

# **Tecnologías de Desarrollo de Software IDE**

## **Implementación de una API Rest usando Net Core**

---



# Implementación de una API Rest usando Net Core

---

Para empezar podemos distinguir tres partes:

- 1. Definir la base de datos.
- 2. Generar el acceso a la base de datos
- 3. Definir los controladores y sus funcionalidades. Estos métodos serán los encargados de brindar los servicios web para su posterior uso.

Empecemos!

# Implementación de una API Rest usando Net Core

---

Para definir la Base de Datos, tomaremos de referencia la tabla Alumnos:

Nombre	Tipo de datos	Permitir valores NULL	Valor predeterminado
DNI	varchar(20)	<input type="checkbox"/>	
ApellidoNombre	varchar(50)	<input type="checkbox"/>	
Email	varchar(50)	<input checked="" type="checkbox"/>	(NULL)
FechaNacimiento	datetime	<input type="checkbox"/>	
NotaPromedio	decimal(4,2)	<input checked="" type="checkbox"/>	

# Implementación de una API Rest usando Net Core

---

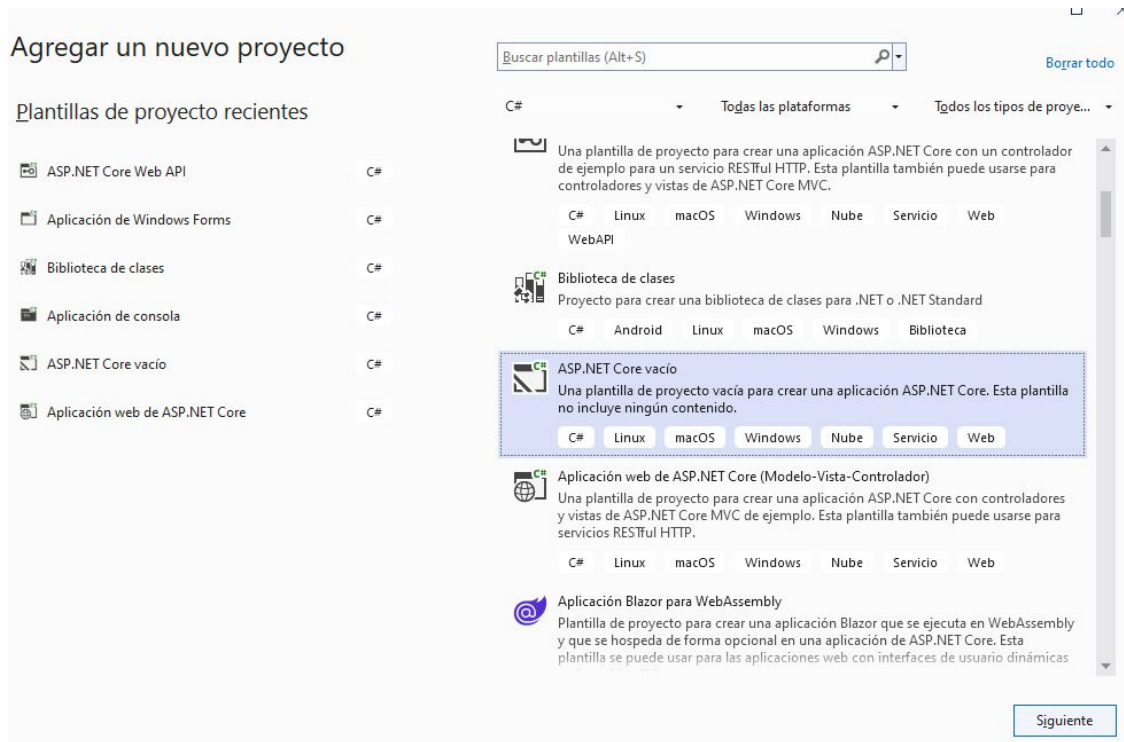
La sentencia SQL para crear la tabla Alumnos es:

```
CREATE TABLE [dbo].[Alumnos] (  
    [DNI]          VARCHAR (20)    NOT NULL,  
    [ApellidoNombre] VARCHAR (50)  NOT NULL,  
    [Email]        VARCHAR (50)    DEFAULT (NULL) NULL,  
    [FechaNacimiento] DATETIME     NOT NULL,  
    [NotaPromedio]  DECIMAL (4, 2) NULL,  
    CONSTRAINT [PK_Alumnos] PRIMARY KEY CLUSTERED ([DNI] ASC)  
);
```

Con la Base de Datos ya preparada podemos comenzar con la implementación del primer proyecto para el desarrollo de servicios API Rest.

# Implementación de una API Rest usando Net Core

En Visual Studio 2022 lo primero que haremos es crear un nuevo proyecto de tipo ASP.Net Core Vacío:



# Implementación de una API Rest usando Net Core

Recuerden verificar que en la siguiente pantalla las selecciones sean:

Información adicional

ASP.NET Core vacío   C#   Linux   macOS   Windows   Nube   Servicio   Web

Framework ⓘ

.NET 7.0 (Soporte técnico de términos estándar) ▼

☒ Configurar para HTTPS ⓘ

☐ Habilitar Docker ⓘ

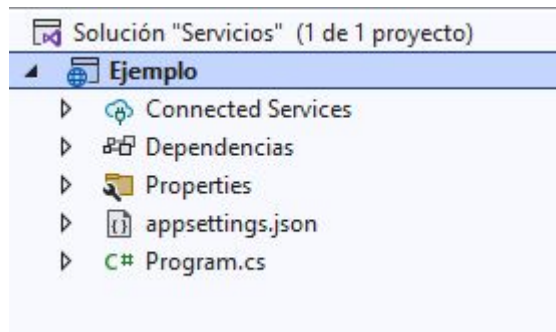
Sistema operativo de Docker ⓘ

Linux ▼

☐ No usar instrucciones de nivel superior ⓘ

# Implementación de una API Rest usando Net Core

Una vez creado el proyecto, en el Explorador de Soluciones veremos algo así:



Dentro del mismo vamos a crear tres carpetas (folders):

- **Controllers**
- **Models**
- **Context**

donde vamos a crear toda la funcionalidad de la aplicación.



# Implementación de una API Rest usando Net Core

---

En la carpeta **Controllers** es donde vamos a tener las clases con todos los métodos expuestos: GET, POST, PUT y DELETE.

**Models** es donde vamos a tener las clases con la estructura de la información que vamos a manipular, para este ejemplo es la misma estructura de la tabla de base de datos.

**Context** es donde vamos a tener la clase con el mapeo para la base de datos; donde vamos a asociar las clases de modelos y las tablas de la base de datos.



# Implementación de una API Rest usando Net Core

---

Para poder crear las entidades a través de clases y la conexión de la base de datos se puede emplear Entity Framework, el cual permite hacer *scaffolding* desde la base de datos hacia el proyecto, es decir, generar clases automáticamente de acuerdo a las entidades establecidas en la base de datos y la conexión en el proyecto. En nuestro caso lo vamos a hacer manualmente para ir siguiendo el paso a paso.

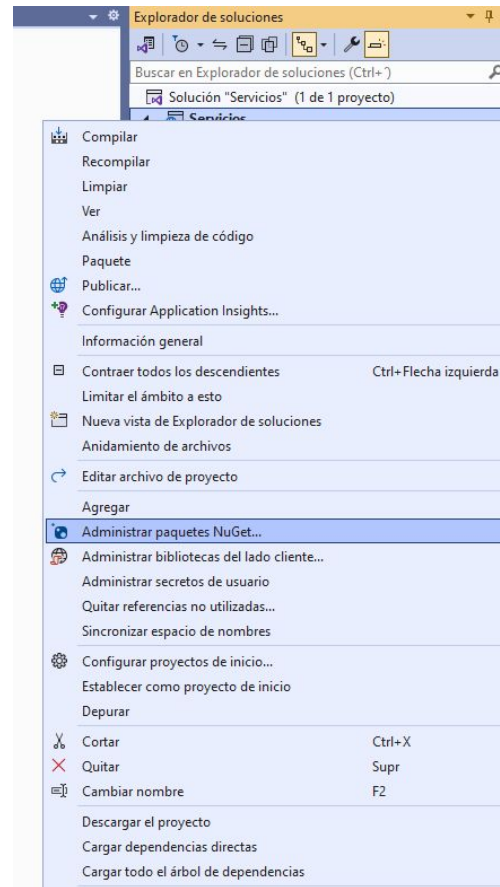
Para este propósito, es necesario instalar tres paquetes NuGet en nuestro proyecto:

- `Microsoft.EntityFrameworkCore.SqlServer`
- `Microsoft.Extensions.Configuration.Abstractions`
- `Swashbuckle.AspNetCore`

Esta última es opcional pero recomendable. La vamos a usar para poder documentar y probar nuestra aplicación con Swagger.

# Implementación de una API Rest usando Net Core

**Observación:** para encontrar el centro de administración de los paquetes NuGet podemos dirigirnos al menú de opciones -> proyecto -> Administrar paquetes NuGet.



# Implementación de una API Rest usando Net Core














Al abrirse la ventana vamos a la opción Examinar y vamos poniendo cada una de las librerías que queremos agregar para luego instalarlas:


Administrador de paquetes NuGet: Servicios

Origen del paquete: [nuget.org](#)

Examinar Instalado Actualizaciones 1

Microsoft.EntityFrameworkCore.SqlServer x ↻ ☐ Incluir versión preliminar

	<b>Microsoft.EntityFrameworkCore.SqlServer</b>  por Microsoft, 378M descargas 7.0.11 Microsoft SQL Server database provider for Entity Framework Core.
	<b>Microsoft.EntityFrameworkCore</b>  por Microsoft, 822M descargas 7.0.11 Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB,...
	<b>Microsoft.EntityFrameworkCore.SqlServer.NetTopologySuite</b>  por Microsoft, 12,7M descarg 7.0.11 NetTopologySuite support for the Microsoft SQL Server database provider for Entity Framework Core.
	<b>Microsoft.EntityFrameworkCore.SqlServer.Design</b>  por Microsoft, 11,5M descargas 1.1.6 ⚠️ Design-time Entity Framework Core Functionality for Microsoft SQL Server. Esta versión del paquete está obsoleta.
	<b>Z.EntityFramework.Extensions.EFCore</b>  por ZZZ Projects, 25,5M descargas Microsoft.EntityFrameworkCore Extension Methods
	<b>Stenn.EntityFrameworkCore.SqlServer</b> por Stenn International, 68K descargas 7.9.50 Entity Framework Core SqlServer specific extensions
	<b>Stenn.EntityFrameworkCore.StaticMigrations</b> por Stenn International, 91,6K descargas 7.9.50 Entity Framework Core static migrations
	<b>Stenn.EntityFrameworkCore.SqlServer.Extensions.DependencyInjection</b> por Stenn Internatic 7.9.50 Dependency injection extensions for Stenn.AspNetCore.OData.Versioning

**Microsoft.EntityFrameworkCore**  [nuget.org](#)

Versión: [Versión estable más reciente 7.0.11](#) [Instalar](#)

La asignación del origen del paquete está desactivada. [Configur](#)

**Opciones**

**Descripción**  
Microsoft SQL Server database provider for Entity Framework Core.

**Versión:** 7.0.11  
**Autores:** Microsoft  
**Licencia:** [MIT](#)  
**Descargas:** 378.268.844  
**Fecha de publicación:** martes, 12 de septiembre de 2023 (12/09/2023)  
**URL del proyecto:** <https://docs.microsoft.com/ef/core/>  
**Notificar abuso:** <https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.SqlServer/7.0.11/ReportAbuse>

**Etiquetas:** Entity, Framework, Core, entity-framework-core, EF, Data, O/RM, EntityFramework, EntityFrameworkCore, EFCore, SQL, Server



**Dependencias**

- net6.0
- Microsoft.EntityFrameworkCore.Relational (>= 7.0.11)
- Microsoft.Data.SqlClient (>= 5.1.1)


# Implementación de una API Rest usando Net Core




Una vez realizada la instalación deberíamos ver:

[Examinar](#) [Instalado](#) [Actualizaciones](#)

  ☐ Incluir versión preliminar

---

 Paquetes de nivel superior (3)

	<b>Microsoft.EntityFrameworkCore.SqlServer</b> por Microsoft Microsoft SQL Server database provider for Entity Framework Core.	7.0.11
	<b>Microsoft.Extensions.Configuration.Abstractions</b> por Microsoft Abstractions of key-value pair based configuration.	7.0.0
	<b>Swashbuckle.AspNetCore</b> por Swashbuckle.AspNetCore Swagger tools for documenting APIs built on ASP.NET Core	6.5.0

# Implementación de una API Rest usando Net Core

Ahora vamos a crear la conexión con la base de datos. Para esto abrimos el archivo «appsettings.json» donde veremos una estructura similar a este código:

```
1  {  
2    "Logging": {  
3      "LogLevel": {  
4        "Default": "Information",  
5        "Microsoft.AspNetCore": "Warning"  
6      }  
7    },  
8    "AllowedHosts": "*"   
9  }  
10
```



# Implementación de una API Rest usando Net Core

---

Lo que tenemos que hacer es agregar la conexión a la Base de Datos. Para esto tenemos que poner una sentencia similar a esta adecuando el nombre del servidor, el nombre de la Base de Datos y, si usa seguridad integrada o si se debe pasar el usuario y contraseña:

```
"ConnectionStrings": {  
  "DefaultConnection": "data source=DESKTOP-NUTBUG5\\SQLEXPRESS;initial  
Catalog=Net.DB.Alumno;Integrated Security=True;TrustServerCertificate=True;"  
}
```

# Implementación de una API Rest usando Net Core

---

El resultado final del archivo debería ser:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "data source=DESKTOP-NUTBUG5\\SQLEXPRESS;initial Catalog=Net.DB.Alumno;Integrated Security=True;TrustServerCertificate=True;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

# Implementación de una API Rest usando Net Core

---

Ahora abrimos la clase llamada «Program.cs», la que estaremos modificando en la medida que vayamos avanzando. Actualmente el código que se van a encontrar es:

```
1  var builder = WebApplication.CreateBuilder(args);  
2  var app = builder.Build();  
3  
4  app.MapGet("/", () => "Hello World!");  
5  
6  app.Run();
```

---



# Implementación de una API Rest usando Net Core

Vamos a borrar la línea 4 (que dice `app.MapGet(...)`) y vamos a cambiar el contenido por el siguiente código:

```
1  using System.Data.Common;
2  using Microsoft.EntityFrameworkCore;
3  using Ejemplo.Context;
4
5  var builder = WebApplication.CreateBuilder(args);
6  builder.Services.AddControllers();
7  // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
8  builder.Services.AddEndpointsApiExplorer();
9  builder.Services.AddSwaggerGen();
10 var app = builder.Build();
11 // Configure the HTTP request pipeline.
12 if (app.Environment.IsDevelopment())
13 {
14     app.UseSwagger();
15     app.UseSwaggerUI();
16 }
17 app.UseHttpsRedirection();
18 app.UseAuthorization();
19 app.MapControllers();
20 app.Run();
```



# Implementación de una API Rest usando Net Core

---

Como pueden ver genera un error en la línea 3. Eso es porque todavía no generamos el namespace Context. No se preocupen, ya lo estaremos haciendo en los siguientes pasos.

Lo que hacemos en estas líneas es agregar la configuración para poder utilizar swagger con nuestra aplicación, el mapeo a los controladores y la redirección para el https.

# Implementación de una API Rest usando Net Core

El siguiente paso es crear las clases necesarias para el programa. Vamos a seleccionar la carpeta **Models** y agregamos una clase llamada **Alumnos.cs** y copiamos el siguiente código:

La etiqueta Key es para marcar que DNI es la clave primaria en la tabla en la Base de Datos. Si tuvieramos un Id que fuera la clave primaria no es necesario identificarla de esta manera ya que es la clave primaria que se asume por defecto.

```
1  using System.ComponentModel.DataAnnotations;
2
3  namespace Ejemplo.Models
4  {
5      public class Alumno
6      {
7
8          [Key]
9          public String DNI { get; set; }
10         public String ApellidoNombre { get; set; }
11         public String Email { get; set; }
12         public DateTime FechaNacimiento { get; set; }
13         public decimal NotaPromedio { get; set; }
14     }
15 }
16
17
```

# Implementación de una API Rest usando Net Core

Ahora, vamos a crear un contexto de base de datos para nuestro modelo Alumno. Para esto, creamos una clase llamada **MyDbContext.cs** en la carpeta **Context** y escribimos el siguiente código:

```
1 using Microsoft.EntityFrameworkCore;
2 using Ejemplo.Models;
3
4 namespace Ejemplo.Context
5 {
6     public class MyDbContext : DbContext
7     {
8         public MyDbContext(DbContextOptions<MyDbContext> options) : base(options) { }
9         public DbSet<Alumno> Productos { get; set; }
10    }
11 }
```

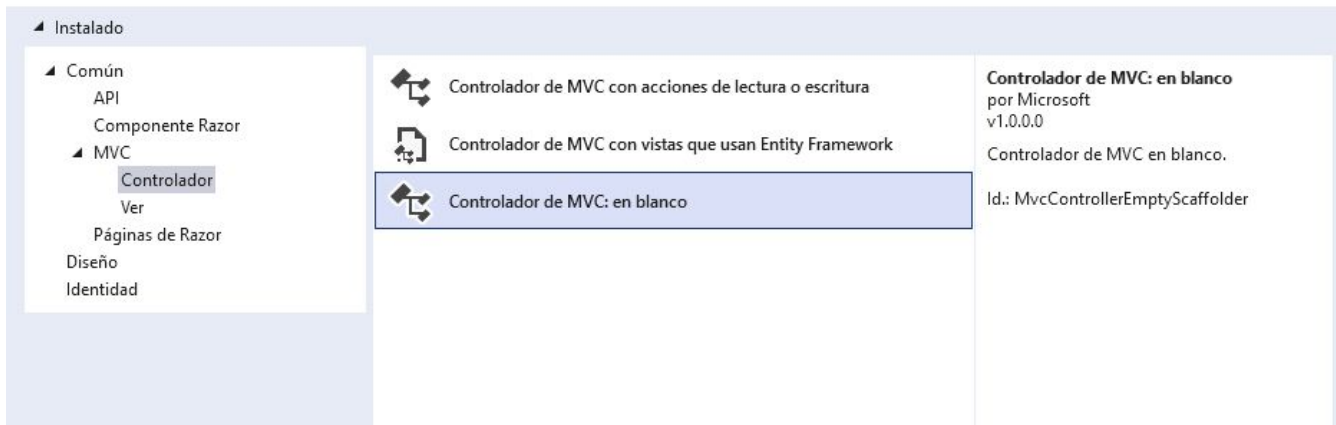
Como podemos ver, la clase hereda de **DbContext** y cada modelo o tabla nueva debemos agregarla en esta clase para poder manipularla en nuestra aplicación.

# Implementación de una API Rest usando Net Core

El siguiente paso es crear el controlador, donde vamos a tener los métodos que va a utilizar nuestra aplicación. En la carpeta **Controllers** click derecho, add, y seleccionamos **Controller**, ahí seleccionamos la primera opción MVC Controller – Empty y en la siguiente pantalla escribimos el nombre de la clase **AlumnoController.cs**.

Esta clase tendrá los métodos GET, POST, PUT y DELETE.

Agregar nuevo elemento con scaffolding



# Implementación de una API Rest usando Net Core

Dentro copiamos el siguiente código:

```
1 using Microsoft.AspNetCore.Mvc;
2 using Microsoft.EntityFrameworkCore;
3 using Ejemplo.Context;
4 using Ejemplo.Models;
5 namespace Ejemplo.Controllers
6 {
7     [ApiController]
8     [Route("api/[controller]")]
9     public class AlumnoController : ControllerBase
10    {
11        private readonly MyDbContext _context;
12        public AlumnoController(MyDbContext context)
13        {
14            _context = context;
15        }
16        [HttpGet(Name = "GetAlumno")]
17        public ActionResult<IEnumerable<Alumno>> GetAll()
18        {
19            return _context.Alumnos.ToList();
20        }
21        [HttpGet("{DNI}")]
22        public ActionResult<Alumno> GetById(String DNI)
23        {
24            var alumno = _context.Alumnos.Find(DNI);
25            if (alumno == null)
26            {
27                return NotFound();
28            }
29            return alumno;
30        }
31    }
```

# Implementación de una API Rest usando Net Core

Dentro copiamos el siguiente código:

```
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62

[HttpPost]
0 referencias
public ActionResult<Alumno> Create(Alumno alumno)
{
    _context.Alumnos.Add(alumno);
    _context.SaveChanges();
    return CreatedAtAction(nameof(GetById), new { DNI = alumno.DNI }, alumno);
}

[HttpPut("{DNI}")]
0 referencias
public ActionResult Update(string DNI, Alumno alumno)
{
    if (DNI != alumno.DNI)
    {
        return BadRequest();
    }
    _context.Entry(alumno).State = EntityState.Modified;
    _context.SaveChanges();
    return NoContent();
}

[HttpDelete("{DNI}")]
0 referencias
public ActionResult<Alumno> Delete(String DNI)
{
    var alumno = _context.Alumnos.Find(DNI);
    if (alumno == null)
    {
        return NotFound();
    }
    _context.Alumnos.Remove(alumno);
    _context.SaveChanges();
    return alumno;
}
}
```



# Implementación de una API Rest usando Net Core

---

El controlador `AlumnoController` contiene métodos para:

- recuperar todos los alumnos (GetAll),
- recuperar un alumno por su DNI (GetById),
- crear un nuevo alumno(Create),
- actualizar un alumno existente (Update) y
- eliminar un alumno (Delete).





# Implementación de una API Rest usando Net Core

---

Regresamos a la clase Program.cs para agregar la referencia y configuración de nuestra base de datos. Copiamos la siguiente línea y la insertamos debajo de `builder.Services.AddControllers()`.

```
builder.Services.AddDbContext<MyDbContext>(
    options => options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
```

# Implementación de una API Rest usando Net Core

El resultado final de la clase debe verse así:

```
1  using System.Data.Common;
2  using Microsoft.EntityFrameworkCore;
3  using Ejemplo.Context;
4
5  var builder = WebApplication.CreateBuilder(args);
6  builder.Services.AddControllers();
7  builder.Services.AddDbContext<MyDbContext>(
8      options => options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
9  // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
10 builder.Services.AddEndpointsApiExplorer();
11 builder.Services.AddSwaggerGen();
12 var app = builder.Build();
13 // Configure the HTTP request pipeline.
14 if (app.Environment.IsDevelopment())
15 {
16     app.UseSwagger();
17     app.UseSwaggerUI();
18 }
19 app.UseHttpsRedirection();
20 app.UseAuthorization();
21 app.MapControllers();
22 app.Run();
```

# Implementación de una API Rest usando Net Core

---

Como paso final vamos a modiicar el archivo `launchSettings.json` que está dentro de la carpeta `Properties`.

Vamos a agregar

```
“$schema”: “https://json.schemastore.org/launchsettings.json”,
```

en la primera linea del json.

Después agregamos

```
“launchUrl”: “swagger”
```

dentro del nodo profiles Ejemplo, debajo de la linea “launchBrowser”: “true”, también agregamos esa misma línea en el nodo «IIS Express.

# Implementación de una API Rest usando Net Core

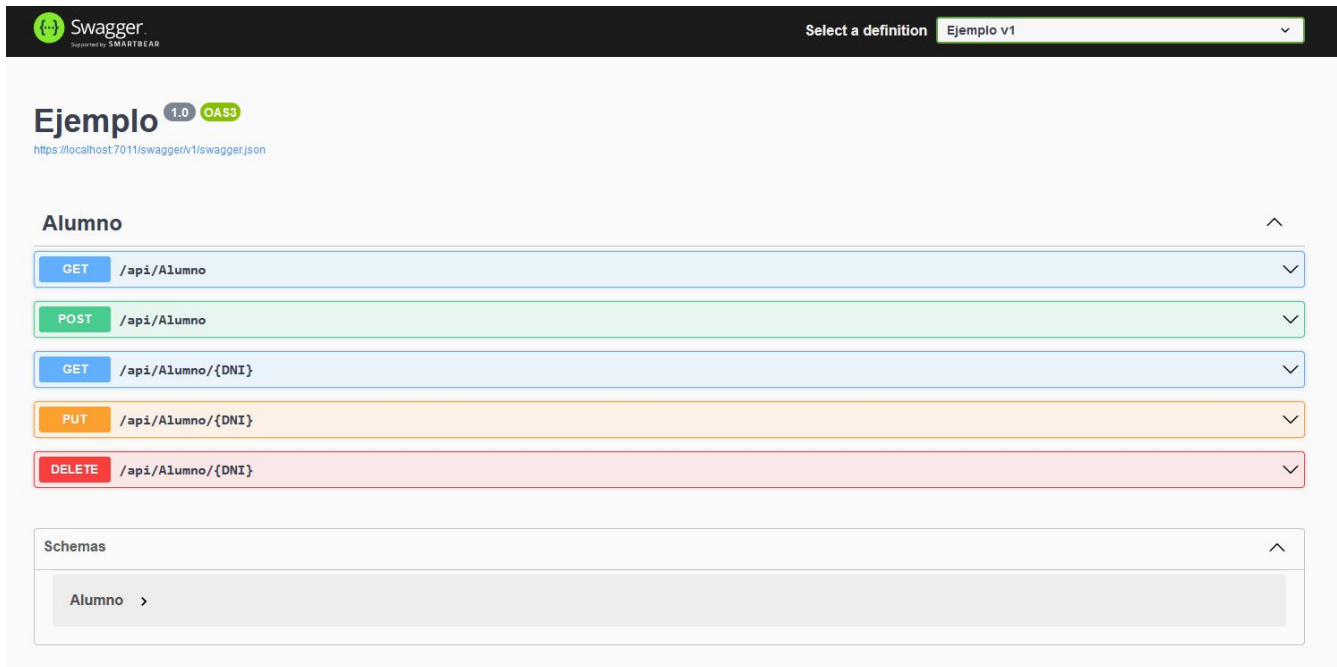
El resultado debería ser el siguiente teniendo en cuenta que los puertos sean otros.

```
1  {
2    "$schema": "https://json.schemastore.org/launchsettings.json",
3    "iisSettings": {
4      "windowsAuthentication": false,
5      "anonymousAuthentication": true,
6      "iisExpress": {
7        "applicationUrl": "http://localhost:11065",
8        "sslPort": 44323
9      }
10   },
11   "profiles": {
12     "Ejemplo": {
13       "commandName": "Project",
14       "dotnetRunMessages": true,
15       "launchBrowser": true,
16       "launchUrl": "swagger",
17       "applicationUrl": "https://localhost:7011;http://localhost:5013",
18       "environmentVariables": {
19         "ASPNETCORE_ENVIRONMENT": "Development"
20       }
21     },
22     "IIS Express": {
23       "commandName": "IISExpress",
24       "launchBrowser": true,
25       "launchUrl": "swagger",
26       "environmentVariables": {
27         "ASPNETCORE_ENVIRONMENT": "Development"
28       }
29     }
30   }
31 }
```

# Implementación de una API Rest usando Net Core

Ya podemos ejecutar la aplicación y debería funcionar la API conectada a la base de datos.

Debe de abrirse en el browser el API en Swager donde podremos probar los métodos que creamos en el controlador.



The screenshot displays the Swagger UI interface for an API named "Ejemplo". The top bar includes the Swagger logo, the text "Supported by SMARTBEAR", and a dropdown menu labeled "Select a definition" with "Ejemplo v1" selected. Below the header, the API title "Ejemplo" is shown with a "1.0" version tag and an "OAS3" specification tag. The URL "https://localhost:7011/swagger/v1/swagger.json" is listed. The main section, titled "Alumno", contains a list of five API endpoints, each with a colored button indicating the HTTP method: GET (blue), POST (green), GET (blue), PUT (orange), and DELETE (red). The endpoints are: "/api/Alumno", "/api/Alumno", "/api/Alumno/{DNI}", "/api/Alumno/{DNI}", and "/api/Alumno/{DNI}". A "Schemas" section at the bottom shows a single schema named "Alumno" with a right-pointing arrow.

Swagger  
Supported by SMARTBEAR

Select a definition Ejemplo v1

**Ejemplo** 1.0 OAS3  
<https://localhost:7011/swagger/v1/swagger.json>

**Alumno**

- GET /api/Alumno
- POST /api/Alumno
- GET /api/Alumno/{DNI}
- PUT /api/Alumno/{DNI}
- DELETE /api/Alumno/{DNI}

Schemas

Alumno >

# Implementación de una API Rest usando Net Core

Si entramos al primer método: GET, vamos a poder ver la estructura del json para esta clase, y cuando presionamos Try It Out y Execute nos debe de retornar los valores de la tabla:

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7011/api/Alumno' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:7011/api/Alumno
```

Server response

Code Details

200

Response body

```
{
  "dni": "27890765",
  "apellidoNombre": "García Eliseo",
  "email": "geliseo@frro.utn.edu.ar",
  "fechaNacimiento": "1981-01-10T00:00:00",
  "notaPromedio": 8.8
},
{
  "dni": "28675888",
  "apellidoNombre": "Lopez Stefania",
  "email": "slopez@frro.utn.edu.ar",
  "fechaNacimiento": "1982-05-20T00:00:00",
  "notaPromedio": 7.8
},
{
  "dni": "29345777",
  "apellidoNombre": "Wingdam Raul",
  "email": "urraul@frro.utn.edu.ar",
  "fechaNacimiento": "1983-03-01T00:00:00",
  "notaPromedio": 9.33
},
{
  "dni": "30425782",
  "apellidoNombre": "Losteau Pedro",
  "email": "plosteau@frro.utn.edu.ar",
  "fechaNacimiento": "1982-08-25T00:00:00",
  "notaPromedio": 4.5
}
```

Response headers

```
content-type: application/json; charset=utf-8
date: Tue, 26 Sep 2023 23:29:47 GMT
server: Kestrel
x-firefox-spdy: h2
```

Download



# Implementación de una API Rest usando Net Core

---

Ahora puedes probar los otros métodos, modificar, insertar nuevos y borrar registros.

Con esto, hemos creado una API REST básica con C# en .NET Core 7.0 que utiliza una base de datos SQL Server para almacenar y recuperar información.