

Brewing Creators

Creando cerveceros

Trabajo Práctico 4

UTNfra - Laboratorio

Contenido

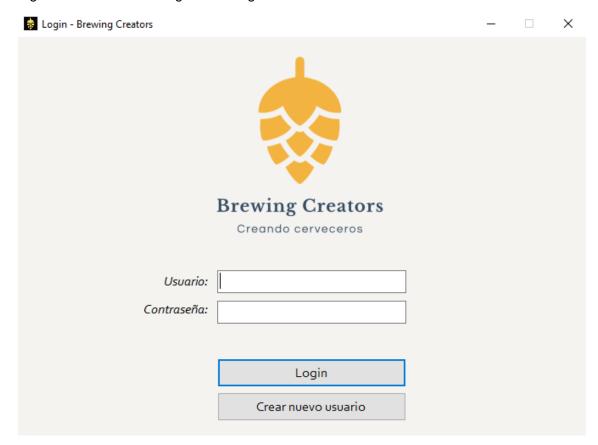
Deta	alle del proyecto	3
	Formulario de login	3
	Formulario de alta de usuario	4
	Formulario menú principal	5
	Formulario de carga de productos	6
	Formulario lista de productos	7
	Formulario de carga de venta	7
	Formulario lista de ventas	8
	Formulario alta de nuevo cliente	9
In	nplementación de temas de cursada	10
	Excepciones (Clase 15)	10
	Test unitarios (Clase 16)	11
	Tipos genéricos, interfaces y serialización (Clase 17, 18 y 19)	12
	SQL y bases de datos (Clase 21 y 22)	13
	Hilos (Clase 23)	14
	Eventos (Clase 24)	15
	Métodos de extensión (Clase 25)	16

Detalle del proyecto

Brewing Creators es un sistema de ventas de insumos para la fabricación de cervezas estilo artesanal. A continuación se detallarán todos los formularios del proyecto y sus distintos usos.

Formulario de login

Al iniciar el aplicativo se iniciará un formulario de login que permitirá al usuario poder loguearse al sistema o registrarse según se desee.



Formulario de alta de usuario

Este formulario se iniciará al crear un nuevo usuario en el formulario de Login. Permitirá que quien use la aplicación cree un nuevo usuario para poder loguearse y hacer uso de las distintas herramientas. Se validará que el usuario y la contraseña sean válidos y que no exista otro usuario con la misma cuenta.

Nuevo usuario							
Brewing Creators							
Creando cerveceros							
Nombre:							
Apellido:							
Usuario:							
Contraseña:							
Cancelar			Acepta	r			

Formulario menú principal

Una vez logueados pasará a ser nuestro formulario principal. De aquí podremos acceder a cualquiera de nuestros otros 5 formularios disponibles a través de unos botones que veremos en nuestra ventana.



Formulario de carga de productos

Nos permitirá realizar el alta de nuevos productos.



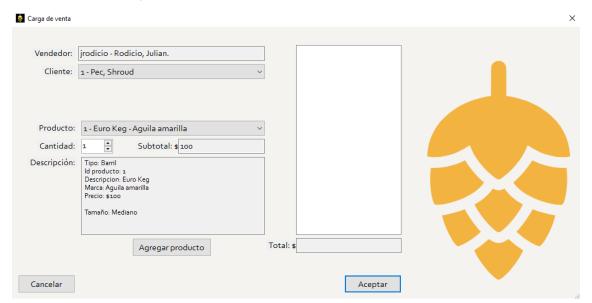
Formulario lista de productos

Se listaran todos los productos disponibles. Podrán ser exportados en todos en un mismo archivo en formato texto, o bien, solo el producto seleccionado en formato XML.



Formulario de carga de venta

Se utilizará para cargar una venta nueva.



Formulario lista de ventas

Se listarán todas las ventas registradas en una grilla simple.



Formulario alta de nuevo cliente

Se utilizará para dar de alta un nuevo cliente.



Implementación de temas de cursada

Excepciones (Clase 15)

Se generó un proyecto "Excepciones" donde se crearon algunas propias a utilizarse en la solución. Las mismas se lanzan en gran parte de la solución, dentro del proyecto Entidades tenemos por ejemplo:

- Clase: Usuario – Método: ValidarNombreUsuario

```
private static string ValidarNombreUsuario(string username)
{
    if (username.Length > 0 && username.Length <= 32)
    {
        foreach (char c in username)
        {
            if (char.IsWhiteSpace(c) || char.IsSymbol(c))
              {
                 throw new UsuarioInvalidoException("El nombre de usuario posee un caracter inválido.");
        }
    }
    else
    {
        throw new UsuarioInvalidoException("El usuario debe tener entre 1 y 32 caracteres.");
    }
    return username;
}</pre>
```

- Clase: BrewingCreator – Propiedad: UsuarioLogueado.

```
public Usuario UsuarioLogueado
{
    get
    {
        return this.usuarioLogueado;
    }
    set
    {
        if (value is null || this.ComprobarCredenciales(value))
        {
            this.usuarioLogueado = value;
        }
        else
        {
            throw new UsuarioInvalidoException("El usuario o la clave son incorrectos.");
        }
}
```

Test unitarios (Clase 16)

Se generó un proyecto "TestUnitarios" con 3 clases de pruebas:

- 1. XMLTest: Se testea la serialización y deserealización de productos.
- 2. TXTTest: Se testea el guardado de un producto.
- AltaUsuarioTest: Se testea la creación de un usuario incorrecto y otro correcto.

Tipos genéricos, interfaces y serialización (Clase 17, 18 y 19)

Se utilizan clases genéricas en el proyecto "Archivos" para detallar en las interfaces IArchivosGuardar<T> e IArchivosLeer<T> que el tipo puede ser cualquier clase Producto.

Las clases Txt y Xml hacen uso de las interfaces mencionadas y se puede ver implementado en el formulario ListaProductosForm, en el Click de los botones de guardado.

La serialización podrá verse en las mismas clases ya mencionadas e implementadas tanto en el formulario ListaProductosForm para guardar distintos productos en formato XML. También hay algunas muestras donde se comprueba una correcta serialización y deserealización de un producto en los test unitarios.

```
public class Xml<T> : IArchivosGuardar<Producto>, IArchivosLeer<Producto>
{
    Propiedades
    Métodos
}
```

```
public void Guardar(string nombreArchivo, Producto producto)
{
    try
    {
        using (XmlTextWriter writer = new XmlTextWriter($"{this.GetDirectoryPath}{nombreArchivo}", Encoding
        {
            writer.Formatting = Formatting.Indented;
            XmlSerializer serializer = new XmlSerializer(typeof(T));
            serializer.Serialize(writer, producto);
        }
        catch (Exception ex)
        {
            throw new ErrorArchivoException("Error al guardar archivo.", ex);
        }
}
```

SQL y bases de datos (Clase 21 y 22)

Dentro del proyecto Entidades, se creó una clase estática SQL para mantener toda la parte de SQL y base de datos separada del resto de la lógica. Dentro de esta clase estarán distintos métodos que leerán información o insertaran registros en la base de datos, por ejemplo:

 LeerProductos(): Método estático que retorna una lista de productos leída de la base de datos.

```
public static List<Producto> LeerProductos()
   List<Producto> listaProductos = new List<Producto>();
   int idProducto;
   string descripcion;
   string marca;
   float precio;
   string tamaño;
   string tipo;
   try
       string textCommand = @"SELECT id producto,
                                       descripcion,
                                       precio,
                                       tamaño
                               FROM dbo.Productos ORDER BY id_producto";
       SqlCommand sqlCommand = new SqlCommand(textCommand, SQL.GetConnection);
       SQL.GetConnection.Open();
       SqlDataReader reader = sqlCommand.ExecuteReader();
       while (reader.Read())
           idProducto = reader.GetInt32(0);
           tipo = reader.GetString(1);
           descripcion = reader.GetString(2);
           marca = reader.GetString(3);
           precio = (float)reader.GetDecimal(4);
```

 InsertUsuario(Usuario nuevoUsuario): Método estático que inserta un registro en la tabla dbo.Usuarios de la base de datos.

Hilos (Clase 23)

Se utilizan en varias oportunidades. Algunos hilos podemos encontrarlos en métodos o propiedades de la clase BrewingCreators (Proyecto Entidades) como por ejemplo:

- Propiedad AppendVenta: Inserta una nueva venta en la lista de ventas del singleton BrewingCreators e inicia en un nuevo hilo el cambio de estado de "En depósito" a "Retirado".

```
public Venta AppendVenta
{
    set
    {
        SQL.InsertVenta(value);
        this.listaVentas.Add(value);

        Thread thread = new Thread(new ParameterizedThreadStart(this.RetiroDeVenta));
        thread.Start(value);
        this.Threads.Add(thread);
    }
}
```

Campo List<Thread> threads: Listará todo thread que se inicie en el sistema.
 Al finalizar la aplicación se comprobaran que todos los threads hayan concluido. Esta verificación se hace en el método
 BrewingCreatorsPrincipalForm_FormClosing del formulario
 BrewingCreatorsPrincipalForm, ejecutado al momento de finalizar la aplicación.

```
private void BrewingCreatorsPrincipalForm_FormClosing(object sender, FormClosingEventArgs e)
{
    BrewingCreator bc = BrewingCreator.GetBrewingCreatorsSystem();
    foreach (Thread thread in bc.Threads)
    {
        while (thread.IsAlive)
        {
            MessageBox.Show("Se están finalizando algunos procesos internos, aguarde unos sellon }
    }
    Application.Exit();
}
```

Eventos (Clase 24)

Podremos ver varios eventos personalizados con sus respectivos delegados:

- Proyecto Entidades:
 - Clase SQL:
 - Evento: InformarVentaActualizada: Se invoca en el método ActualizarEstadoVenta(Venta venta) de la misma clase con el fin de informar a quien esté suscrito a dicho evento que se acaba de actualizar una venta.

```
#region Campos y eventos
public static event VentaActualizadaDB InformarVentaActualizada;
```

- Proyecto BrewingCreatorsForms:
 - o Clase ListaVentasForm:
 - Método ListaVentas_Load(object sender, EventArgs e): En este método suscribimos al método CargarListaVentas al evento InformarVentaActualizada con el fin de poder actualizar automáticamente la grilla en pantalla cada vez que una venta cambie de estado.

```
private void ListaVentas_Load(object sender, EventArgs e)
{
    this.CargarListaVentas();
    SQL.InformarVentaActualizada += CargarListaVentas;
}
```

Método CargarListaVentas(): Se agrega un Callback para poder actualizar objetos que hayan sido declarados en un hilo distinto al principal. Esto se hace verificando la propiedad InvokeRequired del elemento grdListaVentas.

```
public void CargarListaVentas()
{
    DataTable dataTable = SQL.LeerVentas();
    if (this.grdListaVentas.InvokeRequired)
    {
        Callback callback = new Callback(CargarListaVentas);
        this.Invoke(callback);
    }
    else
    {
        this.grdListaVentas.DataSource = dataTable;
    }
}
```

Métodos de extensión (Clase 25)

Se declara una nueva clase (ExtensionDeMetodos) y la misma posee un método de extensión que permite a un objeto de la clase int convertirse en un número random entre 5000 y 30000. El método se llama Randomizer. La misma se implementa en la función RetiroDeVenta(object venta) de la clase BrewingCreator

```
public static void Randomizer(this int randomInt)
{
    Random rnd = new Random();
    randomInt = rnd.Next(5000, 30000);
}
```

```
public void RetiroDeVenta(object venta)
{
    Venta miVenta = (Venta)venta;
    int tiempo = 0;
    tiempo.Randomizer();

    Thread.Sleep(tiempo);

    miVenta.EstadoVenta = Venta.EVentaEstado.Retirado;

    if (!object.ReferenceEquals(this.InformaEstadoVenta, null))
    {
        this.InformaEstadoVenta.Invoke(miVenta);
    }
}
```