

CLASE 3

Laboratorio

Nota importante: Esta ejercitación en etapas e integradora, cubre fundamentalmente los conceptos teóricos explicados en la clase 3, y agrega también conceptos relacionados sobre trabajo con datos y funcionalidad CRUD de las clases 4 y 6, según este detalle:

- o Clase 3:
 - o Creación de un proyecto Web Api con scaffolding.
 - o Uso de del ORM Entity Framework Core.
 - o Creación de una API CRUD con EF y bases de datos Sql Server.
 - o Pruebas con Postman.
- o Clase 4
 - o Trabajo con Controladores y Acciones.
 - o Ruteo.
 - o Programación asincrónica.
 - o Inyección de dependencias.
 - o Instanciación de servicios.
- o Clase 6
 - o Trabajo con recursos.
 - o DTOs y AutoMapper.
 - o Creación de recursos con POST.
 - o Actualización completa de recursos con PUT.
 - o Eliminación de recursos con DELETE.

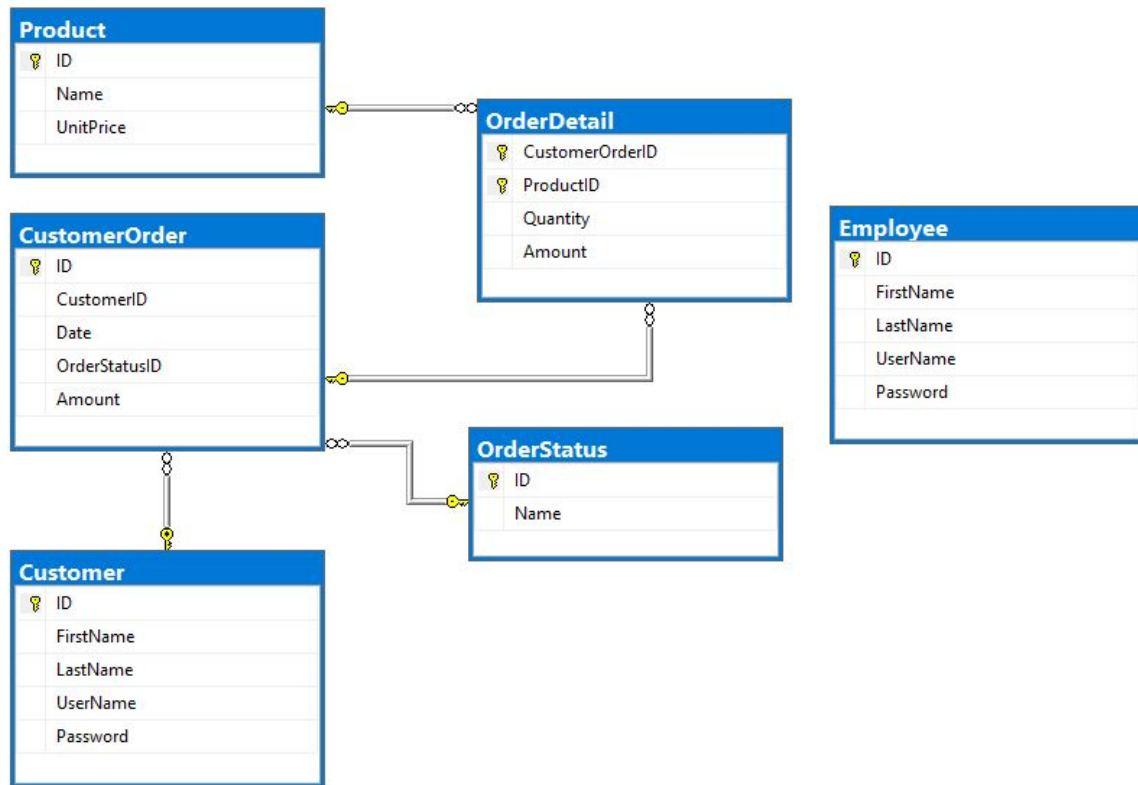
El código completo del proyecto web api terminado es **StoreWebApi** y *está en la sección de **Descargas de Clase 3***, donde también está el script para crear la base de datos en Sql Server.

Creación de un WebApi con Entity Framework Core y AutoMapper

Veremos cómo crear una webapi backend que acceda a la base de datos para que luego alguna aplicación web, mobile, etc. la pueda consumir, usando **ASP .NET Core** y **Entity Framework Core** como tecnología de acceso a los datos del almacenamiento. La base de datos está en **SQL Server**. Además,

utilizaremos **AutoMapper** para construir **DTOs** a partir de los modelos generados por EF Core.

La base de datos con la que vamos a trabajar tiene la siguiente estructura, *en la sección de descargas correspondiente a esta clase está el script sql* para que la pueda crear.



1. Conéctate a **SQL Server Management Studio** y ejecuta el script **StoreDB_Script.sql** para crear la base de datos.

```
create database StoreDB
go
use StoreDB
go

create table Employee(
    ID int identity(1,1) primary key not null,
    FirstName varchar(50) not null,
    LastName varchar(50) not null,
    UserName varchar(32) not null,
    Password varchar(32) not null
)

create table Customer(
    ID int identity(1,1) primary key not null,
    FirstName varchar(50) not null,
```

```

        LastName varchar(50) not null,
        UserName varchar(32) not null,
        Password varchar(32) not null
    )

insert into Customer values ('Juan', 'Perez', '123',
    CONVERT(VARCHAR(32), HashBytes('MD5', 'abc'), 2))

select * from customer

create table Product(
    ID int identity(1,1) primary key not null,
    Name varchar(255) not null,
    UnitPrice decimal(8,2) not null
)

create table OrderStatus(
    ID int identity(1,1) primary key not null,
    Name varchar(255) not null
)

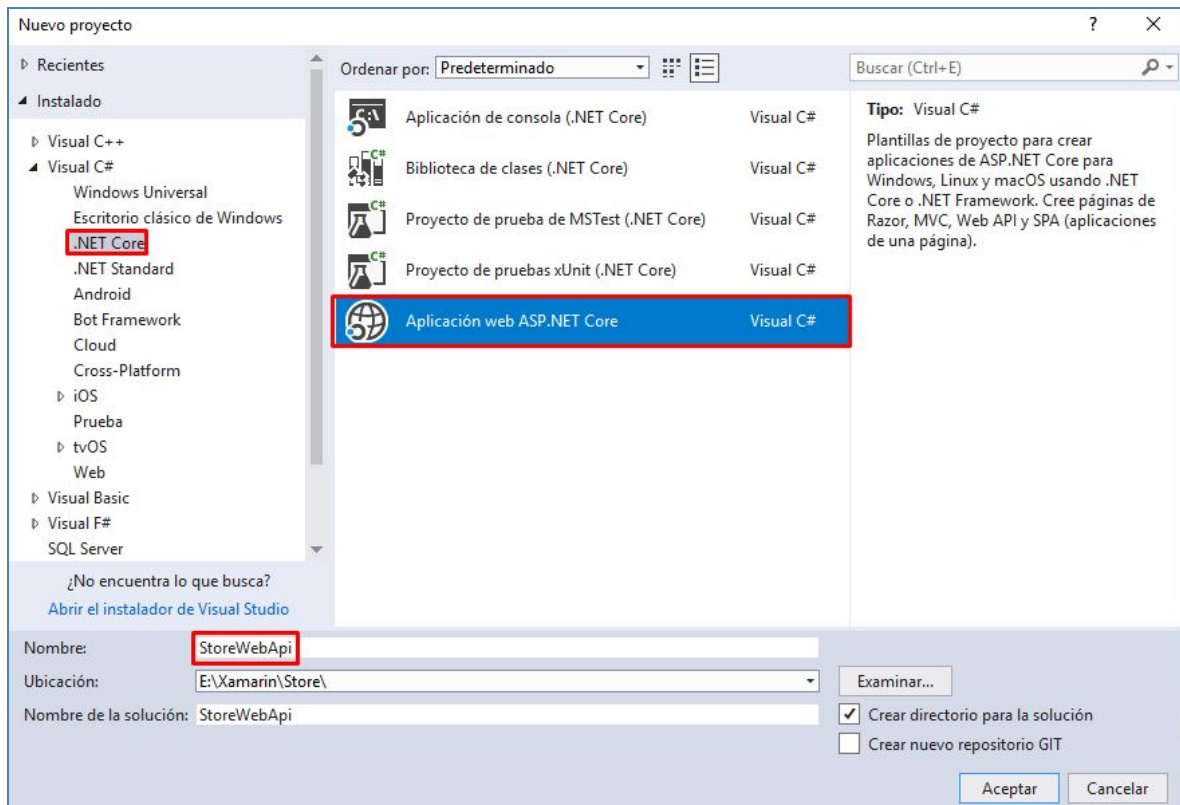
insert into OrderStatus(Name)
values ('1-Unpaid'), ('2-Paid'), ('3-Processing'), ('4-Shipped'),
('5-Delivered')

create table CustomerOrder(
    ID int identity(1,1) primary key not null,
    CustomerID int not null,
    Date smalldatetime not null,
    OrderStatusID int not null,
    Amount decimal(10,2) not null,
    constraint FK_Customer_CustomerOrder foreign key (CustomerID) references
Customer(ID),
    constraint FK_OrderStatus_CustomerOrder foreign key (OrderStatusID)
references OrderStatus(ID)
)

create table OrderDetail(
    CustomerOrderID int not null,
    ProductID int not null,
    Quantity int not null,
    Amount decimal(10,2) not null,
    constraint FK_CustomerOrder_OrderDetail foreign key (CustomerOrderID)
references CustomerOrder(ID),
    constraint FK_Product_OrderDetail foreign key (ProductID) references
Product(ID),
    constraint PK_OrderDetail primary key (CustomerOrderID, ProductID)
)

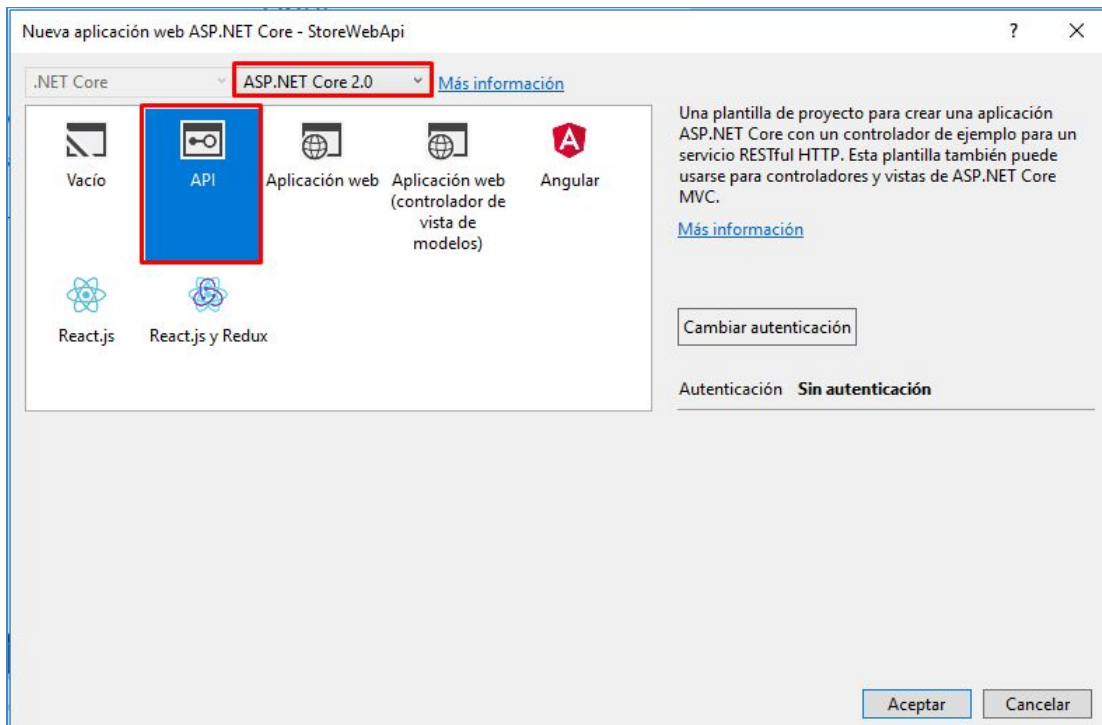
```

2. En Visual Studio y cree un proyecto de tipo **Aplicación web ASP .NET Core** (dentro de la categoría **.NET Core** en el **Visual C#**).

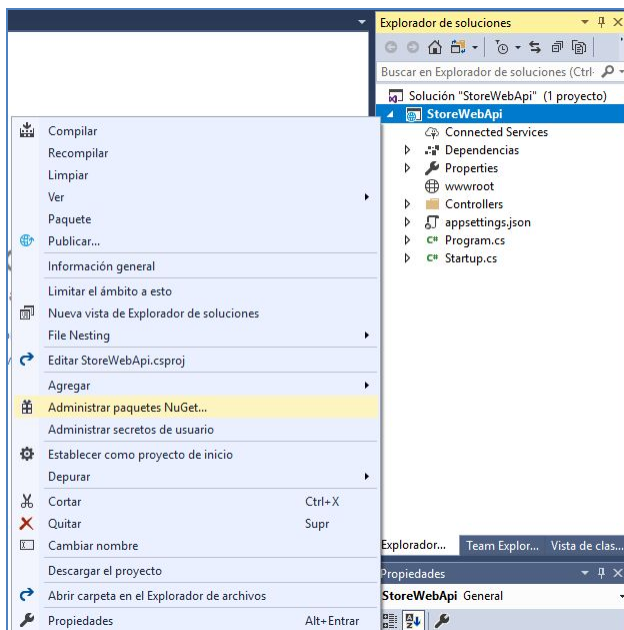


3. Selecciona el tipo de proyecto **API**, este proyecto de ejemplo se puede usar para versión ASP .NET Core 2.0 o superior.

Nota: tener en cuenta que para versión ASP.NET Core 3.0 hay una variación en la forma de agregar el middleware de Entity Framework Core al proyecto, más detalles en este link ver código del método **ConfigureServices** de la clase **startup** <https://www.syncfusion.com/blogs/post/build-crud-application-with-asp-net-core-entity-framework-visual-studio-2019.aspx>.

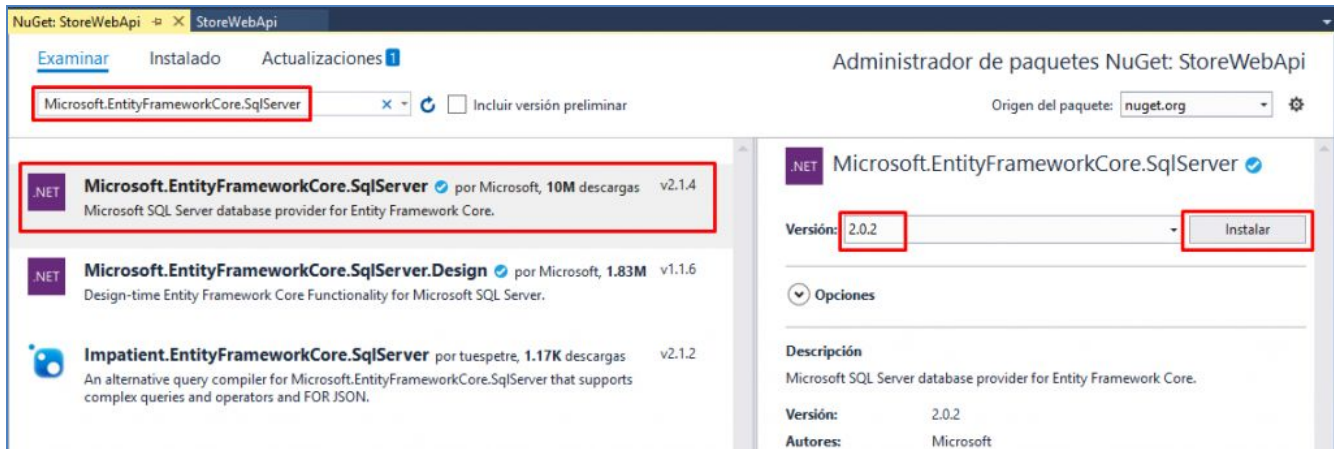


4. Una vez creado el proyecto, click con el botón derecho sobre el nombre del proyecto y selecciona **Administrar paquetes Nuget**. Vamos a agregar paquetes necesarios para usar **Entity Framework Core** con **SQL Server** en el backend.

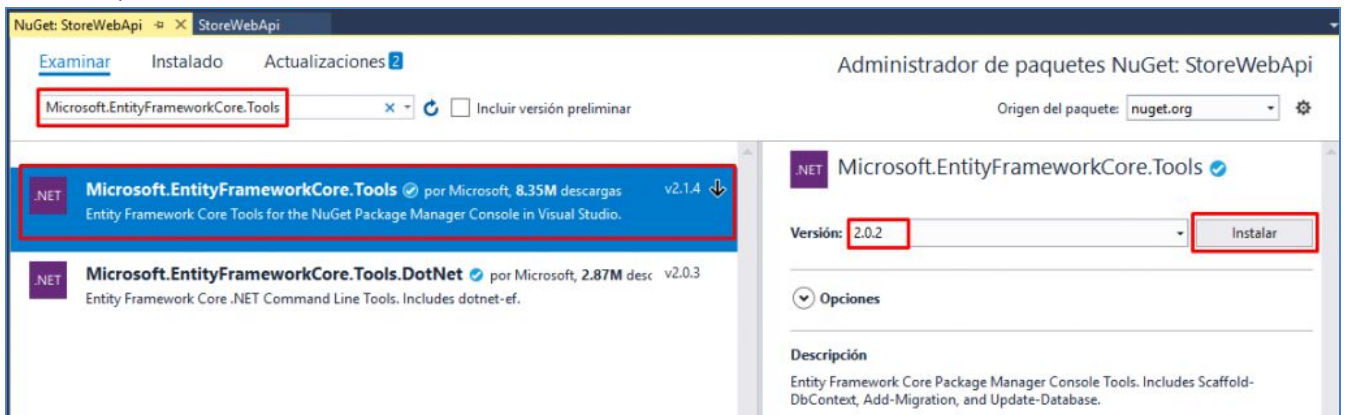


5. El primer paquete a agregar es **Microsoft.EntityFrameworkCore.SqlServer**. Importante: usaremos la versión 2.0.2, si usa una versión superior tenga en

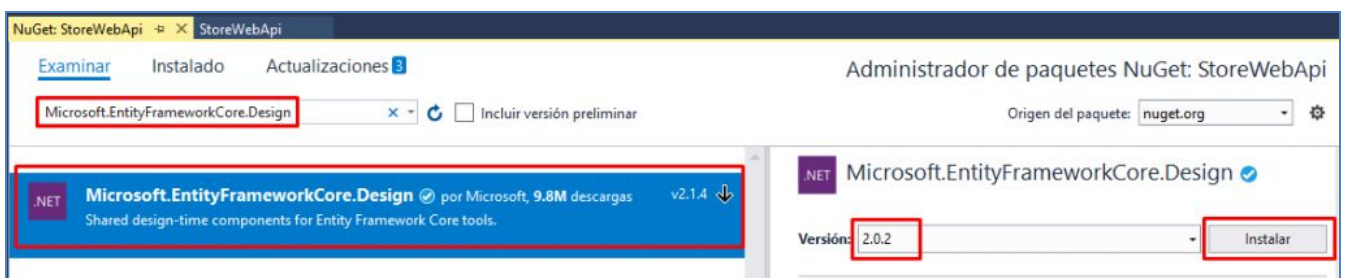
cuenta que todos los paquetes de EF Core deben ser de la misma versión.



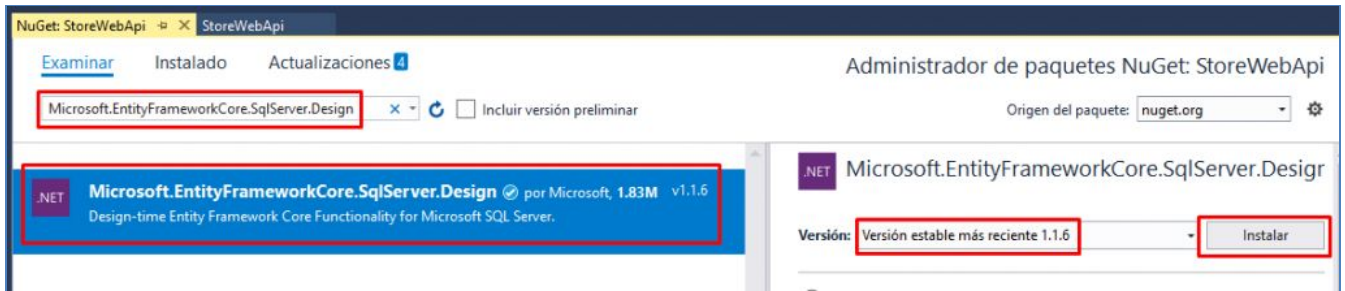
6. El siguiente paquete es **Microsoft.EntityFrameworkCore.Tools** (también versión 2.0.2).



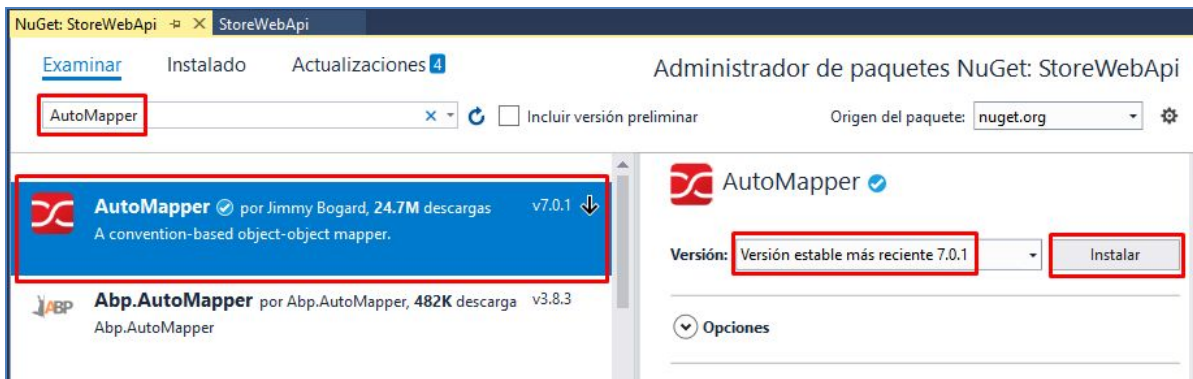
7. El siguiente paquete **Microsoft.EntityFrameworkCore.Design** versión 2.0.2:



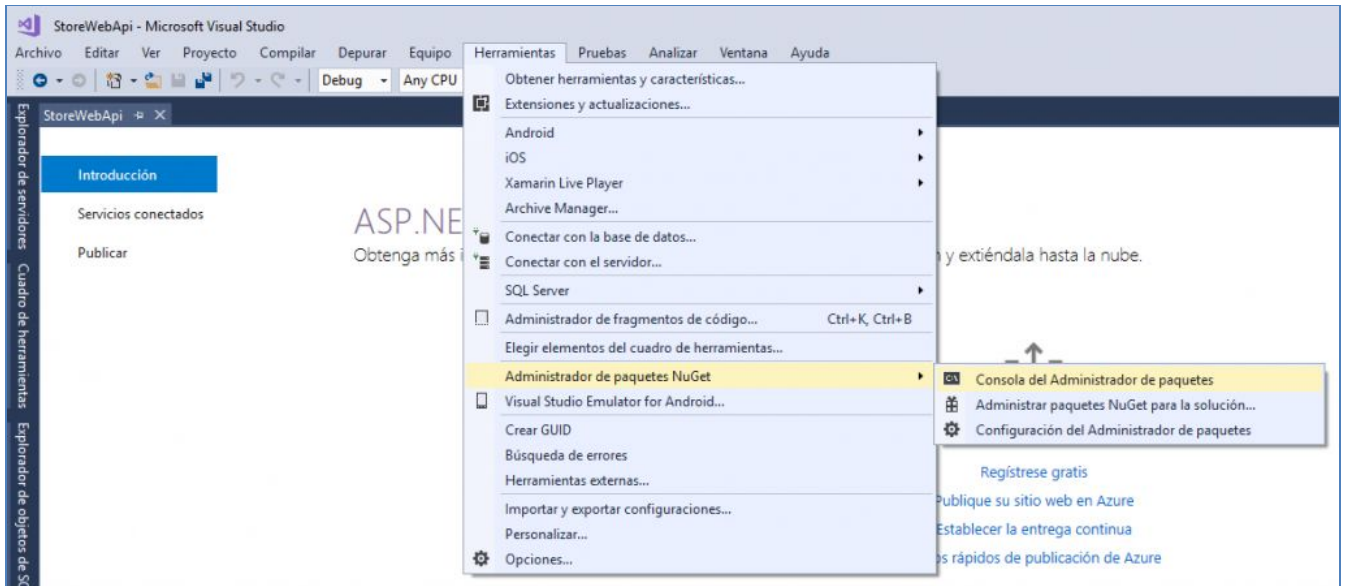
8. Ahora instalamos **Microsoft.EntityFrameworkCore.SqlServer.Design**, la última versión disponible.



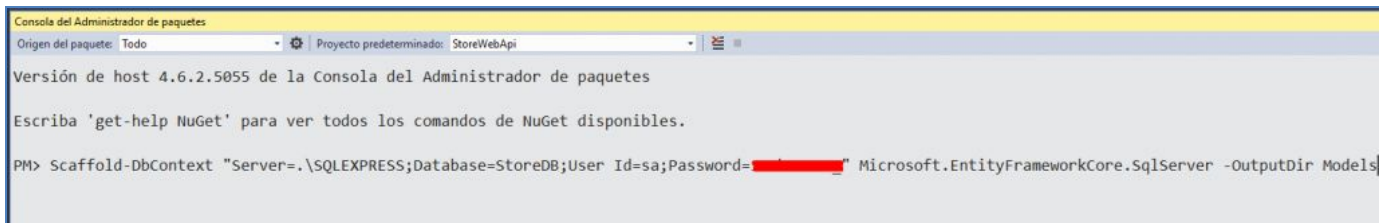
9. Finalmente, instalamos **AutoMapper**.



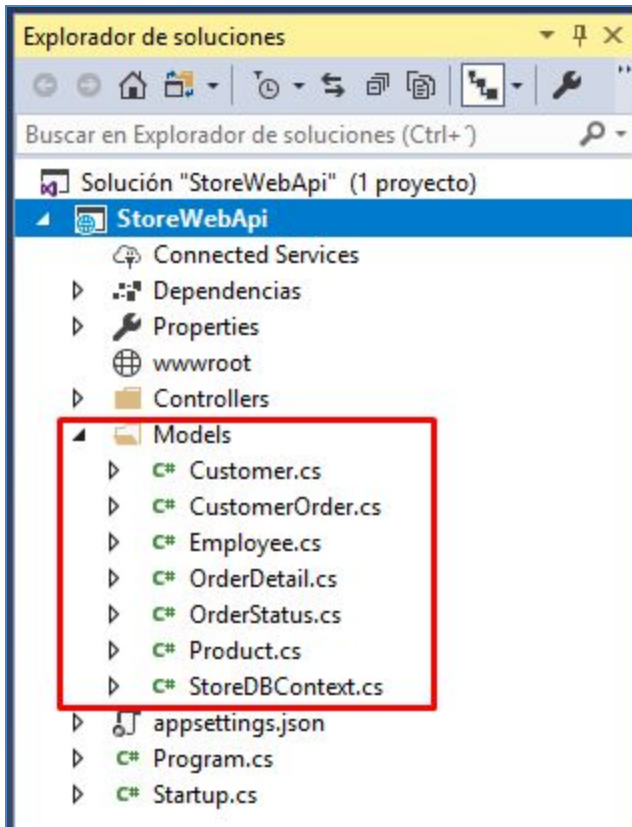
10. El siguiente paso es hacer el **scaffolding** (generación automática y básica de código CRUD) a partir de la base de datos y usando la consola del manejador de paquetes. El resultado serán clases que modelan las tablas incluidas en el almacenamiento. Acceda al menú **Herramientas > Administrador de paquetes Nuget > Consola del Administrador de paquetes**:



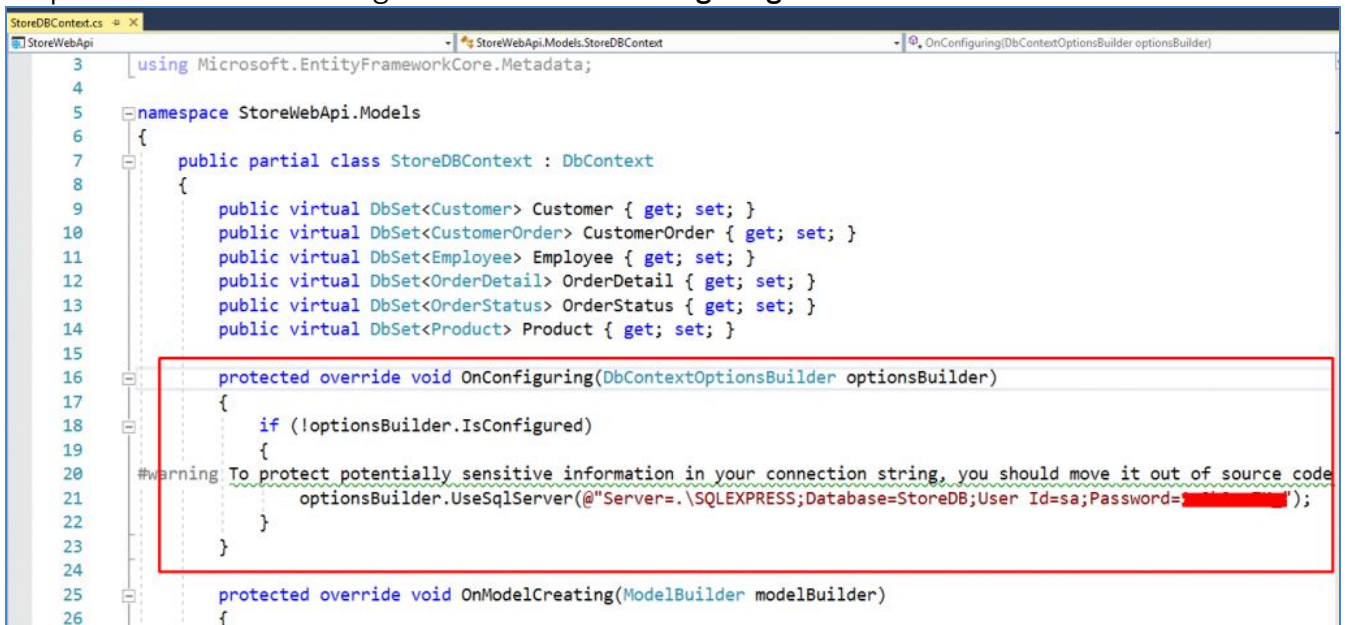
11. A continuación, escriba el siguiente comando en la consola:
Scaffold-DbContext "Server=TuServidor;Database=StoreDB;User ID=TuUsuario;Password=TuContraseña" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models



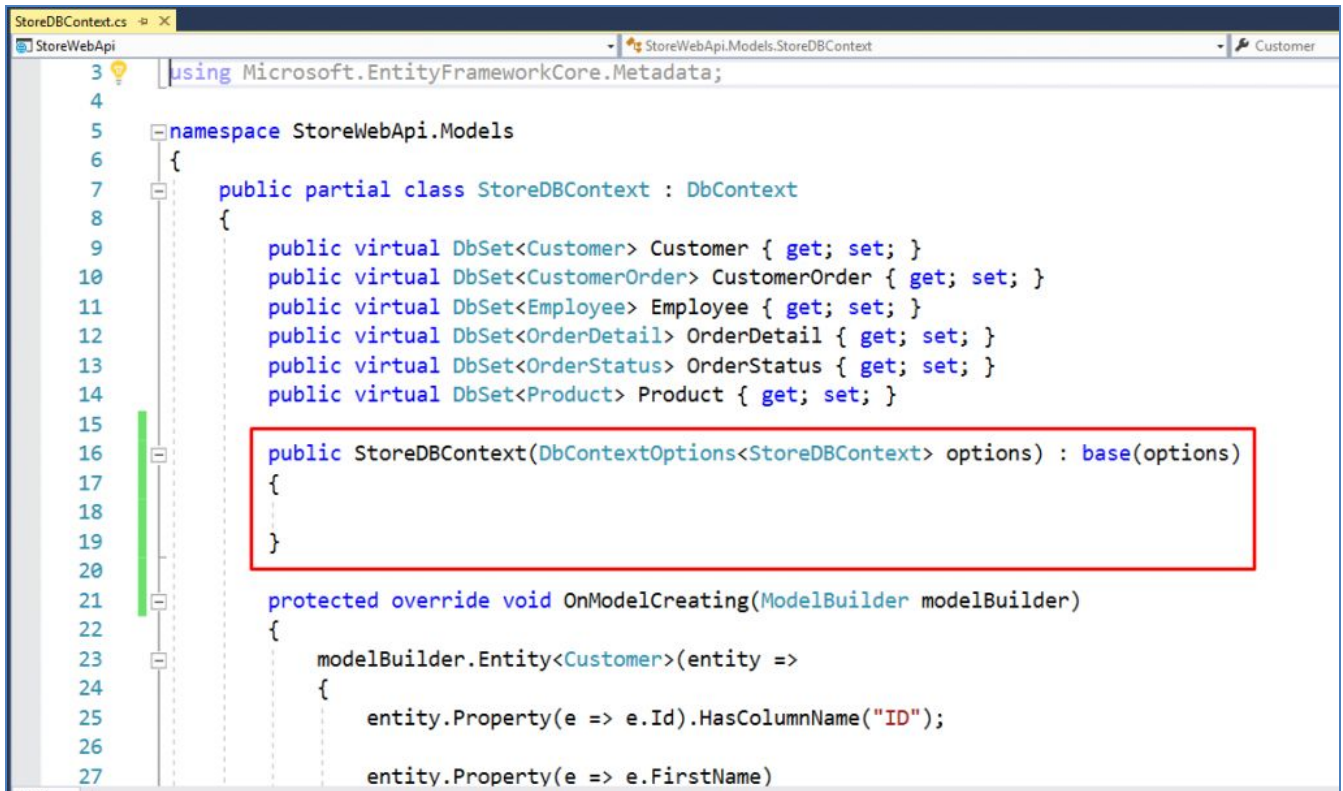
Observe que luego de ejecutar el comando Scaffold-DbContext, las clases que se mapean a las tablas de la base de datos fueron creadas en la carpeta **Models**. Cada clase tiene como propiedades los campos de las tablas correspondientes.



12. Otra de las clases creadas es una clase de contexto, **StoreDbContext**, la cual tiene la cadena de conexión a la base de datos del servidor correspondiente. Sin embargo, éste no es el lugar indicado para dicha cadena, por lo que vamos a reubicarla y utilizar **Dependency Injection** (inyección de dependencias). El primer paso es eliminar el código del método **OnConfiguring**...



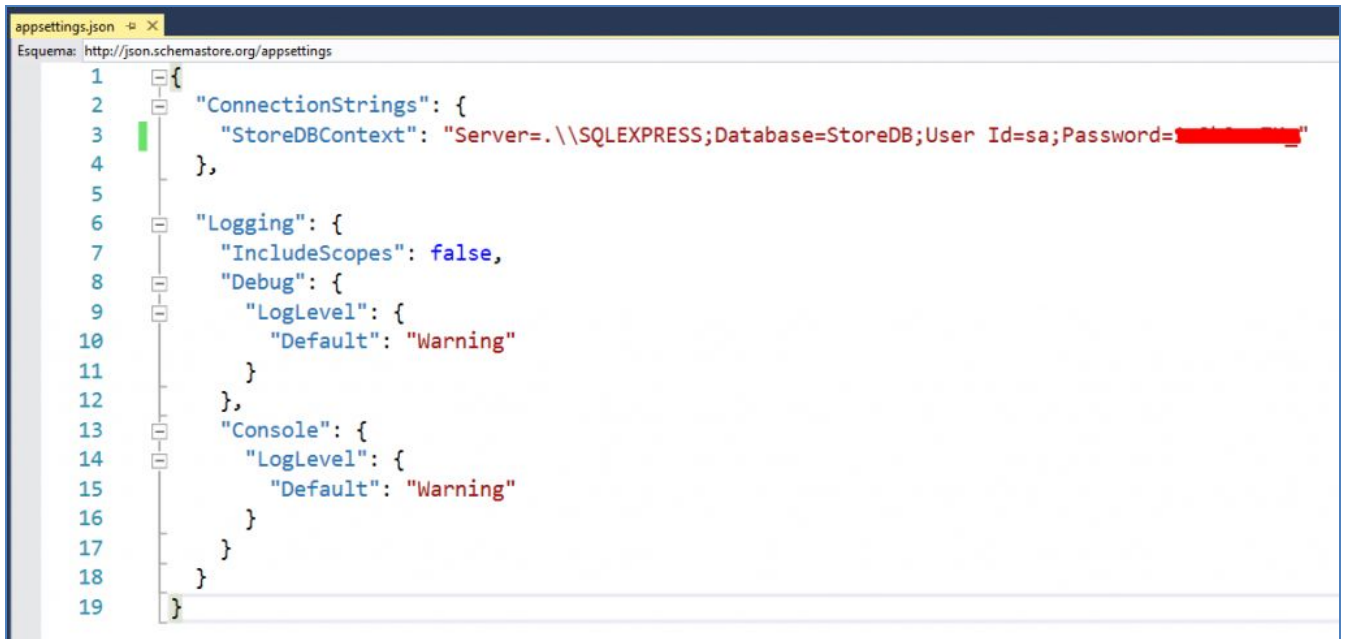
13. ...y reemplazarlo por el siguiente código que es el constructor de la clase **StoreDbContext** que maneja el contexto de Entity Framework, llamando al constructor de la clase base **DbContext** pero pasando como parámetro **DbContextOptions**)



```
3 using Microsoft.EntityFrameworkCore.Metadata;
4
5 namespace StoreWebApi.Models
6 {
7     public partial class StoreDbContext : DbContext
8     {
9         public virtual DbSet<Customer> Customer { get; set; }
10        public virtual DbSet<CustomerOrder> CustomerOrder { get; set; }
11        public virtual DbSet<Employee> Employee { get; set; }
12        public virtual DbSet<OrderDetail> OrderDetail { get; set; }
13        public virtual DbSet<OrderStatus> OrderStatus { get; set; }
14        public virtual DbSet<Product> Product { get; set; }
15
16        public StoreDbContext(DbContextOptions<StoreDbContext> options) : base(options)
17        {
18        }
19
20        protected override void OnModelCreating(ModelBuilder modelBuilder)
21        {
22            modelBuilder.Entity<Customer>(entity =>
23            {
24                entity.Property(e => e.Id).HasColumnName("ID");
25                entity.Property(e => e.FirstName)
```

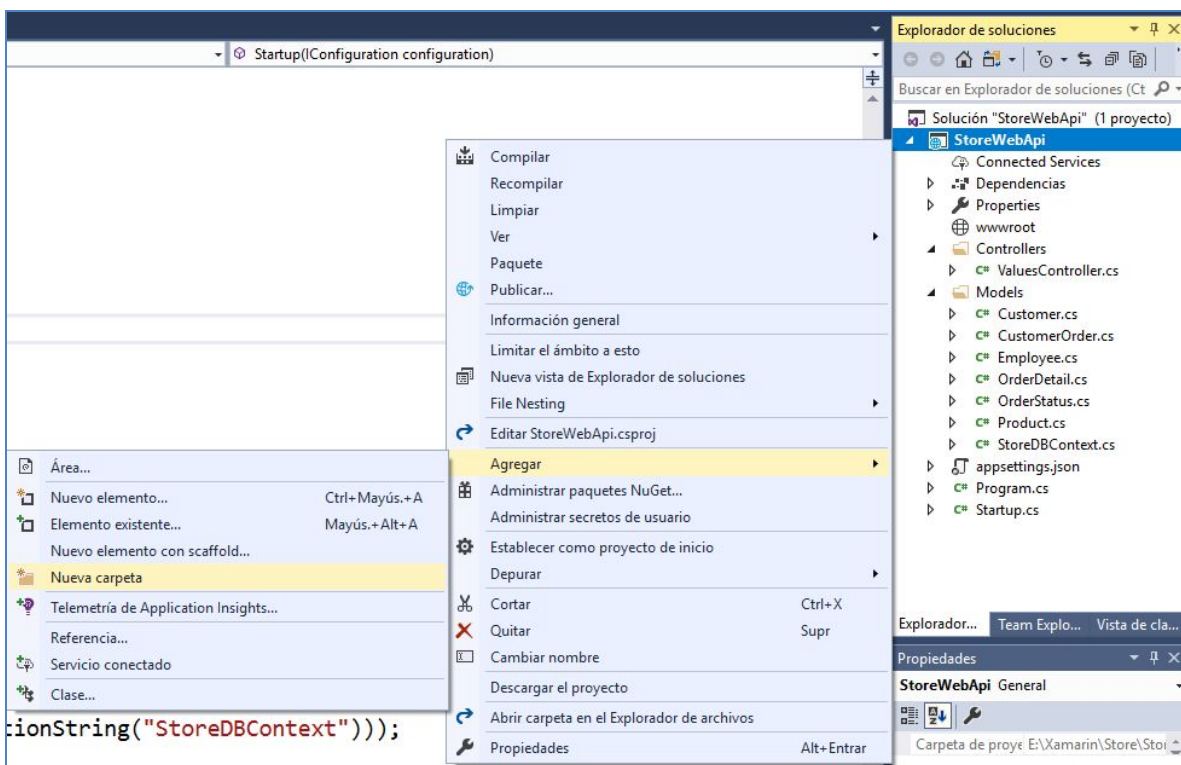
14. Ahora agregamos una **cadena de conexión** en el archivo **appsettings.json**. Recuerde que en este archivo se manejan configuraciones/seteos varios de la aplicación, entre ellos la cadena de conexión con la ubicación y forma de acceso a la base de datos:

```
"ConnectionStrings": {
  "StoreDbContext": "Server=TuServidor;Database=StoreDB;User
ID=TuUsuario;Password=TuContraseña"
},
```

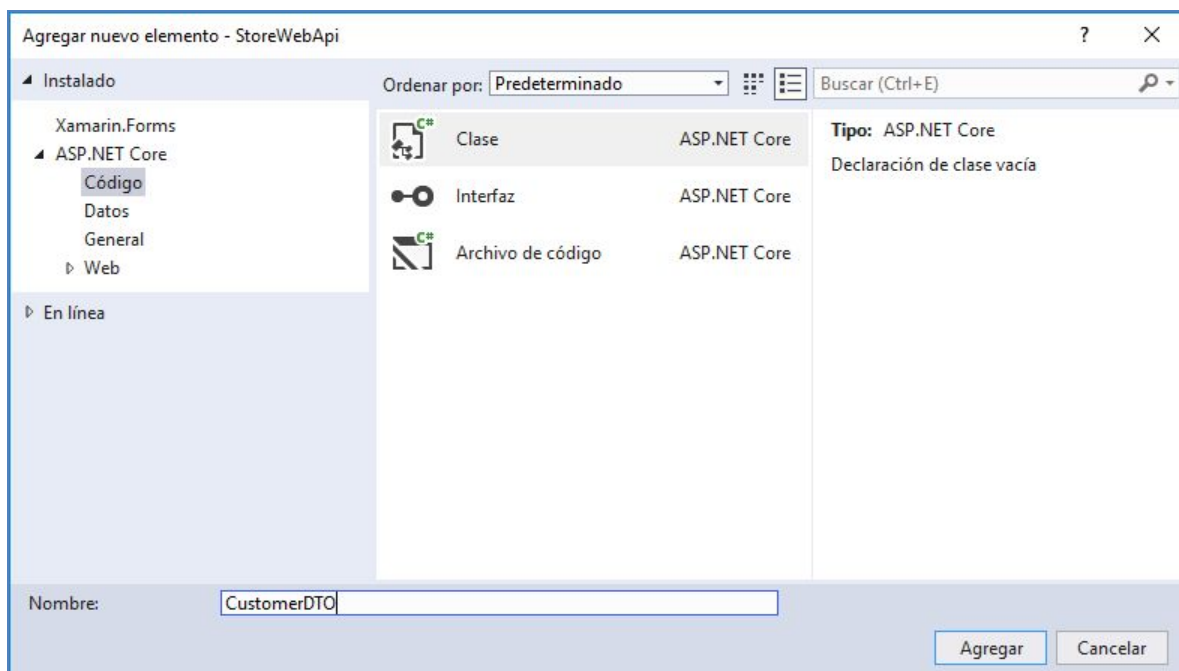
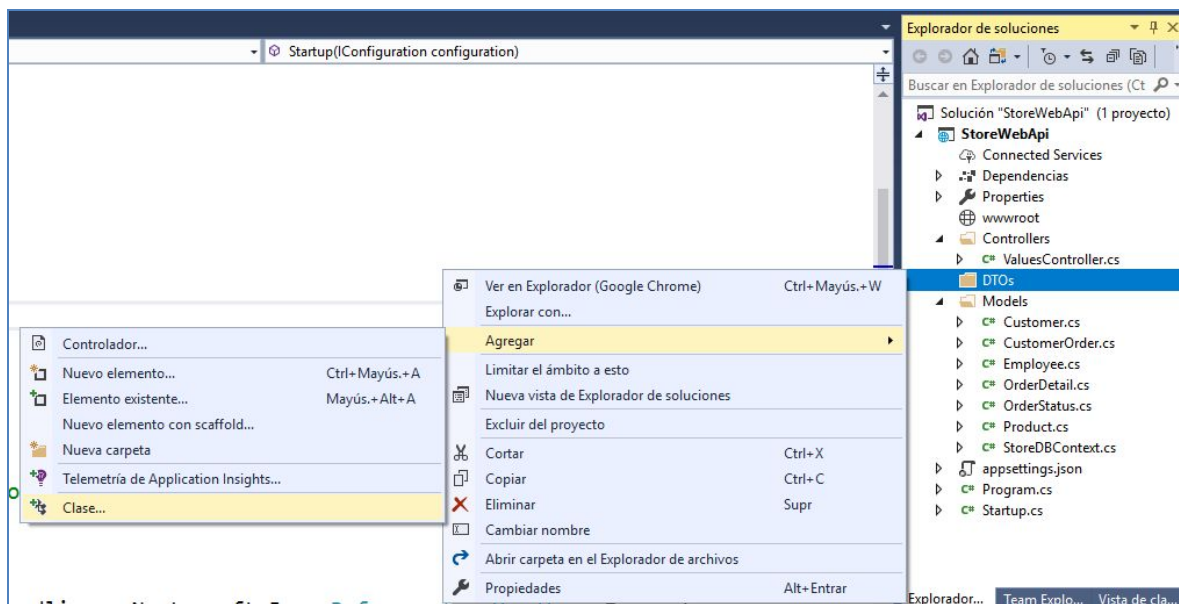


```
1 {
2   "ConnectionStrings": {
3     "StoreDBContext": "Server=.;\\SQLEXPRESS;Database=StoreDB;User Id=sa;Password=XXXXXXXXXX"
4   },
5
6   "Logging": {
7     "IncludeScopes": false,
8     "Debug": {
9       "LogLevel": {
10        "Default": "Warning"
11      }
12    },
13    "Console": {
14      "LogLevel": {
15        "Default": "Warning"
16      }
17    }
18  }
19 }
```

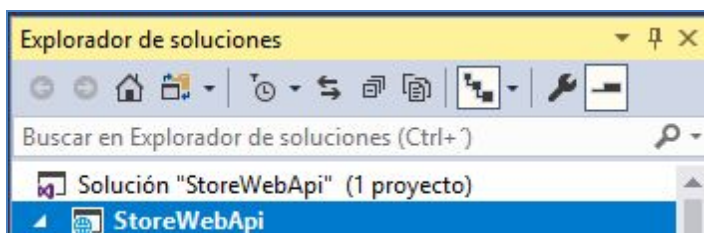
15. Agregamos una nueva carpeta llamada **DTOs**. El detalle conceptual de ventaja y uso de las clases DTO **objetos de transferencia de datos** (*Data Transfer Object*), se estudiará y profundizará en contenidos de la clase 6.



16. En dicha carpeta, agregamos una clase llamada **CustomerDTO** (representando el modelo **Customer**).



17. Repetimos el paso anterior para tener las clases DTOs (**Data Transfer Objects**) del resto de los modelos generados previamente.



18. Cada DTO tendrá la misma estructura (propiedades) que el modelo que representa. Sin embargo, los elementos **ICollection** se convierten en **List** y si una clase utiliza objetos de otro modelo, se reemplaza por su respectivo DTO. A continuación detallamos el código de cada DTO:

- **CustomerDTO.cs:** (CustomerOrder era un objeto ICollection<CustomerOrder> pero ahora se convierte en List<CustomerOrderDTO>))

```
using System;
using System.Collections.Generic;

namespace StoreWebApi.DTOs
{
    public class CustomerDTO
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string UserName { get; set; }
        public string Password { get; set; }

        public List<CustomerOrderDTO> CustomerOrder { get; set; }
    }
}
```

- **CustomerOrderDTO.cs:**

```
using System;
using System.Collections.Generic;

namespace StoreWebApi.DTOs
{
    public class CustomerOrderDTO
    {
        public int Id { get; set; }
        public int CustomerId { get; set; }
        public DateTime Date { get; set; }
        public int OrderStatusId { get; set; }
    }
}
```

```

        public decimal Amount { get; set; }

        public CustomerDTO Customer { get; set; }
        public OrderStatusDTO OrderStatus { get; set; }
        public List<OrderDetailDTO> OrderDetail { get; set; }
    }
}

```

- **EmployeeDTO.cs:**

```

using System;
using System.Collections.Generic;

namespace StoreWebApi.DTOs
{
    public class EmployeeDTO
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string UserName { get; set; }
        public string Password { get; set; }
    }
}

```

- **OrderDetailDTO.cs:**

```

using System;
using System.Collections.Generic;

namespace StoreWebApi.DTOs
{
    public class OrderDetailDTO
    {
        public int CustomerOrderId { get; set; }
        public int ProductId { get; set; }
        public int Quantity { get; set; }
        public decimal Amount { get; set; }

        public CustomerOrderDTO CustomerOrder { get; set; }
        public ProductDTO Product { get; set; }
    }
}

```

- **OrderStatusDTO.cs:**

```

using System;
using System.Collections.Generic;

namespace StoreWebApi.DTOs
{
    public class OrderStatusDTO
    {
        public int Id { get; set; }
    }
}

```



```

        public string Name { get; set; }

        public List<CustomerOrderDTO> CustomerOrder { get; set; }
    }
}

```

- **ProductDTO.cs:**

```

using System;
using System.Collections.Generic;

namespace StoreWebApi.DTOs
{
    public class ProductDTO
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public decimal UnitPrice { get; set; }

        public List<OrderDetailDTO> OrderDetail { get; set; }
    }
}

```

19. A continuación creamos la clase **AutoMapperConfiguration** que mapeará los modelos a los DTOs (Automapper se estudiará y profundizará en contenidos de la clase 6). Además vamos a ignorar las propiedades de navegación a fin de que los objetos JSON generados por los controladores no tengan referencias circulares. Ubique la clase en la misma carpeta DTOs con el siguiente código:

```

using AutoMapper;
using StoreWebApi.Models;

namespace StoreWebApi.DTOs
{
    public static class AutoMapperConfiguration
    {
        public static void Configure()
        {
            Mapper.Initialize(cfg =>
            {
                cfg.CreateMap<Customer, CustomerDTO>()
                    .ForMember(x => x.CustomerOrder, o => o.Ignore())
                    .ReverseMap();

                cfg.CreateMap<CustomerOrder, CustomerOrderDTO>()
                    .ForMember(x => x.OrderDetail, o => o.Ignore())
                    .ReverseMap();

                cfg.CreateMap<Employee, EmployeeDTO>()
                    .ReverseMap();

                cfg.CreateMap<OrderDetail, OrderDetailDTO>()
                    .ReverseMap();
            }
        }
    }
}

```



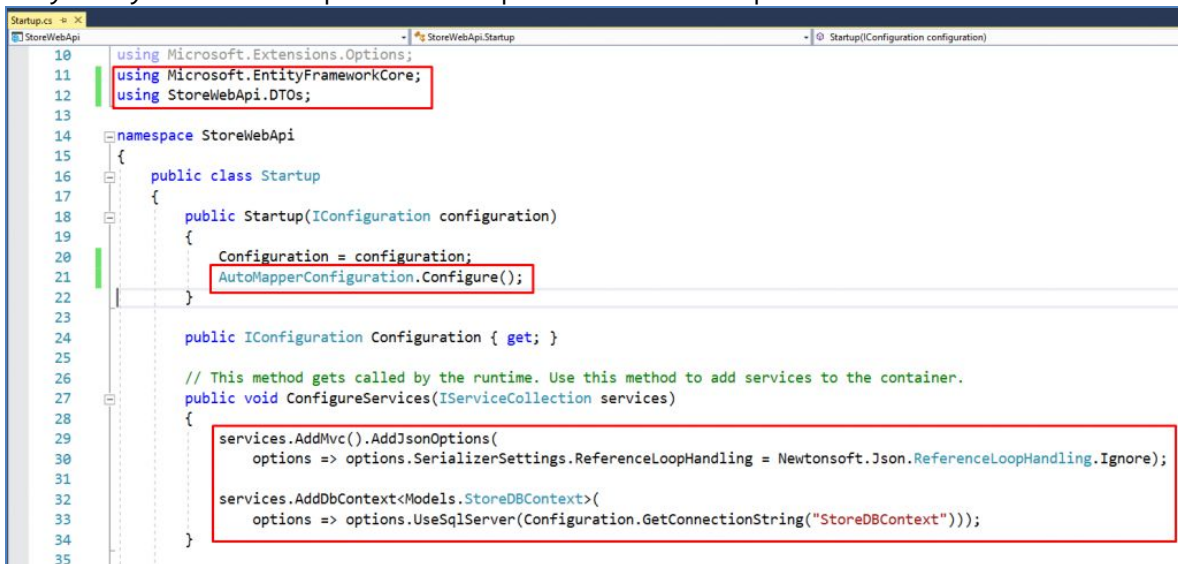
```

        cfg.CreateMap<OrderStatus, OrderStatusDTO>()
            .ForMember(x => x.CustomerOrder, o => o.Ignore())
            .ReverseMap();

        cfg.CreateMap<Product, ProductDTO>()
            .ForMember(x => x.OrderDetail, o => o.Ignore())
            .ReverseMap();
    });
}
}
}

```

20. Modificamos la clase **Startup.cs** para incluir la configuración de **AutoMapper**, la eliminación de referencias circulares por parte de la serialización de objetos Json y la inyección de dependencias que mencionamos para el contexto.



La clase **Startup** queda con el siguiente código:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using Microsoft.EntityFrameworkCore;
using StoreWebApi.DTOS;

namespace StoreWebApi
{

```

```

public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
        AutoMapperConfiguration.Configure();
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add
    services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc().AddJsonOptions(
            options => options.SerializerSettings.ReferenceLoopHandling =
Newtonsoft.Json.ReferenceLoopHandling.Ignore);

        services.AddDbContext<Models.StoreDbContext>(
            options =>
options.UseSqlServer(Configuration.GetConnectionString("StoreDbContext")));

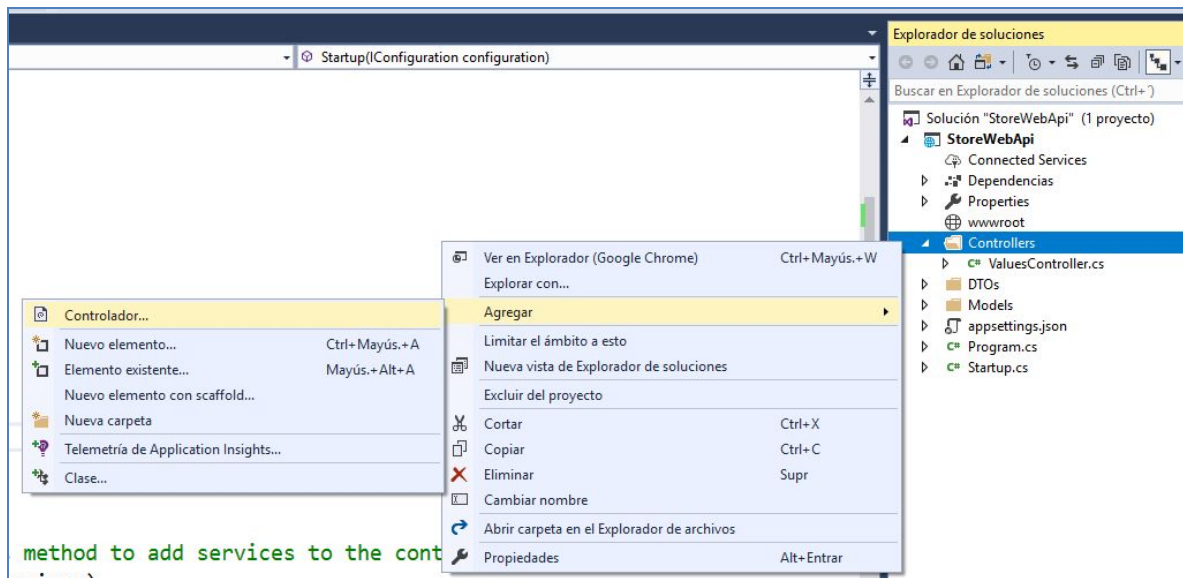
        //5
        services.AddSignalR(options => options.EnableDetailedErrors = true);
    }

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

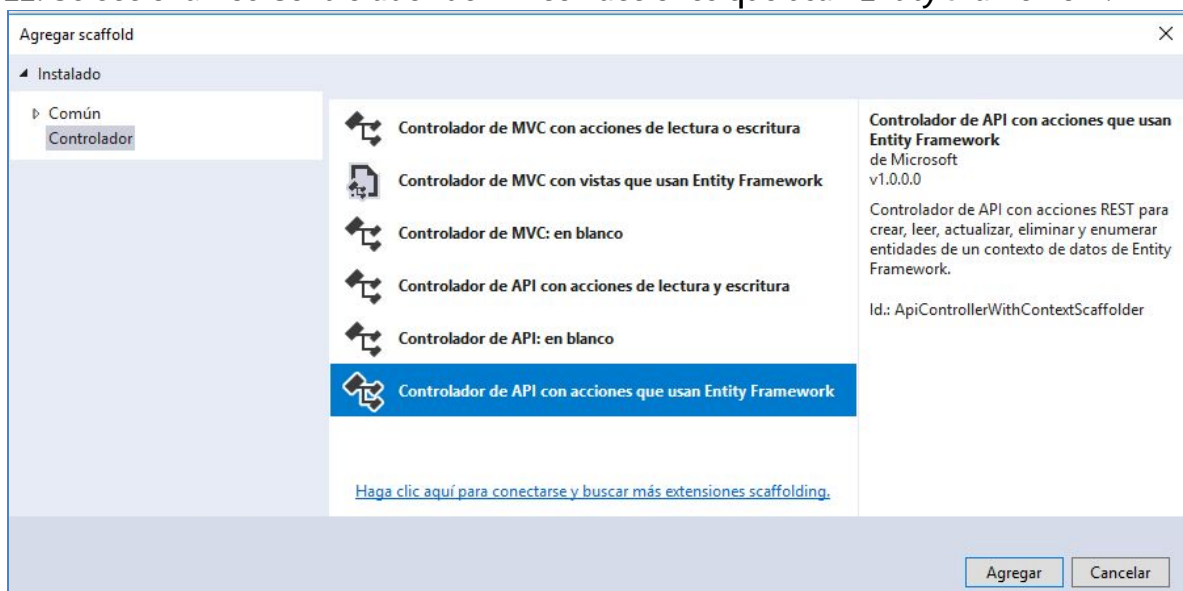
        app.UseMvc();
    }
}

```

21. Ahora vamos a agregar los controladores, que son los métodos expuestos por el Web Api para realizar operaciones CRUD sobre las tablas de la base de datos. Sobre la carpeta **Controllers**, botón derecho y selecciona **Agregar > Controlador**:



22. Seleccionamos **Controlador de API con acciones que usan Entity Framework**.



23. En clase de modelo seleccionamos **Customer** (de la carpeta **Models**), luego en clase de contexto seleccionamos **StoreDBContext**. El nombre del controlador será **CustomersController**.

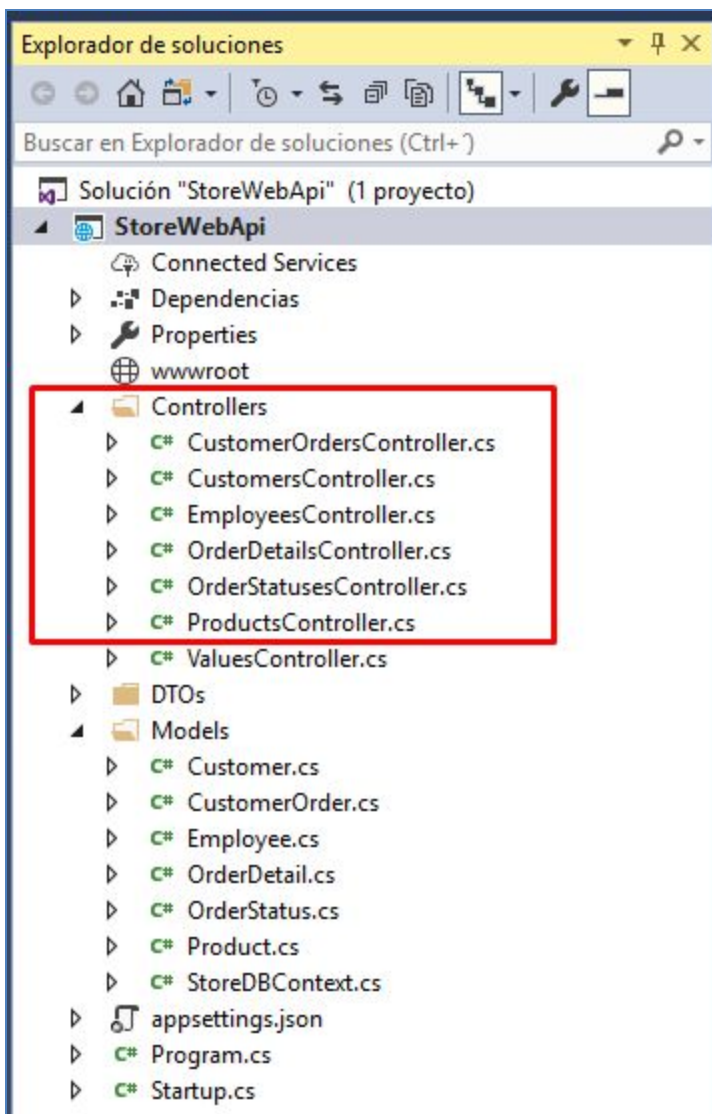
Agregar Controlador de API con acciones que usan Entity Framework

Clase de modelo: Customer (StoreWebApi.Models)

Clase de contexto de datos: StoreDbContext (StoreWebApi.Models) +

Nombre de controlador: CustomersController

Agregar Cancelar



24. Repetimos este proceso para generar los controladores del resto de los modelos:

Recordamos que cada controlador tiene métodos para acceder a la información (**Get**), insertar datos (**Post**), modificar datos (**Put**) y eliminar datos (**Delete**). Vamos a modificar cada controlador para que funcione con los DTOs, así como agregar y modificar algunos métodos. A continuación se muestra el código final de cada controlador:

- CustomerOrdersController.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using StoreWebApi.Models;
using AutoMapper;
using StoreWebApi.DTOs;

namespace StoreWebApi.Controllers
{
    [Produces("application/json")]
    [Route("api/CustomerOrders")]
    public class CustomerOrdersController : Controller
    {
        private readonly StoreDbContext _context;

        public CustomerOrdersController(StoreDbContext context)
        {
            _context = context;
        }

        // GET: api/CustomerOrders
        [HttpGet]
        public IEnumerable<CustomerOrderDTO> GetCustomerOrder()
        {
            return
                Mapper.Map<IEnumerable<CustomerOrderDTO>>(_context.CustomerOrder.OrderByDescending(x => x.Date));
        }

        // GET: api/CustomerOrders/Customer/5
        [HttpGet("Customer/{customerId}")]
        public IEnumerable<CustomerOrderDTO>
            GetCustomer_CustomerOrder([FromRoute] int customerId)
        {
            return
                Mapper.Map<IEnumerable<CustomerOrderDTO>>(_context.CustomerOrder
                    .Include(x => x.OrderStatus)
                    .Where(x => x.CustomerId == customerId)
                    .OrderByDescending(x => x.Date));
        }

        // GET: api/CustomerOrders/5
        [HttpGet("{id}")]
        public async Task<IActionResult> GetCustomerOrder([FromRoute] int id)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }
        }
    }
}

```

```

        var customerOrder = await
_context.CustomerOrder.SingleOrDefaultAsync(m => m.Id == id);

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

        return Ok(Mapper.Map<CustomerOrderDTO>(customerOrder));
    }

    // PUT: api/CustomerOrders/5
    [HttpPut("{id}")]
    public async Task<IActionResult> PutCustomerOrder([FromRoute] int id,
[FromBody] CustomerOrderDTO customerOrder)
    {
        customerOrder.OrderDetail = null;
        customerOrder.OrderStatus = null;
        customerOrder.Customer = null;

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

        if (id != customerOrder.Id)
        {
            return BadRequest();
        }

        _context.Entry(Mapper.Map<CustomerOrder>(customerOrder)).State =
EntityState.Modified;

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!CustomerOrderExists(id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }

        return NoContent();
    }

    // POST: api/CustomerOrders
    [HttpPost]

```

```

        public async Task<IActionResult> PostCustomerOrder([FromBody]
CustomerOrderDTO customerOrder)
        {
            customerOrder.OrderDetail = null;
            customerOrder.OrderStatus = null;
            customerOrder.Customer = null;

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

            var co = Mapper.Map<CustomerOrder>(customerOrder);
            _context.CustomerOrder.Add(co);
            await _context.SaveChangesAsync();
            customerOrder.Id = co.Id;

            return CreatedAtAction("GetCustomerOrder", new { id = co.Id },
customerOrder);
        }

        // DELETE: api/CustomerOrders/5
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteCustomerOrder([FromRoute] int id)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var customerOrder = await
_context.CustomerOrder.SingleOrDefaultAsync(m => m.Id == id);
            if (customerOrder == null)
            {
                return NotFound();
            }

            _context.CustomerOrder.Remove(customerOrder);
            await _context.SaveChangesAsync();

            return Ok(Mapper.Map<CustomerOrderDTO>(customerOrder));
        }

        private bool CustomerOrderExists(int id)
        {
            return _context.CustomerOrder.Any(e => e.Id == id);
        }
    }
}

```

- **CustomersController.cs:**

```

using System;
using System.Collections.Generic;

```



```

using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using StoreWebApi.Models;
using AutoMapper;
using StoreWebApi.DTOs;
using Newtonsoft.Json.Linq;
using Newtonsoft.Json;

namespace StoreWebApi.Controllers
{
    [Produces("application/json")]
    [Route("api/Customers")]
    public class CustomersController : Controller
    {
        private readonly StoreDbContext _context;

        public CustomersController(StoreDbContext context)
        {
            _context = context;
        }

        // GET: api/Customers
        [HttpGet]
        public IEnumerable<CustomerDTO> GetCustomer()
        {
            return Mapper.Map<IEnumerable<CustomerDTO>>(_context.Customer.OrderBy(x
            => x.LastName).ThenBy(x => x.FirstName));
        }

        // GET: api/Customers/5
        [HttpGet("{id}")]
        public async Task<IActionResult> GetCustomer([FromRoute] int id)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var customer = await _context.Customer.SingleOrDefaultAsync(m => m.Id ==
            id);

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

            return Ok(Mapper.Map<CustomerDTO>(customer));
        }

        [HttpPost("Login")]
        public async Task<IActionResult> Login([FromBody] dynamic credentials)

```

```

{
var username = (string)credentials["username"];
var password = (string)credentials["password"];

var customer = await _context.Customer.SingleOrDefaultAsync(m =>
m.UserName == username && m.Password == password);

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

return Ok(Mapper.Map<CustomerDTO>(customer));
}

// PUT: api/Customers/5
[HttpPut("{id}")]
public async Task<IActionResult> PutCustomer([FromRoute] int id,
[FromBody] CustomerDTO customer)
{
customer.CustomerOrder = null;

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

if (id != customer.Id)
{
return BadRequest();
}

_context.Entry(Mapper.Map<Customer>(customer)).State =
EntityState.Modified;

try
{
await _context.SaveChangesAsync();
}
catch (DbUpdateConcurrencyException)
{
if (!CustomerExists(id))
{
return NotFound();
}
else
{
throw;
}
}

return NoContent();
}

```

```

// POST: api/Customers
[HttpPost]
public async Task<IActionResult> PostCustomer([FromBody] CustomerDTO
customer)
{
    customer.CustomerOrder = null;

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

    var c = Mapper.Map<Customer>(customer);
    _context.Customer.Add(c);
    await _context.SaveChangesAsync();
    customer.Id = c.Id;

    return CreatedAtAction("GetCustomer", new { id = c.Id }, customer);
}

// DELETE: api/Customers/5
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteCustomer([FromRoute] int id)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var customer = await _context.Customer.SingleOrDefaultAsync(m => m.Id ==
id);
    if (customer == null)
    {
        return NotFound();
    }

    _context.Customer.Remove(customer);
    await _context.SaveChangesAsync();

    return Ok(Mapper.Map<CustomerDTO>(customer));
}

private bool CustomerExists(int id)
{
    return _context.Customer.Any(e => e.Id == id);
}
}

```

- **EmployeesController.cs:**

```

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

```

```

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using StoreWebApi.Models;
using AutoMapper;
using StoreWebApi.DTOs;
using Newtonsoft.Json;

namespace StoreWebApi.Controllers
{
    [Produces("application/json")]
    [Route("api/Employees")]
    public class EmployeesController : Controller
    {
        private readonly StoreDBContext _context;

        public EmployeesController(StoreDBContext context)
        {
            _context = context;
        }

        // GET: api/Employees
        [HttpGet]
        public IEnumerable<EmployeeDTO> GetEmployee()
        {
            return
                Mapper.Map<IEnumerable<EmployeeDTO>>(_context.Employee.OrderBy(x =>
                    x.LastName).ThenBy(x => x.FirstName));
        }

        // GET: api/Employees/5
        [HttpGet("{id}")]
        public async Task<IActionResult> GetEmployee([FromRoute] int id)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var employee = await _context.Employee.SingleOrDefaultAsync(m =>
                m.Id == id);

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

            return Ok(Mapper.Map<EmployeeDTO>(employee));
        }

        [HttpPost("Login")]
        public async Task<IActionResult> Login([FromBody] dynamic credentials)
        {
            var username = (string)credentials["username"];

```

```

        var password = (string)credentials["password"];

        var employee = await _context.Employee.SingleOrDefaultAsync(m =>
m.UserName == username && m.Password == password);

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

        return Ok(Mapper.Map<EmployeeDTO>(employee));
    }

    // PUT: api/Employees/5
    [HttpPut("{id}")]
    public async Task<IActionResult> PutEmployee([FromRoute] int id,
[FromBody] EmployeeDTO employee)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        if (id != employee.Id)
        {
            return BadRequest();
        }

        _context.Entry(Mapper.Map<Employee>(employee)).State =
EntityState.Modified;

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!EmployeeExists(id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }

        return NoContent();
    }

    // POST: api/Employees
    [HttpPost]
    public async Task<IActionResult> PostEmployee([FromBody] EmployeeDTO
employee)

```

```

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

    var e = Mapper.Map<Employee>(employee);
    _context.Employee.Add(e);
    await _context.SaveChangesAsync();
    employee.Id = e.Id;

    return CreatedAtAction("GetEmployee", new { id = e.Id }, employee);
}

// DELETE: api/Employees/5
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteEmployee([FromRoute] int id)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var employee = await _context.Employee.SingleOrDefaultAsync(m =>
m.Id == id);
    if (employee == null)
    {
        return NotFound();
    }

    _context.Employee.Remove(employee);
    await _context.SaveChangesAsync();

    return Ok(Mapper.Map<EmployeeDTO>(employee));
}

private bool EmployeeExists(int id)
{
    return _context.Employee.Any(e => e.Id == id);
}
}
}

```

- **OrderDetailsController.cs:**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using StoreWebApi.Models;
using AutoMapper;

```

```

using StoreWebApi.DTOs;

namespace StoreWebApi.Controllers
{
    [Produces("application/json")]
    [Route("api/OrderDetails")]
    public class OrderDetailsController : Controller
    {
        private readonly StoreDBContext _context;

        public OrderDetailsController(StoreDBContext context)
        {
            _context = context;
        }

        // GET: api/OrderDetails
        [HttpGet]
        public IEnumerable<OrderDetail> GetOrderDetail()
        {
            return _context.OrderDetail;
        }

        // GET: api/OrderDetails/Order/5
        [HttpGet("Order/{orderId}")]
        public async Task<IActionResult> GetOrder_OrderDetail([FromRoute] int
            orderId)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var orderDetail = _context.OrderDetail
                .Include(x => x.Product)
                .Where(m => m.CustomerOrderId == orderId);

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

            return Ok(Mapper.Map<IEnumerable<OrderDetailDTO>>(orderDetail));
        }

        // POST: api/OrderDetails
        [HttpPost]
        public async Task<IActionResult> PostOrderDetail([FromBody]
            List<OrderDetailDTO> orderDetail)
        {
            foreach (var item in orderDetail)
            {
                item.CustomerOrder = null;
                item.Product = null;
            }
        }
    }
}

```



```

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

        var od = Mapper.Map<IEnumerable<OrderDetail>>(orderDetail);
        _context.OrderDetail.AddRange(od);
        await _context.SaveChangesAsync();

        return CreatedAtAction("GetOrderDetail", new { id =
            od.First().CustomerOrderId }, orderDetail);
    }

    // DELETE: api/OrderDetails/5
    [HttpDelete("{orderId}")]
    public async Task<IActionResult> DeleteOrderDetail([FromRoute] int
        orderId)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        var orderDetail = _context.OrderDetail.Where(m => m.CustomerOrderId ==
            orderId);
        if (orderDetail == null)
        {
            return NotFound();
        }

        _context.OrderDetail.RemoveRange(orderDetail);
        await _context.SaveChangesAsync();

        return Ok(orderDetail);
    }

    private bool OrderDetailExists(int id)
    {
        return _context.OrderDetail.Any(e => e.CustomerOrderId == id);
    }
}

```

- **OrderStatusesController.cs:**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using StoreWebApi.Models;
using AutoMapper;

```

```

using StoreWebApi.DTOs;

namespace StoreWebApi.Controllers
{
    [Produces("application/json")]
    [Route("api/OrderStatuses")]
    public class OrderStatusesController : Controller
    {
        private readonly StoreDBContext _context;

        public OrderStatusesController(StoreDBContext context)
        {
            _context = context;
        }

        // GET: api/OrderStatuses
        [HttpGet]
        public IEnumerable<OrderStatusDTO> GetOrderStatus()
        {
            return
                Mapper.Map<IEnumerable<OrderStatusDTO>>(_context.OrderStatus.OrderBy(x =>
                    x.Name));
        }

        // GET: api/OrderStatuses/5
        [HttpGet("{id}")]
        public async Task<IActionResult> GetOrderStatus([FromRoute] int id)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var orderStatus = await _context.OrderStatus.SingleOrDefaultAsync(m
                => m.Id == id);

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

            return Ok(Mapper.Map<OrderStatusDTO>(orderStatus));
        }

        // PUT: api/OrderStatuses/5
        [HttpPut("{id}")]
        public async Task<IActionResult> PutOrderStatus([FromRoute] int id,
            [FromBody] OrderStatusDTO orderStatus)
        {
            orderStatus.CustomerOrder = null;

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

```

```

    }

    if (id != orderStatus.Id)
    {
        return BadRequest();
    }

    _context.Entry(Mapper.Map<OrderStatus>(orderStatus)).State =
EntityState.Modified;

    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!OrderStatusExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return NoContent();
}

// POST: api/OrderStatuses
[HttpPost]
public async Task<IActionResult> PostOrderStatus([FromBody]
OrderStatusDTO orderStatus)
{
    orderStatus.CustomerOrder = null;

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

    var os = Mapper.Map<OrderStatus>(orderStatus);

    _context.OrderStatus.Add(os);
    await _context.SaveChangesAsync();
    orderStatus.Id = os.Id;

    return CreatedAtAction("GetOrderStatus", new { id = os.Id },
orderStatus);
}

// DELETE: api/OrderStatuses/5
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteOrderStatus([FromRoute] int id)

```

```

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

        var orderStatus = await _context.OrderStatus.SingleOrDefaultAsync(m
=> m.Id == id);
        if (orderStatus == null)
        {
            return NotFound();
        }

        _context.OrderStatus.Remove(orderStatus);
        await _context.SaveChangesAsync();

        return Ok(Mapper.Map<OrderStatusDTO>(orderStatus));
    }

    private bool OrderStatusExists(int id)
    {
        return _context.OrderStatus.Any(e => e.Id == id);
    }
}

```

- **ProductsController.cs:**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using StoreWebApi.Models;
using AutoMapper;
using StoreWebApi.DTOs;

namespace StoreWebApi.Controllers
{
    [Produces("application/json")]
    [Route("api/Products")]
    public class ProductsController : Controller
    {
        private readonly StoreDBContext _context;

        public ProductsController(StoreDBContext context)
        {
            _context = context;
        }

        // GET: api/Products
        [HttpGet]
        public IEnumerable<ProductDTO> GetProduct()

```

```

    {
        return
Mapper.Map<IEnumerable<ProductDTO>>(_context.Product.OrderBy(x => x.Name));
    }

    // GET: api/Products/5
    [HttpGet("{id}")]
    public async Task<IActionResult> GetProduct([FromRoute] int id)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        var product = await _context.Product.SingleOrDefaultAsync(m => m.Id
== id);

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

        return Ok(Mapper.Map<ProductDTO>(product));
    }

    // PUT: api/Products/5
    [HttpPut("{id}")]
    public async Task<IActionResult> PutProduct([FromRoute] int id,
[FromBody] ProductDTO product)
    {
        product.OrderDetail = null;

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

        if (id != product.Id)
        {
            return BadRequest();
        }

        _context.Entry(Mapper.Map<Product>(product)).State =
EntityState.Modified;

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!ProductExists(id))
            {
                return NotFound();
            }
        }
    }

```

```

        }
        else
        {
            throw;
        }
    }

    return NoContent();
}

// POST: api/Products
[HttpPost]
public async Task<IActionResult> PostProduct([FromBody] ProductDTO
product)
{
    product.OrderDetail = null;

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

    var p = Mapper.Map<Product>(product);

    _context.Product.Add(p);
    await _context.SaveChangesAsync();
    product.Id = p.Id;

    return CreatedAtAction("GetProduct", new { id = p.Id }, product);
}

// DELETE: api/Products/5
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteProduct([FromRoute] int id)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var product = await _context.Product.SingleOrDefaultAsync(m => m.Id
== id);
    if (product == null)
    {
        return NotFound();
    }

    _context.Product.Remove(product);
    await _context.SaveChangesAsync();

    return Ok(Mapper.Map<ProductDTO>(product));
}

private bool ProductExists(int id)

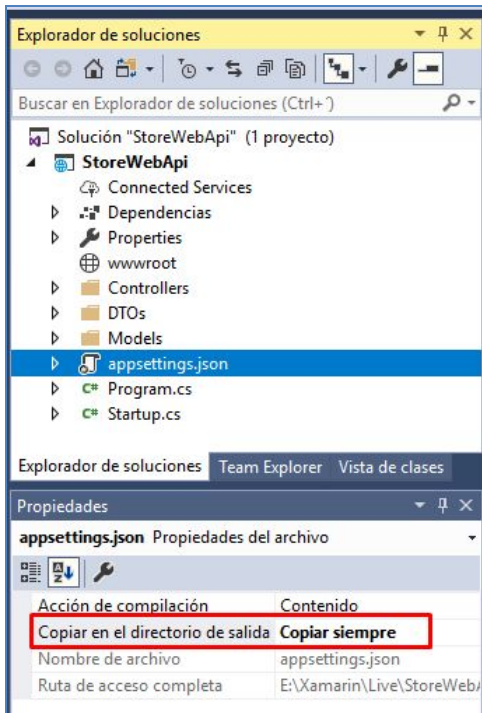
```

```

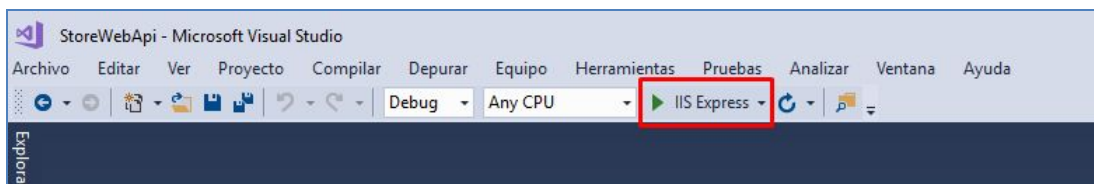
    {
        return _context.Product.Any(e => e.Id == id);
    }
}

```

25. En el explorador de soluciones selecciona el archivo **appsettings.json** y en su propiedad **Copiar en el directorio de salida** elige **Copiar siempre**.



26. A continuación vamos a probar el web api. Hacemos clic en el botón **IIS Express** para compilar y desplegar la aplicación, si tiene otro navegador como Chrome o Firefox, también puede usarlo:



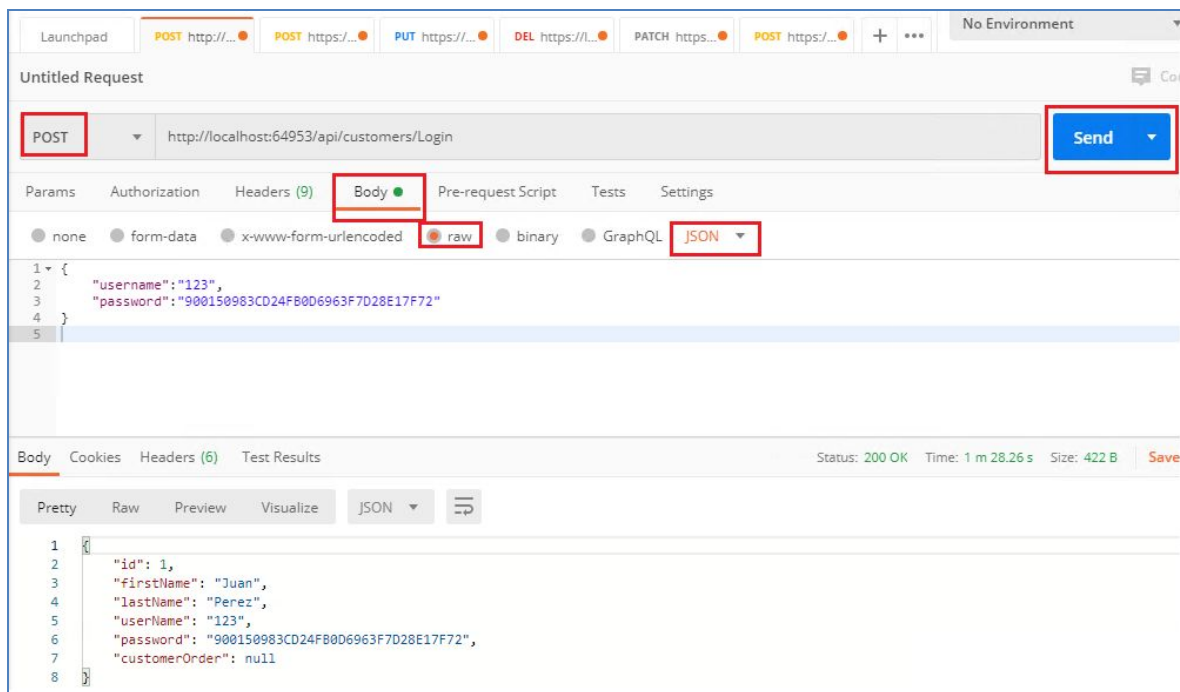
27. Cuando el web api esté en ejecución, vamos a **Postman** y hacemos la prueba del método **Login** (que pertenece al controlador Customers)
- o Seleccionamos **POST**

- o Revisamos el puerto asignado cuando se lanzó la página web para conocer el puerto de la aplicación (también se puede revisar en las propiedades del proyecto).
- o En la Url escribimos **http://localhost:puerto/api/customers/Login**
- o Elegimos **Body**
- o El tipo **raw** e indicamos los datos que recibirá el método Login del controlador **Customers** del web api (este valor corresponde al registro de *Juan Perez* de la base de datos)

```
{
  "username": "123",
  "password": "900150983CD24FB0D6963F7D28E17F72"
}
```

- o El formato **JSON (application/json)**

Al hacer clic en el botón **Send**, observamos que la consulta se realiza correctamente (200 OK) y el registro específico es retornado:



Este es el registro almacenado en la base de datos que la acción **POST** está recuperando desde el método **Login**:

SQLQuery1.sql - BA...E18A\Student (55)) * X

`select * from customer`

120 %

Results Messages

	ID	FirstName	LastName	UserName	Password
1	1	Juan	Perez	123	900150983CD24FB0D6963F7D28E17F72

28. Podemos probar también la consulta de los status de las órdenes con el método **GET** y el controlador **OrderStatuses**:

Launchpad GET http://... POST https://... PUT https://... DEL https://... PATCH https://... POST https://... + ... No Environment

GET http://localhost:64953/api/orderstatuses Send

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 237 ms Size: 535 B Save

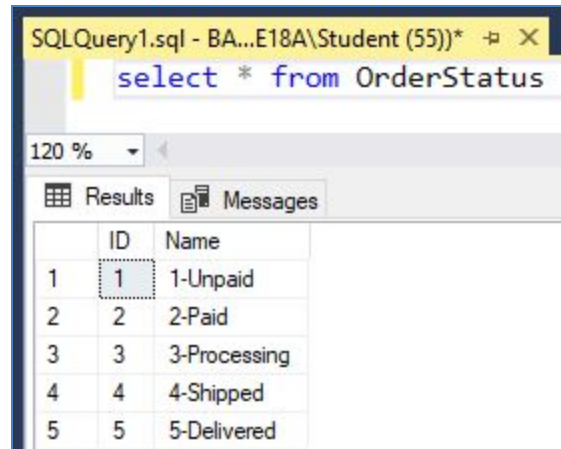
Pretty Raw Preview Visualize JSON

```

1  {
2    {
3      "id": 1,
4      "name": "1-Unpaid",
5      "customerOrder": null
6    },
7    {
8      "id": 2,
9      "name": "2-Paid",
10     "customerOrder": null
11   },
12   {
13     "id": 3,
14     "name": "3-Processing",
15     "customerOrder": null
16   },
17   {
18     "id": 4,
19     "name": "4-Shipped",
20     "customerOrder": null
21   }
22 }

```

Estos son los registros almacenados en la base de datos que la acción **GET** está recuperando desde el método **GetOrderStatus**:



The screenshot shows a SQL query window titled "SQLQuery1.sql - BA...E18A\Student (55))*". The query text is "select * from OrderStatus". Below the query, there is a zoom level of "120 %". The results are displayed in a table with two columns: "ID" and "Name". The table contains five rows of data. The first row is highlighted with a dashed border.

	ID	Name
1	1	1-Unpaid
2	2	2-Paid
3	3	3-Processing
4	4	4-Shipped
5	5	5-Delivered

29. Con esta mecánica de pruebas planteada, puede probar todos los métodos de los controladores de la api vía Postman.