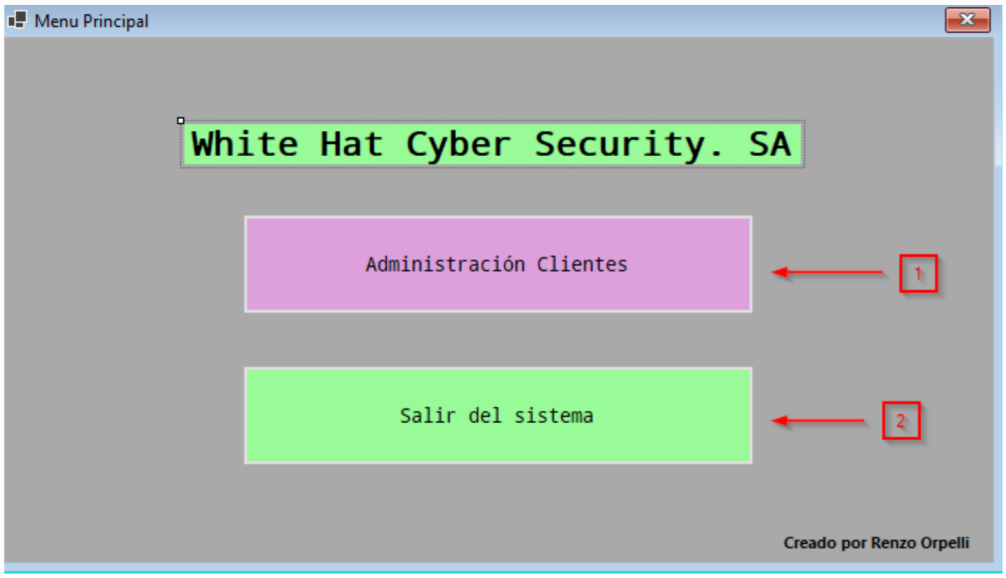
TRABAJO PRÁCTICO 3 - LABORATORIO II
RENZO ORPELLI - DIV 2 E.
EMPRESA DE CIBERSEGURIDAD.

PARTE I:

Explicación del programa:
El programa se basa en un panel de administración de una empresa de ciberseguridad en el que se podrán agregar clientes(cantidad máxima 50), modificar algunos de sus datos y también eliminarlos. Cada cliente tendrá una lista de servicios realizados y la posibilidad de imprimirles una factura detallando qué servicios le fueron realizados.

Funcionamiento del programa:

Al iniciar la aplicación, nos encontraremos el formulario principal, donde
1) Nos llevará al panel de Administración de los clientes.
2) Nos sacará del programa.



Entrando en el panel "Administracion Clientes" nos encontraremos con distintas botones los cuales representan distintas acciones a realizar:

1)Alta Cliente:
Se mostrará por pantalla un nuevo formulario el cual permitirá cargar los datos de un nuevo cliente.

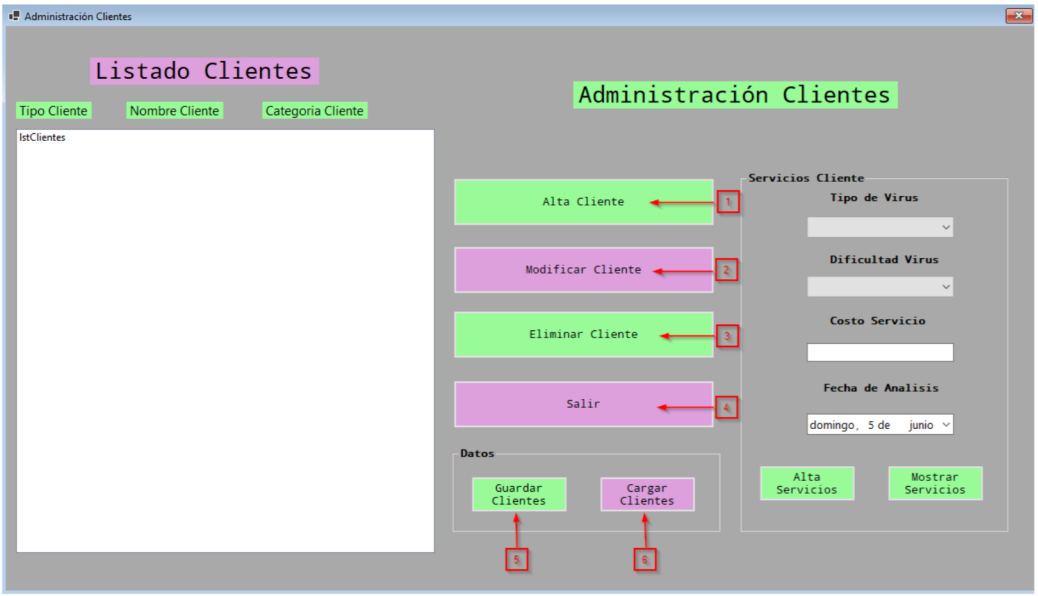
2)Modificar Cliente:
Nos mostrará por pantalla un nuevo formulario con los datos del cliente seleccionado previamente en la listbox "Listado Clientes" y nos permitirá modificar algunos datos del mismo.

3)Eliminar Cliente:
Removerá el cliente seleccionado de la listbox "Listado Clientes".

4) Salir:
Dará la oportunidad al usuario de salir del formulario.

5)Guardar Clientes:
Tomará el estado actual de la lista y en caso de que esta no esté vacía, se la guardará en un archivo de tipo ".xml", mostrándole al usuario por un mensaje donde estará guardado el archivo.

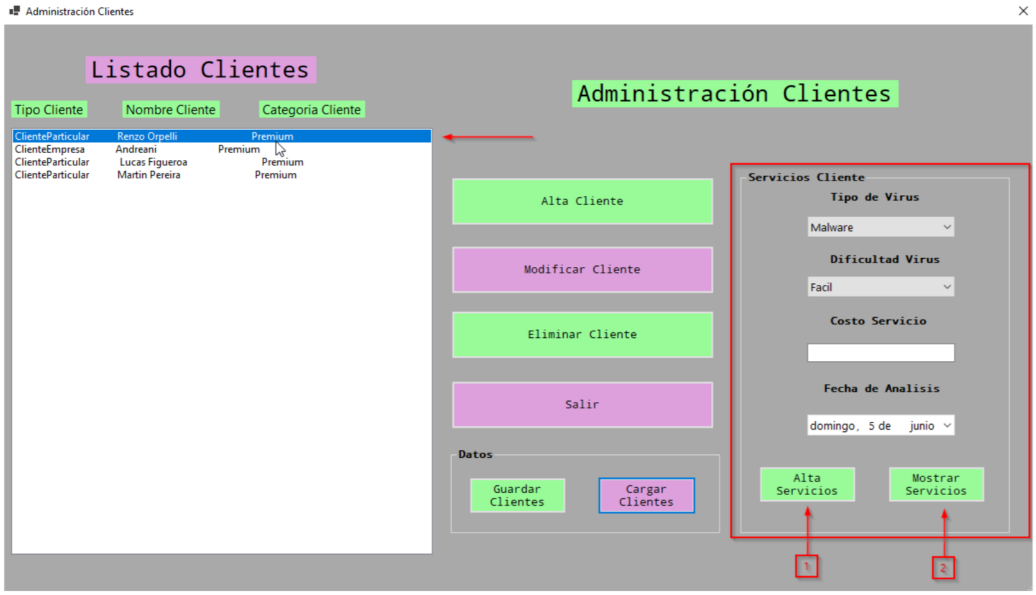
6)Cargar Clientes:
Cargará una lista de clientes en caso de que exista en la ruta especificada por el Gestor de Archivos.
(Abajo la imagen correspondiente).



Sobre los servicios, para agregar un servicio a un cliente, se tendrá que seleccionar un cliente del listbox "Listado de clientes", para cargar un servicio. En el grupbox "Servicios Cliente" se tendrán que completar con todos los atributos del servicio, los cuales son el tipo de virus, la dificultad del virus, el costo del servicio y la fecha del análisis. Las acciones a realizar serán:
1)Alta servicios:
Dará de alta el servicio para el cliente previamente seleccionado y se lo agregara a su lista.

2) Mostrar Servicios:
Abrirá un formulario mostrando todos los servicios que tiene dicho cliente y le dará la oportunidad de remover un servicio de la lista o también emitir una factura del cliente.
Dentro de este form, si se quisiera, se podría eliminar un servicio seleccionado.

(Abajo la imagen correspondiente).



PARTE II :

Implementaciones de temas utilizados para la realización del trabajo (Clase 10-15):

Excepciones:

Este tema fue aplicado varias veces en el proyecto con el fin de manejar y controlar posibles errores, un ejemplo de una implementación seria :

```
/// <summary>
/// Valida la cantidad de clientes de la lista no sea mayor a la cantidad de clientes permitidos
/// </summary>
/// <returns>true si no fue alcanzada, de lo contrario lanzara una excepcion de tipo CantidadDeClientesAlcanzadaException</returns>
1 referencia
private bool ValidarCantidadClientes()
{
    if (this.ListaClientes.Count == this.cantidadClientesMaxima)
    {
        throw new CantidadDeClientesAlcanzadaException("Cantidad de clientes máxima alcanzada");
    }
    return true;
}
```

```
/// <summary>
/// crea una excepcion con un mensaje
/// </summary>
/// <param name="message">mensaje de la excepcion</param>
1 referencia
public CantidadDeClientesAlcanzadaException(string message) : base(message)
{
}


/// <summary>
/// crea una expcecion con un emnsaje y su innerException
/// </summary>
/// <param name="message">el mensaje de la excepcion</param>
/// <param name="innerException">la innerException de la excepcion</param>
0 referencias
public CantidadDeClientesAlcanzadaException(string message, Exception innerException) : base(message, innerException)
{
}
```

Test Unitarios:

Este tema fue implementado con el fin de verificar la funcionalidad de ciertos métodos de la aplicación, un ejemplo de su aplicación de un test unitario sería:

```
[TestMethod]
0 referencias
public void AlAgregarUnServicioAUnClienteConCategoriaPremium_DeberiaAplicarleUnDescuentoAlServicio()
{
    #region Arrange
    Cliente empresa = new ClienteEmpresa(Cliente.categoriaCliente.Premium, "Andrean1", "123123123123", "lobos 123");
    Servicio servico = new Servicio(20000, Servicio.dificultadVirus.Dificil, Servicio.tipoVirus.Adware, new System.DateTime(2020, 03, 23));
    double result = 0;
    double expected = 18000;
    #endregion

    #region Act
    if (empresa + servico)
    {
        result = servico.CostoServicio;
    }

    #endregion
    #region Assert

    Assert.AreEqual(expected, result);

    #endregion
}
```

Tipos Genericos:

Este tema fue implementado en la clase “Serializador” y en la interfaz “IArchivo” para adaptar el método “Escribir” y “Validar Extensión” y tener la misma funcionalidad aunque maneje distintos tipo de datos en las clases que se los implementa.

```
/// <summary>
/// metodo encargado de leer un archivo de tipo xml, deserializandolo y devolviendolo el objeto T
/// </summary>
/// <param name="nombreArchivo">string con el nombre del archivo</param>
/// <returns>returna el objeto deserializado, caso contrario lanzara excepcioones del tipo ArchivoNullException</returns>
/// <exception cref="ArchivoNullException"></exception>
1 referencia
public T Leer(string nombreArchivo)
{
    try
    {
        if (tipo == ETipo.ClienteXML)
        {
            if (ValidarExtension(nombreArchivo) && ExisteArchivo($"{rutaBase}\\{nombreArchivo}"))
            {
                using (XmlTextReader xmlTextReader = new XmlTextReader($"{rutaBase}\\{nombreArchivo}"))
                {
                    XmlSerializer serializer = new XmlSerializer(typeof(T));
                    return serializer.Deserialize(xmlTextReader) as T;
                }
            }
        }
    }
    catch (ArchivoNullException ex)
    {
        throw ex;
    }
    catch (Exception ex)
    {
        throw new ArchivoNullException("Error al leer el archivo xml", ex);
    }
    return null;
}
```

```
public interface IArchivo<T>
{
    /// <summary>
    /// Guarda un archivo del tipo Objeto Generico en la ruta indicada
    /// </summary>
    /// <param name="nombreArchivo">el nombre del archivo</param>
    /// <param name="contenido">el contenido del objeto generico</param>
    /// <returns>un string que contiene el resultado de la operacion</returns>
    6 referencias
    string Escribir(string nombreArchivo, T contenido);



    /// <summary>
    /// valida la extension de un archivo
    /// </summary>
    /// <param name="nombreArchivo">nombre del archivo</param>
    /// <returns>true si es correcta, de lo contrario una excepcion de tipo ArchivoNullException</returns>
    6 referencias | 1/1 pasando
    bool ValidarExtension(string nombreArchivo);
```

Interfaces:

Este tema fue implementado con la finalidad de agrupar métodos y que estos puedan ser utilizados en otras clases (Un ejemplo de implementación de las interfaces utilizadas se puede ver en la imagen de arriba).

Archivos:

Este tema fue implementado con el fin de emitir la factura de cada cliente en formato .txt, con todos sus datos, todos los servicios, y el total a pagar del mismo.

```
/// <summary>
/// Recibe el nombre del archivo y el contenido a escribir del mismo de tipo string y lo guarda en un archivo de tipo txt
/// </summary>
/// <param name="nombreArchivo">el nombre del archivo</param>
/// <param name="contenido">el contenido del archivo en tipo string</param>
/// <returns>devuelve un mensaje con el resultado de la operacion</returns>
/// <exception cref="ArchivoNullException"></exception>
3 referencias
public string Escribir(string nombreArchivo, string contenido)
{
    string puedoLeer = "Error al escribir el archivo de texto";
    try
    {
        if (ValidarExtension(nombreArchivo))
        {
            using (StreamWriter streamWriter = new StreamWriter($"{base.rutaBase}\\{nombreArchivo}"))
            {
                streamWriter.WriteLine(contenido);
                puedoLeer = $"Archivo de Text Escrito con exito en {rutaBase}\\{nombreArchivo}";
            }
        }

    }
    catch (Exception ex)
    {
        throw new ArchivoNullException("Error al escribir archivo de texto", ex);
    }
    return puedoLeer;


}
```

Serialización:

Este tema fue implementado en la clase “Serializador” del tipo genérica la cual también utiliza la misma interfaz “IArchivo” del mismo tipo, para serializar o deserializar la lista de clientes.

```
    /// <summary>
    /// metodo encargado de serializar la lista de clientes, recibe el nombre archivo y contenido de tipo generico
    /// lo serializara y guardara en la ruta especificada en la clase GestorDeArchivo
    /// </summary>
    /// <param name="nombreArchivo">string con el nombre del archivo</param>
    /// <param name="elemento"></param>
    /// <returns>un string con la ruta del archivo guardado o de caso contrario lanzara excepciones del tipo ArchivoNullException</returns>
    /// <exception cref="ArchivoNullException"></exception>
    3 referencias
    public string Escribir(string nombreArchivo, T elemento)
    {
        string resultado = "Error al Guardar";

        try
        {
            if (tipo == ETipo.ClienteXML)
            {
                if (ValidarExtension(nombreArchivo))
                {
                    using (XmlTextWriter xmlTextWriter = new XmlTextWriter($"{rutaBase}\\{nombreArchivo}", Encoding.UTF8))
                    {
                        xmlTextWriter.Formatting = Formatting.Indented;
                        XmlSerializer serializer = new XmlSerializer(typeof(T));
                        serializer.Serialize(xmlTextWriter, elemento);
                        resultado = $"Archivo Guardado Existosamente en {this.rutaBase}{nombreArchivo}";
                    }
                }

            }
        }
        catch (ArchivoNullException ex)
        {
            resultado = ex.Message;
        }
        catch (Exception ex)
        {
            throw new ArchivoNullException("Error al escribir el archivo xml", ex);
        }
        return resultado;
    }
```