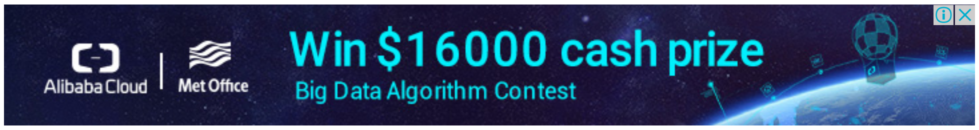


Handle Ajax Requests in ASP.NET Core Razor Pages

October 30, 2017 (http://www.talkingdotnet.com/handle-ajax-requests-in-asp-net-core-razor-pages/) Talking Dotnet (http://www.talkingdotnet.com/author/talkingdotnet/) ASP.NET Core (http://www.talkingdotnet.com/asp-net-core-1-0/)

Razor Pages (http://www.talkingdotnet.com/tag/razor-pages/) are a new feature of ASP.NET Core that makes coding page-focused scenarios easier and more productive. Razor pages use handler methods to deal with the incoming HTTP request (GET/POST/PUT/Delete). These are similar to Action methods of ASP.NET MVC or WEB API. Razor Pages follow particular naming convention and that is also true for Handler methods. They also follow particular naming conventions and prefixed with "On": and HTTP Verb like `OnGet()`, `OnPost()` etc. The handler methods also have asynchronous version: `OnGetAsync()`, `OnPostAsync()` etc. Calling these handler methods from jQuery Ajax is tricky. This post talks how to handle Ajax requests in ASP.NET Core Razor Pages.

Handle Ajax Requests in ASP.NET Core Razor Pages

Before we look into handling Ajax requests in ASP.NET Core Razor Pages, it's important to understand how handler methods work. BTW, if you are new to ASP.NET Core Razor Pages, following articles will help.

- Getting started with Razor Pages in ASP.NET Core (https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/razor-pages-start)
- Introduction to Razor Pages in ASP.NET Core (https://docs.microsoft.com/en-us/aspnet/core/mvc/razor-pages/?tabs=visual-studio)
- Introduction to ASP.NET Core Razor Pages (https://www.youtube.com/watch?v=yyBijyCI5Sk)

As mentioned earlier, the handler method follows a pattern. They are prefixed with "On" and the name of HTTP verb like,

- OnGet
- OnPost
- OnPut
- OnGetAsync
- OnPostAsync
- OnPutAsync

The `OnGet` method gets called on the page load and `onPost` gets called when the form gets submitted. The default template for ASP.NET Core 2.0 web application comes with a couple of razor pages. When you open About.cs.html file, you should see the following code.

```
public void OnGet()
{
    Message = "Your application description page.";
}
```

The `onGet()` gets called when the request comes for the About page. Besides these default Handlers, we can also specify custom names. The custom name must come after the followed naming convention like,

- OnGetCountries()
- OnPostUserMaster()
- OnPostUserDetails()

In case of multiple POST handlers on the same page, how do you call them? You need to use `asp-page-handler` Tag Helper and assign the handler name. Like,

```
<form asp-page-handler="usermaster" method="post">
    <input type="submit" id="btnSubmit" value="Save Master" />
</form>
<form asp-page-handler="userdetail" method="post">
    <input type="submit" id="btnSubmit" value="Save Details" />
</form>
```

Or, we can achieve the same thing with one form, and two submit inputs inside of that form:

```
<form method="post">
   <input type="submit" asp-page-handler="usermaster" value="Save Master" />
   <input type="submit" asp-page-handler="userdetail" value="Save Details" />
</form>
```

That's enough for quick understanding. Read Razor Pages – Understanding Handler Methods (https://www.mikesdotnetting.com/article/308/razor-pages-understanding-handler-methods) for detailed information about handler methods.

Making Ajax Requests in ASP.NET Core Razor Pages

Now, let talk about calling the handler methods from jQuery Ajax. You must be thinking what is so different. Here is a `GET` handler method defined in "Demo" razor page.

```
public JsonResult OnGetList()
{
    List<string> lstString = new List<string>
    {
        "Val 1",
        "Val 2",
        "Val 3"
    };
    return new JsonResult(lstString);
}
```

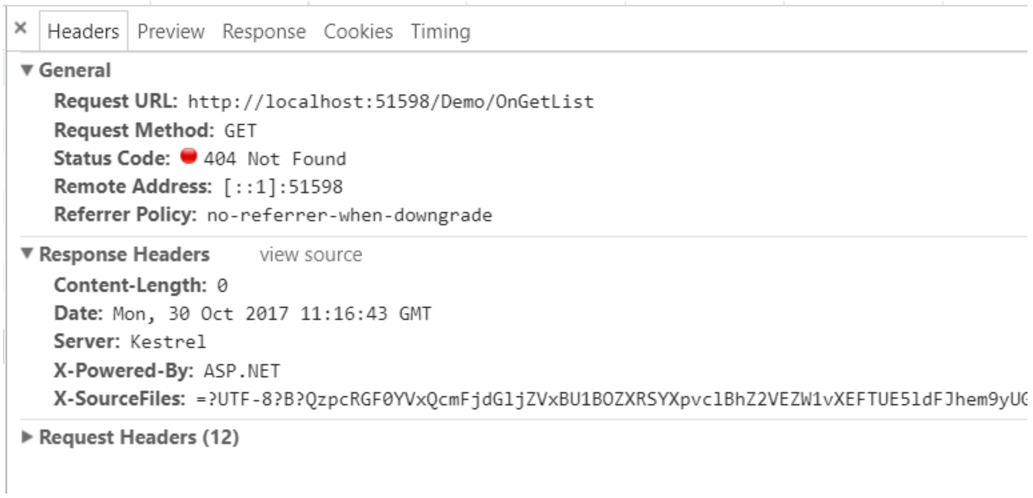
The HTML contains only div element without a form tag.

```
<div id="dvItems" style="font-size:24px;">
</div>
```

The following jQuery code will call the `OnGetList` handler method available in Demo razor page and populate the list.

```
$.ajax({
    type: "GET",
    url: "/Demo/OnGetList",
    contentType: "application/json",
    dataType: "json",
    success: function (response) {
        var dvItems = $("#dvItems");
        dvItems.empty();
        $.each(response, function (i, item) {
            var $tr = $('<li>').append(item).appendTo(dvItems);
        });
    },
    failure: function (response) {
        alert(response);
    }
});
```

But, you will be surprised to see 404 not found error. See below screenshot.



▼ General
Request URL: http://localhost:51598/Demo/OnGetList
Request Method: GET
Status Code: ● 404 Not Found
Remote Address: [::1]:51598
Referrer Policy: no-referrer-when-downgrade

▼ Response Headers view source
Content-Length: 0
Date: Mon, 30 Oct 2017 11:16:43 GMT
Server: Kestrel
X-Powered-By: ASP.NET
X-SourceFiles: =?UTF-8?B?QzpcRGF0YVxQcmFjdGljZVxBU1BOZXRSYXpvclBhZ2VEZW1vXEFTUE51dFJhem9yUG

▶ Request Headers (12)

(http://www.talkingdotnet.com/wp-content/uploads/2017/10/Handle-Ajax-requests-in-ASP.NET-Core-Razor-Pages.png)To understand the reason for getting 404 error, we need to see how the handler methods are rendered. In one the earlier code sample, we created 2 forms and called 2 handler methods. Following is the output of generated HTML on the client side. For now, ignore the `__RequestVerificationToken` and look at the value of the `action` attribute of form tag.

```
<form method="post" action="/Demo?handler=usermaster">
    <button class="btn btn-default">Save Master</button>
<input name="__RequestVerificationToken" type="hidden" value="CfDJ8KW5cuB058RCnNyZSLI7AUjUAtTwe54jQ4Z9Goyn3WKPcpVFYSFUM5J-JDFC3E

<form method="post" action="/Demo?handler=userdetail">
    <button class="btn btn-default">Save Details</button>
<input name="__RequestVerificationToken" type="hidden" value="CfDJ8KW5cuB058RCnNyZSLI7AUjUAtTwe54jQ4Z9Goyn3WKPcpVFYSFUM5J-JDFC3E
```

The thing to notice here is, the name of the handler is added to the form's action as a query string parameter. We need to use similar URL pattern while making the jQuery Ajax call. Like,

```
$.ajax({
    type: "GET",
    url: "/Demo?handler=List",
    contentType: "application/json",
    dataType: "json",
    success: function (response) {
        var dvItems= $("#dvItems");
        dvItems.empty();
        $.each(response, function (i, item) {
            var $tr = $('<li>').append(item).appendTo(dvItems);
        });
    },
    failure: function (response) {
        alert(response);
    }
});
```

Now, this should work. You can use the same approach for `POST` requests as well. Here is a `POST` handler method defined in "Demo" razor page.

```
public ActionResult OnPostSend()
{
    List<string> lstString = new List<string>
    {
        "Val 1",
        "Val 2",
        "Val 3"
    };
    return new JsonResult(lstString);
}
```

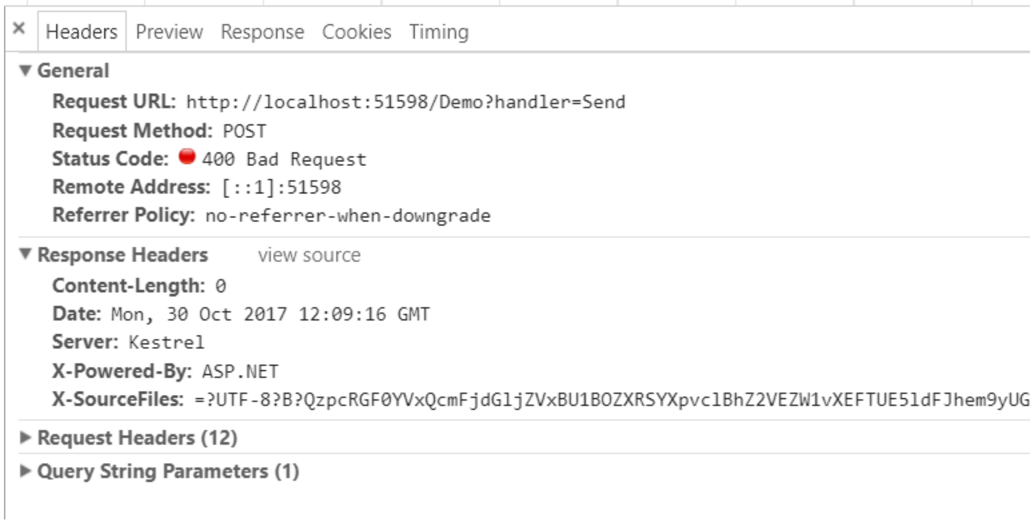
The HTML contains only div element without a form tag.

```
<div id="dvPostItems" style="font-size:24px;">
</div>
```

Here is jQuery Ajax call for the POST method.

```
$.ajax({
    type: "POST",
    url: "/Demo?handler=Send",
    contentType: "application/json; charset=utf-8",
    dataType: "json",
    success: function (response) {
        var dvItems = $("#dvPostItems");
        dvItems.empty();
        $.each(response, function (i, item) {
            var $tr = $('<li>').append(item).appendTo(dvItems);
        });
    },
    failure: function (response) {
        alert(response);
    }
});
```

Bang!!! This gives you a 400 bad request error.



▼ General

Request URL: http://localhost:51598/Demo?handler=Send
Request Method: POST
Status Code: ● 400 Bad Request
Remote Address: [::1]:51598
Referrer Policy: no-referrer-when-downgrade

▼ Response Headers view source

Content-Length: 0
Date: Mon, 30 Oct 2017 12:09:16 GMT
Server: Kestrel
X-Powered-By: ASP.NET
X-SourceFiles: =?UTF-8?B?QzpcRGF0YVxQcmFjdGljZVxBU1BOZXRSYXpvclBhZ2VEZW1vXEFTUE5ldFJhem9yUG

▶ Request Headers (12)

▶ Query String Parameters (1)

(http://www.talkingdotnet.com/wp-content/uploads/2017/10/Handle-Ajax-requests-in-ASP.NET-Core-Razor-Pages_1.png)You must be wondering now, what's wrong with the above code. Well, there is nothing wrong with the above code. The reason is,

Razor Pages are designed to be automatically protected from cross-site request forgery (https://en.wikipedia.org/wiki/Cross-site_request_forgery) (CSRF/XSRF) attacks. You don't have to write any additional code. Antiforgery token generation and validation is automatically included in Razor Pages. Here the request fails, there is no AntiForgeryToken present on the page.

There are 2 ways to add the AntiForgeryToken.

- In ASP.NET Core MVC 2.0 the FormTagHelper injects anti-forgery tokens for HTML form elements. For example, the following markup in a Razor file will automatically generate anti-forgery tokens:

```
<form method="post">
  <!-- form markup -->
</form>
```

- Add explicitly using `@Html.AntiForgeryToken()`

To add AntiForgeryToken, we can use any of the approaches. Both the approaches add an input type hidden with name `__RequestVerificationToken`. The Ajax request should send the anti-forgery token in request header to the server. So, the modified Ajax request looks like,

```
$.ajax({
    type: "POST",
    url: "/Demo?handler=Send",
    beforeSend: function (xhr) {
        xhr.setRequestHeader("XSRF-TOKEN",
            $('input:hidden[name="__RequestVerificationToken"]').val());
    },
    contentType: "application/json; charset=utf-8",
    dataType: "json",
    success: function (response) {
        var dvItems = $("#dvPostItems");
        dvItems.empty();
        $.each(response, function (i, item) {
            var $tr = $('<li>').append(item).appendTo(dvItems);
        });
    },
    failure: function (response) {
        alert(response);
    }
});
```

Since the script sends the token in a header called `X-CSRF-TOKEN`, configure the antiforgery service to look for the `X-CSRF-TOKEN` header:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddAntiforgery(o => o.HeaderName = "XSRF-TOKEN");
}
```

Now, when the Post request is executed, it should work as expected. Below is another working example of `POST` request, sending data from client to server.

```
$('#btnPost').on('click', function () {
    var item1 = $('#txtItem1').val();
    var item2 = $('#txtItem2').val();
    var item3 = $('#txtItem3').val();
    $.ajax({
        type: "POST",
        url: "/Demo?handler=Send",
        beforeSend: function (xhr) {
            xhr.setRequestHeader("XSRF-TOKEN",
                $('input:hidden[name="__RequestVerificationToken"]').val());
        },
        data: JSON.stringify({
            Item1: item1,
            Item2: item2,
            Item3: item3
        }),
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        success: function (response) {
            var dvItems = $("#dvPostItems");
            dvItems.empty();
            $.each(response, function (i, item) {
                var $tr = $('<li>').append(item).appendTo(dvItems);
            });
        },
        failure: function (response) {
            alert(response);
        }
    });
})
```

Here, we can’t access the passed data via query string as the querystring contains “?handler=Send” value. Hence, to access the passed value on the server, read the Request object’s body. Like,

```
public ActionResult OnPostSend()
{
    string sPostValue1 = "";
    string sPostValue2 = "";
    string sPostValue3 = "";
    {
        MemoryStream stream = new MemoryStream();
        Request.Body.CopyTo(stream);
        stream.Position = 0;
        using (StreamReader reader = new StreamReader(stream))
        {
            string requestBody = reader.ReadToEnd();
            if(requestBody.Length > 0)
            {
                var obj = JsonConvert.DeserializeObject<PostData>(requestBody);
                if(obj != null)
                {
                    sPostValue1 = obj.Item1;
                    sPostValue2 = obj.Item2;
                    sPostValue3 = obj.Item3;
                }
            }
        }
    }
    List<string> lstString = new List<string>
    {
        sPostValue1,
        sPostValue2,
        sPostValue3
    };
    return new JsonResult(lstString);
}
```

The above code also deserialize the JSON data into `PostData` class object and below is the definition of `PostData` class.

```
public class PostData
{
    public string Item1 { get; set; }
    public string Item2 { get; set; }
    public string Item3 { get; set; }
}
```

You can find the source code in the Github (https://github.com/talkingdotnet/ASPNETCoreRazorPageDemoApp/tree/ASPNETCoreRazorAjax).

Summary

The post talks about ASP.NET Core Razer Pages handler methods naming conventions and creating named handler method. Razor Pages are designed to be protected from (CSRF/XSRF) attacks. Hence, Antiforgery token generation and validation are automatically included in Razor Pages. The post also provide solutions for making Ajax requests for Razor pages handler methods.

Thank you for reading. Keep visiting this blog and share this in your network. Please put your thoughts and feedback in the comments section.

PS: If you found this content valuable and want to return the favour, then Buy me a coffee (https://www.buymeacoffee.com/talkingdotnet)

Spread this:

(http://www.talkingdotnet.com/handle-ajax-requests-in-asp-net-core-razor-pages/?share=email&nb=1)

(http://www.talkingdotnet.com/handle-ajax-requests-in-asp-net-core-razor-pages/#print)

(http://www.talkingdotnet.com/handle-ajax-requests-in-asp-net-core-razor-pages/?share=facebook&nb=1)

(http://www.talkingdotnet.com/handle-ajax-requests-in-asp-net-core-razor-pages/?share=twitter&nb=1)

(http://www.talkingdotnet.com/handle-ajax-requests-in-asp-net-core-razor-pages/?share=google-plus-1&nb=1)

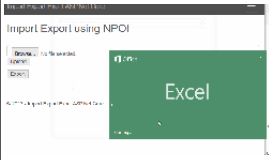
(http://www.talkingdotnet.com/handle-ajax-requests-in-asp-net-core-razor-pages/?share=pinterest&nb=1)

(http://www.talkingdotnet.com/handle-ajax-requests-in-asp-net-core-razor-pages/?share=linkedin&nb=1)

(http://www.talkingdotnet.com/handle-ajax-requests-in-asp-net-core-razor-pages/?share=reddit&nb=1)

(http://www.talkingdotnet.com/handle-ajax-requests-in-asp-net-core-razor-pages/?share=tumblr&nb=1)

Related

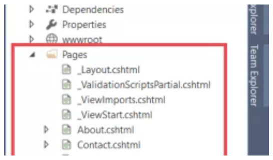


(http://www.talkingdotnet.com/import-export-excel-asp-net-core-2-razor-pages/)

Import and Export excel in ASP.NET Core 2.0 Razor Pages (http://www.talkingdotnet.com/import-export-excel-asp-net-core-2-razor-pages/)
In "ASP.NET Core"

(http://www.talkingdotnet.com/uploading-multiple-files-asp-net-core-razor-pages/)

Uploading Multiple Files in ASP.NET Core Razor Pages (http://www.talkingdotnet.com/uploading-multiple-files-asp-net-core-razor-pages/)
In "ASP.NET Core"



(http://www.talkingdotnet.com/change-asp-net-core-razor-pages-default-directory-name/)

Change ASP.NET Core Razor Pages default directory name (http://www.talkingdotnet.com/change-asp-net-core-razor-pages-default-directory-name/)
In "ASP.NET Core"

ASP.NET Core (http://www.talkingdotnet.com/tag/asp-net-core-1-0/) ASP.NET Core 2.0 (http://www.talkingdotnet.com/tag/asp-net-core-2-0/)
Razor Pages (http://www.talkingdotnet.com/tag/razor-pages/)

4 thoughts on "Handle Ajax Requests in ASP.NET Core Razor Pages"

Pingback: Handle Ajax Requests in ASP.NET Core Razor Pages – Site Design (http://blog.sitedesign.gr/handle-ajax-requests-in-asp-net-core-razor-pages/)
Pingback: Handle Ajax Requests in ASP.NET Core Razor Pages – MyIT101 (http://www.myit101.com/handle-ajax-requests-in-asp-net-core-razor-pages/)

YOGI
November 2, 2017 at 1:20 PM (http://www.talkingdotnet.com/handle-ajax-requests-in-asp-net-core-razor-pages/#comment-1599)

Thank you for this excellent AJAX tutorial.

REPLY (HTTP://WWW.TALKINGDOTNET.COM/HANDLE-AJAX-REQUESTS-IN-ASP-NET-CORE-RAZOR-PAGES/?REPLYTOCOM=1599#RESPOND)

Pingback: Uploading Multiple Files in ASP.NET Core Razor Pages (http://www.talkingdotnet.com/uploading-multiple-files-asp-net-core-razor-pages/)

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Please enter an answer in digits:

nine − 8 =

POST COMMENT

Notify me of follow-up comments by email.

Notify me of new posts by email.

Change ASP.NET Core Razor Pages default directory name (http://www.talkingdotnet.com/change-asp-net-core-razor-pages-default-directory-name/)

How to Get Client IP Address in ASP.NET Core 2.0 Razor Pages (http://www.talkingdotnet.com/get-client-ip-address-asp-net-core-2-0-razor-pages/)

Buy me a coffee (https://www.buymeacoffee.com/talkingdotnet)

FOLLOW US

(http://www.talkingdotnet.com/feed/)

(https://twitter.com/talkingdotnet)

(https://www.facebook.com/talkingdotnet)

(https://plus.google.com/+TalkingdotNet/)

(https://www.pinterest.com/talkingdotnet/)

GET UPDATES IN YOUR INBOX

Join us to get valuable tips, articles, and resources straight to your inbox.

Email Address

SUBSCRIBE

Search...

POPULAR POSTS

Bind Select DropDown List in Angular 2 (http://www.talkingdotnet.com/bind-select-dropdown-list-in-angular-js-2/)

How to create an Angular 4 app with Visual Studio 2017 (http://www.talkingdotnet.com/how-to-create-an-angular-4-app-with-visual-studio-2017/)

Tools for bundling and minification in ASP.NET Core (http://www.talkingdotnet.com/tools-for-bundling-minification-aspnet-core/)

Don't use hidden attribute with Angular 2 (http://www.talkingdotnet.com/dont-use-hidden-attribute-angularjs-2/)