UNIVERSIDAD DE QUINTANA ROO

División de Ciencias e Ingeniería

Desarrollo de aplicación cliente/servidor para administración de restaurantes

Trabajo de Tesis

para obtener el grado de:

Ingeniero en Redes

PRESENTA

Mauricio Alejandro Colunga Velazco

Directora de Tesis

MSI Melissa Blanqueto Estrada

Asesores

MTI Vladimir Veniamin Cabañas Victoria

MSI Laura Yésica Dávalos Castilla

Chetumal, Quintana Roo, México, Septiembre de 2011.

UNIVERSIDAD DE QUINTANA ROO

División de Ciencias e Ingeniería

Trabajo de Tésis elaborado bajo supervisión del Comité de Asesoría y aprobada como requisito parcial para obtener el grado de:

INGENIERO EN REDES

Comité de Trabajo Monográfico o Tesis

Directora:

MTI Melissa Blanqueto Estrada

Asesor:

MTI Vladimir Veniamin Cabañas Victoria

Asesora:

MSI Laura Yésica Dávalos Castilla

Chetumal, Quintana Roo, México, Julio de 2011.

Agradecimientos

Agradezco a la Universidad de Quintana Roo por haberme brindado la oportunidad de crecer profesionalmente.

A mis padres Juan Margarito Colunga Gil y Rosa María del Socorro Velazco Cachón por apoyarme en esta etapa de mi vida.

A los maestros que me brindaron parte de sus conocimientos durante toda la carrera y que sin ellos esto no habría sido posible. En especial a mi tutor el M. C. Javier Vázquez Castillo quien le dio seguimiento a mi carrera, a la MTI Melissa Blanqueto Estrada quien dirigió este trabajo de titulación, así como mis asesores el MTI Vladimir Veniamin Cabañas Victoria y la MSI Laura Yésica Dávalos Castilla. De igual manera al Ing. Rubén Enrique González Elixavide, coordinador de la carrera quien me brindó su confianza y me dio su apoyo durante todo este tiempo.

Este trabajo fue financiado en la Convocatoria 2011 “Apoyo a la titulación”, de la División de Ciencias e Ingeniería.

Dedicatoria

Dedicado mi padre Juan Margarito Colunga Gil por confiar en mí y quien nunca se separó de mi lado, apoyándome durante todo este tiempo.

A mi esposa Cristina Comi Ramírez, mis hijos Abdiel Alejandro Colunga Comi y Christopher Leobardo Colunga Comi quienes son mi mayor motivo de superación para brindarles un mejor futuro.

Resumen

El presente trabajo documenta las fases de desarrollo de un software para un restaurante, observando las etapas del ciclo de vida del producto (PLC).

En la etapa de análisis se realizó la recolección de información necesaria para el desarrollo del software, comenzando con un listado de los problemas y sus posibles soluciones. Durante la etapa de diseño se realizó un bosquejo del software comenzando por el diseño de interfaces y la descripción de su operación hasta ser aprobado.

En la siguiente etapa se llevó a cabo el desarrollo de lo acordado en la etapa de diseño y al termino se realizaron las pruebas necesarias para minimizar los problemas al momento de la implementación. Durante la implementación se monitoreo diariamente por un lapso de una semana para registrar los fallos que presentó el sistema y después se realizó el mantenimiento del sistema, donde se corrigieron los errores detectados y se realizaron las modificaciones solicitadas. La etapa de implementación y mantenimiento deben ser llevadas con mucho cuidado ya que permitieron obtener el software sin errores para obtener el producto final.

Para el desarrollo del software se utilizó Visual Basic 2010 y fue programado con el lenguaje C# (C Sharp), también se implementó la herramienta Fluent NHibernate que permitió el mapeo de clases reduciendo la cantidad de código y tiempo para la programación del software.

Con la correcta implementación del PLC se redujeron los errores durante la elaboración del software ahorrando tiempo y recursos.

Contenido

Tabla de figuras

Capítulo 1. Introducción

Antecedentes

Actualmente las aplicaciones de software representan una oportunidad de rápido crecimiento en diversos tipos de empresas, estas aplicaciones proporcionan herramientas y técnicas para minimizar pérdidas y maximizar ganancias. Hoy en día existe una gran cantidad de sistemas estándar para una gran variedad de empresas y organizaciones, sin embargo no todas ellas se administran de la misma manera.

Las aplicaciones estándar (comerciales o no) no siempre son la mejor opción para algunas empresas; algunas de las razones se deben al hecho de que hace falta habilidades tecnologícas por parte de los usuarios, en muchos de los casos, los sistemas no se pueden “adaptar” al modo de operación de la empresa y también porque las implementaciones de tecnología son percibidas como de alto costo y apenas un medio beneficio.

Las tecnologías de información son importantes para las micro, pequeñas y medianas empresas por tres razones principales:

- Ha demostrado que puede dar valor agregado a los bienes y servicios de una organización, permite transformarlos o mejorar la coordinación de las actividades relacionadas con el proceso de generación de éstos. Asimismo, la informática puede ayudar a transformar la manera en que una organización compite, afectando las fuerzas que controlan la competencia en una industria.
- Gracias a ellas algunas organizaciones han podido crear barreras de entrada, reducir la amenaza de productos o servicios sustitutos, cambiar su forma de competir de costos a diferenciación o a especialización, y aumentar su poder de proveedores o de compradores.
- Finalmente, y posiblemente ésta sea la razón más importante, pueden ayudar a reinventar la manera como una organización opera. La mayoría de los procesos de una organización operan de acuerdo a reglas obsoletas y no toman en cuenta las ventajas que proporcionan las tecnologías de información. Las bases de datos compartidas que permiten tener información accesible en diferentes puntos en forma simultánea, el uso de los sistemas expertos para representar y utilizar el conocimiento, y el uso de redes para intercambiar información, son solamente algunas de las nuevas tecnologías que nos permiten rediseñar la manera como operan las organizaciones.

Descripción del problema

El restaurant el “Pargo” localizado a las afueras de la ciudad de Chetumal en el estado de Quintana Roo, tiene como misión brindar un servicio rápido y de calidad siendo el principal negocio que ofrezca una variedad de platillos del mar. De igual manera tiene como visión en los próximos cinco años dar apertura a una sucursal dentro de la ciudad para tener un servicio más accesible a sus clientes.

El principal problema es la engorrosa labor de generar las cuentas y/o facturas de los clientes que atiende a diario, pues cuenta con alrededor de treinta mesas y en las horas pico más de un cliente pide su cuenta en mismo tiempo, por lo que tiene que tomar todas las comandas de la mesa que solicita la cuenta, calcular manualmente el precio de cada artículo pedido y sumar el total, lo que ocasiona un retraso generar las cuentas y/o facturas causando molestia a los clientes y en ciertos casos la pérdida de capital ya que los clientes se van sin pagar.

El segundo problema es la compra excesiva de materia prima (camarón, pulpo, caracol, etc.) y la poca ganancia que esto genera. Esto es debido a que no se lleva un control de entradas y salidas, dando lugar a malos manejos de insumos.

La tercera problemática se da cuando un cliente realiza un pedido y el mesero tiene que ir hasta la barra para entregar la orden de bebidas y después a la cocina para entregar la orden de comida, esto demora mucho tiempo y en horas pico hay clientes que desean que sea rápido el servicio e incluso otros que esperan el desalojo de una mesa.

Objetivos

Objetivo general

Mejorar la satisfacción del cliente en el restaurante “El Pargo”, mediante el desarrollo e implementación de una aplicación que permita realizar en forma eficiente el proceso de elaboración de notas y facturas, así como llevar un adecuado control de las entradas y salidas de insumos.

Objetivos específicos

- Análisis del problema.
- Diseño de la solución.
- Codificación de la solución.
- Pruebas de software.
- Implementación de software.

Justificación

El rápido crecimiento de la población en las ciudades ocasiona que las empresas tengan una mayor afluencia de personas obligándolas a agilizar sus operaciones, en particular los restaurantes ya que un servicio eficiente lo ayudará en su crecimiento y al crecer necesitará una mejor administración de todas sus operaciones diarias.

Las nuevas tecnologías y el fácil acceso a ellas como son las computadoras, las redes, los dispositivos, etc. permiten a las empresas implementar soluciones para mejorar su operatividad.

Del mismo modo la existencia de una gran cantidad de software en el mercado y la gran cantidad de empresas que buscan la implementación de tecnologías para un mayor crecimiento abre una ventana para el desarrollo de software que se acople a las empresas que tienen una operativa diferente.

La experiencia de desarrollar una solución totalmente operativa para un negocio de la localidad sienta las bases que me permitirán como ingeniero en redes crear una empresa desarrolladora de software.

Alcances y Restricciones

a) Este trabajo comprende el análisis, diseño y desarrollo del software.

b) Para el desarrollo del software sólo se usarán componentes gratuitos de Visual Studio 2010.

c) El la entrega del sistema debía ser en un lapso máximo de un mes.

d) Las pruebas se deben realizar de lunes a viernes de siete a once de la mañana.

e) Se asesoro al cliente para que comprara el equipo necesario para la implementación del software.

Capítulo 2. Metodología

Introducción

En este capítulo se describen las etapas del ciclo de vida del producto para el desarrollo del software, así como el lenguaje de programación y herramientas empleadas.

Etapas del Ciclo de Vida del Producto (PLC)

Antes de comenzar a elaborar un producto es importante tener en cuenta el ciclo de vida del producto, que representa un proceso por etapas a seguir para desarrollar cualquier nuevo producto. Existen siete etapas definidas para el PLC (Sharch, 2005):

1. Análisis
2. Diseño
3. Desarrollo
4. Pruebas
5. Implementación
6. Mantenimiento
7. Fin de vida

Etapa de Análisis

El análisis es el proceso de investigar un problema a solucionar con un producto. Entre otras tareas, análisis consiste en:

Tener claramente la definición del problema a solucionar, el mercado a cubrir, o el sistema a crear. También saber el límite de un problema, como el alcance del proyecto.

La identificación de los subcomponentes claves que componen un producto total se puede apreciar en la Figura 1.

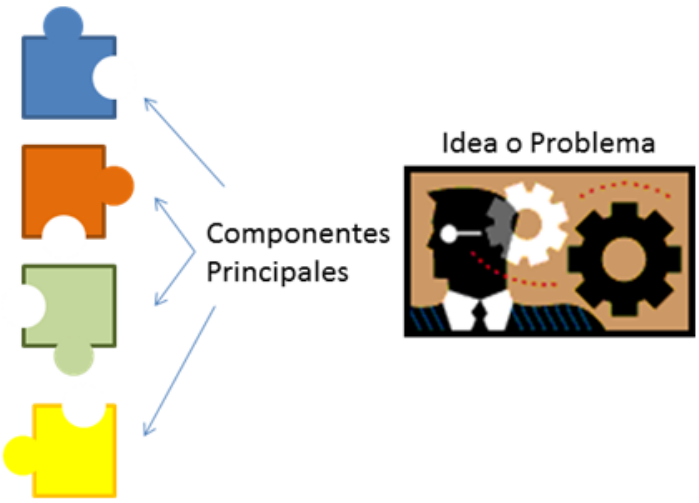


Figura 1. Etapa del análisis del PLC

En esta etapa es muy útil utilizar un cuestionario previamente elaborado con el fin de recopilar la mayor cantidad de información que le sea posible, para que al final de esta etapa se realice la descripción del problema.

Descripción del problema

La descripción del problema es un resumen preliminar de necesidades que sirve como punto de partida para comprender los requisitos del sistema. Aquí se trata de simular una descripción preparada por un cliente, la cual debe evolucionar por medio del modelo de requisitos, con el objeto de lograr la especificación final del sistema a desarrollarse. La descripción del problema debe ser una especificación de necesidades y no una propuesta de solución. La descripción inicial puede ser incompleta e informal, pues al realizarse sin un análisis completo, no hay ninguna razón para esperar que sea correcta.

Etapa de Diseño

El diseño es el proceso de aplicar las conclusiones que hechas durante la etapa de análisis al diseño real de su producto. La tarea primaria durante la etapa de diseño es la de desarrollar cianotipos o datos específicos para los productos o componentes en su sistema (Ver Figura 2).

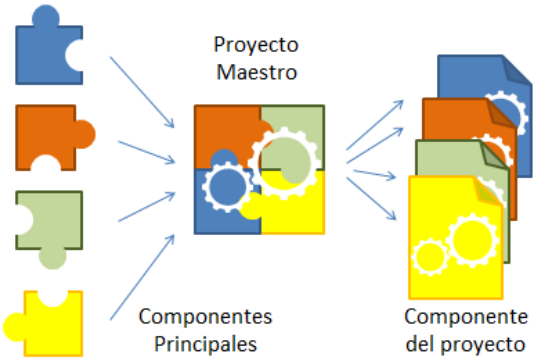


Figura 2. Etapa de Diseño del PLC

Modelo de caso de uso

El modelo de caso de uso describe un sistema en términos de sus distintas formas de utilización, cada una de las cuales se conoce como caso de uso. Cada caso de uso o flujo se compone de una secuencia de eventos iniciada por el usuario; dado que los casos de uso describen el sistema a desarrollarse, los cambios en los requisitos significarán cambios en los casos de uso. Por ejemplo, un caso de uso para manejar un automóvil sería la secuencia de eventos desde que el conductor entra al coche y enciende el motor hasta llegar a su destino final.

Por lo tanto, para comprender los casos de uso de un sistema primero es necesario saber quiénes son sus usuarios; por ejemplo, conducir un automóvil es distinto de arreglarlo, pues los usuarios también son diferentes: el dueño del automóvil y el mecánico, respectivamente. Para ello, se define el concepto de actor, que es el tipo de usuario que está involucrado en la utilización de un sistema, y que además es una entidad externa al propio sistema.

Juntos, el actor y el caso de uso, representan los dos elementos básicos de este modelo (Figura 3).

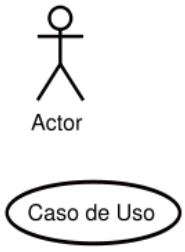


Figura 3. El actor y el caso de uso son las entidades básicas del modelo de caso de uso

Actores

Los actores son entidades distintas a los usuarios, en el sentido de que éstos son las personas reales que utilizarán el sistema, mientras que los actores representan cierta función que una persona real realiza. En la terminología orientada a objetos, se considera al actor una clase de usuario, mientras que los usuarios se consideran como objetos o instancias de esa clase. Incluso, una misma persona puede aparecer como diversas instancias de diferentes actores.

Antes de identificar los casos de uso, se identifican los actores del sistema, esto es para que éstos sean la herramienta principal que permita encontrar los casos de uso. Cada actor ejecuta un número específico de casos de uso en el sistema. Una vez definidos todos los actores y casos de uso en el sistema, se establece la funcionalidad completa de éste.

Para especificar los actores de un sistema, se dibuja un diagrama correspondiente a la **delimitación del sistema** la cual representa al sistema como una "caja negra", y a los diferentes actores, como entidades externas a ésta (Figura 4).

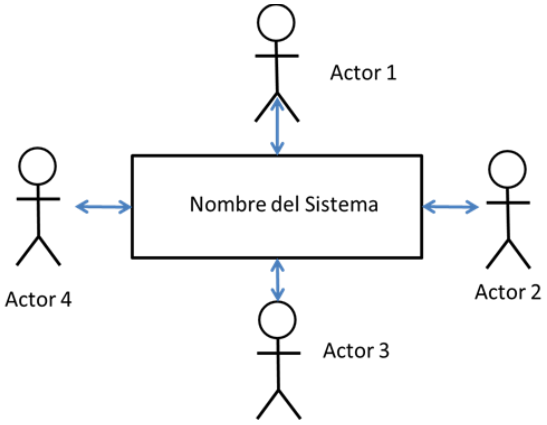


Figura 4. Delimitación de un sistema según los actores

Para reconocer los casos de uso, es necesario identificar primero a los actores del sistema, comenzando por aquellos que son la razón principal del sistema, conocidos como actores primarios. Estos actores típicamente rigen la secuencia lógica de ejecución del sistema.

Además de los actores primarios, existen actores supervisando y dando manteniendo al sistema, a los que se llama actores secundarios y existen primordialmente como complemento a los actores primarios, siendo esta distinción importante para dedicarle el esfuerzo principal a las necesidades de los actores primarios. Al contrario de éstos, que típicamente pertenecen a personas físicas, los actores secundarios corresponden, por lo general a máquinas o sistemas externos, estos últimos son más difíciles de identificar (Figura 5).

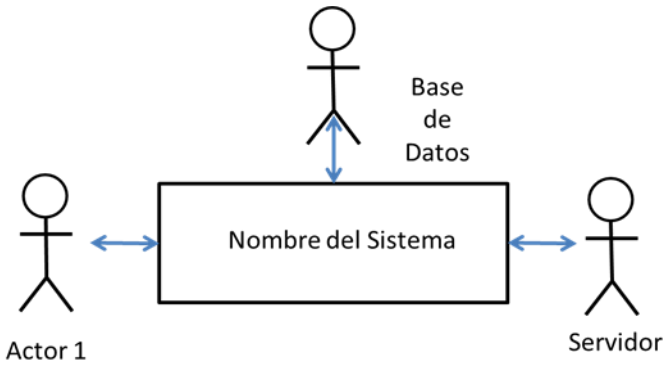


Figura 5. Actores primarios y secundarios

Caso de uso

Después de haber definido a los actores del sistema, se establece la funcionalidad propia del sistema por medio de los **casos de uso**. Cada caso de uso constituye un flujo completo de eventos, que especifican la interacción que toma lugar entre el actor y el sistema.

La descripción de los casos de uso se basa en diagramas similares a los de transición de estados. Se puede ver a cada caso de uso como si representara un estado en el sistema, donde un estímulo enviado entre un actor y el sistema ocasiona una transición entre estados.

A continuación se muestra un ejemplo de caso de uso, donde un mesero y un cajero comparten ciertos casos de uso, pero el cajero tiene un caso de uso diferente (Figura 6).

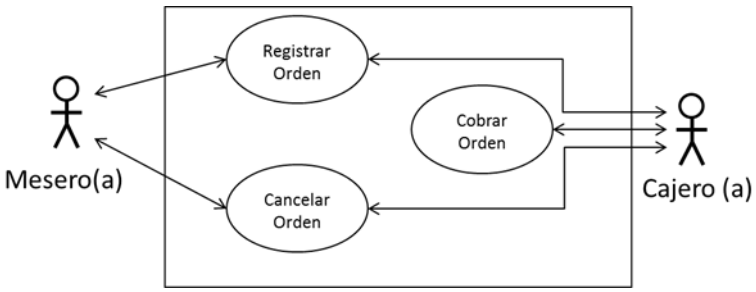


Figura 6. Casos de uso

Para identificar los casos de uso, se puede leer la descripción del problema y discutirlo con aquellos que actuarán como actores, empleando preguntas como:

- ¿Cuáles son las tareas principales de cada actor?
- ¿Tendrá el actor que consultar y modificar información del sistema?
- ¿Deberá el actor informar al sistema sobre cambios externos?

- ¿Desea el actor ser informado sobre cambios inesperados?

Extensión

Un concepto importante que se utiliza para estructurar y relacionar casos de uso es la **extensión** la cual especifica cómo un caso de uso puede insertarse en otro para extender la funcionalidad del anterior. El caso de uso donde se insertará la nueva funcionalidad debe ser un flujo completo, por lo cual éste es independiente del caso de uso a insertarse. De esta manera, el caso de uso inicial no requiere consideraciones adicionales al caso de uso a ser insertado, únicamente se especifica su punto de inserción (Figura 7).

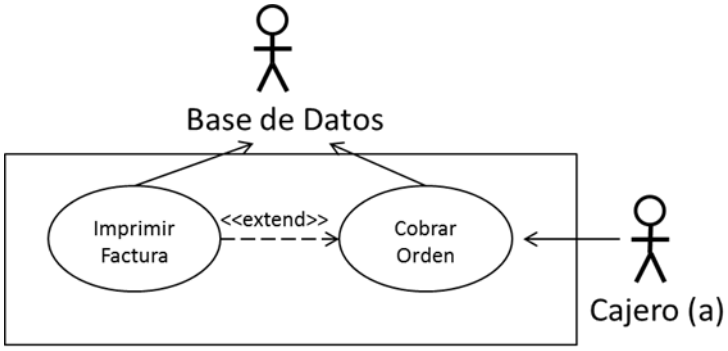


Figura 7. Extensión

Inclusión

Una relación adicional entre casos de uso es la inclusión. A diferencia de una extensión, la **inclusión** se define como una sección de un caso de uso que es parte obligatoria del caso de uso básico. El caso de uso donde se insertará la funcionalidad depende del caso de uso a ser insertado. Esta relación se etiqueta como “include” (Figura 8).

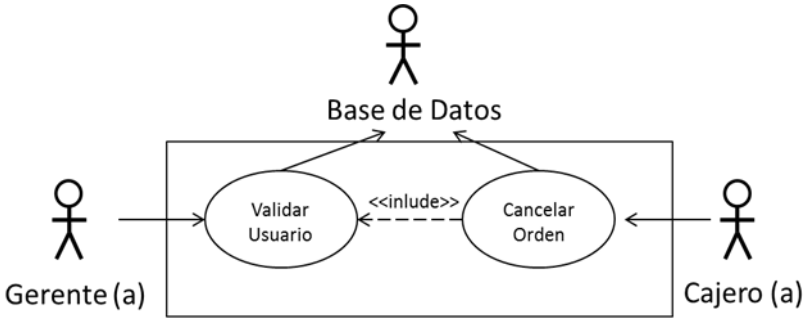


Figura 8. Inclusión

Generalización

Una relación adicional entre casos de uso es la **generalización**, la cual apoya a reutilización de los casos de uso. Mediante la relación de generalización es necesario describir las partes similares

una sola vez, en lugar de repetirlas para todos los casos de uso con comportamiento común. Los casos de uso extraídos se conocen como casos de uso abstractos, ya que no serán instanciados independientemente, y servirán sólo para describir partes que son comunes a otros casos de uso. Los casos de uso que realmente serán instanciados se llaman **casos de uso concretos**. Las descripciones de los casos de uso abstractos se incluyen en las descripciones de los casos de uso concretos.

Un ejemplo es cuando se usa el caso de uso **Pagar orden** de forma general, pero sin embargo este contempla dos casos de uso diferentes como es pagar con efectivo o con tarjeta.

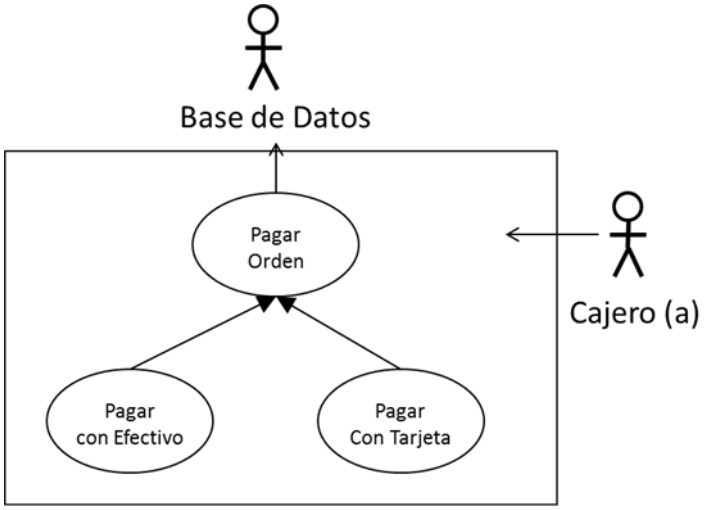


Figura 9. Generalización

Documentación

Parte fundamental del modelo de casos de uso es la descripción textual detallada de cada uno de los actores y casos de uso identificados. Estos documentos son sumamente críticos ya que a partir de ellos se desarrollará el sistema completo (Tabla 1).

Las descripciones de los casos de uso representan todas las posibles interacciones de los actores con el sistema en los eventos enviados o recibidos por éstos. En esta etapa no se incluyen eventos internos al propio sistema, ya que esto se estudiará durante el análisis, y únicamente agregará una complejidad innecesaria (Tabla 2).

ACTOR	ACCIONES
Nombre	Acciones que realiza

Tabla 1. Formato de documentación para cada actor

CASO DE USO	Nombre del caso de uso.
ACTORES	Actores primarios y secundarios que interaccionan con el caso de uso.
PROPÓSITO	Razón de ser del caso de uso.
RESUMEN	Resumen del caso de uso.

Tabla 2. Formato de documentación para cada caso de uso

Modelado de interfaces

El modelado de interfaces describe la presentación de información entre los actores y el sistema. Se especifica en detalle cómo se verán las interfaces de usuario al ejecutar cada uno de los casos de uso. Esto ayuda al usuario a visualizar los casos de uso según se mostrarán en el sistema a construirse (Figura 10).

Una vez que elaboradas todas las interfaces, es momento de realizar una presentación al cliente para que vea si el proceso de operación propuesto va de acuerdo con sus expectativas, de aprobarse la propuesta operativa se procederá a la identificación de objetos para el desarrollo del sistema y si el proyecto no es aprobado deberán realizarse los cambios pertinentes hasta que el cliente esté satisfecho.

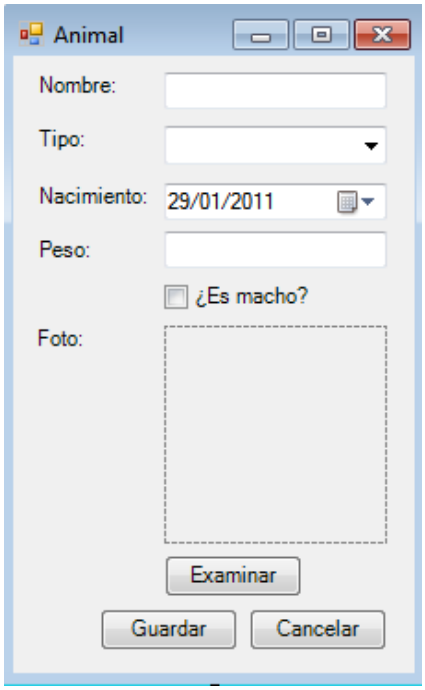


Figura 10. Modelado de interfaces

Modelo del dominio del problema

El modelo del dominio del problema define un modelo de clases común para todos los involucrados en el modelo de requisitos, tanto analistas como clientes. Este modelo de clases consiste en los objetos del dominio del problema, o sea objetos que tienen una correspondencia directa en el área de la aplicación.

Como los usuarios y clientes deben reconocer todos los conceptos, se puede desarrollar una terminología común al razonar sobre los casos de uso y por lo tanto, disminuir la probabilidad de malos entendidos entre el analista y el usuario.

Al discutirlo, se evolucionará en el modelo, para ello es conveniente darle al cliente un papel y un lápiz y pedirle que dibuje su visión del sistema.

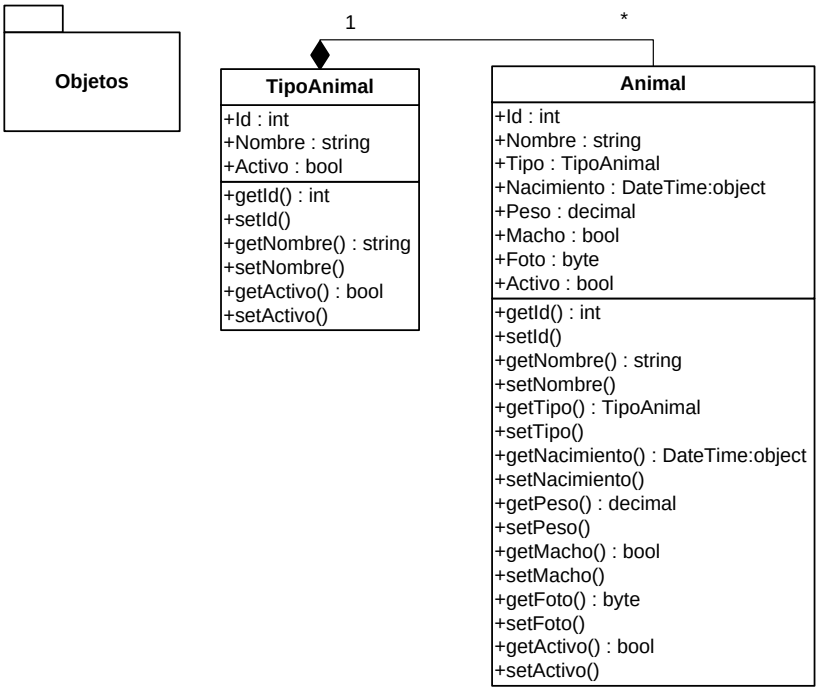


Figura 11. Modelo del dominio del problema

Objetos

Los objetos son entidades básicas del modelo de objeto. La palabra objeto proviene del latín objectus, donde ob significa bacia y jacere es arrojar; o sea, que teóricamente un objeto es cualquier cosa que se pueda arrojar. Los objetos son más que simples cosas que se puedan arrojar, son conceptos que a su vez se dividen en abstractos o concretos.

Clases

Una clase describe un grupo de objetos con estructura y comportamiento común. (Clase y tipo no son necesariamente equivalentes, tipo se define por las manipulaciones que se le puede dar a un objeto dentro de un lenguaje y clase involucra una estructura, que puede corresponder a una implementación particular de un tipo.

Las estructuras o propiedades de la clase se conocen como atributos y el comportamiento como operaciones. Una clase define o más objetos que pertenecen a la clase y que tienen características comunes. En general, el nombre de una clase debe iniciar con letra mayúscula (Figura 12).

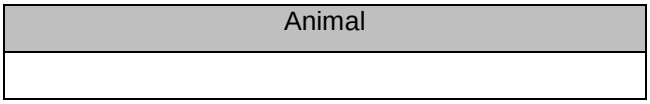


Figura 12. Notación para la clase

Instanciación

El proceso de crear objetos pertenecientes a una clase se conoce como instanciación, donde los objetos son las instancias de la clase. El objeto, es la instancia de la clase a la que pertenece. Se utiliza una flecha punteada para mostrar los objetos como instancias de las clases (Figura 13).

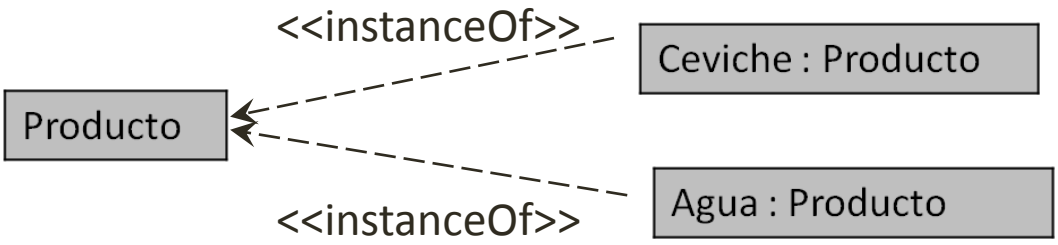


Figura 13. Notación para la instancia de objetos de cierta clase

Atributos

Los atributos definen la estructura de una clase y de sus correspondientes objetos. El atributo define el valor de un dato para todos los objetos pertenecientes a una clase.

Se debe definir un valor para cada atributo de una clase. Los valores pueden ser iguales o distintos en los diferentes objetos. No se puede dar un valor en un objeto si no existe un atributo correspondiente en la clase.

Dentro de una clase, los nombres de los atributos deben ser únicos (aunque puede aparecer el mismo nombre de atributo en diferentes clases).

Se puede asociar con cada atributo un tipo de dato, por ejemplo, entero, cadena, etcétera, para restringir sus posibles valores. El tipo se añade, separado por dos puntos, al diagrama de clases inmediatamente.

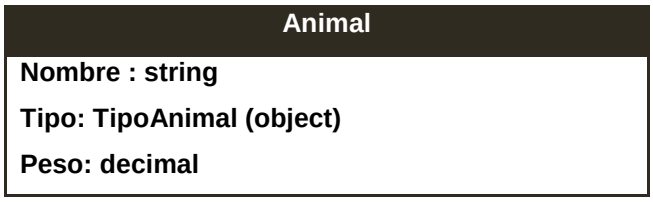


Figura 14. Atributos de la clase

Modificadores de Acceso

Una de las características más importantes y esenciales del lenguaje de programación orientada a objetos son sus modificadores de acceso, los cuales permiten conocer el alcance que tendrán tanto las clases y las variables que se definen (Tabla 3).

Visibilidad	Significado	C#	UML
Publica	Se puede acceder al miembro de la clase desde cualquier lugar.	public	+
Protegida	Sólo se puede acceder al miembro de la clase desde la propia clase o desde la clase que herede de ella.	protected	#
Privada	Sólo se puede acceder al miembro de la clase desde la propia clase.	private	-

Tabla 3. Modificadores de acceso

Métodos

El término método se utiliza para distinguir la operación como concepto de alto nivel de la propia implementación de la operación, algo correspondiente a un concepto de más bajo nivel.

Los métodos son la forma más fácil y eficiente para la reutilización de código, ya que cada método está compuesta por su propio bloque de código y puede ser invocado cada vez que se necesite evitando reescribir un trozo de código varias veces. La estructura básica de un método es la siguiente:

```
C#
public void NombreMetodo(){
        //Bloque de código
        }
```

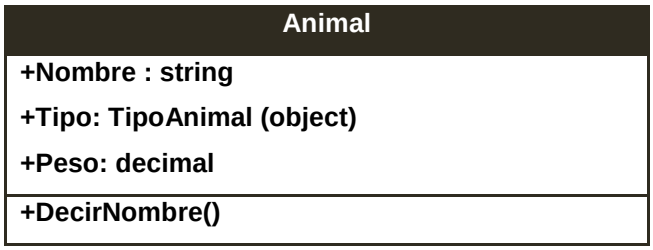


Figura 15. Métodos de la clase

Paso de valor

Los métodos pueden modificar el valor de variables que se encuentren en la línea de tiempo principal pero también podemos crear métodos que hagan uso de sus propias variables. Al proceso de enviarle un dato a un método se le llama paso de valor.

```
C#
private void DecirNombre (string Nombre){
            MessageBox.Show("Me llamo "+nombre);
        }
```

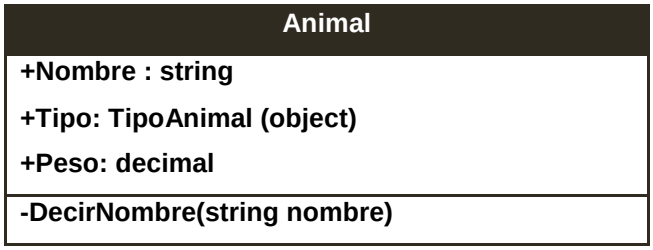


Figura 16. Paso de valor

Regreso de valores

Los métodos pueden terminar de dos formas: una es cuando el código que existe dentro del método ya se ha ejecutado o bien cuando aparece la palabra "return". Dicha palabra no solo sirve para terminar la función si no que también sirve para que el método regrese un valor a la línea de tiempo principal.

```
C#
private string getNombre(){
            return nombre;
        }
```

Producto
+Nombre : string +Tipo: TipoAnimal (object) +Peso: decimal
-DecirNombre(string nombre) -getNombre():string

Figura 17. Regreso de valores

Etapa de Desarrollo

El desarrollo consiste en usar los cianotipos creados durante la etapa de diseño para crear componentes reales (Figura 18).

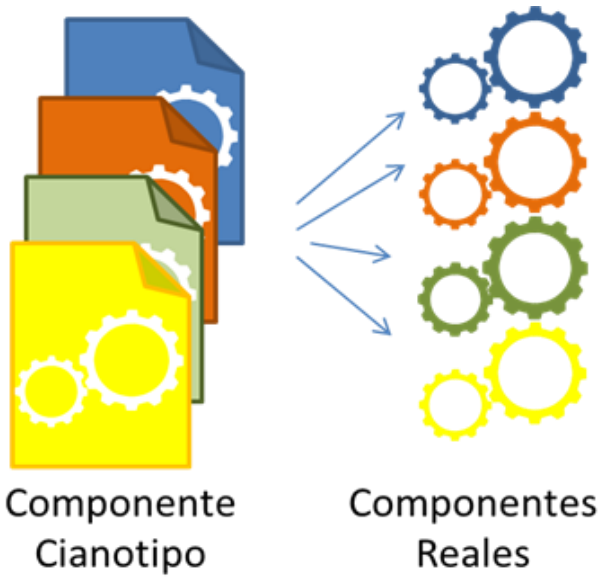


Figura 18. Etapa de desarrollo del PLC

Una vez obtenido el modelo del sistema, lo que sigue es definir de acuerdo a las necesidades del cliente el tipo de sistema a desarrollar, si será vía web arquitectura será cliente-servidor, el lenguaje que vayamos usar, etc.

Una vez que haya definido todas las herramientas que usara para el desarrollo es momento de poner manos a la obra.

Etapa de Prueba

Las pruebas consisten en la aseguración de aquellos componentes individuales, o el producto total, encontrar las exigencias de la especificación creada durante la etapa diseño. (El buen análisis del problema conduce a un buen diseño de la solución para así disminuir los errores y el tiempo de pruebas).

Las pruebas por lo general son realizadas por un equipo que consiste en gente ajena a la solución. Un equipo asegura que el producto es probado sin cualquier tendencia por parte del desarrollador (Figura 19).

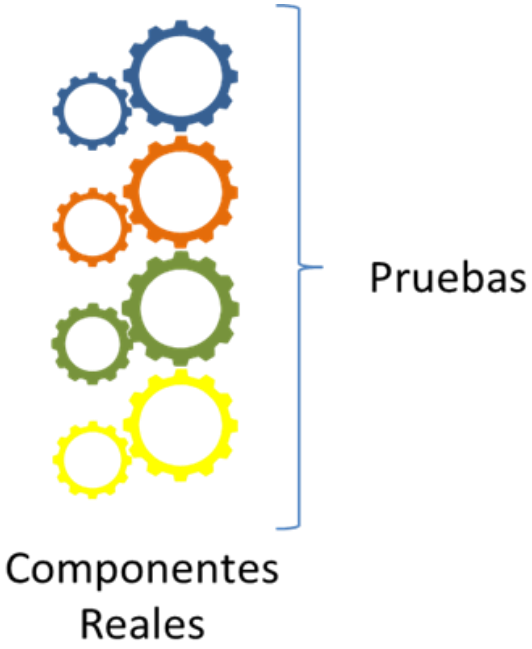


Figura 19. Etapa de pruebas del PLC

Etapa de Implementación

La implementación consiste en poner el producto disponible a los clientes para que lo utilicen (Figura 20).

Figura 20. Etapa de implementación del PLC

Etapa de mantenimiento

El mantenimiento consiste en reparar los problemas, volver a realizar las pruebas con el producto y liberar una nueva versión del producto con la revisión.

Este proceso puede realizar un número infinito de veces hasta que el producto se considere listo (Figura 21).

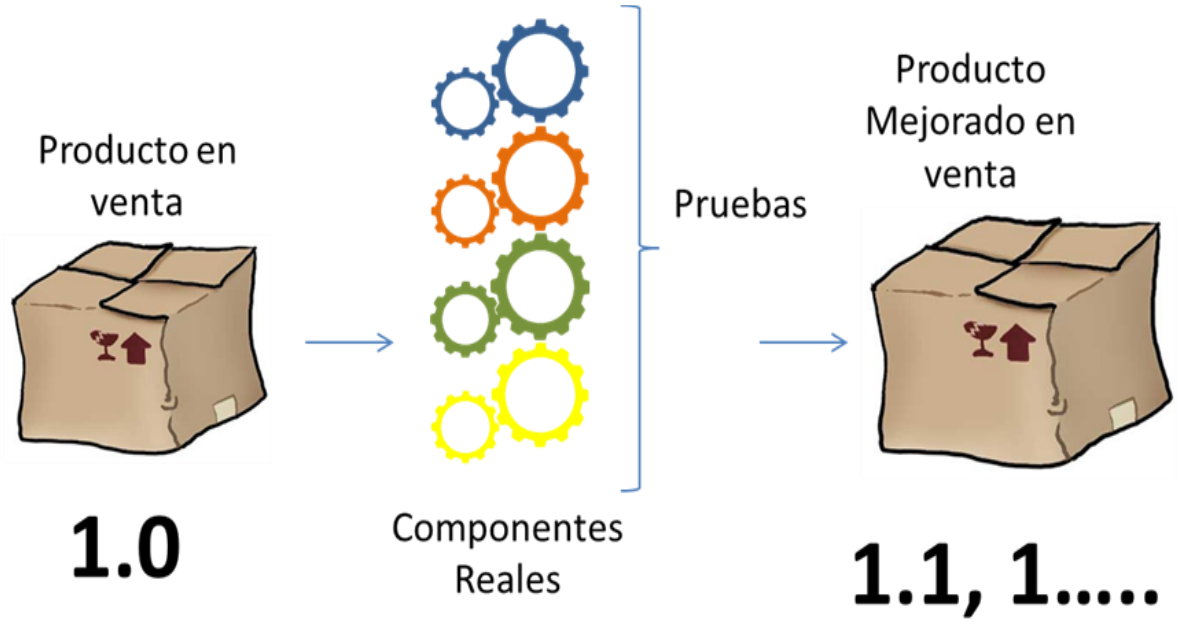


Figura 21. Etapa de mantenimiento del PLC

Etapa fin de vida

Una vez realizados los cambios necesarios y que el cliente está satisfecho es hora de finalizar con nuestra solución.

Mientras el PLC no tiene una etapa separada para el principio de un concepto o el proyecto, esto realmente tiene una etapa para el final de un proyecto. El fin de vida consiste en realizar todas las tareas necesarias de asegurar que los clientes y empleados son conscientes de que un producto

más está siendo vendido, apoyado y que un nuevo producto está disponible (

Figura 22).

Figura 22. Etapa fin de vida del PLC

Capítulo 3. Marco teórico

Sistema de información

Un sistema es una serie de artefactos (componentes) que en conjunto logran algún resultado. Un sistema de información es aquel que logra un resultado empresarial. Dicho con más detalle, un sistema de información recopila, manipula, almacena y crea reportes de información respecto de las actividades de negocios de una empresa, con el fin de ayudar a la administración de esa empresa en el manejo de las operaciones de negocio.

Hay dos categorías importantes de sistemas de información computarizados: los sistemas de información personalizados y los paquetes comerciales de distribución general (COST: Comercial off-the-shelf). Un sistema de información personalizado es aquel que se ha desarrollado para un cliente específico, del mismo modo que se hace un traje a la medida para un individuo. Por ejemplo, el sistema de información desarrollado para Jethro's Boot Emporium es un sistema de información personalizado, ninguna otra compañía utiliza el componente de detección de la tendencia en la moda de las botas.

Los tres principales interesados cuando se desarrolla (construye) un sistema de información son (Sharch, 2005):

- El cliente, quien paga por el sistema de información que se va a desarrollar.
- Los usuarios futuros del sistema de información.
- Los desarrolladores de ese sistema de información.

Arquitectura cliente-servidor

Una red cliente-servidor puede utilizarse de varias maneras, incluidos los métodos básicos siguientes (Figura 23):

- La manera más simple (y la más despilfarradora) consiste en que la red cliente-servidor funcione como una computadora de tiempo compartido. En otras palabras, los usuarios emplean sus computadoras personales únicamente como dispositivos de comunicación, es decir, como si fueran terminales tontas. Utilizan sus teclados para enviar instrucciones y datos al servidor, el cual después ejecuta sus programas. Los resultados se envían de regreso a las pantallas. Los usuarios no utilizan la capacidad de cómputo de sus computadoras personales, sólo el teclado y la pantalla.

- La red cliente-servidor funciona como una computadora distribuida. Los usuarios descargan los programas del servidor a sus propias computadoras y ejecutan los

programas ahí. Los datos para sus programas también pueden descargarse desde el servidor, ser introducidos por los usuarios o ambos. Si un cálculo requiere una computadora más grande que la que el usuario tiene en su escritorio, entonces el usuario ejecutará ese programa en el servidor, pero en general, la idea detrás de una red cliente servidor es realizar tanto trabajo como sea posible en la computadora de cada usuario (Sharch, 2005).

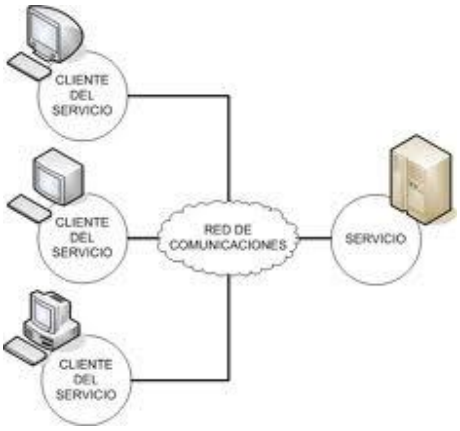


Figura 23. Arquitectura cliente - servidor

Servidor

Es el responsable de prestar los servicios requeridos por los clientes. Es común que la mayor parte de una aplicación con una arquitectura cliente-servidor se encuentre del lado del servidor.

Cliente

Es la persona que ejecuta la aplicación y quien demanda al servidor información para que funcione correctamente la aplicación.

Fluent NHibernate

Fluent NHibernate es una herramienta que se deriva de NHibernate y este último es derivado de Hibernate (Fluent NHibernate, 2010).

Hibernate comenzó siendo una herramienta de Mapeo objeto-relacional para la plataforma Java y posteriormente estuvo disponible también para .Net con el nombre de NHibernate. Esta herramienta facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

NHibernate es la conversión de Hibernate de lenguaje Java a C# para su integración en la plataforma .NET. Al igual que muchas otras herramientas libres para esta plataforma.

Al usar NHibernate para el acceso a datos el desarrollador se asegura que su aplicación es agnóstica en cuanto al motor de base de datos a utilizar en producción, pues NHibernate soporta los más habituales en el mercado: MySQL, PostgreSQL, Oracle, MS SQL Server, etc. Sólo se necesita cambiar una línea en el fichero de configuración para que podamos utilizar una base de datos distinta.

NHibernate es software libre, distribuido bajo los términos de la LGPL (Licencia Pública General Menor de GNU) (Wikipedia, NHibernate, 2011).

Al realizar mapeos con NHibernate se utiliza XML, se genera una gran cantidad de código, es por ello que nace Fluent NHibernate. Fluent NHibernate ofrece una alternativa a la norma NHibernate archivos de mapeo XML. En lugar de escribir documentos XML (archivos. Hbm.xml), Fluent NHibernate permite escribir en las asignaciones de código C #. Esto permite refactorización fácil y mejorar la legibilidad del código.

Fluent NHibernate también tiene varias herramientas, incluyendo:

- **Auto mappings**. Cuando las asignaciones se infieren a partir del diseño de sus entidades.
- **Prueba de especificación de persistencia**. Ronda de pruebas para sus entidades, sin tener que escribir una línea de CRUD (Create, Read, Update, Delete).
- Configuración de la aplicación completa con Fluent NHibernate API.
- Configuración de la base de datos en el código.

Fluent NHibernate es externo al núcleo de NHibernate , pero es totalmente compatible con la versión de NHibernate 2.1.

NHibernate es un Object Relational Mapping(ORM), entre los mapas de datos relacionales y objetos. Define sus asignaciones en un formato XML llamado HBM. Cada clase tiene su correspondiente archivo XML que se asigna a una determinada estructura en la base de datos. Si bien la separación de código y XML está bien, puede dar lugar a varias situaciones no deseadas:

- Debido a que los XML no son evaluados por el compilador, puede cambiar el nombre de propiedades en las clases que no se actualizan en sus asignaciones; en esta situación, no se enteraría de la rotura hasta que las asignaciones se analizan en tiempo de ejecución.
- XML es prolijo. NHibernate ha reducido gradualmente la obligatoriedad de elementos XML, pero todavía no puede escapar de la cantidad de mensajes XML.
- Asignaciones repetitivas. Las asignaciones de HBM pueden llegar a ser bastante detalladas, si se encuentra especificando las mismas reglas de nuevo. Por ejemplo, si es

necesario asegurar que todas las cadenas de propiedades no deben aceptar valores NULL y deben tener una longitud de 1000, y todos los int deben tener un valor predeterminado de -1.

Para contrarrestar los problemas mencionados anteriormente, Fluent NHibernate mueve sus asignaciones en el código actual, por lo que son compilados junto con el resto de la aplicación; las refactorizaciones permiten cambiar el nombre de las asignaciones y el compilador producirá un error en cualquier error tipográfico. En cuanto a la repetición, Fluent NHibernate tiene un sistema de configuración convencional, donde puede especificar patrones para neutralizar las convenciones de nomenclatura y muchas otras cosas, se establece cómo las cosas deben ser nombradas una vez, luego Fluent NHibernate hace el resto (Fluent NHibernate, 2010).

Programación por capas

La programación por capas es un estilo de programación en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario.

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado. Un buen ejemplo de este método de programación sería el modelo de interconexión de sistemas abiertos.

Además, permite distribuir el trabajo de creación de una aplicación por niveles; de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, de forma que basta con conocer la API que existe entre niveles.

En el diseño actual de sistemas informáticos se suelen usar las arquitecturas multinivel o **programación por capas**. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

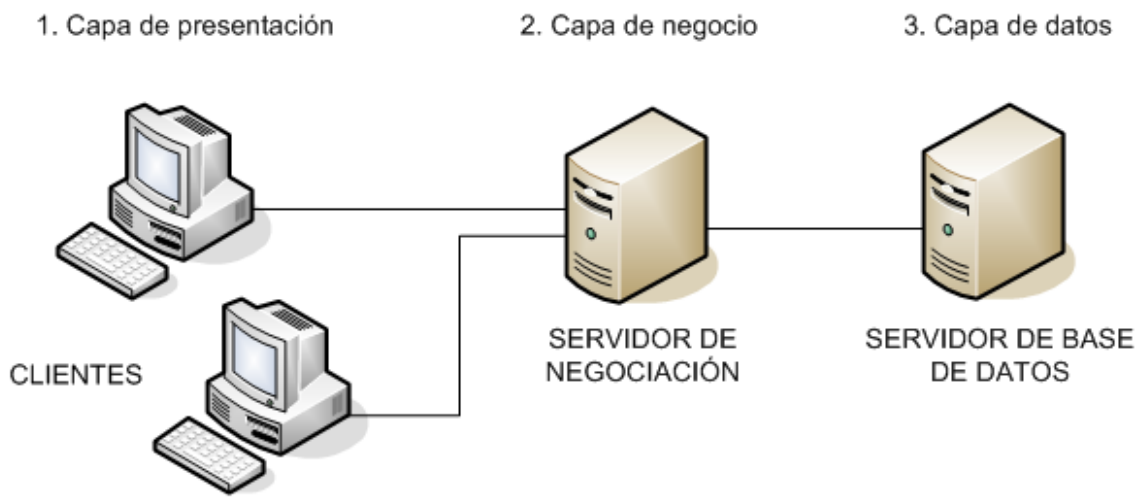


Figura 24. Modelo de la programación en capas

El diseño más utilizado actualmente es el diseño en tres niveles o en tres capas (Figura 24):

Capas y Niveles

- Capa de presentación: es la que ve el usuario (también se la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario. Esta capa se comunica únicamente con la capa de negocio.
- Capa de negocio: es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación.
- Capa de datos: es donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio (Wikipedia, Programación por capas, 2011).

.Net

Microsoft.NET es el conjunto de nuevas tecnologías en las que Microsoft ha estado trabajando durante los últimos años con el objetivo de obtener una plataforma sencilla y potente para distribuir

el software en forma de servicios que puedan ser suministrados remotamente y que puedan comunicarse y combinarse unos con otros de manera totalmente independiente de la plataforma, lenguaje de programación y modelo de componentes con los que hayan sido desarrollados. Ésta es la llamada plataforma .NET, y a los servicios antes comentados se les denomina servicios Web.

Para crear aplicaciones para la plataforma .NET, tanto servicios Web como aplicaciones tradicionales (aplicaciones de consola, aplicaciones de ventanas, servicios de Windows NT, etc.), Microsoft ha publicado el denominado kit de desarrollo de software conocido como .NET Framework SDK, que incluye las herramientas necesarias tanto para su desarrollo como para su distribución y ejecución y Visual Studio.NET, que permite hacer todo la anterior desde una interfaz visual basada en ventanas.

El concepto de Microsoft.NET también incluye al conjunto de nuevas aplicaciones que Microsoft y terceros desarrollan para ser utilizadas en la plataforma .NET. Entre ellas podemos destacar aplicaciones desarrolladas por Microsoft tales como: Windows.NET, Hailstorm, Visual Studio.NET, MSN.NET, Office.NET, y los nuevos servidores para empresas de Microsoft (SQL Server.NET, Exchange.NET) (Seco, 2010).

.NET Framework

.NET Framework es un entorno de desarrollo de software que abarca las bases de datos Access y otros tipos de bases de datos. En lugar de trabajar directamente con .NET Framework, los programadores utilizan su lenguaje de programación favorito. Para los programadores de Visual Basic y VBA, Visual Basic .NET representa la elección natural. Este lenguaje de programación funciona de forma similar a Microsoft Visual Basic 6.0, pero contiene características especiales, muchas de las cuales permiten sacar mayor partido de .NET Framework.

.NET Framework a través de Microsoft Visual Studio .NET proporciona varias plantillas para comenzar diferentes tipos de soluciones. Los programadores Access normalmente crean soluciones para una red de área local (LAN) o incluso un único equipo. La plantilla Aplicación Windows sirve especialmente bien en este tipo de solución. Tras abrir la plantilla, se sentirá como en casa con una barra de herramientas que permite arrastrar controles en u formulario que se abre automáticamente.

El formulario es una instancia de la clase *Form* de Windows. Aunque el formulario está adaptado a .NET Framework, funciona de forma muy similar a los formularios Visual Basic o Access. Por ejemplo, aún se puede utilizar el modelo de desarrollo familiar de formulario asociado a código que ya conocemos y amamos. La plantilla Aplicación Windows es la que se utilizará cuando la solución

deba beneficiarse de explorar completamente las características de interfaz de usuario del sistema operativo Microsoft Windows (Dobson, 2003).

.NET Framework 4

NET Framework 4. .NET Framework es un componente integral de Windows que admite la creación y funcionamiento de la próxima generación de aplicaciones y servicios Web. Los componentes clave de la NET Framework son: Common Language Runtime (CLR) y. NET Framework -incluye ADO.NET, ASP.NET, Windows Forms y Windows Presentation Foundation (WPF)- .NET Framework proporciona un entorno de ejecución administrado, desarrollo e implementación simplificados, y la integración con una amplia variedad de lenguajes de programación. Para una introducción a la arquitectura y las características principales del NET Framework (Microsoft, 2011).

C#

C# (leído en inglés "C Sharp" y en español "C Almohadilla") es el nuevo lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el lenguaje nativo de .NET

La sintaxis y estructuración de C# es muy similar a la C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son equiparables a los de Visual Basic.
Un lenguaje que hubiese sido ideal utilizar para estos menesteres es Java, pero debido a problemas con la empresa creadora del mismo -Sun-, Microsoft ha tenido que desarrollar un nuevo lenguaje que añadiese a las ya probadas virtudes de Java las modificaciones que Microsoft tenía pensado.

C# es un lenguaje de programación visual, controlado por eventos, en el cual se crean programas mediante el uso de un Entorno de Desarrollo Integrado (IDE). (Deitel & Deitel, 2007)

En resumen, C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo. El hecho de ser relativamente reciente no implica que sea inmaduro, pues Microsoft ha escrito la mayor parte de la BCL usándolo, por lo que su compilador es el más depurado y optimizado de los incluidos en el .NET Framework SDK. (Seco, 2010)

Una buena implementación de las etapas del ciclo de vida del producto y la correcta elección de herramientas para el desarrollo de un proyecto permiten tener una idea clara del proyecto a desarrollar para todos los integrantes involucrados, evitar errores y ahorrar tiempo, lo que se traduce en eficiencia.

Capítulo 4. Desarrollo

Etapa de análisis

El restaurant el “Pargo” localizado a las afueras de la ciudad de Chetumal en el estado de Quintana Roo, tiene como misión brindar un servicio rápido y de calidad siendo el principal negocio que ofrezca una variedad de platillos del mar. De igual manera tiene como visión en los próximos cinco años dar apertura a una sucursal dentro de la ciudad para tener un servicio más accesible a sus clientes.

El principal problema es cómo se lleva a cabo la generación de las cuentas y/o facturas de los clientes que atiende a diario, pues cuenta con alrededor de treinta mesas y en las horas pico más de un cliente pide su cuenta en mismo tiempo, por lo que tiene que tomar todas las comandas de la mesa que solicita la cuenta, calcular manualmente el precio de cada artículo pedido y sumar el total. Esto ocasiona un retraso al generar las cuentas y/o facturas causando molestia a los clientes y, en ciertos casos, la pérdida de capital ya que los clientes se van sin pagar.

El segundo problema es la compra excesiva de materia prima (camarón, pulpo, caracol, etc.) y la poca ganancia que esto genera. Esto es debido a que no se lleva un control de entradas y salidas, dando lugar a malos manejos de insumos.

La tercera problemática se da cuando un cliente realiza un pedido y el mesero tiene que ir hasta la barra para entregar la orden de bebidas y después a la cocina para entregar la orden de comida, esto demora mucho tiempo y en horas pico hay clientes que desean que sea rápido el servicio e incluso otros que esperan el desalojo de una mesa.

Etapa de diseño

Actores

Para este proyecto se han identificado los siguientes actores:

ACTOR	ACCIONES
Administrador	Controla todas las funciones del sistema.
Cajero	Imprime las cuentas de los clientes, cobra y entrega cambios.
Mesero	Levanta, registra y cancela las ordenes.
Cliente	Compra y paga los productos.

Cocinero	Recibe y elabora platillos.
Barman	Recibe y elabora bebidas.

Diagrama de Clases

De acuerdo a la información recabada de cada uno de los actores se ha diseñado el siguiente diagrama de clases.

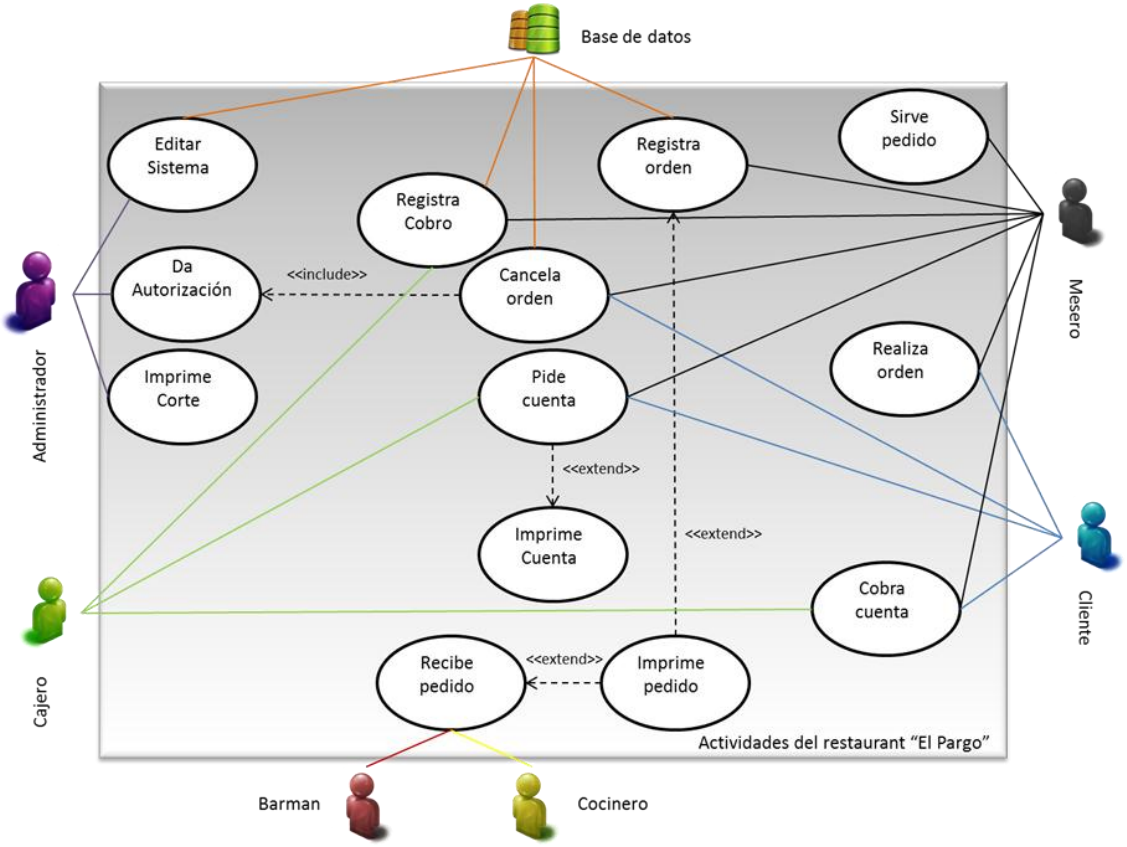


Figura 25. Diagrama de Casos de Uso del negocio

Casos de uso

CASO DE USO	Realiza orden
ACTORES	Cliente(inicia), Mesero
PROPÓSITO	Solicitar Pedido
RESUMEN	El cliente llama al mesero para solicitarle un pedido.

Caso de uso 1. Realiza orden

CASO DE USO	Registra orden
ACTORES	Mesero
PROPÓSITO	Capturar orden en el sistema

RESUMEN	El mesero va a una terminal donde captura toda la orden y al terminar la envía a impresión. Incluye **Imprime pedido**

Caso de uso 2. Registra orden

CASO DE USO	Imprime pedido
ACTORES	Sistema
PROPÓSITO	Informar a la cocina y barra de movimientos
RESUMEN	Cuando el mesero ingresa o cancela una orden, se envía la información a la barra o cocina para su ejecución.

Caso de uso 3. Imprime pedido

CASO DE USO	Cancela orden
ACTORES	Cliente(inicia),Mesero, Administrador
PROPÓSITO	Cancelar en el sistema una orden
RESUMEN	El cliente solicita la cancelación de un artículo o de la orden completa o mesero puede confundirse al momento de captura. Requiere **Dar autorización**.

Caso de uso 4. Cancela orden

CASO DE USO	Dar autorización
ACTORES	Mesero(inicia), Administrador
PROPÓSITO	Permitir algún movimiento
RESUMEN	El mesero solicita autorización para cancelar algún artículo u orden y necesita la autorización del administrador.

Caso de uso 5. Dar autorización

CASO DE USO	Recibe pedido
ACTORES	Mesero(inicia), Cocinero, Barman
PROPÓSITO	Producir platillos y bebidas
RESUMEN	El mesero captura un pedido o cancela alguno, y reciben de inmediato en la cocina o la barra el movimiento para elaborar los platillos y/o bebidas, así como cancelar la elaboración del mismo. Incluye **Imprime pedido.**

Caso de uso 6. Recibe pedido

CASO DE USO	Sirve pedido
ACTORES	Mesero(inicia), Cliente
PROPÓSITO	Servir al cliente
RESUMEN	El mesero recibe una alerta de que algún pedido ya está listo, el mesero toma el pedido y lo lleva a su respectivo cliente.

Caso de uso 7. Sirve pedido

CASO DE USO	Pide cuenta
ACTORES	Cliente(inicia), Mesero, Cajero
PROPÓSITO	Solicitar la cuenta
RESUMEN	El cliente llama al mesero para que le entregué la cuenta que va a pagar, el mesero se dirige a al cajero y solicita la cuenta o la puede imprimir el directamente. Requiere **Imprime cuenta.**

Caso de uso 8. Pide cuenta

CASO DE USO	Imprime cuenta
ACTORES	Sistema
PROPÓSITO	Imprimir cuenta
RESUMEN	El cajero imprime un ticket de cobro con la descripción de los artículos de la orden solicitada y/o elabora la factura del cliente que solicita la cuenta.

Caso de uso 9. Imprime cuenta

CASO DE USO	Cobra cuenta
ACTORES	Cajero(inicia), Mesero, Cliente
PROPÓSITO	Llevar cuenta
RESUMEN	El cajero llama al mesero para que lleve el ticket y/o factura al cliente que la solicito.

Caso de uso 10. Cobra cuenta

CASO DE USO	Registra cobro
ACTORES	Mesero(inicia), Cajero
PROPÓSITO	Registrar el cobro
RESUMEN	El mesero toma la cuenta pagada y se la lleva al cajero para que registre el cobro en el sistema.

Caso de uso 11. Registra cobro

CASO DE USO	Imprime corte
ACTORES	Administrador
PROPÓSITO	Imprimir corte diario
RESUMEN	El finalizar el día el administrador accede al sistema e imprime el corte del día o de otros días, que incluyen las órdenes y los montos cobrados.

Caso de uso 12. Imprime corte

CASO DE USO	Editar sistema
ACTORES	Administrador
PROPÓSITO	Configurar parámetros del sistema
RESUMEN	El administrador accede al sistema y: Agrega, Modifica y Elimina Ordenes Agrega, Modifica y Elimina Mesas Agrega, Modifica y Elimina Empleados Agrega, Modifica y Elimina Artículos

	Agrega, Modifica y Elimina Tipos de Empleados Agrega, Modifica y Elimina Categorías Agrega, Modifica y Elimina Impresoras

Caso de uso 13. Editar sistema

Modelado de interfaces

En esta etapa se diseñarán las interfaces que permitirán al usuario interactuar con el sistema, es momento de comenzar a darle forma al sistema. Es muy importante en este punto que además de realizar el diseño de las interfaces se describa el funcionamiento y/o comportamiento de la interfaz, con el fin de tener una idea clara de cómo será empleada y el programador pueda realizar una adecuada labor de desarrollo. No omitir ningún dato del funcionamiento de la interfaz por más obvio que pueda parecer, de lo contrario dará pie a que el desarrollador interprete en cualquier forma.

Este punto es muy crucial porque de aquí dependerá que un buen trabajo permita al diseñador y programador realizar una tarea mutua y bien coordinada. Es importante recordar que la programación por capas permite este tipo de trabajo en equipo, ya que no es necesario que el programador realice ambas tareas.

 Interfaz

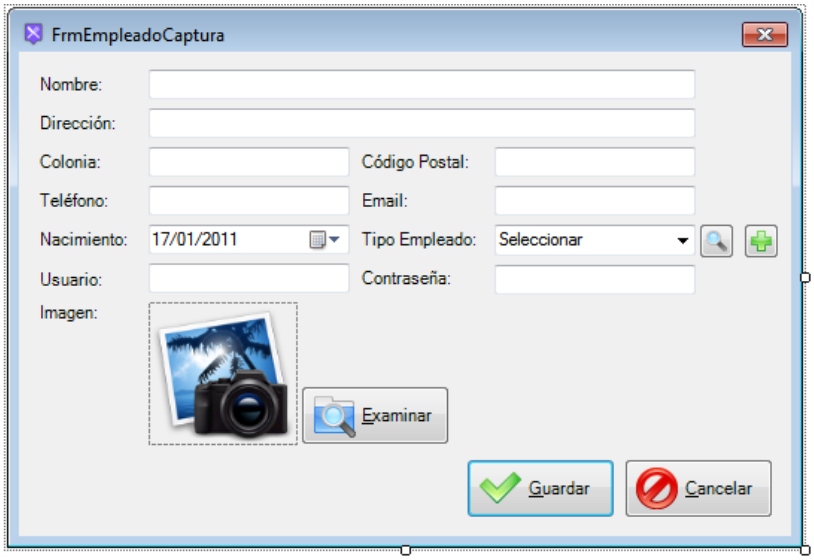


Figura 26. Formulario captura de empleados


Descripción

Esta interfaz permitirá al usuario capturar los datos de los empleados que tienen acceso al sistema. Los empleados serán clasificados por varios tipos como: Administrador, Cajero o Mesero, donde cada uno tendrá acceso a diferentes secciones del sistema según sea el caso.

La interfaz tendrá dos funciones: agregar nuevos empleados y modificar los datos de un empleado previamente agregado. Cuando la ventana tenga la función de agregar, el titulo deberá ser "Agregar Empleado" y si la ventana tiene la función de modificar, el titulo deberá ser "Modificar Empleado".

La ventana será de un tamaño definido y no podrá ser ampliada. Cuando el usuario pulse la tecla Enter hará referencia a la acción del botón Guardar y cuando la tecla que se pulse sea Escape hará referencia a la acción del botón Cancelar.

Cuando el usuario pulse el botón de Guardar, se tendrá que validar que todos los campos estén llenos de lo contrario deberá enviarse el mensaje de "Llene todos los campos.". Si todos los campos están llenos se procederá agregar o modificar los datos según sea el caso.

Al guardar los cambios se le informará que los datos han sido guardados con el mensaje de "Información guardada correctamente. ¿Desea continuar?", que le brindará al usuario la opción seguir agregando o modificando datos según sea el caso.

Si el usuario pulsa el botón Cancelar aparecerá un mensaje de "Desea guardar los cambios", brindando la opción de que el usuario pueda guardas sus modificaciones en caso de haber pulsado el botón erróneamente.

Modelo del dominio del problema

Una vez que se tiene definido el comportamiento del sistema y que el cliente ha aceptado la propuesta, se procederá al diseño de Clases que permitirán la interacción del usuario con la base de datos.

Pará definir las clases es recomendable comenzar identificando los objetos que interactúan, los cual puede realizarse en el modelado de interfaces y sus descripciones. El diagrama de clases permite apreciar visualmente el comportamiento del sistema de forma sintetizada. (Ver figura 27).

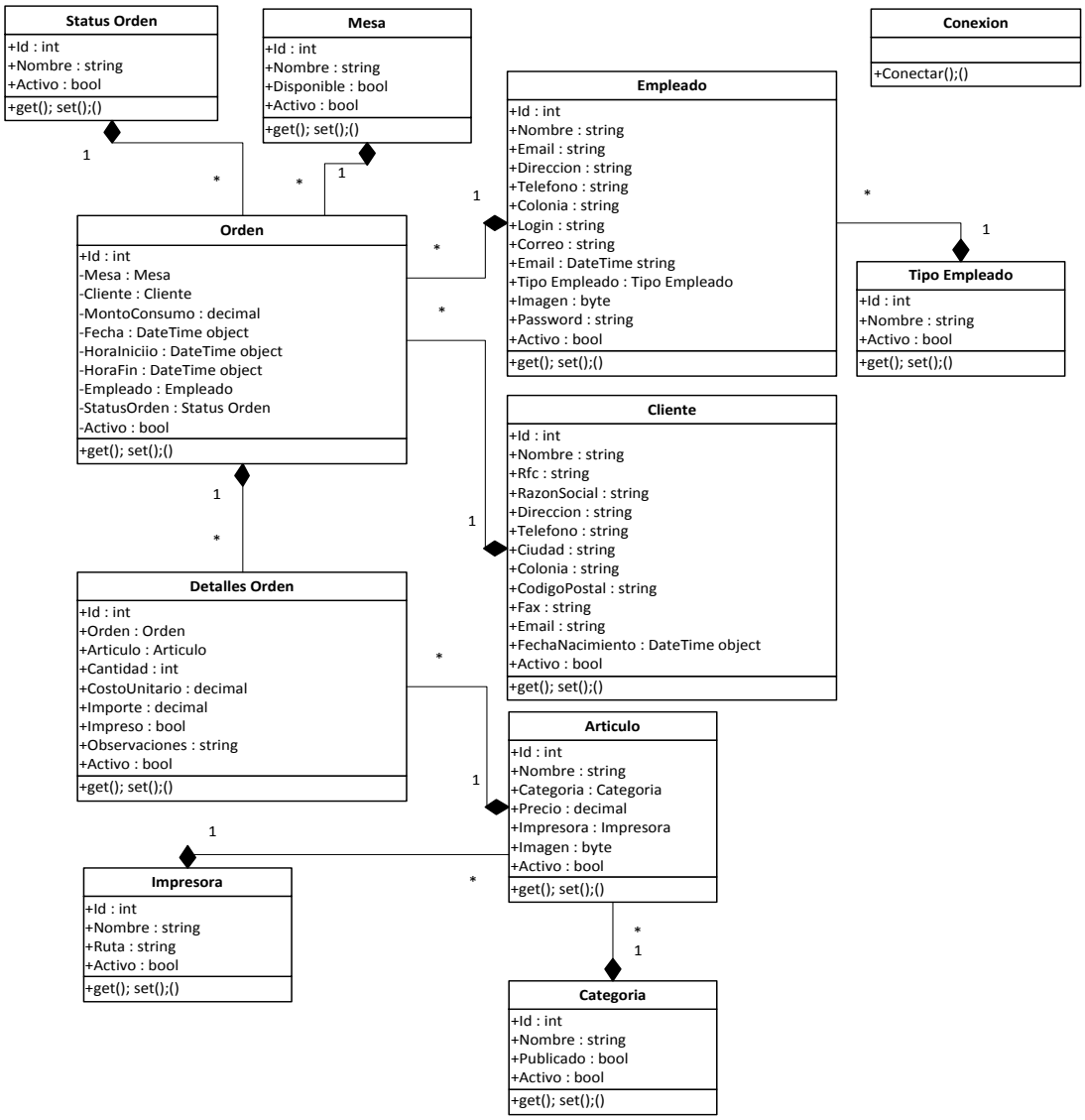


Figura 27. Diagrama de Clases

Etapa de desarrollo

Elección de tecnología

Lo primero que se debe realizar antes de comenzar a desarrollar cualquier sistema es definir la tecnología con la que se cuenta para su desarrollo, así como definir la forma en que se llevará a cabo la codificación.

Para este proyecto se utilizará Visual Studio 2010 usando el lenguaje de programación C Sharp (C#). Además para realizar la interacción entre el sistema y la base de datos usaremos Fluent NHibernate.

Creando los proyectos

La solución está compuesta por dos proyectos el primero es un Windows Forms Application y el otro por una Class Library (Figura 28).

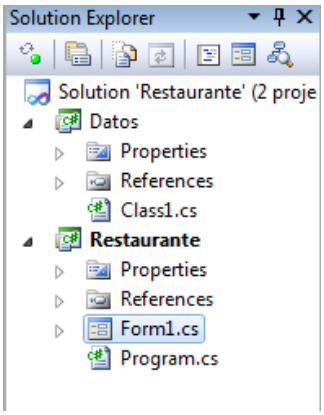


Figura 28. Proyectos

El Windows Forms Application en Visual 2010 tendrá las siguientes características (Figura 29):

- El lenguaje es de tipo Visual C#
- Del tipo Windows Forms Application
- El nombre del proyecto es Restaurante.
- El Framework 4

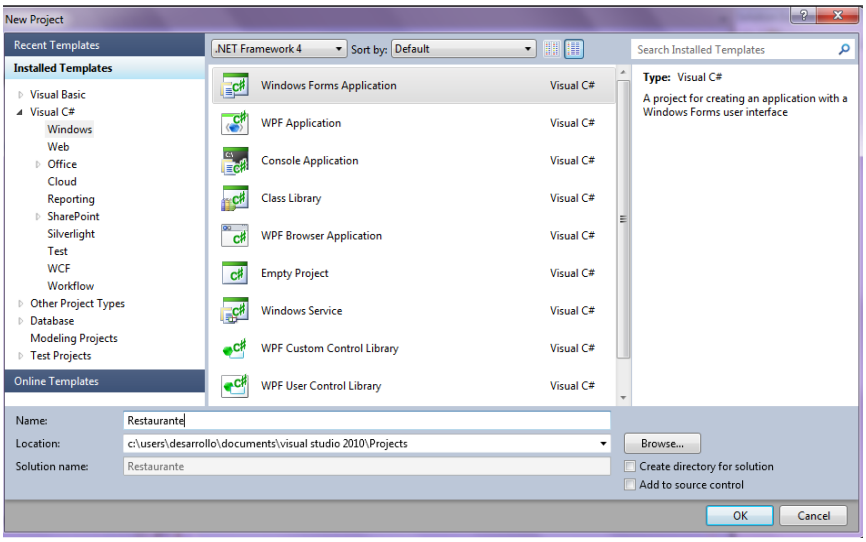


Figura 29. Creación del Windows Form Application

Por defecto el nombre de la solución es el mismo que el nombre asignado al proyecto. Este proyecto contendrá toda la interfaz del sistema y funcionalidad del sistema.

La Class Library tendrá las siguientes características (Figura 30):

- El lenguaje es de tipo Visual C#
- Del tipo Class Library
- El nombre del proyecto es Restaurante.
- El Framework 4

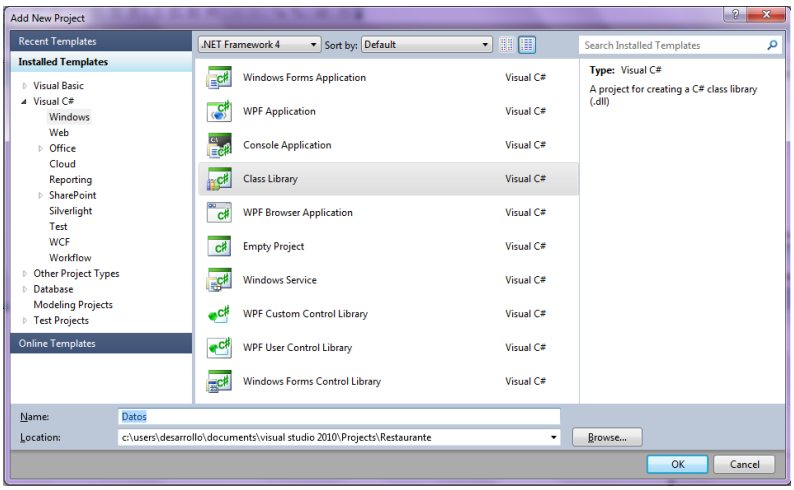


Figura 30. Creación de la Class Library

Este proyecto es una librería de clases que servirá para los tres procesos necesarios, el de administración, el de captura y el de caja.

Las clases para NHibernate forman el papel más importante ya que no sólo permitirán el fácil acceso a la los datos, sino que de igual forma permiten la fácil migración a otras bases de datos.

Instalación de Fluent NHibernate

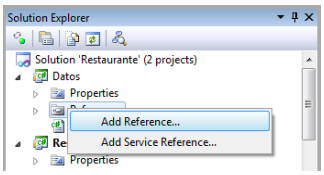


Figura 31. Agregar dlls

Para usar Fluent NHibernate se necesita agregar una referencia a las dlls que permita usarlas. Las dll pueden ser descargadas del portal web de esta compañía (Fluent NHibernate, 2010) o podrán ser encontradas en el CD en la carpeta de dll.

Para agregar una referencia se debe pulsar el botón secundario sobre las referencias del proyecto Datos y seleccione Agregar referencia (Figura 31).

En la ventana emergente se pulsa el botón de buscar para localizar la ubicación de las dlls y agregarlas (Figura 32).

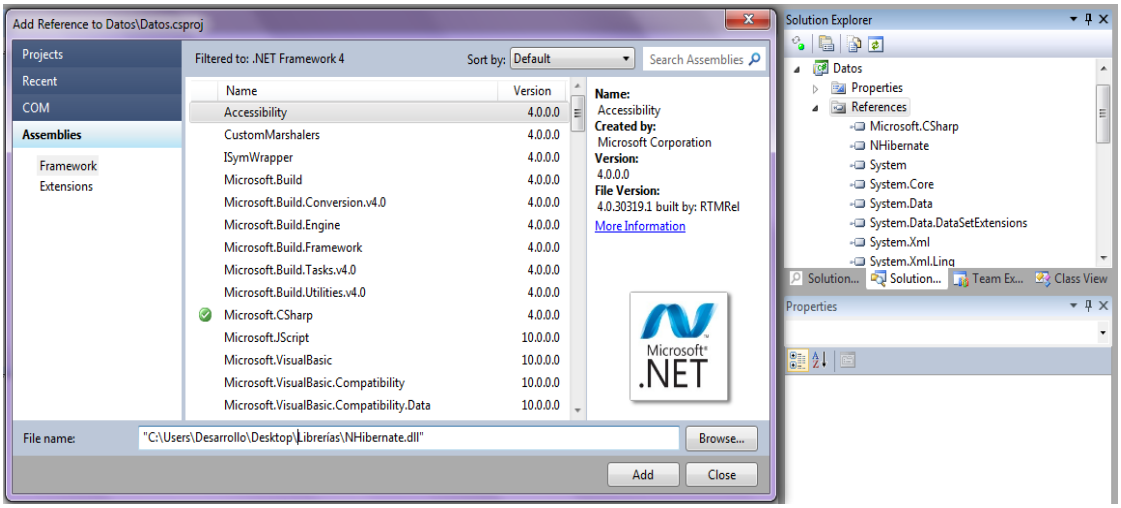


Figura 32. Buscar dlls

Creación de una base datos

Figura 33. Conexión con el servidor SQL

Para crear una base de datos se instala SQL Server 2008 Management Studio, posteriormente se abre el programa y se selecciona el servidor (Figura 33).

Después de conectarse al servidor se debe hacer clic con el botón secundario sobre la carpeta Bases de datos y seleccionar Nueva base de datos en el menú contextual. Se debe escribir el nombre de la base de datos y pulsar el botón de Aceptar para crear la base de datos (Figura 34).

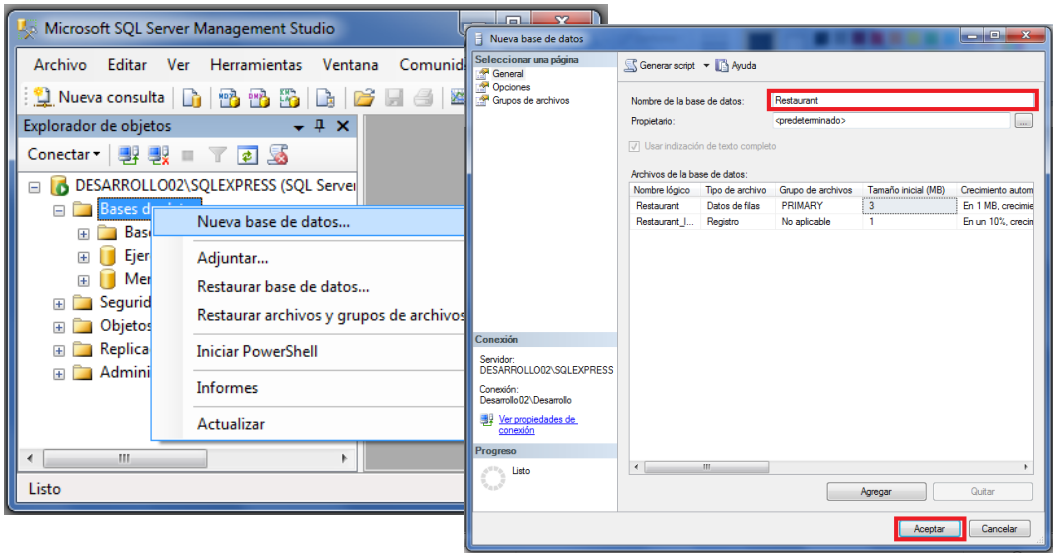


Figura 34. Creación de una base de datos

Creación de una conexión en Visual Studio 2010

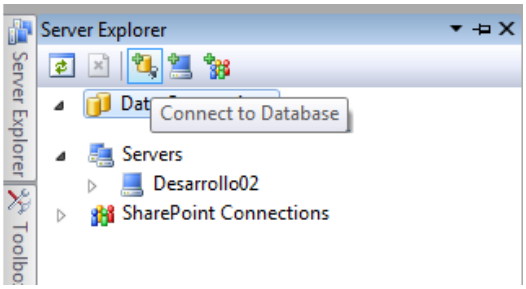


Figura 35. Explorador de Servidores

Una conexión en Visual Studio 2010 permite explorar los datos que se generan en la base de datos y sus tablas durante el desarrollo. Para crear una conexión se abre el explorador de servidores y pulsa el botón de Connect to Database (Figura 35).

Se debe seleccionar el servidor donde se creó la base de datos, la base que se creó y pulsar el botón OK para conectarse[1] (Figura 36).

[1] Si se elige el servidor de tipo SQLEXPRESS hay que agregarle al nombre del servidor \sqlexpress. Si el servidor es local podemos usar (local)\sqlexpress

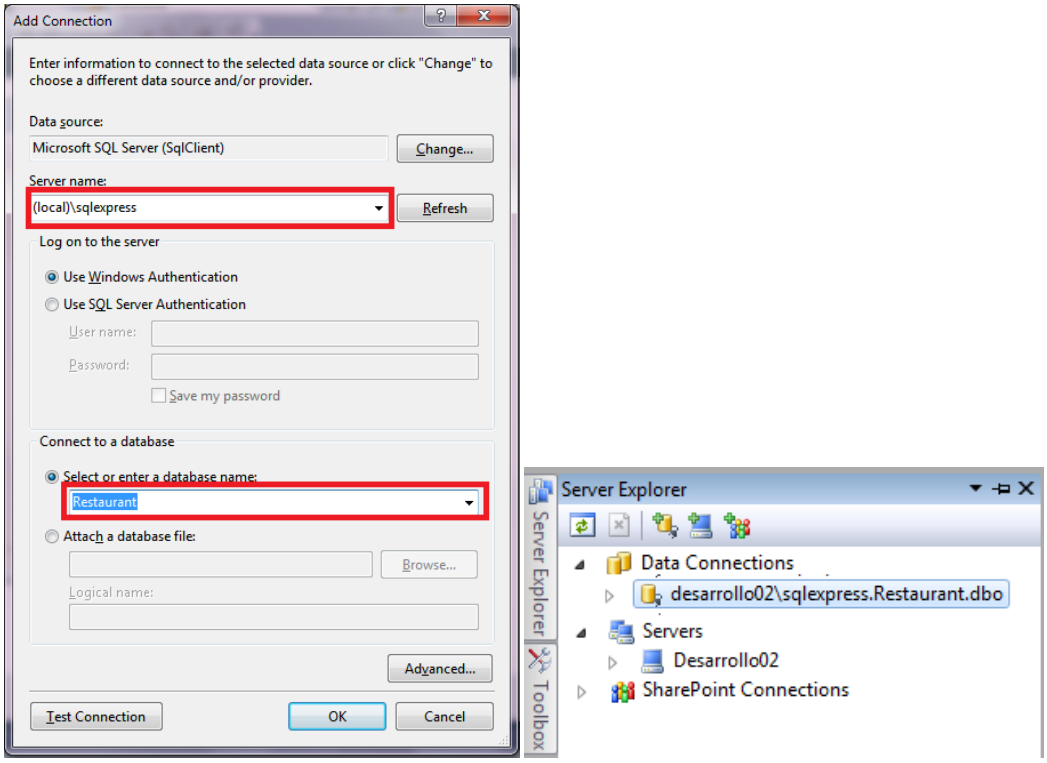


Figura 36. Agregar conexión

Configuración de Fluent NHibernate

Fluent NHibernate proporciona una API (Código 1) para configurar completamente Nhibernate y utilizarlo con la aplicación. La API se divide en cinco métodos principales, tres de los cuales son necesarios: `Database, Mappings y BuildSessionFactory`.

```
C#
Fluently.Configure()
  .Database(/* your database settings */)
  .Mappings(/* your mappings */)
  .ExposeConfiguration(/* alter Configuration */) // optional
  .BuildSessionFactory();
```

Código 1. API Fluent Nhibernate

Se pueden combinar estos métodos de diversas maneras para configurar la aplicación:

- `Fluently.Configure()` inicia el proceso de configuración
- `Database()` es donde se especifica la configuración de su base de datos mediante la configuración de la base API
- `Mappings()` es donde la oferta que las asignaciones de que está utilizando.

- `ExposeConfiguration()` es opcional, pero le permite modificar el objeto de configuración (BuildSchema).

- `BuildSessionFactory()` es la última llamada, y se crea la instancia de SessionFactory NHibernate de su configuración.

La API es bastante pequeña y relativamente visible, así que es fácil de configurar de una manera que funcione con la solicitud.

Generar Esquema

Si no se ha creado manualmente el esquema para esta aplicación, entonces se producirá un error en la primera vez que lo ejecute. Para esto se puede llamar al método `ExposeConfiguration` y generar en tiempo de ejecución el esquema de la base de datos (Código 2).

C#
```
private static void BuildSchema(Configuration config)
{
  // Cada vez que esta line de código se ejecute se crearán nuevamente las tablas
  // este código solo deberá ejecutarse una vez y después ser comentado
   new SchemaExport(config)
    .Create(false, true);
}
```

Código 2. BuildSchema

Conexión con Fluent NHibernate

Para crear la conexión de Fluent NHibernate a la base datos, dentro del proyecto **Restaurante** se agregó una clase con el nombre de Conexion.cs la cual contiene los siguientes datos.

C#
```
using FluentNHibernate.Cfg;
using FluentNHibernate.Cfg.Db;
using NHibernate;
using NHibernate.Cfg;
using NHibernate.Tool.hbm2ddl;

namespace Datos
{
    public class Conexion
    {
        public static ISessionFactory CreateSessionFactory()
        {
            return Fluently.Configure()
                .Database(MsSqlConfiguration.MsSql2008
                            .ConnectionString(c => c.FromConnectionStringWithKey("conexionString")))
                .Mappings(m =>
                          m.FluentMappings.AddFromAssemblyOf<ClienteClass>())
                .ExposeConfiguration(BuildSchema)
                .BuildSessionFactory();
        }


        private static void BuildSchema(Configuration config)
        {
            new SchemaExport(config).Create(false, true);
        }
    }
}
```

Configurando app.config

El app.config permite a Fluent NHibernate identificar la ruta de la base de datos. Fluent NHibernate funciona únicamente con Framework 4. Para verificar si tiene el Framework correcto se hace clic secundario sobre el proyecto **Datos** y se selecciona Propiedades en el menú contextual[2] (Figura 37).

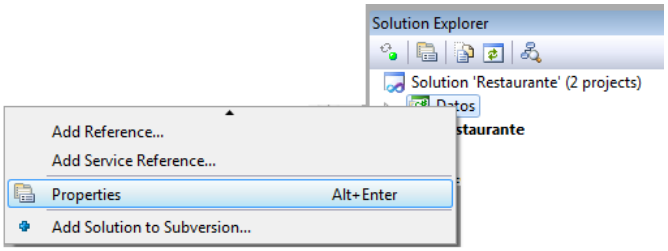


Figura 37. Propiedades de un proyecto

Se verifica en la pestaña de Application que el Framework que esté seleccionado sea el 4 y no el 4 Client Profile (Figura 38).

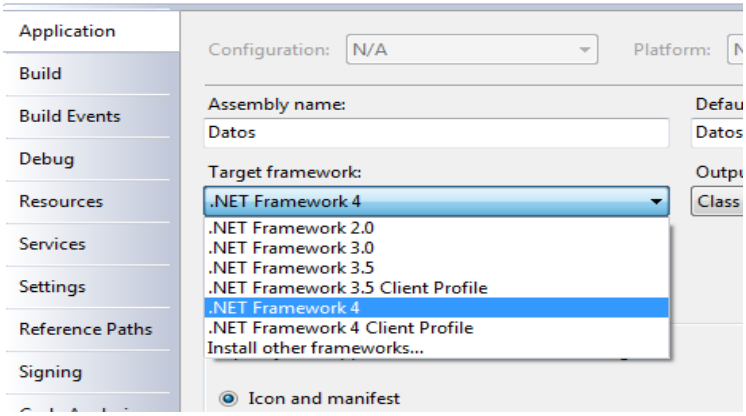
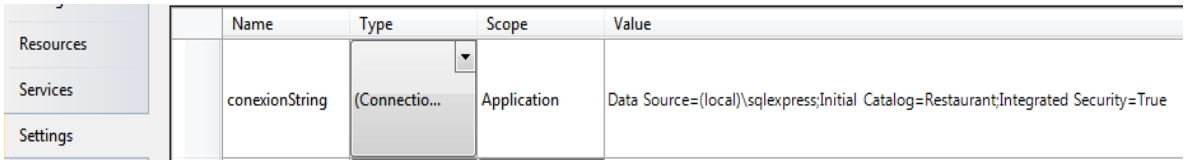


Figura 38. Framework 4

Después en la pestaña de Settings se creó una conexión de tipo cadena, la cual llama a **conexionString (**Figura 39**)** y en la columna de Value se hace clic para abrir una ventana como la de conexión a la base de datos, donde se realizó el mismo proceso (Figura 36).



Una vez terminada la configuración se abre el archivo app.config del proyecto Datos y se edita

Figura 39. Settings del proyecto

[2] Este proceso se realiza para ambos proyectos (Datos y Restaurante).

dejando únicamente el nombre de conexionString o el mismo nombre que definido en el archivo de conexión NHibernate (Figura 40).

Figura 40. app.config

Creación de Clases

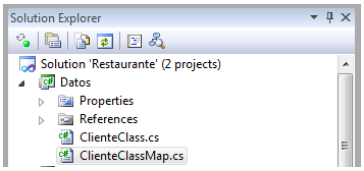


Figura 41. Class y Class Map

Para comenzar agregar las clases, son necesarios dos elementos una que será la clase para manipular la información dentro de los proyectos y otro que servirá para decirle a Fluent NHibernate como manipulará la base de datos. Los elementos que vamos a crear deberán seguir la siguiente estructura.

El nombre para la clase es el nombre del objeto en singular más el sufijo Class y para el mapeo de la clase será el nombre del objeto en singular más el sufijo ClassMap (Figura 41).

Clase Cliente

C#

```
using System;
```

```
namespace Datos
{
    public class ClienteClass
    {
        public virtual int Id { get; set; }
        public virtual string Nombre { get; set; }
        public virtual string Rfc { get; set; }
        public virtual string RazonSocial { get; set; }
        public virtual string Direccion { get; set; }
        public virtual string Telefono { get; set; }
        public virtual string Ciudad { get; set; }
        public virtual string Colonia { get; set; }
        public virtual string CodigoPostal { get; set; }
        public virtual string Fax { get; set; }
        public virtual string Email { get; set; }
        public virtual DateTime FechaNacimiento { get; set; }
        public virtual bool Activo { get; set; }
    }
}
```

Toda clase deberá llevar el campo Activo, el motivo es porque anteriormente se solía eliminar un registro completamente de la base de datos, pero eso tenía el riesgo de crear incongruencias en el sistema cuando ese registro tenía algún enlace con otro.

Por ese motivo se comenzó a realizar validaciones a la base de datos para verificar que el registro a eliminar no tenga ninguna referencia, pero esto ocasiona más tiempo de desarrollo, más código al ejecutar y más consultas a la base de datos. Así que se comenzó agregar el campo de Activo con el fin de cuando se elimine un registro sólo será de forma lógica, evitando incongruencias.

Mapeo Clase Cliente3

C#

```
using FluentNHibernate.Mapping;

namespace Datos
{
    public class ClienteClassMap : ClassMap<ClienteClass>
    {
        public ClienteClassMap()
        {
            /*Determinamos el nombre de la tabla*/
            Table("Clientes");
            /*Determinamos la columna llave*/
            Id(x => x.Id).Column("Id_Cliente").GeneratedBy.Identity();
            /*Determinamos los campos*/
            /*La opción Column ; indica el nombre del campo que se va crear en la base de datos*/
            /*La opción de Length; especifica el tamaño del campo, si no se espeficica toma uno por defec
to*/
            Map(x => x.Nombre).Column("Nombre").Length(100);
            Map(x => x.Rfc).Column("Descripcion").Length(20);
            Map(x => x.RazonSocial).Column("RazonSocial");
            Map(x => x.Direccion).Column("Direccion");
            Map(x => x.Telefono).Column("Telefono").Length(20);
            Map(x => x.Ciudad).Column("Ciudad").Length(20);
            Map(x => x.Colonia).Column("Colonia").Length(30);
            Map(x => x.CodigoPostal).Column("Codigo_Postal").Length(15);
            Map(x => x.Fax).Column("Fax").Length(20);
            Map(x => x.Email).Column("Email").Length(50);
            Map(x => x.FechaNacimiento).Column("Fecha_Nacimiento");
            Map(x => x.Activo).Column("Activo");
        }
    }
}
```

[3] Las clases faltantes se encuentran en el Anexo B

De igual manera toda clase deberá llevar el campo Id, el motivo es porque se solía dar nombres diferentes a los campos llave de las tablas y estos campos llave en algún momento podrían ser modificados.

Un ejemplo es que un empleado que desea cambiar su usuario y éste fue utilizado en todas las tablas que tienen relación con él, obligaría a realizar una consulta a casi todas las tablas del sistema buscando al usuario anterior y remplazándolo por el nuevo usuario, esto representa una gran cantidad de consultas a la base de datos que podría consederarse excesiva además de que no todos los registros serán modificados (aquí se corre el riesgo de incongruencia en la base de datos). Es por eso que todas las clases deberán llevar un Id único independientemente si tuviesen un campo de código, clave, etc.

Recursos (Resources)

Los recursos almacenarán todas las imágenes que se usarán para el diseño de los formulario, si ya se tienen planificado cuales imágenes serán usadas en el diseño pueden ser agregadas antes de comenzar a crear los formularios, en caso contrario se puede hacer durante el proceso.

Para agregar recursos al proyecto, se selecciona el proyecto Restaurante y pulsa la combinación de teclas ALT + ENTER para abrir las propiedades del proyecto. En la pestaña de Resources se debe pulsar en la barra de menú el botón de Add Resources y en el menú contextual se selecciona la opción de Add Existing File.
En la ventana emergente se deben seleccionar todas las imágenes que van a ser usadas y se agregan (Figura 42).

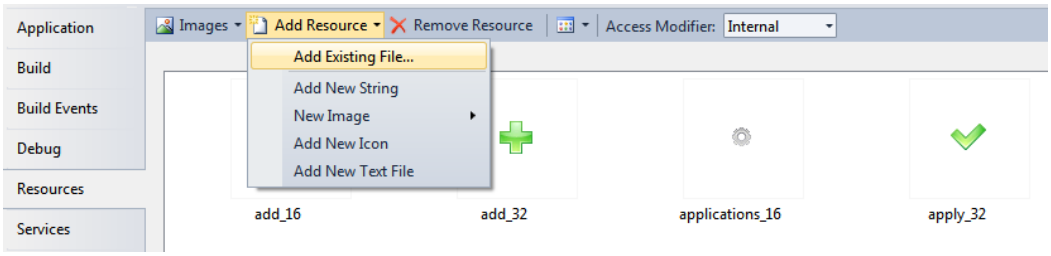


Figura 42. Agregar recursos al proyecto

Controles de Visual Studio 2010

Para el desarrollo de los formularios se necesitan diferentes controles que ofrece Visual Studio 2010. El tipo de control que se usan son de acuerdo al tipo de dato que el objeto necesite (Figura 43).

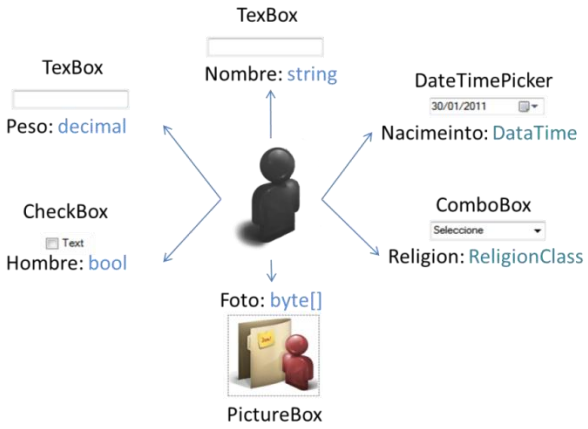
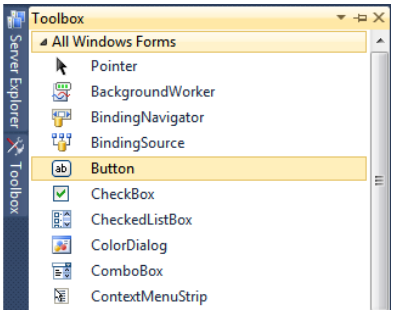


Figura 43. Relación de componentes con tipos de datos

Creación de controles

Para crear cualquier control dentro de los formularios existen dos maneras de hacerlo. La primera y la más sencilla es arrastrando el control desde la caja de herramientas hacia el formulario. La segunda es crear el control por medio de código C# creando una instancia de la clase, asignarle sus propiedades y finalmente agregarla a los controles del formulario (Figura 44).




```
private void Form1_Load(object sender, EventArgs e)
{
    TextBox texBox = new TextBox();
    texBox.Size = new Size(40,20);
    texBox.Name = "txtTexto";
    texBox.Location = new Point(250,10);
    this.Controls.Add(texBox);
}
```

Figura 44. Creación de instancias

Propiedades

Todos los controles tienen una cierta cantidad de propiedades según sea su caso, estas propiedades permiten al desarrollador definir cómo se comportará el control. La forma más sencilla de editar las propiedades de un control es seleccionarlo y modificarlo desde el panel de propiedades (Figura 45).

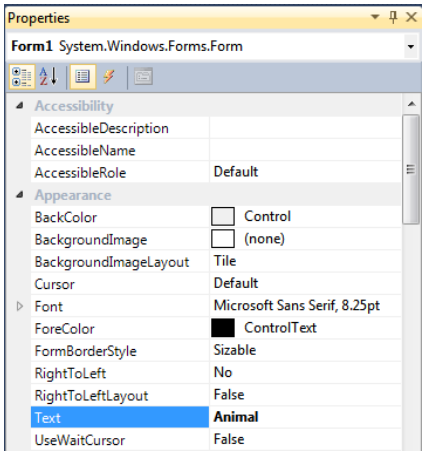


Figura 45. Propiedades de los controles

Manejo de datos

Cada control es diferente de otro y por lo tanto es usado en diferentes circunstancias, por lo que si va a manejar un tipo de dato string no sería conveniente usar un control para manejar fechas. Visual Studio nos ofrece una gran cantidad de controles de forma gratuita los cuales se acoplan de forma muy práctica a sus necesidades, como por ejemplo: TextBox, ComboBox, Button, etc.

Eventos

Un evento se refiere a cualquier acontecimiento, circunstancia, suceso o caso posible al que un control es sometido. De la misma forma que las propiedades cada control tiene una cierta cantidad de eventos propios. Un evento de por sí solo no desencadena la ejecución de código, por ello debemos escribir explícitamente con código que reacciones tendrá (Figura 46).

Los tipos de eventos pueden agruparse conceptualmente en dos categorías: Eventos de Usuario, donde es necesaria una acción de una persona física ajena al sistema.

Para tener acceso a los eventos de un control únicamente selecciónelo y en el panel de propiedades seleccione el evento que desee usar.

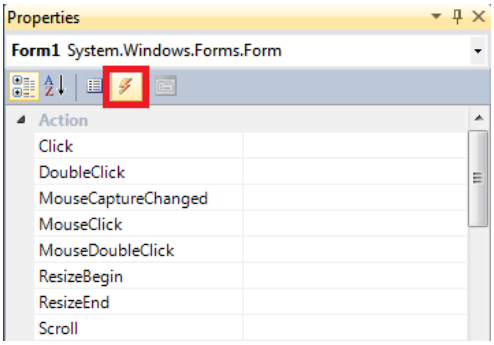
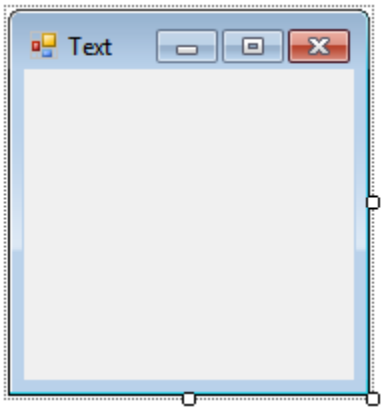


Figura 46. Eventos de los controles

Principales Controles de Interfaz Gráfica para Usuario

A continuación se describen los controles gráficos utilizados para el desarrollo de la aplicación y se pueden consultar en Microsoft Software Developer (Network).

Form (Clase)



Descripción

Un formulario Form es una representación de cualquier ventana mostrada en su aplicación. La clase Form se puede utilizar para crear ventanas estándar, de herramientas, sin bordes y flotantes. También puede utilizar la clase Form para crear las ventanas modales como un cuadro de diálogo. Para identificar un Form es recomendable usar el prefijo “Frm” al nombre (frmNombre).

Propiedades

Propiedad	Descripción
AcceptButton	Obtiene o establece el botón del formulario que se activa cuando el usuario presiona la tecla ENTRAR.
CancelButton	Obtiene o establece el control de botón que se activará cuando el usuario presione la tecla ESC.
Height	Obtiene o establece el alto del control.
Icon	Obtiene o establece el icono del formulario.
KeyPreview	Obtiene o establece un valor que indica si el formulario recibe los eventos clave antes de que pasen al control que tiene el foco.
MaximizeBox	Obtiene o establece un valor que indica si se muestra el botón Maximizar en la barra de título del formulario.
MinimizeBox	Obtiene o establece un valor que indica si se muestra el botón Minimizar en la barra de título del formulario.
Name	Obtiene o establece el nombre del control.
StartPosition	Obtiene o establece la posición inicial del formulario en tiempo de ejecución.
Text	Obtiene o establece el titulo de la ventana.
Width	Obtiene o establece el ancho del control.
WindowsState	Obtiene o establece el estado de la ventana del formulario.

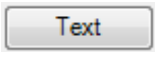
Manejo de datos

Para los formularios el manejo de datos esta defino por los eventos que se realizan.

Eventos

Evento	Descripción	Actor
KeyDown	Se produce cuando se presiona una tecla mientras el control tiene el foco.	Usuario
Load	Se produce antes de que se muestre un formulario por primera vez.	Sistema

Button (Clase)



Descripción

El Button servirá para cuando se desee realizar un evento controlado al hacer clic en Button utilizando el mouse (ratón), la tecla ENTRAR o la BARRA ESPACIADORA si el botón tiene foco. Para identificar un Button es recomendable usar el prefijo “btn” al nombre (btnNombre).

Se establece la propiedad AcceptButton o CancelButton de un objeto Form para permitir a los usuarios hacer clic en un botón presionando ENTRAR o ESCAPE incluso si el botón no tiene foco. Esto proporciona al formulario el comportamiento de un cuadro de diálogo.

Propiedades

Propiedad	Descripción
Anchor	Obtiene o establece los bordes del contenedor al que está enlazado un control y determina cómo se cambia el tamaño de un control con su elemento primario.
Height	Obtiene o establece el alto del control.
Image	Obtiene o establece la imagen que se muestra en un control de botón.
Name	Obtiene o establece el nombre del control.
TabIndex	Obtiene o establece un valor que determina el orden en el que los elementos reciben el foco cuando el usuario navega por los controles utilizando la tecla TAB.
Text	Obtiene o establece el titulo de la ventana.
TextImageRelation	Obtiene o establece la posición del texto y de la imagen entre sí.
Visible	Obtiene o establece un valor que indica si se muestran el control y todos sus controles primarios.
Width	Obtiene o establece el ancho del control.

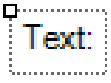
Manejo de datos

Para los formularios el manejo de datos está definido por los eventos que se realizan.

Eventos

Evento	Descripción	Actor
Clic	Se produce cuando se hace clic en el control.	Usuario

Label (Clase)



Descripción

Los controles Label se utilizan normalmente para proporcionar texto descriptivo de un control. Por ejemplo, se puede utilizar un objeto Label para agregar texto descriptivo para un control TextBox e informar al usuario del tipo de datos que se espera tener en el control. Los controles Label se pueden utilizar también para agregar texto descriptivo a un Form para proporcionar al usuario información útil.

Por ejemplo, se puede agregar Label en la parte superior de Form que proporciona al usuario instrucciones sobre cómo especificar datos en los controles del formulario. Los controles Label se pueden utilizar también para mostrar información en tiempo de ejecución sobre el estado de una aplicación. Por ejemplo, se puede agregar un control Label a un formulario para mostrar el estado de cada archivo cuando se procesa una lista de archivos.

Para identificar un Label es recomendable usar el prefijo “lbl” al nombre (lblNombre).

Propiedades

Propiedad	Descripción
Name	Obtiene o establece el nombre del control.
Text	Obtiene o establece el titulo de la ventana.
Visible	Obtiene o establece un valor que indica si se muestran el control y todos sus controles primarios.

Manejo de datos

C#

```
//Creación de variable de tipo string
string texto;

//Creación de variable tipo entero
int numero = 10;

//Se estable al control una cadena de texto
lblNombre.Text = "hola";

//Se estable al control un numero por medio de una cadena de texto
lblNombre.Text = numero.ToString();

//Se obtiene del control una cadena de texto
texto = lblNombre.Text;
try
{
      //Se obtiene del control una cadena de texto y se convierte a un numero
      numero = Convert.ToInt32(lblNombre.Text);
}
catch {
      //Si la cadena de texto no es numérica enviará un error al convertir
      MessageBox.Show("Ingrese un número.");
}
```

Eventos

Para este proyecto no es usado ninguno de sus eventos.

TextBox (Clase)

Descripción

Un TextBox es ideal para la manipulación de datos de tipo string, int, decimal y double. Para identificar un Textbox es recomendable usar el prefijo “txt” al nombre (txtNombre).

Propiedades

Propiedad	Descripción
CharacterCasing	Se puede utilizar la propiedad CharacterCasing para cambiar la condición de mayúscula o minúscula de los caracteres según requiera la aplicación. Por ejemplo, puede cambiar la condición de mayúscula o minúscula de todos los caracteres escritos en un control TextBox que se utiliza como entrada de contraseña a mayúsculas o minúsculas con el fin de exigir una directiva para las contraseñas.
Height	Obtiene o establece el alto sugerido del elemento.
Name	Obtiene o establece el nombre del control.
PasswordChar	Establece un carácter que sustituye a los caracteres capturados.
ReadOnly	Obtiene o establece un valor que indica si el control de edición de texto es de sólo lectura para un usuario que interactúa con el control.
TabIndex	Obtiene o establece un valor que determina el orden en el que los elementos reciben el foco cuando el usuario navega por los controles utilizando la tecla TAB.
Text	Obtiene o establece el contenido del texto del cuadro de texto.
Visible	Obtiene o establece un valor que indica si este elemento está visible en la interfaz de usuario (UI).
Width	Obtiene o establece el ancho del elemento.

Manejo de datos

C#

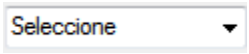
```
//Creación de variable de tipo string
string texto;

//Creación de variable tipo entero
int numero = 10;

//Se estable al control una cadena de texto
txtNombre.Text = "hola";

//Se estable al control un numero por medio de una cadena de texto
txtNombre.Text = numero.ToString();

//Se obtiene del control una cadena de texto
texto = txtNombre.Text;
try
```

```
{
      //Se obtiene del control una cadena de texto y se convierte a un numero
      numero = Convert.ToInt32(txtNombre.Text);
}
catch {
      //Si la cadena de texto no es numérica enviará un error al convertir
      MessageBox.Show("Ingrese un número.");
}
```

Eventos

Para este proyecto no es usado ninguno de sus eventos.

ComboBox (Clase)



Descripción

Un ComboBox muestra un campo de edición de cuadro de texto combinado con un ListBox y permite al usuario seleccionar elementos de la lista o escribir texto nuevo. El comportamiento predeterminado de ComboBox es mostrar un campo de edición con una lista desplegable oculta.

En el proyecto se usa el ComboBox cuando se necesita hacer una lista de objetos para que el usuario seleccione uno. Para identificar un ComboBox es recomendable usar el prefijo “cbo” al nombre (cboNombre).

Propiedades

Propiedad	Descripción
Name	Obtiene o establece el nombre del control.
ReadOnly	Obtiene o establece un valor que indica si el control de edición de texto es de sólo lectura para un usuario que interactúa con el control.
TabIndex	Obtiene o establece un valor que determina el orden en el que los elementos reciben el foco cuando el usuario navega por los controles utilizando la tecla TAB.
Text	Obtiene o establece valor null del control.
Visible	Obtiene o establece un valor que indica si este elemento está visible en la interfaz de usuario (UI).

Manejo de datos

```
C#
//Creación de una instancia al objeto Categoría
private CategoriaClass categoria;

private void CargarCategorias()
{
  //Se crea una lista de los objetos categorías
  var lista = from c in Session.Query<CategoriaClass>()
              where c.Activo
              select c;

  //Se estable el atributo del objeto se va a mostrar
  cboCategorias.DisplayMember = "Nombre";

  //Se estable el atributo del objeto que servirá de valor
```

```
  cboCategorias.ValueMember = "Id";

  //Se llena el ComboBox con la lista
  cboCategorias.DataSource = lista.ToList();

  //Se selecciona por defecto el campo null
  cboCategorias.SelectedIndex = -1;
}

private void  AsignarCategoria(){
   //Se verifica si el usuario selecciono un valor del ComboBox
   if (cboCategorias.SelectedValue == null){

       //Si no ha seleccionado ningún valor termina el proceso
       return;
   }

   //Se obtiene el objeto seleccionado
   categoria = (CategoriaClass)cboCategorias.SelectedValue;
}
```

Eventos

Evento	Descripción	Actor
SelectedValueChanged	Se produce cuando cambia la propiedad SelectedValue.	Usuario

CheckBox (Clase)

Descripción

CheckBox da al usuario una opción del tipo verdadero/falso o sí/no. Este control es recomendable usarlo cuando se maneja datos de tipo booleano. Para identificar un CheckBox es recomendable usar el prefijo “chk” al nombre (chkNombre).

Propiedades

Propiedad	Descripción
Checked	Obtiene o establece un valor que indica si CheckBox está en el estado activado.
Name	Obtiene o establece el nombre del control.
ReadOnly	Obtiene o establece un valor que indica si el control de edición de texto es de sólo lectura para un usuario que interactúa con el control.
TabIndex	Obtiene o establece un valor que determina el orden en el que los elementos reciben el foco cuando el usuario navega por los controles utilizando la tecla TAB.
Text	Obtiene o establece la etiqueta del control.
Visible	Obtiene o establece un valor que indica si este elemento está visible en la interfaz de usuario (UI).

Manejo de datos

C#

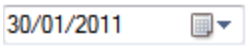
```
//Creación de variable de tipo bool
bool Activo = true;
```

```
//Se estable el valor al control
chkNombre.Checked = Activo;

//Se obtiene el valor del control
Activo = chkNombre.Checked;
```

Eventos

Evento	Descripción	Actor
CheckedChanged	Se produce cuando cambia el valor de la propiedad Checked.	Usuario

DateTimePicker (Clase)



Descripción

El control DateTimePicker se utiliza para permitir al usuario seleccionar una fecha y una hora, y mostrarlas en el formato especificado. Es posible limitar las fechas y las horas que se pueden seleccionar al establecer las propiedades MinDate y MaxDate. Este control es recomendable usarlo cuando se manjea el tipo de datos DataTime. Para identificar un DateTimePicker es recomendable usar el prefijo “dtp” al nombre (dtpNombre).

Propiedades

Propiedad	Descripción
Format	Obtiene o establece el formato de fecha y hora que se muestra en el control.
Name	Obtiene o establece el nombre del control.
ReadOnly	Obtiene o establece un valor que indica si el control de edición de texto es de sólo lectura para un usuario que interactúa con el control.
TabIndex	Obtiene o establece un valor que determina el orden en el que los elementos reciben el foco cuando el usuario navega por los controles utilizando la tecla TAB.
Text	Obtiene o establece la etiqueta del control.
Visible	Obtiene o establece un valor que indica si este elemento está visible en la interfaz de usuario (UI).

Manejo de datos

```
C#
//Creación de variable de tipo DateTime con la fecha actual
DateTime fecha = DateTime.Today;

//Se estable el valor al control
dtpNombre.Value = fecha;

//Se obtiene el valor del control
fecha = dtpNombre.Value;
```

Eventos

Evento	Descripción	Actor
ValueChanged	Se produce cuando cambia la propiedad Value.	Usuario

PictureBox (Clase)

Descripción

Se suele utilizar el control PictureBox para mostrar gráficos de un archivo de mapa de bits, metarchivo, icono, JPEG, GIF o PNG. Este control es recomendable usarlo cuando se maneja un tipo de datos byte[]. [4]Para identificar un PictureBox es recomendable usar el prefijo “pb” al nombre (pbNombre).

Propiedades

Propiedad	Descripción
Image	Obtiene o establece la imagen que se muestra mediante un PictureBox.
Name	Obtiene o establece el nombre del control.
Enable	Obtiene o establece un valor que indica si el control puede responder a la interacción del usuario.
SizeMode	Indica cómo se muestra la imagen.
Visible	Obtiene o establece un valor que indica si este elemento está visible en la interfaz de usuario (UI).

C#

```
using System.IO;

//Transforma un imagen en un array de byte
public static byte[] ImageToByteArray(System.Drawing.Image imagen)
{
    MemoryStream ms = new MemoryStream();
    imagen.Save(ms, System.Drawing.Imaging.ImageFormat.Jpeg);
    return ms.ToArray();
}

//Transforma un array de byte en una imagen
public static Image ByteArrayToImage(byte[] byteArrayIn)
{
    MemoryStream ms = new MemoryStream(byteArrayIn);
    Image returnImage = Image.FromStream(ms);
```

[4] Para el manejo de las imágenes se requirieron métodos adicionales, que permiten convertir las imágenes en byte y viceversa

```
            return returnImage;
        }

        //Transforma un imagen en un array de byte desde la ruta del archivo
        public static byte[] ObtenerByteArrayDeArchivo(string ruta_archivo)
        {
            FileStream fs = new FileStream(ruta_archivo, FileMode.Open, FileAccess.Read);
            byte[] ImageData = new byte[fs.Length];
            fs.Read(ImageData, 0, System.Convert.ToInt32(fs.Length));
            fs.Close();
            return ImageData;
        }
```

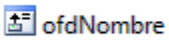
Manejo de datos

```
C#
//Creación de variable de tipo array de byte
byte[] foto;

//Ruta física del archivo
string ruta = @"C:\Imagen\foto.jpg";

//Transforma el archivo en un array de byte
foto = ObtenerByteArrayDeArchivo(ruta);

//Transforma el archivo en un array de byte desde una imagen del PictureBox
foto = ImageToByteArray(pbNombre.Image);

//Transforma el array de byte en una imagen
pbNombre.Image = ByteArrayToImage(foto);
```

Eventos

Para este proyecto no es usado ninguno de sus eventos.

OpenFileDialog (Clase)



Descripción

Esta clase permite comprobar si existe un archivo y abrirlo. Para identificar un OpenFileDialog es recomendable usar el prefijo “ofd” al nombre (ofdNombre).

Propiedades

Propiedad	Descripción
FileName	Obtiene o establece una cadena que contiene el nombre de archivo seleccionado en el cuadro de diálogo de archivo.
Name	Obtiene o establece el nombre del control.

Manejo de datos

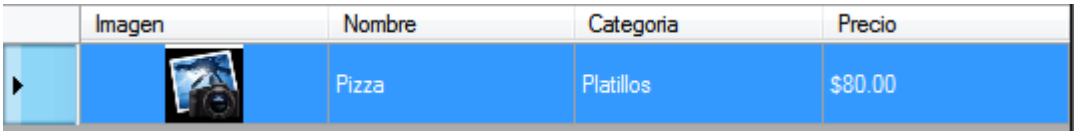
```
C#
private void ObtenerRutaArchivo() {

//Abrimos la ventana de búsqueda
ofdNombre.ShowDialog(this);

//Creación de variable de tipo array de byte
byte[] foto;

//Ruta física del archivo
```

```
string ruta = ofdNombre.FileName;

//Si se selecciono algún archivo
if (ofdNombre.FileName != "")
{
//Transforma el archivo en un array de byte
foto = ObtenerByteArrayDeArchivo(ruta);
}

}
```

Evento

Evento	Descripción	Actor
FileOk	Se desencadena cuando el usuario hace clic en el botón Abrir o Guardar de un cuadro de diálogo de archivo.	Usuario

DataGridView (Clase)



Descripción

El control **DataGridView** proporciona Una tabla personalizable mostrar Datos párr. La Clase **DataGridView** celdas Personalizar permite, Filas, Columnas y Bordes. El control es muy útil para desplegar un listado de objetos, con el fin de visualizar los datos más destacados de dicho objeto y poder seleccionarlo.

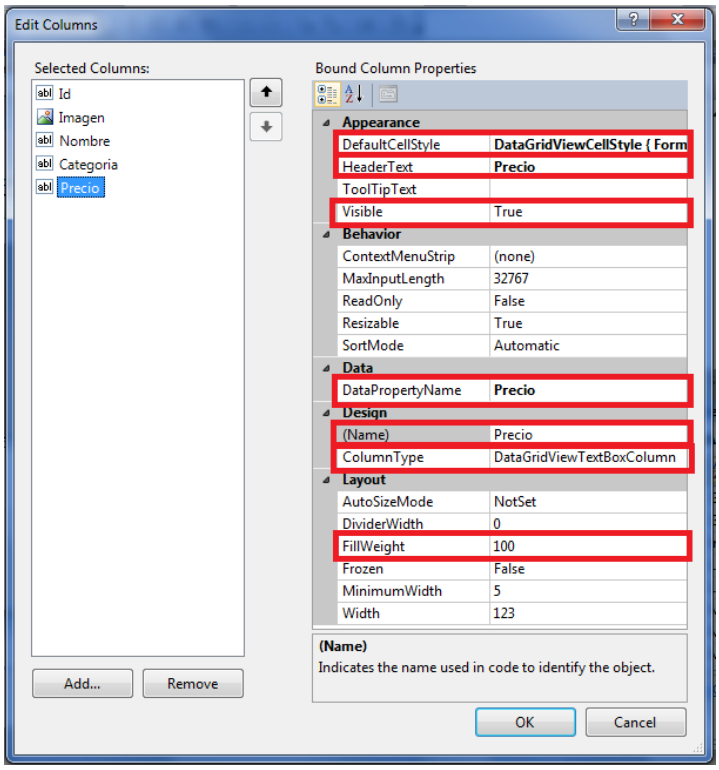
Propiedades

Propiedad	Descripción
AutoGenerateColumns	Obtiene o establece un valor que indica si se crean columnas automáticamente cuando se establece la propiedad DataSource o DataMember.
AutoSizeColumnsMode	Obtiene o establece un valor que indica cómo se determina el ancho de las columnas.
Columns	Obtiene una colección que contiene todas las columnas del control.
CurrentRow	Obtiene la fila que contiene la celda actual.
DataSource	Obtiene o establece el origen de datos cuyos datos se están mostrando en el control DataGridView.
Dock	Obtiene o establece que los bordes del control se acoplarán a su control principal y determina cómo se cambia el tamaño de un control con su elemento primario.
MultiSelect	Obtiene o establece un valor que indica si el usuario puede seleccionar a la vez varias celdas, filas o columnas del control DataGridView.
Name	Obtiene o establece el nombre del control.
ReadOnly	Obtiene un valor que indica si el usuario puede editar las celdas del control DataGridView.

RowTemplate	Obtiene o establece la fila que representa la plantilla de todas las filas del control.
SelectionMode	Obtiene o establece un valor que indica cómo se pueden seleccionar las celdas de DataGridView.
TabIndex	Obtiene o establece el orden de tabulación del control en su contenedor.
Text	Reemplazado. Obtiene o establece el texto asociado al control.

Columns

Para editar que columnas podremos visualizar de un listado y con qué formato, existe la propiedad Columns que nos permite por medio de una interfaz personalizar el grid.



DataGridViewColumn (Propiedades)

Propiedad	Descripción
ColumnType	Indica el tipo de componente que tendrá cada celda del grid según sea el tipo de dato que se desee visualizar.
DataPropertyName	Obtiene o establece El Nombre de la Columna de la Base de Datos o la Propiedad del origen de Datos al estilo de Que se Enlaza DataGridViewColumn .
DefaultCellStyle	Obtiene o establece el estilo de celda predeterminado que se va a aplicar a las celdas del control DataGridView si no se establece ninguna otra propiedad de estilo de celda.

DefaultCellStyle	Obtiene o establece El Estilo de Celda Predeterminado Que se va aplicar una A Las celdas del control DataGridView si no sí establece Ninguna Otra Propiedad de Estilo de Celda.
FillWeight	Obtiene o establece un valor que representa el ancho de la columna cuando se encuentra en modo de relleno, respecto del ancho de las demás columnas del control que estén en modo de relleno.
HeaderText	Obtiene o establece el texto de título en la celda de encabezado de columna.
Name	Obtiene o establece el nombre de la columna.
ReadOnly	Obtiene un valor que indica si el usuario puede editar las celdas del control DataGridView.
RowTemplate	Obtiene o establece la fila que representa la plantilla de todas las filas del control.
SelectionMode	Obtiene o establece un valor que indica cómo se pueden seleccionar las celdas de DataGridView.
TabIndex	Obtiene o establece el orden de tabulación del control en su contenedor.
Text	Reemplazado. Obtiene o establece el texto asociado al control.
Visible	Obtiene o establece un valor que indica si se muestran el control y todos sus controles primarios.

Manejo de datos

C#

```
private void CargarDatos()
        {
            //Se llena una lista con los obejetos y las propiedades
            //que se visualizan
            var lista = from c in Session.Query<ArticuloClass>()
                        where c.Activo
                        select new {
                        c.Id,
                        c.Imagen,
                        c.Nombre,
                        Categoria = c.Categoria.Nombre,
                        c.Precio
                        };

            //Se limpia el Grid
            grdArticulos.DataSource = null;
            //Se asignan el listado al Grid
            grdArticulos.DataSource = lista.ToList();
        }

 //Cuando se llama el evento doble clic de la celda
        private void grdArticulos_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
        {
             //Se obtiene la fila en selección
             DataGridViewRow registro = grdArticulos.CurrentRow;
             //Se obtiene el valor de la celda con el nombre de Id
             int IdRegistro = (int)registro.Cells["Id"].Value;
        }
```

Evento

Evento	Descripción	Actor
CellClick	Se produce cuando el contenido dentro de una celda se hace clic.	Usuario
CellDoubleClick	Se produce cuando el usuario hace doble clic en cualquier parte de una celda.	Usuario

Definir las propiedades de las interfaces

El definir las propiedades ayudará a que el diseñador configure la interfaz al momento de estarla elaborando y a su vez al programador a realizar de forma más sencilla la programación. Las propiedades que se definen no necesariamente tienen que ser todas, si no las que en su momento el programador va hacer uso.

De este modo el programador ya tendrá una guía de las interfaces y podrá programarlas con mayor facilidad.

Para especificar los detalles de las propiedades es recomendable poner como encabezado el nombre del control, seguidamente todos los controles que sean similares comenzando con su nombre y finalmente un listado de las propiedades que se necesitan del control.

 Propiedades de la interfaz Empleado[5]

Windows Form

1. Name : FrmEmpleadoCaptura
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

TextBox

2. Name : txtNombre
    a. Tabindex : 0
3. Name : txtDireccion
    a. TabIndex : 1

[5] La definición de las propiedades las interfaces faltantes se encuentran el Anexo A

4. Name : txtColonia
    a. TabIndex : 2
5. Name : txtCodigoPostal
    a. TabIndex : 3
6. Name : txtTelefono
    a. TabIndex : 4
7. Name : txtEmail
    a. TabIndex : 5
8. Name : txtUsuario
    a. TabIndex : 10
    b. CharacterCasing : Upper
9. Name : txtPassword
    a. TabIndex: 11
    b. CharacterCasing : Upper

DateTimePicker

10. Name : dtpNacimiento
    a. Format : Short
    b. TabIndex : 6

ComboBox

11. Name : cboTiposEmpleados
    a. TabIndex : 7
    b. Text : Seleccionar

PictureBox

12. Name : pictureBox
    a. SizeMode : Zoom
    b. Image : A consideración

OpenFileDialog

13. Name : openFileDialog
    a. FileName : openFileDialog

Button

14. Name : btnBuscarTipoEmpleado
    a. TabIndex : 8
    b. Width : 24
    c. Height : 24
    d. Image : A consideración
15. Name : btnAgregarTipoEmpleado
    a. TabIndex : 9
    b. Width : 24
    c. Height : 24
    d. Image : A consideración

16. Name : btnExaminar
    a. Text : &Examinar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 12
    d. Width : 100
    e. Height : 40
    f. Image : A consideración
17. Name : btnGuardar
    a. Text : &Guardar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 13
    d. Width : 100
    e. Height : 40
    f. Image : A consideración
18. Name : btnCancelar
    a. Text : &Cancelar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 14
    d. Width : 100
    e. Height : 40
    f. Image : A consideración

Abrir conexión con Fluent NHibernate

Para poder editar los objetos con Fluent NHibernate es necesario abrir una sesión con la base de datos y obtener una copia fidedigna de los datos almacenados en ella.

Es recomendable abrir la sesión cada vez que una interfaz cargue y antes de realizar cualquier operación, debe asegurarse de no abrir varias sesiones o de lo contrario no podrá ver los cambios realizados en cada una de ellas (Código 3).

C#
```
//Se crea un objeto ISession público o privado para usarlo en toda la clase
private ISession Session;

public FrmCorte()
{
       InitializeComponent();
}

private void FrmEmpleadoCaptura_Load (object sender, EventArgs e)
{
      //Se abre la conexión
      Session = Conexion.CreateSessionFactory().OpenSession();
}
```

Código 3. Abrir sesión

Crear un nuevo objeto

Para crear un nuevo objeto es necesario crear una nueva instancia del objeto como se muestra en el (Código 4).

```
C#
public ISession Session;
public EmpleadoClass Empleado;

private void FrmEmpleadoCaptura_Load(object sender, EventArgs e)
{
   Session = Conexion.CreateSessionFactory().OpenSession();
   NuevoEmpleado();
}

private void NuevoEmpleado()
{
   Empleado = new EmpleadoClass();
}
```

Código 4. Creación de la instancia de un objeto

Cargar un objeto

Si el objeto ya existe es necesario cargarlo a la sesión haciendo referencia al campo llave, en este caso es el Id como se muestra en el (Código 5).

```
C#
public ISession Session;
public EmpeladoClass Empleado;
private int IdRegistro = 1;

private void FrmEmpleadoCaptura_Load(object sender, EventArgs e)
{
   Session = Conexion.CreateSessionFactory().OpenSession();
   CargarEmpleado();
}

private void CargarEmpleado()
{
   Empleado = Session.Get<EmpleadoClass>(IdRegistro);
}
```

Código 5. Cargar un objeto

Editar datos de un objeto con los controles

Para poder llevar a cabo la edición de los datos de un objeto es necesario hacer uso de los controles, para eso se deberán cargar en ellos los datos actuales que tenga el objeto como se muestra en el (Código 6).

```
C#
public ISession Session;
public EmpleadoClass Empleado;
private bool nuevo;

public FrmEmpleadoCaptura()
{
   InitializeComponent();
}

private void FrmEmpleadoCaptura_Load(object sender, EventArgs e)
{
   Session = Conexion.CreateSessionFactory().OpenSession();

   Empleado = Session.Get<EmpleadoClass>(1);

   CargarDatos();
}

private void CargarDatos()
{
            //TextBox
```

```
            txtNombre.Text = Empleado.Nombre;
            txtDireccion.Text = Empleado.Direccion;
            txtColonia.Text = Empleado.Colonia;
            txtCodigoPostal.Text = Empleado.CodigoPostal;
            txtTelefono.Text = Empleado.Telefono;
            txtEmail.Text = Empleado.Email;
            txtUsuario.Text = Empleado.Usuario;
            txtPassword.Text = Empleado.Password;

            //DateTimePicker

            dtpNacimiento.Value = Empleado.FechaNacimiento;

            //ComboBox; Si el valor de Tipo empleado es diferente de null carga el id
            //de lo contrario regresa el valor de Null
            cboTiposEmpleados.SelectedValue = Empleado.TipoEmpleado != null ?
(object)Empleado.TipoEmpleado.Id :
          null;

            //PictureBox; si el valor de la imagen es diferente de Null
            //se carga la imagen, de lo contrario se carga una imagen por defecto
            if (Empleado.Imagen != null)
                pictureBox.Image = UtilidadesImagenes.ByteArrayToImage(Empleado.Imagen);
            else
                pictureBox.Image = Properties.Resources.image_default;
}
```

Código 6. Editar datos del objeto

Actualizar datos del objeto

Después de la edición con los datos con los controles se deben actualizar los nuevos valores del objeto como se muestra en el (Código 7).

```
C#
private bool ActualizarDatos()
{
Empleado.Nombre = txtNombre.Text;
Empleado.Direccion = txtDireccion.Text;
Empleado.Colonia = txtColonia.Text;
Empleado.CodigoPostal = txtCodigoPostal.Text;
Empleado.Telefono = txtTelefono.Text;
Empleado.Email = txtEmail.Text;
Empleado.Usuario = txtUsuario.Text;
Empleado.Password = txtPassword.Text;

Empleado.FechaNacimiento = dtpNacimiento.Value;

Empleado.TipoEmpleado = Session.Load<TipoEmpleadoClass>(cboTiposEmpleados.SelectedValue);

Empleado.Imagen = UtilidadesImagenes.ObtenerByteArrayDeArchivo(openFileDialog.FileName);
}
```

Código 7. Actualizar datos del objeto

Guardar los cambios del objeto

Para guardar los cambios realizados de un objeto es necesario usar el método Session.Save(objeto); pero este punto la creación y/o edición de un objeto únicamente se ha realizado en la sesión y no ha sido actualizada en la base de datos. Para enviar los datos a la base de datos se necesita la función Session.Flush(); como se muestra en el (Código 8).

```
C#
private bool ActualizarDatos()
{
Empleado.Nombre = txtNombre.Text;
Empleado.Direccion = txtDireccion.Text;
Empleado.Colonia = txtColonia.Text;
Empleado.CodigoPostal = txtCodigoPostal.Text;
Empleado.Telefono = txtTelefono.Text;
Empleado.Email = txtEmail.Text;
```

```
Empleado.Usuario = txtUsuario.Text;
Empleado.Password = txtPassword.Text;

Empleado.FechaNacimiento = dtpNacimiento.Value;

Empleado.TipoEmpleado = Session.Load<TipoEmpleadoClass>(cboTiposEmpleados.SelectedValue);

Empleado.Imagen = UtilidadesImagenes.ObtenerByteArrayDeArchivo(openFileDialog.FileName);

//Únicamente guarda los datos modificados en la sesión
Session.Save(Empleado);

//Actualiza la sesión con la base de datos
Session.Flush();
}
```

Código 8. Guardar los cambios del objeto

Convención de código utilizado[6]

Para una mejor comprensión del código se recomienda apegarse a las siguientes normativas:

- Ningún nombre de variable comienza con número.
- Los nombres de las variables y de métodos serán escritos de forma CamelCase, que es la práctica de la escritura de palabras compuestas o frases en las que los elementos se unen sin espacios , con la primera letra de cada elemento en mayúsculas, como en " LaBelle ", BackColor," McDonalds "o" iPod ".
- Las variables declaradas públicas deberán comenzar con mayúsculas y las variables declaradas privadas deberán iniciar con minúsculas.
- Los nombres de los métodos deberán ser descriptivos de acuerdo a su funcionamiento. Por ejemplo: un método para guardar información debería ser Guardar(); y no Enviar(); ya que de esta manera quien lea el código podrá tener una idea general de cómo funciona el método al momento de ser llamado.

Reportes

Un reporte permitirá extraer información de la base de datos por medio de una consulta específica y mostrársela al usuario de forma legible para poder imprimirla.

Crear un reporte

Para crear un reporte en Visual Studio, se puede acceder con la combinación de teclas CTRL+ATL+A para abrir la ventana de agregar un elemento, en la sección de Reportes se debe seleccionar la opción de Report, se asigna un nombre y pulsa el botón de agregar como se muestra en Figura 47.

[6] La documentación del código de las interfaces se encuentran el Anexo A

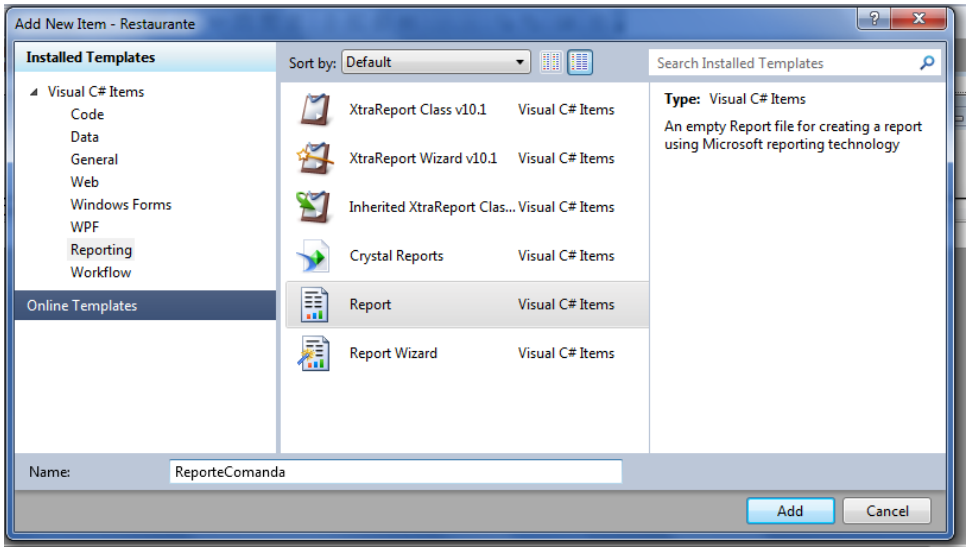


Figura 47. Creación de un reporte

Partes de un reporte

Cuando un reporte es creado lo primero que se ve es el cuerpo del documento donde se enseñara la información recabada, pero no es la única parte del reporte. Un reporte está dividido en tres partes el encabezado de página, el cuerpo del documento y el pie de página (Figura 48).

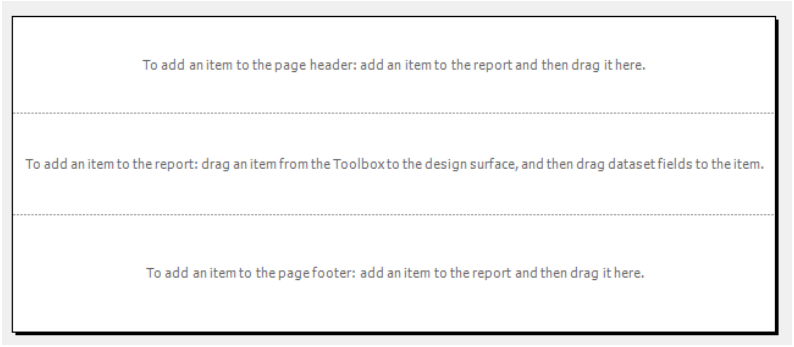


Figura 48. Secciones del reporte

Para agregar el encabezado o pie de página al reporte se debe pulsar con el botón secundario del ratón sobre el cuerpo del documento y en menú contextual dentro de la opción Insertar se encuentran amabas opciones (Figura 49).

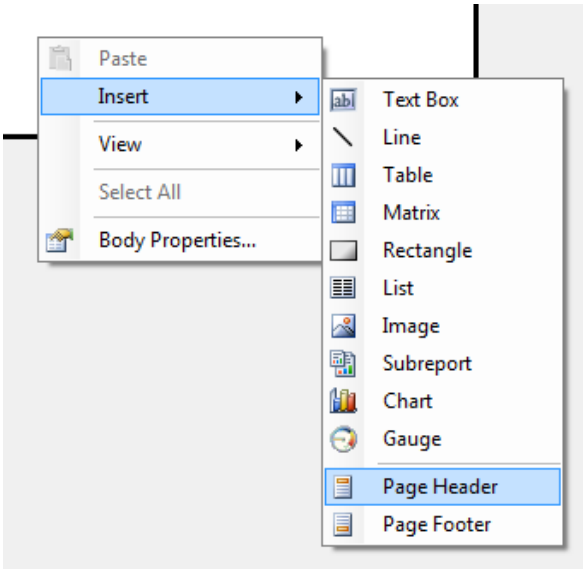


Figura 49. Agregar encabezado y pie de página al reporte

El encabezado y pie de página del reporte son partes fijas, que los datos puestos en esas secciones no varían en caso de que el reporte se extienda a más de una página, en cambio la sección de contenido es un área dinámica que toda la información desplegada en ella será variable.

Despliegue de información en un reporte

Para poder visualizar una gran cantidad de datos es necesario insertar una tabla dentro del cuerpo del documento. El control tabla se encuentra en la barra de herramienta y al colocarlo sobre el reporte se abrirá una venta para configurar un DataSet. Un DataSet es una memoria caché de los datos recuperados de un origen de datos, en este caso de la base de datos (Figura 50).

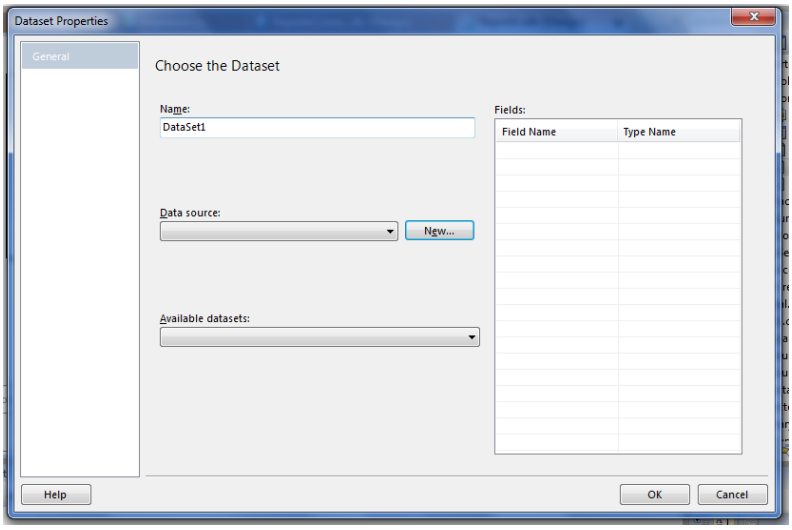


Figura 50. DataSet

Como ya se había dicho un DataSet requiere de datos, para eso es necesario definirle el origen de los datos, para ello pulse el botón de Nuevo de la sección DataSource para que se abra una ventana para seleccionar el tipo del origen de los datos. Para este caso es Base de Datos (Figura 51).

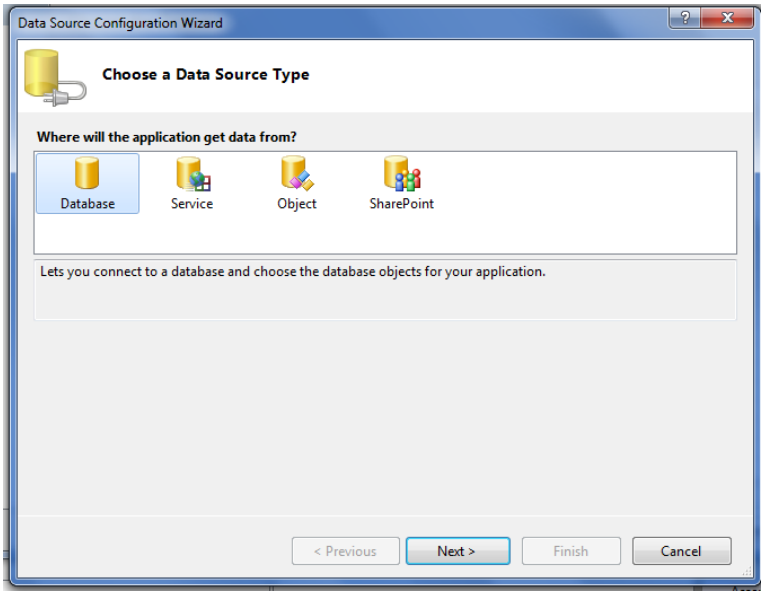


Figura 51. Origen de los datos del DataSet

Después de seleccionar el origen de los datos, se selecciona el modelo de la base de datos, que como ya se había comentado es un DataSet (Figura 52).

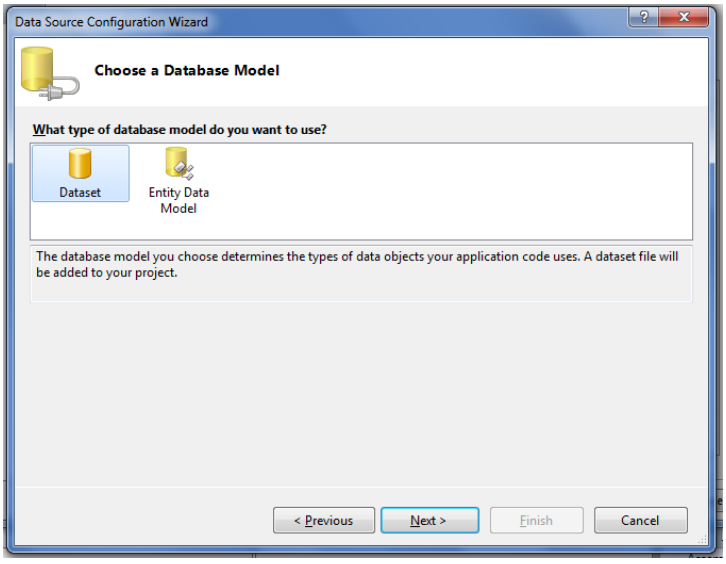


Figura 52. Modelo de la base de datos

El paso siguiente es elegir una conexión a nuestra base de datos, que es la que ya previamente se había creado (Figura 53).

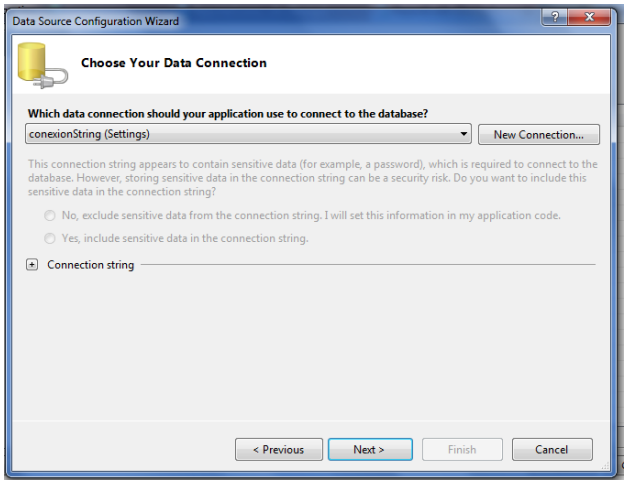


Figura 53. Seleccionar conexión para el DataSet

Finalmente se seleccionaran todas las tablas que contengan los datos que se vayan a utilizar (Figura 54).

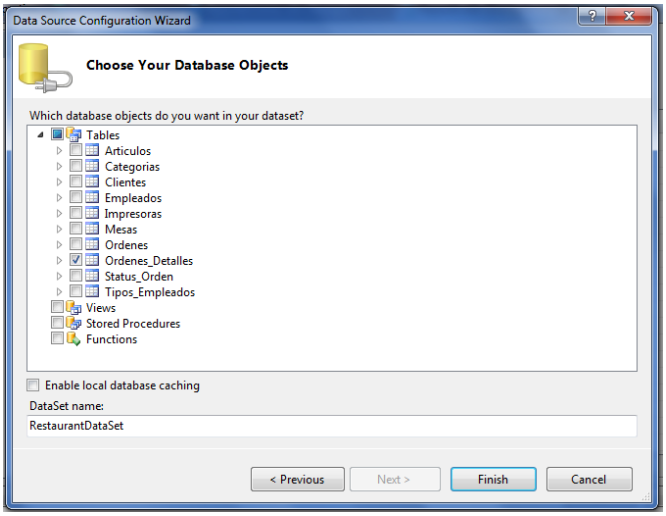


Figura 54. Selección de datos del DataSource

Una vez creado el DataSource se selecciona el que vaya ser usado ya que puede haber varios y del mismo modo seleccionamos la tabla principal de consulta (Figura 55).

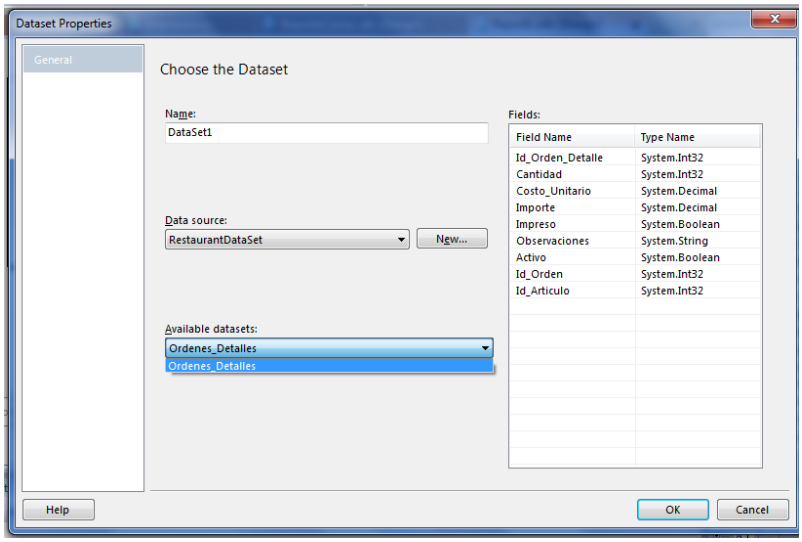


Figura 55. Configuración del DataSet

Configurar tabla

Una vez creado el DataSet se creará una tabla dentro del documento donde se elegirán los campos que se deseen enseñar (Figura 56).

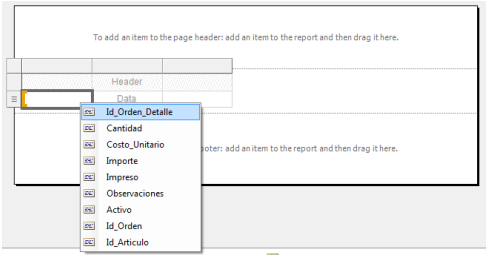


Figura 56. Tabla del reporte

Mediante la ventana de datos del reporte veremos los datos disponibles en nuestra consulta y que podrán arrastrarse dentro de nuestro encabezado o pie de página para mostrar un dato en específico (Figura 57).

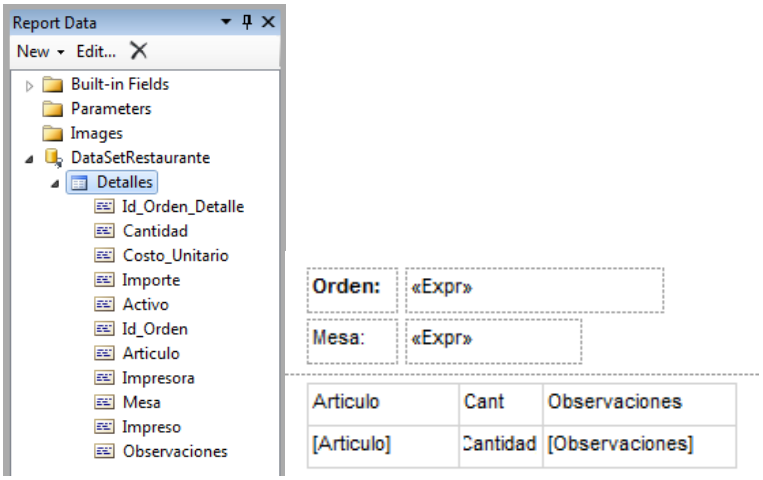


Figura 57. Datos del reporte

Filtrado de datos para el reporte

Un DataSet despliega todos los datos contenidos en la tabla que se va a consultar. Para filtrar los datos que se desean mostrar es necesario crear variables dentro del DataSet, para ello se hace doble clic sobre el DataSet que se creó, una vez adentro se hace clic con el botón secundario del ratón sobre la tabla que fue elegida como principal y se selecciona la opción de Configurar (Figura 58).

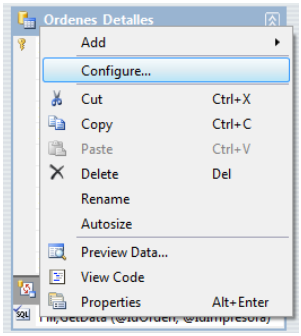


Figura 58. Configurar la consulta de la tabla

En la ventana de configuración se encuentra la consulta que se realiza a la base de datos la cual puede ser editada para obtener un mejor resultado. Para especificar variables en la consulta se usa el carácter de @ antes del nombre de la variable (Figura 59).

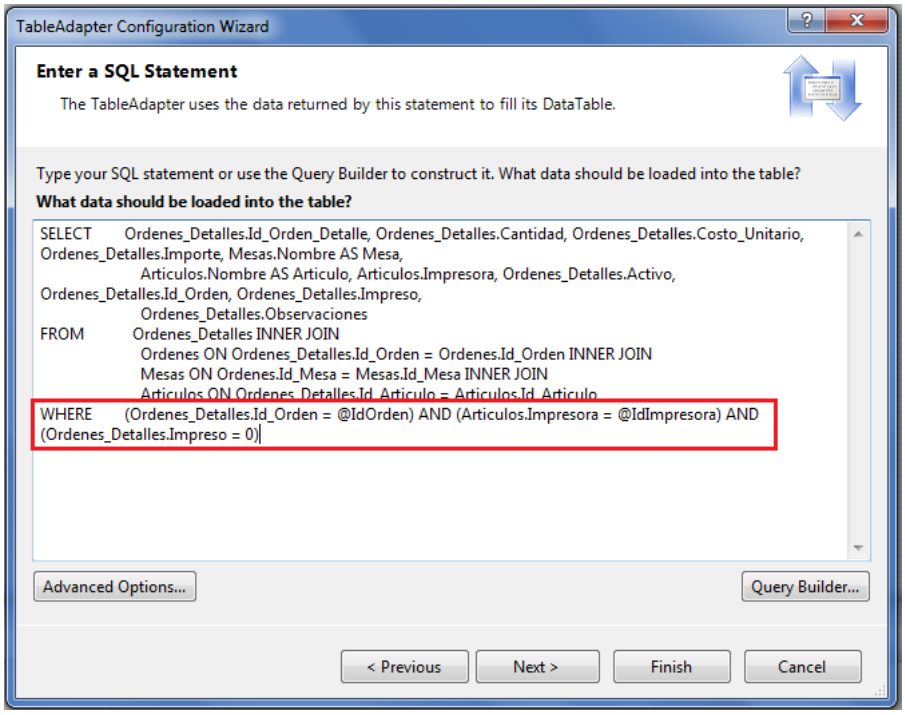


Figura 59. Editar la consulta del DataSet

Una vez editada la consulta se puede apreciar en el método Fill del TableAdapter las variables que necesita para realizar el filtrado (Figura 60).

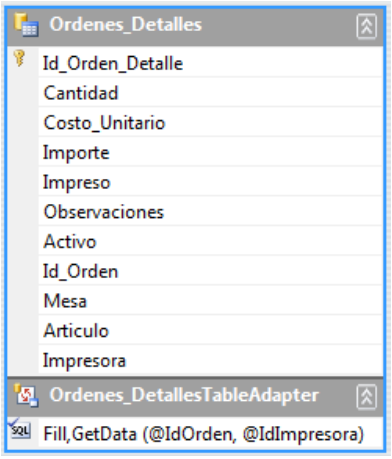


Figura 60. Método Fill del TableAdapter

Visualizar el reporte

Para poder visualizar el reporte es necesario crear un formulario e insertar dentro el control ReportViewer. En el ReportViewer se selecciona el reporte que se va visualizar ya que puede haber varios (Figura 61).

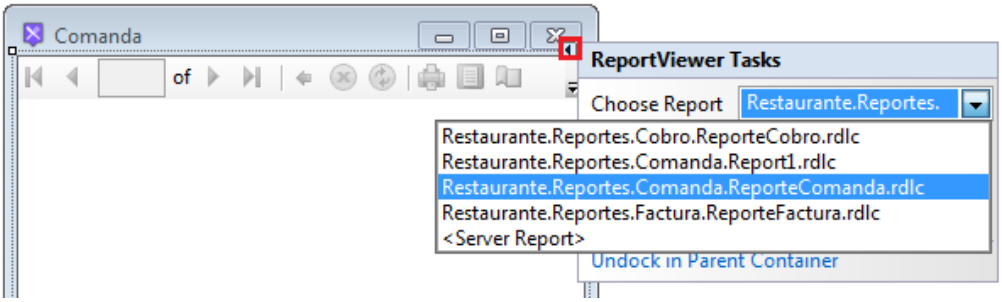


Figura 61. Seleccionar el reporte

Después de seleccionar el reporte se debe seleccionar el DataSource que permitirá llenar el reporte (Figura 62).

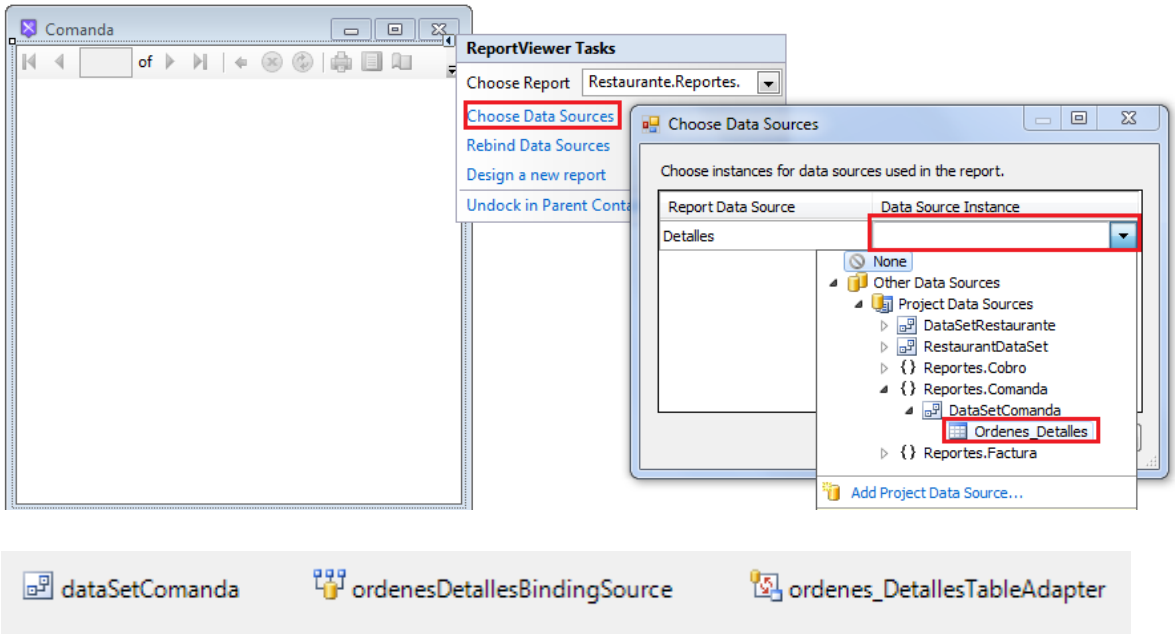


Figura 62. Seleccionar el DataSource

Finalmente dentro del evento cargar del formulario se llama el método Fill del TableAdapter donde se pasan los parámetros de búsqueda que fueron definidos y se refresca el ReportViewer como se muestra en el Código 9.

C#

```
using System;
using System.Windows.Forms;

namespace Restaurante.Reportes.Comanda
{
    public partial class FrmReporteComanda : Form
    {
        public int IdOrden;
        public int IdImpresora;

        public FrmReporteComanda()
        {
            InitializeComponent();
        }

        private void FrmReporteComanda_Load(object sender, EventArgs e)
```

```
        {
            this.ordenes_DetallesTableAdapter.Fill(this.dataSetComanda.Ordenes_Detalles, IdOrden, IdImpresora);
            this.reportViewer1.RefreshReport();
        }
    }
}
```

Código 9. Llamar el método Fill

Imprimir automáticamente un reporte

Para imprimir un reporte de manera automática no existe un método que lo realice de forma sencilla, para ello es necesaria la creación de una clase que permita realizar esta tarea (Código 10).

C#

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing.Imaging;
using System.Drawing.Printing;
using System.IO;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Datos;
using Microsoft.Reporting.WinForms;
using NHibernate;
using Restaurante.DataSetRestauranteTableAdapters;

namespace Restaurante.Reportes.Comanda
{
    public class Impresion{

        private int mCurrentPageIndex;
        private IList<Stream> mStreams;

        private Stream CreateStream(string name,
                                    string fileNameExtension, Encoding encoding,
                                    string mimeType, bool willSeek)
        {

            Stream stream = new MemoryStream();
            mStreams.Add(stream);
            return stream;
        }

        // Export the given report as an EMF (Enhanced Metafile) file.
        private void Export(LocalReport report)
        {
            string deviceInfo =
                "<DeviceInfo>" +
                "  <OutputFormat>EMF</OutputFormat>" +
                "  <PageWidth>7.9cm</PageWidth>" +
                "  <PageHeight>0cm</PageHeight>" +
                "  <MarginTop>0.0cm</MarginTop>" +
                "  <MarginLeft>0.0cm</MarginLeft>" +
                "  <MarginRight>0.0cm</MarginRight>" +
                "  <MarginBottom>0.0cm</MarginBottom>" +
                "</DeviceInfo>";
            Warning[] warnings;
            mStreams = new List<Stream>();
            report.Render("Image", deviceInfo, CreateStream,
                          out warnings);
            foreach (Stream stream in mStreams)
                stream.Position = 0;
        }

        // Handler for PrintPageEvents
        private void PrintPage(object sender, PrintPageEventArgs ev)
        {
            Metafile pageImage = new
                Metafile(mStreams[mCurrentPageIndex]);
            ev.Graphics.DrawImage(pageImage, ev.PageBounds);
            mCurrentPageIndex++;
```

```
            ev.HasMorePages = (mCurrentPageIndex < mStreams.Count);
        }

        private void Print(string printerName)
        {
            /*const string printerName =
               "Microsoft XPS Document Writer";*/
            if (mStreams == null || mStreams.Count == 0)
                return;
            PrintDocument printDoc = new PrintDocument();
            printDoc.PrinterSettings.PrinterName = printerName;
            printDoc.DefaultPageSettings.PaperSize = new PaperSize("A4",311, 600);
            if (!printDoc.PrinterSettings.IsValid)
            {
                string msg = String.Format(
                    "No se encuentra la impresora: \"{0}\".", printerName);
                MessageBox.Show(msg, "Error de impresión");
                return;
            }
            printDoc.PrintPage += PrintPage;
            printDoc.Print();
        }

        public void ImprimirTickets(int idOrden, ImpresoraClass impresora)
        {
            LocalReport report = new LocalReport();

            // Imprimir el ticket
            // Se crea una instancia del DataSet
            DataSetRestaurante ds = new DataSetRestaurante();
            // Se crea una instancia del TableAdapter
            Ordenes_DetallesTableAdapter adapter = new Ordenes_DetallesTableAdapter();

            ds.EnforceConstraints = true;
            // Se llama la function Fill del TableAdapter y se envían las variables
            adapter.Fill(ds.Ordenes_Detalles, idOrden, impresora.Id);
            // Se crea una instancia del DataSorce
            ReportDataSource nuevo = new ReportDataSource("Detalles");
            // Se llena asigna al DataSorce el DataSet
            nuevo.Value = ds.Ordenes_Detalles;
            // Se llena el reporte
            report.DataSources.Add(nuevo);
            // Se establece la localización del reporte
            report.ReportPath = @"..\..\Reportes\Comanda\ReporteComanda.rdlc";
            // Se exporta un reporte temporal
            Export(report);
            // Se inicializa la pagina
            mCurrentPageIndex = 0;
            // Se imprime el reporte en la impresora designada
            Print(impresora.Ruta);
        }

        public new void Dispose()
        {
            if (mStreams != null)
            {
                foreach (Stream stream in mStreams)
                    stream.Close();
                mStreams = null;
            }
        }
    }
}
```

Código 10. Clase para impresión

Para visualizar un reporte basta con abrir el formulario que lo contiene y enviar las variables que sean necesarias como se muestra en el Código 11.

```
C#
FrmReporteComanda frmReporteComanda = new FrmReporteComanda
   {
     IdOrden = Orden.Id,
     IdImpresora = IdImpresora
   };

frmReporteComanda.ShowDialog(this);
```

Código 11. Visualizar un reporte

Para imprimir de manea automática un reporte es necesario crear una instancia de la clase que se creó y llamar al método ImprimirTickets(int idOrden, ImpresoraClass impresora); y se le pasan las variables necesarias como se muestra en el Código 12.

```
C#
Impresion impresion = new Impresion();

impresion.ImprimirTickets(Orden.Id,x.Articulo.Impresora);
```

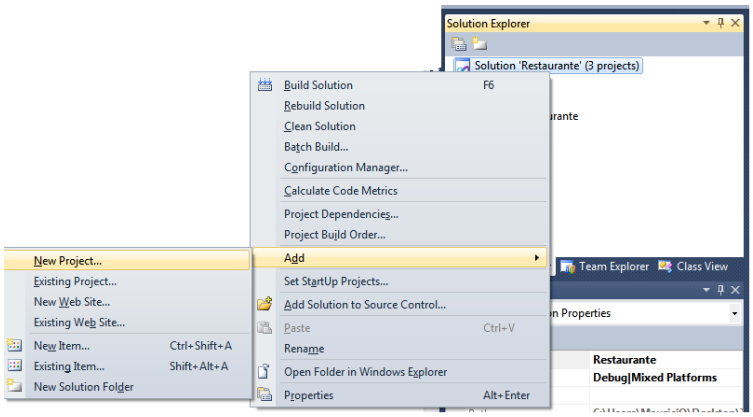
Código 12. Imprimir un reporte automáticamente

Creación del instalador

Para finalizar el proyecto es necesario crear un instalado para facilitar la instalación del software.

Crear un proyecto de instalación

Para crear un proyecto de instalación se debe pulsar sobre el nombre de la solución, seleccionar la opción de agregar nuevo proyecto.



En la ventana para agregar un nuevo proyecto en la sección otros tipos de proyecto, se selecciona Setup and Deployment, después Visual Studio Installer, Setup Wizard y finalmente se asigna un nombre.

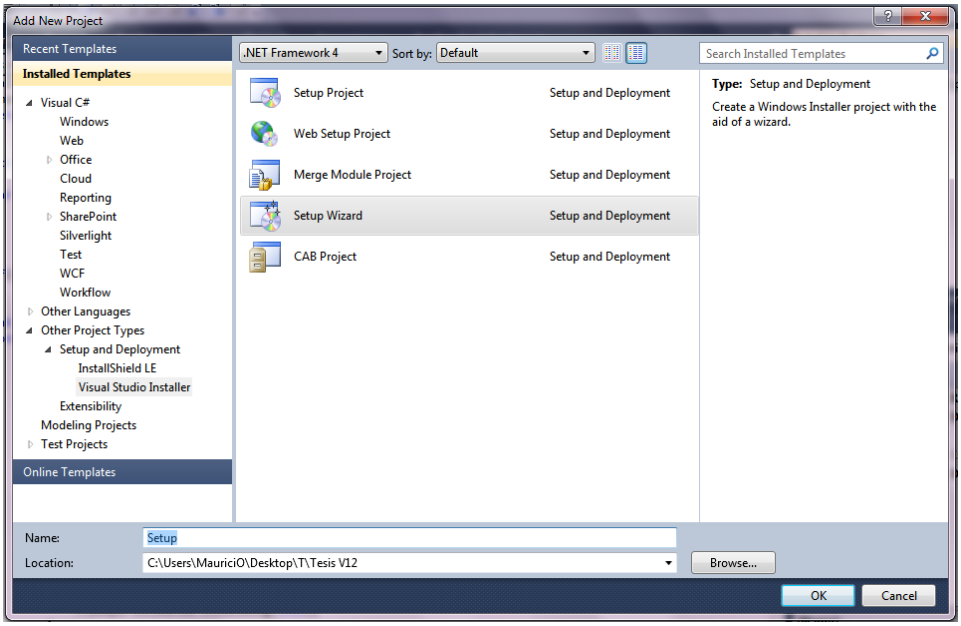


Figura 63. Creación del proyecto para instalación

En la primera interfaz es de bienvenida al instalador, pulse el botón de siguiente.

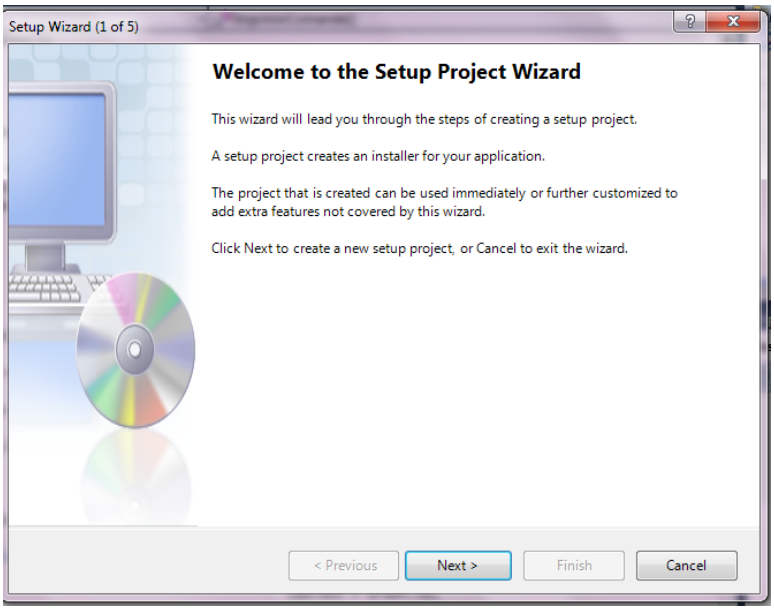


Figura 64. Interfaz 1 del Setup Wizard

En la segunda interfaz seleccione crear un instalador para aplicación de Windows y pulse el botón de siguiente.

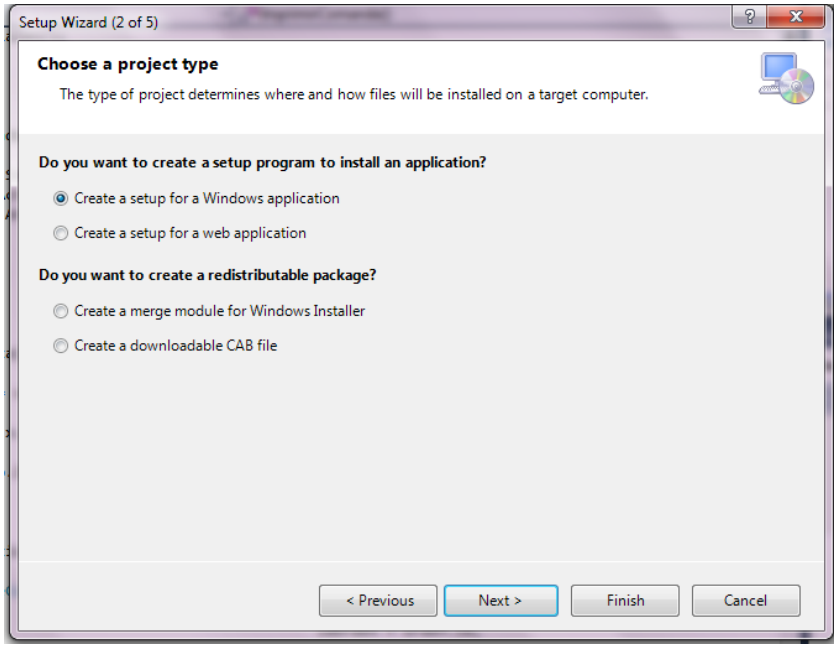


Figura 65. Interfaz 2 del Setup Wizard

En la tercera interfaz seleccione los archivos de salida primaria de cada uno de los proyectos y pulse Finalizar.

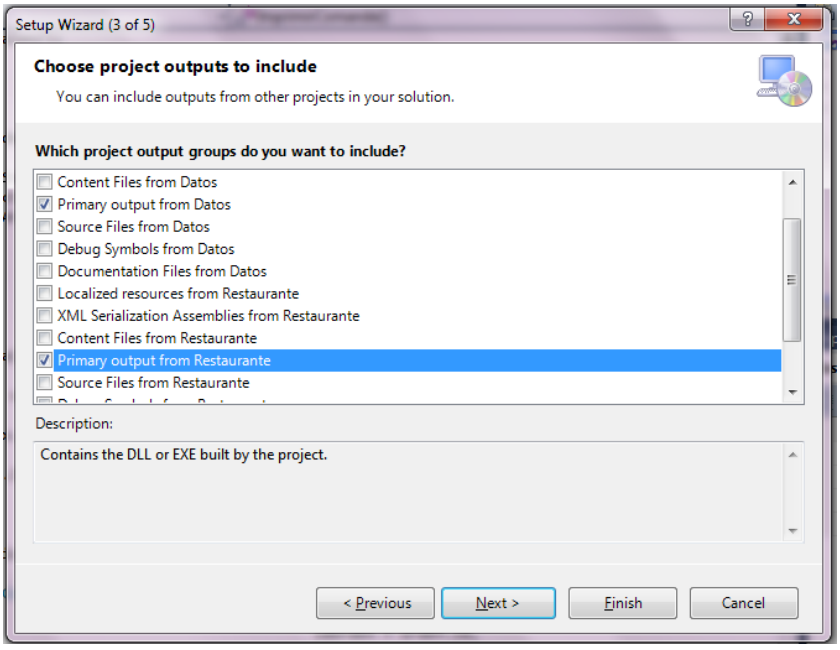


Figura 66. Interfaz 3 del Setup Wizard

Crea acceso directo en el escritorio

Para crear un icono después de la instalación en el escritorio se selecciona la carpeta de aplicación, en el listado se hace clic con el botón secundario sobre el archivo primario del proyecto de Windows y se selecciona crear acceso rápido.

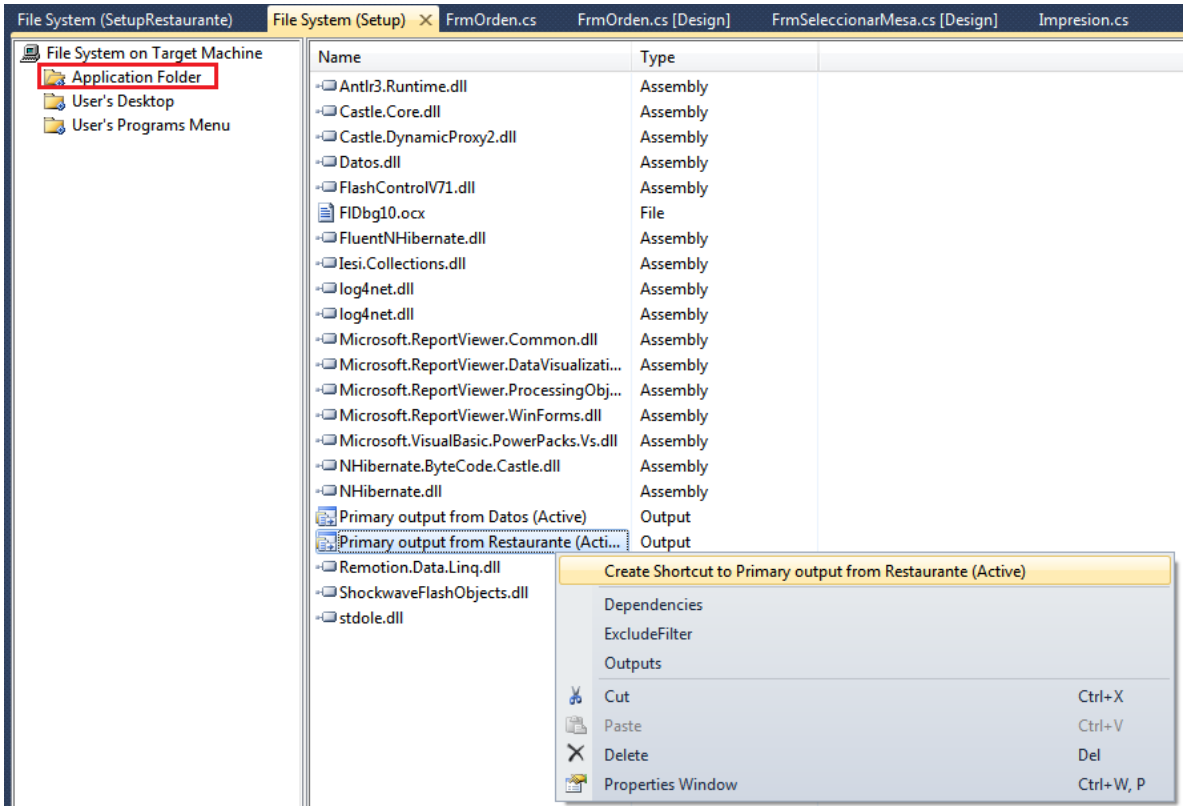


Figura 67. Carpeta aplicación

Se renombra el archivo y se copia dentro de la carpeta Escritorio de usuario.

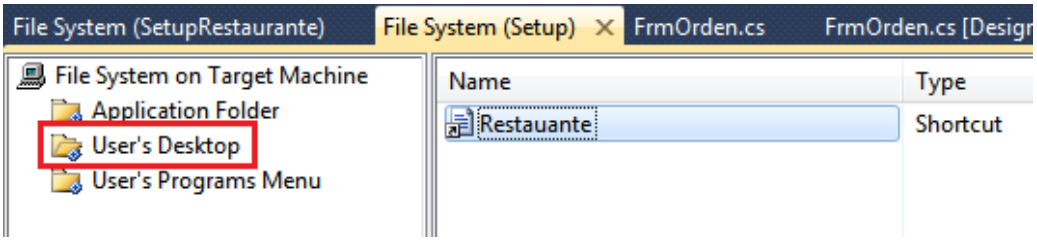


Figura 68. Carpeta escritorio de usuario

Establecer requisitos previos de instalación

Lo más recomendable para instalar un software es asegurarse de que el equipo donde va ser instalado cumpla con los requisitos necesarios para no tener ningún problema. Los requisitos de instalación forzaran a que si el equipo no cuenta con ellos no podrá ser instalado el software.

Para definir los requisitos de instalación acceda a las propiedades del proyecto de instalación y en la ventana emergente seleccione requisitos previos.

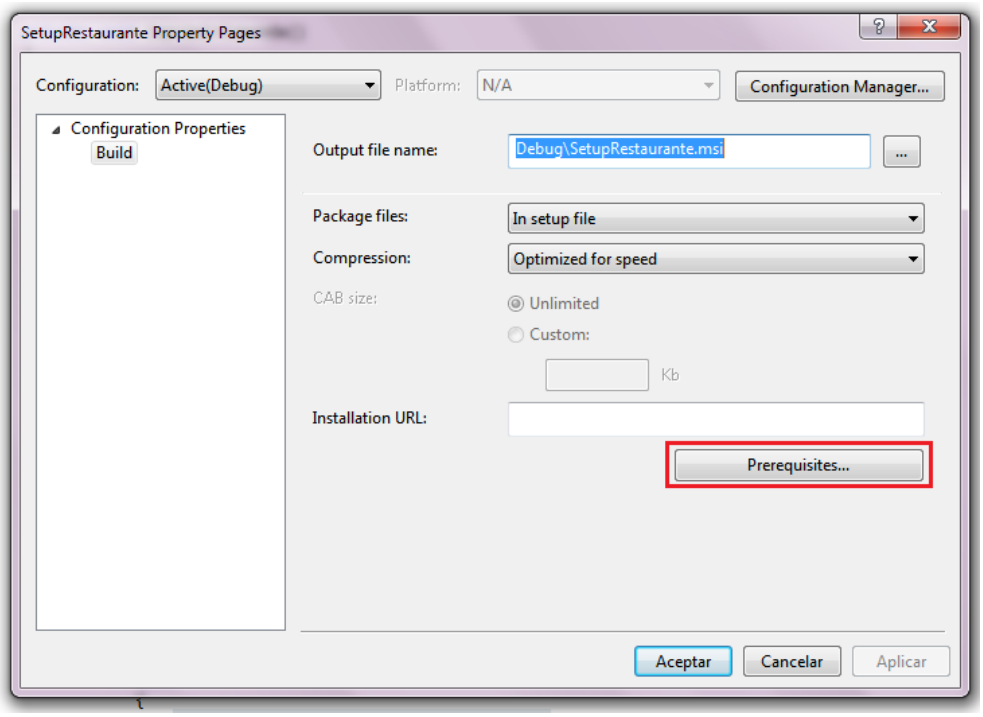


Figura 69. Propiedades de la instalación

En la interfaz de requisitos previos deben estar activados los campos de Figura 70:

- Microsoft .Net Framework 4 (x86 and x64), para verificar que tenga instalado el Framework necesario.
- SQL Server 2008 Express, para verificar que este instalado el servidor de SQL.
- Windows Installer 4.5, para verificar que tenga el instalador de Windows.
- Download prerequisites from the component vendor's web site, permitirá descargar los componentes faltantes desde la pagina del fabricante automáticamente.

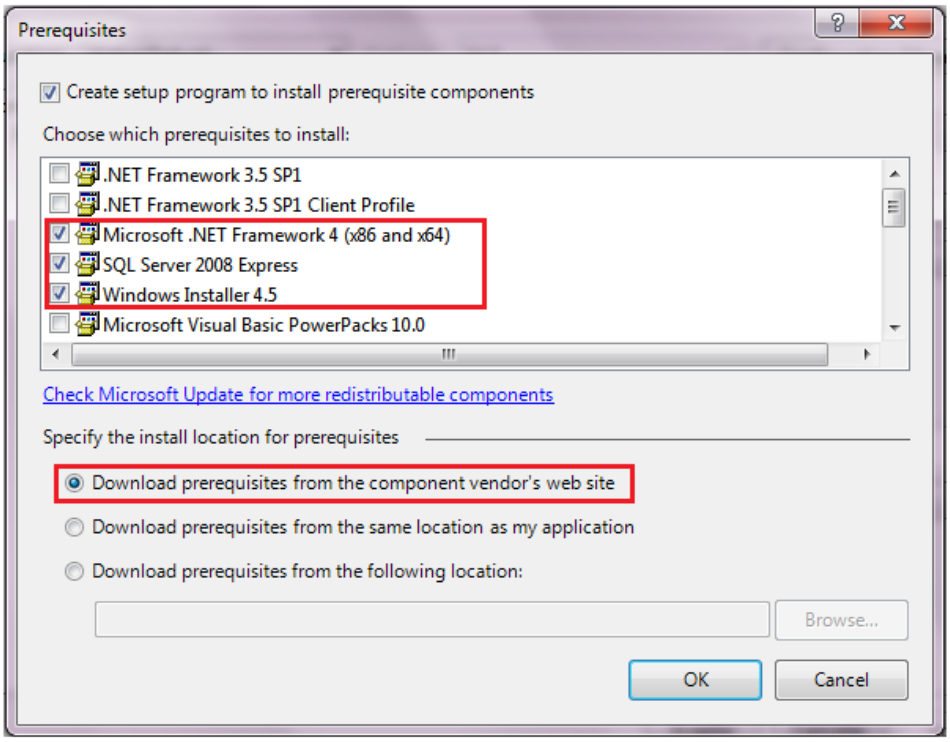


Figura 70. Requisitos previos

Una vez configuradas las propiedades se aceptan los cambios.

Propiedades de instalación

Para finalizar se pueden editar las propiedades del autor, el desarrollador, el nombre del producto, el titulo del proyecto y la versión del software (Figura 72).

Una vez editadas las propiedades finales se pulsa con el clic secundario sobre el proyecto de instalación y selecciona construir para generar el archivo dentro de la carpeta Debug del proyecto (Figura 71).

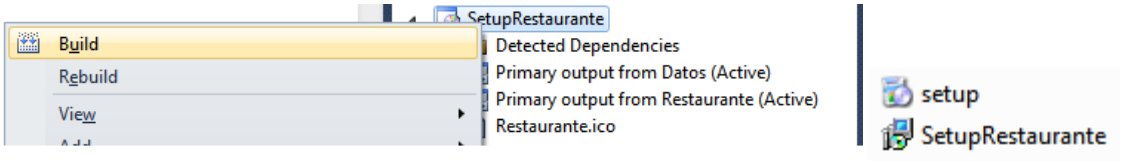


Figura 71. Construir el proyecto de instalación

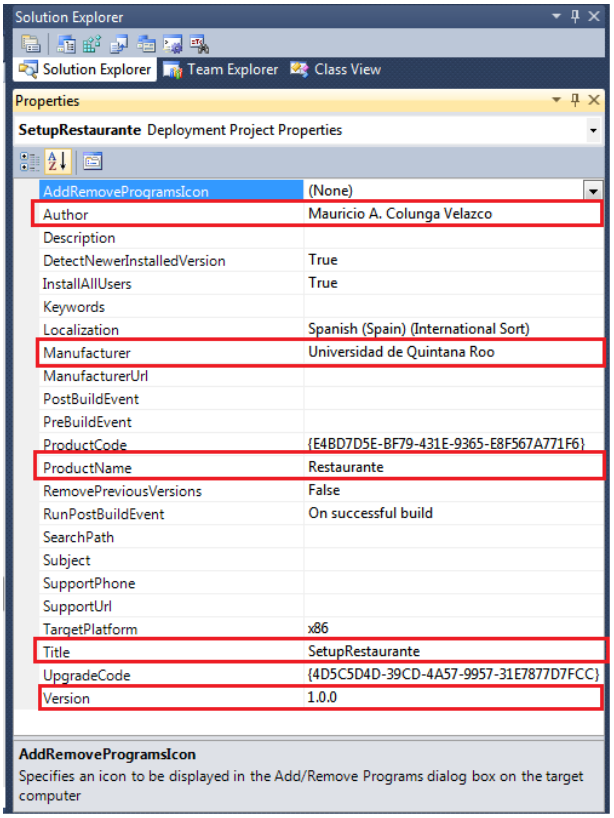


Figura 72. Propiedades del proyecto instalación

Etapa de prueba

Durante esta etapa se verifica el correcto funcionamiento del sistema de acuerdo al funcionamiento del sistema, en esta etapa deben ser evaluados todos los puntos. Si alguna prueba falla, deberá ser corregido el sistema y volver a realizarse toda la evaluación.

El sistema fue evaluado por tres personas ajenas al proyecto con el siguiente equipo:

- 3 Computadoras Intel Dual Core 2 Duo 3.2ghz con 160gb de disco duro

Tabla de Características Técnica del Equipo			
Puertos USB Frontales	2 Puertos USB 2.0	Puertos PS2	2
Puertos USB Traseros	4 Puertos USB 2.0	Puerto de Red	1 Puerto de Red 10/100 Ethernet
Tecnología de Audio	6 Canales - High Definition Audio	Wireless	No incluido, pero se puede agregar tarjeta
Puertos de Audio Frontales	1 Micrófono y 1 Audífonos	Capacidad Máx. de Memoria	4 GB
Puertos de Audio Traseros	1 Micrófono, 1 Audífonos y 1 Line-In	Tipo de Memoria	DDR2
Chipset	G31 & ICH7	Gráficos	Intel GMA 3100 Media Acelerator
Puertos PCI	2 x PCI, 1 x PCI Express x1, 1 PCI Express x16	Ahorro de Energía	Sí
Puertos Sata Internos	4	Garantía	1 año

- Sistema operativo Windows XP SP3 y Windows 7
- 2 Monitores Lcd 15.6
- 1 Monitor Touch Elo Modelo 1937L

- 3 Impresoras de Ticket Epson Tm-u220pa

Los puntos que para evaluar el sistema son los que se mencionan en la Tabla 4.

Descripción	Estado
Instalación del sistema.	✅
Creación de la base de datos.	✅
Crear datos iniciales del sistema.	✅
Autentificación de acceso.	✅
Dar de alta a un usuario administrador, un cajero y un mesero.	✅
Verificar que cada usuario tenga acceso a sus secciones asignadas.	✅
Listar, Crear, modificar y eliminar artículos.	✅
Listar, Crear, modificar y eliminar clientes.	✅
Listar, Crear, modificar y eliminar empleados.	✅
Listar, Crear, modificar y eliminar tipos de empleados.	✅
Listar, Crear, modificar y eliminar impresoras.	✅
Listar, Crear, modificar y eliminar mesero.	✅
Agregar un pedido desde la interfaz de comanda.	✅
Editar un pedido desde la interfaz de comanda.	✅
Imprimir cuenta desde la interfaz de comanda.	✅
Imprimir ticket en cocina.	✅
Agregar un pedido desde la interfaz de caja.	✅
Editar un pedido desde la interfaz de caja.	✅
Imprimir cuenta desde la interfaz de caja.	✅
Facturar desde la interfaz caja.	✅
Realizar el cobro de una orden desde la interfaz caja.	✅
Cancelar una orden desde la interfaz caja.	✅
Realizar el corte del día desde la interfaz de corte.	✅
Comentarios	
Si durante la evaluación del sistema uno o más puntos fallaron, serán anotados en esta área todos los fallos encontrados para su corrección.	

Tabla 4. Evaluación del sistema.

Después aprobar la evaluación del sistema, se da por hecho que se tiene una versión estable y ya puede ser implementada.

Etapa de implementación

En esta etapa se instaló el sistema en el restaurante y se monitorio a lo largo de una semana. Al término de una semana se realizó una entrevista con los diferentes actores del sistema para

evaluar si no tuvieron ningún problema que pudiera haber pasado desapercibido en la etapa de prueba, así como posibles cambios o mejoras al sistema.

Etapa de mantenimiento

Después de recabar los comentarios acerca del funcionamiento del sistema, así como errores y mejoras son corregidos para entregar el producto final al cliente. Al término de esta etapa se llevara un ciclo regresando a la etapa de desarrollo hasta que no haya ningún cambio al sistema (Figura 73).

Figura 73. Ciclo de la etapa de mantenimiento.

Durante esta etapa se corrigieron, modificaron y mejoraron los siguientes puntos:

Problema	Causa	Corrección
Para instalar el sistema el cliente debía contactar al proveedor.	No se tenía un instalador.	Se creó un instalador para que el cliente lo pueda hacer de forma autónoma.
Las ordenes salían con diferentes, fechas y horarios.	La fecha que se tomaba para generar la orden era la de la computadora cliente.	Se implemento una clase que consultara la fecha y hora del servidor.
Como el sistema se implemento en computadoras con pantalla táctiles, en la interfaz para capturar la orden el mesero tenia dificultad para seleccionar una mesa del ComboBox.	El ComboBox tiene un tamaño muy pequeño para usarlo como componente para táctiles.	Se creó un botón de seleccionar mesa, que al púlsalo se abría una interfaz para que el mesero seleccione la mesa.

En la interfaz de capturar orden, el mesero no sabía que articulo tenía seleccionado.	No había nada que permitiera identificar que articulo estaba en selección.	Se agregó un PictureBox para mostrar la imagen del artículo seleccionado y debajo de la imagen el nombre y precio del mismo.
El restaurante se quedaba sin papel para las impresoras de ticket y el sistema seguía enviado a impresión del ticket.	No se tenía alguna manera de deshabilitar la impresión de ticket.	Se agregaron la opción de "No imprimir" y "Solo vista previa" para configurar la ruta de la impresora.

Capítulo 5. Conclusiones y Trabajos Futuros

A lo largo de este trabajo se desarrolló y documentó un sistema para facilitar la administración de un restaurante. El sistema permite a los dueños de un restaurante facilitar el manejo de sus comandas, el cobro de órdenes y realizar el corte de caja diario.

Para poder desarrollar este trabajo fue necesaria una metodología, que permitió seguir un proceso de manera segura, evitando que el proyecto no pudiese ser concluido. También facilitó la obtención de una idea clara del problema que se deseaba solucionar y la manera que debía ser solucionado, además de ayudar a organizar el trabajo.

La etapa de análisis es el primer paso para desarrollar cualquier proyecto de programación, un buen trabajo en esta etapa nos ayudará a prevenir posibles errores a lo largo del proyecto. Deben de anotarse todos los puntos que el cliente desea solucionar y no dar por hecho que se entendió el problema.

La etapa de diseño es el la más importante de todo el proceso, porque en esta etapa se lleva a cabo la logística que va tener el sistema y se realiza un bosquejo de cómo el sistema deberá operar. Esta etapa no puede pasarse por alto, ya que en ella el cliente debe aprobar el desarrollo del sistema. Un mal trabajo puede significar que se debe corregir el sistema durante su elaboración y que se verán reflejados en tiempos y costos no previstos para el desarrollador del proyecto.

Antes de la etapa de desarrollo es necesario establecer los lineamientos a seguir y definir los tiempos en que deberán ser concluidas las diferentes tareas. Si en la etapa de desarrollo hay más de un responsable para la programación y diseño no siguen los lineamentos causarán retrasos al momento de unir las tareas encargadas a cada uno.

La etapa de pruebas permite de forma interna detectar la mayor parte de errores, es muy importante que las personas encargadas de evaluar el sistema no tengan ninguna relación con el proyecto. En caso de encontrar errores es recomendable realizar la evaluación con personas diferentes a la prueba anterior.

La etapa de implementación no debe ser la última, ya que la etapa de pruebas solo es elaborada en un corto tiempo y se necesita monitorear el comportamiento del sistema por un tiempo prolongado con el fin de asegurar su correcto funcionamiento, así como la practicidad de usurarlo.

Un punto no tomado en cuenta puede causar perdida de información, lo cual no dejara una buena imagen del sistema.

La etapa de mantenimiento debe ser tomada en cuenta dentro del presupuesto del proyecto, ya sea como un límite de posibles cambios y mejoras. Por lo general, los clientes después de la etapa de implementación solicitan cambios y adecuaciones que no estaban planeados en la etapa de diseño, por este motivo se deberá acoplar al proyecto que fue aprobado y cualquier cambio mayor debe ser tratado como una nueva implementación.

La utilización de herramientas gratuitas permite a los lectores recrear el proyecto sin ningún problema. Además para las personas que comienzan a incursionar en el campo de desarrollo les permite un crecimiento previo a la utilización de herramientas que sin duda nos ayudan al desarrollo pero tienen un costo.

La herramienta de Fluent NHibernate permitió un ahorro en el tiempo de codificación ya que todo es más sintetizado. La programación orientada a objetos permitió tener un desarrollo más fácil ya que al leer el código se tiene una idea más clara de lo que se realiza sin la necesidad de consultar a los programadores, este punto se facilita a partir del diseño.

La programación por capas permitió la distribución del trabajo al menos en dos partes: la programación y el diseño, ahorrando tiempo al programador.

Un problema de usar herramientas gratuitas es que no tenemos acceso a todas sus ventajas y la implementación de algunas cosas puede ser complicada, como fue el caso de la impresión de tickets de manera automática ya que el generador de reportes de Visual Studio 2010 no tiene una instrucción para hacerlo y se necesitó de una clase alterna.

A pesar de todos los problemas que un proyecto puede tener durante su desarrollo el resultado es satisfactorio ya que se logra solucionar el problema de una organización de forma planificada. De igual manera, la experiencia que se adquiere a lo largo del desarrollo de varios proyectos ayudará a tener una mejor organización en las etapas de desarrollo.

La implementación de este sistema redujo el tiempo de cobro que era uno de los principales problemas, ayudó al dueño del restaurante a tener un mejor control sobre sus ventas diarias y a dar un mejor servicio a los clientes del negocio, que al final de cuentas fueron los más beneficiados.

Bibliografía

Deitel, H. M., & Deitel, P. J. (2007). *Cómo programar en C#.* México: Pearson Educación.

Dobson, R. (2003). Programación en Microsoft visual Basic .NET para bases de datos Microsoft Access. Mc Graw Hill.

Fluent NHibernate. (2010). *Fluent NHibernate*. (J. Gregory, Editor) Recuperado el 01 de 02 de 2011, de http://fluentnhibernate.org

Microsoft. (05 de 02 de 2011). *MSDN Microsoft*. Obtenido de http://127.0.0.1:47873/help/1-2252/ms.help?method=page&id=F61F02F2-2F20-483D-8F56-A9C8F3A54986&topicversion=100&topiclocale=EN-US&SQM=1&product=VS&productVersion=100&locale=EN-US

Microsystems, S. (s.f.). *Sun educational services, Basic Track I JAVA SCC-043-V020.*

Network, M. S. (s.f.). *MSDN Library*. Recuperado el 01 de 02 de 2011, de http://msdn.microsoft.com/library/ms123401

NHibernate. (s.f.). *NHibernate Forge*. Recuperado el 16 de 02 de 2011, de The official new home for the NHibernate for .NET community : http://nhforge.org/Default.aspx

Seco, J. A. (s.f.). El lenguaje de programación C#, PDF, Programación en castellano.

Sharch, S. R. (2005). *Análisis y diseño orientado a objetos con UML y el proceso unificado.* Mc Graw Hill.

Weitzenfeld, A. (2004). *Ingeniería de Software Orientada a Objetos con UML, JAVA e Internet.* Thompson.

Wikipedia. (05 de 02 de 2011). *NHibernate*. Obtenido de http://es.wikipedia.org/wiki/NHibernate

Wikipedia. (05 de 02 de 2011). *Programación por capas*. Obtenido de http://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas

Anexo A

Cada una de las interfaces mencionadas a continuación tiene una relación con los casos de uso como se muestra en la Tabla 5.

	Caso de uso												
Interfaz	1	2	3	4	5	6	7	8	9	10	11	12	13
1		⊕											
2													
3		⊕								⊕			⊕
4													⊕
5													⊕
6													⊕
7													⊕
8													⊕
9													⊕
10													⊕
11													⊕
12													⊕
13													⊕
14													⊕
15													⊕
16													⊕
17													⊕
18				⊕				⊕	⊕				
19								⊕	⊕				
20		⊕	⊕										
21		⊕											
22		⊕		⊕	⊕								
23								⊕					
24											⊕		
25												⊕	

Tabla 5. Relación de interfaces con los casos de uso

Casos de uso

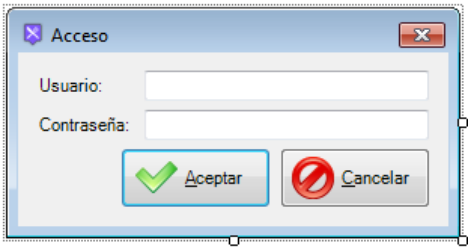
Interfaces

Este apartado se divido en cuatro secciones que estarán identificadas por un icono, como se muestra en la Tabla 6.

Icono	Concepto	Persona encargada
	Diseño	Diseñador y Programador
	Descripción del formulario	Programador
	Configuración de las propiedades	Diseñador
	Código	Programador

Tabla 6. Desarrollo de interfaz

Interfaz de acceso



Interfaz 1. Acceso

Esta interfaz permitirá al usuario identificarse ente el sistema permitiéndole acceder si sus datos son correctos.

Cuando el usuario pulse el botón de Aceptar validará si el usuario existe y si la contraseña es correcta, si ambos casos son cierto se abrirá la venta de inicio, de lo contrario se le notificará al usuario que el “Usuario o contraseña incorrectos.” y se reiniciarán los campos para que el usuario pueda intentarlo de nuevo. Si el usuario pulsa el botón Cancelar se cerrará la ventana.

Cuando esta interfaz cargue se verificará que exista la base de datos de lo contrario se abrirá una interfaz que permita crearla. Si la base datos ya existe se crearán las datos predeterminados que necesita el sistema para funcionar.

Windows Form

1. Name : FrmAcceso
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnAceptar
    g. Text : Acceso
    h. Icon : A consideración

TextBox

2. Name : txtUsuario
    a. TabIndex : 0
    b. CharacterCasing : Upper
3. Name : txtPassword
    a. TabIndex: 1
    b. CharacterCasing : Upper
    c. PasswordChar : *

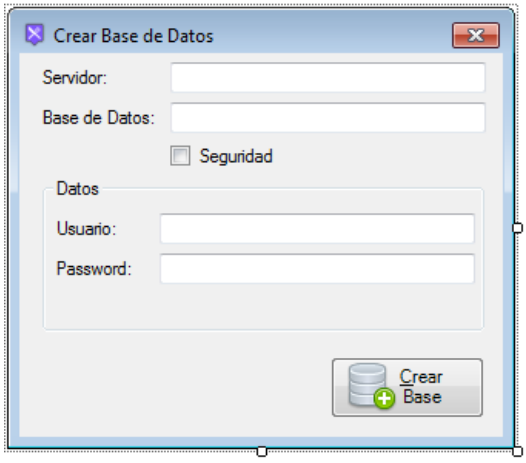
Button

4. Name : btnAceptar
    a. Text : &Aceptar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 2
    d. Width : 100
    e. Height : 40
    f. Image : A consideración
5. Name : btnCancelar
    a. Text : &Cancelar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 3
    d. Width : 100
    e. Height : 40
    f. Image : A consideración

```
using System;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;

namespace Restaurante
{
    public partial class FrmAcceso : Form
    {
        private ISession Session;

        public FrmAcceso()
        {
            InitializeComponent();
        }

        private void FrmAcceso_Load(object sender, EventArgs e)
        {
            try
            {
                Session = Conexion.CreateSessionFactory().OpenSession();
            }
            catch (Exception)
            {
                FrmCrearBaseDatos baseDatos = new FrmCrearBaseDatos();
                baseDatos.ShowDialog(this);
                Close();
            }
        }

        private void btnAceptar_Click(object sender, EventArgs e)
        {
            if (ValidarUsaurio())
            {
                DialogResult = DialogResult.OK;
            }
            else
            {
                MessageBox.Show("Usuario o contraseña incorrectos.");
                txtUsuario.Text = "";
                txtPassword.Text = "";
                txtUsuario.Focus();
            }
        }

        private bool ValidarUsaurio()
        {
            try
            {
                var lista = from c in Session.Query<EmpleadoClass>()
                                    where (c.Activo)
                                    select c;

                            foreach (var x in lista)
                            {
                                if (x.Usuario == txtUsuario.Text && x.Password == txtPassword.Text)
                                {
                                    Global.empleadoGlobal = x;
                                    return true;
                                }
                            }
                            return false;
            }catch (Exception)
            {
                Global.PrimerVez = true;
                Session = Conexion.CreateSessionFactory().OpenSession();
                IniciarDatos.InicializarUsuarios(Session);
                IniciarDatos.InicializarEstadoOrden(Session);
                MessageBox.Show("Tablas creadas correctamente.");
                return false;
            }
```

```
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

Interfaz para crear la base de datos



Interfaz 2. Creación de la base de datos

Esta interfaz permitirá al administrador crear la base de datos que requiere el sistema para funcionar la primera vez que este se ejecute. Por defecto estará configurado para crear la base de datos de forma local, con seguridad integrada y el nombre de la base de datos por defecto es "Restaurante2011".

La sección de datos estará oculta hasta que el usuario active el campo de Seguridad, esta sección servirá para personalizar si el administrador quiere definir un usuario y contraseña especifico a la base de datos.

Al pulsar el botón de Crear base será creada la base y además será modificado el archivo de configuración del sistema con los nuevos datos para que pueda realizar la conexión correctamente.

Si la base ya existe deberá desplegar un mensaje que nos informe de ello y del mismo modo si se creó correctamente.

Windows Form

1. Name : FrmCrearBaseDatos
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnCrearBase
    g. Text : Crear Base de Datos
    h. Icon : A consideración

TextBox

2. Name : txtServidor
    a. TabIndex : 0
3. Name : txtBase
    a. TabIndex: 1

11. Name : txtUsuario
    c. TabIndex: 3
12. Name : txtPassword
    d. TabIndex: 4

Button

11. Name : btnCrearBase
    a. Text : &Crear Base
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 5
    d. Width : 100
    e. Height : 40
    f. Image : A consideración

CheckBox

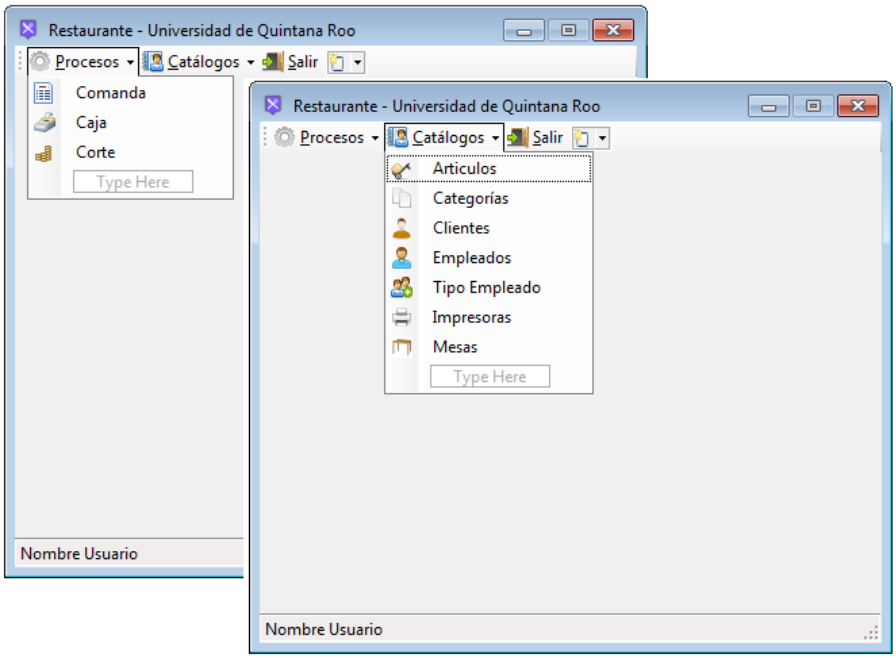
12. Name : chkSeguridad
    a. Text : Seguridad
    b. TabIndex : 2

GroupBox

13. Name : gpSeguridad
    a. Text : Datos

```
using System;
using System.Windows.Forms;
using Restaurante.Funciones;

namespace Restaurante
{
    public partial class FrmCrearBaseDatos : Form
    {
        public FrmCrearBaseDatos()
        {
            InitializeComponent();
        }

        private void FrmCrearBaseDatos_Load(object sender, EventArgs e)
        {
            txtServidor.Text = @"(local)\sqlexpress";
            txtBase.Text = "Restaurant";
            this.Height = 169;
            gpSeguridad.Visible = false;
        }

        private void btnCrearBase_Click(object sender, EventArgs e)
        {
            string baseDatos = txtBase.Text;
            string servidor = txtServidor.Text;
            string usuario = txtUsuario.Text;
            string password = txtPassword.Text;
            string seguridad = "";
            string connexion = "<add name=\"conexionString\" connectionString=\"Data Source=";
            connexion += servidor + ";Initial Catalog=" + baseDatos + ";";

            if (usuario != "" && password != "")
            {
                seguridad += "User ID=" + usuario + ";Password=" + password;
            }
            else
            {
                seguridad += "Integrated Security=True";
            }

            connexion += seguridad + "\" providerName=\"System.Data.SqlClient\" />";

            FunGenerarApp.Generar(connexion);
            CrearBaseDatos.Crear(servidor,seguridad,baseDatos);
            Close();
        }

        private void chkSeguridad_CheckedChanged(object sender, EventArgs e)
        {
            txtUsuario.Text = txtPassword.Text = "";
            if (chkSeguridad.Checked)
            {
                gpSeguridad.Visible = true;
                this.Height = 287;
            }
            else
            {
                gpSeguridad.Visible = false;
                this.Height = 169;
            }
        }

    }
}
```

Interfaz de inicio



Interfaz 3. Inicio

Esta interfaz contiene el menú principal del sistema donde después de autenticarse en él, tendrá acceso el usuario. De acuerdo al usuario esta interfaz le permitirá acceder a diferentes secciones, como se describe a continuación.

Administrador; Tendrá acceso a todos las secciones.

Cajero; Únicamente tendrá acceso a la sección de caja.

Mesero; Únicamente tendrá acceso a la sección de comanda.

La interfaz tendrá en la parte inferior el nombre del usuario que ha tenido acceso. Al seleccionar una sección se abrirá la interfaz de listado de la selección en curso y todas las interfaces que se abran serán de manera nodal. Una interfaz nodal quiere decir que no permitirá seleccionar otra interfaz a menos que la actual sea cerrada.

Windows Form

1. Name : Inicio
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : True
    e. KeyPreview : True
    f. Text : Crear Base de Datos
    g. Restaurante - Universidad de Quintana Roo
    h. Icon : A consideración

ToolStrip

2. Name : tsMenu

ToolStripDropDownButton

3. Name : tsdProcesos
    a. Text : &Procesos
    b. DisplayStyle : ImageAndText
    c. Image : A consideración
4. Name : tsdCatalogos
    a. Text : &Catálogos
    b. DisplayStyle : ImageAndText
    c. Image : A consideración

ToolStripButton

5. Name : tsbSalir
    a. Text : &Salir
    b. DisplayStyle : ImageAndText
    c. Image : A consideración

ToolStripMenuItem

6. Name : comandaToolStripMenuItem
    a. Text : Comanda
    b. DisplayStyle : ImageAndText
    c. Image : A consideración
7. Name : cajaToolStripMenuItem
    a. Text : Caja
    b. DisplayStyle : ImageAndText
    c. Image : A consideración
8. Name : corteToolStripMenuItem
    a. Text : Corte

   b. DisplayStyle : ImageAndText
   c. Image : A consideración
9. Name : articulosToolStripMenuItem
   a. Text : Artículos
   b. DisplayStyle : ImageAndText
   c. Image : A consideración
10. Name : categoriasToolStripMenuItem
   a. Text : Categorías
   b. DisplayStyle : ImageAndText
   c. Image : A consideración
11. Name : clientesToolStripMenuItem
   a. Text : Clientes
   b. DisplayStyle : ImageAndText
   c. Image : A consideración
12. Name : empleadosToolStripMenuItem
   a. Text : Empleados
   b. DisplayStyle : ImageAndText
   c. Image : A consideración
13. Name : tipoToolStripMenuItem
   a. Text : Tipo Empleado
   b. DisplayStyle : ImageAndText
   c. Image : A consideración
14. Name : impresorasToolStripMenuItem
   a. Text : Impresoras
   b. DisplayStyle : ImageAndText
   c. Image : A consideración
15. Name : mesasToolStripMenuItem
   a. Text : Mesas
   b. DisplayStyle : ImageAndText
   c. Image : A consideración

StatusStrip

24. Name : ssDatos

ToolStripStatusLabel

25. Name: lblUsuario
   a. Text: Nombre Usuario

```
using System;
using System.Windows.Forms;
using Restaurante.Catalogos;
using Restaurante.Procesos.Caja;
using Restaurante.Procesos.Comanda;
using Restaurante.Procesos.Corte;

namespace Restaurante
```

```
{
    public partial class Inicio : Form
    {
        private const int ADMINISTRADOR = 1;
        private const int MESERO = 2;
        private const int CAJERO = 3;

        public Inicio()
        {
            InitializeComponent();
        }

        private void Inicio_Load(object sender, EventArgs e)
        {
            lblUsuario.Text = Global.empleadoGlobal.Nombre;
            ValidarPermisos();
        }

        private void ValidarPermisos()
        {
            if (Global.empleadoGlobal.TipoEmpleado.Id > ADMINISTRADOR)
                tsbCatalogos.Visible = false;

            if (Global.empleadoGlobal.TipoEmpleado.Id == MESERO)
            {
                cajaToolStripMenuItem.Visible = false;
                corteToolStripMenuItem.Visible = false;
            }

            if (Global.empleadoGlobal.TipoEmpleado.Id == CAJERO)
            {
                comandaToolStripMenuItem.Visible = false;
                corteToolStripMenuItem.Visible = false;
            }
        }

        private void comandaToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FrmComanda frmComanda = new FrmComanda();
            frmComanda.ShowDialog(this);
        }

        private void cajaToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FrmCaja frmCaja = new FrmCaja();
            frmCaja.ShowDialog(this);
        }

        private void corteToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FrmCorte frmCorte = new FrmCorte();
            frmCorte.ShowDialog(this);
        }

        private void articulosToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FrmArticuloListado frmArticuloListado = new FrmArticuloListado();
            frmArticuloListado.ShowDialog(this);
        }

        private void categoríasToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FrmCategoriaListado frmCategoriaListado = new FrmCategoriaListado();
            frmCategoriaListado.ShowDialog(this);
        }

        private void clientesToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FrmClienteListado frmClienteListado = new FrmClienteListado();
            frmClienteListado.ShowDialog(this);
        }

        private void empleadoToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FrmEmpleadoListado frmEmpleadoListado = new FrmEmpleadoListado();
            frmEmpleadoListado.ShowDialog(this);
        }

        private void tipoToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FrmTipoEmpleadoListado frmTipoEmpleadoListado = new FrmTipoEmpleadoListado();
```

```
            frmTipoEmpleadoListado.ShowDialog(this);
        }

        private void impresorasToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FrmImpresoraListado frmImpresoraListado = new FrmImpresoraListado();
            frmImpresoraListado.ShowDialog(this);
        }

        private void mesasToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FrmMesaListado frmMesaListado = new FrmMesaListado();
            frmMesaListado.ShowDialog(this);
        }

        private void toolStripButton1_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

Interfaz para capturar artículos

Interfaz 4. Captura de artículos

Esta interfaz permitirá al administrador capturar los artículos que aparecerán en el menú del restaurante, los cuales serán clasificados por categorías y tendrán configurada una impresora en la cual será impreso el pedido. Podrá agregarse una imagen si se desea para mostrar en el menú de lo contario usará una por defecto.

La interfaz tendrá dos funciones agregar nuevos artículos y modificar los datos de un artículo previamente agregado. Cuando la ventana tenga la función de agregar, el titulo deberá ser “Agregar Artículo” y si la ventana tiene la función de modificar, el titulo deberá ser “Modificar Artículo”.

La ventana será de un tamaño definido y no podrá ser ampliada. Cuando el administrador pulse la tecla Enter hará referencia a la acción del botón Guardar y cuando la tecla que se pulse sea Escape hará referencia a la acción del botón Cancelar.

Cuando el administrador pulse el botón de Guardar, se tendrá que validar que todos los campos estén llenos de lo contrario deberá enviarse el mensaje de “Llene todos los campos.”. Si todos los campos están llenos se procederá agregar o modificar los datos según sea el caso.

Al guardar los cambios se le informará que los datos han sido guardados con el mensaje de “Información guardada correctamente. ¿Desea continuar?”, que le brindará al administrador la opción seguir agregando o modificando datos según sea el caso.

Si el administrador pulsa el botón Cancelar aparecerá un mensaje de “Desea guardar los cambios”, brindando la opción de que el administrador pueda guardas sus modificaciones en caso de haber pulsado el botón erróneamente.

Windows Form

1. Name : FrmArticuloCaptura
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

TextBox

2. Name : txtNombre
    a. Tabindex : 0

10. Name : txtPrecio
    a. Tabindex : 7

ComboBox

11. Name : cboCategorias
    c. TabIndex : 1
    d. Text : Seleccionar

12. Name : cboImpresoras
    a. TabIndex : 4
    b. Text : Seleccionar

PictureBox

13. Name : pictureBox
    a. SizeMode : Zoom
    b. Image : A consideración

OpenFileDialog

14. Name : openFileDialog
    a. FileName : openFileDialog

Button

15. Name : btnBuscarCategoria
    a. TabIndex : 2
    b. Width : 24
    c. Height : 24
    d. Image : A consideración
16. Name : btnAgregarCategoria
    a. TabIndex : 3
    b. Width : 24
    c. Height : 24
    d. Image : A consideración
17. Name : btnBuscarImpresora
    a. TabIndex : 5
    b. Width : 24
    c. Height : 24
    d. Image : A consideración
18. Name : btnAgregarImpresora
    a. TabIndex : 6
    b. Width : 24
    c. Height : 24
    d. Image : A consideración
19. Name : btnExaminar
    a. Text : &Examinar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 8
    d. Width : 100
    e. Height : 40
    f. Image : A consideración
20. Name : btnGuardar
    a. Text : &Guardar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 9
    d. Width : 100
    e. Height : 40
    f. Image : A consideración
21. Name : btnCancelar
    a. Text : &Cancelar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 10

d. Width : 100
e. Height : 40
f. Image : A consideración

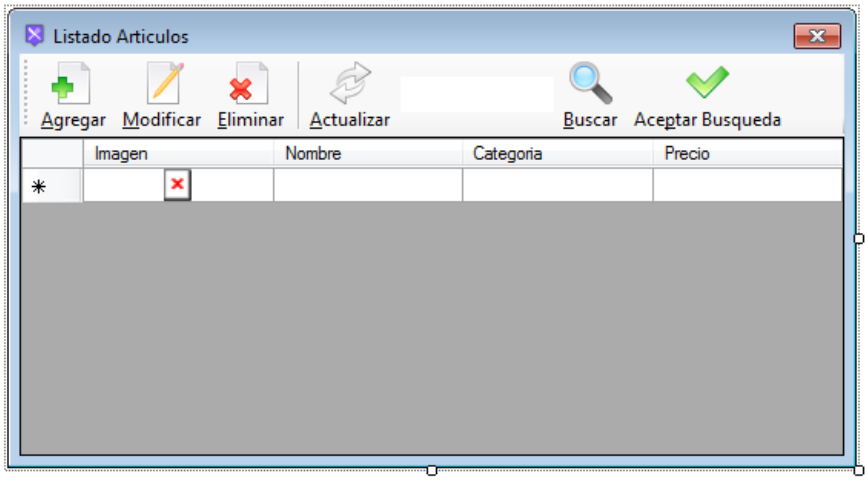
```
using System;
using System.ComponentModel;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;

namespace Restaurante.Catalogos
{
    public partial class FrmArticuloCaptura : Form
    {
        public ISession Session;
        public ArticuloClass Articulo;
        private bool nuevo;

        public FrmArticuloCaptura()
        {
            InitializeComponent();
        }

        private void FrmArticuloCaptura_Load(object sender, EventArgs e)
        {
            CargarCategorias();
            CargarImpresoras();

            if (Articulo == null)
            {
                InicializarDatos();
                nuevo = true;
            }
            else
            {
                CargarDatos();
                nuevo = false;
            }
        }

        private void CargarDatos()
        {
            txtNombre.Text = Articulo.Nombre;
            cboCategorias.SelectedValue = Articulo.Categoria != null ? (object)Articulo.Categoria.Id :
            null;
            cboImpresoras.SelectedValue = Articulo.Impresora != null ? (object)Articulo.Impresora.Id :
           null;
            txtPrecio.Text = Articulo.Precio.ToString("N2");

            if (Articulo.Imagen != null)
                pictureBox.Image = UtilidadesImagenes.ByteArrayToImage(Articulo.Imagen);
            else
                pictureBox.Image = Properties.Resources.image_default;

            this.Text = "Modificar Artículo";
        }

        private void InicializarDatos()
        {
            Articulo = new ArticuloClass();
            Articulo.Activo = true;

            txtNombre.Text = "";
            txtPrecio.Text = "";

            pictureBox.Image = Properties.Resources.image_default;
            openFileDialog.FileName = "openFileDialog";

            this.Text = "Agregar Artículo";
            txtNombre.Focus();
        }
```

```
        private void CargarCategorias()
        {
           var lista = from c in Session.Query<CategoriaClass>()
                       where c.Activo
                       select c;

            cboCategorias.DisplayMember = "Nombre";
            cboCategorias.ValueMember = "Id";
            cboCategorias.DataSource = lista.ToList();

            cboCategorias.SelectedIndex = -1;
        }

        private void CargarImpresoras()
        {
            var lista = from c in Session.Query<ImpresoraClass>()
                        where c.Activo
                        select c;

            cboImpresoras.DisplayMember = "Nombre";
            cboImpresoras.ValueMember = "Id";
            cboImpresoras.DataSource = lista.ToList();

            cboImpresoras.SelectedIndex = -1;
        }

        private void btnGuardar_Click(object sender, EventArgs e)
        {
            DialogResult opcion = MessageBox.Show("Información guardada correctamente ¿Desea continuar?",
"Guardar", MessageBoxButtons.YesNoCancel,
                                        MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
                if (!Guardar())
                    return;

                if (nuevo)
                {
                    InicializarDatos();
                }
                else
                {
                    CargarDatos();
                }
            }
            else if (opcion == DialogResult.No)
            {
                if (!Guardar())
                    return;

                Close();
            }
        }

        private bool Guardar()
        {
            if (!ValidarCampos())
            {
                MessageBox.Show("Llene todos los campos.");
                return false;
            }

            Articulo.Nombre = txtNombre.Text;
            Articulo.Precio = Convert.ToDecimal(txtPrecio.Text);
            Articulo.Categoria = Session.Load<CategoriaClass>(cboCategorias.SelectedValue);
            Articulo.Impresora = Session.Load<ImpresoraClass>(cboImpresoras.SelectedValue);

            if (openFileDialog.FileName != "openFileDialog")
            {
                Articulo.Imagen = UtilidadesImagenes.ObtenerByteArrayDeArchivo(openFileDialog.FileName);
            }
            else
            {
                if (nuevo)
                    Articulo.Imagen = UtilidadesImagenes.ImageToByteArray(Properties.Resources.image_defa
ult);
            }

            Session.Save(Articulo);
            Session.Flush();
```

```
            return true;
        }

        private bool ValidarCampos()
        {
            bool completo = true;

            if (txtNombre.Text == "")
                completo = false;

            if (!Validacion.EsDecimal(txtPrecio.Text))
            {
                MessageBox.Show("El precio debe ser un numero.");
                completo = false;
            }

            if (cboCategorias.SelectedValue == null)
                completo = false;

            if (cboImpresoras.SelectedValue == null)
                completo = false;

            string ImagenExtension = openFileDialog.FileName;
            ImagenExtension = UtilidadesImagenes.CortarCadena(ImagenExtension, '.');


            if (openFileDialog.FileName != "openFileDialog")
            {
                if (!Validacion.ValidarArchivoImagen(ImagenExtension))
                    completo = false;
            }

            return completo;
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            Cancelar();
        }

        private void Cancelar()
        {
            DialogResult opcion = MessageBox.Show("¿Desea guardar los cambios?", "Guardar", MessageBoxBut
tons.YesNoCancel,
                                                  MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
                if (!Guardar())
                    return;

                Close();
            }
            else if (opcion == DialogResult.No)
            {
                Close();
            }
        }

        private void FrmArticuloCaptura_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                Cancelar();
        }

        private void btnExaminar_Click(object sender, EventArgs e)
        {
            openFileDialog.ShowDialog(this);
        }

        private void openFileDialog_FileOk(object sender, CancelEventArgs e)
        {
            pictureBox.Image = UtilidadesImagenes.RetornarThumbnail(openFileDialog.FileName);
        }

        private void btnBuscarCategoria_Click(object sender, EventArgs e)
        {
            FrmCategoriaListado frmCategoriaListado = new FrmCategoriaListado
            {
                EsBusqueda = true,
                Session = Session
            };
```

```
            if (frmCategoriaListado.ShowDialog(this) != DialogResult.OK)
                return;

            CargarCategorias();

            cboCategorias.SelectedValue = frmCategoriaListado.CategoriaSeleccionado;
        }

        private void btnAgregarCategoria_Click(object sender, EventArgs e)
        {
            FrmCategoriaCaptura frmCategoriaCaptura = new FrmCategoriaCaptura
            {
                Session = Session
            };
            frmCategoriaCaptura.ShowDialog(this);

            CargarCategorias();

            if (!nuevo)
                CargarDatos();
        }

        private void btnBuscarImpresora_Click(object sender, EventArgs e)
        {
            FrmImpresoraListado frmImpresoraListado = new FrmImpresoraListado
            {
                EsBusqueda = true,
                Session = Session
            };
            if (frmImpresoraListado.ShowDialog(this) != DialogResult.OK)
                return;

            CargarImpresoras();

            cboImpresoras.SelectedValue = frmImpresoraListado.ImpresoraSeleccionado;
        }

        private void btnAgregarImpresora_Click(object sender, EventArgs e)
        {
            FrmImpresoraCaptura frmImpresoraCaptura = new FrmImpresoraCaptura
            {
                Session = Session
            };
            frmImpresoraCaptura.ShowDialog(this);

            CargarImpresoras();

            if (!nuevo)
                CargarDatos();
        }

    }
}
```

Interfaz para listar artículos



Interfaz 5. Listado de artículos

Esta interfaz tiene la finalidad de listar todos los artículos que hayan sido creados y que estén activos, además de las siguientes opciones:

1. Agregar; Al seleccionar esta opción abrirá la interfaz de captura correspondiente al listado, para crear un nuevo registro.

2. Modificar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de “No hay registro.”, de lo contrario abrirá la interfaz de captura correspondiente al listado para la edición del registro seleccionado.

3. Eliminar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de “No hay registro.”, de lo contrario enviará un mensaje de confirmación que diga “¿Esta seguro que desea eliminar este registro?”. Si se confirma la acción se procederá a deshabilitar el registro.

4. Actualizar; Esta opción permitirá recargar la búsqueda.

5. Campo de búsqueda; Por defecto se enlistan todos los registros activos, pero si el administrador coloca en el cuadro de búsqueda un número entero y recarga la búsqueda se buscará el registro con ese número. También el administrador podrá escribir una frase y al recargar se filtrarán los campos de texto que tengan alguna coincidencia.

6. Buscar; Esta opción permite recargar la búsqueda.

7. Aceptar Búsqueda; Esta opción únicamente estará activa cuando la ventana de listado sea usada para seleccionar un registro y devuelva el mismo a otra interfaz.

La interfaz no podrá se maximizada, ni minimizada pero si podrá cambiar. Al pulsar la tecla Escape la ventana deberá cerrarse.

Windows Form

1. Name : FrmArticuloListado
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

ToolStrip

2. Name : tsMenu

ToolStripButton

3. Name : btnAgregar
    a. Text : &Agregar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
4. Name : btnModificar
    a. Text : &Modificar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
5. Name : btnEliminar
    a. Text : &Eliminar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
6. Name : btnActalizar
    a. Text : A&ctualizar
    b. DisplayStyle : ImageAndText

   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
7. Name : btnBuscar
   a. Text : &Buscar
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
8. Name : btnAceptarBusqueda
   a. Text : Ace&ptar Busqueda
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración

ToolStripTextBox

9. Name : txtBuscar

DataGridView

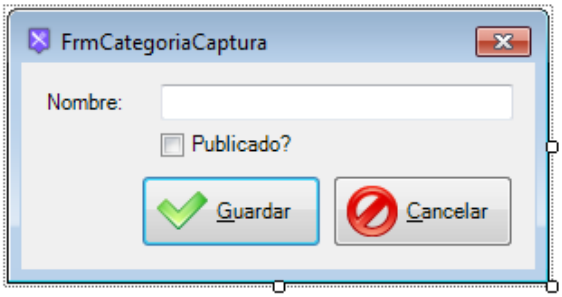
10. Name : grdArticulos
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
       i. Height : 40
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas

```
using System;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;

namespace Restaurante.Catalogos
{
    public partial class FrmArticuloListado : Form
    {
        public ISession Session;
        public bool EsBusqueda;
        public int ArticuloSeleccionado;

        public FrmArticuloListado()
        {
            InitializeComponent();
        }

        private void FrmArticuloListado_Load(object sender, EventArgs e)
```

```
{
    if (!EsBusqueda)
    {
        Session = Conexion.CreateSessionFactory().OpenSession();

        btnAceptarBusqueda.Visible = false;
    }

    grdArticulos.AutoGenerateColumns = false;

    CargarDatos();
}

private void CargarDatos()
{
    //Se llena una lista con los obejetos y las propiedades
    //que se viluazizán
    var lista = from c in Session.Query<ArticuloClass>()
                where c.Activo
                select new {
                c.Id,
                c.Imagen,
                c.Nombre,
                Categoria = c.Categoria.Nombre,
                c.Precio
                };

    if (txtBuscar.Text != "")
    {
        if (Validacion.EsEntero((txtBuscar.Text)))
            lista = lista.Where(c => c.Id == Convert.ToInt64(txtBuscar.Text));
        else
            lista = lista.Where(c => c.Nombre.Contains(txtBuscar.Text));
    }
    //Se limpia el Grid
    grdArticulos.DataSource = null;
    //Se asignan el listado al Grid
    grdArticulos.DataSource = lista.ToList();
}

private void btnActualizar_Click(object sender, EventArgs e)
{
    CargarDatos();
}

private void btnBuscar_Click(object sender, EventArgs e)
{
    CargarDatos();
}

private void txtBuscar_KeyDown(object sender, KeyEventArgs e)
{
    if(e.KeyCode == Keys.Return)
        CargarDatos();
}

private void btnAgregar_Click(object sender, EventArgs e)
{
    FrmArticuloCaptura frmArticuloCaptura = new FrmArticuloCaptura
                                            {
                                                Session = Session
                                            };
    frmArticuloCaptura.ShowDialog(this);
    CargarDatos();
}

private void btnModificar_Click(object sender, EventArgs e)
{
    Modificar();
}

private void Modificar()
{
    try
    {
        if (grdArticulos.CurrentRow == null)
            return;

        DataGridViewRow registro = grdArticulos.CurrentRow;
        int IdRegistro = (int)registro.Cells["Id"].Value;
        ArticuloClass Articulo = Session.Get<ArticuloClass>(IdRegistro);
```

```
                FrmArticuloCaptura frmArticuloCaptura = new FrmArticuloCaptura
                {
                    Session = Session,
                    Articulo = Articulo
                };
                frmArticuloCaptura.ShowDialog(this);
                CargarDatos();
            }
            catch (Exception)
            {
                MessageBox.Show("No hay registro.");
            }
        }

        private void btnEliminar_Click(object sender, EventArgs e)
        {
            try
            {
                if (grdArticulos.CurrentRow == null)
                    return;

                if (MessageBox.Show("¿Esta seguro que desea eliminar este registro?", "Eliminar",
                                    MessageBoxButtons.YesNo, MessageBoxIcon.Warning, MessageBoxDefaultBut
ton.Button2) ==
                    DialogResult.Yes){

                    //Se obtiene la fila en seleccion
                        DataGridViewRow registro = grdArticulos.CurrentRow;
                    //Se obtiene el valor de la celda con el nombre de Id
                        int IdRegistro = (int)registro.Cells["Id"].Value;
                        ArticuloClass Articulo = Session.Get<ArticuloClass>(IdRegistro);
                        Articulo.Activo = false;
                        Session.Save(Articulo);
                        Session.Flush();

                        CargarDatos();
                }
            }
            catch (Exception)
            {
                MessageBox.Show("No hay registro.");
            }
        }
        //Cuando se llama el evento doble clic
        private void grdArticulos_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
        {
            Modificar();
        }

        private void btnAceptarBusqueda_Click(object sender, EventArgs e)
        {
            if (grdArticulos.CurrentRow == null)
                return;

            DataGridViewRow row = grdArticulos.CurrentRow;
            ArticuloSeleccionado = (int)row.Cells["Id"].Value;
            DialogResult = DialogResult.OK;
            Close();
        }

        private void FrmArticuloListado_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                Close();
        }
    }
}
```

Interfaz para capturar categorías



Interfaz 6. Captura de categorías

Esta interfaz permitirá al administrador capturar las categorías que permitirán clasificar los artículos. Además se podrá definir si esa categoría estará publicada en el menú.

La interfaz tendrá dos funciones agregar nuevas categorías y modificar los datos de una categoría previamente agregada. Cuando la ventana tenga la función de agregar, el titulo deberá ser "Agregar Categoría" y si la ventana tiene la función de modificar, el titulo deberá ser "Modificar Categoría".

La ventana será de un tamaño definido y no podrá ser ampliada. Cuando el administrador pulse la tecla Enter hará referencia a la acción del botón Guardar y cuando la tecla que se pulse sea Escape hará referencia a la acción del botón Cancelar.

Cuando el administrador pulse el botón de Guardar, se tendrá que validar que todos los campos estén llenos de lo contrario deberá enviarse el mensaje de "Llene todos los campos.". Si todos los campos están llenos se procederá agregar o modificar los datos según sea el caso.

Al guardar los cambios se le informará que los datos han sido guardados con el mensaje de "Información guardada correctamente. ¿Desea continuar?", que le brindará al administrador la opción seguir agregando o modificando datos según sea el caso.

Si el administrador pulsa el botón Cancelar aparecerá un mensaje de "Desea guardar los cambios", brindando la opción de que el administrador pueda guardas sus modificaciones en caso de haber pulsado el botón erróneamente.

Windows Form

1. Name : FrmCategoríaCaptura
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

TextBox

2. Name : txtNombre
    a. Tabindex : 0

CheckBox

3. Name : chkPublicado
    a. Text : ¿Publicado?
    b. TabIndex : 2

Button

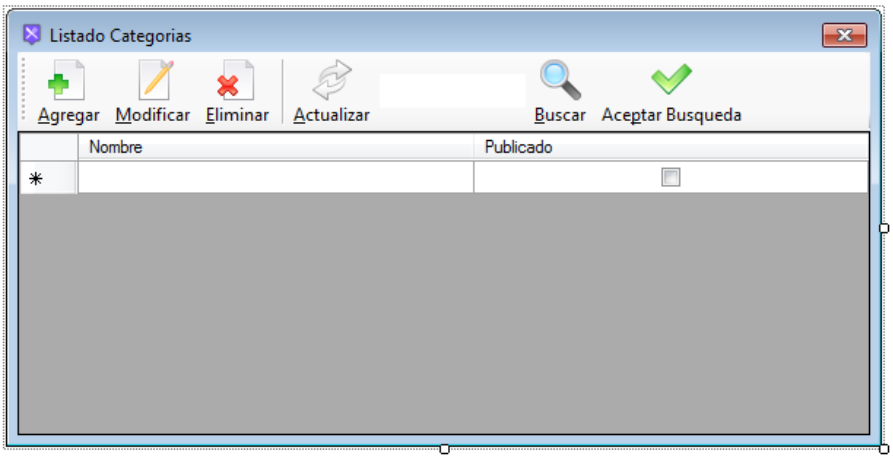
4. Name : btnGuardar
    a. Text : &Guardar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 3
    d. Width : 100
    e. Height : 40
    f. Image : A consideración
5. Name : btnCancelar
    a. Text : &Cancelar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 4
    d. Width : 100
    e. Height : 40
    f. Image : A consideración

```
using System;
using System.Windows.Forms;
using Datos;
using NHibernate;

namespace Restaurante.Catalogos
{
    public partial class FrmCategoriaCaptura : Form
    {
```

```
        public ISession Session;
        public CategoriaClass Categoria;
        private bool nuevo;

        public FrmCategoriaCaptura()
        {
            InitializeComponent();
        }

        private void FrmCategoriaCaptura_Load(object sender, EventArgs e)
        {
            if (Categoria == null)
            {
                InicializarDatos();
                nuevo = true;
            }
            else
            {
                CargarDatos();
                nuevo = false;
            }
        }

        private void CargarDatos()
        {
            txtNombre.Text = Categoria.Nombre;
            chkPublicado.Checked = Categoria.Publicado;
            this.Text = "Modificar Categoría";
        }

        private void InicializarDatos()
        {
            Categoria = new CategoriaClass();
            Categoria.Activo = true;
            txtNombre.Text = "";
            chkPublicado.Checked = false;

            this.Text = "Agregar Categoría";
            txtNombre.Focus();
        }

        private void btnGuardar_Click(object sender, EventArgs e)
        {
            DialogResult opcion = MessageBox.Show("¿Desea continuar?", "Guardar", MessageBoxButtons.YesNo
Cancel,
                                        MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
                if (!Guardar())
                    return;

                if (nuevo)
                {
                    InicializarDatos();
                }
                else
                {
                    CargarDatos();
                }
            }
            else if (opcion == DialogResult.No)
            {
                if (!Guardar())
                    return;

                Close();
            }
        }

        private bool Guardar()
        {
            if (!ValidarCampos())
            {
                MessageBox.Show("Llene todos los campos.");
                return false;
            }
            Categoria.Nombre = txtNombre.Text;
            Categoria.Publicado = chkPublicado.Checked;
            Session.Save(Categoria);
            Session.Flush();
            return true;
```

```
        }

        private bool ValidarCampos()
        {
            bool completo = true;

            if (txtNombre.Text == "")
                completo = false;

            return completo;
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            Cancelar();
        }

        private void Cancelar()
        {
            DialogResult opcion = MessageBox.Show("Información guardada correctamente ¿Desea continuar?",
 "Guardar", MessageBoxButtons.YesNoCancel,
                                                  MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
                if (!Guardar())
                    return;

                Close();
            }
            else if (opcion == DialogResult.No)
            {
                Close();
            }
        }

        private void FrmCategoriaCaptura_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                Cancelar();
        }
    }
}
```

Interfaz para listar categorías



Interfaz 7. Listado de categorías

Esta interfaz tiene la finalidad de listar todas las categorías que hayan sido creadas y que estén activas, además de las siguientes opciones:

1. Agregar; Al seleccionar esta opción abrirá la interfaz de captura correspondiente al listado, para crear un nuevo registro.

2. Modificar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de “No hay registro.”, de lo contrario abrirá la interfaz de captura correspondiente al listado para la edición del registro seleccionado.

3. Eliminar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de “No hay registro.”, de lo contrario enviará un mensaje de confirmación que diga “¿Esta seguro que desea eliminar este registro?”. Si se confirma la acción se procederá a deshabilitar el registro.

4. Actualizar; Esta opción permitirá recargar la búsqueda.

5. Campo de búsqueda; Por defecto se enlistan todos los registros activos, pero si el administrador coloca en el cuadro de búsqueda un número entero y recarga la búsqueda se buscará el registro con ese número. También el administrador podrá escribir una frase y al recargar se filtrarán los campos de texto que tengan alguna coincidencia.

6. Buscar; Esta opción permite recargar la búsqueda.

7. Aceptar Búsqueda; Esta opción únicamente estará activa cuando la ventana de listado sea usada para seleccionar un registro y devuelva el mismo a otra interfaz.

La interfaz no podrá se maximizada, ni minimizada pero si podrá cambiar. Al pulsar la tecla Escape la ventana deberá cerrarse.

Windows Form

1. Name : FrmCategoriaListado
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

ToolStrip

2. Name : tsMenu

ToolStripButton

3. Name : btnAgregar
    a. Text : &Agregar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
4. Name : btnModificar
    a. Text : &Modificar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
5. Name : btnEliminar
    a. Text : &Eliminar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
6. Name : btnActalizar
    a. Text : A&ctualizar
    b. DisplayStyle : ImageAndText

c. ImageScaling : None
d. TextImageRelation : ImageAboveText
e. Image : A consideración

7. Name : btnBuscar
   a. Text : &Buscar
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
8. Name : btnAceptarBusqueda
   a. Text : Ace&ptar Busqueda
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración

ToolStripTextBox

9. Name : txtBuscar

DataGridView

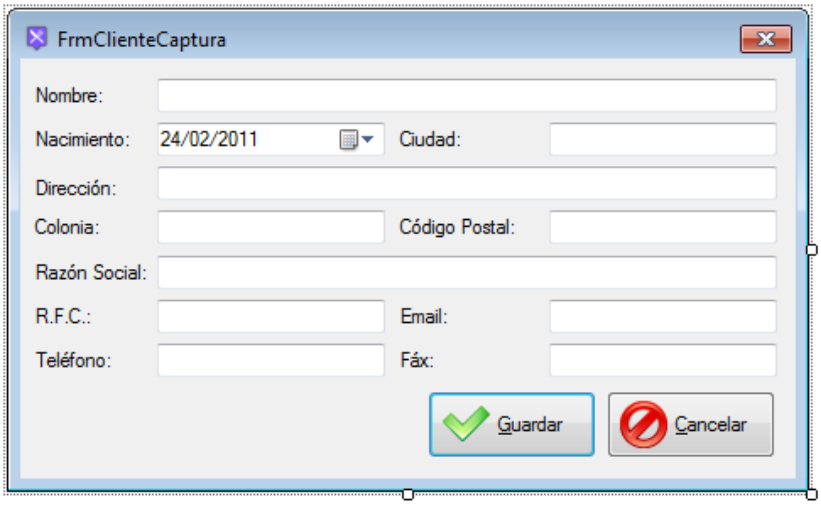
10. Name : grdCategorias
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas

```
using System;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;

namespace Restaurante.Catalogos
{
    public partial class FrmCategoriaListado : Form
    {
        public ISession Session;
        public bool EsBusqueda;
        public int CategoriaSeleccionado;

        public FrmCategoriaListado()
        {
            InitializeComponent();
        }

        private void FrmCategoriaListado_Load(object sender, EventArgs e)
        {
            if (!EsBusqueda)
            {
                Session = Conexion.CreateSessionFactory().OpenSession();
```

```
            btnAceptarBusqueda.Visible = false;
        }

        grdCategorias.AutoGenerateColumns = false;

        CargarDatos();
    }

    private void CargarDatos()
    {
        var lista = from c in Session.Query<CategoriaClass>()
                    where c.Activo
                    select c;

        if (txtBuscar.Text != "")
        {
            if (Validacion.EsEntero((txtBuscar.Text)))
                lista = lista.Where(c => c.Id == Convert.ToInt64(txtBuscar.Text));
            else
                lista = lista.Where(c => c.Nombre.Contains(txtBuscar.Text));
        }
        grdCategorias.DataSource = lista.ToList();
    }

    private void btnActualizar_Click(object sender, EventArgs e)
    {
        CargarDatos();
    }

    private void btnBuscar_Click(object sender, EventArgs e)
    {
        CargarDatos();
    }

    private void txtBuscar_KeyDown(object sender, KeyEventArgs e)
    {
        if(e.KeyCode == Keys.Return)
            CargarDatos();
    }

    private void btnAgregar_Click(object sender, EventArgs e)
    {
        FrmCategoriaCaptura frmCategoriaCaptura = new FrmCategoriaCaptura
                                                  {
                                                      Session = Session
                                                  };
        frmCategoriaCaptura.ShowDialog(this);
        CargarDatos();
    }

    private void btnModificar_Click(object sender, EventArgs e)
    {
        Modificar();
    }

    private void Modificar()
    {
        try
        {
            if (grdCategorias.CurrentRow == null)
                return;

            DataGridViewRow registro = grdCategorias.CurrentRow;
            int IdRegistro = (int)registro.Cells["Id"].Value;
            CategoriaClass Categoria = Session.Get<CategoriaClass>(IdRegistro);

            FrmCategoriaCaptura frmCategoriaCaptura = new FrmCategoriaCaptura
            {
                Session = Session,
                Categoria = Categoria
            };
            frmCategoriaCaptura.ShowDialog(this);
            CargarDatos();
        }
        catch (Exception)
        {
            MessageBox.Show("No hay registro.");
        }
    }

    private void btnEliminar_Click(object sender, EventArgs e)
```

```
        {
            try
            {
                if (grdCategorias.CurrentRow == null)
                    return;

                if (MessageBox.Show("¿Esta seguro que desea eliminar este registro?", "Eliminar",
                                    MessageBoxButtons.YesNo, MessageBoxIcon.Warning, MessageBoxDefaultBut
ton.Button2) ==
                    DialogResult.Yes){

                            DataGridViewRow registro = grdCategorias.CurrentRow;
                            int IdRegistro = (int)registro.Cells["Id"].Value;
                            CategoriaClass Categoria = Session.Get<CategoriaClass>(IdRegistro);
                            Categoria.Activo = false;
                            Session.Save(Categoria);
                            Session.Flush();

                            CargarDatos();
                }
            }
            catch (Exception)
            {
                MessageBox.Show("No hay registro.");
            }
        }

        private void grdCategorias_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
        {
            Modificar();
        }

        private void btnAceptarBusqueda_Click(object sender, EventArgs e)
        {
            if (grdCategorias.CurrentRow == null)
                return;

            DataGridViewRow row = grdCategorias.CurrentRow;
            CategoriaSeleccionado = (int)row.Cells["Id"].Value;
            DialogResult = DialogResult.OK;
            Close();
        }

        private void FrmCategoriaListado_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                Close();
        }
    }
}
```

Interfaz para capturar clientes



Interfaz 8. Captura de clientes

Esta interfaz permitirá al administrador capturar los clientes y sus datos necesarios para realizar facturas.

La interfaz tendrá dos funciones agregar nuevos clientes y modificar los datos de un cliente previamente agregada. Cuando la ventana tenga la función de agregar, el titulo deberá ser "Agregar Cliente" y si la ventana tiene la función de modificar, el titulo deberá ser "Modificar Cliente".

La ventana será de un tamaño definido y no podrá ser ampliada. Cuando el administrador pulse la tecla Enter hará referencia a la acción del botón Guardar y cuando la tecla que se pulse sea Escape hará referencia a la acción del botón Cancelar.

Cuando el administrador pulse el botón de Guardar, se tendrá que validar que todos los campos estén llenos de lo contrario deberá enviarse el mensaje de "Llene todos los campos.". Si todos los campos están llenos se procederá agregar o modificar los datos según sea el caso.

Al guardar los cambios se le informará que los datos han sido guardados con el mensaje de "Información guardada correctamente. ¿Desea continuar?", que le brindará al administrador la opción seguir agregando o modificando datos según sea el caso.

Si el administrador pulsa el botón Cancelar aparecerá un mensaje de “Desea guardar los cambios”, brindando la opción de que el administrador pueda guardas sus modificaciones en caso de haber pulsado el botón erróneamente.

Windows Form

1. Name : FrmClienteCaptura
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

TextBox

2. Name : txtNombre
    a. Tabindex : 0
3. Name : txtCiudad
    a. TabIndex : 2
4. Name : txtDireccion
    a. TabIndex : 3
5. Name : txtColonia
    a. TabIndex : 4
6. Name : txtCodigoPostal
    a. TabIndex : 5
7. Name : txtRazonSocial
    a. TabIndex : 6
8. Name : txtRfc
    a. TabIndex: 7
9. Name : txtEmail
    a. TabIndex : 8
10. Name : txtTelefono
    a. TabIndex : 9
11. Name : TxtFax
    a. TabIndex : 10

DateTimePicker

12. Name : dtpNacimiento
    a. Format : Short
    b. TabIndex : 1

Button

13. Name : btnGuardar

a. Text : &Guardar
b. TextImageRelation : ImageBeforeText
c. TabIndex : 11
d. Width : 100
e. Height : 40
f. Image : A consideración

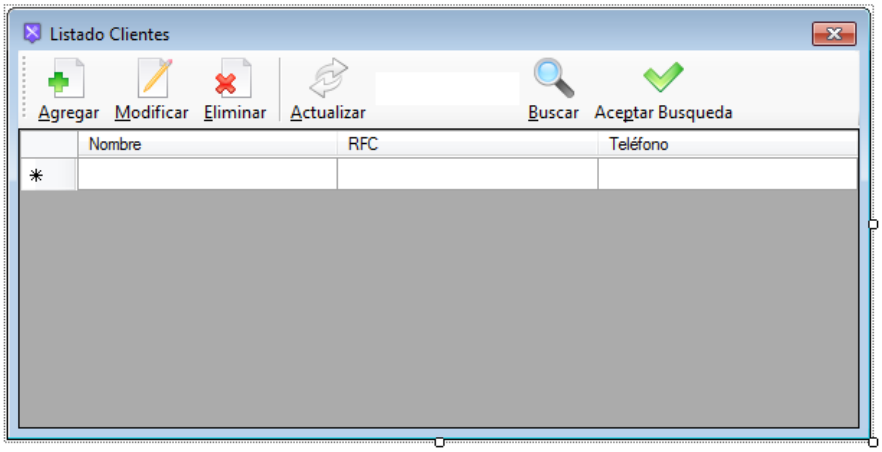
14. Name : btnCancelar
    a. Text : &Cancelar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 12
    d. Width : 100
    e. Height : 40
    f. Image : A consideración

```
using System;
using System.Windows.Forms;
using Datos;
using NHibernate;

namespace Restaurante.Catalogos
{
    public partial class FrmClienteCaptura : Form
    {
        public ISession Session;
        public ClienteClass Cliente;
        private bool nuevo;

        public FrmClienteCaptura()
        {
            InitializeComponent();
        }

        private void FrmClienteCaptura_Load(object sender, EventArgs e)
        {
            if (Cliente == null)
            {
                InicializarDatos();
                nuevo = true;
            }
            else
            {
                CargarDatos();
                nuevo = false;
            }
        }

        private void CargarDatos()
        {
            txtNombre.Text = Cliente.Nombre;
            dtpNacimiento.Value = Cliente.FechaNacimiento;
            txtCiudad.Text = Cliente.Ciudad;
            txtDireccion.Text = Cliente.Direccion;
            txtColonia.Text = Cliente.Colonia;
            txtCodigoPostal.Text = Cliente.CodigoPostal;
            txtRazonSocial.Text = Cliente.RazonSocial;
            txtRfc.Text = Cliente.Rfc;
            txtEmail.Text = Cliente.Email;
            txtTelefono.Text = Cliente.Telefono;
            txtFax.Text = Cliente.Fax;

            this.Text = "Modificar Cliente";
        }

        private void InicializarDatos()
        {
            Cliente = new ClienteClass();
            Cliente.Activo = true;
```

```
            txtNombre.Text = "";
            dtpNacimiento.Value = DateTime.Today;
            txtCiudad.Text = "";
            txtDireccion.Text = "";
            txtColonia.Text = "";
            txtCodigoPostal.Text = "";
            txtRazonSocial.Text = "";
            txtRfc.Text = "";
            txtEmail.Text = "";
            txtTelefono.Text = "";
            txtFax.Text = "";

            this.Text = "Agregar Cliente";
            txtNombre.Focus();
        }

        private void btnGuardar_Click(object sender, EventArgs e)
        {
            DialogResult opcion = MessageBox.Show("Información guardada correctamente ¿Desea continuar?",
"Guardar", MessageBoxButtons.YesNoCancel,
                                        MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
                if (!Guardar())
                    return;

                if (nuevo)
                {
                    InicializarDatos();
                }
                else
                {
                    CargarDatos();
                }
            }
            else if (opcion == DialogResult.No)
            {
                if (!Guardar())
                    return;

                Close();
            }
        }

        private bool Guardar()
        {
            if (!ValidarCampos())
            {
                MessageBox.Show("Llene todos los campos.");
                return false;
            }

            Cliente.Nombre = txtNombre.Text;
            Cliente.FechaNacimiento = dtpNacimiento.Value;
            Cliente.Ciudad = txtCiudad.Text;
            Cliente.Direccion = txtDireccion.Text;
            Cliente.Colonia = txtColonia.Text;
            Cliente.CodigoPostal = txtCodigoPostal.Text;
            Cliente.RazonSocial = txtRazonSocial.Text;
            Cliente.Rfc = txtRfc.Text;
            Cliente.Email = txtEmail.Text;
            Cliente.Telefono = txtTelefono.Text;
            Cliente.Fax = txtFax.Text;

            Session.Save(Cliente);
            Session.Flush();
            return true;
        }

        private bool ValidarCampos()
        {
            bool completo = true;

            if (txtNombre.Text == "")
                completo = false;

            return completo;
        }

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

```
            Cancelar();
        }

        private void Cancelar()
        {
            DialogResult opcion = MessageBox.Show("¿Desea guardar los cambios?", "Guardar", MessageBoxBut
tons.YesNoCancel,
                                                  MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
                if (!Guardar())
                    return;

                Close();
            }
            else if (opcion == DialogResult.No)
            {
                Close();
            }
        }

        private void FrmClienteCaptura_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                Cancelar();
        }
    }
}
```

Interfaz para listar clientes



Interfaz 9. Listado de clientes

Esta interfaz tiene la finalidad de listar todos los clientes que hayan sido creados y que estén activos, además de las siguientes opciones:

1. Agregar; Al seleccionar esta opción abrirá la interfaz de captura correspondiente al listado, para crear un nuevo registro.

2. Modificar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de “No hay registro.”, de lo contrario abrirá la interfaz de captura correspondiente al listado para la edición del registro seleccionado.

3. Eliminar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de “No hay registro.”, de lo contrario enviará un mensaje de confirmación que diga “¿Esta seguro que desea eliminar este registro?”. Si se confirma la acción se procederá a deshabilitar el registro.

4. Actualizar; Esta opción permitirá recargar la búsqueda.

5. Campo de búsqueda; Por defecto se enlistan todos los registros activos, pero si el administrador coloca en el cuadro de búsqueda un número entero y recarga la búsqueda se buscará el registro con ese número. También el administrador podrá escribir una frase y al recargar se filtrarán los campos de texto que tengan alguna coincidencia.

6. Buscar; Esta opción permite recargar la búsqueda.

7. Aceptar Búsqueda; Esta opción únicamente estará activa cuando la ventana de listado sea usada para seleccionar un registro y devuelva el mismo a otra interfaz.

La interfaz no podrá se maximizada, ni minimizada pero si podrá cambiar. Al pulsar la tecla Escape la ventana deberá cerrarse.

Windows Form

1. Name : FrmClienteListado
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

ToolStrip

2. Name : tsMenu

ToolStripButton

3. Name : btnAgregar
    a. Text : &Agregar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
4. Name : btnModificar
    a. Text : &Modificar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
5. Name : btnEliminar
    a. Text : &Eliminar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
6. Name : btnActalizar
    a. Text : A&ctualizar
    b. DisplayStyle : ImageAndText

   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
7. Name : btnBuscar
   a. Text : &Buscar
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
8. Name : btnAceptarBusqueda
   a. Text : Ace&ptar Busqueda
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración

ToolStripTextBox

9. Name : txtBuscar

DataGridView

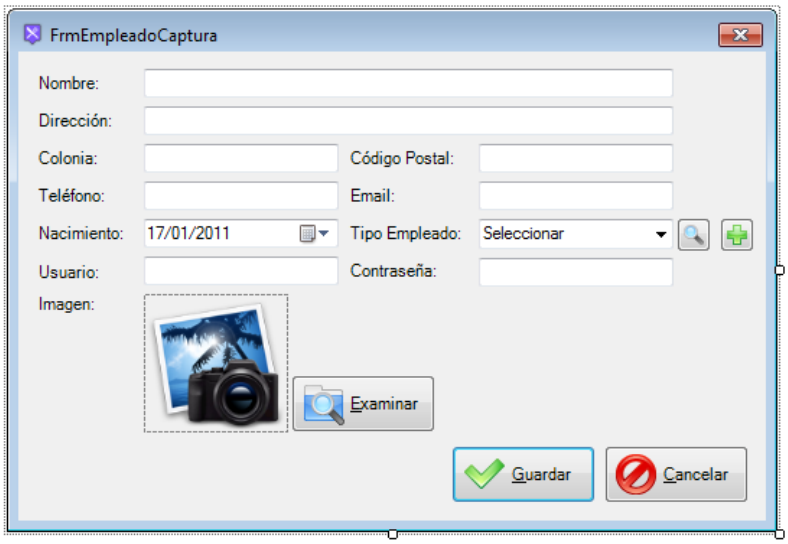
10. Name : grdClientes
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas

```
using System;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;

namespace Restaurante.Catalogos
{
    public partial class FrmClienteListado : Form
    {
        public ISession Session;
        public bool EsBusqueda;
        public int ClienteSeleccionado;

        public FrmClienteListado()
        {
            InitializeComponent();
        }

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

```
        if (!EsBusqueda)
        {
            Session = Conexion.CreateSessionFactory().OpenSession();

            btnAceptarBusqueda.Visible = false;
        }

        grdClientes.AutoGenerateColumns = false;

        CargarDatos();
    }

    private void CargarDatos()
    {
        var lista = from c in Session.Query<ClienteClass>()
                    where c.Activo
                    select c;

        if (txtBuscar.Text != "")
        {
            if (Validacion.EsEntero((txtBuscar.Text)))
                lista = lista.Where(c => c.Id == Convert.ToInt64(txtBuscar.Text));
            else
                lista = lista.Where(c => c.Nombre.Contains(txtBuscar.Text));
        }
        grdClientes.DataSource = lista.ToList();
    }

    private void btnActualizar_Click(object sender, EventArgs e)
    {
        CargarDatos();
    }

    private void btnBuscar_Click(object sender, EventArgs e)
    {
        CargarDatos();
    }

    private void txtBuscar_KeyDown(object sender, KeyEventArgs e)
    {
        if(e.KeyCode == Keys.Return)
            CargarDatos();
    }

    private void btnAgregar_Click(object sender, EventArgs e)
    {
        FrmClienteCaptura frmClienteCaptura = new FrmClienteCaptura
                                                {
                                                    Session = Session
                                                };
        frmClienteCaptura.ShowDialog(this);
        CargarDatos();
    }

    private void btnModificar_Click(object sender, EventArgs e)
    {
        Modificar();
    }

    private void Modificar()
    {
        try
        {
            if (grdClientes.CurrentRow == null)
                return;

            DataGridViewRow registro = grdClientes.CurrentRow;
            int IdRegistro = (int)registro.Cells["Id"].Value;
            ClienteClass Cliente = Session.Get<ClienteClass>(IdRegistro);

            FrmClienteCaptura frmClienteCaptura = new FrmClienteCaptura
            {
                Session = Session,
                Cliente = Cliente
            };
            frmClienteCaptura.ShowDialog(this);
            CargarDatos();
        }
        catch (Exception)
        {
            MessageBox.Show("No hay registro.");
```

```
            }
        }

        private void btnEliminar_Click(object sender, EventArgs e)
        {
            try
            {
                if (grdClientes.CurrentRow == null)
                    return;

                if (MessageBox.Show("¿Esta seguro que desea eliminar este registro?", "Eliminar",
                                    MessageBoxButtons.YesNo, MessageBoxIcon.Warning, MessageBoxDefaultBut
ton.Button2) ==
                    DialogResult.Yes){

                            DataGridViewRow registro = grdClientes.CurrentRow;
                            int IdRegistro = (int)registro.Cells["Id"].Value;
                            ClienteClass Cliente = Session.Get<ClienteClass>(IdRegistro);
                            Cliente.Activo = false;
                            Session.Save(Cliente);
                            Session.Flush();

                            CargarDatos();
                }
            }
            catch (Exception)
            {
                MessageBox.Show("No hay registro.");
            }
        }

        private void grdClientes_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
        {
            Modificar();
        }

        private void btnAceptarBusqueda_Click_1(object sender, EventArgs e)
        {
            if (grdClientes.CurrentRow == null)
                return;

            DataGridViewRow row = grdClientes.CurrentRow;
            ClienteSeleccionado = (int)row.Cells["Id"].Value;
            DialogResult = DialogResult.OK;
            Close();
        }

        private void FrmClienteListado_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                Close();
        }
    }
}
```

Interfaz para capturar empleados



Interfaz 10. Captura de empleados

Esta interfaz permitirá al administrador capturar los datos de los empleados que tienen acceso al sistema. Los empleados serán clasificados por varios tipos como: Administrador, Cajero o Mesero, donde cada uno tendrá acceso a diferentes secciones del sistema según sea el caso.

La interfaz tendrá dos funciones agregar nuevos empleados y modificar los datos de un empleado previamente agregado. Cuando la ventana tenga la función de agregar, el titulo deberá ser “Agregar Empleado” y si la ventana tiene la función de modificar, el titulo deberá ser “Modificar Empleado”.

La ventana será de un tamaño definido y no podrá ser ampliada. Cuando el usuario pulse la tecla Enter hará referencia a la acción del botón Guardar y cuando la tecla que se pulse sea Escape hará referencia a la acción del botón Cancelar.

Cuando el administrador pulse el botón de Guardar, se tendrá que validar que todos los campos estén llenos de lo contrario deberá enviarse el mensaje de “Llene todos los campos.”. Si todos los campos están llenos se procederá agregar o modificar los datos según sea el caso.

Al guardar los cambios se le informará que los datos han sido guardados con el mensaje de "Información guardada correctamente. ¿Desea continuar?", que le brindará al administrador la opción seguir agregando o modificando datos según sea el caso.

Si el administrador pulsa el botón Cancelar aparecerá un mensaje de "Desea guardar los cambios", brindando la opción de que el administrador pueda guardas sus modificaciones en caso de haber pulsado el botón erróneamente.

Windows Form

1. Name : FrmEmpleadoCaptura
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

TextBox

2. Name : txtNombre
    a. Tabindex : 0
3. Name : txtDireccion
    a. TabIndex : 1
4. Name : txtColonia
    a. TabIndex : 2
5. Name : txtCodigoPostal
    a. TabIndex : 3
6. Name : txtTelefono
    a. TabIndex : 4
7. Name : txtEmail
    a. TabIndex : 5
8. Name : txtUsuario
    a. TabIndex : 10
    b. CharacterCasing : Upper
9. Name : txtPassword
    a. TabIndex: 11
    b. CharacterCasing : Upper

DateTimePicker

10. Name : dtpNacimiento
    c. Format : Short
    d. TabIndex : 6

ComboBox

11. Name : cboTiposEmpleados
    a. TabIndex : 7
    b. Text : Seleccionar

PictureBox

12. Name : pictureBox
    a. SizeMode : Zoom
    b. Image : A consideración

OpenFileDialog

13. Name : openFileDialog
    a. FileName : openFileDialog

Button

14. Name : btnBuscarTipoEmpleado
    a. TabIndex : 8
    b. Width : 24
    c. Height : 24
    d. Image : A consideración
15. Name : btnAgregarTipoEmpleado
    a. TabIndex : 9
    b. Width : 24
    c. Height : 24
    d. Image : A consideración
16. Name : btnExaminar
    a. Text : &Examinar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 12
    d. Width : 100
    e. Height : 40
    f. Image : A consideración
17. Name : btnGuardar
    a. Text : &Guardar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 13
    d. Width : 100
    e. Height : 40
    f. Image : A consideración
18. Name : btnCancelar
    a. Text : &Cancelar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 14
    d. Width : 100
    e. Height : 40
    f. Image : A consideración

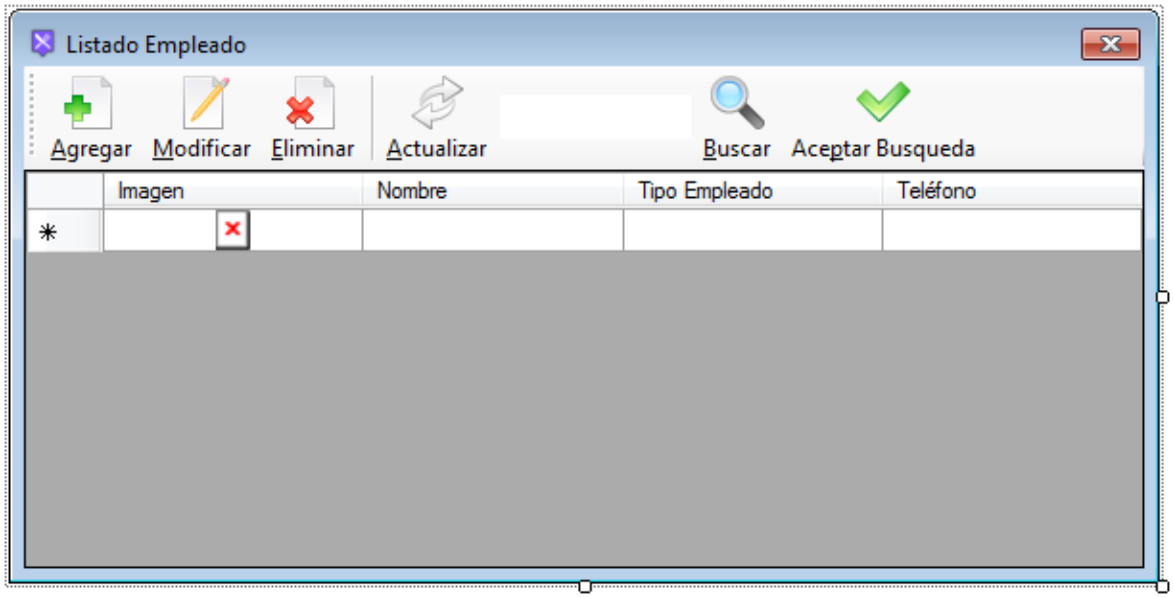
```
using System;
using System.ComponentModel;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;

namespace Restaurante.Catalogos
{
    public partial class FrmEmpleadoCaptura : Form
    {
        public ISession Session;
        public EmpleadoClass Empleado;
        private bool nuevo;

        public FrmEmpleadoCaptura()
        {
            InitializeComponent();
        }

        private void FrmEmpleadoCaptura_Load(object sender, EventArgs e)
        {
            CargarTiposEmpleados();
            if (Empleado == null)
            {
                InicializarDatos();
                nuevo = true;
            }
            else
            {
                CargarDatos();
                nuevo = false;
            }
        }

        private void CargarDatos()
        {
            txtNombre.Text = Empleado.Nombre;
            txtDireccion.Text = Empleado.Direccion;
            txtColonia.Text = Empleado.Colonia;
            txtCodigoPostal.Text = Empleado.CodigoPostal;
            txtTelefono.Text = Empleado.Telefono;
            txtEmail.Text = Empleado.Email;
            dtpNacimiento.Value = Empleado.FechaNacimiento;
            cboTiposEmpleados.SelectedValue = Empleado.TipoEmpleado != null ? (object)Empleado.TipoEmplea
do.Id :
               null;
            txtUsuario.Text = Empleado.Usuario;
            txtPassword.Text = Empleado.Password;

            if (Empleado.Imagen != null)
                pictureBox.Image = UtilidadesImagenes.ByteArrayToImage(Empleado.Imagen);
            else
                pictureBox.Image = Properties.Resources.image_default;

            this.Text = "Modificar Empleado";
        }

        private void InicializarDatos()
        {
            Empleado = new EmpleadoClass();
            Empleado.Activo = true;

            txtNombre.Text = "";
            txtDireccion.Text = "";
            txtColonia.Text = "";
            txtCodigoPostal.Text = "";
            txtTelefono.Text = "";
            txtEmail.Text = "";
            dtpNacimiento.Value = DateTime.Today;
            cboTiposEmpleados.SelectedIndex = -1;
            txtUsuario.Text = "";
            txtPassword.Text = "";
```

```
            pictureBox.Image = Properties.Resources.image_default;
            openFileDialog.FileName = "openFileDialog";

            this.Text = "Agregar Empleado";
            txtNombre.Focus();
        }

        private void CargarTiposEmpleados()
        {
            var lista = from c in Session.Query<TipoEmpleadoClass>()
                        where c.Activo
                        select c;
            cboTiposEmpleados.DisplayMember = "Nombre";
            cboTiposEmpleados.ValueMember = "Id";
            cboTiposEmpleados.DataSource = lista.ToList();

            cboTiposEmpleados.SelectedIndex = -1;
        }

        private void btnGuardar_Click(object sender, EventArgs e)
        {
            DialogResult opcion = MessageBox.Show("Información guardada correctamente ¿Desea continuar?",
 "Guardar", MessageBoxButtons.YesNoCancel,
                                        MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
                if (!Guardar())
                    return;

                if (nuevo)
                {
                    InicializarDatos();
                }
                else
                {
                    CargarDatos();
                }
            }
            else if (opcion == DialogResult.No)
            {
                if (!Guardar())
                    return;

                Close();
            }
        }

        private bool Guardar()
        {
            if (!ValidarCampos())
            {
                MessageBox.Show("Llene todos los campos.");
                return false;
            }

            Empleado.Nombre = txtNombre.Text;
            Empleado.Direccion = txtDireccion.Text;
            Empleado.Colonia = txtColonia.Text;
            Empleado.CodigoPostal = txtCodigoPostal.Text;
            Empleado.Telefono = txtTelefono.Text;
            Empleado.Email = txtEmail.Text;
            Empleado.FechaNacimiento = dtpNacimiento.Value;
            Empleado.TipoEmpleado = Session.Load<TipoEmpleadoClass>(cboTiposEmpleados.SelectedValue);
            Empleado.Usuario = txtUsuario.Text;
            Empleado.Password = txtPassword.Text;

            if (openFileDialog.FileName != "openFileDialog")
            {
                Empleado.Imagen = UtilidadesImagenes.ObtenerByteArrayDeArchivo(openFileDialog.FileName);
            }
            else
            {
                if (nuevo)
                    Empleado.Imagen = UtilidadesImagenes.ImageToByteArray(Properties.Resources.image_defa
ult);
            }

            Session.Save(Empleado);
            Session.Flush();
            return true;
```

```
        }

        private bool ValidarCampos()
        {
            bool completo = true;

            if (txtNombre.Text == "")
                completo = false;

            if (txtDireccion.Text == "")
                completo = false;

            if (txtColonia.Text == "")
                completo = false;

            if (txtCodigoPostal.Text == "")
                completo = false;

            if (txtTelefono.Text == "")
                completo = false;

            if (txtEmail.Text == "")
                completo = false;

            if (cboTiposEmpleados.SelectedValue == null)
                completo = false;

            if (txtUsuario.Text == "")
                completo = false;

            if (txtPassword.Text == "")
                completo = false;

            string ImagenExtension = openFileDialog.FileName;
            ImagenExtension = UtilidadesImagenes.CortarCadena(ImagenExtension, '.');


            if (openFileDialog.FileName != "openFileDialog")
            {
                if (!Validacion.ValidarArchivoImagen(ImagenExtension))
                    completo = false;
            }

            return completo;
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            Cancelar();
        }

        private void Cancelar()
        {
            DialogResult opcion = MessageBox.Show("¿Desea guardar los cambios?", "Guardar", MessageBoxBut
tons.YesNoCancel,
                                                  MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
                if (!Guardar())
                    return;

                Close();
            }
            else if (opcion == DialogResult.No)
            {
                Close();
            }
        }

        private void FrmEmpleadoCaptura_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                Cancelar();
        }

        private void btnExaminar_Click(object sender, EventArgs e)
        {
            openFileDialog.ShowDialog(this);
        }

        private void openFileDialog_FileOk(object sender, CancelEventArgs e)
```

```
        {
            pictureBox.Image = UtilidadesImagenes.RetornarThumbnail(openFileDialog.FileName);
        }

        private void btnBuscarTipoEmpleado_Click(object sender, EventArgs e)
        {
            FrmTipoEmpleadoListado frmTipoEmpleadoListado = new FrmTipoEmpleadoListado
            {
                EsBusqueda = true,
                Session = Session
            };
            if (frmTipoEmpleadoListado.ShowDialog(this) != DialogResult.OK)
                return;

            CargarTiposEmpleados();

          cboTiposEmpleados.SelectedIndex = frmTipoEmpleadoListado.TipoEmpleadoSeleccionado;
        }

        private void btnAgregarTipoEmpleado_Click(object sender, EventArgs e)
        {
            FrmTipoEmpleadoCaptura frmTipoEmpleadoCaptura = new FrmTipoEmpleadoCaptura
            {
                Session = Session
            };
            frmTipoEmpleadoCaptura.ShowDialog(this);

            CargarTiposEmpleados();

            if (!nuevo)
                CargarDatos();
        }
    }
}
```

Interfaz para listar empleados



Interfaz 11. Listado de empleados

Esta interfaz tiene la finalidad de listar todos los empleados que hayan sido creados y que estén activos, además de las siguientes opciones:

1. Agregar; Al seleccionar esta opción abrirá la interfaz de captura correspondiente al listado, para crear un nuevo registro.

2. Modificar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de “No hay registro.”, de lo contrario abrirá la interfaz de captura correspondiente al listado para la edición del registro seleccionado.

3. Eliminar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de “No hay registro.”, de lo contrario enviará un mensaje de confirmación que diga “¿Esta seguro que desea eliminar este registro?”. Si se confirma la acción se procederá a deshabilitar el registro.

4. Actualizar; Esta opción permitirá recargar la búsqueda.

5. Campo de búsqueda; por defecto se enlistan todos los registros activos, pero si el administrador coloca en el cuadro de búsqueda un número entero y recarga la búsqueda el registro será localizado con ese número. También el administrador podrá escribir una frase y al recargar se filtrarán los campos de texto que tengan alguna coincidencia.

6. Buscar; Esta opción permite recargar la búsqueda.

7. Aceptar Búsqueda; Esta opción únicamente estará activa cuando la ventana de listado sea usada para seleccionar un registro y devuelva el mismo a otra interfaz.

La interfaz no podrá se maximizada, ni minimizada pero si podrá cambiar. Al pulsar la tecla Escape la ventana deberá cerrarse.

Windows Form

1. Name : FrmEmpleadoListado
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

ToolStrip

2. Name : tsMenu

ToolStripButton

3. Name : btnAgregar
    a. Text : &Agregar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
4. Name : btnModificar
    a. Text : &Modificar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
5. Name : btnEliminar
    a. Text : &Eliminar
    b. DisplayStyle : ImageAndText

   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
6. Name : btnActalizar
   a. Text : A&ctualizar
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
7. Name : btnBuscar
   a. Text : &Buscar
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
8. Name : btnAceptarBusqueda
   a. Text : Ace&ptar Busqueda
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración

ToolStripTextBox

9. Name : txtBuscar

DataGridView

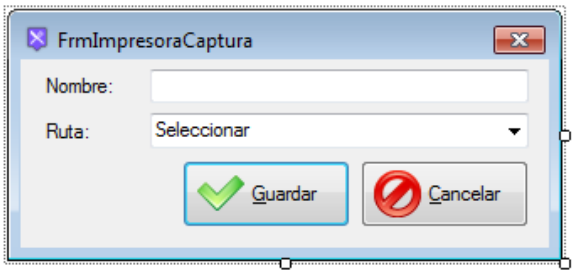
10. Name : grdEmpleados
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
       i. Height : 40
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas

```
using System;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;

namespace Restaurante.Catalogos
{
    public partial class FrmEmpleadoListado : Form
    {
```

```
public ISession Session;
public bool EsBusqueda;
public int EmpleadoSeleccionado;

public FrmEmpleadoListado()
{
    InitializeComponent();
}

private void FrmEmpleadoListado_Load(object sender, EventArgs e)
{
    if (!EsBusqueda)
    {
        Session = Conexion.CreateSessionFactory().OpenSession();

        btnAceptarBusqueda.Visible = false;
    }

    grdEmpleados.AutoGenerateColumns = false;

    CargarDatos();
}

private void CargarDatos()
{
    var lista = from c in Session.Query<EmpleadoClass>()
                where c.Activo
                select new {
                c.Id,
                c.Imagen,
                c.Nombre,
                TipoEmpleado = c.TipoEmpleado.Nombre,
                c.Telefono
                };

    if (txtBuscar.Text != "")
    {
        if (Validacion.EsEntero((txtBuscar.Text)))
            lista = lista.Where(c => c.Id == Convert.ToInt64(txtBuscar.Text));
        else
            lista = lista.Where(c => c.Nombre.Contains(txtBuscar.Text));
    }
    grdEmpleados.DataSource = lista.ToList();
}

private void btnActualizar_Click(object sender, EventArgs e)
{
    CargarDatos();
}

private void btnBuscar_Click(object sender, EventArgs e)
{
    CargarDatos();
}

private void txtBuscar_KeyDown(object sender, KeyEventArgs e)
{
    if(e.KeyCode == Keys.Return)
        CargarDatos();
}

private void btnAgregar_Click(object sender, EventArgs e)
{
    FrmEmpleadoCaptura frmEmpleadoCaptura = new FrmEmpleadoCaptura
                                            {
                                                Session = Session
                                            };
    frmEmpleadoCaptura.ShowDialog(this);
    CargarDatos();
}

private void btnModificar_Click(object sender, EventArgs e)
{
    Modificar();
}

private void Modificar()
{
    try
    {
        if (grdEmpleados.CurrentRow == null)
```

```
                    return;

                DataGridViewRow registro = grdEmpleados.CurrentRow;
                int IdRegistro = (int)registro.Cells["Id"].Value;
                EmpleadoClass Empleado = Session.Get<EmpleadoClass>(IdRegistro);

                FrmEmpleadoCaptura frmEmpleadoCaptura = new FrmEmpleadoCaptura
                {
                    Session = Session,
                    Empleado = Empleado
                };
                frmEmpleadoCaptura.ShowDialog(this);
                CargarDatos();
            }
            catch (Exception)
            {
                MessageBox.Show("No hay registro.");
            }
        }

        private void btnEliminar_Click(object sender, EventArgs e)
        {
            try
            {
                if (grdEmpleados.CurrentRow == null)
                    return;

                if (MessageBox.Show("¿Esta seguro que desea eliminar este registro?", "Eliminar",
                                    MessageBoxButtons.YesNo, MessageBoxIcon.Warning, MessageBoxDefaultBut
ton.Button2) ==
                    DialogResult.Yes){

                            DataGridViewRow registro = grdEmpleados.CurrentRow;
                            int IdRegistro = (int)registro.Cells["Id"].Value;
                            EmpleadoClass Empleado = Session.Get<EmpleadoClass>(IdRegistro);
                            Empleado.Activo = false;
                            Session.Save(Empleado);
                            Session.Flush();

                            CargarDatos();
                }
            }
            catch (Exception)
            {
                MessageBox.Show("No hay registro.");
            }
        }

        private void grdEmpleados_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
        {
            Modificar();
        }

        private void btnAceptarBusqueda_Click(object sender, EventArgs e)
        {
            if (grdEmpleados.CurrentRow == null)
                return;

            DataGridViewRow row = grdEmpleados.CurrentRow;
            EmpleadoSeleccionado = (int)row.Cells["Id"].Value;
            DialogResult = DialogResult.OK;
            Close();
        }

        private void FrmEmpleadoListado_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                Close();
        }

    }
}
```

Interfaz para capturar impresoras



Interfaz 12. Captura de impresoras

Esta interfaz permitirá al administrador capturar las impresoras que servirán para configurar en qué lugar se imprimirá un artículo al momento de ser ordenado. Tendrá un listado con las impresoras instaladas en nuestro equipo y tendrá dos opciones por defecto que son:

No imprimir; Cuando esta opción esta seleccionada no enviará la impresión del ticket.

Solo Vista Previa; Cuando esta opción esta seleccionada solo se pre visualiza el ticket.

La interfaz tendrá dos funciones agregar nuevas impresoras y modificar los datos de una impresora previamente agregada. Cuando la ventana tenga la función de agregar, el titulo deberá ser “Agregar Impresora” y si la ventana tiene la función de modificar, el titulo deberá ser “Modificar Impresora”.

La ventana será de un tamaño definido y no podrá ser ampliada. Cuando el usuario pulse la tecla Enter hará referencia a la acción del botón Guardar y cuando la tecla que se pulse sea Escape hará referencia a la acción del botón Cancelar.

Cuando el administrador pulse el botón de Guardar, se tendrá que validar que todos los campos estén llenos de lo contrario deberá enviarse el mensaje de “Llene todos los campos.”. Si todos los campos están llenos se procederá agregar o modificar los datos según sea el caso.

Al guardar los cambios se le informará que los datos han sido guardados con el mensaje de “Información guardada correctamente. ¿Desea continuar?”, que le brindará al administrador la opción seguir agregando o modificando datos según sea el caso.

Si el administrador pulsa el botón Cancelar aparecerá un mensaje de “Desea guardar los cambios”, brindando la opción de que el administrador pueda guardas sus modificaciones en caso de haber pulsado el botón erróneamente.

Windows Form

1. Name : FrmImpresoraCaptura
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

TextBox

2. Name : txtNombre
    a. Tabindex : 0

ComboBox

3. Name : cboImpresoras
    a. TabIndex : 1
    b. Text : Seleccionar

Button

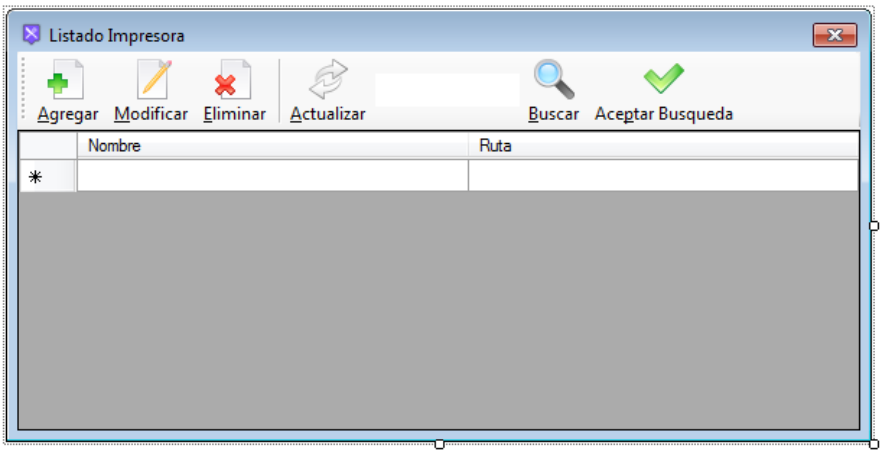
4. Name : btnGuardar
    a. Text : &Guardar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 2
    d. Width : 100
    e. Height : 40
    f. Image : A consideración
5. Name : btnCancelar
    a. Text : &Cancelar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 3
    d. Width : 100
    e. Height : 40
    f. Image : A consideración

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using Datos;
using NHibernate;

namespace Restaurante.Catalogos
{
    public partial class FrmImpresoraCaptura : Form
    {
        public ISession Session;
        public ImpresoraClass Impresora;
        private bool nuevo;

        public FrmImpresoraCaptura()
        {
            InitializeComponent();
        }

        private void FrmImpresoraCaptura_Load(object sender, EventArgs e)
        {
            CargarImpresoras();
            if (Impresora == null)
            {
                InicializarDatos();
                nuevo = true;
            }
            else
            {
                CargarDatos();
                nuevo = false;
            }
        }

        private void CargarImpresoras()
        {
            IList<ImpresoraClass> impresoras = new List<ImpresoraClass>();

            ImpresoraClass nueva;

            nueva = new ImpresoraClass();
            nueva.Nombre = "No imprimir";
            nueva.Ruta = "0";
            impresoras.Add(nueva);

            nueva = new ImpresoraClass();
            nueva.Nombre = "Solo vista previa";
            nueva.Ruta = "1";
            impresoras.Add(nueva);

            foreach (String strPrinter in System.Drawing.Printing.PrinterSettings.InstalledPrinters)
            {
                nueva = new ImpresoraClass();
                nueva.Nombre = strPrinter;
                nueva.Ruta = strPrinter;
                impresoras.Add(nueva);
            }

            cboImpresoras.DisplayMember = "Nombre";
            cboImpresoras.ValueMember = "Ruta";
            cboImpresoras.DataSource = impresoras;
            cboImpresoras.SelectedIndex = 0;
        }

        private void CargarDatos()
        {
            txtNombre.Text = Impresora.Nombre;
            try
            {
                cboImpresoras.SelectedValue = Impresora.Ruta;
            }
            catch (Exception)
            {
                cboImpresoras.SelectedIndex = 0;
            }
```

```
            this.Text = "Modificar Impresora";
        }

        private void InicializarDatos()
        {
            Impresora = new ImpresoraClass();
            Impresora.Activo = true;

            txtNombre.Text = "";

            this.Text = "Agregar Impresora";
            txtNombre.Focus();
        }

        private void btnGuardar_Click(object sender, EventArgs e)
        {
            DialogResult opcion = MessageBox.Show("Información guardada correctamente ¿Desea continuar?",
 "Guardar", MessageBoxButtons.YesNoCancel,
                                        MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
                if (!Guardar())
                    return;

                if (nuevo)
                {
                    InicializarDatos();
                }
                else
                {
                    CargarDatos();
                }
            }
            else if (opcion == DialogResult.No)
            {
                if (!Guardar())
                    return;

                Close();
            }
        }

        private bool Guardar()
        {
            if (!ValidarCampos())
            {
                MessageBox.Show("Llene todos los campos.");
                return false;
            }

            Impresora.Nombre = txtNombre.Text;
            Impresora.Ruta = (string) cboImpresoras.SelectedValue;

            Session.Save(Impresora);
            Session.Flush();
            return true;
        }

        private bool ValidarCampos()
        {
            bool completo = true;

            if (txtNombre.Text == "")
                completo = false;

            return completo;
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            Cancelar();
        }

        private void Cancelar()
        {
            DialogResult opcion = MessageBox.Show("¿Desea guardar los cambios?", "Guardar", MessageBoxBut
tons.YesNoCancel,
                                                  MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
```

```
                if (!Guardar())
                    return;

                Close();
            }
            else if (opcion == DialogResult.No)
            {
                Close();
            }
        }

        private void FrmImpresoraCaptura_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                Cancelar();
        }
    }
}
```

Interfaz para listar impresoras



Interfaz 13. Listado de impresoras

Esta interfaz tiene la finalidad de listar todas las impresoras que hayan sido creadas y que estén activas, además de las siguientes opciones:

1. Agregar; Al seleccionar esta opción abrirá la interfaz de captura correspondiente al listado, para crear un nuevo registro.

2. Modificar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de “No hay registro.”, de lo contrario abrirá la interfaz de captura correspondiente al listado para la edición del registro seleccionado.

3. Eliminar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de "No hay registro.", de lo contrario enviará un mensaje de confirmación que diga "¿Esta seguro que desea eliminar este registro?". Si se confirma la acción se procederá a deshabilitar el registro.

4. Actualizar; Esta opción permitirá recargar la búsqueda.

5. Campo de búsqueda; Por defecto se enlistan todos los registros activos, pero si el administrador coloca en el cuadro de búsqueda un número entero y recarga la búsqueda se buscará el registro con ese número. También el administrador podrá escribir una frase y al recargar se filtrarán los campos de texto que tengan alguna coincidencia.

6. Buscar; Esta opción permite recargar la búsqueda.

7. Aceptar Búsqueda; Esta opción únicamente estará activa cuando la ventana de listado sea usada para seleccionar un registro y devuelva el mismo a otra interfaz.

La interfaz no podrá se maximizada, ni minimizada pero si podrá cambiar. Al pulsar la tecla Escape la ventana deberá cerrarse.

Windows Form

1. Name : FrmImpresoraListado
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

ToolStrip

2. Name : tsMenu

ToolStripButton

3. Name : btnAgregar
    a. Text : &Agregar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
4. Name : btnModificar

   a. Text : &Modificar
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
5. Name : btnEliminar
   a. Text : &Eliminar
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
6. Name : btnActalizar
   a. Text : A&ctualizar
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
7. Name : btnBuscar
   a. Text : &Buscar
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
8. Name : btnAceptarBusqueda
   a. Text : Ace&ptar Busqueda
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración

ToolStripTextBox

9. Name : txtBuscar

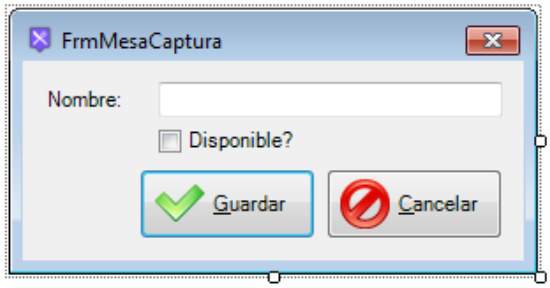
DataGridView

10. Name : grdImpresora
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas

```
using System;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;

namespace Restaurante.Catalogos
{
    public partial class FrmImpresoraListado : Form
    {
        public ISession Session;
        public bool EsBusqueda;
        public int ImpresoraSeleccionado;

        public FrmImpresoraListado()
        {
            InitializeComponent();
        }

        private void FrmImpresoraListado_Load(object sender, EventArgs e)
        {
            if (!EsBusqueda)
            {
                Session = Conexion.CreateSessionFactory().OpenSession();

                btnAceptarBusqueda.Visible = false;
            }

            grdImpresoras.AutoGenerateColumns = false;

            CargarDatos();
        }

        private void CargarDatos()
        {
            var lista = from c in Session.Query<ImpresoraClass>()
                        where c.Activo
                        select c;

            if (txtBuscar.Text != "")
            {
                if (Validacion.EsEntero((txtBuscar.Text)))
                    lista = lista.Where(c => c.Id == Convert.ToInt64(txtBuscar.Text));
                else
                    lista = lista.Where(c => c.Nombre.Contains(txtBuscar.Text));
            }
            grdImpresoras.DataSource = lista.ToList();
        }

        private void btnActualizar_Click(object sender, EventArgs e)
        {
            CargarDatos();
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            CargarDatos();
        }

        private void txtBuscar_KeyDown(object sender, KeyEventArgs e)
        {
            if(e.KeyCode == Keys.Return)
                CargarDatos();
        }

        private void btnAgregar_Click(object sender, EventArgs e)
        {
            FrmImpresoraCaptura frmImpresoraCaptura = new FrmImpresoraCaptura
                                                      {
                                                          Session = Session
                                                      };
            frmImpresoraCaptura.ShowDialog(this);
            CargarDatos();
```

```
        }

        private void btnModificar_Click(object sender, EventArgs e)
        {
            Modificar();
        }

        private void Modificar()
        {
            try
            {
                if (grdImpresoras.CurrentRow == null)
                    return;

                DataGridViewRow registro = grdImpresoras.CurrentRow;
                int IdRegistro = (int)registro.Cells["Id"].Value;
                ImpresoraClass Impresora = Session.Get<ImpresoraClass>(IdRegistro);

                FrmImpresoraCaptura frmImpresoraCaptura = new FrmImpresoraCaptura
                {
                    Session = Session,
                    Impresora = Impresora
                };
                frmImpresoraCaptura.ShowDialog(this);
                CargarDatos();
            }
            catch (Exception)
            {
                MessageBox.Show("No hay registro.");
            }
        }

        private void btnEliminar_Click(object sender, EventArgs e)
        {
            try
            {
                if (grdImpresoras.CurrentRow == null)
                    return;

                if (MessageBox.Show("¿Esta seguro que desea eliminar este registro?", "Eliminar",
                                    MessageBoxButtons.YesNo, MessageBoxIcon.Warning, MessageBoxDefaultBut
ton.Button2) ==
                    DialogResult.Yes){

                            DataGridViewRow registro = grdImpresoras.CurrentRow;
                            int IdRegistro = (int)registro.Cells["Id"].Value;
                            ImpresoraClass Impresora = Session.Get<ImpresoraClass>(IdRegistro);
                            Impresora.Activo = false;
                            Session.Save(Impresora);
                            Session.Flush();

                            CargarDatos();
                }
            }
            catch (Exception)
            {
                MessageBox.Show("No hay registro.");
            }
        }

        private void grdImpresoras_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
        {
            Modificar();
        }
         private void btnAceptarBusqueda_Click(object sender, EventArgs e)
        {
            if (grdImpresoras.CurrentRow == null)
                return;

            DataGridViewRow row = grdImpresoras.CurrentRow;
            ImpresoraSeleccionado = (int)row.Cells["Id"].Value;
            DialogResult = DialogResult.OK;
            Close();
        }
         private void FrmImpresoraListado_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                Close();
        }
    }
}
```

Interfaz para capturar mesas



Interfaz 14. Captura de mesas

Esta interfaz permitirá al administrador capturar las mesas que servirán para levantar los pedidos, así como configurar si la mesa está disponible.

La interfaz tendrá dos funciones agregar nuevas mesas y modificar los datos de una mesa previamente agregada. Cuando la ventana tenga la función de agregar, el titulo deberá ser "Agregar Mesa" y si la ventana tiene la función de modificar, el titulo deberá ser "Modificar Mesa".

La ventana será de un tamaño definido y no podrá ser ampliada. Cuando el usuario pulse la tecla Enter hará referencia a la acción del botón Guardar y cuando la tecla que se pulse sea Escape hará referencia a la acción del botón Cancelar.

Cuando el administrador pulse el botón de Guardar, se tendrá que validar que todos los campos estén llenos de lo contrario deberá enviarse el mensaje de "Llene todos los campos.". Si todos los campos están llenos se procederá agregar o modificar los datos según sea el caso.

Al guardar los cambios se le informará que los datos han sido guardados con el mensaje de "Información guardada correctamente. ¿Desea continuar?", que le brindará al administrador la opción seguir agregando o modificando datos según sea el caso.

Si el administrador pulsa el botón Cancelar aparecerá un mensaje de "Desea guardar los cambios", brindando la opción de que el administrador pueda guardas sus modificaciones en caso de haber pulsado el botón erróneamente.

Windows Form

1. Name : FrmMesaCaptura
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

TextBox

2. Name : txtNombre
    a. Tabindex : 0

CheckBox

3. Name : chkDisponibles
    g. TabIndex : 1
    h. Text : ¿Disponible?

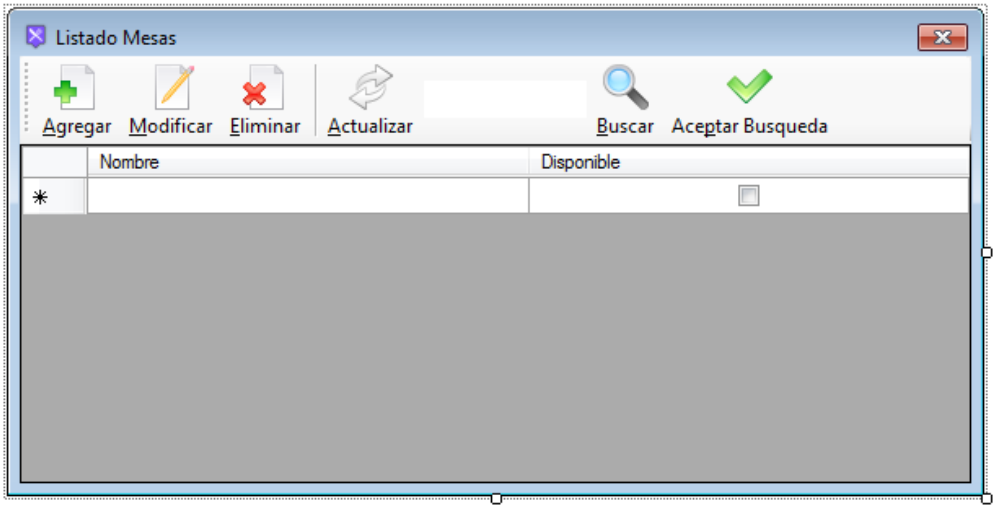
Button

6. Name : btnGuardar
    a. Text : &Guardar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 2
    d. Width : 100
    e. Height : 40
    f. Image : A consideración
7. Name : btnCancelar
    a. Text : &Cancelar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 3
    d. Width : 100
    e. Height : 40
    f. Image : A consideración

```
using System;
using System.Windows.Forms;
using Datos;
using NHibernate;

namespace Restaurante.Catalogos
{
    public partial class FrmMesaCaptura : Form
    {
        public ISession Session;
        public MesaClass Mesa;
        private bool nuevo;

        public FrmMesaCaptura()
        {
            InitializeComponent();
        }

        private void FrmMesaCaptura_Load(object sender, EventArgs e)
        {
            if (Mesa == null)
            {
                InicializarDatos();
                nuevo = true;
            }
            else
            {
                CargarDatos();
                nuevo = false;
            }
        }

        private void CargarDatos()
        {
            txtNombre.Text = Mesa.Nombre;
            chkDisponible.Checked = Mesa.Disponible;

            this.Text = "Modificar Mesa";
        }

        private void InicializarDatos()
        {
            Mesa = new MesaClass();
            Mesa.Activo = true;

            txtNombre.Text = "";
            chkDisponible.Checked = false;

            this.Text = "Agregar Mesa";
            txtNombre.Focus();
        }

        private void btnGuardar_Click(object sender, EventArgs e)
        {
            DialogResult opcion = MessageBox.Show("Información guardada correctamente ¿Desea continuar?",
 "Guardar", MessageBoxButtons.YesNoCancel,
                                        MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
                if (!Guardar())
                    return;

                if (nuevo)
                {
                    InicializarDatos();
                }
                else
                {
                    CargarDatos();
                }
            }
            else if (opcion == DialogResult.No)
            {
                if (!Guardar())
                    return;
```

```
                Close();
            }
        }

        private bool Guardar()
        {
            if (!ValidarCampos())
            {
                MessageBox.Show("Llene todos los campos.");
                return false;
            }

            Mesa.Nombre = txtNombre.Text;
            Mesa.Disponible = chkDisponible.Checked;

            Session.Save(Mesa);
            Session.Flush();
            return true;
        }

        private bool ValidarCampos()
        {
            bool completo = true;

            if (txtNombre.Text == "")
            {
                completo = false;
            }

            return completo;
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            Cancelar();
        }

        private void Cancelar()
        {
            DialogResult opcion = MessageBox.Show("¿Desea guardar los cambios?", "Guardar", MessageBoxBut
tons.YesNoCancel,
                                                  MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
                if (!Guardar())
                    return;

                Close();
            }
            else if (opcion == DialogResult.No)
            {
                Close();
            }
        }

        private void FrmMesaCaptura_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                Cancelar();
        }
    }
}
```

Interfaz para listar mesas



Interfaz 15. Listado de mesas

Esta interfaz tiene la finalidad de listar todas las mesas que hayan sido creadas y que estén activas, además de las siguientes opciones:

1. Agregar; Al seleccionar esta opción abrirá la interfaz de captura correspondiente al listado, para crear un nuevo registro.

2. Modificar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de “No hay registro.”, de lo contrario abrirá la interfaz de captura correspondiente al listado para la edición del registro seleccionado.

3. Eliminar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de “No hay registro.”, de lo contrario enviará un mensaje de confirmación que diga “¿Esta seguro que desea eliminar este registro?”. Si se confirma la acción se procederá a deshabilitar el registro.

4. Actualizar; Esta opción permitirá recargar la búsqueda.

5. Campo de búsqueda; Por defecto se enlistan todos los registros activos, pero si el administrador coloca en el cuadro de búsqueda un número entero y recarga la búsqueda

se buscará el registro con ese número. También el administrador podrá escribir una frase y al recargar se filtrarán los campos de texto que tengan alguna coincidencia.

6. Buscar; Esta opción permite recargar la búsqueda.

7. Aceptar Búsqueda; Esta opción únicamente estará activa cuando la ventana de listado sea usada para seleccionar un registro y devuelva el mismo a otra interfaz.

La interfaz no podrá se maximizada, ni minimizada pero si podrá cambiar. Al pulsar la tecla Escape la ventana deberá cerrarse.

Windows Form

1. Name : FrmMesaListado
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

ToolStrip

2. Name : tsMenu

ToolStripButton

3. Name : btnAgregar
    a. Text : &Agregar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
4. Name : btnModificar
    a. Text : &Modificar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
5. Name : btnEliminar
    a. Text : &Eliminar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText

   e. Image : A consideración
6. Name : btnActalizar
   a. Text : A&ctualizar
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
7. Name : btnBuscar
   a. Text : &Buscar
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
8. Name : btnAceptarBusqueda
   a. Text : Ace&ptar Busqueda
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración

ToolStripTextBox

9. Name : txtBuscar

DataGridView

10. Name : grdArticulos
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas

```
using System;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;

namespace Restaurante.Catalogos
{
    public partial class FrmMesaListado : Form
    {
        public ISession Session;
        public bool EsBusqueda;
        public int MesaSeleccionado;

        public FrmMesaListado()
        {
            InitializeComponent();
        }

        private void FrmMesaListado_Load(object sender, EventArgs e)
        {
            if (!EsBusqueda)
            {
                Session = Conexion.CreateSessionFactory().OpenSession();

                btnAceptarBusqueda.Visible = false;
            }

            grdMesas.AutoGenerateColumns = false;

            CargarDatos();
        }

        private void CargarDatos()
        {
            var lista = from c in Session.Query<MesaClass>()
                        where c.Activo
                        select c;

            if (txtBuscar.Text != "")
            {
                if (Validacion.EsEntero((txtBuscar.Text)))
                    lista = lista.Where(c => c.Id == Convert.ToInt64(txtBuscar.Text));
                else
                    lista = lista.Where(c => c.Nombre.Contains(txtBuscar.Text));
            }
            grdMesas.DataSource = lista.ToList();
        }

        private void btnActualizar_Click(object sender, EventArgs e)
        {
            CargarDatos();
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            CargarDatos();
        }

        private void txtBuscar_KeyDown(object sender, KeyEventArgs e)
        {
            if(e.KeyCode == Keys.Return)
                CargarDatos();
        }

        private void btnAgregar_Click(object sender, EventArgs e)
        {
            FrmMesaCaptura frmMesaCaptura = new FrmMesaCaptura
                                                {
                                                    Session = Session
                                                };
            frmMesaCaptura.ShowDialog(this);
            CargarDatos();
```

```
        }

        private void btnModificar_Click(object sender, EventArgs e)
        {
            Modificar();
        }

        private void Modificar()
        {
            try
            {
                if (grdMesas.CurrentRow == null)
                    return;

                DataGridViewRow registro = grdMesas.CurrentRow;
                int IdRegistro = (int)registro.Cells["Id"].Value;
                MesaClass Mesa = Session.Get<MesaClass>(IdRegistro);

                FrmMesaCaptura frmMesaCaptura = new FrmMesaCaptura
                {
                    Session = Session,
                    Mesa = Mesa
                };
                frmMesaCaptura.ShowDialog(this);
                CargarDatos();
            }
            catch (Exception)
            {
                MessageBox.Show("No hay registro.");
            }
        }

        private void btnEliminar_Click(object sender, EventArgs e)
        {
            try
            {
                if (grdMesas.CurrentRow == null)
                    return;
                 if (MessageBox.Show("¿Esta seguro que desea eliminar este registro?", "Eliminar",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Warning, MessageBoxDefaultBut
ton.Button2) ==
                    DialogResult.Yes){

                            DataGridViewRow registro = grdMesas.CurrentRow;
                            int IdRegistro = (int)registro.Cells["Id"].Value;
                            MesaClass Mesa = Session.Get<MesaClass>(IdRegistro);
                            Mesa.Activo = false;
                            Session.Save(Mesa);
                            Session.Flush();
                             CargarDatos();
                }
            }
            catch (Exception)
            {
                MessageBox.Show("No hay registro.");
            }
        }

        private void grdMesas_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
        {
            Modificar();
        }

        private void btnAceptarBusqueda_Click(object sender, EventArgs e)
        {
            if (grdMesas.CurrentRow == null)
                return;

            DataGridViewRow row = grdMesas.CurrentRow;
            MesaSeleccionado = (int)row.Cells["Id"].Value;
            DialogResult = DialogResult.OK;
            Close();
        }

        private void FrmMesaListado_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                Close();
        }
    }
}
```

Interfaz para capturar tipos de empleados

Interfaz 16. Captura de tipos de empleados

Esta interfaz permitirá al administrador capturar los tipos de empleados que servirán para clasificar a los empleados. Por defecto el sistema cuando inicia crea los tres tipos por defecto que son: Administrador, Cajero y Mesero.

La interfaz tendrá dos funciones agregar nuevos tipos de empleados y modificar los datos de un tipo de empleado previamente agregado. Cuando la ventana tenga la función de agregar, el titulo deberá ser “Agregar Tipo Empleado” y si la ventana tiene la función de modificar, el titulo deberá ser “Modificar Tipo Empleado”.

La ventana será de un tamaño definido y no podrá ser ampliada. Cuando el usuario pulse la tecla Enter hará referencia a la acción del botón Guardar y cuando la tecla que se pulse sea Escape hará referencia a la acción del botón Cancelar.

Cuando el administrador pulse el botón de Guardar, se tendrá que validar que todos los campos estén llenos de lo contrario deberá enviarse el mensaje de “Llene todos los campos.”. Si todos los campos están llenos se procederá agregar o modificar los datos según sea el caso.

Al guardar los cambios se le informará que los datos han sido guardados con el mensaje de “Información guardada correctamente. ¿Desea continuar?”, que le brindará al administrador la opción seguir agregando o modificando datos según sea el caso.

Si el administrador pulsa el botón Cancelar aparecerá un mensaje de “Desea guardar los cambios”, brindando la opción de que el administrador pueda guardas sus modificaciones en caso de haber pulsado el botón erróneamente.

Windows Form

1. Name : FrmTipoEmpleadoCaptura
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

TextBox

2. Name : txtNombre
    a. Tabindex : 0

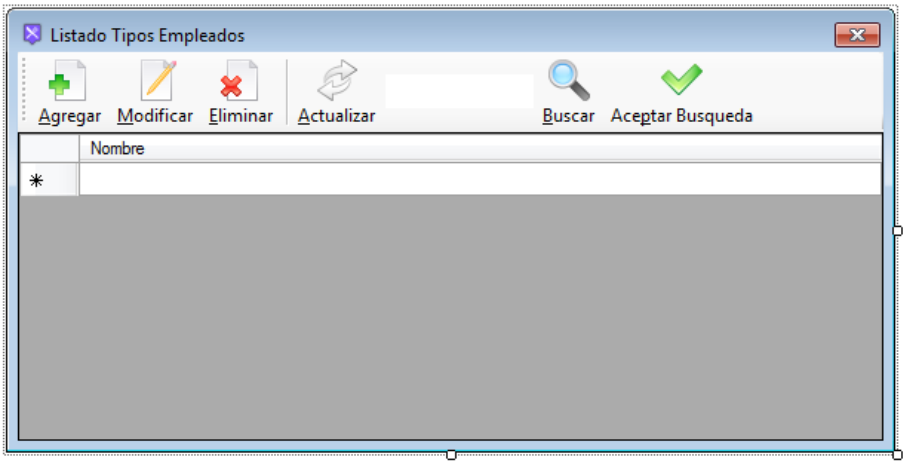
Button

3. Name : btnGuardar
    a. Text : &Guardar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 2
    d. Width : 100
    e. Height : 40
    f. Image : A consideración
4. Name : btnCancelar
    a. Text : &Cancelar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 3
    d. Width : 100
    e. Height : 40
    f. Image : A consideración

```
using System;
using System.Windows.Forms;
using Datos;
using NHibernate;

namespace Restaurante.Catalogos
{
    public partial class FrmTipoEmpleadoCaptura : Form
    {
        public ISession Session;
        public TipoEmpleadoClass TipoEmpleado;
        private bool nuevo;

        public FrmTipoEmpleadoCaptura()
        {
            InitializeComponent();
        }

        private void FrmTipoEmpleadoCaptura_Load(object sender, EventArgs e)
        {
            if (TipoEmpleado == null)
            {
                InicializarDatos();
                nuevo = true;
            }
            else
            {
                CargarDatos();
                nuevo = false;
            }
        }

        private void CargarDatos()
        {
            txtNombre.Text = TipoEmpleado.Nombre;

            this.Text = "Mofificar Tipo Empleado";
        }

        private void InicializarDatos()
        {
            TipoEmpleado = new TipoEmpleadoClass();
            TipoEmpleado.Activo = true;

            txtNombre.Text = "";

            this.Text = "Agregar Tipo Empleado";
            txtNombre.Focus();
        }

        private void btnGuardar_Click(object sender, EventArgs e)
        {
            DialogResult opcion = MessageBox.Show("Información guardada correctamente ¿Desea continuar?",
 "Guardar", MessageBoxButtons.YesNoCancel,
                                        MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
                if (!Guardar())
                    return;

                if (nuevo)
                {
                    InicializarDatos();
                }
                else
                {
                    CargarDatos();
                }
            }
            else if (opcion == DialogResult.No)
            {
                if (!Guardar())
                    return;

                Close();
```

```
            }
        }

        private bool Guardar()
        {
            if (!ValidarCampos())
            {
                MessageBox.Show("Llene todos los campos.");
                return false;
            }

            TipoEmpleado.Nombre = txtNombre.Text;

            Session.Save(TipoEmpleado);
            Session.Flush();
            return true;
        }

        private bool ValidarCampos()
        {
            bool completo = true;

            if (txtNombre.Text == "")
                completo = false;

            return completo;
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            Cancelar();
        }

        private void Cancelar()
        {
            DialogResult opcion = MessageBox.Show("¿Desea guardar los cambios?", "Guardar", MessageBoxBut
tons.YesNoCancel,
                                                  MessageBoxIcon.Question);
            if (opcion == DialogResult.Yes)
            {
                if (!Guardar())
                    return;

                Close();
            }
            else if (opcion == DialogResult.No)
            {
                Close();
            }
        }

        private void FrmTipoEmpleadoCaptura_KeyDown(object sender, KeyEventArgs e)
        {
            if(e.KeyCode == Keys.Escape)
                Cancelar();
        }
    }
}
```

Interfaz para listar tipos de empleados



Interfaz 17. Listado de tipos de empleados

Esta interfaz tiene la finalidad de listar todos los tipos de empleados que hayan sido creados y que estén activos, además de las siguientes opciones:

1. Agregar; Al seleccionar esta opción abrirá la interfaz de captura correspondiente al listado, para crear un nuevo registro.

2. Modificar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de “No hay registro.”, de lo contrario abrirá la interfaz de captura correspondiente al listado para la edición del registro seleccionado.

3. Eliminar; Al seleccionar esta opción se verificará que haya un registro seleccionado en el DataGridView de no ser así enviará el mensaje de “No hay registro.”, de lo contrario enviará un mensaje de confirmación que diga “¿Esta seguro que desea eliminar este registro?”. Si se confirma la acción se procederá a deshabilitar el registro.

4. Actualizar; Esta opción permitirá recargar la búsqueda.

5. Campo de búsqueda; Por defecto se enlistan todos los registros activos, pero si el administrador coloca en el cuadro de búsqueda un número entero y recarga la búsqueda se buscará el registro con ese número. También el administrador podrá escribir una frase y al recargar se filtrarán los campos de texto que tengan alguna coincidencia.

6. Buscar; Esta opción permite recargar la búsqueda.

7. Aceptar Búsqueda; Esta opción únicamente estará activa cuando la ventana de listado sea usada para seleccionar un registro y devuelva el mismo a otra interfaz.

La interfaz no podrá se maximizada, ni minimizada pero si podrá cambiar. Al pulsar la tecla Escape la ventana deberá cerrarse.

Windows Form

1. Name : FrmTipoEmpleadoListado
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnGuardar
    g. Icon : A consideración

ToolStrip

2. Name : tsMenu

ToolStripButton

3. Name : btnAgregar
    a. Text : &Agregar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
4. Name : btnModificar
    a. Text : &Modificar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
5. Name : btnEliminar
    a. Text : &Eliminar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración
6. Name : btnActalizar
    a. Text : A&ctualizar
    b. DisplayStyle : ImageAndText

   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
7. Name : btnBuscar
   a. Text : &Buscar
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración
8. Name : btnAceptarBusqueda
   a. Text : Ace&ptar Busqueda
   b. DisplayStyle : ImageAndText
   c. ImageScaling : None
   d. TextImageRelation : ImageAboveText
   e. Image : A consideración

ToolStripTextBox

9. Name : txtBuscar

DataGridView

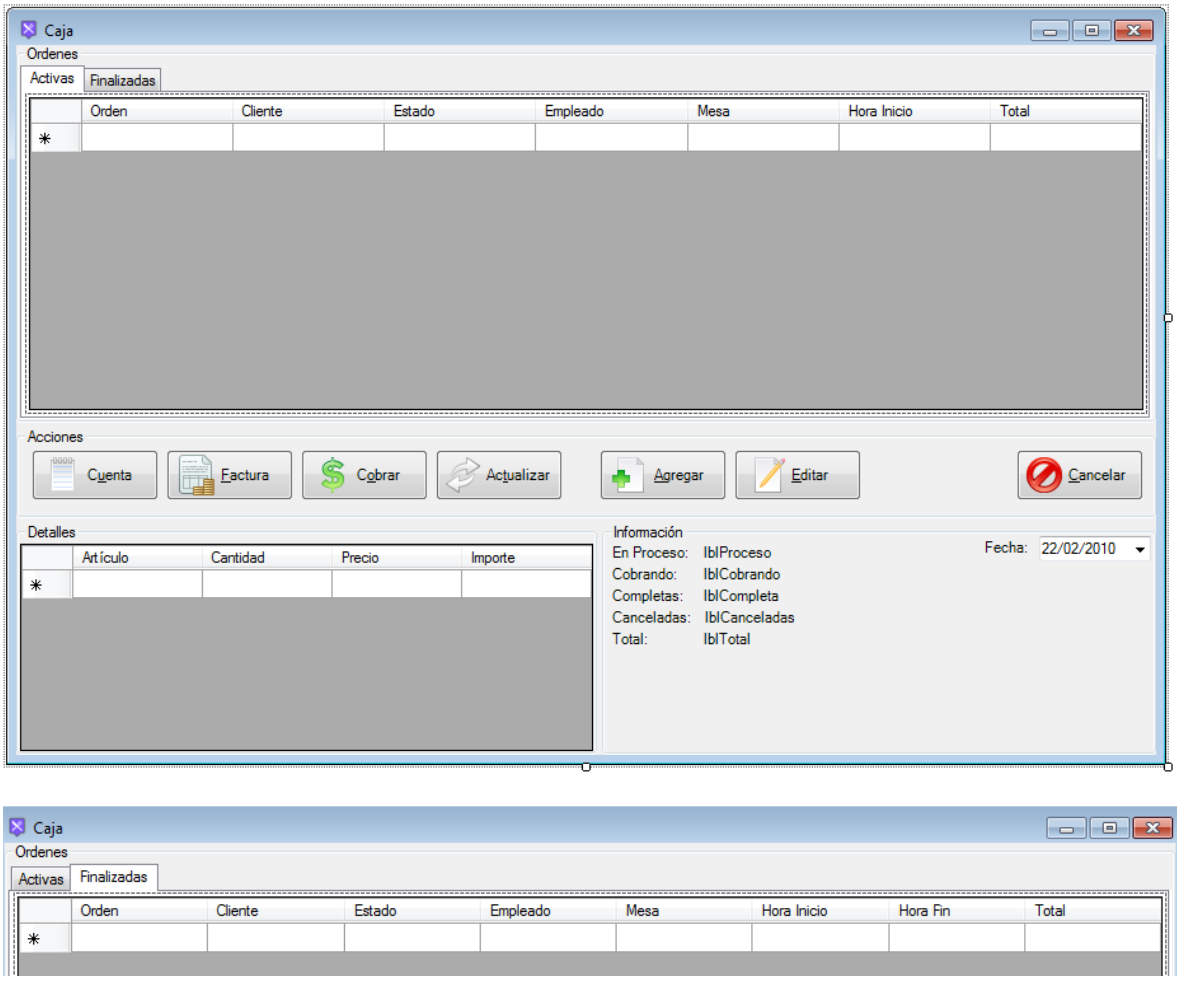
10. Name : grdArticulos
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas

```
using System;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;

namespace Restaurante.Catalogos
{
    public partial class FrmTipoEmpleadoListado : Form
    {
        public ISession Session;
        public bool EsBusqueda;
        public int TipoEmpleadoSeleccionado;

        public FrmTipoEmpleadoListado()
        {
            InitializeComponent();
        }

        private void FrmTipoEmpleadoListado_Load(object sender, EventArgs e)
        {
            if (!EsBusqueda)
            {
                Session = Conexion.CreateSessionFactory().OpenSession();

                btnAceptarBusqueda.Visible = false;
            }

            grdTiposEmpleados.AutoGenerateColumns = false;

            CargarDatos();
        }

        private void CargarDatos()
        {
            var lista = from c in Session.Query<TipoEmpleadoClass>()
                        where c.Activo
                        select c;

            if (txtBuscar.Text != "")
            {
                if (Validacion.EsEntero((txtBuscar.Text)))
                    lista = lista.Where(c => c.Id == Convert.ToInt64(txtBuscar.Text));
                else
                    lista = lista.Where(c => c.Nombre.Contains(txtBuscar.Text));
            }
            grdTiposEmpleados.DataSource = lista.ToList();
        }

        private void btnActualizar_Click(object sender, EventArgs e)
        {
            CargarDatos();
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            CargarDatos();
        }

        private void txtBuscar_KeyDown(object sender, KeyEventArgs e)
        {
            if(e.KeyCode == Keys.Return)
                CargarDatos();
        }

        private void btnAgregar_Click(object sender, EventArgs e)
        {
            FrmTipoEmpleadoCaptura frmTipoEmpleadoCaptura = new FrmTipoEmpleadoCaptura
                                                            {
                                                                Session = Session
                                                            };
            frmTipoEmpleadoCaptura.ShowDialog(this);
            CargarDatos();
```

```
        }

        private void btnModificar_Click(object sender, EventArgs e)
        {
            Modificar();
        }

        private void Modificar()
        {
            try
            {
                if (grdTiposEmpleados.CurrentRow == null)
                    return;

                DataGridViewRow registro = grdTiposEmpleados.CurrentRow;
                int IdRegistro = (int)registro.Cells["Id"].Value;
                TipoEmpleadoClass TipoEmpleado = Session.Get<TipoEmpleadoClass>(IdRegistro);

                FrmTipoEmpleadoCaptura frmTipoEmpleadoCaptura = new FrmTipoEmpleadoCaptura
                {
                    Session = Session,
                    TipoEmpleado = TipoEmpleado
                };
                frmTipoEmpleadoCaptura.ShowDialog(this);
                CargarDatos();
            }
            catch (Exception)
            {
                MessageBox.Show("No hay registro.");
            }
        }

        private void btnEliminar_Click(object sender, EventArgs e)
        {
            try
            {
                if (grdTiposEmpleados.CurrentRow == null)
                    return;

                if (MessageBox.Show("¿Esta seguro que desea eliminar este registro?", "Eliminar",
                                    MessageBoxButtons.YesNo, MessageBoxIcon.Warning, MessageBoxDefaultBut
ton.Button2) ==
                    DialogResult.Yes){

                            DataGridViewRow registro = grdTiposEmpleados.CurrentRow;
                            int idRegistro = (int)registro.Cells["Id"].Value;

                            if (idRegistro <= 3)
                            {
                                MessageBox.Show("Este afecta la integridad del sistema, no puede ser elim
onado.");
                                return;
                            }

                            TipoEmpleadoClass TipoEmpleado = Session.Get<TipoEmpleadoClass>(idRegistro);
                            TipoEmpleado.Activo = false;
                            Session.Save(TipoEmpleado);
                            Session.Flush();

                            CargarDatos();
                }
            }
            catch (Exception)
            {
                MessageBox.Show("No hay registro.");
            }
        }

        private void grdTiposEmpleados_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
        {
            Modificar();
        }

        private void btnAceptarBusqueda_Click(object sender, EventArgs e)
        {
            if (grdTiposEmpleados.CurrentRow == null)
                return;

            DataGridViewRow row = grdTiposEmpleados.CurrentRow;
            TipoEmpleadoSeleccionado = (int)row.Cells["Id"].Value;
            DialogResult = DialogResult.OK;
```

```
            Close();
        }

        private void FrmTipoEmpleadoListado_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                Close();
        }
    }
}
```

Interfaz para el cajero



Interfaz 18. Caja

Esta interfaz permitirá al cajero ver lar ordenes que están en curso y las que ya fueron finalizadas del día actual, así como el estado en que se encuentran y un resumen de los estados. Los estados por defecto son:

1. En Proceso; Mostrará las cuentas que aún están siendo atendidas.
2. Cobrando; Mostrará las cuentas que están siendo cobradas.
3. Cobradas; Mostrará las cuentas que han sido cobradas.
4. Canceladas; Mostrará las cuentas que han sido canceladas.
5. Total; Mostrará el total de cuentas que se han realizado en el día.

Al seleccionar una orden desplegara los detalles de la orden. Entre las opciones que tiene el cajero se encuentran las siguientes:

1. Cuenta; Al seleccionar esta opción se abrirá una interfaz donde el cajero vera los detalles de la orden y podrá enviar a imprimir la orden. Una vez que se solicite el ticket de cobro la orden pasa al estado de Cobrando.
2. Factura; Al seleccionar esta opción se abrirá una interfaz donde el cajero podrá seleccionar un cliente para emitir su factura. Una vez que sea impresa la factura la orden pasa al estado Cobrando.
3. Cobro; Al seleccionar esta opción se abrirá una interfaz donde el cajero capturara el monto de pago. Una vez pagada la cuenta la orden pasa al estado Completado y la mesa que estaba asignada a la orden vuelve a estar disponible.
4. Actualizar; Al seleccionar esta opción se recarga la lista de ordenes.
5. Agregar; Al seleccionar esta opción se abrirá una interfaz donde el cajero podrá capturar un pedido. Al guardar el pedido la orden tienen el estado En proceso y la mesa a la que fue asignada la orden deja de estar disponible.
6. Editar; Al seleccionar esta opción se abrirá una interfaz donde el cajero podrá editar la orden seleccionada.
7. Cancelar; Al seleccionar esta opción se abrirá una interfaz donde el administrador deberá teclear su contraseña para dar autorización de cancelación.

La interfaz tendrá un campo de fecha para poder acceder a las órdenes de días pasados.

Windows Form

1. Name : FrmCaja
    1. Text : Caja
    2. StartPosition : CenterScreen
    3. KeyPreview : True
    4. Icon : A consideración

Label

2. Name : lblProceso
    1. Text : lblProceso
3. Name : lblCobrando
    1. Text : lblCobrando
4. Name : lblCompleta
    1. Text : lblCompleta
5. Name : lblCanceladas
    1. Text : lblCanceladas
6. Name : lblTotal
    1. Text : lblTotal

DataGridView

7. Name : grdOrdenesActivas
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas
8. Name : grdOrdenesFinalizadas
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas
9. Name : grdDetalles
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas

Button

10. Name : btnCuenta
    a. Text : C&uenta
    b. TextImageRelation : ImageBeforeText
    c. Width : 100
    d. Height : 40
    1. Image : A consideración
11. Name : btnFactura
    a. Text : &Factura
    b. TextImageRelation : ImageBeforeText
    c. Width : 100
    d. Height : 40
    2. Image : A consideración
12. Name : btnCobrar
    e. Text : C&obrar
    f. TextImageRelation : ImageBeforeText
    g. Width : 100
    h. Height : 40
    3. Image : A consideración
13. Name : btnAgregar
    i. Text : &Agregar
    j. TextImageRelation : ImageBeforeText
    k. Width : 100
    l. Height : 40
    4. Image : A consideración
14. Name : btnEditar
    m. Text : &Editar
    n. TextImageRelation : ImageBeforeText
    o. Width : 100
    p. Height : 40
    5. Image : A consideración
15. Name : btnCancelar
    q. Text : C&ancelar
    r. TextImageRelation : ImageBeforeText
    s. Width : 100
    t. Height : 40
    6. Image : A consideración

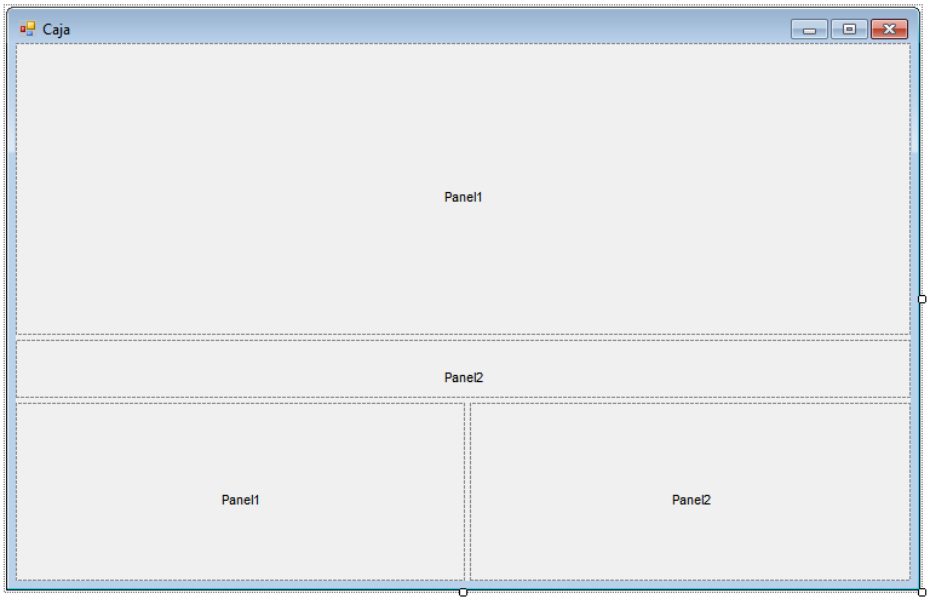
DateTimePicker

16. Name : dtpFechaActual
    a. Format : Short

GroupBox

17. Name : gpOrdenes
    a. Text : Ordenes
    b. Dock : Fill
18. Name : gpAcciones
    a. Text : Acciones
    b. Dock : Fill
19. Name : gpDetalles
    a. Text : Detalles
    b. Dock : Fill
20. Name : gpInformacion
    a. Text : Información
    b. Dock : Fill

Los GroupBox fueron colocados dentro de unos Panel que fueron acomodados como se muestra a continuación.



```
using System;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;
using Restaurante.Procesos.Orden;
using Restaurante.Reportes.Cobro;

namespace Restaurante.Procesos.Caja
{
    public partial class FrmCaja : Form
    {
        private ISession session;
        private const int ODENPROCESO = 1;
```

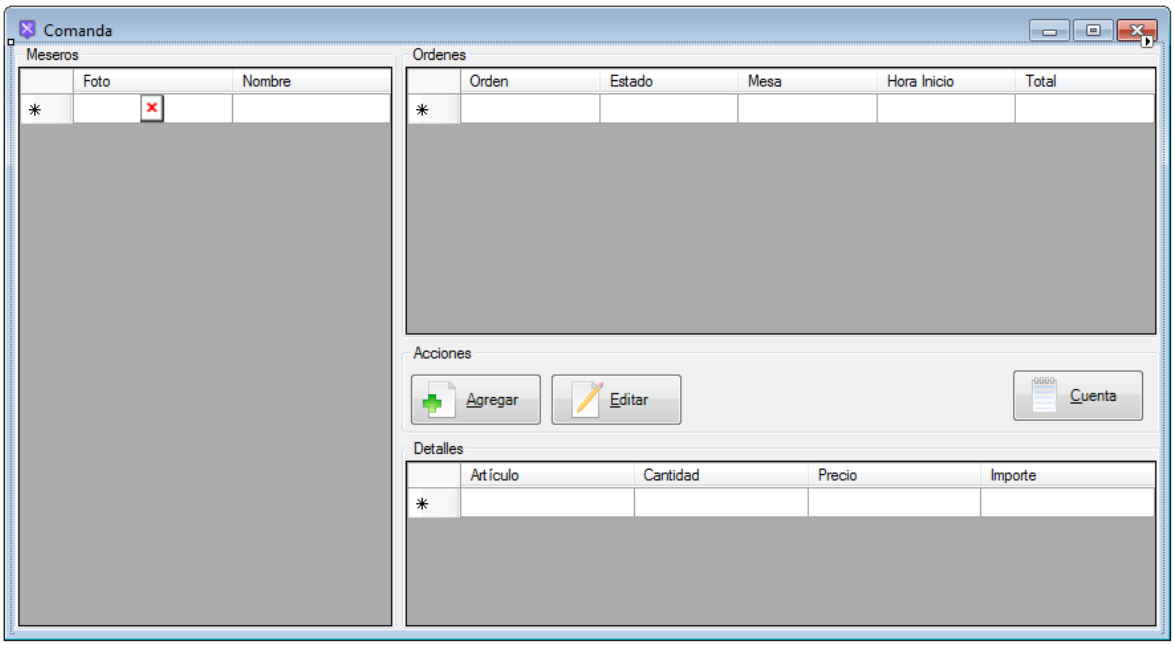
```
private const int ODENCOBRANDO= 2;
private const int ODENCOMPLETA = 3;
private const int ODENCANCELADA = 4;

public FrmCaja()
{
    InitializeComponent();
}

private void FrmCaja_Load(object sender, EventArgs e)
{
    session = Conexion.CreateSessionFactory().OpenSession();
    dtpFechaActual.Value = FunServidor.FechaServidor(session);
    grdOrdenesActivas.AutoGenerateColumns = false;
    grdOrdenesFinalizadas.AutoGenerateColumns = false;
    grdDetalles.AutoGenerateColumns = false;
    CargarOrdenes();
    CargarOrdenesFinalizadas();
    CargarInforme();
}

private void CargarInforme()
{
    int OrdenEnProceso = 0, OrdenCobrando = 0, OrdenCompleta = 0, OrdenCancelada = 0;

    var lista = from c in session.Query<OrdenClass>()
                where
                    (c.Activo && c.Fecha == dtpFechaActual.Value)
                select c;
    foreach (var x in lista)
    {
        switch (x.StatusOrden.Id)
        {
            case ODENPROCESO:
                OrdenEnProceso++;
                break;
            case ODENCOBRANDO:
                OrdenCobrando++;
                break;
            case ODENCOMPLETA:
                OrdenCompleta++;
                break;
            case ODENCANCELADA:
                OrdenCancelada++;
                break;
        }
    }
    lblProceso.Text = OrdenEnProceso.ToString();
    lblCobrando.Text = OrdenCobrando.ToString();
    lblCompleta.Text = OrdenCompleta.ToString();
    lblCanceladas.Text = OrdenCancelada.ToString();
    lblTotal.Text = lista.Count().ToString();
}

private void CargarOrdenesFinalizadas()
{
    grdOrdenesFinalizadas.DataSource = null;
    grdDetalles.DataSource = null;

    var lista = from c in session.Query<OrdenClass>()
                where
                    (c.Activo && c.StatusOrden.Id >= ODENCOMPLETA &&
                     c.Fecha == dtpFechaActual.Value)
                select c;

    grdOrdenesFinalizadas.DataSource = lista.ToList();
}

private void CargarOrdenes()
{
    grdOrdenesActivas.DataSource = null;
    grdDetalles.DataSource = null;

    var lista = from c in session.Query<OrdenClass>()
                where
                    (c.Activo && c.StatusOrden.Id < ODENCOMPLETA &&
                     c.Fecha == dtpFechaActual.Value)
                select c;

    grdOrdenesActivas.DataSource = lista.ToList();
}
```

```
private void btnFactura_Click(object sender, EventArgs e)
{
    if (grdOrdenesActivas.CurrentRow == null)
    {
        MessageBox.Show("No hay ordenes.");
        return;
    }

    DataGridViewRow registro = grdOrdenesActivas.CurrentRow;
    int idOrden = (int)registro.Cells["IdOrden"].Value;
    OrdenClass orden = session.Get<OrdenClass>(idOrden);

    FrmFacturacion frmFacturacion = new FrmFacturacion
    {
        Session = session,
        Orden = orden
    };
    frmFacturacion.ShowDialog(this);

    orden.StatusOrden = session.Load<StatusOrdenClass>(ODENCOBRANDO);
    session.Save(orden);
    session.Flush();

    grdOrdenesFinalizadas.DataSource = null;
    grdDetalles.DataSource = null;

    CargarOrdenes();
    CargarInforme();


}

private void btnCuenta_Click(object sender, EventArgs e)
{
    if (grdOrdenesActivas.CurrentRow == null)
    {
        MessageBox.Show("No hay ordenes.");
        return;
    }

    DataGridViewRow registro = grdOrdenesActivas.CurrentRow;
    int idOrden = (int)registro.Cells["IdOrden"].Value;
    OrdenClass orden = session.Get<OrdenClass>(idOrden);

    FrmReporteCobro frmReporteCobro = new FrmReporteCobro
                                      {
                                          IdOrden = orden.Id
                                      };
    frmReporteCobro.ShowDialog(this);

    orden.StatusOrden = session.Load<StatusOrdenClass>(ODENCOBRANDO);
    session.Save(orden);
    session.Flush();

    grdOrdenesFinalizadas.DataSource = null;
    grdDetalles.DataSource = null;

    CargarOrdenes();
    CargarInforme();
}

private void btnCobrar_Click(object sender, EventArgs e)
{
    if (grdOrdenesActivas.CurrentRow == null)
    {
        MessageBox.Show("No hay ordenes.");
        return;
    }

    DataGridViewRow registro = grdOrdenesActivas.CurrentRow;
    int idOrden = (int)registro.Cells["IdOrden"].Value;
    OrdenClass orden = session.Get<OrdenClass>(idOrden);

    FrmCobro frmCobro = new FrmCobro()
                        {
                            Total = orden.MontoConsumo
                        };
    frmCobro.ShowDialog(this);

    if (!frmCobro.Cobrado)
```

```
            return;

        orden.StatusOrden = session.Load<StatusOrdenClass>(ODENCOMPLETA);
        orden.HoraFin = FunServidor.HoraServidor(session);
        orden.Mesa.Disponible = true;

        session.Save(orden);
        session.Flush();

        grdOrdenesFinalizadas.DataSource = null;
        grdDetalles.DataSource = null;

        CargarOrdenes();
        CargarOrdenesFinalizadas();
        CargarInforme();
    }

    private void btnCancelar_Click(object sender, EventArgs e)
    {
        if (grdOrdenesActivas.CurrentRow == null)
        {
            MessageBox.Show("No hay ordenes.");
            return;
        }

        FrmTeclado frmTeclado = new FrmTeclado()
        {
            Session = session,
            password = true
        };
        frmTeclado.ShowDialog(this);

        if (!frmTeclado.Permiso)
            return;


        DataGridViewRow registro = grdOrdenesActivas.CurrentRow;
        int idOrden = (int)registro.Cells["IdOrden"].Value;
        OrdenClass orden = session.Get<OrdenClass>(idOrden);


        orden.StatusOrden = session.Load<StatusOrdenClass>(ODENCANCELADA);
        orden.Mesa.Disponible = true;
        orden.HoraFin = FunServidor.HoraServidor(session);

        session.Save(orden);
        session.Flush();

        grdOrdenesFinalizadas.DataSource = null;
        grdDetalles.DataSource = null;

        CargarOrdenes();
        CargarOrdenesFinalizadas();
    }

    private void grdOrdenesActivas_CellClick(object sender, DataGridViewCellEventArgs e)
    {
        CargarDetallesOrden();
    }

    private void CargarDetallesOrden()
    {
        if (grdOrdenesActivas.CurrentRow == null)
            return;

        DataGridViewRow registro = grdOrdenesActivas.CurrentRow;
        int idOrden = (int)registro.Cells["IdOrden"].Value;
        OrdenClass orden = session.Get<OrdenClass>(idOrden);

        var lista = from c in session.Query<OrdenDetalleClass>()
                    where (c.Activo && c.Orden == orden)
                    select new
                    {
                        c.Id,
                        c.Articulo,
                        c.Cantidad,
                        Precio = c.CostoUnitario,
                        c.Importe
                    };

        grdDetalles.DataSource = lista.ToList();
```

```
            gpDetalles.Text = "Detalles Orden " + orden.Id;
        }

        private void FrmCaja_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.F5)
            {
                CargarOrdenes();
                CargarOrdenesFinalizadas();
                CargarInforme();
            }

            if(e.KeyCode == Keys.Escape)
                Close();
        }

        private void btnActualizar_Click(object sender, EventArgs e)
        {
            CargarOrdenes();
            CargarOrdenesFinalizadas();
            CargarInforme();
        }

        private void btnAgregar_Click(object sender, EventArgs e)
        {
            FrmOrden frmOrden = new FrmOrden
            {
                Session = session,
                Empleado = Global.empleadoGlobal
            };
            frmOrden.ShowDialog(this);
            CargarOrdenes();
            CargarInforme();
        }

        private void btnEditar_Click(object sender, EventArgs e)
        {
            if (grdOrdenesActivas.CurrentRow == null)
            {
                MessageBox.Show("Seleccione una orden.");
                return;
            }

            DataGridViewRow registro = grdOrdenesActivas.CurrentRow;
            int idOrden = (int)registro.Cells["IdOrden"].Value;
            OrdenClass orden = session.Get<OrdenClass>(idOrden);

            if (orden.StatusOrden.Id > ODENPROCESO)
            {
                MessageBox.Show("La orden ya fue cerrada no puede ser editada.");
                return;
            }

            FrmOrden frmOrden = new FrmOrden
            {
                Session = session,
                Empleado = Global.empleadoGlobal,
                Orden = orden
            };
            frmOrden.ShowDialog(this);
            CargarOrdenes();
            CargarInforme();
        }

        private void dtpFechaActual_ValueChanged(object sender, EventArgs e)
        {
            CargarOrdenes();
            CargarOrdenesFinalizadas();
            CargarInforme();
        }

    }
}
```

Interfaz para el mesero



Interfaz 19. Comanda

Esta interfaz le permite al mesero agregar y editar las ordenes que tiene asignadas, así como imprimir la cuenta. Del lado izquierdo hay un listado de todos los meseros disponibles, cuando un mesero sea seleccionado del lado derecho superior se desplegara el listado de ordenes que tiene a su disposición y del mismo modo al seleccionar una orden se desplegaran los detalles de la misma en la parte derecha inferior.

Esta interfaz tiene las siguientes acciones:

1. Agregar; Al seleccionar esta opción se abrirá una interfaz donde el cajero podrá capturar un pedido. Al guardar el pedido la orden tienen el estado En proceso y la mesa a la que fue asignada la orden deja de estar disponible.

2. Editar; Al seleccionar esta opción se abrirá una interfaz donde el cajero podrá editar la orden seleccionada.

3. Cuenta; Al seleccionar esta opción se abrirá una interfaz donde el cajero vera los detalles de la orden y podrá enviar a imprimir la orden. Una vez que se solicite el ticket de cobro la orden pasa al estado de Cobrando.

Windows Form

1. Name : FrmComanda
    a. Text : Comanda
    b. StartPosition : CenterScreen
    c. KeyPreview : True
    d. Icon : A consideración

GroupBox

2. Name : gpMeseros
    a. Text : Meseros
    b. Dock : Fill
3. Name : gpOrdenes
    c. Text : Ordenes
    d. Dock : Fill
4. Name : gpAcciones
    e. Text : Acciones
    f. Dock : Fill
5. Name : gpDetalles
    g. Text : Detalles
    h. Dock : Fill

DataGridView

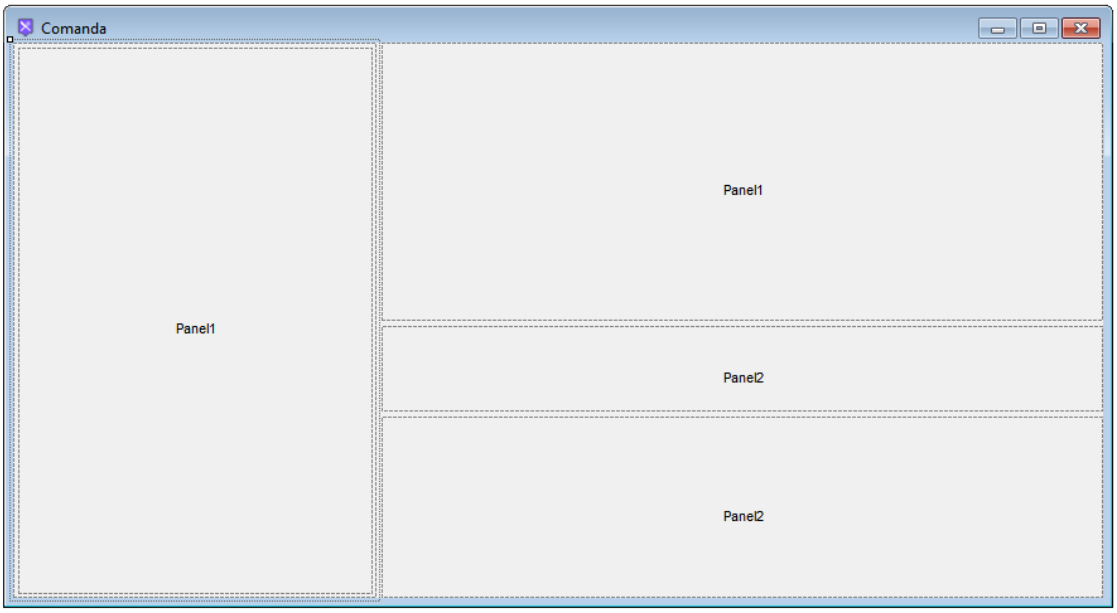
6. Name : grdMeseros
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas
7. Name : grdOrdenes
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas
8. Name : grdDetalles

a. AutoSizeColumsMode : Fill
b. Dock : Fill
c. MultiSelect : False
d. ReadOnly : True
e. RowTemplante
f. SelectionMode : FullRowSelect
g. Columns : Las indicadas

Button

9. Name : btnAgregar
    a. Text : &Agregar
    b. TextImageRelation : ImageBeforeText
    c. Width : 100
    d. Height : 40
    7. Image : A consideración
10. Name : btnEditar
    a. Text : &Editar
    b. TextImageRelation : ImageBeforeText
    c. Width : 100
    d. Height : 40
    8. Image : A consideración
11. Name : btnCuenta
    a. Text : C&uenta
    b. TextImageRelation : ImageBeforeText
    c. Width : 100
    d. Height : 40
    9. Image : A consideración

Los GroupBox fueron colocados dentro de unos Panel que fueron acomodados como se muestra a continuación.

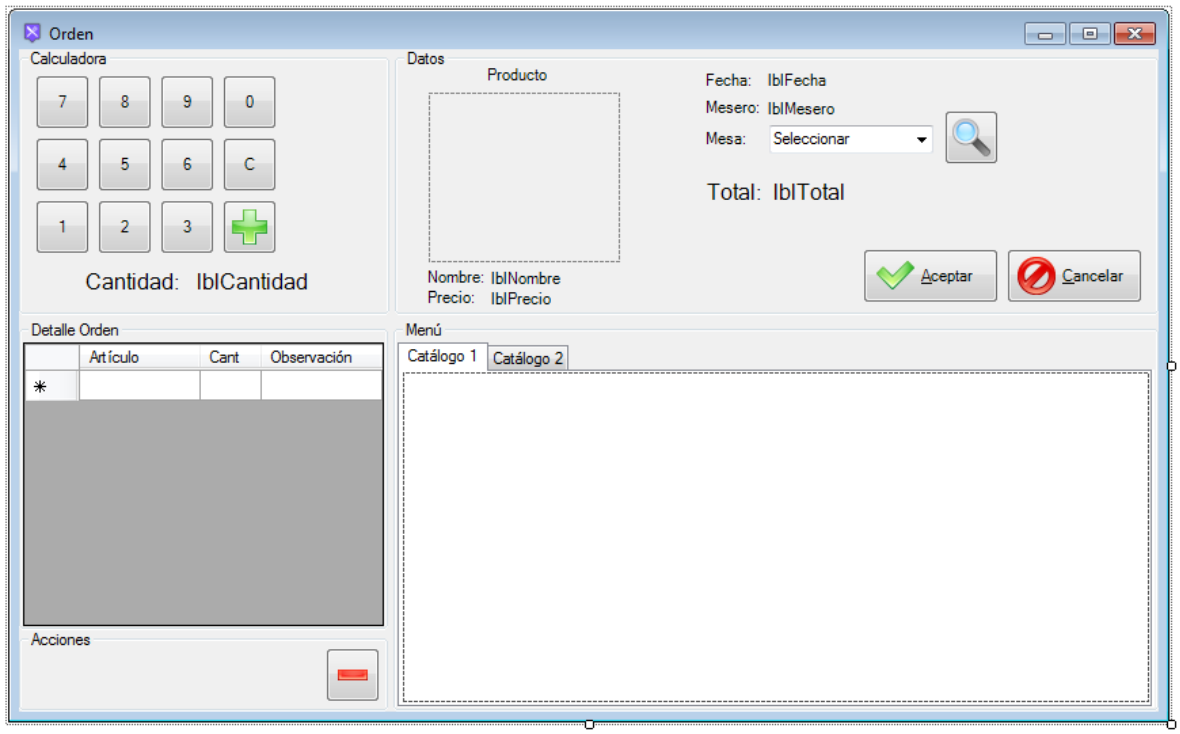


```
using System;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;
using Restaurante.Procesos.Orden;
using Restaurante.Reportes.Cobro;

namespace Restaurante.Procesos.Comanda
{
    public partial class FrmComanda : Form
    {
        private ISession session;
        private const int TIPOMESERO = 2;
        private const int ORDENPROCESO = 1;
        private const int ODENCOBRANDO = 2;
        private const int ODENCOMPLETA = 3;

        public FrmComanda()
        {
            InitializeComponent();
        }

        private void btnAgregar_Click(object sender, EventArgs e)
        {
            if (grdMeseros.CurrentRow == null)
            {
                MessageBox.Show("Seleccione un mesero.");
                return;
            }

            DataGridViewRow registro = grdMeseros.CurrentRow;
            int idMesero = (int)registro.Cells["Id"].Value;
            EmpleadoClass empelado = session.Get<EmpleadoClass>(idMesero);

            FrmOrden frmOrden = new FrmOrden
            {
                Session = session,
                Empleado = empelado
            };
            frmOrden.ShowDialog(this);
            CargarOrdenes();
        }

        private void FrmComanda_Load(object sender, EventArgs e)
        {
            session = Conexion.CreateSessionFactory().OpenSession();
            grdOrdenes.AutoGenerateColumns = false;
            grdMeseros.AutoGenerateColumns = false;
            grdDetalles.AutoGenerateColumns = false;
            CargarMeseros();
            grdMeseros.Focus();
            CargarOrdenes();
            grdOrdenes.Focus();
            CargarDetallesOrdenes();
        }

        private void CargarMeseros()
        {
            var lista = from c in session.Query<EmpleadoClass>()
                        where (c.Activo && c.TipoEmpleado.Id == TIPOMESERO)
                        select new
                        {
                            c.Id,
                            c.Nombre,
                            c.Imagen
                        };
            grdMeseros.DataSource = lista.ToList();
        }

        private void grdMeseros_CellClick(object sender, DataGridViewCellEventArgs e)
```

```
        {
            CargarOrdenes();
        }

        private void CargarOrdenes()
        {
            if (grdMeseros.CurrentRow == null)
                return;

            DataGridViewRow registro = grdMeseros.CurrentRow;
            int idMesero = (int)registro.Cells["Id"].Value;
            EmpleadoClass empleado = session.Get<EmpleadoClass>(idMesero);

            grdOrdenes.DataSource = null;
            grdDetalles.DataSource = null;

            var lista = from c in session.Query<OrdenClass>()
                        where (c.Activo && c.Empleado == empleado && c.StatusOrden.Id < ODENCOMPLETA && c
.Fecha == FunServidor.FechaServidor(session))
                        select new
                        {
                            c.Id,
                            c.StatusOrden,
                            c.Fecha,
                            c.Mesa,
                            c.HoraInicio,
                           c.MontoConsumo
                        };

            grdOrdenes.DataSource = lista.ToList();
        }

        private void grdOrdenes_CellClick(object sender, DataGridViewCellEventArgs e)
        {
            CargarDetallesOrdenes();
        }

        private void CargarDetallesOrdenes()
        {
            if (grdOrdenes.CurrentRow == null)
                return;

            DataGridViewRow registro = grdOrdenes.CurrentRow;
            int idOrden = (int)registro.Cells["IdOrden"].Value;
            OrdenClass orden = session.Get<OrdenClass>(idOrden);

            var lista = from c in session.Query<OrdenDetalleClass>()
                        where (c.Activo && c.Orden == orden)
                        select new
                        {
                            c.Id,
                            c.Articulo,
                            c.Cantidad,
                            c.Importe
                        };

            grdDetalles.DataSource = lista.ToList();
        }

        private void btnEditar_Click(object sender, EventArgs e)
        {
            Editar();
        }

        private void Editar()
        {
            if (grdOrdenes.CurrentRow == null)
            {
                MessageBox.Show("Seleccione una orden.");
                return;
            }

            DataGridViewRow registro = grdMeseros.CurrentRow;
            int idMesero = (int)registro.Cells["Id"].Value;
            EmpleadoClass empelado = session.Get<EmpleadoClass>(idMesero);

            registro = grdOrdenes.CurrentRow;
            int idOrden = (int)registro.Cells["IdOrden"].Value;
            OrdenClass orden = session.Get<OrdenClass>(idOrden);

            if (orden.StatusOrden.Id > ORDENPROCESO)
```

```
            {
                MessageBox.Show("La orden ya fue cerrada no puede ser editada.");
                return;
            }

            FrmOrden frmOrden = new FrmOrden
            {
                Session = session,
                Empleado = empelado,
                Orden = orden
            };
            frmOrden.ShowDialog(this);
            CargarOrdenes();
            CargarDetallesOrdenes();
        }

        private void btnCuenta_Click(object sender, EventArgs e)
        {
            if (grdOrdenes.CurrentRow == null)
            {
                MessageBox.Show("No hay ordenes.");
                return;
            }

            DataGridViewRow registro = grdOrdenes.CurrentRow;
            int idOrden = (int)registro.Cells["IdOrden"].Value;
            OrdenClass orden = session.Get<OrdenClass>(idOrden);

            FrmReporteCobro frmReporteCobro = new FrmReporteCobro
            {
                IdOrden = orden.Id
            };
            frmReporteCobro.ShowDialog(this);

            orden.StatusOrden = session.Load<StatusOrdenClass>(ODENCOBRANDO);
            session.Save(orden);
            session.Flush();

            grdDetalles.DataSource = null;

            CargarOrdenes();
        }

        private void grdOrdenes_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
        {
            Editar();
        }
    }
}
```

Interfaz para capturar la orden



Interfaz 20. Capturar orden

Esta interfaz es la más importante del sistema porque permite capturar el pedido del cliente. Tiene las siguientes secciones:

1. Menú; en esta sección aparecerán todos los artículos agregados en forma de botones con su imagen y nombre, que estarán divididos por categorías. Cada categoría que esta publicada aparecerá en un TabPage del TabControl. Cuando un artículo sea seleccionado aparecerá su imagen, nombre y precio en la sección de Datos.

2. Datos; en esta sección el usuario podrá visualizar los datos del artículo en selección, la fecha actual, su nombre, las mesas disponibles y el total de la orden.

3. Calculadora; En esta sección el usuario colocara la cantidad de artículos que desea agregar a la orden y pulsando sobre el botón de más el articulo se agregará a la lista de detalles. La calculadora tendrá un uso similar a la calculadora digital.

4. Detalles Orden; esta sección mostrará los artículos que se han agregado a la orden. También al pulsar doble clic sobre un artículo se abrirá una interfaz donde se podrán agregar comentarios al detalle, antes de ser enviado a impresión. Si un detalle ya fue impreso no podrá ser editado, ni eliminado sin autorización del administrador.

5. Acciones; esta sección permitirá quitar detalles de la orden, pero si el detalle ya fue impreso necesitará autorización del administrador.

También cuenta con las acciones siguientes:

1. Seccionar mesa; esta acción abrirá una interfaz donde el usuario podrá elegir una mesa disponible por medio de botones.

2. Aceptar; esta acción creará una orden nueva o actualizara los datos de la orden que se esté editando. Envía a impresión los artículos a la impresora que tenga predeterminado cada uno y pone como no disponible la mesa que se ha seleccionado.

3. Cancelar; cerrará la interfaz sin actualizar los cambios.

Windows Form

1. Name : FrmOrden
    a. Text : Orden
    b. StartPosition : CenterScreen
    c. KeyPreview : True
    d. Icon : A consideración

GroupBox

2. Name : gpCalculadora
    a. Text : Calculadora
    b. Dock : Fill
3. Name : gpDatos
    a. Text : Datos
    b. Dock : Fill
4. Name : gpDetalles
    c. Text : Detalles Orden
    d. Dock : Fill
5. Name : gpMenu
    a. Text : Menú
    b. Dock : Fill
6. Name : gpAcciones
    a. Text : Acciones
    b. Dock : Fill

DataGridView

7. Name : grdDetalles
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas

Button

Para los números del 0 al 9

8. Name : btnNumero
    a. Text : Numero
    b. TextImageRelation : Overlay
    c. Width : 40
    d. Height : 40

9. Name : btnC
    a. Text : C
    b. TextImageRelation : Overlay
    c. Width : 40
    d. Height : 40
10. Name : btnAgregar
    a. Text : ninguno
    b. TextImageRelation : Overlay
    c. Width : 40
    d. Height : 40
    e. Imagen : A consideraión
11. Name : btnQuitar
    a. Text : ninguno
    b. TextImageRelation : Overlay
    c. Width : 40
    d. Height : 40
    e. Imagen : A consideraión
12. Name : btnSeleccionaMesa
    a. Text : ninguno
    b. TextImageRelation : Overlay
    c. Width : 40
    d. Height : 40
    e. Imagen : A consideraión
22. btnAceptar
    a. Text : &Aceptar
    b. TextImageRelation : ImageBeforeText
    c. Width : 100
    d. Height : 40

e. Image : A consideración

23. Name : btnCancelar
    a. Text : &Cancelar
    b. TextImageRelation : ImageBeforeText
    c. Width : 100
    d. Height : 40
    e. Image : A consideración

PictureBox

24. Name : pictureBox
    a. SizeMode : Zoom
    b. Image : A consideración

OpenFileDialog

25. Name : openFileDialog
    a. FileName : openFileDialog

Label

26. Name : lblCantidad
    a. Text : lblCantidad
27. Name : lblNombre
    a. Text : lblNombre
28. Name : lblPrecio
    a. Text : lblPrecio
29. Name : lblFecha
    a. lblFecha
30. Name : lblMesero
    a. lblMesero
31. Name : lblTotal
    a. lblTotal

TabControl

32. Name : tbMenu
    a. Dock : Fill

ComboBox

33. Name : cboMesas
    a. Text : Seleccionar

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;
using Restaurante.Reportes.Comanda;

namespace Restaurante.Procesos.Orden
{
    public partial class FrmOrden : Form
    {
        public ISession Session;
        private OrdenDetalleTemporalClass articuloSelescionado;
        private IList<OrdenDetalleTemporalClass> listaOrdenInicial;
        private IList<OrdenDetalleTemporalClass> listaOrden;
        private int cont = 0;
        public OrdenClass Orden;
        public EmpleadoClass Empleado;
        private bool nuevo;

        public FrmOrden()
        {
            InitializeComponent();
        }

        private void FrmOrden_Load(object sender, EventArgs e)
        {
            grdDetalles.AutoGenerateColumns = false;
            lblCantidad.Text = "0";
            articuloSelescionado = new OrdenDetalleTemporalClass();
            lblFecha.Text = DateTime.Today.ToString("dd/MM/yyyy");
            CargarMesas();
            CargarMenu();
            if (Orden == null)
            {
                nuevo = true;
                InicializarDatos();
            }
            else
            {
                Empleado = Orden.Empleado;
                nuevo = false;
                CargarDatos();
            }
            lblMesero.Text = Empleado.Nombre;
        }

        private void CargarDatos()
        {
            listaOrden = new List<OrdenDetalleTemporalClass>();
            listaOrdenInicial = new List<OrdenDetalleTemporalClass>();


            cboMesas.SelectedValue = Orden.Mesa != null ? (object)Orden.Mesa.Id : null;
            cboMesas.Enabled = false;
            lblFecha.Text = Orden.Fecha.ToString("dd/MM/yyyy");

            var lista = from c in Session.Query<OrdenDetalleClass>()
                        where (c.Activo && c.Orden == Orden)
                        select c;

            foreach (var m in lista)
            {
                articuloSelescionado = new OrdenDetalleTemporalClass();
                articuloSelescionado.Nuevo = false;
                articuloSelescionado.Activo = true;
                articuloSelescionado.IdLista = cont;
                articuloSelescionado.Id = m.Id;
                articuloSelescionado.Cantidad = m.Cantidad;
                articuloSelescionado.Importe = m.Importe;
```

```
            articuloSelescionado.CostoUnitario = m.CostoUnitario;
            articuloSelescionado.Articulo = m.Articulo;
            articuloSelescionado.Observaciones = m.Observaciones;
            cont++;

            listaOrden.Add(articuloSelescionado);
            listaOrdenInicial.Add(articuloSelescionado);
        }

        grdDetalles.DataSource = null;
        grdDetalles.DataSource = listaOrden;

        lblTotal.Text = CalcularTotal().ToString("c2");
        articuloSelescionado = new OrdenDetalleTemporalClass();
    }

    private void InicializarDatos()
    {
        listaOrden = new List<OrdenDetalleTemporalClass>();
        listaOrdenInicial = new List<OrdenDetalleTemporalClass>();
        grdDetalles.DataSource = null;
    }

    private void CargarMenu()
    {
        int columna, maxcol = 5;
        int size, espacio = 10;
        int fila;
        int largo = tbMenu.Width;
        largo = largo - espacio * maxcol - espacio * 2;
        size = largo / maxcol;

        var lista = from c in Session.Query<CategoriaClass>()
                    where (c.Activo && c.Publicado)
                    select c;

        tbMenu.TabPages.Clear();


        foreach (var x in lista)
        {
            columna = 0;
            fila = 0;

            TabPage tabPage = new TabPage();
            tabPage.Text = x.Nombre;

            Font font = new Font("Arial", 12);
            tabPage.Font = font;

            var aticulossCategoria = from c in Session.Query<ArticuloClass>()
                                     where (c.Activo && c.Categoria == x)
                                     select c;
            foreach (var t in aticulossCategoria)
            {
                if (columna == maxcol)
                {
                    columna = 0;
                    fila++;
                }

                PictureBox pictureBox = new PictureBox();
                pictureBox.Height = (int)(size * 0.7);
                pictureBox.Width = (int)(size * 0.8);
                pictureBox.Location = new Point((int)(size * 0.1), (int)(size * 0.1));
                pictureBox.SizeMode = PictureBoxSizeMode.Zoom;
                pictureBox.Enabled = false;
                if (t.Imagen == null)
                {
                    pictureBox.Image = Properties.Resources.image_default;
                }
                else
                {
                    pictureBox.Image = UtilidadesImagenes.ByteArrayToImage(t.Imagen);
                }

                font = new Font("Arial", 8);

                Button nuevoBoton = new Button();

                nuevoBoton.Height = size;
```

```
                    nuevoBoton.Width = size;
                    nuevoBoton.Text = t.Nombre;
                    nuevoBoton.Name = t.Id.ToString();
                    nuevoBoton.Font = font;
                    nuevoBoton.TextImageRelation = TextImageRelation.ImageAboveText;
                    nuevoBoton.TextAlign = ContentAlignment.BottomCenter;
                    nuevoBoton.Click += new EventHandler(clicProducto);
                    nuevoBoton.Location = new Point(size * columna + columna * espacio + espacio, size *
fila + fila * espacio + espacio);

                    nuevoBoton.Controls.Add(pictureBox);


                    tabPage.Controls.Add(nuevoBoton);

                    columna++;
                }
                tbMenu.TabPages.Add(tabPage);
            }
        }

        private void clicProducto(object sender, EventArgs e)
        {
            Button producto = (Button)sender;

            lblNombre.Text = producto.Text;
            int idArticulo = Convert.ToInt32(producto.Name);

            ArticuloClass articulo = Session.Load<ArticuloClass>(idArticulo);

            articuloSelescionado.Articulo = articulo;
            articuloSelescionado.CostoUnitario = articulo.Precio;
            articuloSelescionado.Impresora = articulo.Impresora;

            lblPrecio.Text = articulo.Precio.ToString("c2");

            if (articulo.Imagen == null)
                {
                    pictureBox.Image = Properties.Resources.image_default;
                }
                else
                {
                    pictureBox.Image = UtilidadesImagenes.ByteArrayToImage(articulo.Imagen);
                }

            btnAgregar.Focus();
        }

        private void CargarMesas()
        {

            var lista = from c in Session.Query<MesaClass>()
                        where (c.Activo && c.Disponible)
                        select c;

            if (Orden != null)
            {
                lista = from c in Session.Query<MesaClass>()
                        where c.Activo
                        select c;
            }

            cboMesas.DisplayMember = "Nombre";
            cboMesas.ValueMember = "Id";
            cboMesas.DataSource = lista.ToList();

            cboMesas.SelectedValue = -1;
        }

        private void btn1_Click(object sender, EventArgs e)
        {
            if (lblCantidad.Text == "0")
                lblCantidad.Text = "";

            lblCantidad.Text += "1";
        }

        private void btn2_Click(object sender, EventArgs e)
        {
            if (lblCantidad.Text == "0")
                lblCantidad.Text = "";
```

```
    lblCantidad.Text += "2";
}

private void btn3_Click(object sender, EventArgs e)
{
    if (lblCantidad.Text == "0")
        lblCantidad.Text = "";

    lblCantidad.Text += "3";
}

private void btn4_Click(object sender, EventArgs e)
{
    if (lblCantidad.Text == "0")
        lblCantidad.Text = "";

    lblCantidad.Text += "4";
}

private void btn5_Click(object sender, EventArgs e)
{
    if (lblCantidad.Text == "0")
        lblCantidad.Text = "";

    lblCantidad.Text += "5";
}

private void btn6_Click(object sender, EventArgs e)
{
    if (lblCantidad.Text == "0")
        lblCantidad.Text = "";

    lblCantidad.Text += "6";
}

private void btn7_Click(object sender, EventArgs e)
{
    if (lblCantidad.Text == "0")
        lblCantidad.Text = "";

    lblCantidad.Text += "7";
}

private void btn8_Click(object sender, EventArgs e)
{
    if (lblCantidad.Text == "0")
        lblCantidad.Text = "";

    lblCantidad.Text += "8";
}

private void btn9_Click(object sender, EventArgs e)
{
    if (lblCantidad.Text == "0")
        lblCantidad.Text = "";

    lblCantidad.Text += "9";
}

private void btn0_Click(object sender, EventArgs e)
{
    if (lblCantidad.Text == "0")
        lblCantidad.Text = "";

    lblCantidad.Text += "0";
}

private void btnC_Click(object sender, EventArgs e)
{
    lblCantidad.Text = "0";
}

private void FrmOrden_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.NumPad0)
        btn0_Click(sender, e);

    if (e.KeyCode == Keys.NumPad1)
        btn1_Click(sender, e);
```

```
            if (e.KeyCode == Keys.NumPad2)
                btn2_Click(sender, e);

            if (e.KeyCode == Keys.NumPad3)
                btn3_Click(sender, e);

            if (e.KeyCode == Keys.NumPad4)
                btn4_Click(sender, e);

            if (e.KeyCode == Keys.NumPad5)
                btn5_Click(sender, e);

            if (e.KeyCode == Keys.NumPad6)
                btn6_Click(sender, e);

            if (e.KeyCode == Keys.NumPad7)
                btn7_Click(sender, e);

            if (e.KeyCode == Keys.NumPad8)
                btn8_Click(sender, e);

            if (e.KeyCode == Keys.NumPad9)
                btn9_Click(sender, e);

            if (e.KeyCode == Keys.Escape)
                btnC_Click(sender, e);

            if (e.KeyCode == Keys.Return)
                btnAgregar_Click(sender, e);
        }

        private void btnAgregar_Click(object sender, EventArgs e)
        {
            if (Convert.ToInt64(lblCantidad.Text) <= 0)
            {
                MessageBox.Show("La cantidad debe ser mayor a 0.");
                return;
            }

            if (articuloSelescionado.Articulo.Id == 0)
            {
                MessageBox.Show("Selesccione un producto.");
                return;
            }

            OrdenDetalleTemporalClass nuevaOrdenDesglose = new OrdenDetalleTemporalClass();
            nuevaOrdenDesglose.Id = articuloSelescionado.Id;
            nuevaOrdenDesglose.Articulo = articuloSelescionado.Articulo;
            nuevaOrdenDesglose.Activo = true;
            nuevaOrdenDesglose.Nuevo = true;
            nuevaOrdenDesglose.IdLista = cont;
            nuevaOrdenDesglose.Cantidad = (int) Convert.ToInt64(lblCantidad.Text);
            nuevaOrdenDesglose.CostoUnitario = articuloSelescionado.Articulo.Precio;
            nuevaOrdenDesglose.Importe = nuevaOrdenDesglose.Cantidad * articuloSelescionado.Articulo.Prec
io;
            nuevaOrdenDesglose.Impresora = articuloSelescionado.Impresora;
            nuevaOrdenDesglose.Observaciones = articuloSelescionado.Observaciones;

            listaOrden.Add(nuevaOrdenDesglose);

            grdDetalles.DataSource = null;
            grdDetalles.DataSource = listaOrden;

            cont++;
            lblCantidad.Text = "0";
            lblTotal.Text = CalcularTotal().ToString("c2");
        }

        private decimal CalcularTotal()
        {
            decimal Total = 0;
            foreach (var x in listaOrden)
            {
                Total += x.Importe;
            }
            return Total;
        }

        private void btnQuitar_Click(object sender, EventArgs e)
        {
            try
```

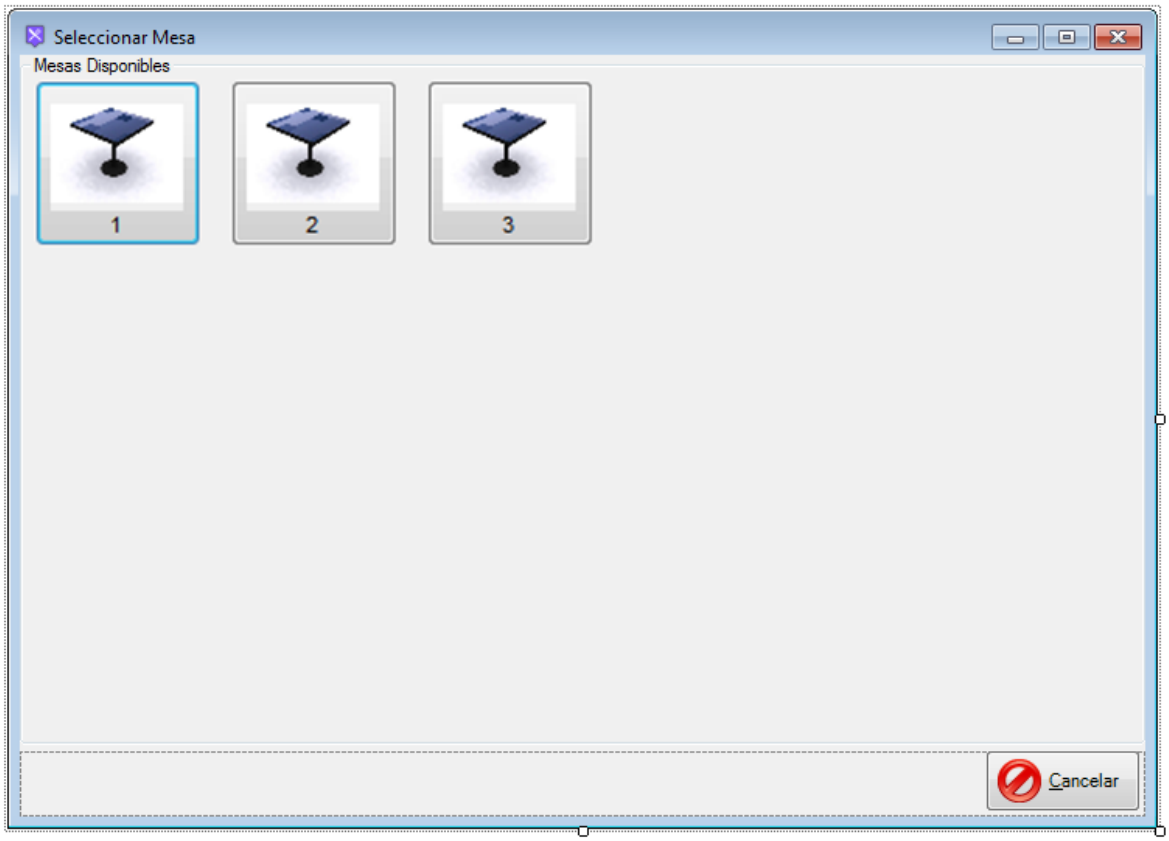
```
        {
            if (grdDetalles.CurrentRow == null)
                return;

            DataGridViewRow registro = grdDetalles.CurrentRow;
            int idDesglose = (int)registro.Cells["IdLista"].Value;

            bool ProductoNuevo = (bool)registro.Cells["ProductoNuevo"].Value;

            if (!ProductoNuevo)
            {
                FrmTeclado frmTeclado = new FrmTeclado {
                                                        Session = Session,
                                                        password = true
                                                        };
                frmTeclado.ShowDialog(this);

                if(!frmTeclado.Permiso)
                    return;
            }

            foreach (var g in listaOrden)
            {
                if (g.IdLista == idDesglose)
                {
                    listaOrden.Remove(g);

                    break;
                }
            }

            grdDetalles.DataSource = null;
            grdDetalles.DataSource = listaOrden;

            lblTotal.Text = CalcularTotal().ToString("c2");
        }
        catch (Exception ex)
        {
            MessageBox.Show("No hay registro." + ex.ToString());
        }
    }

    private void btnCancelar_Click(object sender, EventArgs e)
    {
        Close();
    }

    private void btnAceptar_Click(object sender, EventArgs e)
    {
            if (!Guardar())
                return;

            Close();
    }

    private bool Guardar()
    {
        if (!ValidarCampos() && nuevo)
        {
            MessageBox.Show("Seleccione una mesa.");
            return false;
        }

        if (listaOrden.Count == 0)
        {
            MessageBox.Show("No hay ningun movimiento en la lista.");
            return false;
        }


        if (nuevo)
        {
            Orden = new OrdenClass();
            Orden.Activo = true;
            Orden.StatusOrden = Session.Get<StatusOrdenClass>(1);
            MesaClass mesa = Session.Load<MesaClass>(cboMesas.SelectedValue);
            mesa.Disponible = false;
            Session.Save(mesa);
            Orden.Mesa = mesa;
            Orden.Fecha = FunServidor.FechaServidor(Session);
            Orden.HoraInicio = FunServidor.HoraServidor(Session);
```

```
            Orden.HoraFin = DateTime.Today;
        }

        Orden.MontoConsumo = CalcularTotal();
        Orden.Empleado = Empleado;
        Session.Save(Orden);
        Session.Flush();

        ActualizarDesglsoe();
        ValidarListaDatosIniciales();
        ImprimirComanda();
        return true;
    }

    private void ImprimirComanda()
    {
        var lista = from c in Session.Query<OrdenDetalleClass>()
                    where (c.Activo && c.Orden == Orden && c.Impreso == false)
                    orderby c.Articulo.Impresora
                    select c;

        if(lista.Count() <= 0)
            return;

        int IdImpresora = 0;
        foreach (var x in lista)
        {
            if (IdImpresora != x.Articulo.Impresora.Id)
            {
                IdImpresora = x.Articulo.Impresora.Id;

                if (x.Articulo.Impresora.Ruta == "0")
                {

                }
                else if (x.Articulo.Impresora.Ruta == "1")
                {
                    FrmReporteComanda frmReporteComanda = new FrmReporteComanda
                                                          {
                                                              IdOrden = Orden.Id,
                                                              IdImpresora = IdImpresora
                                                          };
                    frmReporteComanda.ShowDialog(this);
                }
                else
                {
                     Impresion impresion = new Impresion();
                     impresion.ImprimirTickets(Orden.Id,x.Articulo.Impresora);
                }

            }
            x.Impreso = true;
            Session.Save(x);
        }
        Session.Flush();
    }

    private void ValidarListaDatosIniciales()
    {
        if (listaOrdenInicial.Count == 0)
            return;

        bool igual;
        foreach (var i in listaOrdenInicial)
        {
            igual = false;
            foreach (var d in listaOrden)
            {
                if (i.IdLista == d.IdLista)
                {
                    igual = true;
                }
            }

            //Si fue eliminado
            if (!igual)
            {
                OrdenDetalleClass desgloseEliminado =
                    Session.Load<OrdenDetalleClass>(i.Id);

                desgloseEliminado.Activo = false;
```

```
                Session.Save(desgloseEliminado);
            }
        }
        Session.Flush();
    }

    private void ActualizarDesglsoe()
    {
        foreach (OrdenDetalleTemporalClass h in listaOrden)
        {
            if (h.Nuevo)
            {
                OrdenDetalleClass desgloseNuevo = new OrdenDetalleClass();
                desgloseNuevo.Articulo = h.Articulo;
                desgloseNuevo.Cantidad = h.Cantidad;
                desgloseNuevo.CostoUnitario = h.CostoUnitario;
                desgloseNuevo.Importe = h.Importe;
                desgloseNuevo.Activo = h.Activo;
                desgloseNuevo.Observaciones = h.Observaciones;
              //  desgloseNuevo.Impresora = h.Impresora;
                desgloseNuevo.Orden = Orden;

                Session.Save(desgloseNuevo);
            }
        }
        Session.Flush();
    }

    private bool ValidarCampos()
    {
        bool completo = true;

        if (cboMesas.SelectedValue == null)
            completo = false;

        return completo;
    }

    private void grdDetalles_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
    {
        try
        {
            if (grdDetalles.CurrentRow == null)
                return;

            DataGridViewRow registro = grdDetalles.CurrentRow;
            int idDesglose = (int)registro.Cells["IdLista"].Value;

            bool ProductoNuevo = (bool)registro.Cells["ProductoNuevo"].Value;

            if (ProductoNuevo)
            {
                FrmTeclado frmTeclado = new FrmTeclado();
                frmTeclado.ShowDialog(this);

                foreach (var g in listaOrden)
                {
                    if (g.IdLista == idDesglose)
                    {
                        g.Observaciones = frmTeclado.Observacion;

                        break;
                    }
                }
            }

            grdDetalles.DataSource = null;
            grdDetalles.DataSource = listaOrden;

            lblTotal.Text = CalcularTotal().ToString("c2");
        }
        catch (Exception ex)
        {
            MessageBox.Show("No hay registro." + ex.ToString());
        }
    }

    private void btnSeleccionarMesa_Click(object sender, EventArgs e)
    {
        FrmSeleccionarMesa frmSeleccionarMesa = new FrmSeleccionarMesa {
                                                                        Session = Session
```

```
                                                                    };
            frmSeleccionarMesa.ShowDialog(this);

            cboMesas.SelectedValue = frmSeleccionarMesa.MesaSeleccionada;
        }
    }
}
```

Interfaz para seleccionar mesas



Interfaz 21. Selección de mesa

Debido a que el sistema ha sido planeado tanto para pantallas normales como táctiles, esta interfaz permite seleccionar una mesa disponible de forma sencilla. Al seleccionar una mesa se cerrará interfaz y se selecionará la mesa en cuestión en la interfaz de Orden.

Windows Form

1. Name : FrmSeleccionarMesa
    a. Text : Seleccionar Mesa
    b. StartPosition : CenterScreen
    c. KeyPreview : True
    d. Icon : A consideración

GroupBox

2. Name : gpMesas
    a. Text : Mesas Disponibles
    b. Dock : Top

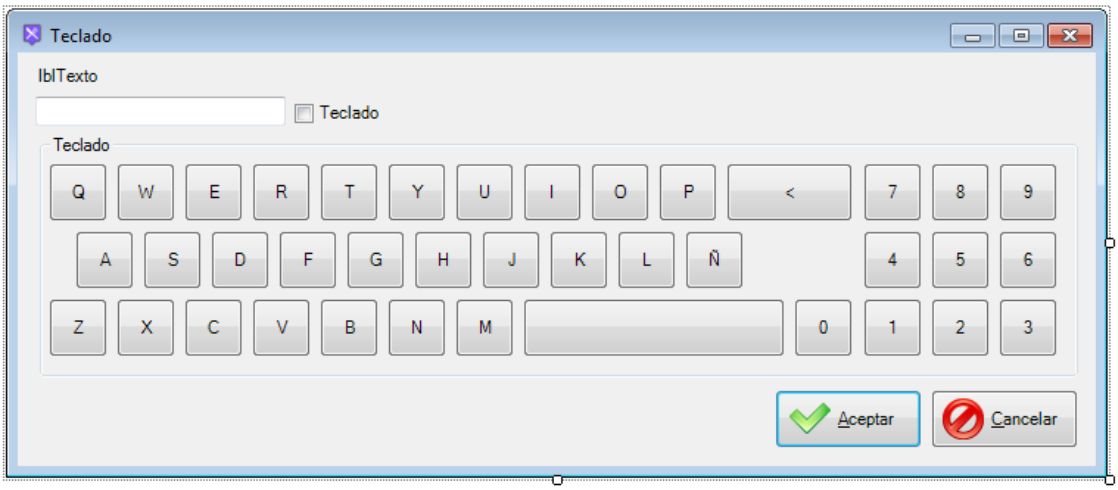
Button

3. Name : btnCancelar
    a. Text : &Cancelar
    b. TextImageRelation : ImageBeforeText
    c. Width : 100
    d. Height : 40
    e. Image : A consideración

```
using System;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using NHibernate;
using NHibernate.Linq;
using Datos;

namespace Restaurante.Procesos.Orden
{
    public partial class FrmSeleccionarMesa : Form
    {
        public ISession Session;
        public int MesaSeleccionada;

        public FrmSeleccionarMesa()
        {
            InitializeComponent();
        }

        private void FrmSeleccionarMesa_Load(object sender, EventArgs e)
        {
            CargarMesas();
        }

        private void CargarMesas()
        {
            int columna, maxcol = 7;
            int size, espacio = 15;
            int fila;
            int largo = gpMesas.Width;
            largo = largo - espacio * maxcol - espacio * 2;
            size = largo / maxcol;

             var lista = from c in Session.Query<MesaClass>()
                         where (c.Activo && c.Disponible)
                         select c;
             columna = 0;
             fila = 0;
             foreach (var x in lista)
```

```
            {
                if (columna == maxcol)
                {
                    columna = 0;
                    fila++;
                }

                PictureBox pictureBox = new PictureBox();
                pictureBox.Height = (int)(size * 0.7);
                pictureBox.Width = (int)(size * 0.8);
                pictureBox.Location = new Point((int)(size * 0.1), (int)(size * 0.1));
                pictureBox.SizeMode = PictureBoxSizeMode.Zoom;
                pictureBox.Enabled = false;
                pictureBox.Image = Properties.Resources.mesa_02;

                Font font = new Font("Arial", 8);

                Button nuevoBoton = new Button();

                nuevoBoton.Height = size;
                nuevoBoton.Width = size;
                nuevoBoton.Text = x.Nombre;
                nuevoBoton.Name = x.Id.ToString();
                nuevoBoton.Font = font;
                nuevoBoton.TextImageRelation = TextImageRelation.ImageAboveText;
                nuevoBoton.TextAlign = ContentAlignment.BottomCenter;
                nuevoBoton.Click += new EventHandler(clicMesa);
                nuevoBoton.Location = new Point(size * columna + columna * espacio + espacio, size * fil
a + fila * espacio + espacio);

                nuevoBoton.Controls.Add(pictureBox);

                gpMesas.Controls.Add(nuevoBoton);

                columna++;
            }
        }

        private void clicMesa(object sender, EventArgs e)
        {
            Button producto = (Button)sender;
            MesaSeleccionada = Convert.ToInt32(producto.Name);
            Close();
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            Close();
        }

    }
}
```

Interfaz teclado táctil



Interfaz 22. Teclado táctil

Esta interfaz tiene dos funciones agregar un comentario al detalle de una orden y dar autorización de cancelar ordenes y detalles. El teclado estará oculto por defecto y se desplegará cuando se active el ChekBox Teclado. Cuando tenga la función de dar autorización el campo de texto tendrá los caracteres ocultos, si al ingresar la contraseña no coicide con la de ningún administrador mostrará el mensaje “Contraseña incorrecta”.

Windows Form

1. Name : FrmTeclado
    a. Text : Teclado
    b. StartPosition : CenterScreen
    c. KeyPreview : True
    d. Icon : A consideración

GroupBox

3. Name : gpTeclado
    c. Text : Teclado

TextBox

4. Name : txtText
    a. TabIndex : 0

Button

Para las letras de la A a la Z y para los números del 0 al 9

5. Name : btnLetra/Número
    a. Text : &Letra/Número
    b. TextImageRelation : Overlay
    c. Width : 40
    d. Height : 40

6. Name : btnEspacio
    a. Text : (Un espacio)
    b. TextImageRelation : Overlay
    c. Width : 178
    d. Height : 40
7. Name : btnBorrar
    a. Text : <
    b. TextImageRelation : Overlay
    c. Width : 86
    d. Height : 40
8. Name : btnAceptar
    a. Text : &Aceptar
    b. TextImageRelation : ImageBeforeText
    c. Width : 100
    d. Height : 40
    e. Image : A consideración
9. Name : btnCancelar
    a. Text : &Cancelar
    b. TextImageRelation : ImageBeforeText
    c. Width : 100
    d. Height : 40
    e. Image : A consideración

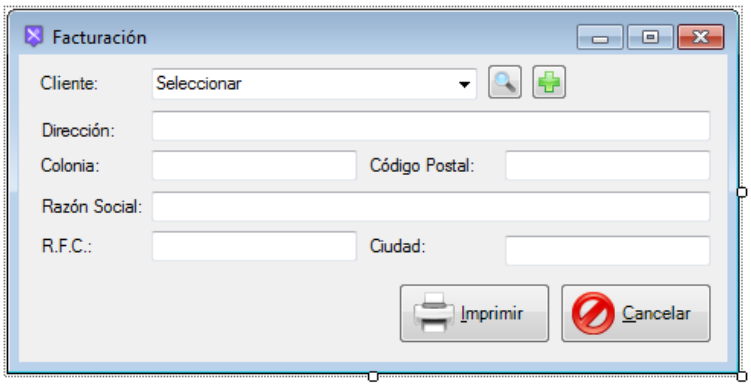
CheckBox

10. Name : chkTeclado
    a. Text :Teclado
    b. TabIndex : 1

```
using System;
using System.Linq;
using System.Windows.Forms;
using NHibernate;
using NHibernate.Linq;
using Datos;

namespace Restaurante.Procesos
{
    public partial class FrmTeclado : Form
    {
        public ISession Session;
        public bool password;
        public bool Permiso = false;
        public string Observacion;
        private const int ADMIN = 1;

        public FrmTeclado()
        {
            InitializeComponent();
        }

        private void FrmPermiso_Load(object sender, EventArgs e)
        {
            lblTexto.Text = "Observaciones:";
            if (password) {
                txtTexto.PasswordChar = '*';
                lblTexto.Text = "Contraseña:";
            }

            this.Height = 145;
            this.Width = 273;
            gpTeclado.Visible = false;
        }

        private void chkTeclado_CheckedChanged(object sender, EventArgs e)
        {
            if (chkTeclado.Checked)
            {
                this.Height = 319;
                this.Width = 748;
                gpTeclado.Visible = true;
            }
            else
            {
                this.Height = 145;
                this.Width = 273;
                gpTeclado.Visible = false;
            }
        }

        private void buton_Click(object sender, EventArgs e)
        {
            Button boton = (Button)sender;
            txtTexto.Text += boton.Text;
        }

        private void btnBorrar_Click(object sender, EventArgs e)
        {
            Array a = txtTexto.Text.ToCharArray();
            txtTexto.Text = "";
            for (int i = 0; i < (a.Length - 1); i++)
            {
                txtTexto.Text += a.GetValue(i);
            }
        }


        private void btnAceptar_Click(object sender, EventArgs e)
        {
            if (password)
            {
                var lista = from c in Session.Query<EmpleadoClass>()
```

```
                        where (c.Activo && c.TipoEmpleado.Id == ADMIN)
                        select c;

                foreach (var x in lista)
                {
                    if (x.Password == txtTexto.Text)
                    {
                        Permiso = true;
                        Close();
                        break;
                    }
                }

                if (!Permiso)
                {
                    MessageBox.Show("Contraseña incorrecta.");
                }
            }else {
                Observacion = txtTexto.Text;
                Close();
            }
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

Interfaz para facturación



Interfaz 23. Facturación

Esta interfaz permite asignar un cliente a la orden en cuestión para poder generar su factura. Al imprimir la factura la orden pasa al estado de Cobrando. Desde esta interfaz el cajero podrá agregar nuevos clientes y acceder a un listado para realizar una búsqueda o edición de los registros. Al seleccionar un cliente del listado sus datos se cargaran automáticamente. Si un dato es modificado al imprimir su facura los cambios serán guardados.

Windows Form

1. Name : FrmFacturacion
    a. Text : Facturación
    a. StartPosition : CenterScreen
    b. FormBorderStyle : FixedSingle
    c. MaximizeBox : False
    d. MinimizeBox : False
    e. KeyPreview : True
    f. AcceptButton : btnImprimir
    g. Icon : A consideración

TextBox

2. Name : txtDireccion
    a. TabIndex : 3
3. Name : txtColonia
    a. TabIndex : 4
4. Name : txtCodigoPostal
    a. TabIndex : 5
5. Name : txtRazonSocial
    a. TabIndex : 6
6. Name : txtRfc
    a. TabIndex: 7
7. Name : txtCiudad
    a. TabIndex : 8

ComboBox

8. Name : cboClientes
    a. TabIndex : 0
    b. Text : Seleccionar

Button

9. Name : btnBuscarCliente
    a. TextImageRelation : Overlay
    b. TabIndex : 1
    c. Width : 24
    d. Height : 24
    e. Image : A consideración
10. Name : btnAgregarCliente
    a. TextImageRelation : Overlay
    b. TabIndex : 2
    c. Width : 24
    d. Height : 24
    e. Image : A consideración
11. Name : btnImprimir

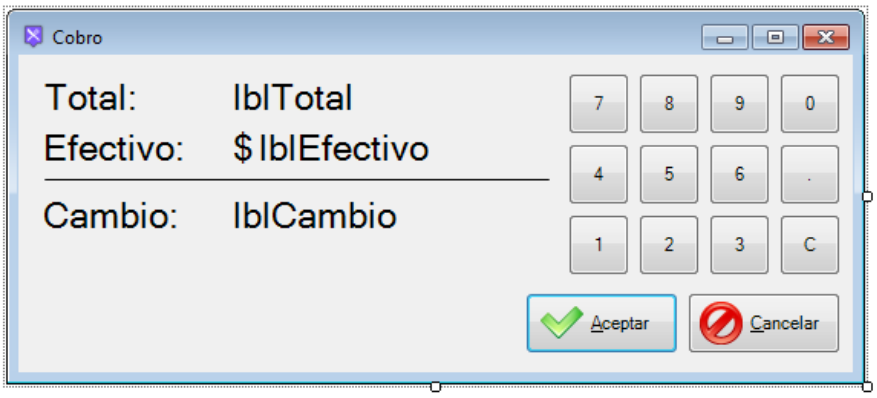
a. Text : &Imprimir
b. TextImageRelation : ImageBeforeText
c. TabIndex : 9
d. Width : 100
e. Height : 40
f. Image : A consideración

12. Name : btnCancelar
    a. Text : &Cancelar
    b. TextImageRelation : ImageBeforeText
    c. TabIndex : 10
    d. Width : 100
    e. Height : 40
    f. Image : A consideración

```
using System;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Catalogos;
using Restaurante.Reportes.Factura;

namespace Restaurante.Procesos
{
    public partial class FrmFacturacion : Form
    {
        public ISession Session;
        public OrdenClass Orden;
        private ClienteClass Cliente;

        public FrmFacturacion()
        {
            InitializeComponent();
        }

        private void FrmFacturacion_Load(object sender, EventArgs e)
        {
            CargarClientes();
        }

        private void CargarClientes()
        {
            var lista = from c in Session.Query<ClienteClass>()
                        where c.Activo
                        select c;

            cboClientes.DisplayMember = "Nombre";
            cboClientes.ValueMember = "Id";
            cboClientes.DataSource = lista.ToList();

            cboClientes.SelectedIndex = -1;
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void cboClientes_SelectedIndexChanged(object sender, EventArgs e)
        {
            if(cboClientes.SelectedValue == null)
                return;

            Cliente = Session.Load<ClienteClass>(cboClientes.SelectedValue);
            txtCiudad.Text = Cliente.Ciudad;
            txtCodigoPostal.Text = Cliente.CodigoPostal;
            txtColonia.Text = Cliente.Colonia;
```

```
            txtDireccion.Text = Cliente.Direccion;
            txtRazonSocial.Text = Cliente.RazonSocial;
            txtRfc.Text = Cliente.Rfc;
        }

        private void btnImprimir_Click(object sender, EventArgs e)
        {
            if (cboClientes.SelectedValue == null)
            {
                MessageBox.Show("Seleccione un cliente.");
                return;
            }

            Cliente.Ciudad = txtCiudad.Text;
            Cliente.CodigoPostal = txtCodigoPostal.Text;
            Cliente.Colonia = txtColonia.Text;
            Cliente.Direccion = txtDireccion.Text;
            Cliente.RazonSocial = txtRazonSocial.Text;
            Cliente.Rfc = txtRfc.Text;
            Session.Save(Cliente);
            Orden.Cliente = Cliente;
            Session.Save(Orden);
            Session.Flush();

            FrmReporteFactura frmReporteFactura = new FrmReporteFactura
                                                  {
                                                      IdOrden = Orden.Id
                                                  };
            frmReporteFactura.ShowDialog(this);

            Close();
        }

        private void btnBuscarCliente_Click(object sender, EventArgs e)
        {
            FrmClienteListado frmClienteListado = new FrmClienteListado
            {
                EsBusqueda = true,
                Session = Session
            };
            if (frmClienteListado.ShowDialog(this) != DialogResult.OK)
                return;

            CargarClientes();

            cboClientes.SelectedValue = frmClienteListado.ClienteSeleccionado;
        }

        private void btnAgregarCliente_Click(object sender, EventArgs e)
        {
            FrmClienteCaptura frmClienteCaptura = new FrmClienteCaptura
                                                  {
                                                      Session = Session
                                                  };
            frmClienteCaptura.ShowDialog(this);

            CargarClientes();
        }
    }
}
```

Interfaz para cobro de cuentas



Interfaz 24. Cobro de cuentas

Esta interfaz permite al cajero realizar el cobro de la cuenta. La interfaz muestra el total de la orden en cuestión y el cajero teclea el monto que recibe.

Si el monto es mayor a la cuenta, la interfaz se cierra regresa un valor positivo de cobrado, de la contrario muestra el mensaje de “No hay suficiente efectivo.”.

Mientras el monto de efectivo sea menor que del total los números del cambio serán de color rojo, cuando el monto de efectivo sea mayor los números serán de color verde.

La calculadora deberá funcionar con el teclado númerico del teclado.

Windows Form

1. Name : FrmCobro
    a. Text : Cobro
    b. StartPosition : CenterScreen
    c. KeyPreview : True
    d. AcceptButton : btnAceptar
    e. Icon : A consideración

Label

4. Name : lblTotal

   a. Text : lblTotal
5. Name : lblEfectivo
   a. Text : lblEfectivo
6. Name : lblCambio
   a. Text : lblCambio

Button

Para los números del 0 al 9

7. Name : btnNúmero
   e. Text : Número
   f. TextImageRelation : Overlay
   g. Width : 40
   h. Height : 40

8. Name : btnPunto
   a. Text : .
   b. TextImageRelation : Overlay
   c. Width : 40
   d. Height : 40
9. Name : btnC
   a. Text : C
   b. TextImageRelation : Overlay
   c. Width : 40
   d. Height : 40
10. Name : btnAceptar
   a. Text : &Aceptar
   b. TextImageRelation : ImageBeforeText
   c. TabIndex : 9
   d. Width : 100
   e. Height : 40
   f. Image : A consideración
11. Name : btnCancelar
   a. Text : &Cancelar
   b. TextImageRelation : ImageBeforeText
   c. TabIndex : 10
   d. Width : 100
   e. Height : 40
   f. Image : A consideración

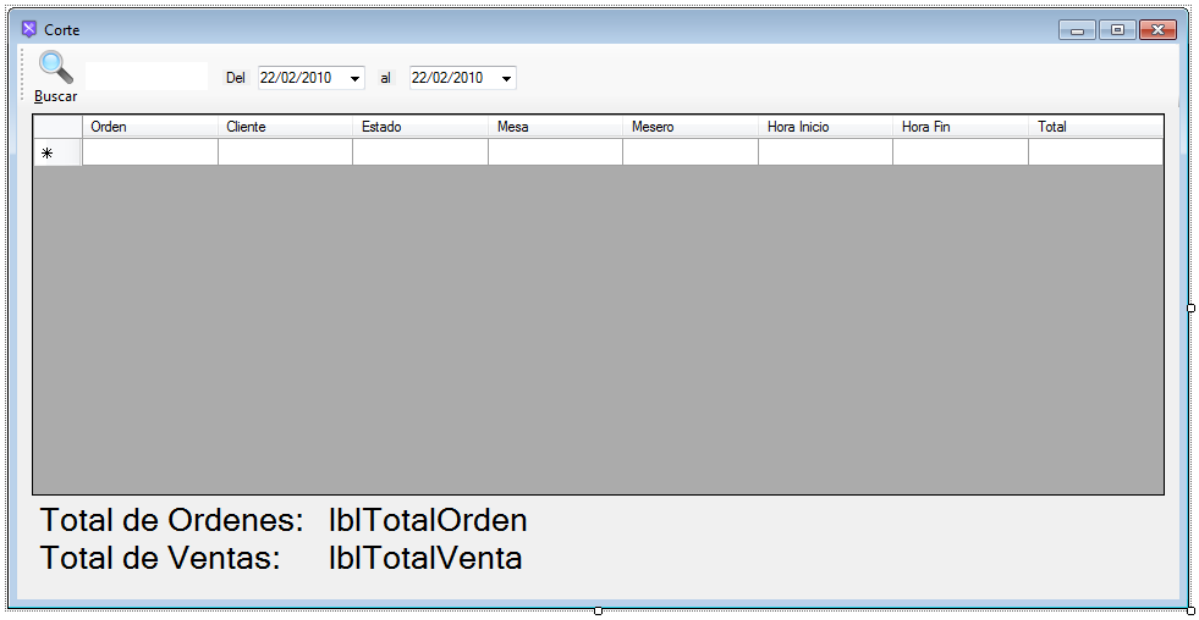
```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Restaurante.Procesos
{
    public partial class FrmCobro : Form
    {
        public Decimal Total;
```

```
private Decimal Efectivo;
public bool Cobrado;

public FrmCobro()
{
    InitializeComponent();
}

private void FrmCobro_Load(object sender, EventArgs e)
{
    Efectivo = 0;
    Cobrado = false;
    lblTotal.Text = Total.ToString("c2");
    lblEfectivo.Text = "0";
    CalcularCambio();
    lblCambio.Text = "";
    btnAceptar.Focus();
}

private void btn1_Click(object sender, EventArgs e)
{
    if (lblEfectivo.Text == "0")
        lblEfectivo.Text = "";

    lblEfectivo.Text += "1";
    CalcularCambio();
}

private void btn2_Click(object sender, EventArgs e)
{
    if (lblEfectivo.Text == "0")
        lblEfectivo.Text = "";

    lblEfectivo.Text += "2";
    CalcularCambio();
}

private void btn3_Click(object sender, EventArgs e)
{
    if (lblEfectivo.Text == "0")
        lblEfectivo.Text = "";

    lblEfectivo.Text += "3";
    CalcularCambio();
}

private void btn4_Click(object sender, EventArgs e)
{
    if (lblEfectivo.Text == "0")
        lblEfectivo.Text = "";

    lblEfectivo.Text += "4";
    CalcularCambio();
}

private void btn5_Click(object sender, EventArgs e)
{
    if (lblEfectivo.Text == "0")
        lblEfectivo.Text = "";

    lblEfectivo.Text += "5";
    CalcularCambio();
}

private void btn6_Click(object sender, EventArgs e)
{
    if (lblEfectivo.Text == "0")
        lblEfectivo.Text = "";

    lblEfectivo.Text += "6";
    CalcularCambio();
}

private void btn7_Click(object sender, EventArgs e)
{
    if (lblEfectivo.Text == "0")
        lblEfectivo.Text = "";

    lblEfectivo.Text += "7";
    CalcularCambio();
}
```

```
private void btn8_Click(object sender, EventArgs e)
{
    if (lblEfectivo.Text == "0")
        lblEfectivo.Text = "";

    lblEfectivo.Text += "8";
    CalcularCambio();
}

private void btn9_Click(object sender, EventArgs e)
{
    if (lblEfectivo.Text == "0")
        lblEfectivo.Text = "";

    lblEfectivo.Text += "9";
    CalcularCambio();
}

private void btn0_Click(object sender, EventArgs e)
{
    if (lblEfectivo.Text == "0")
        lblEfectivo.Text = "";

    lblEfectivo.Text += "0";
    CalcularCambio();
}

private void btnC_Click(object sender, EventArgs e)
{
    lblEfectivo.Text = "0";
    CalcularCambio();
}

private void btnPunto_Click(object sender, EventArgs e)
{
    if (TienePunto())
        return;

    lblEfectivo.Text += ".";
}

private bool TienePunto()
{
    if (lblEfectivo.Text.Contains("."))
        return true;
    else
        return false;
}

private void FrmCobro_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.NumPad0)
        btn0_Click(sender, e);

    if (e.KeyCode == Keys.NumPad1)
        btn1_Click(sender, e);

    if (e.KeyCode == Keys.NumPad2)
        btn2_Click(sender, e);

    if (e.KeyCode == Keys.NumPad3)
        btn3_Click(sender, e);

    if (e.KeyCode == Keys.NumPad4)
        btn4_Click(sender, e);

    if (e.KeyCode == Keys.NumPad5)
        btn5_Click(sender, e);

    if (e.KeyCode == Keys.NumPad6)
        btn6_Click(sender, e);

    if (e.KeyCode == Keys.NumPad7)
        btn7_Click(sender, e);

    if (e.KeyCode == Keys.NumPad8)
        btn8_Click(sender, e);

    if (e.KeyCode == Keys.NumPad9)
        btn9_Click(sender, e);
```

```
            if (e.KeyCode == Keys.Decimal)
                btnPunto_Click(sender, e);

            if (e.KeyCode == Keys.Escape)
                btnC_Click(sender, e);

            if (e.KeyCode == Keys.Return)
                btnAceptar_Click(sender, e);

            if(e.KeyCode == Keys.Escape)
                btnCancelar_Click(sender, e);
        }

        private void CalcularCambio()
        {
            Decimal Cambio;
            Efectivo = Convert.ToDecimal(lblEfectivo.Text);
            Cambio = Efectivo - Total;

            if (Cambio >= 0)
                lblCambio.ForeColor = Color.Green;
            else
                lblCambio.ForeColor = Color.Red;

            lblCambio.Text = Cambio.ToString("c2");
            btnAceptar.Focus();
        }

        private void btnAceptar_Click(object sender, EventArgs e)
        {
            if (Efectivo >= Total)
            {
                Cobrado = true;
                Close();
            }
            else
            {
                MessageBox.Show("No hay suficiente efectivo.");
            }
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            Cobrado = false;
            Close();
        }
    }
}
```

Interfaz para el corte diario



Interfaz 25. Corte de caja

Esta interfaz permite al administrador obtener un reporte de todas las ventas entre un periodo de fechas. Además podrá realizar un filtrado por un cliente, el estado de una orden, una mesa y un empleado.

Windows Form

1. Name : FrmCorte
    a. Text : Corte
    b. StartPosition : CenterScreen
    c. KeyPreview : True
    d. Icon : A consideración

Label

7. Name : lblTotalOrden
    b. Text : lblTotalOrden
8. Name : lblTotalVenta

b. Text : lblVenta

DataGridView

9. Name : grdOrdenes
    a. AutoSizeColumsMode : Fill
    b. Dock : Fill
    c. MultiSelect : False
    d. ReadOnly : True
    e. RowTemplante
    f. SelectionMode : FullRowSelect
    g. Columns : Las indicadas

ToolStrip

10. Name : tsMenu

ToolStripButton

11. Name : btnBuscar
    a. Text : &Buscar
    b. DisplayStyle : ImageAndText
    c. ImageScaling : None
    d. TextImageRelation : ImageAboveText
    e. Image : A consideración

ToolStripTextBox

12. Name : txtBuscar

DateTimePicker

13. Name : dtpInicial
    a. Format : Short
14. Name : dtpFinal
    a. Format : Short

```
using System;
using System.Linq;
using System.Windows.Forms;
using Datos;
using NHibernate;
using NHibernate.Linq;
using Restaurante.Funciones;

namespace Restaurante.Procesos.Corte
{
    public partial class FrmCorte : Form
    {
        private ISession Session;

        public FrmCorte()
        {
            InitializeComponent();
        }
```

```
        private void FrmCorte_Load(object sender, EventArgs e)
        {
            Session = Conexion.CreateSessionFactory().OpenSession();

            grdOrdenes.AutoGenerateColumns = false;
            dtpFinal.Value = DateTime.Today;
            dtpInicial.Value = DateTime.Today;

            CargarDatos();
        }

        private void CargarDatos()
        {
            var lista = from c in Session.Query<OrdenClass>()
                        where (c.Activo && c.Fecha >= dtpInicial.Value && c.Fecha <= dtpFinal.Value)
                        select new
                                    {
                                        Orden = c.Id,
                                        Cliente = c.Cliente.Nombre,
                                        c.StatusOrden,
                                        Estado = c.StatusOrden.Nombre,
                                        Mesa = c.Mesa.Nombre,
                                        Mesero = c.Empleado.Nombre,
                                        c.HoraInicio,
                                        c.HoraFin,
                                        Total = c.MontoConsumo
                                    };

            if (txtBuscar.Text != "")
            {
                if (Validacion.EsEntero((txtBuscar.Text)))
                    lista = lista.Where(c => c.Orden == Convert.ToInt64(txtBuscar.Text));
                else
                    lista = lista.Where(c => c.Cliente.Contains(txtBuscar.Text) || c.Estado.Contains(txtB
uscar.Text) || c.Mesa.Contains(txtBuscar.Text) || c.Mesero.Contains(txtBuscar.Text));
            }

            grdOrdenes.DataSource = lista.ToList();


            decimal TotalVentas = 0;
            foreach (var x in lista)
            {
                if (x.StatusOrden.Id == 3)
                    TotalVentas += x.Total;
            }

            lblTotalOrden.Text = lista.Count().ToString();
            lblTotalVenta.Text = TotalVentas.ToString("c2");

        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            CargarDatos();
        }

        private void txtBuscar_KeyDown(object sender, KeyEventArgs e)
        {
            if(e.KeyCode == Keys.Return)
                CargarDatos();
        }

        private void dtpInicial_ValueChanged(object sender, EventArgs e)
        {
            CargarDatos();
        }

        private void dtpFinal_ValueChanged(object sender, EventArgs e)
        {
            CargarDatos();
        }
    }
}
```

Anexo B

Clase Status Orden

C#
```
namespace Datos
{
    public class StatusOrdenClass
    {
        public virtual int Id { get; set; }
        public virtual String Nombre { get; set; }
        public virtual bool Activo { get; set; }

        public override string ToString()
        {
            return Nombre;
        }
    }
}
```

Mapeo Clase Status Orden

C#
```
using FluentNHibernate.Mapping;

namespace Datos
{
    public class StatusOrdenClassMap : ClassMap<StatusOrdenClass>
    {
        public StatusOrdenClassMap()
        {
            Table("Status_Orden");
            Id(x => x.Id).Column("Id_Status_Ordem").GeneratedBy.Identity();
            Map(x => x.Nombre).Column("Nombre").Length(30);
            Map(x => x.Activo).Column("Activo");
        }
    }
}
```

Clase Mesa

C#
```
namespace Datos
{
    public class MesaClass
    {
        public virtual int Id { get; set; }
        public virtual string Nombre { get; set; }
        public virtual bool Disponible { get; set; }
        public virtual bool Activo { get; set; }

        public override string ToString()
        {
            return Nombre;
        }
    }
}
```

Mapeo Mesa

```C#
using FluentNHibernate.Mapping;

namespace Datos
{
    public class MesaClassMap: ClassMap<MesaClass>
    {
        public MesaClassMap()
        {
            Table("Mesas");
            Id(x => x.Id).Column("Id_Mesa").GeneratedBy.Identity();
            Map(x => x.Nombre).Column("Nombre").Length(30);
            Map(x => x.Disponible).Column("Disponible");
            Map(x => x.Activo).Column("Activo");
        }
    }
}
```

Clase Empleado

```C#
using System;

namespace Datos
{
    public class EmpleadoClass
    {
        public virtual int Id { get; set; }
        public virtual string Nombre { get; set; }
        public virtual DateTime FechaNacimiento { get; set; }
        public virtual string Telefono { get; set; }
        public virtual string Direccion { get; set; }
        public virtual string Colonia { get; set; }
        public virtual string CodigoPostal { get; set; }
        public virtual string Email { get; set; }
        public virtual TipoEmpleadoClass TipoEmpleado { get; set; }
        public virtual byte[] Imagen { get; set; }
        public virtual string Usuario { get; set; }
        public virtual string Password { get; set; }
        public virtual bool Activo { get; set; }
    }
}
```

Mapeo Empleado

```C#
using FluentNHibernate.Mapping;

namespace Datos
{
    public class EmpleadoClassMap : ClassMap<EmpleadoClass>
    {
        public EmpleadoClassMap()
        {
            Table("Empleados");
            Id(x => x.Id).Column("Id_Empleado").GeneratedBy.Identity();
            Map(x => x.Nombre).Column("Nombre").Length(100);
            Map(x => x.FechaNacimiento).Column("Fecha_Nacimiento");
            Map(x => x.Telefono).Column("Telefono").Length(30);
            Map(x => x.Direccion).Column("Direccion");
            Map(x => x.Colonia).Column("Colonia").Length(30);
            Map(x => x.CodigoPostal).Column("Codigo_Postal").Length(15);
            Map(x => x.Email).Column("Email").Length(50);
            References(x => x.TipoEmpleado).Column("Id_Tipo_Empleado");
            Map(x => x.Imagen).Column("Imagen").Length(999999999);
            Map(x => x.Usuario).Column("Usuario").Length(20);
            Map(x => x.Password).Column("Password").Length(20);
            Map(x => x.Activo).Column("Activo");
        }
    }
}
```

Clase Tipo Empleado

C#
```
namespace Datos
{
    public class TipoEmpleadoClass
    {
        public virtual int Id { get; set; }
        public virtual string Nombre { get; set; }
        public virtual bool Activo { get; set; }
    }
}
```

Mapeo Tipo Empleado

C#
```
using FluentNHibernate.Mapping;

namespace Datos
{
    public class TipoEmpleadoClassMap : ClassMap<TipoEmpleadoClass>
    {
        public TipoEmpleadoClassMap()
        {
            Table("Tipos_Empleados");
            Id(x => x.Id).Column("Id_Tipo_Empleado").GeneratedBy.Identity();
            Map(x => x.Nombre).Column("Nombre");
            Map(x => x.Activo).Column("Activo");
        }
    }
}
```

Clase Categoría

C#
```
namespace Datos
{
    public class CategoriaClass
    {
        public virtual int Id { get; set; }
        public virtual string Nombre { get; set; }
        public virtual bool Publicado { get; set; }
        public virtual bool Activo { get; set; }
    }
}
```

Mapeo Categoría

C#
```
using FluentNHibernate.Mapping;

namespace Datos
{
    public class CategoriaClassMap : ClassMap<CategoriaClass>
    {
        public CategoriaClassMap()
        {
            Table("Categorias");
            Id(x => x.Id).Column("Id_Categoria").GeneratedBy.Identity();
            Map(x => x.Nombre).Column("Nombre").Length(50);
            Map(x => x.Publicado).Column("Publicado");
            Map(x => x.Activo).Column("Activo");
        }
    }
}
```

Clase Impresora

C#
```
namespace Datos
{
    public class ImpresoraClass
    {
        public virtual int Id { get; set; }
        public virtual string Nombre { get; set; }
        public virtual string Ruta { get; set; }
        public virtual bool Activo { get; set; }
    }
}
```

Mapeo Impresora

C#
```
using FluentNHibernate.Mapping;

namespace Datos
{
    public class ImpresoraClassMap : ClassMap<ImpresoraClass>
    {
        public ImpresoraClassMap()
        {
            Table("Impresoras");
            Id(x => x.Id).Column("Id_Impresora").GeneratedBy.Identity();
            Map(x => x.Nombre).Column("Nombre").Length(60);
            Map(x => x.Ruta).Column("Ruta");
            Map(x => x.Activo).Column("Activo");
        }
    }
}
```

Clase Articulo

C#
```
namespace Datos
{
    public class ArticuloClass
    {
        public virtual int Id { get; set; }
        public virtual string Nombre { get; set; }
        public virtual CategoriaClass Categoria { get; set; }
        public virtual ImpresoraClass Impresora { get; set; }
        public virtual decimal Precio { get; set; }
        public virtual byte[] Imagen { get; set; }
        public virtual bool Activo { get; set; }

        public override string ToString()
        {
            return Nombre;
        }
    }
}
```

Mapeo Articulo

C#
```
using FluentNHibernate.Mapping;

namespace Datos
{
    public class ArticuloClassMap : ClassMap<ArticuloClass>
    {
        public ArticuloClassMap()
        {
            Table("Articulos");
            Id(x => x.Id).Column("Id_Articulo").GeneratedBy.Identity();
            Map(x => x.Nombre).Column("Nombre").Length(50);
            References(x => x.Categoria).Column("Categoria");
            References(x => x.Impresora).Column("Impresora");
            Map(x => x.Precio).Column("Precio");
            Map(x => x.Imagen).Column("Imagen").Length(999999999);
            Map(x => x.Activo).Column("Activo");
        }
    }
}
```

Clase Orden Detalle

C#

```
namespace Datos
{
    public class OrdenDetalleClass
    {
        public virtual int Id { get; set; }
        public virtual OrdenClass Orden { get; set; }
        public virtual ArticuloClass Articulo { get; set; }
        public virtual int Cantidad { get; set; }
        public virtual decimal CostoUnitario { get; set; }
        public virtual decimal Importe { get; set; }
        public virtual bool Impreso { get; set; }
        public virtual string Observaciones { get; set; }
        public virtual bool Activo { get; set; }
    }
}
```

Mapeo Orden Detalle

C#
```
using FluentNHibernate.Mapping;

namespace Datos
{
    public class OrdenDetalleClassMap : ClassMap<OrdenDetalleClass>
    {
        public OrdenDetalleClassMap()
        {
            Table("Ordenes_Detalles");
            Id(x => x.Id).Column("Id_Orden_Detalle").GeneratedBy.Identity();
            References(x => x.Orden).Column("Id_Orden");
            References(x => x.Articulo).Column("Id_Articulo");
            Map(x => x.Cantidad).Column("Cantidad");
            Map(x => x.CostoUnitario).Column("Costo_Unitario");
            Map(x => x.Importe).Column("Importe");
            Map(x => x.Impreso).Column("Impreso");
            Map(x => x.Observaciones).Column("Observaciones");
            Map(x => x.Activo).Column("Activo");
        }
    }
}
```

Clase Orden

C#
```
using System;

namespace Datos
{
    public class OrdenClass
    {
        public virtual int Id { get; set;}
        public virtual MesaClass Mesa { get; set; }
        public virtual ClienteClass Cliente { get; set; }
        public virtual decimal MontoConsumo { get; set; }
        public virtual DateTime Fecha { get; set; }
        public virtual DateTime HoraInicio { get; set; }
        public virtual DateTime HoraFin { get; set; }
        public virtual EmpleadoClass Empleado { get; set; }
        public virtual StatusOrdenClass StatusOrden { get; set; }
        public virtual bool Activo { get; set; }
    }
}
```

Mapeo Orden

C#
```
using FluentNHibernate.Mapping;

namespace Datos
{
    public class OrdenClassMap : ClassMap<OrdenClass>
    {
        public OrdenClassMap()
        {
            Table("Ordenes");
            Id(x => x.Id).Column("Id_Orden").GeneratedBy.Identity();
            References(x => x.Mesa).Column("Id_Mesa");
            References(x => x.Cliente).Column("Id_Cliente");
            Map(x => x.MontoConsumo).Column("Monto_Consumo");
            Map(x => x.Fecha).Column("Fecha");
            Map(x => x.HoraInicio).Column("Hora_Inicio");
            Map(x => x.HoraFin).Column("Hora_Fin");
            References(x => x.Empleado).Column("Id_Empleado");
            References(x => x.StatusOrden).Column("Id_Status_Orden");
            Map(x => x.Activo).Column("Activo");
        }
    }
}
```

Anexo C

En este anexo se encuentran las clases que se necesitaron para realizar una tare especifica en el sistema.

CrearBaseDatos.cs

La función de esta clase abrir una conexión con la base de datos del servidor y generar la base de datos que se solicitó en la Interfaz 2.

```
using System;
using System.Windows.Forms;
using System.Data.SqlClient;
using System.Data;

namespace Restaurante.Funciones
{
    public class CrearBaseDatos
    {
        public static void Crear(string servidor, string seguridad, string baseDatos) {
            // La conexión a usar, indicando la base master
            SqlConnection cnn = new SqlConnection(
                "Server="+servidor+";"+seguridad);

            // La orden T-SQL para crear la tabla
            string s = "CREATE DATABASE "+baseDatos;
            SqlCommand cmd = new SqlCommand(s, cnn);

            try
            {
                // Abrimos la conexión y ejecutamos el comando
                cnn.Open();
                cmd.ExecuteNonQuery();
                //
                MessageBox.Show("Base de datos creada correctamente");
            }
            catch (Exception ex)
            {
               MessageBox.Show(ex.Message,
                    "Error al crear la base",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            finally
            {
                // Por si se produce un error,
                // comprobar si la conexión está abierta
                if (cnn.State == ConnectionState.Open)
                {
                    cnn.Close();
                }
            }

        }

    }
}
```

FunGenerarApp.cs

La función de esta clase sobre escribir el archivo de configurar del sistema que se solicitó en la Interfaz 2.

```
using System.Windows.Forms;
using System.Xml;

namespace Restaurante.Funciones
{
    public static class FunGenerarApp
    {
        public static void Generar(string Conexion)
        {
            XmlDocument configXml = new XmlDocument();
            string ficConfig;

            ficConfig = Application.ExecutablePath + ".config";
            //ficConfig = "../../app.config";
            //
            // comprobar si existe el fichero de configuración
            if (System.IO.File.Exists(ficConfig) == true)
            {
                System.Text.StringBuilder sXml = new System.Text.StringBuilder();
                //

                // crear la cadena a asignar al objeto XmlDocument
                sXml.Append("<configuration>");
                sXml.Append("<connectionStrings>");
                sXml.Append(Conexion);
                sXml.Append("</connectionStrings>");
                sXml.Append("<startup>");
                sXml.Append("<supportedRuntime version=\"v4.0\" sku=\".NETFramework,Version=v4.0\" />");
                sXml.Append("</startup>");
                sXml.Append("</configuration>");
                // asignamos la cadena al objeto
                configXml.LoadXml(sXml.ToString());
                //
                // Guardamos el contenido de configXml y creamos el fichero
                configXml.Save(ficConfig);
            }
            else
            {
                // solo es necesario leerlo si no lo hemos creado
                configXml.Load(ficConfig);
            }
        }
    }
}
```

FunServidor.cs

La función de esta clase es devolver únicamente la fecha actual del servidor y no de la maquina local, así como devolver la fecha y hora del mismo.

```
using System;
using NHibernate;

namespace Restaurante.Funciones
{
    public class FunServidor
    {
        public static DateTime FechaServidor(ISession session)
        {
            DateTime fecha = (DateTime)session.CreateSQLQuery("Select GetDate()").UniqueResult();
            DateTime fechaNueva = new DateTime(fecha.Year, fecha.Month, fecha.Day);
            return fechaNueva;
        }

        public static DateTime HoraServidor(ISession session)
        {
            DateTime fecha = (DateTime)session.CreateSQLQuery("Select GetDate()").UniqueResult();
            return fecha;
        }
    }
}
```

InicializarDatos.cs

La función de esta clase es crear datos por defecto cuando no haya ninguno. Contiene dos métodos el primero que genera los estados de la orden, estos datos en ningún lugar del sistema podrán ser modificados. El segundo método genera los tipos de empleados por defecto que son: Administrador, Cajero y Mesero, además de crear un Empleado de tipo Administrador con su usuario y contraseña “ADMIN”.

```
using System;
using System.Linq;
using Datos;
using NHibernate;
using NHibernate.Linq;

namespace Restaurante.Funciones
{
    public class IniciarDatos
    {
        public static void InicializarEstadoOrden(ISession session)
        {
            var datos = session.Query<StatusOrdenClass>().Count();

            if (datos > 0)
                return;

            StatusOrdenClass estadoOrden;

            estadoOrden = new StatusOrdenClass
            {
                Nombre = "EN PROCESO",
                Activo = true
            };
            session.Save(estadoOrden);

            estadoOrden = new StatusOrdenClass
            {
                Nombre = "COBRANDO",
                Activo = true
            };
            session.Save(estadoOrden);

            estadoOrden = new StatusOrdenClass
            {
                Nombre = "COMPLETADO",
                Activo = true
            };
            session.Save(estadoOrden);

            estadoOrden = new StatusOrdenClass
            {
                Nombre = "CANCELADO",
                Activo = true
            };
            session.Save(estadoOrden);

            session.Flush();
        }

        public static void InicializarUsuarios(ISession session)
        {
            var datos = session.Query<EmpleadoClass>().Count();

            if (datos > 0)
                return;

            TipoEmpleadoClass tipoEmpleado = new TipoEmpleadoClass
            {
                Nombre = "ADMINISTRADOR",
                Activo = true
            };

            session.Save(tipoEmpleado);

            EmpleadoClass empleado = new EmpleadoClass
                                     {
```

```
                        Nombre = "ADMINISTRADOR",
                        TipoEmpleado = tipoEmpleado,
                        Usuario = "ADMIN",
                        Password = "ADMIN",
                        FechaNacimiento = DateTime.Today,
                        Imagen = UtilidadesImagenes.ImageToByteArray(Properties.Resources.image_default),
                        Activo = true
                                    };

            session.Save(empleado);

            tipoEmpleado = new TipoEmpleadoClass
            {
                Nombre = "MESERO(A)",
                Activo = true
            };

            session.Save(tipoEmpleado);

            tipoEmpleado = new TipoEmpleadoClass
            {
                Nombre = "CAJERO(A)",
                Activo = true
            };

            session.Save(tipoEmpleado);

            session.Flush();
        }
    }
}
```

UtilidadesImagenes.cs

Esta clase permite convertir las imágenes en byte[] y viceversa para poderlos guardar en la base de datos.

```
using System;
using System.Drawing;
using System.IO;

namespace Restaurante.Funciones
{
    public class UtilidadesImagenes
    {

        // Obtiene última palabra después de carácter buscado.
        public static string CortarCadena(string cadena, char cortar)
        {
            string n = "";
            Array a = cadena.Split(cortar);
            for (int i = 0; i < a.Length; i++)
            {
                n = (string)a.GetValue(i);
            }
            return n;
        }

       // Convierte una imagen en byte.
        public static byte[] ImageToByteArray(System.Drawing.Image imagen)
        {
            MemoryStream ms = new MemoryStream();
            imagen.Save(ms, System.Drawing.Imaging.ImageFormat.Jpeg);
            return ms.ToArray();
        }

       // Convierte los byte en Imagen
        public static Image ByteArrayToImage(byte[] byteArrayIn)
        {
            MemoryStream ms = new MemoryStream(byteArrayIn);
            Image returnImage = Image.FromStream(ms);
            return returnImage;
        }

       // Convierte una imagen desde una ruta física en byte.
        public static byte[] ObtenerByteArrayDeArchivo(string ruta_archivo)
        {
            FileStream fs = new FileStream(ruta_archivo, FileMode.Open, FileAccess.Read);
            byte[] ImageData = new byte[fs.Length];
            fs.Read(ImageData, 0, System.Convert.ToInt32(fs.Length));
            fs.Close();
            return ImageData;
        }

       // Obtiene una imagen desde una ruta física en la computadora.
        public static Image RetornarThumbnail(string ruta_archivo)
        {
            Image imageThumb = null;
            try
            {
                imageThumb = Image.FromFile(ruta_archivo).GetThumbnailImage(100, 100, null, new IntPtr())
;
            }
            catch
            {
            }
            return imageThumb;
        }

     }
}
```

Validacion.cs

Esta clase permite validar si una cadena texto es un numero entero, es un numero decimal y si contiene una serie de palabras.

```
using System;
using System.Windows.Forms;

namespace Restaurante.Funciones
{
    public class Validacion
    {
        public static bool EsEntero (string Numero)
        {
            try
            {
                Convert.ToInt64(Numero);
                return true;
            }
            catch (Exception)
            {
                return false;
            }
        }

        public static bool EsDecimal(string Numero)
        {
            try
            {
               Convert.ToDecimal(Numero);
                return true;
            }
            catch (Exception)
            {
                return false;
            }
        }

        public static bool ValidarArchivoImagen(string Archivo)
        {

            if (Archivo.Contains("jpg"))
                return true;

            if (Archivo.Contains("jpng"))
                return true;

            if (Archivo.Contains("png"))
                return true;

            MessageBox.Show("El archivo debe ser una imagen (\"PNG,JPG,JPNG\").");

            return false;
        }
    }
}
```