

# Developing Web Parts using SharePoint Framework

**Lab Time:** 60 minutes

**Lab Folder:** C:\Student\Modules\13\_SharePointFramework\Lab

**Lab Overview:** In this lab, you will begin by creating a simple SharePoint Framework project using the Yeoman generator and by editing the code for a simple WebPart using Visual Studio Code. In the next exercise, you will move through the steps of testing your WebPart in the local SharePoint Workbench and the Chrome Debugger extension for Visual Studio Code. The lab will also teach you how to add custom properties to a WebPart and to program a SPFx WebPart against the SharePoint REST API.

**Lab Prerequisite:** This lab assumes you've already installed Node.JS, GIT and Visual Studio Code as described the setup up document for SharePoint Framework provider by your instructor.

## Exercise 1: Create and Test Your First SPFx Web Part

In this exercise, you will install a few Node.JS packages required for SharePoint Framework development including the gulp task runner utility and the Yeoman template generator. After that, you will create a simple SharePoint Framework project containing a single WebPart and begin editing the project's source files with Visual Studio Code.

1. Install the Node.JS packages required for working with SharePoint Framework.
  - a) Launch the Node.JS command prompt.
  - b) Run the following **npm** command to globally install the packages for **gulp** and the Yeoman Generator (**yo**).

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

- c) Run the following **npm** command to globally install the yo template for creating SharePoint Framework projects.

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

- d) Run this

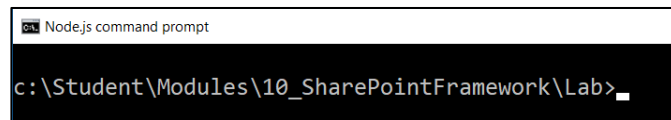
```
npm install -g @microsoft/generator-sharepoint@latest
```

Now you can create a new SPFx project by creating a new empty folder and running the Yeoman generator to create the starter files.

2. Create a new SPFx project named **spfx-lab**.
  - a) From the Node.JS command prompt, run the following command to set your current folder to the folder for this lab.

```
cd C:\Student\Modules\10_SharePointFramework\Lab
```

- b) The current directory for the console should now be at the folder for this lab inside the Student folder.



```
Node.js command prompt
c:\Student\Modules\10_SharePointFramework\Lab>
```

- c) Type the following command and execute it by pressing **Enter** to create a new folder for your project.

```
md spfx-lab
```

- d) Type the following command and execute it by pressing **Enter** to create move the current directory into the new folder.

```
cd spfx-lab
```

- e) The current directory for the console should now be located at the new folder you just created named **spfx-lab**.

```
Select Node.js command prompt

c:\Student\Modules\10_SharePointFramework\Lab>md spfx-lab

c:\Student\Modules\10_SharePointFramework\Lab>cd spfx-lab

c:\Student\Modules\10_SharePointFramework\Lab\spfx-lab>_
```

- f) Type the following command and execute it to launch the Yeoman generator with the SPFx project template.

```
yo @microsoft/sharepoint
```

- g) When prompted with **What is your solution name?**, press **Enter** to accept the default value which is the name of the folder.

```
c:\Student\Modules\10_SharePointFramework\Lab\spfx-lab>yo @microsoft/sharepoint

  _ _ _ _ _
  | _ _ _ |
  | (o)  |
  | _ _ _ |
  |  U  |
  |  A  |
  | _ _ _ |
  |  ~  |
  | _ _ _ |
  |  o  |
  | _ _ _ |
  |  V  |
  | _ _ _ |

Welcome to the
SharePoint Client-side
Solution Generator

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

- h) When prompted with **Which type of client-side component to create?**, press **Enter** to accept the default value of **WebPart**.

```
Let's create a new SharePoint solution.
? What is your solution name? spfx-lab
? Which type of client-side component to create? (Use arrow keys)
> WebPart
  Extension (Preview)
```

- i) When prompted with **What is your WebPart name?**, type **WalmartGreeter** and press **Enter** to submit your value.

```
Let's create a new SharePoint solution.
? What is your solution name? spfx-lab
? Which type of client-side component to create? WebPart
? What is your WebPart name? (HelloWorld) WalmartGreeter ←
```

- j) When prompted with **What is your WebPart description?**, type in a short description and press **Enter**.

```
Let's create a new SharePoint solution.
? What is your solution name? spfx-lab
? Which type of client-side component to create? WebPart
? What is your WebPart name? WalmartGreeter
? What is your WebPart description? (WalmartGreeter description) My first SPFx webpart_
```

- k) When prompted with **Which framework would you like to use?**, press **Enter** to accept **No JavaScript Framework**.

```
? What is your WebPart name? WalmartGreeter
? What is your WebPart description? My first SPFx webpart
? Which framework would you like to use? (Use arrow keys)
> No JavaScript framework ←
  React
  Knockout
```

Once you have answered all the questions, the Yeoman generator will run and add the starter files to your project folder.

- l) Wait until the Yeoman generator completes its work and display a message indicating the new solution has been created..

```
added 1624 packages in 45.897s

_+#####!
#####!
###/  (##) (@)
### ##### \
###/  /###  (@)
#####  ## /
###   /##  (@)
#####!
**_+#####!

c:\Student\Modules\10_SharePointFramework\Lab\spfx-lab>
```

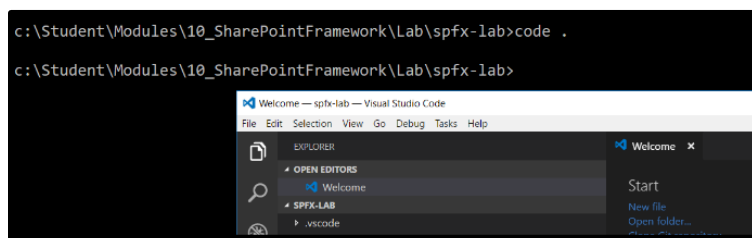
Congratulations!  
Solution **spfx-lab** is created.  
Run **gulp serve** to play with it!

3. Open the project with Visual Studio Code

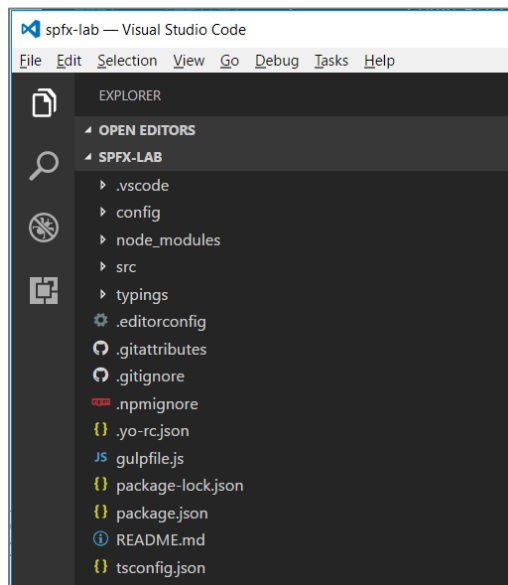
- a) Type the following command and execute it by pressing **Enter** to open your new project in Visual Studio Code.

**code .**

- b) As the command executes, it should open your new project folder with Visual Studio Code.

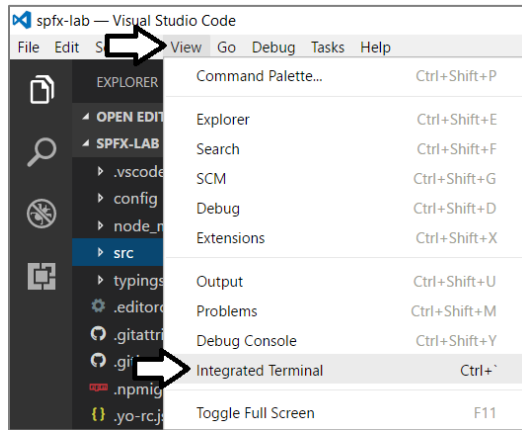


- c) Take a moment to familiarize yourself with the files and folders at the root of the **spfx-lab** project.

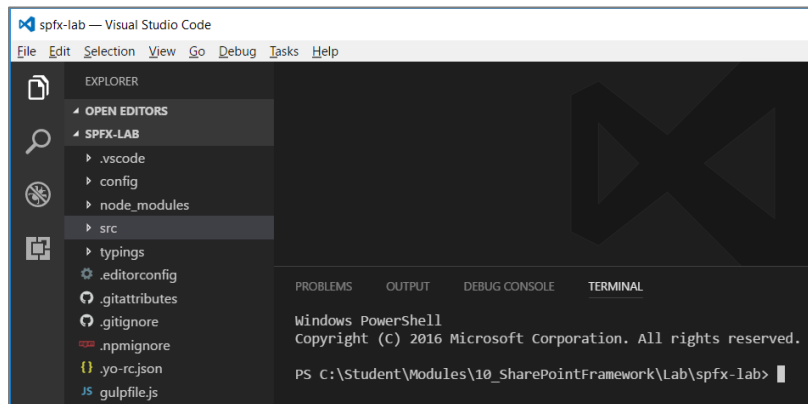


Several of these project files such as **package.json**, **tsconfig.json** and **gulpfile.js** should already be familiar to you from the work you did in the previous lab titled *Developing with Node.JS, NPM and Visual Studio Code*.

4. Open the console window from the Integrated Terminal.
  - a) Use the **View > Integrated Terminal** menu command in Visual Studio Code to display the Integrated Terminal.



- b) The Integrated Terminal should provide a console with its current directory located at your project folder.

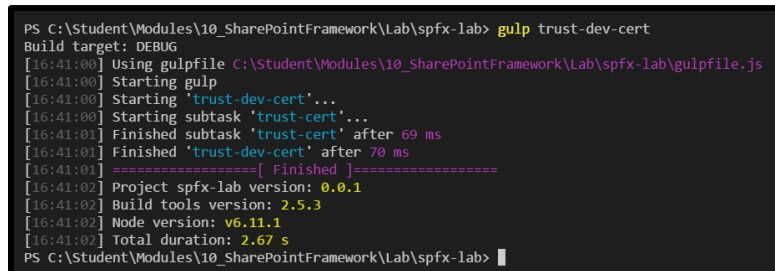


Now you have the ability to run npm command and gulp commands from within Visual Studio Code.

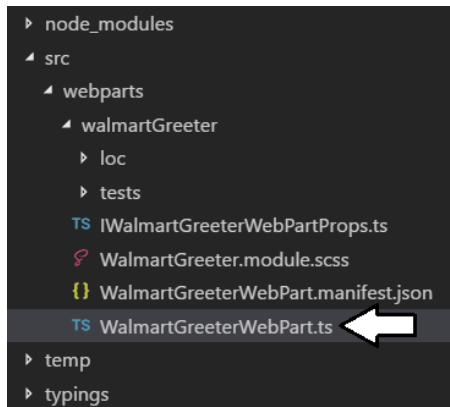
5. Run the gulp task named **trust-dev-cert** to configure your project with an SSL certificate for testing the project at <https://localhost>.
  - a) Type and execute the following command to execute the gulp task named **trust-dev-cert** that is provided by SPFx.

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

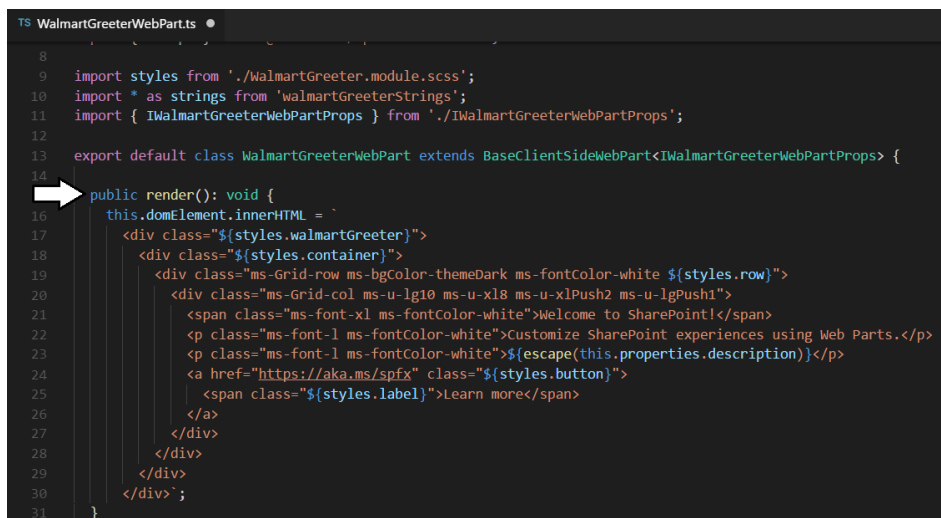
- b) Verify that the **trust-dev-cert** gulp task executes successfully.



6. Update the starter TypeScript code for the WebPart class definition inside **WalmartGreeterWebPart.ts**.
- a) Inside the **src/webparts/walmartGreeter** folder, locate and open the TypeScript file named **WalmartGreeterWebPart.ts**.



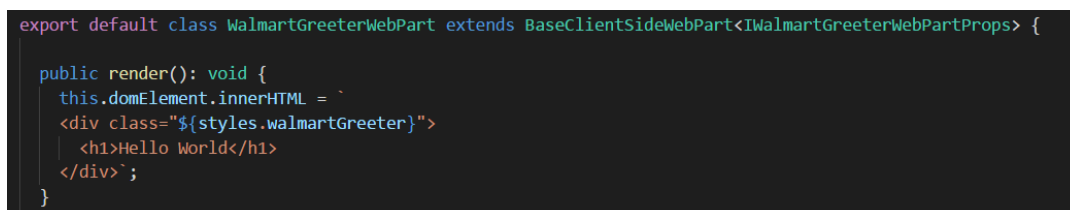
- b) You should see a TypeScript definition for a class named **WalmartGreeterWebPart**.
- c) Inside the **WalmartGreeterWebPart** class, locate the implementation of **render** method.



- d) Replace the existing **render** method implementation using the following code.

```
public render(): void {
    this.domElement.innerHTML = `
        <div class="${styles.walmartGreeter}">
            <h1>Hello world</h1>
        </div>`;
}
```

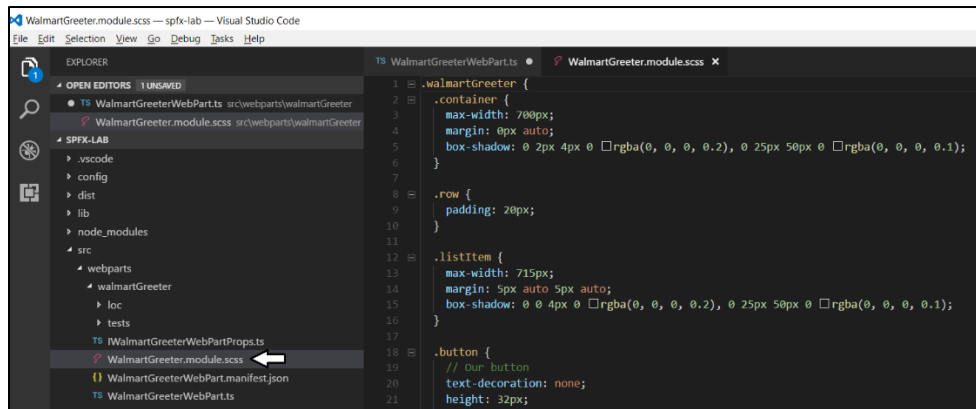
- e) The **render** method of the **WalmartGreeterWebPart** class should now match the following screenshot.



- f) Save your changes to **WalmartGreeterWebPart.ts** and leave this file open.

7. Modify the CSS styles in the SCSS file named **WalmartGreeter.module.scss**.

- a) Inside the **src/webparts/walmartGreeter** folder, locate and open the SCSS file named **WalmartGreeter.module.scss**.



- b) Delete all the existing content inside **WalmartGreeter.module.scss**.  
c) Add the following CSS code to **WalmartGreeter.module.scss**.

```
.walmartGreeter {  
  max-width: 700px;  
  border: 2px solid black;  
  border-radius: 12px;  
  background-color: lightyellow;  
  padding: 12px;  
}
```

So far, you have just added styles that are valid in an CSS file. The real advantage to using stylistically awesome style sheets (SASS) such as to **WalmartGreeter.module.scss** is that they provide syntactic features not available in standard CSS files such as the use of variables and nested classes which improve productivity and maintainability. You will now update **WalmartGreeter.module.scss** using special SASS syntax that is not allowed in standard CSS file.

- d) Add a nested class inside the **walmartGreeter** class to style **h1** elements as shown in the following code listing.

```
.walmartGreeter {  
  max-width: 700px;  
  border: 2px solid darkblue;  
  border-radius: 12px;  
  background-color: lightyellow;  
  padding: 12px;  
  
  h1{  
    color: darkblue;  
    font-size: 2.5em;  
  }  
}
```

- e) Add two new variables to the top of **WalmartGreeter.module.scss** named **\$background-color** and **\$font-color**.

```
$background-color: lightyellow;  
$font-color: darkblue;
```

- f) Update the **background-color** property of the **walmartGreeter** class to use the variable named **\$background-color**.

```
background-color: $background-color;
```

- g) Update the **color** property of the **h1** class to use the variable named **\$font-color**.

```
h1{  
  color: $font-color;  
  font-size: 2.5em;  
}
```

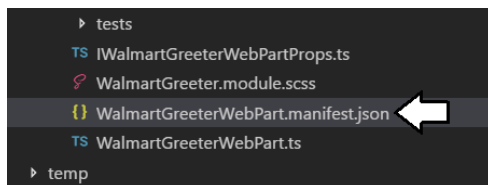
- h) At this point, the contents of **WalmartGreeter.module.scss** should match the following code listing.

```
$background-color: lightyellow;
$font-color: darkblue;

.walmartGreeter {
  max-width: 700px;
  border: 2px solid black;
  border-radius: 12px;
  background-color: $background-color;
  padding: 12px;

  h1{
    color: $font-color;
    font-size: 2.5em;
  }
}
```

- i) Save your changes to **WalmartGreeter.module.scss** and leave this file open.
8. Update the manifest file for the Walmart Greeter WebPart.
- a) Inside **src/webparts/walmartGreeter**, open the WebPart manifest file named **WalmartGreeterWebPart.manifest.json**.



- b) At the bottom of file named **WalmartGreeterWebPart.manifest.json**, locate the **preconfiguredEntries** section.

```
"preconfiguredEntries": [{
  "groupId": "8aee034e-29cc-4c44-8490-fc96a9175734",
  "group": { "default": "Under Development" },
  "title": { "default": "WalmartGreeter" },
  "description": { "default": "My First SPFx WebPart" },
  "officeFabricIconFontName": "Page",
  "properties": {
    "description": "WalmartGreeter"
  }
}]
```

- c) Inside the **preconfiguredEntries** section, modify the **default** value of **title** from **WalmartGreeter** to **Walmart Greeter**.

```
"title": { "default": "Walmart Greeter" },
```

- d) Modify the value of **officeFabricIconFontName** from **Page** to **Emoji2**.

```
"officeFabricIconFontName": "Emoji2",
```

- e) Your edits should match what is shown in the following screenshot.

```
"preconfiguredEntries": [{
  "groupId": "8aee034e-29cc-4c44-8490-fc96a9175734",
  "group": { "default": "Under Development" },
  "title": { "default": "Walmart Greeter" },
  "description": { "default": "My First SPFx WebPart" },
  "officeFabricIconFontName": "Emoji2",
  "properties": {
    "description": "WalmartGreeter"
  }
}]
```

- f) Save your changes to **WalmartGreeterWebPart.manifest.json** and leave this file open.

Now you have done enough initial work on the project to test it for the first time.

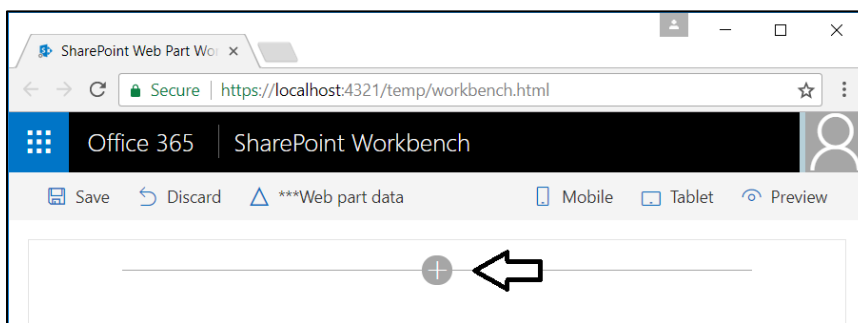
## Exercise 2: Test and Debug Your WebPart in the Local SharePoint Workbench

In the previous lab exercise, you created a new SharePoint Framework project and you modified its source files to prepare it for testing. In this exercise, you will learn how to run your project by serving it up through a local web server and testing your WebPart in the local SharePoint Workbench. Along the way, you will also learn how to configure client-side debugging support for your project using the Chrome Debugger extension for Visual Studio Code.

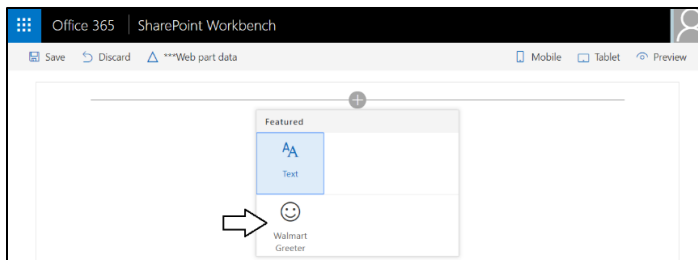
1. Test out the **spfx-lab** project by running it in the local SharePoint workbench
  - a) Navigate to the console of the Integrated Terminal.
  - b) Execute the **gulp serve** command to start up the project and test it out using the local workbench.

```
gulp serve
```

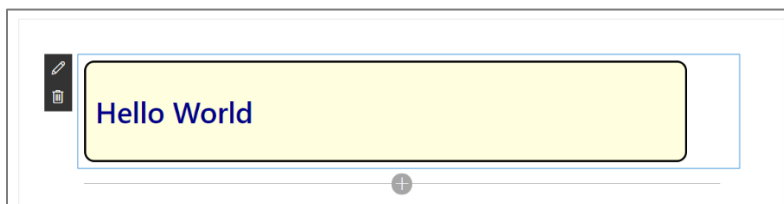
- c) The browser should launch and display a page for adding modern WebParts like the one shown in the following screenshot. Click on the button with the **+** sign in the middle of the page to add your WebPart to the page so you can test it.



- d) Select the **Walmart Greeter** to add it to the page as a new SPFx WebPart.



- e) The WebPart should appear display the text "Hello World".



- f) Click the **Preview** button to transition the page from edit mode to preview mode.

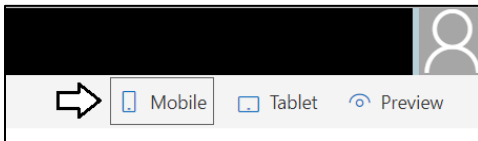




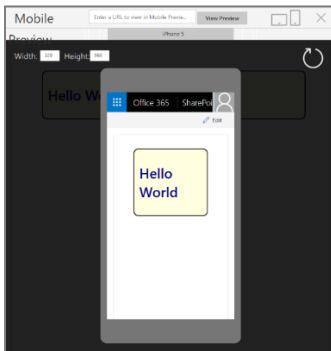
- g) Click on the **Edit** button to move the page back into edit mode.



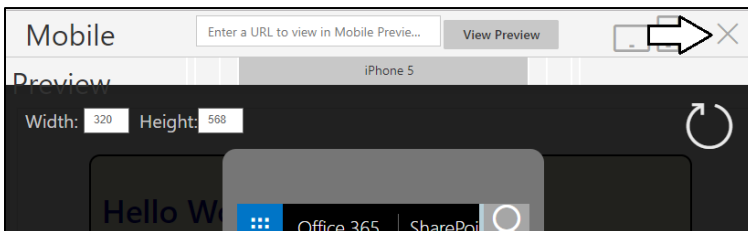
- h) Click on the **Mobile** button to see a simulation of what your WebPart looks when viewed using a mobile device.



- i) You should now see a view of the WebPart as it will look when viewed with a mobile device such as an iPhone.



- j) Once you have seen the Web Part in the mobile view, click the **X** at the top right of the mobile view dialog to close it.



- k) You should now be looking at the WebPart in Edit view.
2. Modify the WebPart's source files and observe the file watching behavior that automatically updates the WebPart in the browser.
- a) Return to the editor window for the TypeScript file named **WalmartGreeterWebPart.ts**.
- b) Locate the **render** method.
- c) Modify the text inside the **h1** element from "**Hello World**" to "**Hello World of SPFx WebParts**".

```
public render(): void {
  this.domElement.innerHTML = `
    <div class="${styles.walmartGreeter}">
      <h1>Hello world of SPFx WebParts</h1>
    </div>;
}
```

- d) Save your changes to **WalmartGreeterWebPart.ts**.
- e) Return to the browser and verify that the WebPart has been automatically updated with the new text for the **h1** element.
- f) Return to the editor window for the SCSS file named **WalmartGreeter.module.scss**.

- g) Modify the two new variables named **\$background-color** and **\$font-color** to use different colors.

```
$background-color: lightyellow;  
$font-color: darkblue;
```

- h) Save your changes to **WalmartGreeter.module.scss**.  
i) Return to the browser and verify that the WebPart has been automatically updated with the new colors.

3. Stop the web server process for the current debugging session.

- a) Return to the console in the Integrated Terminal.  
b) Make sure the console is the active window  
c) Press the **Ctrl + C** keyboard combination to stop the web server from running.

```
Request: '/node_modules/@microsoft/sp-webpart-base/dist/sp-webpart-base_en-u  
Request: '/node_modules/@microsoft/sp-webpart-workbench/dist/sp-webpart-work  
Request: '/lib/webparts/walmartGreeter/loc/en-us.js'  
Request: '/dist/walmart-greeter.bundle.js'  
[17:57:43] Server stopped  
Terminate batch job (Y/N)?
```

- d) When prompted to **Terminate the batch job (Y/N)**, type **Y** and press **Enter**.

```
Request: '/lib/webparts/walmartGreeter/loc/en-us.js'  
Request: '/dist/walmart-greeter.bundle.js'  
[17:57:43] Server stopped  
Terminate batch job (Y/N)? Y  
PS C:\Student\Modules\10_SharePointFramework\Lab\spfx-lab>
```

- e) Type **cls** and then press **Enter** to clear to console window.

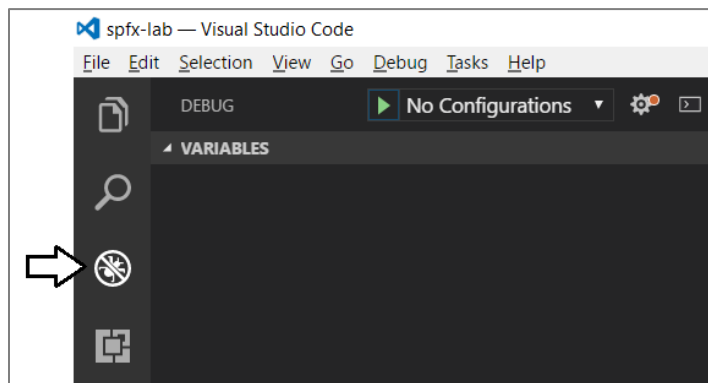
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  
  
PS C:\Student\Modules\10_SharePointFramework\Lab\spfx-lab>
```

This lab assumes you have completed the previous lab and you have already installed the Chrome Debugger extension for Visual Studio Code. If you did not complete the previous lab and have not installed this extension, follow the steps at the following URL:

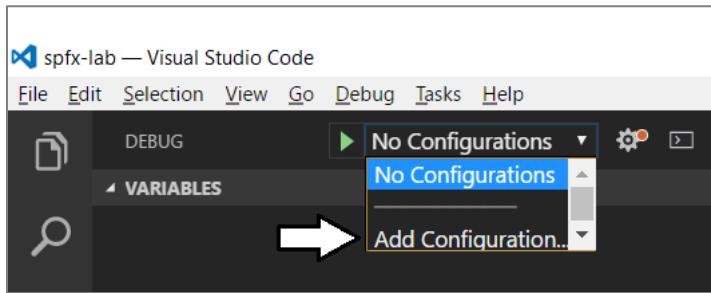
<https://github.com/SharePoint/sp-dev-docs/blob/master/docs/spfx/debug-in-vscode.md>

4. Configure support for client-side debugging of TypeScript code using Visual Studio Code and the Chrome Debugger extension.

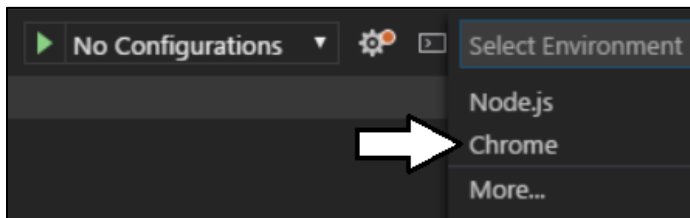
- a) Click on the **Debug** tab in the left navigation.



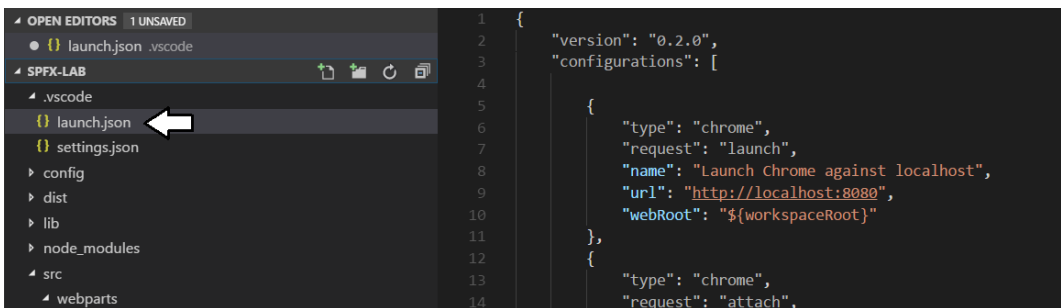
- b) Drop down the **Configuration** menu and select **Add Configuration....**



- c) From the **Select Environment** menu, select **Chrome**.



- d) You will notice that a new file named **launch.json** has been added to the **.vscode** folder in project.



- e) Replace with contents of the **launch.json** file with the following JSON code.

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Local workbench",
      "type": "chrome",
      "request": "launch",
      "url": "https://localhost:4321/temp/workbench.html",
      "webRoot": "${workspaceRoot}",
      "sourceMaps": true,
      "sourceMapPathOverrides": {
        "webpack:///./../../src/*": "${webRoot}/src/*",
        "webpack:///./../../src/*": "${webRoot}/src/*",
        "webpack:///./../../src/*": "${webRoot}/src/*"
      },
      "runtimeArgs": [
        "--remote-debugging-port=9222"
      ]
    }
  ]
}
```

This JSON code is also available a file named **launch.json.starter.txt** in the **StarterFiles** folder.

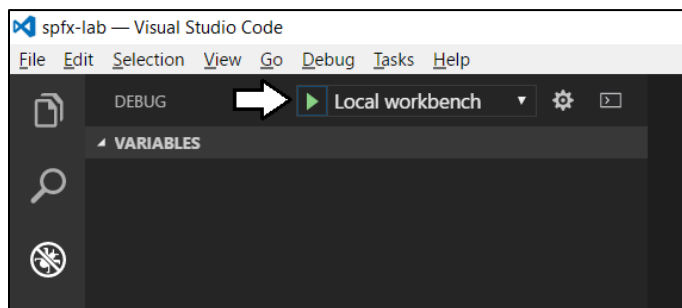
- f) Once you have added the JSON code to match the following screenshot, save and close **launch.json**.

```
launch.json
1 {
2   "version": "0.2.0",
3   "configurations": [
4     {
5       "name": "Local workbench",
6       "type": "chrome",
7       "request": "launch",
8       "url": "https://localhost:4321/temp/workbench.html",
9       "webRoot": "${workspaceRoot}",
10      "sourceMaps": true,
11      "sourceMapPathOverrides": {
12        "webpack:///.../src/*": "${webRoot}/src/*",
13        "webpack:///.../src/*": "${webRoot}/src/*",
14        "webpack:///.../src/*": "${webRoot}/src/*"
15      },
16      "runtimeArgs": [
17        "--remote-debugging-port=9222"
18      ]
19    }
20  ]
21 }
```

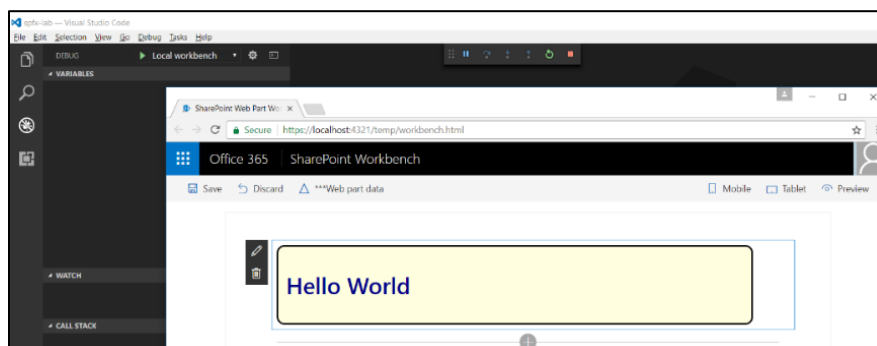
5. Start the web server process and launch a debugging session using the Chrome Debugger extension in Visual Studio Code.
- Return to the console in the Integrated Terminal.
  - Execute the **gulp serve** task using the **--nobrowser** argument to start the web server without launching the browser.

```
gulp serve --nobrowser
```

- Click the **Debug** tab in the left navigation.
- Click the button with the green arrow to begin a debugging session in Visual Studio Code.

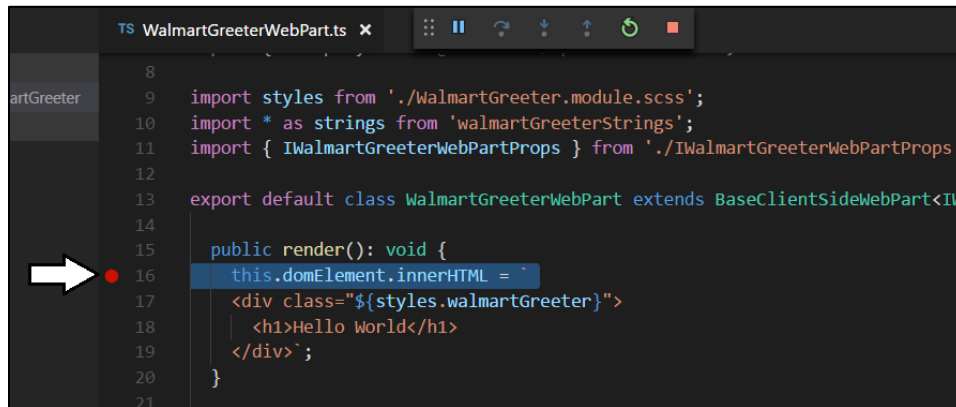


- The local SharePoint Workbench should launch in the browser.
- Add the Walmart Greeter WebPart to the page as you did in previous steps of this exercise.
- Once you see your WebPart, you should also be able to see the debugging toolbar appear in Visual Studio Code.

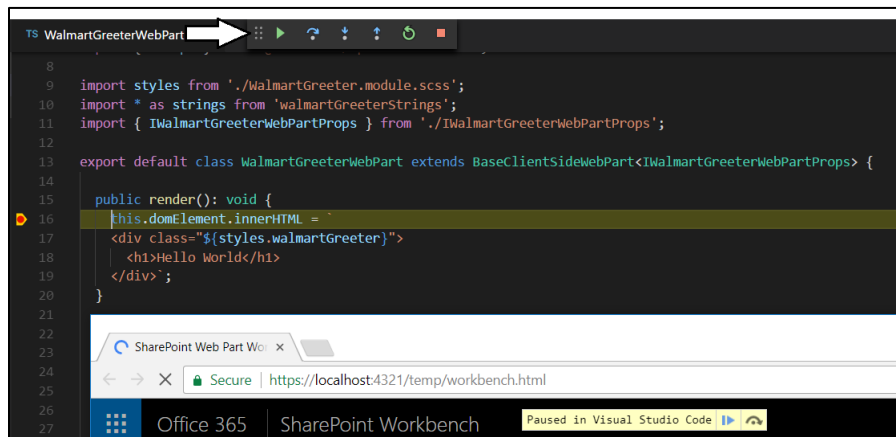


Now you are going to set a breakpoint to test see if you can single step through your code using the Visual Studio Code debugger.

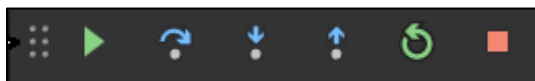
- h) Return to Visual Studio Code.
- i) Navigate to the code editor window for the TypeScript file named **WalmartGreeterWebPart.ts**.
- j) Select the first line of code in the **displayNewQuote** method and set a breakpoint by pressing the **{F9}** key.



- k) Return to the browser window with the local SharePoint Workbench that is displaying the browser.
- l) Refresh the page displaying the WebPart.
- m) Return to Visual Studio Code and you should see that code execution has suspended at the breakpoint you set inside **render**.



- n) Experiment with the button on the debugging toolbar which let you step into and step over code while debugging



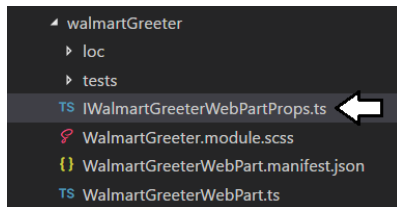
As you can see, it's not overly difficult to set up debugging so you can single step through the TypeScript code you are writing.

- 6. Close down the current debugging session.
  - a) Close the browser which is displaying the WebPart and return to Visual Studio Code.
  - b) Navigate to the console of the Integrated Terminal.
  - c) Make sure the console is the active window
  - d) Press the **Ctrl + C** keyboard combination to stop the web server from running.
  - e) When prompted to **Terminate the batch job (Y/N)**, type **Y** and press **Enter**.
  - f) Type **cls** and then press **Enter** to clear to console window.

## Exercise 3: Extend The WebPart with Custom Properties

In this exercise, you will extend the **WalmartGreeterWebPart** with a set of custom properties. You will accomplish this by designing an interface that defines a set of custom properties and then you will work through the steps to integrate the interface with your WebPart class. You will also create a customized user experience for editing these WebPart properties specific types of user input elements to the property pane that can be seen in WebPart edit mode.

1. Redesign the interface definition inside the WebPart source files named **IWalmartGreeterWebPartProps.ts**.
  - a) Open the TypeScript source file named **IWalmartGreeterWebPartProps.ts**.



- b) The source file contains an interface named **IWalmartGreeterWebPartProps** with a single property named **description**.

```
TS IWalmartGreeterWebPartProps.ts
1  export interface IWalmartGreeterWebPartProps {
2    |  description: string;
3  }
4
```

- c) Modify the **IWalmartGreeterWebPartProps** interface by removing the **description** property and adding four new properties named **greeting**, **fontBold**, **fontSize** and **fontType** as shown in the following code listing.

```
export interface IWalmartGreeterWebPartProps {
  greeting: string;
  fontBold: boolean;
  fontSize: number;
  fontType: string;
}
```

- d) Your code inside **IWalmartGreeterWebPartProps.ts** should now match the following interface definition.

```
TS IWalmartGreeterWebPartProps.ts
1  export interface IWalmartGreeterWebPartProps {
2    |  greeting: string;
3    |  fontBold: boolean;
4    |  fontSize: number;
5    |  fontType: string;
6  }
```

- e) Save your changes and close **IWalmartGreeterWebPartProps.ts**.
2. Set the default values for the four WebPart properties in the WebPart manifest.
    - a) Open the WebPart manifest file named **WalmartGreeterWebPart.manifest.json** in a code editor window.
    - b) Locate the **properties** section inside the **preconfiguredEntries** section.
    - c) Update the **properties** section to match the following code listing.

```
"properties": {
  "greeting": "Welcome to Walmart" ,
  "fontBold": false,
  "fontType": "Arial",
  "fontSize": 36
}
```

- d) Save and close **WalmartGreeterWebPart.manifest.json**.

3. Modify the **render** method of the **WalmartGreeterWebPart** class
  - a) Return to the code editor window for the TypeScript file named **WalmartGreeterWebPart.ts**.
  - b) Replace the current implementation of the **render** method with the following implementation.

```
public render(): void {  
  
    var fontStyle = `font-weight:${this.properties.fontBold ? "bold" : "normal"};  
                    font-family:${this.properties.fontType};  
                    font-size:${this.properties.fontSize}px`;  
  
    this.domElement.innerHTML = `  
    <div class="${styles.walmartGreeter}" >  
      <h1 style="${fontStyle}" >${this.properties.greeting}</h1>  
    </div>`;  
  
}in WalmartGreeterWebPart.ts and
```

This new implementation of **render** **reads** the current value of all 4 custom WebPart properties and uses them to control how its output to the page is displayed. Remember that each time one of these properties is updated, the WebPart will automatically execute the **render** method to keep its view in sync with its underlying property values.

4. Customize the property pane editing experience for each of the four custom properties.
  - a) Move up in **WalmartGreeterWebPart.ts** and locate the **import** statement for **@microsoft/sp-webpart-base**.

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

- b) Extend this import statement with the types **PropertyPaneToggle**, **PropertyPaneDropdown** and **PropertyPaneSlider**.

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

- c) Move down in **WalmartGreeterWebPart.ts** and locate the implement of **getPropertyPaneConfiguration**.
  - d) Replace the current implementation of **getPropertyPaneConfiguration** with the following starter implementation.

```
protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {  
    return {  
        pages: [ {  
            header: { description: "Greeter Web Part" },  
            groups: []  
        }  
    ]  
};  
}
```

- e) Inside the **groups** section, add two groups named **General Properties** and **Cosmetic Properties** using the following code.

```
groups: [  
    {  
        groupName: "General Properties",  
        groupFields: []  
    },  
    {  
        groupName: "Cosmetic Properties",  
        groupFields: []  
    }  
]
```

- f) In the **groupFields** section of the **General Properties** group, add a single text field for **greeting** using the following code.

```
{
  groupName: "General Properties",
  groupFields: [
    PropertyPaneTextField('greeting', { label: 'Greeting' }),
  ]
}
```

- g) In the **groupFields** section of the **Cosmetic Properties** group, add the user interface elements for a toggle, a dropdown menu and a slider as shown in the following code listing.

```
{
  groupName: "Cosmetic Properties",
  groupFields: [
    PropertyPaneToggle('fontBold', {
      label: 'Font Bold',
      onText: 'On',
      offText: 'Off'
    }),
    PropertyPaneDropdown('fontType', {
      label: 'Font Type',
      options: [
        { key: 'Arial', text: 'Arial' },
        { key: 'Times New Roman', text: 'Times New Roman' },
        { key: 'Courier', text: 'Courier' },
        { key: 'Verdana', text: 'Verdana' }
      ]
    }),
    PropertyPaneSlider("fontSize", {
      label: "Font Size",
      min: 24,
      max: 64
    })
  ]
}
```

If you would rather just copy-and-paste the completed implementation of the **getPropertyPaneConfiguration** method, you can find it inside the **StarterFiles** folder in a file named **getPropertyPaneConfiguration.ts.txt**.

- h) Save your changes to **WalmartGreeterWebPart.ts**.
5. Test the WebPart and work with the custom properties.
- a) Return to the console in the Integrated Terminal.
- b) Execute the **gulp serve** task using the **--nobrowser** argument to start the web server without launching the browser.

```
gulp serve --nobrowser
```

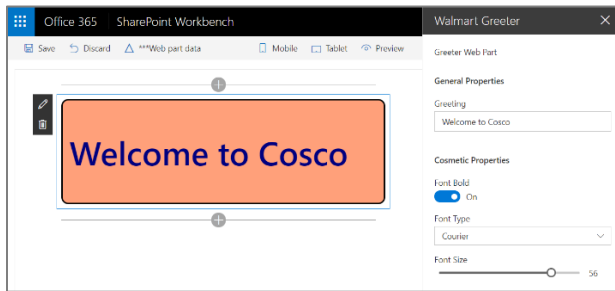
- c) Click the **Debug** tab in the left navigation.
- d) Click the button with the green arrow to begin a debugging session in Visual Studio Code.
- e) When the SharePoint Workbench launches, add the Walmart Greeter WebPart as you have done in previous steps.
- f) When the WebPart displays, click the Edit button to display the properties pane.



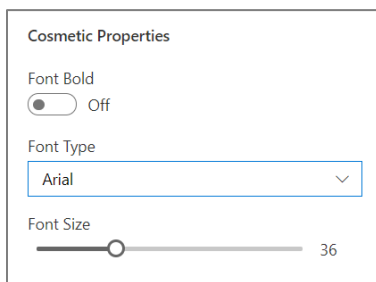
Remember that you are still in the local SharePoint workbench and there is not yet any connection to a SharePoint Online environment. Yet it is still possible for you to develop, test and debug WebParts with custom properties.x



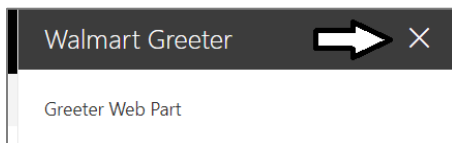
- g) At this point, you should see the WebPart property pane on the right-hand side of the page.
- h) Modify the text for the **Greeting** property and see how your changes are instantly reflected in the WebPart.



- i) Experiment by updating cosmetic properties and seeing how it affects the WebPart's display.

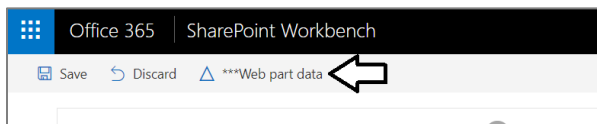


- j) When you are done, close the property pane for the **WalMart Greeter** WebPart by clicking the X in the upper right corner.



The SharePoint Workbench provides you with a viewer which makes it possible to see how WebPart property values are serialized for storage in SharePoint Online. Keep in mind that the SharePoint Framework defines its own new serialization format which is used for WebPart instances on modern pages. For backwards compatibility with classic pages, the SharePoint Framework also supports serializing WebPart instances using the classic WebPart format.

- k) On the toolbar of the SharePoint Workbench, click on the **Web part data button** to view the WebPart in a serialized format.



- l) Note that the Web Part Data viewer has one view for modern pages and a second for classic pages.



## Exercise 4: Run the WebPart in the SharePoint Online Environment

In this exercise, you will extend the WebPart with code that cannot be fully tested in the local SharePoint Workbench. Therefore, you will now learn the steps required to run and test the WebPart in a hosted version of the SharePoint Workbench running inside the SharePoint Online environment.

1. Modify the **render** method of the **WalmartGreeterWebPart** class
  - a) Return to the code editor window for the TypeScript file named **WalmartGreeterWebPart.ts**.
  - b) Locate the **render** method.
  - c) Add the following line of code to the top of the render method before any other code.

```
var userName: string = this.context.pageContext.user.displayName;
```

- d) Update the code that generates the HTML to add **Hello \${userName}** as shown in the following code.

```
this.domElement.innerHTML = `  
<div class="${styles.walmartGreeter}">  
  <h1 style="${fontStyle}">Hello ${userName}, ${this.properties.greeting}</h1>  
</div>`;
```

- e) At this point, the completed implementation of render should match the following code listing.

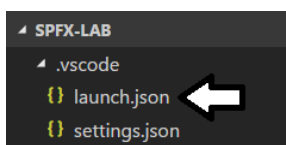
```
public render(): void {  
  var userName: string = this.context.pageContext.user.displayName;  
  
  var fontStyle = `font-weight:${this.properties.fontBold ? "bold" : "normal"};  
                  font-family:${this.properties.fontType};  
                  font-size:${this.properties.fontSize}px;`;  
  
  this.domElement.innerHTML = `  
    <div class="${styles.walmartGreeter}">  
      <h1 style="${fontStyle}">Hello ${userName}, ${this.properties.greeting}</h1>  
    </div>`;  
}
```

- f) Save your changes to **WalmartGreeterWebPart.ts**.
2. Start a debugging session with the SharePoint Workbench.
  - a) Press **{F5}** to launch the SharePoint Workbench.
  - b) Add the Walmart Greeter as you have done in previous steps.
  - c) When the WebPart displays, you should see the WebPart displays **User 1** for the user display name.



The local SharePoint Workbench is not connected to any real SharePoint environment. Therefore, it cannot provide any information about users who have authenticated with Office 365 and SharePoint Online.

3. Add support for testing and debugging in the SharePoint Online environment.
  - a) Open the **launch.json** file in the **.vscode** folder.



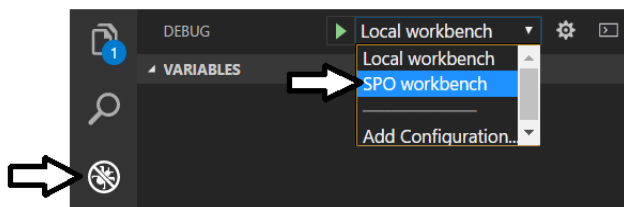
- b) Inside **launch.json**, you should see that there is currently a single configuration named **Local workbench**.
- c) Add a second configuration named **SPO workbench** as shown in the following code listing.

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Local workbench",
      "type": "chrome",
      "request": "launch",
      "url": "https://localhost:4321/temp/workbench.html",
      "webRoot": "${workspaceRoot}",
      "sourceMaps": true,
      "sourceMapPathOverrides": {
        "webpack:///./../../src/*": "${webRoot}/src/*",
        "webpack:///./../../src/*": "${webRoot}/src/*",
        "webpack:///./../../src/*": "${webRoot}/src/*"
      },
      "runtimeArgs": [
        "--remote-debugging-port=9222"
      ]
    },
    {
      "name": "SPO workbench",
      "type": "chrome",
      "request": "launch",
      "url": "https://YOUROFFICE365TENANT.sharepoint.com/_layouts/workbench.aspx",
      "webRoot": "${workspaceRoot}",
      "sourceMaps": true,
      "sourceMapPathOverrides": {
        "webpack:///./../../src/*": "${webRoot}/src/*",
        "webpack:///./../../src/*": "${webRoot}/src/*",
        "webpack:///./../../src/*": "${webRoot}/src/*"
      },
      "runtimeArgs": [
        "--remote-debugging-port=9222"
      ]
    }
  ]
}
```

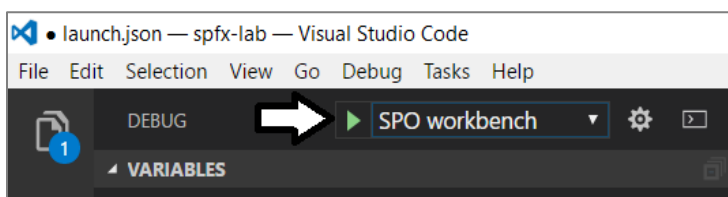
- d) Locate the line with YOUROFFICE365TENANT and replace it with the name of the Office 365 development tenant.

```
"url": "https://YOUROFFICE365TENANT.sharepoint.com/_layouts/workbench.aspx",
```

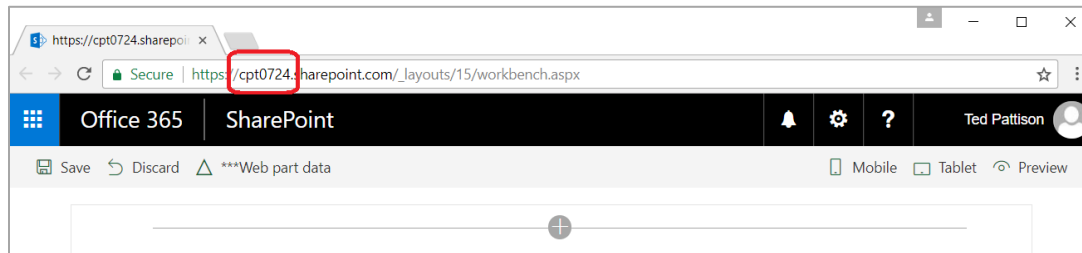
- e) Save your changes and close **launch.json**.
- f) Navigate to the **Debug** tab and then select **SPO workbench** in the dropdown configuration menu



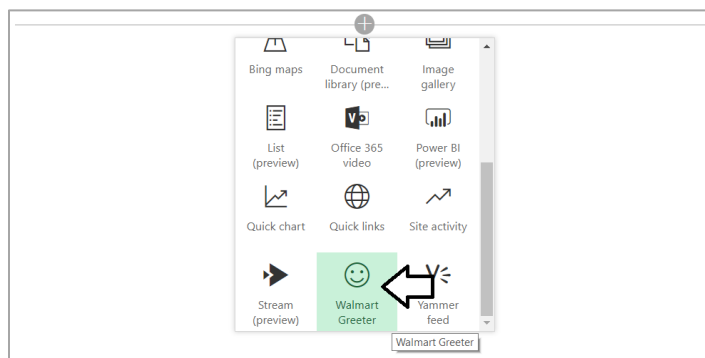
- g) Click on the button with the green arrow or press **{F5}** to start a new debugging session.



- h) The Chrome browser should launch using a URL inside the SharePoint Online environment.
- i) If you are prompted to sign in, enter the credentials of your Office 365 developer account and sign in to continue.
- j) After you are authenticated, you should see hosted page in SharePoint Online running the SharePoint Workbench.



- k) Add the Walmart Greeter WebPart as you have done in previous steps.



Note that quite a few other WebParts are available once you are running inside your own tenancy in SharePoint Online.

- l) The WebPart should now display the actual display name for the user account you have used to sign in.



Congratulations. You have now completed this lab and learned how to get up and running with the SharePoint Framework toolchain.