All about the SharePoint Framework (SPFx)

Fabio Franzini

Fabiofranzini.com

#CodeGen

#dotnetconf

@cloudgen_verona

Main sponsor

24/CO

Community sponsors

Local Event

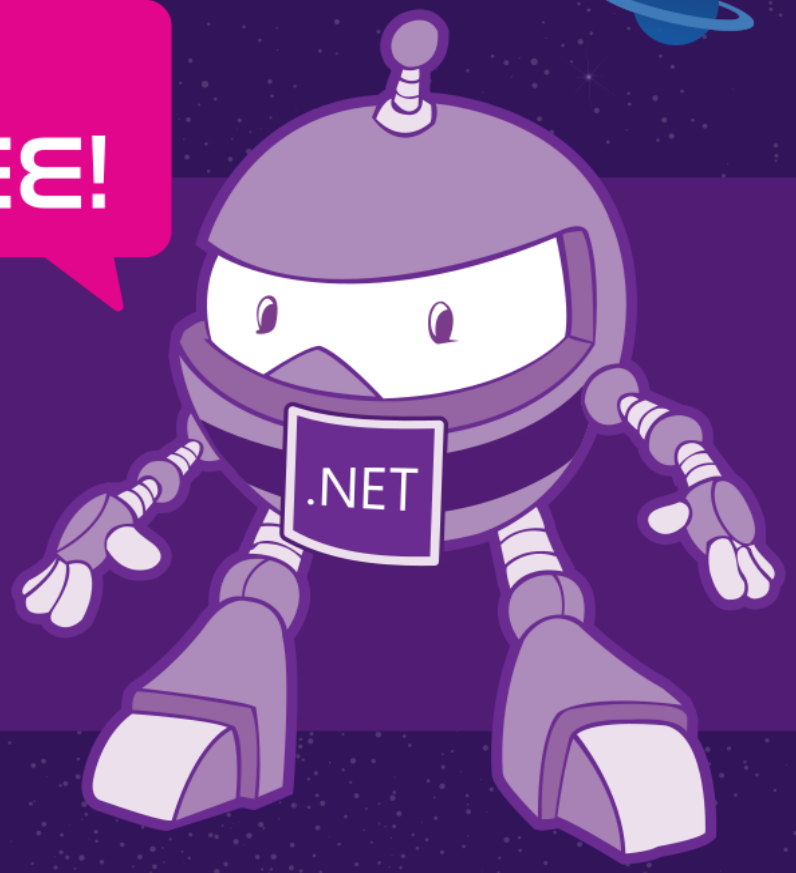
CloudGen

It's FREE!

.NET Conf

Discover the world of .NET

September 12-14, 2018



Tune in: www.dotnetconf.net

Agenda

- Introduction
- Web stack tooling
- Project Structure
- Debugging
- Package & Deploy
- References

Introduction

SharePoint UX – Evolving cross versions

SharePoint
Portal Server 2001

2001

ASP

Early Server
Widget Tech

Single Box

SharePoint
Portal Server 2003

2003

ASP.NET

WebParts

CAML

Office SharePoint
Server 2007

2006

SharePoint
Publishing

SharePoint
Server 2010

2009

FAST acquisition

Shared Services
(e.g. taxonomy)

First Client Side
Rendering Using
XSL

SharePoint
Server 2013

2012

Search driven
publishing sites

Client Side
Rendering
sprinkled in, e.g.
CBS, MDS, etc.

SharePoint
Server 2016, SPO

2016 …

Hybrid

Modern standalone
cloud apps, e.g. O365
Video, Delve, etc.

…

SPFx and Office Fabric

**Modern SharePoint
pages**

What's a modern page?

- Client-side rendered
- Natively responsive
- Client-side web parts
- Client-side and persisted caches
- Mobile optimized

SharePoint

SharePoint Framework

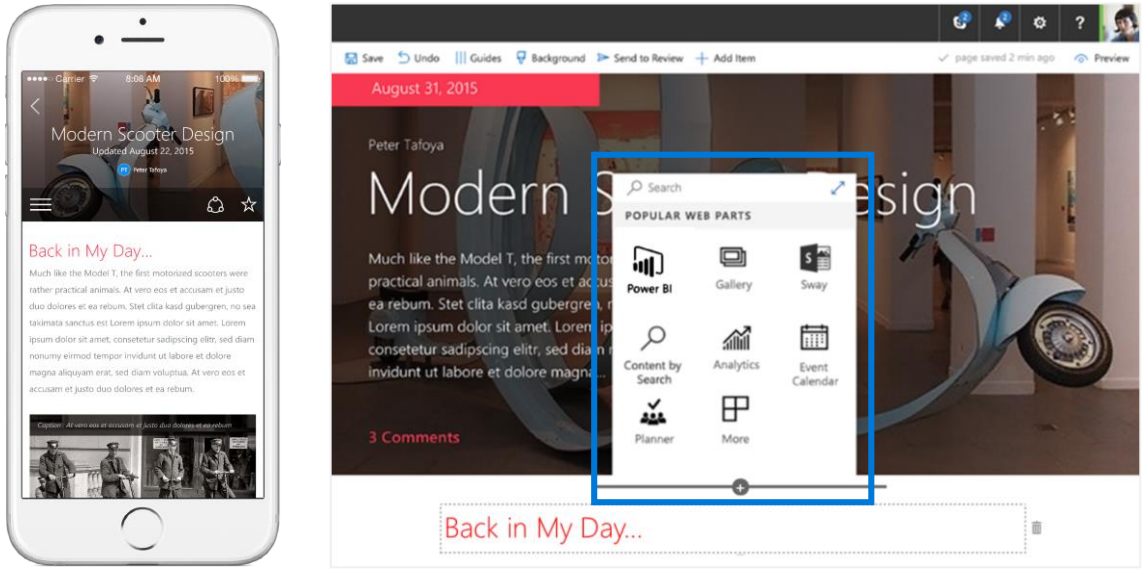
Modern client-side development

Lightweight web and mobile

Powers our own experiences

Backward compatible

Supports open source tools
and JavaScript web frameworks



SharePoint framework

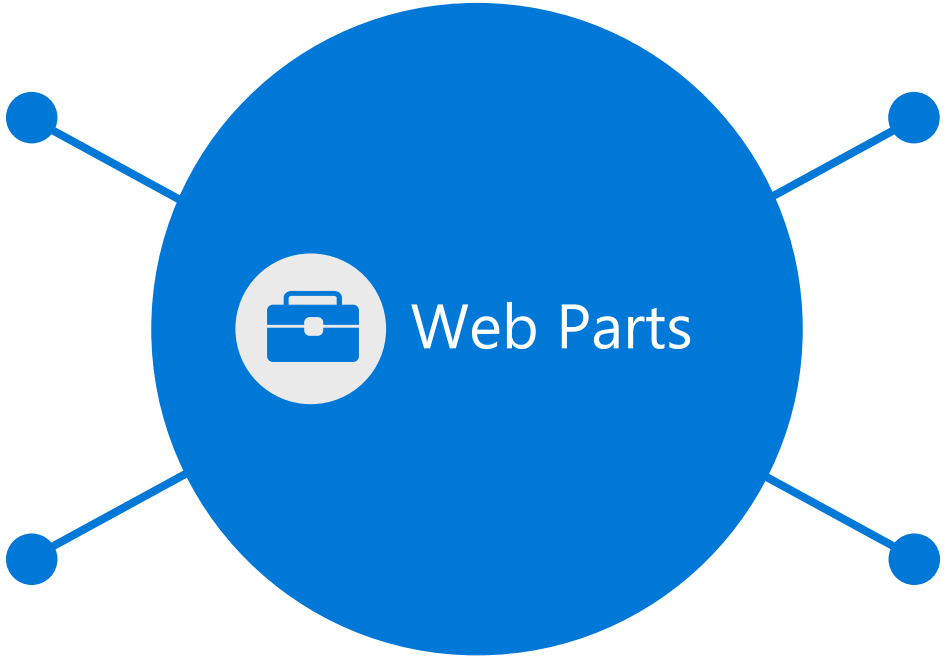
Microsoft Graph

LOB Systems and Cloud Services

Client-Side Web Parts

Configurable, reusable, purpose built components

Framework for connecting related components



Add functionality to SharePoint experiences

Context aware parts

Extensions

SharePoint Framework Extensions enable you to extend the SharePoint user experience within modern pages and document libraries, while using the familiar SharePoint Framework tools and libraries for client-side development.

Application Customizers

Adds scripts to the page, and accesses well-known HTML element **placeholders** and extends them with custom renderings

Field Customizers

Provides **modified views** to data **for fields** within a list

Command Sets

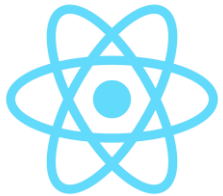
Extends the SharePoint command surfaces to **add new actions**, and provides client-side code that you can use to implement behaviors

Web stack tooling

Typical tooling for SharePoint Framework

- Tooling
  - Node.js
  - Yeoman
  - Gulp
  - TypeScript
  - Visual Studio (Code)
- Frameworks – Choose yours
  - React
  - Angular.js
  - Knockout
  - Etc.

YEOMAN



Web stack tooling compared to classic MS tools

MS Build

Visual Studio

VS Project →
New → <Template>

C#

Node.js – Development time hosting

https://nodejs.org/en/

- Development-time hosting platform
- Local JavaScript runtime environment
  - Can be considered as the IIS Express in typical Microsoft stack
  - Can also work as backend system with server-side code, if needed

Yeoman - Templates

```
> yo @Microsoft/sharepoint
```

YEOMAN

http://yeoman.io

- Scaffolding tool for development projects
- Used to create SharePoint Framework templates on your development machine
- Installed on your machine with npm
- Creates new projects, updates existing projects, adds web parts to existing projects

Client-side Web Part Build Flow

npm install @Microsoft/generator-sharepoint -g

Install SharePoint Generator

yo sharepoint

Scaffold SharePoint Web Part Project

Code .

Build Web Part Code

gulp serve

Test

workbench

Local

UAT / Pre-production

Ship?

gulp package-solution

Package/Deploy

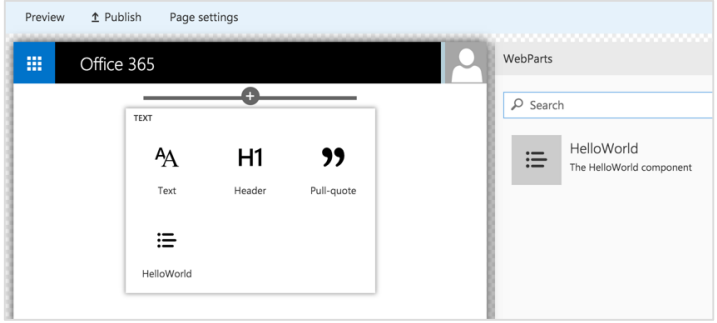
gulp deploy-azure-storage

Package/Deploy

manual upload of the app

Release using app catalog*

Available on Classic and Client-Side Pages



Gulp Tasks

- Defined and implemented in **@microsoft\sp-build-core-tasks** npm package
- List the available Gulp tasks

```
> gulp -T
```

Select Command Prompt

```
C:\SPFx\SPFx\todd>gulp -T
[09:44:14] Using gulpfile C:\SPFx\SPFx\todd\gulpfile.js
(node:15824) DeprecationWarning: os.tmpDir() is deprecated. Use os.tmpdir() instead.
[09:44:14] Tasks for C:\SPFx\SPFx\todd\gulpfile.js
[09:44:14] ├── nuke
[09:44:14] ├── build
[09:44:14] ├── default
[09:44:14] ├── bundle
[09:44:14] ├── dev-deploy
[09:44:14] ├── deploy-azure-storage
[09:44:14] ├── package-solution
[09:44:14] ├── test
[09:44:14] ├── serve
[09:44:14] ├── trust-dev-cert
[09:44:14] └── untrust-dev-cert
```

What the Gulp tasks do

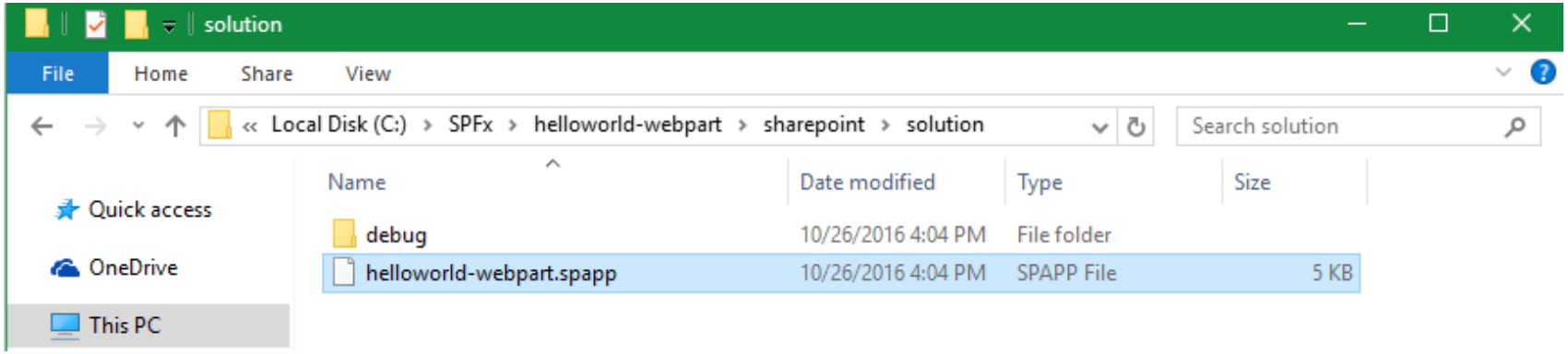
- clean - deletes SharePoint Framework build folders and intermediate Sass files in the src folder
- build - build the project
- default - equivalent to bundle
- bundle - build, localize, and bundle the project
- dev-deploy - deploy the current project to a development Azure CDN for sharing builds with colleagues
- deploy-azure-storage - upload the assets to a Azure storage container
- package-solution - package the project into a SPPKG
- test - build, localize, and bundle the project and run tests, and verify the coverage
- serve - build and bundle the project and run the development server
- trust-dev-cert - generates and trusts a development certificate if one isn't already present
- untrust-dev-cert - untrusts and deletes the development certificate if it exists

Solution Packaging

- Gulp tasks package solutions

```
> gulp package-solution
```

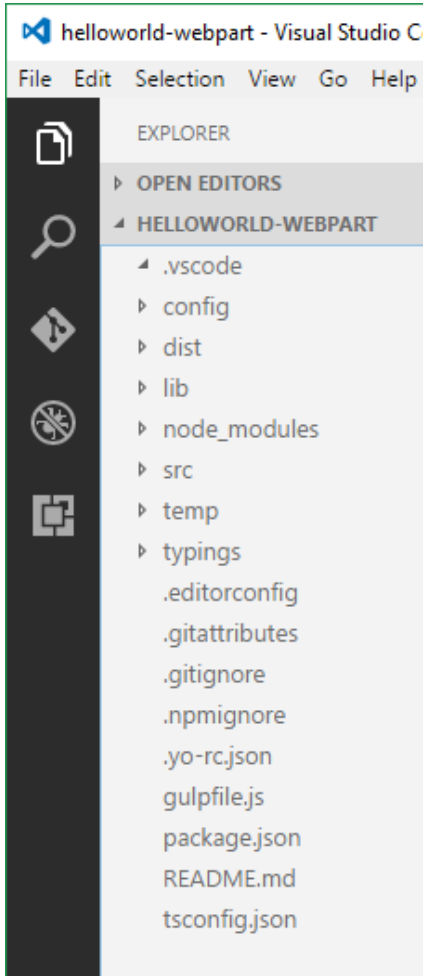
- Package contents are put into an .spapp file



- The JavaScript files, CSS and other assets are not packaged, they must be deployed to an external location such as a CDN.

Project Structure

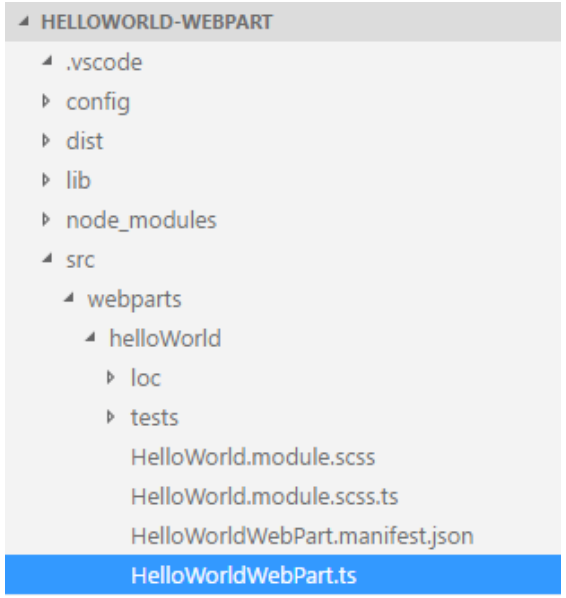
Project Structure



Top level folders

- .vscode: includes Visual Studio Code integration files
- config: includes all config files
- dist: created automatically when you build the project – holds debug builds
- lib: created automatically when you build the project
- node_modules: this is created automatically when you build your project, it includes all the npm packages your solution relies upon and their dependencies
- src: this is the main folder of the project, it includes the web part, styles, and a test file
- temp: created automatically when you build your project – holds production builds
- typings: includes some type definition files. Most type definitions are installed in node_modules\@types

Key Files – web part class

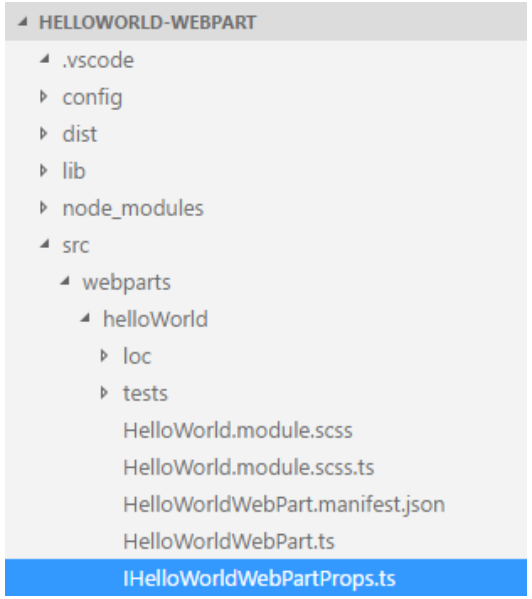


- Defines the main entry point for the web part
- Extends the BaseClientSideWebPart.
- All client-side webs part must extend the BaseClientSideWebPart class in order to be defined as a valid web part

```
import { IHelloWorldWebPartProps } from './IHelloWorldWebPartProps';

export default class HelloWorldWebPart extends
BaseClientSideWebPart<IHelloWorldWebPartProps>
{
        // code omitted
}
```

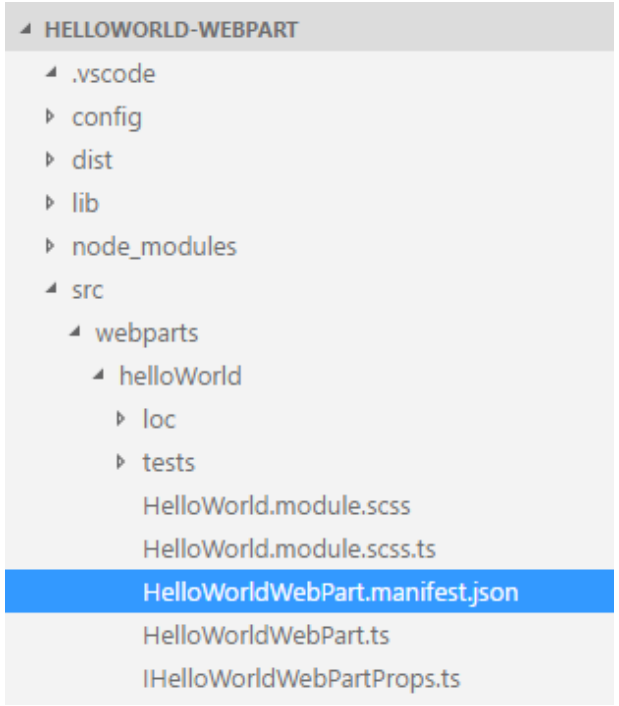
Key Files – web part properties class



- Defines the interface for moving properties between different classes in the web part

```
export interface IHelloWorldWebPartProps {
        description: string;
}
```

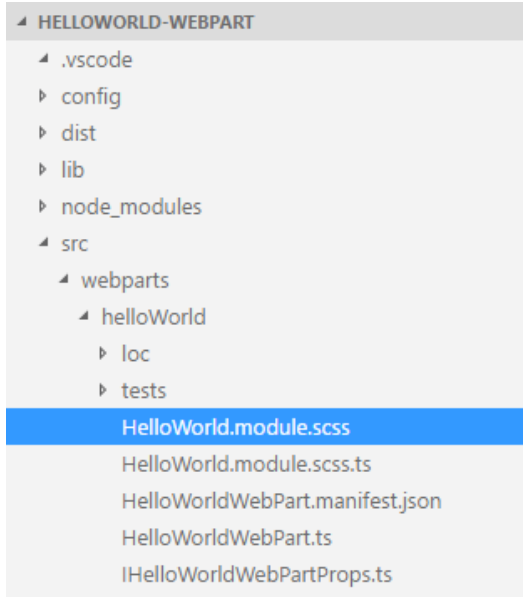
Key Files – web part manifest



- Defines the web part metadata

```
{
  "$schema": "../../../node_modules/@microsoft/sp-module-
interfaces/lib/manifestSchemas/jsonSchemas/clientSideComponentManifestSchema.json",
  "id": "318dd20d-0c02-4c3d-acc5-e2c0fa84cf3f",
  "alias": "HelloWorldWebPart",
  "componentType": "WebPart",
  "version": "0.0.1",
  "manifestVersion": 2,
  "preconfiguredEntries": [{
    "groupId": "318dd20d-0c02-4c3d-acc5-e2c0fa84cf3f",
    "group": { "default": "Under Development" },
    "title": { "default": "HelloWorld" },
    "description": { "default": "HelloWorld description" },
    "officeFabricIconFontName": "Page",
    "properties": {
      "description": "HelloWorld"
    }
  }]
}
```

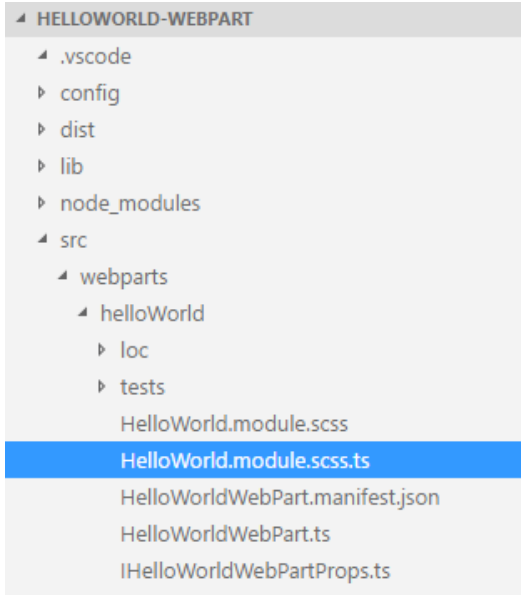
Key Files – SCSS file



- Defines the web part styles

```
.container {
    max-width: 700px;
    margin: 0px auto;
    box-shadow: 0 2px 4px 0 rgba(0, 0, 0, 0.2), 0 25px 50px 0 rgba(0, 0, 0, 0.1);
}
.row {
    padding: 20px;
}
.listItem {
    max-width: 715px;
    margin: 5px auto 5px auto;
    box-shadow: 0 0 4px 0 rgba(0, 0, 0, 0.2), 0 25px 50px 0 rgba(0, 0, 0, 0.1);
}
```

Key Files – SCSS TypeScript file

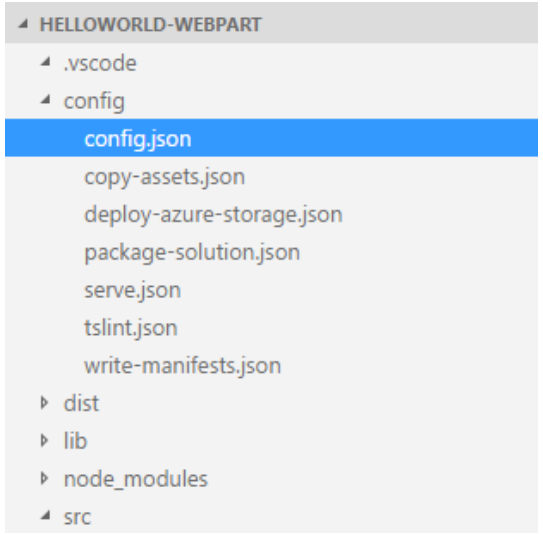


- Defines the TypeScript typings for the web part styles

```
/* tslint:disable */
require('./HelloWorld.module.css');
const styles = {
  helloWorld: 'helloWorld_68b3b0f6',
  container: 'container_68b3b0f6',
  row: 'row_68b3b0f6',
  listItem: 'listItem_68b3b0f6',
  button: 'button_68b3b0f6',
  label: 'label_68b3b0f6',
};

export default styles;
/* tslint:enable */
```

Key Files – config file



- Contains information about your bundle(s), any external dependencies, localized resources
- Specifies the AMD script libraries used in the web part

```
{
  "entries": [
    {
      "entry": "./lib/webparts/helloWorld/HelloWorldWebPart.js",
      "manifest": "./src/webparts/helloWorld/HelloWorldWebPart.manifest.json",
      "outputPath": "./dist/hello-world.bundle.js"
    }
  ],
  "externals": {
    “jquery": "node_modules/jquery/dist/jquery.min.js"
  },
  "localizedResources": {
    "helloWorldStrings": "webparts/helloWorld/loc/{locale}.js"
  }
}
```

Debugging

Debugging

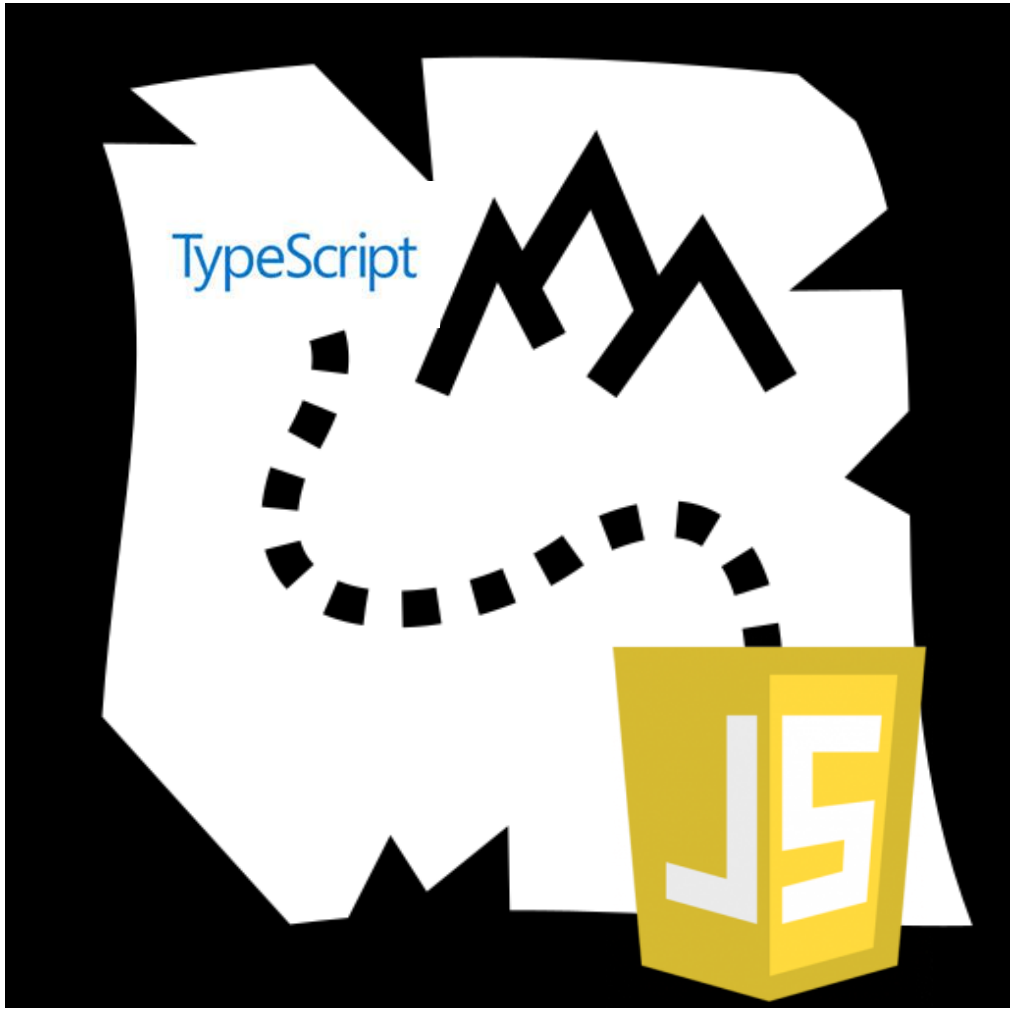
- Build and run on local server **and** automatically launch local SharePoint Workbench

```
> gulp serve
```

- Build and run solution on local server

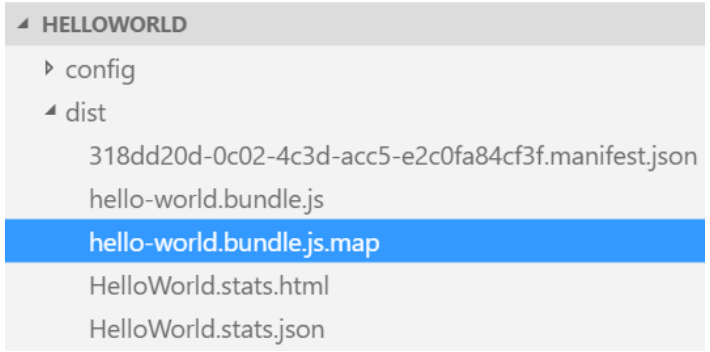
```
> gulp serve --nobrowser
```

Mapping Files Making Debugging Easier



- SPFx web parts are authored in TypeScript
- The build process changes the TypeScript into JavaScript and bundles it all into a single file
- As a result, it can be hard to debug the JavaScript the build generates
- Source code mapping files make it possible to debug the original unbundled TypeScript code

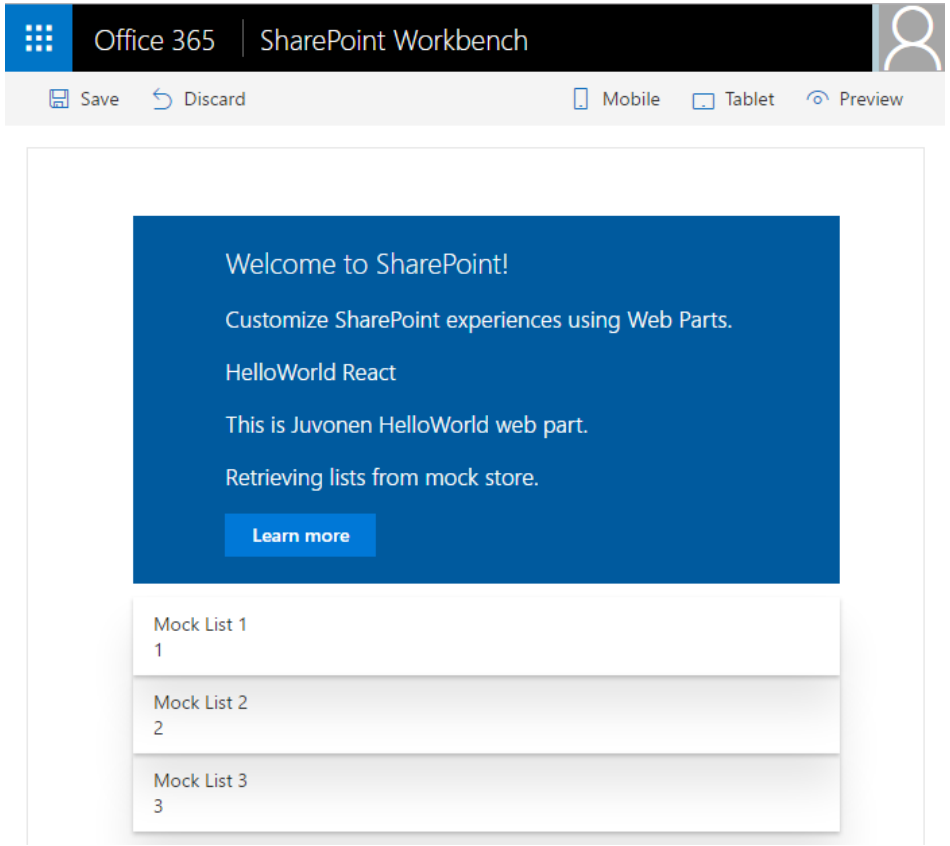
Source Code Mapping Files Details



- SPFx generates source code mapping files in debug builds
- TypeScript files are mapped to the generated JavaScript files
- Source code mapping files are included in the generated JavaScript files

SharePoint Workbench

- Local development time experience
- Test your changes immediately even in offline mode



Local Workbench vs. SharePoint Workbench

- Local
  - Runs on https://localhost
  - Has no SharePoint Context
  - Uses mock data
- SharePoint
  - Runs on SharePoint Site (https://<your-sharepoint-site>/_layouts/workbench.aspx)
  - Has SharePoint Context
  - Uses SharePoint data

Package & Deploy

Package the web part

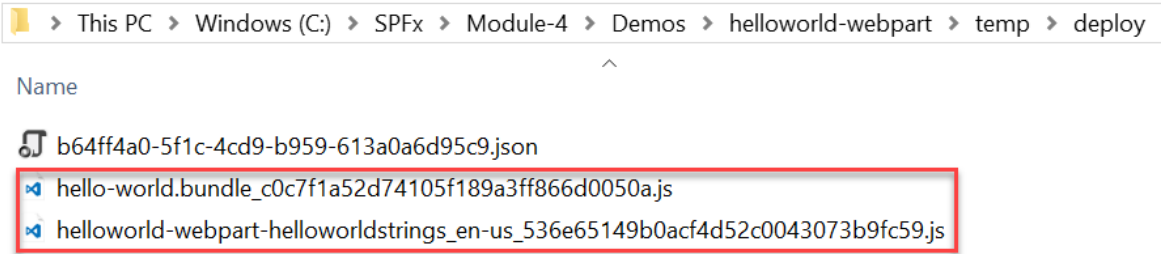
Use the **bundle** gulp task to build, localize, and bundle the project

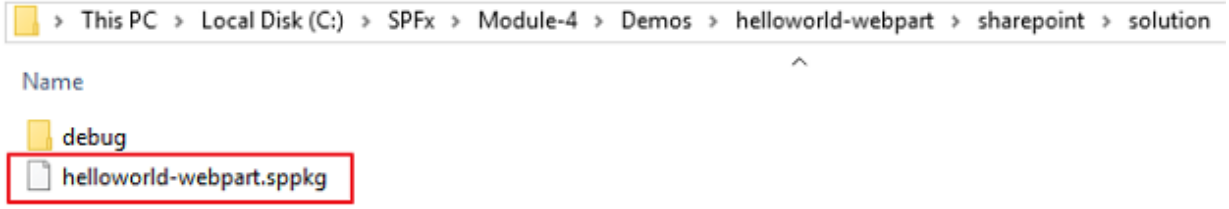
```
> gulp bundle --ship
```

Use the **package-solution** gulp task to package the project into a .sppkg file

```
> gulp package-solution --ship
```

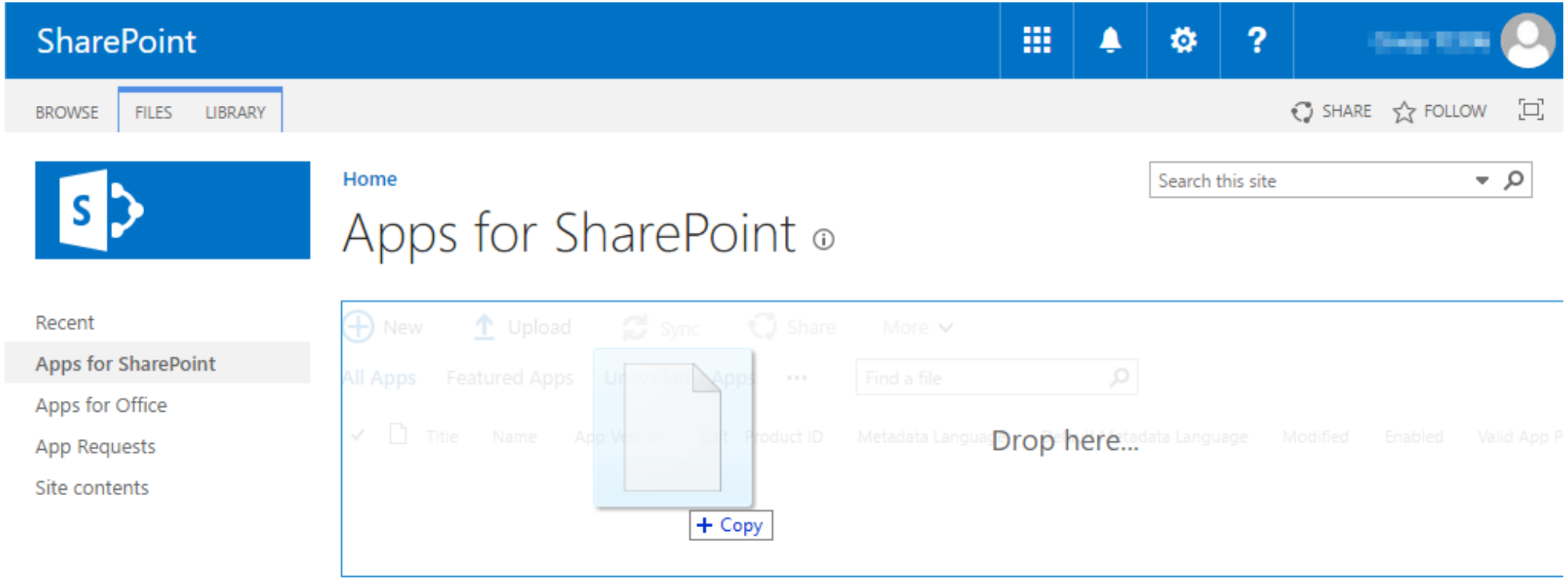
The **ship parameter** build task creates a minified version of the bundle and copies all of the web part assets, including the web part bundle, into the temp\deploy folder. The .sppkg file is generated in the sharepoint\solution folder.





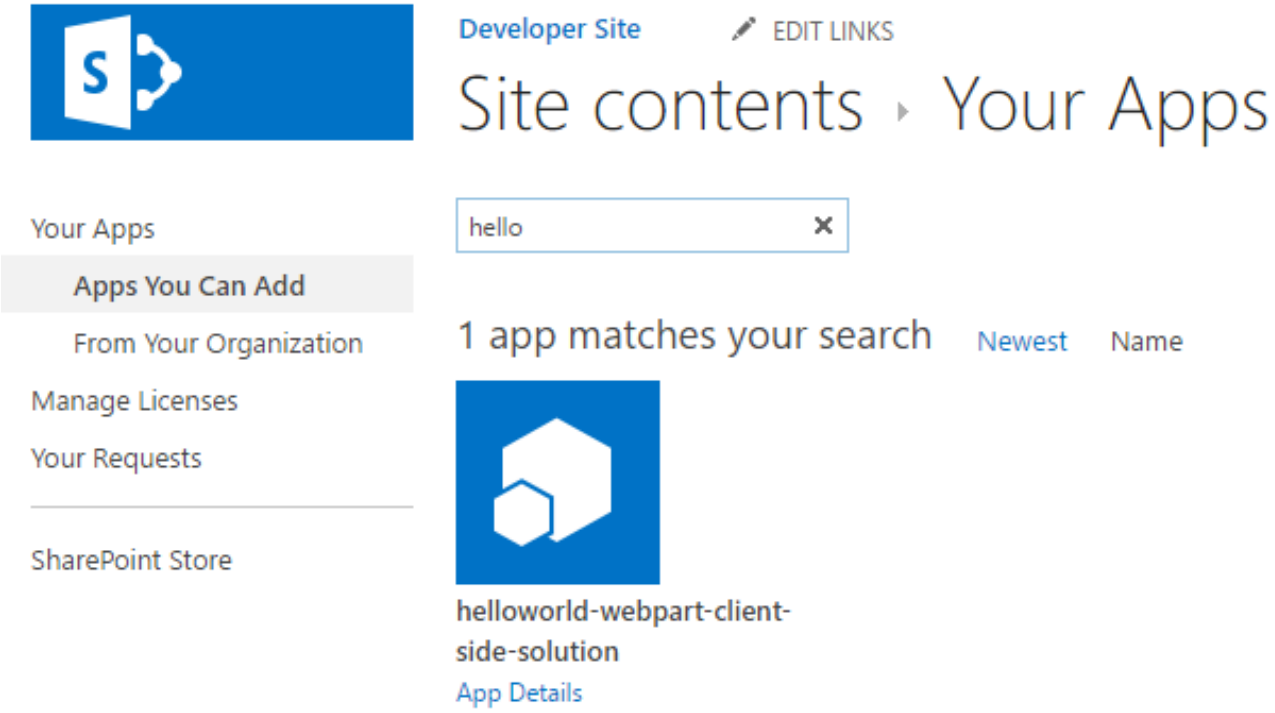
Deploy App to the SharePoint App Catalog

- Go to your Office 365 App Catalog site
- In the left sidebar, choose Apps for SharePoint
- Upload the package (.sppkg file) for the web part



Install the App

- Go to your Office 365 site
- Add the App you just deployed to the SharePoint App Catalog

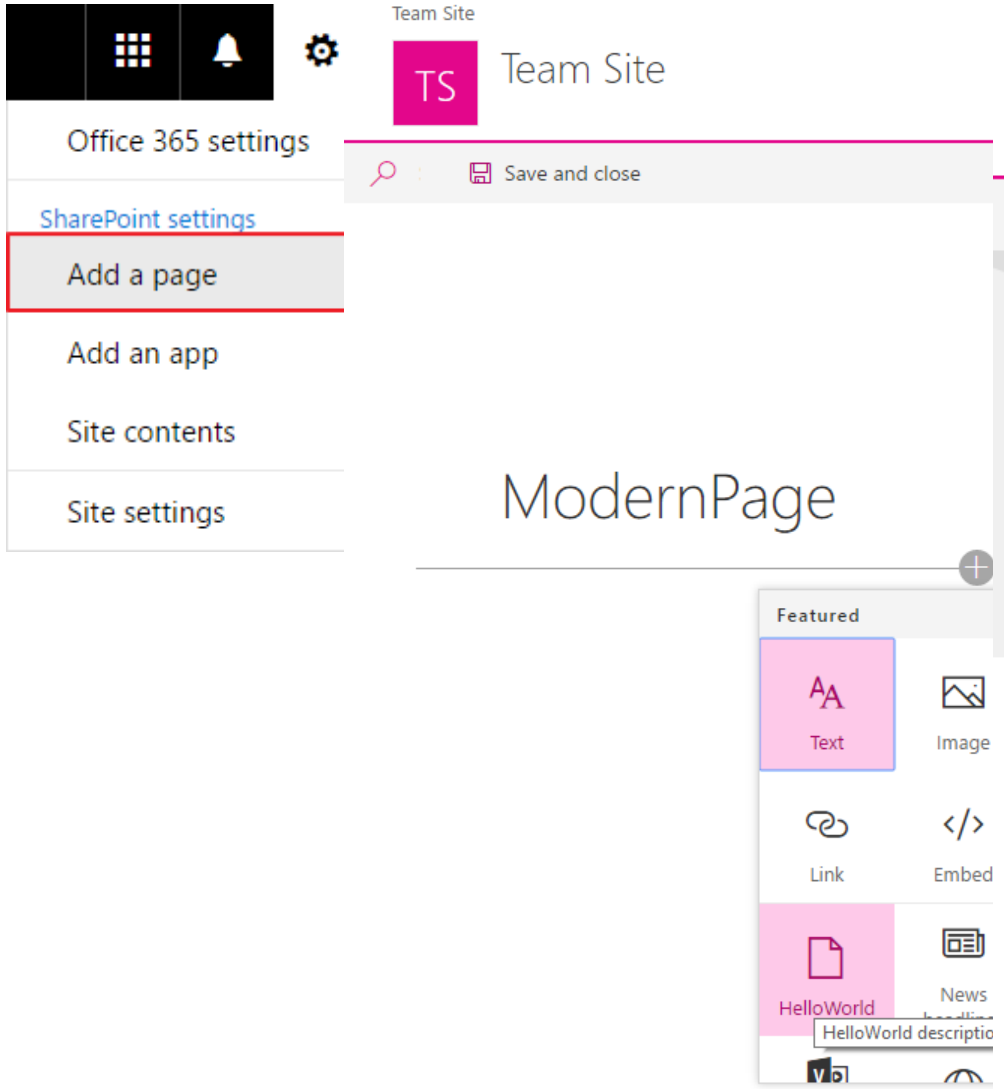


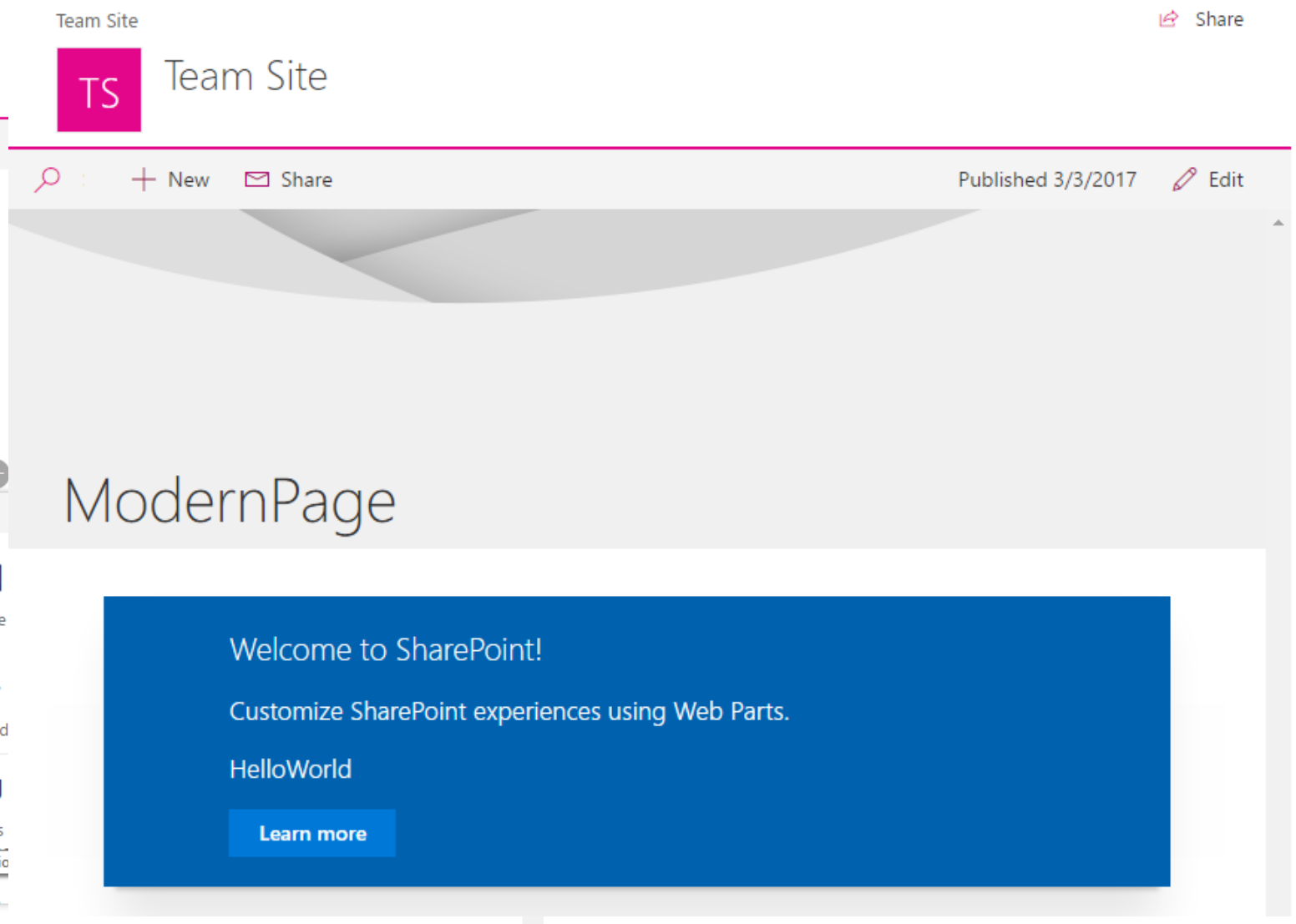
Deploy assets to a CDN and configure web part

- At this point the web part will not run
- You must first
  - deploy the web part assets to a CDN location
    - SharePoint CDN
    - Azure Storage CDN
  - update the cdnBasePath in the write-manifests.json file

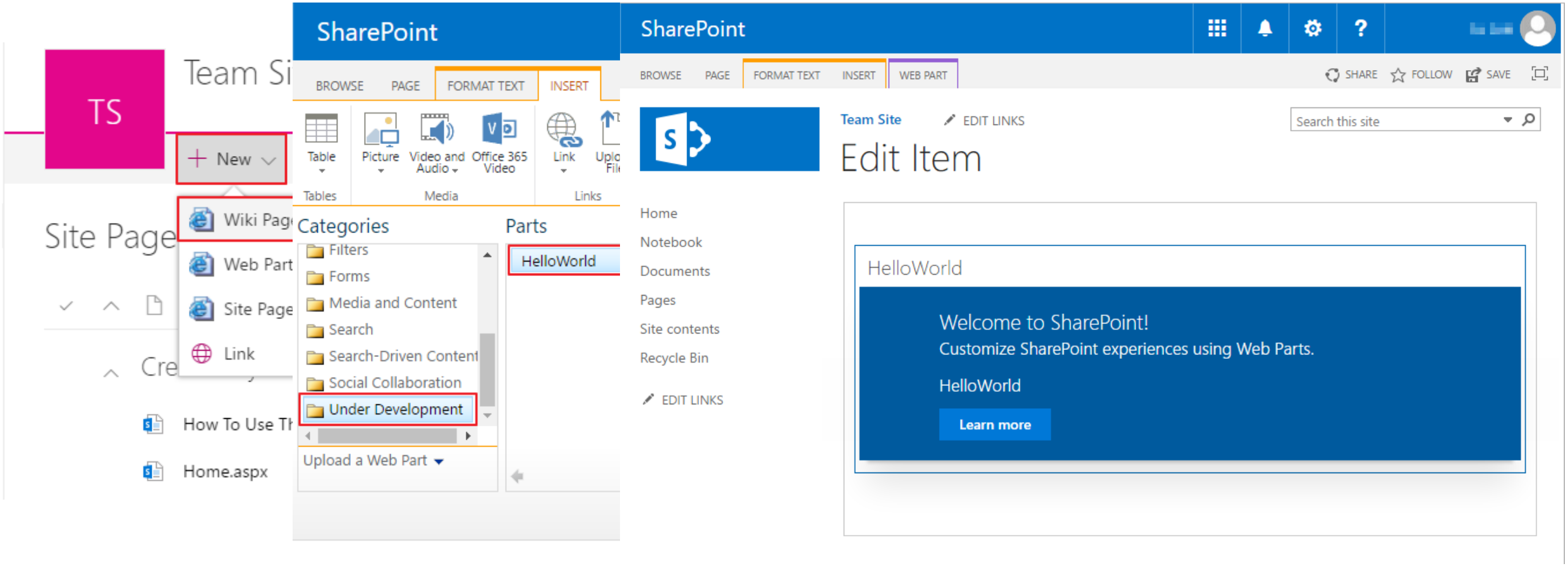
```
{
  "cdnBasePath": "<!-- PATH TO CDN -->"
}
```

Add the web part to a Modern Page





Add the web part to a Classic Page



References

- http://dev.office.com
- https://docs.microsoft.com/en-us/sharepoint/dev/spfx/sharepoint-framework-overview

Fabio Franzini

@franzinifabio

fabiofranzini

fabiofranzini

Grazie

Domande?

fabiofranzini

@franzinifabio

fabiofranzini