UNIVERSIDAD TÉCNICA DE AMBATO
Facultad de Ingeniería en Sistemas, Electrónica e Industrial
Marzo 2025 – Julio 2025

Título:	Diseño Cliente Servidor
Carrera:	"Software"
Nivel y Paralelo:	6°to "A"
Estudiantes participantes:	Kevin Carrasco
Asignatura:	Aplicaciones Distribuidas
Docente:	Ing. José Rubén Caiza Caizabuano, Mg.
Fecha:	6/04/2025

1. Introducción

En el desarrollo de software moderno, las APIs RESTful juegan un papel fundamental como puente de comunicación entre aplicaciones y servicios. ASP.NET Core se ha consolidado como una de las tecnologías más robustas y eficientes para construir estas interfaces, gracias a su alto rendimiento, modularidad y compatibilidad multiplataforma.

Este informe presenta el proceso de desarrollo de una API RESTful implementada en ASP.NET Core, que gestiona entidades fundamentales como Persona, Usuario, Cliente y Pedido. Además, se incorporó un mecanismo de autenticación basado en JSON Web Tokens (JWT) para asegurar los endpoints, junto con su integración en Swagger para facilitar pruebas seguras durante el desarrollo.

2. Marco Teórico

Una **API RESTful (Representational State Transfer)** es una interfaz de comunicación que sigue los principios de arquitectura REST, permitiendo acceder y manipular recursos a través de métodos HTTP como GET, POST, PUT y DELETE. Este tipo de APIs son ampliamente utilizadas en el desarrollo de servicios web por su simplicidad, escalabilidad y facilidad de integración.

ASP.NET Core es un framework de desarrollo web de código abierto y multiplataforma de Microsoft. Es especialmente adecuado para construir APIs modernas y de alto rendimiento. Cuenta con un potente sistema de inyección de dependencias, manejo de middleware y soporte nativo para Entity Framework Core, que permite mapear entidades a una base de datos relacional de manera eficiente.

La **autenticación JWT (JSON Web Token)** es un método seguro y sin estado para autenticar usuarios. El servidor genera un token firmado digitalmente que el cliente debe enviar con cada solicitud a rutas protegidas. Este token contiene información (claims) que puede ser verificada sin necesidad de mantener sesiones en el servidor.

Swagger, por su parte, es una herramienta ampliamente adoptada para documentar y probar APIs REST. Su integración con autenticación JWT permite simular fácilmente

llamadas a endpoints protegidos, facilitando así el proceso de desarrollo y validación de seguridad.

3. Desarrollo del Proyecto

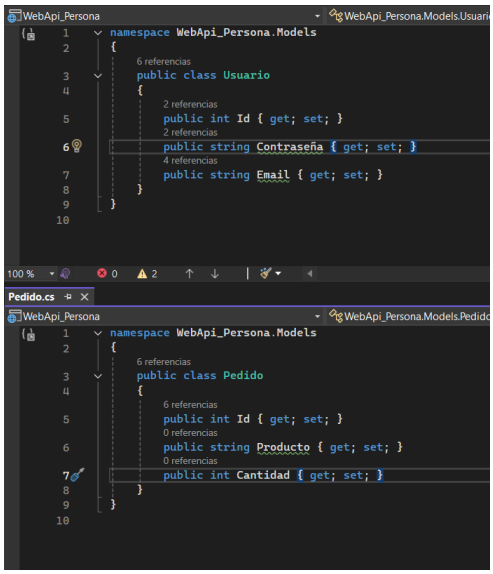
El desarrollo de esta API se estructuró en varias etapas clave:

3.1. Modelado de Entidades

Se diseñaron cuatro entidades principales:

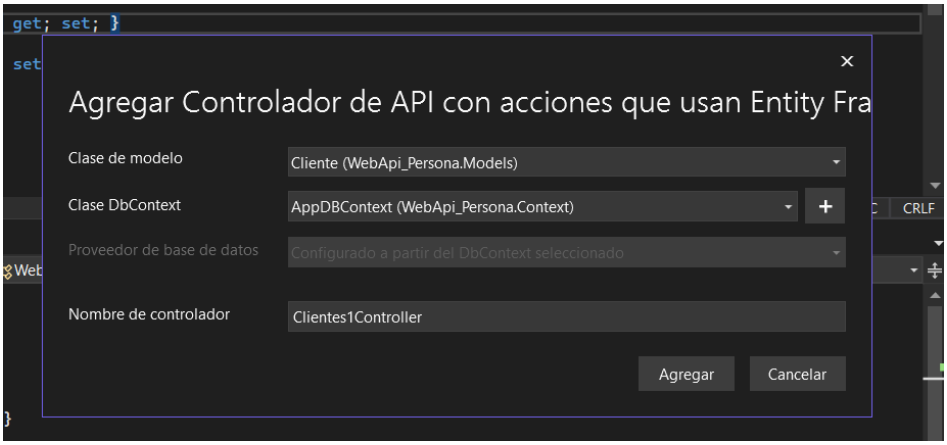
- **Persona**: almacena datos personales generales.
- **Usuario**: representa a quienes acceden al sistema, con credenciales y roles.
- **Cliente**: se relaciona con persona, incluyendo datos específicos para quienes realizan pedidos.
- **Pedido**: contiene los registros de compras o solicitudes realizadas por clientes.

Estas entidades se configuraron mediante Entity Framework Core, generando relaciones adecuadas y permitiendo operaciones CRUD completas.



3.2. Generación de Controladores

Para cada entidad, se generaron controladores tipo ApiController con soporte para operaciones estándar como creación, lectura, actualización y eliminación. Esta automatización mediante scaffolding permitió un desarrollo más ágil y consistente.



3.3. Implementación de Autenticación JWT

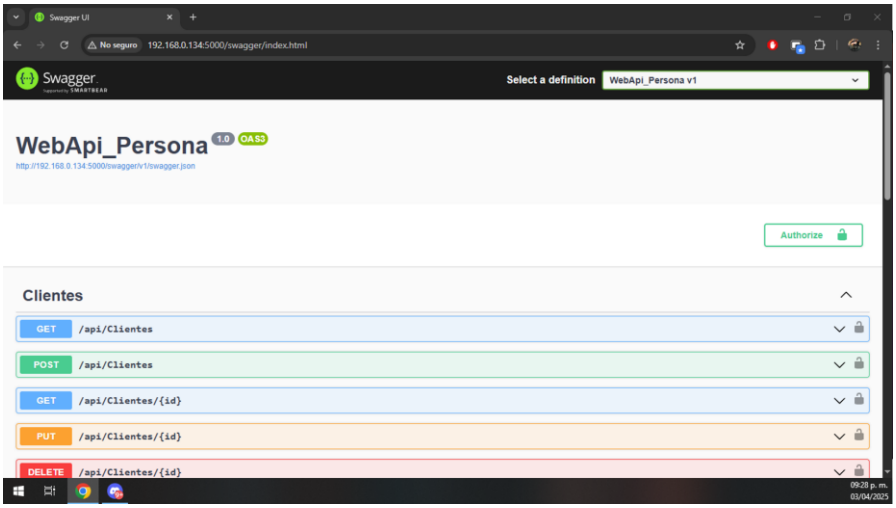
Se implementó un sistema de autenticación mediante JWT. Para ello, se creó una clase denominada TokenProvider, responsable de generar los tokens con información del usuario y su rol. Este token es necesario para acceder a rutas protegidas, garantizando así que solo usuarios autenticados puedan operar sobre los recursos del sistema.

```
1 referencia
internal sealed class TokenProvider(IConfiguration configuration)
{
    1 referencia
    public string Create(Usuario usuario)
    {
        string secretKey = configuration["Jwt:Secret"];
        var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));
        var credentials= new SigningCredentials(securityKey,SecurityAlgorithms.HmacSha256);
        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(new[] {
                new Claim(JwtRegisteredClaimNames.Sub,usuario.Id.ToString()),
                new Claim(JwtRegisteredClaimNames.Email,usuario.Email.ToString())
                }),
            Expires = DateTime.UtcNow.AddMinutes(configuration.GetValue<int>("Jwt:TiempoExpira")),
            SigningCredentials = credentials,
            Audience = configuration["Jwt:Audience"],
            Issuer = configuration["Jwt:Issuer"]
        };
        var handler = new JsonWebTokenHandler();
        var token = handler.CreateToken(tokenDescriptor);
        return token;

    }
}
```

3.4. Integración con Swagger

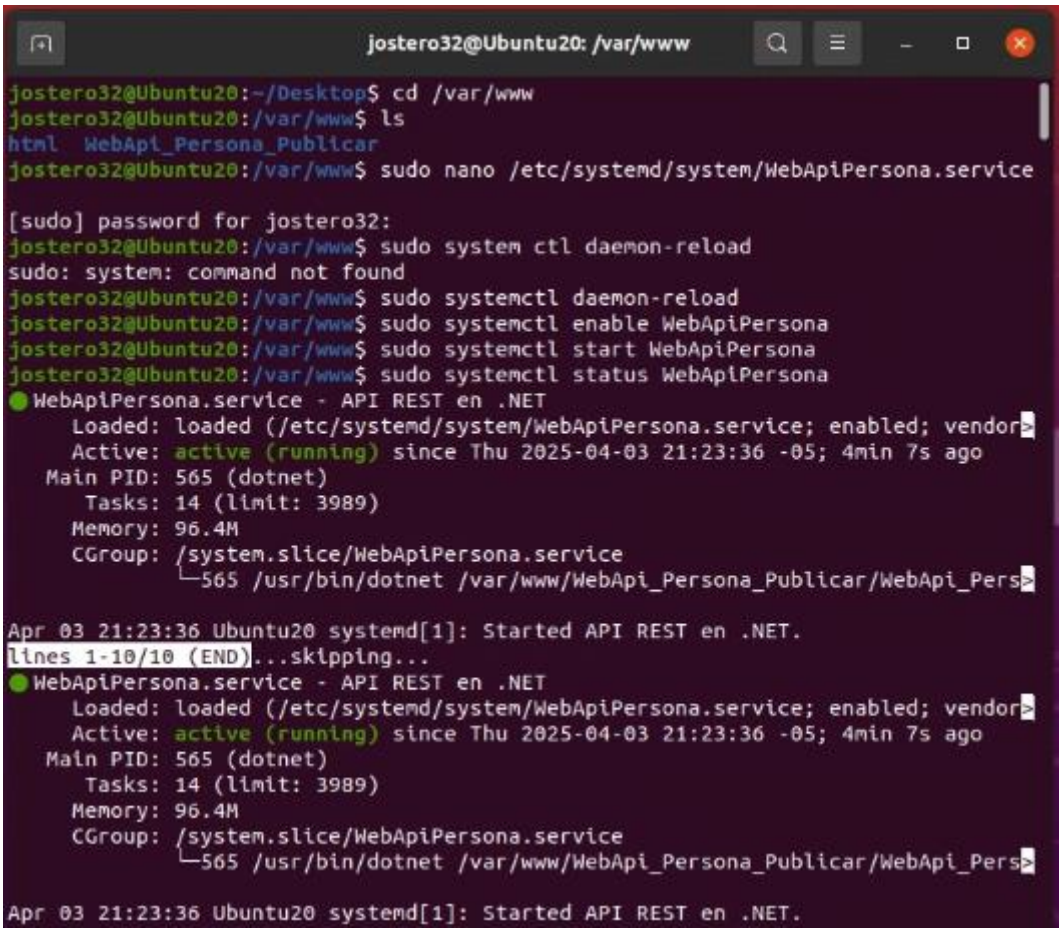
Swagger se configuró para soportar autenticación JWT. Esto permitió realizar pruebas directamente desde la interfaz gráfica, ingresando el token en el encabezado de autorización para simular solicitudes autenticadas a la API. Esta funcionalidad mejoró la experiencia de desarrollo y permitió una validación eficiente de la seguridad.



3.5. Publicación desde Visual Studio y Ejecución en Ubuntu

El proyecto fue **publicado directamente desde Visual Studio**, generando los archivos necesarios para su ejecución. Posteriormente, estos archivos fueron trasladados a un sistema operativo Ubuntu, donde se ejecutó la API utilizando el runtime de .NET.

En el entorno Linux, se creó un **servicio de sistema (demonio)** que permite iniciar la API automáticamente con el sistema operativo y mantenerla funcionando en segundo plano. Esta configuración asegura la disponibilidad permanente del servicio sin necesidad de intervención manual tras cada reinicio del servidor.



4. Resultados

Al finalizar el desarrollo, se obtuvo una API funcional con los siguientes logros:

- Gestión completa de entidades Persona, Usuario, Cliente y Pedido.
- Autenticación robusta basada en JWT, sin necesidad de mantener sesiones en el servidor.
- Protección de rutas sensibles mediante autorización basada en roles.
- Documentación automática de endpoints con Swagger.
- Capacidad de realizar pruebas de endpoints autenticados desde Swagger, gracias a la integración del sistema de tokens.
- Despliegue exitoso en Ubuntu, con ejecución automática como servicio del sistema.

Las pruebas de funcionalidad y seguridad realizadas confirmaron el correcto comportamiento de la API, cumpliendo los objetivos planteados al inicio del proyecto.

5. Conclusiones

La implementación de una API RESTful en ASP.NET Core permitió demostrar la potencia y flexibilidad del framework en el desarrollo de servicios modernos. Gracias a la integración con Entity Framework Core, se logró un mapeo eficiente entre objetos y base de datos. Por otra parte, el uso de autenticación JWT ofreció una solución segura y escalable para proteger los endpoints, mientras que la documentación con Swagger mejoró significativamente la usabilidad y el proceso de pruebas.

El uso de controladores generados automáticamente a partir de las entidades permitió mantener una estructura limpia y coherente, acelerando el desarrollo y garantizando una mejor mantenibilidad del código. En conjunto, esta solución representa una base sólida para cualquier sistema que requiera manejo seguro de usuarios y operaciones CRUD sobre entidades relacionadas.

6. Recomendaciones

Para futuros desarrollos y mejoras sobre esta API, se recomienda lo siguiente:

- **Ampliar la lógica de negocio** utilizando servicios personalizados que separen la lógica del controlador, mejorando la escalabilidad.
- **Implementar roles más específicos y políticas de autorización** para controlar mejor el acceso a cada endpoint.
- **Agregar validaciones personalizadas** en los modelos y controladores para garantizar la integridad de los datos.
- **Utilizar bases de datos en la nube y despliegue en servicios como Azure o Render**, lo que permitirá probar la API en entornos productivos reales.
- **Continuar utilizando Swagger como herramienta de documentación**, asegurando que siempre esté actualizada a medida que evolucione la API.