



Creación de una Aplicaciones Web y API REST con ASP.NET Core

Francisco de Borja Cabeza Rozas

5



Francisco de Borja Cabeza Rozas




- Arquitecto de Software
- Microsoft Certified Trainer (MCT)
- Microsoft Certified Solution Expert (MCSE)
Cloud Platform and Infrastructure
- Microsoft Certified Solutions Associate (MCSA)
Cloud Platform and Web Applications
- Microsoft Certified Solution Developer (MCSD)
App Builder, Web Applications y Windows Store Apps



Microsoft
CERTIFIED
Trainer



6

 Microsoft

1

Introducción a .NET Core

2

Desarrollo de Web APIs

3

Desarrollo de Aplicaciones MVC

4


EF Core y LINQ


5

Pruebas Unitarias



7

 Microsoft



Introducción a .NET Core

8

ASP.NET Core

ASP.NET es un Framework de desarrollo para crear aplicaciones Web y APIRest en la Plataforma .NET de Microsoft:

- versión de código abierto de ASP.NET
- multiplataformas, se ejecuta en macOS, Linux y Windows
- tiene más de 100.000 contribuciones en GitHub
- 3.700 empresas colaboran y contribuyen
- Red Hat y Microsoft colaboran para garantizar que .NET funcione bien en RHEL
- las aplicaciones se pueden transformar en contenedores Docker

El rendimiento es un enfoque clave de ASP.NET Core.

Es más rápido que otros populares Frameworks Web según las pruebas comparativas de [TechEmpower](https://www.techempower.com/).



9

¿Por qué elegir ASP.NET Core?

Millones de desarrolladores usan o han usado ASP.NET 4.x para crear aplicaciones web.

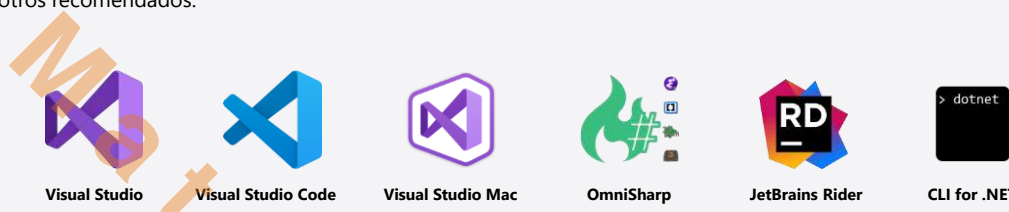
ASP.NET Core es un nuevo diseño de ASP.NET 4.x que incluye cambios en la arquitectura que dan como resultado un marco más sencillo y modular. ASP.NET Core ofrece las siguientes ventajas:

- Formato unificado para crear API web y una interfaz de usuario web.
- Permite la integración de Pruebas Unitarias.
- Capacidad para desarrollarse y ejecutarse en Windows, macOS y Linux.
- De código abierto y centrado en la comunidad.
- Integración con Frameworks del lado cliente modernos y flujos de trabajo de desarrollo.
- Compatibilidad con el hospedaje de servicios de llamada a procedimiento remoto (RPC) con gRPC.
- Inserción de dependencias integrada.
- Una canalización de solicitudes HTTP ligera, modular y de alto rendimiento.
- Capacidad para hospedarse en Kestrel, IIS, HTTP.sys, Nginx, Apache, Docker, ...
- Control de versiones en paralelo.

10

Entorno de Desarrollo Integrado (IDE)

La Plataforma .NET y por consiguiente ASP.NET Core cuenta con diferentes IDE algunos desarrollados por Microsoft y otros recomendados.



11

Visual Studio

Visual Studio es una herramienta de desarrollo eficaz que permite completar todo el ciclo de desarrollo.

Es un entorno de desarrollo integrado (IDE) completo que puede usar para escribir, editar, depurar y compilar el código y, luego, implementar la aplicación.

Aparte de la edición y depuración del código, Visual Studio incluye compiladores, herramientas de finalización de código, control de código fuente, extensiones y muchas más características para mejorar cada fase del proceso de desarrollo de software.



12

Instalación de Visual Studio

Para empezar, descargamos el Instalador de Visual Studio y procedemos con su instalación.

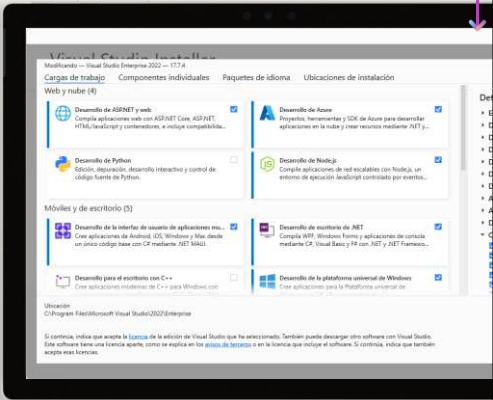
Finalizada la instalación de *Programa Instalador*, se ejecuta y comenzaremos seleccionando las cargas de trabajo que necesitamos en función de los tipos de proyectos que vamos a desarrollar.

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

Las cargas de trabajo que necesitamos para proyectos ASP.Net Core

- Desarrollo de ASP.NET y web
- Desarrollo de escritorio de .NET (recomendó)

También puede seleccionar el idioma del entorno de desarrollo e instalar componente individuales como Git para Windows y Herramientas de LINQ para SQL



Instalación de Visual Studio Code

Descargamos e instalamos Visual Studio Code.

Finalizada la instalación instalamos las extensiones que añaden características al entorno de desarrollo.

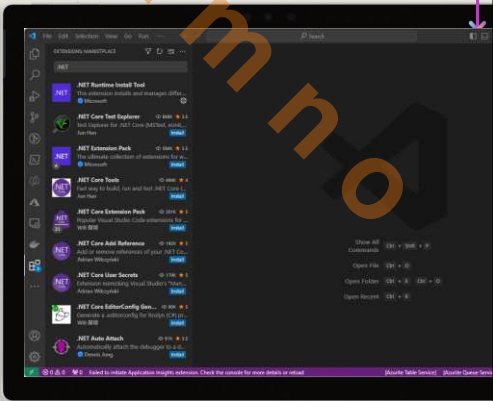
Las extensiones de *VSCo*de son paquetes de código que se ejecutan dentro del entorno de desarrollo.

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

Las extensiones que necesitamos para proyectos ASP.Net Core

- C# y C# Dev Kit
- vscode-icons (recomendó)

También se recomienda alguna extensión que ofrezca *helpers* para la escritura de código, y funciones complementarias para Git.



¿Qué es el SDK de .NET?

El SDK de .NET es un conjunto de bibliotecas y herramientas que permiten a los desarrolladores crear aplicaciones y bibliotecas de .NET.

Contiene los siguientes componentes que se usan para compilar y ejecutar aplicaciones:

- La CLI de .NET
- El entorno de ejecución y las bibliotecas de .NET

.NET es gratuito, de código abierto y es un proyecto de [.NET Foundation](#). Microsoft y la comunidad de GitHub mantienen .NET en varios [repositorios](#).

15

Versiones del SDK de .NET Core

Alumnos:

Revisar en la web las versiones admitidas y las versiones sin soporte

La versión más reciente de .NET es la 7.

Hay dos tipos de versiones admitidas, las versiones de soporte técnico de larga duración (LTS) o el soporte de duración estándar (STS).

La calidad de todas las versiones es la misma. La única diferencia es la duración del soporte técnico.

- Las versiones LTS obtienen soporte técnico y revisiones gratuitas durante 3 años.
- Las versiones STS obtienen soporte técnico y revisiones gratuitas durante 18 meses.

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

16

Instalación con Windows Installer

La página de descarga de .NET proporciona ejecutables de *Windows Installer*.

A partir de noviembre de 2021, no se puede cambiar la ruta de instalación de .NET con el paquete de *Windows Installer*. Para instalar .NET en otra ruta de acceso, use los scripts **dotnet-install**.

Si quiere instalar .NET de forma silenciosa, como en un entorno de producción o para admitir la integración continua, use las expresiones siguientes:

- **/install** instala el SDK de .NET
- **/quiet** impide que se muestren interfaces de usuario y solicitudes
- **/norestart** suprime los intentos de reinicio

```
dotnet-sdk-7.0.100-win-x64.exe /install /quiet /norestart
```

<https://dotnet.microsoft.com/es-es/download>

17

Instalación con Administrador de paquetes de Windows

Puede instalar y administrar .NET a través del servicio Administrador de paquetes de Windows mediante la herramienta **winget**.

Si va a instalar .NET en todo el sistema, instale con privilegios administrativos.

Instalación del SDK

El SDK de .NET permite desarrollar aplicaciones con .NET. Si instala el SDK de .NET, no es necesario instalar los entornos de ejecución correspondientes. Para instalar el SDK de .NET, ejecute el comando siguiente:

```
winget install Microsoft.DotNet.SDK.7
```

<https://dotnet.microsoft.com/es-es/download>

Instalación de la instancia en tiempo de ejecución

Para Windows, hay tres entornos de ejecución de .NET que puede instalar.

.NET Desktop Runtime

```
winget install Microsoft.DotNet.DesktopRuntime.7
```

ASP.NET Core Runtime

```
winget install Microsoft.DotNet.AspNetCore.7
```

Runtime de .NET

```
winget install Microsoft.DotNet.Runtime.7
```

18

Instalación mediante la automatización de PowerShell

Los scripts de **dotnet-install** se usan para la automatización de CI y las instalaciones que no son de administrador del entorno de ejecución.

Puede elegir una versión concreta especificando el modificador **Channel**. Incluya el modificador **Runtime** para instalar un entorno de ejecución, de lo contrario, el script instala el SDK.

Instale el SDK omitiendo el modificador **-Runtime**.

```
dotnet-install.ps1 -Channel STS
dotnet-install.ps1 -Channel 7.0 -Runtime aspnetcore
```

<https://dotnet.microsoft.com/es-es/download>

19

Buscar carpetas de instalación

Es posible que .NET esté instalado, pero no se haya agregado a la variable PATH del sistema operativo o el perfil de usuario. En este caso, es posible que no funcionen los comandos de las secciones anteriores.

Como alternativa, puede comprobar que existen las carpetas de instalación de .NET.

Al instalar .NET desde un instalador o un script, la instalación se efectúa en una carpeta estándar. La mayor parte del tiempo, el instalador o el script que usa para instalar .NET le ofrece la opción de realizar la instalación en otra carpeta. Si decide instalar en una carpeta diferente, ajuste el inicio de la ruta de acceso de la carpeta.

- **archivo ejecutable de dotnet** C:\Archivos de programa\dotnet\dotnet.exe
- **SDK de .NET** C:\Archivos de programa\dotnet\sdk\{versión}\
- **Entorno de ejecución .NET** C:\Archivos de programa\dotnet\shared\{tipo-de-runtime}\{versión}\

<https://dotnet.microsoft.com/es-es/download>

20

Cómo comprobar que .NET Core ya está instalado

```
C:\>dotnet --list-sdks
```

```
3.1.424 [C:\program files\dotnet\sdk]
5.0.100 [C:\program files\dotnet\sdk]
6.0.402 [C:\program files\dotnet\sdk]
7.0.100 [C:\program files\dotnet\sdk]
```

```
C:\>dotnet --list-runtimes
```

```
Microsoft.AspNetCore.App 3.1.30 [C:\Program Files\dotnet\shared\Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 6.0.10 [C:\Program Files\dotnet\shared\Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 7.0.0 [C:\Program Files\dotnet\shared\Microsoft.AspNetCore.App]
Microsoft.NETCore.App 3.1.30 [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
Microsoft.NETCore.App 5.0.17 [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
Microsoft.NETCore.App 6.0.10 [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
Microsoft.NETCore.App 7.0.0 [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
Microsoft.WindowsDesktop.App 3.1.30 [C:\Program Files\dotnet\shared\Microsoft.WindowsDesktop.App]
Microsoft.WindowsDesktop.App 6.0.10 [C:\Program Files\dotnet\shared\Microsoft.WindowsDesktop.App]
Microsoft.WindowsDesktop.App 7.0.0 [C:\Program Files\dotnet\shared\Microsoft.WindowsDesktop.App]
```



21

Administración de plantillas de proyectos y otros elementos

.NET proporciona un sistema de plantillas para iniciar proyectos. Los usuarios pueden instalar o desinstalar los paquetes que contienen plantillas desde *NuGet*, un archivo de paquete *NuGet* o un directorio del sistema de archivos.

Los paquetes de plantilla se instalan mediante el comando [dotnet new install](#) del SDK.

```
dotnet new install Microsoft.DotNet.Web.Spa.ProjectTemplates::2.2.6
```

```
dotnet new install c:\code\nuget-packages\Some.Templates.1.0.0.nupkg
```

```
dotnet new install c:\code\nuget-packages\some-folder\
```

Si hemos instalado diferentes versiones del SDK secuencialmente (por ejemplo, si ha instalado el SDK 6.0, después el SDK 7.0, etc.), tendrá instaladas las plantillas de cada SDK.

Con .NET, podemos crear e implementar plantillas personales que generan proyectos, archivos y recursos.

22

Configuraciones en Visual Studio

Podemos personalizar Visual Studio de varias maneras para que se ajuste mejor a nuestro propio estilo.

Al iniciar sesión en Visual Studio en varios equipos con la misma cuenta de personalización, se puede sincronizar la configuración entre todos los equipos.

Podemos usar el asistente para exportar e importar la configuración de entorno o importar categorías específicas de configuración, incluyendo la recibida previamente de otra persona.

También podemos usar el asistente para restablecer el entorno a una de las colecciones predeterminadas de configuración.

Restablecer todas las configuraciones

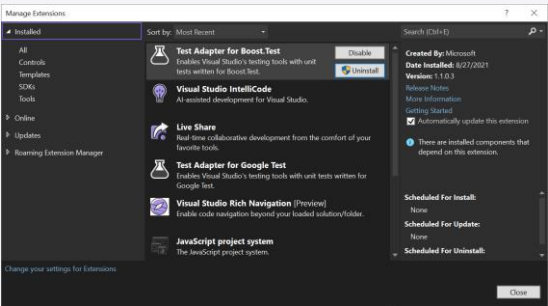
- 1. En la barra de menús, elija **Herramientas>Importar y exportar configuraciones**.



- 2. En el **Asistente para importar y exportar configuraciones**, seleccione **Restablecer todas las configuraciones** y luego **Siguiente**
- 3. En la página **Guardar configuración actual**, seleccione **Sí**, guarde la configuración actual o **No**, simplemente **restablezca la configuración**, **sobrescriba la configuración actual** y, a continuación, seleccione **Siguiente**.
- 4. En la página **Elija una colección de configuraciones predeterminada**, elija una colección y, a continuación, seleccione **Finalizar**.
- 5. En la página **Restablecimiento completado**, haga clic en **Cerrar**.

23

Búsqueda e instalación de extensiones para Visual Studio



Las extensiones son paquetes de código que se ejecutan dentro de Visual Studio y que ofrecen características nuevas o mejoradas.

Las extensiones pueden ser controles, ejemplos, plantillas, herramientas u otros componentes que agregan función a Visual Studio, como [Live Share](#) o [GitHub Copilot](#).

En el IDE de Visual Studio, el cuadro de diálogo **Administrar extensiones** es la herramienta que se usa para buscar, instalar y administrar extensiones de Visual Studio.

24

Comandos adicionales recomendados

Para favorecer los trabajos de depuración y pruebas se recomiendan dos comandos.

- **curl**, instalado con la SDK
- **HttpRepl**, basado en *swagger* y que podemos instalar mediante el comando:

```
dotnet tool install -g microsoft.dotnet-httprepl
```

<https://github.com/dotnet/HttpRepl>

Explorar e invocar servicios HTTP mediante el comando HttpRepl

El equipo de ASP.NET ha creado una herramienta de línea de comandos llamada **HttpRepl**.

Le permite explorar e invocar servicios HTTP de forma similar a trabajar con archivos y carpetas.

Le das un punto de partida (una URL base) y luego puedes ejecutar comandos como **"dir"** y **"cd"** para navegar por la API:

```
C:\> dotnet httprepl http://localhost:65369/
(Disconnected)~ set base http://localhost:65369
Using swagger metadata from http://localhost:65369/swagger/v1/swagger.json

http://localhost:65369/~ dir
.
Fruits [get|post]
People [get|post]

http://localhost:65369/~ cd People
/People [get|post]

http://localhost:65369/People~ dir
.
[get|post]
..
[id] [get]

http://localhost:65369/People~ get
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Wed, 24 Jul 2019 20:33:07 GMT
Server: Microsoft-IIS/10.0
Transfer-Encoding: chunked
X-Powered-By: ASP.NET

[
  {
    "id": 1,
    "name": "Scott Hunter"
  },
  {
    "id": 0,
```

25



Desarrollo de Web APIs

26

Creación de un proyecto Web API con ASP.NET Core

Tenemos como objetivo desarrollar un servicio *RESTful* multiplataforma.

Comenzamos por la creación de un proyecto Web API de ASP.NET Core programando en C#.

Desde *VSCode* abrimos el terminal integrado y ejecutamos los siguientes comandos:

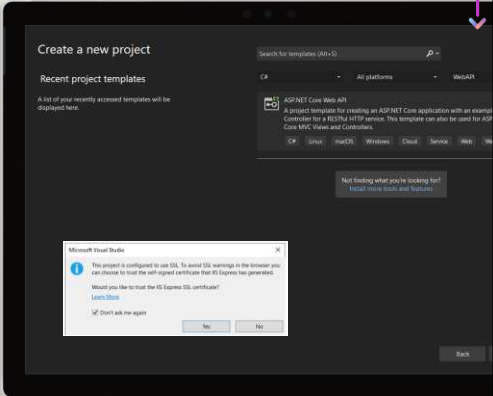
```
CLI de .NET

dotnet new webapi -o TodoApi --language "C#" --framework "net8.0"
cd TodoApi
dotnet add package <PackageName>
code -r ../TodoApi

dotnet dev-certs https --trust
```

Creación de un Proyecto Web API desde Visual Studio

- En el menú **Archivo**, seleccione **Nuevo>Proyecto**.
- Escriba **API web** en el cuadro de búsqueda.
- **Seleccione** la plantilla **API web de ASP.NET Core** y seleccione **Siguiente**.
- Cumplimente Información adicional como la versión del Framework, y Seleccione **Crear**.



27

Estructura de un proyecto Web API

.NET Core 3.1

```
.
├── Controllers
│   └── WeatherForecastController.cs
├── Microsoft.AspNetCore.Net31.WebAPI.TodoAPI.csproj
├── Program.cs
├── Properties
│   └── launchSettings.json
├── Startup.cs
├── WeatherForecast.cs
├── appsettings.Development.json
├── appsettings.json
└── bin
```

.NET Core 7.0

```
.
├── Controllers
│   └── WeatherForecastController.cs
├── Microsoft.AspNetCore.Net7.WebAPI.TodoAPI.csproj
├── Microsoft.AspNetCore.Net7.WebAPI.TodoAPI.csproj.user
├── Program.cs
├── Properties
│   └── launchSettings.json
├── WeatherForecast.cs
├── appsettings.Development.json
├── appsettings.json
└── bin
```

28

Archivo de Proyecto

Microsoft Build Engine es una plataforma para compilar aplicaciones.

También conocido como *MSBuild*, proporciona un esquema XML para un archivo del proyecto que controla cómo la plataforma de compilación procesa y compila el software.

XML

```
Rsôkêçtj Şđl Nıçsôşôğğj NÉT Şđl Wêç

RsôřêstjŸGsôuıř
TăsôğğjGsăñêxôsl nêŸ' TăsôğğjGsăñêxôsl
Nulltăçlê êñăçlê Nulltăçlê
İñřlîçitŸŸşîngş êñăçlê İñřlîçitŸŸşîngş
RsôřêstjŸGsôuıř

İŸêñGsôuıř
RăçlăğêRêğêsênğê İñçludê Nıçsôşôğğj AşřNêŸCôsê ÖřênAřl Lêsşîñ ' . . .
RăçlăğêRêğêsênğê İñçludê Şxăşhçuçlê AşřNêŸCôsê Lêsşîñ _ .
İŸêñGsôuıř

Rsôkêçtj
```

Elemento <PackageReference>

El archivo del proyecto contiene etiquetas <PackageReference> con la estructura siguiente:

```
<PackageReference Include="PACKAGE_ID" Version="PACKAGE_VERSION" />
```

- El atributo **Include** especifica el identificador del paquete que se va a agregar al proyecto.
- El atributo **Version** especifica la versión que se va a obtener.

Las versiones se especifican en función del [Versionamiento Semántico 2.0.0](#) que cumplen las publicaciones paquetes en el repositorio [NuGet](#).

Para agregar una dependencia, agregue un elemento <PackageReference> dentro de un elemento <ItemGroup>, o puede utilizar los comandos:

```
dotnet add package Microsoft.EntityFrameworkCore
dotnet remove package Microsoft.EntityFrameworkCore
```

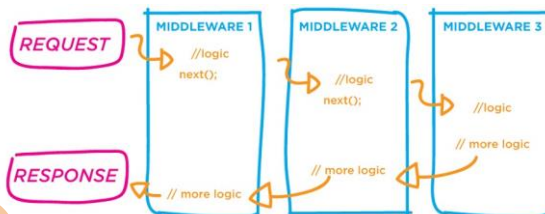


Clase Program

Las nuevas versiones de ASP.NET Core fusionan en **Program** las clases Startup y Program.

Las aplicaciones de ASP.NET Core creadas con las plantillas web contienen el código de inicio de la aplicación en el archivo **Program.cs**. El archivo **Program.cs** es donde:

- Se configuran los servicios requeridos por la aplicación (Inyección de dependencias).
- Se define los componentes de middleware, que representan un conjunto de canalizaciones por donde fluyen las solicitudes de la aplicación.



32

Inserción de Servicios

ASP.NET Core incluye inyección de dependencias que hace que estén disponibles los servicios configurados.

Cuando se crea una instancia de **WebApplicationBuilder**, se agregan muchos servicios proporcionados por el Framework.

```

C#
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddRazorPages();
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<RazorPagesMovieContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("TodoAPIContext")));

var app = builder.Build();
  
```

Los servicios se suelen resolver desde la inserción de dependencias mediante los constructores, proporcionando una instancia del este servicio en tiempo de ejecución.



33

Duración de los Servicios

Adición de Singleton

Los servicios de duración de singleton se crean de alguna de las formas siguientes:

- La primera vez que se solicitan.
- Mediante el desarrollador, al proporcionar una instancia de implementación directamente al contenedor.

Cada solicitud siguiente de la implementación del servicio desde el contenedor de inserción de dependencias utiliza la misma instancia.

Adición de Scoped

Una duración con ámbito indica que los servicios se crean una vez por solicitud de cliente (conexión).

Cuando se usa Entity Framework Core, el método de extensión **AddDbContext** registra tipos de DbContext con una duración de ámbito de forma predeterminada.

Adición de Transient

Los servicios de duración transitoria se crean cada vez que el contenedor del servicio los solicita.

Esta duración funciona mejor para servicios sin estado ligeros.

En las aplicaciones que procesan solicitudes, los servicios transitorios se eliminan al final de la solicitud.

Canalizaciones

La canalización de control de solicitudes se compone de una serie de componentes de software intermedio.

Cada componente lleva a cabo las operaciones en un contexto **HttpContext** e invoca el middleware siguiente de la canalización, o bien finaliza la solicitud.

Normalmente, se agrega un componente de middleware a la canalización mediante la invocación de un método de extensión **Use{Feature}**.

```
C#  
  
// Configure the HTTP request pipeline.  
if (!app.Environment.IsDevelopment()) {  
    app.UseExceptionHandler("/Error");  
    app.UseHsts();  
}  
app.UseHttpsRedirection();  
app.UseStaticFiles();  
app.UseAuthorization();  
app.UseCustomElement();
```

Orden del Middleware

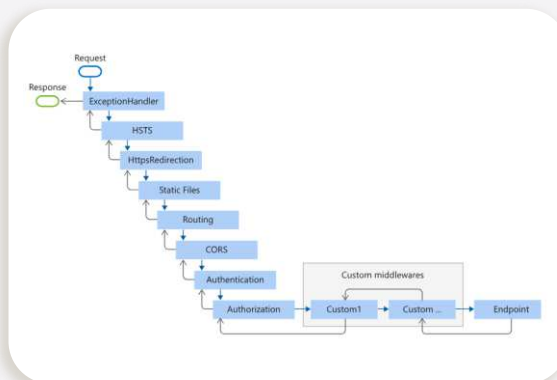
En el diagrama siguiente se muestra el procesamiento de solicitudes para las aplicaciones de ASP.NET Core.

Se ordenan los middleware existentes y se agregan los middleware personalizados.

Tenemos control total para reordenar los middleware existentes o insertar según sea necesario.

El orden en el que se agregan los componentes de software intermedio en el método **Program.cs** define el orden en el que se invocarán en las solicitudes y el orden inverso de la respuesta.

Por motivos de seguridad, rendimiento y funcionalidad, el orden es crítico.



36

.MapControllers() el enrutamiento Web API 2

El enrutamiento es cómo la API web coincide con un URI con una acción.

Web API 2 admite un nuevo tipo de enrutamiento, denominado *enrutamiento de atributos*. Como el nombre implica, el enrutamiento de atributos usa atributos para definir rutas. El enrutamiento de atributos proporciona más control sobre los URI de la API web.

El estilo anterior de enrutamiento, denominado *enrutamiento basado en definiciones* se mantiene por compatibilidad, pero se considera obsoleto.

Puede combinar ambas técnicas en el mismo proyecto.

Una de las ventajas del enrutamiento basado en convención es que las plantillas se definen en un solo lugar y las reglas de enrutamiento se aplican de forma coherente en todos los controladores.

```
[Route("customers/{customerId}/orders")]
public IEnumerable<Order> GetOrdersByCustomer(int customerId) { ... }
```

37

Comparativa de la sintaxis del Enrutamiento

Anterior a .NET 6

```
app.UseRouting();
app.UseEndpoints(endpoints => {
    endpoints.MapGet("/", () => "Hello World");
});

app.UseEndpoints(e => { e.MapControllers(); });

app.UseEndpoints(endpoints => {
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller}/{action}/{id?}");
});

app.MapControllerRoute(name: "default",
    pattern: "{controller}/{action}/{id?}");
```

Desde .NET 6

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

38

API mínimo con ASP.NET Core

Las *API mínimas* están diseñadas para crear API HTTP con dependencias mínimas.

No es ni más ni menos que una API con una cantidad de código muy reducida, sin **Controllers**, e ideales para microservicios con dependencias mínimas.

C#

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

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

app.Run();
```

39

Controladores

ASP.NET Core admite la creación de API web mediante controladores o mediante API mínimas.

Los controladores de una API web son clases que se derivan de **ControllerBase**. Los controladores se activan y eliminan por solicitud.

```
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
```

Normalmente, los controladores de API web deben derivarse de **ControllerBase**, en lugar de **Controller**.

Controller se deriva de **ControllerBase** y agrega compatibilidad con vistas, por lo que sirve para gestionar páginas web, no solicitudes de API web. Si el mismo controlador debe admitir vistas y API web, debe derivarse de **Controller**.

La clase **ControllerBase** ofrece muchas propiedades y métodos que son útiles para gestionar solicitudes HTTP.

Crear un Controlador

dotnet aspnet-codegenerator: ejecuta el motor de scaffolding de ASP.NET Core.

dotnet aspnet-codegenerator solo es necesario para aplicar *scaffolding* desde la línea de comandos, no es necesario para usar la técnica de *scaffolding* con Visual Studio.

```
dotnet tool install -g dotnet-aspnet-codegenerator

dotnet add package Microsoft.VisualStudio.Web.CodeGeneration

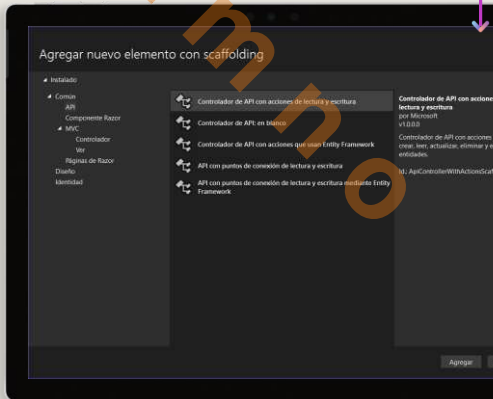
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design

dotnet aspnet-codegenerator controller
    -name Demo -async -api -outDir Controllers
```

Crear un controlador desde Visual Studio

La manera más fácil de crear un nuevo controlador es hacer **clic con el botón derecho en la carpeta Controladores** de la ventana de visual Studio Explorador de soluciones y **seleccionar la opción de menú *Agregar, controlador***

Al seleccionar esta opción de menú, **se abre el cuadro de diálogo *Agregar controlador***



Atributos para controladores

El espacio de nombres **Microsoft.AspNetCore.Mvc** proporciona atributos que se pueden usar para configurar el comportamiento de los métodos en los controladores API web:

- **[Route]**, especifica el patrón de dirección URL de un controlador o una acción.
- **[Bind]**, especifica el prefijo y las propiedades que se incluirán en el enlace de modelo.
- **[HttpGet]**, identifica una acción que admite el verbo de acción GET HTTP.
- **[Consumes]**, especifica los tipos de datos que acepta una acción.
- **[Produces]**, especifica los tipos de datos que devuelve una acción.

```
[HttpPost]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public ActionResult<Pet> Create(Pet pet)
{ }
```

42

El atributo ApiController

El atributo **[ApiController]** se puede aplicar a una clase de controlador para permitir los siguientes comportamientos específicos de la API:

- Requisito de enrutamiento mediante atributos
- Respuestas HTTP 400 automáticas
- Validan automáticamente el estado del modelo y devuelven una respuesta 400
- Deducción de los parámetros, suprimiendo la necesidad de identificarlos con atributos
- Detalles de problemas de los códigos de estado de error

43

Atributo de Enrutamiento

La cadena "**customers/{customerId}/orders**" es la plantilla de URI de la ruta. La API intenta hacer coincidir el URI de solicitud con la plantilla.

```
public class OrdersController : ControllerBase
{
    [Route("customers/{customerId}/orders")]
    [HttpGet]
    public IEnumerable<Order> FindOrdersByCustomer(int customerId) { ... }
}
```

En el ejemplo, "**customers**" y "**orders**" son segmentos literales y "**{customerId}**" es un parámetro de variable. Los siguientes URI coincidirían con esta plantilla:

- http://localhost/customers/1/orders
- http://localhost/customers/bob/orders
- http://localhost/customers/1234-5678/orders

44

Atributo de Enrutamiento – Nombre reservados

Las siguientes palabras clave son nombres de parámetros de ruta reservados al usar Controladores o Razor páginas:

- action
- area
- controller
- handler
- page

```
public class OrdersController : ControllerBase
{
    [Route("[controller]/{customerId}/orders")]
    [HttpGet]
    public IEnumerable<Order> FindOrdersByCustomer(int customerId)
    { ... }
}
```



45

Atributo de Enrutamiento - Métodos

La API web también selecciona acciones basadas en el método HTTP de la solicitud (GET, POST, etc.).

De forma predeterminada, la API web busca una coincidencia sin distinción entre mayúsculas y minúsculas con el inicio del nombre del método de controlador.

Por ejemplo, un método de controlador denominado **PutCustomers** coincide con una solicitud **HTTP PUT**.

Puede invalidar esta convención decorando el método con cualquiera de los atributos siguientes:

- [HttpDelete]
 - [HttpGet]
 - [HttpHead]
 - [HttpOptions]
 - [HttpPatch]
 - [HttpPost]
 - [HttpPut]
- ```
[Route("api/books")]
[HttpPost]
public HttpResponseMessage CreateBook(Book book)
{ ... }
```



46

## Atributo de Enrutamiento – Prefijos de Ruta

1. A menudo, las rutas de un controlador comienzan con el mismo prefijo.
2. Puede establecer un prefijo común para un controlador completo mediante el atributo [RoutePrefix] o [Route] y puede incluir parámetros.
3. Usa una tilde (~) en el atributo de método para invalidar el prefijo de ruta.

```
1 public class BooksController : ControllerBase
{
 [Route("api/books")]
 public IEnumerable<Book> GetBooks()
 { ... }

 [Route("api/books/{id:int}")]
 public Book GetBook(int id) { ... }
}

2 [RoutePrefix("api/books")]
public class BooksController : ControllerBase
{
 // GET api/books
 [Route("")] public IEnumerable<Book> Get()
 [HttpGet]
 { ... }

 // GET api/books/5
 [Route("{id:int}")]
 public Book Get(int id)
 { ... }
}

3 [Route("api/books")]
public class BooksController : ControllerBase
{
 // GET /api/authors/1/books
 [Route("~/api/authors/{authorId:int}/books")]
 public IEnumerable<Book> GetByAuthor(int authorId)
 { ... }

 // ...
}
```



47

## Atributo de Enrutamiento – Restricciones de Ruta

1. Las restricciones de ruta permiten restringir cómo coinciden los parámetros de la plantilla de ruta.

La sintaxis general es:

`{parameter:constraint}`

2. Observe que algunas de las restricciones, como "min", toman argumentos entre paréntesis.

```
1 [Route("users/{id:int}")]
 public User GetUserById(int id)
 { ... }

[Route("users/{name}")]
public User GetUserByName(string name)
{ ... }
```

```
2 [Route("users/{id:int:min(1)}")]
 public User GetUserById(int id)
 { ... }
```



48

## Atributo de Enrutamiento – Parámetros opcionales y valores predeterminados

1. Puede hacer que un parámetro URI sea opcional agregando un signo de interrogación al parámetro de ruta.
2. Como alternativa, puede especificar un valor predeterminado dentro de la plantilla de ruta.

```
1 public class BooksController : ControllerBase
 {
 [Route("api/books/locale/{lcid:int}")]
 public IEnumerable<Book> GetBooksByLocale(int? lcid)
 { ... }
 }

2 public class BooksController : ControllerBase
 {
 [Route("api/books/locale/{lcid:int}")]
 public IEnumerable<Book> GetBooksByLocale(int lcid = 1033)
 { ... }
 }
```



49

## Enlace de modelos en ASP.NET Core

### Los controladores trabajan con datos que provienen de solicitudes HTTP.

Por ejemplo, los datos de ruta pueden proporcionar una clave de registro y los campos de formulario publicados pueden proporcionar valores para las propiedades del modelo.

La escritura de código para recuperar cada uno de estos valores y convertirlos de cadenas a tipos de .NET sería tediosa y propensa a errores. **El enlace de modelos automatiza este proceso.**

El sistema de enlace de modelos:

- Recupera datos de diversos orígenes, como datos de ruta, campos de formulario y cadenas de consulta
- Proporciona los datos a los controladores en parámetros de método y propiedades públicas
- Convierte datos de cadena en tipos de .NET
- Actualiza las propiedades de tipos complejos



50

## Orígenes

De forma predeterminada, el enlace de modelos obtiene datos en forma de pares clave-valor de los siguientes orígenes de una solicitud HTTP:

- Campos de formulario
- El cuerpo de la solicitud (para controladores que tienen el atributo [ApiController]).
- Datos de ruta
- Parámetros de cadena de consulta
- Archivos cargados
- Motor de Inyección de Dependencia

Para cada parámetro o propiedad de destino, se examinan los orígenes en el orden indicado en la lista anterior.

51

## Atributos de Origen

Usamos uno de los atributos siguientes para especificar el origen:

- `[FromQuery]` obtiene valores de la cadena de consulta.
- `[FromRoute]` obtiene valores de los datos de ruta.
- `[FromForm]` obtiene valores de los campos de formulario publicados.
- `[FromBody]` obtiene valores del cuerpo de la solicitud.
- `[FromHeader]` obtiene valores de encabezados HTTP.
- `[FromService]` obtiene valores del motor de Inyección de Dependencia.

```
[HttpPost]
public ActionResult OnPost([FromBody] Instructor instructor)
{...}
```

52

## Filtros en ASP.NET Core

Los filtros en ASP.NET Core permiten que se ejecute el código antes o después de determinadas fases de la canalización del procesamiento de la solicitud.

Los filtros integrados se encargan de tareas como las siguientes:

- Autorización, que impide el acceso a recursos para los que un usuario no está autorizado.
- Almacenamiento en caché de la respuesta, que permite cortocircuitar la canalización de la solicitud para devolver una respuesta almacenada en caché.

Se pueden crear filtros personalizados que se encarguen de cuestiones transversales evitando la duplicación de código. Así, por ejemplo, un filtro de excepción de control de errores puede consolidar el control de errores.

53



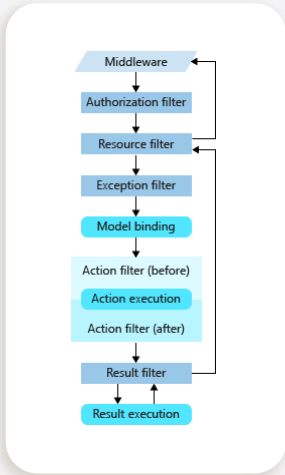
## Funcionamiento de los filtros

Los filtros se ejecutan dentro de la canalización de invocación de acciones de ASP.NET Core.

La canalización de filtro se ejecuta después de que ASP.NET Core seleccione la acción que se va a ejecutar.

### Filtros de autorización:

- Se ejecutan primero.
- Determinan si el usuario está autorizado para la solicitud.
- Cortocircuitan la canalización si una solicitud no está autorizada.



54

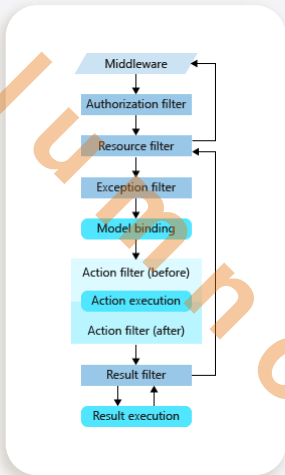
## Funcionamiento de los filtros II

### Filtros de recursos:

- Se ejecutan después de la autorización.
- **OnResourceExecuting** ejecuta código antes que el resto de la canalización del filtro.
- **OnResourceExecuted** ejecuta el código una vez que el resto de la canalización se haya completado.

### Filtros de acciones:

- Se ejecutan inmediatamente antes y después de llamar a un método de acción del controlador.
- Pueden cambiar los argumentos pasados a una acción.
- Pueden cambiar el resultado devuelto de la acción.



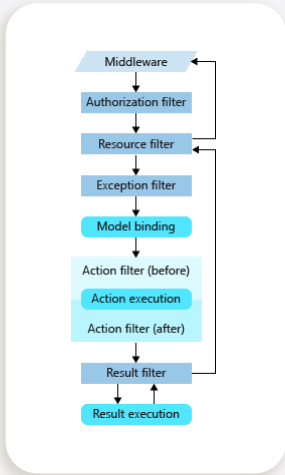
55

### Funcionamiento de los filtros III

Los **Filtros de Excepciones** aplican directivas globales a las excepciones no controladas que se producen antes de que se escriba el cuerpo de respuesta.

**Filtros de resultados:**

- Se ejecutan inmediatamente antes y después de la ejecución de resultados de acción.
- Solo se ejecutan cuando el método de acción se ejecuta correctamente.
- Son útiles para la lógica que debe regir la ejecución de la vista o el formateador.



56

### Orden de ejecución predeterminado

Cuando hay varios filtros en una determinada fase de la canalización, el ámbito determina el orden predeterminado en el que esos filtros se van a ejecutar.

Los filtros globales abarcan a los filtros de clase, que a su vez engloban a los filtros de método.

| Secuencia | Ámbito del filtro | Método de filtro  |
|-----------|-------------------|-------------------|
| 1         | Global            | OnActionExecuting |
| 2         | Controlador       | OnActionExecuting |
| 3         | Acción            | OnActionExecuting |
| 4         | Acción            | OnActionExecuted  |
| 5         | Controlador       | OnActionExecuted  |
| 6         | Global            | OnActionExecuted  |

57

## Implementación

Los filtros admiten implementaciones tanto sincrónicas como asincrónicas a través de diferentes definiciones de interfaz:

- Implementan la interfaz **IActionFilter** o **IAsyncActionFilter**.
- Su ejecución rodea la ejecución de los métodos de acción.

```
C#
public class SampleActionFilter : IActionFilter
{
 public void OnActionExecuting(ActionExecutingContext context)
 {
 // Do something before the action executes.
 }

 public void OnActionExecuted(ActionExecutedContext context)
 {
 // Do something after the action executes.
 }
}
```

Implementación  
síncrona



58

## Implementación II

**ActionExecutingContext** ofrece las siguientes propiedades:

- **ActionArguments**: permite leer las entradas de un método de acción.
- **Controller**: permite manipular la instancia del controlador.
- **Result**: si se establece **Result**, se cortocircuita la ejecución del método de acción y de los filtros de acciones posteriores.

Inicio de una excepción en un método de acción:

- Impide la ejecución de los filtros subsiguientes.
- A diferencia del establecimiento de **Result**, se trata como un error en lugar de como un resultado correcto.

59

## Implementación III

**ActionExecutedContext** proporciona **Controller** y **Result**, además de las siguientes propiedades:

- **Canceled**: es true si otro filtro ha cortocircuitado la ejecución de la acción.
- **Exception**: es un valor distinto de NULL si la acción o un filtro de acción de ejecución anterior han producido una excepción. Si se establece esta propiedad en un valor NULL:
  - Controla la excepción eficazmente.
  - **Result** se ejecuta como si se devolviera desde el método de acción.

En un **IAsyncActionFilter**, una llamada a **ActionExecutionDelegate**:

- Ejecuta cualquier filtro de acciones posterior y el método de acción.
- Devuelve **ActionExecutedContext**.

60

## Implementación IV

Los filtros admiten implementaciones tanto sincrónicas como asincrónicas a través de diferentes definiciones de interfaz.

```
C#

public class SampleAsyncActionFilter : IAsyncActionFilter
{
 public async Task OnActionExecutionAsync(
 ActionExecutingContext context, ActionExecutionDelegate next)
 {
 // Do something before the action executes.
 await next();
 // Do something after the action executes.
 }
}
```

Implementación  
asíncrona



61

## Implementación V

Se puede usar el filtro de acción **OnActionExecuting** para:

- Validar el estado del modelo.
- Devolver un error si el estado no es válido.

```
C#

public class ValidateModelAttribute : ActionFilterAttribute
{
 public override void OnActionExecuting(ActionExecutingContext context)
 {
 if (!context.ModelState.IsValid)
 {
 context.Result = new BadRequestObjectResult(context.ModelState);
 }
 }
}
```



62



## Desarrollo de Aplicaciones MVC



63

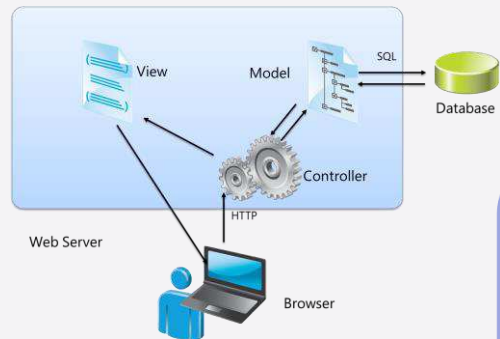
## Modelos (models)

Los **Modelos** representan la estructura de los datos que se manejarán en la aplicación.

En ASP.NET Core, un modelo generalmente corresponde a una clase que define los datos que serán almacenados en la base de datos y cómo se manipulan.

Funciones principales:

- Contienen propiedades que representan los datos que se van a mostrar o procesar.
- Validan la información y aplican reglas de negocio.
- Son responsables de la comunicación con la base de datos a través de Entity Framework u otros mecanismos de persistencia de datos.



64

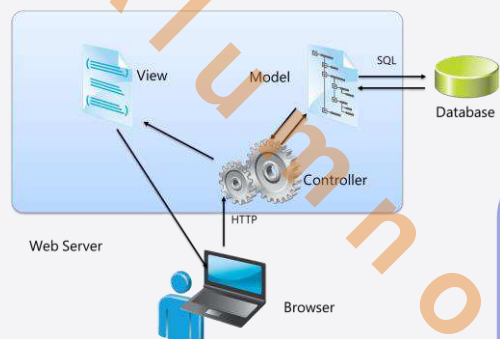
## Vistas (views)

Las **Vistas** son las encargadas de presentar los datos al usuario, generalmente en forma de HTML.

En ASP.NET Core, las vistas se implementan como archivos .cshtml, que combinan HTML con sintaxis Razor para integrar dinámicamente los datos provenientes del modelo.

Funciones principales:

- Presentan la interfaz de usuario.
- Usan datos del modelo para generar contenido dinámico.
- Pueden contener formularios de entrada para que los usuarios interactúen con la aplicación.



65

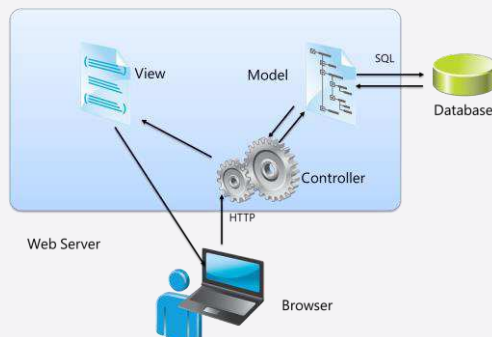
## Controladores (controllers)

Los **Controladores** actúan como intermediarios entre los modelos y las vistas.

Reciben las solicitudes del usuario, procesan la lógica de negocio y luego seleccionan la vista apropiada para mostrar los resultados.

Funciones principales:

- Manejan las peticiones HTTP (GET, POST, etc.).
- Llamam a los modelos para recuperar o actualizar los datos.
- Pasan los datos procesados a las vistas para que sean presentados.



66

## Configuración de aplicaciones MVC

En ASP.NET Core, los servicios y el middleware se configuran en la clase **Program** para configurar la aplicación web.

### Servicios en la clase **Program**.

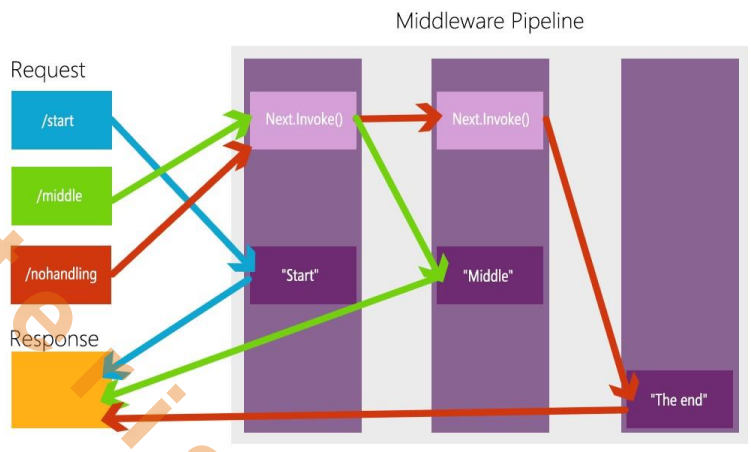
- los servicios son componentes que se registran en el contenedor de dependencias (DI) y se pueden inyectar en diferentes partes de la aplicación
- se configuran en la clase **Program**

### Middleware en la clase **Program**.

- el middleware es un componente que maneja las solicitudes HTTP
- se ejecuta en una secuencia, modificando o gestionando la solicitud/respuesta
- se configura en la clase **Program**, y su ejecución sigue el orden en que se registra

67

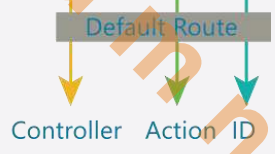
# Configuración de aplicaciones MVC



68

# Rutas MVC

El enrutamiento determina qué controlador y acción se deben llamar para manejar una solicitud.



## MapControllerRoute

Define la ruta de los controladores.

- name es un nombre opcional para la ruta (en este caso, "default")
- pattern especifica el patrón de la URL que debe coincidir. En este caso:
  - {controller=Home}: Si no se proporciona un nombre de controlador, se usa "Home" valor predeterminado.
  - {action=Index}: Si no se proporciona un nombre de acción, se usa "Index" valor predeterminado.
  - {id?}: El parámetro id es opcional.

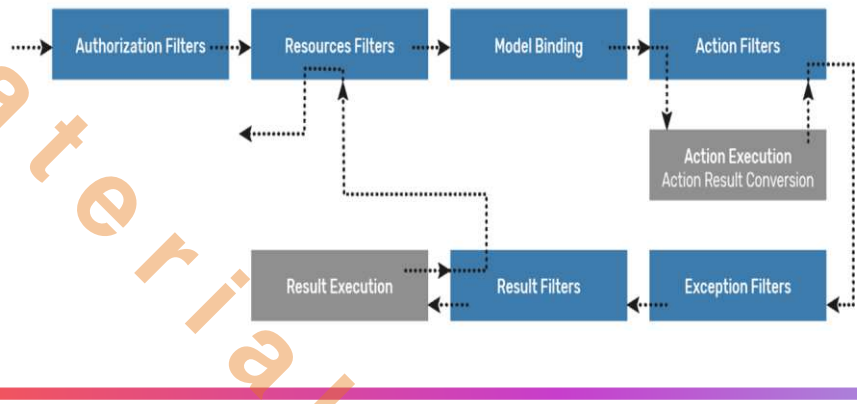
```
app.MapControllerRoute(name: "default",
 pattern: "{controller=Home}/{action=Index}/{id?}");
```

69



## Filtros en MV

Los filtros en ASP.NET Core permiten que se ejecute el código antes o después de determinadas fases de la canalización del procesamiento de la solicitud.



70

## Implementación

Se puede usar el filtro de acción **OnActionExecuting** para:

- Validar el estado del modelo.
- Devolver un error si el estado no es válido.

```
C#
public class ValidateModelAttribute : ActionFilterAttribute
{
 public override void OnActionExecuting(ActionExecutingContext context)
 {
 if (!context.ModelState.IsValid)
 {
 context.Result = new BadRequestObjectResult(context.ModelState);
 }
 }
}
```



71

## Sintaxis Razor

Las vistas en ASP.NET Core MVC son componentes clave para la presentación de la interfaz de usuario.

Utilizan la sintaxis Razor, que es un motor de plantillas ligero y eficiente para generar HTML dinámicamente, combinando el código C# con HTML.

En la sintaxis de Razor, el símbolo @ tiene varios usos. Puede:

- Usar @ para identificar código C# del lado del servidor
- Usar @@ para representar un símbolo @ en una página HTML
- Usar @: para declarar explícitamente una línea de texto como contenido y no como código
- Usar <text> para declarar explícitamente varias líneas de texto como contenido y no como código

Las vistas permiten que los datos del servidor se muestren en el navegador del cliente de manera dinámica.

72

## Vistas

### Pasar datos desde el Controlador

Los controladores pasan datos a las vistas a través de **ViewData**, **ViewBag**, o **Model**.

```
public IActionResult Index()
{
 var product = new Product { Name = "Laptop", Price = 1000 };
 return View(product); // Pasa el modelo a la vista
}
```

En la vista, el modelo **Product** se puede acceder utilizando **@Model**.

73

# Vistas

## ViewData y ViewBag

**ViewData** y **ViewBag** son maneras dinámicas de pasar datos desde el controlador a la vista, aunque se usan de manera ligeramente diferente.

**ViewData** es un diccionario **ViewData["key"]**, mientras que **ViewBag** utiliza una sintaxis **ViewBag.Property**.

## Layouts

Los layouts permiten definir una estructura común para varias vistas.

Es útil para agregar elementos comunes (como encabezados, pie de página, menús) en varias páginas sin repetir el código en cada vista. Un layout se define en un archivo **\_Layout.cshtml**, que se utiliza en las vistas principales.

74

# Vistas

## Vistas Parciales

Las vistas parciales permiten reutilizar fragmentos de código HTML que pueden ser insertados en otras vistas.

Son útiles para evitar la duplicación de contenido común en varias vistas (como formularios, tablas, etc.).

```
<partial name="_MyPartialView" />
@await Html.PartialAsync("_MyPartialView")
```



75



## EF Core y LINQ

76

### ¿Qué es ADO.NET?

ADO.NET es un conjunto de clases que exponen servicios de acceso a datos para programadores de .NET.

- proporciona acceso a orígenes de datos como SQL Server, XML, y otros expuestos mediante OLEDB y ODBC
- separa el acceso a datos de la manipulación de datos
- incluye proveedores de datos .NET para conectarse a una base de datos
- Incluye objetos para ejecutar comandos y recuperar resultados

Los resultados se procesan directamente o se colocan en un objeto **DataSet** con el fin de exponerlos al usuario para un propósito específico, combinados con datos de varios orígenes, o de pasarlos entre niveles.

El objeto **DataSet** también puede utilizarse independientemente de un proveedor de datos .NET para administrar datos que son locales de la aplicación o que proceden de un origen XML.

77

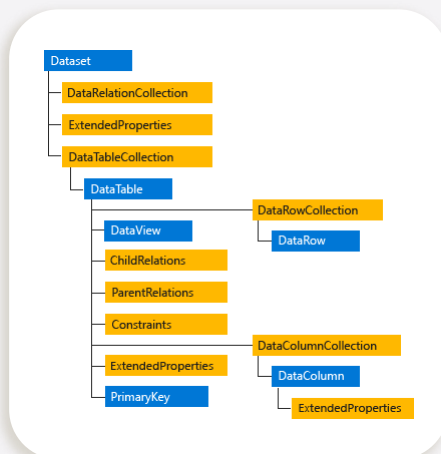
## Componentes de ADO.NET

Los dos componentes principales de ADO.NET para el acceso a los datos y su manipulación son los proveedores de datos y el objeto DataSet.

Los proveedores de datos .NET Framework son componentes diseñados explícitamente para la manipulación de datos y el acceso rápido a datos de solo lectura y solo avance.

**DataSet** de ADO.NET está expresamente diseñado para el acceso a datos independientemente del origen de datos.

Se puede utilizar con múltiples y distintos orígenes de datos, contiene una colección de uno o más objetos **DataTable** formados por filas y columnas de datos, así como información sobre claves principales, claves externas, restricciones y de relación relacionada con los datos incluidos en los objetos **DataTable**.



78

## Elegir un DataReader o un DataSet

Una aplicación puede utilizar un elemento **DataReader** o, dependiendo de la funcionalidad que requiere la aplicación.

Use un **DataSet** para hacer lo siguiente:

- Almacene datos en la memoria caché de la aplicación para poder manipularlos. Si solamente necesita leer los resultados de una consulta, el **DataReader** es la mejor elección.
- Interactuar con datos dinámicamente, por ejemplo, para enlazar con un control de Windows Forms o para combinar y relacionar datos procedentes de varios orígenes.
- Realizar procesamientos exhaustivos de datos sin necesidad de tener una conexión abierta con el origen de datos, lo que libera la conexión para que la utilicen otros clientes.

Si no necesita la funcionalidad proporcionada por el **DataSet**, puede mejorar el rendimiento de su aplicación si utiliza el **DataReader** para devolver sus datos de solo avance y de solo lectura.

79

## Oracle y ADO.NET

El proveedor de datos de .NET para Oracle proporciona acceso a bases de datos Oracle mediante la Interfaz de llamada de Oracle (OCI) que se suministra con el software Oracle Client.

La funcionalidad del proveedor de datos está diseñada para ser similar a la de los proveedores de datos de .NET para SQL Server, OLE DB y ODBC.

**Oracle Data Provider for .NET (ODP.NET) incluye acceso a datos ADO.NET optimizado a la base de datos de Oracle. ODP.NET permite a los desarrolladores aprovechar la funcionalidad avanzada de Oracle Database.**

Hay tres tipos de controladores:

- ODP.NET Core, diseñado para aplicaciones NET (Core) multiplataforma
- ODP.NET Managed Driver, proveedor .NET código administrado al 100%.
- ODP.NET Unmanaged Driver, proveedor tradicional de Oracle ADO.NET que utiliza Oracle Database Client

80

## Entity Framework Core

Entity Framework Core

Entity Framework (EF) Core es una versión ligera, extensible, de código abierto y multiplataforma de la popular tecnología de acceso a datos Entity Framework.

EF Core puede actuar como asignador relacional de objetos, que se encarga de lo siguiente:

- Permite a los desarrolladores de .NET trabajar con una base de datos usando objetos .NET.
- Permite prescindir de la mayor parte del código de acceso a datos que normalmente es necesario escribir.

**El acceso a datos se realiza mediante un modelo.**

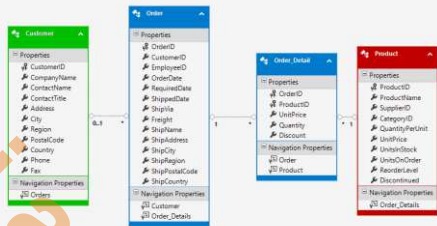
Un modelo se compone de clases de entidad y un objeto de contexto que representa una sesión con la base de datos. Este objeto de contexto permite consultar y manipular los datos.

81

## Creación del Modelo de EF

EF admite los siguientes métodos de desarrollo de modelos:

- Generar un modelo a partir de una base de datos existente.
- Codificar un modelo manualmente para que coincida con la base de datos.
- Una vez creado un modelo, usar *Migraciones de EF* para crear una base de datos a partir del modelo. *Migraciones* permite que la base de datos evolucione a medida que el modelo va cambiando.



82

## Scaffolding (utilización de técnicas de ingeniería inversa)

Las técnicas de ingeniería inversa constituyen el proceso de **scaffolding** de clases de tipo de entidad y una clase **DbContext** basada en un esquema de base de datos.

Se puede realizar mediante el comando **Scaffold-DbContext** de las herramientas de consola del administrador de paquetes o mediante el comando **dotnet ef dbcontext scaffold** de las herramientas de la interfaz de línea de comandos (CLI) de .NET.

Ambos comandos tienen dos argumentos necesarios:

- la cadena de conexión a la base de datos
- el proveedor de bases de datos de EF Core que se va a usar

```
C:\Users\carlos>dotnet ef dbcontext scaffold "Server=(localdb)\Instance1;Database=Adventureworks2012" Microsoft.EntityFrameworkCore.SqlServer

Usage: dotnet ef dbcontext scaffold [arguments] [options]

Arguments:
 The connection string to the database.
 The provider to use. (e.g. Microsoft.EntityFrameworkCore.SqlServer)

Options:
 --context <context> The name of the DbContext.
 --output-dir <output-dir> The directory to put files in. Paths are relative to the project directory.
 --output <output> The output file name.
 --table <table> <table>... The tables to generate entity types for.
 --use-unicode <use-unicode> Use unicode and column names directly from the database.
 --no-build <no-build> Don't build the project, only use this when the build is up-to-date.
 --verbose <verbose> Show verbose output.
 --prefix-output <prefix-output> Prefix output with level.
 --help <help> Show this help message.
```

83

## Ejecutando Scaffolding

El primer argumento para el comando es una cadena de conexión a la base de datos. Las herramientas usarán esta cadena de conexión para leer el esquema de base de datos.

El segundo argumento es el nombre del proveedor. El nombre del proveedor suele ser el mismo que el nombre del paquete NuGet del proveedor.

```
Scaffold-DbContext "Server=LOCALHOST;Database=NORTHWIND;Trusted_Connection=True;Encrypt=False;"
 "Microsoft.EntityFrameworkCore.SqlServer" -OutputDir "Models" -ContextDir "Models"
 -Context "ModelNorthwind" -UseDatabaseNames -Force
```

```
dotnet ef dbcontext scaffold "Server=localhost;Database=NORTHWIND;Trusted_Connection=True;"
 Microsoft.EntityFrameworkCore.SqlServer -c Model --use-database-names
```

<https://learn.microsoft.com/es-es/ef/core/managing-schemas/scaffolding/>



84

## Seguimiento de Cambios

Cada instancia de **DbContext** realiza un seguimiento de los cambios realizados en las entidades.

Estas entidades de las que se realiza un seguimiento, a su vez, impulsan los cambios en la base de datos cuando se llama a **SaveChanges**.

Cada entidad está asociada a un elemento **EntityState** determinado:

- **DbContext** ya no realiza el seguimiento de las entidades **Detached**.
- Las entidades **Added** son nuevas y aún no se han insertado en la base de datos. Esto significa que se insertarán cuando se llame a **SaveChanges**.
- Las entidades **Unchanged** no han cambiado desde que se consultaron desde la base de datos. Todas las entidades devueltas de las consultas se encuentran inicialmente en este estado.
- Las entidades **Modified** han cambiado desde que se consultaron desde la base de datos. Esto significa que se actualizarán cuando se llame a **SaveChanges**.
- Existen entidades **Deleted** en la base de datos, pero se marcan para eliminarse cuando se llama a **SaveChanges**.



85



## Tabla de los seguimientos de cambios

En la tabla siguiente se resumen los diferentes estados:

Estado de entidad	Seguido por DbContext	Existe en la base de datos	Propiedades modificadas	Acción en SaveChanges
Detached	No	-	-	-
Added	Sí	No	-	Insertar
Unchanged	Sí	Sí	No	-
Modified	Sí	Sí	Sí	Actualizar
Deleted	Sí	Sí	-	Eliminar

86

## Language-Integrated Query (LINQ)

Language-Integrated Query (LINQ) es el nombre de un conjunto de tecnologías basadas en la integración de capacidades de consulta directamente en el lenguaje C#.

Las expresiones de consulta se escriben con una sintaxis de consulta declarativa.

Con la sintaxis de consulta, puede realizar operaciones de filtrado, ordenación y agrupamiento en orígenes de datos con el mínimo código.

Utilice los mismos patrones de expresión de consulta básica para consultar y transformar:

- datos de bases de datos SQL
- conjuntos de datos de ADO.NET
- secuencias y documentos XML
- colecciones. NET



87

## Operaciones CRUD con Customer



**Leer todos los registros:**

```
using (var context = new NorthwindContext())
{
 var customers = context.Customers.ToList();
 customers.ForEach(c => Console.WriteLine($"{c.CustomerID}: {c.CompanyName}"));
}
```



88

## Operaciones CRUD con Customer



**Leer un solo registro:**

```
using (var context = new NorthwindContext())
{
 var customer = context.Customers.FirstOrDefault(c => c.CustomerID == "ALFKI");
 if (customer != null)
 {
 Console.WriteLine($"{customer.CustomerID}: {customer.CompanyName}");
 }
}
```



89

## Operaciones CRUD con Customer



### Insertar un nuevo registro:

```
using (var context = new NorthwindContext())
{
 var newCustomer = new Customer
 {
 CustomerID = "NEW01",
 CompanyName = "New Company",
 ContactName = "John Doe"
 };
 context.Customers.Add(newCustomer);
 context.SaveChanges();
}
```



90

## Operaciones CRUD con Customer



### Actualizar un registro existente:

```
using (var context = new NorthwindContext())
{
 var customer = context.Customers.FirstOrDefault(c => c.CustomerID == "NEW01");
 if (customer != null)
 {
 customer.CompanyName = "Updated Company";
 context.SaveChanges();
 }
}
```



91

## Operaciones CRUD con Customer



### Eliminar un registro:

```
using (var context = new NorthwindContext())
{
 var customer = context.Customers.FirstOrDefault(c => c.CustomerID == "NEW01");
 if (customer != null)
 {
 context.Customers.Remove(customer);
 context.SaveChanges();
 }
}
```



92

## IEnumerable VS IQueryable

Hay dos conjuntos de operadores de consulta estándar de LINQ:

- uno que actúa sobre objetos de tipo **IEnumerable<T>**
- uno que actúa sobre objetos de tipo **IQueryable<T>**

Los métodos que forman cada conjunto son miembros estáticos de las clases **Enumerable** y **Queryable**, respectivamente.

El objeto **IEnumerable<T>** captura los argumentos que se han pasado al método. Cuando se enumera el objeto, se emplea la lógica del operador de consulta y se devuelven los resultados.

Los métodos que extienden **IQueryable<T>** no implementan ningún comportamiento de realización de consultas. Compilan un árbol de expresión que representa la consulta que se va a realizar y el procesamiento de consultas se controla en el origen de datos.



93



## Pruebas Unitarias



94

### ¿Qué es una Prueba Unitaria?

En programación, una **prueba unitaria** es la forma de comprobar el funcionamiento de una unidad de código automáticamente.

Las pruebas unitarias se caracterizan por:

- Prueban que cada **unidad de código funciona y es eficientemente por separado**.
- Suelen instanciar una clase y llamar a uno de sus métodos.
- Verifican aspectos individuales del código sin depender de servidores de bases de datos, conexiones de red y otra infraestructura.
- Se enfocan en el código que se escribe.



95

## ¿Qué es una Prueba de Integración?

Las **pruebas de integración** se caracterizan por:

- Prueban la **interacción entre dos o más elementos**.
- Los elementos probados pueden ser clases, módulos, paquetes, subsistemas, ...
- Se realizan una vez que se han aprobado las pruebas unitarias.
- Se centra principalmente en probar la comunicación entre los componentes y sus comunicaciones ya sea hardware o software.



96

## Pruebas de Software

### Pruebas Unitarias:

- Menos código, más rápidas en ejecutarse.
- Utilización de *Mocks* para emular elementos de sistemas.
- Mantenimientos sencillos.
- Determina el punto concreto del error, por ejemplo, un método.
- Mayor número de Test.

### Pruebas de Integración:

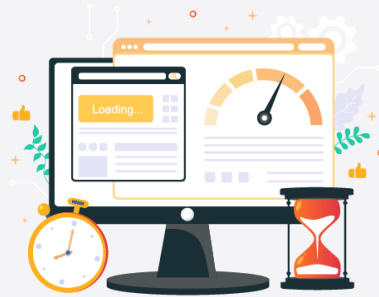
- Más código, más lentas en ejecutarse.
- No se utilizan *Mocks*.
- Mantenimientos más complejos.
- Determina una visión más global capaz de detectar errores producidos durante el tratamiento de la información.
- Menor número de Test.

97

## ¿Qué es una Prueba de Carga?

Determinar si un sistema puede controlar una carga especificada:

- número de usuarios simultáneos
- capacidad de esta de controlar las interacciones
- estabilidad de la aplicación al ejecutarse en condiciones extremas
- determinan si una aplicación sometida a un esfuerzo puede recuperarse de un error y volver a un comportamiento correcto



98

## Test Driven Development (TDD)

Puede utilizar pruebas unitarias en cualquier metodología de desarrollo, incluidos proyectos en cascada, proyectos iterativos y proyectos ágiles para detectar posibles errores y mejorar la calidad de la aplicación final.

La metodología de desarrollo llamada **Test Driven Development (TDD)**, coloca las pruebas unitarias en el centro de las prácticas de trabajo.

TDD a menudo se describe como una metodología de desarrollo separada. Algunos autores consideran que es un principio fundamental de la metodología de Programación Extrema.



99

## Test Driven Development (TDD)

### Éșçsîcîs ȳñă Rsûêčă

**El desarrollador comienza creando y codificando la prueba unitaria.**

Este paso requiere una comprensión completa del requisito funcional, que se puede obtener a partir de casos de uso o historias de usuarios.

Debido a que el desarrollador no ha escrito ningún código en la aplicación, la prueba falla.

### Rășê dê Ță Rsûêčă

**El desarrollador escribe un código rápido y simple en la aplicación para que pase la prueba.**

Durante la primera iteración, este código es frecuentemente poco elegante y puede incluir suposiciones falsas, como valores codificados.

### RêğăçțȳsîcăcîŃ

El desarrollador limpia el código de la aplicación, elimina el código duplicado, elimina los valores codificados, mejora la legibilidad y realiza otras mejoras técnicas.

**El desarrollador vuelve a ejecutar la prueba para asegurarse de que la refactorización no haya provocado un error.**

100

## Desarrollo Controlado por Comportamiento

**Behavior-Driven Development (BDD)** es el desarrollo controlado por el comportamiento.

Es un proceso que proviene de la evolución del TDD.

En BDD también se escriben pruebas antes del código, pero en vez de ser pruebas unitarias son pruebas que van a verificar que el comportamiento del código es correcto desde el punto de vista de negocio.

Se parte de historias de usuario, como, por ejemplo:

*“Como usuario quiero recibir una alerta cuando mi cuenta se quede al descubierto”.*

Ahora en vez de utilizar el “lenguaje natural” se va a utilizar un lenguaje específico de BDD como **Gherkin**, un lenguaje que entienden humanos y ordenadores.

101

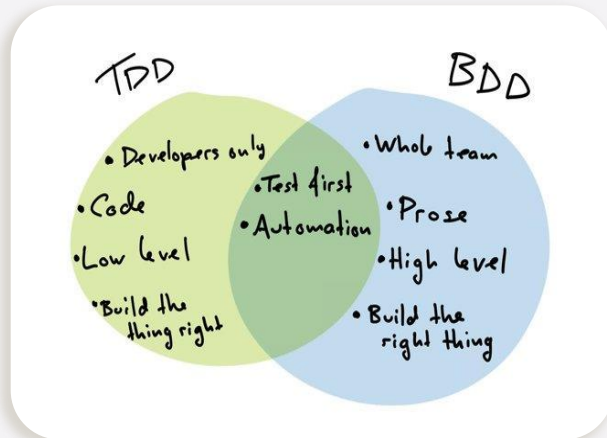


## Desarrollo Controlado por Comportamiento II

BDD sugiere probar comportamientos en lugar de ver cómo implementar el código.

Expresar una prueba es decir "debería hacer esto".

La BDD recoge las mejores prácticas del TDD e intenta guiar al desarrollador a la hora de elegir qué parte de la aplicación probar, sumándole test de aceptación puede ayudar a priorizar el desarrollo de los requisitos que más valor aporten al usuario.



102

## Lenguaje Gherkin

El **lenguaje Gherkin** define la estructura y una sintaxis básica para la descripción de las pruebas que pueden ser entendidas:

- Integrantes técnicos del equipo
- Analistas
- Programadores
- Representante del cliente.

De esta manera **mientras se generan pruebas se está generando documentación** viva que describe perfectamente como, se comporta el sistema enriqueciendo y manteniendo la documentación

103

## UN LENGUAJE COMÚN

Gherkin es un **DSL** (*Domain-Specific Language*) Lenguaje Específico de Dominio

Característica	Comportamiento	Acción	Otros
Feature	Scenario	Given	"""
	Example	When	#
	Scenario Outline	Then	@
	Background	And	
	Rule	But	

104

## Lenguaje Gherkin II

### Geătjÿsê

El elemento **Feature** proporciona el encabezado o el marco para el archivo **Feature**.

Tiene un título y un texto con una descripción de alto nivel de la función de la aplicación que se detalla en el archivo.

Contiene el listado de **Scenarios** que componen el **Feature**, los cuales se pueden agrupar por tags (por ejemplo: @HappyPath)

### Şcênşîşîp

Un **Scenario** es una lista de pasos que comienza por estas palabras claves:

- Given (|Dado|Dada)
- When (Cuando)
- Then (Entonces)
- But (Pero)
- And (Y)

### Băçlşşşşşşş

Utilizamos el **Background** para definir precondiciones para cada uno de los **Scenarios** a correr y así no ser repetitivos y focalizar los **Scenarios** en la prueba específica.

```
Feature: Búsqueda en Google
 Como usuario web, quiero buscar en Google para poder responder mis dudas.

 Scenario: Búsqueda simple en Google
 Given un navegador web en la página de Google
 When se introduce la palabra de búsqueda "pingüino"
 Then se muestra el resultado de "pingüino"
 And se muestran los siguientes resultados relacionados
 | related |
 | Pingüino emperador |
 | Videos Pingüinos |
 | Pingüino Rey |
```

105

## Ejemplo de Lenguaje Gherkin

### Característica

### Servir café

El café no se debe servir hasta que se pague  
El café no se debe servir hasta que se haya pulsado el botón  
Si no queda café entonces el dinero debe ser devuelto

### Antecedentes

Dado que estoy frente a la máquina de café  
Y tengo 1\$ para depositar

### Escenario

### Comprar último café

Dado que hay 1 café que quedan en la máquina  
Y he depositado 1 \$  
Cuando presiono el botón de café  
Entonces 1 café debería ser servido

106

## Herramientas para el Desarrollo Controlado por Comportamiento

La herramienta más destacada, basado en el patrón 'Given-When-Then' es Cucumber, un framework de test con soporte BDD donde las especificaciones están escritas en lenguaje Gherkin.

Hay muchas otras herramientas, tantas como lenguajes de programación:

- Java: JBehave, JDave, Instinct, beanSpec
- C: CSpec
- **C#:** NSpec, NBehave
- **.NET:** NSpec, NBehave, SpecFlow
- PHP: PHPSpec, Behat
- Ruby: RSpec, Cucumber
- JavaScript: JSSpec, jspec
- Python: Freshen

107

## Beneficios de las Pruebas Unitarias

### Menos tiempo para realizar pruebas funcionales

Las pruebas funcionales son costosas. Normalmente implican la apertura de la aplicación y la realización de una serie de pasos que usted (u otro usuario) debe seguir para validar el comportamiento esperado.

Por otra parte, las pruebas unitarias duran milisegundos, se pueden ejecutar con solo presionar un botón y no exigen necesariamente ningún conocimiento del sistema en general.

### Protección frente a regresión

Los defectos de regresión son aquellos que se presentan cuando se realiza un cambio en la aplicación.

Con las pruebas unitarias, es posible volver a ejecutar el conjunto completo de pruebas después de cada compilación o incluso después de cambiar una línea de código. Eso da confianza en que el nuevo código no interrumpa la funcionalidad existente.

108

## Beneficios de las Pruebas Unitarias

### Documentación ejecutable

Es posible que no siempre sea evidente lo que hace un método determinado o cómo se comporta ante una acción concreta.

Si tiene un conjunto de pruebas unitarias con un nombre adecuado, cada prueba debe ser capaz de explicar con claridad el resultado esperado de una acción determinada. Además, debe ser capaz de comprobar que funciona.

### Menos código acoplado

Cuando el código está estrechamente acoplado, puede resultar difícil realizar pruebas unitarias. Sin crear pruebas unitarias para el código que se está escribiendo, el acoplamiento puede ser menos evidente.

Al escribir pruebas para el código, este se desacopla naturalmente, ya que, de otra forma, sería más difícil de probar.

109

## Fases de una Prueba Unitaria

**Arrange**  
Organizar

**Assert**  
Afirmar

**Act**  
Actuar

110

## Características de una buena prueba unitaria

- **Rápida.** No es infrecuente que los proyectos maduros tengan miles de pruebas unitarias. Las pruebas unitarias deberían tardar muy poco tiempo en ejecutarse. Milisegundos.
- **Aislada.** Las pruebas unitarias son independientes, se pueden ejecutar de forma aislada y no tienen ninguna dependencia en ningún factor externo, como sistemas de archivos o bases de datos.
- **Reiterativa.** La ejecución de una prueba unitaria debe ser coherente con sus resultados, es decir, devolver siempre el mismo resultado si no cambia nada entre ejecuciones.
- **Autocomprobada.** La prueba debe ser capaz de detectar automáticamente si el resultado ha sido correcto o incorrecto sin necesidad de intervención humana.
- **Oportuna.** Una prueba unitaria no debe tardar un tiempo desproporcionado en escribirse en comparación con el código que se va a probar.

111

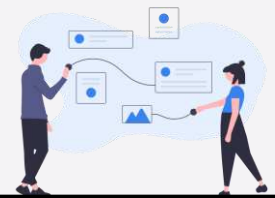
## Buenas Practicas

### Asignación de nombres a las Pruebas Unitarias

*Los estándares de nomenclatura son importantes porque expresan de forma explícita la intención de la prueba. Las pruebas van más allá de garantizar que el código funciona, también proporcionan documentación.*

El nombre de la prueba debe constar de tres partes:

- Nombre del método que se va a probar.
- Escenario en el que se está probando.
- Comportamiento esperado al invocar al escenario.



112

## Buenas Practicas

### Definir claramente las fases de las Pruebas Unitarias

*La legibilidad es uno de los aspectos más importantes a la hora de escribir una prueba. Al separar cada una de estas acciones dentro de la prueba, se resaltan claramente las dependencias necesarias para llamar al código, la forma de llamarlo y lo que se intenta declarar.*

Organizar, actuar, afirmar es un patrón común al realizar pruebas unitarias.

Como el propio nombre implica, consta de tres acciones principales:

- Organizar los objetos, crearlos y configurarlos según sea necesario.
- Actuar en un objeto.
- Afirmar que algo es como se espera.



113

## Buenas Practicas

Escribir Pruebas Unitarias lo más sencillas posible

*Las pruebas que incluyen más información de la necesaria para superarse tienen una mayor posibilidad de incorporar errores en la prueba y pueden hacer confusa su intención. Al escribir pruebas, queremos centrarnos en el comportamiento.*

El código de una prueba unitaria debe ser lo más sencillo posible que permita determinar el comportamiento que se está probando.



114

## Buenas Practicas

Evitar la lógica en las pruebas

*Al incorporar lógica al conjunto de pruebas, aumenta considerablemente la posibilidad de agregar un error. El último lugar en el que se quiere encontrar un error es el conjunto de pruebas.*

*Debe tener mucha confianza en que las pruebas funcionan, de lo contrario, no confiará en ellas.*

Al escribir las pruebas unitarias, evite la concatenación de cadenas y las condiciones lógicas como if, while, for, switch, etc.



115

## Buenas Practicas

Evitar varias comprobaciones

*Varias actuaciones deben declararse individualmente y no se garantiza que se ejecuten todas las declaraciones. En la mayoría de los marcos de pruebas unitarias, cuando se produce un error de una declaración en una prueba unitaria, se considera de forma automática que las pruebas siguientes tienen errores. Esto puede ser confuso, ya que funciones que realmente están funcionando se muestran como erróneas.*

Al escribir las pruebas, intente incluir solo una actuación por prueba.

Entre los enfoques comunes para usar solo una actuación se incluyen los siguientes:

- Crear una prueba independiente para cada actuación.
- Usar pruebas con parámetros.



116

## Buenas Practicas

Validar métodos privados mediante la prueba unitaria de métodos públicos

*Con este punto de vista, si ve un método privado, busque el método público y escriba las pruebas en ese método. Simplemente porque un método privado devuelva el resultado esperado, no significa que el sistema que finalmente llama al método privado use el resultado correctamente.*

En la mayoría de los casos, no debería haber necesidad de probar un método privado. Los métodos privados son un detalle de implementación.

Los métodos privados nunca existen de forma aislada. En algún momento, va a haber un método público que llame al método privado como parte de su implementación.

Lo que debería importarle es el resultado final del método público que llama al privado.



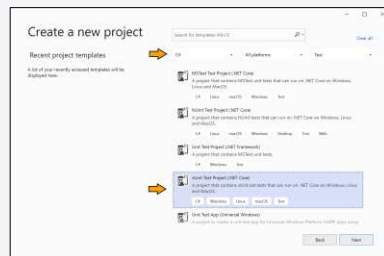
117



## xUnit



- Framework de Pruebas Unitarias de código abierto basado en .NET
- Proyecto de .NET Foundation con licencia Apache 2.0
- Diseñado por los creadores de NUnit desde cero, para no añadir características a NUnit
- Basado en el modelo de pruebas únicas
- Basado en atributos en los métodos como [Test] y [TestFixture]
- Muy extensible en comparación con NUnit y MSTest
- Permite pruebas parametrizadas
- Más de 7.500 entradas en Stackoverflow en Marzo del 2021



.NET Foundation: <https://dotnetfoundation.org/projects>

118

## Pruebas Unitarias con xUnit

Para crear las Pruebas Unitarias en el proyecto xUnit:

- Codificamos métodos que contienen las pruebas
- Nombraremos los métodos especificando el método a probar, el escenario y el comportamiento esperado
- Marcaremos los métodos con los atributos **[Fact]** o **[Theory]**

**[Fact]** indica que un método es prueba, que es ejecutado por el ejecutor de pruebas

**[Fact(Skip="Thats how you ignore a test")]** omite una prueba indicando el motivo

**[Theory]** indica un conjunto de pruebas con el mismo código y diferentes argumentos de entrada

**[InlineData]** especifica valores para los argumentos de entrada de *Theory*

119

## Pruebas Unitarias con xUnit

Para la fase de *Assert*(afirmar) el objeto **Xunit.Assert** contiene varios métodos estáticos que podemos utilizar verificar las condiciones durante la ejecución de las Pruebas Unitarias.

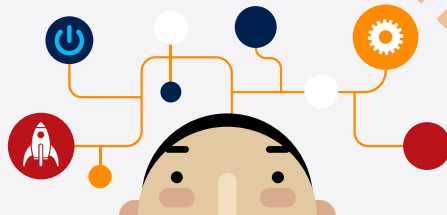
```
[Theory]
[InlineData(2)]
[InlineData(3)]
[InlineData(5)]
[InlineData(7)]
public void IsPrime_PrimesLessThan10_ReturnTrue(int value)
{
 var result = _primeService.IsPrime(value);

 Assert.True(result, $"{value} should be prime");
}
```

120

## Métodos de la Clase Xunit.Assert

- **Equal** son iguales
- **Same** referencia al mismo objeto
- **True** valor o condición verdadera
- **Null** el valor es null
- **IsType<T>** es una instancia del tipo T
- **Throws<T>** asegura que el código arroje una excepción exacta
- **InRange** asegura que un valor esté en un rango inclusivo dado
- **IsAssignableFrom<T>** valida que T se puede asignar, lo que también significa que tiene un tipo similar o deriva de T
- **NotEqual** son diferentes
- **NotSame** referencia distinto objeto
- **False** valor o condición falsa
- **NotNull** el valor distinto de null
- **IsNotType<T>** no es una instancia del tipo T



121

## Pruebas Unitarias con xUnit

### Ejecución de Pruebas en Paralelo

xUnit.net utiliza un concepto colección de pruebas para determinar la ejecución en paralelo. De forma predeterminada, cada clase de prueba es una colección de pruebas única.

Las pruebas dentro de la misma clase no se ejecutarán en paralelo entre sí.

Las pruebas de diferentes colecciones, diferentes clases de prueba pueden ejecutarse en paralelo entre sí.

Si necesitamos indicar que varias clases de prueba no deben ejecutarse en paralelo entre sí, las decoramos con el atributo `[Collection]` y un nombre de colección único.

```
[Collection("Our Test Collection #1")]
```

122

## Pruebas Unitarias con xUnit

### Ejecución de Pruebas en Paralelo II

De forma predeterminada todas las pruebas se ejecutan en paralelo. Esto se puede modificar por ensamblaje definiendo el `CollectionBehavior` en su `AssemblyInfo.cs` en su proyecto de prueba.

Para uso de separación por ensamblado:

```
[assembly: CollectionBehavior(CollectionBehavior.CollectionPerAssembly)]
```

O para ninguna paralelización en absoluto:

```
[assembly: CollectionBehavior(DisableTestParallelization = true)]
```

También se puede establecer el número máximo de subprocesos en paralelo:

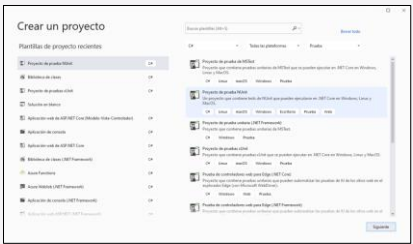
Predeterminado: número de CPU virtuales en la PC

```
[assembly: CollectionBehavior(MaxParallelThreads = n)]
```

123



- Framework de Pruebas Unitarias de código abierto adaptado de JUnit
- Proyecto de .NET Foundation con licencia MIT
- La última versión NUnit3, tiene un amplio soporte para toda la plataforma .NET
- Muy utilizado para las pruebas de automatización de Selenium
- Las pruebas se pueden realizar en GUI de Visual Studio y la consola de NUnit
- Muy válido para realizar pruebas basadas en TDD
- Más de 24.000 entradas en Stackoverflow en Marzo del 2021



.NET Foundation: <https://dotnetfoundation.org/projects>

## Pruebas Unitarias con NUnit

Para crear las Pruebas Unitarias en el proyecto NUnit:

- Codificamos métodos que contienen las pruebas
- Nombraremos los métodos especificando el método a probar, el escenario y el comportamiento esperado
- Marcaremos las clases con el atributo **[TestFixture]**
- Marcaremos los métodos con los atributos **[Test]** o **[TestCase]**

**[TestFixture]** indica una clase que contiene pruebas unitarias

**[SetUp]** indica un método que se ejecutan antes de cada método de prueba

**[TearDown]** indica un método que se ejecutan después de cada método de prueba

**[Test]** indica que un método es prueba, que es ejecutado por el ejecutor de pruebas

**[TestCase]** indica un conjunto de pruebas con el mismo código y diferentes argumentos

**[Ignore("Ignore a test")]** se usa para indicar que una prueba no debe ejecutarse por alguna razón

## Pruebas Unitarias con NUnit

```
public class BaseClass
{
 [SetUp]
 public void BaseSetUp() { /* ... */ } // Exception thrown!
 [TearDown]
 public void BaseTearDown() { /* ... */ }
}

[TestFixture]
public class DerivedClass : BaseClass
{
 [SetUp]
 public void DerivedSetUp() { /* ... */ }
 [TearDown]
 public void DerivedTearDown() { /* ... */ }

 [Test]
 public void TestMethod() { ... }
}
```

126

## Pruebas Unitarias con NUnit

Para la fase de *Assert*(afirmar) el objeto `NUnit.Framework.Assert` contiene varios métodos estáticos que podemos utilizar verificar las condiciones durante la ejecución de las Pruebas Unitarias.

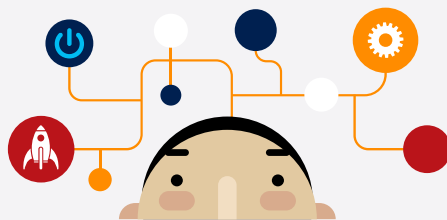
```
[TestCase(-1)]
[TestCase(0)]
[TestCase(1)]
public void IsPrime_ValuesLessThan2_ReturnFalse(int value)
{
 var result = _primeService.IsPrime(value);

 Assert.IsFalse(result, $"{value} should not be prime");
}
```

127

## Métodos de la Clase NUnit.Assert

- **AreEqual** son iguales
- **AreSame** referencia al mismo objeto
- **True** valor o condición verdadera
- **Null** el valor es null
- **IsInstanceOf** es una instancia de
- **Throws** asegura que el código arroje una excepción exacta
- **Less** valida que un valor es menor que otro
- **Greater** valida que un valor es mayor que otro
- **Contains** valida que un objeto o valor está contenido en una colección
- **AreNotEqual** son diferentes
- **AreNotSame** referencia distinto objeto
- **False** valor o condición falsa
- **NotNull** el valor distinto de null
- **IsNotInstanceOf** no es una instancia de



128

## Pruebas Unitarias con NUnit

### Múltiples Asserts

Una vez que falla una afirmación, debe finalizar la prueba.

En ocasiones es deseable continuar y acumular fallas adicionales para que puedan tratar todas a la vez, útil para probar inicialización de objetos y la apariencia de la interfaz de usuario, así como ciertos tipos de pruebas de integración.

```
[Test]
public void ComplexNumberTest()
{
 ComplexNumber result = SomeCalculation();
 Assert.Multiple(() => {
 Assert.AreEqual(5.2, result.RealPart, "Real part");
 Assert.AreEqual(3.9, result.ImaginaryPart, "Imaginary part");
 });
}
```

NUnit Constraints: <https://docs.nunit.org/articles/nunit/writing-tests/constraints/Constraints.html>

130

# MSTest



- Framawork de Pruebas Unitarias predeterminado de Visual Studio
- La última versión MSTestV2 es código abierto con licencia MIT
- Tiene soporte multiplataforma y ejecución en paralelo
- Es extensible, permite atributos y afirmaciones personalizadas
- Muy utilizado para las pruebas de automatización de Selenium
- Más de 10.000 entradas en Stackoverflow en Marzo del 2021



GitHub: <https://github.com/microsoft/testfx>

131

## Pruebas Unitarias con MSTest

Para crear las Pruebas Unitarias en el proyecto MSTest:

- Codificamos métodos que contienen las pruebas
- Nombraremos los métodos especificando el método a probar, el escenario y el comportamiento esperado
- Marcaremos las clases con el atributo **[TestClass]**
- Marcaremos los métodos con los atributos **[TestMethod]**

**[TestClass]** indica una clase que contiene pruebas unitarias

**[TestInitialize]** indica un método que se ejecutan antes de cada método de prueba

**[TestCleanup]** indica un método que se ejecutan después de cada método de prueba

**[TestMethod]** indica que un método es prueba, que es ejecutado por el ejecutor de pruebas

**[DataRow]** especifica valores para los argumentos de entrada de *TestMethod* o *DataTestMethod*

**[Ignore]** se usa para indicar que una prueba no debe ejecutarse por alguna razón

132

## Pruebas Unitarias con MSTest

El propósito de estos atributos es establecer un estado para ejecutar la prueba unitaria.

Por ejemplo, puede utilizar el método `[TestInitialize]` o `[ClassInitialize]` para copiar, modificar o crear algunos archivos de datos que su prueba utilizará.

Cree métodos marcados con el atributo `[ClassCleanup]` o `[TestCleanup]` para devolver el entorno a un estado inicial después de que se haya ejecutado una prueba.

`[ClassInitialize]` indica un método que se ejecuta antes de hacer la primera prueba en la clase.

`[ClassCleanup]` indica un método que se ejecuta cuando todas las pruebas de una clase se hayan ejecutado.

`[TestInitialize]` indica un método que se ejecutan antes de cada método de prueba

`[TestCleanup]` indica un método que se ejecutan después de cada método de prueba

133

## Pruebas Unitarias con MSTest

El objeto `Microsoft.VisualStudio.TestTools.UnitTesting.Assert` contiene varios métodos estáticos que podemos utilizar verificar las condiciones durante la ejecución de las Pruebas Unitarias.

```
[DataTestMethod]
[DataRow(-1)]
[DataRow(0)]
[DataRow(1)]
public void IsPrime_ValuesLessThan2_ReturnFalse(int value)
{
 var result = _primeService.IsPrime(value);

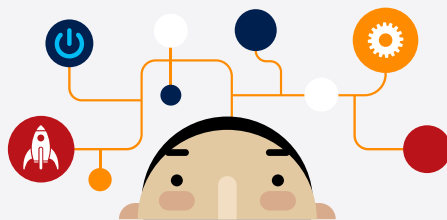
 Assert.IsFalse(result, $"{value} should not be prime");
}
```

134



## Métodos de la Clase MTest.Assert

- **AreEqual** son iguales
- **AreSame** referencia al mismo objeto
- **IsTrue** valor o condición verdadera
- **IsNull** el valor es null
- **IsInstanceOfType** es una instancia del tipo
- **Fail** se produce un error de la aserción sin hacer ninguna comprobación
- **Inconclusive** no se puede comprobar la aserción
- **AreNotEqual** son diferentes
- **AreNotSame** referencia distinto objeto
- **IsFalse** valor o condición falsa
- **IsNotNull** el valor distinto de null
- **IsNotInstanceOfType** no es una instancia del tipo



135

## archivo de proyecto

El espacio de nombres **Microsoft.VisualStudio.TestTools.UnitTesting** proporciona varias clases *Assert*:

- La clase **Assert** tiene muchos métodos para elegir y muchos de ellos tienen varias sobrecargas.
- La clase **CollectionAssert** para comparar colecciones de objetos y comprobar el estado de una o más colecciones.
- La clase **StringAssert** para comparar cadenas. Esta clase contiene una variedad de métodos útiles como **StringAssert.Contains**, **StringAssert.Matches** y **StringAssert.StartsWith**.

136

## Ejecución de Pruebas Unitarias



Test Explorer  
Live Unit Testing  
dotnet test  
VSTest.Console



Test Explorer  
Live Unit Testing  
dotnet test  
VSTest.Console  
NUnit3-Console



MSTest

Test Explorer  
Live Unit Testing  
dotnet test  
VSTest.Console

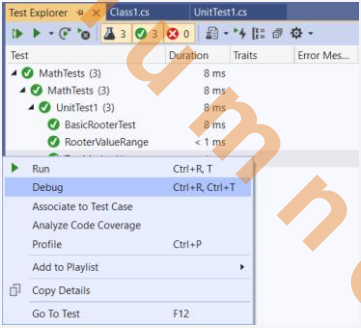
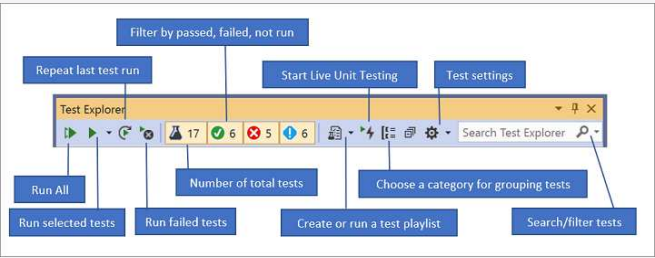


137

## Explorador Pruebas Unitarias

Se puede usar el Explorador de pruebas para iniciar una sesión de depuración para las pruebas.

La ejecución paso a paso del código con el depurador de Visual Studio permite avanzar y retroceder sin problemas entre las pruebas unitarias y el proyecto objeto de prueba.



138

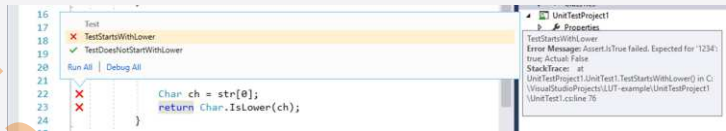
## Live Unit Testing

*Live Unit Testing* ejecuta las pruebas unitarias automáticamente y en tiempo real a medida que se realizan cambios de código. Esto le permite refactorizar y cambiar código con mayor confianza.

*Live Unit Testing* ejecuta automáticamente todas las pruebas afectadas mientras se edita el código para asegurarse de que los cambios no introducen regresiones.

Las líneas representadas con:

- ✓ cuando las pruebas son superadas
- ✗ cuando una o varias pruebas no son superadas
- cuando la línea no está afectada por ninguna prueba



Al mantener el puntero sobre el símbolo de operación correcta o con error en la ventana de código, puede ver cuántas pruebas alcanzan esa línea.

139

## Ejecución de Pruebas Unitarias

Tenemos como objetivo desarrollar un servicio *RESTful* multiplataforma y comenzamos por la creación de un proyecto Web API de ASP.NET Core programando en C#.

140

Comprobar si existe y se retorna una vista por la acción de controlador

## Ejemplo Pruebas en Controladores

```
namespace WebApplication.Controllers.Tests
{
 [TestClass()]
 public class ProductsControllerTests
 {
 [TestMethod]
 public void IndexViewTest()
 {
 var controller = new ProductsController();
 ViewResult result = controller.Index() as ViewResult;
 Assert.IsNotNull(result);
 }
 }
}
```



141

Comprobar si una acción de controlador le redirige a una segunda acción de controlador

## Ejemplo Pruebas en Controladores

```
namespace WebApplication.Controllers.Tests
{
 [TestClass()]
 public class ProductsControllerTests
 {
 [TestMethod]
 public void DetailsRedirectTest()
 {
 var controller = new ProductsController();
 var result = (RedirectToRouteResult)controller.Details(-1);
 Assert.AreEqual("Index", result.RouteValues["action"]);
 }
 }
}
```



142

Comprobar si existe y se retorna una vista por la acción de controlador

## Ejemplo Pruebas en Controladores

```
namespace WebApplication.Controllers.Tests
{
 [TestClass]
 public class ProductsControllerTests
 {
 [TestMethod]
 public void IndexViewTest()
 {
 var controller = new ProductsController();
 ViewResult result = controller.Index() as ViewResult;
 Assert.IsNotNull(result);
 }
 }
}
```



143

## ¿Qué es hacer Mocking?

En nuestras Pruebas Unitarias necesitaremos:

- conectar con bases de datos
- realizar operaciones contra el sistema de ficheros
- interactuar con APIs externas

Esto hace difícil que nuestros test unitarios sean realmente unitarios, ya que añaden una dependencia de elementos sobre los no siempre vamos a tener control.

Dicha dependencia causa a menudo problemas de velocidad de ejecución, lo que hace lento la ejecución de las pruebas.

Para solucionar estos problemas hacemos *Mocking*, utilizando *mocks*, objetos simulados que imitan el comportamiento de objetos reales.

144

## Mocking en .NET

Los frameworks hacen sencillo y rápido hacer Mocking.

En .NET disponemos de muchos frameworks:

- FakeItEasy
- JustMock
- **Moq**
- RhinoMocks
- CreateAndFake
- Magento
- LightMock

*Con Moq podemos crear  
mocks fácilmente,  
incluso sin conocimientos  
previos de técnicas de  
mocking.*

145

## InMemory

**Microsoft.EntityFrameworkCore.InMemory** es un proveedor de base de datos en memoria para Entity Framework Core.

Es útil cuando desea probar componentes utilizando algo que se aproxime a la conexión a la base de datos real, sin la sobrecarga de las operaciones reales de la base de datos.

- La ventaja del proveedor en memoria es que puede probar el código de su aplicación contra una base de datos en memoria en lugar de instalar y configurar la base de datos real.
- Es una base de datos de propósito general, diseñada estrictamente con el propósito de probar el código de su aplicación y no una base de datos relacional.

146

## Código de Ejemplo, InMemory para EF Core

### Mocking de la Base de Datos

```
var builder = new DbContextOptionsBuilder<ModelNorthwind>()
 .UseInMemoryDatabase(databaseName: "Test")
 .Options;

_context = new ModelNorthwind(builder);

var customers = new Customers
{
 CustomerID = "ANATR",
 CompanyName = "Ana Trujillo Emparedados y helados",
 ContactName = "Ana Trujillo",
};

_context.Customers.Add(customers);
_context.SaveChanges();
```



147



148