

TEMA DE LA PRÁCTICA: Diseño cliente-Servidor

MÓ DULO: Aplicaciones Distribuidas

DOCENTE: Ing. Caiza Caizabuan Jose Ruben

ESTUDIANTE: Quishpe López Luis Alexander

NIVEL: Sexto Software “A”

FECHA: 05/04/2025



1. OBJETIVOS

General:

- Implementar y documentar un API RESTful que ayude el acceso de datos desde un servidor local (Ubuntu), siguiendo los pasos estudiados en clase.

Específicos:

- Desarrollar un API RESTful con todos los endpoints para operaciones CRUD, utilizando la tecnología de ASP.NET Core Web API.
- Configurar un servidor remoto en Ubuntu que permita alojar los datos consumidos por el API, garantizando la comunicación mediante protocolo HTTP y formato JSON.
- Validar y documentar el funcionamiento del API y del servidor mediante pruebas manuales en Postman y en el navegador.

2. INTRODUCCION

El diseño de APIs RESTful se ha consolidado como un estándar de desarrollo, con la finalidad de ser un proveedor de servicios web que pueden ser consumidos por otros desarrolladores en distintos tipos de aplicaciones como sitios web, aplicaciones móviles, entre otros [1]. Este estilo arquitectónico, basado en los principios del protocolo HTTP, permite la interoperabilidad entre sistemas distribuidos, facilitando la integración de aplicaciones cliente-servidor. En la parte académica y profesional, su adaptabilidad ha crecido exponencialmente, especialmente en entornos como en la nube y microservicios [2]. En esta práctica se implementará un API RESTful funcional, con el fin de consumir los datos desde un servidor remoto que se implementará en Ubuntu.

Lo importante de este trabajo es aplicar los conceptos y conocimientos impartidos por el docente en clases, donde se priorizará la implementación de la API RESTful en el servidor y que permita el manejo de los datos.

3. HERRAMIENTAS EMPLEADAS

- Plataforma estudiantil
- Internet
- Inteligencia Artificial
- Computadora
- Word y apuntes de clase.
- Windows, Ubuntu, SQL server .Net.

4. DESARROLLO

A continuación, se describen los pasos realizados para implementar el API RESTful y la integración con el servidor local en Ubuntu.



UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL

CARRERA DE SOFTWARE

CICLO ACADÉMICO: MARZO – JULIO 2025



Paso 1: Creación del proyecto API RESTful

Se desarrolló una API RESTful en Visual Studio aplicando el patrón MVC, definiendo las entidades Persona, Cliente y Pedido como modelos, junto con sus respectivos controladores CRUD. Para la documentación interactiva se implementó Swagger como interfaz de visualización. Adicionalmente, se integró seguridad mediante JWT (JSON Web Tokens), configurando autenticación y autorización en los endpoints críticos.

```
1 using Microsoft.AspNetCore.Authentication.JwtBearer;
2 using Microsoft.EntityFrameworkCore;
3 using Microsoft.Extensions.Options;
4 using Microsoft.IdentityModel.Tokens;
5 using Microsoft.OpenApi.Models;
6 using System.Configuration;
7 using System.Text;
8 using WebApiPerson.Context;
9
10 var builder = WebApplication.CreateBuilder(args); //crea la app web
11
12 // Add services to the container.
13 //crear variable para la cadena de conexion.
14 var connectionString = builder.Configuration.GetConnectionString("Connection");
15 //Registrar el contexto para la conexion
16 builder.Services.AddDbContext<AppDbContext>(>
17     options => options.UseSqlServer(connectionString));
18 //Autenticación
19 //Configuración de autenticación JWT
20 builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
21     .AddJwtBearer(options =>
22     {
23         options.RequireHttpsMetadata = false;
24         options.SaveToken = true;
25         options.TokenValidationParameters = new TokenValidationParameters
26         {
27             ValidateIssuerSigningKey = true,
28             IssuerSigningKey = new SymmetricSecurityKey(
29                 Encoding.UTF8.GetBytes("LuCpKvE77P7pKJ2QSS1duBf8ay7Wx1kt1TSH1")),
30             ValidateIssuer = false,
31             ValidateAudience = false
32         });
33     });
34
35 builder.Services.AddAuthorization(); // Asegurar que la autorización esté habilitada
36
37 // Configuración de seguridad para JWT en Swagger
38 builder.Services.AddSwaggerGen(c =>
39 {
40     c.SwaggerDoc("v1", new OpenApiInfo { Title = "WebApiPerson", Version = "v1" });
41     // Configuración de autenticación con JWT en Swagger
42     c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
43     {
44         Name = "Bearer",
45         In = "header",
46         Type = "http",
47         Scheme = "bearer",
48         BearerFormat = "JWT",
49         OpenIdConnectUrl = null,
50     });
51     c.AddSecurityRequirement(new OpenApiSecurityRequirement()
52     {
53         { "Bearer", new OpenApiSecurityScheme() {} },
54     });
55 });
```

Gráfico 1. Configuración del program.cs

Creación modelos que funcionarán como de entidades estructuradas de manera que se organiza y actúa como intermediario entre la lógica de negocios y la base de datos.

```
1 namespace WebApiPerson.Models
2 {
3     public class Cliente
4     {
5         public int Id { get; set; }
6         public string Name { get; set; }
7         public string Email { get; set; }
8     }
9 }
10
11 namespace WebApiPerson.Models
12 {
13     public class Pedido
14     {
15         public int Id { get; set; }
16         public string Producto { get; set; }
17         public int Cantidad { get; set; }
18     }
19 }
```

Gráfico 2. Entidades: clientes, pedidos y personas.



UNIVERSIDAD TÉCNICA DE AMBATO
FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL
CARRERA DE SOFTWARE
CICLO ACADÉMICO: MARZO – JULIO 2025



Creación de los controllers de cada entidad que maneja la API RESTful. Cada uno contiene todos los métodos de acceso para peticiones generando respuestas desde la base de datos.

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using WebApiPerson.Services;

namespace WebApiPerson.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private JwtSettings _jwtSettings;

        public AuthController(JwtSettings jwtSettings)
        {
            _jwtSettings = jwtSettings;
        }

        [HttpPost("token")]
        public async Task GenerateToken()
        {
            var claims = new[]
            {
                new Claim(JwtRegisteredClaimNames.Sub, "testuser"),
                new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
            };

            var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_jwtSettings.SecretKey));
            var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

            var token = new JwtSecurityToken(
                issuer: _jwtSettings.Issuer,
                audience: _jwtSettings.Audience,
                claims: claims,
                expires: DateTime.Now.AddMinutes(_jwtSettings.ExpiryMinutes),
                signingCredentials: creds
            );

            return Ok(new JwtSecurityTokenHandler().WriteToken(token));
        }
    }
}
```

Gráfico 3. Controllers: personas, pedidos, clientes

Paso 2: JWT (Json Web Token)

La clase JwtSettings.cs y AuthController.cs contiene la implementación y configuración para la autenticación JWT, es utilizado para firmar los tokens de accesos a las distintos endpoints.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Authorization;
using Microsoft.AspNetCore.Mvc.Filters;
using WebApiPerson.Context;
using WebApiPerson.Models;

namespace WebApiPerson.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize]
    public class PersonasController : ControllerBase
    {
        private readonly AppDbContext _context;

        public PersonasController(AppDbContext context)
        {
            _context = context;
        }

        // GET: api/Personas
        [HttpGet]
        [Authorize]
        public async Task GetPersonas()
        {
            return await _context.Personas.ToListAsync();
        }

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

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

            return Ok(persona);
        }
    }
}
```

Gráfico 4. Implementación JWT



UNIVERSIDAD TÉCNICA DE AMBATO
FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL
CARRERA DE SOFTWARE
CICLO ACADÉMICO: MARZO – JULIO 2025



Paso 3: Creación de la Base de Datos

Se agrega el paquete NuGet “Microsoft.EntityFrameworkCore.SqlServer” mapear las clases (modelos) a tablas SQL Server, generando automáticamente la base de datos mediante migraciones. Para esto se necesita la configuración de la cadena de conexión esto se hace en el appsettings.json, sin la necesidad de escribir SQL permite tener el esquema directamente en la base de datos.

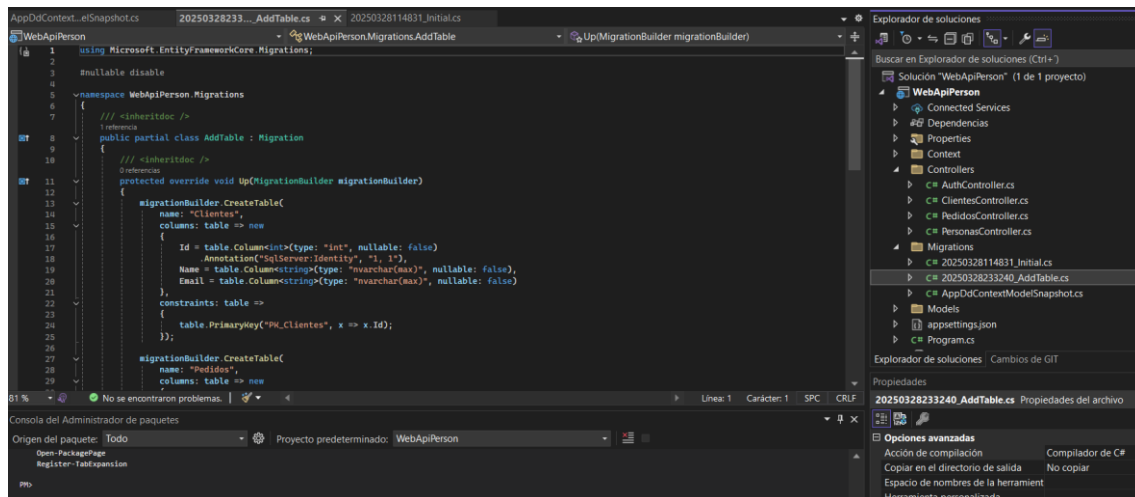


Gráfico 5. Migración y creación de la base de datos.

Paso 4: Verificación de la base de datos.

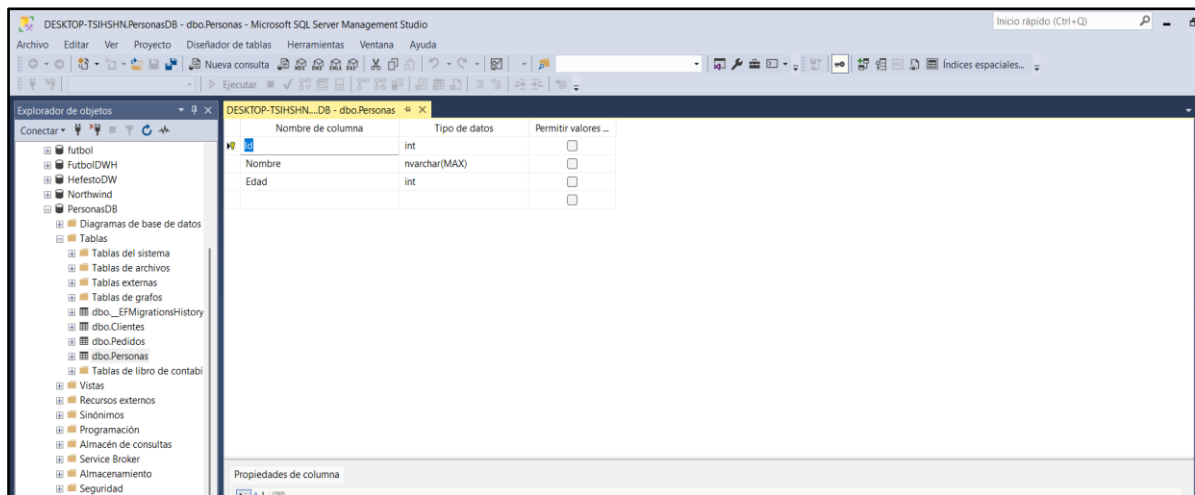


Gráfico 6. Base de Datos y tablas.



Paso 5: Funcionamiento de la API con Swagger

Una vez configurado la base de datos y los controladores. Se ejecuta la aplicación para verificar el funcionamiento de la API RESTful, se utiliza la Vista el SwaggerUI. Como se observa en la imagen aparecen los diferentes endpoints que contiene el proyecto organizado por controladores además de la ruta la cual se manejan. Incluido el JWT para seguridad.

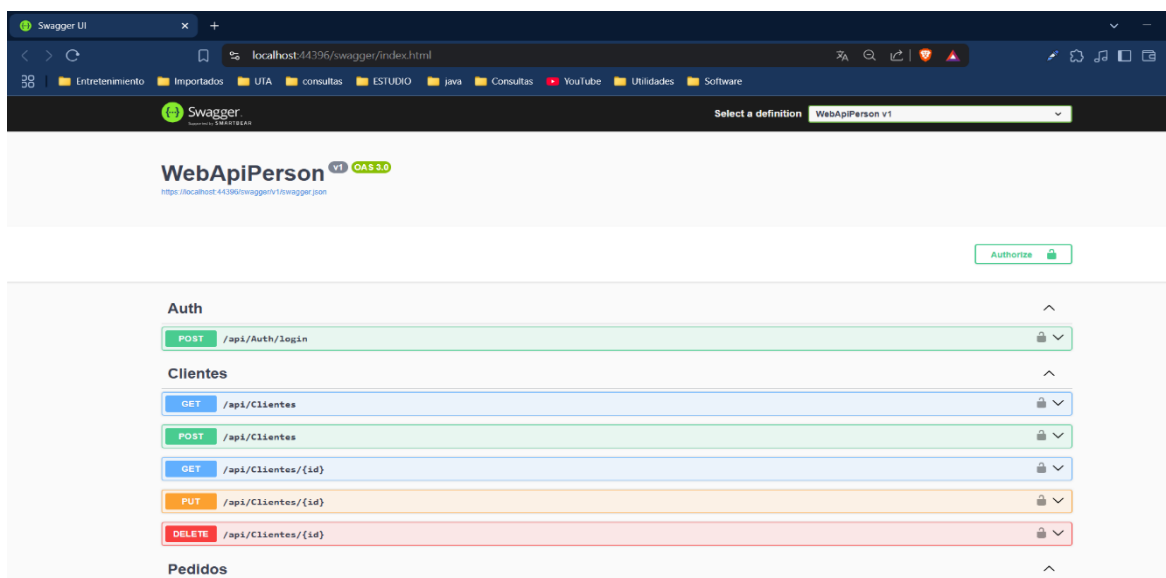


Gráfico 7. Funcionamiento de endpoints de la API RESTful.

Implementación del servidor en Ubuntu conjunto con el proyecto.

Paso 1: Creación de la carpeta compartida

Una vez ingresado a la máquina virtual y con la ayuda de comandos desde la terminal se crea la carpeta compartida, asignando permisos de acceso y usuarios, además de utilizar samba para el servicio compartido.

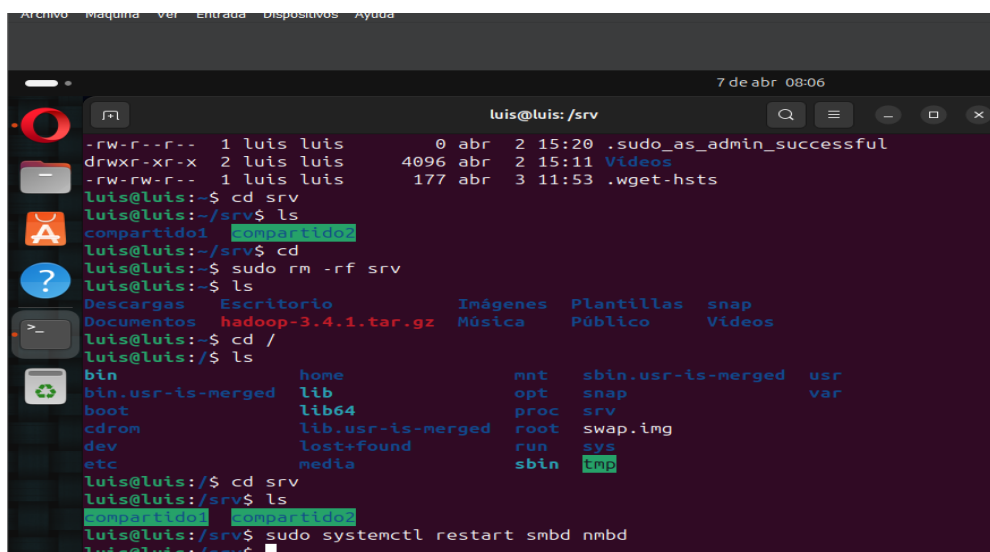


Gráfico 8. Creación de la carpeta compartida Ubuntu-Windows.



Verificación de la carpeta compartida desde Ubuntu a Windows accediendo desde la ruta **//IP/compartido2**.

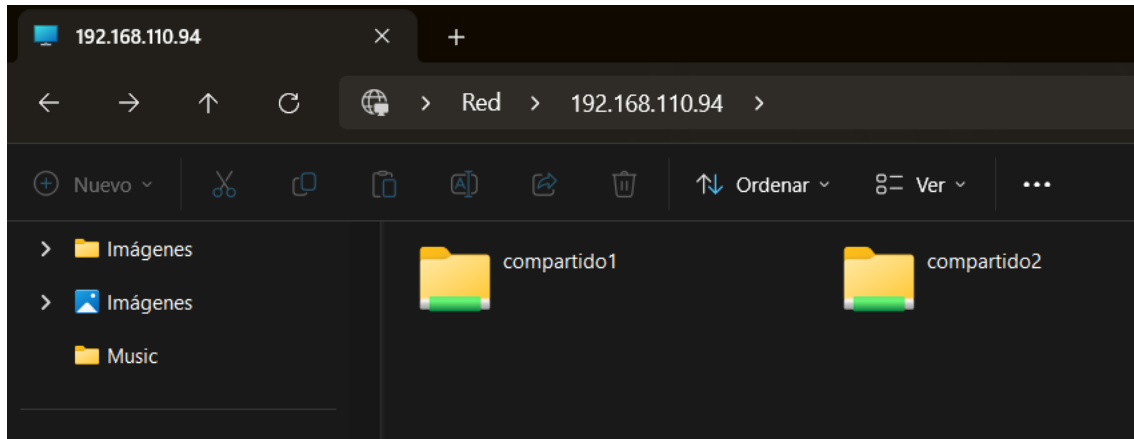


Gráfico 9. Carpeta compartida Windows

Paso 2: Publicación de la API

Desde Visual Studio se publica el proyecto dentro de la carpeta compartida para luego utilizar el proyecto desde Ubuntu y levantar el servidor. Configurando la ruta de destino y el Runtime de destino.

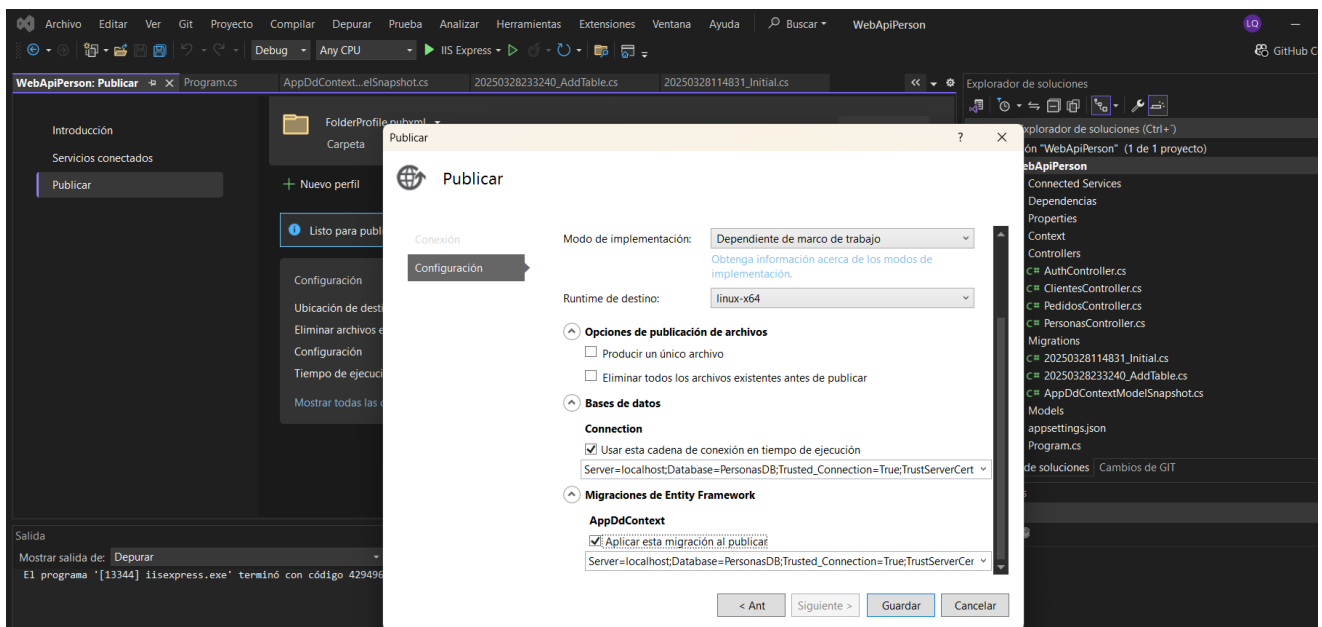


Gráfico 10. Publicación del proyecto dentro de la carpeta compartida.



UNIVERSIDAD TÉCNICA DE AMBATO
FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL
CARRERA DE SOFTWARE
CICLO ACADÉMICO: MARZO – JULIO 2025

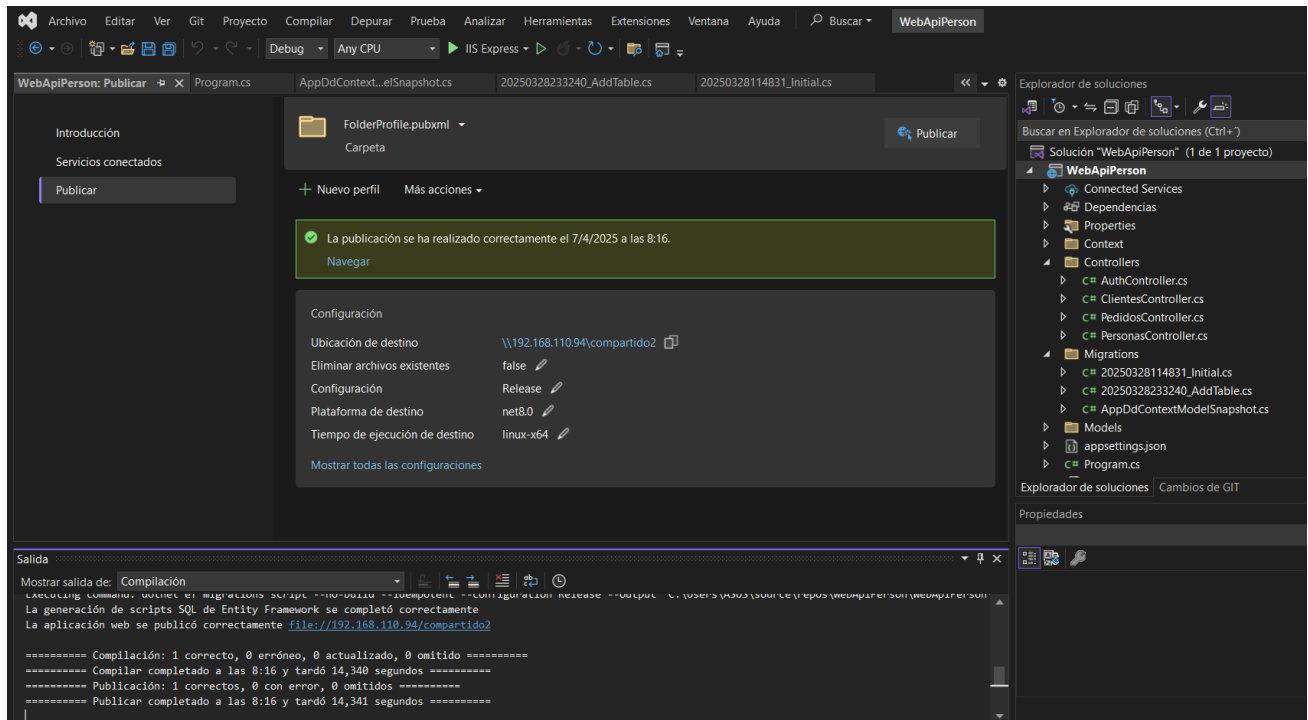


Gráfico 11. Proyecto publicado.

Ejecución del proyecto en Ubuntu para probar el funcionamiento de la API RESTful. Para ejecutar el API RESTful se debe colocar el siguiente comando: `dotnet WebApiPerson.dll`

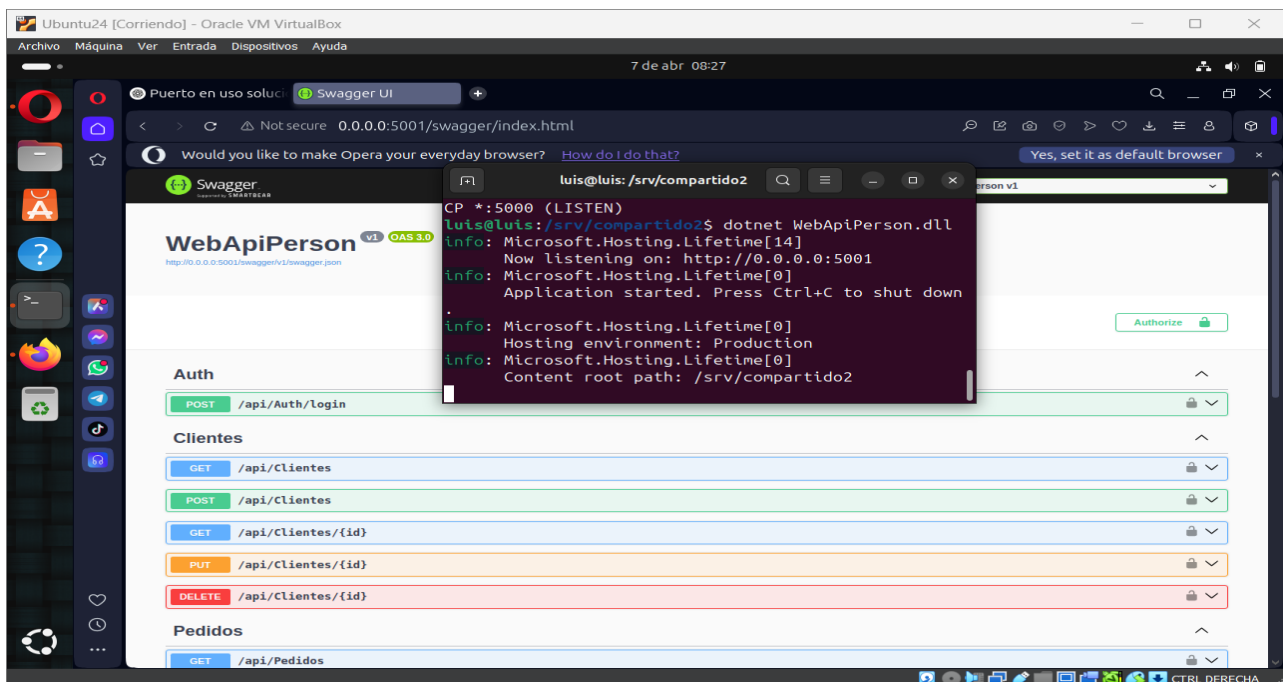


Gráfico 12. Ejecución de la API RESTful



UNIVERSIDAD TÉCNICA DE AMBATO
FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL
CARRERA DE SOFTWARE
CICLO ACADÉMICO: MARZO – JULIO 2025



Prueba de funcionamiento de la API RESTful desde el servidor local.

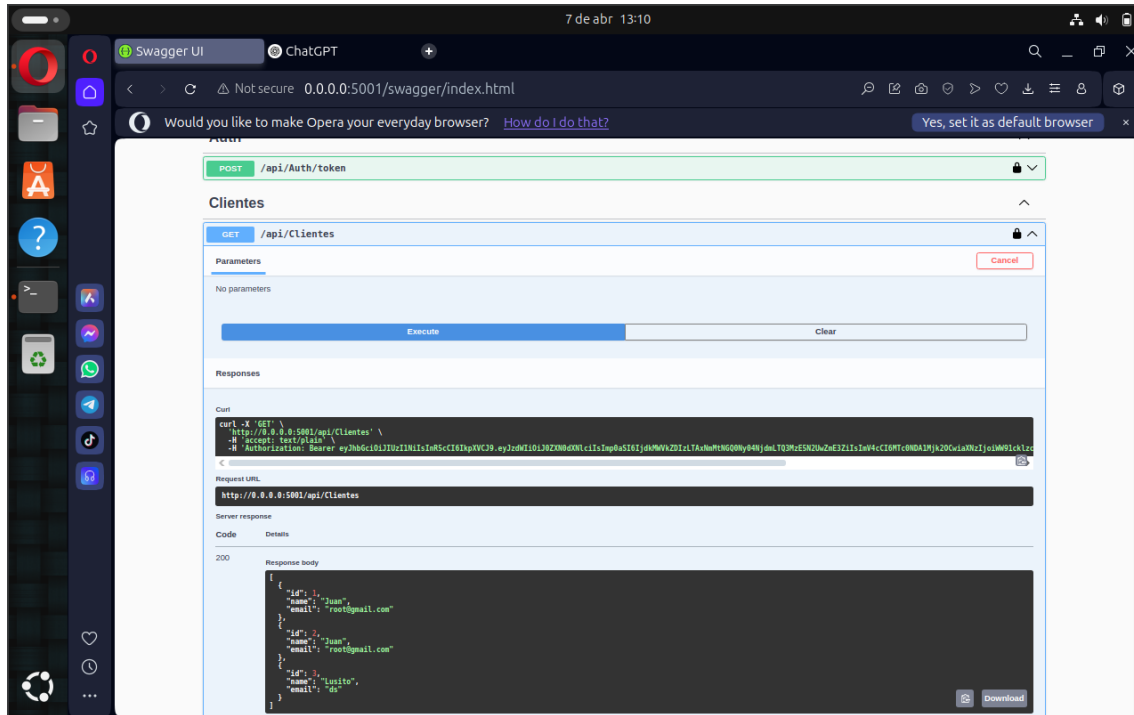


Gráfico 13. Api Restfull iniciado desde Ubuntu servidor local.

Prueba de funcionamiento de la API RESTful desde Windows manejado por la dirección <http://192.168.4.213:5001/Sagger>

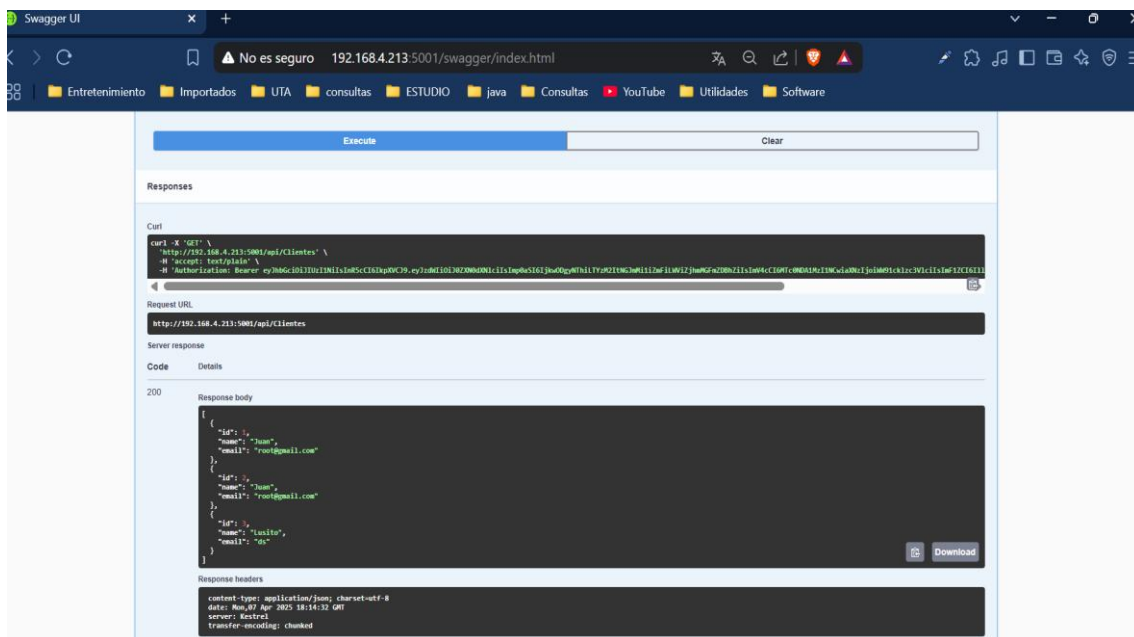


Gráfico 14. API RESTful en Windows.

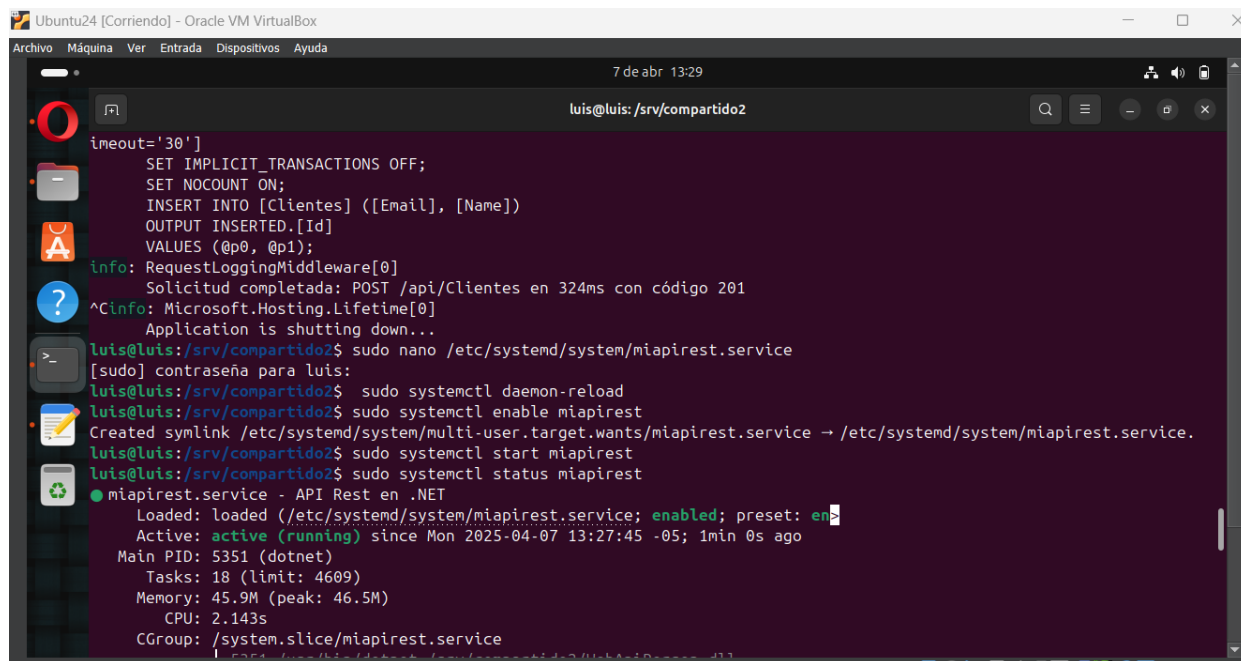


Paso 4: Creación del servicio systemd para que la API arranque automáticamente.

Para que la API arranque de manera automática cuando se encienda el servidor hay que crear un archivo de servicio con las siguientes configuraciones:

```
[Unit]
Description=API Resy en .NET
After=network.target
[Service]
WorkingDirectory=/srv/compartido1
ExecStart=/usr/bin/dotnet /srv/compartido1/MiApiRest.dll
Restart=always
RestartSec=10
# TimeoutSec=30
KillSignal=SIGINT
User=www-data
Environment=ASPNETCORE_ENVIRONMENT=Production
Environment=DOTNET_PRINT_TELEMETRY_INFORMATION=false
SyslogIdentifier=miapi
[Install]
WantedBy=multi-user.target
```

Una vez configurado el archivo se inicia el proceso una sola vez con el siguiente comando: **sudo systemctl start miapi**.



```
luis@luis:/srv/compartido2
timeout='30']
SET IMPLICIT_TRANSACTIONS OFF;
SET NOCOUNT ON;
INSERT INTO [Clientes] ([Email], [Name])
OUTPUT INSERTED.[Id]
VALUES (@p0, @p1);
info: RequestLoggingMiddleware[0]
Solicitud completada: POST /api/Clientes en 324ms con código 201
^Cinfo: Microsoft.Hosting.Lifetime[0]
Application is shutting down...
luis@luis:/srv/compartido2$ sudo nano /etc/systemd/system/miapi.service
[sudo] contraseña para luis:
luis@luis:/srv/compartido2$ sudo systemctl daemon-reload
luis@luis:/srv/compartido2$ sudo systemctl enable miapi.service
Created symlink /etc/systemd/system/multi-user.target.wants/miapi.service -> /etc/systemd/system/miapi.service.
luis@luis:/srv/compartido2$ sudo systemctl start miapi.service
luis@luis:/srv/compartido2$ sudo systemctl status miapi.service
● miapi.service - API Rest en .NET
   Loaded: loaded (/etc/systemd/system/miapi.service; enabled; preset: ena
   Active: active (running) since Mon 2025-04-07 13:27:45 -05; 1min 0s ago
     Main PID: 5351 (dotnet)
       Tasks: 18 (limit: 4609)
      Memory: 45.9M (peak: 46.5M)
         CPU: 2.143s
        CGroup: /system.slice/miapi.service
```

Gráfico 15. Status del servicio running.



5. CONCLUSIONES

En conclusión, se implementó con éxito una API RESTful funcional alojada en un servidor Ubuntu, capaz de gestionar las entidades clientes, pedidos y personas con un rendimiento óptimo. La integración de Entity Framework agilizó el desarrollo al automatizar la creación y consulta de la base de datos, eliminando la necesidad de escribir comandos SQL manuales y garantizando un acceso eficiente a los datos.

Además, como capa crítica de seguridad, se implementó JWT (JSON Web Tokens), que autentica y autoriza a los usuarios, protegiendo los endpoints contra accesos no autorizados y mitigando riesgos de ataques externos. Finalmente, se configuró el servidor Ubuntu para exponer la API mediante una dirección IP específica, permitiendo el acceso remoto a los endpoints desde cualquier cliente autorizado.

6. REFERENCIAS BIBLIOGRÁFICAS

Bibliografía

- [1] T. A. V. C. a. K. E. U. Arriaza, «Implementación de API RESTFUL para uso en APP de ofertas (Chaplist), desarrollada con IONIC,» *Ph.D. dissertation, Universidad de San Carlos de Guatemala*, 2016.
- [2] S. Newman, «O'Reilly Media Inc. Building microservice,» 2015.

joseru82@hotmail.com