Sharepoint Journey

Become Awesome in SharePoint

SharePoint Framework

Table of Contents

38. Summary

SharePoint Framework Development - Setup Dev Env

Sharepoint Journey

Become Awesome in SharePoint

A Brief History of SharePoint Development Model

In the older SharePoint Days, we were mostly using Server Side Object Model to customize SharePoint. Since the code ran within the main W3WP thread, deployments resulted in brief downtime and increased the risk that came along with the untested code. Moreover, Server Side code was not feasible with Office 365 based SharePoint Online implementations. To overcome the sharing of the SharePoint thread by the code, Sandboxed solutions were introduced that executed in its own SPUCWorkerProcess. However, they had their own limitations and are in a deprecated stage now.

Later, SharePoint Add-ins were introduced which leveraged the client side development approach and helped developers deploy solutions to Office 365 SharePoint Online as well as On-Premise. In addition to it, the Add-ins were installed in the app web which was separate from the main SharePoint Host web creating the required isolation. However, the add-ins had the IFrame and App web dependency which forced developers to use workarounds for the limitations that came along with it (Cross-Domain Library for instance). To overcome some of the downsides of Add-ins, developers embraced a mix of Add-in + Client Side Content Editor web part development which exposed the page code and anyone with 'Add and Customize Pages' permission could break the existing code from the browser.

Though not a replacement for any of the previous development models, with the SharePoint Framework model, Microsoft is providing an

advanced full client side development model which can be used on any platform and with any JavaScript framework like React, Angular, Knockout, Handlebar etc.: to build SharePoint Solutions. It aims at building of client-side customization much easier with a streamlined deployment process. SPFx solutions are rendered within the current user context in the current page DOM (not in an IFRAME) which results in faster performance. Moreover, the strong dependency with Visual Studio has been removed and we can use modern web technologies and tools in our preferred development environment to build SPFx solutions. We will go over these development tools in the upcoming sections.

This is the first article of the SharePoint Framework series aimed to give the users a jumpstart in the SharePoint Framework Development. In this article, we will see how to set up the environment for getting started with the development using SharePoint Framework. 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)

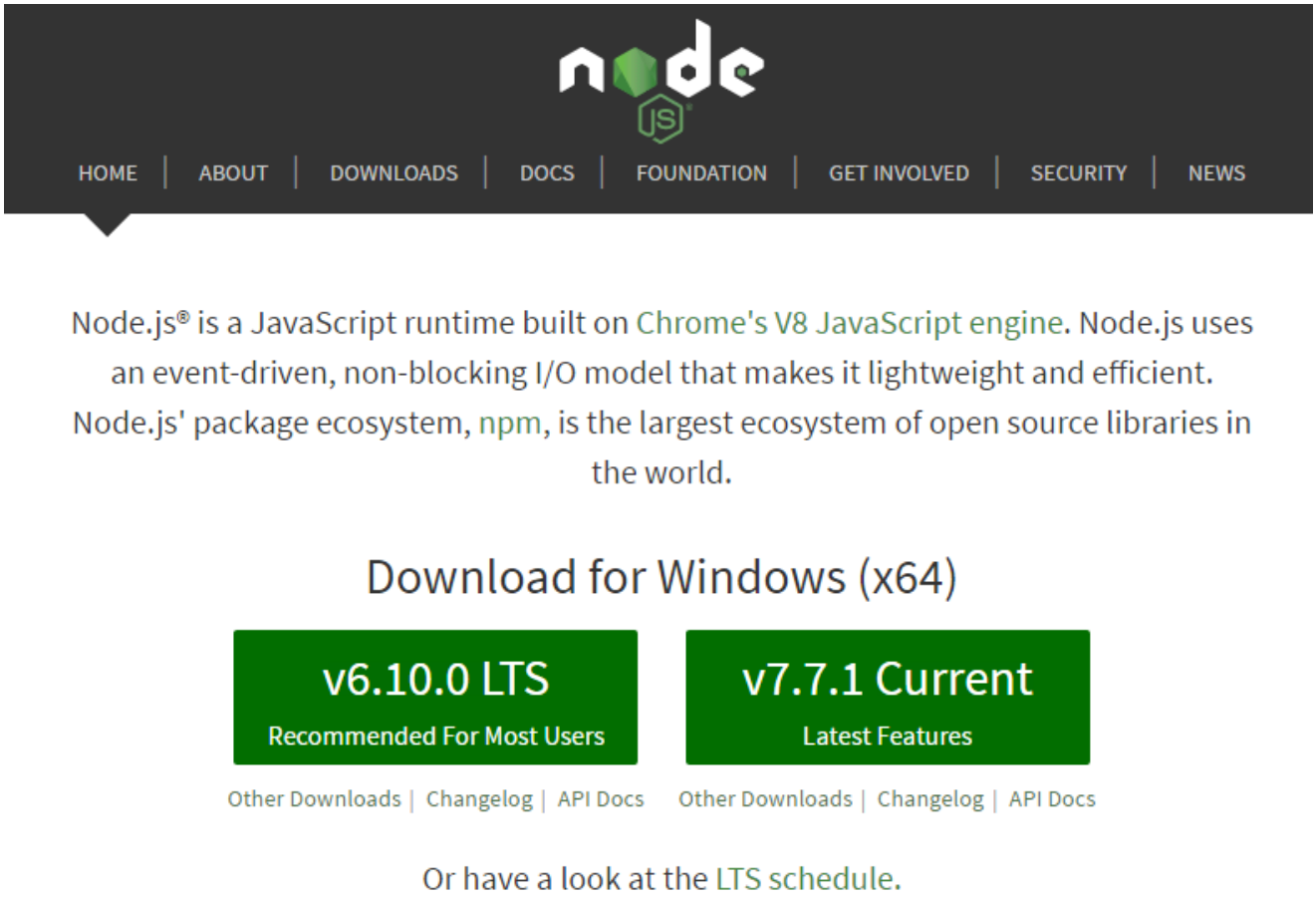
Install Node JS

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It is a cross-platform runtime environment for hosting and serving JavaScript code. Node.js' package manager, npm, is a command-line package manager that's similar in concept to the NuGet Package Manager found in Visual Studio. npm package consists of one or more reusable JavaScript code files called modules. SharePoint Framework consists of several npm packages that work together to help developers build client-side experiences in SharePoint, like: *@microsoft/sp-core-library, @microsoft/sp-webpart-base, @microsoft/sp-lodash-subset* etc.: So, We will be making use of npm to

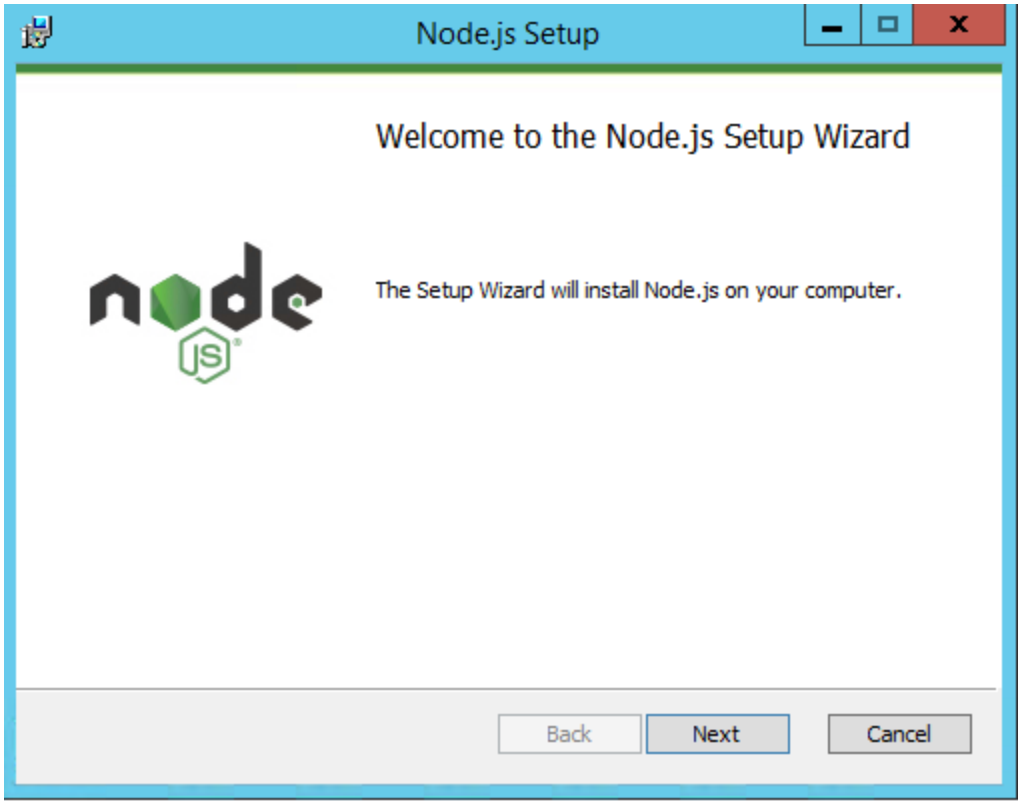
install the package and dependencies. We will be later using the 'import' command to use them within the SPFx project. To draw a comparison to the existing .Net Development model, Node.js is like IIS Express/ IIS and NPM is like NuGet.

As the first step, we will install NodeJS Long Term Support Version (LTS) which sets up the development environment and adds NPM. We can

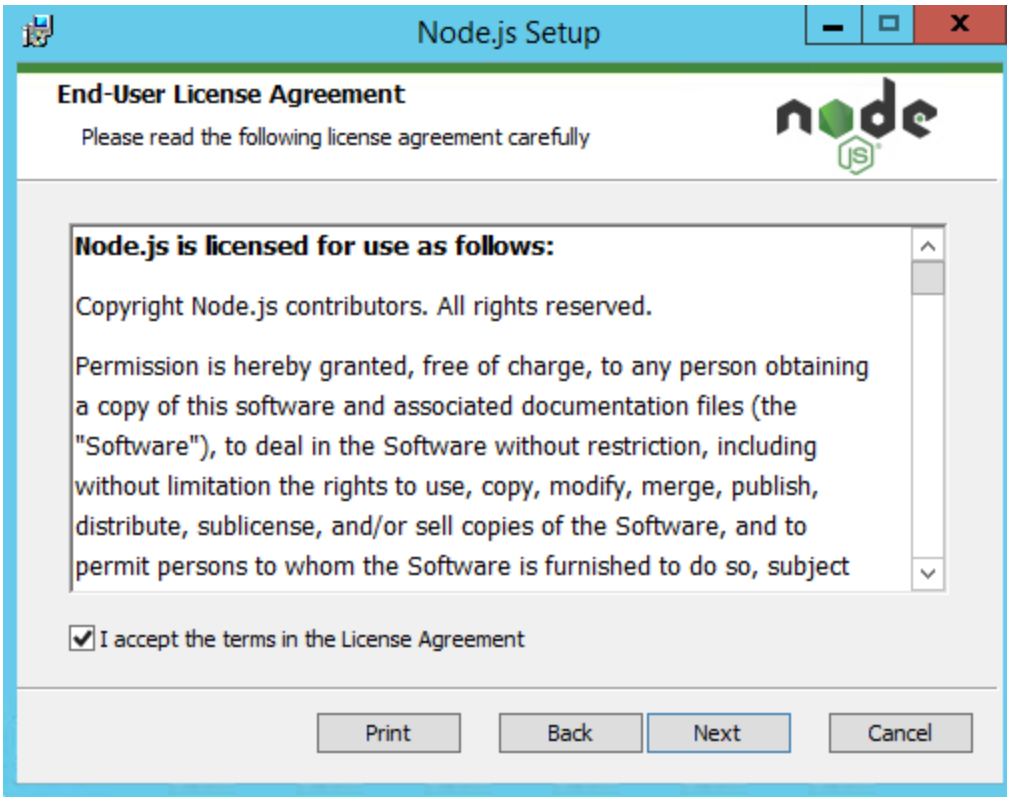
install Node JS from this link.



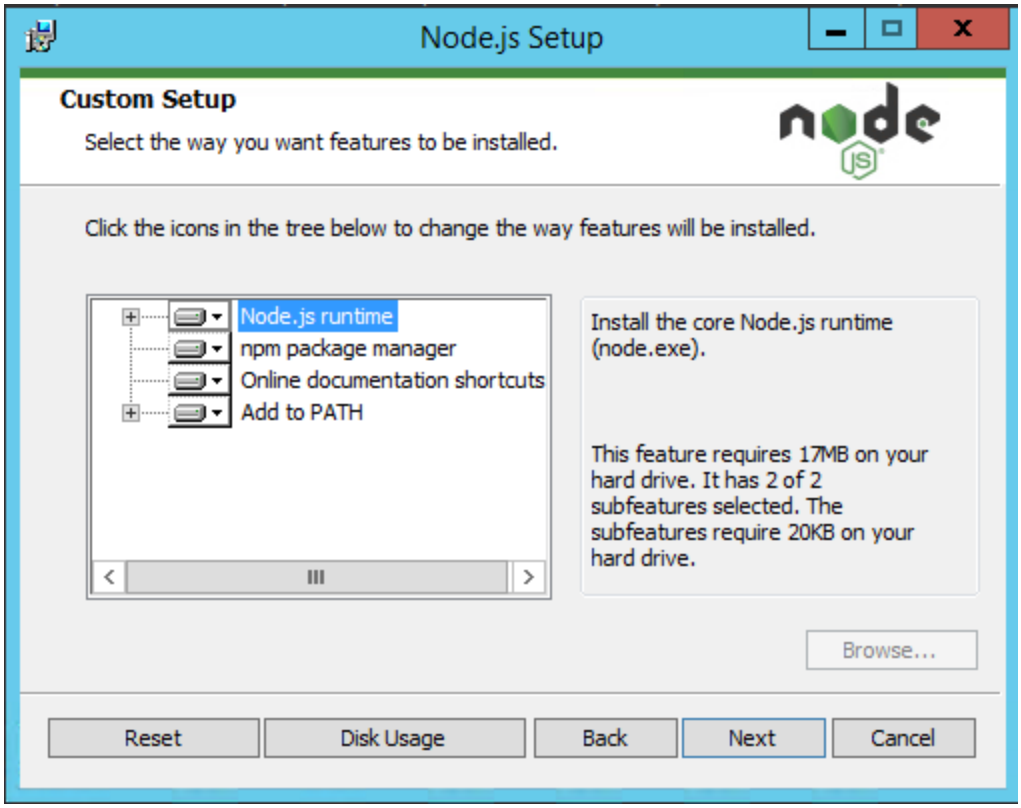
Once we have downloaded the LTS version, run the executable file and proceed.



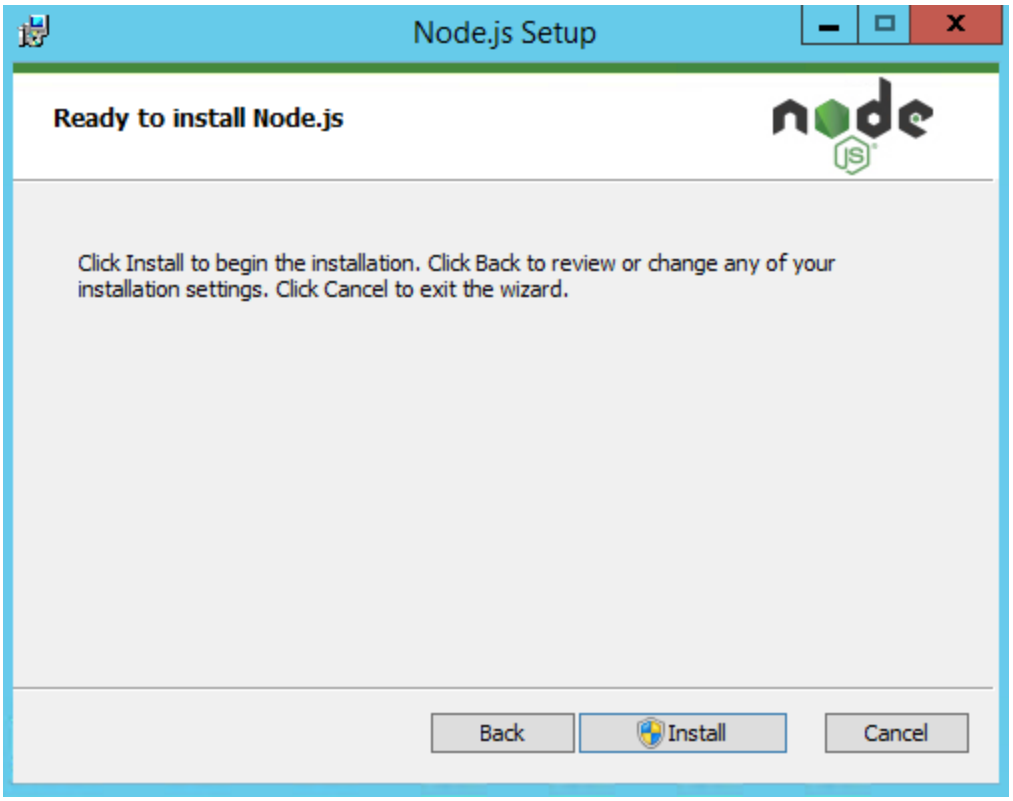
Accept the license agreement and click Next.



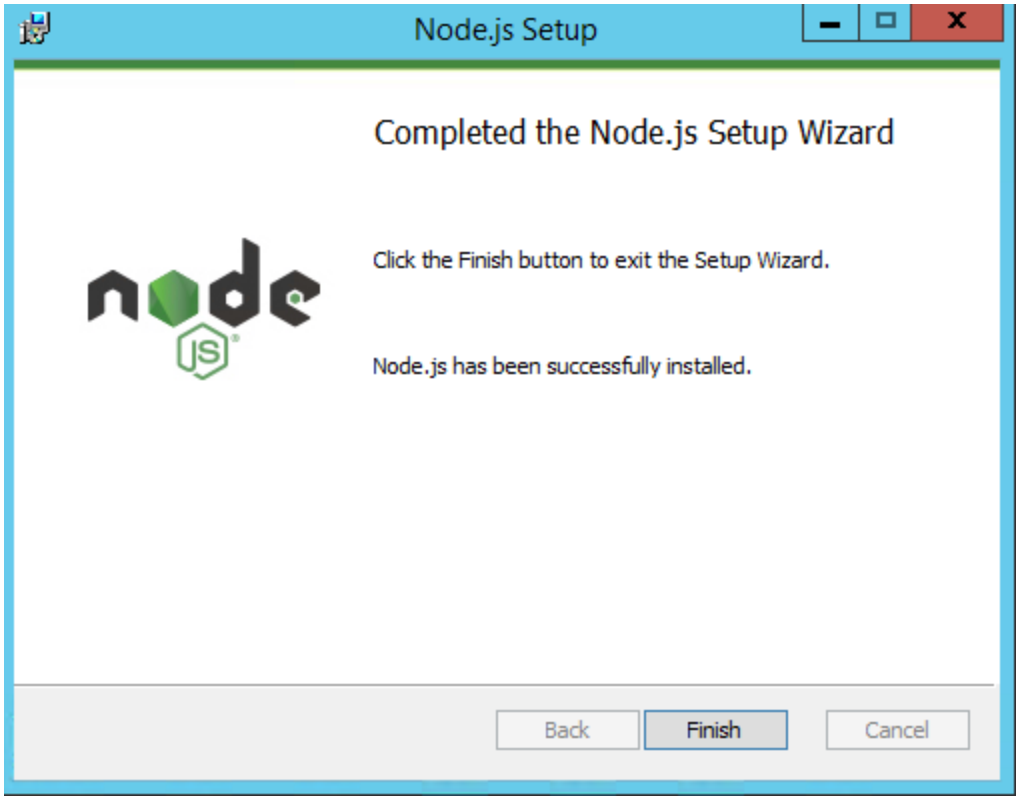
We will select Node.js runtime installation.



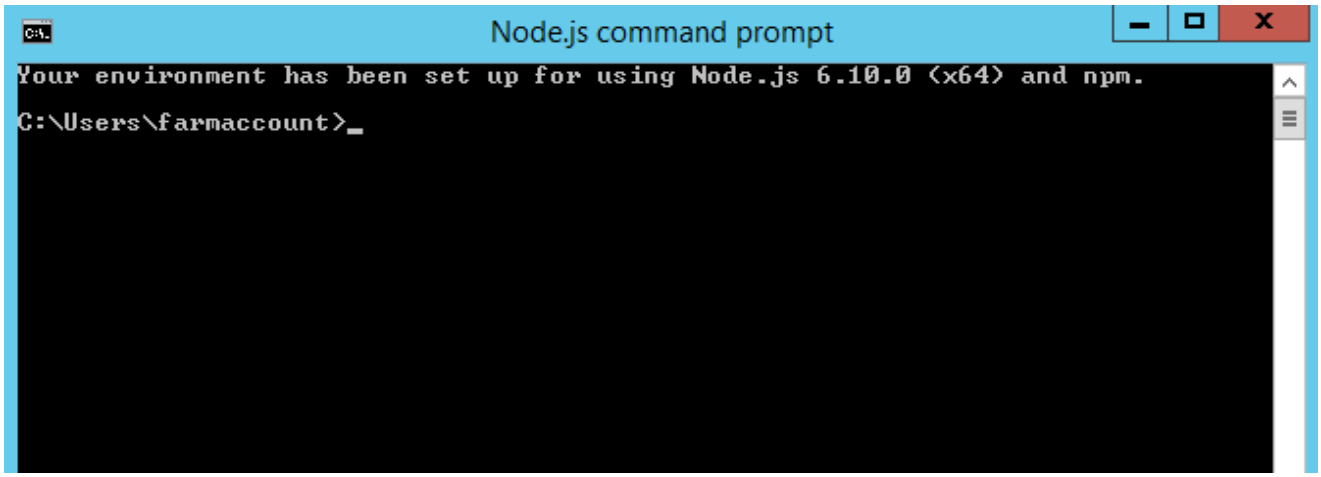
Click on Install to start the installation of Node.js.



Finally, we are done with the installation of Node.js.



If we run the Node.js command prompt we will get the message as shown below. Thus we can confirm that Node.js has been successfully installed on 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.

Node.js command prompt

```
Your environment has been set up for using Node.js 6.10.0 (x64) and npm.

C:\Users\farmaccount>npm -v
3.10.10

C:\Users\farmaccount>_
```

Install Yeoman and Gulp

Yeoman is a scaffolding tool for modern web apps. When it comes to SharePoint it has a plugin named '*Yeoman SharePoint Generator*' to work with SharePoint Framework. It downloads and configures the required tool-chain components for the specified client-side project. It also creates the necessary project structure for the project based on the input we give like Project name and Framework to be used. To sum up, it provides with the common build tools, boilerplate code (standard default code), and test web site (SharePoint Workbench) to host the web parts for testing.

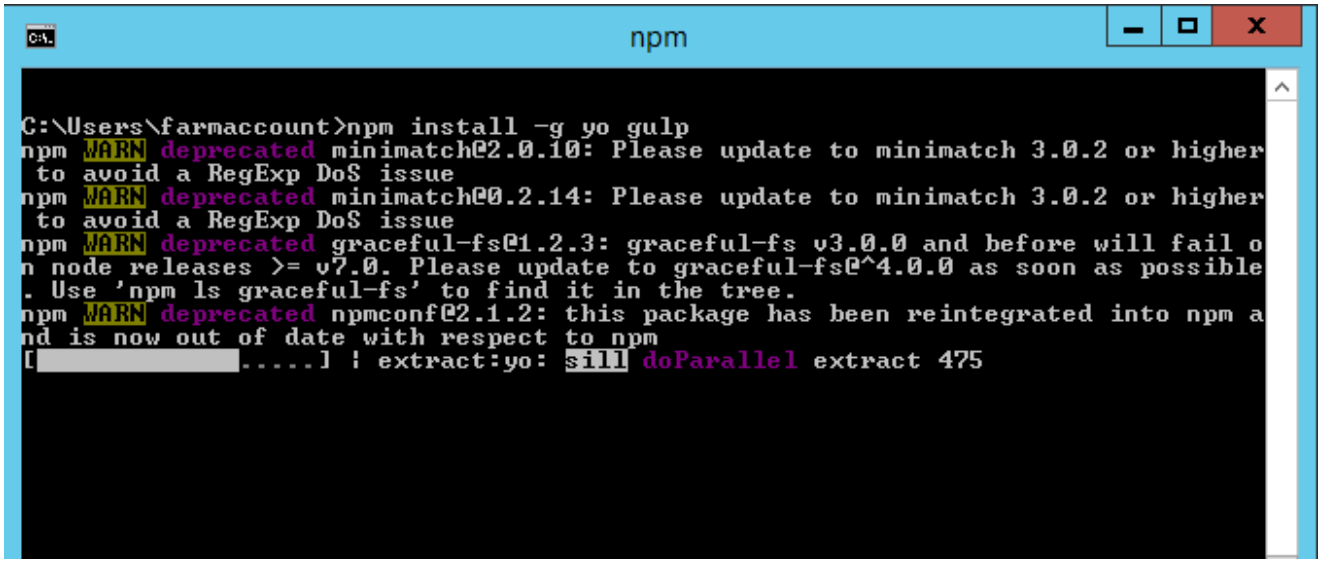
SharePoint Framework uses *Gulp* as its task runner. Gulp is a JavaScript task runner that helps us automate common tasks like refreshing our browser when we save a file, bundling libraries and CSS, Copying modified files to the output directory etc. To name a few, it consists of the following gulp tasks that we frequently use to work with the Client Side Project

- Build : Builds the client-side solution project.
- Bundle : Bundles the client-side solution project entry point and all its dependencies into a single JavaScript file.
- Serve : Serves the client-side solution project and assets from the local machine

To initiate different tasks, append the task name with the gulp command.Say, for instance if we want to compile and preview the web part in the SharePoint Workbench we use the *Gulp Serve*command. Each time the Gulp Server command is issued any changes in the code is recompiled and serves the client side solution in the SharePoint Workbench.

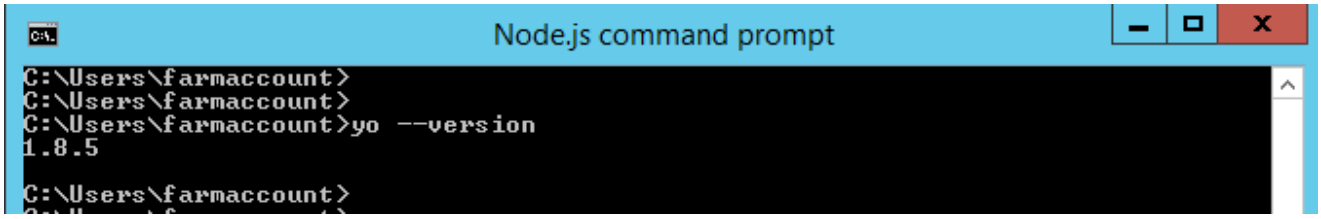
So, let's install Yeoman and Gulp simultaneously by running the below command:

npm install -g yo gulp



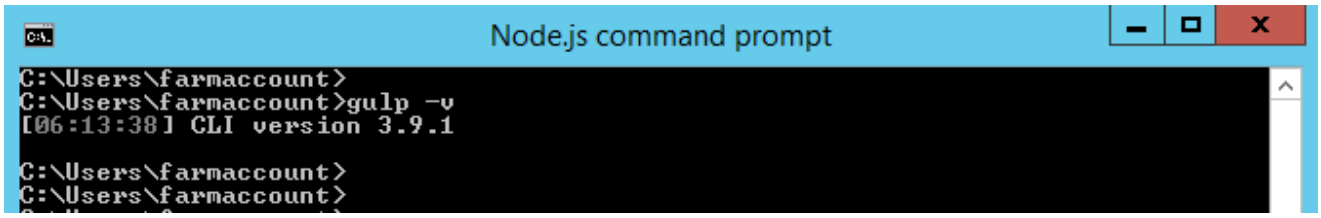
We can get the version of Yeoman using the command:

yo --version



Similarly, We can get the Gulp Version using the command:

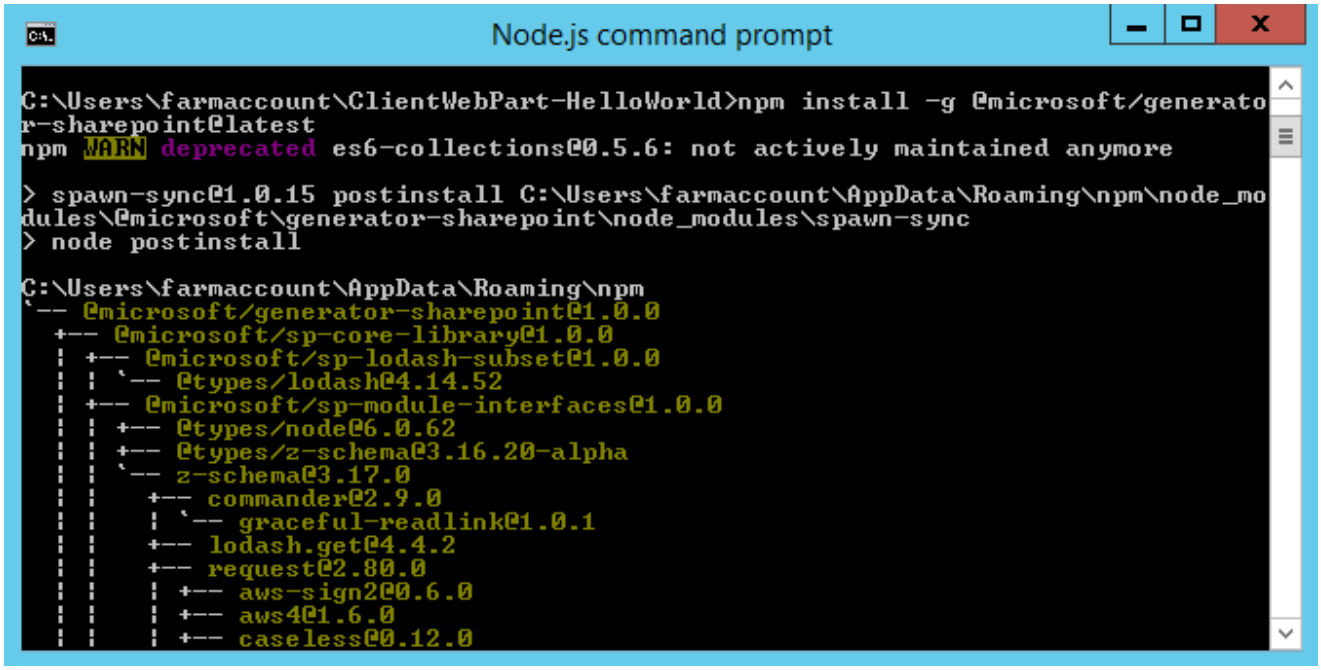
gulp –v



Install Yeoman SharePoint Generator

Once we have installed Yeoman, lets install The *Yeoman SharePoint Generator* which is the Yeoman Plugin for scaffolding SharePoint Framework client side 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



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

npm view @microsoft/generator-sharepoint version

Node.js command prompt

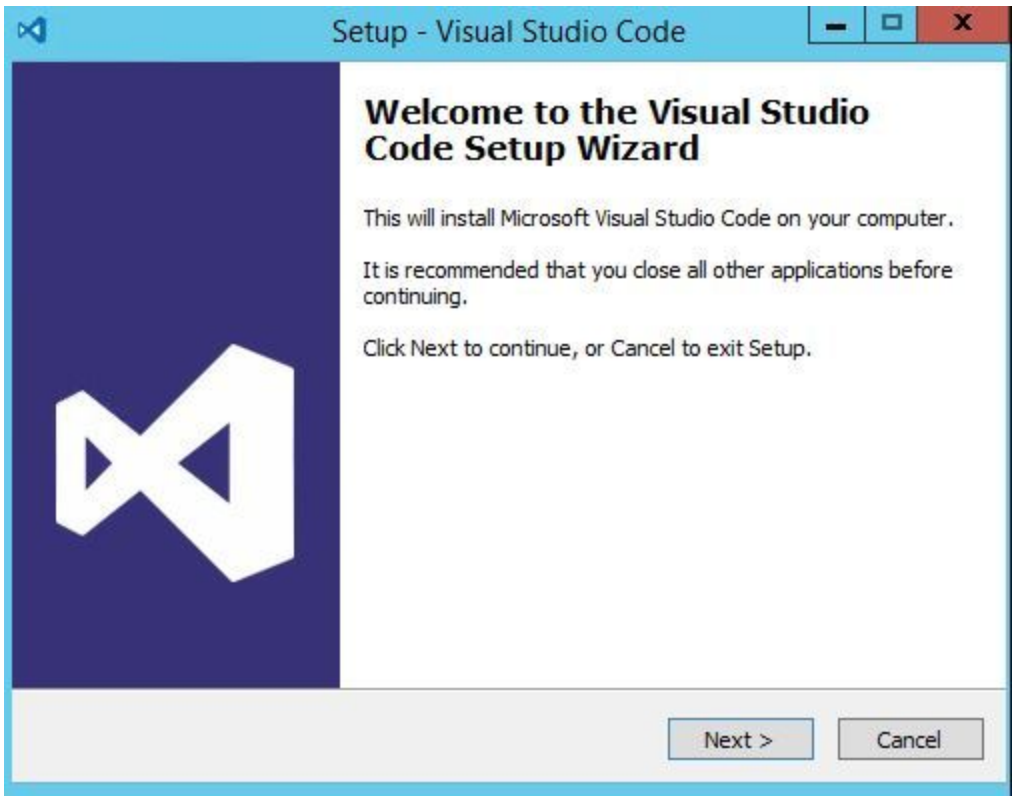
```
Your environment has been set up for using Node.js 6.10.0 (x64) and npm.

C:\Users\farmaccount>npm view @microsoft/generator-sharepoint version
1.0.0

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

Code Editor

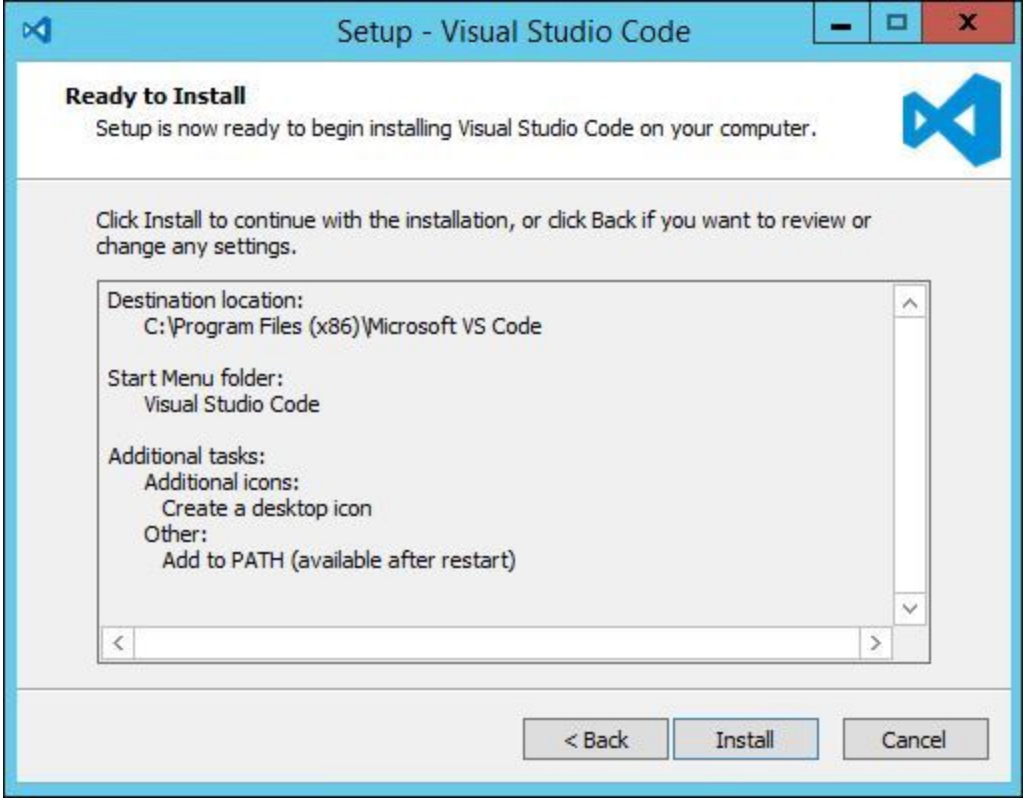
Next we need a code editor that will help us with code editing.Unlike old days where we relied heavily on Visual Studio, We can use any code editor or IDE that supports client-side development to build our web part, such as:

- Visual Studio Code
- Atom
- Webstorm

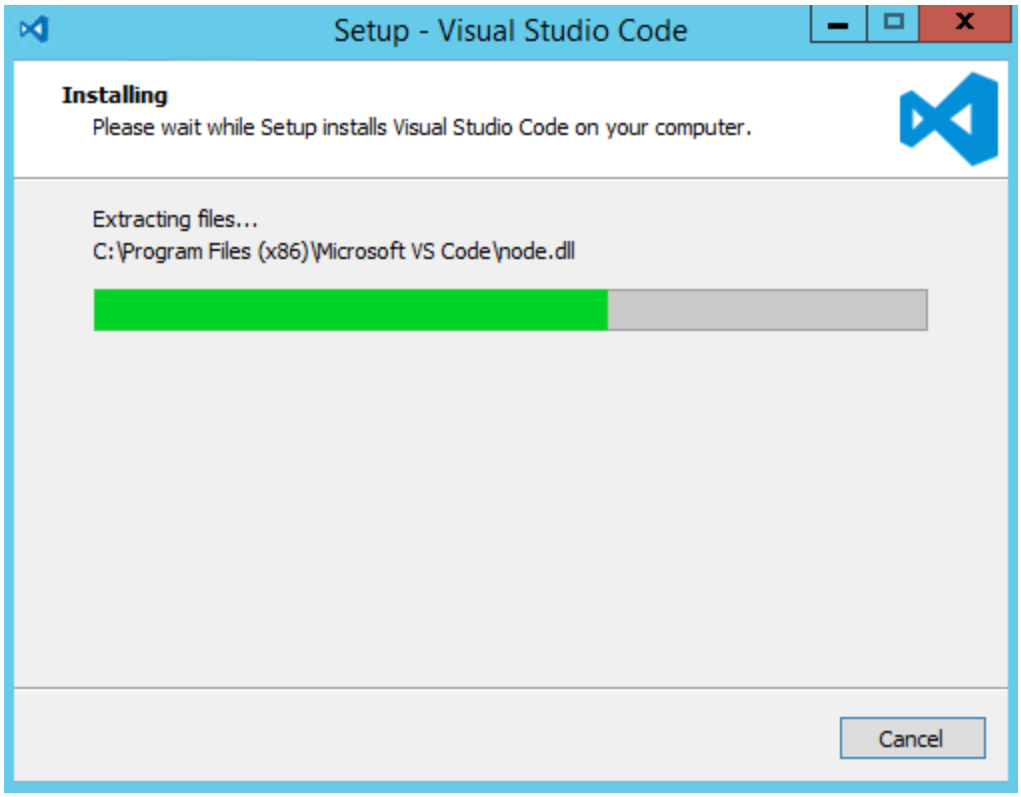
We will use Visual Studio Code in this walk-through which you can get it from here.



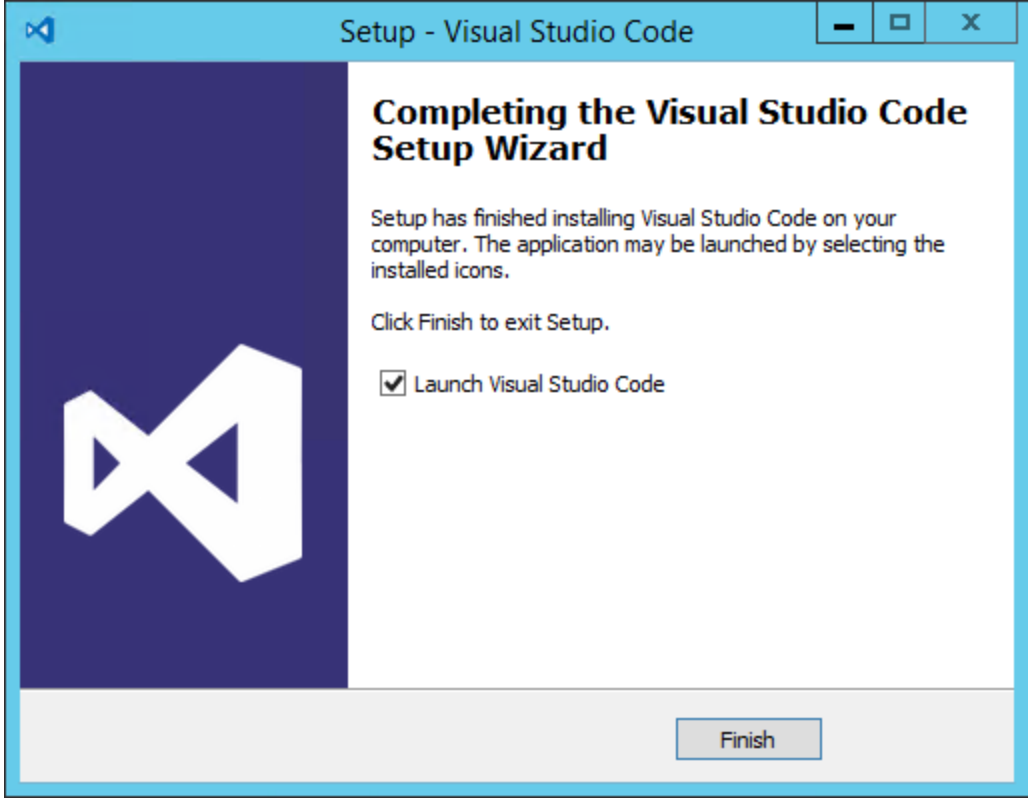
Once we have downloaded the exe proceed with the installation.



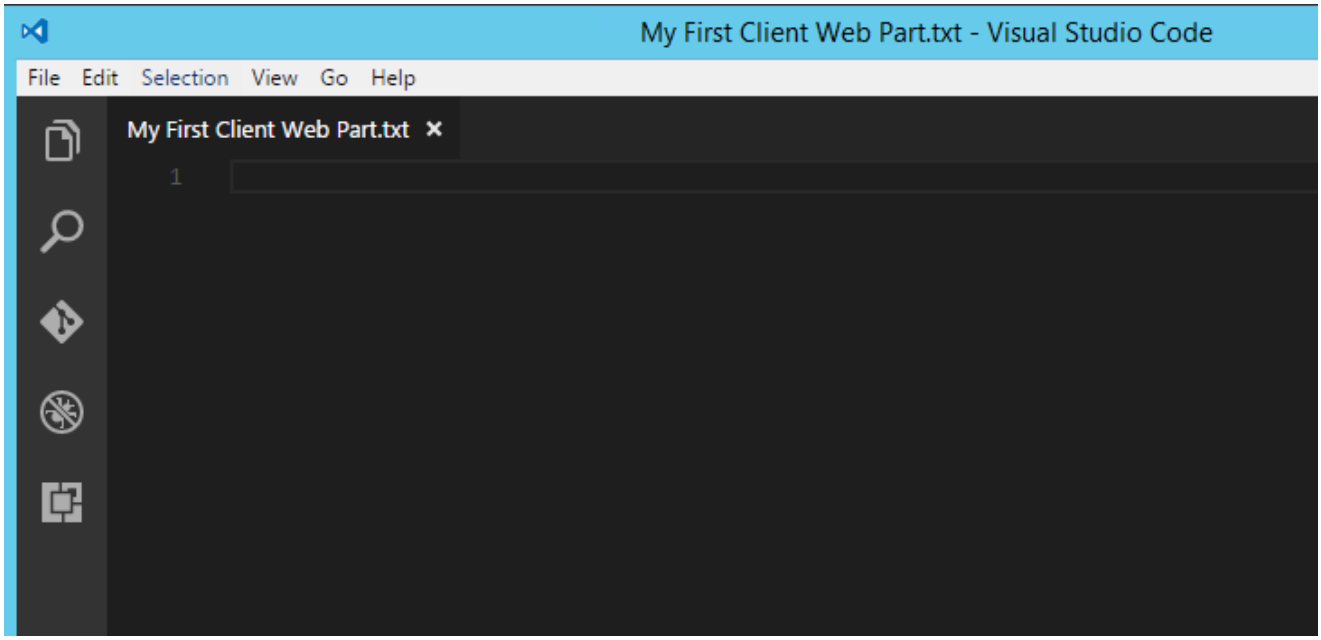
Click on Install to start the installation procedure.



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



Visual Studio Code:

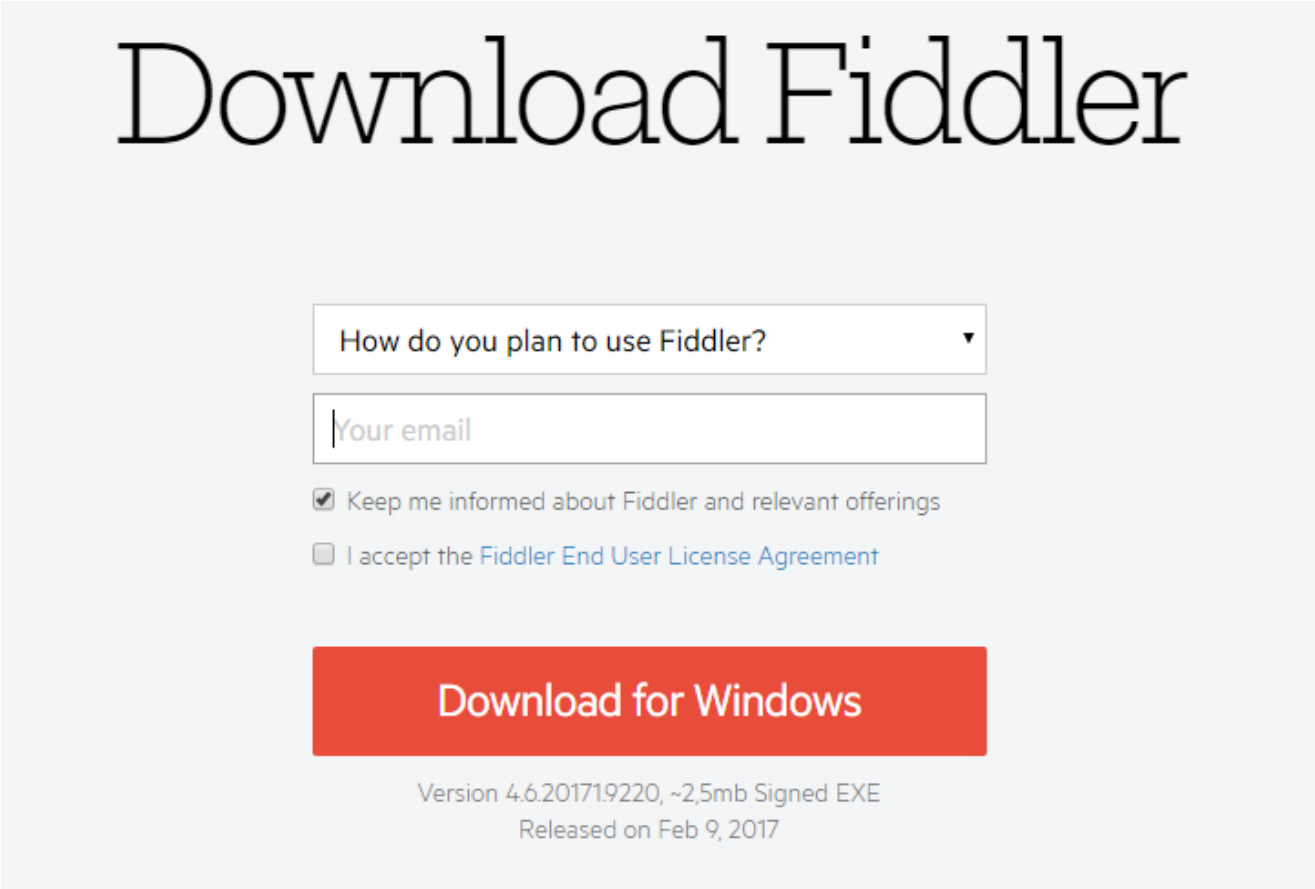


Additional Tools for Development and Debugging

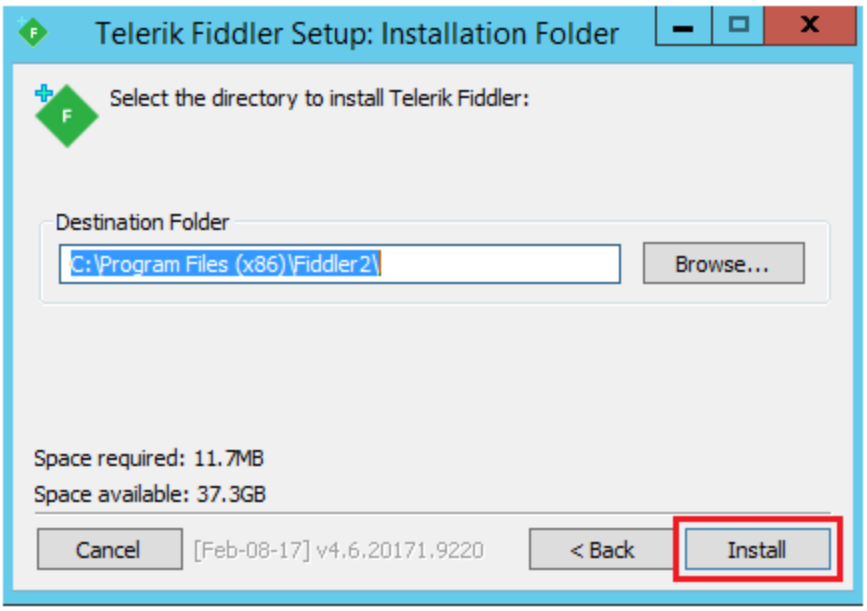
Once we start the development, we will have to debug or test the application. Fiddler and Postman can help us in this task.

Fiddler

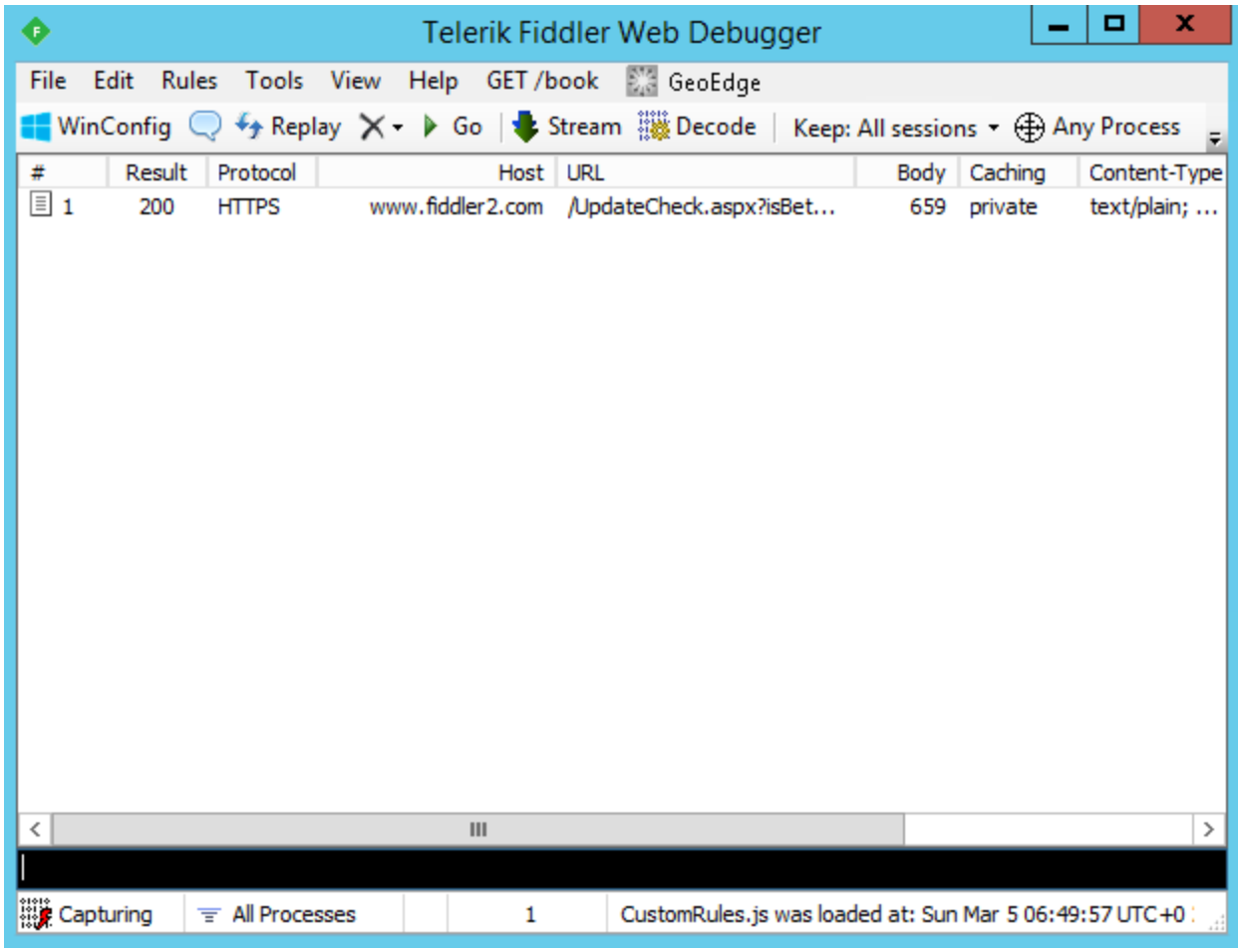
Fiddler is an HTTP debugging proxy server application. It captures HTTP and HTTPS traffic and logs it for the user to review. You can get fiddler from here.



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



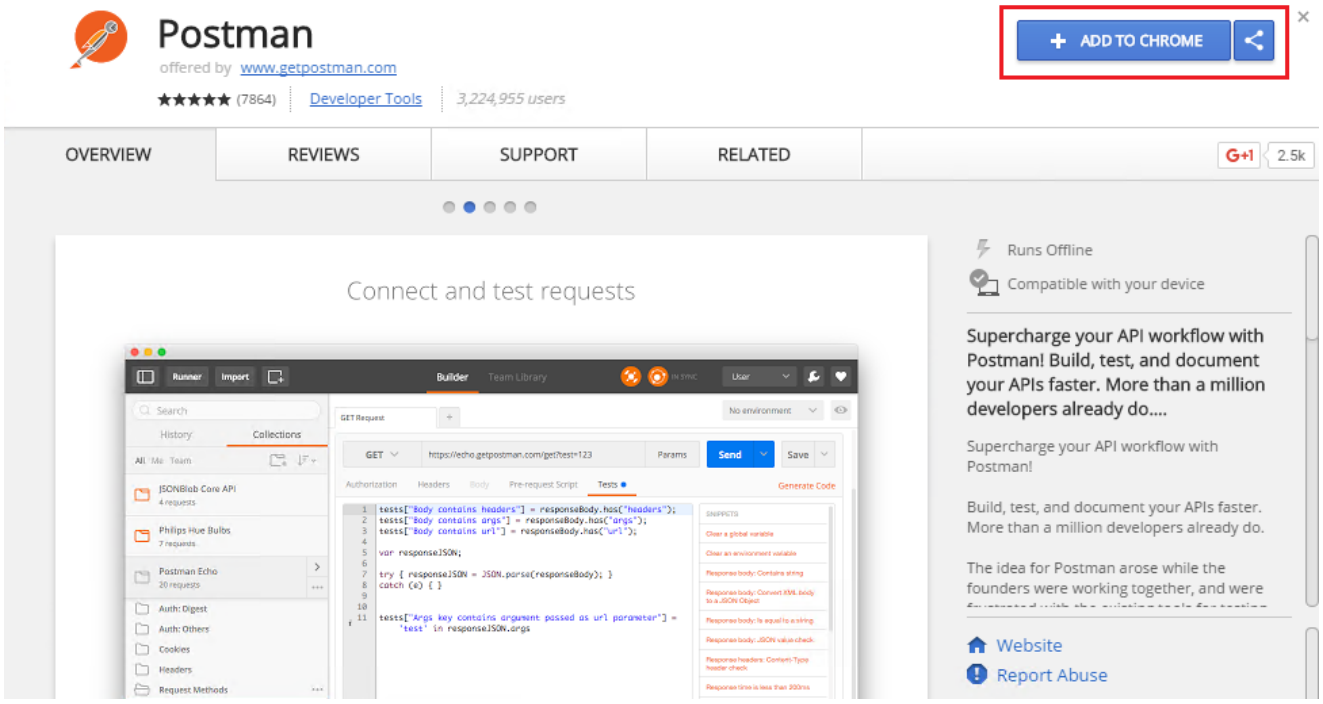
Using Fiddler we can examine the traffic as it is being sent or received.



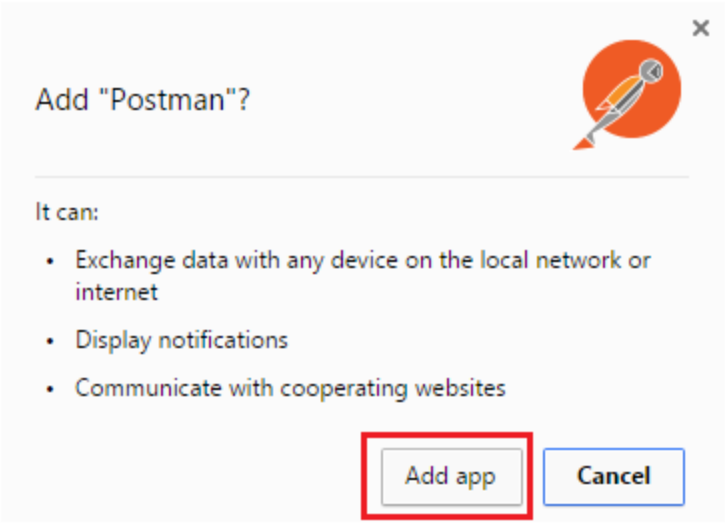
Postman

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

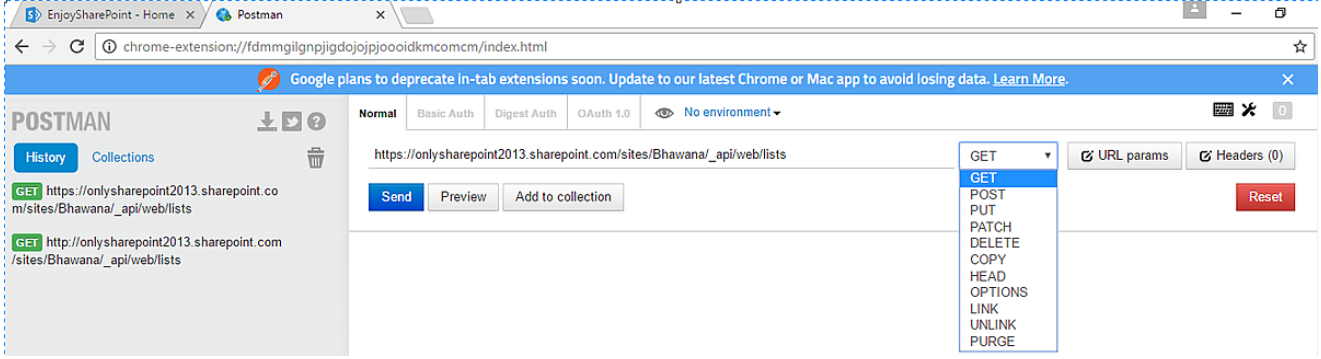
from here.



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.We will see more of how these tools come in handy as we progress through the series.


EnjoySharePoint - Home
Postman
chrome-extension://fdmmgilgnpjigdojojpjoooidkmcomcm/index.html
Google plans to deprecate in-tab extensions soon. Update to our latest Chrome or Mac app to avoid losing data. Learn More.
POSTMAN
History
Collections
GET https://onlysharepoint2013.sharepoint.com/sites/Bhawana/_api/web/lists
GET http://onlysharepoint2013.sharepoint.com/sites/Bhawana/_api/web/lists
Normal
Basic Auth
Digest Auth
OAuth 1.0
No environment
https://onlysharepoint2013.sharepoint.com/sites/Bhawana/_api/web/lists
GET
URL params
Headers (0)
Send
Preview
Add to collection
Reset
GET
POST
PUT
PATCH
DELETE
COPY
HEAD
OPTIONS
LINK
UNLINK
PURGE

SharePoint Framework Development - Creating Client Webpart

Sharepoint Journey

Become Awesome in SharePoint

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

```
yo
Your environment has been set up for using Node.js 6.10.0 (x64) and npm.

C:\Users\farmaccount>npm -v
4.3.0

C:\Users\farmaccount>md ClientWebPart-HelloWorld

C:\Users\farmaccount>cd ClientWebPart-HelloWorld

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

        Welcome to the
      SharePoint Client-side
        Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? (client-web-part-hello-world) _
```

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

- What is your solution name? : Accept the default client-web-part-hello-world as your solution name and choose Enter.

- Where do you want to place your files : Use Current Folder.

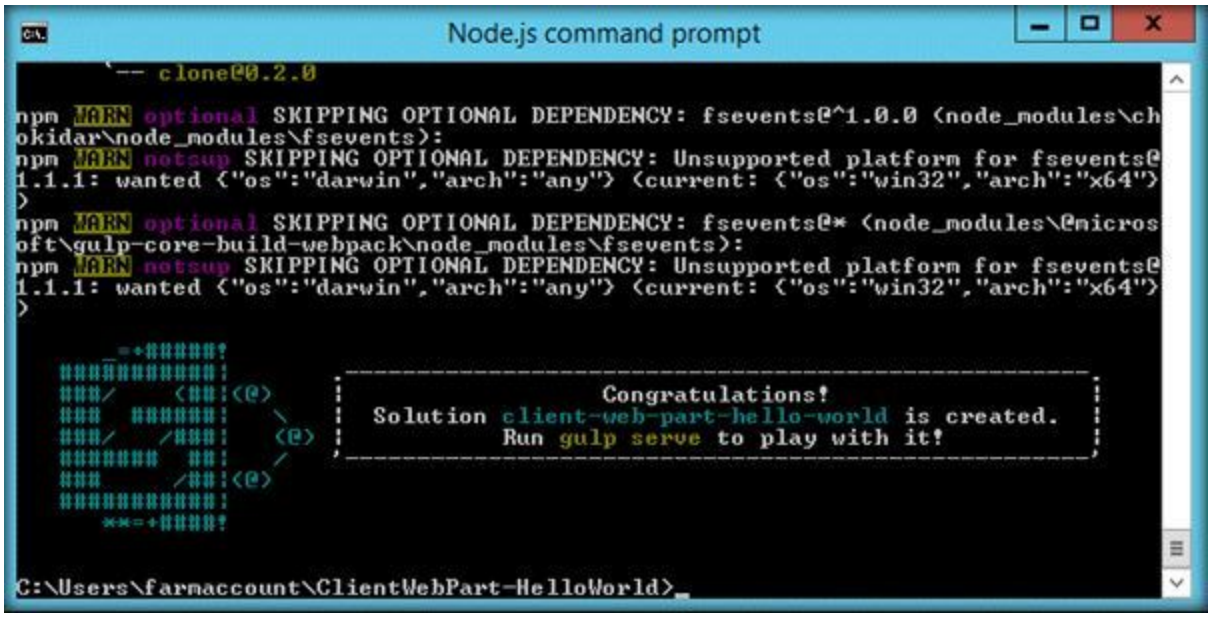
- What framework would you like to start with : Select “No javaScript web framework” for the time being as this is a sample web part.

```
yo
C:\Users\farmaccount>cd ClientWebPart-HelloWorld

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

        Welcome to the
      SharePoint Client-side
        Solution Generator

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

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

Press enter to accept the default Web part description as *HelloWorld* description.

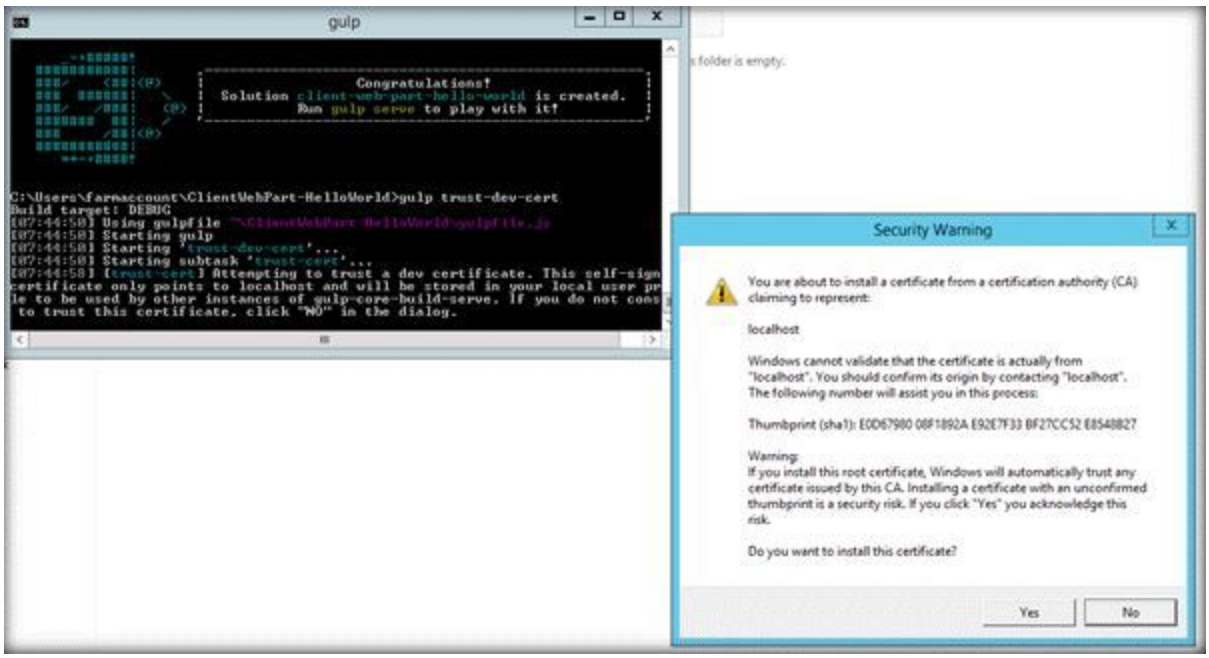
Yeoman has started working on the creation of the project structure. It will install the required dependencies and scaffold the solution files for the *HelloWorld* web part which will take some time to complete. Once completed, we will get a Congratulations message.



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.

```
Node.js command prompt

C:\Users\farmaccount\ClientWebPart-HelloWorld>gulp trust-dev-cert
Build target: DEBUG
[07:44:50] Using gulpfile ~\ClientWebPart-HelloWorld\gulpfile.js
[07:44:50] Starting gulp
[07:44:50] Starting 'trust-dev-cert'...
[07:44:50] Starting subtask 'trust-cert'...
[07:44:58] [trust-cert] Attempting to trust a dev certificate. This self-sign
certificate only points to localhost and will be stored in your local user pr
le to be used by other instances of gulp-core-build-serve. If you do not cons
 to trust this certificate, click "NO" in the dialog.
[07:45:49] Finished subtask 'trust-cert' after 59 s
[07:45:49] Finished 'trust-dev-cert' after 59 s
[07:45:49] ==================[ Finished ]==================
[07:45:50] Project client-web-part-hello-world version: 0.0.1
[07:45:50] Build tools version: 2.4.0
[07:45:50] Node version: v6.10.0
[07:45:50] Total duration: 1.03 min
```

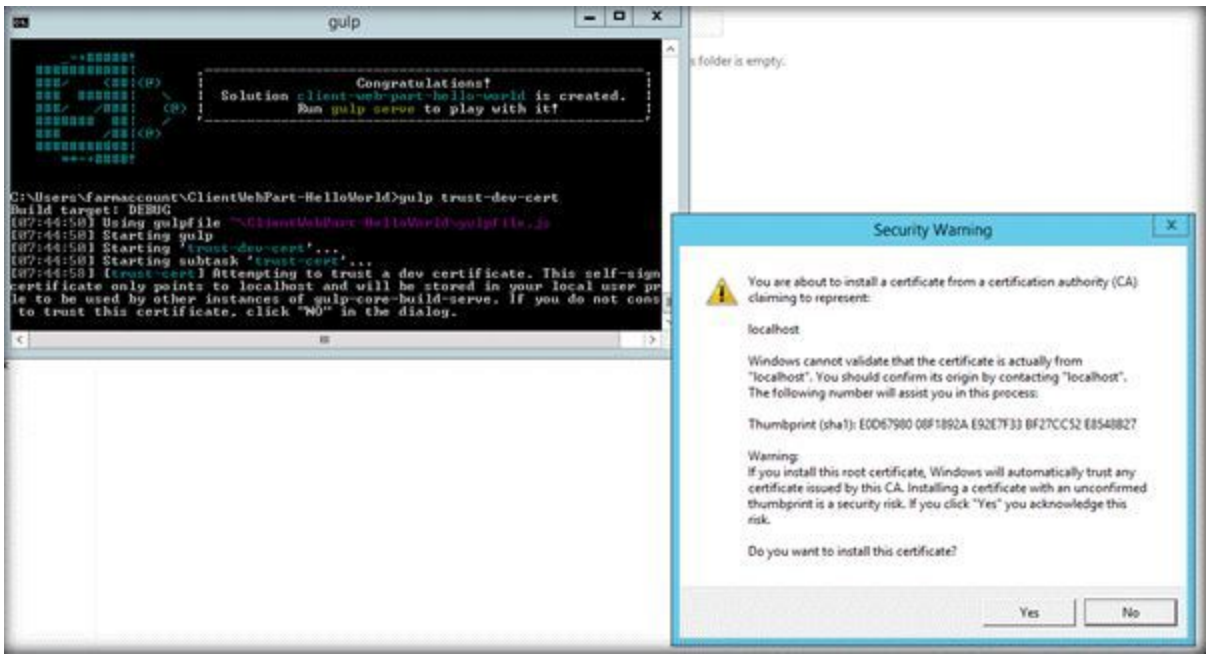
Now, let's preview the web part by running the gulp server command. This command will execute a series of gulp tasks and will create a Node-based HTTPS server at 'localhost:4321'. It will then open the browser and display the client web part.

```
gulp

C:\Users\farmaccount\ClientWebPart-HelloWorld>gulp serve
Build target: DEBUG
[07:46:47] Using gulpfile ~\ClientWebPart-HelloWorld\gulpfile.js
[07:46:47] Starting gulp
[07:46:47] Starting 'serve'...
[07:46:47] Starting subtask 'pre-copy'...
[07:46:47] Finished subtask 'pre-copy' after 12 ms
[07:46:47] Starting subtask 'copy-static-assets'...
[07:46:47] Starting subtask 'sass'...
[07:46:48] Finished subtask 'copy-static-assets' after 1.18 s
[07:46:48] Finished subtask 'sass' after 1.15 s
[07:46:48] Starting subtask 'tslint'...
[07:46:48] Starting subtask 'typescript'...
[07:46:48] [typescript] TypeScript version: 2.1.6
[07:46:52] Finished subtask 'tslint' after 4.40 s
[07:46:52] Finished subtask 'typescript' after 4.46 s
[07:46:52] Starting subtask 'ts-npm-lint'...
[07:46:53] Finished subtask 'ts-npm-lint' after 34 ms
[07:46:53] Starting subtask 'api-extractor'...
[07:46:53] Finished subtask 'api-extractor' after 1.63 ms
[07:46:53] Starting subtask 'post-copy'...
[07:46:53] Finished subtask 'post-copy' after 1.19 ms
[07:46:53] Starting subtask 'collectLocalizedResources'...
[07:46:53] Finished subtask 'collectLocalizedResources' after 11 ms
[07:46:53] Starting subtask 'configure-webpack'...
[07:46:53] Finished subtask 'configure-webpack' after 516 ms
[07:46:53] Starting subtask 'webpack'...
[07:46:54] Finished subtask 'webpack' after 844 ms
[07:46:54] Starting subtask 'configure-webpack-external-bundling'...
[07:46:54] Finished subtask 'configure-webpack-external-bundling' after 1.97
[07:46:54] Starting subtask 'copy-assets'...
[07:46:54] Finished subtask 'copy-assets' after 36 ms
[07:46:54] Starting subtask 'write-manifests'...
[07:46:55] Finished subtask 'write-manifests' after 797 ms
[07:46:55] Starting subtask 'serve'...
Starting api server on port 5432.
Registring api: /getwebparts
Registring api: /*.*
```

SharePoint Workbench

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.

```
Node.js command prompt

C:\Users\farmaccount\ClientWebPart-HelloWorld>gulp trust-dev-cert
Build target: DEBUG
[07:44:50] Using gulpfile ~\ClientWebPart-HelloWorld\gulpfile.js
[07:44:50] Starting gulp
[07:44:50] Starting 'trust-dev-cert'...
[07:44:50] Starting subtask 'trust-cert'...
[07:44:58] [trust-cert] Attempting to trust a dev certificate. This self-sign
certificate only points to localhost and will be stored in your local user pr
le to be used by other instances of gulp-core-build-serve. If you do not cons
 to trust this certificate, click "NO" in the dialog.
[07:45:49] Finished subtask 'trust-cert' after 59 s
[07:45:49] Finished 'trust-dev-cert' after 59 s
[07:45:49] ==================[ Finished ]==================
[07:45:50] Project client-web-part-hello-world version: 0.0.1
[07:45:50] Build tools version: 2.4.0
[07:45:50] Node version: v6.10.0
[07:45:50] Total duration: 1.03 min
```

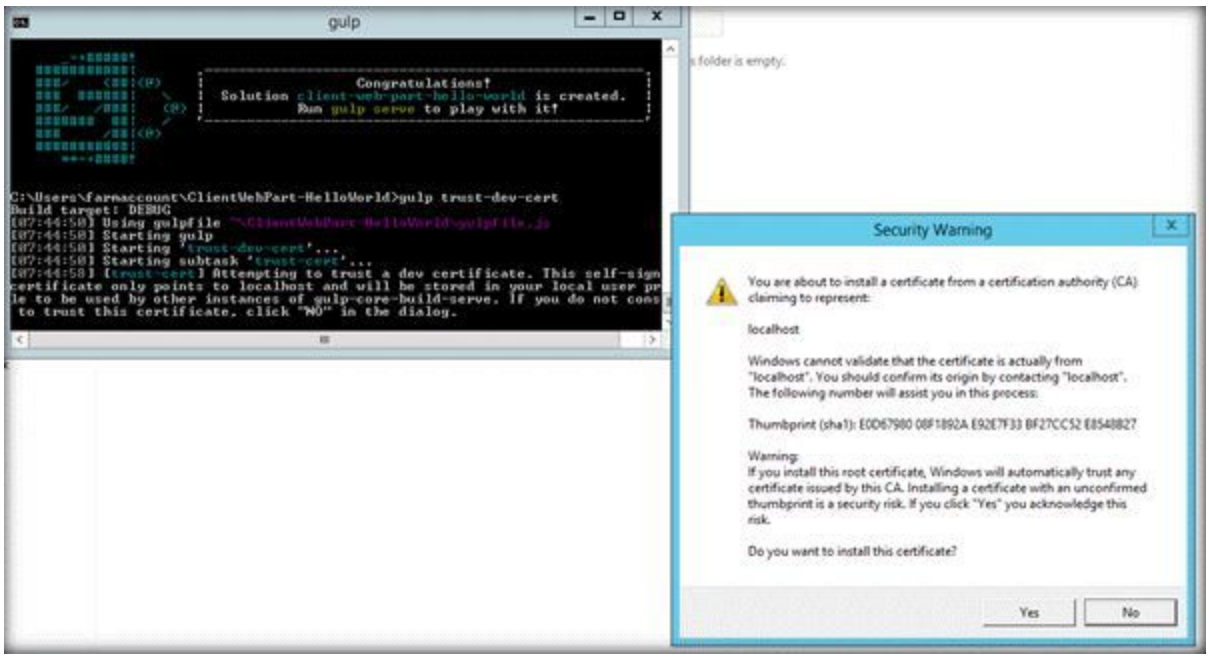
Now, let's preview the web part by running the gulp server command. This command will execute a series of gulp tasks and will create a Node-based HTTPS server at 'localhost:4321'. It will then open the browser and display the client web part.

```
gulp

C:\Users\farmaccount\ClientWebPart-HelloWorld>gulp serve
Build target: DEBUG
[07:46:47] Using gulpfile ~\ClientWebPart-HelloWorld\gulpfile.js
[07:46:47] Starting gulp
[07:46:47] Starting 'serve'...
[07:46:47] Starting subtask 'pre-copy'...
[07:46:47] Finished subtask 'pre-copy' after 12 ms
[07:46:47] Starting subtask 'copy-static-assets'...
[07:46:47] Starting subtask 'sass'...
[07:46:48] Finished subtask 'copy-static-assets' after 1.18 s
[07:46:48] Finished subtask 'sass' after 1.15 s
[07:46:48] Starting subtask 'tslint'...
[07:46:48] Starting subtask 'typescript'...
[07:46:48] [typescript] TypeScript version: 2.1.6
[07:46:52] Finished subtask 'tslint' after 4.40 s
[07:46:52] Finished subtask 'typescript' after 4.46 s
[07:46:52] Starting subtask 'ts-npm-lint'...
[07:46:53] Finished subtask 'ts-npm-lint' after 34 ms
[07:46:53] Starting subtask 'api-extractor'...
[07:46:53] Finished subtask 'api-extractor' after 1.63 ms
[07:46:53] Starting subtask 'post-copy'...
[07:46:53] Finished subtask 'post-copy' after 1.19 ms
[07:46:53] Starting subtask 'collectLocalizedResources'...
[07:46:53] Finished subtask 'collectLocalizedResources' after 11 ms
[07:46:53] Starting subtask 'configure-webpack'...
[07:46:53] Finished subtask 'configure-webpack' after 516 ms
[07:46:53] Starting subtask 'webpack'...
[07:46:54] Finished subtask 'webpack' after 844 ms
[07:46:54] Starting subtask 'configure-webpack-external-bundling'...
[07:46:54] Finished subtask 'configure-webpack-external-bundling' after 1.97
[07:46:54] Starting subtask 'copy-assets'...
[07:46:54] Finished subtask 'copy-assets' after 36 ms
[07:46:54] Starting subtask 'write-manifests'...
[07:46:55] Finished subtask 'write-manifests' after 797 ms
[07:46:55] Starting subtask 'serve'...
Starting api server on port 5432.
Registring api: /getwebparts
Registring api: /*.*
```

Edit 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.

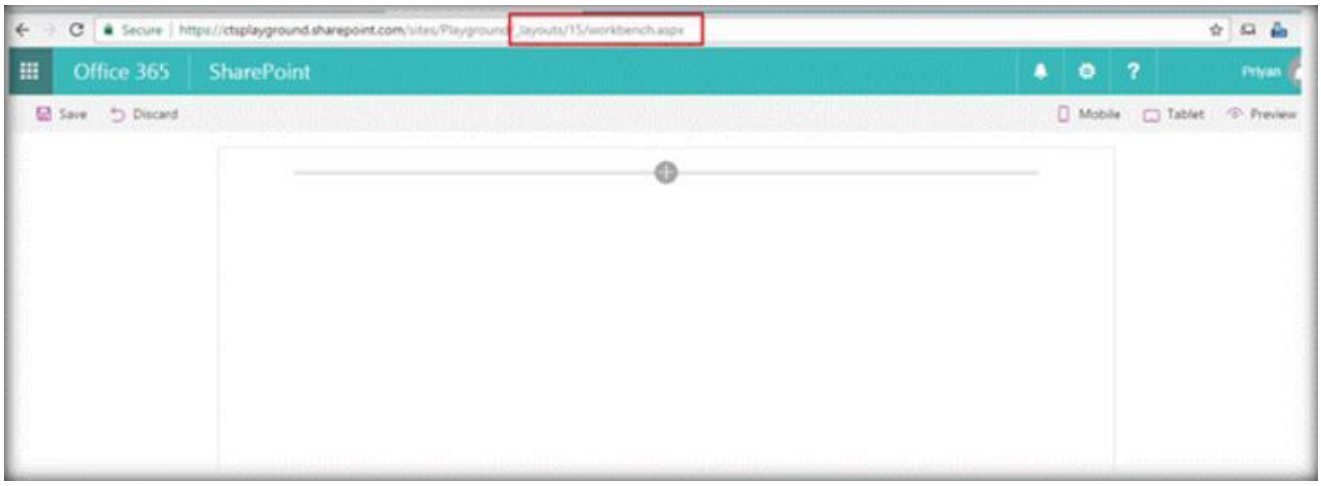
```
Node.js command prompt

C:\Users\farmaccount\ClientWebPart-HelloWorld>gulp trust-dev-cert
Build target: DEBUG
[07:44:50] Using gulpfile ~\ClientWebPart-HelloWorld\gulpfile.js
[07:44:50] Starting gulp
[07:44:50] Starting 'trust-dev-cert'...
[07:44:50] Starting subtask 'trust-cert'...
[07:44:58] [trust-cert] Attempting to trust a dev certificate. This self-sign
certificate only points to localhost and will be stored in your local user pr
le to be used by other instances of gulp-core-build-serve. If you do not cons
 to trust this certificate, click "NO" in the dialog.
[07:45:49] Finished subtask 'trust-cert' after 59 s
[07:45:49] Finished 'trust-dev-cert' after 59 s
[07:45:49] ==================[ Finished ]==================
[07:45:50] Project client-web-part-hello-world version: 0.0.1
[07:45:50] Build tools version: 2.4.0
[07:45:50] Node version: v6.10.0
[07:45:50] Total duration: 1.03 min
```

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

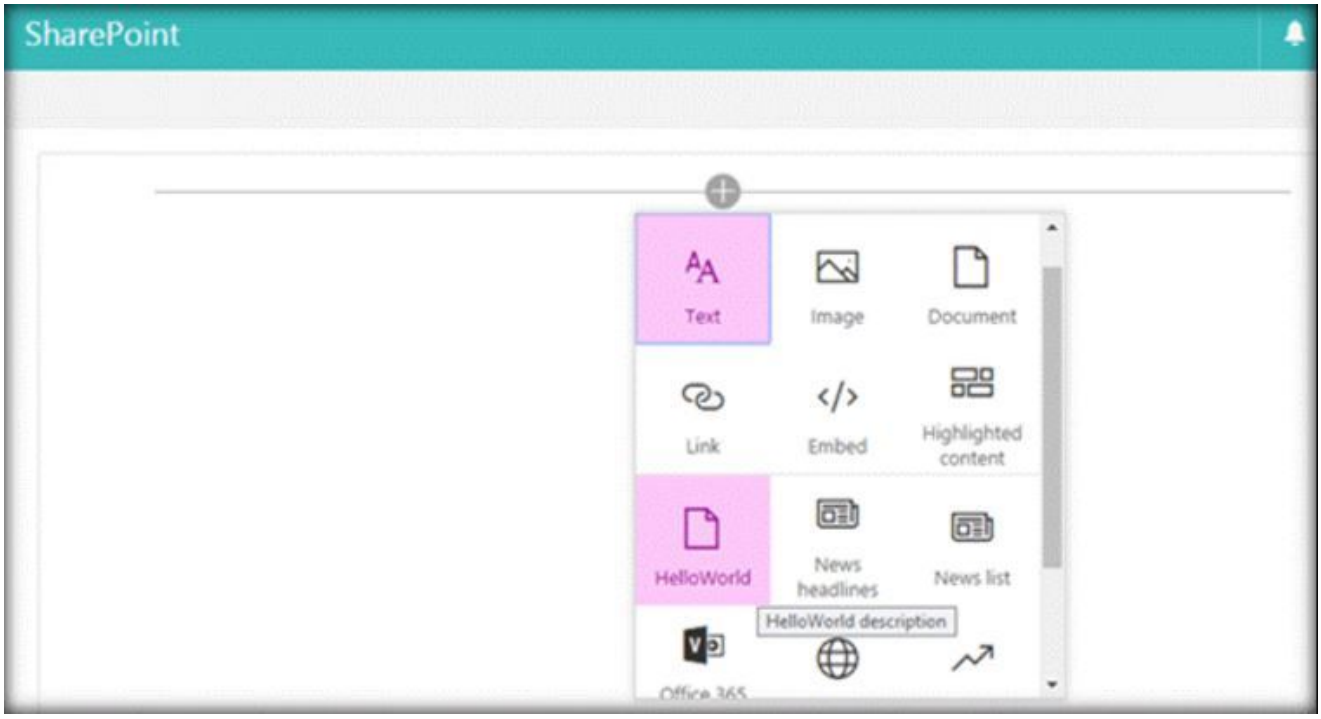
```
gulp

C:\Users\farmaccount\ClientWebPart-HelloWorld>gulp serve
Build target: DEBUG
[07:46:47] Using gulpfile ~\ClientWebPart-HelloWorld\gulpfile.js
[07:46:47] Starting gulp
[07:46:47] Starting 'serve'...
[07:46:47] Starting subtask 'pre-copy'...
[07:46:47] Finished subtask 'pre-copy' after 12 ms
[07:46:47] Starting subtask 'copy-static-assets'...
[07:46:47] Starting subtask 'sass'...
[07:46:48] Finished subtask 'copy-static-assets' after 1.18 s
[07:46:48] Finished subtask 'sass' after 1.15 s
[07:46:48] Starting subtask 'tslint'...
[07:46:48] Starting subtask 'typescript'...
[07:46:48] [typescript] TypeScript version: 2.1.6
[07:46:52] Finished subtask 'tslint' after 4.40 s
[07:46:52] Finished subtask 'typescript' after 4.46 s
[07:46:52] Starting subtask 'ts-npm-lint'...
[07:46:53] Finished subtask 'ts-npm-lint' after 34 ms
[07:46:53] Starting subtask 'api-extractor'...
[07:46:53] Finished subtask 'api-extractor' after 1.63 ms
[07:46:53] Starting subtask 'post-copy'...
[07:46:53] Finished subtask 'post-copy' after 1.19 ms
[07:46:53] Starting subtask 'collectLocalizedResources'...
[07:46:53] Finished subtask 'collectLocalizedResources' after 11 ms
[07:46:53] Starting subtask 'configure-webpack'...
[07:46:53] Finished subtask 'configure-webpack' after 516 ms
[07:46:53] Starting subtask 'webpack'...
[07:46:54] Finished subtask 'webpack' after 844 ms
[07:46:54] Starting subtask 'configure-webpack-external-bundling'...
[07:46:54] Finished subtask 'configure-webpack-external-bundling' after 1.97
[07:46:54] Starting subtask 'copy-assets'...
[07:46:54] Finished subtask 'copy-assets' after 36 ms
[07:46:54] Starting subtask 'write-manifests'...
[07:46:55] Finished subtask 'write-manifests' after 797 ms
[07:46:55] Starting subtask 'serve'...
Starting api server on port 5432.
Registring api: /getwebparts
Registring api: /*.*
```

Add the Web Part to SharePoint

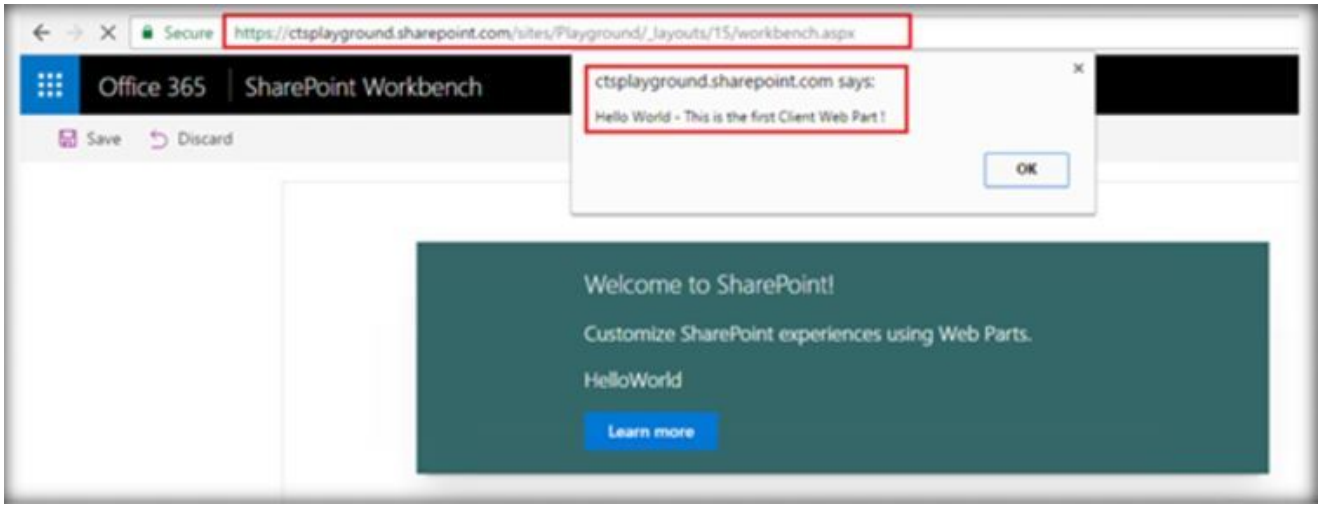
So far we were testing the web part in SharePoint Workbench locally, now let's try to test it within the SharePoint Context. SharePoint Workbench is also hosted in SharePoint Online to preview the web part. It can be accessed by adding ' _layouts/15/workbench.aspx' to the SharePoint Online URL.



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.



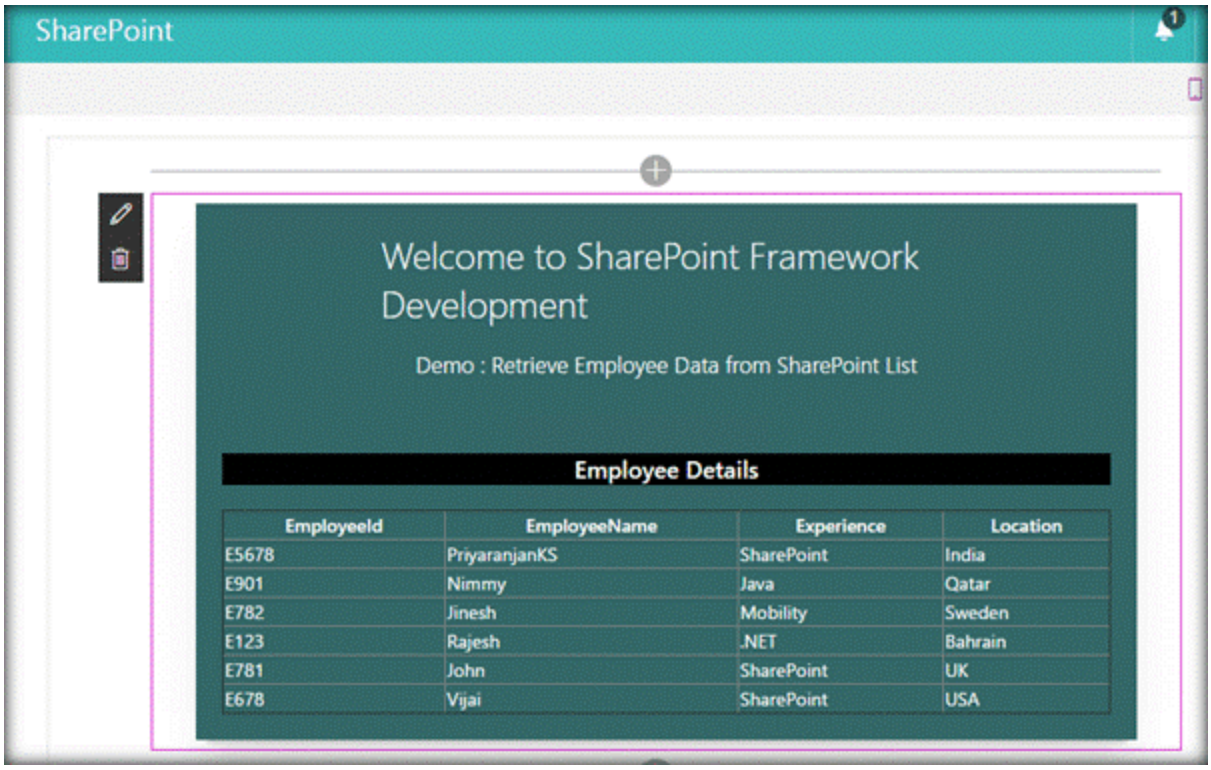
SharePoint Framework Development - Create Client Web Part to Retrieve and Display List Items

Sharepoint Journey

Become Awesome in SharePoint

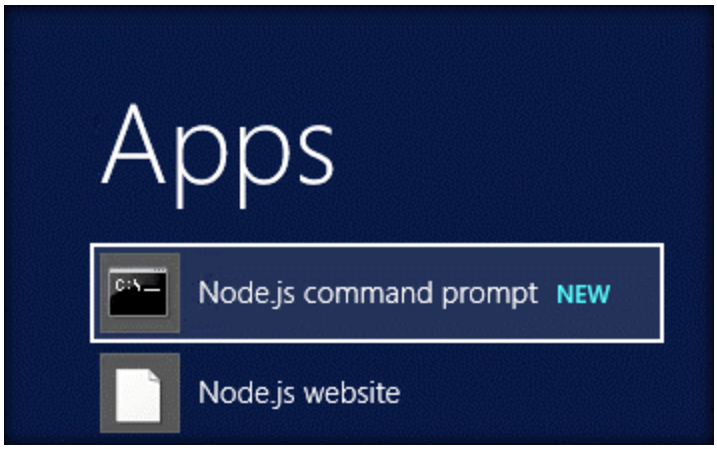
Tabular Form Client Web Part

This is the continuation of the SharePoint Framework Development series. In the earlier article, we saw how to set up the environment and the tool chains required to start the development using SharePoint Framework(SPFx). Let's create a client web part using TypeScript and SPFx which will be retrieving the list items from SharePoint List (EmployeeList) and will display it in the tabular form in the client Web part, as shown below.



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.

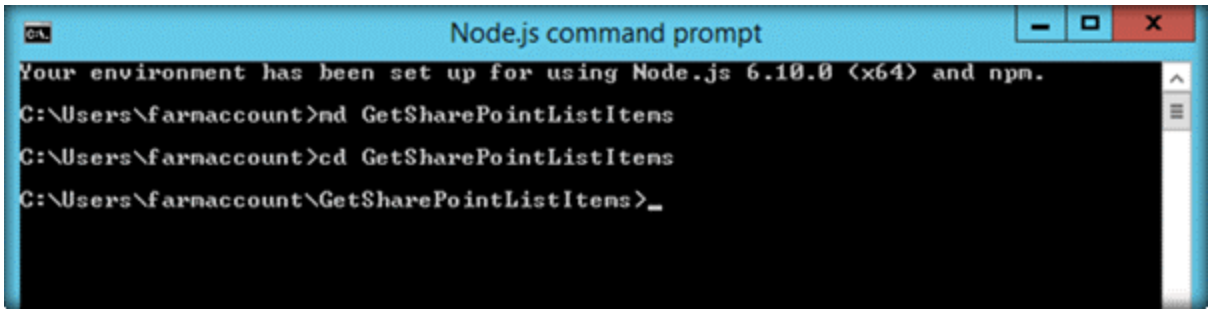


We can create the directory, where we will be adding the solution, using the command given below.

md GetSharePointListItems

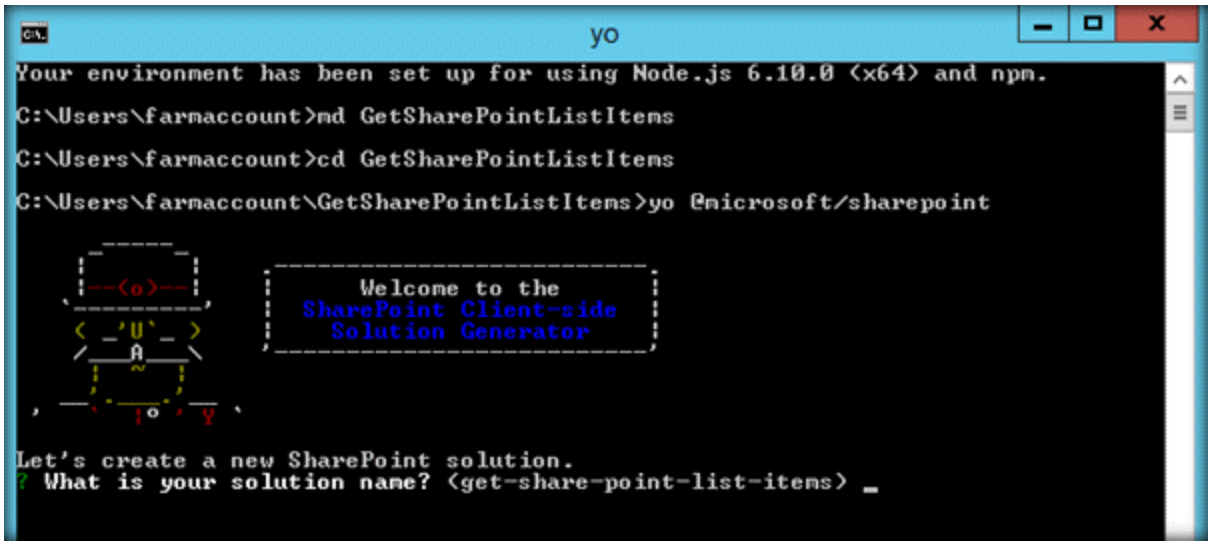
Let's move to the newly created working directory, using the command.

cd GetSharePointListItems



We will then create the client web part by running the Yeoman SharePoint Generator:

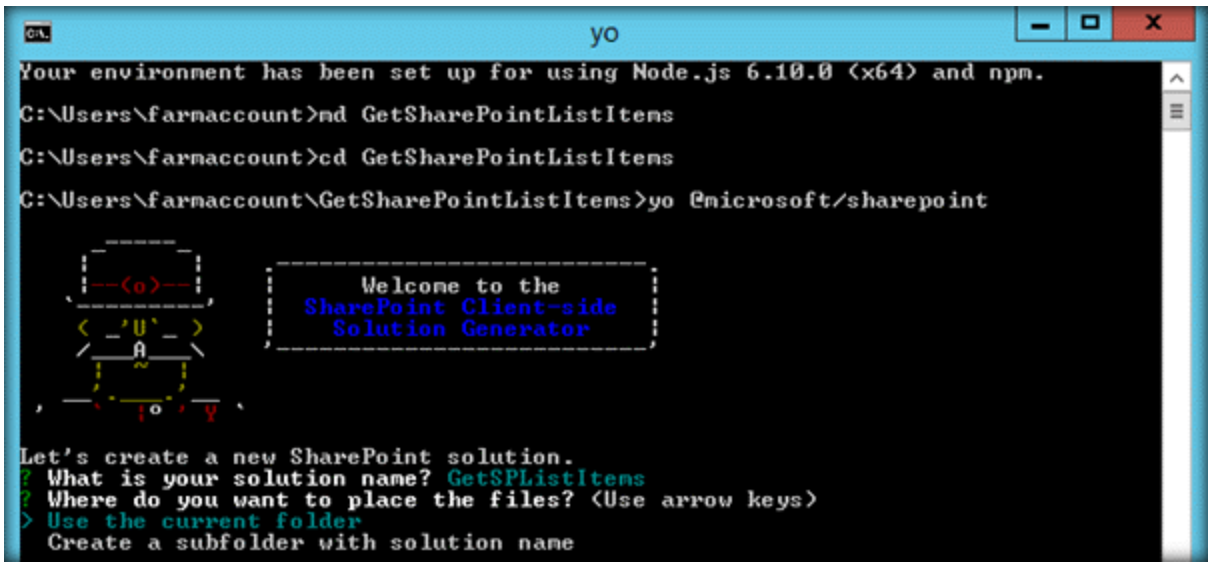
yo @microsoft/sharepoint



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.



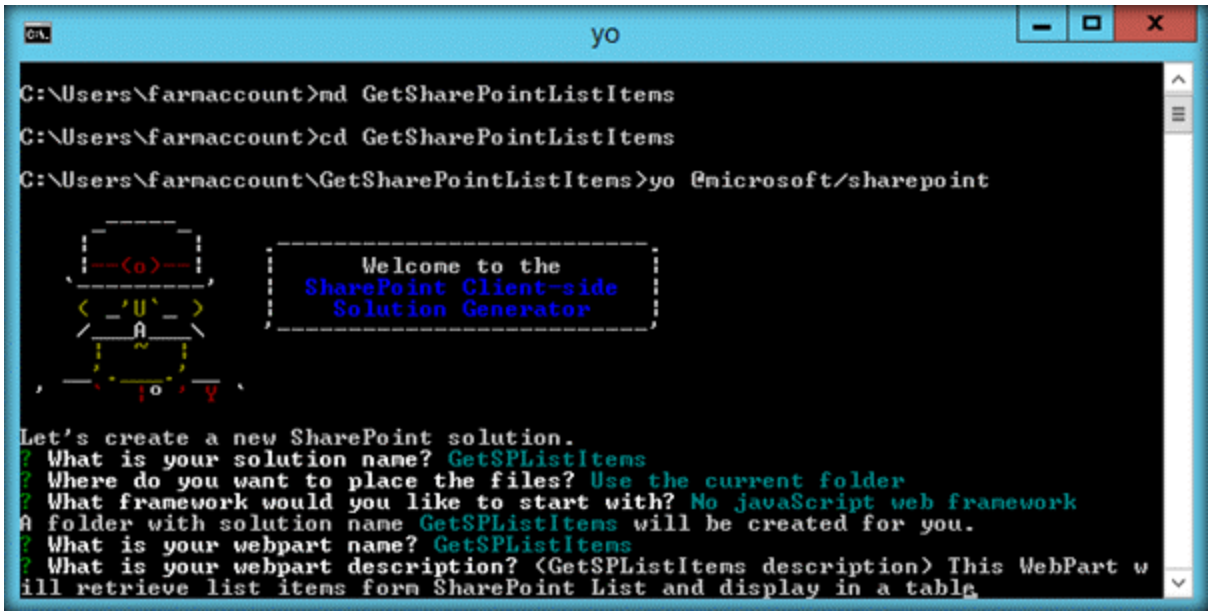
- Where do you want to place your files- *Use current folder.*

- What framework would you like to start with- Select "*No javaScript*

web framework" for the time being, as this is a sample Web part.

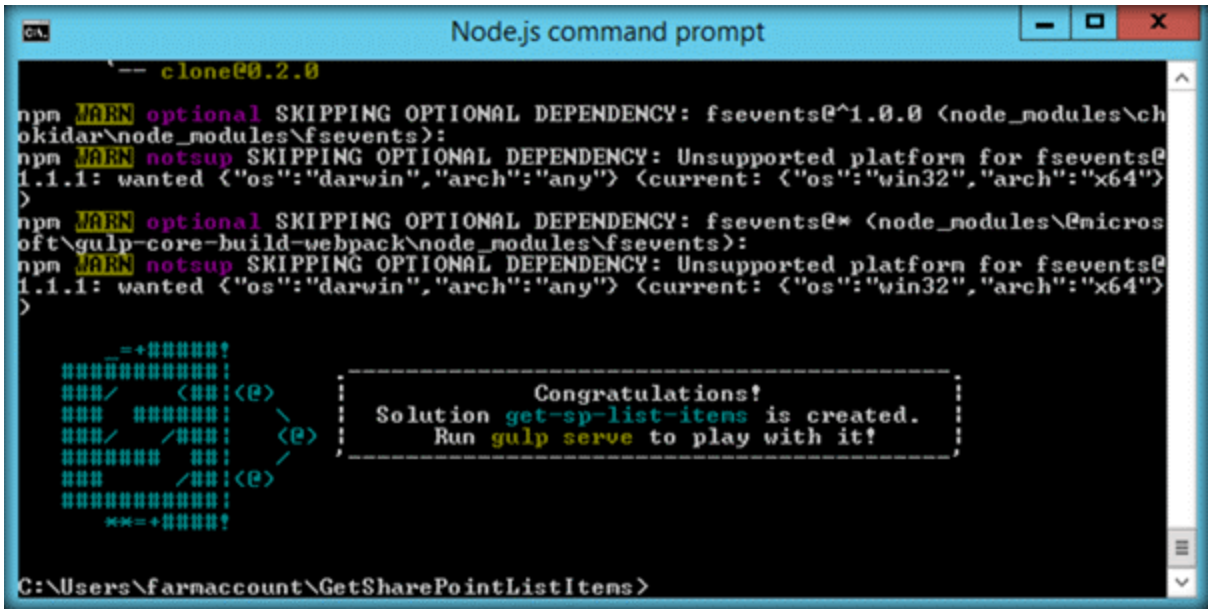
- What is your Web part name- We will specify it as '*GetSPListItems*' and press Enter

- What is your Web part description- We will specify it as *this Web part 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

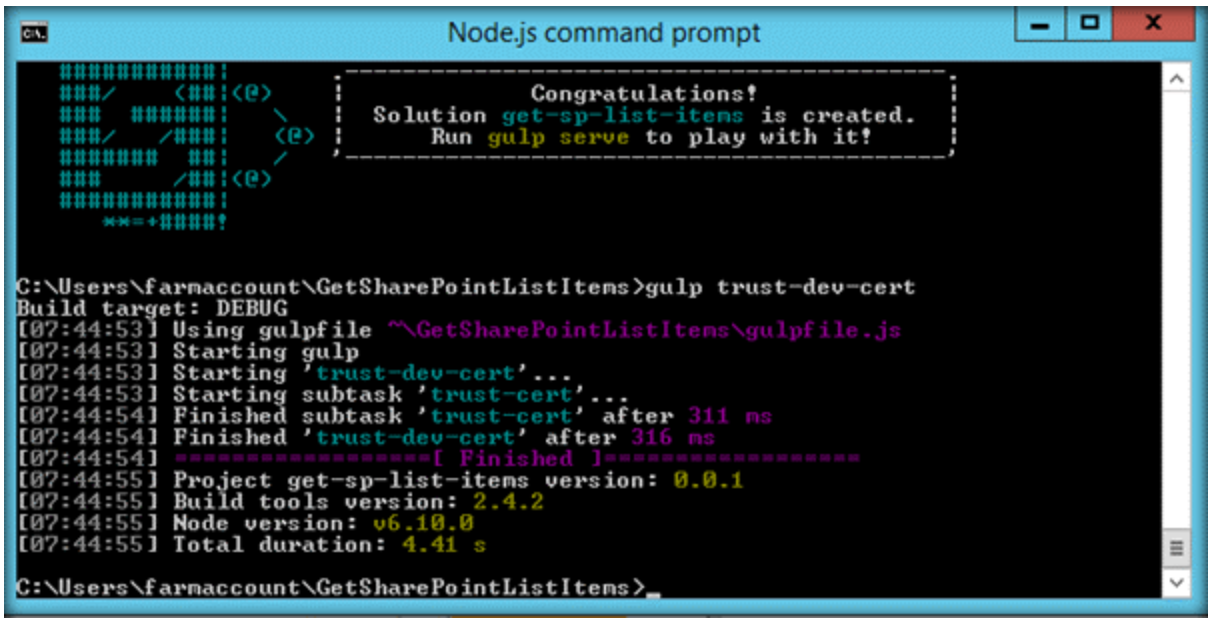
completed, we will get a congratulations message.



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 toolchain 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.

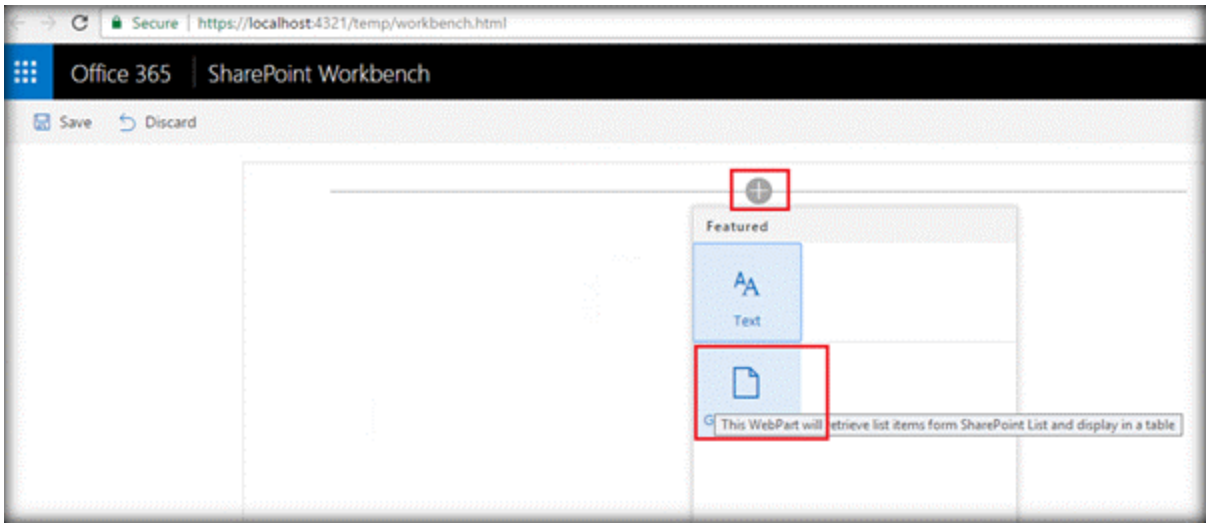
gulp trust-dev-cert



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

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

C:\Users\farmaccount\GetSharePointListItems>gulp serve
Build target: DEBUG
[07:46:14] Using gulpfile ~\GetSharePointListItems\gulpfile.js
[07:46:14] Starting gulp
[07:46:14] Starting 'serve'...
[07:46:14] Starting subtask 'pre-copy'...
[07:46:14] Finished subtask 'pre-copy' after 16 ms
[07:46:14] Starting subtask 'copy-static-assets'...
[07:46:14] Starting subtask 'sass'...
[07:46:16] Finished subtask 'copy-static-assets' after 1.14 s
[07:46:16] Finished subtask 'sass' after 1.11 s
[07:46:16] Starting subtask 'tslint'...
[07:46:16] Starting subtask 'typescript'...
```

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

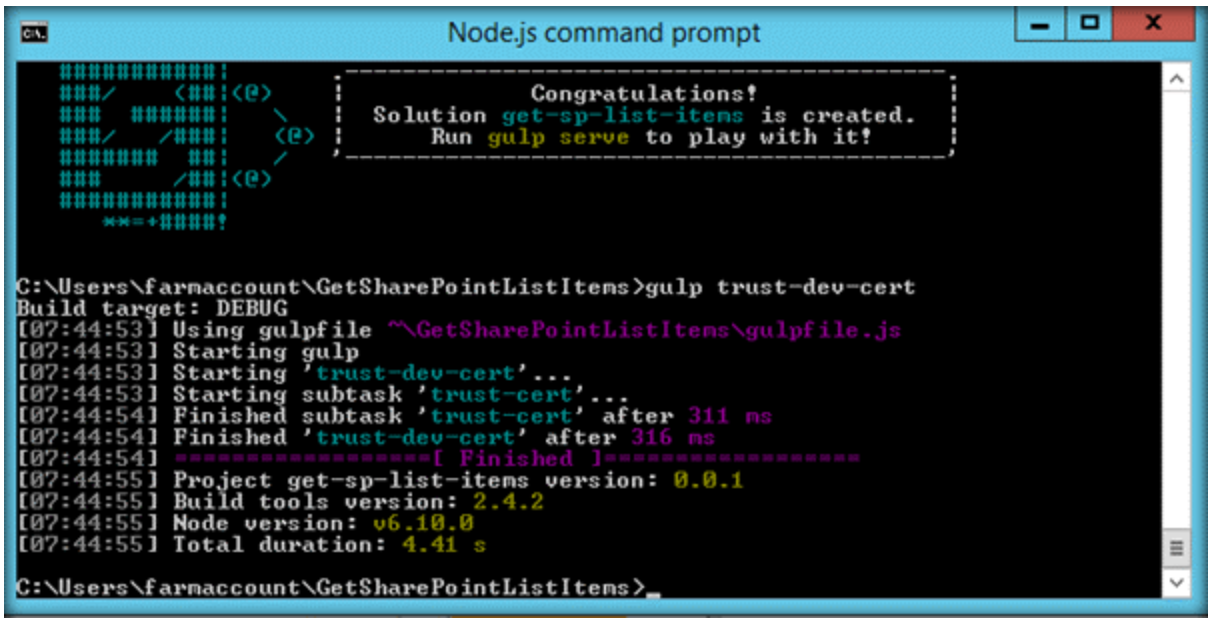


This indicates that the project structure is set up correctly. We will now open the solution in Visual Studio Code to add the logic to retrieve the list items from SharePoint and display it as a table on this page.

Edit 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. SharePoint Framework toolchain 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.

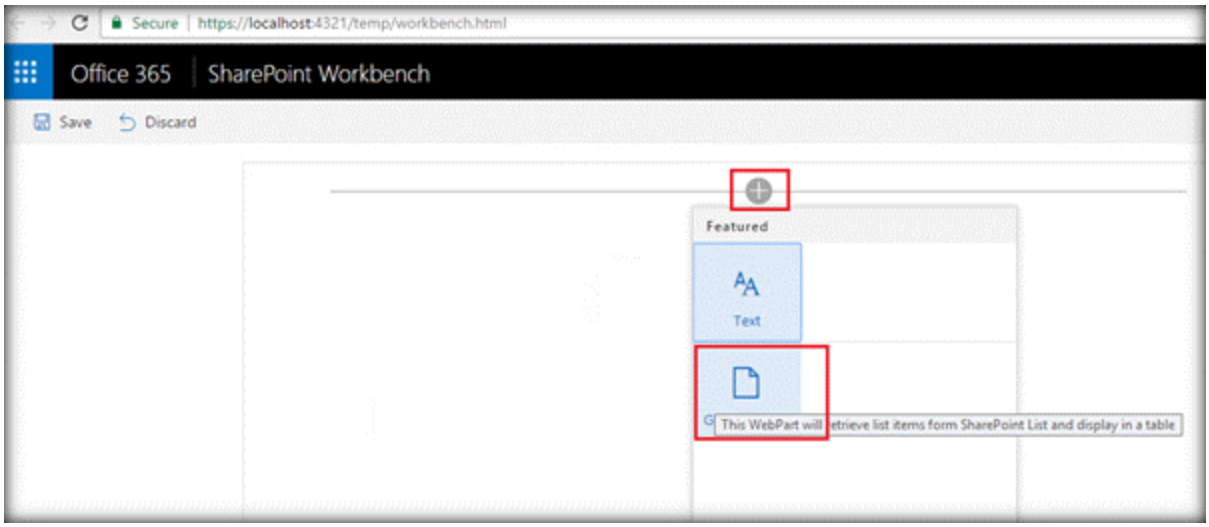
gulp trust-dev-cert



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

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

C:\Users\farmaccount\GetSharePointListItems>gulp serve
Build target: DEBUG
[07:46:14] Using gulpfile ~\GetSharePointListItems\gulpfile.js
[07:46:14] Starting gulp
[07:46:14] Starting 'serve'...
[07:46:14] Starting subtask 'pre-copy'...
[07:46:14] Finished subtask 'pre-copy' after 16 ms
[07:46:14] Starting subtask 'copy-static-assets'...
[07:46:14] Starting subtask 'sass'...
[07:46:16] Finished subtask 'copy-static-assets' after 1.14 s
[07:46:16] Finished subtask 'sass' after 1.11 s
[07:46:16] Starting subtask 'tslint'...
[07:46:16] Starting subtask 'typescript'...
```

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



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

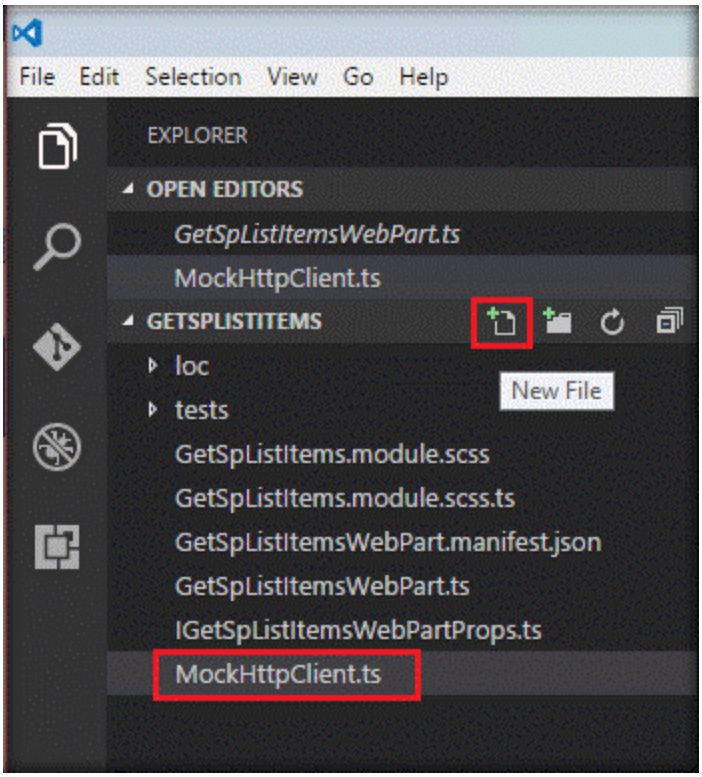
Define List Model

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.

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

Create Mock HTTPClient to test data locally

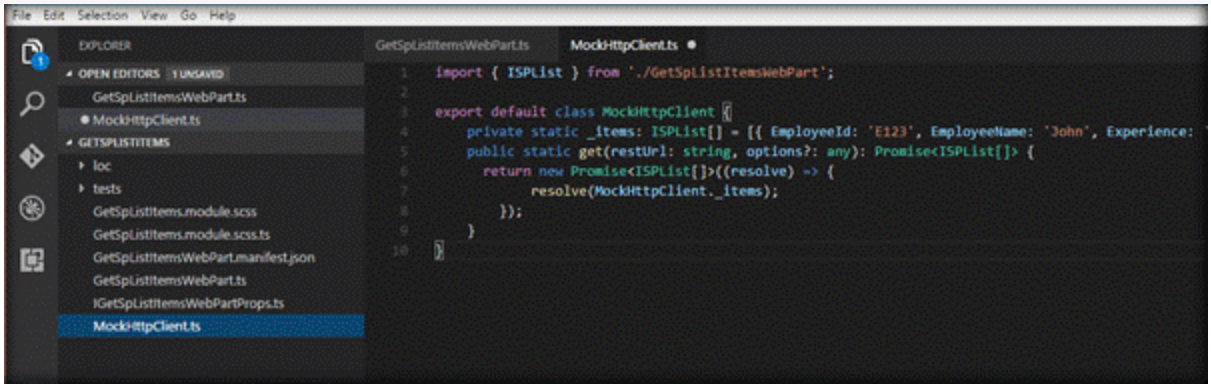
In order to test the list item retrieval in the local workbench, we will create a mock store, which returns mock Employee list data. We will create a new file inside '*src\webparts\ getSpListItems'* folder named *MockHttpClient.ts,* as shown below. This is something that is optional. However, to check the functionality locally without connecting to SharePoint, we can make use of the mock client.



We will then copy the code given below into MockHttpClient.ts, as shown below. You may also click here to get the code.

```
import { ISPList } from

 './GetSpListItemsWebPart';
```

```
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 '*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';*"

import MockHttpClient from './MockHttpClient';

We will also add the mock list item retrieval method within the 'GetSpListItemsWebPart' class. You can click here to get the code.

```
private _getMockListData(): Promise<ISPLists> {
    return

 MockHttpClient.get(this.context.pageContext.web.absoluteUr
l).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>;
}
```

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.

```
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();
    });
}
```

You may also check the code online by clicking this.

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 will be called else the method that calls REST API to retrieve SharePoint list items will be called. To view the code online, click here.

```
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. Click here to view the code online.

```
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;
}
```

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

the code online, click here.

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

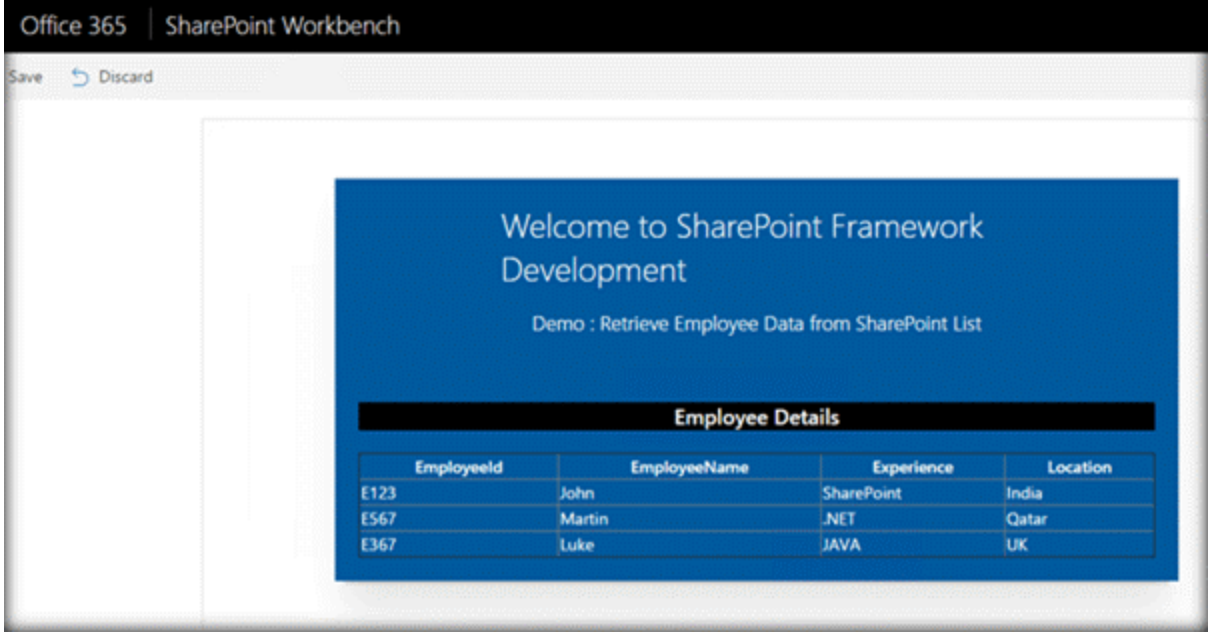
```
    <div class="ms-Grid-row ms-bgColor-themeDark ms-
fontColor-white ${styles.row}">
      <div class="ms-Grid-col ms-u-lg10 ms-u-xl8 ms-u-
xlPush2   ms-u-lgPush1">
        <span class="ms-font-xl ms-fontColor-white"
style="font-size:28px">Welcome to SharePoint Framework
Development</span>

        <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();
  }
```

Test the Web part in local SharePoint Workbench

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

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

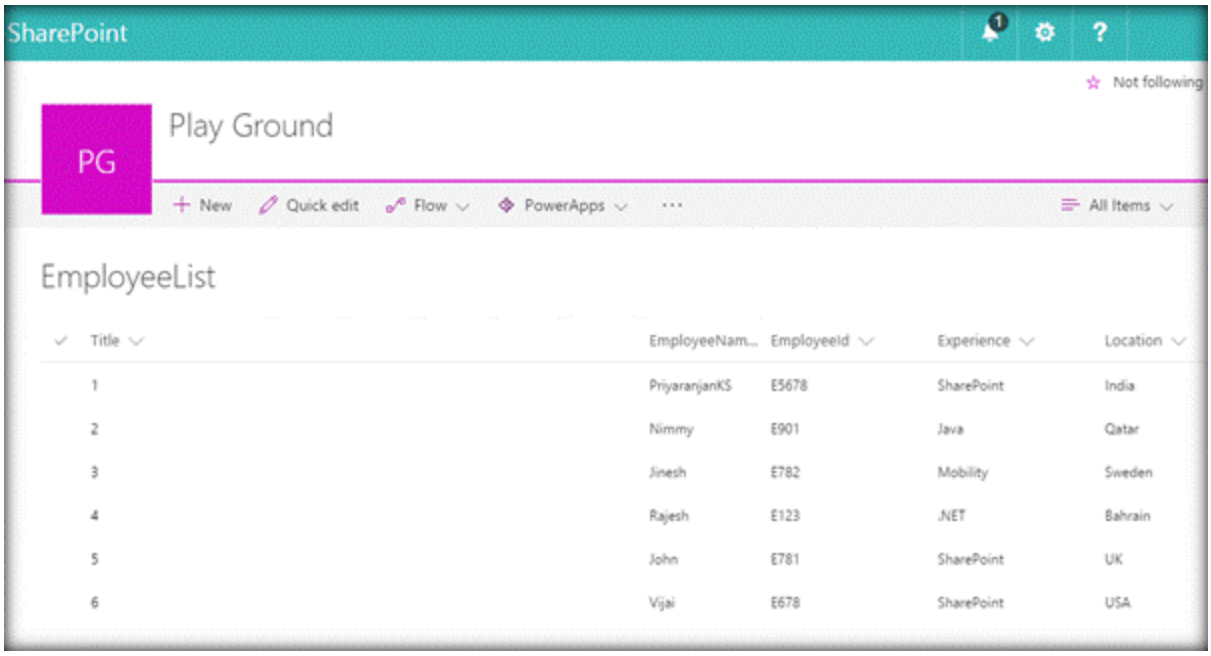
Since the environment is local, the mock data has been used to generate the table, as shown below.



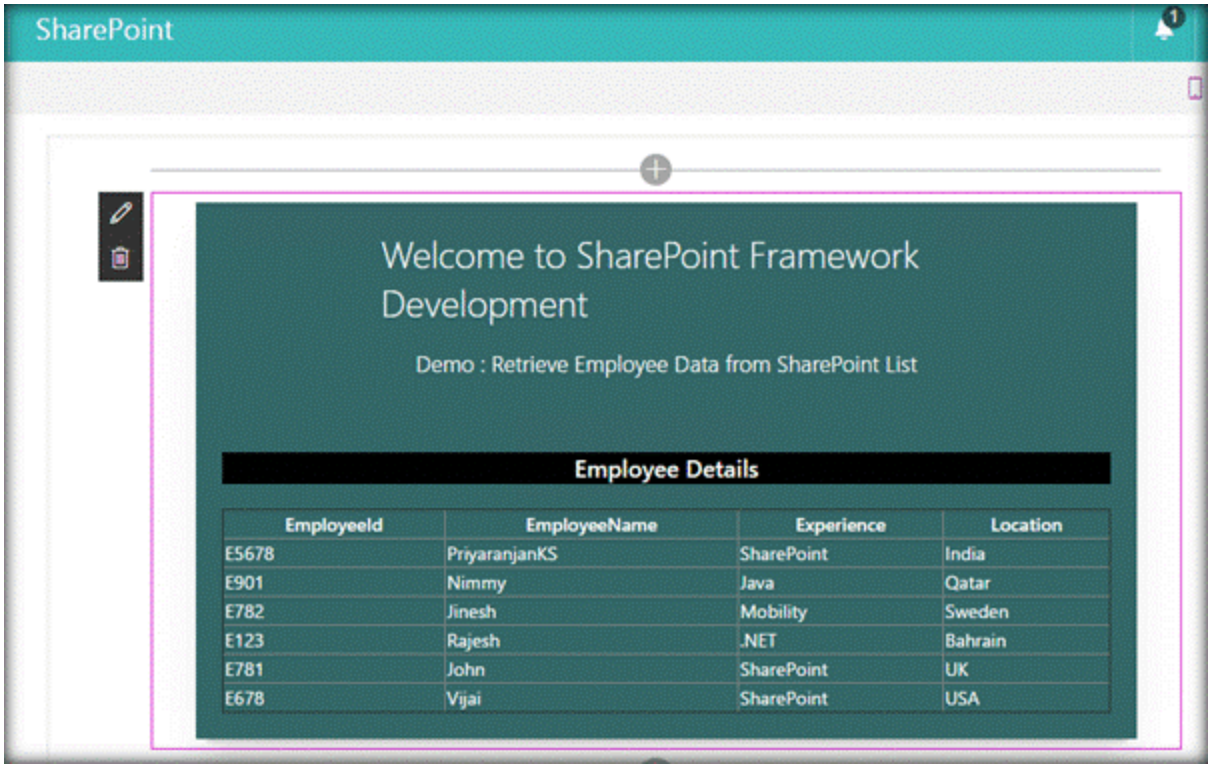
Thus, we have successfully tested the client Web part locally.

Test the Web Part in SharePoint Online

Now, let's test the Web part in SharePoint Workbench available in SharePoint Online. This time, the 'EnvironmentType' check will evaluate to SharePoint and REST API endpoint method will be called to retrieve the list items from SharePoint list. 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 site collection 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 main TS file contents are shown below. You can download the main solution files used for the demo from here. To sum up, the control flow would be:

- render() is invoked which builds the basic UI and calls the _renderListAsync().
- _renderListAsync() method checks if the environment is local or

- SharePoint Online.
- If it is local environment, _getMockListData() is invoked and the mock data is retrieved. Else the _getListData() is invoked and SharePoint data is retrieved using REST api.
- Finally, the retrieved data is passed to _renderList to create the table at the predefined div named *spListContainer.*

You may check the code online by clicking here .

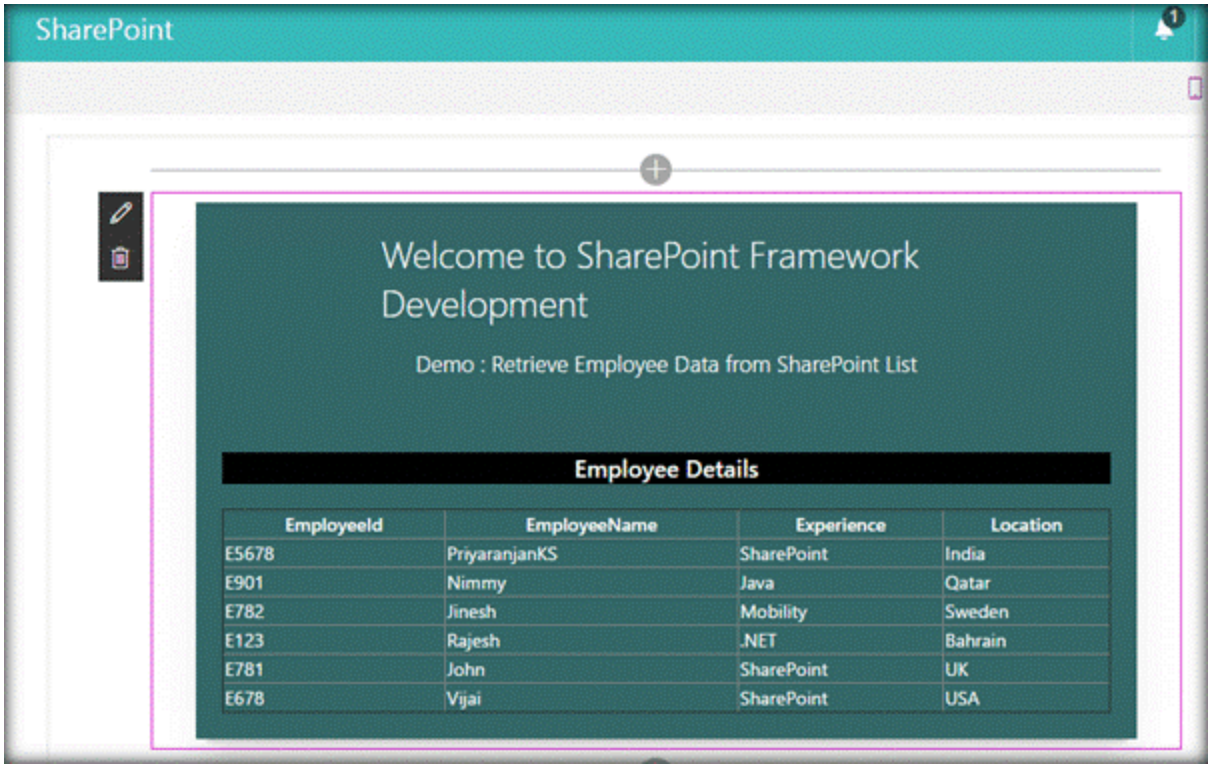
```
import { Version } from '@microsoft/sp-core-library';
import {
  BaseClientSideWebPart,
  IPropertyPaneConfiguration,
  PropertyPaneTextField
} from '@microsoft/sp-webpart-base';
import { escape } from '@microsoft/sp-lodash-subset';

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


import styles from './GetSpListItems.module.scss';
import * as strings from 'getSpListItemsStrings';
import { IGetSpListItemsWebPartProps } from
'./IGetSpListItemsWebPartProps';
import MockHttpClient from './MockHttpClient';

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


        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
                })
              ]
            }
          ]
        }
      ]
    };
  }
}
```


SharePoint
Welcome to SharePoint Framework Development
Demo : Retrieve Employee Data from SharePoint List
Employee Details
EmployeeId | EmployeeName | Experience | Location
E5678 | PriyaranjanKS | SharePoint | India
E901 | Nimmy | Java | Qatar
E782 | Jinesh | Mobility | Sweden
E123 | Rajesh | .NET | Bahrain
E781 | John | SharePoint | UK
E678 | Vijai | SharePoint | USA

Provision Custom List using SharePoint Framework

Sharepoint Journey

Become Awesome in SharePoint

Create the Web Part Project

```
yo
Your environment has been set up for using Node.js 6.10.0 (x64) and npm.

C:\Users\farmaccount>md RetrieveSearchResults

C:\Users\farmaccount>cd RetrieveSearchResults

C:\Users\farmaccount\RetrieveSearchResults>yo @microsoft/sharepoint

     _-----_     ___________________________
    |       |    |      Welcome to the      |
    |--(o)--|    |  SharePoint Client-side  |
   `---------´   |    Solution Generator    |
    ( _´U`_ )    |__________________________|
    /___A___\
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

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

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 project structure by running the Yeoman SharePoint Generator.

yo @microsoft/sharepoint

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

     _-----_     ╭──────────────────────────╮
    |       |    │      Welcome to the      │
    |--(o)--|    │  SharePoint Client-side  │
   `---------´   │    Solution Generator    │
    ( _´U`_ )    ╰──────────────────────────╯
    /___A___\
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

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

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

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

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

- Where do you want to place your files- Use current folder.

- What framework would you like to start with- Select "No javaScript web framework" for the time being, as this is a sample Web part.

- What is your Webpart name- We will specify it as 'CustomList 'and press Enter

- What is your Webpart description- We will specify it as 'Custom List Created using SharePoint Framework'

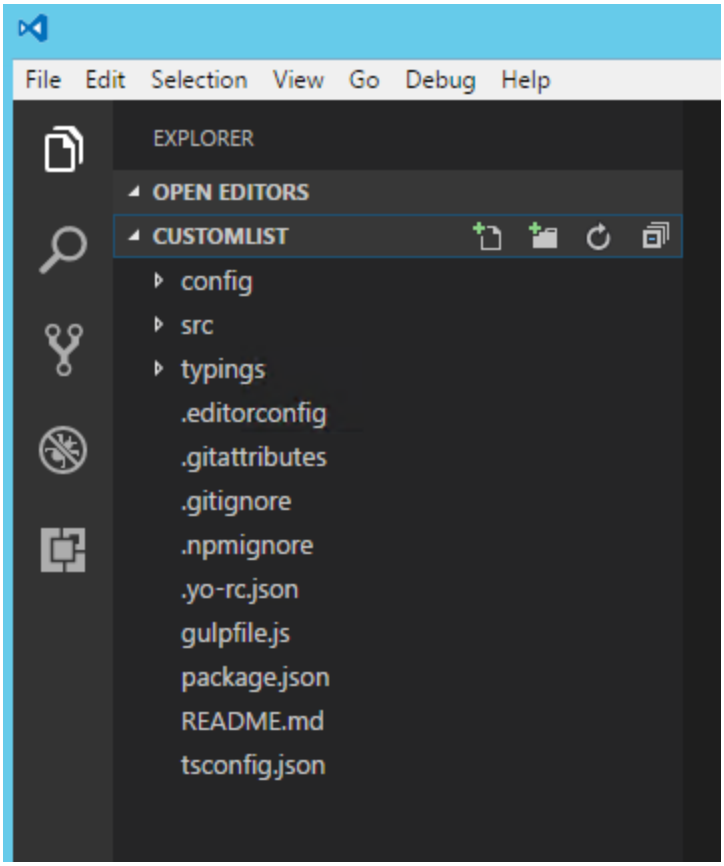
```
     _-----_     ╭──────────────────────────╮
    |       |    │      Welcome to the      │
    |--(o)--|    │  SharePoint Client-side  │
   `---------´   │    Solution Generator    │
    ( _´U`_ )    ╰──────────────────────────╯
    /___A___\
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

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

Yeoman has started working on the scaffolding of the project. It will install the required dependencies and scaffold the solution files for the

'CustomList' Web part, which will take some time to complete. Once completed, we will get a congratulations message.

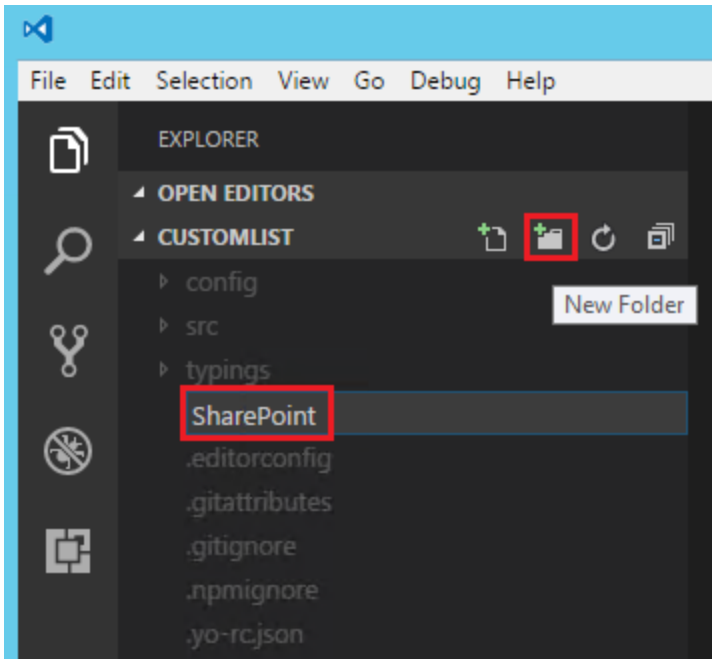
```
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.0.0 (node_modules\ch
okidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@
1.1.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"}
)
npm WARN optional SKIPPING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

        _=+#####!
   ############!        .-----------------------------------.
   ###/     (##!(@)     |          Congratulations!         |
   ###  #######!   \    |  Solution custom-list is created. |
   ###/    /###!   (@)  |   Run gulp serve to play with it! |
   #######  ##!    /    '-----------------------------------'
   ###      /##!(@)
   ############!
      **=+#####!


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

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

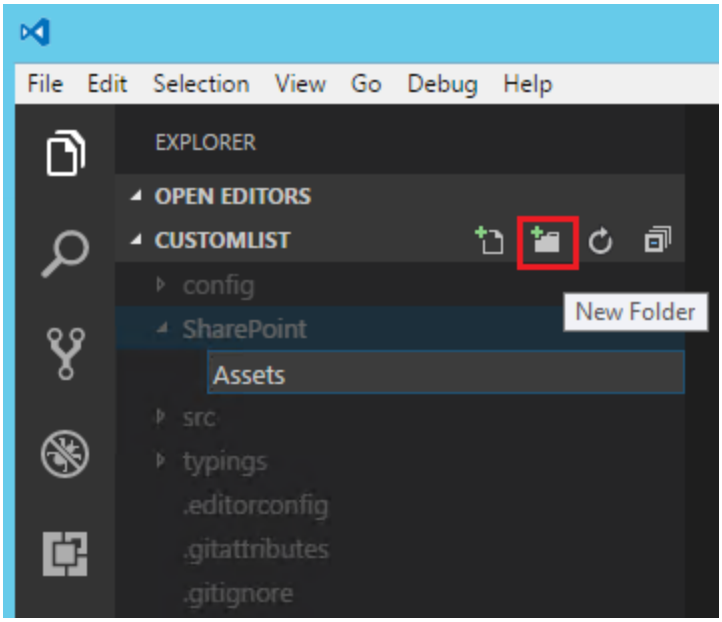


Edit the Web Part

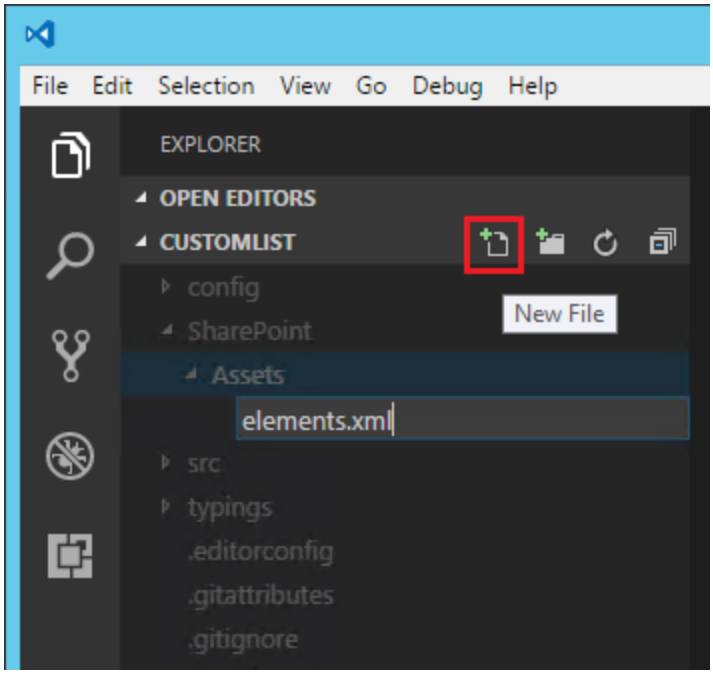
Now let's add the folder named 'SharePoint' to maintain the SharePoint files that will be deployed as a package.



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

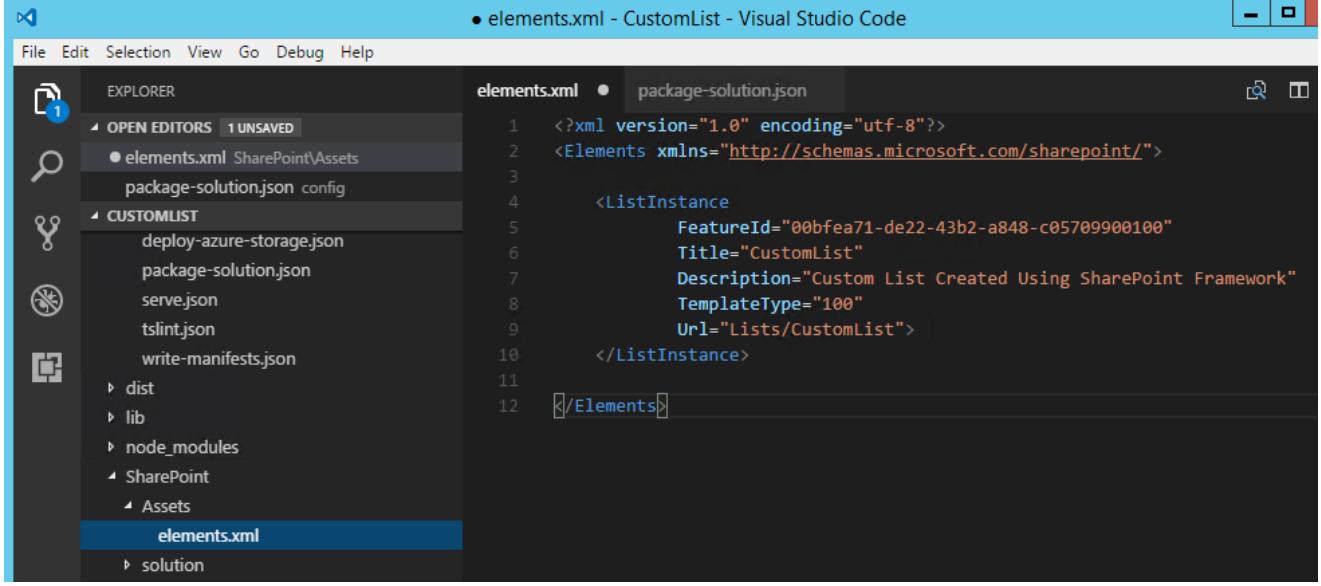


We will be creating an XML file - elements.xml which will hold the information required to provision the list. Let's create the first supporting XML file elements.xml.



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. Click here to view it raw.

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

    <ListInstance
            FeatureId="00bfea71-de22-43b2-a848-c05709900100"
            Title="CustomList"
            Description="Custom List Created Using SharePoint Framework"
            TemplateType="100"
            Url="Lists/CustomList">
    </ListInstance>

</Elements>
```



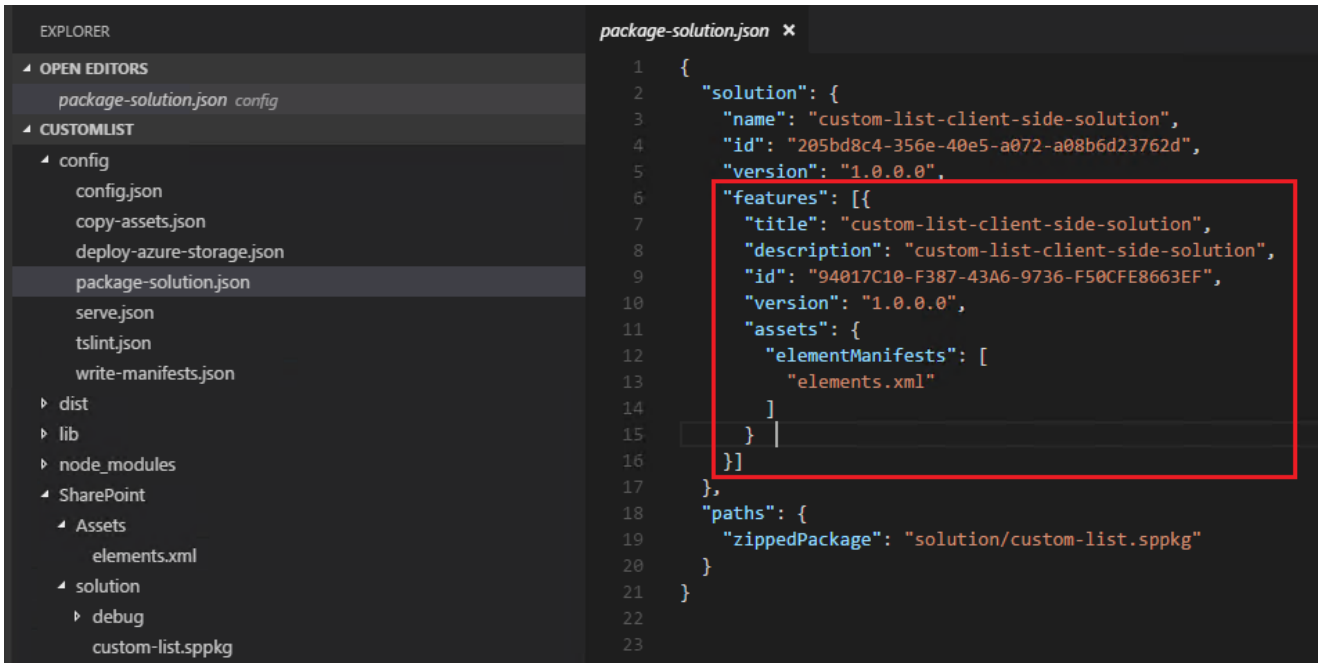
Thus we have created the solution structure and added the list information in the XML file which will be automatically provisioned when the solution is deployed. However, we have to make sure that this elements.xml file is packaged as part of the solution. In the older SharePoint Server-side solutions, we would be using Features to implement this. Similarly, we have a Feature Framework for SPFx as well. In order to do this, we will be adding a feature definition in the package-solution.json file in the configuration folder. To view it raw, click here

```
"features": [{
      "title": "custom-list-client-side-solution",
      "description": "custom-list-client-side-solution",
      "id": "94017C10-F387-43A6-9736-F50CFE8663EF",
      "version": "1.0.0.0",
      "assets": {
        "elementManifests": [
          "elements.xml"
        ]
      }
    }]
```

After adding the feature definition in the package-solution.json, it will look like below:

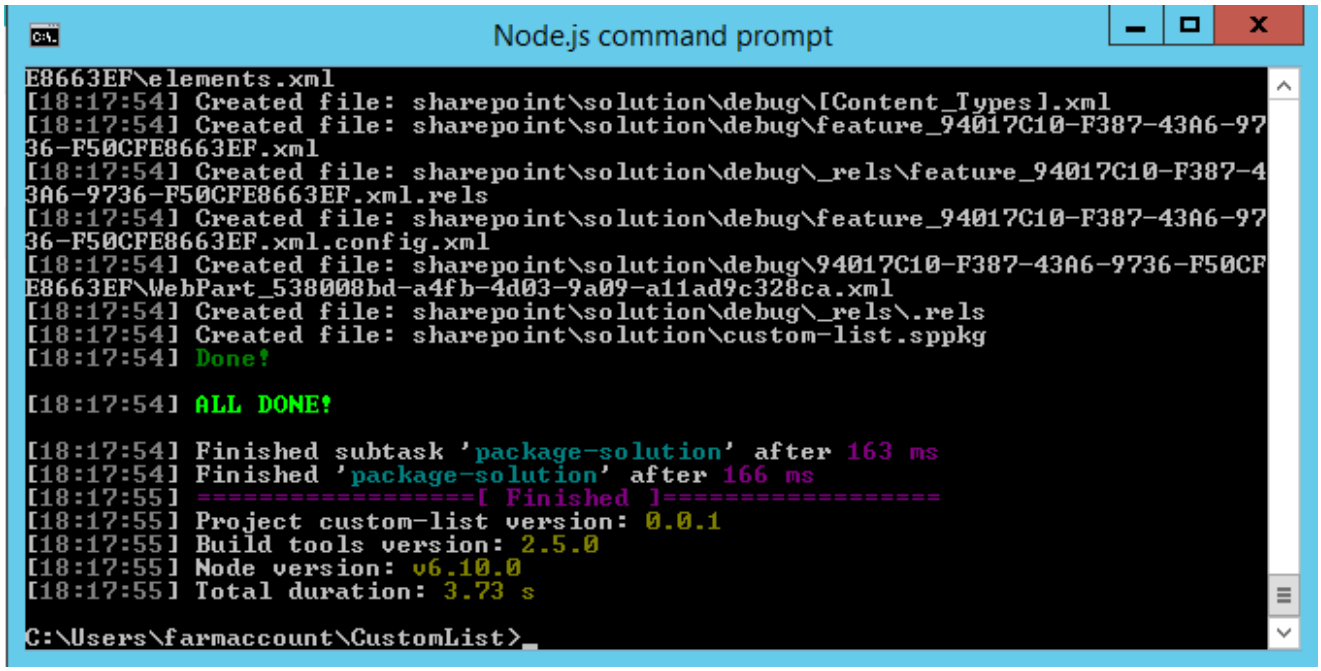
```json
{
  "solution": {
    "name": "custom-list-client-side-solution",
    "id": "205bd8c4-356e-40e5-a072-a08b6d23762d",
    "version": "1.0.0.0",
    "features": [{
      "title": "custom-list-client-side-solution",
      "description": "custom-list-client-side-solution",
      "id": "94017C10-F387-43A6-9736-F50CFE8663EF",
      "version": "1.0.0.0",
      "assets": {
        "elementManifests": [
          "elements.xml"
        ]
      }
    }]
  },
  "paths": {
    "zippedPackage": "solution/custom-list.sppkg"
  }
}
```

Click here to view it raw.

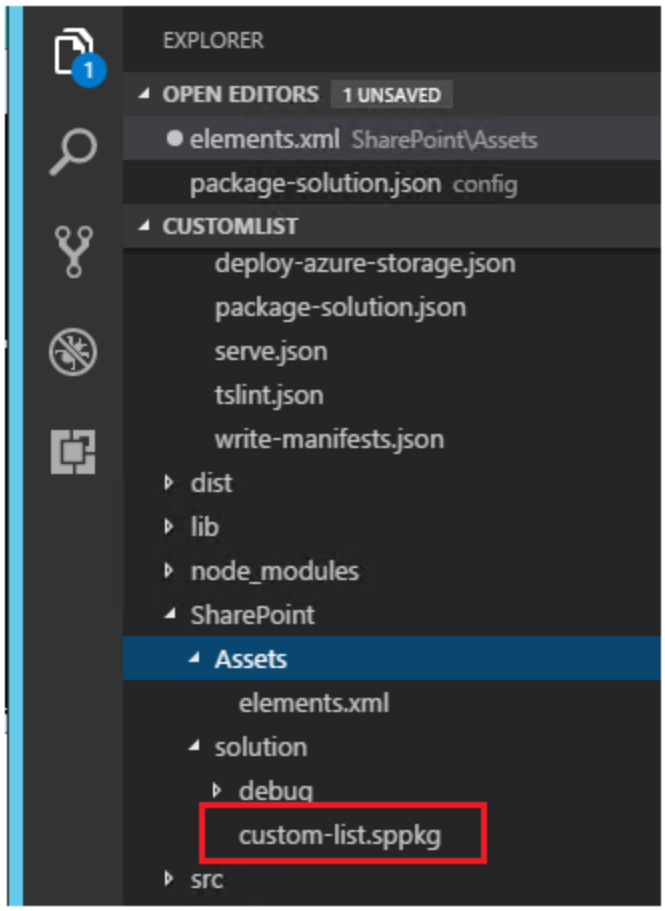


Package and Deploy the Solution

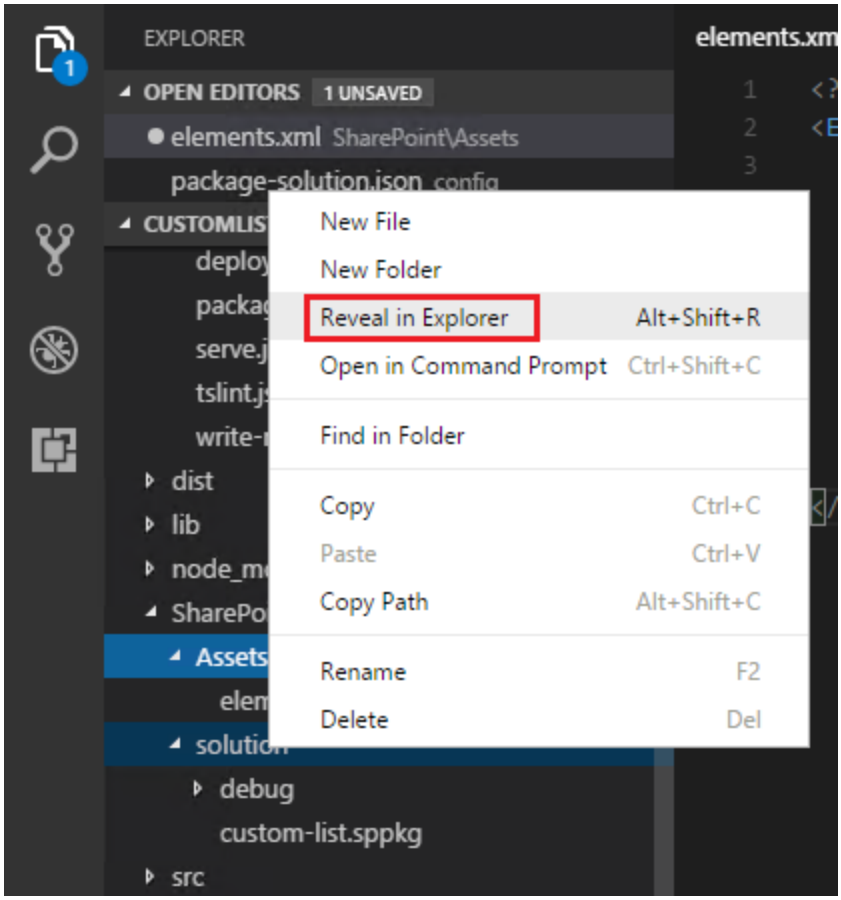
Thus we are done with the addition of the list creation information. Now let's create the deployment package by running gulp serve command from the Node.js command prompt.



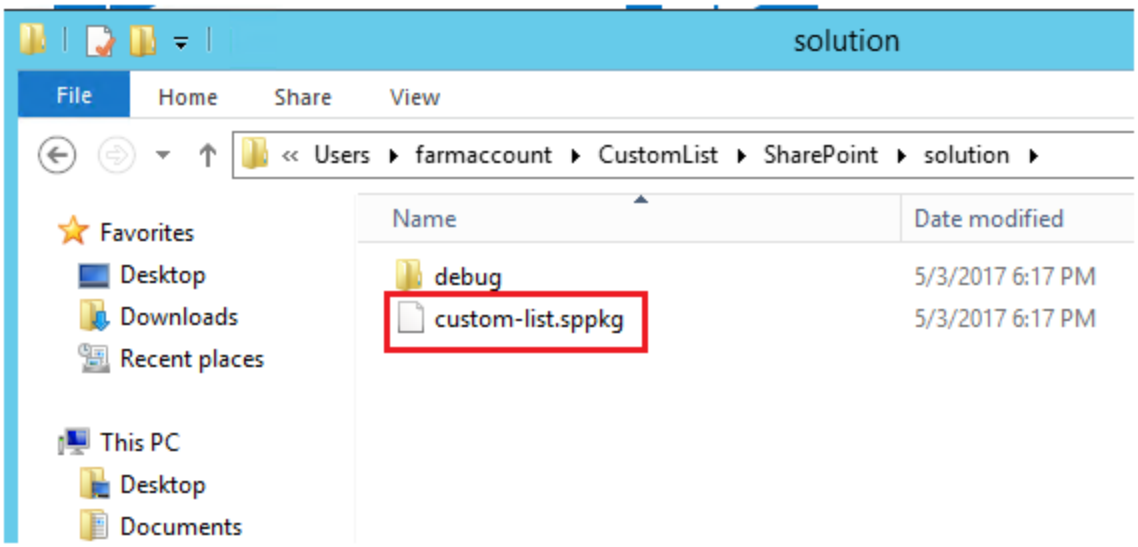
This will create the sppkg package in the solutions folder as shown below.



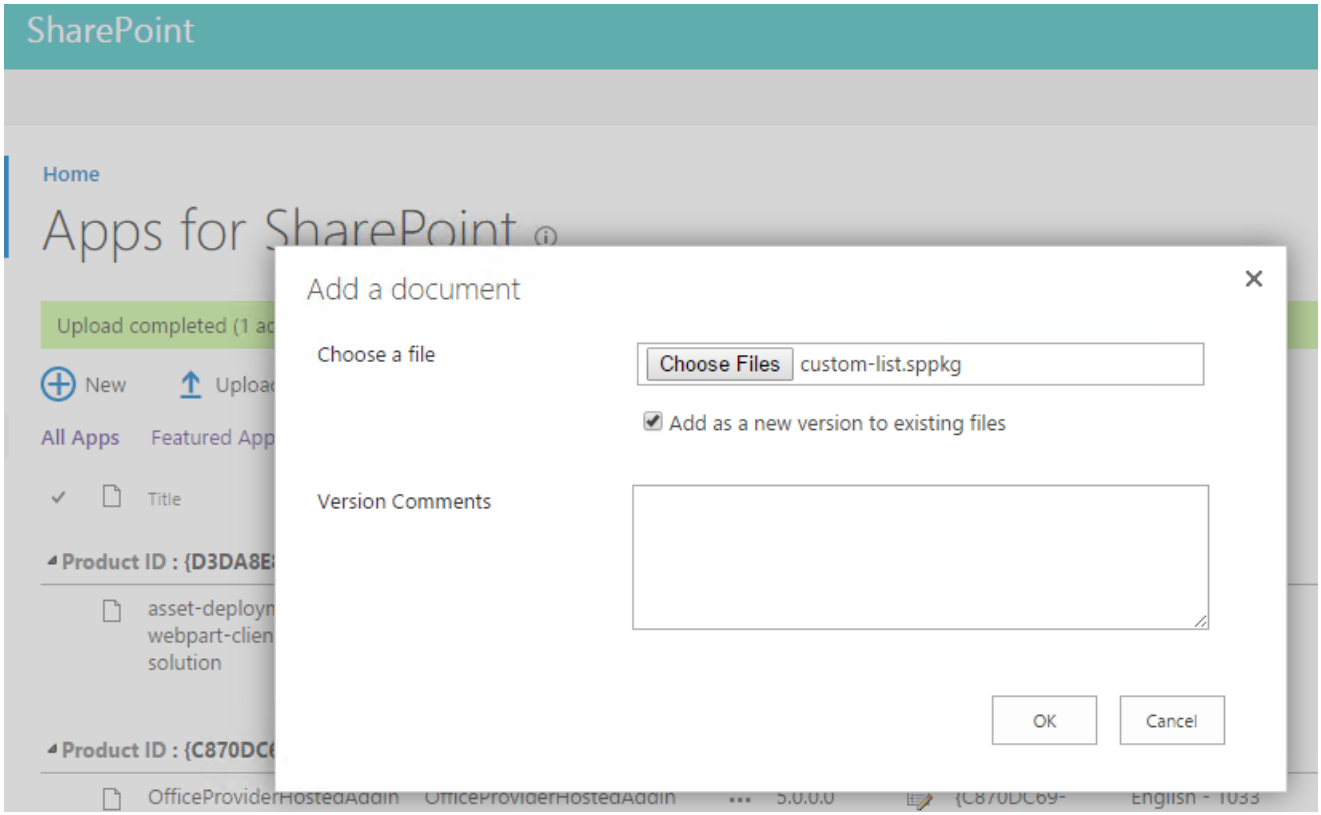
Take a note of the sppkg file url by opening it in File Explorer.



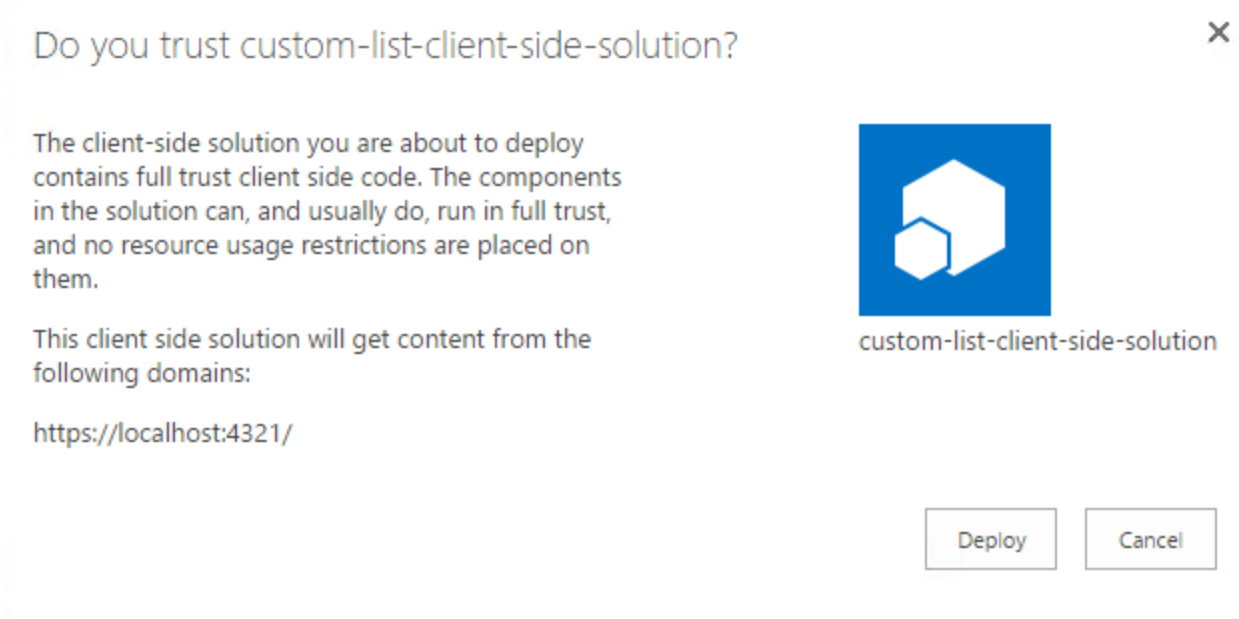
We will be uploading this package to the App Catalog in the next step.



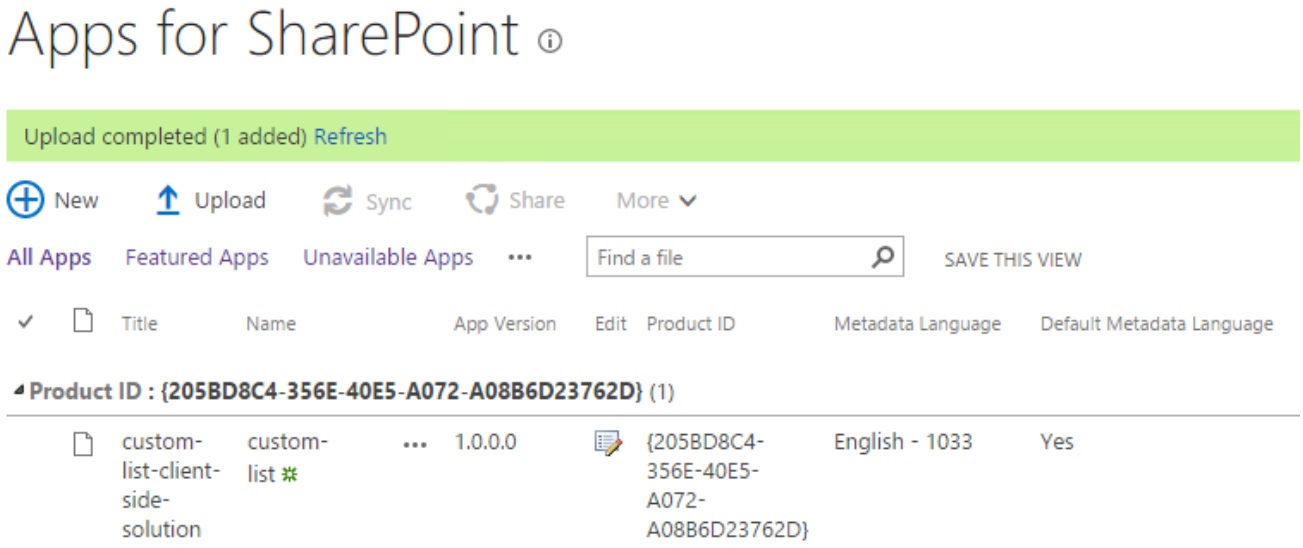
Head over to the App Catalog and upload the package.



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



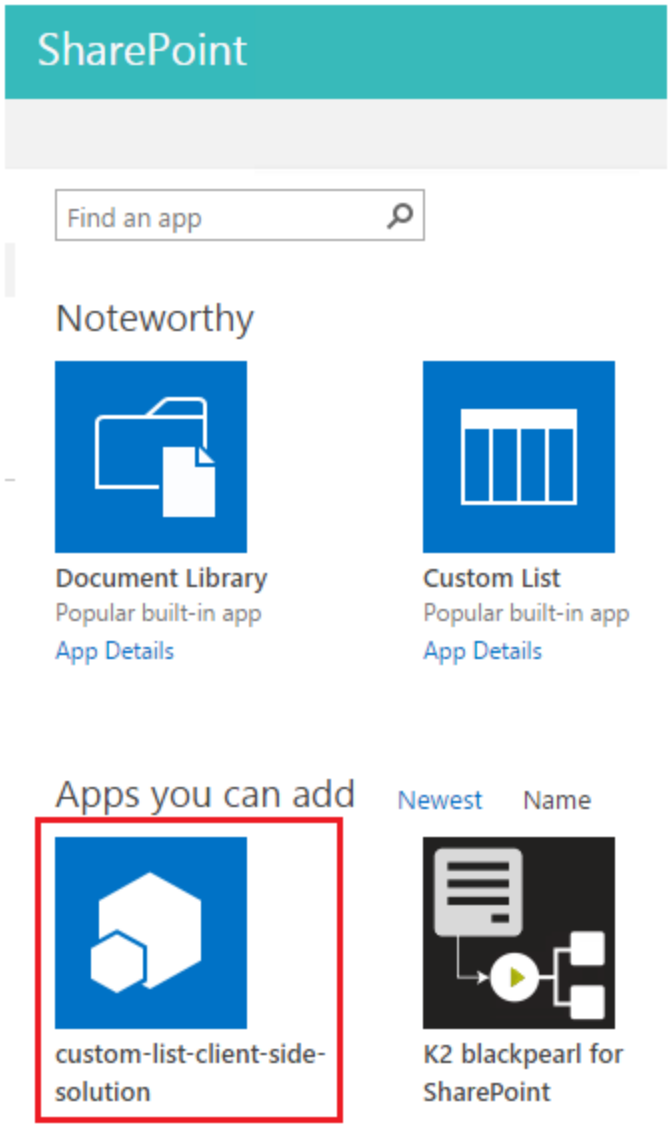
If we refresh the App Catalog page we can see the uploaded solution.



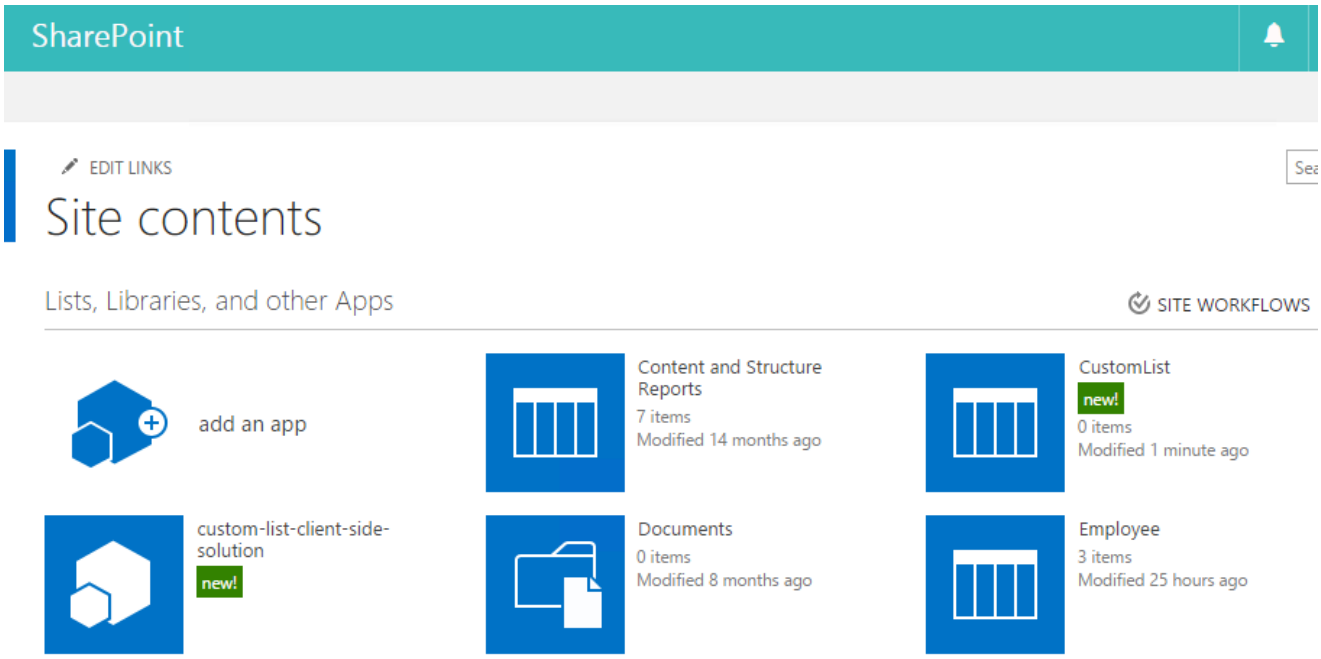
Ensure that there are no errors for the uploaded package by checking the below columns. In case there are some errors, we won't be able to add the solution to the site.

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

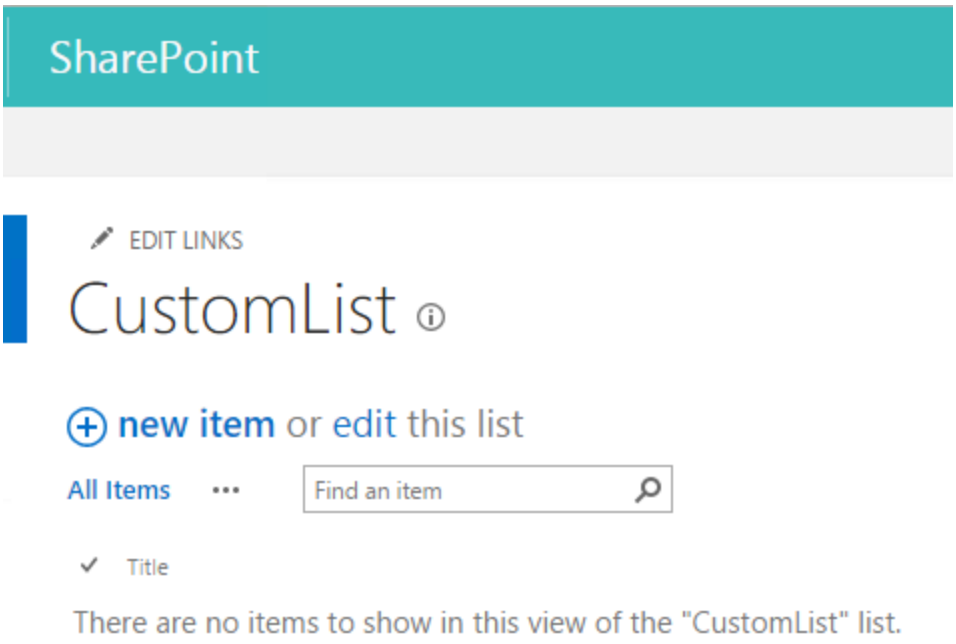
Now if we head over to the site, we can see the newly uploaded solution in the site contents.



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.



Provision Site Columns, Content Type and Custom List using SharePoint Framework

Sharepoint Journey

Become Awesome in SharePoint

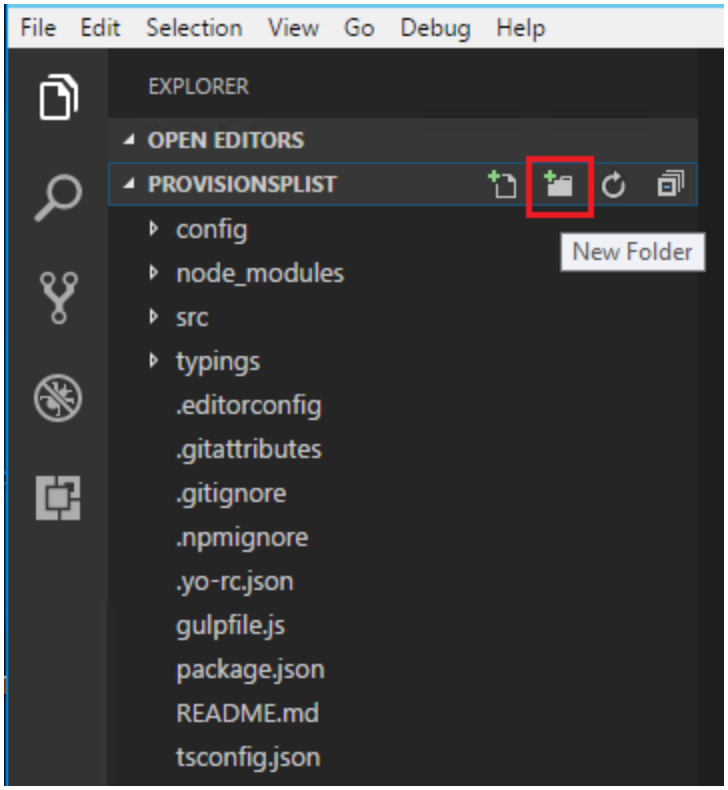
Create the Web Part Project

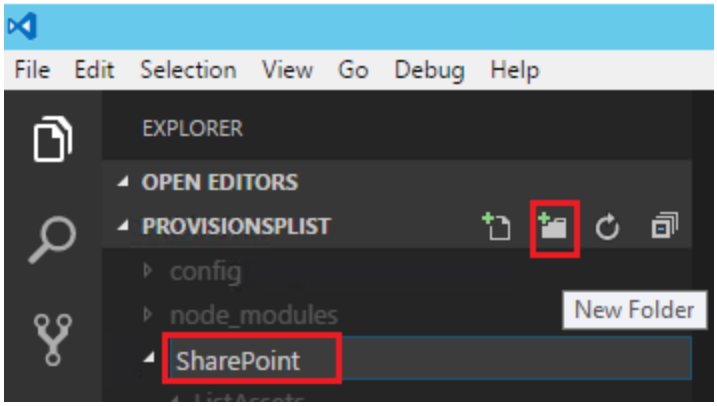
Edit the Web Part

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

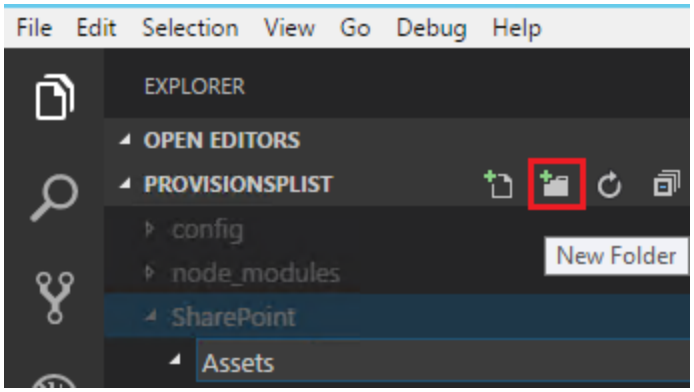
```
C:\Users\farmaccount\ProvisionSPList>code .
```

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

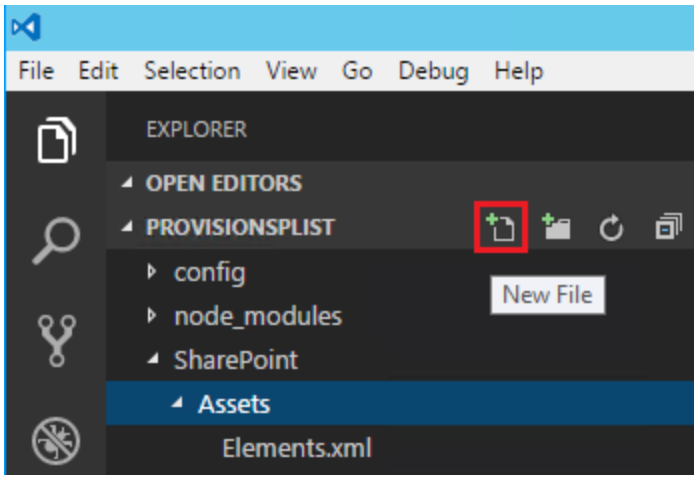




Within the SharePoint folder, let's add another subfolder named Assets.



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

Now let's add 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:

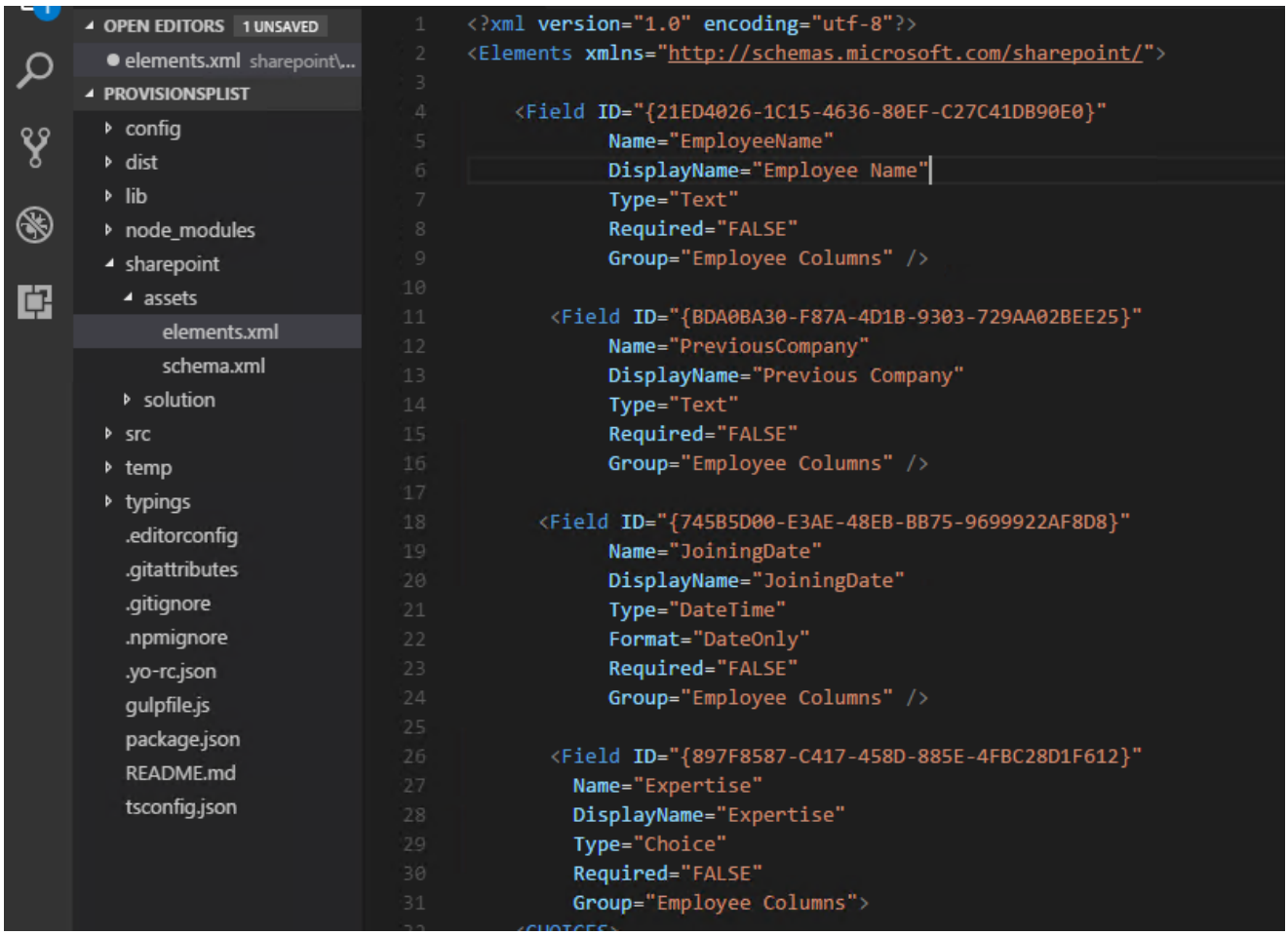
```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

    <Field ID="{11ED4026-1C15-4636-80EF-C27C41DB90E0}"
            Name="EmployeeName"
            DisplayName="Employee Name"
            Type="Text"
            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>
```

You may also click here to view the raw version.

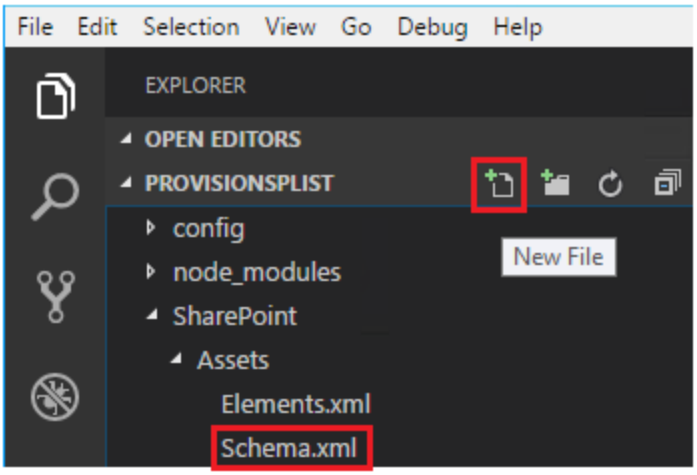


Schema.XML

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

```
        <ContentTypes>
              <ContentTypeRef
ID="0x010100FA0963FA69A646AA916D2E41284FC9D9" />
        </ContentTypes>
```

To view the raw version of this, click here



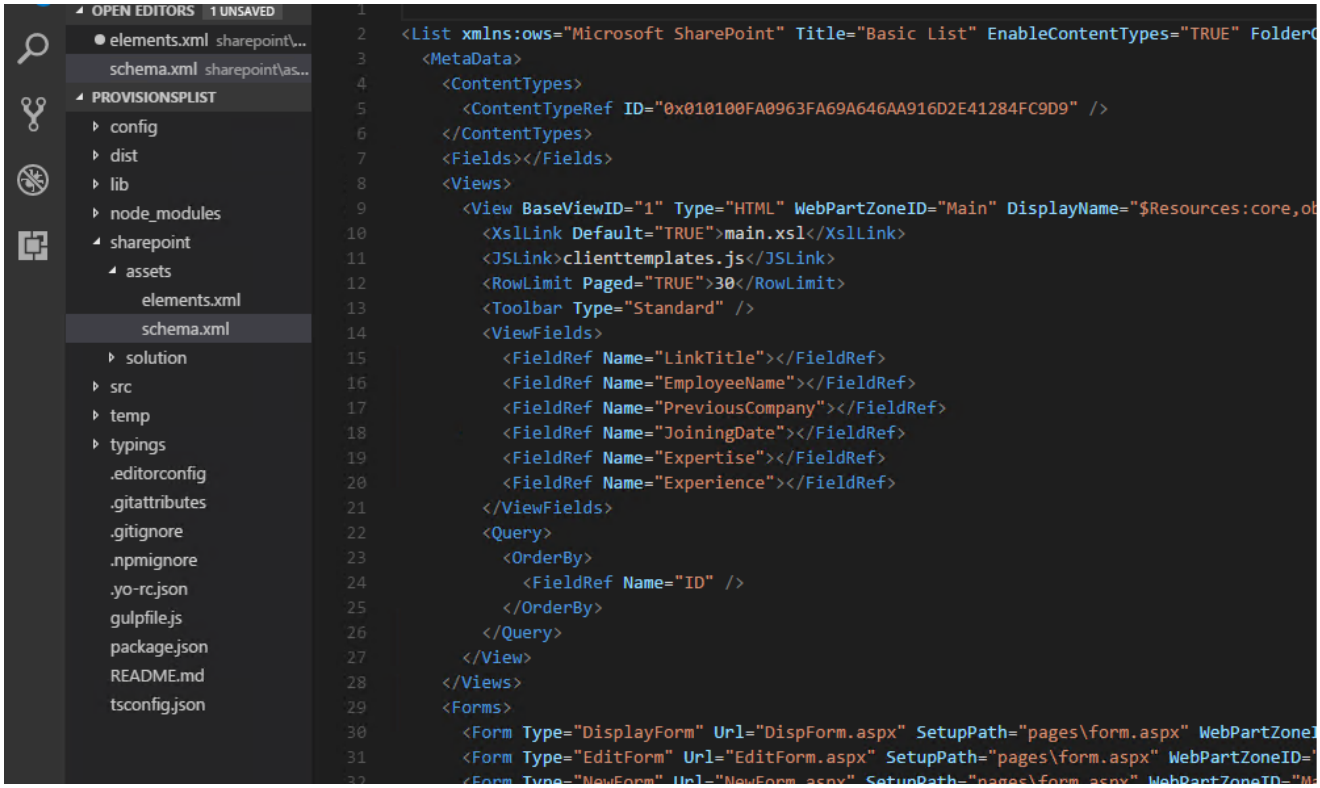
The complete schema.xml will look like below:

```
<List xmlns:ows="Microsoft SharePoint" Title="Basic List"
EnableContentTypes="TRUE" FolderCreation="FALSE"
Direction="$Resources:Direction;" Url="Lists/Basic List"
BaseType="0"


 xmlns="http://schemas.microsoft.com/sharepoint/">
   <MetaData>
```

```
    <ContentTypes>
      <ContentTypeRef
ID="0x010100FA0963FA69A646AA916D2E41284FC9D9" />
    </ContentTypes>
    <Fields></Fields>
    <Views>
      <View BaseViewID="1" Type="HTML" WebPartZoneID="Main"
DisplayName="$Resources:core,objectiv_schema_mwsidcamlidC24
;" DefaultView="TRUE" MobileView="TRUE"
MobileDefaultView="TRUE" SetupPath="pages\viewpage.aspx"
ImageUrl="/_layouts/images/generic.png"
Url="AllItems.aspx">
        <XslLink Default="TRUE">main.xsl</XslLink>
        <JSLink>clienttemplates.js</JSLink>
        <RowLimit Paged="TRUE">30</RowLimit>
        <Toolbar Type="Standard" />
        <ViewFields>
          <FieldRef Name="LinkTitle"></FieldRef>
          <FieldRef Name="EmployeeName"></FieldRef>
          <FieldRef Name="PreviousCompany"></FieldRef>
          <FieldRef Name="JoiningDate"></FieldRef>
          <FieldRef Name="Expertise"></FieldRef>
          <FieldRef Name="Experience"></FieldRef>
        </ViewFields>
        <Query>
          <OrderBy>
            <FieldRef Name="ID" />
          </OrderBy>
        </Query>
      </View>
    </Views>
    <Forms>
      <Form Type="DisplayForm" Url="DispForm.aspx"
SetupPath="pages\form.aspx" WebPartZoneID="Main"

 />
      <Form Type="EditForm" Url="EditForm.aspx"
```

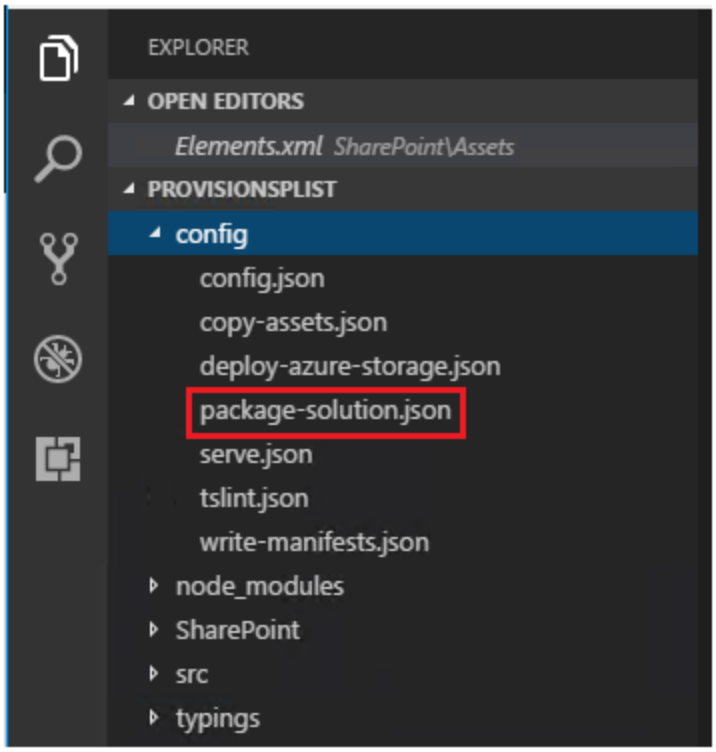
```
 SetupPath="pages\form.aspx" WebPartZoneID="Main"  />
      <Form Type="NewForm" Url="NewForm.aspx"
SetupPath="pages\form.aspx" WebPartZoneID="Main" />
    </Forms>
  </MetaData>
</List>
```

Click here to view the raw version of this.

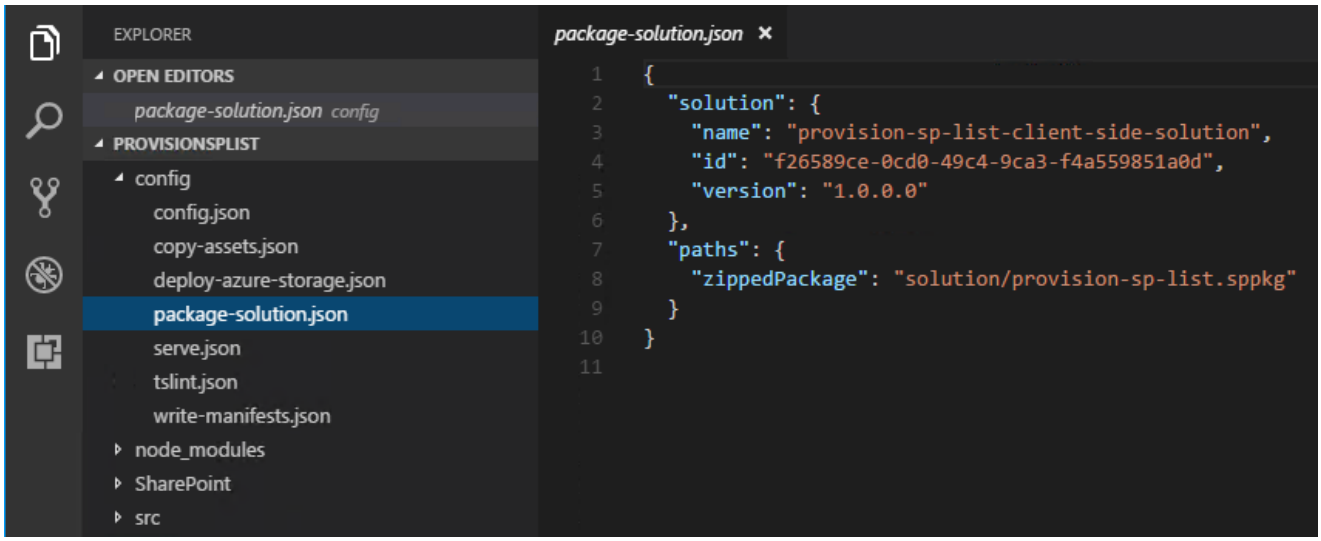


Before we can deploy the package we have to update the feature in the package-solution.json file.

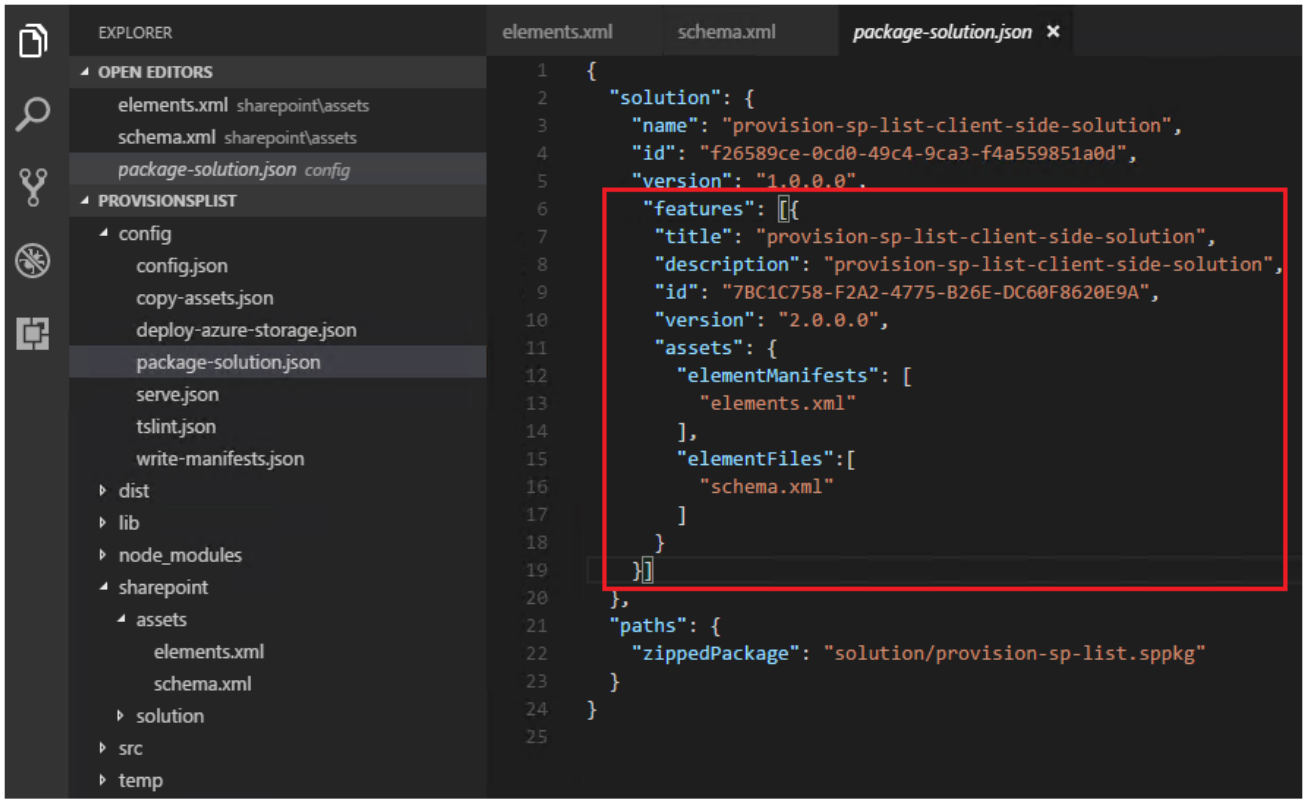
Update Package-Solution.json



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"
  }
}
```

Click here to view the raw version.

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>
```

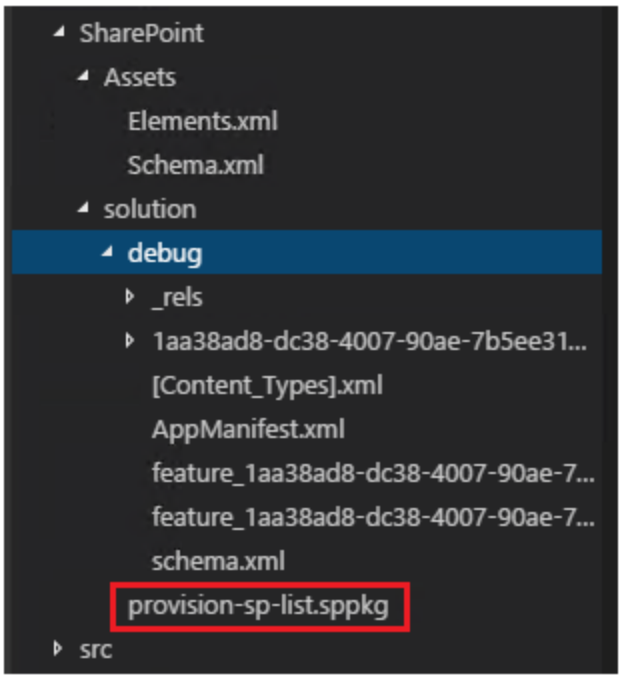
gulp package-solution

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

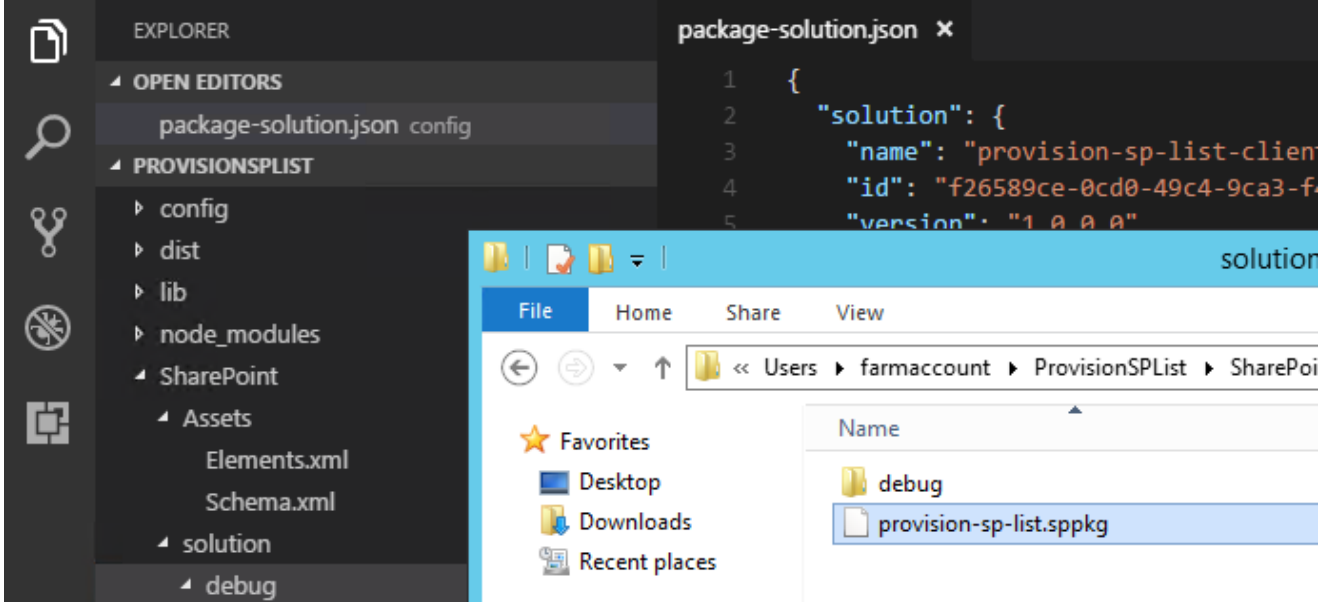
Thus, we are done with the packaging of the solution.

```
[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>_
```

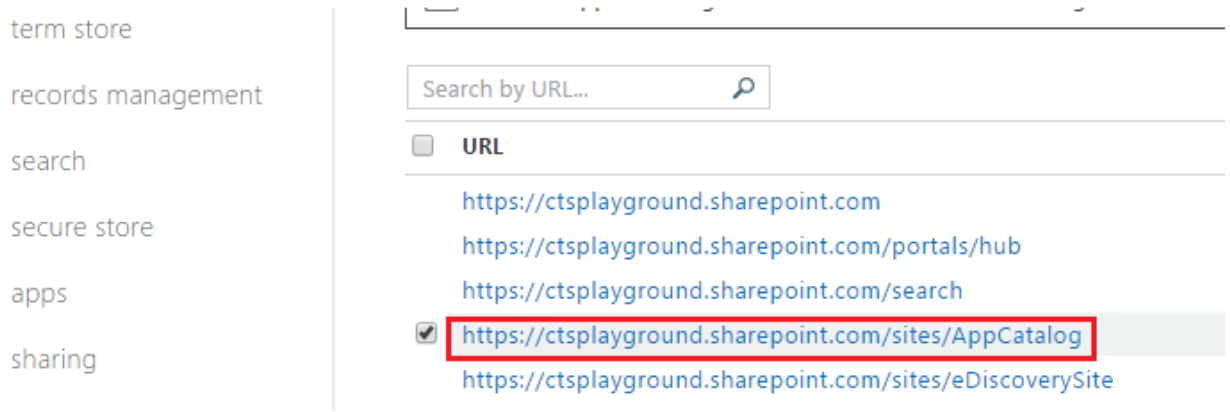
If we head over to the solutions folder, we can see the 'provision-sp-list package' which we will be uploading to SharePoint.



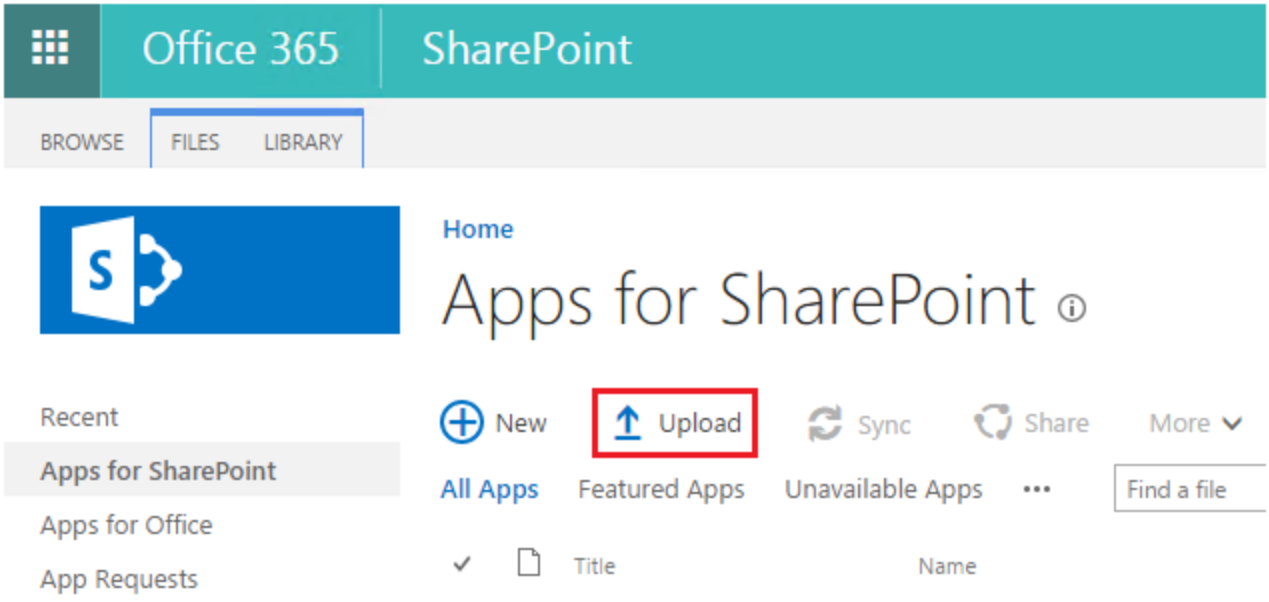
Make a note of the solution URL in the local computer as we will need it to upload to SharePoint.



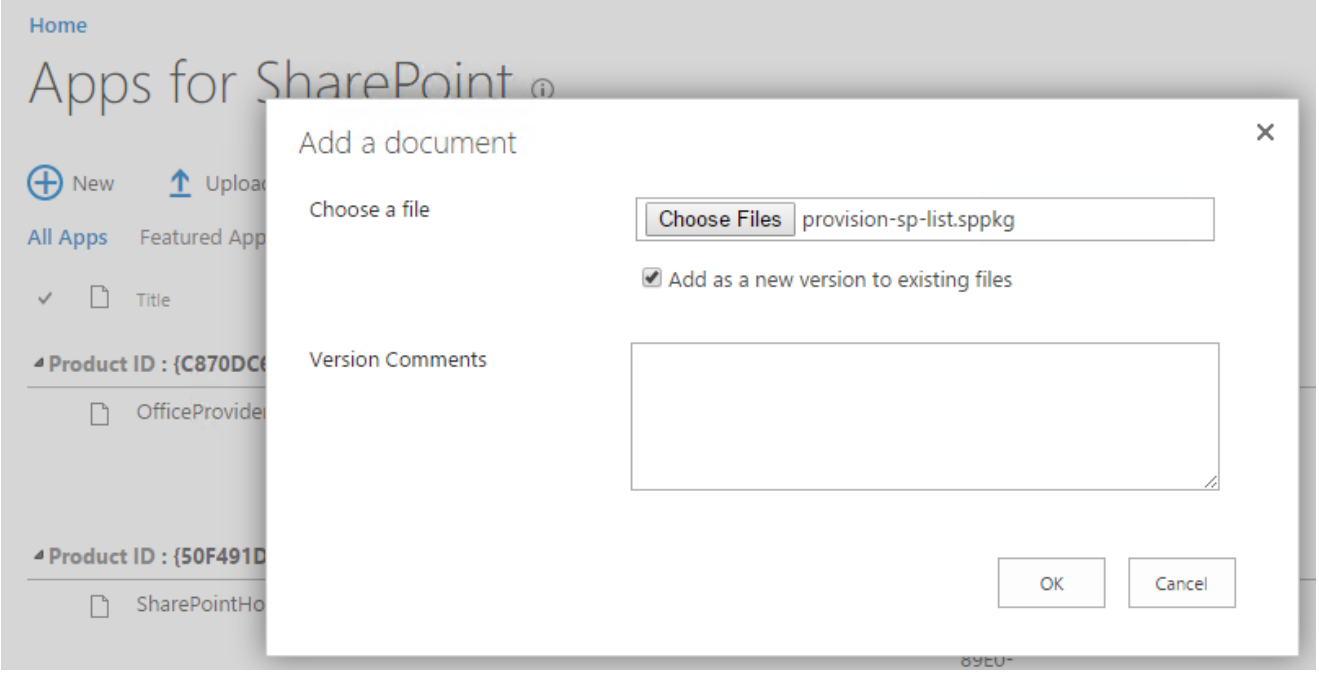
Let's head over to the SharePoint App Catalog site to where we will be uploading the solution.



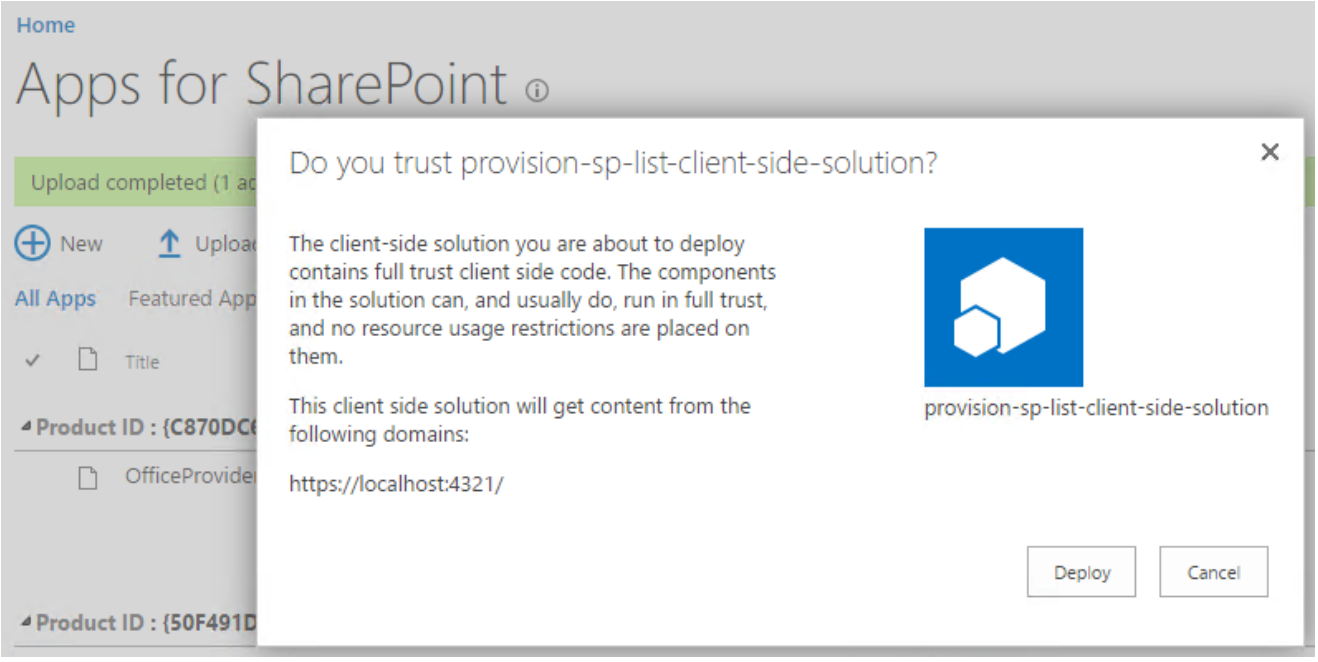
Click on Upload to add the solution file to the site.



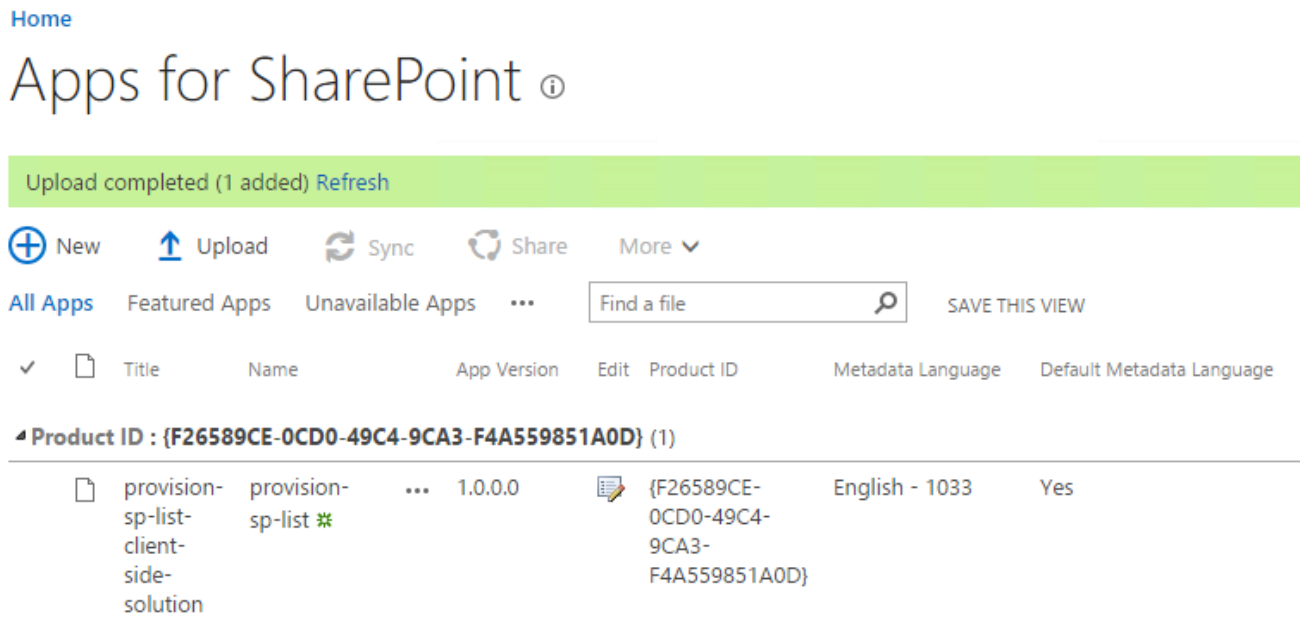
Click on OK to complete the upload.



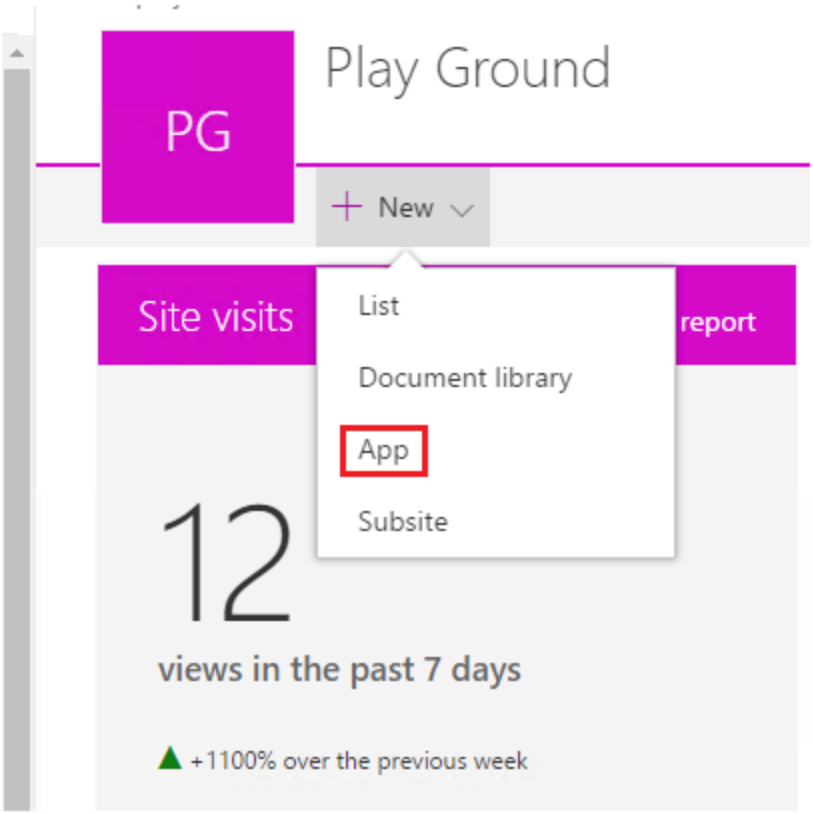
It will ask to trust and Deploy the solution to SharePoint.



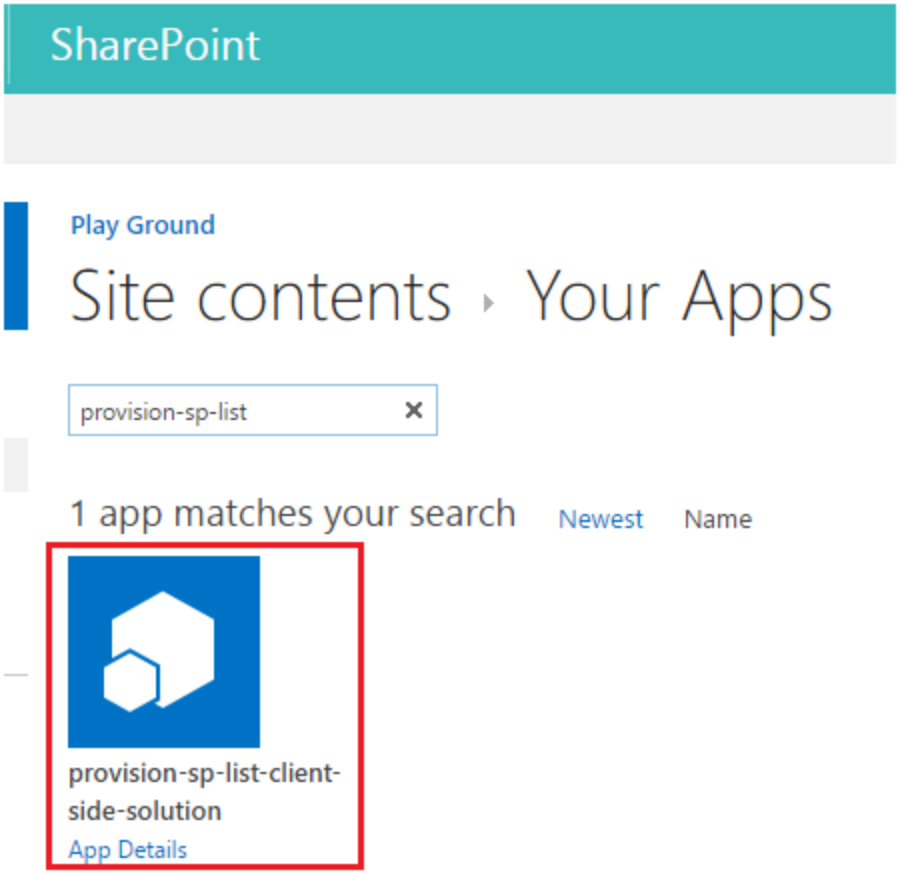
We can see the uploaded solution in the App Catalog.



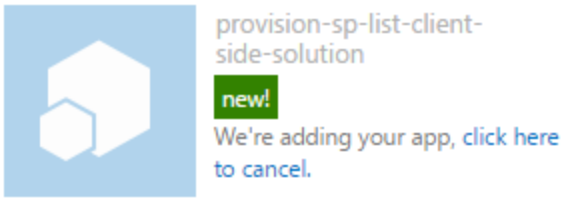
Now let's head over to the site contents and add the solution to the site.



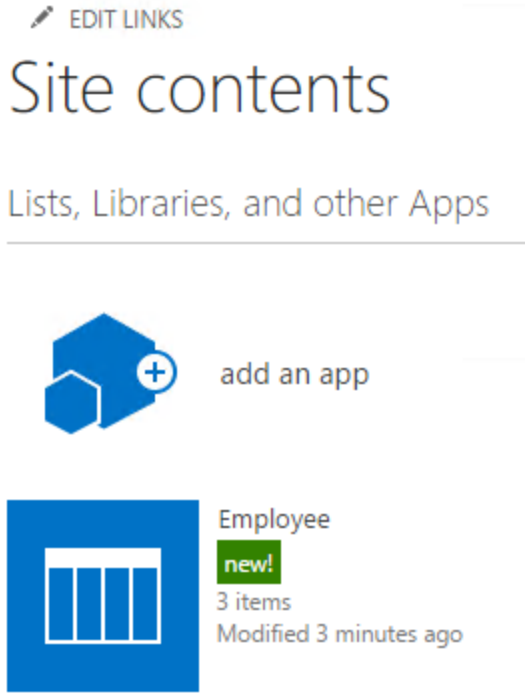
On searching for the deployed app, it will list out the recently added solution.



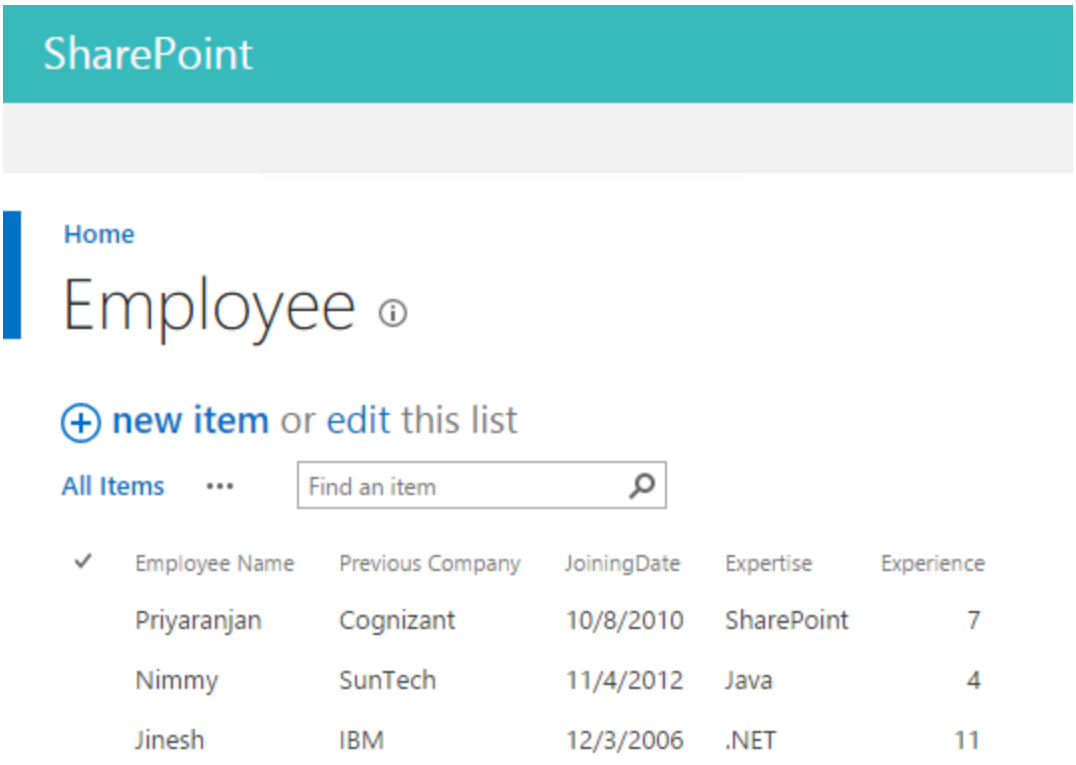
Click on it to add the solution to the site.



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 here. Feel free to download it.

Read more about
SharePoint
Framework

Summary

Thus we saw how to set up the environment for working with SharePoint Framework development model.In the upcoming posts, we will see how to get started with the creation of the first client side web part and test it against the SharePoint Workbench. In the subsequent posts, we will explore how to integrate PnP JS and React JS with SharePoint Framework and use them to build User Profile and Search Web Parts. SharePoint Framework is evolving constantly and more features will be added to it in the coming releases. As of now, it is available for use only with SharePoint Online but Microsoft has promised its roll out for SharePoint On-Premise as part of a future release.