

**You**

TOTAL INTERVIEW QNAs FOR SPFX WITH KNOCKOUT JS DEVELOPMENT, CUSTOMIZATION, WORKFLOWS, PLUGINS, HTML,CSS,XML JS & UNIFIED INTERFACE INTEGRATION IN A TABULAR REPRESENTATION WITH ALL FEATURES MENTIONED POINT WISE, BEST PRACTICES, HOW TO OPTIMISE IT, PROS & CONS, WHAT'S LATEST ADVANCEMENTS !!

**Copilot**

Certainly! SharePoint Framework (SPFx) is a popular technology for building custom solutions in SharePoint. If you're preparing for an SPFx interview, here are some key points and questions to consider:

### 1. Delegate Controls in SharePoint:

- Three new delegate controls were introduced to display the top suite bar with links like SkyDrive, NewsFeed, Sync, and follow:
  - SuiteBarBrandingDelegate
  - SuiteLinksDelegate
  - PromotedActions Delegate Control.

### 2. Changes in CSOM and REST APIs:

- Microsoft improved both Client Side Object Model (CSOM) and Representational State Transfer (REST) based APIs.
- Added support for Search, User Profiles, Taxonomies, and Publishing Object Model.
- The Client.svc service now accepts HTTP GET, PUT, and POST requests.

### 3. App Permissions in SharePoint:

- An app for SharePoint requests permissions during installation.
- Developers specify required permissions in the app manifest file.

### 4. SPSecurityEventReceiver:

- SharePoint introduced a Cloud App Model for creating apps.
- Apps for SharePoint are self-contained functionality pieces that extend SharePoint capabilities.
- They can include SharePoint components (lists, workflows, site pages) or surface remote web applications and data in SharePoint.

### 5. Base Classes in SPFx:

- Two base classes: `Microsoft.SharePoint.WebPartPages.WebPart` and SharePoint WebPart base class.
- Use the SharePoint WebPart base class for backward compatibility.
- Benefits of using it:
  - Cross-page connections
  - Connections between Web Parts outside a Web Part zone
  - Client-side connections (Web Part Page Services Component).

Remember to review these topics thoroughly and practice answering related questions. Good luck with your interview! 😊

**You**

INCLUDE MORE KNOCKOUT JS TOP INTERVIEW QNAs

**Copilot**

Certainly! Here are some additional Knockout.js interview questions to help you prepare:

1. **How does Knockout.js support data-binding?** Knockout.js enables two-way data binding by using observables. These observables track changes in the model and reflect them in the UI, and vice versa. For example:

```
function AppViewModel() {  
    this.firstName = ko.observable('Bob');  
    this.lastName = ko.observable('Smith');  
}
```

In the HTML, you can bind these properties using the data-bind attribute:

```
<p>First name: <input data-bind="value: firstName"></p>  
<p>Last name: <input data-bind="value: lastName"></p>
```

When you change the input field's value, Knockout.js automatically updates the corresponding property in your view model.

2. **What are observables in Knockout.js, and how do they differ from plain JavaScript objects?** Observables in Knockout.js automatically detect dependencies and update the UI when data changes. Unlike plain JavaScript objects, Knockout's observables are functions that maintain state and notify subscribers about changes. They provide get and set methods. When an observable is modified, Knockout.js triggers a UI update, ensuring synchronization between the model and view.
3. **How would you handle dependency tracking in Knockout.js?** Dependency tracking is automatic in Knockout.js due to observables. When an observable changes, any computed observables or bindings that depend on it are automatically updated.
4. **Explain the difference between Knockout's computed and pureComputed functions.**
  - ko.computed: Automatically tracks dependencies and updates when its dependencies change.
  - ko.pureComputed: Similar to ko.computed, but only updates when its dependencies change and not during intermediate evaluations. Useful for performance optimization.
5. **What are Knockout's templates, and how do they improve code reusability?** Knockout templates allow you to define reusable UI components. They separate the UI structure from the data, making it easier to maintain and reuse code. Templates can be used with the foreach binding to render lists of items dynamically.

Feel free to explore these questions further, and good luck with your interview preparation! 😊🚀

**You**

Tell me about custom bindings in Knockout.js.

**Copilot**

Certainly! Custom bindings in Knockout.js allow you to create your own reusable data-binding handlers. These custom bindings extend the built-in bindings provided by Knockout and enable you to define specific behavior for your UI elements.

Here's how you can create a custom binding:

### 1. Define the Custom Binding:

- To create a custom binding, use the ko.bindingHandlers object. You'll specify a name for your custom binding (e.g., "myCustomBinding").
- Define an object with properties that handle different aspects of the binding:
  - init: Called when the binding is first applied to an element. Set up any initial state or event listeners.

- **update:** Called whenever the associated observable value changes. Update the UI or perform any necessary actions.
- Optionally, you can define other properties like `after`, `before`, and `dispose` for additional lifecycle management.

## 2. Usage in HTML:

- In your HTML, use the `data-bind` attribute to apply your custom binding to an element. For example:

```
<div data-bind="myCustomBinding: someObservable"></div>
```

Here, `someObservable` is the data you want to bind to your custom behavior.

## 3. Implement the Custom Behavior:

- Inside your custom binding definition, access the observable value using the `valueAccessor()` function.
- You can manipulate the DOM, set event listeners, or perform any other actions based on the observable value.
- Don't forget to call `ko.utils.domNodeDisposal.addDisposeCallback(element, callback)` to clean up any resources when the element is removed from the DOM.

## 4. Example Custom Binding: Let's say you want to create a custom binding that highlights an element's background color based on a boolean value. Here's a simplified example:

```
ko.bindingHandlers.highlightBackground = {
  init: function (element, valueAccessor) {
    var value = ko.unwrap(valueAccessor()); // Get the observable value
    if (value) {
      element.style.backgroundColor = 'yellow';
    }
  },
  update: function (element, valueAccessor) {
    var value = ko.unwrap(valueAccessor());
    if (value) {
      element.style.backgroundColor = 'yellow';
    } else {
      element.style.backgroundColor = ''; // Reset background color
    }
  }
};
```

In your HTML:

```
<div data-bind="highlightBackground: isHighlighted"></div>
```

Remember to adjust the custom binding behavior according to your specific requirements. Custom bindings are powerful tools for creating dynamic and customized interactions in your Knockout.js applications! 😊🚀

## You

Can you show me an example of a more complex custom binding?

## Copilot

Certainly! Let's dive into a more complex custom binding example in Knockout.js. Custom bindings allow you to create powerful and reusable interactions between your view model and the DOM elements.

Suppose we want to create a custom binding that integrates with a jQuery UI component, such as an autocomplete input. We'll build a custom binding called "autocomplete" that wraps the jQuery UI Autocomplete functionality.

Here's how you can create this custom binding:

## 1. Register the Custom Binding:

```
ko.bindingHandlers.autocomplete = {
  init: function (element, valueAccessor) {
    // Initialize the Autocomplete widget
    var options = ko.unwrap(valueAccessor());
    $(element).autocomplete(options);
  },
  update: function (element, valueAccessor) {
    // Handle updates (e.g., when the observable value changes)
    var options = ko.unwrap(valueAccessor());
    $(element).autocomplete("option", options);
  }
};
```

## 2. Usage in HTML: In your HTML, use the data-bind attribute to apply the custom binding:

```
<input type="text" data-bind="autocomplete: { source: availableTags }">
```

Here, availableTags is an observable array containing the autocomplete suggestions.

## 3. Example View Model:

```
function MyViewModel() {
  this.availableTags = ko.observableArray(["Apple", "Banana", "Cherry"]);
}

ko.applyBindings(new MyViewModel());
```

## 4. Explanation:

- The `init` function initializes the Autocomplete widget when the binding is first applied to an element. It accepts an options object (e.g., `{ source: availableTags }`).
- The `update` function handles updates (e.g., when the observable value changes). It ensures that the Autocomplete options are kept in sync with the observable data.

Now, whenever the `availableTags` array changes, the Autocomplete widget updates accordingly. You can adapt this pattern for other jQuery UI components or any custom behavior you need.

Feel free to customize this example further based on your specific requirements. Happy coding! 😊🚀

## You

Can you explain the lifecycle of a custom binding in more detail?

## Copilot

Certainly! The lifecycle of a custom binding in Knockout.js involves two key phases: **initialization** and **update**. Let's break it down:

### 1. Initialization Phase:

- When a custom binding is first applied to an element, the `init` function is called.
- In the `init` function, you can:
  - Set up initial state (e.g., event listeners, UI configuration).
  - Access the observable values associated with the binding.
  - Perform any necessary setup for your custom behavior.
- Parameters passed to the `init` function:
  - `element`: The DOM element involved in the binding.
  - `valueAccessor`: A function to get the current model property associated with the binding.
  - `allBindings`: An object containing all model values bound to the element.
  - `viewModel` (deprecated): Use `bindingContext.$data` instead.
  - `bindingContext`: Provides access to the binding context (including parent data).

## 2. Update Phase:

- The update function is called initially when the binding is applied and whenever any observable/computed accessed by the binding changes.
- In the update function, you can:
  - Update the DOM element based on the supplied values (e.g., modify styles, content).
  - React to changes in observables or computed values.
- Parameters passed to the update function are the same as in the init phase.

3. **Example Custom Binding:** Suppose you create a custom binding called "slideVisible" that animates an element's visibility using jQuery's `slideUp` and `slideDown`:

```
ko.bindingHandlers.slideVisible = {  
  update: function(element, valueAccessor) {  
    var isVisible = ko.unwrap(valueAccessor());  
    if (isVisible) {  
      $(element).slideDown();  
    } else {  
      $(element).slideUp();  
    }  
  }  
};
```

In your HTML:

```
<div data-bind="slideVisible: shouldShowElement"></div>
```

Remember that you can choose to provide either the init or update callback based on your needs. Custom bindings empower you to encapsulate sophisticated behaviors and create reusable interactions! 😊🚀

For more details, refer to the [official Knockout documentation](#).