

CLASE 2

Preparando el ambiente de trabajo

Introducción

Es importante que conozcamos un poco acerca de las herramientas de desarrollo que vamos a utilizar y también debemos preparar el ambiente de desarrollo.

Comenzaremos hablando sobre **ASP.NET Core**, de lo que podemos hacer con él y cuando deberíamos utilizarlo. Más adelante veremos cómo preparar el ambiente de desarrollo ya sea que utilicemos **Microsoft Visual Studio Net** o **Visual Studio Code**. Visual Studio lo podemos utilizar en Windows o Mac, mientras que Visual Studio Code lo podemos utilizar tanto en Windows, Linux, y Mac.

Comenzaremos hablando acerca de ASP.NET Core.

Introducción a a ASP.NET Core

ASP. NET Core es la evolución del framework ASP.NET. Con ASP.NET podemos desarrollar aplicaciones web modernas listas para funcionar en la nube y que puedan ejecutarse en distintas plataformas como Windows, Linux y Mac OS, además que sean fáciles de probar con pruebas automáticas y que sean de alto rendimiento.

Cuando hablamos de que ese motor no funciona en la nube nos referimos a que se presta por omisión a trabajar en ambientes de alta demanda donde puede ocurrir el escalamiento horizontal y/o vertical.

Un **escalamiento horizontal** es cuando agregamos más servidores para procesar información, y un **escalamiento vertical** es cuando mejoramos las características de los servidores existentes, por ejemplo agregar memoria RAM entre otras cosas.

El alto rendimiento de ese ASP.NET Core hace que este framework se preste para soluciones **muti tenancy** que es un principio de arquitectura de software.

Una de las características fundamentales de ASP.NET Core es que las aplicaciones hechas en este framework se pueden ejecutar en distintos sistemas operativos como Windows, Linux, o MacOS.

El software moderno y robusto está acompañado de un conjunto de **pruebas automáticas**, por ejemplo, pruebas unitarias (Unit Testing), que verifican el correcto funcionamiento de dicho software. La arquitectura de ASP.NET Core está pensada para poder desarrollar aplicaciones que sean fáciles de probar con pruebas automáticas. ASP.NET Core nos permite hacer **inyección de dependencias** (Dependency Injection) de una manera muy simple pues viene por omisión con un contenedor de **inversión de controles** (inversion of control), en el pasado nos veíamos obligados a utilizar una librería de terceros para este fin.

Como ya hemos dicho tenemos .NET Core y .NET Framework. La pregunta ahora es ¿cuál debemos utilizar?

Se recomienda utilizar .NET Core si va a crear microservicios pues las apps de .NET Core son de alto rendimiento; si va a utilizar contenedores **Docker** necesita capacidad multiplataforma; quieres utilizar un framework moderno de desarrollo, necesitas una línea de comandos para automatizar procesos relacionados con el desarrollo, prueba o publicación de las aplicaciones. En general entre utilizar .NET Framework o .NET Core, debería elegir .NET Core para nuevos desarrollos salvo el hecho que tenga una buena razón para elegir .NET Framework.

Algunas de las razones por las que tendría que utilizar .NET Framework son:

- utilizarás una tecnología la cual no es soportada por .Net Core como WCF o Webforms.
- Necesita utilizar APIs específicos de Windows dado que .Net Core es multiplataforma y este no asume el sistema operativo donde se va a ejecutar.
- Existen funcionalidades de .NET que aún no están disponibles en .NET CORE.

Ya hemos visto que es ASP.NET Core y en qué escenarios podemos utilizarlo. Vamos ahora a aprender a configurar el ambiente de trabajo para poder utilizar ese .NET CORE.

Existen muchas herramientas con las que podemos programar en ASP.NET Core y cualquier editor de texto sirve para esto. Nosotros nos concentraremos en dos herramientas: **Visual Studio** y **Visual Studio Code**:

- Para usuarios de Windows y Mac con computadoras de alto rendimiento se puede utilizar Visual Studio.
- Para usuarios de cualquier sistema operativo incluyendo Linux o con computadoras de bajo rendimiento podemos utilizar Visual Studio Code.

Nota: **muti tenancy** corresponde a un principio de arquitectura de software en la cual *una sola instancia de la aplicación se ejecuta en el servidor*, pero sirviendo a múltiples clientes u organizaciones. Este modelo se diferencia de las arquitecturas con múltiples instancias donde cada organización o cliente tiene su propia instancia instalada de la aplicación. Con una arquitectura muti tenancy, la aplicación puede particionar virtualmente sus datos y su configuración para que *cada cliente tenga una instancia virtual adaptada a sus requerimientos*. Algunos expertos consideran a muti tenancy como un factor decisivo del paradigma de computación en la nube.

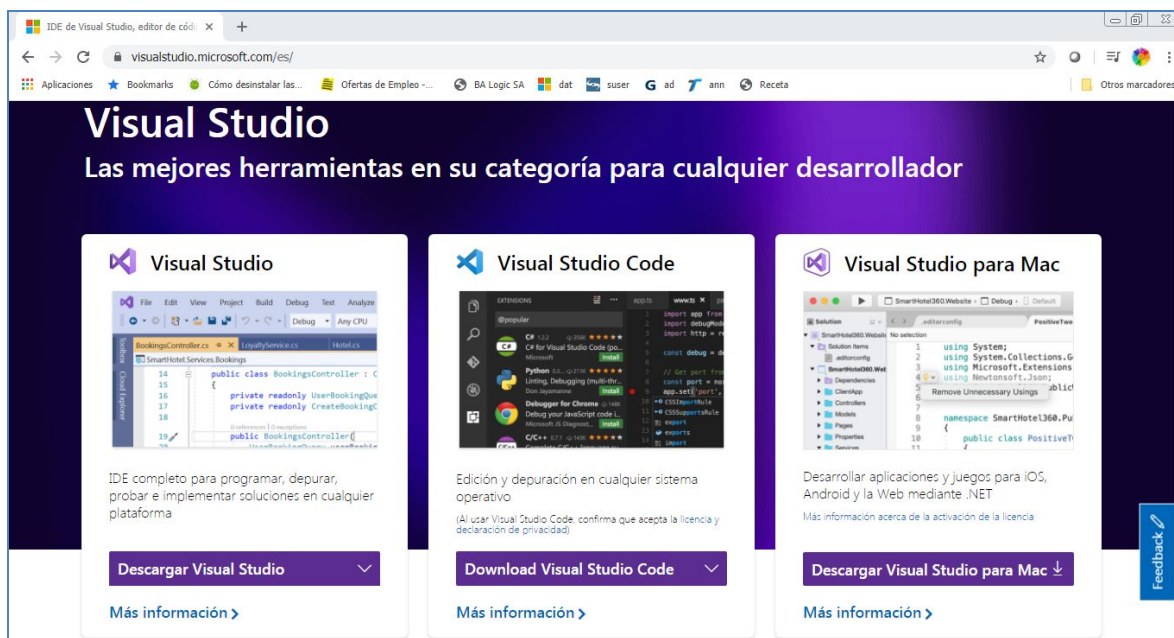
Nota: **Docker** es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de *contenedores de software*, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos (<https://www.docker.com/>).

Instalando Visual Studio y .NET Core

Lo más directo empezar a desarrollar aplicaciones web en ASP.NET Core con Visual Studio y podemos utilizar Visual Studio tanto en Windows como en Mac.

En este curso trabajaremos en Windows con lo que debemos ir al sitio de Visual Studio, y podrá usar desde la versión 2017 en adelante:

<https://visualstudio.microsoft.com/es/>



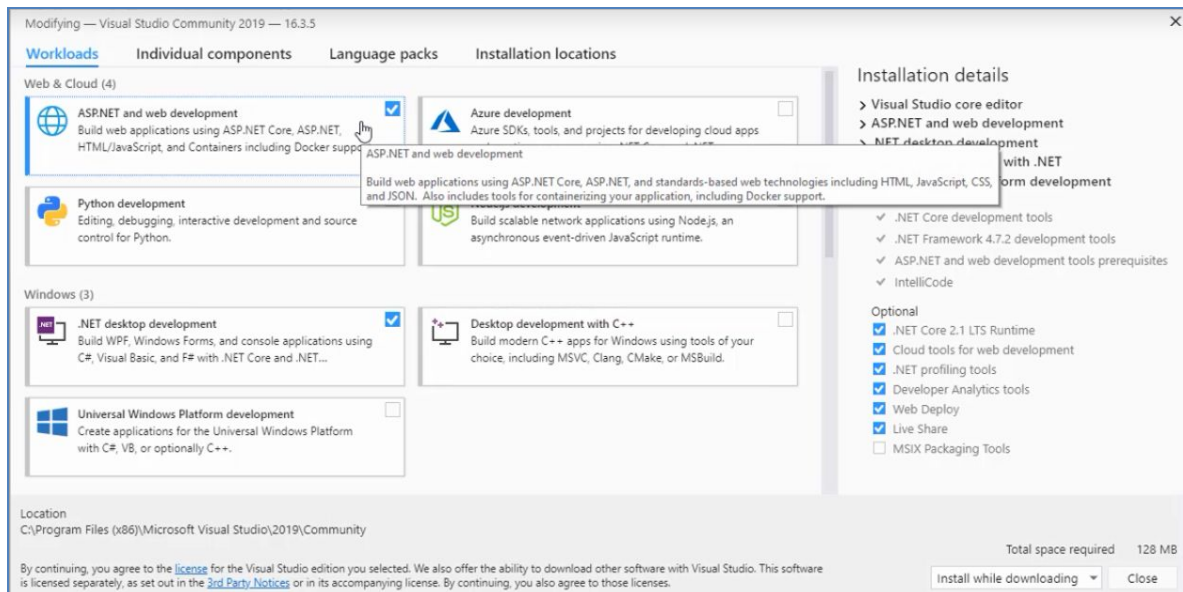
En la página anterior vemos que tenemos Visual Studio para elegir entre las tres versiones de Visual Studio:



La versión Community es una versión gratuita y suficientemente completa para el propósito de este curso, así que utilizaremos esa pero todo lo que aprendamos en este curso aplica para las otras dos versiones también.

También tenemos la opción para los usuarios de Mac de utilizar Visual Studio para Mac.

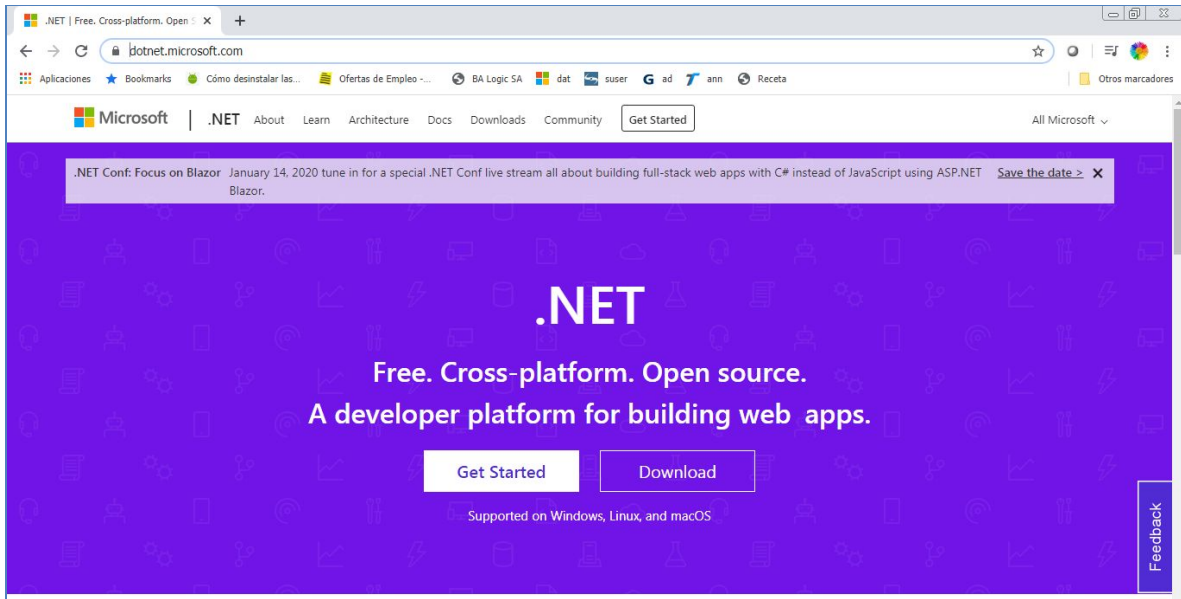
Cuando descargue el instalador de Visual Studio deberá ejecutarlo. Luego y para instalar Visual Studio aparecerá una pantalla similar a esta:



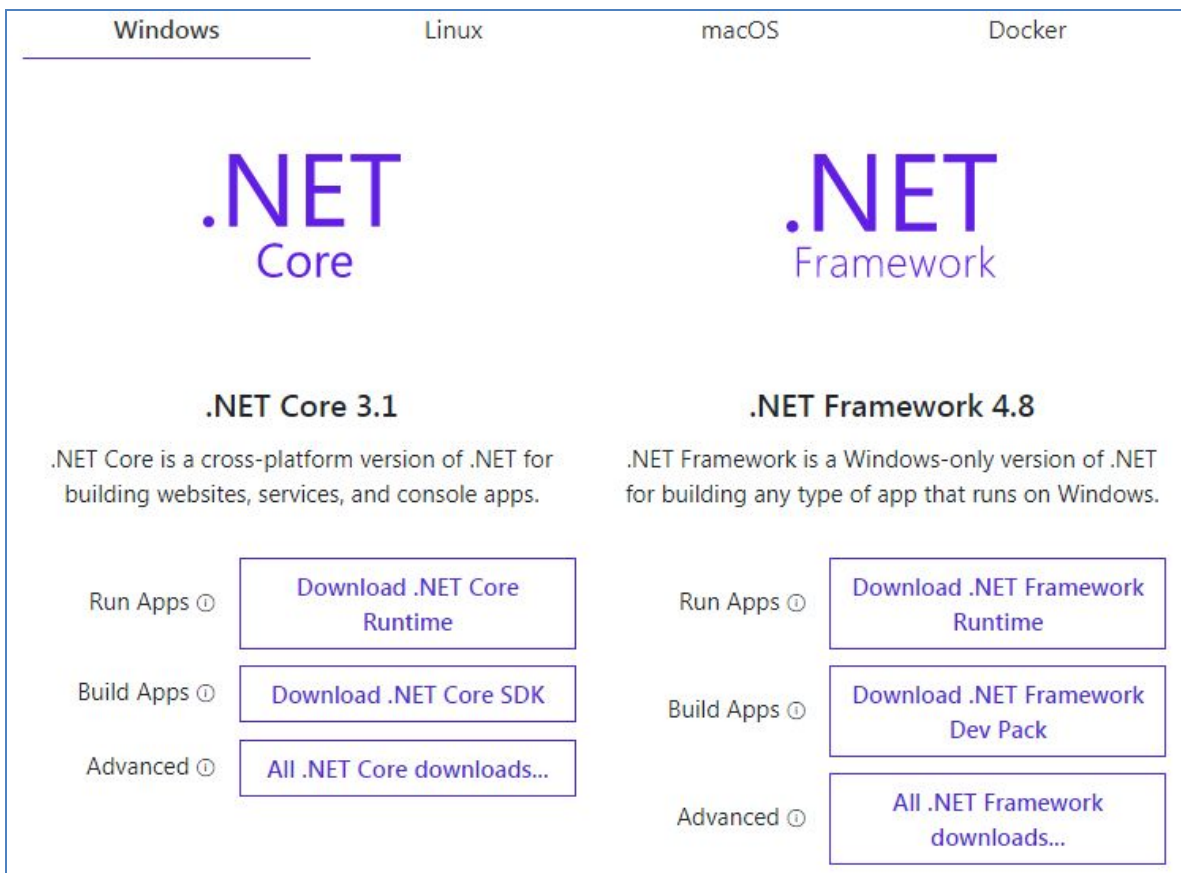
Aquí puede decidir qué herramientas de Visual Studio utilizar, en nuestro caso utilizaremos las herramientas para desarrollo de **ASP.NET and Web Development** y también puede observar en la lista de la derecha el detalle de las herramientas y versiones que se instalarán como ASP.NET, ASP.NET Core, etc. También tiene opciones de otros lenguajes como Python, aplicaciones desktop con C++, etc., si a futuro también usará otros lenguajes.

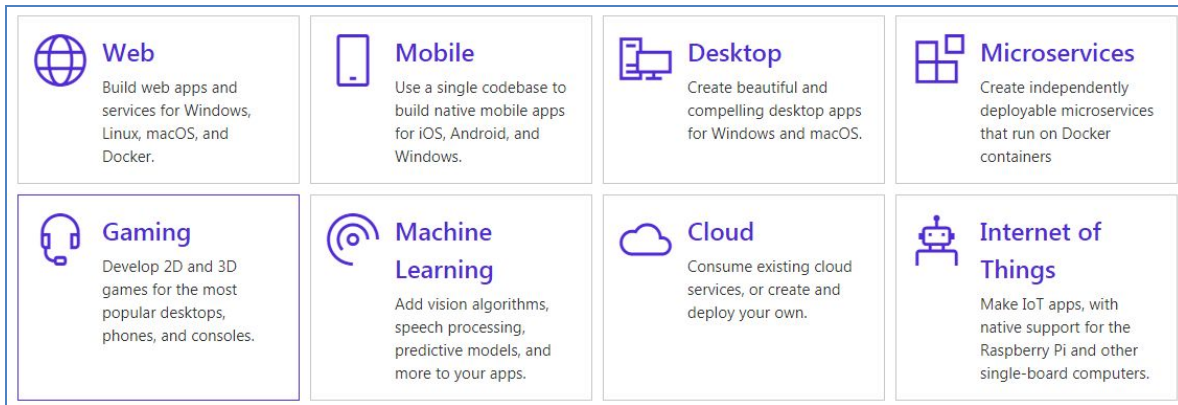
Observe que entre las herramientas de la derecha figuran las versiones de los frameworks que se van a instalar, pero si no figura la versión del SDK de .Net Core que uno necesita, por ejemplo, SDK de .Net Core 3.0., luego de instalar Visual Studio y para actualizar esta herramienta, deberá ir a la siguiente página:

<https://dotnet.microsoft.com/>



Desde esta página accediendo a botón **Download** puede descargar nuevas herramientas y actualizaciones de herramientas existentes de los frameworks, y relacionadas con todos estos sistemas operativos y con todas estas categorías de aplicaciones:





Si ya tiene una instalación de Visual Studio o luego de completar una instalación nueva, algo muy importante es verificar que versión del framework está instalada, y para ello usaremos la línea de comandos **dotnet --version**

```
Command Prompt
Microsoft Windows [Version 10.0.16299.15]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Student>dotnet --version
2.1.505
```

Hola Mundo Web API con Visual Studio Net

En Visual Studio podemos desarrollar, probar, y hacer el despliegue de proyectos de distintas tecnologías.

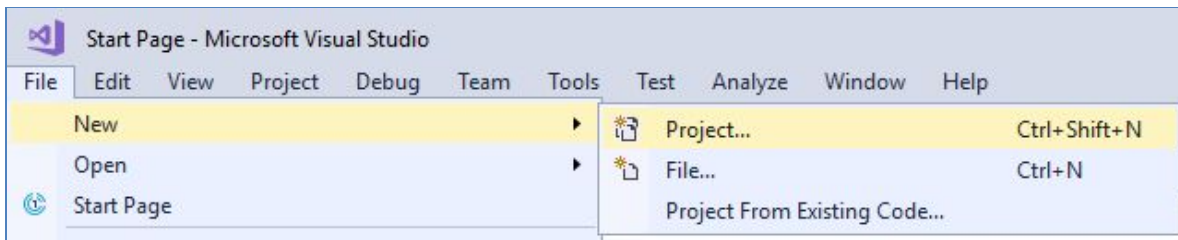
Visual Studio es un IDE (*Integrated Development Environment*) que nos asiste y facilita el trabajo en las distintas etapas del ciclo de vida de desarrollo de un proyecto.

Nuestra tarea ahora va a ser verificar que todo funcione correctamente para poder hacer los ejercicios de este curso.

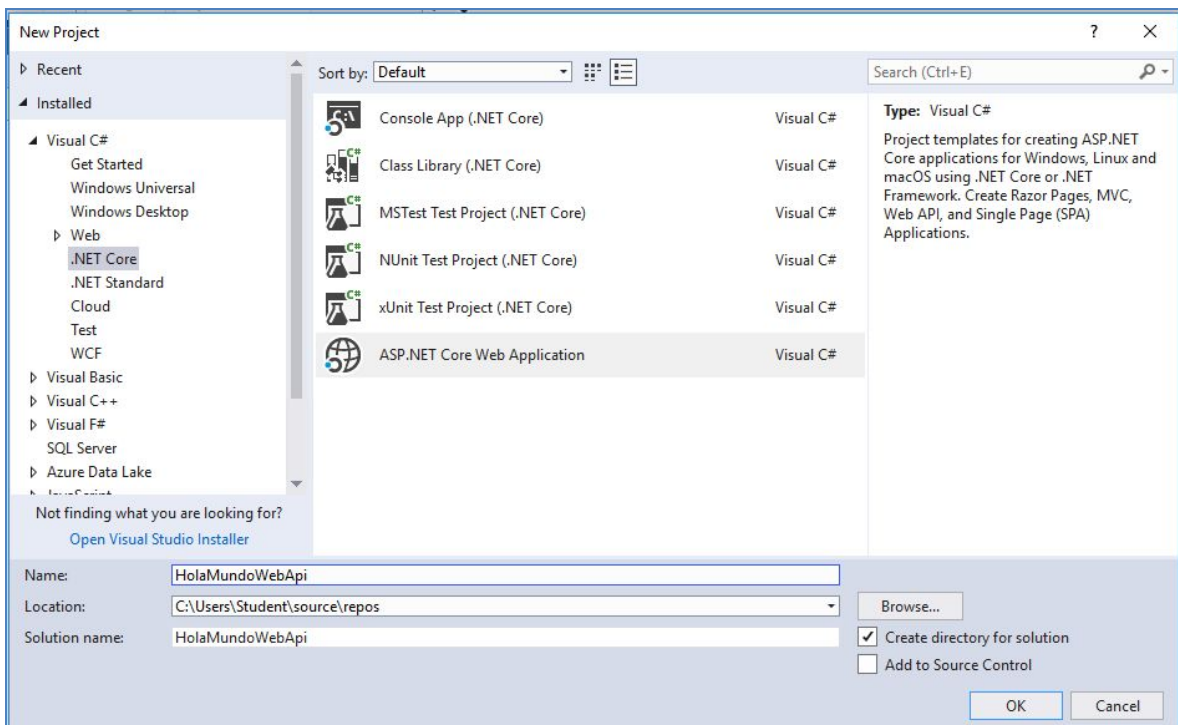
Ahora es momento de crear un nuevo proyecto Web API y ejecutarlo. Las prácticas las haremos con la versión Visual Studio 2017 pero puede perfectamente usar la versión 2019 que tiene las mismas funcionalidades.

Para crear un nuevo proyecto y teniendo Visual Studio abierto:

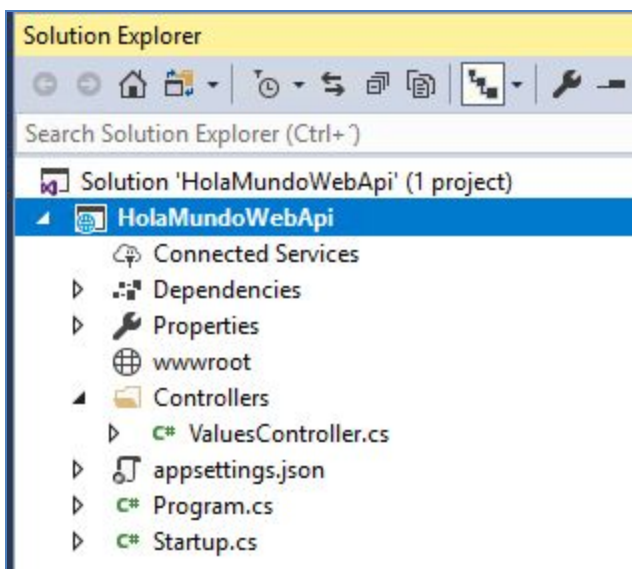
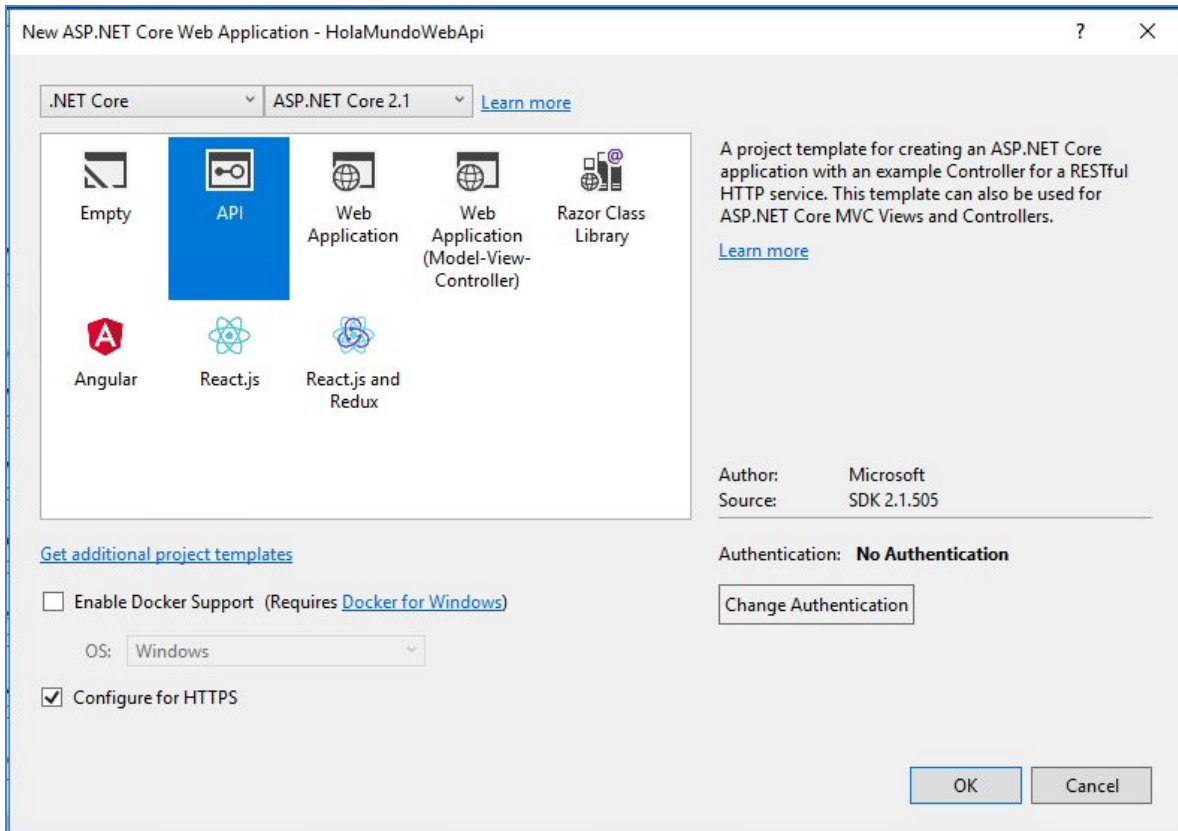
1. Seleccione menú **File – New - Project**:



2. Seleccione **ASP.NET Core Web Application** e indique un nombre para el proyecto, una ubicación y un nombre para la solución si no quiere que el archivo de solución tenga el mismo nombre del proyecto:



3. Luego de dar click en **OK**, aparece un conjunto de plantillas que podemos utilizar para el proyecto, en nuestro caso vamos a utilizar la plantilla API y observe que en la misma ventana hay un combo donde podemos seleccionar la versión de ASP.NET Core que usaremos. Luego click en el botón **OK** para crear el proyecto.



4. Si observamos el Solution Explorer, veremos una estructura de archivos y carpetas, y en la carpeta llamada **Controllers** están los controladores de la aplicación con un controlador con una clase que tiene métodos que se van a ejecutar cuando hagamos una petición HTTP a una o URL del web API.

El código por omisión de la plantilla de un proyecto Web Api de VS 2017, trae el controlador **ValuesController.cs** con el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```



```

using Microsoft.AspNetCore.Mvc;

namespace HolaMundoWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ValuesController : ControllerBase
    {
        // GET api/values
        [HttpGet]
        public ActionResult<IEnumerable<string>> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET api/values/5
        [HttpGet("{id}")]
        public ActionResult<string> Get(int id)
        {
            return "value";
        }

        // POST api/values
        [HttpPost]
        public void Post([FromBody] string value)
        {
        }

        // PUT api/values/5
        [HttpPut("{id}")]
        public void Put(int id, [FromBody] string value)
        {
        }

        // DELETE api/values/5
        [HttpDelete("{id}")]
        public void Delete(int id)
        {
        }
    }
}

```

Este es una clase que tiene un conjunto de métodos.

GET:

Tenemos la función GET decorada con el atributo HTTP GET

```
// GET api/values
[HttpGet]
public ActionResult<IEnumerable<string>> Get()
{
    return new string[] { "value1", "value2" };
}
```

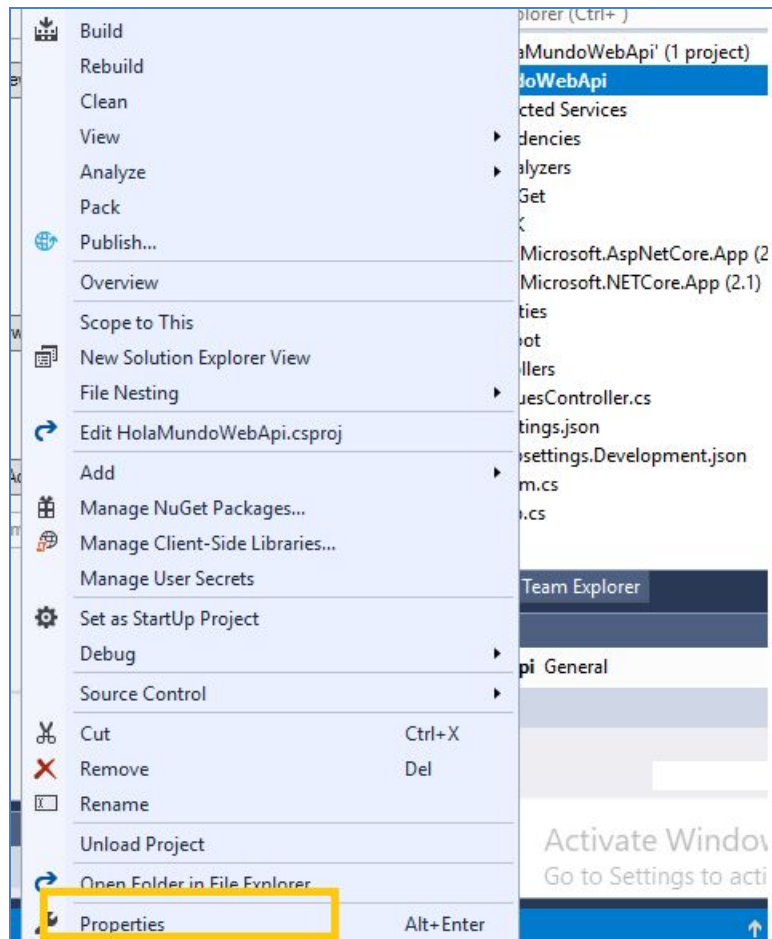
Esta función se va a ejecutar cuando hagamos un HTTP GET hacia la URL que se corresponde con este controlador. Cada controlador típicamente va a tener una URL diferente, la del controlador de este ejemplo es **API/** y el nombre del controlador en este caso **Values**. Esta información está indicada en el atributo **Route** ubicado arriba de la cabecera de la clase y en la cabecera de la clase, y el nombre del controlador como ya dijimos es **Values** y no **ValuesController**, porque se sobreentiende por convención que todos los controladores terminan en controller (observe al comienzo de la clase la referencia using al namespace `Microsoft.AspNetCore.Mvc`);).

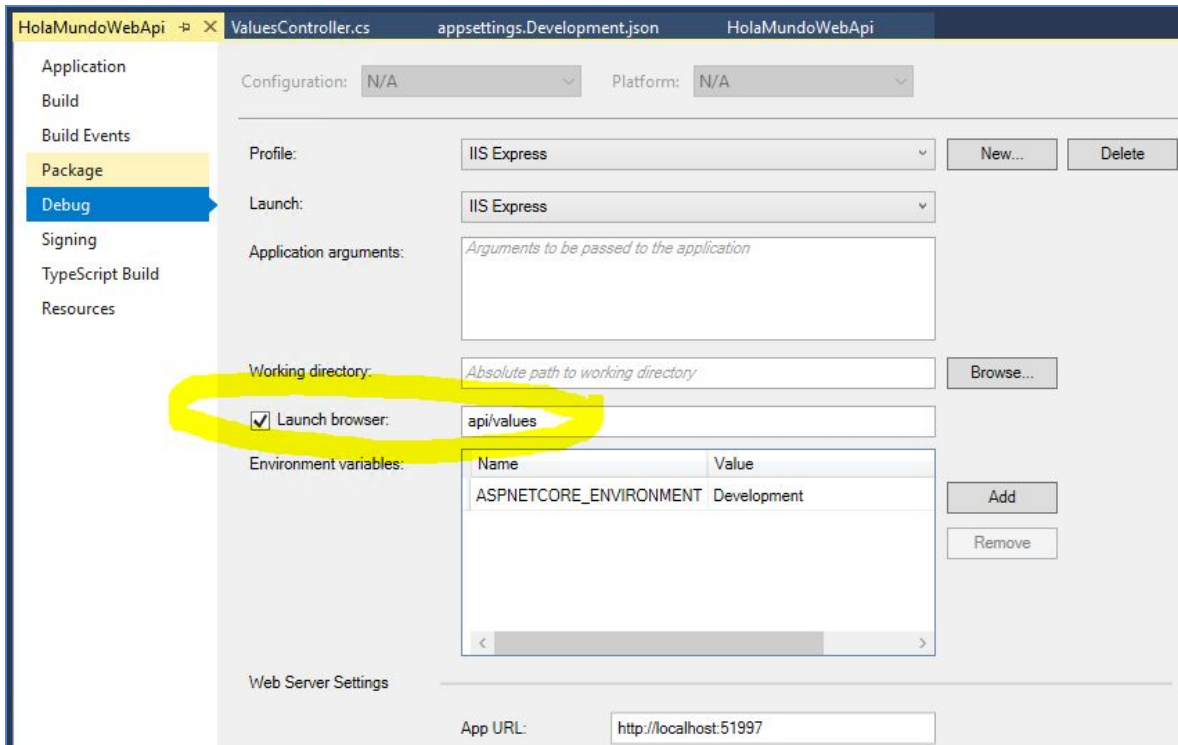
```
[Route("api/[controller]")]
[ApiController]
public class ValuesController : ControllerBase
```

Por tanto para que un cliente de nuestro web ejecute esta la función Get, hacemos un HTTP GET hacia la URL de nuestro dominio barra API barra Values.

<https://localhost:44393/api/values>

Parte de la información sobre el inicio de la aplicación, está en las propiedades del proyecto. En el Solution Explorer si hace click con el botón derecho en el nombre del proyecto y accede a la opción Properties, llegará a las propiedades, y luego seleccione la solapa Debug.

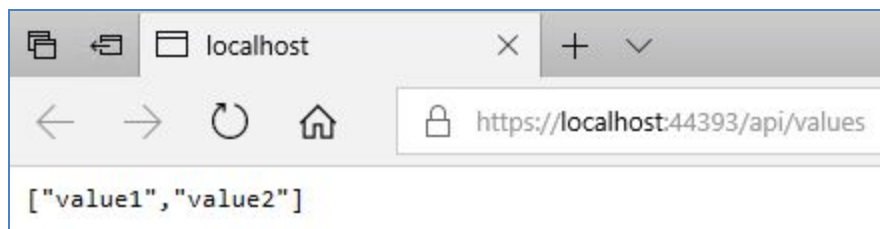




En la opción **Launch browser** figura la ruta del atributo Route de la clase del controlador.

Ahora vamos a ejecutar este el web api, pulsamos <CTRL+F5> o hacer click en botón de Play de la barra de herramientas o en el **menú Debug opción Start Without Debugging**. Se cargará el navegador por omisión para consumir el web api.

La URL en el navegador será similar a es esta:

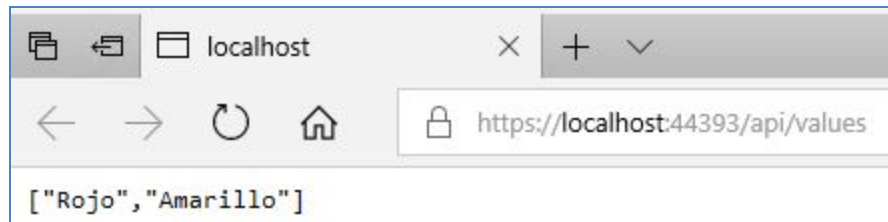


Estamos haciendo un HTTP GET hacia la URL **localhost:puerto/api/values** y esto nos retorna este resultado **value1** y **value2** y si regresamos hacia el código del método Get vemos que esto es lo que devuelve esta función:

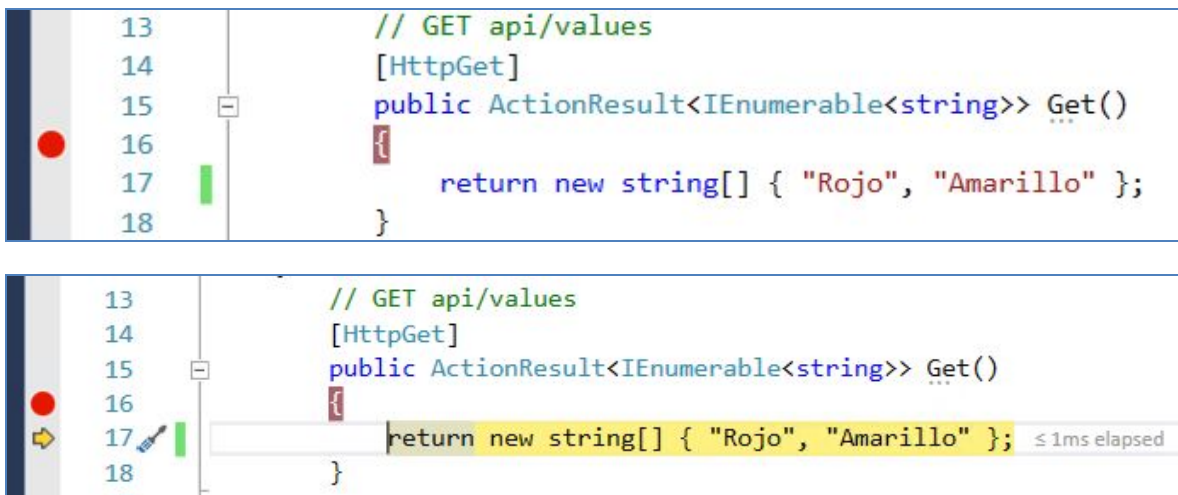
```
// GET api/values
[HttpGet]
public ActionResult<IEnumerable<string>> Get()
{
    return new string[] { "value1", "value2" };
}
```

Para comprobar que efectivamente estamos llamando a Get, modificaremos los valores del string de retorno por rojo y amarillo, y luego de ejecutar observamos el cambio:

```
// GET api/values
[HttpGet]
public ActionResult<IEnumerable<string>> Get()
{
    return new string[] { "Rojo", "Amarillo" };
}
```



Tenga en cuenta que también puede ubicar un breakpoint en el método Get, ejecutar y seguirlo con el Debugger con la tecla <F11>.



Volviendo al código del controlador, observe que tiene más funciones y estas funciones son tan importantes que reciben el nombre de “**acciones**”. Una **acción** es una función del controlador la cual responde a un método HTTP realizada a una URL de nuestro dominio (...y recuerde que también desde el perspectiva de POO la función es un método de la clase), entonces la función GET que vimos recibe el nombre especial de acción. Esas funciones públicas que se corresponden a un método HTTP sobre una URL del web api, reciben el nombre de acciones.

GET con parámetro:

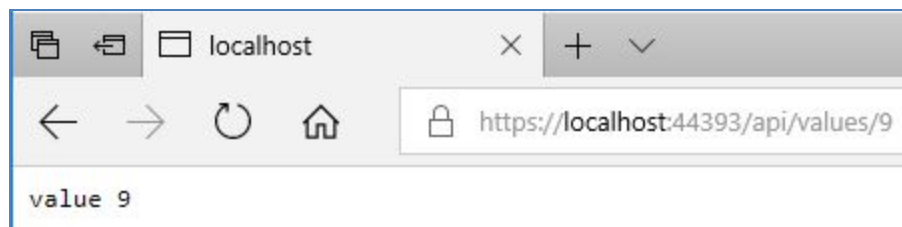
En el código del controlador **Values**, hay otro método GET pero que espera un parámetro.

```
// GET api/values/5
[HttpGet("{id}")]
public ActionResult<string> Get(int id)
{
    return "value";
}
```

Este parámetro lo podemos indicar en la URL pasándole el nombre del parámetro entre claves al atributo HTTP Get. Esto significa que este valor de parámetro del método va a venir desde la URL. Para comprobador que el método realmente funciona, vamos a modificar la función por esta:

```
// GET api/values/5
[HttpGet("{id}")]
public ActionResult<string> Get(int id)
{
    return "value " + id.ToString();
}
```

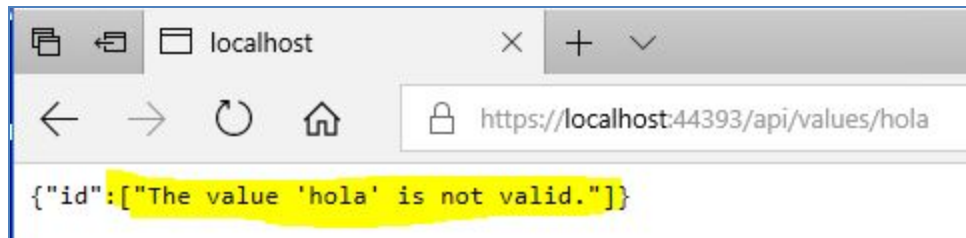
Ejecutamos nuevamente la aplicación y en la URL del navegador escribimos <https://localhost:44393/api/values/9> y luego pulsando <ENTER>:



Observe que se visualiza la palabra value con la concatenación del parámetro.

Vuelva a probar esta acción pero con esta URL

<https://localhost:44393/api/values/hola>



POST, PUT, DELETE:

No estamos limitados solamente a hacer HTTP GET, también podemos HTTP POST, HTTP PUT, HTTP DELETE:

```
// POST api/values
[HttpPost]
public void Post([FromBody] string value)
{
}

// PUT api/values/5
[HttpPut("{id}")]
public void Put(int id, [FromBody] string value)
{
}

// DELETE api/values/5
[HttpDelete("{id}")]
public void Delete(int id)
{
}
```

Veremos ejemplos de cómo consumir estas acciones en las siguientes clases, cuando creemos un primer web API completo, aquí sólo el objetivo fue ver cómo preparar el ambiente para trabajar con Visual Studio.

Reforzamos el siguiente punto prerequisite para este curso: podemos ejecutar nuestras aplicaciones en modo debugging como un ejemplo que vimos reciente, es decir, ejecutar el software línea por línea e ir viendo la ejecución de todo el código. Eso nos ayuda a corregir problemas en nuestro software y generar código robusto.

Veremos un ejemplo de esto pero previamente modificaremos el código del método GET por el indicado en “Segundo ejemplo”:

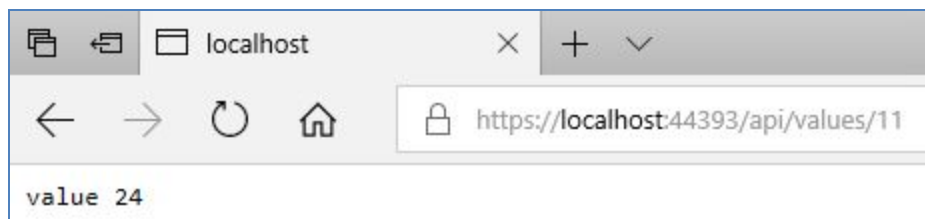
```
// GET api/values/5
[HttpGet("{id}")]
public ActionResult<string> Get(int id)
{
    //Primer ejemplo
    //return "value " + id.ToString();

    //Segundo ejemplo
    id++;
    var aux = id * 2;
    return "value " + aux.ToString();
}
```

Vuelva a compilar pulsando <CTRL+SHIFT+B> (Build Solution), y luego ejecute la aplicación.

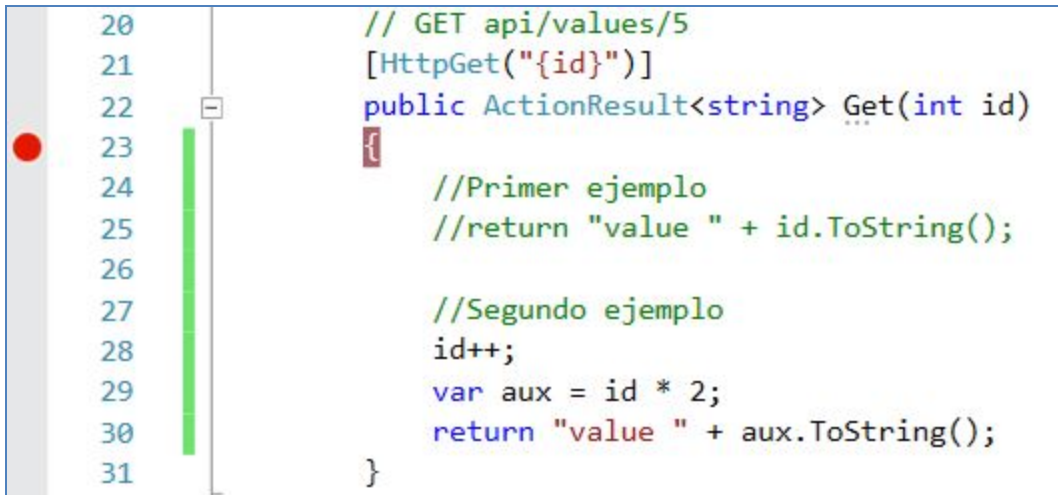
Si ahora en la URL le pasamos 11, el valor que retorna es 24:

<https://localhost:44393/api/values/11>



Supongamos que no queda claro de dónde viene ese 24 y quisiéramos analizar línea por línea por línea la ejecución del código.

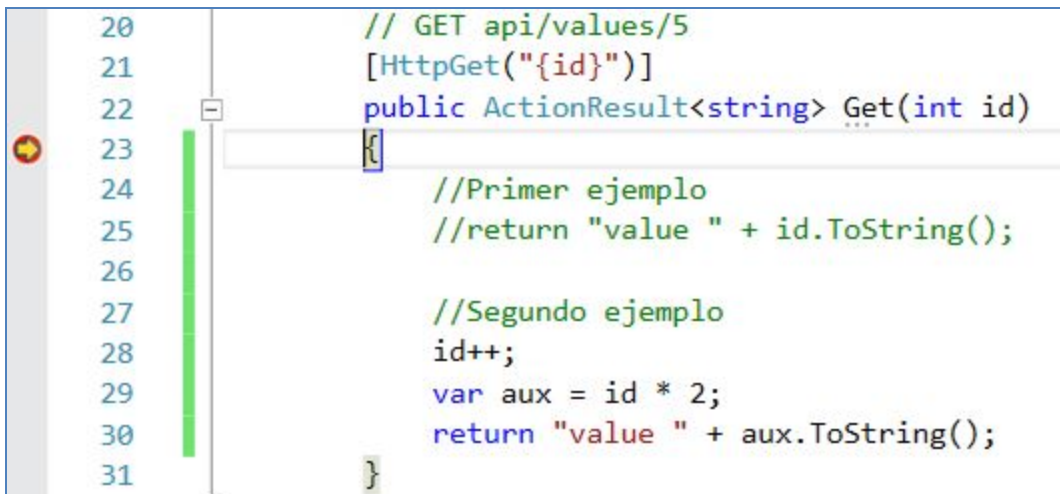
Entonces regresamos a Visual Studio, y colocamos un breakpoint, haciendo click en la barra ubicada a la izquierda de los números de línea:



```
20 // GET api/values/5
21 [HttpGet("{id}")]
22 public ActionResult<string> Get(int id)
23 {
24     //Primer ejemplo
25     //return "value " + id.ToString();
26
27     //Segundo ejemplo
28     id++;
29     var aux = id * 2;
30     return "value " + aux.ToString();
31 }
```

El efecto de este breakpoint será que cuando se esté por ejecutar esa línea de código específica, se detendrá la ejecución y podremos explorar a partir de ahí el código paso a paso.

Pulse <F5>, escriba la URL anterior <https://localhost:44393/api/values/11> y verá que la ejecución se detiene en el breakpoint y una flecha amarilla indica en qué línea de código está en pausa la ejecución.

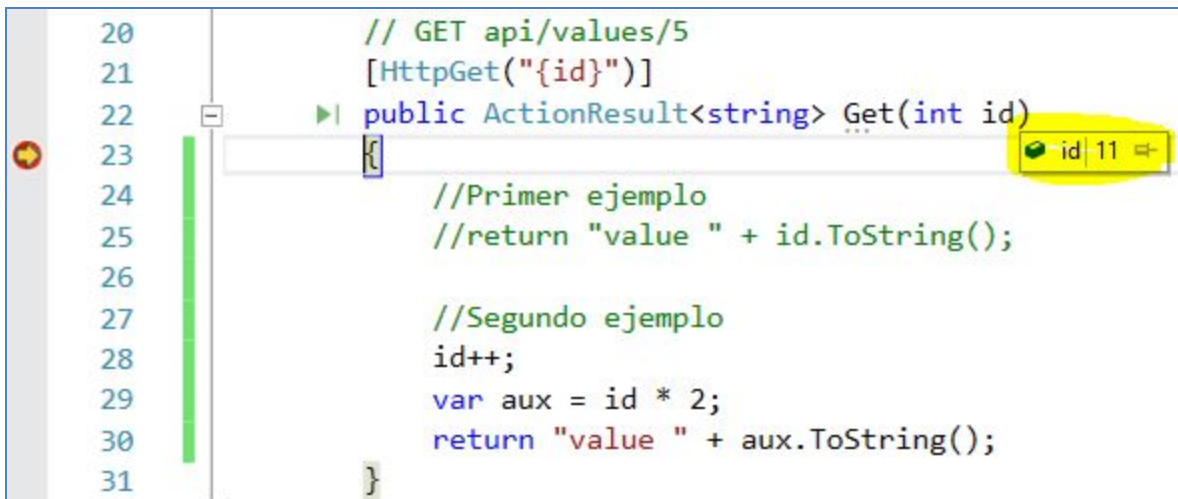



```
20 // GET api/values/5
21 [HttpGet("{id}")]
22 public ActionResult<string> Get(int id)
23 {
24     //Primer ejemplo
25     //return "value " + id.ToString();
26
27     //Segundo ejemplo
28     id++;
29     var aux = id * 2;
30     return "value " + aux.ToString();
31 }
```

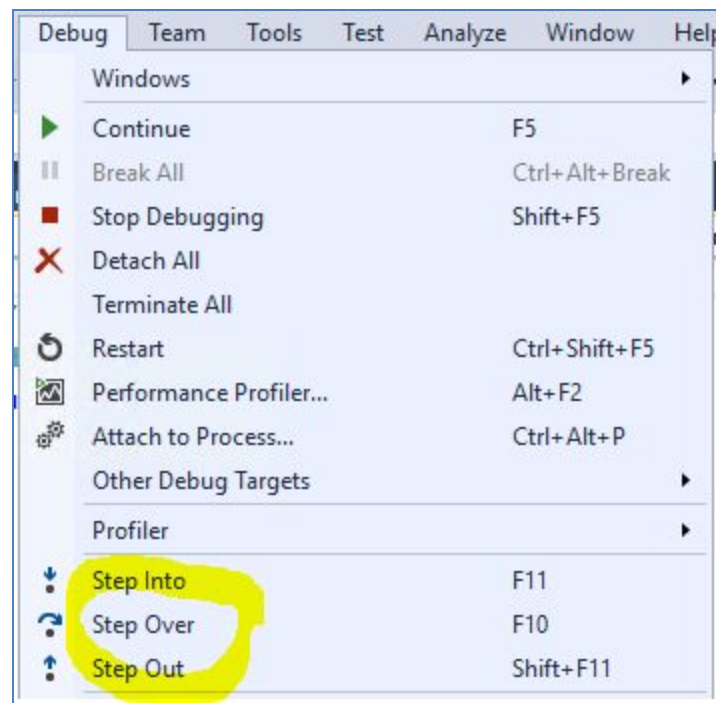
Yo veo utilizar esta opción aquí ahora la que vamos a hacer es que vamos a volver a colocar nuestros

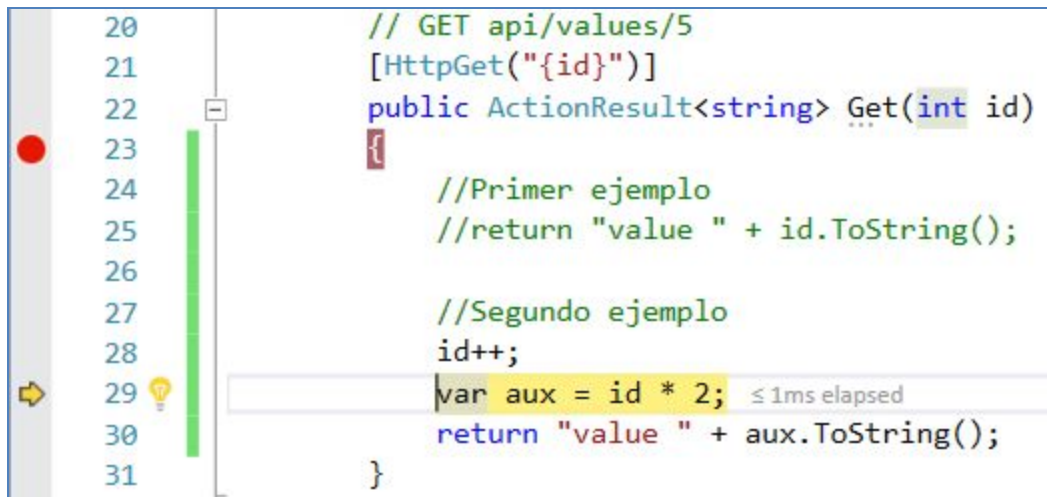
Ahora podemos ver el valor de la id que es 11 ubicando el puntero del mouse sobre el parámetro.

```
20 // GET api/values/5
21 [HttpGet("{id}")]
22 public ActionResult<string> Get(int id)
23 {
24     //Primer ejemplo
25     //return "value " + id.ToString();
26
27     //Segundo ejemplo
28     id++;
29     var aux = id * 2;
30     return "value " + aux.ToString();
31 }
```



Podemos utilizar las opciones o botones  de step para ir avanzando en la ejecución:





```
20 // GET api/values/5
21 [HttpGet("{id}")]
22 public ActionResult<string> Get(int id)
23 {
24     //Primer ejemplo
25     //return "value " + id.ToString();
26
27     //Segundo ejemplo
28     id++;
29     var aux = id * 2; ≤ 1ms elapsed
30     return "value " + aux.ToString();
31 }
```

La idea es que con el proceso de debugging se puede ejecutar la aplicación línea por línea y ver el valor de todas las variables y expresiones, y así poder analizar la lógica de programación del código que estamos desarrollando.

Podemos presionar F5 o continue para continuar con la ejecución frenando en el siguiente breakpoint si lo hay, hasta terminar la ejecución del código.

Preparando una solución con Visual Studio Code


Vamos a utilizar Visual Studio Code para crear un web api Hola mundo. Visual Studio Code es un editor de código de alto rendimiento que puede ser ejecutado en cualquiera de los tres sistemas operativos más reconocidos, Windows, Mac y Linux. Es bastante, liviano, ágil y rápido y ha ganado popularidad en los últimos años por su flexibilidad y facilidad de uso.

Lo primero que debemos hacer es ir al sitio <https://visualstudio.microsoft.com/es/>, y desde aquí **descargar e instalar** la versión de Visual Studio Code correspondiente al sistema operativo donde lo usaremos.

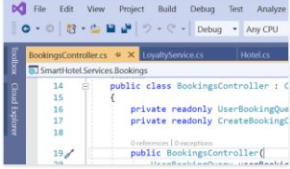
visualstudio.microsoft.com/es/
Aplicaciones
Bookmarks
Cómo desinstalar las...
Ofertas de Empleo -...
BA Logic SA
dat
suser
G ad
ann
Receta

Visual Studio

Las mejores herramientas en su categoría para cualquier desarrollador




Visual Studio



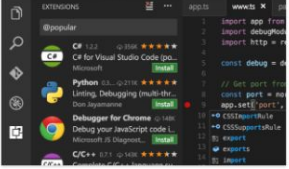
IDE completo para programar, depurar, probar e implementar soluciones en cualquier plataforma

[Descargar Visual Studio](#)

[Más información >](#)



Visual Studio Code




Edición y depuración en cualquier sistema operativo

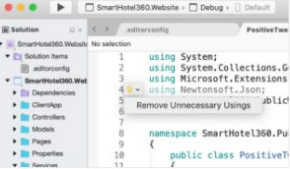
(Al usar Visual Studio Code, confirma que acepta la licencia y declaración de privacidad)

[Download Visual Studio Code](#)

[Más información >](#)



Visual Studio para Mac




Desarrollar aplicaciones y juegos para iOS, Android y la Web mediante .NET

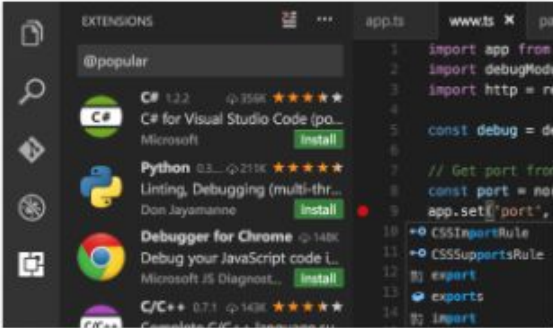
Más información acerca de la activación de la licencia

[Descargar Visual Studio para Mac](#)

[Más información >](#)



Visual Studio Code



Edición y depuración en cualquier sistema operativo

(Al usar Visual Studio Code, confirma que acepta la [licencia](#) y [declaración de privacidad](#))

Download Visual Studio Code

Windows x64User Installer

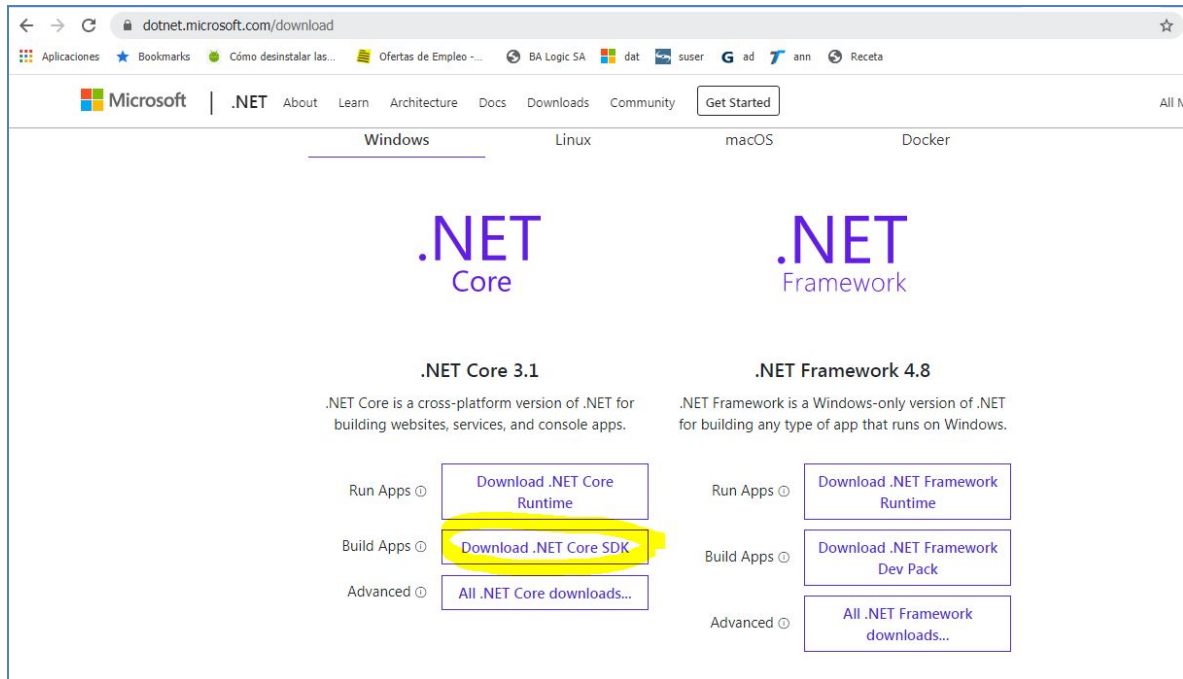
macOSPackage

Linux x64.deb

Linux x64.rpm

Es importante notar que Visual Studio Code no hace suposiciones acerca de la tecnología en la cual se programará, el desarrollador debe preparar la PC para trabajar con la tecnología deseada. En nuestro caso debemos descargar el SDK de .Net Core.

Para eso iremos hacia la página <https://dotnet.microsoft.com/download> y luego en la sección .Net Core, .Net Core 3.0 o .Net Core 3.1, descargar el .Net Core SDK y con eso se iniciará la descarga del SDK.



Nota: Descarga del SDK en

<https://dotnet.microsoft.com/download/visual-studio-sdks>

- Si usará Visual Studio 2019 o Visual Studio Code, deberá bajar la versión del SDK 3.0 o superior.
- Si usará Visual Studio 2017 o Visual Studio Code, deberá bajar la versión del SDK 2.2.

dotnet.microsoft.com/download/visual-studio-sdks

Microsoft | .NET About Learn Architecture Docs Downloads Community Get Started

⚠ These downloads may update Visual Studio and the .NET Framework on your machine. Some of the downloads may only support certain Visual Studio versions.

.NET Core

.NET Core is a cross-platform version of .NET for building websites, services, and console apps.

Version	Visual Studio 2017 SDK	Visual Studio 2019 SDK	Runtime	Release notes
.NET Core 3.1	N/A	x64 SDK x86 SDK (v3.1.101)	x64 Runtime x86 Runtime (v3.1.1)	Release notes
.NET Core 3.0	N/A	x64 SDK x86 SDK (v3.0.102)	x64 Runtime x86 Runtime (v3.0.2)	Release notes
.NET Core 2.2	x64 SDK x86 SDK (v2.2.110)	x64 SDK x86 SDK (v2.2.207)	x64 Runtime x86 Runtime (v2.2.8)	Release notes
.NET Core 2.1	x64 SDK x86 SDK (v2.1.511)	x64 SDK x86 SDK (v2.1.803)	x64 Runtime x86 Runtime (v2.1.15)	Release notes
.NET Core 2.0	x64 SDK x86 SDK (v2.1.202)	x64 SDK x86 SDK (v2.1.202)	x64 Runtime x86 Runtime (v2.0.9)	Release notes

Cuando termine la descarga deberá instalar este SDK y para verificar que la instalación ha sido correcta, debe abrir una ventana de comando o terminal como por ejemplo en Windows podemos utilizar PowerShell, y en esa ventana escribir **dotnet --versión** y verificar que debe salir el número de versión que instalamos.

```
Windows PowerShell
Copyright (C) Microsoft Corporation

Try the new cross-platform PowerShell
PS C:\Users\ASUS> dotnet --version
3.0.100
PS C:\Users\ASUS>
```

Ahora vamos a ver cómo crear un web api con .NET CLI. Para eso indicamos la instrucción **dotnet -- help** y luego de ejecutarla veremos el listado de comandos disponibles.


```
Windows PowerShell
PS C:\Users\Student> dotnet -- help
Unknown option: --
.NET Command Line Tools (2.1.505)
Usage: dotnet [runtime-options] [path-to-application] [arguments]

Execute a .NET Core application.

runtime-options:
  --additionalprobingpath <path>      Path containing probing policy and assemblies to probe for.
  --additional-deps <path>            Path to additional deps.json file.
  --fx-version <version>              Version of the installed Shared Framework to use to run the application.
  --roll-forward-on-no-candidate-fx  Roll forward on no candidate shared framework is enabled.

path-to-application:
  The path to an application .dll file to execute.

Usage: dotnet [sdk-options] [command] [command-options] [arguments]

Execute a .NET Core SDK command.

sdk-options:
  -d|--diagnostics  Enable diagnostic output.
  -h|--help         Show command line help.
  --info            Display .NET Core information.
  --list-runtimes   Display the installed runtimes.
  --list-sdks       Display the installed SDKs.
  --version         Display .NET Core SDK version in use.

SDK commands:
  add          Add a package or reference to a .NET project.
  build        Build a .NET project.
  build-server Interact with servers started by a build.
  clean        Clean build outputs of a .NET project.
  help         Show command line help.
  list         List project references of a .NET project.
  migrate      Migrate a project.json project to an MSBuild project.
  msbuild      Run Microsoft Build Engine (MSBuild) commands.
  new          Create a new .NET project or file.
  nuget        Provides additional NuGet commands.
  pack         Create a NuGet package.
  publish      Publish a .NET project for deployment.
  remove       Remove a package or reference from a .NET project.
  restore      Restore dependencies specified in a .NET project.
  run          Build and run a .NET project output.
  sln          Modify Visual Studio solution files.
  store        Store the specified assemblies in the runtime package store.
  test         Run unit tests using the test runner specified in a .NET project.
```

Tenemos comandos como:

- **Build** para compilar un proyecto
- **New** para crear un nuevo proyecto
- **Test** para correr las pruebas unitarias de un proyecto, etc.

En nuestro caso usaremos **new**, obtendremos ayuda acerca de este comando escribiendo **dotnet new --help** y aparecerá un listado con todas las opciones que tenemos de plantillas de nuevos proyectos.

```
Windows PowerShell
PS C:\Users\Student> dotnet new --help
Getting ready...
Usage: new [options]

Options:
  -h, --help            Displays help for this command.
  -l, --list            Lists templates containing the specified name. If no name is specified, lists all templates.
  -n, --name            The name for the output being created. If no name is specified, the name of the current directory is used.
  -o, --output          Location to place the generated output.
  -i, --install         Installs a source or a template pack.
  -u, --uninstall       Uninstalls a source or a template pack.
  --nuget-source        Specifies a NuGet source to use during install.
  --type               Filters templates based on available types. Predefined values are "project", "item" or "other".
  --force              Forces content to be generated even if it would change existing files.
  -lang, --language    Filters templates based on language and specifies the language of the template to create.

Templates
```

	Short Name	Language	Tags
Console Application	console	[C#], F#, VB	Common/Console
Class library	classlib	[C#], F#, VB	Common/Library
Unit Test Project	mstest	[C#], F#, VB	Test/MSTest
NUnit 3 Test Project	nunit	[C#], F#, VB	Test/NUnit
NUnit 3 Test Item	nunit-test	[C#], F#, VB	Test/NUnit
xUnit Test Project	xunit	[C#], F#, VB	Test/xUnit
Razor Page	page	[C#]	Web/ASP.NET
MVC ViewImports	viewimports	[C#]	Web/ASP.NET
MVC ViewStart	viewstart	[C#]	Web/ASP.NET
ASP.NET Core Empty	web	[C#], F#	Web/Empty
ASP.NET Core Web App (Model-View-Controller)	mvc	[C#], F#	Web/MVC
ASP.NET Core Web App	razor	[C#]	Web/MVC/Razor Pages
ASP.NET Core with Angular	angular	[C#]	Web/MVC/SPA
ASP.NET Core with React.js	react	[C#]	Web/MVC/SPA
ASP.NET Core with React.js and Redux	reactredux	[C#]	Web/MVC/SPA
Razor Class Library	razorclasslib	[C#]	Web/Razor/Library/Razor Class Library
ASP.NET Core Web API	webapi	[C#], F#	Web/WebAPI
global.json file	globaljson		Config
NuGet Config	nugetconfig		Config
Web Config	webconfig		Config
Solution File	sln		Solution

```
Examples:
  dotnet new mvc --auth Individual
  dotnet new nunit -f net472
  dotnet new --help
```

Por ejemplo tenemos plantillas para crear distintos tipos de proyectos:

- aplicaciones de consola,
- librería de clases,
- aplicaciones de WPF,
- Windows Form,
- Worker Services,
- aplicaciones de Blazor, y
- ASP. Net Core.

Nosotros crearemos Web API y para ello debemos primero crear un directorio donde vamos ubicaremos el nuevo proyecto (en la imagen lo creamos a partir de un directorio llamado **proyectos** ubicado en el disco **C**):

Mkdir primerWebApiVSCode

```
Windows PowerShell
PS C:\proyectos> Mkdir primerWebApiVSCode

Directory: C:\proyectos

Mode                LastWriteTime         Length Name
----                -
d-----          1/28/2020  10:11 AM                primerWebApiVSCode
```

Vamos a ir hacia esa carpeta y luego crearemos una solución. Recordamos que una solución permite agrupar distintos proyectos de .Net en un solo lugar, e indicamos:

Dotnet new sln y luego <ENTER> para crear una solución con el nombre de la carpeta en la cual se encuentra.

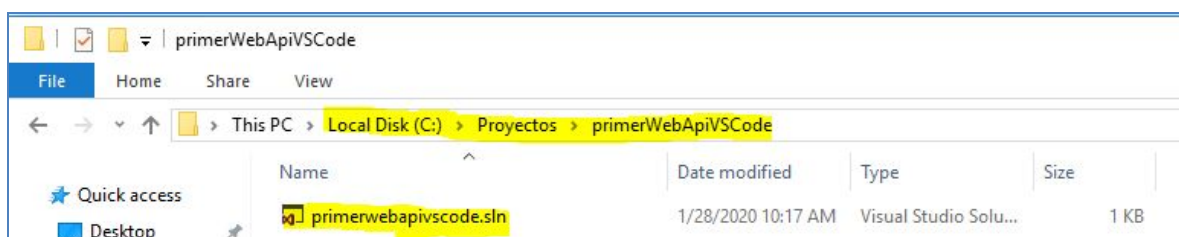
```
Windows PowerShell
PS C:\proyectos\primerwebapivscod> Dotnet new sln
The template "Solution File" was created successfully.
PS C:\proyectos\primerwebapivscod>
```

Si vamos al directorio en el cual creamos la solución veremos que efectivamente tenemos la solución de nombre **primerWebApiVSCode**.

```
Windows PowerShell
PS C:\proyectos\primerwebapivscod> dir

Directory: C:\proyectos\primerwebapivscod

Mode                LastWriteTime         Length Name
----                -
-a----          1/28/2020  10:17 AM             540 primerwebapivscod.sln
```



Vamos a regresar hacia PowerShell e indicaremos **dotnet new webapi** por qué queremos crear un web API.

```
Windows PowerShell
PS C:\proyectos\primerwebapivscore> dotnet new webapi
The template "ASP.NET Core Web API" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on C:\proyectos\primerwebapivscore\primerwebapivscore.csproj...
  Restoring packages for C:\proyectos\primerwebapivscore\primerwebapivscore.csproj...
  Generating MSBuild file C:\proyectos\primerwebapivscore\obj\primerwebapivscore.csproj.nuget.g.props.
  Generating MSBuild file C:\proyectos\primerwebapivscore\obj\primerwebapivscore.csproj.nuget.g.targets.
  Restore completed in 8.97 sec for C:\proyectos\primerwebapivscore\primerwebapivscore.csproj.

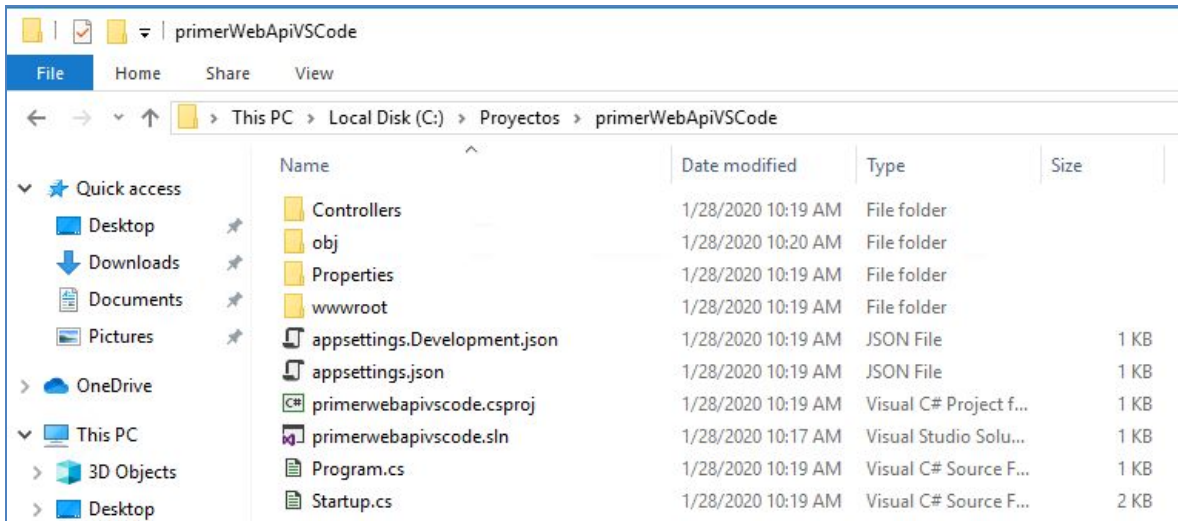
Restore succeeded.
```

Si vamos a la carpeta donde estamos grabando, veremos que efectivamente se ha creado la estructura de un web API como lo hicimos con Visual Studio, tenemos la carpeta **Controllers**, los archivos de proyecto, etc.

```
Windows PowerShell
PS C:\proyectos\primerwebapivscore> dir

Directory: C:\proyectos\primerwebapivscore

Mode                LastWriteTime         Length Name
----                -
d-----          1/28/2020 10:19 AM                Controllers
d-----          1/28/2020 10:20 AM                obj
d-----          1/28/2020 10:19 AM                Properties
d-----          1/28/2020 10:19 AM                wwwroot
-a----          1/28/2020 10:19 AM             146 appsettings.Development.json
-a----          1/28/2020 10:19 AM             105 appsettings.json
-a----          1/28/2020 10:19 AM             416 primerwebapivscore.csproj
-a----          1/28/2020 10:17 AM             540 primerwebapivscore.sln
-a----          1/28/2020 10:19 AM             640 Program.cs
-a----          1/28/2020 10:19 AM            1436 Startup.cs
```



Faltaría agregar una referencia del archivo **.csproj** hacia la solución. Para eso regresamos power Shell, e indicamos **dotnet sln add** y el nombre del archivo de proyecto:

```
Windows PowerShell
PS C:\proyectos\primerwebapivscod> dotnet sln primerwebapivscod.sln add primerwebapivscod.csproj
Project 'primerwebapivscod.csproj' added to the solution.
PS C:\proyectos\primerwebapivscod>
```

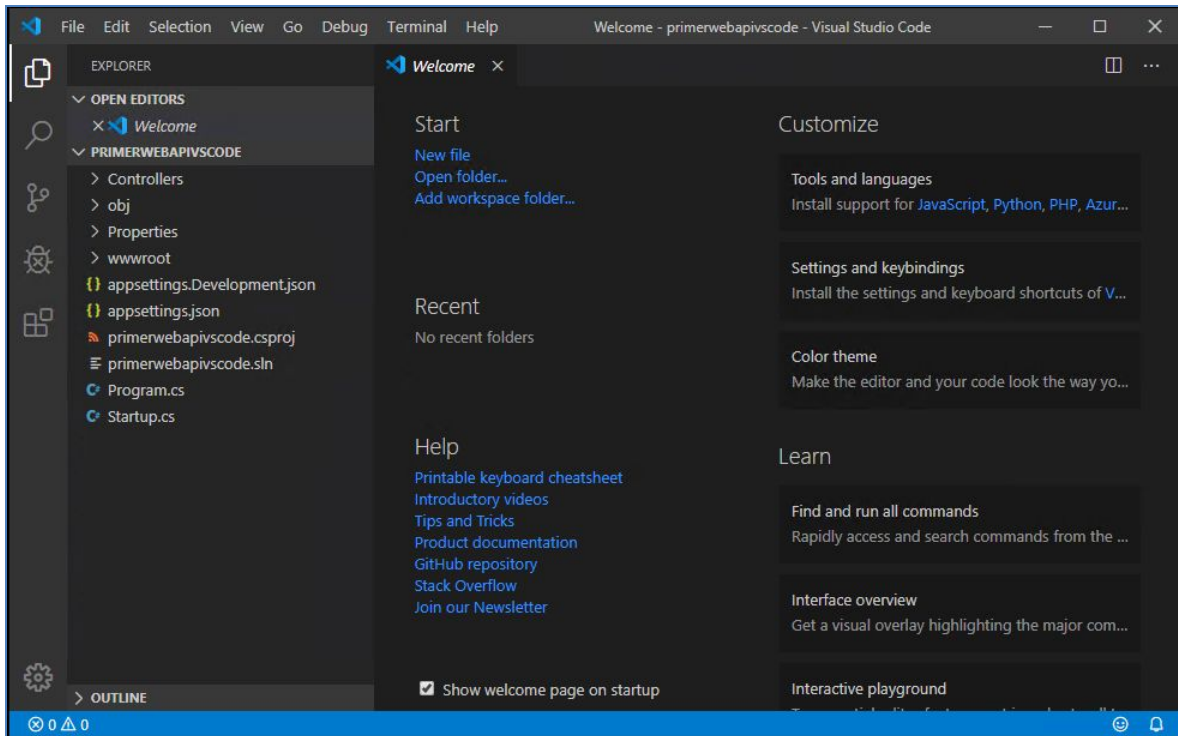
Todos estos pasos extra los tuvimos que hacer para trabajar con Visual Studio Code; si hubiéramos trabajado con Visual Studio .Net la forma de crearlo es vía menús y cuadros de diálogo, es más intuitiva visualmente.

Ahora podemos abrir Visual Studio Code apuntando a la carpeta en la cual nos encontramos, indicando lo siguiente:

Code .

```
Windows PowerShell
PS C:\proyectos\primerwebapivscod> code .
```

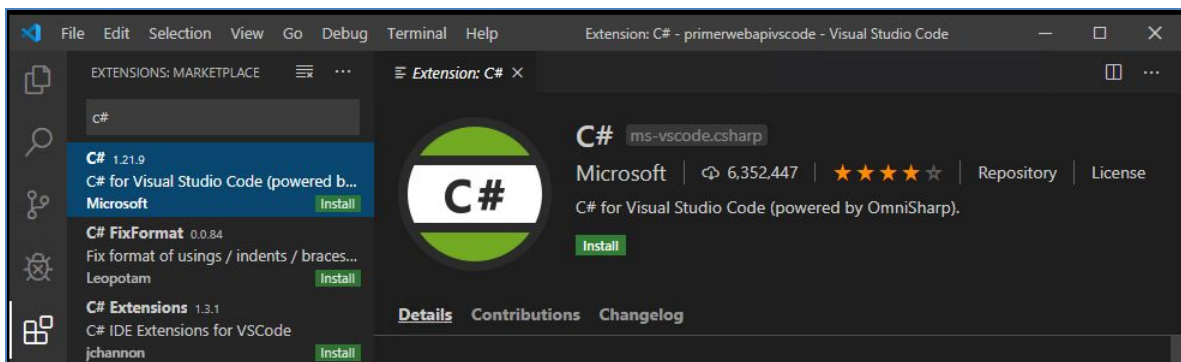
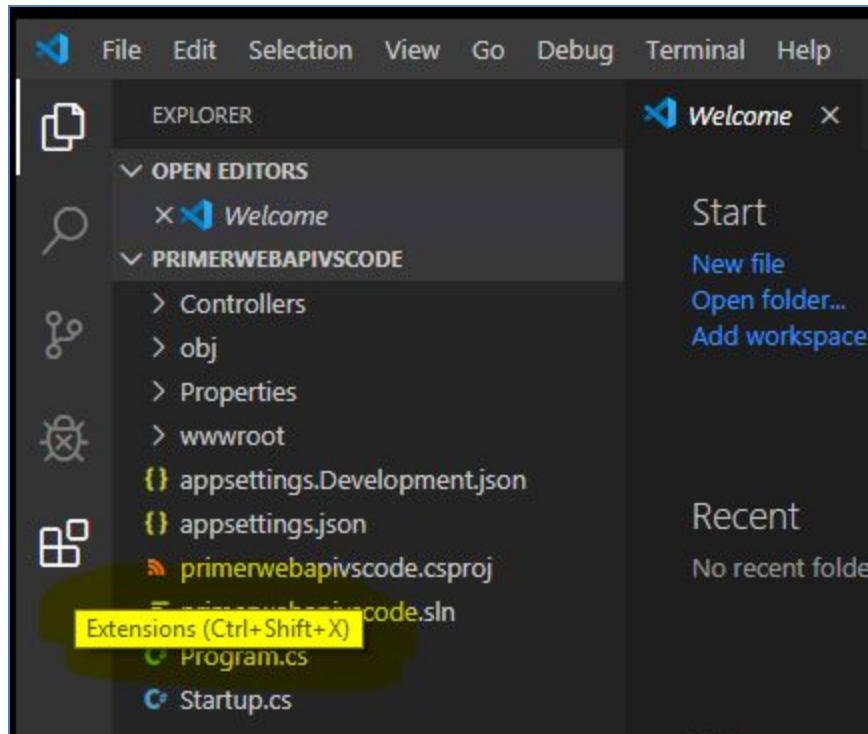
Y la instrucción anterior permitirá a visual studio code abrir lo que haya en la carpeta:



Extensiones de Visual Studio Code para trabajar con WebApi

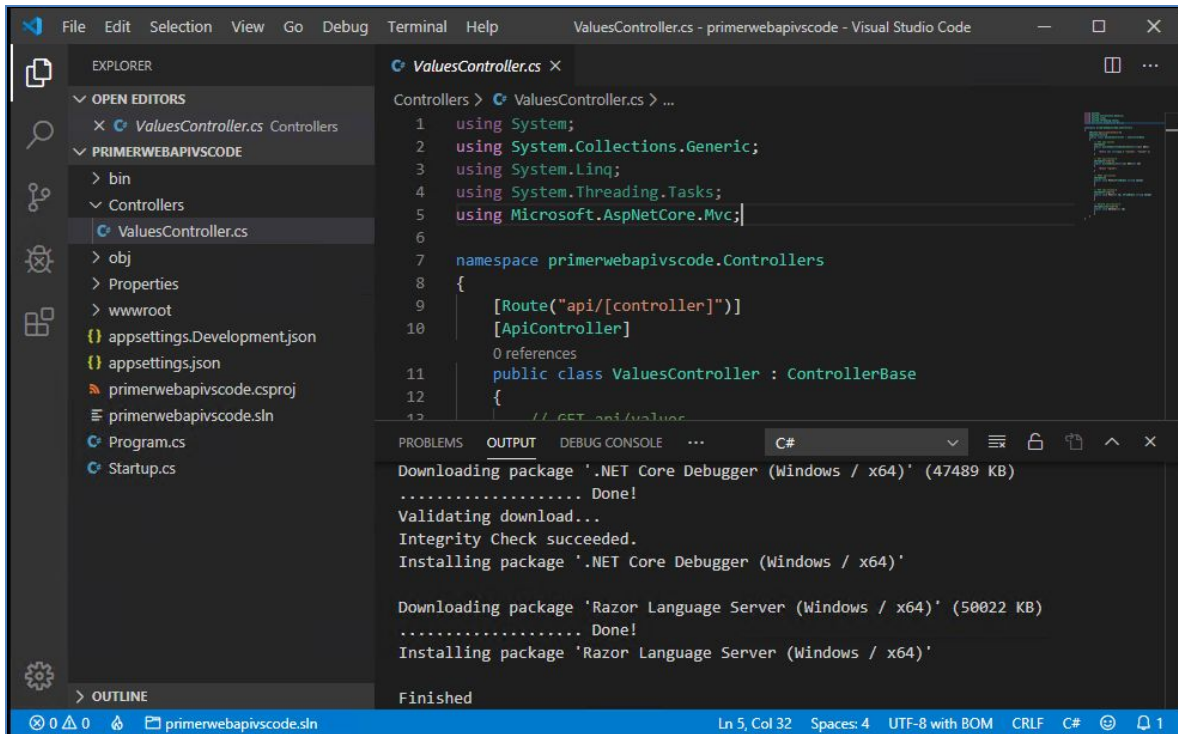
Estando en Visual Studio Code, observamos que utiliza extensiones que son utilidades que sirven para trabajar, por ejemplo, extensiones para trabajar con C#, Python, e incluso tenemos extensiones que nos sirven para hacer nuestra experiencia de desarrollo más agradable.

Deberá tener instalada la extensión de C# como vemos en la siguiente imagen:



Vemos detalles sobre la extensión base del lenguaje C# así como su versión. Y la instalamos.

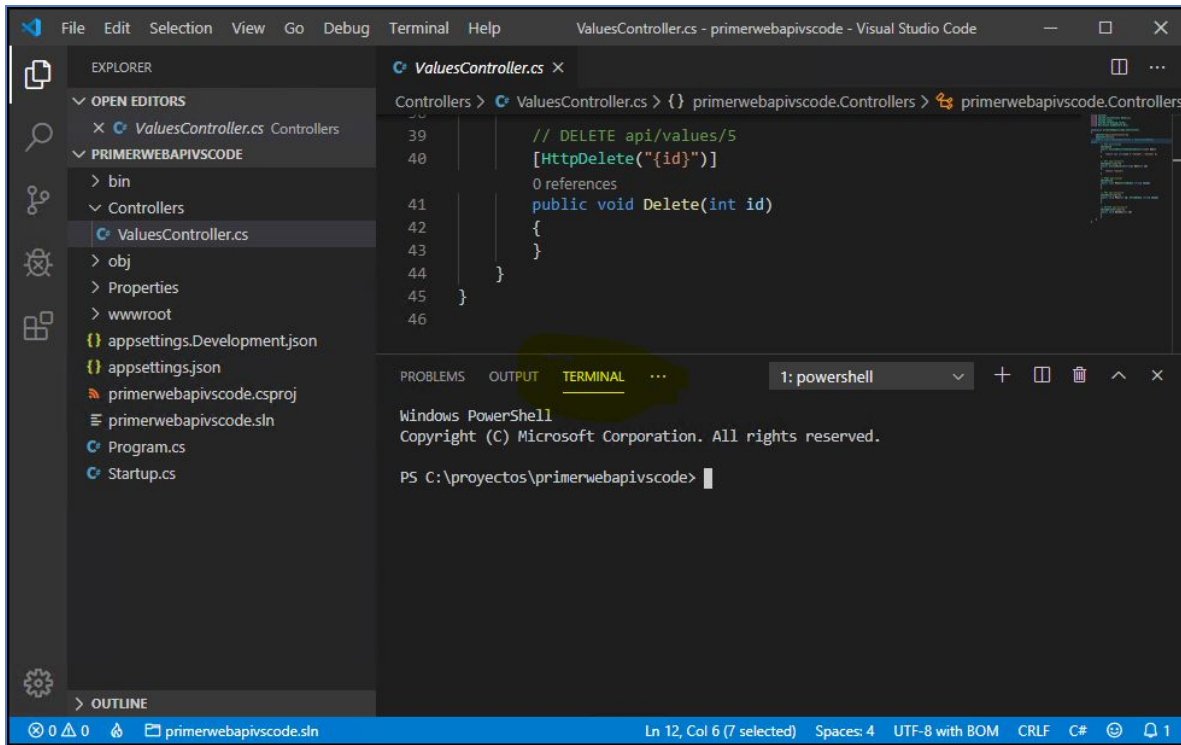
Luego vamos al Explorer y examinaremos la carpeta Controllers y vemos que tenemos el controlador base **ValuesController** con las acciones que vimos en el proyecto creado con Visual Studio .Net.



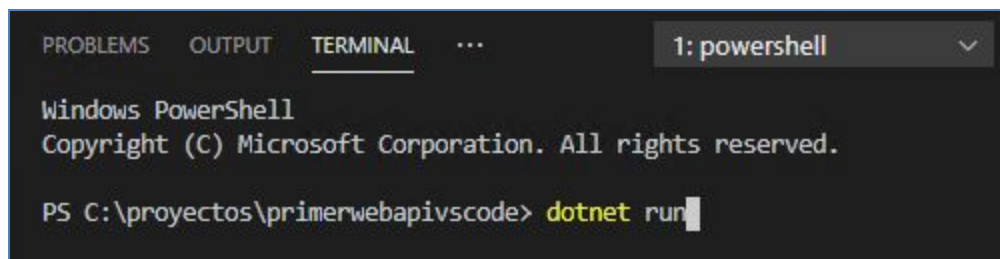
Recordamos que podemos ver un controlador como una clase que tiene funciones las cuales se van a ejecutar cuando hagamos una petición HTTP a una URL del web api. Y es válido todo lo mencionado anteriormente cuando hicimos el proyecto HolaMundo con Visual Studio .Net.

Ejecutaremos el webapi desde Visual Studio Code, para eso vamos a utilizar la terminal que usamos en su momento para instalar VS Code o podemos utilizar las terminales integradas en el IDE de VS Code.

Para utilizar las terminales integradas de VS Code, vamos al menú **View – Terminal**:



Y en la pestaña del terminal haremos un **dotnet run**:



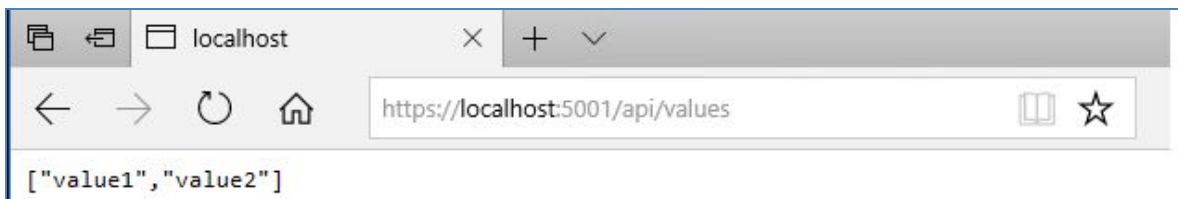
Este comando se utiliza para poder compilar y ejecutar el proyecto. Observamos que ya terminó el proceso y ahora se nos dice que se está escuchando en el puerto 5001 de localhost.

```
PROBLEMS OUTPUT TERMINAL ... 1: dotnet + [icon] [icon]
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\proyectos\primerwebapivscore> dotnet run
Using launch settings from C:\proyectos\primerwebapivscore\Properties\launchSettings.json...
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Users\Student\AppData\Local\ASP.NET\DataProtection-Keys' as key repository and Windows DPAPI to encrypt keys at rest.
Hosting environment: Development
Content root path: C:\proyectos\primerwebapivscore
Now listening on: https://localhost:5001
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

Si hacemos click en el link <https://localhost:5001> carga el navegador, pero no está la URL completa por lo que dará un error.

Recordamos que la URL completa es **<https://localhost:5001/api/values>**



Observe que el resultado de valor 1 y valor 2 y corresponde al método **Get** del controlador.

Para comprobar que realmente esta es la función que se está ejecutando, retorne en el método Get un tercer valor.

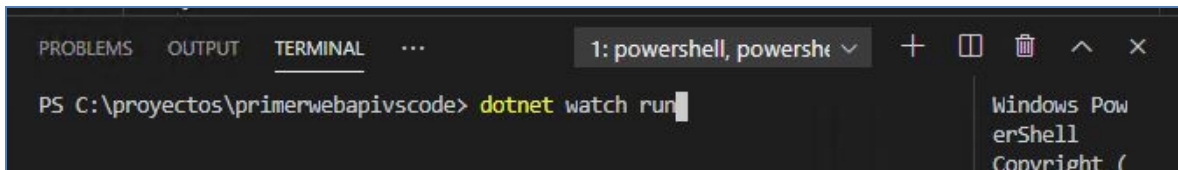
```
11 public class ValuesController : ControllerBase
12 {
13     // GET api/values
14     [HttpGet]
15     public ActionResult<IEnumerable<string>> Get()
16     {
17         return new string[] { "value1", "value2", "value3" };
18     }
```

Para que se refleje el nuevo resultado, debemos guardar y presionar <F5>.

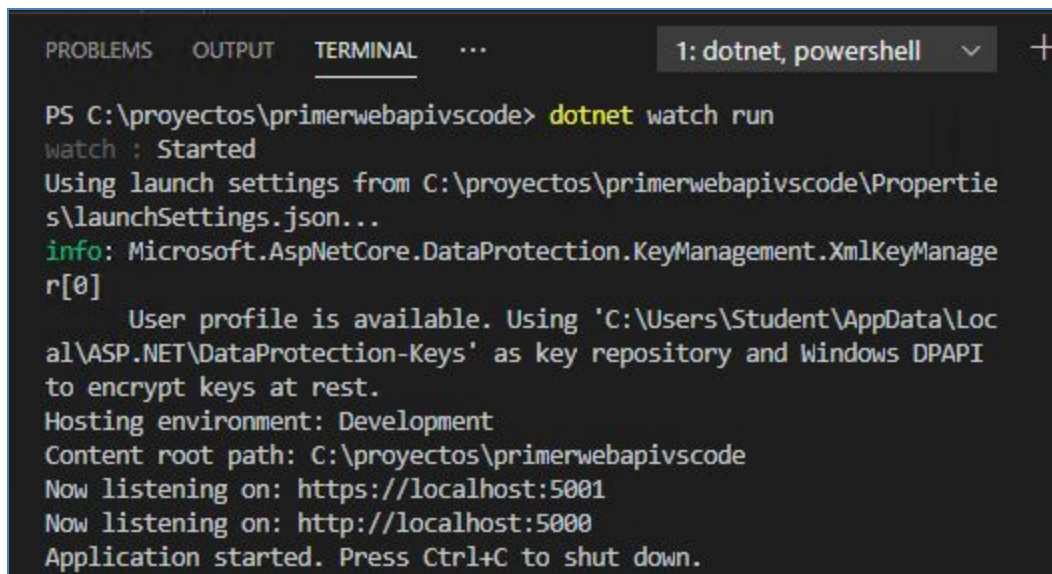
Vamos a ver que no se refleja en nuestro navegador.

Deberíamos volver a ejecutar el comando `dotnet run` nuevamente. Pero para no tener que hacer esto reiteradamente utilizaremos un WATCH.

En la ventana de terminal hacemos <Control+C> para detener la aplicación e ingresamos el comando **dotnet watch run**:



```
PROBLEMS OUTPUT TERMINAL ... 1: powershell, powershell + - [x]
PS C:\proyectos\primerwebapivscod> dotnet watch run
```



```
PROBLEMS OUTPUT TERMINAL ... 1: dotnet, powershell +
PS C:\proyectos\primerwebapivscod> dotnet watch run
watch : Started
Using launch settings from C:\proyectos\primerwebapivscod\Properties\launchSettings.json...
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Users\Student\AppData\Local\ASP.NET\DataProtection-Keys' as key repository and Windows DPAPI to encrypt keys at rest.
Hosting environment: Development
Content root path: C:\proyectos\primerwebapivscod
Now listening on: https://localhost:5001
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

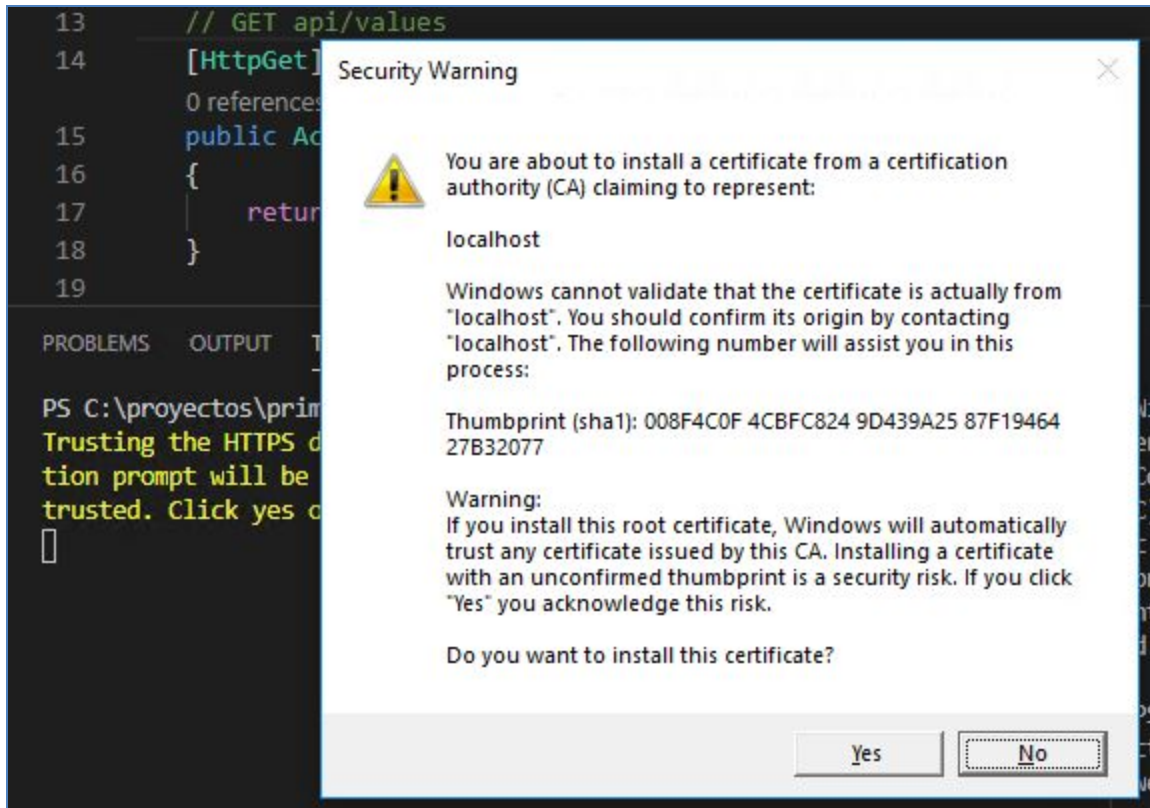
Con este comando lo que estamos diciendo es que cuando hagamos cambios en nuestra aplicación automáticamente el comando `dotnet run` se va a ejecutar.

Vuelva a hacer un cambio en el código para comprobarlo, grábelo y ejecute con <F5>.

Es posible que al intentar abrir esta página a través del HTTPS te salga un error del navegador indicando que no debes confiar en el certificado de este sitio claro porque es un sitio de nuestra propia computadora y el certificado no es válido pero no es una buena experiencia de usuario y queremos corregir eso.

Una forma de corregirlo es volver hacia Visual Studio Code, detenemos la ejecución de la aplicación e indicamos **dotnet dev-certs https -trust**:

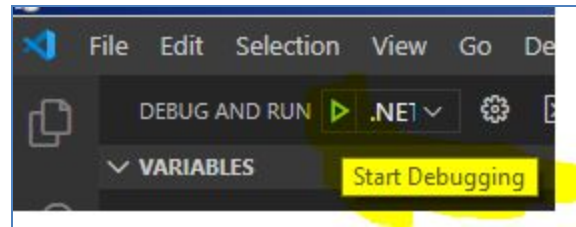
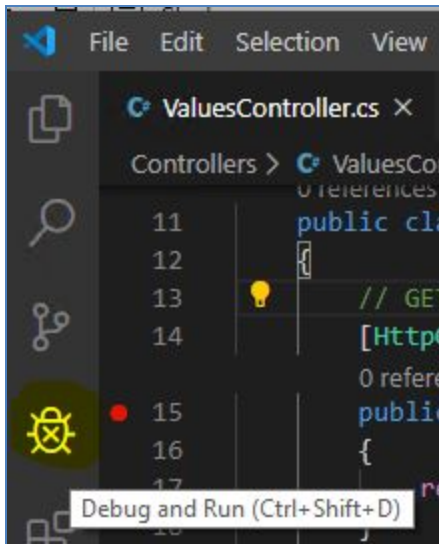

```
PROBLEMS OUTPUT TERMINAL ... 1: powershell, powershell +
PS C:\proyectos\primerwebapivscod> dotnet dev-certs https --trust
```



Aparece una ventana advirtiéndolo que estamos por instalar un certificado de seguridad y si realmente confiamos en él. Instalamos el certificado y a partir de allí podemos volver a hacer un **dotnet watch run** y no saldrá más el mensaje de seguridad.

Si quisiéramos trabajar en modo Debug, con las herramientas de depuración que conocemos de Visual Studio .Net, por ejemplo un breakpoint, un watch, nos manejamos con el menú **Debug** y las teclas rápidas o atajos de sus opciones que ya conocemos.

Habilitamos la solapa de Debug, desde el botón **Debug and Run**, y luego con un breakpoint incluido en el código, pulsamos <F5> y click en el botón **Start Debugging**.

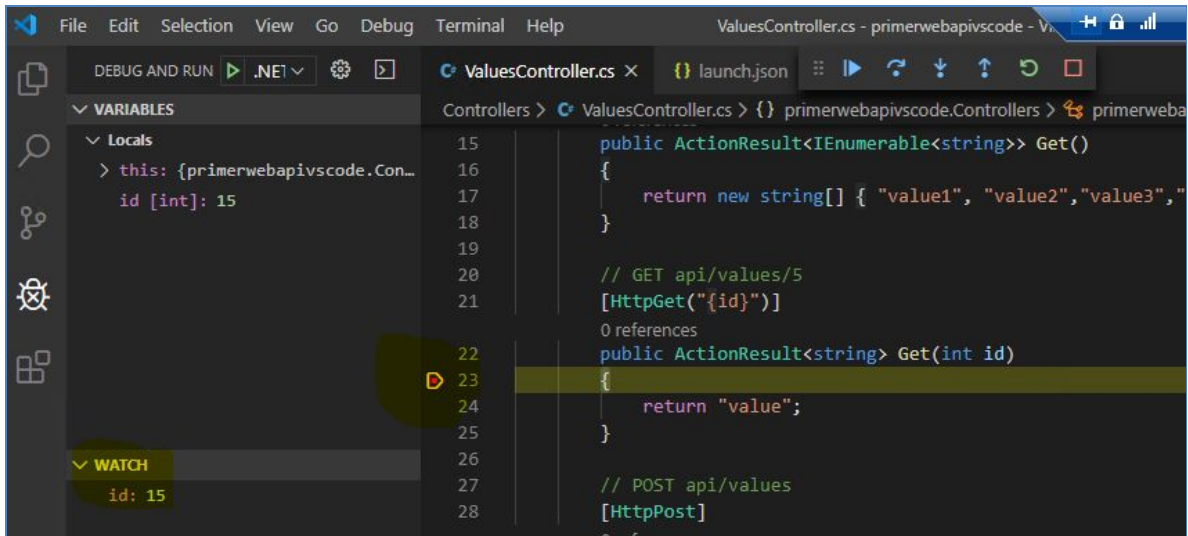


Puede dar el siguiente error porque el puerto actual donde estaba escuchando la aplicación, está ocupado mostrando el mensaje **"Failed to bind to address https://127.0.0.1:5001: address already in use."**

```
Exception has occurred: CLR/System.IO.IOException
An unhandled exception of type 'System.IO.IOException' occurred in System.Private.CoreLib.dll: 'Failed to bind to address https://127.0.0.1:5001: address already in use.'
Inner exceptions found, see $exception in variables window for more details.
Innermost exception System.Net.Sockets.SocketException : Only one usage of each socket address (protocol/network address/port) is normally permitted
    at System.Net.Sockets.Socket.UpdateStatusAfterSocketErrorAndThrowException(SocketError error, String callerName)
    at System.Net.Sockets.Socket.DoBind(EndPoint endPointSnapshot, SocketAddress socketAddress)
    at System.Net.Sockets.Socket.Bind(EndPoint localEP)
    at Microsoft.AspNetCore.Server.Kestrel.Transport.Sockets.SocketTransport.BindAsync()
```

Entonces vamos a detener la aplicación en el terminal con <Ctrl+C> y ahora nuevamente vamos a hacer ese proceso. Este error es normal pues no se pueden tener dos aplicaciones en ejecución en la misma URL.

Luego de detener la aplicación, solamente la estamos ejecutando en modo de Debugging, volvemos al navegador y escribimos en la URL <https://localhost:5001/api/values/15> y si vamos a Visual Studio Code, vemos que se está ejecutando en modo Debugging y la ejecución ha frenado en la llamada al método donde ubicamos el breakpoint.



Y así sucesivamente, usar **watch** para explorar el valor de las variables, y así analizar la ejecución del código línea por línea.

También podemos usar los botones de step  para ir línea por línea, función por función, regresar a una línea de código anterior, etc.