

UNIVERSIDAD DON BOSCO
DEPARTAMENTO DE ESTUDIOS TECNOLOGICOS
Desarrollo de Aplicaciones con Software Propietario
DSP404



Código EVA111: Proyecto Final
Categoría GRUPAL

GRUPO 02L

INTEGRANTES:

Apellidos	Nombres
Alejo Gálvez	Víctor Alejandro
Roque Realeño	Sergio Daniel
Velásquez Rivas	Álvaro Aníbal

Carnet
AG181471
RR180598
VR180536

DOCENTE:
Karens Lorena Medrano

FECHA DE PRESENTACIÓN:
Martes, 6 de noviembre de 2018

Contenido


MANUAL ORIENTADO AL USUARIO.	3
Desktop.....	3
Login.....	3
Categoría.....	5
Empleados.....	6
Cuenta.	7
Web.....	8
Inicio.....	8
Registro.	9
Catálogo.	10
MANUAL ORIENTADO AL PROGRAMADOR.....	11
Creación de clases	11
Funcionalidad para diseño	18
AddForm.....	18
CatalogoForm	22

MANUAL ORIENTADO AL USUARIO.


En este apartado explicaremos como el usuario deberá interactuar con el sitio para asegurar la mejor experiencia tanto en web como desktop.


Desktop.


Login.

Iniciar Sesion  


Sistema de administracion

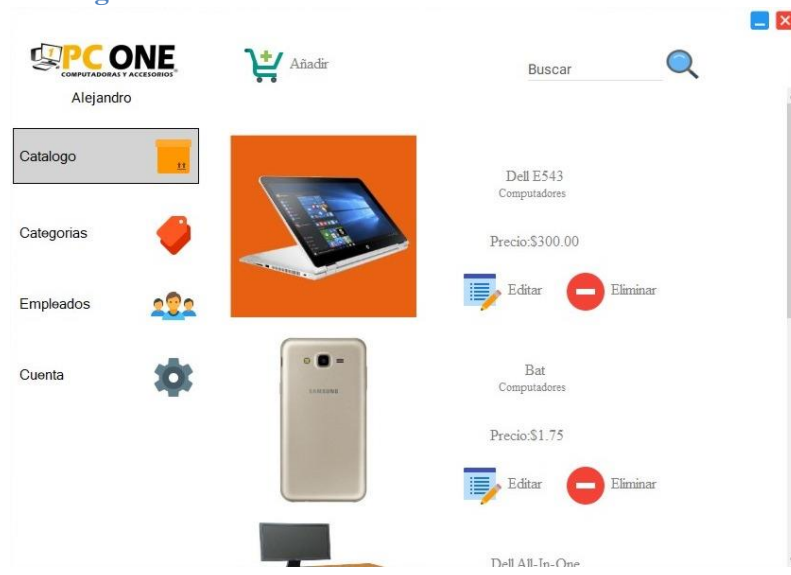




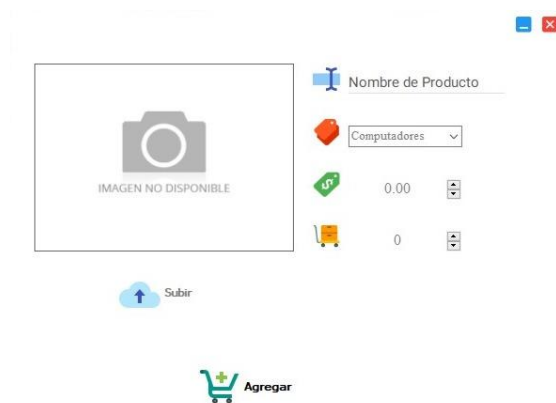
 Iniciar Sesion

Para empezar, tenemos lo que es el login. El administrador del sitio necesita ser alguien de confianza con la empresa y de preferencia las personas que puedan manejar esta manipulación de datos sean pocas para mayor seguridad, nosotros como equipo hemos diseñado un login amigable y sencillo pero a la vez muy seguro. Acá solo deberá escribir correctamente sus credenciales y podrá ingresar al panel de administradores.

Catálogo.



Al ingresar nos encontramos con todas las opciones correspondientes que como administrador tiene acceso. Lo primero es el apartado catalogo donde usted puede observar el catalogo disponible editar cada artículo, borrar o agregar uno nuevo si así lo desea.



Al momento de agregar o editar usted puede seleccionar la imagen que se visualizara en la web, así como agregar su nombre, asignarle una categoría, el precio las unidades disponibles.

Categoría.



En el apartado categorías el usuario puede ver las categorías disponibles, editarlas, borrarlas o agregar nuevas si así lo desea.



Al agregar o editar una categoría usted podrá nombrar la categoría, agregar una breve descripción y subir una fotografía que las identifique.

Empleados.

PC ONE
COMPUTADORAS Y ACCESORIOS
Alejandro

Catalogo

Categorías

Empleados

Cuenta

Ficha de Empleado

Administrador

23355681

Administrador@gmail.com

78182424

500.00

Administrador

Ficha de Empleado

Alejandro

47591657-0

alejandroalejo714@gmail.com

78128424

650.00

Administrador

En el apartado usuarios el administrador puede ver y controlar quienes son las personas con acceso a la aplicación desktop y la que posee el control sobre todo el inventario de la tienda. Aquí se podrá añadir un nuevo empleado o un nuevo cargo que será usado para identificar al empleado.

Agregar Empleado

Nombres

DUI

Email

Contraseña

Teléfono

5/11/2018

300.00

Agregar

Al agregar un empleado deberá ingresar su nombre su número de DUI su correo que también será su nombre de usuario, su contraseña para acceder al sitio, su teléfono celular, su fecha de nacimiento, su sueldo y el cargo al que pertenece.

Agregar Cargo

Ejecutivo

Ejecutivo de la empresa

Agregar

Al agregar un cargo deberá nombrar el tipo de cargo y una breve descripción sobre que desempeñara la persona asignada a ese cargo.

Cuenta.

PC ONE

COMPUTADORAS Y ACCESORIOS

Administrador

Datos del Usuario

Catalogo

Categorias

Empleados

Cuenta

Administrador

23355681

Administrador@gmail.com

78182424

500.00

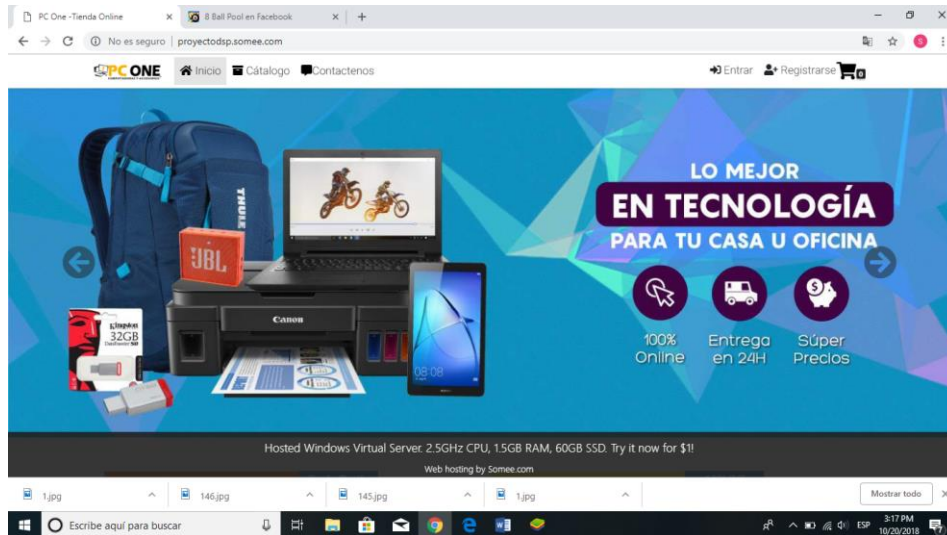
Administrador

14/7/2000 00:00:00

Por ultimo tenemos el apartado cuenta donde el usuario podrá visualizar todos sus datos.

Web.

Inicio.



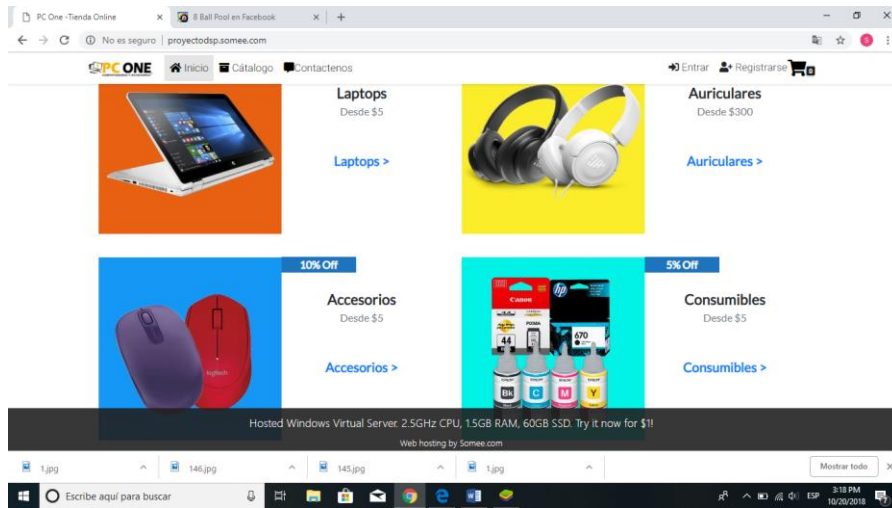
La página tiene un diseño sencillo y amigable con el cliente donde el cliente podrá visualizar todos los productos que estén agregados en el catalogo mediante la aplicación Desktop.

Registro.

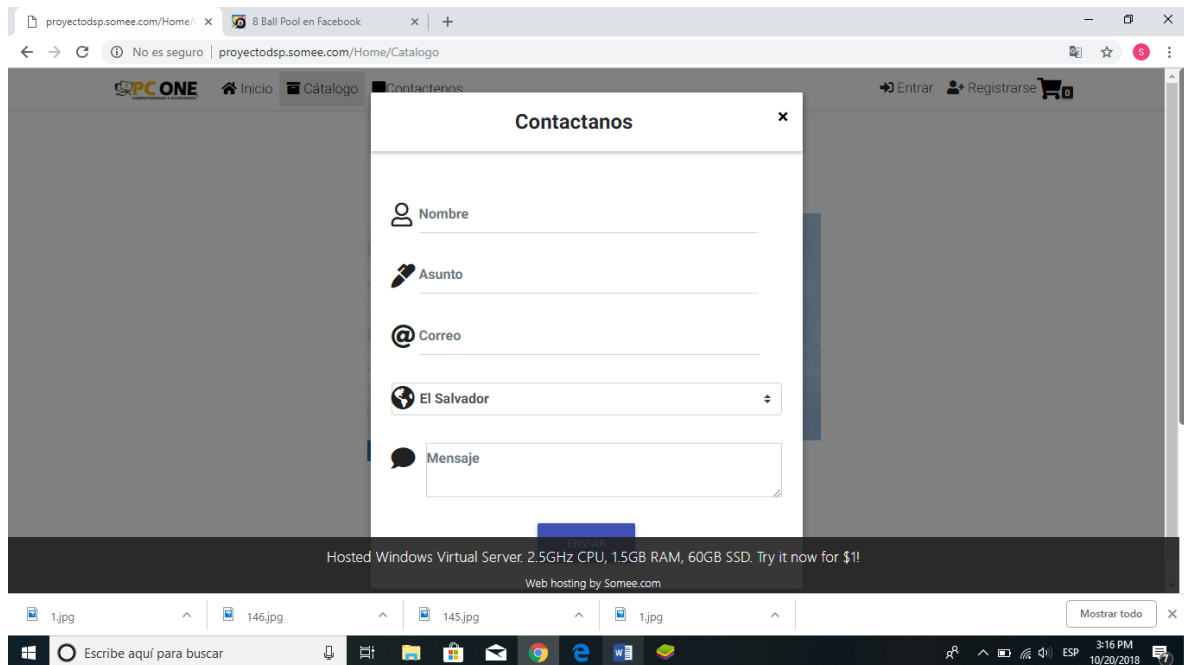
The screenshot displays a web browser window with the address bar showing `proyectodsp.somee.com/Home/Catalogo`. The website's navigation bar includes links for `Inicio`, `Catalogo`, and `Contactenos`, along with `Entrar` and `Registrarse` buttons. A modal window titled `Registro` is centered on the screen, featuring three input fields: `Nombre` (with a person icon), `Email` (with an envelope icon), and `Contraseña` (with a lock icon and a toggle eye icon). A blue `REGISTRARSE` button is positioned below the fields. The browser's status bar at the bottom indicates the page is not secure and provides server specifications: `Hosted Windows Virtual Server. 2.5GHz CPU, 1.5GB RAM, 60GB SSD. Try it now for $1!`. The Windows taskbar at the very bottom shows the search bar and various application icons.

Si el cliente no posee registro para hacer compras solo debe registrarse agregando su nombre de usuario su correo electrónico el cual sera el medio que la empresa usara para poder contactarlo y una contraseña que posea por lo menos una letra mayúscula una letra minúscula y un número.

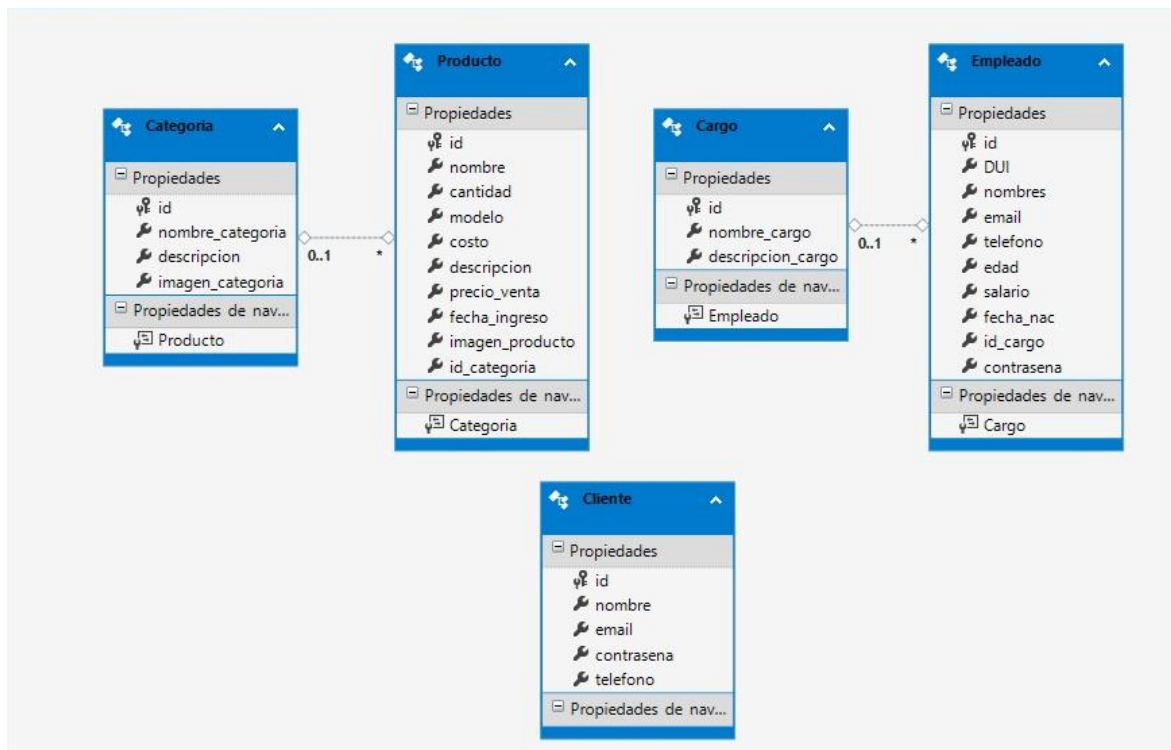
Catálogo.



En el catálogo el cliente puede ver todos los artículos disponibles a la venta, podrá consultar su precio y podrá agregarlos al carrito de comparas para luego agregar una forma de pago. También podrá contactar la empresa en caso de que el cliente posea alguna duda o tenga alguna queja en cuanto a servicio.



MANUAL ORIENTADO AL PROGRAMADOR.



Creación de clases

En esta parte el programador debe tener en cuenta varios de los aspectos de las aplicaciones que desarrollará en las cuales podría especificar procesos que tendrá que llevar a cabo en función a características de la misma en nuestro caso es el desarrollo de una tienda de artículos computacionales.

Se debe analizar cuantas clases se ocuparán para cada función de la aplicación en nuestro caso una clase para login, catalogo y producto.

En la clase login se utilizan propiedades como nombre y contraseña las cuales se utilizan como parámetros en un método llamado IniciarSesion el cual se encarga de establecer una conexión abierta con una base de datos de SQL server inicializando una instancia de una clase dada una cadena de conexión que obtiene una serie de datos para dicha configuración actual de la aplicación “PcOne”.

```
7 namespace DSPDesktop.Class
8 {
9     2 referencias
10    class Login
11    {
12        0 referencias
13        public string Nombre { get; set; }
14        0 referencias
15        public string Contraseña { get; set; }
16
17        1 referencia
18        public bool IniciarSesion(string Usuario, string Contraseña)
19        {
20            SqlConnection conn = new SqlConnection(ConfigurationManager.ConnectionStrings["PcOne"].ConnectionString);
21            try {
22                conn.Open();
23            }
24            catch (Exception e)
25            {
26                System.Windows.Forms.MessageBox.Show("No se pudo Conectar, Posible error de red" );
27            }
28        }
29    }
30 }
```

Para este proceso se declara una variable llamada conn con la cual se aplica la abertura para una conexión a una base de datos con los valores que se le especifica según la clase sqlconnection en la cual si la conexión representa algún error se muestra un mensaje notificando dicho error pero por medio de un if si el estado de la conexión está abierta en el cual si todo va bien entonces se crea un objeto en el cual se envía la propiedad sqlcommand el cual hace uso de un if en el cual por medio de una variable llamada ;reader se obtiene un valor el cual indica si sqlDataReader contiene una o más filas.

```
18 SqlConnection conn = new SqlConnection(ConfigurationManager.ConnectionStrings["PcOne"].ConnectionString);
19 try {
20     conn.Open();
21 }
22 catch (Exception e)
23 {
24     System.Windows.Forms.MessageBox.Show("No se pudo Conectar, Posible error de red" );
25 }
26
27 if(conn.State == System.Data.ConnectionState.Open)
28 {
29     string Query = string.Format("SELECT* FROM administradores WHERE Nombre = '{0}'", Usuario);
30     SqlCommand command = new SqlCommand(Query, conn);
31     SqlDataReader reader = command.ExecuteReader();
32     if (!reader.HasRows)
33     {
34         conn.Close();
35         return false;
36     }
37     else
38     {
39         while (reader.Read()) {
```

Como último proceso de esta clase es la obtención de la cadena de texto de la contraseña ingresada por medio de un if la cual si es correcta retornará true de lo contrario será false.

```

32     if (!reader.HasRows)
33     {
34         conn.Close();
35         return false;
36     }
37     else
38     {
39         while (reader.Read()) {
40             if (reader.GetString(1) == Contraseña)
41             {
42                 conn.Close();
43                 return true;
44             }
45             else
46             {
47                 return false;
48             }
49         }
50     }
51 }
52 }
53 }

```

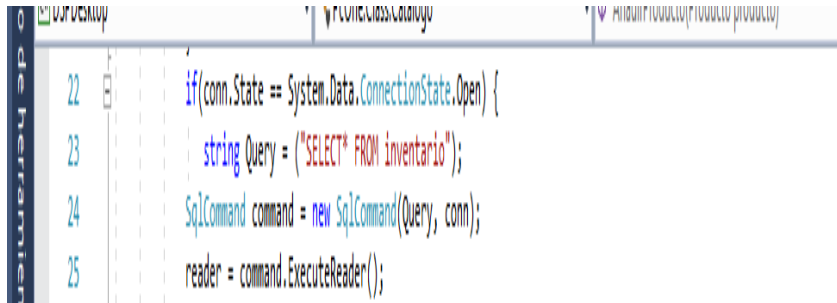
En la clase de catálogo se puede visualizar igual que en la anterior la representación de una conexión abierta hacia la base de datos la cual se vinculara con la aplicación creada por medio de la proporción del acceso de los archivos en la configuración que servirán en el desarrollo de las aplicaciones en el cliente, luego de eso se presentan una lista de objetos fuertemente tipados a la que se puede tener acceso por medio de un índice con el cual se proporcionan métodos con los cuales se pueden buscar ordenar y manipular listas con la cual se inicializa un método llamado obtener productos con el cual se crea una variable local llamada lista y se inicializa una clase llamada list que tiene una capacidad predeterminada en la que se usa sqlDataReader el cual ofrece una manera de leer un flujo de filas desde una base de datos SQL server por medio de la variable local llamada reader en su valor obtenido null en la cual se abre la conexión dado el caso de que haya alguna anomalía entonces se efectua Exception en el método catch el cual sirve para poder especificar errores que se dan cuando se ejecuta la aplicación si esto sucede entonces se muestra un mensaje el cual dirá error de conexión.

```

4 using system;
5
6 namespace PcOne.Class
7 {
8     4 referencias
9     class Catalogo
10    {
11        SqlConnection conn = new SqlConnection(ConfigurationManager.ConnectionStrings["PcOne"].ConnectionString);
12
13        1 referencia
14        public List<Producto> ObtenerProductos() {
15            List<Producto> Lista = new List<Producto>();
16            SqlDataReader reader = null;
17            try {
18                conn.Open();
19            }
20            catch (Exception e)
21            {
22                System.Windows.Forms.MessageBox.Show("Error en la conexion");
23            }
24            if (conn.State == System.Data.ConnectionState.Open) {
25                string Query = ("SELECT* FROM inventario");
26                SqlCommand command = new SqlCommand(Query, conn);

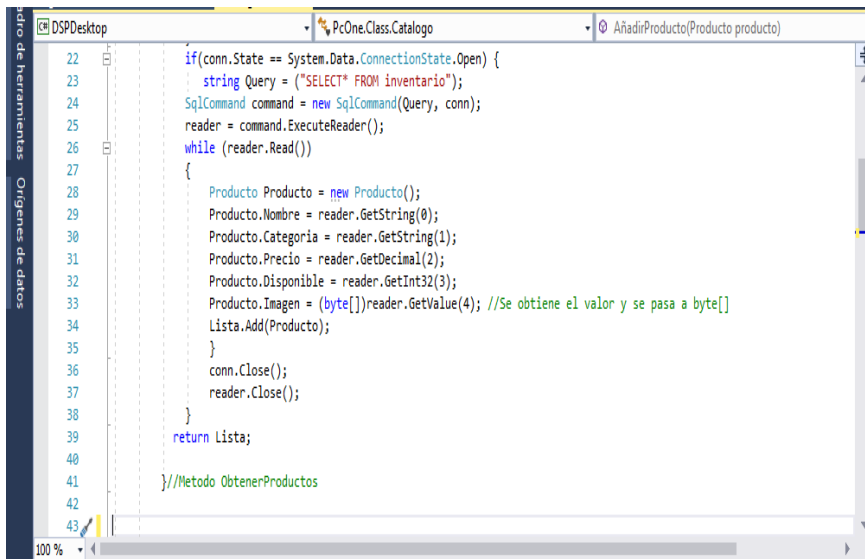
```

Luego de esto utilizamos un if en el cual se evalúa si el estado de la conexión según la variable local que manipula la conexión que se establece si esta conexión esta en su propiedad open entonces se hace uso de SqlCommand con el cual se representa el procedimiento almacenado que se ejecuta en la base de datos.



```
22 if(conn.State == System.Data.ConnectionState.Open) {  
23     string Query = ("SELECT* FROM inventario");  
24     SqlCommand command = new SqlCommand(Query, conn);  
25     reader = command.ExecuteReader();
```

Luego se muestra el manejo de una lista de una clase llamada producto esto haciendo uso de un while que tiene como parámetros la propiedad Read de la variable local reader la cual cuenta con la inicialización de un nuevo objeto el cual obtiene valores según las propiedades que se especifican con las cuales dichos valores se convierten a bytes luego de esto se cierra la conexión con la base de datos con el método close() y luego se retorna la lista utilizada.



```
22 if(conn.State == System.Data.ConnectionState.Open) {
23     string Query = ("SELECT* FROM inventario");
24     SqlCommand command = new SqlCommand(Query, conn);
25     reader = command.ExecuteReader();
26     while (reader.Read())
27     {
28         Producto Producto = new Producto();
29         Producto.Nombre = reader.GetString(0);
30         Producto.Categoria = reader.GetString(1);
31         Producto.Precio = reader.GetDecimal(2);
32         Producto.Disponible = reader.GetInt32(3);
33         Producto.Imagen = (byte[])reader.GetValue(4); //Se obtiene el valor y se pasa a byte[]
34         Lista.Add(Producto);
35     }
36     conn.Close();
37     reader.Close();
38 }
39 return Lista;
40
41 }//Metodo ObtenerProductos
42
43
```

Luego tenemos el método añadir producto el cual tiene como parámetro la clase producto con su respectivo objeto producto el cual evalúa si la conexión a la base de datos está abierta y si no se muestra un mensaje diciendo no se puede conectar pero si está todo funcionando correctamente entonces un if se encarga de verificar el estado del origen de los datos en donde se pide insertar valores para cada uno de los campos que se visualizan según una nueva clase instanciada con el texto de una consulta y una conexión la cual se maneja con una variable local llamada command la cual obtiene la estructura parameterCollection en el cual se agrega un valor final a la colección.

```
1 referencia
44 public bool AñadirProducto(Producto producto)
45 {
46     try
47     {
48         conn.Open();
49     }
50     catch(Exception e)
51     {
52         System.Windows.Forms.MessageBox.Show("No se pudo Conectar");
53     }
54     if (conn.State == System.Data.ConnectionState.Open)
55     {
56         using (conn) {
57             string Query = ("INSERT INTO inventario VALUES(@Nombre,@Categoria,@Precio,@Disponible,@Imagen)");
58             SqlCommand command = new SqlCommand(Query, conn);
59             command.Parameters.AddWithValue("@Nombre", producto.Nombre);
60             command.Parameters.AddWithValue("@Categoria", producto.Categoria);
61             command.Parameters.AddWithValue("@Precio", producto.Precio);
62             command.Parameters.AddWithValue("@Disponible", producto.Disponible);
63             command.Parameters.AddWithValue("@Imagen", producto.Imagen);
64             command.ExecuteNonQuery();
65         }
66     }
67 }
```

En lo siguiente usamos un método de tipo bool llamado **Eliminarproducto** que cuenta con un parámetro de tipo string llamado **Todelete** el cual evalúa si la conexión está abierta o cerrada si está abierta entonces la variable local de tipo string llamada **Query** se encarga de que una cadena de texto en su propiedad **Format** que reemplaza uno o varios elementos del formato de una cadena especificado el cual obtiene el parámetro **Todelete** con el cual se debe utilizar **sqlcommand** para poder hacer una nueva consulta para poder realizar lo que resta del procedimiento para luego cerrar la conexión.


```

1 referencia
74 public bool EliminarProducto(string ToDelete)
75 {
76     try
77     {
78         conn.Open();
79     }
80     catch (Exception e)
81     {
82         System.Windows.Forms.MessageBox.Show("No se pudo Conectar");
83     }
84     if (conn.State == System.Data.ConnectionState.Open)
85     {
86         using (conn)
87         {
88             string Query = String.Format("DELETE FROM inventario WHERE Nombre = '{0}'", ToDelete);
89             SqlCommand comm = new SqlCommand(Query, conn);
90             comm.ExecuteNonQuery();
91             comm.Dispose();
92             conn.Close();
93         }
94         return true;

```

Pasando a la clase de producto en la cual consta solamente de que el programador como tal se encargue de personalizar las propiedades o características que podría solicitar al usuario.

```

2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace PcOne.Class
8 {
9     11 referencias
10     class Producto
11     {
12         6 referencias
13         public string Nombre { get; set; }
14         4 referencias
15         public string Categoria { get; set; }
16         4 referencias
17         public decimal Precio { get; set; }
18         3 referencias
19         public int Disponible { get; set; }
20         4 referencias
21         public Byte[] Imagen { get; set; } //Objeto tipo BLOB guarda una imagen
22     }
23 }

```

Funcionalidad para diseño

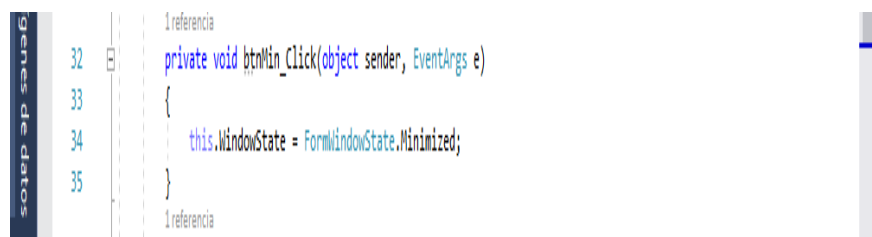
AddForm

Para poder realizar la parte de añadir producto se debe tomar en cuenta lo que le va a pedir al administrador que ingrese cuando quiera actualizar su disponibilidad de productos.

Se utiliza un método para poder realizar un arrastre con el puntero y para eso se utiliza un vínculo dinámico en el cual se especifica un punto de entrada.



Luego se hace uso de una clase catalogo para poder realizar el arrastre según parámetros específicos luego pasamos al handler de `Btn_min` en su propiedad click en el cual se hace uso de la propiedad `WindowState` la cual nos permite modificar como se muestra la ventana del form en este caso `Minimized`.



Luego pasamos a darle funcionalidad al Btn_Close para esto hacemos referencia al mismo form con la palabra This en su propiedad Close con la cual se cierra la ventana.

```
1 referencia
36 private void btnClose_Click(object sender, EventArgs e)
37 {
38     //Depuracion
39     // Application.Exit();
40     this.Close();
41 }
42
```

Como siguiente nos basaremos en btnSubir en el cual se obtiene una imagen por medio del metodo showDialog y por medio de un if determinamos que atraves de la propiedad FileName el nombre del archivo debe ser diferente de vacio y luego se muestra la imagen por pictureBox gracias a la propiedad Image.

```
1 referencia
43 private void btnSubir_Click(object sender, EventArgs e)
44 {
45     Subir.ShowDialog(); //Se obtiene la Imagen
46     if(Subir.FileName != "") {
47         PreviewImg.Image = Image.FromFile(Subir.FileName);
48     }
49 }
50
51
```

Con respecto a btnOk se declaran 3 variables locales las cuales se encargan de determinar valores específicos y por medio del if poder decirle al sistema que si la condición se cumple se cree un Image apartir de la propiedad FromFile según el archivo especificado se usa una instancia de memoria y se guarda la imagen según el formato especificado con la propiedad ImageFormat.

```
52 private void btnOk_Click(object sender, EventArgs e)
53 {
54     int Valid = 1;
55     byte[] ImgBytes = null;
56     if (Subir.FileName != "")
57     {
58         Image image = Image.FromFile(Subir.FileName);
59         MemoryStream Memory = new MemoryStream();
60         image.Save(Memory, ImageFormat.Jpeg);
61         ImgBytes = Memory.ToArray();
62     }
63     else
64     {
65         Valid = 0;
66     }
```

Lo siguiente que haremos será declarar una variable local llamada nombre y por medio de un if indicamos que no puede estar vacía y si esto no se cumple mostramos un mensaje en el cual decimos que nombre no puede estar vacío.

Lo siguiente es la categoría para esto declaramos una variable de tipo string que debe ser igual a vacío y con un if indicamos que en comboCategoria en su propiedad selecItem debe ser diferente de null osea que no se escoja nada en el control si esto se cumple devuelve una cadena de texto de lo contrario se muestra un mensaje diciendo categoría no especificada

Para el precio del producto se declara una variable de tipo decimal llamada precio en la cual se obtiene el valor que tenga el control por medio de su propiedad value y luego en un if se determina que el precio no de ser igual a cero si no se cumple se muestra un mensaje diciendo el producto no puede costar nada.

Para el proceso de disponibilidad se declara una variable local de tipo entero llamada disponible con la cual se hara una conversión por medio de Parse para poder capturar el valor del control utilizado y decimos con un if que disponible debe ser igual a cero para poder mostrar un mensaje que dira disponible no puede ser igual a cero.

```
66 }
67 string Nombre = txtNombre.Text;
68 if(Nombre == "")
69 {
70     Valid = 0;
71     MessageBox.Show("Nombre no puede estar Vacio");
72 }
73 string Categoria = "";
74 if (comboCategoria.SelectedItem != null)
75 {
76     Categoria = comboCategoria.SelectedItem.ToString();
77 }
78 else
79 {
80     Valid = 0;
81     MessageBox.Show("Categoria no seleccionada");
82 }
83 decimal Precio = NumPrecio.Value;
84 if(Precio == 0)
85 {
86     Valid = 0;
87     MessageBox.Show("No puede costar nada");
```

```
83 decimal Precio = NumPrecio.Value;
84 if(Precio == 0)
85 {
86     Valid = 0;
87     MessageBox.Show("No puede costar nada");
88 }
89 int Disponible =int.Parse(NumDis.Value.ToString());
90 if(Disponible == 0)
91 {
92     Valid = 0;
93     MessageBox.Show("Disponible no puede ser 0");
94 }
95 if(Valid == 1)
96 {
97     Producto n = new Producto();
```

Si todos los procesos se llevan con normalidad entonces procedemos a crear un objeto que contenga todo lo que el administrador de la tienda especifico luego de esto ponemos un if que utilizemos como respuesta a todo en nuestro caso answer si esta variable da como resultado true se le muestra al administrador producto agregado de lo contrario lo sentimos hubo un error.

```

97     Producto p = new Producto();
98     p.Nombre = Nombre;
99     p.Categoria = Categoria;
100    p.Precio = Precio;
101    p.Disponible = Disponible;
102    p.Imagen = ImgBytes;
103    bool answer = catalogo.AñadirProducto(p);
104    if(answer == true)
105    {
106        MessageBox.Show("Producto Agregado");
107        this.Close();
108    }
109    else
110    {
111        MessageBox.Show("Lo Sentimos hubo un error");
112    }
113
114 }

```

CatalogoForm

En el funcionamiento del diseño del catálogo se instancia un objeto llamado inventario además de este se crea una lista que contiene objetos, primeramente, se utiliza un método **CatalogoForm** para poder utilizar el control Editar en su propiedad click se crea una variable local de tipo string en la cual se obtiene el valor para poder mostrar el valor y decirle al administrador click desde: mas el valor obtenido.

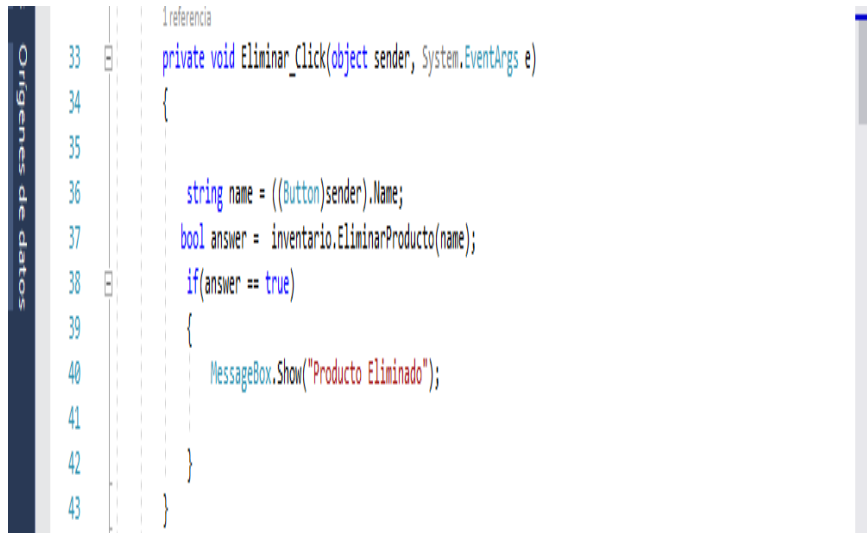
```

13 1 referencia
14 public Image byteArrayToImage(byte[] byteArrayIn)
15 {
16     MemoryStream ms = new MemoryStream(byteArrayIn);
17     Image returnImage = Image.FromStream(ms);
18     return returnImage;
19 }
20 Catalogo inventario = new Catalogo(); //Instancia Objeto
21 List<Producto> Lista = new List<Producto>(); //Lista con los Objetos
22
23 2 referencias
24 public CatalogoForm()
25 {
26     Inicio();
27 }
28
29 1 referencia
30 private void Editar_Click(object sender, System.EventArgs e)
31 {
32     string name = ((Button)sender).Name;
33     MessageBox.Show("Click desde : " + name );
34 }

```

En el botón eliminar lo que haremos será declarar una variable llamada name para luego capturar ese valor declaramos otra variable de tipo bool de nombre answer en la cual

accedemos al metodo Todelete con los parámetros de la variable name. Luego en un if si el valor de la variable answer es true entonces le mostramos un mensaje al administrador que diga producto eliminado



```
1 referencia
33 private void Eliminar_Click(object sender, System.EventArgs e)
34 {
35
36     string name = ((Button)sender).Name;
37     bool answer = inventario.EliminarProducto(name);
38     if(answer == true)
39     {
40         MessageBox.Show("Producto Eliminado");
41     }
42 }
43 }
```