# Documentação

Para usar o NSwag no middleware do ASP.NET Core é preciso instalar o NuGet package NSwag.AspNetCore. Este package contém o middleware para gerar e servir a especificação do Swagger, Swagger UI (V2 e V3) e ReDoc UI.

Primeiro é preciso adicionar o método AddVersionedApiExplorer para o NSwag ser capaz de detetar e substituir o parâmetro versionAPI passado na route dos endpoints pela versão da API que está a ser utilizada de momento, ou para a versão da especificação a ser utilizada caso tenhamos documentação para mais do que uma versão da nossa API.

```
services.AddVersionedApiExplorer(options => options.SubstituteApiVersionInUrl =
true);
```

O que resulta, na versão atual da SKD, em:

```
protected virtual void AddMvc(IServiceCollection services)
{
    // API controllers with views
    IMvcBuilder builder = services
        .AddApiControllersWithViews(
            (options) =>
            {
                // API versionin
                options.UseVersioning = true;
                options.DefaultApiVersion = new ApiVersion(1, 0);

                // Throttling
                options.UseThrottling = true;
            });

    // Custom configuration

    this.ConfigureMvc(services, builder);

    services.AddVersionedApiExplorer(options => options.SubstituteApiVersionInUrl
= true);
}
```

Posteriomente é preciso adicionar o middleware do NSwag à classe Startup. Tanto no método Configure como ConfigureServices, este middleware pode estar no fim da pipeline, ou pelo menos deverá estar a seguir ao versioning e routing.

***Nota:*** Podem haver casos pontuais em que será necessário meter o middleware num sitio especifico. Por exemplo, durante os testes do gRPC o Swagger não funcionava porque o mapping dos serviços do gRPC fazia

conflito com o routing da documentação Swagger e retornava um 405 apesar desta route ser detetada. Foi, neste caso, necessário meter o método UseDocumentation antes do UseEndpoints para ambos funcionarem.

No ***ConfigureServices*** podemos criar um método addDocumention(services) ao qual temos que passar o parâmetro services do tipo IServiceCollection.

```
// Add Documentation

this.AddDocumentation(services);
```

Aqui, o middleware para registar os serviços do Swagger requeridos é o AddOpenApiDocument() e, dentro deste estão todas as opções para documentar o modelo do objeto, de forma a facilitar o consumo da Web API. Pode ser passada uma ação de configuração para adicionar informação adicional e também, como neste caso, para adicionar autenticação. No caso da autenticação esta pode ser de vários tipo. Neste caso está a ser implementada para autenticar através do Identity Server da Primavera, porém podia ser também feita através de um token JWT diretamente.

```
protected virtual void AddDocumentation(IServiceCollection services)
{
    // NSwag
    // Register the Swagger services
    HostConfiguration hostConfiguration = services.GetHostConfiguration();

    services.AddOpenApiDocument(config =>
    {
        config.FlattenInheritanceHierarchy = true;
        config.PostProcess = document =>
        {
            document.Info.Version = hostConfiguration.Information.Version;
            document.Info.Title = hostConfiguration.Information.HostTitle;
            document.Info.Description = hostConfiguration.Information.ProductName;
            document.Info.TermsOfService = new
Uri("https://pt.primaverabss.com/pt/site/termos-de-utilizacao/").ToString();
            document.Info.Contact = new NSwag.OpenApiContact
            {
                Name = hostConfiguration.Information.Company,
                Email = string.Empty,
                Url = new Uri("https://pt.primaverabss.com/pt/").ToString()
            };
            document.Info.License = new NSwag.OpenApiLicense
            {
                Name = hostConfiguration.Information.Company + " " +
hostConfiguration.Information.Copyright,
                Url = new Uri("https://pt.primaverabss.com/pt/").ToString()
            };
        };

        config.AddSecurity("bearer", Enumerable.Empty<string>(), new
OpenApiSecurityScheme
```

```
        {
            Type = OpenApiSecuritySchemeType.OAuth2,
            Description = "Lithium Sample API",
            Flow = OpenApiOAuth2Flow.Implicit, //Application,
            Flows = new OpenApiOAuthFlows()
            {
                Implicit = new OpenApiOAuthFlow()
                {
                    Scopes = new Dictionary<string, string>
                    {
                        { "lithium-sample", "Access lithium-sample API." }
                    },
                    AuthorizationUrl = $"
{hostConfiguration.IdentityServerBaseUri.Trim()}/connect/authorize",
                    TokenUrl = $"
{hostConfiguration.IdentityServerBaseUri.Trim()}/connect/token",
                    RefreshUrl = $"
{hostConfiguration.IdentityServerBaseUri.Trim()}/connect/token"
                },
            }
        });

        config.OperationProcessors.Add(
                new AspNetCoreOperationSecurityScopeProcessor("bearer"));
    });
}
```

No ***Configure*** podemos também criar um método, nest caso, UseDocumentation(app) ao qual temos que passar o parâmetro app do tipo IApplicationBuilder.

```
// Documentation

this.UseDocumentation(app);
```

Dentro deste método o middleware servirá para gerar a especificação Swagger (UseOpenApi()) e o SwaggerUI (UseSwaggerUi3). Em ambos é possível passar parâmetros adicionais para a configuração adicional das mesmas funcionalidades.

```
protected virtual void UseDocumentation(IApplicationBuilder app)
{
    // NSwag
    // Register the Swagger generator and the Swagger UI middlewares
    // Add Swagger 2.0 document serving middleware
    app.UseOpenApi();
    app.UseSwaggerUi3(settings =>
    {
        settings.DocExpansion = "list";
        settings.OAuth2Client = new OAuth2ClientSettings
        {
```

```
            ClientId = "lithium-sample-implicit",
            ClientSecret =
"A8vYHgFsy3DnASwBUGwtsgabYtEnUodj8uDdiy8wmPZV0XdU2YVDrtHrpVUEAOKh4PlkYfGgLjyf+1ioZ
h+DIw== "
        };

        settings.OAuth2Client.AdditionalQueryStringParameters.Add("redirect_uri",
"http://localhost:4200");
    });
}
```

Isto é tudo o que é necessário para gerar a documentação e o respetivo documento JSON com as especificações do Swagger.

## Usar diferentes versões da API

Para ser possível ter documentação para várias versões da API é preciso criar diferentes instâncias do middleware do Swagger (AddOpenApiDocument) em que cada uma representa uma versão diferente da API. De forma a criar versões diferentes é preciso passar o parâmetro DocumentName e ApiGroupNames para a respetiva versão e tudo o resto é, em principio, transversal a todas as versões da API sendo só preciso ser escrito na primeira versão porque deverá ser herdado por todas as outras.

É também necessário que no AddVersionedApiExplorer se adicione o GroupNameFormat com o tipo de formatação que estamos a utilizar para versionar a nossa API. Mais informações acerca desta formatação e dos parâmetros aceites: https://github.com/microsoft/aspnet-api-versioning/wiki/Version-Format

Já no AddApiVersioning, o AssumeDefaultVersionWhenUnspecified deve ser passado a true para que, caso não seja especificada uma versão da API, ele retorne a versão default.

***Nota:*** Isto muda um pouco consoante a versão do .NET Core e do NSwag a ser utilizada.

```
public void ConfigureServices(IServiceCollection services)
{
    services
        .AddMvc(config =>
        {
            config.InputFormatters.Add(new CustomTextInputFormatter());
            config.OutputFormatters.Add(new CustomTextOutputFormatter());
        })
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    services.AddApiVersioning(options =>
    {
        options.AssumeDefaultVersionWhenUnspecified = true;
        options.ApiVersionReader = new UrlSegmentApiVersionReader();
    })
    .AddMvcCore()
    .AddVersionedApiExplorer(options =>
    {
        options.GroupNameFormat = "VVV";
```

```
            options.SubstituteApiVersionInUrl = true;
        });

        services
            .AddOpenApiDocument(document =>
            {
                document.DocumentName = "v1";
                document.ApiGroupNames = new[] { "1" };
            })
            .AddOpenApiDocument(document =>
            {
                document.DocumentName = "v2";
                document.ApiGroupNames = new[] { "2" };
            })
            .AddOpenApiDocument(document =>
            {
                document.DocumentName = "v3";
                document.ApiGroupNames = new[] { "3" };
            });
    }
```

Sample Project:
https://github.com/RicoSuter/NSwag/blob/master/src/NSwag.Generation.AspNetCore.Tests.Web/Startup.cs