Universidad ORT Uruguay
Facultad de Ingeniería

OBLIGATORIO – DOCUMENTO FINAL

Analista en Tecnologías de la Información – Analista Programador

Grupo N2G

Docente: Liliana Pino

Mariano Nuñez Nº 298013

Fabrizio Ferranty Nº 307494

Noviembre 2023

Índice

RESUMEN

¿En qué consiste el proyecto?

Este proyecto consiste en la creación de una aplicación de consola de redes sociales. La misma constará de usuarios, administradores y publicaciones.

Funcionalidades de la aplicación

Los usuarios pueden ser de tipo Miembro o Administrador.

Los Miembros puede solicitar a otro un vínculo de amistad, generando una invitación. Cuando el Miembro solicitado acepta la invitación, se establece un vinculo de amistad entre ambos. A su vez los Miembros son los únicos encargados de realizar Publicaciones, estas Publicaciones pueden ser de tipo Post o Comentarios. Los Post pueden ser públicos o privados y en cada Post debemos encontrar una lista de Comentarios.

Los Administradores pueden bloquear a los Miembros y en este caso el Miembro verá restringidas algunas funcionalidades. Además, los Administradores podrán censurar Comentarios dentro de un Post.

Los Post y Comentarios pueden recibir una Reacción por parte de los Miembros, la cual puede ser Like o Dislike. El Miembro solamente puede realizar una Reacción por Publicación. Además, las Publicaciones van a tener un valor de aceptación (VA) cuya formula varia dependiendo el tipo de Publicación.

SITIO WEB SOMEE

http://oblig.somee.com/

DIAGRAMA DE CLASES

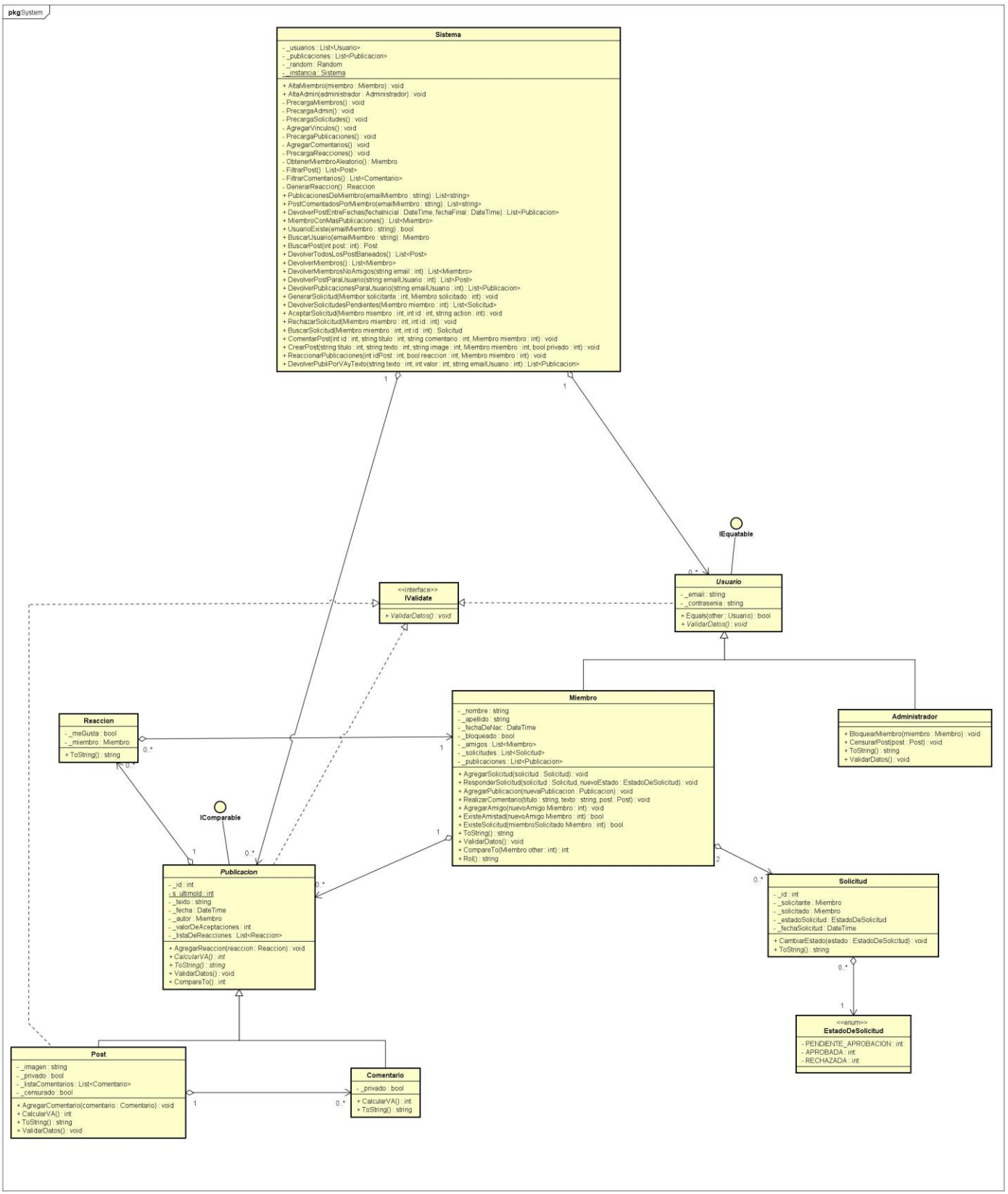
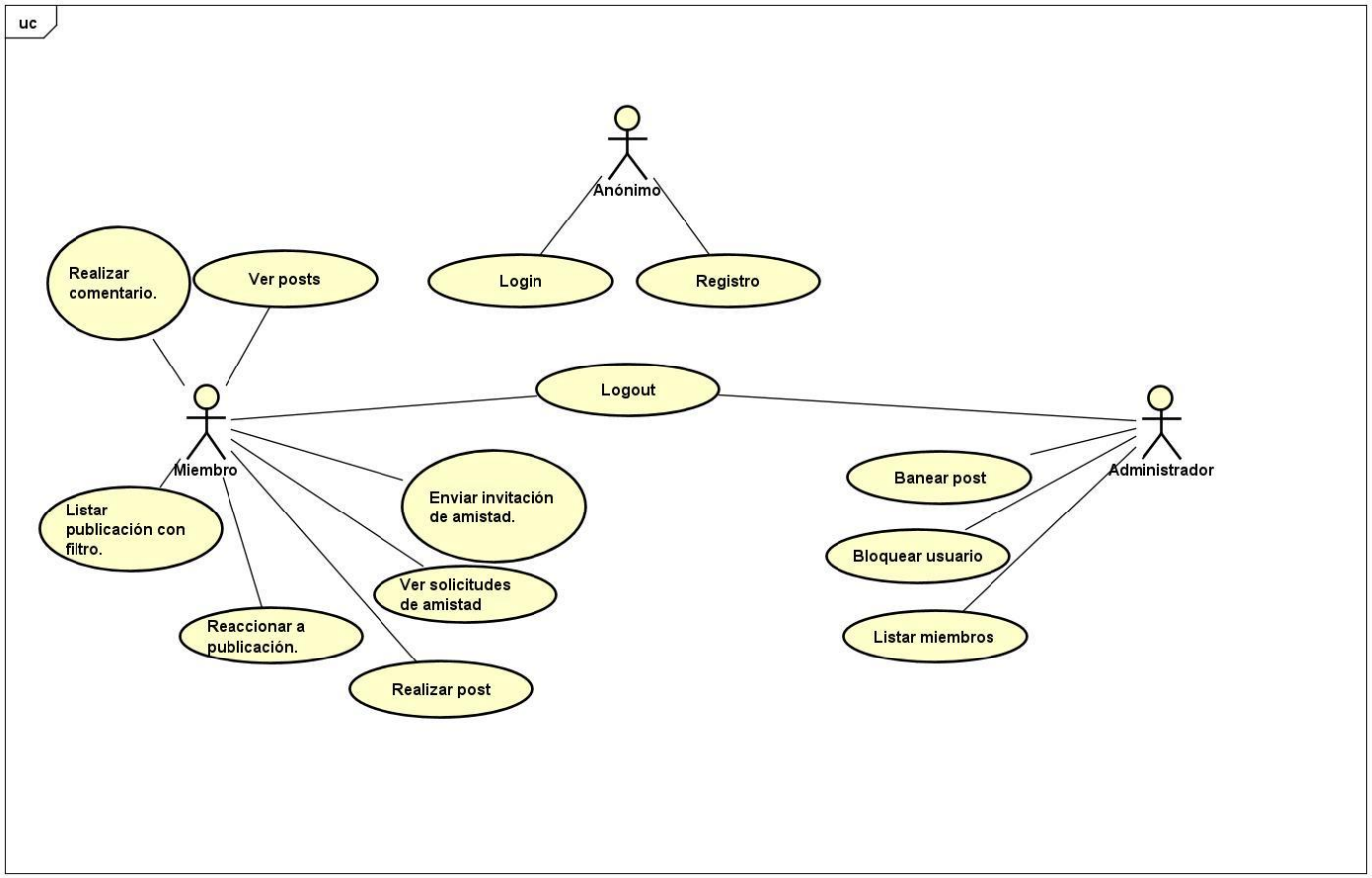


DIAGRAMA DE CASO DE USOS



PRECARGA DE DATOS

Precarga de Miembros

Email	Contraseña	Nombre	Apellido	Fecha de Nac.
Miembro_0	Contra_0	Nombre_0	Apellido_0	DateTime.Now
Miembro_1	Contra_1	Nombre_1	Apellido_1	DateTime.Now
Miembro_2	Contra_2	Nombre_2	Apellido_2	DateTime.Now
Miembro_3	Contra_3	Nombre_3	Apellido_3	DateTime.Now
Miembro_4	Contra_4	Nombre_4	Apellido_4	DateTime.Now
Miembro_5	Contra_5	Nombre_5	Apellido_5	DateTime.Now
Miembro_6	Contra_6	Nombre_6	Apellido_6	DateTime.Now
Miembro_7	Contra_7	Nombre_7	Apellido_7	DateTime.Now
Miembro_8	Contra_8	Nombre_8	Apellido_8	DateTime.Now
Miembro_9	Contra_9	Nombre_9	Apellido_9	DateTime.Now

Precarga de Administradores

Email	Contraseña
Admin	Contrasenia_admin

Solicitudes

Solicitante	Solicitado	Estado de Solicitud
Miembro_0	ObtenerMiembroAleatorio();	GenerarEstadoRandom();
Miembro_1	ObtenerMiembroAleatorio();	GenerarEstadoRandom();
Miembro_2	ObtenerMiembroAleatorio();	GenerarEstadoRandom();
Miembro_3	ObtenerMiembroAleatorio();	GenerarEstadoRandom();
Miembro_4	ObtenerMiembroAleatorio();	GenerarEstadoRandom();
Miembro_5	ObtenerMiembroAleatorio();	GenerarEstadoRandom();
Miembro_6	ObtenerMiembroAleatorio();	GenerarEstadoRandom();
Miembro_7	ObtenerMiembroAleatorio();	GenerarEstadoRandom();
Miembro_8	ObtenerMiembroAleatorio();	GenerarEstadoRandom();
Miembro_9	ObtenerMiembroAleatorio();	GenerarEstadoRandom();

NOTA: Se genera mediante dos métodos un Miembro aleatorio y Estado de Solicitud Aleatorio

Post

Titulo	Texto	Imagen	Miembro
Titulo_0	Publicacion_0	0_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_1	Publicacion_1	1_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_2	Publicacion_2	2_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_3	Publicacion_3	3_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_4	Publicacion_4	4_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_5	Publicacion_5	5_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_6	Publicacion_6	6_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_7	Publicacion_7	7_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_8	Publicacion_8	8_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_9	Publicacion_9	9_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_10	Publicacion_10	10_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_11	Publicacion_11	11_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_12	Publicacion_12	12_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_13	Publicacion_13	13_imagen.jpg	ObtenerMiembroAleatorio();
Titulo_14	Publicacion_14	14_imagen.jpg	ObtenerMiembroAleatorio();

Comentario

Titulo	Texto	Post	Miembro
Titulo_Comentario_0	Comentario_0	_publicaciones[i]	ObtenerMiembroAleatorio();
Titulo_Comentario_1	Comentario_1	_publicaciones[i]	ObtenerMiembroAleatorio();
Titulo_Comentario_2	Comentario_2	_publicaciones[i]	ObtenerMiembroAleatorio();

NOTA: En la solución se generan 3 comentarios por cada Post que se encuentre en la lista _publicaciones.

Reaccion

Like o Dislike	Miembro
True or False (Random)	ObtenerMiembroAleatorio();

NOTA: En la solución se generan 20 reacciones por Post y 20 Reacciones por Comentarios. Los datos se obtienen de forma randomica.

EVIDENCIA DE TESTING (ALTA MIEMBRO)

Escenario de Test	Pasos	Datos Utilizados	Resultado Esperado	Resultado Obtenido
Ingreso de Datos de Forma Correcta	1 – Ingreso un Nombre 2 – Ingreso un Apellido 3 – Ingreso la Fecha de Nacimiento 4 – Ingreso un Email 5 – Ingreso la Contraseña	Nombre: Agustin Apellido: Perez Fecha de Nacimiento: 01/09/2000 Email: agustin@email.com Contraseña: Hola123	Mensaje: Alta de Miembro Exitosa	Mensaje: Alta de Miembro Exitosa
Ingreso los Datos Vacíos	Avanzo hasta el final	Nombre: “” Apellido: “” Fecha de Nacimiento: “” Email: “” Contraseña: “”	Mensaje: Los Datos son Incorrectos, Vuelve a Ingresarlos	Mensaje: Los Datos son Incorrectos, Vuelve a Ingresarlos

CODIGO FUENTE

Sistema

```
using System.Collections.Generic;
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace Logica_De_Negocio
{
    public class Sistema
    {
        private List<Usuario> _usuarios = new List<Usuario>();
        private List<Publicacion> _publicaciones = new List<Publicacion>();
        private Random _random = new Random();
        private static Sistema _instancia;

        public Sistema()
        {
            PrecargaMiembros();
            PrecargaAdmin();
            PrecargaSolicitudes();
            AgregarVinculos();
            PrecargaPublicaciones();
            PrecargaReacciones();
        }

        public static Sistema Instancia
        {
            get
            {
                if (_instancia == null) _instancia = new Sistema();
                return _instancia;
            }
        }

        public void AltaMiembro(Miembro miembro)
        {
            miembro.ValidarDatos();

            if (!_usuarios.Contains(miembro))
            {
                _usuarios.Add(miembro);
            }
        }

        public void AltaAdmin(Administrador administrador)
        {
            administrador.ValidarDatos();

            if (!_usuarios.Contains(administrador))
            {
                _usuarios.Add(administrador);
            }
        }

        private void PrecargaMiembros()
        {
            for(int i = 0; i < 10; i++)
            {
                string email = "Miembro_" + i;
```

```
                string contrasenia = "Contra_" + i;
                string nombre = "Nombre_" + i;
                string apellido = "Apellido_" + i;
                DateTime fechaNac = DateTime.Now;

                Usuario miembroNuevo = new Miembro(email, contrasenia, nombre,
apellido, fechaNac);
                AltaMiembro((Miembro)miembroNuevo);
            }
        }

        private void PrecargaAdmin()
        {
            Usuario adminNuevo = new Administrador("Admin", "Admin_1");

            AltaAdmin((Administrador)adminNuevo);
        }

        // Metodo para precarga de solicitudes. Para cada miembro de la lista le
precarga una solicitud de amistad,
        // si la solicitud existe o la amistad existe la nueva solicitud NO se crea. A
su vez si la solicitud se crea con el Estado Aprobado
        // se crea tambien la amistad reciproca.
        private void PrecargaSolicitudes()
        {
            for (int i = 0; i < _usuarios.Count; i++)
            {
                if (_usuarios[i] is Miembro)
                {
                    Miembro miembroSolicitante = (Miembro)_usuarios[i];
                    Miembro miembroSolicitado;

                    do
                    {
                        miembroSolicitado = ObtenerMiembroAleatorio();
                    } while (miembroSolicitante == miembroSolicitado);



                    if(!miembroSolicitado.ExisteAmistad(miembroSolicitado) &&
!miembroSolicitado.ExisteSolicitud(miembroSolicitado))
                    {
                        EstadoDeSolicitud estadoRandom =
EstadoDeSolicitud.PENDIENTE_APROBACION;

                        Solicitud nuevaSolicitud = new Solicitud(miembroSolicitante,
miembroSolicitado, estadoRandom);

                        miembroSolicitado.AgregarSolicitud(nuevaSolicitud);

                        if (estadoRandom == EstadoDeSolicitud.APROBADA)
                        {
                            miembroSolicitado.AgregarAmigo(miembroSolicitante);
                            miembroSolicitante.AgregarAmigo(miembroSolicitado);

                        }
                    }
                }
            };
        }

        //Metodo para generar al menos dos vinculos de amistad de forma reciproca
        private void AgregarVinculos()
```

```
        {
            for (int i = 0; i < 2; i++)
            {
                Miembro miembroUno = ObtenerMiembroAleatorio();
                Miembro miembroDos;

                do
                {
                    miembroDos = ObtenerMiembroAleatorio();
                } while (miembroUno == miembroDos);

                miembroUno.AgregarAmigo(miembroDos);
                miembroDos.AgregarAmigo(miembroUno);
            }
        }

        //Precarga 15 Publicaciones y luego Precarga 3 Comentarios a cada una
        private void PrecargaPublicaciones()
        {
            for (int i = 0; i < 15; i++)
            {
                string titulo = "Titulo_" + i;
                string texto = "Publicacion_" + i;
                string imagen = i + "_imagen.jpg";
                bool censurado  = _random.Next(2) == 1;
                Miembro miembro = ObtenerMiembroAleatorio();

                Publicacion nuevaPublicacion = new Post(titulo, texto, miembro, imagen,
censurado);

                Post nuevoPost = (Post)nuevaPublicacion;

                nuevoPost.ValidarDatos();

                _publicaciones.Add(nuevaPublicacion);

                miembro.AgregarPublicacion(nuevaPublicacion);
            }

            AgregarComentarios();
        }

        private void AgregarComentarios()
        {
            List<Publicacion> nuevosComentarios = new List<Publicacion>();

            foreach (Publicacion post in _publicaciones)
            {
                if(post is Post)
                {
                    for(int i = 0;i < 3; i++)
                    {
                        string titulo = "Titulo_Comentario_" + i;
                        string texto = "Comentario_" + i;
                        Post post2 = (Post)post;
                        Miembro miembro = ObtenerMiembroAleatorio();


                        Publicacion nuevoComentario = new Comentario(titulo, texto,
miembro, post2);

                        nuevoComentario.ValidarDatos();
                        nuevosComentarios.Add(nuevoComentario);
```

```
                    miembro.AgregarPublicacion(nuevoComentario);
                    post2.AgregarComentario((Comentario)nuevoComentario);
                }
            }
        }

        _publicaciones.AddRange(nuevosComentarios);
    }

    //Precarga de Reacciones para Post y Comentarios
    private void PrecargaReacciones()
    {
        List<Post> posts = FiltrarPost();
        List<Comentario> comentarios = FiltrarComentarios();

        for (int i = 0; i < 20; i++)
        {
            int numRan = _random.Next(0, posts.Count);
            Reaccion nuevaReaccion = GenerarReaccion();

            posts[numRan].AgregarReaccion(nuevaReaccion);
        }

        for (int i = 0; i < 20; i++)
        {
            int numRan = _random.Next(0, comentarios.Count);
            Reaccion nuevaReaccion = GenerarReaccion();

            comentarios[numRan].AgregarReaccion(nuevaReaccion);
        }

    }

    private Miembro ObtenerMiembroAleatorio()
    {
        Miembro miembro = null;

        for (int i = 0; i < _usuarios.Count; i++)
        {
            int numRan = _random.Next(_usuarios.Count);

            if (_usuarios[numRan] is Miembro)
            {
                miembro = (Miembro)_usuarios[numRan];
                break;
            }
        }

        return miembro;
    }

    private List<Post> FiltrarPost()
    {
        List<Post> postList = new List<Post>();

        foreach (Publicacion post in _publicaciones)
        {
            if(post is Post) { postList.Add((Post)post); }
        }

        return postList;
    }
```

```
private List<Comentario> FiltrarComentarios()
{
    List<Comentario> comentList = new List<Comentario>();

    foreach (Publicacion coment in _publicaciones)
    {
        if (coment is Comentario) { comentList.Add((Comentario)coment); }
    }

    return comentList;
}

private Reaccion GenerarReaccion()
{
    bool meGusta = _random.Next(2) == 0;
    Miembro miembro = ObtenerMiembroAleatorio();

    Reaccion nuevaReaccion = new Reaccion(meGusta, miembro);

    return nuevaReaccion;
}

public bool UsuarioExiste(string emailMiembro)
{
    bool existe = false;

    if (BuscarUsuario(emailMiembro) != null)
    {
        existe = true;
    }

    return existe;
}

public Usuario? BuscarUsuario(string emailMiembro)
{
    Usuario? usuarioBuscado = null;

    foreach (Usuario usuario in _usuarios)
    {
        if (usuario.Email == emailMiembro)
        {
            usuarioBuscado = usuario;
        }
    }

    return usuarioBuscado;
}

public Publicacion BuscarPost(int id)
{
    Publicacion postBuscado = null;

    foreach (Publicacion publicacion in _publicaciones)
    {
        if (publicacion.Id == id)
        {
            postBuscado = publicacion;
        }
    }

    return postBuscado;
```

```
        }

        //Metodo creado para devolver todos los post que contiene la lista publicacion
no baneados
        public List<Post> DevolverTodosLosPostNoBaneados()
        {
            List<Post> listaDePost = new List<Post>();

            foreach (Publicacion publicacion in _publicaciones)
            {
                if(publicacion is Post post && !post.Censurado) listaDePost.Add(post);
            }

            return listaDePost;
        }

        public List<Miembro> DevolverMiembros()
        {
            List<Miembro> listaMiembros = new List<Miembro>();

            foreach (Usuario usuario in _usuarios)
            {
                if(usuario is Miembro miembro && !miembro.Bloqueado)
                {
                    listaMiembros.Add((Miembro)usuario);
                }
            }

            listaMiembros.Sort();

            return listaMiembros;
        }


        //Devuelvo los Miembros que no son mis amigos
        public List<Miembro> DevolverMiembrosNoAmigos(string email)
        {
            Miembro miembroSolicitante = (Miembro)BuscarUsuario(email);

            List<Miembro> listaMiembros = new List<Miembro>();

            foreach (Usuario usuario in _usuarios)
            {
                if (usuario is Miembro miembro && !miembro.Bloqueado &&
!miembroSolicitante.ExisteAmistad(miembro))
                {
                    listaMiembros.Add((Miembro)usuario);
                }
            }

            listaMiembros.Sort();

            return listaMiembros;
        }

        //Devuelve solamente los Post del Usuario
        public List<Post> DevolverPostParaUsuario(string emailUsuario)
        {
            List <Post> posts = new List<Post>();

            Miembro miembroUno = (Miembro)BuscarUsuario(emailUsuario);

            foreach (Publicacion publicacion in _publicaciones)
```

```
            {
                if (publicacion is Post post)
                {
                    if (post.Privado)
                    {
                        string emailAutor = post.Autor.Email;

                        Miembro autorPost = (Miembro)BuscarUsuario(emailAutor);

                        if (miembroUno.ExisteAmistad(autorPost))
                        {
                            posts.Add(post);
                        }
                    }
                    else
                    {
                        if (!post.Censurado)
                        {
                            posts.Add(post);
                        }
                    }
                }
            }
            return posts;
        }

        //Devuelve todas las publicaciones(Post y Comentario) del Usuario
        public List<Publicacion> DevolverPublicacionesParaUsuario(string emailUsuario)
        {
            List<Publicacion> publicaciones = new List<Publicacion>();

            List<Post> postsUsuario = DevolverPostParaUsuario(emailUsuario);

            foreach (Post post in postsUsuario)
            {
                publicaciones.Add(post);

                foreach (Comentario comentario in post.Comentarios)
                {
                    publicaciones.Add(comentario);
                }
            }

            return publicaciones;
        }

        public void GenerarSolicitud(Miembro solicitante, Miembro solicitado)
        {
            Solicitud nuevaSolicitud = new Solicitud(solicitante, solicitado,
EstadoDeSolicitud.PENDIENTE_APROBACION);

            solicitado.AgregarSolicitud(nuevaSolicitud);
        }

        public List<Solicitud> DevolverSolicitudesPendientes(Miembro miembro)
        {
            List<Solicitud> solicitudesPendientes = new List<Solicitud>();

            foreach(Solicitud solicitud in miembro.Solicitudes)
            {
                if(solicitud.EstadoDeSolicitud ==
EstadoDeSolicitud.PENDIENTE_APROBACION)
                {
```

```
                    solicitudesPendientes.Add(solicitud);
                }
            }

            return solicitudesPendientes;
        }

        public void AceptarSolicitud(Miembro miembro, int id, string action)
        {
            Solicitud solicitud = BuscarSolicitud(miembro, id);

            solicitud.CambiarEstado(EstadoDeSolicitud.APROBADA);

            solicitud.Solicitante.AgregarAmigo(solicitud.Solicitado);
            solicitud.Solicitado.AgregarAmigo(solicitud.Solicitante);
        }

        public void RechazarSolicitud(Miembro miembro, int id, string action)
        {
            Solicitud solicitud = BuscarSolicitud(miembro, id);

            solicitud.CambiarEstado(EstadoDeSolicitud.RECHAZADA);
        }

        private Solicitud BuscarSolicitud(Miembro miembro, int id)
        {
            Solicitud? solicitudBuscada = null;

            foreach (Solicitud solicitud in miembro.Solicitudes)
            {
                if(solicitud.Id == id)
                {
                    solicitudBuscada = solicitud;
                }
            }

            return solicitudBuscada;
        }

        public void CrearPost(string titulo, string texto, string imagen, Miembro
miembro, bool privado)
        {
            Post nuevoPost = new Post(titulo,texto,miembro,imagen, privado);

            nuevoPost.ValidarDatos();

            _publicaciones.Add(nuevoPost);
        }

        public void ComentarPost(int id, string titulo, string comentario, Miembro
miembro)
        {
            Post post = (Post)BuscarPost(id);

            Comentario nuevoComentario = new Comentario(titulo,comentario,miembro,
post);

            nuevoComentario.ValidarDatos();

            post.AgregarComentario(nuevoComentario);
        }

        public void ReaccionarPublicacion(int idPost, bool reaccion, Miembro miembro)
```

```
        {
            Reaccion nuevaReaccion = new Reaccion(reaccion, miembro);

            Publicacion publicacion = BuscarPost(idPost);

            publicacion.AgregarReaccion(nuevaReaccion);
        }

        public List<Publicacion> DevolverPubliPorVAyTexto(string texto, int valor,
string emailUsuario)
        {
            List<Publicacion> publicacionesPorTextyVA = new List<Publicacion>();

            List<Publicacion> publicaciones =
DevolverPublicacionesParaUsuario(emailUsuario);

            foreach (Publicacion publicacion in publicaciones)
            {
                if (publicacion.CalcularVA() > valor &&
publicacion.Titulo.ToLower().Contains(texto.ToLower())) {
publicacionesPorTextyVA.Add(publicacion); }
            }

            return publicacionesPorTextyVA;
        }
    }
}
```

Usuario

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Logica_De_Negocio.Interfaces;

namespace Logica_De_Negocio
{
    public abstract class Usuario : IValidate, IEquatable<Usuario>
    {
        private string _email;
        private string _contrasenia;

        public Usuario(string email, string contrasenia)
        {
            this._email = email;
            this._contrasenia = contrasenia;
        }

        public string Email
        {
            get { return _email; }
        }

        public string Contrasenia
        {
            get { return _contrasenia; }
        }

        public bool Equals(Usuario? other)
```

```
        {
            return _email.Trim().ToLower() == other._email.Trim().ToLower();
        }

        public abstract void ValidarDatos();

        public abstract string Rol();
    }
}
```

Miembro

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Logica_De_Negocio
{
    public class Miembro : Usuario, IComparable<Miembro>
    {
        private string _nombre;
        private string _apellido;
        private DateTime _fechaNac;
        private bool _bloqueado;
        private List<Miembro> _amigos = new List<Miembro>();
        private List<Solicitud> _solicitudes = new List<Solicitud>();
        private List<Publicacion> _publicaciones = new List<Publicacion>();

        public Miembro(string email, string contrasenia,string nombre, string apellido,
DateTime fechaNac) : base (email, contrasenia)
        {
            this._nombre = nombre;
            this._apellido = apellido;
            this._fechaNac = fechaNac;
            this._bloqueado = false;
        }

        public string Nombre
        {
            get { return _nombre; }
        }

        public string Apellido
        {
            get { return _apellido; }
        }

        public DateTime FechaNac
        {
            get { return _fechaNac; }
        }

        public bool Bloqueado
        {
            get { return _bloqueado; }
        }
```

```
        public List<Miembro> Amigos
        {
            get { return _amigos; }
        }

        public List<Solicitud> Solicitudes
        {
            get { return _solicitudes; }
        }

        public List<Publicacion> Publicaciones
        {
            get { return _publicaciones; }
        }

        public void AgregarSolicitud(Solicitud solicitud)
        {
            _solicitudes.Add(solicitud);
        }

        public void ResponderSolicitud(Solicitud solicitud, EstadoDeSolicitud
nuevoEstado)
        {
            solicitud.CambiarEstado(nuevoEstado);

            if(nuevoEstado == EstadoDeSolicitud.APROBADA)
            {
                AgregarAmigo(solicitud.Solicitante);
            }
        }

        public void AgregarPublicacion(Publicacion nuevaPublicacion)
        {
            _publicaciones.Add(nuevaPublicacion);
        }

        public void RealizarComentario(string titulo, string texto, Post post)
        {
            Publicacion nuevaPublicacion = new Comentario(titulo, texto, this, post);

            _publicaciones.Add(nuevaPublicacion);
        }

        public void AgregarAmigo(Miembro nuevoAmigo)
        {
            if (!ExisteAmistad(nuevoAmigo))
            {
                _amigos.Add(nuevoAmigo);
            }

        }

        public bool ExisteAmistad(Miembro nuevoAmigo)
        {
            bool existeAmistad = false;

            foreach (Miembro amigo in _amigos)
            {
                if(amigo == nuevoAmigo)
                {
                    existeAmistad = true;
                }
            }
```

```
            return existeAmistad;
        }

        public bool ExisteSolicitud(Miembro miembroSolicitado)
        {
            bool existeSolicitud = false;

            foreach (Solicitud solicitud in _solicitudes)
            {
                if (solicitud.Solicitante == miembroSolicitado)
                {
                    existeSolicitud = true;
                }
            }

            return existeSolicitud;
        }

        public void CambiarEstado(bool estado)
        {
            _bloqueado = estado;
        }

        public override string ToString()
        {
            string datos = $" Tipo: Miembro\n Email: {Email}\n Nombre: {_nombre}\n
Apellido: {_apellido}\n Fecha de Nacimiento: {_fechaNac}\n\n Lista de Amigos:\n";

            foreach (Miembro amigo in _amigos)
            {
                datos += $"\n {amigo._nombre} \n";
            }

            datos += "\n Solicitudes de Amistad: \n";

            foreach (Solicitud solicitud in _solicitudes)
            {
                datos += $"\n {solicitud} \n";
            }

            return datos;
        }

        public override void ValidarDatos()
        {
            if (Email.Trim().Length == 0)
            {
                throw new Exception("El Email No Puede Estar Vacio");
            }

            if (Contrasenia.Trim().Length == 0)
            {
                throw new Exception("La Contraseña No Puede Estar Vacia");
            }

            if (_nombre.Trim().Length == 0)
            {
                throw new Exception("El Nombre No Puede Estar Vacio");
            }

            if (_apellido.Trim().Length == 0)
            {
```

```
                throw new Exception("El Apellido No Puede Estar Vacio");
            }

            if (_fechaNac.Year > DateTime.Now.Year)
            {
                throw new Exception("La Fecha de Nacimiento es Incorrecto");
            }
        }

        public int CompareTo(Miembro? other)
        {
            int resultado = _apellido.CompareTo(other._apellido);

            if(resultado == 0 )
            {
                resultado = _nombre.CompareTo(other._nombre);
            }

            return resultado;
        }

        public override string Rol()
        {
            return "MIEMBRO";
        }
    }
}
```

Administrador

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Logica_De_Negocio
{
    public class Administrador : Usuario
    {
        public Administrador(string email, string contrasenia) : base(email,
contrasenia) { }

        public void BloquearMiembro(Miembro miembro){}

        public void CensurarPost(Post post){}

        public override string ToString()
        {
            return $" Tipo: Administrador\n email: {Email}";
        }
```

```
        public override void ValidarDatos()
        {
            if(Email.Trim().Length == 0)
            {
                throw new Exception("El Email No Puede Estar Vacio");
            }

            if (Contrasenia.Trim().Length == 0)
            {
                throw new Exception("La Contraseña No Puede Estar Vacia");
            }
        }

        public override string Rol()
        {
            return "ADMIN";
        }

    }
}
```

Solicitud

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Logica_De_Negocio
{
    public class Solicitud
    {
        private int _id;
        private Miembro _solicitante;
        private Miembro _solicitado;
        private EstadoDeSolicitud _estadoSolicitud;
        private DateTime _fechaSolicitud;

        public Solicitud(Miembro solicitante, Miembro solicitado, EstadoDeSolicitud
estadoSolicitud)
        {
            this._id = new Random().Next(1,1000);
            this._solicitante = solicitante;
            this._solicitado = solicitado;
            this._estadoSolicitud = estadoSolicitud;
            this._fechaSolicitud = DateTime.Now;
        }

        public int Id { get { return _id; } }

        public Miembro Solicitante { get { return _solicitante; } }

        public Miembro Solicitado { get { return _solicitado; } }

        public EstadoDeSolicitud EstadoDeSolicitud { get { return _estadoSolicitud; } }

        public void CambiarEstado(EstadoDeSolicitud estado)
        {
```

```
            this._estadoSolicitud = estado;
        }

        public override string ToString()
        {
            return $"Solicitud Id: " + _id;
        }
    }
}
```

Estado de Solicitud

```
namespace Logica_De_Negocio
{
  public enum EstadoDeSolicitud
  {
    PENDIENTE_APROBACION,
    APROBADA,
    RECHAZADA
  }
}
```

Publicacion

```
using Logica_De_Negocio.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Logica_De_Negocio
{
    public abstract class Publicacion : IValidate, IComparable<Publicacion>
    {
        private int _id;
        private static int s_ultimoId;
        private string _titulo;
        private string _texto;
        private DateTime _fecha;
        private Miembro _autor;
        private int _valorDeAceptaciones;
        private List<Reaccion> _reacciones = new List<Reaccion>();

        public Publicacion(string titulo, string texto, Miembro miembro)
        {
            s_ultimoId++;
            this._id = s_ultimoId;
            this._titulo = titulo;
            this._texto = texto;
            this._autor = miembro;
            this._fecha = DateTime.Now;
        }

        public int Id { get { return _id; } }
```

```
public DateTime DateTime { get { return _fecha; } }

public string Titulo { get { return _titulo; } }

public string Texto { get { return _texto; } }

public Miembro Autor { get { return _autor; } }

public List<Reaccion> Reacciones
{
    get { return _reacciones; }
}

public void AgregarReaccion(Reaccion reaccion)
{
    _reacciones.Add(reaccion);
}

public virtual int CantLike()
{
    int cantLike = 0;

    foreach (Reaccion reaccion in _reacciones)
    {
        if(reaccion.MeGusta) { cantLike++; }
    }

    return cantLike;
}

public virtual int CantDisLike()
{
    int cantDisLike = 0;

    foreach (Reaccion reaccion in _reacciones)
    {
        if (!reaccion.MeGusta) { cantDisLike++; }
    }

    return cantDisLike;
}

public virtual int CalcularVA()
{
    return (CantLike() * 5) + (CantDisLike() * (-2));
}

public abstract override string ToString();

public void ValidarDatos()
{
    if (_texto.Trim().Length ==0)
    {
        throw new Exception("El Texto No Puedo Estar Vacio");
    }

    if (_titulo.Trim().Length < 3)
    {
        throw new Exception("El Titulo debe Contener al Menos 3 Caracteres");
    }
}

public int CompareTo(Publicacion? other)
```

```
        {
            return _fecha.CompareTo(other._fecha) * -1;
        }
    }
}
```

Post

```
using Logica_De_Negocio.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Logica_De_Negocio
{
    public class Post : Publicacion, IValidate
    {
        private string _imagen;
        private bool _privado;
        private List<Comentario> _comentarios = new List<Comentario>();
        private bool _censurado;

        public Post(string titulo, string texto, Miembro miembro, string imagen, bool
privado) : base(titulo, texto, miembro)
        {
            this._imagen = imagen;
            this._privado = privado;
            this._censurado = false;
        }

        public bool Privado { get { return _privado; } }

        public bool Censurado { get { return _censurado; } }

        public List<Comentario> Comentarios { get {  return _comentarios; } }

        public void AgregarComentario(Comentario comentario)
        {
            _comentarios.Add(comentario);
        }

        public void CambiarEstado(bool estado)
        {
            _censurado = estado;
        }

        public override int CalcularVA()
        {
            int valorVA = base.CalcularVA();

            if (!_privado) { valorVA = valorVA +  10; }

            return valorVA;
        }

        public override string ToString()
        {
            string texto = Texto;
```

```
            if(texto.Length > 50) texto = Texto.Substring(0,50) + "...";

            return $" Id: {Id}\n Fecha: {DateTime}\n Titulo: {Titulo}\n Texto:
{texto}\n";
        }

        public void ValidarDatos()
        {
            base.ValidarDatos();

            if(_imagen.Trim().Length == 0)
            {
                throw new Exception("La Imagen No Puede Estar Vacia");
            }

            string extension = _imagen.Substring(_imagen.LastIndexOf("."));

            if (extension != ".jpg" && extension != ".png")
            {
                throw new Exception("La Extension Debe Ser .jpg o .png");
            }
        }
    }
}
```

Comentario

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Net.Mime.MediaTypeNames;

namespace Logica_De_Negocio
{
    public class Comentario : Publicacion
    {
        private Post _post;
        private bool _privado;

    public Comentario(string titulo, string texto, Miembro autor, Post post) :
base(titulo, texto, autor)
    {
        this._post = post;
        this._privado = post.Privado;
    }

    public Post Post
    {
        get { return this._post; }
    }

    public override string ToString()
        {
            string datos = $" Tipo: Comentario\n Post_Id: {_post.Id}\n\n -- Lista de
Reaciones --\n";

            foreach (Reaccion reaccion in Reacciones)
```

```
            {
                datos += $"\n -{reaccion}\n";
            }

            return datos;
        }
    }
}
```

Reaccion

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Logica_De_Negocio
{
    public class Reaccion
    {
        private bool _meGusta;
        private Miembro _miembro;

        public Reaccion(bool meGusta, Miembro miembro)
        {
            this._meGusta = meGusta;
            this._miembro = miembro;
        }

        public bool MeGusta
        {
            get { return _meGusta; }
        }

        public override string ToString()
        {
            return $" Reaccion\n Like: {_meGusta}\n Miembro: \n {_miembro.Nombre}";
        }
    }
}
```

IValidate <<Interface>>

```
namespace Logica_De_Negocio.Interfaces
{
  public interface IValidate
  {
    public void ValidarDatos();
  }
}
```