Swashbuckle creates the open API information.

Right now it doesn’t show everything on the API information. Just gives 200. For created – it returns 201 but not 200. We know that because we have written the code.

Open API definition is not gonna go and analyse the entire source code. It simply relies on the signature and attributes associated with the method. And makes the assumptions like the return status code.

Asp.net gives us some attributes to explain behavior of our API.

Consumes – attribute – contains the information of content type,

Produces – attribute to tell what is the response (like the object) not the status code. Like we can put it on the Get method to tell the Swashbuckle that it returns the Order object.

Right now Get(int id), returns, 404 (not found), 400, 200 but it thinks only 200.

ProducesResponseType = this gives the extra status codes that can be returned from the get (id)

[ProducesResponseType(200, Type = typeof(Order))] => in this if 200 returned, then Order is returned.

If the type is missing for 400 and 404, problem details object would be returned in the response.

Now swagger shows more details.

Asp.net provides API conventions. DefaultApiConvention is provided OOTB (out o the box).

ApiConventionMethodAttribute – on method.

ApiConventionTypeAttribute – on controller

[ApiConventionMethod(typeof(DefaultApiConventions), "Post")] -> if on Action 2 parameters requd

If on controller only one parameter is reqd. If both on controller and action, then action overrides.

Now API convention gives a lot of status code already, everything should come from convention. If we add any produces attribute ,everything would override the convention.

Also we can create our own ApiConvention

https://raw.githubusercontent.com/aspnet/Mvc/release/2.2/src/Microsoft.AspNetCore.Mvc.Core/DefaultApiConventions.cs

--- contains source code. We copy whatever is already provided, add what we want, remove what we don’t.

[ApiConventionNameMatch(ApiConventionNameMatchBehavior.Prefix)]

// Summary:

       //     The parameter or method name in the convention is a proper prefix.

       //     Casing is used to delineate words in a given name. For instance, with this behavior

       //     the convention name "Get" will match "Get", "GetPerson" or "GetById", but not

       //     "getById", "Getaway".

On controller we are adding the convention. For AddAsync, we rename it to PostAsync, so that out convention works on it.

Any convention that need to be applied at the project clas add them to the staturt up class on top of the namespace. Assembly level attribute. [assembly: ApiConventionType(typeof(CustomAPIConvention))]

Analysers works with controllers decorated with API controller attribute.

Notifies when an endpoint

* Returns an undeclared status code
* Returns an undeclared success result
* Documents a status code that is not returned.

Enable the analyser on the project leve. Add it to the csproj file. <IncludeOpenAPIAnalyzers>True</IncludeOpenAPIAnalyzers>

It will put a green line under the below if the conventions are not applied to the below.

if (order == null)

            {

                return NotFound(); //returns 404 status code.

            }

Return types:

* Specific type like List<Product> , drawback that we cannot return Ok, return …. Just the list.

Returnng different status codes is virtually impossible. With 200 – Order, with 400, Problem details object. This option hinders as a developer, industry standard things cannot be implemented.

* Iactionresult – it has helper methods that return any HTTP status code and any custom object attached to them.

Return notfound(), return ok(product),

* ActionResult<T> - this implements Iactionresult. Little bit more actually. I can actually explicitly specify, what object to be returned in case of 200.

Return notfound(), return product -> t is Product defined already but am I forced to return Product. If something else you want to return return Ok(order)

But API definition will only have product has return type.

Response Formatters:

* Action result can be explicitly controlled, JsonResult, ContentResult, Product[]
* Content negotiation occurs when client specifies an Accept header:
  + Supported – application/json, text/plain
  + Ingored accepted header from browser requests
  + Use formatfilter with {format} route segment
* Formatter can be added or removed in the ConfigureServices. Right now only format supported is JSON.

You can use accept header on JSON as well.

Model Binding

Form, Body, URL (Query or Route)

Custom model binding we can do. Right now framework just does it for us. Day to day scenarios taken care of.

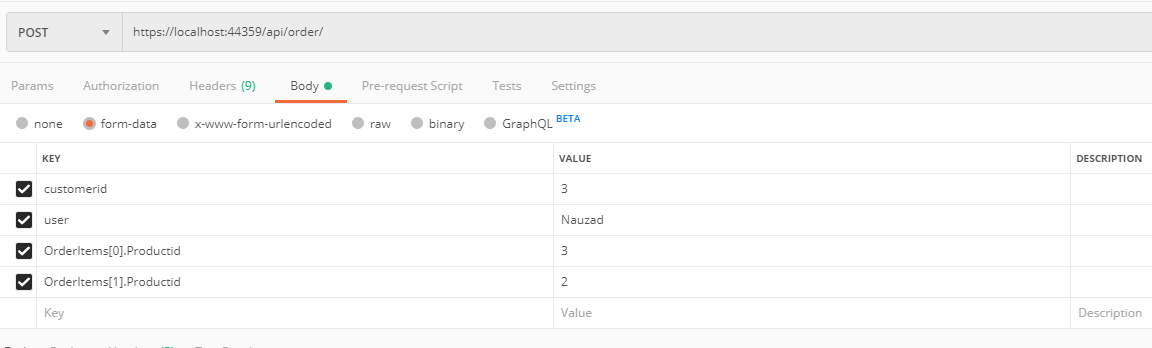
Nullable simple types are set to Null.

Non-nullable value types are set to default value like int to 0 .

Use Bind attribute. BindReuired instead of Required. Bind attributes work only on forms collection not on body Bind works with value providers not with input formatter.

Model binding -> use value provider for primitive and input formatter for complex object.

For FromForm



Suppose you have an input as below:

#### Raw certificate metadata

pub:u:4096:72C2CD6A05B514F3:1572693640:1604229640:nauzadk@lexomi.com

This has to be converted to

public class Certificate

{

public CertificateType Type { get; set; }

public TrustLevel TrustLevel { get; set; }

public int Length { get; set; }

public string Id { get; set; }

public string Fingerprint { get; set; }

public DateTime Expiry { get; set; }

public string AssociatedWith { get; set; }

}

But the string processing logic has to be written in the model binder. Take the data and convert it in the object. If you write that code in the action then you can’t reuse the object. Or if in a method, then that string processing logic would have to be called everytime you need the object.

Certificates Controller has 2 methods Post for creating a new certificate and Put for modifying an existing one.

They do not contain any string as input but a complex object -> Certificate. Custom deserialization will take place.

This code is string processing code: could have been in a method.

But here in model binder.

var certificateComponents = line.Split(':');

           Certificate certificate = new Certificate

           {

               Type = certificateComponents[0] == "pub" ? CertificateType.Public : CertificateType.Private,

               TrustLevel = certificateComponents[1] == "u" ? TrustLevel.Ultimate : TrustLevel.NotTrusted,

               Length = Int32.Parse(certificateComponents[2]),

               Id = certificateComponents[3],

               Fingerprint = certificateComponents[4],

               Expiry = DateTimeOffset.FromUnixTimeSeconds(Int64.Parse(certificateComponents[5])).UtcDateTime,

               AssociatedWith = certificateComponents[6]

           };

Associate this custom model binder with the certificate object. How ?? User Model binder with this type on the model.

Our program works on the valueprovider but we are posting the data via body, so null when calling the post method.

For valueprovider to work we post the data in the forms collection. So we use FromForm in the signature of the Post method and we use certificate : pub:u:4096:72C2CD6A05B514F3:1572693640:1604229640:nauzadk@lexomi.com

In the forms data.

Check the difference where a normal MVC will look for data and where the API will look for data. (form, body, URL)

What if we want to work with the body of the request. We would have to use IinputFormatter. Then a custom input formatter needs to be created.

Custom value provider are rarely created because only primitive data types used.

using (var reader = new StreamReader(request.Body, encoding))

in the CertiicateInputFormatter picks the data from the request body.