



INFORME DE API REST FULL

I. PORTADA

Tema:	Despliegue de un API RESTful en Ubuntu con .NET
Unidad de Organización Curricular:	PROFESIONAL
Nivel y Paralelo:	Nivel - Paralelo
Alumnos participantes:	Albán Chávez Melanie Elizabeth
Asignatura:	Aplicaciones Distribuidas
Docente:	Ing. José Caiza, Mg.

II. INFORME DE PRÁCTICA

2.1 Objetivos

General:

Desarrollar y desplegar una API utilizando el framework .NET en un entorno Windows, y configurar su ejecución en una máquina virtual con sistema operativo Ubuntu, permitiendo su acceso desde ambos entornos a través de una red punto a punto.

Específicos:

- Diseñar y construir una API funcional utilizando .NET sobre el sistema operativo Windows.
- Configurar una máquina virtual Ubuntu y establecer la conectividad adecuada para el despliegue de la API.
- Verificar el funcionamiento y la accesibilidad de la API tanto desde el entorno Windows como desde Ubuntu, asegurando su correcto despliegue.

2.2 Resultados obtenidos

En el desarrollo del proyecto se construyó un API RESTful utilizando ASP.NET Web API, empleando SQL Server como sistema gestor de base de datos y Entity Framework Core como ORM para facilitar el acceso y manipulación de los datos. Se implementaron los controladores AuthController, ClientesController y PedidosController, cada uno basado en sus respectivos modelos de dominio, permitiendo gestionar la autenticación, clientes y pedidos de manera estructurada. La seguridad se abordó mediante autenticación JWT (JSON Web Token), protegiendo los endpoints y restringiendo el acceso solo a usuarios autorizados. En la configuración del archivo `Program.cs`, se definió la conexión a base de datos, la autenticación JWT y la integración con Swagger para documentar y probar la API. Además, se incorporaron Middlewares personalizados, como `RequestLoggingMiddleware` y `ValidationMiddleware`, los cuales permiten registrar solicitudes entrantes y validar datos, mejorando así la trazabilidad y robustez del sistema.



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



```
using Microsoft.EntityFrameworkCore;
using WebApiPerson.Context;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Obtener la cadena de conexión desde el archivo de configuración
var connectionString = builder.Configuration.GetConnectionString("Connection");

// Configurar el DbContext con SQL Server
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(connectionString));

// Configurar la autenticación JWT
builder.Services.AddAuthentication("Bearer")
    .AddJwtBearer(options =>
    {
        var secretKey = builder.Configuration["Jwt:SecretKey"];

        if (string.IsNullOrEmpty(secretKey))
        {
            throw new Exception("La clave secreta JWT no está configurada correctamente.");
        }

        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,

            ValidIssuer = builder.Configuration["Jwt:Issuer"],
            ValidAudience = builder.Configuration["Jwt:Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey))
        };
    });

// Agregar los servicios de controladores
builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
```

```
// Configuración de Swagger para incluir el JWT Bearer
builder.Services.AddSwaggerGen(c =>
{
    c.AddSecurityDefinition("Bearer", new Microsoft.OpenApi.Models.OpenApiSecurityScheme
    {
        In = Microsoft.OpenApi.Models.ParameterLocation.Header,
        Description = "Please enter a valid token",
        Name = "Authorization",
        Type = Microsoft.OpenApi.Models.SecuritySchemeType.ApiKey
    });

    c.AddSecurityRequirement(new Microsoft.OpenApi.Models.OpenApiSecurityRequirement
    {
        {
            new Microsoft.OpenApi.Models.OpenApiSecurityScheme
            {
                Reference = new Microsoft.OpenApi.Models.OpenApiReference
                {
                    Type = Microsoft.OpenApi.Models.ReferenceType.SecurityScheme,
                    Id = "Bearer"
                }
            },
            new string[] { }
        }
    });
});

var app = builder.Build();

// Configurar la canalización de solicitudes HTTP
app.UseSwagger(); // Habilitar Swagger en desarrollo
app.UseSwaggerUI(); // Interfaz de usuario de Swagger

//app.UseHttpsRedirection();

app.UseAuthentication(); // Asegúrate de que la autenticación esté habilitada
app.UseMiddleware<WebApiPerson.Middleware.RequestLoggingMiddleware>();
app.UseMiddleware<ValidationMiddleware>();

app.UseAuthorization(); // Asegúrate de que la autorización esté habilitada
app.MapControllers(); // Mapear los controladores de la API

app.Run();
```

Ilustración 1. Configuración Program.cs para JWT Y Swagger



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



```
using System.Security.Claims;
using System.Text;
using WebApiPerson.Models; // Asegúrate de tener acceso a tu modelo de usuarios

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

        public AuthController(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        [HttpPost("login")]
        public IActionResult Login([FromBody] Login login)
        {
            // Validar las credenciales (esto debería ser reemplazado por un servicio real de autenticación)
            if (login.Username == "admin" && login.Password == "password123")
            {
                var claims = new[]
                {
                    new Claim(ClaimTypes.Name, login.Username),
                    new Claim(ClaimTypes.Role, "User")
                };

                var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:SecretKey"]));
                var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

                var token = new JwtSecurityToken(
                    issuer: _configuration["Jwt:Issuer"],
                    audience: _configuration["Jwt:Audience"],
                    claims: claims,
                    expires: DateTime.Now.AddHours(1),
                    signingCredentials: creds
                );

                return Ok(new
                {
                    Token = new JwtSecurityTokenHandler().WriteToken(token)
                });
            }

            return Unauthorized("Invalid credentials");
        }
    }
}
```

Ilustración 2. Controller para Autenticación con JWT

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

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

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

        // GET: api/Clientes
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Cliente>>> GetCliente()
        {
            return await _context.Cliente.ToListAsync();
        }

        // GET: api/Clientes/5
        [HttpGet("{id}")]
        public async Task<ActionResult<Cliente>> GetCliente(int id)
        {
            var cliente = await _context.Cliente.FindAsync(id);

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

            return cliente;
        }

        // PUT: api/Clientes/5
        // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
    }
}
```

Ilustración 3. Controller para Clientes



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



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

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

        // GET: api/Pedidos
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Pedido>>> GetPedido()
        {
            return await _context.Pedido.ToListAsync();
        }

        // GET: api/Pedidos/5
        [HttpGet("{id}")]
        public async Task<ActionResult<Pedido>> GetPedido(int id)
        {
            var pedido = await _context.Pedido.FindAsync(id);

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

            return pedido;
        }

        // PUT: api/Pedidos/5
        // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPut("{id}")]
        public async Task<IAActionResult> PutPedido(int id, Pedido pedido)
        {
            if (id != pedido.Id)
            {
                return BadRequest();
            }

            _context.Entry(pedido).State = EntityState.Modified;
        }
    }
}
```

Ilustración 4. Controller para Pedidos

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "Connection": "Server=MELANIES;Database=Person;User Id=api_user;Password=Password1231;TrustServerCertificate=True;" //conexion a SQL server con usuario para Ubuntu
  },
  "Jwt": {
    "SecretKey": "YourSecretKeyForAuthenticationOfMinimumLength256Bits",
    "Issuer": "YourIssuer",
    "Audience": "YourAudience",
    "ExpiryInMinutes": 60
  },
  "Hosts": {
    "Endpoints": {
      "Http": {
        "Url": "http://0.0.0.0:5000" // Escucha en todas las interfaces
      }
    }
  },
  "Urls": "http://0.0.0.0:5000"
}
```

Ilustración 5. Configuración de appsettings.json para conexión a SQL SERVER y acceso desde todas las rutas

Una vez finalizado el desarrollo del API, se procedió a generar el publicable desde el entorno de desarrollo Visual Studio 2022, utilizando la herramienta integrada de publicación. Para ello, se hizo clic derecho sobre el proyecto en el Explorador de soluciones y se seleccionó la opción "Publicar". A continuación, se eligió destino de publicación la opción "Carpeta", especificando una ruta local donde se generarían los archivos necesarios para el despliegue. Se seleccionó el modo de compilación Release y se completó el proceso haciendo clic en el botón "Publicar", en el cual la



configuración de este archivo tiene que ser Release y el tiempo de ejecución de destino Linux-x64. Visual Studio compiló el proyecto y generó una carpeta con todos los archivos requeridos para su ejecución, incluyendo el archivo ejecutable (.dll), bibliotecas y configuraciones. Este conjunto de archivos fue posteriormente transferido al servidor Ubuntu para su ejecución, permitiendo así el despliegue del API en un entorno Linux de manera eficiente y organizada.

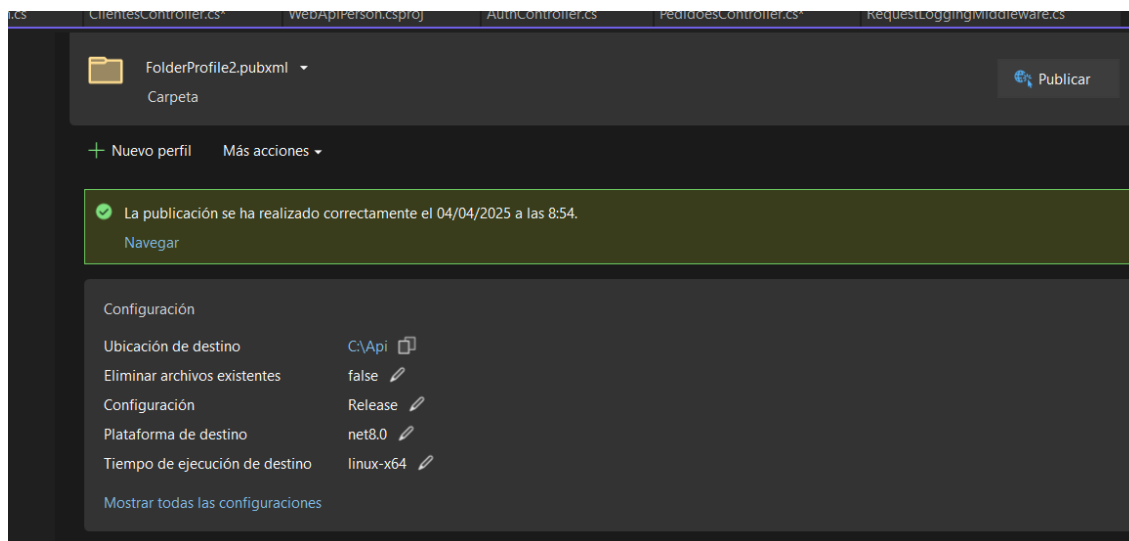


Ilustración 6. Creación y configuración de publicable de API REST FULL

Para facilitar la transferencia del publicable generado en Visual Studio en Windows hacia el servidor Ubuntu, se configuró una carpeta compartida utilizando el servicio Samba. Primero, se instaló Samba en Ubuntu y se creó una carpeta para compartir, a la cual se le otorgaron los permisos necesarios. Luego, se configuró Samba editando su archivo de configuración para definir la ruta de la carpeta compartida, permitiendo el acceso desde Windows. Para asegurar la autenticación, se configuró una contraseña para el usuario de Samba. Posteriormente, se reinició el servicio de Samba para aplicar los cambios. Desde Windows, se accedió a la carpeta compartida a través del Explorador de Archivos utilizando la dirección IP del servidor Ubuntu. De esta manera, los archivos generados por Visual Studio fueron transferidos al servidor Ubuntu para proceder con su ejecución.



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



```
Actividades Terminal 31 de mar 16:25 melanie@melanie-VB: ~
melanie@melanie-VB:~$ mkdir ~/compartido
melanie@melanie-VB:~$ chmod 755 ~/compartido1
chmod: no se puede acceder a '/home/melanie/compartido1': No existe el archivo
o el directorio
melanie@melanie-VB:~$ ls
compartido Documentos Imágenes Plantillas Videos
Descargas Escritorio Música Público
melanie@melanie-VB:~$ chmod 755 ~/compartido
chmod: no se puede acceder a '/home/melanie/compartido': No existe el archivo o
el directorio
melanie@melanie-VB:~$ chmod 755~/compartido
chmod: falta un operando después de «755~/compartido»
Pruebe 'chmod --help' para más información.
melanie@melanie-VB:~$ chmod 755~/compartido1
chmod: falta un operando después de «755~/compartido1»
Pruebe 'chmod --help' para más información.
melanie@melanie-VB:~$ chmod 755~/compartido
chmod: falta un operando después de «755~/compartido»
Pruebe 'chmod --help' para más información.
melanie@melanie-VB:~$ chmod 755 ~/compartido
chmod: no se puede acceder a '/home/melanie/compartido': No existe el archivo o
el directorio
melanie@melanie-VB:~$ ls
compartido Documentos Imágenes Plantillas Videos
Descargas Escritorio Música Público
melanie@melanie-VB:~$ chmod 755 ~/compartido
melanie@melanie-VB:~$ sudo nano etc/samba/smb.conf
[sudo] contraseña para melanie:
melanie@melanie-VB:~$ sudo nano /etc/samba/smb.conf
```

Ilustración 7. Creación de carpeta compartida en Ubuntu

```
GNU nano 4.8 /etc/samba/smb.conf
printable = yes
guest ok = no
read only = yes
create mask = 0700

# Windows clients look for this share name as a source of downloadable
# printer drivers
[print$]
comment = Printer Drivers
path = /var/lib/samba/printers
browseable = yes
read only = yes
guest ok = no
# Uncomment to allow remote administration of Windows print drivers.
# You may need to replace 'lpadmin' with the name of the group your
# admin users are members of.
# Please note that you also need to set appropriate Unix permissions
# to the drivers directory for these users to have write rights in it
; write list = root, @lpadmin
[compartido]
comment= Carpeta Compartida en Ubuntu
path= /home/melanie/compartido
read only = no
guest ok = yes

^G Ver ayuda ^O Guardar ^W Buscar ^K Cortar Texto ^J Justificar
^X Salir ^R Leer fich. ^_ Reemplazar ^U Pegar ^T Ortografía
```

Ilustración 8. Configuración para Samba

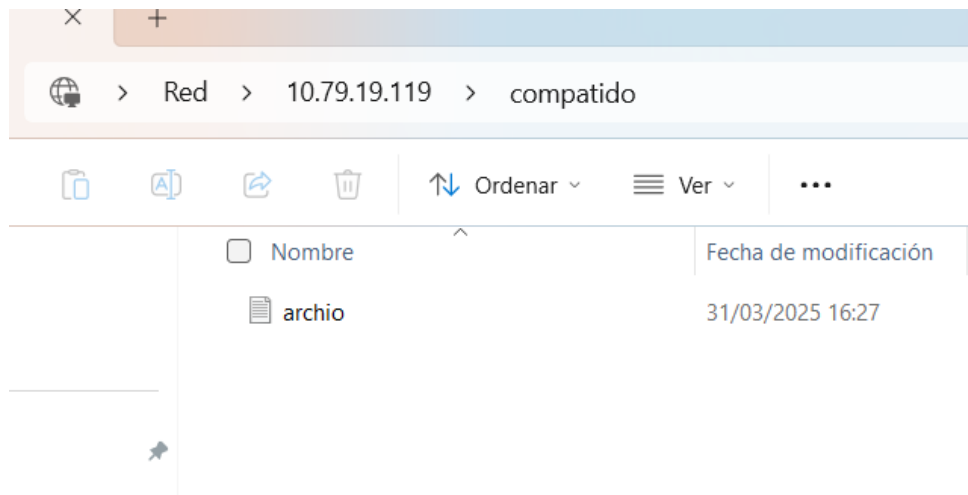


Ilustración 9. Carpeta compartida en el explorador de archivos de Windows

Una vez configurada la carpeta compartida entre Windows y Ubuntu, se procedió a transferir los archivos del publicable generado desde Visual Studio hacia la carpeta compartida en el servidor Ubuntu. Esta transferencia se realizó simplemente copiando los archivos del publicable desde el equipo con Windows hacia la ruta de la carpeta compartida previamente configurada, permitiendo que los archivos estuvieran disponibles en el entorno Ubuntu para su posterior ejecución.

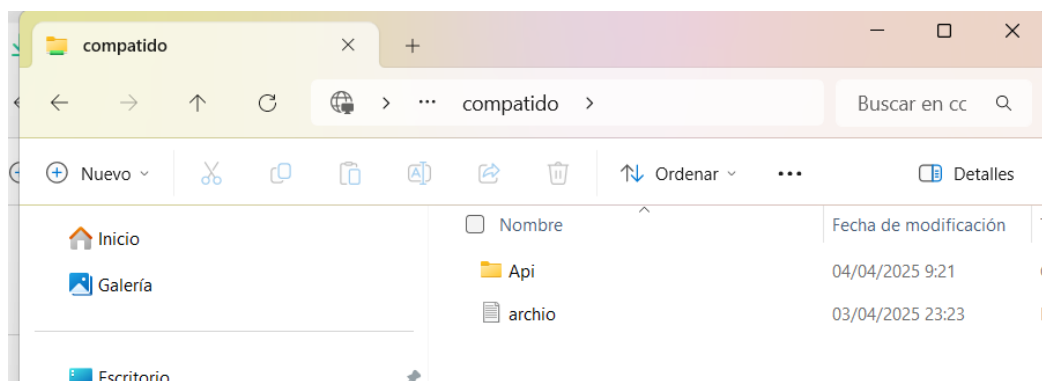


Ilustración 10. Transferencia de archivo de API REST FULL a carpeta compartida

Para la instalación de la API en Ubuntu, primero se creó el directorio `/var/www/webapiperson` utilizando el comando `sudo mkdir -p`, asegurando la estructura necesaria para alojar los archivos. A continuación, se copiaron todos los contenidos de la carpeta `~/compartido/Api/*` hacia el nuevo directorio con el comando `sudo cp -r`, garantizando que los componentes multimedia, como las imágenes, estuvieran disponibles para el correcto funcionamiento de la interfaz o cualquier funcionalidad dependiente. Este proceso preparó el entorno para la posterior configuración y despliegue de la API en el sistema.



The image shows two terminal windows. The top window displays the command `sudo mkdir -p /var/www/webapiperson`. The bottom window displays the command `sudo cp -r ~/compatido/Api/* /var/www/webapiperson`.

Ilustración 11. Transferencia del archivo API desde la carpeta compartida a un directorio de instalación en Ubuntu

Primero se descargó el paquete oficial de Microsoft para configurar los repositorios (.deb) usando `wget`, lo que permite acceder a las actualizaciones de .NET. Luego, con `dpkg -i` se instaló este paquete para habilitar dichos repositorios en el sistema. Después, `apt update` sincronizó los paquetes disponibles con los nuevos repositorios añadidos. Finalmente, `apt install` agregó el ASP.NET Core Runtime 8.0, que proporciona todo lo necesario para ejecutar aplicaciones .NET sin necesidad del SDK completo. Cada paso fue esencial para preparar el entorno antes de desplegar la API.

The image shows four terminal windows. The first window shows the command `wget https://packages.microsoft.com/config/ubuntu/20.4/packages-microsoft-prod.deb -O packages-microsoft-prod.deb`. The second window shows `sudo dpkg -i packages-microsoft-prod.deb`. The third window shows `sudo apt update`. The fourth window shows `sudo apt install -y aspnetcore-runtime-8.0`.

Ilustración 12. Configuración del entorno de .NET en Ubuntu para el despliegue de la API

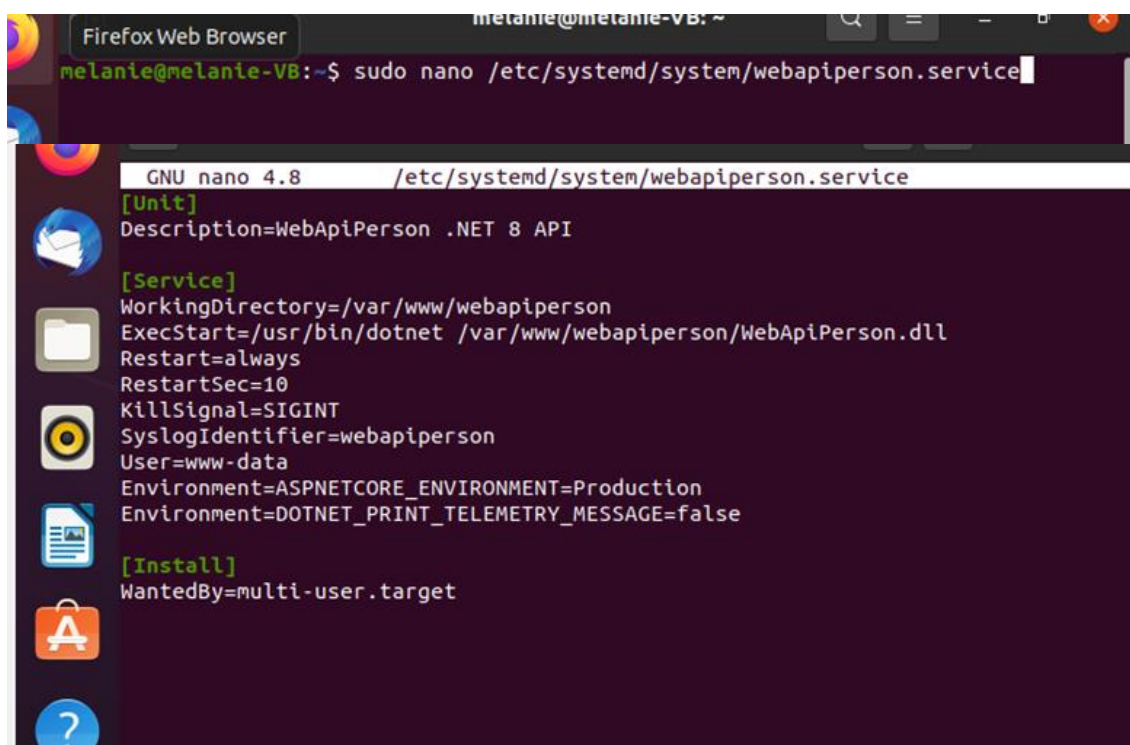
Se establecieron los permisos adecuados para el directorio de la API ejecutando dos comandos esenciales. Primero, `sudo chown -R www-data:www-data /var/www/webaptperson` asignó la propiedad recursiva al usuario y grupo `www-data`, asegurando que el servidor web (como Apache o Nginx) tuviera control total sobre los archivos. Segundo, `sudo chmod -R 755 /var/www/webaptperson` configuró los permisos de acceso, dando al propietario (`www-data`) permisos de lectura, escritura y ejecución (7), mientras que a

grupos y otros usuarios se les concedió solo lectura y ejecución (5). Esta configuración garantiza seguridad y funcionalidad, permitiendo al servidor web operar correctamente mientras restringe accesos no autorizados.

```
melanie@melanie-VB:~$ sudo chown -R www-data:www-data /var/www/webapiperson
melanie@melanie-VB:~$ sudo chmod -R 755 /var/www/webapiperson
```

Ilustración 13. Configuración de permisos para el directorio de la API en Ubuntu

Para garantizar el funcionamiento continuo de la API, se creó un servicio systemd en `/etc/systemd/system/webapiperson.service` con la configuración esencial: se especificó el directorio de trabajo `/var/www/webapiperson`, el comando de inicio `/usr/bin/dotnet /var/www/webapiperson/WebApiPerson.dll`, y se configuró para reiniciarse automáticamente (`Restart=always`) tras 10 segundos en caso de fallos (`RestartSec=10`). El servicio opera bajo el usuario `www-data` para mantener coherencia con los permisos del servidor web, y se definieron variables de entorno para producción (`ASPNETCORE_ENVIRONMENT=Production`) y desactivación de telemetría (`DOTNET_PRINT_TELEMETRY_MESSAGE=false`). Además, se vinculó al target `multi-user.target` para asegurar su inicio con el sistema, permitiendo gestionar la API como un servicio nativo mediante systemd (ej. `sudo systemctl start webapiperson`).



```
melanie@melanie-VB:~$ sudo nano /etc/systemd/system/webapiperson.service
GNU nano 4.8 /etc/systemd/system/webapiperson.service
[Unit]
Description=WebApiPerson .NET 8 API

[Service]
WorkingDirectory=/var/www/webapiperson
ExecStart=/usr/bin/dotnet /var/www/webapiperson/WebApiPerson.dll
Restart=always
RestartSec=10
KillSignal=SIGINT
SyslogIdentifier=webapiperson
User=www-data
Environment=ASPNETCORE_ENVIRONMENT=Production
Environment=DOTNET_PRINT_TELEMETRY_MESSAGE=false

[Install]
WantedBy=multi-user.target
```

Ilustración 14. Creación y configuración de un servicio systemd para la API en Ubuntu

Para asegurar que la API se inicie automáticamente al arrancar el sistema, se ejecutó `sudo systemctl enable webapiperson.service`, lo que



habilita el servicio creado previamente. Inmediatamente después, con `sudo systemctl start webapiperson.service`, se inició manualmente el servicio para poner en funcionamiento la API sin necesidad de reiniciar el servidor. Estos comandos garantizan que la aplicación esté disponible de forma permanente y se reinicie correctamente ante cualquier reinicio del sistema.

```
melanie@melanie-VB:~$ sudo systemctl enable webapiperson.service  
  
melanie@melanie-VB:~$ sudo systemctl start webapiperson.service
```

Ilustración 15. Habilitación y arranque automático de la API con systemd en Ubuntu

Para permitir el tráfico hacia el puerto 5000 (utilizado por la API), se ejecutó `sudo ufw allow 5000/tcp`, abriendo específicamente este puerto en el firewall. A continuación, se activó el cortafuegos con `sudo ufw enable` para aplicar las reglas configuradas y asegurar que solo el tráfico autorizado pueda acceder al sistema, protegiendo así el servidor mientras mantiene accesible el servicio de la API.

```
melanie@melanie-VB:~$ sudo ufw allow 5000/tcp  
  
s Terminal ▾ abr 3 15:54  
  
melanie@melanie-VB:~$ sudo ufw enable
```

Ilustración 16. Configuración del firewall para permitir el tráfico hacia el puerto 5000 de la API en Ubuntu

Para asegurar que la API pueda conectarse correctamente a SQL Server, fue necesario habilitar la comunicación a través del puerto 1433 en el firewall del servidor. Esto se logró ejecutando una regla de firewall en Windows con el comando `New-NetFirewallRule`, permitiendo el tráfico entrante en el puerto TCP 1433, utilizado por SQL Server. Posteriormente, se verificó la conectividad al servidor utilizando el comando `telnet`, que intentó establecer una conexión al puerto 1433 de la dirección IP del servidor SQL. La respuesta de "connected to 192.168.3.225" confirmó que la conexión al servidor SQL se había establecido correctamente, asegurando que la API pudiera interactuar con la base de datos sin problemas.



```
PS C:\Users\Melan> New-NetFirewallRule -DisplayName "SQL Server" -Direction Inbound -Protocol TCP -LocalPort 1433 -Action Allow

Name                : {c7c448d6-2ceb-40b9-8590-4d7a5fcf6fa4}
DisplayName          : SQL Server
Description          :
DisplayGroup         :
Group                :
Enabled              : True
Profile              : Any
Platform             : {}
Direction            : Inbound
Action               : Allow
EdgeTraversalPolicy  : Block
LooseSourceMapping   : False
LocalOnlyMapping     : False
Owner                :
PrimaryStatus        : OK
Status               : Se analizó la regla correctamente desde el almacén. (65536)
EnforcementStatus    : NotApplicable
PolicyStoreSource    : PersistentStore
PolicyStoreSourceType : Local
RemoteDynamicKeywordAddresses : {}
PolicyAppId          :
```

Ilustración 17. Configuración del firewall en Windows para permitir la conexión a SQL Server desde la API

```
melanie@melanie-VB:~$ telnet 192.168.3.225 1433
Trying 192.168.3.225...
Connected to 192.168.3.225.
Escape character is '^['
```

Ilustración 18. Verificación de conexión de Ubuntu a SQL Server

Para asegurar la correcta resolución de nombres entre el servidor Ubuntu y SQL Server, se editó el archivo `/etc/hosts` en Ubuntu, agregando una entrada que asocia la dirección IP del servidor SQL con su nombre de host. Al abrir el archivo `/etc/hosts` con el editor de texto nano, se añadió la siguiente línea: `192.168.3.175 MELANIE`, donde MELANIE es el nombre del servidor de la base de datos SQL Server, y `192.168.3.175` es la dirección IP correspondiente. Esto permitió que el servidor Ubuntu pudiera reconocer y acceder correctamente al servidor SQL a través del nombre de host, facilitando la conexión a la base de datos desde la API desplegada en Ubuntu. Además, para garantizar que la IP de MELANIE se mantenga actualizada en caso de cambios, se creó un script llamado `update_sql_host.sh` en `/usr/local/bin`. Este script realiza un ping al servidor MELANIE para obtener su IP, elimina la entrada anterior en `/etc/hosts` y agrega la nueva IP junto al nombre de host. El script fue configurado para ejecutarse automáticamente cada 5 minutos mediante una tarea programada con cron, asegurando que la API siempre pueda acceder al servidor SQL con la IP correcta.

```
Connection closed by foreign host.
melanie@melanie-VB:~$ sudo nano /etc/hosts
[sudo] password for melanie:
melanie@melanie-VB:~$
```

Ilustración 19. Configuración de resolución de nombres en Ubuntu mediante el archivo `/etc/hosts` para SQL Server



Para habilitar la conexión remota al servidor SQL Server desde Ubuntu, primero se debe acceder a SQL Server Management Studio (SSMS) y hacer clic derecho sobre el servidor, seleccionando "Propiedades". En la ventana de propiedades, se debe ir a la sección de Seguridad y elegir la opción de autenticación SQL Server y Windows para permitir el acceso mediante ambos métodos. A continuación, se debe ir a Conexiones y marcar la opción Allow remote connections to this server, lo que permite conexiones externas. Luego, en el Administrador de configuración de SQL Server, se debe activar el protocolo TCP/IP en la configuración de redes, asegurándose de que todas las conexiones estén habilitadas. Finalmente, se debe verificar que la conexión esté utilizando el puerto 1433, que es el puerto predeterminado para las conexiones de SQL Server, lo que garantiza que el servidor sea accesible desde otros dispositivos en la red.

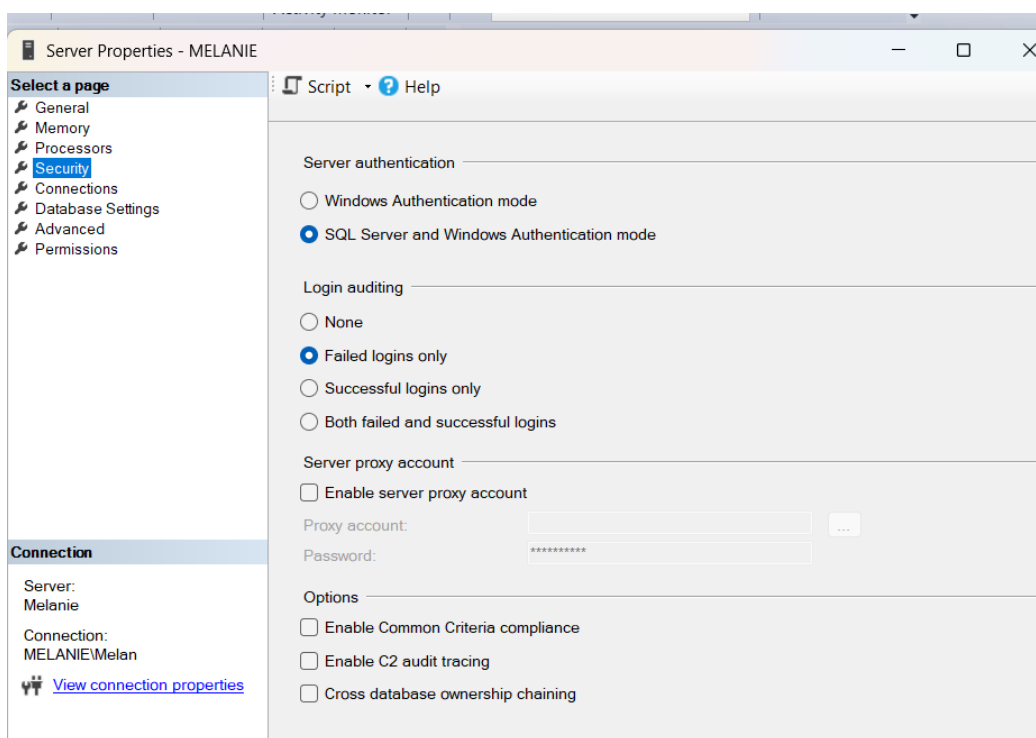


Ilustración 24. Configuración de la autenticación en SQL Server para habilitar conexiones remotas



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

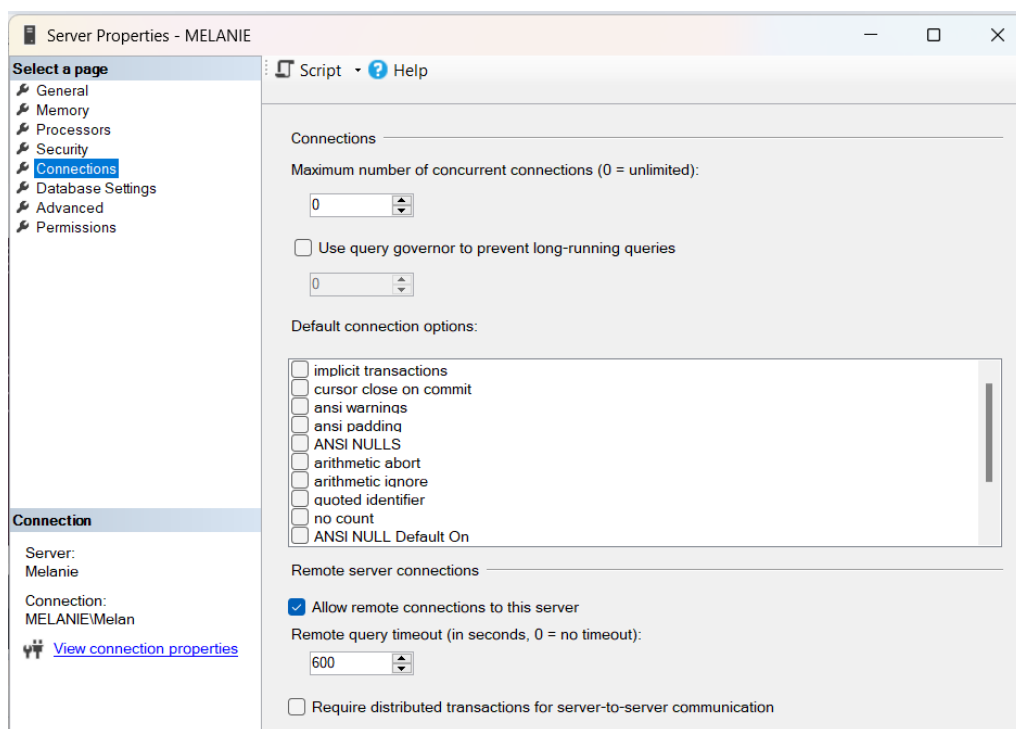


Ilustración 25. Con Habilitación de conexiones remotas en SQL Server para acceso externo

Protocol Name	Status
Shared Memory	Enabled
Named Pipes	Disabled
TCP/IP	Enabled

Ilustración 26. Activación del protocolo TCP/IP en SQL Server para permitir conexiones de red

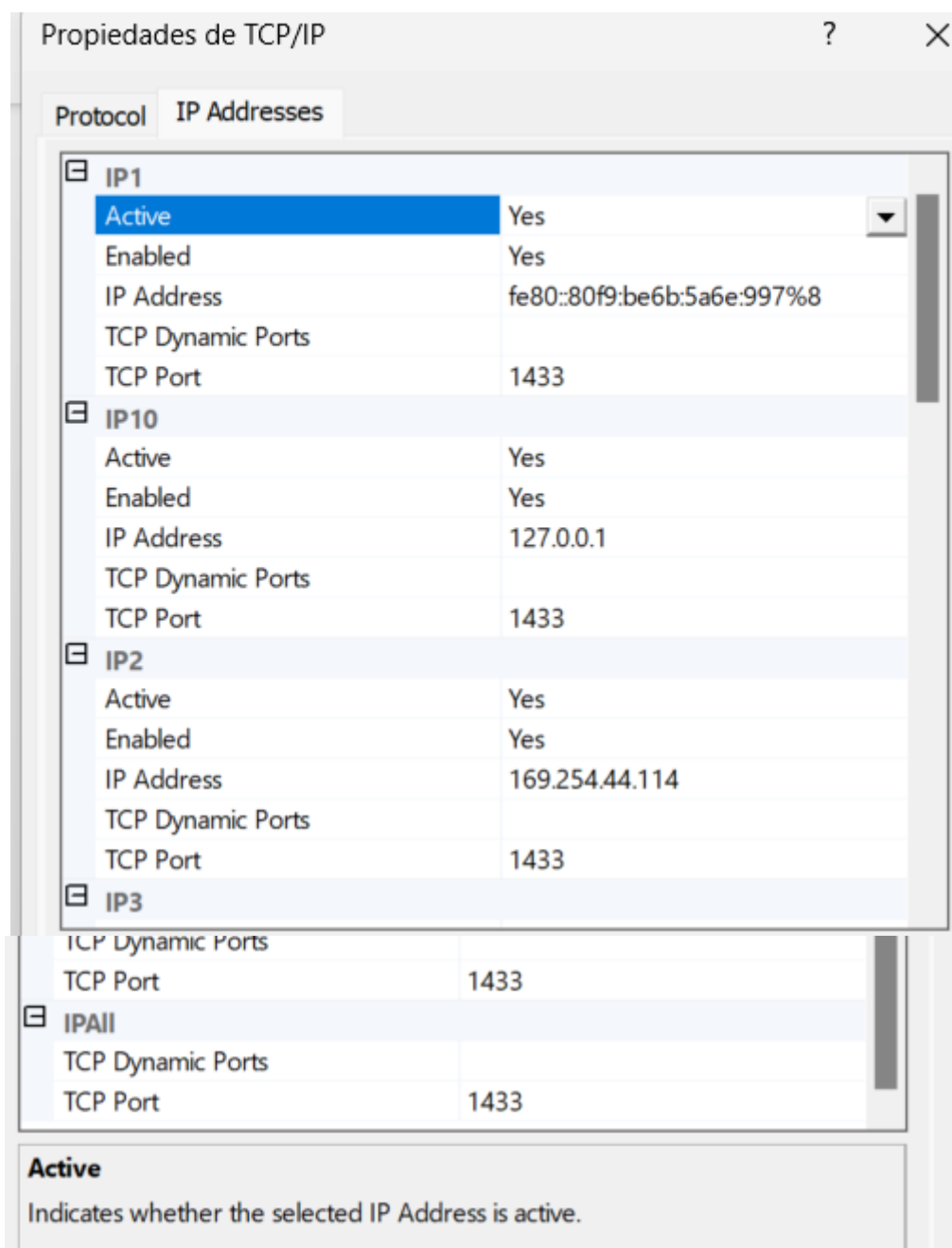


Ilustración 27. Verificación del puerto 1433 para conexiones remotas en SQL Server

Para interactuar con SQL Server desde Ubuntu, se instalaron las herramientas necesarias utilizando el comando `sudo apt install mssql-tools unixodbc-dev`, que incluye `mssql-tools` y `unixodbc-dev`. Estas herramientas permiten ejecutar comandos y consultas SQL directamente desde la terminal en Ubuntu. Luego, para verificar la conexión al servidor SQL y realizar una consulta, se utilizó el comando `sqlcmd`, especificando el nombre del servidor MELANIE, el usuario `api_user` y la contraseña 'Password123!'. La opción `-Q "SELECT * FROM PEDIDO"` ejecuta una consulta SQL que selecciona todos los registros de la tabla PEDIDO, lo que permite verificar que la conexión fue exitosa y que la base de datos está accesible para consultas desde el servidor Ubuntu.



```
melanie@melanie-VB:~$ sudo apt install -y mssql-tools unixodbc-dev
```

Ilustración 28. Instalación de herramientas necesarias para interactuar con SQL Server en Ubuntu

```
melanie@melanie-VB:~$ sqlcmd -S MELANIE -U api_user -P 'Password123!' -d Person
-Q "SELECT * FROM PEDIDO"
Id          Producto
-----
1 zapato
5
(1 rows affected)
melanie@melanie-VB:~$ s
```

Ilustración 29. Verificación de la conexión a SQL Server desde Ubuntu utilizando sqlcmd

Finalmente, una vez completados todos los pasos de configuración, se pudo acceder a la API desplegada en Ubuntu desde ambos sistemas, tanto desde Windows como desde Ubuntu, utilizando la dirección IP del servidor Ubuntu. Ingresando a <http://192.168.3.225:5000/swagger/index.html> en el navegador, se accede a la interfaz de Swagger, que permite interactuar con la API y realizar pruebas de sus endpoints de forma visual y sencilla. Esta dirección IP y puerto garantizan que la API esté disponible y accesible desde cualquier dispositivo dentro de la red, facilitando la interacción y verificación de las funcionalidades de la API.

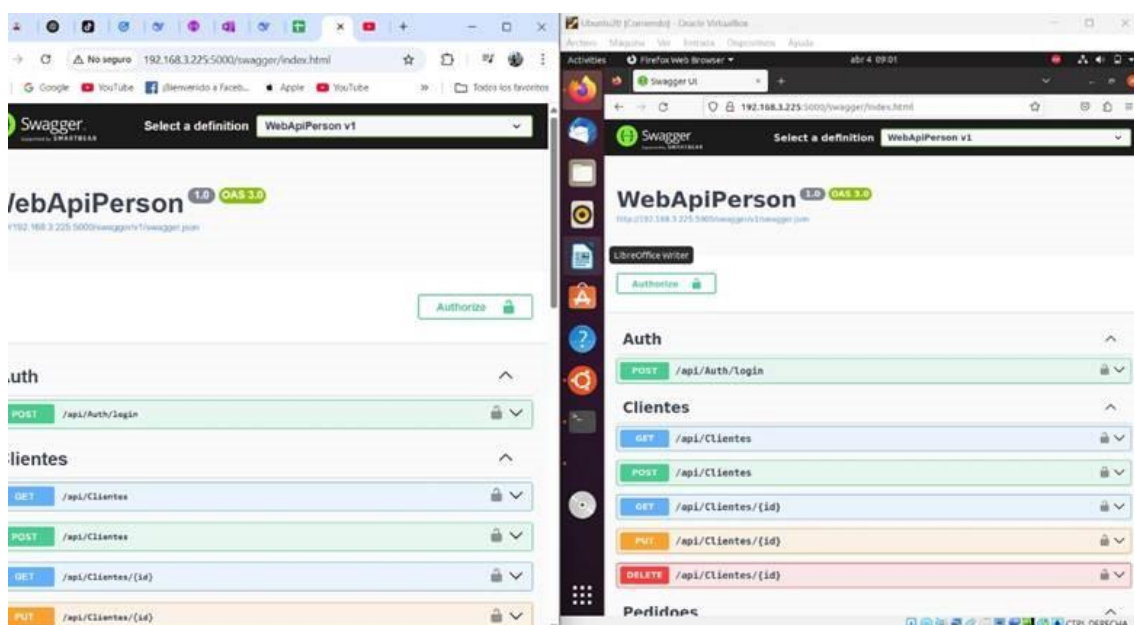


Ilustración 30. Acceso a la API desplegada en Ubuntu desde sistemas Windows y Ubuntu a través de Swagger

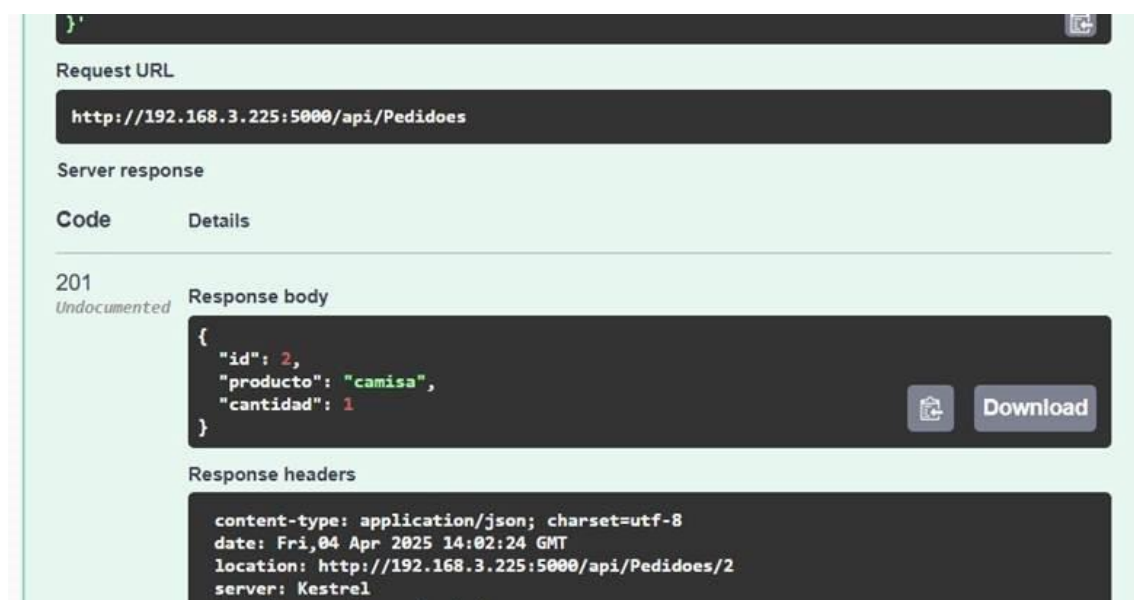


Ilustración 31. Comprobación de inserción de datos a la base mediante la API

2.3 Conclusiones

El desarrollo y despliegue de la API en .NET permitió comprobar que una aplicación construida en un entorno Windows puede ser ejecutada satisfactoriamente en un sistema operativo Ubuntu dentro de una máquina virtual, siempre que se configure adecuadamente el entorno de ejecución y la red.

Durante el proceso se evidenció la interoperabilidad entre plataformas, destacando la flexibilidad que ofrece .NET al ser multiplataforma. Además, se logró establecer una red punto a punto entre la máquina anfitriona (Windows) y la máquina virtual (Ubuntu), permitiendo acceder a la API desde ambos sistemas, lo cual valida la correcta implementación y configuración del despliegue.



2.4 Recomendaciones

Se recomienda verificar que tanto el entorno de desarrollo como el de despliegue cuenten con versiones compatibles de .NET, además de configurar correctamente los puertos y adaptadores de red en la máquina virtual para asegurar la visibilidad del servicio. Es importante documentar detalladamente cada paso del proceso de despliegue, ya que esto facilita futuras implementaciones y solución de errores.

2.5 Referencias bibliográficas

<https://github.com/melanieAlban/aplicaciones-distribuidas.git>