La aplicación permite la producción de tres tipos de consola de videojuegos,para ello entramos a la parte de producción, seleccionamos la consola deseada ,rellenamos los datos solicitados en la parte izquierda y luego procedemos con los botones para su produccion, dependiendo del tipo de usuario logueado tendrá más o menos opciones , por ejemplo un supervisor puede añadir más materia prima para la producción , mientras que un operario solo podrá chequear los valores.

El sistema también permite a un supervisor ,ver todos los usuarios registrados en el sistema y editar sus atributos o también eliminarlo de sistema .

Cuenta con una bodega donde se guardan las consolas una vez finalizada su producción.

Utilizo la herencia a la hora de crear las consolas, para ello tengo una clase abstracta

Que jamas será instanciada llamada Consola , de la que heredan las tres consolas que se pueden producir, La clase Consola cuenta con un método virtual , para que sus clases puedan sobrescribirla.

Utilice la sobrecarga de constructores para que me permita crear consolas para colocar en la bodega a fin de ser una especia de “historial” de consolas que fueron creadas con anterioridad.

Para gestionar el inventario utilizo la colección llamada Listas .

Los datos de la aplicación están manejados en una clase estatica llamda Datos,donde están las

Listas y métodos necesarios para gestionar la app, vendría a ser como una “base de datos”,

Donde están guardados los usuarios, la bodega con las consolas producidas etc.

Excepciones:

A lo largo del programa utilizo bloques try catch para evitar excepciones en tiempo de ejecución.

Para ello también cuento con dos excepciones propias:

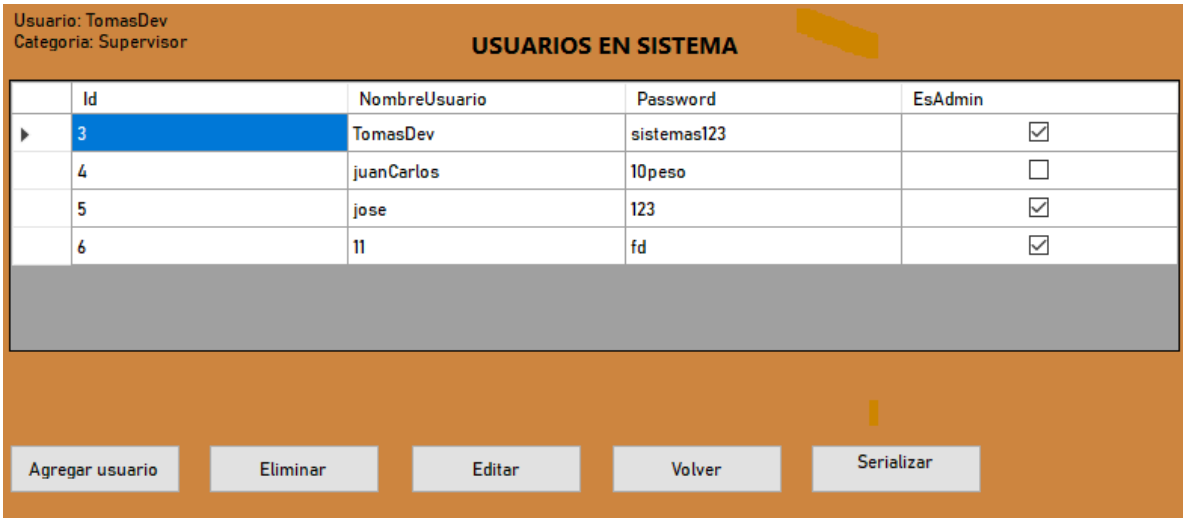
```
11 referencias
public class StockInsuficienteExeption : MiExcepcion
{
    4 referencias
    public StockInsuficienteExeption(string material):base($"No hay suficiente stock de : {material}, hable con un supervisor")
    {

    }
}
```

Aquí hay un ejemplo de una excepción que creé .La misma se lanzara si quiero fabricar una consola y no dispongo de material suficiente para ello.

Archivos y serializacion:

Cuento con una clase genérica llamada Archivos.

En ella cuento con los métodos necesarios para poder serializar y deserializar objetos.



Al mostrar los usuarios que hay en el sistema, se cuenta con un botón Serializar.

El mismo me creara un archivo xml como el siguiente:

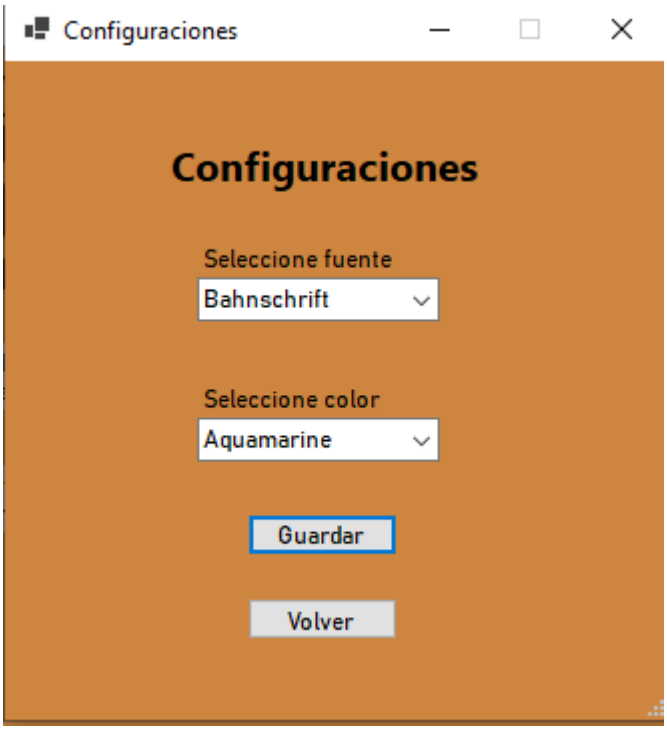
```
▼<ArrayOfUsuario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  ▼<Usuario>
     <Id>3</Id>
     <NombreUsuario>TomasDev</NombreUsuario>
     <Password>sistemas123</Password>
     <EsAdmin>true</EsAdmin>
   </Usuario>
  ▼<Usuario>
     <Id>4</Id>
     <NombreUsuario>juanCarlos</NombreUsuario>
     <Password>10peso</Password>
     <EsAdmin>false</EsAdmin>
   </Usuario>
  ▼<Usuario>
     <Id>5</Id>
     <NombreUsuario>jose</NombreUsuario>
     <Password>123</Password>
     <EsAdmin>true</EsAdmin>
   </Usuario>
 </ArrayOfUsuario>
```

Este método trabajara de forma asincrónica de la siguiente manera:

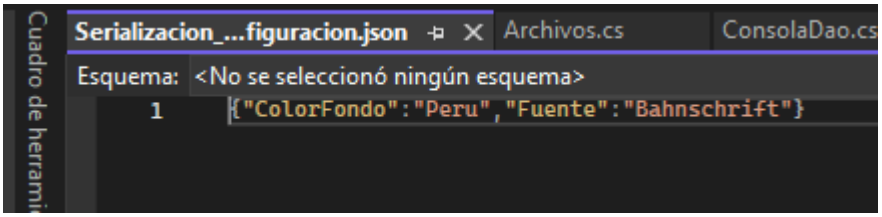
```
1 referencia
public static async Task EscribirXmlAsync(T datos, string archivo)
{
    string completa = ruta + @"\Serializacion_" + archivo + ".xml";
    try
    {
        if (!Directory.Exists(ruta))
        {
            Directory.CreateDirectory(ruta);
        }

        using (StreamWriter sw = new StreamWriter(completa))
        {
            XmlSerializer xmlSerializer = new XmlSerializer(typeof(T));
            await Task.Run(() => xmlSerializer.Serialize(sw, datos));
        }
    }
    catch (Exception)
    {
        throw new MiExcepcion("Error al serializar");
    }
}
```

De esta manera la escritura se realizara en un hilo alternativo, diferente al principal.

El programa cuenta con una opción de Configuraciones que es la siguiente:



Estas configuraciones son guardadas y leidas desde un archivo .json.



Cuento con dos interfaces.

Esta es una de ellas:

```
10 referencias
public interface IConfiguraciones
{
    21 referencias
    public void AplicarConfiguraciones();
}
```

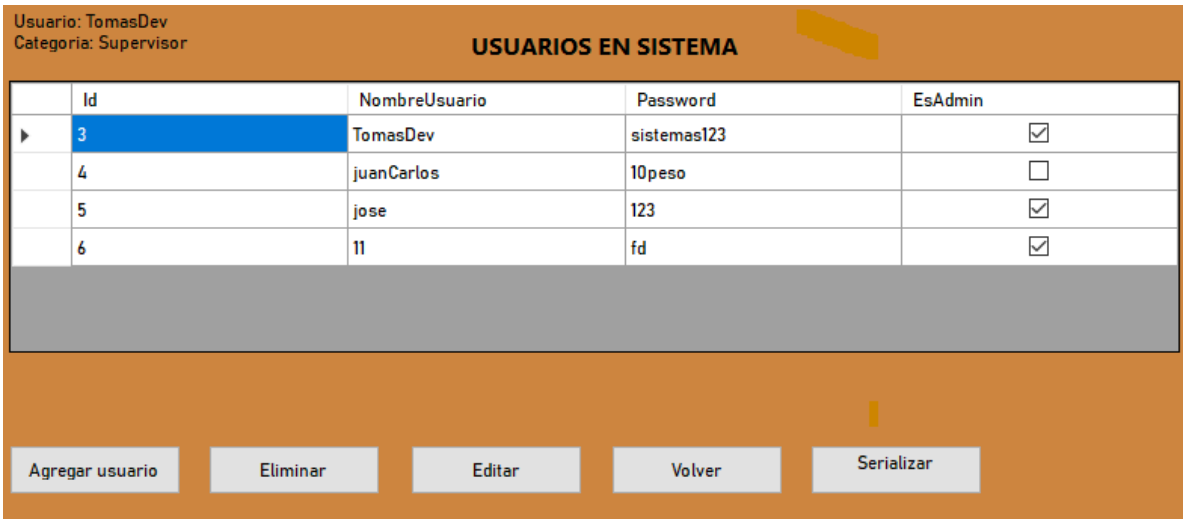
La cual la utilizaran todos los formularios para aplicar las configuraciones que son traídas desde un archivo .json de la siguiente manera:

```
public void AplicarConfiguraciones()
{
    Configuracion config = Archivos<Configuracion>.LeerConfiguracion("configuracion");
    FontFamily fontFamily = new FontFamily(config.Fuente);
    Font font = new Font(fontFamily, this.Font.Size, FontStyle.Regular);
    this.Font = font;
    this.BackColor = config.ColorFondo;

    this.Text = "Login -Salcedo Play-";
    this.txtBoxPassword.UseSystemPasswordChar = true;
}
```

Base de datos.

El programa cuenta con una base de datos.

Donde se guardan las consolas , usuarios y stock de materiales.



Aquí se pueden aplicar sentencias de tipo CRUD

Se pueden crear nuevos usuarios en el sistema.

Leer un usuario en especifico

Modificar sus atributos

Eliminar un usuario en especifico.

En la parte de la bodega cuento con un botón Ordenar por precio

El cual utilizara una expresión lambda para ordenar las consolas según su precio de manera ascendente.