Developing Cross-Platform Web Apps With Blazor

Wael Kdouh - @waelkdouh

Senior Customer Engineer

v1.0

Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at

http://www.microsoft.com/about/legal/permissions/

How to View This Presentation

- To switch to **Notes Page** view:
  - On the ribbon, click the **View** tab, and then click **Notes Page**

- To navigate through notes, use the Page Up and Page Down keys
  - Zoom in or zoom out, if required

- In the **Notes Page** view, you can:
  - Read any supporting text—now or after the delivery
  - Add notes to your copy of the presentation, if required

- Take the presentation files home with you

Module 8: JavaScript Two-Way Interop

Module Overview

Module 8: JavaScript Two-Way Interop

Section 1: JavaScript Two-Way Interop

Lesson: Overview

JavaScript Interop

- At present, there are a number of features WebAssembly does not support, therefore Blazor does not natively support them

- These are typically browser API features such as:
  - Media Capture
  - Popups
  - Web GL
  - Web Storage

- To access these browser features you need to use JavaScript as an intermediary between Blazor and the Browser

Javascript Interop Caveats

- There are a few caveats when working with JSInterop:
  - Do not invoke JSInterop during the server pre-rendering phase
  - Do not use ElementReference objects too soon
  - Avoid memory leaks by disposing of resources
  - Avoid invoking methods on disposed .NET references
  - Do not invoke .NET methods before Blazor has initialized

Module 8: JavaScript Two-Way Interop

Section 1: JavaScript Two-Way Interop

Lesson: Calling JavaScript From .NET

Calling JavaScript From .NET

- JavaScript can be added into either /Pages/_Host.cshtml in Server-side Blazor apps, or in wwwroot/index.html for Web Assembly Blazor apps

- JavaScript can then be invoked from Blazor by injecting the IJSRuntime service into the component

```
public interface IJSRuntime
{
  ValueTask<TValue> InvokeAsync<TValue>(string identifier, object[] args);
  ValueTask<TValue> InvokeAsync<TValue>(string identifier, CancellationToken
cancellationToken, object[] args);
  // Via an extension class
  void InvokeVoid(string identifier, params object[] args);
}
```

The identifier must be a JavaScript function scoped to the global window variable, but it is not necessary to include window in the identifier

So, to invoke window.alert you only need to specify alert as the identifier

Blazor JavaScript isolation and object references

- Starting with .Net 5 Blazor enables you to isolate your JavaScript as standard JavaScript modules. This has a couple of benefits:
  - Imported JavaScript no longer pollutes the global namespace
  - Consumers of your library and components no longer need to manually import the related JavaScript

Demo: Calling JavaScript From .NET & JavaScript Isolation

Passing Parameters

- The previous example passed the string “Hello world” as a parameter to the JavaScript alert function. It is also possible to pass complex objects to JavaScript

- Parameters are serialized to JSON and then deserialized in JavaScript before being passed by-value as an anonymous object type to the function being invoked

- All parameter types passed to JavaScript must be basic types (string / int / etc) or be JSON serializable

Demo: Passing Parameters

Accessing Javascript Return Values

- So far only the IJSRuntime extension method InvokeVoidAsync have been used

- If you want to receive the return value from a JavaScript function, you need to use the InvokeAsync<TValue> method

Demo: Accessing Javascript Return Values

Module 8: JavaScript Two-Way Interop

Section 1: JavaScript Two-Way Interop

Lesson: Calling .Net From JavaScript

Calling .Net From JavaScript

- Sometimes the .NET application code needs to be executed from JavaScript

- Blazor enables asynchronously calling methods on instances of objects, or static methods on classes

Identifying Invokable .NET Code

- Blazor does not allow JavaScript to call just any static or instance method in the .NET code

- There are conditions
  - The **method** must be **decorated with the JsInvokableAttribute**
  - The **method** must be **public**
  - The **parameters** of the method must be **Json serializable**
  - The **return type** of the method must be **Json serializable**, void, a Task, or a Task<T> where T is Json serializable
  - If specifying the identifier parameter on JsInvokable, the **value must be unique per class hierarchy** (**if an instance method**) or **unique per assembly** (**if a static method**)

Making .NET Code Invokable

- To call a method on a .NET object instance, you first need to pass a reference to the object over to JavaScript

- You cannot pass the object directly because you want to give JavaScript a reference to the object rather than a Json serialized representation of its state

- This is achieved by creating an **instance of the DotNetObjectReference** class

Demo: Calling .Net From JavaScript

Module 8: JavaScript Two-Way Interop

Section 1: JavaScript Two-Way Interop

Lesson: Lifetimes and Memory Leaks

Lifetimes And Memory Leaks

- Inspecting the browser console window for the previous example, you will see that upon navigating to another page JavaScript is still calling back the component

- What's worse, if you look in the Visual Studio output window you will see that the component is still being invoked and outputting the values passed from JavaScript, which means the component has not been garbage collected

- When creating a DotNetObjectReference, Blazor will generate a unique ID (integer for WASM, GUID for server side) and store a lookup to the object in the current JSRuntime

- This means that unless the references are correctly disposed, the app is going to leak memory

Lifetimes And Memory Leaks

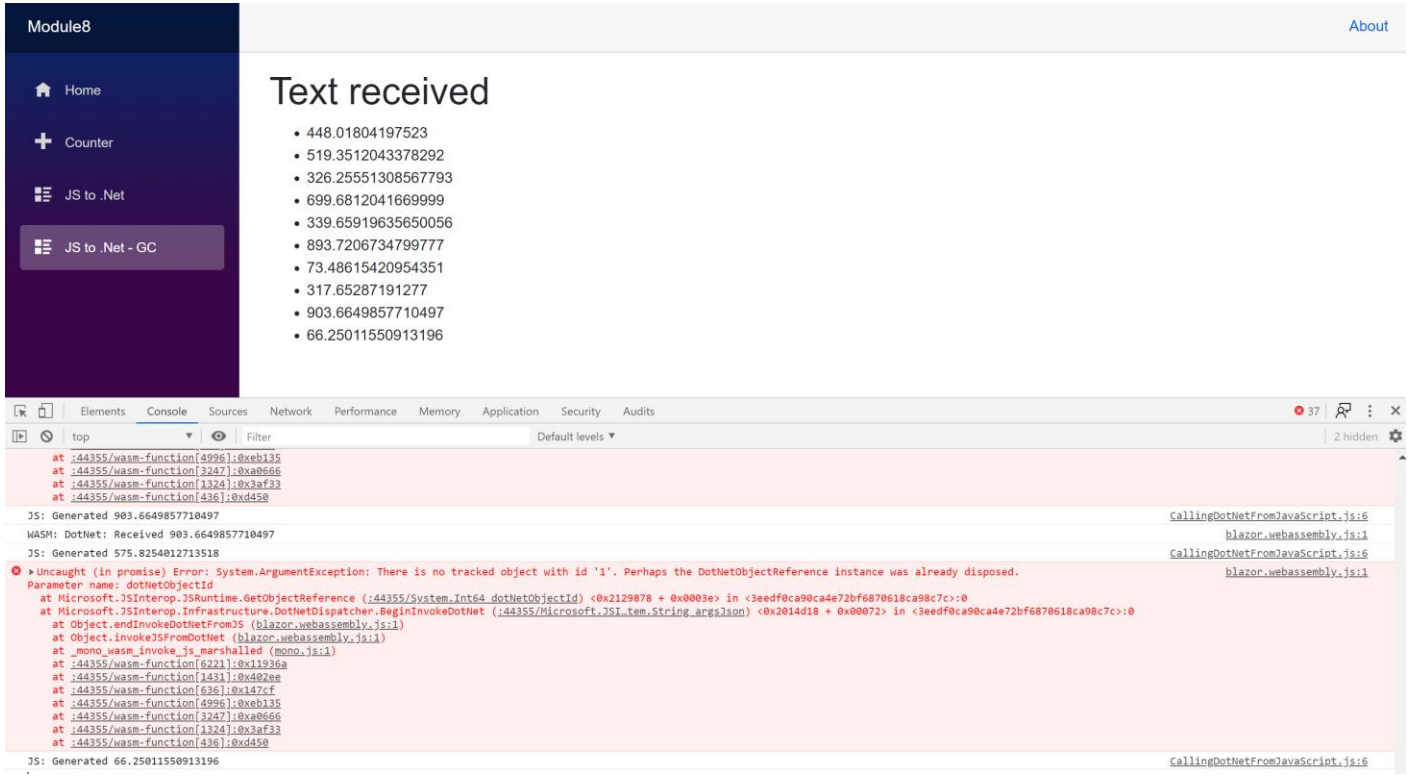
- The DotNetObjectReference class implements IDisposable. To solve the memory leak problem you need to do the following:
  - The component should keep a reference to the DotNetObjectReference
  - The component should implement IDisposable and dispose the DotNetObjectReference

Demo: Lifetimes And Memory Leaks

Invoking Methods On Disposed .NET References

- If you run the previous example the component no longer receives random numbers from JavaScript after implementing the Dispose method

- However, if you look in the browser's console window you will see an error being raised every second



Invoking Methods On Disposed .NET References

- Once the DotNetObjectReference has been disposed it is removed from the JSRuntime, allowing the component to be garbage collected
- As a result of the cleanup the reference is no longer valid and should not be used by JavaScript
- The component needs to be modified to cancel the JavaScript setInterval so it is no longer executed once the component has been destroyed

Demo: Avoid Invoking Methods On Disposed .NET References

Avoid Reverting Back To Your Comfort Zone

- Having access to JavaScript doesn't mean you should revert to utilizing familiar libraries like jQuery
  - For example if you are trying to use bootstrap in your application its better to introduce Blazor components that abstract the JavaScript code in the background

Demo: BlazorStrap

Module Summary

- In this module, you learned about:
  - Introduction to JavaScript Two-Way Interop
  - Calling JavaScript From .Net
  - Passing Parameters To JS
  - Access JavaScript Return Values
  - Calling .Net From JavaScript
  - Lifetimes and Memory Leaks

Lab 8: JavaScript Two-Way Interop

References

- Microsoft Docs

- Blazor University


Microsoft