**HRM CODE TUTORIAL COLLECTION**

**Collection By:** R&D Team

Index

[CONFIGURE SWAGGER API 2](#_Toc23858531)

[Features 2](#_Toc23858532)

[Update Note 2](#_Toc23858533)

[How to use 3](#_Toc23858536)

# CONFIGURE SWAGGER API

## Features

### Cấu hình Swagger API ASP.NET Core với SwashBuckle.AspNetCore

Có 3 Component chính của SwashBuckle

* Swashbuckle.AspNetCore.Swagger: Một Swagger Object Model và Middleware để hiển thị SwaggerDocument object dưới dạng Json endpoints.
* Swashbuckle.AspNetCore.SwaggerGen: Là Swagger generator có nhiệm vụ build SwaggerDocument object trực tiếp từ routes, controllers, và models. Được kết hợp từ Swagger endpoint middleware để tự động phát sinh Swagger JSON
* Swashbuckle.AspNetCore.SwaggerUI: là một phiên bản nhúng của Swagger UI Tool tạo nên giao diện thể hiện API

## Update Note

* 28/12/2020: Create docs

## How to use

### Cài đặt SwashBuckle với ASP.NET Core Application

Cài đặt thông qua Package Manager với lệnh Install-Package SwashBuckle.AspNetCore

### Cấu hình Startup Class

Using Namespace

using Microsoft.OpenApi.Models;

Cấu hình ConfigureServices method

public void ConfigureServices(IServiceCollection services)

{

services.AddMvc();

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "My API", Version = "v1" });

});

}

### Cấu hình API với Attribute

Đảm bảo API của chúng ta được thêm các Attribute “Http” và “From”, nếu chúng ta không chỉ rõ Parameter Binding, Generator sẽ hiểu chúng mặc định là “Query” params.

[HttpPost]

public void AddToDo([FromBody] ToDo toDo)

{

}

### Cấu hình Middleware để hiển thị Swagger UI tại Startup Configure method

Cấu hình đầy đủ cơ bản

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

#region Swagger

app.UseSwagger();

// Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.),

// specifying the Swagger JSON endpoint.

app.UseSwaggerUI(c =>

{

c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");

c.RoutePrefix = string.Empty;

});

#endregion Swagger

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

**Tại app.UseSwaggerUI**

* c.SwaggerEndPoint dùng để thêm Swagger JSON Endpoint UI Page
* c.RoutePrefix = string.Empty để chuyển trang chủ của App thành Swagger Page

### Routing

Cấu hình Startup Configure method tương tự cấu hình Middleware ở trên

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

Tại Controller cần thêm Route Attribute để tạo đường dẫn tới Controller

[Route("api/[controller]")]

public class ToDoController : ControllerBase

{

[HttpGet]

public async Task<List<ToDo>> Get()

{

// to do

return null;

}

[HttpGet("{id:long}")]

public async Task<ToDo> Get(long id)

{

// to do

return null;

}

[HttpPost]

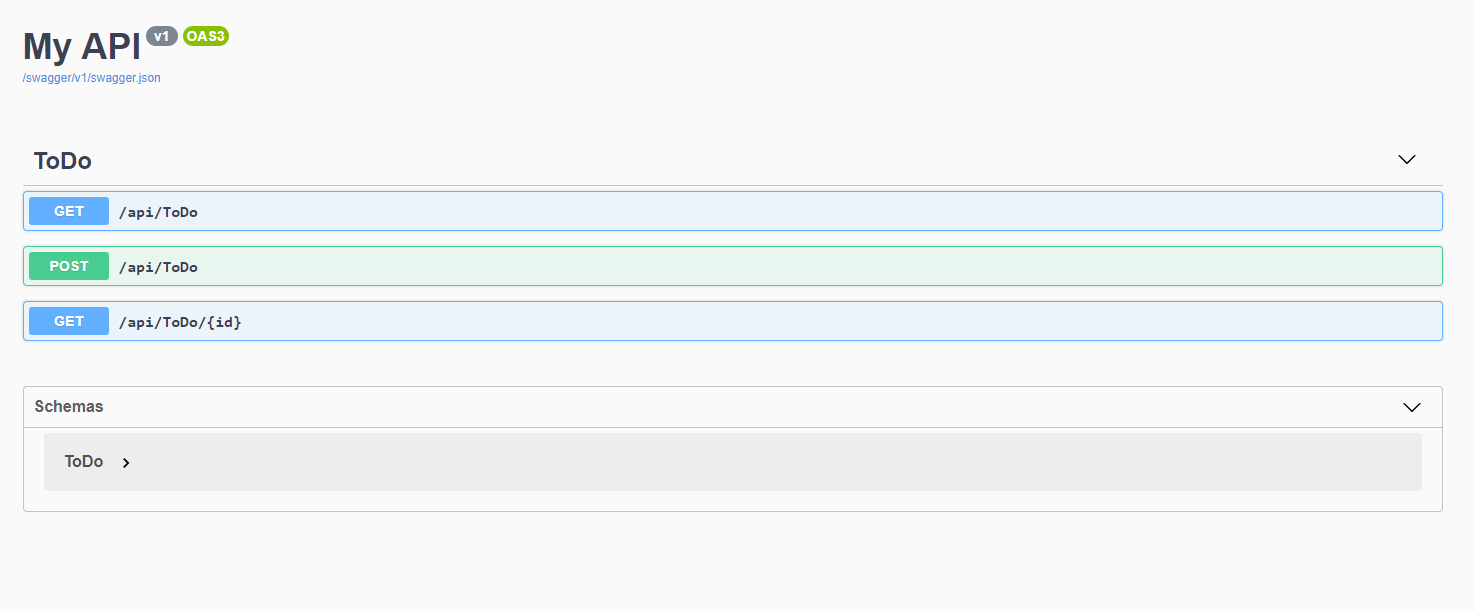
public async Task<long> Add(ToDo toDo)

{

return 1;

}

}

Kết quả chúng ta sẽ có được 

### Cấu hình và tùy biến

#### Swashbuckle.AspNetCore.Swagger

Đường dẫn mặc định tới Swagger JSON là “/swagger/{documentname}/swagger.json”. Nếu cần thiết, chúng ta có thể thay đổi đường dẫn này bằng cách sửa lại cấu hình Middleware cho Swagger

app.UseSwagger(c =>

{

c.RouteTemplate = "api-docs/{documentName}/swagger.json";

});

app.UseSwaggerUI(c =>

{

c.SwaggerEndpoint("/api-docs/v1/swagger.json", "My API V1");

c.RoutePrefix = string.Empty;

});

Nếu sử dụng SwaggerUi, chúng ta cũng cần cập nhật SwaggerEndpoint match với RouteTemplate của Swagger

##### Modify Swagger với Request Context

Nếu chúng ta cần thiết set Swagger Metadatab dựa trên Request hiện tại, chúng ta có thể cấu hình một Filter được thực thi trước khi serializing document …

#### Swashbuckle.AspNetCore.SwaggerGen

##### Assign Explicit OperationIds

Trong Swagger, một vài Operations CÓ THỂ được assign một operationId. ID PHẢI là Unique giữa các Operations được mô tả trong API. Tool và Libraries CÓ THỂ sử dụng Unique OperationId này để định danh một Operation. Vì vậy những QUY TẮC ĐẶT TÊN sau đây ĐƯỢC KHUYÊN DÙNG.

Tự động phát sinh một ID phù hợp với yêu cầu hoặc…

##### List Operation Response

Swashbuckle mặc định sẽ tự động tạo ra “200” response cho mỗi Operation. Nếu action return một Response DTO, việc này sẽ tạo ra một Schema cho Response Body

Ví dụ tạo một Request

[HttpPost("{id}")]

public ToDo GetById(int id)

Response metadata sẽ có dạng

responses: {

200: {

description: "Success",

content: {

"application/json": {

schema: {

$ref: "#/components/schemas/ToDo"

}

}

}

}

}

##### Explicit Responses

Nếu chúng ta cần chỉ định cụ thể từng Status Code cho mỗi Response, hoặc trả về IActionResult thay vì Response DTO, chúng ta có thể thực hiện thông qua ProducesResponseTypeAttribute của ASP.NET Core

Ví dụ

[ProducesResponseType(typeof(ToDo), 200)]

[ProducesResponseType(typeof(IDictionary<string, string>), 400)]

[ProducesResponseType(typeof(ToDo), 404)]

[ProducesResponseType(500)]

[HttpGet("{id:long}")]

public async Task<IActionResult> Get(long id)

{

var toDo = await \_service.Get(id);

return Ok(toDo);

}

Và Response sẽ có dạng

"/api/ToDo/{id}": {

"get": {

"tags": [

"ToDo"

],

"parameters": [

{

"name": "id",

"in": "path",

"required": true,

"schema": {

"type": "integer",

"format": "int64"

}

}

],

"responses": {

"200": {

"description": "Success",

"content": {

"application/json": {

"schema": {

"$ref": "#/components/schemas/ToDo"

}

}

}

},

"400": {

"description": "Bad Request",

"content": {

"application/json": {

"schema": {

"type": "object",

"additionalProperties": {

"type": "string"

}

}

}

}

},

…

…

##### Flag Required Parameters và Schema Properties

Trong Swagger Document, chúng ta có thể gắn Flag Parameters và Schema Properties cho các Properties Request đòi hỏi. Nếu một Parameter (Top-level hoặc Property-based) được đánh dấu bởi BindRequiredAttribute hoặc RequiredAttribute, thì Swashbuckle sẽ tự động gắn Flag “Required” cho các Parameter được tạo bởi Swagger

public IActionResult Search([FromQuery, BindRequired] string keywords, [FromQuery] PagingParams pagingParams)

{

if (!ModelState.IsValid)

return BadRequest(ModelState);

...

}

// SearchParams.cs

public class PagingParams

{

[Required]

public int PageNo { get; set; }

public int PageSize { get; set; }

}

##### Include Descriptions với XML Comment

Để có thể tăng mức độ trực quan của Swagger UI, chúng ta có thể thêm các ghi chú cho Controller Action và Models với XML Comments và cấu hình Swashbuckle để tạo ra một giao diện thân thiện người dùng hơn.

###### XML Comment

Visual Studio

Double click vào Project để sửa file Projectname.csproj và thêm code XML như sau:

<PropertyGroup>

<GenerateDocumentationFile>true</GenerateDocumentationFile>

<NoWarn>$(NoWarn);1591</NoWarn>

</PropertyGroup>

Visual Studio for MAC

Từ Solution Pad, nhấn Control và Click vào Project Name -> Chọn Tools -> Edit File

Thêm XML code

<PropertyGroup>

<GenerateDocumentationFile>true</GenerateDocumentationFile>

<NoWarn>$(NoWarn);1591</NoWarn>

</PropertyGroup>

Visual Studio Code

Thêm XML code vào file .csproj

<PropertyGroup>

<GenerateDocumentationFile>true</GenerateDocumentationFile>

<NoWarn>$(NoWarn);1591</NoWarn>

</PropertyGroup>

Cấu hình Swagger để có thể sử dụng XML Comment

public void ConfigureServices(IServiceCollection services)

{

services.AddMvc();

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "My API", Version = "v1" });

// Set the comments path for the Swagger JSON and UI.

var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";

var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);

c.IncludeXmlComments(xmlPath);

});

services.AddScoped<IToDoService, ToDoService>();

}

Trong đoạn code ở trên, Reflection được sử dụng để build một XML file name phù hợp với Web API Project.

AppContext.BaseDirectory property được sử dụng để tạo đường dẫn tới XML file.

Thêm Comment Cho Controller Action

/// <summary>

/// Add New Product

/// </summary>

/// <remarks>

/// Sample request:

///

/// POST /

/// {

/// "name": "Do HomeWork",

/// }

///

/// </remarks>

/// <param name="request"></param>

/// <returns></returns>

/// <response code="201">To do created with ID</response>

/// <response code="400">To do has missing/invalid values</response>

/// <response code="500">Oops! Can't create your To do right now</response>

[HttpPost]

[ProducesResponseType(typeof(int), 201)]

[ProducesResponseType(typeof(IDictionary<string, string>), 400)]

[ProducesResponseType(500)]

public async Task<IActionResult> Add(CreateToDoRequest request)

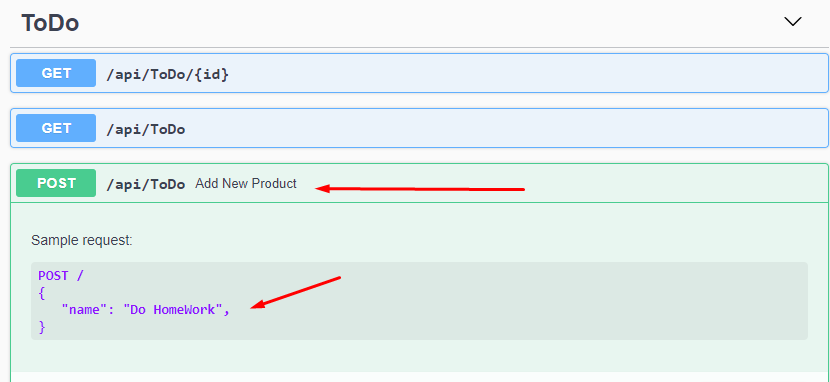
{

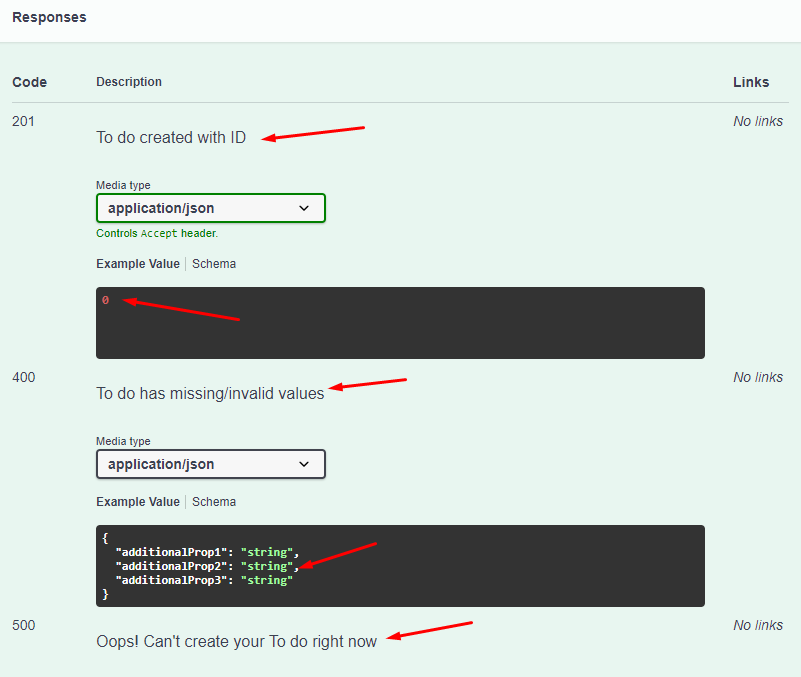
var result = await \_service.Add(request);

return Ok(result);

}

Kết quả





##### Cung cấp Global API Metadata

Swagger ngoài cung cấp “PathItems”, “Operations” and “Responses” còn hỗ trợ Global Metadata. Ví dụ, chúng ta có thể cung cấp đầy đủ mô tả cho API, bao gồm “Term Of Service”, “Contact”, …

Cấu hình lại ConfigureServices như sau

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "My API - V1",

Version = "v1",

Description = "A sample API to demo Swashbuckle",

TermsOfService = new Uri("http://tempuri.org/terms"),

Contact = new OpenApiContact

{

Name = "Minh Van Developer",

Email = "van.nguyen@vnresource.org"

},

License = new OpenApiLicense

{

Name = "Apache 2.0",

Url = new Uri("http://www.apache.org/licenses/LICENSE-2.0.html")

}

});

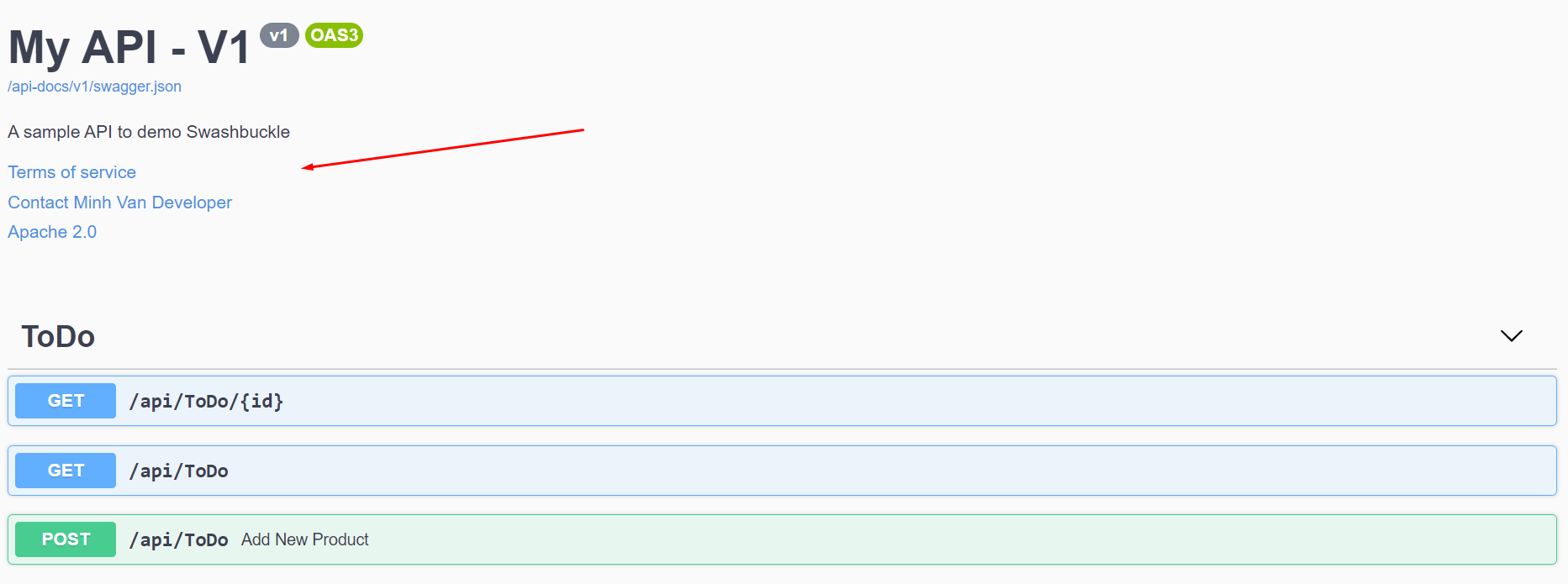
// Set the comments path for the Swagger JSON and UI.

var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";

var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);

c.IncludeXmlComments(xmlPath);

});



##### Tạo Multiple Swagger Documents

Với các Setup ở trên, Generator sẽ bao gồm toàn bộ API Operations trên một Swagger Document duy nhất. Tuy nhiên chúng ta có thể tạo ra tùy ý số lượng Document nếu mong muốn.

Ví dụ, chúng ta có thể chia Document cho các Version của API. Để thực hiện việc này, chúng ta cần định nghĩa Multiple Swagger Docs tại Startup.cs

###### Tại ConfigureServices Method

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "My API - V1",

Version = "v1",

Description = "A sample API to demo Swashbuckle",

TermsOfService = new Uri("http://tempuri.org/terms"),

Contact = new OpenApiContact

{

Name = "Minh Van Developer",

Email = "van.nguyen@vnresource.org"

},

License = new OpenApiLicense

{

Name = "Apache 2.0",

Url = new Uri("http://www.apache.org/licenses/LICENSE-2.0.html")

}

});

c.SwaggerDoc("v2", new OpenApiInfo { Title = "My API - V2", Version = "v2" });

// Set the comments path for the Swagger JSON and UI.

var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";

var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);

c.IncludeXmlComments(xmlPath);

});

###### Cấu hình Middleware Configure Method

app.UseSwagger(c =>

{

c.RouteTemplate = "api-docs/{documentName}/swagger.json";

});

// Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.),

// specifying the Swagger JSON endpoint.

app.UseSwaggerUI(c =>

{

c.SwaggerEndpoint("/api-docs/v1/swagger.json", "My API V1");

c.SwaggerEndpoint("/api-docs/v2/swagger.json", "My API V2");

c.RoutePrefix = string.Empty;

});

###### Thêm Attribute cho Controller Action

[HttpPost]

[ApiExplorerSettings(GroupName = "v2")]

[ProducesResponseType(typeof(int), 201)]

[ProducesResponseType(typeof(IDictionary<string, string>), 400)]

[ProducesResponseType(500)]

public async Task<IActionResult> Add(CreateToDoRequest request)

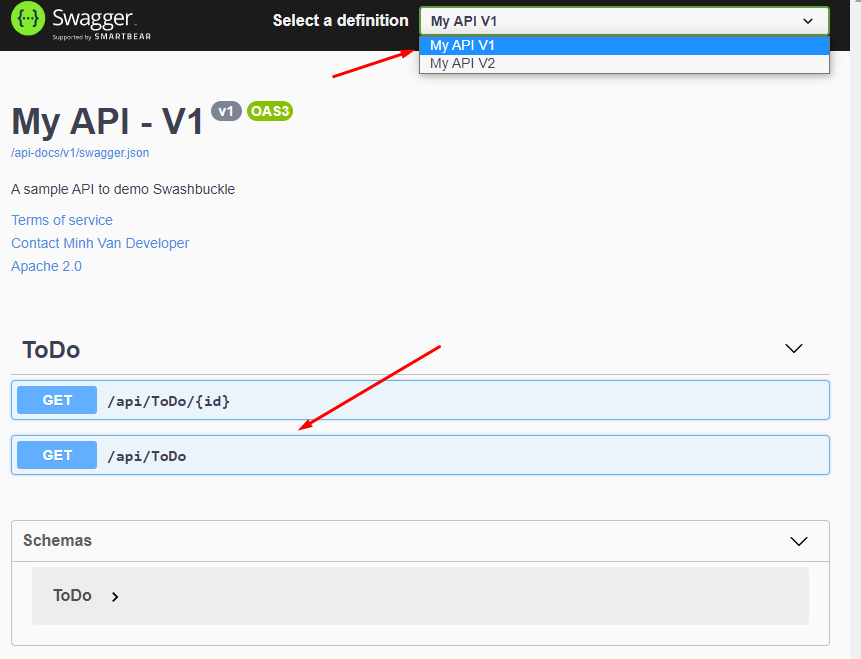
{

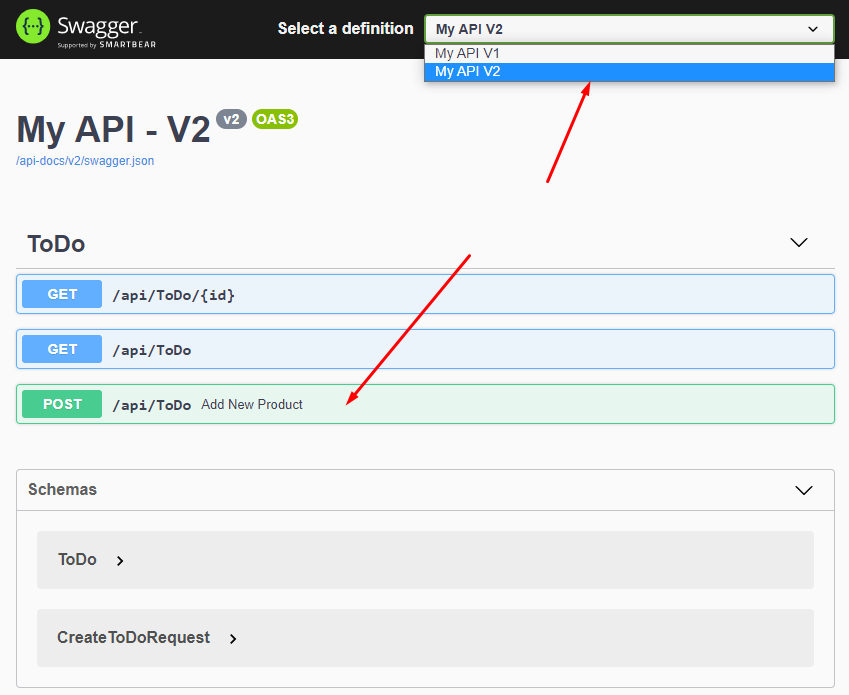
var result = await \_service.Add(request);

return Ok(result);

}

Thêm ApiExplorerSettingsAttribute cho Add Action, từ đó Add Todo sẽ thuộc về API Version 2 thay vì Version1

Api version 1  


Api Version 2  


Như cấu hình, hiện tại Add Todo Action chỉ hiển thị khi sử dụng API Version 2. Actions không được cấu hình chỉ định thuộc về Version nào mặc định sẽ có tại tất cả Version.

##### Loại bỏ Obsolete Operations và/hoặc Schema Property

Swagger Spec bao gồm một Flag là “Deprecated” ám chỉ một Operation đã lỗi thời và nên hạn chế được sử dụng. Swagger Generator sẽ tự động tạo Flag này nếu một Action được gán Attribute ObsoleteAttribute. Tuy nhiên, thay vì tạo một Obsolete Flag, chúng ta có thể cấu hình để Generator bỏ qua Obsolete Actions

…

##### Loại bỏ Operations bất kỳ

…

##### Customize Operation Tag

Swagger Spec cho phép hơn một Tag được Assign cho một Operation. Swagger Generator sẽ Assign Default Tag là tên của Controller. Việc này quan trọng bởi vì nếu chúng ta sử dụng SwaggerUI middleware, chúng sẽ sử dụng những giá trị này để tạo nhóm Operations.

Chúng ta có thể Override Default Tag bằng cách cung cấp một Function để Apply Tag.

Ví dụ về cấu hình Tag và Group Operation trên UI bằng HTTP Method (Obsolete)

services.AddSwaggerGen(c =>

{

…

c.TagActionsBy(api => api.HttpMethod);

…

});

##### Thêm Security Definitions và Requirements

Với Swagger, chúng ta có thể mô tả việc API được Secured bằng cách định nghĩa một hoặc nhiều Security Schemes (basic, api key, oath2 …) và khai báo chúng cho việc sử dụng trên toàn bộ API hoặc các Operaiton cụ thể.

Swashbuckle cho phép chúng ta Define Schemes bằng cách invoke AddSecurityDefinition method, cung cấp tên và một Instance của OpenApiSecurityScheme

Ví dụ định nghĩa OAuth 2.0 implicit flow

services.AddSwaggerGen(c =>

{

c.AddSecurityDefinition("oauth2", new OpenApiSecurityScheme

{

Type = SecuritySchemeType.OAuth2,

Flows = new OpenApiOAuthFlows

{

Implicit = new OpenApiOAuthFlow

{

AuthorizationUrl = new Uri("/auth-server/connect/authorize", UriKind.Relative),

Scopes = new Dictionary<string, string>

{

{ "readAccess", "Access read operations" },

{ "writeAccess", "Access write operations" }

}

}

}

});

});

Với API Key

options.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

In = ParameterLocation.Header,

Description = "Please insert JWT with Bearer into field",

Name = "Authorization",

Type = SecuritySchemeType.ApiKey

});

Chi tiết Authentication tại Authentication Docs (…)

##### …

#### Swashbuckle.AspNetCore.SwaggerUI

##### Thay đổi UI Relative Path

Mặc định, Swagger UI có địa chỉ tại /swagger. Nếu cần thiết, chúng ta có thể thay đổi địa chỉ này trong SwaggerUI middleware tại Configure method

app.UseSwaggerUI(c =>

{

c.SwaggerEndpoint("/api-docs/v1/swagger.json", "My API V1");

c.SwaggerEndpoint("/api-docs/v2/swagger.json", "My API V2");

c.RoutePrefix = string.Empty;

});

c.RoutePrefix = string.Empty sẽ chuyển SwaggerUI thành trang chủ của Application.

##### Liệt kê danh sách Swagger Document

Đã được giới thiệu tại Swashbuckle.AspNetCore.SwaggerGen Tạo Multiple Swagger Document

app.UseSwaggerUI(c =>

{

c.SwaggerEndpoint("/api-docs/v1/swagger.json", "My API V1");

c.SwaggerEndpoint("/api-docs/v2/swagger.json", "My API V2");

c.RoutePrefix = string.Empty;

});

##### Apply SwaggerUI Parameters

Danh sách Swagger Parameters

app.UseSwaggerUI(c =>

{

c.DefaultModelExpandDepth(2);

c.DefaultModelRendering(ModelRendering.Model);

c.DefaultModelsExpandDepth(-1);

c.DisplayOperationId();

c.DisplayRequestDuration();

c.DocExpansion(DocExpansion.None);

c.EnableDeepLinking();

c.EnableFilter();

c.MaxDisplayedTags(5);

c.ShowExtensions();

c.ShowCommonExtensions();

c.EnableValidator();

c.SupportedSubmitMethods(SubmitMethod.Get, SubmitMethod.Head);

});

…

##### Inject Custom CSS

Chúng ta có thể custom CSS Stylesheets nếu muốn bằng cách thêm chúng vào wwwroot folder và chỉ cấu hình SwaggerUI tới CSS file bằng cách

app.UseSwaggerUI(c =>

{

...

c.InjectStylesheet("/swagger-ui/custom.css");

}

##### Customize index.html

app.UseSwaggerUI(c =>

{

c.IndexStream = () => GetType().Assembly

.GetManifestResourceStream("CustomUIIndex.Swagger.index.html");

// requires file to be added as an embedded resource

})

##### Enable OAuth2.0

SwaggerUI có built-in hỗ trợ việc OAuth2.0 Authorization bằng cách tương tác vơi Authorization và/hoặc Token Endpoints như được chỉ định trong Swagger JSON để nhận được Access Token cho các lần gọi API sau.

Nếu chúng ta sử dụng Swagger Endpoint bao gồm Security Metadata thích hợp, UI sẽ được tự động cập nhật hỗ trợ Authorization.

#### Swashbuckle.AspNetCore.Annotations

##### Cài đặt

###### Install Nuget package cho ASP.NET Core Application

Package Manager: Install-Package Swashbuckle.AspNetCore.Annotations

CLI: dotnet add package Swashbuckle.AspNetCore.Annotations

###### Cấu hình ConfigureServices

services.AddSwaggerGen(c =>

{

c.EnableAnnotations();

});

##### Enrich Operation Metadata

Một khi Annotations được cài cài đặt thành công, chúng ta có thể sử dụng SwaggerOperationAttribute cho Controller Action

[HttpPost]

[SwaggerOperation(

Summary = "Creates a new SomeModel",

Description = "Requires admin privileges",

OperationId = "CreateSomeModel",

Tags = new[] { "ModelA", "ModelB" }

)]

public async Task<IActionResult> Create([FromBody] SomeModel model)

{

var some = new SomeModel

{

Id = 1,

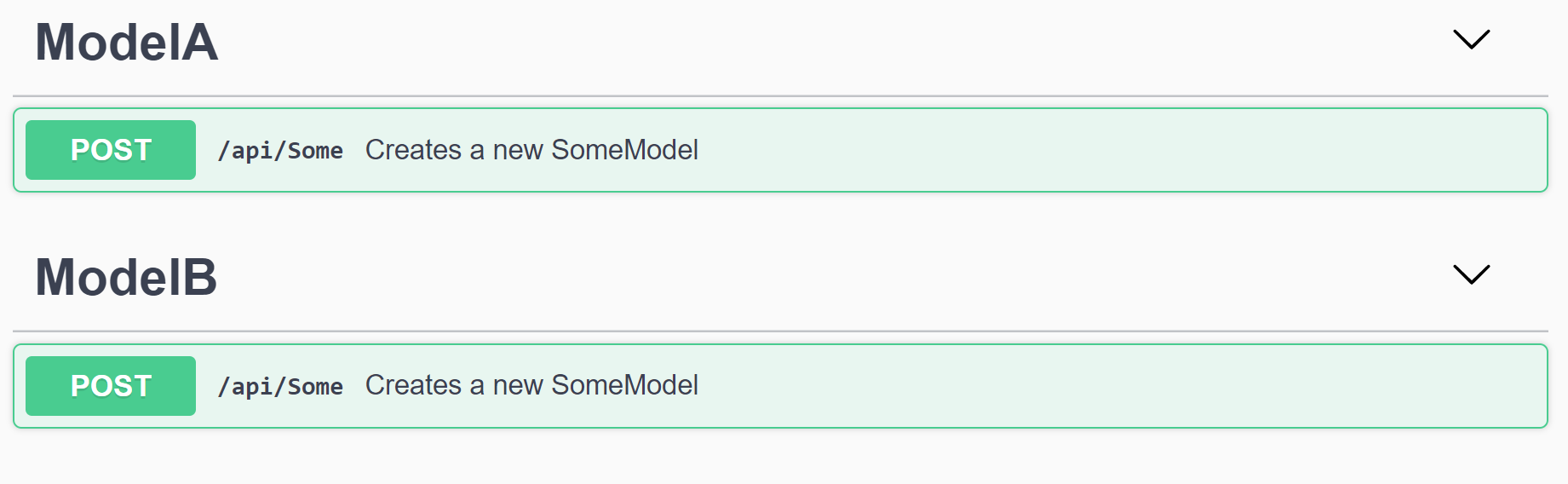
Name = "Some thing"

};

return Ok(some);

}

Kêt quả hiển thị trên UI



##### Enrich Response Metadata

ASP.NET Core cung cấp ProducesResponseTypeAttribute để liệt kê các Responses khác nhau có thể trả về bởi Action. Những Attributes đó có thể kết hợp với XML Comments như đã đề cập ở các phần trên. Tuy nhiên nếu chúng ta không muốn sử dụng XML Comments thì có thể sử dụng SwaggerResponseAttribute

[HttpPost]

[SwaggerOperation(

Summary = "Creates a new SomeModel",

Description = "Requires admin privileges",

OperationId = "CreateSomeModel",

Tags = new[] { "ModelA", "ModelB" }

)]

[SwaggerResponse(201, "SomeModel was created", typeof(SomeModel))]

[SwaggerResponse(400, "SomeModel data is invalid")]

public async Task<IActionResult> Create([FromBody] SomeModel model)

{

var some = new SomeModel

{

Id = 1,

Name = "Some thing"

};

return Ok(some);

}

##### Enrich Parameter Metadata

Chúng ta có thể thêm Annotation “path”, “query”, “header” bọc ngoài Parameters hoặc Properties (sử dụng [FromRoute], [FromQuery], hoặc [FromHeader]) với SwaggerParameterAttribute để Enrich Parameter Metadata được tạo bởi Swashbuckle

[HttpGet]

public async Task<IActionResult> Get([FromQuery, SwaggerParameter("Search Keywords", Required = true)] string searchKeywords)

{

return Ok(1);

}

##### Enrich RequestBody Metadata

Chúng ta có thể thêm Annotation “body” bọc ngoài Parameters hoặc Properties (sử dụng với [FromBody]) với SwaggerRequestBodyAttribute để Enrich RequestBody Metadata được tạo bởi Swashbuckle

public async Task<IActionResult> Create([FromBody, SwaggerRequestBody("The Model payload", Required = true)] SomeModel model)

{

return Ok(1);

}

##### Enrich Schema Metadata

Chúng ta có thể thêm Annotation Classes hoặc Properties với SwaggerSchemaAttribute để Enrich Schema được tạo bởi Swashbuckle

##### Apply Schema Filter cho Specific Types

SwaggerGen Package cung cấp một vài Extension bao gồm Schema Filters cho việc Customize ALL Generate Schemas. Tuy nhiên trong một vài trường hợp đặc biệt chúng ta mong muốn Apply những Filters này cho một vài Schema cụ thể. Ví dụ nếu chúng ta mong muốn bao gồm một Example cho một Specific Type của API, việc này có thể hoàn tất bằng việc sử dụng SwaggerShemaFilterAttribute

public class TodoSchemaFilter : ISchemaFilter

{

public void Apply(OpenApiSchema schema, SchemaFilterContext context)

{

schema.Example = new OpenApiObject

{

["Id"] = new OpenApiInteger(1),

["Name"] = new OpenApiString("To do name"),

["Description"] = new OpenApiString("To do description")

};

}

}

###### Tạo một Dropdown select thay vì textbox SwaggerUI với Enum sử dụng SchemaFilter

DescribeAllEnumsAsStrings (Desprecated)

Theo mặc định, Swashbuckle mô tả mọi Enum theo kiểu Interger. Tuy nhiên chúng ta có thể thay đổi Serializer Behavior thông qua DescribeAllEnumsAsStrings, điểu này giúp chuyển đổi mọi giá trị Enum từ Interger sang String.

Tuy nhiên DescribeAllEnumsAsStrings bây giờ đã bị đánh dấu Obsolete (Nguyên nhân liên quan tới NewtonSoft.Json), vì vậy chúng ta có thể tạo ra một cách thay thế bằng cách sử dụng SchemaFilter

Ví dụ với Model

public class CreateToDoRequest

{

public TodoName Name { get; set; }

public string Description { get; set; }

public bool TrueOrFalse { get; set; }

}

public enum TodoName

{

[Description("Home work")]

HomeWork,

[Description("Not home work")]

NotHomeWork

}

Và Controller Action

[HttpPost]

[SwaggerResponse(200, "Created", typeof(int))]

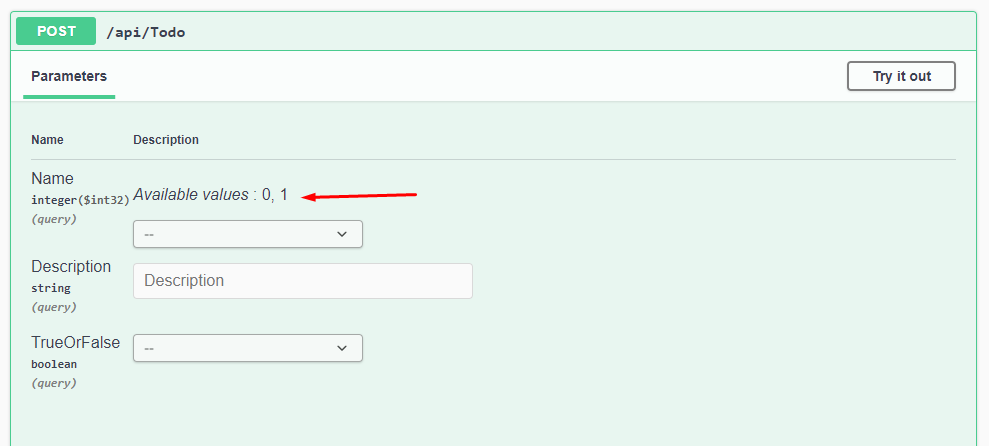
[SwaggerResponse(400, " Request invalid")]

public async Task<IActionResult> Add(CreateToDoRequest request)

{

return Ok(1);

}



Sử dụng SChemaFilter để mô tả Enum theo kiểu String thay vì Interger

Tạo một class kế thừa từ ISchemaFilter

public class EnumSchemaFilter : ISchemaFilter

{

public void Apply(OpenApiSchema model, SchemaFilterContext context)

{

if (context.Type.IsEnum)

{

model.Enum.Clear();

Enum.GetNames(context.Type)

.ToList()

.ForEach(n => model.Enum.Add(new OpenApiString(n)));

}

}

}

Sử dụng SchemaFilter này tại Startup ConfgurationServices

services.AddSwaggerGen(c =>

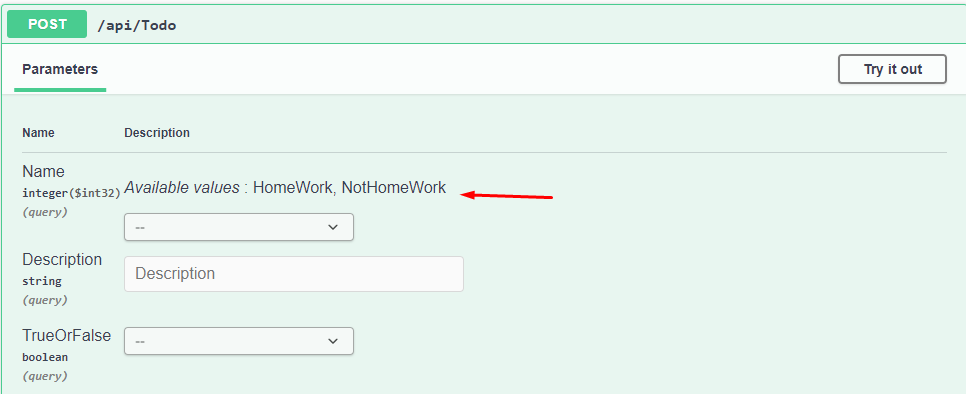
{

c.SchemaFilter<EnumSchemaFilter>();

c.SwaggerDoc("v1", new OpenApiInfo { Title = "My API", Version = "v1" });

});

Kết quả thay đổi so với trước khi Filter



##### Add Tag Metadata

Theo mặc định, Swagger Generator sẽ tự động tag tất cả Operation với Controller Name. Tag này sẽ được sử dụng để gom nhóm các Operation trên SwaggerUI. Nếu chúng ta muôn cung cấp thông tin mô tả cho mỗi nhóm đó, chúng ta có thể thêm Metadata cho mỗi Controller Name thông qua SwaggerTagAttribute

[SwaggerTag("Create, read, update and delete Todos")]

public class ProductsController

{

}

##### List Know Subtypes for Inheritance and Polymorphism

##### Enrich Polymorphic Base Classes with Discriminator Metadata

#### Swashbuckle.AspNetCore.Cli

##### Retrieve Swagger Directly from Startup Assembly

Một khi Application được setup với Swashbuckle, chúng ta có thể sử dụng Swashbuckle CLI tool để retrieve Swagger/OpenAPI JSON trực tiếp từ Application’s Startup Assembly và ghi chúng vào file. Điều này có thể có ích nếu chúng ta mong muốn liên kết Swagger Generation với một CI/CD process, hoặc nếu chúng ta muốn sử dụng chúng như một static file trong run-tim