



Introducción a API RESTFuI

Aplicaciones Web

Introducción

En esta semana se revisará los siguientes temas

- Teoría
- Service-oriented architecture (SOA).
- Servicios Web.
- SOAP.

- REST.
- REST : Representational State Transfer
- Buenas practicas servicios RESTFul •
Comparativa entre Net Framework Vs Net Core.
- Tecnología
- .NET Core
- ASP.NET Core
- ASP.NET Core Web API
- Entity Framework Core
- Instalación
- Demo

Teoría

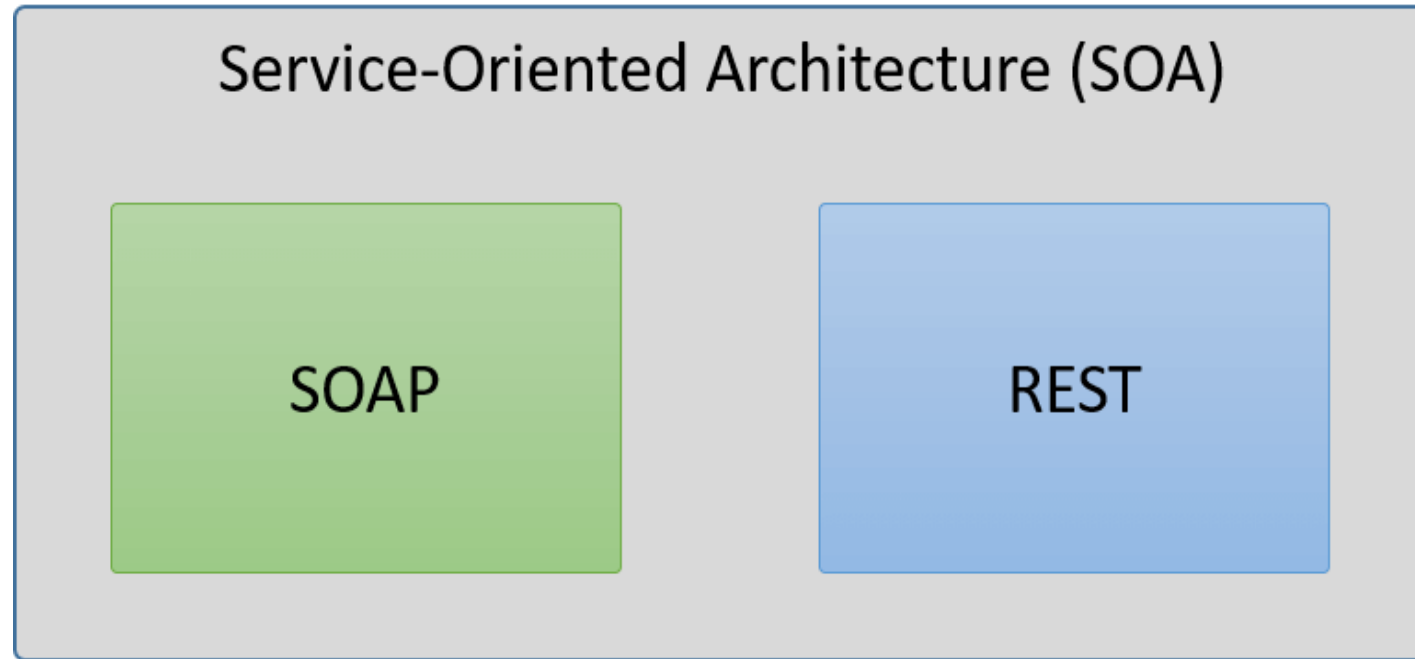
Service-oriented architecture (SOA)

- Es un tipo de Arquitectura de Software y no una tecnología o producto. SOA es una arquitectura que se base en la integración de aplicaciones mediante Servicios, los servicios representan la medida más granular de la arquitectura, sobre la que se construyen otros artefactos como: composiciones, proxys, fachadas, BPM e incluso API completas.
- El activo principal que ofrece SOA son los **Contratos** que expone y es que con una definición correcta de un contrato podríamos incluso cambiar el aplicativo por otro y nadie se diera cuenta. Un contrato define la forma en que se realizara la comunicación, establece el formato de entrada del servicio y de salida lo que ayuda a que cualquier cliente pueda interpretarlos sin ningún problema.
- Otra de las ventajas que ofrece SOA es que utiliza protocolos de comunicación estándar como WebServices lo cual le permite comunicarse con cualquier aplicación sin importar en que lenguaje este desarrollada.
- Cada servicio que se desarrolla en SOA se realiza bajo la premisa de que sera reutilizado al máximo y que será de utilidad para otros sistemas por lo cual cada servicio deberá ser planeado con cuidado para que sea lo más simple posible y a si aumentar el grado de re-utilización que tendrá.
- Finalmente indicar es que **SOA no es igual que SOAP y tener Web Services no significa que tenemos SOA.**

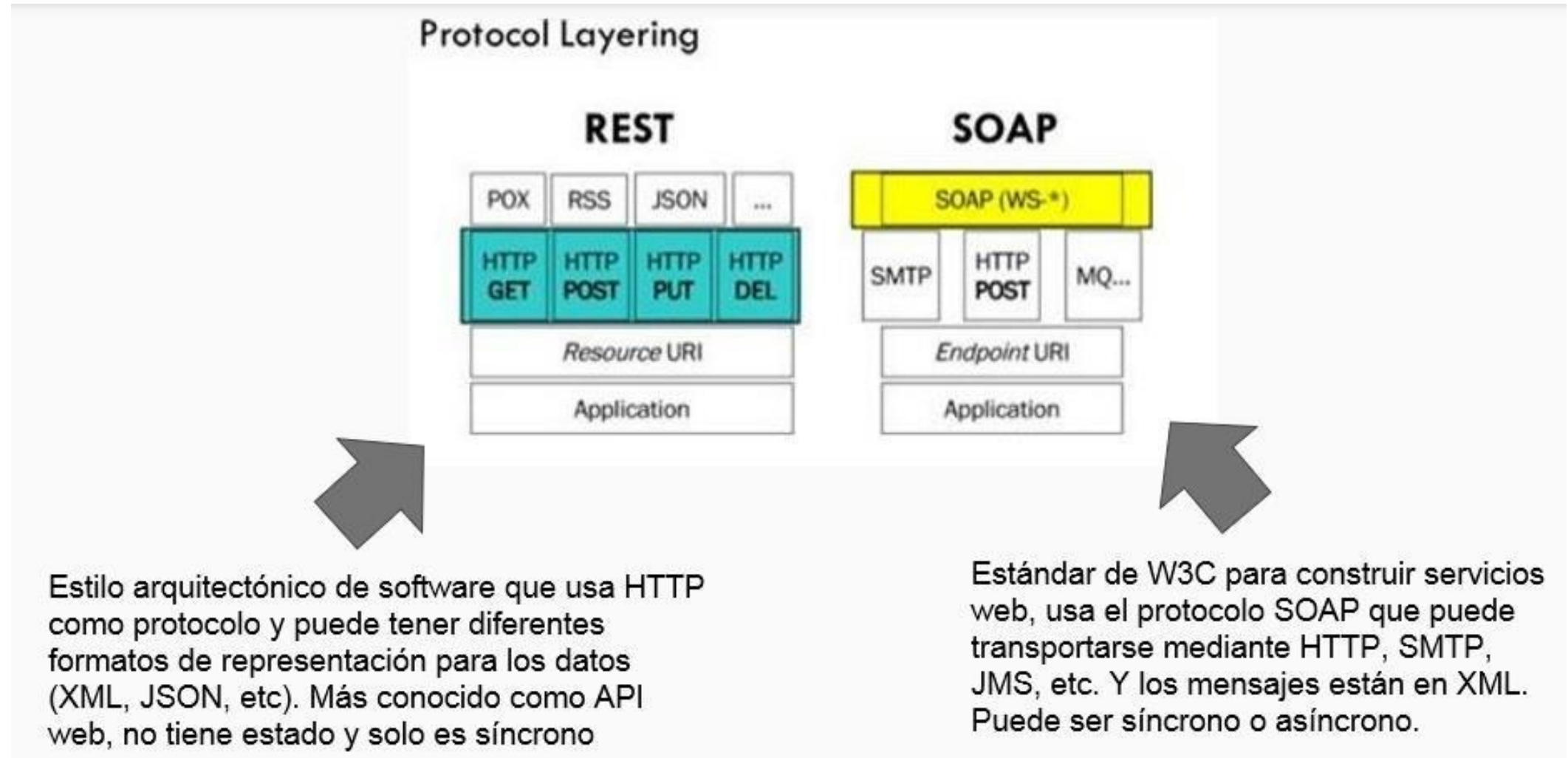
Servicios Web



Servicios Web



Servicios Web

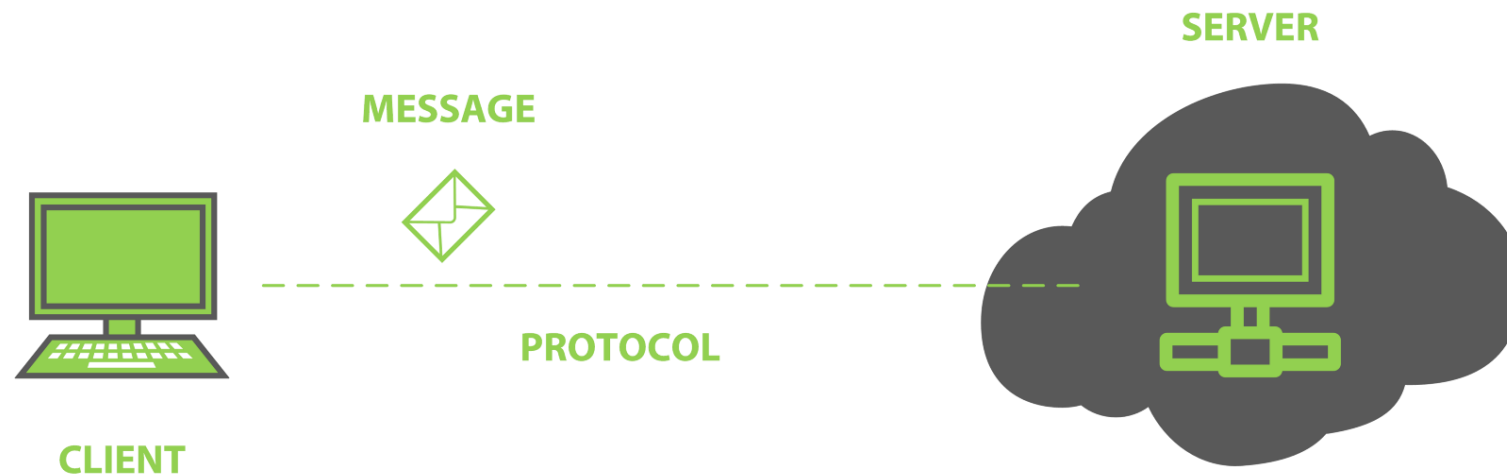


REST-Representational State Transfer.



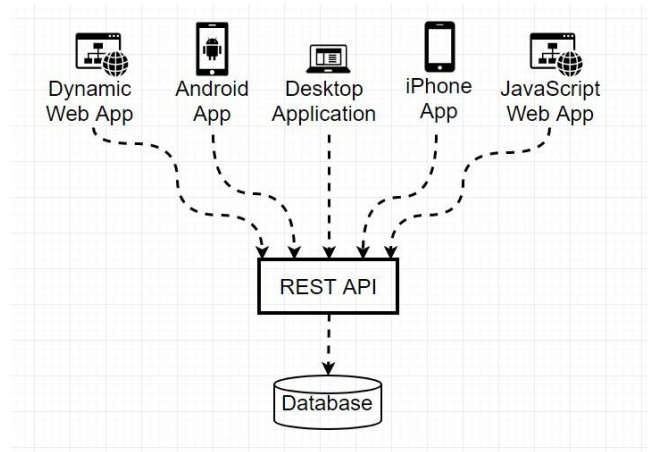
REST-Representational State Transfer.

- REST es un estilo de arquitectura de software. Como se describe en una disertación de Roy Fielding, REST es un "estilo arquitectónico" que básicamente explota la tecnología y los protocolos existentes de la Web.



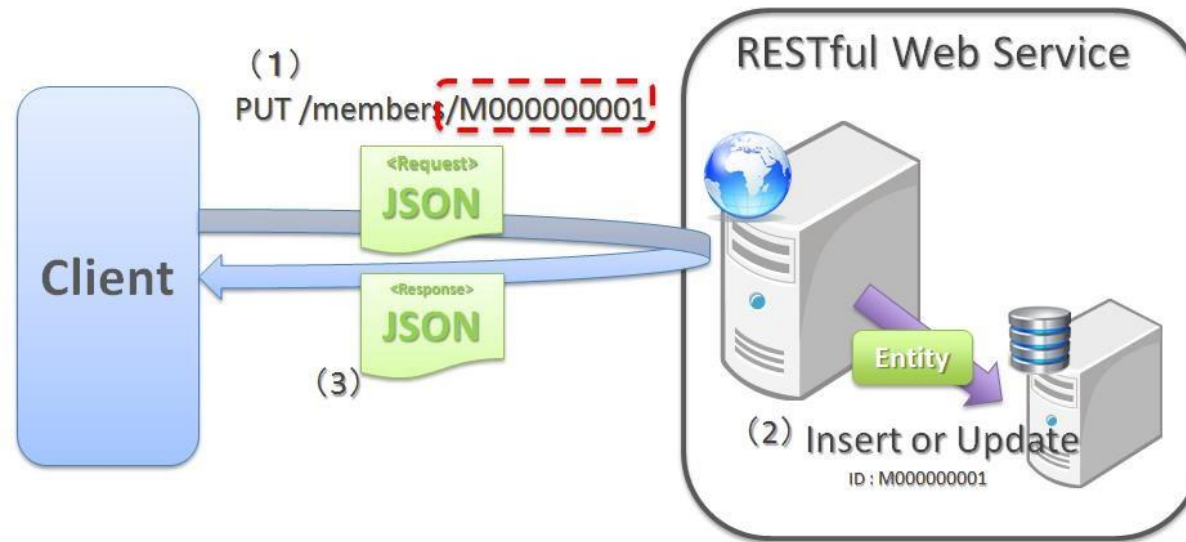
API REST.

- Una interfaz de programación de aplicaciones (API) es un conjunto particular de reglas ('código') y especificaciones que los programas de software pueden seguir para comunicarse entre sí. Sirve como una interfaz entre diferentes programas de software y facilita su interacción, de manera similar a la forma en que la interfaz de usuario facilita la interacción entre humanos y computadoras.



Servicios Web RESTful

- RESTful se usa generalmente para referirse a servicios web que implementan la arquitectura REST.



Servicios Web RESTFul

Ejemplo

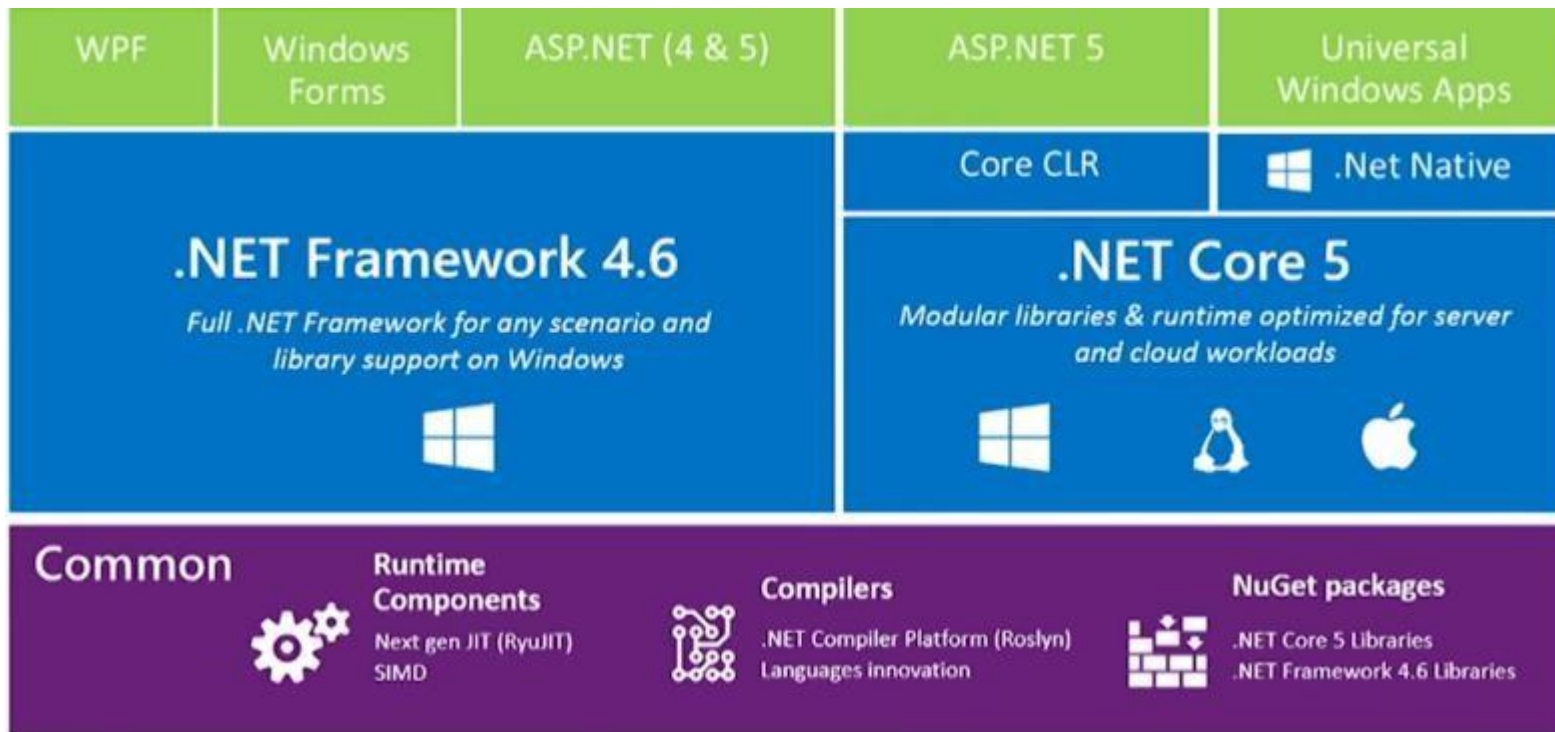
- GET / tickets: recupera una lista de tickets GET / tickets / 12 - Recupera un ticket específico POST / tickets: crea un nuevo ticket PUT / tickets / 12 - Actualiza ticket # 12 DELETE / tickets / 12 - Elimina el ticket # 12

Buenas practicas de diseño de servicios web

RESTFuI

- Validaciones.
- Manejo de Excepciones.
- [Documentación.](#)
- Pruebas de Software.
- [Modelo de Ricardson.](#)
- [Métodos de petición HTTP: GET/POST/PUT/DELETE](#)
- [Response Status.](#)

Comparativa entre Net Framework y Net Core



<https://docs.microsoft.com/es-es/dotnet/standard/choosing-core-framework-server>

Tecnología

.NET Core

- .NET Core es una versión nueva (ni tan nueva) de Microsoft que ha cambiado su panorama con respecto al desarrollo de las aplicaciones .NET ya que, esta ha sido diseñado bajo el concepto de que sea opensource y multiplataforma. Así es, podemos tener ahora aplicaciones .NET en Windows, Linux, Mac y/o aportar a la contribución del código mediante su repositorio en [GitHub](#).
- A diferencia de NET Framework, este último es modular y se descarga mediante el nuget centrado sus características en paquetes separados del nuget lo cual permite hacerlo más escalable y usar lo que queramos en el desarrollo de nuestras aplicaciones.

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

.NET Core

¿Cuándo usar NET Core?

- Cuando queramos aplicaciones multiplataformas,
- Para mejorar la escalabilidad de nuestras aplicaciones. Pero con NET Core todo se vuelve más escalable,
- Tener como objetivo el desarrollo de microservicios,
- Queramos trabajar con contenedores de Docker.

.NET Core

¿Cuándo no usarlo?

- Cuando tengamos aplicaciones en producción basadas en el framework anterior, migrar todo es un costo muy alto para el equipo de desarrollo y la empresa en general,
- Si es que dependes de librerías, features exclusivos de la versión anterior y no se encuentra en este.

.NET Core

¿Qué tenemos disponible con NET Core?

Basta con abrir el Visual Studio y ver lo que hay disponible para comenzar como proyecto nuevo:

- Aplicaciones de Consola
- Bibliotecas de clase
- Proyectos Unitarios
- Proyecto ASP.NET Core (Aplicaciones Web)

ASP.NET Core

- ASP.NET Core es un rediseño de ASP.NET con cambios arquitectónicos que dan como resultado un framework más compacto y modular. Al ser ASP.NET Core un framework de código abierto, multiplataforma y de alto rendimiento, nos permite construir aplicaciones web modernas, conectadas a internet y basadas en la nube.

<https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2>

<https://dotnet.microsoft.com/learn/web/what-is-aspnet-core>

ASP.NET Core Web API

- Es un Framework que forma parte de ASP.NET MVC y que permite construir APIs habilitadas para REST. Las APIs habilitadas para REST ayudan a que sistemas externos utilicen la lógica de negocios implementada en una aplicación, incrementando la reutilización de dicha lógica.

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-2.2&tabs=visual-studio>

Entity Framework Core

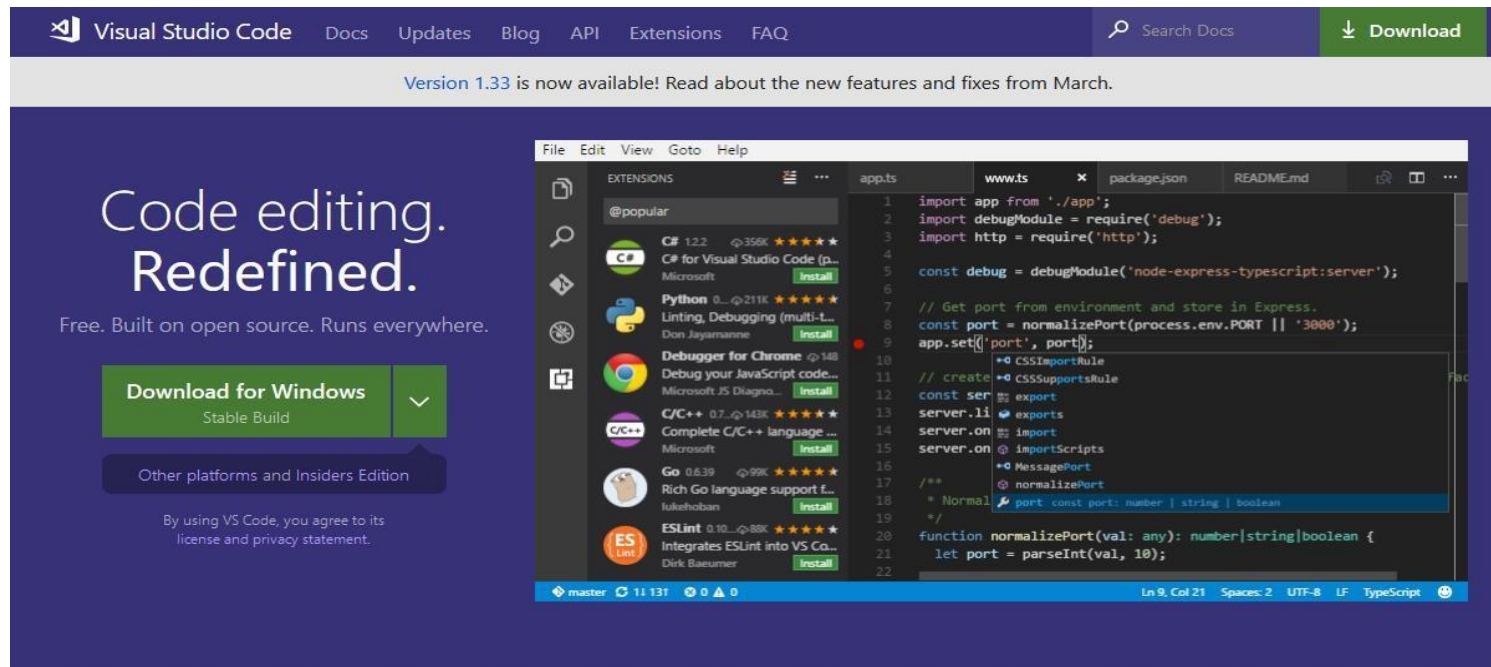
- Entity Framework Core es un framework ORM de código abierto que facilita y reduce considerablemente el tiempo de desarrollo de código de acceso a datos en aplicaciones .NET.
- Entity Framework representa una abstracción de ADO.NET que al mismo tiempo que reduce el tiempo de desarrollo, puede disminuir el rendimiento cuando no es utilizado de forma correcta siguiendo buenas prácticas.

<https://docs.microsoft.com/en-us/ef/core/>

Instalación

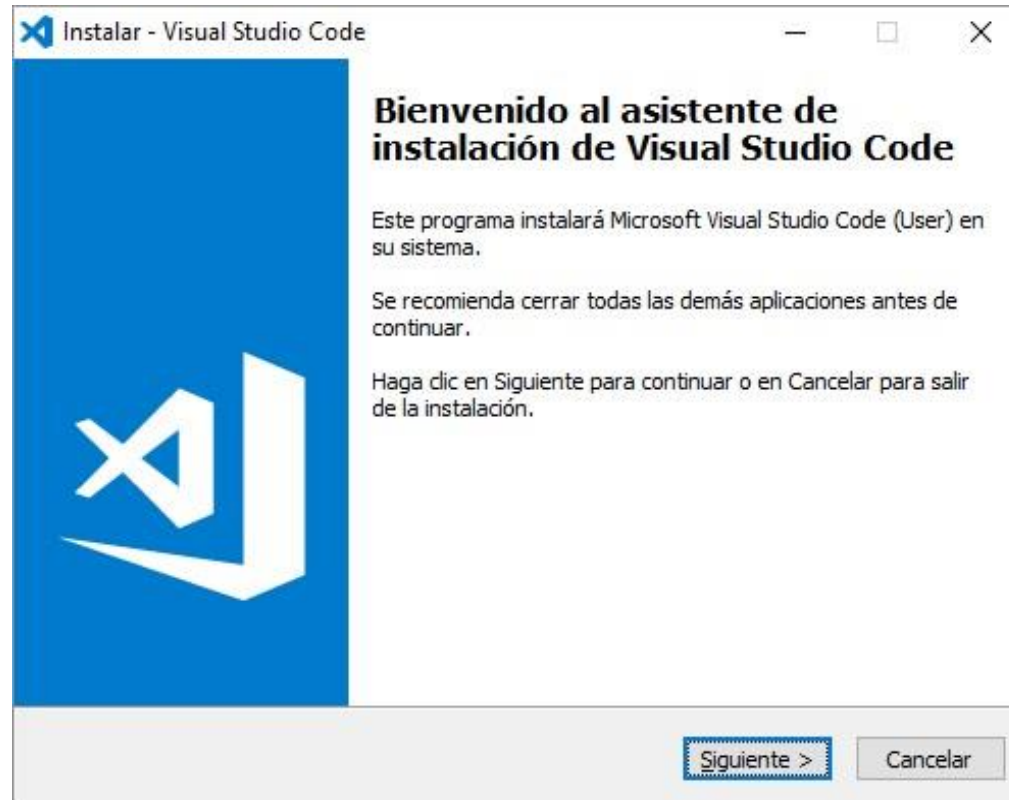
Instalación

- Descargamos Visual Studio Code <https://code.visualstudio.com/>



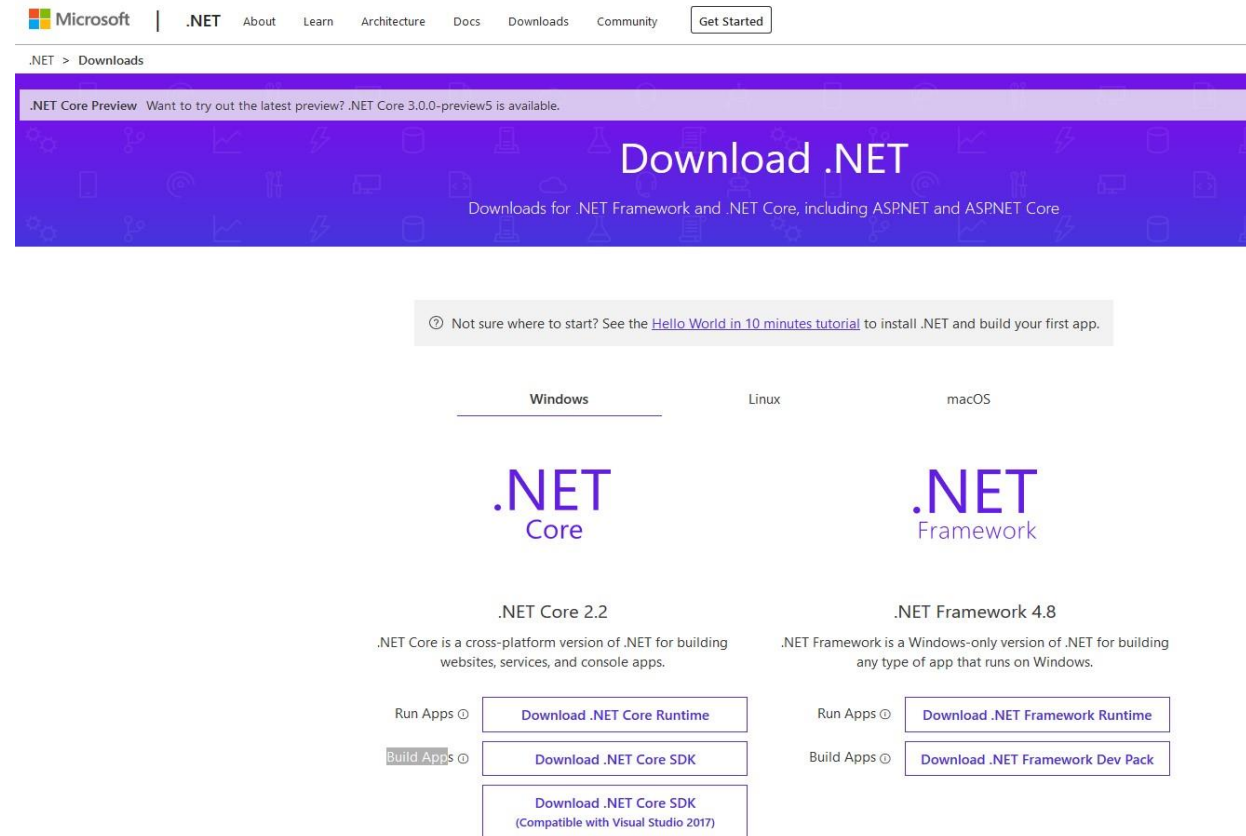
- Damos doble clic en el instalador VSCodeUserSetup-x64-1.33.1 para iniciar la instalación.

Instalación



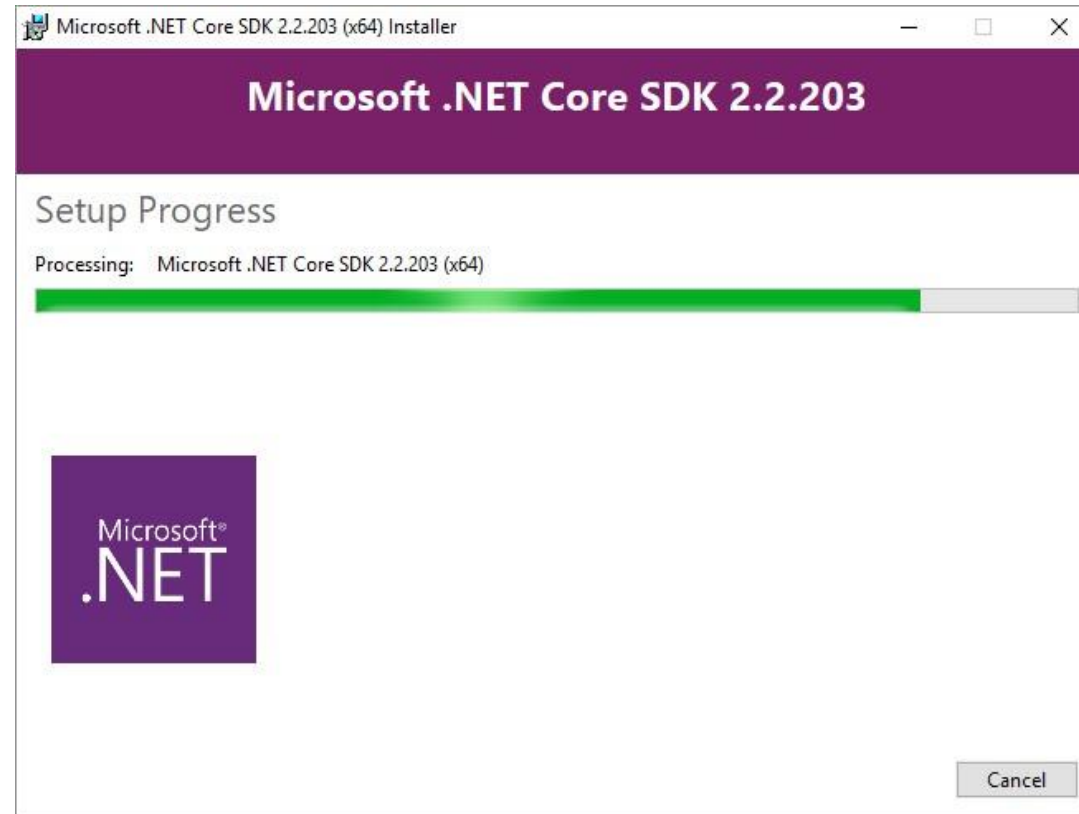
- Descargamos Net Core SDK <https://dotnet.microsoft.com/download>

Instalación



- Damos doble clic en el instalador dotnet-sdk-2.2.203-win-x64 para iniciar la instalación.

Instalación



Instalación

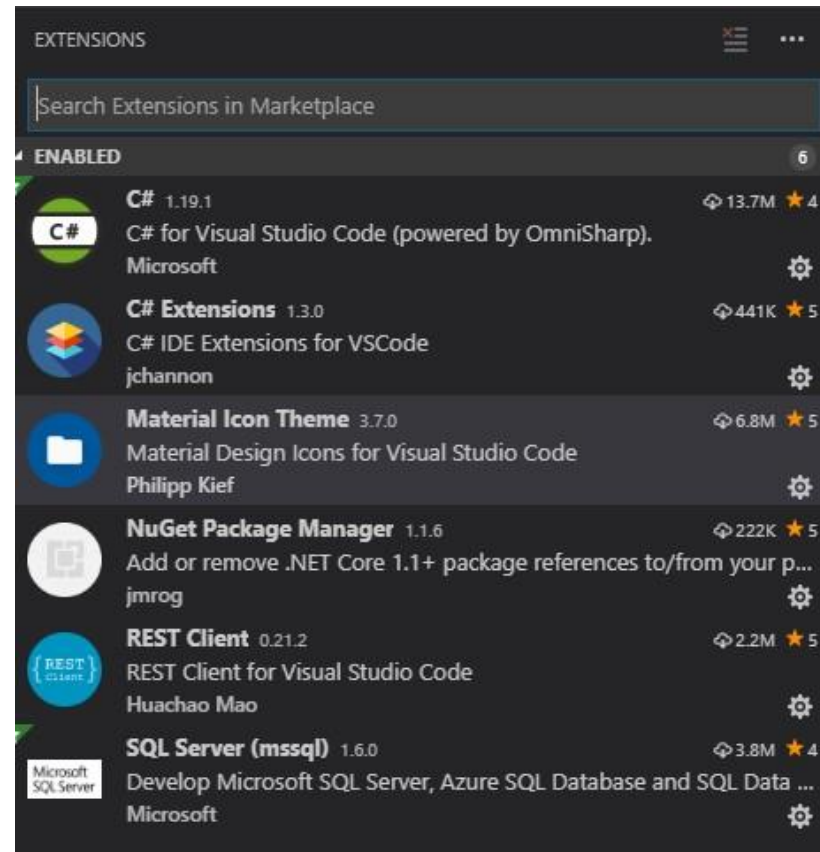
- Abrimos consola del sistema y escribimos el siguiente comando para confirmar la correcta instalación de net core.

```
C:\  
λ dotnet --version  
2.2.203
```

Instalación

- Instalar las siguientes extensiones en Visual Studio Code.

Demo



Demo

- Creamos la solución con el nombre de Event utilizando el siguiente comando. **dotnet new sln -n Event**

```
USUARIO@DESKTOP-MV5A26M /c/demo  
λ dotnet new sln -n Event  
La plantilla "Solution File" se creó correctamente.
```

Demo

- Creamos el proyecto tipo Web Api con el nombre de Event.Api utilizando el siguiente comando.
dotnet new webapi -n Event.Api

```
USUARIO@DESKTOP-MV5A26M /c/demo
λ dotnet new webapi -n Event.Api
La plantilla "ASP.NET Core Web API" se creó correctamente.

Procesando acciones posteriores...
Ejecutando "dotnet restore" en Event.Api\Event.Api.csproj...
  Restauración realizada en 1,42 sec para C:\demo\Event.Api\Event.Api.csproj.

Restauración correcta.
```

- Creamos el proyecto tipo Class Library con el nombre de Event.Domain utilizando el siguiente comando. **dotnet new classlib -n Event.Domain**

Demo

```
USUARIO@DESKTOP-MV5A26M /c/demo
λ dotnet new classlib -n Event.Domain
La plantilla "Class library" se creó correctamente.

Procesando acciones posteriores...
Ejecutando "dotnet restore" en Event.Domain\Event.Domain.csproj...
  Restauración realizada en 132,99 ms para C:\demo\Event.Domain\Event.Domain.csproj.
```

- Creamos el proyecto tipo Class Library con el nombre de Event.Repository utilizando el siguiente comando. **dotnet new classlib -n Event.Repository**

```
USUARIO@DESKTOP-MV5A26M /c/demo
λ dotnet new classlib -n Event.Repository
La plantilla "Class library" se creó correctamente.

Procesando acciones posteriores...
Ejecutando "dotnet restore" en Event.Repository\Event.Repository.csproj...
  Restauración realizada en 138,19 ms para C:\demo\Event.Repository\Event.Repository.csproj.

Restauración correcta.
```

Demo

- Creamos el proyecto tipo Class Library con el nombre de Event.Service utilizando el siguiente comando. **dotnet new classlib -n Event.Service**

```
USUARIO@DESKTOP-MV5A26M /c/demo
λ dotnet new classlib -n Event.Service
La plantilla "Class library" se creó correctamente.

Procesando acciones posteriores...
Ejecutando "dotnet restore" en Event.Service\Event.Service.csproj...
  Restauración realizada en 125,2 ms para C:\demo\Event.Service\Event.Service.csproj.

Restauración correcta.
```

- Establecer la referencia de Event.Repository a Event.Domain

**dotnet add Event.Repository/Event.Repository.csproj reference
Event.Domain/Event.Domain.csproj**

Demo

```
USUARIO@DESKTOP-MV5A26M /c/demo
λ dotnet add Event.Repository/Event.Repository.csproj reference Event.Domain/Event.Domain.csproj
Se ha agregado la referencia "..\Event.Domain\Event.Domain.csproj" al proyecto.
```

- Establecer la referencia de Event.Service a Event.Domain y Event.Service a Event.Repository
dotnet add Event.Service/Event.Service.csproj reference Event.Repository/Event.Repository.csproj

```
USUARIO@DESKTOP-MV5A26M /c/demo
λ dotnet add Event.Service/Event.Service.csproj reference Event.Repository/Event.Repository.csproj
Se ha agregado la referencia "..\Event.Repository\Event.Repository.csproj" al proyecto.
```

dotnet add Event.Service/Event.Service.csproj reference

```
USUARIO@DESKTOP-MV5A26M /c/demo
λ dotnet add Event.Service/Event.Service.csproj reference Event.Domain/Event.Domain.csproj
Se ha agregado la referencia "..\Event.Domain\Event.Domain.csproj" al proyecto.
```

Event.Domain/Event.Domain.csproj

Demo

- Establecer la referencia de Event.Api a Event.Domain y Event.Api a Event.Service **dotnet add Event.Api/Event.Api.csproj reference Event.Domain/Event.Domain.csproj**

```
USUARIO@DESKTOP-MV5A26M /c/demo  
λ dotnet add Event.Api/Event.Api.csproj reference Event.Domain/Event.Domain.csproj  
Se ha agregado la referencia "..\Event.Domain\Event.Domain.csproj" al proyecto.
```

dotnet add Event.Api/Event.Api.csproj reference Event.Service/Event.Service.csproj

- Agregar proyectos a la solución **dotnet sln Event.sln add Event.Api/Event.Api.csproj**

```
USUARIO@DESKTOP-MV5A26M /c/demo  
λ dotnet add Event.Api/Event.Api.csproj reference Event.Service/Event.Service.csproj  
Se ha agregado la referencia "..\Event.Service\Event.Service.csproj" al proyecto.
```

Event.Repository/Event.Repository.csproj

Event.Domain/Event.Domain.csproj Event.Service/Event.Service.csproj

Demo

```
USUARIO@DESKTOP-MV5A26M /c/demo
λ dotnet sln Event.sln add Event.Api/Event.Api.csproj Event.Repository/Event.Repository.csproj Event.Domain/Event.Domain.csproj Event.Service/Event.Service.csproj
Se ha agregado el proyecto "Event.Api\Event.Api.csproj" a la solución.
Se ha agregado el proyecto "Event.Repository\Event.Repository.csproj" a la solución.
Se ha agregado el proyecto "Event.Domain\Event.Domain.csproj" a la solución.
Se ha agregado el proyecto "Event.Service\Event.Service.csproj" a la solución.
```

- Compilar la solución. **dotnet build**

```
USUARIO@DESKTOP-MV5A26M /c/demo
λ dotnet build
Microsoft (R) Build Engine versión 16.0.450+ga8dc7f1d34 para .NET Core
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Restauración realizada en 35,22 ms para C:\demo\Event.Domain\Event.Domain.csproj.
Restauración realizada en 134,88 ms para C:\demo\Event.Repository\Event.Repository.csproj.
Restauración realizada en 134,88 ms para C:\demo\Event.Service\Event.Service.csproj.
Restauración realizada en 648,92 ms para C:\demo\Event.Api\Event.Api.csproj.
Event.Domain -> C:\demo\Event.Domain\bin\Debug\netstandard2.0\Event.Domain.dll
Event.Repository -> C:\demo\Event.Repository\bin\Debug\netstandard2.0\Event.Repository.dll
Event.Service -> C:\demo\Event.Service\bin\Debug\netstandard2.0\Event.Service.dll
Event.Api -> C:\demo\Event.Api\bin\Debug\netcoreapp2.2\Event.Api.dll

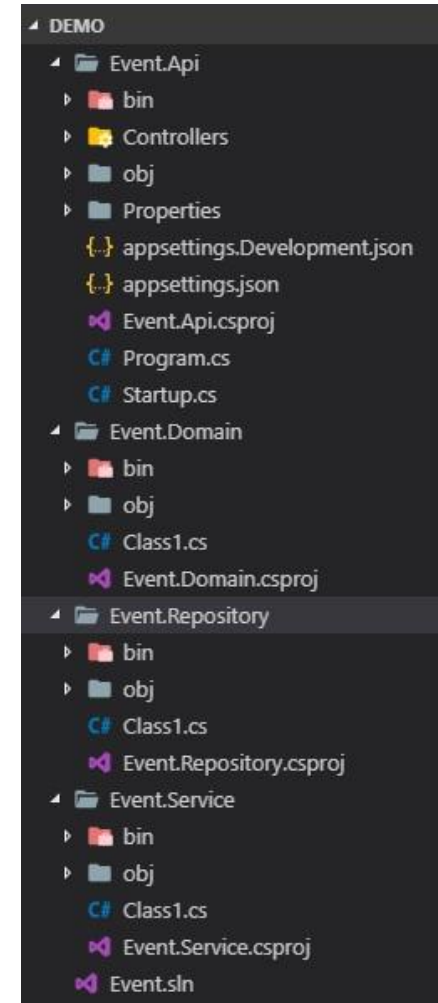
Compilación correcta.
    0 Advertencia(s)
    0 Errores

Tiempo transcurrido 00:00:05.83
```

Demo

```
USUARIO@DESKTOP-MV5A26M /c/demo  
λ code .
```

- Abrir la solución en Visual Studio Code utilizando el comando **code .**



Demo

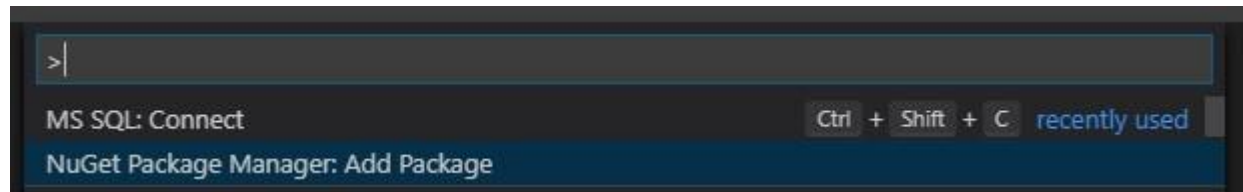
- Creamos la entidad Activity en el proyecto Event.Domain.

```
using System;

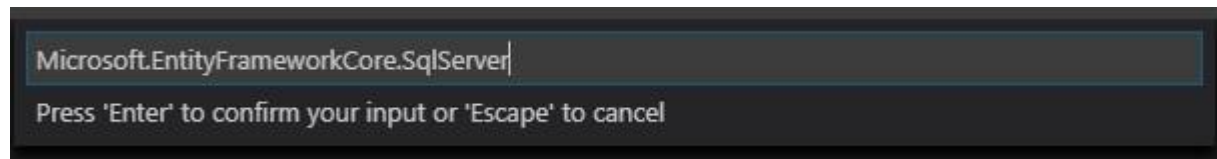
namespace Event.Domain
{
    0 references
    public class Activity
    {
        0 references
        public int Id { get; set; }
        0 references
        public string Local { get; set; }
        0 references
        public DateTime DateActivity { get; set; }
        0 references
        public string Theme { get; set; }
        0 references
        public int QuantityPeople { get; set; }
        0 references
        public string ImageUrl { get; set; }
    }
}
```

Demo

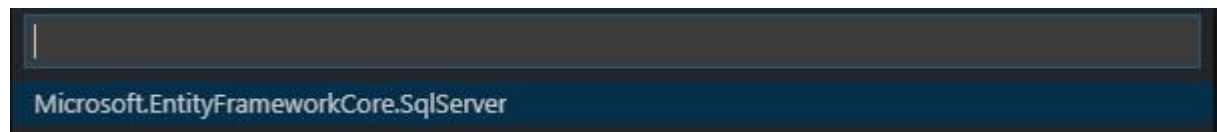
- Instalamos el package de Microsoft.EntityFrameworkCore.SqlServer.
- Presionamos las teclas Ctrl+Shift+P para elegir Nuget Package Manager.



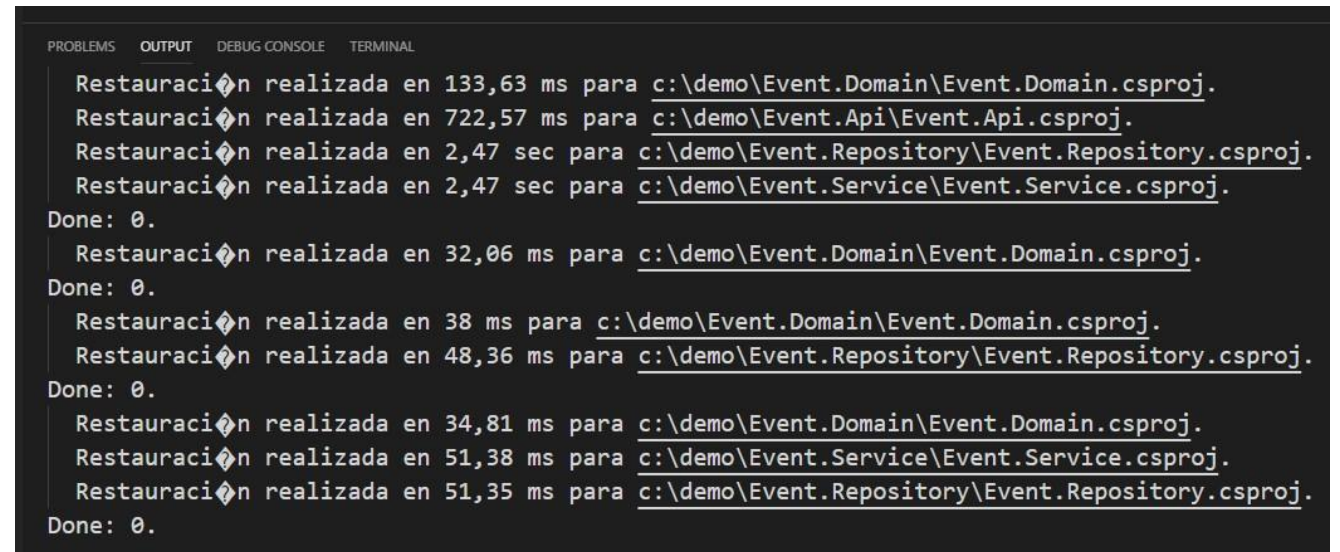
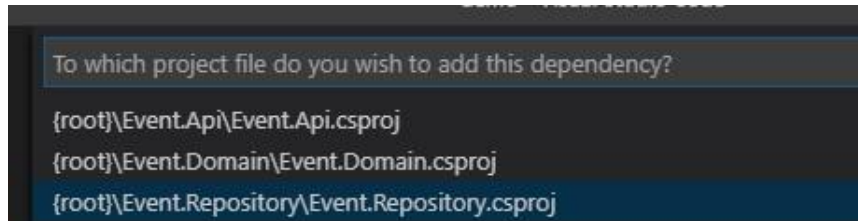
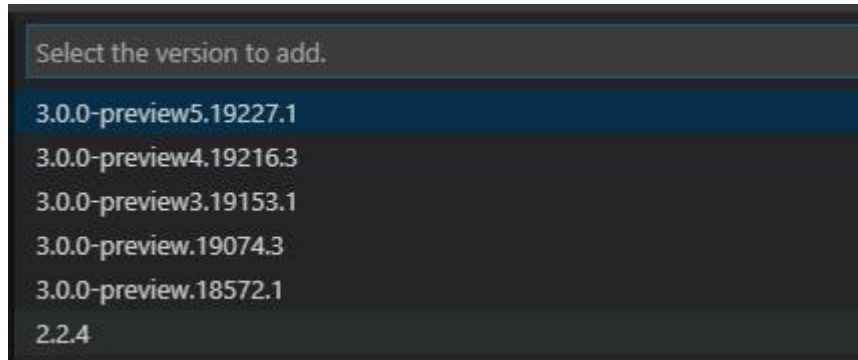
- Escribimos el Microsoft.EntityFrameworkCore.SqlServer.



- Elegimos la versión 2.2.4 y proyecto donde instalar.

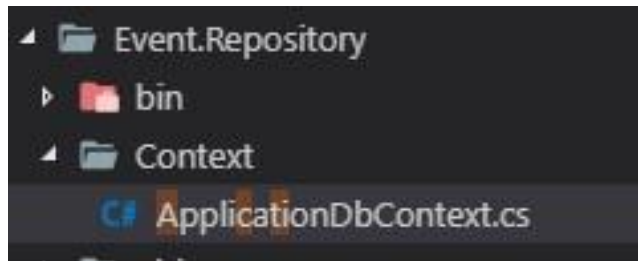


Demo



- Creamos el folder Context y dentro la clase ApplicationDbContext en el proyecto Event.Repository.

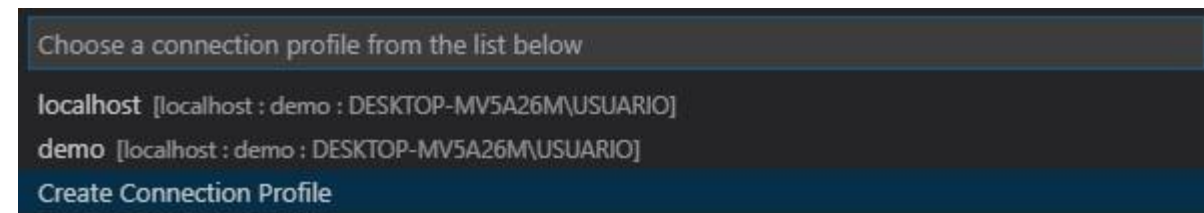
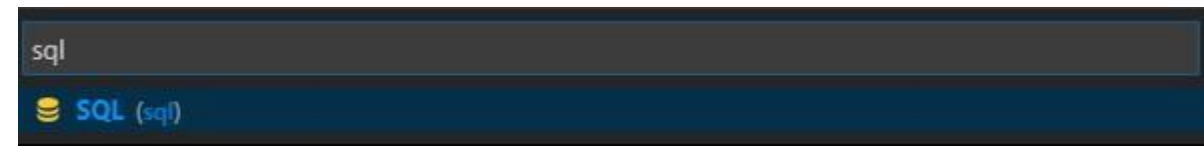
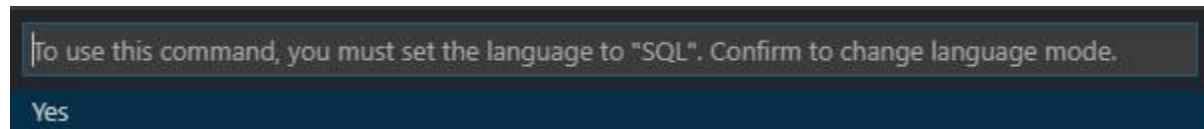
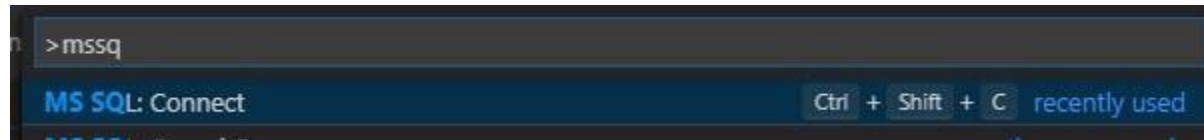
Demo



```
1 using Event.Domain;
2 using Microsoft.EntityFrameworkCore;
3
4 namespace Event.Repository.Context
5 {
6     1 reference
7     public class ApplicationDbContext: DbContext
8     {
9         0 references
10        public DbSet<Activity> Activities { get; set; }
11
12        0 references
13        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
14            : base(options)
15        {
16        }
17    }
18 }
```

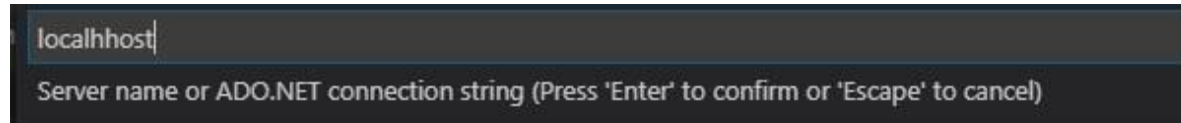
Demo

- Debemos crear una base de datos en SqlServer con el nombre de Event. Presionamos las teclas CTRL+Shift +P para realizar la conexión desde Visual Studio Code.

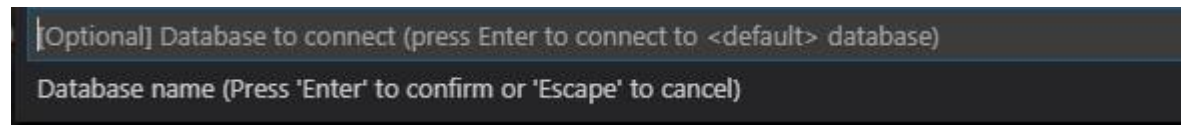


Demo

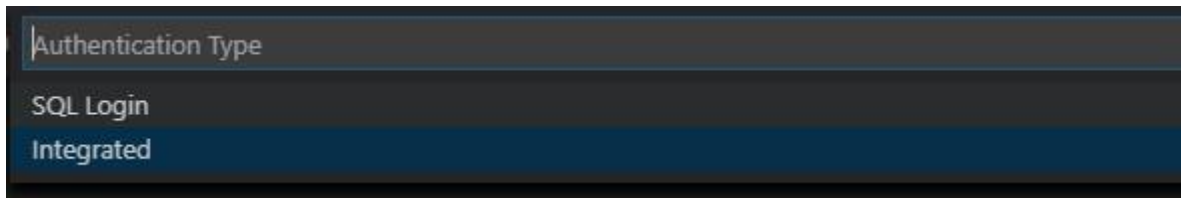
- Debemos crear una base de datos en SqlServer con el nombre de Event. Presionamos las teclas CTRL+Shift +P para realizar la conexión desde Visual Studio Code.



localhost
Server name or ADO.NET connection string (Press 'Enter' to confirm or 'Escape' to cancel)

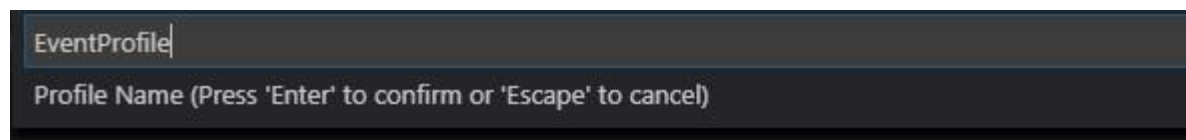


[Optional] Database to connect (press Enter to connect to <default> database)
Database name (Press 'Enter' to confirm or 'Escape' to cancel)



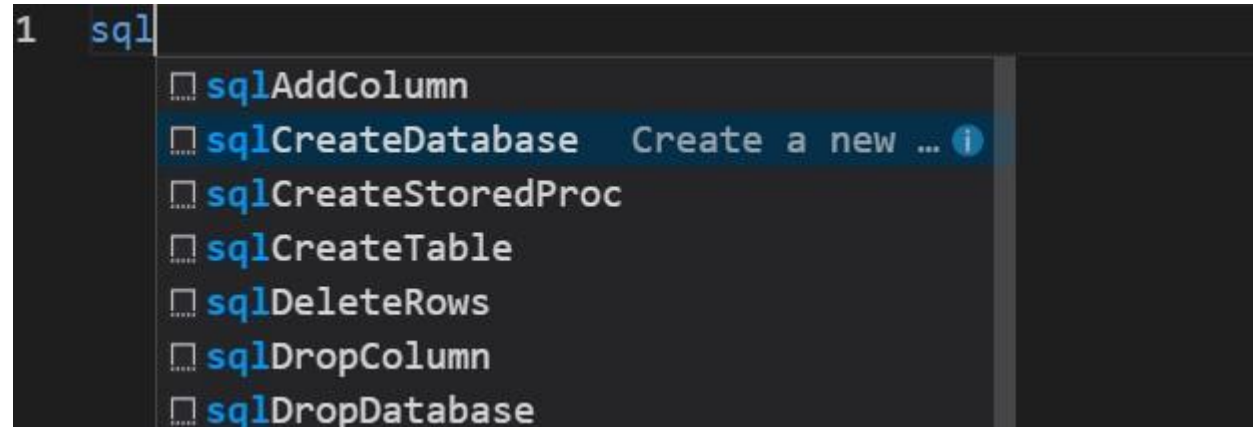
Authentication Type
SQL Login
Integrated

- Escribimos sql y del menú elegimos sqlCreateDatabase.

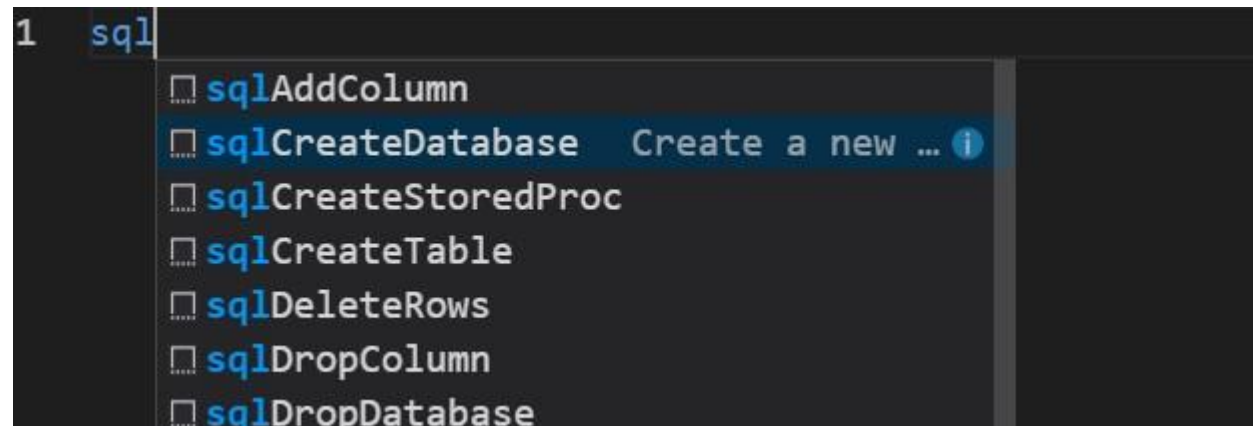


EventProfile
Profile Name (Press 'Enter' to confirm or 'Escape' to cancel)

Demo

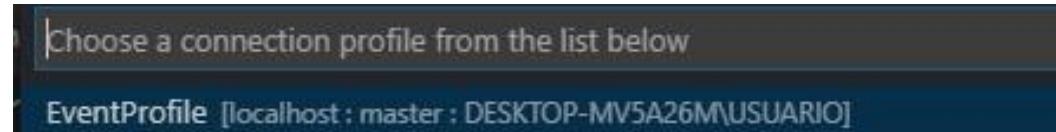


- Escribimos sql y del menú elegimos sqlCreateDatabase.



- Para ejecutar el script presionamos Ctrl+Shift+E, es probable que nos solicite el profile.

Demo



```
1  -- Create a new database called 'Event'
2  -- Connect to the 'master' database to run this sn
3  USE master
4  GO
5  -- Create the new database if it does not exist al
6  IF NOT EXISTS (
7      SELECT name
8      FROM sys.databases
9      WHERE name = N'Event'
10 )
11 CREATE DATABASE Event
12 GO
```

MESSAGES

[11:14:29] Started executing query at [Line 1](#)
Commands completed successfully.

[11:14:29] Started executing query at [Line 4](#)
Commands completed successfully.
Total execution time: 00:00:00.312

- Creamos la cadena de conexión en el archivo appsettings.json del proyecto Event.Api

Demo

```
1 {  
2   "connectionStrings": {  
3     "DefaultConnection": "Data Source=.;Initial Catalog=Event;Integrated Security=True"  
4   },  
}
```

- Debemos agregar el servicio de DbContext en el archivo Startup del proyecto Event.Api, donde además se indica el uso de la cadena de conexión.

```
public void ConfigureServices(IServiceCollection services)  
{  
    services.AddDbContext<ApplicationDbContext>(options =>  
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));  
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);  
}
```

Demo

- Ahora ejecutamos nuestra primera migración para lo cual debemos ubicarnos dentro del proyecto Event.Repository

dotnet ef --startup-project ../Event.Api migrations add init

```
USUARIO@DESKTOP-MV5A26M /c/demo/Event.Repository
λ dotnet ef --startup-project ../Event.Api migrations add init
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.2.4-servicing-10062 initialized 'ApplicationDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None
Done. To undo this action, use 'ef migrations remove'
```

- Ahora el siguiente comando para actualizar nuestra base de datos según la migración generada.

dotnet ef --startup-project ../Event.Api database update

Demo

```
USUARIO@DESKTOP-MV5A26M /c/demo/Event.Repository  
λ dotnet ef --startup-project ../Event.Api database update
```

Ciclo de vida de nuestras dependencias

- **Transient:** las dependencias son creadas cada vez que son solicitadas. Por ejemplo, en cada request de nuestra aplicación estas son creadas de nuevo. Es conveniente para liberar de la memoria cuando se dejen de usar.
- **Scoped:** a diferencia del transient, mientras que estamos en el request y solicitamos una dependencia esta será creada una sola vez hasta que el request muera.
- **Singleton:** se creará una sola dependencia para todo el ciclo de vida de nuestro proyecto o hasta que el servidor sea reiniciado. Ideal para inicializar clases con parámetros.
- El StartUp.cs es nuestra clase padre en la cual vamos a configurar el comportamiento de nuestra aplicación, para esto nos vamos a ir al método ConfigureServices y agregar nuestras dependencias.

```
services.AddTransient<IActivityRepository,ActivityRepository>();
```

Demo

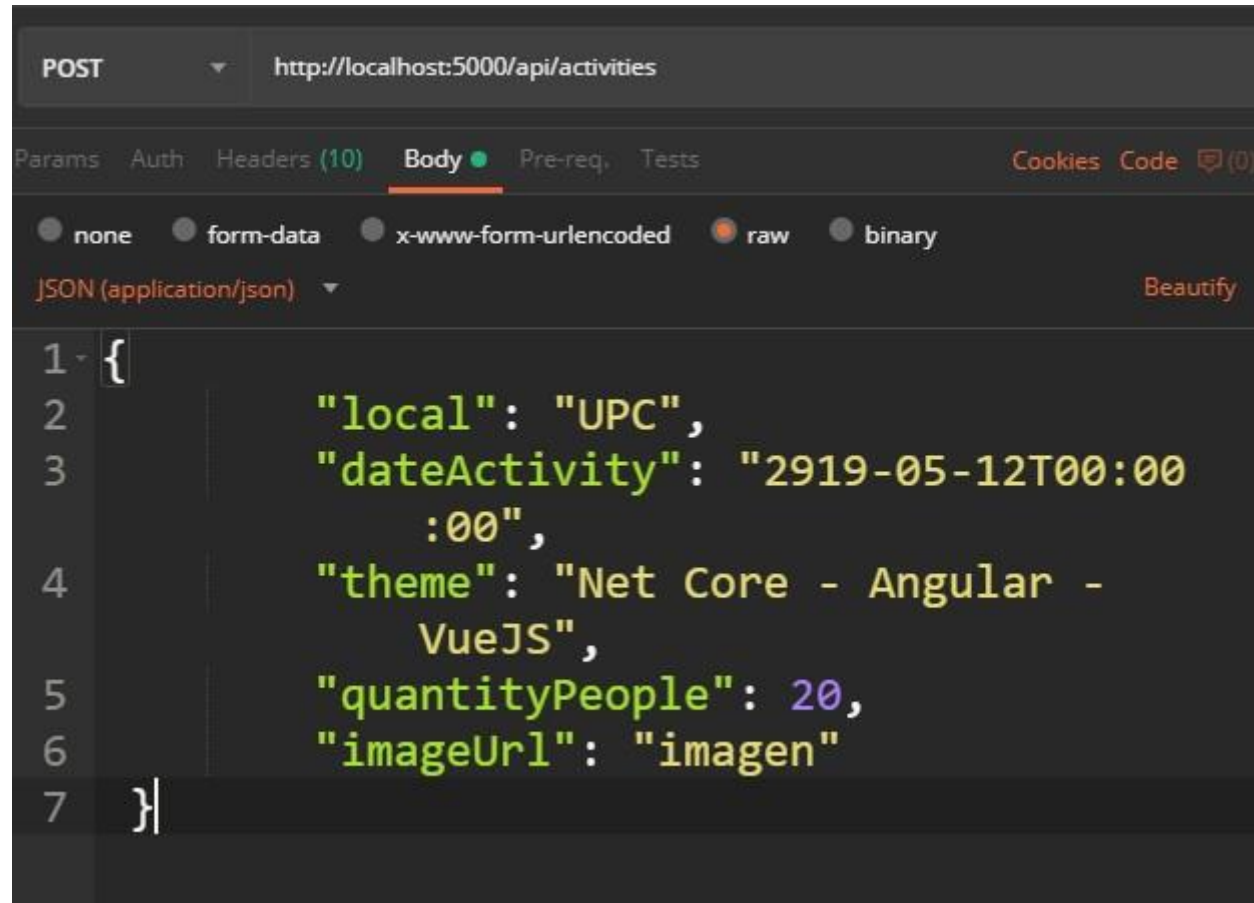
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddTransient<IActivityRepository, ActivityRepository>();

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
}
```

- El resto del código lo encuentran en el demo, probaremos el API.

Demo



The screenshot shows a REST client interface with a POST request to `http://localhost:5000/api/activities`. The 'Body' tab is selected, and the request body is a JSON object. The body is formatted with syntax highlighting and line numbers. The JSON object contains the following fields: `local` (UPC), `dateActivity` (2919-05-12T00:00:00), `theme` (Net Core - Angular - VueJS), `quantityPeople` (20), and `imageUrl` (imagen).

```
1 {  
2     "local": "UPC",  
3     "dateActivity": "2919-05-12T00:00:  
4         :00",  
5     "theme": "Net Core - Angular -  
6         VueJS",  
7     "quantityPeople": 20,  
8     "imageUrl": "imagen"  
9 }
```

Demo

```
Startup
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Hospital.Repositoy;
using Hospital.Repositoy.dbcontext;
using Hospital.Repositoy.implementation;
using Hospital.Service;
using Hospital.Service.implementation;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using Swashbuckle.AspNetCore.Swagger;

namespace Hospital.Api
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        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.AddDbContext<ApplicationDbContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("De
faultConnection")));

    services.AddTransient<IPacienteRepository,
PacienteRepository>();
    services.AddTransient<IPacienteService,
PacienteService>();

    services.AddTransient<IConsultaRepository,
ConsultaRepository>();
    services.AddTransient<IConsultaService,
ConsultaService>();

    services.AddMvc().SetCompatibilityVersion(CompatibilityVer
sion.Version_2_2);

    services.AddSwaggerGen(swagger =>
{
    var contact = new Contact() { Name =
SwaggerConfiguration.ContactName, Url =
SwaggerConfiguration.ContactUrl };
}
```

Demo

```
swagger.SwaggerDoc(SwaggerConfiguration.DocNameV1,
                    new Info
                    {
                        Title =
SwaggerConfiguration.DocInfoTitle,
                        Version =
SwaggerConfiguration.DocInfoVersion,
                        Description =
SwaggerConfiguration.DocInfoDescription,
                        Contact = contact
                    }
                    );

        });

        services.AddCors(options =>
        {
            options.AddPolicy("Todos",
                builder =>
builder.WithOrigins("*").WithHeaders("*").WithMethods("*")
);
        });

        //https://localhost:5001/swagger/index.html

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

```
            // Enable middleware to serve generated
Swagger as a JSON endpoint.
            app.UseSwagger();

            // Enable middleware to serve swagger-ui
(HTML, JS, CSS, etc.), specifying the Swagger JSON
endpoint.
            app.UseSwaggerUI(c =>
            {
                c.SwaggerEndpoint(SwaggerConfiguration.EndpointUrl,
SwaggerConfiguration.EndpointDescription);
            });

            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                // The default HSTS value is 30 days. You
may want to change this for production scenarios, see
https://aka.ms/aspnetcore-hsts.
                app.UseHsts();
            }
            app.UseCors("Todos");
            //app.UseHttpsRedirection();
            app.UseMvc();
        }
    }
}
```

Demo

Controller

```
using Hospital.Entity;
using Hospital.Service;
using Microsoft.AspNetCore.Mvc;

namespace Hospital.Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class PacienteController : ControllerBase
    {
        private IPacienteService pacienteService;

        public PacienteController(IPacienteService
pacienteService)
        {
            this.pacienteService = pacienteService;
        }

        [HttpGet]
        public ActionResult Get()
        {
            return Ok(
                pacienteService.GetAll()
            );
        }

        [HttpPost]
```

```
        public ActionResult Post([FromBody] Paciente
paciente)
        {
            return Ok(
                pacienteService.Save(paciente)
            );
        }

        [HttpPut]
        public ActionResult Put([FromBody] Paciente
paciente)
        {
            return Ok(
                pacienteService.Update(paciente)
            );
        }

        [HttpDelete("{id}")]
        public ActionResult Delete(int id)
        {
            return Ok(
                pacienteService.Delete(id)
            );
        }
    }
}
```


Demo

AppSettings

```
{
  "connectionStrings": {
    "DefaultConnection": "Data Source=.;Initial
Catalog=hospital;Integrated Security=True"
  },
```

Repository Impl

```
using System.Collections.Generic;
using System.Linq;
using Hospital.Entity;
using Hospital.Repository.dbcontext;
```

```
namespace Hospital.Repository.implementation
{
  public class PacienteRepository : IPacienteRepository
  {

    private ApplicationDbContext context;

    public PacienteRepository(ApplicationDbContext context)
    {
      this.context = context;
    }
  }
}
```

```
"Logging": {
  "LogLevel": {
    "Default": "Warning"
  }
},
"AllowedHosts": "*"
}
```

```
public Paciente Get(int id)
{
  var result = new Paciente();
  try
  {
    result = context.Pacientes.Single(x => x.Id == id);
  }

  catch (System.Exception)
  {

    throw;
  }
  return result;
}
```

Demo

```
public IEnumerable<Paciente> GetAll()
{
    var result = new List<Paciente>();
    try
    {
        result = context.Pacientes.ToList();
    }

    catch (System.Exception)
    {
        throw;
    }
    return result;
}
```

```
public bool Save(Paciente entity)
{
    try
    {
        context.Add(entity);
        context.SaveChanges();
    }
    catch (System.Exception)
    {

```

```
        return false;
    }
    return true;
}
```

```
public bool Update(Paciente entity)
{
    try
    {
        var pacienteOrigina = context.Pacientes.Single(
            x => x.Id == entity.Id
        );

        pacienteOrigina.Id=entity.Id;
        pacienteOrigina.Nombres=entity.Nombres;
        pacienteOrigina.Apellidos=entity.Apellidos;
        pacienteOrigina.Dni=entity.Dni;
        pacienteOrigina.Direccion=entity.Direccion;
        pacienteOrigina.Telefono=entity.Telefono;

        context.Update(pacienteOrigina);
        context.SaveChanges();
    }
    catch (System.Exception)
    {

```

Demo

```
        return false;
    }
    return true;
}

public bool Delete(int id)
{
    throw new System.NotImplementedException();
}
}
```