

API REST en ASP

- Web API con ASP Net Core
- Controladores
- Ruteo de controladores
- Solicitud HTTP – Pasar parámetros a un controlador



NET 7+

Web API con ASP Net Core

Las siglas ASP significan Active Server Pages. ASP es una tecnología de programación desarrollada por Microsoft como parte de la plataforma ASP.NET. Se utiliza para crear aplicaciones web dinámicas y sitios web interactivos.

ASP permite la ejecución de código del lado del servidor para generar contenido web personalizado antes de ser enviado al navegador del usuario.



Web API con ASP Net Core

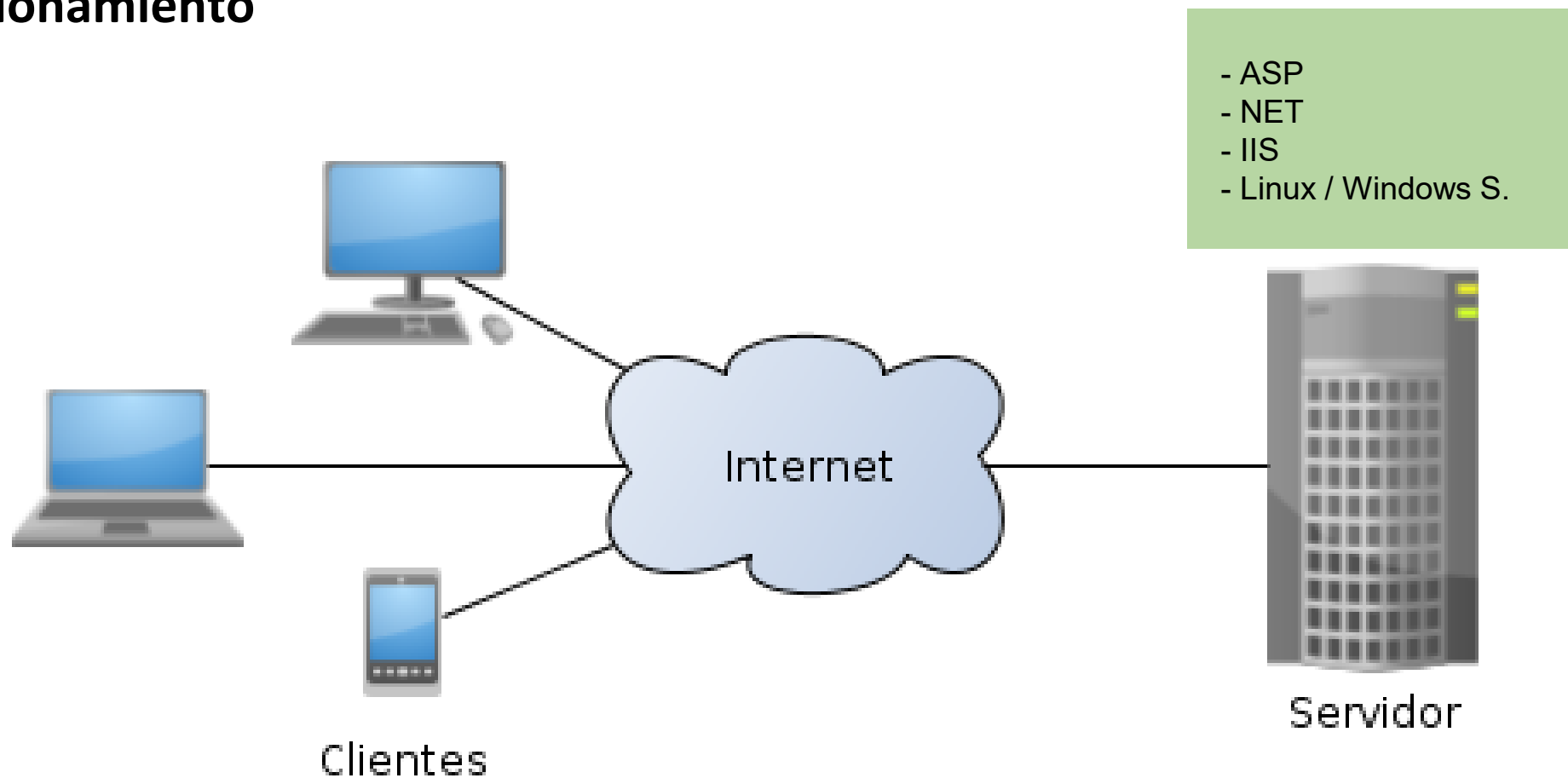
Al día de hoy con ASP.NET Core contamos con dos posibilidades para crear WEB APIs

- WebAPI basadas en controladores (heredan de ControllerBase)
- WebAPI sin controladores (mínimal API)



Web API con ASP Net Core

Funcionamiento



Web API con ASP Net Core

Opciones de WebAPI

WebAPI basadas en controladores (heredan de ControllerBase)

Proponen un enfoque más extenso y robusto para crear APIs HTTP con ASP.NET Core. Permite construir end-points REST completamente funcionales utiliza el scaffolding tradicional de ASP.Net Core y el aportan el uso controlador para la separación lógica de responsabilidades y funcionalidades.

WebAPI sin controladores (minimal API)

Proponen un enfoque simplificado para crear rápidamente APIs HTTP con ASP.NET Core. Puedes construir end-points REST completamente funcionales con un código y configuración mínimos. Evita el scaffolding tradicional y los controladores innecesarios al declarar de manera fluida las rutas y acciones de la API.

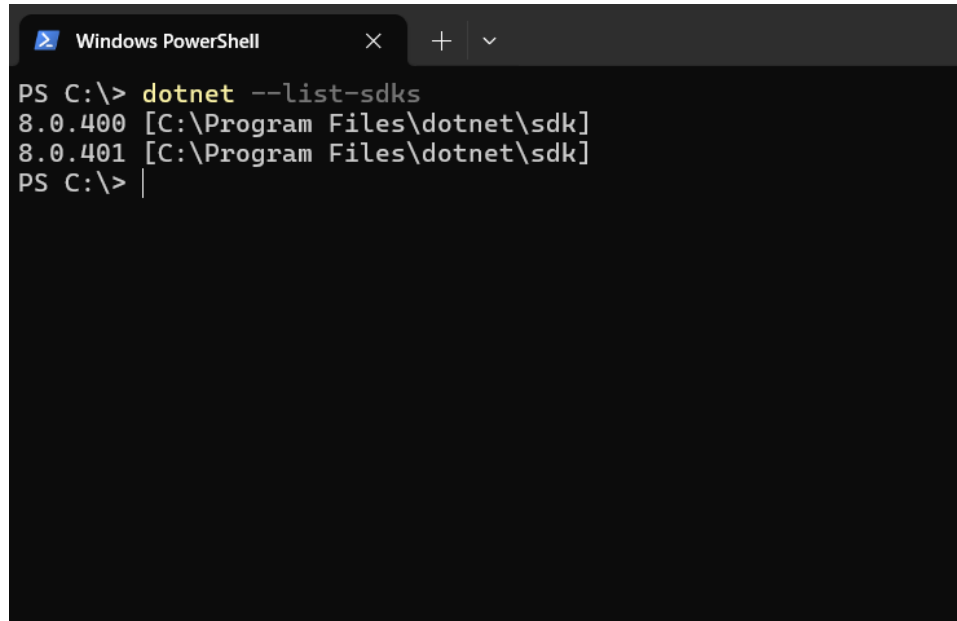
Web API con ASP Net Core

Crear una nuevo proyecto webAPI

Comandos generales:

- `dotnet --list-sdks`

Lista los SDKs Instalados



```
Windows PowerShell
PS C:\> dotnet --list-sdks
8.0.400 [C:\Program Files\dotnet\sdk]
8.0.401 [C:\Program Files\dotnet\sdk]
PS C:\> |
```

Listado de SDKs .Net instalados

Web API con ASP Net Core

Crear una nuevo proyecto webAPI

Comandos generales:

- `dotnet new --list`

Devuelve el listado de templates que se pueden crear con el SDK

```
PS C:\> dotnet new --list
Advertencia: el uso de 'dotnet new --list' está en desuso. Use 'dotnet new list' en su lugar.
Para más información, ejecute:
    dotnet new list -h

Estas plantillas coinciden con su entrada:
```

Nombre de la plantilla	Nombre corto	Idioma	Etiquetas
.NET MAUI ContentPage (C#)	maui-page-csharp	[C#]	MAUI/Android/iOS/macOS/Mac Catalyst/WinUI/Tizen/Xaml/Code
.NET MAUI ContentPage (XAML)	maui-page-xaml	[C#]	MAUI/Android/iOS/macOS/Mac Catalyst/WinUI/Tizen/Xaml/Code
.NET MAUI ContentView (C#)	maui-view-csharp	[C#]	MAUI/Android/iOS/macOS/Mac Catalyst/WinUI/Tizen/Xaml/Code
.NET MAUI ContentView (XAML)	maui-view-xaml	[C#]	MAUI/Android/iOS/macOS/Mac Catalyst/WinUI/Tizen/Xaml/Code
.NET MAUI Multi-Project App	maui-multiproject	[C#]	MAUI/Android/iOS/macOS/Mac Catalyst/Windows/Mobile
.NET MAUI ResourceDictionary (XAML)	maui-dict-xaml	[C#]	MAUI/Android/iOS/macOS/Mac Catalyst/WinUI/Xaml/Code
API web ASP.NET Core (native AOT)	webapiot	[C#]	Web/Web API/API/Service
Aplicación .NET MAUI	maui	[C#]	MAUI/Android/iOS/macOS/Mac Catalyst/Windows/Tizen/Mobile
Aplicación .NET MAUI Blazor	maui-blazor	[C#]	MAUI/Android/iOS/macOS/Mac Catalyst/Windows/Tizen/Blazor/Blazor Hybrid/Mobile
Aplicación Android Wear	androidwear	[C#]	Android/Mobile
Aplicación Blazor para WebAssembly	blazorwasm	[C#]	Web/Blazor/WebAssembly/PWA
Aplicación con pestañas de iOS	ios-tabbed	[C#]	iOS/Mobile
Aplicación de Android	android	[C#]	Android/Mobile
Aplicación de consola	console	[C#], [F#], [VB]	Common/Console
Aplicación de inicio de .NET Aspire	aspire-starter	[C#]	Common/.NET Aspire/Blazor/Web/Web API/API/Service/Cloud
Aplicación de iOS	ios	[C#], [F#], [VB]	iOS/Mobile
Aplicación de Windows Forms	winforms	[C#], [VB]	Common/WinForms
Aplicación Mac Catalyst	maccatalyst	[C#], [VB]	macOS/Mac Catalyst
Aplicación vacía de .NET Aspire	aspire	[C#]	Common/.NET Aspire/Cloud/Web/Web API/API/Service
Aplicación web Blazor	blazor	[C#]	Web/Blazor/WebAssembly
Aplicación web de ASP.NET Core	webapp, razor	[C#]	Web/MVC/Razor Pages
Aplicación web de ASP.NET Core (Modelo-Vista-Controlador)	mvc	[C#], [F#]	Web/MVC
Aplicación WPF	wpf	[C#], [VB]	Common/WPF
Archivo de búfer de protocolo	proto		Web/gRPC
Archivo de la solución	sln, solution		Solution
Archivo de manifiesto de la herramienta local de dotnet	tool-manifest		Config
Archivo Directory.Build.props de MSBuild	buildprops		MSBuild/props
Archivo Directory.Build.targets de MSBuild	buildtargets		MSBuild/props
Archivo EditorConfig	editorconfig, editorconfig		Config
archivo gitignore de dotnet	gitignore, gitignore		Config
archivo global.json	globaljson, global.json		Config
ASP.NET Core vacío	web	[C#], [F#]	Web/Empty
ASP.NET Core Web API	webapi	[C#], [F#]	Web/WebAPI/Web API/API/Service
ASP.NET Core with Angular	angular	[C#]	Web/MVC/SPA
ASP.NET Core with React.js	react	[C#]	Web/MVC/SPA
Biblioteca de clases	classlib	[C#], [F#], [VB]	Common/Library
Biblioteca de clases .NET MAUI	maui.lib	[C#]	MAUI/Android/iOS/macOS/Mac Catalyst/Windows/Tizen/Mobile
Biblioteca de clases de Android	androidlib	[C#]	Android/Mobile

Listado de templates de proyectos en .Net

Web API con ASP Net Core

Crear una nuevo proyecto webAPI

Comandos para crear y abrir un proyecto WebAPI basado en controladores:

- `dotnet new webapi -n MiWebAPI`
- `dotnet new webapi --use-controllers -o MiWebAPI`
- `code -r MiWebAPI`

← Crea un nuevo proyecto API
Con versión 7 del SDK

← Crea un nuevo proyecto API
Con versión 8+ del SDK

← Abre Vs code desde la
terminal con el proyecto

```
Windows PowerShell
PS C:\Repositorio\taller 2> dotnet new webapi -n MiWebAPI
La plantilla "ASP.NET Core Web API" se creó correctamente.

Procesando acciones posteriores a la creación...
Restaurando C:\Repositorio\taller 2\MiWebAPI\MiWebAPI.csproj:
  Determinando los proyectos que se van a restaurar...
  Se ha restaurado C:\Repositorio\taller 2\MiWebAPI\MiWebAPI.csproj (en 205 ms).
Restauración realizada correctamente.

PS C:\Repositorio\taller 2> |
```

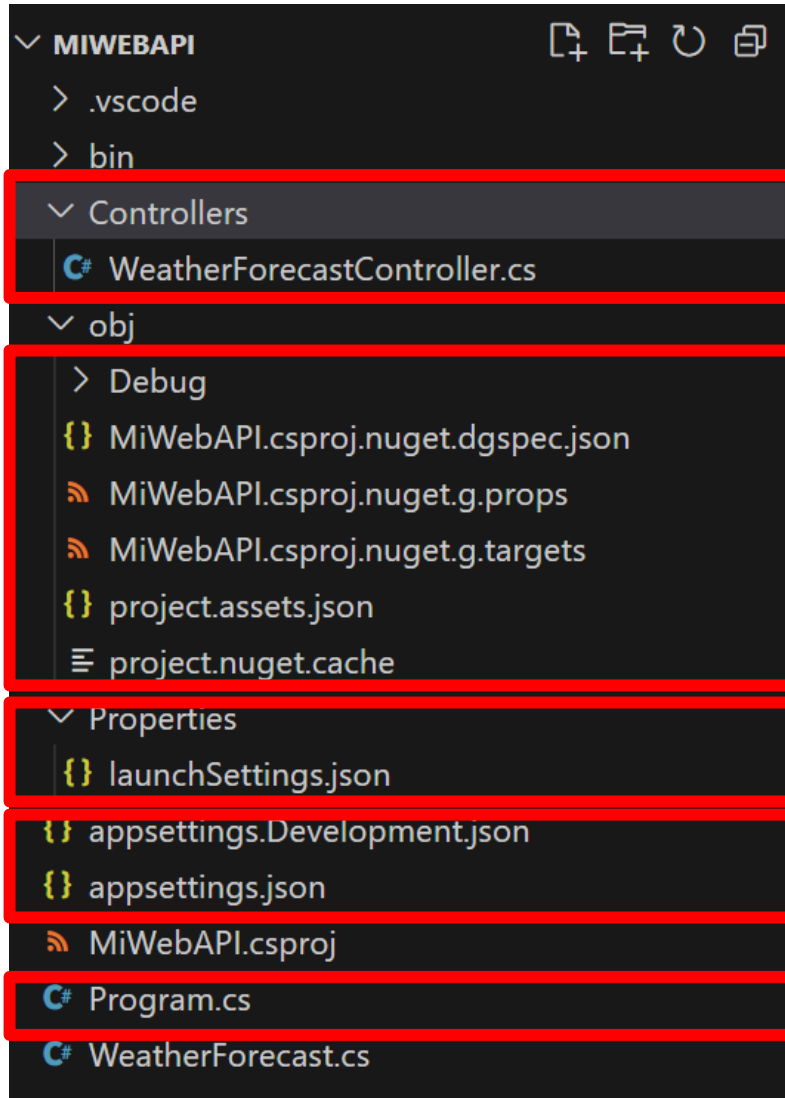
Web API con ASP Net Core

Archivos WebAPI para un proyecto basado en controllers

This PC > New Volume (D:) > WebAPI > MyWebAPI					
	Name	Date modified	Type	Size	
★	Controllers	24-06-2022 11:57	File folder		
★	obj	24-06-2022 11:57	File folder		
★	Properties	24-06-2022 11:57	File folder		
★	appsettings.Development	24-06-2022 11:57	JSON File	1 KB	
★	appsettings	24-06-2022 11:57	JSON File	1 KB	
	MyWebAPI.csproj	24-06-2022 11:57	C# Project File	1 KB	
ntialing	Program.cs	24-06-2022 11:57	C# Source File	1 KB	
sonal	WeatherForecast.cs	24-06-2022 11:57	C# Source File	1 KB	

Web API con ASP Net Core

Archivos webAPI para un proyecto basado en controllers



Controladores: aquí encontramos los diferentes controladores del proyecto

Dependencias de nuget

Archivos de configuración del proyecto

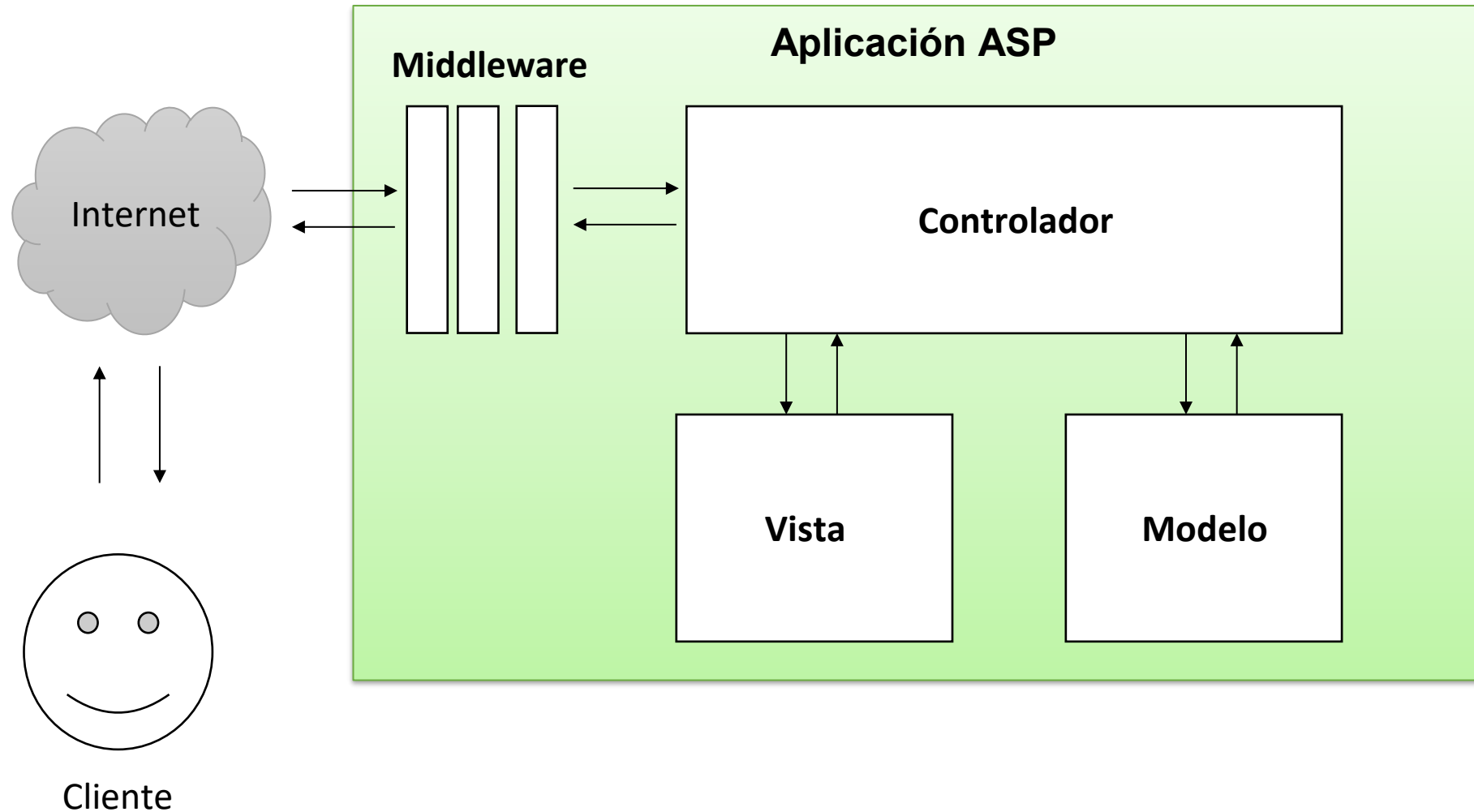
Rutas de recursos y configuraciones

Json de configuración del proyecto

Procesando Request

Controladores

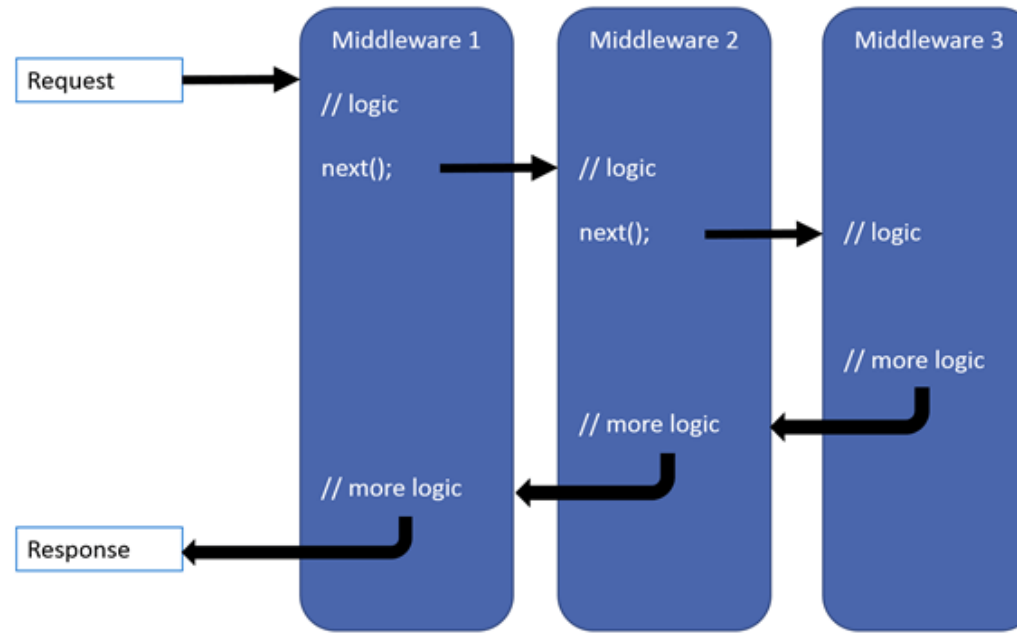
Funcionamiento – Request / Response



MiddleWare

Definición Básica

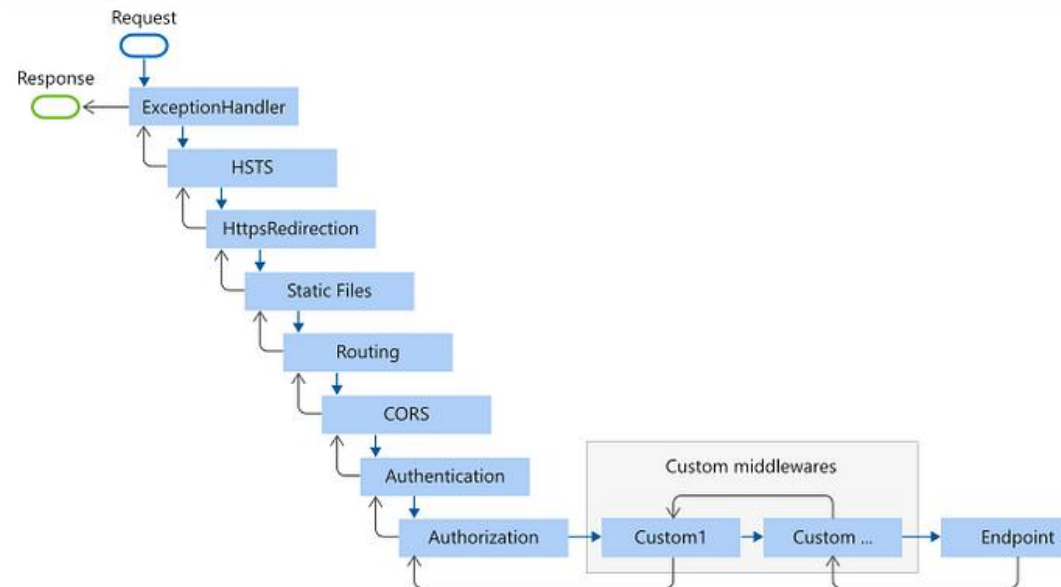
Middleware es un concepto esencial en ASP.NET que hace posible la personalización y el procesamiento de solicitudes HTTP de manera modular y organizada en las aplicaciones web.



MiddleWare

Funcionamiento

- Los middlewares se organizan en una cadena o "pipeline" de procesamiento. Cuando una solicitud HTTP entra en la aplicación, pasa secuencialmente a través de esta cadena de middlewares.
- Cada middleware en la cadena tiene la oportunidad de procesar la solicitud, tomar decisiones y modificar la respuesta antes de que pase al siguiente middleware o se envíe una respuesta al cliente.



MiddleWare

Orden y Funcionalidad

Los middlewares se ejecutan en un orden específico, lo que permite una personalización y procesamiento gradual de las solicitudes. Cada middleware realiza una tarea específica.

Algunos ejemplos comunes incluyen:

- Autenticación
- Autorización
- Enrutamiento
- Registro compresión y manipulación de encabezados HTTP.

MiddleWare

Implementación

```
Program.cs
1 var builder = WebApplication.CreateBuilder(args);
2 // Add services to the container.
3 builder.Services.AddControllers();
4 // Learn more about configuring Swagger/OpenAPI at https
5 builder.Services.AddEndpointsApiExplorer();
6 builder.Services.AddSwaggerGen();
7
8 var app = builder.Build();
9
10 // Configure the HTTP request pipeline.
11 if (app.Environment.IsDevelopment())
12 {
13     app.UseSwagger();
14     app.UseSwaggerUI();
15 }
16
17 app.UseHttpsRedirection();
18
19 app.UseAuthorization();
20
21 app.MapControllers();
22
23
24 app.Run();
25
```

Archivo Program.cs

Agrego middlewares al pipeline

Construyo el objeto Aplicación

Determino los elementos que se
usarán en el pipe de la aplicación

Aquí ejecuto efectivamente la
Aplicación web como fue seteada

Controladores

Que es un controlador en ASP

En ASP.NET Web API, un controlador es una clase que controla las solicitudes HTTP. Los métodos públicos del controlador se denominan **métodos de acción** o simplemente **acciones**. Cuando el marco de api web recibe una solicitud, enruta la solicitud a una acción.

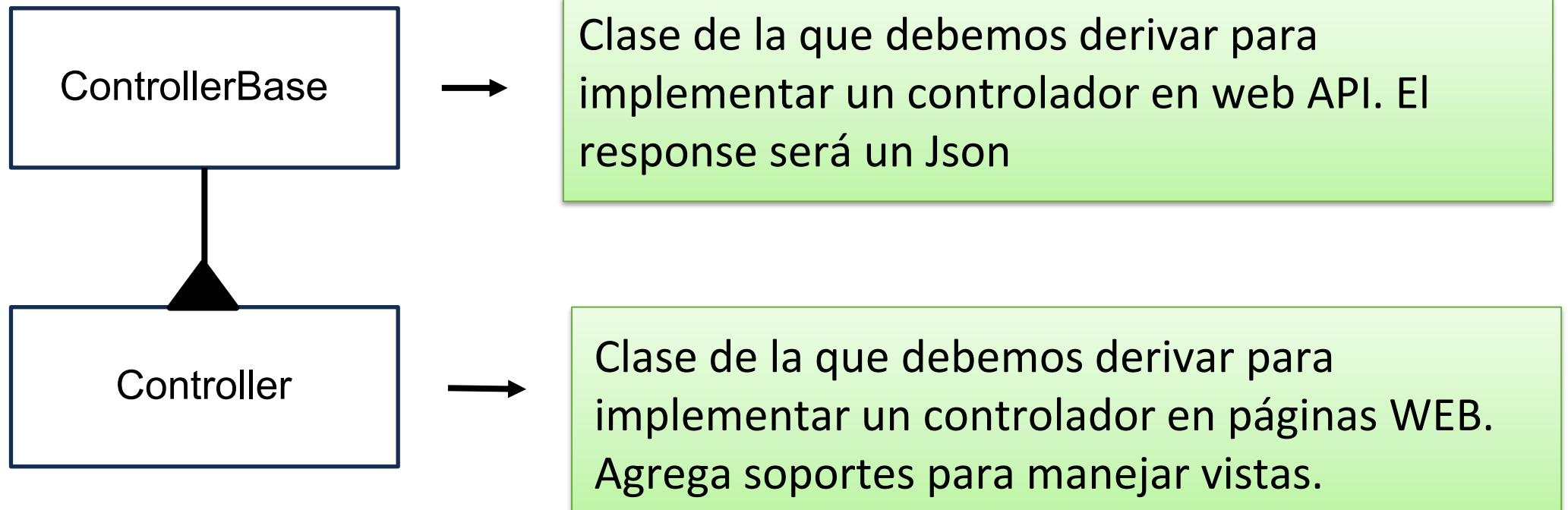
Son Clases qué:

- Se pueden marcar con de atributos [ApiController]
- Heredan de la Clase BaseController o Controller
- Es el punto de entrada para las solicitudes HTTP.
- Emiten las respuestas para las respuestas HTTP

Controladores

Clase ControllerBase

Una API web basada en controladores consta de una o más clases de controladores que derivan de **ControllerBase**.



Notas:

Si el mismo controlador debe admitir vistas y API web, entonces debe derivar de Controller.

Controladores

Clase ControllerBase

```
namespace Microsoft.AspNetCore.Mvc
{
    ...public abstract class ControllerBase
    {
        protected ControllerBase();

        ...public HttpResponseMessage Response { get; }
        ...public HttpRequest Request { get; }
        ...public HttpContext HttpContext { get; }
        ...public ControllerContext ControllerContext { get; set; }
        ...public IModelMetadataProvider MetadataProvider { get; set; }
        ...public IModelBinderFactory ModelBinderFactory { get; set; }
        ...public IUrlHelper Url { get; set; }
        ...public ClaimsPrincipal User { get; }
        ...public ProblemDetailsFactory ProblemDetailsFactory { get; set; }
        ...public IObjectModelValidator ObjectValidator { get; set; }
        ...public RouteData RouteData { get; }
        ...public ModelStateDictionary ModelState { get; }

        ...public virtual AcceptedResult Accepted(Uri uri, [ActionResultObjectValue] object value);
        ...public virtual AcceptedResult Accepted(string uri, [ActionResultObjectValue] object value);
        ...public virtual AcceptedResult Accepted(Uri uri);
        ...public virtual AcceptedResult Accepted([ActionResultObjectValue] object value);
        ...public virtual AcceptedResult Accepted();
        ...public virtual AcceptedResult Accepted(string uri);
        ...public virtual AcceptedAtActionResult AcceptedAtAction(string actionName, string controllerName, object routeValues, [ActionResultObjectValue] object value);
        ...public virtual AcceptedAtActionResult AcceptedAtAction(string actionName, object routeValues, [ActionResultObjectValue] object value);
        ...public virtual AcceptedAtActionResult AcceptedAtAction(string actionName);
        ...public virtual AcceptedAtActionResult AcceptedAtAction(string actionName, string controllerName);
        ...public virtual AcceptedAtActionResult AcceptedAtAction(string actionName, [ActionResultObjectValue] object value);
        ...public virtual AcceptedAtActionResult AcceptedAtAction(string actionName, string controllerName, [ActionResultObjectValue] object routeValues);
        ...public virtual AcceptedAtRouteResult AcceptedAtRoute(string routeName);
        ...public virtual AcceptedAtRouteResult AcceptedAtRoute(object routeValues, [ActionResultObjectValue] object value);
        ...public virtual AcceptedAtRouteResult AcceptedAtRoute(string routeName, object routeValues, [ActionResultObjectValue] object value);
        ...public virtual AcceptedAtRouteResult AcceptedAtRoute(string routeName, object routeValues);
        ...public virtual AcceptedAtRouteResult AcceptedAtRoute([ActionResultObjectValue] object routeValues);
        ...public virtual BadRequestResult BadRequest();
    }
}
```

Controladores

Ruteo de recursos

Hay algunas formas de controlar el enrutamiento en una aplicación ASP.NET Core, nos concentraremos en las dos formas más comunes:

Enrutamiento convencional: La ruta se determina en función de convenciones que se definen en plantillas de ruta que, en tiempo de ejecución, mapearán las solicitudes a controladores (clases) y acciones (métodos).

Enrutamiento basado en atributos: La ruta se determina en función de los atributos que establece en sus controladores y métodos. Estos atributos definirán la asignación a las acciones del controlador.

Ruteo

Ruteo de recursos – Convencional

Los controladores de ASP.NET Core utilizan el middleware Routing para hacer coincidir las URL de las solicitudes entrantes y asignarlas a acciones. Plantillas de ruta se definen al inicio en Program.cs o en atributos.

Forma convencional de rutear un recurso:

```
"{controller}/{action}/{id?}"
```

- El primer parámetro del path, {controller}, Mapea el controlador.
- El segundo parámetro de path, {action}, mapea el nombre de la acción vinculada.
- El tercer parámetro, {id?} es usado como parámetro opcional id. El ? en {id?} hace que sea opcional. id es usado para mapear al model de entidad.

Atributos - Attributes

Definición

En C#, los atributos son clases que se heredan de la clase base `Attribute`. Cualquier clase que se hereda de **`Attribute`** puede usarse como una especie de "etiqueta" en otros fragmentos de código.

¡Cuidado con las traducciones!

En algunas bibliografías en español, por una mala traducción se los denominó "decoradores" lo que es incorrecto. Forma de referirnos a ellos dentro del ámbito de C# es `Attribute` o `Atributo`.

Atributos - Attributes

Usos

Los atributos proporcionan un método eficaz para asociar metadatos, o información declarativa, con código (ensamblados, Clases, tipos, métodos, propiedades, etc.).

Se los utiliza colocando el nombre de la clase entre corchetes "[Atributo]" y anteponiéndolo a un fragmento de código.

Ejemplo de uso:

[Atributo]

```
public class MiNuevaClase{ //elementos de la clase... }
```

[Atributo]

```
public void Método{ //código de del método... }
```


Crear un nuevo proyecto MVC

Ruteo de recursos – Atributos

El Ruteo por atributos reescribe el ruteo convencional y aplica nuevas reglas de ruteo, esto lo hace mediante el uso de atributos, se puede aplicar estas nuevas rutas por medio de el atributo [ApiController] o [Route]. También puede usarse en los diferentes Métodos de acción.

El atributo [ApiController] debe aplicarse a sus controladores para identificarlos como controladores.

El atributo [Route] permite definir la ruta de acceso a dicho controlador.

Controladores

Controlador ejemplo con atributos

Aquí se puede ver como se declara una clase que hereda de la clase ControllerBase, y que tiene los atributos necesarios para que se la declare

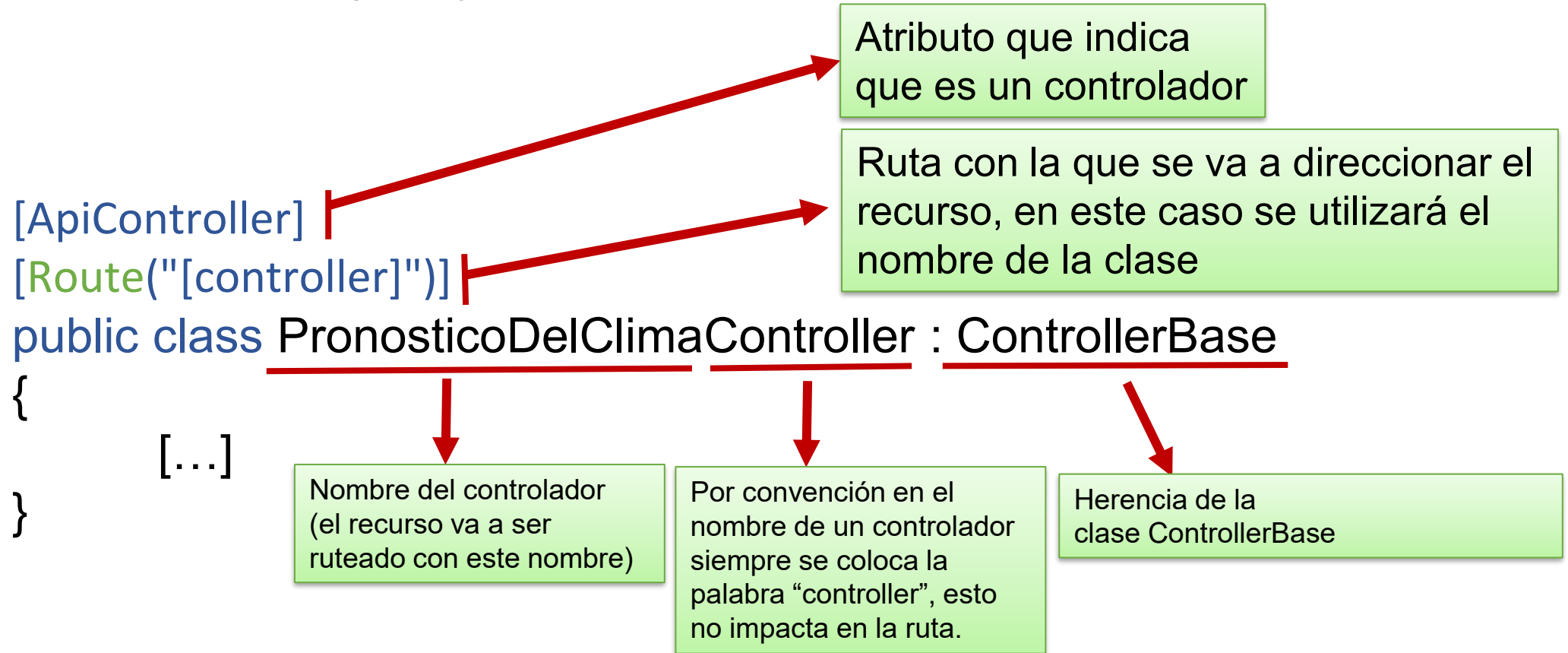
```
[ApiController]
[Route("[controller]")]
public class PronosticoDelClimaController : ControllerBase
{
    [...]
}
```

Ruta http para acceder al controlador en cuestión

<https://localhost:5001/PronosticoDelClima>

Controladores

Controlador ejemplo con atributos



Ruta http para acceder al controlador en cuestión

<https://localhost:5001/PronosticoDelClima>

Ruta del servidor

Controlador

Request HTTP

Atributos para Identificar un verbo HTTP en un Action

Atributo	Verbo
[HttpGet]	Get
[HttpPut]	Put
[HttpPost]	Post
[HttpDelete]	Delete
[HttpHead]	Head
[HttpOptions]	Options
[HttpPatch]	Patch

Request HTTP

Atributos para recuperar información de un request

Ejemplo de uso de atributos en ASP Net Core

Captura un Request
de tipo Post



`[HttpPost]`

```
public IActionResult Login(string email, string password)
{
    ...
}
```

Captura un Request
de tipo Get



`[HttpGet]`

```
public IActionResult GetPedidos(string IdCadete)
{
    ...
}
```

Request HTTP

Request ejemplo sin parámetros

El acceso al recurso quedaría determinado de la siguiente forma:

```
[ApiController]
```

```
[Route("[controller]")]
```

```
public class PronosticoDelClimaController : ControllerBase
```

```
{
```

```
    [HttpGet]
```

```
    public IActionResult GetClimaHoy()
```

```
    {
```

```
        //codigo que devuelve el clima
```

```
    }
```

```
}
```



Nuevo Método de
Acción

Ruta http para acceder al controlador en cuestión

<https://localhost:5001/PronosticoDelClima/GetClimaHoy>

Request HTTP

Método Get sin parámetros

[ApiController]

[Route("[controller]")]

public class **PronosticoDelClimaController** : **ControllerBase**

{

[HttpGet]

public IActionResult **GetClimaHoy()**

{

//codigo que devuelve el clima

}

}

Atributo que indica el verbo http de la solicitud

Action que devuelve un recurso

Nombre del controlador al que se accede

Nombre del action al que se accede

Ruta http para acceder al controlador en cuestión

https://localhost:5001/PronosticoDelClima/GetClimaHoy

Request HTTP

Método Get con 1 parámetro y reescribiendo la ruta

[ApiController]

[Route("[controller]")]

public class **PronosticoDelClimaController** : **ControllerBase**

{

[HttpGet("GetClima/{fecha}")]

public IActionResult **GetClima**(datetime fecha)

{

//codigo que devuelve el clima

}

}

Atributo que indica el verbo http de la solicitud

parámetro

Nombre del controlador al que se accede

Nombre del action al que se accede

Valor del parámetro

Ruta http para acceder al controlador en cuestión


https://localhost:5001/PronosticoDelClima/GetClima/22-11-2018

Request HTTP

Método Get con dos parámetros ruta convencional

https://localhost:{PORT}/Paciente/Buscar?edad=3&nombre=Carlos

```
[ApiController]
[Route("[controller]")]
public class PacienteController : ControllerBase
{
    [HttpGet("Buscar")]
    public IActionResult Buscar(int edad, string nombre)
    {
        //codigo que devuelve el clima
    }
}
```



Request HTTP

Método Get con 1 parámetros y reescribiendo la ruta

[ApiController]

[Route("[controller]")]

public class **PronosticoDelClimaController** : **ControllerBase**

{

[HttpGet("GetClima/{fecha}")]

public IActionResult **GetClima**(datetime fecha)

{

//codigo que devuelve el clima

}

}

Atributo que indica el verbo http de la solicitud

parámetro

Nombre del controlador al que se accede

Nombre del action al que se accede

Valor del parámetro

Ruta http para acceder al controlador en cuestión

https://localhost:5001/PronosticoDelClima/GetClima/22-11-2018

Request HTTP

Atributos para recuperar información de un request

Cuando se trabaja con APIs, es común recibir datos de diferentes fuentes, como envíos de formularios, cuerpos de solicitud o parámetros de consulta. En ASP.Net Core, puedes utilizar los atributos `FromForm`, `FromBody` y `FromQuery` para vincular fácilmente los datos entrantes a los parámetros del método de acción

Atributo	Fuente donde se obtienen los datos
<code>[FromBody]</code>	Desde el cuerpo del Request
<code>[FromForm]</code>	Desde los datos en formulario en el cuerpo de un Request
<code>[FromHeader]</code>	Request desde el header del request
<code>[FromQuery]</code>	Desde los parámetros del query string
<code>[FromRoute]</code>	Desde los datos de una ruta en el Request

Request Http

Método Post con un objeto en el cuerpo del request

https://localhost:{PORT}/Paciente/Alta

```
[ApiController]
[Route("[controller]")]
public class PacienteController : ControllerBase
{
    [HttpGet("Alta")]
    public IActionResult Alta([FromBody] paciente paciente)
    {
        //codigo que devuelve el clima
    }
}
```

Response HTTP

Método Get retornando valores

Tipo específico: La acción más simple devuelve un tipo de dato primitivo o complejo (por ejemplo, una cadena o un objeto personalizado).

Considera la siguiente acción, que devuelve una colección de objetos Product personalizados

```
[HttpGet]
public List<Producto> Get()
{
    return new list<Producto>();
}
```

Response HTTP

Método Get retornando valores

IActionResult y ActionResult<T> : es una interfaz comúnmente utilizada para representar el resultado de una acción en un controlador ASP.NET Core. Esta interfaz permite que una acción devuelva una variedad de tipos de resultados, lo que proporciona flexibilidad al desarrollador para elegir el tipo de respuesta que mejor se adapte a la situación.

```
[HttpGet]
public ActionResult<Producto> Get()
{
    return Ok(Product);
}
```

Response HTTP

Status code

Estos son algunos de los códigos de estado HTTP comunes y los métodos de ASP.NET Core que los devuelven

Código de status	Retorno desde ASP
200 OK	<code>return Ok();</code>
201 Created	<code>return Created("/api/productos/1", newProduct);</code>
204 No Content	<code>return NoContent();</code>
400 Bad Request	<code>return BadRequest("Solicitud incorrecta");</code>
401 Unauthorized	<code>return Unauthorized();</code>
403 Forbidden	<code>return Forbid();</code>
404 Not Found	<code>return NotFound("Recurso no encontrado");</code>
500 Internal Server Error	<code>return StatusCode(500, "Error interno del servidor");</code>

Interactuar con los endpoints

Open API

El **estándar OpenAPI** fue originalmente desarrollado como **Swagger** por una empresa llamada **SmartBear Software**. En 2015, la especificación fue transferida a la **OpenAPI Initiative (OAI)**, que es parte de la **Linux Foundation**, con el objetivo de convertirlo en un estándar abierto y ampliamente aceptado para describir APIs. Desde entonces, el nombre oficial es **OpenAPI Specification (OAS)**.

Miembros de la OAI incluyen empresas como Google, Microsoft, IBM, Red Hat, Amazon, PayPal, Atlassian, eBay, SAP, y muchas más.

Documentación de API

Interactuar con los endpoints

Documentación API WEB

Cuando desarrollamos una **API Web**, normalmente exponemos “endpoints” (URL + método HTTP) que hacen cosas como por ejemplo:

GET /api/pacientes → devuelve la lista de pacientes.

POST /api/pacientes → crea un paciente.

PUT /api/pacientes/5 → actualiza un paciente.

DELETE /api/pacientes/5 → elimina un paciente.

EL PROBLEMA: Cómo **documentar** esto de manera clara para que otros desarrolladores o sistemas puedan consumir la API sin tener que leer el código fuente.

Interactuar con los endpoints

Documentación API WEB

OpenAPI es un **estándar abierto** para describir APIs REST.

Antes se llamaba **Swagger Specification**, pero desde 2016 pasó a ser gestionado por la **OpenAPI Initiative** (fundada por empresas como Google, Microsoft, IBM, etc.).

Es un **lenguaje común** (normalmente en formato JSON o YAML) para describir **qué hace tu API**.



Interactuar con los endpoints

Documentación API WEB

GET /api/pacientes → devuelve la lista de pacientes.

POST /api/pacientes → crea un paciente.

PUT /api/pacientes/5 → actualiza un paciente.

DELETE /api/pacientes/5 → elimina un paciente.



```
openapi: 3.0.1
info:
  title: API de Pacientes
  version: 1.0.0
paths:
  /pacientes:
    get:
      summary: Obtener todos los pacientes
      responses:
        '200':
          description: Lista de pacientes
    post:
      summary: Crear un nuevo paciente
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Paciente'
      responses:
        '201':
          description: Paciente creado
  components:
    schemas:
      Paciente:
        type: object
        properties:
          id:
            type: integer
          nombre:
            type: string
```

Interactuar con los endpoints

Documentación API WEB

1. Documentación estandarizada: Cualquiera que consuma el API puede leer un documento OpenAPI y saber:

- Qué endpoints existen.
- Qué parámetros aceptan.
- Qué devuelve cada uno.

2. Interoperabilidad: Como es un estándar, herramientas de terceros pueden leerlo automáticamente.

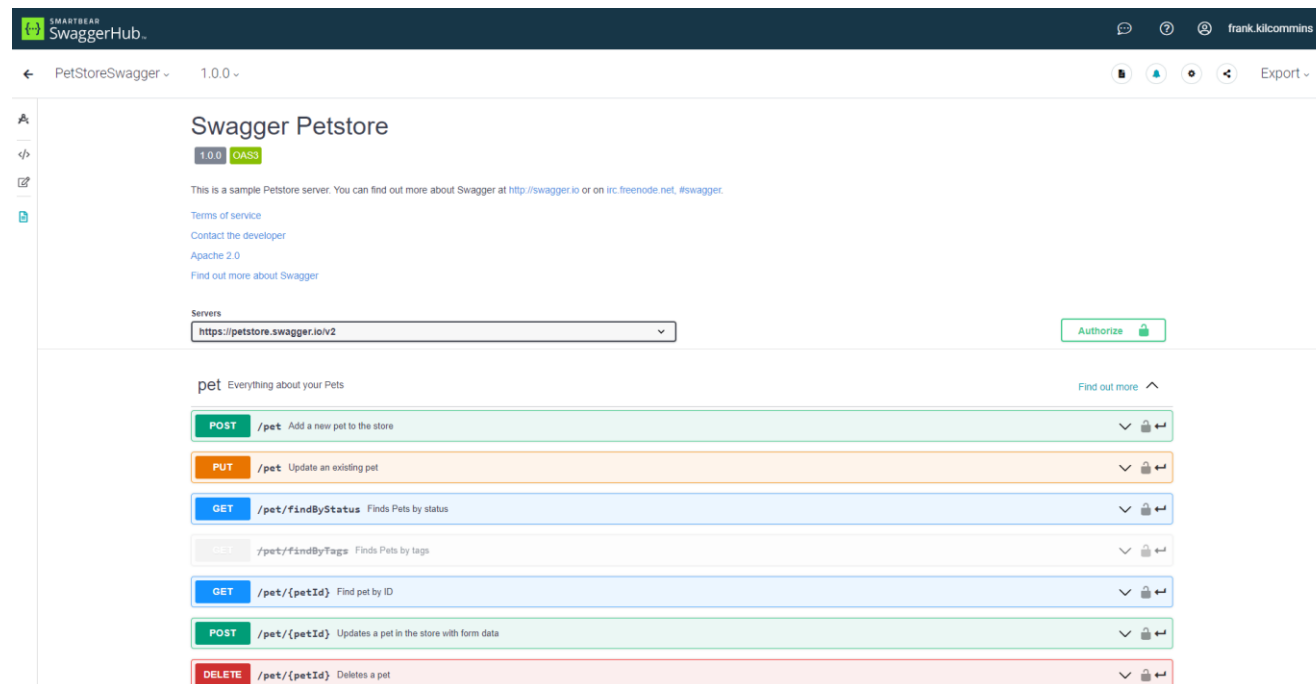
3. Generación de clientes y servidores: Existen herramientas que, a partir del documento OpenAPI, pueden generar código cliente (por ejemplo, un SDK en C#, Java o TypeScript) o stubs de servidor para implementar la API más rápido.

4. Pruebas y exploración interactivas: Interfaces como Swagger UI o Scalar leen ese documento y te muestran una UI donde puedes probar la API con un clic.

Interactuar con los endpoints

swagger-ui

Swagger UI es una herramienta que forma parte del ecosistema de **Swagger** (ahora conocido como **OpenAPI**). Fue creada para ofrecer una **interfaz gráfica** interactiva que permite a los usuarios explorar y probar los endpoints de una API directamente desde el navegador, sin necesidad de usar herramientas externas como Postman.



Interactuar con los endpoints

swagger-ui

1. Instalar el paquete **SwashBuckle**

```
dotnet add package Swashbuckle.AspNetCore
```

2. Agregar la siguiente línea en la configuración de los middlewares

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

3. Agregar el siguiente bloque de código en el if que controla si estamos en modo “**development**”

```
app.UseSwaggerUI(c =>
{
    c.SwaggerEndpoint("/swagger/v1/swagger.json", "Mi API v1"); // ruta del documento OpenAPI
    c.RoutePrefix = string.Empty; // configura para que, si navegamos la raíz podamos ver la UI
de swagger
});
```

Opcional quitar las referencias al uso de OpenApi

Interactuar con los endpoints

Como documentar API

En **.NET (a partir de 5, 6, 7, 8 y 9)** podemos generar documentación automáticamente usando:

Comentarios XML en el código fuente (/// sobre clases y métodos).

- Explican qué hace cada controlador, acción o modelo.
- Se guardan en un archivo .xml al compilar.

Swagger / OpenAPI (con Swashbuckle.AspNetCore).

- Lee esos comentarios XML.
- Muestra una interfaz interactiva (Swagger UI) para probar la API y ver la documentación en un navegador.

De esta forma, con el **mismo código** logramos:

- Código más entendible para otros desarrolladores.
- Una documentación siempre actualizada.
- Una herramienta para probar endpoints en vivo.

Interactuar con los endpoints

Como documentar API

1. Habilitar generación de XML en el .csproj:

Se debe agregar la siguiente configuración en

```
<PropertyGroup>  
    <GenerateDocumentationFile>true</GenerateDocumentationFile>  
    <NoWarn>$(NoWarn);1591</NoWarn>  
</PropertyGroup>
```

¿Qué hace el código anterior?

GenerateDocumentationFile permite generar un archivo XML con los comentarios.

NoWarn 1591 → evita advertencias por miembros públicos sin comentarios XML (útil en pruebas o proyectos chicos).

Interactuar con los endpoints

Como documentar API

1. <summary> - Descripción principal

```
/// <summary>  
/// Obtiene todos los estudiantes activos del sistema  
/// </summary>
```

2. <param> - Descripción de parámetros

```
/// <param name="id">ID único del estudiante </param>  
/// <param name="activo">True para activar, False para desactivar</param>
```

3. <returns> - Descripción del valor de retorno

```
/// <returns>Lista completa de estudiantes registrados</returns>
```

4. <response> - Descripción del valor de retorno

```
/// <response code="200">Operación exitosa - devuelve los datos</response>  
/// <response code="404">Estudiante no encontrado</response>  
/// <response code="400">Datos de entrada inválidos</response>
```

Interactuar con los endpoints

Como documentar API

Resultado en Swagger UI:

Summary: Título del endpoint

Param: Descripción de parámetros en la interfaz

Returns: Descripción de la respuesta

Response codes: Lista de posibles códigos HTTP con sus descripciones

Bibliografía

<https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/http/httpclient>

https://en.wikipedia.org/wiki/Query_string

<https://www.c-sharpcorner.com/article/create-a-net-core-web-api-using-command-line/>

<https://www.roundthecode.com/dotnet-tutorials/fromquery-fromform-what-do-the-net-web-api-attributes-do>