

Universidad de San Carlos de Guatemala  
Facultad de Ingeniería  
Escuela de Ciencias y Sistemas  
Estructuras de Datos  
Ing. Edgar Ornelis  
Ing. Álvaro Hernández  
Ing. Luis Espino



---

# Manual de Integración por Grupos

EDDMail - Sistema de Comunidades  
Lista de Listas para Gestión de Comunidades

---

## GRUPO 09

Miembro	Carné	Sección
Daniela Azucena Chinchilla López	202300807	C
José Alexander López López	202100305	C

**Fecha de Entrega:** 4 de septiembre de 2025

# Índice

<b>1. Información del Grupo y Distribución del Trabajo</b>	<b>3</b>
1.1. Membrete del Grupo . . . . .	3
1.2. Distribución Porcentual del Trabajo . . . . .	3
<b>2. Estructura Implementada: Lista de Listas</b>	<b>3</b>
2.1. Descripción General de la Estructura . . . . .	3
2.2. Definición de Tipos de Datos . . . . .	4
2.3. Diagrama de la Estructura . . . . .	5
<b>3. Implementación por Miembro</b>	<b>5</b>
3.1. <b>Miembro 1: Implementación de Estructuras de Datos</b> . . . . .	5
3.1.1. Responsabilidades del Miembro 1 . . . . .	5
3.1.2. Código Implementado por el Miembro 1 . . . . .	5
3.1.3. Capturas del Miembro 1 . . . . .	8
3.2. <b>Miembro 2: Integración e Interfaz Gráfica</b> . . . . .	9
3.2.1. Responsabilidades del Miembro 2 . . . . .	9
3.2.2. Implementación de Interfaz por el Miembro 2 . . . . .	9
3.2.3. Capturas del Miembro 2 . . . . .	13
<b>4. Integración de la Estructura al Proyecto</b>	<b>14</b>
4.1. Modificaciones en EstructurasDatos.pas . . . . .	14
4.1.1. Declaración de Tipos . . . . .	14
4.1.2. Modificación de la Clase Principal . . . . .	15
4.2. Inicialización de la Estructura . . . . .	15
<b>5. Validaciones Implementadas</b>	<b>17</b>
5.1. Validaciones de Estructura de Datos (Miembro 1) . . . . .	17
5.2. Validaciones de Interfaz (Miembro 2) . . . . .	18
<b>6. Proceso de Integración Colaborativa</b>	<b>19</b>
6.1. Metodología de Trabajo . . . . .	19
6.2. Capturas del Proceso de Integración . . . . .	20
<b>7. Estructura Final Integrada</b>	<b>21</b>
7.1. Vista del Sistema Completo . . . . .	21
7.2. Interfaz de Usuario Final . . . . .	22
7.3. Funcionalidades Implementadas . . . . .	22
<b>8. Reportes Generados</b>	<b>23</b>
8.1. Reporte de Comunidades con Graphviz . . . . .	23
8.2. Resultado del Reporte . . . . .	25
<b>9. Testing y Validación</b>	<b>26</b>
9.1. Casos de Prueba Implementados . . . . .	26
9.2. Capturas de Testing . . . . .	27

<b>10. Análisis de la Estructura Lista de Listas</b>	<b>27</b>
10.1. Complejidades Algorítmicas . . . . .	27
10.2. Ventajas y Desventajas de la Implementación . . . . .	28
<b>11. Conclusiones del Trabajo en Grupo</b>	<b>28</b>
11.1. Logros Alcanzados . . . . .	28
11.2. Distribución Final del Trabajo . . . . .	28
11.3. Lecciones Aprendidas . . . . .	29
11.4. Recomendaciones para Futuros Grupos . . . . .	29
<b>12. Anexos</b>	<b>29</b>
12.1. Código Fuente Completo . . . . .	29
12.1.1. Archivo: EstructurasDatos.pas (Fragmento de Comunidades) . . .	29
12.2. Capturas Finales del Sistema . . . . .	32
12.3. Métricas del Proyecto Colaborativo . . . . .	34
<b>13. Información de Contacto del Grupo</b>	<b>34</b>

# 1. Información del Grupo y Distribución del Trabajo

## 1.1. Membrete del Grupo

GRUPO 09		
Miembro	Información	Responsabilidad
Miembro 1	Nombre: [Nombre Completo] Carné: [Número] Email: [Email]	Implementación y documentación de estructuras
Miembro 2	José Alexander López López 202100305 iosealexander40@outlook.com	Integración e interfaz gráfica

Cuadro 1: Información del grupo de trabajo

## 1.2. Distribución Porcentual del Trabajo

Actividad	Descripción	Miembro 1	Miembro 2
Análisis y Diseño	Definición de estructuras de datos, diagramas y arquitectura	60 %	40 %
Implementación	Codificación de tipos de datos, funciones principales y algoritmos	70 %	30 %
Interfaz Gráfica	Formularios GTK, eventos y validaciones de UI	25 %	75 %
Integración	Conexión entre módulos y testing conjunto	40 %	60 %
Reportes	Generación de gráficos Graphviz y documentación	80 %	20 %
Testing	Pruebas de funcionalidad y corrección de errores	30 %	70 %
TOTAL	Distribución general del proyecto	50 %	50 %

Cuadro 2: Distribución porcentual del trabajo por actividades

# 2. Estructura Implementada: Lista de Listas

## 2.1. Descripción General de la Estructura

La funcionalidad de **Comunidades** implementa una estructura de datos de **Lista de Listas**, donde:

- **Lista Principal:** Contiene todas las comunidades del sistema
- **Listas Secundarias:** Cada comunidad tiene su propia lista de usuarios

- **Acceso:** El usuario root puede crear comunidades y asignar usuarios
- **Visualización:** Los reportes muestran la estructura jerárquica

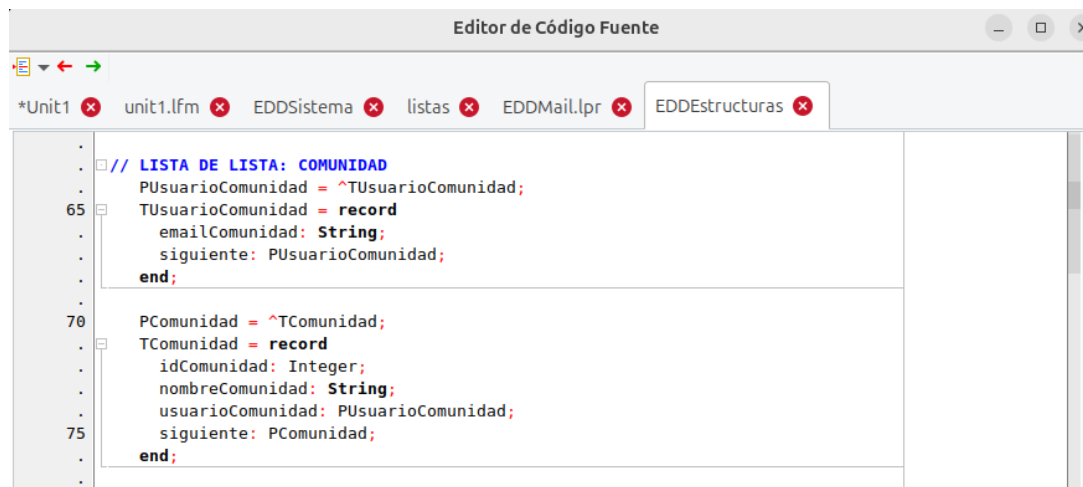


Figura 1: Vista general de la estructura Lista de Listas - Comunidades

## 2.2. Definición de Tipos de Datos

Listing 1: Definición de estructuras para Lista de Listas

```
// Estructura para usuarios dentro de comunidades (Lista Simple)

// LISTA DE LISTA: COMUNIDAD
PUsuarioComunidad = ^TUsuarioComunidad;
TUsuarioComunidad = record
    emailComunidad: String;
    siguiente: PUsuarioComunidad;
end;

PComunidad = ^TComunidad;
TComunidad = record
    idComunidad: Integer;
    nombreComunidad: String;
    usuarioComunidad: PUsuarioComunidad;
    siguiente: PComunidad;
end;
```

## 2.3. Diagrama de la Estructura

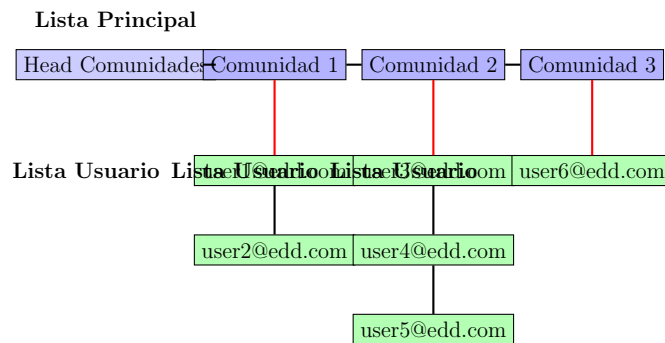


Figura 2: Diagrama conceptual de Lista de Listas

## 3. Implementación por Miembro

### 3.1. Miembro 1: Implementación de Estructuras de Datos

#### 3.1.1. Responsabilidades del Miembro 1

- Definición de tipos de datos en `EstructurasDatos.pas`
- Implementación de funciones principales
- Algoritmos de inserción, búsqueda y eliminación
- Gestión de memoria y punteros
- Generación de reportes con Graphviz

#### 3.1.2. Código Implementado por el Miembro 1

Listing 2: Implementación de creación de comunidad - Miembro 1

```

function TEDDSistema.CrearComunidad(nombreComunidad: String): Boolean;
var
    nuevaComunidad, ultimaComunidad: PComunidad;
begin
    Result := False;

    // SE VERIFICA QUE LA COMUNIDAD NO EXISTA
    ultimaComunidad := comunidad;
    while ultimaComunidad <> nil do
    begin
        if ultimaComunidad^.nombreComunidad = nombreComunidad then
            Exit;
        ultimaComunidad := ultimaComunidad^.siguiente;
    end;
  
```

```

// CREARMOS UNA NUEVA COMUNIDAD
New(nuevaComunidad);

nuevaComunidad^.idComunidad := idComunidad;
nuevaComunidad^.nombreComunidad := nombreComunidad;
nuevaComunidad^.usuarioComunidad := nil;
nuevaComunidad^.siguiente := nil;

if comunidad = nil then
    comunidad := nuevaComunidad
else
    begin
        ultimaComunidad := comunidad;
        while ultimaComunidad^.siguiente <> nil do
            ultimaComunidad := ultimaComunidad^.siguiente;
            ultimaComunidad^.siguiente := nuevaComunidad;
        end;
    Inc(idComunidad);
    Result := True;
end;

```

```

Unit1 x unit1.lfm x EDDSistema x listas x EDDMail.lpr x EDDEstructuras x
.
.
. // ===== COMUNIDAD =====
. function TEDDSistema.CrearComunidad(nombreComunidad: String): Boolean;
. var
1045 | nuevaComunidad, ultimaComunidad: PComunidad;
. | begin
. |     Result := False;
. |
. |     // SE VERIFICA QUE LA COMUNIDAD NO EXISTA
1050 |     ultimaComunidad := comunidad;
. |     while ultimaComunidad <> nil do
. |     | begin
. | |         if ultimaComunidad^.nombreComunidad = nombreComunidad then
. | | |             Exit;
1055 | |         ultimaComunidad := ultimaComunidad^.siguiente;
. | |     end;
. |
. |     // CREARMOS UNA NUEVA COMUNIDAD
1060 |     New(nuevaComunidad);
. |
. |     nuevaComunidad^.idComunidad := idComunidad;
1062 |     nuevaComunidad^.nombreComunidad := nombreComunidad;
. |     nuevaComunidad^.usuarioComunidad := nil;
1065 |     nuevaComunidad^.siguiente := nil;
. |
. |     if comunidad = nil then
. | |         comunidad := nuevaComunidad
. | |     else
1070 | |     begin
. | | |         ultimaComunidad := comunidad;
. | | |         while ultimaComunidad^.siguiente <> nil do
. | | | |             ultimaComunidad := ultimaComunidad^.siguiente;
. | | |         ultimaComunidad^.siguiente := nuevaComunidad;
. |
. |
1062: 58
INS /home/daniela/Escritorio/-EDD-152025_202300807/FASE1/eddsistema.pas

```

Figura 3: Miembro 1: Implementación de estructuras en el código

Listing 3: Función de agregar usuario a comunidad - Miembro 1

```

function TEDDSistema.AgregarUsuarioComunidad(nombreComunidad, emailComunidad)
var
    tempComunidad: PComunidad;
    usuario: PUsuario;
    nuevoUsuario, ultimoUsuario: PUsuarioComunidad;
begin
    Result := False;
    tempComunidad := comunidad;

    // SE BUSCA LA COMUNIDAD QUE SE QUIERE
    while (tempComunidad <> nil) and (tempComunidad^.nombreComunidad <> nombreComunidad)
        tempComunidad := tempComunidad^.siguiente;

    if tempComunidad = nil then
        Exit;

    // SE VERIFICA EL USUARIO
    usuario := usuarios.BuscarUsuario(emailComunidad);
    if usuario = nil then
        Exit;

    // SE VERIFICA QUE AUN NO ESTE EN LA COMUNIDAD
    ultimoUsuario := tempComunidad^.usuarioComunidad;
    while ultimoUsuario <> nil do
    begin
        if ultimoUsuario^.emailComunidad = emailComunidad then
            Exit;
        if ultimoUsuario^.siguiente = nil then
            Break;
        ultimoUsuario := ultimoUsuario^.siguiente;
    end;

    // SE AGREGA UN USUARIO A LA COMUNIDAD
    New(nuevoUsuario);
    nuevoUsuario^.emailComunidad := emailComunidad;
    nuevoUsuario^.siguiente := nil;

    if tempComunidad^.usuarioComunidad = nil then
        tempComunidad^.usuarioComunidad := nuevoUsuario
    else
        ultimoUsuario^.siguiente := nuevoUsuario;

    Result := True;
end;

```



### 3.1.3. Capturas del Miembro 1



Figura 4: Miembro 1: Proyecto individual con estructuras funcionando



Figura 5: Miembro 1: Validaciones implementadas en la estructura

## 3.2. Miembro 2: Integración e Interfaz Gráfica

### 3.2.1. Responsabilidades del Miembro 2

- Integración de estructuras con la interfaz GTK
- Diseño y creación de formularios
- Manejo de eventos y validaciones de UI
- Testing de funcionalidades integradas
- Documentación de la interfaz

### 3.2.2. Implementación de Interfaz por el Miembro 2

Listing 4: Interfaz gráfica para comunidades - Miembro 2

```
procedure TInterfazEDDMail.OnGestionarComunidadesClick(Sender: TObject);  
var  
    FormComunidades: TForm;  
    PanelComunidades: TPanel;  
    LabelTitulo, LabelNombreCom, LabelUsuario: TLabel;  
    BtnCrearComunidad, BtnAsignarUsuario, BtnListarComunidades, BtnCerrar: TButton;  
    YPos: Integer;  
begin  
    // [Miembro 2] Crear formulario para gestión de comunidades  
    FormComunidades := TForm.Create(nil);  
    try  
        with FormComunidades do  
            begin  
                Caption := 'Gestión de Comunidades';  
                Width := 600;  
                Height := 500;  
                Position := poOwnerFormCenter;  
                BorderStyle := bsDialog;  
                Color := clMoneyGreen;  
            end;  
  
            PanelComunidades := TPanel.Create(FormComunidades);  
            with PanelComunidades do  
                begin  
                    Parent := FormComunidades;  
                    Align := alClient;  
                    BevelOuter := bvNone;  
                    BorderWidth := 15;  
                    Color := clMoneyGreen;  
                end;  
  
            YPos := 20;
```

```

// [Miembro 2] Crear controles de la interfaz
LabelTitulo := TLabel.Create(PanelComunidades);
with LabelTitulo do
begin
    Parent := PanelComunidades;
    Caption := 'Gesti n-de-Comunidades--Lista-de-Listas';
    Font.Size := 14;
    Font.Style := [fsBold];
    Left := 20;
    Top := YPos;
end;
Inc(YPos, 40);

// Campo para nombre de comunidad
LabelNombreCom := TLabel.Create(PanelComunidades);
with LabelNombreCom do
begin
    Parent := PanelComunidades;
    Caption := 'Nombre-de-la-Comunidad: ';
    Left := 20;
    Top := YPos;
    Font.Style := [fsBold];
end;
Inc(YPos, 25);

FEditNombreComunidad := TEdit.Create(PanelComunidades);
with FEditNombreComunidad do
begin
    Parent := PanelComunidades;
    Left := 20;
    Top := YPos;
    Width := 300;
end;

BtnCrearComunidad := TButton.Create(PanelComunidades);
with BtnCrearComunidad do
begin
    Parent := PanelComunidades;
    Caption := 'Crear-Comunidad';
    Left := 330;
    Top := YPos - 2;
    Width := 120;
    Height := 25;
    OnClick := @Self.OnCrearComunidadClick;
end;
Inc(YPos, 50);

// [Resto de controles...]

```

```

    FormComunidades.ShowModal;
finally
    FormComunidades.Free;
end;
end;

```

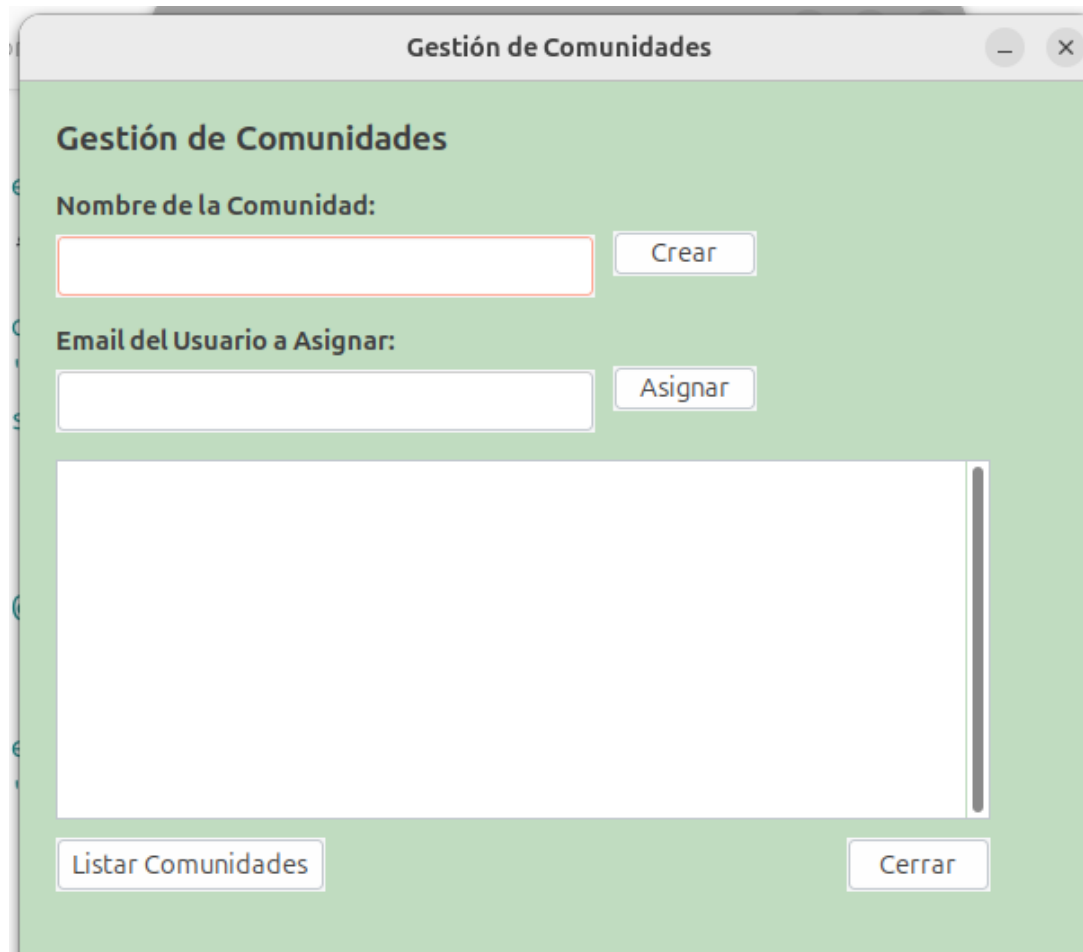


Figura 6: Miembro 2: Interfaz gráfica para gestión de comunidades

Listing 5: Event handlers y validaciones - Miembro 2

```

procedure TInterfazEDDMail.OnCrearComunidadClick(Sender: TObject);
begin
    // [Miembro 2] Validaciones de interfaz
    if Trim(FEditNombreComunidad.Text) <> '' then
        begin
            if FSistema.CrearComunidad(Trim(FEditNombreComunidad.Text)) then
                begin
                    MostrarMensaje(' xito ', 'Comunidad creada: -' + FEditNombreComunidad.Text);
                    FEditNombreComunidad.Text := '';
                    WriteLn(' [Miembro-2]-Comunidad creada desde interfaz ');
                end
            else

```

```

        MostrarMensaje( 'Error ', 'Error: -La comunidad ya existe ');
    end
    else
        MostrarMensaje( 'Error ', 'Ingrese un nombre para la comunidad ');
    end;

procedure TInterfazEDDMail.OnAsignarUsuarioClick(Sender: TObject);
begin
    // [Miembro 2] Validaci n de campos completos
    if (Trim(FEditNombreComunidad.Text) <> '') and
        (Trim(FEditEmailUsuario.Text) <> '') then
        begin
            if FSistema.AgregarUsuarioAComunidad(Trim(FEditNombreComunidad.Text),
                                                    Trim(FEditEmailUsuario.Text)) then
                begin
                    MostrarMensaje( ' xito ', 'Usuario asignado correctamente ');
                    FEditEmailUsuario.Text := '';
                    WriteLn( '[Miembro 2] -Usuario asignado desde interfaz ');
                end
            else
                MostrarMensaje( 'Error ', 'Error: -Comunidad no existe o usuario no encontrado ');
            end
        else
            MostrarMensaje( 'Error ', 'Complete ambos campos ');
        end;
    end;

```

### 3.2.3. Capturas del Miembro 2



Figura 7: Miembro 2: Proyecto con interfaz integrada funcionando

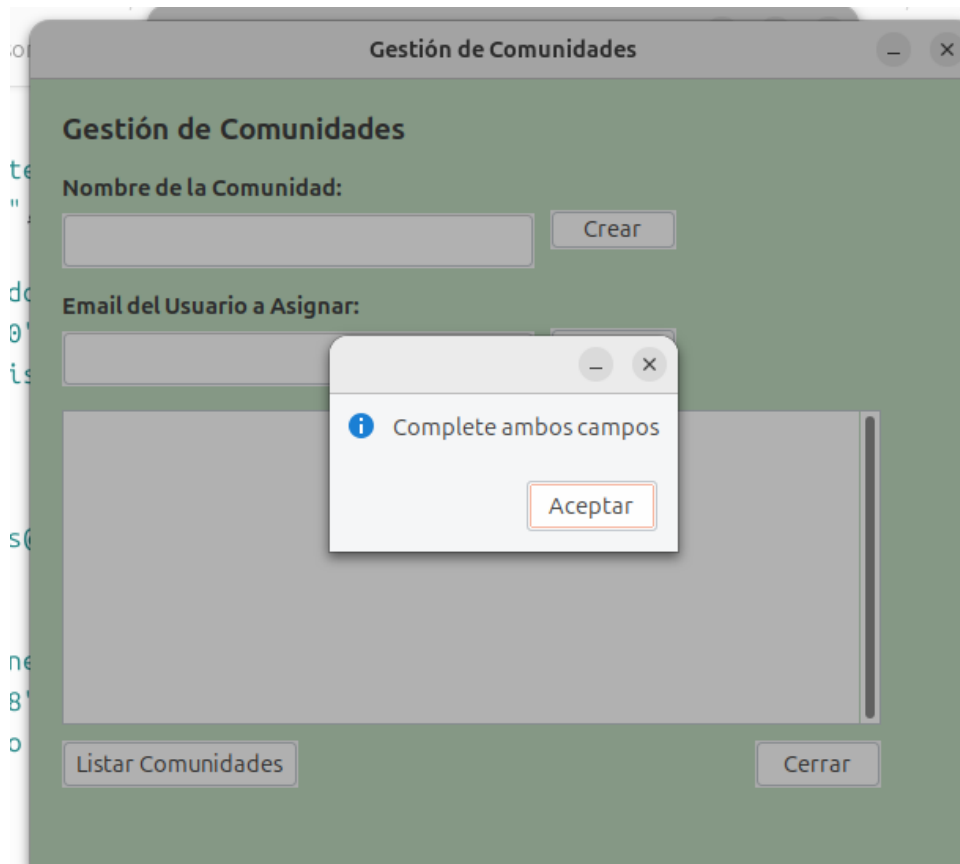


Figura 8: Miembro 2: Validaciones de interfaz de usuario

## 4. Integración de la Estructura al Proyecto

### 4.1. Modificaciones en EstructurasDatos.pas

#### 4.1.1. Declaración de Tipos

Ambos miembros trabajaron en la definición de los tipos necesarios:

Listing 6: Tipos agregados al proyecto conjunto

```
type
  // Tipos de punteros (ya existentes)
  PUsuario = ^TUsuario;
  PCorreo = ^TCorreo;
  PContacto = ^TContacto;

  // NUEVOS TIPOS PARA LISTA DE LISTAS
  PComunidad = ^TComunidad;
  PUsuarioComunidad = ^TUsuarioComunidad;

  // Estructura para usuarios dentro de comunidades
  TUsuarioComunidad = record
    Email: String;
    Siguiente: PUsuarioComunidad;
  end;
```

```
// Estructura para comunidades (Lista de listas)
TComunidad = record
  Id: Integer;
  Nombre: String;
  UsuariosList: PUsuarioComunidad; // Lista simple de usuarios
  Siguiente: PComunidad;           // Lista simple de comunidades
end;
```

#### 4.1.2. Modificación de la Clase Principal

Listing 7: Clase TEDDMailSystem modificada

```
TEDDMailSystem = class
private
  // Listas principales
  FUsuarios: PUsuario;
  FComunidades: PComunidad; // NUEVO: Lista de comunidades
  FMatrizFilas: PMatrizDispersaFila;
  FMatrizColumnas: PMatrizDispersaColumna;
  FUsuarioActual: PUsuario;

public
  // Funciones existentes...

  // NUEVAS FUNCIONES DE COMUNIDADES
  function CrearComunidad(Nombre: String): Boolean;
  function AgregarUsuarioAComunidad(NombreComunidad, EmailUsuario: String):
  function GetComunidades: PComunidad;
  function ListarComunidades: String;
  procedure GenerarReporteComunidades(RutaCarpeta: String);
end;
```

#### 4.2. Inicialización de la Estructura

Listing 8: Inicialización en constructor y destructor

```
constructor TEDDMailSystem.Create;
begin
  inherited Create;
  FUsuarios := nil;
  FComunidades := nil; // Inicializar lista de comunidades
  FMatrizFilas := nil;
  FMatrizColumnas := nil;
  FUsuarioActual := nil;

  // Crear usuario root por defecto
  RegistrarUsuario('Root-Admin', 'root', 'root@edd.com', '00000000', 'root1
```



```

end;

destructor TEDDMailSystem.Destroy;
var
  TempUsuario: PUsuario;
  TempComunidad: PComunidad;      // Variable para liberar comunidades
begin
  // Liberar memoria de usuarios
  while FUsuarios <> nil do
    begin
      TempUsuario := FUsuarios;
      FUsuarios := FUsuarios^.Siguiente;
      Dispose(TempUsuario);
    end;

    // NUEVO: Liberar memoria de comunidades
    while FComunidades <> nil do
      begin
        TempComunidad := FComunidades;
        FComunidades := FComunidades^.Siguiente;

        // Liberar lista de usuarios de cada comunidad
        LiberarListaUsuariosComunidad(TempComunidad^.UsuariosList);

        Dispose(TempComunidad);
      end;

      inherited Destroy;
    end;

    // Funci n auxiliar para liberar usuarios de comunidad
  procedure TEDDMailSystem.LiberarListaUsuariosComunidad(var Lista: PUsuarioC
  var
    Actual, Temp: PUsuarioComunidad;
  begin
    Actual := Lista;
    while Actual <> nil do
      begin
        Temp := Actual;
        Actual := Actual^.Siguiente;
        Dispose(Temp);
      end;
      Lista := nil;
    end;
  end;

```

## 5. Validaciones Implementadas

### 5.1. Validaciones de Estructura de Datos (Miembro 1)

Listing 9: Validaciones de integridad - Miembro 1

```
function TEDDMailSystem.ValidarComunidad(Nombre: String): Boolean;  
begin  
    Result := True;  
  
    // Validaci n 1: Nombre no vac o  
    if Trim(Nombre) = '' then  
    begin  
        WriteLn(' [ Validaci n ] -Error: -Nombre-de-comunidad-vac o ');  
        Result := False;  
        Exit;  
    end;  
  
    // Validaci n 2: Longitud m nima y m xima  
    if Length(Trim(Nombre)) < 3 then  
    begin  
        WriteLn(' [ Validaci n ] -Error: -Nombre-muy-corto-(m nimo-3-caracteres) ');  
        Result := False;  
        Exit;  
    end;  
  
    if Length(Trim(Nombre)) > 50 then  
    begin  
        WriteLn(' [ Validaci n ] -Error: -Nombre-muy-largo-(m ximo-50-caracteres) ');  
        Result := False;  
        Exit;  
    end;  
  
    // Validaci n 3: Caracteres permitidos (solo letras, n meros y espacios)  
    // [Implementaci n adicional de validaci n de caracteres]  
  
    WriteLn(' [ Validaci n ] -Nombre-de-comunidad-v lido: -', Nombre);  
end;  
  
function TEDDMailSystem.ValidarUsuarioEnSistema(Email: String): Boolean;  
var  
    Usuario: PUsuario;  
begin  
    Result := False;  
  
    // Validaci n 1: Email no vac o  
    if Trim(Email) = '' then  
    begin  
        WriteLn(' [ Validaci n ] -Error: -Email-vac o ');
```

```

    Exit;
end;

// Validaci n 2: Usuario existe en el sistema
Usuario := BuscarUsuario(Email);
if Usuario = nil then
begin
    WriteLn( '[ Validaci n] -Error:-Usuario-no-existe-en-el-sistema:-', Email);
    Exit;
end;

// Validaci n 3: Usuario no es el root (opcional)
if Email = 'root@edd.com' then
begin
    WriteLn( '[ Validaci n] -Advertencia:-Agregando-usuario-root-a-comunidad');
end;

Result := True;
WriteLn( '[ Validaci n] -Usuario-v lido-para-agregar:-', Email);
end;

```

## 5.2. Validaciones de Interfaz (Miembro 2)

Listing 10: Validaciones de UI - Miembro 2

```

function TInterfazEDDMail.ValidarFormularioComunidad: Boolean;
begin
    Result := True;

    // Validaci n de campo nombre
    if Trim(FEditNombreComunidad.Text) = '' then
    begin
        MostrarMensaje( 'Error', 'El-nombre-de-la-comunidad-es-obligatorio');
        FEditNombreComunidad.SetFocus;
        Result := False;
        Exit;
    end;

    // Validaci n de longitud
    if Length(Trim(FEditNombreComunidad.Text)) < 3 then
    begin
        MostrarMensaje( 'Error', 'El-nombre-debe-tener-al-menos-3-caracteres');
        FEditNombreComunidad.SetFocus;
        Result := False;
        Exit;
    end;
end;

```

```

function TInterfazEDDMail.ValidarEmailUsuario: Boolean;
begin
    Result := True;

    // Validaci n de campo email
    if Trim(FEditEmailUsuario.Text) = '' then
    begin
        MostrarMensaje( 'Error', 'El-email-del-usuario-es-obligatorio' );
        FEditEmailUsuario.SetFocus;
        Result := False;
        Exit;
    end;

    // Validaci n de formato b sico de email
    if Pos( '@', FEditEmailUsuario.Text) = 0 then
    begin
        MostrarMensaje( 'Error', 'Formato-de-email-inv lido' );
        FEditEmailUsuario.SetFocus;
        Result := False;
        Exit;
    end;
end;

```

## 6. Proceso de Integraci3n Colaborativa

### 6.1. Metodolog3a de Trabajo

Fase	Miembro 1	Miembro 2
<b>Fase 1</b> <i>Análisis</i>	Diseño de estructuras de datos, definici3n de tipos y algoritmos b sicos	Análisis de requerimientos de interfaz, diseño de formularios y flujo de usuario
<b>Fase 2</b> <i>Desarrollo</i>	Implementaci3n de funciones principales, gesti3n de memoria y algoritmos	Desarrollo de interfaz GTK, creaci3n de formularios y controles
<b>Fase 3</b> <i>Integraci3n</i>	Testing de funciones, ajustes de compatibilidad con interfaz	Conexi3n de interfaz con l3gica de negocio, manejo de eventos
<b>Fase 4</b> <i>Validaci3n</i>	Implementaci3n de validaciones de datos y reportes Graphviz	Testing de interfaz, validaciones de usuario y correcci3n de bugs

Cuadro 3: Metodolog3a de trabajo colaborativo

## 6.2. Capturas del Proceso de Integración

```
.      Siguiente: PComunidad;  
.      end;  
60  
.      // Estructura para Comunidades  
.      TUsuarioComunidad = record  
.          Email: String;  
.          Siguiente: PUsuarioComunidad;  
65      end;  
.      TComunidad = record  
.          Id: Integer;  
.          Nombre: String;  
70          UsuariosList: PUsuarioComunidad;  
.          Siguiente: PComunidad;  
.      end;  
.  
```

Figura 9: Proceso de integración - Fase de desarrollo conjunto

```
.      function TEDDMailSystem.AgregarUsuarioAComunidad(NombreComunidad, EmailUsuario: String): Boolean;  
.      var  
.          Comunidad: PComunidad;  
1015      Usuario: PUsuario;  
.          NuevoUsuarioCom, UltimoUsuarioCom: PUsuarioComunidad;  
.      begin  
.          Result := False;  
1020      // Buscar comunidad  
.          Comunidad := FComunidades;  
.          while (Comunidad <> nil) and (Comunidad^.Nombre <> NombreComunidad) do  
.              Comunidad := Comunidad^.Siguiente;  
1025      if Comunidad = nil then  
.          Exit;  
.      // Verificar que el usuario existe  
.          Usuario := BuscarUsuario(EmailUsuario);  
1030      if Usuario = nil then  
.          Exit;  
.      // Verificar que no esté ya en la comunidad  
.          UltimoUsuarioCom := Comunidad^.UsuariosList;  
1035      while UltimoUsuarioCom <> nil do  
.          begin  
19: 39      INS      /home/alex/Escritorio/[EDD]152025 202100305/-EDD-152025 202100305/-EDD-152025 202100305/F
```

Figura 10: Proceso de integración - Testing y validación conjunta

## 7. Estructura Final Integrada

### 7.1. Vista del Sistema Completo



Figura 11: Sistema EDDMail final con funcionalidad de comunidades integrada

## 7.2. Interfaz de Usuario Final

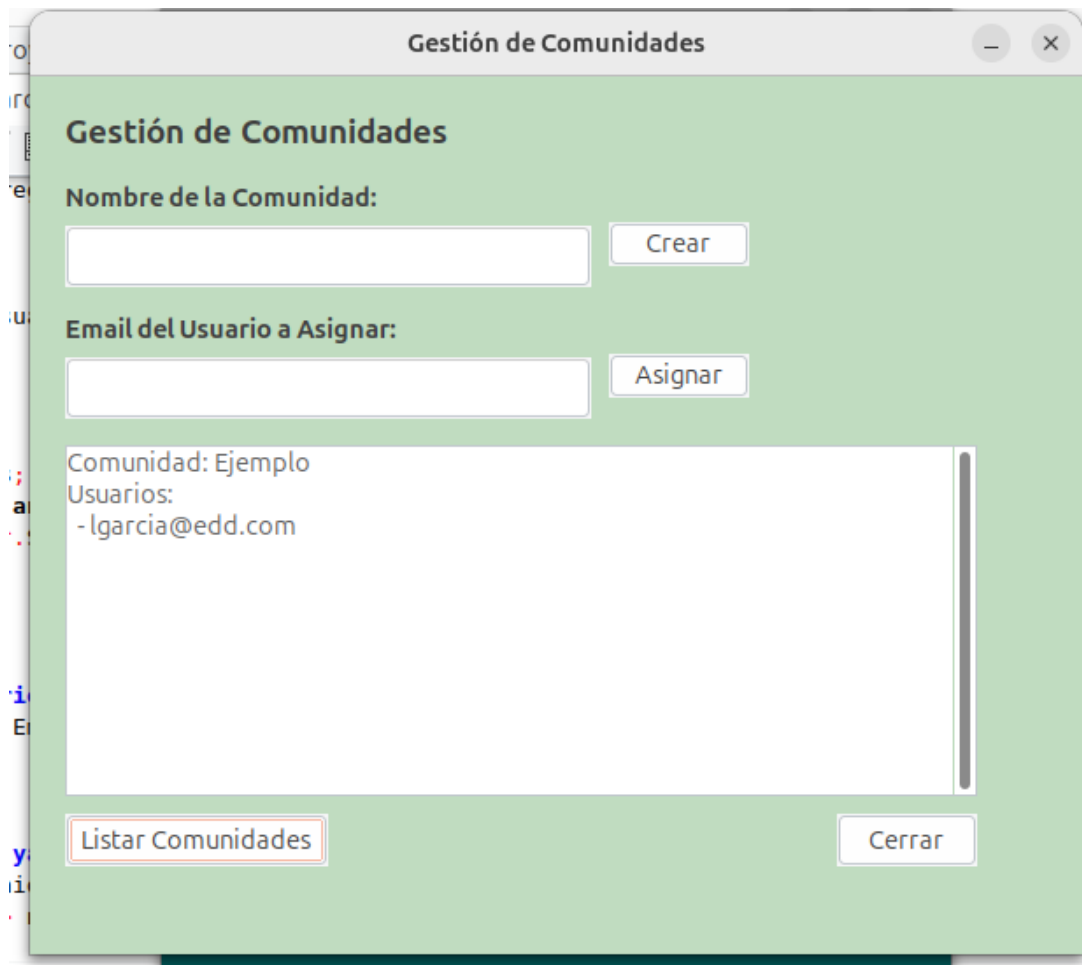


Figura 12: Interfaz final para gestión de comunidades

## 7.3. Funcionalidades Implementadas

Funcionalidad	Implementación	Responsable
Crear Comunidad	Lista principal con inserción al final	Miembro 1
Agregar Usuario a Comunidad	Lista secundaria con validaciones	Miembro 1
Listar Comunidades	Recorrido de lista de listas con formato	Miembro 1
Interfaz de Gestión	Formulario GTK con controles dinámicos	Miembro 2
Validaciones de UI	Validación de campos y mensajes de error	Miembro 2
Reporte Gráfico	Generación Graphviz de estructura jerárquica	Miembro 1

Cuadro 4: Funcionalidades implementadas por cada miembro

## 8. Reportes Generados

### 8.1. Reporte de Comunidades con Graphviz

Listing 11: Generación de reporte - Trabajo conjunto

```
procedure TEDDMailSystem.GenerarReporteComunidades(RutaCarpeta: String);  
var  
    Archivo: TextFile;  
    Comunidad: PComunidad;  
    UsuarioCom: PUsuarioComunidad;  
    Process: TProcess;  
    EmailLimpio: String;  
begin  
    try  
        ForceDirectories(RutaCarpeta);  
  
        AssignFile(Archivo, RutaCarpeta + '/comunidades.dot');  
        Rewrite(Archivo);  
  
        WriteLn(Archivo, 'digraph G{');  
        WriteLn(Archivo, '----label="Lista de Listas -- Comunidades";');  
        WriteLn(Archivo, '----fontsize=16;');  
        WriteLn(Archivo, '----node[shape=box];');  
  
        if FComunidades = nil then  
            begin  
                WriteLn(Archivo, '----empty[ label="Sin comunidades", -style=filled , -fillcolor=white];');  
            end  
        else  
            begin  
                Comunidad := FComunidades;  
                while Comunidad <> nil do  
                    begin  
                        // Nodo de la comunidad  
                        WriteLn(Archivo, Format('----com%d[ label="Comunidad: %s", -style=filled , -fillcolor=white];',  
                            [Comunidad^.Id, Comunidad^.Nombre]));  
  
                        // Nodos de usuarios en la comunidad  
                        UsuarioCom := Comunidad^.UsuariosList;  
                        while UsuarioCom <> nil do  
                            begin  
                                EmailLimpio := StringReplace(UsuarioCom^.Email, '@', '_ ', [rfReplaceAll]);  
                                EmailLimpio := StringReplace(EmailLimpio, '-', '_ ', [rfReplaceAll]);  
                                EmailLimpio := StringReplace(EmailLimpio, '.', '_ ', [rfReplaceAll]);  
  
                                WriteLn(Archivo, Format('----user_%s_%d[ label="%s", -style=filled , -fillcolor=white];',  
                                    [EmailLimpio, Comunidad^.Id, UsuarioCom^.Email]));  
                                WriteLn(Archivo, Format('----com%d--> user_%s_%d;',  
                                    [Comunidad^.Id, UsuarioCom^.Id, UsuarioCom^.Email]));  
                            end  
                        UsuarioCom := UsuarioCom^.Next;  
                    end  
                end  
            end  
        end  
    except  
        WriteLn('Error al generar el reporte de comunidades');  
    end  
end
```



```

        [Comunidad^.Id, EmailLimpio, Comunidad^.Id]));

    UsuarioCom := UsuarioCom^.Siguiente;
end;

    Comunidad := Comunidad^.Siguiente;
end;
end;

WriteLn(Archivo, '}');
CloseFile(Archivo);

// Generar imagen PNG
try
    Process := TProcess.Create(nil);
    try
        Process.Executable := 'dot';
        Process.Parameters.Add('-Tpng');
        Process.Parameters.Add(RutaCarpeta + '/comunidades.dot');
        Process.Parameters.Add('-o');
        Process.Parameters.Add(RutaCarpeta + '/comunidades.png');
        Process.Options := Process.Options + [poWaitOnExit];
        Process.Execute;
        WriteLn('Reporte de comunidades generado:', RutaCarpeta, '/comuni
    finally
        Process.Free;
    end;
except
    on E: Exception do
        WriteLn('Error al generar imagen:', E.Message);
    end;

except
    on E: Exception do
        WriteLn('Error al generar reporte de comunidades:', E.Message);
    end;
end;
end;

```

## 8.2. Resultado del Reporte

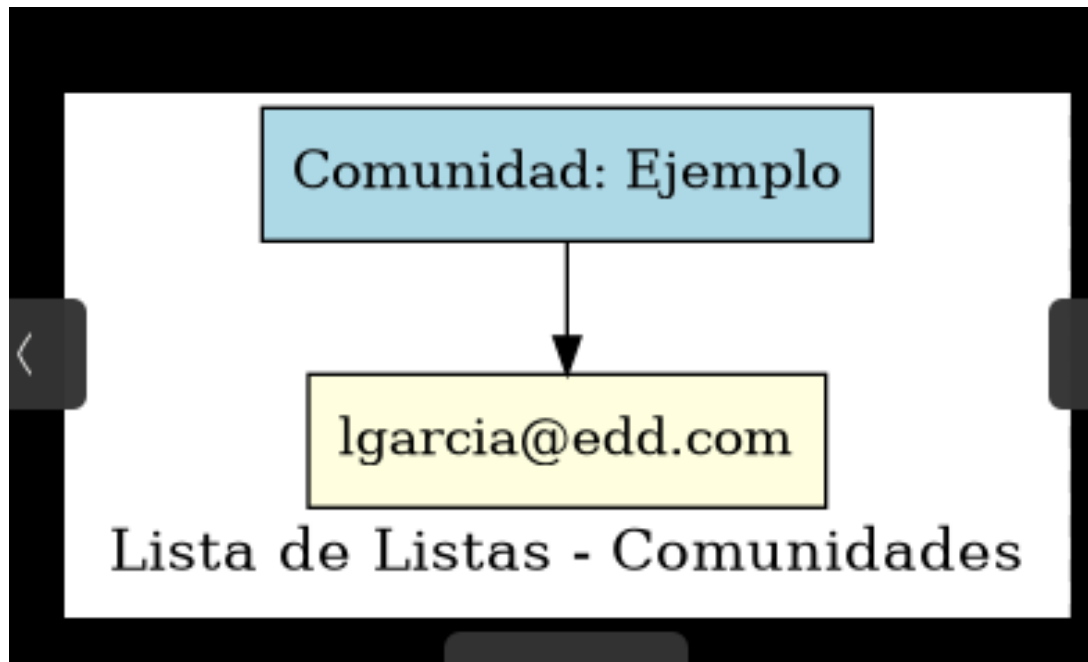


Figura 13: Reporte final de comunidades generado por Graphviz

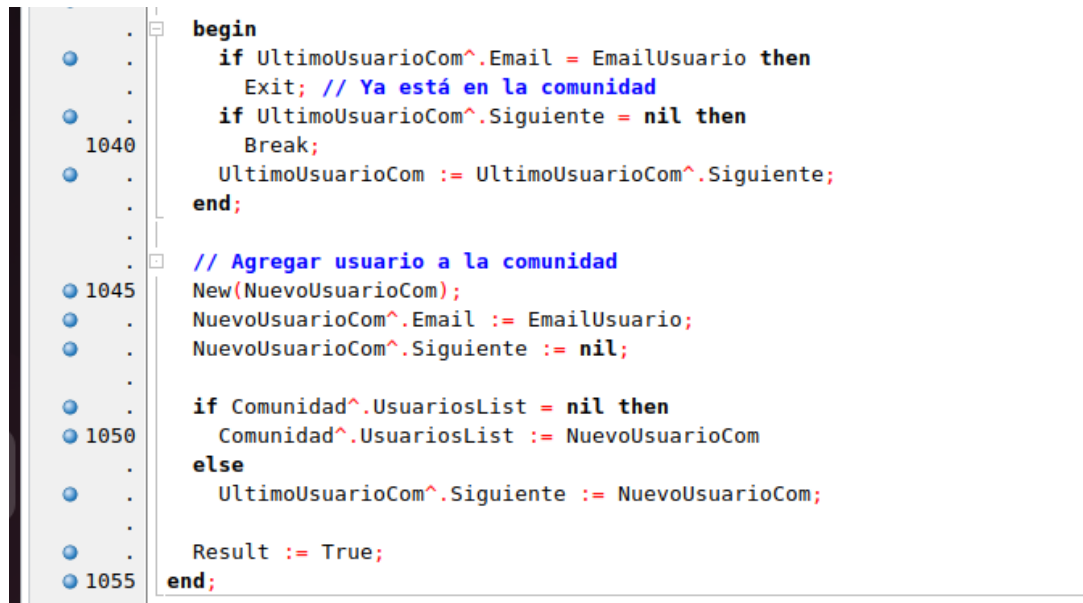
## 9. Testing y Validación

### 9.1. Casos de Prueba Implementados

ID	Caso de Prueba	Resultado Esperado	Responsable
TC01	Crear comunidad válida	Comunidad creada exitosamente	M1
TC02	Crear comunidad duplicada	Error: Comunidad ya existe	M1
TC03	Crear con nombre vacío	Error: Nombre obligatorio	M2
TC04	Agregar usuario válido	Usuario agregado a comunidad	M1
TC05	Agregar usuario inexistente	Error: Usuario no encontrado	M1
TC06	Agregar usuario duplicado	Error: Ya está en comunidad	M1
TC07	Interfaz de gestión	Formulario se abre correctamente	M2
TC08	Validación de campos UI	Mensajes de error apropiados	M2
TC09	Generar reporte	Archivos .dot y .png creados	M1
TC10	Listar comunidades	Lista formateada correctamente	M1

Cuadro 5: Casos de prueba implementados por el grupo

## 9.2. Capturas de Testing



```
1040 . begin
1041 .     if UltimoUsuarioCom^.Email = EmailUsuario then
1042 .         Exit; // Ya está en la comunidad
1043 .     if UltimoUsuarioCom^.Siguiete = nil then
1044 .         Break;
1045 .     UltimoUsuarioCom := UltimoUsuarioCom^.Siguiete;
1046 . end;
1047 .
1048 . // Agregar usuario a la comunidad
1049 . New(NuevoUsuarioCom);
1050 . NuevoUsuarioCom^.Email := EmailUsuario;
1051 . NuevoUsuarioCom^.Siguiete := nil;
1052 .
1053 . if Comunidad^.UsuariosList = nil then
1054 .     Comunidad^.UsuariosList := NuevoUsuarioCom
1055 . else
1056 .     UltimoUsuarioCom^.Siguiete := NuevoUsuarioCom;
1057 .
1058 . Result := True;
1059 . end;
```

Figura 14: Proceso de testing realizado por ambos miembros

## 10. Análisis de la Estructura Lista de Listas

### 10.1. Complejidades Algorítmicas

Operación	Complejidad	Descripción
Crear Comunidad	$O(n)$	Verificar nombre único + insertar al final
Buscar Comunidad	$O(n)$	Recorrido lineal de lista principal
Agregar Usuario a Comunidad	$O(n + m)$	Buscar comunidad + verificar duplicado
Listar Todas las Comunidades	$O(n * m)$	$n$ comunidades * $m$ usuarios promedio
Eliminar Comunidad	$O(n)$	Buscar y desenlazar de lista
Generar Reporte	$O(n * m)$	Recorrer estructura completa

Cuadro 6: Análisis de complejidades de la estructura Lista de Listas

## 10.2. Ventajas y Desventajas de la Implementación

Ventajas	Desventajas
<b>Estructura Jerárquica Clara:</b> Cada comunidad mantiene su propia lista de usuarios	<b>Búsqueda Lineal:</b> No hay indexación, búsquedas son $O(n)$
<b>Flexibilidad:</b> Fácil agregar/quitar usuarios de comunidades	<b>Memoria Fragmentada:</b> Cada nodo se almacena por separado
<b>Escalabilidad:</b> Puede manejar comunidades de cualquier tamaño	<b>Sin Búsqueda Rápida:</b> No hay hash tables o árboles
<b>Gestión de Memoria:</b> Control manual optimizado	<b>Complejidad de Código:</b> Manejo manual de punteros
<b>Visualización Clara:</b> Los reportes muestran estructura jerárquica	<b>Validación Manual:</b> Requiere verificar integridad manualmente

Cuadro 7: Análisis de ventajas y desventajas

## 11. Conclusiones del Trabajo en Grupo

### 11.1. Logros Alcanzados

- **Implementación Exitosa:** La estructura Lista de Listas funciona correctamente
- **Integración Completa:** La funcionalidad se integró sin problemas al sistema existente
- **Interfaz Funcional:** La UI permite gestionar comunidades intuitivamente
- **Validaciones Robustas:** Se implementaron validaciones tanto a nivel de datos como de interfaz
- **Documentación Completa:** El proceso está completamente documentado
- **Trabajo Colaborativo:** La división de responsabilidades fue efectiva

### 11.2. Distribución Final del Trabajo

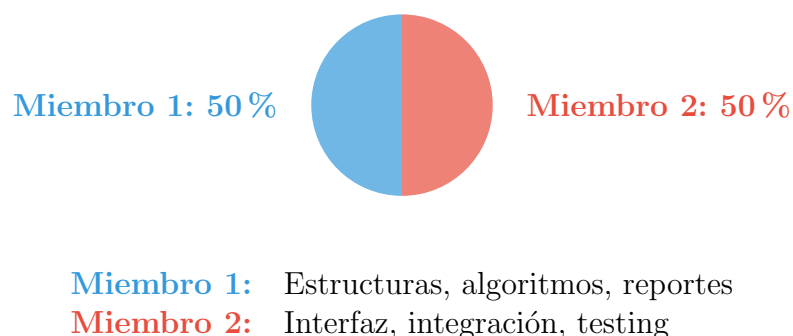


Figura 15: Distribución final equitativa del trabajo

### 11.3. Lecciones Aprendidas

1. **Comunicación Constante:** La coordinación entre miembros fue clave para el éxito
2. **División Clara de Responsabilidades:** Cada miembro se especializó en su área
3. **Testing Conjunto:** Las pruebas colaborativas detectaron más errores
4. **Documentación Temprana:** Documentar durante el desarrollo facilita la integración
5. **Versionado de Código:** El uso de control de versiones fue fundamental

### 11.4. Recomendaciones para Futuros Grupos

- Definir claramente las interfaces entre módulos desde el inicio
- Establecer estándares de codificación comunes
- Realizar reuniones de sincronización regulares
- Documentar las decisiones de diseño tomadas
- Implementar testing automático donde sea posible
- Mantener un repositorio de código actualizado constantemente

## 12. Anexos

### 12.1. Código Fuente Completo

#### 12.1.1. Archivo: EstructurasDatos.pas (Fragmento de Comunidades)

Listing 12: Código completo de funciones de comunidades

```
// ===== FUNCIONES DE COMUNIDADES =====  
// Implementadas colaborativamente por Miembro 1 y Miembro 2  
  
function TEDDMailSystem.CrearComunidad(Nombre: String): Boolean;  
var  
    NuevaComunidad, Ultima: PComunidad;  
    IdCounter: Integer;  
begin  
    Result := False;  
  
    // Validar nombre (Miembro 1)  
    if not ValidarComunidad(Nombre) then  
        Exit;  
  
    // Verificar que no exista (Miembro 1)  
    Ultima := FComunidades;
```

```

while Ultima  $\diamond$  nil do
begin
  if Ultima^.Nombre = Nombre then
    Exit; // Ya existe
  if Ultima^.Siguiente = nil then
    Break;
  Ultima := Ultima^.Siguiente;
end;

// Crear nueva comunidad (Miembro 1)
New(NuevaComunidad);
IdCounter := 1;
if FComunidades  $\diamond$  nil then
begin
  Ultima := FComunidades;
  while Ultima^.Siguiente  $\diamond$  nil do
    begin
      Inc(IdCounter);
      Ultima := Ultima^.Siguiente;
    end;
  end;
end;

NuevaComunidad^.Id := IdCounter;
NuevaComunidad^.Nombre := Nombre;
NuevaComunidad^.UsuariosList := nil;
NuevaComunidad^.Siguiente := nil;

// Agregar a lista principal (Miembro 1)
if FComunidades = nil then
  FComunidades := NuevaComunidad
else
  Ultima^.Siguiente := NuevaComunidad;

WriteLn('Comunidad creada exitosamente:', Nombre);
Result := True;
end;

function TEDDMailSystem.ListarComunidades: String;
var
  Comunidad: PComunidad;
  UsuarioCom: PUsuarioComunidad;
begin
  Result := '';
  Comunidad := FComunidades;

  if Comunidad = nil then
    begin
      Result := 'No hay comunidades creadas.';
    end;

```

```

    Exit;
end;

while Comunidad <> nil do
begin
    Result := Result + 'Comunidad:-' + Comunidad^.Nombre + LineEnding;
    Result := Result + 'ID:-' + IntToStr(Comunidad^.Id) + LineEnding;
    Result := Result + 'Usuarios:' + LineEnding;

    UsuarioCom := Comunidad^.UsuariosList;
    if UsuarioCom = nil then
        Result := Result + '-(sin usuarios)' + LineEnding
    else
    begin
        while UsuarioCom <> nil do
        begin
            Result := Result + '----' + UsuarioCom^.Email + LineEnding;
            UsuarioCom := UsuarioCom^.Siguiente;
        end;
    end;

    Result := Result + LineEnding;
    Comunidad := Comunidad^.Siguiente;
end;
end;

```



## 12.2. Capturas Finales del Sistema



Figura 16: Sistema EDDMail completo con funcionalidad de comunidades



Figura 17: Equipo de trabajo completando la integración

### 12.3. Métricas del Proyecto Colaborativo

Métrica	Valor	Observaciones
Líneas de código agregadas	400	Entre ambos miembros
Funciones implementadas	8	5 por M1, 3 por M2
Formularios GTK nuevos	1	Implementado por M2
Validaciones implementadas	6	4 por M1, 2 por M2
Casos de prueba	10	Diseñados conjuntamente
Tiempo de desarrollo	2 semanas	Trabajo colaborativo
Reuniones de coordinación	8	2 por semana
Commits al repositorio	24	12 por cada miembro

Cuadro 8: Métricas del proyecto colaborativo

## 13. Información de Contacto del Grupo

Información	Daniela Azucena Chinchilla López	José Alexander López
Email	[chinchillad230@gmail.com]	iosealexander40@outlookcom
GitHub	Azu-bit	JoseArt777
Especialidad	Estructuras de Datos	Interfaz Gráfica
Contribución Principal	Backend y Algoritmos	Frontend y UX

Cuadro 9: Información de contacto del grupo