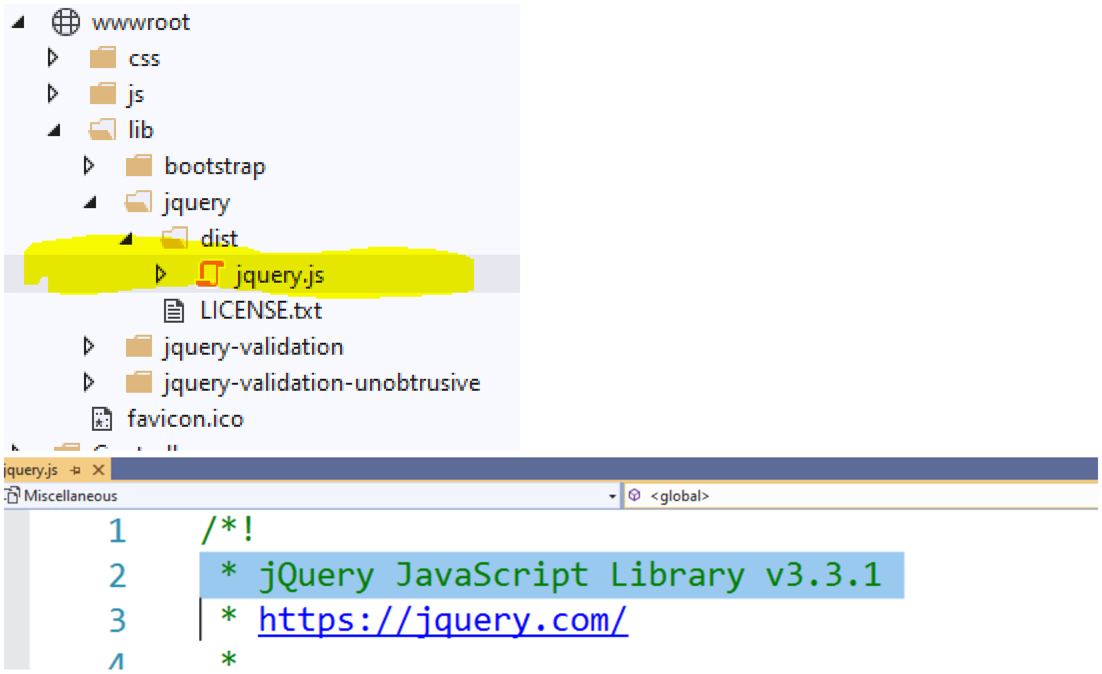
Introduction

Much work is often done on the client side of development. JavaScript is essential for dynamic pages and enhanced user experience. From client side validation to asynchronous calls to the server. From manipulation of html and styles to orchestrating single page applications. jQuery helps us by providing libraries that encapsulate JavaScript in easy to use functions. Ajax is just the name provided for the jQuery functions that provide asynchronous activity.

In this exercise, we'll walk through some of the common functionality we use every day in developing good MVC applications.

Enabling jQuery in the App

jQuery is included in MVC projects when you create them in Visual Studio:



```
/*!
 * jQuery JavaScript Library v3.3.1
 * https://jquery.com/
 *
```

If you want to work with jQuery just in a plain old html file, then include a CDN link:

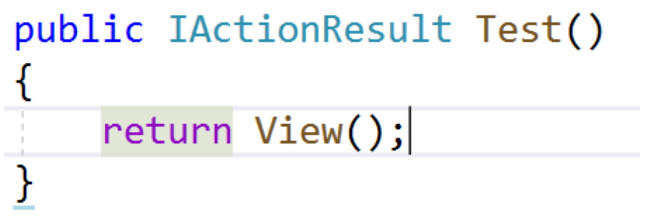
```
<script   src="https://code.jquery.com/jquery-3.5.0.js"   integrity="sha256-
r/AaFHrszJtwpe+tHyNi/XCfMxYpbsRg2Uqn0x3s2zc="   crossorigin="anonymous"></script>
```

In the _Layout.cshtml file(the master page), the link is found at the bottom of the html markup:

```
<script src="~/lib/jquery/dist/jquery.min.js"></script>
```

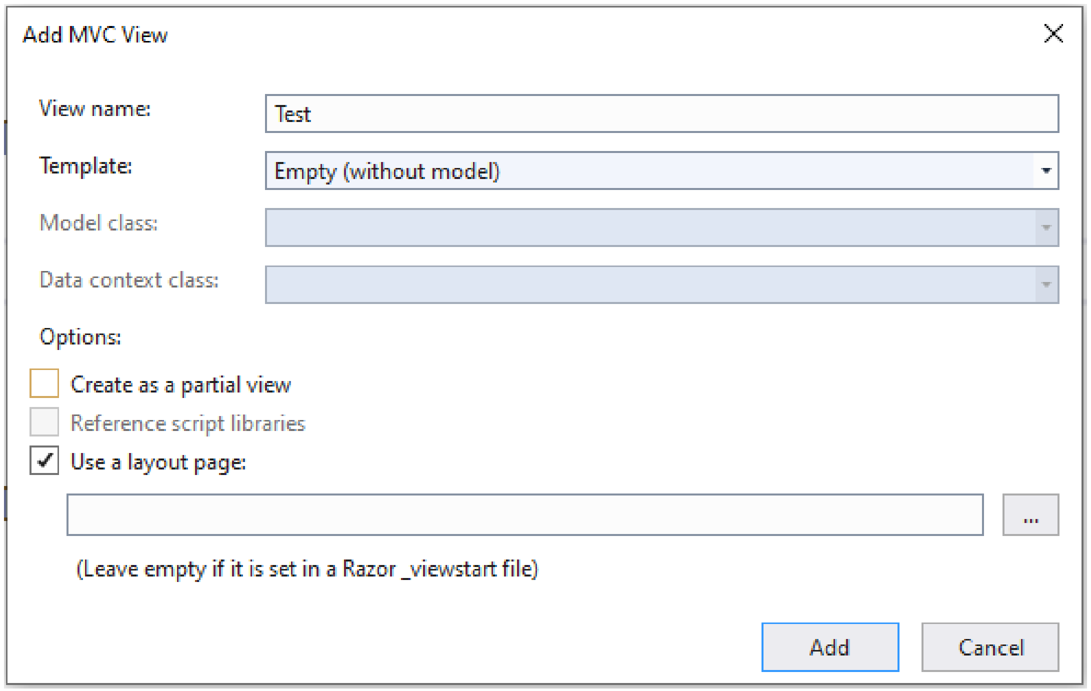
Is the DOM Ready

Any view added to the project and called from a controller can be used for this exercise. Add a controller method called Test and a view that can be called from this method. I am using the Home controller in these examples.



```
public IActionResult Test()
{
    return View();
}
```

Click View(), right click and select “Add View”. Name should Test. Empty template.



Add MVC View

View name: Test

Template: Empty (without model)

Model class:

Data context class:

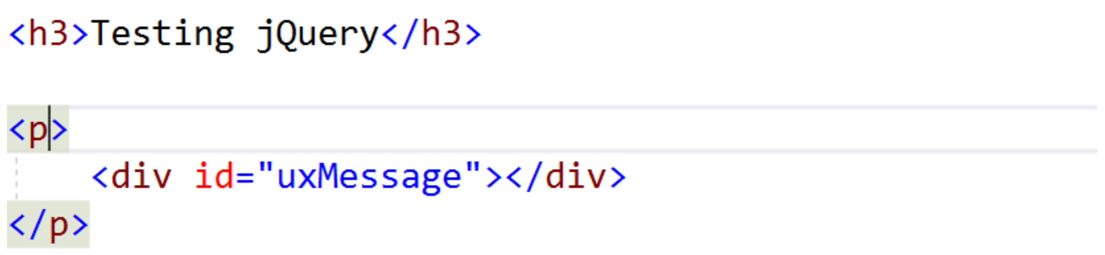
Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [x] Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

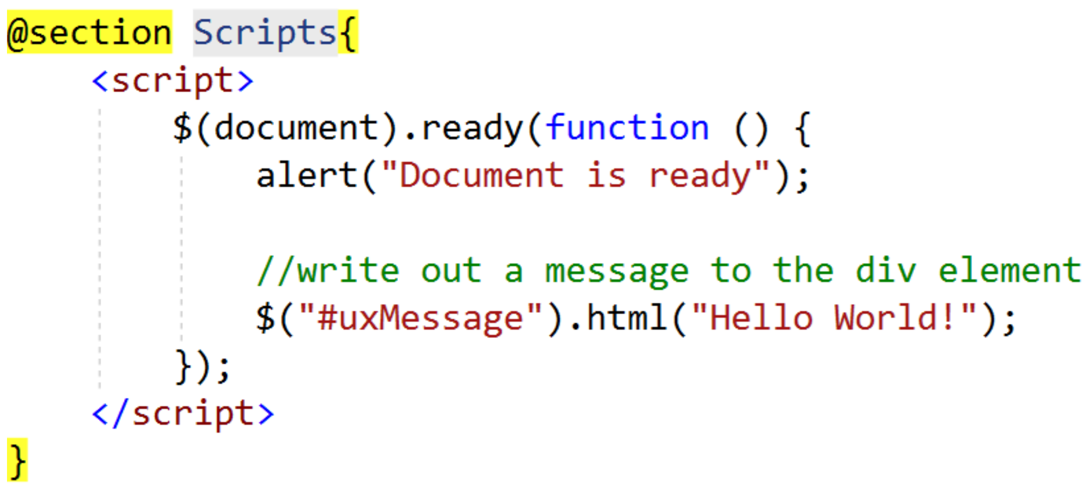
After the view is created, type the code on the following page into this file and run the app

. The url is: /Home/Test (after http://localhost:12345 of course where 12345 is your port #)



```
<h3>Testing jQuery</h3>

<p>
    <div id="uxMessage"></div>
</p>
```



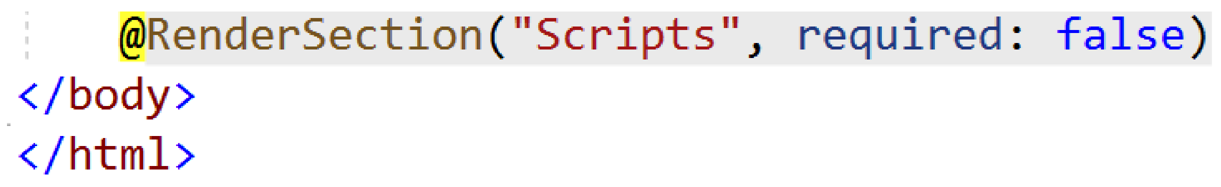
```
@section Scripts{
    <script>
        $(document).ready(function () {
            alert("Document is ready");

            //write out a message to the div element
            $("#uxMessage").html("Hello World!");
        });
    </script>
}
```

The script tag contains the jQuery document.ready function. You can start with just the alert line of code and run the app. This ensures the jQuery is working before writing tons of code and wondering why it doesn't work. Then you can add the code to write out the message. Any string passed into the html method of the specified element will render in that element. So the string can be any html/text. The control is identified with the # character followed by the id of the element you are targeting. The div element has id="uxMessage" so that is the element we are targeting in the script.

Where Does The Script Code Go

The section is called Scripts (highlighted in yellow as it is C# code). And the section is found in the _Layout page at the bottom of the <body> section:



```
    @RenderSection("Scripts", required: false)
</body>
</html>
```

So this is where the script we wrote earlier will render before being sent to the client browser.

Capturing Change Events

Next, add this code to just after the paragraph with the div element:

```
<p>
    <select id="uxColors">
        <option value="Red">Red</option>
        <option value="Green">Green</option>
        <option value="Blue">Blue</option>
    </select>
</p>
```

Then add more code to the jQuery doc.ready section:

```
//manipulating CSS
$("#uxColors").change(function () {
    var color = $("#uxColors").val();
    alert(color);
});
```

When you run the code and select a color from the drop down, you should see the name of the color selected display in the alert.

Add the following style element at the top of the file:

```
<style>
    .red {
        color: red;
    }

    .blue {
        color: blue;
    }

    .green {
        color: green;
    }
</style>
```

Then replace the alert(color); line of code in the change function with this:

```
$("#uxMessage").removeClass();
$("#uxMessage").addClass(color);
```

Now when you run the code you should see the Hello World text change color to what you selected. We need to remove the previous class so we can add the new class to the uxMessage Div section.

Manipulating HTML

Let's add a button to the form and code the click event. We will append text to the div element:

```
<p>
    <button id="uxAddText" class="btn btn-primary">Add Text</button>
</p>
```

Next, add the click handler in the document ready function:

```
$("#uxAddText").click(function () {
    $("#uxMessage").append("<br/>Can you hear me now?");
});
```

Run the app and click the button multiple times to see the result.

Creating JavaScript Files

We have been writing JavaScript contained between open and closing <script> tags. This can get out of hand as more and more JavaScript is added to the file. To clean up our code, we have the option to move our JavaScript code to an external file and reference the file from our code with a <script src="filename"></script> line of code at the bottom of the file.

To set this section up, type the following code after the add text button paragraph:

```
<p>
    <div id="uxDynamicDisplay"></div>
</p>
<p>
    <button id="uxDisplayData" class="btn btn-primary"
            onclick="displayDynamicData">Display Data</button>
</p>
```

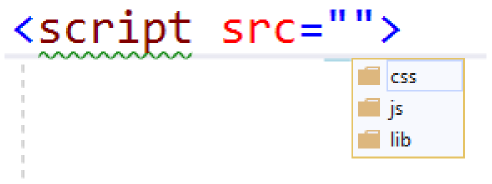
Next we need to add the JavaScript code. We will add a new script tag but after the document ready script . We are coding the “displayDynamicData” function that the onClick event of the button is wired to in the new script tag we just added. So it looks like the following:

```
<script>
    function displayDynamicData() {
        $("#uxDynamicDisplay").html("This is dynamic data.");
    }
</script>
```

So there are now two script tags in the @section Scripts code block. One containing the document.ready cod block and one with the displayDynamicData function. Run the app and click the display data button. This confirms the code works if you see the data string display above the button you clicked.

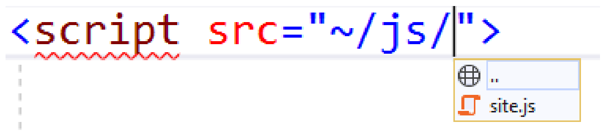
Now, let’s move this function out of this file and into its own file.

1. Copy and paste just the function code (without the script tags) to a file called Site.js found in the js folder under wwwroot in the project in solution explorer. Add a javascript file with this name if it does not exist (but it should in an MVC project).
2. Delete the function code from inside this script tag.
3. Reference this javascript file from your file you have been working in. To do this we need to add a src property and assign the file location of Site.js. In the opening script tag, put a space after the word *Script* and type src= to see the intellisense display:



```
<script src="">
```

4. Select js and the intellisense continues so you see the Site.js file. Select it.



```
<script src="~/js/">
```

5. The final script tag displays as follows. Run the app and you should see the same behavior as before when the function code was still in this file and not its own.

```
<script src="~/js/Site.js"></script>
```

Using Ajax

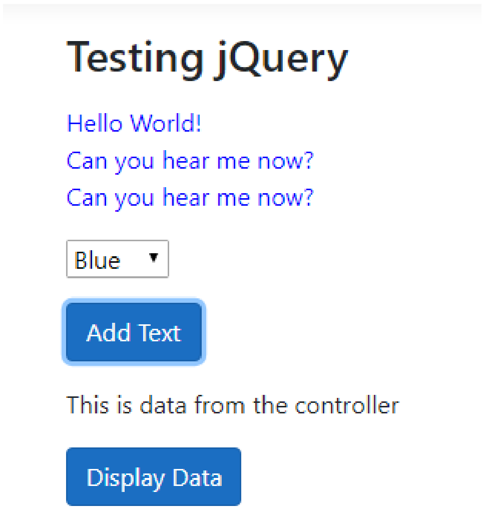
The use of Ajax is essential for executing asynchronous message calls to the server. In this way, the client browser can call a controller method to return json result, a partial view or a view component. Add an action method to the Home controller call Display and the method returns Content() instead of View(). The code looks like this:

```
public IActionResult Display()
{
    return Content("This is data from the controller");
}
```

Add the Ajax code to the Site.js file in the existing function and comment out the old code:

```
function displayDynamicData() {
    //$("#uxDynamicDisplay").html("This is dynamic data.");
    $.ajax({
        method: 'GET',
        url: '/Home/Display'
    }).done(function (result, statusText, xhdr) {
        $("#uxDynamicDisplay").html(result);
    });
}
```

Result view:


Testing jQuery
Hello World!
Can you hear me now?
Can you hear me now?
Blue
Add Text
This is data from the controller
Display Data