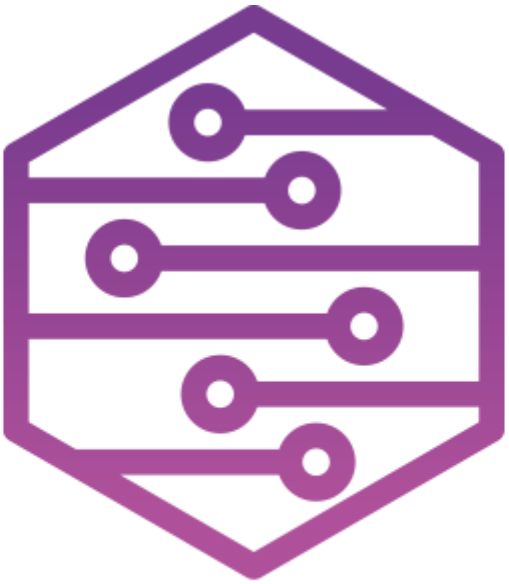
Trabajo Práctico Nº4



CONNECTED

2ºD Turno Noche
Ezequiel Matías Unía

ÍNDICE

INTRODUCCIÓN - DESARROLLO DEL MANUAL DE USUARIO

El CONNECTED es un sistema de gestión de la fábrica China “CONNECTED ATTENDANCE S.A.” la cual se dedica a la fabricación de equipos de control horario y control de acceso. El operador que producirá los dispositivos tendrá a disposición órdenes de trabajo (Ordenes de trabajo Interna) donde deberá seleccionar el pedido/oti para comenzar a ensamblar los equipos mencionados en dicha orden.Luego tendrá que ensamblar los dispositivos correspondientes hasta completar el pedido elegido, una vez que se encuentren terminados y listos para enviar a depósito, debe confirmar su orden de trabajo. Todos los dispositivos pueden ser auditados en el apartado Stock > Devices.

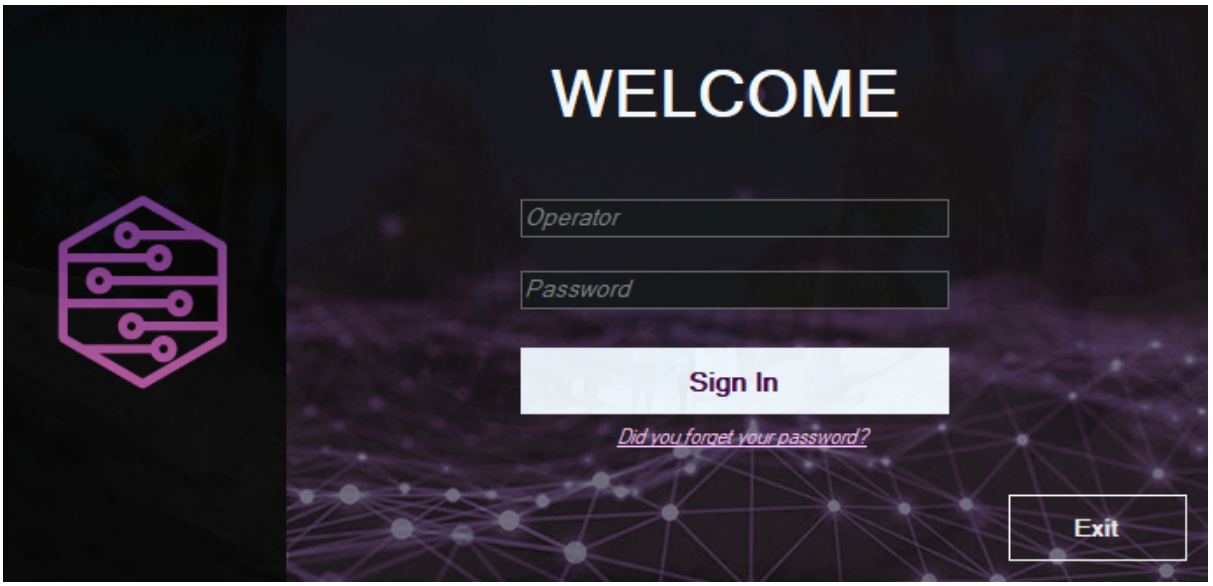
LOGIN

Cada operador de la fábrica deberá loguearse con su ID de usuario y su contraseña antes de comenzar el proceso de producción.

Los operadores registrados hasta el momento son:

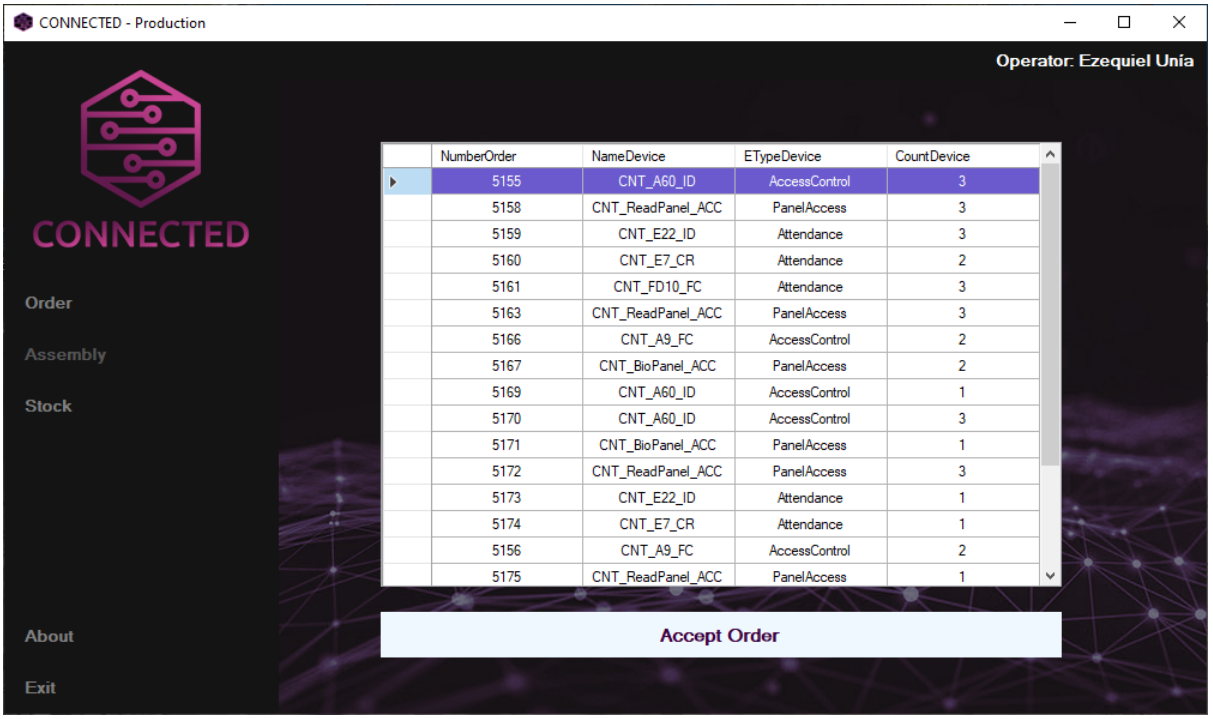
Nombre	Apellido	ID Usuario	Contraseña
Federico	Dávila	1001	8090
Mauricio	Cerizza	1002	5020
Ezequiel	Oggioni	1003	3535
Esteban	Prietto	1004	1331
Lautaro	Galarza	1005	3030
Lucas	Rodríguez	1006	2020
Ezequiel	Unía	1007	9999

En caso de olvidarse su ID de usuario y contraseña pueden consultar en el apartado *"Did you forget your password?"* en donde se le abrirá una nueva ventana con toda la información de los usuarios.



ORDER

En el apartado de órdenes de trabajo, el operador tiene que seleccionar un pedido de la lista que figura en pantalla y deberá aceptar la orden. Cada OT cuenta con su número de orden, el nombre del dispositivo a crear, el tipo del equipo y la cantidad que se deben fabricar.

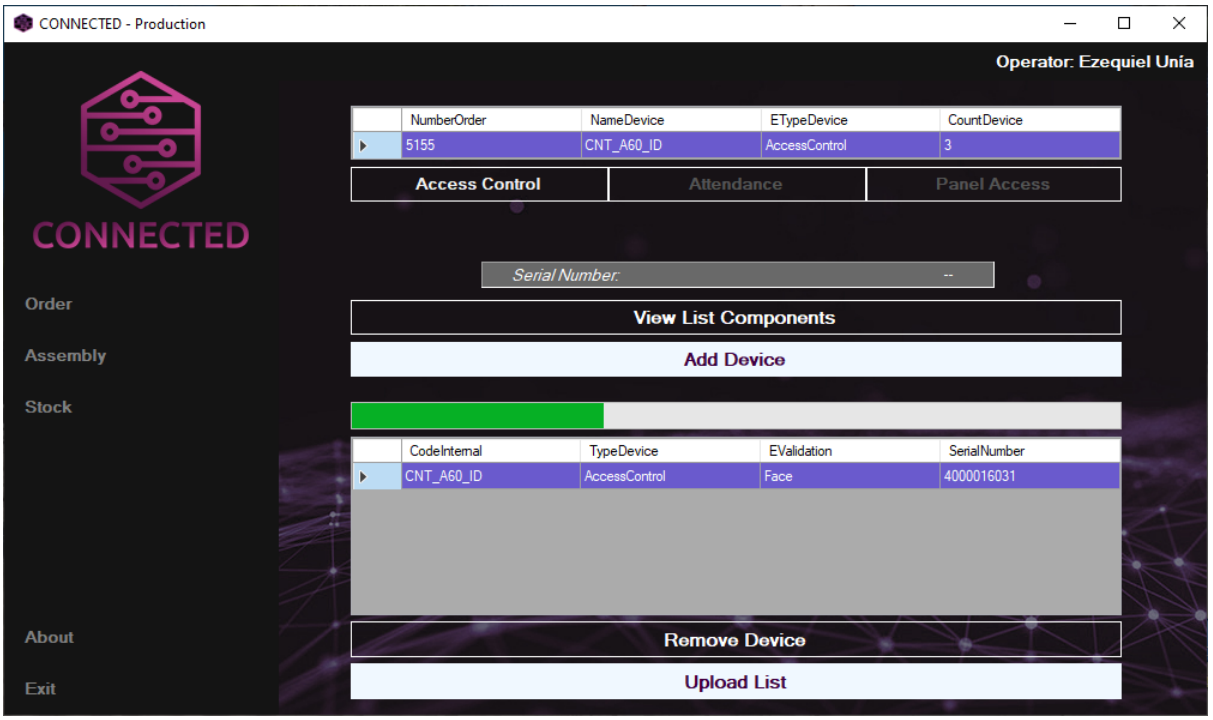


Al darle click al botón *“Accept Order”,* esta acción habilitará el apartado de ensamblado y lo redireccionará automáticamente.

ASSEMBLY

En el apartado de ensamblado es donde el operador deberá ir fabricando los dispositivos que sean necesarios hasta completar la orden de trabajo, la misma se muestra en el apartado superior para facilitar que el pedido se produzca más eficazmente.

El operador debe seleccionar el modelo y su tipo de verificación que se quiera fabricar, esto generará su número de serie correlativo a su tipo.



(Se bloqueará automáticamente los modelos que no apliquen para la orden seleccionada)

El botón *"View List Component"* nos abrirá una ventana con todos los componentes que va a utilizar el dispositivo a crear. (No es obligatorio utilizarlo)

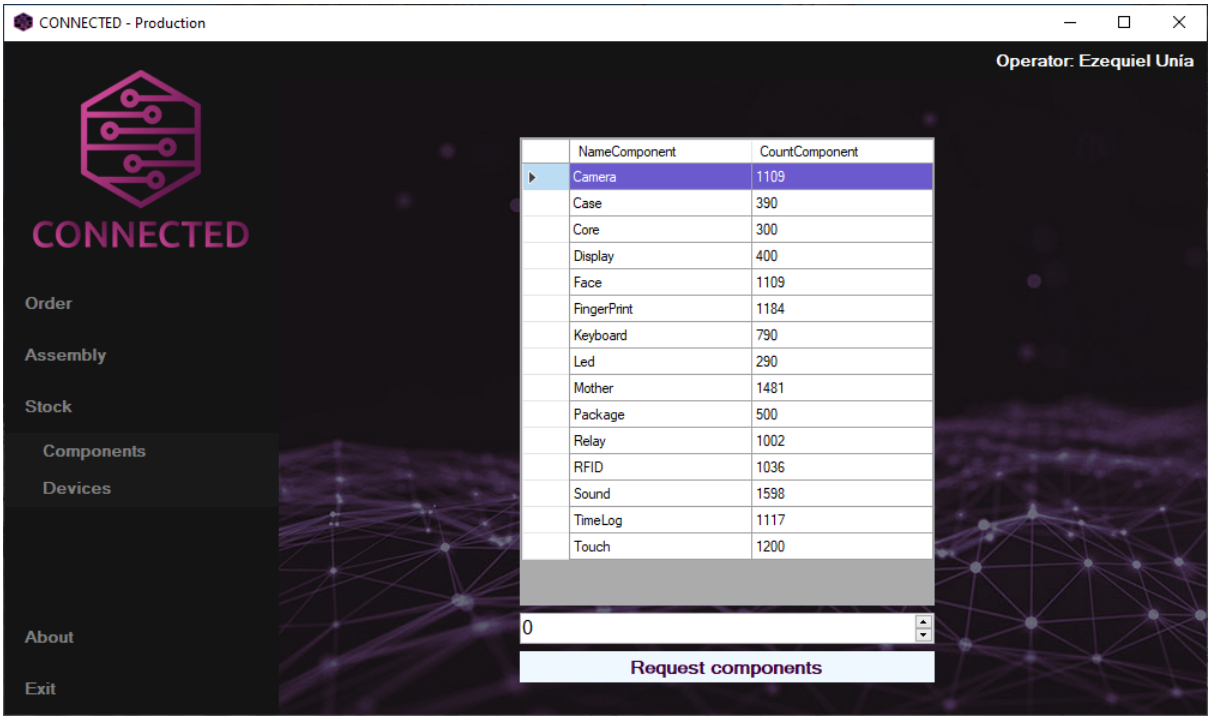
El botón *"Add Device"* nos va a fabricar el dispositivo seleccionado y lo va a almacenar en la mesa de trabajo. (Grilla inferior)

El botón *"Remove Device"* destruye el dispositivo seleccionado de la mesa de trabajo.

Una vez finalizada la orden de trabajo, respetando que la cantidad de dispositivos creados corresponda a la orden, se debe utilizar el botón *"Upload List"* para enviar todos los dispositivos fabricados al stock del depósito. (Tener en cuenta que el programa permitirá crear más dispositivos de lo que pide la orden pero avisará que está pasando, lo mismo cuando se quiera cerrar la orden de trabajo si el pedido se encuentra incompleto y sobrepasado

STOCK - COMPONENTS

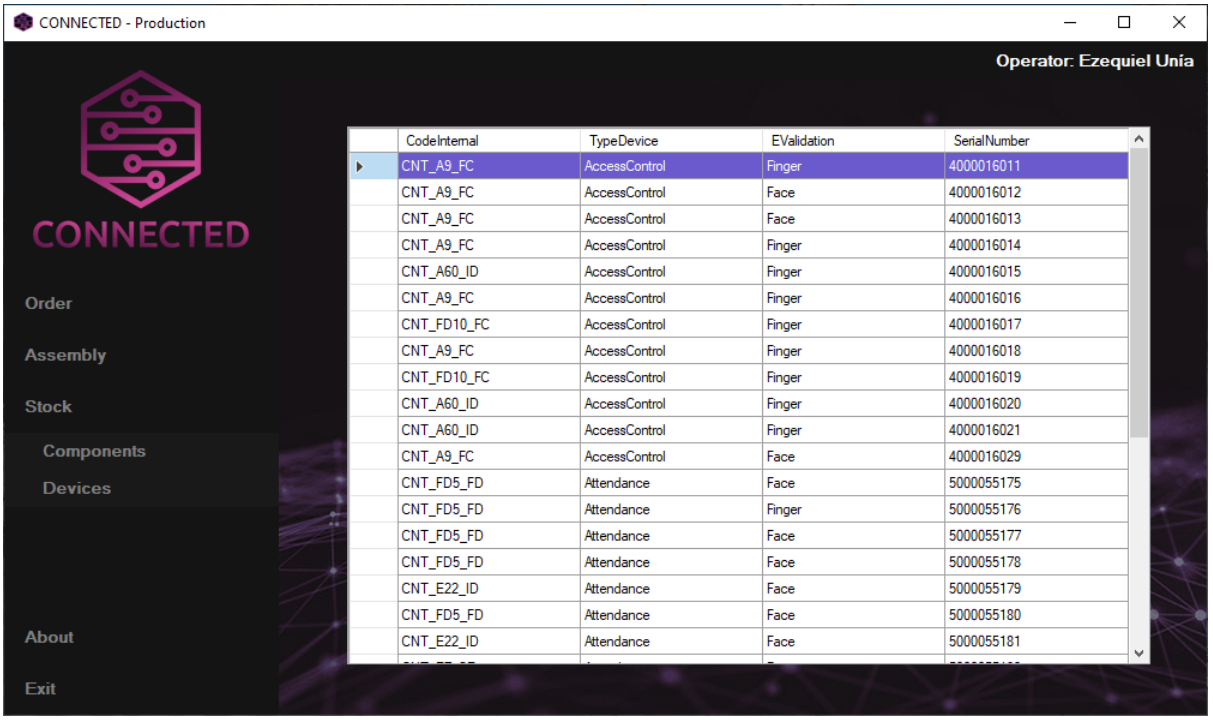
En el apartado “Stock - Components” obtendremos la información de la cantidad de componentes que se encuentren en stock.



En caso de no tener más stock, se debe colocar la cantidad a solicitar y presionar el botón *“Request Components”*. La cantidad máxima para solicitar por vez es de 100 unidades.

STOCK - DEVICES

En el apartado *“Stock - Devices”* solamente se podrán ver los dispositivos creados con su número de serie.



ABOUT

En el apartado about se puede visualizar la información del alumno que desarrolló el programa “CONNECTED”

EXIT - SIGN OFF

El apartado “Sign Off” desconectará al operador logueado.

EXIT - EXIT

El apartado “Exit” desconectará al operador y cerrará el programa.

EXCEPTIONS

Las excepciones fueron utilizadas en varios apartados pero las más importantes están aplicadas en biblioteca de clases “Files - Xml”

La excepción es lanzada:

```
public bool Save(string file, T data)
{
    try
    {
        if (!Directory.Exists(folder))
        {
            Directory.CreateDirectory(folder);
        }
        file = folder + file;
        using (XmlTextWriter writer = new XmlTextWriter(file, Encoding.UTF8))
        {
            XmlSerializer serializer = new XmlSerializer(typeof(T));
            serializer.Serialize(writer, data);
            return true;
        }
    }
    catch (Exception ex)
    {
        throw new Exception($"Failed to save file: {file}",ex);
    }
}
```

La excepción es capturada, lanzada y luego será capturada en el formulario Assembly:

```
public static bool SaveDevices()
{
    try
    {
        if (new Xml<List<Device>>().Save(@"\DevicesStock.xml", devicesStock))
        {
            return true;
        }
        return false;
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

EXCEPTIONS

Todas las excepciones fueron burbujeadas hasta los formularios donde se capturan y se muestra un *“MessageBox”* o se imprime por pantalla un *“Label”.*

```
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "NO SAVE FILE", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

TEST UNITARIOS

Los test unitarios los podemos encontrar en el proyecto *“TestDevices”.* En uno de los test se agregan 3 nuevos dispositivos a una lista previa, la cual corresponde a la “mesa de trabajo” y se llama al evento que sube esa lista al stock.

```
[TestMethod]
0 referencias
public void Test_Devices_Update_Stock()
{
    //Arrange
    AccessControl accessControl = new AccessControl(ECode.CNT_A60_ID, EType.AccessControl, EValidation.Card);
    Attendance attendance = new Attendance(ECode.CNT_A60_ID, EType.Attendance, EValidation.Card);
    AccessPanel accessPanel = new AccessPanel(ECode.CNT_A60_ID, EType.PanelAccess, 2, EValidation.Card);

    //Act
    CoreSystem.PreviewDevices.Add(accessControl);
    CoreSystem.PreviewDevices.Add(attendance);
    CoreSystem.PreviewDevices.Add(accessPanel);
    Stock.UpdateDevicesStock();

    //Assert
    Assert.IsNotNull(Stock.DevicesStock);
    Assert.IsTrue(Stock.DevicesStock.Contains(accessPanel));
    Assert.IsTrue(Stock.DevicesStock.Contains(attendance));
    Assert.IsTrue(Stock.DevicesStock.Contains(accessControl));
}
```

TIPOS GENERICOS

Los tipos genéricos fueron aplicados en el proyecto *“Files - Xml”*. que se utiliza en sus métodos de guardar y leer.

```
public class Xml<T> : IFiles<T>
```

```
public bool Read(string file, out T data)
{
    try
    {
        if (!Directory.Exists(folder))
        {
            Directory.CreateDirectory(folder);
        }
        file = folder + file;
        using (XmlTextReader reader = new XmlTextReader(file))
        {
            XmlSerializer serializer = new XmlSerializer(typeof(T));
            data = (T)serializer.Deserialize(reader);
            return true;
        }
    }
    catch (Exception ex)
    {
        throw new Exception($"Failed to read file: {file}", ex);
    }
}
```

INTERFACES

Las interfaces se pueden encontrar en el proyecto “*Files - IFiles”.*

```
2 referencias
public interface IFiles<T>
{
    /// <summary>
    /// Saves the data it receives as a parameter in a file
    /// </summary>
    /// <param name="file">File path </param>
    /// <param name="data">Data to save</param>
    /// <returns>True or False if the file was saved well</returns>
    13 referencias
    bool Save(string file, T data);

    /// <summary>
    /// Read saved data from a file
    /// </summary>
    /// <param name="file">File path</param>
    /// <param name="data">Where it records read data</param>
    /// <returns>True or False if the file was read well</returns>
    7 referencias
    bool Read(string file, out T data);
}
```

Siendo implementada en las clases de *“Text” y “Xml”*.

ARCHIVOS

Los archivos fueron utilizados para generar logs, tanto de errores como de reporte. Se encuentran en la ruta del proyecto:

\\tp_laboratorio_2\Trabajo Práctico Nº3\FrmCore\bin\Debug\LogsTxt

Así mismo, dentro del proyecto *"Files - Text"* se aprovecharon las excepciones para generar cada uno de los logs o algunos métodos de información.

```
public bool Save(string file, string data)
{
    try
    {
        if (!Directory.Exists(folder))
        {
            Directory.CreateDirectory(folder);
        }
        file = folder + file;
        using (StreamWriter sw = new StreamWriter(file, true))
        {
            sw.Write(data);
            return true;
        }
    }
    catch (Exception)
    {
        throw new Exception($"Failed to save file: {file}");
    }
}
```

```
public static bool SaveErrorLogComponents(string data)
{
    try
    {
        return new Text().Save(@"\LogErrorListComponents.txt", data + " " + DateTime.Now + "\n");
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

SERIALIZACIÓN

La serialización es parte excluyente para que el sistema funcione correctamente, está utilizada como base de datos, que de todas formas cuenta con excepciones capturadas que en caso de faltar alguno de sus archivos el programa no termine forzosamente. Así mismo, cada archivo generado se aloja en la carpeta del proyecto:

\\tp_laboratorio_2\Trabajo Práctico Nº3\FrmCore\bin\Debug\SerializationXml

Se puede encontrar en el proyecto *"Files - Xml"*.

```
public bool Read(string file, out T data)
{
    try
    {
        if (!Directory.Exists(folder))
        {
            Directory.CreateDirectory(folder);
        }
        file = folder + file;
        using (XmlTextReader reader = new XmlTextReader(file))
        {
            XmlSerializer serializer = new XmlSerializer(typeof(T));
            data = (T)serializer.Deserialize(reader);
            return true;
        }
    }
    catch (Exception ex)
    {
        throw new Exception($"Failed to read file: {file}", ex);
    }
}
```

SERIALIZACIÓN

En este caso, al igual que en archivos, se aprovechan las excepciones para sacar y burbujear algún error.

```
public static void ReadInternalOrder()
{
    try
    {
        List<InternalOrder> aux = new List<InternalOrder>();
        if (new Xml<List<InternalOrder>>().Read(@"\InternalOrders.xml", out aux))
        {
            CoreSystem.InternalOrders = aux;
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

Tomando el caso de la serialización de los números de serie, dentro de la clase “SerialsNumber” encontramos los métodos que nos permiten guardar y cargar la información:

```
public static bool SaveSerialsNumbers()
{
    try
    {
        SerializeSN aux = new SerializeSN();
        aux.AccessControl = AccessControl;
        aux.Attendance = Attendance;
        aux.PanelAccess = PanelAccess;
        if (new Xml<SerializeSN>().Save(@"\SerialsNumbers.xml", aux))
        {
            return true;
        }
        return false;
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

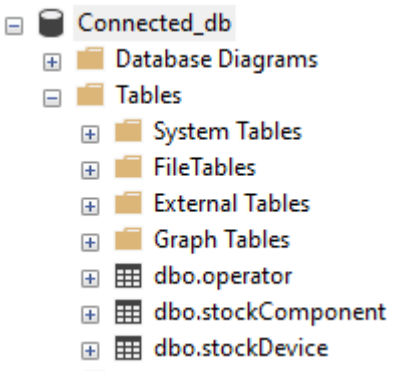
SERIALIZACIÓN

```
public static void ReadSerialsNumbers()
{
    try
    {
        SerializeSN aux = new SerializeSN();
        if (new Xml<SerializeSN>().Read("\\SerialsNumbers.xml", out aux))
        {
            AccessControl = aux.AccessControl;
            Attendance = aux.Attendance;
            PanelAccess = aux.PanelAccess;
        }
    }
    catch (Exception ex)
    {
        throw new Exception("The file with the serial numbers could not be found.", ex);
    }
}
```

Éstos métodos son utilizados para levantar la información del último número de serie utilizado y el guardar se utiliza en el formulario “FrmAssembly” al presionar el botón “Add Device”.

SQL - Base de Datos

La base de datos se llama “Connected_db” y cuenta con las tablas para alojar el stock de dispositivos, componentes y para la información de los operadores.



En la raíz del proyecto se encuentra una carpeta “DB_backup” donde se encuentra el archivo para levantar el backup de la base de datos de la aplicación. En caso de no tener conexión con la base de datos, al intentar iniciar sesión da aviso de la falla.

Comenzando en el código, contamos con los métodos para guardar y leer la información mencionada anteriormente sobre las tablas.

Método para guardar la lista de dispositivos en la base de datos:

```
public static void SaveDevice(List<Device> devices)
{
    try
    {
        using (SqlConnection connection = new SqlConnection(connectionStr))
        {
            SqlCommand sqlCommand = new SqlCommand();
            connection.Open();
            sqlCommand.CommandType = System.Data.CommandType.Text;
            sqlCommand.Connection = connection;
            sqlCommand.CommandText = "INSERT INTO stockDevice VALUES (@serialNumber, @codeInternal, @typeDevice, @eValidation)";
            foreach (Device item in devices)
            {
                sqlCommand.Parameters.AddWithValue("@serialNumber", (long)item.SerialNumber);
                sqlCommand.Parameters.AddWithValue("@codeInternal", item.CodeInternal.ToString());
                sqlCommand.Parameters.AddWithValue("@typeDevice", item.TypeDevice.ToString());
                sqlCommand.Parameters.AddWithValue("@eValidation", item.EValidation.ToString());
                sqlCommand.ExecuteNonQuery();
                sqlCommand.Parameters.Clear();
            }
        }
    }
    catch (Exception)
    {
        throw new Exception("The list of devices could not be saved in the database.");
    }
}
```

El método se llama al presionar el botón “Upload List”, generando que los dispositivos creados se guarden en la base.

El método para levantar la información del stock de los dispositivos es:

```
public static List<Device> LoadDevice()
{
    List<Device> devices = new List<Device>();
    try
    {
        using (SqlConnection connection = new SqlConnection(connectionStr))
        {
            connection.Open();
            SqlCommand sqlCommand = new SqlCommand();
            sqlCommand.CommandType = System.Data.CommandType.Text;
            sqlCommand.Connection = connection;
            sqlCommand.CommandText = "SELECT * FROM stockDevice";
            using (SqlDataReader dataReader = sqlCommand.ExecuteReader())
            {
                while (dataReader.Read() != false)
                {
                    Enum.TryParse(dataReader["codeInternal"].ToString(), out ECode eCode);
                    Enum.TryParse(dataReader["typeDevice"].ToString(), out EType eType);
                    Enum.TryParse(dataReader["eValidation"].ToString(), out EValidation eValidation);
                    switch (eType)
                    {
                        case EType.AccessControl:
                            devices.Add(new AccessControl(Convert.ToDouble(dataReader["serialNumber"]), eCode, eType, eValidation));
                            break;
                        case EType.PanelAccess:
                            devices.Add(new AccessPanel(Convert.ToDouble(dataReader["serialNumber"]), eCode, eType, eValidation));
                            break;
                        case EType.Attendance:
                            devices.Add(new Attendance(Convert.ToDouble(dataReader["serialNumber"]), eCode, eType, eValidation));
                            break;
                    }
                }
            }
            return devices;
        }
    }
    catch (Exception)
    {
        throw new Exception("The device list was not found in the database.");
    }
}
```

Éste se llama en el load del formulario “FrmProduction”, para que cuando se inicie sesión la información ya se encuentre en sistema.

HILOS

Se utilizó un hilo para animar el logo de la aplicación, lo podemos encontrar en el formulario “FrmProduction” donde se instancia y se da inicio en el evento load:

```
private void FrmProduction_Load(object sender, EventArgs e)
{
    try
    {
        this.lblTitleOperator.Text = Login.OperatorLog.ToString();
        ButtonAssemblyFalse();
        Stock.DevicesStock = DAO.LoadDevice();
        Stock.ComponentsStock = DAO.LoadComponent();
        thread = new Thread(ThreadLogo);
        thread.Start();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "ERROR DATA BASE", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
```

Llamando al método que realiza la animación:

```
public void ThreadLogo()
{
    do
    {
        for (int i = 1; i < 13; i++)
        {
            this.pnlConnected.BackgroundImage = (Image)Properties.Resources.ResourceManager.GetObject($"TP4_LOGO_Hilos_{i}");
            System.Threading.Thread.Sleep(100);
        }
    } while (true);
}
```

Se utilizan varias imágenes para simular su animación:

EVENTOS Y DELEGADOS

Se utilizaron eventos y delegados para realizar el traspaso de un formulario a otro, cuando el operador acepta una orden de trabajo se lanza un evento para que el formulario principal cierre el “FrmInternalOrder” y abra el formulario “FrmAssembly”:

```
public delegate void CallBackOrder();
5 referencias
public partial class FrmInternalOrder : Form
{
    public event CallBackOrder ChangeForm;
```

El evento se invoca cuando se hace click en el botón “Accept Order” y se ejecuta en el siguiente método:

```
private void ChangeInternalOrder()
{
    if (!(this.ChangeForm is null))
    {
        this.ChangeForm.Invoke();
    }
}
```

En el formulario principal (FrmProduction), dentro del evento de click del botón que nos abre el formulario de ordenes de trabajo asignamos el manejador al evento:

```
private void btnOrder_Click(object sender, EventArgs e)
{
    FrmInternalOrder internalOrder = new FrmInternalOrder();
    internalOrder.ChangeForm += this.ChangeChild;
    OpenChildForm(internalOrder);
}
```

El método “ChangeChild” realiza la siguiente acción y también encadena otro evento

```
public void ChangeChild()
{
    this.btnAssembly.Enabled = true;
    ButtonAssemblyFormat();
    FrmAssembly assembly = new FrmAssembly();
    assembly.ChangeForm += this.ChangeChildAssemble;
    OpenChildForm(assembly);
}
```

MÉTODOS DE EXTENSIÓN

Se realizó un método de extensión para obtener un componente a través del nombre desde una lista de componentes:

```
public static Components DataExtension(this List<Components> components, string name)
{
    foreach (Components item in components)
    {
        if(item.NameComponent.ToString() == name)
        {
            return item;
        }
    }
    return null;
}
```

Se utiliza en el formulario “FrmComponents” para modificar el componente seleccionado en el datagrid en la base de datos:

```
private void btnAddComponents_Click(object sender, EventArgs e)
{
    try
    {
        Components components = (Components)this.dgvComponents.CurrentRow.DataBoundItem;
        int newCount = (int)this.nudCountComponents.Value;
        if (components + newCount)
        {
            this.dgvComponents.DataSource = null;
            this.dgvComponents.DataSource = Stock.ComponentsStock;
            DAO.ModifyComponent(Stock.ComponentsStock.DataExtension(components.NameComponent.ToString()));
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "NO SAVE COMPONENTS", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```