MANUAL DE USUARIO

EVALUACIÓN DE DESARROLLO EN .NET/SQL

Autor:
Guadalupe Meza Carrillo

2024

Índice

1 Introducción **1**

2 Base de Datos **1**
2.1 Introducción 1
2.2 Script 2

3 Aplicación **4**
3.1 BL 4
3.1.1 Script 4
3.2 DL 5
3.2.1 Script 6
3.3 ML 6
3.3.1 Tickets 6
3.3.2 Result 7
3.4 SWL 7
3.4.1 Script 7
3.5 Servicio de Windows 10
3.6 Configuración del Servicio de Windows 12

4 Evidencias **15**

5 Repositorio de GitHub **18**

1. Introducción

Este manual tiene la intención de mostrar el desarrollo de la Aplicación, cuya finalidad será consumir los archivos `*.fct` guardados en la carpeta de `Pendientes`. Los archivos, se limpiarán de los caracteres no deseados y se seccionarán en los campos correspondientes para intentar agregar el ticket a nuestra tabla `Tickets`, alojada en la base de datos mediante un Stored Procedure.

Posteriormente, a través de un Trigger, se alimentará la tabla de Resumen para ofrecer un histórico que contiene el Id de la Tienda, Id de la Registradora y el Total de Tickets con esta combinación.

Sea correcta o no la inserción, los archivos se moverán de la carpeta a `Procesados`. Si la inserción no fue correcta, su extensión cambiará a `*.fct_error`.

He expuesto la lógica de negocio a través de una REST API, utilizando ASP.NET Web API. Se probó en POSTMAN y posteriormente fue consumida desde un Servicio Windows.

2. Base de Datos

2.1. Introducción

En esta sección, se presenta el script de la Base de Datos utilizado. La base de datos utilizada se llama `evaluacion_gmeza`. Contiene dos tablas: `Tickets` y `Resumen`.

Se realizó un stored procedure para insertar un ticket en la tabla de tickets, y un trigger que se dispara después de realizar la inserción. El trigger obtiene el `IdTienda` y `IdRegistradora`. Primero verifica si la combinación de estos ids ya existe. Si existe, actualiza el campo `Ticket` incrementando su valor en 1. Si no existe, inserta la combinación `IdTienda` e `IdRegistradora` con el valor de `Ticket` igual a 1.

2.2. Script

```
USE [master]
GO
/****** Object:  Database [evaluacion_gmeza]******/
CREATE DATABASE [evaluacion_gmeza]
GO
USE [evaluacion_gmeza]
GO
/****** Object:  Table [dbo].[Resumen]******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Resumen](
[Id_Tienda] [varchar](15) NULL,
[Id_Registradora] [varchar](15) NULL,
[Tickets] [int] NULL
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Tickets]******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Tickets](
[Id_Tienda] [varchar](15) NULL,
[Id_Registradora] [varchar](15) NULL,
[FechaHora] [datetime] NULL,
[Ticket] [int] NOT NULL,
[Impuesto] [money] NULL,
[Total] [money] NULL,
[FechaHora_Creacion] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
[Ticket] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  StoredProcedure [dbo].[AddTicket] ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
CREATE PROCEDURE [dbo].[AddTicket]
( @Id_Tienda VARCHAR(15), @Id_Registradora VARCHAR(15), @FechaHora DATETIME,
@Ticket INT, @Impuesto MONEY, @Total MONEY)
AS
INSERT INTO Tickets(Id_Tienda, Id_Registradora, FechaHora,
Ticket, Impuesto, Total, FechaHora_Creacion)
VALUES(@Id_Tienda, @Id_Registradora, @FechaHora, @Ticket,
@Impuesto, @Total, GETDATE())
GO
/****** Object:  Trigger [dbo].[TicketsResumen] ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TRIGGER [dbo].[TicketsResumen]
ON [dbo].[Tickets]
AFTER INSERT
AS
BEGIN
IF EXISTS( SELECT 1 FROM Resumen
INNER JOIN inserted
ON Resumen.Id_Tienda = inserted.Id_Tienda
   AND Resumen.Id_Registradora = inserted.Id_Registradora)
BEGIN
UPDATE Resumen
SET Tickets = Tickets + 1
FROM Resumen
INNER JOIN inserted ON Resumen.Id_Tienda = inserted.Id_Tienda
   AND Resumen.Id_Registradora = inserted.Id_Registradora
END
ELSE
BEGIN
INSERT INTO Resumen(Id_Tienda, Id_Registradora, Tickets)
SELECT Id_Tienda, Id_Registradora, 1
FROM inserted
END
END
GO
ALTER TABLE [dbo].[Tickets] ENABLE TRIGGER [TicketsResumen]
GO
USE [master]
GO
ALTER DATABASE [evaluacion_gmeza] SET  READ_WRITE
GO
```

3. Aplicación

Esta aplicación se realizo en .NET Framework con el lenguaje de `C#`. La solución está dividida en 4 capas.

1. BL → Capa de la lógica de negocio
2. DL → Capa de datos
3. ML → Capa del modelo
4. SWL → Capa del Servicio Web

Además, se agregó al proyecto un Servicio de Windows llamado `ServicioTicket`.

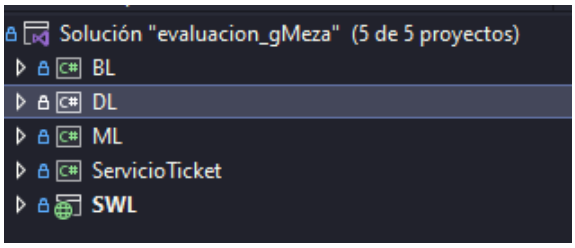


Figura 1: Solución del Proyecto

3.1. BL

En esta capa, se tiene la lógica de negocio. Se creó el método `Add` que recibe un objeto de tipo `Ticket`. Este método se encarga de realizar la conexión a la base de datos y enviar los parámetros para poder ejecutar el Stored Procedure `AddTicket`, insertando así el ticket en la base de datos. El método retorna un objeto de tipo `Result`, el cual se explicará en la capa del Modelo.

3.1.1. Script

```
namespace BL
{
    public class Tickets
    {
        public static ML.Result Add(ML.Tickets ticket)
        {
            ML.Result result = new ML.Result();
            try
            {
                using (SqlConnection con = new SqlConnection(DL.Conexion.Get()))
                {
                    con.Open();
                    SqlCommand cmd = new SqlCommand("AddTicket", con);
```

```
                    cmd.CommandType = CommandType.StoredProcedure;

                    cmd.Parameters.AddWithValue("Id_Tienda", ticket.Id_Tienda);
                    cmd.Parameters.AddWithValue("Id_Registradora", ticket.Id_Registradora
                    cmd.Parameters.AddWithValue("FechaHora", ticket.FechaHora);
                    cmd.Parameters.AddWithValue("Ticket", ticket.Ticket);
                    cmd.Parameters.AddWithValue("Impuesto", ticket.Impuesto);
                    cmd.Parameters.AddWithValue("Total", ticket.Total);

                    int rowsAffected = cmd.ExecuteNonQuery();
                    if (rowsAffected > 0)
                    {
                        result.Correct = true;
                    }
                    else
                    {
                        result.Correct = false;
                        result.Message = "Ocurrio un error al añadir el ticket";
                    }
                }
            }
            catch (Exception ex)
            {
                result.Correct = false;
                result.Ex = ex;
                result.Message = "Ocurrio un error al añadir";
            }
            return result;
        }
    }
}
```

3.2. DL

En esta capa, tenemos el método `Get`, que se encargará de retonar la cadena de conexión de la base de datos a la capa de negocio. Algo a tener en consideración es el tipo de autenticación que estamos utilizando para nuestra base de datos, ya que la sintaxis de la cadena de conexión cambia.
Si es por SQL se verá de esta manera:

```
string connectionString = "Server=myServerAddress;Database=myDataBase;
                           User Id=myUsername;Password=myPassword;";
```

Mientras que si es por Windows, se verá de la siguiente manera:

```
string conexion = "Data Source=myServer;
                   Initial Catalog=myDataBase; Integrated Security=True;";
```

3.2.1. Script

```
namespace DL
{
    public class Conexion
    {
        public static string Get()
        {
            string conexion = "Data Source=MEZACARRILLOGUA\\LUPIZ;
            Initial Catalog=evaluacion_gmeza; Integrated Security=True;";

            return conexion;
        }
    }
}
```

3.3. ML

En esta capa, se tienen los modelos que se usaron para la aplicación:

1. Tickets
2. Result

3.3.1. Tickets

Esta clase `Tickets`, sirve para crear instancias de un ticket con sus respectivas propiedades:

Script

```
namespace ML
{
    public class Tickets
    {
        public string Id_Tienda { get; set; }
        public string Id_Registradora { get; set; }
        public DateTime FechaHora { get; set; }
        public DateTime FechaHora_Creacion { get; set; }
        public int Ticket { get; set; }
        public decimal Impuesto { get; set; }
        public decimal Total { get; set; }
    }
}
```

3.3.2. Result

La clase `Result` se utiliza para encapsular la información de retorno de un método. Permite indicar si el proceso fue exitoso mediante el booleano `Correct`, proporcionar un mensaje con `Message`, capturar cualquier excepción con `Ex`, y puede contener un objeto específico con `Object`, o una lista de objetos con `Objects`.

Script

```
namespace ML
{
    public class Result
    {
        public bool Correct { get; set; }
        public string Message { get; set; }
        public Exception Ex { get; set; }
        public List<object> Objects { get; set; }
        public object Object { get; set; }
    }
}
```

3.4. SWL

En esta capa, se implementó el API REST, creando el controlador `Tickets`. Dentro de este controlador, se desarrolló un método de tipo `[HttpPost]` llamado `ProcesarTicket`. Este método se encarga de procesar los archivos de tickets en las carpetas `Pendientes` y `Procesados`. Primero, busca todos los archivos con la extensión `*.fct` en la carpeta `Pendientes`, los ordena del más antiguo al más nuevo y selecciona el más antiguo.
Si el archivo seleccionado no es nulo, se lee y se limpian los caracteres no deseados. Luego, se separan los datos para asignarlos a un modelo de ticket.
Si el ticket no es nulo, se envía el objeto del ticket a la capa de negocio para intentar insertarlo en la base de datos. Si la inserción del ticket es exitosa, se mueve el archivo a la carpeta `Procesados`. En caso contrario, o si el ticket es nulo, el archivo también se mueve a la carpeta `Procesados`, pero su extensión se cambia a `*.fct_error`.

3.4.1. Script

```
namespace SWL.Controllers
{
    public class TicketsController : ApiController
    {
        [HttpPost]
        [Route("api/tickets/procesar")]
        public IHttpActionResult ProcesarTicket()
        {
            ML.Tickets ticket = new ML.Tickets();
            string rutaPendientes = Path.Combine(AppDomain.CurrentDomain.
```

```
BaseDirectory, ConfigurationManager.AppSettings["rutaPendientes"]);
string rutaProcesados = Path.Combine(AppDomain.CurrentDomain.
BaseDirectory, ConfigurationManager.AppSettings["rutaProcesados"]);
string[] archivos = Directory.GetFiles(rutaPendientes, "*.fct");
string archivoSeleccionado = archivos.
OrderBy(f => new FileInfo(f).CreationTime).FirstOrDefault();

if (archivoSeleccionado != null)
{
    string nombreArchivo = Path.GetFileName(archivoSeleccionado);

    using (StreamReader reader = new StreamReader(archivoSeleccionado))
    {
        string ln = reader.ReadLine();
        if (ln != null)
        {
            string lnLimpia = ln.Replace(" ", string.Empty)
                            .Replace("-", string.Empty)
                            .Replace(">", string.Empty);
            string[] campos = lnLimpia.Split('|');

            ticket = new ML.Tickets
            {
                Id_Tienda = campos[0],
                Id_Registradora = campos[1],
                FechaHora = DateTime.ParseExact(campos[2] +
                            campos[3], "yyyyMMddHHmmss",
                            CultureInfo.InvariantCulture),
                Ticket = int.Parse(campos[4]),
                Impuesto = decimal.Parse(campos[5]),
                Total = decimal.Parse(campos[6]),
            };
        }
        else
        {
            Console.WriteLine("El documento esta vacío");
        }

    }
    if (ticket != null)
    {
        ML.Result result = BL.Tickets.Add(ticket);
        //Si leo el archivo nice y se agrega -> procesados
        if (result.Correct)
        {
```

```
                        string nuevaRuta = Path.Combine(rutaProcesados, nombreArchivo);
                        File.Move(archivoSeleccionado, nuevaRuta);
                        return Ok($"{nombreArchivo} insertado y procesado");
                    }
                    else
                    {
                        string nuevaExtension = Path.
                        ChangeExtension(archivoSeleccionado, ".fct_error");
                        string nuevaRutaError = Path.
                        Combine(rutaProcesados, Path.GetFileName(nuevaExtension));
                        File.Move(archivoSeleccionado, nuevaRutaError);
                        return Ok($"{Path.GetFileName(nuevaExtension)}
                        no fue insertado, pero sí procesado");
                    }
                }
                //Si lo leo mal y no se registra en ticket, el nombre del archivo
                se pasa y se cambia la extension a *.fct_error a procesados
                else
                {
                    string nuevaExtension = Path.
                    ChangeExtension(archivoSeleccionado, ".fct_error");
                    string nuevaRutaError = Path.
                    Combine(rutaProcesados, Path.GetFileName(nuevaExtension));
                    File.Move(archivoSeleccionado, nuevaRutaError);
                    return Ok($"{Path.GetFileName(nuevaExtension)}
                    procesado con Error, no se inserto");
                }
            }
            else
            {
                return Ok($"No se encontraron tickets en esta carpeta");
            }
        }
    }
}
```

En el `Web.config` se agregaron keys para definir las rutas de las carpetas:

```
<add key="rutaPendientes" value="..\Tickets\Pendientes" />
<add key="rutaProcesados" value="..\Tickets\Procesados" />
```

Esto permite una configuración flexible de las rutas y facilita la gestión del sistema.

3.5. Servicio de Windows

Por último, se creó un Servicio de Windows que consumirá la API REST cada 1 minuto para insertar los tickets. Posteriormente, en la carpeta `Debug` del servicio, se encontrará una carpeta `Log` que contendrá un archivo que mostrará el historial de todos los tickets que se intentaron insertar.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Net.Http;
using System.Net.NetworkInformation;
using System.Runtime.InteropServices.ComTypes;
using System.ServiceProcess;
using System.Text;
using System.Threading.Tasks;
using System.Timers;
using System.Windows.Forms;

namespace ServicioTicket
{
    public partial class ServicioTicket : ServiceBase
    {
        private System.Timers.Timer timer1;
        private EventLog eventLog1;
        private HttpClient httpClient;

        public ServicioTicket()
        {
            InitializeComponent();

            eventLog1 = new EventLog();
            if (!EventLog.SourceExists("ServicioTicket"))
            {
                EventLog.CreateEventSource("ServicioTicket", "Application");
            }
            eventLog1.Source = "ServicioTicket";
            eventLog1.Log = "Application";

            httpClient = new HttpClient();

            timer1 = new System.Timers.Timer();
```

```
        timer1.Interval = 60000;
        timer1.Elapsed += new ElapsedEventHandler(OnTimerElapsed);
    }

    protected override void OnStart(string[] args)
    {
        timer1.Start();
    }

    protected override void OnStop()
    {
        timer1.Stop();
    }

    private async void OnTimerElapsed(object sender, ElapsedEventArgs e)
    {
        try
        {
            HttpResponseMessage response = await
            httpClient.PostAsync("http://localhost:51427/api/tickets/procesar", null)

            if (response.IsSuccessStatusCode)
            {
                string data = await response.Content.ReadAsStringAsync();
                eventLog1.WriteEntry("Respuesta de la API: " + data);

                string rutaServicioTicket = AppDomain.CurrentDomain.BaseDirectory;

                string rutaCompleta = System.IO.Path.
                Combine(rutaServicioTicket,"Log","HistoricoTickets");

                string logLn = DateTime.Now.
                ToString("yyyy-MM-dd HH:mm:ss") + "| " + data + Environment.NewLine;

                using (StreamWriter sw = File.AppendText(rutaCompleta))
                {
                    sw.Write(logLn);
                }

            }
            else
            {
                eventLog1.WriteEntry("Error al llamar a la API.
                Código de estado: " + response.StatusCode);
            }
```

```
            }
            catch (Exception ex)
            {
                eventLog1.WriteEntry("Error al llamar a la API: " + ex.Message);
            }

        }
    }
}
```

3.6. Configuración del Servicio de Windows

1. Se abre Service1.cs y se agregá un Instalador.

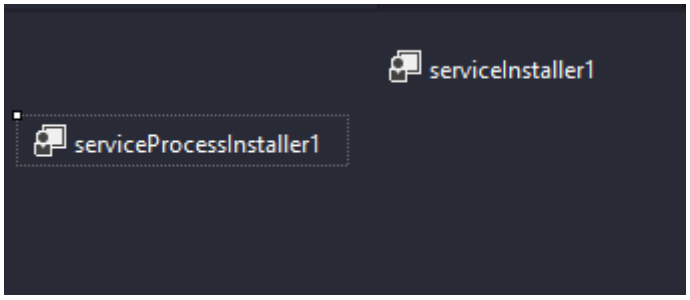


Figura 2: Agregando un Instalador

2. Se modifican las Propiedades de `serviceInstaller1`, para agregarle una descripción al servicio y un `DisplayName` de esta manera es que se verá en la aplicación de `Servicios` y que se ejecute de manera automatica.

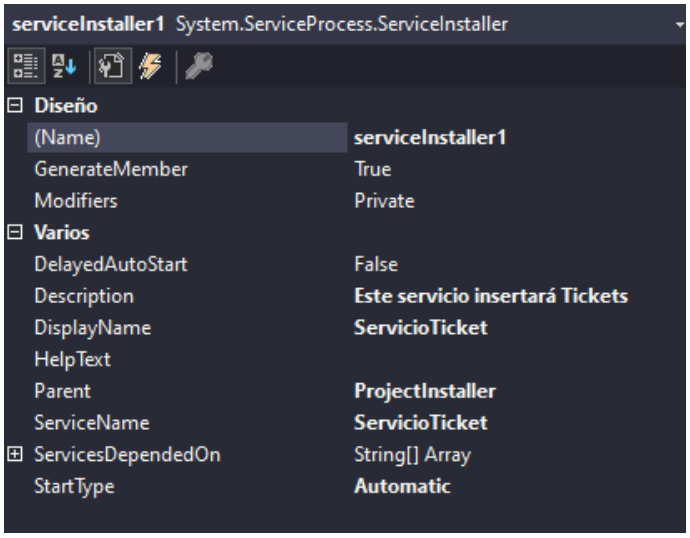


Figura 3: Propiedades de serviceInstaller1

3. Posteriormente, se modifica las Propiedad de `serviceProcessInstaller1`, la cual es `Account` a `LocalSystem`.

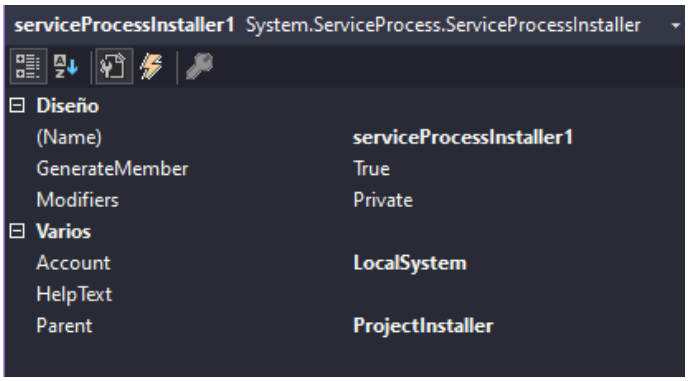


Figura 4: Propiedades de serviceProcessInstaller1

4. Se compila el servicio para obtener en la carpeta de `Debug` el `ServicioTicket.exe`
5. Se ejecuta la aplicación `Developer Command Prompt for VS` en modo `Administrador`
6. Se ejecuta el comando cd + la carpeta en donde se encuentra el `ServicioTicket.exe`, la cual esta en `Debug`

```
cd C:\Users\lupiz\source\repos\evaluacion_gMeza\ServicioTicket\bin\Debug
```

7. Posteriormente ejecutamos, Se ejecuta el comando que me permite instalar mi servicio en la computadora.

```
installutil ServicioTicket.exe
```

8. Ahora, se abre la aplicacion de `Servicios`, en la parte posterior selecciona `Estándar` y busca el nombre de tu servicio.

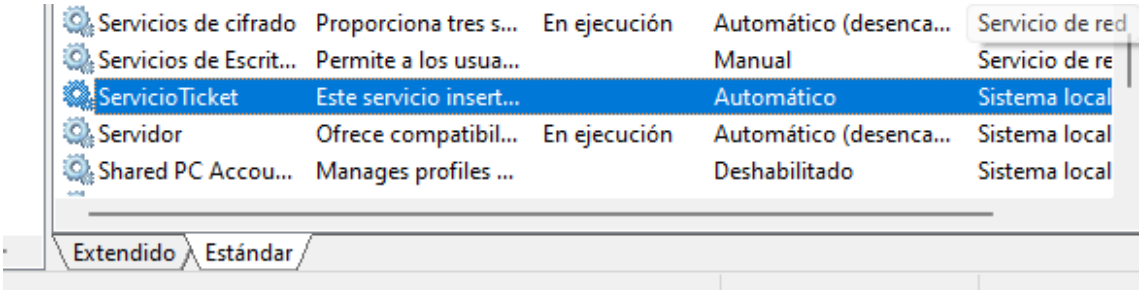


Figura 5: Buscando el Servicio creado en la Aplicación de Servicios

9. Click derecho al servicio y le damos en `Iniciar`.

10. A continuación, se abre la aplicación `Visor de Eventos`, posteriormente se selecciona `Registros de Windows` y `Aplicación`.

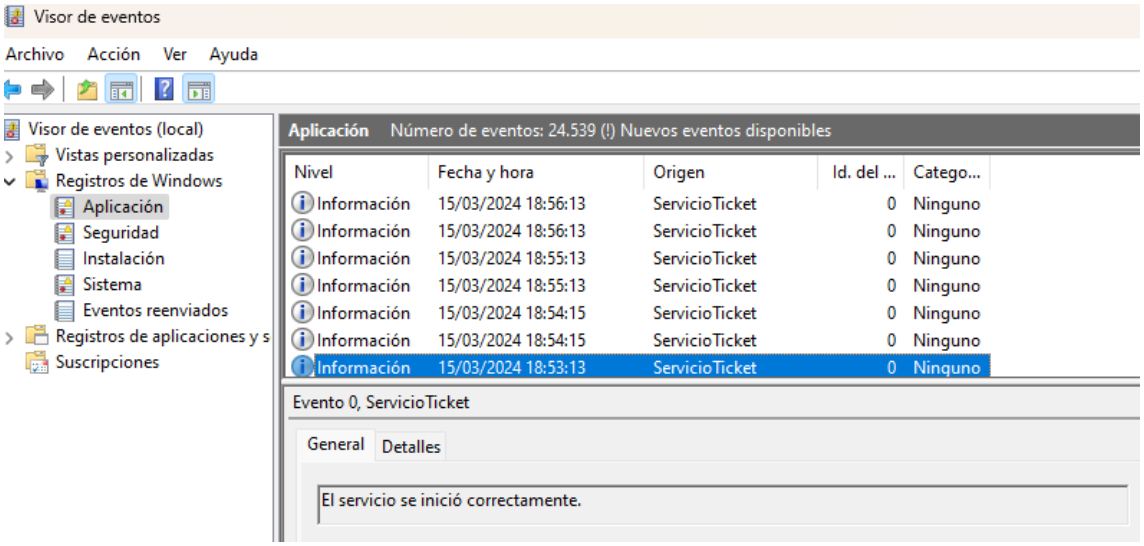


Figura 6: Visor de eventos, el servicio se inicio correctamente

11. En el `Visor de Eventos`, se puede observar todas las veces que se consumió la api con su respectivo resultado.

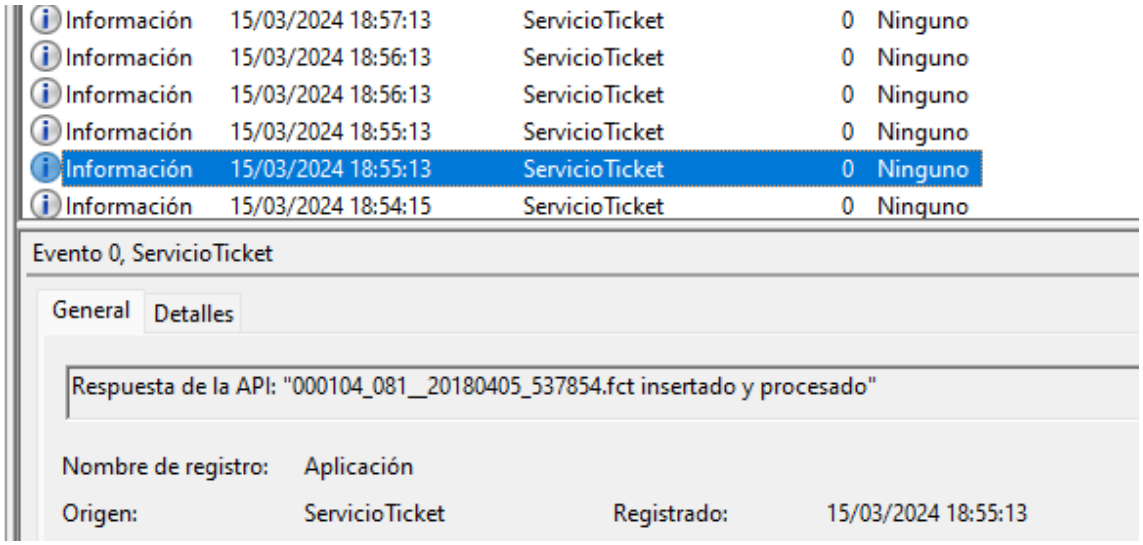


Figura 7: Ejecución del servicio cada 1 minuto, y como se muestra el resultado

4. Evidencias

1. Base de Datos

```
SELECT * FROM Tickets
SELECT * FROM Resumen
```

176 %

Resultados | Mensajes

	Id_Tienda	Id_Registradora	FechaHora	Ticket	Impuesto	Total	FechaHora_Creacion
24	000104	081	2018-04-05 17:43:17.000	537919	77200,00	5600,00	2024-03-15 22:12:18.497
25	000104	081	2018-04-05 17:44:01.000	537920	20700,00	1500,00	2024-03-15 22:13:18.510
26	000104	081	2018-04-05 17:45:32.000	537921	26200,00	1900,00	2024-03-15 22:14:18.503
27	000104	081	2018-04-05 17:45:50.000	537922	89700,00	6500,00	2024-03-15 22:15:18.503
28	000104	081	2018-04-05 17:52:41.000	537923	415200,00	30100,00	2024-03-15 22:16:18.500
29	000104	081	2018-04-05 17:53:45.000	537924	158600,00	11500,00	2024-03-15 22:17:18.507
30	000104	081	2018-04-05 17:56:33.000	537926	211000,00	15300,00	2024-03-15 22:18:18.500
31	000104	081	2018-04-05 17:59:08.000	537927	313100,00	22700,00	2024-03-15 22:19:18.513
32	000104	081	2018-04-05 18:00:36.000	537928	67600,00	4900,00	2024-03-15 22:20:18.507
33	000104	081	2018-04-05 18:00:59.000	537929	17900,00	1300,00	2024-03-15 22:21:18.513
34	000104	081	2018-04-05 18:07:08.000	537930	402800,00	29200,00	2024-03-15 22:22:18.513
35	000104	081	2018-04-05 18:08:23.000	537931	273100,00	19800,00	2024-03-15 22:23:18.530
36	000104	081	2018-04-05 18:13:55.000	537932	31700,00	2300,00	2024-03-15 22:24:18.510

	Id_Tienda	Id_Registradora	Tickets
1	000104	081	36

Figura 8: Evidencia de los Tickets registrados y de la Tabla de Resumen alimentada por el Trigger

2. Carpeta Pendientes

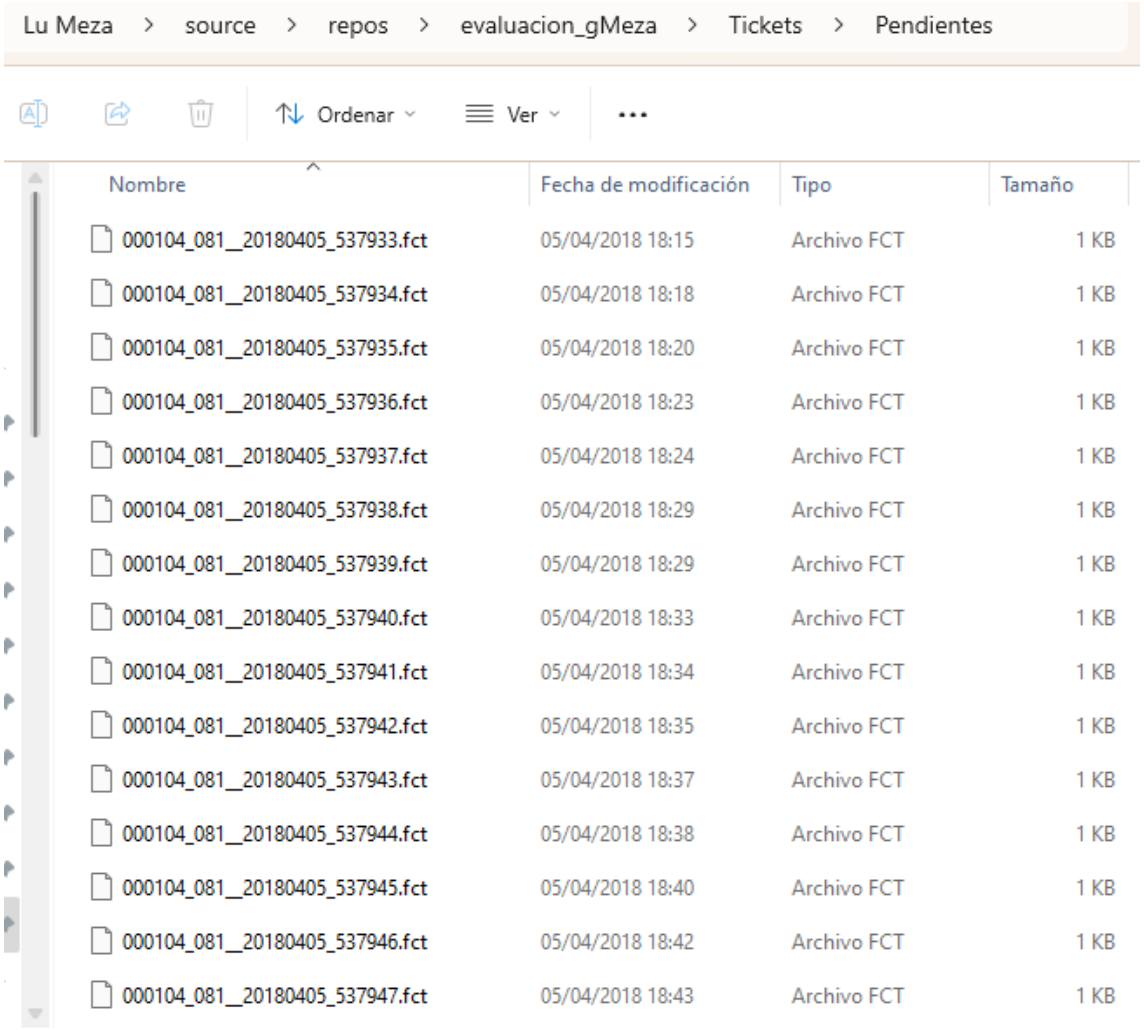
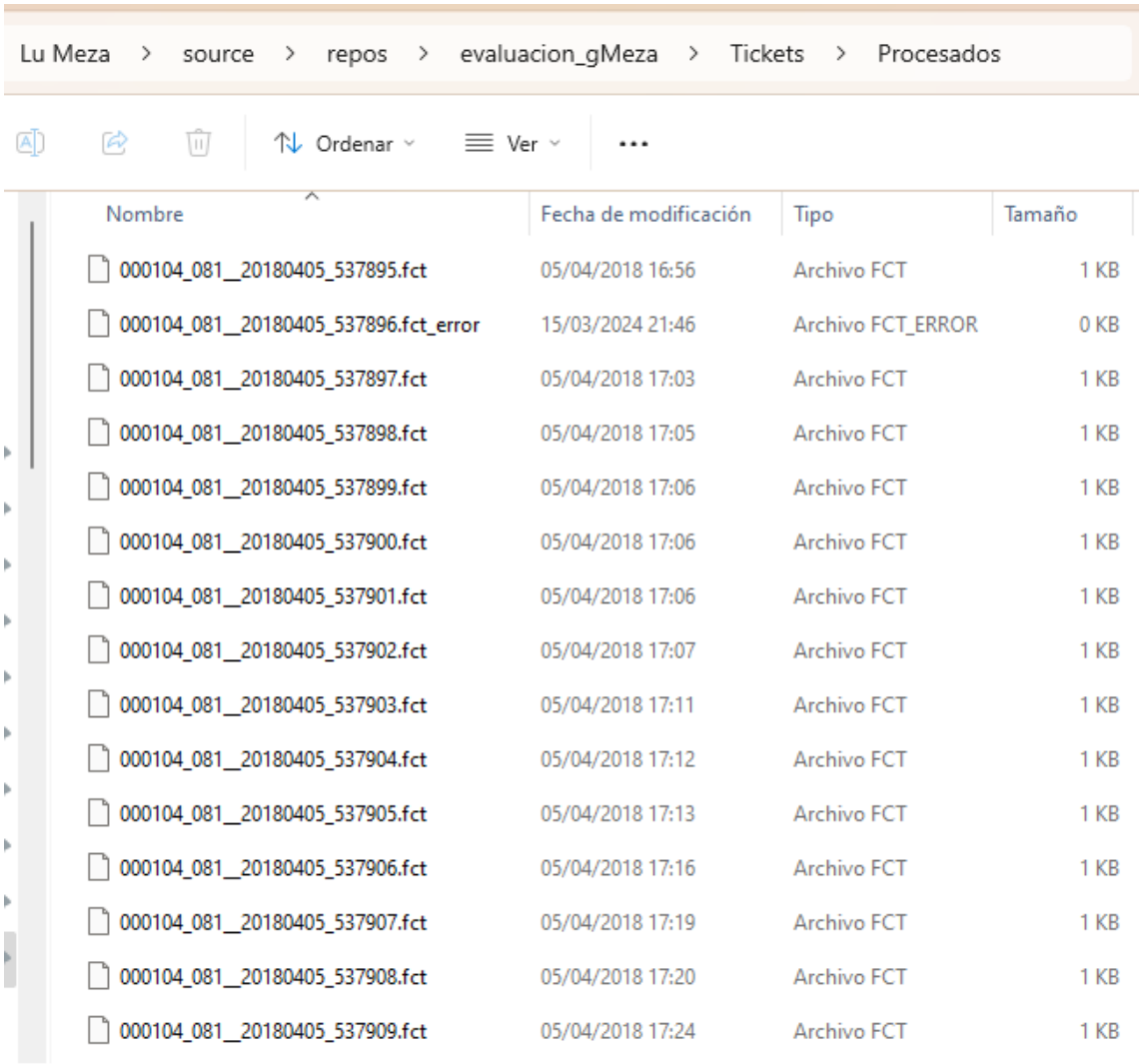


Figura 9: Evidencia de los Tickets que quedan por Procesar

3. Carpeta Procesados



Lu Meza > source > repos > evaluacion_gMeza > Tickets > Procesados

Ordenar Ver ...

Nombre	Fecha de modificación	Tipo	Tamaño
000104_081__20180405_537895.fct	05/04/2018 16:56	Archivo FCT	1 KB
000104_081__20180405_537896.fct_error	15/03/2024 21:46	Archivo FCT_ERROR	0 KB
000104_081__20180405_537897.fct	05/04/2018 17:03	Archivo FCT	1 KB
000104_081__20180405_537898.fct	05/04/2018 17:05	Archivo FCT	1 KB
000104_081__20180405_537899.fct	05/04/2018 17:06	Archivo FCT	1 KB
000104_081__20180405_537900.fct	05/04/2018 17:06	Archivo FCT	1 KB
000104_081__20180405_537901.fct	05/04/2018 17:06	Archivo FCT	1 KB
000104_081__20180405_537902.fct	05/04/2018 17:07	Archivo FCT	1 KB
000104_081__20180405_537903.fct	05/04/2018 17:11	Archivo FCT	1 KB
000104_081__20180405_537904.fct	05/04/2018 17:12	Archivo FCT	1 KB
000104_081__20180405_537905.fct	05/04/2018 17:13	Archivo FCT	1 KB
000104_081__20180405_537906.fct	05/04/2018 17:16	Archivo FCT	1 KB
000104_081__20180405_537907.fct	05/04/2018 17:19	Archivo FCT	1 KB
000104_081__20180405_537908.fct	05/04/2018 17:20	Archivo FCT	1 KB
000104_081__20180405_537909.fct	05/04/2018 17:24	Archivo FCT	1 KB

Figura 10: Evidencia de los Tickets Procesados, ya sea correcta o incorrectamente

4. Archivo Histórico

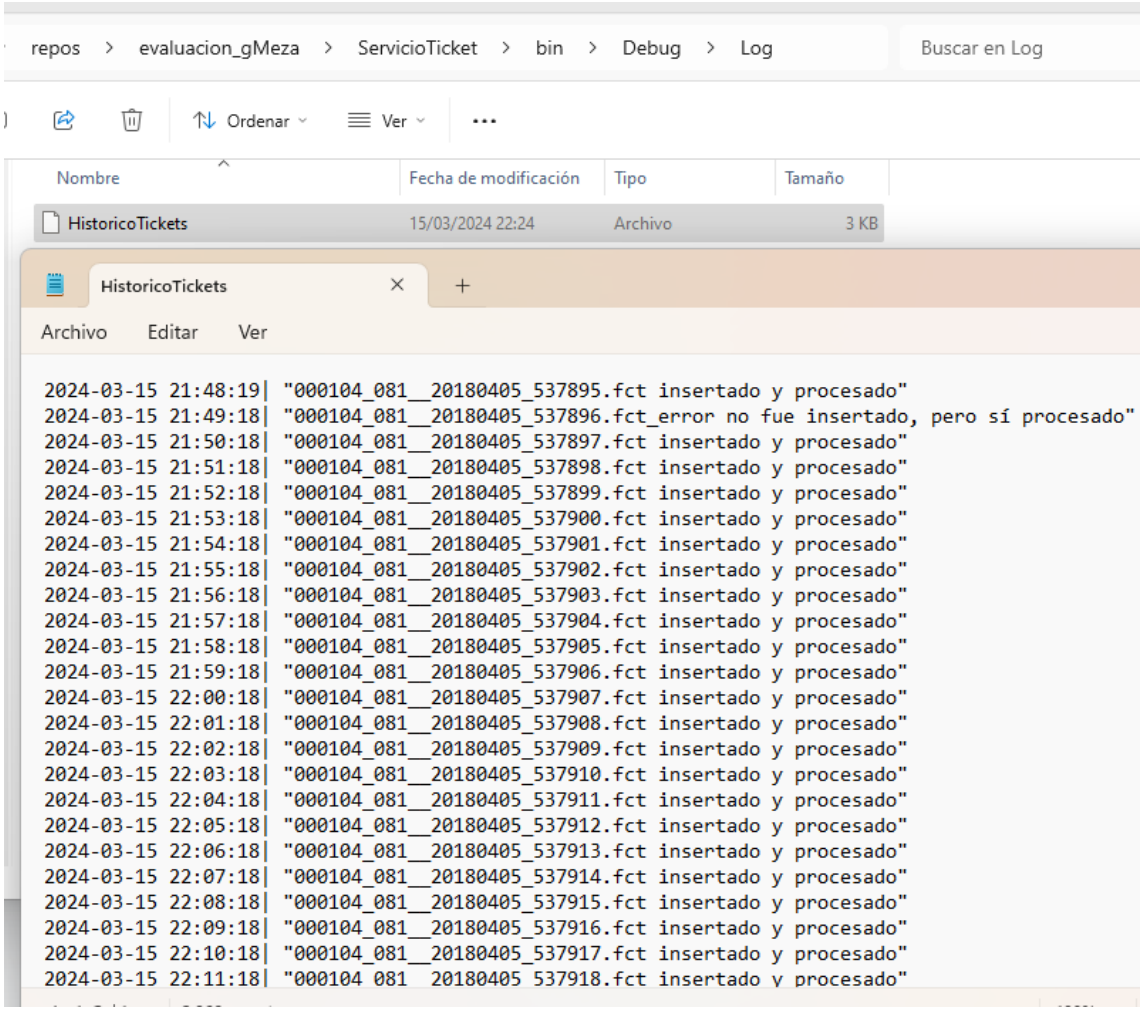


Figura 11: Evidencia del archivo generado con los Tickets Procesados por el servicio de Windows cada 1 minuto.

5. Repositorio de GitHub

El código fuente de este proyecto está disponible en GitHub, dándole click al siguiente texto: GMezaRepo.