Mastering Office 365 development with SharePoint Framework

Elio Struyf, Valo Intranet
Paolo Pialorsi, PiaSys.com
Elio Struyf
Developer & Architect

@eliostruyf
www.eliostruyf.com
info@estruyf.be

Office Servers & Services MVP
Azure / Office 365 / SharePoint
Paolo Pialorsi
Senior Consultant

@paolopia
www.piasys.com
paolo@pialorsi.com

Office Servers & Services MVP
Azure / Office 365 / SharePoint
Schedule

• 9:30 – START
• 10:30 – COFFEE BREAK
• 12:30 – LUNCH
• 13:30 – RESTART
• 15:30 – COFFEE BREAK
• 17:30 – THE END
A Huge “Thank You!” To Our Sponsors ...
INTRODUCTION TO SHAREPOINT FRAMEWORK
A BRIEF HISTORY LESSON
What’s wrong with Sandboxed Solutions?
Add-Ins maybe?

- Still supported in SharePoint Online
- Can work side-by-side with SPFx
- Allow for great flexibility
- Can become complicated very quickly
- Use a different security model (IFrame)
- Are not responsive (IFrame)
- Can be distributed through the Store
Uncontrolled JavaScript code

- Much simpler than SharePoint Hosted Add-Ins
- But doesn’t use any model
- Typically through content/script editor webparts
- Heavy dependency on the DOM (Document Object Model)
- Tendency to break when SharePoint Online changes
- Difficult to control by the administrators:
  - On or Off only
A NEW MODEL
What is SharePoint Framework?

• (not) New client-side development model
• Using modern client-side tools
• Focus on JavaScript and TypeScript
• Web, mobile, responsive

• Dev preview in August 2016
• Released in February 2017 (GA)
Building out the User Experience

- SharePoint Framework
  - Experiences: Web Parts, Extensions
  - Development: Client side Tools, Package Deployment

- Provider Hosted
  - Add-ins
  - MS Graph
  - SP APIs
  - CSOM

SPS EVENTS
Milan
Advantages of SharePoint Framework

• Guidance
• Governance
• Future proof
• Use what you already know
Limitations

- Runs in the context of the user
- Running long operations is not recommended
  - Best to offload these tasks to Azure Functions
- Need for (external) APIs
  - Elevated privileges
- Everyone can read the code, so be careful about tokens / keys / ...
Typical tooling for SharePoint Framework

• Tooling
  • Node.js
  • Yeoman
  • Gulp
  • TypeScript
  • Visual Studio (Code)

• Frameworks – Choose yours
  • React
  • Angular.js
  • Knockout
  • Etc.
Web stack tooling comparison

- Node script
- Gulp
- Office 365 CLI
- PowerShell
- VSCode extension

IIS Express → VS Project
VS Project → New → <Template>

- MS Build
- NuGet
- TypeScript
- C#
Node.js
What is Node.js

- Run server-side JavaScript
- Makes use of Google’s V8 JavaScript engine
- Supports the latest ECMAScript features
- Open source
- Cross-platform
- “Can” act as a web server, but more is possible
- Check your version: `node -v`
SPFx usage:
The main engine for dev builds and creating production packages
Node.js installation:
- Current LTS version of NodeJS is 8.11.4
- https://nodejs.org
What is npm?

• JavaScript package manager
• Similar like NuGet
• Install and manage dependencies / modules
• `npm init`
• `npm install <package_name>`
package.json

Helps locally manage (dev) dependencies and project information:

- Project name
- Version
- Dependencies / devDependencies
- Scripts
- Author
- license
Installing packages

• Locally: npm install <package_name>
• Locally: npm install <package_name> --save
• Locally: npm install <package_name> --save-dev

• Globally: npm install -g <package_name>
Find packages on:
https://www.npmjs.com/
Important: semantic versioning

• Patch releases: 1.0 or 1.0.x or ~1.0.4
  → bug fixes

• Minor releases (default install): 1 or 1.x or ^1.0.4
  → new features, but no breaking changes

• Major releases: * or x
  → Major releases, can have breaking changes
SPFx usage: installing dependencies
Gulp
Gulp: Automate and enhance your workflow
Why using Gulp?

- Transpiling TypeScript > JavaScript / SASS > CSS
- Building debug / production packages
- Uploading files to CDN / FTP / SharePoint
- Execute tasks after file changes
- Warming-up scripts
- ...
gulp --tasks
# SPFx usage

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>gulp serve</em></td>
<td>Serving the local development environment (--nobrowser is optional)</td>
</tr>
<tr>
<td><em>gulp build</em></td>
<td>Builds the project (transpiling) → check “lib” folder</td>
</tr>
<tr>
<td><em>gulp test</em></td>
<td>Runs the unit tests if you have written any</td>
</tr>
<tr>
<td>*gulp bundle</td>
<td>gulp*</td>
</tr>
<tr>
<td>*gulp bundle --ship</td>
<td>gulp --ship*</td>
</tr>
<tr>
<td><em>gulp package-solution</em></td>
<td>Packages the DEBUG solution as an app package → check “sharepoint” folder</td>
</tr>
<tr>
<td><em>gulp package-solution --ship</em></td>
<td>Packages the production ready solution → check “sharepoint” folder</td>
</tr>
<tr>
<td><em>gulp deploy-azure-storage</em></td>
<td>Add the files to your Azure CDN</td>
</tr>
<tr>
<td><em>gulp clean</em></td>
<td>Cleans the project from build artifacts (remove previous builds)</td>
</tr>
<tr>
<td><em>gulp trust-dev-cert</em></td>
<td>Add a developer certificate to your client → necessary for local development</td>
</tr>
<tr>
<td><em>gulp untrust-dev-cert</em></td>
<td></td>
</tr>
</tbody>
</table>
Webpack the module bundler
SPFx usage: bundling all src files to one big JS file
Yeoman
Yeoman: the web's scaffolding tool for modern webapps
SPFx usage:
Scaffolding the necessary files / folders for a SharePoint client-side web part
INSTALLING THE TOOLS
Setup your development machine:

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

Set up your Office 365 environment:

https://docs.microsoft.com/en-us/sharepoint/dev/spfx/set-up-your-developer-tenant
CREATING YOUR FIRST PROJECT
SOLUTION TYPES
Available types

- Web Parts
- Extensions
  - Application Extension
  - Field Customizer Extension
  - ListView Customizer Extension
Web Parts

Available in classic and modern
Client-Side Extensions

- Application Customizer

- Command Set

- Field Customizer
LET US CREATE OUR FIRST PROJECT
yo @microsoft/sharepoint
PROJECT STRUCTURE
TYPESCRIPT
What is TypeScript?
It’s a strongly-typed superset of JavaScript
Benefits of using TypeScript

• Allows for large-scale JavaScript applications
• Follows ECMAScript future proposal
• Supports plain-old JavaScript
• Not excluding any JavaScript feature
• Improved consistency
• IDE support (e.g. Intellisense)
• Compile-time type checking
Benefits of using TypeScript

```typescript
function argumentOverloading(a: number, b: number) {
    // Nothing to do here
}

function argumentOverloading(a: number, b: number): void
    argumentOverloading(1, 2, 3);
```

```typescript
(class) ConferenceRoom.getConferenceRoomInfo(): void
    conf1.getConferenceRoomInfo('Pass a string');
```

```
(class) ConferenceRoomAvailable(1);
```


[ts] Argument of type '1' is not assignable to parameter of type 'boolean'.

START BUILDING
gulp serve | gulp serve --nobrowser
LOCAL WORKBENCH
HTTPS://LOCALHOST:4321/TEMP/WORKBENCH.HTML
HOSTED WORKBENCH
HTTPS://<TENANT>.SHAREPOINT.COM/_LAYOUTS/15/WORKBENCH.ASPX
CLIENT-SIDE WEB PARTS
Client-Side Web Parts

• Can be leveraged both in modern and classic UI
• Provide the same “feeling” of classic web parts
  • But with the “modern” UI/UX
• Can be implemented using
  • Plain JavaScript
  • React
  • Knockout
  • Etc.
Property Pane Controls

- Label
- TextField
- Checkbox
- DropDown
- Toggle
- Button
- Horizontal Rule
- Slider
- Link
Provisioning of Artifacts

- You can deploy custom artifacts through your custom SPFx solutions
- Can be provisioned only
  - Fields/Site Columns
  - Content Types
  - List Instances
  - List instances with custom schema
- It leverages the “old school” feature framework
  - Life could be better 😞 ...
- You need to create a sharepoint/assets folder in the root folder of your project
  - Create elements.xml files and schema.xml files based on CAML syntax
- Declare the assets in the package-solution.json configuration file
Versioning of provisioned artifacts

- You can use the Upgrade Actions of the feature framework
- Supported upgrade actions are
  - ApplyElementManifest
  - AddContentTypeField
- Provide one or more upgradeActions items in the package-solution.json configuration file
- It’s feature framework, the old and well know one
  - And you will struggle with all the challenges of feature framework versioning
Security Model of client-side Web Parts

• Client-side web parts run with the same permissions as the current user
  • They can do whatever the current user can do
• They can be installed by tenant administrators only
  • It is responsibility of tenant admins to trust (or to not trust) a SPFx component
  • In the future it will change at site collection scope ...
• They share the same DOM and the same code with the whole page
  • Another client-side web part can do malicious things
  • Don’t store sensitive data in the client-side web part
  • Rely on server-side infrastructural services
SHAREPOINT FRAMEWORK EXTENSIONS
User navigates to the list, library or page

Server Returns:
- Application Data (List data or Page data)
- SPFx JavaScript Libraries
- Manifest for all active SPFx components (extensions, web parts)
Application starts rendering
SPFx requests the component scripts from their location (normally a CDN)

Application finishes rendering
Web parts and extensions are executed

Note: for field customizers data is not rendered until extension code is executed
List View Command Sets can be used to introduce new custom actions to a list. Can be configured to be active when numerous items are selected.

Specific areas in the page are available for Application Customizers to embed customizations for end users. Application Customizers can be also invisible for the end users.

List View Field customizers can be used to customize experiences around the specific fields. They can be associated to a specific field instance to make a customization execute when it’s used.
Application Customizer

- Build custom UI elements when `onInit()` fires
- Or when placeholders change
- Use `this.context.placeholderProvider` to read available placeholders
- Available placeholders (so far):
  - Top
  - Bottom
- Debug URL
Reference the elements.xml file in package-solution.json

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

  <CustomAction
    Title="SPFxApplicationCustomizer"
    Location="ClientSideExtension.ApplicationCustomizer"
    ClientSideComponentId="46606aa6-5dd8-4792-b017-1555ec0a43a4"
    ClientSideComponentProperties="{"Top":"Top area of the page","Bottom":"Bottom area in the page"}"
  />

</Elements>
```
Command Set

- Enable/Disable commands by overriding `onListViewUpdated(event)` method
- Override `onExecute(event)` method and switch based on `event.commandId`
- Available locations:
  - `ClientSideExtension.ListViewCommandSet.ContextMenu`: ECB
  - `ClientSideExtension.ListViewCommandSet.CommandBar`: Top menu
  - `ClientSideExtension.ListViewCommandSet`: Both
- Debug URL:
  - `?loadSpfx=true&debugManifestsFile=https://localhost:4321/temp/manifests.js&customActions={"a8047e2f-30d5-40fc-b880-b2890c7c16d6":{"location":"ClientSideExtension.ListViewCommandSet.CommandBar","properties":{"sampleTextOne":"One item is selected in the list.","sampleTextTwo":"This command is always visible."}}}"
Command Set Provisioning

<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <CustomAction
    Title="SPFxListViewCommandSet"
    RegistrationId="100"
    RegistrationType="List"
    Location="ClientSideExtension.ListViewCommandSet.CommandBar"
    ClientSideComponentId="5fc73e12-8085-4a4b-8743-f6d02ffe1240"
    ClientSideComponentProperties="{"sampleTextOne":"One item is selected in the list.","sampleTextTwo":"This command is always visible."}"
  />
</Elements>

• Reference the elements.xml file in package-solution.json
Field Customizer

• Override `onRenderCell(event)` method
  • Play with `event.domElement` to replace/update the HTML of the cell

• Debug URL:
  • ?loadSPFX=true&debugManifestsFile=https://localhost:4321/temp/manifests.js&fieldCustomizers={"Percent":{"id":"45a1d299-990d-4917-ba62-7cb67158be16","properties":{"sampleText":"Hello!"}}}
Field Customizer Provisioning

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

<Field ID="{060E50AC-E9C1-3D3C-B1F9-DE0BCAC200F6}"  
   Name="SPFxPercentage" 
   DisplayName="Percentage" 
   Type="Number" 
   Min="0" 
   Required="FALSE" 
   Group="SPFx Columns" 
   ClientSideComponentId="7e7a4262-d02b-49bf-bfcb-e6ef1716aaef">
</Field>
</Elements>

• Reference the elements.xml file in package-solution.json
A BRIEF MOMENT TO THE PAST
The old way of working
Data binding

**HTML**

```html
<body data-ng-app="feedbackApp" data-ng-controller="feedbackController">
  <ul id="tracks">
    <li data-ng-repeat="track in tracks " class="track">
      <h2>{{track.title}}</h2>
      <ul class="trackSessions">
        <li data-ng-repeat="session in track.sessions" class="session">
          <div class="sessionDetails">
            <a data-ng-href="{{session.url}}">{{session.title}}</a>
            <span ng-if="session.speaker">by {{ session.speaker}} ({{ session.start}} - {{session.end}})</span>
            <span class="note" ng-if="session.note">{{session.note}}</span>
          </div>
        </li>
      </ul>
    </li>
  </ul>
</body>
```

**JavaScript**

```javascript
var app = angular.module('feedbackApp', []);
app.controller('feedbackController', ['$scope', '$http', ($scope, $http) => {
  $scope.oneAtATime = true;
  $http.get('tracks.json?ver=0.1').success(data => {
    $scope.tracks = data;
  });

  $http.get('tracks.json?ver=0.1').success((data) => {
    $scope.tracks = data;
  });
});
```
REACT IS A LIBRARY FOR BUILDING USER INTERFACES
ALL ABOUT COMPONENTS
Component based
NO CODE AND HTML SEPERATION
YOU WRITE HTML IN TYPESCRIPT
JSX and TSX

Adds XML syntax to JavaScript / TypeScript

---

**JSX**

```jsx
<a href="https://www.eliostruyf.com" title="Elio Struyf">My blog</a>
```

**Plain JavaScript**

```javascript
```
Variables in JSX / TSX

JSX

```jsx
<a href={blogLink.url} title={blogLink.title}>{blogLink.text}</a>
```

Plain JavaScript

```javascript
React.createElement("a", { href: blogLink.url, title: blogLink.title }, blogLink.text)
```
Simple component

class ComponentName extends React.Component<{}, {}> {
    public render() {
        return (<div>
            Component name should start with a capital letter
        </div>);
    }
}
Using the component component

<(ComponentName) />
class Sample extends React.Component<{}, {}> {
  private _linkClick() {
    console.log('Hey, you clicked me!');
  }

  public render(): React.ReactElement<{}> {
    return (<a href="javascript:;" onClick={this._linkClick}>Click me!</a>);
  }
}
HOW DO WE CONVERT THIS?
You don’t have to worry about it. In SPFx everything is already in place.
JSX / TSX LIMITATIONS
Some keywords are reserved in JS / TS

- `class` → `className`
- `colspan` → `colSpan`
- `innerHTML` → `dangerouslySetInnerHTML`
- `for` → `htmlFor`
- `style` → `style` (it is an object!)
INPUTS (PROPS) AND STATE
Input properties or props

• Passing information to your component
• Do not update properties in your component

```html
<div>
  Description: {this.props.description}
</div>
```
Passing props to components

<ComponentName description="This is the description" />

<ComponentName description="This is the description" />
Component state or state

- Keep hold of the current components variables
- Considered private to the component

```javascript
constructor(props: IProps) {
    super(props);

    // initialize the state
    this.state = {
        show: false
    };
}
```
Update the component state

- Update state by using `setState`
- Think of it as a `request` rather than an immediate command
- For better performance, React may delay it

```javascript
this.setState((prevState, props) => {
  return {
    show: !prevState.show
  };
});
```
LISTING ITEMS
Important things when listing items

• Always use a key
• Try to avoid using the item index

```javascript
public render() {
    const itemList = [{ id: 0, title: "Hello" }, { id: 2, title: "," },
                      { id: 4, title: "World" }];

    return (
      <ul>
        itemList.map(item => {
            return <li key={item.id}>{item.title}</li>;
        })
      </ul>
    );
}
```
VIRTUAL DOM
Virtual DOM and change detection
LIFECYCLE HOOKS
Lifecycle hooks

- **Mounting:**
  - constructor, componentWillMount, componentDidMount

- **Updating:**
  - componentWillReceiveProps, shouldComponentUpdate, 
    componentWillUpdate, componentDidUpdate

- **Unmounting**
  - componentWillUnmount
Web APIs and Permission Scopes

Configuration

- Integrated in Developers Tools
  - Developers specify the Web API they need access to as well as permission scopes as part of the package-solution.json file

- Approved by Administrators
  - Tenant administrators are notified that solution requires access to additional Web APIs and can manage such requests from SharePoint directly

- Be careful: permission grants are tenant-wide, not application-specific
WORKING WITH SCSS IN SPFX
Reusable property pane controls for the SharePoint Framework solutions

This repository provides developers with a set of reusable property pane controls that can be used in SharePoint Framework (SPFx) solutions.

https://sharepoint.github.io/sp-dev-fx-property-controls/

**Attention**

The controls project has a minimal dependency on SharePoint Framework version 1.3.0. Be aware that the controls might not work in solutions your building for on-premises. As for on-premises solutions version 1.1.0 will get used.

**Getting started**

**Installation**

To get started you have to install the following dependency to your project: @pnp/spfx-property-controls.
Reusable React controls for your SharePoint Framework solutions

This repository provides developers with a collection of reusable React controls that can be used in SharePoint Framework (SPFx) solutions. The project provides controls for building web parts and extensions.

⚠️ Attention

The controls project has a minimal dependency on SharePoint Framework version 1.3.0. Be aware that the controls might not work in solutions your building for on-premises. As for on-premises solutions version 1.1.0 is currently used.

Getting started

Installation

To get started you have to install the following dependency to your project: `@pnp/spfx-controls-react`
CALLING APIS AND STORING DATA
Where to call for data

• Do not do async call in the constructor
• Use the `componentDidMount` lifecycle hook

```javascript
public componentDidMount(): void {
  this._getAsyncData().then(items => {
    if (items) {
      this.setState({
        items
      });
    }
  });
}
```
CONSUMING MS GRAPH AND 3RD PARTY API
Consuming the Microsoft Graph

SharePoint Framework
- Web Parts
- Extensions

Microsoft Graph
- Teams
- Groups
- Tasks
- Chats
- Calendar
- Files
- Insights
- Coworkers
- Devices
- People
- Meetings
- Messages
- User

SPS EVENTS Milan
Consuming Web APIs

- The browser (JavaScript) makes a call directly to AAD (via ADAL.js) to obtain the authorization (token) to access a specific resource.

  - App ID: 00000003-0000-0ff1-ce00-000000000000
  - audience: https://graph.microsoft.com
Consuming Web APIs

- AAD checks if in the Service Principal the Web API has been registered and, if yes, returns back the token with the configured (per tenant) permission scopes.
Consuming Web APIs

- Browser then has the necessary token to make the call to Microsoft Graph

GET https://graph.microsoft.com/v1.0/groups/{id}/conversations

Data, Script Tags, SPO bearer Token

SharePoint Online

Azure Active Directory

https://graph.microsoft.com

No Token Returned

App ID: 00000003-0000-0ff1-ce00-000000000000
Audience: https://graph.microsoft.com

No

Yes

Audience Registered

Delegation Token with registered (per Tenant) scopes

ReturnURL: tenant.SharePoint.com

{ } API
MSGraphClient

- Provides fluent API to consume the Microsoft Graph
  - SPFx “edition” of the Microsoft Graph SDK
  - Released in SPFx 1.6
- Main members
  - select
  - expand
  - orderby
  - filter
  - top
  - skip
  - count
  - headers
  - post
  - put
  - create
  - update
  - del
  - get
  - etc.
Web APIs and Permission Scopes Configuration

• Uses AAD implicit flow and ADAL.JS library
  • Part of SPFx client libraries
  • Supports delegated tokens only (no app-only)
• Tenant Admins can control access through per tenant AAD Service Principal
• Managed by SharePoint Online infrastructure
A Common Business Requirement
Architecture

SharePoint Framework
- Web Parts
- Extensions

Azure AD Application
- Custom REST API

LOB System
- Custom API
- DBMS
AadHttpClient

• Provides integrated API to consume external APIs
  • Released in SPFx 1.6
• To consume third-party APIs or even Microsoft Graph
• Same approach and architecture of MSGraphClient
  • Service Principal with tenant-wide granted permission scopes
  • Permissions are not application-specific
WHAT IS SHAREPOINT PATTERNS AND PRACTICES?
SHAREPOINT PNP = SHAREPOINT DEV COMMUNITY
Why wasting time on figuring things by yourself?

What if there would be starting point and assistance to get started?
Reusable open-source components and solutions built together with the engineering
2013
**Internal**
Internal initiative in Microsoft to start collecting learnings from add-in model.

2014
**Going public**
Going first public in CodePlex, then moved to GitHub. Concentration on add-in model enterprise scenarios.

2015
**Reusability**

2016
**Partnership**
PnP ownership moved to engineering, team works with a partnership on aligning roadmap. Work with SP Framework starts.

2017
**Integration**
SharePoint Developer Community = SharePoint PnP. End-to-end coordination of the SharePoint development guidance and engagement.

2018
**Solutions**
From reusable components alignment to reusable solutions. Concentration on modern transformation and experiences.

- More than 450 samples to learn and take advantage in your work.
- More than 840 contributors in GitHub organization.
- More than 52,000 unique visitors during past 2 weeks in GitHub repositories.
- More than 50,000 tenants have used open-source PnP components and controls in SharePoint Online.
- More than 1.9 million watch time minutes in SP Dev YouTube channel in past 365 days.
PnP JS

• Collection of fluent libraries for JS/TypeScript
• Type-safe client code for:
  • SharePoint REST
  • Graph
  • Office 365 REST APIs
• https://pnp.github.io/pnpjs/
CODE IS READY, LET’S SHIP IT!
Client-side solution creation flow

- **yo @microsoft/sharepoint**: Scaffold SharePoint Web Part Project
- **code .**: Build Web Part Code

**gulp serve**
- Test
- Local
- UAT / Pre-production

**Ship?**

**gulp bundle --ship**
- Minify Assets
- Package/Deploy

**Release using app catalog**
- Available on Classic and Client-Side Pages
Manual steps for shipping your solutions

- Clean your solution
  → gulp clean
- Bundle your solution
  → gulp bundle --ship
- Package your solution
  → gulp package-solution --ship
- (Optionally) upload your files to your CDN
  • Or use the native CDN provided by SPO
- Upload the solution package and deploy it in the app catalog (site or tenant)
```json
{
    "$schema": "https://dev.office.com/json-schemas/spfx-build/package-solution.json",
    "solution": {
        "name": "alm-demo-client-side-solution",
        "id": "64b7e3b0-6124-42d-8b5-b8e91c14d45",
        "version": "1.0.0.0",
        "includeClientSideAssets": true,
        "skipFeatureDeployment": true
    },
    "paths": {
        "zippedPackage": "solution/alm-demo.sppkg"
    }
}
```
"cdnBaseUrl": "←!→ PATH TO CDN →"
SHAREPOINT APP CATALOG
{  
  "$schema": "https://dev.office.com/json-schemas/spfx-build/package-solution.json",  
  "solution": {  
    "name": "alm-demo-client-side-solution",  
    "id": "64bcb7b0-6124-4c2d-bad5-e27e91c14d45",  
    "version": "1.0.0.0",  
    "includeClientSideAssets": true,  
    "skipFeatureDeployment": true  
  },  
  "paths": {  
    "zippedPackage": "solution/alm-demo.sppkg"  
  }  
}
Do you trust alm-demo-client-side-solution?

The client-side solution you are about to deploy contains full trust client side code. The components in the solution can, and usually do, run in full trust, and no resource usage restrictions are placed on them.

This client side solution will get content from the following domains:

SharePoint Online

☐ Make this solution available to all sites in the organization

If you clear this setting, users won't be able to add the web part to pages. The web part will continue to work if it was already added to pages.
Tenant scoped deployment impact for deployment

Tenant scoped deployment enabled?

- Components immediately available in sites
- Web parts visible immediately in sites

- Solution need to be explicitly installed to each site
- Possible feature framework elements applied in installation
Tenant-wide Extensions

• Reusable components (extensions) available tenant-wide

• Sample use cases
  • Use the same header/footer across all tenant
  • Use the same
QUESTIONS & ANSWERS
Thank You!