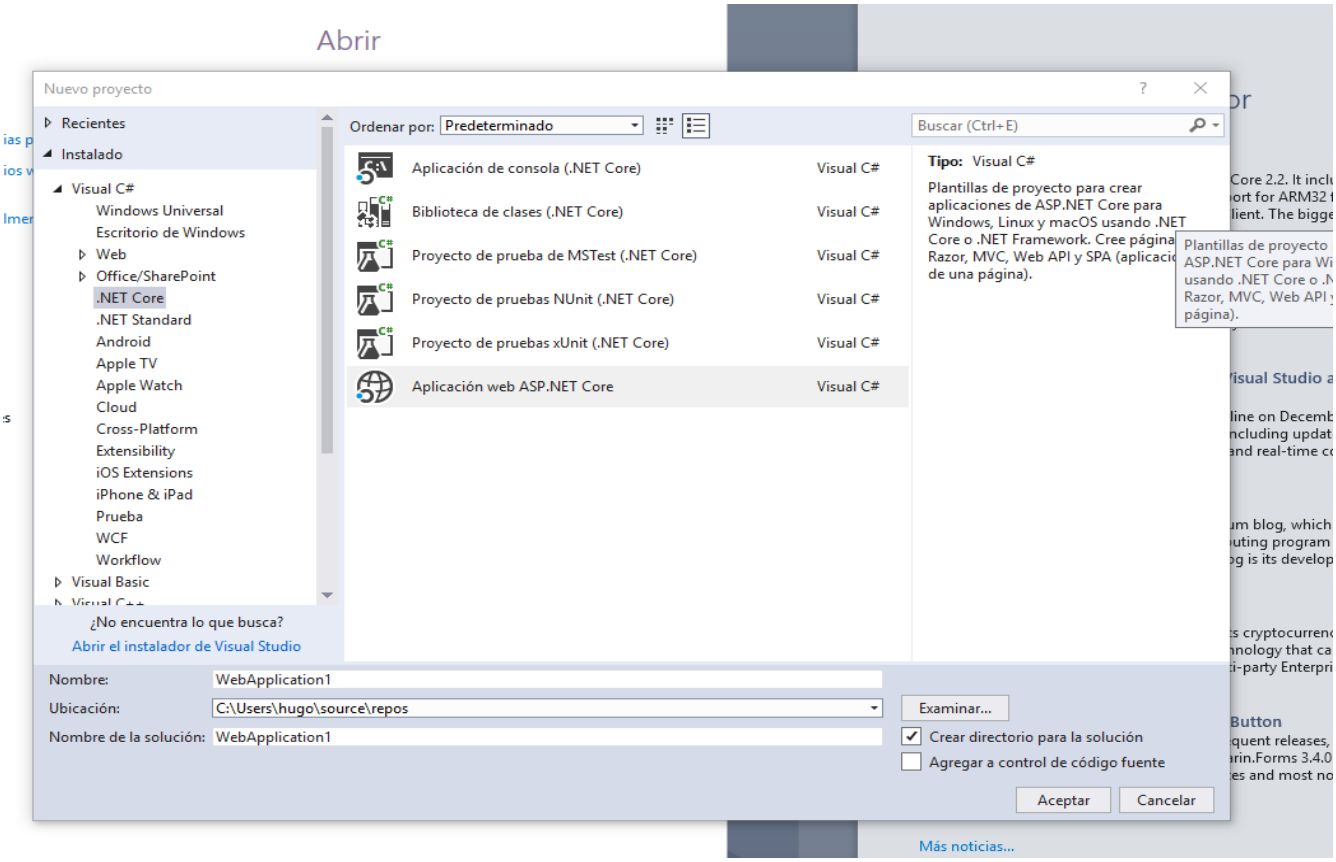
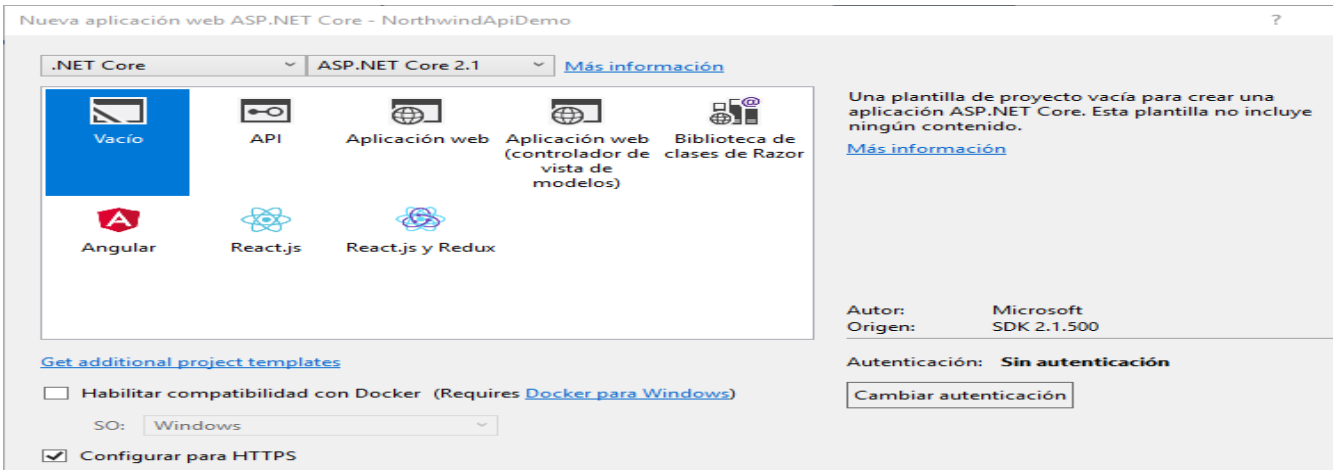
S03C33EliminarRegistroBBDDASPNET

Nuevo Proyecto

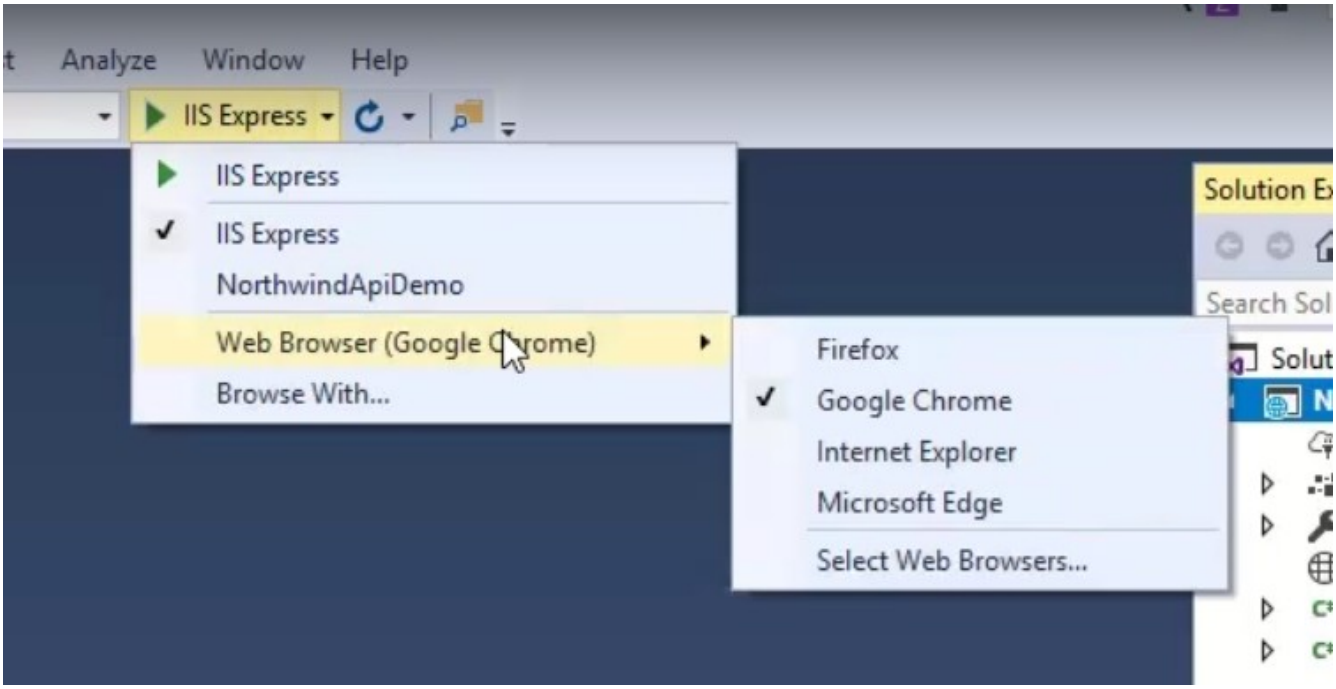
.NetCore ---- App web ASP.NETCore



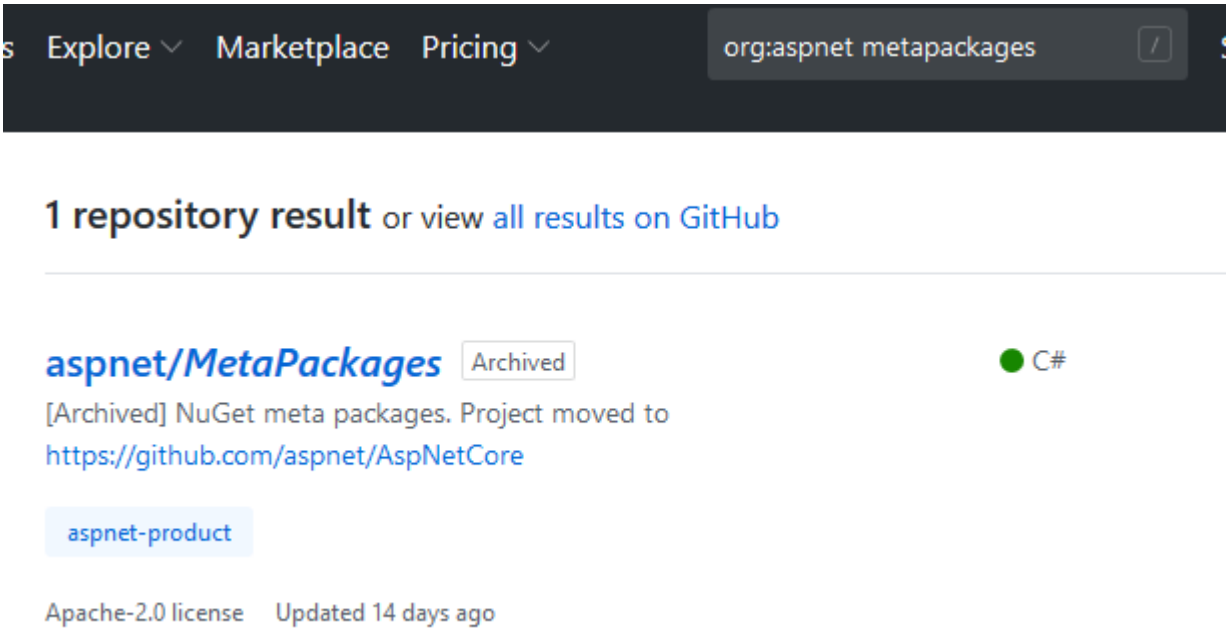

ProyectoVacio---- ASP.NET CORE 2.1



Elegir navegador y lanzar Proyecto




https://github.com/aspnet

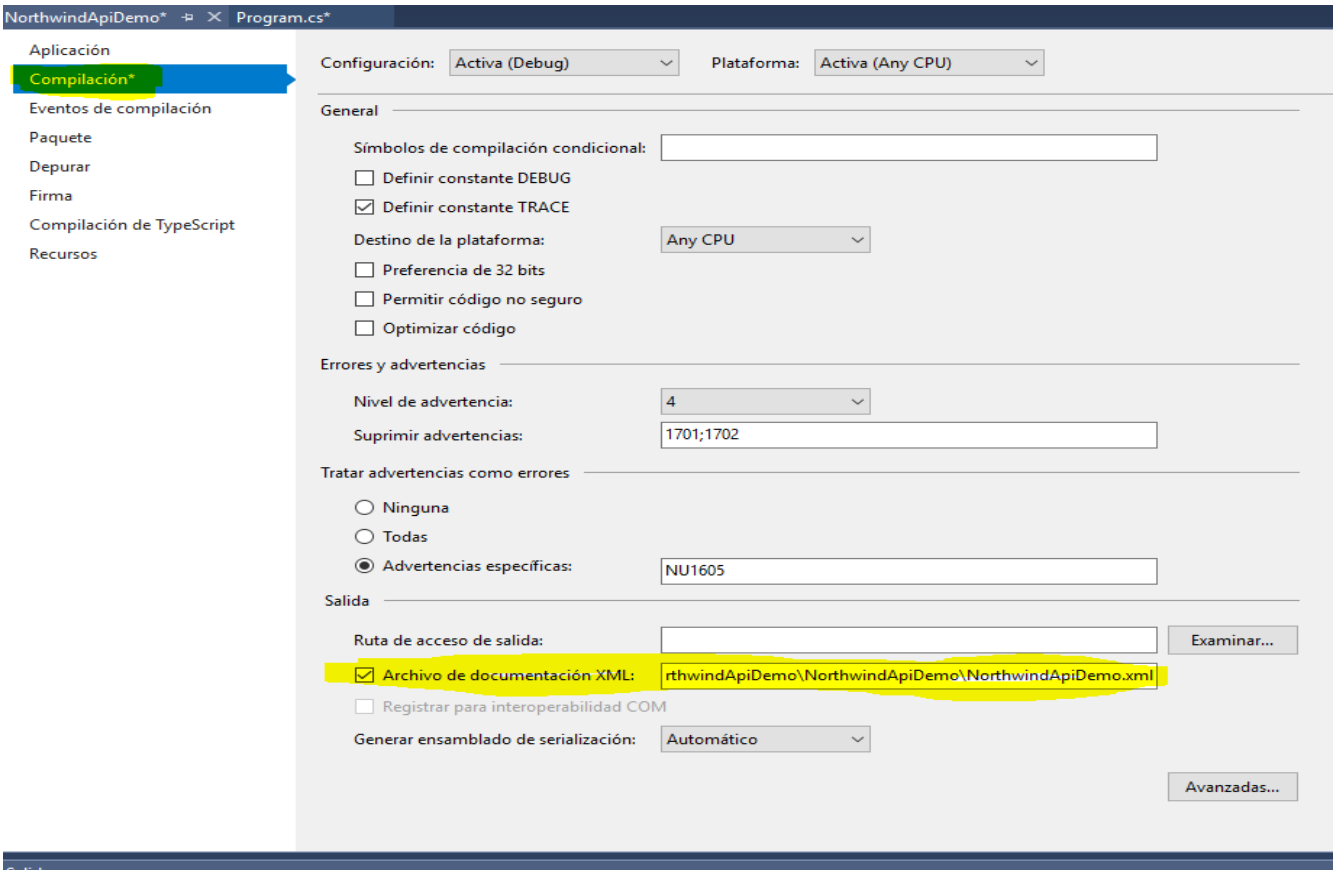



Program.cs basado webHost.cs

```
public static void Main(string[] args)
{
    CreateWebHostBuilder(args).Build().Run();
}

///WebHost.xxx  podemos configurarlo como queramos. Este es el default que hay de base
///Startup classe de configuracion de diferentes servicios que crearemos se  llama asi por defecto
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>();
```

```
///WebHost.xxx  podemos configurarlo como queramos. Este es el default que hay de base
///Startup classe de configuracion de diferentes servicios que crearemos se  llama asi por
defecto
        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>();
```

crearDocumentacion
fichero ---- BD – Propiedades



BA---Compilar- Recompilar

proyecto --- BD  ---- Abrir carpeta Explorador windows ---
explorador--\bin\Debug\netcoreapp2.1\nombreProyecto.xml

**S01C08ConfiguracionAppWebAPINETCore**

**CONFIGURAR APP**

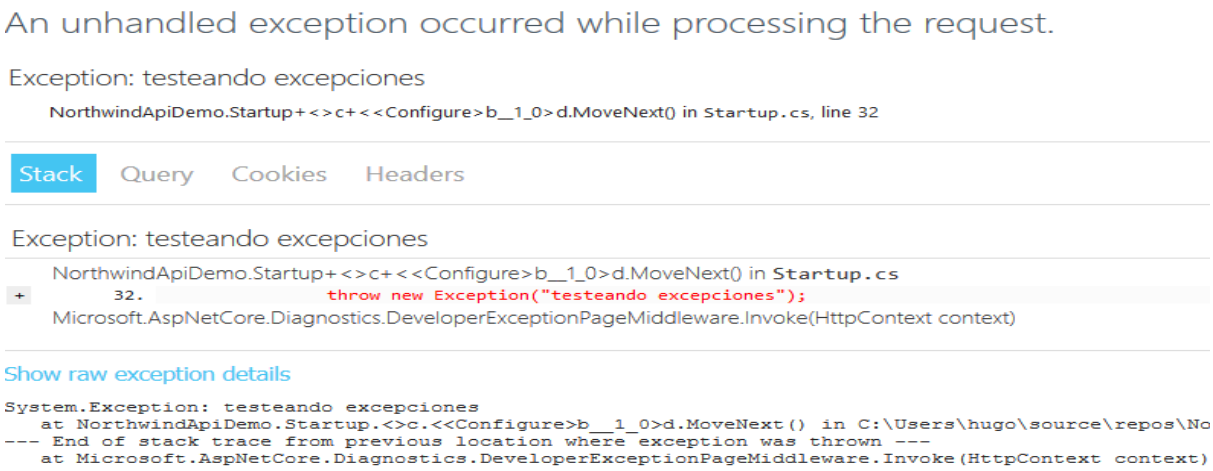
Startup.cs

```
  public void ConfigureServices(IServiceCollection services)
```

CrearExcepcion:

```
// For more information on how to configure your application, visit ...
public void ConfigureServices(IServiceCollection services)
{
}

// This method gets called by the runtime. Use this method to configu
public void Configure(IApplicationBuilder app, IHostingEnvironment env
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    ///Aqui es donde se configurala accion que sucedera:
    ///lanzamos excepcion personalizada
    app.Run(async (context) =>
    {
        throw new Exception("testeando excepciones");
    });
```

**Visualizacio excepcion:**

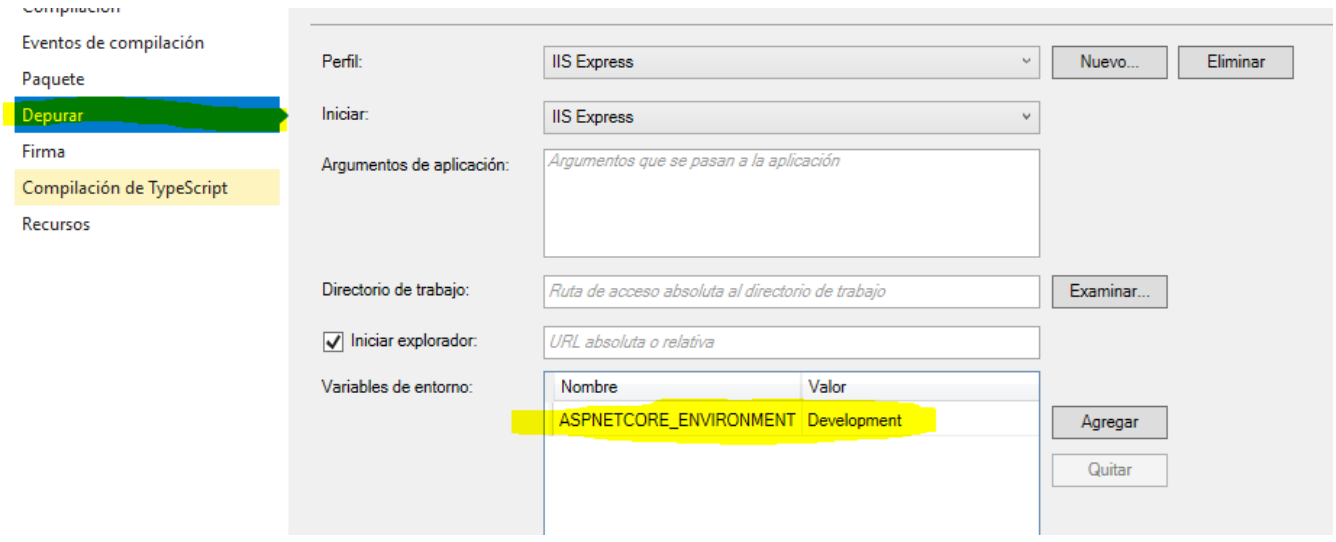


**Entorno:**

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
/// <summary>
/// Configuramos entorno y como sale
/// </summary>
/// <param name="app"></param>
/// <param name="env">Aqui llamanos al entorno con el qual trabajamos</param>
///  if (env.IsDevelopment()) aqui chequeamos si el entorno es de desarollo
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    ///Aqui es donde se configurala accion que sucedera:
    ///lanzamos excepcion personalizada
    app.Run(async (context) =>
    {
        throw new Exception("testeando excepciones");
    });
```

**Proyecto + BD + propiedades.**

Aqui podemos cambiar al variable por loo que queramos (integracion, produccion,..). Para que este cambio se produzca hay que reiniciar VS.



Middleware es software que se sitúa entre un sistema operativo y las aplicaciones que se ejecutan en él. Básicamente, funciona como una capa de traducción oculta para permitir la comunicación y la administración de datos en aplicaciones distribuidas. A veces, se le denomina “plumbing” (tuberías), porque conecta dos aplicaciones para que se puedan pasar fácilmente datos y bases de datos por una “canalización”.
los middleware normalmente en ASP.net se inician con app.Usexxxx , app.RunXXX, app.Mapxxx.

```
///  if (env.IsDevelopment()) aqui chequeamos si el entorno es de desarollo
///  else{ app.UseExceptionHandler(); }si el evento del entono no esta declarado
///  app.UseStatusCodePages();  middleware (tuberia codigo pagina)
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler();
    }

    app.UseStatusCodePages();
```

Middleware muy utilizado y sencillo para marcar errores app.UseStatusCodePages();

**MVC**

**Modelo-vista-controlador** (**MVC**) es un patrón de arquitectura de software, que separa los **datos** y la **lógica de negocio** de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres **componentes** distintos que son el **modelo**, la **vista** y el **controlador**, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario

El **Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (**lógica de negocio**). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.[12]

- El **Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo' (véase ***Middleware***).
- La **Vista**: Presenta el 'modelo' (información y *lógica de negocio*) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho 'modelo' la información que debe representar como salida.

**S01C10AddMildwareAndServiceMVC**

StarUp.cs

Llamanos a los servicios MVC  services.AddMvc();:

```
public void ConfigureServices(IServiceCollection services)
    {
            services.AddMvc();
     }
```

Llamar al midlware  del MVC:

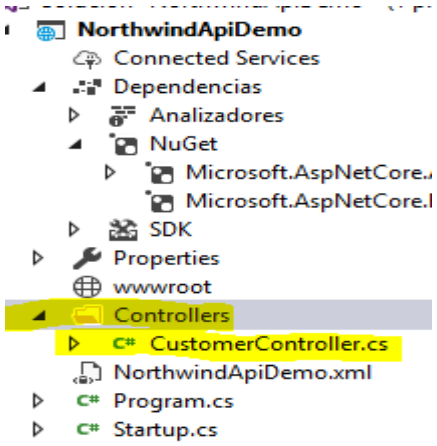
```
 public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
               app.UseMvc();
       }
```

Ademas de cometar todos los app.run

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler();
    }

    app.UseStatusCodePages();

    app.UseMvc();

    ///Aqui es donde se configurala accion que sucedera:
    ///lanzamos excepcion personalizada
    //app.Run(async (context) =>
    // {
    //     throw new Exception("testeando excepciones");
    //});

    ///Aqui es donde se configurala accion que sucedera:
    ///en este caso sino hay valor devuelve esta respuesta escrita
    /*
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello World!");
    });
    */
```

**S01C12MetaPaquetesASP.NET**

Un controlador nos va a permitir  hacer operaciones hacia unos datos (BBDD,XML,...)
Crear nuestro propio customer controller:



```
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace NorthwindApiDemo.Controllers
{
    public class CustomerController:Controller
    {
        public JsonResult GetCustomers()
        {
            return new JsonResult(new List<object>()
            {
                new {CustomerID = 1, ContactName = "Anderson"},
                new {CustomerID = 2, ContactName = "Solaris"}
            });
        }
    }
}
```

**S01C13RoutingASPNET**

routing predefinifo

crear nuestro ruta:

StatUp.cs  y dentro  public void Configure:

```
app.UseMvc(config =>
    {
    config.MapRoute(
        name: "Default",
        template: "{contoller}/{action}/{id?}",
        defaults: new
        {
            controller = "Home",
            action ="Index"
        });

    });
```

**routing por atributo**

StartUp.cs  y metodo  Configure:

```
app.UseMvc();
```

CustomerController.cs :

```
public class CustomerController:Controller
    {
        //[HttpGet("prueba")]
       [HttpGet("api/customers")]
        public JsonResult GetCustomers()
        {

            return new JsonResult(new List<object>()
            {
                new {CustomerID = 1, ContactName = "Anderson"},
                new {CustomerID = 2, ContactName = "Solaris"}
             });
        }
    }
```

navegador:  https://localhost:44320/api/customers

RUTA DEFAULT:

```
  //como rutaDefault
    [Route("api/customers")]
    public class CustomerController:Controller
    {
        //[HttpGet("prueba")]
        //Ruta especifica
        //[HttpGet("api/customers")]
        //como rutaDefault
        [HttpGet()]
        public JsonResult GetCustomers()
        {

            return new JsonResult(new List<object>()
            {
                new {CustomerID = 1, ContactName = "Anderson"},
                new {CustomerID = 2, ContactName = "Solaris"}
             });
        }
    }
```

S01C14PrimerModeloASPNET

1)Creamos carpeta “Models” en  el proyecto

1.a) creamos un clase CustomersDTO:

```
namespace NorthwindApiDemo.Models
{
    public class CustomersDTO
    {
        public  int  ID { get; set; }
        public string CustomerID { get; set; }
        public string CompanyName { get; set; }
        public string ContactName { get; set; }
        public string ContactTitle { get; set; }
        public string Address { get; set; }
        public string City { get; set; }
        public string Region { get; set; }
        public string PostalCode { get; set; }
        public string Country { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }



    }
}
```

2) Crear a nivel raiz Proyecto una clase Repository:

```
using Foundation.ObjectHydrator;
namespace NorthwindApiDemo
{
    public class Repository
    {
        //crearemos una propiedad que se Inicializara al inicio de proyecto
        //por ese motivo lo ponemos static
        public static Repository Instance { get; } = new Repository();
        //Llamaremos  al modelo Customers  mediante una lista
        public IList<Models.CustomersDTO> Customers { get; set; }
        //crear contructor
        //raiz Proyecto , BD, paquete Nuguet buscar: "hydrator" ,
       //foundation.ObjectHydrator:core (instalar)
        public Repository()
        {
            Hydrator<Models.CustomersDTO> hydrator = new Hydrator<Models.CustomersDTO>();
            //creamos el conjunto de 5 resultados
            Customers = hydrator.GetList(5);
        }

    }
}
```

3)/Conttollers/CustomerControllers.cs:

```
  public JsonResult GetCustomers()
        {
            //C14 llamamos a una lista creada en un repositorio Repository mediante Hydrator
            return new JsonResult(Repository.Instance.Customers);
```

S01C15DevolverUnSoloResultadoASPNET

Obtener datos de un valor especifico,, creando una controlador especifico. Llamandolo especificamente:

1)/Controllers/CustomerControllers.cs:

```
 //C15 creamos un elemento para llamar a un valor especifico
        //IActionResult cambiamos por JsonResult porque es mas generico y hereda de el JsonResult
        //evitandonos problemas como el NotFound
        //sino encontramos nadassugue este metodo
        //[HttpGet("{id})" el obetenido sera el que valor url (http://localhost/api/customers/3)
        [HttpGet("{id}")]
        public IActionResult GetCustomer(int id)
        {

            var result = Repository.Instance.Customers.FirstOrDefault(c => c.ID == id);


            if(result == null)
            {
                return NotFound();
            }
            return Ok(result);
        }
```

https://localhost:44320/api/customers/3

2) Llammarlo : https://localhost:44320/api/customers/3

S01C16DevolviendoSubconjuntoDatosASPNET

Vamos a llamar a una lista de valores que vamos a crear dentro del tipo  Clientes

1)/Models/CustomersDTO.cs:

```
public class CustomersDTO
    {
        ...

        public ICollection <OrdersDTO> Orders { get; set; }

    }
```

2)Crear las classe OrdersDTO en models (/Models/OrdersDTO.cs:):

```
public class OrdersDTO
    {
        public int OrderId { get; set; }
        ...
        public string ShipCountry { get; set; }
    }
```

3) Repository.cs(dentro constructor):

```
Random random = new Random();
            Hydrator<Models.OrdersDTO> ordersHydrator = new Hydrator<Models.OrdersDTO>();
            foreach (var customer in Customers)
            {
                customer.Orders = ordersHydrator.GetList(random.Next(1, 10));
            }
```

4)Creamos un controlador de Orders(/Contollers/OrdersControllers.cs) api/customers/23/orders/1:

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

namespace NorthwindApiDemo.Controllers
{
    //Esta controladora controlara el tema de las ordenes
    //Creamos la ruta inicial
    [Route("api/customers")]
    public class OrdersController:Controller
    {
        //la id del cliente y las ordenes
       [HttpGet("{customerId}/orders")]
        public  IActionResult GetOrders(int customerId)
        {
            var customer = Repository.Instance.Customers.FirstOrDefault(c => c.ID ==
customerId);
            if (customer == null){ return NotFound();}
            return Ok(customer.Orders);
        }
```

```
        //creamos la lista de orders
       [HttpGet("{customerId}/orders/{id}")]
       public IActionResult GetOrder(int customerId,int id)
        {
            var customer = Repository.Instance.Customers.FirstOrDefault(c => c.ID ==
customerId);

            if (customer == null)
            {
                return NotFound();
            }
            //podemos ver una orden especifica de un cliente
            var order = customer.Orders.FirstOrDefault(o => o.OrderId == id);
            //return Ok(customer.Orders);
            if (customer == null)
            {
                return NotFound();
            }
            return Ok(order);
        }
    }
}
```

S02C17CrearNuevoRegistroASPNET

Vamos a crear una clase en modelo para hacer creaciones de nuevo registro solo apareceran los datos a rellenar.

Creamos un nuevo modelo para crear registros, sin el ID */Models*/OdersForCreactionDTO.cs

```
 public class OrdersForCreationDTO
    {

        public string CustomerId { get; set; }
        public int EmployeeId { get; set; }
        public DateTime OrderDate { get; set; }
        public DateTime RequiredDate { get; set; }
        public DateTime ShippedDate { get; set; }
        public int ShipVia { get; set; }
        public decimal Freight { get; set; }
        public string ShipName { get; set; }
        public string ShipAddress { get; set; }
        public string ShipCity { get; set; }
        public string ShipRegion { get; set; }
        public string ShipPostalCode { get; set; }
        public string ShipCountry { get; set; }

    }
```

Añadimos en *Controller/OrdersControllers* un nuevo proceso para esta funcion.

```
 //S02c17CreacionNuevoregistro
        //[FromBody] este dato se cogera de dentro de la solicitud
        //[HttpPost("{customerId/orders}")] al coger el dato de la se pone post
        [HttpPost("{customerId}/orders")]
        public IActionResult CreaterOrder(int customerId, [FromBody]
Models.OrdersForCreationDTO order)
        {
            //Si el campos es erroneo o vacio damos este error
            if (order == null)
            {
                return BadRequest();
            }

            var customer = Repository.Instance.Customers.FirstOrDefault(c => c.ID ==
customerId);

            if (customer == null)
            {
                return NotFound();
            }

            //extraer el valor  maximo de la ID
            var maxOrderId = Repository.Instance.Customers.SelectMany(c => c.Orders).Max(o =>
o.OrderId);

            //crear un a Orden nuevo con un autoSuma de OrderId
            var finalOrder = new Models.OrdersDTO()
            {
                OrderId = maxOrderId++,
                CustomerId = order.CustomerId,
                EmployeeId = order.EmployeeId,
                OrderDate = order.OrderDate,
                RequiredDate = order.RequiredDate,
                ShippedDate = order.ShippedDate,
                ShipVia = order.ShipVia,
                Freight = order.Freight,
                ShipName = order.ShipName,
                ShipAddress = order.ShipAddress,
                ShipCity = order.ShipCity,
                ShipRegion = order.ShipRegion,
                ShipPostalCode = order.ShipPostalCode,
                ShipCountry = order.ShipCountry
            };

            //CreatedAtRoute(string routeName, object routeValues, object value)
            return CreatedAtRoute("GetOrder", new { customerId = customerId, id =
finalOrder.OrderId},finalOrder);

        }
```

Y modificamos metodo GetOrder para poder trabajar:

```
 //creamos la lista de orders
        //Name ="GetOrder" es para llamar al metodo desde la misma funcion
        [HttpGet("{customerId}/orders/{id}",Name ="GetOrder")]
        public IActionResult GetOrder(int customerId, int id)
```

Chequear en Postman:

obtener listas clientes, ordenes

**GET** https://localhost:44320/api/customers (lista de clientes)
https://localhost:44320/api/customers/3 (cliente especifico)
https://localhost:44320/api/customers/3/orders

Crear una nueva orden:

**POST** BODY – tipo raw – C/P una orden y eliminar ordenId
HEADER CONTENT-TYPE app/json
SEND (BOTTON)
CHECK STATUS 201 CREATED

```
{
        "customerId": "testing create",
        "employeeId": 55,
        "orderDate": "2011-08-18T12:34:41+02:00",
        "requiredDate": "2028-09-16T17:13:50+02:00",
        "shippedDate": "2012-05-13T04:30:27+02:00",
        "shipVia": 11,
        "freight": 0,
        "shipName": "Lorem",
        "shipAddress": "24836 NOTTINGHAM Pl.",
        "shipCity": "Lorem ipsum dolor sit amet, consectetur adip",
        "shipRegion": "Lorem ipsum",
        "shipPostalCode": "87816-7931",
        "shipCountry": "GH"
}
```

chequear nuevo registro:

HEADER – LOCATION (Copiar dirrecion)
POST → GET P dirrecion + send (button)

S02C18ValidacionDatosASPNET

En Modelo o classe por ejemplo (*/Models*/OdersForCreactionDTO.cs) y añadiños un metodo para chequear lo.

La biblioteca **System.ComponentModel.DataAnnotations** tiene unas validaciones automaticas configuradas.

```
[System.ComponentModel.DataAnnotations.MinLength(10,ErrorMessage ="demasiado corto")]
        public string ShipAddress { get; set; }
```

En Controller/OrdersController.cs se dentro de un metodo añadimos este codigo para escribir el mensaje en el modelo.

S02C19ActualizarRegistrosASPNET

Creamos un nuevo modelo (/Models/OrdersForUpdateDTO.cs) lo copiamos OrderForCreation

Metodo dentro (/Controllers/OrdersController)

```
   //S02c19ActualizarRegistro
        //[HttpPut("{customerId}/orders/{id}")] actualizar
        //[FromBody] este dato se cogera de dentro de la solicitud
        // return NoContent()  Noactualizar la url de este registro
        [HttpPut("{customerId}/orders/{id}")]
        public IActionResult UpdateOrder(int customerId,int id,[FromBody]
Models.OrdersForUpdateDTO order)
        {
            //Si el campos es erroneo o vacio damos este error
            if (order == null)
            {
                return BadRequest();
            }
            //sino valida se escribe el mensaje de la classe mediante ModelState
            if (!ModelState.IsValid) { return BadRequest(ModelState); }

            var customer = Repository.Instance.Customers.FirstOrDefault(c => c.ID ==
customerId);

            if (customer == null)
            {
                return NotFound();
            }

            var orderFromRepository = customer.Orders.FirstOrDefault(o => o.OrderId == id);

            if (orderFromRepository == null)
            {
                return NotFound();
            }

            orderFromRepository.CustomerId = order.CustomerId;
            orderFromRepository.EmployeeId = order.EmployeeId ;
```

```
            orderFromRepository.OrderDate = order.OrderDate;
            orderFromRepository.RequiredDate = order.RequiredDate;
            orderFromRepository.ShippedDate = order.ShippedDate;
            orderFromRepository.ShipVia = order.ShipVia;
            orderFromRepository.Freight = order.Freight;
            orderFromRepository.ShipName = order.ShipName;
            orderFromRepository.ShipAddress = order.ShipAddress;
            orderFromRepository.ShipCity = order.ShipCity;
            orderFromRepository.ShipRegion = order.ShipRegion;
            orderFromRepository.ShipPostalCode = order.ShipPostalCode;
            orderFromRepository.ShipCountry = order.ShipCountry;

            return NoContent();
        }
```

chequear Postman:

obtener listas clientes, ordenes

**GET** https://localhost:44320/api/customers (lista de clientes)
https://localhost:44320/api/customers/3 (cliente especifico)
https://localhost:44320/api/customers/3/orders

Crear una nueva orden:

**PUT** BODY – tipo raw – C/P una orden y eliminar ordenId
HEADER CONTENT-TYPE app/json
SEND (BOTTON)
CHECK STATUS 204 NO CONTENT

```
{
    "customerId": "UPDATE Lorem ipsum",
    "employeeId": 26,
    "orderDate": "2012-05-08T10:12:48+02:00",
    "requiredDate": "2028-10-31T16:12:00+01:00",
    "shippedDate": "2026-05-14T15:01:56+02:00",
    "shipVia": 74,
    "freight": 0,
    "shipName": "UPDATE Lorem",
    "shipAddress": "UPDATE 17211 ELLIOT Way",
    "shipCity": "Lorem ipsum dolor sit a",
    "shipRegion": "Lorem ipsum dolor sit amet",
    "shipPostalCode": "74690-3761",
    "shipCountry": "RE"
}
```

chequear nuevo registro:

HEADER – LOCATION (Copiar dirrecion)
POST → GET P dirrecion + send (button)

S02C20ActualizarRegistrosParcialASPNET

Metodo dentro (/Controllers/OrdersController):

```
  //S02c20ActualizarRegistroParcial
        //[HttpPut("{customerId}/orders/{id}")] actualizar
        //[FromBody] este dato se cogera de dentro de la solicitud
        // return NoContent()  Noactualizar la url de este registro
        //ejecutar la instancia patchDocument.ApplyTo(orderToUpdate);
        [HttpPatch("{customerId}/orders/{id}")]
        public IActionResult UpdateOrder(int customerId, int id, [FromBody]
JsonPatchDocument<Models.OrdersForUpdateDTO> patchDocument)
        {
            //Si el campos es erroneo o vacio damos este error
            if (patchDocument == null)
            {
                return BadRequest();
            }
            //sino valida se escribe el mensaje de la classe mediante ModelState
            if (!ModelState.IsValid) { return BadRequest(ModelState); }

            var customer = Repository.Instance.Customers.FirstOrDefault(c => c.ID ==
customerId);

            if (customer == null)
            {
                return NotFound();
            }

            var orderFromRepository = customer.Orders.FirstOrDefault(o => o.OrderId == id);

            if (orderFromRepository == null)
            {
                return NotFound();
            }
            var orderToUpdate = new Models.OrdersForUpdateDTO()
            {
                   CustomerId = orderFromRepository.CustomerId,
                  EmployeeId = orderFromRepository.EmployeeId,
                OrderDate = orderFromRepository.OrderDate,
                RequiredDate = orderFromRepository.RequiredDate,
                ShippedDate = orderFromRepository.ShippedDate,
                ShipVia = orderFromRepository.ShipVia,
                Freight = orderFromRepository.Freight,
                ShipName = orderFromRepository.ShipName,
                ShipAddress = orderFromRepository.ShipAddress,
                ShipCity = orderFromRepository.ShipCity,
                ShipRegion  = orderFromRepository.ShipRegion,
                ShipPostalCode = orderFromRepository.ShipPostalCode,
                ShipCountry = orderFromRepository.ShipCountry,

            };

            patchDocument.ApplyTo(orderToUpdate, ModelState);
```

```
            if(!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            orderFromRepository.CustomerId = orderToUpdate.CustomerId;
            orderFromRepository.EmployeeId = orderToUpdate.EmployeeId;
            orderFromRepository.OrderDate =orderToUpdate.OrderDate;
            orderFromRepository.RequiredDate =orderToUpdate.RequiredDate;
            orderFromRepository.ShippedDate =orderToUpdate.ShippedDate;
            orderFromRepository.ShipVia =orderToUpdate.ShipVia;
            orderFromRepository.Freight =orderToUpdate.Freight;
            orderFromRepository.ShipName =orderToUpdate.ShipName;
            orderFromRepository.ShipAddress =orderToUpdate.ShipAddress;
            orderFromRepository.ShipCity =orderToUpdate.ShipCity;
            orderFromRepository.ShipRegion =orderToUpdate.ShipRegion;
            orderFromRepository.ShipPostalCode =orderToUpdate.ShipPostalCode;
            orderFromRepository.ShipCountry =orderToUpdate.ShipCountry;
            return NoContent();
        }

       chequear Postman:

obtener listas clientes, ordenes

GET    https://localhost:44320/api/customers (lista de clientes)
       https://localhost:44320/api/customers/3  (cliente especifico)
       https://localhost:44320/api/customers/3/orders

Crear una nueva orden:

PATCH  BODY – tipo raw –  C/P una orden y eliminar ordenId
       HEADER CONTENT-TYPE app/json
       SEND (BOTTON)
       CHECK STATUS 204 NO CONTENT
[
       {
       "op": "replace",
       "path": "/shipName",
       "value": "I was patched"
       }
]

chequear nuevo registro:

HEADER  – LOCATION (Copiar dirrecion)
POST → GET P dirrecion + send (button)
```

S02C21EliminarRegistroASPNET

*Controllers*/OrdersController.cs  y crear metodo:

```
//S02c20EliminaRegistro
        //[HttpDelete("{customerId}/orders/{id}")] actualizar
        //[FromBody] este dato se cogera de dentro de la solicitud
        // return NoContent()  Noactualizar la url de este registro
        [HttpDelete("{customerId}/orders/{id}")]
        public IActionResult DeleteOrder(int customerId, int id)
        {
            var customer = Repository.Instance.Customers.FirstOrDefault(c => c.ID ==
customerId);

            if (customer == null)
            {
                return NotFound();
            }
            //podemos ver una orden especifica de un cliente
            var order = customer.Orders.FirstOrDefault(o => o.OrderId == id);
            //return Ok(customer.Orders);
            if (order == null)
            {
                return NotFound();
            }

            customer.Orders.Remove(order);
            return NoContent();
        }
```

obtener listas clientes, ordenes

**GET**  https://localhost:44320/api/customers (lista de clientes)
https://localhost:44320/api/customers/3  (cliente especifico)
https://localhost:44320/api/customers/3/orders

Crear una nueva orden:

**DELETE** BODY – tipo raw –  C/P una orden y eliminar ordenId
HEADER CONTENT-TYPE app/json
SEND (BOTTON)
CHECK STATUS 204 NO CONTENT
https://localhost:44320/api/customers/3/orders/3

chequear nuevo registro:

HEADER  – LOCATION (Copiar dirrecion)
POST → GET P dirrecion + send (button)

# S03C33EliminarRegistroBBDDASPNET

S02C22InstalarBBDDASPNET

navegador:

g:  northwind database

https://docs.microsoft.com/es-es/dotnet/framework/data/adonet/sql/linq/downloading-sample-databases

https://github.com/Microsoft/sql-server-samples/tree/master/samples/databases/northwind-pubs

abrir manager sql server  conectar bbdd con  “windows Autentificacion”

crear script, pegar script, execute (icono play)

 si marca error elminar lineas errores

VE---BS---Herramientas---Administrar paquetes Nuguets

       Microsoft.EntityFrameworkCore.SqlServer
       Microsoft.EntityFrameworkCore.Design
       Microsoft.EntityFrameworkCore.Tools

VE---BS---Herramientas---Administrar paquetes Nuguets---consola

Scaffold-DbContext "Server =.\SQLEXPRESS;DataBase= Northwind;Trusted_Connection=True"

Microsoft.EntityFrameworkCore.SqlServer  -OutputDir EFModels

/Startup.cs añadir:

```
 public void ConfigureServices(IServiceCollection services)
        {
            ...
            //services.AddDbContext<NorthwindContext>(options =>
            services.AddDbContext<NorthwindContext>(options =>
            {
                options.UseSqlServer("Server =.\\SQLEXPRESS;DataBase=
Northwind;Trusted_Connection=True");
            });
        }
```

or lo ideal

EFModels/NorthwindContext.cs(onConfiguring metodo) y comentarlo y copiar server L32:

```
"Server =.\\SQLEXPRESS;DataBase= Northwind;Trusted_Connection=True"

 //seconfigura en Starup.cs y copiar "Server =.\\SQLEXPRESS;DataBase=
Northwind;Trusted_Connection=True" en starup.cs
        /*
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            if (!optionsBuilder.IsConfigured)
            {
#warning To protect potentially sensitive information in your connection string, you should
move it out of source code. See http://go.microsoft.com/fwlink/?LinkId=723263 for guidance on
storing connection strings.
                optionsBuilder.UseSqlServer("Server =.\\SQLEXPRESS;DataBase=
Northwind;Trusted_Connection=True");
            }
        }
        */
```

S03C25RegistroServicioRepositorio

Crear un una carpeta en el proyecto ***Services***. *Y* posteriormente una nueva Interface **Services/IcustomerRepository.cs:**

```
 //inyectar dentro de las mismas controladores
    //
    public interface ICustomerRepository
    {
        //ObtenerCliente y saber si queremos ordenes o no
        IEnumerable<EFModels.Customers> GetCustomers();
        EFModels.Customers GetCustomers(string customerId, bool includeOrders);
        //obtener ordenes
        IEnumerable<EFModels.Orders> GetOrders(string customerId);
        EFModels.Orders GetOrders(string customerId, int orderId);

    }
```

**crear una clase Services/CustomerRepository.cs:**

```
using Microsoft.EntityFrameworkCore;

 public class CustomerRepository : ICustomerRepository
    {

        private EFModels.NorthwindContext _context;

        public CustomerRepository(NorthwindContext context)
        {
            this._context = context;
        }
```

```
        public IEnumerable<Customers> GetCustomers()
        {

            //retornar lista ordenada por compañias
            return _context.Customers.OrderBy(c => c.CompanyName).ToList();

            //throw new NotImplementedException();
        }

        public Customers GetCustomers(string customerId, bool includeOrders)
        {
            //si tiene ordenes idCliente comoordernes
            if (includeOrders)
            {
                //using Microsoft.EntityFrameworkCore;   Include
                return _context.Customers.Include(c => c.Orders).Where(c =>  c.CustomerId ==
customerId).FirstOrDefault();
            }

            //Solo cliente
            return _context.Customers.Where(c => c.CustomerId == customerId).FirstOrDefault();
            //throw new NotImplementedException();
        }

        public IEnumerable<Orders> GetOrders(string customerId)
        {

            return _context.Orders.Where(c => c.CustomerId == customerId).ToList();
            //throw new NotImplementedException();
        }

        public Orders GetOrders(string customerId, int orderId)
        {

            return _context.Orders.Where(c => c.CustomerId == customerId && c.OrderId ==
orderId).FirstOrDefault();
            //throw new NotImplementedException();
        }
    }
```

**StartUp.cs:**

```
 public void ConfigureServices(IServiceCollection services)
        {
            ...

            //s03c25
            //addTransient (crear servicios cada vez solicitados)
            //addScope //crear servicios 1 vez por solicitud
            //addSingleton //crear servicio 1 solicitado
            services.AddScoped<Services.ICustomerRepository, Services.CustomerRepository>();
        }
```

*EFModels*/NorthwindContext.cs, crear constructor:

```
 //S03C26
        //utilizar servicio de repositorio
        public NorthwindContext(DbContextOptions<NorthwindContext> options)
            : base(options)
        {
        }
```

S03C26MostrarResultadosdesdeBBDDASPNET

a)metodo guarro

Controllers/CustomerController.cs :

```
 //S03C26
        //using NorthwindApiDemo.Services;
        private ICustomerRepository _customerRepository;
        public CustomerController(ICustomerRepository customerRepository)
        {
            _customerRepository = customerRepository;
        }

        //[HttpGet("prueba")]
        //Ruta especifica
        //[HttpGet("api/customers")]
        //como rutaDefault
        [HttpGet()]
        public JsonResult GetCustomers()
        {

            #region ApartirS03C26orUsode ServiciosRepositorios
            //metodo guarro o resultado directo
            var customers = _customerRepository.GetCustomers();
            //return new JsonResult(customers);


            #endregion

            #region previously S03c26

            //C14 llamamos a una lista creada en un repositorio Repository mediante Hydrator
```

```
            //Descomentar linea  inferior solamente sino se hace servir Servicios de
repositorio hasta C03S26
            //return new JsonResult(Repository.Instance.Customers);

            //ejemplo sencillo de lista C12 C13
            /*
            return new JsonResult(new List<object>()
            {
                new {CustomerID = 1, ContactName = "Anderson"},
                new {CustomerID = 2, ContactName = "Solaris"}
             });
             */

            #endregion

        }
```

b)metodo elagante:

Inyectar de forma Bonita lista clientes, creamos un nuevo modelo,
Models/CustomerWithoutOrders.cs:

```
public class CustomerWithoutOrdersDTO
    {
        public int ID { get; set; }
        public string CustomerID { get; set; }
        public string CompanyName { get; set; }
        public string ContactName { get; set; }
        public string ContactTitle { get; set; }
        public string Address { get; set; }
        public string City { get; set; }
        public string Region { get; set; }
        public string PostalCode { get; set; }
        public string Country { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }


    }
```

Controllers/CustomersController.cs:

```
 [HttpGet()]
        public JsonResult GetCustomers()
        {

            #region ApartirS03C26orUsode ServiciosRepositorios
            //metodo guarro o resultado directo
            var customers = _customerRepository.GetCustomers();
            //return new JsonResult(customers);
            //metodo elegante
            var results = new List<Models.CustomerWithoutOrdersDTO>();
            foreach (var customer in customers)
            {
                results.Add(new Models.CustomerWithoutOrdersDTO()
```

```
                {

                    CustomerID  = customer.CustomerId,
                    CompanyName  = customer.CompanyName,
                    ContactName  = customer.ContactName,
                    ContactTitle  = customer.ContactTitle,
                    Address  = customer.Address,
                    City  = customer.City,
                    Region  = customer.Region,
                    PostalCode  = customer.PostalCode,
                    Country  = customer.Country,
                    Phone  = customer.Phone,
                    Fax  = customer.Fax
                });
            }

            return new JsonResult(results);

            #endregion
```

Verificacion by postman:

obtener listas clientes, ordenes

**GET** https://localhost:44320/api/customers (lista de clientes)

S03C27AutoMapperBBDDASPNET

Añadir paquete “automapper”(herraminetas-nuget--proyecto – BD- ref-add).

/Startup.cs :

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
              ...

            #region mapeo

            //mappeo odea linko Customer con su modelo
            AutoMapper.Mapper.Initialize(config =>
            {
                config.CreateMap<Customers, Models.CustomerWithoutOrdersDTO>();

            });

            #endregion

        …
}
```

Controller/CustomerControllers.cs:

```
         [HttpGet()]
        public JsonResult GetCustomers()
        {
            ...
            #region ApartirS03C26orUsode ServiciosRepositorios
            //metodo guarro o resultado directo
            var customers = _customerRepository.GetCustomers();
            //return new JsonResult(customers);
            //metodo elegante

            #endregion


            //S03C27AUTOMAPEO
            var results =
AutoMapper.Mapper.Map<IEnumerable<Models.CustomerWithoutOrdersDTO>>(customers);

            ...
            return new JsonResult(results);
        }
```

S03C28RegresandoUnicoRegistroBBDDASPNET

AutoMappeo y directamente desde la BBDD escogemos sqi queremos clientes con ordenes o sin ordenes. Añadimos Booleano  bool includeOrders = false

Controllers.CustomerController :

```
//S03C28 trabajando desde BBDD, verificamos si quiere orden de trabajo
        [HttpGet("{id}")]
        public IActionResult GetCustomer(string id,bool includeOrders = false)
        {

            var customer = _customerRepository.GetCustomers(id, includeOrders);


            if (customer == null)
            {
                return NotFound();
            }
            //S03C28 Pasar cliente con ordenes directamente BBDD
            if (includeOrders)
            {
                var customerResult = AutoMapper.Mapper.Map<Models.CustomersDTO>(customer);
                return Ok(customerResult);
            }

            //S03C28 Pasar cliente sin ordenes  BBDD
            var customerResultOnly =
AutoMapper.Mapper.Map<Models.CustomerWithoutOrdersDTO>(customer);
            return Ok(customerResultOnly);  }
```

Ahora configuramos el mapeo anterior en StartUp.cs:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
       {

             ...

            //S02C27 AUTOMAPEO

            #region mapeo

            //mappeo odea linko Customer con su modelo
            AutoMapper.Mapper.Initialize(config =>
            {
                config.CreateMap<Customers, Models.CustomerWithoutOrdersDTO>();
                //S03C28 Mapeo de cliente con o sin ordenes
                config.CreateMap<Customers, Models.CustomersDTO>();
                config.CreateMap<Orders, Models.OrdersDTO>();

            });

            #endregion

              …
       }
```

Verificacion by postman:

obtener listas clientes, ordenes

**GET** https://localhost:44320/api/customers (lista de clientes)

Obtener Cliente y ordenes:

**GET** https://localhost:44320/api/customers/Paco?includeOrders=true

S03C29RegresandoOrdenesBBDDASPNET

Creamos Nuevo metodo valipar ordenes *Services*/IcustomerRepository:

```
 public interface ICustomerRepository
    {
       ...
        //S03cC29 creamos un booleano para verificar si cliente existe
        bool CustomerExists(string customerId);
    }
```

Implementamos *Services*/CustomerRepository:

```
//S03C029 extraer ordenes por metodo
        //si el cliente existe o no
        public bool CustomerExists(string customerId)
        {

            return _context.Customers.Any(c => c.CustomerId == customerId);
            //throw new NotImplementedException();
        }
```

Controller/OrdersController:

```
 //S03C29 crear variable  y constructor
        //using NorthwindApiDemo.Services;
        private Services.ICustomerRepository _customerRepository;
        public OrdersController(Services.ICustomerRepository customerRepository)
        {
            _customerRepository = customerRepository;
        }
```

Controller/OrdersController:

```
 public IActionResult GetOrders(string customerId)
        {
            //S03C29 Creamos variable para extraerOrdenes
            //Verificamos si hay un cliente anterior
            if (!_customerRepository.CustomerExists(customerId))
            {
                return NotFound();
            }

            var orders = _customerRepository.GetOrders(customerId);

            //S03C29 Creamos una variable para crear una lista  obtenida de una variable de
datos
            var ordersResult = AutoMapper.Mapper.Map<IEnumerable<Models.OrdersDTO>>(orders);

            return Ok(ordersResult);

       }

        //> 03C29 public IActionResult GetOrder(int customerId, int id)
        [HttpGet("{customerId}/orders/{id}",Name ="GetOrder")]
        public IActionResult GetOrder(string customerId, int id)
        {

            if (!_customerRepository.CustomerExists(customerId))
            {
                return NotFound();
            }

            var order = _customerRepository.GetOrders(customerId, id);

            if(order == null)
            {
                return NotFound();
            }

            var orderResult = AutoMapper.Mapper.Map<Models.OrdersDTO>(order);

            return Ok(orderResult);
        }
```

Verificacion by postman:

obtener listas clientes, ordenes

**GET** https://localhost:44320/api/customers (lista de clientes)

Obtener Cliente y ordenes:

**GET** https://localhost:44320/api/customers/Paco/orders (solo listado ordene de cliente)
Obtener orden especifica de Cliente:

**GET** https://localhost:44320/api/customers/Paco/orders/10450 (orden especifica de cliente)

S03C30CrearRegistroBBDDASPNET

Service/ICustomerRepository

```
 public interface ICustomerRepository
    {
       ...
        //S03c30 crear registro BBDD
        void AddOrder(string customerId, EFModels.Orders orders);
        bool Save();
    }
```

Service/CustomerRepository

```
 //s03c30
        //obtener cliente y guradar orden
        public void AddOrder(string customerId, Orders orders)
        {
            var customer = GetCustomers(customerId, false);
            customer.Orders.Add(orders);
            //throw new NotImplementedException();
        }

 //c03s30 hacemo persistencia para guardar ordenes
        //si hacemos la validacion sobre uno o mas registros
        public bool Save()
        {

            return (_context.SaveChanges() >= 0);
            //throw new NotImplementedException();
        }
```

Modficar Controllers/OrderController:

```
  //S03c30CreacionNuevoregistro
        //[FromBody] este dato se cogera de dentro de la solicitud
        //[HttpPost("{customerId/orders}")] al coger el dato de la se pone post
        [HttpPost("{customerId}/orders")]
        public IActionResult CreaterOrder(string customerId, [FromBody]
Models.OrdersForCreationDTO order)
        {
            //Si el campos es erroneo o vacio damos este error
            if (order == null)
            {
                return BadRequest();
            }
            //sino valida se escribe el mensaje de la classe mediante ModelState
           if( !ModelState.IsValid) { return BadRequest(ModelState); }


            if (!_customerRepository.CustomerExists(customerId))
            {
```

```
                return NotFound();
            }

            //crear un a Orden nuevo con un autoSuma de OrderId
            var finalOrder = AutoMapper.Mapper.Map<Orders>(order);

               //S03c30
            _customerRepository.AddOrder(customerId, finalOrder);

            if(!_customerRepository.Save())
            {
                return StatusCode(500, "Please verify your data");
            }


            var customerCreated = AutoMapper.Mapper.Map<Models.OrdersDTO>(finalOrder);



            //CreatedAtRoute(string routeName, object routeValues, object value)
            return CreatedAtRoute("GetOrder", new { customerId = customerId, id =
customerCreated.OrderId}, customerCreated);
        }

/starUP:

 //mappeo odea linko Customer con su modelo
            AutoMapper.Mapper.Initialize(config =>
            {
                config.CreateMap<Customers, Models.CustomerWithoutOrdersDTO>();
                //S03C28 Mapeo de cliente con o sin ordenes
                config.CreateMap<Customers, Models.CustomersDTO>();
                config.CreateMap<Orders, Models.OrdersDTO>();
                //S03C30 Crear registro
                config.CreateMap<Models.OrdersForCreationDTO,Orders>();

            });

obtener listas clientes, ordenes

GET    https://localhost:44320/api/customers (lista de clientes)

Obtener Cliente y ordenes:

GET    https://localhost:44320/api/customers/Paco/orders (solo listado ordene de cliente)
Obtener orden especifica de Cliente:

Copiar una orden, pegarla en pestña Body(tipo raw, Json(app/json)) e eliminar IdOrder:

POST   https://localhost:44320/api/customers/Paco/orders/ (orden especifica de cliente)
```

S03C31ActualizarRegistroBBDDASPNET

Controller/orderController

```
 //S03c31ActualizarRegistro
        //[HttpPut("{customerId}/orders/{id}")] actualizar
        //[FromBody] este dato se cogera de dentro de la solicitud
        // return NoContent()  Noactualizar la url de este registro
        [HttpPut("{customerId}/orders/{id}")]
        public IActionResult UpdateOrder(string customerId,int id,[FromBody]
Models.OrdersForUpdateDTO order)
        {
            //Si el campos es erroneo o vacio damos este error
            if (order == null)
            {
                return BadRequest();
            }
            //sino valida se escribe el mensaje de la classe mediante ModelState
            if (!ModelState.IsValid) { return BadRequest(ModelState); }

            //S03C30 actulizar registro
            if (!_customerRepository.CustomerExists(customerId))
            {
                return NotFound();
            }

            var existingOrder = _customerRepository.GetOrders(customerId,id);
            if(existingOrder ==null)
            {
                return NotFound();
            }
            //sobrecargamos metodo para acutilizar orden existente
            AutoMapper.Mapper.Map(order, existingOrder);

            if (!_customerRepository.Save())
            {
                return StatusCode(500, "Please verify your data");
            }

            return NoContent();
        }
```

StartUp :

```
 //mappeo odea linko Customer con su modelo
            AutoMapper.Mapper.Initialize(config =>
            {
               ...
                //S03C1 uPDATE REGISTRO
                config.CreateMap<Models.OrdersForUpdateDTO, Orders>();

            });
```

obtener listas clientes, ordenes

**GET** https://localhost:44320/api/customers (lista de clientes)

Obtener Cliente y ordenes:

**GET** https://localhost:44320/api/customers/Paco/orders (solo listado ordene de cliente)
Obtener orden especifica de Cliente:

Copiar una orden, pegarla en pestña Body(tipo raw, Json(app/json)) e eliminar IdOrder:

**PUT** https://localhost:44320/api/customers/Paco/orders/66666 (orden especifica de cliente)

cofigo 204 Funciona Ok

verificamos rl cambio

**GET** https://localhost:44320/api/customers/Paco/orders/66666

S03C32ActualizarRegistroParcialmenteBBDDASPNET

Conroller/orderControllers:

```
 //S03c32ActualizarRegistroParcial
        //[HttpPut("{customerId}/orders/{id}")] actualizar
        //[FromBody] este dato se cogera de dentro de la solicitud
        // return NoContent()  Noactualizar la url de este registro
        [HttpPatch("{customerId}/orders/{id}")]
        public IActionResult UpdateOrder(string customerId,int id,[FromBody]
JsonPatchDocument<Models.OrdersForUpdateDTO> patchDocument)
        {
            //Si el campos es erroneo o vacio damos este error
            if (patchDocument == null)
            {
                return BadRequest();
            }
            //sino valida se escribe el mensaje de la classe mediante ModelState
            if (!ModelState.IsValid) { return BadRequest(ModelState); }

            if (!_customerRepository.CustomerExists(customerId))
            {
                return NotFound();
            }

            var existingOrder = _customerRepository.GetOrders(customerId, id);
            if (existingOrder == null)
            {
                return NotFound();
            }

            var orderToUpdate =
AutoMapper.Mapper.Map<Models.OrdersForUpdateDTO>(existingOrder);

            patchDocument .ApplyTo(orderToUpdate, ModelState);
```

```
            //S03C32UpdatePartialMappeo
            //para hacer correciones y validaciones metodo especificado
            TryValidateModel(orderToUpdate);


            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }


            //S03C32 UpdatePartialMappeo
            //mapeados los datos cambiados
            AutoMapper.Mapper.Map(orderToUpdate, existingOrder);

            if (!_customerRepository.Save())
            {
                return StatusCode(500, "Please verify your data");
            }

            return NoContent();

        }
```

obtener listas clientes, ordenes

**GET** https://localhost:44320/api/customers (lista de clientes)

Obtener Cliente y ordenes:

**GET** https://localhost:44320/api/customers/Paco/orders (solo listado ordene de cliente)
Obtener orden especifica de Cliente:

Body(tipo raw, Json(app/json)) e eliminar IdOrder, poner campo y valor a cambiar:

```
[
	{
		“op”: “replace”,
		“path”: “/shipName”,
		“value”: “PACTHED”
	}
]
```

**PATCH** https://localhost:44320/api/customers/Paco/orders/66666 (orden especifica de cliente)

cofigo 204 Funciona Ok

verificamos rl cambio

**GET** https://localhost:44320/api/customers/Paco/orders/66666

S03C33EliminarRegistroBBDDASPNET

Services/ICustomerRepository:

```
 public interface ICustomerRepository
    {
      ...
        //S03C33 EliminarRegistro
        void DeleteOrder(EFModels.Orders orders);
    }
```

Services/CustomerRepository:

```
//S03C33 EliminarRegistro
        public void DeleteOrder(Orders order)
        {
            _context.Orders.Remove(order);
            //throw new NotImplementedException();
        }
```

Conroller/orderControllers:

```
 //S03C33EliminaRegistroMapeo
        //[HttpDelete("{customerId}/orders/{id}")] actualizar
        //[FromBody] este dato se cogera de dentro de la solicitud
        // return NoContent()  Noactualizar la url de este registro
        [HttpDelete("{customerId}/orders/{id}")]
        public IActionResult DeleteOrder(string customerId, int id)
        {
            //sino valida se escribe el mensaje de la classe mediante ModelState
            if (!ModelState.IsValid) { return BadRequest(ModelState); }

            if (!_customerRepository.CustomerExists(customerId))
            {
                return NotFound();
            }

            var existingOrder = _customerRepository.GetOrders(customerId, id);
            if (existingOrder == null)
            {
                return NotFound();
            }

            /*
            //podemos ver una orden especifica de un cliente
            var order = customer.Orders.FirstOrDefault(o => o.OrderId == id);
            //return Ok(customer.Orders);
            if (order == null)
            {
                return NotFound();
            }
            */
            _customerRepository.DeleteOrder(existingOrder);
```

```
            if (!_customerRepository.Save())
            {
                return StatusCode(500, "Please verify your data");
            }

            return NoContent();
        }
```

obtener listas clientes, ordenes

**GET** https://localhost:44320/api/customers (lista de clientes)

Obtener Cliente y ordenes:

**GET** https://localhost:44320/api/customers/Paco/orders (solo listado ordene de cliente)

**DELETE** https://localhost:44320/api/customers/Paco/orders/66666 (orden especifica de cliente)

verificamos rl cambio

**GET** https://localhost:44320/api/customers/Paco/orders/66666