Trabajo Practico 4 Final – Laboratorio II

Nombre: Anzor, Maia María Luz - División n° 2

NOODLES

Delivery de Comida Asiática

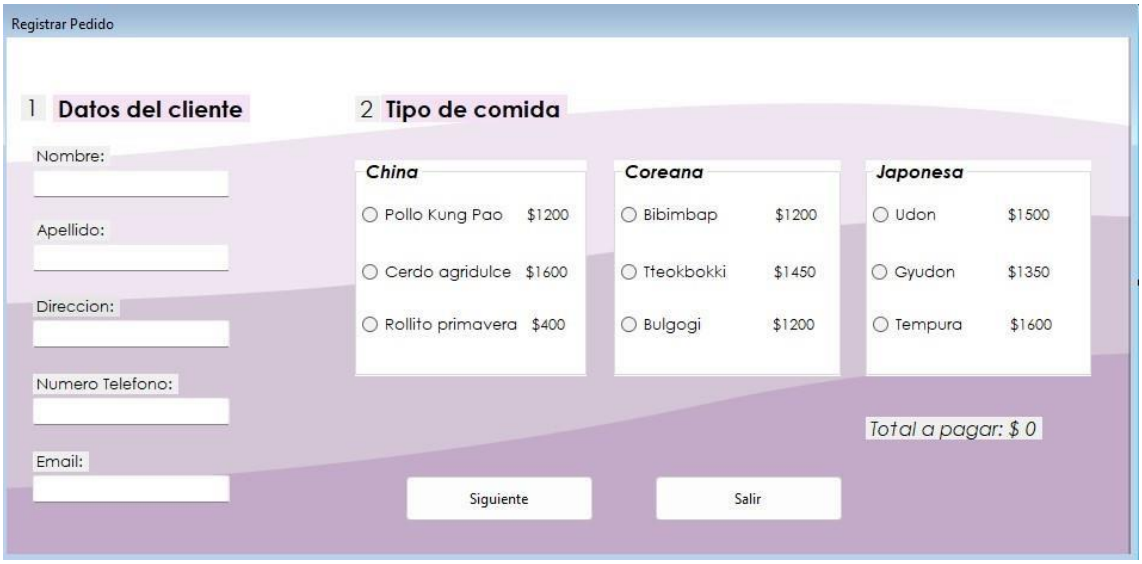
Introducción de la aplicación

Este programa representa la simulación de una aplicación con el fin de gestionar la venta y el envío de comida asiática.

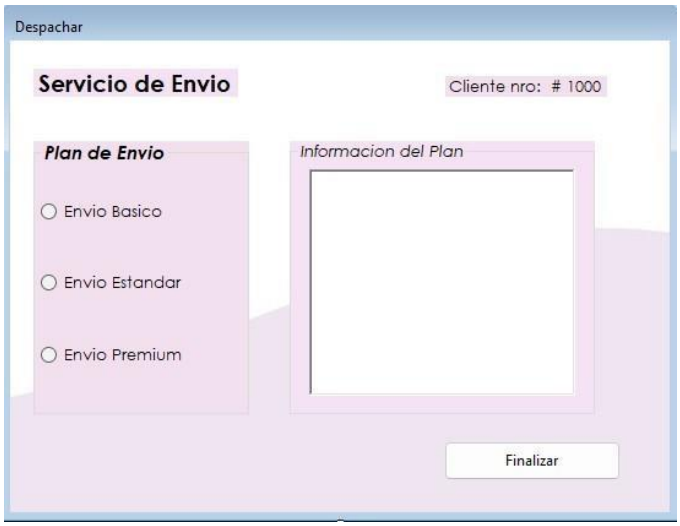
Al iniciarse el programa se podrá observar su menú, el cual posee las opciones de realizar un pedido, o realizar algún cambio sobre los pedidos ya realizados anteriormente.

Al realizar un pedido, esta opción llevara al ingreso de datos del cliente, donde podrá elegir el menú de comida que desee.

(Posee lista con datos precargada, la misma se podrá ver al ingresar a las “opciones de pedidos “esta misma se encuentra en: tps_laboratorio_ii\TP3\Anzor.Maia.2A.TP3\TP3\bin\Debug\net5.0-windows)



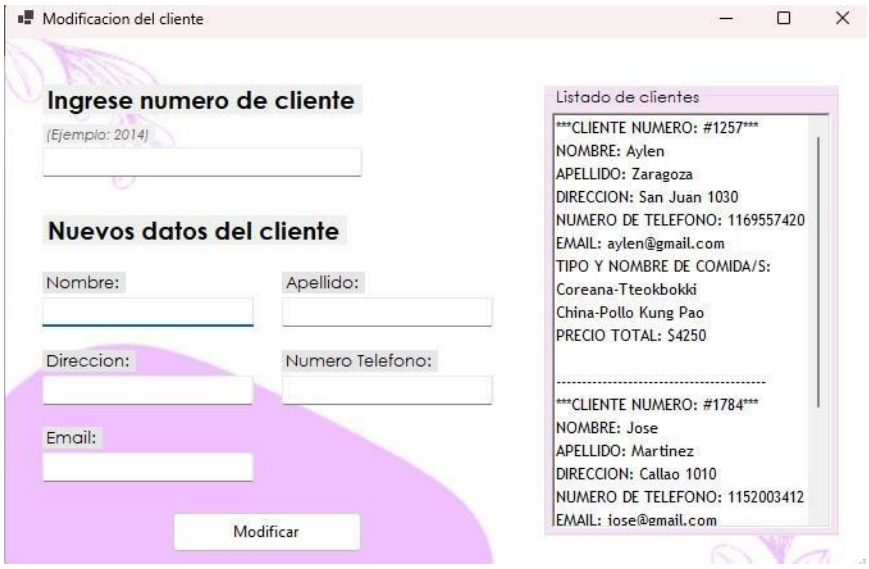
Finalizando el ingreso de datos, a continuación, se podrá elegir el tipo de envió que desee el usuario; donde también se podrá ver el detalle de cada tipo de envío.



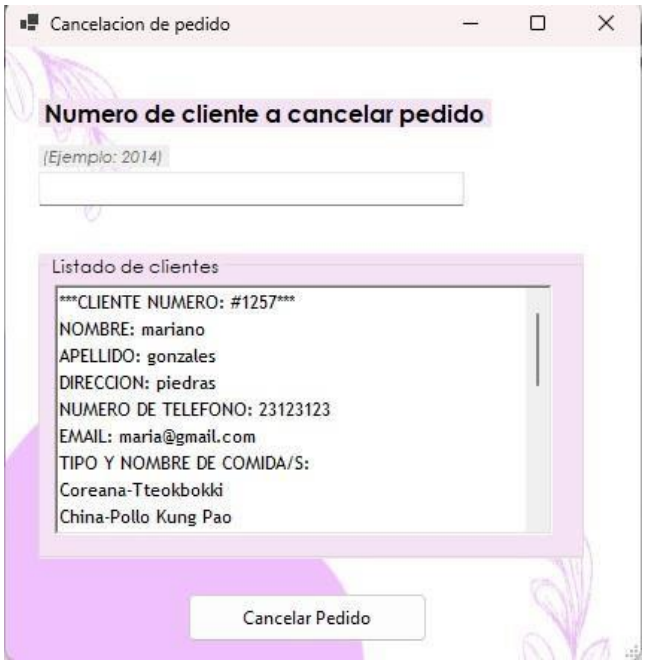
elegidos por el usuario, se mostrará nueva ventana donde se podrá ver la facturación total realizada hasta el momento.

Al finalizar el ingreso de datos En las opciones de pedido es donde se podrá modificar un pedido que de la lista o cancelar; Donde se podrá hacer también un backup de los datos con tres tipos de extensión distintos.

Al intentar modificar algún pedido de la lista, se debe ingresar el numero de cliente a cambiar con los nuevos datos para actualizar.



En la opción de cancelado, se abrirá una nueva ventana donde se cancelara un pedido según el número de cliente.



Las opciones de guardado se podrán realizar con tres tipos de extensiones: .json - .txt - .xml.

(Los archivos guardados se encuentra en la carpeta:

tps_laboratorio_ii\TP3\Anzor.Maia.2A.TP3\TP3\bin\Debug\net5.0-windows)

<u>Temas utilizados en este programa:</u>

- Archivos
- Serialización

- Tipos Genéricos
- Interfaces
- Pruebas unitarias
- Excepciones

Archivos:

Se usa este tema para hacer un tipo de guardado de datos(.txt).

Implementa también una interfaz, a partir la cual brindara los métodos para realizar la lectura de guardado de datos en archivos txt.

(Ejemplo del evento encargado de escribir en un archivo txt)

```
private void btnBackUp_Click(object sender, EventArgs e)
{
    //Escritura de archivo .txt
    string ruta = $"{AppDomain.CurrentDomain.BaseDirectory}" + @"BackupListaPedidos.txt";

    if (this.unRegistro.HacerBackUp(this.Contenido, ruta))
    {
        MessageBox.Show("Backup con extension TXT realizado con exito", "Backup finalizado");
    }
}
```

Serialización:

Hará la lectura y escritura con extensión Json y Xml al momento de elegir el tipo de guardado deseado.

Estos métodos se encuentran en la clase encargada de esto.

```
//seralizacion de archivo json
1 referencia
public bool SerializarJson(string ruta, List<Cliente> lista)
{
    bool retorno = true;
    string serializado = String.Empty;
    try
    {
        using (StreamWriter sw = new StreamWriter(ruta))
        {
            serializado = JsonSerializer.Serialize(lista);
            sw.WriteLine(serializado);
            retorno = true;
        }

    }
    catch (ExceptionControllers e)
    {

        throw new ExceptionControllers("Excepcion controlada...", e);
    }
    return retorno;
}
```

```
//Serializacion de archivo xml
4 referencias
public bool SerializarXml(string ruta, List<Cliente> lista)
{
    bool retorno = true;
    try
    {
        using (StreamWriter sw = new StreamWriter(ruta))
        {
            XmlSerializer xmlSerializador = new XmlSerializer(typeof(List<Cliente>));

            xmlSerializador.Serialize(sw, lista);
        }
        retorno = true;
    }
    catch (ExceptionControllers e)
    {
        throw new ExceptionControllers("Excepcion controlada...", e);
    }
    return retorno;
}
```

Tipos Genéricos e Interfaces:

Aplicado en la interfaz, la cual brinda los métodos para leer y escribir archivos de extensión txt.

Este mismo se implementa en la clase.

```
public class Registro : IBackUp<string>
{
    private List<Cliente> listaClientes;
    private Cliente cliente;
    private string path = null;

    3 referencias | 1/2 pasando
```

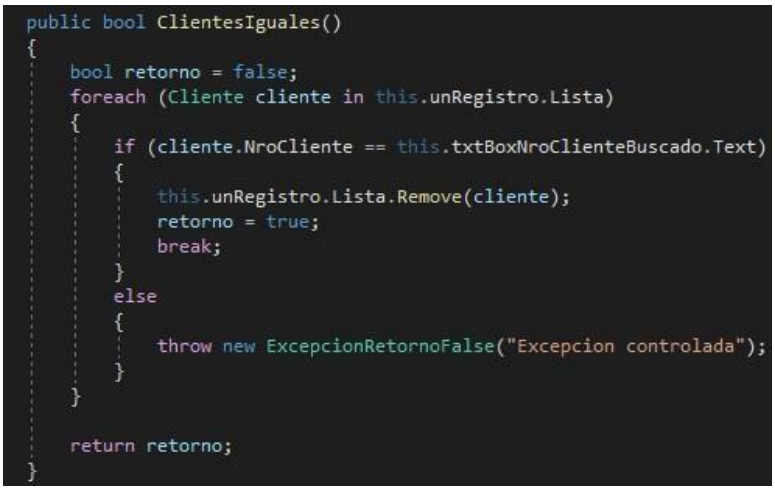
```
public interface IBackUp<T>
{
    1 referencia
    string Ruta { get; set; }

    2 referencias
    bool HacerBackUp(string contenido, string ruta);
    1 referencia
    T LeerArchivo(string ruta);
}
```

Excepciones:

Aplicado en el proyecto para captura posibles errores que puedan ocurrir en el programa.

Las excepciones mostraran un mensaje al aparecer.

Ejemplo de una excepción aplicada en caso de haber error.



```
public bool ClientesIguales()
{
    bool retorno = false;
    foreach (Cliente cliente in this.unRegistro.Lista)
    {
        if (cliente.NroCliente == this.txtBoxNroClienteBuscado.Text)
        {
            this.unRegistro.Lista.Remove(cliente);
            retorno = true;
            break;
        }
        else
        {
            throw new ExcepcionRetornoFalse("Excepcion controlada");
        }
    }

    return retorno;
}
```

Temas aplicados TP4 Final:

- Delegados y expresiones Lambda
- Hilos
- Método de extensión
- Eventos
- Conexión a base de datos

Eventos:

Aplicado en el proyecto para el cambio de mensaje por cada cliente, mostrando así la información de pedido de este mismo.

```
1 referencia
private void btnFinalizar_Click(object sender, EventArgs e)
{
    frmCargadnoComida f = new frmCargadnoComida(unRegistro);

    this.OnAvisarPedidoHecho += Mensaje;
    this.OnAvisarPedidoHecho.Invoke();


    Task t = Task.Run(f.ShowDialog);


    this.Hide();
    this.Close();

}
```

```
public void Mensaje()
{
    ServicioEnvio servicioEnvio = new ServicioEnvio(this.PlanEnvioElegido);
    frmAlta formAlta = new frmAlta(unRegistro);

    c = this.unRegistro.Lista.Last();
    MessageBox.Show($"{c.Mostrar() + servicioEnvio.Mostrar()}", "INFORMACION COMPLETA DE FACTURACION", MessageBoxButtons.OK);

}
```

Conexión a base de datos:

Sera el encargado de leer los repartidores existentes en la lista y agregar un nuevo repartidor a dicha lista

Este mismo se podrá ver accediendo a la información de empleados

Agregar repartidores permitirá ingresar nuevos repartidores a la lista, y serán agregados a la base de datos

```
public static void AgregarRepartidor(this Repartidores repartidor)
{
    string query = "INSERT INTO Repartidores (NOMBRE, APELLIDO, SEXO, NUMERO_DE_CONTACTO, ESTADO_REPARTIDOR

    SqlConnection connection = null;

    try
    {
        connection = new SqlConnection(GestorSQL.cadenaConexion);
        connection.Open();
        SqlCommand cmd = new SqlCommand(query, connection);

        cmd.Parameters.AddWithValue("NOMBRE", repartidor.Nombre);
        cmd.Parameters.AddWithValue("APELLIDO", repartidor.Apellido);
        cmd.Parameters.AddWithValue("SEXO", repartidor.Sexo);
        cmd.Parameters.AddWithValue("NUMERO_DE_CONTACTO", repartidor.NumeroContacto);
        cmd.Parameters.AddWithValue("ESTADO_REPARTIDOR", repartidor.EstadoEmplado);
        cmd.ExecuteNonQuery();
    }
```

string Repartidores.EstadoEn


```
Repartidores r = new Repartidores(this.txtBoxNuevoNombreRepar.Text, this.txtBoxNuevoApellidoRepar.Text,
r.AgregarRepartidor();
```

<u>Delegados y expresiones lambda:</u>

Permitirá la actualización de mensaje dependiendo de para que sea el uso. Este será llamado en distintos lugares y cambiará en cada caso.

```
public delegate void DelegadoAviso(string mensaje, string tipoMensaje);
DelegadoAviso delegadoAviso = Avisar;
```

```
1 referencia
static void Avisar(string mensaje, string tipoMensaje)
{
    MessageBox.Show($"{mensaje}", $"{tipoMensaje}", MessageBoxButtons.OK);
}
```

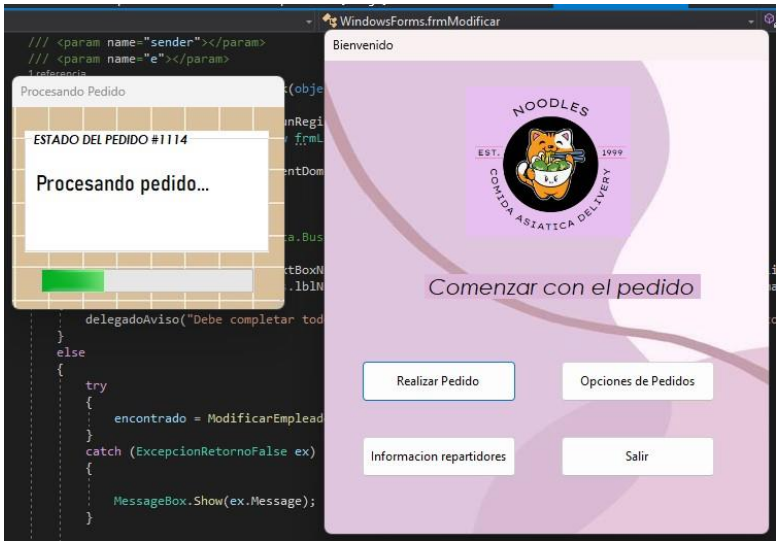
```
delegadoAviso("Debe completar todos los datos con la informacion del cliente", "Datos incompletos");
```

Expresiones Lambda utilizado al llamar a la función que hará el proceso de hacer dinámico el mensaje y cargar la barra progresiva

```
Task t = Task.Run(() =>
{
    FuncionArranque();

});
```

Hilos:

Este fue aplicado en el proyecto para lograr así mantener otra ventana al mismo tiempo del cualquier otro fomr; en donde se estará procesando cada pedido junto con una barra de carga simulando el proceso de dicho pedido, al finalizar la preparación del pedido este se cierra.
(Nota: los formularios nuevos se abren uno encima de otro ya que no logre encontrar una manera de que no se superpongan)



Métodos de extensión:

Aplicado en el proyecto en el método que agregara un nuevo empleado.

```
/// <param name="repartidor"></param>
1 referencia
public static void AgregarRepartidor(this Repartidores repartidor)
{
    string query = "INSERT INTO Repartidores (NOMBRE, APELLIDO, SEXO, NU

    SqlConnection connection = null;

    try
```

```
Repartidores r = new Repartidores(this.txtBoxNuevoNombreRepar.Text, this.txtBoxNuevoApellidoRepar.Text,
r.AgregarRepartidor();
```