

Crear una Rest API mediante un proyecto que Exporte Datos de una Base de Datos en MySQL a una Tabla Excel con el uso de Clases y Variables de Entorno

Principios del Diseño del Estilo Arquitectónico REST para una API:

REST (*Representational State Transfer*) es un estilo arquitectónico para diseñar servicios web que permite la comunicación entre sistemas en la web., a continuación, se detallan los principios fundamentales de REST y cómo se aplican al diseño de una API.

1.- Interfaz Uniforme

Esto significa que todas las interacciones con los recursos de la API deben ser consistentes y predecibles.

1.1.- Identificación de recursos: Cada recurso (como un producto, cliente, etc.) se identifica de manera única a través de una URL específica.

1.2.- Representaciones de recursos: Los recursos pueden ser representados en diferentes formatos como JSON, XML, etc., pero deben ser consistentes en toda la API.

1.3.- Manipulación de recursos a través de representaciones: Si envías una representación de un recurso con una solicitud PUT, puedes actualizar ese recurso en el servidor.

2.- Desacoplamiento Cliente-Servidor

El cliente y el servidor deben ser independientes entre sí.

2.1.- Autonomía del cliente: El cliente maneja la interfaz de usuario y la experiencia, mientras que el servidor se encarga de la lógica de negocio y el almacenamiento de datos.

2.2.- Desarrollo independiente: El cliente y el servidor pueden ser desarrollados y escalados de manera independiente.

3.- Sin Estado

Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender y procesar la solicitud.

3.1.- Independencia de las solicitudes: No se debe almacenar el estado del cliente en el servidor entre solicitudes. Cada solicitud debe ser autónoma.

3.2.- Idempotencia: Las operaciones PUT y DELETE deben ser idempotentes, es decir, aplicarlas múltiples veces no debe cambiar el resultado.

4.- Caché

Las respuestas de la API deben ser explícitamente definidas como cachéables o no cachéables para mejorar el rendimiento.

4.1.- Control de caché: Las respuestas pueden incluir información en los encabezados HTTP que indica si y por cuánto tiempo una respuesta puede ser almacenada en caché.

5. Sistema en Capas

La arquitectura REST puede tener múltiples capas, donde cada capa tiene una funcionalidad específica.

5.1.- Intermediarios: Puedes tener intermediarios como servidores proxy, gateways y firewalls para mejorar la escalabilidad y la seguridad.

5.2.- Transparencia: Cada capa no necesita conocer los detalles completos de otras capas. Esto permite más flexibilidad y modularidad.

6.- Código Bajo Demanda (Opcional)

Permite que el servidor proporcione código ejecutable al cliente.

6.1.- Scripts y applets: Puedes enviar scripts o applets para que el cliente los ejecute, aunque este principio es opcional y no siempre se aplica.

Desarrollar una API REST en C# y .NET8 que permita interactuar con una base de datos MySQL y exportar datos a un archivo Excel. La API debe seguir los principios de diseño del estilo arquitectónico REST y utilizar los verbos HTTP estándar (GET, POST, PUT, DELETE) para interactuar con los recursos.

1.- Base de Datos: **Import_Tech**

Sus Tablas:

categoria_producto

cliente

factura_venta

producto

proveedor

venta_producto

A continuación, se detallan los requisitos y características de la API:

a.- Interfaz Uniforme: Todas las solicitudes para el mismo recurso deben tener el mismo aspecto, independientemente de su origen.

b.- Desacoplamiento Cliente-Servidor: Las aplicaciones cliente y servidor deben ser independientes entre sí.

c.- Sin Estado: Cada solicitud debe contener toda la información necesaria para procesarla, sin depender de solicitudes anteriores.

d.- Orientación a Recursos: Utiliza las operaciones estándar de los verbos HTTP (GET, POST, PUT, DELETE) para interactuar con los recursos.

2.- La Estructura del Proyecto es la siguiente:

El proyecto debe estar compuesto por las siguientes clases y componentes:

a.- DatabaseConnection: Clase para obtener y verificar la cadena de conexión a la base de datos MySQL.

b.- DatabaseHelper: Clase para ejecutar consultas SQL y manejar la conexión a la base de datos.

c.- ExcelExporter: Clase para exportar datos desde un DataTable a un archivo Excel.

d.- Program: Clase principal que coordina la ejecución de las clases anteriores y es el punto de entrada del programa.

e.- Controllers: Controladores para manejar las solicitudes HTTP y definir los endpoints de la API.

f.- Services: Servicios que contienen la lógica de negocio y se comunican con los controladores y la base de datos.

3.- El siguiente es un Bosquejo del Paso a Paso que se debe mejorar, si se puede:

3.1.- Configurar el Proyecto:

a.- Crear nuevo proyecto en Visual Studio.

b.- Instalar los paquetes necesarios: MySql.Data, ClosedXML, y Microsoft.AspNetCore.Mvc.

3.2.- Definir Estructura del Proyecto:

a.- Crear las carpetas Controllers, Models, Services, y Helpers.

3.3.- Configurar Conexión a la Base de Datos:

a.- En el archivo appsettings.json, agregar la cadena de conexión a la base de datos MySQL.

b.- Crear la clase DatabaseConnection para manejar la conexión a la base de datos.

3.4.- Crear Modelos de Datos:

a.- Definir las clases que representan las entidades de la base de datos en la carpeta Models.

3.5.- Implementar Servicios:

a.- Crear la clase DatabaseHelper para ejecutar consultas SQL.

b.- Crear la clase ExcelExporter para exportar datos a un archivo Excel.

3.6.- Desarrollar Controladores:

a.- Crear controladores en la carpeta Controllers para manejar las solicitudes HTTP.

b.- Definir los endpoints para las operaciones CRUD (Create, Read, Update, Delete).

3.7.- Configurar el Middleware:

a.- En el archivo Startup.cs, configurar los servicios y el middleware

b.- necesarios para la API.

3.8.- Probar API:

a.- Utilizar herramientas como Postman para probar los endpoints de la API.

b.- Asegurarse de que todas las operaciones (GET, POST, PUT, DELETE) funcionen correctamente.

Paso a paso para desarrollar una API REST

1.- Planificación y Diseño

1.1.- Definir los requisitos:

¿Qué funciones debe realizar la API?

¿Qué datos necesita manejar?

1.2.- Identificar recursos:

¿Cuáles son los principales recursos de la API (por ejemplo, usuarios, productos, pedidos)?

Define las URL para cada recurso.

1.3.- Decidir el formato de los datos:

Generalmente JSON o XML.

2.- Configuración del Entorno de Desarrollo

2.1.- Seleccionar el lenguaje y framework:

Por ejemplo, .NET, Django, Flask, Express.js, Spring Boot.

2.2.- Instalar herramientas necesarias:

IDE (Visual Studio, IntelliJ, etc.), gestores de paquetes (npm, NuGet), bases de datos (MySQL, PostgreSQL, etc.).

3.- Inicialización del Proyecto

3.1.- Crear el proyecto:

Por ejemplo, en .NET: dotnet new webapi -n MyAPI.

3.2.- Configurar el archivo de configuración:

Configura el archivo appsettings.json para incluir la conexión a la base de datos y otras configuraciones relevantes.

4.- Definir Modelos de Datos

4.1.- Crear clases de modelos:

Define las clases que representan las entidades de la base de datos. Por ejemplo, en una API para productos podrías tener una clase Product.

5.- Configurar la Conexión a la Base de Datos

5.1.- Definir la cadena de conexión:

Añadir la cadena de conexión en el archivo de configuración.

5.2.- Implementar la clase de conexión:

Crear una clase que maneje la conexión a la base de datos.

6. Crear los Endpoints de la API

6.1.- Definir los controladores:

Crear controladores para manejar las solicitudes HTTP y definir los endpoints.

6.2.- Implementar las acciones:

Crear métodos que manejen las acciones CRUD (Create, Read, Update, Delete).

7.- Añadir Lógica de Negocio

7.1.- Servicios:

Crear servicios que contengan la lógica de negocio y se comuniquen con los controladores y la base de datos.

8.- Configuración del Middleware y Pruebas

8.1.- Configurar el Middleware:

Configurar los servicios y el middleware en Startup.cs.

8.2.- Probar la API:

Utilizar herramientas como Postman o cURL para probar los endpoints de la API.

Asegurarse que todas las operaciones (GET, POST, PUT, DELETE) funcionen correctamente.

9.- Documentación

9.1.- Documentar la API:

Utilizar herramientas como Swagger para documentar la API.

Asegurarse de incluir ejemplos de solicitudes y respuestas.

10. –Despliegue

10.1.- Preparar para despliegue:

Configurar el entorno de producción.

Desplegar la API en un servidor o en la nube (por ejemplo, Azure, AWS).

10.2.- Monitoreo y mantenimiento:

Configurar herramientas para monitorear la salud y el rendimiento de la API.

Planificar actualizaciones y mantenimiento regular.

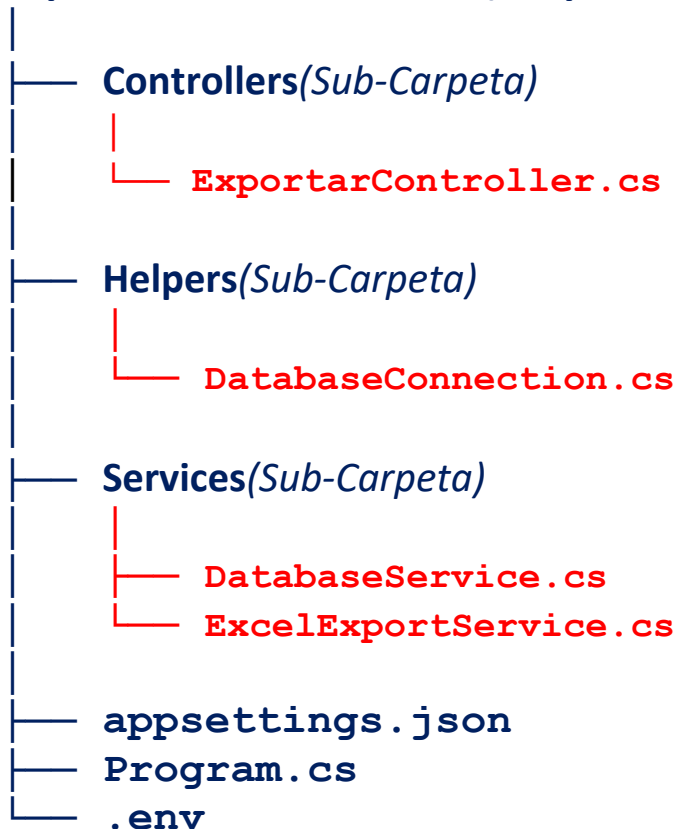
Propósito y Ubicación de cada archivo

- 1.- **appsettings.json**: Guardar en la carpeta **Raíz** del proyecto.
Contiene las configuraciones necesarias, como la cadena de conexión a la base de datos.
Guarda configuraciones generales del proyecto, sin incluir datos sensibles.
- 2.- **DatabaseConnection.cs**: Guardar en la carpeta **Helpers**.
Clase para obtener y verificar la cadena de conexión a la base de datos MySQL.
- 3.- **DatabaseService.cs**: Guardar en la carpeta **Services**.
Clase para ejecutar consultas SQL y manejar la conexión a la base de datos.
- 4.- **ExcelExportService.cs**: Guardar en la carpeta **Services**.
Clase para exportar datos desde un DataTable a un archivo Excel.
- 5.- **ExportarController.cs**: Guardar en la carpeta **Controllers**.
Controlador para manejar las solicitudes HTTP y definir los endpoints de la API.
- 6.- **Program.cs**: Guardar en la carpeta **Raíz** del proyecto
Clase principal que coordina la ejecución de las clases y es el punto de entrada del proyecto.
- 7.- **env**: Guardar en la carpeta **Raíz** del proyecto y **NO** se agrega al control de versiones.
Contiene la cadena de conexión completa con credenciales sensibles.

PROYECTO DEFINITIVO DE LA API

Estructura del Proyecto

ExportarDatosAExcelAPI(*Carpeta Raíz del Proyecto*)



Archivos del Proyecto

1.- appsettings.json (Guardar en la carpeta raíz del proyecto)

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

2.- .env (Guardar en la carpeta raíz del proyecto) - **NO agregar al control de versiones**

```
DB_CONNECTION_STRING=Server=localhost;Database=Import_Tech;Port=3306;UserId=root;Password=3001Epica**
```

3.- ExportarController.cs (Guardar en la carpeta Controllers)

```
using Microsoft.AspNetCore.Mvc;
using System.Data;
using ExportarDatosAExcelAPI.Helpers;
using ExportarDatosAExcelAPI.Services;
using System;

namespace ExportarDatosAExcelAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ExportarController : ControllerBase
    {
        private readonly DatabaseConnection _databaseConnection;
        private readonly DatabaseService _databaseService;
        private readonly ExcelExportService _excelExportService;

        public ExportarController(DatabaseConnection databaseConnection,
            DatabaseService databaseService, ExcelExportService excelExportService)
        {
            _databaseConnection = databaseConnection;
            _databaseService = databaseService;
            _excelExportService = excelExportService;
        }

        [HttpGet("{table}")]
        public IActionResult ExportarDatosAExcel(string table)
        {

```

```

string connectionString = _databaseConnection.GetConnectionString();
    string query = $"SELECT * FROM {table}";
    string filePath = $"{table}_Export.xlsx";

    try
    {
        DataTable dataTable = _databaseService.ExecuteQuery(query);
        if (dataTable.Rows.Count > 0)
        {
            _excelExportService.ExportToExcel(dataTable, filePath, table);
            return Ok(new { Message = "Datos exportados exitosamente",
FilePath = filePath });
        }

        else
        {
            return NotFound(new { Message = "No se encontraron datos para exportar" });
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { Message = "Error: " + ex.Message });
    }
}
}
}

```

4.- DatabaseConnection.cs *(Guardar en la carpeta Helpers)*

```

using System;
using MySql.Data.MySqlClient;
using DotNetEnv;
using System.IO;

namespace ExportarDatosAExcelAPI.Helpers
{
    public class DatabaseConnection
    {
        public string GetConnectionString()
        {
            string connectionString =
Environment.GetEnvironmentVariable("DB_CONNECTION_STRING");
            if (string.IsNullOrEmpty(connectionString))
            {
                connectionString = BuildConnectionStringFromUserInput();
            }
        }
    }
}

```

```

        return connectionString;
    }
    private string BuildConnectionStringFromUserInput()
    {
        string connectionString = null;
        bool isConnected = false;
        while (!isConnected)
        {
            Console.Write("Ingrese el usuario de la base de datos: ");
            string user = Console.ReadLine();
            Console.Write("Ingrese la contraseña de la base de datos: ");
            string password = ReadPassword();
            Console.Write("Ingrese Nombre del Servidor: ");
            string server = Console.ReadLine();
            Console.Write("Ingrese Nombre de la Base de Datos: ");
            string database = Console.ReadLine();
            Console.Write("Ingrese Número de Puerto: ");
            string port = Console.ReadLine();

            connectionString =
$"Server={server};Database={database};Port={port};UserId={user};Password={password}";

            try
            {
                using var connection = new MySqlConnection(connectionString);
                connection.Open();
                isConnected = true;
            }
            catch (MySqlException)
            {
                Console.WriteLine("Error de conexión. Inténtelo de nuevo.");
            }
        }
        return connectionString;
    }

    private string ReadPassword()
    {
        string password = string.Empty;
        ConsoleKeyInfo key;
        do
        {
            key = Console.ReadKey(true);
            if (key.Key != ConsoleKey.Backspace && key.Key != ConsoleKey.Enter)

```



```

        {
            password += key.KeyChar;
            Console.Write("*");
        }
        else if (key.Key == ConsoleKey.Backspace && password.Length > 0)
        {
            password = password.Substring(0, password.Length - 1);
            Console.Write("\b \b");
        }
    } while (key.Key != ConsoleKey.Enter);
    Console.WriteLine();
    return password;
}
}
}

```

5.- DatabaseService.cs *(Guardar en la carpeta Services)*

```

using System;
using System.Data;
using ExportarDatosAExcelAPI.Helpers;
using MySql.Data.MySqlClient;

namespace ExportarDatosAExcelAPI.Services
{
    public class DatabaseService
    {
        private readonly string _connectionString;

        public DatabaseService(DatabaseConnection databaseConnection)
        {
            _connectionString = databaseConnection.GetConnectionString();
        }

        public DataTable ExecuteQuery(string query)
        {
            DataTable dataTable = new DataTable();
            try
            {
                using MySqlConnection connection = new MySqlConnection(_connectionString);
                connection.Open();
                using MySqlCommand command = new MySqlCommand(query, connection);
                using MySqlDataAdapter dataAdapter = new MySqlDataAdapter(command);
                dataAdapter.Fill(dataTable);
            }
        }
    }
}

```

```

        catch (Exception ex)
        {
            Console.WriteLine("Error: " + ex.Message);
        }
        return dataTable;
    }
}
}

```

6.- ExcelExportService.cs *(Guardar en la carpeta Services)*

```

using System.Data;
using ClosedXML.Excel;

namespace ExportarDatosAExcelAPI.Services
{
    public class ExcelExportService
    {
        public void ExportToExcel(DataTable dataTable, string filePath, string table)
        {
            using var workbook = new XLWorkbook();
            var worksheet = workbook.Worksheets.Add(table);
            worksheet.Cell(1, 1).InsertTable(dataTable);
            workbook.SaveAs(filePath);
        }
    }
}

```

7.- Program.cs *(Guardar en la carpeta raíz del proyecto)*

```

using ExportarDatosAExcelAPI.Helpers;
using ExportarDatosAExcelAPI.Services;
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

var builder = WebApplication.CreateBuilder(args);

// Configurar servicios
builder.Services.AddControllers();
builder.Services.AddSingleton<DatabaseConnection>();
builder.Services.AddSingleton<DatabaseService>();
builder.Services.AddSingleton<ExcelExportService>();

var app = builder.Build();

```

```
// Configurar middleware
app.MapControllers();
app.Run();
```

8.- launchSettings.json (Archivo se crea cuando compilamos el Proyecto)

Es importante revisar este archivo **.json**; toda vez que, este archivo se encuentra en la carpeta **Properties** y define cómo se inicia la aplicación durante el desarrollo.

Particularmente en este proyecto, podremos definir el puerto en el cual trabajaremos.

```
{
  "profiles": { //Define diferentes perfiles de lanzamiento.

    "ExportarDatosAExcelAPI": { //Nombre del perfil.

      "commandName": "Project", //Indica que se va a ejecutar como un proyecto.

      "launchBrowser": true, //Si es true, abre el navegador automáticamente al iniciar la aplicación.

      "environmentVariables": { //Variables de entorno que se usarán durante el desarrollo .

        "ASPNETCORE_ENVIRONMENT": "Development" /* Variable de entorno utilizada por ASP.NET Core
para especificar el entorno en el que se está ejecutando la aplicación, indica el modo o entorno en el que se está
ejecutando la aplicación ASP.NET Core. Esta variable es crucial porque permite que la aplicación se comporte de
manera diferente en función del entorno en el que esté operando.
Los entornos más comunes que puedes especificar son:
1.- Development: Entorno de desarrollo (donde se desarrolla y prueba la aplicación).
2.- Staging: Entorno de prueba antes de ir a producción (puede ser opcional).
3.- Production: Entorno de producción (donde la aplicación está disponible para los
usuarios finales).*/

      },

      "applicationUrl": "https://localhost:5001;http://localhost:5000" /*Define las
URLs donde se ejecutará la aplicación (https en el puerto 5001 y http en el puerto
5000). Por defecto, ASP.NET Core admite estos tres entornos, pero se pueden definir
otros personalizados si es necesario.*/

    }
  }
}
```

"ASPNETCORE_ENVIRONMENT": "Development"

Cuando `ASPNETCORE_ENVIRONMENT` se establece en "Development", la aplicación asume que se está ejecutando en un entorno de desarrollo.

Esto tiene varias implicaciones:

- 1.- **Registro detallado:** *ASP.NET Core* habilita el nivel de registro más detallado (*Information* y *Debug*), lo que te proporciona más información en la consola y archivos de registro para depurar problemas.
- 2.- **Manejo de errores amigable:** Si ocurre una excepción, la aplicación mostrará una página detallada de errores que incluye el *stack trace* (*seguimiento de pila*) y otra información útil para la depuración. En el **entorno de producción**, en cambio, se mostraría una página de error genérica para no exponer detalles técnicos a los usuarios.
- 3.- **Recarga automática:** En el **entorno de desarrollo**, Visual Studio y el servidor Kestrel permiten la recarga automática de la aplicación (hot reload). Si se realizan cambios en el código y se guarda la aplicación se reinicia automáticamente para reflejar esos cambios.
- 4.- **Configuración de archivos:** *ASP.NET Core* cargará configuraciones específicas desde el archivo *appsettings.Development.json*, si existiese. Esto te permite tener configuraciones separadas para cada entorno, como cadenas de conexión a bases de datos y claves API.
- 5.- **Herramientas y servicios adicionales:** Algunas herramientas como **Swagger**, que es útil para documentar y probar las **API's**, se suelen habilitar solo en el entorno de desarrollo para evitar exponer la documentación en producción.

¿Dónde se configura ASPNETCORE_ENVIRONMENT?

El entorno se puede configurar en varios lugares:

- 1.- **Archivo launchSettings.json** (durante el desarrollo): Este archivo se encuentra en la carpeta *Properties* del proyecto y es utilizado por Visual Studio para establecer la configuración del entorno local.
- 2.- **Variables de entorno del sistema:** Se puede establecer la variable de entorno directamente en el sistema operativo.

2.1.- Windows (CMD):

Se puede establecer la variable de entorno temporalmente en la línea de comandos (CMD)

cmd

```
setx ASPNETCORE_ENVIRONMENT Development /* setx establece la variable de entorno de manera permanente para futuras sesiones de terminal.*/
```

cmd

```
set ASPNETCORE_ENVIRONMENT=Development /* set establece la variable de entorno si solo se quiere configurar para la sesión actual.*/
```

2.2.- En PowerShell:

```
powershell
```

```
$Env:ASPNETCORE_ENVIRONMENT = "Development"
```

2.3.- En el Editor de Configuración del Sistema:

También se puede configurar la variable desde el **Panel de Control**:

- a.- Ir a **Sistema** → **Configuración avanzada del sistema**.
- b.- Hacer clic en **Variables de entorno**.
- c.- Añadir una nueva variable con:
Nombre: ASPNETCORE_ENVIRONMENT
Valor: Development

3.- Configuración en Servicios de Hosting

Cuando se despliega una aplicación **ASP.NET Core** en un servidor web como **IIS**, **Nginx**, o **Azure App Services**, es necesario asegurarse de que el entorno está configurado correctamente.

3.1.- En Azure App Services: Se puede configurar la variable en el **Portal de Azure**

- a.- Ir a **App Service**.
- b.- Seleccionar **Configuración** → **Configuración de la aplicación**.
- c.- Añadir una nueva variable con:
Nombre: ASPNETCORE_ENVIRONMENT
Valor: Production (u otro entorno, según sea necesario)

4.- ¿Qué pasa si no se establece **ASPNETCORE_ENVIRONMENT**?

Si no se configura explícitamente la variable, **ASP.NET Core** asume por defecto el entorno de **Production**, lo que implica:

- a.- Menos información de depuración en los registros.
- b.- Páginas de error genéricas para los usuarios finales (sin detalles del stack trace).
- c.- Deshabilitación de herramientas de desarrollo como **Swagger**.

Esto es ideal para proteger la seguridad y estabilidad de la aplicación en producción, evitando que información sensible sea expuesta a los usuarios.

5.- En Resumen:

ASPNETCORE_ENVIRONMENT permite distinguir entre entornos como **Development**, **Staging**, y **Production**.

5.1.- Se puede configurar a través de:

- a.- El archivo **launchSettings.json** en proyectos locales.
- b.- Variables de entorno del sistema.
- c.- Configuraciones en servicios de hosting como **Azure App Services**.

5.2.- Afecta la forma en que **se registran los errores**, **la información de depuración**, y la **carga de configuraciones específicas**.

EXPLICACIÓN DE CADA PARTE DE LA ESTRUCTURA DEL PROYECTO

1.- appsettings.json

```
{
  "Logging": { /*Esta sección controla cómo se gestionan los registros (logs) dentro de la aplicación */
    "LogLevel": { /*Define el nivel de severidad de los mensajes que se van a registrar en los logs */

      "Default": "Information", /*Este es el nivel de registro por defecto para tu aplicación. En este caso,
está configurado en Information, lo que significa que se registrarán mensajes informativos, advertencias y errores,
pero no mensajes de Debug (depuración) */

      "Microsoft": "Warning", /*Controla el nivel de registro para los logs generados por las bibliotecas
internas de Microsoft. Aquí está configurado en Warning, por lo que solo se registrarán advertencias y errores
provenientes de las bibliotecas de Microsoft */

      "Microsoft.Hosting.Lifetime": "Information" /*Se refiere específicamente al ciclo de vida del
servidor de la aplicación (por ejemplo, cuando la aplicación se inicia o se detiene). Está configurado en Information
para obtener detalles informativos sobre eventos importantes como el inicio o parada de la aplicación */
    },
    "AllowedHosts": "*" /*Especifica qué nombres de host (dominios) pueden enviar solicitudes a tu aplicación.
El valor "*" significa que se permiten todos los hosts. Es útil para entornos de desarrollo donde no quieres restringir
el acceso a tu aplicación, pero no es recomendable para entornos de producción debido a preocupaciones de
seguridad */
  }
}
```

En resumen:

- 1.- **Logging:** Controla qué tipo de mensajes se registran y en qué nivel de detalle.
- 2.- **AllowedHosts:** Establece qué dominios pueden acceder a la aplicación.
- 3.- **El archivo appsettings.json:** Es una herramienta flexible que permite ajustar configuraciones de forma sencilla y específica para diferentes entornos.

2.- DatabaseConnection.cs

```
using System; /*Importa funciones básicas del lenguaje C#, como la entrada y salida en
la consola.*/
using MySql.Data.MySqlClient; /*Importa el paquete para trabajar con bases de datos
MySQL.*/
using DotNetEnv; /*Permite cargar variables de entorno desde un archivo .env. Esto es
útil para mantener las credenciales seguras y fuera del código fuente.*/
using System.IO; /*Proporciona funciones para manejar operaciones de archivos y
flujos.*/

namespace ExportarDatosAExcelAPI.Helpers /*Define un espacio de nombres que agrupa las
clases relacionadas dentro de tu proyecto. Aquí, Helpers indica que esta clase tiene
funciones de ayuda o utilidad.*/
```

```

{
    public class DatabaseConnection /*Define una clase pública llamada DatabaseConnection
    que contiene métodos para manejar la conexión a la base de datos.*/
    {
        public string GetConnectionString() /*Es el método principal que intenta obtener
        la cadena de conexión.*/
        {
            string connectionString = Environment.GetEnvironmentVariable("DB_CONNECTION_STRING");
            /*Intenta obtener la cadena de conexión desde una variable de entorno llamada
            DB_CONNECTION_STRING. Si está configurada, la aplicación la usará para conectarse
            a la base de datos.*/

            if (string.IsNullOrEmpty(connectionString)) /*Si la variable de entorno no
            está configurada (null o está vacía), invoca el método connectionStringFromUserInput()
            para que el usuario introduzca manualmente los datos de conexión.*/
            {
                connectionString = BuildConnectionStringFromUserInput();
            }
            return connectionString;
        }

        private string BuildConnectionStringFromUserInput() /*Método que construye la
        cadena de conexión solicitando al usuario que introduzca los datos manualmente.*/
        {
            string connectionString = null; /*Declara una variable llamada connectionString
            de tipo string.
            Se inicializa la variable con un valor null, lo que significa que al inicio no
            contiene ninguna cadena.
            Esta variable almacenará la cadena de conexión a la base de datos una vez que el
            usuario introduzca todos los datos requeridos (servidor, usuario, contraseña,
            base de datos, puerto).*/

            bool isConnected = false; /*Declara una variable llamada isConnected de
            tipo bool (booleano).
            Se inicializa la variable con el valor false.
            Esta variable se usa para controlar el bucle while.
            Su propósito es intentar establecer la conexión de forma repetida hasta
            que sea exitosa.*/

            while (!isConnected) /*Inicia un bucle while que se ejecutará mientras la
            variable isConnected sea false.
            El signo ! antes de isConnected significa "no", por lo que la condición se
            traduce como "mientras no
            esté conectado". El bucle continuará solicitando al usuario que ingrese
            los datos (usuario, contraseña,
            servidor, etc) hasta que la conexión sea exitosa.*/

```

```

{
    Console.WriteLine("Ingrese el usuario de la base de datos: ");
    string user = Console.ReadLine(); /*Nombre del usuario de la base de
    datos.*/
    Console.WriteLine("Ingrese la contraseña de la base de datos: ");
    string password = ReadPassword(); /*La contraseña se ingresa de forma
    oculta (ver método ReadPassword() más abajo).*/
    Console.WriteLine("Ingrese Nombre del Servidor: ");
    string server = Console.ReadLine(); /*Dirección IP o nombre del
    servidor donde está alojada la base de datos.*/
    Console.WriteLine("Ingrese Nombre de la Base de Datos: ");
    string database = Console.ReadLine(); /*El nombre de la base de datos
    a la que se quiere conectar.*/
    Console.WriteLine("Ingrese Número de Puerto: ");
    string port = Console.ReadLine(); /*Número de puerto (generalmente 3306
    para MySQL).*/
    /*construye la cadena de conexión a la base de datos utilizando los
    datos que el usuario ha ingresado (server, database, port, user,
    password).
    El signo $ al inicio de la cadena indica que es una cadena interpolada.
    Esto permite incluir variables directamente dentro de la cadena usando
    {}.*/
    connectionString =
$"Server={server};Database={database};Port={port};UserId={user};Password={password}";

    try
    {
        using var connection = new MySqlConnection(connectionString);
        connection.Open(); /*Intenta conectar a la base de datos utilizando la
        cadena proporcionada "connectionString"*/
        isConnected = true; /*Si la conexión es exitosa, se establece
        isConnected = true y el bucle while finaliza.*/
    }
    catch (MySqlException) /*Si ocurre un error (por ejemplo, credenciales
    incorrectas o servidor inalcanzable), muestra el mensaje de error y
    permite al usuario intentar nuevamente.*/
    {
        Console.WriteLine("Error de conexión. Inténtelo de nuevo.");
    }
}
return connectionString;
}

```



```

private string ReadPassword() /*Método que tiene como objetivo leer una contraseña
desde la consola de forma segura.
Cuando el usuario escribe, los caracteres que introduce no se muestran en la
pantalla; en su lugar, se muestra un asterisco (*) por cada carácter ingresado.
Además, el método permite al usuario usar la tecla de retroceso (Backspace) para
corregir la entrada si es necesario.*/
{
    string password = string.Empty; /*Inicializa la variable password como una
cadena vacía y esta variable almacenará la contraseña ingresada por el usuario.*/

    ConsoleKeyInfo key; /*Declara una variable llamada key de tipo ConsoleKeyInfo.
Esta variable se utiliza para almacenar la tecla que el usuario presiona.*/
    do /*Inicia un bucle do-while que continuará hasta que el usuario presione
la tecla Enter (ConsoleKey.Enter).*/
    {
        key = Console.ReadKey(true); /*Lee una tecla que el usuario presiona.
El parámetro true indica que la tecla no se mostrará en la consola, lo
que significa que la entrada será oculta.*/
        //Manejo de Caracteres Ingresados
        if (key.Key != ConsoleKey.Backspace && key.Key != ConsoleKey.Enter
            /*Condición que asegura de que la tecla presionada no sea Backspace ni que
tampoco sea Enter..*/
        {
            password += key.KeyChar; /*Si la tecla presionada no es Backspace ni
Enter, entonces agrega el carácter ingresado por el usuario a la
variable password.*/

            Console.Write("*"); /*Muestra un asterisco (*) en la consola en
lugar del carácter real para ocultar la entrada del usuario.*/
        }
        //Manejo de la Tecla Backspace
        else if (key.Key == ConsoleKey.Backspace && password.Length > 0) /*Condición
que Verifica si la tecla presionada es Backspace y si la contraseña tiene al
menos un carácter (password.Length > 0).
Si ambas condiciones son verdaderas ejecuta la sección siguiente*/
        {
            password = password.Substring(0, password.Length - 1);/*Elimina el
último carácter de la cadena password.*/
            Console.Write("\b \b"); /*secuencia especial para eliminar el
último asterisco (*) de la consola.
\b (backspace): Mueve el cursor una posición hacia atrás.
(espacio en blanco): Sobrescribe el asterisco con un espacio en blanco.
\b (backspace): Mueve el cursor una posición hacia atrás nuevamente para
que quede en posición para el próximo carácter.*/
        }
    }
}

```

```

        //Fin del Bucle y Retorno del Resultado
        while (key.Key != ConsoleKey.Enter); /*El bucle continúa repitiéndose
        hasta que el usuario presiona la tecla Enter.*/
        Console.WriteLine();
        return password; /*Retorna la contraseña ingresada como una cadena de
        texto (string).*/
    }
}
}

```

En resumen

1.- La clase DatabaseConnection realiza las siguientes funciones:

- 1.1.- **Obtiene una cadena de conexión** de una variable de entorno (DB_CONNECTION_STRING).
- 1.2.- **Si no está configurada**, solicita al usuario que introduzca manualmente los datos.
- 1.3.- **Verifica la conexión** a la base de datos antes de proceder, para asegurarse de que los datos proporcionados son correctos.

2.- Ventajas del Enfoque Utilizado

- 2.1.- **Seguridad:** No se almacenan credenciales directamente en el código fuente.
- 2.2.- **Flexibilidad:** Permite cambiar fácilmente la configuración de la base de datos sin necesidad de recompilar la aplicación.
- 2.3.- **Interactividad:** Si no se encuentra la variable de entorno, el usuario puede proporcionar la información manualmente.

3.- DatabaseService.cs

```

using System; /*Importa el espacio de nombres System, que proporciona funcionalidades
básicas, como manejo de excepciones (Exception).*/
using System.Data; /*Importa el espacio de nombres para trabajar con estructuras de
datos, como DataTable.*/
using ExportarDatosAExcelAPI.Helpers; /*Permite usar la clase DatabaseConnection, que
obtendrá la cadena de conexión a la base de datos.*/
using MySql.Data.MySqlClient; /*Importa la biblioteca de MySQL para trabajar con bases
de datos MySQL en .NET*/

//Definición del Espacio de Nombres y la Clase
namespace ExportarDatosAExcelAPI.Services /*Define el espacio de nombres donde se
encuentra esta clase (Services).*/
{
    public class DatabaseService /*Declara la clase pública DatabaseService que
    manejará la conexión y las consultas a la base de datos.*/
    {

```

//VARIABLE DE CLASE

```
private readonly string _connectionString; /*Declara una variable privada y de solo lectura llamada _connectionString. Esto significa que solo se puede asignar un valor a esta variable una vez, generalmente durante la inicialización (en el constructor).
```

Esta variable (_connectionString) almacenará la cadena de conexión obtenida de la clase DatabaseConnection.*/
*/

//CONSTRUCTOR DE CLASE

```
public DatabaseService(DatabaseConnection databaseConnection) /*Declara un constructor que se llama automáticamente cuando se crea una instancia de la clase DatabaseService.
```

Recibe un parámetro del tipo DatabaseConnection.*/
*/

```
{  
    _connectionString = databaseConnection.GetConnectionString(); /*Usa el método GetConnectionString() de la clase DatabaseConnection para obtener la cadena de conexión.  
    Asigna esta cadena a la variable de solo lectura ( _connectionString )*/  
}
```

//METODO ExecuteQuery: Ejecutar Consultas SQL

```
public DataTable ExecuteQuery(string query) /*Declara un método público llamado ExecuteQuery que recibe un parámetro de tipo string llamado query.
```

Devuelve un DataTable, que es una tabla en memoria que almacena los resultados de la consulta.*/
*/

```
{  
    DataTable dataTable = new DataTable(); /*Crea una nueva instancia de DataTable, que se utilizará para almacenar los resultados de la consulta SQL.*/  
    */
```

//Manejo de la Conexión y Ejecución de la Consulta

```
try /*Inicia un bloque try para manejar excepciones que puedan ocurrir durante la conexión o la consulta.*/  
*/
```

```
{  
    using MySqlConnection connection = new MySqlConnection(_connectionString); /*Crea una nueva conexión con la base de datos utilizando la cadena de conexión almacenada en ( _connectionString ).  
    El uso de using asegura que la conexión se cierre automáticamente al salir del bloque.*/  
    */
```

```
    connection.Open(); /*Abre la conexión con la base de datos*/  
    */
```

//Crear y Ejecutar el Comando SQL

```
using MySqlCommand command = new MySqlCommand(query, connection); /*Crea un
comando SQL utilizando la consulta proporcionada (query) y la conexión
previamente abierta.
El comando es el que se ejecutará en la base de datos.*/
using MySqlDataAdapter dataAdapter = new MySqlDataAdapter(command); /*Crea un
adaptador de datos que se encarga de ejecutar el comando SQL y llenar el
DataTable con los resultados.*/

dataAdapter.Fill(dataTable); /*Llena el DataTable con los resultados
de la consulta SQL ejecutada.*/
}
```

//MANEJO DE ERRORES

```
catch (Exception ex) /*Si ocurre algún error durante la ejecución de la
consulta o la conexión, se captura en este bloque.*/
{
    Console.WriteLine("Error: " + ex.Message); /*Imprime en la consola un
mensaje de error, junto con el detalle del error (ex.Message).*/
}
return dataTable; /*Finalmente, el método devuelve el DataTable que
contiene los resultados de la consulta.
Si hubo un error durante la ejecución, se devolverá un DataTable vacío.*/
}
}
```

La clase **DatabaseService** está diseñada para **ejecutar consultas SQL** y **gestionar la conexión** a una base de datos MySQL. Utiliza la conexión que se obtiene a través de la clase **DatabaseConnection** y está orientada a **consultar datos** (es decir, ejecutar sentencias SELECT) y devolverlos en un formato que se puede manejar en la aplicación (DataTable).

Resumen del Flujo del Método ExecuteQuery

- 1.- Recibe una consulta SQL como parámetro (query).
- 2.- Abre una conexión a la base de datos utilizando la cadena de conexión.
- 3.- Ejecuta la consulta y llena un DataTable con los resultados.
- 4.- Si ocurre un error, lo muestra en la consola.
- 5.- Devuelve el DataTable con los datos obtenidos

4.- ExcelExportService.cs

```
using System.Data; /*Importa el espacio de nombres para trabajar con estructuras de
datos, como el DataTable.*/
using ClosedXML.Excel; /*Importa la biblioteca ClosedXML, que facilita la creación,
edición y manipulación de archivos Excel (.xlsx) en .NET.*/
```

//DEFINICION DEL ESPACIO DE NOMBRES Y LA CLASE

```
namespace ExportarDatosAExcelAPI.Services /*Define el espacio de nombres donde se
encuentra esta clase (en Services), lo que ayuda a organizar el proyecto.*/
{
    public class ExcelExportService /*Declara la clase pública ExcelExportService, que
    contiene métodos para exportar datos a un archivo Excel.*/
    {
        //METODO ExportToExcel
        public void ExportToExcel(DataTable dataTable, string filePath, string table)
        /*Define un método público llamado ExportToExcel.
        No devuelve ningún valor (void).
        Recibe tres parámetros:
            1.- DataTable dataTable: La tabla con los datos que se van a exportar a Excel.
            2.- string filePath: La ruta donde se guardará el archivo Excel.
            3.- string table: El nombre de la hoja (worksheet) que se creará en el archivo Excel.*/
        {
            //CREAR LIBRO DE Excel (workbook)
            using var workbook = new XLWorkbook(); /*Crea un nuevo libro de Excel
            (XLWorkbook).
            El uso de (using) asegura que los recursos utilizados por workbook se
            liberen automáticamente al salir del bloque (es decir, no tendrás fugas de
            memoria).*/
            //AGREGAR UNA HOJA DE CALCULO (worksheet)
            var worksheet = workbook.Worksheets.Add(table); /*Crea una nueva hoja de
            cálculo dentro del libro (workbook) y le asigna el nombre proporcionado
            por el parámetro table.
            La hoja se utilizará para insertar y organizar los datos del DataTable.*/
            //INSERTAR EL DataTable EN LA HOJA DE CALCULO
            worksheet.Cell(1, 1).InsertTable(dataTable); /*Selecciona la celda A1
            (primera fila, primera columna) de la hoja de cálculo.
            Usa el método InsertTable(dataTable) para insertar los datos del DataTable
            en la hoja de cálculo.
            El método automáticamente genera encabezados de columnas basados en los
            nombres de las columnas del DataTable.*/
            //GUARDAR EL ARCHIVO
            workbook.SaveAs(filePath); /*Guarda el libro de Excel (workbook) en la
            ruta especificada por el parámetro filePath.
            Si el archivo ya existe, lo sobrescribirá.*/
        }
    }
}
```

La clase **ExcelExportService** tiene la responsabilidad de **exportar datos** que se encuentran en un **DataTable** hacia un archivo **Excel** utilizando la biblioteca **ClosedXML**. Esta funcionalidad es útil cuando se necesitan **reportes** o exportar datos en un formato legible y estándar, como Excel.

Resumen del Flujo del Método ExportToExcel

- 1.- Recibe un DataTable, la ruta del archivo y el nombre de la hoja.
- 2.- Crea un nuevo libro de Excel (workbook).
- 3.- Añade una hoja de cálculo con el nombre proporcionado.
- 4.- Inserta los datos del DataTable en la hoja de cálculo, comenzando desde la celda A1.
- 5.- Guarda el archivo en la ruta especificada (filePath).

5.- ExportarController.cs

```
//Importación de espacios de nombres
using Microsoft.AspNetCore.Mvc; /*Importa el espacio de nombres necesario para trabajar con
controladores y acciones en ASP.NET Core.*/
using System.Data; /*Permite el uso de la clase DataTable, que se utiliza para manipular
datos tabulares.*/
using ExportarDatosAExcelAPI.Helpers; /*Importa las clases personalizadas del proyecto
(DatabaseConnection)*/
using ExportarDatosAExcelAPI.Services; /*Importa las clases personalizadas del
proyecto (DatabaseService, y ExcelExportService).*/
using System; /*Permite trabajar con excepciones y tipos básicos.*/

//Definición del Espacio de Nombres y la Clase
namespace ExportarDatosAExcelAPI.Controllers /*Define el espacio de nombres para
organizar el código.*/
{
    [ApiController] /*Marca la clase como un controlador de API.
    Esto habilita funcionalidades automáticas como validación de modelos y respuestas
    de error.*/

    [Route("api/[controller]")] /*Define la ruta base para este controlador.
    [controller] se reemplaza por el nombre del controlador (Exportar), por lo que la
    ruta final será: api/Exportar */

    public class ExportarController : ControllerBase /*Declara la clase pública
    ExportarController, que hereda de ControllerBase para manejar solicitudes HTTP.*/
    {
        //Definición de los Campos Privados
        private readonly DatabaseConnection _databaseConnection; /*Declara campo privado
        para almacenar instancias de DatabaseConnection (Para obtener la cadena de
        conexión).*/

        private readonly DatabaseService _databaseService; /*Declara campo privado
        para almacenar instancias de DatabaseService (Para ejecutar consultas SQL).*/

        private readonly ExcelExportService _excelExportService; /*Declara campo privado
        para almacenar instancias de ExcelExportService (Para exportar datos a Excel).*/
    }
}
```

//Constructor del Controlador

```
public ExportarController(DatabaseConnection databaseConnection,
DatabaseService databaseService, ExcelExportService excelExportService)
/*Constructor que recibe tres parámetros:
    1.- databaseConnection: Inyecta una instancia de DatabaseConnection.
    2.- databaseService: Inyecta una instancia de DatabaseService.
    3.- excelExportService: Inyecta una instancia de ExcelExportService.
Inyección de dependencias: Permite que estas clases se inyecten automáticamente al
crear el controlador, facilitando el manejo de dependencias y la modularidad.*/
{
    _databaseConnection = databaseConnection;
    _databaseService = databaseService;
    _excelExportService = excelExportService;
}
```

//Método ExportarDatosAExcel

[HttpGet("{table}")] /*Indica que este método responde a solicitudes HTTP GET en la ruta: api/Exportar/{table}.
En donde {table} es un parámetro de la ruta que representa el nombre de la tabla a consultar*/

```
public IActionResult ExportarDatosAExcel(string table) /*Define un método que devuelve un IActionResult, lo que permite devolver diferentes tipos de respuestas HTTP como: (200 OK, 404 Not Found, 500 Internal Server Error).*/
{
    //Obtener la Cadena de Conexión y la Consulta
    string connectionString = _databaseConnection.GetConnectionString(); /*Obtiene la cadena de conexión desde la clase DatabaseConnection*/

    string query = $"SELECT * FROM {table}"; /*Define la consulta SQL para seleccionar todos los registros de la tabla*/

    string filePath = $"{table}_Export.xlsx"; /*Define el nombre del archivo Excel que se generará.*/

    //Ejecutar la Consulta y Exportar a Excel
    try /*Inicia un bloque try-catch para manejar errores.*/
    {
        DataTable dataTable = _databaseService.ExecuteQuery(query); /*Ejecuta la consulta y devuelve un DataTable con los resultados.*/
    }
}
```

```

        if (dataTable.Rows.Count > 0) /*Verifica si la consulta retornó datos*/
        {
            _excelExportService.ExportToExcel(dataTable, filePath, table); /*llama
            al método ExportToExcel para exportar los datos al archivo Excel.*/

            return Ok(new { Message = "Datos exportados exitosamente", FilePath =
filePath }); /*Devuelve una respuesta 200 OK con un mensaje y la ruta del archivo generado.*/
        }
        else
        {
            return NotFound(new { Message = "No se encontraron datos para exportar" });
            /*Si no hay datos, devuelve una respuesta 404 Not Found con un mensaje
            indicando que no se encontraron datos.*/
        }
    }
}
catch (Exception ex) /*Captura cualquier excepción que pueda ocurrir.*/
{
    return StatusCode(500, new { Message = "Error: " + ex.Message });
    /*Devuelve una respuesta 500 Internal Server Error con el mensaje de
    error.*/
}
}
}
}
}

```

Este controlador está diseñado para:

- 1.- Recibir solicitudes HTTP (en este caso, solicitudes GET).
- 2.- Ejecutar una **consulta en la base de datos** para obtener los datos de una tabla.
- 3.- Exportar esos datos a un **archivo Excel** utilizando el servicio ExcelExportService.
- 4.- Devolver una **respuesta** indicando si la exportación fue exitosa.

Resumen del Flujo del Endpoint

- 1.- Recibe una solicitud GET con el nombre de la tabla como parámetro.
- 2.- Obtiene la **cadena de conexión** y define una consulta SQL.
- 3.- Ejecuta la consulta en la base de datos usando DatabaseService.
- 4.- Si hay resultados, los **exporta a un archivo Excel**.
- 5.- Devuelve una **respuesta JSON** indicando el estado (éxito, no encontrado, o error).

6.- Program.cs

```
using ExportarDatosAExcelAPI.Helpers; /*Importa la clase DatabaseConnection que se
encuentra en carpeta Helpers.*/
using ExportarDatosAExcelAPI.Services; /*Importa las clases DatabaseService y
ExcelExportService de la carpeta Services.*/
using Microsoft.AspNetCore.Builder; /*Contiene funcionalidades para configurar y
construir la aplicación.*/
using Microsoft.Extensions.DependencyInjection; /*Proporciona herramientas para
registrar servicios que se utilizarán en la aplicación.*/
using Microsoft.Extensions.Hosting; /*Proporciona soporte para configurar y ejecutar
el host de la aplicación.*/

//CREAR EL WebApplicationBuilder
var builder = WebApplication.CreateBuilder(args);/*
var: Es un tipo de declaración de variable en C#.
Se utiliza para que el compilador deduzca automáticamente el tipo de dato que va a
contener esa variable en función del valor que se le asigna.
builder: Variable que será de tipo WebApplicationBuilder, ya que el método
WebApplication.CreateBuilder() devuelve un objeto de ese tipo.
WebApplication.CreateBuilder(args): Inicializa un objeto WebApplicationBuilder, que se
utiliza para configurar la aplicación.
args: Son los argumentos que puedes pasar al ejecutar tu aplicación desde la línea de
comandos.
Esta línea configura los servicios, la configuración, y el entorno para tu aplicación.*/

//CARGAR VARIABLES DE ENTORNO DESDE EL ARCHIVO .env
Env.Load();

//CONFIGURAR SERVICIOS
builder.Services.AddControllers(); /*Registra el servicio de controladores para que
ASP.NET Core pueda manejar solicitudes HTTP y endpoints.*/

builder.Services.AddSingleton<DatabaseConnection>();
/*Registra la clase DatabaseConnection como un servicio de tipo Singleton.
Esto significa que se creará una única instancia de DatabaseConnection para toda la
aplicación y se reutilizará cada vez que se necesite.
Útil cuando necesitas conservar la misma cadena de conexión durante la vida útil de la
aplicación.*/

builder.Services.AddSingleton<DatabaseService>();
/*Similar al anterior, registra DatabaseService como un Singleton, ya que también se
beneficiará de una única
instancia.*/
```

```
builder.Services.AddSingleton<ExcelExportService>();  
/*Registra ExcelExportService como un Singleton para exportar datos a Excel de forma eficiente.
```

Usar AddSingleton es adecuado aquí ya que estos servicios no tienen estado y no necesitan ser recreados para cada solicitud.*/

//CONSTRUIR LA APLICACION

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

1.- builder.Build(): El método Build() se llama sobre el objeto builder (de tipo WebApplicationBuilder), el cual se configuró previamente en la línea anterior: **var builder = WebApplication.CreateBuilder(args);**

1.1.- Cuando se llama a builder.Build(), se construye (creando) la aplicación ASP.NET Core con todas las configuraciones y servicios que has definido anteriormente.

1.2.- Este método realiza varias tareas importantes:

a.- Compila la configuración: Toma toda la configuración que has establecido a través de builder, incluyendo servicios, middleware, y otras configuraciones que se haya añadido.

b.- Prepara la aplicación para ser ejecutada: Convierte el builder en un objeto WebApplication que puede ser usado para configurar el comportamiento de la aplicación y finalmente, ejecutarla.

c.- Configura el pipeline de la aplicación: Prepara el pipeline de middleware que manejará las solicitudes entrantes como controladores, autenticación, y otros componentes.

2.- var app: Después de ejecutar builder.Build(), el resultado se guarda en la variable app.

Esta variable ahora contiene un objeto WebApplication, que es la instancia de la aplicación web.

2.1.- El objeto app tiene la capacidad de:

a.- Configurar middleware, por ejemplo: (app.UseAuthentication(), app.UseAuthorization()).

b.- Mapear endpoints como: app.MapControllers() para que tus controladores manejen las solicitudes.

c.- Iniciar el servidor llamando a app.Run() para que la aplicación comience a escuchar solicitudes HTTP.*/

//CONFIGURACION DE middleware

```
app.MapControllers(); /*Configura los endpoints de la aplicación basados en los controladores que se han definido.
```

Esto significa que cualquier solicitud entrante se enruta a los controladores correspondientes.*/

```
app.Run(); /*Inicia la aplicación y pone el servidor a la escucha de las solicitudes HTTP en el puerto configurado por defecto, estos puertos suelen ser el puerto 5000 para HTTP y 5001 para HTTPS.*/  
  

```

Este archivo tiene dos responsabilidades principales

- 1.- Configurar los servicios necesarios para la aplicación tales como:
 - a.- DatabaseConnection
 - b.- DatabaseService
 - c.-ExcelExportService).
- 2.- Construir y ejecutar la aplicación, mapeando los controladores y ejecutando el servidor web.

Resumen del Flujo del Código

- 1.- **Importa** las clases necesarias.
- 2.- **Crea** un WebApplicationBuilder.
- 3.- **Registra los servicios** necesarios (DatabaseConnection, DatabaseService, ExcelExportService).
- 4.- **Construye** el objeto WebApplication.
- 5.- **Mapea los controladores** para que las rutas funcionen correctamente.
- 6.- **Ejecuta** la aplicación para comenzar a escuchar solicitudes

7.- Probar la API con Postman o Swagger (Paso a Paso).

Paso 1: Descargar e instalar Postman

- 1.1.- Visitar la página de Postman.
- 1.2.- Descarga la versión adecuada para el sistema operativo (Windows, macOS o Linux).
- 1.3.- Instalarlo siguiendo las instrucciones proporcionadas.

Paso 2: Iniciar la API

Antes de usar Postman, asegurarse de que la API REST ExportarDatosAExcelAPI está en ejecución:

- 2.1.- Abre el proyecto en Visual Studio.
- 2.2.- Ejecuta tu aplicación usando Ctrl + F5 o el botón Run.
- 2.3.- La consola debería indicar un mensaje *como Now listening on: http://localhost:5000* (el puerto puede variar).

Paso 3: Abre Postman

- 3.1.- Iniciar Postman.
- 3.2.- Una vez abierto, hacer clic en el botón New (Nuevo) y seleccionar HTTP Request.

Paso 4: Configurar la solicitud GET en Postman

Ahora configuraremos la solicitud para el endpoint que exporta los datos a Excel.

4.1: Configurar la URL de la solicitud

- a.- En el campo URL en la parte superior de Postman, escribe la dirección de la API.
Ejemplo: <http://localhost:5000/api/Exportar/{table}>
- b.- Reemplazar `{table}` con el nombre de la tabla en la Base Datos que se desea exportar.
Ejemplo: <http://localhost:5000/api/Exportar/producto>

4.2: Configura el método HTTP

- a.- Asegurarse de que el método esté configurado como GET (el método por defecto).

4.3: Agregar Headers (opcional)

- a.- Si la API requiere algún header adicional, puede ser configurado en la pestaña Headers.
- b.- Si no se necesita ningún header adicional, se puede omitir este paso.

Paso 5: Ejecutar la solicitud

- a.- Hacer clic en el botón Send (Enviar).
- b.- La API debería procesar la solicitud y devolver una respuesta.

Paso 6: Revisar la respuesta

- a.- Si todo funciona correctamente, en Postman se debería ver una respuesta JSON como esta:

```
json
{
  "Message": "Datos exportados exitosamente",
  "FilePath": "Import_Tech_Export.xlsx"
}
```

- b.- El archivo Excel (Import_Tech_Export.xlsx) debería haberse guardado en la carpeta raíz del proyecto (o en su defecto la ruta que fue especificada en el código).

Paso 7: Manejo de errores

Dependiendo de la respuesta, se podrá ver:

- a.- 200 OK → Exportación exitosa.
- b.- 404 Not Found → No se encontraron datos en la tabla.
- c.- 500 Internal Server Error → Ocurre un error en el servidor (posiblemente un error en la conexión a la base de datos).

Paso 8: Verificación del archivo generado

- a.- Navegar a la carpeta especificada en (filePath) para verificar que el archivo Excel ha sido creado.
- b.- Abrirlo para confirmar que los datos han sido exportados correctamente.

Consejos adicionales

- 1.- **Probar con diferentes tablas:** Se debes repetir el proceso cambiando el parámetro `{table}` en la URL para probar diferentes tablas en la base de datos.
- 2.- **Ver el log en la consola:** Revisar la consola donde la API está en ejecución para ver mensajes de depuración o posibles errores.
- 3.- **Guardar la solicitud en Postman:** Se puede guardar la solicitud para no tener que configurarla de nuevo en el futuro:
 - a.- Hacer clic en **Save** y seleccionar o crear una colección para almacenar la solicitud.Con estos pasos, se debe poder usar **Postman** de manera efectiva para probar la **API REST** y exportar datos a archivos Excel de la base de datos.

COMENCEMOS A CORRER EL PROGRAMA PARA PROBAR LA API CON POSTMAN

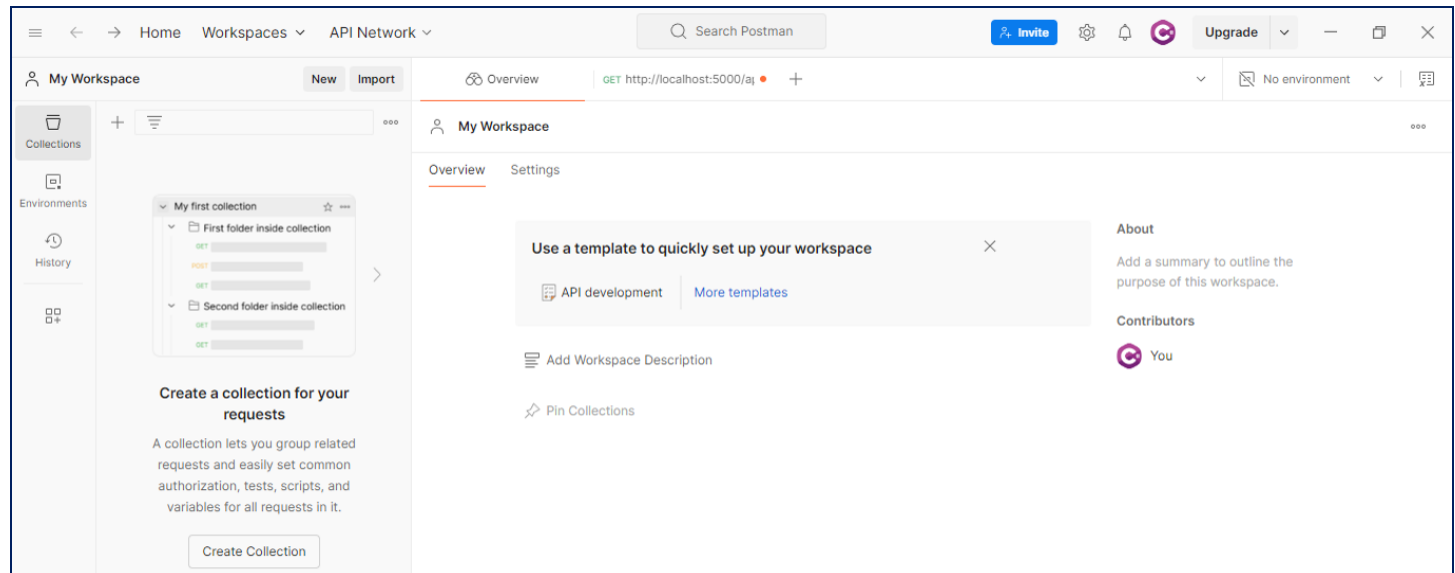
Paso-2:

C:\Users\epica\CSharp_Project\ExportarDatosAExcelAPI\bin\Debug\net8.0\ExportarDatosAExcelAPI.exe

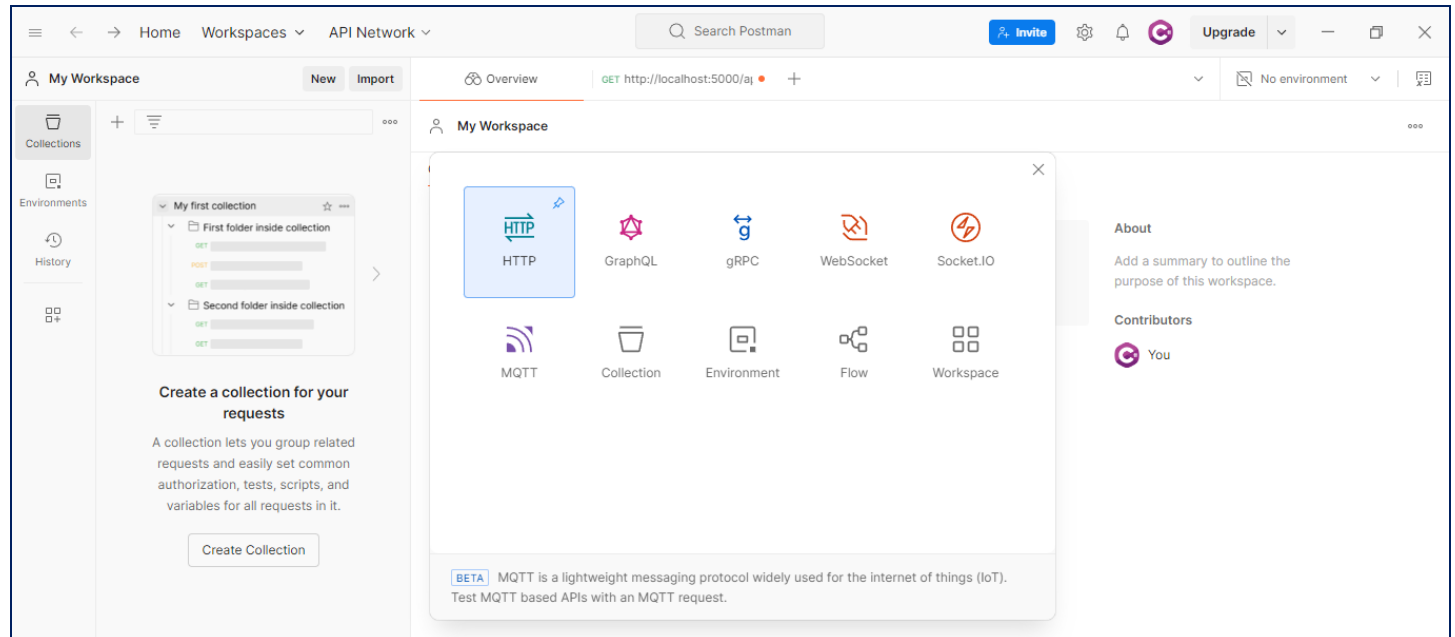
```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\epica\CSharp_Project\ExportarDatosAExcelAPI\bin\Debug\net8.0
```

Paso-3

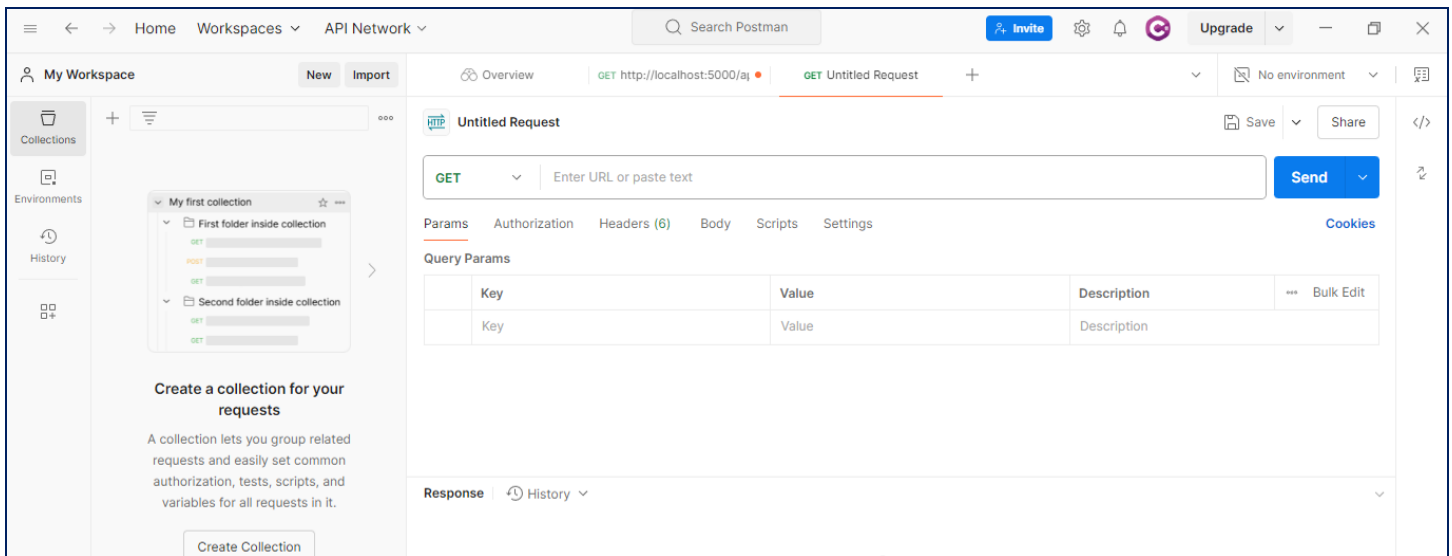
Paso 3.1



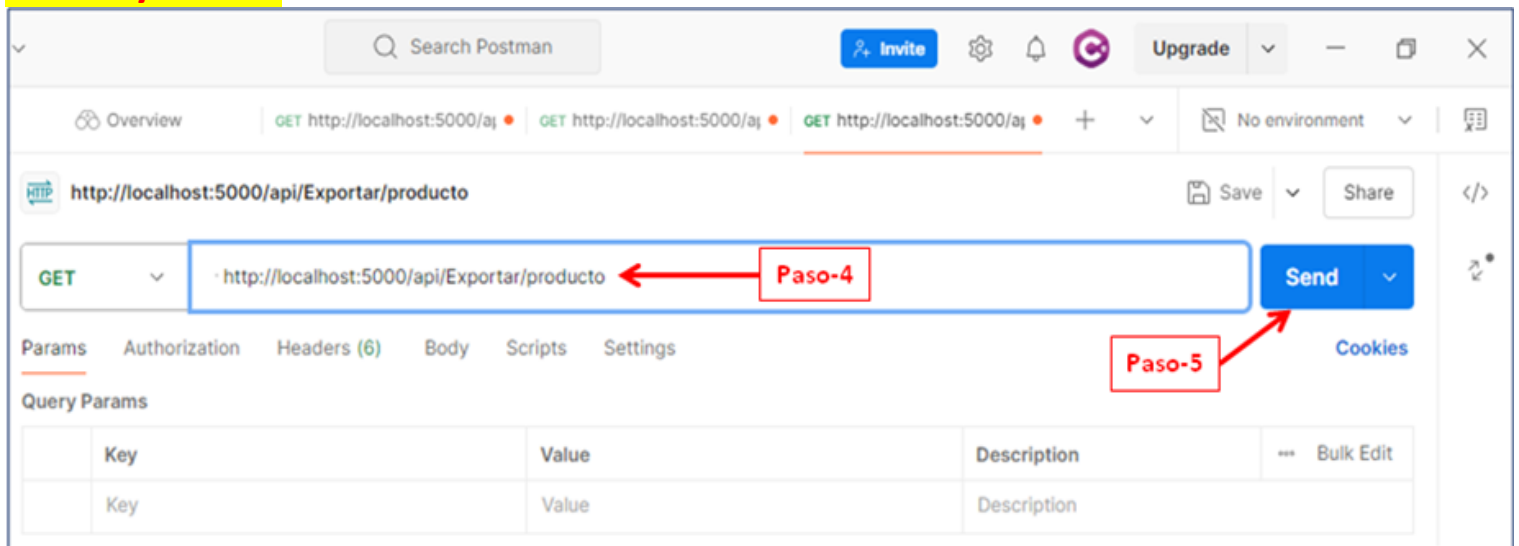
Paso 3.2



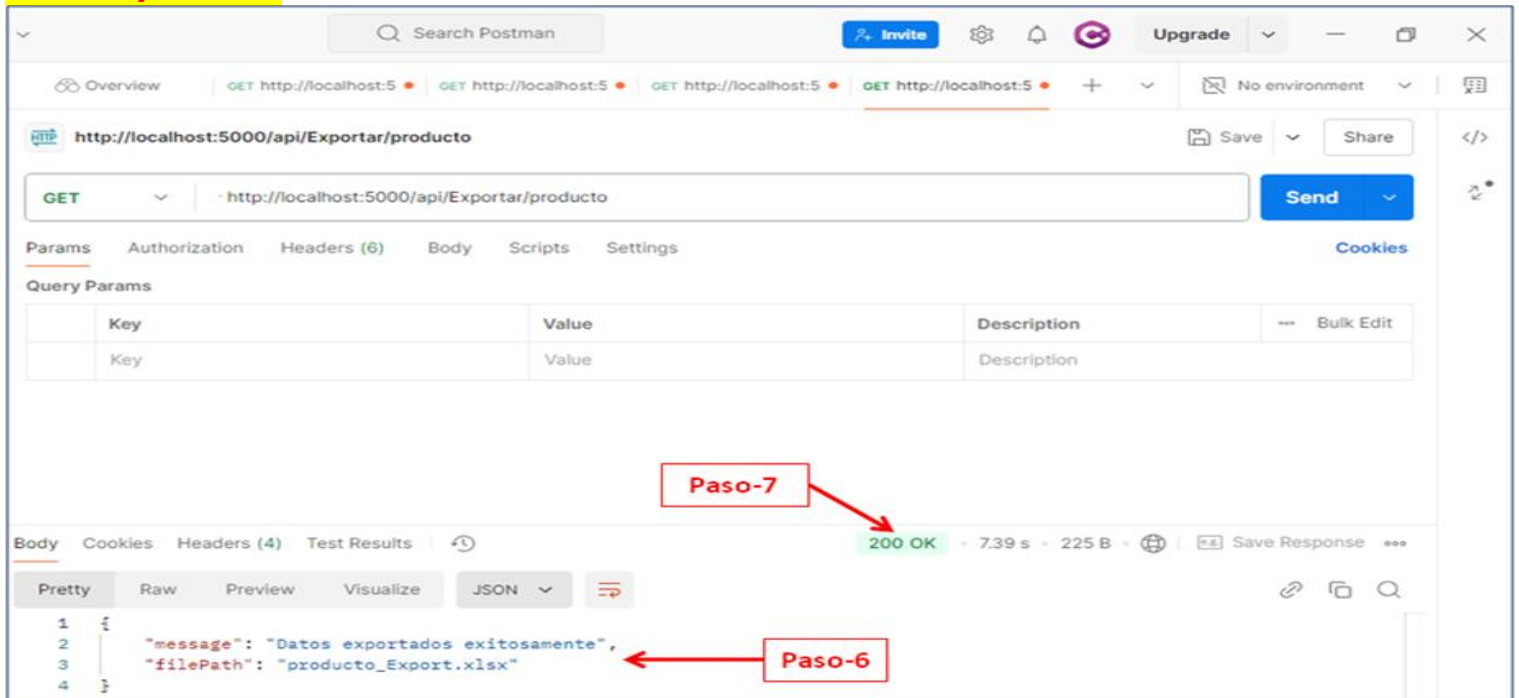
Ing. Rafael J. Rivas R.



Paso-4 y Paso-5



Paso-6 y Paso-7



Paso-8

a.- Navegar a la carpeta especificada en (filePath) para verificar que el archivo Excel ha sido creado.

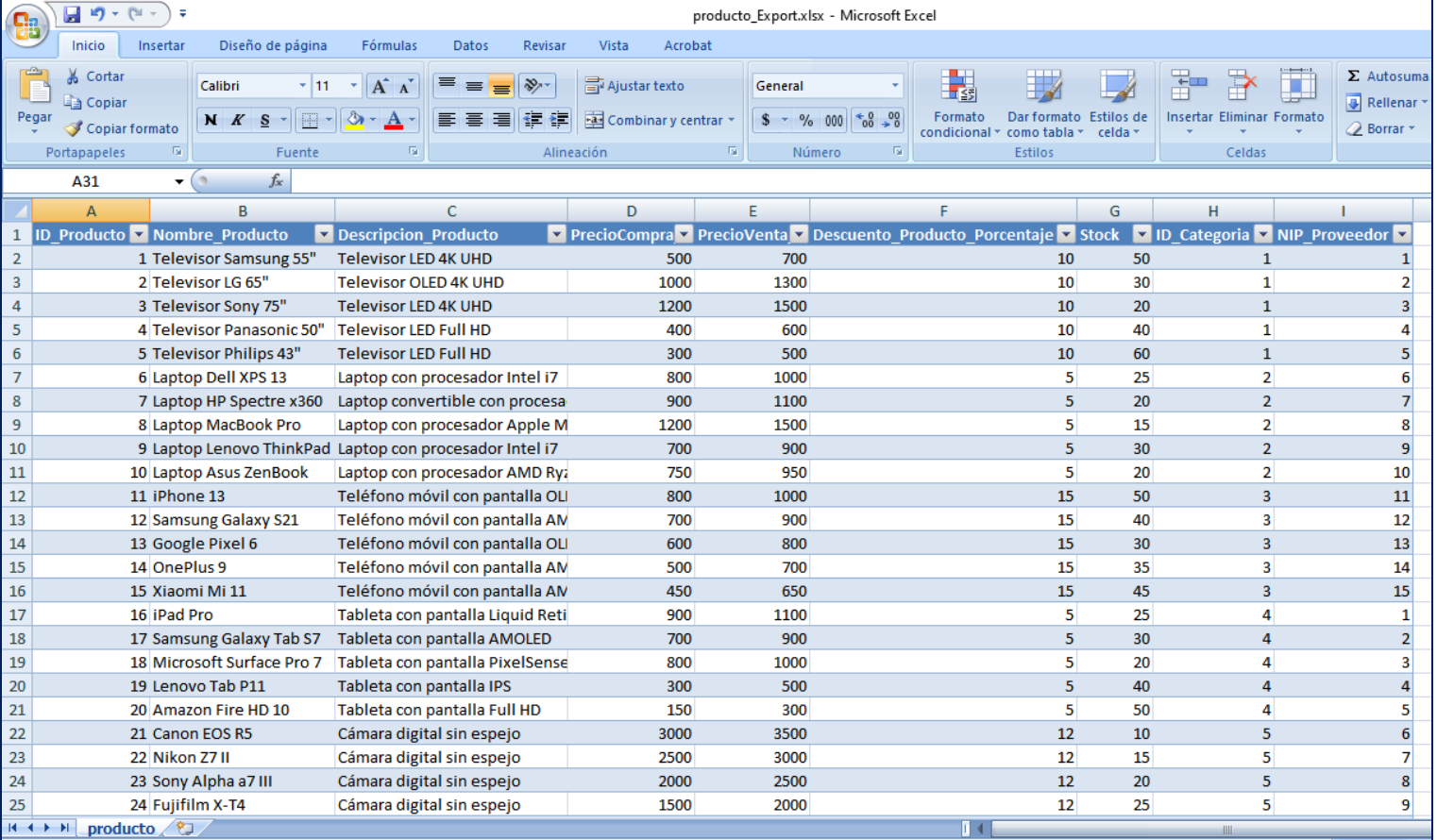
Cuando coompilamos el proyecto y, en el filePath no definimos una ruta para salvar el archivo a exportar, Visual Studio lo gurda en la dirección en donde corre el proyecto.

C:\Users\epica\CSharp_Project\ExportarDatosAExcelAPI\bin\Debug\net8.0

Color Azul: Dirección en donde el desarrollador ha creado el proyecto.

Color Rojo: Carpetas creadas por Visual Studio al compilar el proyecto.

b.- Abrir archivo .xlsx para confirmar que los datos han sido exportados correctamente.



ID_Producto	Nombre_Producto	Descripción_Producto	PrecioCompra	PrecioVenta	Descuento_Producto	Porcentaje	Stock	ID_Categoría	NIP_Proveedor	
1	1 Televisor Samsung 55"	Televisor LED 4K UHD	500	700	10	50	1	1		
2	2 Televisor LG 65"	Televisor OLED 4K UHD	1000	1300	10	30	1	2		
3	3 Televisor Sony 75"	Televisor LED 4K UHD	1200	1500	10	20	1	3		
4	4 Televisor Panasonic 50"	Televisor LED Full HD	400	600	10	40	1	4		
5	5 Televisor Philips 43"	Televisor LED Full HD	300	500	10	60	1	5		
6	6 Laptop Dell XPS 13	Laptop con procesador Intel i7	800	1000	5	25	2	6		
7	7 Laptop HP Spectre x360	Laptop convertible con procesa	900	1100	5	20	2	7		
8	8 Laptop MacBook Pro	Laptop con procesador Apple M	1200	1500	5	15	2	8		
9	9 Laptop Lenovo ThinkPad	Laptop con procesador Intel i7	700	900	5	30	2	9		
10	10 Laptop Asus ZenBook	Laptop con procesador AMD Ry	750	950	5	20	2	10		
11	11 iPhone 13	Teléfono móvil con pantalla OLI	800	1000	15	50	3	11		
12	12 Samsung Galaxy S21	Teléfono móvil con pantalla AM	700	900	15	40	3	12		
13	13 Google Pixel 6	Teléfono móvil con pantalla OLI	600	800	15	30	3	13		
14	14 OnePlus 9	Teléfono móvil con pantalla AM	500	700	15	35	3	14		
15	15 Xiaomi Mi 11	Teléfono móvil con pantalla AM	450	650	15	45	3	15		
16	16 iPad Pro	Tableta con pantalla Liquid Reti	900	1100	5	25	4	1		
17	17 Samsung Galaxy Tab S7	Tableta con pantalla AMOLED	700	900	5	30	4	2		
18	18 Microsoft Surface Pro 7	Tableta con pantalla PixelSense	800	1000	5	20	4	3		
19	19 Lenovo Tab P11	Tableta con pantalla IPS	300	500	5	40	4	4		
20	20 Amazon Fire HD 10	Tableta con pantalla Full HD	150	300	5	50	4	5		
21	21 Canon EOS R5	Cámara digital sin espejo	3000	3500	12	10	5	6		
22	22 Nikon Z7 II	Cámara digital sin espejo	2500	3000	12	15	5	7		
23	23 Sony Alpha a7 III	Cámara digital sin espejo	2000	2500	12	20	5	8		
24	24 Fujifilm X-T4	Cámara digital sin espejo	1500	2000	12	25	5	9		

Con esto se ha confirmado la exportación de los datos contenidos en la tabla **Producto** perteneciente a la Base de Datos **Import_Tech**