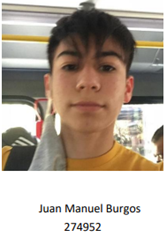
Obligatorio 2

Programación 2



Docente: Joaquín Rodríguez

Fecha de entrega: 22/11/2021

Grupo M2A

Table of Contents

[Diagrama de clases 3](#_Toc88645009)

[Datos precargados 4](#_Toc88645010)

[Código fuente 6](#_Toc88645011)

[Lugar 6](#_Toc88645012)

[Abierto 7](#_Toc88645013)

[Cerrado 9](#_Toc88645014)

[Usuario 11](#_Toc88645015)

[Usuario cliente 13](#_Toc88645016)

[Actividad 14](#_Toc88645017)

[Categoría 17](#_Toc88645018)

[Compra 19](#_Toc88645019)

[Sistema 21](#_Toc88645020)

[Testing 46](#_Toc88645021)

[Usuario no registrado 46](#_Toc88645022)

[Usuario Registrado 48](#_Toc88645023)

[Usuario Operador 49](#_Toc88645024)

# Diagrama de clases

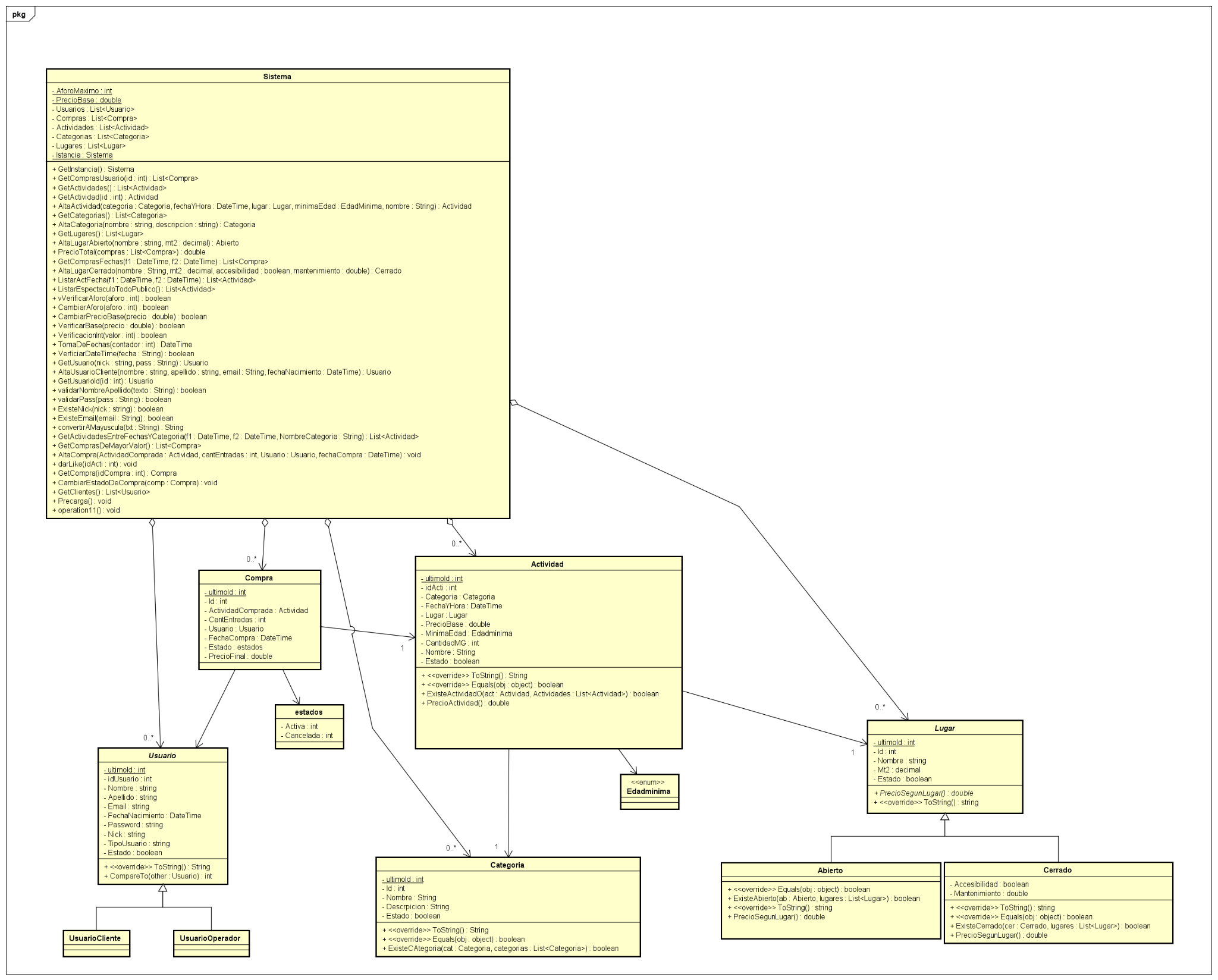


Diagrama realizado en Astah UML, mostrando las clases y sus correspondientes relaciones y cardinalidades.

# Datos precargados

public void Precarga()

{

Categoria cat1 = AltaCategoria("Cine", "Noche de cine");

Categoria cat2 = AltaCategoria("Teatro", "Stand up");

Categoria cat3 = AltaCategoria("Concierto", "The police");

Categoria cat4 = AltaCategoria("Feria gastronomica", "Varios food trucks");

Abierto a1 = AltaLugarAbierto("Estadio Centenario", 35000);

Abierto a2 = AltaLugarAbierto("Teatro de Verano", 2500);

Abierto a3 = AltaLugarAbierto("Rural del Prado", 2000);

Abierto a4 = AltaLugarAbierto("Plaza de Mayo", 300);

Abierto a5 = AltaLugarAbierto("Faro Punta Carretas", 500);

Cerrado c1 = AltaLugarCerrado("Antel Arena", 800, true, 20000);

Cerrado c2 = AltaLugarCerrado("Teatro Metro", 200, false, 2);

Cerrado c3 = AltaLugarCerrado("Teatro Solis", 700, true, 2);

Cerrado c4 = AltaLugarCerrado("Movie Center", 150, true, 2);

Cerrado c5 = AltaLugarCerrado("Montevideo shopping", 650, true, 2);

Actividad act1 = AltaActividad(cat3, DateTime.Parse("2024-09-21 16:00"), a2, EdadMinima.c16, 10000, "Espectaculo musical");

Actividad act2 = AltaActividad(cat2, DateTime.Parse("2023-08-21 20:00"), c1, EdadMinima.c18, 5800, "Espectaculo de stand up");

Actividad act3 = AltaActividad(cat4, DateTime.Parse("2022-07-16 12:00"), c3, EdadMinima.p, 50, "Evento de gastronomia y animales");

Actividad act4 = AltaActividad(cat4, DateTime.Parse("2022-07-10 10:00"), c5, EdadMinima.p, 150, "Evento de gastronomia");

Actividad act5 = AltaActividad(cat1, DateTime.Parse("2023-09-21 16:20"), c4, EdadMinima.c13, 503, "Rapido y furioso 32");

Actividad act6 = AltaActividad(cat1, DateTime.Parse("2022-10-20 17:00"), a5, EdadMinima.p, 1450, "Cine al aire libre");

Actividad act7 = AltaActividad(cat3, DateTime.Parse("2022-01-11 22:00"), a1, EdadMinima.c18, 324, "Espectaculo musical nro 23");

Actividad act8 = AltaActividad(cat3, DateTime.Parse("2021-12-05 17:00"), a4, EdadMinima.p, 1, "The police en vivo");

Actividad act9 = AltaActividad(cat2, DateTime.Parse("2024-09-21 16:00"), c2, EdadMinima.c16, 1, "Espectaculo de stand up");

Actividad act10 = AltaActividad(cat1, DateTime.Parse("2022-09-21 20:00"), c3, EdadMinima.p, 1, "Cine antiguo");

//Precarga usuarios

UsuarioOperador op1 = new UsuarioOperador("Andres", "Fernandez", "AFernaandez@gmail.com", DateTime.Parse("1994-09-21"), "Abcd1234", "OPERADOR1");

Usuarios.Add(op1);

UsuarioOperador op2 = new UsuarioOperador("Martina", "Acevedo", "MartinaAC@gmail.com", DateTime.Parse("1990-05-11"), "Abcd1234", "OPERADOR2");

Usuarios.Add(op2);

Usuario u1 = AltaUsuarioCliente("Federico", "Perez", "FP@gmail.com", DateTime.Parse("2000-08-29"), "Abcd1234", "USUARIO1");

Usuario u2 = AltaUsuarioCliente("Ana", "Rapal", "rapalA@gmail.com", DateTime.Parse("2005-03-01"), "Abcd1234", "USUARIO2");

//COMPRAS

DateTime ahora = DateTime.Now;

AltaCompra(act10, 3, u1, ahora);

AltaCompra(act6, 2, u1, ahora);

AltaCompra(act8, 5, u1, ahora);

AltaCompra(act3, 1, u2, ahora);

AltaCompra(act5, 4, u2, ahora);

AltaCompra(act1, 4, u2, ahora);

AltaCompra(act2, 3, u1, ahora);

AltaCompra(act3, 2, u1, ahora);

# Código fuente

## Lugar

public abstract class Lugar

{

protected static int ultimoId = 1;

public int Id { get; set; }

public string Nombre { get; set; }

public decimal Mt2 { get; set; }

public bool Estado { get; set; }

//Calcula el precio de la actividad mediante polimorfismo

public abstract double PrecioSegunLugar();

protected Lugar(string nombre, decimal mt2)

{

Nombre = nombre;

Mt2 = mt2;

Estado = true;

}

protected Lugar()

{

}

public abstract override string ToString();

}

## Abierto

public class Abierto : Lugar

{

public Abierto(string nombre, decimal mt2)

{

Id = ultimoId;

ultimoId++;

Nombre = nombre;

Mt2 = mt2;

}

public override bool Equals(object obj)

{

return obj is Abierto abierto &&

Nombre == abierto.Nombre &&

Mt2 == abierto.Mt2;

}

//-------------------------------------------------------------------------

//Verificamos si existe algún lugar abierto

//-------------------------------------------------------------------------

public static bool ExisteAbierto(Abierto ab, List<Lugar> lugares)

{

bool existe = false;

foreach (Lugar item in lugares)

{

if (ab.Equals(item))

{

existe = true;

ultimoId--;

}

}

return existe;

}

public override string ToString()

{

return $"Lugar: {Nombre}\n" +

$"Area mt2: {Mt2}\n";

}

public override double PrecioSegunLugar()

{

double Precio = Sistema.PrecioBase;

if (Mt2 > 1000)

{

Precio += Precio \* 0.10;

}

return Precio;

}

}

## Cerrado

public class Cerrado : Lugar

{

public bool Accesibilidad { get; set; }

public double Mantenimiento { get; set; }

public Cerrado(string nombre, decimal mt2, bool accesibilidad, double mantenimiento)

{

Id = ultimoId;

ultimoId++;

Nombre = nombre;

Mt2 = mt2;

Accesibilidad = accesibilidad;

Mantenimiento = mantenimiento;

}

public override string ToString()

{

return $"Lugar: {Nombre}\n" +

$"Area mt2: {Mt2}\n" +

$"Accesibilidad: {Accesibilidad}\n" +

$"Mantenimiento: {Mantenimiento}\n";

}

public override bool Equals(object obj)

{

return obj is Cerrado cerrado &&

Nombre == cerrado.Nombre &&

Mt2 == cerrado.Mt2 &&

Accesibilidad == cerrado.Accesibilidad &&

Mantenimiento == cerrado.Mantenimiento;

}

//-------------------------------------------------------------------------

//Verificamos si existe lugar cerrado

//-------------------------------------------------------------------------

public static bool ExisteCerrado(Cerrado cer, List<Lugar> lugares)

{

bool existe = false;

foreach (Lugar item in lugares)

{

if (cer.Equals(item))

{

existe = true;

ultimoId--;

}

}

return existe;

}

public override double PrecioSegunLugar()

{

int aforo = Sistema.AforoMaximo;

double Precio = Sistema.PrecioBase;

if (aforo < 50 && aforo > 0)

{

Precio += Precio \* 0.30;

}

else if (aforo > 50 && aforo < 70)

{

Precio += Precio \* 0.15;

}

return Precio;

}

}

## Usuario

public abstract class Usuario : IComparable<Usuario>

{

protected static int ultimoId = 1;

public int idUsuario { get; set; }

public string Nombre { get; set; }

public string Apellido { get; set; }

public string Email { get; set; }

public DateTime FechaNacimiento { get; set; }

public string Password { get; set; }

public string Nick { get; set; }

public string TipoUsuario { get; set; }

public bool Estado { get; set; }

protected Usuario(string nombre, string apellido, string email, DateTime fechaNacimiento, string password, string nick)

{

idUsuario = ultimoId;

ultimoId++;

Nombre = nombre;

Apellido = apellido;

Email = email;

FechaNacimiento = fechaNacimiento;

Password = password;

Nick = nick;

Estado = true;

}

public override string ToString()

{

return $"Nombre {Nombre}\n" +

$"Apellido {Apellido}\n" +

$"Email: {Email}\n" +

$"Fecha de nacimiento: {FechaNacimiento}\n";

}

public int CompareTo([AllowNull] Usuario other)

{

if (this.Nombre.CompareTo(other.Nombre) > 0)

{

return 1;

}

else if (this.Nombre.CompareTo(other.Nombre) < 0)

{

return -1;

}

else

{

if (this.Apellido.CompareTo(other.Apellido) > 0)

{

return 1;

}

else if (this.Apellido.CompareTo(other.Apellido) < 0)

{

return -1;

}

else

{

return 0;

}

}

}

}

## Usuario cliente

namespace Obli\_2\_p2.Models

{

public class UsuarioCliente : Usuario

{

public UsuarioCliente(string nombre, string apellido, string email, DateTime fechaNacimiento, string password, string nick) : base(nombre, apellido, email, fechaNacimiento, password, nick)

{

TipoUsuario = "cliente";

}

}

Usuario operador

public class UsuarioOperador : Usuario

{

public UsuarioOperador(string nombre, string apellido, string email, DateTime fechaNacimiento, string password, string nick) : base(nombre, apellido, email, fechaNacimiento, password, nick)

{

TipoUsuario = "operador";

}

}

## Actividad

public class Actividad

{

private static int ultimoId = 1;

public int idActi { get; set; }

public Categoria Categoria { get; set; }

public DateTime FechaYHora { get; set; }

public Lugar Lugar { get; set; }

public double PrecioBase { get; set; }

public EdadMinima MinimaEdad { get; set; }

public int CantidadMG { get; set; }

public string Nombre { get; set; }

public bool Estado { get; set; }

public enum EdadMinima

{

p = 1,

c13 = 2,

c16 = 3,

c18 = 4,

unknown = 5

}

public Actividad(Categoria categoria, DateTime fechaYHora, Lugar lugar, EdadMinima minimaEdad, int cantidadMG, string nombre)

{

idActi = ultimoId;

ultimoId++;

Categoria = categoria;

FechaYHora = fechaYHora;

Lugar = lugar;

PrecioBase = Sistema.PrecioBase;

MinimaEdad = minimaEdad;

CantidadMG = cantidadMG;

Nombre = nombre;

Estado = true;

}

// recibe un decimal de lugar que se considera un descuento para asi multiplicarlo por el precio base y obtener el precio final

public override string ToString()

{

return $"Categoria: {Categoria}" +

$"Fecha y hora: {FechaYHora}\n" +

$"Lugar actividad: {Lugar}" +

$"Precio base: {PrecioBase}\n" +

$"Categoría edad: {MinimaEdad}\n";

}

public override bool Equals(object obj)

{

return obj is Actividad actividad &&

EqualityComparer<Categoria>.Default.Equals(Categoria, actividad.Categoria) &&

FechaYHora == actividad.FechaYHora &&

EqualityComparer<Lugar>.Default.Equals(Lugar, actividad.Lugar) &&

PrecioBase == actividad.PrecioBase &&

MinimaEdad == actividad.MinimaEdad &&

CantidadMG == actividad.CantidadMG &&

Nombre == actividad.Nombre;

}

//-------------------------------------------------------------------------

//Verificamos que exista una actividad

//-------------------------------------------------------------------------

public static bool ExisteActividad(Actividad act, List<Actividad> actividades)

{

bool validado = false;

foreach (Actividad item in actividades)

{

if (act.Equals(item))

{

validado = true;

}

}

return validado;

}

//Calcula el precio final de cada actividad

public double PrecioActividad()

{

return Lugar.PrecioSegunLugar();

}

}

## Categoría

public class Categoria

{

private static int ultimoId = 1;

public int Id { get; set; }

public string Nombre { get; set; }

public string Descripcion { get; set; }

public bool Estado { get; set; }

public Categoria(string nombre, string descripcion)

{

Id = ultimoId;

ultimoId++;

Nombre = nombre;

Descripcion = descripcion;

Estado = true;

}

public override string ToString()

{

return $"Nombre: {Nombre} Descripcion: {Descripcion}\n";

}

public override bool Equals(object obj)

{

return obj is Categoria categoria &&

Nombre == categoria.Nombre &&

Descripcion == categoria.Descripcion;

}

//-------------------------------------------------------------------------

//Verificamos que exista la categoría

//-------------------------------------------------------------------------

public static bool ExisteCategoria(Categoria cat, List<Categoria> categorias)

{

bool validado = false;

foreach (Categoria item in categorias)

{

if (cat.Equals(item))

{

validado = true;

ultimoId--;

}

}

return validado;

}

## Compra

public class Compra

{

private static int ultimoId = 1;

public int Id { get; set; }

public Actividad ActividadComprada { get; set; }

public int CantEntradas { get; set; }

public Usuario Usuario { get; set; }

public DateTime FechaCompra { get; set; }

public Estados Estado { get; set; }

public double PrecioFinal { get; set; }

public enum Estados

{

activa = 1,

cancelada = 2

}

public Compra(Actividad actividadComprada, int cantEntradas, Usuario usuario, DateTime fechaCompra, double precioFinal)

{

Id = ultimoId;

ultimoId++;

ActividadComprada = actividadComprada;

CantEntradas = cantEntradas;

Usuario = usuario;

FechaCompra = fechaCompra;

Estado = Estados.activa;

PrecioFinal = precioFinal;

}

public override string ToString()

{

return $"Nombre de actividad comprada: {ActividadComprada.Nombre}\n" +

$"Cantidad de entradas compradas: {CantEntradas}\n" +

$"Adquiere: {Usuario.Nombre}\n" +

$"Fecha de compra: {FechaCompra}\n" +

$"Estado de la compra: {Estado}\n";

}

public double PrecioTotal()

{

double total = -1;

double pa = ActividadComprada.PrecioActividad();

total = pa \* CantEntradas;

return total;

}

}

## Sistema

public class Sistema

{

public static int AforoMaximo { get; set; }

public static double PrecioBase { get; set; }

public List<Usuario> Usuarios = new List<Usuario>();

public List<Compra> Compras = new List<Compra>();

public List<Actividad> Actividades = new List<Actividad>();

public List<Categoria> Categorias = new List<Categoria>();

public List<Lugar> Lugares = new List<Lugar>();

//Constructor sistema

private Sistema()

{

AforoMaximo = 100;

PrecioBase = 1500;

Precarga();

}

//Singleton

private static Sistema instancia = null;

public static Sistema GetInstancia()

{

if (instancia == null)

{

instancia = new Sistema();

}

return instancia;

}

public List<Compra> GetComprasUsuario(int? id)

{

List<Compra> ComprasHechas = new List<Compra>();

foreach (Compra c in Compras)

{

if (c.Usuario.idUsuario.Equals(id) && c.Estado == Compra.Estados.activa)

{

ComprasHechas.Add(c);

}

}

return ComprasHechas;

}

public List<Actividad> GetActividades()

{

return Actividades;

}

//Busca y devuelve una actividad 11-11-2021 Hora: 15:12 pm

public Actividad GetActividad(int id)

{

Actividad buscada = null;

foreach (Actividad act in Actividades)

{

if (act.idActi.Equals(id))

{

buscada = act;

}

}

return buscada;

}

//-------------------------------------------------------------------------------

//Altas verificando que no se repitan instancias (c/u con sus respectivos existe)

//-------------------------------------------------------------------------------

public Actividad AltaActividad(Categoria categoria, DateTime fechaYHora, Lugar lugar, EdadMinima minimaEdad, int cantidadMG, string nombre)

{

Actividad nuevaActividad = null;

if (categoria != null && lugar != null && fechaYHora != null && minimaEdad != EdadMinima.unknown && nombre != "" && cantidadMG >= 0)

{

nuevaActividad = new Actividad(categoria, fechaYHora, lugar, minimaEdad, cantidadMG, nombre);

if (!Actividad.ExisteActividad(nuevaActividad, GetActividades()))

{

Actividades.Add(nuevaActividad);

}

}

return nuevaActividad;

}

public List<Categoria> GetCategorias()

{

return Categorias;

}

public Categoria AltaCategoria(string nombre, string descripcion)

{

Categoria nuevaCategoria = null;

if (nombre != "" && descripcion != "")

{

nuevaCategoria = new Categoria(nombre, descripcion);

if (Categoria.ExisteCategoria(nuevaCategoria, GetCategorias()) == false)

{

Categorias.Add(nuevaCategoria);

}

}

return nuevaCategoria;

}

public List<Lugar> GetLugares()

{

return Lugares;

}

public Abierto AltaLugarAbierto(string nombre, decimal mt2)

{

Abierto nuevoAbierto = null;

if (nombre != null && mt2 > 0)

{

nuevoAbierto = new Abierto(nombre, mt2);

if (Abierto.ExisteAbierto(nuevoAbierto, GetLugares()) == false)

{

Lugares.Add(nuevoAbierto);

}

}

return nuevoAbierto;

}

public double PrecioTotal(List<Compra> compras)

{

double res = 0;

foreach (Compra c in compras)

{

res+= c.PrecioFinal;

}

return res;

}

public List<Compra> GetComprasFechas(DateTime f1, DateTime f2)

{

List<Compra> res = new List<Compra>();

if (f1 != null && f2 != null && f1 != f2)

{

if (f1 > f2)

{

DateTime aux = f1;

f1 = f2;

f2 = aux;

}

foreach (Compra c in Compras)

{

if (c.FechaCompra > f1 && c.FechaCompra < f2)

{

res.Add(c);

}

}

}

return res;

}

public Cerrado AltaLugarCerrado(string nombre, decimal mt2, bool accesibilidad, double mantenimiento)

{

Cerrado nuevoCerrado = null;

if (nombre != "" && mt2 > 0 && accesibilidad != null && mantenimiento >= 0)

{

nuevoCerrado = new Cerrado(nombre, mt2, accesibilidad, mantenimiento);

if (!Cerrado.ExisteCerrado(nuevoCerrado, GetLugares()) == false)

{

Lugares.Add(nuevoCerrado);

}

}

return nuevoCerrado;

}

//-------------------------------------------------------------------------

//Lista las actividades entre dos fechas

//-------------------------------------------------------------------------

public List<Actividad> ListarActFecha(DateTime f1, DateTime f2)

{

List<Actividad> actEntreFechas = new List<Actividad>();

if (f1 != null && f2 != null && f1 != f2)

{

if (f1 > f2)

{

DateTime aux = f1;

f1 = f2;

f2 = aux;

}

foreach (Actividad a in GetActividades())

{

if (a.FechaYHora > f1 && a.FechaYHora < f2)

{

actEntreFechas.Add(a);

}

}

}

return actEntreFechas;

}

//-------------------------------------------------------------------------

//Lista las actividades aptas todo público

//-------------------------------------------------------------------------

public List<Actividad> ListarEspectaculoTodoPublico()

{

List<Actividad> actividadesTodoPublico = new List<Actividad>();

foreach (Actividad a in GetActividades())

{

if (a.MinimaEdad == EdadMinima.p)

{

actividadesTodoPublico.Add(a);

}

}

return actividadesTodoPublico;

}

//-------------------------------------------------------------------------

//Verificamos que el aforo cumpla los requerimientos, y luego en

// cambiarAforo realizamos el cambio del mismo

//-------------------------------------------------------------------------

public bool VerificarAforo(int aforo)

{

bool ret = false;

if (!CambiarAforo(aforo))

{

while (aforo < 0 || aforo > 100)

{

Console.WriteLine("Ingrese un valor mayor a 0 y menor a 100");

aforo = Int32.Parse(Console.ReadLine());

}

ret = true;

}

return ret;

}

public bool CambiarAforo(int aforo)

{

bool ret = false;

if (aforo > 0 && aforo <= 100)

{

AforoMaximo = aforo;

ret = true;

}

return ret;

}

//-------------------------------------------------------------------------

//Idem Cambio de aforo

//-------------------------------------------------------------------------

public bool CambiarPrecioBase(double precio)

{

bool ret = false;

if (precio > 0 && precio != PrecioBase)

{

PrecioBase = precio;

ret = true;

}

return ret;

}

public bool VerificarBase(double precio)

{

bool ret = false;

if (!CambiarPrecioBase(precio))

{

while (precio < 0 && !ret)

{

Console.WriteLine("Ingrese un valor mayor a 0");

precio = Double.Parse(Console.ReadLine());

}

ret = true;

}

return ret;

}

//-------------------------------------------------------------------------

//Verifica que el valor ingresado sea un int

//-------------------------------------------------------------------------

public bool VerificacionInt(int valor)

{

bool bandera = false;

while (!bandera)

{

try

{

Console.WriteLine("Ingrese un valor numérico válido");

valor = Int32.Parse(Console.ReadLine());

if (valor > 0)

bandera = true;

}

catch { }

}

return bandera;

}

//-------------------------------------------------------------------------

//TomaDeFechas, TOMA NUEVAMENTE LOS DATOS DEL USUARIO DENTRO DEL CATCH.

//Recibe un int unicamente para saber de qué fecha se trata.

//Devuelve un DateTime al program (VALIDO).

//No va a salir de la funcion hasta que se pueda devolver un DateTime Valido si o si.

//-------------------------------------------------------------------------

public DateTime TomaDeFechas(int contador)

{

DateTime nuevaFecha = DateTime.Now;

bool bandera = false;

while (!bandera)

{

Console.WriteLine($"Ingrese fecha {contador}");

string fecha = Console.ReadLine();

if (VerficiarDateTime(fecha))

{

nuevaFecha = DateTime.Parse(fecha);

bandera = true;

}

else

{

Console.WriteLine("Ingrese un valor de fecha correcto, vamos usted puede");

}

}

return nuevaFecha;

}

//-------------------------------------------------------------------------

//VerificarDateTime VERIFICA DENTRO DE LA FUNCION TomaDeDatosDos

//Recibe un String, Desde TomaDeDatosDos.

//-------------------------------------------------------------------------

public bool VerficiarDateTime(string fecha)

{

DateTime valor;

bool bandera = false;

try

{

valor = DateTime.Parse(fecha);

bandera = true;

}

catch (Exception)

{

}

return bandera;

}

//---------------------------------------------------------------------------------------------------------

//------------------------------------------------USUARIO-------------------------------------------------

//---------------------------------------------------------------------------------------------------------

public Usuario GetUsuario(string nick, string pass)

{

Usuario encontrado = null;

nick = convertirAMayuscula(nick);

foreach (Usuario us in Usuarios)

{

if (us.Nick.Equals(nick) && us.Password.Equals(pass))

{

encontrado = us;

}

}

return encontrado;

}

public Usuario AltaUsuarioCliente(string nombre, string apellido, string email, DateTime fechaNacimiento, string password, string nick)

{

Usuario nuevo = null;

if (validarNombreApellido(nombre) == true && validarNombreApellido(apellido) == true && ExisteEmail(email) == false && ValidarFechaNacimiento(fechaNacimiento) == true && validarPass(password) == true && ExisteNick(nick) == false)

{

nick = convertirAMayuscula(nick);

nuevo = new UsuarioCliente(nombre, apellido, email, fechaNacimiento, password, nick);

Usuarios.Add(nuevo);

}

return nuevo;

}

// obtener usuario mediante un ID

public Usuario GetUsuarioId(int? id)

{

Usuario encontrado = null;

foreach (Usuario usu in Usuarios)

{

if (usu.idUsuario.Equals(id))

{

encontrado = usu;

}

}

return encontrado;

}

//------------------------

//VALIDACIONES DE REGISTRO

//------------------------

private bool validarNombreApellido(string texto)

{

bool valido = false;

bool largo = false;

if (texto != null)

{

if (texto.Count() >= 2)

{

largo = true;

}

}

if (largo == true)

{

valido = true;

}

return valido;

}

private bool validarPass(string pass)

{

bool valido = false;

bool mayuscula = false, numero = false, minuscula = false, largo = false;

if (pass != null)

{

for (int i = 0; i < pass.Count(); i++)

{

if (char.IsNumber(pass, i))

{

numero = true;

}

if (Encoding.ASCII.GetBytes(pass.ToString())[i] >= 65 & Encoding.ASCII.GetBytes(pass.ToString())[i] <= 90)

{

mayuscula = true;

}

if (Encoding.ASCII.GetBytes(pass.ToString())[i] >= 97 & Encoding.ASCII.GetBytes(pass.ToString())[i] <= 122)

{

minuscula = true;

}

if (pass.Count() >= 6)

{

largo = true;

}

}

if (mayuscula && numero && minuscula && largo)

{

valido = true;

}

}

return valido;

}

private bool ExisteNick(string nick)

{

bool valido = false;

nick = convertirAMayuscula(nick);

foreach (Usuario us in Usuarios)

{

if (us.Nick.Equals(nick))

{

valido = true;

}

}

return valido;

}

private bool ExisteEmail(string email)

{

bool valido = false;

foreach (Usuario us in Usuarios)

{

if (us.Email.Equals(email))

{

valido = true;

}

}

return valido;

}

private bool ValidarFechaNacimiento(DateTime fecha)

{

bool valido = false;

if (fecha < DateTime.Now)

{

valido = true;

}

return valido;

}

//Convierte un txt a mayuscula

private string convertirAMayuscula(string txt)

{

return txt.ToUpper();

}

//listar actividades con el mismo nombre sirve para las estadisticas de operador 10 11 2021

public List<Actividad> GetNombreActividadesLugar(string Nombre)

{

List<Actividad> ActividadesA = new List<Actividad>();

foreach (Actividad a in Actividades)

{

if (a.Lugar.Nombre.Equals(Nombre))

{

ActividadesA.Add(a);

}

}

return ActividadesA;

}

// lista actividades con un limite de fechas y un nombre de categoria sirve para estadisticas de operador 11 11 2021

public List<Actividad> GetActividadesEntreFechasYCategoria(DateTime f1, DateTime f2, string NombreCategoria)

{

List<Actividad> ActividadesA = new List<Actividad>();

if (f1 != null && f2 != null && f1 != f2)

{

if (f1 > f2)

{

DateTime aux = f1;

f1 = f2;

f2 = aux;

}

foreach (Actividad a in Actividades)

{

if (a.FechaYHora > f1 && a.FechaYHora < f2 && a.Categoria.Nombre.Equals(NombreCategoria))

{

ActividadesA.Add(a);

}

}

}

return ActividadesA;

}

//------------------------------------------------------------

//--------------------------COMPRAS---------------------------

//------------------------------------------------------------

//retorna las compras con mayor valor sirve para estadisticas de operador 11 11 2021

public List<Compra> GetComprasDeMayorValor()

{

List<Compra> Comprasdemayorvalor = new List<Compra>();

double Mayorvalor = 0;

foreach (Compra C in Compras)

{

if (C.PrecioTotal() > Mayorvalor)

{

Comprasdemayorvalor.Add(C);

Mayorvalor = C.PrecioTotal();

Comprasdemayorvalor.Clear();

}

if (C.PrecioTotal().Equals(Mayorvalor))

{

Comprasdemayorvalor.Add(C);

}

}

return Comprasdemayorvalor;

}

// ALTA DE COMPRA

public void AltaCompra(Actividad actividadComprada, int cantEntradas, Usuario usuario, DateTime fechaCompra)

{

Compra nueva = null;

double precioFinal = actividadComprada.PrecioActividad() \* cantEntradas;

if (cantEntradas > 5)

{

precioFinal -= precioFinal \* 0.05;

}

nueva = new Compra(actividadComprada, cantEntradas, usuario, fechaCompra, precioFinal);

Compras.Add(nueva);

}

//intento de dar like

public void darLike(int idActi)

{

Actividad activi = GetActividad(idActi);

activi.CantidadMG++;

}

//Devuelve una compra

public Compra GetCompra(int idCompra)

{

Compra buscada = null;

foreach (Compra item in Compras)

{

if (item.Id.Equals(idCompra))

{

buscada = item;

}

}

return buscada;

}

// Cambiar estado a cancelado de una compra

public void CambiarEstadoDeCompra(Compra comp)

{

Actividad activComp = comp.ActividadComprada;

TimeSpan dato = activComp.FechaYHora- DateTime.Now;

if (dato.TotalHours > 24)

{

comp.Estado = Compra.Estados.cancelada;

}

}

//retorna clientes ordenados alfabeticamente

public List<Usuario> GetClientes()

{

List<Usuario> Clientes = new List<Usuario>();

foreach(Usuario u in Usuarios)

{

if (u.TipoUsuario == "cliente")

{

Clientes.Add(u);

}

}

Clientes.Sort();

return Clientes;

}

public void Precarga()

{

Categoria cat1 = AltaCategoria("Cine", "Noche de cine");

Categoria cat2 = AltaCategoria("Teatro", "Stand up");

Categoria cat3 = AltaCategoria("Concierto", "The police");

Categoria cat4 = AltaCategoria("Feria gastronomica", "Varios food trucks");

Abierto a1 = AltaLugarAbierto("Estadio Centenario", 35000);

Abierto a2 = AltaLugarAbierto("Teatro de Verano", 2500);

Abierto a3 = AltaLugarAbierto("Rural del Prado", 2000);

Abierto a4 = AltaLugarAbierto("Plaza de Mayo", 300);

Abierto a5 = AltaLugarAbierto("Faro Punta Carretas", 500);

Cerrado c1 = AltaLugarCerrado("Antel Arena", 800, true, 20000);

Cerrado c2 = AltaLugarCerrado("Teatro Metro", 200, false, 2);

Cerrado c3 = AltaLugarCerrado("Teatro Solis", 700, true, 2);

Cerrado c4 = AltaLugarCerrado("Movie Center", 150, true, 2);

Cerrado c5 = AltaLugarCerrado("Montevideo shopping", 650, true, 2);

Actividad act1 = AltaActividad(cat3, DateTime.Parse("2024-09-21 16:00"), a2, EdadMinima.c16, 10000, "Espectaculo musical");

Actividad act2 = AltaActividad(cat2, DateTime.Parse("2023-08-21 20:00"), c1, EdadMinima.c18, 5800, "Espectaculo de stand up");

Actividad act3 = AltaActividad(cat4, DateTime.Parse("2022-07-16 12:00"), c3, EdadMinima.p, 50, "Evento de gastronomia y animales");

Actividad act4 = AltaActividad(cat4, DateTime.Parse("2022-07-10 10:00"), c5, EdadMinima.p, 150, "Evento de gastronomia");

Actividad act5 = AltaActividad(cat1, DateTime.Parse("2023-09-21 16:20"), c4, EdadMinima.c13, 503, "Rapido y furioso 32");

Actividad act6 = AltaActividad(cat1, DateTime.Parse("2022-10-20 17:00"), a5, EdadMinima.p, 1450, "Cine al aire libre");

Actividad act7 = AltaActividad(cat3, DateTime.Parse("2022-01-11 22:00"), a1, EdadMinima.c18, 324, "Espectaculo musical nro 23");

Actividad act8 = AltaActividad(cat3, DateTime.Parse("2021-12-05 17:00"), a4, EdadMinima.p, 1, "The police en vivo");

Actividad act9 = AltaActividad(cat2, DateTime.Parse("2024-09-21 16:00"), c2, EdadMinima.c16, 1, "Espectaculo de stand up");

Actividad act10 = AltaActividad(cat1, DateTime.Parse("2022-09-21 20:00"), c3, EdadMinima.p, 1, "Cine antiguo");

//Precarga usuarios

UsuarioOperador op1 = new UsuarioOperador("Andres", "Fernandez", "AFernaandez@gmail.com", DateTime.Parse("1994-09-21"), "Abcd1234", "OPERADOR1");

Usuarios.Add(op1);

UsuarioOperador op2 = new UsuarioOperador("Martina", "Acevedo", "MartinaAC@gmail.com", DateTime.Parse("1990-05-11"), "Abcd1234", "OPERADOR2");

Usuarios.Add(op2);

Usuario u1 = AltaUsuarioCliente("Federico", "Perez", "FP@gmail.com", DateTime.Parse("2000-08-29"), "Abcd1234", "USUARIO1");

Usuario u2 = AltaUsuarioCliente("Ana", "Rapal", "rapalA@gmail.com", DateTime.Parse("2005-03-01"), "Abcd1234", "USUARIO2");

//COMPRAS

DateTime ahora = DateTime.Now;

AltaCompra(act10, 3, u1, ahora);

AltaCompra(act6, 2, u1, ahora);

AltaCompra(act8, 5, u1, ahora);

AltaCompra(act3, 1, u2, ahora);

AltaCompra(act5, 4, u2, ahora);

AltaCompra(act1, 4, u2, ahora);

AltaCompra(act2, 3, u1, ahora);

AltaCompra(act3, 2, u1, ahora);

}

}

# Testing

## Usuario no registrado

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Escenario de test | Pasos | Datos utilizados | Resultado esperado | Resultado obtenido. | Estado  Falla = F  Aprobado = A |
| Ventana de inicio con usuario sin registrar. | Ingreso a la aplicación,  Ventana de inicio. |  | Visualizo  Lista actividades  ,Log in,  Registro. | Visualizo  Lista actividades  ,Log in,  Registro. | A |
| Sección  Lista de actividades,  Con usuario sin identificar | Ingreso a lista de actividades. |  | Visualizo lista de actividades con,  Nombre  Categoria  Fecha y hora  Lugar  Edad minima  Precio | Visualizo lista de actividades con,  Nombre  Categoria  Fecha y hora  Lugar  Edad minima  Precio | A |
| Sección log in  Con usuario sin identificar | Ingreso a log in. |  | Visualizo campo para rellenar nombre de usuario, password, boton de log in y link a registro. | Visualizo campo para rellenar nombre de usuario, password, boton de log in y link a registro. | A |
| Sección Registro con usuario sin identificar. | Ingreso a registro. |  | Visualizo, campo para rellenar, nombre,  Apellido,  Email, fecha de nacimiento, contraseña,  Nombre de usuario y boton de registro. | Visualizo, campo para rellenar, nombre,  Apellido,  Email, fecha de nacimiento, contraseña,  Nombre de usuario y boton de registro. | A |
| Sección Registro con usuario sin identificar. | Ingreso a registro. | Completo campos con:  Nombre: pepe  Apellido: Rodriguez  Email: [pepe@gmail.com](mailto:pepe@gmail.com)  Pass: Abcd1234  Nick: pepe  Oprimo registrar. | Se vacian los campos y se muestra registro exitoso. | Se vacian los campos y se muestra registro exitoso. | A |
| Sección Registro con usuario sin identificar. | Ingreso a registro. | Oprimo registrar sin completar los campos. | Se muestra Verifique los datos. | Se muestra Verifique los datos. | A |
| Seccion Login, con usuario sin identificar | Ingreso a login | Completo los campos con  Nombre de usuario: Usuario1  Pass: Abcd1234  Y oprimo boton log in | Se e ingresa correctamente a la sección de usuario registrado | Se e ingresa correctamente a la sección de usuario registrado | A |
| Seccion Login, con usuario sin identificar | Ingreso a login | Completo los campos con  Nombre de usuario: operador1  Pass: Abcd1234  Y oprimo boton log in | Se e ingresa correctamente a la sección de operador | Se e ingresa correctamente a la sección de operador | A |

## Usuario Registrado

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Escenario de test | Pasos | Datos utilizados | Resultado deseado | Resultado obtenido | Estado  Falla = F  Aprobado = A |
| Inicio de usuario registrado | Visualizo elementos | Login  Usuario1  Abcd1234 | Visualizo lista de actividades con,  Nombre  Categoria  Fecha y hora  Lugar  Edad minima  Precio, boton de me gusta, cantidad de me gusta,  Y link a comprar.  Visualizo nav con Lista de actividades,  Compras,  Cerrar sesion. | Visualizo lista de actividades con,  Nombre  Categoria  Fecha y hora  Lugar  Edad minima  Precio, boton de me gusta, cantidad de me gusta,  Y link a comprar.  Visualizo nav con Lista de actividades,  Compras,  Cerrar sesion. | A |
| Inicio de usuario registrado | Boton cerrar sesion. |  | Oprimo boton cerrar sesion, se pasa a la ventana de confirmacion para cerrar la sesion. Se confirma y vuelve a inicio de usuario no identificado. | Oprimo boton cerrar sesion, se pasa a la ventana de confirmacion para cerrar la sesion. Se confirma y vuelve a inicio de usuario no identificado. | A |
| En usuario registrado, compras. | Ingreso a compras desde usuario registrado. |  | Se listan las compras y su correspondiente boton para cancelar la compra, si se toca el boton la compra queda cancelada, siempre y cuando sea antes de 24 horas del evento. | Se listan las compras y su correspondiente boton para cancelar la compra, si se toca el boton la compra queda cancelada, siempre y cuando sea antes de 24 horas del evento. | A |

## Usuario Operador

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Escenario de test | Pasos | Datos utilizados | Resultado deseado | Resultado obtenido | Estado  Falla = F  Aprobado = A |
| Inicio usuario operador | Visualizo inicio de usuario operador | Login  Operador1  Abcd1234 | Visualizo nav con,  Compras, clientes, estadísticas, cerrar sesion. | Visualizo nav con,  Compras, clientes, estadísticas, cerrar sesion. | A |
| Inicio de usuario operador | Botón cerrar sesión. |  | Oprimo botón cerrar sesión, se pasa a la ventana de confirmación para cerrar la sesión. Se confirma y vuelve a inicio de usuario no identificado. | Oprimo botón cerrar sesión, se pasa a la ventana de confirmación para cerrar la sesión. Se confirma y vuelve a inicio de usuario no identificado. | A |
| Compras en usuario operador | Rellenar datos de fecha  Y oprimir botón aceptar. |  | Se listan todas las compras que se efectuaron entre 2 fechas | Se listan todas las compras que se efectuaron entre 2 fechas | A |
| Compras en usuario operador | Oprimo botón aceptar sin rellenar campos. |  | Se muestra verifique las fechas | Se muestra verifique las fechas | A |
| Clientes en usuario operador. | Ingreso a clientes desde inicio usuario operador. |  | Se listan los usuarios registrados que existen | Se listan los usuarios registrados que existen | A |
| Estadísticas usuario operador. | Ingreso a estadísticas desde inicio usuario operador. |  | Se listan 3 opciones,  Compras de mayor valor, Actividades por fecha y categoría, Actividades por lugar | Se listan 3 opciones,  Compras de mayor valor, Actividades por fecha y categoría, Actividades por lugar | A |
| Compra de mayor valor usuario operador | Ingreso a compras de mayor valor mediante estadísticas en usuario operador. |  | Se lista la compra o las compras de mayor valor | Se lista la compra o las compras de mayor valor | A |
| Actividades de fecha y categoría  Usuario operador | Ingreso actividades de fecha y categoría mediante estadísticas en usuario operador. |  | Se visualiza, campos para ingresar fecha, campo para ingresar categoría, botón aceptar, y se listan las categorías disponibles. | Se visualiza, campos para ingresar fecha, campo para ingresar categoría, botón aceptar, y se listan las categorías disponibles. | A |
| Actividades de fecha y categoría  Usuario operador | Relleno campos de Actividades de fecha y categoría y oprimo en aceptar. | Ingreso 2 fechas con bastante diferencia entre ellas,  Categoría concierto | Se listan todos los eventos de esa categoría. | Se listan todos los eventos de esa categoría. | A |
| Actividades de fecha y categoría  Usuario operador | Oprimo aceptar con los campos vacíos. |  | Se muestra verifique los datos ingresados | Se muestra verifique los datos ingresados | A |
| Actividades por lugar usuario operador | Ingreso actividades por lugar mediante estadísticas de usuario operador. |  | Se visualiza input para ingresar nombre de lugar, y se listan los lugares disponibles | Se visualiza input para ingresar nombre de lugar, y se listan los lugares disponibles | A |
| Actividades por lugar usuario operador | Se ingresa un lugar y se oprime aceptar. | Teatro solis | Se lista las actividades en el teatro solis | Se lista las actividades en el teatro solis | A |
| Actividades por lugar usuario operador | Oprimo aceptar sin rellenar campo. |  | Se muestra verifique los datos | Se muestra verifique los datos | A |