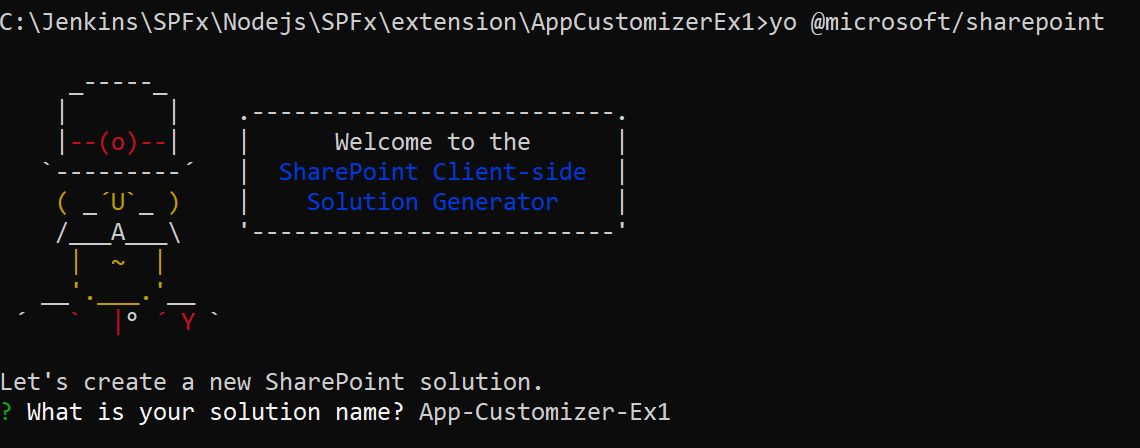
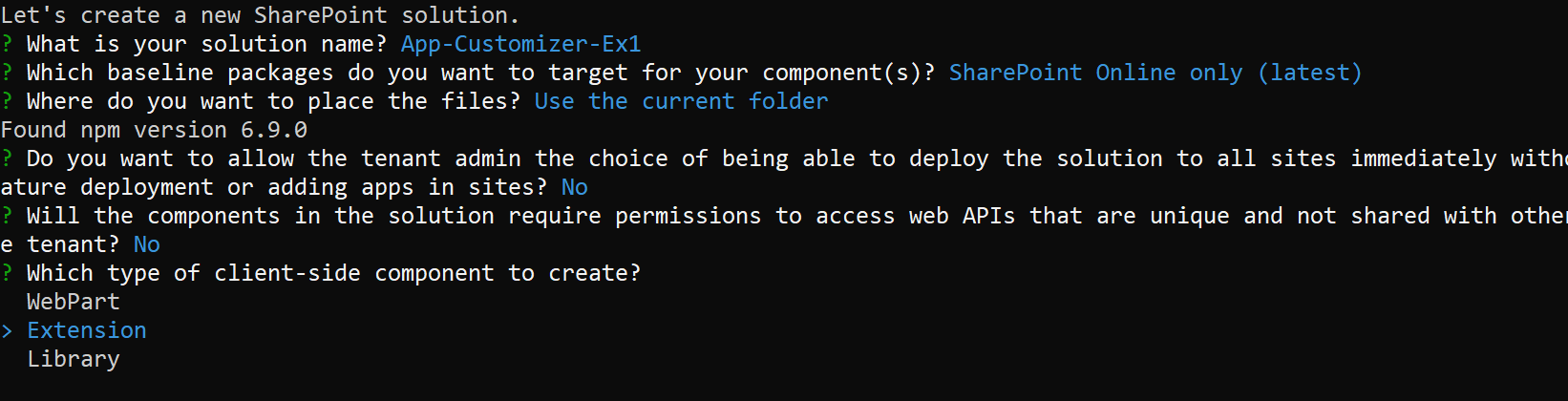
**SPFx Extension Application customizer**

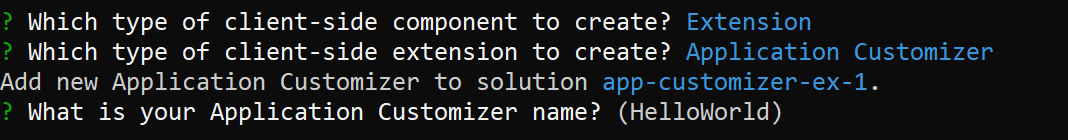
* Open Command prompt
* Navigate to your favourite folder
* Ex: C:\Jenkins\SPFx\Nodejs\SPFx\extension
* Create new folder for app customizer
* Ex: AppCustomizerEx1
* Create spfx solution >yo @microsoft/sharepoint



* While creating SPFx solution, select extension



* Then select Application customizer

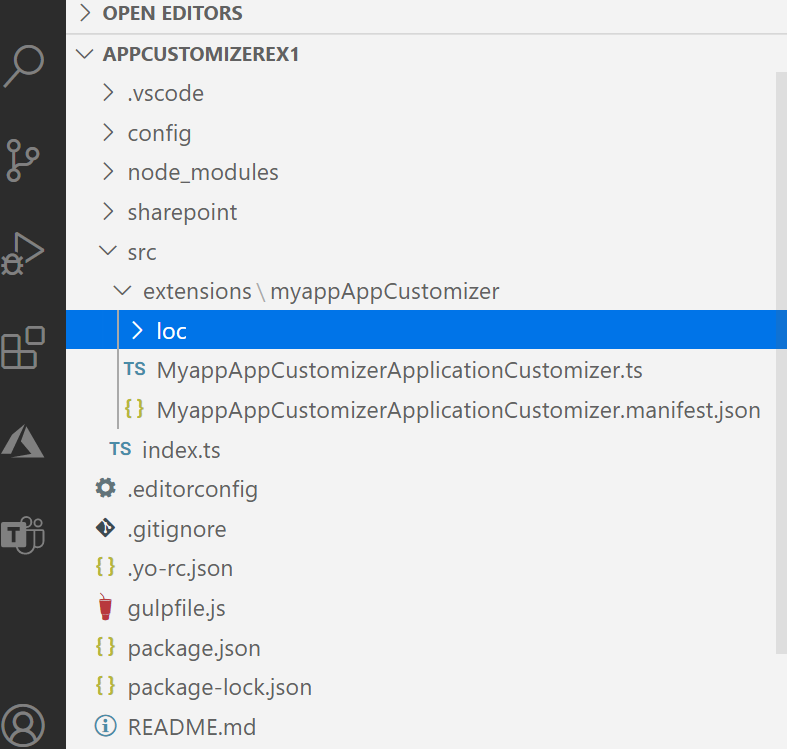


**Note**:

If you use a name for the extension that is too long, you might encounter issues. The entries provided are used to generate an alias entry for the Application Customizer manifest JSON file. If the alias is longer than 40 characters, you get an exception when you try to serve the extension by using **gulp serve --nobrowser**. You can resolve this by updating the alias entry afterward.

At this point, Yeoman installs the required dependencies and scaffolds the solution files along with the **MyappAppCustomizer** extension.

Notice how the default solution structure looks like the solution structure for client-side web parts. This is the basic SharePoint Framework solution structure, with similar configuration options across all solution types.



Code your Application Customizer

Open the **./src/extensions/helloWorld/MyappAppCustomizerApplicationCustomizer.ts** file.

The logic for your Application Customizer is contained in the onInit() method, which is called when the client-side extension is first activated on the page. This event occurs after this.context and this.properties are assigned. As with web parts, onInit() returns a promise that you can use to do asynchronous operations.

Default extension having below alert message below code,

let message: string = this.properties.testMessage;

    if (!message) {

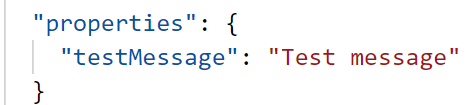
      message = '(No properties were provided.)';

    }

    Dialog.alert(`Hello from ${strings.Title}:\n\n${message}`);

////

this.properties.testMessage 🡪 value from serve.json file



Test the Extension

**Note**: Extension will not support workbench.aspx / workbench.html

Open the **./config/serve.json** file.

Find : https://contoso.sharepoint.com/sites/mySite/SitePages/myPage.aspx

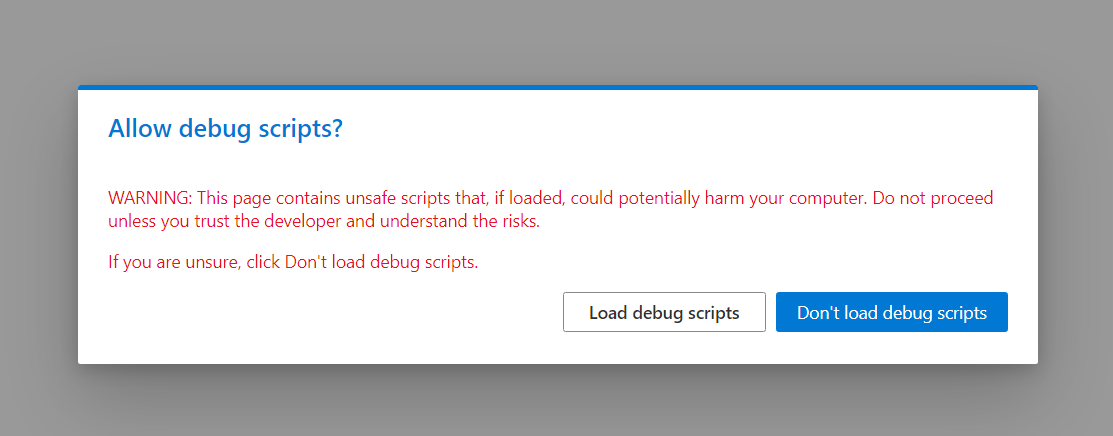
And update **pageURL** to match your own tenant, which you want to use for testing. You can use any URL with modern experience

Ex: <http://jpower4mvp.sharepoint.com/sites/SPFxTraining/SitePages/Home.aspx>

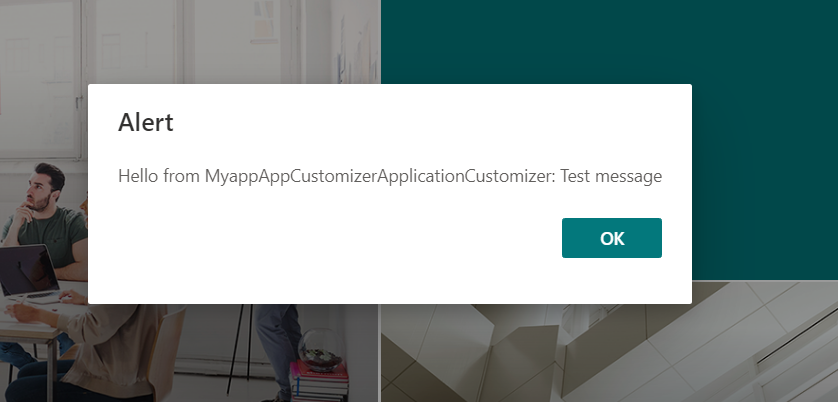
Back to command prompt and execute the code

>gulp serve

Click Load debug scripts and access the application



Output



**Ex2: Use page placeholders from Application Customizer**

Application Customizers provide access to well-known locations on SharePoint pages that you can modify based on your business and functional requirements.

## **Get access to page placeholders**

Application Customizer extensions are supported with **Site**, **Web**, and **List** scopes. You can control the scope by deciding where or how the Application Customizer is registered in your SharePoint tenant.

When the Application Customizer exists in the scope and is being rendered, you can use the following method to get access to the placeholder.

Install the **@microsoft/sp-office-ui-fabric-core** package to enable importing from **SPFabricCore.scss**. We'll use this for defining rendering styles for our place holders.

* Install npm install @microsoft/sp-office-ui-fabric-core
* Create a new file named **./src/extensions/myappAppCustomizer/AppCustomizer.module.scss**.

And add below given code.

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

.app {

.top {

height:60px;

text-align:center;

line-height:2.5;

font-weight:bold;

display: flex;

align-items: center;

justify-content: center;

background-color: $ms-color-themePrimary;

color: $ms-color-white;

}

.bottom {

height:40px;

text-align:center;

line-height:2.5;

font-weight:bold;

display: flex;

align-items: center;

justify-content: center;

background-color: $ms-color-themePrimary;

color: $ms-color-white;

}

}

**Navigate to**

**./src/extensions/myappAppCustomizer/MyappAppCustomizerApplicationCustomizer.ts.**

Add the PlaceholderContent and PlaceholderName to the import statement from **@microsoft/sp-application-base** by updating the import statement as follows:

import {

BaseApplicationCustomizer,

PlaceholderContent,

PlaceholderName

} from '@microsoft/sp-application-base';

Also add the following import statements after the strings import at the top of the file:

import styles from './AppCustomizer.module.scss';

import { escape } from '@microsoft/sp-lodash-subset';

In the **MyappAppCustomizerApplicationCustomizer.ts** file, update the **IMyappAppCustomizerApplicationCustomizerProperties** interface to add specific properties for Header and Footer, as follows:

export interface IHelloWorldApplicationCustomizerProperties {

Top: string;

Bottom: string;

}

Add the following private variables inside the **MyappAppCustomizerApplicationCustomizer** class. In this scenario, these can just be local variables in an onRender() method, but if you want to share them with other objects, define them as private variables.

// These have been added

private \_topPlaceholder: PlaceholderContent | undefined;

private \_bottomPlaceholder: PlaceholderContent | undefined;

Replace the init method with below code

@override

public onInit(): Promise<void> {

Log.info(LOG\_SOURCE, `Initialized ${strings.Title}`);

// Wait for the placeholders to be created (or handle them being changed) and then

// render.

this.context.placeholderProvider.changedEvent.add(this, this.\_renderPlaceHolders);

return Promise.resolve();

}

Create a new \_renderPlaceHolders() private method with the following code:

private \_renderPlaceHolders(): void {

console.log("HelloWorldApplicationCustomizer.\_renderPlaceHolders()");

console.log(

"Available placeholders: ",

this.context.placeholderProvider.placeholderNames

.map(name => PlaceholderName[name])

.join(", ")

);

// Handling the top placeholder

if (!this.\_topPlaceholder) {

this.\_topPlaceholder = this.context.placeholderProvider.tryCreateContent(

PlaceholderName.Top,

{ onDispose: this.\_onDispose }

);

// The extension should not assume that the expected placeholder is available.

if (!this.\_topPlaceholder) {

console.error("The expected placeholder (Top) was not found.");

return;

}

if (this.properties) {

let topString: string = this.properties.Top;

if (!topString) {

topString = "(Top property was not defined.)";

}

if (this.\_topPlaceholder.domElement) {

this.\_topPlaceholder.domElement.innerHTML = `

<div class="${styles.app}">

<div class="${styles.top}">

<i class="ms-Icon ms-Icon--Info" aria-hidden="true"></i> ${escape(

topString

)}

</div>

</div>`;

}

}

}

// Handling the bottom placeholder

if (!this.\_bottomPlaceholder) {

this.\_bottomPlaceholder = this.context.placeholderProvider.tryCreateContent(

PlaceholderName.Bottom,

{ onDispose: this.\_onDispose }

);

// The extension should not assume that the expected placeholder is available.

if (!this.\_bottomPlaceholder) {

console.error("The expected placeholder (Bottom) was not found.");

return;

}

if (this.properties) {

let bottomString: string = this.properties.Bottom;

if (!bottomString) {

bottomString = "(Bottom property was not defined.)";

}

if (this.\_bottomPlaceholder.domElement) {

this.\_bottomPlaceholder.domElement.innerHTML = `

<div class="${styles.app}">

<div class="${styles.bottom}">

<i class="ms-Icon ms-Icon--Info" aria-hidden="true"></i> ${escape(

bottomString

)}

</div>

</div>`;

}

}

}

}

Add below method to handle dispose object

private \_onDispose(): void {

console.log('[HelloWorldApplicationCustomizer.\_onDispose] Disposed custom top and bottom placeholders.');

}

Then go to In the **./config/serve.json** file, update properties section in the file to have Top and Bottom messages.

**From**

"properties": {

                        "testMessage": "Test message"

                    }

**to**

"properties": {

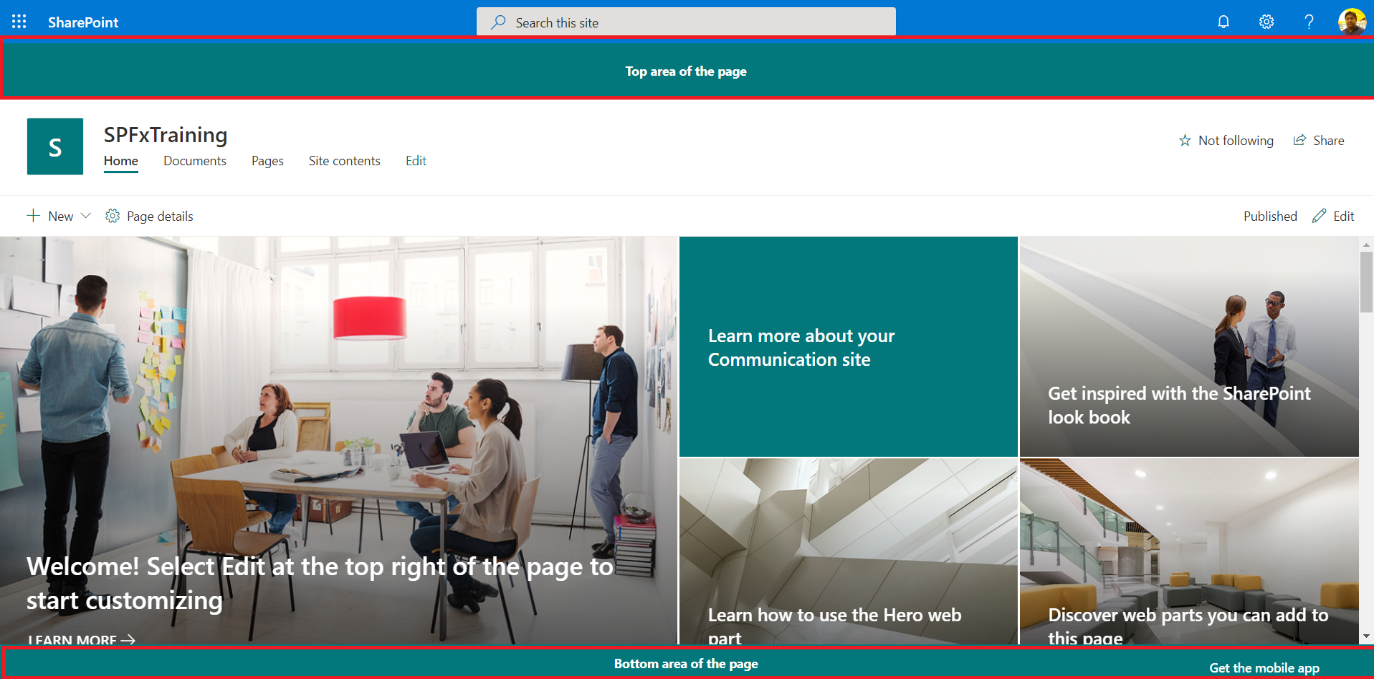
"Top": "Top area of the page",

"Bottom": "Bottom area of the page"

}

Test the app customizer

>gulp serve



**Ex3: Add menus to Top placeholder**

**Navigate to**

**./src/extensions/myappAppCustomizer/MyappAppCustomizerApplicationCustomizer.ts.**

if (this.\_topPlaceholder.domElement) {…}

replace if statement to below given code

if (this.\_topPlaceholder.domElement) {

          this.\_topPlaceholder.domElement.innerHTML = `

          <div class="${styles.app}">

          <div class="${styles.top}">

          <div class="${styles.navbar}">

  <a href="#home">Home</a>

  <a href="#news">News</a>

  <a href="#contact">Contact</a>

</div></div></div>`;

        }

**Navigate to**

**./src/extensions/myappAppCustomizer/AppCustomizer.module.scss**

**Add below css classes, after the existing css classes**

.navbar {

    overflow: hidden;

    background-color: $ms-color-themePrimary;

    top: 0;

    width: 100%;

}

.navbar a {

    float: left;

    display: block;

    color: #f2f2f2;

    text-align: center;

    padding: 14px 16px;

    text-decoration: none;

    font-size: 17px;

}

.navbar a:hover {

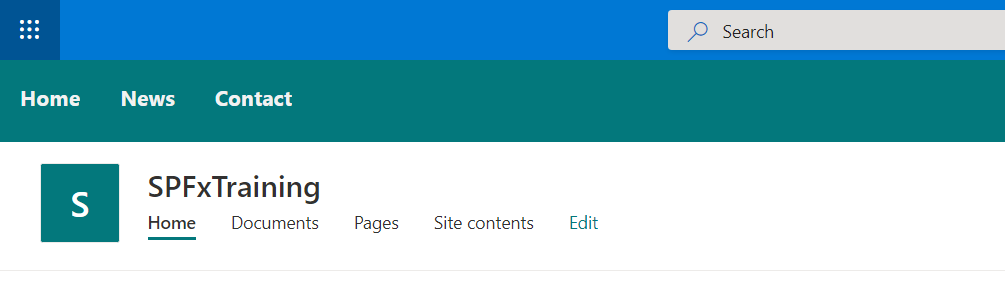
    background: #ddd;

    color: black;

}

Test again

>gulp serve



Ex 3: add google analysis botton

Find below code

if (this.\_bottomPlaceholder.domElement) {

and replace below code

          if (this.\_bottomPlaceholder.domElement) {

            this.\_bottomPlaceholder.domElement.innerHTML = `

          <div id="spfxCookieConsent" class="${styles.app}">

            <div class="${styles.bottom}">

<div>© Copyright 2019 JPOWER4 | All rights reserved.</div>

            </div>

          </div>`;

        }

Test the changes

>gulp serve

