API´s CON C# / ASP.NET

Programa: _____ADSO___

Competencia: Modelado de los artefactos del software.

R.A:

Verificar los entregables de la fase de diseño del software de acuerdo con lo establecido en el informe de análisis.

Aprendiz: _____

Ficha: _____

Instructor: Taller preparado por: Ing. Jairo Augusto Arboleda Londoño (PMP®) **Fecha**:

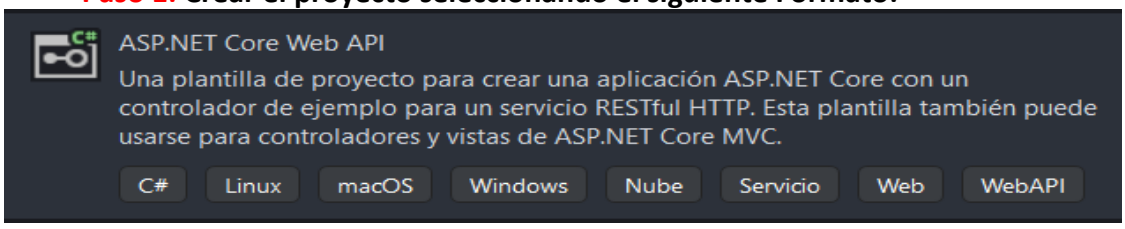
Puntuación máxima: **Tiempo estimado:** 2 horas

Taller Creación de una Web API Usando dotNetCore

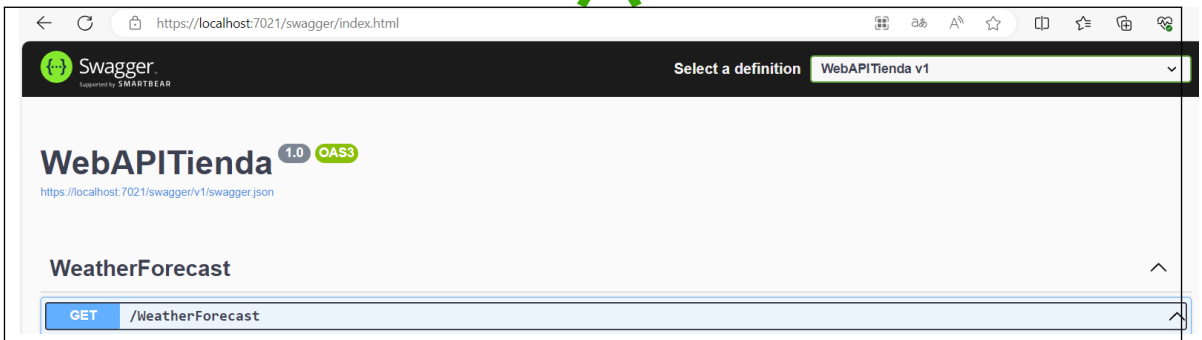
ESTRUCTURA Y COMPOENENTES

- Crear un proyecto usando SDK .NET 5
- Paquetes
  - Entity framework core
  - Entity framework tools
  - Entity framework sql server
- Modelo
  - Productos
- Contexto
- Controllador (Scafolding)
- Probar la webAPI con swagger

Paso 1: Crear el proyecto seleccionando el siguiente Formato:

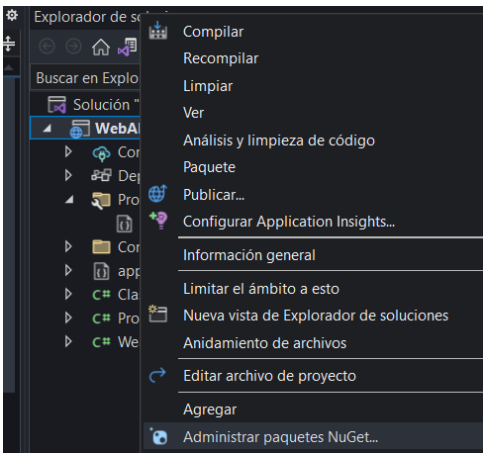


Al ejecutar la App sale este Demo:

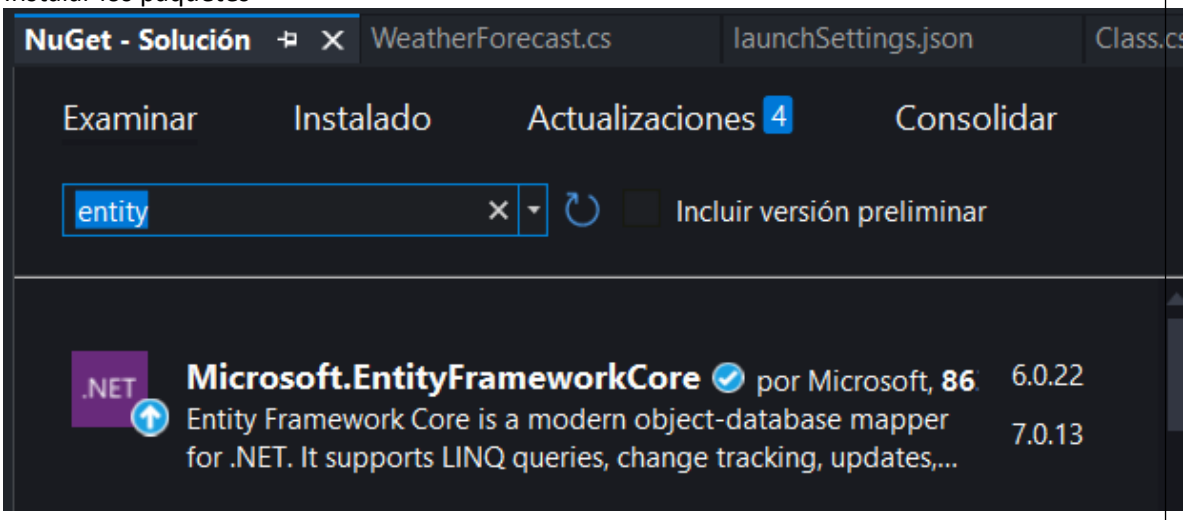


Paso 2: Adicionar los paquetes Nuget (OJO INSTALE LOS DE LA VERSIÓN 6.024) NO LA 6.022

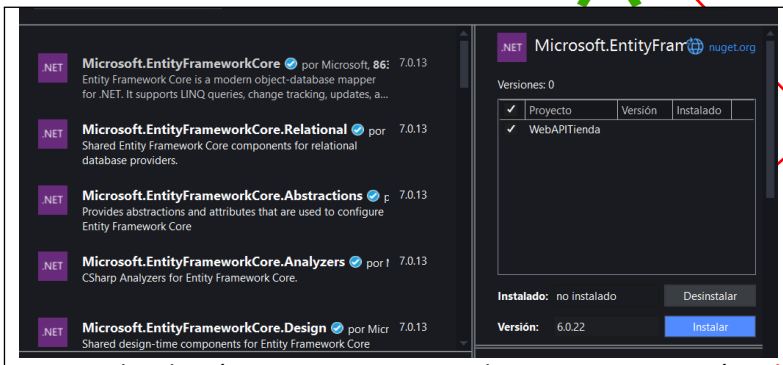
- Haga Clic derecho sobre el Proyecto.



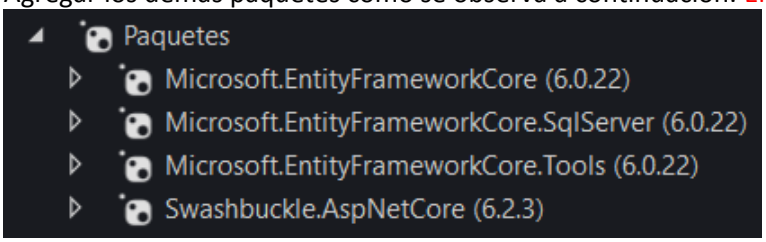
Instalar los paquetes



Paso 3: Elegir la versión, en este caso ya está instalada



Agregar los demás paquetes como se observa a continuación. El ultimo se agrega solo

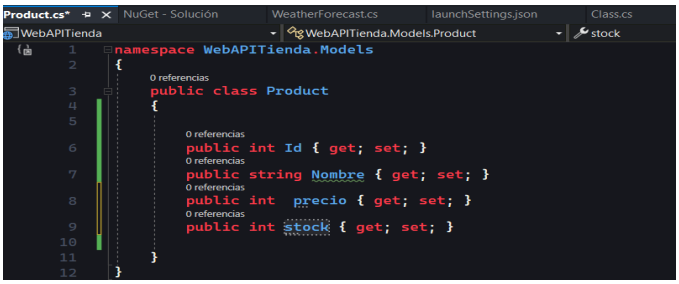


Paso 4: **Clic derecho sobre nombre del Proyecto**

Paso 5: Agregar una carpeta con el nombre **Models**

Paso 6: Dentro de **Models** Agregar una clase con el nombre de **Product**

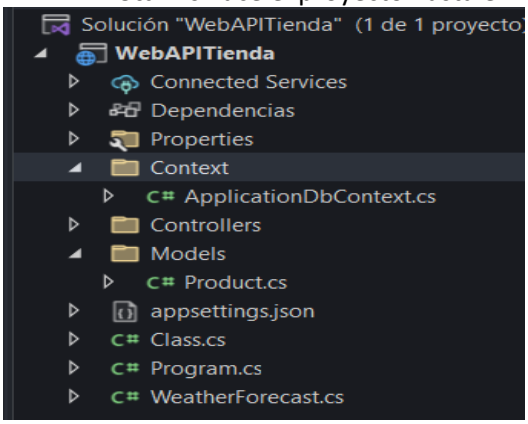
Paso 7: **Codificar Product**



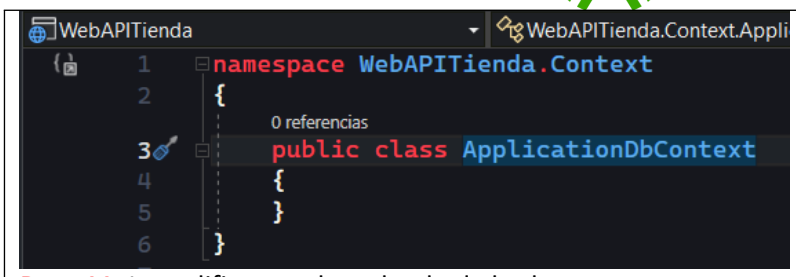
Paso 8: Crear otra carpeta de nombre **Context** al mismo nivel de Models

Paso 9: Dentro de **Context** agregamos un nuevo elemento o nueva **clase llamada** ApplicationDbContext.cs

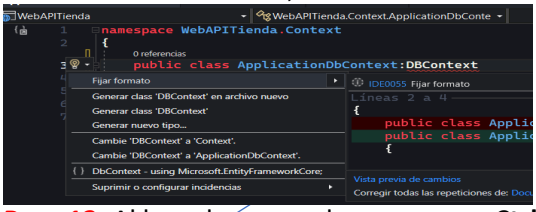
Nota: Así luce el proyecto hasta el momento.



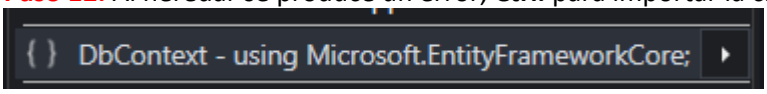
Paso 10 : Ahora abrimos esta clase que es la original



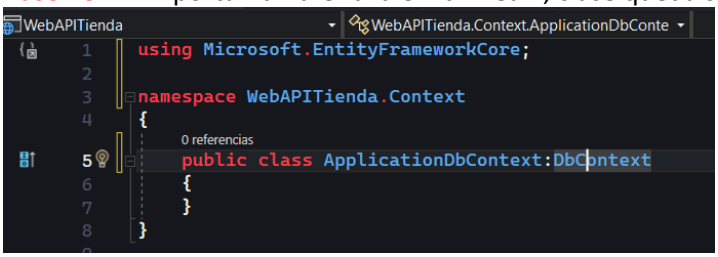
Paso 11: La codificamos, heredando de la clase **context**



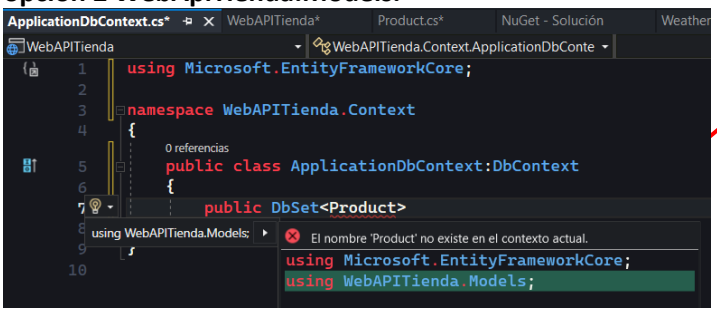
Paso 12: Al heredar se produce un error, **Ctrl.** para importar la clase que se necesita y elegir:



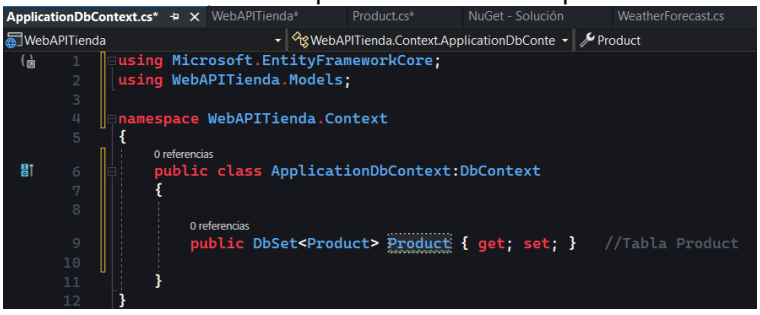
Paso 13: Al importar la librería la en la línea 1, clase queda sin error, así:



Paso 14: Ahora a codificarla, comience por la línea 7 y resuelva el error con **Ctrl. y elija la opción 2 WebApiTienda.Models.**

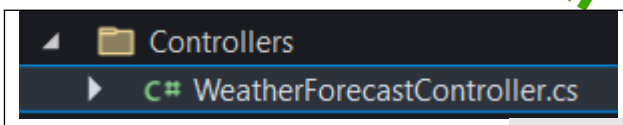


Nota: Ahora esta clase mapea nuestro modelo producto con la tabla Product y luce así:



Paso 15: Elimine la clase WeatherForecast que no se requiere.

Paso 16: Elimine también el controlador (**NO** la carpeta)de esta clase como se muestra



Paso 17: Luego agregar el contexto en el **`Program.cs para ello vaya a la documentación en la siguiente url:`**

https://learn.microsoft.com/es-es/ef/core/dbcontext-configuration/

Paso 18: Busque el siguiente código y cópielo sin las llaves como se indica en la figura siguiente:

Las aplicaciones de ASP.NET Core se configuran mediante la inserción de dependencias. EF Core se puede agregar a esta configuración mediante AddDbContext en el método ConfigureServices de `Startup.cs`. Por ejemplo:

C# Copiar

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

    services.AddDbContext<ApplicationDbContext>(
        options => options.UseSqlServer("name=ConnectionStrings:DefaultConne
}
```

Paso 19: Vaya a **program.cs** y copie este código anterior como se muestra a continuación.

```
builder.Services.AddSwaggerGen();
services.AddDbContext<ApplicationDbContext>(
        options => options.UseSqlServer("name=ConnectionStrings:DefaultConnection"));
var app = builder.Build();
```

Paso 20: Solucione el error en **services** agregando builder antes de services, y quedará así

```
builder.Services.AddSwaggerGen();
builder.Services.AddDbContext<ApplicationDbContext>(
        options => options.UseSqlServer("name=ConnectionStrings:DefaultConnection"));
var app = builder.Build();
```

Paso 21: Luego presione **Ctrl.** para usar el Framework de entity y solucionar el error en **UseSqlServer**

Finalmente, después de importar las dependencias debe quedar así y sin errores :

```
builder.Services.AddSwaggerGen();
builder.Services.AddDbContext<ApplicationDbContext>(
        options => options.UseSqlServer("Conexion"));
var app = builder.Build();
```

Crear string de conexión

Paso 22: Para conectarse a la Bd debes ingresar al Archivo **appsettings.json y modificar el archivo así:**

```
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "Conexion": "server= DESKTOP-P58V141\\SQLEXPRESS;Initial Catalog=Tienda;Integrated Security=True;"
  }

}
```

Recuerde que esta conexión varía dependiendo del equipo

Paso 23: Debes terminar de configurar la conexión a la BD así:

```
builder.Services.AddSwaggerGen();
builder.Services.AddDbContext<ApplicationDbContext>(
        options => options.UseSqlServer(builder.Configuration.GetConnectionString("Conexion")));
var app = builder.Build();
```

Paso 24: Luego debes adicionar este código en las líneas 12 a 16 en este caso en el **ApplicationDbContext.cs**

```
PITienda                                    WebAPITienda.Context.ApplicationDbContext
1   using Microsoft.EntityFrameworkCore;
2   using WebAPITienda.Models;
3
4   namespace WebAPITienda.Context
5   {
        3 referencias
6       public class ApplicationDbContext:DbContext
7       {
8           // mapea nuestro modelo producto con la tabla Product
                0 referencias
9           public DbSet<Product> Product { get; set; }   //Tabla Product
10
11          //Crear un controlador que recibe como parámetro el DbContextOptions
                0 referencias
12          public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
13          : base(options)
14          {
15
16          }
17
18      }
19  }
```

Paso 26: Crear la Migración: ¿qué es una migración?, ¿para qué se utiliza?

Vaya a herramientas Administrador de paquetes Nuget – consola de administrador de paquetes.

Paso 27: Escriba en la consola el siguiente comando para que se cree LA MIGRACIÓN.

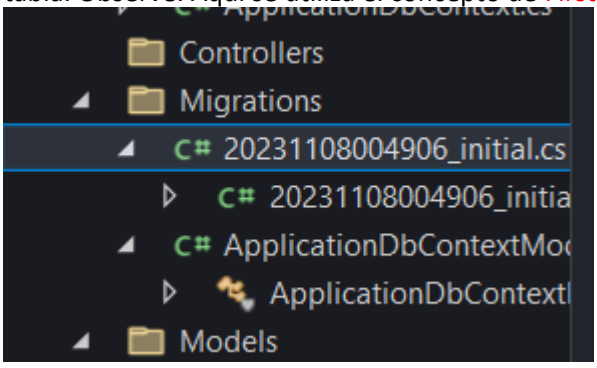
```
La licencia de cada paquete la concede su propietario. NuGet no se hace responsable de los paquetes de terceros ni concede ninguna licencia sobre ellos. Algu
rigen por licencias adicionales. Siga la URL de origen del paquete (fuente) para determinar cualquier dependencia.

Versión de host 6.6.0.61 de la Consola del Administrador de paquetes

Escriba 'get-help NuGet' para ver todos los comandos de NuGet disponibles.

PM> Add-Migration initial
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 6.0.22 initialized 'ApplicationDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer:6.0.22' with options: None
To undo this action, use Remove-Migration.
PM>
```

Paso 28: Al ejecutar el comando anterior, se creó la migración y a su vez la estructura de la tabla. Observe. Aquí se utiliza el concepto de First code then **Database**



Paso 28: Abrir el archivo **_initial.cs** y analizar la estructura de la Tabla que se creó:

```
APITienda                                   WebAPITienda.Migrations.initial                    Up(Mi
1   using Microsoft.EntityFrameworkCore.Migrations;
2
3   #nullable disable
4
5   namespace WebAPITienda.Migrations
6   {
        1 referencia
7       public partial class initial : Migration
8       {
            0 referencias
9           protected override void Up(MigrationBuilder migrationBuilder)
10          {
11              migrationBuilder.CreateTable(
12                  name: "Product",
13                  columns: table => new
14                  {
15                      Id = table.Column<int>(type: "int", nullable: false)
16                          .Annotation("SqlServer:Identity", "1, 1"),
17                      Nombre = table.Column<string>(type: "nvarchar(max)", nullable: false),
18                      precio = table.Column<int>(type: "int", nullable: false),
19                      stock = table.Column<int>(type: "int", nullable: false)
20                  },
21                  constraints: table =>
22                  {
23                      table.PrimaryKey("PK_Product", x => x.Id);
24                  });
25          }
```

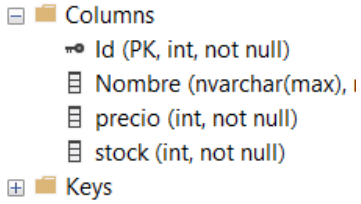
¿A QUÉ CONCLUSIÓN LLEGAS SOBRE ESTA CLASE?

Paso 29: Crear el modelo en SQL

Para crear la BD en SQL Server escriba el siguiente comando:

```
PM> Update-database
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 6.0.22 initialized 'ApplicationDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer:6.0.22' with options: None
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (2,307ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
      CREATE DATABASE [Tienda];
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (491ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
      IF SERVERPROPERTY('EngineEdition') <> 5
      BEGIN
          ALTER DATABASE [Tienda] SET READ_COMMITTED_SNAPSHOT ON;
      END;
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (19ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT 1
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (398ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE [__EFMigrationsHistory] (
          [MigrationId] nvarchar(150) NOT NULL,
          [ProductVersion] nvarchar(32) NOT NULL,
          CONSTRAINT [PK___EFMigrationsHistory] PRIMARY KEY ([MigrationId])
```

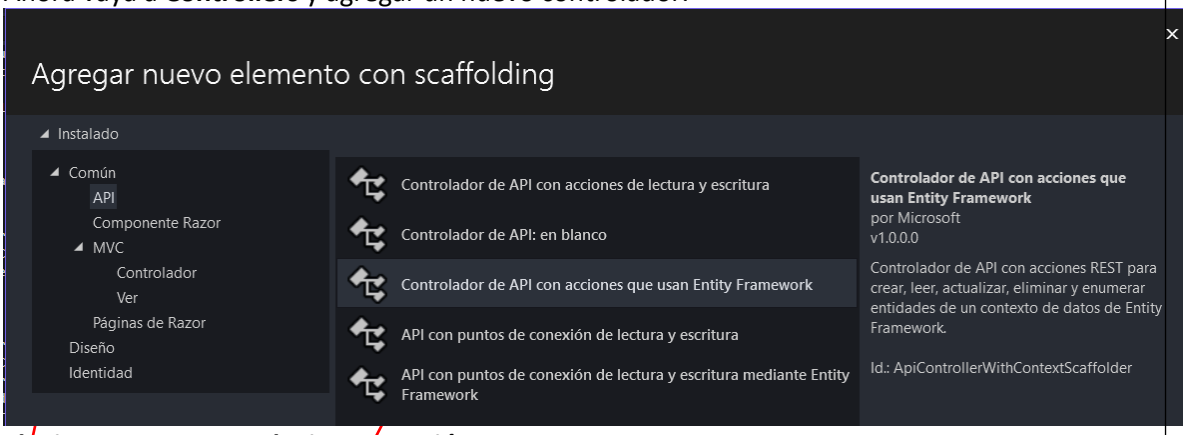
Paso 30: Vaya a SQL Server y refresque el servidor, ya le debería aparecer esto:



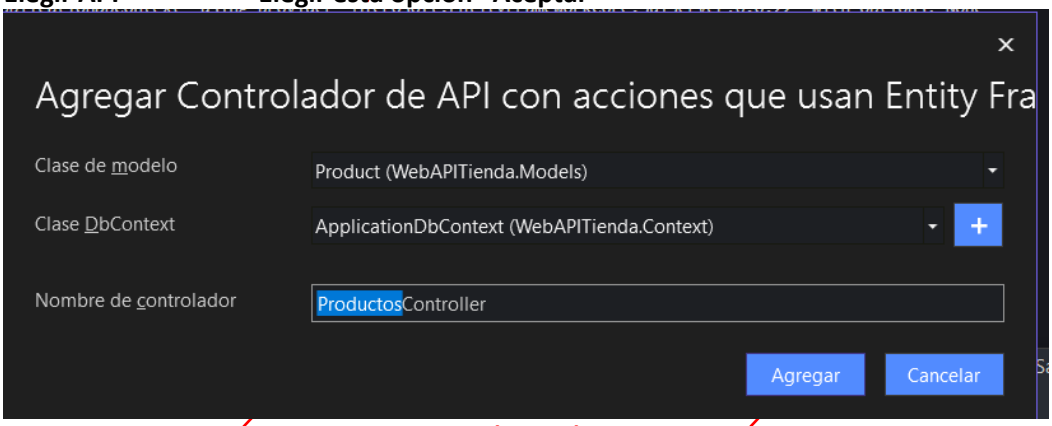
La tabla de migración contiene el histórico de los cambios en la BD.

Paso 31: Poblar la tabla con 10 registros (Ingrese datos de vehiculos)Preguntar Instructor)
Paso 32: Agregar el controlador

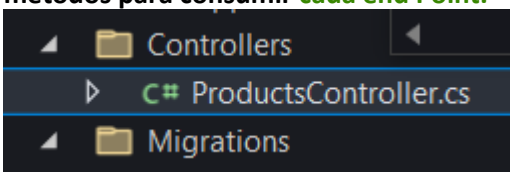
Ahora vaya a **Controllers** y agregar un nuevo controlador.



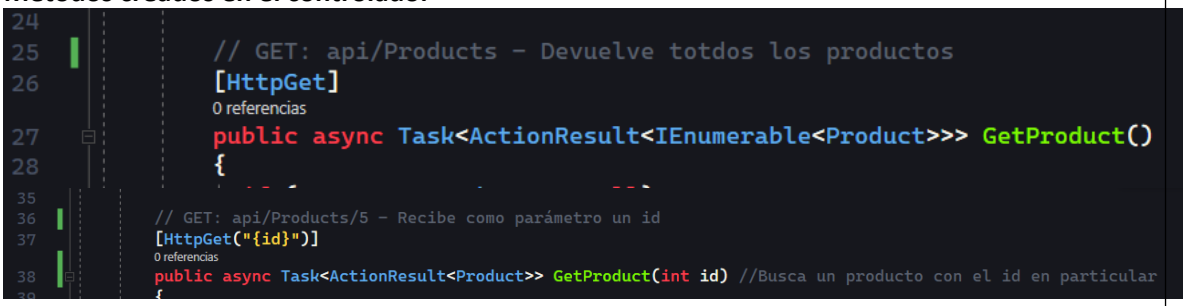
Elegir API **Elegir esta opción** **Aceptar**



Paso 33: Colocar nombre al controlador(arriba)
Paso 34: Vaya al controlador creado y analice el contenido de este archivo, contiene todos los métodos para consumir cada end Point.

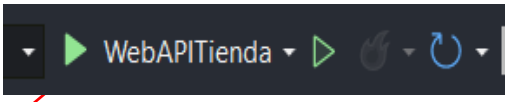


Métodos creados en el controlador



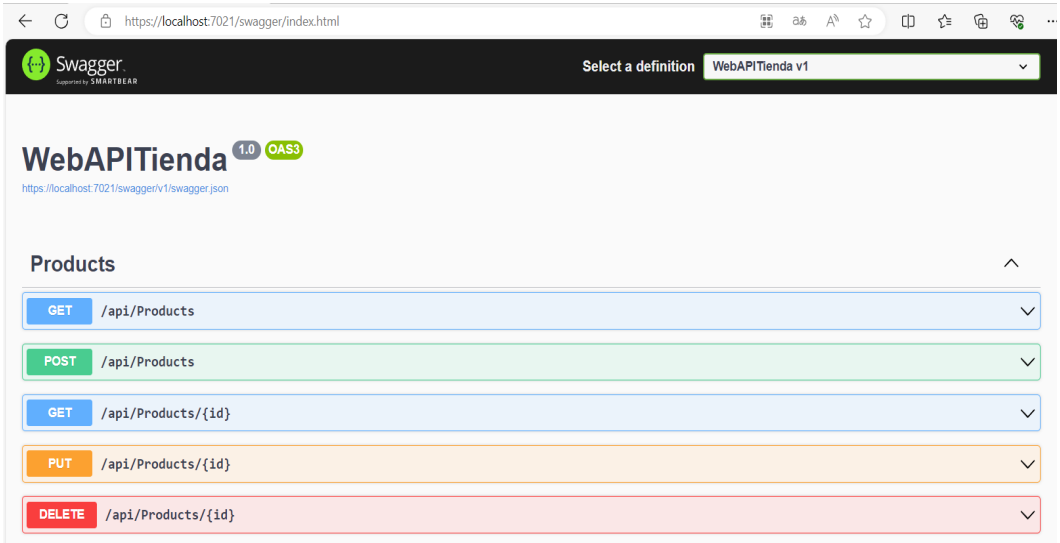
```
54          // PUT: api/Products/5
55          // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
56          [HttpPut("{id}")] // Actualiza un producto en particular
            0 referencias
85          // POST: api/Products - Se encarga de agregar un nuevo producto
86          // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
87          [HttpPost]
            0 referencias
88          public async Task<ActionResult<Product>> PostProduct(Product product)
89          {
99
100             // DELETE: api/Products/5  - Elimina u producto de la BD
101             [HttpDelete("{id}")]
                0 referencias
102             public async Task<IActionResult> DeleteProduct(int id)
103             {
```

Paso 36: Finalmente, abra **Swagger** dando Click en este botón



Paso 37: Probar la API

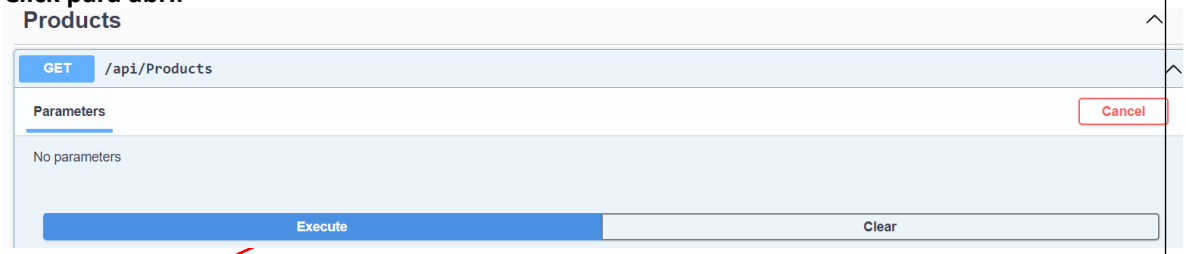
Al ejecutar la **API** debes visualizar Swagger que es un formato de descripción de API para API REST. Podrás consumir los end Points desde este.



Paso 38: Probar cada End Point

Probemos el primer End Point Método GET

Click para abrir



Click aquí

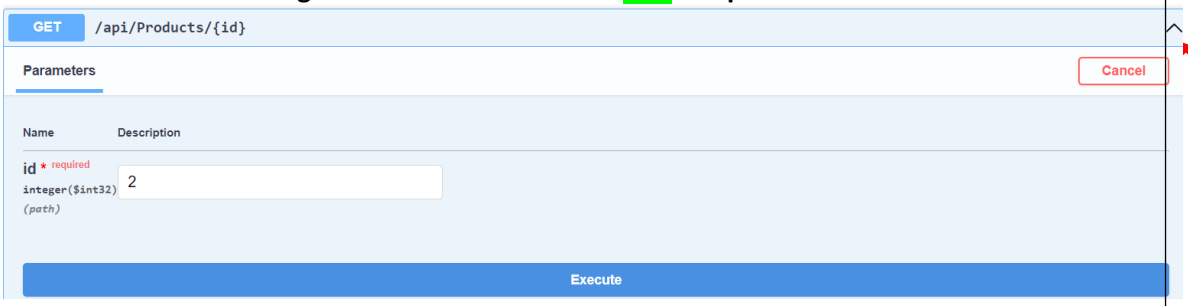
Resultados: debe devolver todos los productos así:

200

Response body


```
[
  {
    "id": 1,
    "nombre": "Renault sandero",
    "precio": 2000,
    "stock": 12
  },
  {
    "id": 2,
    "nombre": "Renault Stepway",
    "precio": 1456,
    "stock": 10
  },
  {
    "id": 3,
    "nombre": "Renault sandero",
    "precio": 2000,
    "stock": 12
  },
  {
    "id": 4,
    "nombre": "Renault Stepway",
    "precio": 1456,
    "stock": 10
  },
  {
    "id": 5,
    "nombre": "BMW Salento",
    "precio": 10000,
    "stock": 32
  },
  {
    "id": 6,
    "nombre": "Toyota Prado",
    "precio": 56456,
    "stock": 1
  },
  {
    "id": 7,
    "nombre": "Topolino Sakura",
    "precio": 40000,
    "stock": 3
  }
]
```


Paso 39: **Probemos el segundo End Point** **Método GET con parámetro**



Resultados: debe devolver los productos con Id = 2

Request URL

```
https://localhost:7021/api/Products/2
```

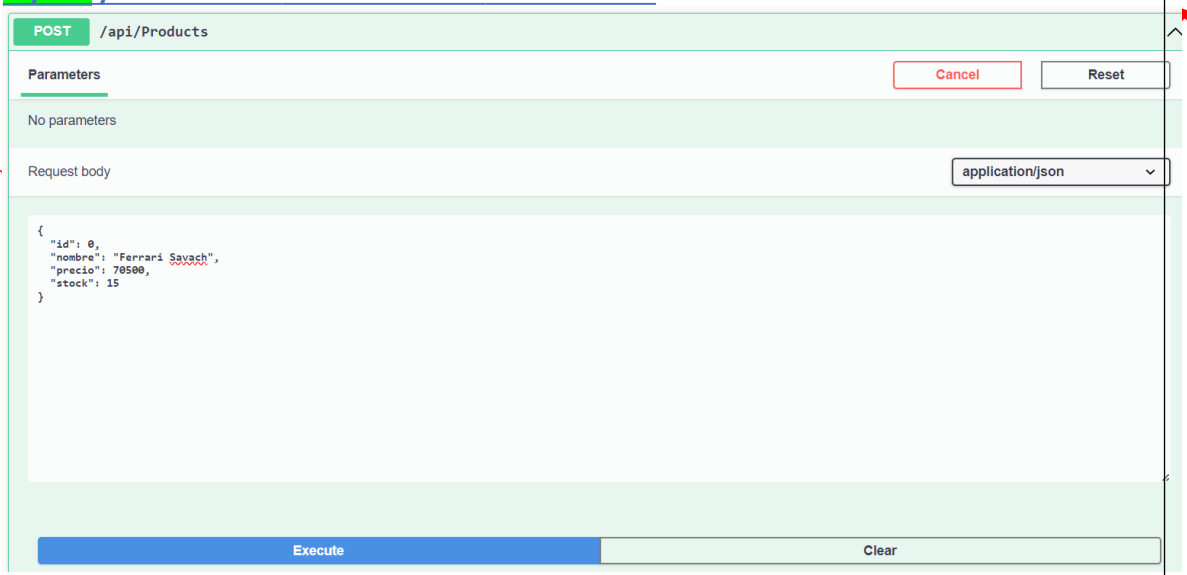
Server response

Code	Details
200	Response body

```
{
  "id": 2,
  "nombre": "Renault Stepway",
  "precio": 1456,
  "stock": 10
}
```

Nota: recuerde usar el botón Try it out" para que se active el End Point y pueda agregar los datos.

Paso 40: Probemos el tercer End Point Método POST (Insertar registros) recuerde dar clic en Try out y como el id es autoincremental no lo cambie.



Después del paso anterior vaya al método get y retorne todos los productos, ahí verá si se creó el nuevo producto

Resultados:

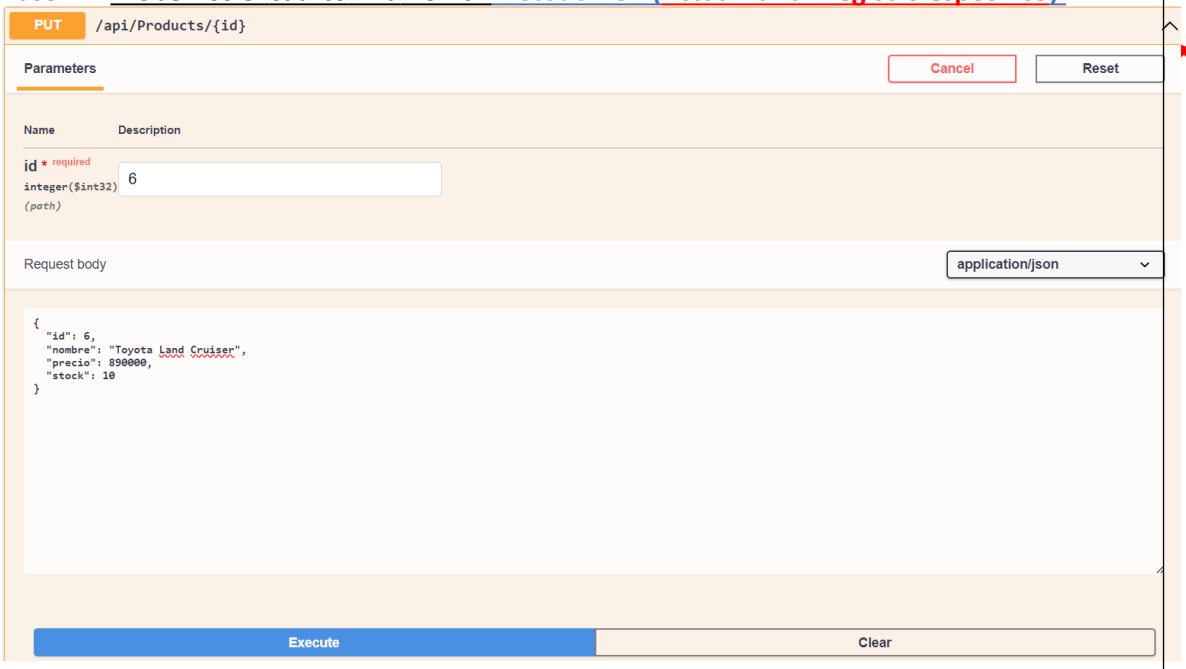
```
{
  "id": 8,
  "nombre": "Ferrari Savach",
  "precio": 70500,
  "stock": 15
}
```


O bien en la Base de datos se reflejaría también.

Results | Messages

	Id	Nombre	precio	stock
1	1	Renault sandero	2000	12
2	2	Renault Stepway	1456	10
3	3	Renault sandero	2000	12
4	4	Renault Stepway	1456	10
5	5	BMW Salento	10000	32
6	6	Toyota Prado	56456	1
7	7	Topolino Sakura	40000	3
8	8	Ferrari Savach	70500	15

Paso 41: Probemos el cuarto End Point Método PUT (Actualizar un registro especifico).



Resultados:

Server response

Code	Details
200	Response body

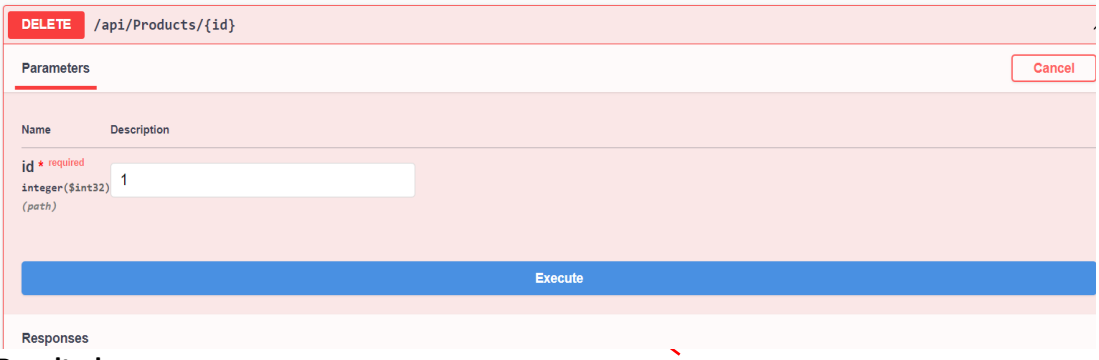
```
{
  "id": 6,
  "nombre": "Toyota Land Cruiser",
  "precio": 890000,
  "stock": 10
}
```

Se observa que el registro 6 fue cambiado según tabla anterior.

Results | Messages

	Id	Nombre	precio	stock
1	1	Renault sandero	2000	12
2	2	Renault Stepway	1456	10
3	3	Renault sandero	2000	12
4	4	Renault Stepway	1456	10
5	5	BMW Salento	10000	32
6	6	Toyota Land Cruiser	890000	10
7	7	Topolino Sakura	40000	3
8	8	Ferrari Savach	70500	15

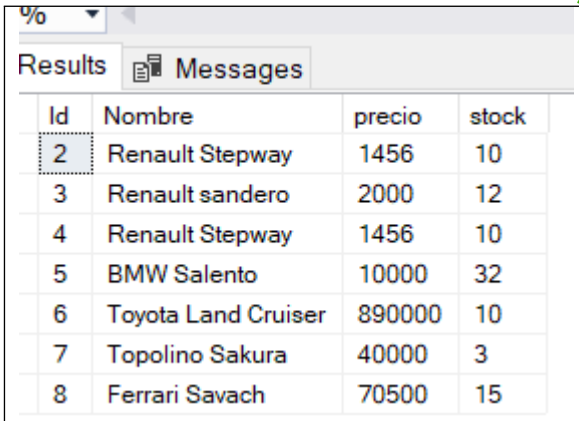
Paso 42: Probemos el quinto End Point Método DELETE (Eliminar registros)



Resultado:

Se observa el mensaje Success desde el método get

Si observamos los resultados de la tabla en SQL ya no se visualiza el registro 1.



Results Messages

Id	Nombre	precio	stock
2	Renault Stepway	1456	10
3	Renault sandero	2000	12
4	Renault Stepway	1456	10
5	BMW Salento	10000	32
6	Toyota Land Cruiser	890000	10
7	Topolino Sakura	40000	3
8	Ferrari Savach	70500	15

Te invito a realizar una lista de lecciones aprendidas durante el desarrollo de este tema, API Rest

Si llegaste hasta aquí fue porque lo disfrutaste

"Felicitaciones"

FIN