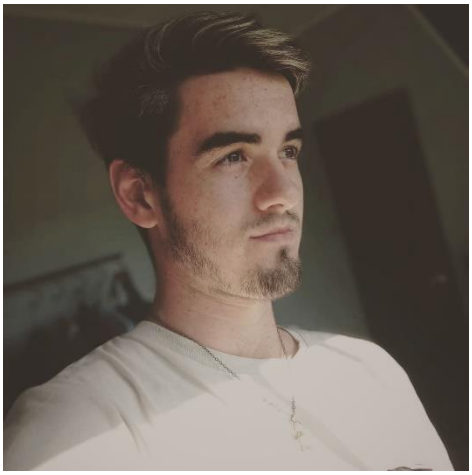
Universidad ORT Uruguay
Facultad de Ingeniería
Escuela de Tecnología

OBLIGATORIO PROGRAMACION 2

Grupo M2A

Agustín Butrico – 339579



Valentín Latorre – 339103

Profesora de la materia: Liliana Pino

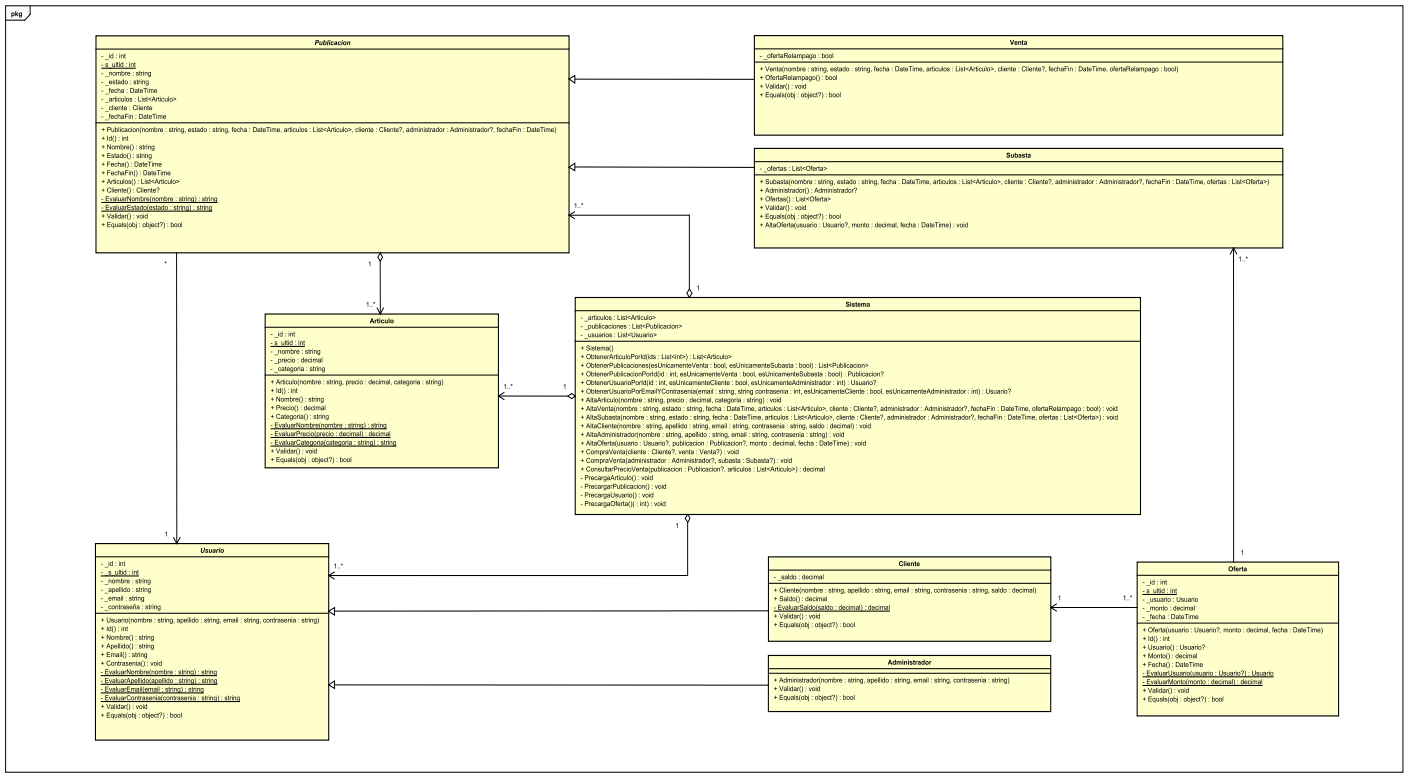
Enlace pagina Azure

Enlace Valentin –

https://paginaobligatorio2-g7gyfqbebheka4a6.canadacentral-01.azurewebsites.net

Enlace Agustin –

Diagrama de Clases completo del dominio del problema



Tablas de datos Precargados

PrecargarPublicacion()

AltaVenta("Verano en la playa", "ABIERTA", DateTime.ParseExact("05/01/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 11, 24, 35, 54 }), null, DateTime.MinValue, false);

AltaVenta("Juego gimnasio", "ABIERTA", DateTime.ParseExact("13/12/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 14, 15, 25, 26, 28, 38 }), null, DateTime.MinValue, false);

AltaVenta("Caminata en el bosque", "ABIERTA", DateTime.ParseExact("12/02/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 1, 3, 4, 5 }), null, DateTime.MinValue, false);

AltaVenta("Paseo en bicicleta", "ABIERTA", DateTime.ParseExact("15/03/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 6, 8, 9, 10 }), null, DateTime.MinValue, false);

AltaVenta("Clase de yoga", "ABIERTA", DateTime.ParseExact("22/04/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 12, 13, 16, 18, 20 }), null, DateTime.MinValue, false);

AltaVenta("Día de spa", "ABIERTA", DateTime.ParseExact("30/05/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 21, 22, 23, 29 }), null, DateTime.MinValue, true);

AltaVenta("Concierto al aire libre", "ABIERTA", DateTime.ParseExact("01/08/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 30, 31, 32, 34, 37 }), null, DateTime.MinValue, false);

AltaVenta("Cata de vinos", "ABIERTA", DateTime.ParseExact("10/09/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 40, 41, 42 }), null, DateTime.MinValue, false);

AltaVenta("Taller de pintura", "CERRADA", DateTime.ParseExact("15/10/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 43, 44, 45, 46 }), ObtenerUsuarioPorId(3, true, false) as Cliente, DateTime.ParseExact("05/11/2024", "dd/MM/yyyy", null), false);

AltaVenta("Excursión a la montaña", "CERRADA", DateTime.ParseExact("25/11/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 47, 48, 49 }), ObtenerUsuarioPorId(3, true, false) as Cliente, DateTime.ParseExact("26/11/2024", "dd/MM/yyyy", null), false);

AltaSubasta("Vuelta ciclista", "CERRADA", DateTime.ParseExact("06/01/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 27, 33, 39 }), ObtenerUsuarioPorId(8, true, false) as Cliente, ObtenerUsuarioPorId(1, false, true) as Administrador, DateTime.ParseExact("30/07/2024", "dd/MM/yyyy", null), new List<Oferta>());

AltaSubasta("Set camping", "ABIERTA", DateTime.ParseExact("21/07/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 7, 34 ,36 }), null, null, DateTime.MinValue, new List<Oferta>());

AltaSubasta("Torneo de ajedrez", "ABIERTA", DateTime.ParseExact("12/03/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 50, 51, 52 }), null, null, DateTime.MinValue, new List<Oferta>());

AltaSubasta("Subasta de arte", "ABIERTA", DateTime.ParseExact("20/04/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 51, 53, 54 }), null, null, DateTime.MinValue, new List<Oferta>());

AltaSubasta("Rally de coches", "ABIERTA", DateTime.ParseExact("01/06/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 36, 37, 38 }), null, null, DateTime.MinValue, new List<Oferta>());

AltaSubasta("Subasta de antigüedades", "ABIERTA", DateTime.ParseExact("15/07/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 29, 20, 21 }), null, null, DateTime.MinValue, new List<Oferta>());

AltaSubasta("Concurso de cocina", "ABIERTA", DateTime.ParseExact("05/08/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 42, 43, 44 }), null, null, DateTime.MinValue, new List<Oferta>());

AltaSubasta("Maratón de lectura", "ABIERTA", DateTime.ParseExact("12/09/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 45, 46, 47 }), null, null, DateTime.MinValue, new List<Oferta>());

AltaSubasta("Competencia de fotografía", "ABIERTA", DateTime.ParseExact("30/10/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 18, 19, 20 }), null, null, DateTime.MinValue, new List<Oferta>());

AltaSubasta("Fiesta de disfraces", "ABIERTA", DateTime.ParseExact("15/11/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 21, 22, 23 }), null, null, DateTime.MinValue, new List<Oferta>());

PrecargaUsuario()

```
AltaAdministrador("Valentin", "Latorre", "ValentinLatorre@Gmail.com", "Valentin1234");
AltaAdministrador("Agustin", "Butrico", "AgustinButrico@gmail.com", "Agustin1234");
AltaCliente("Juan", "Peres", "Juanperes@hmail.com", "Juan1234", 5600);
AltaCliente("Esteban", "Lopez", "EstebanLopez@hmail.com", "5566AS43", 27000);
AltaCliente("Carlos", "Medina", "CarlosMedina@hmail.com", "Medina1234", 7500);
AltaCliente("Mariano", "Morales", "MarianoMorales@hmail.com", "Mariano2", 5000);
AltaCliente("Estela", "Rosales", "EstelaRosales@hmail.com", "Rosalia46", 1700);
AltaCliente("Marcos", "Sauce", "MarcosSauce@hmail.com", "Sauce311", 30000);
AltaCliente("Lucia", "Gomez", "LuciaGomezs@hmail.com", "Lucia1990", 7200);
AltaCliente("Rodrigo", "Barrios", "RodrigoBarrios@hmail.com", "RodrigoBarrios12", 900);
AltaCliente("Pepe", "Argento", "PepeArgento@gmail.com", "PepeArgento1113", 3300);
AltaCliente("Felipe", "Castañeda", "FelipeCastañeda@gmail.com", "FeliCastañeda032", 3300);
```

PrecargaOferta()

```
AltaOferta(ObtenerUsuarioPorId(3, true, false), ObtenerPublicacionPorId(10, false, true), 120, DateTime.ParseExact("06/01/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(6, true, false), ObtenerPublicacionPorId(10, false, true), 1500, DateTime.ParseExact("24/07/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(4, true, false), ObtenerPublicacionPorId(10, false, true), 3400, DateTime.ParseExact("24/07/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(8, true, false), ObtenerPublicacionPorId(10, false, true), 3500, DateTime.ParseExact("24/07/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(8, true, false), ObtenerPublicacionPorId(11, false, true), 100, DateTime.ParseExact("24/07/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(5, true, false), ObtenerPublicacionPorId(11, false, true), 500, DateTime.ParseExact("21/07/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(3, true, false), ObtenerPublicacionPorId(11, false, true), 20000, DateTime.ParseExact("24/07/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(6, true, false), ObtenerPublicacionPorId(12, false, true), 200, DateTime.ParseExact("12/03/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(8, true, false), ObtenerPublicacionPorId(12, false, true), 400, DateTime.ParseExact("20/04/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(7, true, false), ObtenerPublicacionPorId(12, false, true), 700, DateTime.ParseExact("01/06/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(10, true, false), ObtenerPublicacionPorId(15, false, true), 300, DateTime.ParseExact("05/08/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(9, true, false), ObtenerPublicacionPorId(15, false, true), 600, DateTime.ParseExact("15/07/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(11, true, false), ObtenerPublicacionPorId(17, false, true), 450, DateTime.ParseExact("12/09/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(3, true, false), ObtenerPublicacionPorId(17, false, true), 1550, DateTime.ParseExact("30/10/2024",
"dd/MM/yyyy", null));
AltaOferta(ObtenerUsuarioPorId(4, true, false), ObtenerPublicacionPorId(17, false, true), 1600, DateTime.ParseExact("30/10/2024",
"dd/MM/yyyy", null));
```

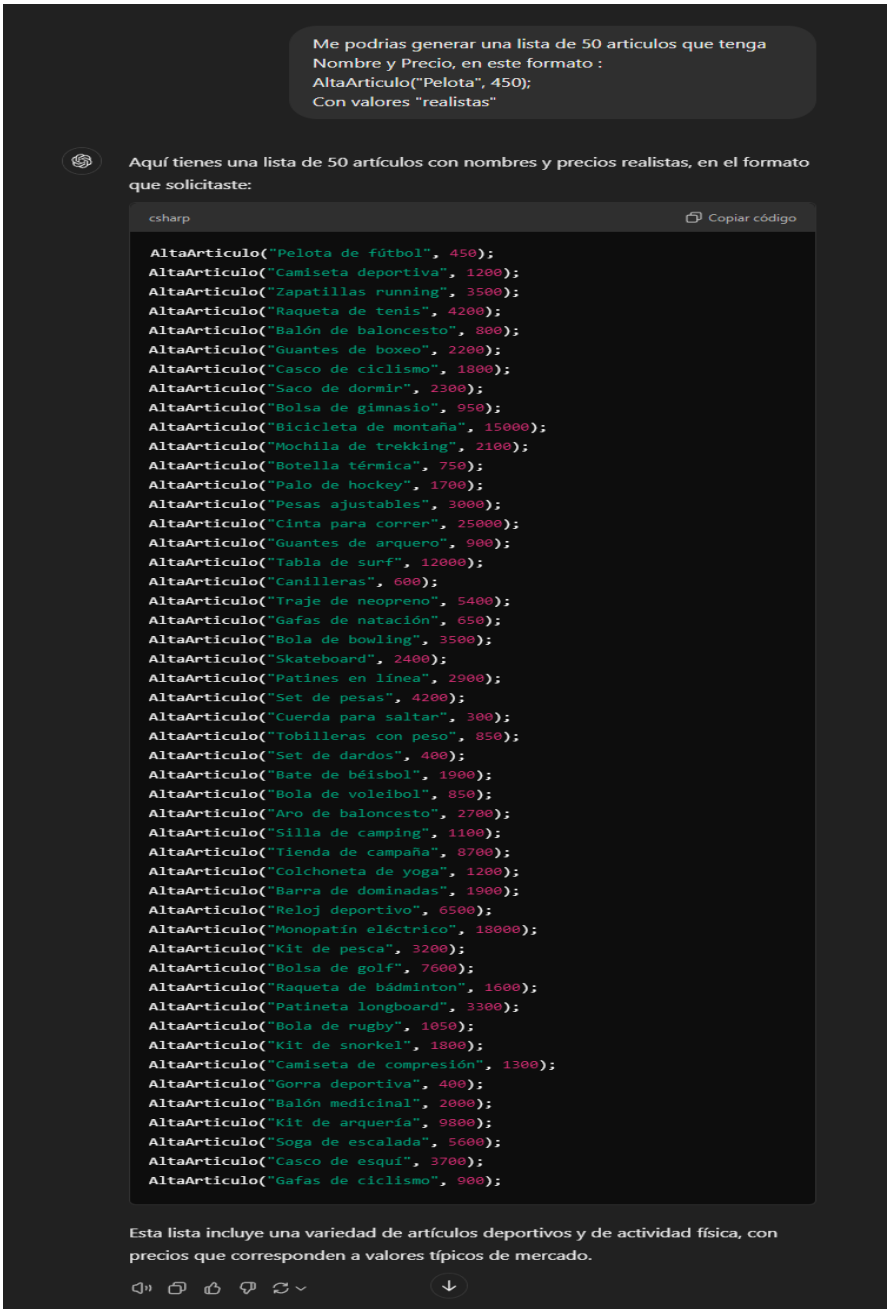
PrecargaArticulo()

```
AltaArticulo("Pelota de football", 450, "Football");
AltaArticulo("Camiseta deportiva", 1200, "Deporte");
AltaArticulo("Zapatillas treking", 3500, "Treking");
AltaArticulo("Raqueta de tenis", 4200, "Tenis");
AltaArticulo("Balón de basquetball", 800, "Basquetball");
AltaArticulo("Guantes de boxeo", 2200, "Boxeo");
AltaArticulo("Casco de ciclismo", 1800, "Ciclismo");
AltaArticulo("Saco de dormir", 2300, "Camping");
AltaArticulo("Bolsa de gimnasio", 950, "Boxeo");
AltaArticulo("Bicicleta de montaña", 15000, "Ciclismo");
AltaArticulo("Mochila de trekking", 2100, "Treking");
AltaArticulo("Protector solar", 320, "Playa");
AltaArticulo("Botella térmica", 750, "Camping");
AltaArticulo("Palo de hockey", 1700, "Hokey");
AltaArticulo("Pesas ajustables", 3000, "Gimnasio");
AltaArticulo("Cinta para correr", 25000, "Gimnasio");
AltaArticulo("Guantes de arquero", 900, "Arquería");
AltaArticulo("Tabla de surf", 12000, "Surf");
AltaArticulo("Canilleras", 600, "Football");
AltaArticulo("Traje de neopreno", 5400, "Surf");
AltaArticulo("Gafas de natación", 650, "Natación");
AltaArticulo("Bola de bowling", 3500, "Bowling");
AltaArticulo("Skateboard", 2400, "Skating");
AltaArticulo("Patines en línea", 2900, "Patinaaje");
AltaArticulo("Salvavidas", 1200, "Playa");
AltaArticulo("Set de pesas", 4200, "Gimnasio");
AltaArticulo("Cuerda para saltar", 300, "Gimnasio");
AltaArticulo("Bicicleta de carrera", 18500, "Ciclismo");
AltaArticulo("Tobilleras con peso", 850, "Gimnasio");
AltaArticulo("Set de dardos", 400, "Juegos");
AltaArticulo("Bate de baseball", 1900, "Baseball");
AltaArticulo("Bola de volleyball", 850, "Volleyball");
AltaArticulo("Aro de basquetball", 2700, "Basquetball");
AltaArticulo("Zapatilla de ciclismo", 1900, "Ciclismo");
AltaArticulo("Silla de camping", 1100, "Camping");
AltaArticulo("Sombrilla", 1600, "Playa");
AltaArticulo("Tienda de campaña", 8700, "Camping");
AltaArticulo("Colchoneta de yoga", 1200, "Deporte");
AltaArticulo("Barra de dominadas", 1900, "Gimnasio");
AltaArticulo("Malla", 600, "Ciclismo");
AltaArticulo("Reloj deportivo", 6500, "Deporte");
AltaArticulo("Monopatín eléctrico", 18000, "Ciclismo");
AltaArticulo("Kit de pesca", 3200, "Pesca");
AltaArticulo("Bolsa de golf", 7600, "Golf");
AltaArticulo("Raqueta de badminton", 1600, "Badminton");
AltaArticulo("Patineta longboard", 3300, "Skating");
AltaArticulo("Bola de rugby", 1050, "Rugby");
AltaArticulo("Kit de snorkel", 1800, "Natacion");
AltaArticulo("Camiseta de compresión", 1300, "Deporte");
AltaArticulo("Gorra deportiva", 400, "Deporte");
AltaArticulo("Balón medicinal", 2000, "Salud");
```

```
AltaArticulo("Kit de arquería", 9800, "Arquería");
AltaArticulo("Soga de escalada", 5600, "Escalada");
AltaArticulo("Casco de ski", 3700, "Ski");
AltaArticulo("Balde", 1050, "Playa");
AltaArticulo("Gafas de ciclismo", 900, "Ciclismo");
```

Consultas realizadas a ChatGPT

Para la generación de los 50 artículos



Para las ventas y las subastas

Aquí tienes 8 ventas y 8 subastas siguiendo el mismo formato que proporcionaste:

Ventas

csharp Copy code

```csharp
AltaVenta("Caminata en el bosque", "ABIERTA", DateTime.ParseExact("12/02/2024", "dd/MM/
AltaVenta("Paseo en bicicleta", "ABIERTA", DateTime.ParseExact("15/03/2024", "dd/MM/yyy
AltaVenta("Clase de yoga", "ABIERTA", DateTime.ParseExact("22/04/2024", "dd/MM/yyyy", n
AltaVenta("Día de spa", "ABIERTA", DateTime.ParseExact("30/05/2024", "dd/MM/yyyy", null
AltaVenta("Concierto al aire libre", "ABIERTA", DateTime.ParseExact("01/08/2024", "dd/M
AltaVenta("Cata de vinos", "ABIERTA", DateTime.ParseExact("10/09/2024", "dd/MM/yyyy", n
AltaVenta("Taller de pintura", "ABIERTA", DateTime.ParseExact("15/10/2024", "dd/MM/yyyy
AltaVenta("Excursión a la montaña", "ABIERTA", DateTime.ParseExact("25/11/2024", "dd/MM
```

Subastas

csharp Copy code

```csharp
AltaSubasta("Torneo de ajedrez", "ABIERTA", DateTime.ParseExact("12/03/2024", "dd/MM/yy
AltaSubasta("Subasta de arte", "ABIERTA", DateTime.ParseExact("20/04/2024", "dd/MM/yyyy
AltaSubasta("Rally de coches", "ABIERTA", DateTime.ParseExact("01/06/2024", "dd/MM/yyyy
AltaSubasta("Subasta de antigüedades", "ABIERTA", DateTime.ParseExact("15/07/2024", "dd
AltaSubasta("Concurso de cocina", "ABIERTA", DateTime.ParseExact("05/08/2024", "dd/MM/y
AltaSubasta("Maratón de lectura", "ABIERTA", DateTime.ParseExact("12/09/2024", "dd/MM/y
AltaSubasta("Competencia de fotografía", "ABIERTA", DateTime.ParseExact("30/10/2024", "
AltaSubasta("Fiesta de disfraces", "ABIERTA", DateTime.ParseExact("15/11/2024", "dd/MM/
```

Si necesitas más ajustes o variaciones, házmelo saber.

Message ChatGPT

INTERFAZUSUARIO (MVC)

Controlers

AccountControlller.cs

```
using LogicaNegocio;
using Microsoft.AspNetCore.Mvc;


namespace InterfazUsuario.Controllers
{
    public class AccountController : Controller
    {
        // Se llama a la instancia con patron singleton
        private Sistema sistema = Sistema.Instancia;


        [HttpGet]
        public IActionResult Login()
        {
            return View();
        }
        [HttpPost]
        public IActionResult Login(string email, string contrasenia)
        {
            try
            {
                // Manejo de errores
```

```
        if (string.IsNullOrEmpty(email) || string.IsNullOrEmpty(contrasenia))
        {
            ViewBag.Mensaje = "No pueden haber campos vacios";
        }
        else if (email.IndexOf('@') == -1)
        {
            ViewBag.Mensaje = "Debe incluir el dominio del email";
        }
        else if (contrasenia.Length < 8)
        {
            ViewBag.Mensaje = "La contrasenia debe ser de al menos 8 digitos";
        }
        else if (!contrasenia.Any(char.IsLetter))
        {
            ViewBag.Mensaje = "La contrasenia debe tener al menos una letra";
        }
        else if (!contrasenia.Any(char.IsDigit))
        {
            ViewBag.Mensaje = "La contrasenia debe tener al menos un dígito";
        }
        else
        {
            Usuario? usuario = sistema.ObtenerUsuarioPorEmailYContrasenia(email,
contrasenia, false, false);
            // Comprobar si el usuario existe
            if (usuario != null)
```

```
            {
                if (usuario is Cliente)
                {
                    HttpContext.Session.SetString("UserRole", "Cliente"); // Guardar el rol en
la sesión
                    HttpContext.Session.SetInt32("UserId", usuario.Id);  // Almacenar el Id del
usuario
                    return RedirectToAction("ListPublications", "Publications");
                }
                else
                {
                    HttpContext.Session.SetString("UserRole", "Administrador"); // Guardar el
rol en la sesión
                    HttpContext.Session.SetInt32("UserId", usuario.Id);  // Almacenar el Id del
usuario
                    return RedirectToAction("ListPublications", "Publications");
                }
            }
        }
    }
    catch (InvalidOperationException ex)
    {
        ViewBag.Mensaje = $"Error de operación: {ex.Message}";
    }
    catch (ArgumentNullException ex)
    {
        ViewBag.Mensaje = $"{ex.Message}";
```

```
        }
        catch (ArgumentException ex)
        {
            ViewBag.Mensaje = $"{ex.Message}";
        }
        catch (Exception ex)
        {
            ViewBag.Mensaje = $"Error inesperado: {ex.Message}";
        }

        // Si algo falla, permanece en la vista de login con el mensaje correspondiente
        return View();
    }

    [HttpGet]
    public IActionResult Register()
    {
        return View();
    }

    [HttpPost]
    public IActionResult Register(string nombre, string apellido, string email, string
contrasenia)
    {
        try
        {
```

```
        // Manejo de errores
        if (string.IsNullOrEmpty(nombre) || string.IsNullOrEmpty(apellido) ||
string.IsNullOrEmpty(email) || string.IsNullOrEmpty(contrasenia))
        {
           ViewBag.Mensaje = "No pueden haber campos vacios";
        }
        else if (email.IndexOf('@') == -1)
        {
           ViewBag.Mensaje = "Debe incluir el dominio del email";
        }
        else if (contrasenia.Length < 8)
        {
           ViewBag.Mensaje = "La contrasenia debe ser de al menos 8 digitos";
        }
        else if (!contrasenia.Any(char.IsLetter))
        {
           ViewBag.Mensaje = "La contrasenia debe tener al menos una letra";
        }
        else if (!contrasenia.Any(char.IsDigit))
        {
           ViewBag.Mensaje = "La contrasenia debe tener al menos un dígito";
        }
        else
        {
           // Crea el nuevo cliente si es que no existe previamente
           sistema.AltaCliente(nombre, apellido, email, contrasenia, 0);
```

```
            ViewBag.Confirmacion = "El usuario fue registrado correctamente";
        }
    }
    catch (InvalidOperationException ex)
    {
        ViewBag.Mensaje = $"Error de operación: {ex.Message}";
    }
    catch (ArgumentNullException ex)
    {
        ViewBag.Mensaje = $"{ex.Message}";
    }
    catch (ArgumentException ex)
    {
        ViewBag.Mensaje = $"{ex.Message}";
    }
    catch (Exception ex)
    {
        ViewBag.Mensaje = $"Error inesperado: {ex.Message}";
    }

    // Si algo falla, permanece en la vista de login con el mensaje correspondiente
    return View();
}

[HttpGet]
public IActionResult Logout()
```

```
        {
            // Limpiar la sesión
            HttpContext.Session.Clear();

            // Redirigir al login o a la página principal
            return RedirectToAction("Login", "Account");
        }
    }
}
```

AuctionsController.cs

```
using InterfazUsuario.Models;
using LogicaNegocio;
using Microsoft.AspNetCore.Mvc;

namespace InterfazUsuario.Controllers
{
    public class AuctionsController : Controller
    {
        // Se llama a la instancia con patron singleton
        private Sistema sistema = Sistema.Instancia;

        [HttpGet]
        public IActionResult ListAuctions()
        {
            if (HttpContext.Session.GetString("UserRole") != null)
```

```
        {
            // Almacena en una variable las subastas
            List<Publicacion> subastas = sistema.ObtenerPublicaciones(false, true);
            // Casteo explícito de subastas
            List<Subasta> listaSubastas = subastas.OfType<Subasta>().ToList();
            // Ordenar por fecha (ascendente)
            List<Subasta> listaSubastasOrdenada = listaSubastas.OrderBy(p =>
p.Fecha).ToList();

            // Crear el modelo con ambas listas
            var model = new ListAuctionsViewModel
            {
               Subastas = listaSubastasOrdenada
            };

            // Pasar el modelo a la vista
            return View(model);
        }
        return View();
      }
   }
}
```

HomeController.cs

```
using InterfazUsuario.Models;
using LogicaNegocio;
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;
```

```
namespace InterfazUsuario.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult IndexAdmin()
        {
            return View();
        }

        public IActionResult Privacy()
        {
            return View();
        }
```

```
    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore =
true)]
    public IActionResult Error()
    {
        return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
    }
  }
}
```

PublicationsControler.cs

```
using InterfazUsuario.Models;
using LogicaNegocio;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.CodeAnalysis.FlowAnalysis.DataFlow;
using
Microsoft.VisualStudio.Web.CodeGenerators.Mvc.Templates.BlazorIdentity.Pages.Manage;
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace InterfazUsuario.Controllers
{
  public class PublicationsController : Controller
  {
    // Se llama a la instancia con patron singleton
    private Sistema sistema = Sistema.Instancia;
```

```
[HttpGet]
public IActionResult ListPublications()
{
   if (HttpContext.Session.GetString("UserRole") != null)
   {
      // Almacena en una variable las ventas
      List<Publicacion> ventas = sistema.ObtenerPublicaciones(true, false);
      // Casteo explícito de ventas
      List<Venta> listaVentas = ventas.OfType<Venta>().ToList();
      // Ordenar por fecha (ascendente)
      List<Venta> listaVentasOrdenada = listaVentas.OrderBy(p => p.Fecha).ToList();

      // Almacena en una variable las subastas
      List<Publicacion> subastas = sistema.ObtenerPublicaciones(false, true);
      // Casteo explícito de subastas
      List<Subasta> listaSubastas = subastas.OfType<Subasta>().ToList();
      // Ordenar por fecha (ascendente)
      List<Subasta> listaSubastasOrdenada = listaSubastas.OrderBy(p =>
p.Fecha).ToList();

      // Crear el modelo con ambas listas
      var model = new ListPublicationsViewModel
      {
         Ventas = listaVentasOrdenada,
         Subastas = listaSubastasOrdenada
```

```
        };

        // Pasar el modelo a la vista
        return View(model);
    }
    return View();
}

[HttpGet]
public IActionResult SaleDetails(int id)
{
    if (HttpContext.Session.GetString("UserRole") != null)
    {
        // Almacena en una variable la venta activa
        Publicacion? venta = sistema.ObtenerPublicacionPorId(id, true, false);
        // Casteo exlpícito de Venta
        Venta? ventaActiva = (Venta?)venta;
        // Almacena la venta en una variable temporal
        if (ventaActiva != null)
        {
            ViewBag.Venta = ventaActiva;
        }

        // Obtiene el rol actual del usuario
        string? currentRole = HttpContext.Session.GetString("UserRole");
        if (currentRole == "Cliente")
```

```
        {
            // Obtiene el id del Usuario con el dato almacenado al realizar el login
            int idUser = HttpContext.Session.GetInt32("UserId") ?? 0;
            // Almacena en una variable el usuario activo
            Usuario? cliente = sistema.ObtenerUsuarioPorId(idUser, true, false);
            // Casteo explícito de Cliente
            Cliente? clienteActivo = (Cliente?)cliente;
            // Almacena el cliente en una variable temporal
            if (clienteActivo != null)
            {
                ViewBag.Cliente = clienteActivo;
            }
        }
    }
    return View();
}

[HttpPost]
public IActionResult SaleDetails(int ventaId, string action)
{
    try
    {
        // Obtiene el rol actual del usuario
        string? currentRole = HttpContext.Session.GetString("UserRole");

        // Almacena en una variable la venta activa
```

```
Publicacion? venta = sistema.ObtenerPublicacionPorId(ventaId, true, false);
// Casteo exlpícito de Venta
Venta? ventaActiva = (Venta?)venta;

if (ventaActiva != null)
{
    // Almacena la venta en una variable
    ViewBag.Venta = ventaActiva;

    // Logica para la compra de Publicaciones de tipo Venta
    if (currentRole == "Cliente" && action == "BuySale")
    {
        // Obtiene el id del Usuario con el dato almacenado al realizar el login
        int idUser = HttpContext.Session.GetInt32("UserId") ?? 0;
        // Almacena en una variable el usuario activo
        Usuario? cliente = sistema.ObtenerUsuarioPorId(idUser, true, false);
        // Casteo explícito de Cliente
        Cliente? clienteActivo = (Cliente?)cliente;

        if (clienteActivo != null)
        {
            // Almacena el cliente en una variable
            ViewBag.Cliente = clienteActivo;

            // Almacena en una variable el precio de venta
```

```
                decimal precioVenta = sistema.ConsultarPrecioVenta(ventaActiva,
ventaActiva.Articulos);


                // Manejo de errores
                if (clienteActivo?.Saldo < precioVenta)
                {
                   ViewBag.Mensaje = "Saldo insuficiente";
                }
                else if (ventaActiva?.Estado.ToUpper() != "ABIERTA")
                {
                   ViewBag.Mensaje = "La venta no se encuentra activa";
                }
                else
                {
                   if (clienteActivo != null)
                   {
                      // Cambia de estado la venta y registra el Cliente que la compró
                      // Cobra el valor de la venta al usuario activo
                      sistema.CompraVenta(clienteActivo, ventaActiva);


                      // Mensaje de Confirmación
                      ViewBag.Confirmacion = "La compra fue registrada correctamente, la
misma debe ser autorizada por un administrador";
                   }
                }
             }
          }
```

```
        }
    }
    catch (InvalidOperationException ex)
    {
        ViewBag.Mensaje = $"{ex.Message}";
    }
    catch (ArgumentNullException ex)
    {
        ViewBag.Mensaje = $"{ex.Message}";
    }
    catch (ArgumentException ex)
    {
        ViewBag.Mensaje = $"{ex.Message}";
    }
    catch (Exception ex)
    {
        ViewBag.Mensaje = $"{ex.Message}";
    }

    // Si algo falla, permanece en la vista
    return View();
}

[HttpGet]
public IActionResult AuctionDetails(int id)
{
```

```
if (HttpContext.Session.GetString("UserRole") != null)
{
    // Almacena en una variable la subasta activa
    Publicacion? subasta = sistema.ObtenerPublicacionPorId(id, false, true);
    // Casteo explícito a Subasta
    Subasta? subastaActiva = (Subasta?)subasta;
    // Crear el modelo con la Subasta
    if (subastaActiva != null)
    {
        ViewBag.Subasta = subastaActiva;
    };

    // Obtiene el rol actual del usuario
    string? currentRole = HttpContext.Session.GetString("UserRole");
    if (currentRole == "Cliente")
    {
        // Obtiene el id del Usuario con el dato almacenado al realizar el login
        int idUser = HttpContext.Session.GetInt32("UserId") ?? 0;
        // Almacena en una variable el usuario activo
        Usuario? cliente = sistema.ObtenerUsuarioPorId(idUser, true, false);
        // Casteo explícito de Cliente
        Cliente? clienteActivo = (Cliente?)cliente;
        // Almacena el cliente en una variable temporal
        if (clienteActivo != null)
        {
            ViewBag.Cliente = clienteActivo;
```

```
            }
        }
    }
    return View();
}

[HttpPost]
public IActionResult AuctionDetails(int subastaId, decimal monto, string action)
{
    try
    {
        // Obtiene el rol actual del usuario
        string? currentRole = HttpContext.Session.GetString("UserRole");

        // Almacena en una variable la subasta activa
        Publicacion? subasta = sistema.ObtenerPublicacionPorId(subastaId, false, true);
        // Casteo explícito a Subasta
        Subasta? subastaActiva = (Subasta?)subasta;

        if (subastaActiva != null)
        {
            // Almacena la subastaActiva en una variable temporal
            ViewBag.Subasta = subastaActiva;

            // Logica para la Oferta del Cliente en una Publicacion de tipo Subasta
            if (currentRole == "Cliente" && action == "OfferAuction")
```

```
{
    // Obtiene el id del Usuario con el dato almacenado al realizar el login
    int idUser = HttpContext.Session.GetInt32("UserId") ?? 0;
    // Almacena en una variable el usuario activo
    Usuario? cliente = sistema.ObtenerUsuarioPorId(idUser, true, false);
    // Casteo explícito de Cliente
    Cliente? clienteActivo = (Cliente?)cliente;

    if (clienteActivo != null)
    {
        // Almacena el cliente en una variable temporal
        ViewBag.Cliente = clienteActivo;

        // Almacena en una variable la oferta más alta
        decimal mejorOferta = 0;
        // Almacena en una variable si el cliente realizo una oferta para esta subasta anteriormente
        bool ofertoAnteriormente = false;

        for (int i = 0; i < subastaActiva.Ofertas.Count; i++)
        {
            // Calcula la oferta más alta
            if (subastaActiva.Ofertas[i].Monto > mejorOferta)
            {
                mejorOferta = subastaActiva.Ofertas[i].Monto;
            }
```

```
    // Verifica si el cliente actual es el mismo que el usuario de la oferta
    if (clienteActivo.Nombre == subastaActiva.Ofertas[i].Usuario?.Nombre)
    {
        ofertoAnteriormente = true;
    }
}

// Manejo de errores
if (monto <= 0)
{
    ViewBag.Mensaje = "El monto debe ser mayor que 0";
}
else if (monto % 1 != 0)
{
    ViewBag.Mensaje = "El monto debe ser un número entero";
}
else if (monto > clienteActivo.Saldo)
{
    ViewBag.Mensaje = "Saldo insuficiente";
}
else if (monto <= mejorOferta)
{
    ViewBag.Mensaje = "El monto debe ser mayor que la mejor oferta";
}
else if (ofertoAnteriormente)
{
```

```
            ViewBag.Mensaje = "Solo está permitido ofertar una vez por subasta";
        }
        else
        {
            // Crea la nueva oferta para la subasta actual
            sistema.AltaOferta(clienteActivo, subastaActiva, monto, DateTime.Now);

            // Mensaje de Confirmación
            ViewBag.Confirmacion = "La oferta fue registrada correctamente";
        }
    }
}
// Logica para el cierre de una Publicacion de tipo Subasta
else if (currentRole == "Administrador" && action == "CloseAuction")
{
    // Obtiene el id del Usuario con el dato almacenado al realizar el login
    int idUser = HttpContext.Session.GetInt32("UserId") ?? 0;
    // Almacena en una variable el usuario activo
    Usuario? administrador = sistema.ObtenerUsuarioPorId(idUser, false, true);
    // Casteo explícito de Administrador
    Administrador? administradorActivo = (Administrador?)administrador;

    if (administradorActivo != null)
    {
        // Almacena el administrador en una variable
        ViewBag.Administrador = administradorActivo;
```

```
            // Manejo de errores
            if (subastaActiva?.Estado.ToUpper() != "ABIERTA")
            {
                ViewBag.Mensaje = "La subasta no se encuentra activa";
            }
            else if (subastaActiva.Ofertas.Count == 0)
            {
                ViewBag.Mensaje = "No hay ofertas realizadas para esta subasta";
            }
            else
            {
                // Cambia de estado la subasta y registra el cliente que la ganó
                // Registra el Administrador que cerro la subasta y la fecha fin
                sistema.CompraSubasta(administradorActivo, subastaActiva);

                // Mensaje de Confirmación
                ViewBag.Confirmacion = "La subasta fue cerrada correctamente";
            }
        }
    }
  }
}
catch (InvalidOperationException ex)
{
  ViewBag.Mensaje = $"{ex.Message}";
```

```
        }
        catch (ArgumentNullException ex)
        {
            ViewBag.Mensaje = $"{ex.Message}";
        }
        catch (ArgumentException ex)
        {
            ViewBag.Mensaje = $"{ex.Message}";
        }
        catch (Exception ex)
        {
            ViewBag.Mensaje = $"{ex.Message}";
        }

        // Si algo falla, permanece en la vista
        return View();
    }
  }
}
```

SalesController

```
using InterfazUsuario.Models;
using LogicaNegocio;
using Microsoft.AspNetCore.Mvc;

namespace InterfazUsuario.Controllers
{
```

```
public class SalesController : Controller
{
    // Se llama a la instancia con patron singleton
    private Sistema sistema = Sistema.Instancia;

    [HttpGet]
    public IActionResult ListSales()
    {
        if (HttpContext.Session.GetString("UserRole") != null)
        {
            // Almacena en una variable las ventas
            List<Publicacion> ventas = sistema.ObtenerPublicaciones(true, false);
            // Casteo explícito de ventas
            List<Venta> listaVentas = ventas.OfType<Venta>().ToList();
            // Ordenar por fecha (ascendente)
            List<Venta> listaVentasOrdenada = listaVentas.OrderBy(p => p.Fecha).ToList();

            // Crear el modelo con ambas listas
            var model = new ListSalesViewModel
            {
                Ventas = listaVentasOrdenada
            };

            // Pasar el modelo a la vista
            return View(model);
        }
```

```
            return View();
        }
    }
}
```

WalletController.cs

```
using InterfazUsuario.Models;
using LogicaNegocio;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace InterfazUsuario.Controllers
{
    public class WalletController : Controller
    {
        // Se llama a la instancia con patron singleton
        private Sistema sistema = Sistema.Instancia;

        [HttpGet]
        public IActionResult Funds()
        {
            if (HttpContext.Session.GetString("UserRole") == "Cliente")
            {
                // Obtiene el id del Usuario con el dato almacenado al realizar el login
                int idUser = HttpContext.Session.GetInt32("UserId") ?? 0;
                // Almacena en una variable el usuario activo
```

```
        Usuario? cliente = sistema.ObtenerUsuarioPorId(idUser, true, false);
        // Casteo explícito de Cliente
        Cliente? clienteActivo = (Cliente?)cliente;

        // Crear el modelo con ambas listas
        var model = new FundsViewModel
        {
            Cliente = clienteActivo
        };

        // Pasar el modelo a la vista
        return View(model);
    }
    return View();
}
[HttpGet]
public IActionResult AddFunds()
{
    return View();
}

[HttpPost]
public IActionResult AddFunds(decimal saldo)
{
    try
    {
```

```
    // Manejo de errores
    if (saldo <= 0)
    {
        ViewBag.Mensaje = "El saldo a añadir debe ser mayor que 0";
    }
    else if (saldo % 1 != 0)
    {
        ViewBag.Mensaje = "El saldo a añadir debe ser un número entero";
    }
    else
    {
        // Obtiene el id del Usuario con el dato almacenado al realizar el login
        int idUser = HttpContext.Session.GetInt32("UserId") ?? 0;
        // Almacena en una variable el usuario activo
        Usuario? cliente = sistema.ObtenerUsuarioPorId(idUser, true, false);
        // Casteo explícito de Cliente
        Cliente? clienteActivo = (Cliente?)cliente;

        if (clienteActivo != null)
        {
            // Añade el saldo al usuario activo
            clienteActivo.Saldo += saldo;
            ViewBag.Confirmacion = "Saldo añadido correctamente correctamente";
        }
    }
}
```

```
        catch (InvalidOperationException ex)
        {
            ViewBag.Mensaje = $"Error de operación: {ex.Message}";
        }
        catch (ArgumentNullException ex)
        {
            ViewBag.Mensaje = $"{ex.Message}";
        }
        catch (ArgumentException ex)
        {
            ViewBag.Mensaje = $"{ex.Message}";
        }
        catch (Exception ex)
        {
            ViewBag.Mensaje = $"Error inesperado: {ex.Message}";
        }

        return View();
    }
  }
}
```

Models

ErrorViewModel.cs

```
namespace InterfazUsuario.Models
```

```
{
    public class ErrorViewModel
    {
        public string? RequestId { get; set; }

        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}
```

FoundsViewModel.cs

```
using LogicaNegocio;

namespace InterfazUsuario.Models
{
    public class FundsViewModel
    {
        public Cliente? Cliente { get; set; }
    }
}
```

ListAuctionsViewModel.cs

```
using LogicaNegocio;

namespace InterfazUsuario.Models
{
    public class ListAuctionsViewModel
```

```
    {
        public List<Subasta>? Subastas { get; set; }
    }
}
```

ListPublicationsViewModel.cs

```
using LogicaNegocio;

namespace InterfazUsuario.Models
{
    public class ListPublicationsViewModel
    {
        public List<Venta>? Ventas { get; set; }
        public List<Subasta>? Subastas { get; set; }
    }
}
```

ListSalesViewModel.cs

```
using LogicaNegocio;

namespace InterfazUsuario.Models
{
    public class ListSalesViewModel
    {
        public List<Venta>? Ventas { get; set; }
    }
}
```

View

Account

Login.cshtmml

```
@{
   ViewData["Title"] = "Login";
}

<div class="wide account">
   <h1>Login</h1>

   <div class="spaced frame form-container">
      <form method="post">
         <label>Email</label>
         <input type="text" name="email" />
         <label>Contraseña</label>
         <input type="text" name="contrasenia" />
         <input type="submit" value="Ingresar" class="btn btn-primary"/>
      </form>
   </div>

   <div class="frame action-container">
      <p>¿Todavia no estás registrado?</p>
      <!-- Botón con redirección a la página de registro -->
      <a href="/Account/Register" class="btn btn-outline-secondary">Registrarse</a>
   </div>
```

```
    @if (ViewBag.Mensaje != null)
    {
        <div class="spaced alert alert-danger">
            @ViewBag.Mensaje
        </div>
    }
</div>
```

Register. cshtmml

```
@{
    ViewData["Title"] = "Register";
}

<div class="wide account">
    <h1>Registro</h1>

    <div class="spaced frame form-container">
        <form method="post">
            <label>Nombre</label>
            <input type="text" name="nombre" />
            <label>Apellido</label>
            <input type="text" name="apellido" />
            <label>Email</label>
            <input type="text" name="email" />
            <label>Contraseña</label>
```

```
            <input type="text" name="contrasenia" />
            <input type="submit" value="Registrarse" class="btn btn-primary" />
        </form>
    </div>

    <div class="frame action-container">
        <p>¿Ya estás registrado?</p>
        <!-- Botón con redirección a la página de login -->
        <a href="/Account/Login" class="btn btn-outline-secondary">Iniciar sesion</a>
    </div>

    @if (ViewBag.Mensaje != null)
    {
        <div class="spaced alert alert-danger">
            @ViewBag.Mensaje
        </div>
    }

    @if (ViewBag.Confirmacion != null)
    {
        <div class="spaced alert alert-info">
            @ViewBag.Confirmacion
        </div>
    }
</div>
```

Auctions

ListAuctions.cshtml

```
@{
   ViewData["Title"] = "ListAuctions";
}

@{
   string? currentRole = ViewContext.HttpContext.Session.GetString("UserRole");
}

<!-- Muestra el contenido si el usuario está loggeado -->
@if (currentRole != null)
{
   <!-- Define el modelo a usar -->
   @model InterfazUsuario.Models.ListAuctionsViewModel

   <div class="text-center">
      <h1 class="display-4">Subastas</h1>
   </div>

   @if (Model.Subastas != null && Model.Subastas.Any())
   {
      <div class="spaced frame">
         <table class="table custom-row-height">
            <thead>
               <tr>
```

```
        <th>Id</th>
        <th>Nombre</th>
        <th>Estado</th>
        <th>Fecha</th>
        <th>Articulos</th>
        <th>Cliente</th>
        @if (currentRole == "Administrador")
        {
            <th>Administrador</th>
        }
        <th>Fecha fin</th>
        <th>Ofertas</th>
        <th>Precio</th>
        <th>Acción</th>
    </tr>
</thead>
<tbody>
    @for (int i = 0; i < Model.Subastas.Count; i++)
    {
        var subasta = Model.Subastas[i];
        decimal precio = 0;
        for (int j = 0; j < subasta.Ofertas.Count; j++)
        {
            decimal precioTemp = subasta.Ofertas[j].Monto;
            if (precioTemp > precio)
            {
```

```
                precio = precioTemp;
            }
        }
        <tr>
            <td>@subasta.Id</td>
            <td>@subasta.Nombre</td>
            <td>@subasta.Estado</td>
            <td>@subasta.Fecha.ToString("dd/MM/yyyy")</td>
            <td>@subasta.Articulos.Count</td>
            <td>@subasta.Cliente?.Nombre</td>
            @if (currentRole == "Administrador")
            {
                <td>@subasta.Administrador?.Nombre</td>
            }
            <td>
                @if (subasta.FechaFin != DateTime.MinValue)
                {
                    @subasta.FechaFin.ToString("dd/MM/yyyy")
                }
            </td>
            <td>@subasta.Ofertas.Count</td>
            <td>$@precio</td>
            <td>
                @if (subasta.Estado.ToUpper() == "ABIERTA" && currentRole ==
"Cliente")
                {
```

```
                        <a href="/Publications/AuctionDetails/@subasta.Id" class="btn btn-primary">Ofertar</a>
                    }
                    else if (currentRole == "Administrador")
                    {
                        <a href="/Publications/AuctionDetails/@subasta.Id" class="btn btn-primary">Administrar</a>
                    }
                </td>
            </tr>
        }
    </tbody>
</table>
</div>
}
else if (Model.Subastas == null)
{
<div class="widest spaced frame">
    <h2>No hay subastas</h2>
</div>
}
}
else
{
<div class="text-center">
    <h1 class="display-4">Acceso restringido</h1>
    <label>Ir a login</label>
```

```
    <!-- Botón con redirección a la página de login -->
    <a href="/Account/Login" class="btn btn-primary">Login</a>
  </div>
}
```

Home

Index. Cshtmml

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

@{
  string? currentRole = ViewContext.HttpContext.Session.GetString("UserRole");
}

<!-- Muestra el contenido si el usuario está loggeado -->
@if (currentRole != null)
{
  <div class="text-center">
    <h1 class="display-4">Bienvenido Cliente</h1>
    <p>Learn about <a href="https://learn.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
  </div>
}
else
{
  <div class="text-center">
```

```
        <h1 class="display-4">Acceso restringido</h1>
        <label>Ir a login</label>
        <!-- Botón con redirección a la página de login -->
        <a href="/Account/Login" class="btn btn-primary">Login</a>
    </div>
}
```

IndexAdmin. Cshtmml

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

@{
    string? currentRole = ViewContext.HttpContext.Session.GetString("UserRole");
}

<!-- Muestra el contenido si el usuario está loggeado como Administrador -->
@if (currentRole == "Administrador")
{
    <div class="text-center">
        <h1 class="display-4">Bienvenido Admin</h1>
        <p>Learn about <a href="https://learn.microsoft.com/aspnet/core">building Web
apps with ASP.NET Core</a>.</p>
    </div>
}
else
{
```

```
    <div class="text-center">
        <h1 class="display-4">Acceso restringido</h1>
        <label>Ir a login</label>
        <!-- Botón con redirección a la página de login -->
        <a href="/Account/Login" class="btn btn-primary">Login</a>
    </div>
}
```

Publications

AuctionDetails.cshtml

```
@{
    ViewData["Title"] = "OfferDetails";
}


@{
    string? currentRole = ViewContext.HttpContext.Session.GetString("UserRole");
    var cliente = ViewBag.Cliente;
    var subasta = ViewBag.Subasta;
    bool hayOfertas = ViewBag.Subasta != null && ViewBag.Subasta.Ofertas != null &&
ViewBag.Subasta.Ofertas.Count > 0;
}


<!-- Muestra el contenido si el usuario está loggeado -->
@if (currentRole != null)
{
    <div class="text-center">
```

```
    <h1 class="display-4">Detalles oferta</h1>
</div>

<div class="widest spaced frame">
    @if (ViewBag.Cliente != null && currentRole == "Cliente")
    {
        <p class="text-dark">Saldo disponible: $@cliente.Saldo</p>
    }

    @if (ViewBag.Subasta != null)
    {
        <!-- Datos sobre la subasta -->
        <div class="spaced">
            <table class="table custom-row-height">
                <thead>
                    <tr>
                        <th>Id</th>
                        <th>Nombre</th>
                        <th>Estado</th>
                        <th>Fecha</th>
                        <th>Articulos</th>
                        @if (currentRole == "Administrador")
                        {
                            <th>Cliente</th>
                        }
                        <th>Ofertas</th>
```

```
            <th>Precio</th>
        </tr>
    </thead>
    <tbody>
        @{
            <!-- La ultima oferta realizada siempre es la más alta -->
            decimal precio = 0;
            if (hayOfertas)
            {
                precio = @subasta.Ofertas[@subasta.Ofertas.Count - 1].Monto;
            }
            <tr>
                <td>@subasta.Id</td>
                <td>@subasta.Nombre</td>
                <td>@subasta.Estado</td>
                <td>@subasta.Fecha.ToString("dd/MM/yyyy")</td>
                <td>@subasta.Articulos.Count</td>
                @if (currentRole == "Administrador")
                {
                    <td>@subasta.Cliente?.Nombre</td>
                }
                <td>@subasta.Ofertas.Count</td>
                <td>$@precio</td>
            </tr>
        }
    </tbody>
```

```
    </table>
</div>
<!-- Datos sobre ofertas realizadas por otros clientes-->
@if (hayOfertas)
{
    <div class="spaced">
        <h2>Histórico</h2>
        <table class="table custom-row-height">
            <thead>
                <tr>
                    <th>Usuario</th>
                    <th>Fecha</th>
                    <th>Monto</th>
                </tr>
            </thead>
            <tbody>
                @for (int i = 0; i < subasta.Ofertas.Count; i++)
                {
                    <tr>
                        <td>@subasta.Ofertas[i].Usuario.Nombre</td>
                        <td>@subasta.Ofertas[i].Fecha.ToString("dd/MM/yyyy")</td>
                        <td>$@subasta.Ofertas[i].Monto</td>
                    </tr>
                }
            </tbody>
        </table>
```

```
            </div>
        }
    }

    <!-- Inputs para comprar o cerrar ventas -->
    <form method="post">
        <!-- Pasa el id de subasta obtenido de la URL al metodo post -->
        <input type="hidden" name="subastaId" value="@subasta.Id" />
        @if (subasta.Estado.ToUpper() == "ABIERTA" && currentRole == "Cliente")
        {
            <label>Monto</label>
            <input type="number" name="monto" />
            <button type="submit" name="action" value="OfferAuction" class="btn btn-
primary">Ofertar</button>
        }
        else if (subasta.Estado.ToUpper() == "ABIERTA" && currentRole == "Administrador")
        {
            <button type="submit" name="action" value="CloseAuction" class="btn btn-
secondary">Cerrar subasta</button>
        }
    </form>
</div>

<!-- Muestra mensajes de error -->
@if (ViewBag.Mensaje != null)
{
    <div class="widest spaced alert alert-danger">
```

```
            @ViewBag.Mensaje
        </div>
    }
    <!-- Muestra mensajes de confirmación de procesos -->
    @if (ViewBag.Confirmacion != null)
    {
        <div class="widest spaced alert alert-info">
            @ViewBag.Confirmacion
        </div>
    }
    <!-- Permite volver al menu anterior -->
    <div class="align-center widest frame action-container">
        <p>Haga click aquí para volver a la vista de publicaciones</p>
        <!-- Botón con redirección a la página de Listar Publicaciones -->
        <a href="/Publications/ListPublications" class="btn btn-outline-
secondary">Volver</a>
    </div>
}
else
{
    <div class="text-center">
        <h1 class="display-4">Acceso restringido</h1>
        <label>Ir a login</label>
        <!-- Botón con redirección a la página de login -->
        <a href="/Account/Login" class="btn btn-primary">Login</a>
    </div>
```

```
}
```

ListPublications. Cshtmml

```
@{
    ViewData["Title"] = "ListPublications";
}

@{
    string? currentRole = ViewContext.HttpContext.Session.GetString("UserRole");
}

<!-- Muestra el contenido si el usuario está loggeado -->
@if (currentRole != null)
{
    <!-- Define el modelo a usar -->
    @model InterfazUsuario.Models.ListPublicationsViewModel

    <div class="text-center">
        <h1 class="display-4">Publicaciones</h1>
    </div>

    @if (Model.Ventas != null && Model.Ventas.Any())
    {
        <div class="spaced frame">
            <h2>Ventas</h2>
```

```
<table class="table custom-row-height">
    <thead>
        <tr>
            <th>Id</th>
            <th>Nombre</th>
            <th>Estado</th>
            <th>Fecha</th>
            <th>Articulos</th>
            <th>Cliente</th>
            <th>Fecha fin</th>
            <th>Oferta relampago</th>
            <th>Precio</th>
            <th>Acción</th>
        </tr>
    </thead>
    <tbody>
        @for (int i = 0; i < Model.Ventas.Count; i++)
        {
            var venta = Model.Ventas[i];
            decimal precio = 0;
            for (int j = 0; j < venta.Articulos.Count; j++)
            {
                decimal precioTemp = venta.Articulos[j].Precio;
                if (venta.OfertaRelampago)
                {
                    precioTemp = (precioTemp * 80) / 100;
```

```
                    }
                    precio += precioTemp;
                }
                <tr>
                    <td>@venta.Id</td>
                    <td>@venta.Nombre</td>
                    <td>@venta.Estado</td>
                    <td>@venta.Fecha.ToString("dd/MM/yyyy")</td>
                    <td>@venta.Articulos.Count</td>
                    <td>@venta.Cliente?.Nombre</td>
                    <td>
                        @if (venta.FechaFin != DateTime.MinValue)
                        {
                            @venta.FechaFin.ToString("dd/MM/yyyy");
                        }
                    </td>
                    <td>@(venta.OfertaRelampago ? "Sí" : "No")</td>
                    <td>$@precio</td>
                    <td>
                        @if (venta.Estado.ToUpper() == "ABIERTA" && currentRole == "Cliente")
                        {
                            <a href="/Publications/SaleDetails/@venta.Id" class="btn btn-
primary">Comprar</a>
                        }
                        else if (currentRole == "Administrador")
                        {
```

```
                        <a href="/Publications/SaleDetails/@venta.Id" class="btn btn-
primary">Administrar</a>
                    }
                </td>
            </tr>
        }
    </tbody>
</table>
</div>
}
else if (Model.Ventas == null)
{
<div class="widest spaced frame">
    <h2>No hay ventas</h2>
</div>
}

@if (Model.Subastas != null && Model.Subastas.Any())
{
<div class="spaced frame">
    <h2>Subastas</h2>
    <table class="table custom-row-height">
        <thead>
            <tr>
                <th>Id</th>
                <th>Nombre</th>
```

```
        <th>Estado</th>
        <th>Fecha</th>
        <th>Articulos</th>
        <th>Cliente</th>
        @if (currentRole == "Administrador")
        {
            <th>Administrador</th>
        }
        <th>Fecha fin</th>
        <th>Ofertas</th>
        <th>Precio</th>
        <th>Acción</th>
    </tr>
</thead>
<tbody>
    @for (int i = 0; i < Model.Subastas.Count; i++)
    {
        var subasta = Model.Subastas[i];
        decimal precio = 0;
        for (int j = 0; j < subasta.Ofertas.Count; j++)
        {
            decimal precioTemp = subasta.Ofertas[j].Monto;
            if (precioTemp > precio)
            {
                precio = precioTemp;
            }
```

```
                }
                <tr>
                    <td>@subasta.Id</td>
                    <td>@subasta.Nombre</td>
                    <td>@subasta.Estado</td>
                    <td>@subasta.Fecha.ToString("dd/MM/yyyy")</td>
                    <td>@subasta.Articulos.Count</td>
                    <td>@subasta.Cliente?.Nombre</td>
                    @if (currentRole == "Administrador")
                    {
                        <td>@subasta.Administrador?.Nombre</td>
                    }
                    <td>
                        @if (subasta.FechaFin != DateTime.MinValue)
                        {
                            @subasta.FechaFin.ToString("dd/MM/yyyy")
                        }
                    </td>
                    <td>@subasta.Ofertas.Count</td>
                    <td>$@precio</td>
                    <td>
                        @if (subasta.Estado.ToUpper() == "ABIERTA" && currentRole ==
"Cliente")
                        {
                            <a href="/Publications/AuctionDetails/@subasta.Id" class="btn btn-
primary">Ofertar</a>
                        }
```

```
                    else if (currentRole == "Administrador")
                    {
                        <a href="/Publications/AuctionDetails/@subasta.Id" class="btn btn-primary">Administrar</a>
                    }
                </td>
            </tr>
        }
    </tbody>
</table>
</div>
}
else if (Model.Subastas == null)
{
<div class="widest spaced frame">
    <h2>No hay subastas</h2>
</div>
}
}
else
{
<div class="text-center">
    <h1 class="display-4">Acceso restringido</h1>
    <label>Ir a login</label>
    <!-- Botón con redirección a la página de login -->
    <a href="/Account/Login" class="btn btn-primary">Login</a>
```

```
    </div>
}
```

SaleDetails.cshtml

```
@{
    ViewData["Title"] = "SaleDetails";
}

@{
    string? currentRole = ViewContext.HttpContext.Session.GetString("UserRole");
    var cliente = ViewBag.Cliente;
    var venta = ViewBag.Venta;
}

<!-- Muestra el contenido si el usuario está loggeado -->
@if (currentRole != null)
{
    <div class="text-center">
        <h1 class="display-4">Detalles venta</h1>
    </div>

    <div class="widest spaced frame">
        @if (ViewBag.Cliente != null && currentRole == "Cliente")
        {
            <p class="text-dark">Saldo disponible: $@cliente.Saldo</p>
        }
```

```
@if (ViewBag.Venta != null)
{
    <!-- Datos sobre la venta -->
    <div class="spaced">
        <table class="table custom-row-height">
            <thead>
                <tr>
                    <th>Id</th>
                    <th>Nombre</th>
                    <th>Estado</th>
                    <th>Fecha</th>
                    <th>Articulos</th>
                    @if (currentRole == "Administrador")
                    {
                        <th>Cliente</th>
                    }
                    <th>Oferta relampago</th>
                    <th>Precio</th>
                </tr>
            </thead>
            <tbody>
                @{
                    decimal precio = 0;
                    for (int i = 0; i < venta.Articulos.Count; i++)
                    {
```

```
                decimal precioTemp = venta.Articulos[i].Precio;
                if (venta.OfertaRelampago)
                {
                    precioTemp = (precioTemp * 80) / 100;
                }
                precio += precioTemp;
            }
        }
        <tr>
            <td>@venta.Id</td>
            <td>@venta.Nombre</td>
            <td>@venta.Estado</td>
            <td>@venta.Fecha.ToString("dd/MM/yyyy")</td>
            <td>@venta.Articulos.Count</td>
            @if (currentRole == "Administrador")
            {
                <td>@venta.Cliente?.Nombre</td>
            }
            <td>@(venta.OfertaRelampago ? "Sí" : "No")</td>
            <td>$@precio</td>
        </tr>
    </tbody>
</table>
</div>
<!-- Datos sobre los articulos asociados a la venta -->
<div class="spaced">
```

```
        <table class="table custom-row-height">
            <thead>
                <tr>
                    <th>Nombre</th>
                    <th>Categoría</th>
                    <th>Precio</th>
                </tr>
            </thead>
            <tbody>
                @for (int i = 0; i < @venta.Articulos.Count; i++)
                {
                    <tr>
                        <td>@venta.Articulos[i].Nombre</td>
                        <td>@venta.Articulos[i].Categoria</td>
                        <td>$@venta.Articulos[i].Precio</td>
                    </tr>
                }
            </tbody>
        </table>
    </div>
}

<!-- Inputs para comprar o cerrar ventas -->
<form method="post">
    <!-- Pasa el id de venta obtenido de la URL al metodo post -->
    <input type="hidden" name="ventaId" value="@venta.Id" />
```

```
        @if (currentRole == "Cliente")
        {
            <button type="submit" name="action" value="BuySale" class="btn btn-primary">Confirmar compra</button>
        }
    </form>
</div>

<!-- Muestra mensajes de error -->
@if (ViewBag.Mensaje != null)
{
    <div class="widest spaced alert alert-danger">
        @ViewBag.Mensaje
    </div>
}
<!-- Muestra mensajes de confirmación de procesos -->
@if (ViewBag.Confirmacion != null)
{
    <div class="widest spaced alert alert-info">
        @ViewBag.Confirmacion
    </div>
}
<!-- Permite volver al menu anterior -->
<div class="align-center widest frame action-container">
    <p>Haga click aquí para volver a la vista de publicaciones</p>
    <!-- Botón con redirección a la página de Listar Publicaciones -->
```

```
        <a href="/Publications/ListPublications" class="btn btn-outline-
secondary">Volver</a>
    </div>
}
else
{
    <div class="text-center">
        <h1 class="display-4">Acceso restringido</h1>
        <label>Ir a login</label>
        <!-- Botón con redirección a la página de login -->
        <a href="/Account/Login" class="btn btn-primary">Login</a>
    </div>
}
```

Sales

ListSales.cshtml

```
@{
    ViewData["Title"] = "ListSales";
}

@{
    string? currentRole = ViewContext.HttpContext.Session.GetString("UserRole");
}

<!-- Muestra el contenido si el usuario está loggeado -->
@if (currentRole != null)
```

```
{
    <!-- Define el modelo a usar -->
    @model InterfazUsuario.Models.ListSalesViewModel

    <div class="text-center">
        <h1 class="display-4">Ventas</h1>
    </div>

    @if (Model.Ventas != null && Model.Ventas.Any())
    {
        <div class="spaced frame">
            <table class="table custom-row-height">
                <thead>
                    <tr>
                        <th>Id</th>
                        <th>Nombre</th>
                        <th>Estado</th>
                        <th>Fecha</th>
                        <th>Articulos</th>
                        <th>Cliente</th>
                        <th>Fecha fin</th>
                        <th>Oferta relampago</th>
                        <th>Precio</th>
                        <th>Acción</th>
                    </tr>
                </thead>
```

```
<tbody>
    @for (int i = 0; i < Model.Ventas.Count; i++)
    {
        var venta = Model.Ventas[i];
        decimal precio = 0;
        for (int j = 0; j < venta.Articulos.Count; j++)
        {
            decimal precioTemp = venta.Articulos[j].Precio;
            if (venta.OfertaRelampago)
            {
                precioTemp = (precioTemp * 80) / 100;
            }
            precio += precioTemp;
        }
        <tr>
            <td>@venta.Id</td>
            <td>@venta.Nombre</td>
            <td>@venta.Estado</td>
            <td>@venta.Fecha.ToString("dd/MM/yyyy")</td>
            <td>@venta.Articulos.Count</td>
            <td>@venta.Cliente?.Nombre</td>
            <td>
                @if (venta.FechaFin != DateTime.MinValue)
                {
                    @venta.FechaFin.ToString("dd/MM/yyyy")
                    ;
```

```
                    }
                </td>
                <td>@(venta.OfertaRelampago ? "Sí" : "No")</td>
                <td>$@precio</td>
                <td>
                    @if (venta.Estado.ToUpper() == "ABIERTA" && currentRole == "Cliente")
                    {
                        <a href="/Publications/SaleDetails/@venta.Id" class="btn btn-
primary">Comprar</a>
                    }
                    else if (currentRole == "Administrador")
                    {
                        <a href="/Publications/SaleDetails/@venta.Id" class="btn btn-
primary">Administrar</a>
                    }
                </td>
            </tr>
        }
    </tbody>
</table>
</div>
}
else if (Model.Ventas == null)
{
<div class="widest spaced frame">
    <h2>No hay ventas</h2>
</div>
```

```
    }
}
else
{
    <div class="text-center">
        <h1 class="display-4">Acceso restringido</h1>
        <label>Ir a login</label>
        <!-- Botón con redirección a la página de login -->
        <a href="/Account/Login" class="btn btn-primary">Login</a>
    </div>
}
```

Shared

_Layout. Cshtmml

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - InterfazUsuario</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/InterfazUsuario.styles.css" asp-append-version="true" />
</head>
<body>
    @{
```

```
    string? currentAction = ViewContext.RouteData.Values["Action"]?.ToString();
    string? currentRole = ViewContext.HttpContext.Session.GetString("UserRole");
  }
  @if (currentAction != "Login" && currentAction != "Register")
  {
    <header>
      <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
        <div class="container-fluid">
          <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">InterfazUsuario</a>
          <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls="navbarSupportedContent"
          aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
          </button>
          <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
            <ul class="navbar-nav flex-grow-1">
              @if (currentRole == "Cliente")
              {
                <li class="nav-item">
                  <a class="nav-link text-dark" asp-area="" asp-controller="Wallet" asp-action="Funds">Billetera</a>
                </li>
              }
              @if (currentRole == "Cliente" || currentRole == "Administrador")
              {
```

```
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-
controller="Publications" asp-action="ListPublications">Publicaciones</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-controller="Sales" asp-
action="ListSales">Ventas</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-controller="Auctions"
asp-action="ListAuctions">Subastas</a>
                    </li>
                    <li class="nav-item order-last">
                        <a class="nav-link text-dark" asp-area="" asp-controller="Account"
asp-action="Logout">Logout</a>
                    </li>
                }
            </ul>
        </div>
    </div>
</nav>
</header>
}
<div class="container">
<main role="main" class="pb-3">
    @RenderBody()
</main>
```

```
    </div>
    @if (currentAction != "Login" && currentAction != "Register")
    {
        <footer class="border-top footer text-muted">
            <div class="container">
                &copy; 2024 - InterfazUsuario - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
            </div>
        </footer>
    }
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

_ValidationScriptPartial. Cshtmml

```
<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
<script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>
```

Error. Cshtmml

```
@model ErrorViewModel
@{
    ViewData["Title"] = "Error";
}
```

```
<h1 class="text-danger">Error.</h1>

<h2 class="text-danger">An error occurred while processing your request.</h2>


@if (Model.ShowRequestId)

{

   <p>

      <strong>Request ID:</strong> <code>@Model.RequestId</code>

   </p>

}


<h3>Development Mode</h3>

<p>

   Swapping to <strong>Development</strong> environment will display more detailed information about the error that occurred.

</p>

<p>

   <strong>The Development environment shouldn't be enabled for deployed applications.</strong>

   It can result in displaying sensitive information from exceptions to end users.

   For local debugging, enable the <strong>Development</strong> environment by setting the <strong>ASPNETCORE_ENVIRONMENT</strong> environment variable to <strong>Development</strong>

   and restarting the app.

</p>
```

Wallet

AddFunds. Cshtmml

```
@{
   ViewData["Title"] = "AddFunds";
}

@{
   string? currentRole = ViewContext.HttpContext.Session.GetString("UserRole");
}

<!-- Muestra el contenido si el cliente está loggeado -->
@if (currentRole == "Cliente")
{
   <div class="text-center">
      <h1 class="display-4">Añadir fondos</h1>
   </div>

   <div class="wider spaced frame form-container">
      <form method="post">
         <label>Saldo</label>
         <input type="number" name="saldo">
         <input type="submit" value="Añadir saldo" class="btn btn-primary" />
      </form>
   </div>
```

```
    <div class="align-center wider frame action-container">
        <p>Haga click aquí para volver a la Billetera</p>
        <!-- Botón con redirección a la página de mostrar fondos -->
        <a href="/Wallet/Funds" class="btn btn-outline-secondary">Volver</a>
    </div>

    @if (ViewBag.Mensaje != null)
    {
        <div class="wider spaced alert alert-danger">
            @ViewBag.Mensaje
        </div>
    }

    @if (ViewBag.Confirmacion != null)
    {
        <div class="wider spaced alert alert-info">
            @ViewBag.Confirmacion
        </div>
    }
}
else
{
    <div class="text-center">
        <h1 class="display-4">Acceso restringido</h1>
        <label>Ir a login</label>
        <!-- Botón con redirección a la página de login -->
```

```
        <a href="/Account/Login" class="btn btn-primary">Login</a>
    </div>
}
```

Funds. cshtmml

```
@{
    ViewData["Title"] = "Funds";
}

@{
    string? currentRole = ViewContext.HttpContext.Session.GetString("UserRole");
}

<!-- Muestra el contenido si el cliente está loggeado -->
@if (currentRole == "Cliente")
{
    <!-- Define el modelo a usar -->
    @model InterfazUsuario.Models.FundsViewModel

    <div class="text-center">
        <h1 class="display-4">Billetera</h1>
    </div>
    @if (Model.Cliente != null)
    {
        <div class="wider spaced frame form-container">
```

```
            <p>Fondos disponibles en su cuenta</p>
            <p>$@Model.Cliente.Saldo</p>
        </div>
    }
    <div class="align-center wider frame action-container">
        <p>Haga click aquí para añadir saldo a su cuenta</p>
        <!-- Botón con redirección a la página de añadir saldo -->
        <a href="/Wallet/AddFunds" class="btn btn-outline-secondary">Añadir fondos</a>
    </div>
}
else
{
    <div class="text-center">
        <h1 class="display-4">Acceso restringido</h1>
        <label>Ir a login</label>
        <!-- Botón con redirección a la página de login -->
        <a href="/Account/Login" class="btn btn-primary">Login</a>
    </div>
}
```

Program.cs (MVC)

```
using LogicaNegocio;


namespace InterfazUsuario
{
    public class Program
```

```
{
    public static void Main(string[] args)
    {
        // Obtener la instancia única de Sistema
        Sistema sistema = Sistema.Instancia;

        var builder = WebApplication.CreateBuilder(args);

        // Add services to the container.
        builder.Services.AddControllersWithViews();
        builder.Services.AddSession();

        var app = builder.Build();

        // Configure the HTTP request pipeline.
        if (!app.Environment.IsDevelopment())
        {
            app.UseExceptionHandler("/Home/Error");
            app.UseHsts();
        }
        app.UseHttpsRedirection();

        app.UseStaticFiles();

        app.UseRouting();
```

```
            app.UseSession();

            app.UseAuthorization();

            app.MapControllerRoute(
                name: "default",
                pattern: "{controller=Account}/{action=Login}/{id?}");

            app.Run();
        }
    }
}
```

LOGICANEGOCIO

ADMINISTRADOR.CS

```
using LogicaNegocio.Interfaces;
```

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

namespace LogicaNegocio
{
   public class Administrador : Usuario
   {
      #region Constructor
      public Administrador(string nombre, string apellido, string email, string contrasenia)
         : base(nombre, apellido, email, contrasenia) // Llamada al constructor de la clase base (Usuario)
      {
      }
      #endregion

      #region Validación
      // Validación de Administrador, hereda de Usuario
      public override void Validar()
      {
      }
      #endregion

      #region Método Equals
```

```
        // Sobre escritura del metodo Equals que es usado por Contains
        public override bool Equals(object? obj)
        {
            if (obj != null && obj is Usuario)
            {
                Usuario usuario = (Usuario)obj;
                return Nombre == usuario.Nombre && Apellido == usuario.Apellido;
            }
            return false;
        }
        #endregion
    }
}
```

ARTICULO.CS

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

namespace LogicaNegocio
{
    public class Articulo : IValidate
    {
```

```
#region  Atributos de la clase
private int _id;
private static int s_ultId = 0; // Inicializado con el id siguiente a la ultima precarga
private string _nombre = string.Empty; // Inicializado con una cadena vacía
private decimal _precio = 0; // Inicializado con 0
private string _categoria = string.Empty; // Inicializado con una cadena vacía
#endregion

#region Propiedades
public int Id
{
   get { return _id; }  // Solo lectura, asignado internamente.
}
public string Nombre
{
   get { return _nombre; }
   set { _nombre = EvaluarNombre(value); }
}
public decimal Precio
{
   get { return _precio; }
   set { _precio = EvaluarPrecio(value); }
}
public string Categoria
{
   get { return _categoria; }
```

```
    set { _categoria = EvaluarCategoria(value); }
}
#endregion

#region Constructor
public Articulo(string nombre, decimal precio, string categoria)
{
    _id = Articulo.s_ultId; // Asigna el ID único
    Articulo.s_ultId++; // Incrementa el ID único
    Nombre = nombre;
    Precio = precio;
    Categoria = categoria;
}
#endregion

#region Validación
// Evaluaciones
private static string EvaluarNombre(string nombre)
{
    if (string.IsNullOrEmpty(nombre))
    {
        throw new ArgumentException("El nombre no puede ser vacío");
    }
    return nombre;
}
private static decimal EvaluarPrecio(decimal precio)
```

```
{
    if (precio < 0)
    {
        throw new InvalidOperationException("El precio no puede ser negativo");
    }
    return precio;
}
private static string EvaluarCategoria(string categoria)
{
    if (string.IsNullOrEmpty(categoria))
    {
        throw new ArgumentException("La categoria no puede ser vacía");
    }
    return categoria;
}

// Validación de Articulo
public void Validar()
{
}
#endregion

#region Método Equals
// Sobre escritura del metodo Equals que es usado por Contains
public override bool Equals(object? obj)
{
```

```
            if (obj != null && obj is Articulo)
            {
                Articulo articulo = (Articulo)obj;
                return Nombre == articulo.Nombre;
            }
            return false;
        }
        #endregion
    }
}
```

CLIENTE.CS

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

namespace LogicaNegocio
{
    public class Cliente : Usuario, IValidate
    {
        #region Atributos de la clase
        private decimal _saldo = 0; // Inicializado en 0
        #endregion
```

```
    #region Propiedades
    public decimal Saldo
    {
       get { return _saldo; }
       set { _saldo = EvaluarSaldo(value); }
    }
    #endregion


    #region Constructor
    public Cliente(string nombre, string apellido, string email, string contrasenia, decimal
saldo)
       : base(nombre, apellido, email, contrasenia) // Llamada al constructor de la clase
base (Usuario)
    {
       Saldo = saldo;
    }
    #endregion


    #region Validación
    // Evaluaciones
    private static decimal EvaluarSaldo(decimal saldo)
    {
       if (saldo < 0)
       {
          throw new InvalidOperationException("El saldo no puede ser negativo");
       }
```

```
            return saldo;
        }

        // Validación de Cliente, hereda de Usuario
        public override void Validar()
        {
        }
        #endregion

        #region Método Equals
        // Sobre escritura del metodo Equals que es usado por Contains
        public override bool Equals(object? obj)
        {
            if (obj != null && obj is Usuario)
            {
                Usuario usuario = (Usuario)obj;
                return Nombre == usuario.Nombre && Apellido == usuario.Apellido;
            }
            return false;
        }
        #endregion
    }
}
```

OFERTA.CS

```
using LogicaNegocio.Interfaces;
```

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

namespace LogicaNegocio
{
    public class Oferta : IValidate
    {
        #region Atributos de la clase
        private int _id;
        private static int s_ultId = 0; // Inicializado con el id siguiente a la ultima precarga
        private Usuario? _usuario; // Inicializado con una instancia por defecto
        private decimal _monto = 0; // Inicializado con 0
        private DateTime _fecha = DateTime.Now; // Inicializado con la fecha actual
        #endregion

        #region Propiedades
        public int Id
        {
            get { return _id; }  // Solo lectura, asignado internamente.
        }
        public Usuario? Usuario
        {
```

```
    get { return _usuario; }
    set { _usuario = EvaluarUsuario(value); }
}
public decimal Monto
{
    get { return _monto; }
    set { _monto = EvaluarMonto(value); }
}
public DateTime Fecha
{
    get { return _fecha; }
    set { _fecha = value; }
}
#endregion

#region Constructor
public Oferta(Usuario? usuario, decimal monto, DateTime fecha)
{
    _id = Oferta.s_ultId; // Asigna el ID único
    Oferta.s_ultId++; // Incrementa el ID único
    Usuario = usuario;
    Monto = monto;
    Fecha = fecha;
}
#endregion
```

```
    #region Validación
    // Evaluaciones
    private static Usuario EvaluarUsuario(Usuario? usuario)
    {
        if (usuario == null)
        {
            throw new ArgumentNullException("No se encontró un usuario correspondiente
a los datos proporcionados");
        }
        return usuario;
    }
    private static decimal EvaluarMonto(decimal monto)
    {
        if (monto <= 0)
        {
            throw new InvalidOperationException("El monto no puede ser menor a 0");
        }
        if (monto % 1 != 0)
        {
            throw new InvalidOperationException("El monto debe ser un número entero");
        }
        return monto;
    }
    // Validación de Oferta
    public void Validar()
    {
```

```
        }
        #endregion

        #region Método Equals
        // Sobre escritura del metodo Equals que es usado por Contains
        public override bool Equals(object? obj)
        {
            if (obj != null && obj is Oferta)
            {
                Oferta oferta = (Oferta)obj;
                return Usuario == oferta.Usuario;
            }
            return false;
        }
        #endregion
    }
}
```

PUBLICACION.CS

```
using LogicaNegocio.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
```

```
using System.Threading.Tasks;


namespace LogicaNegocio
{
    public class Publicacion : IValidate
    {
        #region Atributos de la clase
        private int _id;
        private static int s_ultId = 0; // Inicializado con el id siguiente a la ultima precarga
        private string _nombre = string.Empty; // Inicializado con una cadena vacía
        private string _estado = string.Empty; // Inicializado con una cadena vacía
        private DateTime _fecha = DateTime.Now; // Inicializado con la fecha actual
        private List<Articulo> _articulos = new List<Articulo>(); // Inicializado con una lista vacía
        private Cliente? _cliente; // Inicializado con una instancia por defecto
        private DateTime _fechaFin = DateTime.Now; // Inicializado con la fecha actual
        #endregion


        #region Propiedades
        public int Id
        {
            get { return _id; }  // Solo lectura, asignado internamente.
        }
        public string Nombre
        {
            get { return _nombre; }
```

```
    set { _nombre = EvaluarNombre(value); }
}
public string Estado
{
    get { return _estado; }
    set { _estado = EvaluarEstado(value); }
}
public DateTime Fecha
{
    get { return _fecha; }
    set { _fecha = value; }
}
public DateTime FechaFin
{
    get { return _fechaFin; }
    set { _fechaFin = value; }
}
public List<Articulo> Articulos
{
    get { return _articulos; }
    set { _articulos = value; }
}
public Cliente? Cliente
{
    get { return _cliente; }
    set { _cliente = value; }
```

```
    }
    #endregion

    #region Constructor
    public Publicacion(string nombre, string estado, DateTime fecha, List<Articulo>
articulos, Cliente? cliente, DateTime fechaFin)
    {
        _id = Publicacion.s_ultId; // Asigna el ID único
        Publicacion.s_ultId++; // Incrementa el ID único
        Nombre = nombre;
        Estado = estado;
        Fecha = fecha;
        Articulos = articulos;
        Cliente = cliente;
        FechaFin = fechaFin;
    }
    #endregion

    #region Validación
    // Evaluaciones
    private static string EvaluarNombre(string nombre)
    {
        if (string.IsNullOrEmpty(nombre))
        {
            throw new ArgumentException("El nombre no puede ser vacío");
        }
```

```
        return nombre;
    }
    private static string EvaluarEstado(string estado)
    {
        if (estado != "ABIERTA" && estado != "CERRADA" && estado != "CANCELADA")
        {
            throw new ArgumentException("El estado de la publicacion tiene que ser ABIERTA, CERRADA o CANCELADA");
        }
        return estado;
    }

    // Validación de Publicacion, es virtual ya que le hereda a otras clases
    public virtual void Validar()
    {
        if (FechaFin <= Fecha && FechaFin != DateTime.MinValue)
        {
            throw new InvalidOperationException("La fecha de fin debe ser posterior a la fecha de inicio.");
        }
    }
    #endregion

    #region Método Equals
    // Sobre escritura del metodo Equals que es usado por Contains
    public override bool Equals(object? obj)
    {
```

```
            if (obj != null && obj is Publicacion)
            {
                Publicacion publicacion = (Publicacion)obj;
                return Nombre == publicacion.Nombre;
            }
            return false;
        }
        #endregion
    }
}
```

SISTEMA.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using System.Timers;
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace LogicaNegocio
{
    public class Sistema
    {
```

```
#region Patron singleton
private static Sistema? _instancia; // Instancia única
private static readonly object _bloqueo = new object(); // Para hilos seguros

// Propiedad para acceder a la instancia única
public static Sistema Instancia
{
   get
   {
      // Double-check locking para garantizar seguridad en multithreading
      if (_instancia == null)
      {
         lock (_bloqueo)
         {
            if (_instancia == null)
            {
               _instancia = new Sistema();
            }
         }
      }
      return _instancia;
   }
}
#endregion

#region Constructor
```

```
// Atributos de la clase con propiedades automaticas (shortHand)
private List<Usuario> _usuarios {  get; set; }
private List<Publicacion> _publicaciones { get; set; }
private List<Articulo> _articulos { get; set; }

// Ejecucion principal
private Sistema()
{
   _usuarios = new List<Usuario>();
   _publicaciones = new List<Publicacion>();
   _articulos = new List<Articulo>();
   PrecargaArticulo();
   PrecargaUsuario();
   PrecargarPublicacion();
   PrecargaOferta();
}
#endregion

/// <summary>
/// Las lista son utilizadas en todas las funciones de impresión.
/// Por ejemplo si queremos imprimir clientes, debemos pasarle a la función
/// imprimirUsuario una lista de clientes.
/// Esta lista se puede conseguir con:
/// ObtenerCliente, almacena en una lista todos los usuarios que sean clientes
/// obtenerClientePorId, almacena en una lista los clientes de ids determinados
```

```
    /// obtenerClientePorNombre, almacena en una lista los clientes de nombres determinados
    /// </summary>
    #region Obtención de listas
    #region Articulo
    public List<Articulo> ObtenerArticuloPorId(List<int> ids)
    {
        bool hayArticulo = false;
        List<Articulo> articulos = new List<Articulo>();  // Inicializamos la lista que contendrá los artículos
        for (int i = 0; i < _articulos.Count; i++)
        {
            if (ids.Contains(_articulos[i].Id)) // Si la lista de ids contiene algún artículo
            {
                hayArticulo = true;
                articulos.Add(_articulos[i]); // Se añade el artículo a la lista artículos
            }
        }
        if (!hayArticulo)
        {
            throw new ArgumentException("No hay ningún artículo con los ids proporcionados");
        }
        return articulos;
    }
    #endregion
    #region Publicacion
```

```
    public List<Publicacion> ObtenerPublicaciones(bool esUnicamenteVenta, bool esUnicamenteSubasta)
    {
        bool hayVenta = false;
        bool haySubasta = false;
        List<Publicacion> publicaciones = new List<Publicacion>();  // Inicializamos la lista que contendrá las publicaciones
        for (int i = 0; i < _publicaciones.Count; i++)
        {
            if ((esUnicamenteVenta && !esUnicamenteSubasta) || (!esUnicamenteVenta && !esUnicamenteSubasta))
            {
                if (_publicaciones[i] is Venta venta)
                {
                    hayVenta = true;
                    publicaciones.Add(venta); // Se añade la venta a la lista publicaciones
                }
            }
            if ((esUnicamenteSubasta && !esUnicamenteVenta) || (!esUnicamenteVenta && !esUnicamenteSubasta))
            {
                if (_publicaciones[i] is Subasta subasta)
                {
                    haySubasta = true;
                    publicaciones.Add(subasta); // Se añade la subasta a la lista publicaciones
                }
            }
```

```
        }
        if (!hayVenta && !haySubasta)
        {
            throw new ArgumentException("No hay ningúna pubicación en el sistema");
        }
        if (!hayVenta && esUnicamenteVenta)
        {
            throw new ArgumentException("No hay ningúna venta en el sistema");
        }
        if (!haySubasta && esUnicamenteSubasta)
        {
            throw new ArgumentException("No hay ningúna subasta en el sistema");
        }
        return publicaciones;
    }
    public Publicacion? ObtenerPublicacionPorId(int id, bool esUnicamenteVenta, bool
esUnicamenteSubasta)
    {
        bool hayVenta = false;
        bool haySubasta = false;
        Publicacion? publicacion = null;
        int indice = 0;
        while (indice < _publicaciones.Count && !hayVenta && !haySubasta)
        {
            if (id == _publicaciones[indice].Id) // Si la lista de ids contiene algúna publicación
            {
```

```
            if ((esUnicamenteVenta && !esUnicamenteSubasta) || (!esUnicamenteVenta
&& !esUnicamenteSubasta))
            {
               if (_publicaciones[indice] is Venta venta)
               {
                  hayVenta = true;
                  publicacion = venta; // Se asigna la publicación
               }
            }
            if ((esUnicamenteSubasta && !esUnicamenteVenta) || (!esUnicamenteVenta
&& !esUnicamenteSubasta))
            {
               if (_publicaciones[indice] is Subasta subasta)
               {
                  haySubasta = true;
                  publicacion = subasta; // Se asigna la publicación
               }
            }
         }
         indice++;
      }
      if (!hayVenta && !haySubasta)
      {
         throw new ArgumentException("No hay ningúna publicación con el id
proporcionado");
      }
      if (!hayVenta && esUnicamenteVenta)
```

```
        {
            throw new ArgumentException("No hay ningúna venta con el id proporcionado");
        }
        if (!haySubasta && esUnicamenteSubasta)
        {
            throw new ArgumentException("No hay ningúna subasta con el id proporcionado");
        }
        return publicacion;
    }
    #endregion
    #region Usuario
    public Usuario? ObtenerUsuarioPorId(int id, bool esUnicamenteCliente, bool esUnicamenteAdministrador)
    {
        bool hayCliente = false;
        bool hayAdministrador = false;
        Usuario? usuario = null;
        int indice = 0;
        while (indice < _usuarios.Count && !hayCliente && !hayAdministrador)
        {
            if (id == _usuarios[indice].Id) // Si la lista de ids contiene algúna usuario
            {
                if ((esUnicamenteCliente && !esUnicamenteAdministrador) || (!esUnicamenteCliente && !esUnicamenteAdministrador))
                {
                    if (_usuarios[indice] is Cliente cliente)
```

```
                {
                    hayCliente = true;
                    usuario = cliente; // Se asigna el usuario
                }
            }
            if ((esUnicamenteAdministrador && !esUnicamenteCliente) ||
(!esUnicamenteCliente && !esUnicamenteAdministrador))
            {
                if (_usuarios[indice] is Administrador administrador)
                {
                    hayAdministrador = true;
                    usuario = administrador; // Se asigna el usuario
                }
            }
        }
        indice++;
    }
    if (!hayCliente && !hayAdministrador)
    {
        throw new ArgumentException("No hay ningún usuario con el id
proporcionado");
    }
    if (!hayCliente && esUnicamenteCliente)
    {
        throw new ArgumentException("No hay ningún cliente con el id proporcionado");
    }
    if (!hayAdministrador && esUnicamenteAdministrador)
```

```
        {
            throw new ArgumentException("No hay ningún administrador con el id proporcionado");
        }
        return usuario;
    }
    public Usuario? ObtenerUsuarioPorEmailYContrasenia(string email, string contrasenia, bool esUnicamenteCliente, bool esUnicamenteAdministrador)
    {
        bool hayCliente = false;
        bool hayAdministrador = false;
        Usuario? usuario = null;
        int indice = 0;
        while (indice < _usuarios.Count && !hayCliente && !hayAdministrador)
        {
            if (email.Contains(_usuarios[indice].Email) && contrasenia.Contains(_usuarios[indice].Contrasenia)) // Si la lista de email y contraseñas contiene algún usuario
            {
                if ((esUnicamenteCliente && !esUnicamenteAdministrador) || (!esUnicamenteCliente && !esUnicamenteAdministrador))
                {
                    if (_usuarios[indice] is Cliente cliente)
                    {
                        hayCliente = true;
                        usuario = cliente; // Se asigna el usuario
                    }
                }
```

```
            if ((esUnicamenteAdministrador && !esUnicamenteCliente) ||
(!esUnicamenteCliente && !esUnicamenteAdministrador))
            {
                if (_usuarios[indice] is Administrador administrador)
                {
                    hayAdministrador = true;
                    usuario = administrador; // Se asigna el usuario
                }
            }
        }
        indice++;
    }
    if (!hayCliente && !hayAdministrador)
    {
        throw new ArgumentException("No hay ningún usuario con el email y contraseña
proporcionados");
    }
    if (!hayCliente && esUnicamenteCliente)
    {
        throw new ArgumentException("No hay ningún cliente con el email y contraseña
proporcionados");
    }
    if (!hayAdministrador && esUnicamenteAdministrador)
    {
        throw new ArgumentException("No hay ningún administrador con el email y
contraseña proporcionados");
    }
```

```
        return usuario;
    }
    #endregion
    #endregion

    /// <summary>
    /// Las funciones de alta se encargan de llamar a los constructores de las
    /// diferentes clases y pasar los parametros obtenidos en Program.
    /// </summary>
    #region Altas
    #region Articulo
    public void AltaArticulo(string nombre, decimal precio, string categoria)
    {
        Articulo nuevoArticulo = new Articulo(nombre, precio, categoria);
        // Validación de la relacion entre los datos ingresados
        nuevoArticulo.Validar();
        // Si los datos son validos entonces se registra el Articulo
        if (!_articulos.Contains(nuevoArticulo))
        {
            _articulos.Add(nuevoArticulo);
        }
        else
        {
            throw new ArgumentException("Ya existe un articulo registrado con el nombre
proporcionado");
        }
```

```
    }
    #endregion
    #region Publicacion
    public void AltaVenta(string nombre, string estado, DateTime fecha, List<Articulo>
articulos, Cliente? cliente, DateTime fechaFin, bool ofertaRelampago)
    {
        Venta nuevaVenta = new Venta(nombre, estado, fecha, articulos, cliente, fechaFin,
ofertaRelampago);
        // Validación de la relacion entre los datos ingresados
        nuevaVenta.Validar();
        // Si los datos son validos entonces se registra la Venta
        if (!_publicaciones.Contains(nuevaVenta))
        {
            _publicaciones.Add(nuevaVenta);
        }
        else
        {
            throw new ArgumentException("Ya existe una publicacion registrada con el
nombre proporcionado");
        }
    }
    public void AltaSubasta(string nombre, string estado, DateTime fecha, List<Articulo>
articulos, Cliente? cliente, Administrador? administrador, DateTime fechaFin, List<Oferta>
ofertas)
    {
        Subasta nuevaSubasta = new Subasta(nombre, estado, fecha, articulos, cliente,
administrador, fechaFin, ofertas);
        // Validación de la relacion entre los datos ingresados
```

```
            nuevaSubasta.Validar();
            // Si los datos son validos entonces se registra la Subasta
            if (!_publicaciones.Contains(nuevaSubasta))
            {
                _publicaciones.Add(nuevaSubasta);
            }
            else
            {
                throw new ArgumentException("Ya existe una publicacion registrada con el nombre proporcionado");
            }
        }
        #endregion
        #region Usuario
        public void AltaCliente(string nombre, string apellido, string email, string contrasenia, decimal saldo)
        {
            Cliente nuevoCliente = new Cliente(nombre, apellido, email, contrasenia, saldo);
            // Validación de la relacion entre los datos ingresados
            nuevoCliente.Validar();
            // Si los datos son validos entonces se registra el Cliente
            if (!_usuarios.Contains(nuevoCliente))
            {
                _usuarios.Add(nuevoCliente);
            }
            else
            {
```

```
            throw new ArgumentException("Ya existe un usuario registrado con el nombre y apellido proporcionados");
        }
    }
    public void AltaAdministrador(string nombre, string apellido, string email, string contrasenia)
    {
        Administrador nuevoAdministrador = new Administrador(nombre, apellido, email, contrasenia);
        // Validación de la relacion entre los datos ingresados
        nuevoAdministrador.Validar();
        // Si los datos son validos entonces se registra el Administrador
        if (!_usuarios.Contains(nuevoAdministrador))
        {
            _usuarios.Add(nuevoAdministrador);
        }
        else
        {
            throw new ArgumentException("Ya existe un usuario registrado con el nombre y apellido proporcionados");
        }
    }
    #endregion
    #region Ofertas
    public void AltaOferta(Usuario? usuario, Publicacion? publicacion, decimal monto, DateTime fecha)
    {
        if (publicacion != null && publicacion is Subasta subasta)
```

```
        {
            subasta.AltaOferta(usuario, monto, fecha);
        }
        else if (publicacion == null)
        {
            throw new ArgumentNullException("No fue posible acceder a la subasta proporcionada");
        }
        else
        {
            throw new ArgumentException("No fue posible realizar la oferta debido a que no fue posible identificar la subasta");
        }
    }
    #endregion
    #endregion

    /// <summary>
    /// Las funciones de transacción se encargan de cobrar y hacer la lógica de compra
    /// </summary>
    #region Transacciones
    public void CompraVenta(Cliente? cliente, Venta? venta)
    {
        // Cambia de estado la venta, registra el Cliente que la compró y la fecha de fin
        venta.Estado = "CERRADA";
        venta.Cliente = cliente;
        venta.FechaFin = DateTime.Now;
```

```
    // Cobra el valor de la venta al cliente
    decimal precioVenta = ConsultarPrecioVenta(venta, venta.Articulos);
    cliente.Saldo -= precioVenta;
}
public void CompraSubasta(Administrador? administrador, Subasta? subasta)
{
    // Variable para determinar el cliente con el mayor monto que puede pagar
    bool fueCobrada = false;

    // Cobra el valor de la venta al cliente con la oferta más alta y saldo disponible
    for (int i = subasta.Ofertas.Count - 1; i >= 0 || !fueCobrada; i--)
    {
        Cliente? clienteActual = subasta.Ofertas[i].Usuario as Cliente;

        if (clienteActual.Saldo >= subasta.Ofertas[i].Monto)
        {
            // Cambia de estado la subasta y registra el cliente que la ganó
            subasta.Estado = "CERRADA";
            subasta.Cliente = clienteActual;
            // Registra el Administrador que cerro la subasta y la fecha fin
            subasta.Administrador = administrador;
            subasta.FechaFin = DateTime.Now;

            clienteActual.Saldo -= subasta.Ofertas[i].Monto;
```

```
            fueCobrada = true;
        }
    }
}
#endregion

/// <summary>
/// Las funciones de consulta tienen el objetivo de obtener datos calculados.
/// Por ejemplo ConsultarPrecioVentaDeListaVenta obtiene los precios de las ventas buscadas.
/// Este es un dato calculado ya que es necesario acceder a la venta y sumar el precio de todos sus articulos.
/// </summary>
#region Consultas
public decimal ConsultarPrecioVenta(Publicacion? publicacion, List<Articulo> articulos)
{
    decimal precio = 0;
    for (int i = 0; i < articulos.Count; i++)
    {
        precio += articulos[i].Precio;
    }
    if (publicacion is Venta venta)
    {
        if (venta.OfertaRelampago)
        {
            // se aplica descuento si corresponde a la venta en especifico
```

```
            precio = precio * 80 / 100;
        }
    }
    return precio;
}
#endregion

/// <summary>
/// Las precargas son relizadas a travez de las funciones de alta,
/// esto se hace de este modo para que el id autoincremental se asigne correctamente
/// </summary>
#region Precargas
#region Articulo
private void PrecargaArticulo()
{
    AltaArticulo("Pelota de football", 450, "Football");
    AltaArticulo("Camiseta deportiva", 1200, "Deporte");
    AltaArticulo("Zapatillas treking", 3500, "Treking");
    AltaArticulo("Raqueta de tenis", 4200, "Tenis");
    AltaArticulo("Balón de basquetball", 800, "Basquetball");
    AltaArticulo("Guantes de boxeo", 2200, "Boxeo");
    AltaArticulo("Casco de ciclismo", 1800, "Ciclismo");
    AltaArticulo("Saco de dormir", 2300, "Camping");
    AltaArticulo("Bolsa de gimnasio", 950, "Boxeo");
    AltaArticulo("Bicicleta de montaña", 15000, "Ciclismo");
    AltaArticulo("Mochila de trekking", 2100, "Treking");
```

```
AltaArticulo("Protector solar", 320, "Playa");
AltaArticulo("Botella térmica", 750, "Camping");
AltaArticulo("Palo de hockey", 1700, "Hokey");
AltaArticulo("Pesas ajustables", 3000, "Gimnasio");
AltaArticulo("Cinta para correr", 25000, "Gimnasio");
AltaArticulo("Guantes de arquero", 900, "Arquería");
AltaArticulo("Tabla de surf", 12000, "Surf");
AltaArticulo("Canilleras", 600, "Football");
AltaArticulo("Traje de neopreno", 5400, "Surf");
AltaArticulo("Gafas de natación", 650, "Natación");
AltaArticulo("Bola de bowling", 3500, "Bowling");
AltaArticulo("Skateboard", 2400, "Skating");
AltaArticulo("Patines en línea", 2900, "Patinaaje");
AltaArticulo("Salvavidas", 1200, "Playa");
AltaArticulo("Set de pesas", 4200, "Gimnasio");
AltaArticulo("Cuerda para saltar", 300, "Gimnasio");
AltaArticulo("Bicicleta de carrera", 18500, "Ciclismo");
AltaArticulo("Tobilleras con peso", 850, "Gimnasio");
AltaArticulo("Set de dardos", 400, "Juegos");
AltaArticulo("Bate de baseball", 1900, "Baseball");
AltaArticulo("Bola de volleyball", 850, "Volleyball");
AltaArticulo("Aro de basquetball", 2700, "Basquetball");
AltaArticulo("Zapatilla de ciclismo", 1900, "Ciclismo");
AltaArticulo("Silla de camping", 1100, "Camping");
AltaArticulo("Sombrilla", 1600, "Playa");
AltaArticulo("Tienda de campaña", 8700, "Camping");
```

```
            AltaArticulo("Colchoneta de yoga", 1200, "Deporte");
            AltaArticulo("Barra de dominadas", 1900, "Gimnasio");
            AltaArticulo("Malla", 600, "Ciclismo");
            AltaArticulo("Reloj deportivo", 6500, "Deporte");
            AltaArticulo("Monopatín eléctrico", 18000, "Ciclismo");
            AltaArticulo("Kit de pesca", 3200, "Pesca");
            AltaArticulo("Bolsa de golf", 7600, "Golf");
            AltaArticulo("Raqueta de badminton", 1600, "Badminton");
            AltaArticulo("Patineta longboard", 3300, "Skating");
            AltaArticulo("Bola de rugby", 1050, "Rugby");
            AltaArticulo("Kit de snorkel", 1800, "Natacion");
            AltaArticulo("Camiseta de compresión", 1300, "Deporte");
            AltaArticulo("Gorra deportiva", 400, "Deporte");
            AltaArticulo("Balón medicinal", 2000, "Salud");
            AltaArticulo("Kit de arquería", 9800, "Arquería");
            AltaArticulo("Soga de escalada", 5600, "Escalada");
            AltaArticulo("Casco de ski", 3700, "Ski");
            AltaArticulo("Balde", 1050, "Playa");
            AltaArticulo("Gafas de ciclismo", 900, "Ciclismo");
        }
        #endregion
        #region Publicacion
        private void PrecargarPublicacion()
        {
            AltaVenta("Verano en la playa", "ABIERTA", DateTime.ParseExact("05/01/2024",
"dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 11, 24, 35, 54 }), null,
DateTime.MinValue, false);
```

AltaVenta("Juego gimnasio", "ABIERTA", DateTime.ParseExact("13/12/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 14, 15, 25, 26, 28, 38 }), null, DateTime.MinValue, false);

AltaVenta("Caminata en el bosque", "ABIERTA", DateTime.ParseExact("12/02/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 1, 3, 4, 5 }), null, DateTime.MinValue, false);

AltaVenta("Paseo en bicicleta", "ABIERTA", DateTime.ParseExact("15/03/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 6, 8, 9, 10 }), null, DateTime.MinValue, false);

AltaVenta("Clase de yoga", "ABIERTA", DateTime.ParseExact("22/04/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 12, 13, 16, 18, 20 }), null, DateTime.MinValue, false);

AltaVenta("Día de spa", "ABIERTA", DateTime.ParseExact("30/05/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 21, 22, 23, 29 }), null, DateTime.MinValue, true);

AltaVenta("Concierto al aire libre", "ABIERTA", DateTime.ParseExact("01/08/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 30, 31, 32, 34, 37 }), null, DateTime.MinValue, false);

AltaVenta("Cata de vinos", "ABIERTA", DateTime.ParseExact("10/09/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 40, 41, 42 }), null, DateTime.MinValue, false);

AltaVenta("Taller de pintura", "CERRADA", DateTime.ParseExact("15/10/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 43, 44, 45, 46 }), ObtenerUsuarioPorId(3, true, false) as Cliente, DateTime.ParseExact("05/11/2024", "dd/MM/yyyy", null), false);

AltaVenta("Excursión a la montaña", "CERRADA", DateTime.ParseExact("25/11/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 47, 48, 49 }), ObtenerUsuarioPorId(3, true, false) as Cliente, DateTime.ParseExact("26/11/2024", "dd/MM/yyyy", null), false);

AltaSubasta("Vuelta ciclista", "CERRADA", DateTime.ParseExact("06/01/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 27, 33, 39 }), ObtenerUsuarioPorId(8, true, false) as Cliente, ObtenerUsuarioPorId(1, false, true) as Administrador, DateTime.ParseExact("30/07/2024", "dd/MM/yyyy", null), new List<Oferta>());

```
        AltaSubasta("Set camping", "ABIERTA", DateTime.ParseExact("21/07/2024",
"dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 7, 34 ,36 }), null, null,
DateTime.MinValue, new List<Oferta>());

        AltaSubasta("Torneo de ajedrez", "ABIERTA", DateTime.ParseExact("12/03/2024",
"dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 50, 51, 52 }), null, null,
DateTime.MinValue, new List<Oferta>());

        AltaSubasta("Subasta de arte", "ABIERTA", DateTime.ParseExact("20/04/2024",
"dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 51, 53, 54 }), null, null,
DateTime.MinValue, new List<Oferta>());

        AltaSubasta("Rally de coches", "ABIERTA", DateTime.ParseExact("01/06/2024",
"dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 36, 37, 38 }), null, null,
DateTime.MinValue, new List<Oferta>());

        AltaSubasta("Subasta de antigüedades", "ABIERTA",
DateTime.ParseExact("15/07/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new
List<int> { 29, 20, 21 }), null, null, DateTime.MinValue, new List<Oferta>());

        AltaSubasta("Concurso de cocina", "ABIERTA", DateTime.ParseExact("05/08/2024",
"dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 42, 43, 44 }), null, null,
DateTime.MinValue, new List<Oferta>());

        AltaSubasta("Maratón de lectura", "ABIERTA", DateTime.ParseExact("12/09/2024",
"dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 45, 46, 47 }), null, null,
DateTime.MinValue, new List<Oferta>());

        AltaSubasta("Competencia de fotografía", "ABIERTA",
DateTime.ParseExact("30/10/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new
List<int> { 18, 19, 20 }), null, null, DateTime.MinValue, new List<Oferta>());

        AltaSubasta("Fiesta de disfraces", "ABIERTA", DateTime.ParseExact("15/11/2024",
"dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 21, 22, 23 }), null, null,
DateTime.MinValue, new List<Oferta>());

    }

    #endregion

    #region Usuario

    private void PrecargaUsuario()

    {
```

```
        AltaAdministrador("Valentin", "Latorre", "ValentinLatorre@Gmail.com",
"Valentin1234");

        AltaAdministrador("Agustin", "Butrico", "AgustinButrico@gmail.com",
"Agustin1234");

        AltaCliente("Juan", "Peres", "Juanperes@hmail.com", "Juan1234", 5600);

        AltaCliente("Esteban", "Lopez", "EstebanLopez@hmail.com", "5566AS43", 27000);

        AltaCliente("Carlos", "Medina", "CarlosMedina@hmail.com", "Medina1234", 7500);

        AltaCliente("Mariano", "Morales", "MarianoMorales@hmail.com", "Mariano2",
5000);

        AltaCliente("Estela", "Rosales", "EstelaRosales@hmail.com", "Rosalia46", 1700);

        AltaCliente("Marcos", "Sauce", "MarcosSauce@hmail.com", "Sauce311", 30000);

        AltaCliente("Lucia", "Gomez", "LuciaGomezs@hmail.com", "Lucia1990", 7200);

        AltaCliente("Rodrigo", "Barrios", "RodrigoBarrios@hmail.com", "RodrigoBarrios12",
900);

        AltaCliente("Pepe", "Argento", "PepeArgento@gmail.com", "PepeArgento1113",
3300);

        AltaCliente("Felipe", "Castañeda", "FelipeCastañeda@gmail.com",
"FeliCastañeda032", 3300);

    }

    #endregion

    #region Oferta

    private void PrecargaOferta()

    {

        AltaOferta(ObtenerUsuarioPorId(3, true, false), ObtenerPublicacionPorId(10, false,
true), 120, DateTime.ParseExact("06/01/2024", "dd/MM/yyyy", null));

        AltaOferta(ObtenerUsuarioPorId(6, true, false), ObtenerPublicacionPorId(10, false,
true), 1500, DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

        AltaOferta(ObtenerUsuarioPorId(4, true, false), ObtenerPublicacionPorId(10, false,
true), 3400, DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));
```

```
            AltaOferta(ObtenerUsuarioPorId(8, true, false), ObtenerPublicacionPorId(10, false,
true), 3500, DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

            AltaOferta(ObtenerUsuarioPorId(8, true, false), ObtenerPublicacionPorId(11, false,
true), 100, DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

            AltaOferta(ObtenerUsuarioPorId(5, true, false), ObtenerPublicacionPorId(11, false,
true), 500, DateTime.ParseExact("21/07/2024", "dd/MM/yyyy", null));

            AltaOferta(ObtenerUsuarioPorId(3, true, false), ObtenerPublicacionPorId(11, false,
true), 20000, DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

            AltaOferta(ObtenerUsuarioPorId(6, true, false), ObtenerPublicacionPorId(12, false,
true), 200, DateTime.ParseExact("12/03/2024", "dd/MM/yyyy", null));

            AltaOferta(ObtenerUsuarioPorId(8, true, false), ObtenerPublicacionPorId(12, false,
true), 400, DateTime.ParseExact("20/04/2024", "dd/MM/yyyy", null));

            AltaOferta(ObtenerUsuarioPorId(7, true, false), ObtenerPublicacionPorId(12, false,
true), 700, DateTime.ParseExact("01/06/2024", "dd/MM/yyyy", null));

            AltaOferta(ObtenerUsuarioPorId(10, true, false), ObtenerPublicacionPorId(15, false,
true), 300, DateTime.ParseExact("05/08/2024", "dd/MM/yyyy", null));

            AltaOferta(ObtenerUsuarioPorId(9, true, false), ObtenerPublicacionPorId(15, false,
true), 600, DateTime.ParseExact("15/07/2024", "dd/MM/yyyy", null));

            AltaOferta(ObtenerUsuarioPorId(11, true, false), ObtenerPublicacionPorId(17, false,
true), 450, DateTime.ParseExact("12/09/2024", "dd/MM/yyyy", null));

            AltaOferta(ObtenerUsuarioPorId(3, true, false), ObtenerPublicacionPorId(17, false,
true), 1550, DateTime.ParseExact("30/10/2024", "dd/MM/yyyy", null));

            AltaOferta(ObtenerUsuarioPorId(4, true, false), ObtenerPublicacionPorId(17, false,
true), 1600, DateTime.ParseExact("30/10/2024", "dd/MM/yyyy", null));

        }

        #endregion

        #endregion

    }

}
```

SUBASTA.CS

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

namespace LogicaNegocio
{
    public class Subasta : Publicacion
    {
        #region Atributos de la clase
        private Administrador? _administrador; // Inicializado con una instancia por defecto
        private List<Oferta> _ofertas = new List<Oferta>(); // Inicializado con una lista vacía
        #endregion

        #region Propiedades
        public Administrador? Administrador
        {
            get { return _administrador; }
            set { _administrador = value; }
        }
        public List<Oferta> Ofertas
        {
            get { return _ofertas; }
            set { _ofertas = value; }
```

```
    }
    #endregion

    #region Constructor
    public Subasta(string nombre, string estado, DateTime fecha, List<Articulo> articulos, Cliente? cliente, Administrador? administrador, DateTime fechaFin, List<Oferta> ofertas)
        : base(nombre, estado, fecha, articulos, cliente, fechaFin) // Llamada al constructor de la clase base (Publicacion)
    {
        Administrador = administrador;
        Ofertas = ofertas;
    }
    #endregion

    #region Validación
    // Validación de Subasta, hereda de Publicacion
    public override void Validar()
    {
    }
    #endregion

    #region Método Equals
    // Sobre escritura del metodo Equals que es usado por Contains
    public override bool Equals(object? obj)
    {
        if (obj != null && obj is Subasta)
        {
```

```
            Subasta subasta = (Subasta)obj;
            return Nombre == subasta.Nombre;
        }
        return false;
    }
    #endregion

    #region Alta
    public void AltaOferta(Usuario? usuario, decimal monto, DateTime fecha)
    {
        Oferta oferta = new Oferta(usuario, monto, fecha); // Crea una oferta con el costructor de Oferta
        if (!Ofertas.Contains(oferta)) // Utilizando el Equals de Oferta valida que un usuario no haga más de una oferta
        {
            Ofertas.Add(oferta); // Añade a la lista _ofertas
        }
    }
    #endregion
  }
}
```

USUARIO.CS

```
using LogicaNegocio.Interfaces;
using System;
using System.Collections.Generic;
```

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

namespace LogicaNegocio
{
    public class Usuario : IValidate
    {
        #region Atributos de la clase
        private int _id;
        private static int s_ultId = 0; // Inicializado con el id siguiente a la ultima precarga
        private string _nombre = string.Empty; // Inicializado con una cadena vacía
        private string _apellido = string.Empty; // Inicializado con una cadena vacía
        private string _email = string.Empty; // Inicializado con una cadena vacía
        private string _contrasenia = string.Empty; // Inicializado con una cadena vacía
        #endregion

        #region Propiedades
        public int Id
        {
            get { return _id; }  // Solo lectura, asignado internamente.
        }
        public string Nombre
        {
            get { return _nombre; }
            set { _nombre = EvaluarNombre(value); }
```

```
}
public string Apellido
{
   get { return _apellido; }
   set { _apellido = EvaluarApellido(value); }
}
public string Email
{
   get { return _email; }
   set { _email = EvaluarEmail(value); }
}
public string Contrasenia
{
   get { return _contrasenia; }
   set { _contrasenia = EvaluarContrasenia(value); }
}
#endregion

#region Constructor
public Usuario(string nombre, string apellido, string email, string contrasenia)
{
   _id = Usuario.s_ultId; // Asigna el ID único
   Usuario.s_ultId++; // Incrementa el ID único
   Nombre = nombre;
   Apellido = apellido;
   Email = email;
```

```
    Contrasenia = contrasenia;
}
#endregion

#region Validación
// Evaluaciones
private static string EvaluarNombre(string nombre)
{
    if (string.IsNullOrEmpty(nombre))
    {
        throw new ArgumentException("El nombre no puede ser vacío");
    }
    return nombre;
}
private static string EvaluarApellido(string apellido)
{
    if (string.IsNullOrEmpty(apellido))
    {
        throw new ArgumentException("El apellido no puede ser vacío");
    }
    return apellido;
}
private static string EvaluarEmail(string email)
{
    if (string.IsNullOrEmpty(email))
    {
```

```
            throw new ArgumentException("El email no puede ser vacío");
        }
        if (email.IndexOf('@') == -1)
        {
            throw new ArgumentException("El email debe pertenecer a un domino (debe tener @)");
        }
        return email;
    }
    private static string EvaluarContrasenia(string contrasenia)
    {
        if (string.IsNullOrEmpty(contrasenia))
        {
            throw new ArgumentException("El contrasenia no puede ser vacío");
        }
        if (contrasenia.Length < 8)
        {
            throw new ArgumentException("La contraseña debe tener al menos 8 caracteres");
        }
        if (!contrasenia.Any(char.IsLetter))
        {
            throw new ArgumentException("La contraseña debe incluir al menos una letra");
        }
        if (!contrasenia.Any(char.IsDigit))
        {
```

```
            throw new ArgumentException("La contraseña debe incluir al menos un
número");
        }
        return contrasenia;
    }

    // Validación de Usuario, es virtual ya que le hereda a otras clases
    public virtual void Validar()
    {
    }
    #endregion

    #region Método Equals
    // Sobre escritura del metodo Equals que es usado por Contains
    public override bool Equals(object? obj)
    {
        if (obj != null && obj is Usuario)
        {
            Usuario usuario = (Usuario)obj;
            return Nombre == usuario.Nombre && Apellido == usuario.Apellido;
        }
        return false;
    }
    #endregion
  }
}
```

VENTA.CS

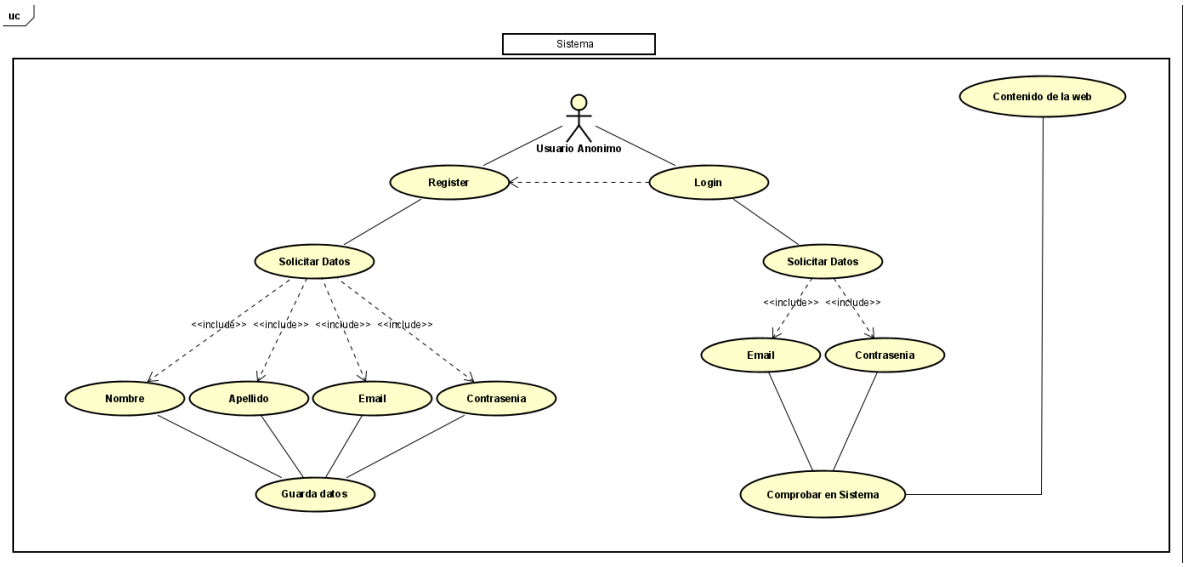
```
using LogicaNegocio.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LogicaNegocio
{
    public class Venta : Publicacion
    {
        #region Atributos de la clase
        private bool _ofertaRelampago = false; // Inicializado en falso
        #endregion

        #region Propiedades
        public bool OfertaRelampago
        {
            get { return _ofertaRelampago; }
            set { _ofertaRelampago = value; }
        }
        #endregion

        #region Constructor
```

```
    public Venta(string nombre, string estado, DateTime fecha, List<Articulo> articulos, Cliente? cliente, DateTime fechaFin, bool ofertaRelampago)
        : base(nombre, estado, fecha, articulos, cliente, fechaFin) // Llamada al constructor de la clase base (Publicacion)
    {
        OfertaRelampago = ofertaRelampago;
    }
    #endregion

    #region Validación
    // Validación de Venta, hereda de Publicacion
    public override void Validar()
    {
    }
    #endregion

    #region Método Equals
    // Sobre escritura del metodo Equals que es usado por Contains
    public override bool Equals(object? obj)
    {
        if (obj != null && obj is Venta)
        {
            Venta venta = (Venta)obj;
            return Nombre == venta.Nombre;
        }
        return false;
    }
```

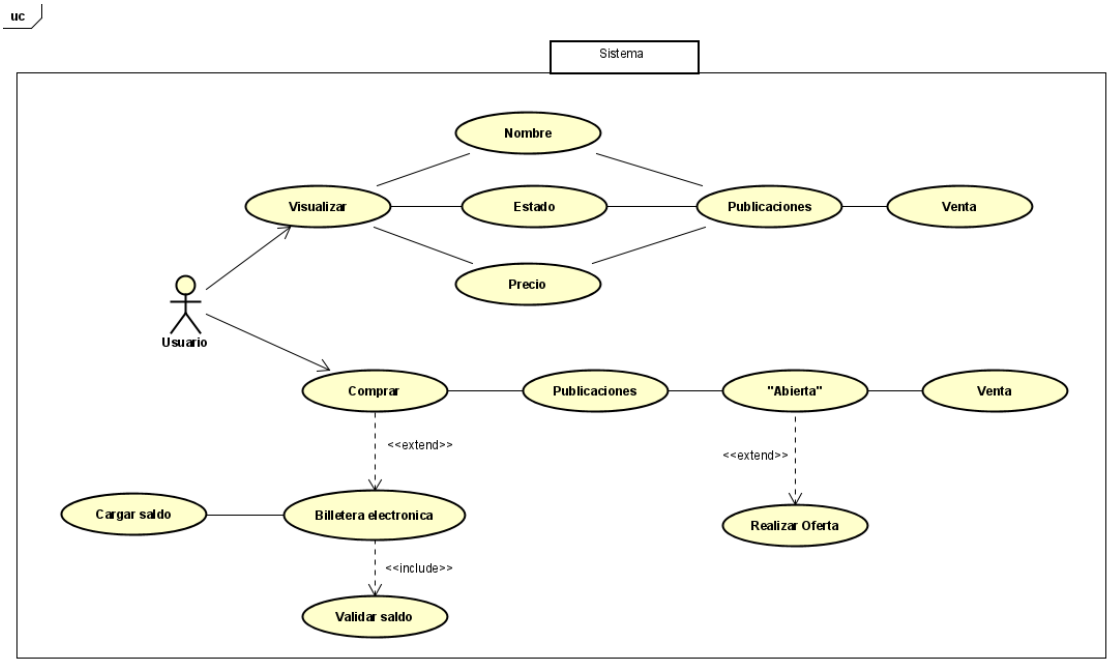
```
        #endregion
    }
}
```

Diagrama de Casos De Uso

Caso de uso Usuario Anónimo



Caso de uso Cliente



Caso de uso Administrador