Universidad ORT Uruguay
Facultad de Ingeniería
Escuela de Tecnología

Obligatorio 2 - Programación 2

Rodrigo Suarez
303425

Pablo Dos Santos
287048

Grupo: N2D
Docente: Luis Dentone

Noviembre de 2023

Tabla de contenido

Diagrama de clases

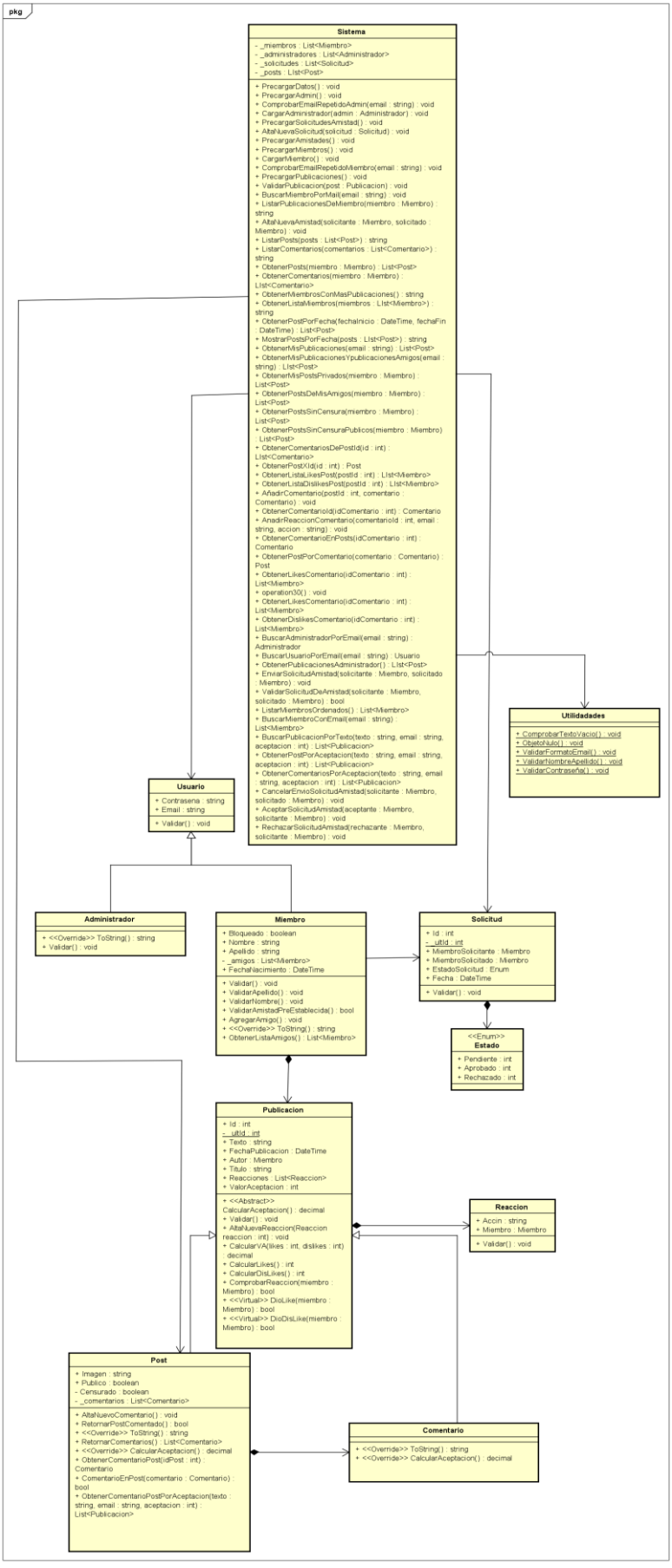
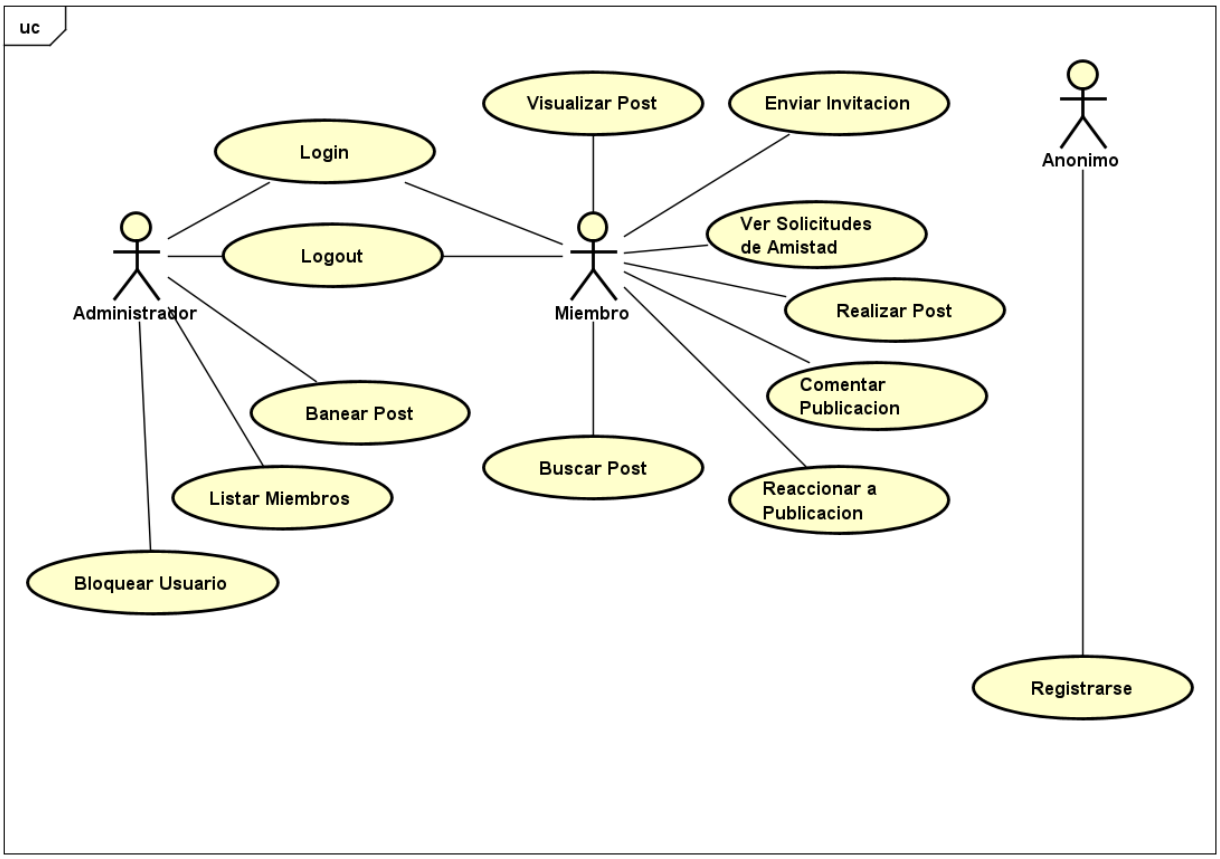


Diagrama de casos de uso



Datos precargados

Administradores:

Correo	Contraseña
admin1@algo.com	password1
admin2@algo.com	Password2

Miembros:

Correo	Contraseña	Nombre	Apellido	Fecha Nacimiento	Bloqueado
miembro1@algo.com	Password1	Nombre1	Apellido1	01/01/2022	false
miembro2@algo.com	Password2	Nombre2	Apellido2	02/02/2022	false
miembro3@algo.com	Password3	Nombre3	Apellido3	03/03/2022	false
miembro4@algo.com	Password4	Nombre4	Apellido4	04/04/2022	false
miembro5@algo.com	Password5	Nombre5	Apellido5	05/05/2022	false
miembro6@algo.com	Password6	Nombre6	Apellido6	06/06/2022	false
miembro7@algo.com	Password7	Nombre7	Apellido7	07/07/2022	false
miembro8@algo.com	Password8	Nombre8	Apellido8	08/08/2022	false
miembro9@algo.com	Password9	Nombre9	Apellido9	09/09/2022	false
miembro10@algo.com	Password10	Nombre10	Apellido10	10/10/2022	false
miembro11@algo.com	Password11	Nombre11	Apellido11	11/11/2022	false
miembro12@algo.com	Password12	Nombre12	Apellido12	12/12/2022	false

Solicitudes de amistad:

Solicitante	Solicitado	Estado	Fecha
Miembro 6	Miembro 9	Proceso	16/09/2023
Miembro 3	Miembro 5	Aceptado	16/09/2023
Miembro 11	Miembro 4	Proceso	16/09/2023
Miembro 4	Miembro 5	Rechazado	16/09/2023
Miembro 5	Miembro 7	Proceso	16/09/2023
Miembro 8	Miembro 9	Aceptado	16/09/2023
Miembro 3	Miembro 4	Proceso	16/09/2023
Miembro 9	Miembro 7	Rechazado	16/09/2023
Miembro 3	Miembro 10	Proceso	16/09/2023
Miembro 11	Miembro 6	Proceso	16/09/2023
Miembro 3	Miembro 9	Rechazado	16/09/2023
Miembro 9	Miembro 4	Aceptado	16/09/2023
Miembro 12	Miembro 4	Proceso	16/09/2023

Amistades:

Miembro	Miembro
Miembro 1	Miembro 2
Miembro 1	Miembro 3
Miembro 1	Miembro 4
Miembro 1	Miembro 5
Miembro 1	Miembro 6
Miembro 2	Miembro 7
Miembro 2	Miembro 8
Miembro 2	Miembro 9
Miembro 2	Miembro 10
Miembro 2	Miembro 11
Miembro 2	Miembro 12

Publicaciones

Post:

Id	**Imagen**	**Censurado**	**Privado**	**Texto**	**Fecha**	**Título**	**Miembro**	**Valor de Aceptación**
1	Imagen1.jpg	False	False	Texto del post 1	14/09/2023	Título de post 1	Miembro 1	0
2	Imagen2.jpg	False	False	Texto del post 2	14/09/2023	Título de post 2	Miembro 1	0
3	Imagen3.jpg	False	False	Texto del post 3	14/09/2023	Título de post 3	Miembro 1	0
4	Imagen4.jpg	False	False	Texto del post 4	14/09/2023	Título de post 4	Miembro 1	0
5	Imagen5.jpg	False	False	Texto del post 5	14/09/2023	Título de post 5	Miembro 1	0

Comentarios:

Id	Texto	Fecha	Título	Miembro	Valor de Aceptación
6	Texto de comentario 1	01/09/2023	Título de comentario 1	Miembro 1	0
7	Texto de comentario 2	02/09/2023	Título de comentario 2	Miembro 1	0
8	Texto de comentario 3	03/09/2023	Título de comentario 3	Miembro 1	0
9	Texto de comentario 4	04/09/2023	Título de comentario 4	Miembro 1	0
10	Texto de comentario 5	05/09/2023	Título de comentario 5	Miembro 1	0
11	Texto de comentario 6	06/09/2023	Título de comentario 6	Miembro 2	0
12	Texto de comentario 7	07/09/2023	Título de comentario 7	Miembro 2	0
13	Texto de comentario 8	08/09/2023	Título de comentario 8	Miembro 2	0
14	Texto de comentario 9	09/09/2023	Título de comentario 9	Miembro 2	0
15	Texto de comentario 10	10/09/2023	Título de comentario 10	Miembro 2	0
16	Texto de comentario 11	11/09/2023	Título de comentario 11	Miembro 3	0
17	Texto de comentario 12	12/09/2023	Título de comentario 12	Miembro 3	0
18	Texto de comentario 13	13/09/2023	Título de comentario 13	Miembro 3	0
19	Texto de comentario 14	14/09/2023	Título de comentario 14	Miembro 3	0
20	Texto de comentario 15	15/09/2023	Título de comentario 15	Miembro 3	0
21	Texto de comentario 16	16/09/2023	Título de comentario 16	Miembro 4	0
22	Texto de comentario 17	17/09/2023	Título de comentario 17	Miembro 4	0
23	Texto de comentario 18	18/09/2023	Título de comentario 18	Miembro 4	0
24	Texto de comentario 19	19/09/2023	Título de comentario 19	Miembro 4	0

25	Texto de comentario 20	20/09/2023	Título de comentario 20	Miembro 4	0
26	Texto de comentario 21	21/09/2023	Título de comentario 21	Miembro 5	0
27	Texto de comentario 22	22/09/2023	Título de comentario 22	Miembro 5	0
28	Texto de comentario 23	23/09/2023	Título de comentario 23	Miembro 5	0
29	Texto de comentario 24	24/09/2023	Título de comentario 24	Miembro 5	0
30	Texto de comentario 25	25/09/2023	Título de comentario 25	Miembro 5	0

Evidencia de Testing

Registro de usuario:

ID	Escenario de test	Pasos	Datos utilizados	Resultado esperado	Resultado obtenido	Estado (F=Falla, P=pasa)
Test-01	Registro correcto de usuario.	1. Ingresa su nombre y apellido. 2. Ingresa email que cumple con formato solicitado. 3. Ingresa contraseña valida. 4. Ingresa fecha de nacimiento correcta, que cumple con formato solicitado.	Todos los campos.	Se registra correctamente.	Se muestra mensaje que indica registro correcto del miembro.	P
Test-02	Registro con uno o varios campos vacíos.	1. No ingresa alguno de los datos y confirma registro.	Todos los campos.	Aviso de que todos los campos son requeridos.	Mensaje de error avisando que ningún campo puede estar vacío, no registra.	P
Test-03	Registro con email ya existente.	1. Se registra con un email ya registrado para otro usuario.	Campo de email.	Aviso de que ya existe un usuario registrado con el email ingresado.	Mensaje de error avisando que ya existe un miembro registrado con el email ingresado, no registra.	P
Test-04	Registro con contraseña que no contiene al menos una mayúscula, una minúscula y un número.	1. Ingresa clave que no cumple con el formato.	Campo de contraseña.	Muestra mensaje error y avisa los caracteres que debe contener.	Mensaje de error, se avisa que la contraseña no cumple con los requisitos.	P
Test-05	Registro con fecha incorrecta	1. Ingresa una fecha que no cumple con el formato solicitado o no es correcta	Campo de fecha.	Muestra mensaje de error.	Mensaje de error, se indica que la fecha es incorrecta.	P

Ingreso de usuario:

ID	Escenario de test	Pasos	Datos utilizados	Resultado esperado	Resultado obtenido	Estado (F=Falla, P=pasa)
Test-01	Login correcto.	1. Ingresa usuario. 2. Ingresa contraseña. 3. Iniciar sesión.	Todos los campos.	Muestra header con funcionalidades para usuario logueado.	Loguea y muestra menú.	P
Test-02	Login con campos vacíos.	1. Ingresa usuario y no contraseña/ ingresa contraseña y no usuario. 2. Iniciar sesión.	Ningún campo o uno a la vez	Muestra mensaje de error.	No loguea.	P
Test-03	Datos no coinciden.	1. Ingresa usuario y contraseña que no coincidan entre sí. 2. Iniciar sesión.	Todos los campos.	Muestra mensaje error "Usuario o contraseña incorrecto".	No loguea.	P
Test-04	Usuario coincide, pero contraseña no respeta sus mayúsculas/ minús culas.	1. Ingresa usuario. 2. Ingresa contraseña. 3. iniciar sesión.	Todos los campos.	Muestra mensaje error "Usuario o contraseña incorrecto".	No loguea.	P

Listar miembros:

ID	Escenario de test	Pasos	Datos utilizados	Resultado esperado	Resultado obtenido	Estado (F=Falla, P=pasa)
Test-01	El administrador logueado clickea botón "Administrar Miembros"	1. Clickea botón para administrar miembros.	Botón, credencial es del administr ador.	Se despliega listado de miembros.	Se muestran todos los usuarios del tipo miembro registrados en el sistema.	P

Bloquear o desbloquear un miembro:

ID	Escenario de test	Pasos	Datos utilizados	Resultado esperado	Resultado obtenido	Estado (F=Falla, P=pasa)
Test-01	El administrador logueado clickea botón "Administrar Miembros"	1. Clickea botón para administrar publicaciones. 2. Clickea botón de bloquear o desbloquear.	Botón, credencial es del administr ador.	El miembro queda bloqueado o desbloqueado dependiendo el estando en que se encuentra.	El botón pasa al estado opuesto al que se encontraba.	P

Banear un post:

ID	Escenario de test	Pasos	Datos utilizados	Resultado esperado	Resultado obtenido	Estado (F=Falla, P=pasa)
Test-01	Clickea botón "Administrar Publicaciones"	1. Clickea botón para administrar publicaciones.	Botón.	Muestra todas las publicaciones .	Información estadística.	P
Test-02	Sección visualizar estadística mayores y menores por departamento, no selecciona departamento.	1. No selecciona departamento. 2. Clickea botón "Mostrar resultados".	Combo desplegable con departamentos.	Muestra mensaje de error.	Muestra mensaje de error.	P
Test-03	Sección visualizar estadística mayores y menores por departamento, con departamento seleccionado.	1. Selecciona departamento. 2. Clickea botón "Mostar resultados".	Combo desplegable con departamentos.	Muestra tabla con información estadística.	Muestra información estadística.	P

Logout:

ID	Escenario de test	Pasos	Datos utilizados	Resultado esperado	Resultado obtenido	Estado (F=Falla, P=pasa)
Test-01	El usuario logueado clickea botón "Cerrar Sesión"	1. Clickea botón para cerrar sesión.	Botón.	Se muestra pantalla de inicio para usuario anónimo	Se muestra index para usuario anonimo.	P

Visualizar Publicaciones:

ID	Escenario de test	Pasos	Datos utilizados	Resultado esperado	Resultado obtenido	Estado (F=Falla, P=pasa)
Test-01	El miembro logueado clickea botón "Ver Publicaiones"	1. Clickea botón para ver publicaciones.	Botón, credencial es de miembro ingresado.	Se muestran sus publicaciones y las de sus amigos. Siempre que están no estén baneadas o sean privadas.	Se muestran publicaciones habilitadas.	P

Enviar Invitación:

ID	Escenario de test	Pasos	Datos utilizados	Resultado esperado	Resultado obtenido	Estado (F=Falla, P=pasa)
Test-01	El miembro logueado clickea botón "Gestionar solicitudes"	1. Clickea botón para gestionar solicitudes de amistad.	Botón, credencial es de miembro ingresado.	Se muestra el listado de miembros registrados y botón para enviar o cancelar solicitud.	Se muestra listado de miembros con sus botones correspondien tes.	P

Ver solicitudes amistad:

ID	Escenario de test	Pasos	Datos utilizados	Resultado esperado	Resultado obtenido	Estado (F=Falla, P=pasa)
Test-01	El miembro logueado clickea botón "Gestionar solicitudes"	1. Clickea botón para gestionar solicitudes de amistad.	Botón, credencial es de miembro ingresado.	Se muestra el listado de miembros registrados y botón para aceptar o rechazar solicitudes pendientes.	Se muestra listado de miembros con sus botones correspondien tes.	P

Realizar Post:

ID	Escenario de test	Pasos	Datos utilizados	Resultado esperado	Resultado obtenido	Estado (F=Falla, P=pasa)
Test-01	Clickea botón "Crear Post" un usuario bloqueado.	1. Clickea botón para crear un nuevo post.	Botón, credencial es de miembro ingresado.	Se muestra formulario para crear nuevo post con botón deshabilitado y mensaje informando que está bloqueado.	No puede realizar el post.	P
Test-02	Clickea botón "Crear Post" un usuario desbloqueado.	1. Clickea botón para crear un nuevo post.	Botón, credenciales de miembro ingresado.	Se muestra formulario para crear nuevo post y si todos los datos ingresados son correctos se crea.	Se crea un nuevo post, el miembro es redirigido a sus publicaciones donde ya puede verlo.	P

Realizar Comentario:

ID	Escenario de test	Pasos	Datos utilizados	Resultado esperado	Resultado obtenido	Estado (F=Falla, P=pasa)
Test-01	Clickea botón "Añadir Comentario" un usuario bloqueado.	1. Clickea botón para agregar un nuevo comentario.	Botón, credencial es de miembro ingresado.	El click en el botón no hace nada.	No puede realizar el post.	P
Test-02	Clickea botón "Añadir Comentario" un usuario desbloqueado.	1. Clickea botón para agregar un nuevo comentario.	Botón, credenciales de miembro ingresado.	Se muestra formulario para crear nuevo comentario y si todos los datos ingresados son correctos lo crea.	Se crea un nuevo comentario.	P

Reaccionar a Publicación:

ID	Escenario de test	Pasos	Datos utilizados	Resultado esperado	Resultado obtenido	Estado (F=Falla, P=pasa)
Test-01	Reaccionar a publicación	1. El miembro clickea en botón “Me gusta” o “No me gusta”	Botón, credenciales de miembro ingresado.	La reacción se guarda en la lista de reacciones de la publicación.	Se muestran nuevos valores de aceptación de la publicación.	P

Buscar Publicación:

ID	Escenario de test	Pasos	Datos utilizados	Resultado esperado	Resultado obtenido	Estado (F=Falla, P=pasa)
Test-01	Buscar publicación por valor de aceptación y texto.	2. El miembro clickea el botón “Buscar Publicacion”	Botón, credenciales de miembro ingresado.	Ver todas las publicaciones de tipo público, de mis amigos o propias con valor mayor al ingresado y texto que coincida con el ingresado	Se muestra listado de publicaciones correspondientes.	P

Código

Administrador.cs

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

namespace Dominio
{
    public class Administrador : Usuario, IValidable
    {
        ////Constructor

        public Administrador()
        {
        }

        public Administrador(string Pemail, string Pcontraseña) :
base(Pemail, Pcontraseña)
        {
            Email = Pemail;
            Contraseña = Pcontraseña;

        }
        //Validaciones
        public void Validar()
        {
            Utilidades.ComprobarTextoVaio(Email);
            Utilidades.ValidarFormatoEmail(Email);
            Utilidades.ComprobarTextoVaio(Contraseña);
            Utilidades.ValidarContraseña(Contraseña);
        }
        //Funcionalidades:
        public override string ToString()
        {
            return $"Email: {Email} \nContrasena: {Contraseña}";
        }

    }

}
```

Comentario.cs

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

namespace Dominio
{
    public class Comentario : Publicacion
    {
        public Comentario()
        {
        }
        public Comentario(string Ptexto, DateTime Pfecha, string Ptitulo,
Miembro unMiembro) : base(Ptexto, Pfecha, Ptitulo, unMiembro)
        {
            Texto = Ptexto;
            FechaPublicacion = Pfecha;
            Titulo = Ptitulo;
            Autor = unMiembro;
            ValorDeAceptacion = CalcularAceptacion();
        }


        public override string ToString()
        {
            return $"\nPublicacion: de tipo Comentario numero
{Id}\nTitulo: {Titulo}\nNombre: {Autor.Nombre} {Autor.Apellido}";
        }
        public override decimal CalcularAceptacion()
        {
            return CalcularVA(CalcularLikes(), CalcularDisLikes());
        }
    }
}
```

Miembro.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace Dominio
{
    public class Miembro:Usuario, IValidable, IComparable<Miembro>
    {
        //Atributos:
        public string Nombre { get; set; }
        public string Apellido { get; set; }
        public DateTime FechaNacimiento { get; set; }
        private List<Miembro> _amigos = new List<Miembro>();
        public List<Miembro> Amigos { get; }
        public bool Bloqueado { get; set; }
        //Constructor:
        public Miembro()
        {

        }
        public Miembro(string Pemail, string Pcontraseña, string Pnombre,
string Papellido, DateTime PfechaNacimiento, bool Pbloqueado
):base(Pemail, Pcontraseña)
        {
            Email = Pemail;
            Contraseña = Pcontraseña;
            Nombre = Pnombre;
            Apellido = Papellido;
            FechaNacimiento = PfechaNacimiento;
            Bloqueado = Pbloqueado;
            Validar();
        }
        public void Validar()
        {
            base.Validar();
            Utilidades.ComprobarTextoVaio(Nombre);
            ValidarNombre(Nombre);
            Utilidades.ComprobarTextoVaio(Nombre);
            ValidarApellido(Apellido);
        }
        public static void ValidarApellido(string apellido)
        {
            if (string.IsNullOrEmpty(apellido))
            {
                throw new Exception("El apellido no debe estar vacio");
            }
        }
        public static void ValidarNombre(string apellido)
        {
            if (string.IsNullOrEmpty(apellido))
            {
                throw new Exception("El apellido no debe estar vacio");
            }
        }
        public override string ToString()
        {
            return $"{Nombre} {Apellido}";
        }
```

```
        public void AgregarAmigo(Miembro NuevoAmigo)
        {
            _amigos.Add(NuevoAmigo);
        }
        public bool ValidarAmistadPreEstablecida(Miembro miembro)
        {
            foreach(Miembro amigo in _amigos)
            {
                if (amigo.Email == miembro.Email) return false;
            }
            return true;
        }
        public int CompareTo(Miembro miembro)
        {
            if (miembro == null)
            {
                return -1;
            }

            return Email.CompareTo(miembro.Email);
        }
        public List<Miembro> ObtenerListaAmigos()
        {
            return _amigos;
        }
    }
}
```

Post.cs

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

namespace Dominio
{
    public class Post : Publicacion, IValidable
    {
        //Atributos
        public string Imagen { get; set; }
        public bool Censurado { get; set; }
        public bool Privado { get; set; }
        private List<Comentario> _comentarios = new List<Comentario>();
        public List<Comentario> Comentarios { get { return _comentarios; }
}
        //Constructor
        public Post() { }
        public Post(string Pimagen, bool Pcensurado, bool Pprivado, string
Ptexto, DateTime Pfecha, string Ptitulo, Miembro unMiembro) : base(Ptexto,
Pfecha, Ptitulo, unMiembro)
        {
            Imagen = Pimagen;
            Censurado = Pcensurado;
            Privado = Pprivado;
            Validar();
        }
        public override string ToString()
        {
            return $"\nPublicacion: de tipo Post numero {Id}\nTitulo:
{Titulo}\nNombre: {Autor.Nombre} {Autor.Apellido}";
        }
        public void AltaNuevoComentario(Comentario Comentario)
        {
            _comentarios.Add(Comentario);
        }

        public bool RetornarPostComentado(string email)
        {
            foreach (Comentario comentario in _comentarios)
            {
                if (comentario.Autor.Email == email) return true;
            }
            return false;
        }

        public List<Comentario> RetornarComentarios(string email)
        {
            List<Comentario> comentarios = new List<Comentario>();
            foreach (Comentario comentario in _comentarios)
            {
                if (comentario.Autor.Email == email)
comentarios.Add(comentario);
            }
            return comentarios;
        }
        public List<Comentario> RetornarMisComentarios()
        {
            return _comentarios;
        }
        public override decimal CalcularAceptacion()
```

```
        {
            decimal ret = 0;
            decimal VA = CalcularVA(CalcularLikes(), CalcularDisLikes());
            ret = VA;
            if (!Privado) ret += 10;
            return ret;
        }
        public Comentario ObtenerComentarioPost(int idComentario)
        {
            foreach (Comentario comentario in _comentarios)
            {
                if (comentario.Id == idComentario)
                    return comentario;
            }
            return null;
        }
        public bool ComentarioEnPost(Comentario comentario)
        {
            if (ObtenerComentarioPost(comentario.Id) != null) return true;
            return false;
        }
        public List<Publicacion> ObtenerComentarioPostPorAceptacion(string
texto, string email, int aceptacion)
        {
            List<Publicacion> aux = new List<Publicacion>();
            foreach (Comentario comentario in _comentarios)
            {
                if (comentario.Texto.ToLower().Contains(texto.ToLower())
&&
                    comentario.CalcularAceptacion() > aceptacion)
                {
                    aux.Add(comentario);
                }
            }
            return aux;
        }
    }
}
```

Publicacion.cs

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

namespace Dominio
{
    public abstract class Publicacion: IValidable, IAutomaticId,
IComparable<Publicacion>
    {
        //Atributos:
        public int Id { get; set; }
        public static int _ultId = 1;
        public string Texto { get; set; }
        public DateTime FechaPublicacion { get; set; }
        public Miembro Autor { get; set; }
        public string Titulo { get; set; }
        public decimal ValorDeAceptacion { get; set; }
        public List<Reaccion> Reacciones { get; set; } = new
List<Reaccion>();

        //Constructor:

        public Publicacion()
        {

        }
        public Publicacion(string Ptexto, DateTime Pfecha, string Ptitulo,
Miembro unMiembro )
        {
            Id = _ultId++;
            Texto = Ptexto;
            FechaPublicacion = Pfecha;
            Titulo = Ptitulo;
            Autor = unMiembro;
            ValorDeAceptacion = CalcularAceptacion();
            Validar();
        }

        //Funcionalidades:

        public void Validar()
        {
            Utilidades.ComprobarTextoVaio(Texto);
            Utilidades.ComprobarTextoVaio(Titulo);
            Utilidades.ObjetoNulo(Autor);
        }

        public virtual void AltaNuevaReaccion(Reaccion Preaccion)
        {
            bool reacciono = ComprobarReaccion(Preaccion.Miembro);
            if(!reacciono)
            {
                Reacciones.Add(Preaccion);
            }
            if(reacciono)
            {
                foreach(Reaccion reaccion in Reacciones)
                {
```

```
                    if(reaccion.Miembro.Email== Preaccion.Miembro.Email &&
reaccion.Accion== Preaccion.Accion)
                    {
                        Reacciones.Remove(reaccion);
                    }
                    else if (reaccion.Miembro.Email ==
Preaccion.Miembro.Email && reaccion.Accion != Preaccion.Accion)
                    {
                        Reacciones.Remove(reaccion);
                        Reacciones.Add(Preaccion);
                    }
                }
            }
        }
        public int CompareTo(Publicacion obj)
        {
            if (obj == null) return -1;
            return FechaPublicacion.CompareTo(obj.FechaPublicacion)*-1;
        }
        public abstract decimal CalcularAceptacion();
        public decimal CalcularVA(int reaccionLike, int reaccionDislike)
        {
            return reaccionLike * 5 + reaccionDislike * (-2);
        }
        public int CalcularLikes()
        {
            int ret = 0;
            foreach(Reaccion reaccion in Reacciones)
            {
                if (reaccion.Accion.ToLower()== "like")ret++;
            }
            return ret;
        }
        public int CalcularDisLikes()
        {
            int ret = 0;
            foreach (Reaccion reaccion in Reacciones)
            {
                if (reaccion.Accion.ToLower() == "dislike") ret++;
            }
            return ret;
        }
        private bool ComprobarReaccion(Miembro miembro)
        {
            foreach(Reaccion reaccion in Reacciones)
            {
                if(miembro== reaccion.Miembro)return true;
            }
            return false;
        }
        public virtual bool DioLike(Miembro miembro)
        {
            foreach(Reaccion reaccion in Reacciones)
            {
                if(reaccion.Miembro.Email==  miembro.Email &&
reaccion.Accion.ToLower()== "like")return true;
            }
            return false;
        }
        public virtual bool DiodisLike(Miembro miembro)
        {
            foreach (Reaccion reaccion in Reacciones)
            {
                if (reaccion.Miembro.Email == miembro.Email &&
reaccion.Accion.ToLower() == "dislike") return true;
```

```
                }
                return false;
            }
        }
    }
```

Reacción.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Collections.Specialized.BitVector32;

namespace Dominio
{
    public class Reaccion: IValidable
    {
        //Atributos:
        public string Accion { get; set; }
        public Miembro Miembro { get; set; }

        //Constructor
        public Reaccion (string Paccion, Miembro Pmiembro)
        {
            Accion = Paccion;
            Miembro = Pmiembro;
            Validar();
        }

        public void Validar()
        {
            Utilidades.ComprobarTextoVaio(Accion);
            Utilidades.ObjetoNulo(Miembro);
            if (Accion.ToLower() != "like" && Accion.ToLower() !=
"dislike")
            {
                throw new Exception("Reaccion no valida");
            }
        }
    }
}
```

Sistema.cs

```
using System;
using System.Collections.Generic;
using System.Collections.Immutable;
using System.Linq;
using System.Runtime.Intrinsics.X86;
using System.Security.AccessControl;
using System.Text;
using System.Threading.Tasks;
using static Dominio.Solicitud;

namespace Dominio
{
    public class Sistema
    {
        private static Sistema _instancia;
        private List<Miembro> _miembros = new List<Miembro>();
        private List<Administrador> _administradores = new
List<Administrador>();
        private List<Solicitud> _solicitudes = new List<Solicitud>();
        private List<Post> _posts = new List<Post>();

        public List<Miembro> Miembros { get { return _miembros; } }
        public List<Solicitud> Solicitudes { get { return _solicitudes; }
}
        public List<Administrador> Administradores { get { return
_administradores; } }
        public List<Post> Publicaciones { get { return _posts; } }

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

        private Sistema()
        {
            PrecargarDatos();
        }

        public void PrecargarDatos()
        {
            PrecargarAdmin();
            PrecargarMiembros();
            PrecargarPublicaciones();
            PrecargarAmistades();
            PrecargarSolicitudesAmistad();
        }

        public void PrecargarAdmin()
        {
            Administrador unAdmin = new Administrador("admin1@algo.com",
"Password1");
            CargarAdministrador(unAdmin);
            Administrador unAdmin2 = new Administrador("admin2@algo.com",
"Password2");
            CargarAdministrador(unAdmin2);
        }

        public void ComprobarEmailRepetidoAdmin(string email)
```

```
        {
            foreach (Administrador admin in _administradores)
            {
                if (admin.Email == email)
                {
                    throw new Exception("El email ingresado ya fue
registrado por otro usuario");
                }
            }
        }

        public void CargarAdministrador(Administrador admin)
        {
            if (admin == null)
            {
                throw new Exception("El administrador no puede ser un
nulo.");
            }
            ComprobarEmailRepetidoAdmin(admin.Email);
            admin.Validar();
            _administradores.Add(admin);
        }
  .
private void PrecargarSolicitudesAmistad()
  {
      Solicitud Solicitud = new Solicitud(_miembros[5], _miembros[8],
Estado.Pendiente, new DateTime(2023, 09, 16));
      AltaNuevaSolicitud(Solicitud);
      Solicitud Solicitud2 = new Solicitud(_miembros[2], _miembros[4],
Estado.Aprobado, new DateTime(2023, 09, 16));
      AltaNuevaSolicitud(Solicitud2);
      Solicitud Solicitud3 = new Solicitud(_miembros[10], _miembros[3],
Estado.Pendiente, new DateTime(2023, 09, 16));
      AltaNuevaSolicitud(Solicitud3);
      Solicitud Solicitud4 = new Solicitud(_miembros[3], _miembros[4],
Estado.Rechazado, new DateTime(2023, 09, 16));
      AltaNuevaSolicitud(Solicitud4);
      Solicitud Solicitud5 = new Solicitud(_miembros[4], _miembros[6],
Estado.Pendiente, new DateTime(2023, 09, 16));
      AltaNuevaSolicitud(Solicitud5);
      Solicitud Solicitud6 = new Solicitud(_miembros[7], _miembros[8],
Estado.Aprobado, new DateTime(2023, 09, 16));
      AltaNuevaSolicitud(Solicitud6);
      Solicitud Solicitud7 = new Solicitud(_miembros[2], _miembros[3],
Estado.Pendiente, new DateTime(2023, 09, 16));
      AltaNuevaSolicitud(Solicitud7);
      Solicitud Solicitud8 = new Solicitud(_miembros[8], _miembros[6],
Estado.Rechazado, new DateTime(2023, 09, 16));
      AltaNuevaSolicitud(Solicitud8);
      Solicitud Solicitud9 = new Solicitud(_miembros[2], _miembros[9],
Estado.Pendiente, new DateTime(2023, 09, 16));
      AltaNuevaSolicitud(Solicitud9);
      Solicitud Solicitud10 = new Solicitud(_miembros[10], _miembros[5],
Estado.Pendiente, new DateTime(2023, 09, 16));
      AltaNuevaSolicitud(Solicitud10);
      Solicitud Solicitud11 = new Solicitud(_miembros[2], _miembros[8],
Estado.Rechazado, new DateTime(2023, 09, 16));
      AltaNuevaSolicitud(Solicitud11);
      Solicitud Solicitud12 = new Solicitud(_miembros[8], _miembros[3],
Estado.Aprobado, new DateTime(2023, 09, 16));
      AltaNuevaSolicitud(Solicitud12);
      Solicitud Solicitud13 = new Solicitud(_miembros[11], _miembros[3],
Estado.Pendiente, new DateTime(2023, 09, 16));
      AltaNuevaSolicitud(Solicitud13);
  }
```

```
  public void AltaNuevaSolicitud(Solicitud solicitud)
  {
      if (solicitud == null)
      {
          throw new Exception("La solicitud es un objeto nulo");
      }
      switch (solicitud.EstadoSolicitud)
      {
          case Estado.Pendiente:
          case Estado.Aprobado:
          case Estado.Rechazado:
              solicitud.Validar();
              if (solicitud.EstadoSolicitud == Estado.Aprobado)
              {
                  AltaNuevaAmistad(solicitud.Solicitante,
solicitud.Solicitado);
              }
              _solicitudes.Add(solicitud);
              break;

          default:
              throw new Exception("El estado es inválido");
      }
  }

  private void PrecargarAmistades()
  {
      //Amistad entre unMiembro1 hasta el 5
      AltaNuevaAmistad(_miembros[0], _miembros[1]);
      AltaNuevaAmistad(_miembros[0], _miembros[2]);
      AltaNuevaAmistad(_miembros[0], _miembros[3]);
      AltaNuevaAmistad(_miembros[0], _miembros[4]);
      AltaNuevaAmistad(_miembros[0], _miembros[5]);
      //Amistad entre unMiembro2 desde el 6 al 12
      AltaNuevaAmistad(_miembros[1], _miembros[6]);
      AltaNuevaAmistad(_miembros[1], _miembros[7]);
      AltaNuevaAmistad(_miembros[1], _miembros[8]);
      AltaNuevaAmistad(_miembros[1], _miembros[9]);
      AltaNuevaAmistad(_miembros[1], _miembros[10]);
      AltaNuevaAmistad(_miembros[1], _miembros[11]);
  }

  private void PrecargarMiembros()
  {
      Miembro unMiembro1 = new Miembro("miembro1@algo.com", "Password1",
"Nombre1", "Apellido1", new DateTime(2022, 01, 01), false);
      CargarMiembro(unMiembro1);
      Miembro unMiembro2 = new Miembro("miembro2@algo.com", "Password2",
"Nombre2", "Apellido2", new DateTime(2022, 02, 02), true);
      CargarMiembro(unMiembro2);
      Miembro unMiembro3 = new Miembro("miembro3@algo.com", "Password3",
"Nombre3", "Apellido3", new DateTime(2022, 03, 03), false);
      CargarMiembro(unMiembro3);
      Miembro unMiembro4 = new Miembro("miembro4@algo.com", "Password4",
"Nombre4", "Apellido4", new DateTime(2022, 04, 04), false);
      CargarMiembro(unMiembro4);
      Miembro unMiembro5 = new Miembro("miembro5@algo.com", "Password5",
"Nombre5", "Apellido5", new DateTime(2022, 05, 05), false);
      CargarMiembro(unMiembro5);
      Miembro unMiembro6 = new Miembro("miembro6@algo.com", "Password6",
"Nombre6", "Apellido6", new DateTime(2022, 06, 06), true);
      CargarMiembro(unMiembro6);
      Miembro unMiembro7 = new Miembro("miembro7@algo.com", "Password7",
"Nombre7", "Apellido7", new DateTime(2022, 07, 07), false);
```

```
        CargarMiembro(unMiembro7);
        Miembro unMiembro8 = new Miembro("miembro8@algo.com", "Password8",
"Nombre8", "Apellido8", new DateTime(2022, 08, 08), false);
        CargarMiembro(unMiembro8);
        Miembro unMiembro9 = new Miembro("miembro9@algo.com", "Password9",
"Nombre9", "Apellido9", new DateTime(2022, 09, 09), false);
        CargarMiembro(unMiembro9);
        Miembro unMiembro10 = new Miembro("miembro10@algo.com",
"Password10", "Nombre10", "Apellido10", new DateTime(2022, 10, 10), true);
        CargarMiembro(unMiembro10);
        Miembro unMiembro11 = new Miembro("miembro11@algo.com",
"Password11", "Nombre11", "Apellido11", new DateTime(2022, 11, 11),
false);
        CargarMiembro(unMiembro11);
        Miembro unMiembro12 = new Miembro("miembro12@algo.com",
"Password12", "Nombre12", "Apellido12", new DateTime(2022, 12, 12),
false);
        CargarMiembro(unMiembro12);
    }

    public void CargarMiembro(Miembro unMiembro)
    {
        if (unMiembro == null)
        {
            throw new Exception("El miembro no puede ser un nulo.");
        }
        unMiembro.Email = unMiembro.Email.ToLower();
        ComprobarEmailRepetidoMiembro(unMiembro.Email);
        unMiembro.Validar();
        _miembros.Add(unMiembro);
    }

    private void ComprobarEmailRepetidoMiembro(string email)
    {
        foreach (Miembro unMiembro in _miembros)
        {
            if (unMiembro.Email == email)
            {
                throw new Exception("El email ya fue registrado para otro
usuario");
            }
        }
    }
```

```
        public void PrecargarPublicaciones()
        {
            // Posts
            Post Post1 = new Post("Imagen1.jpg", false, false, "Texto del
post 1", new DateTime(2022, 9, 1), "Titulo de post 1", _miembros[0]);
            ValidarPublicacion(Post1);
            Post Post2 = new Post("Imagen2.jpg", true, false, "Texto del
post 2", new DateTime(2023, 9, 2), "Titulo de post 2", _miembros[0]);
            ValidarPublicacion(Post2);
            Post Post3 = new Post("Imagen3.jpg", false, false, "Texto del
post 3", new DateTime(2023, 9, 3), "Titulo de post 3", _miembros[1]);
            ValidarPublicacion(Post3);
            Post Post4 = new Post("Imagen4.jpg", false, false, "Texto del
post 4", new DateTime(2023, 9, 4), "Titulo de post 4", _miembros[1]);
            ValidarPublicacion(Post4);
            Post Post5 = new Post("Imagen5.jpg", false, false, "Texto del
post 5", new DateTime(2023, 9, 5), "Titulo de post 5", _miembros[3]);
            ValidarPublicacion(Post5);
            //Reacciones a Post:
            Reaccion reaccion1 = new Reaccion("Like", _miembros[0]);
            Reaccion reaccion2 = new Reaccion("Dislike", _miembros[0]);
            Reaccion reaccion3 = new Reaccion("Like", _miembros[1]);
            Reaccion reaccion4 = new Reaccion("Dislike", _miembros[1]);
            Reaccion reaccion5 = new Reaccion("Like", _miembros[2]);
            Post1.AltaNuevaReaccion(reaccion1);
            Post2.AltaNuevaReaccion(reaccion2);
            Post1.AltaNuevaReaccion(reaccion3);
            Post2.AltaNuevaReaccion(reaccion4);
            Post1.AltaNuevaReaccion(reaccion5);

            // Comentarios
            Comentario Comentario1 = new Comentario("Texto de comentario
1", new DateTime(2023, 9, 1), "Titulo de comentario 1", _miembros[0]);
            Comentario Comentario2 = new Comentario("Texto de comentario
2", new DateTime(2023, 9, 2), "Titulo de comentario 2", _miembros[0]);
            Comentario Comentario3 = new Comentario("Texto de comentario
3", new DateTime(2023, 9, 3), "Titulo de comentario 3", _miembros[0]);
            Comentario Comentario4 = new Comentario("Texto de comentario
4", new DateTime(2023, 9, 4), "Titulo de comentario 4", _miembros[0]);
            Comentario Comentario5 = new Comentario("Texto de comentario
5", new DateTime(2023, 9, 5), "Titulo de comentario 5", _miembros[0]);
            //
            Comentario Comentario6 = new Comentario("Texto de comentario
6", new DateTime(2023, 9, 6), "Titulo de comentario 6", _miembros[1]);
            Comentario Comentario7 = new Comentario("Texto de comentario
7", new DateTime(2023, 9, 7), "Titulo de comentario 7", _miembros[1]);
            Comentario Comentario8 = new Comentario("Texto de comentario
8", new DateTime(2023, 9, 8), "Titulo de comentario 8", _miembros[1]);
            Comentario Comentario9 = new Comentario("Texto de comentario
9", new DateTime(2023, 9, 9), "Titulo de comentario 9", _miembros[1]);
            Comentario Comentario10 = new Comentario("Texto de comentario
10", new DateTime(2023, 9, 10), "Titulo de comentario 10", _miembros[1]);
            //
            Comentario Comentario11 = new Comentario("Texto de comentario
11", new DateTime(2023, 9, 11), "Titulo de comentario 11", _miembros[2]);
            Comentario Comentario12 = new Comentario("Texto de comentario
12", new DateTime(2023, 9, 12), "Titulo de comentario 12", _miembros[2]);
            Comentario Comentario13 = new Comentario("Texto de comentario
13", new DateTime(2023, 9, 13), "Titulo de comentario 13", _miembros[2]);
            Comentario Comentario14 = new Comentario("Texto de comentario
14", new DateTime(2023, 9, 14), "Titulo de comentario 14", _miembros[2]);
```

```
            Comentario Comentario15 = new Comentario("Texto de comentario
15", new DateTime(2023, 9, 15), "Titulo de comentario 15", _miembros[2]);
            //
            Comentario Comentario16 = new Comentario("Texto de comentario
16", new DateTime(2023, 9, 16), "Titulo de comentario 16", _miembros[3]);
            Comentario Comentario17 = new Comentario("Texto de comentario
17", new DateTime(2023, 9, 17), "Titulo de comentario 17", _miembros[3]);
            Comentario Comentario18 = new Comentario("Texto de comentario
18", new DateTime(2023, 9, 18), "Titulo de comentario 18", _miembros[3]);
            Comentario Comentario19 = new Comentario("Texto de comentario
19", new DateTime(2023, 9, 19), "Titulo de comentario 19", _miembros[3]);
            Comentario Comentario20 = new Comentario("Texto de comentario
20", new DateTime(2023, 9, 20), "Titulo de comentario 20", _miembros[3]);
            //
            Comentario Comentario21 = new Comentario("Texto de comentario
21", new DateTime(2023, 9, 21), "Titulo de comentario 21", _miembros[4]);
            Comentario Comentario22 = new Comentario("Texto de comentario
22", new DateTime(2023, 9, 22), "Titulo de comentario 22", _miembros[4]);
            Comentario Comentario23 = new Comentario("Texto de comentario
23", new DateTime(2023, 9, 23), "Titulo de comentario 23", _miembros[4]);
            Comentario Comentario24 = new Comentario("Texto de comentario
24", new DateTime(2023, 9, 24), "Titulo de comentario 24", _miembros[4]);
            Comentario Comentario25 = new Comentario("Texto de comentario
25", new DateTime(2023, 9, 25), "Titulo de comentario 25", _miembros[4]);
            //Agregando comentarios a posts
            Post1.AltaNuevoComentario(Comentario1);
            Post1.AltaNuevoComentario(Comentario2);
            Post1.AltaNuevoComentario(Comentario3);
            Post1.AltaNuevoComentario(Comentario4);
            Post1.AltaNuevoComentario(Comentario5);
            //
            Post2.AltaNuevoComentario(Comentario6);
            Post2.AltaNuevoComentario(Comentario7);
            Post2.AltaNuevoComentario(Comentario8);
            Post2.AltaNuevoComentario(Comentario9);
            Post2.AltaNuevoComentario(Comentario10);
            //
            Post3.AltaNuevoComentario(Comentario11);
            Post3.AltaNuevoComentario(Comentario12);
            Post3.AltaNuevoComentario(Comentario13);
            Post3.AltaNuevoComentario(Comentario14);
            Post3.AltaNuevoComentario(Comentario15);
            //
            Post4.AltaNuevoComentario(Comentario16);
            Post4.AltaNuevoComentario(Comentario17);
            Post4.AltaNuevoComentario(Comentario18);
            Post4.AltaNuevoComentario(Comentario19);
            Post4.AltaNuevoComentario(Comentario20);
            //
            Post5.AltaNuevoComentario(Comentario21);
            Post5.AltaNuevoComentario(Comentario22);
            Post5.AltaNuevoComentario(Comentario23);
            Post5.AltaNuevoComentario(Comentario24);
            Post5.AltaNuevoComentario(Comentario25);
            //Reacciones a Comentarios:
            Reaccion reaccion6 = new Reaccion("Like", _miembros[0]);
            Comentario1.AltaNuevaReaccion(reaccion6);
            Reaccion reaccion7 = new Reaccion("Dislike", _miembros[1]);
            Comentario2.AltaNuevaReaccion(reaccion7);
            Reaccion reaccion8 = new Reaccion("Like", _miembros[0]);
            Comentario3.AltaNuevaReaccion(reaccion8);
            Reaccion reaccion9 = new Reaccion("Dislike", _miembros[1]);
            Comentario4.AltaNuevaReaccion(reaccion9);
            Reaccion reaccion10 = new Reaccion("Like", _miembros[2]);
            Comentario5.AltaNuevaReaccion(reaccion10);
```

```
        }

        public void ValidarPublicacion(Post Ppublicacion)
        {
            if (Ppublicacion == null)
            {
                throw new Exception("La publicacion no puede ser nula.");
            }
            if (string.IsNullOrEmpty(Ppublicacion.Titulo) ||
(Ppublicacion.Titulo).Length < 3)
            {
                throw new Exception("El titulo no puede ser vacio y debe
tener al menos 3 caracteres.");
            }
            if (Ppublicacion.Texto == string.Empty || Ppublicacion.Titulo
== "")
            {
                throw new Exception("El contenido de la publicacion no
puede ser vacio.");
            }
            _posts.Add(Ppublicacion);


        public Miembro? BuscarMiembroPorMail(string Pemail)
        {
            Utilidades.ValidarFormatoEmail(Pemail);
            foreach (Miembro unMiembro in _miembros)
            {
                if (unMiembro.Email == Pemail)
                {
                    return unMiembro;
                }
            }
            return null;
        }

        public string ListarPublicacionesDeMiembro(Miembro Pmiembro)
        {
            if (Pmiembro == null)
            {
                throw new Exception("El miembro no puede ser nulo.");
            }
            Pmiembro.Validar();
            string resultado = "";
            List<Post> posts = ObtenerPosts(Pmiembro);
            resultado += ListarPosts(posts);
            resultado += "\n";
            List<Comentario> Comentarios = ObtenerComentarios(Pmiembro);
            resultado += ListarComentarios(Comentarios);

            if (posts.Count < 1 && Comentarios.Count < 1)
            {
                throw new Exception("No se han encontrado publicaciones
del ususario indicado");
            }
            return resultado;
        }

        public void AltaNuevaAmistad(Miembro solicitante, Miembro
solicitado)
        {
            if (solicitante == null || solicitado == null) throw new
Exception("Uno de los dos miembros no existe");
            solicitante.Validar();
            solicitado.Validar();
```

```
            if (!solicitante.ValidarAmistadPreEstablecida(solicitado))
throw new Exception($"Ya hay una amistad entre usuario email
:{solicitante.Email} y {solicitado.Email}");
            solicitado.AgregarAmigo(solicitante);
            solicitante.AgregarAmigo(solicitado);
        }

        private string ListarPosts(List<Post> posts)
        {
            string ret = string.Empty;
            foreach (Post post in posts)
            {
                ret += post;
            }
            return ret;
        }

        private string ListarComentarios(List<Comentario> comentarios)
        {
            string respuesta = string.Empty;
            foreach (Comentario comentario in comentarios)
            {
                respuesta += comentario;
            }
            return respuesta;
        }

        public List<Post> ObtenerPosts(Miembro miembro)
        {
            List<Post> ret = new List<Post>();
            foreach (Post post in _posts)
            {
                if (post.Autor.Email == miembro.Email) ret.Add(post);
            }
            return ret;
        }

        public List<Comentario> ObtenerComentarios(Miembro miembro)
        {
            List<Comentario> comentarios = new List<Comentario>();
            foreach (Post post in _posts)
            {

comentarios.AddRange(post.RetornarComentarios(miembro.Email));
            }
            return comentarios;
        }

        public List<Post> ObtenerPostComentados(string email)
        {
            List<Post> posts = new List<Post>();
            foreach (Post post in _posts)
            {
                if (post.RetornarPostComentado(email)) posts.Add(post);
            }
            return posts;
        }

        public string ObtenerMiembrosConMasPublicaciones()
        {
            string ret = string.Empty;
            int masPublicaciones = 0;
            List<Miembro> miembros = new List<Miembro>();
            foreach (Miembro miembro in _miembros)
            {
```

```
                int numComentarios = ObtenerComentarios(miembro).Count;
                int numPost = ObtenerPosts(miembro).Count;
                int aux = numComentarios + numPost;
                if (aux == masPublicaciones) miembros.Add(miembro);
                if (aux > masPublicaciones)
                {
                    miembros.Clear();
                    miembros.Add(miembro);
                    masPublicaciones = aux;
                }
            }
            ret = ObtenerListaMiembros(miembros);
            return ret;
        }

        private string ObtenerListaMiembros(List<Miembro> miembros)
        {
            string ret = string.Empty;
            foreach (Miembro miembro in miembros)
            {
                ret += miembro + "\n";
            }
            return ret;
        }

        public List<Post> ObtenerPostPorFecha(DateTime fechaInicio,
DateTime fechaFin)
        {
            List<Post> ret = new List<Post>();
            foreach (Post post in _posts)
            {
                if (post.FechaPublicacion >= fechaInicio &&
post.FechaPublicacion <= fechaFin) ret.Add(post);
            }
            return ret;
        }


        public string MostrarPostsPorFecha(List<Post> posts)
        {
            List<Post> postOrdenados = posts.OrderByDescending(post =>
post.Titulo).ToList();
            string retorno = string.Empty;
            foreach (Post post in postOrdenados)
            {
                retorno += $"Id: {post.Id} \n";
                retorno += $"Fecha: {post.FechaPublicacion} \n";
                retorno += $"Titulo: {post.Titulo} \n";
                if (post.Texto.Length <= 50)
                {
                    retorno += $"Texto: {post.Texto} \n";
                }
                else
                {
                    retorno += $"Texto: {post.Texto.Substring(0,
50)}...\n";
                }
                retorno += "\n";
            }
            return retorno;
        }

        public List<Post> ObtenerMisPublicaciones(string email)
        {
            List<Post> aux = new List<Post>();
```

```
            Miembro miembro = BuscarMiembroPorMail(email);
            if (miembro == null) throw new Exception("El email buscado no
existe.");
            aux = ObtenerMisPublicacionesYpublicacionesAmigos(miembro);
            if (aux.Count == 0) throw new Exception("No se encontraron
posts");
            return aux;
        }
        public List<Post>
ObtenerMisPublicacionesYpublicacionesAmigos(Miembro miembro)
        {
            List<Post> aux = new List<Post>();
            aux = ObtenerPostsSinCensura(miembro);
            aux.AddRange(ObtenerPostsDeMisAmigos(miembro));
            aux.AddRange(ObtenerMisPostsPrivados(miembro));
            aux.Sort();
            return aux;
        }
        private List<Post> ObtenerMisPostsPrivados(Miembro miembro)
        {
            List<Post> aux = new List<Post>();
            foreach (Post post in _posts)
            {
                if (post != null && post.Autor.Email == miembro.Email &&
post.Privado) aux.Add(post);
            }
            return aux;
        }
        public List<Post> ObtenerPostsDeMisAmigos(Miembro miembro)
        {
            List<Post> aux = new List<Post>();
            List<Miembro> amigos = miembro.ObtenerListaAmigos();
            foreach (Miembro amigo in amigos)
            {
                aux.AddRange(ObtenerPostsSinCensuraPublicos(amigo));
            }
            return aux;
        }
        public List<Post> ObtenerPostsSinCensura(Miembro miembro)
        {
            List<Post> ret = new List<Post>();
            foreach (Post post in _posts)
            {
                if (post.Autor.Email == miembro.Email && !post.Censurado
&& !post.Privado) ret.Add(post);
            }
            return ret;
        }
        public List<Post> ObtenerPostsSinCensuraPublicos(Miembro miembro)
        {
            List<Post> ret = new List<Post>();
            foreach (Post post in _posts)
            {
                if (post.Autor.Email == miembro.Email && !post.Censurado
&& !post.Privado) ret.Add(post);
            }
            return ret;
        }
        public List<Comentario> ObtenerComentariosDePostId(int id)
        {
            List<Comentario> aux = new List<Comentario>();
            Post post = ObtenerPostXId(id);
            if (post == null) throw new Exception("El post buscado no se
encontro");
            aux.AddRange(post.RetornarMisComentarios());
```

```
            aux.Sort();
            return aux;
        }
        public Post ObtenerPostXId(int id)
        {
            foreach (Post post in _posts)
            {
                if (post.Id == id) return post;
            }
            return null;
        }
        public void AnadirReaccion(int postId, string email, string
accion)
        {
            Post post = ObtenerPostXId(postId);
            if (post == null) throw new Exception("El post buscado no
existe");
            Miembro miembro = BuscarMiembroPorMail(email);
            if (miembro == null) throw new Exception("El miembro buscado
no existe");
            Reaccion reaccion = new Reaccion(accion, miembro);
            post.AltaNuevaReaccion(reaccion);
        }
        public List<Miembro> ObtenerListaLikesPost(int postId)
        {
            List<Miembro> aux = new List<Miembro>();
            Post post = ObtenerPostXId(postId);
            foreach (Miembro miembro in _miembros)
            {
                if (post.DioLike(miembro)) aux.Add(miembro);
            }
            return aux;
        }
        public List<Miembro> ObtenerListaDisLikesPost(int postId)
        {
            List<Miembro> aux = new List<Miembro>();
            Post post = ObtenerPostXId(postId);
            foreach (Miembro miembro in _miembros)
            {
                if (post.DiodisLike(miembro)) aux.Add(miembro);
            }
            return aux;
        }

        public void AñadirComentario(int PostId, Comentario comentario)
        {
            Post post = ObtenerPostXId(PostId);
            if (post == null) throw new Exception("El post buscado no
existe");
            if (comentario == null) throw new Exception("El comentario no
puede ser nulo");
            post.AltaNuevoComentario(comentario);
        }
        public Comentario ObtenerComentarioId(int idComentario)
        {
            Comentario comentario = new Comentario();
            foreach (Post post in _posts)
            {
                if (post.ObtenerComentarioPost(idComentario) != null)
                {
                    comentario = post.ObtenerComentarioPost(idComentario);
                    return comentario;
                }
            }
```

```
            if (comentario == null) throw new Exception("El comentario
buscado no existe");
            return null;
        }
        public void AnadirReaccionComentario(int comentarioId, string
email, string accion)
        {
            Miembro miembro = BuscarMiembroPorMail(email);
            if (miembro == null) throw new Exception("El miembro buscado
no existe");
            Comentario comentario =
ObtenerComentarioEnPosts(comentarioId);
            if (comentario == null) throw new Exception("El comentario
buscado no existe");
            comentario.AltaNuevaReaccion(new Reaccion(accion, miembro));
        }
        public Comentario ObtenerComentarioEnPosts(int idComentario)
        {
            foreach (Post post in _posts)
            {
                Comentario comentario =
post.ObtenerComentarioPost(idComentario);
                if (comentario != null && comentario.Id == idComentario)
                {
                    return comentario;
                }
            }
            return null;
        }
        public Post ObtenerPostPorComentario(Comentario comentario)
        {
            if (comentario == null) throw new Exception("El comentario
buscado no existe;");
            foreach (Post post in _posts)
            {
                if (post.ComentarioEnPost(comentario)) return post;
            }
            return null;
        }
        public List<Miembro> ObtenerLikesComentario(int idComentario)
        {
            Comentario comentario =
ObtenerComentarioEnPosts(idComentario);
            List<Miembro> aux = new List<Miembro>();
            foreach (Miembro miembro in _miembros)
            {
                if (comentario.DioLike(miembro)) aux.Add(miembro);
            }
            return aux;
        }
        public List<Miembro> ObtenerdisLikesComentario(int idComentario)
        {
            Comentario comentario = ObtenerComentarioId(idComentario);
            List<Miembro> aux = new List<Miembro>();
            foreach (Miembro miembro in _miembros)
            {
                if (comentario.DiodisLike(miembro)) aux.Add(miembro);
            }
            return aux;
        }

        public Administrador BuscarAdministradorPorEmail(string email)
        {
            foreach (Administrador administrador in _administradores)
            {
```

```
                if (administrador.Email == email) return administrador;
            }
            return null;
        }

        public Usuario BuscarUsuarioPorEmail(string email)
        {
            Usuario usuario = BuscarMiembroPorMail(email);
            if (usuario == null) usuario =
BuscarAdministradorPorEmail(email);
            if (usuario == null) throw new Exception("El email ingresado
no se encontro");
            return usuario;
        }
        public List<Post> ObtenerPublicacionesAdministrador()
        {
            List<Post> list = _posts;
            list.Sort();
            return list;
        }

        public void EnviarSolicitudAmistad(Miembro solicitante, Miembro
solicitado)
        {
            if (solicitante == null || solicitado == null)
            {
                throw new Exception("Uno de los dos miembros no existe");
            }

            solicitante.Validar();
            solicitado.Validar();

            if (!solicitante.ValidarAmistadPreEstablecida(solicitado))
            {
                throw new Exception($"Ya hay una amistad entre usuario
email: {solicitante.Email} y {solicitado.Email}");
            }

            if (ValidarSolicitudDeAmistad(solicitante, solicitado))
            {
                throw new Exception($"Ya hay una solicitud enviada de
usuario email: {solicitante.Email} a {solicitado.Email}");
            }

            Solicitud nuevaSolicitud = new Solicitud(solicitante,
solicitado, Solicitud.Estado.Pendiente, DateTime.Now);
            AltaNuevaSolicitud(nuevaSolicitud);
        }

        public bool ValidarSolicitudDeAmistad(Miembro solicitante, Miembro
solicitado)
        {
            foreach (Solicitud solicitud in _solicitudes)
            {
                if ((solicitud.Solicitante.Email == solicitante.Email &&
solicitud.Solicitado.Email == solicitado.Email) ||
                    (solicitud.Solicitante.Email == solicitado.Email &&
solicitud.Solicitado.Email == solicitante.Email))
                {
                    return true;
                }
            }
            return false;
        }
        public List<Miembro> ListarMiembrosOrdenados()
```

```
        {
            List<Miembro> aux = _miembros;
            aux.Sort();
            return aux;
        }
        public List<Miembro> BuscarMiembroConEmail(string email)
        {
            List<Miembro> aux = new List<Miembro>();
            foreach (Miembro miembro in _miembros)
            {
                if (miembro.Email.ToLower().Contains(email.ToLower())) {
aux.Add(miembro); }
            }
            return aux;
        }

        public List<Publicacion> BuscarPublicacionPorTexto(string texto,
string email, int aceptacion)
        {
            List<Publicacion> aux = ObtenerPostPorAceptacion(texto, email,
aceptacion);
            aux.AddRange(ObtenerComentariosPorAceptacion(texto, email,
aceptacion));
            return aux;
        }
        public List<Publicacion> ObtenerPostPorAceptacion(string texto,
string email, int aceptacion)
        {
            List<Publicacion> aux = new List<Publicacion>();
            foreach (Post post in _posts)
            {
                if (post.Texto.ToLower().Contains(texto.ToLower()) &&
                    !post.Censurado &&
                    post.CalcularAceptacion() > aceptacion &&
                    !post.Privado)
                {
                    aux.Add(post);
                }
                if (post.Texto.ToLower().Contains(texto.ToLower()) &&
                    !post.Censurado &&
                    post.CalcularAceptacion() > aceptacion &&
                    !post.Privado &&
                post.Autor.Email == email
                    && post.Privado) aux.Add(post);
            }
            aux.Sort();
            return aux;
        }
        public List<Publicacion> ObtenerComentariosPorAceptacion(string
texto, string email, int aceptacion)
        {
            List<Publicacion> aux = new List<Publicacion>();
            foreach (Post post in _posts)
            {

aux.AddRange(post.ObtenerComentarioPostPorAceptacion(texto, email,
aceptacion));
            }
            aux.Sort();
            return aux;
        }

        public void CancelarEnvioSolicitudAmistad(Miembro solicitante,
Miembro solicitado)
        {
```

```
            if (solicitante == null || solicitado == null)
            {
                throw new Exception("Uno de los dos miembros no existe");
            }
            if (!solicitante.ValidarAmistadPreEstablecida(solicitado))
            {
                throw new Exception($"Ya hay una amistad entre usuario
email: {solicitante.Email} y {solicitado.Email}");
            }

            if (!ValidarSolicitudDeAmistad(solicitante, solicitado))
            {
                throw new Exception($"No hay una solicitud enviada de
usuario email: {solicitante.Email} a {solicitado.Email}");
            }

            foreach (Solicitud solicitud in _solicitudes)
            {
                if (solicitud.EstadoSolicitud is Estado.Pendiente &&
solicitud.Solicitante.Email == solicitante.Email &&
solicitud.Solicitado.Email == solicitado.Email)
                {
                    _solicitudes.Remove(solicitud);
                }
            }
        }

        public void AceptarSolicitudAmistad(Miembro aceptante, Miembro
solicitante)
        {
            if (aceptante == null || solicitante == null)
            {
                throw new Exception("Uno de los dos miembros no existe");
            }
            if (!aceptante.ValidarAmistadPreEstablecida(solicitante))
            {
                throw new Exception($"Ya hay una amistad entre usuario
email: {aceptante.Email} y {solicitante.Email}");
            }
            if (!ValidarSolicitudDeAmistad(aceptante, solicitante))
            {
                throw new Exception($"No hay una solicitud enviada de
usuario email: {aceptante.Email} a {solicitante.Email}");
            }
            foreach (Solicitud solicitud in _solicitudes)
            {
                if (solicitud.EstadoSolicitud is Estado.Pendiente &&
solicitud.Solicitante.Email == solicitante.Email &&
solicitud.Solicitado.Email == aceptante.Email)
                {
                    AltaNuevaAmistad(aceptante, solicitante);
                    solicitud.EstadoSolicitud = Estado.Aprobado;
                    break;
                }
            }
        }

        public void RechazarSolicitudAmistad(Miembro rechazante, Miembro
solicitante)
        {
            if (rechazante == null || solicitante == null)
            {
                throw new Exception("Uno de los dos miembros no existe");
            }
            if (!rechazante.ValidarAmistadPreEstablecida(solicitante))
```

```
            {
                throw new Exception($"Ya hay una amistad entre usuario
email: {rechazante.Email} y {solicitante.Email}");
            }
            if (!ValidarSolicitudDeAmistad(rechazante, solicitante))
            {
                throw new Exception($"No hay una solicitud enviada de
usuario email: {rechazante.Email} a {solicitante.Email}");
            }
            foreach (Solicitud solicitud in _solicitudes)
            {
                if (solicitud.EstadoSolicitud is Estado.Pendiente &&
solicitud.Solicitante.Email == solicitante.Email &&
solicitud.Solicitado.Email == rechazante.Email)
                {
                    solicitud.EstadoSolicitud = Estado.Rechazado;
                    _solicitudes.Remove(solicitud);
                }
            }
        }
    }
}
```

Solicitud.cs

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

namespace Dominio
{
    public class Solicitud : IValidable
    {
        // Atributos:
        public int Id { get; private set; }
        private static int _ultId = 1;
        public Miembro Solicitante { get; set; }
        public Miembro Solicitado { get; set; }
        public Estado EstadoSolicitud { get; set; }
        public DateTime Fecha { get; set; }

        public enum Estado
        {
            Pendiente,
            Aprobado,
            Rechazado
        }

        // Constructor:
        public Solicitud(Miembro Psolicitante, Miembro Psolicitado, Estado
Pestado, DateTime pFecha)
        {
            Id = _ultId++;
            Solicitante = Psolicitante;
            Solicitado = Psolicitado;
            EstadoSolicitud = Pestado;
            Fecha = pFecha;
            Validar();
        }


        public void Validar()
        {
            if (Solicitante == null || Solicitado == null) throw new
Exception("Uno de los miembros en la solicitud es un null");
            if (EstadoSolicitud is not Estado.Aprobado && EstadoSolicitud
is not Estado.Pendiente && EstadoSolicitud is not Estado.Rechazado)
            {
                throw new Exception("Hubo un probelma con el estado de la
solicitud");
            }
        }

    }
}
```

Usuario.cs

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

namespace Dominio
{
    public abstract class Usuario: IValidable, IEquatable<Miembro>
    {
        public string Email { get; set; }
        public string Contraseña { get; set; }
        //Constructor:

        public Usuario()
        {

        }

        public Usuario(string Pemail, string Pcontraseña)
        {
            Email = Pemail;
            Contraseña = Pcontraseña;
            Validar();
        }

        public void Validar()
        {
            Utilidades.ComprobarTextoVaio(Email);
            Utilidades.ValidarFormatoEmail(Email);
            Utilidades.ComprobarTextoVaio(Contraseña);
            Utilidades.ValidarContraseña(Contraseña);
        }
        public bool Equals(Miembro miembro)
        {
            return miembro != null && miembro.Email==Email;
        }


    }
}
```

Utilidades.cs

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

namespace Dominio
{
    public class Utilidades
    {
        public static void ComprobarTextoVaio(string texto)
        {
            if (string.IsNullOrEmpty(texto)) throw new Exception("No
pueden haber campos vacios");
        }
        public static void ObjetoNulo(Object? obj)
        {
            if(obj == null) throw new Exception("Se esperaba un objeto y
se recibio un nulo");
        }
        public static void ValidarFormatoEmail(string email)
        {
            int puntos = 0;
            int arroba = 0;
            for (int i = 0; i < email.Length; i++)
            {
                string unaLetra = email[i].ToString();
                if (unaLetra == ".") puntos++;
                if (unaLetra == "@") arroba++;
            }
            if (puntos == 0 || arroba != 1)
            {
                throw new Exception("Formato de email incorrecto");
            }
        }

        public static void ValidarNombreApellido(string nombreApellido)
        {
            if (string.IsNullOrWhiteSpace(nombreApellido))
            {
                throw new ArgumentException("El nombre o apellido no puede
estar vacío ni contener solo espacios en blanco.");
            }

            if (!nombreApellido.All(char.IsLetter))
            {
                throw new ArgumentException("Formato incorrecto, solo
puede contener letras.");
            }
        }

        public static void ValidarContraseña(string contraseña)
        {

            int Mayus = 0;
            int Num = 0;
            int Min = 0;
            for (int i = 0; i < contraseña.Length; i++)
            {
                char car = contraseña[i];
                if (char.IsUpper(car)) Mayus++;
```

```
                if (char.IsLower(car)) Min++;
                if (char.IsDigit(car)) Num++;
            }
            if (Mayus == 0 || Min == 0 || Num == 0)
            {
                throw new Exception("La contraseña debe contener al menos
una mayúscula, una minúscula y un número.");
            }
        }
    }
}
```

Publicación en Somee

http://obligatoriop2-rodrigo-pablo.somee.com