CAPÍTULO 1

# Introducción a .NET y C#

Para aquellos nuevos al lenguaje creo que esta parte es muy importante de entender antes de pasar a la materia como tal.

Lo primero que tenemos que entender es la diferencia entre C# y .NET;

C#, al igual que F# o Visual Basic con lenguajes de programación, los cuales están dentro de .NET, un framework que opera sobre estos lenguajes.

Para ser sincero el tema de los nombres que da Microsoft a sus frameworks o versiones es algo que personalmente he sido muy crítico, ya que es bastante lioso. Aquí voy a explicar todo.

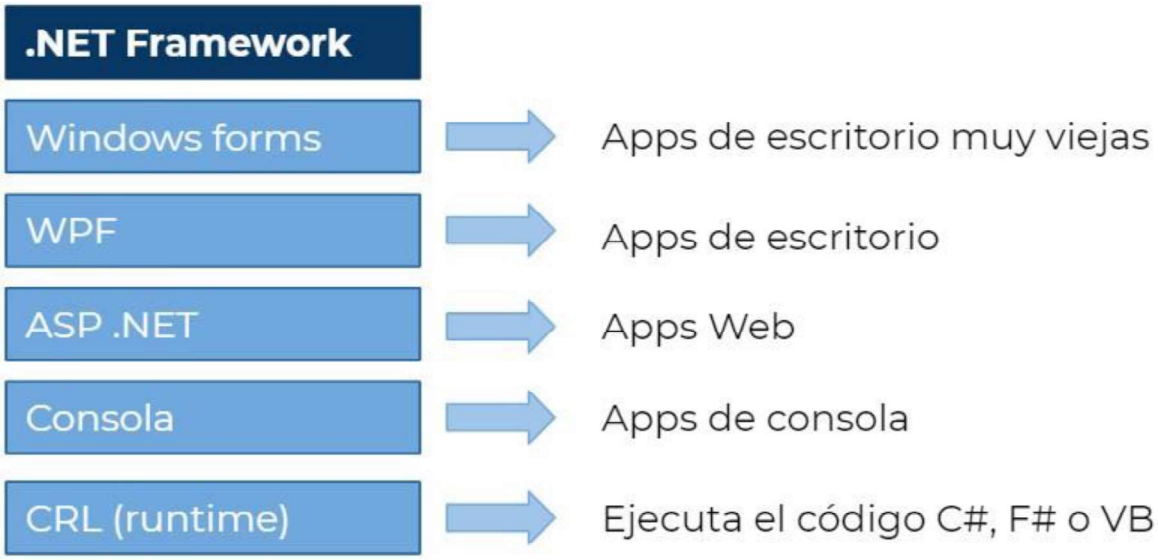
## NET Framework

Para ello vamos atrás en la historia, en febrero de 2002, Microsoft lanzaba la primera versión de .NET Framework, el cual era código propietario, lo que quiere decir, de pago y privado. Junto con la primera versión del framework, Microsoft lanzó la primera versión de Visual Studio, también de pago. Este software permite crear aplicaciones para Windows, en entornos Windows y constituye la base de lo que hacemos hoy en día.

Podemos entender .NET Framework como la base o motor que entiende el código C#, F#, o Visual Basic y lo convierte a código máquina.

Todos los lenguajes de la familia .NET eran y siguen siendo hoy en día, lenguajes de alto nivel, esto quiere decir que es el framework quien se encarga de configurar por ejemplo el recolector de basura (o "GC" en inglés).

De aquella primera versión .NET Framework fue evolucionando y llegó a permitirnos crear aplicaciones de escritorio, aplicaciones web o de consola. El hecho de programar con Windows nos permite acceder a todas las APIs de Windows o de Microsoft, como pueden ser las del paquete Office, lo que dio a .NET una ventaja competitiva sobre los rivales en el entorno empresarial.



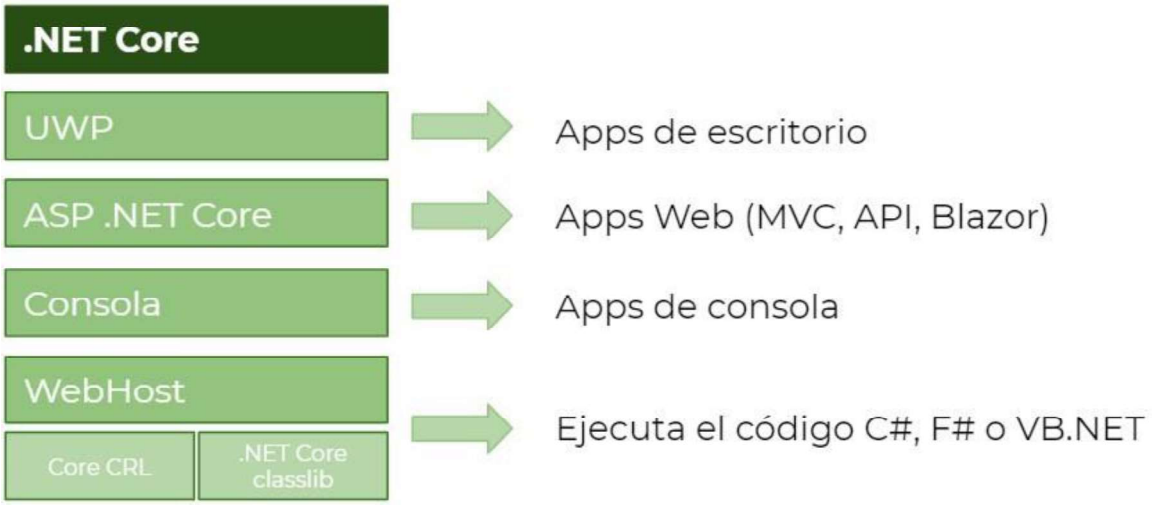

# NET Core

Avanzamos unos años, Microsoft se dio cuenta de que .NET Framework y sus lenguajes no podían competir de tú a tú con los lenguajes populares

del momento y en vez de dar la batalla por perdida, decidió hacer una acción que nadie esperaba.

En 2016, Microsoft lanzó .NET Core, una versión más rápida y ligera que su predecesor, así como Open Source, aunque mucho más importante fue el detalle final, su capacidad de ser multiplataforma. Con este cambio .NET pasaría a poder ejecutarse en cualquier tipo de máquina.

Este cambio no consistió en un simple copiar y pegar de .NET Framework que permitiese correr en múltiples plataformas, sino que en la gran mayoría de casos tuvieron que hacer una refactorización o reescritura de código completa.
Al margen de algunos cambios técnicos sobre cómo se ejecuta cada tipo de aplicación, estas seguían siendo las mismas, aplicaciones de escritorio, web y de consola:

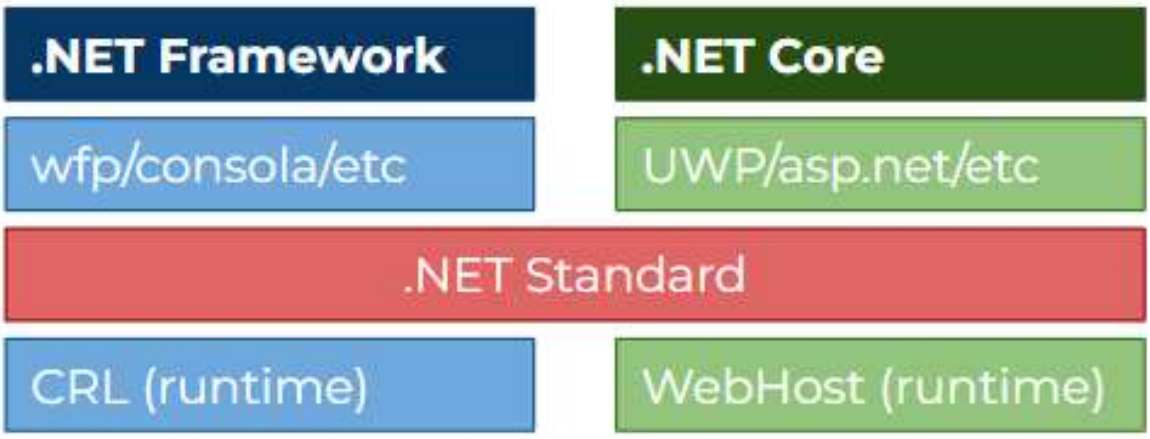



Este cambio parece algo pequeño, pero no es así, antiguamente necesitabas un servidor Windows y una máquina Windows para correr tus aplicaciones. Desde la llegada de .NET Core esto no es necesario ya que las aplicaciones corren en Linux, y si lo piensas bien, resulta una mejora significativa ya que un servidor Linux no tiene coste de licencia, mientras que los servidores con Windows server suelen resultar mucho más caros.

Microsoft se dio cuenta de que el cambio era muy brusco, ya que no solo cambiaron las capas exteriores, sino mucho código interno también es diferente, por lo que junto a .NET Core lanzaron un framework llamado .NET Standard.

# NET Standard

La migración desde .NET Framework a .NET Core resultó ser mucho más compleja de lo que se imaginaron de primeras, así que decidieron crear una serie de paquetes compatibles entre ambos sistemas. Aunque no sea evidente a primera vista, estos paquetes son de gran utilidad para la migración de código antiguo, ya que estos paquetes funcionan como punto de unión entre aplicaciones escritas con .NET Framework y aplicaciones escritas con .NET Core.

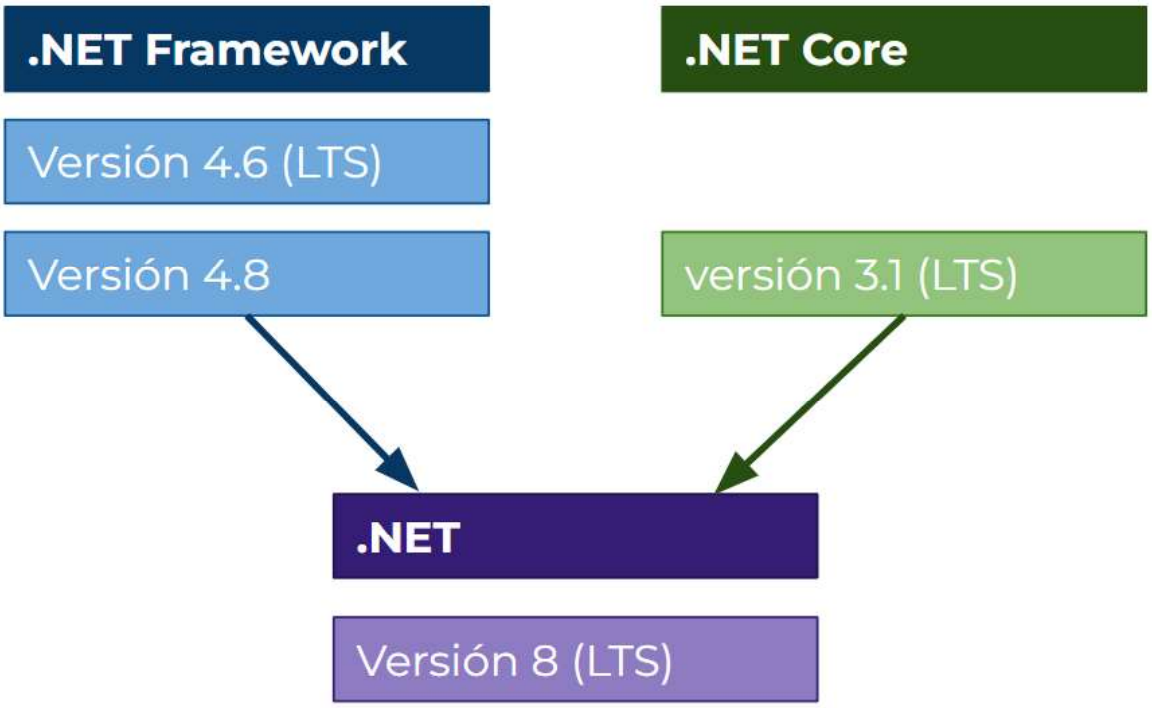



La decisión sobre si debemos crear o utilizar paquetes .NET Standard se puede resumir a si lo que estás haciendo necesita compatibilidad con el sistema .NET Framework o no, si no necesitas dicha compatibilidad puedes ignorar estas versiones y saltar a las más modernas.

# .NET

Finalmente, para rizar más el rizo, Microsoft decidió poner punto final a .NET Framework, pese a que la última versión (.NET Framework 4.8) va a seguir teniendo soporte y siendo mantenido muchos años (no tiene fecha final, simplemente es parte de Windows), no van a seguir desarrollando nuevas funcionalidades.

Para todos los que quieran nuevas funcionalidades van a tener que migrar a .NET, también denominado *dotnet*, en lugar de a .NET Core que hemos visto, sino a .NET (sin coletilla "Core") y es que en 2020 Microsoft anunció .NET 5 quitando el Core de su nombre, y así unificando los frameworks que corren encima de los lenguajes de programación.




La versión actual es .NET 8 la cual es LTS o "Long Term Support". Es importante diferenciar las versiones que son LTS de las que no, esto es debido a que las versiones LTS son así porque tienen un mayor soporte y vida que las versiones sin LTS. Si que es verdad que no van a traer

nuevos cambios en funcionalidades, pero la fecha final de soporte indica la fecha en la que va a dejar de recibir actualizaciones de seguridad.

Por poner un ejemplo claro, la versión .NET 6 fue lanzada en noviembre de 2021 y tiene soporte hasta noviembre de 2024, 3 años (36 meses), mientras que .NET 7 que fue lanzada en noviembre de 2022 termina el soporte en mayo de 2024 (18 meses).

Muchas empresas no permiten utilizar versiones que no son LTS por este motivo.

# El entorno de desarrollo de .NET

Aunque técnicamente es posible programar C# utilizando el bloc de notas, esta práctica no es productiva y la experiencia es nefasta. En el punto anterior he mencionado que Microsoft lanzó junto con los lenguajes Visual Studio.

Visual Studio es un IDE (entorno de desarrollo integrado) que está pensado para facilitar el desarrollo de software, principalmente de los lenguajes de Microsoft, aunque tiene soporte para muchos más (JavaScript, Python, C, entre otros); Este soporte nos proporciona funcionalidades como el autocompletado, detección de errores de sintaxis en el código, o incluso una paleta de colores que nos permite diferenciar diferentes elementos del código con un solo vistazo.

Tiene muchas más funcionalidades, como integración con Git, integración con el modelo de la base de datos, y muchas más, pero todo eso se escapa del objetivo de este libro.

Durante este curso voy a utilizar Visual Studio, actualmente es la versión 2022, no te preocupes no sacan una cada año. Puedes descárgate la última versión desde este enlace:

https://visualstudio.microsoft.com/es/vs/community/
o buscando “Descargar Visual Studio”

Asegúrate de descargarte la Community Edition que es la versión gratuita, siempre y cuando no la utilices para fines comerciales.

Si trabajas con Linux o con Mac personalmente te recomiendo que te descargues Jetbrains Rider, es un IDE similar a Visual Studio, pero con una interfaz diferente y de pago, con una versión de prueba de 30 días.

Hasta mediados de 2023, Visual Studio para Mac era una opción, pero Microsoft decidió cortar el desarrollo y la alternativa es utilizar una extensión de Visual Studio Code llamada DevKit que está bien, pero no es ni de cerca algo similar a un IDE.

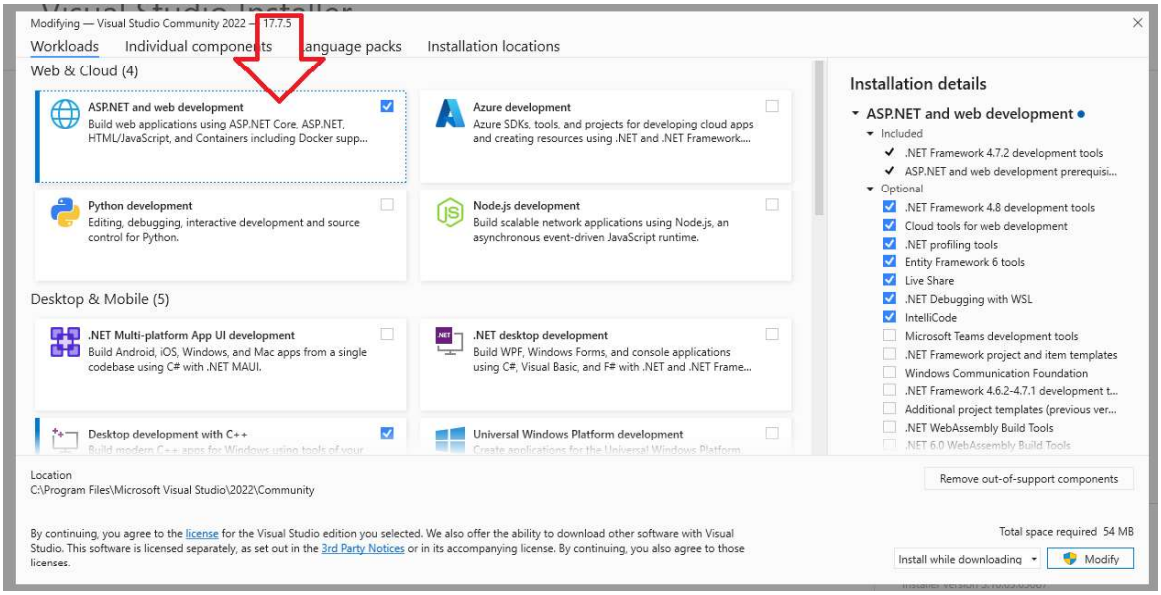
Puedes descargar Jetbrains Rider desde aquí:

https://www.jetbrains.com/es-es/rider/download/
o buscando “Descargar Jetbrains Rider”

Es el IDE que uso en mi día a día en el entorno laboral.

Después, tienes que asegurarte de que tienes descargado el SDK de .NET:
https://dotnet.microsoft.com/en-us/download

Aunque si vas a utilizar Visual Studio, puedes utilizar el instalador e instalar este paquete:




El cual te lo instalará.

Una vez está instalado puedes comprobar que tienes el SDK y el Runtime si ejecutas en la terminal o en Powershell los siguientes comandos:

```
C:\Users\ivana> dotnet --list-sdks
8.0.201 [C:\Program Files\dotnet\sdk]
PS C:\Users\ivana> dotnet --list-runtimes
Microsoft.AspNetCore.App 8.0.2 [C:\Program Files\dotnet\shared\Microsoft.AspNetCore.App]
Microsoft.NETCore.App 8.0.2 [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
Microsoft.WindowsDesktop.App 8.0.2 [C:\Program
Files\dotnet\shared\Microsoft.WindowsDesktop.App]
```

# ¿Qué es NuGet?

Igual de importante que el propio .NET que nos da Microsoft lo son las librerías de terceros que completan la funcionalidad.

NuGet es el gestor de paquetes predeterminado de .NET. Es el espacio virtual donde los desarrolladores que hacemos código libre subimos las librerías para que puedan accedidas y utilizadas de una forma sencilla.

Para instalar estos paquetes lo normal es hacer botón secundario ya bien sea en la solución o en el propio proyecto y ahí tenemos una opción del menú que dice “administrar paquetes NuGet”. Este menú dispone de un buscador que por defecto busca en nuget.org

Alternativamente podemos utilizar la línea de comandos para instalar dichos paquetes, ya que cuando instalamos Visual Studio nos viene con lo que se denomina “Package Manager”.

En este libro, todo lo que vamos a necesitar está disponible en NuGet, pero en el entorno empresarial es muy común tener un repositorio privado para mantener los paquetes comunes de la empresa.

Gracias a NuGet podemos agilizar el desarrollo de proyectos en .NET ya que es el lugar central donde vamos a tener el código que va a ser reutilizado.