Enfoque hacia una **API REST** utilizando **ASP.NET Core Web API** con C#. Este tipo de arquitectura es ideal para separar el backend del frontend (como aplicaciones móviles o frontends modernos en React, Angular o Vue) y trabaja directamente devolviendo datos en formato JSON.

Paso 1: Crear el proyecto Web API

Si estás usando la terminal (CLI de .NET), ejecuta:

```bash
dotnet new webapi -n MiApiRest
cd MiApiRest
```

Si prefieres **Visual Studio**:

1. Selecciona **Crear un nuevo proyecto**.
2. Busca y selecciona **API de ASP.NET Core** (o *ASP.NET Core Web API*).
3. Configura el nombre, selecciona .NET (versión 8.0 o superior) y haz clic en **Crear**.

(Nota: Las versiones recientes de ASP.NET Core ya vienen configuradas con Swagger/OpenAPI por defecto, lo que te permitirá probar la API visualmente de inmediato).

Paso 2: Crear el Modelo de datos

Crea una carpeta llamada Models (si no existe) y dentro añade un archivo llamado Producto.cs:

```csharp
namespace MiApiRest.Models
{
    public class Producto
    {
        public int Id { get; set; }
        public string Nombre { get; set; }
        public decimal Precio { get; set; }
        public int Stock { get; set; }
    }
}
```

Paso 3: Crear el Controlador de la API

El controlador manejará las peticiones HTTP (GET, POST, PUT, DELETE).

1. En la carpeta Controllers, crea un archivo llamado ProductosController.cs.

2. Añade el siguiente código que implementa las operaciones CRUD completas utilizando una lista estática en memoria como repositorio simulado:

```
C#
using Microsoft.AspNetCore.Mvc;
using MiApiRest.Models;

namespace MiApiRest.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ProductosController : ControllerBase
    {
        // Simulación de base de datos en memoria
        private static List<Producto> listaProductos = new List<Producto>
        {
            new Producto { Id = 1, Nombre = "Laptop", Precio = 850.00m, Stock = 10 },
            new Producto { Id = 2, Nombre = "Mouse", Precio = 25.50m, Stock = 50 }
        };

        // GET: api/productos
        [HttpGet]
        public ActionResult<IEnumerable<Producto>> Get()
        {
            return Ok(listaProductos);
        }

        // GET: api/productos/5
        [HttpGet("{id}")]
        public ActionResult<Producto> GetById(int id)
        {
            var producto = listaProductos.FirstOrDefault(p => p.Id == id);
            if (producto == null)
            {
                return NotFound(new { mensaje = "Producto no encontrado" });
            }
            return Ok(producto);
```

```
    }

    // POST: api/productos
    [HttpPost]
    public ActionResult<Producto> Post([FromBody] Producto nuevoProducto)
    {
        nuevoProducto.Id = listaProductos.Count > 0 ? listaProductos.Max(p => p.Id) + 1 : 1;
        listaProductos.Add(nuevoProducto);

        return CreatedAtAction(nameof(GetById), new { id = nuevoProducto.Id },
nuevoProducto);
    }

    // PUT: api/productos/5
    [HttpPut("{id}")]
    public IActionResult Put(int id, [FromBody] Producto productoActualizado)
    {
        var producto = listaProductos.FirstOrDefault(p => p.Id == id);
        if (producto == null)
        {
            return NotFound(new { mensaje = "Producto no encontrado" });
        }

        producto.Nombre = productoActualizado.Nombre;
        producto.Precio = productoActualizado.Precio;
        producto.Stock = productoActualizado.Stock;

        return NoContent(); // Código 204: Actualización exitosa sin contenido de vuelta
    }

    // DELETE: api/productos/5
    [HttpDelete("{id}")]
    public IActionResult Delete(int id)
    {
        var producto = listaProductos.FirstOrDefault(p => p.Id == id);
        if (producto == null)
        {
            return NotFound(new { mensaje = "Producto no encontrado" });
        }
```

```
            listaProductos.Remove(producto);
            return NoContent();
        }
    }
}
```

Paso 4: Ejecutar y Probar la API

1. Ejecuta la aplicación desde la terminal:
   ```
   Bash
   dotnet run
   ```

2. Abre tu navegador web y dirígete a la URL de desarrollo (por ejemplo, https://localhost:5001/swagger o http://localhost:5000/swagger).
3. Se abrirá la interfaz interactiva de **Swagger UI**, donde podrás probar directamente cada endpoint (GET, POST, PUT, DELETE) enviando y recibiendo datos en formato JSON de manera visual.