PONTIFICIA UNIVERSIDAD CATOLICA DEL ECUADOR

FACULTAD DE INGENIERIA

ESCUELA DE SISTEMAS

DISERTACION DE GRADO PREVIA LA OBTENCION DEL TITULO DE INGENIERO EN SISTEMAS Y COMPUTACION

TEMA:

DESARROLLO E IMPLEMENTACION DE UN SISTEMA DE VERIFICACIÓN DE INFORMACIÓN DE CLIENTES PARA EMPRESAS DE COBRANZA

AUTOR:

JULIO CÉSAR NÚÑEZ ZABALA

DIRECTOR:

ING. OSWALDO ESPINOSA VITERI

QUITO – 2010

DEDICATORIA

El presente trabajo está dedicado a todas aquellas personas que de una u otra forma, estuvieron a mi lado prestándome siempre su valiosa ayuda, pero en especial se lo dedico a mi abuelito Tarquino, su ilusión siempre fue ver que termine la carrera y, aunque ya no está con nosotros, se que desde el cielo podrá ver que terminé mis estudios.

AGRADECIMIENTOS

En primer lugar quiero agradecer a Dios por haberme ayudado en todas mis obras ya que sin su ayuda y bendiciones nada habría sido posible hacer.

Agradezco también a mis padres por su apoyo incondicional a lo largo de toda mi vida y haberme guiado por un camino de bien, a mis abuelitos, a mis hermanos, a mis primos, y a mis tios y tías por haber estado siempre a mi lado dándome ánimos para seguir adelante.

A mi profesores, ya que sin sus valiosas enseñanzas no habría podido llegar a este punto y, al final pero no menos importantes, a mis amigas y amigos ya que, al igual que mi familia, también estuvieron a mi lado dispuestos siempre a brindarme su ayuda.

Tabla de contenido

DESARROLLO E IMPLEMENTACION DE UN SISTEMA DE VERIFICACIÓN DE INFORMACIÓN DE CLIENTES PARA EMPRESAS DE COBRANZA

RESUMEN

El presente trabajo muestra el desarrollo de un sistema de verificación de información para empresas de cobranza, para el desarrollo del mismo, se utilizaron varias metodologías como la programación orientada a objetos, ciclo de vida en cascada, etc. La idea de desarrollar este sistema surge de la necesidad de las empresas de cobranza de mantener la información demográfica de sus clientes actualizada. Este trabajo está dividido en cinco capítulos, y una parte introductoria en la que se encuentran los objetivos que se esperan cumplir al final del trabajo así como la justificación y el alcance del mismo, en los cuatro primeros capítulos se desarrolla el sistema de acuerdo a las fases del ciclo de vida en cascada, el quinto capítulo está destinado para las conclusiones y recomendaciones que se obtendrán una vez concluido el desarrollo del sistema.

Las fases del ciclo de vida en cascada generan ciertos documentos o productos, dichos documentos se encuentran en los anexos.

A continuación la parte introductoria de l trabajo:

INTRODUCCIÓN

Justificación:

En la actualidad, la mayoría de las empresas han visto la necesidad de automatizar sus procesos, lo que ha provocado cambios en la forma de trabajo y en la manera en que las empresas prestan sus servicios sin importar que estos tengan o no relación con temas de informática. Cuando una empresa automatiza un proceso, lo que está logrando es mejorar la productividad de la misma y así se obtiene una mayor ventaja frente a otras empresas.

Las tecnologías de la información y la automatización de los diferentes procesos que permitan la comunicación tecnológica entre dos empresas para la verificación de información de los clientes de instituciones financieras, son muy importantes ya que sin esto, se tiene información desactualizada sobre los mismos o en muchos casos, la información se encuentra almacenada en lugares diferentes, lo que produce que sea difícil acceder a ella rápidamente lo que puede causar grandes problemas como la imposibilidad de realizar una gestión de cobranza sobre clientes con grandes deudas, generando así grandes pérdidas.

Otro de los problemas que se puede presentar al no tener automatizado un proceso de verificación es que la carga de trabajo para las personas que realizan las visitas para verificar la información de los clientes sea desigual, lo que produce una baja

productividad por parte de los visitadores ya que, por tener demasiadas visitas, no podrán cumplir con su cuota diaria de visitas ni tampoco se puede llevar un control de cómo están trabajando los visitadores.

Planteamiento del Tema:

Con el fin de mejorar el rendimiento del proceso de verificación de información, se desarrollará un sistema que almacene en un solo lugar toda la información de las visitas realizadas, que genere reportes de cuántas verificaciones se realizaron por los visitadores para tener un mejor control del desempeño de su trabajo, que genere listas de trabajo para que se repartan las visitas de una manera equitativa a los visitadores, y que permita la administración de visitadores, clientes, ciudades y cantones, esto último, con el fin de tener una zonificación del país y que se puedas realizar la gestión de verificación de una forma más ordenada.

Objetivo General:

Desarrollar, e implementar un Sistema de Verificación de Información demográfica y generación de listas de trabajo para facilitar el manejo de información de clientes de empresas de cobranza.

Objetivos específicos:

- Levantar información de los requerimientos de empresas de cobranza para el desarrollo del sistema.
- Definir estándares de desarrollo y calidad para el sistema
- Documentar el proceso de desarrollo del sistema de verificación.
- Implementar un software de calidad que cumpla con estándares definidos y que satisfaga las necesidades del cliente.
- Desarrollar el sistema de Verificación de información aplicando la metodología de Programación Orientada a Objetos.
- Desarrollar un plan de pruebas del sistema tomando en cuenta las situaciones críticas que se pueden presentar en un ambiente real.
- Utilizar el ciclo de vida en Cascada para el desarrollo del producto de software.
- Generar los documentos necesarios para facilitar el uso del sistema, así como un manual técnico del sistema a desarrollarse.
- Finalizar el sistema dentro de los plazos definidos

Alcance:

El trabajo culminará una vez que se hayan realizado las pruebas, en ambiente de desarrollo del Sistema de Verificación de Información y la generación de listas de trabajo y con la generación de toda la documentación necesaria, tanto del desarrollo del Sistema como de los respectivos manuales técnico y el usuario

Metodología:

Para el desarrollo del Módulo, se seleccionaron varias metodologías tanto en la programación como en el ciclo de vida del mismo, para la parte de programación, se seleccionó la Programación Orientada a Objetos, y para el ciclo de vida del proyecto, se seleccionó el ciclo de vida en Cascada por ser el más utilizado para el desarrollo de sistemas.

Programación Orientada a Objetos

Intenta simular el mundo real a través del significado de objetos que contiene características y funciones. Los lenguajes orientados a objetos se clasifican como lenguajes de quinta generación. La programación orientada a objetos se basa en la idea de un objeto, que es una combinación de variables locales y procedimientos llamados métodos que juntos conforman una entidad de programación.

En la programación orientada a objetos, los sistemas se modelan generando clases, que son un conjunto de datos y funcionalidades. Las clases son definiciones, a partir de las que se crean objetos. Los objetos son ejemplares de una clase determinada y como tal, disponen de los datos y funcionalidades definidos en la clase. La programación orientada a objetos se basa en la idea de que un programa grande siempre será más complicado que la suma de varios programas pequeños, con lo que se considera ventajoso dividir un gran sistema en diversos módulos.

La programación orientada a objetos permite concebir los programas de una manera bastante intuitiva y cercana a la realidad, definiéndose modelos de capas para el trabajo e implementación del sistema. La tendencia es que un mayor número de lenguajes de programación adopten la programación orientada a objetos como paradigma para modelar los sistemas.

Ciclo de Vida

Para el desarrollo del sistema se ha seleccionado el ciclo de vida en Cascada ya que este ciclo mantiene una retroalimentación en cada una de sus fases, lo interesante de este modelo es que al final de cada una de sus fases se reúne la documentación necesaria para garantizar que cumple las especificaciones y los requisitos antes de pasar a la fase siguiente. La estructura del modelo en cascada es la siguiente:

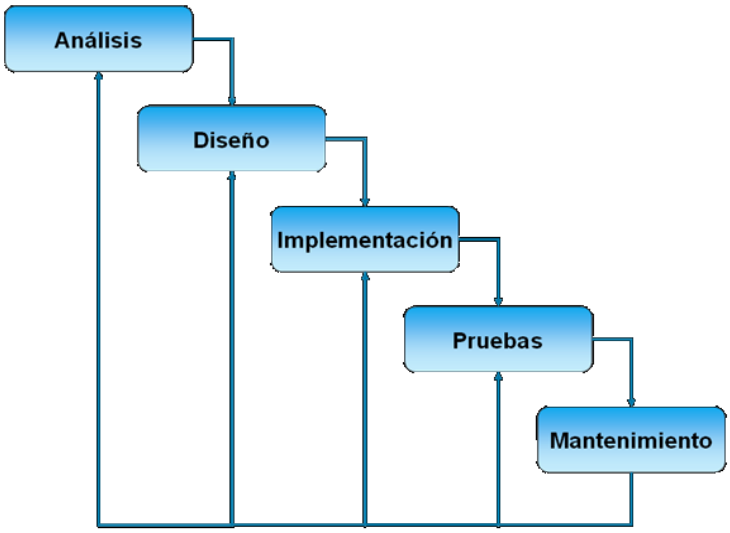


Figura 1: Estructura del ciclo de vida en cascada

Con esta estructura, si se encuentra cualquier error en la etapa de pruebas, se puede detectarlo y saber exactamente en cual de las etapas anteriores se encuentra. Este ciclo de vida de software fue propuesto por Royce en 1970, según un estimado, el 90% de los sistemas se desarrollan de esta manera. Las fases del modelo en cascada son las siguientes:

- **Análisis:** Es la primera fase del modelo en cascada, aquí se analizan las necesidades del cliente y se determinan los objetivos que el producto de software debe cumplir. Es muy importante definir bien los requerimientos en esta fase ya que de estos dependerá el desarrollo del producto de software. Esta fase produce un documento llamado SRD (Documento de Requerimientos de Software)
- **Diseño:** En base al SRD se diseña el sistema, esta fase produce un documento llamado SDD (Documento de Diseño del Software) que contiene la descripción de la estructura relacional global del sistema y la especificación de lo que debe

hacer cada una de sus partes, así como la manera en que se combinan unas con otras, en esta fase se realizan dos diseños:

  - ***Diseño general:*** se toman en cuenta los requisitos generales de la arquitectura de la aplicación
  - ***Diseño a detalle:*** se realiza una definición precisa de cada subconjunto de la aplicación

- **Implementación:** En base al SDD se realizan los diferentes módulos del sistema implementando el código fuente. Dependiendo del lenguaje de programación y su versión se crean las bibliotecas y componentes reutilizables dentro del mismo proyecto para hacer que la programación sea un proceso mucho más rápido. En esta fase se realizan pruebas unitarias de los módulos
- **Pruebas:** Los elementos, ya programados, se ensamblan para componer el sistema y se comprueba que funcionan correctamente antes de ser puesto en producción. El resultado de las pruebas es el producto final listo para entregar.
- **Mantenimiento:** Una vez que el software ha sido puesto en ambiente de producción, se debe realizar un mantenimiento continuo del producto por medio de actualizaciones

Ventajas de utilizar este ciclo de vida:

- Se tiene todo bien organizado y no se mezclan las fases.
- Es perfecto para proyectos que son rígidos, y además donde se especifiquen muy bien los requerimientos y se conozca muy bien la herramienta a utilizar.
- La calidad del producto de software final es alta
- Al llevar todas las fases documentadas, es muy fácil comprender como está estructurado un producto de software sólo con revisar dicha documentación

Desventajas del modelo:

- Es necesario el tener todos los requisitos al principio. Lo normal es que el cliente no tenga perfectamente definidas las especificaciones del sistema, o puede ser que surjan necesidades imprevistas.
- Si se han cometido errores en una fase es difícil volver atrás.
- No se tienen indicadores fiables del progreso del trabajo.

Señalamiento:

Ya que el para el desarrollo del sistema de verificación se utilizó el ciclo de vida en cascada, se tiene diferentes etapas de desarrollo las cuales fueron divididas en cuatro capítulos dentro de este trabajo, hay un quinto capítulo dedicado a las conclusiones obtenidas después de haber realizado el presente trabajo. A continuación, una breve descripción de cada uno de los capítulos:

- **Capítulo 1:** El primer capítulo tiene dos partes, la primera, en donde se realiza un pequeño análisis de la situación actual de las empresas de cobranza en nuestro país, es decir, como realizan sus gestiones de cobranza, la segunda parte, es un análisis de los requerimientos que tienen las empresas de cobranza para la realización del sistema, básicamente son los requerimientos del usuario, con estas información se pueden realizar los diagramas UML que servirán después para el modelado del sistema.
- **Capítulo 2:** Una vez que se cuenta con los diagramas UML, se procede a realizar el diseño de datos del sistema, esto es, el modelo físico y el modelo conceptual de la base de datos del sistema, estos temas son tratados en este segundo capítulo.
- **Capítulo 3:** En este capítulo se procede a explicar como se desarrolló el sistema una vez obtenida la información de los dos capítulos anteriores, básicamente se detalla como está estructurado el sistema y las diferentes funcionalidades que este ofrece, a manera de resumen, las funcionalidades que tiene el sistema son Administración de: clientes, cantones, ciudades, supervisores, verificadores perfiles de usuario, permite la generación de fichas de verificación así como de listas de trabajo y por último, la generación de varios reportes para controlar la gestión de verificación de información.
- **Capítulo 4:** El cuarto capítulo está dedicado a la realización de pruebas del sistema tanto de integración de los diferentes módulos como una prueba del sistema en ambiente de desarrollo.
- **Capítulo 5:** El último capítulo del presente trabajo contiene las conclusiones y recomendaciones que se obtendrán una vez que se haya desarrollado el sistema de verificación.

A más de los cinco capítulos anteriores, se tiene una sección de bibliografía en donde están todas las fuentes que fueron consultadas para le realización del marco teórico y una sección de anexos en donde se colocarán diferentes documentos que servirán como referencia parea una mejor comprensión de cómo fue desarrollado el sistema de verificación de información.

MARCO TEÓRICO

Para el desarrollo del Sistema de Verificación se utilizará varias herramientas de software, como Sistema Gestor de Base de Datos se escogió Microsoft SQL Server 2005, la herramienta seleccionada para realizar el modelamiento de datos fue Power Designer versión 12.0, la tecnología seleccionada para desarrollar el Sistema fue ASP.Net. A continuación una breve descripción de cada una de las herramientas de software.

ASP:

Sus siglas en ingles son **Active Server Pages**. Es una tecnología desarrollada por Microsoft del tipo lado del servidor, ha sido comercializada como un anexo para Internet Information Server (IIS). ASP sirve para páginas web generadas dinámicamente. Este modelo tecnológico permite utilizar diferentes componentes que ya están desarrollados como algunos controles ActiveX y componentes del lado del servidor como los CDONTS los cuales permiten la interacción de los scripts con el servidor SMTP (Simple Mail Transfer Protocol) que integra IIS.

Otra de las características de ASP es que facilita la programación de sitios web ya que tiene objetos de sesión basados en cookies los cuales mantienen las variables mientras se va pasando de página a página.

Desde 2002, ASP está siendo reemplazado por ASP.Net, la diferencia básica entre ASP y ASP.Net es que el segundo utiliza lenguajes compilados como Microsoft Visual Basic o C#, llamados MSIL (Microsoft Intermediate Languaje) en vez de lenguajes interpretados como VBScript o JScript. Los lenguajes MSIL se compilan con posterioridad al código nativo.

Las versiones de ASP anteriores a ASP.Net, son llamadas ASP *Clásico,* la última versión de ASP *Clásico* fue la versión 3.0, la característica de esta última versión es que tiene varios objetos que facilitan la creación de páginas dinámicas como por ejemplo los objetos Request, Response, Server, Session, entre otros.

ASP.Net:

Es a la tecnología sucesora de ASP, fue desarrollada por Microsoft, es un Framework para desarrollar aplicaciones web. ASP**.**Net podría ser considerado como competencia a la plataforma Java de Sun Microsystems y a los diversos Frameworks de desarrollo web basados en PHP.

ASP.Net está construido sobre CLR (Common Languaje Runtime) lo que permite a los programadores escribir código ASP.Net a partir de cualquier lenguaje que admita el Framework como Visual Basic o C#. La tecnología ASP.Net ofrece un ambiente orientado a objetos. A las páginas hechas en ASP.Net se la conoce como formularios web y tienen extensión aspx, otra de las características de ASP.Net es que maneja el modelo Code-Behind en donde lo programadores pueden crear sus propios métodos para programar las páginas, los archivos con el código fuente tiene extensiones aspx.cs o aspx.vb que corresponden a los dos lenguajes más utilizados en esta tecnología: Visual Basic y C# . La versión 2.0 de ASP.Net incluye el concepto de Página Maestra (Master Page) lo que permite el desarrollo de páginas basándose en plantillas web facilitando así la creación de las páginas web.

Microsoft SQL Server:

Es un sistema gestor de base de datos (SGBD o DBMS en inglés) capaz de poner a disposición de muchos usuarios cantidades enormes de datos de manera simultánea. Fue desarrollado por Microsoft como una alternativa a otros fuertes sistemas gestores de bases de datos como Oracle, MySQL, Sybase, entre otros.

Entre sus principales características tenemos:

- Soporte de transacciones.
- Escalabilidad, estabilidad y seguridad.
- Soporta procedimientos almacenados.
- Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y las terminales o clientes de la red sólo acceden a la información.
- Además permite administrar información de otros servidores de datos.

SQL Server soporta diferentes tipos de datos incluyendo tipos de datos primarios como Enteros, Flotantes, tipos de datos para cadenas de caracteres como Char, Varchar, entre otros, soporta también la creación de Procedimientos Almacenados (Stored Procedures), restricciones, vistas, etc. Contenidos en una base de datos. Una base de datos en SQL Server puede contener un máximo de 2^{31} objetos. Los datos dentro de una base de datos son almacenados en archivos de datos primarios con extensión .mdf, los archivos de datos secundarios, que tienen una extensión .ndf se los usa para almacenar los metadatos.

SQL Server también tiene archivos de registro (Log Files) los cuales almacenan la información de los usuarios que han ingresado a una base de datos y las transacciones que ha hecho, esto, con la idea de tener un control sobre como se maneja la información o si se realizaron cambios en las estructuras de tablas o procedimientos almacenados.

Para el desarrollo de aplicaciones más complejas (tres o más capas), *Microsoft SQL Server* incluye interfaces de acceso para varias plataformas de desarrollo, entre ellas .NET, pero el servidor sólo está disponible únicamente para Sistemas Operativos Windows.

Power Designer:

Es una herramienta de modelamiento que permite visualizar, manipular y analizar metadatos de una manera más fácil, esto permite una arquitectura de información empresarial más efectiva.

Al tener una mejor visualización de la arquitectura de información, se pueden implementar más fácil los productos de software que se adapten a las necesidades de las empresas.

Fue diseñado por Sybase, funciona en Windows como una aplicación nativa y sobre Eclipse a través de un plugin. Entre las varias ventajas que tiene Power Designer podemos enumerar las siguientes:

a) Generación automática de código para Java, C#, .NET, JSF, entre otros
b) Modelamiento para Data Warehouse
c) Modelamiento por diagramas UML
d) Modelamiento de datos para SGBD's Relacionales

Power Designer se encuentra en la versión 15.0, liberada en Octubre de 2008, las novedades de esta versión son que incluye a todos los tipos de diagramas UML para modelamiento.

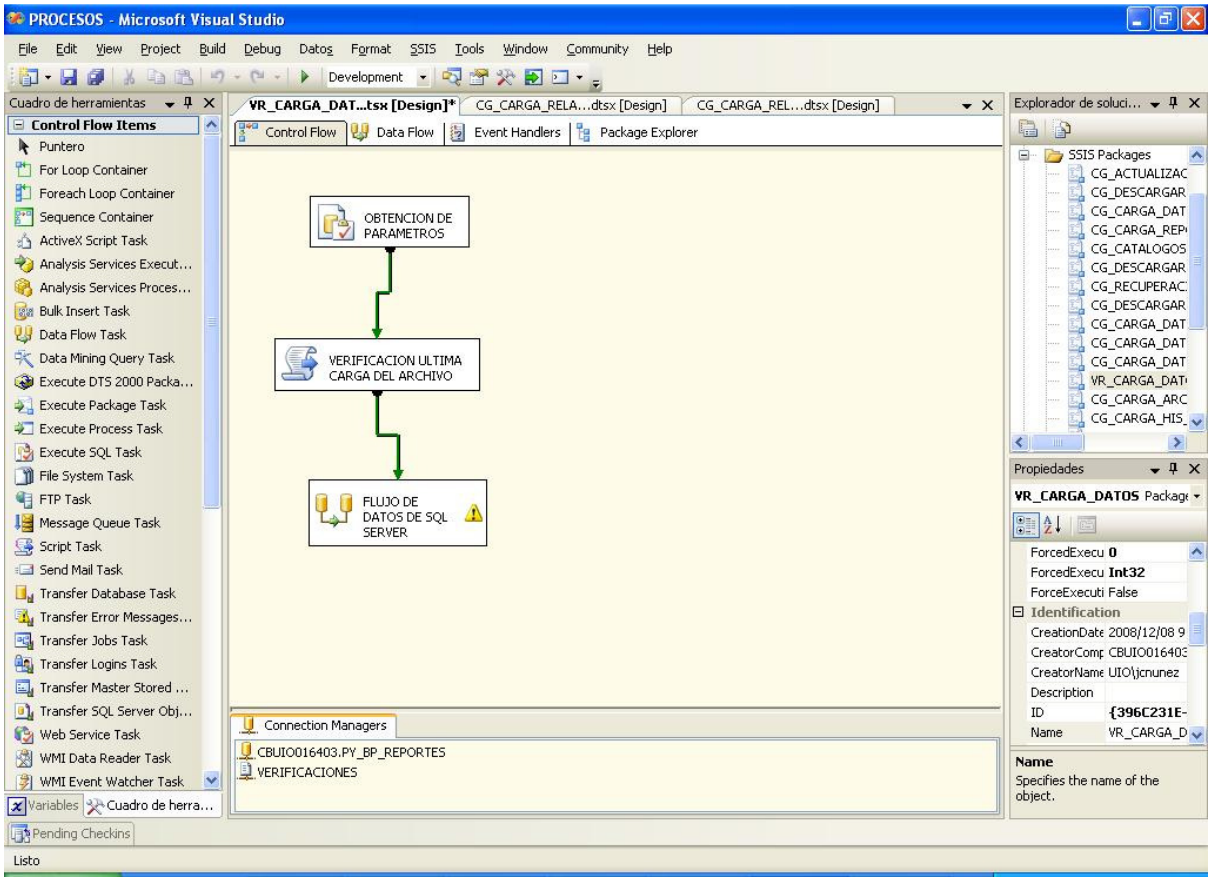
Una vez que se ha dado una breve descripción de las herramientas de software se procederá a hablar sobre el tipo de arquitectura y los diferentes diagramas UML que se utilizarán para el desarrollo del Sistema de Verificación.

DTS:

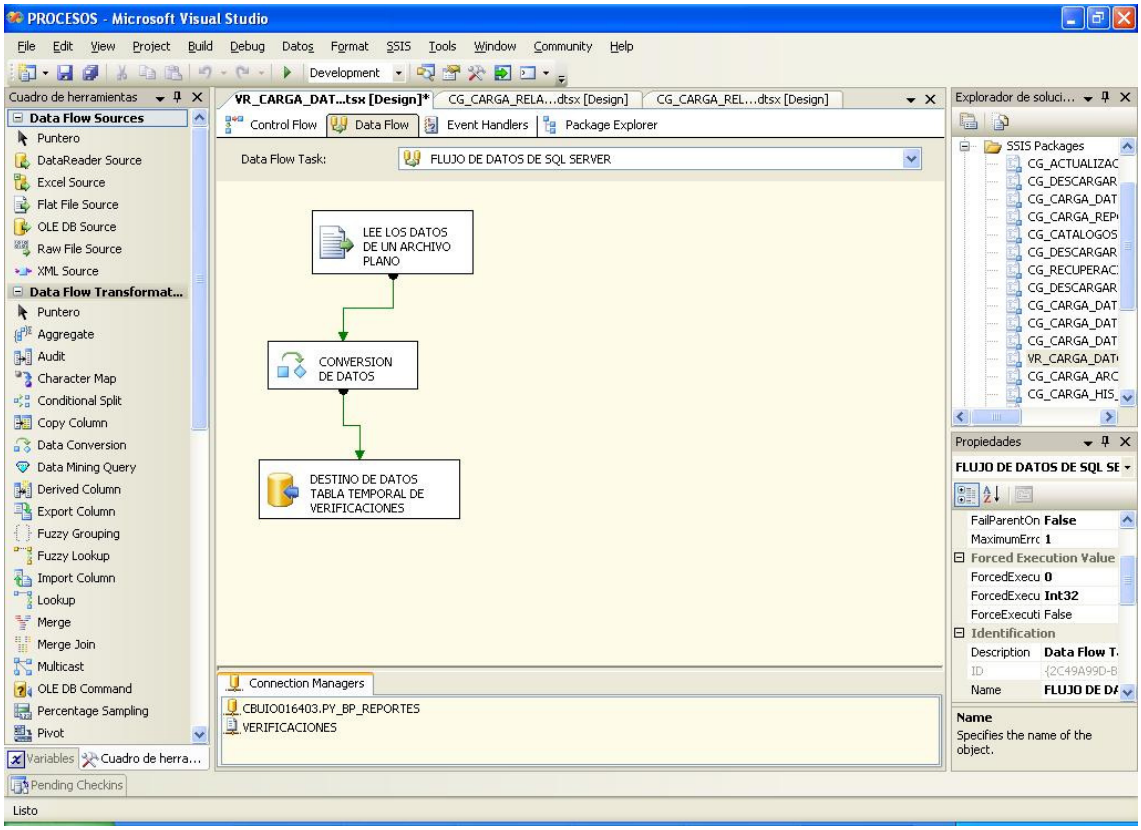
DTS significa Servicios de Transformación de Datos (Data Transformation Services), estas herramientas permiten mover datos entre varios orígenes de datos diferentes o iguales. Entre las principales actividades que nos permiten hacer los DST's está la importación y exportación de datos entre dos orígenes de datos, la conversión de datos, copia de objetos de una base de datos, esta última actividad la puede realizar gracias que

los DTS's tienen una arquitectura OLE BD, con esto se puede trabajar con Oracle, SQL Server, Acces, Excel, Archivos de texto planos (.txt, .cvs).

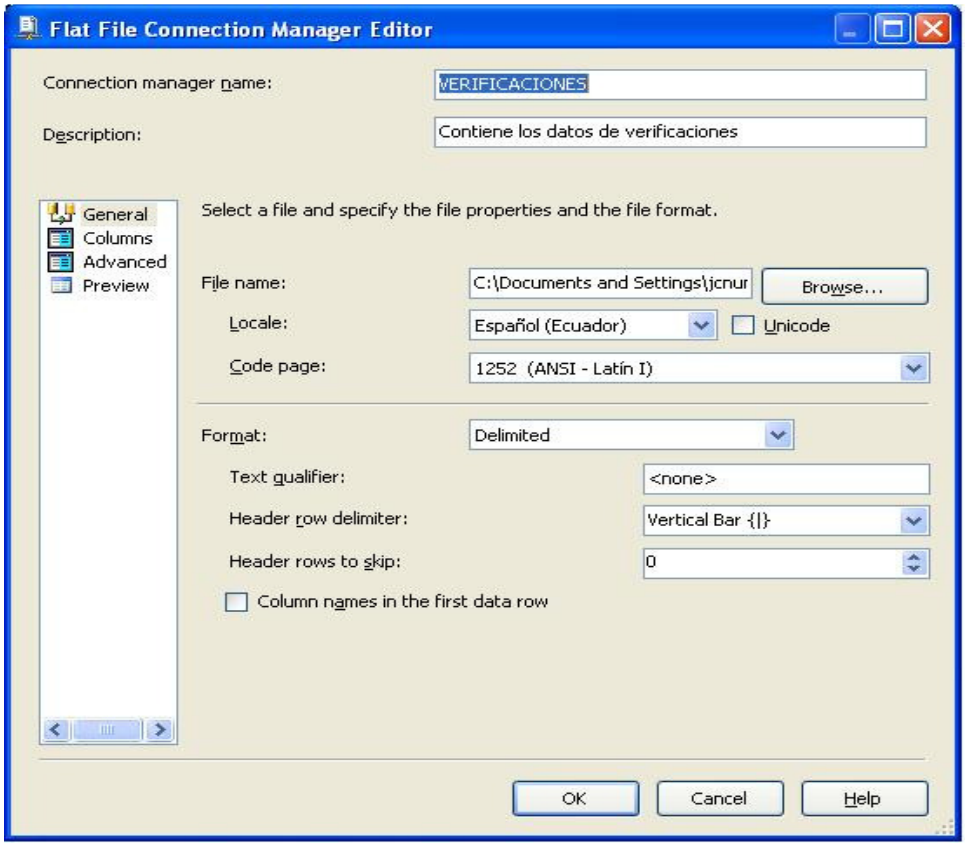
Se escogió la utilización de un DTS para importar los datos de las verificaciones desde un archivo plano a una tabla en la base de datos, esto facilita el trabajo de procesamiento de la información que proporcionan las empresas para que se realice la gestión de verificación, a continuación, se presenta el DTS con el que se trabajará:



Esta es la estructura básica del DTS, se obtienen los parámetros del archivo que vamos a leer, como el nombre, el path de origen del archivo, entre otros, después, se procede a verificar que el archivo no sea el mismo del día anterior, esto se lo realiza con un componente donde se escribe un script para realizar dicha verificación, si el archivo es nuevo, se pasa inmediatamente al flujo de datos, dentro del flujo de datos, se encuentran los siguientes elementos: un componente para leer los datos desde el archivo plano, un componente para convertir los datos al formato de la base de datos y un componente para insertar los datos del archivo, convertidos, en una tabla de la base de datos:



Para que componente pueda leer los datos del archivo plano, se utiliza una conexión, en donde se indica donde está el archivo plano y las columnas que debe leer parea insertar en la tabla de la base de datos:



Una vez que se le ha indicado al DTS de donde leer los archivos, se procede a convertir los datos esto se lo realiza con el componente para conversión de datos, básicamente lo que hace este componente es tomar los datos del archivo plano y convertirlos al formato que tiene cada columna en la tabla de la base de datos.

Una vez que se han convertido los datos, se procede a insertar los datos convertidos en una de las tablas de la base de datos, para esto se utiliza un componente de destino de base de datos, se le indica en qué base y tabla debe colocar los datos convertidos:

Arquitectura de Programación:

La arquitectura que se seleccionó para desarrollar el sistema es la arquitectura por capas, específicamente la arquitectura de 3 capas, este tipo de programación nos permite realizar cambios en cualquiera de las capas sin que estos afecten a las demás capas, evitándonos así tener que revisar entre código mezclado, esto hace que sea la arquitectura más popular en la actualidad.

En la arquitectura de tres capas, a cada nivel se le confía una misión simple, lo que permite que la arquitectura pueda ampliarse con facilidad en caso de que las necesidades aumenten, es decir que la arquitectura sea escalable. Todas las capas pueden estar en un solo ordenador, aunque lo más usual es que la capa de presentación resida en varios computadores ya que esta capa es la capa de los clientes de la arquitectura cliente/servidor. Es aconsejable que las otras dos capas estén en un solo ordenador aunque, si la necesidad o complejidad del sistema lo ameritan, se las puede separa en dos o más ordenadores, siendo el ordenador de la capa del de la capa del negocio quien realice las peticiones a los demás ordenadores. Como se dijo anteriormente, cada capa tiene una tarea diferente, en esta arquitectura de tres capas las tareas son: Capa de presentación o de GUI (graphic user interface), Capa de Negocio o DP (Dominio del Problema) y Capa de datos o MD (Manejo de Datos). A continuación una explicación de cada una de las capas:

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). Esta capa se comunica únicamente con la capa de negocio. También es conocida como interfaz gráfica y debe tener la característica de ser “amigable” (entendible y fácil de usar) para el usuario.

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 para 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.

La siguiente figura nos da una idea de cómo está construida la arquitectura de tres capas:

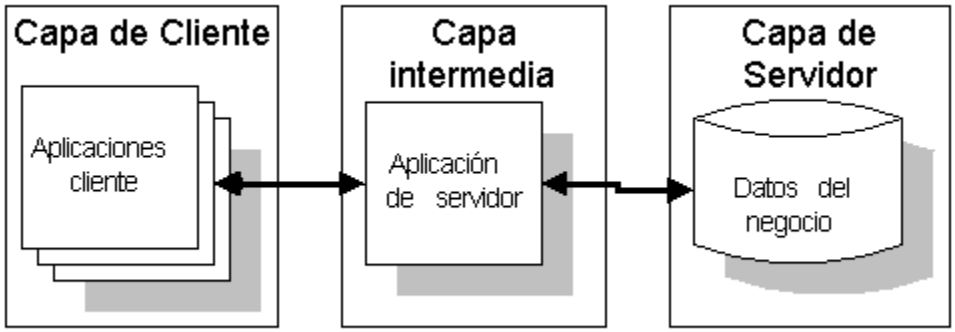


Figura 2: Estructura de la arquitectura de 3 capas

Como podemos observar en esta figura, es en la capa del dominio del problema (DP) en donde se recibe toda la información de las otras dos capas para que sea procesada. La siguiente figura nos muestra como está construida la arquitectura de tres capas en varios servidores:

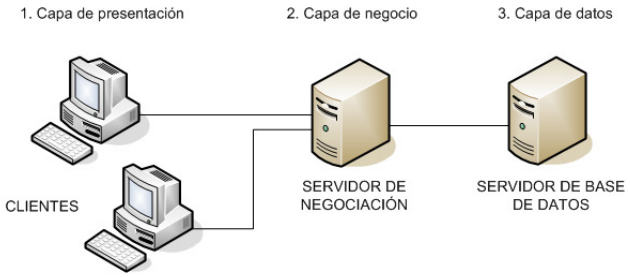


Figura 3: Estructura de la arquitectura de 3 capas en varios servidores

La Figura 3, nos muestra como está construida la arquitectura de 3 capas para varios servidores, esta variación de la arquitectura es utilizada cuando las aplicaciones demandan una gran transaccionalidad de datos como por ejemplo las aplicaciones utilizadas en las grandes empresas financieras.

La siguiente figura, muestra la aplicación de la arquitectura de tres capas para el funcionamiento del Sistema de verificación de información

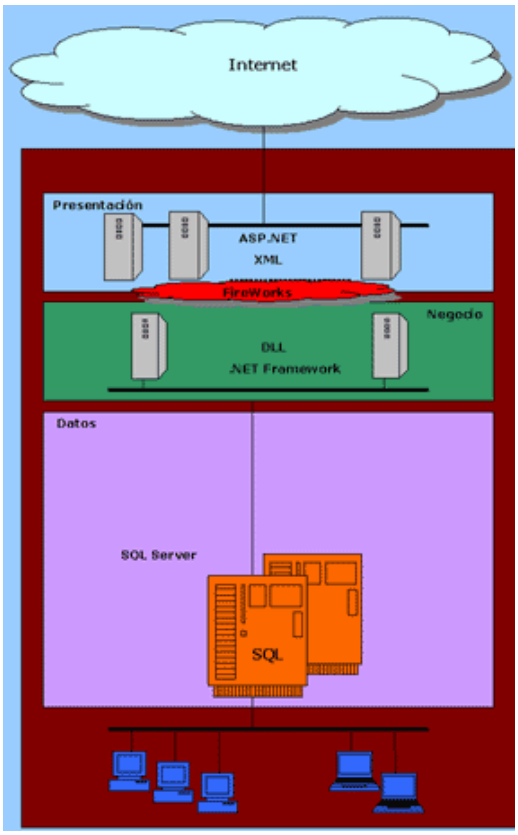


Figura 4: Aplicación de la arquitectura de 3 capas al sistema de verificación

En esta figura se puede observar claramente como las diferentes tecnologías utilizadas para el desarrollo del sistema funcionan y como cada una de las capas realizan sus tareas, a través de internet, los usuarios podrán ver la capa GUI y desde ahí, se realizan las peticiones a la capa del dominio del problema o DP, la cual a su vez, envía la información necesaria a la capa de manejo de datos o MD en donde la información es consultada en la base de datos y devuelta a la capa DP para que esta a su vez, envíe la información procesada a la capa GUI para que el usuario visualice los resultados.

De esta manera se tiene una estructura ordenada en lo que al código se refiere y si se requiere hacer un cambio en cualquiera de las capas, se podrá fácilmente identificar en cual de las capas se lo debe realizar, optimizando de esta forma el tiempo de programación.

Diagramas UML:

Este tipo de diagramas son de gran importancia para conocer cómo está estructurado un producto de software ya que nos muestran de una manera muy detallada, las relaciones que existen entre las diferentes clases, los usuarios que podrán acceder a los diferentes servicios de un programa, los pasos que deben seguir los usuarios para realizar diferentes tareas, etc. además estos diagramas nos sirven para realizar la documentación de un sistema de software.

UML significa Lenguaje Unificado de Modelado (Unified Modeling Languaje, en ingles) este lenguaje de modelado de sistemas de software es el más conocido y utilizado en la actualidad. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Cabe aclarar que UML no puede compararse con la programación estructurada, pues solo se diagrama la realidad de una utilización en un requerimiento. Mientras que, programación estructurada, es una forma de programar como lo es la orientación a objetos, sin embargo, la programación orientada a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML sólo para lenguajes orientados a objetos.

La siguiente figura nos muestra todos los diagramas que tiene UML y la jerarquía que existe entre ellos:

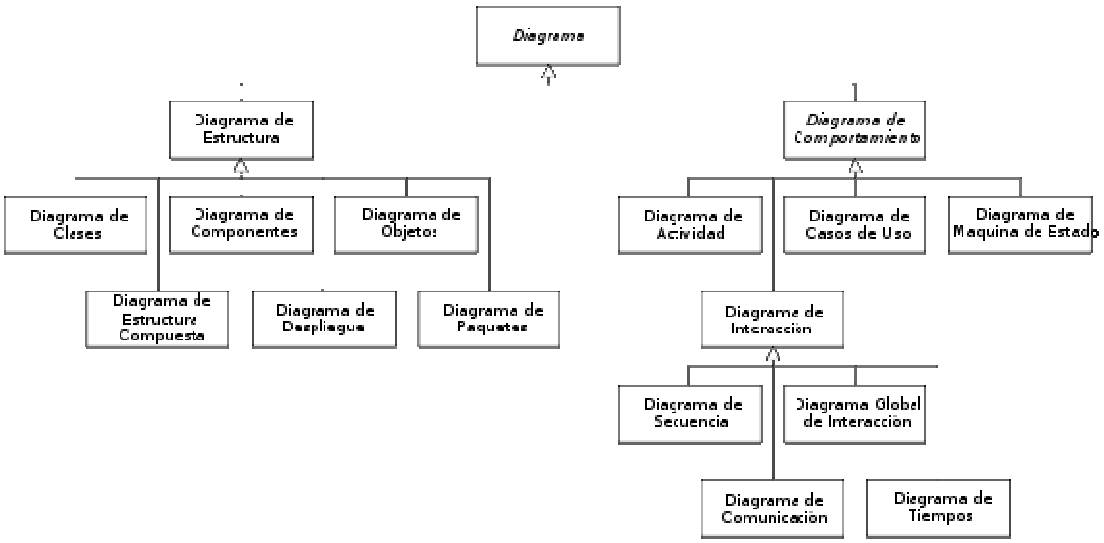


Figura 5: Jerarquía de diagramas UML

Para el modelado del sistema se utilizarán los siguientes diagramas:

- Diagramas de clase
- Diagramas de casos de uso
- Diagramas de secuencia
- Diagramas de procesos

A más de los diagramas anteriores, también se realizarán los modelos físico y conceptual de la base de datos del sistema de verificación.

Más adelante se explicará lo que hace cada uno de los diagramas seleccionados para el modelado del sistema.

Existen varios programas que permiten el modelado UML, los hay tanto en software libre como en software comercial, a continuación una lista de los programas más populares utilizados para el modelado UML:

- **Software libre:** Se permite su estudio y modificación, son gratuitos
  - StarUML
  - Fujaba
  - Papyrus
- **Freeware:** Bajo licencias que no permiten la modificación de los programas, son gratuitos
  - Visual Paradimg for UML
  - JUDE Community
- **Software Comercial**
  - Microsoft Visio
  - Power Designer
  - Poseidon for UML
  - MagicDraw

Como podemos observar, existe una gran variedad de software para modelar UML, los hay tanto en software libre como en software comercial, sin embargo, los estándares que tiene UML son conservados para todas las herramientas de modelado.

DESARROLLO DEL SISTEMA DE VERIFICACIÓN DE INFORMACIÓN DE CLIENTES PARA EMPRESAS DE COBRANZA

Una vez que se ha realizado una explicación sobre las diferentes herramientas y metodologías utilizadas para el desarrollo del sistema de verificación de información de clientes, se procede a detallar como se desarrolló el sistema en sus diferentes etapas desde la fase de diseño hasta la fase de pruebas en ambiente de desarrollo, los dos primeros capítulos son las dos primeras fases del ciclo de vida en cascada, los documentos que producen estas dos fases se encuentran detallados en los anexos.

1 CAPÍTULO 1 Análisis de Requerimientos del Sistema

En este primer capítulo, en primer lugar, se analiza la situación de ciertas empresas de cobranza para conocer cómo realizan la gestión de verificación de información de sus clientes y con este análisis, se procederá a realizar un levantamiento de los requerimientos necesarios para la realización del sistema de verificación de información así como también se generarán los primeros diagramas para tener una idea de cómo está estructurado el sistema, dentro de cada punto que trate sobre los diagramas UML se dará una breve explicación de qué hace cada diagrama .

1.1 Análisis de Situación Actual

En la actualidad existen empresas que prestan servicios de cobranza de las carteras vencidas en las instituciones financieras, dichas empresas, necesitan conocer y tener actualizada la información demográfica de cada uno de los clientes que están en la cartera a gestionar, lamentablemente por el volumen de las carteras, a veces es dificil mantener la información demográfica actualizada, esto produce que las gestiones de la cartera no sean óptimas ya que se pierde bastante tiempo visitando direcciones erradas o llamando a teléfonos que muchas veces son incorrectos.

Varias empresas de cobranza han pensado en la implementación de un sistema que permita a los gestores actualizar la información de los clientes para nuevamente visitarlos y de esta manera poder gestionar la cartera con información actualizada, esto permitirá a las instituciones financieras disminuir el volumen de la cartera vencida (La cartera vencida registra el valor de toda clase de créditos que por más de 30 días dejan de ganar intereses o ingresos. Una vez que una cuota se transfiere a cartera vencida todas las cuotas restantes por vencer y vencidas hasta 30 días, se reclasifican a la cartera que no devenga intereses. (Superintendencia de bancos y compañías, Boletines Financieros, Nota técnica 4)).

En el período comprendido entre el 30 de junio de 2008 y el 30 de junio de 2009, la cartera de créditos y contingentes registró un comportamiento crediticio desfavorable debido a: una expansión marginal de los créditos (1,82%), equivalente a USD 178 millones; un incremento de la morosidad en el 30%, al pasar de 2,42% al 3,15%; y, un aumento de la cartera en riesgo (vencida y que no devenga intereses) de USD 236,8 millones a USD 313,6 millones (32,4%), entre otros aspectos (http://web.superban.gov.ec/medios/PORTALDOCS/downloads/articulos_financieros/Estudios%20Sectoriales/comportamiento_sectorial_bancos_jun_09.pdf).

Como podemos observar, los índices de cartera vencida han aumentado en el último período de tiempo, esto hace que cada vez sea más difícil manejar la información demográfica de los clientes, es por esta razón que se pensó en realizar el presente trabajo, con el fin de mantener la información de los clientes actualizada, y de esta manera poder disminuir el tamaño de la cartera vencida

1.2 Levantamiento de requerimientos del sistema

Para realizar el levantamiento de los requerimientos del sistema, lo que se hizo fue visitar varias empresas de cobranza y conversar con las personas encargadas de realizar la gestión de cobranza sobre cómo almacenan la información demográfica de los clientes y con qué frecuencia la actualizan, también el proceso que tienen para obtener la nueva información demográfica de los clientes, de esta conversación se obtuvieron los siguientes requerimientos:

- El sistema debe permitir ingresar la nueva información demográfica obtenida por los visitadores y almacenar esta información en una base de datos.
- Se debe generar fichas de trabajo para los visitadores.
- Cada visitador debe tener un usuario y una clave que le permitirán ingresar a su lista de trabajo para actualizar la información.

- El sistema debe permitir ingresar una nueva ciudad la cual debe estar relacionada con un cantón.
- Se debe tener niveles en los usuarios del sistema, es decir, un administrador, un supervisor, y los gestores.

Estos son algunos de los requerimientos que se obtuvieron en las entrevistas con las personas de cobranza. Una vez que se realizó el levantamiento de los requerimientos, se procede a realizar los diagramas del sistema.

1.3 Generación de diagramas de clase

Estos son diagramas UML estáticos, los diagramas de clase describen la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Estos diagramas son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

El diagrama de clases incluye mucha más información como la relación entre un objeto y otro, la herencia de propiedades de otro objeto, conjuntos de operaciones/propiedades que son implementadas para una interfaz.

Al diseñar una clase se debe pensar en cómo se puede identificar un objeto real, como una persona, un transporte, etc. Durante el proceso del diseño de las clases se toman las propiedades que identifican como único al objeto y otras propiedades adicionales como datos que corresponden al objeto. Para comprender mejor la definición, tomemos como ejemplo la clase *"persona":*

Una persona tiene número de documento de identificación, nombres, apellidos, fecha de nacimiento, género, dirección postal, posiblemente también tenga número de teléfono de casa, del móvil, FAX y correo electrónico.

Los diagramas de clases para el sistema se encuentran en el documento de requerimientos del sistema, en los anexos, las clases que se presentan son a nivel conceptual, es decir, sin atributos ni métodos, esto corresponde a la fase de diseño, a continuación se presenta la clase cantones:

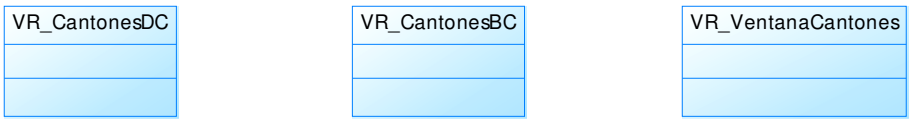


Figura 1.1: Diagrama de clases para Cantones

Como podemos observar, aquí se presentan las tres clases que representan a las tres capas de la arquitectura, la capa DC es la capa de Manejo de Datos, la capa BC es la capa del Dominio del Problema y la capa de Ventana es la capa GUI dentro de la arquitectura seleccionada, en la fase de diseño se explica el significado de los nombres para cada una de las capas.

1.4 Generación de diagramas de casos de uso

UML tiene definida una notación gráfica estándar para la realización de los diagramas de casos de uso, estos diagramas son la descripción escrita del comportamiento que tiene el sistema en cada tarea o requerimiento, es decir, el valor suministrado por el sistema a entidades externas tales como usuarios humanos u otros sistemas.

Un conjunto de casos de uso promueve una imagen fácil del comportamiento del sistema, un entendimiento común entre el cliente-propietario-usuario y el equipo de desarrollo. Es importante señalar que **La interacción entre actores no se ve reflejada** en el diagrama de casos de uso, esta puede ser parte de suposiciones usadas en el caso de uso. Sin embargo, los actores son una especie de rol, un usuario humano u otra entidad externa. Los casos de uso están representados por elipses y los actores están representados por las figuras humanas.

Figura 1.2: Representación de un caso de uso

Figura 1.3: Representación de un actor

A continuación, se presentará el diagrama de casos de uso general del sistema y los casos de uso a detalle para la administración de clientes.

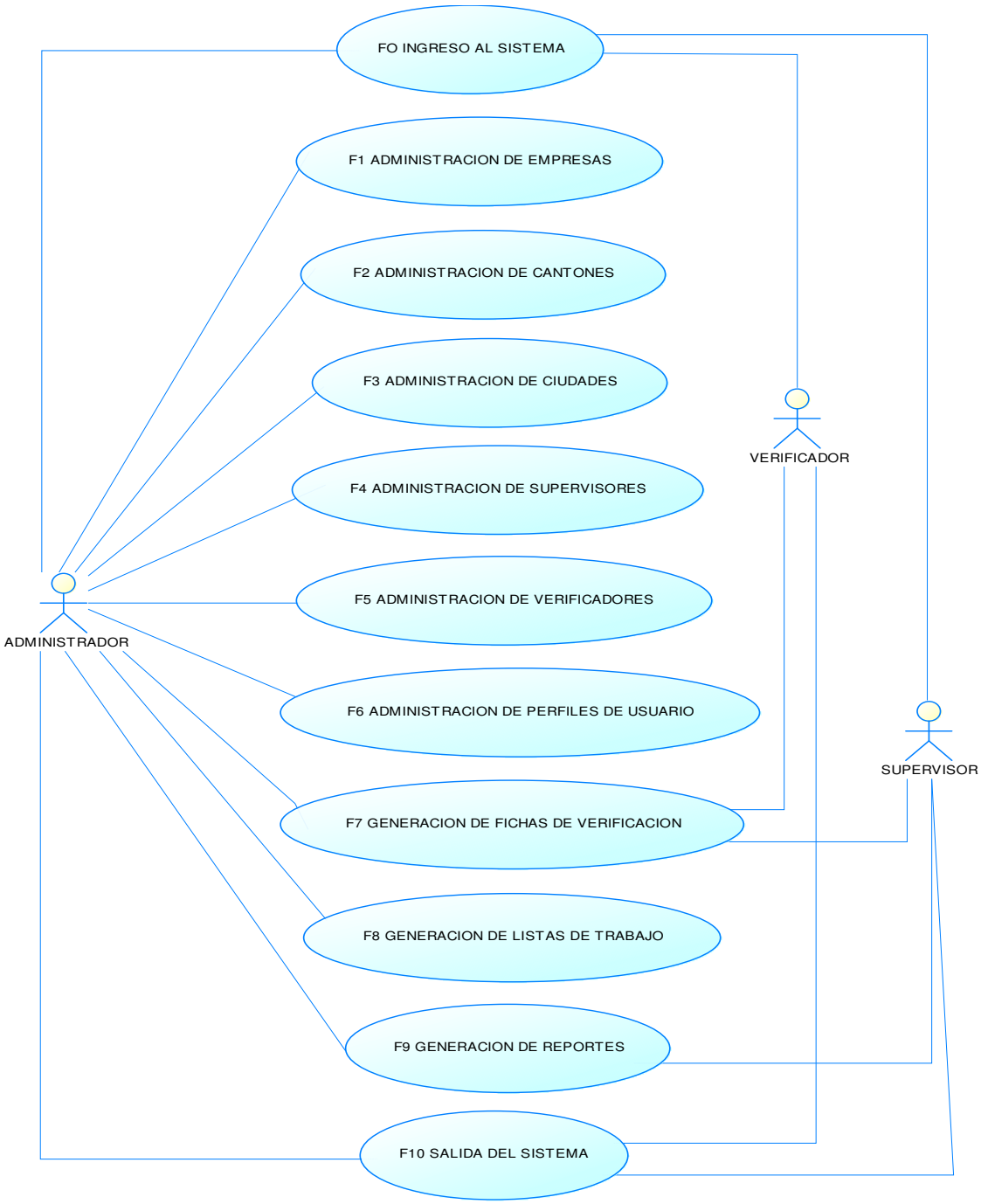


Figura 1.4: Diagrama general de casos de uso del sistema de verificación de información

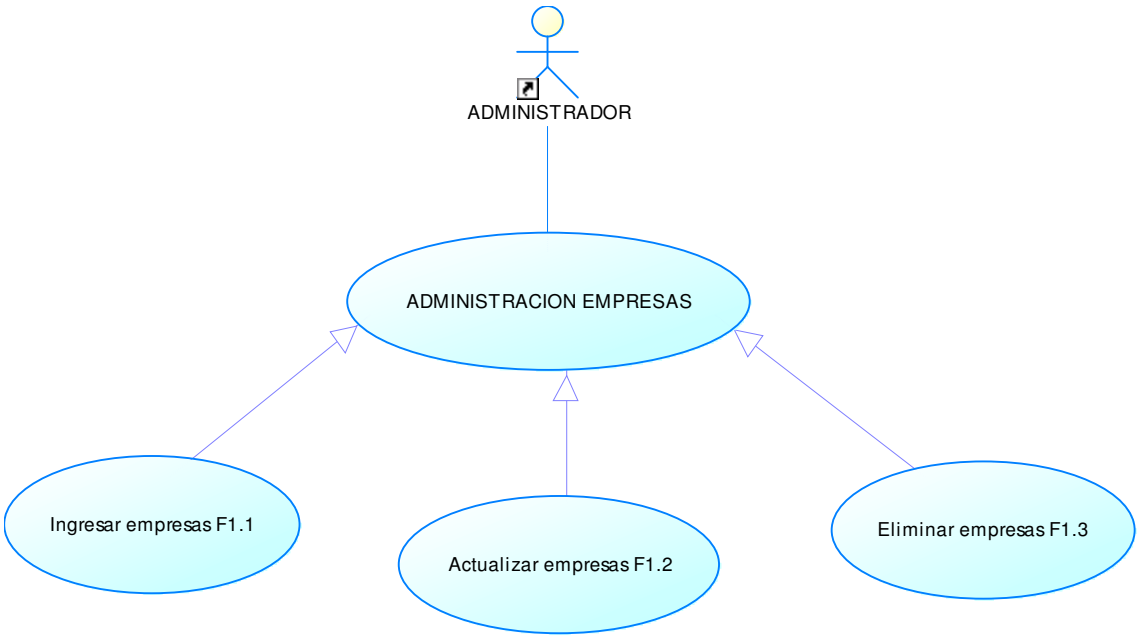


Figura1.5: Diagrama de casos de uso de administración de empresas

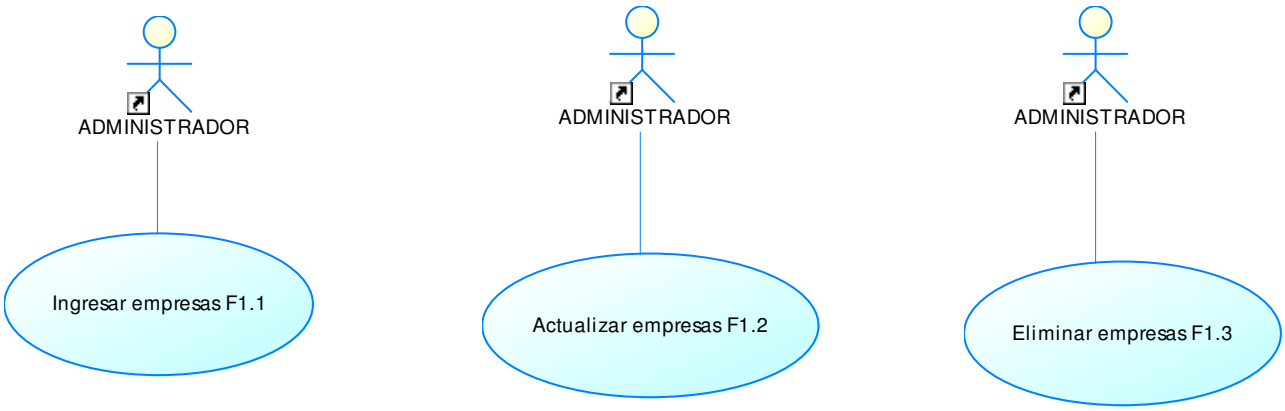


Figura1.6: Diagrama a detalle de casos de uso

Para cada caso de uso, se debe tener una explicación a detalle de cómo funciona, a continuación la explicación a detalle para en caso de uso ingreso de empresas:

Título: Administración Empresas
Caso de uso: Ingresar Empresas
Identificador: F1.1
Este caso de uso sirve para el ingreso de datos
Una nueva empresa
Lo utiliza el administrador del sistema

FLUJO

ACTOR		SISTEMA		
PASO	**ACCION**	**PASO**	**ACCION**	**Excepción**
1	Se emite un evento de ingreso de empresas	2	El sistema presenta plantilla para llenar la información	
3	Se ingresa código de la empresa	4	El sistema valida que el código sea numérico	E1
5	Se ingresa el nombre de la empresa			
6	Se emite un evento de grabar datos	7	El sistema valida que los datos sean correctos	E2
8	Se corrigen los datos mal ingresados			
9	Se emite un evento de grabar datos	10	El sistema valida que los datos sean correctos	
		11	El sistema confirma que los datos han sido grabados	

id	NOMBRE	ACCION
E1	Error formato de datos	Informa al usuario que los datos ingresados no son numéricos
E2	Error datos ingresados	Informa a usuario que los datos ingresados son incorrectos

Tabla 1.1 Explicación del funcionamiento del caso de uso Ingreso de empresas

Una vez que se tienen definidos el funcionamiento de los casos de uso, se debe hacer un cuadro de pruebas del sistema con los resultados esperados en cada una de las funcionalidades del sistema, este plan de pruebas se hace por cada caso de uso del diagrama general, a continuación el plan de pruebas del caso de uso de administración de empresas:

ENTRADAS	**RESULTADOS ESPERADOS**	**CASO USO**
Selecciona opción Adm. Empresas	Desplegar plantilla de empresas	F1
Ingreso de clientes		
correcto	Datos almacenados	
incorrecto	Mensaje de error	F1.1
Actualización de datos		
correcto	Actualizar datos	
incorrecto	Mensaje de error	F1.2
Eliminar datos	Se elimina la información seleccionada	F1.3

Tabla 1.2 Plan de pruebas de Administración de empresas

Los casos de uso completos para las demás administraciones así como para la generación de fichas de verificación y los reportes y los planes de prueba, se encuentran en el anexo 1, que es el documento de requerimientos de software (SRS).

Después de haber definido a todos los casos de uso, se deben realizar algunos prototipos de pantalla para presentarlos al usuario, estos prototipos nos ayudan a darle al usuario una idea de cómo serán las pantallas de presentación para la información, a continuación, el prototipo de interface para la ficha de verificación:

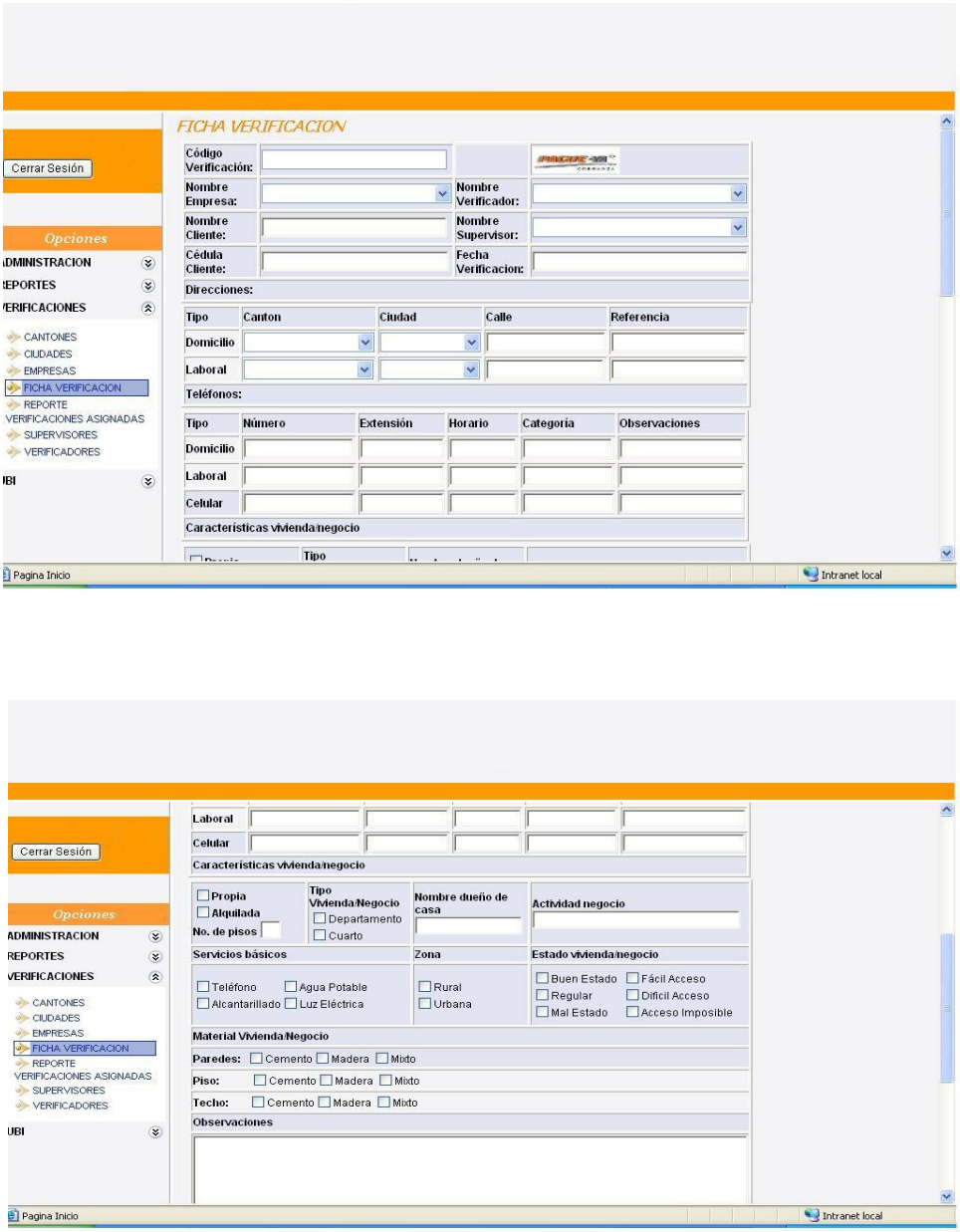


Figura 1.7: Prototipo de pantalla para la ficha de verificación

Esta figura nos muestra como sería la ficha de verificación que se les entregará a los verificadores para que vayan a hacer su gestión diaria, la información se cargará automáticamente en cada campo para el momento de imprimir las fichas.

Una vez que se han generado los diagramas de la etapa de requerimientos, se procede a generar, en base a los diagramas anteriores, los diagramas de la etapa de diseño que son los diagramas de clase con sus métodos y atributos, los diagramas de secuencia y los diagramas de paquetes. Los diagramas completos de clases y los de secuencia se encuentran en el anexo 2 en el documento de especificaciones de diseño de software.

Los primeros diagramas son los diagramas de clases con sus métodos y atributos, se presentan los diagramas de la clase cantones:

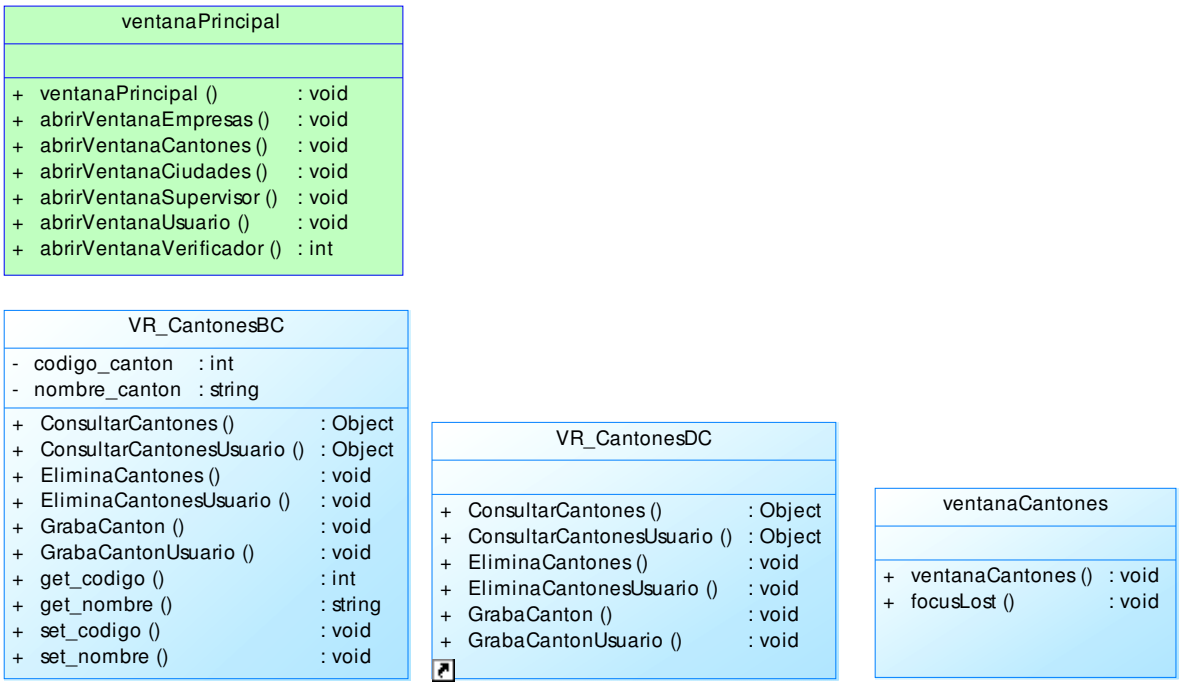


Figura 1.8: Diagrama de clases de la clase cantones

Como podemos observar, se presentan todas las clases con sus métodos para los cantones, la clase ventana principal es una clase genérica para todas las ventanas.

1.5 Generación de diagramas de secuencia

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada método de la clase. Mientras que el diagrama de casos de uso permite el modelado de una vista macro del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos.

Un **diagrama de secuencia** muestra los objetos que intervienen en el escenario con líneas discontinuas verticales, y los mensajes pasados entre los objetos como flechas

horizontales. Se los debe realizar uno por cada caso de uso, a continuación se presentan los diagramas de secuencia para la administración de cantones:

1.5.1 Ingresar Cantones:

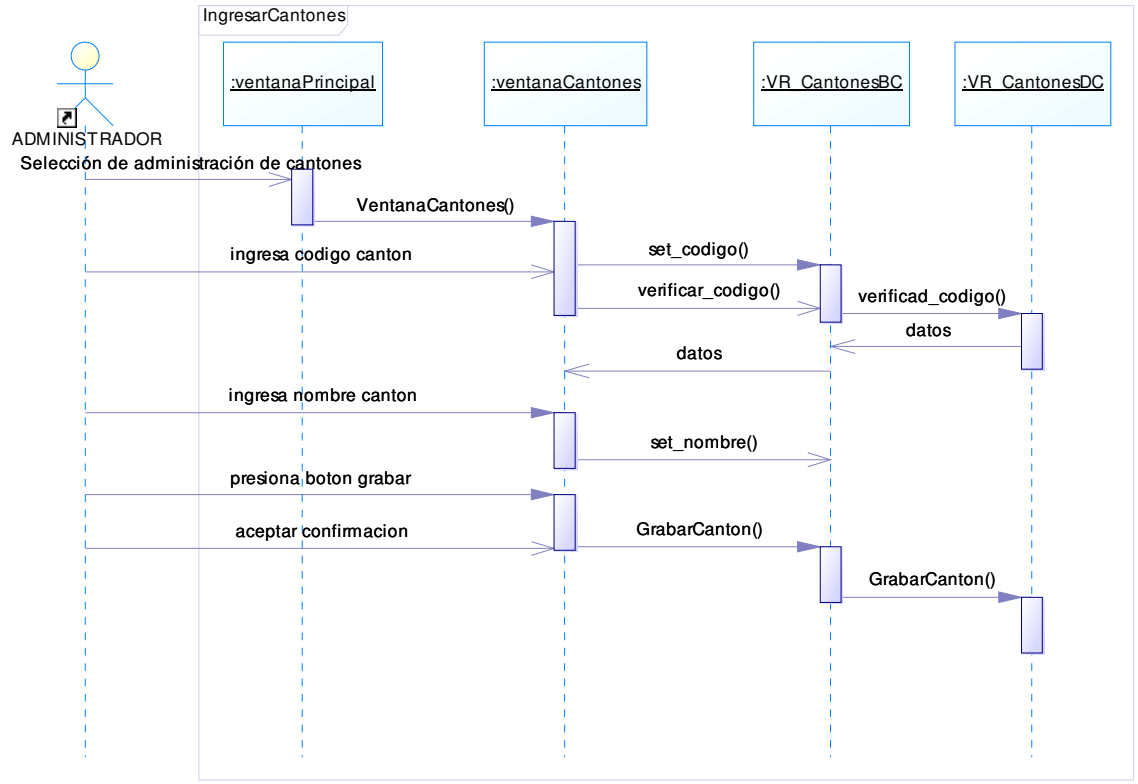


Figura 1.9: Diagrama de secuencia para el ingreso de cantones

Como podemos observar, en este diagrama de secuencia se presentan cada uno de los objetos que intervienen en este caso de uso, se puede ver las llamadas a los diferentes métodos que se necesitan para poder realizar el ingreso de un nuevo cantón. Se puede observar también la interacción que existe entre las tres capas de la arquitectura seleccionada para el desarrollo del sistema, la primera clase llamada ventana principal, pertenece a la capa de interface de usuario.

1.5.2 Actualizar Cantones:

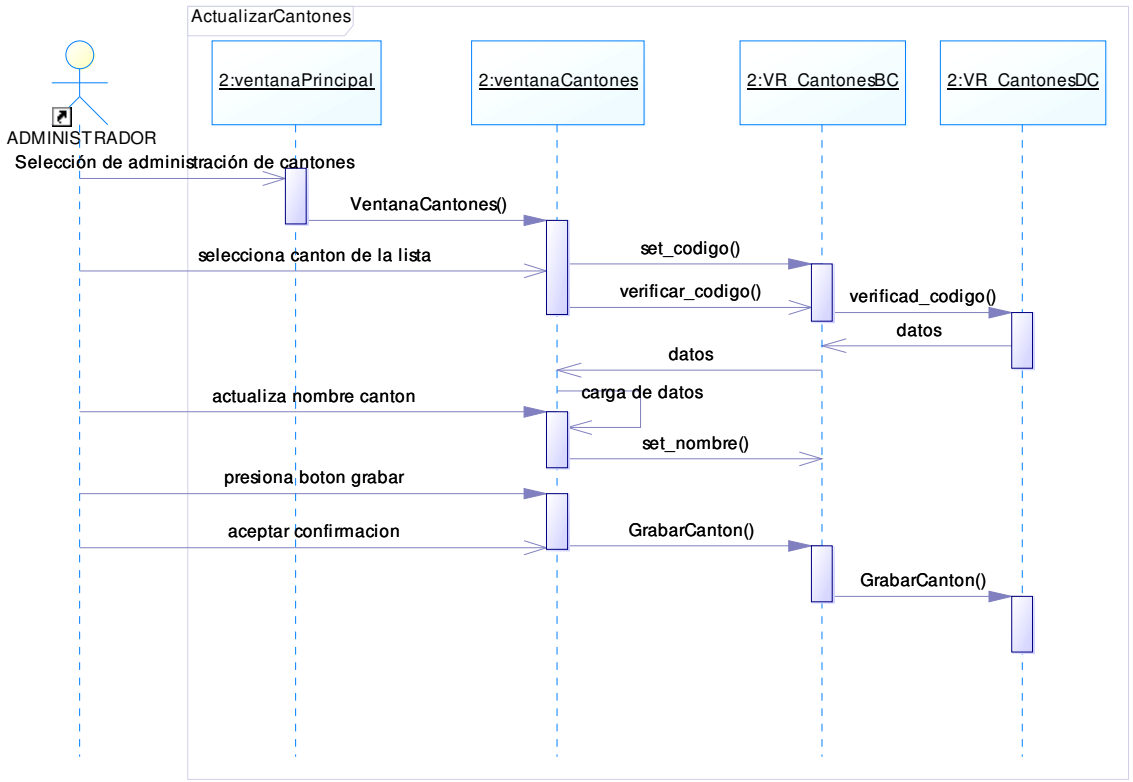


Figura 1.10: Diagrama de secuencia para la actualización de datos de los cantones

Este diagrama es muy similar al diagrama de ingreso de un nuevo cantón, la diferencia entre este diagrama y el anterior es que el usuario únicamente puede cambiar el nombre del cantón mas no en código del mismo.

1.5.3 Eliminar Cantones:

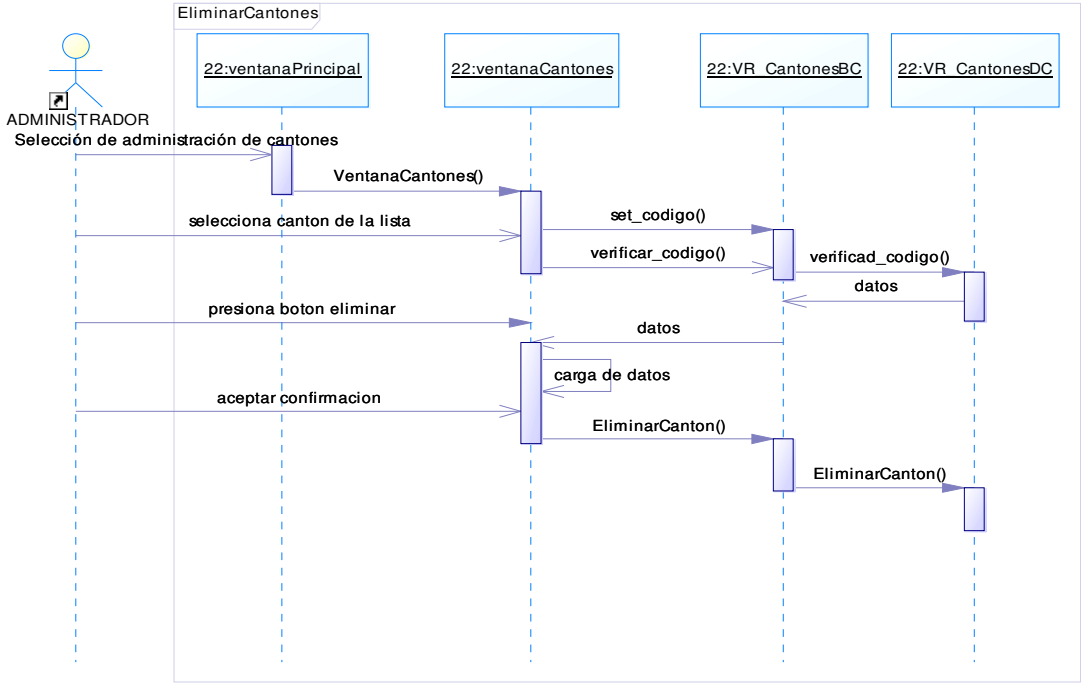


Figura 1.11: Diagrama de secuencia para el caso de uso de eliminar cantones

Este diagrama nos muestra como interactúan los objetos para eliminar un cantón de la base de datos, como podemos observar, el usuario no ingresa ningún nombre, únicamente selecciona un cantón de la lista y lo elimina.

Los demás diagramas de secuencia se encuentran en el anexo 2, documento de especificaciones de diseño de software

1.6 Generación de diagramas de paquetes

Un diagrama de paquetes muestra como un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema.

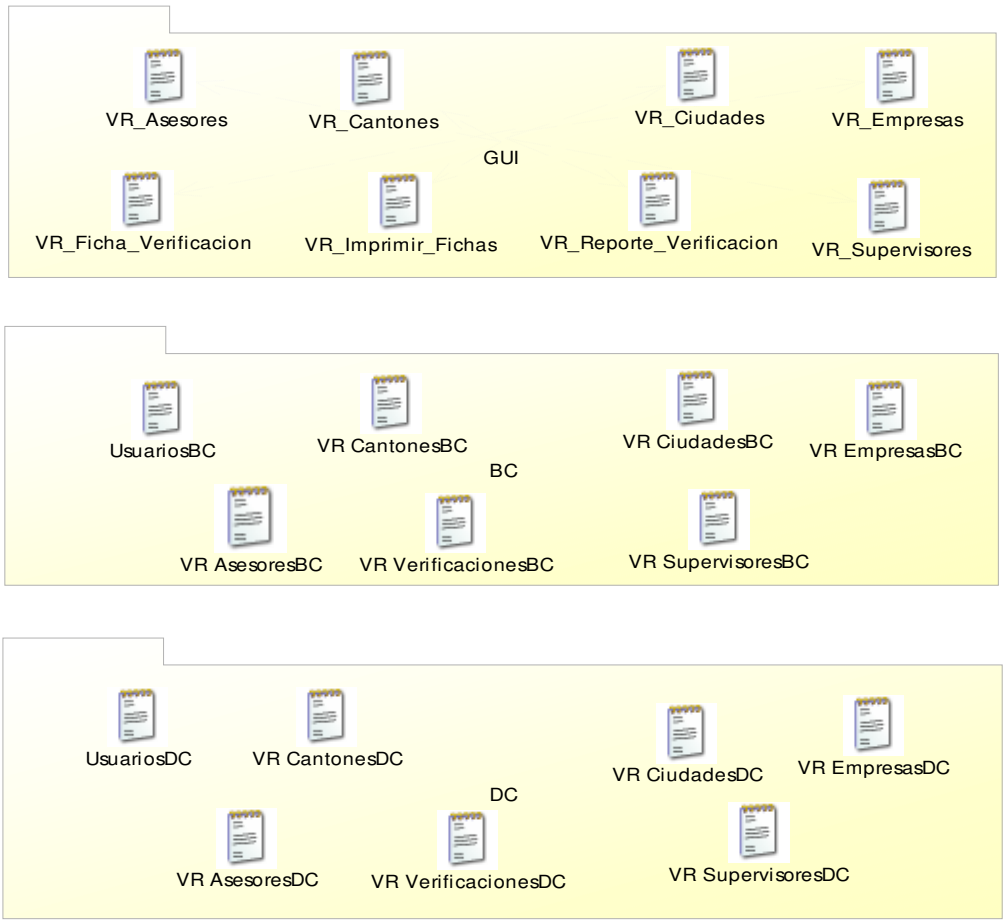


Figura 1.12: Diagrama de paquetes del sistema de verificación de información

Los Paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre los paquetes. Con estas líneas maestras sobre la mesa, los paquetes son buenos elementos de gestión. Cada paquete puede asignarse a un individuo o a un equipo, y las dependencias entre ellos pueden indicar el orden de desarrollo requerido.

2 CAPÍTULO 2 Diseño del Sistema

En este capítulo se realizarán los diseños físico y conceptual del sistema, es decir, se realizará el modelo de la base de datos del sistema con todas las entidades necesarias y las relaciones que existen entre ellas, para saber cuáles son las entidades necesarias nos basamos en la información obtenida en el capítulo anterior, a más de los diseños físico y conceptual, en la etapa de diseño se definen los estándares de desarrollo del sistema, es decir, los nombres que tendrán las variables, los nombres de los componentes, presentación de datos para el usuario, etc.

2.1 Modelo conceptual

En el modelo conceptual se trata de obtener el esquema conceptual de la base de datos a partir de la lista descriptiva de objetos y asociaciones identificadas en la organización durante el análisis.

Se podría definir al modelo conceptual como el conjunto de conceptos y de reglas destinados a representar de forma global los aspectos lógicos de los diferentes tipos de datos existentes en la realidad que está siendo analizada.

El modelo conceptual permite reflejar el contenido semántico de los datos existentes en el sistema, pero no sus propiedades que respondan a características de tipo físico (modo de almacenamiento, caminos de acceso, etc)

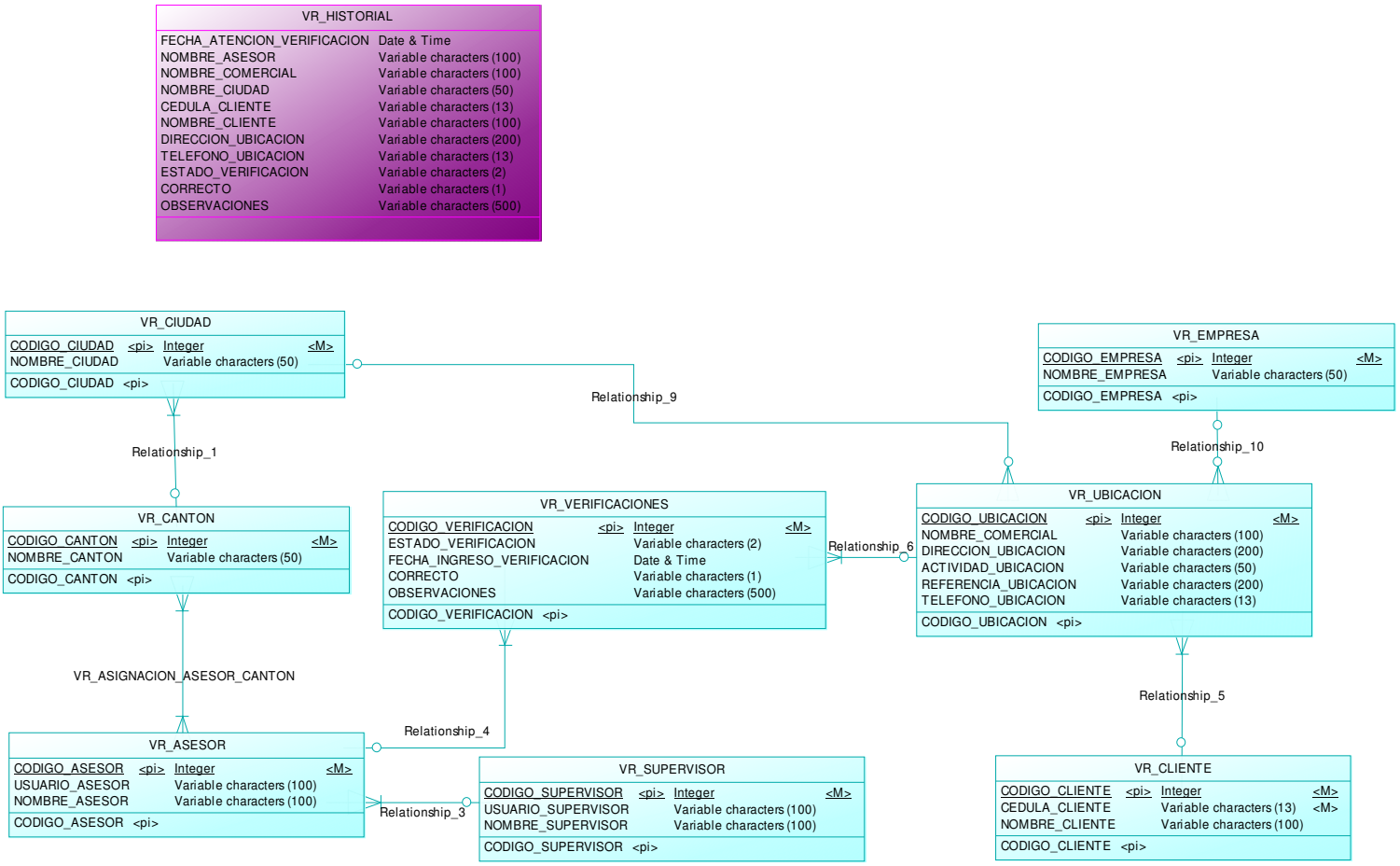


Figura 2.1: Modelo conceptual del sistema

2.2 Modelo físico

Una vez establecido el modelo conceptual del problema o situación, el diseño lógico de los datos permite que estos se puedan representar usando de manera eficiente posibles recursos para estructurar datos y modelar restricciones disponibles en el modelo lógico. El objetivo es convertir el esquema conceptual de datos en un esquema lógico que se ajuste al gestor de la base de datos que va a ser utilizado

El modelo físico es un esquema que presenta de forma conceptual la estructura de una base de datos. Es un esquema que depende del tipo de SGBD que se vaya a utilizar a utilizar.

Se crea a partir del modelo conceptual y serviría para cualquier base de datos comercial del tipo elegido en el esquema (hay esquemas relacionales, en red, jerárquicos).

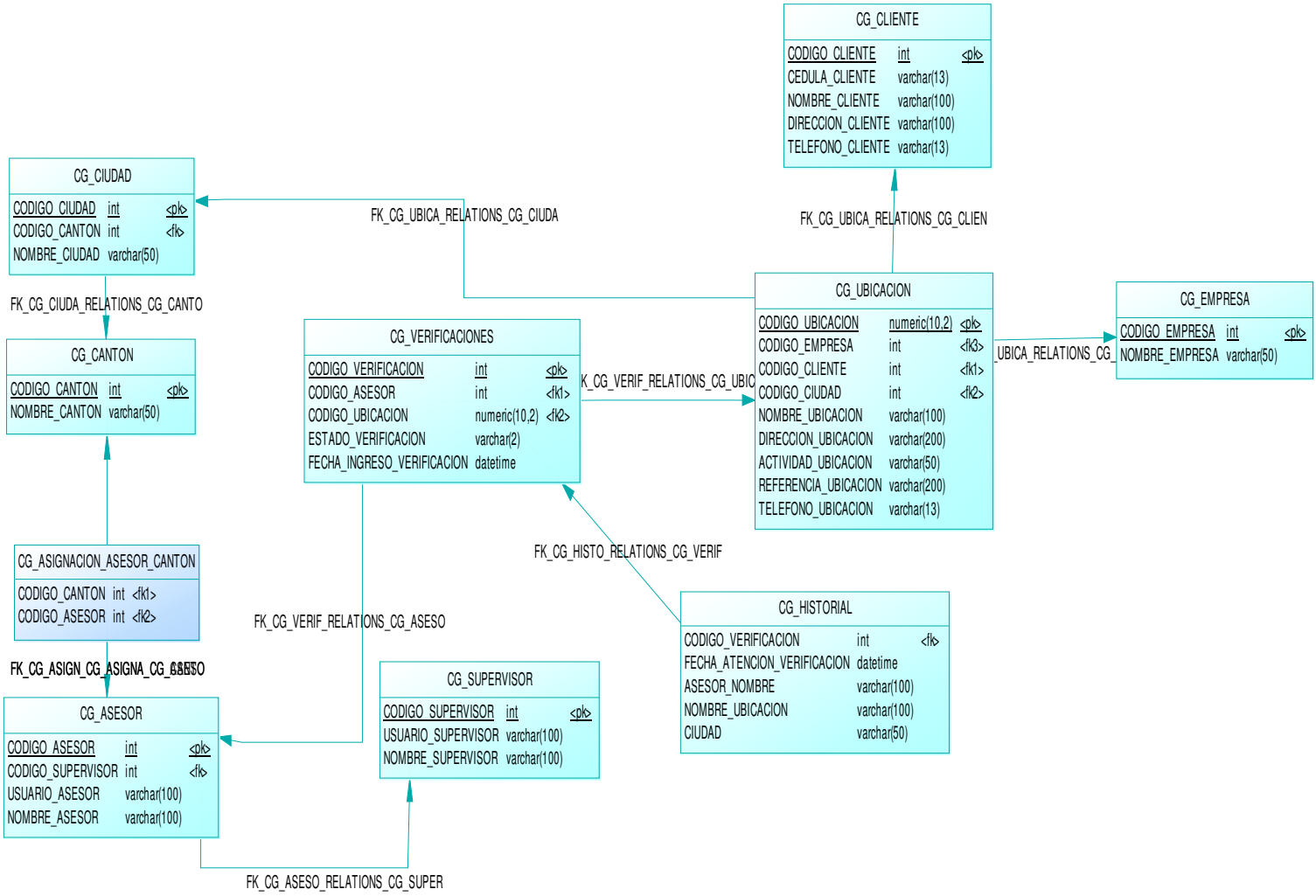


Figura 2.2: Modelo físico del sistema

2.3 Plan de pruebas de integración:

El objetivo principal de las pruebas de integración es detectar las fallas de interacción entre las distintas clases que componen al sistema. Debido a que cada clase probada unitariamente se inserta de manera progresiva dentro de la estructura, además de que a la par se van siguiendo los lineamientos dictados por el diseño, las pruebas de integración son realmente el mecanismo para comprobar el correcto ensamblaje del sistema completo. Al efectuar la integración de los módulos, se debe concentrar la búsqueda de defectos tales como aquellos que puedan provocar las excepciones arrojadas por los métodos; el empleo de operaciones equivocadas, por ejemplo el método *a.m* en vez de *a.h*; e invocación inadecuada de los métodos, es decir, pasaje de parámetros equivocados.

Para la realización de pruebas de integración en sistemas orientados a objetos se cuenta con dos estrategias diferentes. A la primera se le llama **integración por hilos** o integración por reacción directa o indirecta a un evento; consiste en reunir el conjunto de clases requerido para responder a la cadena de mensajes provocada por un evento del sistema, para luego integrarlas y probarlas individualmente. La segunda estrategia es **la integración por dependencia** de la clase, en la cual se comienza el ensamblaje del sistema probando aquellas clases, llamadas independientes, que usan muy poca o ninguna otra clase del sistema; después de que las clases independientes han sido probadas se continúa con la próxima etapa de clases, las cuales usan a las independientes (por ello se les llama clases dependientes). Esta secuencia de pruebas por capas de clases dependientes continúa hasta que el sistema entero se construye.

En el caso de la integración por dependencia de clases, el modelo de uso sirve de guía para la selección del orden en que se efectuará la integración de las clases y sus métodos. La idea es precisamente tratar de unir las clases instanciadoras de aquellos objetos que se requieren para un caso de uso, pero siguiendo la estrategia de las etapas de clases dependientes. El modelo de uso puede ayudar a reducir la complejidad de las precedencias en las pruebas, ya que el grafo descrito por un caso de uso es, por lo general, sencillo, y consta mayormente de "alternativas" y muy ocasionalmente de iteraciones. Con un grafo sencillo se puede emplear el método de la cobertura ciclomática de caminos (CCC), con la misma idea de búsqueda de fronteras y aplicación de "malicia" (dada por experiencias previas) al igual que se hizo con las pruebas unitarias.

En relación a la planificación de las pruebas de integración, es importante tomar en cuenta varios aspectos, tales como el grado de complejidad de las clases, su importancia funcional con respecto a las especificaciones del sistema, la cantidad de módulos que usan o dependen de cada clase, y las estadísticas de error en la fase de pruebas unitarias. En este último caso, el objetivo es tratar de integrar primero las clases que incurrieron en mayor número de fallas, ya que, aun cuando los errores en las pruebas unitarias hallan sido solventados, son las clases más susceptibles de contener errores en la integración porque pudieron cambiar algunas condiciones de entrada y/o salida que no fueron reportadas a tiempo.

Por otro lado, para establecer la interrelación de los módulos del sistema, plantear el grafo de dependencias o colaboración de las clases puede resultar en un diagrama bastante esparcido y extenso, además de contener múltiples regiones no conectadas; en este caso los grafos deben ser cuidadosamente analizados.

Para la realización de este sistema, la integración que se ha seleccionado es la integración por dependencias ya que este modelo nos ayudará como guía para saber cuales clases de deben integrar primero para poder integrar totalmente el sistema, a continuación se presenta el diagrama de integración por dependencias:

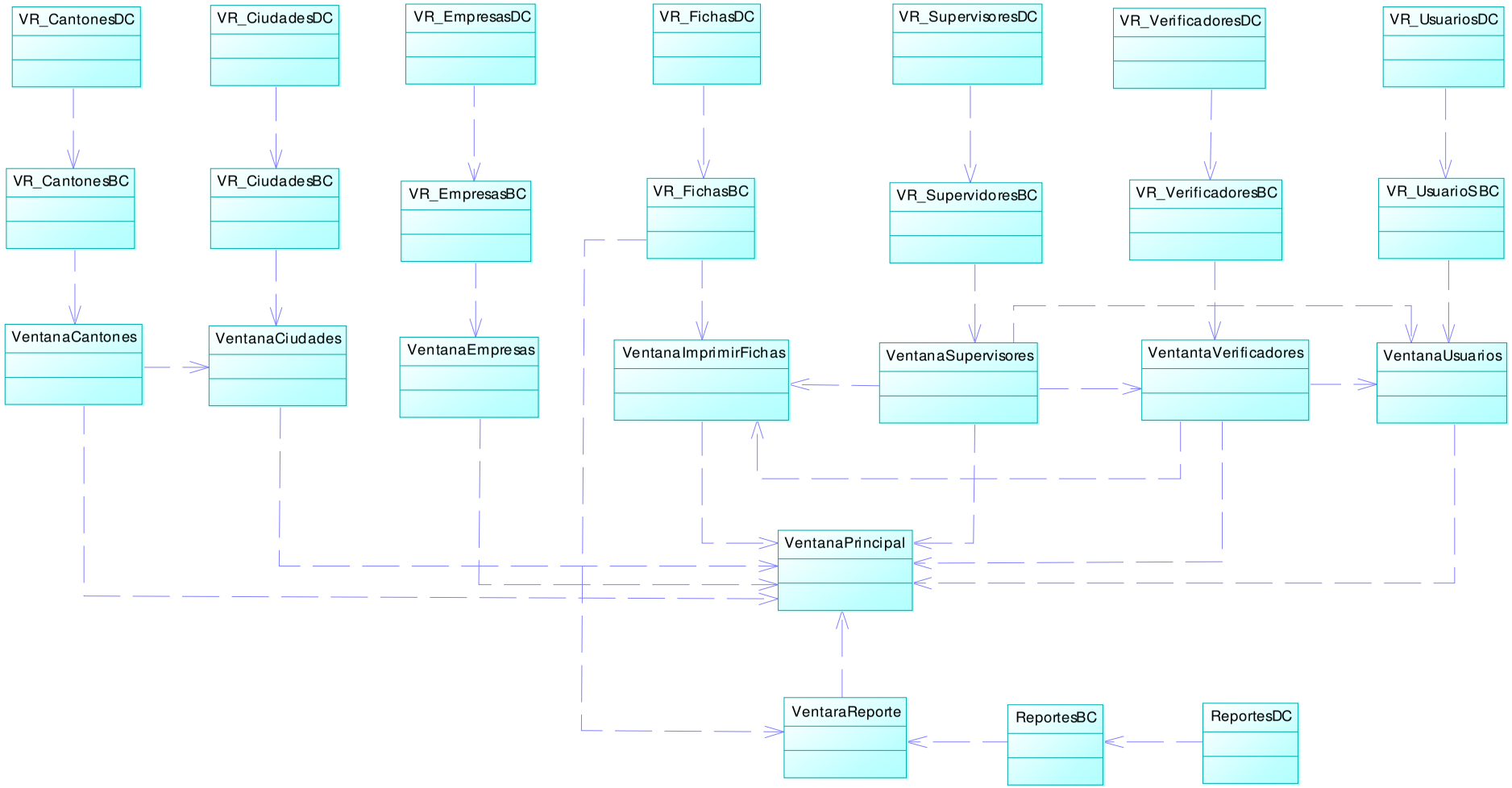


Figura 2.3: Integración por dependencias

Como podemos observar en el plan de pruebas de integración, nos podemos guiar para saber cuales clases deben ser implementadas primero para poder continuar con el desarrollo del sistema y así, al momento de integrarlo, no tener conflictos con los códigos fuentes.

2.4 Definición de estándares de diseño y desarrollo

En el proceso de desarrollo de software, una de las partes más importantes dentro de la etapa de diseño es la definición de estándares de diseño y desarrollo ya que se esta manera se tendrá un código fuente ordenado y se podrá identificar fácilmente que tipo de variable es o las diferentes clases a la que estas variables pertenecen.

A continuación la definición de estándares para el sistema:

2.4.1 Nombre de las variables:

Las variables deben empezar con las iniciales del tipo de dato asignado en letras minúsculas y el nombre de la variable, la primera letra en mayúsculas:

VR_EmpresasDC objEmpresasDC = new VR_EmpresasDC();

Para las variables de la base de datos el estándar será: @i_USUARIO

2.4.2 Nombre de los métodos:

Los nombres de los métodos deberán ser escritos sin guiones bajos y con palabras completas, la primera letra de cada palabra se la debe escribir con mayúscula y si el método recibe parámetros, estos se los debe declarar utilizando el estándar de declaración de variables:

```
public void EliminaEmpresas(int intEmpresa, string strUsuarioAsignacion) { }
```

Los métodos deben estar comentados con una breve explicación de que es lo que realiza cada uno:

```
/// <summary>
/// Método de la capa de 40ódigo que se encarga de llamar a la capa de datos para eliminar
/// las empresas
/// </summary>
/// <param name="strUsuario">40ódigo de usuario</param>
/// <param name="strUsuarioAsignacion">40ódigo de usuario asignación</param>
```

2.4.3 Nombre de las clases:

Los nombres de las clases deben empezar con VR_ para indicar que pertenecen al módulo de verificaciones, el nombre de la clase se lo debe escribir con la primera letra en mayúscula y si pertenece a la capa de las reglas del negocio, deben terminar con las iniciales BC (Business Class) o si

pertenece a la capa de manejo de datos, debe terminar con las iniciales DC(Data Class)

public class VR_EmpresasBC{}

2.4.4 Nombre de los componentes de interfaces:

Los nombres de los componentes de las interfaces del sistema se los debe escribir con las iniciales del tipo de componente que es en letras minúsculas y el nombre del componente con la primera letra en mayúsculas:

Como ejemplo, para un componente ultra web grid la nomenclatura es: uwgCanton.Columns[0].HeaderText = “Código Cantón”;

2.4.5 Interfaces de usuario:

Las interfaces de usuario serán presentadas en español, los textos de los botones se los escribirá la primera letra en mayúsculas y las demás en minúsculas, los botones de las interfaces deben tener textos de ayuda (tool tip text) para facilitar la comprensión de los usuarios del funcionamiento del módulo. Los menús de despliegue deberán ubicarse en la parte izquierda de la pantalla

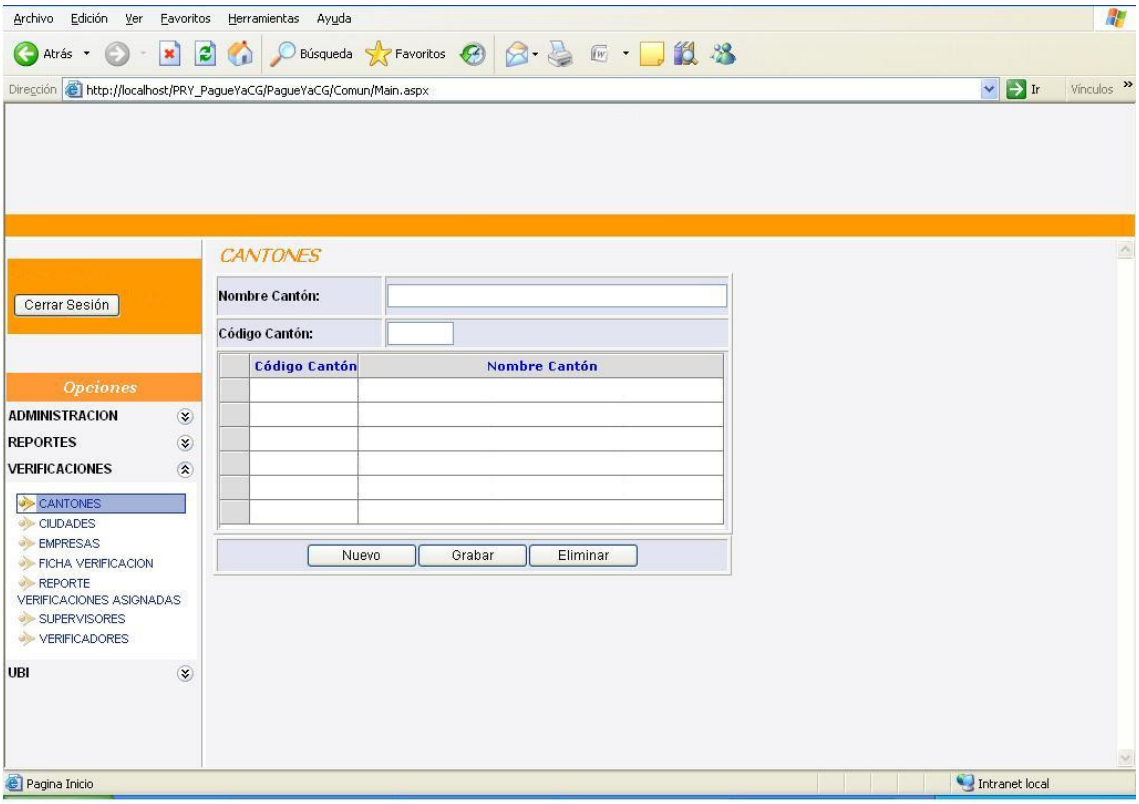


Figura 2.4: Estándares de interface de usuario

2.4.6 Organización de las clases dentro del ambiente de desarrollo:

Las clases deben estar ubicadas en carpetas que las diferencien para no tener confusiones el momento de abrir una u otra clase:

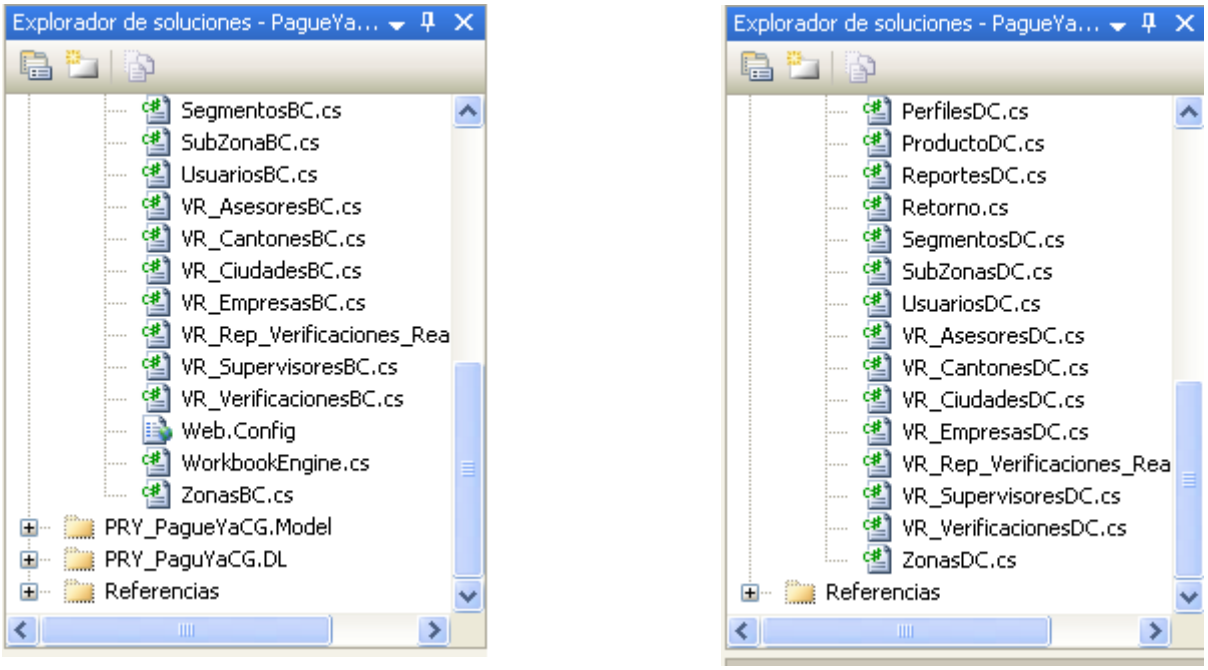


Figura 2.5: Organización de las clases dentro del ambiente de desarrollo

La presentación de los datos se la debe realizar en un componente ultra web grid para que el usuario seleccione de la lista de datos un registro y su información sea cargada en los campos de la interface:

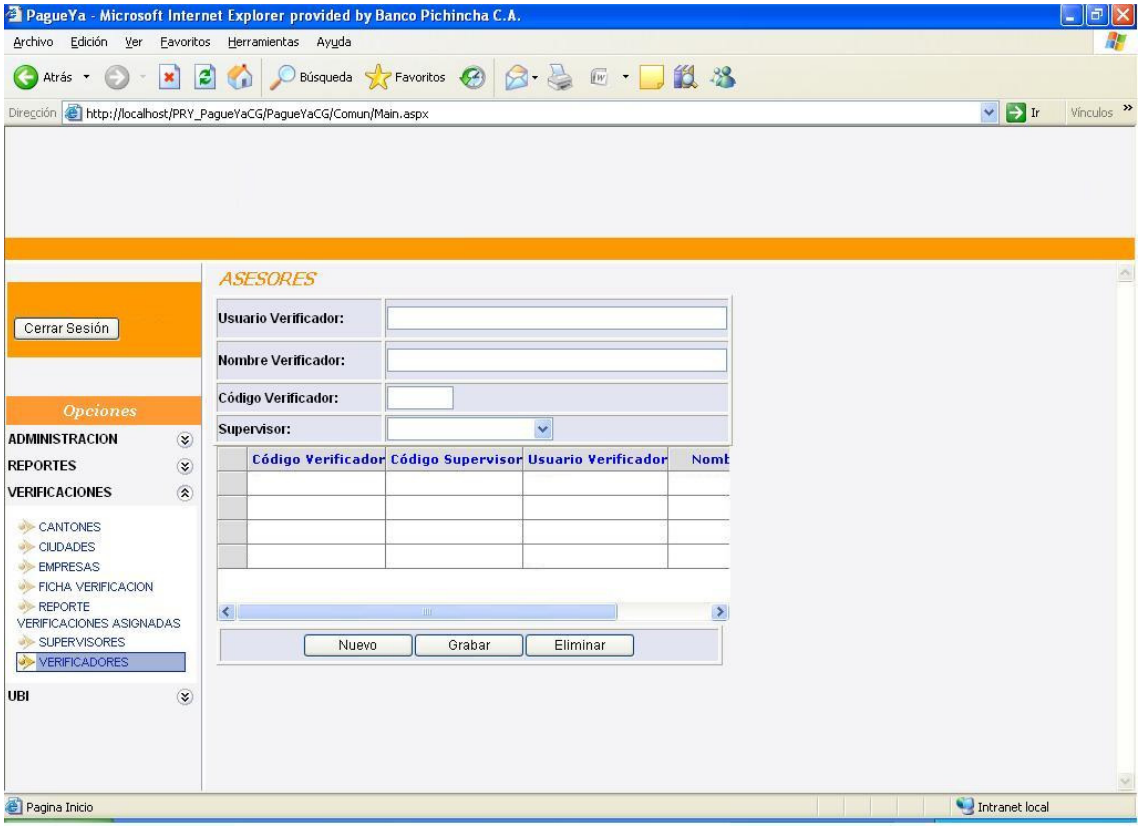


Figura 2.6: Presentación de los datos

Las advertencias que presente el sistema deben estar en español y escritas en minúsculas, únicamete la primera letra del mensaje irá con mayúscula:

Figura 2.7: Estándar de advertencias para ser presentadas en la aplicación

2.4.7 Estándares de nombres de tablas:

Las tablas en la base de datos tendrán el siguiente estándar de nombre: VR_<<NOMBRE DE LA TABLA>>, los nombres de los campos, se los debe escribir con palabras completas y en mayúsculas como ejemplo, veamos la tabla empresas:

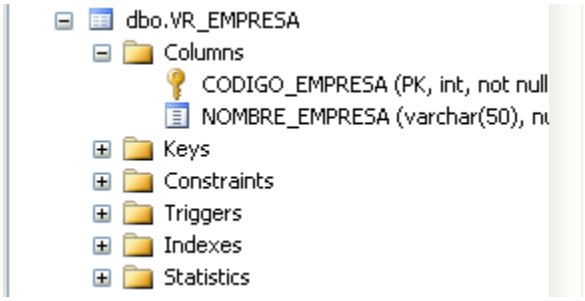


Figura 2.8: Estándar para nombremiento de las tablas en la base de datos

2.4.8 Estándares de los procedimientos almacenados:

El estándar para crear un procedimiento almacenado es el siguiente: PRO_VR_<<NOMBRE DEL PROCEDIMIENTO>>, el nombre del procedimiento debe ser escrito en letras mayúsculas y con palabras completas. Se tiene un ecabezado en todos los procedimientos donde se anotan las modificaciones que se realicen:

```
ALTER PROCEDURE
[dbo].[PRO_VR_CONSULTAR_VERIFICACIONES_FILTRADAS_POR_SUPERVISOR]
(
@i_USUARIO varchar(100)
)


AS
/*------------------------------------------------------------------------
* Nombre ..... : PRO_VR_CONSULTAR_VERIFICACIONES_FILTRADAS_POR_SUPERVISOR
* Proyecto ..... : PAGUEYA_CG
* Fecha creación : 2009/02/19
* Autor ..... : Julio Núñez
* Lugar ..... : Pague YA
* Descripción .. : Obtiene los datos que se presentarán en la pantalla de verificacion
* Objetivo ......: DATOS PARA REPORTES

* Modificaciones
Se aumento la variable de usuario para que presente la lista de trabajo
por supervisor
*/
BEGIN

END
```

Estándares para claves foráneas: Para las claves foráneas de las tablas el estándar de nombre es FK_<<TABLA1>>_<<TABLA2>> como se puede observar en el siguiente ejemplo: ALTER TABLE [dbo].[VR_CIUDAD] WITH CHECK ADD CONSTRAINT [FK_VR_CIUDAD_VR_CANTON] FOREIGN KEY([CODIGO_CANTON]) REFERENCES [dbo].[VR_CANTON] ([CODIGO_CANTON])

3 CAPITULO 3 Desarrollo del Sistema

Una vez que nos aseguramos que el diseño de alto nivel está correctamente realizado, se procede a la implementación del sistema.

Se realiza una revisión de todos los estándares de la etapa de diseño antes de comenzar con la implementación, esto se lo hace con el objetivo de que se los pueda re usar en todos los módulos del sistema.

Luego se procede a realizar una estrategia de implementación, esta estrategia incluye revisión, implementación y pruebas.

En la estrategia de revisión, se revisan todos los componentes, desde el nivel más bajo hasta el más alto, una vez que nos aseguramos que los componentes atómicos del sistema están acordes con las especificaciones externas, se procede a revisar el siguiente nivel. Así se pueden identificar de una manera rápida los problemas que presenten los objetos de los niveles más bajos antes de realizar la integración de todos los módulos. Si los módulos del nivel más bajo son fáciles de re usar, la implementación de un nivel más alto se volverá mucho más fácil.

La estrategia de revisión que se tiene para el sistema de verificación de información es, basándose en el plan de pruebas de integración, probar independientemente cada módulo, primero aquellos que no tengan relación con otros módulos y después aquellos que dependan de los módulos anteriores.

La estrategia de implementación nos permite que los programas realizados sean más fáciles de re usar cuando realizamos un nuevo proyecto. Por ejemplo, utilizar el mismo estándar de comentarios para cada módulo del programa, nos puede proveer toda la información importante para que permita a los usuarios potenciales conocer rápidamente que hace el programa.

La estrategia de implementación para el desarrollo del sistema de verificación de información es utilizar el mismo formato de comentarios para todas las funciones del sistema así como la creación de clases genéricas como la conexión a la base de datos, etc.

La estrategia de pruebas es una de las más importantes ya que con esta se pude desarrollar el sistema de una manera ordenada, y consistente, de acuerdo a los planes de prueba que se crearon en las etapas anteriores.

Básicamente la estrategia de pruebas que se va a utilizar para el desarrollo del sistema es utilizar los planes de prueba e integración realizados en las etapas de diseño y de requerimientos, esto nos da la garantía de que el sistema está desarrollado de acuerdo a los requerimientos de los clientes y que cumple con estándares de diseño y de calidad.

Una vez que se tienen las estrategias de implementación definidas, se procede a la implementación propiamente dicha del sistema, para esto, se ha divido al sistema en varios módulos, algunos módulos tienen dependencias de otros, es decir, primero se

deben construir uno o más módulos para poder ser implementado, un ejemplo de esto es el módulo de ciudades, el cual necesita que se implemente primero el módulo de cantones.

Al sistema de verificación de información se lo ha dividido en diferentes módulos con el objetivo de facilitar la construcción del mismo, primero se realizará las administraciones, en total son 6 administraciones de las cuales 3 son independientes y tres necesitan que una administración haya sido implementada previamente para poder ser implementada, le sigue a las administraciones los módulos de generación, se generan listas de trabajo, fichas de verificación y reportes. Los módulos de generación necesitan que todas las administraciones hayan sido implementadas previamente ya que estos utilizan la información que se ingresa a través de las administraciones. Los nueve módulos forman al sistema de verificación, en cada implementación de un módulo se realizarán pruebas unitarias para que el módulo esté listo para la integración.

La primera administración en ser implementada será la administración de empresas:

3.1 Administración de empresas

Ya que no es una sino varias las empresas que necesitan realizar la gestión de verificación de información, se pensó en crear este módulo, básicamente este módulo sirve para almacenar la información de la empresa que solicita se realice la gestión de verificación, el nombre de la empresa se verá reflejado en la ficha de verificación, el módulo consta de dos partes, la primera es la implementación de la administración propiamente dicha, es decir, ingreso, actualización y eliminación de datos de empresas y la segunda parte que es la construcción de la interface de usuario.

3.1.1 Ingresar una nueva empresa

Es la primera funcionalidad en ser implementada para este módulo, para la implementación del ingreso de una nueva empresa, se utilizó la arquitectura en tres capas, por lo que la implementación se la tiene en tres partes diferentes, la primera es la que pertenece a la parte de la capa de datos (DC), la segunda es la parte de la capa del negocio (BC) y la tercera parte es la capa de interface de usuario (GUI), a continuación se presenta la implementación de la funcionalidad ingresar una nueva empresa:

CAPA DE DATOS DC:

Siguiendo los estándares de diseño, cada método debe contener un encabezado el cual explique en breves palabras lo que hace dicho método, este es el encabezado del método graba empresas usuario:

```
/// <summary>
 /// método de la capa de datos que se encarga de
 /// grabar las empresas por usuario
 /// </summary>
 /// <param name="strModo">modo</param>
```

```
/// <param name="strCodigoFamilia">codigo de la empresa</param>
/// <param name="strUsuario">codigo usuario</param>
/// <param name="strUsuarioAsignacion">usuario asignado</param>
```

Después del encabezado que nos indica que hace el método viene la implementación del primer método que es GrabaEmpresasUsuario, este método almacena la información de las empresas que le han sido asignadas a un usuario para que pueda visualizar la información de estas empresas en los reportes:

```
public void GrabaEmpresasUsuario(string strModo, string strCodigoEmpresa, string strUsuario, string strUsuarioAsignacion)
{
    AccesoDatos objAccesoDatos = new AccesoDatos();
    try
    {
        SqlParameter[] parms = new SqlParameter[4];
        parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
        parms[0].Value = strModo;
        parms[1] = new SqlParameter("@i_CODIGO_EMPRESA", SqlDbType.Int);
        parms[1].Value = strCodigoEmpresa;
        parms[2] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 8);
        parms[2].Value = strUsuario;
        parms[3] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
        parms[3].Value = strUsuarioAsignacion;
        objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_EMPRESA_USUARIO", parms);
    }
    catch (Exception ex)
    {
        throw (ex);
    }
    finally
    {
        if (objAccesoDatos != null)
            objAccesoDatos = null;
    }
}
```

El segundo método en ser implementado es el método GrabaEmpresa, este es el método que modifica la información de las empresas que solicitan se realice la gestión de verificación, este método solo puede ser utilizado por el administrador del sistema:

```
/// <summary>
/// Método de la capa de datos que graba los registros de familias
/// </summary>
/// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
/// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
/// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
public void GrabaEmpresa(string strModo, int intCodigoEmpresa, string strNombreEmpresa)
{
   AccesoDatos objAccesoDatos = new AccesoDatos();
   try
   {
      SqlParameter[] parms = new SqlParameter[3];
      parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
      parms[0].Value = strModo;
      parms[1] = new SqlParameter("@i_CODIGO_EMPRESA", SqlDbType.Int);
      parms[1].Value = intCodigoEmpresa;
      parms[2] = new SqlParameter("@i_NOMBRE_EMPRESA", SqlDbType.VarChar, 50);
      parms[2].Value = strNombreEmpresa;
      objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_EMPRESA", parms);
   }
   catch (Exception ex)
   {
      throw (ex);
   }
   finally
   {
      if (objAccesoDatos != null)
         objAccesoDatos = null;
   }
}
```

CAPA DEL NEGOCIO BC:

Esta capa es en donde se llaman a todos los métodos de la capa de datos, al igual que la capa anterior, la capa del negocio fue implementada utilizando los estándares de diseño:

```
/// <summary>
/// método de la capa de lógica que se encarga de
/// grabar las empresas por usuario
/// </summary>
/// <param name="strModo">modo</param>
/// <param name="strCodigoFamilia">codigo de la empresa</param>
/// <param name="strUsuario">codigo usuario</param>
/// <param name="strUsuarioAsignacion">usuario asignado</param>
public void GrabaEmpresaUsuario(string strModo, string strCodigoEmpresa, string strUsuario, string strUsuarioAsignacion)
{
   try
   {
      VR_EmpresasDC objEmpresasDC = new VR_EmpresasDC();
      objEmpresasDC.GrabaEmpresasUsuario(strModo, strCodigoEmpresa, strUsuario, strUsuarioAsignacion);
   }
   catch (Exception ex)
   {
      throw (ex);
   }
}
```

Como podemos ver, la construcción de esta capa es sencilla ya que lo único que se necesita es crear un objeto de la capa de datos e invocar a sus métodos otorgándole los parámetros que sean necesarios, estos parámetros se los envía desde la capa de interface de usuario

```
/// <summary>
/// Método de la capa de logica que graba los registros de familias
/// </summary>
/// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
/// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
/// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
public void GrabaEmpresa(string strModo, int intCodigoEmpresa, string strNombreEmpresa)
```

```
{
    try
    {
        VR_EmpresasDC objEmpresasDC = new VR_EmpresasDC();
        objEmpresasDC.GrabaEmpresa(strModo,intCodigoEmpresa, strNombreEmpresa);
    }
    catch (Exception ex)
    {
        throw (ex);
        }
    }
```

CAPA DE INTERFACE DE USUARIO GUI:

```
protected void wbtGrabar_Click(object sender, EventArgs e)
{
    try
    {
        if (txtNombre.Text.Length == 0)
        {
            MessageBox("Ingrese el nombre", txtNombre);
            return;
        }
        if (txtCodigo.Text.Length == 0)
        {
            MessageBox("Ingrese el código", txtCodigo);
            return;
        }
        string strModo = "";
        if (blnNuevo)
        {
            strModo = "I";
        }
        else
        {
```

```
            strModo = "M";
        }
        if (blnNuevo)
        {
            for (int i = 0; i < dttEmpresa.Rows.Count; i++)
            {
                if (txtCodigo.Text == dttEmpresa.Rows[i][0].ToString())
                {
                    MessageBox("Ya existe un otro registro con este código", txtNombre);
                    return;
                }
            }
        }
        VR_EmpresasBC objDatos = new VR_EmpresasBC();
        objDatos.GrabaEmpresa(strModo, Convert.ToInt32(txtCodigo.Text), txtNombre.Text.ToUpper());
        MessageBox("El Registro se grabó exitosamente");
        ConsultaEmpresas();
        limpiarDatos();
    }
    catch (System.Exception ex)
    {
        MessageError(ex.Message);
    }
}
```

Como podemos observar, lo que se hace en esta capa es únicamente pasar los datos que el usuario ingresa por pantalla a la capa del negocio la cual a su vez, envía estos parámetros a la capa de datos.

3.1.2 Actualizar datos de las empresas

La actualización de empresas utiliza los mismos métodos que la funcionalidad anterior, la diferencia está en el parámetro modo, si modo tiene el valor de “I”, se está ingresando una nueva empresa, si modo tiene el valor de “M”, estamos modificando los datos de la empresa, a continuación los métodos para actualizar la información de una empresa:

CAPA DE DATOS DC:

```
/// <summary>
/// método de la capa de datos que se encarga de
/// grabar las empresas por usuario
/// </summary>
/// <param name="strModo">modo</param>
/// <param name="strCodigoFamilia">codigo de la empresa</param>
/// <param name="strUsuario">codigo usuario</param>
/// <param name="strUsuarioAsignacion">usuario asignado</param>
```

Después del encabezado que nos indica que hace el método viene la implementación del primer método que es GrabaEmpresasUsuario, este método almacena la información de las empresas que le han sido asignadas a un usuario para que pueda visualizar la información de estas empresas en los reportes:

```
public void GrabaEmpresasUsuario(string strModo, string strCodigoEmpresa, string strUsuario, string strUsuarioAsignacion)
{
    AccesoDatos objAccesoDatos = new AccesoDatos();
    try
    {
        SqlParameter[] parms = new SqlParameter[4];
        parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
        parms[0].Value = strModo;
        parms[1] = new SqlParameter("@i_CODIGO_EMPRESA", SqlDbType.Int);
        parms[1].Value = strCodigoEmpresa;
        parms[2] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 8);
        parms[2].Value = strUsuario;
        parms[3] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
        parms[3].Value = strUsuarioAsignacion;
        objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_EMPRESA_USUARIO", parms);
    }
    catch (Exception ex)
    {
        throw (ex);
    }
    finally
```

```
        {
            if (objAccesoDatos != null)
                objAccesoDatos = null;
        }
    }
```

El segundo método en ser implementado es el método GrabaEmpresa, este es el método que almacena la información de las empresas que solicitan se realice la gestión de verificación, este método solo puede ser utilizado por el administrador del sistema:

```
    /// <summary>
    /// Método de la capa de datos que graba los registros de familias
    /// </summary>
    /// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
    /// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
    /// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
    public void GrabaEmpresa(string strModo, int intCodigoEmpresa, string strNombreEmpresa)
    {
        AccesoDatos objAccesoDatos = new AccesoDatos();
        try
        {
            SqlParameter[] parms = new SqlParameter[3];
            parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
            parms[0].Value = strModo;
            parms[1] = new SqlParameter("@i_CODIGO_EMPRESA", SqlDbType.Int);
            parms[1].Value = intCodigoEmpresa;
            parms[2] = new SqlParameter("@i_NOMBRE_EMPRESA", SqlDbType.VarChar, 50);
            parms[2].Value = strNombreEmpresa;
            objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_EMPRESA", parms);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
        finally
        {
```

```
        if (objAccesoDatos != null)
            objAccesoDatos = null;
    }
}
```

Como podemos observar, esta es la capa en donde se realizan todas las transacciones con la base de datos, es en donde se llama a los procedimientos almacenados que ejecutarán las órdenes que el usuario de a la funcionalidad desde la interface del usuario.

CAPA DEL NEGOCIO BC:

Esta capa es en donde se llaman a todos los métodos de la capa de datos, al igual que la capa anterior, la capa del negocio fue implementada utilizando los estándares de diseño:

```
/// <summary>
/// método de la capa de lógica que se encarga de
/// grabar las empresas por usuario
/// </summary>
/// <param name="strModo">modo</param>
/// <param name="strCodigoFamilia">codigo de la empresa</param>
/// <param name="strUsuario">codigo usuario</param>
/// <param name="strUsuarioAsignacion">usuario asignado</param>
public void GrabaEmpresaUsuario(string strModo, string strCodigoEmpresa, string strUsuario, string strUsuarioAsignacion)
{
    try
    {
        VR_EmpresasDC objEmpresasDC = new VR_EmpresasDC();
        objEmpresasDC.GrabaEmpresasUsuario(strModo, strCodigoEmpresa, strUsuario, strUsuarioAsignacion);
    }
    catch (Exception ex)
    {
        throw (ex);
    }
}
```

Como podemos ver, la construcción de esta capa es sencilla ya que lo único que se necesita es crear un objeto de la capa de datos e invocar a sus métodos otorgándole los parámetros que sean necesarios, estos parámetros se los envía desde la capa de interface de usuario

```
/// <summary>
        /// Método de la capa de logica que graba los registros de familias
        /// </summary>
        /// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
        /// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
        /// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
        public void GrabaEmpresa(string strModo, int intCodigoEmpresa, string strNombreEmpresa)
        {
            try
            {
                VR_EmpresasDC objEmpresasDC = new VR_EmpresasDC();
                objEmpresasDC.GrabaEmpresa(strModo,intCodigoEmpresa, strNombreEmpresa);
            }
            catch (Exception ex)
            {
                throw (ex);
                  }
            }
```

CAPA DE INTERFACE DE USUARIO GUI:

La implementación con HTML se encuentra más adelante, a continuación se presenta la implementación en código C# para el ingreso de empresas:

```
    protected void wbtGrabar_Click(object sender, EventArgs e)
    {
        try
        {
            if (txtNombre.Text.Length == 0)
            {
                MessageBox("Ingrese el nombre", txtNombre);
                return;
            }
            if (txtCodigo.Text.Length == 0)
            {
                MessageBox("Ingrese el código", txtCodigo);
```

```
            return;
        }
        string strModo = "";
        if (blnNuevo)
        {
            strModo = "I";
        }
        else
        {
            strModo = "M";
        }
        if (blnNuevo)
        {
            for (int i = 0; i < dttEmpresa.Rows.Count; i++)
            {
                if (txtCodigo.Text == dttEmpresa.Rows[i][0].ToString())
                {
                    MessageBox("Ya existe un otro registro con este código", txtNombre);
                    return;
                }
            }
        }
        VR_EmpresasBC objDatos = new VR_EmpresasBC();
        objDatos.GrabaEmpresa(strModo, Convert.ToInt32(txtCodigo.Text), txtNombre.Text.ToUpper());
        MessageBox("El Registro se grabó exitosamente");
        ConsultaEmpresas();
        limpiarDatos();
    }
    catch (System.Exception ex)
    {
        MessageError(ex.Message);
    }
}
```

Como podemos observar, lo que se hace en esta capa es únicamente pasar los datos que el usuario ingresa por pantalla a la capa del negocio la cual a su vez, envía estos parámetros a la capa de datos.

3.1.3 Eliminar empresas

Esta es la tercera funcionalidad de este módulo, al igual que las anteriores funcionalidades, esta también se ha dividido en tres capas las cuales se presentan a continuación:

CAPA DE DATOS DC:

```
/// <summary>
        /// Método de la capa de datos que se encarga de eliminar los datos de
        /// las empresas filtradas pro usuario
        /// </summary>
        /// <param name="strUsuario">codigo de usuario</param>
        /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
        public void EliminaEmpresasUsuario(string strUsuario, string strUsuarioAsignacion)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[2];
                parms[0] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 8);
                parms[0].Value = strUsuario;
                parms[1] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
                parms[1].Value = strUsuarioAsignacion;
                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_BORRA_EMPRESAS_USUARIOS", parms); //CREAR ESTE PROCEDURE PARA CIUDADES
            }
            catch (Exception ex)
            {
                throw (ex);
            }
            finally
            {
                if (objAccesoDatos != null)
                    objAccesoDatos = null;
```

```
        }
    }
    /// <summary>
    /// Método de la capa de datos que se encarga de eliminar los datos de
    /// las empresas
    /// </summary>
    /// <param name="strFamilia">codigo de usuario</param>
    /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
    public void EliminaEmpresas(int intEmpresa, string strUsuarioAsignacion)
    {
        AccesoDatos objAccesoDatos = new AccesoDatos();
        try
        {
            SqlParameter[] parms = new SqlParameter[2];
            parms[0] = new SqlParameter("@i_CODIGO_EMPRESA", SqlDbType.Int);
            parms[0].Value = intEmpresa;
            parms[1] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
            parms[1].Value = strUsuarioAsignacion;
            objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure, "PRO_VR_BORRA_EMPRESA",
parms);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
        finally
        {
            if (objAccesoDatos != null)
                objAccesoDatos = null;
        }
    }
```

Cada una de las funcionalidades de la capa de datos tiene un procedimiento almacenado que se ejecuta cada vez que se necesita una de las funcionalidades.

CAPA DEL NEGOCIO BC

```
/// <summary>
/// Método de la capa de logica que se encarga de eliminar los datos de
/// las empresas filtradas pro usuario
/// </summary>
/// <param name="strFamilia">codigo de usuario</param>
/// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
public void EliminaEmpresasUsuario(string strUsuario, string strUsuarioAsignacion)
{
    try
    {
        VR_EmpresasDC objEmpresasDC = new VR_EmpresasDC();
        objEmpresasDC.EliminaEmpresasUsuario(strUsuario, strUsuarioAsignacion);
    }
    catch (Exception ex)
    {
        throw (ex);
    }
}


/// <summary>
/// Método de la capa de logica que se encarga de llamar a la capa de datos para eliminar
/// las empresas
/// </summary>
/// <param name="strUsuario">codigo de usuario</param>
/// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
public void EliminaEmpresas(int intEmpresa, string strUsuarioAsignacion)
{
    try
    {
        VR_EmpresasDC objEmpresasDC = new VR_EmpresasDC();
        objEmpresasDC.EliminaEmpresas(intEmpresa, strUsuarioAsignacion);
    }
    catch (Exception ex)
```

```
        {
            throw (ex);
        }
    }
```

Al igual que las funcionalidades anteriores, esta también únicamente llama a los métodos de la capa de datos en esta capa.

CAPA DE INTERFACE DE USUARIO GUI

```
protected void btnEliminar_Click(object sender, EventArgs e)
{
    try
    {
        VR_EmpresasBC objDatos = new VR_EmpresasBC();
        if (txtCodigo.Text.Trim() == "")
        {
            MessageBox("Seleccione un registro para eliminar", txtNombre);
            return;
        }
        objDatos.EliminaEmpresas(Convert.ToInt32(txtCodigo.Text), strUsuarioAPP[0]);
        MessageBox("El Registro se eliminó exitosamente");
        ConsultaEmpresas();
        limpiarDatos();
        blnNuevo = true;
    }
    catch (System.Exception ex)
    {
        MessageError(ex.Message);
    }
}
```

3.1.4 Interfaces de usuario

En esta sección se presenta parte del código HTML de la administración de empresas, el código fuente de la administración se encuentra en el anexo 3:

```
<table id="Table1" align="center" border="1" cellpadding="1" style="z-index: 102;
        left: 7px; width: 469px; position: absolute; top: 42px; height: 90px">
        <tr class="fconsult">
```

```
            <td style="width: 362px; height: 32px">
                <strong>Nombre Empresa: </strong></td>
            <td colspan="2" style="height: 32px">
                <asp:TextBox ID="txtNombre" runat="server" CssClass="UpperText" MaxLength="150"
Width="301px"></asp:TextBox></td>
        </tr>
        <tr class="fconsult">
            <td style="width: 362px">
                <strong>Código Empresa:</strong></td>
            <td colspan="2">
                <asp:TextBox ID="txtCodigo" runat="server" CssClass="UpperText" MaxLength="3"
Width="54px"></asp:TextBox><asp:RegularExpressionValidator
                    ID="revNivel1" runat="server" ControlToValidate="txtCodigo" ErrorMessage="Debe ingresar solo números"
                    ValidationExpression="\d{1,3}"></asp:RegularExpressionValidator></td>
        </tr>
    </table>
```

El resultado de la implementación de la administración de empresas es la siguente pantalla:

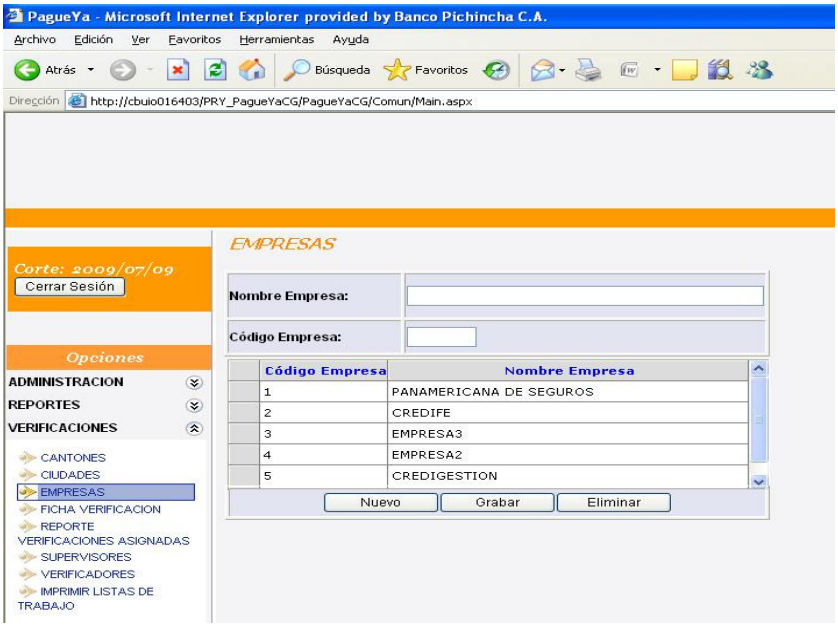


Figura 3.1: Presentación de la interface de usuario de administración de empresas

3.1.5 Pruebas unitarias

Una vez que se ha terminado con toda la implementación del módulo de empresas, se procede a realizar las pruebas unitarias del mismo:

INGRESANDO UNA NUEVA EMPRESA:

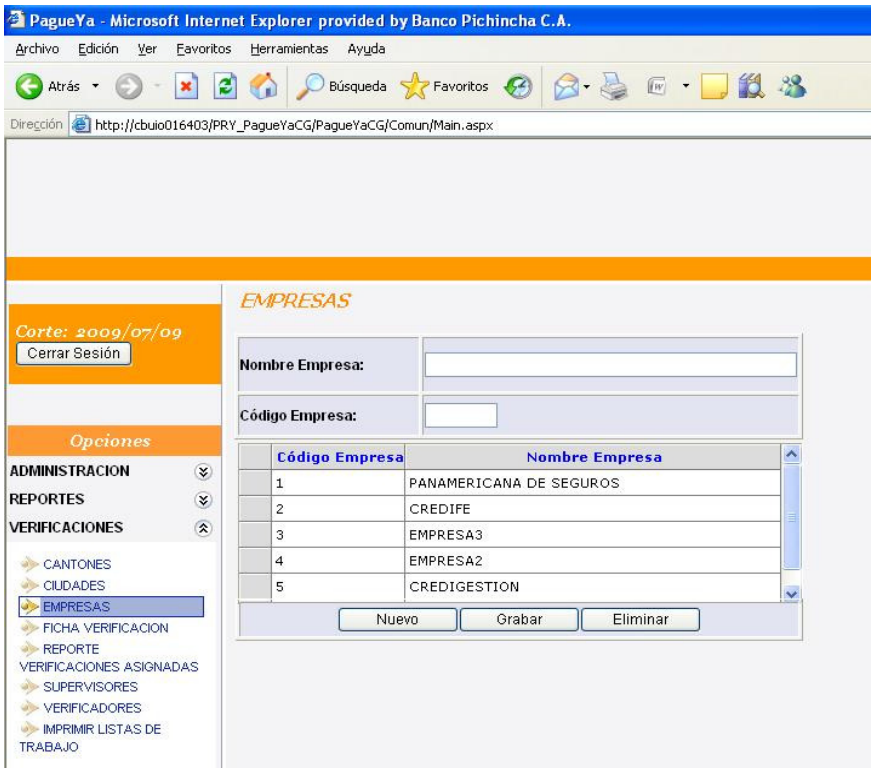


Figura 3.2: Primera pantalla del ingreso de una nueva empresa

Como podemos observar, ya existen empresas ingresadas en la base de datos, estas empresas son presentadas en una lista, para ingresar una nueva empresa se debe llenar la información del nombre de la empresa y el código que esta tendrá en la base de datos:

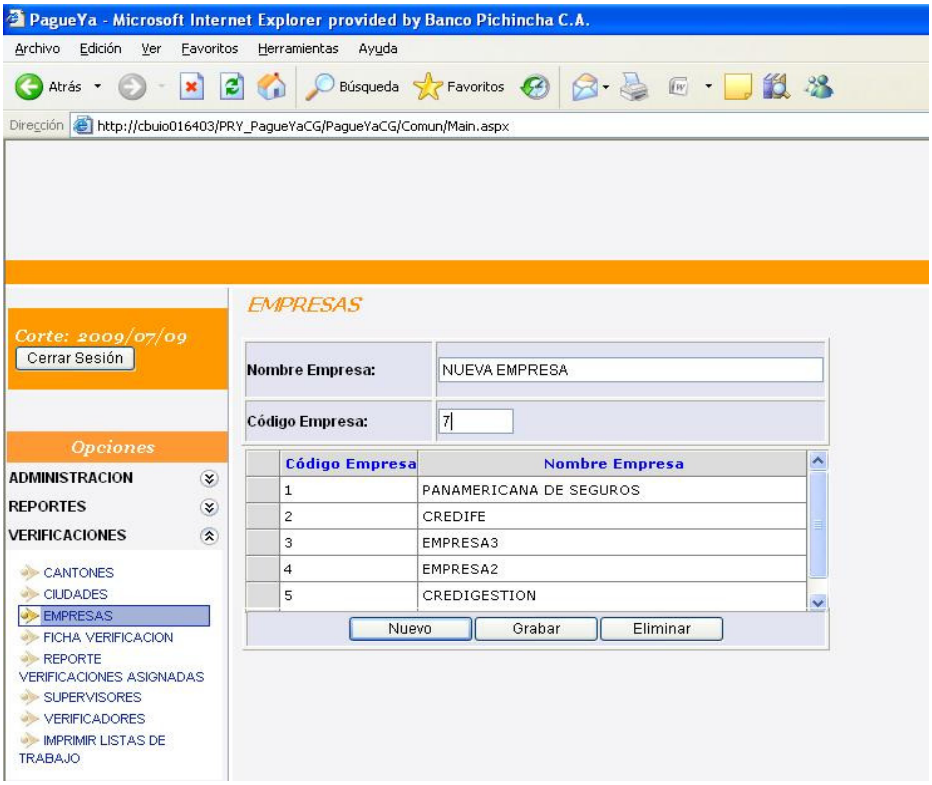


Figura 3.3: Ingresando la información de la nueva empresa

Una vez que la información ha sido ingresada por el usuario, se presiona el botón de grabar y el sistema despliega un mensaje avisando al usuario que el registro ha sido grabado exitosamente:

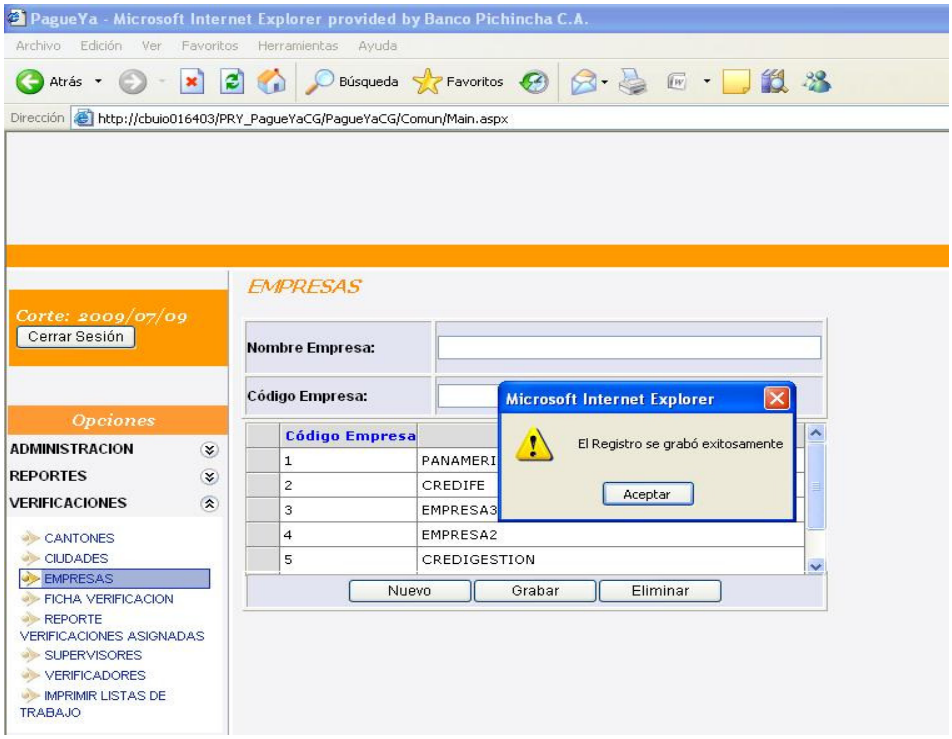


Figura 3.4: Mensaje de alerta al usuario del éxito al ingresar una nueva empresa

ACTUALIZANDO LOS DATOS DE UNA EMPRESA:

Esta es otra de las funcionalidades del módulo, para actualizar los datos de una empresa lo primero que debemos hacer es seleccionar una empresa de la lista, en este caso, se seleccionará la empresa que se ingresó anteriormente:

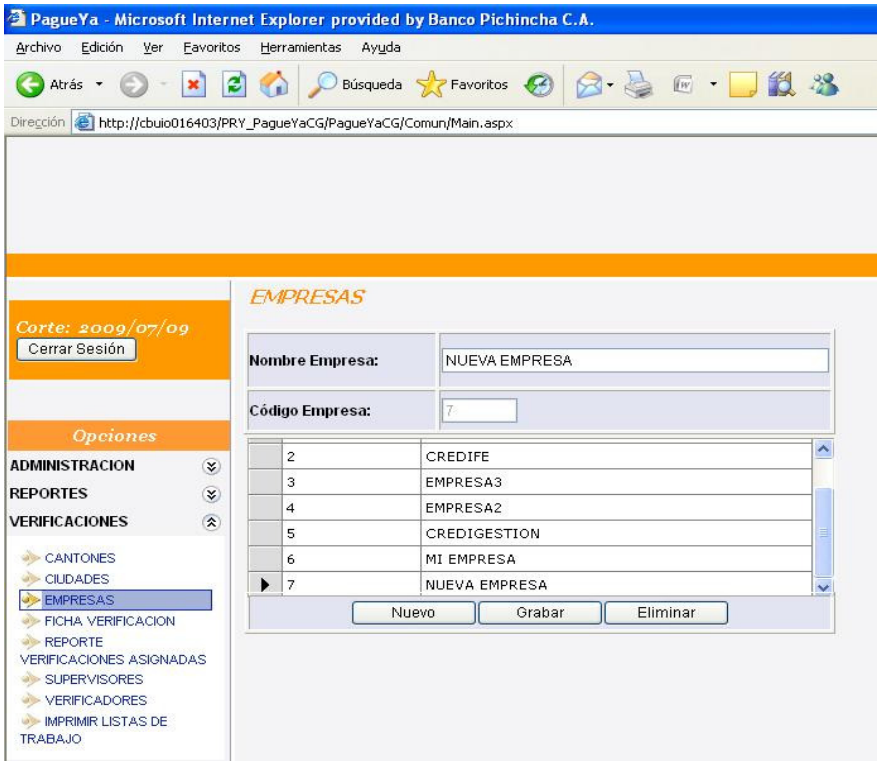


Figura 3.5: Seleccionando una empresa de la lista de empresas

Luego de que la empresa ha sido seleccionada, se procede a cambial el nombre de la misma, no se permite cambiar el código de la empresa ya que este código es para identificación del registro en la base de datos:

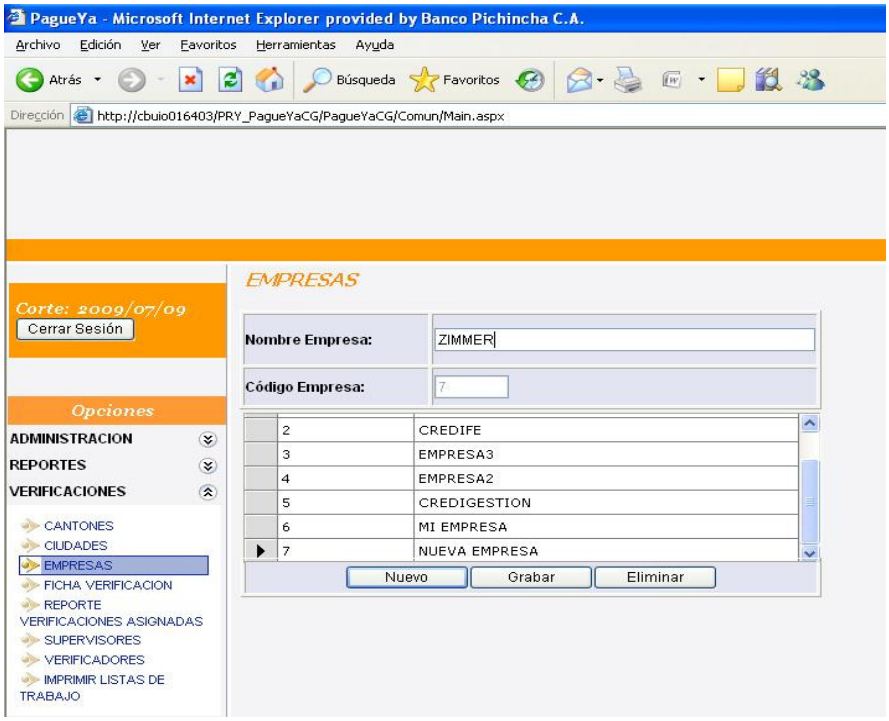


Figura 3.6: Cambiando de nombre a la empresa

En este caso, se cambió el nombre de la empresa original por el de “ZIMMER”, una vez que se ha cambiado el nombre de la empresa, se presiona el botón de grabar los datos y el sistema alerta al usuario que su registro se ha grabado exitosamente:

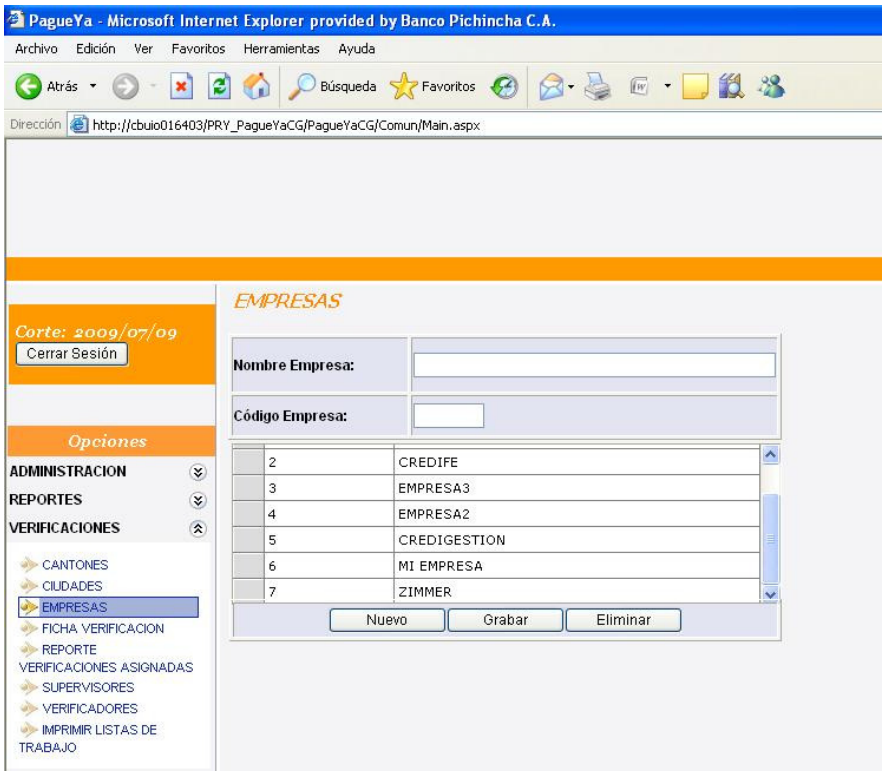


Figura 3.7: El cambio ha sido realizado con éxito.

ELIMINANDO UNA EMPRESA:

La última de las funcionalidades de este módulo es la de eliminar la información de una empresa, al igual que en la actualización de datos, si queremos eliminar una empresa, lo primero que debemos hacer es seleccionar una empresa de la lista y se presiona el botón de eliminar el sistema pregunta al usuario si en verdad desea eliminar el registro seleccionado, para este caso de prueba, utilizamos el registro con el código número 7:

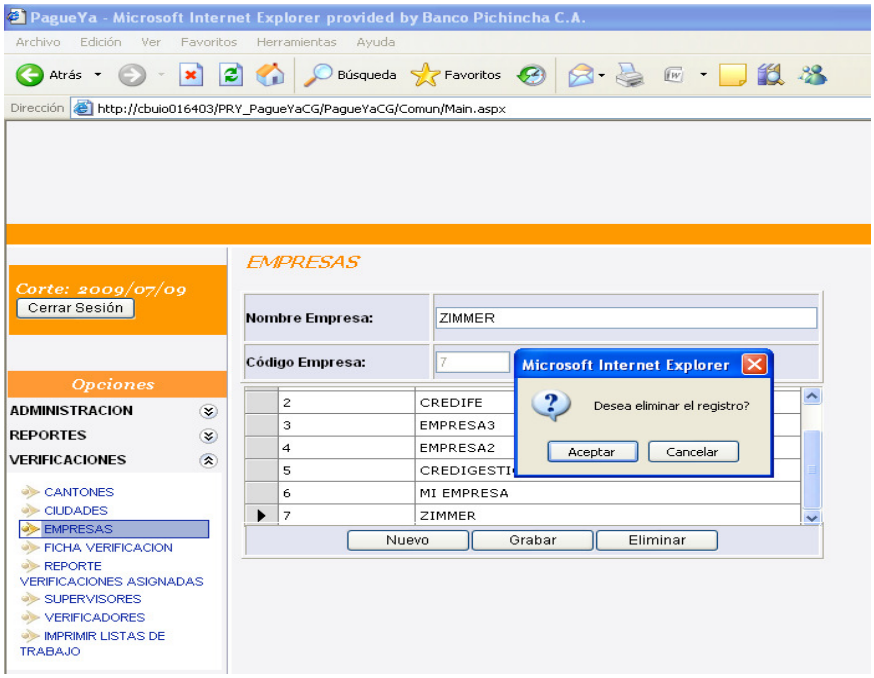


Figura 3.8: Solicitud de confirmación de eliminación de un registro

Una vez que el usuario ha aceptado eliminar el registro, el sistema lo borra de la base de datos y presenta al usuario la lista de las empresas que se encuentran almacenadas en la base de datos:

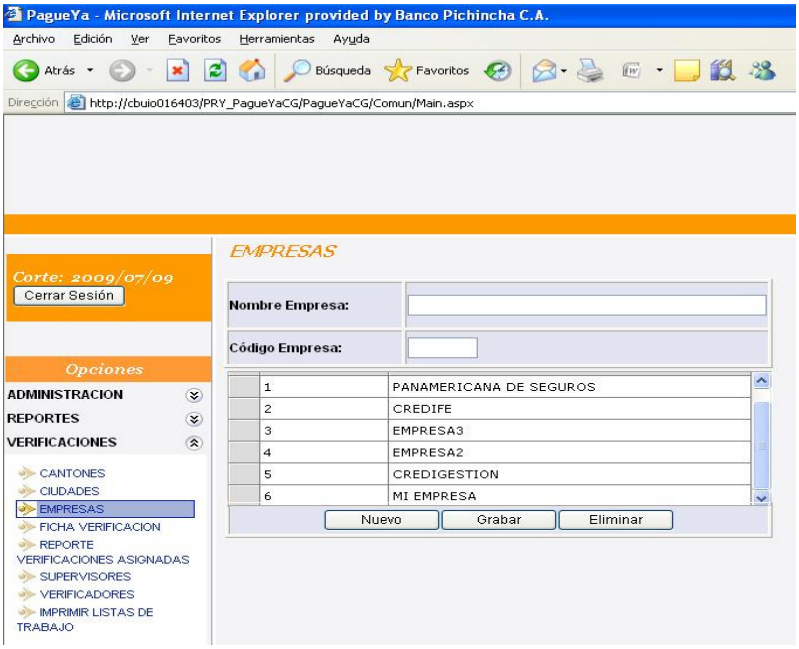


Figura 3.9: Registro eliminado del sistema

Como podemos observar, el registro ha sido eliminado exitosamente de la base de datos.

Estas son las pruebas unitarias que se realizaron para el módulo de administración de empresas, más pruebas para este módulo se las realizará en las pruebas de integración del sistema.

Después de la administración de empresas se implementó la administración de cantones:

3.2 Administración de cantones

Es importante para la empresa que va a hacer la gestión de verificación tener una zonificación para realizar la gestión de cobranza de una manera más efectiva, es por esto que se decidió crear un módulo para manejar cantones,

3.2.1 Ingresar un nuevo cantón

Al igual que en la administración de empresas, para los cantones también se dividió la implementación del módulo en las tres capas, el ingreso de cantones permite al usuario ingresar la información de un nuevo cantón, se ingresa únicamente código nombre del cantón, a continuación el código del ingreso de cantones separado por capas:

CAPA DE DATOS DC:

```
/// <summary>

/// método de la capa de datos que se encarga de

/// grabar los cantones por usuario

/// </summary>

/// <param name="strModo">modo</param>

/// <param name="strCodigoFamilia">codigo del canton</param>
```

```
        /// <param name="strUsuario">codigo usuario</param>
        /// <param name="strUsuarioAsignacion">usuario asignado</param>
        public void GrabaCantonesUsuario(string strModo, string strCodigoCanton, string strUsuario, string strUsuarioAsignacion)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[4];
                parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
                parms[0].Value = strModo;
                parms[1] = new SqlParameter("@i_CODIGO_CANTON", SqlDbType.Int);
                parms[1].Value = strCodigoCanton;
                parms[2] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 8);
                parms[2].Value = strUsuario;
                parms[3] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
                parms[3].Value = strUsuarioAsignacion;

                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_CIUDAD_USUARIO", parms); //CREAR ESTE PROCEDURE PARA CIUDADES
            }
            catch (Exception ex)
            {
                throw (ex);
            }
            finally
            {
                if (objAccesoDatos != null)
                    objAccesoDatos = null;
            }
        }
        /// <summary>
        /// Método de la capa de datos que graba los registros de cantones
        /// </summary>
        /// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
        /// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
```

```
        /// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
        public void GrabaCanton(string strModo, int intCodigoCanton, string strNombreCanton)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[3];
                parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
                parms[0].Value = strModo;
                parms[1] = new SqlParameter("@i_CODIGO_CANTON", SqlDbType.Int);
                parms[1].Value = intCodigoCanton;
                parms[2] = new SqlParameter("@i_NOMBRE_CANTON", SqlDbType.VarChar, 50);
                parms[2].Value = strNombreCanton;
                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_CANTON", parms);             }
            catch (Exception ex)
            {
                throw (ex);
            }
            finally
            {
                if (objAccesoDatos != null)
                    objAccesoDatos = null;
            }
        }
```

Los métodos para el ingreso de un nuevo cantón fueron implementados con los estándares de diseño definidos en la etapa de diseño del sistema, como se puede ver, en el método de ingreso de todas las administraciones, se utiliza la estrategia del re uso ya que se utiliza el mismo método tanto para ingresar como para actualizar los datos de un cantón. Una vez implementada la capa de datos, se procede a la implementación de la capa del negocio:

CAPA DEL NEGOCIO BC:

En esta capa lo que se hace es llamar a los métodos de la capa de datos y se les pasa los parámetros necesarios para que la capa de datos pueda trabajar:

```
/// <summary>
/// método de la capa de logica que se encarga de
/// grabar los cantones por usuario
/// </summary>
/// <param name="strModo">modo</param>
/// <param name="strCodigoFamilia">codigo de la familia</param>
/// <param name="strUsuario">codigo usuario</param>
/// <param name="strUsuarioAsignacion">usuario asignado</param>
public void GrabaCantonUsuario(string strModo, string strCodigoCanton, string strUsuario, string strUsuarioAsignacion)
{
   try
   {
      VR_CantonesDC objCantonesDC = new VR_CantonesDC();
      objCantonesDC.GrabaCantonesUsuario(strModo, strCodigoCanton, strUsuario, strUsuarioAsignacion);
   }
   catch (Exception ex)
   {
      throw (ex);
   }
}

/// <summary>
/// Método de la capa de logica que graba los registros de cantones
/// </summary>
/// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
/// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
/// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
public void GrabaCanton(string strModo, int intCodigoCanton, string strNombreCanton)
{
   try
```

```
        {
            VR_CantonesDC objCantonesDC = new VR_CantonesDC();
            objCantonesDC.GrabaCanton(strModo,intCodigoCanton, strNombreCanton);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
    }
```

CAPA DE INTERFACE DE USUARIO GUI

Una vez que se han implementado las dos capas de la arquitectura de programación, se procede a implementar el ingreso de cantones para la capa de la interface de usuario:

```
protected void wbtGrabar_Click(object sender, EventArgs e)
{
    try
    {
        if (txtNombre.Text.Length == 0)
        {
            MessageBox("Ingrese el nombre", txtNombre);
            return;
        }
        if (txtCodigo.Text.Length == 0)
        {
            MessageBox("Ingrese el código", txtCodigo);
            return;
        }
        string strModo = "";
        if (blnNuevo)
        {
            strModo = "I";
        }
        else
        {
            strModo = "M";
```

```
        }


        if (blnNuevo)
        {
            for (int i = 0; i < dttCanton.Rows.Count; i++)
            {
                if (txtCodigo.Text == dttCanton.Rows[i][0].ToString())
                {
                    MessageBox("Ya existe un otro registro con este código", txtNombre);
                    return;
                }
            }
        }
        VR_CantonesBC objDatos = new VR_CantonesBC();
        objDatos.GrabaCanton(strModo,Convert.ToInt32(txtCodigo.Text),txtNombre.Text.ToUpper());
        MessageBox("El Registro se grabó exitosamente");
        ConsultaCantones();
        limpiarDatos();
    }
    catch (System.Exception ex)
    {
        MessageError(ex.Message);
    }
}
```

Como podemos observar, es en la capa de la interface de usuario en donde se da el valor al parámetro modo para que indique al procedimiento almacenado si debe ingresar o actualizar los datos de un cantón.

3.2.2 Actualizar datos de los cantones

La siguiente funcionalidad en ser implementada es la de actualización de datos de los cantones ingresados en la base de datos, se utiliza la estrategia del re uso y se utiliza el mismo código que se utiliza para el ingreso de datos de un cantón así como también se utilizaron los estándares de diseño en lo que se refiere a comentarios y nombre de variables y métodos:

CAPA DE DATOS DC:

```
/// <summary>
/// método de la capa de datos que se encarga de
/// grabar los cantones por usuario
/// </summary>
/// <param name="strModo">modo</param>
/// <param name="strCodigoFamilia">codigo del canton</param>
/// <param name="strUsuario">codigo usuario</param>
/// <param name="strUsuarioAsignacion">usuario asignado</param>
public void GrabaCantonesUsuario(string strModo, string strCodigoCanton, string strUsuario, string strUsuarioAsignacion)
{
   AccesoDatos objAccesoDatos = new AccesoDatos();
   try
   {
      SqlParameter[] parms = new SqlParameter[4];
      parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
      parms[0].Value = strModo;
      parms[1] = new SqlParameter("@i_CODIGO_CANTON", SqlDbType.Int);
      parms[1].Value = strCodigoCanton;
      parms[2] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 8);
      parms[2].Value = strUsuario;
      parms[3] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
      parms[3].Value = strUsuarioAsignacion;

      objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_CIUDAD_USUARIO", parms); //CREAR ESTE PROCEDURE PARA CIUDADES
      }
      catch (Exception ex)
      {
         throw (ex);
      }
      finally
      {
         if (objAccesoDatos != null)
```

```
                objAccesoDatos = null;
            }
        }
        /// <summary>
        /// Método de la capa de datos que graba los registros de cantones
        /// </summary>
        /// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
        /// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
        /// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
        public void GrabaCanton(string strModo, int intCodigoCanton, string strNombreCanton)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[3];
                parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
                parms[0].Value = strModo;
                parms[1] = new SqlParameter("@i_CODIGO_CANTON", SqlDbType.Int);
                parms[1].Value = intCodigoCanton;
                parms[2] = new SqlParameter("@i_NOMBRE_CANTON", SqlDbType.VarChar, 50);
                parms[2].Value = strNombreCanton;
                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_CANTON", parms);             }
            catch (Exception ex)
            {
                throw (ex);
            }
            finally
            {
                if (objAccesoDatos != null)
                    objAccesoDatos = null;
            }
        }
```

Como podemos observar, es el mismo método que para el ingreso de datos de un nuevo cantón.

CAPA DEL NEGOCIO BC

```
/// <summary>
/// método de la capa de logica que se encarga de
/// grabar los cantones por usuario
/// </summary>
/// <param name="strModo">modo</param>
/// <param name="strCodigoFamilia">codigo de la familia</param>
/// <param name="strUsuario">codigo usuario</param>
/// <param name="strUsuarioAsignacion">usuario asignado</param>
public void GrabaCantonUsuario(string strModo, string strCodigoCanton, string strUsuario, string strUsuarioAsignacion)
{
   try
   {
      VR_CantonesDC objCantonesDC = new VR_CantonesDC();
      objCantonesDC.GrabaCantonesUsuario(strModo, strCodigoCanton, strUsuario, strUsuarioAsignacion);
   }
   catch (Exception ex)
   {
      throw (ex);
   }
}
/// <summary>
/// Método de la capa de logica que graba los registros de cantones
/// </summary>
/// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
/// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
/// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
public void GrabaCanton(string strModo, int intCodigoCanton, string strNombreCanton)
{
   try
   {
      VR_CantonesDC objCantonesDC = new VR_CantonesDC();
      objCantonesDC.GrabaCanton(strModo,intCodigoCanton, strNombreCanton);
   }
```

```
        catch (Exception ex)
        {
            throw (ex);
        }
    }
```

Siguiendo la estrategia del re uso, se utilizó el mismo método para la actualización de datos

CAPA DE INTERFACE DE USUARIO GUI

```
protected void wbtGrabar_Click(object sender, EventArgs e)
{
    try
    {
        if (txtNombre.Text.Length == 0)
        {
            MessageBox("Ingrese el nombre", txtNombre);
            return;
        }
        if (txtCodigo.Text.Length == 0)
        {
            MessageBox("Ingrese el código", txtCodigo);
            return;
        }
        string strModo = "";
        if (blnNuevo)
        {
            strModo = "I";
        }
        else
        {
            strModo = "M";
        }

        if (blnNuevo)
        {
            for (int i = 0; i < dttCanton.Rows.Count; i++)
```

```
            {
                if (txtCodigo.Text == dttCanton.Rows[i][0].ToString())
                {
                    MessageBox("Ya existe un otro registro con este código", txtNombre);
                    return;
                }
            }
        }
        VR_CantonesBC objDatos = new VR_CantonesBC();
        objDatos.GrabaCanton(strModo,Convert.ToInt32(txtCodigo.Text),txtNombre.Text.ToUpper());
        MessageBox("El Registro se grabó exitosamente");
        ConsultaCantones();
        limpiarDatos();
    }
    catch (System.Exception ex)
    {
        MessageError(ex.Message);
    }
}
```

Ya que es un único botón que se utiliza tanto para insertar como para actualizar los datos de un cantón, se marca la diferencia en el parámetro modo que se envía a la capa del negocio.

3.2.3 Eliminar cantones

Esta es la última funcionalidad para este módulo, ésta permite al usuario eliminar la información de un cantón que se encuentre en la base de datos.

CAPA DE DATOS DC:

```
/// <summary>
/// Método de la capa de datos que se encarga de eliminar los datos de
/// los cantones filtrados por usuario
/// </summary>
/// <param name="strUsuario">codigo de usuario</param>
/// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
public void EliminaCantonesUsuario(string strUsuario, string strUsuarioAsignacion)
{
    AccesoDatos objAccesoDatos = new AccesoDatos();
```

```
            try
            {
                SqlParameter[] parms = new SqlParameter[2];
                parms[0] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 8);
                parms[0].Value = strUsuario;
                parms[1] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
                parms[1].Value = strUsuarioAsignacion;
                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_BORRA_CANTONES_USUARIOS", parms); //CREAR ESTE PROCEDURE PARA CIUDADES
            }
            catch (Exception ex)
            {
                throw (ex);
            }
            finally
            {
                if (objAccesoDatos != null)
                    objAccesoDatos = null;
            }
        }
        /// <summary>
        /// Método de la capa de datos que se encarga de eliminar los datos de
        /// los cantones
        /// </summary>
        /// <param name="strFamilia">codigo de usuario</param>
        /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
        public void EliminaCantones(int intCanton, string strUsuarioAsignacion)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[2];
                parms[0] = new SqlParameter("@i_CODIGO_CANTON", SqlDbType.Int);
                parms[0].Value = intCanton;
                parms[1] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
```

```
            parms[1].Value = strUsuarioAsignacion;
            objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure, "PRO_VR_BORRA_CANTON",
parms);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
        finally
        {
            if (objAccesoDatos != null)
                objAccesoDatos = null;
        }
    }
```

La lógica de implementación para el método de eliminación de cantones es la misma que para las empresas, la única diferencia con la administración anterior es que se debe añadir un control extra para cuando un cantón tenga asociadas una o varias ciudades, en este caso, no se permite la eliminación del cantón.

CAPA DEL NEGOCIO BC

```
    /// <summary>
    /// Método de la capa de logica que se encarga de eliminar los datos de
    /// los cantones filtrados pro usuario
    /// </summary>
    /// <param name="strFamilia">codigo de usuario</param>
    /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
    public void EliminaCantonesUsuario(string strUsuario, string strUsuarioAsignacion)
    {
        try
        {
            VR_CantonesDC objCantonesDC = new VR_CantonesDC();
            objCantonesDC.EliminaCantonesUsuario(strUsuario, strUsuarioAsignacion);
        }
        catch (Exception ex)
        {
            throw (ex);
```

```
        }
    }

    /// <summary>
    /// Método de la capa de logica que se encarga de llamar a la capa de datos para eliminar
    /// los asesores
    /// </summary>
    /// <param name="strUsuario">codigo de usuario</param>
    /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
    public void EliminaCantones(int intCanton, string strUsuarioAsignacion)
    {
        try
        {
            VR_CantonesDC objCantonesDC = new VR_CantonesDC();
            objCantonesDC.EliminaCantones(intCanton, strUsuarioAsignacion);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
    }
```

Aquí se presenta la capa del negocio para la administración de cantones, como podemos observar, el código es bastante similar al de la administración de empresas, difiere únicamente en los nombres de los métodos que se utilizan, de ahí, la lógica de programación es la misma.

CAPA DE INTERFACE DE USUARIO GUI

```
protected void btnEliminar_Click(object sender, EventArgs e)
{
    try
    {
        VR_CantonesBC objDatos = new VR_CantonesBC();
        if (txtCodigo.Text.Trim() == "")
        {
            MessageBox("Seleccione un registro para eliminar", txtNombre);
```

```
            return;
        }
        VR_CiudadesBC objCiudad = new VR_CiudadesBC();
        DataTable dttCiudades = objCiudad.ConsultaCiudadesPorCanton(txtNombre.Text).Tables[0];
        if (dttCiudades.Rows.Count > 0)
        {
            MessageBox("No se puede eliminar este cantón por que tiene ciudades asociadas.", btnEliminar);
            return;
        }
        objDatos.EliminaCantones(Convert.ToInt32(txtCodigo.Text), strUsuarioAPP[0]);
        MessageBox("El Registro se eliminó exitosamente");
        ConsultaCantones();
        limpiarDatos();
        blnNuevo = true;
    }
    catch (System.Exception ex)
    {
        MessageError(ex.Message);
    }
}
```

Es en esta capa en donde se informa al usuario que un cantón tiene asociadas ciudades y por lo tanto no se lo puede eliminar.

3.2.4 Interfaces de usuario

Al igual que la administración anterior, a continuación se presenta parte del código HTML de la interface de usuario, el código completo se lo puede ver en el anexo 3:

```
<table align="center" border="1" cellpadding="1" designtimedragdrop="304" style="z-index: 104;
    left: 7px; width: 468px; position: absolute; top: 267px; height: 32px">
    <tr align="center" class="fconsult">
        <td style="width: 462px; height: 26px">
            <asp:Button ID="btnNuevo" runat="server" CausesValidation="False" OnClick="btnNuevo_Click"
                TabIndex="17" Text="Nuevo" ToolTip="Nuevo Registro" Width="100px" /><asp:Button ID="wbtGrabar"
                    runat="server" OnClick="wbtGrabar_Click" TabIndex="17" Text="Grabar" ToolTip="Grabar Datos del Registro"
                    Width="100px" /><asp:Button ID="btnEliminar" runat="server" OnClick="btnEliminar_Click"
                        TabIndex="17" Text="Eliminar" ToolTip="Elimina un registro" Width="100px" />
```

```
                </td>
            </tr>
        </table>
```

El código que se presenta es la parte de los botones de la interface, el resultado de la implementación del módulo de cantones es el siguiente:

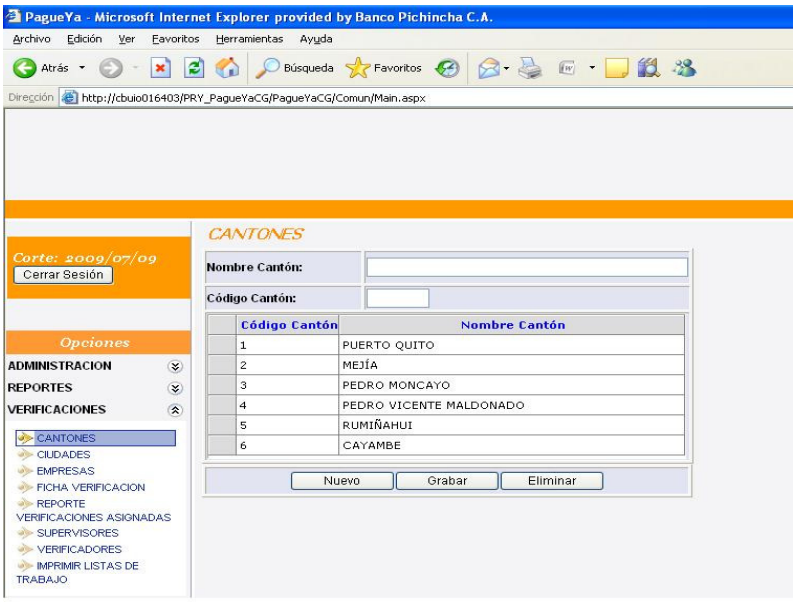


Figura 3.10: Resultado de la implementación del módulo de cantones

3.2.5 Pruebas unitarias

Al igual que el módulo anterior, se realizaron pruebas en ambiente de desarrollo del módulo, en condiciones ideales, a continuación los resultados de las pruebas:

INGRESANDO UN CANTÓN:

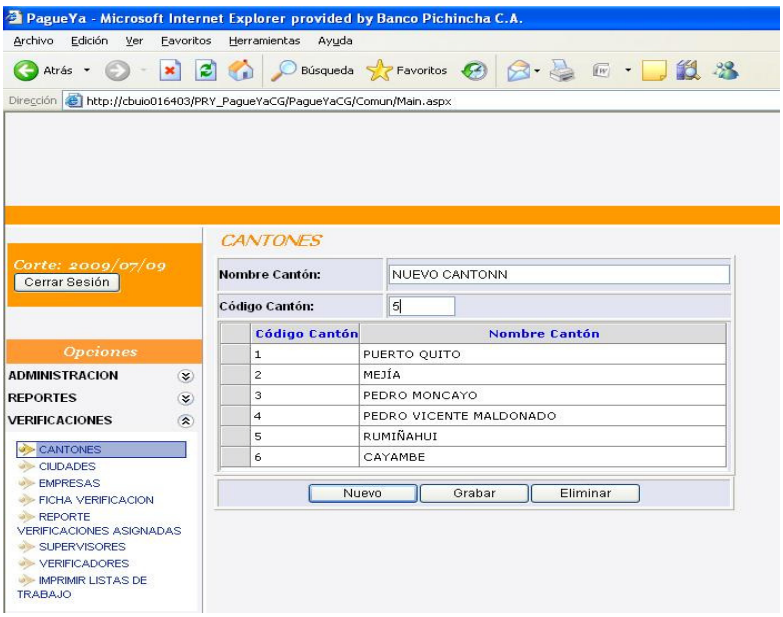


Figura 3.11: Ingresando los datos de un nuevo cantón

Para este caso, se va a ingresar un código repetido, el sistema debe advertir al usuario que ese código ya existe y que debe ingresar un código diferente:

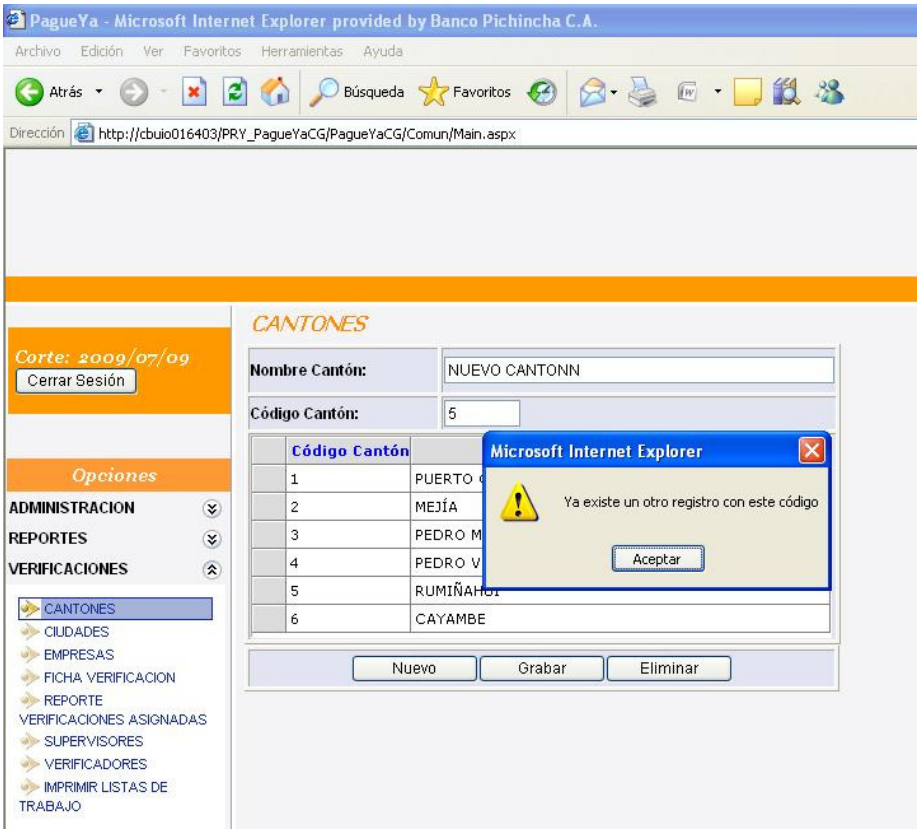


Figura 3.12: Advertencia de código repetido

Una vez que se ha corregido el error del código, el sistema permite el ingreso del nuevo registro a la base de datos.

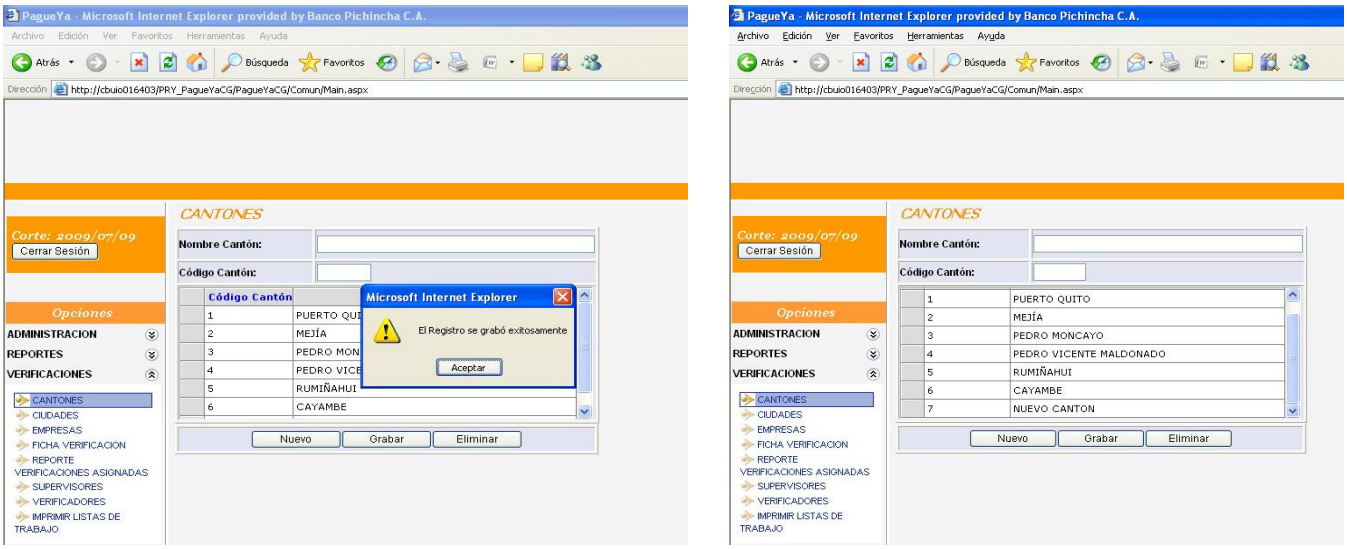


Figura 3.13: Ingresando un cantón correctamente

ACTUALIZANDO UN CANTÓN:

La actualización de los datos de un cantón tiene el mismo mecanismo que la actualización de datos que la administración de empresas, es decir, se selecciona un cantón de la lista y se cambia su nombre:

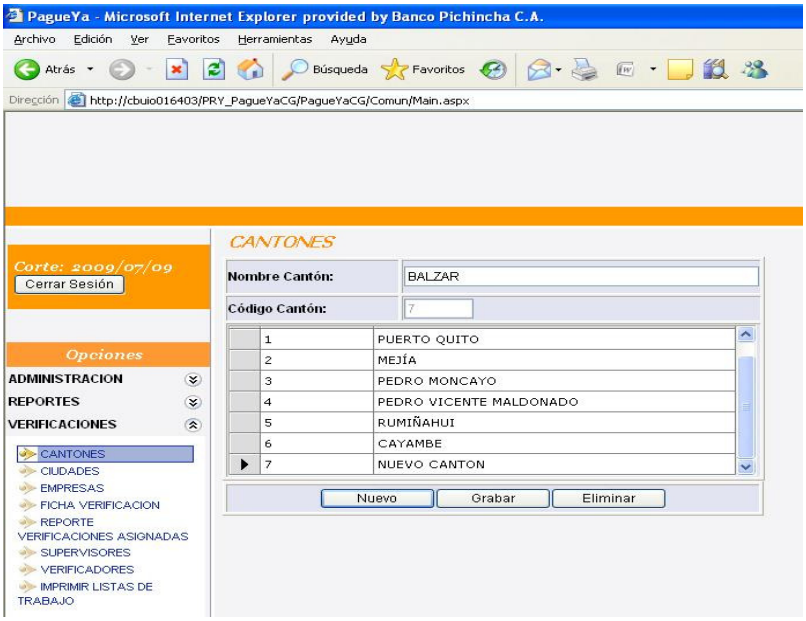


Figura 3.14: Actualización de los datos de cantones

En este caso, se cambiará el nombre del cantón al de BALZAR, se presiona el botón de grabar y se vuelve a presentar al usuario la lista de cantones con los cambios realizados:

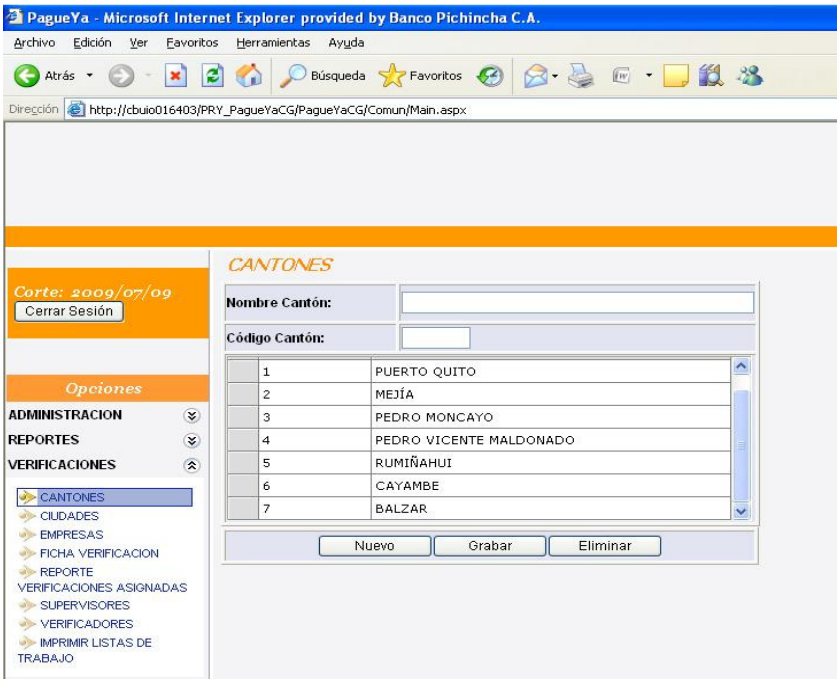


Figura 3.15: Presentación al usuario de cambios realizados en la información de cantones

ELIMINANDO UN CANTÓN:

Para esta funcionalidad se realizaron dos pruebas, la eliminación de un cantón si ciudades asociadas y el intento de eliminar un cantón con ciudades asociadas:

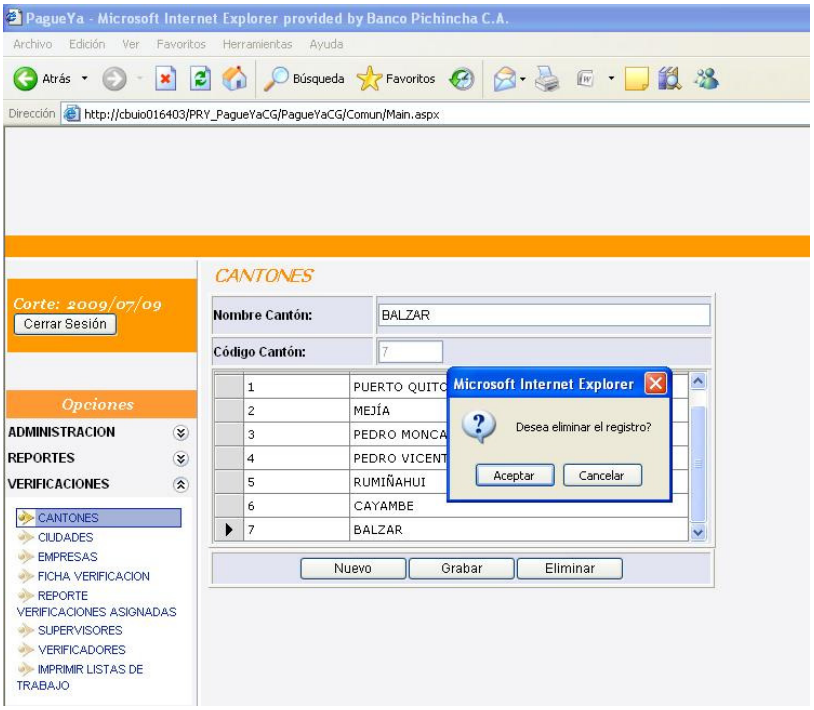


Figura 3.16: Eliminado un cantón sin ciudades asociadas

El proceso de eliminación de un cantón es básicamente en mismo que para las empresas, se selecciona un cantón de la lista y el sistema debe confirmar la eliminación de dicho registro, si es aceptada por el usuario, el sistema lo elimina:

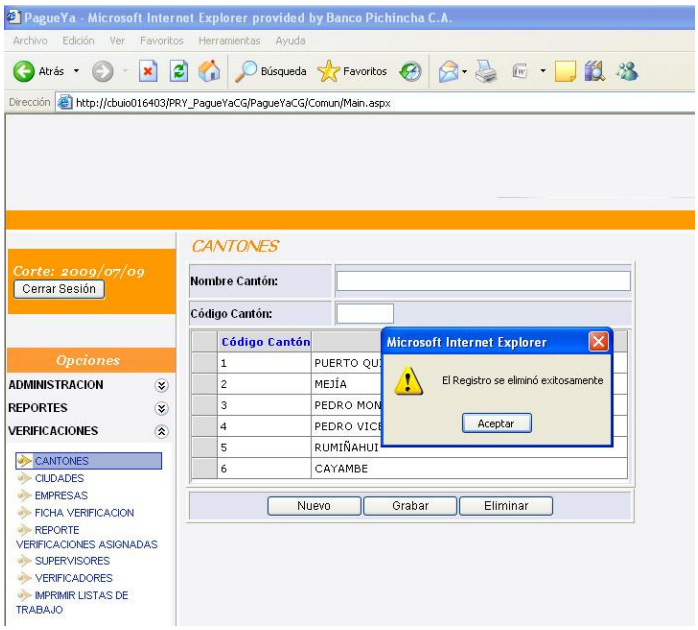


Figura 3.17: Eliminación exitosa de un cantón

Por el contrario, si se desea eliminar un cantón con ciudades asociadas, el mensaje al momento de confirmar la eliminación es el siguiente:

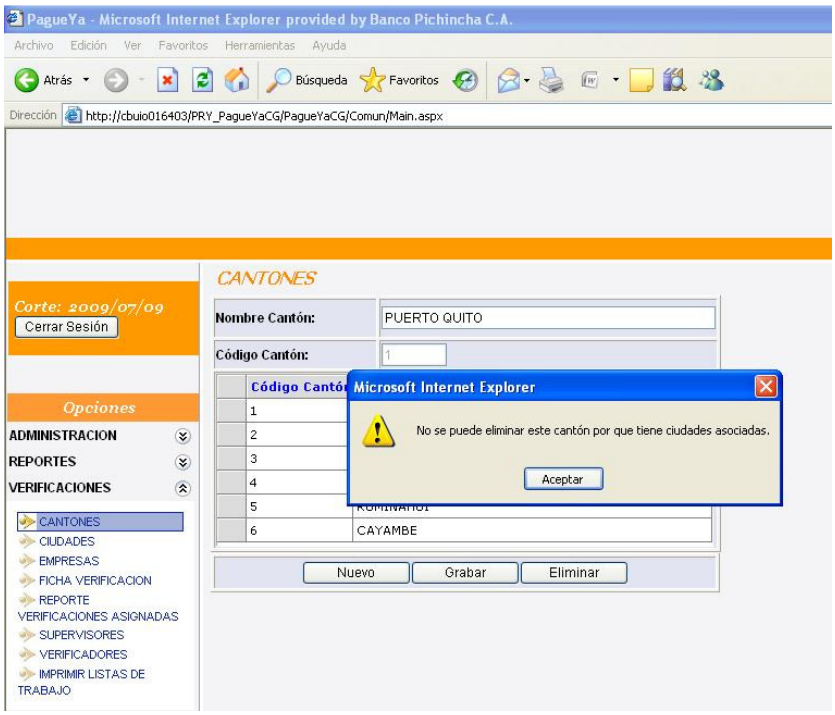


Figura 3.18: Intento de eliminar un cantón con ciudades asociadas.

Cabe recalcar que, no teniendo implementada aún la administración de ciudades, para realizar esta prueba se tuvo que ingresar directamente en la tabla de ciudades algunos registros que estén asociados al cantón que se intentaba eliminar.

Con la administración de cantones implementada, se puede realizar la implementación de la administración de ciudades ya que una ciudad debe tener asociada un cantón ingresado en la base de datos:

3.3 Administración de ciudades

Continuando con la zonificación, a los cantones se los ha dividido en ciudades, una ciudad tiene asociado un cantón, en la interface de usuario se debe dar la opcion de seleccionar un cantón para asociarlo con la ciudad.

3.3.1 Ingresar una nueva ciudad

Como en las administraciones anteriores, es la primera funcionalidad en ser implementada, la diferencia con las administraciones anteriores es que a más de insertar el código y el nombre de la ciudad, también se debe incluir el código del cantón, esto se lo realiza en el procedimiento almacenado destinado a insertar los datos de la ciudad, a continuación la implementación para el módulo de ciudades:

CAPA DE DATOS DC:

```
/// <summary>
        /// método de la capa de datos que se encarga de p
        /// grabar las ciudades por usuario
        /// </summary>
        /// <param name="strModo">modo</param>
        /// <param name="strCodigoFamilia">codigo de la ciudad</param>
        /// <param name="strUsuario">codigo usuario</param>
        /// <param name="strUsuarioAsignacion">usuario asignado</param>
        public void GrabaCiudadesUsuario(string strModo, string strCodigoCiudad, string strUsuario, string strUsuarioAsignacion)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[4];
                parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
                parms[0].Value = strModo;
                parms[1] = new SqlParameter("@i_CODIGO_CIUDAD", SqlDbType.Int);
                parms[1].Value = strCodigoCiudad;
                parms[2] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 8);
                parms[2].Value = strUsuario;
                parms[3] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
                parms[3].Value = strUsuarioAsignacion;
                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_CIUDAD_USUARIO", parms); //CREAR ESTE PROCEDURE PARA CIUDADES
            }
            catch (Exception ex)
            {
                throw (ex);
            }
            finally
            {
                if (objAccesoDatos != null)
                    objAccesoDatos = null;
```

```
        }
    }
    /// <summary>
    /// Método de la capa de datos que graba los registros de ciudades
    /// </summary>
    /// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
    /// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
    /// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
    ///
    //cambiar aqui el metodo del sp
    public void GrabaCiudad(string strModo, int intCodigoCiudad,string strNombreCanton, string strDescripcion)
    {
        AccesoDatos objAccesoDatos = new AccesoDatos();
        try
        {
            SqlParameter[] parms = new SqlParameter[4];
            parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
            parms[0].Value = strModo;
            parms[1] = new SqlParameter("@i_CODIGO_CIUDAD", SqlDbType.Int);
            parms[1].Value = intCodigoCiudad;
            parms[2] = new SqlParameter("@i_NOMBRE_CANTON", SqlDbType.VarChar,100);
            parms[2].Value = strNombreCanton;
            parms[3] = new SqlParameter("@i_DESCRIPCION", SqlDbType.VarChar, 50);
            parms[3].Value = strDescripcion;
            objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_CIUDAD", parms);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
        finally
        {
            if (objAccesoDatos != null)
                objAccesoDatos = null;
```

```
        }
    }
```

Al igual que en las demás administraciones, la administración de ciudades maneja en la capa de datos todas las llamadas que se realizan a la base de datos.

CAPA DEL NEGOCIO BC

```
/// <summary>
    /// método de la capa de logica que se encarga de
    /// grabar las ciudades por usuario
    /// </summary>
    /// <param name="strModo">modo</param>
    /// <param name="strCodigoFamilia">codigo de la ciudad</param>
    /// <param name="strUsuario">codigo usuario</param>
    /// <param name="strUsuarioAsignacion">usuario asignado</param>
    public void GrabaCiudadUsuario(string strModo, string strCodigoCiudad, string strUsuario, string strUsuarioAsignacion)
    {
        try
        {
            VR_CiudadesDC objCiudadesDC = new VR_CiudadesDC();
            objCiudadesDC.GrabaCiudadesUsuario(strModo, strCodigoCiudad, strUsuario, strUsuarioAsignacion);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
    }
    /// <summary>
    /// Método de la capa de logica que graba los registros de familias
    /// </summary>
    /// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
    /// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
    /// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
    public void GrabaCiudad(string strModo, int intCodigoCiudad, string strNombreCanton, string strDescripcion)
    {
        try
```

```
        {
            VR_CiudadesDC objCiudadesDC = new VR_CiudadesDC();
            //cambiar aqui el canton
            objCiudadesDC.GrabaCiudad(strModo, intCodigoCiudad, strNombreCanton, strDescripcion);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
    }
```

El código de la implementación de la capa del negocio es mucho menor que la capa de datos ya que en la capa del negocio simplemente lo que se hace es llamar a los métodos de la capa de datos.

CAPA DE INTERFACE DE USUARIO GUI

```
protected void wbtGrabar_Click(object sender, EventArgs e)
{
    try
    {
        if (txtNombre.Text.Length == 0)
        {
            MessageBox("Ingrese el nombre", txtNombre);
            return;
        }
        if (txtCodigo.Text.Length == 0)
        {
            MessageBox("Ingrese el código", txtCodigo);
            return;
        }
        string strModo = "";
        if (blnNuevo)
        {
            strModo = "I";
        }
        else
```

```
        {
            strModo = "M";
        }
        if (blnNuevo)
        {
            for (int i = 0; i < dttCiudad.Rows.Count; i++)
            {
                if (txtCodigo.Text == dttCiudad.Rows[i][0].ToString())
                {
                    MessageBox("Ya existe un otro registro con este código", txtNombre);
                    return;
                }
            }
        }
        VR_CiudadesBC objDatos = new VR_CiudadesBC();
        //Cambiar aqui el canton
        objDatos.GrabaCiudad(strModo,Convert.ToInt32(txtCodigo.Text), ddlCanton.Text, txtNombre.Text.ToUpper());
        MessageBox("El Registro se grabó exitosamente");
        ConsultaCiudades();
        limpiarDatos();
    }
    catch (System.Exception ex)
    {
        MessageError(ex.Message);
    }
}
```

Este es el código de la interface de usuario para el ingreso de ciudades, al igual que para las demás administraciones, se alerta al usuario que el registro se ha grabado correctamente.

3.3.2 Actualizar datos de las ciudades

Esta es la segunda funcionalidad del módulo y permite al usuario actualizar la información de una ciudad.

CAPA DE DATOS DC:

```
/// <summary>
        /// método de la capa de datos que se encarga de
        /// grabar las ciudades por usuario
        /// </summary>
        /// <param name="strModo">modo</param>
        /// <param name="strCodigoFamilia">codigo de la ciudad</param>
        /// <param name="strUsuario">codigo usuario</param>
        /// <param name="strUsuarioAsignacion">usuario asignado</param>
        public void GrabaCiudadesUsuario(string strModo, string strCodigoCiudad, string strUsuario, string strUsuarioAsignacion)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[4];
                parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
                parms[0].Value = strModo;
                parms[1] = new SqlParameter("@i_CODIGO_CIUDAD", SqlDbType.Int);
                parms[1].Value = strCodigoCiudad;
                parms[2] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 8);
                parms[2].Value = strUsuario;
                parms[3] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
                parms[3].Value = strUsuarioAsignacion;
                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_CIUDAD_USUARIO", parms); //CREAR ESTE PROCEDURE PARA CIUDADES
            }
            catch (Exception ex)
            {
                throw (ex);
            }
            finally
            {
                if (objAccesoDatos != null)
                    objAccesoDatos = null;
```

```
        }
    }
    /// <summary>
    /// Método de la capa de datos que graba los registros de ciudades
    /// </summary>
    /// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
    /// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
    /// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
    ///
    //cambiar aqui el metodo del sp
    public void GrabaCiudad(string strModo, int intCodigoCiudad,string strNombreCanton, string strDescripcion)
    {
        AccesoDatos objAccesoDatos = new AccesoDatos();
        try
        {
            SqlParameter[] parms = new SqlParameter[4];
            parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
            parms[0].Value = strModo;
            parms[1] = new SqlParameter("@i_CODIGO_CIUDAD", SqlDbType.Int);
            parms[1].Value = intCodigoCiudad;
            parms[2] = new SqlParameter("@i_NOMBRE_CANTON", SqlDbType.VarChar,100);
            parms[2].Value = strNombreCanton;
            parms[3] = new SqlParameter("@i_DESCRIPCION", SqlDbType.VarChar, 50);
            parms[3].Value = strDescripcion;
            objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_CIUDAD", parms);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
        finally
        {
            if (objAccesoDatos != null)
                objAccesoDatos = null;
```

```
        }
    }
```

Ya que el método para actualización de ciudades es el mismo que para el ingreso de ciudades, se colocó el mismo código en esta sección.

CAPA DEL NEGOCIO BC

```
/// <summary>
    /// método de la capa de logica que se encarga de
    /// grabar las ciudades por usuario
    /// </summary>
    /// <param name="strModo">modo</param>
    /// <param name="strCodigoFamilia">codigo de la ciudad</param>
    /// <param name="strUsuario">codigo usuario</param>
    /// <param name="strUsuarioAsignacion">usuario asignado</param>
    public void GrabaCiudadUsuario(string strModo, string strCodigoCiudad, string strUsuario, string strUsuarioAsignacion)
    {
       try
       {
          VR_CiudadesDC objCiudadesDC = new VR_CiudadesDC();
          objCiudadesDC.GrabaCiudadesUsuario(strModo, strCodigoCiudad, strUsuario, strUsuarioAsignacion);
       }
       catch (Exception ex)
       {
          throw (ex);
       }
    }
    /// <summary>
    /// Método de la capa de logica que graba los registros de familias
    /// </summary>
    /// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
    /// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
    /// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
    public void GrabaCiudad(string strModo, int intCodigoCiudad, string strNombreCanton, string strDescripcion)
    {
       try
```

```
        {
            VR_CiudadesDC objCiudadesDC = new VR_CiudadesDC();
            //cambiar aqui el canton
            objCiudadesDC.GrabaCiudad(strModo, intCodigoCiudad, strNombreCanton, strDescripcion);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
    }
```

El presente código pertenece al método de actualización de los datos de ciudades, en esta capa se llaman a los métodos de la capa de datos para comunicarse con la base de datos.

CAPA DE INTERFACE DE USUARIO GUI

```
protected void wbtGrabar_Click(object sender, EventArgs e)
{
    try
    {
        if (txtNombre.Text.Length == 0)
        {
            MessageBox("Ingrese el nombre", txtNombre);
            return;
        }
        if (txtCodigo.Text.Length == 0)
        {
            MessageBox("Ingrese el código", txtCodigo);
            return;
        }
        string strModo = "";
        if (blnNuevo)
        {
            strModo = "I";
        }
        else
        {
```

```
            strModo = "M";
        }
        if (blnNuevo)
        {
            for (int i = 0; i < dttCiudad.Rows.Count; i++)
            {
                if (txtCodigo.Text == dttCiudad.Rows[i][0].ToString())
                {
                    MessageBox("Ya existe un otro registro con este código", txtNombre);
                    return;
                }
            }
        }
        VR_CiudadesBC objDatos = new VR_CiudadesBC();
        //Cambiar aqui el canton
        objDatos.GrabaCiudad(strModo,Convert.ToInt32(txtCodigo.Text), ddlCanton.Text, txtNombre.Text.ToUpper());
        MessageBox("El Registro se grabó exitosamente");
        ConsultaCiudades();
        limpiarDatos();
    }
    catch (System.Exception ex)
    {
        MessageError(ex.Message);
    }
}
```

En la interface de usuario es en donde se pasan los parametros que ingresa el usuario para que las demás capas puedan procesar la información del usuario.

3.3.3 Eliminar ciudades

Al igual que las demás administraciones, este módulo tambien tiene la funcionalidad de eliminar la información de las ciudades almancenadas en la base de datos, a continuación el código del módulo separado por capas:

CAPA DE DATOS DC:

```
        /// <summary>
        /// Método de la capa de datos que se encarga de eliminar los datos de
        /// las ciudades filtradas pro usuario
        /// </summary>
        /// <param name="strUsuario">codigo de usuario</param>
        /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
        public void EliminaCiudadesUsuario(string strUsuario, string strUsuarioAsignacion)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[2];
                parms[0] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 8);
                parms[0].Value = strUsuario;
                parms[1] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
                parms[1].Value = strUsuarioAsignacion;
                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_BORRA_CIUDADES_USUARIOS", parms);
            }
            catch (Exception ex)
            {
                throw (ex);
            }
            finally
            {
                if (objAccesoDatos != null)
                    objAccesoDatos = null;
            }
        }
        /// <summary>
        /// Método de la capa de datos que se encarga de eliminar los datos de
        /// las ciudades
        /// </summary>
```

```
    /// <param name="strFamilia">codigo de usuario</param>
    /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
    public void EliminaCiudades(int intCiudad, string strUsuarioAsignacion)
    {
        AccesoDatos objAccesoDatos = new AccesoDatos();
        try
        {
            SqlParameter[] parms = new SqlParameter[2];
            parms[0] = new SqlParameter("@i_CODIGO_CIUDAD", SqlDbType.Int);
            parms[0].Value = intCiudad;
            parms[1] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
            parms[1].Value = strUsuarioAsignacion;
            objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure, "PRO_VR_BORRA_CIUDAD",
parms);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
        finally
        {
            if (objAccesoDatos != null)
                objAccesoDatos = null;
        }
    }
```

Este es el método de la capa de datos para eliminar ciudades, primero se deben eliminar todas la ciudades asociadas a un cantón para poder eliminar dicho cantón.

CAPA DEL NEGOCIO BC:

```
    /// <summary>
    /// Método de la capa de logica que se encarga de eliminar los datos de
    /// las ciudades filtradas pro usuario
    /// </summary>
    /// <param name="strFamilia">codigo de usuario</param>
    /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
```

```
public void EliminaCiudadesUsuario(string strUsuario, string strUsuarioAsignacion)
{
    try
    {
        VR_CiudadesDC objCiudadesDC = new VR_CiudadesDC();
        objCiudadesDC.EliminaCiudadesUsuario(strUsuario, strUsuarioAsignacion);
    }
    catch (Exception ex)
    {
        throw (ex);
    }
}
/// <summary>
/// Método de la capa de logica que se encarga de llamar a la capa de datos para eliminar
/// las ciudades
/// </summary>
/// <param name="strUsuario">codigo de usuario</param>
/// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
public void EliminaCiudades(int intCiudad, string strUsuarioAsignacion)
{
    try
    {
        VR_CiudadesDC objCiudadesDC = new VR_CiudadesDC();
        objCiudadesDC.EliminaCiudades(intCiudad, strUsuarioAsignacion);
    }
    catch (Exception ex)
    {
        throw (ex);
    }
}
```

Como podemos observar, en la capa del negocio, para poder comunicarnos con la capa de datos simplemente se debe crear un objeto que pertenezca a esta capa y se podrán utilizar sus métodos.

CAPA DE INTERFACE DE USUARIO GUI:

```
protected void btnEliminar_Click(object sender, EventArgs e)
{
    try
    {
        VR_CiudadesBC objDatos = new VR_CiudadesBC();
        if (txtCodigo.Text.Trim() == "")
        {
            MessageBox("Seleccione un registro para eliminar", txtNombre);
            return;
        }
        objDatos.EliminaCiudades(Convert.ToInt32(txtCodigo.Text), strUsuarioAPP[0]);
        MessageBox("El Registro se eliminó exitosamente");
        ConsultaCiudades();
        limpiarDatos();
        blnNuevo = true;
    }
    catch (System.Exception ex)
    {
        MessageError(ex.Message);
    }
}
```

En la interface de usuario se presenta el código del botón que permite eliminar los datos de una ciudad, como se puede observar, en esta capa se pasan los parámetros a las demás capas.

3.3.4 Interfaces de usuario

```
<table align="center" border="1" cellpadding="1" designtimedragdrop="304" style="z-index: 102;
    left: 8px; width: 468px; position: absolute; top: 297px; height: 32px">
    <tr align="center" class="fconsult">
        <td style="width: 462px; height: 26px">
            <asp:Button ID="btnNuevo" runat="server" CausesValidation="False" OnClick="btnNuevo_Click"
                TabIndex="17" Text="Nuevo" ToolTip="Nuevo Registro" Width="100px" /><asp:Button ID="wbtGrabar"
                    runat="server" OnClick="wbtGrabar_Click" TabIndex="17" Text="Grabar" ToolTip="Grabar Datos del Registro"
                    Width="100px" /><asp:Button ID="btnEliminar" runat="server" OnClick="btnEliminar_Click"
```

```
                TabIndex="17" Text="Eliminar" ToolTip="Elimina un registro" Width="100px" />
                    </td>
        </tr>
    </table>
```

El código que se presenta es el de la parte de los botones de la interface, como en las demás administraciones, el código completo de lo puede observar en el anexo 3

El resultado de la implementación del módulo de ciudades es el siguiente:

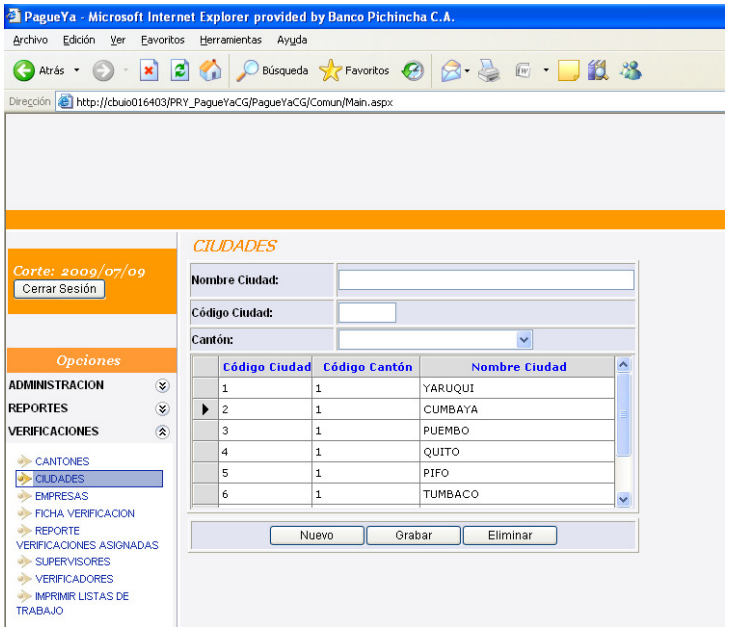


Figura 3.19: Resultado de la implementación del módulo de ciudades

Como podemos observar, en la interface de usuario se presenta una lista desplegable la cual tendrá la información de todos los cantones almacenados en la base de datos, esto facilitará la inserción de datos de ciudades ya que no se cometerán errores al ingresar el nombre de un cantón.

3.3.5 Pruebas unitarias

Ahora se van a realizar las mismas pruebas que para las demás administraciones de inserción actualización y eliminacioón de los datos de una ciudad

INGRESANDO UNA CIUDAD:

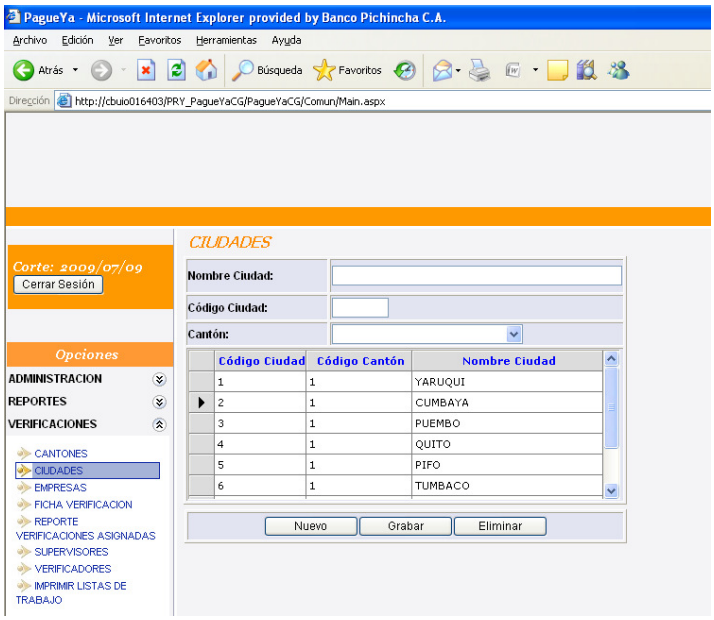


Figura 3.20: Primera pantalla para ingresar una nueva ciudad

En esta pantalla es en donde se ingresarán los datos de la nueva ciudad

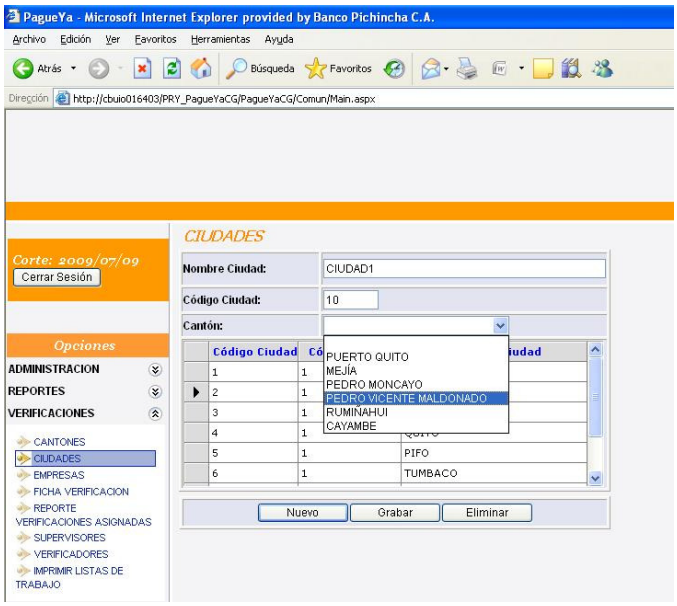


Figura 3.21: Ingresando los datos de la nueva ciudad

Como podemos observar, para asociar una ciudad a un cantón, basta con seleccionar al cantón de la ista desplegable, una vez que se han ingresado todos los datos, se presiona el botón de grabar y el sistema muestra un mensaje en el que se avisa al usuario que el registro se ha grabado exitosamente:

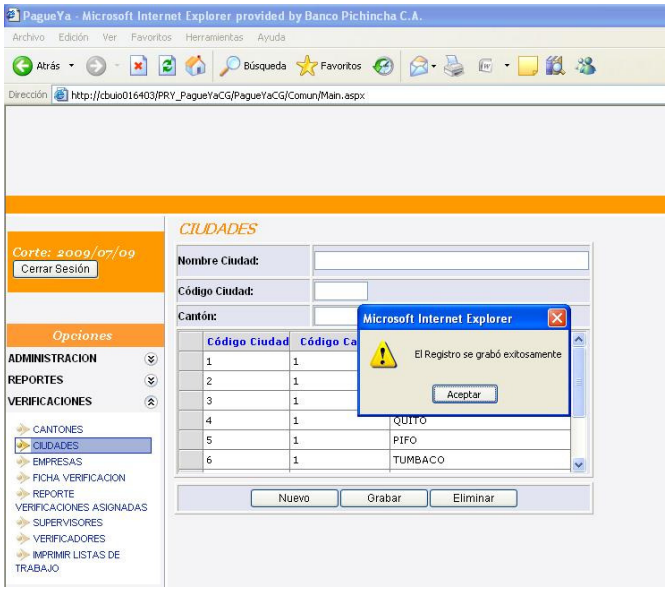


Figura 3.22: Mensaje de ingreso de datos exitoso.

ACTUALIZANDO UNA CIUDAD:

Esta es la segunda funcionalidad de este módulo, permite actualizar los datos de las ciudades ingresadas, a continuación las pruebas realizadas sobre el módulo para la actualización de datos:

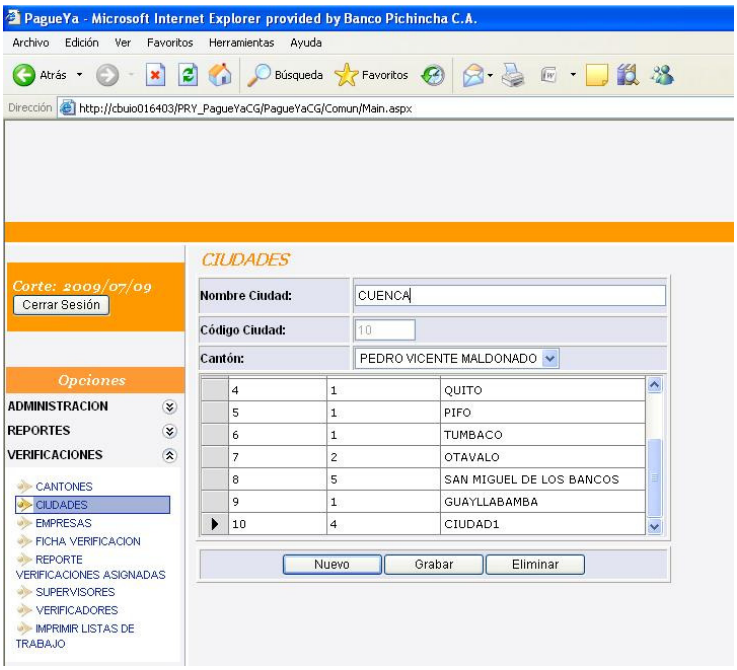


Figura 3.23: Cambiando el nombre de una ciudad

En este caso, vamos a cambiar el nombre de la ciudad ingresada con código 10 y le podremos el nombre de CUENCA

Después de esto se presiona el botón de grabar y el sistema muestra los cambios al usuario desplegándole nuevamente la lista de ciudades:

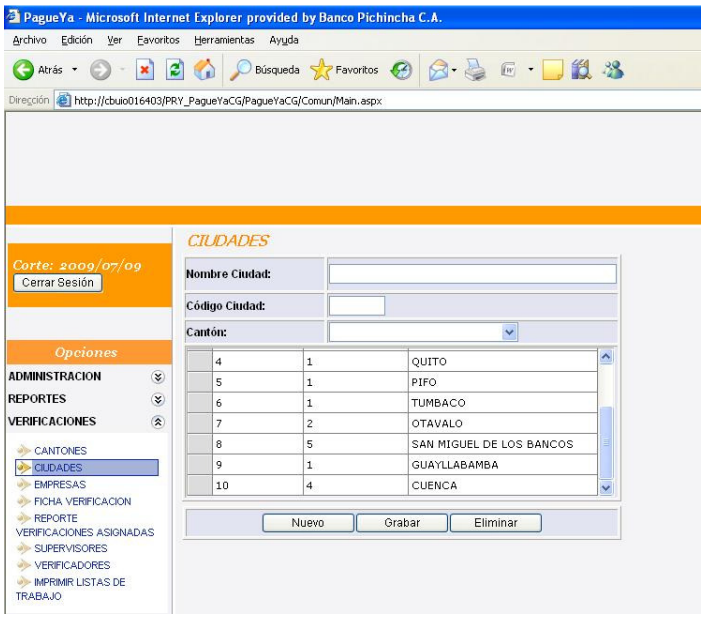


Figura 3.24: Mostrando los cambios realizados sobre un registro

Una vez que se han actualizado los datos de la ciudad, nos queda realizar pruebas a la última funcionalidad que es la de eliminar los datos de una ciudad

ELIMINANDO UNA CIUDAD:

Esta fucionalidad permite eliminar de la base de datos las ciudades que se encuentran almacenadas, esta funcionalidad es muy importante ya que es necesario eliminar todas las ciudades que tengan un cntón asiciado antes de que se pueda eliminar un cantón, el proceso de eliminación de las ciudades es el mismo que en las demás administraciones:

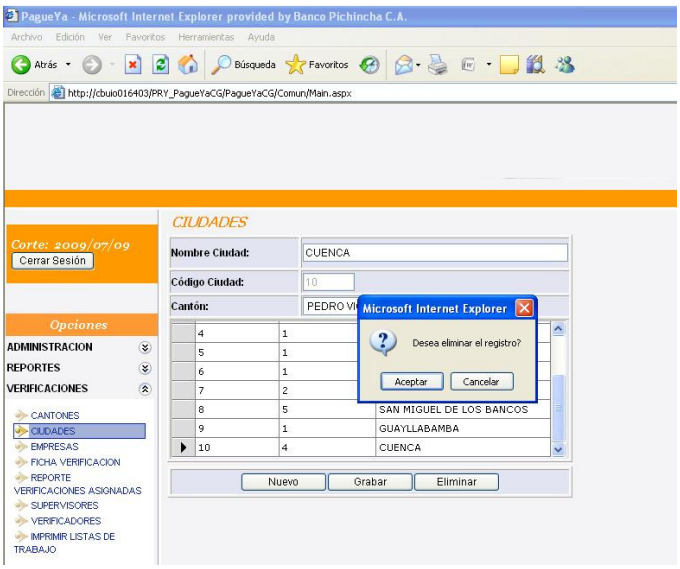


Figura 3.25: Eliminando los datos de una ciudad

Una vez que hemos seleccionado una ciudad de la lista, se presiona el botón de eliminar, y el sistema solicita una confirmación al usuario, si el usuario acepta la eliminación, el sistema borra de la base de datos la ciudad y presenta nuevamente la lista al usuario sin la ciudad que se borró

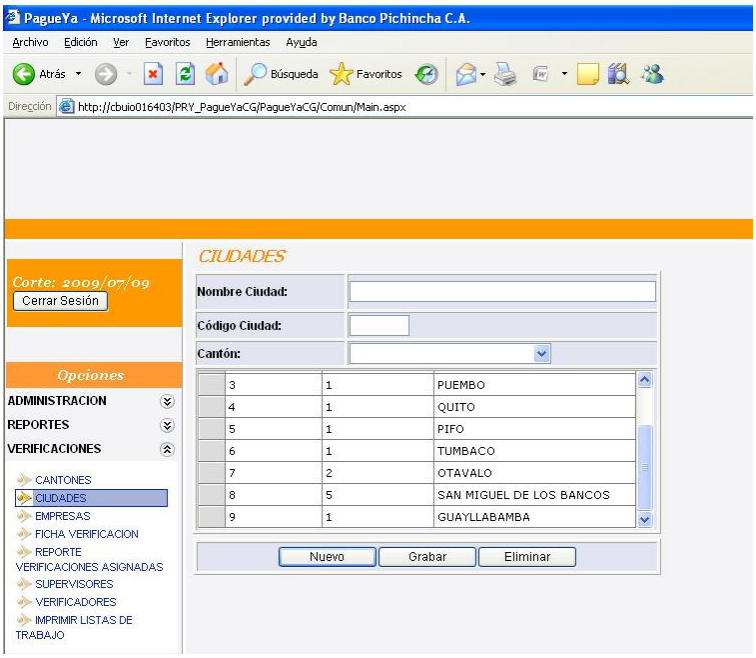


Figura 3.26: Presentación al usuario la lista de las ciudades después de aliminar un registro.

Una vez que se tienen definidas las zonas en las que se va a trabajar, es necesario organizar al personal para que realice la gestión de verificación de la información, para esto se ha decidido crear dos tipos de empleados, los supervisores y los verificadores, con el objetivo de tener un mejor control de la gestión de verificación de la información, a continuación las implementaciones para las administraciones de supervisores y de verificadores:

3.4 Administración de supervisores

3.4.1 Ingresar un nuevo supervisor

Esta es la primera funcionalidad del módulo de administración de supervisores, al igual que las anteriores administraciones, permite ingresar un nuevo registro a la base de datos con la información del supervisor, a continuación, el código fuente separado por capas del ingreso de un supervisor:

CAPA DE DATOS DC:

```
/// <summary>

/// Método de la capa de datos que graba los registros de supervisores

/// </summary>

/// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
```

```
    /// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
    /// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
    public void GrabaSupervisor(string strModo, int intCodigoSupervisor, string strUsuario, string strNombre)
    {
        AccesoDatos objAccesoDatos = new AccesoDatos();
        try
        {
            SqlParameter[] parms = new SqlParameter[4];
            parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
            parms[0].Value = strModo;
            parms[1] = new SqlParameter("@i_CODIGO_SUPERVISOR", SqlDbType.Int);
            parms[1].Value = intCodigoSupervisor;
            parms[2] = new SqlParameter("@i_USUARIO_SUPERVISOR", SqlDbType.VarChar, 100);
            parms[2].Value = strUsuario;
            parms[3] = new SqlParameter("@i_NOMBRE_SUPERVISOR", SqlDbType.VarChar, 100);
            parms[3].Value = strNombre;
            objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_SUPERVISOR", parms);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
        finally
        {
            if (objAccesoDatos != null)
                objAccesoDatos = null;
        }
    }
```

Como podemos observar, en esta capa se realiza la comunicación con la base de datos por medio de métodos que realizan esta tarea.

CAPA DEL NEGOCIO BC:

```
    /// <summary>
    /// método de la capa de logica que se encarga de
```

```
        /// grabar los supervisores por usuario
        /// </summary>
        /// <param name="strModo">modo</param>
        /// <param name="strCodigoFamilia">codigo del supervisor</param>
        /// <param name="strUsuario">codigo usuario</param>
        /// <param name="strUsuarioAsignacion">usuario asignado</param>
        public void GrabaSupervisorUsuario(string strModo, string strCodigoSupervisor, string strUsuario, string
strUsuarioAsignacion)
        {
            try
            {
                VR_SupervisoresDC objSupervisoresDC = new VR_SupervisoresDC();
                objSupervisoresDC.GrabaSupervisoresUsuario(strModo, strCodigoSupervisor, strUsuario, strUsuarioAsignacion);
            }
            catch (Exception ex)
            {
                throw (ex);
            }
        }
```

Esta es la capa que sirve como comunicación entre la capa de datos y la interface de usuario, esta capa recibe los parámetros de que el usuario ingresa por medio de la interface y los pasa a la capa de datos para que esta a su vez realice las consultas a la base de datos y retorne los resultados de dichas consultas.

CAPA DE INTERFACE DE USUARIO GUI:

```
    protected void wbtGrabar_Click(object sender, EventArgs e)
    {
        try
        {
            if (txtNombre.Text.Length == 0)
            {
                MessageBox("Ingrese el nombre", txtNombre);
                return;
            }
            if (txtCodigo.Text.Length == 0)
            {
```

```
            MessageBox("Ingrese el código", txtCodigo);
            return;
        }
        string strModo = "";
        if (blnNuevo)
        {
            strModo = "I";
        }
        else
        {
            strModo = "M";
        }
        if (blnNuevo)
        {
            for (int i = 0; i < dttSupervisor.Rows.Count; i++)
            {
                if (txtCodigo.Text == dttSupervisor.Rows[i][0].ToString())
                {
                    MessageBox("Ya existe un otro registro con este código", txtNombre);
                    return;
                }
            }
        }
        VR_SupervisoresBC objDatos = new VR_SupervisoresBC();
        objDatos.GrabaSupervisor(strModo, Convert.ToInt32(txtCodigo.Text), txtUsuario.Text.ToLower(),
txtNombre.Text.ToUpper());
        MessageBox("El Registro se grabó exitosamente");
        ConsultaSupervisores();
        limpiarDatos();
      }
      catch (System.Exception ex)
      {
        MessageError(ex.Message);
      }
    }
```

Este es el código de la interface de usuario, es en esta capa donde se reciben los parámetros que el usuario ingresa para un nuevo supervisor.

3.4.2 Actualizar datos de los supervisores

Esta es a segunda funcionalidad de este módulo, lo que permite esta funcionalidad es la actualización de los supervisores que se encuentran almacenados en la base de datos, utilizando el re uso, se utiliza el mismo código que para el ingreso de un nuevo supervisor, la diferencia radica en el parámetro tipo que es el que indica que acción se está realizando:

CAPA DE DATOS DC:

```
/// <summary>
/// Método de la capa de datos que graba los registros de supervisores
/// </summary>
/// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
/// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
/// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
public void GrabaSupervisor(string strModo, int intCodigoSupervisor, string strUsuario, string strNombre)
{
   AccesoDatos objAccesoDatos = new AccesoDatos();
   try
   {
      SqlParameter[] parms = new SqlParameter[4];
      parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
      parms[0].Value = strModo;
      parms[1] = new SqlParameter("@i_CODIGO_SUPERVISOR", SqlDbType.Int);
      parms[1].Value = intCodigoSupervisor;
      parms[2] = new SqlParameter("@i_USUARIO_SUPERVISOR", SqlDbType.VarChar, 100);
      parms[2].Value = strUsuario;
      parms[3] = new SqlParameter("@i_NOMBRE_SUPERVISOR", SqlDbType.VarChar, 100);
      parms[3].Value = strNombre;
      objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_SUPERVISOR", parms);
      }
      catch (Exception ex)
      {
         throw (ex);
```

```
        }
        finally
        {
            if (objAccesoDatos != null)
                objAccesoDatos = null;
        }
    }
```

CAPA DEL NEGOCIO BC:

```
    /// <summary>
    /// método de la capa de logica que se encarga de
    /// grabar los supervisores por usuario
    /// </summary>
    /// <param name="strModo">modo</param>
    /// <param name="strCodigoFamilia">codigo del supervisor</param>
    /// <param name="strUsuario">codigo usuario</param>
    /// <param name="strUsuarioAsignacion">usuario asignado</param>
    public void GrabaSupervisorUsuario(string strModo, string strCodigoSupervisor, string strUsuario, string
strUsuarioAsignacion)
    {
        try
        {
            VR_SupervisoresDC objSupervisoresDC = new VR_SupervisoresDC();
            objSupervisoresDC.GrabaSupervisoresUsuario(strModo, strCodigoSupervisor, strUsuario, strUsuarioAsignacion);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
    }
```

CAPA DE INTERFACE DE USUARIO GUI:

```
protected void wbtGrabar_Click(object sender, EventArgs e)
{
    try
```

```
{
    if (txtNombre.Text.Length == 0)
    {
        MessageBox("Ingrese el nombre", txtNombre);
        return;
    }
    if (txtCodigo.Text.Length == 0)
    {
        MessageBox("Ingrese el código", txtCodigo);
        return;
    }
    string strModo = "";
    if (blnNuevo)
    {
        strModo = "I";
    }
    else
    {
        strModo = "M";
    }
    if (blnNuevo)
    {
        for (int i = 0; i < dttSupervisor.Rows.Count; i++)
        {
            if (txtCodigo.Text == dttSupervisor.Rows[i][0].ToString())
            {
                MessageBox("Ya existe un otro registro con este código", txtNombre);
                return;
            }
        }
    }
    VR_SupervisoresBC objDatos = new VR_SupervisoresBC();
    objDatos.GrabaSupervisor(strModo, Convert.ToInt32(txtCodigo.Text), txtUsuario.Text.ToLower(),
txtNombre.Text.ToUpper());
    MessageBox("El Registro se grabó exitosamente");
```

```
            ConsultaSupervisores();
            limpiarDatos();
        }
        catch (System.Exception ex)
        {
            MessageError(ex.Message);
        }
    }
```

3.4.3 Eliminar supervisores

Esta es la última funcionalidad del módulo de supervisores, esta funcionalidad lo que nos permite hacer es eliminar la información almacenada de cualquier supervisor, siempre y cuando este no tenga asociados verificadores, a continuación el código de esta funcionalidad:

CAPA DE DATOS DC:

```
        /// <summary>
        /// Método de la capa de datos que se encarga de eliminar los datos de
        /// los supervisores
        /// </summary>
        /// <param name="strFamilia">codigo de usuario</param>
        /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
        public void EliminaSupervisores(int intSupervisor, string strUsuarioAsignacion)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[2];
                parms[0] = new SqlParameter("@i_CODIGO_SUPERVISOR", SqlDbType.VarChar, 8);
                parms[0].Value = intSupervisor;
                parms[1] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
                parms[1].Value = strUsuarioAsignacion;
                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_BORRA_SUPERVISOR", parms);
            }
            catch (Exception ex)
            {
```

```
            throw (ex);
        }
        finally
        {
            if (objAccesoDatos != null)
                objAccesoDatos = null;
        }
    }
```

CAPA DEL NEGOCIO BC:

```
    /// <summary>
    /// Método de la capa de logica que se encarga de llamar a la capa de datos para eliminar
    /// los supervisores
    /// </summary>
    /// <param name="strUsuario">codigo de usuario</param>
    /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
    public void EliminaSupervisores(int intSupervisor, string strUsuarioAsignacion)
    {
        try
        {
            VR_SupervisoresDC objSupervisoresDC = new VR_SupervisoresDC();
            objSupervisoresDC.EliminaSupervisores(intSupervisor, strUsuarioAsignacion);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
    }
```

CAPA DE INTERFACE DE USUARIO GUI:

```
protected void btnEliminar_Click(object sender, EventArgs e)
{
    try
    {
        VR_SupervisoresBC objDatos = new VR_SupervisoresBC();
        if (txtCodigo.Text.Trim() == "")
```

```
        {
            MessageBox("Seleccione un registro para eliminar", txtNombre);
            return;
        }
        VR_AsesoresBC objAsesor = new VR_AsesoresBC();
        DataTable dttAsesores = objAsesor.ConsultaAsesoresPorSupervisor(txtUsuario.Text).Tables[0];
        if (dttAsesores.Rows.Count > 0)
        {
            MessageBox("No se puede eliminar este supervisor por que tiene verificadores asociados.", btnEliminar);
            return;
        }
        objDatos.EliminaSupervisores(Convert.ToInt32(txtCodigo.Text), strUsuarioAPP[0]);
        MessageBox("El Registro se eliminó exitosamente");
        ConsultaSupervisores();
        limpiarDatos();

        blnNuevo = true;
    }
    catch (System.Exception ex)
    {
        MessageError(ex.Message);
    }
}
```

3.4.4 Interfaces de usuario

Una vez que se tienen implementadas las tres capas de la arquitectura de desarrollo, se procede a la implementación de la interface de usuario, al igual que para las demás administraciones, solo se pondrá una parte del código de la interface de usuario, el resto del código, se encuentra en el anexo 3

```
<table align="center" border="1" cellpadding="1" designtimedragdrop="304" style="z-index: 104;
    left: 8px; width: 468px; position: absolute; top: 302px; height: 32px">
    <tr align="center" class="fconsult">
        <td style="width: 462px; height: 26px">
            <asp:Button ID="btnNuevo" runat="server" CausesValidation="False" OnClick="btnNuevo_Click"
                TabIndex="17" Text="Nuevo" ToolTip="Nuevo Registro" Width="100px" /><asp:Button ID="wbtGrabar"
                    runat="server" OnClick="wbtGrabar_Click" TabIndex="17" Text="Grabar" ToolTip="Grabar Datos del Registro"
```

```
Width="100px" /><asp:Button ID="btnEliminar" runat="server" OnClick="btnEliminar_Click"

TabIndex="17" Text="Eliminar" ToolTip="Elimina un registro" Width="100px" />

</tr>

</table>
```

Esta sección de código pertenece a os botones de la interface, a continuación el resultado de la implementación de supervisores:

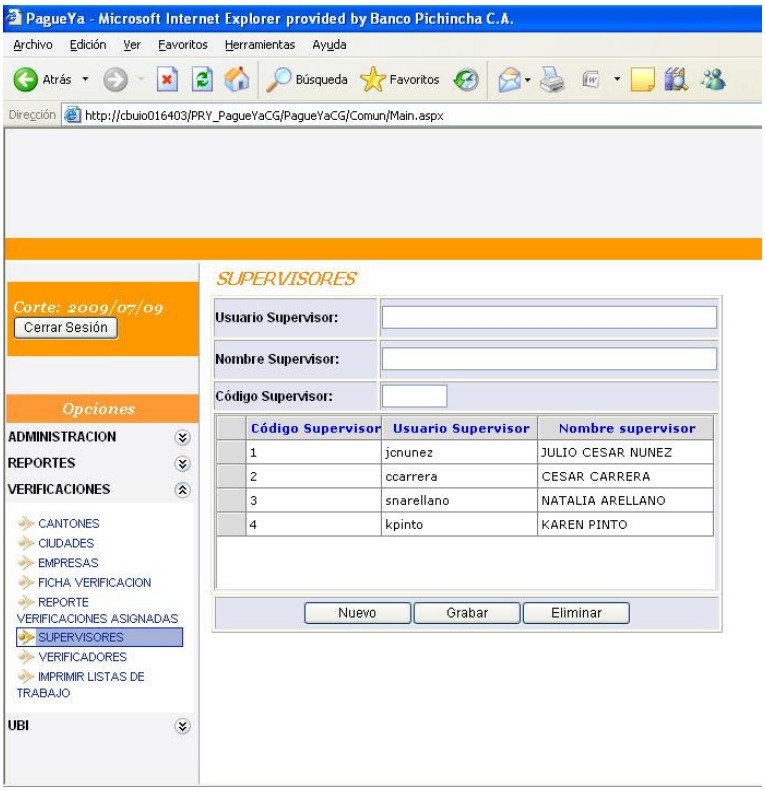


Figura 3.27: Resultado de al implementación del módulo de administración de supervisores

3.4.5 Pruebas unitarias

Siguiendo el ciclo de vida de cascada, una vez que se ha realizado la implementación de uno de los módulos del sistema, se procede realizar pruebas sobre este módulo:

INGRESANDO UN SUPERVISOR:

La primera de las pruebas que se realizan es la de inserción de datos, en este caso se ingresa el nombre, el usuario y el código del supervisor:

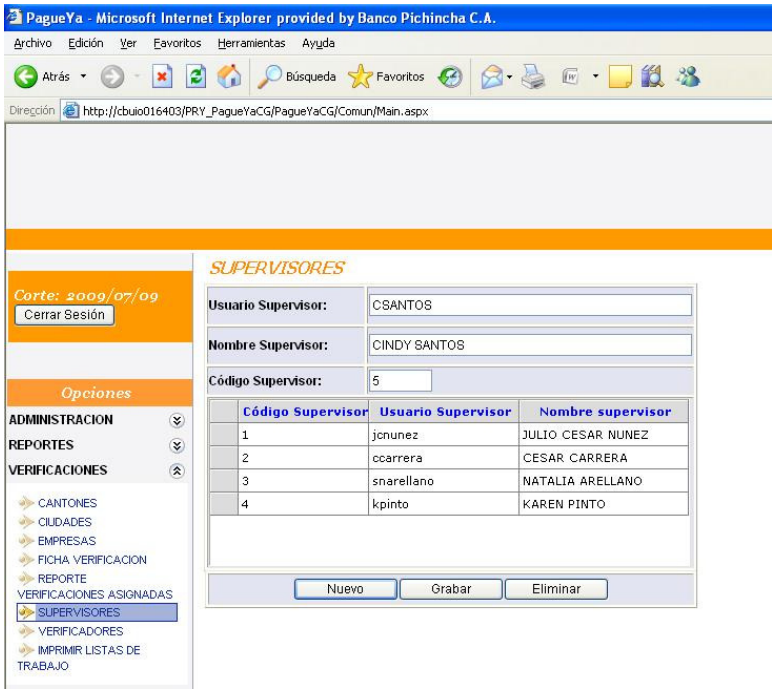


Figura 3.28: Ingresando los datos de un supervisor

Una vez que se han ingresado los datos del nuevo supervisor, se procede a presionar el botón de grabar, si los datos son correctos, el sistema informa al usuario que el registro se grabó exitosamente

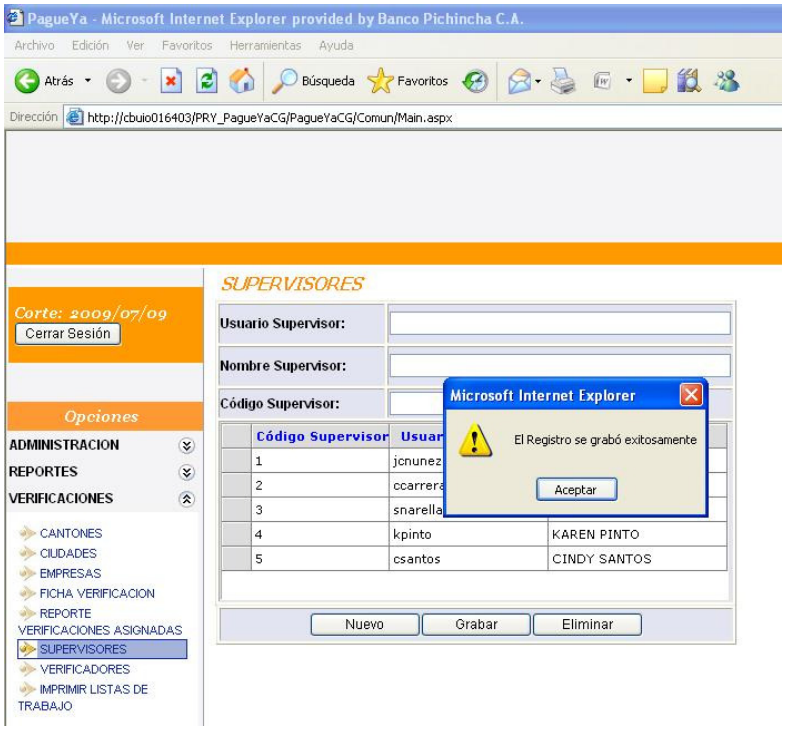


Figura 3.29: El sistema informa al usuario que el registro se ha grabado correctamente

ACTUALIZANDO UN SUPERVISOR:

Con información de supervisores en la base de datos, se pueden realizar modificaciones sobre la información almacenada, para esto basta con seleccionar un supervisor de la lista y modificar la información que se presente al usuario, en este caso, se va a actualizar al nombre del supervisor:

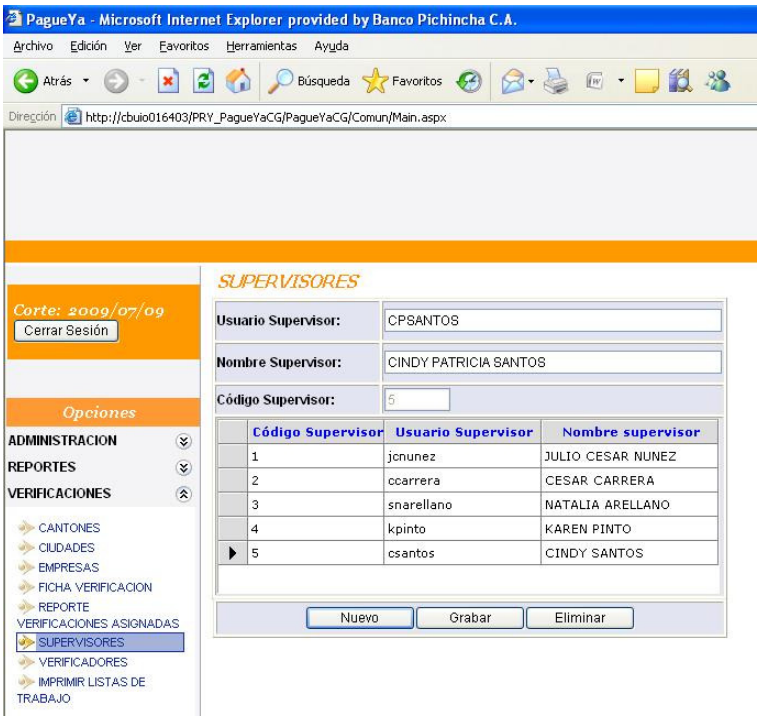


Figura 3.30: Actualizando los datos de un supervisor

Una vez que se ha modificado la información del supervisor, se presiona el botón de grabar y el sistema informa al usuario que se ha grabado la información correctamente:

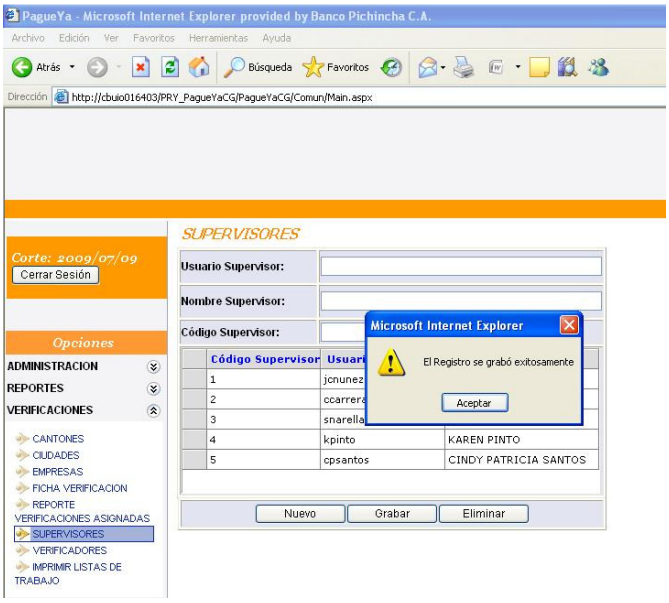


Figura 3.31: El sistema informa al usuario que la información se grabó correctamente.

ELIMINANDO UN SUPERVISOR:

Esta es la última prueba que se realiza sobre este módulo, la eliminación de un supervisor, para esto, se debe seleccionar un supervisor de la lista y se presiona el botón eliminar, el sistema solicita al usuario una confirmación para eliminar el registro:

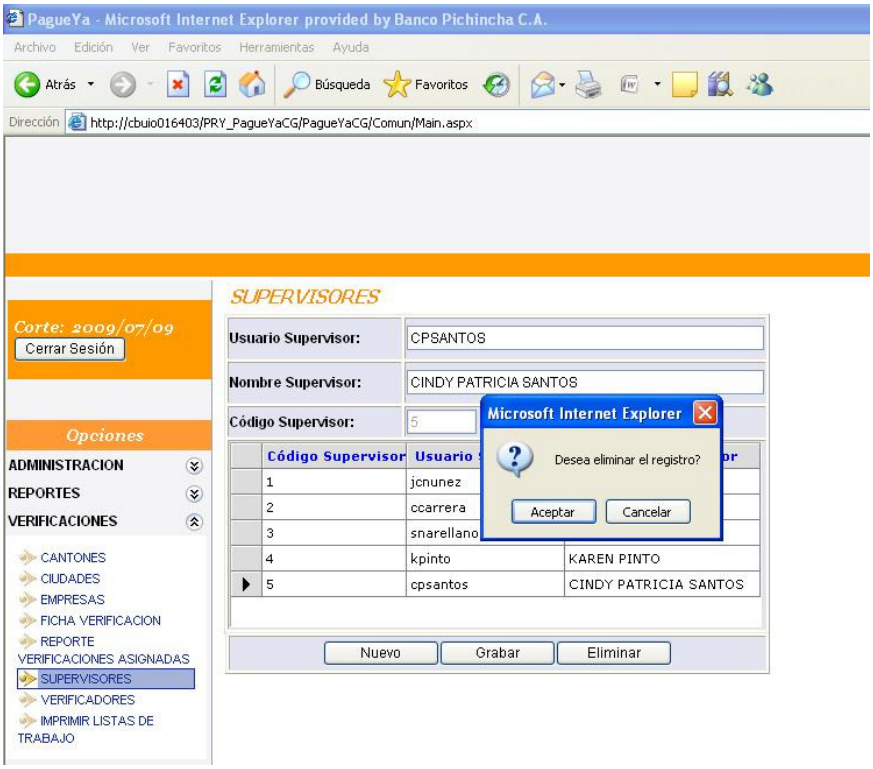


Figura 3.32: Eliminando un supervisor

Si el supervisor no tiene asignado un verificador, se lo eliminará del sistema, de lo contrario, se alerta al usuario que el registro no puede ser eliminado por que tiene asignados verificadores:

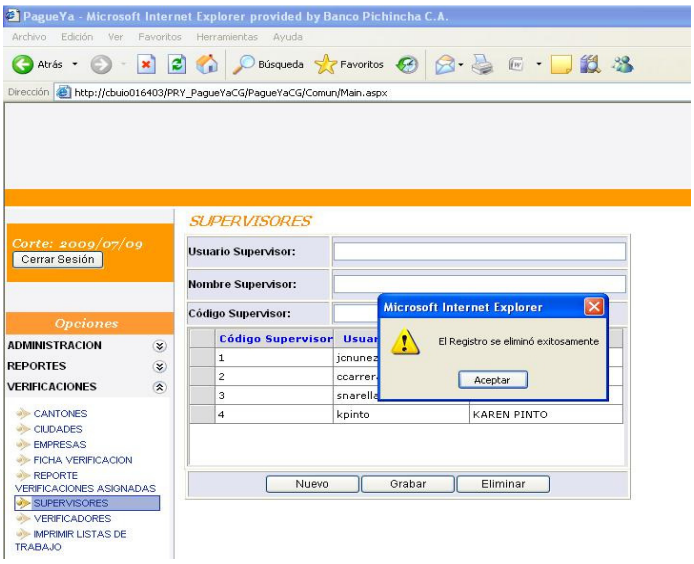


Figura 3.33: Eliminando un registro sin verificadores asignados

A continuación, se procede a eliminar un registro que tiene asignado un verificador, el proceso es el mismo, solo que al momento de aceptar la eliminación del registro, el sistema alerta al usuario que no se puede realizar esta eliminación:

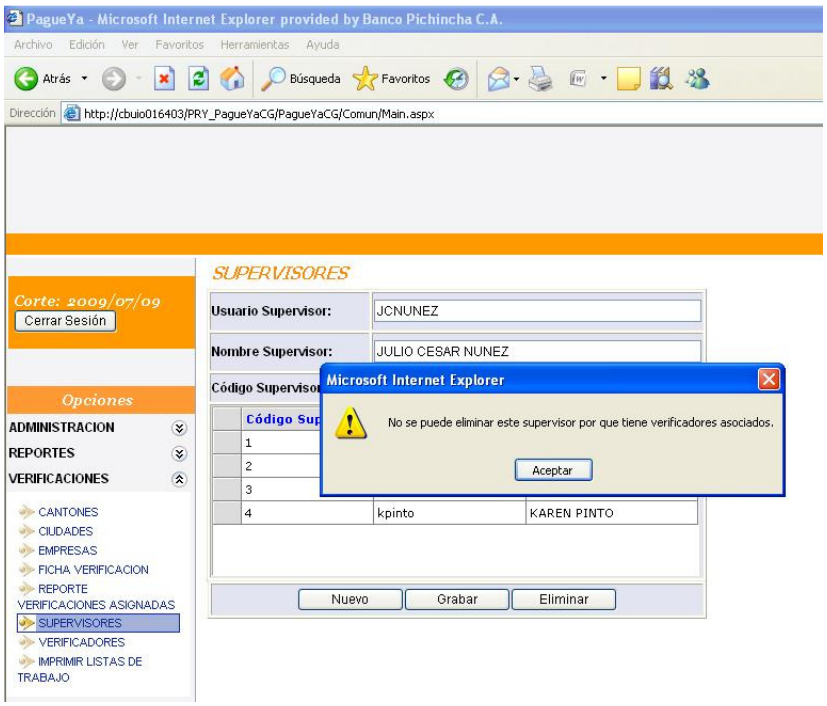


Figura 3.34: Alerta al usuario que no se puede eliminar un registro.

Una vez que ha implementado la administración de supervisores, se puede realizar la administración de verificadores ya que, al igual que una ciudad tiene asociada un cantón, un verificador tiene asociado un supervisor el cual debe supervisar el trabajo del grupo de verificadores que tiene asociados.

3.5 Administración de verificadores

3.5.1 Ingresar un nuevo verificador

Esta es la primera funcionalidad del módulo de administración de verificadores, al igual que las anteriores administraciones, permite ingresar un nuevo registro a la base de datos con la información del verificador, a continuación, el código fuente separado por capas del ingreso de un supervisor:

CAPA DE DATOS DC:

```
/// <summary>
    /// método de la capa de datos que se encarga de
    /// grabar los asesores por usuario
    /// </summary>
    /// <param name="strModo">modo</param>
    /// <param name="strCodigoFamilia">codigo del asesor</param>
    /// <param name="strUsuario">codigo usuario</param>
```

```
        /// <param name="strUsuarioAsignacion">usuario asignado</param>
        public void GrabaAsesoresUsuario(string strModo, string strCodigoAsesor, string strUsuario, string strUsuarioAsignacion)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[4];
                parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
                parms[0].Value = strModo;
                parms[1] = new SqlParameter("@i_CODIGO_ASESOR", SqlDbType.Int);
                parms[1].Value = strCodigoAsesor;
                parms[2] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 8);
                parms[2].Value = strUsuario;
                parms[3] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
                parms[3].Value = strUsuarioAsignacion;

                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_ASESOR_USUARIO", parms);
            }
            catch (Exception ex)
            {
                throw (ex);
            }
            finally
            {
                if (objAccesoDatos != null)
                    objAccesoDatos = null;
            }
        }
        /// <summary>
        /// Método de la capa de datos que graba los registros de asesores
        /// </summary>
        /// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
        /// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
        /// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
```

```
public void GrabaAsesor(string strModo, int intCodigoAsesor, string strNombreSupervisor, string strUsrasesor, string
Nmbrasesor)

    {

        AccesoDatos objAccesoDatos = new AccesoDatos();

        try

        {

            SqlParameter[] parms = new SqlParameter[5];

            parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);

            parms[0].Value = strModo;

            parms[1] = new SqlParameter("@i_CODIGO_ASESOR", SqlDbType.Int);

            parms[1].Value = intCodigoAsesor;

            parms[2] = new SqlParameter("@i_NOMBRE_SUPERVISOR", SqlDbType.VarChar,100);

            parms[2].Value = strNombreSupervisor;

            parms[3] = new SqlParameter("@i_USUARIO_ASESOR", SqlDbType.VarChar,100);

            parms[3].Value = strUsrasesor;

            parms[4] = new SqlParameter("@i_NOMBRE_ASESOR", SqlDbType.VarChar,100);

            parms[4].Value = Nmbrasesor;

            objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_ASESOR", parms);

        }

        catch (Exception ex)

        {

            throw (ex);

        }

        finally

        {

            if (objAccesoDatos != null)

                objAccesoDatos = null;

        }

    }
```

Como podemos observar, en esta capa se realiza la comunicación con la base de datos por medio de métodos que realizan esta tarea y de esta manera se pueden almacenar los datos de los verificadores.

CAPA DEL NEGOCIO BC

```
/// <summary>
        /// método de la capa de logica que se encarga de
        /// grabar los asesores por usuario
        /// </summary>
        /// <param name="strModo">modo</param>
        /// <param name="strCodigoFamilia">codigo del asesor</param>
        /// <param name="strUsuario">codigo usuario</param>
        /// <param name="strUsuarioAsignacion">usuario asignado</param>
        public void GrabaAsesorUsuario(string strModo, string strCodigoAsesor, string strUsuario, string strUsuarioAsignacion)
        {
            try
            {
                VR_AsesoresDC objAsesoresDC = new VR_AsesoresDC();
                objAsesoresDC.GrabaAsesoresUsuario(strModo, strCodigoAsesor, strUsuario, strUsuarioAsignacion);
            }
            catch (Exception ex)
            {
                throw (ex);
            }
        }

        /// <summary>
        /// Método de la capa de logica que graba los registros de asesores
        /// </summary>
        /// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
        /// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
        /// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
        public void GrabaAsesor(string strModo, int intCodigoAsesor, string strNombreSupervisor, string strUsrasesor, string
Nmbrasesor)
        {
            try
            {
                VR_AsesoresDC objAsesoresDC = new VR_AsesoresDC();
                objAsesoresDC.GrabaAsesor(strModo, intCodigoAsesor, strNombreSupervisor, strUsrasesor, Nmbrasesor);
            }
```

```
        catch (Exception ex)
        {
            throw (ex);
        }
    }
```

Esta es la capa que sirve como comunicación entre la capa de datos y la interface de usuario, esta capa recibe los parámetros de que el usuario ingresa por medio de la interface y los pasa a la capa de datos para que esta a su vez realice las consultas a la base de datos y retorne los resultados de dichas consultas.

CAPA DE INTERFACE DE USUARIO GUI

```
protected void wbtGrabar_Click(object sender, EventArgs e)
{
    try
    {
        if (txtNombre.Text.Length == 0)
        {
            MessageBox("Ingrese el nombre", txtNombre);
            return;
        }
        if (txtCodigo.Text.Length == 0)
        {
            MessageBox("Ingrese el código", txtCodigo);
            return;
        }
        string strModo = "";

        if (blnNuevo)
        {
            strModo = "I";
        }
        else
        {
            strModo = "M";
        }
```

```
        if (blnNuevo)
        {
            for (int i = 0; i < dttAsesor.Rows.Count; i++)
            {
                if (txtCodigo.Text == dttAsesor.Rows[i][0].ToString())
                {
                    MessageBox("Ya existe un otro registro con este código", txtNombre);
                    return;
                }
            }
        }
        VR_AsesoresBC objDatos = new VR_AsesoresBC();
        objDatos.GrabaAsesor(strModo, Convert.ToInt32(txtCodigo.Text), ddlSupervisor.Text, txtUsuario.Text.ToLower(),
txtNombre.Text.ToUpper());

        MessageBox("El Registro se grabó exitosamente");
        ConsultaAsesores();
      }
      catch (System.Exception ex)
      {
        MessageError(ex.Message);
      }
  }
```

Este es el código de la interface de usuario, es en esta capa donde se reciben los parámetros que el usuario ingresa para un nuevo verificador; dichos parámetros pasan a través de las demás capas para poder ser amacenados en la base de datos.

3.5.2 Actualizar datos de los verificadores

Esta es a segunda funcionalidad del módulo de verificadores, lo que permite es la actualización de los verificadores que se encuentran almacenados en la base de datos, por medio del re uso, se utiliza el mismo código que para el ingreso de un nuevo supervisor, la diferencia radica en el parámetro tipo que es el que indica que acción se está realizando:

CAPA DE DATOS DC:

```
/// <summary>
```

```
        /// método de la capa de datos que se encarga de
        /// grabar los asesores por usuario
        /// </summary>
        /// <param name="strModo">modo</param>
        /// <param name="strCodigoFamilia">codigo del asesor</param>
        /// <param name="strUsuario">codigo usuario</param>
        /// <param name="strUsuarioAsignacion">usuario asignado</param>
        public void GrabaAsesoresUsuario(string strModo, string strCodigoAsesor, string strUsuario, string strUsuarioAsignacion)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[4];
                parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
                parms[0].Value = strModo;
                parms[1] = new SqlParameter("@i_CODIGO_ASESOR", SqlDbType.Int);
                parms[1].Value = strCodigoAsesor;
                parms[2] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 8);
                parms[2].Value = strUsuario;
                parms[3] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
                parms[3].Value = strUsuarioAsignacion;

                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_ASESOR_USUARIO", parms);
            }
            catch (Exception ex)
            {
                throw (ex);
            }
            finally
            {
                if (objAccesoDatos != null)
                    objAccesoDatos = null;
            }
        }
```

```
        /// <summary>
        /// Método de la capa de datos que graba los registros de asesores
        /// </summary>
        /// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
        /// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
        /// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
        public void GrabaAsesor(string strModo, int intCodigoAsesor, string strNombreSupervisor, string strUsrasesor, string
Nmbrasesor)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[5];
                parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char, 1);
                parms[0].Value = strModo;
                parms[1] = new SqlParameter("@i_CODIGO_ASESOR", SqlDbType.Int);
                parms[1].Value = intCodigoAsesor;
                parms[2] = new SqlParameter("@i_NOMBRE_SUPERVISOR", SqlDbType.VarChar,100);
                parms[2].Value = strNombreSupervisor;
                parms[3] = new SqlParameter("@i_USUARIO_ASESOR", SqlDbType.VarChar,100);
                parms[3].Value = strUsrasesor;
                parms[4] = new SqlParameter("@i_NOMBRE_ASESOR", SqlDbType.VarChar,100);
                parms[4].Value = Nmbrasesor;
                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_ACTUALIZACION_ASESOR", parms);
            }
            catch (Exception ex)
            {
                throw (ex);
            }
            finally
            {
                if (objAccesoDatos != null)
                    objAccesoDatos = null;
            }
```

```
    }
```

CAPA DEL NEGOCIO BC:

```
/// <summary>
    /// método de la capa de logica que se encarga de
    /// grabar los asesores por usuario
    /// </summary>
    /// <param name="strModo">modo</param>
    /// <param name="strCodigoFamilia">codigo del asesor</param>
    /// <param name="strUsuario">codigo usuario</param>
    /// <param name="strUsuarioAsignacion">usuario asignado</param>
    public void GrabaAsesorUsuario(string strModo, string strCodigoAsesor, string strUsuario, string strUsuarioAsignacion)
    {
       try
       {
          VR_AsesoresDC objAsesoresDC = new VR_AsesoresDC();
          objAsesoresDC.GrabaAsesoresUsuario(strModo, strCodigoAsesor, strUsuario, strUsuarioAsignacion);
       }
       catch (Exception ex)
       {
          throw (ex);
       }
    }


    /// <summary>
    /// Método de la capa de logica que graba los registros de asesores
    /// </summary>
    /// <param name="strModo">modo grabacion, I = nuevo, M = modificacion</param>
    /// <param name="strCodigoFamilia">codigo del elemento a ser guardado</param>
    /// <param name="strDescripcion">descripcion del elemento a ser guardado</param>
    public void GrabaAsesor(string strModo, int intCodigoAsesor, string strNombreSupervisor, string strUsrasesor, string
Nmbrasesor)
    {
       try
       {
```

```
            VR_AsesoresDC objAsesoresDC = new VR_AsesoresDC();
            objAsesoresDC.GrabaAsesor(strModo, intCodigoAsesor, strNombreSupervisor, strUsrasesor, Nmbrasesor);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
    }
```

CAPA DE INTERFACE DE USUARIO GUI:

```
protected void wbtGrabar_Click(object sender, EventArgs e)
{
    try
    {
        if (txtNombre.Text.Length == 0)
        {
            MessageBox("Ingrese el nombre", txtNombre);
            return;
        }
        if (txtCodigo.Text.Length == 0)
        {
            MessageBox("Ingrese el código", txtCodigo);
            return;
        }
        string strModo = "";

        if (blnNuevo)
        {
            strModo = "I";
        }
        else
        {
            strModo = "M";
        }
```

```
        if (blnNuevo)
        {
            for (int i = 0; i < dttAsesor.Rows.Count; i++)
            {
                if (txtCodigo.Text == dttAsesor.Rows[i][0].ToString())
                {
                    MessageBox("Ya existe un otro registro con este código", txtNombre);
                    return;
                }
            }
        }
        VR_AsesoresBC objDatos = new VR_AsesoresBC();
        objDatos.GrabaAsesor(strModo, Convert.ToInt32(txtCodigo.Text), ddlSupervisor.Text, txtUsuario.Text.ToLower(),
txtNombre.Text.ToUpper());

        MessageBox("El Registro se grabó exitosamente");
        ConsultaAsesores();
      }
      catch (System.Exception ex)
      {
        MessageError(ex.Message);
      }
    }
```

3.5.3 Eliminar verificadores

Esta es la última funcionalidad del módulo de verificadores, esta funcionalidad lo que nos permite hacer es eliminar la información almacenada de cualquier verificador, a continuación el código de esta funcionalidad:

CAPA DE DATOS DC:

```
/// <summary>
    /// Método de la capa de datos que se encarga de eliminar los datos de
    /// los asesores filtrados por usuario
    /// </summary>
    /// <param name="strUsuario">codigo de usuario</param>
    /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
    public void EliminaAsesoresUsuario(string strUsuario, string strUsuarioAsignacion)
```

```
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[2];
                parms[0] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 8);
                parms[0].Value = strUsuario;
                parms[1] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
                parms[1].Value = strUsuarioAsignacion;
                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_BORRA_ASESORES_USUARIOS", parms);
            }
            catch (Exception ex)
            {
                throw (ex);
            }
            finally
            {
                if (objAccesoDatos != null)
                    objAccesoDatos = null;
            }
        }
        /// <summary>
        /// Método de la capa de datos que se encarga de eliminar los datos de
        /// los asesores
        /// </summary>
        /// <param name="strFamilia">codigo de usuario</param>
        /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
        public void EliminaAsesores(int intAsesor, string strUsuarioAsignacion)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[2];
                parms[0] = new SqlParameter("@i_CODIGO_ASESOR", SqlDbType.Int);
```

```
            parms[0].Value = intAsesor;
            parms[1] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 8);
            parms[1].Value = strUsuarioAsignacion;
            objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure, "PRO_VR_BORRA_ASESOR",
parms);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
        finally
        {
            if (objAccesoDatos != null)
                objAccesoDatos = null;
        }
    }
```

CAPA DEL NEGOCIO BC:

```
/// <summary>
    /// Método de la capa de logica que se encarga de eliminar los datos de
    /// los asesores filtrados pro usuario
    /// </summary>
    /// <param name="strFamilia">codigo de usuario</param>
    /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>

    public void EliminaAsesoresUsuario(string strUsuario, string strUsuarioAsignacion)
    {
        try
        {
            VR_AsesoresDC objAsesoresDC = new VR_AsesoresDC();
            objAsesoresDC.EliminaAsesoresUsuario(strUsuario, strUsuarioAsignacion);
        }
        catch (Exception ex)
        {
            throw (ex);
```

```
        }
    }

    /// <summary>
    /// Método de la capa de logica que se encarga de llamar a la capa de datos para eliminar
    /// los asesores
    /// </summary>
    /// <param name="strUsuario">codigo de usuario</param>
    /// <param name="strUsuarioAsignacion">codigo de usuario asignación</param>
    public void EliminaAsesores(int intAsesor, string strUsuarioAsignacion)
    {
        try
        {
            VR_AsesoresDC objAsesoresDC = new VR_AsesoresDC();
            objAsesoresDC.EliminaAsesores(intAsesor, strUsuarioAsignacion);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
    }
```

CAPA DE INTERFACE DE USUARIO GUI:

```
protected void btnEliminar_Click(object sender, EventArgs e)
  {
    try
    {
      VR_AsesoresBC objDatos = new VR_AsesoresBC();
      if (txtCodigo.Text.Trim() == "")
      {
        MessageBox("Seleccione un registro para eliminar", txtNombre);
        return;
      }
      objDatos.EliminaAsesores(Convert.ToInt32(txtCodigo.Text), strUsuarioAPP[0]);
      MessageBox("El Registro se eliminó exitosamente");
```

```
        ConsultaAsesores();

        limpiarDatos();



        blnNuevo = true;

    }

    catch (System.Exception ex)

    {

        MessageError(ex.Message);

    }

}
```

3.5.4 Interfaces de usuario

Una vez que se tienen implementadas las tres capas de la arquitectura de desarrollo, se procede a la implementación de la interface de usuario, al igual que para las demás administraciones, solo se pondrá una parte del código de la interface de usuario, el resto del código, se encuentra en el anexo 3

```
<table align="center" border="1" cellpadding="1" designtimedragdrop="304" style="z-index: 104;
    left: 10px; width: 457px; position: absolute; top: 327px; height: 32px">
    <tr align="center" class="fconsult">
        <td style="width: 468px; height: 26px">
            <asp:Button ID="btnNuevo" runat="server" CausesValidation="False" OnClick="btnNuevo_Click"
                TabIndex="17" Text="Nuevo" ToolTip="Nuevo Registro" Width="100px" /><asp:Button ID="wbtGrabar"
                    runat="server" OnClick="wbtGrabar_Click" TabIndex="17" Text="Grabar" ToolTip="Grabar Datos del Registro"
                    Width="100px" /><asp:Button ID="btnEliminar" runat="server" OnClick="btnEliminar_Click"
                        TabIndex="17" Text="Eliminar" ToolTip="Elimina un registro" Width="100px" />

                        </td>
    </tr>
</table>
```

Esta sección de código pertenece a los botones de la interface, a continuación el resultado de la implementación de verificadores:

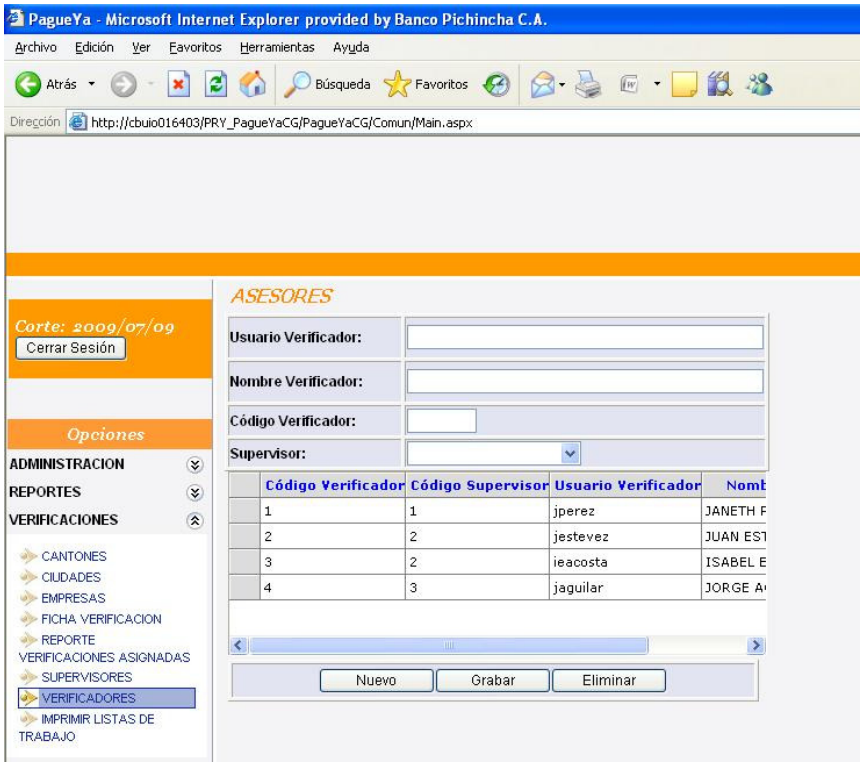


Figura 3.35: Resultado de la implementación del módulo de administración de verificadores

3.5.5 Pruebas unitarias

Al igual que en los módulos anteriores, una vez que se ha termiando la implementación se procede a las pruebas unitarias, la primera prueba que se realizará será el ingreso de un verificador:

INGRESANDO UN VERIFICADOR:

Para el ingreso de un nuevo verificador, se debe tomar en cuenta que es necesario asociarlo con un supervisor ya que es el supervisor quien le asigna las tareas para trabajar.

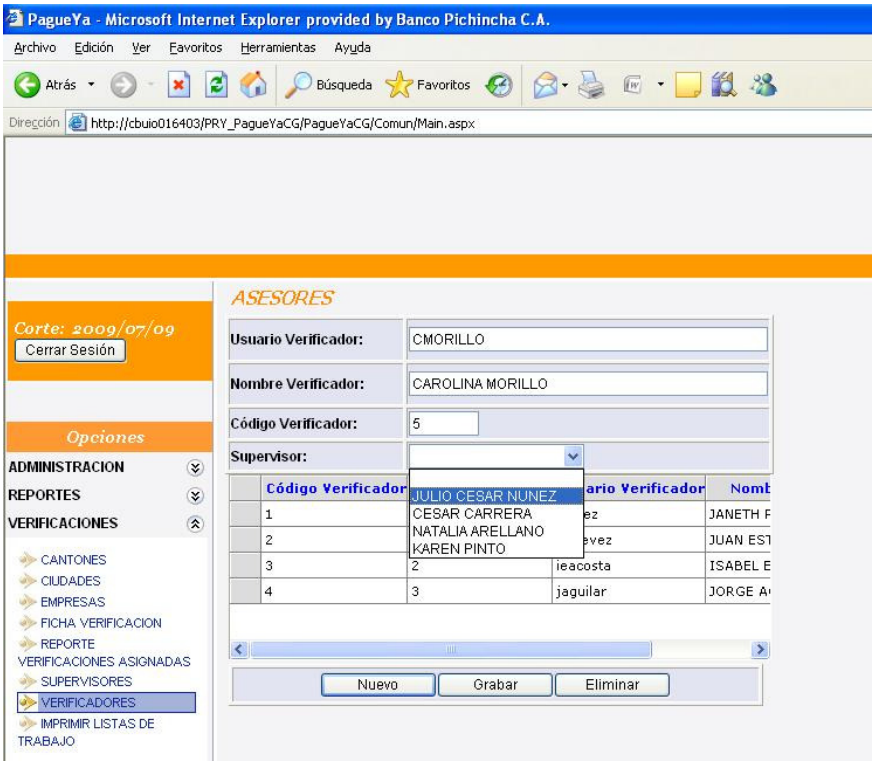


Figura 3.36: Insertando un nuevo verificador

Como podemos observar, para ingresar los datos del verificador, debemos escoger a un supervisor de la lista desplegable.

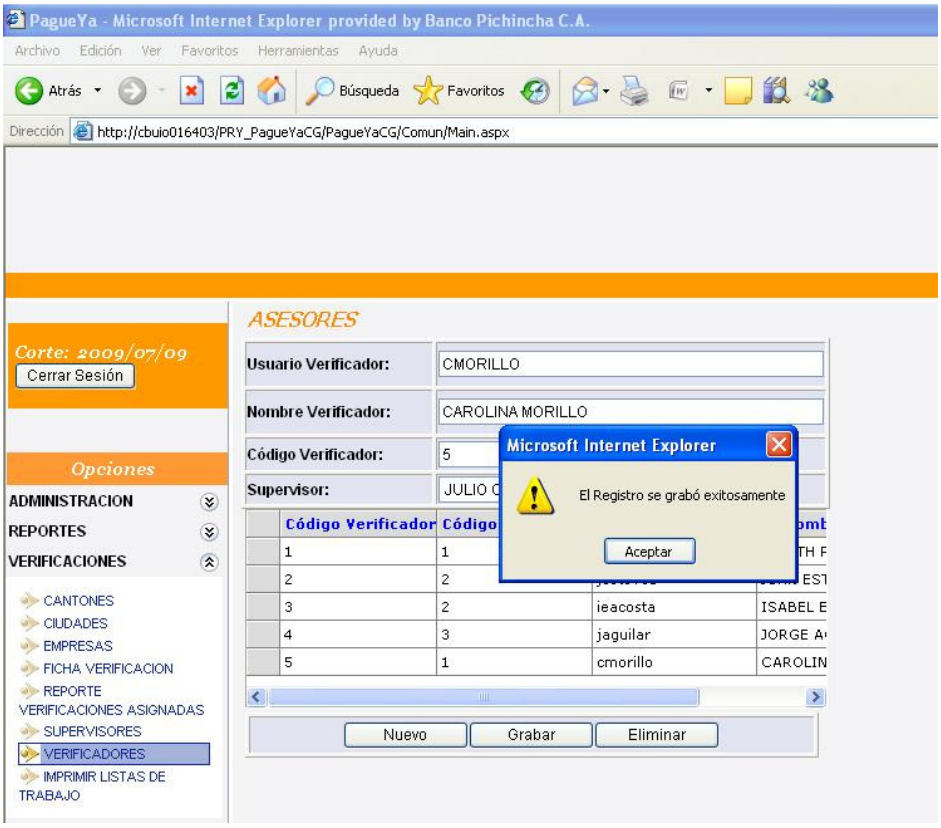


Figura 3.37: El sistema informa al usuario un ingreso de datos exisitoso

Una vez que se ha seleccionado un su´pervisor de la lista y se ha presionado el botón de grabar, se presenta un mensaje de confirmación de ingreso de datos exitoso.

ACTUALIZANDO UN VERIFICADOR:

Con información de verificardores en la base de datos, se pueden realizar modificaciones sobre la información almacenada, para esto basta con seleccionar un verificador de la lista y modificar la información que se presente al usuario, en este caso, se va a actualizar al nombre del supervisor:

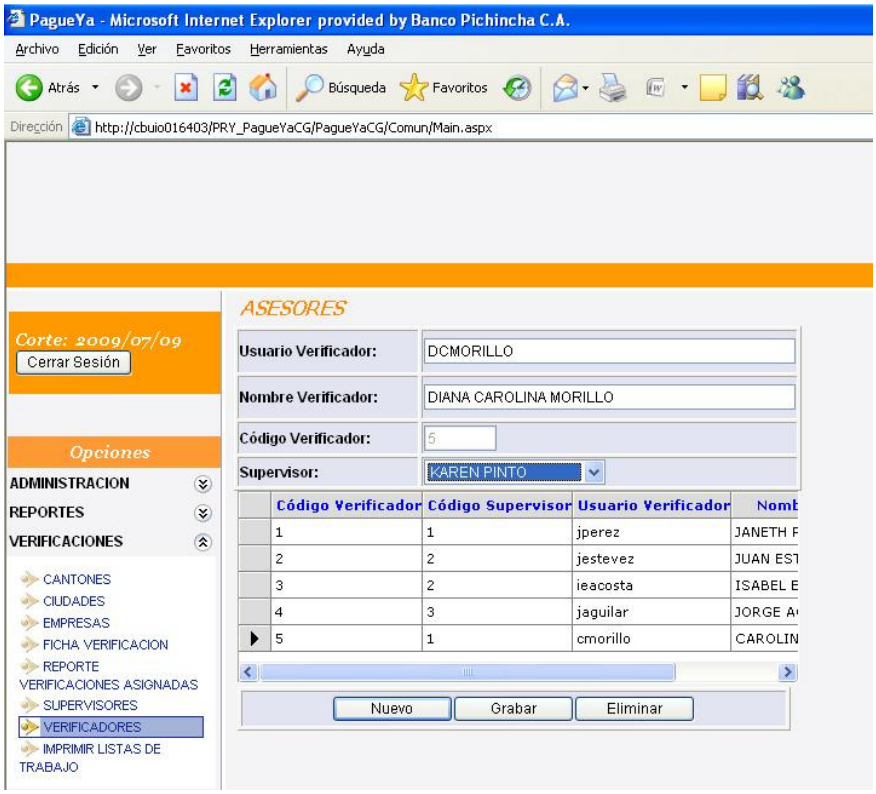


Figura 3.38: Actualizando los datos de un verificador

Al igual que en los módulos anteriores, en la actualización de datos se pueden modificar los datos de un verificador a excepción del código de verificador ya que este es un código para el manejo interno de la base de datos.

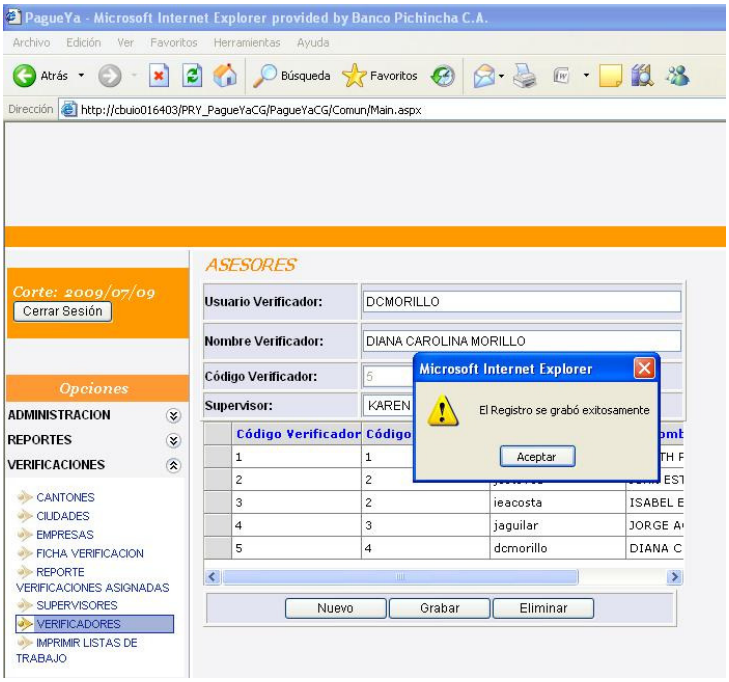


Figura 3.39: El sistema informa al usuario que los datos sa han actualizado correctamente

ELIMINANDO UN VERIFICADOR:

La última prueba que se realiza sobre el módulo de verificadores es la de eliminar un registro de la base de datos, después de seleccionar a un verificador y presionar el botón de eliminación, el sistema pregunta al usuario si desea eliminar el registro:

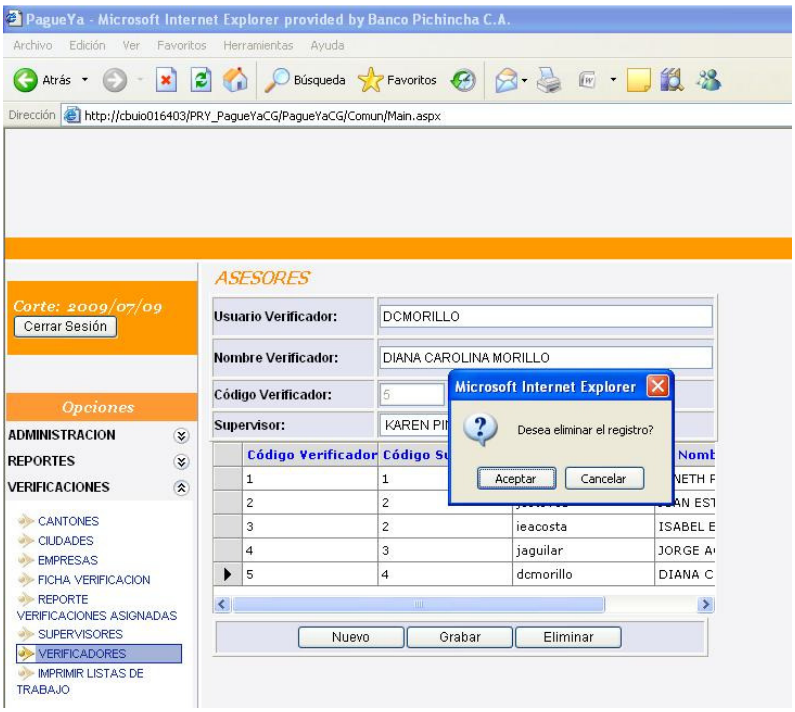


Figura 3.40: Solicitud de confirmación por parte del sistema ántes de eliminar el registro

Después de la confirmación del usuario, el sistema elimina al usuario seleccionado:

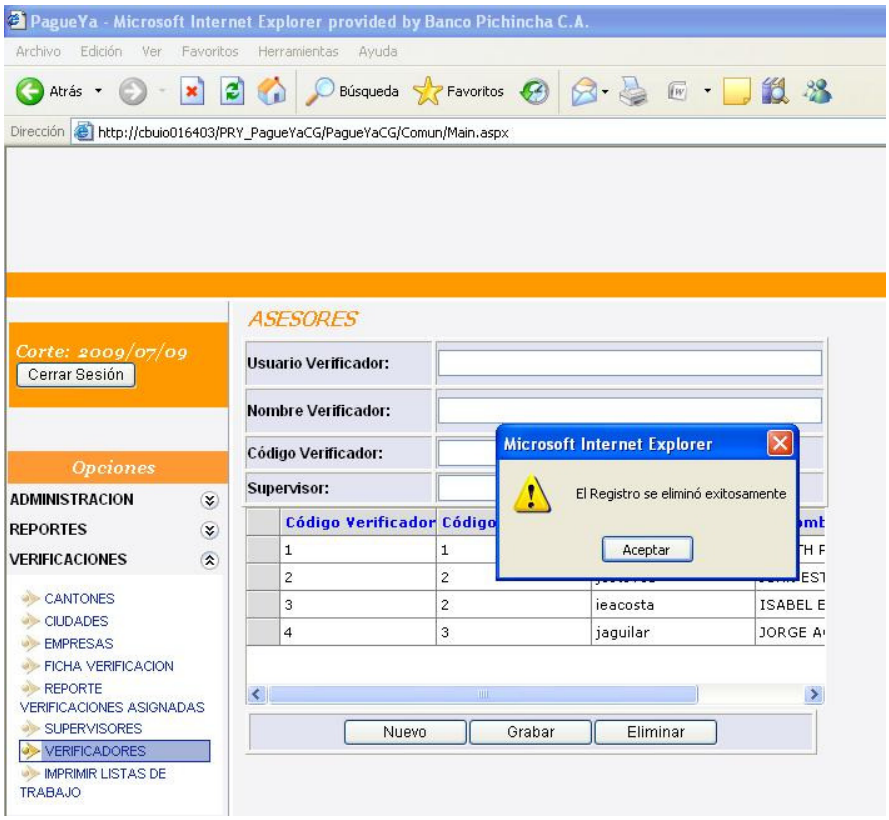


Figura 3.41: El sistema confirma al usuario que el registro se ha eliminado correctamente.

Con los verificadores y supervisores implementados, se puede realizar la implementación de la administración de los perfiles de usuario:

Este módulo nos permite definir los permisos que tendrá cada uno de los usuarios del sistema

3.6 Administración de perfiles de usuario

3.6.1 Crear un nuevo perfil (administrador, supervisor, verificador)

Esta es la primera de las funcionalidades del módulo de usuarios, nos permite ingresar los datos de los usuarios del sistema así como también los permisos que tendrá, ya sea como administrador, supervisor o verificador, la diferencia entre los perfiles radica en que, dependiendo del perfil, podrá ver ciertas pantallas del sistema.

CAPA DE DATOS DC:

```
/// <summary>
/// Método de la capa de datos que permite grabar información
/// de los usuarios
/// </summary>
/// <param name="strModo">codigo del modo de grabacion</param>
```

```
        /// <param name="strUsuario">usuario a se grabado</param>
        /// <param name="strEmail">email del usuario</param>
        /// <param name="strActivo">indicador de actividad del usuario</param>
        /// <param name="strPerfil">perfil del usuario</param>
        public void GrabarUsuario(string strModo, string strUsuario, string strEmail, string strActivo, string strPerfil,string strClave,
            string strNombre, int strCodPerfil, int strCodCiudad)
        {
            AccesoDatos objAccesoDatos = new AccesoDatos();
            try
            {
                SqlParameter[] parms = new SqlParameter[9];
                parms[0] = new SqlParameter("@i_MODO", SqlDbType.Char, 1);
                parms[0].Value = strModo;
                parms[1] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 8);
                parms[1].Value = strUsuario;
                parms[2] = new SqlParameter("@i_MAIL", SqlDbType.VarChar, 80);
                parms[2].Value = strEmail;
                parms[3] = new SqlParameter("@i_ACTIVO", SqlDbType.Char, 1);
                parms[3].Value = strActivo;
                parms[4] = new SqlParameter("@i_PERFIL", SqlDbType.Char, 1);
                parms[4].Value = strPerfil;
                parms[5]=new SqlParameter("@i_CLAVE", SqlDbType.VarChar,5);
                parms[5].Value = strClave;
                parms[6] = new SqlParameter("@i_NOMBRE", SqlDbType.VarChar, 80);
                parms[6].Value = strNombre;
                parms[7] = new SqlParameter("@i_COD_PERFIL", SqlDbType.Int);
                parms[7].Value = strCodPerfil;
                parms[8] = new SqlParameter("@i_COD_CIUDAD", SqlDbType.Int);
                parms[8].Value = strCodCiudad;

                objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_CG_ACTUALIZACION_USUARIOS", parms);
            }
            catch (Exception ex)
            {
```

```
            throw (ex);
        }
        finally
        {
            if (objAccesoDatos != null)
                objAccesoDatos = null;
        }
    }
```

Como podemos observar, en esta capa se realiza la comunicación con la base de datos por medio de métodos que realizan esta tarea.

CAPA DEL NEGOCIO BC

```
    /// <summary>
    /// Método de la capa de logica que permite grabar información
    /// de los usuarios
    /// </summary>
    /// <param name="strModo">codigo del modo de grabacion</param>
    /// <param name="strUsuario">usuario a se grabado</param>
    /// <param name="strEmail">email del usuario</param>
    /// <param name="strActivo">indicador de actividad del usuario</param>
    /// <param name="strPerfil">perfil del usuario</param>
    public void GrabarUsuario(string strModo, string strUsuario, string strEmail, string strActivo, string strPerfil,string strClave,
        string strNombre, int strCodPerfil, int strCodCiudad)
    {
        try
        {
            UsuariosDC objUsuariosDC = new UsuariosDC();
            objUsuariosDC.GrabarUsuario(strModo, strUsuario, strEmail, strActivo, strPerfil, strClave, strNombre, strCodPerfil,
strCodCiudad);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
    }
```

Esta es la capa que sirve como comunicación entre la capa de datos y la interface de usuario, esta capa recibe los parámetros de que el usuario ingresa por medio de la interface y los pasa a la capa de datos para que esta a su vez realice las consultas a la base de datos y retorne los resultados de dichas consultas.

CAPA DE INTERFACE DE USUARIO GUI

```
protected void wbtGrabar_Click(object sender, EventArgs e)
  {
    try
    {
      if (txtUsuario.Text.Length == 0)
      {
        MessageBox("Ingrese el Usuario", txtUsuario);
        return;
      }

      string strModo = "";
      string strActivo = "";
      string strPerfil = "";

      if (blnNuevo)
        strModo = "I";
      else
        strModo = "M";

      if (chkActivo.Checked)
        strActivo = "1";
      else
        strActivo = "0";

      if (chkAdministrador.Checked)
        strPerfil = "A";
      else
        strPerfil = "U";
```

```
        if (blnNuevo && txtClave.Text.Length == 0)
        {
            MessageBox("Debe Ingresar un valor para la Clave");
        }

        if (txtNombreUsuario.Text.Length == 0)
        {
            MessageBox("Debe Ingresar un valor para el Nombre del Usuario");
        }

        UsuariosBC objDatos = new UsuariosBC();

        objDatos.GrabarUsuario(strModo, txtUsuario.Text, txtEmail.Text, strActivo, strPerfil, txtClave.Text,
            txtNombreUsuario.Text, Convert.ToInt16(ddlPerfil.SelectedValue.ToString()),
Convert.ToInt16(ddlCiudad.SelectedValue.ToString()));

        MessageBox("El Registro se grabó exitosamente");
        ConsultarUsuarios(string.Empty);
        limpiarDatos();
        blnNuevo = true;
      }
      catch (System.Exception ex)
      {
        MessageError(ex.Message);
      }
  }
```

Este es el código de la interface de usuario, es en esta capa donde se reciben los parámetros que el usuario ingresa para crear un nuevo perfil de usuario.

3.6.2 Eliminar usuarios

Esta es la última funcionalidad del módulo de perfiles de usuario, y nos permite eliminar la información almacenada de cualquier usuario del sistema, a continuación el código de esta funcionalidad:

CAPA DE DATOS DC:

```
/// <summary>
/// Método de la capa de datos que se encarga de eliminar registros de
/// destino contable por usuario
/// </summary>
/// <param name="strUsuario">id del usuario</param>
/// <param name="strUsuarioAsignacion">clave de asignación</param>
public void EliminaDestinoContableUsuario(string strUsuario, string strUsuarioAsignacion)
{
    AccesoDatos objAccesoDatos = new AccesoDatos();
    try
    {
        SqlParameter[] parms = new SqlParameter[2];
        parms[0] = new SqlParameter("@i_CODIGO_USUARIO", SqlDbType.VarChar, 10);
        parms[0].Value = strUsuario;
        parms[1] = new SqlParameter("@i_USUARIO_ASIGNACION", SqlDbType.VarChar, 10);
        parms[1].Value = strUsuarioAsignacion;
        objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_CG_BORRA_DESTINO_CONTABLE_USUARIOS", parms);
    }
    catch (Exception ex)
    {
        throw (ex);
    }
    finally
    {
        if (objAccesoDatos != null)
            objAccesoDatos = null;
    }
}
```

CAPA DEL NEGOCIO BC:

```
/// <summary>
/// Método de la capa de logica que se encarga de eliminar registros de
/// destino contable por usuario
```

```
    /// </summary>
    /// <param name="strUsuario">id del usuario</param>
    /// <param name="strUsuarioAsignacion">clave de asignación</param>
    public void EliminaDestinoContableUsuario(string strUsuario, string strUsuarioAsignacion)
    {
        try
        {
            UsuariosDC objUsuariosDC = new UsuariosDC();
            objUsuariosDC.EliminaDestinoContableUsuario(strUsuario, strUsuarioAsignacion);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
    }
```

CAPA DE INTERFACE DE USUARIO GUI:

```
protected void wbtEliminar_Click(object sender, EventArgs e)
   {
      try
      {
         if (!chkEliminar.Checked)
         {
            MessageBox("Aceptar la Eliminación");
            chkEliminar.Visible = true;
            return;
         }

         if (txtUsuario.Text.Length != 0)
         {
            string strModo = "D";
            UsuariosBC objDatos = new UsuariosBC();

            objDatos.GrabarUsuario(strModo, txtUsuario.Text, txtEmail.Text, string.Empty, string.Empty, txtClave.Text,
```

```
                txtNombreUsuario.Text, Convert.ToInt16(ddlPerfil.SelectedValue.ToString()),
Convert.ToInt16(ddlCiudad.SelectedValue.ToString()));


            MessageBox("El Registro se grabó exitosamente");

            ConsultarUsuarios(string.Empty);

          }

          else

            MessageBox("Debe Escoger un Usuario a ser eliminado");

        }

        catch (Exception ex)

        {

          MessageError(ex.Message);

        }

    }
```

3.6.3 Interfaces de usuario

Una vez que se tienen implementadas las tres capas de la arquitectura de desarrollo, se procede a la implementación de la interface de usuario, al igual que para las demás administraciones, solo se pondrá una parte del código de la interface de usuario, el resto del código, se encuentra en el anexo 3

```
<table id="Table1" align="center" border="1" cellpadding="1" style="width: 777px; height: 33px">

        <tr class="fconsult">

          <td style="width: 221px; height: 25px">

            <strong> Usua</strong>r<strong>io:  </strong>

            <asp:TextBox ID="txtUsuario" runat="server" MaxLength="8"
Width="82px"></asp:TextBox><strong></strong><asp:ImageButton ID="imbBuscar" runat="server"
ImageUrl="..\Imagenes\search.gif"

              ToolTip="Buscar por Usuario" OnClick="imbBuscar_Click" />

            <asp:LinkButton ID="lkbRefrescar" runat="server" Enabled="False" OnClick="lkbRefrescar_Click"

              ToolTip="Refresca los datos de las busquedas">Refrescar</asp:LinkButton></td>

          <td style="width: 193px; height: 25px">

            <strong>Email: </strong>

            <asp:TextBox ID="txtEmail" runat="server" MaxLength="25" Width="146px"></asp:TextBox></td>

          <td style="height: 25px; text-align: left; width: 163px;">

            <strong>

              <asp:CheckBox ID="chkAdministrador" runat="server" Text="Administrador" />

            </strong><strong>
```

```
                <asp:CheckBox ID="chkActivo" runat="server" Text="Activo" />

                <asp:CheckBox ID="chkEliminar" runat="server" Text="Eliminar Usuario" /></strong></td>

            <td style="height: 25px; text-align: left; width: 135px;"><strong>Clave<asp:TextBox ID="txtClave" runat="server"
MaxLength="5" TextMode="Password"

                ToolTip="Ingrese una contraseña de 5 números" Width="52px"></asp:TextBox></strong></td>

        </tr>

        <tr class="fconsult">

          <td colspan="2" style="height: 25px">

            <strong>Nombre: </strong>

            <asp:TextBox ID="txtNombreUsuario" runat="server" MaxLength="80" Width="358px"></asp:TextBox></td>

          <td style="width: 163px; height: 25px; text-align: left">

             <strong>Ciudad: <asp:DropDownList ID="ddlCiudad" runat="server" Width="113px">

            </asp:DropDownList></strong></td>

          <td style="width: 135px; height: 25px; text-align: left">

            <strong>Perfil: </strong><asp:DropDownList ID="ddlPerfil" runat="server" Width="98px">

            </asp:DropDownList></td>

        </tr>

      </table>
```

Esta seción de código pertenece a los botones de la interface, a continuación el resultado de la implementación de usuarios:

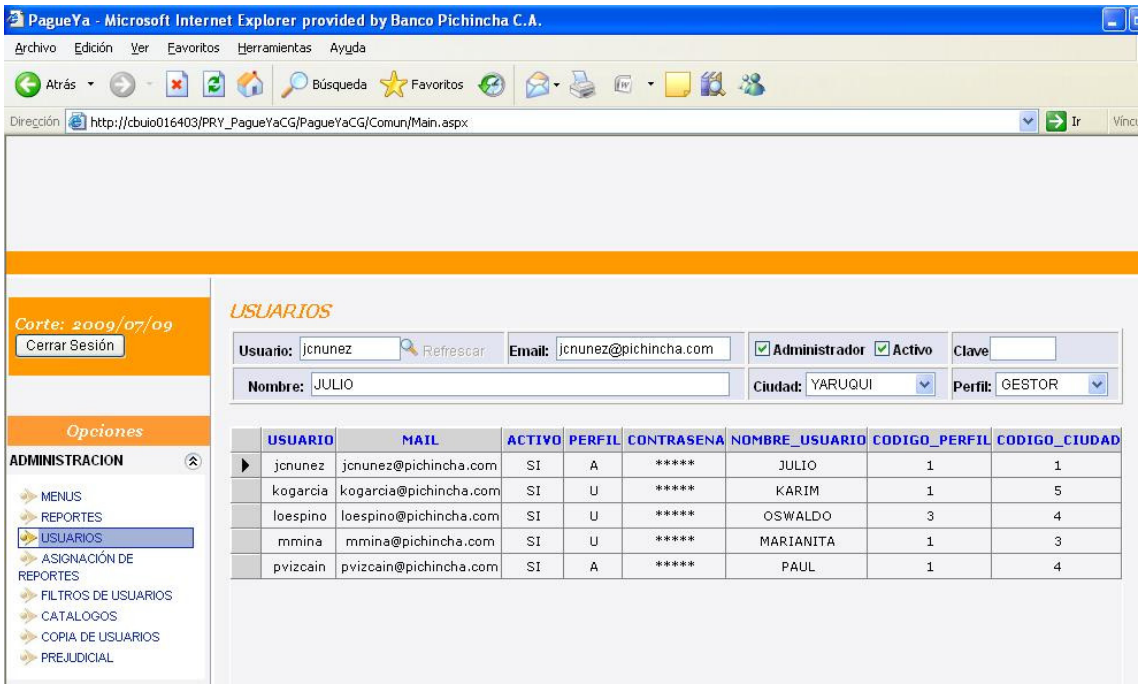


Figura 3.42: Resultado de la implementación del módulo perfiles de usuario

3.6.4 Pruebas unitarias

INGRESANDO UN NUEVO PERFIL DE USUARIO:

La primera de las pruebas que se realizará sobre este módulo, es el ingreso de un nuevo perfil de usuario:

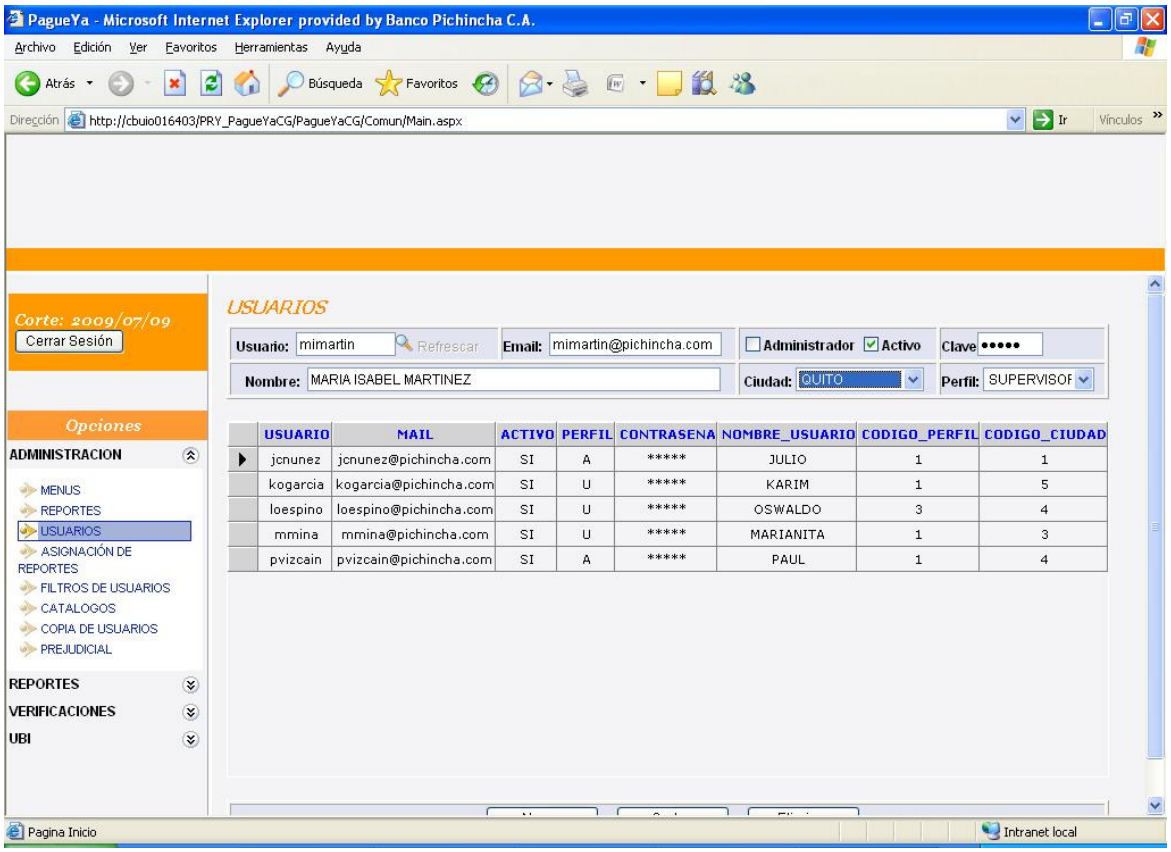


Figura 3.43: Ingresando los datos de un nuevo perfil de usuario

Después de haber ingresado los datos y presionado el botón de grabar, el sistema nos presenta un mensaje de ingreso de datos exisotos:

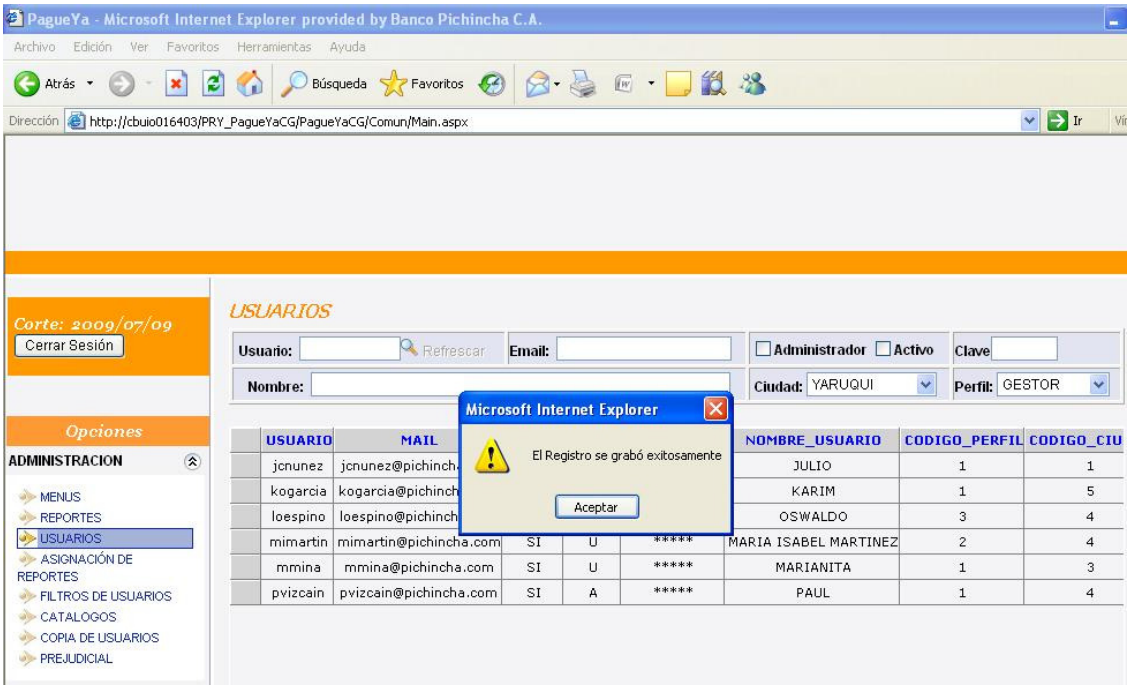


Figura 3.44: Confirmación del sistema de ingreso de datos exitoso

ELIMINANDO UN PERFIL DE USUARIO:

Esta es la últma prueba que se realiza sobre el módulo de perfiles de usuario, se selecciona un usuario de la lista y se presiona el botón de eliminar y aparece un check para confirmar la eliminación, si el check no ha sido seleccionado, se presenta un mensaje que indica al usuario que debe seleccionar la opción de eliminar:

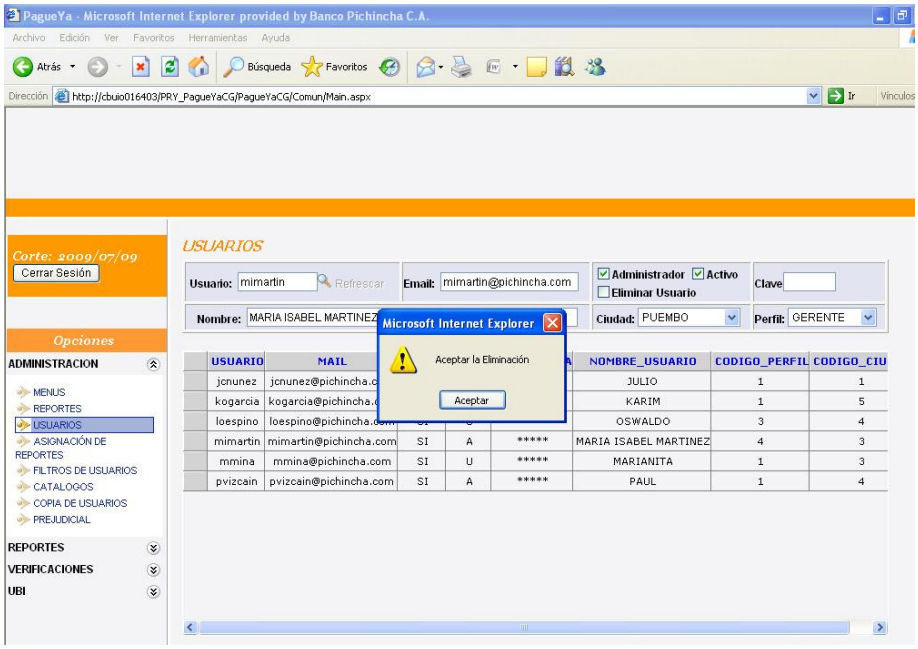


Figura 3.45: Solicitud de confirmación del sistema para eliminar un registro

Después de haber confirmado la eliminación, el usuario se elimina de la base de datos.

Para la generación de las fichas de verificación es necesario que las administraciones estén implementadas ya que los campos que se cargan por defecto en la ficha se toman de la información guardada en la base de datos:

3.7 Generación de fichas de verificación

El sistema permite dos formas de generar las fichas de verificación, se puede generar una sola ficha o se puede generar toda la lista de trabajo, para realizar esta funcionalidad, fue necesario crear las clases DC, BC y GUI, a continuación se presenta el código de las diferentes capas:

3.7.1 Generación de fichas de verificación

CAPA DE DATOS DC:

```
/// <summary>
/// Clase de datos que maneja la información de las ciudades
/// </summary>
public class VR_VerificacionesDC
{
    #region Consultas
    /// <summary>
    /// Método de la capa de datos que se encarga de consultar
    /// los datos de las familias
    /// </summary>
    /// <returns>Objeto de retorno</returns>
    public Retorno ConsultarVerificaciones(string strUsuario)
    {
        AccesoDatos objAccesoDatos = new AccesoDatos();
        try
        {
            SqlParameter[] parms = new SqlParameter[1];
            parms[0] = new SqlParameter("@i_USUARIO", SqlDbType.VarChar, 100);
            parms[0].Value = strUsuario;
            return objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_CONSULTAR_VERIFICACIONES_FILTRADAS_POR_SUPERVISOR",parms);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
        finally
```

```
        {
            if (objAccesoDatos != null)
                objAccesoDatos = null;
        }
    }
```

CAPA DEL NEGOCIO BC:

```
/// <summary>
    /// Metodo de la capa de logica del negocio que consulta los
    /// registros de las ciudades
    /// </summary>
    /// <returns>Dataset con los datos</returns>
    public DataSet ConsultarVerificaciones(string strUsuario)
    {
        try
        {
            VR_VerificacionesDC objVerificacionesDC = new VR_VerificacionesDC();
            return objVerificacionesDC.ConsultarVerificaciones(strUsuario).Dataset;
        }
        catch (Exception ex)
        {
            throw (ex);
        }
    }
```

CAPA DE INTERFACE DE USUARIO GUI:

```
        <table border="1" cellpadding="1"style="z-index: 102;
            left: 88px; width: 471px; position: absolute; top: 1110px; height: 32px">
            <tr align="center" class="fconsult">
                <td style="width: 462px; height: 26px">
                    <asp:Button ID="btnNuevo" runat="server" CausesValidation="False" OnClick="btnNuevo_Click"
                        TabIndex="17" Text="Nuevo" ToolTip="Nuevo Registro" Width="86px" /><asp:Button ID="wbtGrabar"
                            runat="server" OnClick="wbtGrabar_Click" TabIndex="17" Text="Grabar" ToolTip="Grabar Datos del
Registro"
                            Width="94px" /><asp:Button ID="btnEliminar" runat="server" OnClick="btnEliminar_Click"
                                TabIndex="17" Text="Eliminar" ToolTip="Elimina un registro" Width="94px" /><br />
```

```
            <br />
            <input type="button" value="Imprimir Una Ficha" onclick="imprimir();" style="width: 116px" name="btnImprimir"
/>
            <asp:Button ID="Button1" runat="server" Text="Exportar Datos" OnClick="btnImprimirTodo_Click" /></td>
        </tr>
    </table>
```

3.7.2 Pruebas unitarias

A continuación se presenta la antalla del módulo de fichas como resultado de la implementación, las demás pruebas se ralizarán al momento de integrar el sistema:

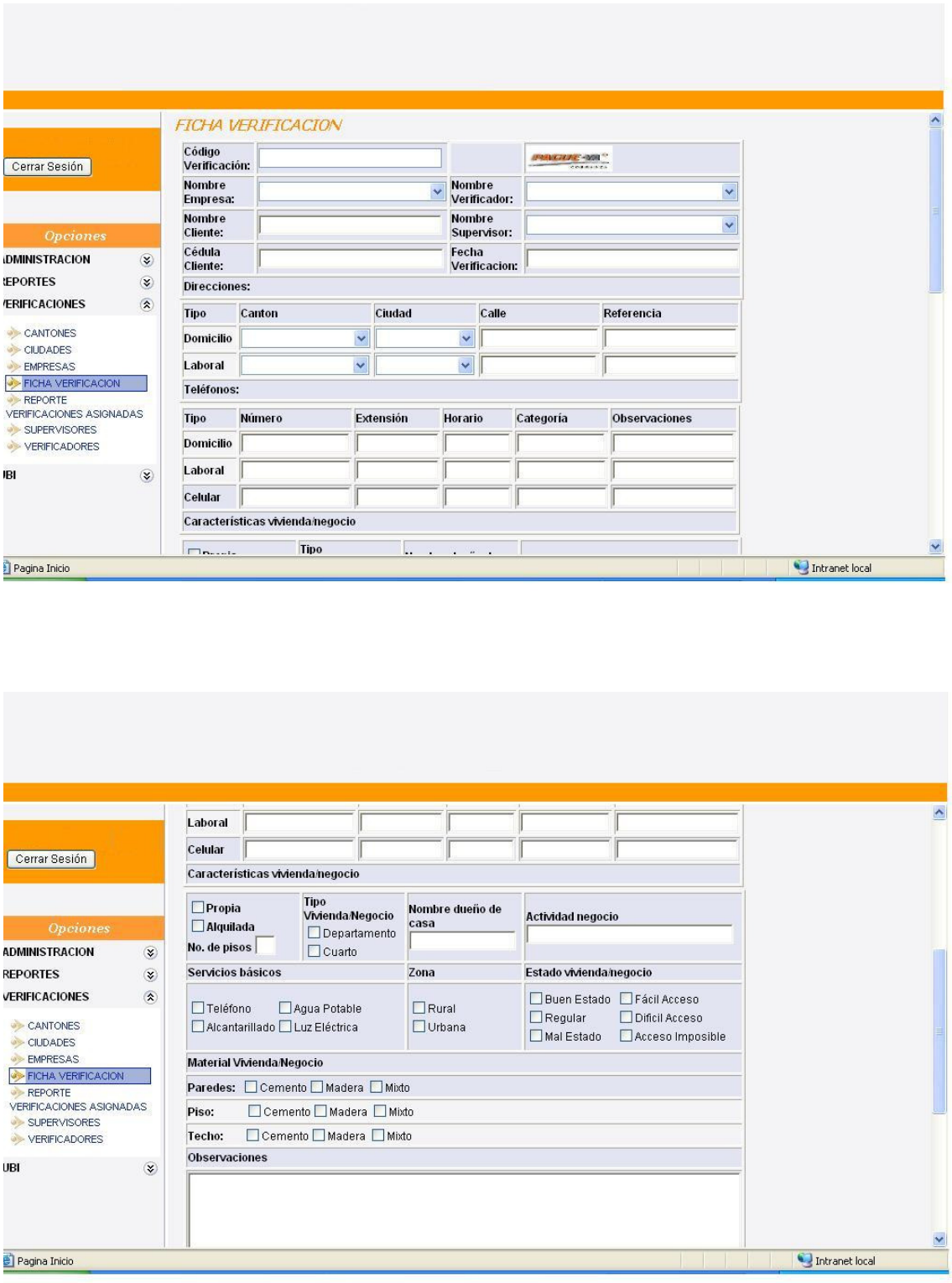


Figura 3.46: Pantalla del módulo de fichas de verificación

Al igual que la generación de fichas, la generación de listas de trabajo necesita también que se tengan implementadas todas las administraciones, la primera prueba que se realizará será imprimir una de las fichas:

IMPRIMIENDO UNA FICHA:

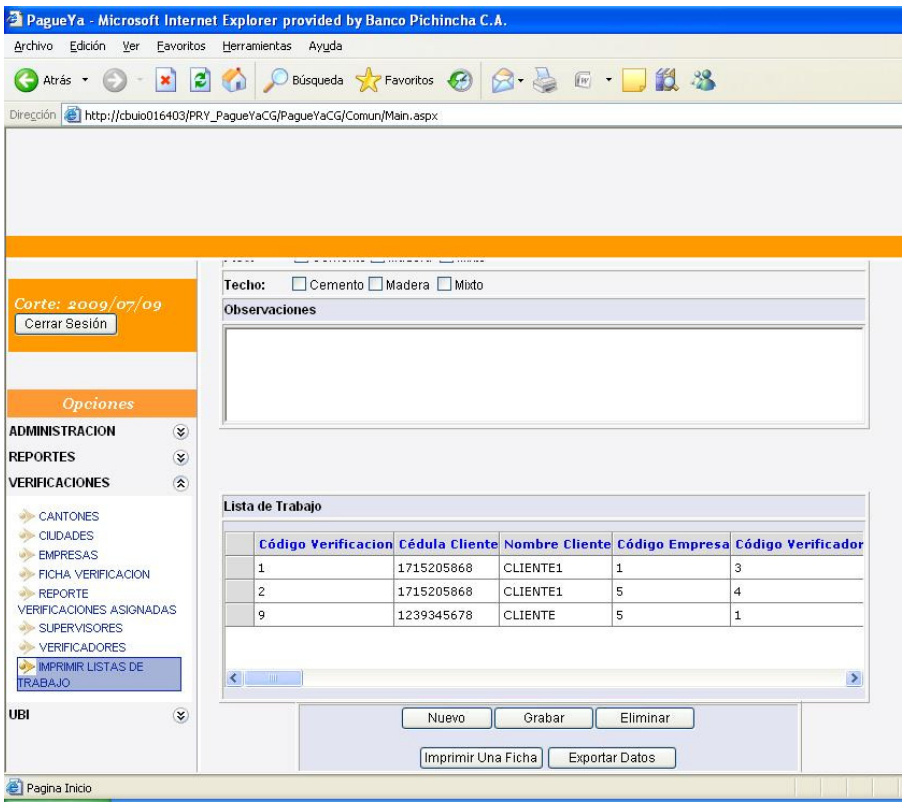


Figura 3.47: Pantalla para impresión de fichas

Cuando se presiona el botón para imrpimir una ficha, se presenta la siguiente pantalla:

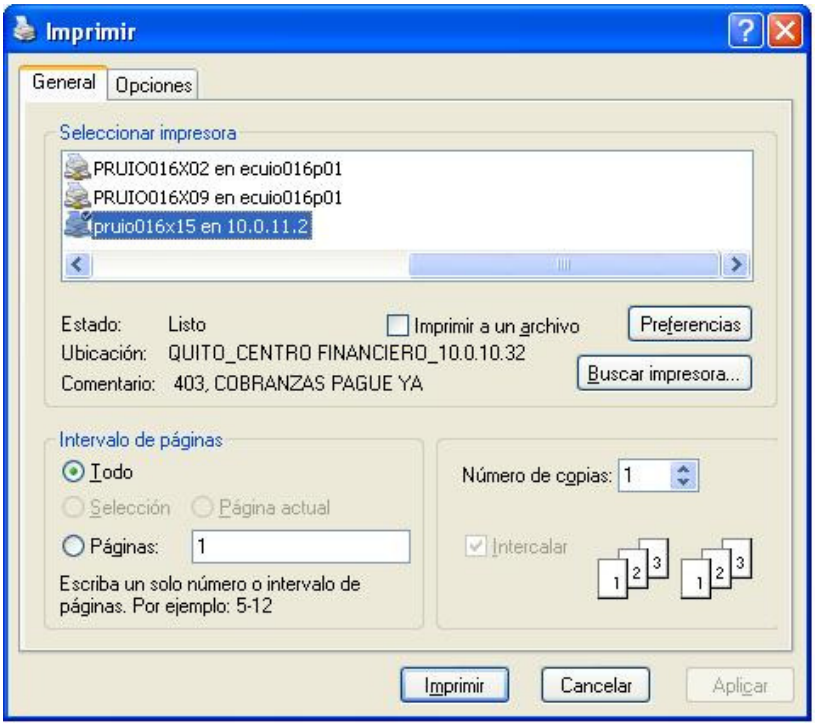


Figura 3.48: Pantalla para la selección de la impresora e eimprimir la ficha seleccioinada

IMPRIMIENDO VARIAS FICHAS:

Esra es la segunda funcionalidad que permite el sistema, sirve para imprimir toda la lista de trabajo para repartirlas a los verificadores, al presionar el botón de imprimir varias fichas, se presenta la siguiente pantalla:

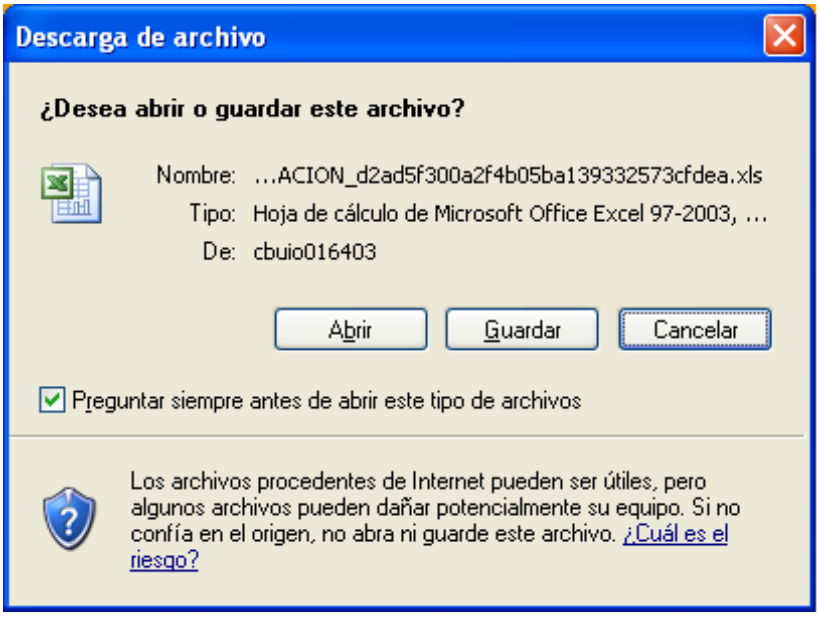


Figura 3.49: Pantalla para imprimir la lista de trabajo

Al presionar el botón de abrir, se abre un archivo Excel que presenta un botón para imrpimir toda la base de datos de las fichas de verificación:

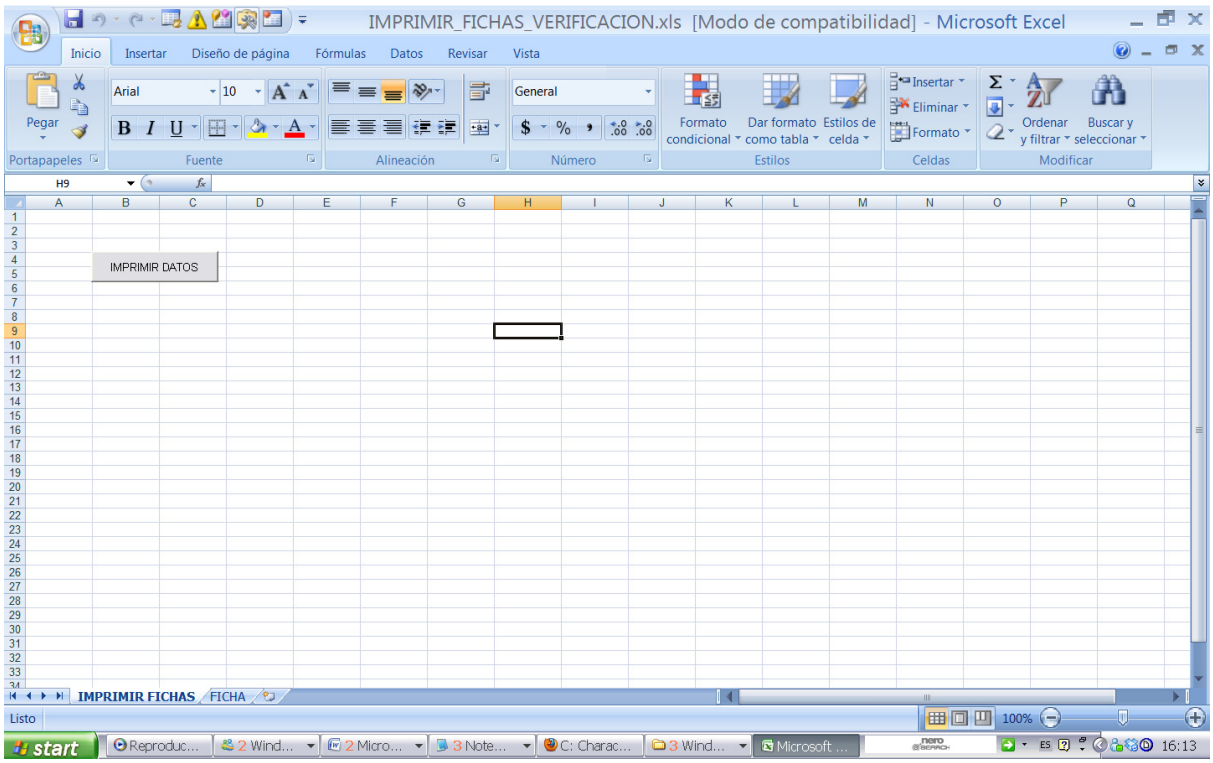


Figura 3.50: Archivo para imprimir la lista de trabajo

Una vez que se haya presionado el botón de IMPRIMIR DATOS, se procede a la impresión de las fichas por medio de una MACRO, el código de la macro se encuentra en el anexo 3, los datos son pasados a una ficha:

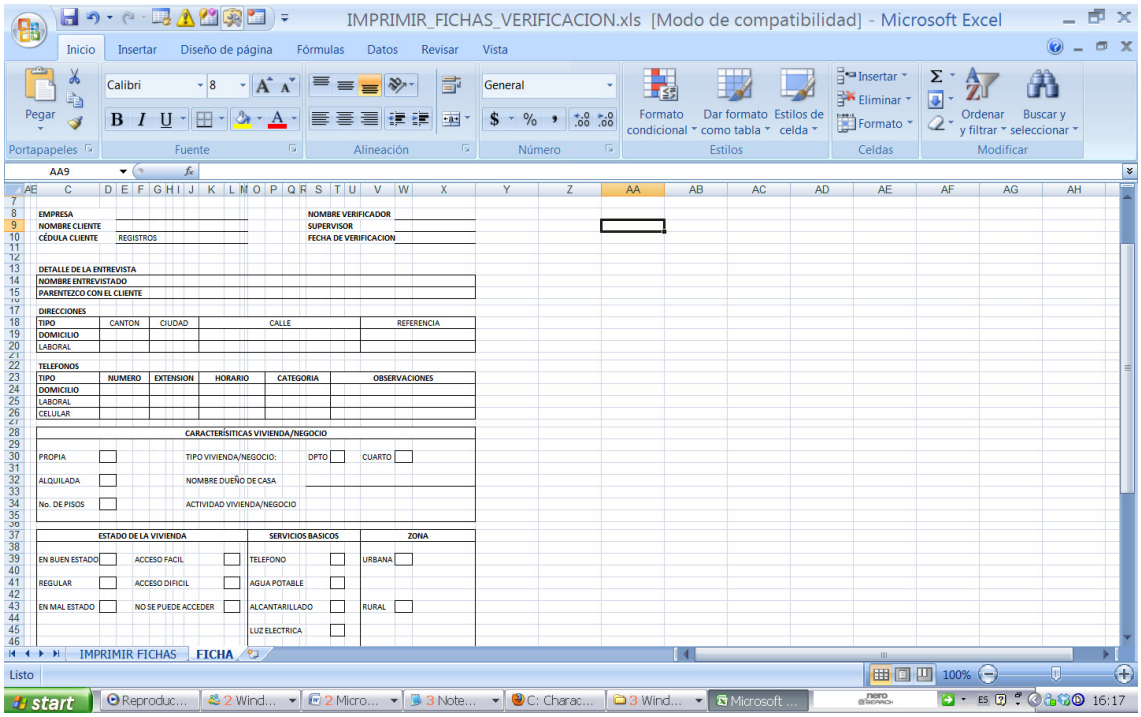


Figura 3.51: Ficha para la impresión de los datos

3.8 Generación de listas de trabajo

La generación de reportes, al igual que los dos módulos anteriores, también necesita que se tengan implementadas las demás administraciones así como también la generación de listas de trabajo ya que esta información se necesita para saber cuantas verificaciones han sido realizadas, este módulo, está relacionado con las fichas de verificación, básicamente una lista de trabajo se genera por cada verificador y se presenta al final de la pantalla de las fichas de verificación, el código de este módulo se encuentra en las fichas de verificaciones:

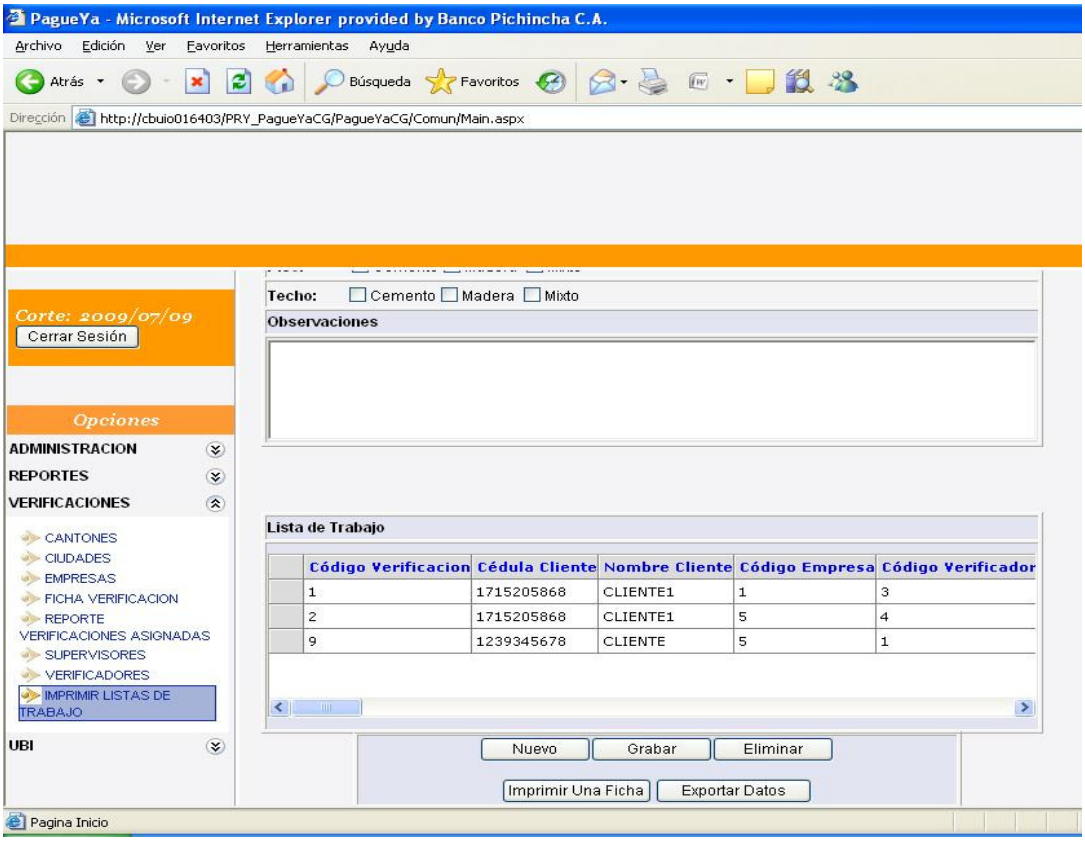


Figura 3.52: Presentación de las listas de trabajo para ser impresas

3.9 Generación de reportes:

Este módulo nos permite ver las verificaciones que se han realizado, básicamente se selecciona un criterio de búsqueda para saber cuántas visitas a los clientes se han realizado, a continuación el código desarrollado para este módulo:

CAPA DE DATOS DC:

```
/// <summary>
    /// Método de la capa de datos que se encarga de consultar
    /// los datos de las verificaciones
    /// </summary>
    /// <returns>Objeto de retorno</returns>
    public Retorno ConsultarRealizadas(string tipo,string valor,String fechaini,String fechafin)
    {
       AccesoDatos objAccesoDatos = new AccesoDatos();
       try
       {
          SqlParameter[] parms = new SqlParameter[4];
          parms[0] = new SqlParameter("@i_TIPO_ACCION", SqlDbType.Char,3);
```

```
            parms[0].Value = tipo;
            parms[1] = new SqlParameter("@i_VALOR", SqlDbType.VarChar, 100);
            parms[1].Value = valor;
            parms[2] = new SqlParameter("@i_FECHA_INICIO", SqlDbType.DateTime);
            parms[2].Value = Convert.ToDateTime(fechaini);
            parms[3] = new SqlParameter("@i_FECHA_FIN", SqlDbType.DateTime);
            parms[3].Value = Convert.ToDateTime(fechafin);
            return objAccesoDatos.ExecuteDataset(Fuentes.PRYCG, CommandType.StoredProcedure,
"PRO_VR_CONSULTA_VERIFICACIONES_REALIZADAS",parms);
        }
        catch (Exception ex)
        {
            throw (ex);
        }
        finally
        {
            if (objAccesoDatos != null)
                objAccesoDatos = null;
        }
    }
```

CAPA DEL NEGOCIO BC:

```
/// <summary>
/// Clase de logica del negocio que maneja la información de los cantones
/// </summary>
public class VR_Rep_Verificaciones_RealizadasBC
{
    #region Consultas

    /// <summary>
    /// Metodo de la capa de logica del negocio que consulta los
    /// registros de las verificaciones
    /// </summary>
    /// <returns>Dataset con los datos</returns>
    public DataSet ConsultarRealizadas(string tipo,string valor,String fechaini, String fechafin)
```

```
    {
      try
      {
        VR_Rep_Verificaciones_RealizadasDC objReporteDC = new VR_Rep_Verificaciones_RealizadasDC();
        return objReporteDC.ConsultarRealizadas(tipo,valor,fechaini,fechafin).Dataset;
      }
      catch (Exception ex)
      {
        throw (ex);
      }
    }
    #endregion
  }
```

CAPA DE INTERFACE DE USUARIO GUI:

```
<table id="Table2" align="center" border="0" cellpadding="1" designtimedragdrop="304"
      style="z-index: 102; left: 55px; width: 469px; position: absolute; top: 90px;
      height: 121px">
      <tr class="fconsult" style="font-weight: bold; color: #000000">
        <td style="width: 21px; height: 19px">
          <asp:CheckBox ID="cbxFecha" runat="server" Text="Fecha" Width="157px"
OnCheckedChanged="BloquearCampos"/></td>
        <td colspan="2" style="width: 32px; height: 19px">
          <asp:CheckBox ID="cbxEmpresa" runat="server" Text="Empresa" Width="157px" /></td>
      </tr>
      <tr class="fconsult">
        <td style="width: 21px; height: 17px">
          <strong>
            <asp:CheckBox ID="cbxCedula" runat="server" Text="Cédula Cliente" Width="157px" /></strong></td>
        <td colspan="2" style="width: 32px; height: 17px">
          <strong>
            <asp:CheckBox ID="cbxNombre" runat="server" Text="Nombre Cliente" Width="157px" /></strong></td>
      </tr>
      <tr class="fconsult">
        <td style="width: 21px; height: 22px">
```

```
            <strong>
              <asp:CheckBox ID="cbxCantonDom" runat="server" Text="Cantón Domicilio" Width="157px" /></strong></td>
          <td colspan="2" style="width: 32px; height: 22px">
            <strong>
              <asp:CheckBox ID="cbxCantonTrabajo" runat="server" Text="Cantón Trabajo" Width="157px" /></strong></td>
        </tr>
        <tr class="fconsult">
          <td style="width: 21px; height: 22px">
            <strong>
              <asp:CheckBox ID="cbxCiudadDomicilio" runat="server" Text="Ciudad Domicilio" Width="157px"
/></strong></td>
          <td colspan="2" style="width: 32px; height: 22px">
            <strong>
              <asp:CheckBox ID="cbxCiudadTrabajo" runat="server" Text="Ciudad Trabajo" Width="157px" /></strong></td>
        </tr>
        <tr class="fconsult">
          <td style="width: 21px; height: 22px">
            <strong>
              <asp:CheckBox ID="cbxVerificador" runat="server" Text="Nombre Verificador" Width="157px" /></strong></td>
          <td colspan="2" style="width: 32px; height: 22px">
            <strong>
              <asp:CheckBox ID="cbxSupervisor" runat="server" Text="Nombre Supervisor" Width="157px" /></strong></td>
        </tr>
      </table>
```

3.10 Archivo para imprimir varias fichas:

Como se mencionó anteriormente, para poder imprimir varias fichas de trabajo, se exporta toda la información de las fichas a un archivo Excel el cual, por medio de programación en MACROS, imprime automáticamente las fichas para ser entregadas a los verificadores, se utilizaron varias funciones en la programación de la MACRO, a continuación se presentan las funciónes de copia y de imprimir:

```
Private Sub COPIA(Origen As String, Destino As String)
   Sheets("DATOS").Select
   Range(Origen).Select
   Selection.Copy
   Sheets("FICHA").Select
   Range(Destino).Select
```

```
    Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks _

        :=False, Transpose:=False


End Sub
```

La function copia recibe dos parámetros, la celda de origen y la celda de destino, la primera, le indca de dónde debe leer los datos, y la segunda le indica al programa en que parte de la ficha debe colocar la información leída, después de haber implementado esta función, se procedió a implementar lafunción IMPRIMIR:

```
Sub IMPRIMIR()

' IMPRIMIR Macro

Dim intFilas, Aux As Double

intFilas = Sheets("DATOS").UsedRange.Rows.Count

Aux = intFilas - 1

Dim CONT As Integer

Dim Rango As String

CONT = 6

While CONT <= Aux

COPIA "B" + CStr(CONT), "E10"

COPIA "C" + CStr(CONT), "E9"

COPIA "D" + CStr(CONT), "E8"

COPIA "E" + CStr(CONT), "W8"

COPIA "F" + CStr(CONT), "W9"

COPIA "G" + CStr(CONT), "G19"

COPIA "H" + CStr(CONT), "D19"

COPIA "I" + CStr(CONT), "G20"

COPIA "J" + CStr(CONT), "D20"

COPIA "K" + CStr(CONT), "K19"

COPIA "L" + CStr(CONT), "K20"

COPIA "M" + CStr(CONT), "S34"

COPIA "N" + CStr(CONT), "V19"

COPIA "O" + CStr(CONT), "V20"

ExecuteExcel4Macro "PRINT(2,1,1,1,,,,,,,2,,,TRUE,,FALSE)"

CONT = CONT + 1

Wend

End Sub
```

Lo que hace esta función es llamar a la función de COPIA y le indica las columnas que debe copiar a la ficha de verificación, para paras a la siguiente fila, se utilizó un contador, el cual va aumentando después de haber completado la copia de toda la información de cada uno de los registros de la base de datos. El código completo de la implementación del archivo para imprimir las fichas, se encuentra en el anexo 3 así como también la ficha de verificaciones impresa.

3.11 Login de usuarios

Esta es la primera pantalla que se les presenta a los usuarios cuando deseen utilizar la aplicación de verificaciones, sirve para que los usuarios accesen al sistema con un nombre de usuario y una contraseña, para trabajar en esta pantalla, se utilizó la clase de usuarios BC y DC

3.11.1 Interface de usuario

El siguiente código se lo colocó en las demás clases de GUI del módulo, lo que hace es redireccionar a la página de login si la sesión ha caducado o si el usuario no ha ingresado aún al sistema, para el caso, tomamos el código de la pantalla de ciudades:

```
protected void Page_Load(object sender, EventArgs e)
{
    try
    {

        //Comprueba que el usuario haya iniciado sesión antes de mostrar la página
        if (Session["strUsuarioAPP"] == null)
        {
            Response.Redirect("../Comun/Login.aspx");
        }
        if (!Page.IsPostBack)
        {
            agregarConfirmacion(btnEliminar, "OnClick", "Desea eliminar el registro?");
            blnNuevo = true;
            strCodigoCiudad = "";
            ConsultaCiudades();
            ConsultarCantones();
            limpiarDatos();
        }
    }
    catch (Exception ex)
    {
        MessageError(ex.Message);
    }
}
```

Si el usuario no ha iniciado sesión, se le presentará la siguente pantalla:

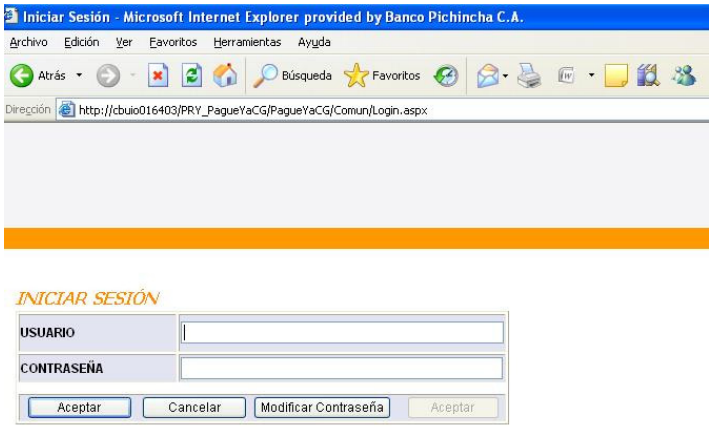


Figura 3.53: pantalla para inicio de sesión de los usuarios

A continuación las pruebas para el ingreso de usuarios al sistema:

3.11.2 Pruebas unitarias

El ingreso de usuarios tiene dos funcionalidades básicas: Ingreso de usuarios al sistema y el cambio de contraseña de los usuarios:

INGRESANDO AL SISTEMA:

La primera pantalla que se le presenta a los usuarios del sistema es la de registro de usuarios:

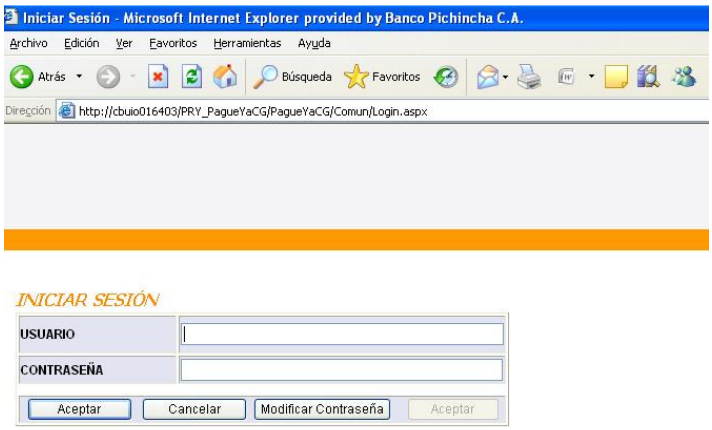


Figura 3.54: Pantalla de registro de usuarios

Si los datos de los usuarios son correctos, se direcciona a la pantalla principal del sistema, de lo contrario, se le presenta la siguiente pantalla de error:

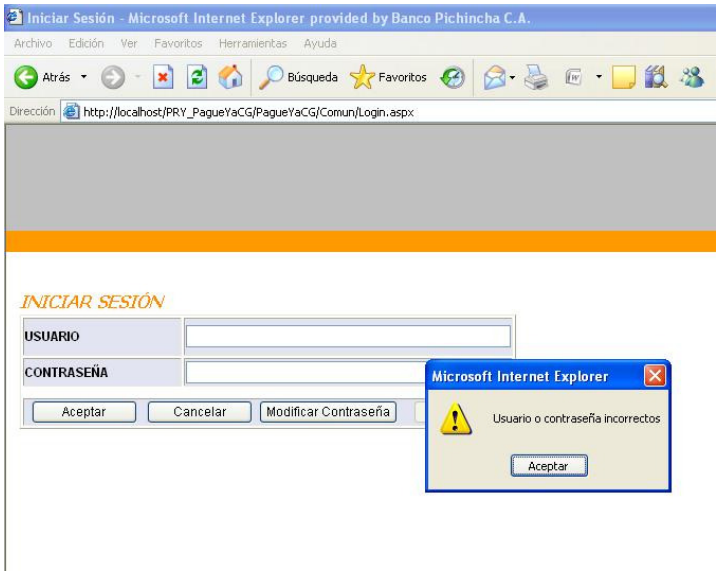


Figura 3.55: Error al ingresar al sistema

Si los datos son correctos, se le presenta la siguiente pantalla al usuario:

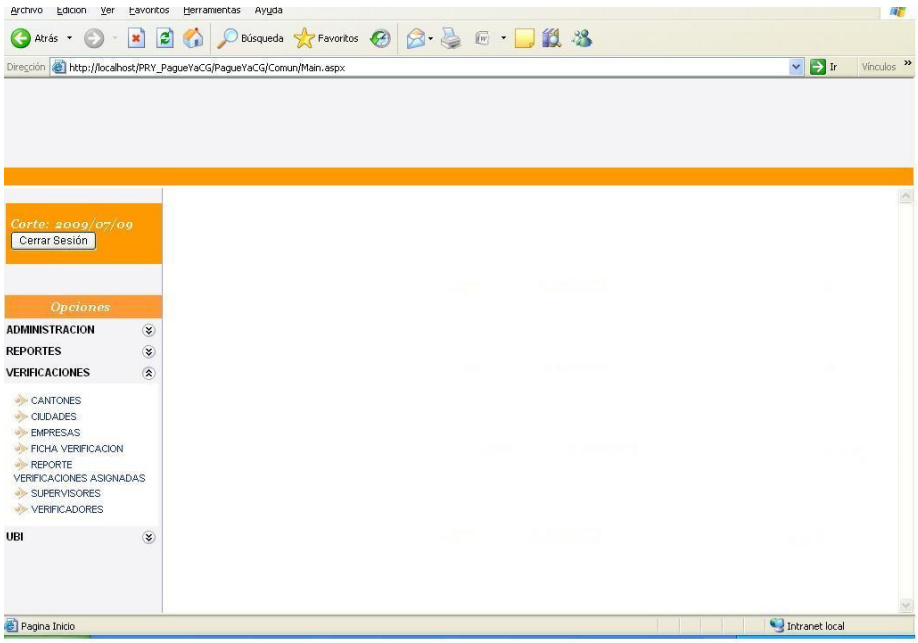


Figura 3.56: Pantalla de menú principal

CAMBIANDO LA CONTRASEÑA DE UN USUARIO:

Esta es la otra funcionalidad del ingreso de usuarios, se le presenta la misma pantalla de ingreso pero si el usuario presiona el botón "MODIFICAR CONTRASEÑA", la pantalla cambia a la siguiente forma:

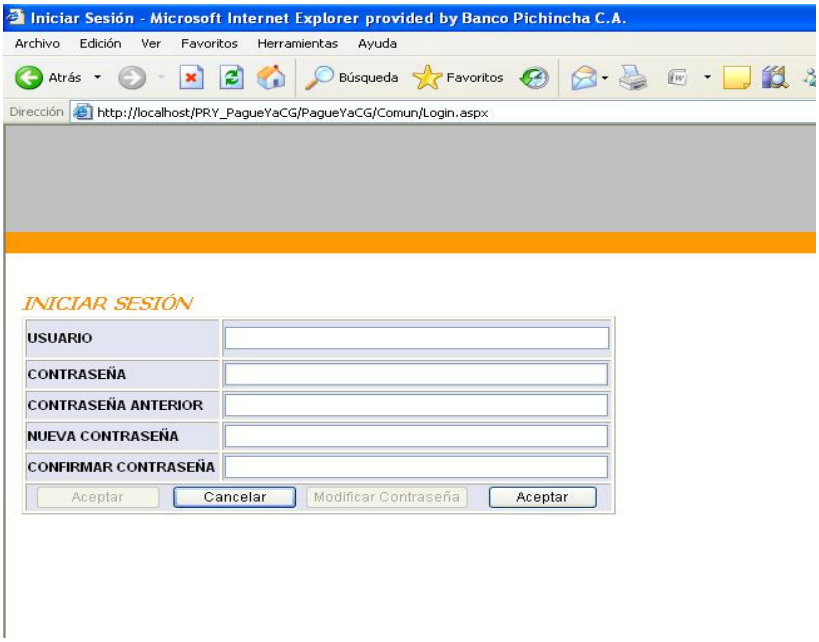


Figura 3.57: Cambio de contraseña

En este caso, el usuario debe ingresar su contraseña anterior, la ueva contraseña y debe reingresar la nueva contraseña para confirmación:

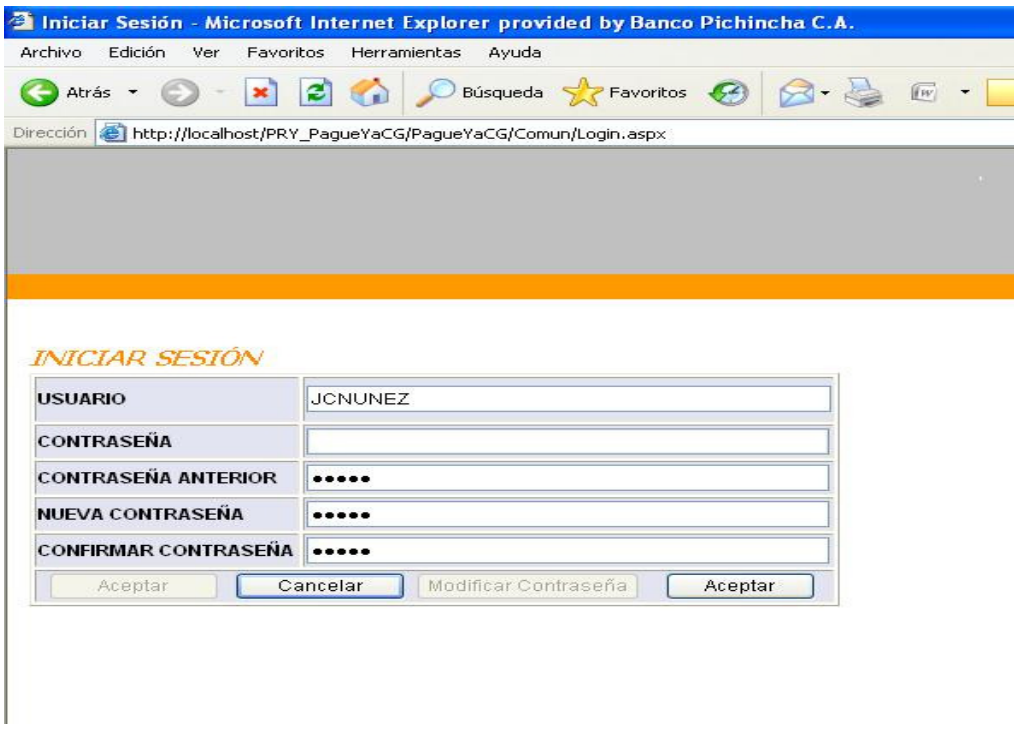


Figura 3.58: Ingesando los datos de nueva contraseña

Si la nueva contraseña y la confirmación o coinciden, se le presenta al usuario el siguiente mensaje:

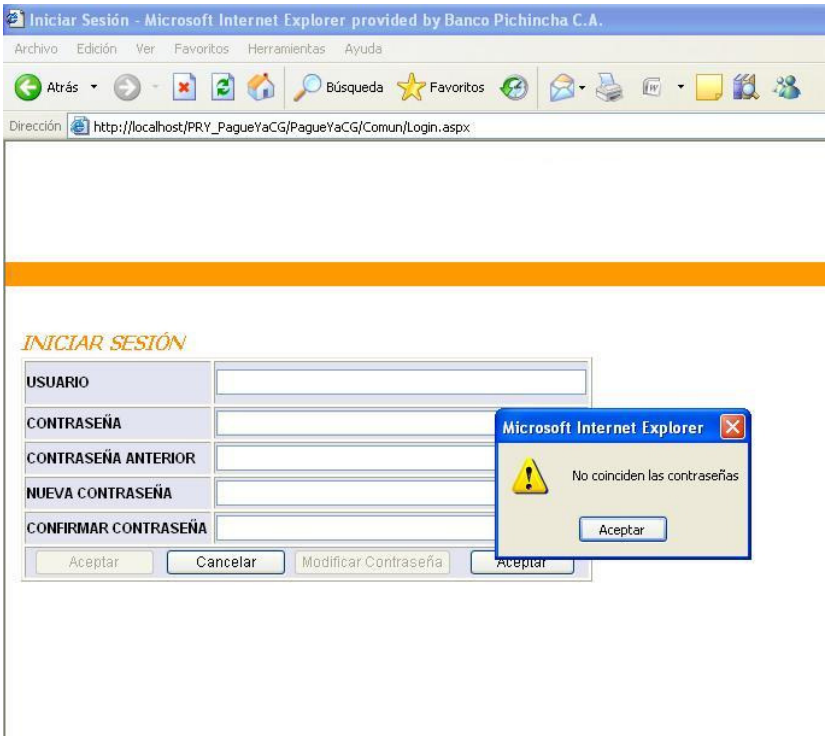


Figura 3.59: La nueva contraseña y la confirmación no coinciden

Si la nueva contraseña y la confirmación coinciden, se redireccionará al usuario a la página principal de la aplicación.

4 CAPÍTULO 4 Pruebas del sistema

El plan de pruebas del sistema es muy importante en el proceso de desarrollo de software ya que nos indica los resultados esperados de cada módulo del sistema. Como se habló en el capitulo 2 del presente trabajo, para el plan de pruebas de integración del sistema, se escogió el plan de pruebas de integración por dependencias ya que en éste modelo se comienza el ensamblaje del sistema probando aquellas clases, llamadas independientes, que usan muy poca o ninguna otra clase del sistema; después de que las clases independientes han sido probadas se continúa con la próxima etapa de clases, las cuales usan a las independientes (por ello se les llama clases dependientes). Esta secuencia de pruebas por capas de clases dependientes continúa hasta que el sistema entero se construye.

En el caso del Sistema de Verificación de Información, las clases independientes son las siguientes:

- CANTONES
- EMPRESAS
- SUPERVISORES
- INGRESO AL SISTEMA

Las clases dependientes son:

- ASESORES (VERIFICADORES)
- CIUDADES
- VERIFICACIONES

A continuación el plan de pruebas de integración para el Sistema de Verificación de Información, para lo cual se seguirá el orden definido en el diagrama de casos de uso:

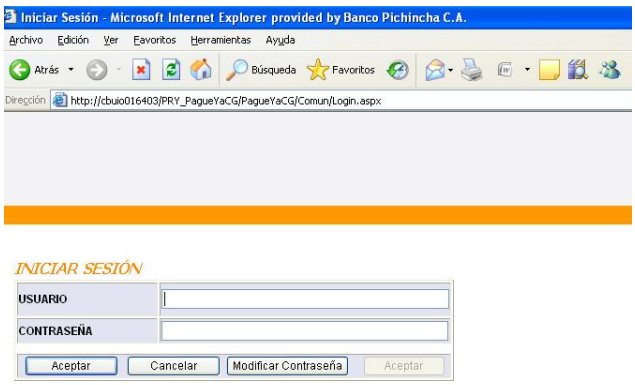
Caso de prueba: F0 Ingreso al sistema
Precondiciones: ninguna

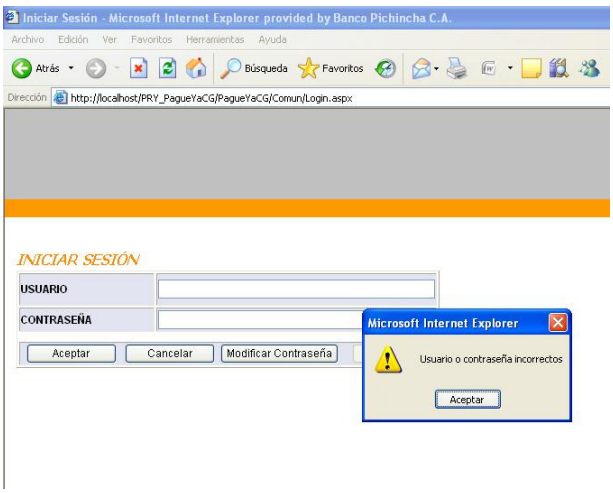
ENTRADAS	**RESULTADOS ESPERADOS**	**CASO USO**
Presenta pantalla de ingreso al sistema	Desplegar plantilla de empresas	F0
Ingreso de datos de usuaario		
correcto	Pantalla menú principal	F0
incorrecto	Mensaje de error	

Tabla 4.1: Caso de prueba de Ingreso al sistema

Como podemos observar, el resultado que se esperan al ingresar los datos correctamente, es que se presente la pantalla del menú principal del sistema, en la cual se podrá escoger las diferentes administraciones que ofrece el sistema, a continuación los resultados del sistema:

4.1 CASO DE USO F0: Ingreso al sistema





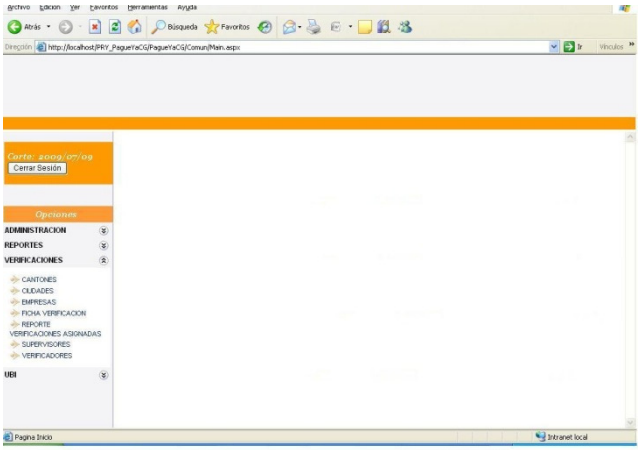


Figura 4.1: Resultados esperados de ingreso al sistema

Siguiendo el diagrama de casos de uso, la siguiente administración el la de Empresas, a continuación las pruebas para éste módulo:

Caso de prueba: F1 Administración de clientes
Precondiciones: Pantalla principal

ENTRADAS	**RESULTADOS ESPERADOS**	**CASO USO**
Selecciona opción Adm. Empresas	Desplegar plantilla de empresas	F1
Ingreso de empresas		
correcto	Datos almacenados	
incorrecto	Mensaje de error	F1.1
Actualización de datos		
correcto	Actualizar datos	
incorrecto	Mensaje de error	F1.2
Eliminar datos	Se elimina la información seleccionada	F1.3

Tabla 4.2: Caso de prueba del módulo Administración de empresas

Como podemos observar, la tabla contiene los resultados esperados cuando se realiza alguna actividad en el módulo de Empresas, a continuación se presentan los resultados del sistema para cada caso de uso:

4.2 CASO DE USO FI: Selecciona opción Administración de empresas

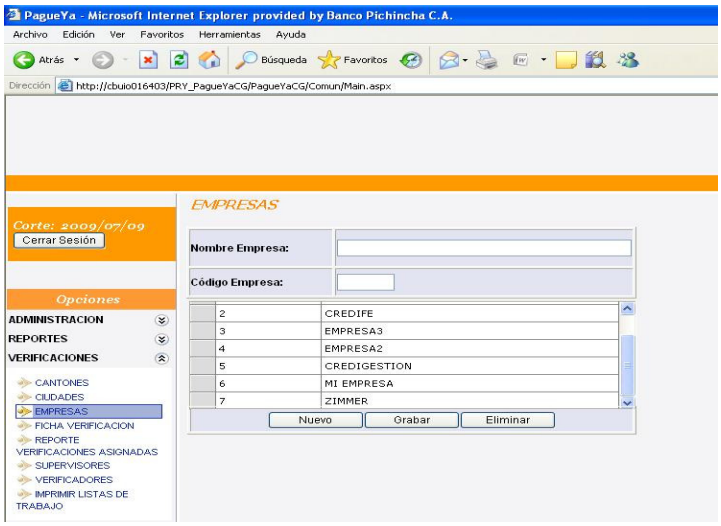


Figura 4.2: Resultados esperados de ingreso al módulo de empresas

4.2.1 CASO DE USO F1.1: Ingreso de Empresas

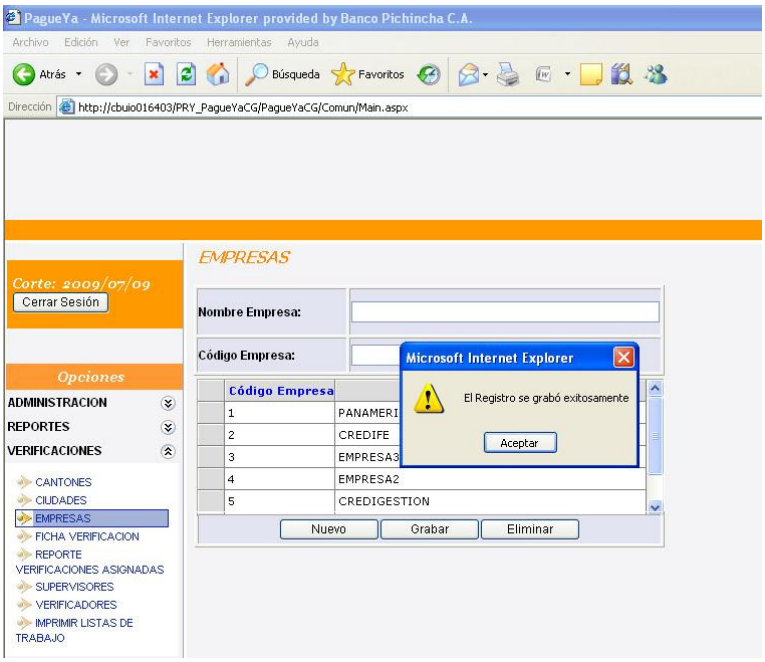


Figura 4.3: Resultados esperados de ingreso de una nueva empresa

4.2.2 CASO DE USO F1.2: Actualización de Empresas

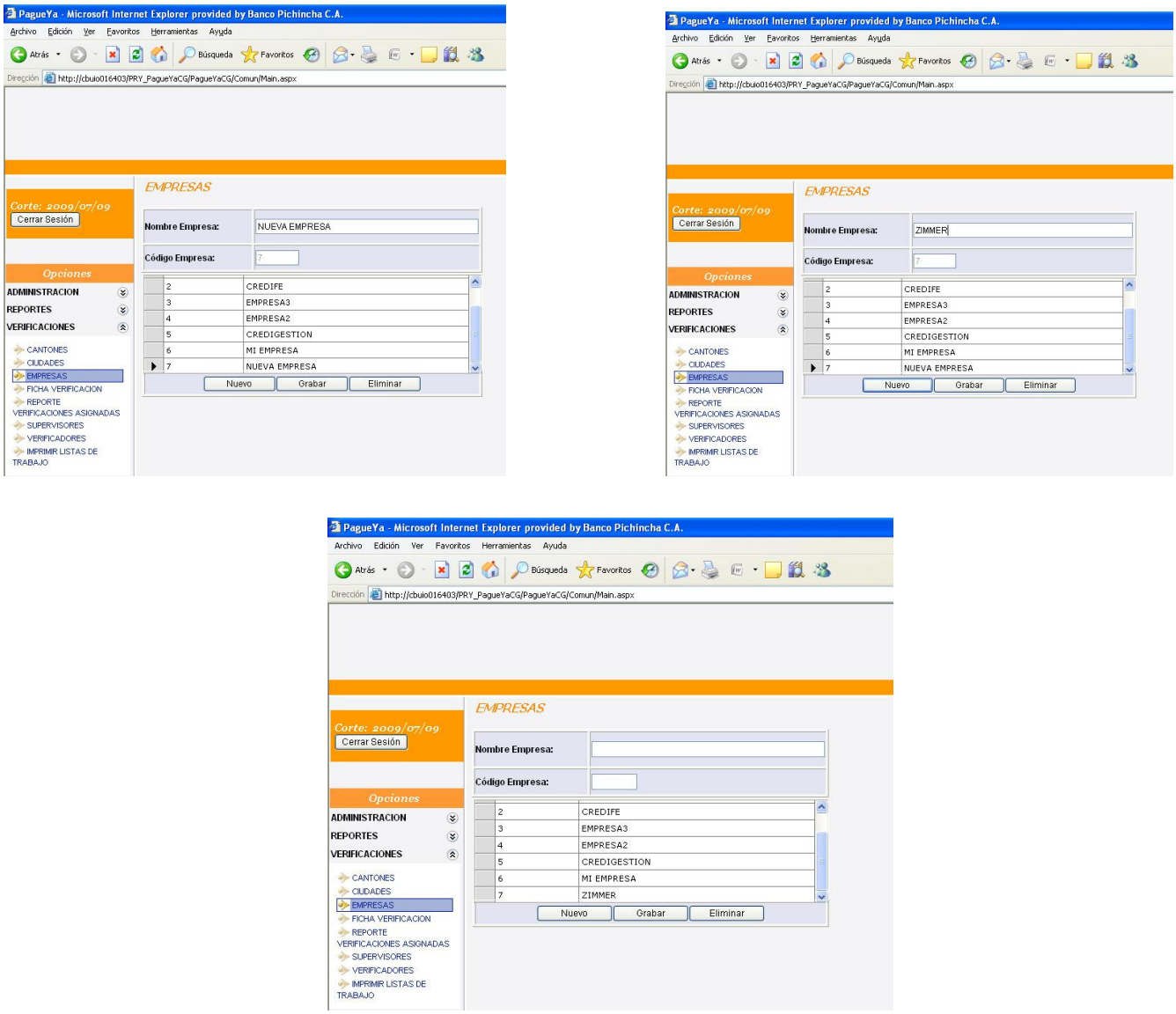


Figura 4.4: Resultados esperados de actualizar una empresa

4.2.3 CASO DE USO F1.3: Eliminar Empresas

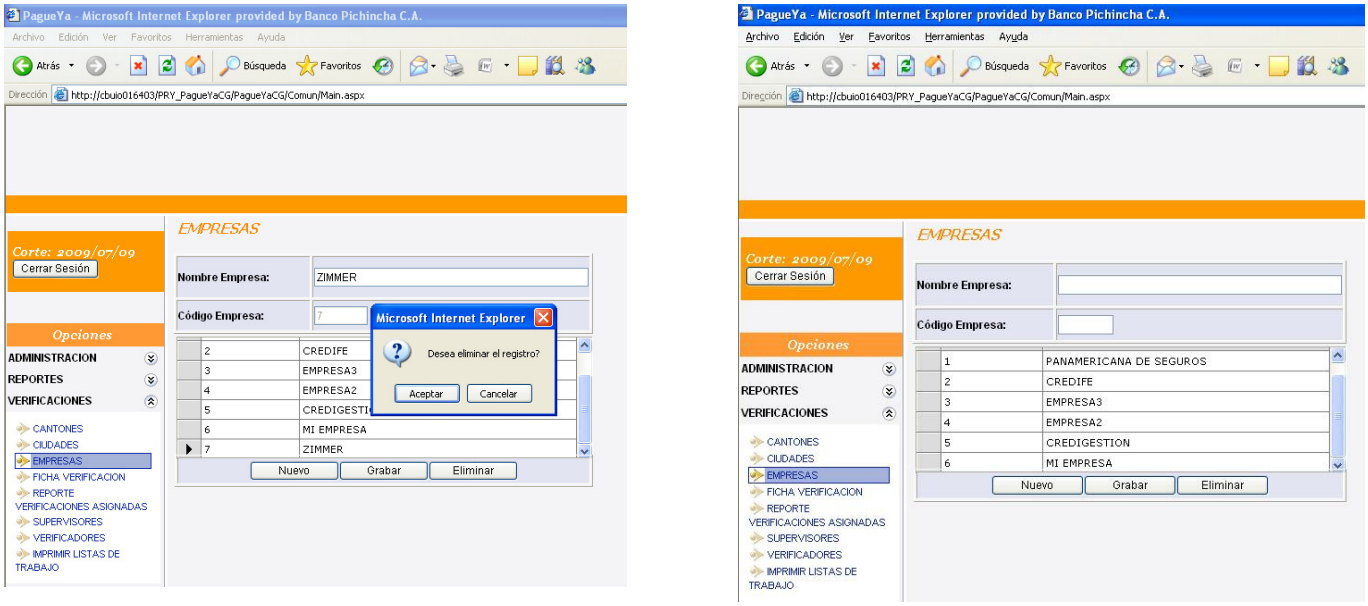


Figura 4.5: Resultados esperados de eliminar una empresa

La siguiente administración, de acuerdo al diagrama de casos de uso es la administración de cantones:

Caso de prueba: F2 Administración de cantones
Precondiciones: ninguna

ENTRADAS	RESULTADOS ESPERADOS	CASO USO
Selecciona opción Adm. Cantones	Pantalla principal	F2
Ingreso de cantones		
correcto	Datos almacenados	
incorrecto	Mensaje de error	F2.1
Actualización de datos		
correcto	Actualizar datos	
incorrecto	Mensaje de error	F2.2
Eliminar datos	Se elimina la información seleccionada	F2.3

Tabla 4.3: Caso de prueba del módulo Administración de cantone

4.3 CASO DE USO F2: Pantalla principal

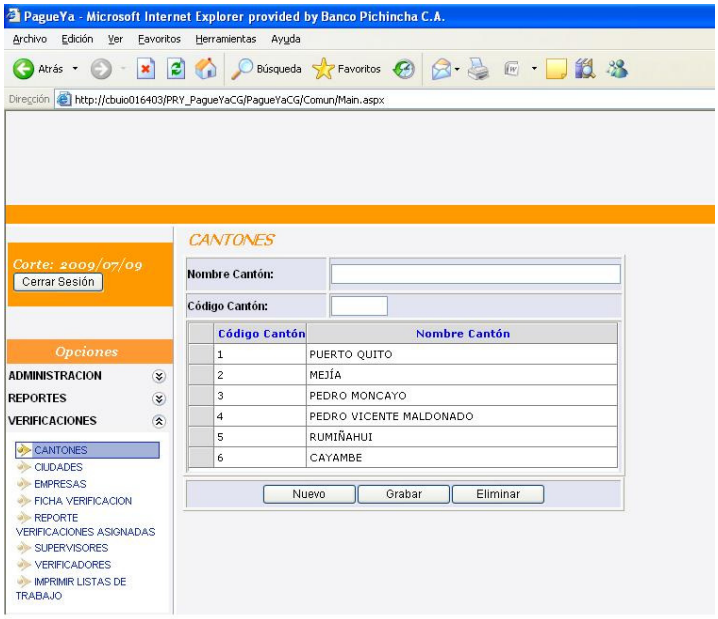


Figura 4.6: Resultados esperados de ingreso al módulo de cantones

4.3.1 CASO DE USO F2.1: Ingresar Cantones

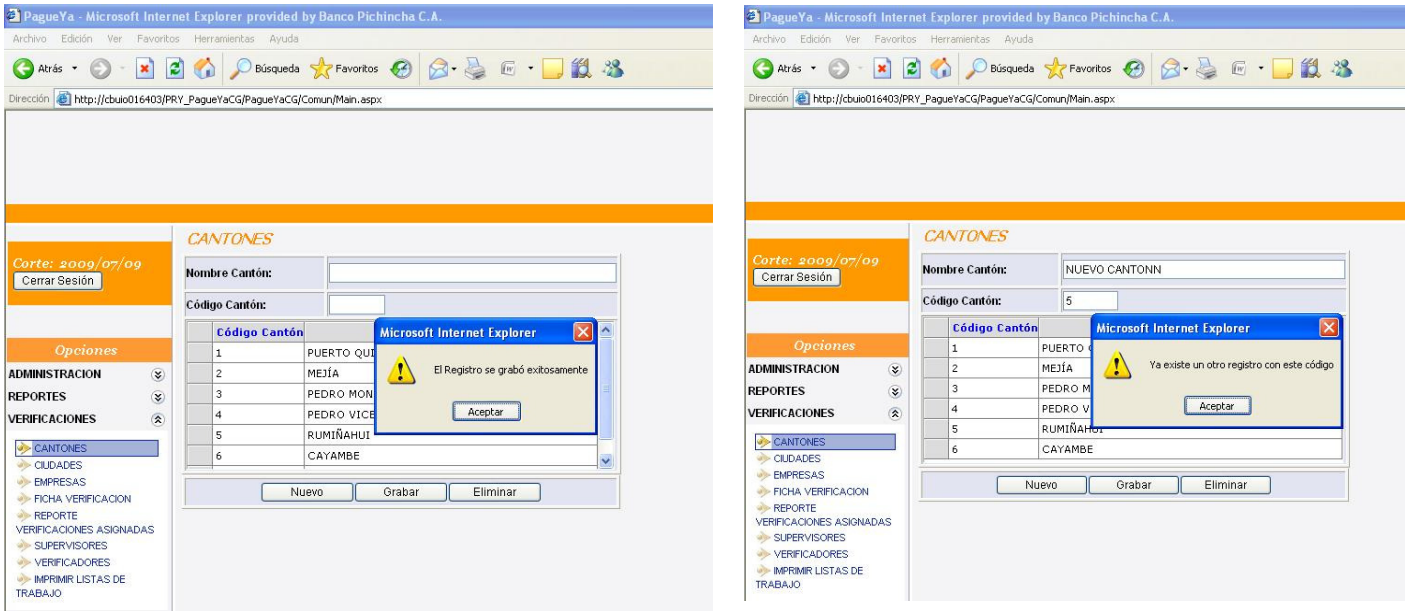


Figura 4.7: Resultados esperados de ingresar un nuevo cantón

4.3.2 CASO DE USO F2.2: Actualizar Cantones

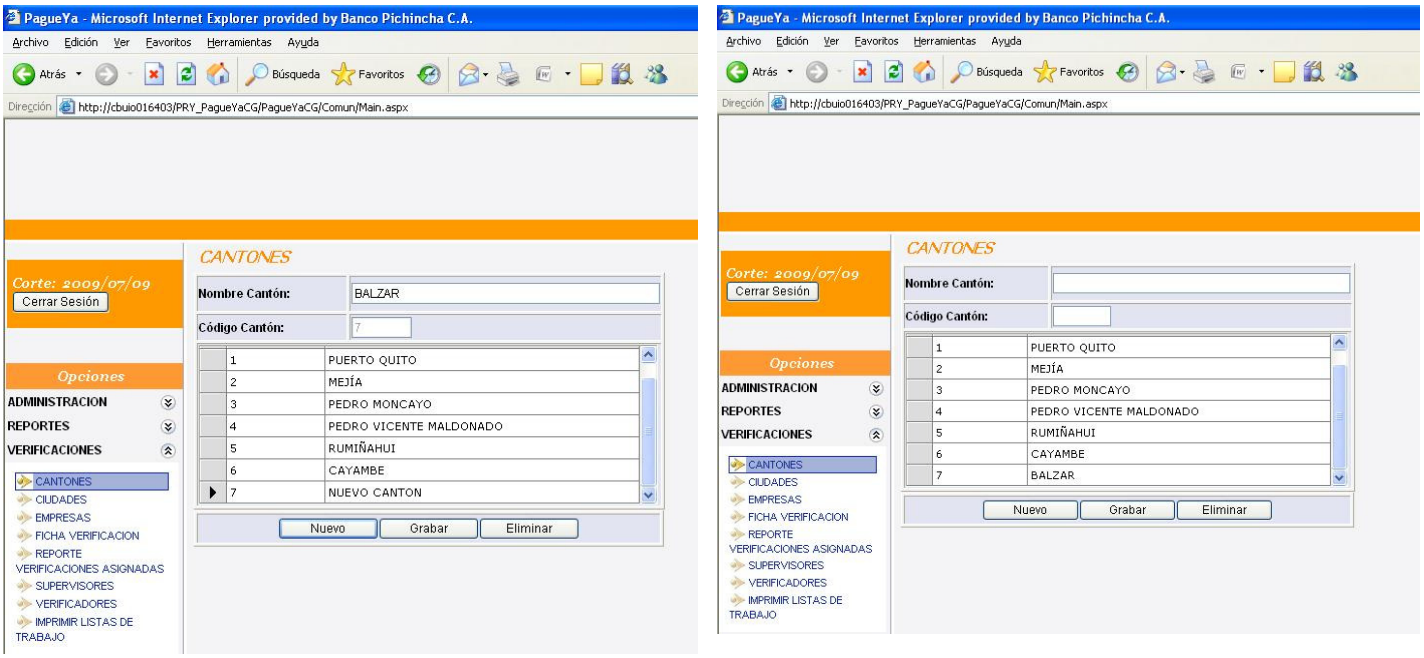
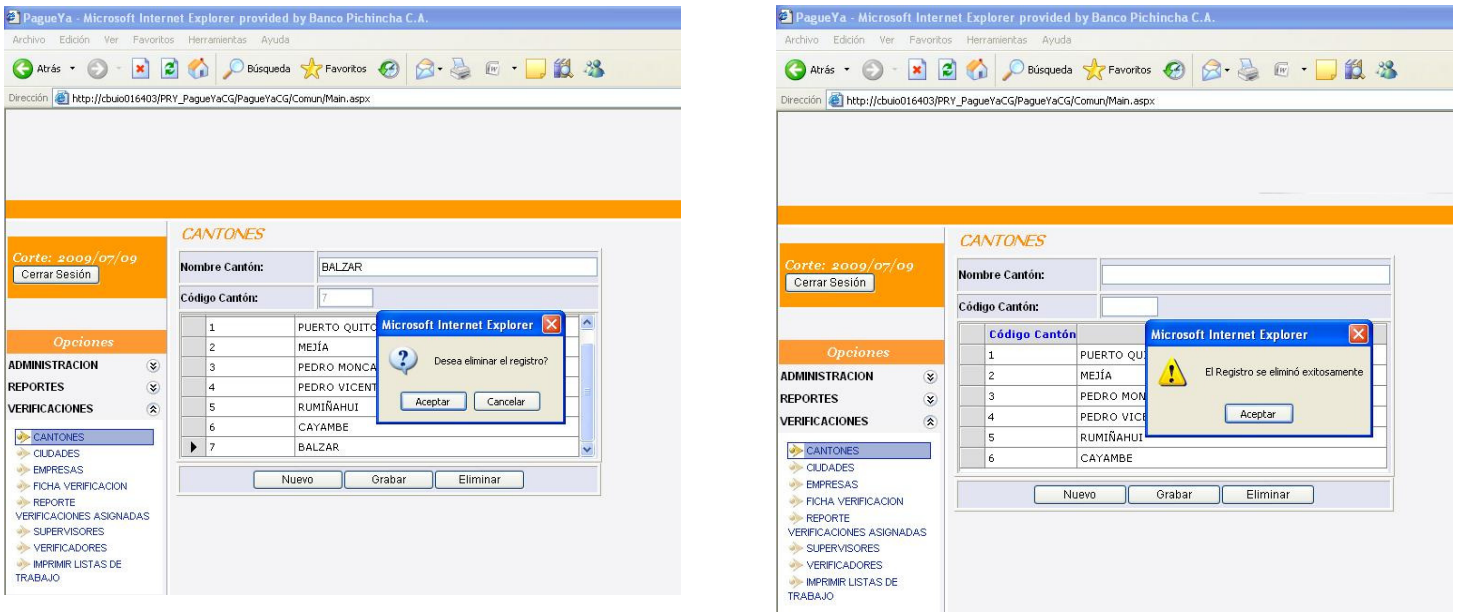


Figura 4.8: Resultados esperados de actualizar un cantón

4.3.3 CASO DE USO F2.3: Borrar Cantones



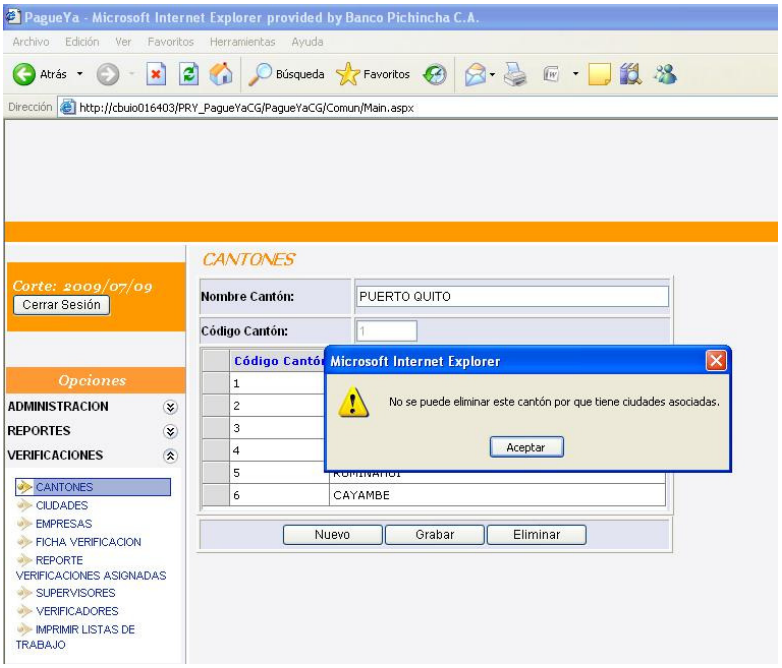


Figura 4.9: Resultados esperados de eliminar un cantón

La siguiente administración, de acuerdo al diagrama de casos de uso es la administración de ciudades, en este caso, es necesario tener ya implementada la administración de cantones ya que se necesitan los datos de los mismos para poder almacenar una ciudad:

Caso de prueba: F3 Administración de ciudades
Precondiciones: cantones debe estar implementado y Pantalla principal

ENTRADAS	**RESULTADOS ESPERADOS**	**CASO USO**
Selecciona opción Adm. Ciudades	Desplegar plantilla de ciudades	F3
Ingreso de cantones		
correcto	Datos almacenados	
incorrecto	Mensaje de error	F3.1
Actualización de datos		
correcto	Actualizar datos	
incorrecto	Mensaje de error	F3.2
Eliminar datos	Se elimina la información seleccionada	F3.3

Tabla 4.4: Caso de prueba del módulo Administración de ciudades

4.4 CASO DE USO F3: Pantalla Principal

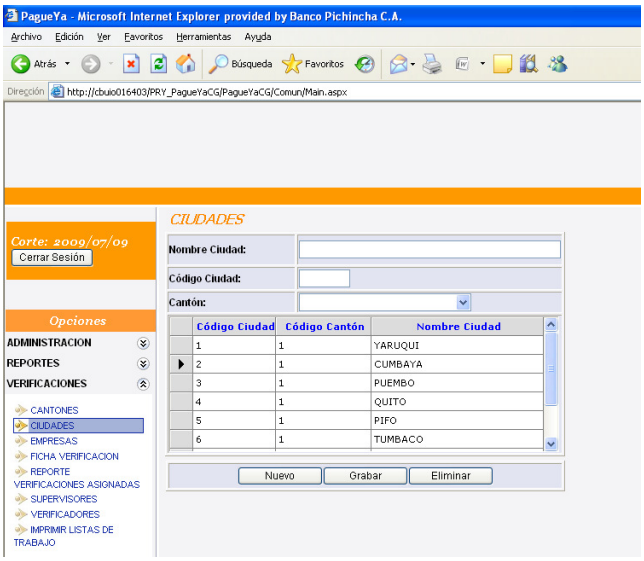


Figura 4.10: Resultados esperados de ingresar al módulo de ciudades

4.4.1 CASO DE USO F3.1: Ingresar Ciudades

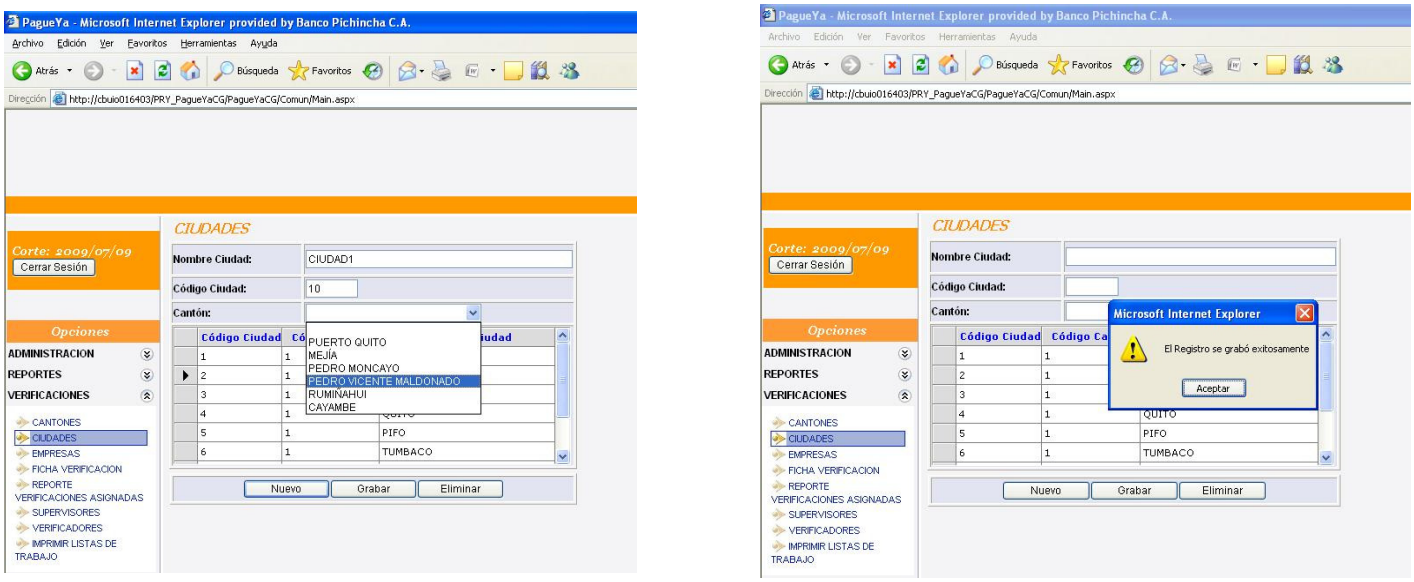


Figura 4.11: Resultados esperados de ingresar una nueva ciudad

4.4.2 CASO DE USO F3.2: Actualizar Ciudades

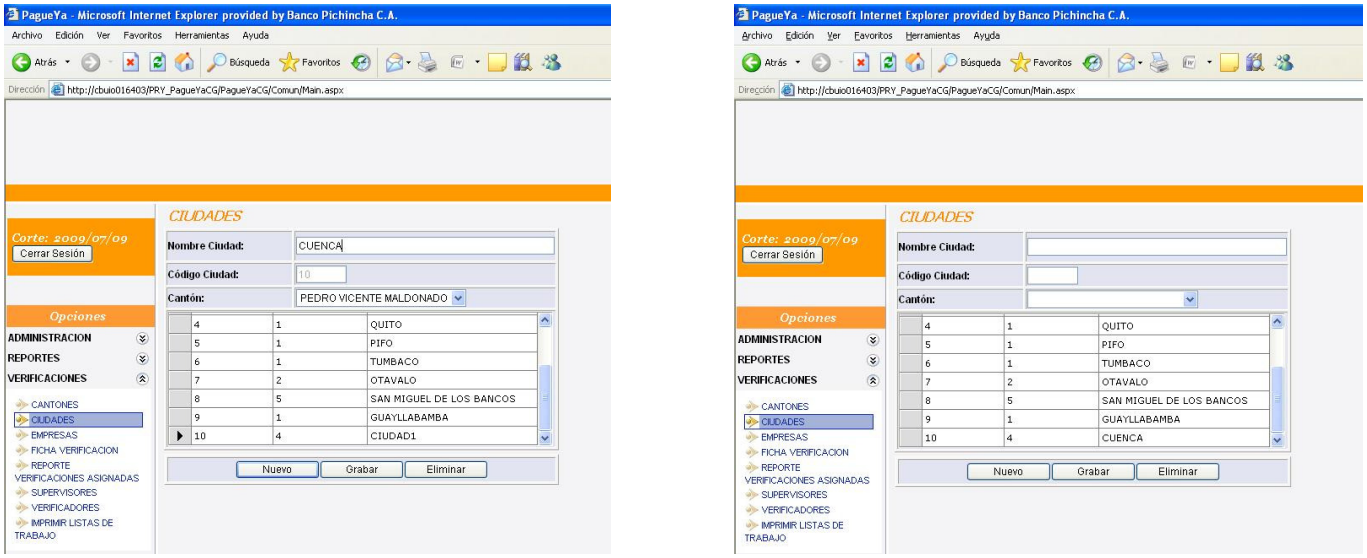


Figura 4.12: Resultados esperados de actualizar una ciudad

4.4.3 CASO DE USO F3.3: Borrar Ciudades

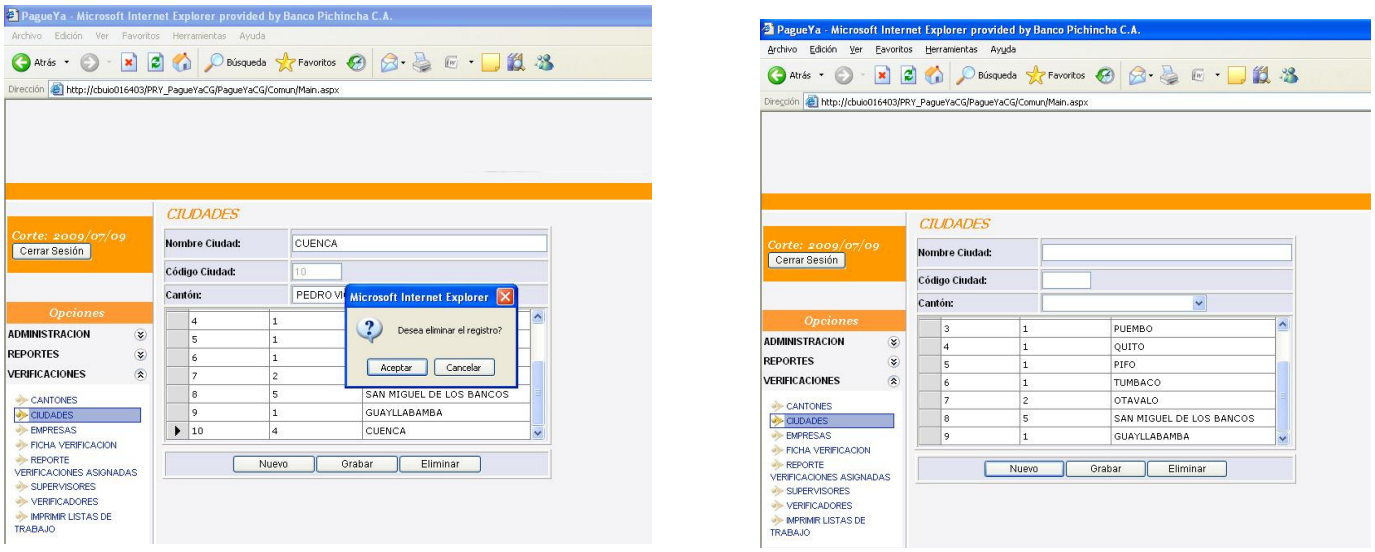


Figura 4.13: Resultados esperados de eliminar una ciudad

La siguiente administración, de acuerdo al diagrama de casos de uso es la administración de supervisores:

Caso de prueba: F4 Administración de supervisores
Precondiciones: Pantalla principal

ENTRADAS	RESULTADOS ESPERADOS	CASO USO
Selecciona opción Adm. Supervisores	Desplegar plantilla de supervisores	F4
Ingreso de cantones		
correcto	Datos almacenados	
incorrecto	Mensaje de error	F4.1
Actualización de datos		
correcto	Actualizar datos	
incorrecto	Mensaje de error	F4.2
Eliminar datos	Se elimina la información seleccionada	F4.3

Tabla 4.5: Caso de prueba del módulo Administración de supervisores

4.5 CASO DE USO F4: Pantalla Prinipal

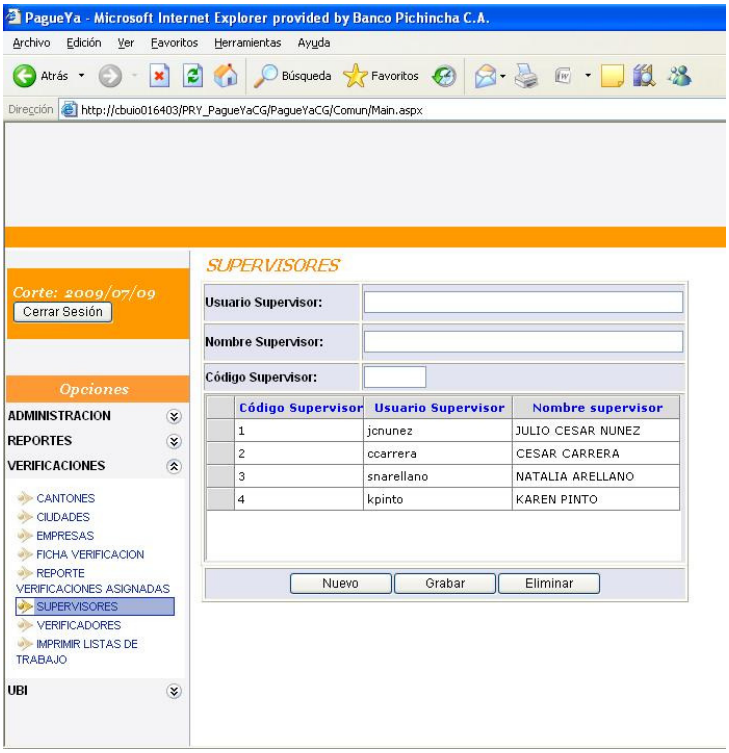


Figura 4.14: Resultados esperados de ingreso al módulo de supervisores

4.5.1 CASO DE USO F4.1: Ingresar Supervisores

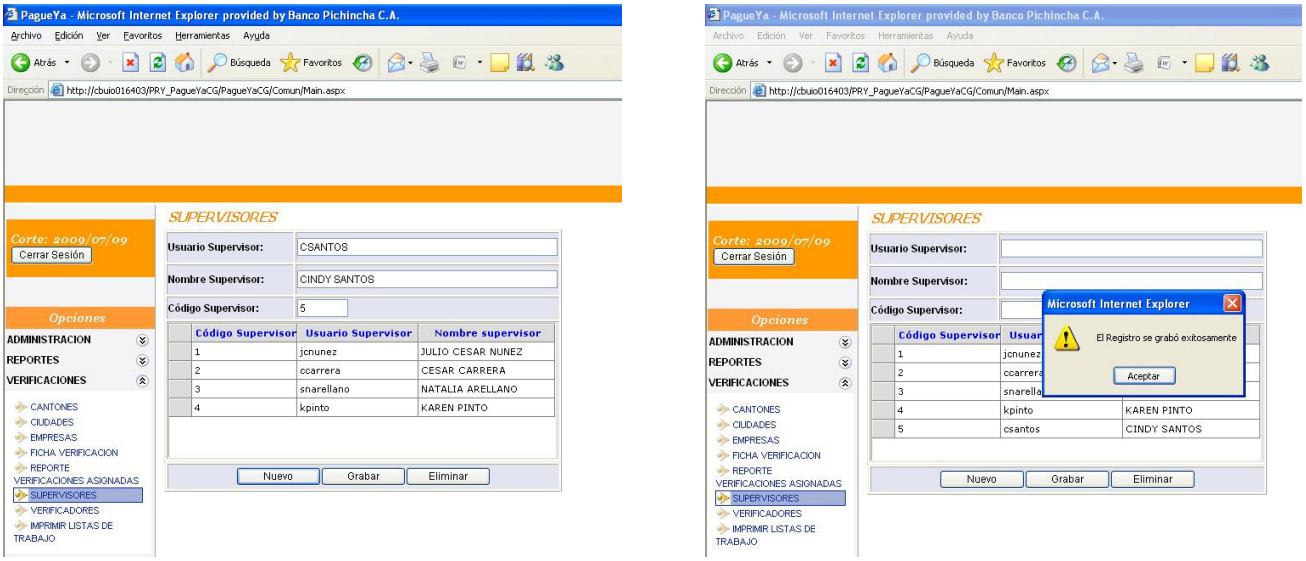


Figura 4.15: Resultados esperados de ingresar un nuevo supervisor

4.5.2 CASO DE USO F4.2: Actualizar Supervisores

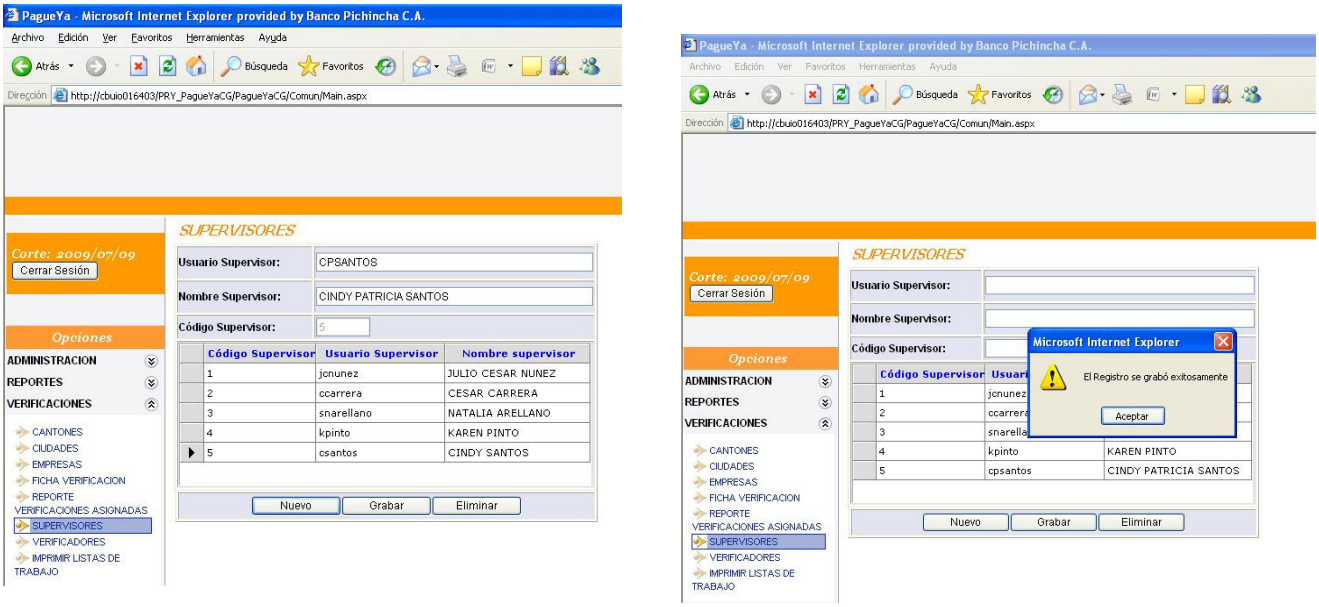
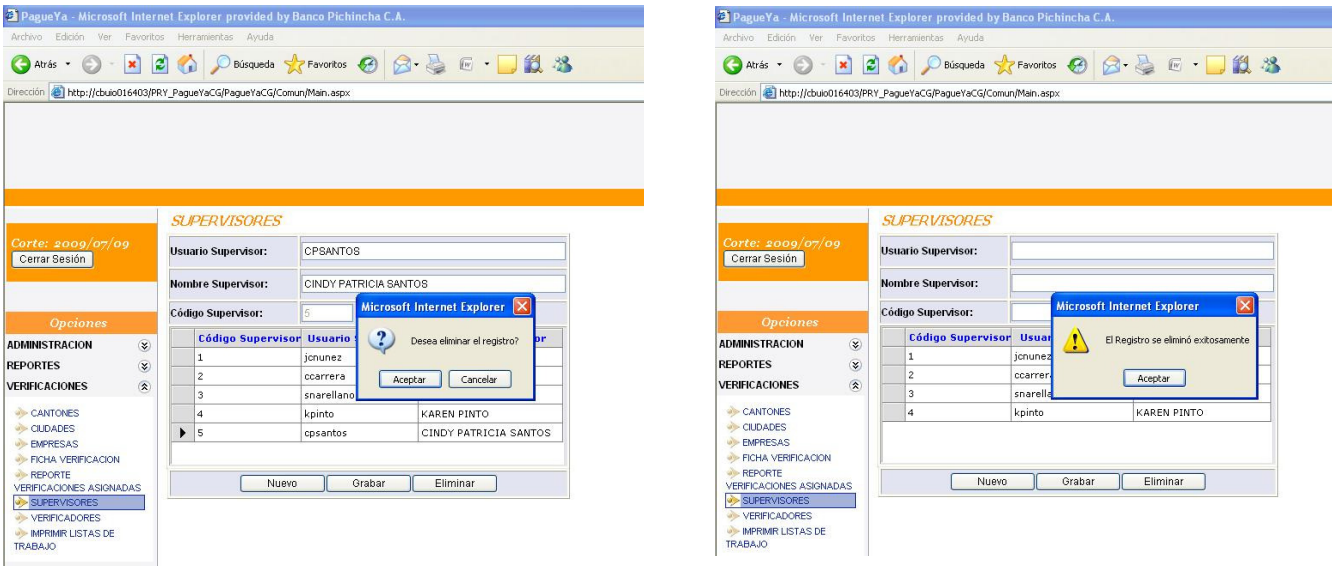


Figura 4.16: Resultados esperados de actualizar un supervisor

4.5.3 CASO DE USO F4.3: Eliminar Supervisores



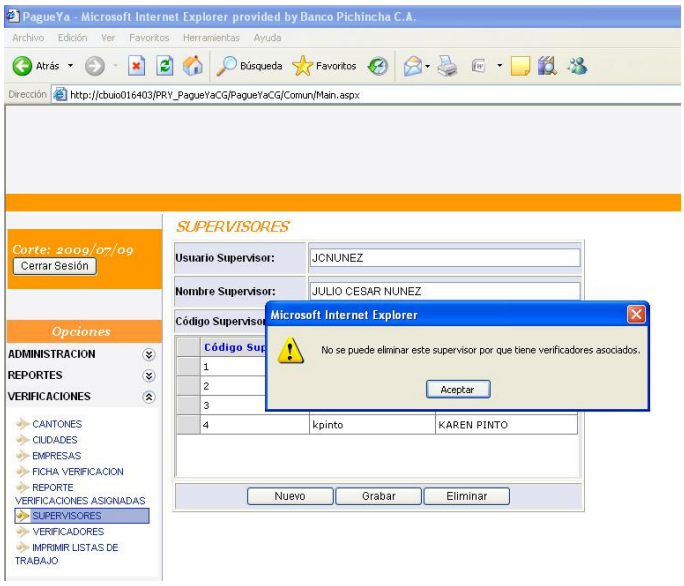


Figura 4.17: Resultados esperados de eliminar un supervisor

La siguiente administración, de acuerdo al diagrama de casos de uso es la administración de verificadores, para lo cual, al igual que las ciudades, se necesita que esté implementada la administración de supervisores:

Caso de prueba: F5 Administración de verificadores
Precondiciones: Verificadores debe estar implementado y Pantalla principal

ENTRADAS	RESULTADOS ESPERADOS	CASO USO
Selecciona opción Adm. Verificadores	Desplegar plantilla de verificadores	F5
Ingreso de cantones		
correcto	Datos almacenados	
incorrecto	Mensaje de error	F5.1
Actualización de datos		
correcto	Actualizar datos	
incorrecto	Mensaje de error	F5.2
Eliminar datos	Se elimina la información seleccionada	F5.3

Tabla 4.6: Caso de prueba del módulo Administración de verificadores

4.6 CASO DE USO F5: Pantalla Principal

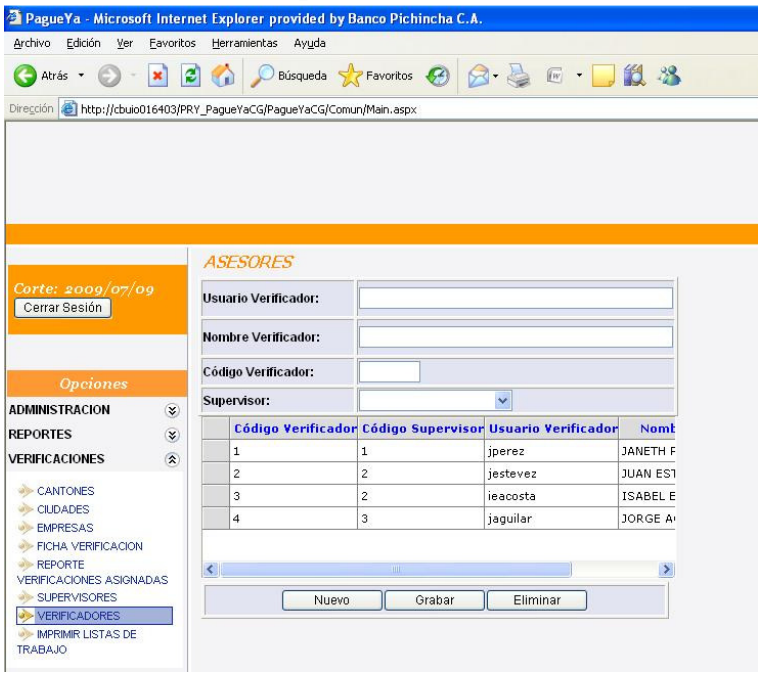
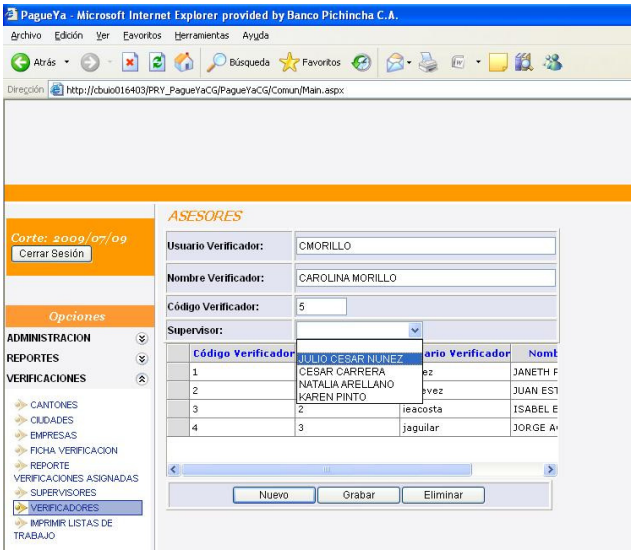


Figura 4.18: Resultados esperados de ingresar al módulo de Verificadores

4.6.1 CASO DE USO F5.1: Ingresar Verificadores



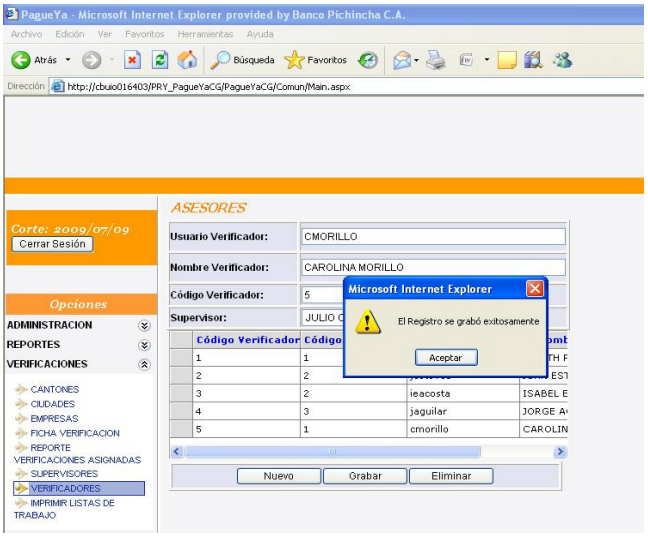
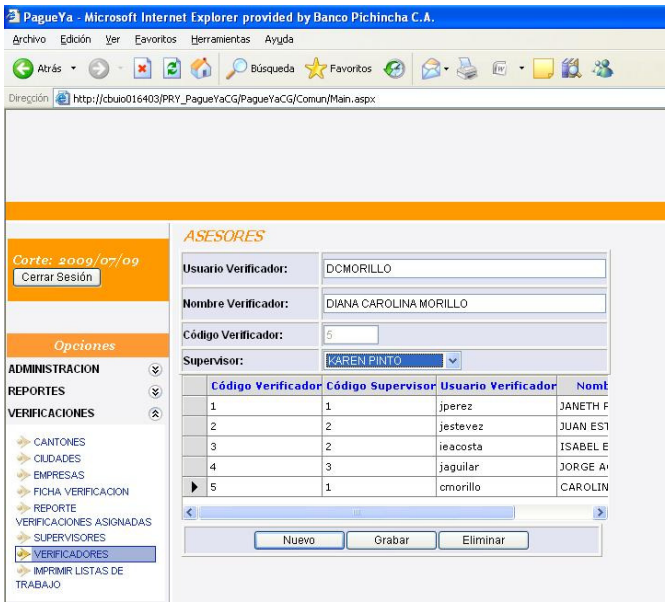


Figura 4.19: Resultados esperados de ingresar un nuevo verificador

4.6.2 CASO DE USO F5.2: Actualizar Verificadores



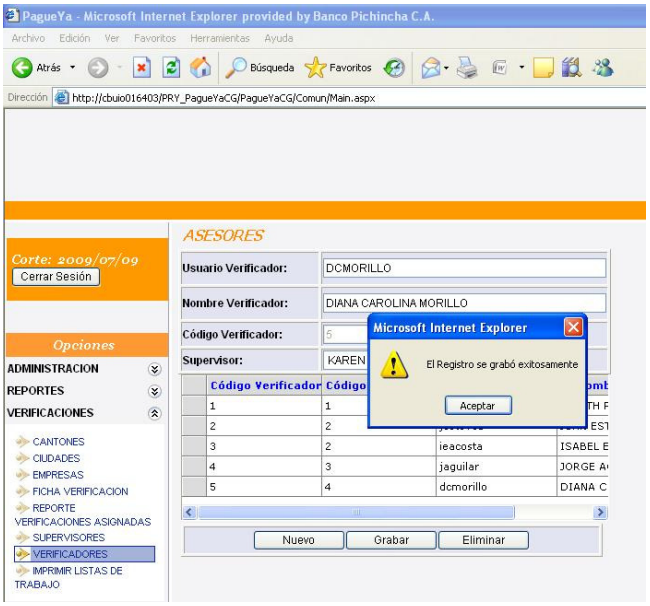


Figura 4.20: Resultados esperados de actualizar un verificador

4.6.3 CASO DE USO F5.3: Eliminar Verificadores

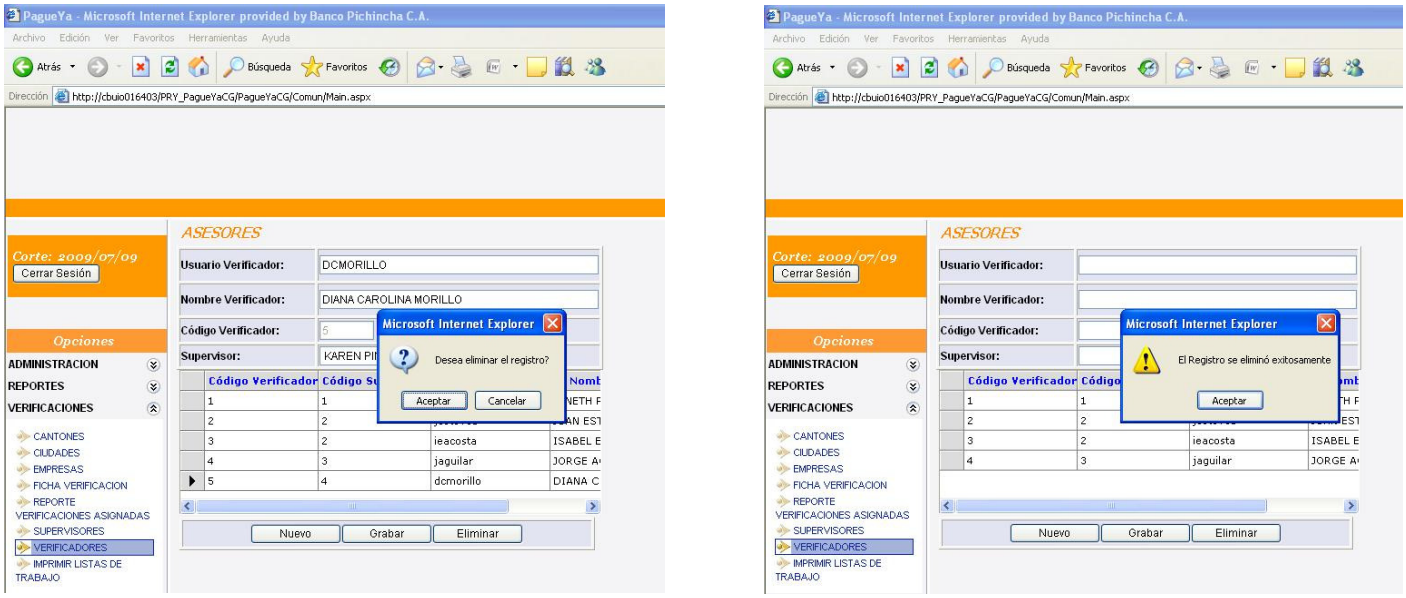


Figura 4.21: Resultados esperados de eliminar un verificador

La siguiente administración, de acuerdo al diagrama de casos de uso es la administración de los usuarios que ingresarán al sistema:

Caso de prueba: F6 Administración de usuarios
Precondiciones: Pantalla principal

ENTRADAS	**RESULTADOS ESPERADOS**	**CASO USO**
Selecciona opción Adm. usuarios	Desplegar plantilla de usuarios	F6
Ingreso de cantones		
correcto	Datos almacenados	
incorrecto	Mensaje de error	F6.1
Actualización de datos		
correcto	Actualizar datos	
incorrecto	Mensaje de error	F6.2
Consultar datos	Datos ingresados reflejados en la consulta	F6.3

Tabla 4.7: Caso de prueba del módulo Administración de usuaios

4.7 CASO DE USO F6: Pantalla Prinicpal

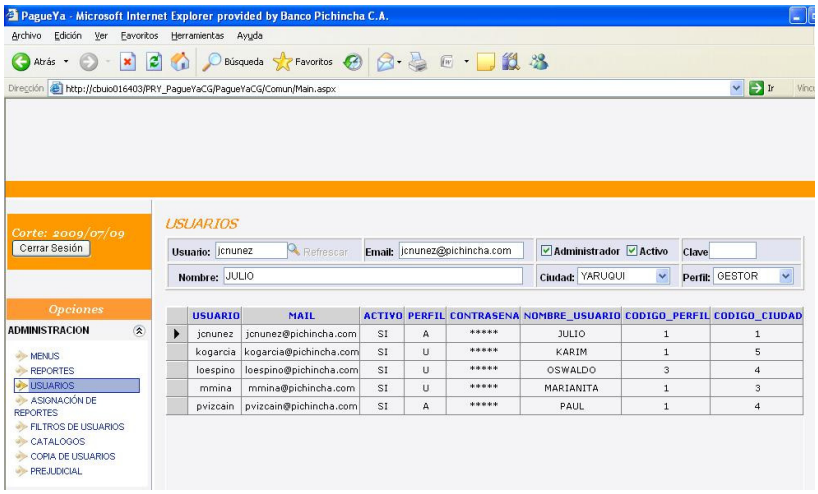
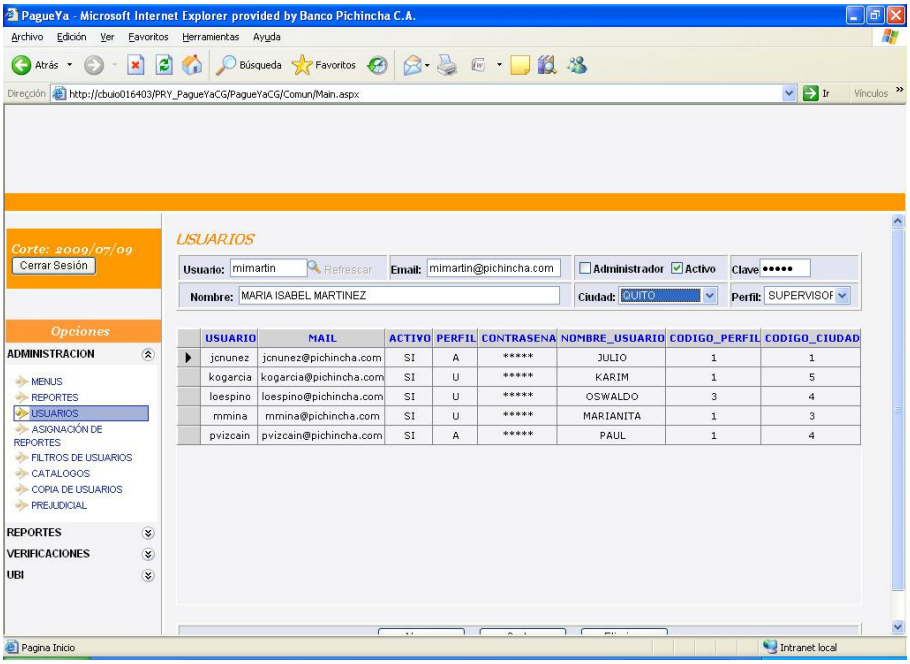


Figura 4.22: Resultados esperados de ingresar al módulo de usuarios

4.7.1 CASO DE USO F6.1: Ingresar Usuarios



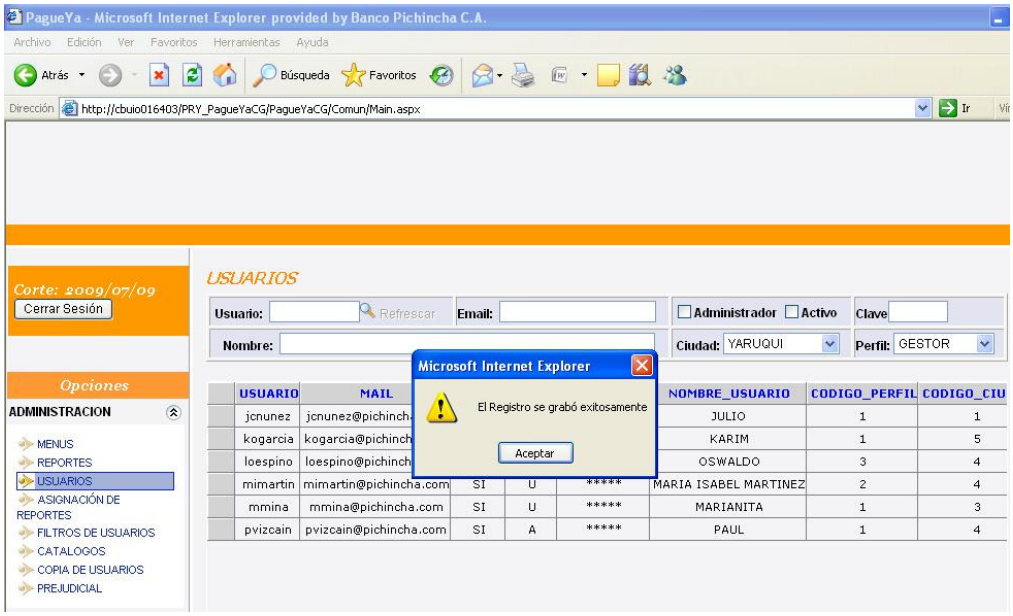


Figura 4.23: Resultados esperados de ingresar un nuevo usuario

4.7.2 CASO DE USO F6.2: Actualizar Usuarios

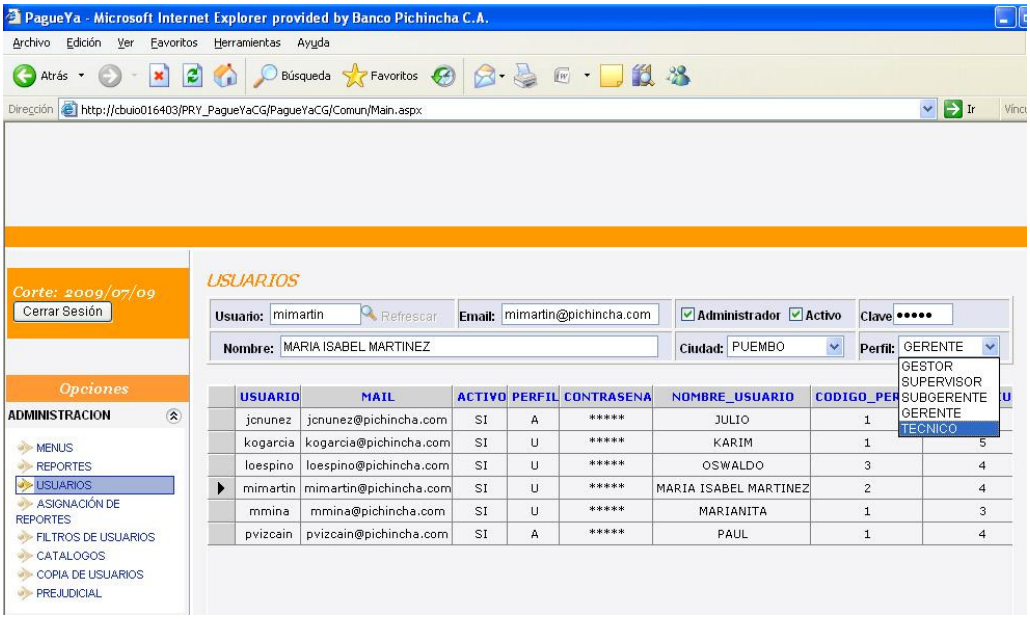
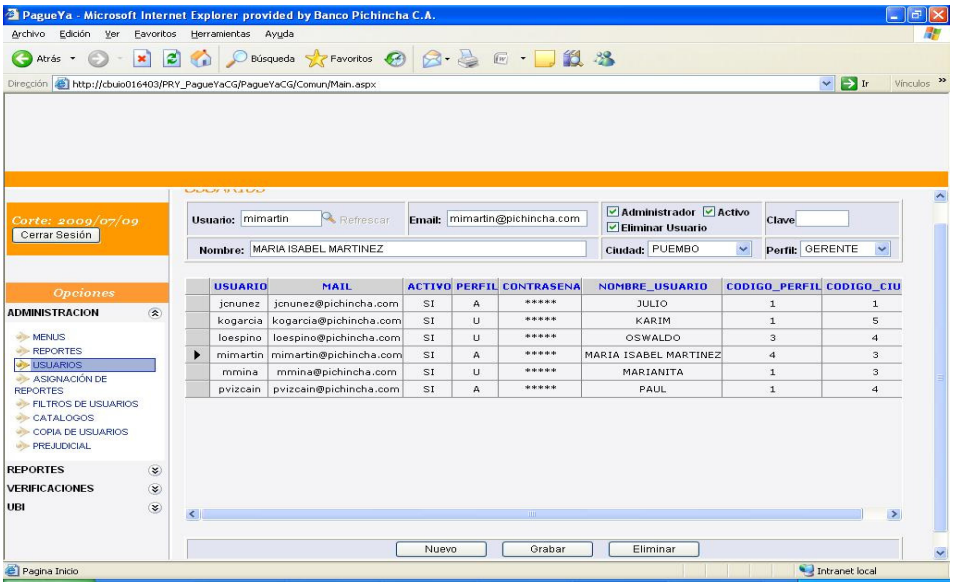


Figura 4.24: Resultados esperados de actualizar un usuario

4.7.3 CASO DE USO F6.3: Eliminar Usuarios



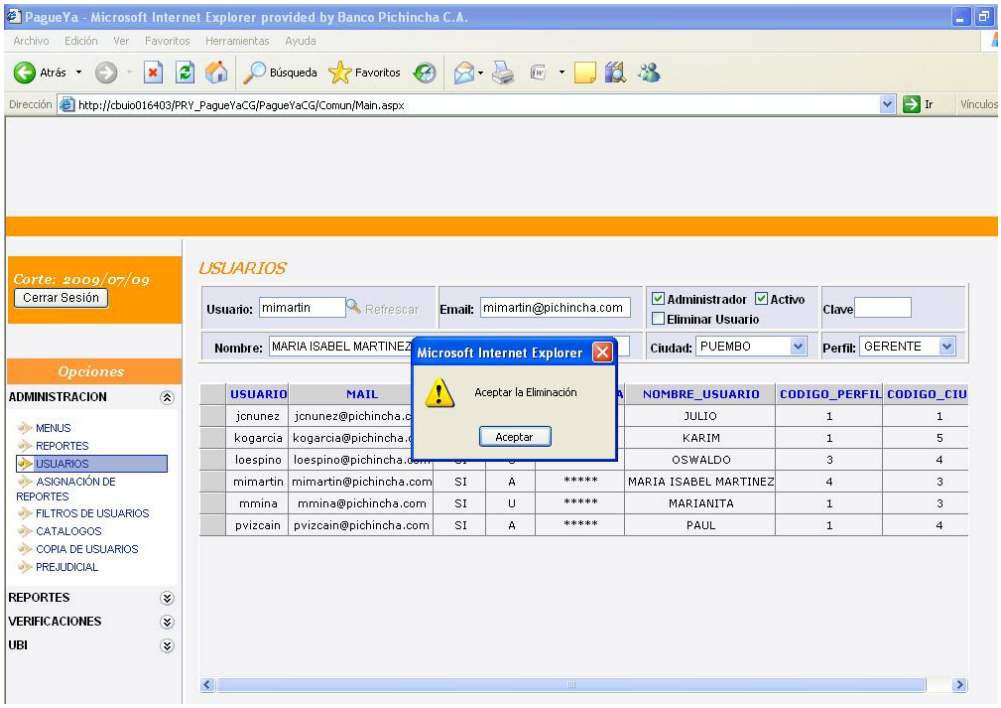


Figura 4.25: Resultados esperados de eliminar un usuario

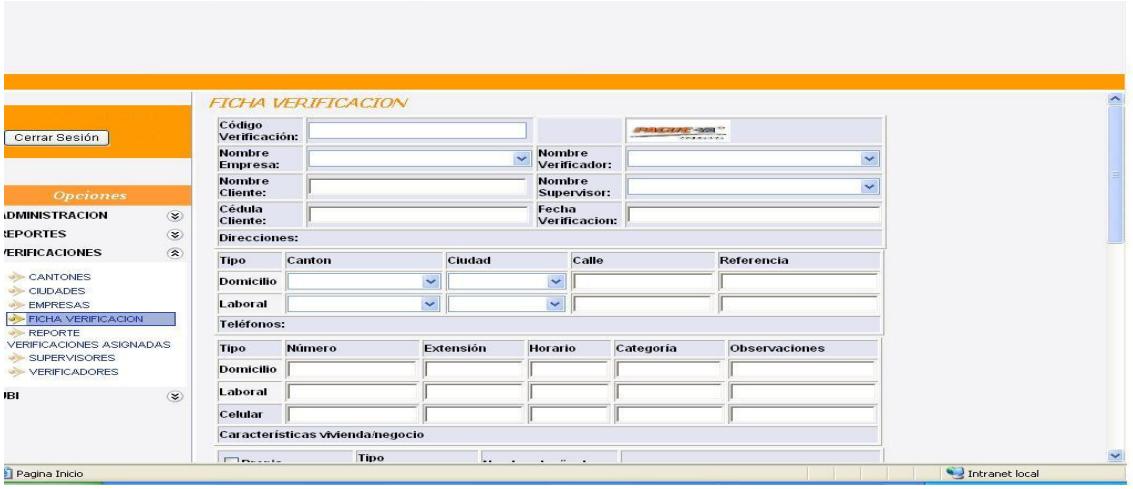
El siguiente módulo, de acuerdo al diagrama de casos de uso es la generación de fichas de verificación, para lo cual, todas las administraciones anteriores, deben estar implementadas:

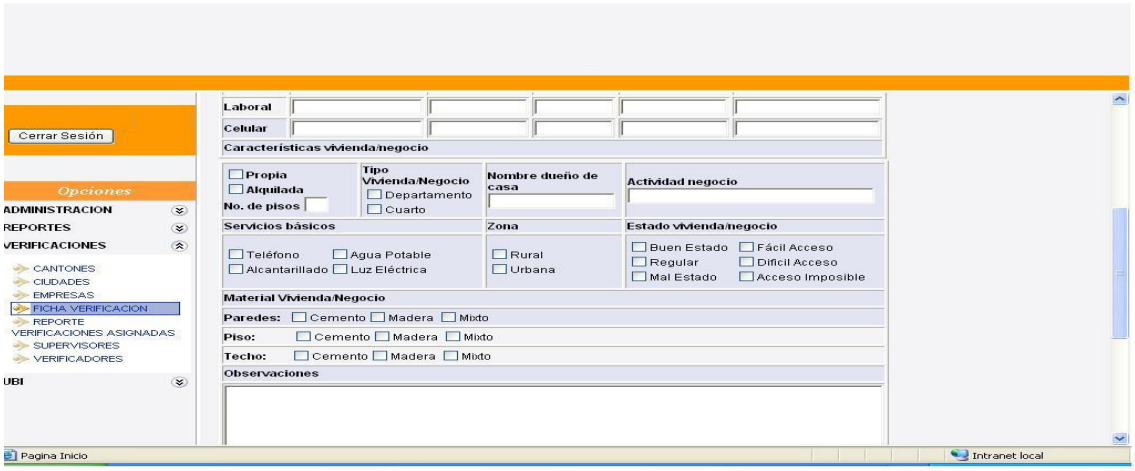
Caso de prueba: F7 Generación fichas
Precondiciones: Administraciones deben estar implementadas Pantalla principal

ENTRADAS	RESULTADOS ESPERADOS	CASO USO
Selecciona opción Generar fichas	Desplegar plantilla de fichas	F7
Selecciona opción para imprimir		
correcto	Se imprimen fichas	
incorrecto	Mensaje de error	F7.1, F7.2

Tabla 4.8: Caso de prueba del módulo Generación de Fichas

4.8 CASO DE USO F7: Pantalla Principal





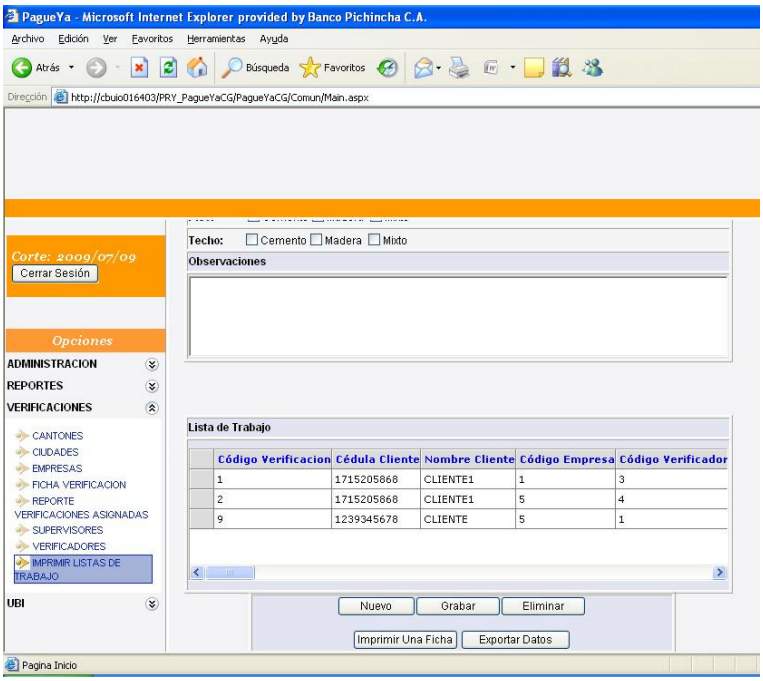


Figura 4.26: Resultados esperados de ingresar al módulo de fichas de verificación

4.8.1 CASO DE USO F7.1: Imprimir una ficha:

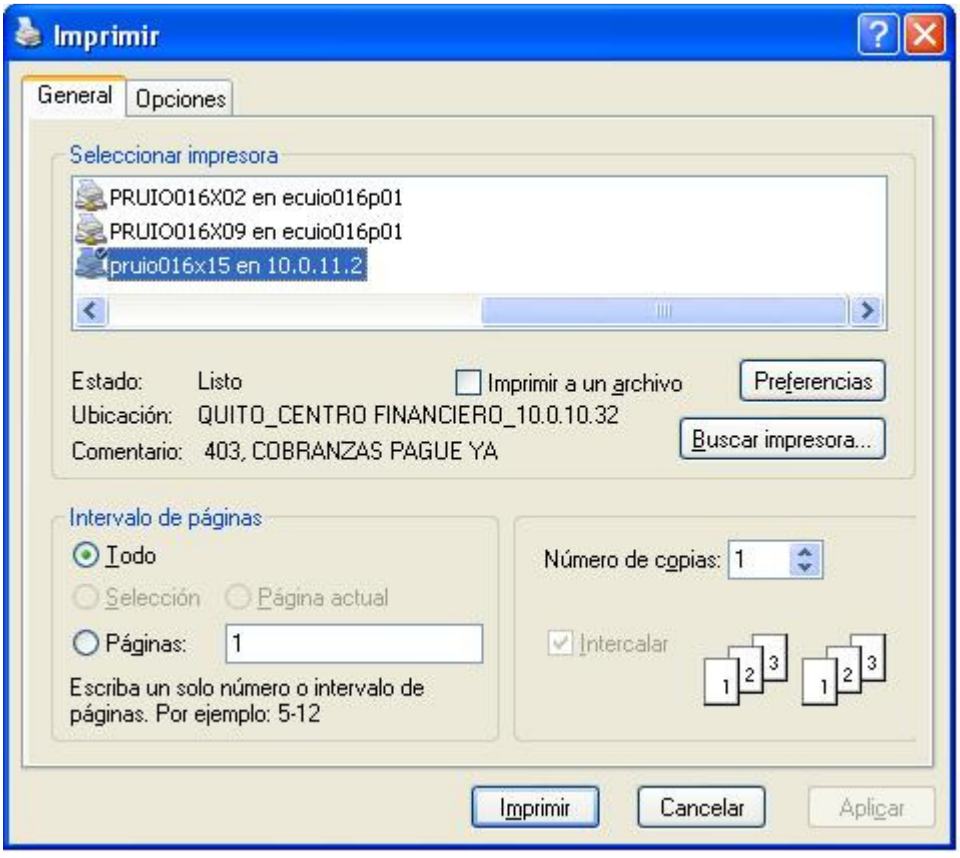


Figura 4.27: Resultados esperados de imprimir una ficha de verificación

4.8.2 CASO DE USO F7.2:

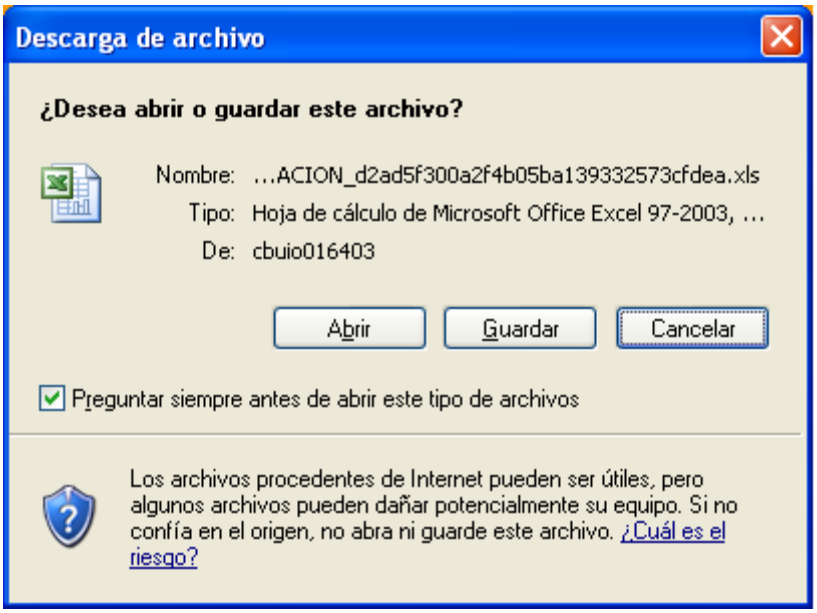


Figura 4.28: Resultados esperados de imprimir varias fichas de verificación

El siguiente módulo, de acuerdo al diagrama de casos de uso es la generación de listas de trabajo:

Caso de prueba: F8 Generación listas de trabajo
Precondiciones: Administraciones deben estar implementadas Pantalla principal

ENTRADAS	RESULTADOS ESPERADOS	CASO USO
Selecciona opción Fichas de verificación	Desplegar plantilla de fichas	F8
Presenta información de listas de trabajo	Datos ingresados reflejados en la consulta	F8.1

Tabla 4.9: Caso de prueba del módulo Generación de Listas de trabajo

4.9CASO DE USO F8: Pantalla principal

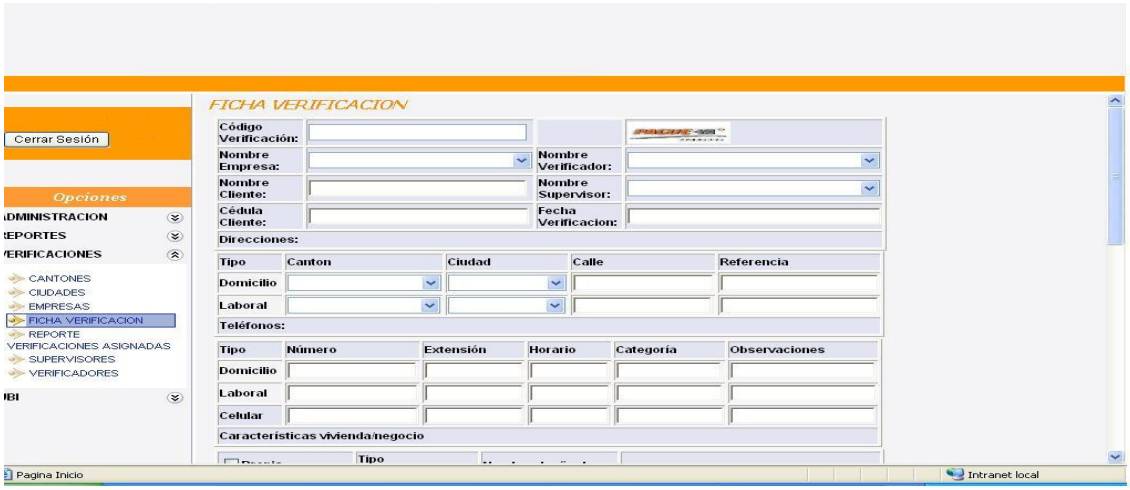


Figura 4.29: Resultados esperados de ingresar al módulo de fichas de verificación

4.9.1 CASO DE USO F8.1: Prersentado la información de las listas de trabajo

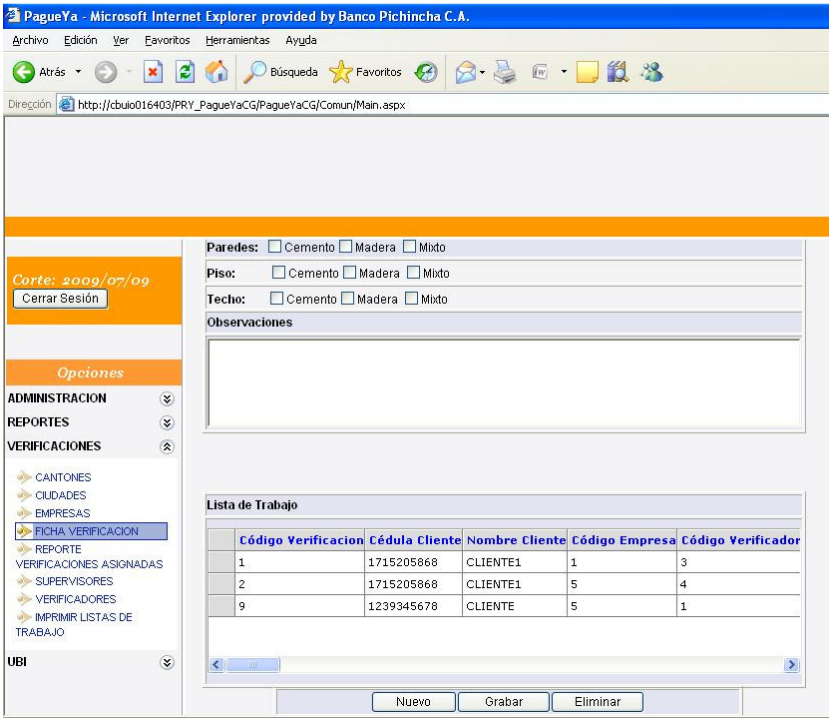


Figura 4.30: Resultados esperados al presentar las listas de trabajo

El siguiente módulo, de acuerdo al diagrama de casos de uso es la generación de reportes

Caso de prueba: F9 Generación reportes
Precondiciones: Administraciones deben estar implementadas Pantalla principal

ENTRADAS	RESULTADOS ESPERADOS	CASO USO
Selecciona opción Generar reportes	Desplegar plantilla de fichas	F9
Ingresa criterio de búsqueda	Datos ingresados reflejados en la consulta	F9.1

Tabla 4.10: Caso de prueba del módulo Generación de Reportes

4.10 CASO DE USO F9: Pantalla Principal

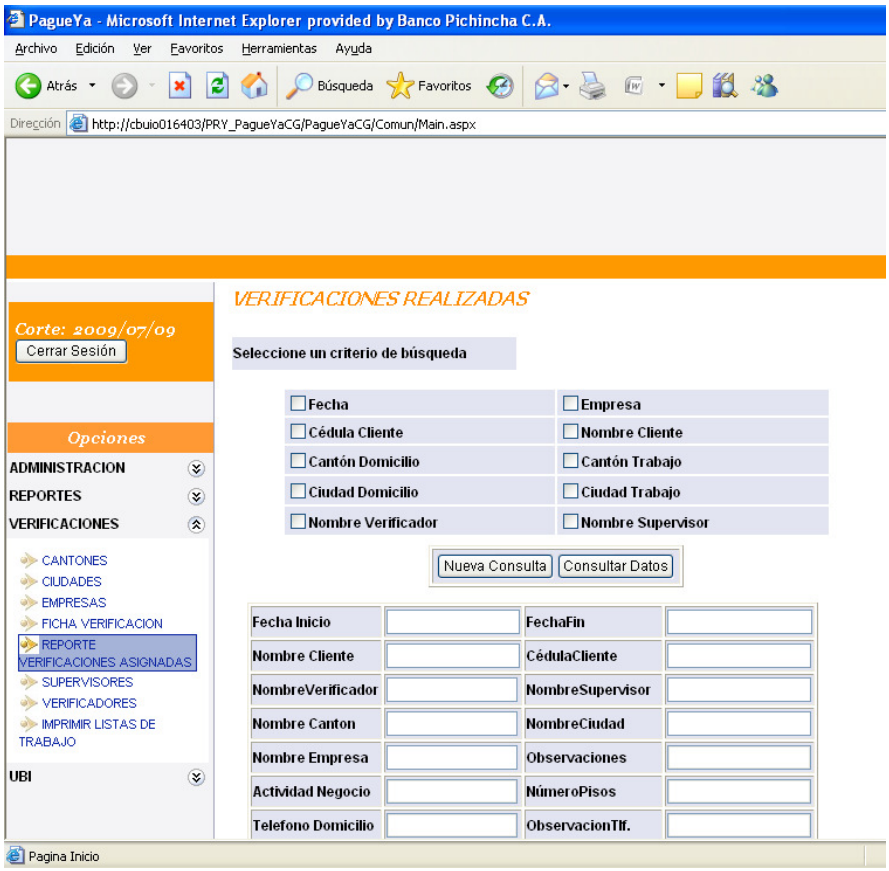


Figura 4.31: Resultados esperados de imprimir varias fichas de verificación

4.10.1 CASO DE USO F9.1: Consulta de verificaciones realizadas

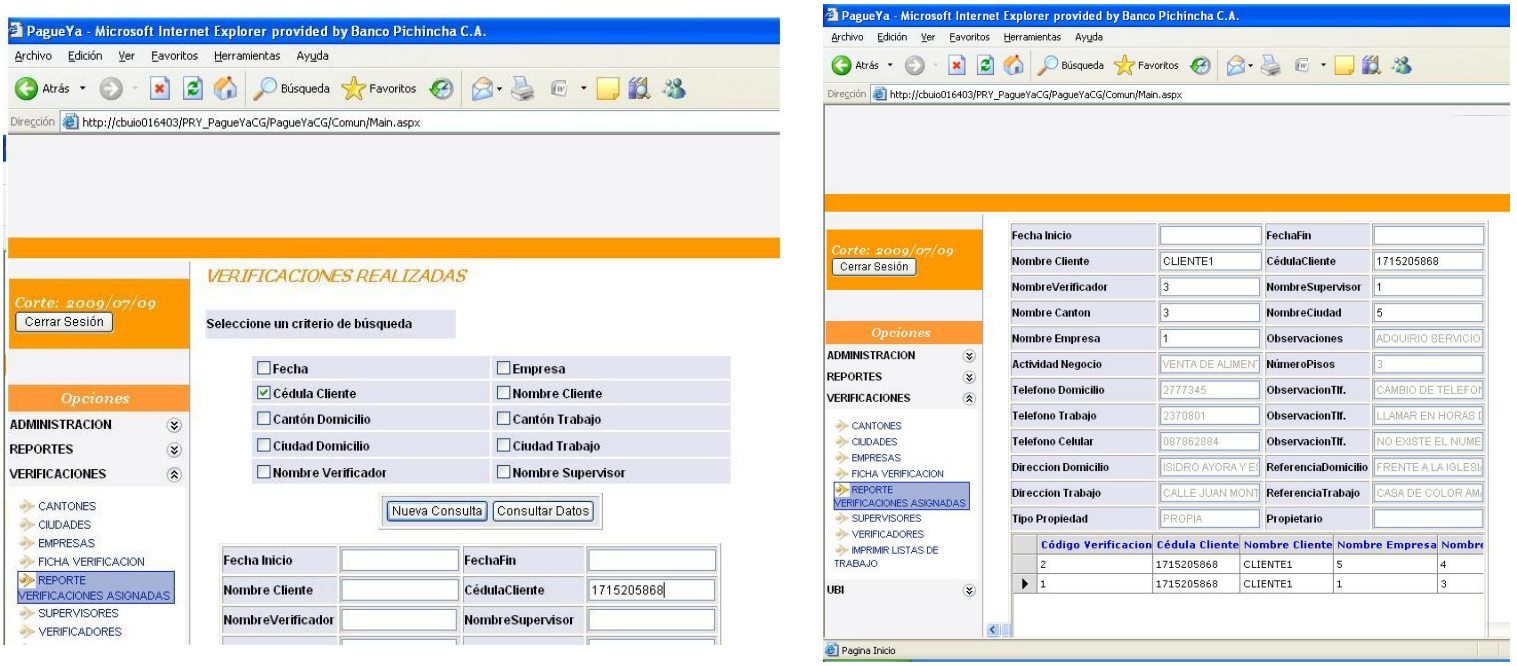


Figura 4.32: Resultados esperados de realizar una consulta de verificaciones realizadas

5 CAPÍTULO 5 Conclusiones y recomendaciones

Después de haber finalizado el presente trabajo y haber realizado pruebas sobre el mismo, se pudieron obtener las siguientes conclusiones:

1. La información demográfica actualizada es muy importante para las empresas de cobranza ya que de esta manera se facilita localizar a los clientes morosos.
2. Generar un sistema de verificaión de información demográfica, facilita a las empresas de cobranza el manejo de la información de sus clientes ya que de esta manera es mas fácil consultar los datos de los clientes para realizar las visitas.
3. Levantar la información de los requerimientos del sistema fue una fase muy importante ya que fue ahí donde se comprendieron las diferentes funcionalidades que el sistema debía tener.
4. La definición de estándares de calidad y desarrollo facilitó la implementación del sistema ya que de esta manera es muy fácil comprender para qué sirve cada función y las variables dentro del sistema.
5. La documentación y elaboración de diagramas es una de las tareas más importantes dentro del proceso de desarrollo ya que de esta manera, si en el futiro se desea realizar cambios en el sistema, basta con acudir a los documentos para comprender como está estructurado cada módulo del sistema.
6. El desarrollo de un plan de pruebas del sistema, permitió detectar ciertos errores generados al momento de integrar los módulos.

7. Desarrollar el sistema bajo un ciclo de vida, permite avanzar de una manera ordenada y si por alguna razón se presentan problemas en las diferentes fases, se puede regresar a las fases anteriores para corregirlos a tiempo.
8. Aplicar el proceso de desarrollo de software al momento de realizar un producto de software, nos permite utilizar estándares de calidad para garantizar que el producto final cumpla con los requerimientos definidos por parte del cliente.
9. Utilizar una arquitectura de tres capas nos permite visualizar de una mejor manera, como se realiza la transmisión de la información desde la capa de manejo de datos, hasta la capa de interface de usuario.
10. Utiliazar el reuso entre las clases, facilita el trabajo de programación ya que el desarrollo de las funciones que se utilizan varias veces, se lo realiza una sola vez, y luego se llaman a los métodos de estas clases, en los demás módulos.

Se pueden obtener las siguientes recomendaciones para que el sistema funcione correctamente:

1. Se recomienda que, ántes de hacer funcionar el sistema en una computadora, revisar que ésta tenga todos los requerimientos para que la aplicación funcione correctamente.
2. Se recomienda que se tenga un servidor para la base de datos y un servidor para la aplicación ya que si es necesario realizar cambios ya sea en la base o en la aplicación, el impacto no sea tan grande.

BIBLIOGRAFÍA

Libros:

- Bona G., Leticia Desarrollo en tres capas .NET, 2008 ©LGB – Analista informática
- Humphrey, Watts S. Introduction to the Team Software $Process^{SM}$ Massachusetts, ADDISON-WESLEY,2000
- Pressman, Roger S. Software Engineering, cuarta edición, Mc Graw Hill, 1998
- Software Engineering Standars Committee of the IEEE Computer Society, IEEE Recommended Practice for Software Requirements Specifications, 1998

Internet:

- Alex Solano: http://www.netveloper.com/contenido2.aspx?IDC=67_0&IDP=4&P=64 Acceso: 2010-03-13
- © 2001-2010, R.S. Pressman & Associates, Inc. http://www.rspa.com Acceso: 2010-03-13
- © 2008: http://alejandria.nidaval.com/scripts/Editorial.dll?SE=2_1_0_T4_A583_78 Acceso: 2010-03-13
- http://www.unalmed.edu.co/~mstabare/diseno_logico.htm Acceso: 2010-03-13
- De Miguel y P. Martínez: http://basesdatos.uc3m.es/fileadmin/Docencia/DBD/Curso0607/Teoria/MODELO_ER.pdf Acceso: 2010-03-16
- Instituto Nacional de Estadística e informática: http://www.ongei.gob.pe/publica/metodologias/Lib5011/n00.htm Acceso: 2010-03-16
- http://es.wikipedia.org/wiki/Diagrama_de_secuencia Acceso: 2010-03-16
- http://es.wikipedia.org/wiki/Diagrama_de_casos_de_uso Acceso: 2010-03-16
- http://es.wikipedia.org/wiki/Diagrama_de_paquetes Acceso: 2010-03-16
- http://es.wikipedia.org/wiki/Diagrama_de_clases Acceso: 2010-03-16

- Copyright © 2009 QUE INFORMATICA!: http://www.que-informatica.com/index.php/software/uml-lenguaje-unificado-de-modelado/ Acceso: 2010-03-17
- © 2008 - 2009 Kernel Error V 2.0: http://kernelerror.net/programacion/php/arquitectura-3-capas/ Acceso: 2010-03-17
- Newcomlab.com © 2005: http://www.newcomlab.com/servicios/b2c_arquitectura.asp Acceso: 2010-03-17
- http://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas Acceso: 2010-03-17
- http://www.mtbase.com/productos/modelamientometadatos/powerdesigner Acceso: 2010-04-17
- © Copyright 2009, Sybase Inc.: http://www.sybase.com/products/modelingdevelopment/powerdesigner Acceso: 2010-04-20
- © Copyright 2009, Sybase Inc.: http://www.sybase.com/products/modelingdevelopment/powerdesigner/datamodeling Acceso: 2010-04-20
  http://en.wikipedia.org/wiki/Data_modeling Acceso: 2010-04-25
- http://wwwhome.cs.utwente.nl/~tcm Acceso: 2010-04-25
- http://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado 2010-04-29
- Wikipedia Foundation Inc: http://en.wikipedia.org/wiki/Microsoft_SQL_Server#Data_storage Acceso: 2010-05-03
- Wikipedia Foundation Inc: http://es.kioskea.net/contents/genie-logiciel/cycle-de-vie.php3 Acceso: 2010-05-03
- Wikipedia Foundation Inc: http://es.wikipedia.org/wiki/ASP.NET Acceso: 2010-05-15
- Wikipedia Foundation Inc: http://es.wikipedia.org/wiki/Active_Server_Pages Acceso: 2010-05-15
- Wikipedia Foundation Inc: http://es.wikipedia.org/wiki/CLR Acceso: 2010-05-16
- Wikipedia Foundation Inc: http://es.wikipedia.org/wiki/Desarrollo_en_cascada Acceso: 2010-05-15
- Wikipedia Foundation Inc: http://es.wikipedia.org/wiki/Framework Acceso: 2010-05-15
- Wikipedia Foundation Inc: http://es.wikipedia.org/wiki/Microsoft_.NET Acceso: 2010-05-17
- Wikipedia Foundation Inc: http://es.wikipedia.org/wiki/Microsoft_SQL_Server Acceso: 2010-05-17
- Wikipedia Foundation Inc: http://es.wikipedia.org/wiki/Object_Linking_and_Embedding Acceso: 2010-05-17
- Wikipedia Foundation Inc: http://es.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol Acceso: 2010-05-17
- © 2009 Microsoft Corporation: http://msdn.microsoft.com/en-us/sqlserver/bb671246.aspx Acceso: 2010-05-17
- © 2008 Iván Romero: http://www.ivanromero.es/proyecto/metodologia.php Acceso: 2010-05-17
- Lambda Servicios Informáticos: http://www.lambdasi.com.ar/textocomp.asp?id=430 Acceso: 2010-05-17
- Lenguajes de Programación © 2009: http://www.lenguajes-de-programacion.com/programacion-orientada-a-objetos.shtml Acceso: 2010-05-17
- © 2009 Microsoft: http://www.microsoft.com/Sqlserver/2005/en/us/express.aspx Acceso: 2010-05-17
- © 1999-2009 Programación en castellano.http://www.programacion.com/asp/ Acceso: 2010-05-18
- www.superban.gov.ec/medios/.../downloads/.../notas_tecnicas_4.doc Acceso: 2010-05-18
- http://web.superban.gov.ec/medios/PORTALDOCS/downloads/articulos_financieros/Estudios%20Sectoriales/comportamiento_sectorial_bancos_jun_09.pdf Acceso: 2010-05-18

Cronograma

Nombre de tarea	Duración	Trabajo	Comienzo	Fin
Ejecución tecnología módulo de Verificación	**80 días**	**640 horas**	**2009/06/09**	**2009/10/13**
ANALISIS DE REQUERIMIENTOS DEL SISTEMA	**4 días**	**32 horas**	**2009/06/09**	**2009/06/12**
Análisis de situacióin actual	1 día	8 horas	2009/06/09	2009/06/09
Levantamiento de requerimientos del sistema	1 día	8 horas	2009/06/10	2009/06/10
Genreación de diagramas de proceso	0.5 días	4 horas	2009/06/11	2009/06/11
Generación de diagramas de clase	0.5 días	4 horas	2009/06/11	2009/06/11
Generación de diagramas de casos de uso	0.5 días	4 horas	2009/06/12	2009/06/12
Generación de diagramas de secuencia	0.5 días	4 horas	2009/06/12	2009/06/12
DISEÑO DEL SISTEMA	**3 días**	**24 horas**	**2009/06/15**	**2009/06/17**
Modelo Conceptual	2 días	16 horas	2009/06/15	2009/06/16
Modelo Físico	1 día	8 horas	2009/06/17	2009/06/17
DESARROLLO DEL SISTEMA	**61 días**	**488 horas**	**2009/06/18**	**2009/09/25**
ADMINISTRACIÓN DE CLIENTES	8 días	64 horas	2009/06/18	2009/06/27
Ingresar un nuevo cliente	2 días	16 horas	2009/06/18	2009/06/19
Actualizar datos del cliente	2 días	16 horas	2009/06/22	2009/06/23
Consulta de clientes por cédula o RUC y por nombre	2 días	16 horas	2009/06/24	2009/06/25
Construcción de interfaces de usuario	1 días	8 horas	2009/06/26	2009/06/26
Pruebas unitarias de administración de clientes	1 día	8 horas	2009/06/29	2009/06/29
ADMINISTRACION DE CANTONES	8 días	64 horas	2009/06/30	2009/07/13
Ingresar un nuevo cantón	2 días	16 horas	2009/06/30	2009/07/02
Actualizar datos del cantón	2 días	16 horas	2009/07/03	2009/07/06
Consulta de cantones por nombres	2 días	16 horas	2009/07/07	2009/07/08
Construcción de interfaces de usuario	1 día	8 horas	2009/07/09	2009/07/10
Pruebas unitarias de administración de cantones	1 día	8 horas	2009/07/13	2009/07/13
ADMINISTRACION DE CIUDADES	8 días	64 horas	2009/07/14	2009/07/27
Ingresar una nueva ciudad	2 días	16 horas	2009/07/14	2009/07/15
Actualizar datos de la ciudad	2 días	16 horas	2009/07/16	2009/07/17
Consulta de ciudades por nombre	2 días	16 horas	2009/07/20	2009/07/21
Consulta de ciudades por cantón	2 días	16 horas	2009/07/22	2009/07/23
Construcción de interfaces de usuario	1 día	8 horas	2009/07/24	2009/07/24
Pruebas unitarias de administración de ciudades	1 día	8 horas	2009/07/27	2009/07/27
ADMINISTRACION DE SUPERVISORES	8 días	64 horas	2009/07/28	2009/08/06
Ingresar un nuevo supervisor	2 días	16 horas	2009/07/28	2009/07/29
Actualizar datos del supervisor	2 días	16 horas	2009/07/30	2009/07/31
Consulta de supervisores por cédula y por nombre	2 días	16 horas	2009/08/03	2009/08/04
Construcción de interfaces de usuario	1 día	8 horas	2009/08/05	2009/08/05
Pruebas unitarias de administración de supervisores	1 día	8 horas	2009/08/06	2009/08/06
ADMINISTRACION DE VERIFICADORES	8 días	64 horas	2009/08/07	2009/08/20
Ingresar un nuevo verificador	2 días	16 horas	2009/08/07	2009/08/10
Actualizar datos del verificador	2 días	16 horas	2009/08/11	2009/08/12
Consulta de verificadores por nombre y por cédula	2 días	16 horas	2009/08/13	2009/08/14
Consulta de ciudades por asesor	2 días	16 horas	2009/08/17	2009/08/18

Construcción de interfaces de usuario	1 día	8 horas	2009/08/19	2009/08/19
Pruebas unitarias de administración de verificadores	1 día	8 horas	2009/08/20	2009/08/20
ADMINISTRACION DE PERFILES DE USUARIO	8 días	64 horas	2009/08/21	2009/09/01
Crear un nuevo perfil	2 días	16 horas	2009/08/21	2009/08/24
Consultar usuarios perfil por nombre	2 días	16 horas	2009/08/25	2009/08/26
Consulta usuarios perfil por perfil	2 días	16 horas	2009/08/27	2009/08/28
Construcción de interfaces de usuario	1 día	8 horas	2009/08/31	2009/08/31
Pruebas unitarias de administración de perfiles de usuario	1 día	8 horas	2009/09/01	2009/09/01
GENERACIÓN DE FICHAS DE VERIFICACIÓN	6 días	48 horas	2009/09/02	2009/09/09
Generación para una ficha	2 días	16 horas	2009/09/02	2009/09/03
Generación para varias fichas	2 días	16 horas	2009/09/04	2009/09/07
Construcción de interfaces de usuario	1 día	8 horas	2009/09/08	2009/09/08
Pruebas unitarias de generación de fichas de verificación	1 día	8 horas	2009/09/09	2009/09/09
GENERACIÓN DE LISTAS DE TRABAJO	3 días	24 horas	2009/09/11	2009/09/15
GENERACIÓN DE REPORTES	4 días	32 horas	2009/09/16	2009/09/25
Verificaciones realizadas	3 días	24 horas	2009/09/22	2009/09/24
Construcción de interfaces de usuario	1 día	8 horas	2009/09/25	2009/09/25
PRUEBAS E IMPLEMENTACIÓN DEL SISTEMA	**10 días**	**80 horas**	**2009/09/28**	**2009/10/09**
Pruebas de integración del sistema	3 días	24 horas	2009/09/28	2009/09/30
Pruebas en embiente de desarrollo	3 días	24 horas	2009/10/01	2009/10/05
Pruebas con el usuario	2 días	16 horas	2009/10/06	2009/10/07
Generación de documentación para el usuario	2 días	16 horas	2009/10/08	2009/10/09
CONCLUSIONES Y RECOMENDACIONES	**2 días**	**16 horas**	**2009/10/12**	**2009/10/13**

A continuación se presenta parte del diagrama de Gantt correspondiente al proyecto de verificaciones:

Dias Duracion	Dia Termina	Actividad o Tarea
84	84	**MODULOS DE VERIFICACIONES**
4	4	**ANÁLISIS DE REQUERIMIENTOS DEL SISTEMA**
1	1	Análisis de situación actual
1	2	Levantamiento de requerimientos del sistema
1	3	Generación de diagramas de proceso
1	3	Generación de diagramas de clase
1	4	Generación de diagramas de casos de uso
1	4	Generación de diagramas de secuencia
3	7	**DISEÑO DEL SISTEMA**
2	6	Modelo Conceptual
1	7	Modelo Físico
61	68	**DESARROLLO DEL SISTEMA**
8	15	**ADMINISTRACIÓN DE CLIENTES**
2	9	Ingresar un nuevo cliente
2	11	Actualizar datos del cliente
2	13	Consulta de clientes por cédula o RUC y por nombre
1	14	Construcción de interfaces de usuario
1	15	Pruebas unitarias de administración de clientes
8	23	**ADMINISTRACIÓN DE CANTONES**
2	17	Ingresar un nuevo cantón
2	19	Actualizar datos del cantón
2	21	Consulta de cantones por nombres
1	22	Construcción de interfaces de usuario