Manual de Integracion - EDDMail

Pablo Daniel Solares Samayoa
Jose Antonio Garcia Roca
Angel Emanuel Rodriguez Corado

Integración de Pablo Daniel Solares Samayoa

LISTA DE LISTAS - COMUNIDADES

```
unit Comunidades;
{$MODE DELPHI}
interface

uses
   SysUtils, Classes, SLL;

type
   TUsuarioData = record
       correo: string;
   end;

   TComunidadData = record
       nombre: string;
   end;

function InsertarUsuario(nombreComunidad,email: string): Boolean;
function InsertarComunidad(nombre: string): Boolean;
function generateGrafComunidades(): string;

implementation

   type
       PUsuarioCell = ^TUsuarioCell;
       TUsuarioCell = record
           correo: string;
           next: PUsuarioCell;
       end;

       PComunidadHeader = ^TComunidadHeader;
       TComunidadHeader = record
           nombre: string;
           listaUsuarios: PUsuarioCell;
           next: PComunidadHeader;
       end;
   var
       Head: PComunidadHeader = nil;
```

```
function InsertarComunidad(nombre: string): Boolean;
var
    nuevaComunidad, actual: PComunidadHeader;
begin
    New(nuevaComunidad);
    nuevaComunidad^.nombre := Trim(nombre);
    nuevaComunidad^.listaUsuarios := nil;
    nuevaComunidad^.next := nil;

    if Head = nil then
    begin
        Head := nuevaComunidad;
    end
    else
    begin
        actual := Head;
        while actual <> nil do
        begin
            if SameText(actual^.nombre, nuevaComunidad^.nombre) then
            begin
                Dispose(nuevaComunidad);
                Result := False;
                Exit;
            end;

            actual := actual^.next;
        end;

        actual := Head;
        while actual^.next <> nil do
        begin
            actual := actual^.next;
        end;
        actual^.next := nuevaComunidad;
    end;
end;

function InsertarUsuario(nombreComunidad,email: string): Boolean;
var
    comunidadActual: PComunidadHeader;
    nuevoUsuario, usuarioActual: PUsuarioCell;
    user: TDataUser;
begin
```

```
       comunidadActual := Head;
       while (comunidadActual  <> nil ) and (comunidadActual^.nombre <>
Trim(nombreComunidad)) do
       begin
           comunidadActual := comunidadActual^.next;
       end;

       if comunidadActual = nil then
       begin
           Result := False;
           Exit;
       end;
       user := SSL_GETBYEMAIL(email);
       if user.email <> '' then
       begin

           New(nuevoUsuario);
           nuevoUsuario^.correo := Trim(email);
           nuevoUsuario^.next := nil;

           if comunidadActual^.listaUsuarios = nil then
           begin
               comunidadActual^.listaUsuarios := nuevoUsuario;
           end
           else
           begin
               usuarioActual := comunidadActual^.listaUsuarios;

               while (usuarioActual^.next <> nil)  do
               begin
                   usuarioActual := usuarioActual^.next;
               end;
               if SameText(usuarioActual^.correo, Trim(email)) then
               begin
                   Dispose(nuevoUsuario);
                   Result := False;
                   Exit;
               end
               else
               begin
                   usuarioActual^.next := nuevoUsuario;
                   Result := True;
```

```
                Exit;
            end;

        end;
    end
    else
    begin
        Result := False;
        Exit;
    end;
end;
function EscapeDotString(const S: string): string;
var
    Res: string;
    i: integer;
begin
    Res := '';
    for i := 1 to Length(S) do
    begin
        case S[i] of
            '"': Res := Res + '\"';
            '\': Res := Res + '\\';
            '|': Res := Res + '\|';
            '{': Res := Res + '\{';
            '}': Res := Res + '\}';
            #10: Res := Res + '\n';
            #13: Res := Res + '\n';
        else
            Res := Res + S[i];

        end;
    end;

    Result := Res;
end;


function generateGrafComunidades(): string;
var
    SL: TStringList;
    comunidadActual: PComunidadHeader;
    usuarioActual: PUsuarioCell;
    contadorComunidad, contadorUsuarios: Integer;
```

```
       nodoComunidad, nodoUsuario, nodoSig: string;
       resultT: string;
   begin
       SL := TStringList.Create;

       SL.Add('digraph Comunidades {');
       SL.Add('  rankdir=LR;');
       SL.Add('  nodesep=0.5;');
       SL.Add('  edge[arrowhead=normal];');
       SL.Add('  node [shape=box, style=filled,
fillcolor=lightblue];');
       SL.Add('');

       if Head = nil then
       begin
           SL.Add('      null [label="LISTAVACIA", shape=plaintext];');
       end
       else
       begin
           contadorComunidad := 0;
           comunidadActual := Head;

           while comunidadActual <> nil do
           begin
               nodoComunidad := Format('comunidad%d',
[contadorComunidad]);
               SL.Add(Format(' %s [label="%s", fillcolor=lightgreen];',
[nodoComunidad, EscapeDotString(comunidadActual^.nombre)]));

               if comunidadActual^.next <> nil then
               begin
                   nodoSig := Format('comunidad%d', [contadorComunidad
+ 1]);
                   SL.Add(Format(' %s -> %s;',
[nodoComunidad,nodoSig]));
               end;

               if comunidadActual^.listaUsuarios <> nil then
               begin
                   contadorUsuarios := 0;
                   usuarioActual := comunidadActual^.listaUsuarios;
                   nodoUsuario := Format('usuario%d_%d',
[contadorComunidad, contadorUsuarios]);
```

```
                    SL.Add(Format('    %s -> %s;', [nodoComunidad,
nodoUsuario]));

                    while usuarioActual <> nil do
                    begin
                        nodoUsuario := Format('usuario%d_%d',
[contadorComunidad, contadorUsuarios]);
                        SL.Add(Format('    %s [label="%s"];' ,
[nodoUsuario, EscapeDotString(usuarioActual^.correo)]));

                        if usuarioActual^.next <> nil then
                        begin
                            nodoSig := Format('usuario%d_%d',
[contadorComunidad, contadorUsuarios + 1]);
                            SL.Add(Format('    %s -> %s; ',
[nodoUsuario, nodoSig]));
                        end;

                        Inc(contadorUsuarios);
                        usuarioActual := usuarioActual^.next;
                    end;

                    SL.Add('{rank=same; ' + nodoComunidad);
                    for contadorUsuarios := 0 to contadorUsuarios - 1 do
                    begin
                        SL.Add(' ' + Format('usuario%d_%d',
[contadorComunidad, contadorUsuarios]));
                    end;
                    SL.Add(' }');
                end;
                Inc(contadorComunidad);
                comunidadActual := comunidadActual^.next;
            end;
        end;

        SL.Add('}');
        resultT := SL.Text;
        SL.Free;

        Result := resultT;
    end;
end.
```

Función en la lista de usuarios para ver si existe el usuario

```
function SSL_GETBYEMAIL(const email: string): TDataUser;
   var
       actual : PNode;

   begin
       Result.id := '';
       Result.name := '';
       Result.email := '';
       Result.phone := '';
       Result.password := '';
       Result.user := '';



       if Head = nil then exit;

       actual := Head;
       while actual <> nil do
       begin
           if actual^.email = Trim(email) then
           begin
               Result.id := actual^.id;
               Result.user := actual^.username;
               Result.name := actual^.name;
               Result.email := actual^.email;
               Result.phone := actual^.phone;
               Result.password := actual^.password;
               Result.Contacts := actual^.Contacts;
               Result.Inbox := actual^.Inbox;
               Result.Papelera := actual^.Papelera;
               Result.Programados := actual^.Programados;




               Exit;
           end;
           actual := actual^.Next;
       end;

   end;
```

Vista donde se llenan los datos

```
unit crearComunidad;

interface
   procedure ShowComunidadesInterface;

implementation
uses
   gtk2, glib2, gdk2, SysUtils, Comunidades, messageUtils, menuAdmin;

var
   ComunidadesInterface: PGtkWidget;
   grid: PGtkWidget;
   lblTitulo, lblNombre, lblComunidad, lblCorreo: PGtkWidget;
   txtNombreInput, txtComunidadInput, txtCorreoInput: PGtkWidget;
   btnCrear, btnAgregar, btnVolver: PGtkWidget;

//FUNCIONES BOTON CREAR COMUNIDAD
procedure OnCrearClick(widget: PGtkWidget; data: gpointer); cdecl;
var
   txtComunidadInput_entry : Pchar;
   response: Boolean;
begin
   txtComunidadInput_entry :=
Pchar(gtk_entry_get_text(GTK_ENTRY(txtNombreInput)));

   response := InsertarComunidad(txtComunidadInput_entry);
   if response then
   begin
       ShowSuccessMessage(ComunidadesInterface, 'Comunidad Creada','La
comunidad ha sido creada con exito');
   end
   else
   begin
       ShowErrorMessage(ComunidadesInterface, 'Comunidad error', 'Ya
existe una comunidad con ese nombre. Intenta con otro nombre' );
   end;
end;



//FUNCIONES BOTON AGREGAR - AGREGAR USUARIO A LA COMUNIDAD
```

```
procedure OnAgregarClick(widget: PGtkWidget; data: gpointer); cdecl;
var
   txtComunidadInput_entry, txtUsuarioInput_entry : Pchar;
   response: Boolean;
begin
    txtComunidadInput_entry :=
Pchar(gtk_entry_get_text(GTK_ENTRY(txtComunidadInput)));
    txtUsuarioInput_entry :=
Pchar(gtk_entry_get_text(GTK_ENTRY(txtCorreoInput)));

   response := InsertarUsuario(txtComunidadInput_entry,
txtUsuarioInput_entry);
   if response then
   begin
       ShowSuccessMessage(ComunidadesInterface, 'Usuario añadido
correctamente','El usuario ha sido agregado a la comunidad
correctamente');
   end
   else
   begin
       ShowErrorMessage(ComunidadesInterface, 'Usuario error',
'Verifica que el nombre de la comunidad exista o que el usuario
ingresado exista o ya lo hagas ingresado' );
   end;
end;
```

```
EN TEORIA ESTO NO SIRVE - DISEÑO Y FUNCION DE ABRIR Y CERRAR


procedure OnVolverClick(widget: PGtkWidget; data: gpointer); cdecl;
begin
   gtk_widget_destroy(ComunidadesInterface);
   ShowmenuAdminInterface;
end;
procedure ShowComunidadesInterface;
begin
   gtk_init(@argc, @argv);

   ComunidadesInterface := gtk_window_new(GTK_WINDOW_TOPLEVEL);
   gtk_window_set_title(GTK_WINDOW(ComunidadesInterface),
'Comunidades');
   gtk_container_set_border_width(GTK_CONTAINER(ComunidadesInterface),
10);
   gtk_window_set_default_size(GTK_WINDOW(ComunidadesInterface), 300,
400);


   grid := gtk_table_new(7, 2, False);
   gtk_container_add(GTK_CONTAINER(ComunidadesInterface), grid);


   lblTitulo := gtk_label_new('COMUNIDADES');
   gtk_table_attach_defaults(GTK_TABLE(grid), lblTitulo, 0,2,0,1);


   lblNombre := gtk_label_new('Nombre');
   txtNombreInput := gtk_entry_new;
   gtk_table_attach_defaults(GTK_TABLE(grid), lblNombre, 0,1,1,2);
   gtk_table_attach_defaults(GTK_TABLE(grid), txtNombreInput, 1,2,1,2);


   btnCrear := gtk_button_new_with_label('Crear');
   gtk_table_attach_defaults(GTK_TABLE(grid), btnCrear, 0,2,2,3);
   g_signal_connect(btnCrear, 'clicked', G_CALLBACK(@OnCrearClick),
nil);


   lblComunidad := gtk_label_new('Comunidad');
   txtComunidadInput := gtk_entry_new;
```

```
   gtk_table_attach_defaults(GTK_TABLE(grid), lblComunidad, 0,1,3,4);
   gtk_table_attach_defaults(GTK_TABLE(grid), txtComunidadInput,
1,2,3,4);

    lblCorreo := gtk_label_new('Correo');
   txtCorreoInput := gtk_entry_new;
   gtk_table_attach_defaults(GTK_TABLE(grid), lblCorreo, 0,1,4,5);
   gtk_table_attach_defaults(GTK_TABLE(grid), txtCorreoInput, 1,2,4,5);


   btnAgregar := gtk_button_new_with_label('Agregar');
   gtk_table_attach_defaults(GTK_TABLE(grid), btnAgregar, 0,2,5,6);
   g_signal_connect(btnAgregar, 'clicked', G_CALLBACK(@OnAgregarClick),
nil);

   btnVolver := gtk_button_new_with_label('Volver');
   gtk_table_attach_defaults(GTK_TABLE(grid), btnVolver, 0,2,6,7);
   g_signal_connect(btnVolver, 'clicked', G_CALLBACK(@OnVolverClick),
nil);

   gtk_widget_show_all(ComunidadesInterface);

   g_signal_connect(ComunidadesInterface, 'destroy',
G_CALLBACK(@gtk_main_quit), nil);
   gtk_main;
end;

end.
```


```
unit reportUtils;
{$mode objfpc}{$H+}


interface

uses
   SysUtils;
   procedure GenerarReportes(const ReportName, FolderN, extension:
string);
implementation
   uses
       Process, Classes;
   procedure GenerarReportes(const ReportName, FolderN, extension:
string);
   var
       DotFilePath, PngFilePath: string;
       FolderCreated: Boolean;
```

```
      DotFile: Text;
      CmdOutPut: AnsiString;
      SL: TStringList;
      i: Integer;
   begin
      if not DirectoryExists(FolderN) then
      begin
          FolderCreated := CreateDir(FolderN);
          if not FolderCreated then
          begin
              Writeln('Error: No se pudo crear la carpeta "', FolderN
, '"');
              exit;
          end;
      end;

      DotFilePath := FolderN + PathDelim + ReportName + '.dot';
      PngFilePath := FolderN + PathDelim + ReportName + '.png';
      SL := TStringList.Create;
      SL.Text := extension;
      Assign(DotFile, DotFilePath);
      Rewrite(DotFile);
      for i := 0 to SL.Count - 1 do
          Writeln(DotFile, SL[i]);
      Close(DotFile);
      SL.Free;

      if  RunCommand('dot', ['-Tpng', DotFilePath, '-o', PngFilePath],
CmdOutPut) then
      begin
          Writeln('Reporte Generado Correctamente: ', PngFilePath);
      end
      else
      begin
          Writeln('Error al generar la imagen');
          Writeln(PngFilePath);
      end;

   end;
```

```
end.
```

Integración Jose Antonio García Roca

Carnet: 202401166

Creación de la clase Comunidad

```
unit Comunidades;

{$mode ObjFPC}{$H+}

interface

uses
  Classes, SysUtils, listaUsuarios, usuario, globals;

type
   TUsuarioData = record
       correo: string;
   end;


   TComunidadData = record
       nombre: string;
   end;


function InsertarUsuario(nombreComunidad,email: string): Boolean;
function InsertarComunidad(nombre: string): Boolean;
function generateGrafComunidades(): string;

implementation

   type
       PUsuarioCell = ^TUsuarioCell;
       TUsuarioCell = record
           correo: string;
           next: PUsuarioCell;
       end;


       PComunidadHeader = ^TComunidadHeader;
       TComunidadHeader = record
           nombre: string;
           listaUsuarios: PUsuarioCell;
           next: PComunidadHeader;
       end;
   var
       Head: PComunidadHeader = nil;
```

Función Insertar Comunidad e Insertar Usuario

```
function InsertarComunidad(nombre: string): Boolean;
   var
       nuevaComunidad, actual: PComunidadHeader;
    begin
       New(nuevaComunidad);
       nuevaComunidad^.nombre := Trim(nombre);
       nuevaComunidad^.listaUsuarios := nil;
       nuevaComunidad^.next := nil;


       if Head = nil then
       begin
           Head := nuevaComunidad;
           Result := True;
           Exit;
       end
       else
       begin
           actual := Head;
           while actual <> nil do
           begin
               if SameText(actual^.nombre, nuevaComunidad^.nombre) then
               begin
                   Dispose(nuevaComunidad);
                   Result := False;
                   Exit;
               end;

               actual := actual^.next;
           end;


           actual := Head;
           while actual^.next <> nil do
           begin
               actual := actual^.next;
           end;
           actual^.next := nuevaComunidad;
           Result := True;
       end;
   end;


function InsertarUsuario(nombreComunidad,email: string): Boolean;
   var
       comunidadActual: PComunidadHeader;
       nuevoUsuario, usuarioActual: PUsuarioCell;
       user: TUsuario;
   begin

       comunidadActual := Head;
        while (comunidadActual  <> nil ) and (comunidadActual^.nombre <> Trim(nombreComunidad)) do
       begin
           comunidadActual := comunidadActual^.next;
       end;


       if comunidadActual = nil then
       begin
           Result := False;
           Exit;
       end;
       user := ListaUsuariosGlobal.Logearse(email);
       if user.getEmail <> '' then
       begin

           New(nuevoUsuario);
           nuevoUsuario^.correo := Trim(email);
           nuevoUsuario^.next := nil;


           if comunidadActual^.listaUsuarios = nil then
           begin
               comunidadActual^.listaUsuarios := nuevoUsuario;
           end
           else
           begin
               usuarioActual := comunidadActual^.listaUsuarios;

               while (usuarioActual^.next <> nil)  do
               begin
                   usuarioActual := usuarioActual^.next;
               end;
               if SameText(usuarioActual^.correo, Trim(email)) then
               begin
                   Dispose(nuevoUsuario);
                   Result := False;
                   Exit;
               end
               else
               begin
                   usuarioActual^.next := nuevoUsuario;
                   Result := True;
                   Exit;
               end;
           end;
       end
       else
       begin
           Result := False;
           Exit;
       end;
   end;
```

Función para generar .dot

```
function generateGrafComunidades(): string;
var
    SL: TStringList;
    comunidadActual: PComunidadHeader;
    usuarioActual: PUsuarioCell;
    contadorComunidad, contadorUsuarios: Integer;
    nodoComunidad, nodoUsuario, nodoSig: string;
    resultT: string;
begin
    SL := TStringList.Create;


    SL.Add('digraph Comunidades {');
    SL.Add('  rankdir=LR;');
    SL.Add('  nodesep=0.5;');
    SL.Add('  edge[arrowhead=normal];');
    SL.Add('  node [shape=box, style=filled, fillcolor=lightblue];');
    SL.Add('');


    if Head = nil then
    begin
        SL.Add('      null [label="LISTAVACIA", shape=plaintext];');
    end
    else
    begin
        contadorComunidad := 0;
        comunidadActual := Head;


        while comunidadActual <> nil do
        begin
            nodoComunidad := Format('comunidad%d', [contadorComunidad]);
            SL.Add(Format(' %s [label="%s", fillcolor=lightgreen];', [nodoComunidad,
EscapeDotString(comunidadActual^.nombre)]));


            if comunidadActual^.next <> nil then
            begin
                nodoSig := Format('comunidad%d', [contadorComunidad + 1]);
                SL.Add(Format(' %s -> %s;', [nodoComunidad,nodoSig]));
            end;


            if comunidadActual^.listaUsuarios <> nil then
            begin
                contadorUsuarios := 0;
                usuarioActual := comunidadActual^.listaUsuarios;
                nodoUsuario := Format('usuario%d_%d', [contadorComunidad, contadorUsuarios]);
                 SL.Add(Format('   %s -> %s;', [nodoComunidad, nodoUsuario]));


                while usuarioActual <> nil do
                begin
                    nodoUsuario := Format('usuario%d_%d', [contadorComunidad, contadorUsuarios]);
                    SL.Add(Format('    %s [label="%s"];' , [nodoUsuario, EscapeDotString(usuarioActual^.correo)]));


                    if usuarioActual^.next <> nil then
                    begin
                        nodoSig := Format('usuario%d_%d', [contadorComunidad, contadorUsuarios + 1]);
                        SL.Add(Format('    %s -> %s; ', [nodoUsuario, nodoSig]));
                    end;


                    Inc(contadorUsuarios);
                    usuarioActual := usuarioActual^.next;
                end;


                SL.Add('{rank=same; ' + nodoComunidad);
                for contadorUsuarios := 0 to contadorUsuarios - 1 do
                begin
                    SL.Add(' ' + Format('usuario%d_%d', [contadorComunidad, contadorUsuarios]));
                end;
                SL.Add(' }');
            end;
            Inc(contadorComunidad);
            comunidadActual := comunidadActual^.next;
        end;
    end;


    SL.Add('}');
    resultT := SL.Text;
    SL.Free;


    Result := resultT;
end;
end.
```

Implementación en la vista, generación de .png y generación de comunidades e inserción de usuarios.

```
unit crearComunidadad;

{$mode ObjFPC}{$H+}

interface

uses
  Classes, SysUtils, Forms, Controls, Graphics, Dialogs, Comunidades, Process, StdCtrls;

type

  { TForm14 }

  TForm14 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
  private

  public

  end;

var
  Form14: TForm14;

implementation
uses menuAdmin;

{$R *.lfm}

{ TForm14 }

procedure TForm14.Button3Click(Sender: TObject);
begin
  Close;
end;

procedure TForm14.Button4Click(Sender: TObject);
var
  SL: TStringList;
  dotContent: string;
  dotFile, pngFile: string;
  AProcess: TProcess;
begin

  dotFile := 'Reporte/comunidades.dot';
  pngFile := 'Reporte/comunidades.png';

  if not DirectoryExists('Reporte') then
    CreateDir('Reporte');

  dotContent := generateGrafComunidades();

  SL := TStringList.Create;
  try
    SL.Text := dotContent;
    SL.SaveToFile(dotFile);
  finally
    SL.Free;
  end;

  AProcess := TProcess.Create(nil);
  try
    AProcess.Executable := 'dot';
    AProcess.Parameters.Add('-Tpng');
    AProcess.Parameters.Add(dotFile);
    AProcess.Parameters.Add('-o');
    AProcess.Parameters.Add(pngFile);
    AProcess.Options := AProcess.Options + [poWaitOnExit];
    AProcess.Execute;
  finally
    AProcess.Free;
  end;

  ShowMessage('Reporte Comunidades Generado');

end;

procedure TForm14.Button1Click(Sender: TObject);
var
  txtComunidadInput_entry : Pchar;
  response: Boolean;
begin
  txtComunidadInput_entry := Pchar(Edit1.Text);
  response := InsertarComunidad(txtComunidadInput_entry);
  if response then
       ShowMessage('La comunidad ha sido creada con exito')
  else
       ShowMessage('Ya existe una comunidad con ese nombre. Intenta con otro nombre' );
end;

procedure TForm14.Button2Click(Sender: TObject);
var
   txtComunidadInput_entry, txtUsuarioInput_entry : Pchar;
   response: Boolean;
begin
   txtComunidadInput_entry := Pchar(Edit2.Text);
   txtUsuarioInput_entry := Pchar(Edit3.Text);

   response := InsertarUsuario(txtComunidadInput_entry, txtUsuarioInput_entry);

   if response then
      ShowMessage('El usuario ha sido agregado a la comunidad correctamente')
   else
      ShowMessage('Verifica que el nombre de la comunidad exista o que el usuario ingresado exista o ya este
ingresado');

end;

procedure TForm14.FormClose(Sender: TObject; var CloseAction: TCloseAction);
begin
  Form2.Show;
end;

end.
```

Integración de Angel Emanuel Rodriguez Corado

Comunidades

Se implemento la estructura de comunidades con sus respectivos metodos, como InsertarComunidad, InsertarUsuario, ambas recorren la respectiva lista para verificar si existen o no la comunidad como el usuario en el sistema, y el metodo que se encarga de generar el .dot

```
unit Comunidades;

{$MODE DELPHI}

interface

uses
  Classes, SysUtils, Forms, Controls, Graphics, Dialogs, StdCtrls, SLL,
Process;

type
  TUsuarioData = record
    correo: string;
  end;

  TComunidadData = record
    nombre: string;
  end;

function InsertarUsuario(nombreComunidad,email: string): Boolean;
function InsertarComunidad(nombre: string): Boolean;
function generateGrafComunidades(): string;

implementation

type
  PUsuarioCell = ^TUsuarioCell;
  TUsuarioCell = record
    correo: string;
    next: PUsuarioCell;
  end;
```

```
  PComunidadHeader = ^TComunidadHeader;
  TComunidadHeader = record
    nombre: string;
    listaUsuarios: PUsuarioCell;
    next: PComunidadHeader;
  end;

var
  Head: PComunidadHeader = nil;

function InsertarComunidad(nombre: string): Boolean;
var
  nuevaComunidad, actual: PComunidadHeader;
begin
  Result := False;
  New(nuevaComunidad);
  nuevaComunidad^.nombre := Trim(nombre);
  nuevaComunidad^.listaUsuarios := nil;
  nuevaComunidad^.next := nil;

  if Head = nil then
  begin
    Head := nuevaComunidad;
    Result := True;
    Exit;
  end;

  actual := Head;
  // Verificar si ya existe
  while actual <> nil do
  begin
    if SameText(actual^.nombre, nuevaComunidad^.nombre) then
    begin
      Dispose(nuevaComunidad);
      Exit;
    end;
    actual := actual^.next;
  end;

  // Insertar al final
  actual := Head;
  while actual^.next <> nil do
    actual := actual^.next;
```

```
  actual^.next := nuevaComunidad;
  Result := True;
end;

function InsertarUsuario(nombreComunidad, email: string): Boolean;
var
  comunidadActual: PComunidadHeader;
  nuevoUsuario, usuarioActual: PUsuarioCell;
  user: TDataUser;
begin
  Result := False;

  comunidadActual := Head;
  while (comunidadActual <> nil) and (comunidadActual^.nombre <>
Trim(nombreComunidad)) do
    comunidadActual := comunidadActual^.next;

  if comunidadActual = nil then Exit; // no encontrada

  // Verificar en SLL
  user := SSL_GETBYEMAIL(email);
  if user.email = '' then Exit;

  New(nuevoUsuario);
  nuevoUsuario^.correo := Trim(email);
  nuevoUsuario^.next := nil;

  if comunidadActual^.listaUsuarios = nil then
  begin
    comunidadActual^.listaUsuarios := nuevoUsuario;
    Result := True;
    Exit;
  end;

  usuarioActual := comunidadActual^.listaUsuarios;
  while usuarioActual <> nil do
  begin
    if SameText(usuarioActual^.correo, Trim(email)) then
    begin
      Dispose(nuevoUsuario);
      Exit;
    end;
```

```
    if usuarioActual^.next = nil then Break;
    usuarioActual := usuarioActual^.next;
  end;

  usuarioActual^.next := nuevoUsuario;
  Result := True;
end;

function EscapeDotString(const S: string): string;
var
  Res: string;
  i: integer;
begin
  Res := '';
  for i := 1 to Length(S) do
  begin
    case S[i] of
      '"': Res := Res + '\"';
      '\': Res := Res + '\\';
      '|': Res := Res + '\|';
      '{': Res := Res + '\{';
      '}': Res := Res + '\}';
      #10, #13: Res := Res + '\n';
    else
      Res := Res + S[i];
    end;
  end;
  Result := Res;
end;

procedure GenerateDotComunidades(const DotFile, PNGFile: string);
var
  SL: TStringList;
  comunidadActual: PComunidadHeader;
  usuarioActual: PUsuarioCell;
  contadorComunidad, contadorUsuarios: Integer;
  nodoComunidad, nodoUsuario, nodoSig: string;
  Output: AnsiString;
begin
  SL := TStringList.Create;
  try
    SL.Add('digraph Comunidades {');
    SL.Add('  rankdir=LR;');
```

```
    SL.Add('  nodesep=0.5;');
    SL.Add('  edge[arrowhead=normal];');
    SL.Add('  node [shape=box, style=filled, fillcolor=lightblue];');
    SL.Add('');

    if Head = nil then
      SL.Add('  null [label="LISTA VACÍA", shape=plaintext];')
    else
    begin
      contadorComunidad := 0;
      comunidadActual := Head;

      while comunidadActual <> nil do
      begin
        nodoComunidad := Format('comunidad%d', [contadorComunidad]);
        SL.Add(Format('  %s [label="%s", fillcolor=lightgreen];',
          [nodoComunidad, EscapeDotString(comunidadActual^.nombre)]));

        if comunidadActual^.next <> nil then
        begin
          nodoSig := Format('comunidad%d', [contadorComunidad + 1]);
          SL.Add(Format('  %s -> %s;', [nodoComunidad, nodoSig]));
        end;

        if comunidadActual^.listaUsuarios <> nil then
        begin
          contadorUsuarios := 0;
          usuarioActual := comunidadActual^.listaUsuarios;
          nodoUsuario := Format('usuario%d_%d', [contadorComunidad,
contadorUsuarios]);

          SL.Add(Format('  %s -> %s;', [nodoComunidad, nodoUsuario]));

          while usuarioActual <> nil do
          begin
            nodoUsuario := Format('usuario%d_%d', [contadorComunidad,
contadorUsuarios]);
            SL.Add(Format('  %s [label="%s"];',
              [nodoUsuario, EscapeDotString(usuarioActual^.correo)]));

            if usuarioActual^.next <> nil then
            begin
              nodoSig := Format('usuario%d_%d', [contadorComunidad,
```

```
contadorUsuarios + 1]);
              SL.Add(Format('  %s -> %s;', [nodoUsuario, nodoSig]));
            end;

            Inc(contadorUsuarios);
            usuarioActual := usuarioActual^.next;
          end;

          SL.Add('{rank=same; ' + nodoComunidad);
          for contadorUsuarios := 0 to contadorUsuarios - 1 do
            SL.Add(' ' + Format('usuario%d_%d', [contadorComunidad,
contadorUsuarios]));
          SL.Add(' }');
        end;

        Inc(contadorComunidad);
        comunidadActual := comunidadActual^.next;
      end;
    end;

    SL.Add('}');
    SL.SaveToFile(DotFile);

    // Ejecuta Graphviz
    if FileExists('/usr/bin/dot') then
      RunCommand('/usr/bin/dot', ['-Tpng', DotFile, '-o', PNGFile], Output)
    else if FileExists('C:\Program Files\Graphviz\bin\dot.exe') then
      RunCommand('C:\Program Files\Graphviz\bin\dot.exe', ['-Tpng',
DotFile, '-o', PNGFile], Output)
    else
      ShowMessage('No se encontró Graphviz (dot). Instálalo o ajusta la
ruta.');

    ShowMessage('Reporte de Comunidades generado en: ' + PNGFile);
  finally
    SL.Free;
  end;
end;

function generateGrafComunidades(): string;
var
  DotFile, PNGFile: string;
begin
```

```
  DotFile := 'comunidades.dot';
  PNGFile := 'comunidades.png';
  GenerateDotComunidades(DotFile, PNGFile);
  Result := PNGFile;
end;

end.
```

uCrearComunidades

Para la implementación e integración de Comunidades al proyecto creamos la vista uCrearComunidades, esta se encarga de realizar tanto las validaciones como las respectivas creaciones de comunidades y agregar usuarios a estas.

```
unit uCrearComunidades;

{$mode delphi}

interface

uses
  Classes, SysUtils, Forms, Controls, Graphics, Dialogs, StdCtrls,
  Comunidades;

type
  { TfrmCrearComunidades }
  TfrmCrearComunidades = class(TForm)
    lblTitulo: TLabel;
    lblNombre: TLabel;
    lblComunidad: TLabel;
    lblCorreo: TLabel;
    txtNombre: TEdit;
    txtComunidad: TEdit;
    txtCorreo: TEdit;
    btnCrear: TButton;
    btnAgregar: TButton;
    btnCerrar: TButton;
    btnReporte: TButton;
    procedure btnCrearClick(Sender: TObject);
    procedure btnAgregarClick(Sender: TObject);
    procedure btnCerrarClick(Sender: TObject);
    procedure btnReporteClick(Sender: TObject);
```

```
  private
  public
  end;

var
  frmCrearComunidades: TfrmCrearComunidades;

implementation

{$R *.lfm}

procedure TfrmCrearComunidades.btnCrearClick(Sender: TObject);
var
  response: Boolean;
begin
  response := InsertarComunidad(txtNombre.Text);
  if response then
    ShowMessage('Comunidad creada correctamente.')
  else
    ShowMessage('Error: Ya existe una comunidad con ese nombre.');
end;

procedure TfrmCrearComunidades.btnAgregarClick(Sender: TObject);
var
  response: Boolean;
begin
  response := InsertarUsuario(txtComunidad.Text, txtCorreo.Text);
  if response then
    ShowMessage('Usuario agregado correctamente a la comunidad.')
  else
    ShowMessage('Error: Verifica que la comunidad o el usuario existan o
que el usuario ya esté en la comunidad.');
end;

procedure TfrmCrearComunidades.btnCerrarClick(Sender: TObject);
begin
  Close;
end;

procedure TfrmCrearComunidades.btnReporteClick(Sender: TObject);
var
  dotFile, pngFile: string;
  dotCode: string;
```

```
  SL: TStringList;
begin
  dotFile := 'reportes/comunidades.dot';
  pngFile := 'reportes/comunidades.png';

  // Generamos el DOT desde Comunidades
  dotCode := generateGrafComunidades();

  // Guardamos el DOT
  SL := TStringList.Create;
  try
    SL.Text := dotCode;
    if not DirectoryExists('reportes') then
      CreateDir('reportes');
    SL.SaveToFile(dotFile);
  finally
    SL.Free;
  end;

  // Ejecutamos Graphviz para crear el PNG
  if ExecuteProcess('dot', ['-Tpng', dotFile, '-o', pngFile], []) = 0 then
    ShowMessage('Reporte generado en: ' + pngFile)
  else
    ShowMessage('Error al generar el reporte de comunidades.');
end;

end.
```

EL unico inconveniente que se tuvo durante la integracion, quiza seria en como se recorria la lista en InsertarUsuario e InsertarComunidad, ya que en InsertarUsuario se recorria de una forma diferente a como tengo mis estructuras, por lo que tuve que realizar ese cambio, al igual que en InsertarComunidad.