27-6-2024

ASP.NET Core

Programación Web

Marisol Peñafiel
Andrés Meneses
Michael Peñaloza
Oscar Merino
UNIVERSIDAD DE CUENCA

1. ¿Qué es .NET?

.NET es una plataforma de desarrollo creada por Microsoft utilizada para construir y ejecutar aplicaciones en diversas plataformas, como son Windows, macOS, Linux, Android, etc. Algunos componentes clave de .NET incluyen .NET Core, que es la versión multiplataforma y de código abierto ideal para aplicaciones de servidor y servicios en la nube, y .NET Framework, la versión original que se ejecuta solo en Windows y es adecuada para aplicaciones de escritorio y servicios web.

Aunque .NET utiliza principalmente el lenguaje de programación C#, también es compatible con otros lenguajes como F# y Visual Basic, y ofrece un conjunto de bibliotecas y herramientas que facilitan el desarrollo, depuración y despliegue de aplicaciones.

2. ¿Qué es ASP.NET CORE?

ASP.NET Core es un marco de trabajo de código abierto y multiplataforma desarrollado por Microsoft para construir aplicaciones web modernas, servicios web y aplicaciones en la nube. Además, es multiplataforma y puede ejecutarse en Windows, macOS y Linux. Incluye un contenedor de inyección de dependencias incorporado, soporte para Razor Pages, y middleware para manejar solicitudes HTTP. ASP.NET Core ofreciendo una configuración flexible y unificada para diferentes entornos, integrándose fácilmente con otras tecnologías y servicios dentro del ecosistema .NET.

3. Instalaciones necesarias para ASP.NET Core

Para crear un proyecto en ASP.NET Core, es necesario instalar .NET versión 8, el editor Microsoft Visual Studio 2022 y SQL Server Management Studio 2022. Las instrucciones detalladas para estas instalaciones han sido enviadas al grupo de WhatsApp para que puedan realizarse con antelación.

4. Creación de la base de datos

Creamos una base de datos en la carpeta “Databases”.

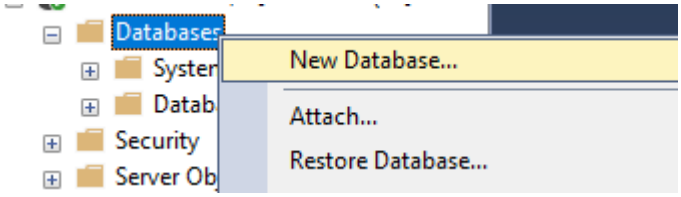

Ilustración 1: Opción para crear la basa de datos

Damos el nombre “PersonasDB” y seleccionamos “OK”

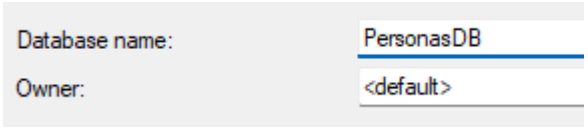

Ilustración 2: Nombre para la base de datos

5. Creación del Proyecto

Abrimos Visual Studio y seleccionamos la opción para crear un nuevo proyecto.

En la selección de lenguaje, elegimos C#. Luego, en el buscador, escribimos "Web App", seleccionamos "ASP.NET Core Web API" y presionamos "Siguiente". En la ventana que se abre, configuramos el nombre del proyecto y continuamos haciendo clic en "Siguiente".

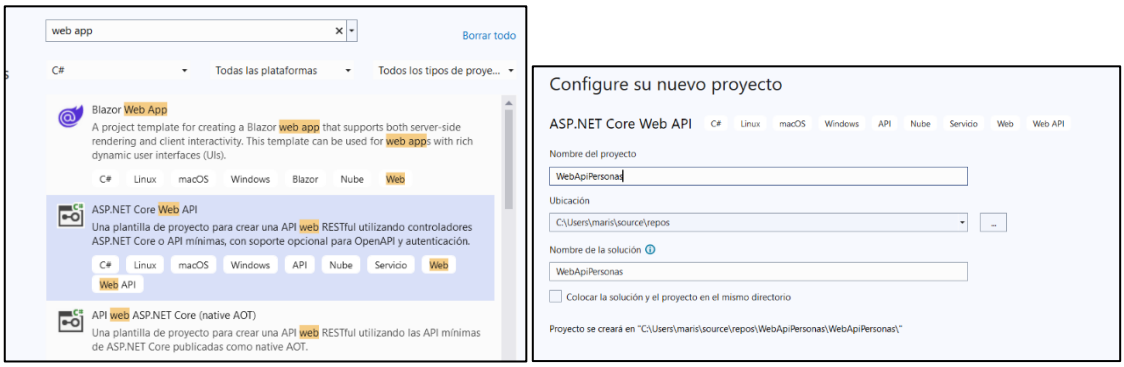

Ilustración 3: Creación el Proyecto

En la ventana de información adicional, seleccionamos el framework .NET 8.0 y dejamos las demás opciones con sus valores por defecto.

Ilustración 4: Información Adicional

Al hacer doble clic sobre el proyecto, se abrirá un archivo con la información detallada del mismo.

Dentro del proyecto, creamos una nueva carpeta llamada "modelo".

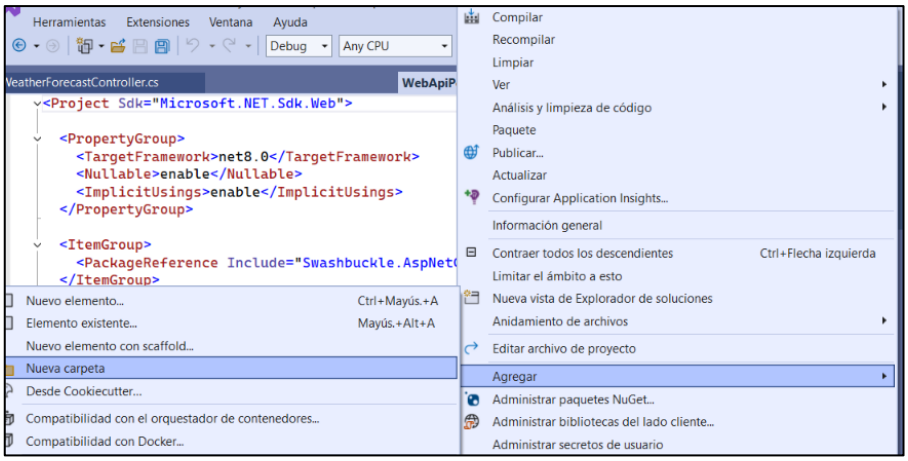

Ilustración 5: Creación Carpeta Modelo

Dentro de esta carpeta, creamos una clase llamada Persona.cs.

Nos ubicamos en la clase Persona.cs y colocamos el siguiente código

```
namespace WebApiPersonas.Modelo
{
   public class Persona
   {
      public int Id { get; set; }
      public required string Nombre { get; set; }
      public required string Apellido { get; set; }
      public required int Edad {  get; set; }
      public required string Email { get; set; }
      public required string Telefono { get; set; }
   }
}
```

Nos dirigimos a **Herramientas**, luego a **Administrador de paquetes NuGet** y después a **Administrar paquetes NuGet para la solución**.

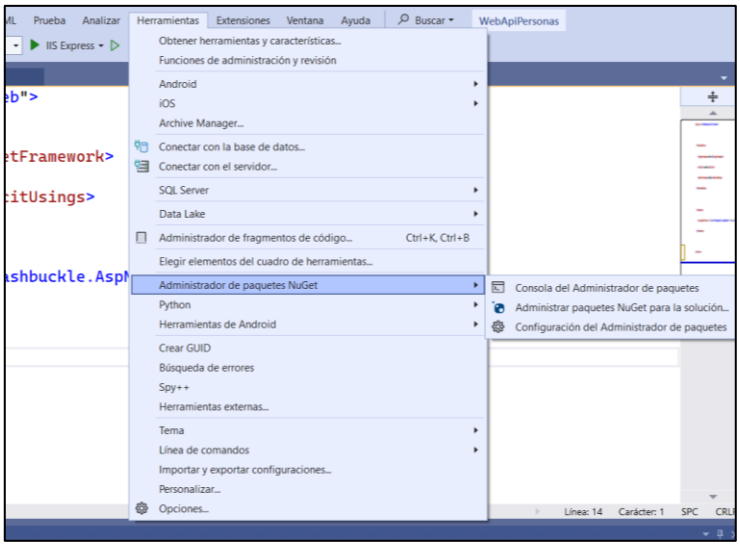

Ilustración 6: Instalación del Paquete 1

En la pestaña Examinar ingresaos la palabra entity, y descargamos Microsoft.EntityFrameworkCore, posteriormente seleccionamos nuestro proyecto y aceptamos.

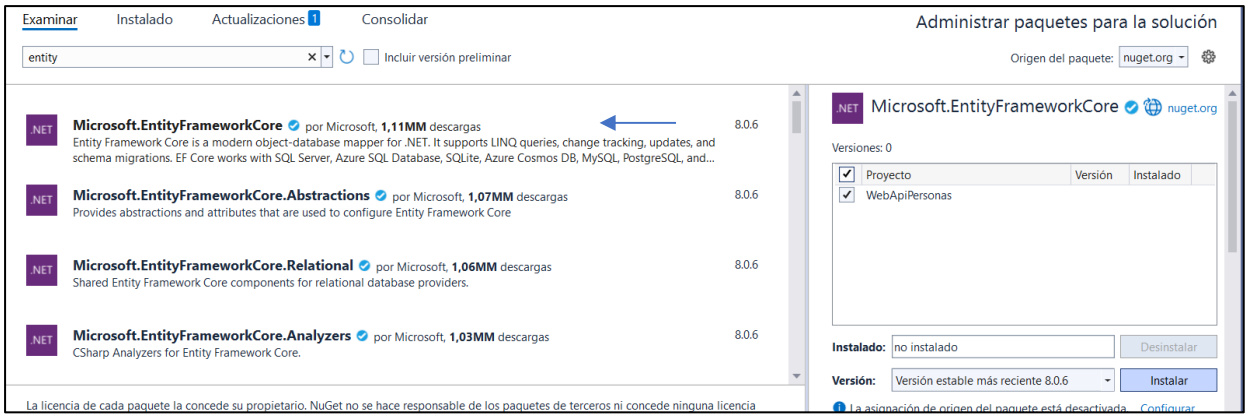

Ilustración 7: Instalación del Paquete 1

Ahora creamos una carpeta llamada Context, dentro de esta carpeta creamos una nueva clase la cual se llamará AppDbContext.cs, esta clase debe tener el siguiente código.

```
using Microsoft.EntityFrameworkCore;
using WebApiPersonas.Modelo;

namespace WebApiPersonas.Context
{
   public class AppDbContext: DbContext
   {
      public AppDbContext(DbContextOptions<AppDbContext> options): base(options)
      {

      }

      public DbSet<Persona> Personas { get; set; }
   }
}
```

Nos dirigimos al archivo appsettings.json, en el cuál colocaremos la línea para la conexión con la base de datos.

```
"ConnectionStrings": {
  "Connection": "Server=.\\SQLExpress; Database=PersonasDB; Trusted_Connection=true;
TrustServerCertificate=true"
}
```

Aquí colocaremos el nombre de la base de datos que queremos crear.

Nos dirigimos nuevamente a Nuget Solution para descargar Microsoft.EntityFrameworkCore.SqlServe. en el buscador escribimos entity sql server.

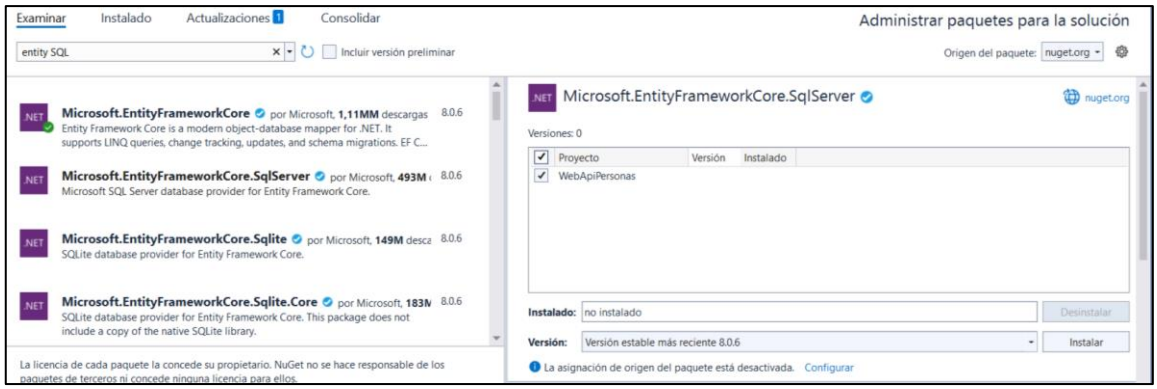

Ilustración 8: Instalación del Paquete 2

Nos dirigimos a la clase Program.cs en la cual agregaremos el código, luego de la línea de código:

// Add services to the container.

```
//Creamos una variable de cadena de configuración
var connectionString = builder.Configuration.GetConnectionString("Connection");
//Registramos los servicios
builder.Services.AddDbContext<AppDbContext>(
    options => options.UseSqlServer(connectionString)
);
```

Descargamos el paquete entity tolos, de la misma manera que se instaló los paquetes anteriores. El nombre del paquete es Microsoft.EntityFrameworkCore.Tools.

Ahora nos dirigimos a herramientas, luego a administrador de paquetes Nuget y luego a Consola del Administrador de Paquetes y ejecutamos los siguientes comandos:

```
Add-Migration Initial
```

Con lo cual se genera el código con el que podemos crear la tabla con los atributos que dimos la clase Persona.

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "personas",
        columns: table => new
        {
            Id = table.Column<int>(type: "int", nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            Nombre = table.Column<string>(type: "nvarchar(max)", nullable: false),
            Apellido = table.Column<string>(type: "nvarchar(max)", nullable: false),
            Edad = table.Column<int>(type: "int", nullable: false),
            Email = table.Column<string>(type: "nvarchar(max)", nullable: false),
            Telefono = table.Column<string>(type: "nvarchar(max)", nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_personas", x => x.Id);
        });
}
```

Ilustración 9: Código que crea la tabla Personas

Para poder ejecutar el código usamos el siguiente comando.

```
Update-database
```

Al abrir SQL Server podemos ver que ya se creó nuestra tabla Personas en la base de datos PersonasDB.

En la carpeta Migrations se puede observar la creación de clases que nos permitieron crear la base de datos y la tabla de manera automática.

Creamos una clase de tipo controlador dentro de la carpeta Controllers, para esto seleccionamos API y posteriormente Controlador de API con acciones que usan Entity Framework.Esta clase controlador se llamará PersonasController.cs.

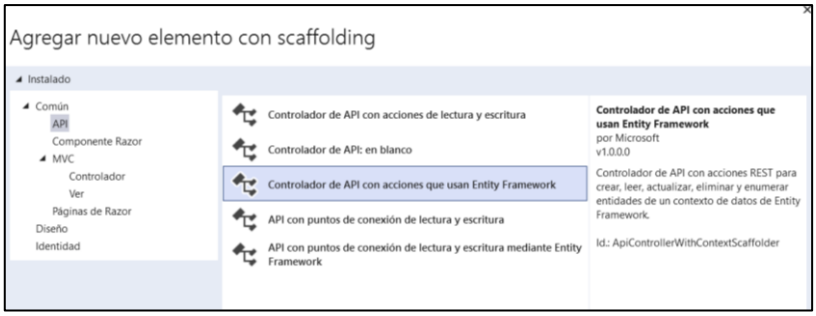

Ilustración 10: Creación PersonasControllers.cs

Seleccionamos la clase del Modelo, la clase del DBContext y agregar.

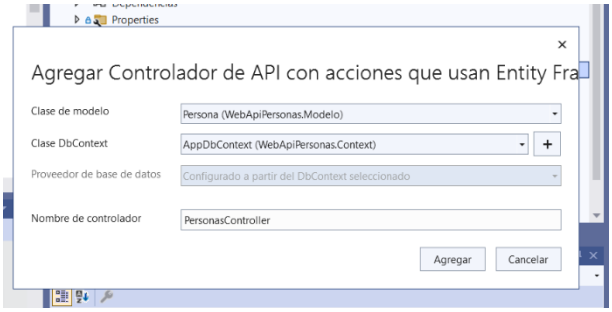

Ilustración 11: Selección de los parámetros del Controlador

Al dirigirnos a la carpeta de Controller observaras que la clase ya esta creada esta clase contine los métodos GET, POST, PUT y DELETE.

```
// GET: api/Personas
[HttpGet]
0 referencias
public async Task<ActionResult<IEnumerable<Persona>>> GetPersonas()
{
    return await _context.Personas.ToListAsync();
}

// GET: api/Personas/5
[HttpGet("{id}")]
0 referencias
public async Task<ActionResult<Persona>> GetPersona(int id)
{
    var persona = await _context.Personas.FindAsync(id);

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

    return persona;
}
```

Ilustración 12: Métodos GET

```
// PUT: api/Personas/5
// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
[HttpPut("{id}")]
0 referencias
public async Task<IActionResult> PutPersona(int id, Persona persona)
{
    if (id != persona.Id)
    {
        return BadRequest();
    }

    _context.Entry(persona).State = EntityState.Modified;

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

    return NoContent();
}
```

Ilustración 13: Método PUT

```
// POST: api/Personas
// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
[HttpPost]
0 referencias
public async Task<ActionResult<Persona>> PostPersona(Persona persona)
{
    _context.personas.Add(persona);
    await _context.SaveChangesAsync();

    return CreatedAtAction("GetPersona", new { id = persona.Id }, persona);
}
```

Ilustración 14: Método POST

```
// DELETE: api/Personas/5
[HttpDelete("{id}")]
0 referencias
public async Task<IActionResult> DeletePersona(int id)
{
    var persona = await _context.personas.FindAsync(id);
    if (persona == null)
    {
        return NotFound();
    }

    _context.personas.Remove(persona);
    await _context.SaveChangesAsync();

    return NoContent();
}
```

Ilustración 15: Método DELETE

Para ver el funcionamiento de la app, nos dirigimos a seleccionar IIS, y ejecutamos.

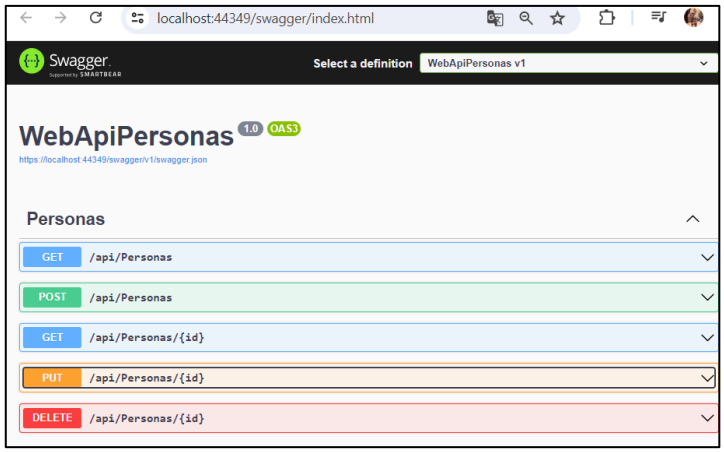

Ilustración 16: Ejecución en IIS

Ahora al ingresar al post, ya podemos realizar nuestras peticiones.

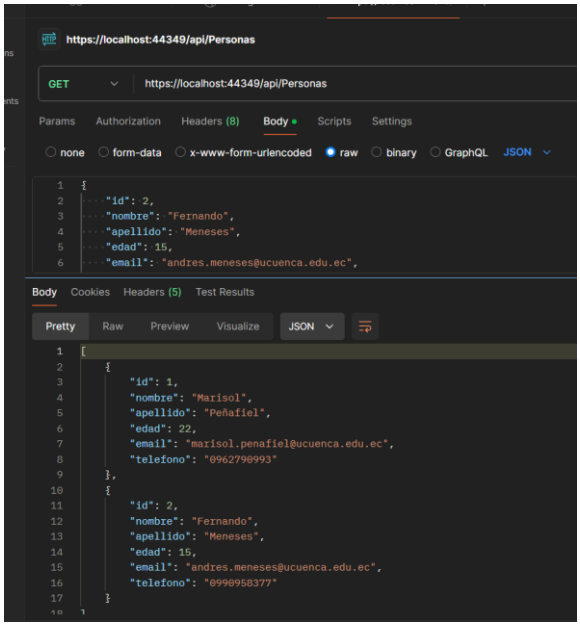

Ilustración 17: Peticiones desde Postman

Si abrimos nuestro SQL Server observarás que ya se agregaron los datos a nuestra tabla.

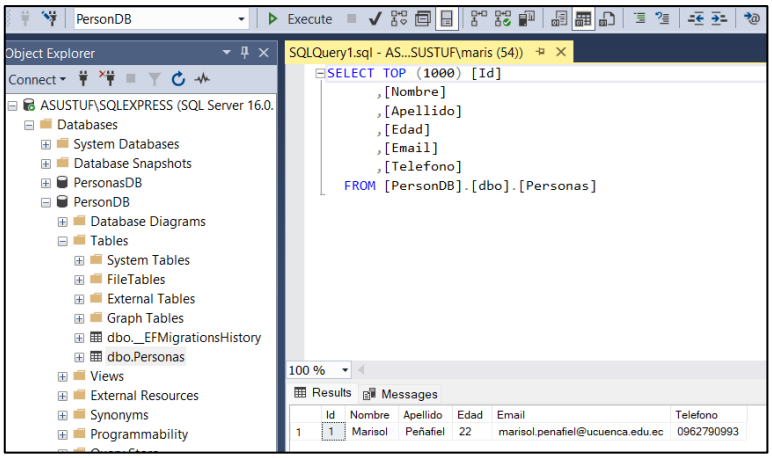

Ilustración 18: Contenido de la DB