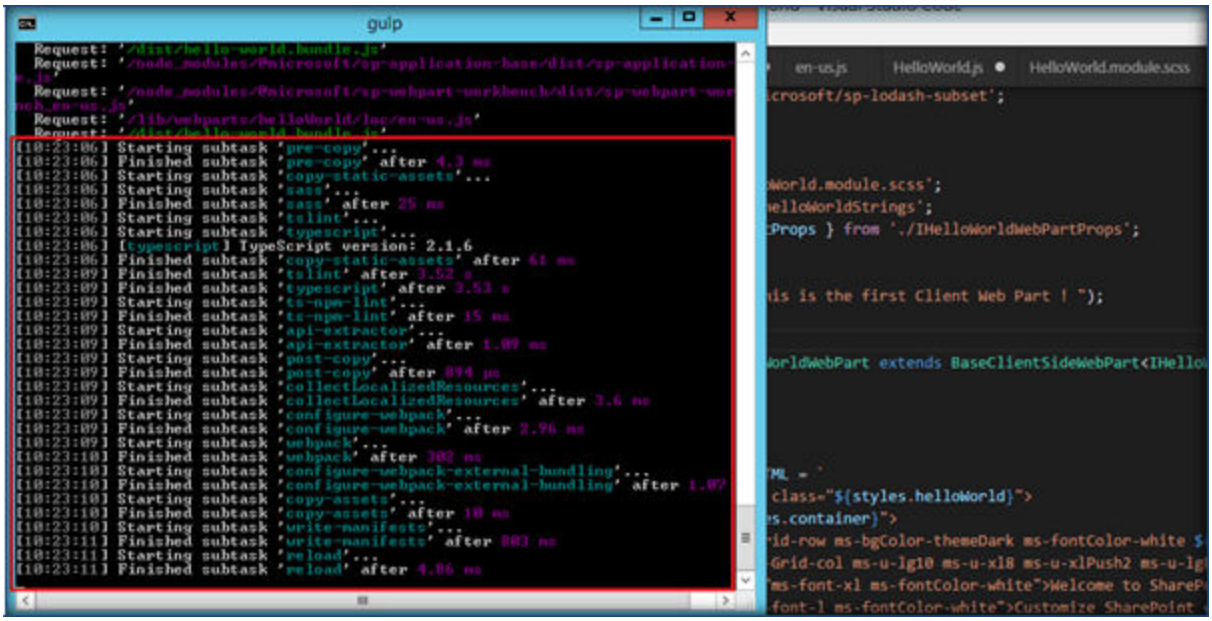
Run 'Code .' in the console which will open up the Visual Studio Code editor window.



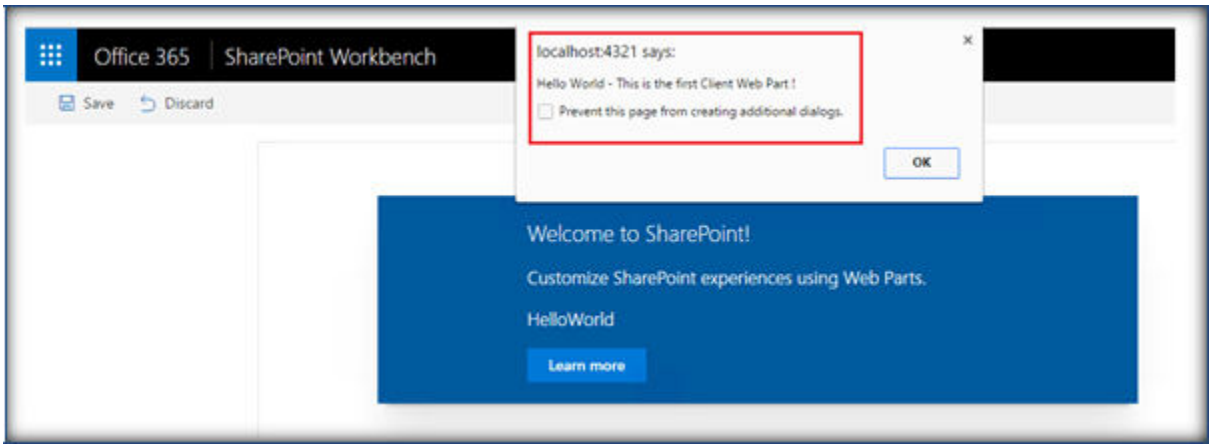
In the left pane of Visual Studio Code, we can see the project structure. The bulk of the logic resides within the *HelloWorldWebPart.ts* file. Let's add JavaScript code to alert a message within this typescript file.



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

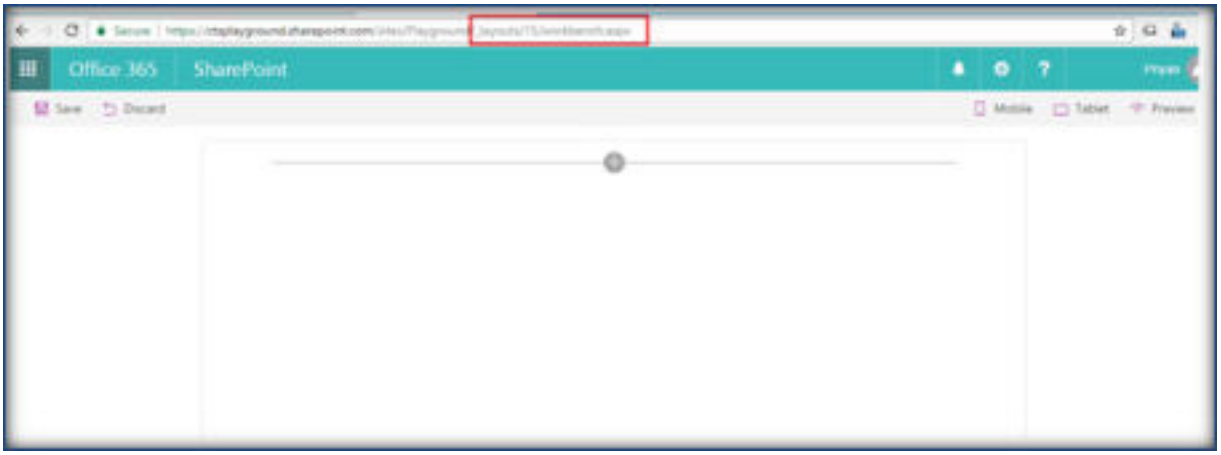


Again, running ‘gulp serve’ will display the updated web part in the browser. This time it will display the alert message as well.

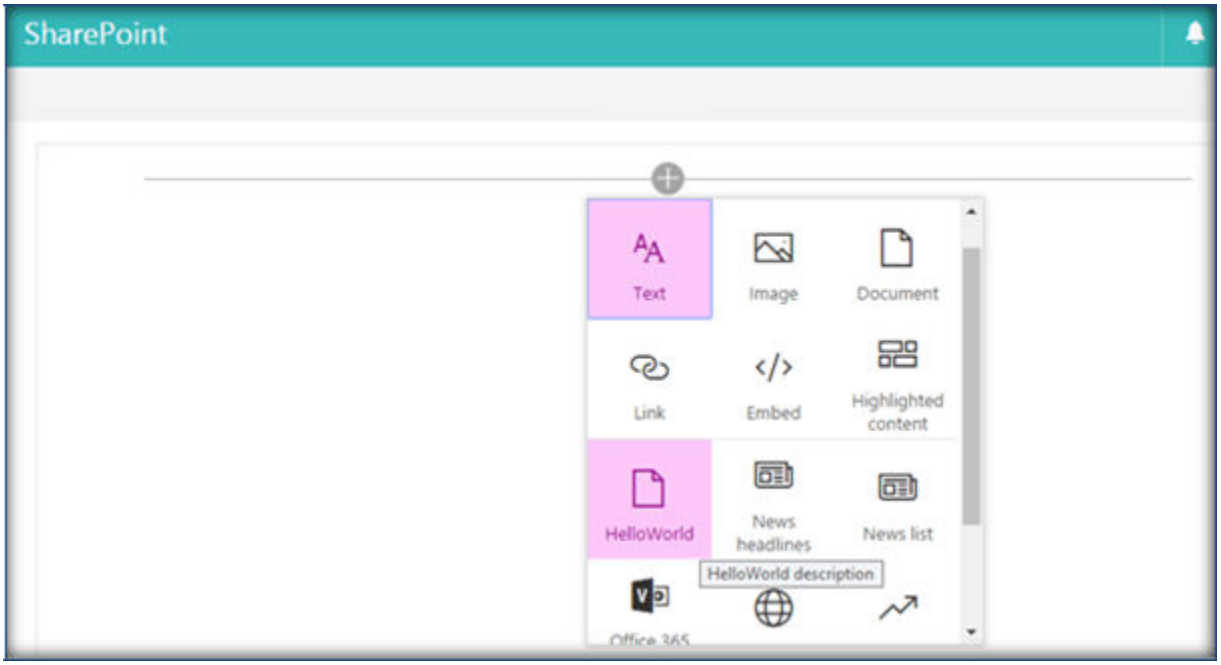


Add the web part to SharePoint

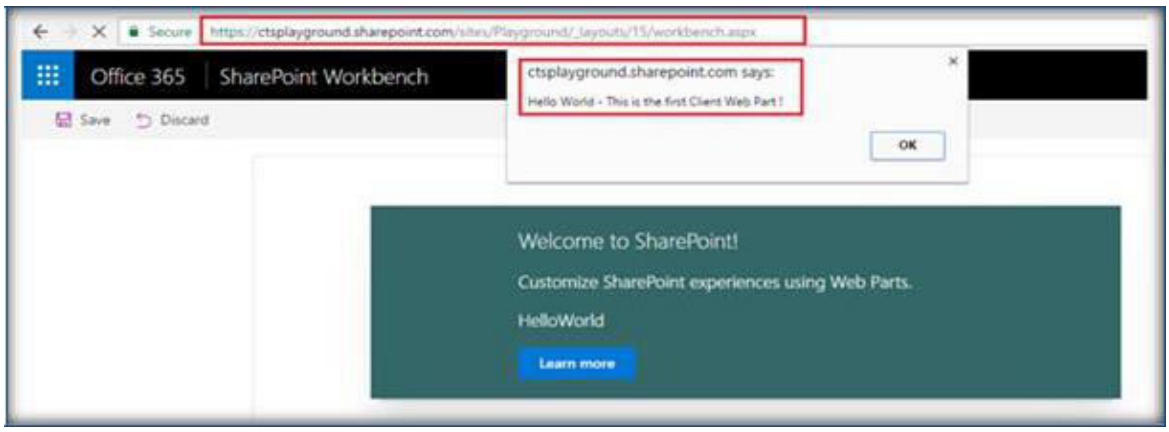
So far we were testing the web part in SharePoint Workbench locally, now let’s try to test it within the SharePoint Context. SharePoint Workbench is also hosted in SharePoint Online to preview the web part. It can be accessed by adding ‘ _layouts/15/workbench.aspx’ to the SharePoint Online URL.



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



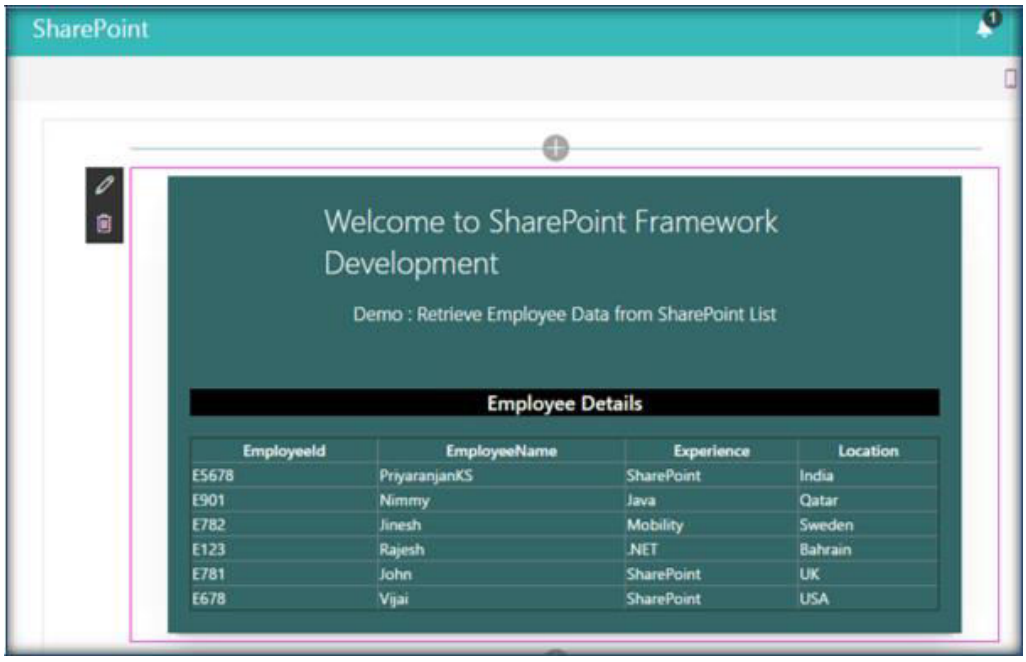
The web part has triggered the alert message in the page indicating successful hosting of the web part within SharePoint.



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

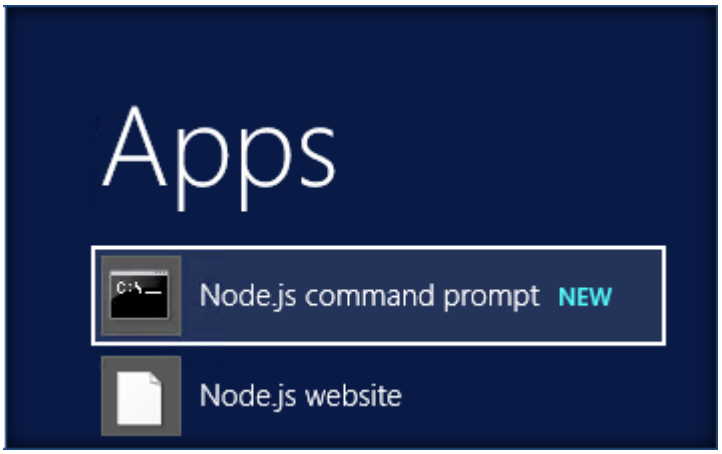
In this section, we will be creating a client Web part using TypeScript, which will be retrieving the list items from SharePoint List (EmployeeList) and display it in the tabular form in the client Web part, as shown below.



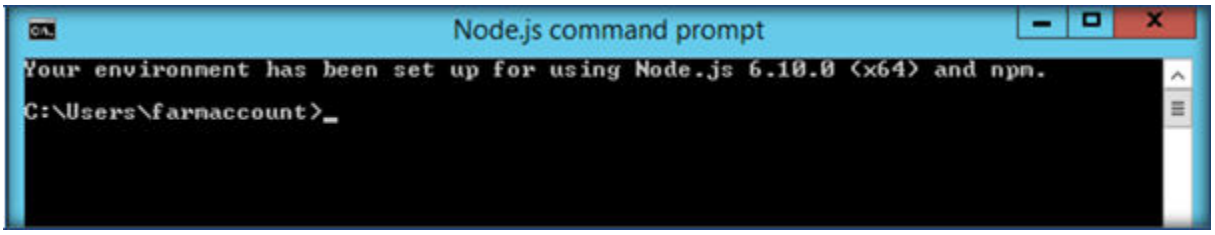
The Solutions files used in this section is zipped and uploaded to the Microsoft TechNet gallery. Feel free to download it.

Create the Web part Project

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

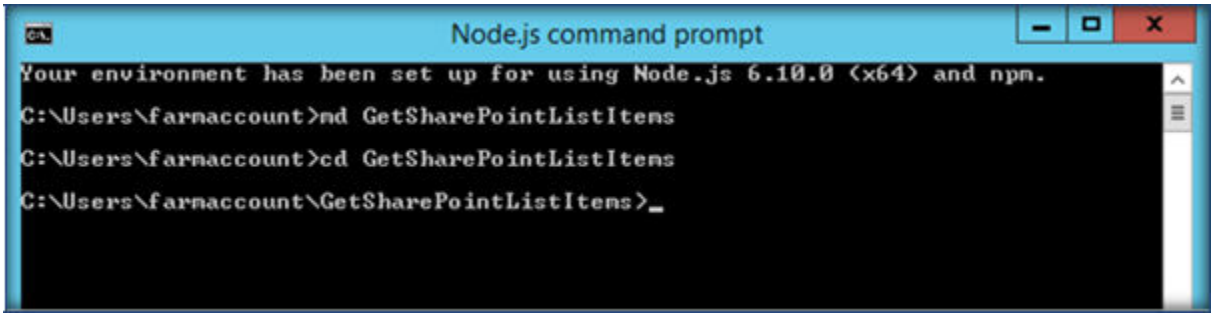


This will open the console where we can create the SharePoint Framework project structure.

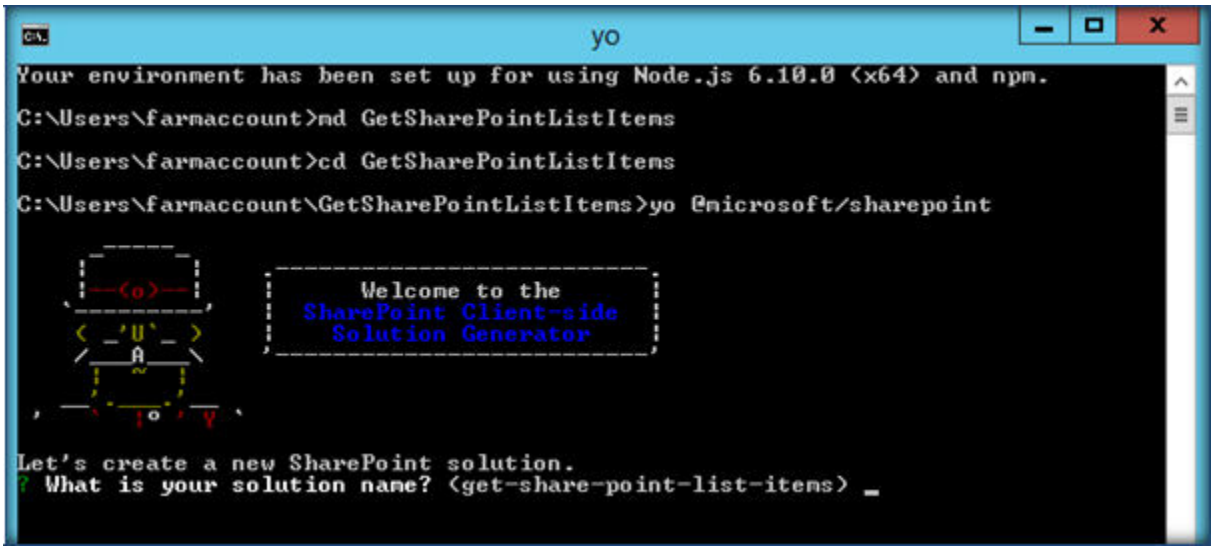


We can create the directory, where we will be adding the solution, using the command given below.
md GetSharePointListItems

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



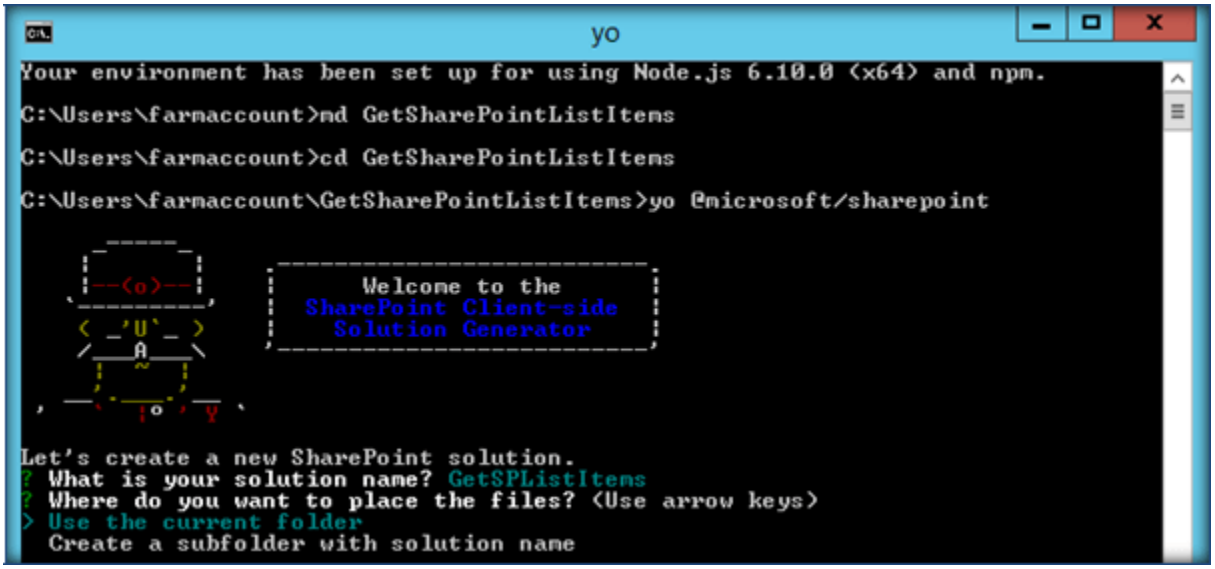
We will then create the client Web part by running the Yeoman SharePoint Generator. *yo @microsoft/sharepoint*



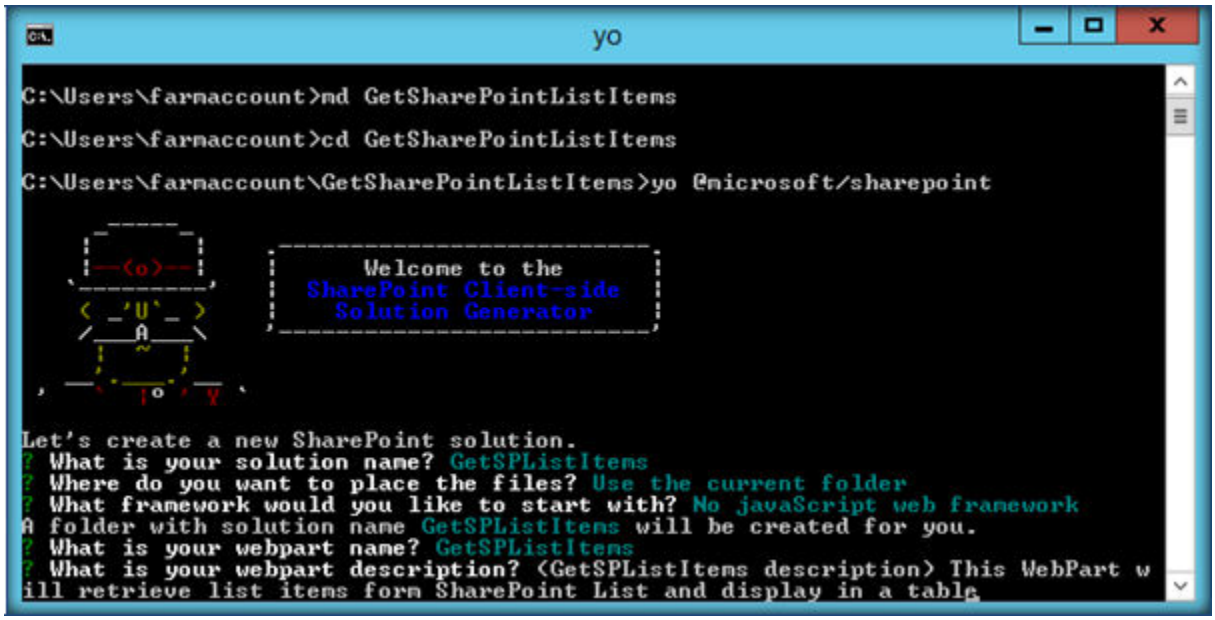
This will display the prompt, which we must fill up, to proceed with the project creation.

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

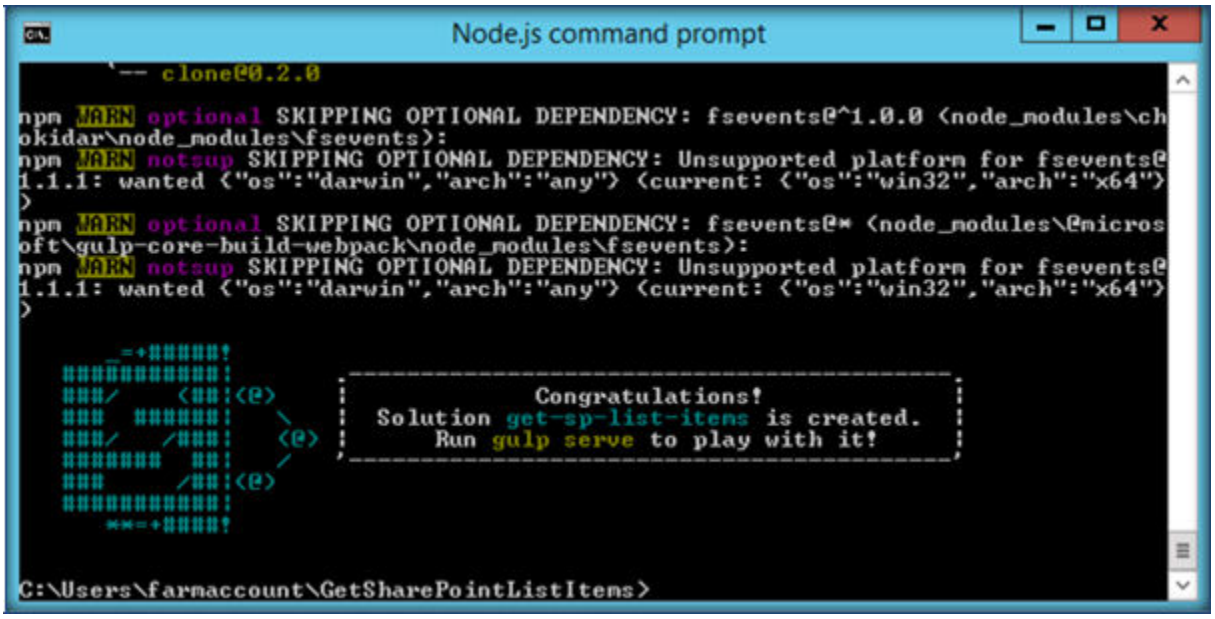
On pressing enter, we will be asked to chose the working folder for the project.



- 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



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.

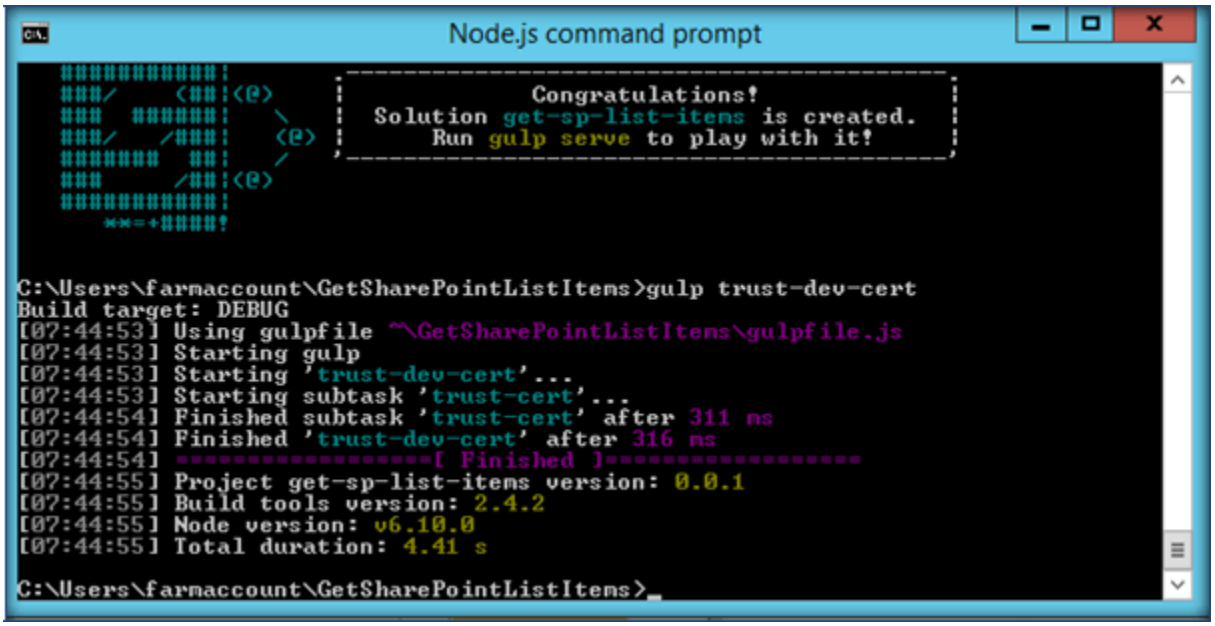


Test the Web part locally

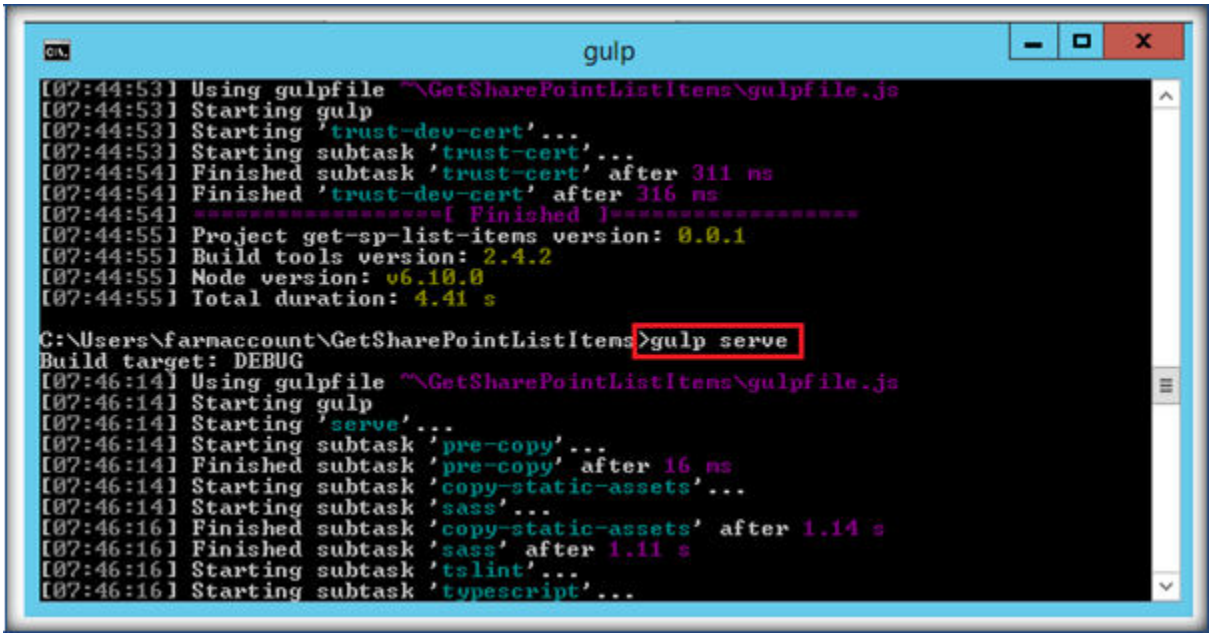
To test the client Web part, we can build and run it on the local Web Server, where we are developing the Web part. SharePoint Framework development uses HTTPS endpoint by default. Since a default certificate is not configured for the local development environment, our Browser will report a

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.

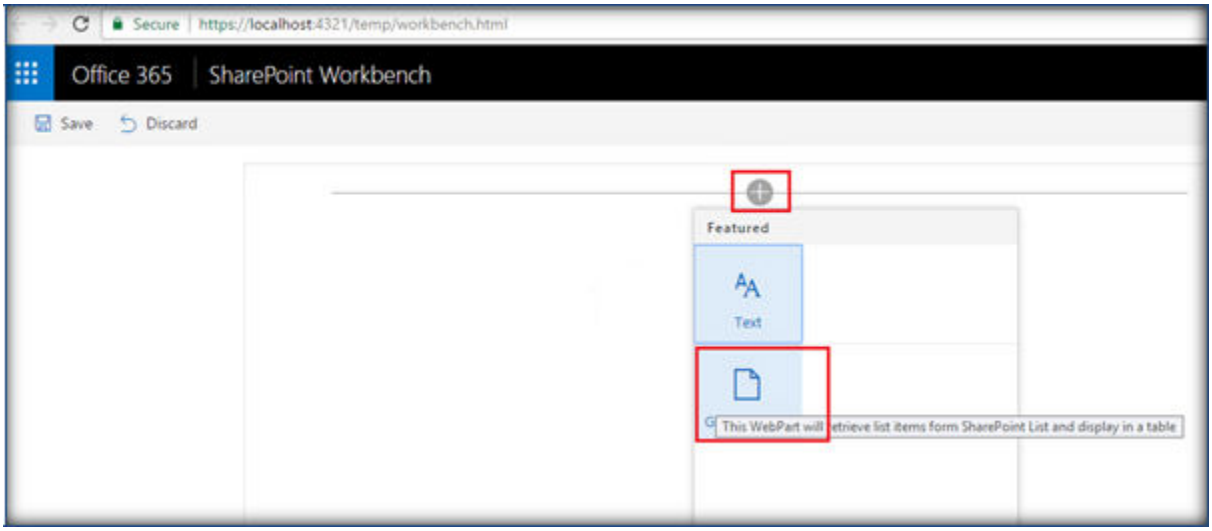
gulp trust-dev-cert



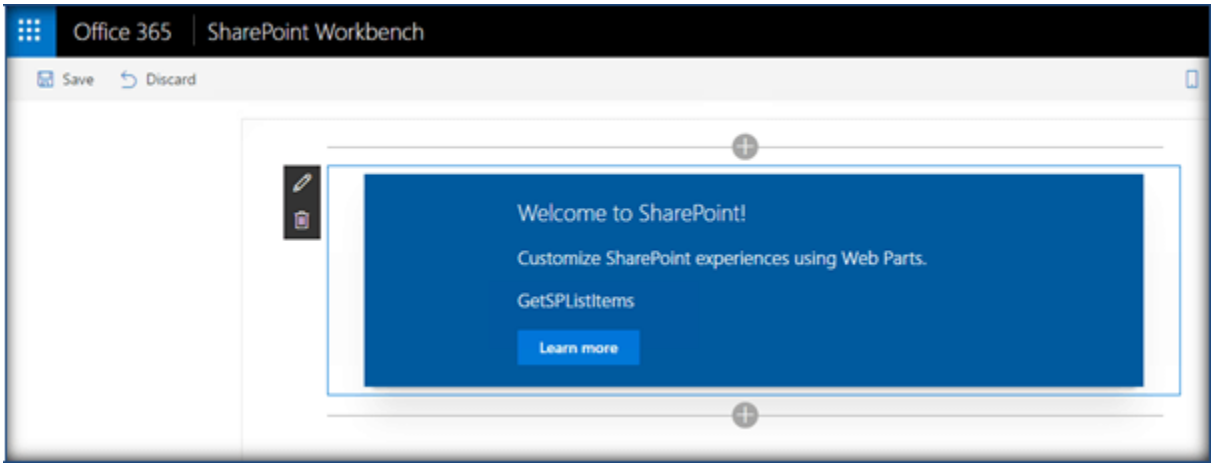
Now, let's preview the Web part by running the *gulp serve* command.



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.



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.

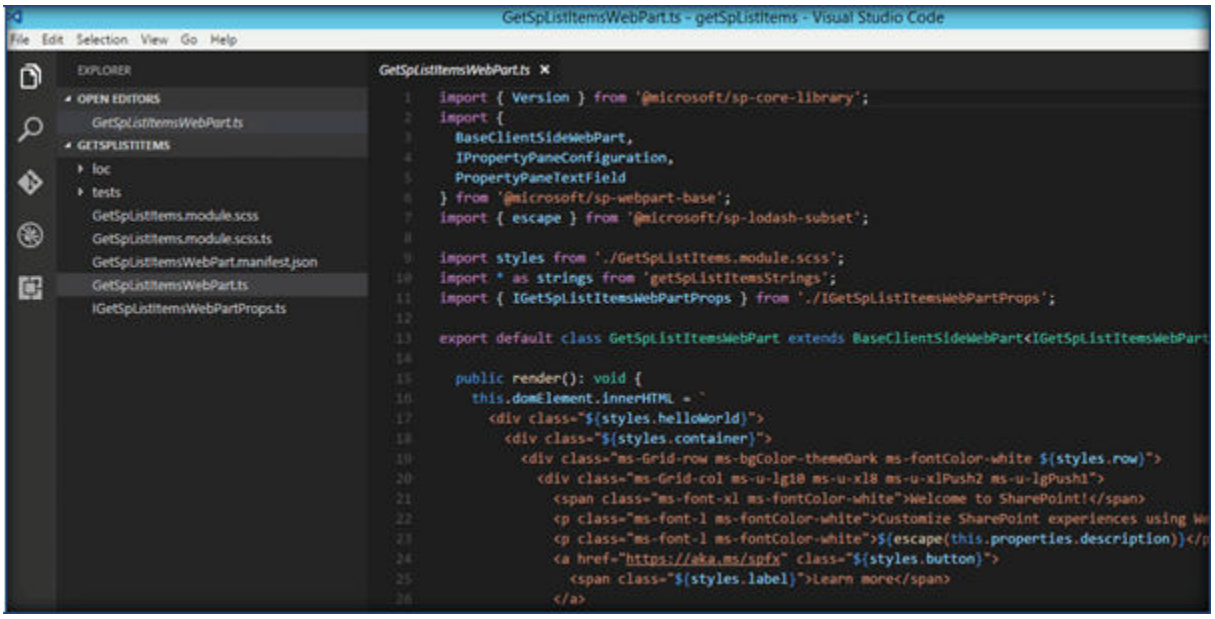


To stop Gulp from listening to the process, we can press '*Control + C'*. This will terminate the Gulp Serve command and stop the Server.

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.

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.



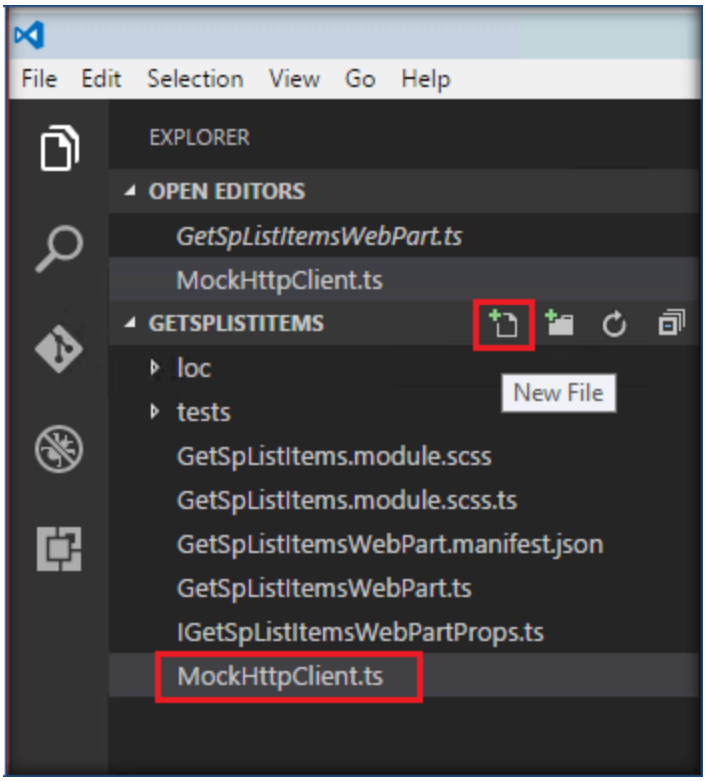
Define List Model

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.    }
```

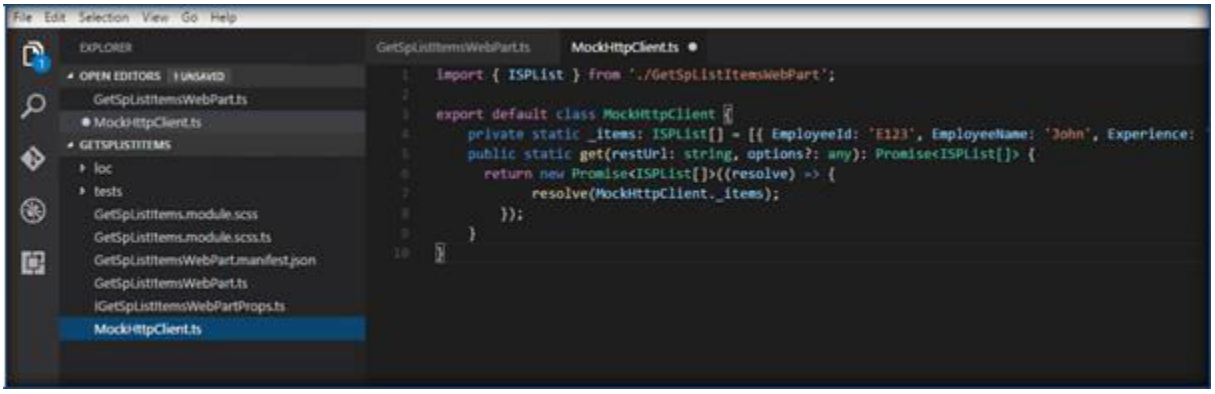
Create Mock HTTPClient to test data locally

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',
    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 '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:
    'SharePoint',Location: 'India' },
7.                   { EmployeeId: 'E567', EmployeeName: 'Martin', Experience:
    '.NET',Location: 'Qatar' },
8.                  { EmployeeId: 'E367', EmployeeName: 'Luke', Experience:
    'JAVA',Location: 'UK' }
9.              ]
10.             };
11.         return listData;
12.     }) as Promise<ISPLists>;
13. }
```

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.      }
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 'GetSpListItemsWebPart' 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</span>
8.
9.  <p class="ms-font-l ms-fontColor-white" style="text-align: center">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

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 './GetSpListItems.module.scss';
16. import * as strings from 'getSpListItemsStrings';
17. import { IGetSpListItemsWebPartProps } from './IGetSpListItemsWebPartProps';
18. import MockHttpClient from './MockHttpClient';
19.
20. import {
21.   SPHttpClient
22. } from '@microsoft/sp-http';
23.
24.
25.       export interface ISPLists {
```

```
        value: ISPList[];
      }
      export interface ISPList {
        EmployeeId: string;
        EmployeeName: string;
        Experience: string;
        Location: string;
      }

export default class GetSpListItemsWebPart extends BaseClientSideWebPart<IGetSpListItemsWebPartProps> {

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

  private _renderListAsync(): void {

    if (Environment.type === EnvironmentType.Local) {
      this._getMockListData().then((response) => {
        this._renderList(response.value);
      });
    }
     else {
       this._getListData()
      .then((response) => {
        this._renderList(response.value);
      });
   }
}

private _renderList(items: ISPList[]): void {
let html: string = '<table class="TFtable" border=1 width=100% style="border-collapse: collapse;">';
html += `<th>EmployeeId</th><th>EmployeeName</th><th>Experience</th><th>Location</th>`;
  items.forEach((item: ISPList) => {
    html += `
         <tr>
        <td>${item.EmployeeId}</td>
        <td>${item.EmployeeName}</td>
        <td>${item.Experience}</td>
        <td>${item.Location}</td>
        </tr>
        `;
  });
  html += `</table>`;
  const listContainer: Element = this.domElement.querySelector('#spListContainer');
  listContainer.innerHTML = html;
}

  public render(): void {
     this.domElement.innerHTML = `
     <div class="${styles.helloWorld}">
  <div class="${styles.container}">
    <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
```

```
      <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
        <span class="ms-font-xl ms-fontColor-white" style="font-size:28px">Welcome to SharePoint Framework Development</span>

<p class="ms-font-l ms-fontColor-white" style="text-align: center">Demo : Retrieve Employee Data from SharePoint List</p>
      </div>
    </div>
    <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
<div style="background-color:Black;color:white;text-align: center;font-weight: bold;font-size:18px;">Employee Details</div>
    <br>
<div id="spListContainer" />
    </div>
  </div>
</div>`;
this._renderListAsync();
  }

private _getMockListData(): Promise<ISPLists> {
          return MockHttpClient.get(this.context.pageContext.web.absoluteUrl).then(() => {
              const listData: ISPLists = {
                  value:
                  [
                      { EmployeeId: 'E123', EmployeeName: 'John', Experience: 'SharePoint',Location: 'India' },
                       { EmployeeId: 'E567', EmployeeName: 'Martin', Experience: '.NET',Location: 'Qatar' },
                      { EmployeeId: 'E367', EmployeeName: 'Luke', Experience: 'JAVA',Location: 'UK' }
                  ]
                  };
              return listData;
          }) as Promise<ISPLists>;
      }

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

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

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

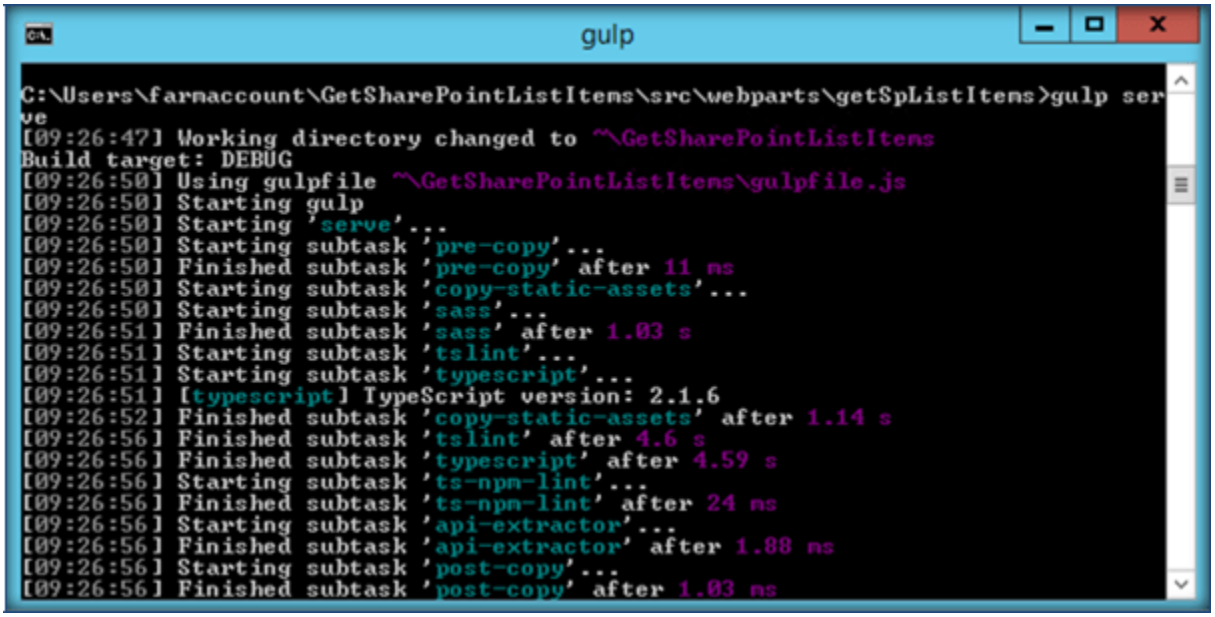
Mock HTTP Client Content

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

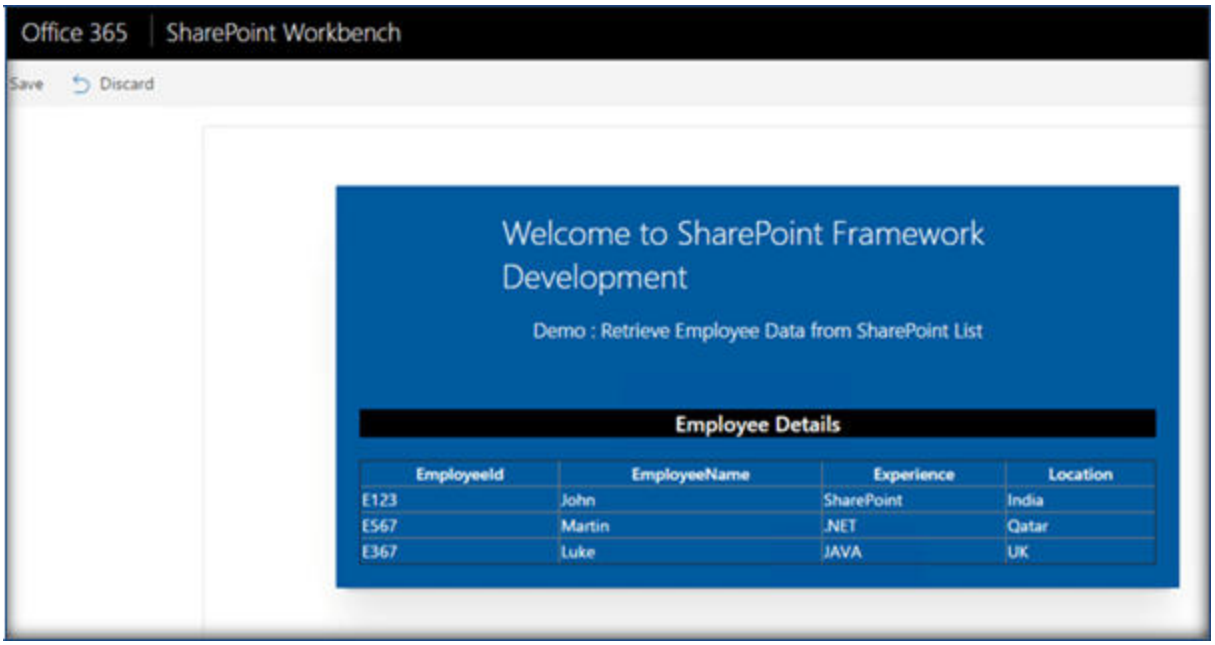
```
1.  import { ISPList } from './GetSpListItemsWebPart';
2.
3.  export default class MockHttpClient {
4.  private static _items: ISPList[] = [{ EmployeeId: 'E123', EmployeeName: 'John',
    Experience: 'SharePoint',Location: 'India' },];
5.      public static get(restUrl: string, options?: any): Promise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

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.



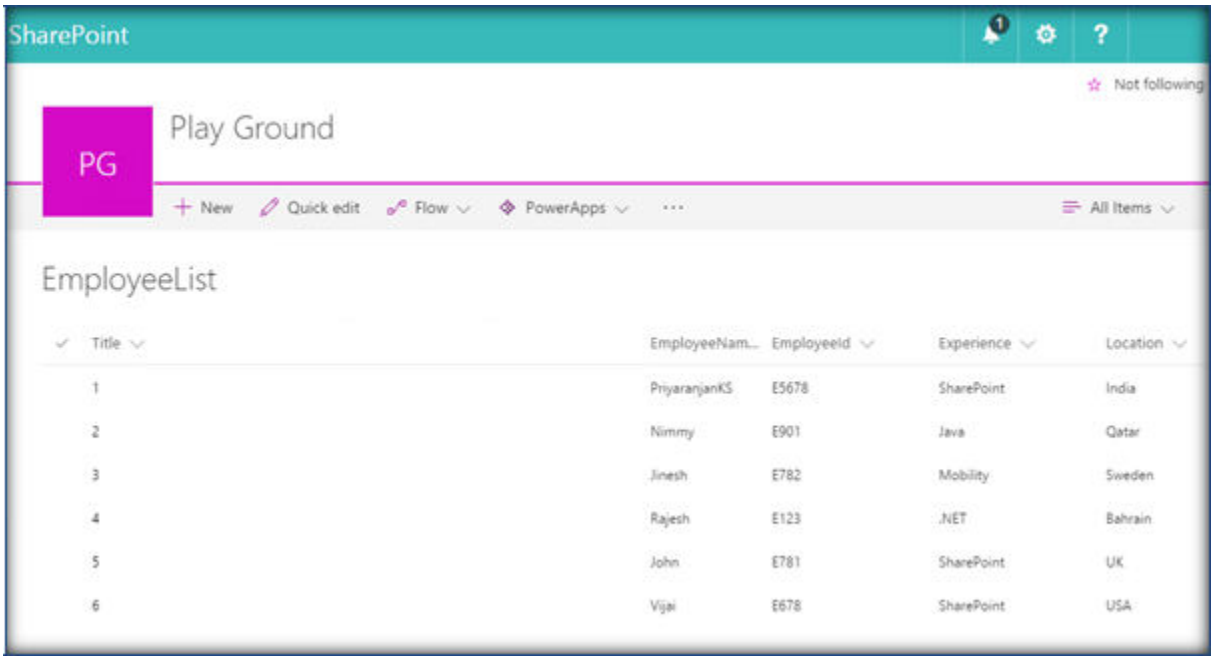
Since the environment is local, the mock data has been used to generate the table, as shown below.



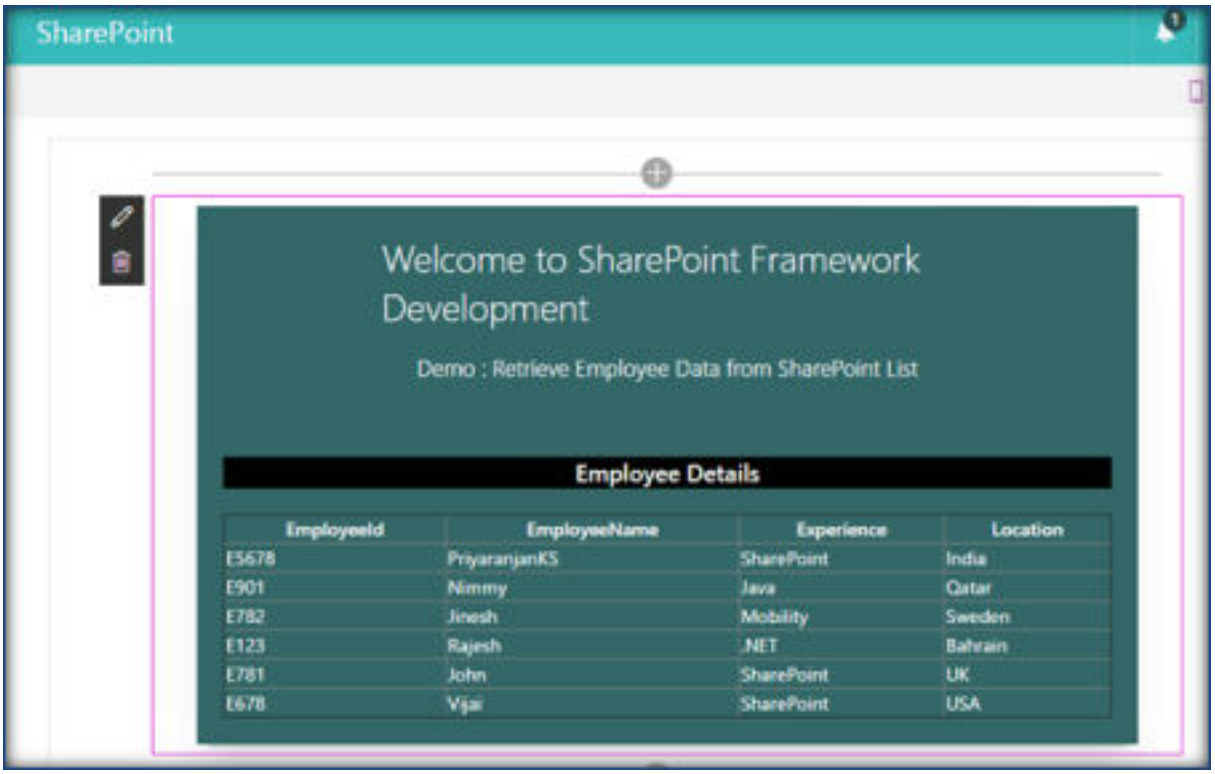
Thus, we have successfully tested the client Web part locally.

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. SharePoint Online list - EmployeesList , to which we are trying to connect, using REST API is given below.



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.



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

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.

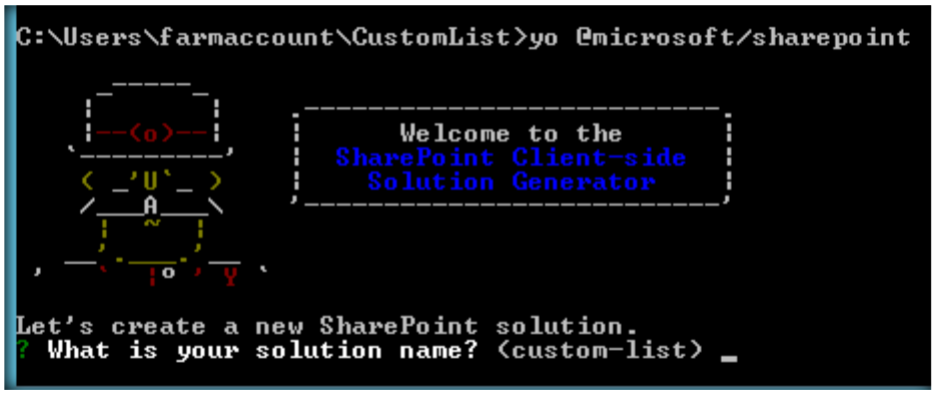
Create the Web part Project

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*

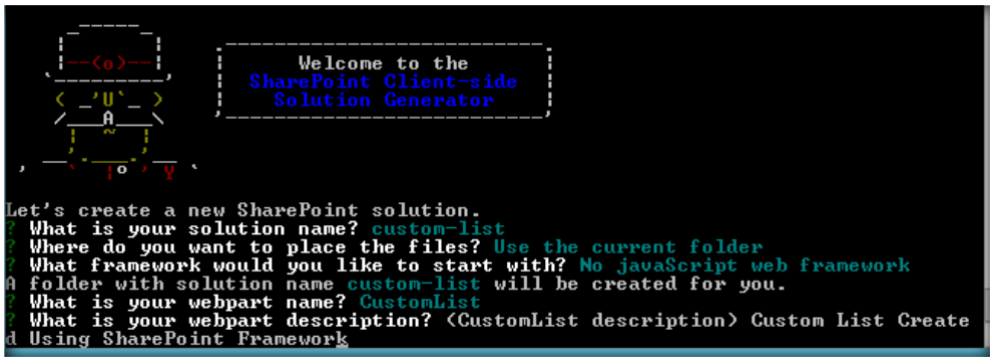


This will display the prompt, which we will have to fill up, so as to proceed with the project creation.

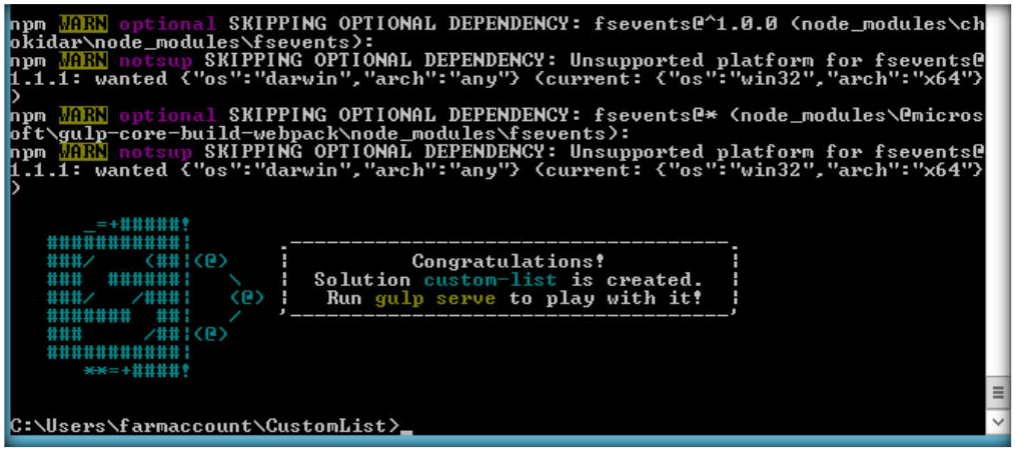
- What is your solution name? : Set it to 'CustomList'.

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

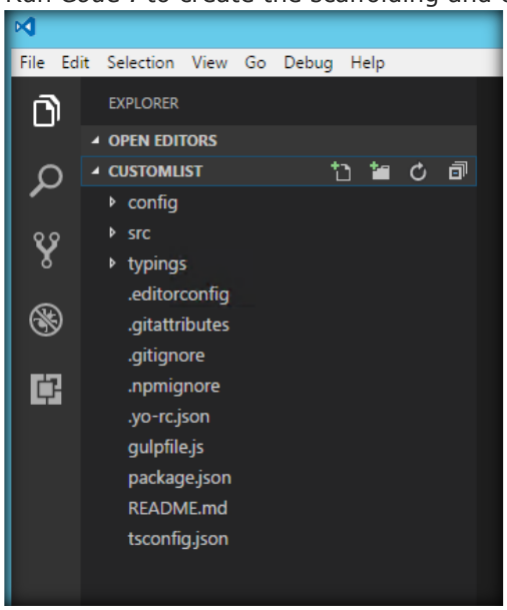
- Where do you want to place your files- Use current folder.
- What framework would you like to start with- Select "No javaScript web framework" for the time being, as this is a sample Web part.
- What is your Webpart name- We will specify it as 'CustomList 'and press Enter
- What is your Webpart description-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



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.

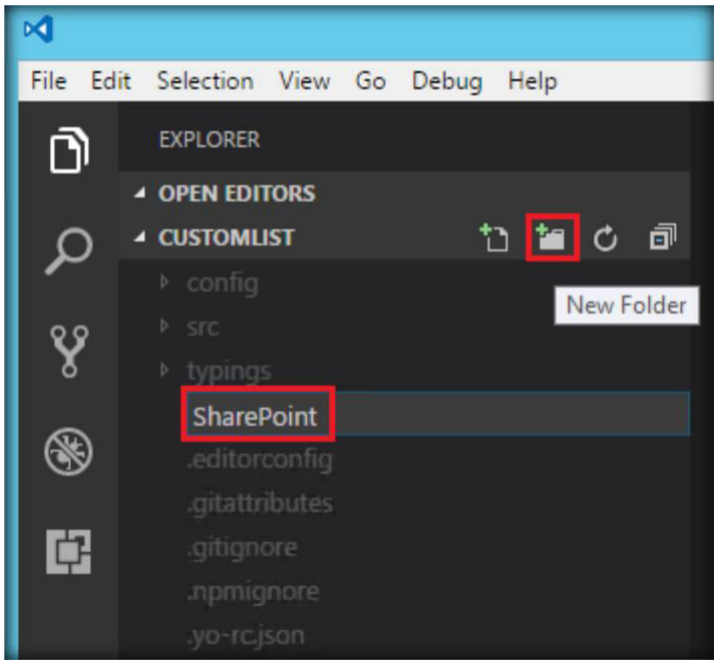


Run *Code .* to create the scaffolding and open the project in Visual Studio Code

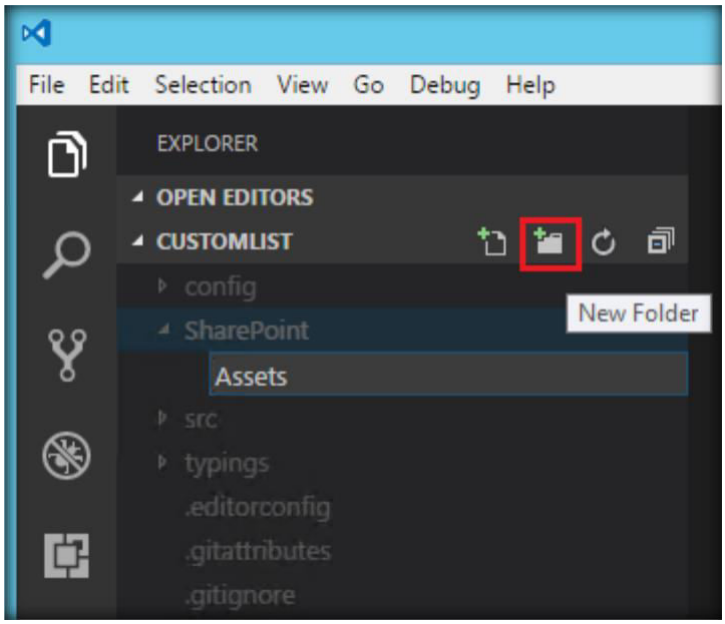


Edit the web part

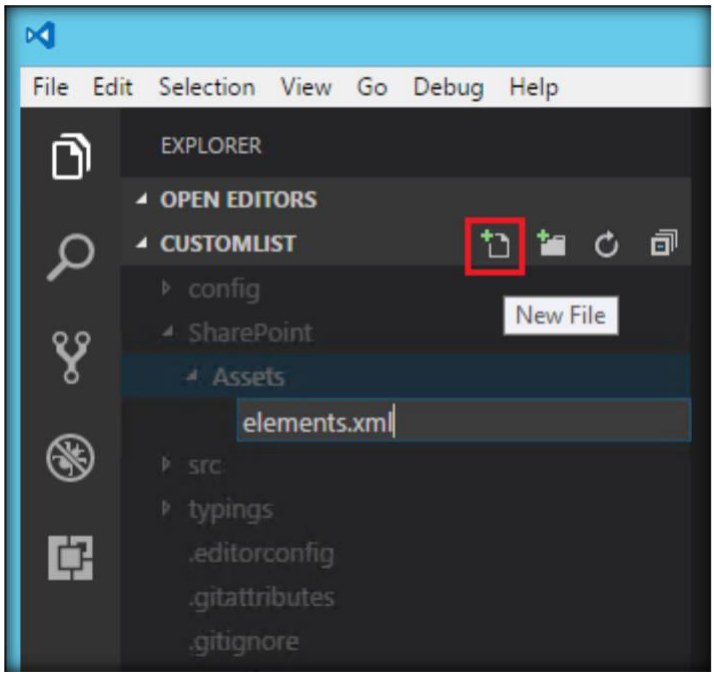
Now let’s add the folder named ‘SharePoint’ to maintain the SharePoint files that will be deployed as a package.



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

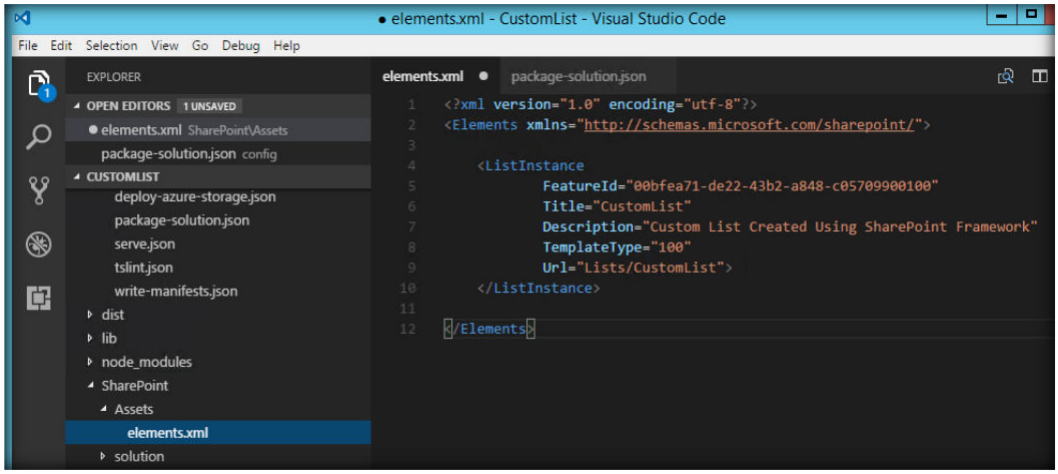


We will be creating an xml file - elements.xml which will hold the information required to provision the list. Let’s create the first supporting xml file elements.xml.



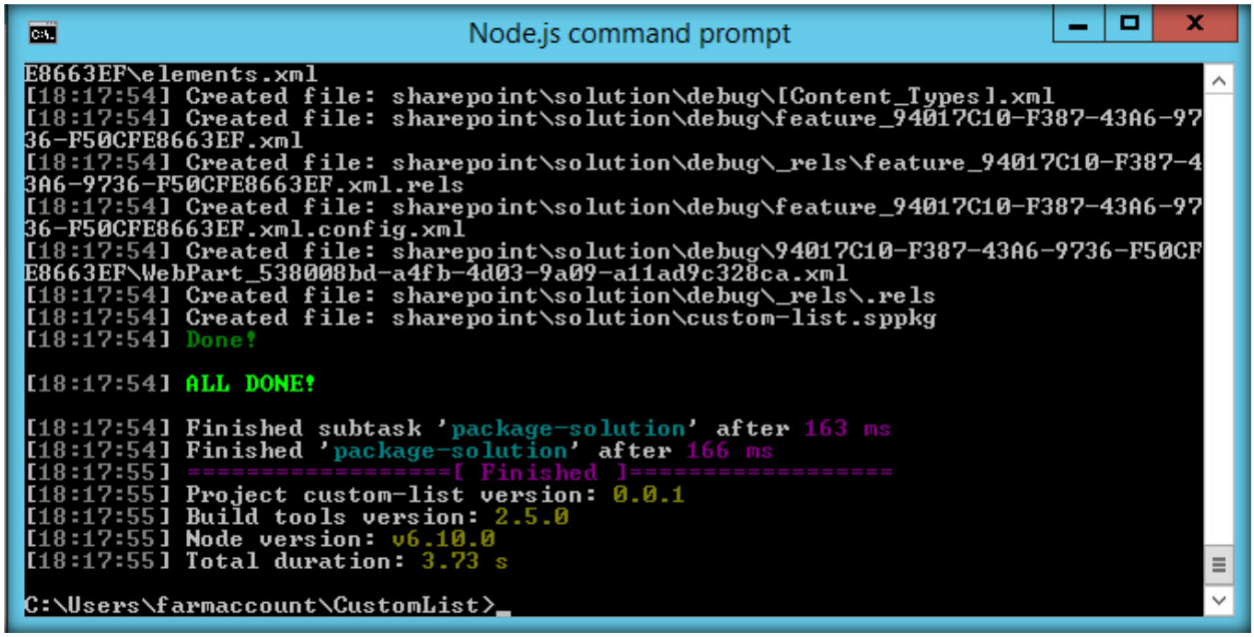
Add the below list information to the elements.xml file which contains the list name and type. The feature Id '00bfea71-de22-43b2-a848-c05709900100' refers to custom list.

```
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>
```

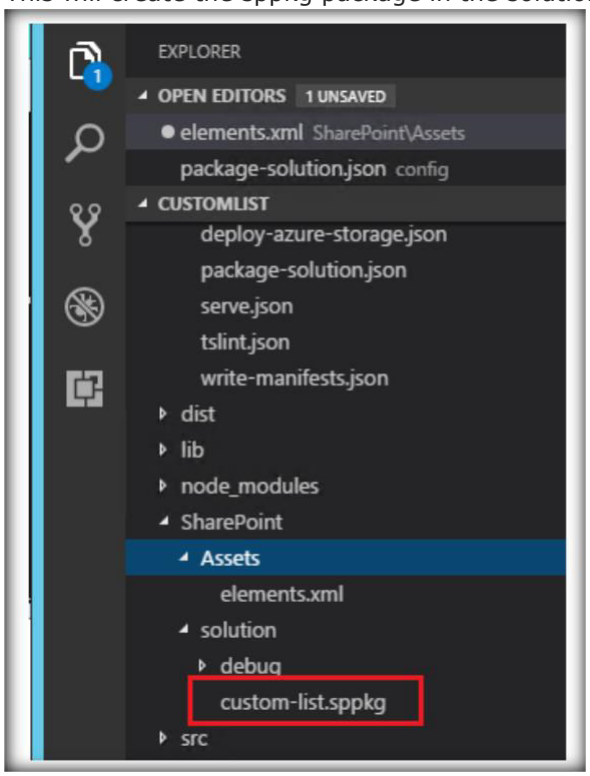


Package and Deploy the Solution

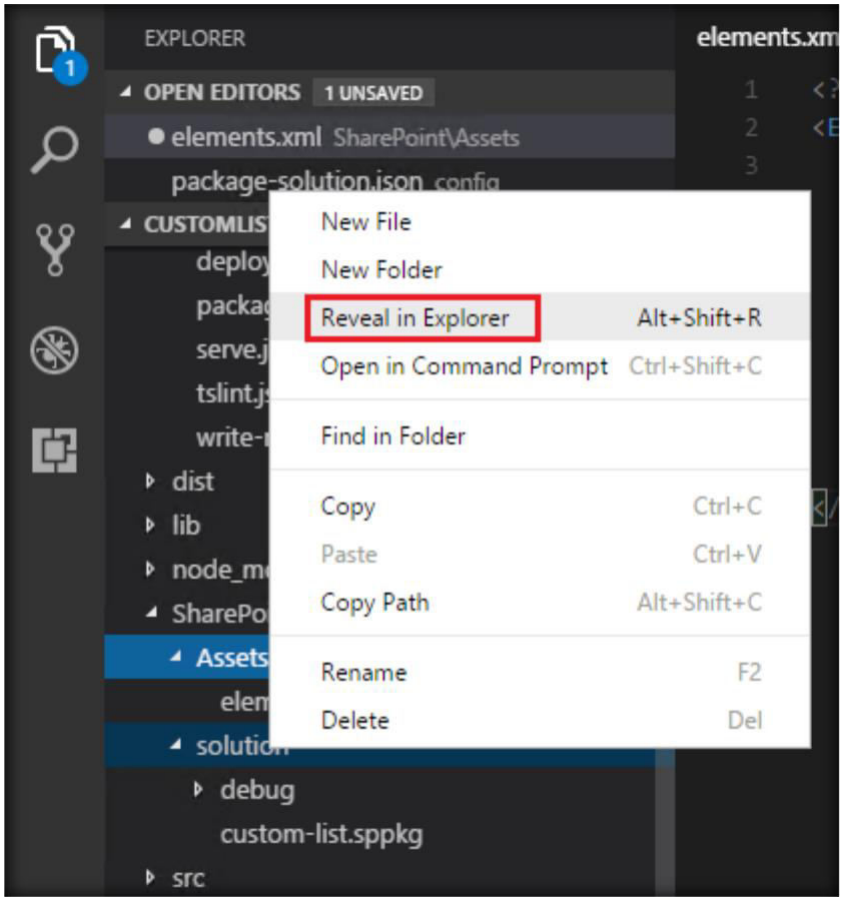
Now let's create the deployment package by running *gulp serve* command from the Node.js command prompt.



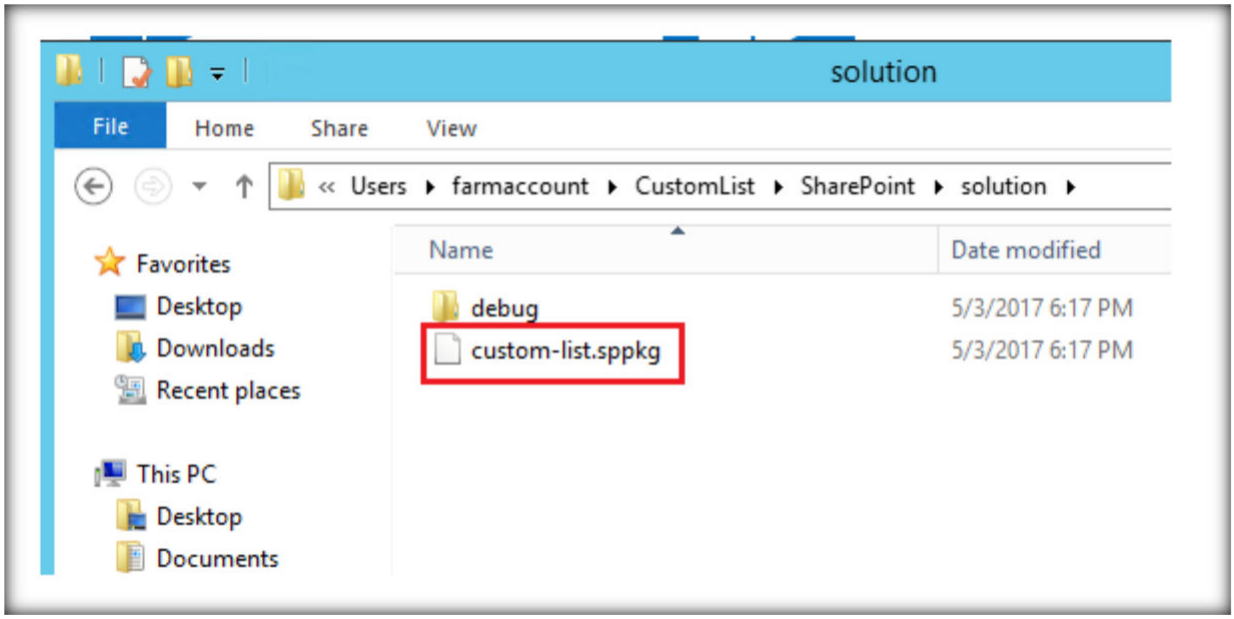
This will create the sppkg package in the solutions folder as shown below.



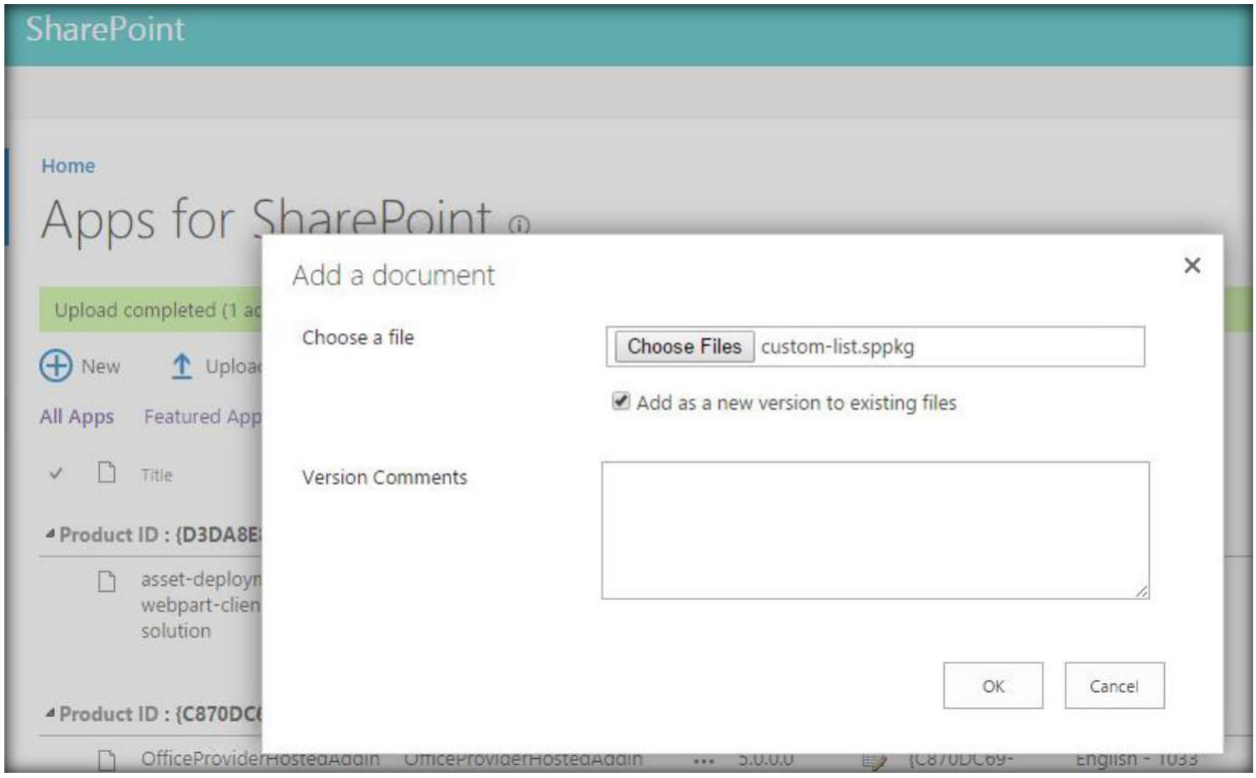
Take a note of the sppkg file url by opening it in File Explorer.



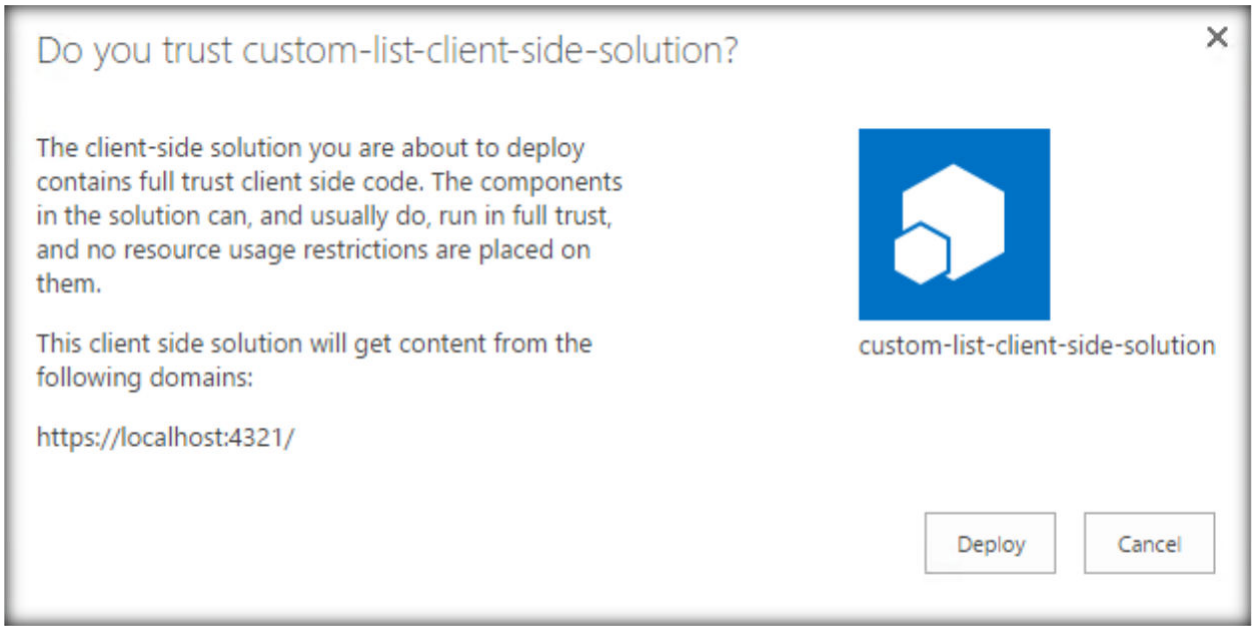
We will be uploading this package to the App Catalog in the next step.



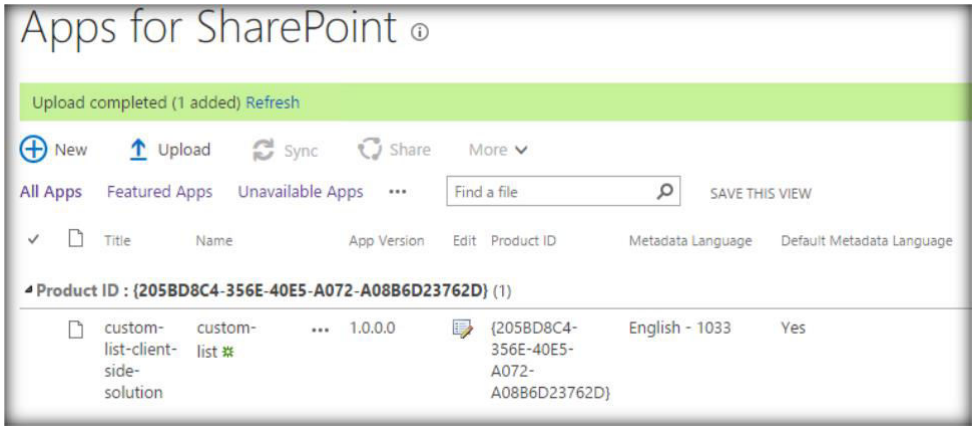
Head over to the App Catalog and upload the package.



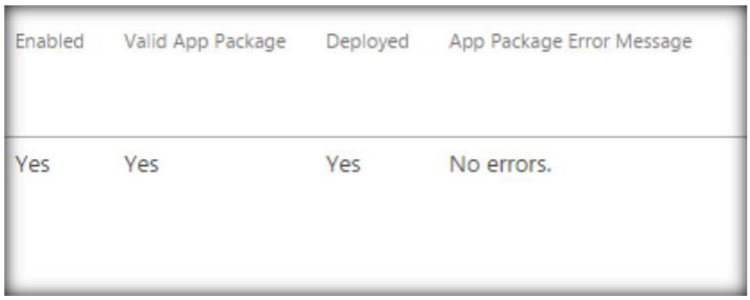
After upload it will ask if we trust the package. Click on Deploy to add the solution.



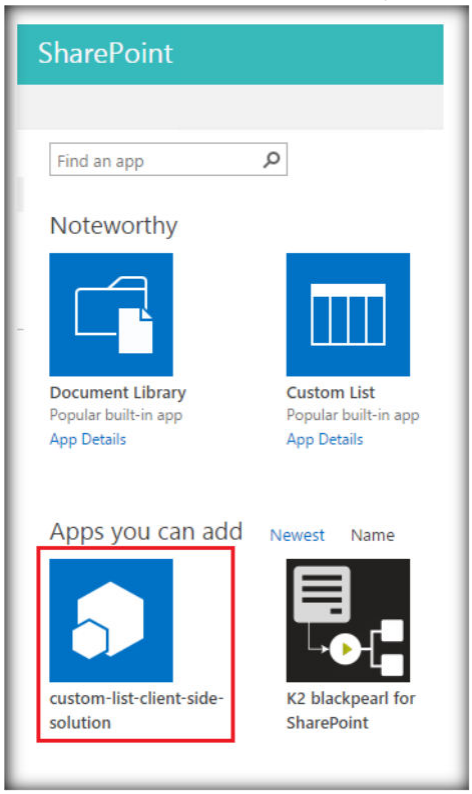
If we refresh the App Catalog page we can see the uploaded solution.



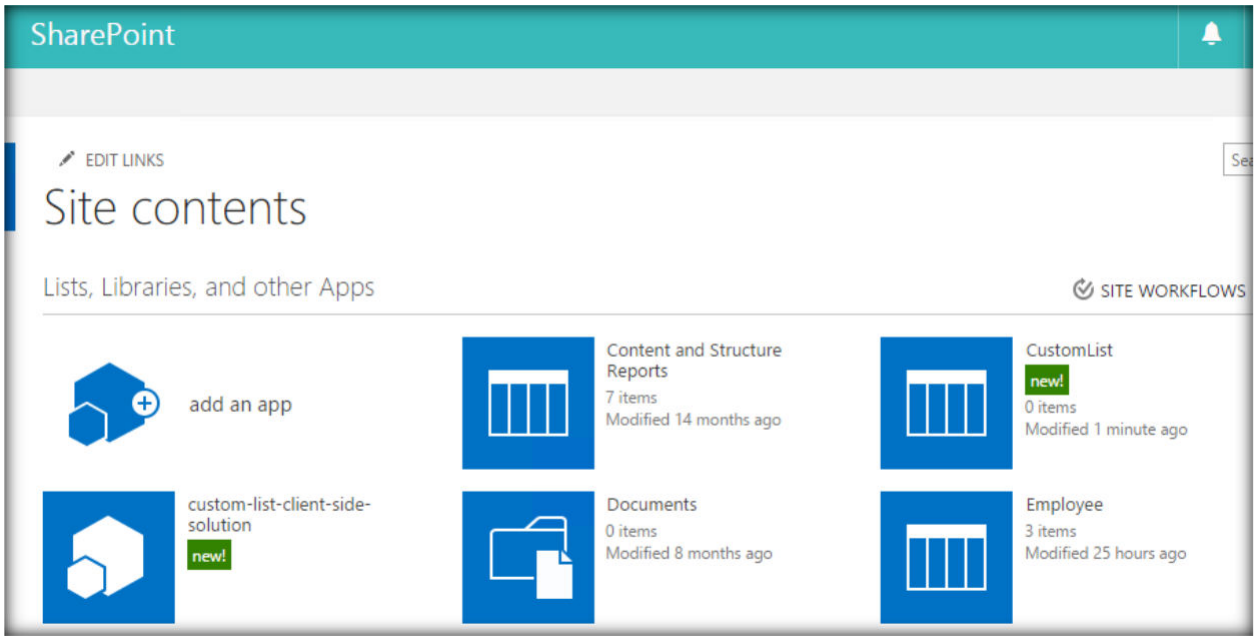
Ensure that there are no errors for the uploaded package by checking the below columns. In case there are some errors, we won't be able to add the solution to the site.



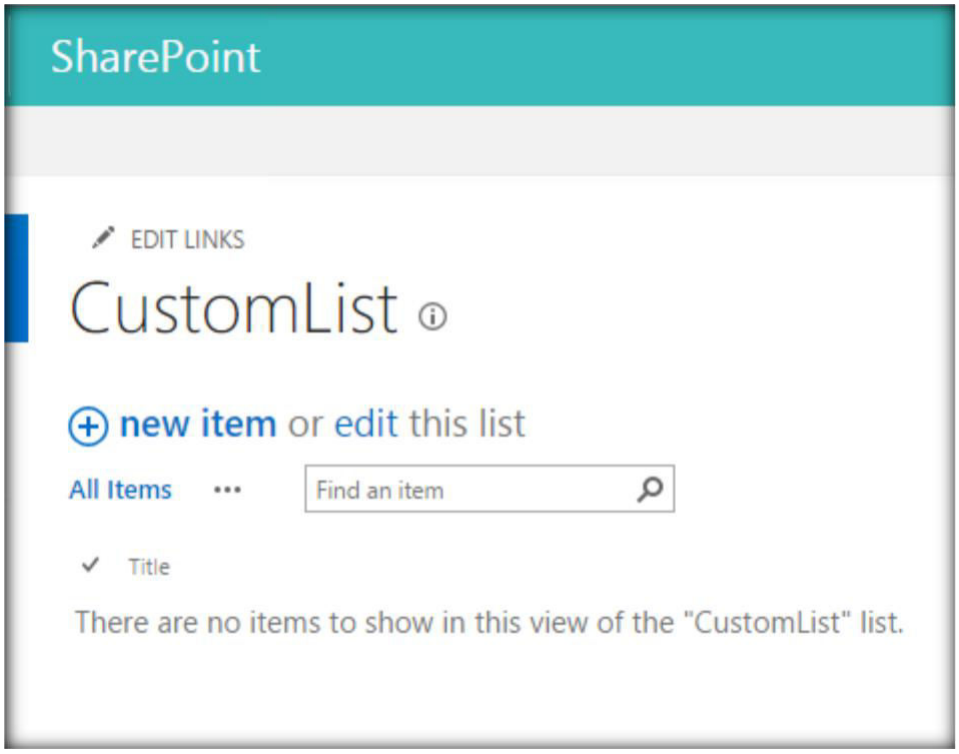
Now if we head over to the site, we can see the newly uploaded solution in the site contents.



Click on it to add the solution to the site.



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.

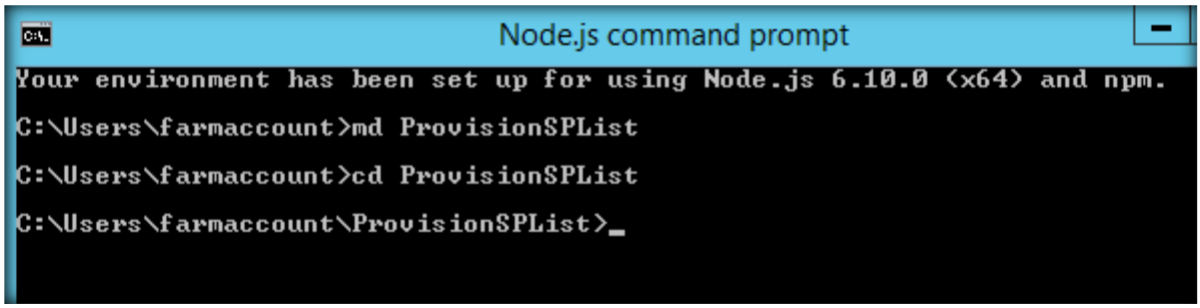


The main solution files used in this section are uploaded in Microsoft TechNet gallery. Feel free to download it.

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

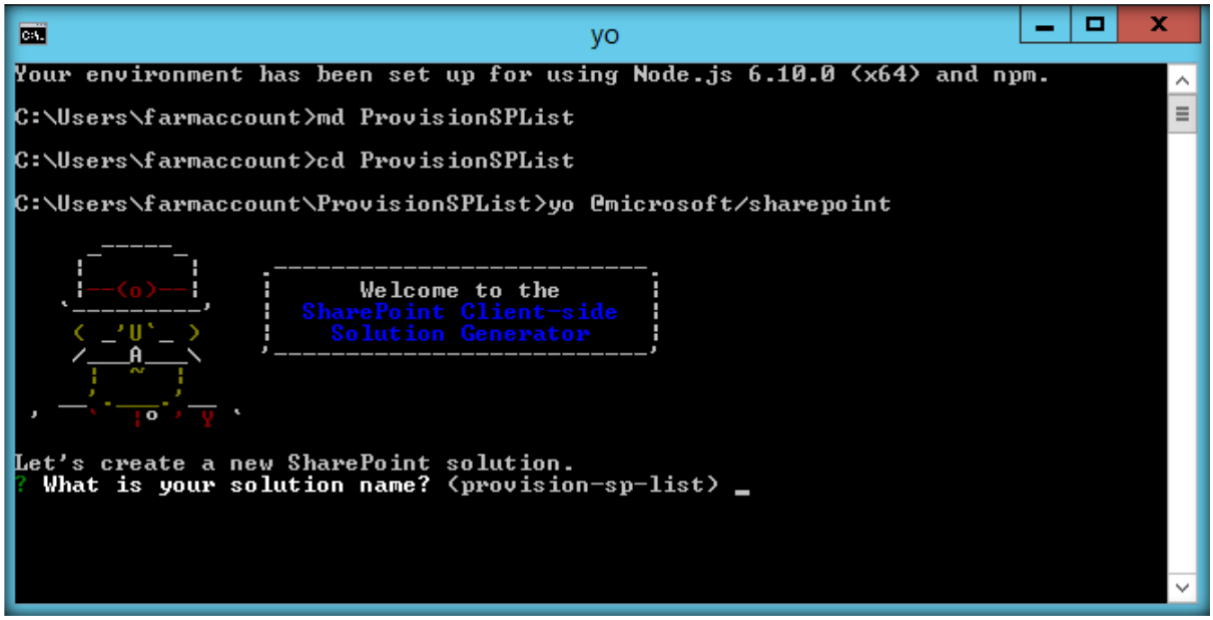


We will then create the client Web part by running the Yeoman SharePoint Generator. *yo @microsoft/sharepoint*

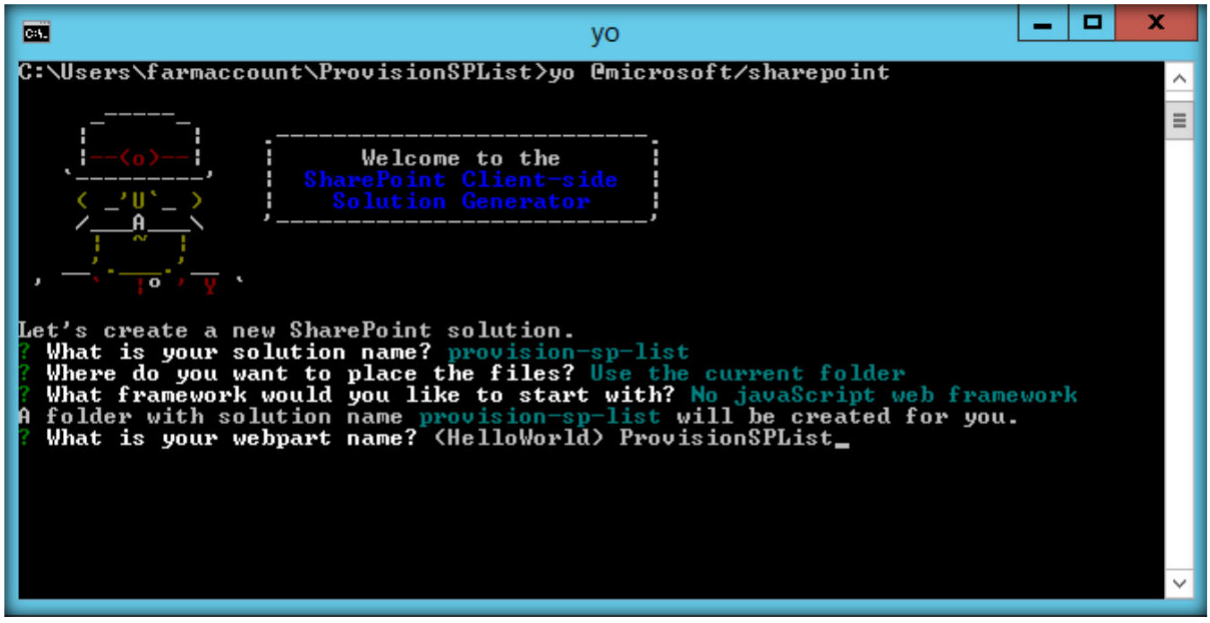
This will display the prompt, which we will have to fill up, so as to proceed with the project creation.

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

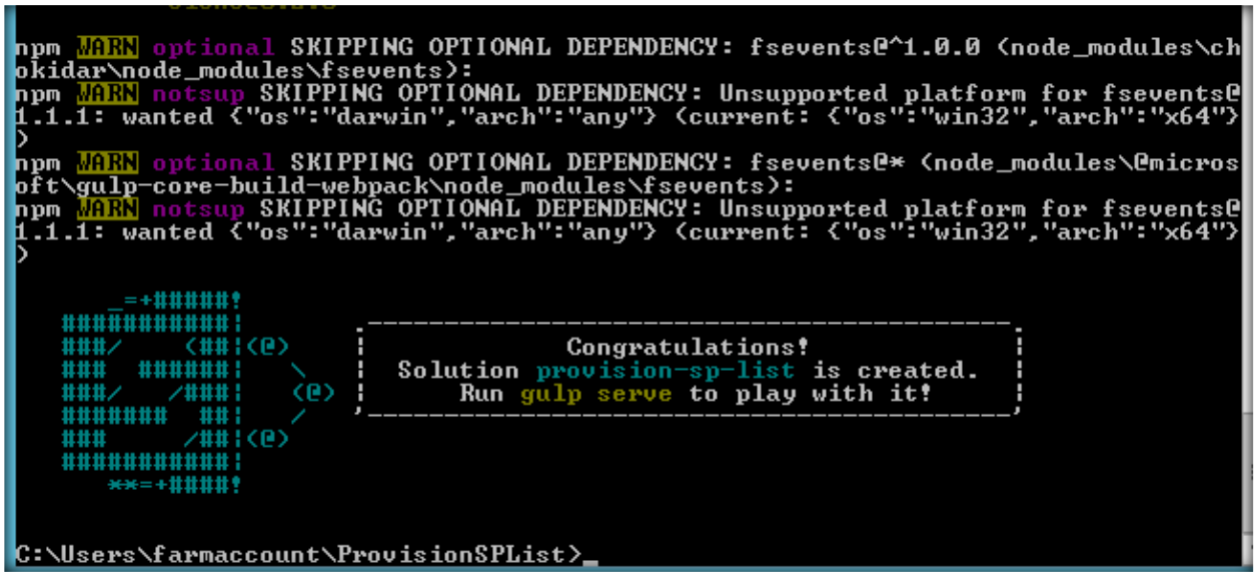
On pressing enter, we will be asked to chose the working folder for the project.



- 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



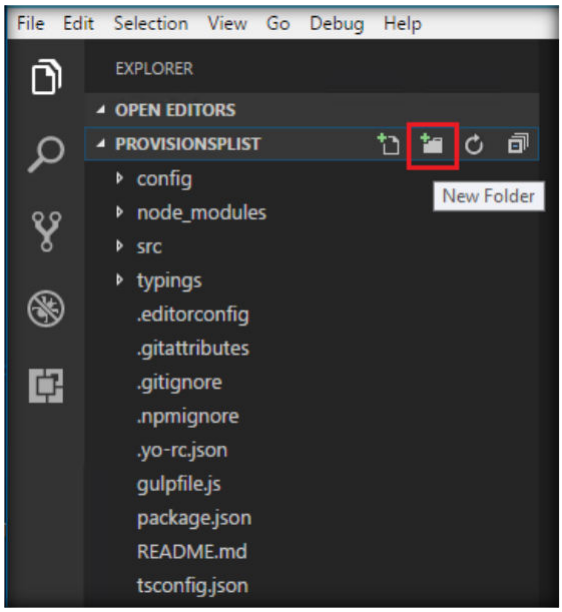
Yeoman has started working on the scaffolding of the project. It will install the required dependencies and scaffold the solution files for the 'ProvisionSPList' Web part, which will take some time to complete. Once completed, we will get a congratulations message.

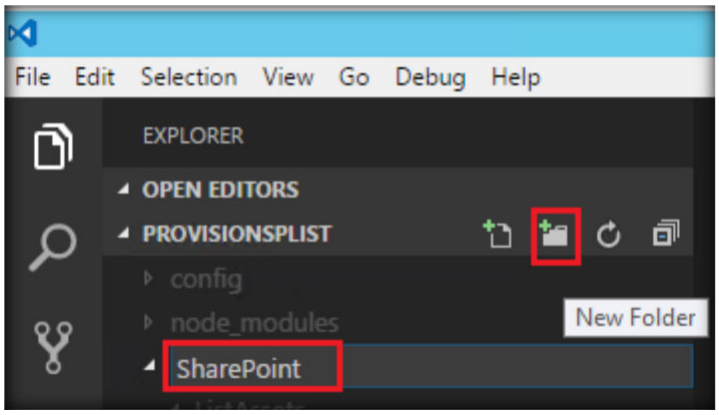


Edit the web part

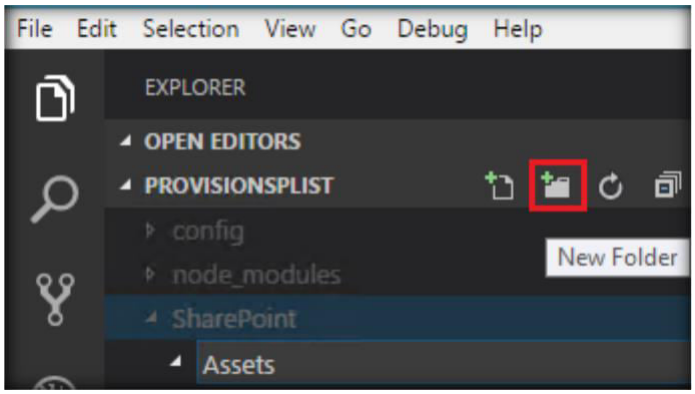
Run the Code . to scaffold and open the project in Visual Studio Code.

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

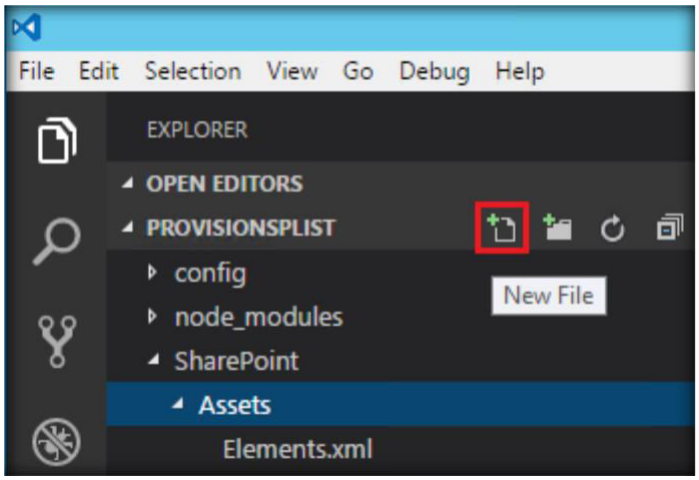




Within the SharePoint folder let's add another sub folder named Assets.



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>
```

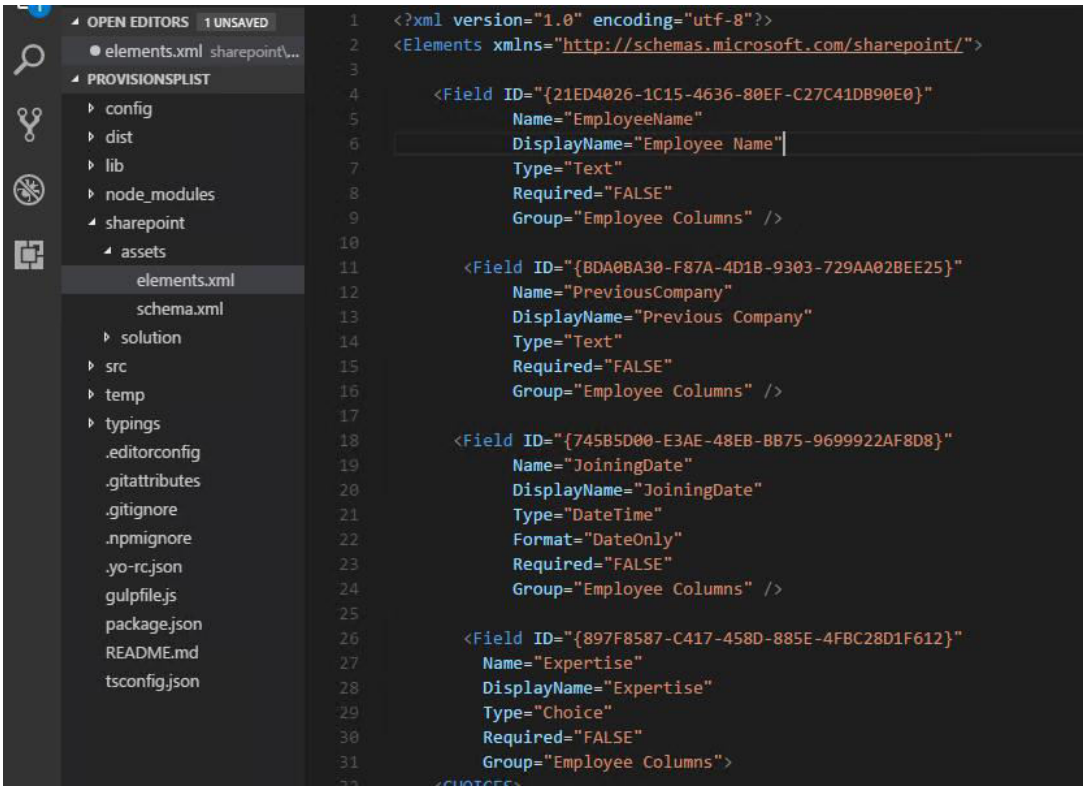
Elements.XML

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"
```

```
            Group="Employee" />

       <Field ID="{1DA0BA30-F87A-4D1B-9303-729AA02BEE25}"
            Name="PreviousCompany"
            DisplayName="Previous Company"
            Type="Text"
            Required="FALSE"
            Group="Employee" />

      <Field ID="{145B5D00-E3AE-48EB-BB75-9699922AF8D8}"
            Name="JoiningDate"
            DisplayName="JoiningDate"
            Type="DateTime"
            Format="DateOnly"
            Required="FALSE"
            Group="Employee" />

       <Field ID="{197F8587-C417-458D-885E-4FBC28D1F612}"
         Name="Expertise"
         DisplayName="Expertise"
         Type="Choice"
         Required="FALSE"
         Group="Employee">
    <CHOICES>
      <CHOICE>SharePoint</CHOICE>
      <CHOICE>Java</CHOICE>
      <CHOICE>.NET</CHOICE>
      <CHOICE>Python</CHOICE>
      <CHOICE>C++</CHOICE>
      <CHOICE>Web Designer</CHOICE>
    </CHOICES>
  </Field>

<Field ID="{10E72105-7577-4E9E-A758-BBBE8FF4E9BA}"
  Name="Experience"
  DisplayName="Experience"
  Group="Employee"
  Type="Number"
  Required="False"
  Min="0"
  Max="30"
  Percentage="FALSE">
</Field>

    <ContentType ID="0x010100FA0963FA69A646AA916D2E41284FC9D1"
            Name="EmployeeContentType"
            Group="Employee Content Types"
            Description="This is the Content Type for Employee Onboarding">
        <FieldRefs>
            <FieldRef ID="{11ED4026-1C15-4636-80EF-C27C41DB90E0}" />
            <FieldRef ID="{1DA0BA30-F87A-4D1B-9303-729AA02BEE25}" />
            <FieldRef ID="{145B5D00-E3AE-48EB-BB75-9699922AF8D8}" />
            <FieldRef ID="{197F8587-C417-458D-885E-4FBC28D1F612}" />
            <FieldRef ID="{10E72105-7577-4E9E-A758-BBBE8FF4E9BA}" />
        </FieldRefs>
    </ContentType>

    <ListInstance
            CustomSchema="schema.xml"
            FeatureId="00bfea71-de22-43b2-a848-c05709900100"
            Title="Employee"
```

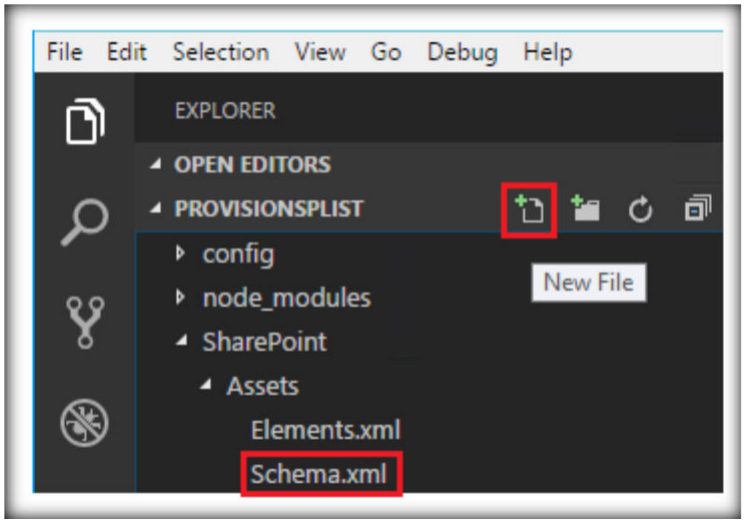
```
            Description="Employee List created using SharePoint Framework"
            TemplateType="100"
            Url="Lists/Employee">
      <Data>
       <Rows>
         <Row>
           <Field Name="EmployeeName">Priyaranjan</Field>
           <Field Name="PreviousCompany">Cognizant</Field>
           <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>
```



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

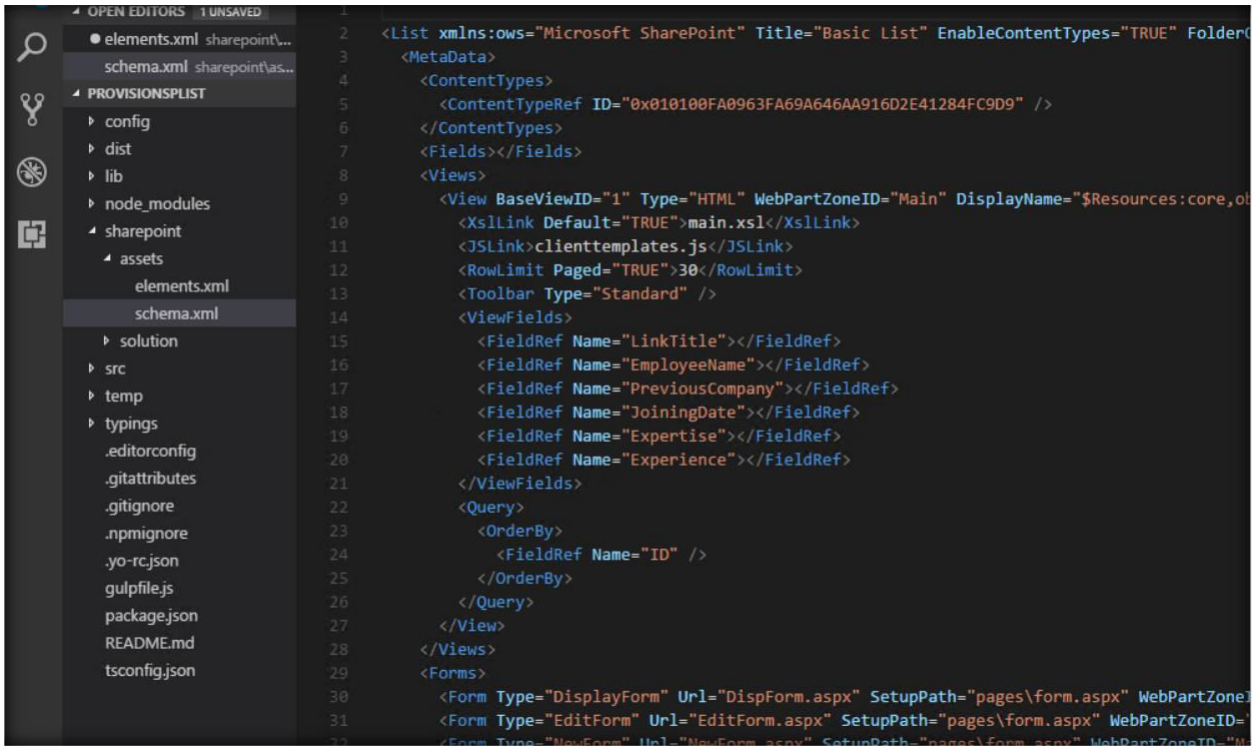
```
1.  <ContentTypes>
2.        <ContentTypeRef ID="0x010100FA0963FA69A646AA916D2E41284FC9D9" />
3.    </ContentTypes>
```



The complete schema.xml will look like below:

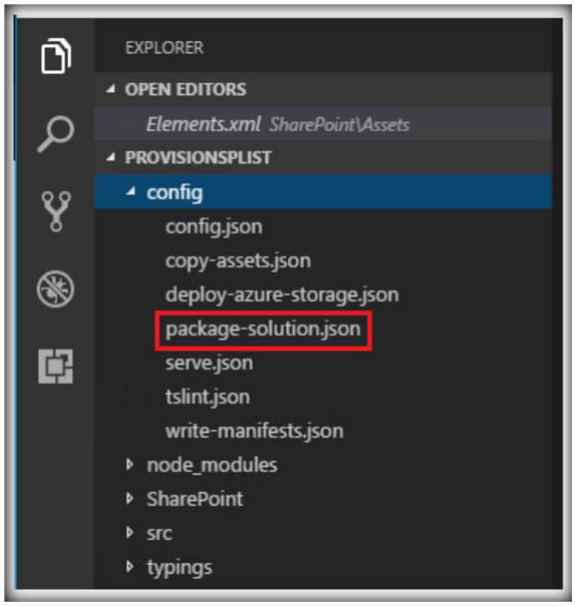
```
1.  <List xmlns:ows="Microsoft SharePoint" Title="Basic List" EnableContentTypes="TRUE"
    FolderCreation="FALSE" Direction="$Resources:Direction;" Url="Lists/Basic List"
    BaseType="0" xmlns="http://schemas.microsoft.com/sharepoint/">
2.    <MetaData>
3.      <ContentTypes>
4.        <ContentTypeRef 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>
```

```
          <OrderBy>
            <FieldRef Name="ID" />
          </OrderBy>
        </Query>
      </View>
    </Views>
    <Forms>
<Form Type="DisplayForm" Url="DispForm.aspx" SetupPath="pages\form.aspx" WebPartZoneID="Main" />
<Form Type="EditForm" Url="EditForm.aspx" SetupPath="pages\form.aspx" WebPartZoneID="Main" />
<Form Type="NewForm" Url="NewForm.aspx" SetupPath="pages\form.aspx" WebPartZoneID="Main" />
    </Forms>
  </MetaData>
</List>
```

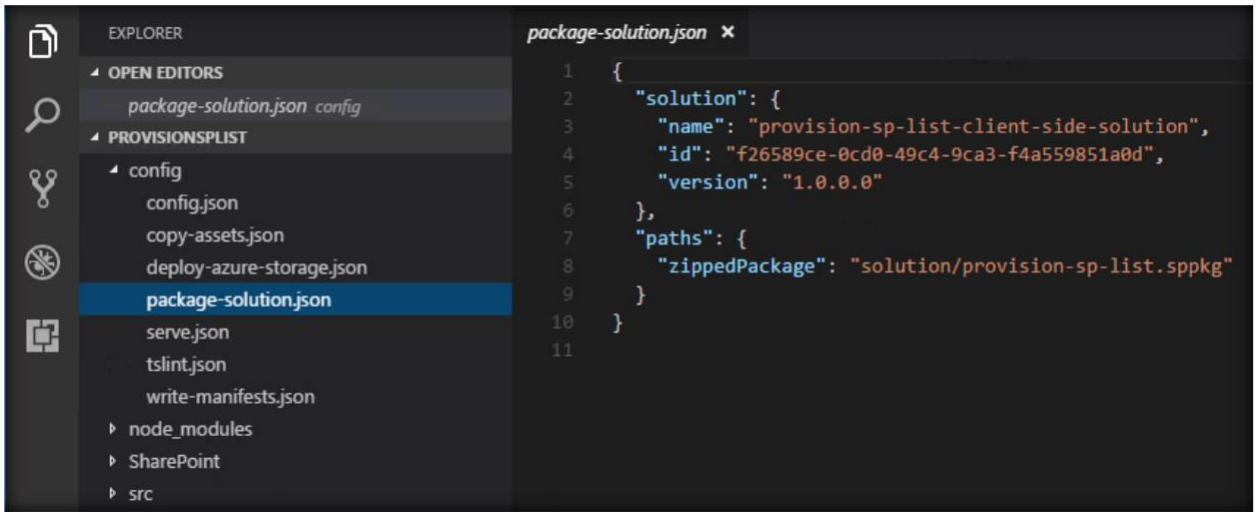


Before we can deploy the package we have to update the feature in the package-solution.json file

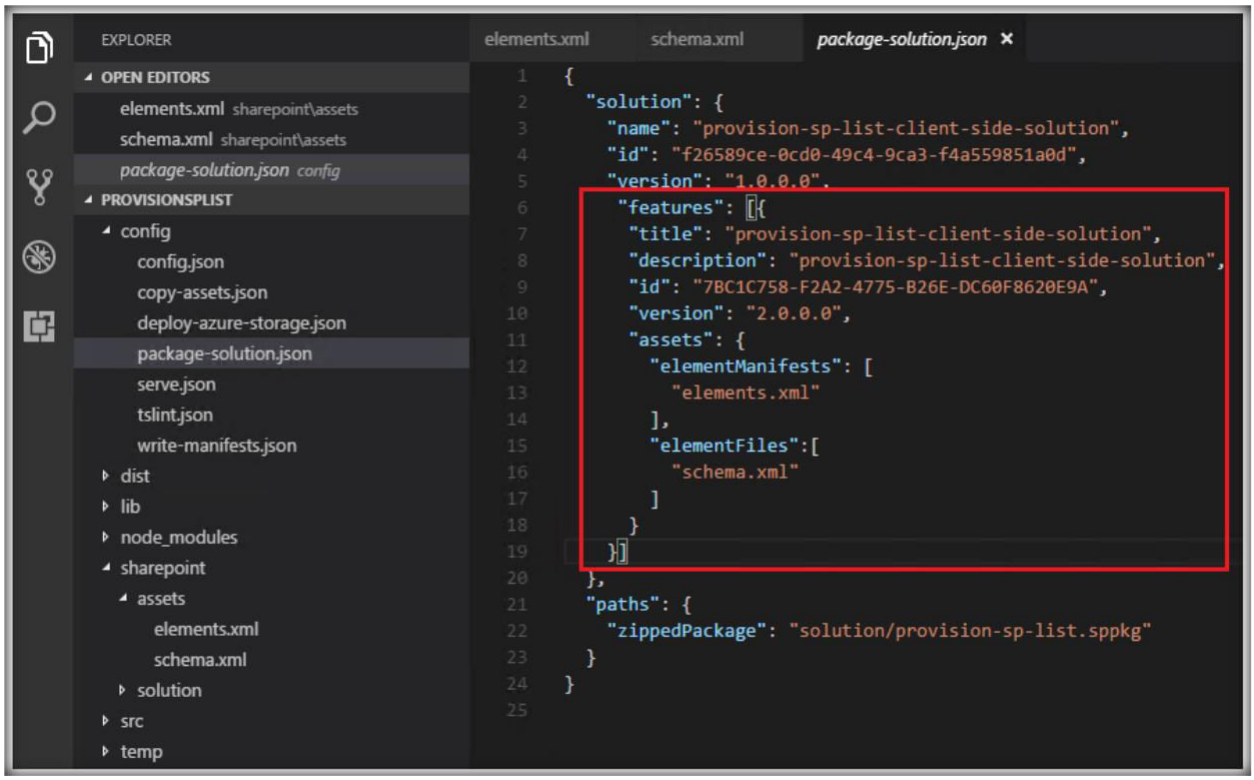
Update Package-Solution.json



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



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



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

```
{
  "solution": {
    "name": "provision-sp-list-client-side-solution",
    "id": "f26589ce-0cd0-49c4-9ca3-f4a559851a0d",
    "version": "1.0.0.0",
     "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"
        ],
        "elementFiles":[
          "schema.xml"
        ]
      }
    }]
  },
  "paths": {
    "zippedPackage": "solution/provision-sp-list.sppkg"
  }
}
```

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>
```

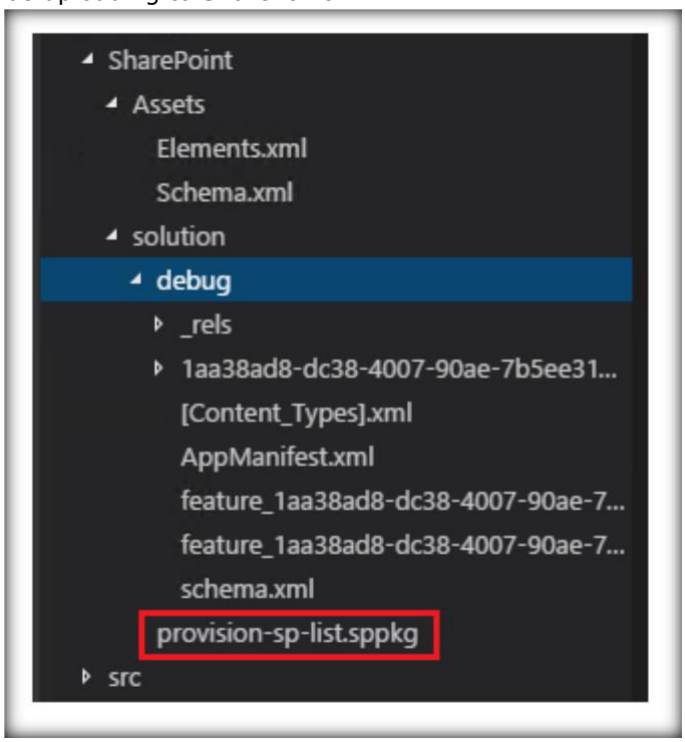
gulp package-solution

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

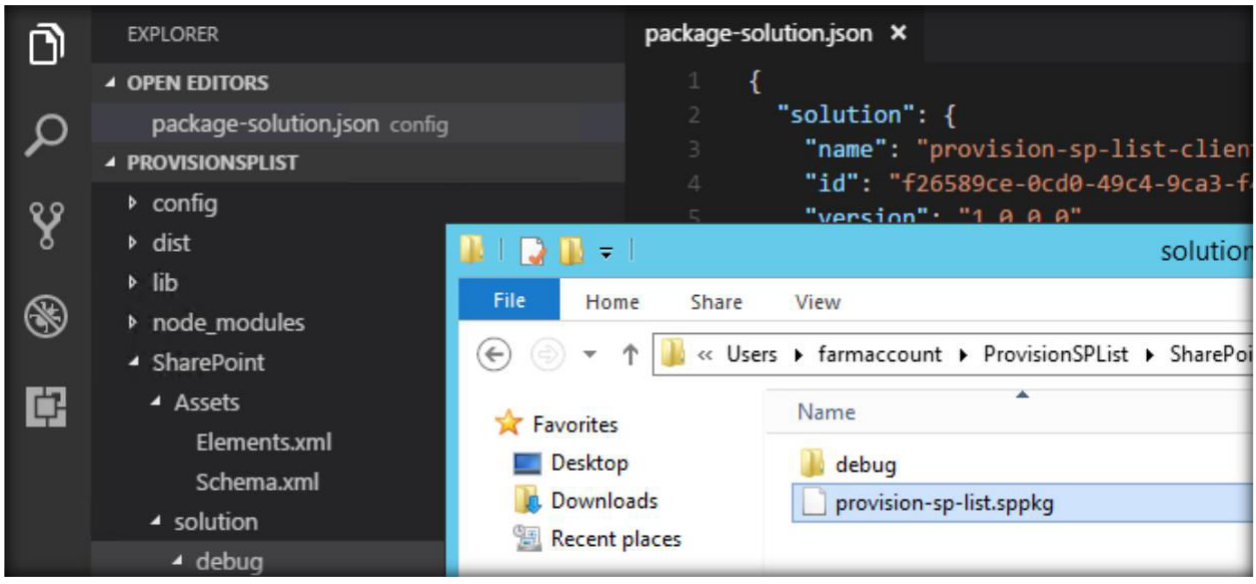
Thus, we are done with the packaging of the solution.

```
[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_e1cca05f-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>
```

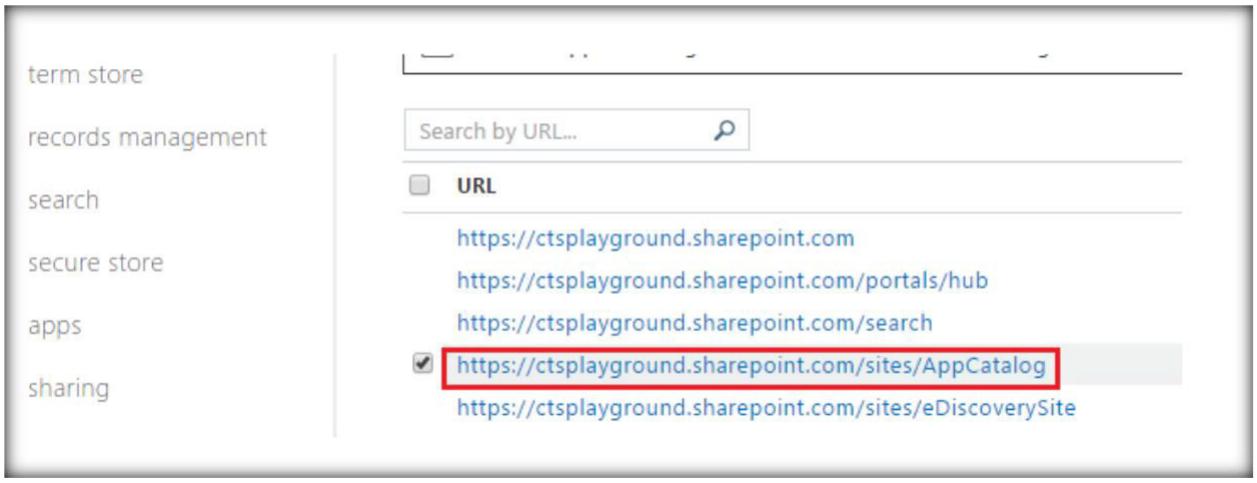
If we head over to the solutions folder, we can see the ‘provision-sp-list package’ which we will be uploading to SharePoint.



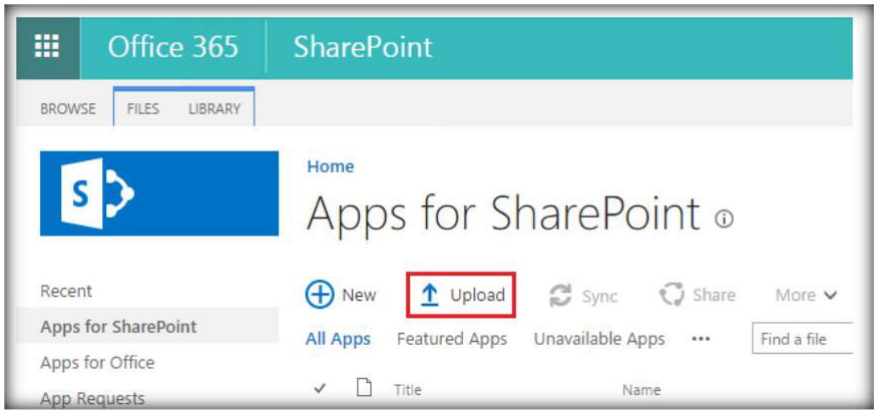
Make a note of the solution URL in the local computer as we will need it to upload to SharePoint.



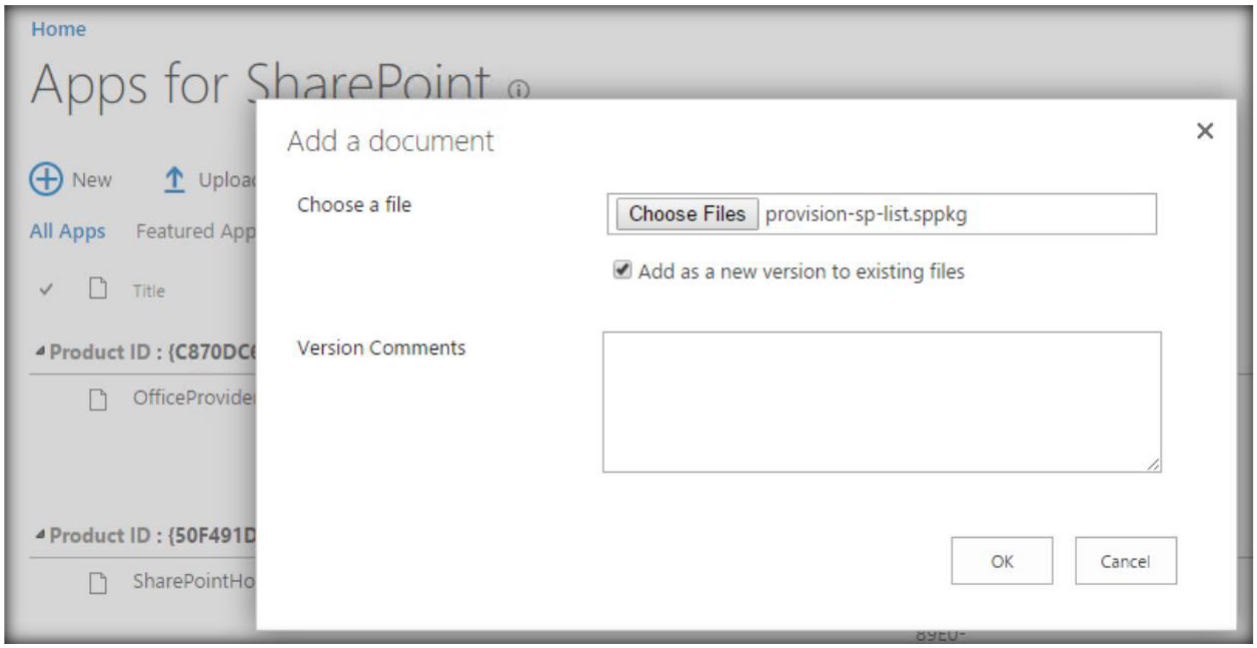
Let’s head over to the SharePoint App Catalog site to where we will be uploading the solution.



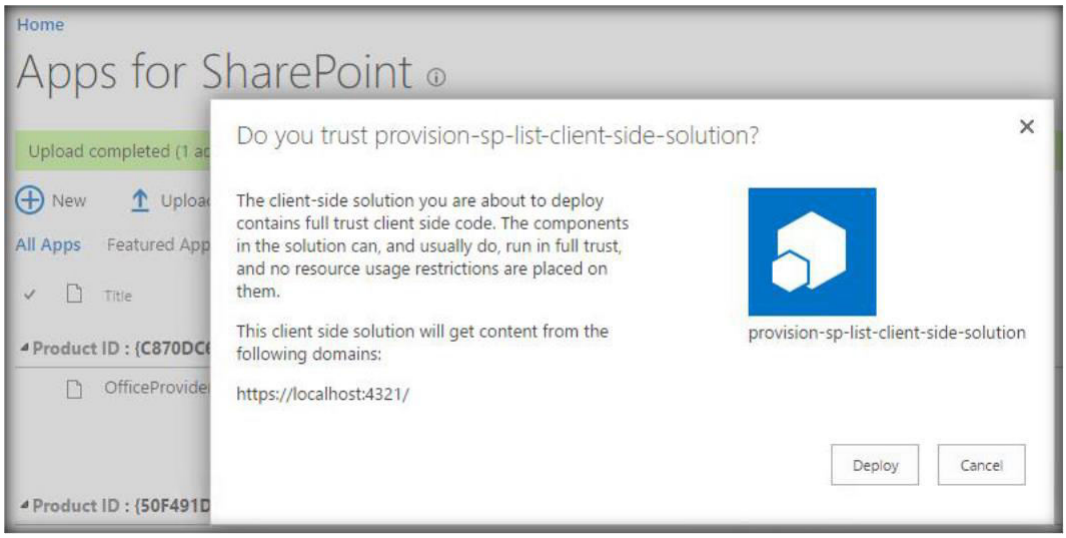
Click on Upload to add the solution file to the site.



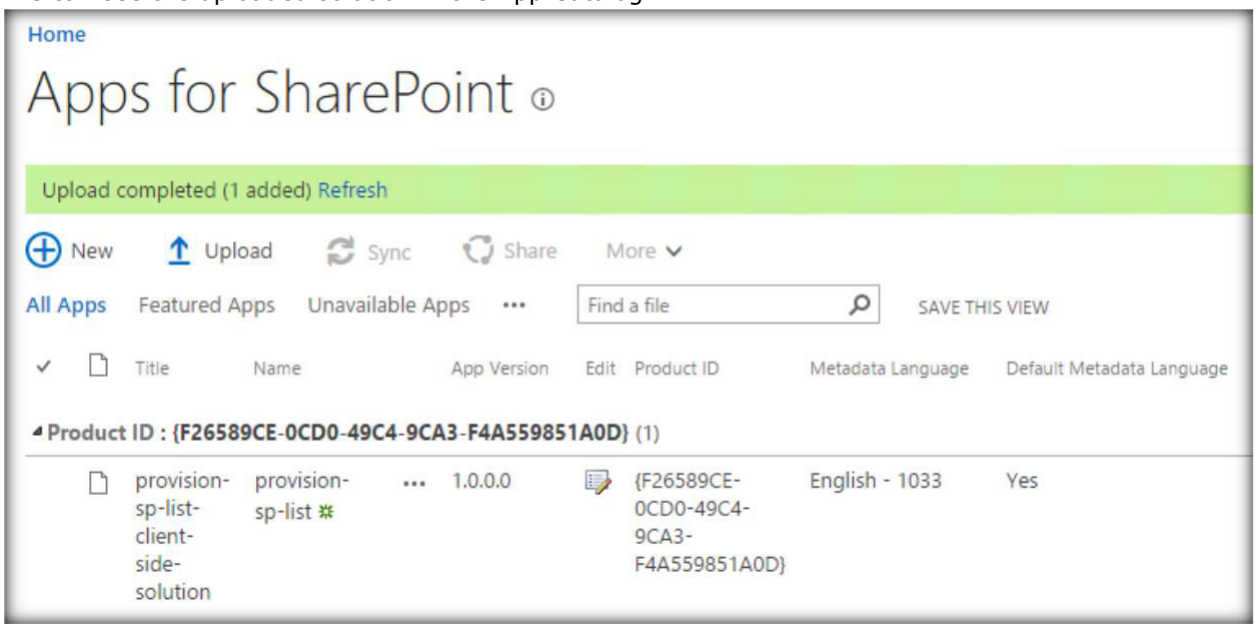
Click on OK to complete the upload.



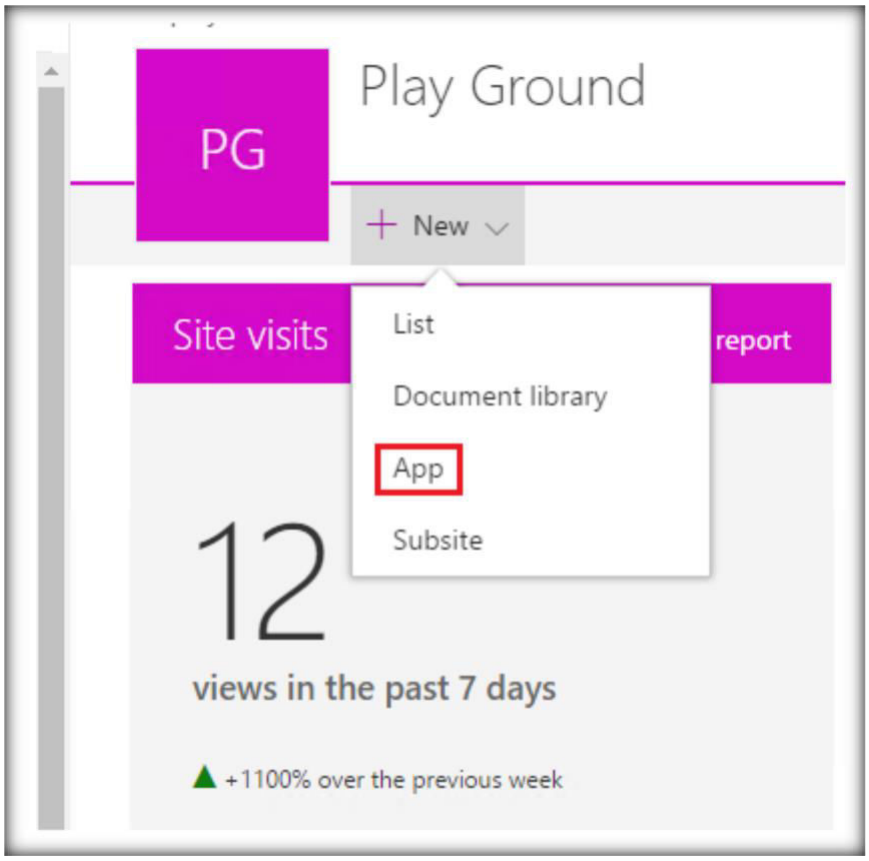
It will ask to trust and Deploy the solution to SharePoint.



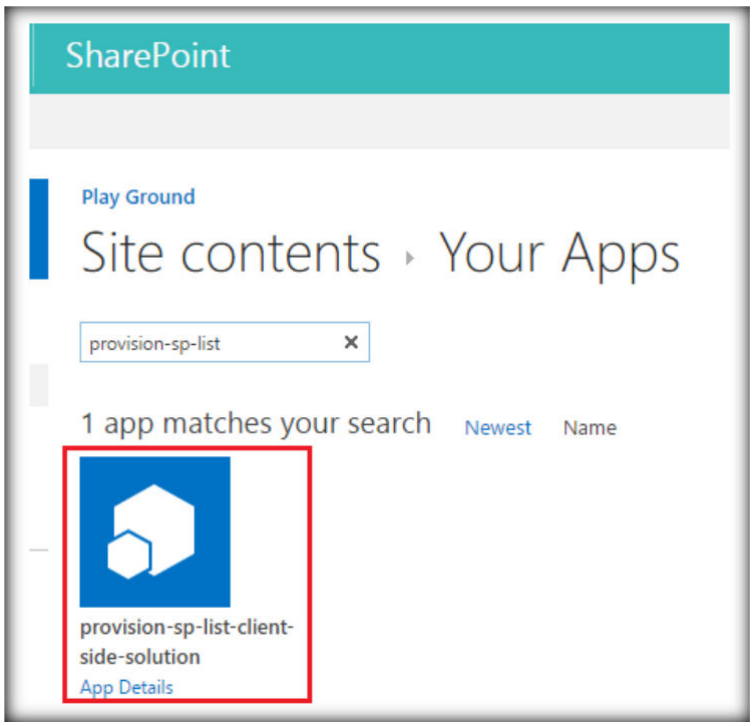
We can see the uploaded solution in the App Catalog.



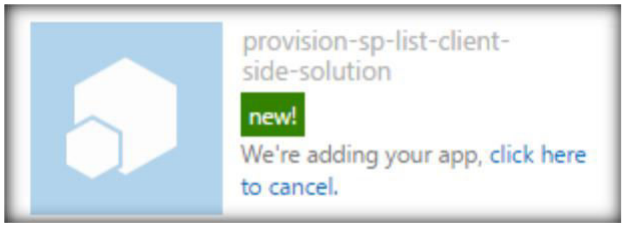
Now let's head over to the site contents and add the solution to the site.



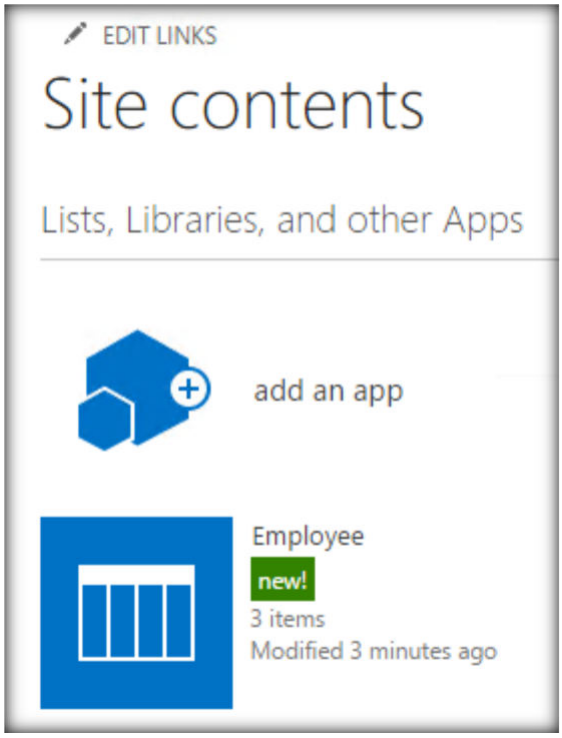
On searching for the deployed app, it will list out the recently added solution.



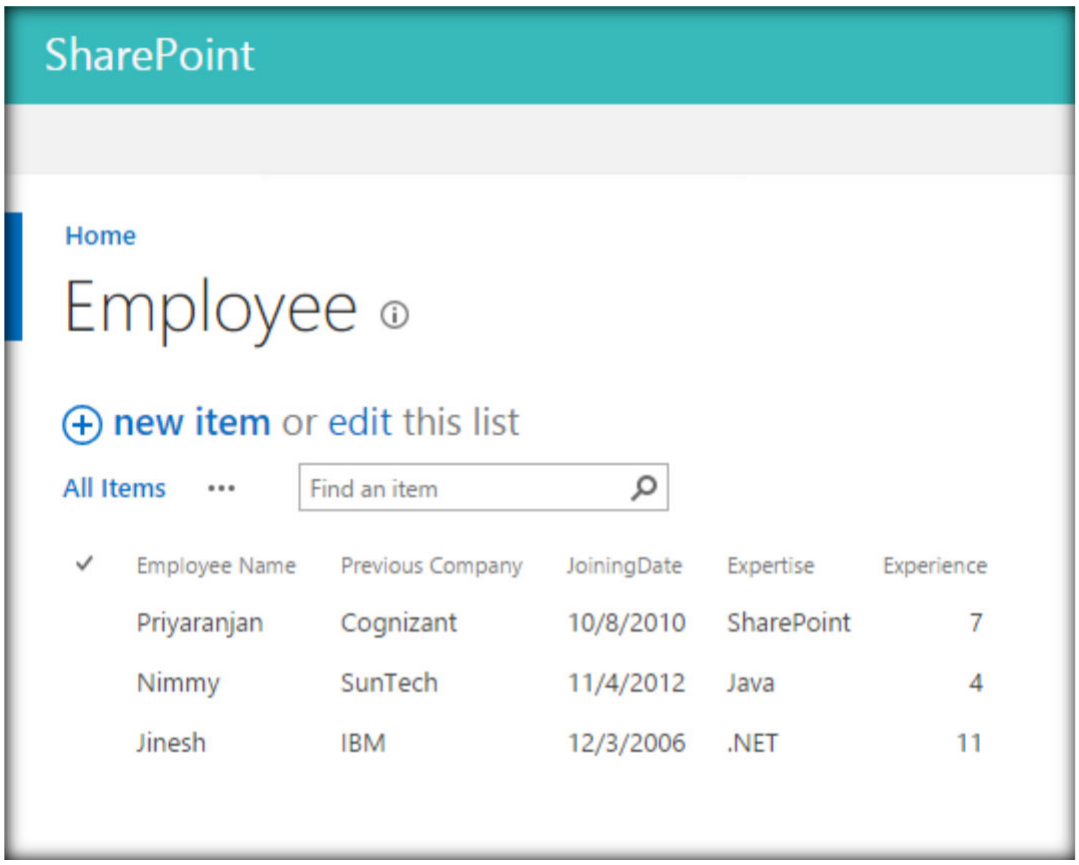
Click on it to add the solution to the site.



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



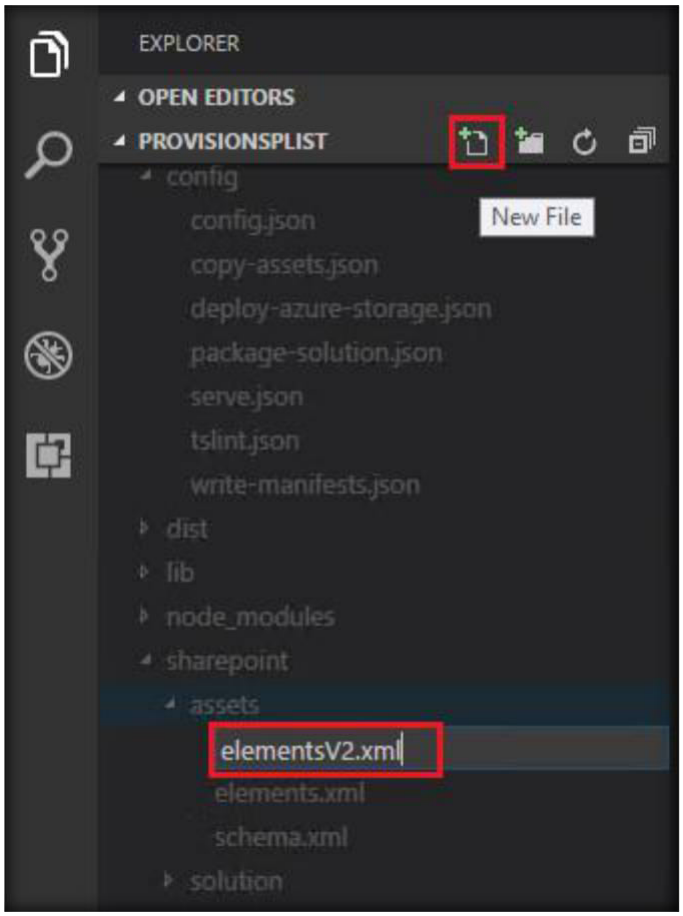
Going inside it we can see the default data that we had added to the list.



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

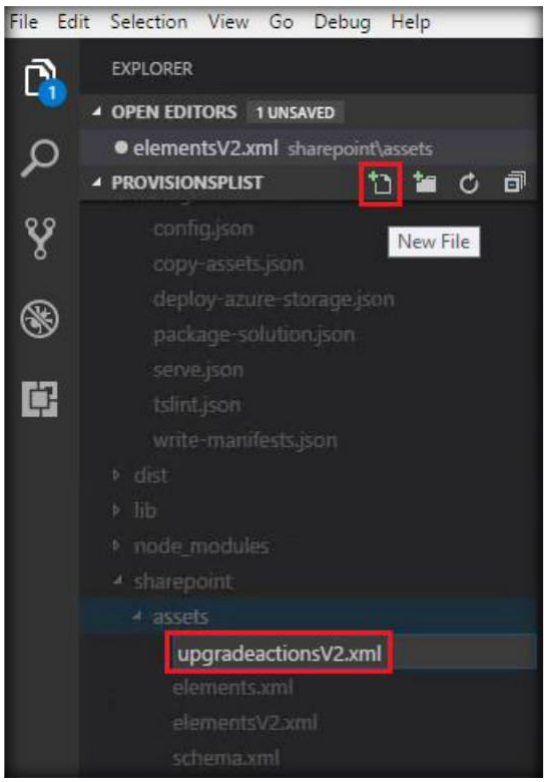
In our case, we are trying to add a new list along with the previously deployed solution. In the new elements file, we will specify the list instance declaration for a custom list.

```
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

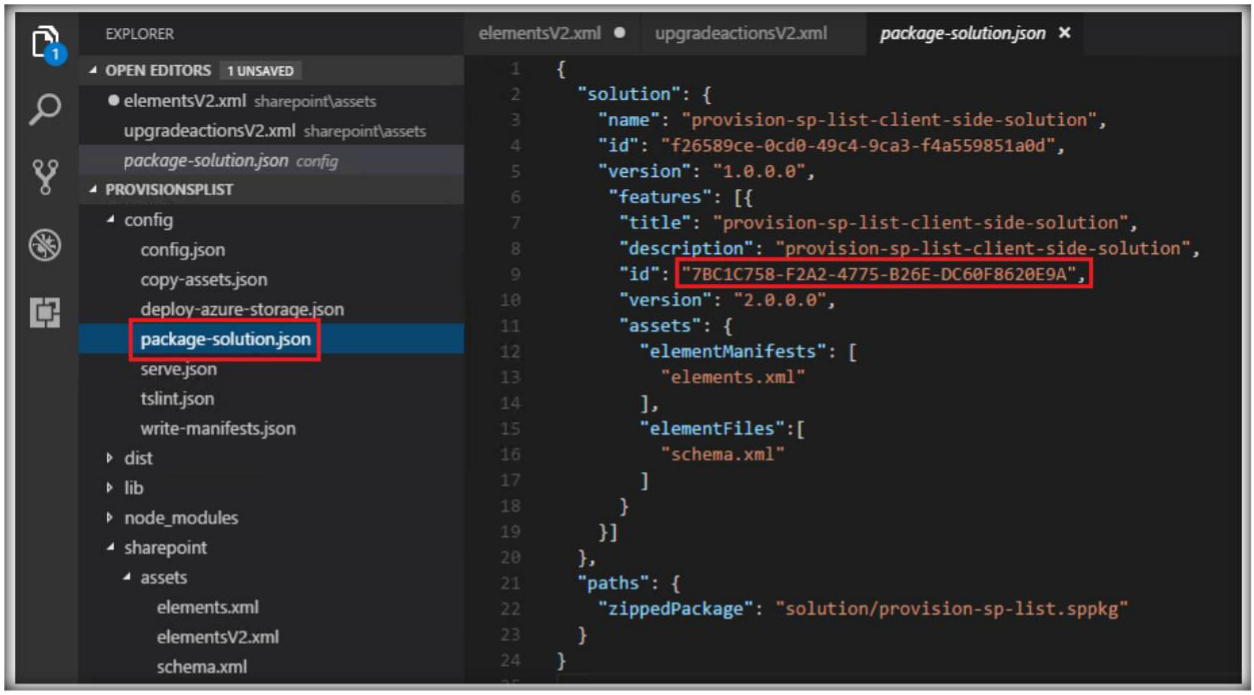
```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

    <ListInstance
            FeatureId="00bfea71-de22-43b2-a848-c05709900100"
            Title="ListAddedViaUpgrade"
            Description="List added through Upgrade Action"
            TemplateType="100"
            Url="Lists/ListAddedViaUpgrade">
    </ListInstance>

</Elements>
```

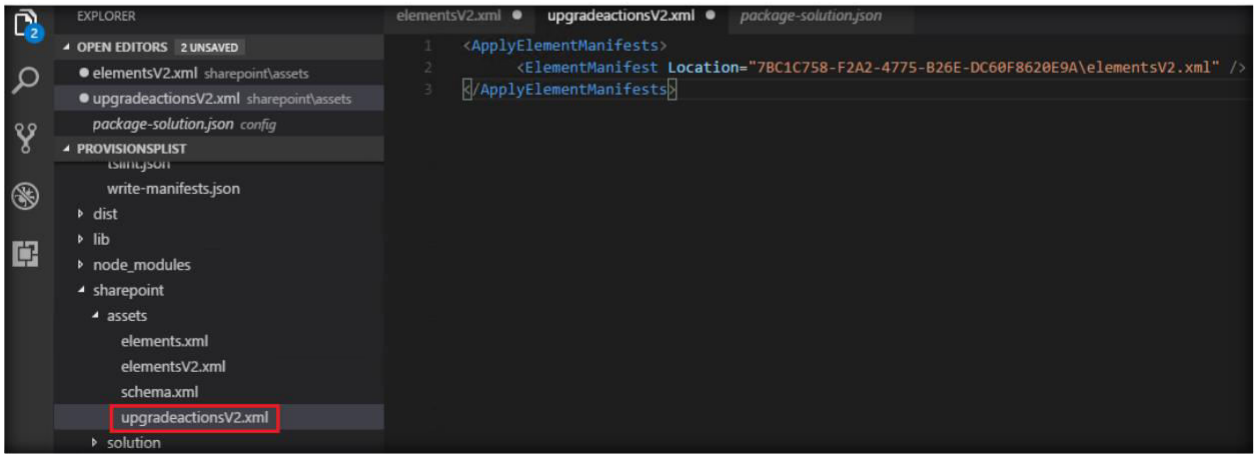
We will also add an upgrade action which will contain the information about the newly added elements.xml file.

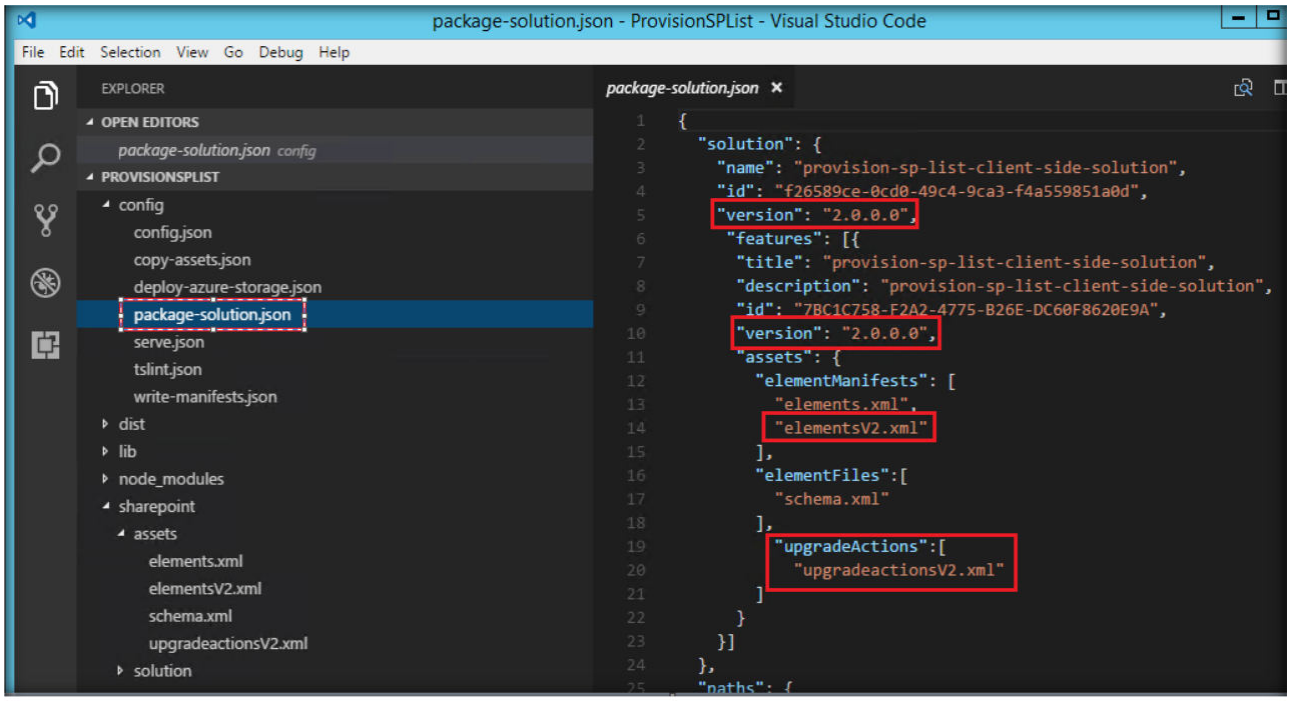


```
1.  <ApplyElementManifests>
2.        <ElementManifest Location="7BC1C758-F2A2-4775-B26E-DC60F8620E9A\elementsV2.xml"
    />
3.  </ApplyElementManifests>
```

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



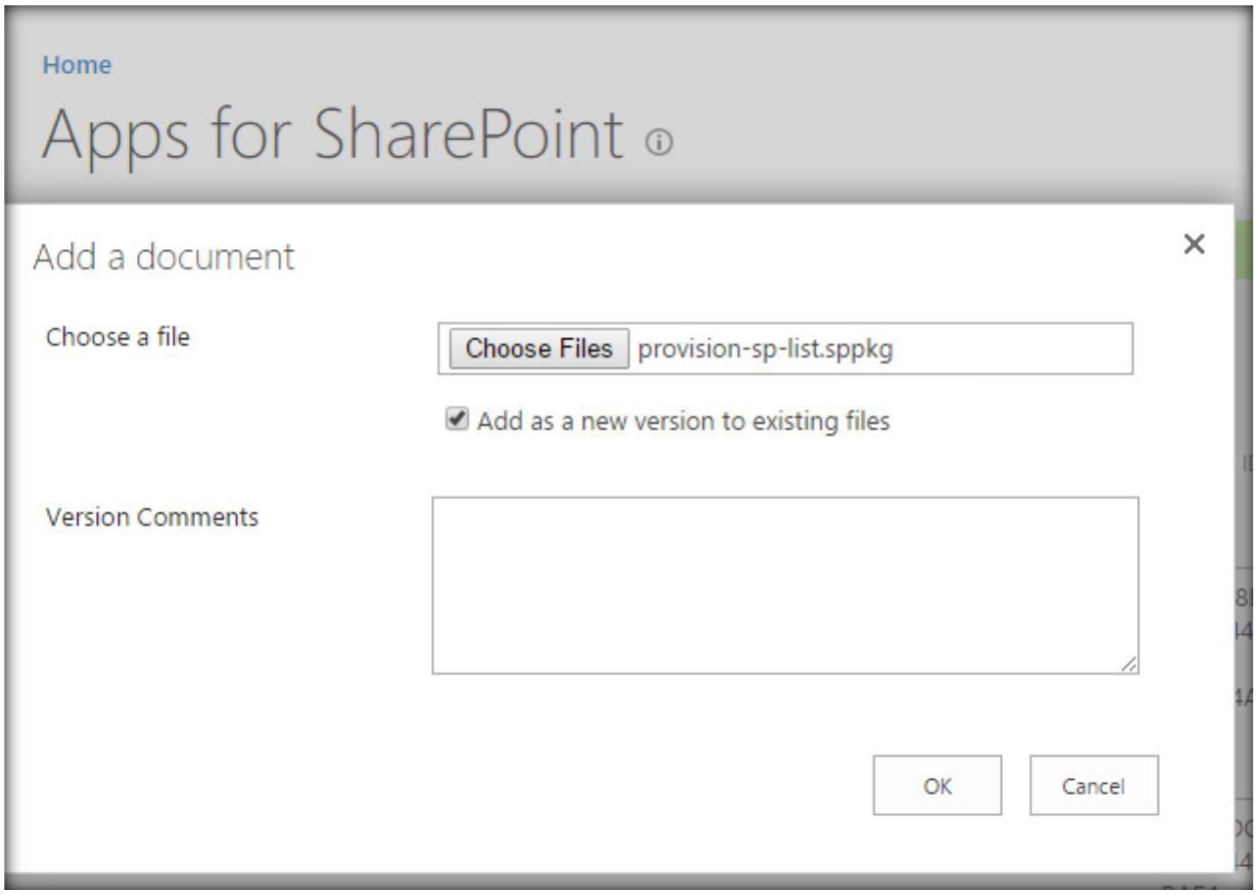




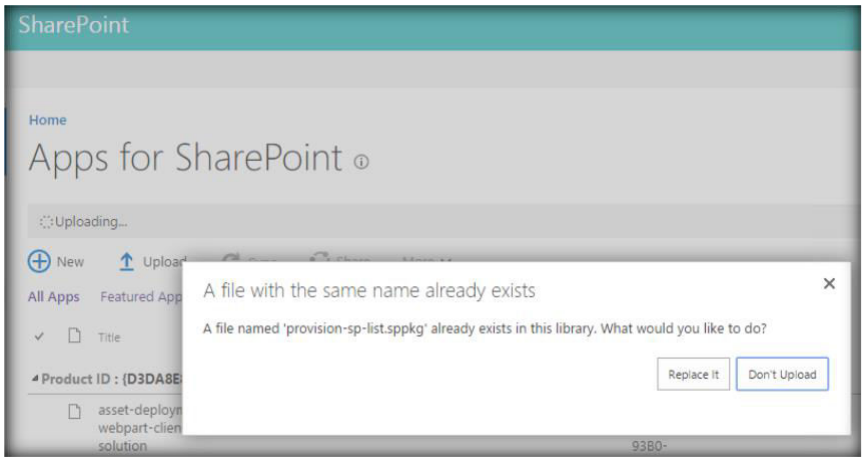

```
1.  {
2.    "solution": {
3.      "name": "provision-sp-list-client-side-solution",
4.      "id": "f26589ce-0cd0-49c4-9ca3-f4a559851a0d",
5.      "version": "2.0.0.0",
6.       "features": [{
7.        "title": "provision-sp-list-client-side-solution",
8.        "description": "provision-sp-list-client-side-solution",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

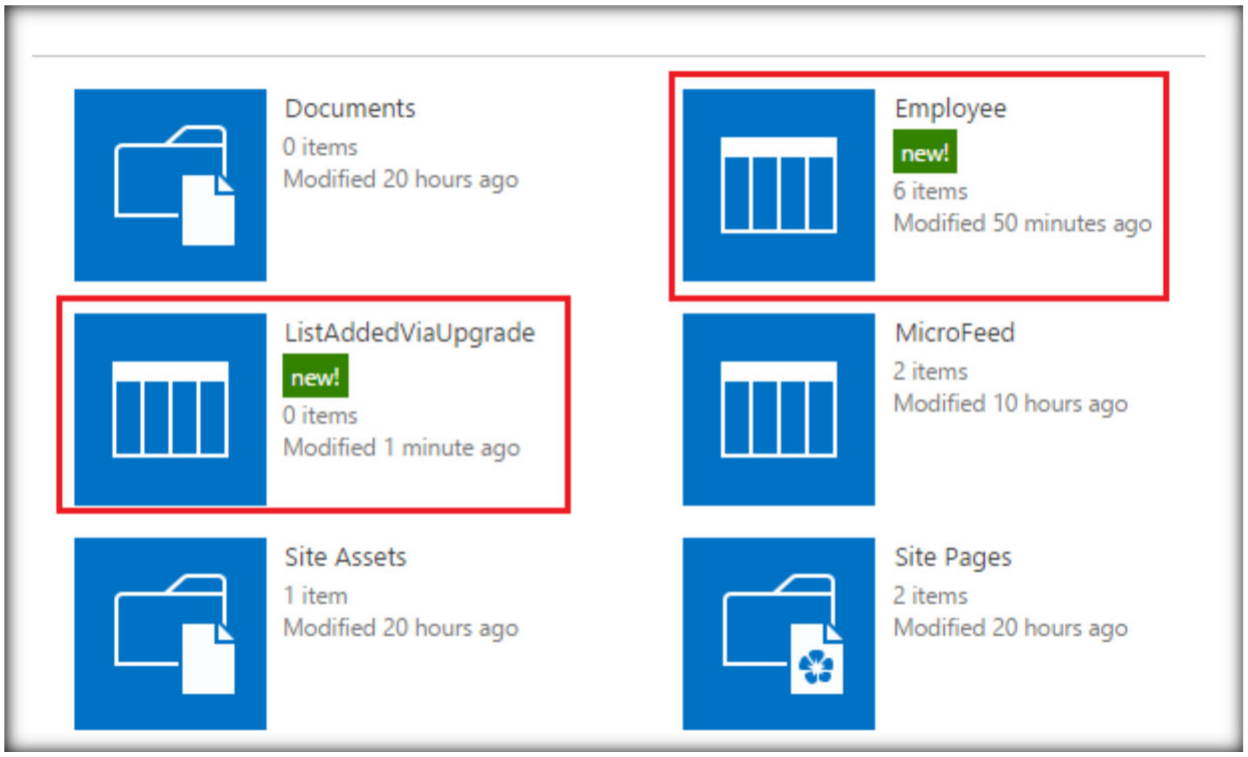


Upon clicking on OK it will give a warning whether to replace it with the new solution. Click on Replace it so that the new version is added.



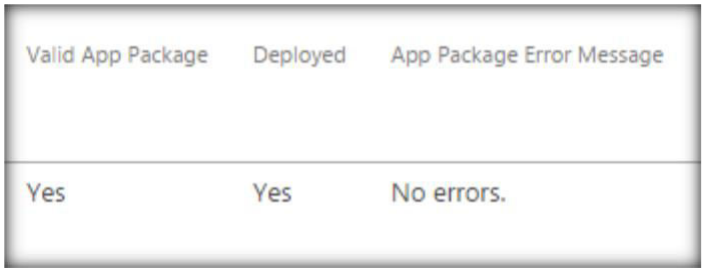
We can see that the version has upgraded from 1.0.0.0 to 2.0.0.0

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

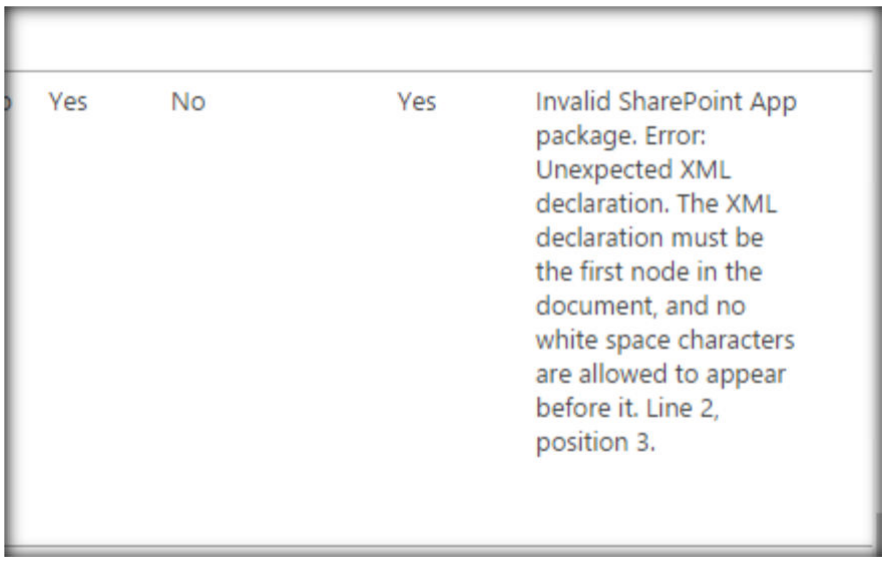


Resolve Package Errors

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



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

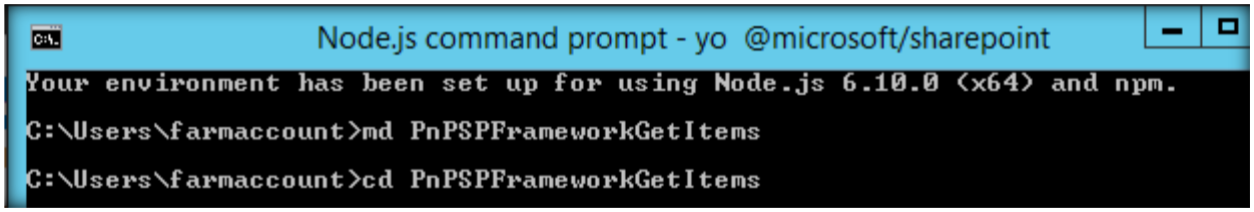
We can seamlessly integrate PnP JS file with SharePoint. The new Patterns and Practices JavaScript Core Library was created to help developers by simplifying common operations within SharePoint. Currently it contains a fluent API for working with the full SharePoint REST API as well as utility and helper 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

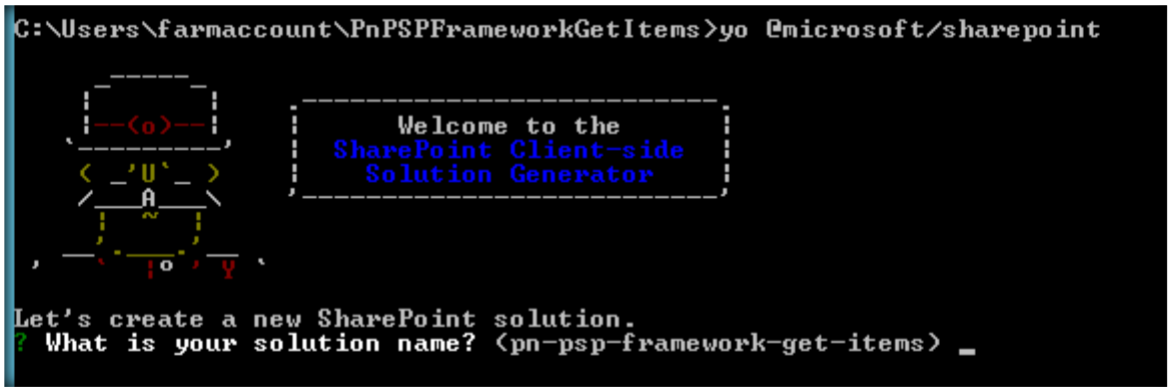
As part of this article we will see how to create a SharePoint Framework webpart that retrieves List Items using PnP JS. The major project files used in this solution has been zipped and uploaded at Microsoft TechNet Gallery. Feel free to download it.

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

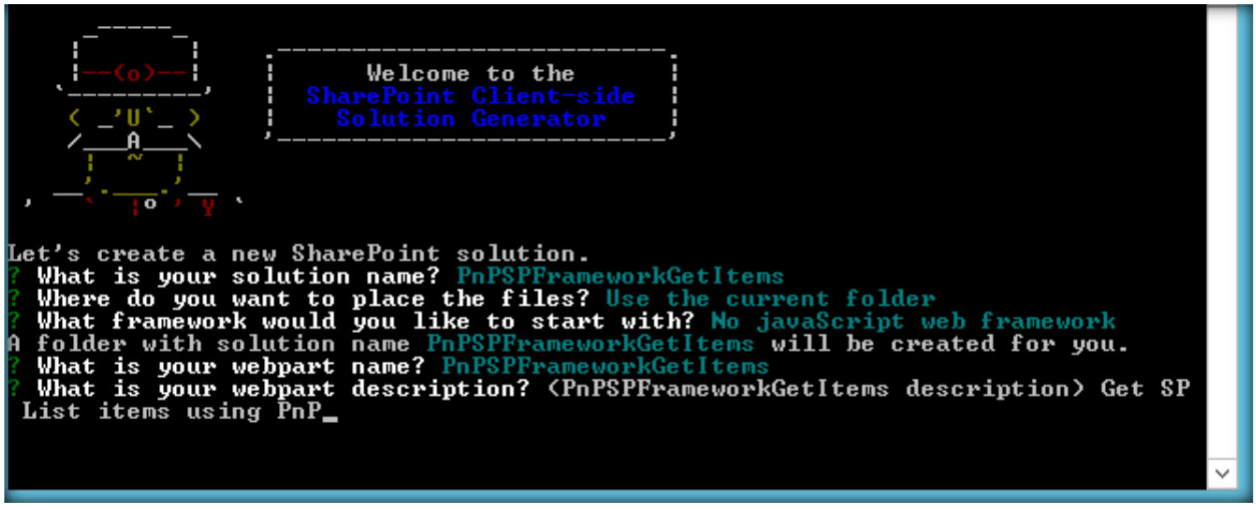


We will then create the client Web part by running the Yeoman SharePoint Generator. *yo @microsoft/sharepoint*

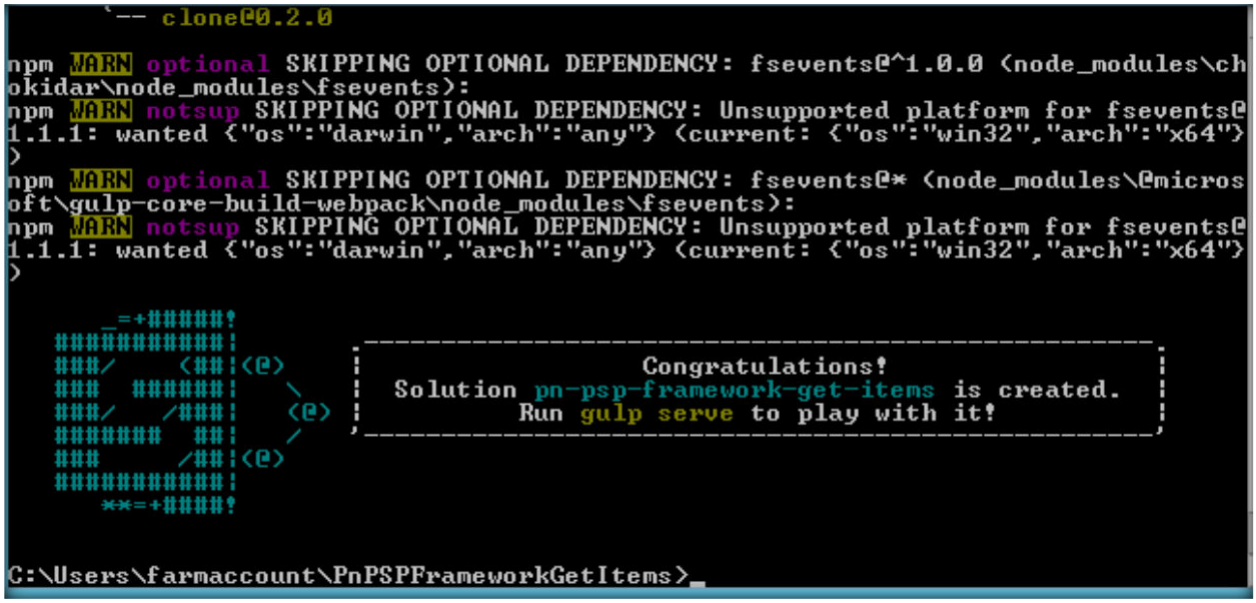


This will display the prompt, which we will have to fill up, so as to proceed with the project creation.

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

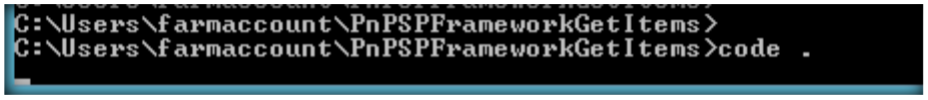


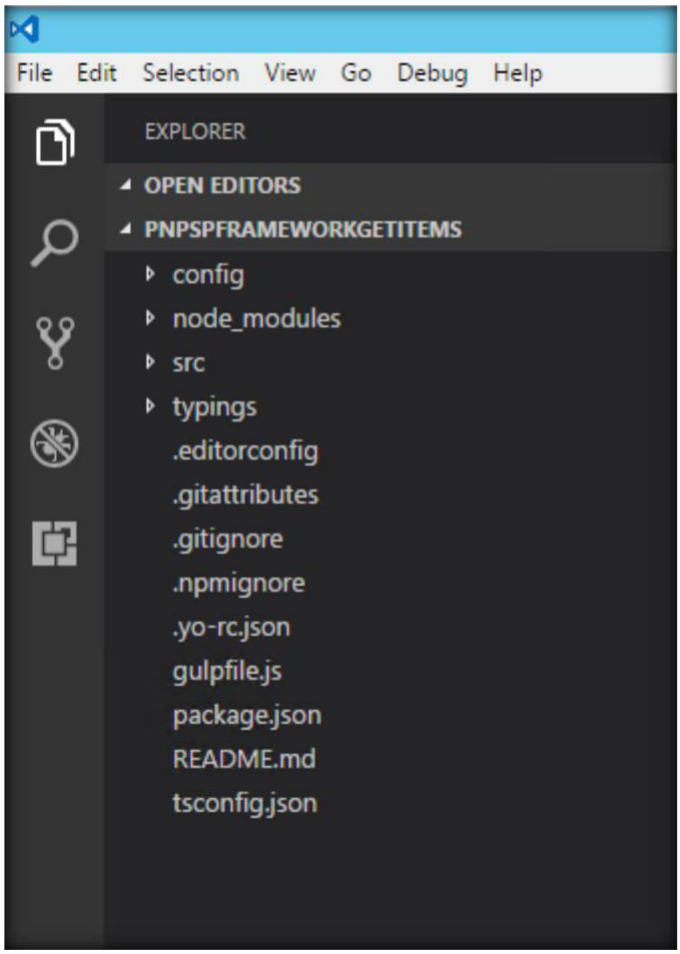
Yeoman has started working on the scaffolding of the project. It will install the required dependencies and scaffold the solution files for the 'PnPSPFrameworkGetItems' Web part, which will take some time to complete. Once completed, we will get a congratulations message.



Edit Webpart

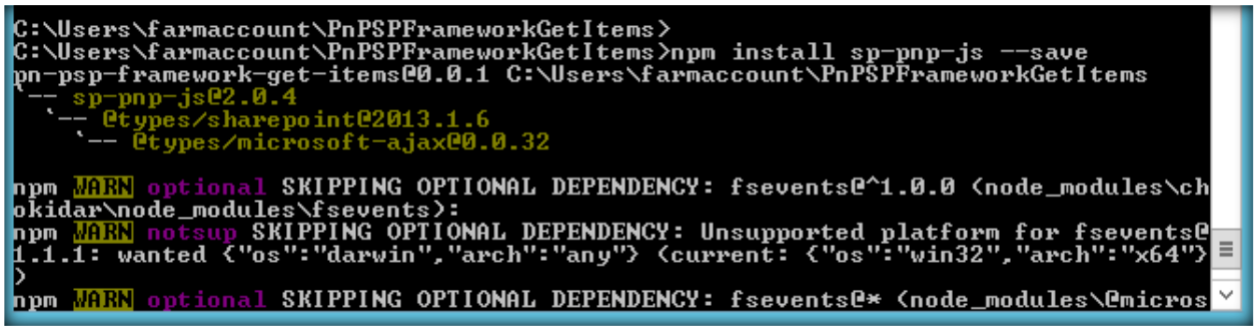
Run Code . to create the scaffolding and open the project in Visual Studio Code





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 as shown below.

npm install sp-pnp-js --save



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.

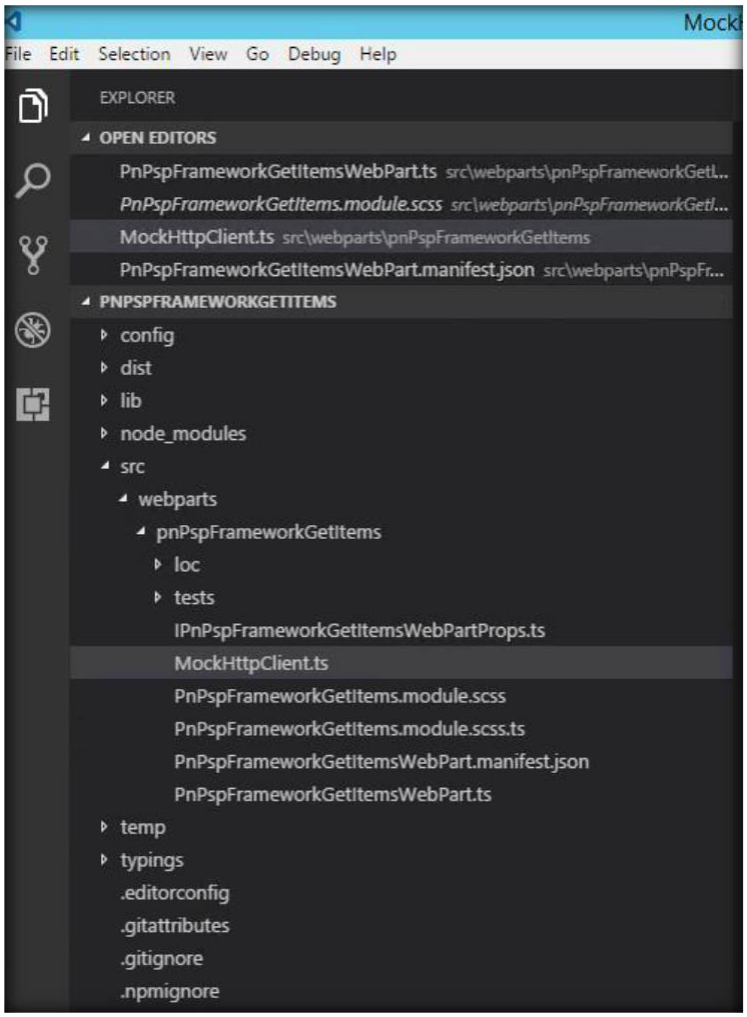
```
export interface ISPList {
     EmployeeId: string;
     EmployeeName: string;
     Experience: string;
     Location: string;
}
```

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.

```
import { ISPList } from './PnPspFrameworkGetItemsWebPart';

export default class MockHttpClient {
private static _items: ISPList[] = [{ EmployeeId: 'E123', EmployeeName: 'John', Experience: 'SharePoint',Location: 'India' },];
    public static get(restUrl: string, options?: any): Promise<ISPList[]> {
      return new Promise<ISPList[]>((resolve) => {
            resolve(MockHttpClient._items);
        });
    }
}
```



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 { I*PnPspFrameworkGetItems*WebPartProps } from './I* PnPspFrameworkGetItems*WebPartProps';"*

```
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' }
```

```
            ];
        return listData;
    }) as Promise<ISPList[]>;
}
```

Retrieve SharePoint List Items

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

```
private _getListData(): Promise<ISPList[]> {
return pnp.sp.web.lists.getByTitle("EmployeeList").items.get().then((response) => {

    return response;
  });

}
```

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.

```
private _renderListAsync(): void {

    if (Environment.type === EnvironmentType.Local) {
      this._getMockListData().then((response) => {
        this._renderList(response.value);
      });
    }
     else {
       this._getListData()
      .then((response) => {
        this._renderList(response.value);
      });
   }
}
```

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:

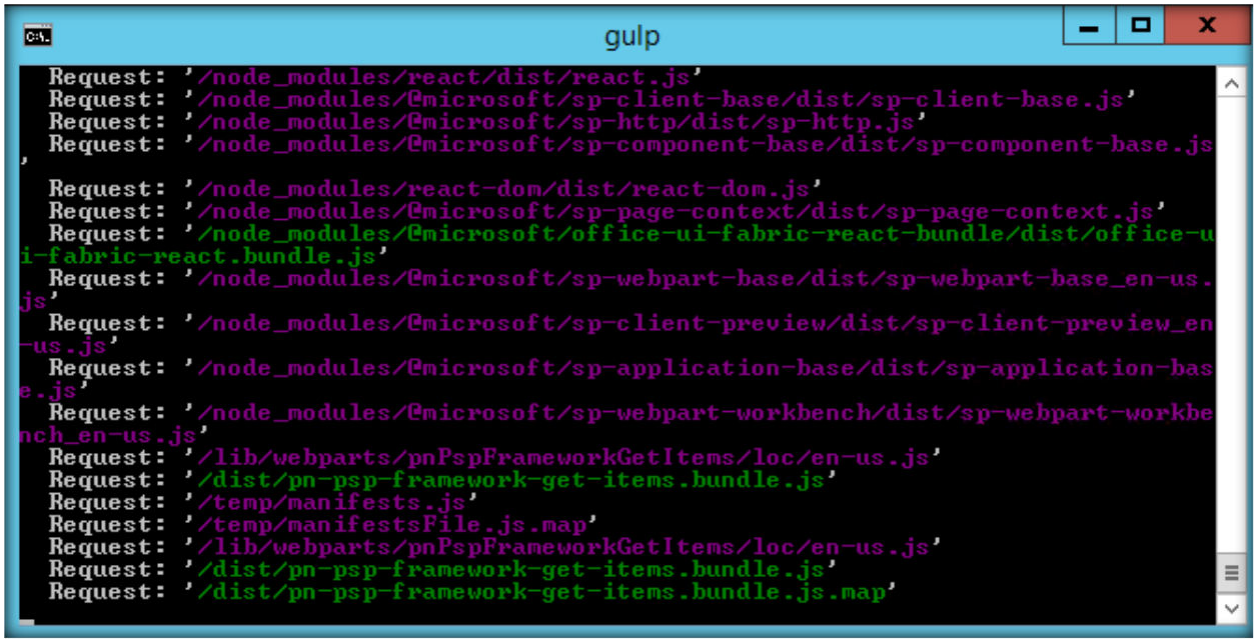
```
import pnp from 'sp-pnp-js';
import { Version } from '@microsoft/sp-core-library';

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

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

import styles from './PnPspFrameworkGetItems.module.scss';
import * as strings from 'pnPspFrameworkGetItemsStrings';
import { IPnPspFrameworkGetItemsWebPartProps }
   from './IPnPspFrameworkGetItemsWebPartProps';
import MockHttpClient from './MockHttpClient';

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


    export interface ISPList {
    EmployeeId: string;
    EmployeeName: string;
    Experience: string;
    Location: string;
  }

export default class PnPspFrameworkGetItemsWebPart extends
   BaseClientSideWebPart<IPnPspFrameworkGetItemsWebPartProps> {


private _getListData(): Promise<ISPList[]> {
return pnp.sp.web.lists.getByTitle("EmployeeList").items.get().then((response) => {

   return response;
 });

}

 private _renderListAsync(): void {

   if (Environment.type === EnvironmentType.Local) {
     this._getMockListData().then((response) => {
       this._renderList(response);
     });
   }
    else {
      this._getListData()
     .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 +=
    `<th>EmployeeId</th><th>EmployeeName</th><th>Experience</th><th>Location</th>`;
62.  items.forEach((item: ISPList) => {
63.    html += `
64.         <tr>
65.        <td>${item.EmployeeId}</td>
66.        <td>${item.EmployeeName}</td>
67.        <td>${item.Experience}</td>
68.        <td>${item.Location}</td>
69.        </tr>
70.        `;
71.  });
72.  html += `</table>`;
73.  const listContainer: Element = this.domElement.querySelector('#spListContainer');
74.  listContainer.innerHTML = html;
75. }
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
    SharePoint Framework Development using PnP JS Library</span>
84.
85. <p class="ms-font-l ms-fontColor-white" style="text-align: left">Demo : Retrieve
    Employee Data from SharePoint List</p>
86.      </div>
87.    </div>
88.    <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
89. <div style="background-color:Black;color:white;text-align: center;font-weight:
    bold;font-size:18px;">Employee Details</div>
90.    <br>
91. <div id="spListContainer" />
92.    </div>
93.  </div>
94. </div>`;
95. this._renderListAsync();
96.  }
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:
    'SharePoint',Location: 'India' },
 102.                      { EmployeeId: 'E567', EmployeeName: 'Martin', Experience:
    '.NET',Location: 'Qatar' },
 103.                     { EmployeeId: 'E367', EmployeeName: 'Luke', Experience:
    'JAVA',Location: 'UK' }
 104.                 ];
 105.             return listData;
 106.         }) as Promise<ISPList[]>;
```

```
    }

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

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

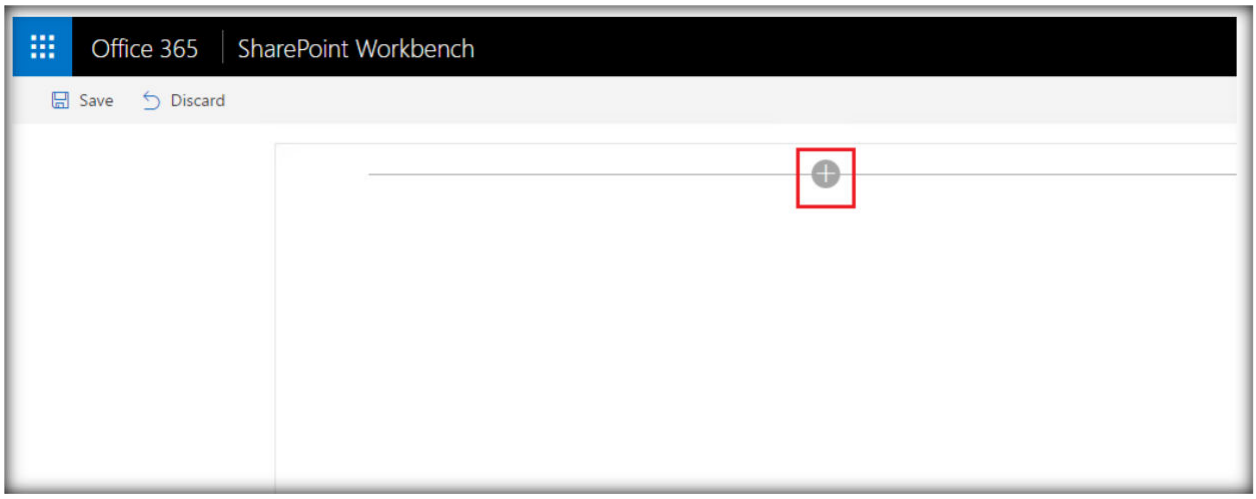
Package and Deploy the Solution

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

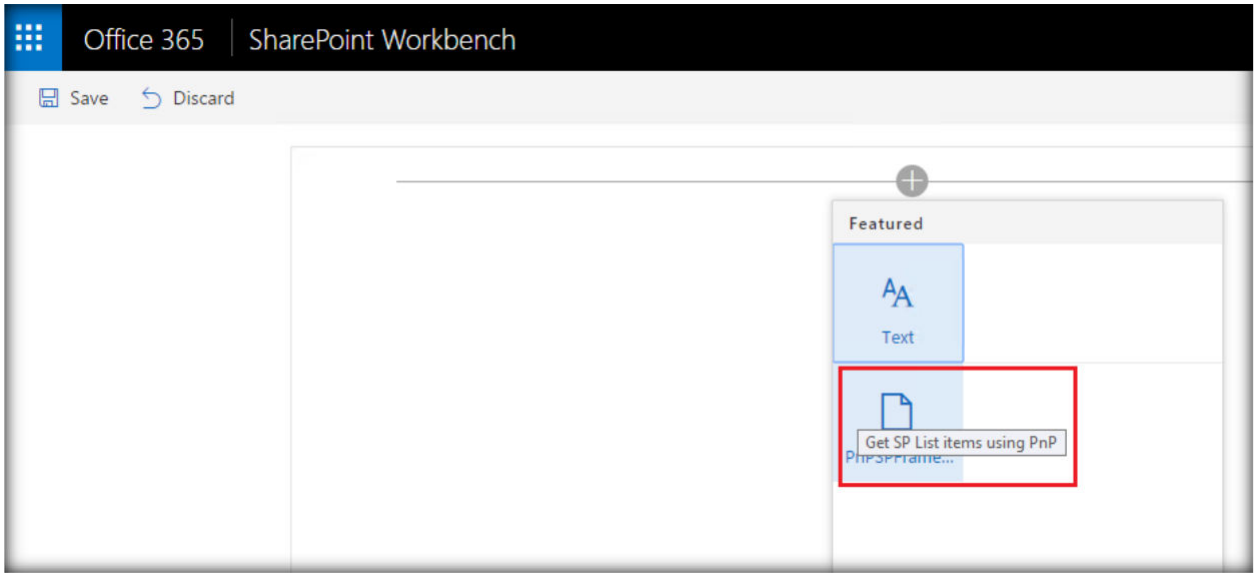


Test the Web part in local SharePoint Workbench

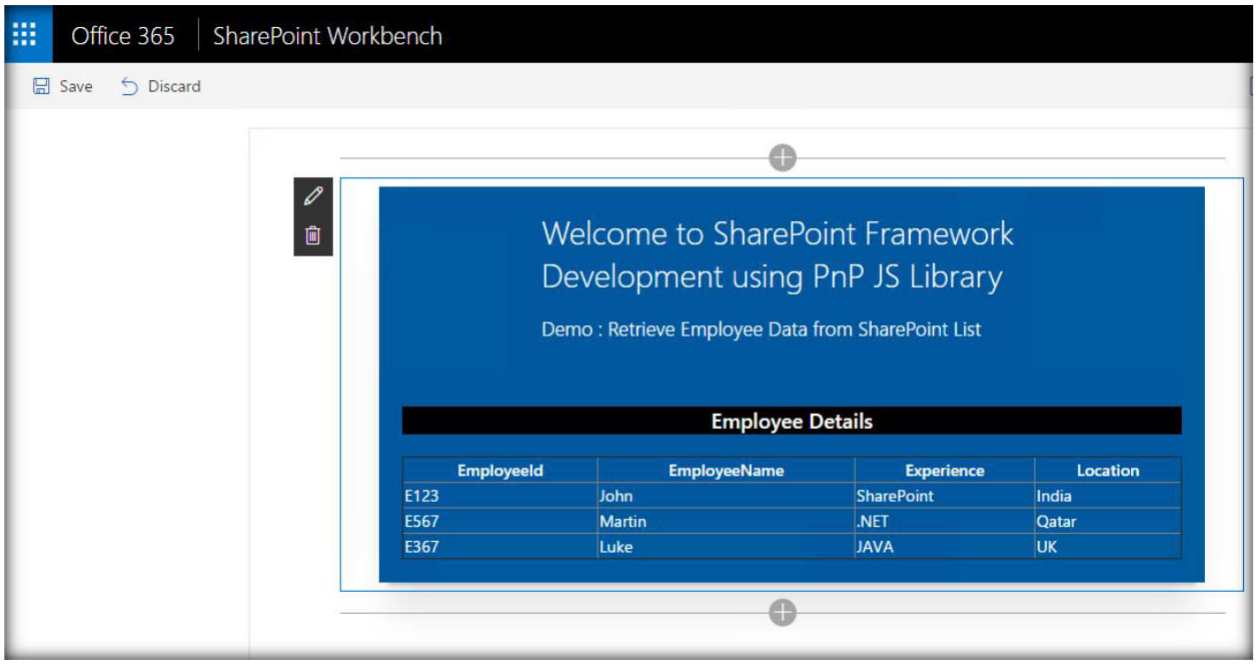
Now, we can see the output generated in the local SharePoint Workbench.



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

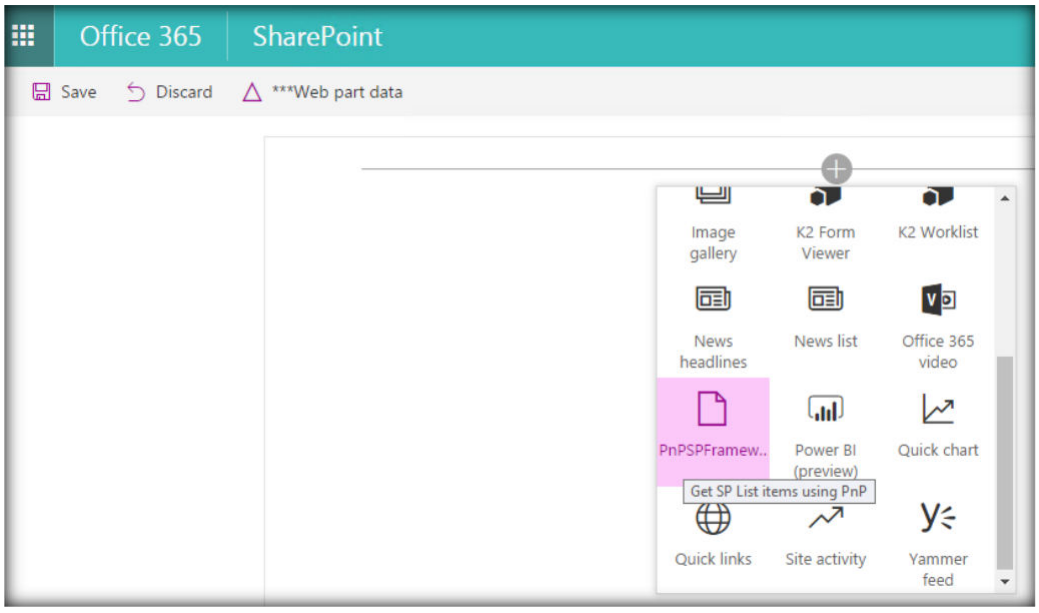


Since the environment is local, the mock data has been used to generate the table, as shown below.

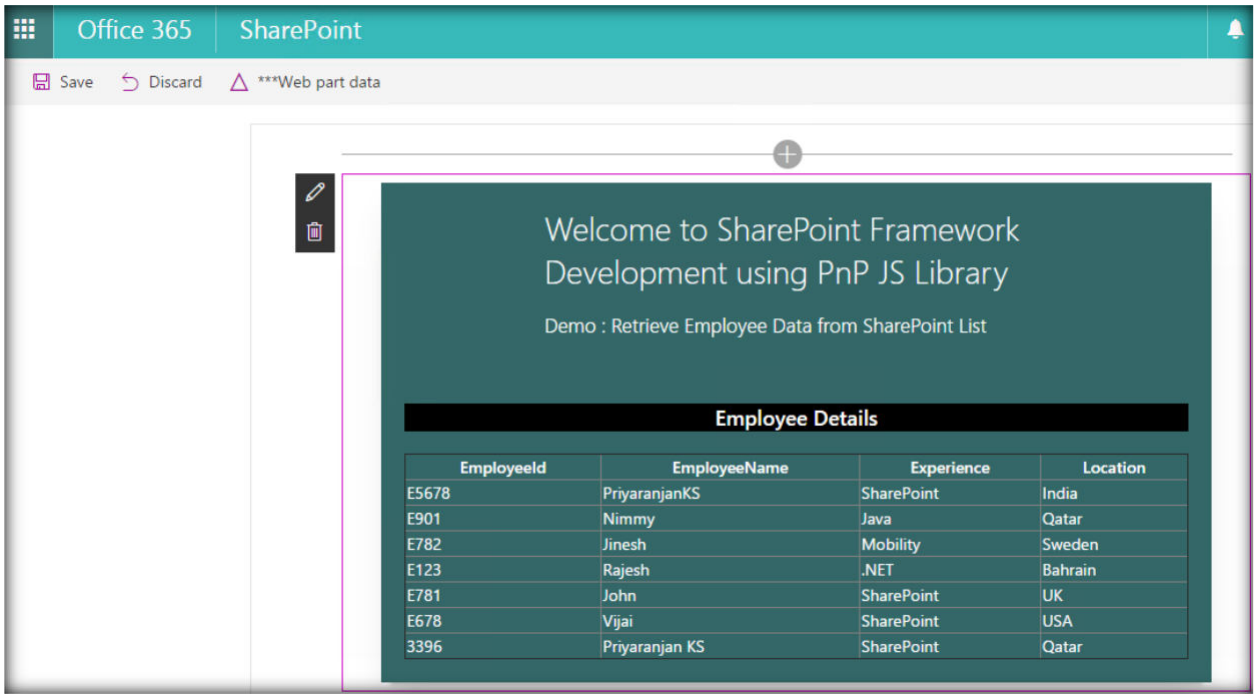


Test the Web part in SharePoint Online

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.



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



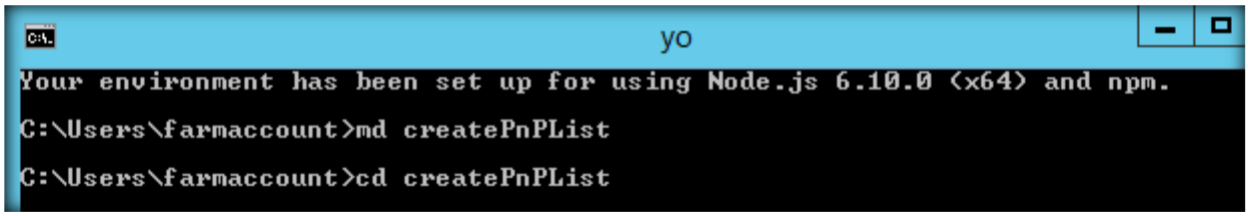
The major project files used in this solution has been zipped and uploaded at Microsoft TechNet Gallery. Feel free to download it.

SharePoint List Creation using PnP and SPFx

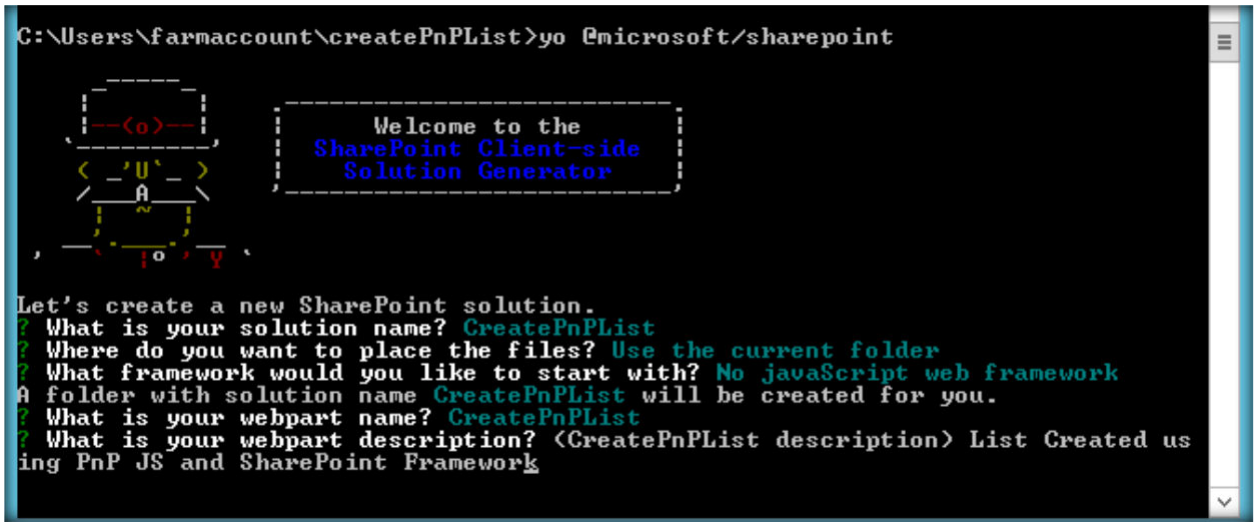
In this section, we will see how to use PnP JS in SPFx to create and provision a custom list. The major project files used in this solution has been zipped and uploaded at Microsoft TechNet Gallery. Feel free to download it.

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



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

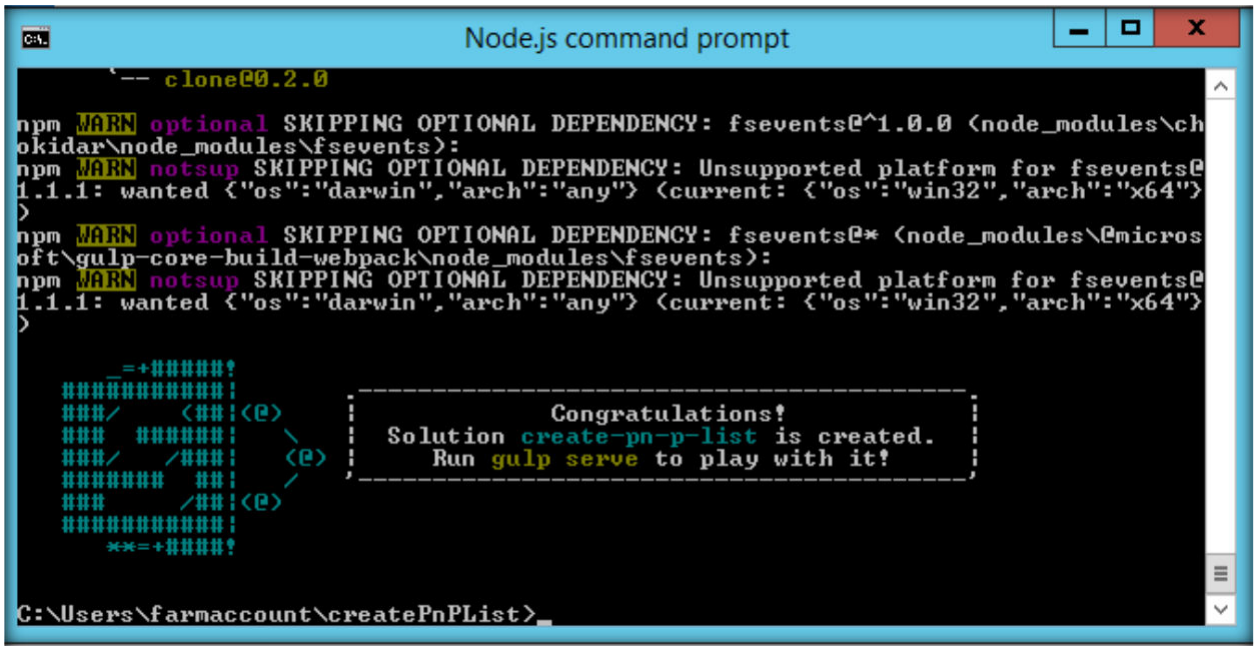


This will display the prompt, which we will have to fill up, so as to proceed with the project creation.

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

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

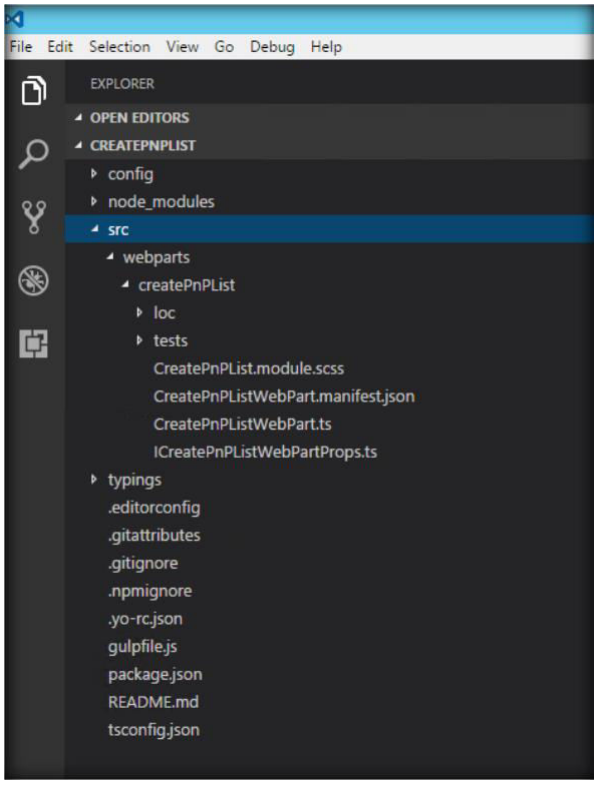
- Where do you want to place your files- Use current folder.
- What framework would you like to start with- Select "No javaScript web framework" for the time being, as this is a sample Web part.
- What is your Webpart name- We will specify it as 'CreatePnPList' and press Enter
- What is your Webpart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



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

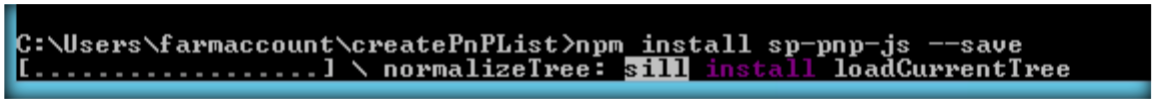
Run 'Code .' to open the project in Visual Studio Code.



Install PnP JS Module

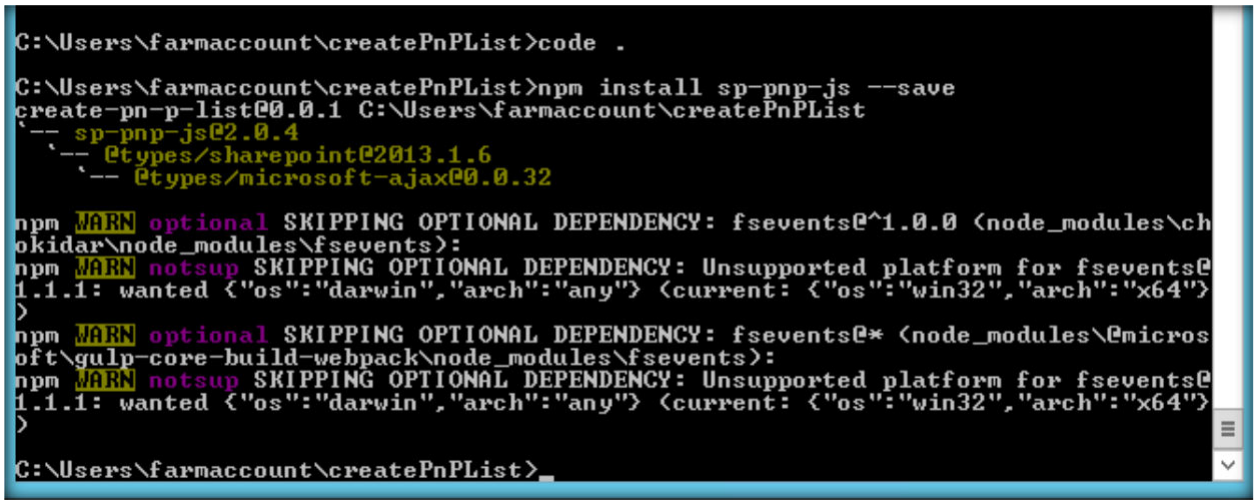
Now we have to load PnP JS file which we will use within the project to create list.We will be using npm to add PnP JS file.

npm install sp-pnp-js --save



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

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



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,
    spEnableCT).then(function(splist){
9.  document.getElementById("ListCreationStatus").innerHTML += `New List `+
    spListTitle+ ` Created`;
10.  });
11. }
```

Once the web part has been created, the status will be updated in the div.

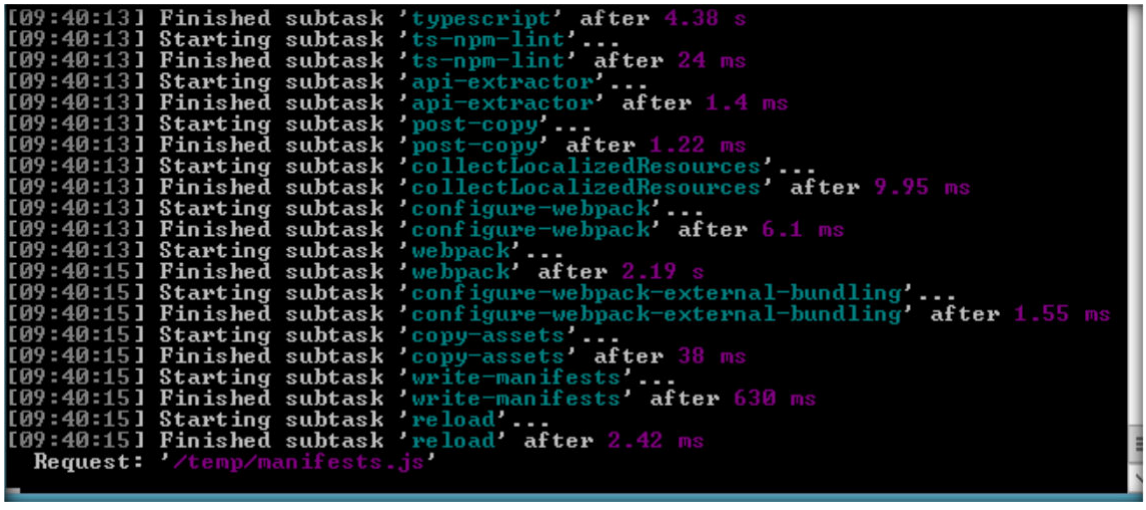
TS File code for Creating the List

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.

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



import { Version } from '@microsoft/sp-core-library';

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


import styles from './CreatePnPList.module.scss';
import * as strings from 'createPnPListStrings';
import { ICreatePnPListWebPartProps } from './ICreatePnPListWebPartProps';

export default class CreatePnPListWebPart extends
    BaseClientSideWebPart<ICreatePnPListWebPartProps> {

private CreateList(): void {

  let spWeb = new Web(this.context.pageContext.web.absoluteUrl);
  let spListTitle = "SPFxPnPList";
  let spListDescription = "SPFxPnP List";
  let spListTemplateId = 100;
  let spEnableCT = false;
spWeb.lists.add(spListTitle, spListDescription,spListTemplateId,
    spEnableCT).then(function(splist){
document.getElementById("ListCreationStatus").innerHTML += `New List `+
    spListTitle+ ` Created`;
  });
}

  public render(): void {
   this.domElement.innerHTML = `
     <div class="${styles.helloWorld}">
  <div class="${styles.container}">
    <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
      <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
        <span class="ms-font-xl ms-fontColor-white" style="font-size:28px">Welcome to
    SharePoint Framework Development using PnP JS Library</span>

        <p class="ms-font-l ms-fontColor-white" style="text-align: left">Demo : Create
    SharePoint List</p>
      </div>
    </div>
```

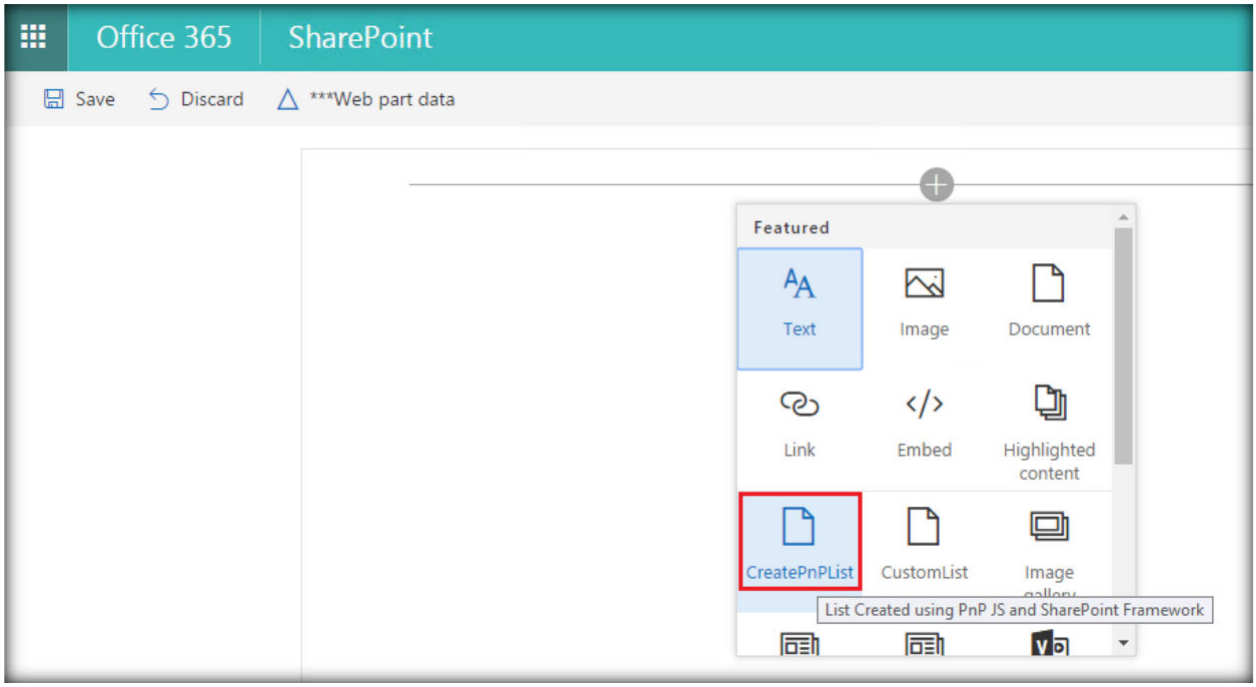
```
    <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
<div style="background-color:Black;color:white;text-align: center;font-weight: bold;font-size:18px;">Employee Details</div>
    <br>
<div id="ListCreationStatus" />
    </div>
  </div>
</div>`;
this.CreateList();
  }

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

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

Run gulp serve to package the solution.

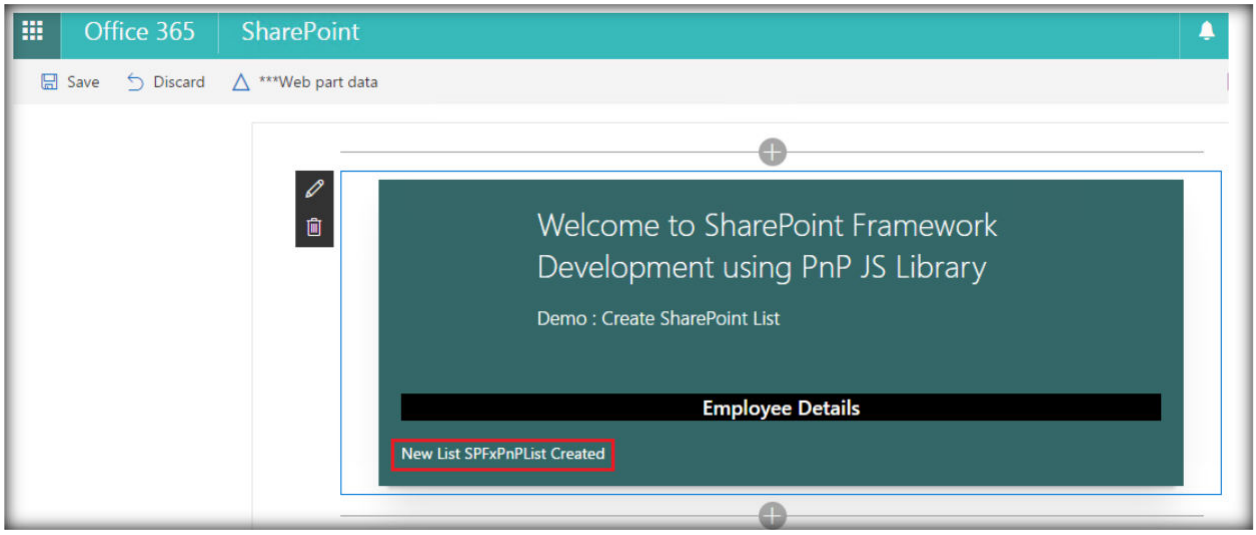


Test the Web part in SharePoint Online

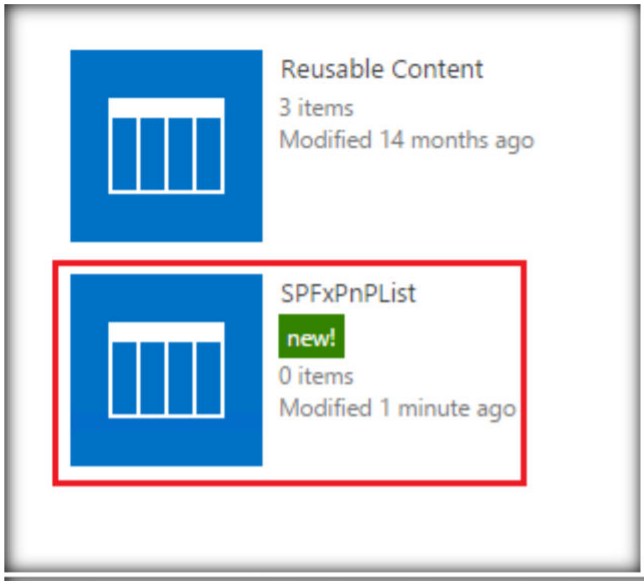
Now, let's test the Web part in SharePoint Workbench available in SharePoint Online. Once we have logged in to SharePoint Online, we can invoke the workbench by appending the text '_layouts/15/workbench.aspx' to SharePoint Online URL. Add the webpart to the page by selecting CreatePnPList icon.

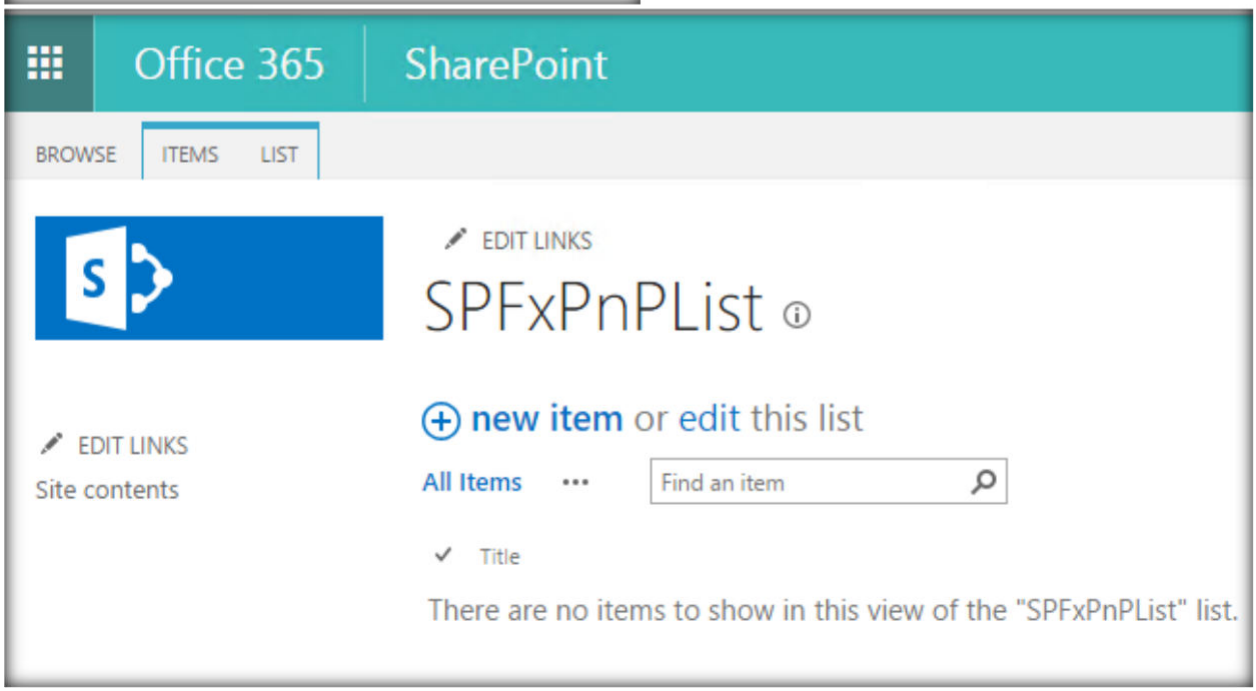


This will deploy the list to SharePoint and will show a success message in the UI as shown below.



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





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

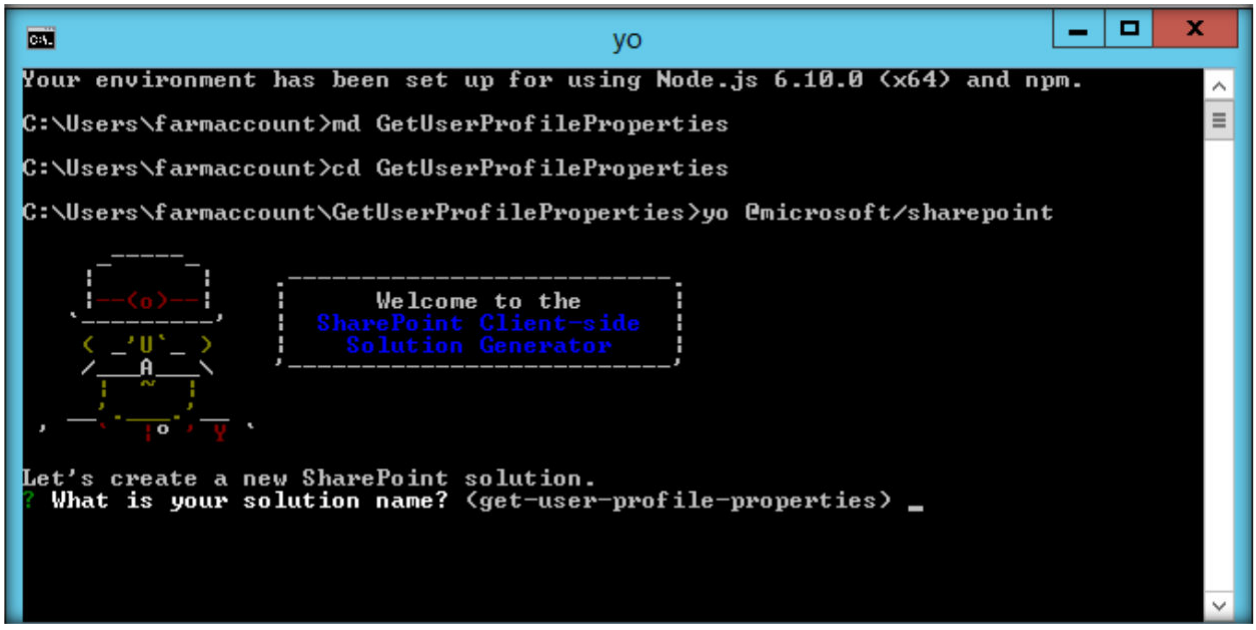
Retrieve User Profile Properties using SPFx and PnP JS

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.

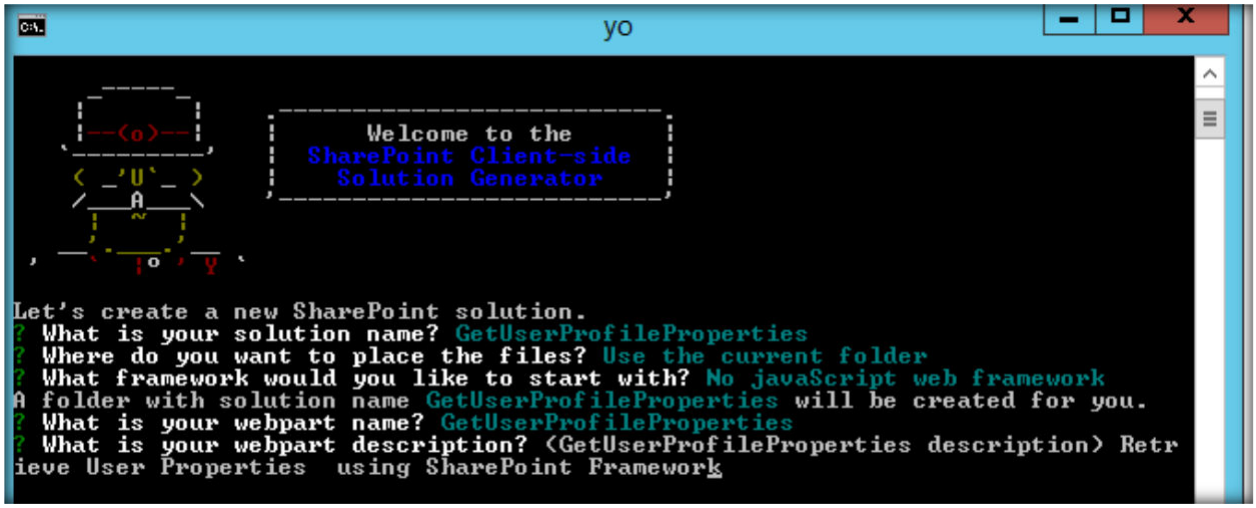
Create the Web part Project

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



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

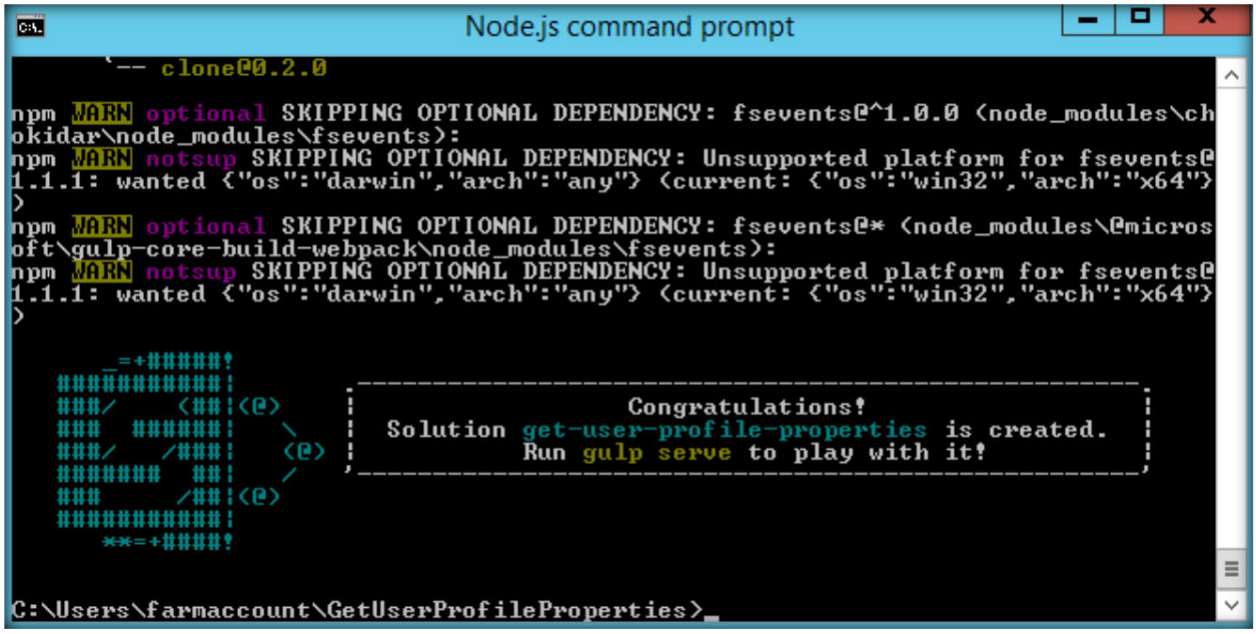


This will display the prompt, which we will have to fill up, so as to proceed with the project creation.

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

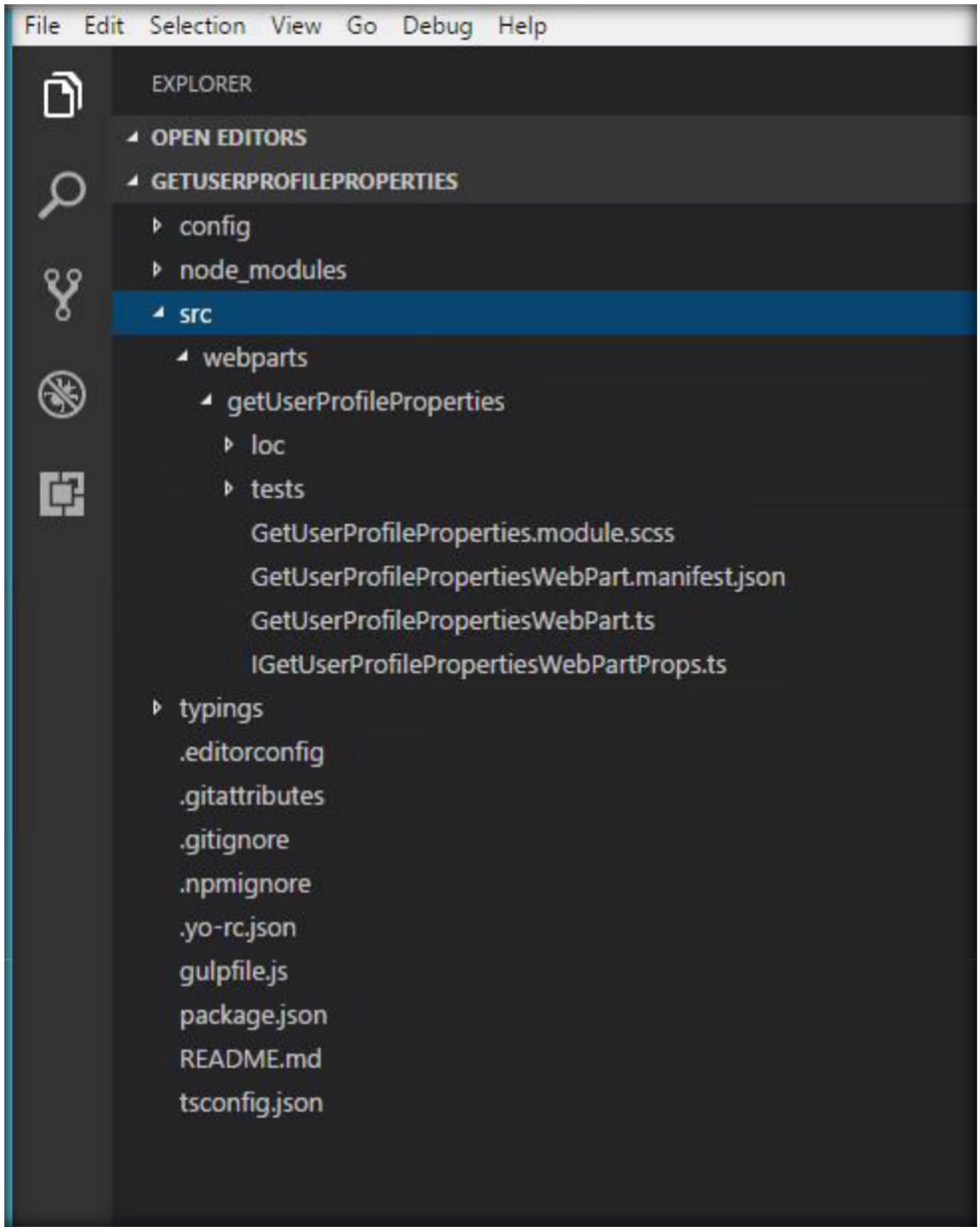
On pressing enter, we will be asked to chose the working folder for the project.

- 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'.



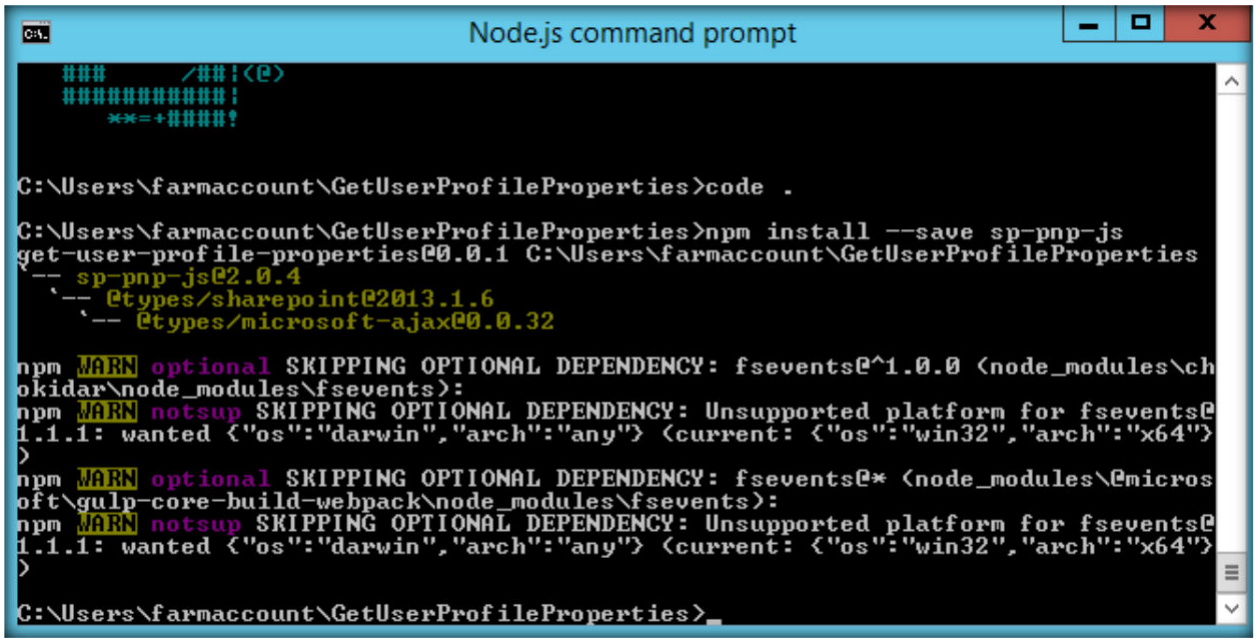
Edit the web part

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



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

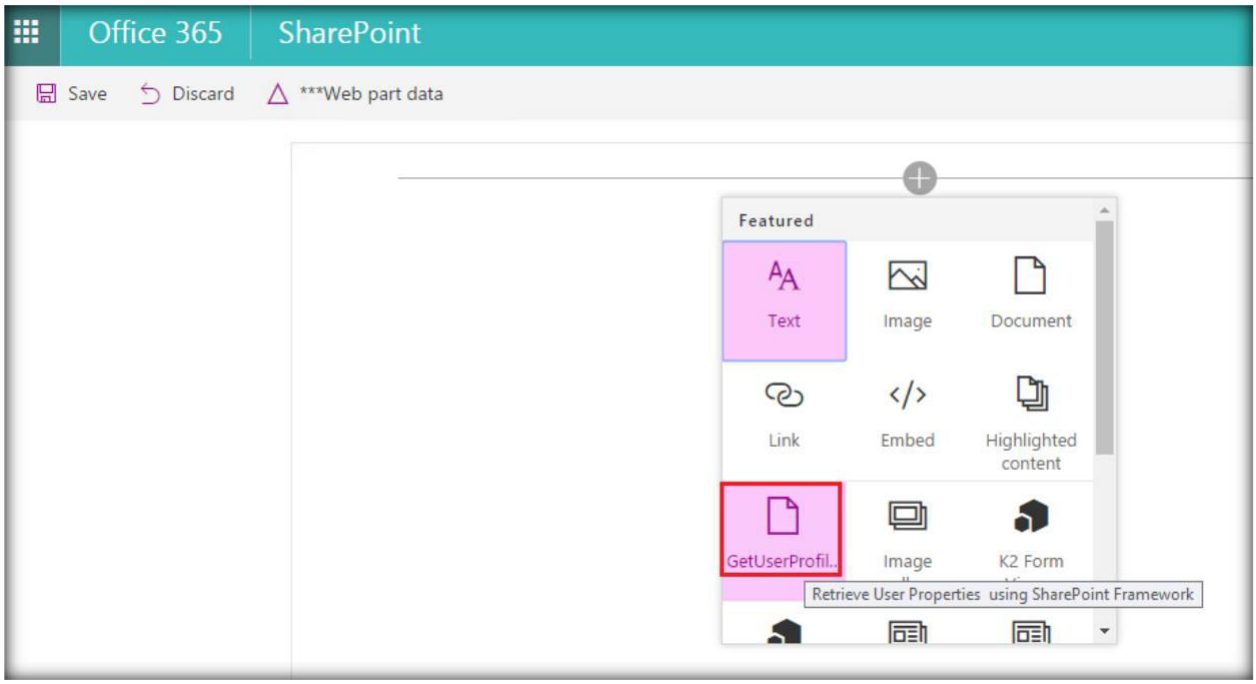
The entire TS file contents is as shown below. this.GetUserProperties() in the render method will call the function that will call the function that gets the User Profile properties of the user. It will be then displayed within the div element declared in the render method.

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

import { Version } from '@microsoft/sp-core-library';
import {
 BaseClientSideWebPart,
 IPropertyPaneConfiguration,
 PropertyPaneTextField
} from '@microsoft/sp-webpart-base';
import { escape } from '@microsoft/sp-lodash-subset';

import styles from './GetUserProfileProperties.module.scss';
import * as strings from 'getUserProfilePropertiesStrings';
import { IGetUserProfilePropertiesWebPartProps }
from './IGetUserProfilePropertiesWebPartProps';

export default class GetUserProfilePropertiesWebPart extends
BaseClientSideWebPart<IGetUserProfilePropertiesWebPartProps> {

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);
});

}

 public render(): void {

    this.domElement.innerHTML = `
    <div class="${styles.helloWorld}">
 <div class="${styles.container}">
   <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
     <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
       <span class="ms-font-xl ms-fontColor-white" style="font-size:28px">Welcome to
SharePoint Framework Development using PnP JS Library</span>

<p class="ms-font-l ms-fontColor-white" style="text-align: left">Demo : Retrieve User
Profile Properties</p>
     </div>
   </div>
   <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
<div style="background-color:Black;color:white;text-align: center;font-weight:
bold;font-size:18px;">User Profile Details</div>
   <br>
<div id="spUserProfileProperties" />
```

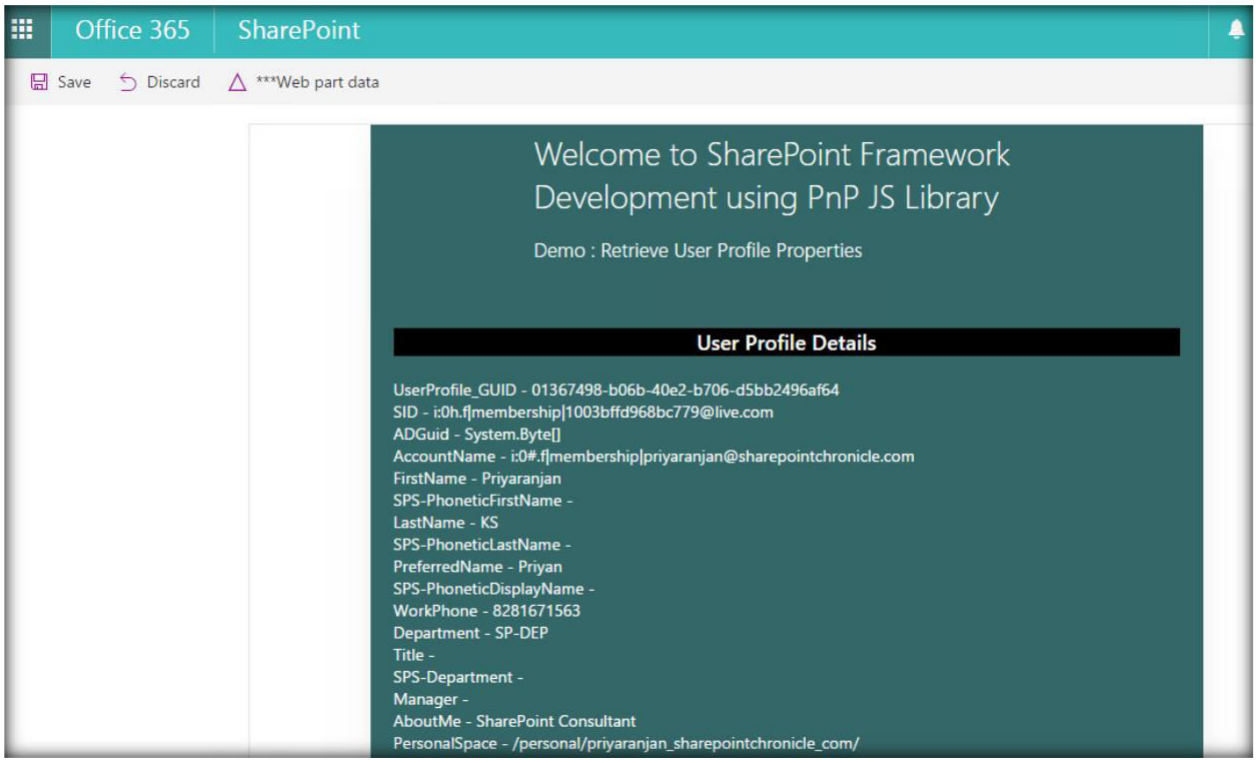
```
    </div>
  </div>
</div>`;
this.GetUserProperties();
  }

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

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

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.



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

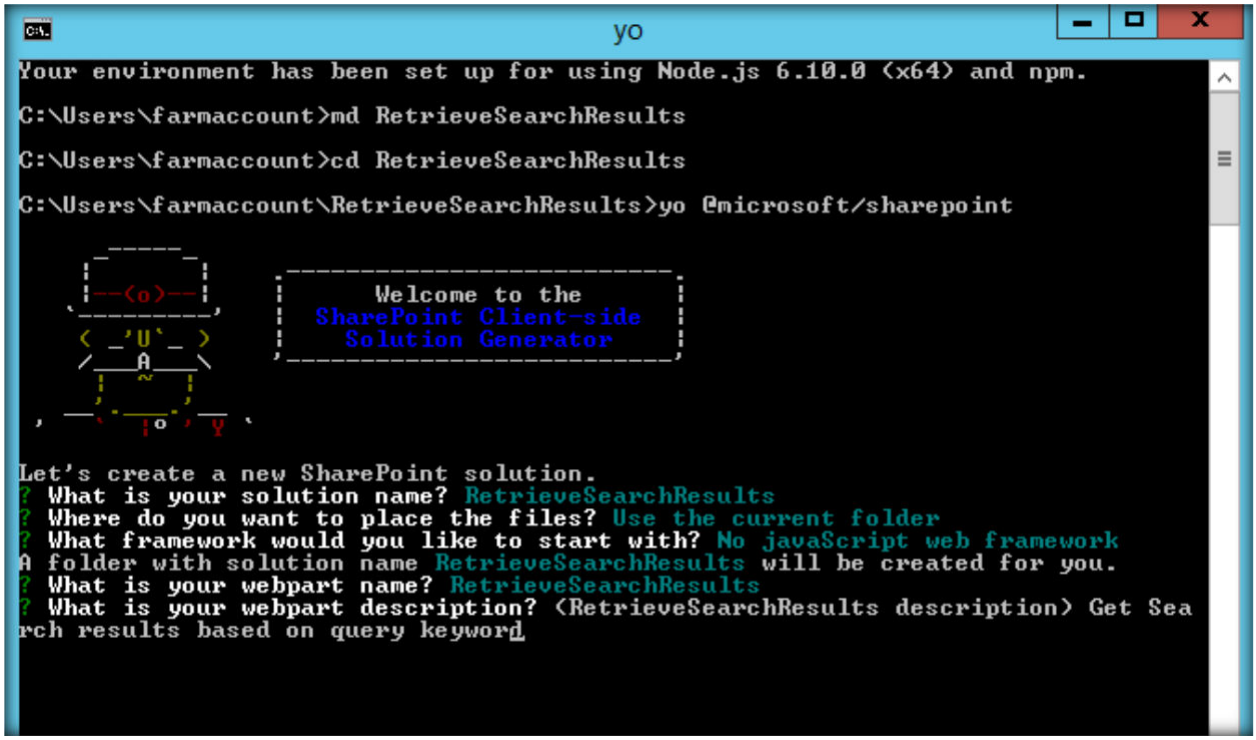
In this section, we will try to work with SharePoint Search and retrieve the search results based on a keyword using SPFx and PnP JS

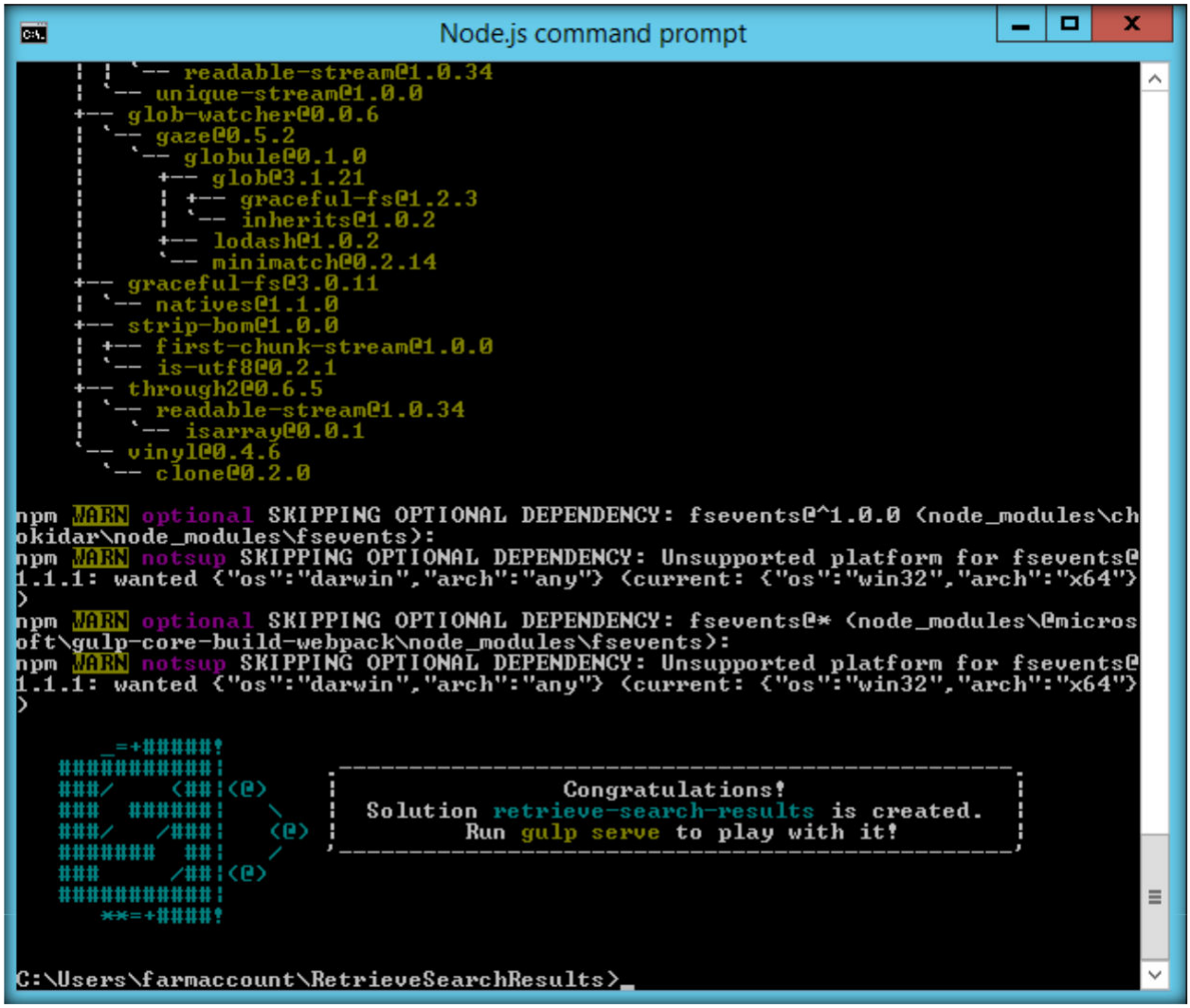
Create the Web part Project

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

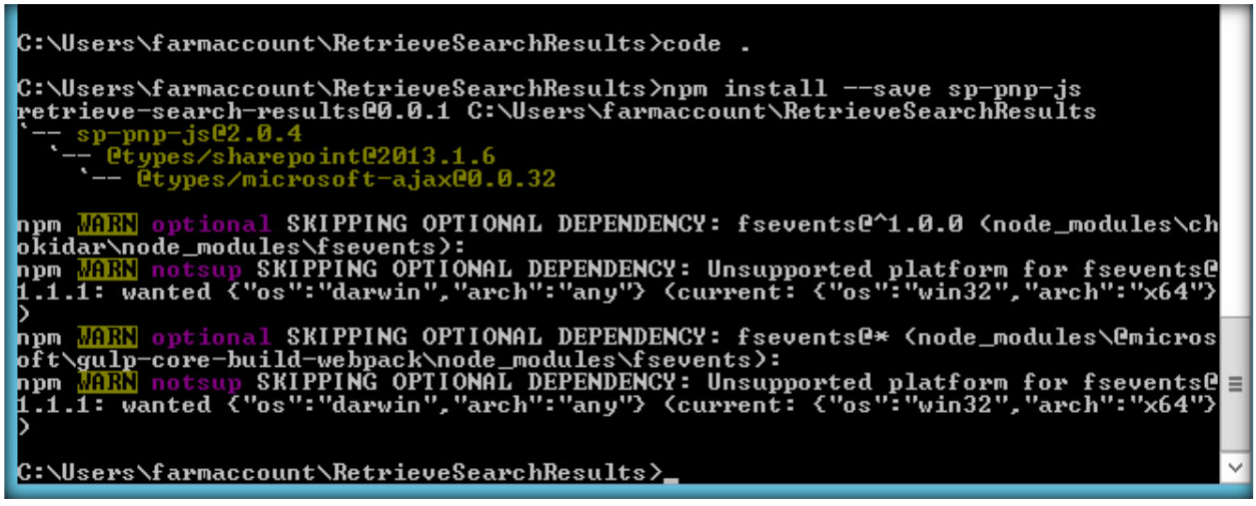
Let's move to the newly created working directory, using the command.
cd RetrieveSearchResults

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

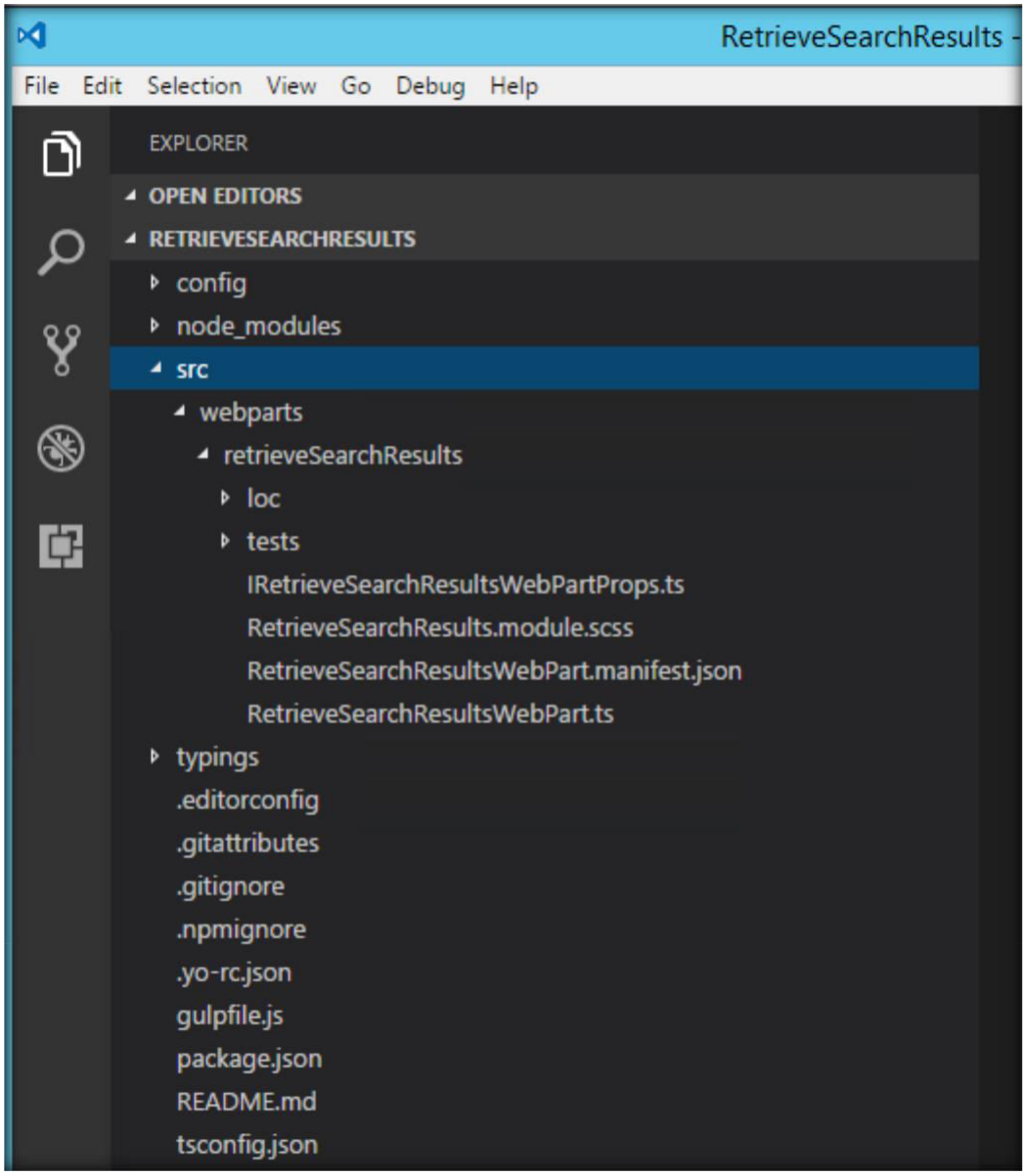




Run Code . to create scaffolding of the project structure.



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 +"<br/>"+"Rank - " + object.Rank
    +"<br/>"+"File Type - " + object.FileType+"<br/>"+ "Original Path - "
    +object.OriginalPath +"<br/>"+" Summary - "+ object.HitHighlightedSummary +
    "<br/>"+"<br/>";
11.  });
12.  document.getElementById("spSearchresults").innerHTML = propValue;
13.  }).catch(function(err) {
14.   console.log("Error: " + err);
15.  });
16.
17. }
```

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.

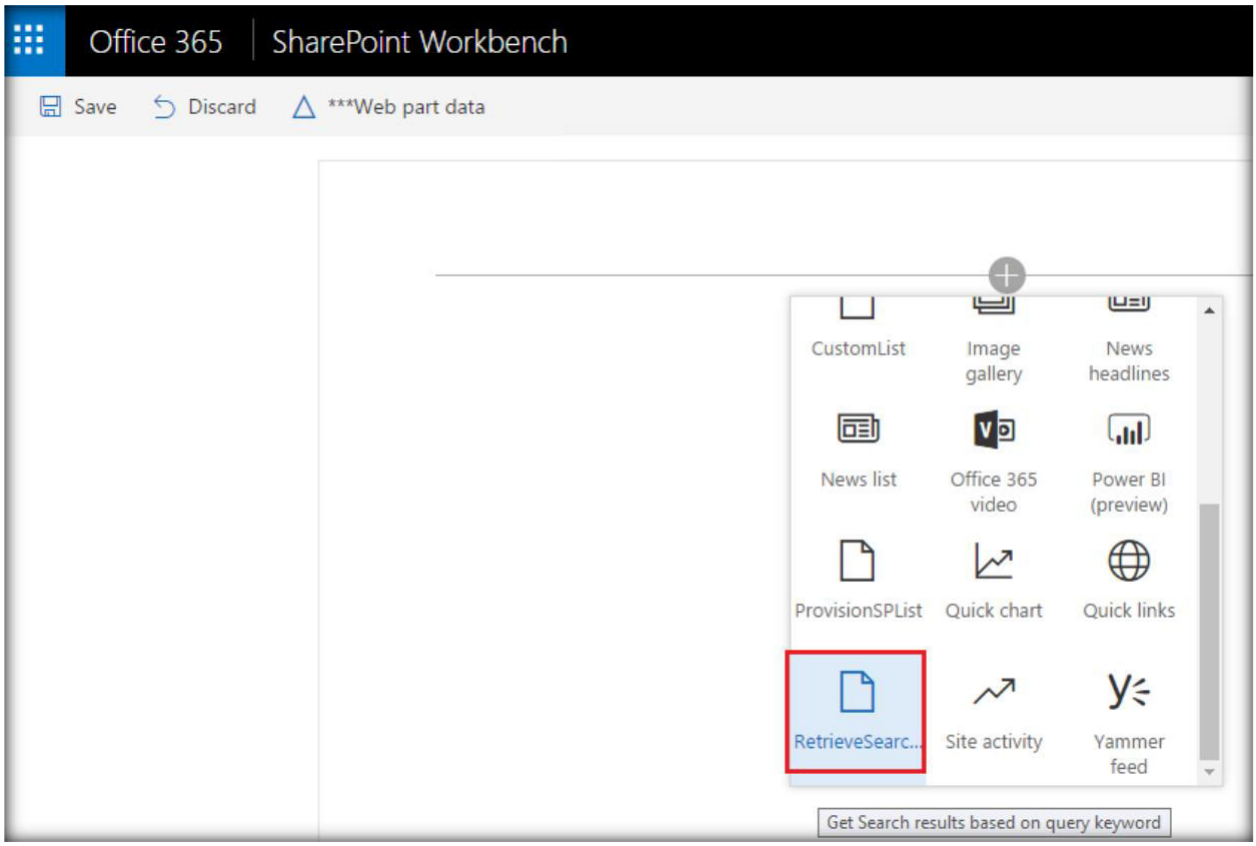
```
1.  import * as pnp from 'sp-pnp-js';
2.  import  { SearchQuery, SearchResults } from "sp-pnp-js";
3.
4.  import { Version } from '@microsoft/sp-core-library';
5.  import {
6.   BaseClientSideWebPart,
7.   IPropertyPaneConfiguration,
8.   PropertyPaneTextField
9.  } from '@microsoft/sp-webpart-base';
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
    './IRetrieveSearchResultsWebPartProps';
16.
17. export default class RetrieveSearchResultsWebPart extends
    BaseClientSideWebPart<IRetrieveSearchResultsWebPartProps> {
18. private GetSearchresults(): void {
19.
```

```
pnp.sp.search("SharePoint").then((result : SearchResults) => {
var props = result.PrimarySearchResults;
debugger;

 var propValue = "";
 var counter = 1;
 props.forEach(function(object) {
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/>";
 });
 document.getElementById("spSearchresults").innerHTML = propValue;
 }).catch(function(err) {
  console.log("Error: " + err);
 });

}


 public render(): void {

    this.domElement.innerHTML = `
    <div class="${styles.helloWorld}">
 <div class="${styles.container}">
   <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
     <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-xlPush2 ms-u-lgPush1">
<span class="ms-font-xl ms-fontColor-white" style="font-size:28px">Welcome to SharePoint Framework Development using PnP JS Library</span>

<p class="ms-font-l ms-fontColor-white" style="text-align: left">Demo : Retrieve SharePoint Search Result</p>
     </div>
   </div>
   <div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
<div style="background-color:Black;color:white;text-align: center;font-weight: bold;font-size:18px;">Search Results </div>
   <br>
<div id="spSearchresults" />
   </div>
 </div>
</div>`;
this.GetSearchresults();
 }

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

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

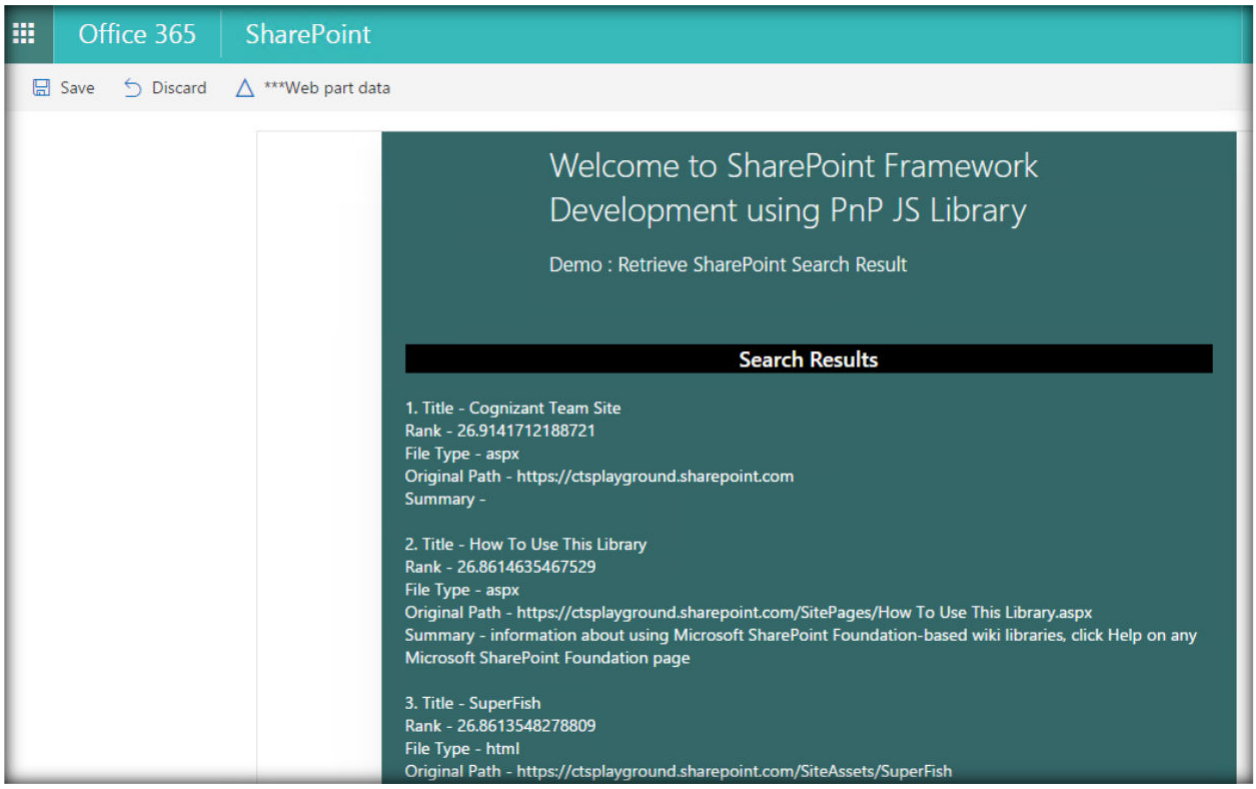
```
                label: strings.DescriptionFieldLabel
              })
            ]
          }
        ]
      }
    ]
  };
 }
}
```

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.

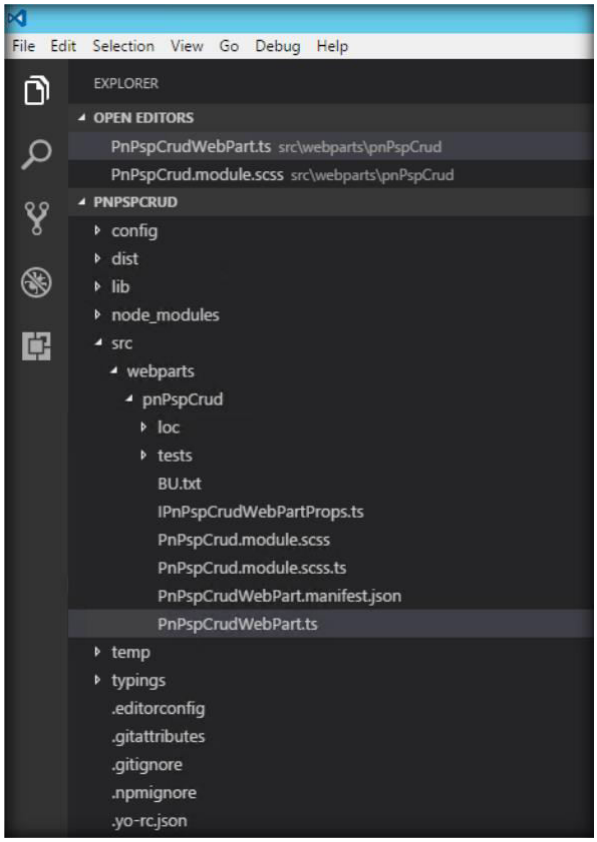
- What is your solution name? : Set it to 'PnPSPCRUD'.

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.

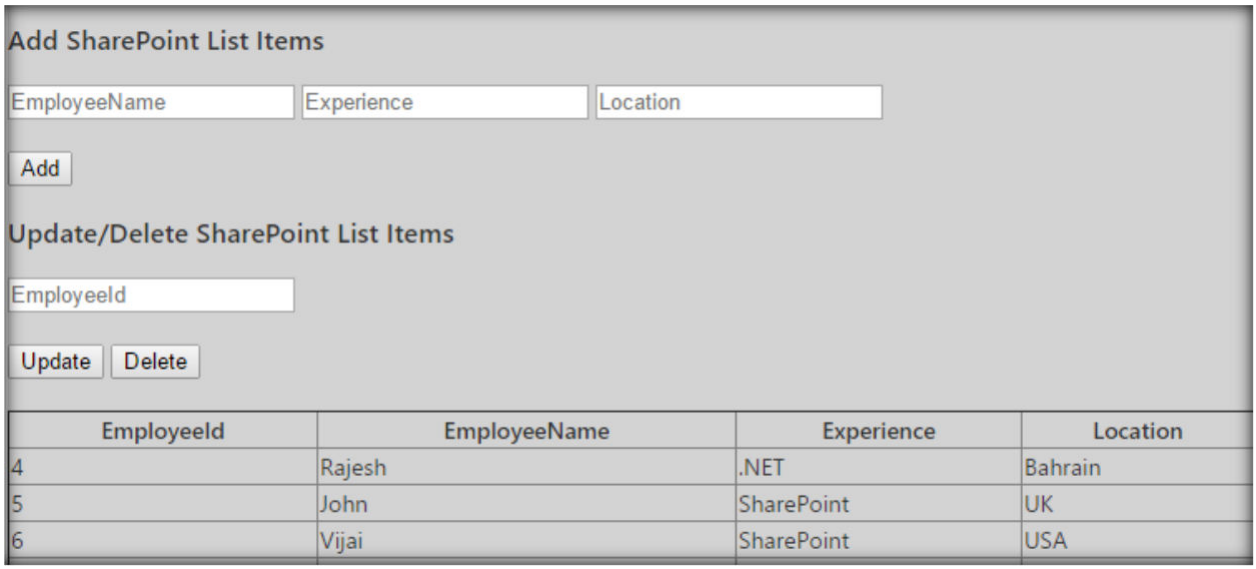
npm install sp-pnp-js --save

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.



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:

```
AddItem()
{

      pnp.sp.web.lists.getByTitle('EmployeeList').items.add({
      EmployeeName : document.getElementById('EmployeeName')["value"],
      Experience : document.getElementById('Experience')["value"],
      Location:document.getElementById('Location')["value"]
     });
alert("Record with Employee Name : "+
document.getElementById('EmployeeName')["value"] + " Added !");

}

UpdateItem()
{

    var id = document.getElementById('EmployeeId')["value"];
    pnp.sp.web.lists.getByTitle("EmployeeList").items.getById(id).update({
     EmployeeName : document.getElementById('EmployeeName')["value"],
     Experience : document.getElementById('Experience')["value"],
     Location:document.getElementById('Location')["value"]
  });
alert("Record with Employee Name : "+ document.getElementById('EmployeeName')["value"]
+ " Updated !");
}

DeleteItem()
{

pnp.sp.web.lists.getByTitle("EmployeeList").items.getById(document.getElementById('Empl
oyeeId')["value"]).delete();
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.

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

import { Version } from '@microsoft/sp-core-library';
import {
  BaseClientSideWebPart,
  IPropertyPaneConfiguration,
  PropertyPaneTextField
} from '@microsoft/sp-webpart-base';
import { escape } from '@microsoft/sp-lodash-subset';

import styles from './PnPspCrud.module.scss';
import * as strings from 'pnPspCrudStrings';
import { IPnPspCrudWebPartProps } from './IPnPspCrudWebPartProps';

```

```
15.              export interface ISPList {
16.          ID: string;
17.          EmployeeName: string;
18.          Experience: string;
19.          Location: string;
20.        }
21.
22. export default class PnPspCrudWebPart extends
    BaseClientSideWebPart<IPnPspCrudWebPartProps> {
23.
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-
    collapse: collapse;">';
49. html +=
    `<th>EmployeeId</th><th>EmployeeName</th><th>Experience</th><th>Location</th>`;
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">
```

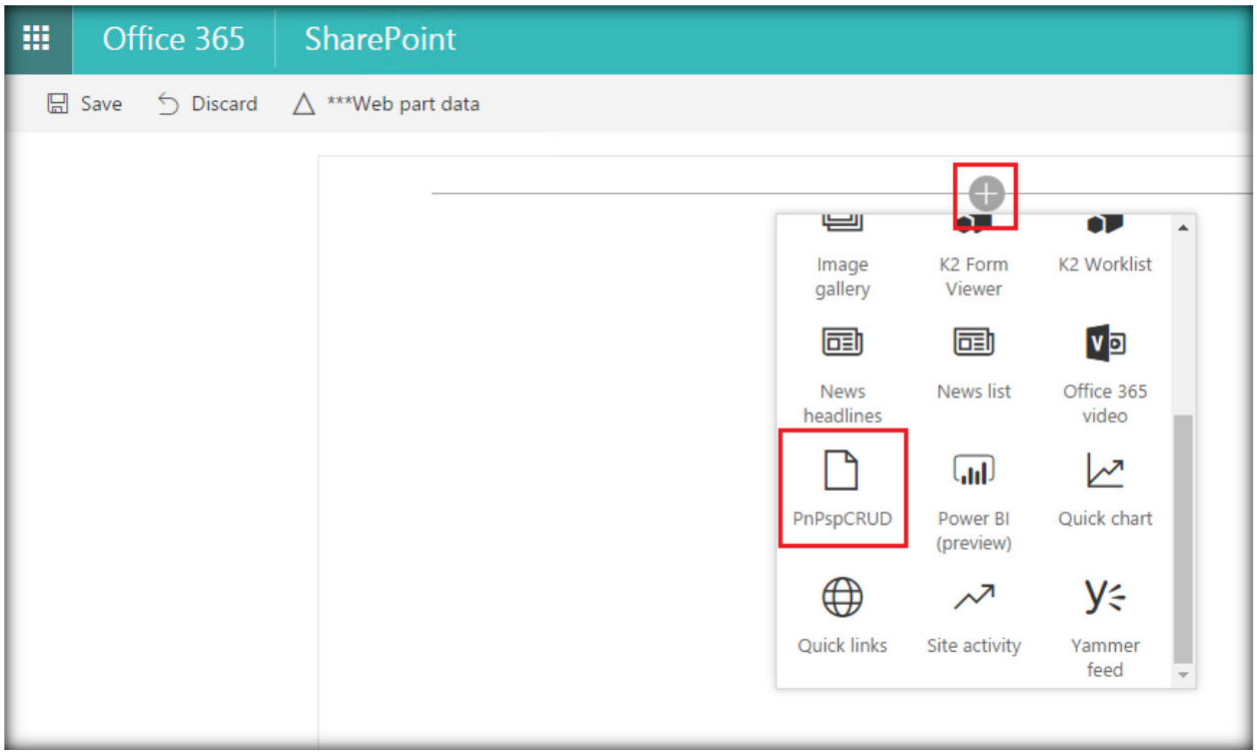
```
<span class="ms-font-xl ms-fontColor-white" style="font-size:28px">Welcome to SharePoint Framework Development using PnP JS Library</span>
<p class="ms-font-l ms-fontColor-white" style="text-align: left">Demo : SharePoint List CRUD using PnP JS and SPFx</p>
   </div>
</div>
<div class="ms-Grid-row ms-bgColor-themeDark ms-fontColor-white ${styles.row}">
   <div style="background-color:Black;color:white;text-align: center;font-weight: bold;font-size:18px;">Employee Details</div>

</div>
<div style="background-color: lightgrey" >
   <form >
      <br>
      <div data-role="header">
         <h3>Add SharePoint List Items</h3>
      </div>
      <div data-role="main" class="ui-content">
         <div >
            <input id="EmployeeName"  placeholder="EmployeeName"/>
            <input id="Experience"  placeholder="Experience"  />
            <input id="Location"  placeholder="Location"/>
         </div>
         <div></br></div>
         <div >
            <button id="AddItem"  type="submit" >Add</button>
         </div>
      </div>
      <div data-role="header">
         <h3>Update/Delete SharePoint List Items</h3>
             </div>
             <div data-role="main" class="ui-content">
                <div >
                   <input id="EmployeeId"   placeholder="EmployeeId"  />
                </div>
                <div></br></div>
                <div >
                   <button id="UpdateItem" type="submit" >Update</button>
                   <button id="DeleteItem"  type="submit" >Delete</button>
                </div>
             </div>
          </form>
       </div>
       <br>
       <div style="background-color: lightgrey" id="spGetListItems" />
       </div>

          `;
       this.getListData();
       this.AddEventListeners();
         }

       AddItem()
       {

            pnp.sp.web.lists.getByTitle('EmployeeList').items.add({
            EmployeeName : document.getElementById('EmployeeName')["value"],
            Experience : document.getElementById('Experience')["value"],
            Location:document.getElementById('Location')["value"]
           });
```

```
            alert("Record with Employee Name : "+ document.getElementById('EmployeeName')["value"] + " Added !");

        }

        UpdateItem()
        {

            var id = document.getElementById('EmployeeId')["value"];
            pnp.sp.web.lists.getByTitle("EmployeeList").items.getById(id).update({
             EmployeeName : document.getElementById('EmployeeName')["value"],
             Experience : document.getElementById('Experience')["value"],
             Location:document.getElementById('Location')["value"]
          });
         alert("Record with Employee Name : "+ document.getElementById('EmployeeName')["value"] + " Updated !");
        }

        DeleteItem()
        {

pnp.sp.web.lists.getByTitle("EmployeeList").items.getById(document.getElementById('Empl
oyeeId')["value"]).delete();
            alert("Record with Employee ID : "+ document.getElementById('EmployeeId')["value"] + " Deleted !");
        }

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

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

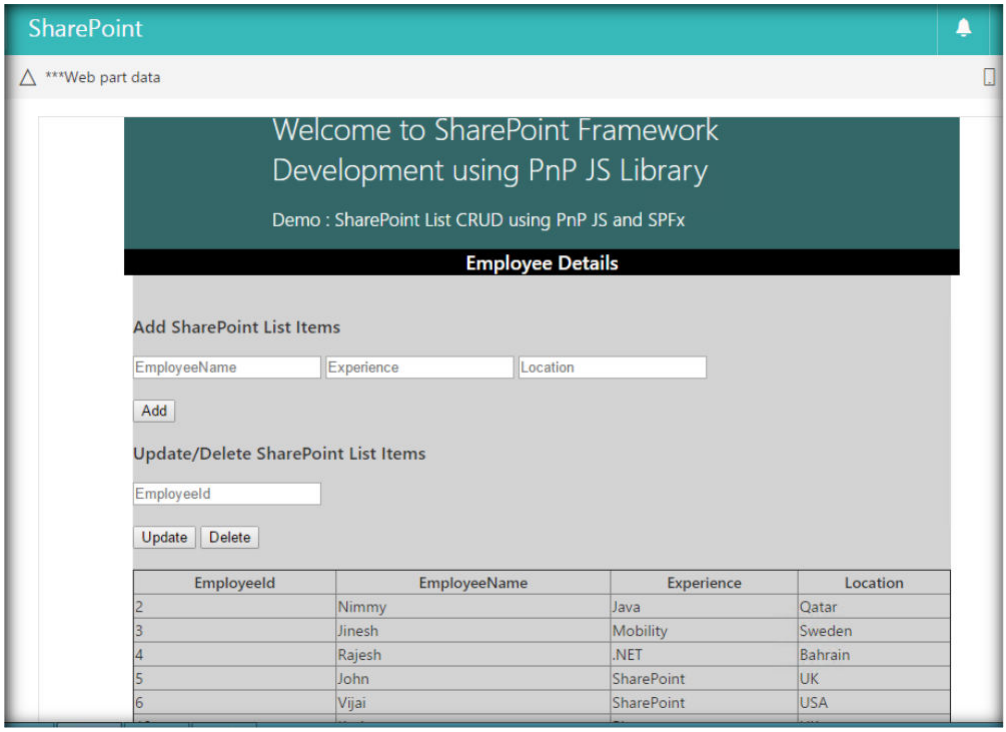
Test the Web part in SharePoint Online

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.

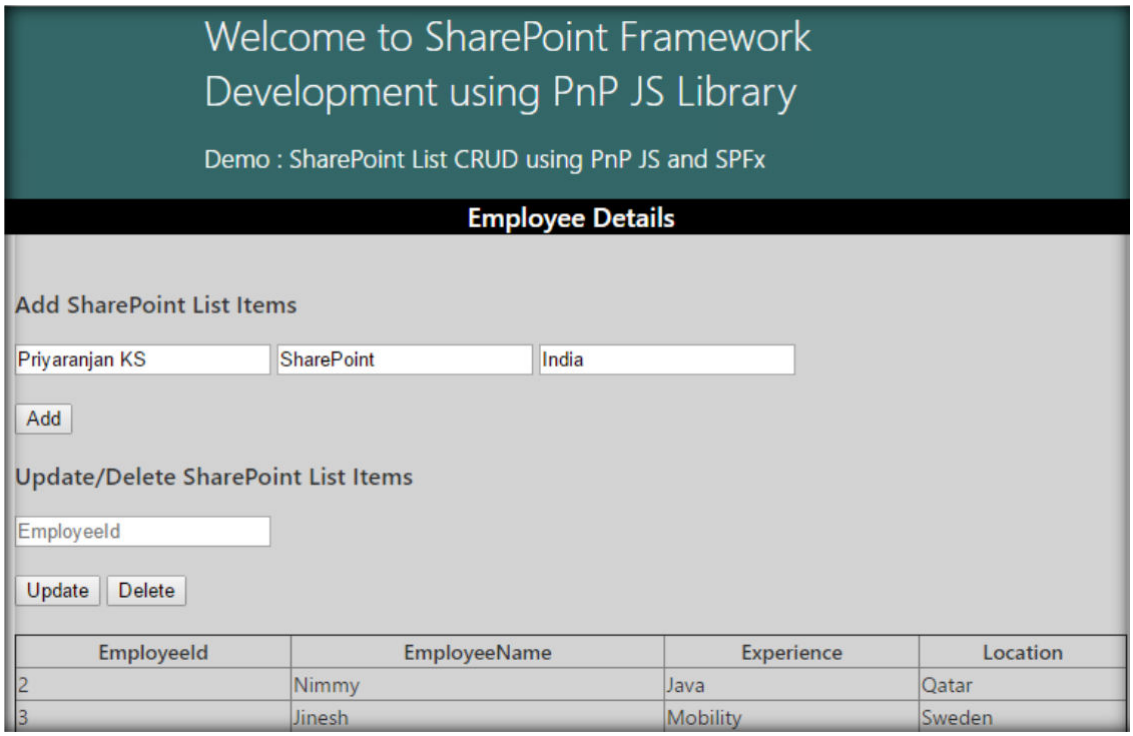


The UI will look like below:

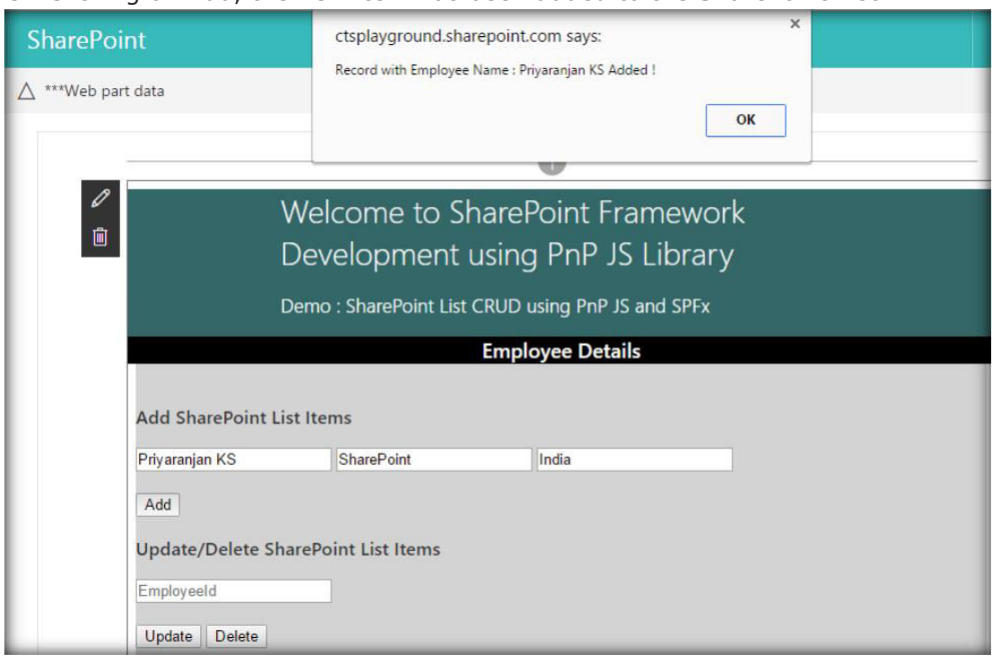


Add Item

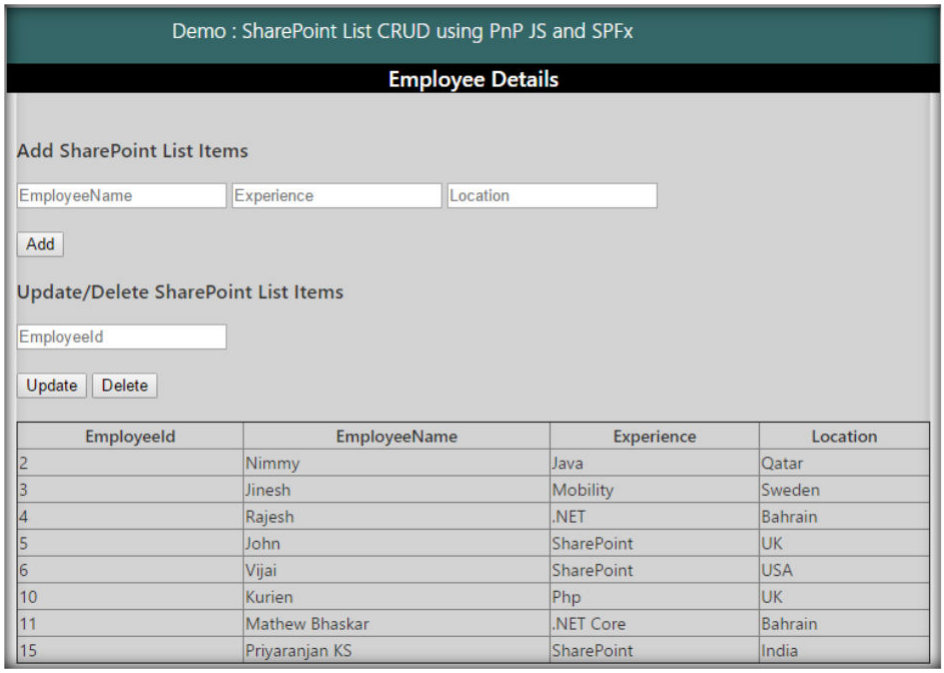
We can input the details and save it to the list by clicking on Add which will create a new list item using the PnP Add method



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

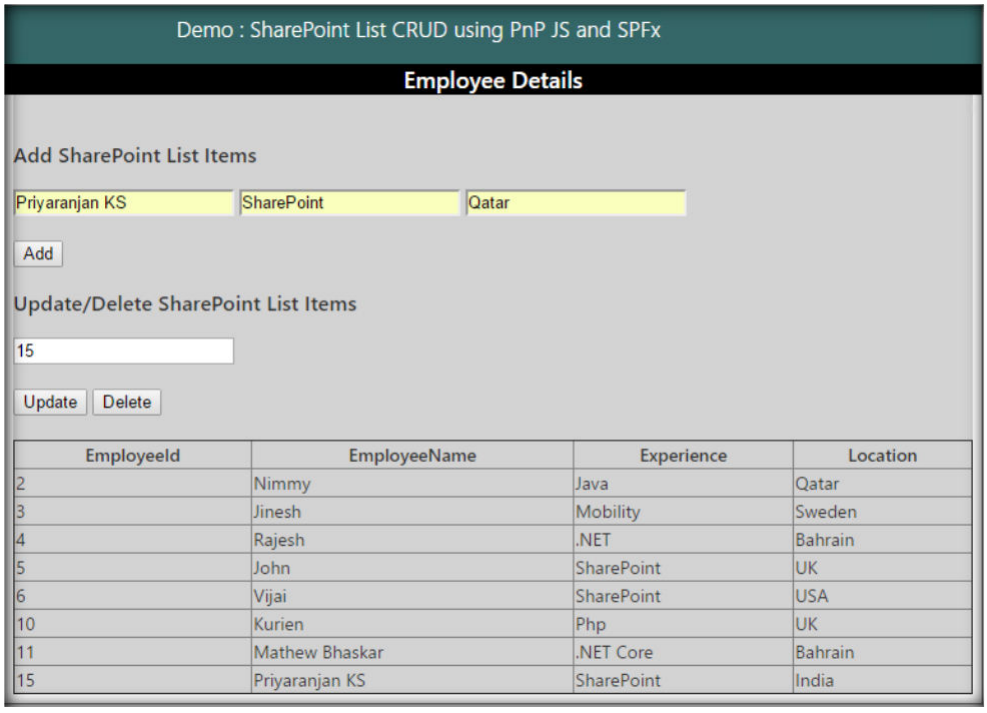


The table has listed the new item at the bottom of the list.

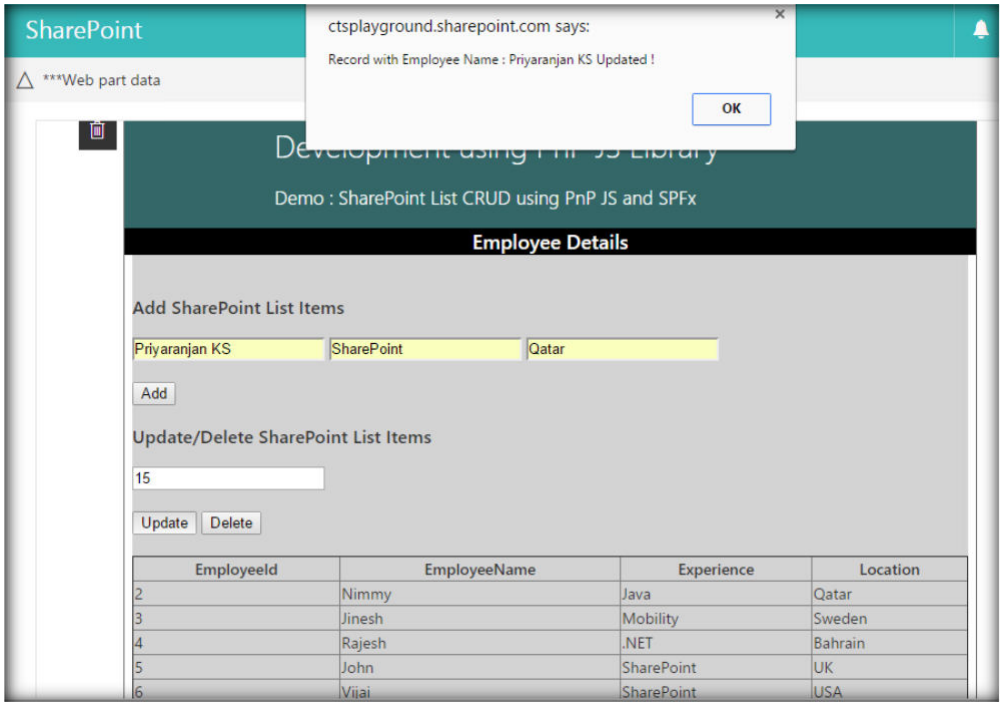


Update Item

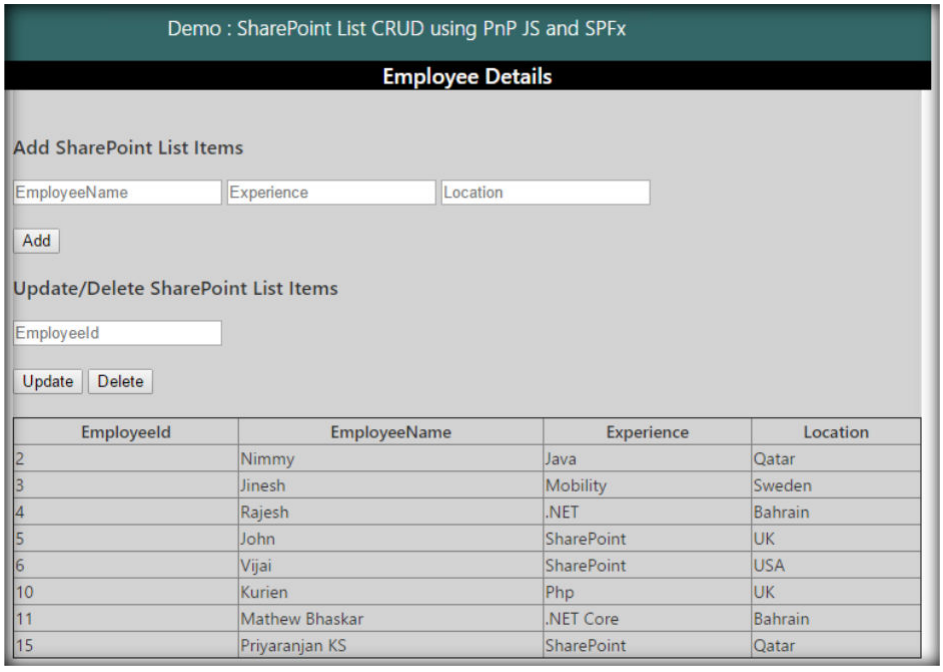
Similarly, we can update the existing list item by adding the data and providing the list item id which will be used to pick the item from the list and update it.



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

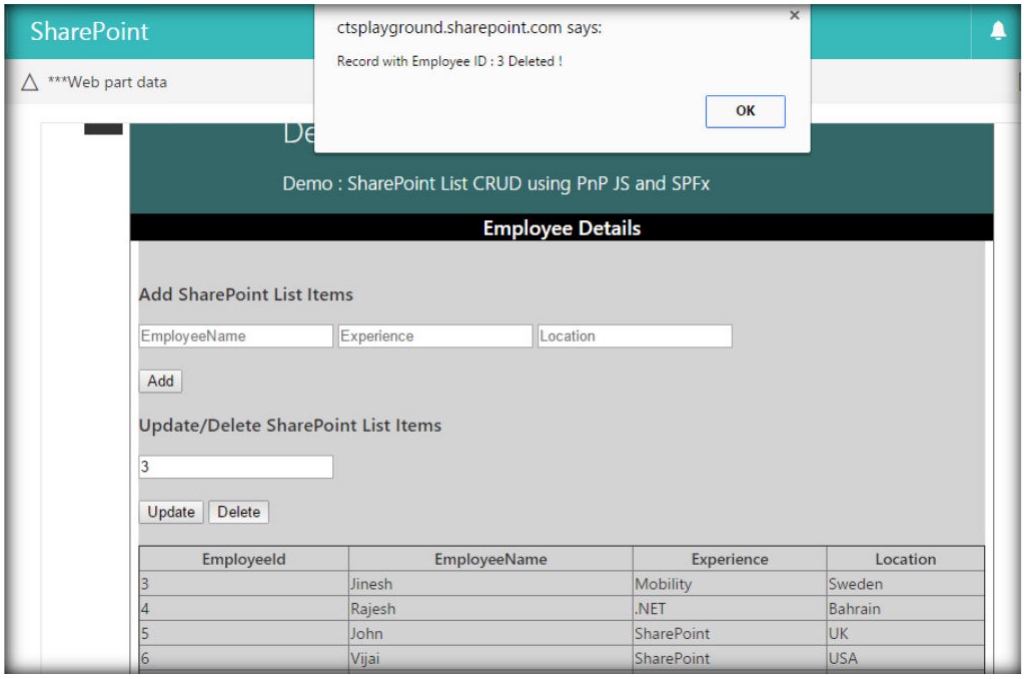


The location column value of the list item has been changed and reflected in the table as shown below.



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 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')
);
```

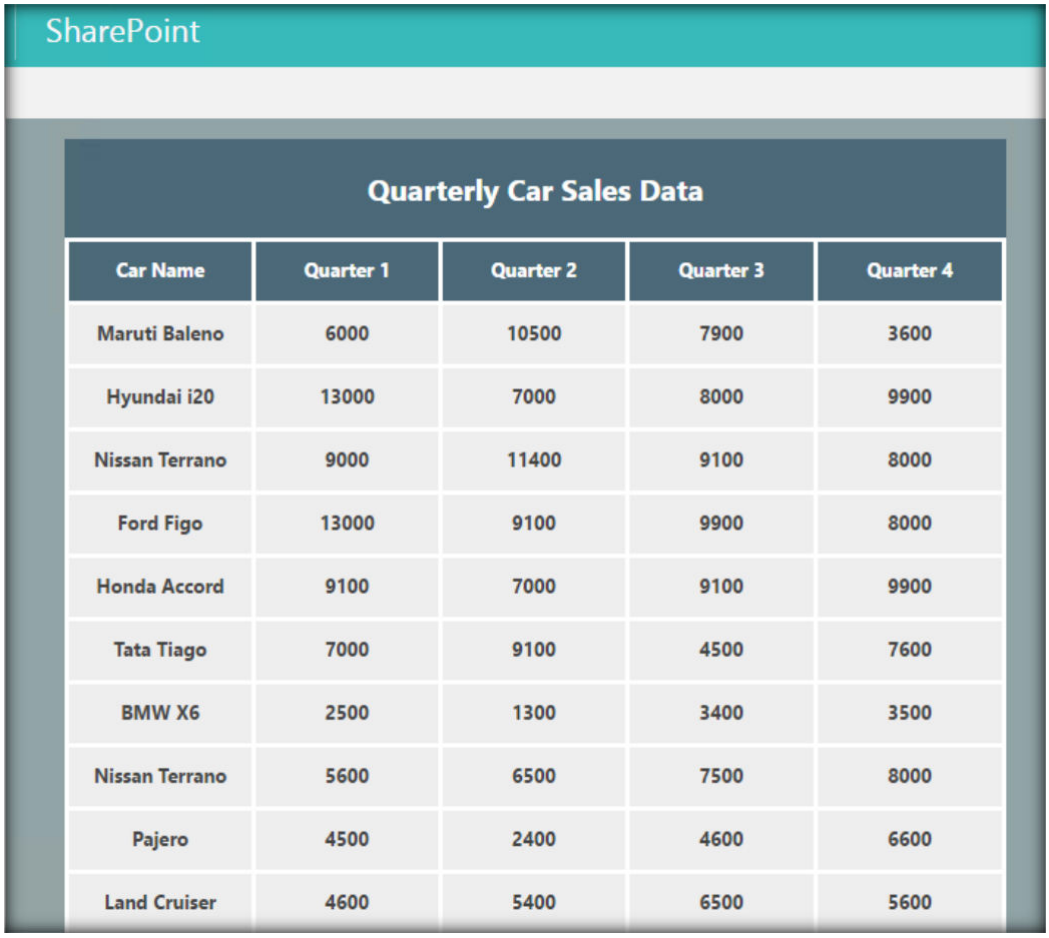
This will display the message at root div.In our case we will be displaying the data retrieved from 'ProductSales' list in the div named 'CarSalesData'.

```
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.



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.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/react.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/react-dom.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-standalone/6.24.0/babel.min.js"></script>
<div id="CarSalesData"></div>
<script type="text/babel">
   var tableStyle  =    {
          display: "table",
          marginLeft : "40px"
      }
   var panelStyle  =    {
         background: "#91A4A7"
      }

   var divStyle = {
    background: "#eee",
    padding: "20px",
    margin: "20px"
   };

   var headerCaptionStyle = {
    background: "#4B6978",
     display: "table-cell",
          border: "solid",
          textAlign : "center",
          width : "500px",
          height : "30px",
          paddingTop : "3px",
          color : "white",
          marginLeft : "80px",
          display : "block"
   };

   var headerStyle = {
    background: "#4B6978",
     display: "table-cell",
          border: "solid",
          textAlign : "center",
          width : "100px",
          height : "30px",
          paddingTop : "10px",
          color : "white"
   };

   var tableCaptionStyle = {
   background: "#c6e2ff",
     display: "block",
     fontSize : "20px",
     fontWeight: "bold",
          border: "solid",
          textAlign : "center",
```

```
          width : "650px",
          height : "30px",
          paddingTop : "3px",
          borderRadius: "25px",
          marginLeft : "30px",
          marginTop : "20px"
    }


    var rowCaptionStyle = {
     width : "600px",
     display :  "table-caption",
     background: "#4B6978",
     textAlign : "center",
     padding: "20px",
      fontSize : "20px",
     fontWeight :"bold",
     color : "white"
    };

    var rowStyle = {
     display :  "table-row",
     background: "#eee",
     padding: "20px",
     margin: "20px",
     fontWeight :"bold"
    };

    var CellStyle  =    {
          display: "table-cell",
          border: "solid",
           borderColor : "white",
          textAlign : "center",
          width : "100px",
          height : "30px",
          paddingTop : "10px"

      }

      class ReactSP extends React.Component{
         debugger;
          constructor(){
              super();
                this.state = {
        items: [
          {
            "CarName": "",
            "Quarter1": "",
            "Quarter2":"",
                   "Quarter3": "",
                   "Quarter4":""
                 }
              ]
             };

            }

                 componentDidMount() {
                     debugger;
                    this.RetrieveSPData();
                 }
```

```
112.
113.                RetrieveSPData(){
114.                    var reactHandler = this;
115.
116.                    var spRequest = new XMLHttpRequest();
117.                    spRequest.open('GET',
   "/sites/playground/_api/web/lists/getbytitle('ProductSales')/items",true);
118.                    spRequest.setRequestHeader("Accept","application/json");
119.
120.                    spRequest.onreadystatechange = function(){
121.
122.                        if (spRequest.readyState === 4 && spRequest.status === 200){
123.                            var result = JSON.parse(spRequest.responseText);
124.
125.                            reactHandler.setState({
126.                                items: result.value
127.                            });
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.
```

```
            );
          }

          }

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

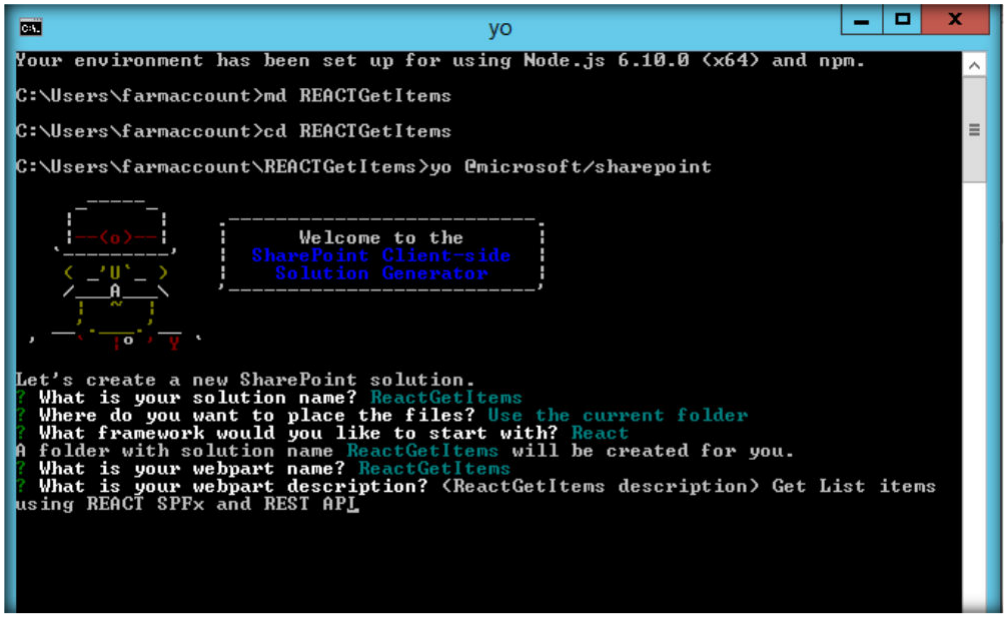
Create SPFx Webpart to retrieve SharePoint List Items using REACT & REST API

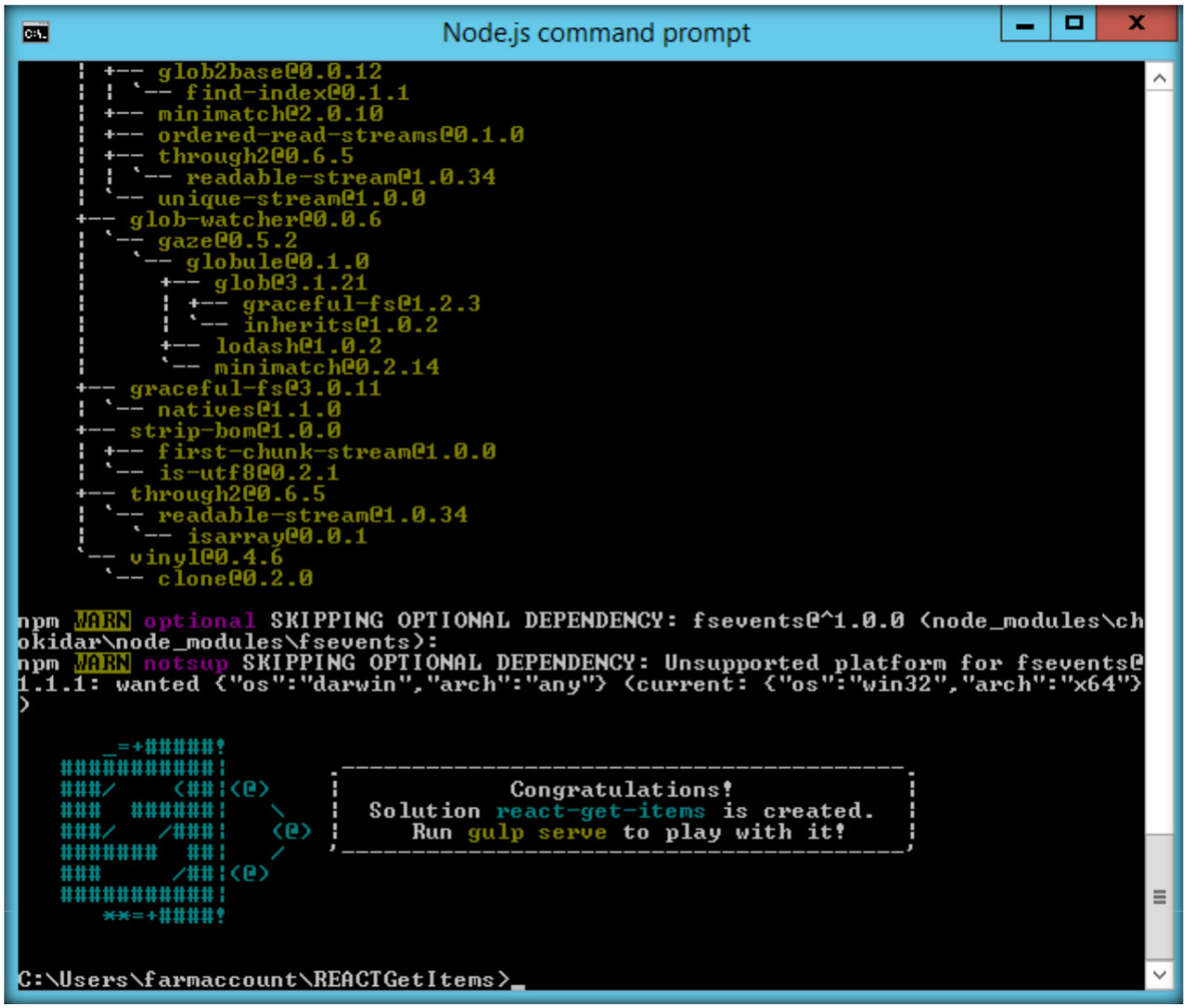
In this section, we will see how to create a client webpart using SharePoint Framework and React JS that displays SharePoint List data retrieved using REST API. The major project files used in this solution has been zipped and uploaded at Microsoft TechNet Gallery. Feel free to download it.

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*'





Edit the web part

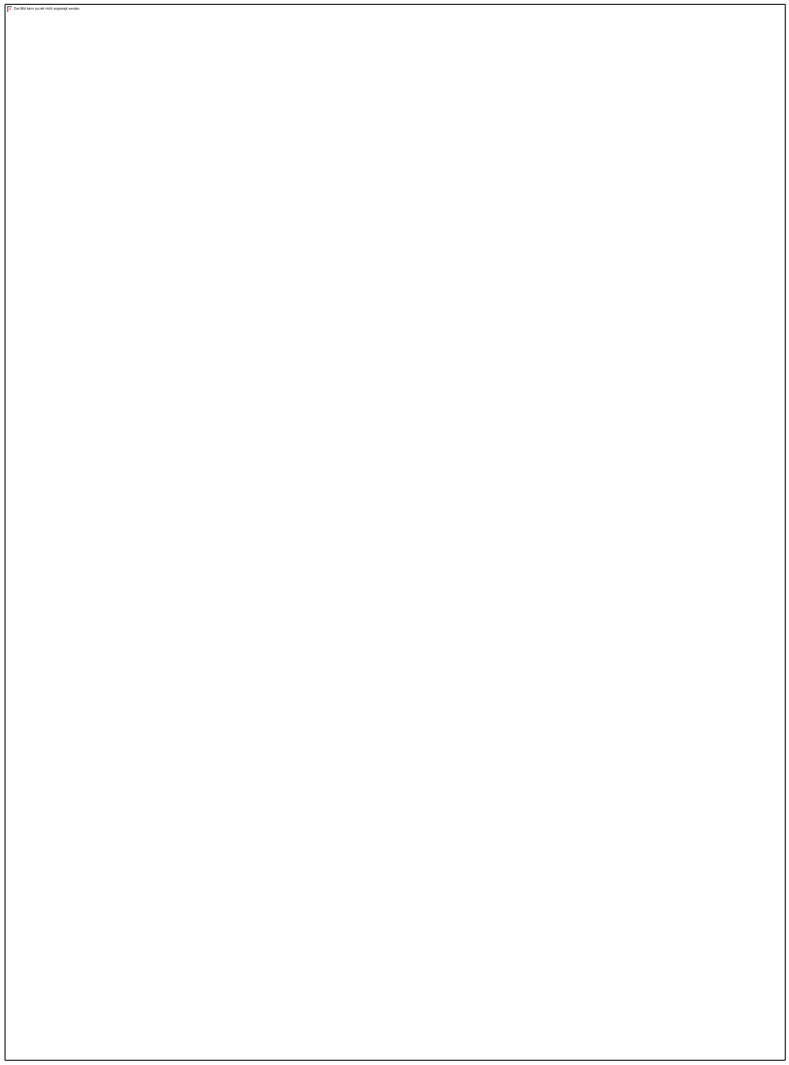
Run Code . to create the scaffolding and open the project in Visual Studio Code. We will be required jQuery to make ajax rest api calls. So let's install jQuery using NPM as shown below.

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\ch
okidar\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\ch
okidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@
```

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 ;
```

```
    background:  #4B6978 ;
    text-align :  center ;
    padding:  20px ;
     font-size :  20px ;
    font-weight : bold ;
    color :  white ;
   }

  .rowStyle{
    display :   table-row ;
    background:  #eee ;
    padding:  20px ;
    margin:  20px ;
    font-weight : bold ;
   }

   .CellStyle{
          display:  table-cell ;
          border:  solid ;
           border-color :  white ;
          text-align :  center ;
          width :  100px ;
          height :  30px ;
          padding-top :  10px ;
      }
```

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.

```
public constructor(props: IReactGetItemsProps, state: IReactGetItemsState){
    super(props);
    this.state = {
      items: [
        {
          "EmployeeName": "",
          "EmployeeId": "",
          "Experience":"",
          "Location":""
        }
      ]
    };
  }
```

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} >
```

```
          <br></br>

          <br></br>
          <div className={styles.tableCaptionStyle} > Demo : Retrieve SharePoint List Items using SPFx , REST API  & React JS  </div>
          <br></br>
            <div className={styles.headerCaptionStyle} > Employee Details</div>
          <div className={styles.tableStyle} >

            <div className={styles.headerStyle} >
              <div className={styles.CellStyle}>Employee Name</div>
              <div className={styles.CellStyle}>Employee Id </div>
              <div className={styles.CellStyle}>Experience</div>
                <div className={styles.CellStyle}>Location</div>

            </div>

              {this.state.items.map(function(item,key){

                return (<div className={styles.rowStyle} key={key}>
                    <div className={styles.CellStyle}>{item.EmployeeName}</div>
                    <div className={styles.CellStyle}>{item.EmployeeId}</div>
                      <div className={styles.CellStyle}>{item.Experience}</div>
                            <div className={styles.CellStyle}>{item.Location}</div>

                  </div>);
              })}

          </div>
        </div>
    );
  }
}
```

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.