

CREAR LA API DE PIZZAS

Servicio RESTful mediante
la API de ASP.NET Core.
(net7.0)



Objetivo:

En esta unidad vamos a crear una **API** con **ASP.NET Core**.

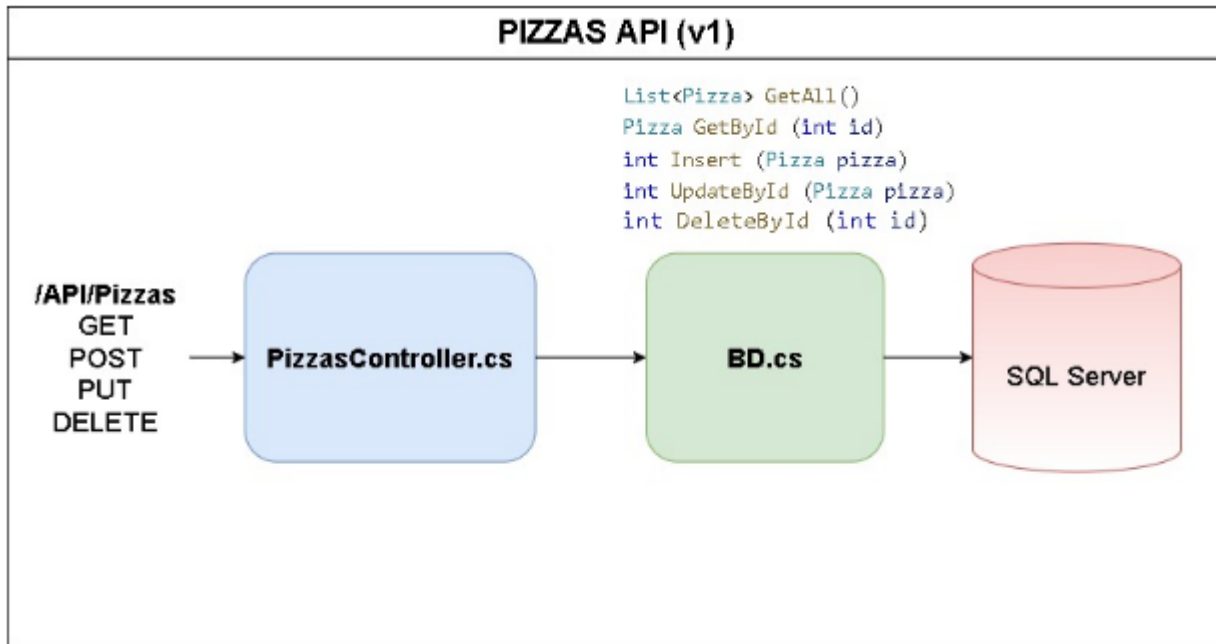
Luego, vamos a agregar una conexión a una base de datos **SQL Server** y un controller que se conecte a la misma.

Por último, vamos a implementar y testear todos los métodos creados en un front end.

A grandes rasgos, nuestro objetivo va a ser crear una API que implemente una lógica de **CRUD**.

Diagrama de cómo quedaría la API

Este es el diagrama de como va a quedar la API.



Entorno

Antes de comenzar es importante realizar estas tareas.

Leer la presentación de [CREAR UNA API REST WeatherForecast](#).

```
dotnet new webapi --no-https -n Pizzas.API
```

Verificar la versión del Visual Studio Code y actualizar a la última versión.

Menu → Help → Check for Updates...

Instalar las Extensiones: en VSCode abriendo una Terminal.

```
code --install-extension ms-dotnettools.csharp
code --install-extension esbenp.prettier-vscode
```

Instalar los paquetes necesarios: en VSCode abriendo una Terminal.

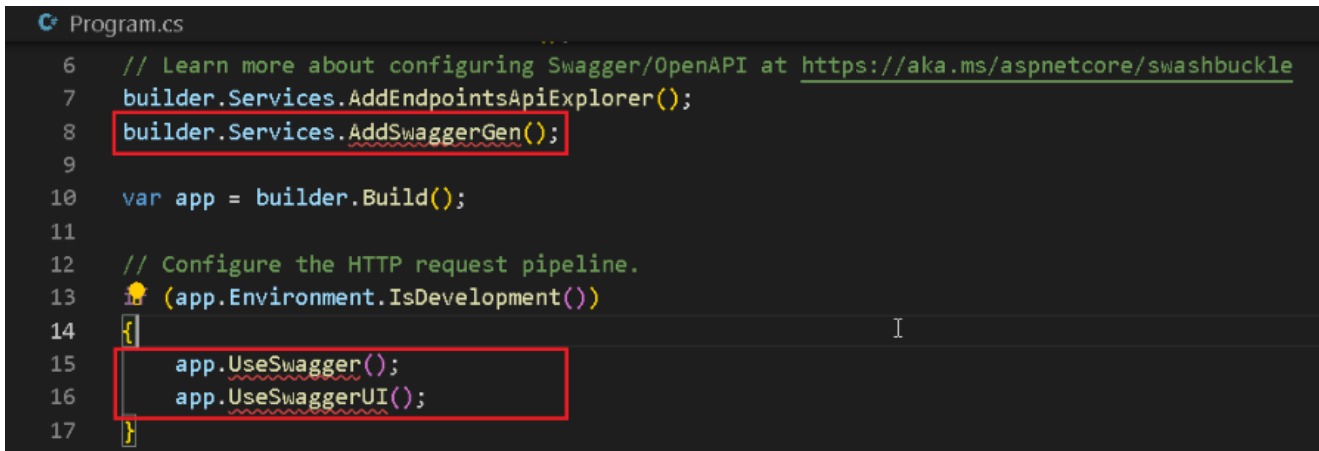
```
dotnet add package Dapper
dotnet add package System.Data.SqlClient
```

Antes de comenzar es importante realizar estas tareas.

Remover los paquetes innecesarios: en VSCode abriendo una Terminal.

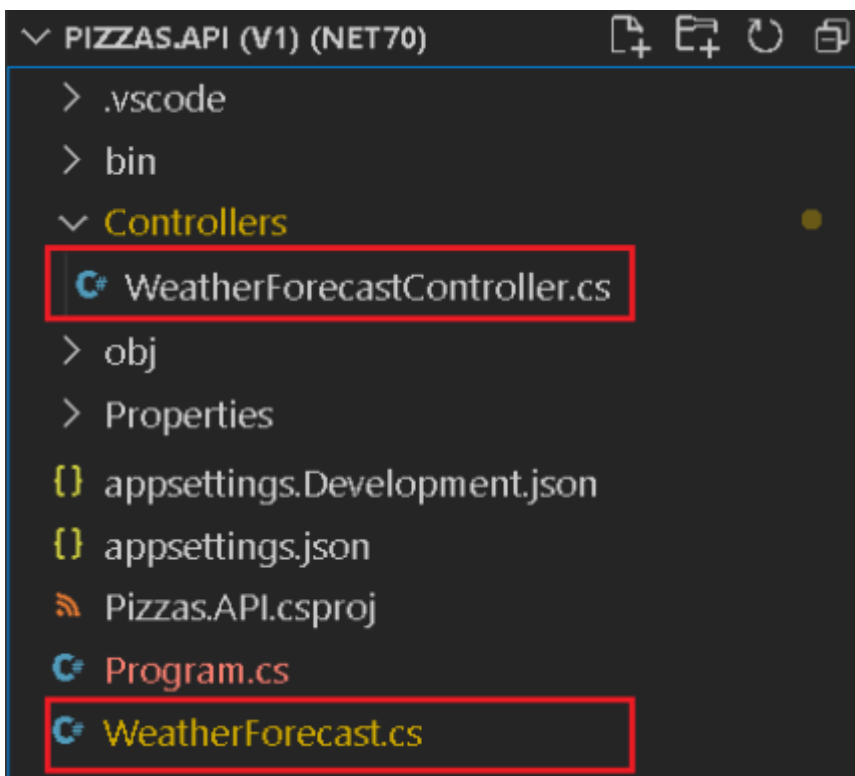
```
dotnet remove package Swashbuckle.AspNetCore
```

En el Program.cs eliminar las siguientes líneas.



```
6 // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
7 builder.Services.AddEndpointsApiExplorer();
8 builder.Services.AddSwaggerGen();
9
10 var app = builder.Build();
11
12 // Configure the HTTP request pipeline.
13 if (app.Environment.IsDevelopment())
14 {
15     app.UseSwagger();
16     app.UseSwaggerUI();
17 }
```

Lo primero es limpiar el código que se generó automáticamente. Para ello vamos a eliminar el controller y la clase correspondiente al Clima. Eliminar los archivos `WeatherForecastController.cs` y `WeatherForecast.cs`



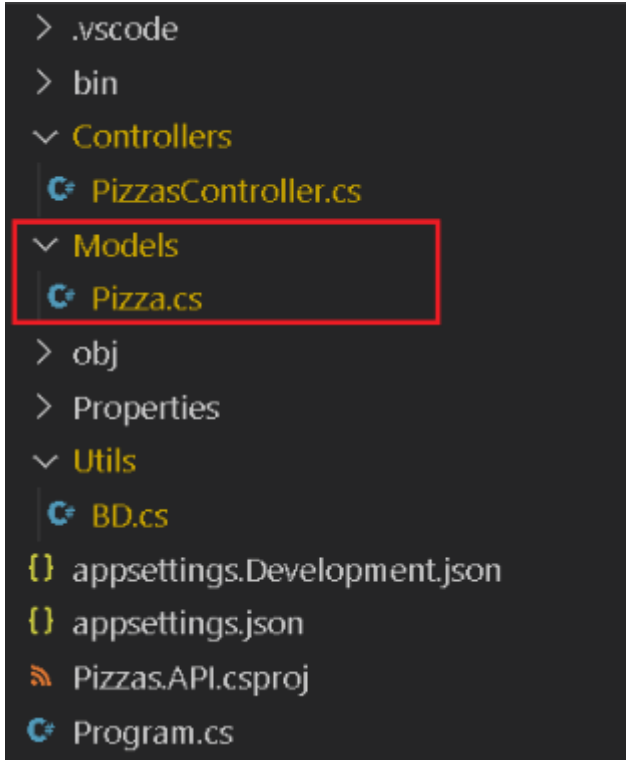
PIZZAS.API (V1) (NET70)

- > .vscode
- > bin
- > Controllers
 - WeatherForecastController.cs
- > obj
- > Properties
- { } appsettings.Development.json
- { } appsettings.json
- 🔗 Pizzas.API.csproj
- 🔗 Program.cs
- WeatherForecast.cs

Model

Debemos crear los modelos en la carpeta Models, que contendrá la información referente a una registro de la tabla Pizzas de la base de datos.

Model: Pizza

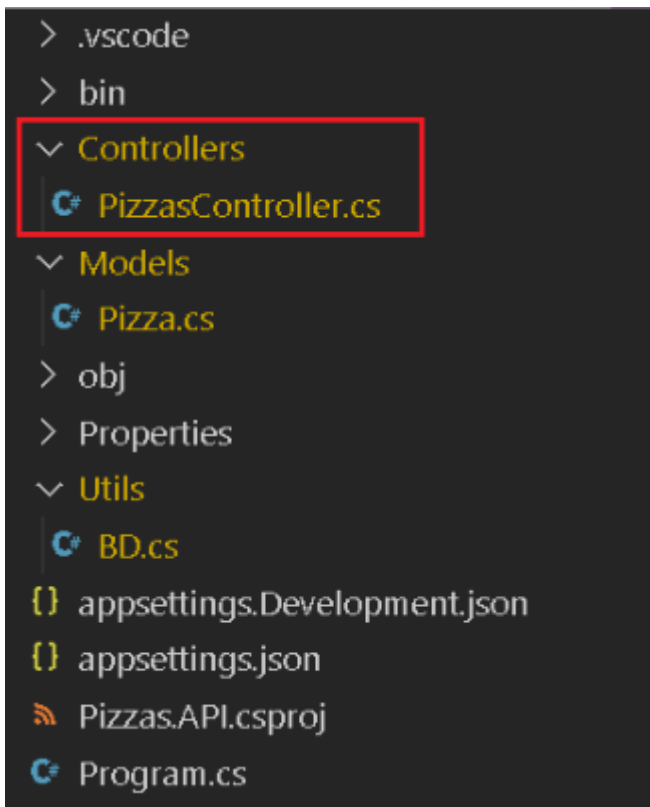


```
namespace Pizzas.API.Models {  
    public class Pizza {  
        public int Id { get; set; }  
        public string Nombre { get; set; }  
        public bool LibreGluten { get; set; }  
        public float Importe { get; set; }  
        public string Descripcion { get; set; }  
    }  
}
```

Controller

Debemos crear un Controller en la carpeta Controllers.

Controller: PizzasController



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Pizzas.API.Models;
using Pizzas.API.Utils;
namespace Pizzas.API.Controllers {
    [ApiController]
    [Route("api/[controller]")]
    public class PizzasController : ControllerBase {
        [HttpGet]
        public IActionResult GetAll() {...}

        [HttpGet("{id}")]
        public IActionResult GetById(int id) {...}

        [HttpPost]
        public IActionResult Create(Pizza pizza) {...}

        [HttpPut("{id}")]
        public IActionResult Update(int id, Pizza pizza) {...}

        [HttpDelete("{id}")]
        public IActionResult DeleteById(int id) {...}
    }
}
```

IActionResult

Los **IActionResult** son cualquier clase que implemente la interfaz **IActionResult**. Básicamente son clases que representan cosas que se supone que el cliente debe hacer como resultado de la acción del controlador. Es posible que necesiten obtener un archivo, redirigir o cualquier otra cosa.

Algunos **IActionResult** simplemente devuelven un código de estado HTTP (como OK 200, NotFound 404, etc.). En resumen, las cosas más comunes que el cliente podría querer hacer después de llamar a una acción del controlador se representan como resultados de la acción.

Hay 5 (cinco) tipos principales de **ActionResult**:

Status Code Results

Status Code w/ Object Results

Redirect Results

File Results

Content Results

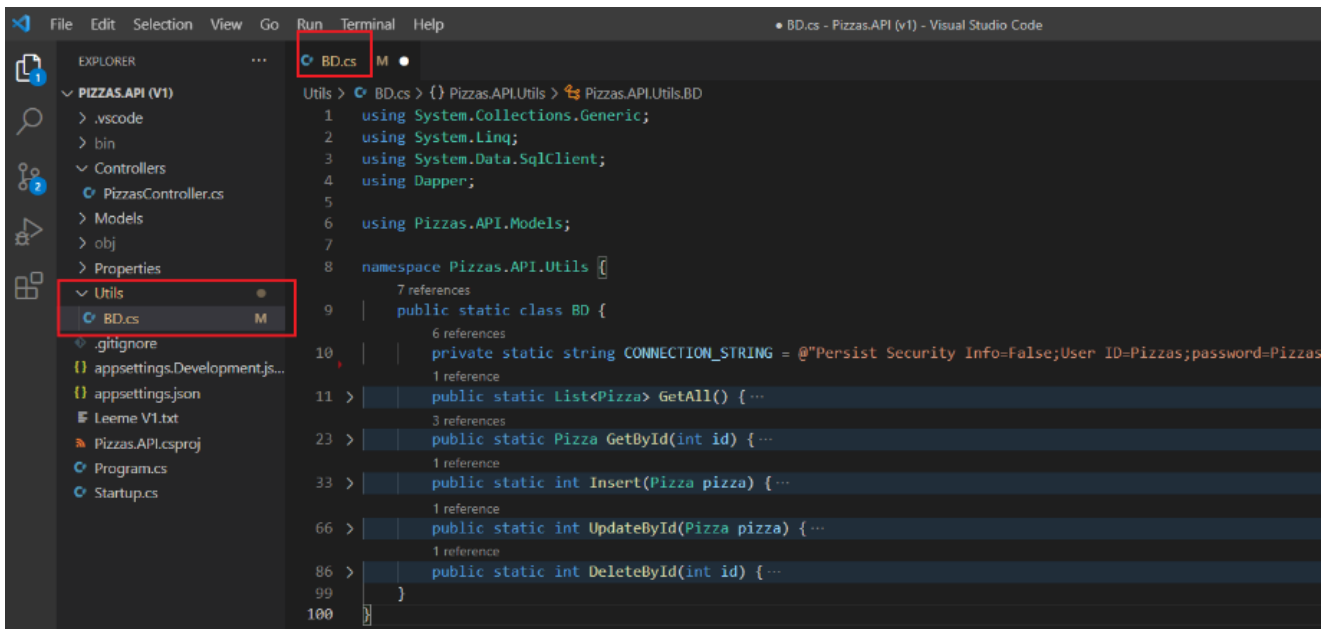
Base de datos

Debemos crear una base de datos llamada: **DAI-Pizzas**

Y ejecutar los archivos SQL **CreateTableAndData** y **CreateLoginAndUser**

BD.cs

En una carpeta **Utils** (namespace **Pizzas.API.Utils**) creamos una clase estática BD.cs que será la encargada de acceder a la **Tabla Pizzas** y **obtener todos los registros, obtener un solo registro, insertar, actualizar y eliminar**.



```
using System.Collections.Generic;
using System.Linq;
using System.Data.SqlClient;
using Dapper;

using Pizzas.API.Models;

namespace Pizzas.API.Utils {
    public static class BD {
        private static string CONNECTION_STRING = @"Persist Security Info=False;User ID=Pizzas;password=Pizzas;Initial Catalog=DAI-Pizzas;Data Source=.;";
```

El método `GetAll` retorna un List correspondiente a todos los registros de la base de datos..

```
public static List<Pizza> GetAll() {
    //
    // Obtiene todos los registro de la base de datos
    //
    string sqlQuery;
    List<Pizza> returnList;

    returnList = new List<Pizza>();
    using (SqlConnection db = new SqlConnection(CONNECTION_STRING)) {
        sqlQuery = "SELECT Id, Nombre, LibreGluten, Importe,
Descripcion ";
        sqlQuery += "FROM Pizzas";
        returnList = db.Query<Pizza>(sqlQuery).ToList();
    }

    return returnList;
}
```

```
}
```

El método **GetById** retorna la Pizza correspondiente al id enviado por parámetro, o null en caso de no encontrarla.

```
public static Pizza GetById(int id) {  
    //  
    // Obtiene un registro de la base de datos segun el Id  
    //  
    string sqlQuery;  
    Pizza returnEntity = null;  
  
    sqlQuery = "SELECT Id, Nombre, LibreGluten, Importe, Descripcion  
";  
    sqlQuery += "FROM Pizzas ";  
    sqlQuery += "WHERE Id = @idPizza";  
    using (SqlConnection db = new SqlConnection(CONNECTION_STRING)) {  
        returnEntity = db.QueryFirstOrDefault<Pizza>(sqlQuery, new {  
idPizza = id });  
    }  
    return returnEntity;  
}
```

El método **Insert** inserta la Pizza enviada por parámetro. Retorna los registros afectados.

```
public static int Insert(Pizza pizza) {  
    //  
    // Inserta un registro y retorna los RowsAffected.  
    //  
    string sqlQuery;  
    int intRowsAffected = 0;  
  
    sqlQuery = "INSERT INTO Pizzas (";  
    sqlQuery += "    Nombre , LibreGluten , Importe , Descripcion";  
    sqlQuery += ") VALUES (";  
    sqlQuery += "    @nombre , @libreGluten , @importe ,  
@descripcion";  
    sqlQuery += ")";  
    using (SqlConnection db = new SqlConnection(CONNECTION_STRING)) {  
        intRowsAffected = db.Execute(sqlQuery, new {  
            nombre = pizza.Nombre,  
            libreGluten = pizza.LibreGluten,  
            importe = pizza.Importe,  

```



```

        descripcion = pizza.Descripcion
    }

    );
}
return intRowsAffected;
}

```

El método **Update** actualiza la Pizza enviada por parámetro. Retorna los registros afectados.

```

public static int UpdateById(Pizza pizza) {
    //
    // Actualiza un registro y retorna los RowsAffected.
    //
    string sqlQuery;
    int intRowsAffected = 0;

    sqlQuery = "UPDATE Pizzas SET ";
    sqlQuery += "    Nombre        = @nombre, ";
    sqlQuery += "    LibreGluten    = @libreGluten, ";
    sqlQuery += "    Importe        = @importe, ";
    sqlQuery += "    Descripcion    = @descripcion ";
    sqlQuery += "WHERE Id = @idPizza";
    using (var db = new SqlConnection(CONNECTION_STRING)) {
        intRowsAffected = db.Execute(sqlQuery, new {
            idPizza    = pizza.Id,
            nombre      = pizza.Nombre,
            libreGluten = pizza.LibreGluten,
            importe     = pizza.Importe,
            descripcion = pizza.Descripcion
        });
    }
    return intRowsAffected;
}

```

El método **DeleteById** elimina la Pizza correspondiente al id enviado por parámetro. Retorna los registros afectados.

```

public static int DeleteById(int id) {
    //
    // Elimina un registro y retorna los RowsAffected.
    //
    string sqlQuery;
    int intRowsAffected = 0;

```

```

        sqlQuery = "DELETE ";
        sqlQuery += "FROM Pizzas ";
        sqlQuery += "WHERE Id = @idPizza";
        using (SqlConnection db = new SqlConnection(CONNECTION_STRING)) {
            intRowsAffected = db.Execute(sqlQuery, new { idPizza = id });
        }
        return intRowsAffected;
    }
}

```

El archivo **BD.cs** quedaría así:

```

using System.Collections.Generic;
using System.Linq;
using System.Data.SqlClient;
using Dapper;

using Pizzas.API.Models;

namespace Pizzas.API.Utils {
    public static class BD {
        private static string CONNECTION_STRING = @"Persist Security
Info=False;User ID=Pizzas;password=Pizzas;Initial Catalog=DAI-Pizzas;Data
Source=.";
        public static List<Pizza> GetAll() {
            //
            // Obtiene todos los registro de la base de datos
            //
            string sqlQuery;
            List<Pizza> returnList;

            returnList = new List<Pizza>();
            using (SqlConnection db = new SqlConnection(CONNECTION_STRING)) {
                sqlQuery = "SELECT Id, Nombre, LibreGluten, Importe,
Descripcion ";
                sqlQuery += "FROM Pizzas";
                returnList = db.Query<Pizza>(sqlQuery).ToList();
            }

            return returnList;
        }
        public static Pizza GetById(int id) {
            //
            // Obtiene un registro de la base de datos segun el Id
            //
            string sqlQuery;
            Pizza returnEntity = null;

```

```

        sqlQuery = "SELECT Id, Nombre, LibreGluten, Importe, Descripcion
";
        sqlQuery += "FROM Pizzas ";
        sqlQuery += "WHERE Id = @idPizza";
        using (SqlConnection db = new SqlConnection(CONNECTION_STRING)) {
            returnEntity = db.QueryFirstOrDefault<Pizza>(sqlQuery, new {
idPizza = id });
        }
        return returnEntity;
    }
    public static int Insert(Pizza pizza) {
        //
        // Inserta un registro y retorna los RowsAffected.
        //
        string sqlQuery;
        int    intRowsAffected = 0;

        sqlQuery = "INSERT INTO Pizzas (";
        sqlQuery += "    Nombre    , LibreGluten    , Importe    , Descripcion";
        sqlQuery += ") VALUES (";
        sqlQuery += "    @nombre , @libreGluten , @importe ,
@descripcion";
        sqlQuery += ")";
        using (SqlConnection db = new SqlConnection(CONNECTION_STRING)) {
            intRowsAffected = db.Execute(sqlQuery, new {
                                nombre =
pizza.Nombre,
                                libreGluten =
pizza.LibreGluten,
                                importe =
pizza.Importe,
                                descripcion =
pizza.Descripcion
                                }
                                );
        }
        return intRowsAffected;
    }

    public static int UpdateById(Pizza pizza) {
        //
        // Actualiza un registro y retorna los RowsAffected.
        //
        string sqlQuery;
        int    intRowsAffected = 0;

        sqlQuery = "UPDATE Pizzas SET ";
        sqlQuery += "    Nombre        = @nombre, ";
        sqlQuery += "    LibreGluten    = @libreGluten, ";
        sqlQuery += "    Importe        = @importe, ";
        sqlQuery += "    Descripcion    = @descripcion ";
        sqlQuery += "WHERE Id = @idPizza";
    }

```

```

        using (var db = new SqlConnection(CONNECTION_STRING)) {
            intRowsAffected = db.Execute(sqlQuery, new {
                idPizza =
pizza.Id,
                nombre =
pizza.Nombre,
                libreGluten =
pizza.LibreGluten,
                importe =
pizza.Importe,
                descripcion =
pizza.Descripcion
            });
        }
        return intRowsAffected;
    }
    public static int DeleteById(int id) {
        //
        // Elimina un registro y retorna los RowsAffected.
        //
        string sqlQuery;
        int intRowsAffected = 0;

        sqlQuery = "DELETE ";
        sqlQuery += "FROM Pizzas ";
        sqlQuery += "WHERE Id = @idPizza";
        using (SqlConnection db = new SqlConnection(CONNECTION_STRING)) {
            intRowsAffected = db.Execute(sqlQuery, new { idPizza = id });
        }
        return intRowsAffected;
    }
}

```

API Controller

El método `GetAll` de `PizzasController`, va a devolver la respuesta del método `GetAll` de BD, cuyo objetivo es retornar todas las pizzas que se encuentran en la lista que maneja internamente.

```

[HttpGet]
public IActionResult GetAll() {
    IActionResult respuesta;
    List<Pizza> entityList;

    entityList = BD.GetAll();
    respuesta = Ok(entityList);
    return respuesta;
}

```

El método Ok() de un controller retorna un Status Code **200 (Ok)**

Para probar el método en Postman, debemos escribir en la URL :

```
http://localhost:5000/Pizzas/
```

Que hace referencia al método GET [HttpGet] que creamos en el controller. Nos devuelve los siguientes resultados.

```
1 [
2   {
3     "id": 1,
4     "nombre": "Pizza Muzzarella",
5     "libreGluten": false,
6     "importe": 800.5,
7     "descripcion": "Pizza con queso Muzzarella."
8   },
9   {
10    "id": 2,
11    "nombre": "Pizza Fugazzeta",
12    "libreGluten": true,
13    "importe": 1000,
14    "descripcion": "Pizza con queso Muzzarella, tiene rica cebolla y te deja un aliento como para hacerle el alisado del pelo a tu pareja."
15  },
16  {
17    "id": 3,
18    "nombre": "Pizza Carbonara",
19    "libreGluten": true,
20    "importe": 1540.5,
21    "descripcion": "Salsa carbonara: huevo, queso parmesano, sal y pimienta. Cebolla, bacon (kevin) y queso rallado por encima, ya que por debajo no se ve."
22  },
23 ]
```

PizzasController GetById(id)

Puede ser que a veces necesitemos obtener los datos de una sola pizza.

Es por esto que .NET Core nos permite especificar la recepción de un parámetro "{id}" en una petición GET.

```
[HttpGet("{id}")]
public IActionResult GetById(int id) {
    IActionResult respuesta = null;
    Pizza entity;

    entity = BD.GetById(id);
    if (entity == null) {
        respuesta = NotFound();
    } else {
        respuesta = Ok(entity);
    }
    return respuesta;
}
```

El método **NotFound()** de un Controller retorna un Status Code 404 (Not Found)

Para probar el método en Postman, debemos escribir en la URL :

```
http://localhost:5000/Pizzas/9
```

Que hace referencia al método GET [HttpGet("{id}")] que creamos en el controller. Nos devuelve el siguiente resultado.

```
1 {  
2   "id": 9,  
3   "nombre": "Pizza Picante.",  
4   "libreGluten": true,  
5   "importe": 1400,  
6   "descripcion": "Masa fina, queso del mercadito de la esquina, aji mejicano  
    recontra picante. ATENCION: Cuando vas al baño se te incendia el papel  
    higienico."  
7 }
```

Probaste qué ocurre si pones un Id de una pizza inexistente?.

Que status code retorna?

PizzasController - Create(Pizza pizza)

Para poder crear nuevos elementos de tipo Pizza, debemos escuchar el método POST [HttpPost]. A este método le llega en el cuerpo (body) del mensaje la información necesaria para poder insertar una Pizza.

```
[HttpPost]  
public IActionResult Create(Pizza pizza) {  
    int intRowsAffected;  
    intRowsAffected = BD.Insert(pizza);  
    return CreatedAtAction(nameof(Create), new { id = pizza.Id }, pizza);  
}
```

El método `CreatedAtAction()` de un Controller retorna un Status Code Created (201), así como el objeto recientemente creado.

Para probar el método Create de la API, debemos escribir la URL estándar de las pizzas que usamos antes, pero marcar que el verbo es POST.

Luego, debemos especificar en el body del request la nueva pizza (obvio que el ID no va en el JSON, es autonumerico).



PizzasController - Update(int id, Pizza pizza)

Para poder actualizar una pizza existente, debemos escuchar el método `PUT [HttpPut("{id}")]`. A este método le llega el ID de la pizza a actualizar, y una nueva pizza en el cuerpo (body) del mensaje que reemplaza totalmente la anterior.

Si la pizza (body) tiene el mismo ID que el que viene por la URL y además existe en la base de datos, la actualiza y devuelve `Ok()` (Status Code 200).

Si la pizza (body) tiene el mismo ID que el que viene por la URL pero no existe en la base de datos, devuelve `NotFound()` (Status Code 404).

En el caso de que el ID de la Pizza que viene por la URL y la del body es diferente devuelve un `BadRequest()` (Status Code 400)

```
[HttpPut("{id}")]
public IActionResult Update(int id, Pizza pizza) {
    IActionResult respuesta = null;
    Pizza entity;
    int intRowsAffected;
    if (id != pizza.Id) {
        respuesta = BadRequest();
    } else {
        entity = BD.GetById(id);
        if (entity == null){
            respuesta = NotFound();
        } else {
            intRowsAffected = BD.UpdateById(pizza);
            if (intRowsAffected > 0){
                respuesta = Ok(pizza);
            } else {
                respuesta = NotFound();
            }
        }
    }
}
```

```
return respuesta;
}
```

Para probar el método Update de la API, debemos escribir la URL estándar de las pizzas y el ID pero marcar que el verbo es **PUT**.

Luego, debemos especificar en el **body** del request la pizza actualizada (obvio que el ID tiene que ser el de la URL)



PizzasController - DeleteById(int id)

Para poder eliminar una pizza existente, debemos escuchar el método **DELETE** [`HttpDelete("{id}")`]. A este método le llega el ID de la pizza a eliminar.

Si el ID que el que viene por la URL existe en la base de datos, se elimina y devuelve **Ok()** (**Status Code 200**) junto con la entidad eliminada en la respuesta.

Si el ID que el que viene por la URL no existe en la base de datos, devuelve **NotFound()** (**Status Code 404**).

```
[HttpDelete("{id}")]
public IActionResult DeleteById(int id) {
    IActionResult respuesta = null;
    Pizza entity;
    int intRowsAffected;

    entity = BD.GetById(id);
    if (entity == null){
        respuesta = NotFound();
    } else {
        intRowsAffected = BD.DeleteById(id);
        if (intRowsAffected > 0){
            respuesta = Ok(entity);
        } else {
            respuesta = NotFound();
        }
    }
}
```



```
return respuesta;  
}
```

Links y Documentación

Siempre podemos acceder a la documentación oficial de Microsoft, bastante bien actualizada y con un buen paso a paso para crear nuestras APIs con .NET Core.

Se encuentra para todos los sistemas operativos, y para seguirla mediante Visual Studio o Visual Studio Code.

Creación de una API web con ASP.NET Core

Tutorial: Creación de una API web con ASP.NET Core

Tutorial: Llamada a una API web de ASP.NET Core con JavaScript

Documentación de la API web de ASP.NET Core con Swagger/OpenAPI

¿Te quedaste manija? No te sale? ¿Querés compararlo con la resolución?.

Pedile al profe amigo....