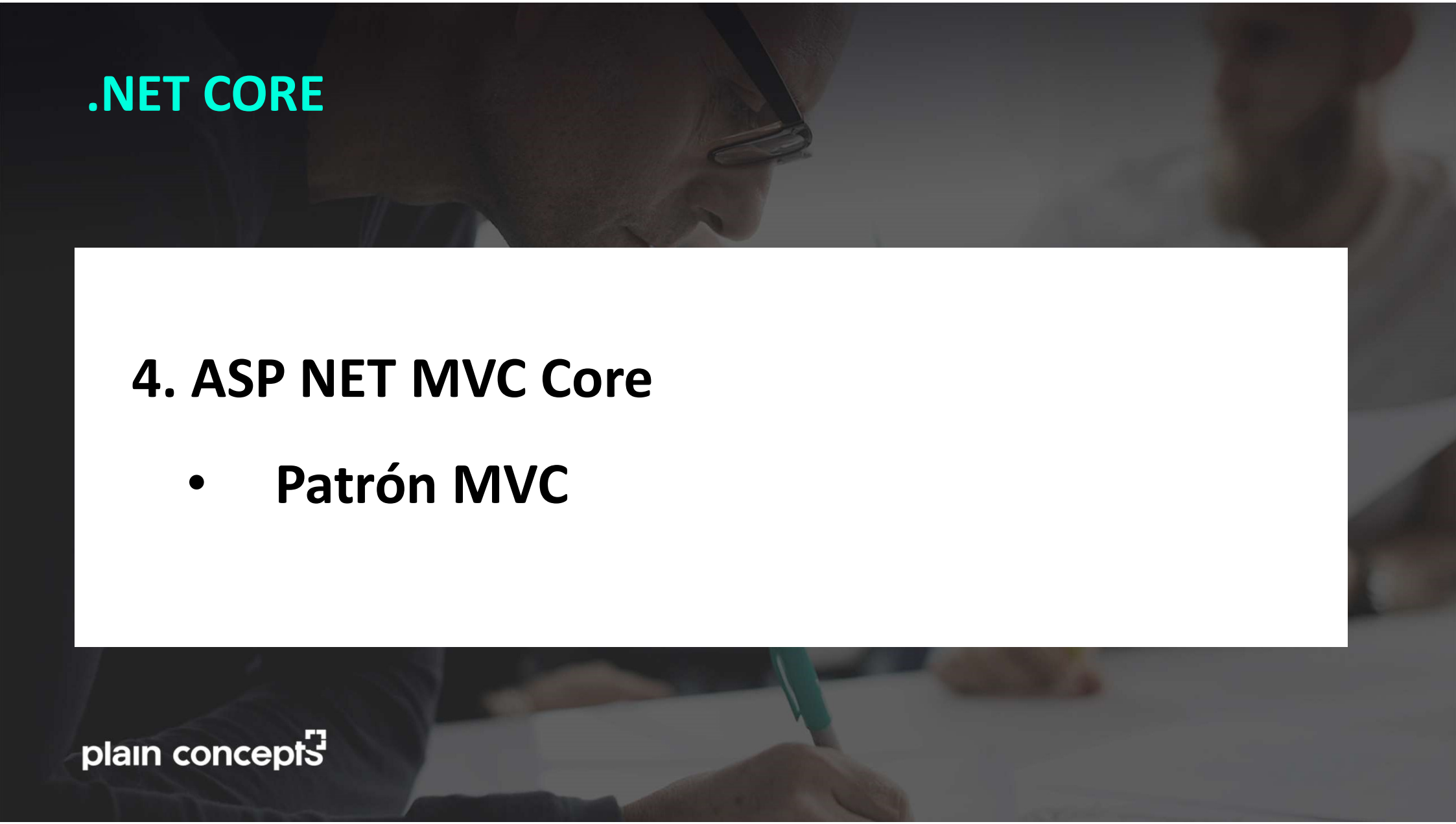
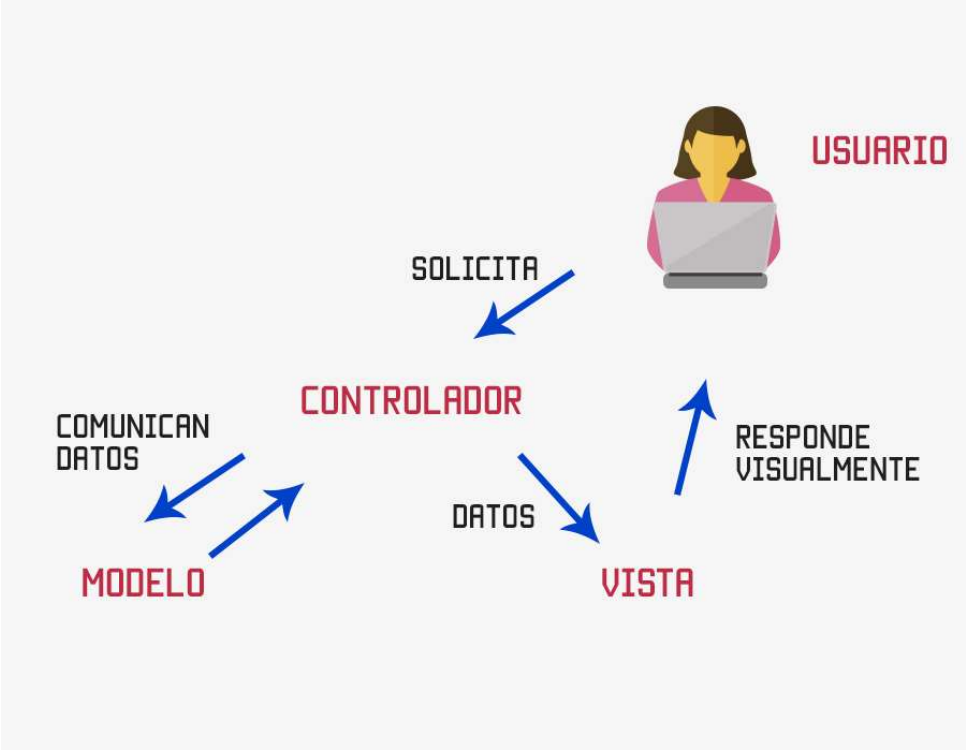

.NET CORE
4. ASP NET MVC Core
• Patrón MVC
plain concepts

ASP NET MVC Core

Patrón MVC

- Su fundamento es la separación del código en tres capas diferentes, acotadas por su responsabilidad
  - M = Modelo: Es la capa donde se definen los datos.
  - V = Vista: Es la visualización de nuestra respuesta.
  - C = Controlador. Contiene el código necesario para responder a las acciones que se solicitan en la aplicación



ASP NET MVC Core

ASP.NET MVC

- Para configurar todos los servicios de MVC hay que añadir los servicios de MVC con
  - AddMvc() en ConfigureServices
  - UseMvc() en Configure

```
1 reference
public class Startup
{
    0 references | 0 exceptions
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();
    }

    0 references | 0 exceptions
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseMvc();
    }
}
```

ASP NET MVC Core

Controladores

- Los controladores son la C de MVC. Son la entrada a nuestra API. Tienen que cumplir estas reglas.
  - Deben heredar de la clase ControllerBase de ASP NET CORE MVC.
  - Para añadirlos como servicios en Startup hay que usar el método AddMvc() en ConfigureServices.
  - Para usarlos hay que usar el UseMvc en Configure en Startup.
  - Tienen que tener una ruta que le indicamos con el Atributo Route.

```
1 reference
public class Startup
{
    0 references | 0 exceptions
    public void ConfigureServices(
        IServiceCollection services)
    {
        services.AddMvc();
    }

    0 references | 0 exceptions
    public void Configure(IApplicationBuilder app)
    {
        app.UseMvc();
    }
}
```

```
[Route("products")]
0 references
public class ProductsController : ControllerBase
{
    [HttpGet]
    0 references | 0 requests | 0 exceptions
    public async Task<ActionResult> GetProduct()
    {
        return Ok( new Product
        {
            Id = 1,
            Name = "Producto1",
            Price = 15
        });
    }
}
```

ASP NET MVC Core

Controladores

- Todos los métodos de los controladores tienen que devolver un objeto ActionResult
- Para devolver el código de estado que queramos podemos usar los métodos que hay en la clase ControllerBase, que devuelven un ActionResult. (Ok(), NotFound(), ServerError(), etc.)

```
[Route("products")]
0 references
public class ProductsController : ControllerBase
{
    [HttpGet("{productId}")]
    0 references | 0 requests | 0 exceptions
    public async Task<ActionResult> GetProduct([FromRoute] int productId)
    {
        if (productId <= 0)
        {
            return NotFound();
        }

        return Ok(new Product(productId, "Prueba", 12));
    }
}
```

ASP NET MVC Core

Controladores

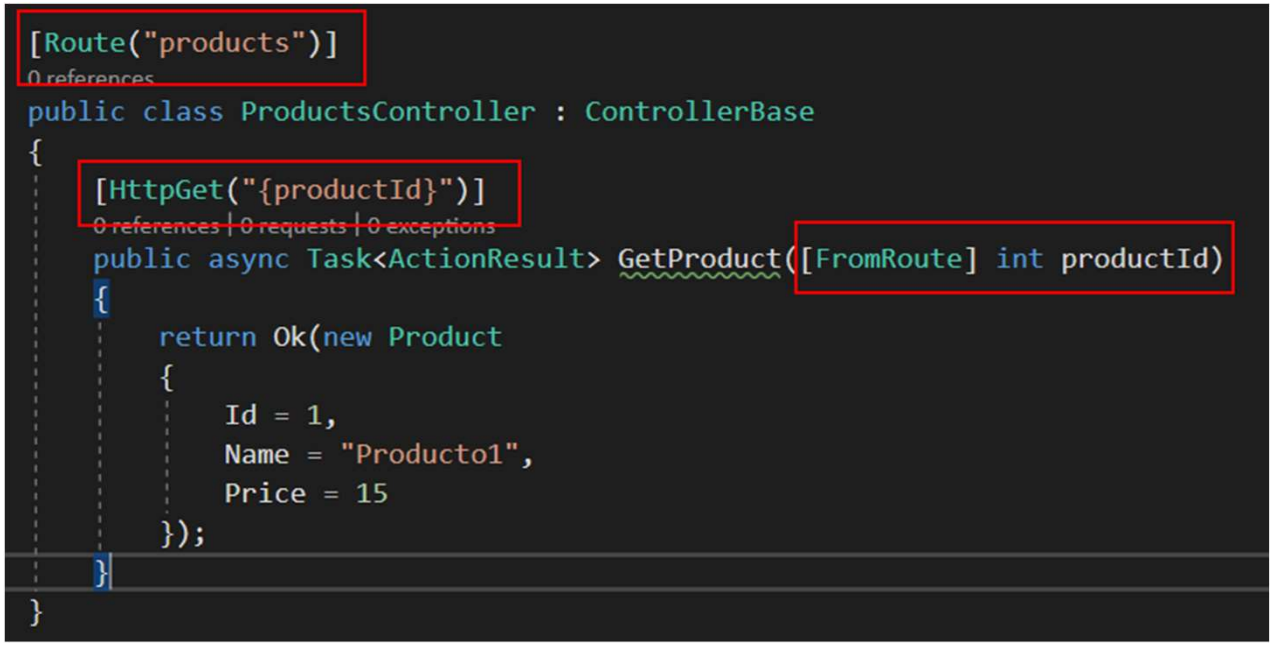
- Ejemplo de un Post

```
[Route("products")]
0 references
public class ProductsController : ControllerBase
{
    [HttpPost]
    0 references | 0 requests | 0 exceptions
    public async Task<ActionResult> CreateProduct([FromBody] Product product)
    {
        if (string.IsNullOrEmpty(product.Name))
        {
            return BadRequest();
        }

        var id = CreateProductBBDD(product);

        return Created($"/products/{id}", product);
    }

    Private Methods
}
```

ASP NET MVC Core

Enrutado

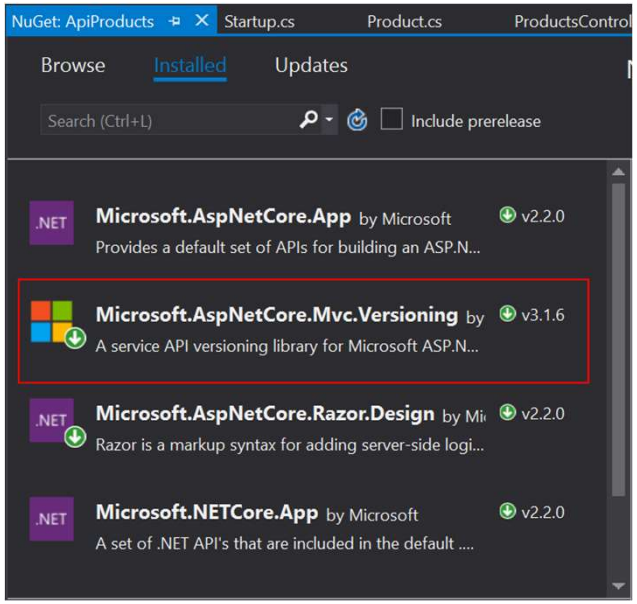
- La ruta principal de un controlador se pone en el atributo Route encima de la clase del mismo.
- La ruta de los métodos se puede poner junto al atributo del verbo. Así se puede definir el recurso completo.
- Los parámetros de ruta pueden ser de tres tipos:
  - Parámetros de ruta: Se usa el atributo [FromRoute] en los parámetros del método.
  - Parámetros del Query String. Se usa el atributo [FromQueryString]
  - Body. Para mapear el Body podemos usar el atributo [FromBody] en los parámetros del método.
  - Incluso para conseguir el valor de alguna cabecera se puede usar el atributo [FronHeader].



ASP NET MVC Core

Versionado

- Para poner el versionado hay que instalar el paquete Nuget: *Microsoft.AspNetCore.Mvc.Versioning*
- Después podemos registrar el versionado en nuestros servicios con: *AddApiVersioning*
- Por último, hay que añadir la versión al controlador para saber que versión usa.
  - Hay muchas configuraciones para la versión. En el ejemplo lo hemos hecho por url, pero también puede ir en el Query String o en una cabecera.



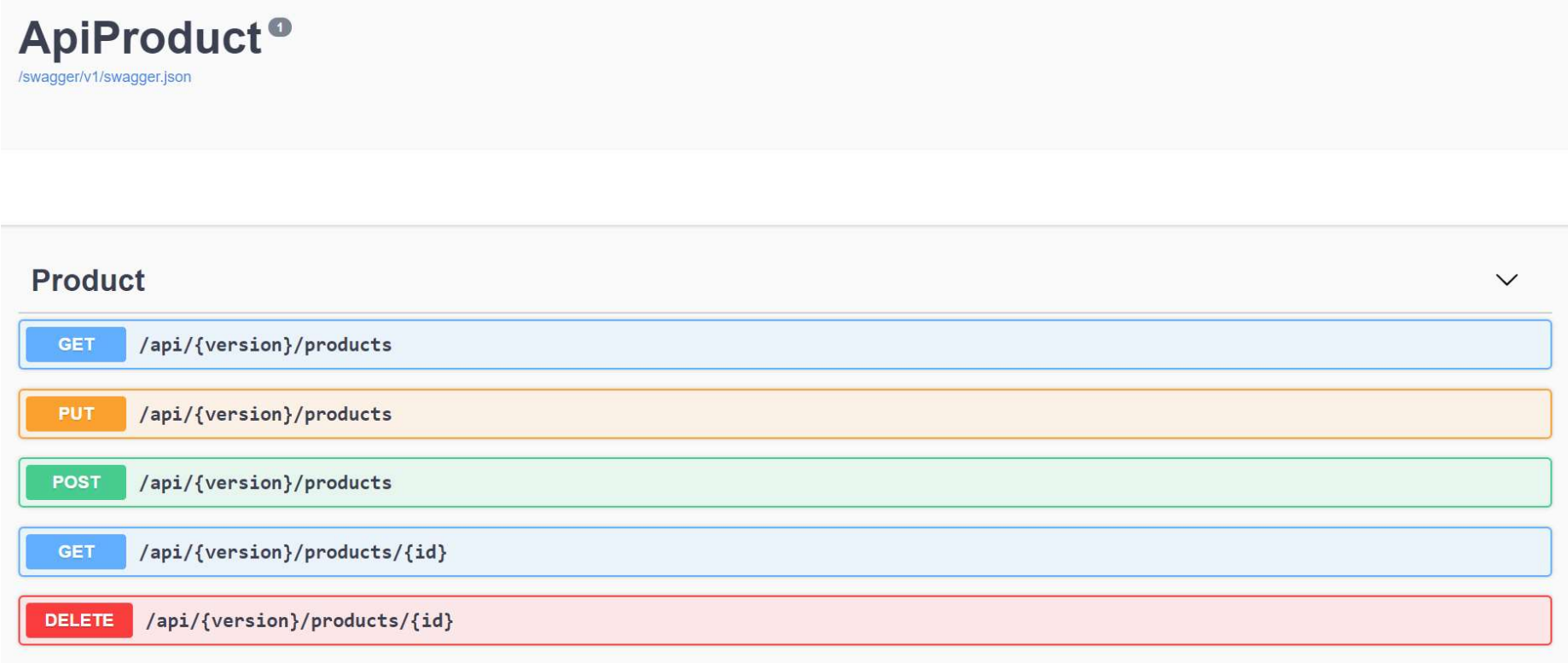

```
1 reference
public class Startup
{
    0 references | 0 exceptions
    public void ConfigureServices(
        IServiceCollection services)
    {
        services.AddApiVersioning(options =>
                {
                    options.ReportApiVersions = true;
                })
                .AddMvc();
    }

    0 references | 0 exceptions
    public void Configure(IApplicationBuilder app)
    {
        app.UseMvc();
    }
}
```

```
[Route("/api/{version:apiVersion}/products")]
[ApiVersion("1.0")]
0 references
public class ProductsController : ControllerBase
{
    [HttpPost]
    0 references | 0 requests | 0 exceptions
    public async Task<ActionResult> CreateProduct(
    {
        if (string.IsNullOrEmpty(product.Name))
        {
            return BadRequest();
        }

        var id = CreateProductBBDD(product);
```

ASP NET MVC Core

Swagger

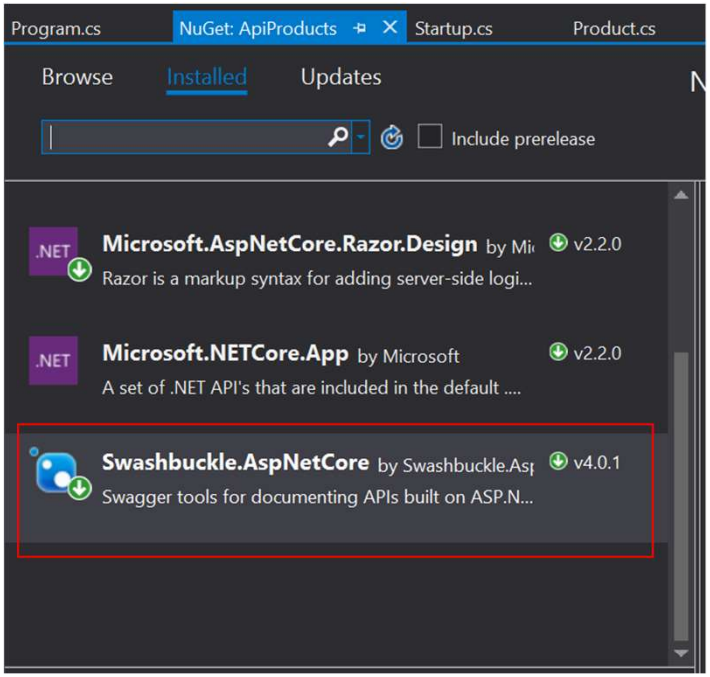
- Swagger es una herramienta que nos permite ver el API y poder consumirlo directamente desde la web.



ASP NET MVC Core

Swagger

- Para configurar Swagger:
  - Añadimos el paquete Nuget: Swashbuckle.AspNetCore
  - Añadimos los servicios en el ConfigureServices (Startup) con *AddSwaggerGen*
  - Añadimos el pipeline (Startup-Configure) de Swagger con *UseSwagger* y *UseSwaggerUI*




```
0 references | 0 exceptions
public void ConfigureServices(IServiceCollection services)
{
    services.AddApiVersioning(options =>
            {
                options.ReportApiVersions = true;
            })
            .AddSwaggerGen(gen =>
            {
                gen.SwaggerDoc("v1",
                    new Info
                    {
                        Title = "ApiProduct",
                        Version = "1"
                    });
            })
            .AddMvc();
}
```

```
0 references
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    app.UseSwagger()
       .UseSwaggerUI(c =>
       {
           c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");
           c.RoutePrefix = "";
       })
       .UseMvc();
}
```

ASP NET MVC Core

Swagger

- Se pueden configurar las llamadas a Swagger poniendo atributos en los métodos del Controlador.
  - El atributo *ProducesResponseType* nos indica las respuestas que nos puede recibir ese método.

```
0 references
public class ProductsController : ControllerBase
{
    [HttpPost]
    [ProducesResponseType(typeof(Product), (int)HttpStatusCode.Created)]
    [ProducesResponseType((int)HttpStatusCode.BadRequest)]
    0 references | 0 requests | 0 exceptions
    public async Task<ActionResult> CreateProduct([FromBody] Product product)
    {
        if (string.IsNullOrEmpty(product.Name))
        {
            return BadRequest();
        }

        var id = CreateProductBBDD(product);

        return Created($"/products/{id}", product);
    }
```