



Tecnicatura Universitaria en Programación
ITG, Extensión Áulica (Goya) de la Universidad Tecnológica
Nacional (UTN) -Regional Resistencia-

Asignatura: Laboratorio 3

Nivel: 2do año – 1er cuatrimestre.

Coordinador de la Carrera

Ing. Germán Gaona

tup@frre.utn.edu.ar

www.frre.utn.edu.ar/tup

Docente:

Comisión única

Nombre y Apellido: José A. Fernandez

Alumnos:

- **Blas Ignacio Cabrera**
- **Marcelo Montenegro**
- **Matías Alberto Ganduya**
- **Sabrina Ayelén Arévalo**

Introducción

En el presente informe se desarrollaran y aplicaran todos los conceptos adquiridos a lo largo del cuatrimestre para poder realizar un programa el cual se encargará de gestionar el flujo de datos que tiene un programa de gestión de correos. Ya que en la era de la comunicación digital, el correo electrónico se ha convertido en una herramienta esencial para la interacción personal y profesional. La gestión eficiente de los correos electrónicos se vuelve fundamental para mantener la productividad y organizar la información de manera efectiva.

Para facilitar la gestión, se ha desarrollado un prototipo de un sistema de gestión de correos utilizando el framework .NET, aplicados gráficamente en una aplicación de escritorio. Este sistema, en el cual se aplica una arquitectura en capas, permite a los usuarios acceder a su bandeja de entrada, enviar y recibir correos electrónicos, así como filtrar y organizar los mensajes de manera efectiva mediante la implementación de métodos los cuales se encargan del paginado de los correos. A continuación, se describirán las principales características y funcionalidades aplicadas en el sistema.

Desarrollo

La aplicación de escritorio desarrollada con WinForms proporciona una interfaz intuitiva y amigable para que los usuarios puedan gestionar sus correos electrónicos de manera eficiente.

Como se menciona anteriormente, se utiliza una capa de presentación, implementada a través de los formularios de Windows Forms. Estos formularios proporcionan la interfaz gráfica de usuario (GUI) donde los usuarios pueden interactuar con la aplicación. Se diseñó de manera intuitiva y amigable, siguiendo las mejores prácticas de usabilidad, con el objetivo de facilitar la gestión de correos electrónicos de manera eficiente.

Se usa una capa de lógica de negocios que maneja las operaciones relacionadas con la gestión de correos electrónicos, como el envío, recepción, eliminación y organización de mensajes. Se diseñó para ser reutilizable y fácilmente mantenible.

La capa de acceso a datos interactúa con el almacenamiento de los correos electrónicos, realizando operaciones de lectura y escritura en la base de datos o sistema de archivos utilizado. Se enfoca en la abstracción de datos y la integración con diferentes proveedores de almacenamiento.

En base a los requisitos que pide la aplicación, se aplican distintas funcionalidades como lo son:

- **Formulario de ingreso/login:** El sistema proporciona un formulario de ingreso o login donde los usuarios pueden autenticarse para acceder a su cuenta de correo electrónico. Cabe aclarar que no se requiere registro de usuarios, por lo que se asume que los usuarios ya están creados previamente en la base de datos.
- **Creación de un correo electrónico:** El sistema permite a los usuarios crear un correo electrónico con las siguientes propiedades: asunto, contenido, remitente y destinatarios. El remitente es un único contacto, mientras que se pueden agregar múltiples contactos en el campo "Para" para enviar el correo a varios destinatarios.
- **Bandeja de Enviados:** Cuando un usuario escribe y envía un correo electrónico, este se guarda automáticamente en su bandeja de enviados. De esta manera, los usuarios pueden tener un registro de los correos que han enviado.
- **Bandeja de Entrada:** Cuando un usuario recibe un correo electrónico, este se muestra en su bandeja de entrada. Los usuarios pueden acceder a la bandeja de entrada para ver los correos electrónicos recibidos y realizar acciones adicionales.
- **Lectura de mensajes de correo:** Los usuarios pueden leer los mensajes de correo electrónico desde cualquier bandeja, ya sea la bandeja de entrada o la bandeja de enviados. Esto les permite ver el contenido completo de los mensajes y responder según sea necesario.

- **Filtrado de correos:** El sistema proporciona opciones de filtrado para ayudar a los usuarios a encontrar correos específicos en cada bandeja. Se pueden aplicar filtros según el texto del asunto o contenido del correo, así como el nombre o correo electrónico de un contacto (remitente o destinatario).

Conexiones

La base de datos desempeña un papel crucial en el sistema gestor de correos, ya que permite el almacenamiento y la organización de los correos electrónicos. Se utiliza un sistema de gestión de bases de datos relacional, como lo es en este caso el programa SQL Server, para asegurar una estructura coherente y un acceso rápido a los datos.

Se crean dos tablas, la primera se encarga de almacenar información sobre los usuarios para poder tener una ventana en la cual se realiza un proceso de login en el cual se verifica si el usuario y la contraseña coinciden con los datos cargados en la base. En cuanto a la segunda tabla esta contiene los datos sobre los correos, como lo son el asunto, el remitente, el destinatario, el contenido. Además, se implementan relaciones entre las tablas para establecer conexiones entre los correos y las columnas de los usuarios, esto se realiza para poder asignarle a cada usuario su propio gestor de correos, sin que los datos de otros usuarios se mezclen.

Para la conexión de la aplicación a la base de datos se utilizaron tanto ADO.NET nativo como Entity Framework para realizar conexiones a la base de datos y facilitar el acceso y manipulación de los datos en aplicaciones .NET.

Mediante la implementación de ADO.NET nativo, el cual utiliza proveedores de datos optimizados proporcionados por Microsoft que permiten el uso de múltiples fuentes de datos, se puede llegar a garantizar una mayor compatibilidad y rendimiento.

Por otro lado, Entity Framework es una tecnología muy utilizada en el desarrollo de aplicaciones .NET. Simplifica el acceso y manipulación de datos a través del mapeo objeto relacional (ORM). Esto significa que las bases de datos se pueden representar como clases, lo que facilita la interacción con los datos. Además, Entity Framework proporciona funcionalidades útiles, como la clase DbContext, que actúa como interfaz entre las entidades y los datos, permitiendo realizar operaciones específicas.

En la capa de acceso a datos, la cual es gestionada por la capa de negocios, se crearon dos clases, MailRepository y MailContext. En la primera mencionada se encuentran todos los métodos los cuales contienen las consultas y operaciones para el paginado y búsqueda de los correos. En MailContext se encuentra la conexión a la base de datos mediante la aplicación de Entity Framework.

En resumen, se utilizaron ADO.NET nativo y Entity Framework para conectarse a la base de datos y simplificar el acceso y manipulación de la información en aplicaciones .NET. Los métodos relacionados con la base de datos se implementaron en la capa de acceso a datos, específicamente en la clase MailRepository, que ofrece funcionalidades necesarias para interactuar con los datos.

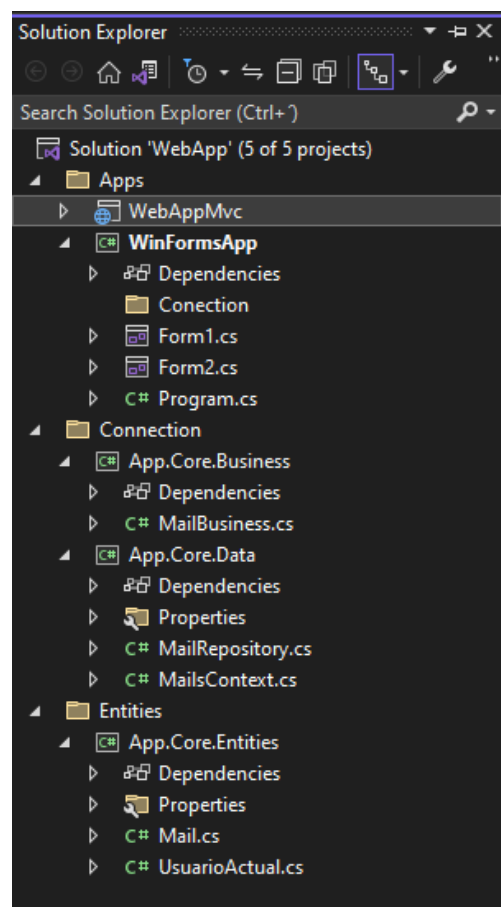
Funcionamiento del programa

Arquitectura

Antes de presentar las funcionalidades aplicadas en este trabajo hay que tener en cuenta de que se aplicó una arquitectura en capas que utiliza un patrón de diseño del tipo repository.

La arquitectura en capas se encarga de organizar el sistema en distintas capas o niveles lógicos en donde cada uno cuenta con una responsabilidad específica. Este enfoque proporciona una estructura modular y escalable al sistema, permitiendo la separación de preocupaciones y promoviendo la reutilización de código.

El patrón de diseño Repository, es una implementación comúnmente utilizada dentro de la arquitectura en capas. Este patrón de diseño se encarga de abstraer el acceso y manipulación de los datos, proporcionando una interfaz coherente y uniforme para interactuar con el almacenamiento de datos subyacente, ya sea una base de datos, un servicio web u otra fuente de datos.



En la aplicación, las capas se dividen en:

- **Capa de usuario (WinFormsApp):** en el contexto de un gestor de correos, la capa de interfaz de usuario se encarga de proporcionar la interfaz gráfica o visual a través de la cual los usuarios interactúan con el sistema. Esta capa permite a los usuarios realizar operaciones como leer, enviar, paginar y buscar los correos electrónicos de manera intuitiva y eficiente. En el caso de la aplicación, esta cuenta con dos winforms, el form2 es el encargado del inicio de sesión en donde se puede autenticar a los usuarios y gestionar sus permisos de

acceso al sistema solo los usuarios autorizados puedan acceder a sus cuentas de correo, en caso de no obtener esa autorización saldrá un mensaje de error.

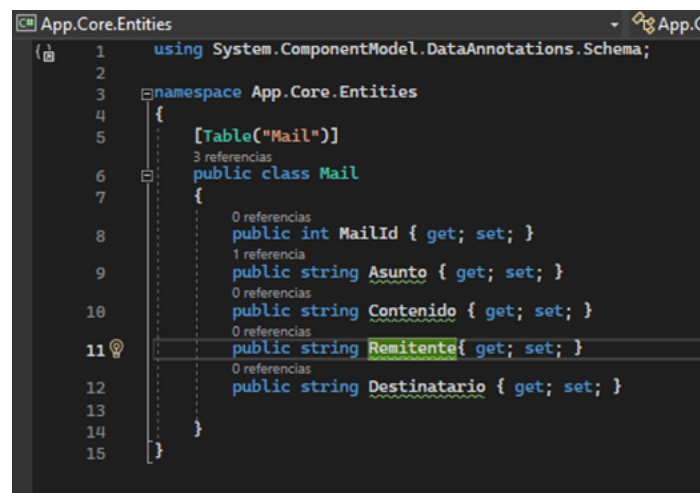
El segundo winform (form1) es la capa en donde el usuario es capaz de enviar los correos y realizar las operaciones que necesite.

- **Capa de negocios (App.Core.Business):** esta es una capa muy importante en la arquitectura Su objetivo principal es encapsular y representar las reglas de negocio y la lógica de la aplicación. La capa de negocios se encuentra entre la capa de presentación (interfaz de usuario) y la capa de acceso a datos. Actúa como una capa intermedia que coordina y procesa la lógica de negocio, interactuando con los datos provenientes de la capa de acceso a datos y proporcionando los resultados a la capa de presentación.
- **Capa de acceso a datos (App.Core.Data):** esta capa es la encargada de generar la conexión con la base de datos, en ella se encuentran todos los métodos necesarios para poder operar la aplicación. El gestor tiene dos clases, MailRepository y MailContext, en las cuales se encuentran los métodos y la conexión a la base utilizando el paquete de Entity Framework.
- **Capa de entidades (App.Core.Entities):** esta capa es la responsable del modelado y del mantenimiento de la integridad de los datos del negocio. Cabe aclarar, que esta capa es totalmente independiente a las demás y contiene sólo las propiedades que son específicas para el programa, en este caso, en la capa se debe incluir en todas las dependencias de la capa de acceso a datos y la capa de negocios, esta contiene dos clases, la clase Mail contiene las propiedades de asunto, contenido, remitente y destinatario; la segunda clase es Usuario actual, esta contiene la propiedad la cual está encargada de guardar el nombre de usuario con el cual se ingreso.

Capa de entidades

Clase Mail

Esta clase es parte de una arquitectura en capas, lo que significa que se utiliza para representar los datos de un correo electrónico en la capa de entidades. La capa de entidades es responsable de definir las estructuras de datos utilizadas en la aplicación y proporcionar métodos para acceder y manipular esos datos.



```
App.Core.Entities
1 using System.ComponentModel.DataAnnotations.Schema;
2
3 namespace App.Core.Entities
4 {
5     [Table("Mail")]
6     public class Mail
7     {
8         0 referencias
9         public int MailId { get; set; }
10        1 referencia
11        public string Asunto { get; set; }
12        0 referencias
13        public string Contenido { get; set; }
14        0 referencias
15        public string Remitente { get; set; }
16        0 referencias
17        public string Destinatario { get; set; }
18    }
19 }
```

En este código se define una clase Mail en donde se representa una entidad de correo electrónico en donde se especifican los requisitos necesarios que debe tener, contiene propiedades para almacenar su identificador, asunto, contenido, remitente y destinatario.

- **MailId:** es un entero que representa el identificador del correo electrónico. En la columna de la base de datos, este parámetro contiene una primary key
- **Asunto:** es una cadena de texto que almacena el asunto del correo electrónico.
- **Contenido:** es una cadena de texto que almacena el contenido del correo electrónico.
- **Remitente:** Es una cadena de texto que almacena la dirección de correo electrónico del remitente.
- **Destinatario:** Es una cadena de texto que almacena la dirección de correo electrónico del destinatario.

Por encima de la clase se encuentra la anotación "[Table("Mail")]", la cual indica que esta clase está asociada a una tabla llamada "Mail" en una base de datos. Esto implica que se espera que haya una correspondencia entre las propiedades de la clase y las columnas de la tabla en la base de datos.

Clase Usuario actual

Esta clase es la encargada de almacenar el nombre del usuario con el que se ingresó a la aplicación, va a ser necesaria para su posterior utilización en el formulario del gestor. Esta clase es la encargada de ayudar en la filtración de los correos teniendo en cuenta únicamente al usuario ingresado.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace App.Core.Data
8 {
9     4 references
10     public class UsuarioActual
11     {
12         4 references
13         public static string NombreUsuario { get; set; }
14     }
15 }
```

Capa de Acceso a datos

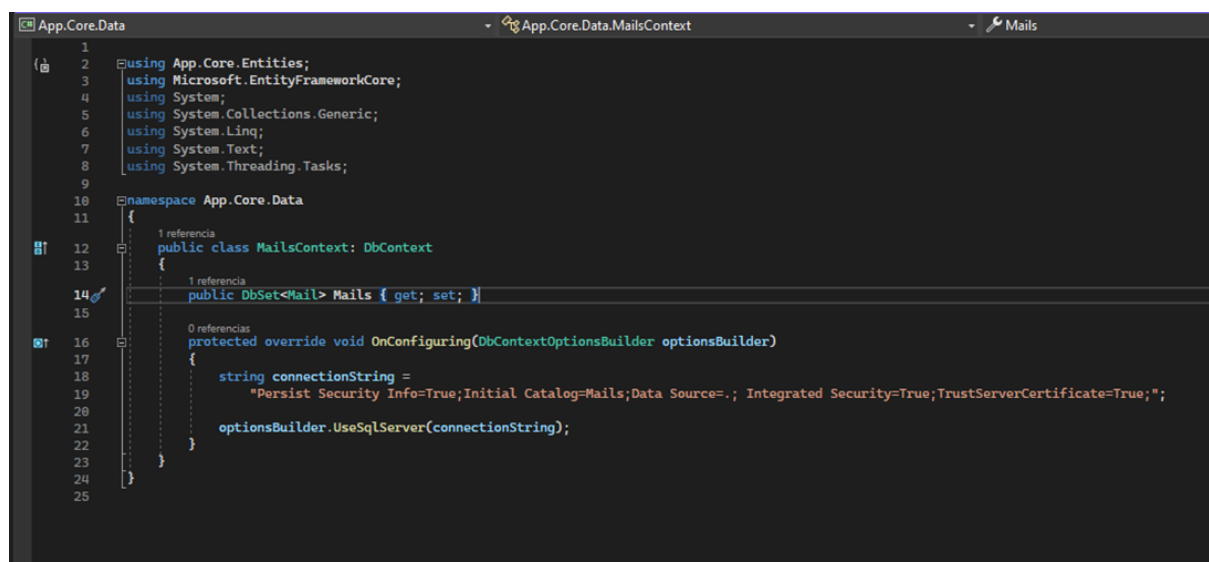
Clase MailContext

Esta clase hereda de la clase **DbContext** de la biblioteca Microsoft.EntityFrameworkCore y se utiliza para proporcionar un contexto de base de datos para la capa de datos de la aplicación.

La clase "MailsContext" contiene una propiedad **DbSet** llamada "Mails" que representa una colección de entidades de correo electrónico de la clase "Mail". DbSet es una clase genérica proporcionada por Entity Framework Core que permite acceder y manipular entidades en la base de datos.

La clase también tiene un método sobrescrito llamado **"OnConfiguring"** que se utiliza para configurar la conexión a la base de datos, el cual toma un objeto del tipo DbContextOptionsBuilder llamado "optionsBuilder" como parámetro. Este objeto se utiliza para configurar las opciones de DbContext. Es importante tener en cuenta que el método "OnConfiguring" se llama automáticamente por Entity Framework Core cuando se crea una instancia de la clase MailsContext() y se necesita configurar las opciones de DbContext.

En este caso, se establece una cadena de conexión específica en el método "optionsBuilder.UseSqlServer" para conectar con una base de datos SQL Server. La cadena de conexión contiene información como el nombre de la base de datos, la ubicación del servidor y las credenciales de autenticación.



```
1  using App.Core.Entities;
2  using Microsoft.EntityFrameworkCore;
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8
9
10 namespace App.Core.Data
11 {
12     public class MailsContext: DbContext
13     {
14         public DbSet<Mail> Mails { get; set; }
15
16         protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
17         {
18             string connectionString =
19                 "Persist Security Info=True;Initial Catalog=Mails;Data Source=.; Integrated Security=True;TrustServerCertificate=True;";
20
21             optionsBuilder.UseSqlServer(connectionString);
22         }
23     }
24 }
25
```

Clase MailRepository

La clase MailRepository es la encargada de la búsqueda de correos electrónicos de una base de datos mediante la implementación de un método llamado **"Search"** que utiliza Entity Framework y crea una instancia de Mail Context para luego realizar una consulta que identifique los correos electrónicos que contengan el texto a buscar en su Asunto.

El propósito de esta clase es proporcionar métodos para buscar correos electrónicos en una base de datos, utilizando un texto de búsqueda, paginación y una consulta LINQ. El método que utiliza tiene distintas operaciones, por ende nos encontramos con:

- El método principal de la clase es **Search**, el cual recibe tres parámetros:
 - o **textToSearch** es una cadena que representa el texto a buscar en el asunto de los correos electrónicos.
 - o **pageSize** es un entero que determina la cantidad de correos electrónicos por página.
 - o **pageIndex** es un entero que indica el índice de la página que se desea obtener.
 - o **destinatario** es una cadena de texto que contiene el nombre del destinatario.
 - o **remitente** es una cadena de texto que contiene el nombre del remitente.
- Se calcula el número de filas que se deben omitir utilizando la fórmula **((pageIndex - 1) * pageSize)**. Esto determina la cantidad de filas que se deben saltar en la consulta para obtener los resultados de la página actual.
- Se crea una instancia del contexto de base de datos **MailsContext**. Este contexto representa la capa de acceso a datos y se utiliza para interactuar con la base de datos.
- Se define una consulta LINQ utilizando la sintaxis de consulta. La consulta se realiza en la tabla **Mails** dentro del contexto y filtra los correos electrónicos cuyo asunto contiene el texto de búsqueda especificado (**m.Asunto.Contains(textToSearch)**) y que se cumplan el resto de las condiciones que se ven en la siguiente captura.
- El resultado de la consulta se almacena en la variable **query**.
- Se utiliza el método **Skip** en la variable **query** para omitir el número de filas calculado anteriormente, lo que permite la paginación de los resultados.
- Se utiliza el método **Take** en la variable **query** para seleccionar la cantidad de correos electrónicos especificada por **pageSize**, limitando así los resultados a la cantidad deseada por página.
- Se llama al método **ToList** en la variable **query** para ejecutar la consulta y obtener los resultados como una lista de objetos de la clase **Mail**.
- Finalmente, se devuelve la lista de correos electrónicos como resultado del método **Search**.

```

9 namespace App.Core.Data
10 {
11     3 references
12     public class MailRepository
13     {
14         1 reference
15         public MailRepository()
16         {
17         }
18         1 reference
19         public List<Mail> Search(string textToSearch, int pageSize, int pageIndex, string destinatario, string remitente)
20         {
21             var skipRows = ((pageIndex - 1) * pageSize);
22
23             using (var context = new MailsContext())
24             {
25                 var query = from m in context.Mails
26                             where (m.Asunto.Contains(textToSearch) &&
27                                   ((m.Destinatario == destinatario && m.Remitente == remitente) ||
28                                    (m.Destinatario == remitente && m.Remitente == destinatario)))
29                 select m;
30
31                 return query.Skip(skipRows)
32                             .Take(pageSize)
33                             .ToList();
34             }
35         }
36     }
37 }

```

Capa de negocio

Esta clase representa a la capa de negocio de la aplicación y se utiliza para realizar operaciones relacionadas con los correos electrónicos.

La clase MailBusiness tiene una propiedad pública llamada `_mailRepository` que representa un objeto de la clase MailRepository. Esta propiedad se utiliza para acceder a los métodos y datos proporcionados por la capa de datos.

En el constructor de la clase MailBusiness, se crea una instancia de la clase MailRepository y se asigna a la propiedad `_mailRepository`. Esto implica que la capa de negocio depende de la capa de datos, ya que utiliza un objeto de la capa de datos para realizar sus operaciones.

```

4 namespace App.Core.Business
5 {
6     4 references
7     public class MailBusiness
8     {
9         2 references
10        public MailRepository _mailRepository { get; set; }
11        2 references
12        public MailBusiness()
13        {
14            _mailRepository = new MailRepository();
15        }
16        2 references
17        public List<Mail> Search(string textToSearch,
18                                int pageSize,
19                                int pageIndex,
20                                string destinatario,
21                                string remitente) {
22
23            return _mailRepository.Search(textToSearch,
24                                        pageSize,
25                                        pageIndex,
26                                        destinatario,
27                                        remitente);
28        }
29    }
30 }

```

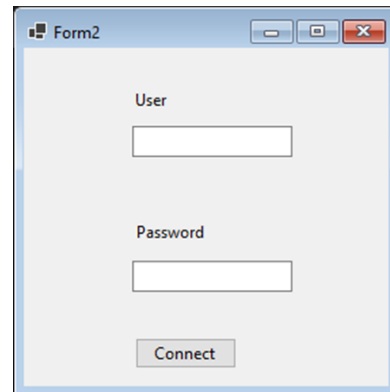
La clase MailBusiness tiene un método llamado Search que realiza una búsqueda de correos electrónicos. Este método toma cinco parámetros: `textToSearch` para especificar el texto a buscar, `pageSize` para especificar el tamaño de la página de resultados, `pageIndex` para especificar el índice de la página, `destinatario` y `remitente`.

Dentro del método Search, se llama al método Search de la clase _mailRepository para realizar la búsqueda de correos electrónicos. Se pasan los parámetros definidos al método Search del repositorio de correo electrónico. El resultado de la búsqueda se devuelve como una lista de objetos Mail.

Aplicación de escritorio

Login

Este formulario de inicio de sesión busca dar la oportunidad de que cada usuario tenga su propia casilla de correo. Para ello se desarrolla una serie de métodos para verificar la veracidad de los datos por medio del ingreso del nombre de usuario y contraseña.



Para empezar, se crea una propiedad privada la cual va a contener la conexión a la base de datos. Luego la clase Form2 hereda de la clase Form, lo que indica que es un formulario en la interfaz de usuario de la aplicación. El formulario contiene una instancia de la clase SqlConnection llamada "conexión" que se utiliza para establecer la conexión con la base de datos, esta conexión hecha en ADO.NET nativo proporciona ciertos métodos que van a ser utilizados para poder enviar y traer los datos correspondientes a las consultas realizadas.

En el constructor de la clase Form2, se inicia el formulario llamando al método "InitializeComponent()". Este método se utiliza para inicializar los componentes visuales del formulario.

```
3 referencias
public partial class Form2 : Form
{
    SqlConnection conexion = new SqlConnection("Persist Security Info=True;Initial Catalog=Mails;Data Source=.; Integrated Security=True;TrustServerCertificate=True;");
    1 referencia
    public Form2()
    {
        InitializeComponent();
    }
}
```

El método button1_Click se llama cuando se hace clic en el botón "button1" del formulario. Este método maneja la lógica de autenticación de usuarios.

Dentro del método button1_Click, se intenta abrir la conexión a la base de datos mediante el uso del método Open() en el objeto "conexión". Si ocurre algún error durante la apertura de la conexión, se muestra un mensaje de error.

A continuación, se construye una consulta SQL utilizando los valores ingresados en los campos de texto txtUser y "txtPassword" del formulario. La consulta selecciona todas las filas de la tabla UserTable donde el "UserName" y la "Password" coinciden con los valores ingresados.

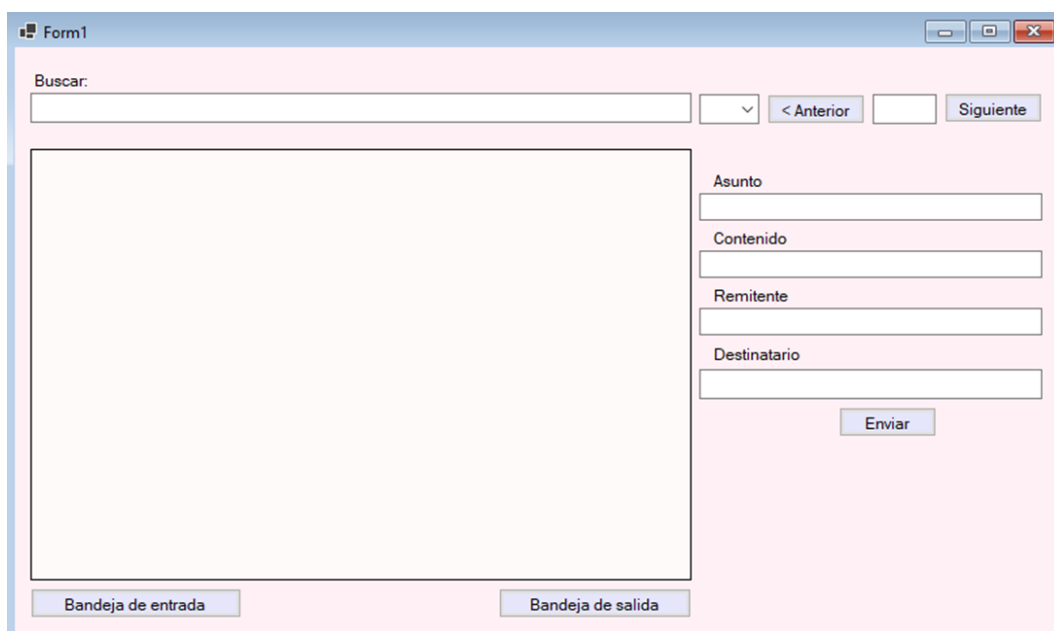
Se crea un objeto SqlCommand llamado comando con la consulta SQL y la conexión establecida anteriormente. Luego, se ejecuta la consulta utilizando el método ExecuteReader() del objeto comando y se obtiene un objeto SqlDataReader llamado reader para leer los resultados de la consulta.

Si el reader contiene filas, lo que significa que las credenciales ingresadas son correctas, se muestra un mensaje de bienvenida, se carga la variable "NombreUsuario" con el texto que contiene el usuario y se muestra el formulario Form1. Si el reader no contiene filas, se muestra un mensaje de error de credenciales.

Finalmente, se cierra la conexión a la base de datos mediante el método Close().

```
17 private void button1_Click(object sender, EventArgs e)
18 {
19     try
20     {
21         conexion.Open();
22     }
23     catch (Exception ex)
24     {
25         MessageBox.Show("Error");
26     }
27     string consulta = "SELECT * FROM UserTable WHERE UserName = @username AND UserPassword = @password";
28     SqlCommand comando = new SqlCommand(consulta, conexion);
29     comando.Parameters.AddWithValue("@username", txtUser.Text);
30     comando.Parameters.AddWithValue("@password", txtPassword.Text);
31     SqlDataReader reader = comando.ExecuteReader();
32
33     if (reader.HasRows == true)
34     {
35         MessageBox.Show("Bienvenido");
36
37         // Almacenar el nombre de usuario en una variable de sesión
38         UsuarioActual.NombreUsuario = txtUser.Text;
39
40         Form1 f1 = new Form1();
41         f1.Show();
42     }
43     else
44     {
45         MessageBox.Show("Error de credenciales");
46     }
47     conexion.Close();
48 }
```

Gestor de correos



Esta clase representa la interfaz de usuario de la aplicación y utiliza la clase "MailBusiness" de la capa de negocio para realizar operaciones relacionadas con los correos electrónicos.

La clase Form1 contiene varios campos privados, incluyendo una instancia de la clase MailBusiness llamada `_mailBusiness`, una variable `currentPageIndex` para realizar el seguimiento del índice de página actual y una variable booleana `_txtPageIndexBlock` para bloquear temporalmente la actualización del índice de página y una instancia de la clase `SqlConnection` llamada "conexion" que se utiliza para establecer la conexión con la base de datos, que al igual que en el formulario de login, esta conexión es realizada a través de la implementación de ADO.NET nativo.

```
namespace WinFormsApp
{
    4 referencias
    public partial class Form1 : Form
    {
        private MailBusiness _mailBusiness;
        private int currentPageIndex = 1;
        private bool _txtPageIndexBlock = false;
        SqlConnection conexion = new SqlConnection("Persist Security Info=True;Initial Catalog=Mails;Data Source=.; Integrated Security=True;TrustServerCertificate=True;");
        1 referencia
    }
}
```

El constructor se llama cuando se crea una instancia del formulario. Dentro del constructor, se realizan las siguientes acciones:

Se crea una nueva instancia de la clase MailBusiness y se asigna a la variable `_mailBusiness`. Esto implica que el formulario tiene una dependencia de la capa de negocio representada por la clase MailBusiness.

Se inicializa la interfaz de usuario del formulario llamando al método `InitializeComponent()`. Este método se genera automáticamente cuando se crea un formulario en Windows Forms y se encarga de configurar los controles y componentes en el formulario.

Se agregan los controladores de eventos para varios eventos del formulario y sus controles asociados. Los eventos incluyen:

- El evento **Load** del formulario, que se activa cuando se carga el formulario.
- El evento **SelectedIndexChanged** del control `cbTamañoPagina`, que se activa cuando se cambia la selección del tamaño de página.
- El evento **TextChanged** del control `txtSearch`, que se activa cuando se cambia el texto de búsqueda.
- El evento **Click** del botón `btnSiguiente`, que se activa cuando se hace clic en el botón Siguiente.
- El evento **Click** del botón `btnAnterior`, que se activa cuando se hace clic en el botón Anterior.

Estos controladores de eventos se utilizan para capturar las interacciones del usuario con el formulario y sus controles, y ejecutar el código correspondiente para responder a esas interacciones.

```
1 referencia
public Form1()
{
    _mailBusiness = new MailBusiness();

    InitializeComponent();
    this.Load += Form1_Load;
    cbTamañoPagina.SelectedIndexChanged += cbTamañoPagina_SelectedIndexChanged;
    txtSearch.TextChanged += txtSearch_TextChanged;
    btnSiguiente.Click += btnSiguiente_Click;
    btnAnterior.Click += btnAnterior_Click;
}
```

El método **Form1_Load()** se ejecuta cuando se carga el formulario y configura el valor predeterminado para el tamaño de página y muestra el índice de página actual en el control `txtPageIndex`.

```
1 referencia
private void Form1_Load(object sender, EventArgs e)
{
    cbTamañoPagina.SelectedIndex = cbTamañoPagina.FindStringExact("10");//por default al inicio carga 10 páginas
    txtPageIndex.Text = currentPageIndex.ToString();
}
```

El método **CargarDatos()** se utiliza para cargar los datos de los correos electrónicos en el control dgvTabla en un DataGridView utilizando el objeto "MailBusiness". Toma el tamaño de página actual y el índice de página como parámetros, también se asigna el valor del nombre de usuario actual a la variable destinatario. Esto indica quién es el destinatario de los correos electrónicos que se van a cargar. Luego se evalúa el valor de la variable destinatario en una serie de condiciones utilizando una estructura if-else if-else. Dependiendo del valor de destinatario, se asigna un valor correspondiente a la variable remitente.

Para finalizar, se realiza la búsqueda de correos electrónicos llamando al método "Search" del objeto "MailBusiness" con los parámetros adecuados.

```
private void CargarDatos(int pageIndex)
{
    var pageSize = int.Parse(cbTamanioPagina.SelectedItem.ToString());

    _txtPageIndexBlock = true;
    txtPageIndex.Text = pageIndex.ToString();
    _txtPageIndexBlock = false;

    string destinatario = UsuarioActual.NombreUsuario;

    string remitente = "";
    if (destinatario == "Pedro" || destinatario == "Juan")
    {
        remitente = "Maria";
    } else if (destinatario == "Maria" || destinatario == "Juan")
    {
        remitente = "Pedro";
    } else if (destinatario == "Maria" || destinatario == "Pedro")
    {
        remitente = "Juan";
    }

    var mails = _mailBusiness.Search(textToSearch: txtSearch.Text,
                                    pageSize: pageSize,
                                    pageIndex: pageIndex,
                                    destinatario: destinatario,
                                    remitente: remitente);

    dgvTabla.DataSource = mails;
}
```

Los métodos **cbTamanioPagina_SelectedIndexChanged()** y **txtSearch_TextChanged()** se ejecutan cuando hay cambios en la selección del tamaño de página y el texto de búsqueda, respectivamente. Estos métodos cuentan con dos eventos, SelectedIndexChanged se usa para capturar y responder a cambios en la selección de un control, mientras que TextChanged se utiliza para capturar y responder a cambios en el texto de un control. Ambos eventos son útiles para realizar acciones personalizadas y sincronizar la interfaz de usuario con la entrada del usuario en tiempo real.

Los dos métodos establecen el índice de página actual en 1 y llaman al método CargarDatos() para cargar los datos actualizados.

```
private void cbTamanoPagina_SelectedIndexChanged(object sender, EventArgs e)
{
    currentPageIndex = 1;
}

1 reference
private void txtSearch_TextChanged(object sender, EventArgs e)
{
    currentPageIndex = 1;
    CargarDatos(currentPageIndex);
}
```

Los métodos btnSiguiente_Click() y btnAnterior_Click() se ejecutan cuando se hace clic en los botones "Siguiente" y "Anterior", respectivamente. Estos métodos actualizan el índice de página actual en consecuencia y llaman al método CargarDatos() para cargar los datos de la página correspondiente.

```
private void btnSiguiente_Click(object sender, EventArgs e)
{
    currentPageIndex++;
    CargarDatos(currentPageIndex);
}

1 reference
private void btnAnterior_Click(object sender, EventArgs e)
{
    if (currentPageIndex == 1)
    {
        return;
    }
    currentPageIndex--;
    CargarDatos(currentPageIndex);
    txtPageIndex.Text = currentPageIndex.ToString();
}
```

El método txtPageIndex_TextChanged() se ejecuta cuando cambia el texto en el control txtPageIndex. Si la variable _txtPageIndexBlock es verdadera, el método simplemente sale temprano. De lo contrario, se intenta convertir el texto del control en un entero y se establece el índice de página actual en ese valor. Luego, se llama al método CargarDatos() para cargar los datos de la página correspondiente.


```

0 referencias
private void txtPageIndex_TextChanged(object sender, EventArgs e)
{
    if (_txtPageIndexBlock)
    {
        return;
    }
    currentPageIndex = int.Parse(txtPageIndex.Text);

    int index;

    if (int.TryParse(txtPageIndex.Text, out index))
    {
        currentPageIndex = index;
    }
    else
    {
        currentPageIndex = 1;
    }

    CargarDatos(currentPageIndex);
}

```

El método btnEnviar_Click_1 se ejecuta cuando se hace clic en el botón Enviar. Este método intenta abrir una conexión a una base de datos SQL Server utilizando el objeto SqlConnection. A continuación, crea una consulta SQL para insertar los datos del correo electrónico en la tabla Mail utilizando los valores de los controles de texto en el formulario. Luego, ejecuta la consulta y muestra un mensaje de éxito o error.

```

1 referencia
private void btnEnviar_Click_1(object sender, EventArgs e)
{
    try
    {
        conexion.Open();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error");
    }

    string query = "INSERT INTO Mail (Remitente, Asunto, Contenido, Destinatario) VALUES (@remitente, @asunto, @contenido, @destinatario)";
    // _mailBusiness.Envia(asunto: TxtAsunto.Text, contenido: TxtContenido.Text, remitente: TxtRemitente.Text, destinatario: TxtDestinatario.Text);
    SqlCommand comando = new SqlCommand(query, conexion);
    comando.Parameters.AddWithValue("@remitente", TxtRemitente.Text);
    comando.Parameters.AddWithValue("@asunto", TxtAsunto.Text);
    comando.Parameters.AddWithValue("@contenido", TxtContenido.Text);
    comando.Parameters.AddWithValue("@destinatario", TxtDestinatario.Text);
    comando.ExecuteNonQuery();
    MessageBox.Show("Insertado");
    conexion.Close();
}

```

Los métodos button1_Click y button2_Click, designados para la bandeja de entrada y bandeja de salida se ejecutan cuando se hace clic en los botones "Bandeja de entrada" y "Bandeja de salida", respectivamente. Ambos métodos realizan consultas SQL mediante una consulta la cual selecciona los datos de la tabla UserTable y Mail en donde los datos del remitente/destinatario coinciden con el usuario de la otra tabla, para seleccionar los datos de la tabla Mail y UserTable en función de ciertas condiciones y llenan el control dboTabla con los resultados de la consulta.

El método `btnBandejaDeEntrada` carga los datos de los correos electrónicos recibidos por el usuario actual en un control `DataGridView`, utilizando una consulta SQL para filtrar los correos electrónicos en función del destinatario. A diferencia del método anterior, `btnBandejaDeSalida` utiliza una consulta SQL para filtrar los correos electrónicos en función del remitente.

```
private void btnBandejaDeEntrada(object sender, EventArgs e)
{
    currentPageIndex = 1;
    CargarDatos(currentPageIndex);
    string query = $"SELECT * FROM Mail WHERE Destinatario = '{UsuarioActual.NombreUsuario}'";

    SqlCommand comando = new SqlCommand(query, conexion);
    SqlDataAdapter adaptador = new SqlDataAdapter();
    adaptador.SelectCommand = comando;
    DataTable tabla = new DataTable();
    adaptador.Fill(tabla);
    dgvTabla.DataSource = tabla;
}

1 reference
private void btnBandejaDeSalida(object sender, EventArgs e)
{
    currentPageIndex = 1;
    CargarDatos(currentPageIndex);
    string query = $"SELECT * FROM Mail WHERE Remitente = '{UsuarioActual.NombreUsuario}'";

    SqlCommand comando = new SqlCommand(query, conexion);
    SqlDataAdapter adaptador = new SqlDataAdapter();
    adaptador.SelectCommand = comando;
    DataTable tabla = new DataTable();
    adaptador.Fill(tabla);
    dgvTabla.DataSource = tabla;
}
```

Conclusión

En conclusión, el desarrollo del gestor de correos ha sido un proyecto que buscó implementar todos los métodos, arquitectura y conexión a base de datos necesarios para crear una aplicación eficiente y funcional. Durante el proceso, se ha adquirido un conocimiento profundo sobre los conceptos clave relacionados con la gestión de correos electrónicos, así como la aplicación práctica de estos conocimientos en un entorno de desarrollo.

El enfoque de implementar una arquitectura en capas ha demostrado ser altamente beneficioso, ya que nos ha permitido dividir las responsabilidades de manera clara y modular. La capa de presentación se encargó de proporcionar una interfaz amigable para los usuarios, mientras que la capa de lógica de negocio se ocupó de procesar y transformar los datos según las reglas del negocio. Por último, la capa de acceso a datos nos permitió interactuar con la base de datos de manera eficiente y segura.

A lo largo del proyecto, hemos aprendido mucho y hemos podido reforzar conceptos esenciales en el desarrollo de software. Desde la comprensión de los principios de la arquitectura en capas hasta la implementación de métodos de paginado de correos, cada paso nos ha llevado a un mayor entendimiento y dominio de las herramientas y técnicas utilizadas.

En resumen, la implementación del gestor de correos electrónicos ha sido un proceso enriquecedor que nos ha permitido aplicar una amplia gama de conocimientos y técnicas. A través de la implementación de una arquitectura en capas y la conexión a una base de datos, hemos logrado crear una aplicación funcional y eficiente. Este proyecto nos ha brindado una valiosa experiencia de aprendizaje y nos ha preparado para enfrentar nuevos desafíos en el desarrollo de software.

Link al repositorio de GitHub:

<https://github.com/sbrn-9/Gestor-De-Mails>

Fuentes:

<https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures>

<https://learn.microsoft.com/es-es/dotnet/framework/data/adonet/ado-net-overview>

<https://learn.microsoft.com/en-us/ef/core/>

<https://dev.to/ebarrioscode/patron-repositorio-repository-pattern-y-unidad-de-trabajo-unit-of-work-en-asp-net-core-webapi-3-0-5goj>