Fecha:
21/09/2025

FORMULARIO MAQUILLAJE

PROgRAMACION VISUAL

Nombre del Alumno:

Andrea Dueñas de Luna.

Nombre de la Mestra:

Margarita Mondragon Arellano.

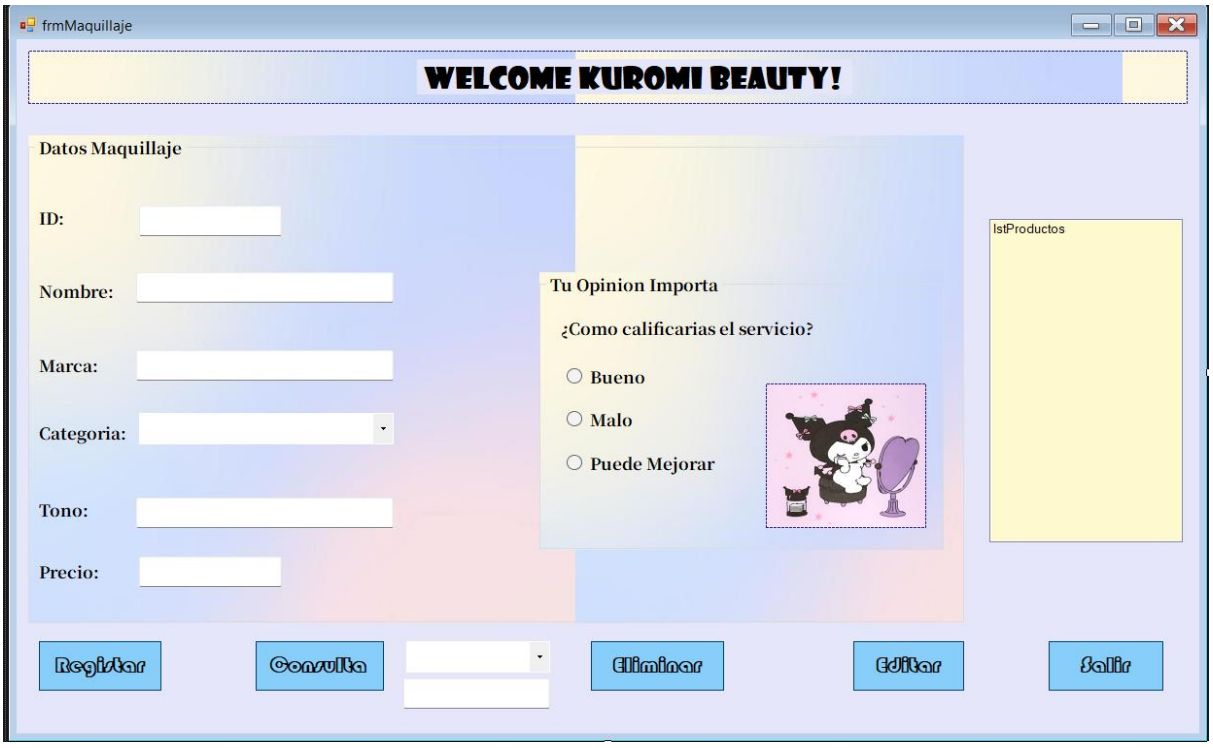
Introducción:

El tema que escogí fue maquillaje, al principio no tenia una idea tan concreta, pero al leer los requerimientos la idea se concretó más.

Mi formulario ingresa 10 productos por ID, nombre, marca, categoría (Cuidado de Piel, Brochas y/o Herramientas, Labios, Ojos y Rostro), tono y precio, además agregue un group box llamado “Tu opinión importa” donde cuenta con 3 opciones: Bueno, Malo y Puede Mejorar, además si se quieren agrega otro producto se muestra un mensagge box que dice “Inventario lleno”. Por otra parte, se muestra el ID y nombre del producto ingresados en la parte derecha en forma de lista, donde se puede seleccionar alguno para editar algún dato que se quiera cambiar apareciendo un mensaje que ha sido actualizado.

Las consultas se seleccionan por dos tipos por el nombre o id desde un combo box y se escribe el nombre del producto o id, apareciendo en la lista solo los que coincida con esa búsqueda, por último, pero no menos importante al seleccionar un producto de la lista y seleccionar el botón de eliminar te pregunta si realmente estas seguro de eliminarlo con un “SI” o “NO”. El ultimo botón es el e salir que cierra el programa y los datos ingresados, solo se mantienen mientras el programa este corriendo.

Diseño:



A continuación, voy a describir las herramientas utilizadas para la creación de mi formulario:

GroupBox (Datos Maquillaje y Tu opinión Importa): Para organizar mejor los datos datos y que se vea más estético y atractivo.

Label: Datos del maquillaje a ingresar.

PictureBox: Se agrego una imagen de Kurumi en el GroupBox de la opinión para hacerlo más atractivo y alusivo al nombre, también fue más que nada una decisión personal.

Button: Se puede observar que hay cuatro botones, cada uno con un nombre y funcionalidad diferente.

RadioButton: Se tiene 3 en el grupo de opinión que califican como fue el servicio, en una escala de Bueno, Malo y Puede Mejorar.

TextBox: Se visualizan 6 en el diseño, 5 de ellos son para ingresar los datos que se irán a registrar y el ultimo en la parte de consultas para escribir el Nombre o ID a buscar.

ListBox: Al momento de agregar un producto automáticamente se registrarán en forma de lista por ID – Nombre, tiene la propiedad de ser invisible solo será visto hasta que se ingrese el primer dato, así como también seleccionar para editar y/o eliminar.

ComboBox: Si bien fue algo que no vimos en clases me di a la tarea de curiosear un poco entre clases las herramientas ya que solo vimos solo las más principales para poder realizar un formulario, pero al momento de realizar mi diseño me di cuenta de que necesitaba usar algo que me permitiera ingresar varias opciones y seleccionar la que se prefiera y hay es cuando vi que necesitaba esa herramienta, lo mismo sucede con las consultas tenían que seleccionar la que se prefiriera.

Código:

```
5 referencias
public struct ProductoMaquillaje
{
    public int Id;
    public string Nombre;
    public string Marca;
    public string Categoria;
    public string Tono;
    public decimal Precio;
    public string Calificacion;
}
```

- Se utilizó un struct que agrupa todas las propiedades de un solo producto de maquillaje, que define qué información se guardará para cada elemento del inventario.
- **Id:** Se eligió un tipo entero (int) para almacenar un identificador numérico único para cada producto.
- **Nombre, Marca, Categoria, Tono, Calificacion:** Se usó un string para las cadenas de texto, ya que permiten almacenar cualquier combinación de letras, números y símbolos.
- **Precio:** Opte por el tipo decimal en lugar de double o float porque es el más preciso.

```
private ProductoMaquillaje[] inventario = new ProductoMaquillaje[10];
private int almacen = 0;
private int indiceSeleccionado = -1;
```

- **private ProductoMaquillaje[ ] inventario:** Esta es un arreglo que puede contener un máximo de 10 elementos, donde cada elemento es una estructura ProductoMaquillaje que funciona como un archivador con 10 cajones fijos, cumpliendo con los requisitos del proyecto.
- **private int almacen:** Esta variable es un contador. Su función es llevar la cuenta de cuántos productos se han registrado realmente en el inventario lo que permite que el programa sepa cuántos espacios están llenos, para no intentar leer espacios vacíos. Se inicializa en 0.
- **private int indiceSeleccionado:** Esta variable funciona como la **memoria** guarda el índice (la posición en la lista) del producto que el usuario ha seleccionado en el ListBox. Es indispensable para que los botones "Editar" y "Eliminar" sepan a qué producto del arreglo inventario deben aplicar la acción. Se inicializa en -1, un valor que indica que no hay ningún producto seleccionado al iniciar el programa.

```
private void LimpiarCampos()
{
    txtID.Clear();
    txtNom.Clear();
    txtMarca.Clear();
    cbCate.SelectedIndex = -1;
    txtTono.Clear();
    txtPrecio.Clear();

    rbBueno.Checked = false;
    rbMalo.Checked = false;
    rbPM.Checked = false;

    lstProductos.ClearSelected();
    indiceSeleccionado = -1;
    txtID.Focus();
}
```

Método privado que función es restablecerlo como estaba al su estado inicial, como si se acabara de abrir el programa. Es un método que se llama después de realizar acciones como registrar, editar o eliminar un producto, para dejar el formulario listo para la siguiente tarea.

```
2 referencias
private string ObtenerCalificacion()
{
    if (rbBueno.Checked) return "Bueno";
    if (rbMalo.Checked) return "Malo";
    if (rbPM.Checked) return "Puede Mejorar";
    return "Sin calificar";
}
```

Método privado llamado **ObtenerCalificacion** que devuelve un dato de tipo **string** (texto). Su única función será revisar cuál de los tres RadioButtons de calificación ha marcado el usuario y pasa revisado cual if fue seleccionado y esa selección a un texto que se pueda guardar, si el código llega a la última línea, significa que ninguno de los RadioButton estaba marcado. En este caso, devuelve un texto por defecto: "Sin calificar".

```
private void ActualizarLista()
{
    lstProductos.Items.Clear();

    for (int i = 0; i < almacen; i++)
    {

        lstProductos.Items.Add(inventario[i].Id + " - " + inventario[i].Nombre);
    }
}
```

Este bloque de código contiene el método ActualizarLista, su función es asegurarse de que el ListBox que ve el usuario refleje la información que está guardada en el arreglo inventario ya que se llama cada vez que se registra, edita o elimina un producto para que los cambios se vean reflejados inmediatamente.

```
private void btnRe_Click(object sender, EventArgs e)
{
    if (almacen >= 10) { MessageBox.Show("Inventario lleno."); return; }
    if (string.IsNullOrWhiteSpace(txtID.Text)) { MessageBox.Show("El ID es obligatorio.");
    return; }

    int idProducto;
    decimal precioProducto;

    if (!int.TryParse(txtID.Text, out idProducto))
    {
        MessageBox.Show("Error: El ID debe ser un numero.");
        return;
    }

    if (!decimal.TryParse(txtPrecio.Text, out precioProducto))
    {
        MessageBox.Show("Error: El Precio debe ser un numero.");
        return;
    }

    for (int i = 0; i < almacen; i++)
    {
        if (inventario[i].Id == idProducto)
        {
            MessageBox.Show("Error: Este ID ya existe.");
            return;
        }
    }

    ProductoMaquillaje nuevoProducto = new ProductoMaquillaje();
    nuevoProducto.Id = idProducto;
    nuevoProducto.Nombre = txtNom.Text;
    nuevoProducto.Marca = txtMarca.Text;
    nuevoProducto.Categoria = cbCate.Text;
    nuevoProducto.Tono = txtTono.Text;
    nuevoProducto.Precio = precioProducto;
    nuevoProducto.Calificacion = ObtenerCalificacion();

    inventario[almacen] = nuevoProducto;
    almacen++;

    if (!lstProductos.Visible)
    {
        lstProductos.Visible = true;
    }

    ActualizarLista();
    LimpiarCampos();
    MessageBox.Show("Producto registrado");
}
```

Este método se empieza a trabajar cuando el usuario hace clic en el botón "Registrar". Su función es validar los datos introducidos, crear un nuevo producto, y guardarlo en el inventario si toda la información es correcta si en el campo ID y Precio escriben letras, aparece un MessageBox que anuncia un error y que debe ser un numero en esos campos, destinados solo para números.

- **int.TryParse(txtID.Text, out idProducto):** intenta convertir el ID. Si lo logra, devuelve true y guarda el número en la variable idProducto.
- **Validación de ID Duplicado**: El bucle for recorre todos los productos ya guardados y compara el ID de cada uno con el nuevo ID. Si encuentra una coincidencia, avisa al usuario que el ID ya existe y detiene el proceso.
- **Creación y Llenado del Producto**: Solo si todas las validaciones anteriores fueron exitosas, el programa procede a crear una nueva "ficha" de producto (ProductoMaquillaje) y a rellenar cada una de sus propiedades con los datos de los controles del formulario.

- **Guardado en el Inventario**: La línea inventario[almacen] = nuevoProducto; guarda la ficha completa en el siguiente espacio disponible del arreglo. Inmediatamente después, almacen++; actualiza el contador para reflejar que ahora hay un producto más.
- **Actualización de la Interfaz**: Al final, se actualiza la parte visual del programa: se hace visible el ListBox si es la primera vez, se llama a ActualizarLista() para que el nuevo producto aparezca en la lista, se llama a LimpiarCampos() para reiniciar el formulario y, por último, un MessageBox le confirma al usuario que todo salió bien.

```
private void btnCon_Click(object sender, EventArgs e)
{
    if (cbConsul.SelectedItem == null || string.IsNullOrWhiteSpace(txtBusqueda.Text))
    {
        MessageBox.Show("Selecciona un tipo de consulta y escribe un valor.");
        return;
    }

    string tipoConsulta = cbConsul.SelectedItem.ToString();
    string valorBusqueda = txtBusqueda.Text;
    bool encontrado = false;

    for (int i = 0; i < almacen; i++)
    {
        if (tipoConsulta == "ID" && inventario[i].Id.ToString() == valorBusqueda)
        {
            lstProductos.SelectedIndex = i;
            encontrado = true;
            break;
        }
        else if (tipoConsulta == "Nombre" && inventario[i].Nombre.Equals(valorBusqueda, StringComparison.OrdinalIgnoreCase))
        {
            lstProductos.SelectedIndex = i;
            encontrado = true;
            break;
        }
    }

    if (!encontrado)
    {
        MessageBox.Show("No se encontró ningún producto.");
        lstProductos.ClearSelected();
    }
}
```

Su función es buscar un producto dentro del inventario, ya sea por ID o por Nombre, y seleccionarlo en la lista para que el usuario lo vea.

- **Validación if (cbConsul.SelectedItem == null || ...)**: La primera línea es una validación. Revisa dos cosas: que el usuario haya seleccionado un tipo de consulta ("ID" o "Nombre") en el ComboBox y que no haya dejado el TextBox de búsqueda vacío. El operador || significa "o". Si alguna de las dos condiciones se cumple, muestra un mensaje de error y el return; detiene la ejecución.
- **bool encontrado = false;** : Se crea una variable "bandera" que empieza en false. Si el programa encuentra un producto, esta variable cambiará a true.
- **Búsqueda en el Inventario for (int i = 0; i < almacen; i++)**: Se inicia un bucle que recorrerá cada producto guardado en el inventario.

- **lstProductos.SelectedIndex = i;**: Si encuentra una coincidencia (ya sea por ID o por Nombre), esta línea selecciona y resalta el producto correspondiente en el ListBox.
- **break;**: Detiene el bucle for inmediatamente. Esto hace el programa más eficiente, ya que si encontró el producto, no necesita seguir buscando en el resto del inventario.

```
1 referencia
private void btnEli_Click(object sender, EventArgs e)
{
    if (indiceSeleccionado < 0)
        {
            MessageBox.Show("Por favor, selecciona un producto de la lista para eliminar.");
            return;
        }

        if (MessageBox.Show("¿Estás seguro de que quieres eliminar este producto?", "Confirmar Eliminación", MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
        {
            for (int i = indiceSeleccionado; i < almacen - 1; i++)
            {
                inventario[i] = inventario[i + 1];
            }
            almacen--;

            ActualizarLista();
            LimpiarCampos();
            MessageBox.Show("Producto eliminado.");
        }
    }
```

Su función es quitar de forma permanente un producto del inventario, pero con medidas de seguridad para evitar borrados accidentales.

```
private void btnEditar_Click(object sender, EventArgs e)
{

    if (indiceSeleccionado < 0)
    {
        MessageBox.Show("Selecciona un producto de la lista para editar.");
        return;
    }

    int idProducto;
    decimal precioProducto;

    if (!int.TryParse(txtID.Text, out idProducto))
    {
        MessageBox.Show("Error: El ID debe ser un numero.");
        return;
    }

    if (!decimal.TryParse(txtPrecio.Text, out precioProducto))
    {
        MessageBox.Show("Error: El Precio debe ser un numero.");
        return;
    }


    inventario[indiceSeleccionado].Id = idProducto;
    inventario[indiceSeleccionado].Nombre = txtNom.Text;
    inventario[indiceSeleccionado].Marca = txtMarca.Text;
    inventario[indiceSeleccionado].Categoria = cbCate.Text;
    inventario[indiceSeleccionado].Tono = txtTono.Text;
    inventario[indiceSeleccionado].Precio = precioProducto;
    inventario[indiceSeleccionado].Calificacion = ObtenerCalificacion();

    ActualizarLista();
    LimpiarCampos();
    MessageBox.Show("Producto actualizado.");
}
```

Su función es tomar los datos modificados por el usuario en el formulario, validarlos de forma segura, y actualizar el producto correspondiente que ya existe en el inventario.

- La primera línea es la más importante porque revisa si el usuario ha seleccionado un producto de la lista **(indiceSeleccionado < 0**). Si no ha seleccionado nada, muestra un mensaje de advertencia y el return; detiene la ejecución para no modificar un producto que no existe.
- Si el usuario introduce letras, la condición se cumple, se muestra un mensaje de error claro y el return; detiene el proceso. Se repite la misma lógica para el Precio con **decimal.TryParse.**
- Si todas las validaciones anteriores fueron exitosas, el programa procede a actualizar.
- Una vez guardados los cambios en la memoria, se llama a **ActualizarLista()** para que la lista (ListBox) muestre la información actualizada.
- Se llama a **LimpiarCampos()** para restablecer el formulario.
- Finalmente, un MessageBox le confirma al usuario que el producto fue actualizado de manera exitosa.

```
1 referencia
private void btnSalir_Click(object sender, EventArgs e)
{
    this.Close();
}
```

- **this** se refiere al objeto actual, que en este caso es el propio formulario (frmMaquillaje).
- **.Close()** es un método que tienen todos los formularios en Windows Forms y su trabajo es cerrarlo de forma segura.

```
1 referencia
private void lstProductos_SelectedIndexChanged(object sender, EventArgs e)
{

    indiceSeleccionado = lstProductos.SelectedIndex;


    if (indiceSeleccionado >= 0)
    {

        ProductoMaquillaje productoSeleccionado = inventario[indiceSeleccionado];
        txtID.Text = productoSeleccionado.Id.ToString();
        txtNom.Text = productoSeleccionado.Nombre;
        txtMarca.Text = productoSeleccionado.Marca;
        cbCate.Text = productoSeleccionado.Categoria;
        txtTono.Text = productoSeleccionado.Tono;
        txtPrecio.Text = productoSeleccionado.Precio.ToString("0.0");

        if (productoSeleccionado.Calificacion == "Bueno")
        {
            rbBueno.Checked = true;
        }
        else if (productoSeleccionado.Calificacion == "Malo")
        {
            rbMalo.Checked = true;
        }
        else if (productoSeleccionado.Calificacion == "Puede Mejorar")
        {
            rbPM.Checked = true;
        }
    }
```

Se activa automáticamente cada vez que el usuario hace clic y selecciona un item diferente en la lista de productos. Su función es tomar los datos completos del producto seleccionado y mostrarlos en los controles del formulario.

- **Captura de la Selección**: **indiceSeleccionado = lstProductos.SelectedIndex;**: Esta línea obtiene el número de la fila que el usuario acaba de seleccionar y lo guarda en la variable **indiceSeleccionado**. Si el usuario selecciona la primera fila, indiceSeleccionado será 0.
- **Validación if (indiceSeleccionado >= 0)**: Se asegura de que el usuario haya hecho clic en un item válido. Si el usuario quita la selección, el **SelectedIndex** puede ser -1, y este if evita que el programa intente buscar un producto en una posición que no existe.
- **Búsqueda del Producto**: **ProductoMaquillaje productoSeleccionado = inventario[indiceSeleccionado];:** Usa el índice guardado para ir al "archivador" (inventario) y sacar la "ficha" completa del producto correspondiente.
- **Relleno de campos: txtID.Text = productoSeleccionado.Id.ToString();**: Esta y las líneas siguientes toman cada dato de la ficha del producto (**productoSeleccionado**) y lo "escriben" en el control correspondiente del formulario (TextBox, ComboBox, etc.), permitiendo que el usuario vea toda la información.

- **.ToString("0.00")**: En el caso del precio, se usa este formato para asegurar que siempre se muestre con dos decimales, como es estándar para la moneda.
- **Selección de la Calificación**: El bloque final de **if-else if** lee el texto de la calificación que estaba guardado (ej. "Bueno") y marca el RadioButton correspondiente (rbBueno.Checked = true;), mostrando lo que se había registrado para ese producto.

Referencias:

https://www.incanatoit.com/2014/11/formularios-controles-programacion-csharp-net.html

https://learn.microsoft.com/es-es/aspnet/web-forms/overview/ajax-control-toolkit/combobox/how-do-i-use-the-combobox-control-cs

https://www.youtube.com/watch?v=41ImYzBMs5E

Mi video:

https://youtu.be/_GRVZMfxm2Y?si=2Zv_UKye1AXq0EA1