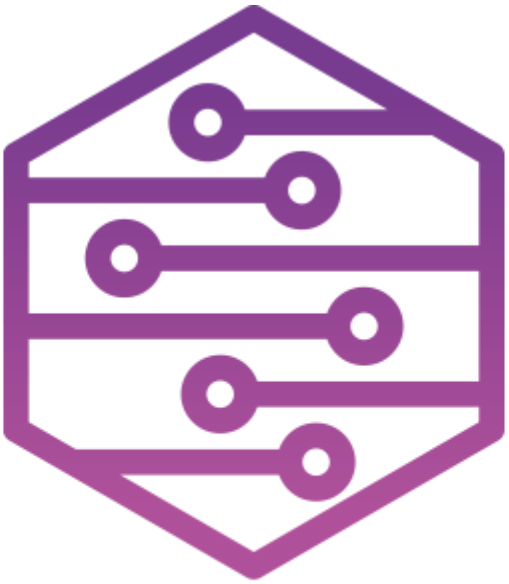
Trabajo Práctico Nº3



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.

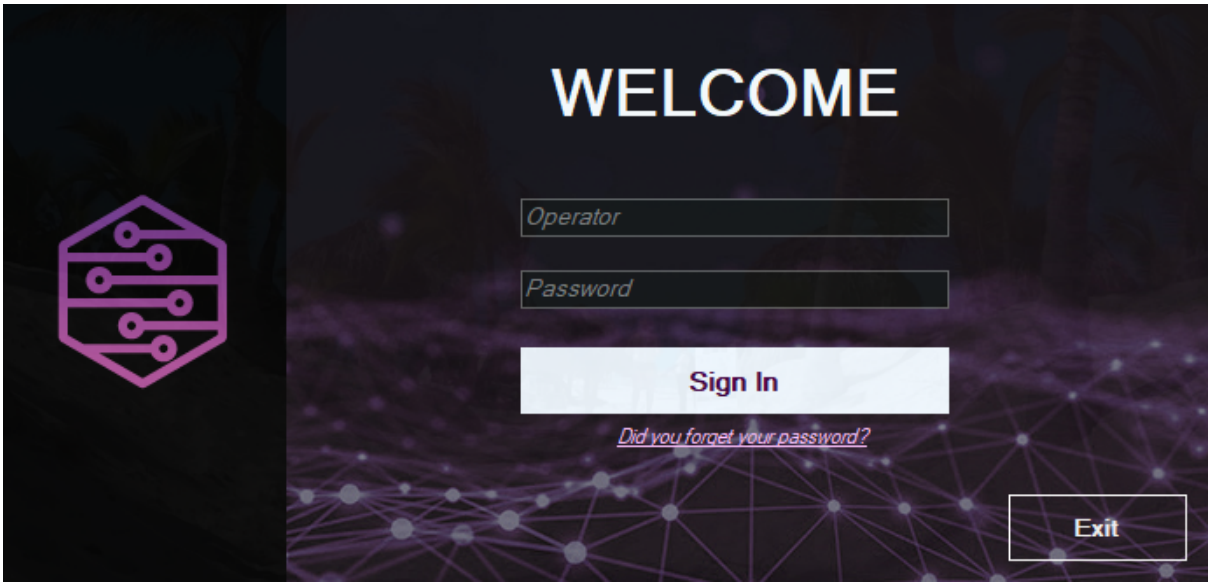
LOGIN

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:

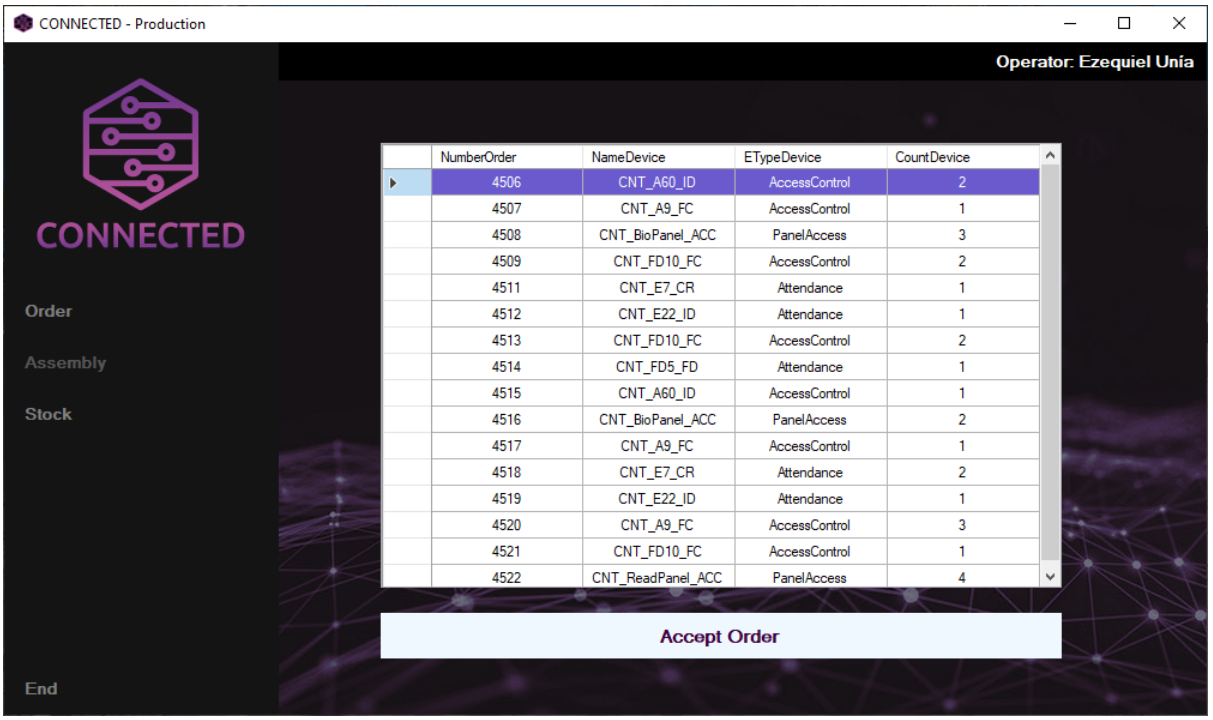
Nombre	Apellido	ID Usuario	Contraseña
Federico	Dávila	8080	8090
Ezequiel	Unía	6666	9999
Mauricio	Cerizza	7894	5020
Ezequiel	Oggioni	9707	3535
Esteban	Prietto	9595	1331
Lautaro	Galarza	4747	3030
Lucas	Rodríguez	1010	2020

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.

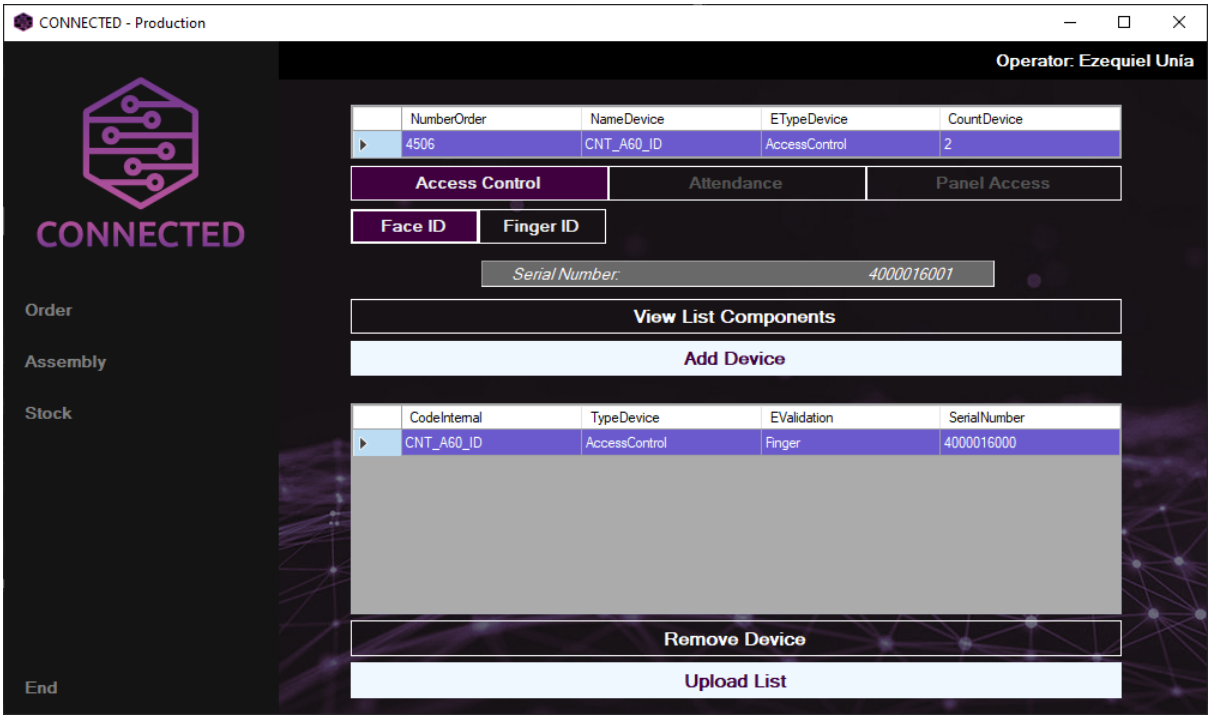


Al darle click al botón *“Accept Order”,* esta acción habilitará el apartado de ensamblado y lo redireccionará automáticamente.

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)

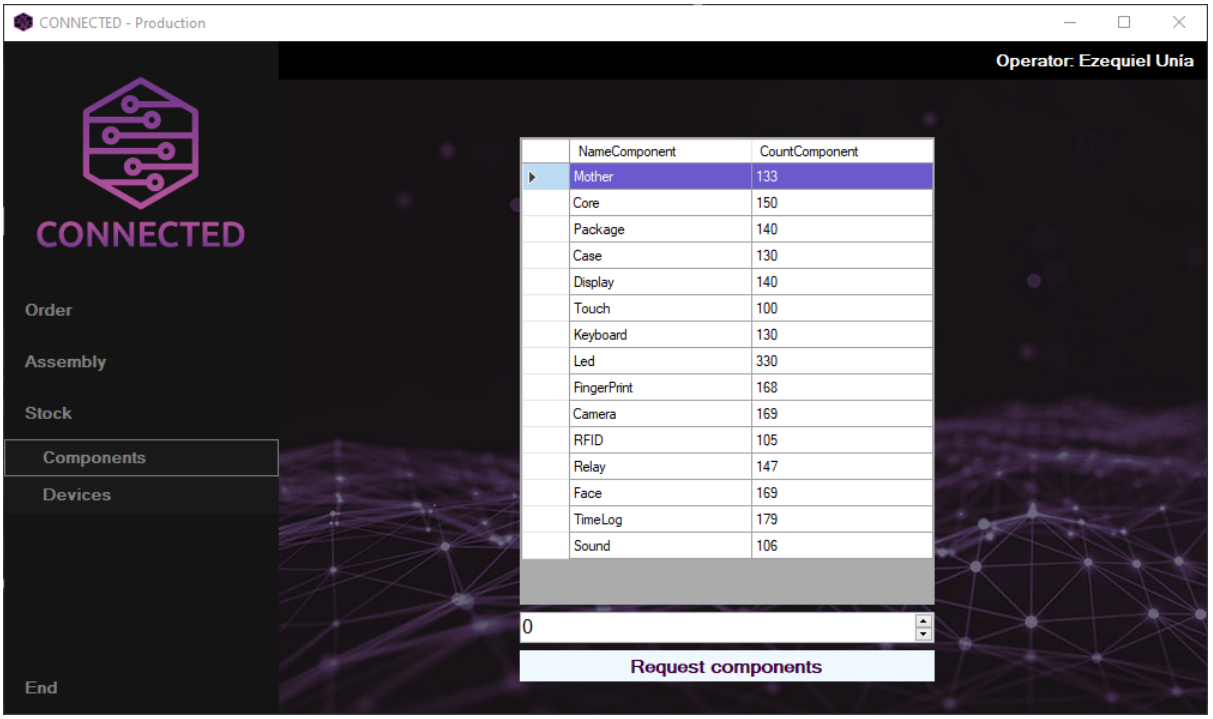
El botón *"Add Device"* nos va a fabricar el dispositivo seleccionado y lo va a almacenar en la mesa de trabajo. (Grilla inferior)

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

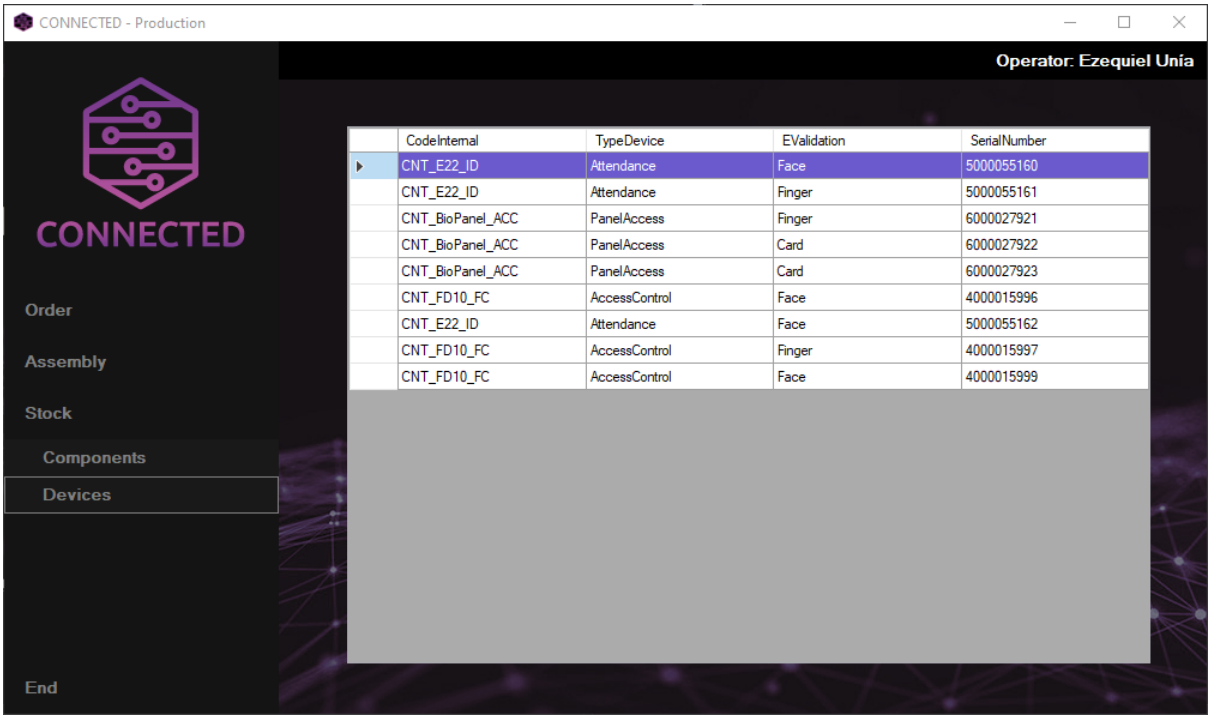
En el apartado “Stock - Components” obtendremos la información de la cantidad de componentes que se encuentren en stock.



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.



END - ABOUT

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

END - SIGN OFF

El apartado “Sign Off” desconectará al operador logueado.

END - 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;
    }
}
```