

UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA
Facultad de Ingeniería en Sistemas.
Docente: ing.



Sprint 1 Análisis de Sistemas.

Estudiante: Derek Bladimir Castañeda Siliezar

Carné: 9989-22-16035

Estudiante: Danny Josue Archilla Alvarez

Carné: 9989-22-19101

Estudiante: William Steven Molina Molina

Carné: 9989-22-16660

Estudiante: Jeffrey Alexander Navarro Muralles

Carné: 9989-22-5310

Historia de Usuario 1: Creación y Gestión de Tickets

Descripción: Como cliente del CRM, quiero crear un ticket de soporte para reportar incidencias o solicitudes, de forma que pueda dar seguimiento a la solución de mi problema.

Criterios de aceptación:

- El cliente puede acceder a un formulario en el frontend para registrar un nuevo ticket con título, descripción, categoría y prioridad.
- El sistema debe validar que los campos obligatorios estén completos antes de guardar.
- Una vez creado, el ticket se almacena en la base de datos y se asigna automáticamente o manualmente a un agente de soporte.
- El cliente recibe una notificación (correo o interna) confirmando la creación del ticket.
- El ticket debe tener un identificador único y un estado inicial de "Abierto".

Historia de Usuario 2: Seguimiento y Respuesta a Tickets

Descripción: Como agente de soporte, quiero ver, responder y cerrar tickets asignados a mí, para dar una atención oportuna al cliente.

Criterios de aceptación:

- El agente puede acceder a un dashboard que muestre todos los tickets asignados, con filtros por estado, categoría y prioridad.
- El agente puede abrir un ticket para ver los detalles y el historial de interacciones.
- El agente puede agregar respuestas o comentarios al ticket.
- El cliente recibe una notificación cuando su ticket es actualizado.

El agente puede cambiar el estado del ticket a "En Proceso" o "Cerrado", registrándose la fecha y hora del cambio.

Código fuente del web service

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ProductosCRUD.Server.Models;

namespace ProductosCRUD.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ProductosController : ControllerBase
    {
        private readonly ProductosCRUDContext _context;

        public ProductosController(ProductosCRUDContext context)
        {
            _context = context;
        }

        [HttpPost]
        [Route("crearProducto")]
        public async Task<IActionResult> CreateProduct(Producto producto)
        {
            //guardar el producto en la base de datos
            await _context.Productos.AddAsync(producto);
            await _context.SaveChangesAsync();

            //devolver un mensaje de exito
        }
    }
}
```

```
        return Ok();

    }

[HttpGet]
[Route("listaProducto")]
public async Task<ActionResult<IEnumerable<Producto>>> GetProducts()
{
    //Obten la lista de productos de la base de datos
    var productos = await _context.Productos.ToListAsync();

    //devuelve una lista de productos
    return Ok(productos);
}

[HttpGet]
[Route("verProducto")]
public async Task<IActionResult> GetProduct(int id)
{
    //obtener el producto de la base de datos
    Producto producto = await _context.Productos.FindAsync(id);

    //devolver el producto
    if (producto == null)
    {
        return NotFound();
    }

    return Ok(producto);
}
```

```

}

[HttpPost]
[Route("editarProducto")]
public async Task<IActionResult> UpdateProduct(int id, Producto producto)
{
    //Actualizar el producto en la base de datos
    var productoExistente = await _context.Productos.FindAsync(id);
    productoExistente.Nombre = producto.Nombre;
    productoExistente.Descripcion = producto.Descripcion;
    productoExistente.Precio = producto.Precio;

    await _context.SaveChangesAsync();

    //devolver un mensaje de exito
    return Ok();
}

[HttpDelete]
[Route("eliminarProducto")]
public async Task<IActionResult> DeleteProduct(int id)
{
    //Eliminar producto de la base de datos
    var productoBorrado = await _context.Productos.FindAsync(id);
    _context.Productos.Remove(productoBorrado!);

    await _context.SaveChangesAsync();

    //Devolver un mensaje de exito
    return Ok();
}

```

```
    }  
}  
  
}
```

Integración con la base de datos

```
using Microsoft.EntityFrameworkCore;  
using ProductosCRUD.Server.Models;  
  
namespace ProductosCRUD.Server  
{  
    public class Program  
    {  
        public static void Main(string[] args)  
        {  
            var builder = WebApplication.CreateBuilder(args);  
  
            // Add services to the container.  
  
            builder.Services.AddControllers();  
            builder.Services.AddDbContext<ProductosCRUDContext>(o =>  
            {  
                o.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));  
            });  
            // Learn more about configuring Swagger/OpenAPI at  
            https://aka.ms/aspnetcore/swashbuckle  
            builder.Services.AddEndpointsApiExplorer();  
            builder.Services.AddSwaggerGen();  
        }  
    }  
}
```

```
var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
}
```

Documentación del web service

Este Web Service está diseñado para gestionar productos de manera sencilla y eficiente a través de una API RESTful. Permite que diferentes aplicaciones o sistemas puedan interactuar con los productos, realizando acciones básicas como listar todos los productos, obtener la información de uno específico, crear nuevos productos, actualizar los existentes o eliminarlos.

La comunicación con el servicio se realiza usando el protocolo HTTP y los datos se envían y reciben en formato JSON, un estándar ligero y fácil de usar. Todas las rutas del servicio comienzan con el prefijo /api/products, lo que facilita la organización y el acceso a los recursos relacionados con productos.

Los principales puntos de acceso (endpoints) son cinco:

1. Obtener todos los productos: usando GET /products se recupera una lista con todos los productos disponibles. Esto es útil para mostrar catálogos o listas completas en una aplicación.
2. Obtener un producto específico: con GET /products/{id}, donde {id} es el identificador único del producto, se accede a la información detallada de ese producto en particular.
3. Crear un nuevo producto: a través de POST /products, enviando en el cuerpo de la petición los datos del producto en formato JSON (nombre, descripción y precio), se añade un nuevo producto a la base de datos o sistema.
4. Actualizar un producto existente: usando PUT /products/{id}, es posible modificar los detalles de un producto ya registrado. Esto es útil cuando se necesitan corregir datos o actualizar precios y descripciones.
5. Eliminar un producto: con DELETE /products/{id}, se elimina el producto correspondiente al identificador enviado.

Cada producto está representado por un conjunto de atributos claros y sencillos: un id que es un número único que identifica el producto, un nombre que describe qué es el producto, una descripción que ofrece más detalles sobre él, y un precio que indica su costo.

Para crear o actualizar un producto, es necesario enviar un objeto JSON que incluya al menos el nombre, la descripción y el precio, todos con valores adecuados y en el formato esperado para evitar errores.

También esta API está documentada utilizando la especificación OpenAPI versión 3.0, que es un estándar ampliamente usado para describir APIs REST. Esto facilita a cualquier desarrollador entender cómo usar el servicio, probar los distintos endpoints y hasta generar código cliente automáticamente. Herramientas como Swagger Editor o Postman pueden cargar esta documentación para una experiencia interactiva.

En caso de que el sistema lo requiera, se puede implementar seguridad para proteger el acceso a la API, generalmente usando tokens JWT en el encabezado de autorización, lo que garantiza que solo usuarios o sistemas autorizados puedan manipular los productos.

Instrucciones de instalación y despliegue del web service

Creación de Base de Datos en SQL server.

Nombre de BD: ProductosDb.

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left, the 'Explorador de objetos' (Object Explorer) tree view shows the database structure under 'DESKTOP-8U33HSC (16.0.1000.6 de S)'. A node for 'ProductosDb' is expanded, showing 'Bases de datos', 'Diagramas de base de datos', and 'Tablas'. The 'Tablas' node is selected. On the right, the 'SQLQuery1.sql - DE...ductosDb (sa (66))' tab contains the following SQL code:

```
use ProductosDb
select * from Productos
```

The results pane shows a single row of data from the 'Productos' table:

	Id	Nombre	Descripcion	Precio
1	1	danny	string	0.00

CRUD.

The screenshot shows the Visual Studio IDE with the 'ProductosController.cs' file open in the code editor. The code implements a CRUD controller for the 'Productos' database:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ProductosCRUD.Server.Models;

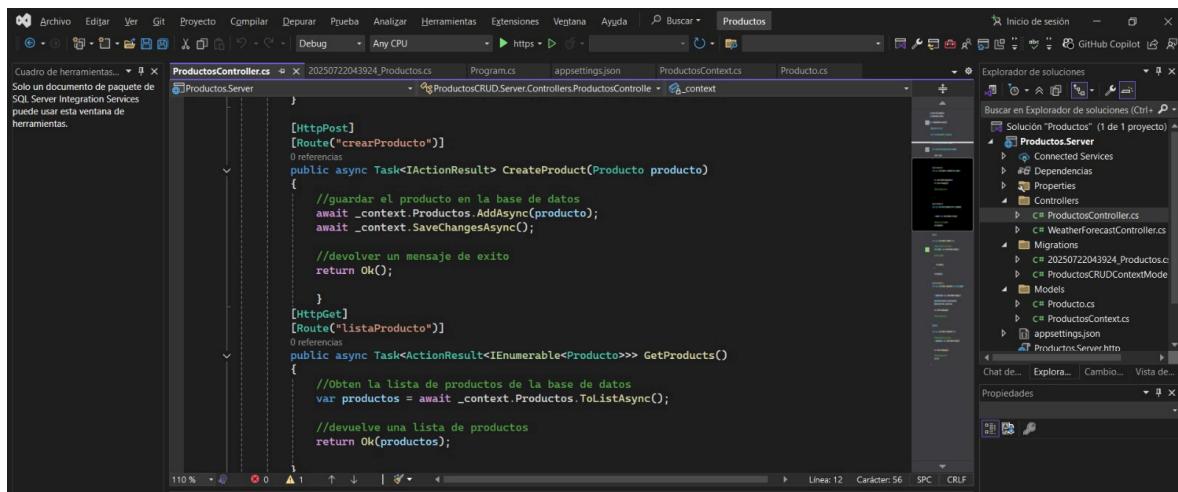
namespace ProductosCRUD.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ProductosController : ControllerBase
    {
        private readonly ProductosCRUDContext _context;

        public ProductosController(ProductosCRUDContext context)
        {
            _context = context;
        }

        [HttpPost]
        [Route("crearProducto")]
        public async Task<IActionResult> CreateProduct(Producto producto)
        {
            //guardar el producto en la base de datos
            await _context.Productos.AddAsync(producto);
            await _context.SaveChangesAsync();
        }
    }
}
```

The solution explorer on the right shows the project structure, including 'Controllers', 'Models', and 'Views' folders. The 'Controllers' folder contains 'ProductosController.cs' and 'WeatherForecastController.cs'. The 'Models' folder contains 'Producto.cs' and 'ProductosContext.cs'. The 'Views' folder contains 'Index.cshtml' and 'Create.cshtml'.

Creación y Listado



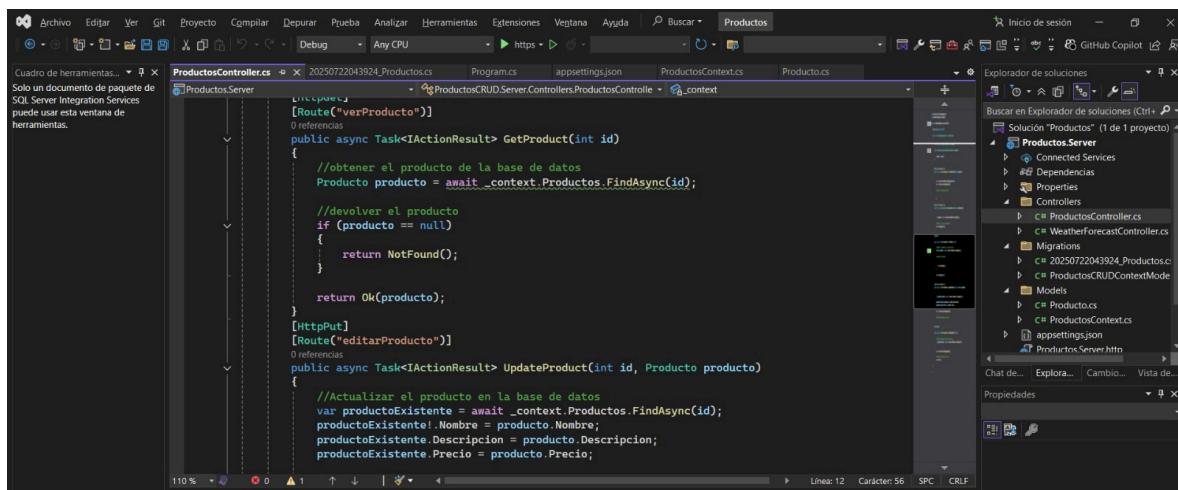
```
[HttpPost]
[Route("crearProducto")]
public async Task<IActionResult> CreateProduct(Producto producto)
{
    //guardar el producto en la base de datos
    await _context.Productos.AddAsync(producto);
    await _context.SaveChangesAsync();

    //devolver un mensaje de exito
    return Ok();
}

[HttpGet]
[Route("listaProducto")]
public async Task<ActionResult<IEnumerable<Producto>>> GetProducts()
{
    //obten la lista de productos de la base de datos
    var productos = await _context.Productos.ToListAsync();

    //devuelve una lista de productos
    return Ok(productos);
}
```

Ver y Editar



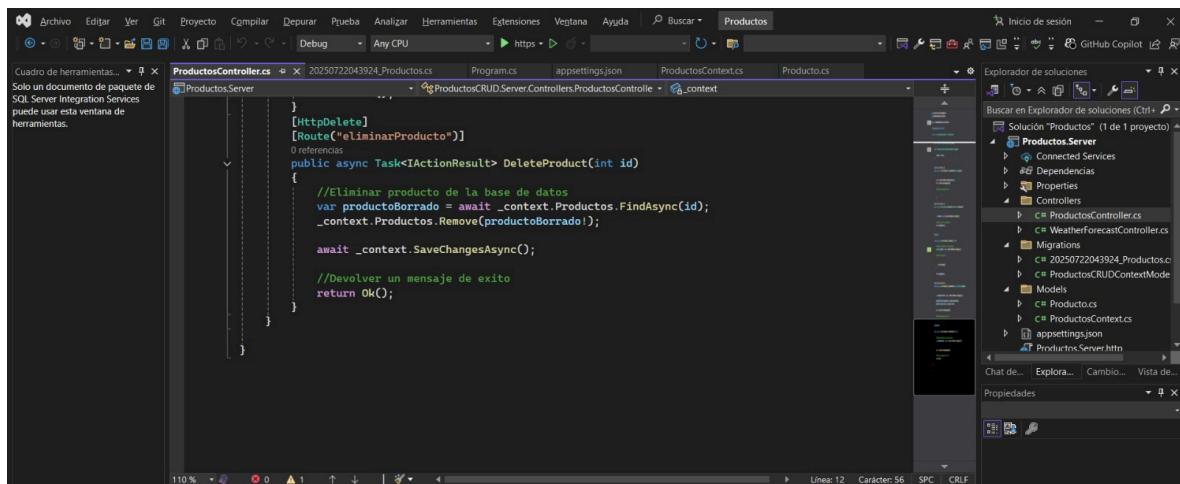
```
[HttpGet]
[Route("verProducto")]
public async Task<IActionResult> GetProduct(int id)
{
    //obtener el producto de la base de datos
    Producto producto = await _context.Productos.FindAsync(id);

    //devolver el producto
    if (producto == null)
    {
        return NotFound();
    }

    return Ok(producto);
}

[HttpPut]
[Route("editarProducto")]
public async Task<IActionResult> UpdateProduct(int id, Producto producto)
{
    //Actualizar el producto en la base de datos
    var productoExistente = await _context.Productos.FindAsync(id);
    productoExistente.Nombre = producto.Nombre;
    productoExistente.Descripcion = producto.Descripcion;
    productoExistente.Precio = producto.Precio;
}
```

eliminación.



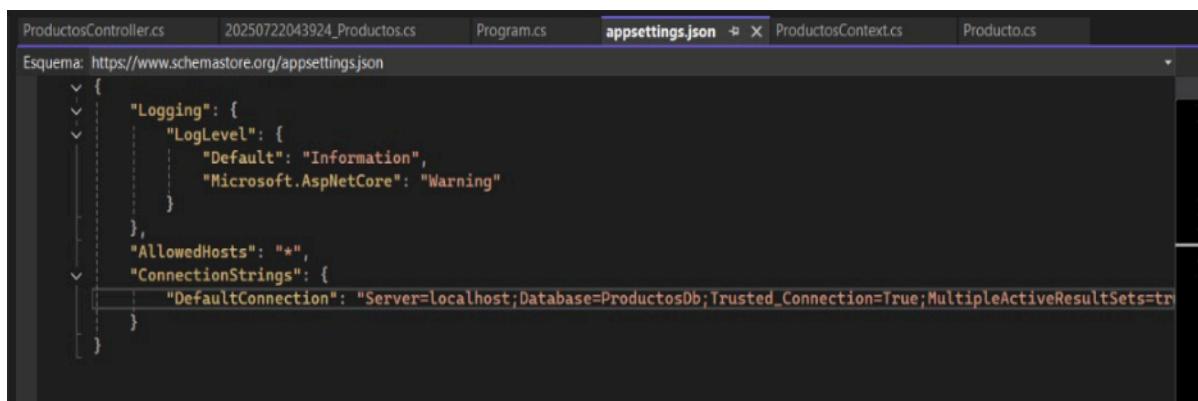
The screenshot shows the Visual Studio IDE with the code editor open to the `ProductsController.cs` file. The code defines a `DeleteProduct` method that performs a database delete operation and returns a success message. The code editor has syntax highlighting and a status bar at the bottom indicating line 12, character 56.

```
        }
        [HttpDelete]
        [Route("eliminarProducto")]
        public async Task<IActionResult> DeleteProduct(int id)
        {
            //Eliminar producto de la base de datos
            var productoBorrado = await _context.Productos.FindAsync(id);
            _context.Productos.Remove(productoBorrado);

            await _context.SaveChangesAsync();

            //Devolver un mensaje de éxito
            return Ok();
        }
    }
```

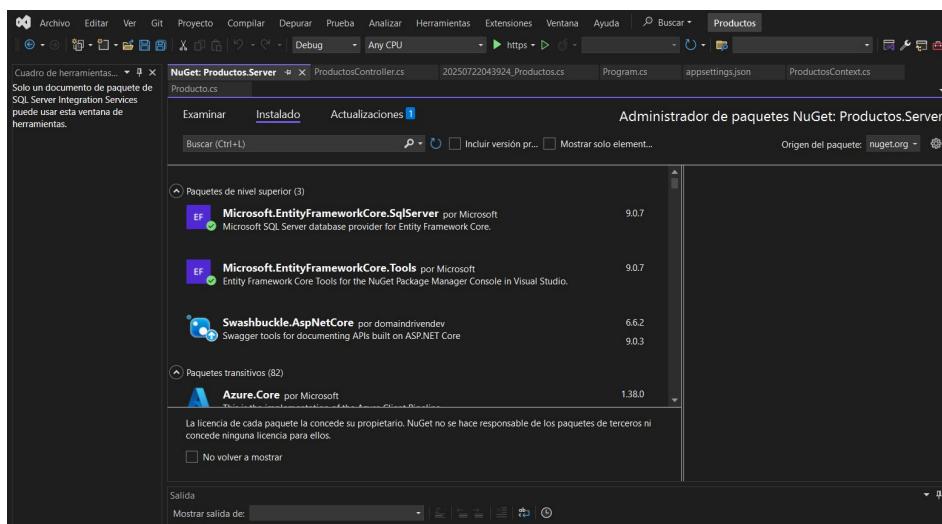
Esta es la conexión entre la Base de datos y el WebServices.



The screenshot shows the Visual Studio IDE with the `appsettings.json` file open. It contains configuration settings for logging, allowed hosts, and connection strings, including the default connection to the local database.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost;Database=ProductosDb;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

paquetes de instalación.



Pruebas para Validar el Funcionamiento.

Esta interfaz es parte de Swagger UI, una herramienta que sirve para probar y documentar APIs de manera visual. En este caso, se trata del proyecto llamado Productos.Server, en su versión 1.0, que está corriendo localmente. Se está usando la especificación OpenAPI 3.0, lo que permite describir claramente los servicios que ofrece la API.

El proyecto cuenta con varias rutas bajo la categoría Productos. Estas permiten crear, listar, ver, editar y eliminar productos. Cada operación está claramente marcada con colores según el tipo de acción: verde para crear, azul para ver o listar, naranja para editar y rojo para eliminar. También hay una sección llamada WeatherForecast, que parece ser parte de una plantilla inicial del proyecto.

La interfaz es clara y fácil de usar. Permite probar directamente cada función con el botón “Try it out” y ver ejemplos de cómo usar la API. En general, es una herramienta muy útil tanto para desarrolladores como para quienes necesiten entender o revisar cómo funciona el sistema.

Prueba para creación:

A screenshot of the Swagger UI interface. At the top, there's a green header bar with the title 'Productos'. Below it, a 'POST /api/Productos/crearProducto' button is highlighted. To the right of the button are two buttons: 'Cancel' and 'Reset'. Underneath the button, there's a section titled 'Parameters' which says 'No parameters'. Below that is a 'Request body' section with a dropdown menu set to 'application/json'. Inside this section, there's a code block containing a JSON object:

```
{
  "id": 0,
  "nombre": "pera",
  "descripcion": "fruta",
  "precio": 4
}
```

Prueba de listado.

The screenshot shows a REST API testing interface. At the top, it says "GET /api/Productos/listaProducto". Below that, under "Parameters", it says "No parameters". There are "Execute" and "Clear" buttons. Under "Responses", there's a "Curl" section with the command:

```
curl -X 'GET' '\n"https://localhost:7270/api/Productos/listaProducto"\n-H accept: text/plain'
```

Below that is a "Request URL" field containing "https://localhost:7270/api/Productos/listaProducto". Under "Server response", it shows a "Code" of 200 and a "Details" section. The "Response body" contains the following JSON array:

```
[ { "id": 6, "nombre": "manzana", "descripcion": "fruta", "precio": 10 }, { "id": 7, "nombre": "pera", "descripcion": "fruta", "precio": 4 } ]
```

There are "Download" and "Copy" buttons next to the JSON.

Base de datos.

The screenshot shows Microsoft SQL Server Management Studio. The title bar says "SQLQuery1.sql - DESKTOP-8U33HSC.ProductosDb (sa (66)) - Microsoft SQL Server Management Studio". The menu bar includes Archivo, Editar, Ver, Consulta, Proyecto, Herramientas, Ventana, Ayuda. The toolbar has various icons for file operations. The left pane is the "Explorador de objetos" (Object Explorer) showing the database structure:

- DESKTOP-8U33HSC (16.0.1000.6 de S)
 - Bases de datos
 - Bases de datos del sistema
 - Instantáneas de bases de datos
 - DIMENSIONAL-NOMINAS
 - DW_Planta
 - DWH_Nominas
 - Empresa
 - ER-NOMINAS
 - PlantAndHealth
 - ProductosDb
 - Diagramas de base de datos
 - Tablas
 - Tablas del sistema
 - Tablas de archivos
 - Tablas externas
 - dbo._EFMigrationsHistor
 - dbo.Productos
 - Tablas de libro de contab
 - Vistas
 - Recursos externos

Prueba de Edición.

The screenshot shows a REST API testing interface. The URL is `PUT /api/Productos/editarProducto`. In the 'Parameters' section, there is a parameter named `id` with the value `6`. The 'Request body' section contains the following JSON payload:

```
{ "id": 0, "nombre": "zanahoria", "descripcion": "vegetal", "precio": 9 }
```

The 'Responses' section shows a curl command to execute the request:

```
curl -X 'PUT' \ 'http://localhost:7270/api/Productos/editarProducto?id=6' \ -H 'accept: */*' \ -H 'Content-type: application/json' \ -d { "id": 0, "nombre": "zanahoria", "descripcion": "vegetal", }
```

Base de datos.

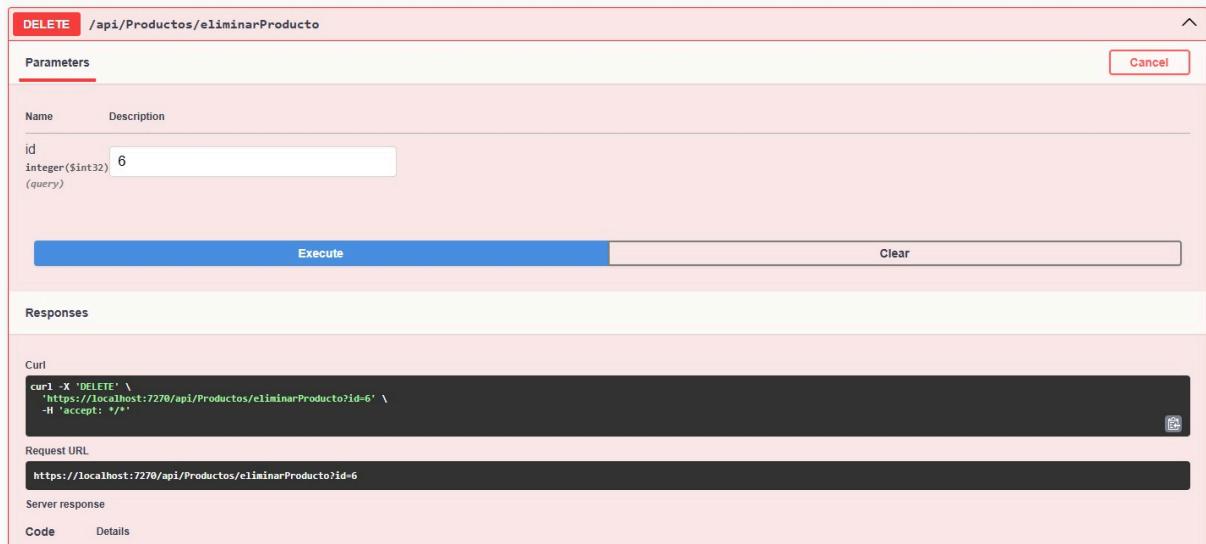
The screenshot shows Microsoft SQL Server Management Studio (SSMS) interface. The left pane shows the 'Explorador de objetos' (Object Explorer) with the 'ProductosDb' database selected. The right pane shows a query window titled 'SQLQuery1.sql - DE...uctosDb (sa (66))'. The query is:

```
use ProductosDb  
select * from Productos
```

The results pane shows the following data:

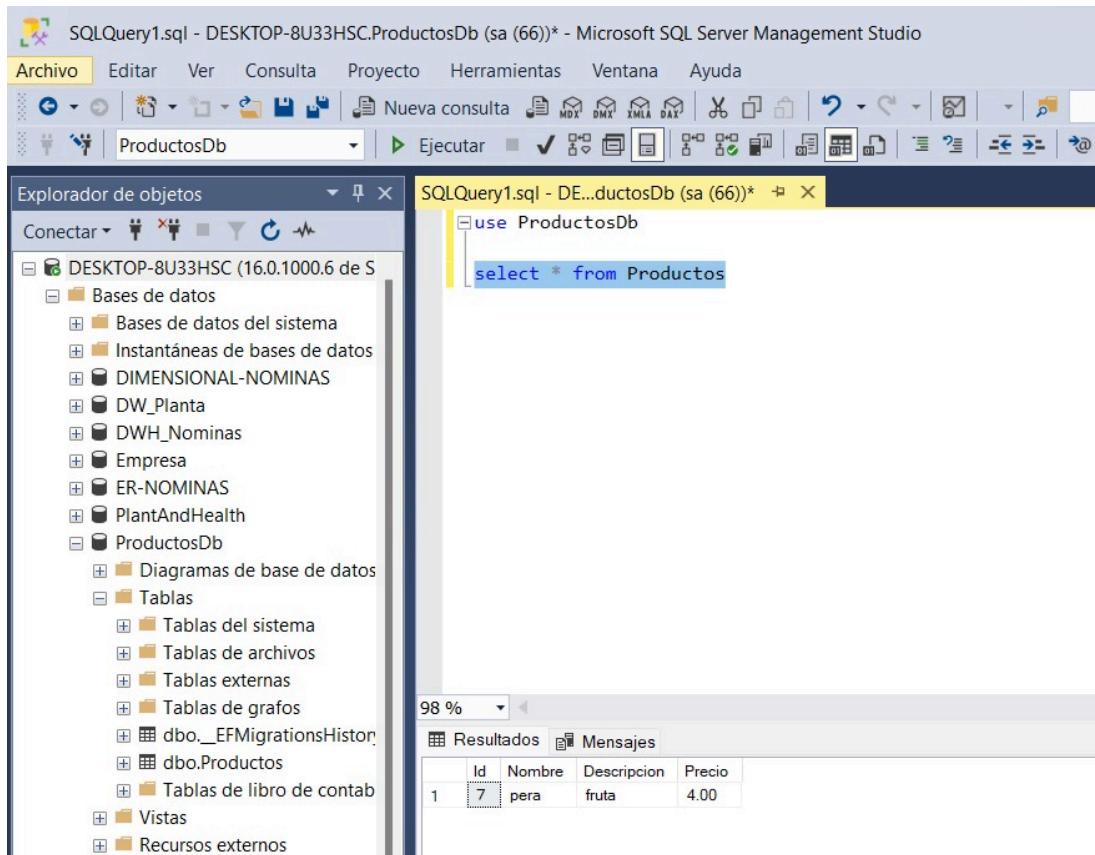
	Id	Nombre	Descripcion	Precio
1	6	zanahoria	vegetal	9.00
2	7	pera	fruta	4.00

Prueba de Eliminación.



The screenshot shows a REST API testing interface. At the top, a red bar indicates a **DELETE** operation to the endpoint `/api/Productos/eliminarProducto`. Below this, the **Parameters** section shows a single parameter `id` with the value `6`. There are buttons for **Execute** and **Clear**. Under the **Responses** section, there are sections for **Curl** (containing the command `curl -X 'DELETE' \https://localhost:7270/api/Productos/eliminarProducto?id=6 \ -H 'accept: */*`), **Request URL** (`https://localhost:7270/api/Productos/eliminarProducto?id=6`), and **Server response**. The **Code** tab is selected.

Base de Datos.



The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. The title bar says "SQLQuery1.sql - DESKTOP-8U33HSC.ProductosDb (sa (66))* - Microsoft SQL Server Management Studio". The toolbar has options like Archivo, Editar, Ver, Consulta, Proyecto, Herramientas, Ventana, Ayuda. The object explorer on the left shows the database structure, including the `ProductosDb` database. The main window contains a query editor with the following SQL code:

```
use ProductosDb
select * from Productos
```

The results pane shows a table with one row of data:

	Id	Nombre	Descripcion	Precio
1	7	pera	fruta	4.00