educación IT
CODE YOUR FUTURE

Programación Web .NET Core

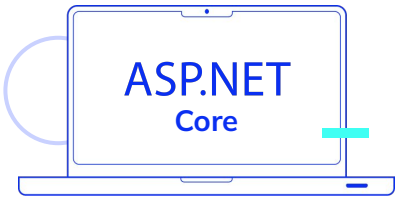
Módulo 4

educación IT
CODE YOUR FUTURE

Introducción a ASP.NET Core

Introducción a ASP.NET Core

- **ASP.NET Core** es:
  - Rediseño moderno de **ASP.NET 4.x**.
  - Multiplataforma y código abierto.
  - Marco modular, eficiente y de alto rendimiento.
  - Listo para la nube.
- Admite los siguientes modelos de programación:
  - **Páginas Razor**.
  - **MVC**.
  - **API web**.



Es un framework de código abierto y multiplataforma para la creación de aplicaciones modernas conectadas a Internet, como aplicaciones web y APIs Web. Con **ASP.NET Core** se puede desarrollar en varias plataformas: **Windows**, **macOS** y **Linux**, implementar la aplicación en la nube, además de crear servicios y backends móviles.

ASP.NET Core puede ejecutarse sobre el **.NET Framework** completo o sobre **.NET Core**.

Aplicación web (controlador de vista de modelos)

Una plantilla de proyecto para crear aplicaciones ASP.NET Core con controladores y vistas de ASP.NET Core MVC de ejemplo. Esta plantilla también puede usarse para servicios RESTful HTTP.

ASP.NET Core es un rediseño completo de ASP.NET. No es una actualización de ASP.NET 4, por lo que su arquitectura ha sido diseñada para resultar más **ligera y modular.**

ASP.NET Core se basa en un conjunto de paquetes **NuGet** granulares y bien factorizados. Esto permite optimizar una aplicación para incluir sólo los paquetes **NuGet** que se necesiten.

ASP.NET Core incluye **tres modelos de programación principales** que pueden ayudar a desarrollar una aplicación sin problemas: **MVC**, **Razor Pages** y **Web API**.

Páginas de Razor

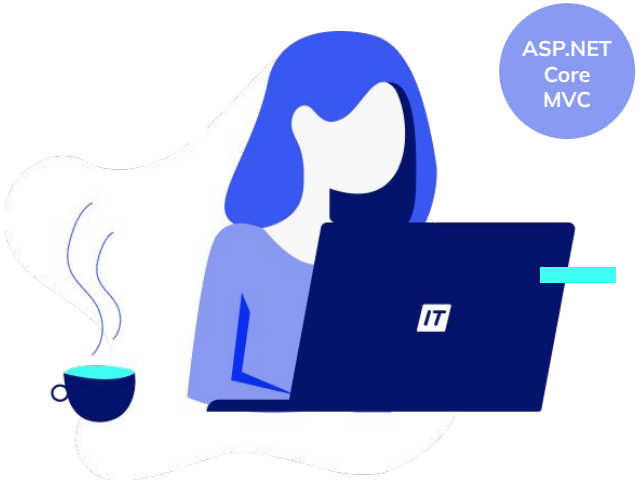
Este modelo de programación se introdujo por primera vez con **ASP.NET Core** y es una alternativa al modelo de **MVC.NET Framework**. Mientras que en **MVC** tenemos un controlador que recibe solicitudes, un modelo y una vista que genera la respuesta, con **Razor Pages** es diferente. La solicitud va directamente a una página que, generalmente, se encuentra en la carpeta *Páginas*.

Cada archivo de la página de **Razor** tiene un archivo ***.cshtml.cs*** adjunto que contiene todos los métodos, controladores de modelo, y lógica.

Contiene una clase que hereda de **`PageModel`** que inicializará el modelo dentro del método llamado **`OnGet()`**. El método `OnGet()` también es el lugar para obtener datos de la base de datos. Cada propiedad pública dentro del modelo se puede mostrar en la página, por medio de la sintaxis de **Razor**.

ASP.NET Core MVC

Cuando se crea un sitio web, **ASP.NET Core MVC** separa su código en tres partes: ***modelo***, ***vista***, y ***controlador***. Esta separación de modelo, vista y código de controlador, asegura que las aplicaciones **MVC** tengan una estructura lógica, incluso para los sitios más complejos. También mejora la capacidad de prueba de la aplicación.



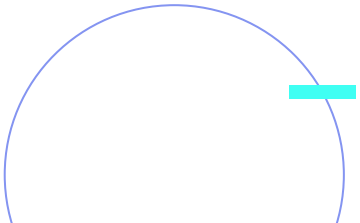
Modelo de programación web Página Razor

Este ejemplo de código muestra cómo se ve el código subyacente de la **página Razor**.

La sintaxis de la **página Razor** se parece mucho a una **vista de Razor**, que se tratará en el Módulo de las "Vistas". Lo que lo hace diferente es la directiva **@page** que debe estar en la primera línea del archivo ***.cshtml***.

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace RazorPagesExample.Pages
{
    0 referencias
    public class HomePageModel : PageModel
    {
        1 referencia
        public string Description { get; set; }
        1 referencia
        public string MoreInfo { get; set; }
        0 referencias
        public void OnGet()
        {
            Description = "Your application description is here.";
            MoreInfo = "Your application extra information is here.";
        }
    }
}
```

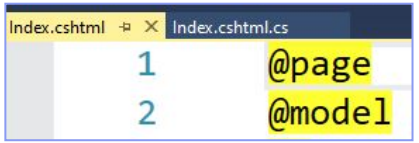


@page

@page convierte el archivo en una **página de Razor**, lo que significa que maneja las solicitudes directamente sin pasar por un controlador como en el modelo de programación **MVC**.

La sintaxis de **Razor** se puede utilizar dentro de la **página Razor**. Permite insertar las propiedades del modelo y otras fuentes en el marcado **HTML**. La sintaxis de Razor comienza con el carácter **@**. Para mostrar propiedades del modelo dentro del marcado HTML, se usará **@model**.

@model tendrá una referencia a las propiedades que inicializó dentro del método **OnGet**.




```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace WebAppTest
{
    5 referencias
    public class IndexModel : PageModel
    {
        0 referencias
        public void OnGet()
        {
        }
    }
}
```

Este ejemplo de código muestra cómo editar la **página Razor** y mostrar valores del modelo dentro de la página:

ASP.NET Core MVC

educación IT
CODE YOUR FUTURE

MVC: Modelos, vistas y controladores

Al desarrollar aplicaciones **ASP.NET Core MVC**, es necesario distinguir entre tres conceptos: *modelos*, *vistas* y *controladores*.

Modelos y datos

Un modelo es una clase .NET que representa un objeto manejado por su sitio web. Por ejemplo, el modelo de una aplicación de comercio electrónico puede incluir una clase de modelo de producto con propiedades como `ID de producto`, `número de pieza`, `número de catálogo`, `nombre`, `descripción` y `precio`.

Controladores y acciones

Un controlador es una clase .NET que responde a las solicitudes del navegador web en una aplicación MVC. Por lo general, hay una clase de controlador para cada clase de modelo. Los controladores incluyen **acciones**, que son métodos que se ejecutan en respuesta a una solicitud de usuario. Por ejemplo, el controlador del producto puede incluir una acción de compra que se ejecuta cuando el usuario hace clic en *Agregar al carrito* en su aplicación web. Los controladores heredan de la clase base cuyo nombre es `Microsoft.AspNetCore.Mvc.Controller`.

Vistas

Una vista es un archivo ***.cshtml*** que incluye tanto el marcado **HTML** como el código de programación. Un motor de vista interpreta los archivos de la vista, ejecuta el código del lado del servidor y también presenta **HTML** al navegador web. **Razor** es el motor de vista predeterminado en **ASP.NET Core MVC**.

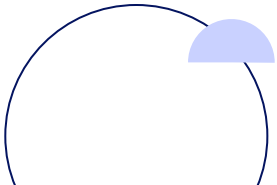
Las siguientes líneas de código son parte de una vista **ASP.NET Core MVC** y usan la sintaxis de **Razor**. El símbolo **@** delimita el código del lado del servidor. Parte de una vista de **Razor**.

```
<h2>Details</h2>
<fieldset>
    <legend>Comment</legend>
    <div class="display-label">
        @Html.DisplayNameFor(model => model.Subject)
    </div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Subject)
    </div>
    <div class="display-label">
        @Html.DisplayNameFor(model => model.Body)
    </div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Body)
    </div>
</fieldset>
```

Solicitar ciclo de vida

El ciclo de vida de la solicitud comprende una serie de eventos que ocurren cuando se procesa una solicitud web. Los siguientes pasos ilustran el proceso que siguen las aplicaciones **MVC** para responder a una solicitud de usuario típica.

Importante: La solicitud es para los detalles de un producto con el ID "1".



Ciclo de vida de la solicitud:

1. El usuario solicita la dirección web: `http://www.adventure-works.com/product/display/1`.
2. El motor de enrutamiento **MVC** examina la solicitud y determina que debe reenviar la solicitud al controlador del producto y la acción de visualización.
3. La acción **`Mostrar`** en el controlador de **`Producto`** crea una nueva instancia de la clase de modelo **`Producto`**.

4. La clase de modelo de **`Producto`** consulta la base de datos para obtener información sobre el producto con **ID “1”**.
5. La acción **`Mostrar`** también crea una nueva instancia de la vista **`Presentación del producto`** y le pasa el modelo **`Producto`**.
6. El motor de vista de **Razor** ejecuta el código del lado del servidor en la vista de visualización del producto para representar el **HTML**. En este caso, el código del lado del servidor inserta propiedades: `Título`, `Descripción`, `Número de catálogo` y `Precio` en **HTML**.
7. La página **HTML** completa se devuelve al navegador para su visualización.

Elige entre .NET Core y .NET Framework

¿Cuándo usar .NET Core para su aplicación?

Se debe usar **.NET Core** en el caso de que la aplicación necesite ejecutarse en **múltiples plataformas** como Windows, Linux y macOS, porque es compatible con esos tres sistemas operativos.

Visual Studio 2019 o **Visual Studio Code** se pueden usar como **IDE**. Muchos editores de terceros, como el llamado *Sublime Text*, funcionan con **.NET Core**.

También, puede evitar el uso de editores de código por completo y usar las **herramientas CLI** de .NET Core en su lugar. Está disponible para todas las plataformas compatibles.

Aplicación web ASP.NET Core

Plantillas de proyecto para crear aplicaciones web de ASP.NET Core y API web para Windows, Linux y macOS con .NET Core o .NET Framework. Cree aplicaciones web con Razor Pages, MVC o aplicaciones de una sola página (SPA) con Angular, React o React+Redux.

C# | Linux | macOS | Windows | Nube | Servicio | Web

Usa **.NET Core** si la aplicación necesita ejecutarse en **múltiples plataformas**.

Discusión: elige entre ASP.NET 4.x y ASP.NET Core

¿Qué versión de **ASP.NET** (**ASP.NET 4.x** o **ASP.NET Core**) usarías en estos escenarios?

- Aplicación de alto rendimiento para el mercado de valores.
- Actualizar un sitio web existente escrito en formularios web.
- Proyecto de código abierto que se ejecuta en diferentes marcos de trabajo.

Aplicación de alto rendimiento para Bolsa de Valores

Quieres desarrollar una aplicación web que muestra las últimas tendencias del mercado económico. Tu aplicación debe tener un alto rendimiento y debe ser muy rápida para obtener la información más reciente de diversas fuentes.

Actualizar un sitio web existente escrito en formularios web

Trabajas en una gran organización que ha estado utilizando Web Forms durante muchos años. Tienen un sitio web existente escrito en formularios web. Se te pide que amplíes su funcionalidad y agregues algunas páginas.

Proyecto de código abierto que se ejecuta en macOS

Tienes una idea para un nuevo sistema de gestión de contenido. Quieres que sea un proyecto de código abierto y desarrollarlo en tu computadora Mac con macOS y eliges usar Visual Studio Code.

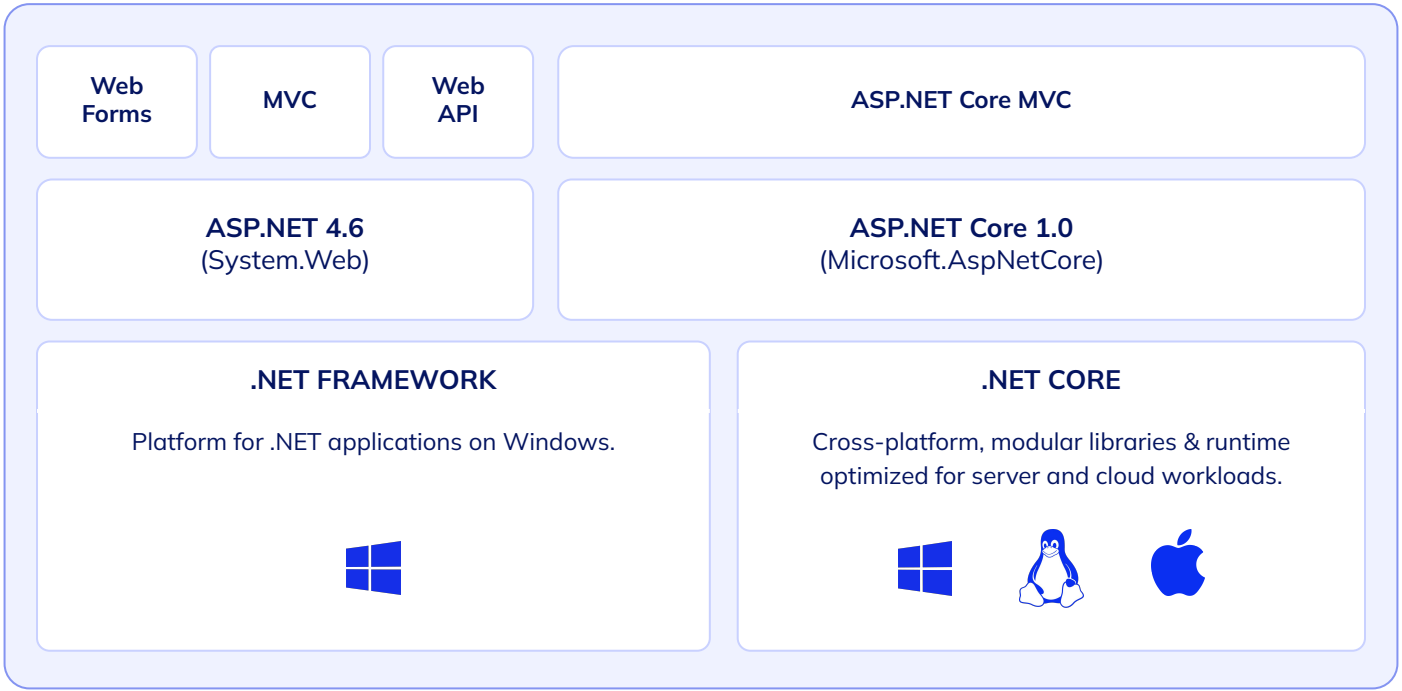
Elige entre .NET Core y .NET Framework

Debes usar .NET Core cuando:

- Quieres que tu código se ejecute en varias plataformas.
- Deseas crear **microservicios**.
- Quieres usar contenedores ***Docker***.
- Tu objetivo es lograr un sistema escalable de alto rendimiento.

Debes utilizar .NET Framework cuando:

- Quieres ampliar una aplicación existente que usa **.NET Framework**.
- Deseas utilizar paquetes ***NuGet*** o bibliotecas **.NET** de terceros que no son compatibles con **.NET Core**.
- Deseas utilizar tecnologías .NET que no son compatibles con **.NET Core**.
- Quieres utilizar una plataforma que no sea compatible con **.NET Core**.


Web Forms
MVC
Web API
ASP.NET Core MVC
ASP.NET 4.6
(System.Web)
ASP.NET Core 1.0
(Microsoft.AspNetCore)
.NET FRAMEWORK
Platform for .NET applications on Windows.
.NET CORE
Cross-platform, modular libraries & runtime optimized for server and cloud workloads.

.NET estándar

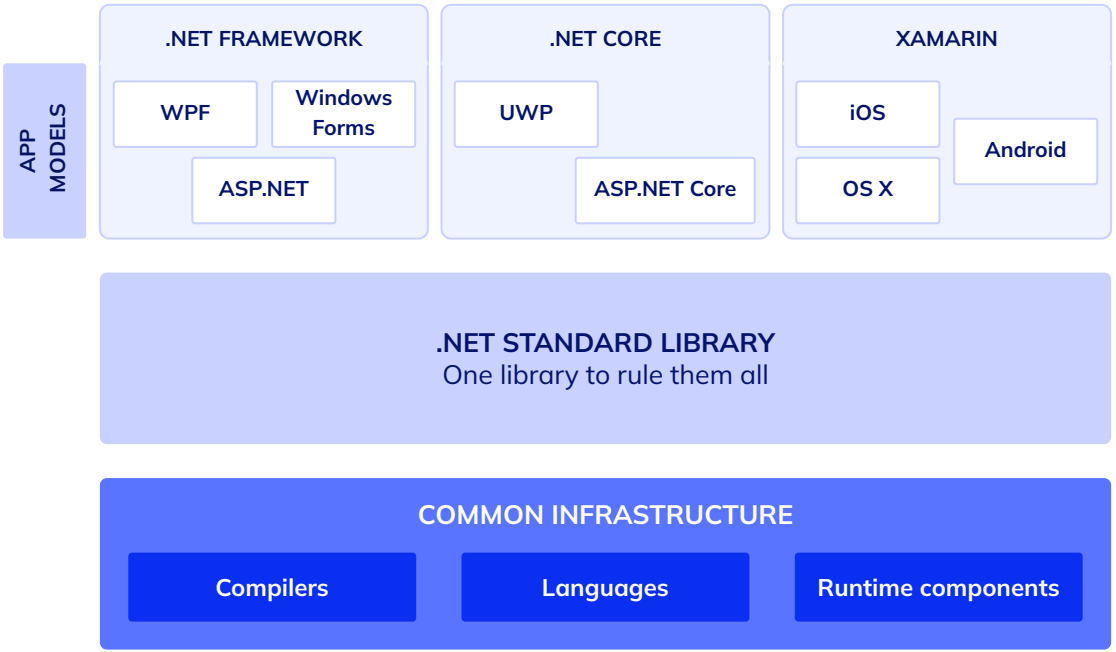
.NET estándar

.NET Standard es una especificación formal del API de .NET. Está destinado a estar **disponible en todas las implementaciones de .NET** y su fin es crear una **unificación del ecosistema .NET**.

Te permite crear código portátil en todas las implementaciones de .NET. **Cada implementación de .NET tiene como objetivo una versión específica de .NET Standard.**

Un número de versión más alto significa que las versiones anteriores también son compatibles, pero tiene algunas extensiones.

Por ejemplo: .NET Framework 4.6.2 implementa .NET Standard 1.5, significa que expone todas las API definido en .NET Standard versiones 1.0 a 1.5. De manera similar, .NET Core 2.0 implementa .NET Standard 2.0 y por esa razón expone todas las API definidas en .NET Standard versiones 1.0 a 2.0. Esto también significa que .NET Core 2.0 admite el código escrito en .NET Framework 4.6.2 porque implementa .NET Estándar 1.5.

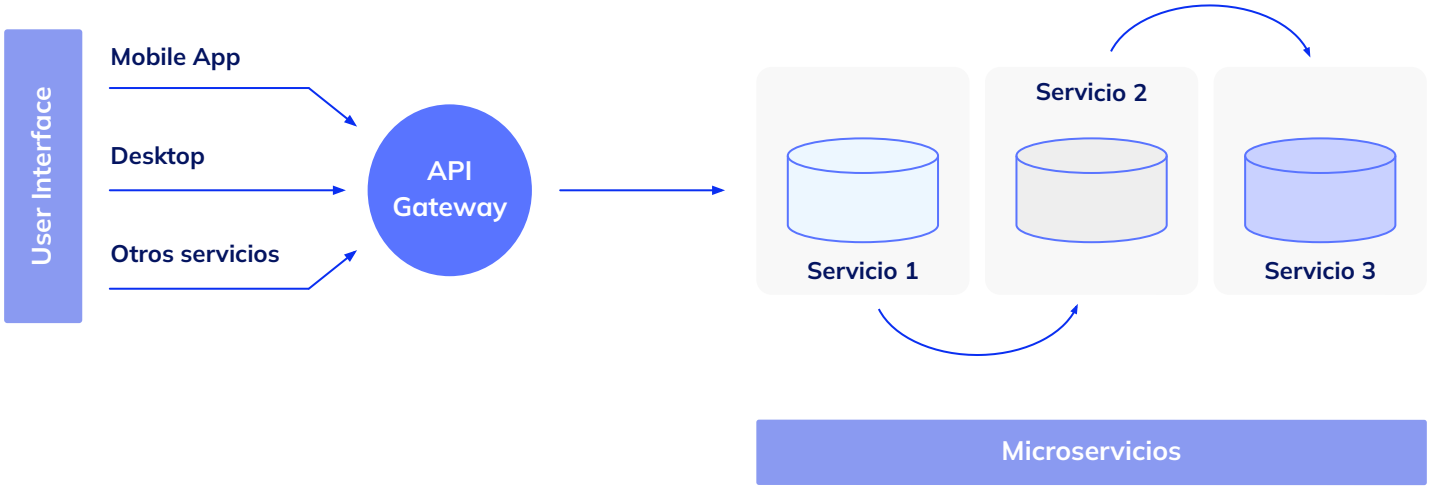

APP MODELS
.NET FRAMEWORK
WPF
Windows Forms
ASP.NET
.NET CORE
UWP
ASP.NET Core
XAMARIN
iOS
Android
OS X
.NET STANDARD LIBRARY
One library to rule them all
COMMON INFRASTRUCTURE
Compilers
Languages
Runtime components

Microservicios

Microservicios

.NET Core se creó teniendo en cuenta la arquitectura de **microservicios**. En **.NET Framework**, la capa de servicios contiene toda la funcionalidad de la aplicación con referencias a bibliotecas y componentes principales. En la arquitectura de **microservicios**, la capa de servicios se divide en varios componentes independientes.

El **microservicio** es un enfoque para crear pequeños servicios, cada uno de los cuales se ejecuta en su propio espacio y puede comunicarse vía mensajes. Estos son servicios independientes que llaman directamente a su propia base de datos.


User Interface
Mobile App
Desktop
Otros servicios
API Gateway
Servicio 1
Servicio 2
Servicio 3
Microservicios

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!