

**UNIVERSIDAD NACIONAL DE CHIMBORAZO FACULTAD DE INGENIERÍA**

**CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN**

**UNIVERSIDAD NACIONAL DE CHIMBORAZO FACULTAD DE INGENIERÍA**

**INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN**

**NOMBRE:**

**ELVIS PILCO**

**FECHA: 2024/06/12 INTEROPERABILIDAD DE PLATAFORMAS**

**Informe Tecnologías .net servicios REST**

**PERIODO ACADÉMICO 2024-1S**

# OBJETIVO GENERAL, ESPECÍFICOS, RECURSOS

* 1. **OBJETIVO GENERAL**

Analizar y evaluar el uso de tecnologías .NET para el desarrollo de servicios REST, destacando sus ventajas, desventajas, y mejores prácticas, con el fin de proporcionar una guía completa para desarrolladores interesados en implementar soluciones web escalables y eficientes.

# OBJETIVOS ESPECÍFICOS

* Analizar las características y beneficios de la plataforma .NET para el desarrollo de servicios REST
* Explorar las herramientas y frameworks de .NET que facilitan el desarrollo de servicios REST.
* Demostrar cómo implementar un servicio REST utilizando .NET desde cero.
* Evaluar las prácticas recomendadas y patrones de diseño en la creación de servicios REST con .NET

# RECURSOS

* + - Computador
    - Visual Studio
    - Postman
    - Navegador web

# INTRODUCCIÓN.

En la era digital actual, el desarrollo de aplicaciones y servicios web se ha vuelto una piedra angular para la operación, la competitividad y el crecimiento sostenido de las empresas. Las tecnologías .NET y los servicios REST (Representational State Transfer) se han consolidado como componentes esenciales en la arquitectura moderna de software, ofreciendo soluciones robustas y escalables que se adaptan a una amplia gama de necesidades empresariales.

.NET, un marco de desarrollo de software creado por Microsoft, proporciona una plataforma flexible y potente para la construcción de aplicaciones de alto rendimiento. Desde su lanzamiento, .NET ha evolucionado significativamente, incorporando características avanzadas que facilitan el desarrollo, la implementación y el mantenimiento de aplicaciones en diferentes entornos, ya sea en la nube, en servidores locales o en dispositivos móviles. Con un ecosistema rico en herramientas, bibliotecas y soporte comunitario, .NET se ha posicionado como una opción preferida por desarrolladores y organizaciones a nivel global.

Por otro lado, los servicios REST se han convertido en el estándar de facto para la creación de APIs (Application Programming Interfaces) debido a su simplicidad, escalabilidad y compatibilidad con los principios del diseño web. REST, basado en el protocolo HTTP, permite la comunicación entre sistemas mediante operaciones CRUD (Crear, Leer, Actualizar, Eliminar), utilizando una arquitectura que se alinea con los principios de la web moderna. La popularidad de REST radica en su capacidad para facilitar la integración de diferentes sistemas y aplicaciones, permitiendo una comunicación eficiente y estandarizada.

Este informe tiene como objetivo explorar en profundidad las tecnologías .NET y su aplicación en la creación de servicios REST. Se abordarán las características distintivas y los beneficios de .NET, destacando cómo esta plataforma puede optimizar el desarrollo de servicios RESTful. Además, se examinarán las herramientas y frameworks específicos de .NET, como ASP.NET Core y Entity Framework Core, que facilitan la creación y gestión de APIs RESTful, proporcionando ejemplos prácticos y estudios de caso que ilustran su implementación en escenarios reales.

Asimismo, se presentará una guía práctica para la implementación de un servicio REST utilizando .NET, que incluirá una descripción detallada de los pasos necesarios para configurar un entorno de desarrollo, diseñar y construir la API, realizar pruebas y finalmente desplegar el servicio en producción. Esta sección se enriquecerá con ejemplos de código y proyectos de referencia que servirán como guía para desarrolladores que deseen adoptar estas tecnologías en sus propios proyectos.

Finalmente, el informe evaluará las prácticas recomendadas y patrones de diseño en la creación de servicios REST con .NET. Se identificarán y describirán los patrones de diseño comunes, así como las mejores prácticas para asegurar, documentar y versionar APIs RESTful. Además, se analizarán métodos para evaluar el rendimiento y la seguridad de los servicios REST implementados en .NET, utilizando pruebas y métricas relevantes.

A través de este análisis exhaustivo, el informe pretende proporcionar una visión clara y comprensiva de cómo las tecnologías .NET pueden ser aprovechadas para desarrollar servicios REST que respondan a las necesidades actuales del mercado. Este informe está dirigido a desarrolladores, arquitectos de software y profesionales de TI que buscan entender y aplicar estas tecnologías en sus proyectos, con el objetivo de mejorar la eficiencia, la escalabilidad y la seguridad de sus soluciones.

# MARCO TEÓRICO.

**3.1 Tecnologías .NET**

.NET es un marco de desarrollo de software creado por Microsoft que proporciona un entorno de programación unificado para construir una amplia variedad de aplicaciones, desde aplicaciones web y de escritorio hasta servicios en la nube y aplicaciones móviles. Desde su lanzamiento inicial en 2002, .NET ha evolucionado significativamente, culminando en su versión más reciente, .NET 6, que unifica las diferentes plataformas .NET bajo un solo marco.

**3.2. Componentes de .NET**

.NET Core: Una versión multiplataforma, de código abierto y modular de .NET que permite desarrollar aplicaciones que se ejecutan en Windows, macOS y Linux. Es especialmente adecuado para aplicaciones en la nube y microservicios.

ASP.NET Core: Un marco para construir aplicaciones web y servicios API, conocido por su alto rendimiento y capacidad para manejar aplicaciones de gran escala.

Entity Framework Core: Un ORM (Object-Relational Mapper) que simplifica el acceso a bases de datos mediante la programación orientada a objetos, permitiendo a los desarrolladores trabajar con datos utilizando objetos .NET.

Xamarin: Una plataforma que permite desarrollar aplicaciones móviles para iOS y Android utilizando .NET y C#.

**3.3. Ventajas de .NET**

Multiplataforma: Permite desarrollar aplicaciones que se pueden ejecutar en diferentes sistemas operativos.

Rendimiento: Ofrece un alto rendimiento y una capacidad de respuesta rápida para aplicaciones web y servicios.

Seguridad: Incorpora características avanzadas de seguridad para proteger las aplicaciones contra amenazas y vulnerabilidades.

Productividad: Herramientas integradas como Visual Studio y una amplia gama de bibliotecas y frameworks facilitan el desarrollo rápido y eficiente.

**3.4 Servicios REST**

REST (Representational State Transfer) es un estilo arquitectónico para sistemas distribuidos, introducido por Roy Fielding en su tesis doctoral en el año 2000. REST se basa en un conjunto de principios y restricciones que utilizan el protocolo HTTP para facilitar la comunicación entre clientes y servidores.

**3.5. Principios de REST**

Stateless: Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender y procesar la solicitud. El servidor no almacena el estado del cliente entre solicitudes.

Client-Server: La arquitectura REST separa las preocupaciones del cliente y el servidor, permitiendo que cada uno evolucione de manera independiente.

Cacheable: Las respuestas de las solicitudes deben ser explícitamente marcadas como cacheables o no cacheables, para mejorar la eficiencia y escalabilidad.

Layered System: La arquitectura puede tener capas jerárquicas que limitan el comportamiento de los componentes a ciertos niveles.

Uniform Interface: REST impone una interfaz uniforme que permite que las interacciones entre diferentes sistemas sean predecibles y consistentes.

**3.6. Métodos HTTP en REST**

GET: Recupera información del servidor.

POST: Envía datos al servidor para crear un nuevo recurso.

PUT: Actualiza un recurso existente en el servidor.

DELETE: Elimina un recurso del servidor.

PATCH: Aplica modificaciones parciales a un recurso.

**3.7 ASP.NET Core para Servicios RESTful**

ASP.NET Core es el marco principal dentro del ecosistema .NET para desarrollar servicios RESTful. Proporciona una infraestructura robusta para crear APIs que cumplen con los principios REST.

Configuración Inicial: Configurar un proyecto ASP.NET Core para desarrollar una API RESTful incluye definir rutas, controladores y acciones.

Controladores y Acciones: Los controladores en ASP.NET Core gestionan las solicitudes HTTP y retornan las respuestas adecuadas. Cada acción en el controlador está vinculada a un método HTTP (GET, POST, PUT, DELETE).

Model Binding y Validación: ASP.NET Core facilita el enlace de datos de las solicitudes a objetos de modelo, así como la validación de los mismos antes de procesar las solicitudes.

**3.8. Entity Framework Core en Servicios REST**

Entity Framework Core (EF Core) es una herramienta clave para gestionar datos en servicios RESTful construidos con .NET.

Mapeo de Entidades: EF Core permite mapear clases de .NET a tablas en una base de datos, facilitando las operaciones CRUD.

Consultas LINQ: Utilizando LINQ (Language Integrated Query), los desarrolladores pueden escribir consultas de manera intuitiva y eficiente sobre las entidades mapeadas.

Migraciones: EF Core soporta migraciones que permiten actualizar el esquema de la base de datos de forma incremental y controlada.

**3.9. Seguridad y Autenticación**

La seguridad es un aspecto crítico en el desarrollo de servicios RESTful. ASP.NET Core ofrece varios mecanismos para proteger APIs, como autenticación basada en tokens JWT (JSON Web Tokens) y autorización mediante políticas y roles. Autenticación: Implementación de OAuth 2.0 y OpenID Connect para autenticar usuarios y aplicaciones.

Autorización: Definición de políticas de acceso y roles para controlar quién puede acceder a qué recursos y operaciones.

**3.10. Documentación y Versionado**

La documentación y el versionado son esenciales para la mantenibilidad y la evolución de las APIs RESTful.

Swagger/OpenAPI: ASP.NET Core integra herramientas como Swashbuckle para generar documentación interactiva de las APIs mediante la especificación OpenAPI.

Versionado: Estrategias para versionar las APIs, asegurando compatibilidad y facilitando la introducción de nuevas funcionalidades sin interrumpir los clientes existentes.

# METODOLOGÍA DE DESARROLLO.

El desarrollo del proyecto se basó el patrón de diseño Vista-Controlador (MVC) en la estructuración de la aplicación web de manera modular y separar claramente la lógica de presentación, la lógica de negocio y el manejo de datos; de acuerdo con las guías proporcionadas por el docente.

En la fase inicial se hizo una revisión exhaustiva de las guías para entender el propósito del proyecto y el diseño del mismo tanto del lado del servidor como del cliente.

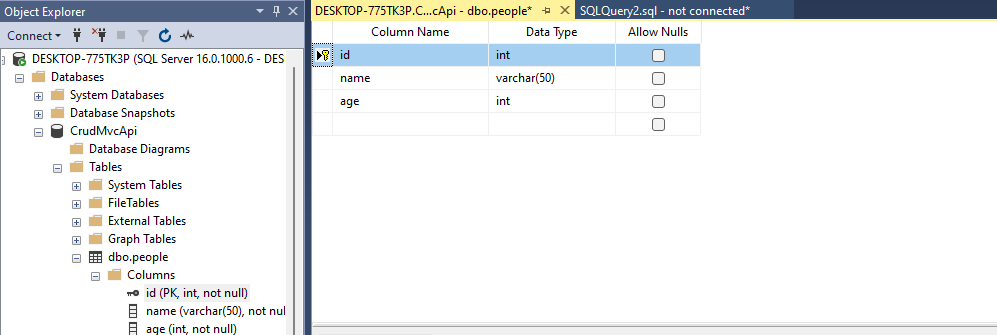
En la fase de desarrollo, se implementaron cada uno de los componentes del lado del cliente a la aplicación web, de modo que sea posible el consumo de la REST.

En la fase de pruebas, se realizaron pruebas unitarias de cada una de las aplicaciones con la base de datos correspondiente a la conexión de la base de datos usando .NET y REST.

# DESARROLLO DE LA PRÁCTICA.

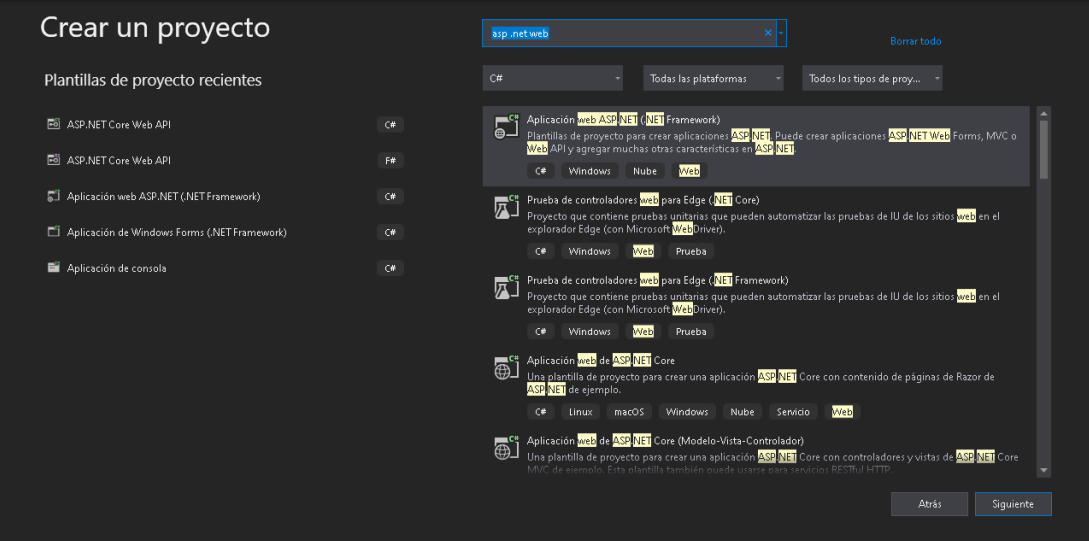
**5.1 SQL server**

En primer instancias se va a utilizar el servidor de base de datos de SQL server en el cual crearemos una base de datos llamada CrudMvcApi en el cual tenemos los atributos de id, nombre y edad.

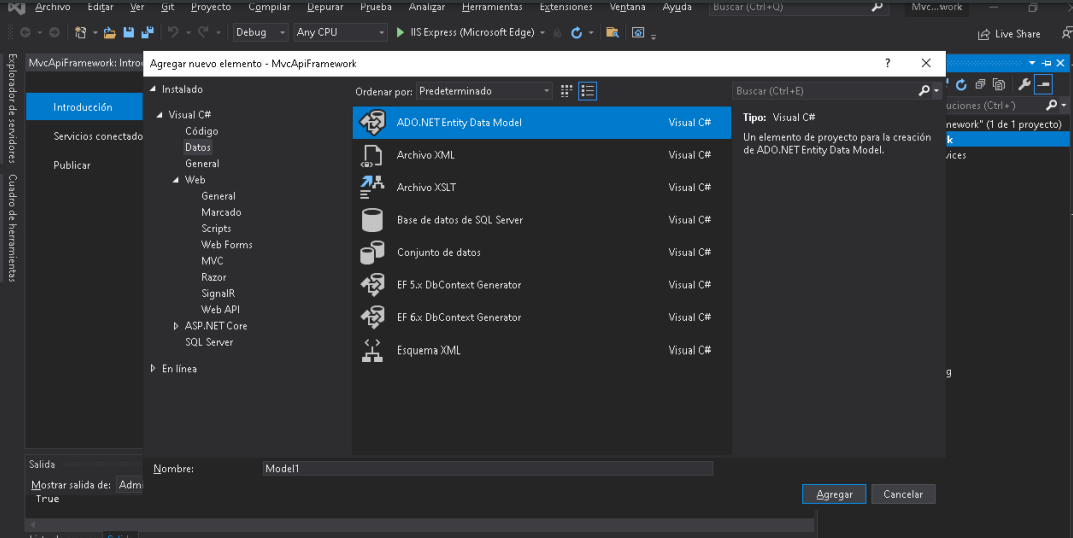


**5.2 Visual Studio**

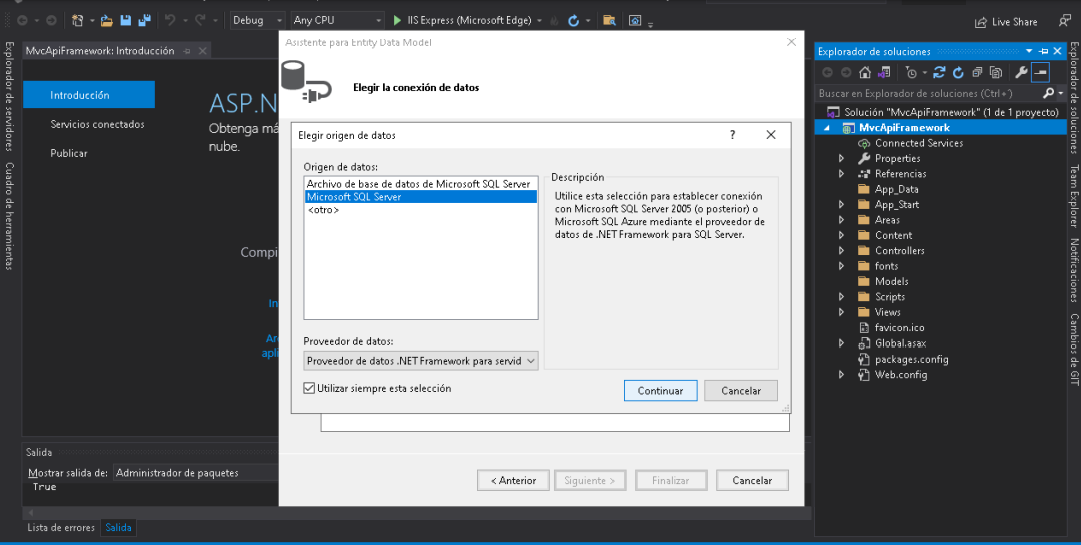
Ahora en visual studio creamos un nuevo proyecto en el cual crearemos nuestra aplicación web en cual se consumira nuestro .NET



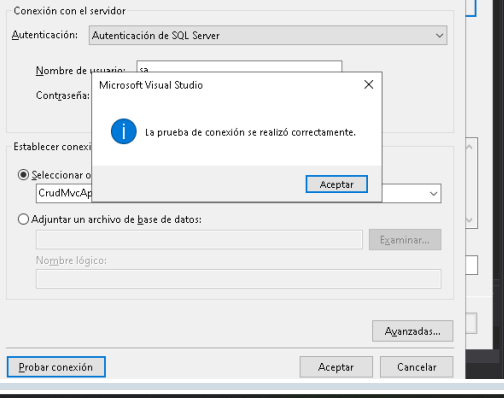
Una vez creado el proyecto hacemos la conexión de la base de datos creada anteriormente en SQL server



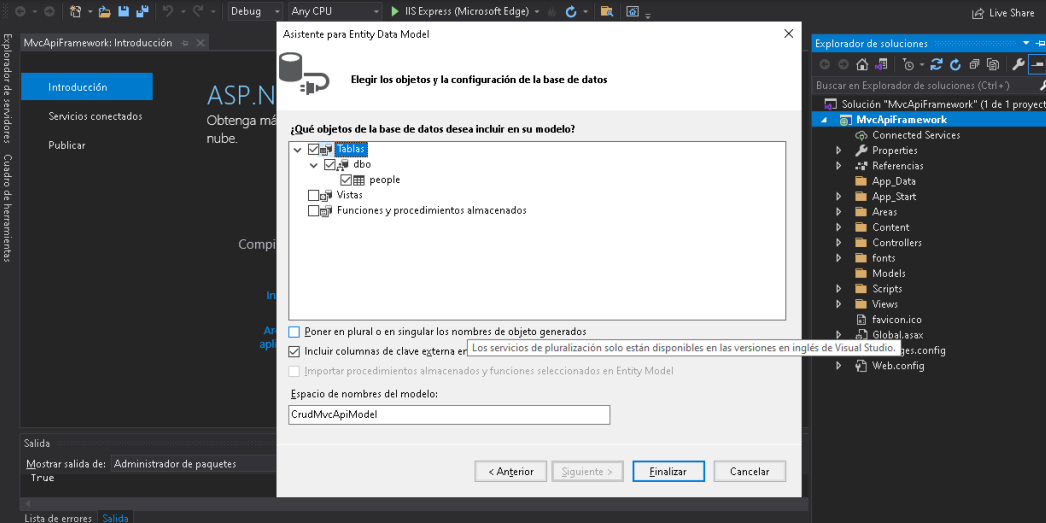
Aquí escogemos el servidor de SQL server



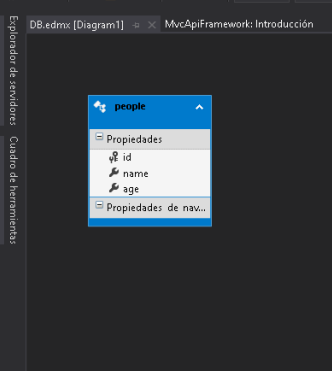
En este paso revisamos la conexión a la base de datos se realizo correctamente



Una vez hecha la conexión importamos la tabla de datos people para que se nos guarde los datos en esa base de datos.



Una vez hecho todo esto y damos click en finalizar, simultáneamente observamos que se importó correctamente la tabla people.



Una vez importada correctamente la base de datos creamos un archivo scaffolding en la parte de Controller llamado peopleController en el cual ya nos da el código predeterminado

Captura de pantalla de un celular

Descripción generada automáticamente

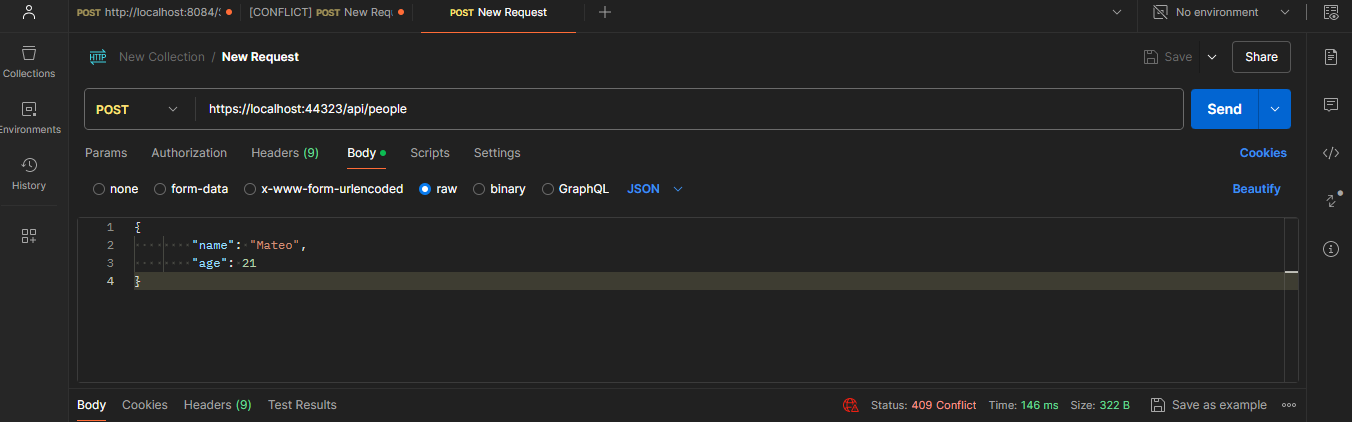
Posteriormente compilamos y observamos que se compile correctamente en nuestro navegador predeterminado.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**5.3 Postman**

Una vez configurado todo en visual studio nos dirigimos a Postman a consumir el servicio y además comprobar si nuestra aplicación está funcionando correctamente.



Agregamos un registro y observamos que si se guardo correctamente en la base de datos de SQL server.

Captura de pantalla de un celular

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

1. **CONCLUSIONES.**

* La plataforma .NET, especialmente con el uso de ASP.NET Core y Entity Framework Core, ofrece una infraestructura robusta y eficiente para la creación de servicios RESTful. Su capacidad para manejar aplicaciones de gran escala, junto con un ecosistema rico en herramientas y bibliotecas, permite a los desarrolladores construir APIs de alto rendimiento y altamente escalables. La flexibilidad de .NET para operar en diferentes entornos y sistemas operativos refuerza su utilidad en diversas situaciones de desarrollo.
* La implementación de servicios RESTful en .NET beneficia significativamente de la adopción de buenas prácticas y patrones de diseño establecidos. El uso de arquitecturas bien definidas, la separación de preocupaciones y la implementación de principios REST como la statelessness y la uniformidad de interfaces, aseguran que las APIs sean consistentes, mantenibles y fáciles de escalar. Además, la utilización de Entity Framework Core para la gestión de datos simplifica las operaciones CRUD y mejora la eficiencia del desarrollo.
* La seguridad en los servicios RESTful es fundamental y .NET proporciona diversas herramientas y mecanismos para asegurar APIs. La integración de autenticación mediante tokens JWT y la implementación de políticas de autorización robustas garantizan que solo usuarios y aplicaciones autorizadas puedan acceder a los recursos, protegiendo los datos y las operaciones sensibles. La capacidad de .NET para incorporar estándares como OAuth 2.0 y OpenID Connect facilita la implementación de soluciones seguras y confiables.
* La documentación clara y el versionado adecuado de las APIs son esenciales para la mantenibilidad y la evolución continua de los servicios RESTful. Herramientas como Swashbuckle y la especificación OpenAPI permiten generar documentación interactiva, lo que facilita la comprensión y el uso de las APIs por parte de los desarrolladores. Estrategias de versionado bien definidas aseguran que las APIs puedan evolucionar sin romper la compatibilidad con los clientes existentes, permitiendo una introducción gradual de nuevas funcionalidades.
* El uso de .NET para el desarrollo de servicios RESTful impacta positivamente en la eficiencia y productividad del proceso de desarrollo. La integración de herramientas de desarrollo como Visual Studio, junto con una comunidad activa y recursos de soporte abundantes, reduce significativamente el tiempo y el esfuerzo necesario para construir, probar y desplegar aplicaciones. Esto no solo acelera el ciclo de desarrollo, sino que también permite a las organizaciones responder rápidamente a las cambiantes necesidades del mercado y a las demandas de los usuarios.

# BIBLIOGRAFÍA

|  |  |
| --- | --- |
| [1] | M. Rosa, “Qué es REST: Conoce su potencia,” *OpenWebinars.net*, May 17, 2018. https://openwebinars.net/blog/que-es-rest-conoce-su-potencia/ (accessed Jun. 12, 2024). |
| [2] | “ASP.NET web de APIs | Rest APIs con .NET y C#,” *Microsoft*, 2024. https://dotnet.microsoft.com/es-es/apps/aspnet/apis (accessed Jun. 12, 2024). |
| [3] | Chakray, “¿Qué diferencias hay entre REST y SOAP y cuál es mejor?,” *Chakray*, Dec. 21, 2016. https://www.chakray.com/es/que-diferencias-hay-entre-rest-y-soap/ (accessed Jun. 12, 2024). |
| [4] | XperTek, “APRENDE a consumir un SERVICIO WEB REST con C# .NET,” *YouTube*. Sep. 10, 2022. Accessed: Jun. 12, 2024. [YouTube Video]. Available: https://www.youtube.com/watch?v=ZEbaMK95G4c  ‌ |

1. **ANEXOS**

# Anexo I: Código peopleController

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Data.Entity.Infrastructure;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

using System.Web.Http.Description;

using MvcApiFramework;

namespace MvcApiFramework.Controllers

{

public class peopleController : ApiController

{

private CrudMvcApiEntities db = new CrudMvcApiEntities();

// GET: api/people

public IQueryable<people> Getpeople()

{

return db.people;

}

// GET: api/people/5

[ResponseType(typeof(people))]

public IHttpActionResult Getpeople(int id)

{

people people = db.people.Find(id);

if (people == null)

{

return NotFound();

}

return Ok(people);

}

// PUT: api/people/5

[ResponseType(typeof(void))]

public IHttpActionResult Putpeople(int id, people people)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

if (id != people.id)

{

return BadRequest();

}

db.Entry(people).State = EntityState.Modified;

try

{

db.SaveChanges();

}

catch (DbUpdateConcurrencyException)

{

if (!peopleExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return StatusCode(HttpStatusCode.NoContent);

}

// POST: api/people

[ResponseType(typeof(people))]

public IHttpActionResult Postpeople(people people)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

db.people.Add(people);

try

{

db.SaveChanges();

}

catch (DbUpdateException)

{

if (peopleExists(people.id))

{

return Conflict();

}

else

{

throw;

}

}

return CreatedAtRoute("DefaultApi", new { id = people.id }, people);

}

// DELETE: api/people/5

[ResponseType(typeof(people))]

public IHttpActionResult Deletepeople(int id)

{

people people = db.people.Find(id);

if (people == null)

{

return NotFound();

}

db.people.Remove(people);

db.SaveChanges();

return Ok(people);

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

private bool peopleExists(int id)

{

return db.people.Count(e => e.id == id) > 0;

}

}

# }