Meetup

Entendiendo Minimal APIs en una hora

17 marzo | 19.00h | Espacio Colaborativo

Madrid

Únete a la Comunidad

Aitor García

Ismael Sánchez

Speakers

Ismael Sánchez Chaves
Solution Architect at atmira

https://www.linkedin.com/in/ismael-sanchez-chaves/

ismael.sanchez.chaves@gmail.com

@rebeldecuantico

Aitor García Miranda
Application Architect at atmira

https://www.linkedin.com/in/aitor-garcia-miranda/

aitor.garcia.miranda@gmail.com

@water_fly

Gracias al espónsor

atmira

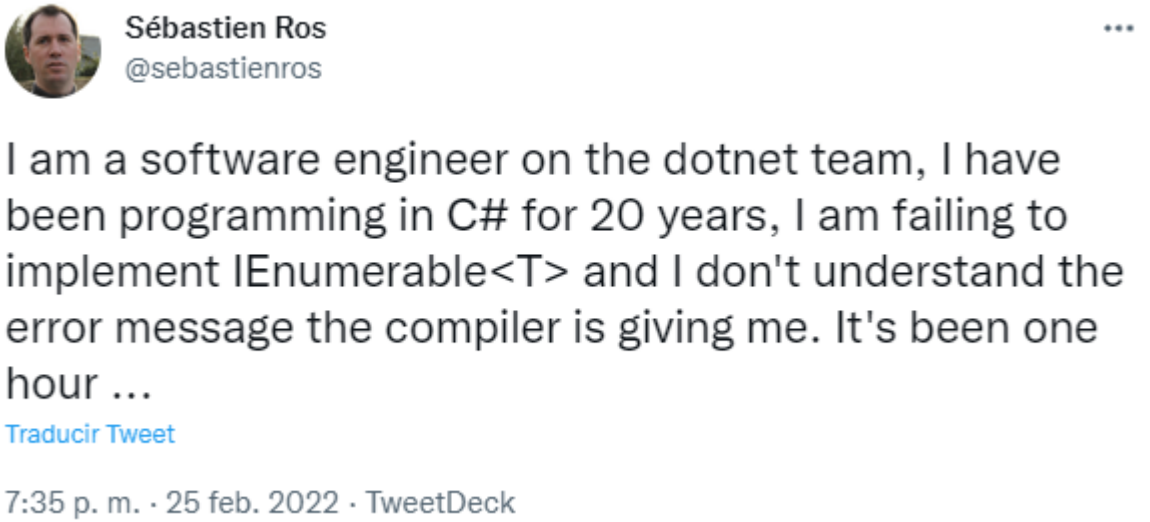
Comunidad **#MicUP**

¿Por qué una comunidad?

- Conecta gente
- Genera discusión
- Estimula el aprendizaje
- Promueve un entorno colaborativo
- Innova

#MicUP

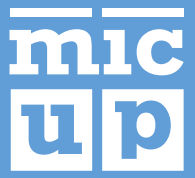
¿Quién se ha sentido así alguna vez?



Comunidad MicUP

Comunidad **#MicUP**

¿Qué son las minimal APIs?



.NET 6 trae novedades

Una nueva forma de hacer APIs

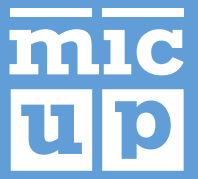
Gracias a .NET 6 y C# 10, tenemos una nueva funcionalidad para crear APIs con muy poco código.

Es la aplicación del Top level programs que ya traía .NET 5 aplicado a un API

Creación proyecto:

```
dotnet new web -o MinimalApi
```

Configuración de endpoints



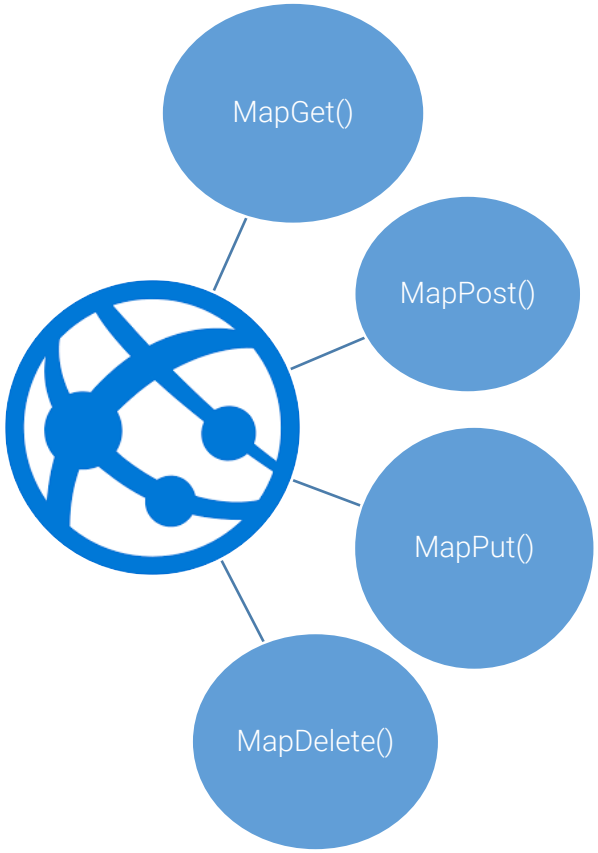
Verbos

.MapVerb()

Así de sencillo

Los métodos reciben un string con
la ruta del endpoint y un delegado

```
public static RouteHandlerBuilder MapGet(
    this IEndpointRouteBuilder endpoints,
    string pattern,
    Delegate handler)
```



Con una expresión lambda daremos
la lógica de nuestro endpoint

```
app.MapGet("/", () => "Hello World!");
```

Async

Expresiones lambda

Podemos complicar mas la lógica de nuestros métodos, y por supuesto, podemos tener una respuesta asíncrona

```
app.MapGet("/getasync", async () =>
{
    HttpClient client = new HttpClient();
    var response = await client.GetAsync("https://jsonplaceholder.typicode.com/todos/1");
    response.EnsureSuccessStatusCode();
    return response.Content.ReadAsStringAsync();
});
```

Es tan sencillo como hacer la expresión lambda asíncrona, añadiendo *async*, y hacer uso del operador *await* para esperar la respuesta, en este caso de un api -> https://jsonplaceholder.typicode.com/

Inyección de dependencias

Poco cambio al respecto

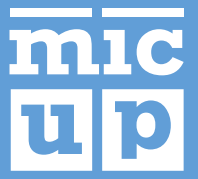
Al igual que anteriormente lo hacíamos en Startup:

```
builder.Services.AddScoped<IService, Service>();

builder.Services.AddSingleton<IService, Service>();

builder.Services.AddTransient<IService, Service>();
```

Open API y autorización



Swagger

Open API

Instalamos el paquete *Swashbuckle.AspNetCore*:

```
dotnet add MinimalApi.csproj package Swashbuckle.AspNetCore
```

Servicios

```
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
```

Middleware

```
app.UseSwagger();
app.UseSwaggerUI();
```

Documentación con Swagger

Podemos añadir algo de documentación a nuestros endpoints.

Pero no es tan completa como si explotamos la documentación en xml desde swagger.

```
public static class OpenApiRouteHandlerBuilderExtensions
{
    public static RouteHandlerBuilder ExcludeFromDescription(this Rou
    public static RouteHandlerBuilder Produces<TResponse>(this Route
    public static RouteHandlerBuilder Produces(this RouteHandlerBuil
    public static RouteHandlerBuilder ProducesProblem(this RouteHandl
    public static RouteHandlerBuilder ProducesValidationProblem(this
    public static RouteHandlerBuilder WithTags(this RouteHandlerBuil
    public static RouteHandlerBuilder Accepts<TRequest>(this RouteHa
    public static RouteHandlerBuilder Accepts<TRequest>(this RouteHa
    public static RouteHandlerBuilder Accepts(this RouteHandlerBuild
    public static RouteHandlerBuilder Accepts(this RouteHandlerBuild
}
```

Autorización y Autenticación

Podemos añadir como es habitual, los servicios y middlewares correspondientes a la autorización y autenticación:

Servicios

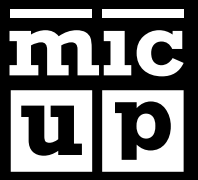
```
#region Authorization & Authentication
builder.Services.AddAuthentication("Bearer")
    .AddJwtBearer("Bearer", options =>
    {
        options.Authority = "https://localhost:5001";

        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateAudience = false
        };
    });
builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("ApiScope", policy =>
    {
        policy.RequireAuthenticatedUser();
        policy.RequireClaim("scope", "api1");
    });
});
#endregion
```

Middleware

```
app.UseAuthentication();
app.UseAuthorization();
```

Let's code



Conclusiones

Pros:

- Sencillo
- Rápido (RPS)
- Poco Desarrollo

Contras:

- Un solo “controlador”
- SOLID
- Muy específico

Muchas Gracias!!!

¿Preguntas?

El código completo con mas ejemplos, esta disponible en:
https://github.com/atmiraio/minimal-apis-sample.git