



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

**Asignatura: PROGRAMACIÓN 3**

**Nivel: 2do año – 1er cuatrimestre.**

**Coordinador de la Carrera**

**Ing. Germán Gaona**

**tup@frre.utn.edu.ar**

**[www.frre.utn.edu.ar/tup](http://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 este informe, se describe cómo se desarrolló una aplicación web MVC para crear un gestor de correo electrónico. El objetivo principal es permitir al usuario enviar y poder gestionar sus correos electrónicos de manera eficiente, conectado a una base de datos, donde se almacenarán la información de los emails. El gestor ofrece una interfaz intuitiva para redactar, enviar y poder ver los email en la bandeja de entrada y salida. El informe detalla los aspectos técnicos y desafíos.

## **Desarrollo**

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

Los usuarios pueden crear nuevos correos electrónicos, especificando el 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.

La aplicación también incluye la funcionalidad de bandeja de enviados, donde se guardan automáticamente los correos electrónicos enviados por los usuarios. Esto permite que los usuarios tengan un registro de los correos que han enviado.

Para facilitar la búsqueda de correos específicos, la aplicación proporciona opciones de filtrado. Los usuarios pueden aplicar filtros basados en el texto del asunto o contenido del correo, así como el nombre o correo electrónico de un contacto (remitente o destinatario).

En resumen, este trabajo práctico de programación ha dado como resultado una aplicación web MVC que permite a los usuarios gestionar sus correos electrónicos de manera eficiente. A lo largo del informe se detallarán los aspectos técnicos, desafíos enfrentados y posibles mejoras futuras para ampliar la funcionalidad y usabilidad de la aplicación.

## **Conexiones**

La conexión entre una aplicación web desarrollada bajo la arquitectura MVC y su base de datos es un componente fundamental para garantizar la persistencia y manipulación de los datos. En este caso específico de aplicación web MVC, se utilizan dos tecnologías para establecer esta conexión: ADO.NET nativo y Entity Framework.

ADO.NET nativo es una tecnología de acceso a datos provista por Microsoft, que permite interactuar con diversas fuentes de datos, incluyendo bases de datos relacionales. Proporciona un conjunto de clases y métodos para realizar operaciones de conexión, consulta, inserción, actualización y eliminación de datos.

En el contexto de la aplicación web MVC, ADO.NET nativo se utiliza para establecer una conexión directa con la base de datos relacional, en este caso, SQL Server. Esta tecnología ofrece un rendimiento óptimo y una alta compatibilidad con la plataforma .NET.

Por otro lado, se emplea Entity Framework, una tecnología ampliamente utilizada en el desarrollo de aplicaciones .NET que facilita el acceso y manipulación de datos mediante el mapeo objeto-relacional (ORM). Entity Framework permite representar la base de datos como clases en el código, lo que simplifica el trabajo con los datos al ofrecer una interfaz orientada a objetos. Además, ofrece funcionalidades avanzadas como el seguimiento de cambios, consultas LINQ (Language Integrated Query) y facilidades para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar). Entity Framework se encarga de traducir las operaciones realizadas en código a consultas SQL para interactuar con la base de datos subyacente.

En el caso de la aplicación web MVC en cuestión, la conexión a la base de datos se realiza mediante ADO.NET nativo y Entity Framework. ADO.NET nativo se encarga de establecer la conexión inicial con la base de datos utilizando proveedores de datos optimizados proporcionados por Microsoft. Esta conexión permite acceder y manipular los datos de manera eficiente. Por otro lado, Entity Framework actúa como una capa de abstracción sobre la base de datos, proporcionando una interfaz orientada a objetos para trabajar con los datos. Esto simplifica la interacción con la base de datos al permitir que las operaciones se realicen directamente sobre objetos de clases definidas en el código de la aplicación.

En resumen, la aplicación web MVC establece una conexión con su base de datos utilizando tanto ADO.NET nativo como Entity Framework. ADO.NET nativo se utiliza para la conexión inicial y la gestión de consultas SQL, mientras que Entity Framework facilita el acceso y manipulación de los datos mediante el mapeo objeto-relacional. Estas tecnologías en conjunto permiten una conexión eficiente y una manipulación sencilla de los datos en la aplicación web MVC.

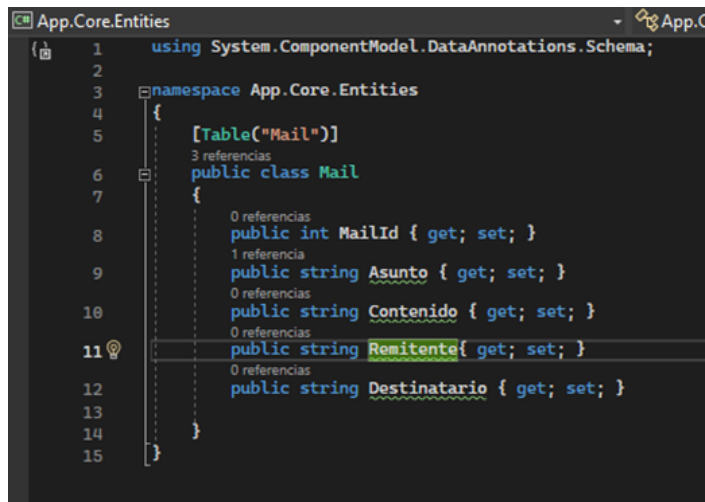
## **Arquitectura en capas**

### **Capa de entidades**

Es la que se encarga del modelado y del mantenimiento de la integridad de los datos del negocio.

## 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.

## Capa de Negocios

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.

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`.

```
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 }
```

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.

## Capa de acceso a datos

Es la encargada de generar conexiones con la base de datos. Posee la Clase **MailContext**, la clase **MailRepository** y la clase **UsuarioActual**

### Clase MailContext

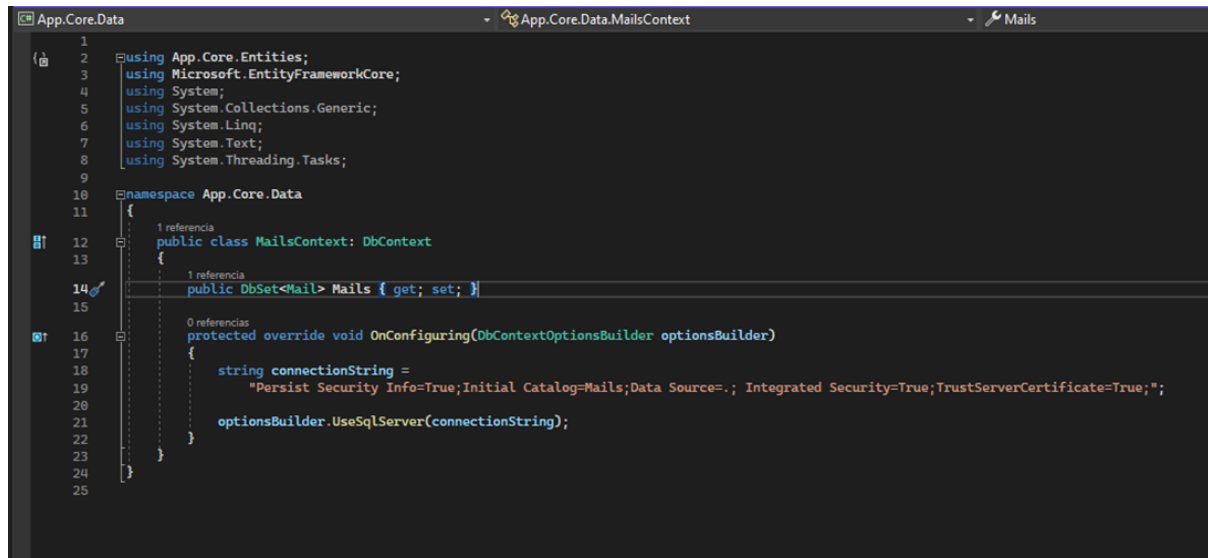
Esta clase hereda de la clase **DbContext** de la biblioteca Microsoft.EntityFrameworkCore y se utiliza para proporcionar un contexto de base de datos para la capa de datos de la aplicación.

La clase "MailContext" 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.



## 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.

- **destinatario** es una cadena de texto que contiene el nombre del destinatario.
- **remite** 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     }
19     1 reference
20     public List<Mail> Search(string textToSearch, int pageSize, int pageIndex, string destinatario, string remitente)
21     {
22         var skipRows = ((pageIndex - 1) * pageSize);
23
24         using (var context = new MailsContext())
25         {
26             var query = from m in context.Mails
27                         where (m.Asunto.Contains(textToSearch) &&
28                               ((m.Destinatario == destinatario && m.Remitente == remitente) ||
29                                (m.Destinatario == remitente && m.Remitente == destinatario)))
30                         select m;
31
32             return query.Skip(skipRows)
33                         .Take(pageSize)
34                         .ToList();
35         }
36     }
37 }

```

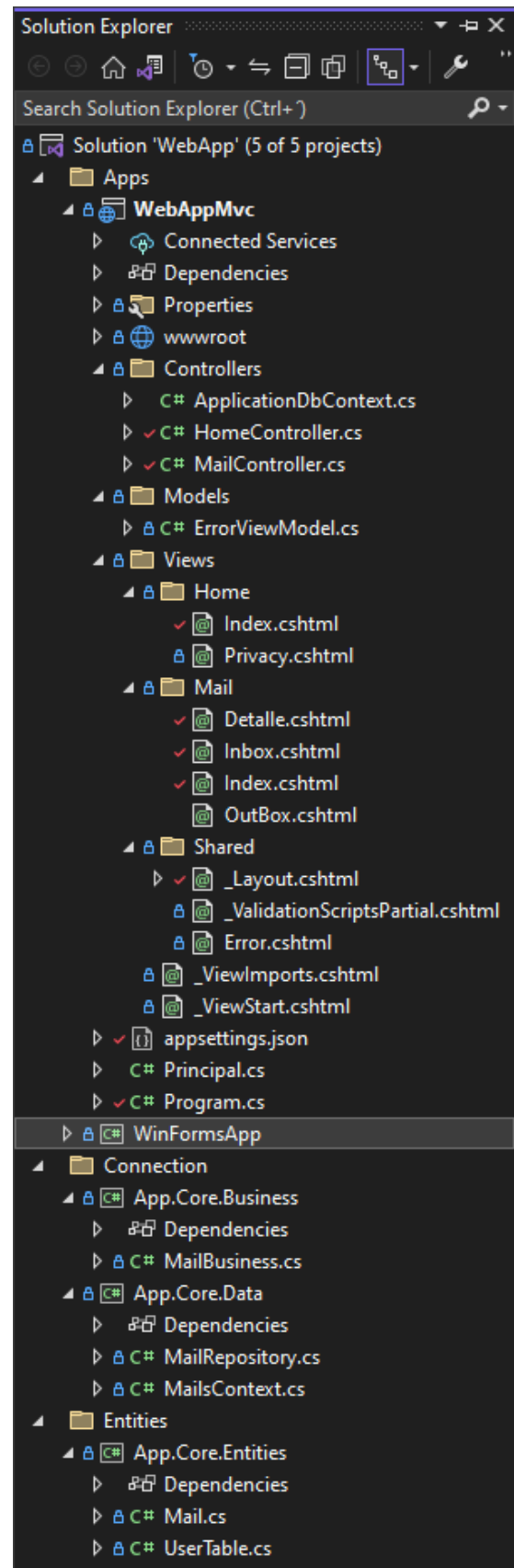
## Funcionamiento de la Aplicación

### MVC

El Modelo-Vista-Controlador, también conocido como MVC, es un patrón de diseño utilizado principalmente en el desarrollo de aplicaciones web, separa la lógica en los tres siguientes componentes principales:

- El Modelo representa los datos y la lógica de negocio de la aplicación. Es responsable de manejar la manipulación y el acceso a los datos, así como de realizar cálculos y procesamiento relacionados con la lógica de la aplicación. El modelo encapsula la lógica subyacente y no está directamente relacionado con la interfaz de usuario.
- La Vista es responsable de la presentación de los datos al usuario y de la interfaz de usuario en general. Puede ser una interfaz gráfica de usuario (GUI), una página web o cualquier otra forma de representación visual. La vista recibe datos del modelo y los muestra de manera adecuada para que el usuario pueda interactuar con ellos.
- El Controlador actúa como intermediario entre el modelo y la vista. Recibe las entradas del usuario desde la vista y las procesa, actualizando el modelo en consecuencia. También puede ser responsable de la navegación entre vistas y la gestión de eventos. El controlador toma las decisiones basadas en las acciones del usuario y actualiza el modelo y la vista en consecuencia.

Dentro de las distintas capas se encuentran las siguientes subcapas:





- La capa de presentación se implementa a través de formularios web, diseñados de manera intuitiva y amigable siguiendo las mejores prácticas de usabilidad. Estos formularios permiten a los usuarios interactuar con la aplicación, realizar acciones como el envío de correos electrónicos, ver la bandeja de entrada y salida, y realizar filtrados de mensajes.
- La capa de lógica de negocios se encarga de manejar las operaciones relacionadas con la gestión de correos electrónicos. Esto incluye funcionalidades como el envío de correos, recepción de mensajes, eliminación de correos no deseados y organización de la bandeja de entrada y salida. Se ha diseñado esta capa para ser reutilizable y fácilmente mantenible, de manera que se pueda agregar o modificar funcionalidades de manera eficiente.
- La capa de acceso a datos interactúa con el sistema de almacenamiento de correos electrónicos. Puede utilizar una base de datos u otro sistema de archivos para almacenar y recuperar la información de los correos. Se ha implementado teniendo en cuenta la abstracción de datos y la integración con diferentes proveedores de almacenamiento, lo que permite una mayor flexibilidad en la elección del sistema de almacenamiento.

### **Capa Controllers**

Es responsable de manejar las solicitudes HTTP y coordinar las interacciones entre el modelo y la vista. El controlador recibe la entrada del usuario y la procesa mediante el modelo y la vista

### **Capa Views**

Es responsable de mostrar los datos al usuario y recibir las entradas del usuario. La vista es la interfaz de usuario que muestra los datos al usuario y recibe las entradas del usuario. La vista se comunica con el controlador para recibir los datos que se mostrarán y enviar las entradas del usuario al controlador.

## **Aplicación web**

### **Login**

El inicio de sesión, o login, es una funcionalidad clave en la aplicación web MVC del gestor de correos electrónicos. El proceso de inicio de sesión permite a los usuarios autenticarse en la aplicación utilizando un nombre de usuario y una contraseña.

Cuando un usuario desea iniciar sesión, se le presenta una página de inicio de sesión que consta de un formulario donde debe ingresar su nombre de usuario y contraseña. Al hacer clic en el botón de inicio de sesión, la información ingresada por el usuario se envía al servidor para su validación.

Usuario  Password

El código que corresponde para la generación del login se encuentra en el archivo Index alojado en la carpeta Mail que la misma es una subcarpeta de Views. Se encarga de la vista y de manejar los datos que se ingresan en las casillas correspondientes a Usuario y Password.

```
1  @{\n2  ViewData["Title"] = "Login";\n3  }\n4  \n5  \n6  <div class="row">\n7  <form id="frm" action="@Url.Action("Enviar", "Home")" method="post">\n8  <div class="col-lg-6 col-lg-offset-3">\n9  Usuario\n10 <input type="text" name="user" /\n11 Password\n12 <input type="password" name="password" /\n13 \n14 <input type="submit" class="btn btn-success" value="Entrar" /\n15 </div>\n16 </form>\n17 </div>
```

```
<script>\n  $(document).ready(function () {\n    $("#frm").submit(function (e) {\n      e.preventDefault();\n\n      var url = "@Url.Content("~/Mail/Enter")";\n      var parametros = $(this).serialize();\n\n      $.post(url, parametros, function (data) {\n        if (data == "1") {\n          window.location.href = "@Url.Content("~/Mail")";\n        } else {\n          alert(data);\n        }\n      });\n    });\n  });\n</script>
```

Los parámetros ingresados por el usuario son mandados a una función que se encuentra en MailController, que se encuentra el método de acción que se encarga de procesar la solicitud de inicio de sesión. Este método verifica si el nombre de usuario y la contraseña proporcionados coinciden con los datos almacenados en la base de datos.

```

public IActionResult Enter(string user, string password)
{
    try
    {
        // Validar los datos de usuario y contraseña utilizando Entity Framework

        bool datosCoinciden = _context.Usuarios.Any(u => u.UserName == user && u.UserPassword == password);

        if (datosCoinciden)
        {
            return RedirectToAction("Index", "Home");
        }
        else
        {
            return Content("Usuario y/o contraseña incorrectos");
        }
    }
    catch (Exception ex)
    {
        return Content("Ocurrió un error: " + ex.Message);
    }
}

```

El controlador utiliza la capa de acceso a datos, la clase Business y las entidades correspondientes para realizar la verificación. La clase Business puede contener métodos que realizan consultas en la base de datos para buscar el usuario en función del nombre de usuario proporcionado y validar la contraseña. Si los datos coinciden, se considera que el inicio de sesión es exitoso y se permite el acceso al usuario a la aplicación.

En caso de que el inicio de sesión falle, por ejemplo, si el nombre de usuario o la contraseña son incorrectos, se pueden mostrar mensajes de error en la vista correspondiente para informar al usuario sobre el problema y permitirle volver a intentarlo.

Una vez que el usuario ha iniciado sesión correctamente, puede acceder a las funcionalidades del gestor de correos electrónicos, como la visualización de la bandeja de entrada, la redacción y el envío de correos, entre otras.

## Gestor de Correos

Esta vista proporciona una interfaz para que los usuarios interactúen y realicen varias acciones relacionadas con los correos electrónicos. Aquí pueden enviar nuevos correos, ver los mensajes recibidos y enviados, y también tienen la opción de buscar correos ingresando palabras que coincidan con el texto.

Asunto

Asunto

Contenido

Contenido

Remitente

Remitente

Destinatario

Destinatario

Enviar

Bandeja de entrada

Bandeja de Salida

Buscar

Buscar

Buscar

En resumen, la vista ofrece las siguientes funcionalidades:

- **Envío de correos:** Los usuarios pueden completar un formulario para redactar y enviar nuevos correos electrónicos.
- **Visualización de mensajes recibidos:** Los usuarios pueden acceder a su bandeja de entrada para ver los mensajes que han recibido de otros usuarios.
- **Visualización de mensajes enviados:** Los usuarios pueden acceder a su bandeja de correos enviados para ver los mensajes que han enviado a otros usuarios.
- **Búsqueda de correos electrónicos:** Se proporciona un campo de búsqueda donde los usuarios pueden ingresar palabras clave. Al realizar la búsqueda, se mostrarán los correos electrónicos que contengan esas palabras en su asunto, destinatario, remitente o contenido.

En general, esta vista brinda a los usuarios una manera conveniente de administrar y buscar sus correos electrónicos de manera eficiente y efectiva.

El código que genera esta vista se encuentra en la subcarpeta Home, en el archivo Index.

```
@{
    ViewData["Title"] = "Home Page";
}

<div class="correo">
    <div class="col-lg-6 col-lg-offset-3">
        <form id="frmEnviar">
            <div class="form-group">
                <label>Asunto</label>
                <input type="text" class="form-control" placeholder="Asunto" name="asunto" />
            </div>

            <div class="form-group">
                <label>Contenido</label>
                <textarea class="form-control" name="contenido" placeholder="Contenido"></textarea>
            </div>

            <div class="form-group">
                <label>Remitente</label>
                <input type="text" class="form-control" placeholder="Remitente" name="remitente" />
            </div>

            <div>
                <div class="form-group">
                    <label>Destinatario</label>
                    <input type="text" class="form-control" placeholder="Destinatario" name="destinatario" />
                </div>

                <div class="form-group">
                    <input class="btn btn-success" type="submit" value="Enviar" />
                    <input class="btn btn-success" type="button" value="Bandeja de entrada" onclick="openInbox()" />
                    <input class="btn btn-success" type="button" value="Bandeja de Salida" onclick="openOutbox()" />
                </div>

                <div class="form-group">
                    <label>Buscar</label>
                    <div class="input-group">
                        <input type="text" class="form-control" placeholder="Buscar" name="buscar" />
                        <div class="input-group-append">
                            <input class="btn btn-primary" type="button" value="Buscar" onclick="buscarCorreo()" />
                        </div>
                    </div>
                </div>
            </div>
        </form>
    </div>
</div>
```

A la hora de enviar un correo el código envía los datos ingresados a MailController, donde se encuentra el método enviar, que lo que hace es cargarlo en la base de datos de acuerdo al destinatario que eligió el usuario, para así poder verlo desde la bandeja de salida o de entrada de acuerdo al usuario que ingreso. Además cuenta con un buscador en el cual se podrán ingresar datos y los buscará en base a las condiciones establecidas.

## Bandeja de entrada

La función openInbox() se encarga de redirigir al usuario a la bandeja de entrada, donde se mostrarán los correos electrónicos recibidos.

Utilizando @Url.Action("Inbox", "Mail"), se construye una URL que apunta a la acción o método `Inbox` del controlador `MailController`. Esto se logra mediante el método `Url.Action()` de ASP.NET MVC.

En cuanto a la función window.open(), se utiliza para abrir una nueva ventana o pestaña en el navegador. Se pasa la URL generada en el paso anterior como parámetro, lo que hace que el navegador cargue la página correspondiente a la acción Inbox del controlador MailController.

En resumen, la función `openInbox()` redirige al usuario a la bandeja de entrada mostrando los correos electrónicos recibidos. Al hacer clic en el botón asociado a esta función, se abrirá una nueva ventana o pestaña del navegador, cargando la página que muestra la bandeja de entrada con los correos electrónicos recibidos. Esto permite al usuario acceder rápidamente a su bandeja de entrada y visualizar los mensajes entrantes.

```
1  @model List<App.Core.Entities.Mail>
2
3  <h2>Bandeja de entrada</h2>
4  <script>
5      function openInbox() {
6          window.open('@Url.Action("Inbox", "Mail")', '_blank');
7      }
8  </script>
9  <table class="table">
10     <thead>
11         <tr>
12             <th>Remitente</th>
13             <th>Destinatario</th>
14             <th>Asunto</th>
15             <th>Contenido</th>
16         </tr>
17     </thead>
18     <tbody>
19         @foreach (var mensaje in Model)
20         {
21             <tr>
22                 <td>@mensaje.Remitente</td>
23                 <td>@mensaje.Destinatario</td>
24                 <td>@mensaje.Asunto</td>
25                 <td>@mensaje.Contenido</td>
26             </tr>
27         }
28     </tbody>
29 </table>
```

```
public IActionResult Outbox()
{
    var user = UsuarioActual/* User.Identity.Name */;

    var mensajes = _dbContext.Mails
        .Where(m => m.Remitente == user)
        .ToList();

    return View(mensajes);
}
```

## Bandeja de salida

La bandeja de salida es una sección de la aplicación donde los usuarios pueden ver los correos electrónicos que han enviado previamente. Proporciona una lista organizada de los mensajes enviados, lo que permite a los usuarios realizar un seguimiento de las comunicaciones enviadas.

Lo que hace es invocar al método que se encuentra en MailController, que su función es traer a los mensajes que correspondan para la bandeja, y luego se muestra ordenadamente en una grilla.

```
1  @model List<App.Core.Entities.Mail>
2
3  <h2>Bandeja de salida</h2>
4  <script>
5      function openOutbox() {
6          window.open('@Url.Action("Outbox", "Mail")', '_blank');
7      }
8  </script>
9  <table class="table">
10     <thead>
11         <tr>
12             <th>Remitente</th>
13             <th>Destinatario</th>
14             <th>Asunto</th>
15             <th>Contenido</th>
16         </tr>
17     </thead>
18     <tbody>
19         @foreach (var mensaje in Model)
20         {
21             <tr>
22                 <td>@mensaje.Remitente</td>
23                 <td>@mensaje.Destinatario</td>
24                 <td>@mensaje.Asunto</td>
25                 <td>@mensaje.Contenido</td>
26             </tr>
27         }
28     </tbody>
29 </table>
```

```
public IActionResult Outbox()
{
    var user = UsuarioActual;

    var mensajes = _dbContext.Mails
        .Where(m => m.Remitente == user)
        .ToList();

    return View(mensajes);
}
```

## Buscador

La función `buscar()` realiza una búsqueda de correos electrónicos en la bandeja de entrada o en una lista de correos electrónicos, filtrando los resultados según un término de búsqueda especificado.

- Obtiene el término de búsqueda ingresado por el usuario desde un campo de texto o mediante algún otro medio de entrada.
- Construye una URL que apunta a una acción o método en un controlador específico (por ejemplo, `MailController` en este caso) utilizando el método `Url.Action()` de ASP.NET MVC. La URL generada representa la acción o método que manejará la búsqueda de correos electrónicos.
- Abre una nueva ventana o pestaña del navegador utilizando la función `window.open()` de JavaScript. La URL generada en el paso anterior se pasa como parámetro, lo que hace que el navegador cargue la página correspondiente al método o acción que maneja la búsqueda.

En resumen, la función `buscar()` se encarga de obtener el término de búsqueda, construir una URL con ese término de búsqueda y abrir una nueva ventana o pestaña del navegador que mostrará los resultados de la búsqueda en la bandeja de entrada o en la página correspondiente. Esto permite al usuario buscar correos electrónicos específicos según el término ingresado.

```
1  @using App.Core.Entities;
2  @model List<Mail>
3
4  <h2>Resultados de búsqueda</h2>
5
6  @if (Model != null && Model.Count > 0)
7  {
8      <table class="table table-bordered">
9          <thead>
10             <tr>
11                 <th>Asunto</th>
12                 <th>Contenido</th>
13                 <th>Remitente</th>
14                 <th>Destinatario</th>
15             </tr>
16         </thead>
17         <tbody>
18             @foreach (var email in Model)
19             {
20                 <tr>
21                     <td>@email.Asunto</td>
22                     <td>@email.Contenido</td>
23                     <td>@email.Remitente</td>
24                     <td>@email.Destinatarior</td>
25                 </tr>
26             }
27         </tbody>
28     </table>
29 }
30 else
31 {
32     <p>No se encontraron resultados.</p>
33 }
```



## Conclusión

En este proyecto se buscó implementar la arquitectura y los métodos necesarios, para poder gestionar los datos de una forma práctica y eficiente, logrando así una aplicación funcional. En el proceso se adquiere lo necesario para poder crear y gestionar una app Web MVC. Se demostró que al utilizar una arquitectura en capas es beneficioso, ya que nos permite dividir las responsabilidades y así tener una vista clara de la función que cumple cada parte. La capa de vista se encargó de proporcionar la interfaz para los usuarios, mientras que la capa 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.

Durante el proceso del proyecto, hemos aprendido y profundizado conceptos esenciales en el desarrollo.

En resumen, el proyecto de la página web MVC que funciona como un gestor de correos electrónicos ha sido desarrollado siguiendo una arquitectura MVC adecuada y utilizando las mejores prácticas. Hemos logrado una separación clara de responsabilidades y una interfaz intuitiva para los usuarios. El resultado es una aplicación web robusta y funcional que cumple con los requerimientos y facilita la gestión eficiente de correos electrónicos.

**Link al repositorio de GitHub:**

[https://github.com/sbrn-9/Gestor-De-Mails\\_Grupo6](https://github.com/sbrn-9/Gestor-De-Mails_Grupo6)

**Fuentes:**

[Introducción a ASP.NET Core MVC | Microsoft Learn](#)

<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>