Need

* When 2 applications/bodies wants to communicate with each other on the internet.

Solution

1. Remoting : .Net to .Net
2. Web Services :

* SOAP Based.
* Doesn’t have WS-Security, WS-Messaging

1. Web Service Enhancement

* SOAP Based
* It contains Everything needed to have interoperability between 2 heterogenous application.

1. WCF

* Service Oriented applications
* Multiple data format support
* Multiple protocol support

1. Web API

* JSON format
* HTTP Protocol

REST Service

* Standard of interactive applications that use web services.
* Provides Web resources in a textual representation and allow them to read and modified with a stateless protocol and a predefined set of operations.

Web APIS

* Accept Http Requests and Generate Responses that contain Data
* Provides Access to an Application Data
* Generally use to provide rich client side applications with Data.
* Combination of URL and HTTP Method describes an Operation that is handled by an action method defined by an ASP.Net Controller.
* Most REStful Services format the response data using JSON format. Its very popular format as its easily consumed by JS clients.

Http Verbs

1. GET : Retrieve on or more data objects
2. POST : Create new Object
3. PUT : Update an Existing Object
4. DELETE : Delete an object.

ControllerBase

* API Controllers derived from the controllerbase class, which provides access to features provided by MVC framework.
* It gives access to

1. HttpContext
2. ModelState
3. Request
4. Response
5. RouteData
6. User.

Controller Attributes

* URL of the Controller is specified by the Route Attribute

[Route(“api/[controller]”)]

* Each Action is decorated like

[HttpGet],[HttpPost] etc

* [HttpGet(“{id}”)]: GET Request for the URL pattern api/products/{id}

Action()

Rule of Thumb

1. GET Requests should be used for all read only information retrieval.
2. POST requests used for any operation that changes the application state.

ASP.Net Core Offer following Return Type

1. Specific Type :

* Returns string, List<string>
* When Multiple return types are possible you can use IActionResult or ActionResult.

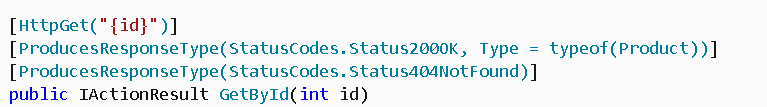
1. IActionResult

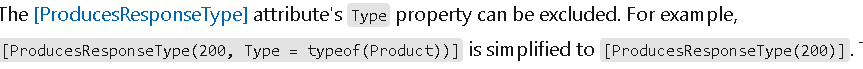
* When Multiple ActionResult Return types are possible in an action.
* ActionResult represent various Http status codes.
* ActionResult can return BadRequest(400), NotFoundResult(404) and OkObjectResult(200)

ActionResult<T> Type

* ProducesResponseType attribute Type property can be excluded. The Actions expected return type is instead inferred from ActionResult<T>

Difference between IActionResult and ActionResult





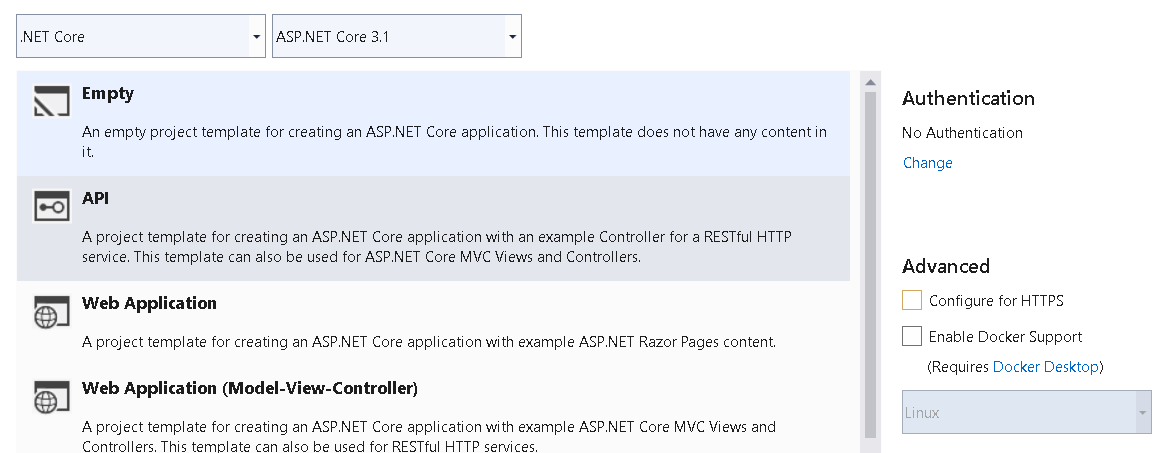
Binding Source

* Defines the location at which an action parameters value is found.

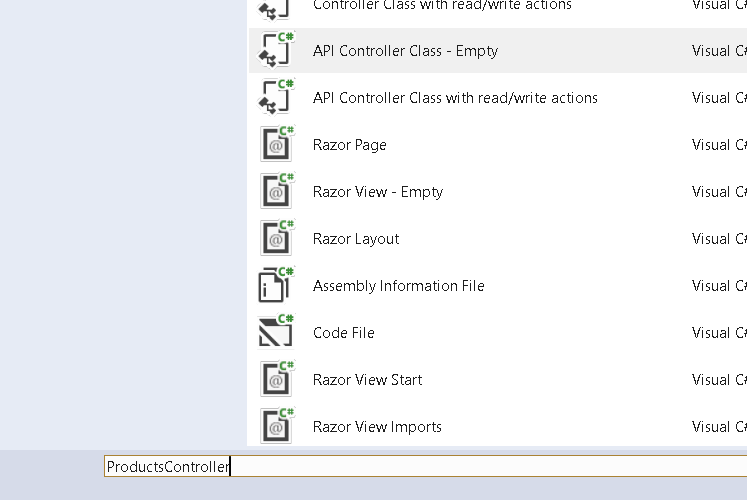
1. FromBody : RequestBody
2. FromHeader : Request Header
3. FromQuery : Request Query String Parameter
4. FromRoute : Route Data from the current Request from the route template.

Demo : Create WebAPI Project and Add Controllers

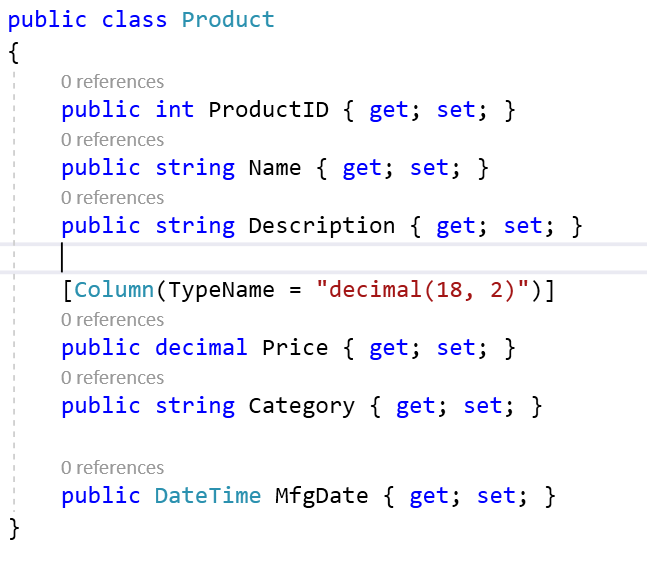
1. Add New WebAPI Project



1. Add ProductsController



1. Add Product Model



1. Add actions in ProductsController



1. Pass following URLS

<http://localhost:5000/api/products>

<http://localhost:5000/api/products/1>

1. Check the Output

Using Dependency Injection in Controllers

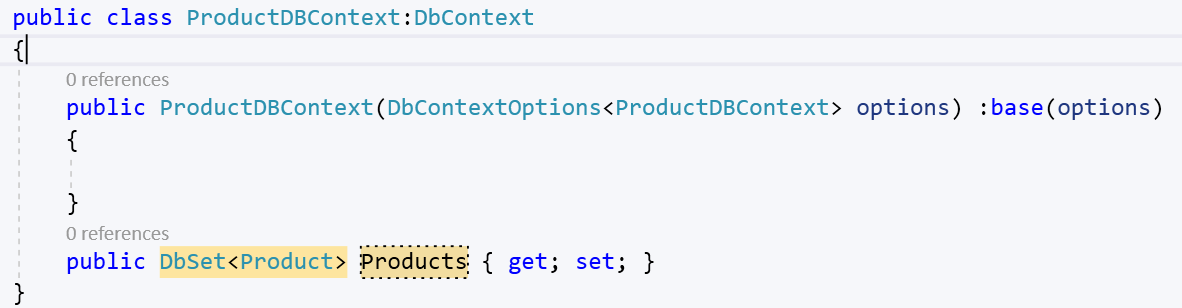
1. Add ConnectionString

"ConnectionStrings": {

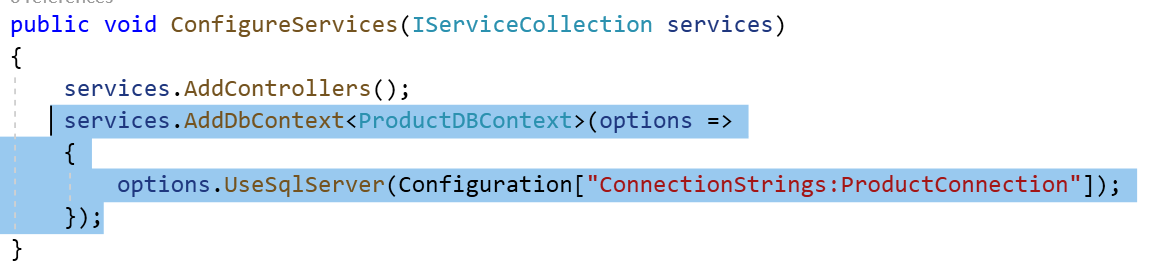
"ProductConnection": "Data Source=SWAPNIL-PC\\SQLEXPRESS;Initial Catalog=BOATestAgain;Integrated Security=true"

}

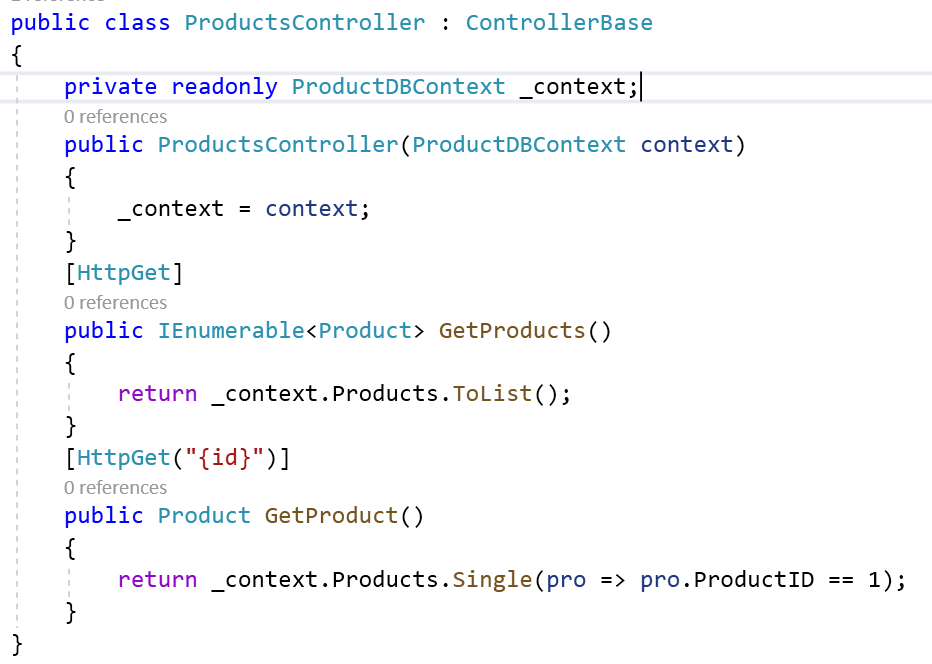
1. Create ProductDBContext class



1. Activate Service for the same



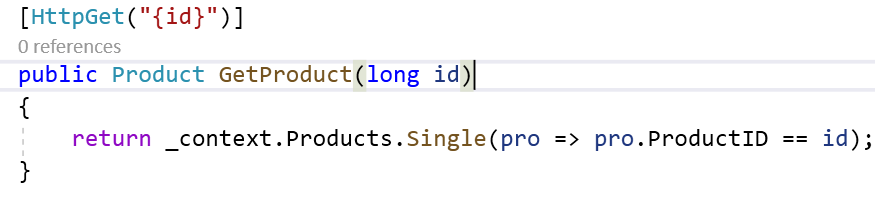
1. Change the ProductsController to Connect DB



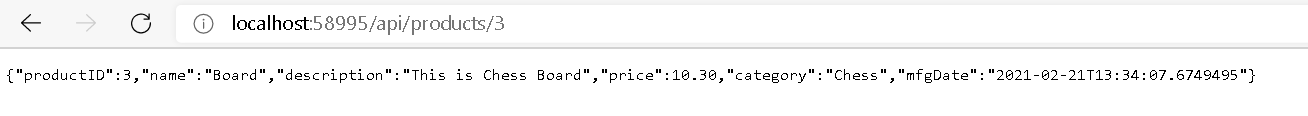
1. Check the Output

Using Model Binding to Access Route Data

1. MVC framework uses request URL to find values for the action method parameters a process known as model binding.
2. Change the Method Parameter



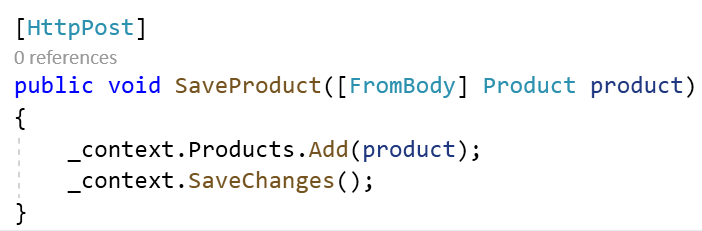
1. Check the Output



Model binding from the Request Body

* Used on the data in the request body which allows clients to send data that is easily received by an action method.
* FromBody attribute is applied to the action parameter and it specifies that the value for this parameter should be obtained by parsing request body.

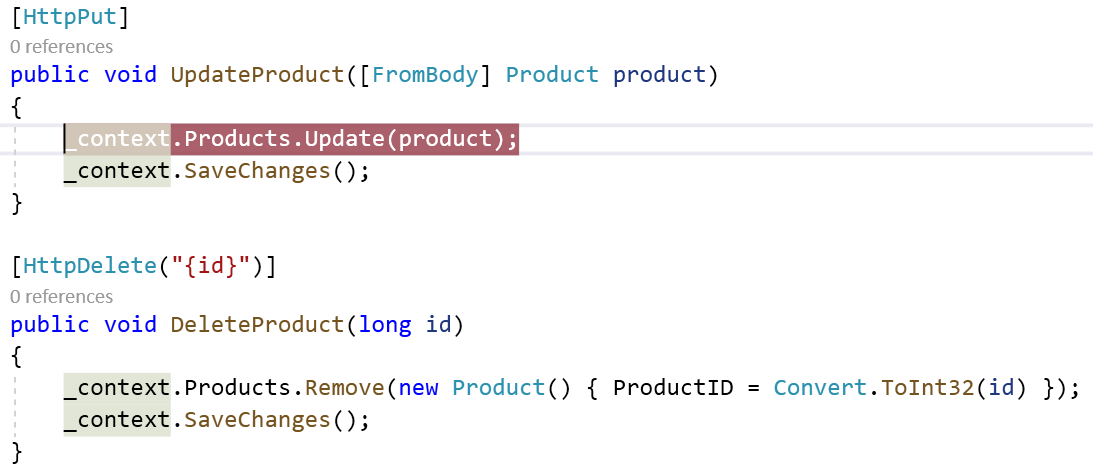
1. Add Action to Post the data



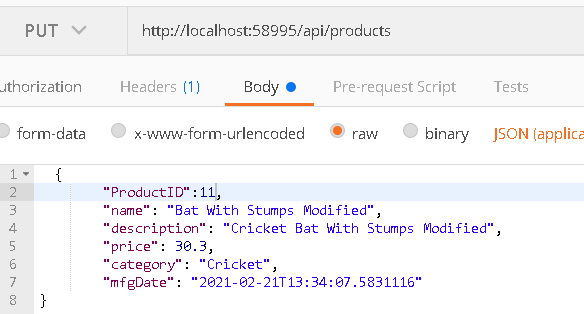
1. Use Postman to Pass the data



1. Add Action Method for Delete and Update



1. Check the Output



Using Asynchronous Actions

* Asp.Net Core Platform Processed each request by assigning a thread from the pool.
* Number Of request that can be processed is limited to size of the pool. In such situation ASP.NET Core request thread might run out of threads we have to process the request.
* This problem can be addressed by defining asynchronous actions which allows thread to process other requests when they would otherwise blocked.

1. Modify Methods of ProductsController to Async actions

[HttpGet]

public IAsyncEnumerable<Product> GetProducts()

{

return \_context.Products;

}

[HttpGet("{id}")]

public async Task<Product> GetProduct(long id)

{

return await \_context.Products.FindAsync(Convert.ToInt32(id));

}

[HttpPost]

public async Task SaveProduct([FromBody] Product product)

{

await \_context.Products.AddAsync(product);

await \_context.SaveChangesAsync();

}

[HttpPut]

public async Task UpdateProduct([FromBody] Product product)

{

\_context.Products.Update(product);

await \_context.SaveChangesAsync();

}

[HttpDelete("{id}")]

public async Task DeleteProduct(long id)

{

\_context.Products.Remove(new Product() { ProductID = Convert.ToInt32(id) });

await \_context.SaveChangesAsync();

}

1. Check the Output

Preventing Over Binding

* The Product class needs a ProductID property but model binding process doesn’t understand the significance of this property and adds any values that client provides . this is called as Over Binding.
* Safest way to prevent over-binding is to create data model classes that are used only for receiving data through model binding process.

1. Add a class in Models folder

public class ProductBindingTarget

{

public string Name { get; set; }

public string Description { get; set; }

[Column(TypeName = "decimal(18, 2)")]

public decimal Price { get; set; }

public string Category { get; set; }

public DateTime MfgDate

{

get; set;

}

public Product ToProduct() => new Product()

{

Name = this.Name,

Price = this.Price,

Category = this.Category,

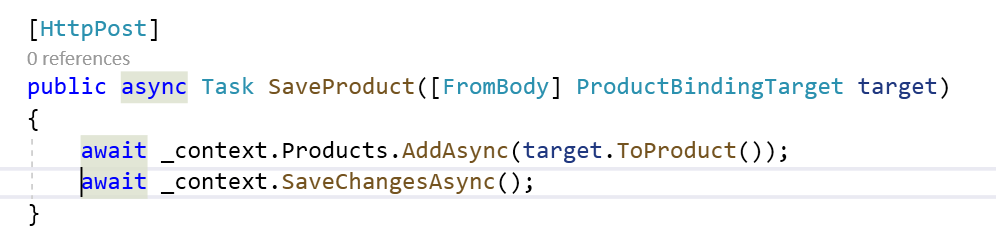
Description = this.Description,

MfgDate = this.MfgDate

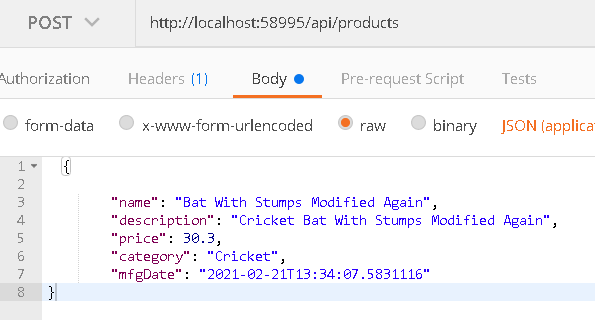
};

}

1. Modify SaveProduct



1. Check the Output



Using Action Results

* MVC Framework sets the status code automatically. But you want always get the result you desire.
* Lets Say GetProduction action method returns null , when the MVC framework returns 204 status code which indicates successful requests that has produced no data. So in this situation we need to send not found.
* Actions Methods can direct MVC framework to send specific response by returning object that implements the IActionResult interface. Which is known as Action Result. This allows action method to specify the type of response that is required.

Following are the ActionResult Methods

1.Ok : Status code : 200 + optional data object in the response body

2. NoContent : Produces 204 NoContent Status code

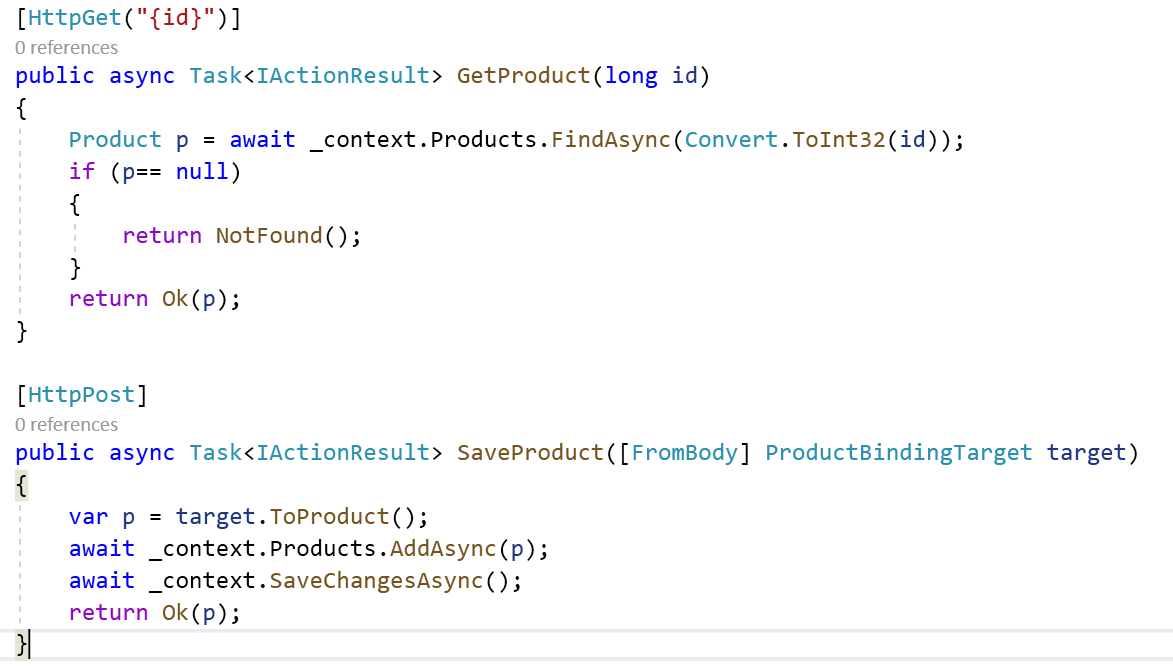
3. BadRequest : 400 Bad Request status code, accepts model state object that describes problem to the client.

4. File : 200Ok + Sends File

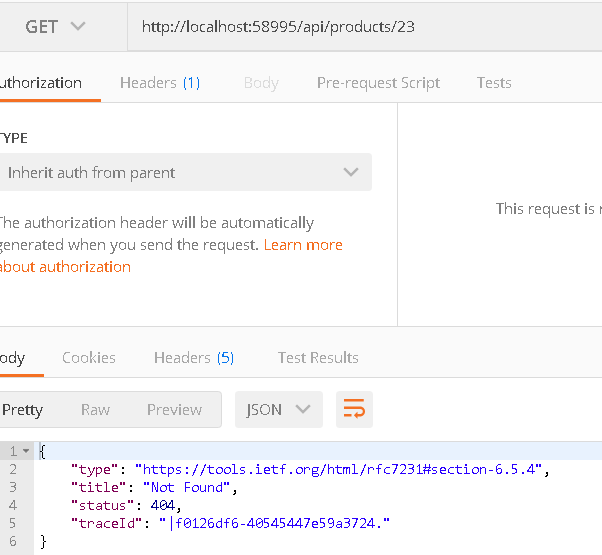
5. NotFound : 404 Not Found status code

Etc.

1. Modify Action Methods

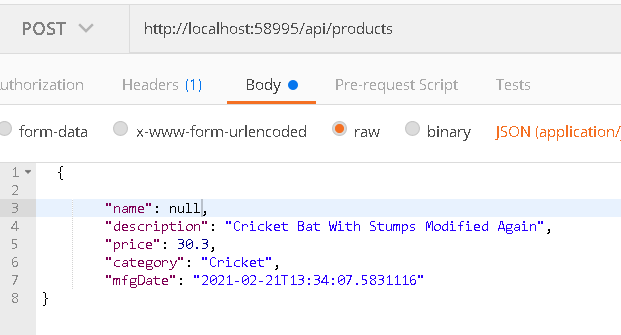


1. Check the Output



Validating Data

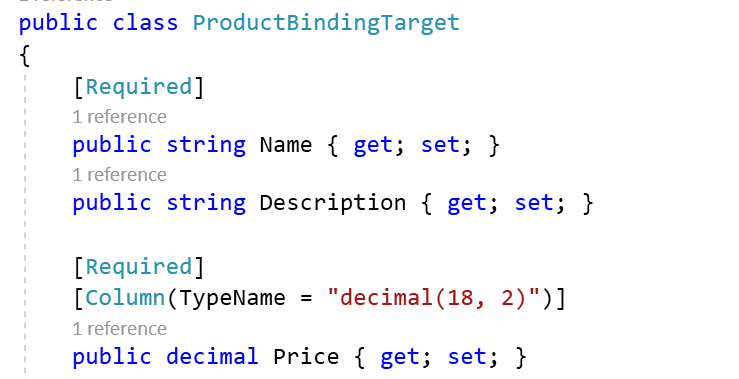
1. Pass null values in the Name attribute



1. Data will be added

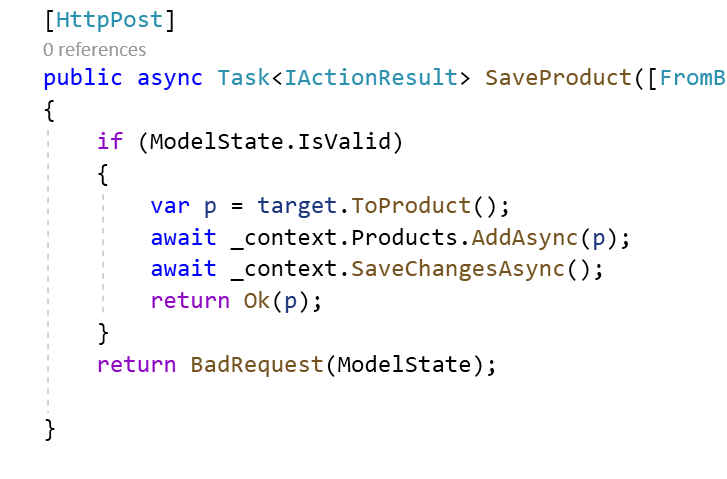


1. Add a model that contains validation code

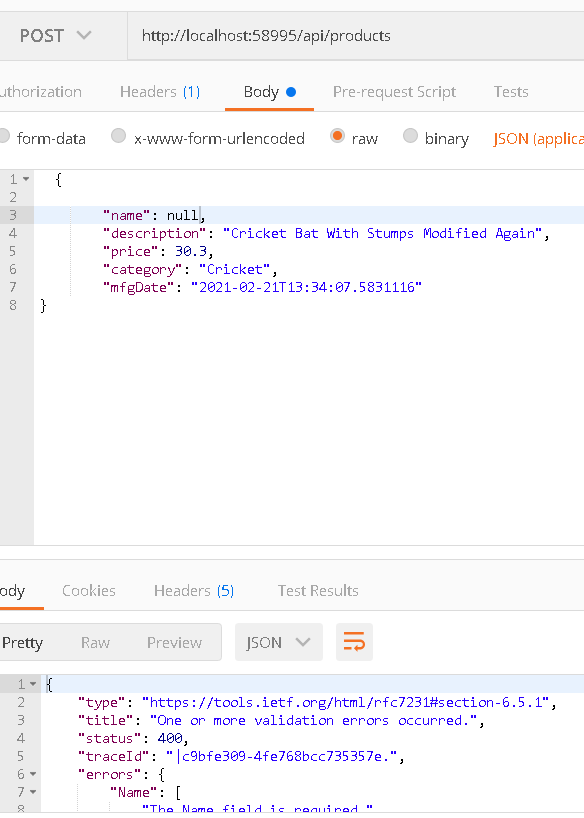


1. Change the Code for SaveProduct.

IsValid property returns true if the model binding process has produced data that meets the validation criteria. If the data received from the client is valid, then the action result from the Ok method is returned. If the data sent by the client fails the validation check, then the IsValid property will be false, and the action result from the BadRequest method is used instead. The BadRequest method accepts the object returned by the ModelState property, which is used to describe the validation errors to the client.

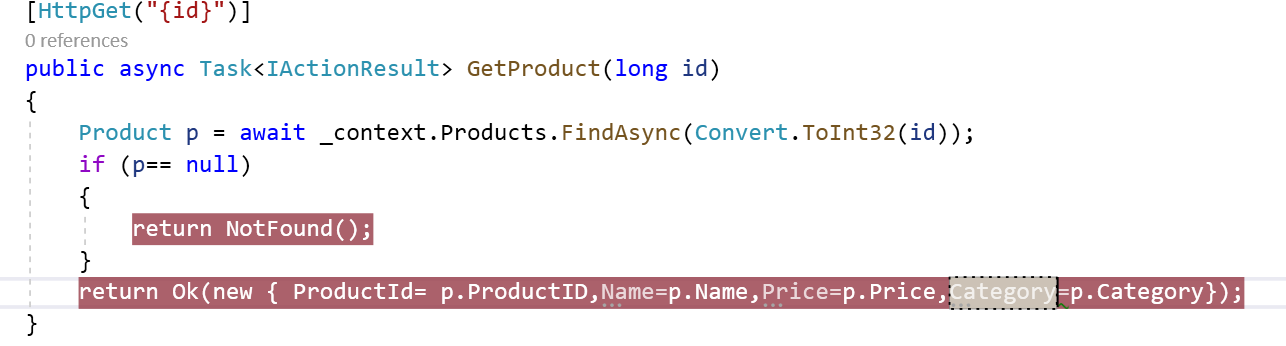


1. Check the Output

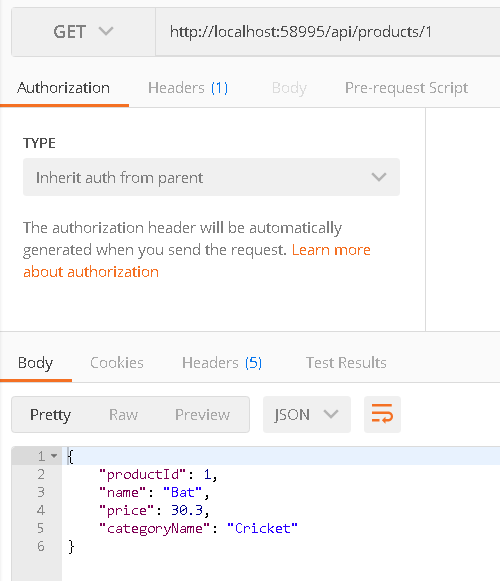


Projecting Selected Properties

1. Return just properties that the client requires. This gives complete control over each response.



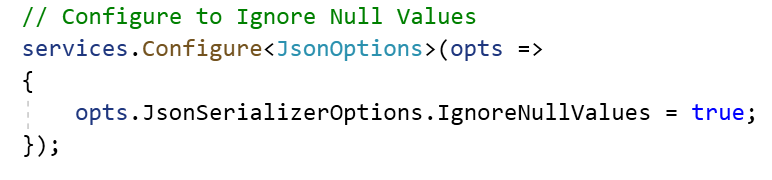
1. Check the Output



Configuring JSON Serializer

* Json Serializer can be configured to omit properties whose values is null when it serializes objects.

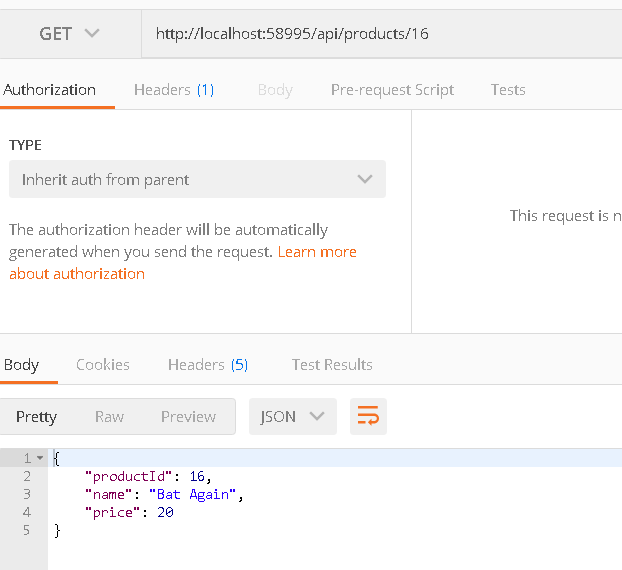
1. Add Configuration options for Omitting null values



1. Insert some null values

Insert into Products values('Bat Again',null,20,null,'1-1-2021')

1. Check the Output



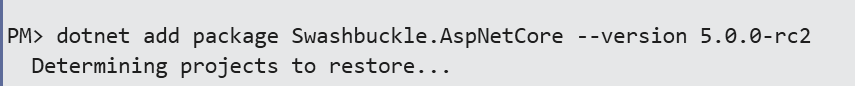
Installing and Configuring Swashbuckle

* Implementation of the OpenAPI specification and will automatically generate a description for the web services .

Demo : How to Enable Swashbuckle package

1. Add Library

dotnet add package Swashbuckle.AspNetCore --version 5.0.0-rc2



1. Include namespace in startup.cs

using Microsoft.OpenApi.Models;

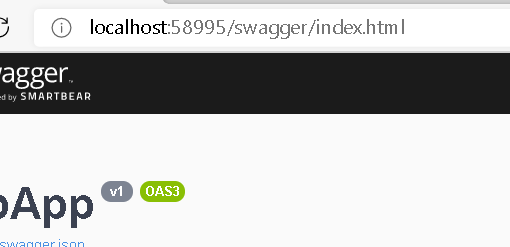
1. Enable Swashbuckle



1. Add Endpoint for the same



1. Go to Following URL to test Swagger.



Content Formatting

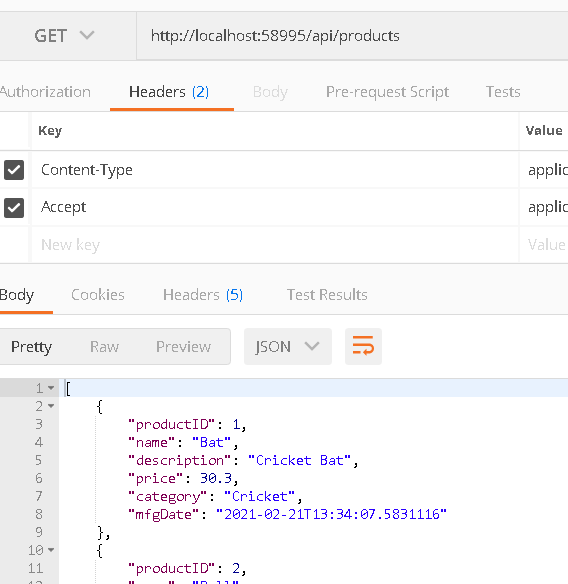
* Web API supports multiple data format that action methods can produce.
* The content format selected for an action result depends on 4 Factors

1. Formats that client will Accept
2. Formats that the application can produce.
3. Content Policy Specified by the action method
4. Type returned by the action method.

Content Negotiation

* Most Clients include an Accept Header in a request which specifies the set of formats that they are willing to receive in the response.

1. You might assume that changing the client format will produce that specific format



1. Check the Output

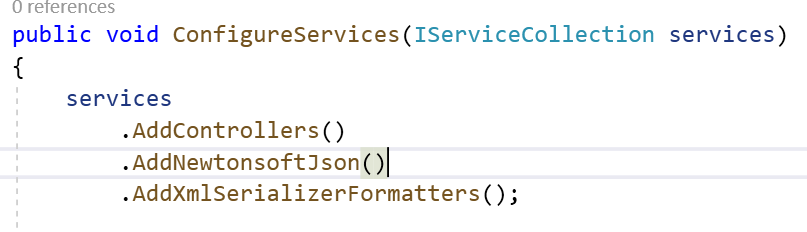


Accept header has no effect on the format,By default MVC framework configured to use JSON. So MVC framework sends JSON format hoping client can process it.

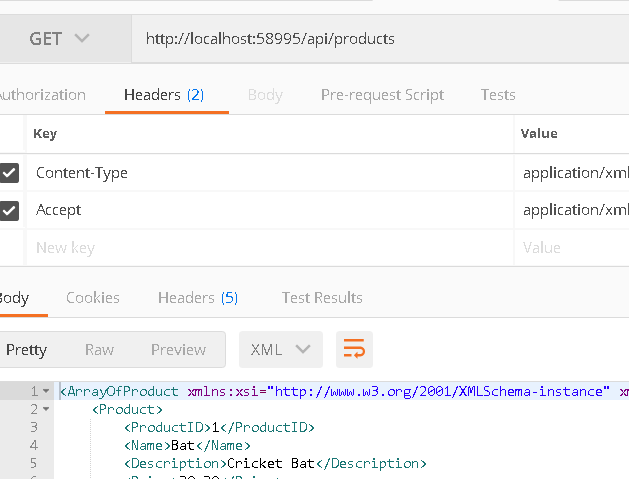
Enabling

* Application must be configured so that it works with some other format.

1. Add XML format support

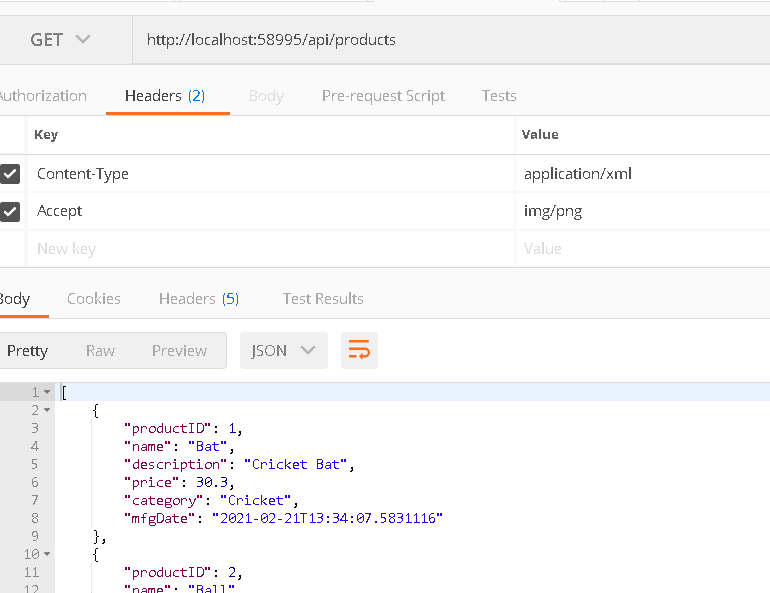


1. Check the Output



Fully Respecting Accept Headers

1. Lets try to get the data in other format



1. You can see that by default its giving us JSON data as particular format is not available.
2. In order to resolve this issue , we need to configure application to return 406 error that’s format not available

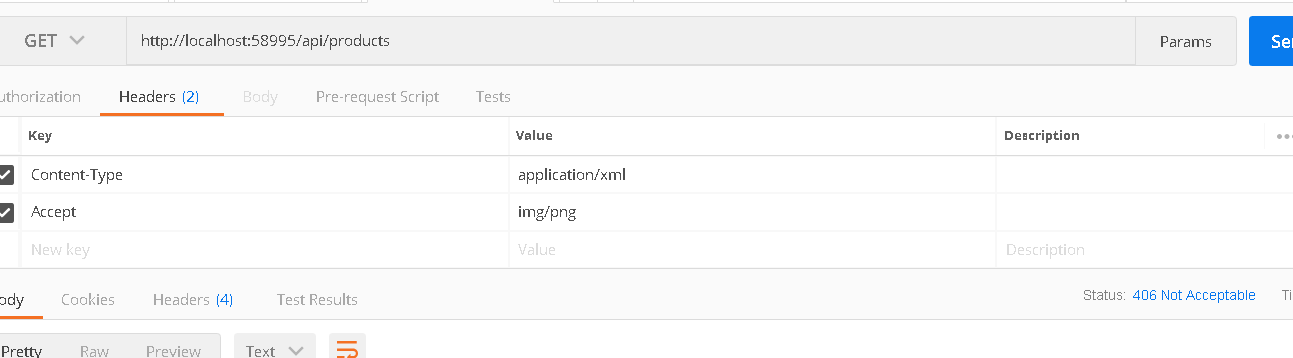
services.Configure<MvcOptions>(opts =>

{

opts.RespectBrowserAcceptHeader = true;

opts.ReturnHttpNotAcceptable = true;

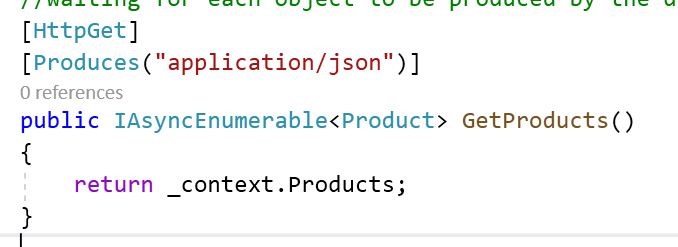
});



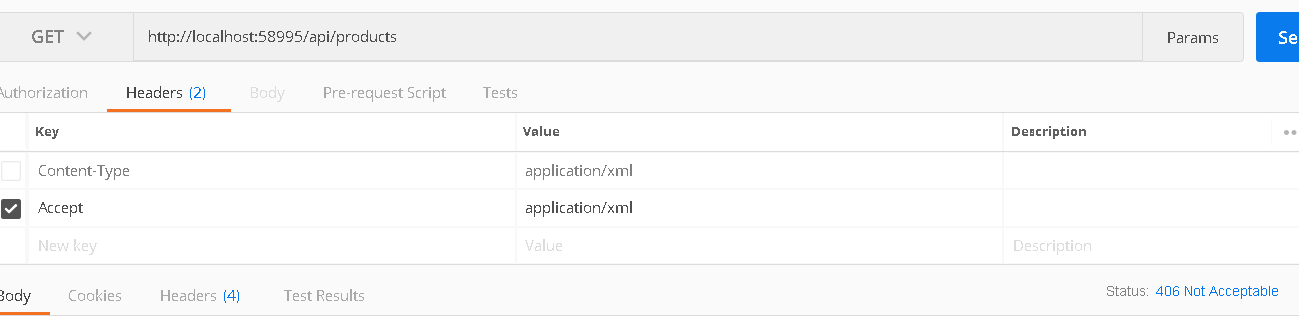
Specifying Action Result format

* Data formats that MVC framework can use for an action method result can be constrained using the produces attribute.

1. Add Produces attribute



1. Try to access it using application/xml



1. Check the Output it returns 406
2. You can add specific format on certain actions

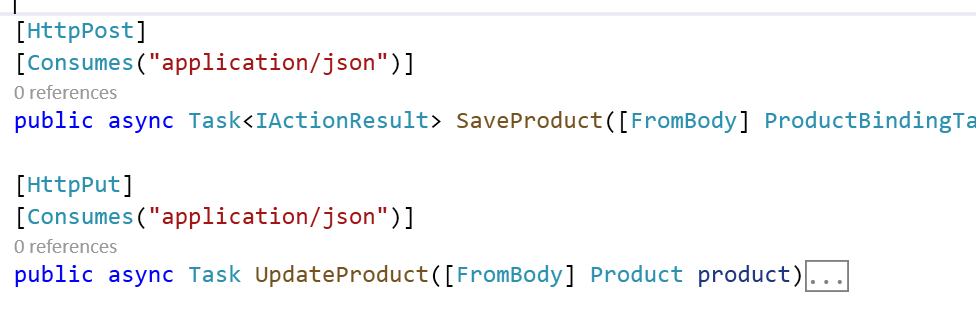


1. Check the Output : Now both the format would be working.

Restricting the format Received by the Action Method

Consumes attributes can be used to handle the Format that can be consumed by the action methods

1. Add Consumes attributes to action methods



1. Check the Output

Demo : Implementation of Produces

1. Change the action method code

[HttpPost]

[Consumes("application/json")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public async Task<IActionResult> SaveProduct([FromBody] ProductBindingTarget target)

{

if (ModelState.IsValid)

{

var p = target.ToProduct();

await \_context.Products.AddAsync(p);

await \_context.SaveChangesAsync();

return Ok(p);

}

return BadRequest(ModelState);

}

[HttpPut]

[Consumes("application/json")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public async Task<IActionResult> UpdateProduct([FromBody] Product product)

{

if (ModelState.IsValid)

{

\_context.Products.Update(product);

await \_context.SaveChangesAsync();

}

return BadRequest(ModelState);

}

[HttpGet("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public async Task<IActionResult> GetProduct(long id)

{

Product p = await \_context.Products.FindAsync(Convert.ToInt32(id));

if (p== null)

{

return NotFound();

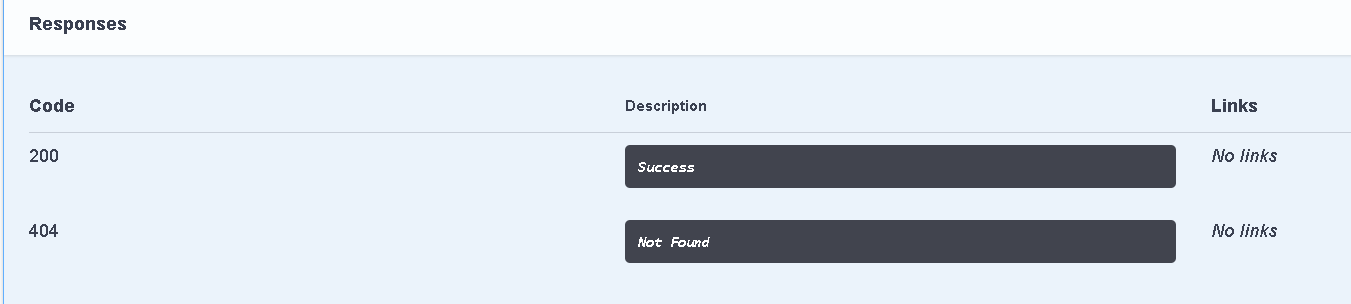
}

return Ok(new { ProductId= p.ProductID,Name=p.Name,Price=p.Price,

Category=p.Category,Description= p.Description});

}

1. Go to Swagger



1. Check because of ProducesResponse type now anyone who is consuming an API will know what will be the possible outcomes of the specific action methods

Web API Calls with JQuery

Following are the jQuery Parameters : [jQuery.ajax() | jQuery API Documentation](https://api.jquery.com/Jquery.ajax/)

1. Type : GET/PUT/POST/DELETE
2. URL : A string containing the URL to which the request is send.
3. Success: A function to be called if the request succeeds.
4. Error : A function to be called if the request fails.
5. datatype : The type of data you are expecting back from the server.
6. beforeSend : Modify jqXHR object. Use to set Custom headers.



Demo

1. Go to About.cshtml Page and Add Following code

<h4> About Us Page</h4>

<button id="btnClick" class="btn btn-primary">Get Products</button>

<button id="btnClickById" class="btn btn-warning">Get Product By ID</button>

<input type="text" id="productID"/>

<ul id="ulProducts">

</ul>

<script type="text/javascript">

$(document).ready(function () {

var ulProducts = $('#ulProducts');

$("#btnClick").click(function () {

$.ajax({

type: 'GET',

url: 'http://localhost:58995/api/products',

dataType: 'json',

success: function (data) {

ulProducts.empty();

$.each(data, function (index, val) {

ulProducts.append('<li>' + val.name + '</li>')

});

}

});

})

$("#btnClickById").click(function () {

var productID = $("#productID").val();

$.ajax({

type: 'GET',

url: 'http://localhost:58995/api/products/' + productID,

dataType: 'json',

success: function (data) {

ulProducts.empty();

ulProducts.append('<li>' + data.name + '</li>')

},

error : function (err) {

alert('No Record Found');

ulProducts.empty();

}

});

})

});

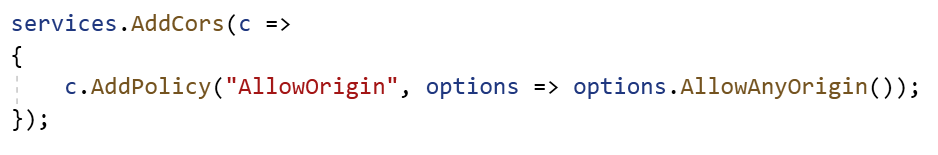
</script>

CORS Enabling : [Enable Cross-Origin Requests (CORS) in ASP.NET Core | Microsoft Docs](https://docs.microsoft.com/en-us/aspnet/core/security/cors?view=aspnetcore-3.1)

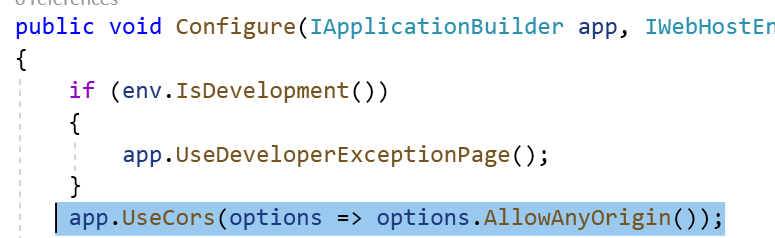
Browser security prevents a web page from making requests to a different domain than the one that served the web page. This restriction is called the *same-origin policy*. The same-origin policy prevents a malicious site from reading sensitive data from another site. Sometimes, you might want to allow other sites to make cross-origin requests to your app.

* CORS is not security feature. CORS is a W3c standard that allows server to relax the same origin policy
* API is not safer by allowing CORS
* UseCors must be placed after UseRouting, but before UseAuthorization.

1. It might give you Some CORS error to enable CORS in .net core application



1. Include this CORS capability in application object



1. Add jquery library at the top in Layout page



1. Check the Output