Blazor

Module 8: JavaScript Interop

Student Lab Manual

Instructor Edition (Book Title Hidden Style)

**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.

© 2020 Microsoft Corporation. All rights reserved.

**Copyright and Trademarks**

© 2020 Microsoft Corporation. All rights reserved.

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/en-us/legal/intellectualproperty/Permissions/default.aspx>

Internet Explorer, Microsoft, SQL Server, Visual Studio, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Contents

[Lab 8: JavaScript Interop 1](#_Toc38805781)

[Exercise 1: Track Orders 3](#_Toc38805782)

# Lab 8: JavaScript Interop

#### Introduction

The aim of this exercise is to explore implementing an application using the new application model introduced under ASP.Net Core 3 called Blazor.

Important

**Blazor WebAssembly in preview**

Blazor Server is supported in ASP.NET Core 3.0. Blazor WebAssembly is in preview for ASP.NET Core 3.1.

#### Objectives

After completing this lab, you will be able to:

* Execute JavaScript code from within Blazor Components

#### Prerequisites

None

#### Scenario

In this scenario, we will explore implementing an online Pizza Delivery application.

#### System Requirements

* Follow the instructions found [on this page](https://docs.microsoft.com/en-us/aspnet/core/blazor/get-started?view=aspnetcore-3.1&tabs=visual-studio) to get started

#### Estimated Time to Complete This Lab

30 minutes

Exercise 1: Track Orders

#### Objectives

In this exercise, you will:

* Introduce the map to track the orders
* Introduce a confirmation dialog to confirm deleting an item from the order

#### Scenario

Users of the pizza store can now track the status of their orders in real time. In this session we'll use JavaScript interop to add a real-time map to the order status page that answers the age-old question, "Where's my pizza?!?".

Task 1: The Map Component

1. Start by opening the solution file BlazingPizza.sln located under \Labs\Module 08 -JavaScript Interop\Begin.
2. Included in the ComponentsLibrary project is a prebuilt Map component for displaying the location of a set of markers and animating their movements over time. We will use this component to show the location of the user's pizza orders as they are being delivered, but first let's look at how the Map component is implemented.

Open *Map.razor* and take a look at the code:

@using Microsoft.JSInterop

@inject IJSRuntime JSRuntime

<div id="@elementId" style="height: 100%; width: 100%;"></div>

@code {

string elementId = $"map-{Guid.NewGuid().ToString("D")}";

[Parameter] public double Zoom { get; set; }

[Parameter] public List<Marker> Markers { get; set; }

protected override async Task OnAfterRenderAsync(bool firstRender)

{

await JSRuntime.InvokeVoidAsync(

"deliveryMap.showOrUpdate",

elementId,

Markers);

}

}

The Map component uses dependency injection to get an IJSRuntime instance. This service can be used to make JavaScript calls to browser APIs or existing JavaScript libraries by calling the InvokeVoidAsync or InvokeAsync<TResult> method. The first parameter to this method specifies the path to the JavaScript function to call relative to the root window object. The remaining parameters are arguments to pass to the JavaScript function. The arguments are serialized to JSON so they can be handled in JavaScript. The Map component first renders a <div> with a unique ID for the map and then calls the deliveryMap.showOrUpdate function to display the map in the specified element with the specified markers pass to the Map component. This is done in the OnAfterRenderAsync component lifecycle event to ensure that the component is done rendering its markup. The deliveryMap.showOrUpdate function is defined in the *wwwroot/deliveryMap.js* file, which then uses *leaflet.js* and OpenStreetMap to display the map. The details of how this code works isn't really important - the critical point is that it's possible to call any JavaScript function this way. How do these files make their way to the Blazor app? For a Blazor library project (using Sdk="Microsoft.NET.Sdk.Razor") any files in the *wwwroot/* folder will be bundled with the library. The server project will automatically serve these files using the static files middleware. The final link is for the page hosting the Blazor client app to include the desired files (in our case .js and .css). The *index.html* includes these files using relative URIs like *\_content/BlazingPizza.ComponentsLibrary/localStorage.js*. This is the general pattern for references files bundled with a Blazor class library - \_content/<library name>/<file path>. Now, when you run the app, you should be able to reach the checkout page by clicking the Order button, and from there can click Place order to confirm it.

1. If you start typing in Map, you'll notice that the editor doesn't offer completion for it. This is because the binding between elements and components are governed by C#'s namespace binding rules. The Map component is defined in the BlazingPizza.ComponentsLibrary.Map namespace, which we don't have an @using for.
2. Add a @using for this namespace to the root *\_Imports.razor* to bring this component into scope:

@using BlazingPizza.ComponentsLibrary.Map

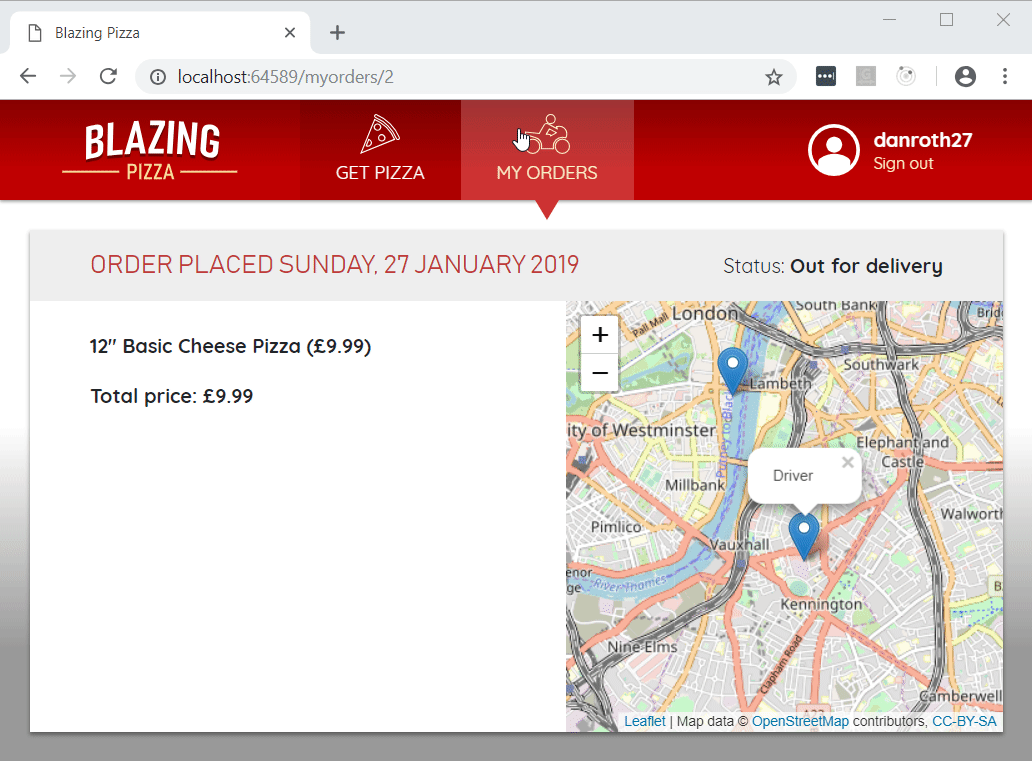
1. Add the Map component to the OrderDetails page by adding the following just below the track-order-details <div>:

<div class="track-order-map">

<Map Zoom="13" Markers="orderWithStatus.MapMarkers" />

</div>

The reason why we haven't needed to add @using for our components before now is that our root *\_Imports.razor* already contains a @using BlazingPizza.Shared, which matches the reusable components we have written. When the OrderDetails component polls for order status updates, an update set of markers is returned with the latest location of the pizzas, which then gets reflected on the map.

[](https://user-images.githubusercontent.com/1874516/51807322-6018b880-227d-11e9-89e5-ef75f03466b9.gif)

Task 2: Add A Confirm Prompt For Deleting Pizzas

1. The JavaScript interop code for the Map component was provided for you. Next, you'll add some JavaScript interop code of your own. It would be a shame if users accidentally deleted pizzas from their order (and ended up not buying them!). Let's add a confirm prompt when the user tries to delete a pizza. We will show the confirm prompt using JavaScript interop.
2. Add a static JSRuntimeExtensions class to the *BlazzingPizza.Client* project with a Confirm extension method off of IJSRuntime. Implement the Confirm method to call the built-in JavaScript confirm function.

public static class JSRuntimeExtensions

{

public static ValueTask<bool> Confirm(this IJSRuntime jsRuntime, string message)

{

return jsRuntime.InvokeAsync<bool>("confirm", message);

}

}

1. Inject the IJSRuntime service into the Index component so that it can be used there to make JavaScript interop calls.

@page "/"

@inject HttpClient HttpClient

@inject OrderState OrderState

@inject NavigationManager NavigationManager

@inject IJSRuntime JS

1. Add an async RemovePizza method to the Index page that calls the Confirm method to verify if the user really wants to remove the pizza from the order.

async Task RemovePizza(Pizza configuredPizza)

{

if (await JS.Confirm($"Remove {configuredPizza.Special.Name} pizza from the order?"))

{

OrderState.RemoveConfiguredPizza(configuredPizza);

}

}

1. In the Index component update the event handler for the ConfiguredPizzaItems to call the new RemovePizza method.

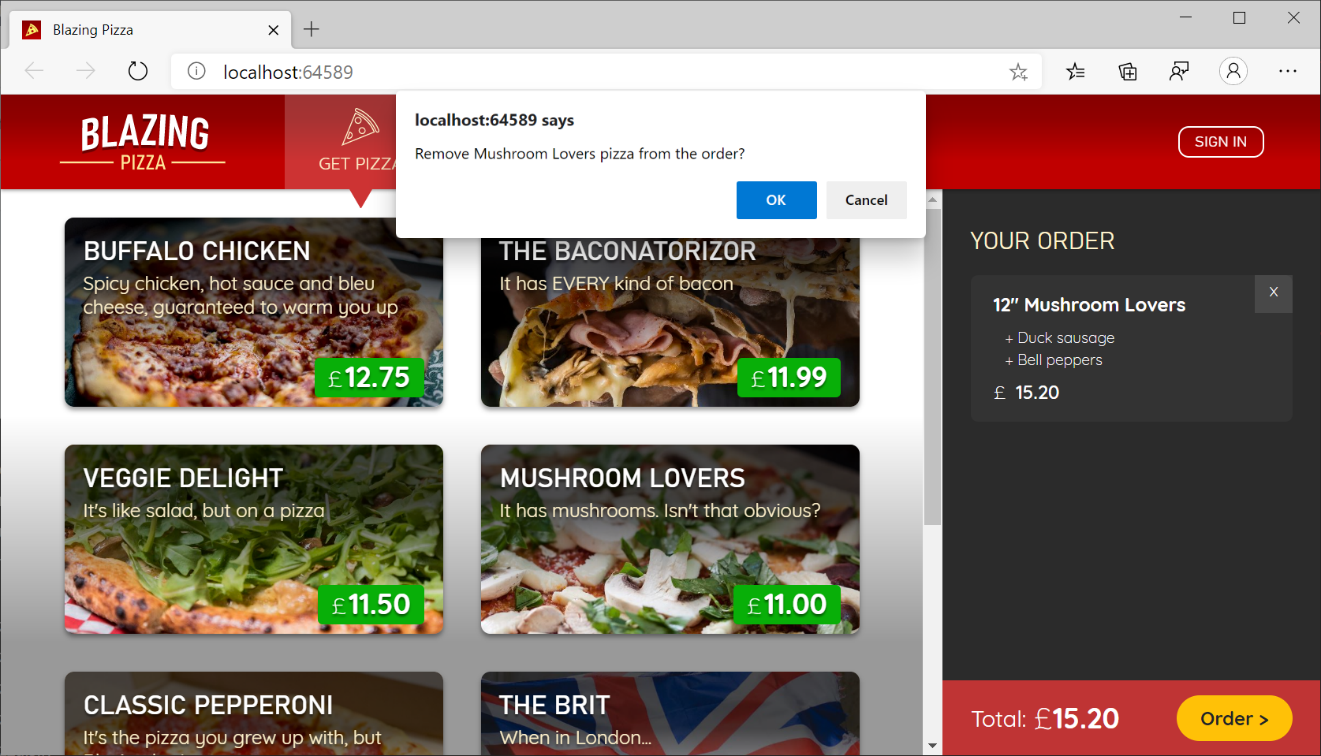
@foreach (var configuredPizza in OrderState.Order.Pizzas)

{

<ConfiguredPizzaItem Pizza="configuredPizza" OnRemoved="@(() => RemovePizza(configuredPizza))" />

}

1. Run the app and try removing a pizza from the order.

[](https://user-images.githubusercontent.com/1874516/77243688-34b40400-6bca-11ea-9d1c-331fecc8e307.png)

Notice that we didn't have to update the signature of ConfiguredPizzaItem.OnRemoved to support async. This is another special property of EventCallback, it supports both synchronous event handlers and asynchronous event handlers.