UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Interfaz de aplicación para encontrar y resolver inconsistencias de los cierres contables automáticamente

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Guillermo Arconada Rey

ASESOR DE INFORME

Ing. Jorge Alberto Solano Gálvez

Ciudad Universitaria, Cd. Mx., 2018

Contenido

Índice de Figuras

Índice de Tablas

INTRODUCCIÓN

Debido al crecimiento constante de la tecnología en el día a día, en el mundo laboral es muy necesario mantenerse actualizados para lograr aprovechar de mejor manera la tecnología para el crecimiento de una empresa. Claro que la mala actualización o el uso incorrecto de la tecnología puede convertirse contraproducente, causando un bloqueo drástico al crecimiento de la empresa, afectando en las ganancias de ésta y en el creciente desgaste de los empleados al tener un impedimento para realizar su trabajo de la mejor manera y sin interrupciones.

El contenido de este reporte redacta las necesidades de una empresa que, aunque busca actualizarse tecnológicamente, no logra hacerlo de forma correcta debido a la constante rotación del personal en el área de Sistemas, impidiendo tener gente ampliamente preparada para las necesidades que deben cubrir. Por consecuencia, quedaron envueltos en una gran cantidad de aplicaciones que no ayudan al correcto funcionamiento de dicha empresa y por los cuales se vieron obligados a realizar la aplicación mencionada en este reporte.

Se divide en cuatro capítulos; el primer capítulo explica cómo está conformado el equipo de trabajo de la empresa, junto con las actividades que debe de realizar cada área. El segundo capítulo contiene toda la información general de los conceptos que se requirieron para la elaboración de la aplicación, abarcando administración del proyecto, desarrollo y base de datos. Dentro del tercer capítulo se encuentra el mayor peso, donde se explica a detalle cómo fue elaborado el proyecto, las necesidades del cliente, el objetivo de la aplicación, la forma en que se organizó todo el equipo de trabajo, planeación del proyecto, herramientas que se utilizaron y mi participación en éste. Por último, en el capítulo cuatro se explican los resultados que se tuvieron, comparando con la forma de operar de la empresa antes de que estuviera en funcionamiento la aplicación.

Con ayuda del Glosario y los Anexos, el lector puede tener una idea más clara de cómo fue la realización del proyecto y la magnitud de éste, ayudando con conceptos de palabras de uso común en el proyecto, así como diagramas y códigos para conocer la planeación y estructura de forma general.

CAPÍTULO 1: ORGANIGRAMA

Para la realización de los distintos proyectos, participé con un equipo de trabajo dividido en varias áreas, esto para agilizar y optimizar los tiempos al hacer que las personas involucradas trabajaran con sus conocimientos más fuertes.

El equipo de trabajo fue algo extenso puesto que trabajaron tres áreas en conjunto (Figura 1.1), para cada una de las áreas, el equipo estaba conformado como se muestra a continuación:

- **Sistemas:**
  - Equipo de Base de Datos
  - Equipo de Back-End
  - Equipo de Front-End
  - Equipo de Documentación
  - Equipo de Testing
- **Contabilidad:**
  - Contadores Note Balance
  - Contadores Base de Transacciones
- **Cartera:**
  - Cartera Note Balance
  - Cartera Base de Transacciones

Cada una de estas áreas contaba con un jefe a cargo y, para el caso de Sistemas, todos le respondían a un Project Manager.

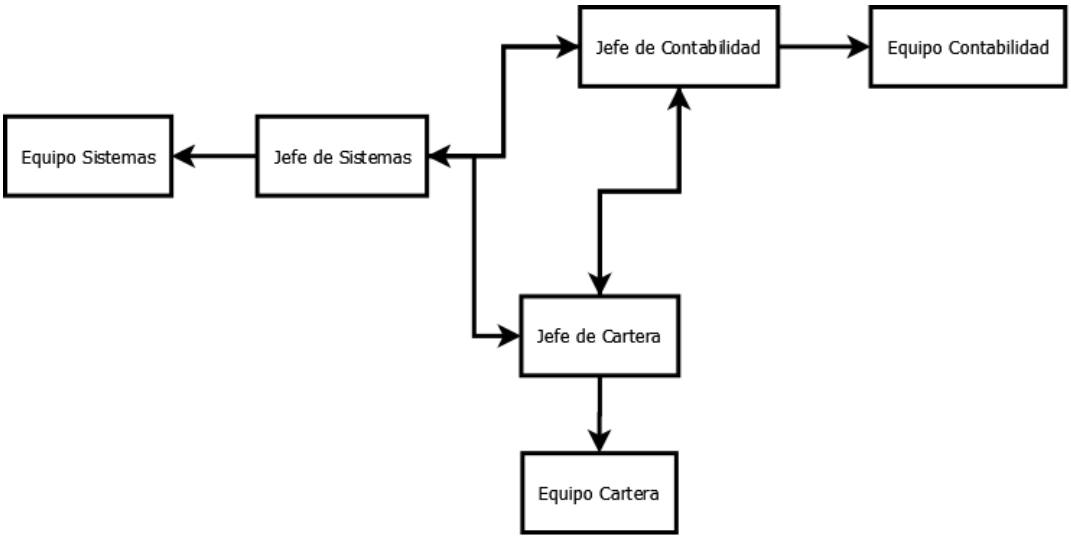


Figura 1.1 Equipo de Trabajo de la Empresa

En cuanto al área de Sistemas, se divide en varios equipos para que se pueda llevar un mejor orden y no se aumente demasiado el tiempo de los proyectos (Figura 1.2). Para este proyecto, el Jefe de Sistemas se mantiene informado con el respectivo Project Manager.

Existe un Project Manager por cada proyecto y, a su vez, sus respectivos equipos de Base de Datos, Desarrolladores, Documentación y Testing. El Project Manager es quien se encarga en liderar el proyecto, haciendo una estimación de tiempos y división de tareas hacia los miembros de cada equipo según convenga para una mejor elaboración del proyecto. Una vez tenga los tiempos y la división de tareas, le informa al jefe de cada equipo lo que le corresponde y, de ahí, se deberá de mantener en una constante comunicación por si ocurre algún imprevisto o sólo para saber el avance que se lleva y así se le pueda informar al Jefe de Sistemas.

Entre los jefes de equipo a los que les informa el Project Manager, se encuentra el analista de Base de Datos, se encarga de liderar al equipo de Base de Datos, además de configurar y monitorear la base de datos en todo momento ya que es una parte muy importante y delicada del proyecto, puesto que es donde se reunirá toda la información con la cual se trabajará. Una vez montado y configurado la Base de Datos, el analista de Base de Datos se reúne con el equipo de Base de Datos para elaborar el Diagrama Entidad-Relación y dividir las actividades, dejando toda la parte de seguridad para él, mientras que el resto del equipo de Base de Datos se ocupa de los procesos que se van a requerir para el proyecto.

Cuando el Project Manager le informa al Desarrollador Sr. sus actividades, éste debe de revisar las tecnologías que serán más útiles a emplear según el proyecto y, una vez decidido, le informa a todo su equipo de dichas actividades, separándolas en Front-End y Back-End. Dependiendo el tipo de proyecto, el equipo de Front-End o Back-End será quien tenga una mayor carga de trabajo, en caso de que las tecnologías a utilizar sean aquellas que los usuarios puedan ver (Java Script, HTML, CSS), entonces son actividades correspondientes para el equipo de Front-End; por el contrario, si las tecnologías a usar no puede visualizarlas el usuario ya que funcionan a nivel servidor (C#, Java Angular JS), entonces le corresponde al equipo de Back-End trabajarlos.

Tanto los equipos de Testing como de Documentación, tienen una participación desfasada en los proyectos. El equipo de Documentación es quien inicia al redactar los requerimientos solicitados por el cliente y, después de eso, tendrá que estar esperando a que el equipo de Testing realice las pruebas necesarias de lo que vayan trabajando los otros tres equipos (Base de Datos, Front-End y Back-End) en conjunto; ya que sin dichas pruebas, no se tendrá nada concreto que documentar y, por consiguiente, nada que se pueda pasar a producción ni entregar al cliente.

En mi caso, me tocó formar parte del equipo de Back-End, pero con el paso del tiempo del proyecto y para evitar retrasos en la entrega de éste, se me permitió de igual forma participar en los equipos de Front-End y Base de Datos. De esta forma tuve mayor libertad al laborar ya que tuve un mayor acceso a toda la información que se requería, claro que todo con aviso anticipado al superior correspondiente (Desarrollador Sr, Analista de Base de Datos o, en dado caso, Project Manajer).

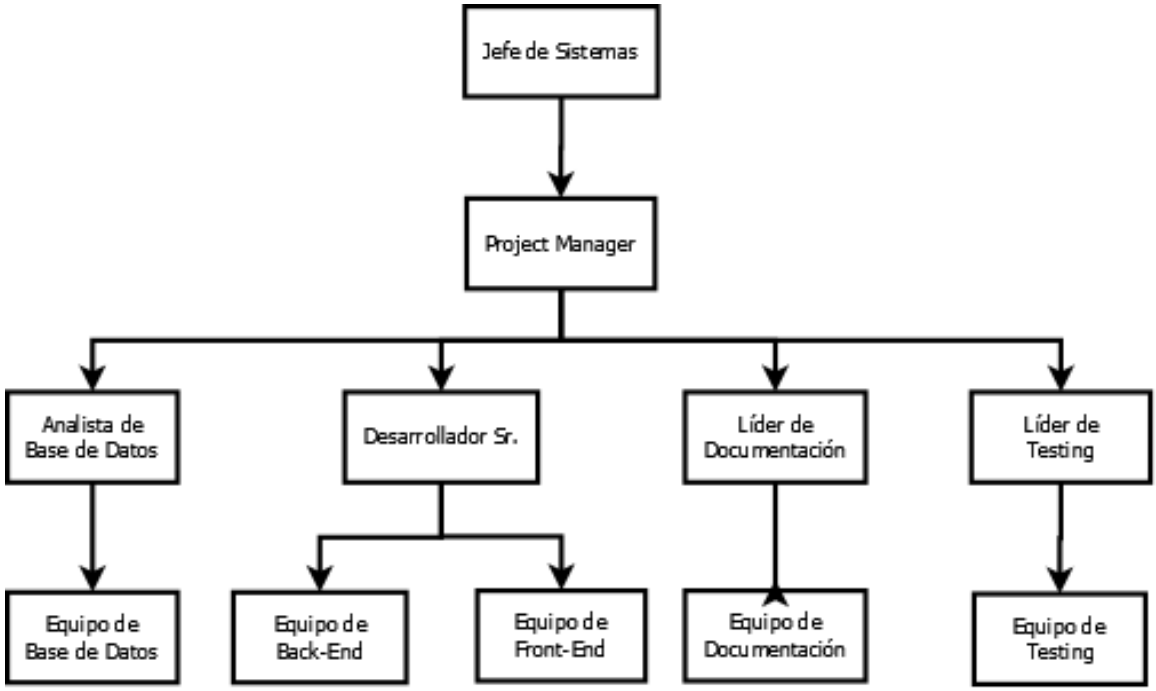


Figura 1.2 Equipo de Sistemas

El área de Cartera (Figura 1.3) es el encargado en revisar que el proceso de las transacciones realizadas por los clientes, así como por el área de Ventas y Call Center, se encuentren de forma correcta, evitando errores que provoquen endeudamientos erróneos, que aparezcan pagos a favor del socio que no corresponden o para saber si se le debe de condenar o agregar pagos a alguien.

A diferencia del área de Sistemas, el área de Cartera sólo se conforma por dos equipos, Notes Balance y Cartera. El equipo de Cartera Notes Balance se encarga en revisar de forma general cómo van los socios en sus pagos, así como penalizar o cancelar a los socios que van rezagados con éstos. Por otro lado, el equipo de Cartera Base de Transacciones es el encargado en revisar más a detalle los movimientos de los socios y, de esta forma, poder condonar algún pago en específico.

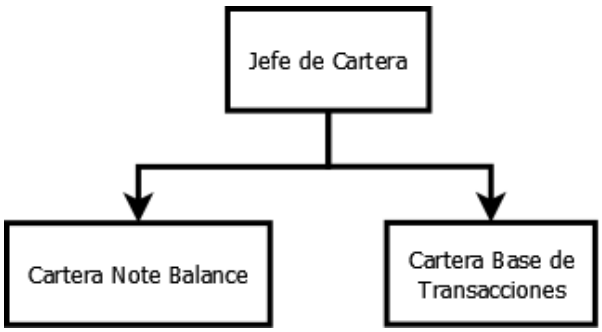


Figura 1.3 Equipo de Cartera

Al igual que el área de Cartera, el área de Contabilidad sólo se compone por dos equipos (Figura 1.4), Notes Balance y Base de Transacciones. Esta área se ocupa en mantener los números de dinero tanto de entrada como salida correctamente, así como vigilar que la contabilidad de todos los tipos de monedas se encuentre correctos y en un promedio parecido al de cada uno de los meses, esto para saber que no están teniendo pérdida de dinero, en caso contrario, se tendrá que revisar cómo hacer que los socios de dicha moneda no dejen de pagar.

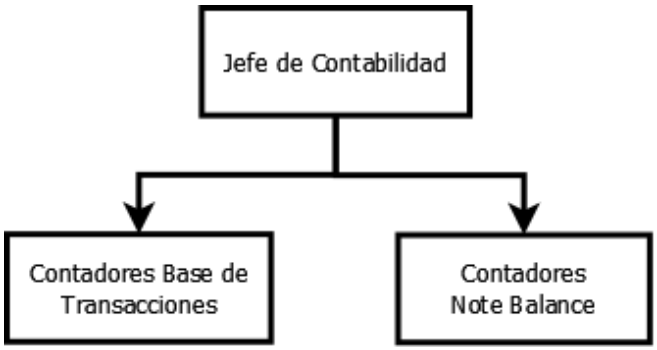


Figura 1.4 Equipo de Contabilidad

CAPÍTULO 2: MARCO TEÓRICO

2.1 Administración del proyecto

La ingeniería de software es una disciplina de ingeniería que se interesa por todos los aspectos de la producción de software, desde las primeras etapas de la especificación del sistema hasta el mantenimiento del sistema después de que se pone en operación.

Los ingenieros de software adoptan en su trabajo un enfoque sistemático y organizado, pues usualmente ésta es la forma más efectiva de producir software de alta calidad. Entre las características que deben de cumplir los sistemas elaborados están mantenimiento, confiabilidad, seguridad, eficiencia y aceptabilidad [1].

El software debe escribirse de tal forma que pueda evolucionar para satisfacer las necesidades cambiantes de los clientes. Éste es un atributo crítico porque el cambio del software es un requerimiento inevitable de un entorno empresarial variable.

La confiabilidad del software incluye un rango de características que abarcan fiabilidad, seguridad y protección. El software confiable no tiene que causar daño físico ni económico, en caso de falla del sistema. Los usuarios malintencionados no deben tener posibilidad de acceder al sistema o dañarlo.

El software no tiene que desperdiciar los recursos del sistema, como la memoria y los ciclos del procesador. Por lo tanto, la eficiencia incluye capacidad de respuesta, tiempo de procesamiento, utilización de memoria, etcétera.

El software debe ser aceptable al tipo de usuarios para quienes se diseña. Esto significa que necesita ser comprensible, utilizable y compatible con otros sistemas que ellos usan.

Busca apoyar el desarrollo de software profesional, en lugar de la programación individual. Incluye técnicas que apoyan la especificación, el diseño y la evolución del programa. Al hablar de ingeniería de software, no sólo se refiere a los programas en sí, sino también a toda la documentación asociada y los datos de configuración requeridos para hacer que estos programas operen de manera correcta. Un sistema de software desarrollado profesionalmente es usualmente más que un solo programa. El sistema por lo regular consta de un número de programas separados y archivos de configuración que se usan para instalar dichos programas. Puede incluir documentación del sistema, que describe la estructura del sistema; documentación del usuario, que explica cómo usar el sistema, y los sitios web para que los usuarios descarguen información reciente del producto.

El enfoque sistemático que se usa en la ingeniería de software se conoce en ocasiones como proceso de software. Un proceso de software es una secuencia de actividades que conducen a la

elaboración de un producto de software. Existen cuatro actividades fundamentales que son comunes a todos los procesos de software, y éstas son:

1. Especificación del software, donde clientes e ingenieros definen el software que se producirá y las restricciones en su operación.
2. Desarrollo del software, donde se diseña y programa el software.
3. Validación del software, donde se verifica el software para asegurar que sea lo que el cliente requiere.
4. Evolución del software, donde se modifica el software para reflejar los requerimientos cambiantes del cliente y del mercado.

La administración de proyectos es la aplicación de conocimientos, habilidades, herramientas y técnicas para realizar proyectos efectiva y eficientemente. Es una capacidad estratégica de las organizaciones, que les permite vincular los resultados de los proyectos con las metas del negocio y así ser más competitivos en sus áreas [2].

Los procesos de la administración de proyectos se clasifican en cinco fases (Figura 2.1):

- Inicio
- Planeación
- Ejecución
- Seguimiento y control
- Cierre

Figura 2.1 Fases de la Administración de Proyectos

En el inicio es donde se realiza un análisis de riesgos, de recursos necesarios y del alcance del proyecto, para determinar su viabilidad.

Durante la planificación del proyecto el Sujeto de Apoyo deberá establecer, coordinar y controlar las variables de alcance, tiempo, costos y calidad del proyecto. A su vez deberá considerar el capital humano que asignará como responsable y administrador del proyecto; así como los riesgos que puede tener el proyecto durante su ejecución.

El alcance del proyecto establece los límites de lo que exactamente se va a hacer. En este sentido, cuando más acotado esté el alcance habrá menos riesgo en el proyecto, pues se conocerá con mayor exactitud lo que está incluido. Para cada objetivo específico se deberá definir lo que será entregado al final del proyecto como prueba del cumplimiento de éste, llamados entregables.

El proyecto debe contar con un plan de trabajo donde se especifiquen la fecha en que arrancará el proyecto, cuando deberá estar terminado, las fechas de entrega de informes tanto técnico como financiero parciales y finales; así como el periodo para ejercer los recursos.

La presentación del presupuesto del proyecto deberá ser programado en términos de las partidas y rubros o conceptos aceptados y establecidos en cada uno de los fondos con los que cuentan.

La fase de ejecución del proyecto corresponde a la realización de las actividades programadas en el plan de trabajo o cronograma establecido durante la planificación del proyecto. Como consecuencia de la ejecución real del proyecto de las actividades y con el propósito de lograr los objetivos planteados en esta fase se debe administrar y/o distribuir los recursos, dar seguimiento a las actividades, ajustar las desviaciones, registrar los avances y recolectar la evidencia de la ejecución.

Con el propósito de asegurar el cumplimiento de los objetivos de los proyectos, se establece como requisito obligatorio que el cliente permita verificar el avance técnico y los resultados del proyecto.

El proyecto entra en la fase de cierre del proyecto una vez que el resultado de las evaluaciones técnicas sea satisfactorio.

A pesar de que la forma de trabajar en todos los proyectos del Área de Sistemas de la empresa están basados en una metodología tradicional, aunque para el caso de este proyecto, combiné dicha metodología con el uso de la metodología ágil haciendo uso de Scrum causando que, a pesar de que se fijaran tiempos y se asignaran actividades para cada uno de los entregables, se siguiera teniendo una constante comunicación con las Áreas de Contabilidad y Cartera (clientes) para corregir rápidamente cualquier error que se encontrara y se tuviera el entregable correcto al momento de llegar a la fecha acordada (con base a la metodología tradicional), aunque significara tener días completos estando en el lugar de trabajo del cliente viendo cómo es que realizaban su trabajo.

Al tratarse de un tema delicado y del cual no se tenía realmente nada conciso en cuanto al funcionamiento, fue requerida una constante comunicación y revisiones continuas para saber si el resultado obtenido nos estaba llevando por un buen camino. El proyecto se terminó desarrollando bajo una metodología ágil, sin embargo, se mantuvieron tiempos para poder realizar todas las actividades del mismo, lo cual nos sirvió para poder conocer las fechas límites de cada objetivo y así no tener atrasos al exceder con los tiempos establecidos (Figura 2.2). Claro que, a pesar de llevar el orden de una correcta administración de proyectos, las fases se ven encimadas puesto que estos tiempos estaban divididos en varios módulos.

Nombre	Duración	Inicio	Fin
⊞Inicio	6d	05/01/2016	13/01/2016
⊞Planificación	22d	14/01/2016	15/02/2016
⊞Ejecución	173d	16/02/2016	14/10/2016
⊞Seguimiento y Control	172d	31/03/2016	28/11/2016
⊞Cierre	143d	23/05/2016	08/12/2016

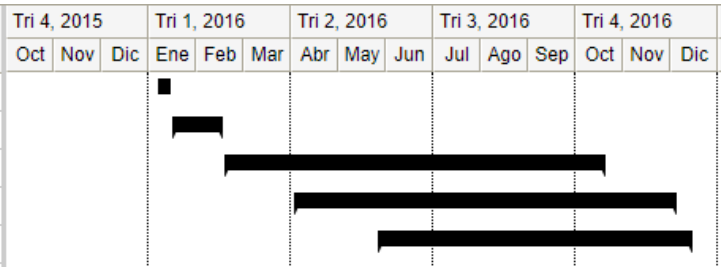


Figura 2.2 Tiempos del proyecto

Para poder tener una buena coordinación en la realización del proyecto y de esta forma poder terminar en tiempo y forma con todos los objetivos, fue requerido tener en un inicio una buena documentación con los requerimientos del proyecto de acuerdo a las necesidades del cliente (Figura 2.3), por ello, se reunieron los Jefes de las Áreas de Sistemas, Contabilidad y Cartera para poner claro el alcance y qué es lo que se quería lograr. Una vez tuvieron dicha reunión, el Jefe de Sistemas se reunió nuevamente con su equipo de trabajo para explicar lo que se tenía que hacer, se aseguraron que todo quedara bien redactado en un documento y se esperó a que lo validaran los Jefes de Contaduría y Cartera puesto que en este caso hicieron la función de cliente. Teniendo esto se le hizo saber al resto del equipo que podían comenzar a trabajar teniendo la confianza de que no existirían atrasos a causa de cambios imprevistos en el proyecto.

Ya que todos los integrantes del equipo contaban con los requerimientos, se procedió a la planeación lógica para llegar al resultado que deseado. Para esto, el equipo de Base de Datos fue el encargado de realizar el flujo de datos y procesos que se requirieron, elaborando un Diagrama Entidad-Relación y, ya teniendo esto, se le informó al equipo de Front-End para que pudieran comenzar a trabajar considerando los campos que debía de contener cada una de las pantallas a realizar y, en paralelo, el equipo de Base de Datos procedió realizando su trabajo.

Conforme el equipo de Front-End terminó las pantallas y el equipo de Base de Datos terminó de hacer las Vistas y los Stored Procedure requeridos para dichas pantallas, se le envió la información al equipo de Back-End para que los pudieran unir y así dar una funcionalidad completa y dinámica a las pantallas. Para este punto del proyecto, los tres equipos (Base de

Datos, Front-End y Back-End) podían trabajar al mismo tiempo para que, de esta forma, se aprovechara lo mejor posible el tiempo que se tenía.

Cuando el equipo de Back-End dio por terminada una pantalla, ésta se pasó a un servidor de pruebas, donde entró el equipo de Testing para revisar que todo funcionara bien al intentar hacer que fallara lo trabajado, no sólo con el flujo ideal que debe de llevar el sistema. En el caso de que se encontraran fallas, se le hacía saber al equipo correspondiente para que hiciera las correcciones y se pudieran hacer pruebas nuevamente; al contrario, en caso de no encontrar fallas, se le informó a todo el equipo del proyecto que fue éxito y así el equipo de Documentación procedió a redactar los manuales necesarios que más adelante se le debían de entregar al cliente.

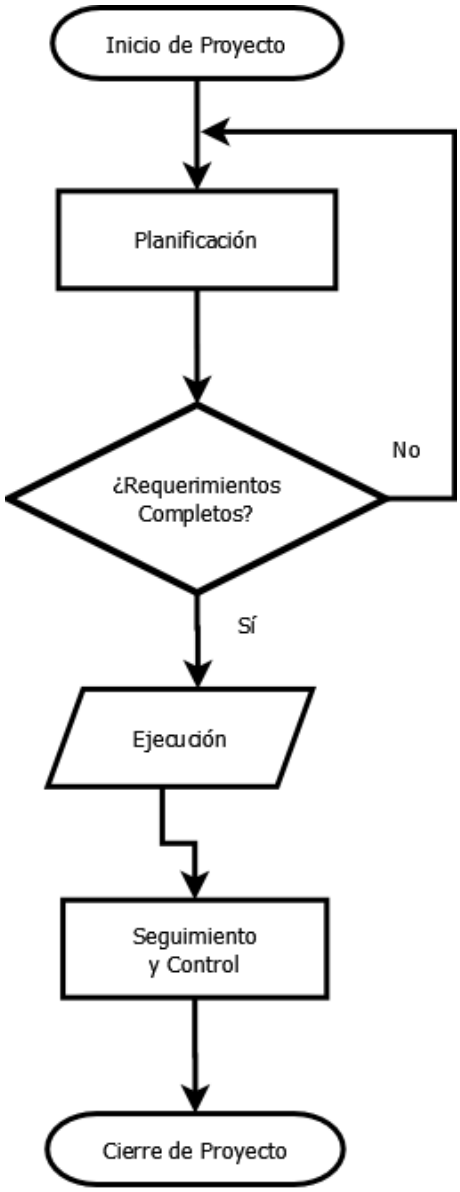


Figura 2.3 Administración del proyecto

Una vez que el equipo de Documentación terminó los manuales requeridos para entregar al cliente, y éstos sean de algún módulo completo, se procedió a colocar el sistema en un área de pre-producción, donde el cliente hizo uso de éste junto con los manuales para revisar que todo se encontraba como se acordó y, en caso de no existir problema alguno, el módulo se liberó y se puso en un área de producción, donde ya no podría ser modificado por el equipo de trabajo.

2.2 Diseño de proyecto

Agrupa el conjunto de principios, conceptos y prácticas que llevan al desarrollo de un sistema o producto de alta calidad, el objetivo del diseño es producir un modelo o representación que tenga resistencia, funcionalidad y belleza.

El modelo del diseño proporciona detalles sobre la arquitectura del software, estructuras de datos, interfaces y componentes que se necesitan para implementar el sistema. También permite modelar el sistema o producto que se va a construir, es el lugar en el que se establece la calidad del software.

El diseño de software comienza una vez que se han analizado y modelado los requerimientos, siempre debe comenzar con el análisis de los datos pues son el fundamento de todos los demás elementos del diseño, el diseño de la arquitectura define la relación entre los elementos principales de la arquitectura de software, los estilos y patrones de diseño de la arquitectura que pueden usarse para alcanzar los requerimientos definidos por el sistema y las restricciones que afectan la forma en que se implementa la arquitectura.

El diseño de interfaz describe la forma en la que el software se comunica con los sistemas que interactúan con él y con los usuarios que lo utilizan. Implica un flujo de información y un tipo específico de comportamiento.

El diseño de software es un proceso iterativo por medio del cual se traducen los requerimientos en un “plano” para construir el software. El diseño se representa en un nivel alto de abstracción en el que se rastrea directamente el adjetivo específico del sistema y los requerimientos más detallados de datos: Funcionamiento y Comportamiento.

2.3 MVC

Es un patrón de software utilizado para implementar sistemas donde se requiere el uso de interfaces de usuario. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos.

La rama de la ingeniería del software se preocupa por crear procesos que aseguren calidad en los programas que se realizan y esa calidad atiende a diversos parámetros que son deseables para todo desarrollo, como la estructuración de los programas o reutilización del código, lo que debe influir positivamente en la facilidad de desarrollo y mantenimiento.

El MVC o Modelo-Vista-Controlador es un patrón de arquitectura de software que, utilizando 3 componentes (Views, Models, Controllers) separa la lógica de la aplicación de la lógica de la vista en una aplicación (Figura 2.4). Es una arquitectura importante puesto que se utiliza tanto en componentes gráficos básicos hasta sistemas empresariales; la mayoría de los frameworks modernos utilizan MVC (o alguna adaptación) para la arquitectura, entre ellos se pueden mencionar Django y Angular JS [3].

- **Modelo:** Es la capa donde se trabaja con los datos, por tanto, contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos se tendrán habitualmente en una base de datos, por lo que en los modelos se tendrán todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.
- **Vista:** Como su nombre nos hace entender, contiene el código de la aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. Sólo se tienen los códigos que nos permite mostrar la salida.
- **Controlador:** contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc. En realidad es una capa que sirve de enlace entre las visar y los modelos, respondiendo a los mecanismos que puedan requerir para implementar las necesidades de la aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

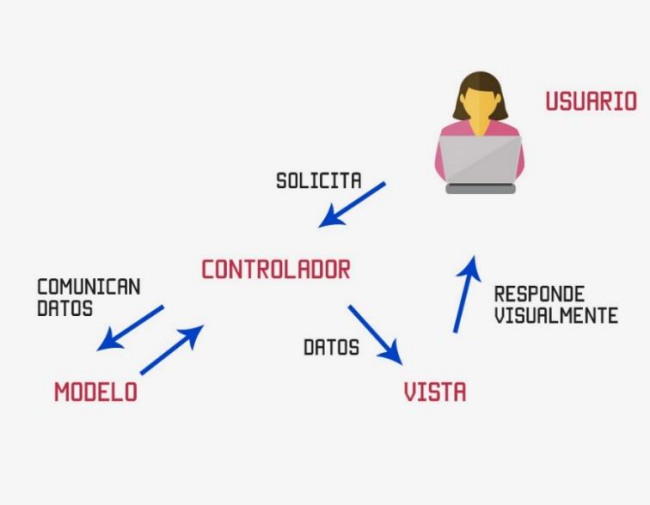


Figura 2.4 Patrón MVC

Los controladores, con su lógica de negocio, hacen de puente entre los modelos y las vistas. Pero además en algunos casos los modelos pueden enviar datos a las vistas. El flujo de trabajo característico en un esquema MVC paso a paso es:

1. El usuario realiza una solicitud al sitio web. Generalmente estará desencadenada por acceder a una página de nuestro sitio. Esa solicitud le llega al controlador.
2. El controlador comunica tanto con modelos como con vistas. A los modelos les solicita datos o les manda realizar actualizaciones de los datos. A las vistas les solicita la salida correspondiente, una vez se hayan realizado las operaciones pertinentes según la lógica de negocio.
3. Para producir la salida, en ocasiones las vistas pueden solicitar más información a los modelos. En ocasiones, el controlador será el responsable de solicitar todos los datos a los modelos y de enviarlos a las vistas, haciendo de puente entre unos y otros. Ninguno de los casos es complicado al momento de programar, todo depende de nuestra implementación.
4. Las vistas envían al usuario la salida. Aunque en ocasiones esa salida puede ir de vuelta al controlador y sería éste el que hace el envío al cliente.

Cabe mencionar que la comparación de ventajas y desventajas de MVC puede ser un tema muy subjetivo y se puede prestar como tema de debate, ya que en términos generales la balanza se inclina a favor del MVC en vez de en su contra.

Ventajas

- La separación del Modelo de la Vista.
- Es mucho más sencillo agregar múltiples representaciones de los mismos datos o información.
- Facilita agregar nuevos tipos de datos según sea requerido por la aplicación ya que son independientes del funcionamiento de las otras capas.
- Crea independencia de funcionamiento.
- Facilita el mantenimiento en caso de errores.
- Ofrece maneras más sencillas para probar el correcto funcionamiento del sistema.
- Permite el escalamiento de la aplicación en caso de ser requerido.

Desventajas

- La separación de conceptos en capas agrega complejidad al sistema.
- La cantidad de archivos a mantener y desarrollar se incrementa considerablemente.

- La curva de aprendizaje del patrón de diseño es más alta que usando otros modelos más sencillos.

Originalmente MVC fue desarrollado para aplicaciones de escritorio, ha sido ampliamente adaptado como arquitectura para diseñar e implementar aplicaciones Web. Los frameworks se diferencian básicamente en la interpretación de como las funciones MVC se dividen entre cliente y servidor, inicialmente MVC estaban completamente alojados en el servidor. Ahora existen frameworks como JavaScriptMVC, Backbone o jQuery que permiten que ciertos componentes MVC se ejecuten parcial o totalmente en el cliente.

2.4 Web Services

Un servicio Web o WebService es un servicio ofrecido por una aplicación que expone su lógica a clientes de cualquier plataforma mediante una interfaz accesible a través de la red utilizando tecnologías (protocolos) estándar de Internet.

La idea de los servicios Web es un conjunto de componentes que ofrecen una serie de servicios, como el acceso a datos, la impresión de informes, el diseño de tablas, etc., aunque éstos no tienen por qué estar en el mismo equipo que el cliente y además son accedidos a través de un servidor Web y de un modo independiente de la plataforma, utilizando protocolos estándar (HTTP, SOAP, WSDL, UDDI) [4].

Para crear un servicio (Figura 2.5) puede utilizarse cualquiera de los lenguajes disponibles en la plataforma .NET. Una vez creado el servicio, para conseguir que sea accesible por los consumidores, es necesario describirlo utilizando un lenguaje estándar llamado WSDL (Web Service Description Language).

Los clientes del servicio podrán estar creados en cualquier lenguaje y ejecutarse sobre cualquier sistema operativo y hardware, lo único necesario es que sean capaces de obtener y entender la descripción WSDL de un servicio. Un archivo WSDL es, en realidad, un archivo XML en el que se identifica el servicio y se indica el esquema para poder utilizarlo, así como el protocolo o protocolos que es posible utilizar.

Además de describir un servicio para que pueda ser utilizado por los clientes es importante publicar el servicio de modo que pueda ser encontrado por clientes que no conozcan necesariamente el componen que ofrece el servicio, pero que busquen un servicio de sus características. Esto se logra mediante el estándar UDDI (Universal Description Discovery and Integration Registry).

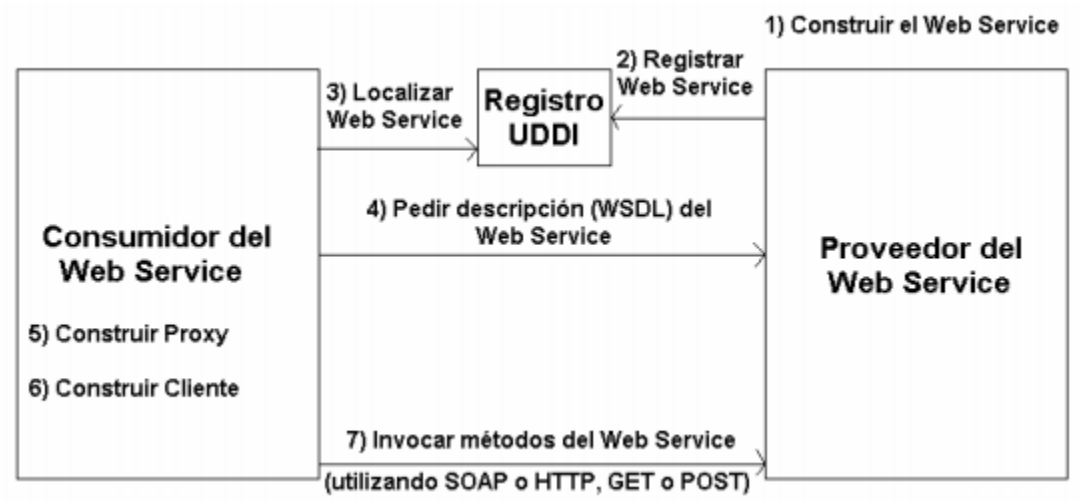


Figura 2.5 Creación y utilización de Web Service

Ventajas

- Ofrecen una "tecnología distribuida de componentes" optimizada.
- Evitan los problemas inherentes a la existencia de firewalls, ya que SOAP utiliza HTTP como protocolo de comunicación.
- Permiten una invocación sencilla de métodos, mediante SOAP.
- Los clientes o "consumidores de servicios" pueden estar en cualquier plataforma (basta con que soporten XML/SOAP, incluso puede sustituirse SOAP por HTTP).
- Permiten centralizar los datos, independientemente de si los WebServices están distribuidos o no.

Desventajas

- Para realizar transacciones no pueden compararse con los estándares abiertos de computación distribuida.
- Su rendimiento es bajo si se compara con otros modelos de computación distribuida.
- Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear la comunicación entre programas.
- Existe poca información de servicios web para algunos lenguajes de programación.

2.5 ETL

Los procesos ETL son un término estándar que se utiliza para referirse al movimiento y transformación de datos. Se trata del proceso que permite a las organizaciones mover datos desde múltiples fuentes, reformatearlos y cargarlos en otra base de datos (denominada data mart o data warehouse) con el objeto de analizarlos. También pueden ser enviados a otro sistema operacional para apoyar un proceso de negocio [5].

En definitiva, el principal objetivo de este proceso es facilitar el movimiento de los datos y la transformación de los mismos, integrando los distintos sistemas y fuentes en la organización moderna (Figura 2.6).

El término ETL corresponde a las siglas en inglés de Extract (Extraer). Transform (Transformar) y Load (Cargar).

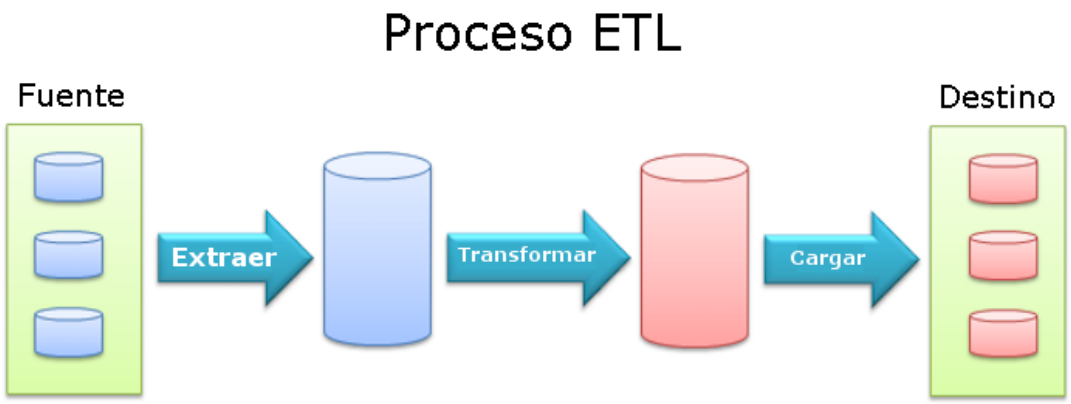


Figura 2.6 Proceso ETL

Fase de Extracción

Para llevar a cabo de manera correcta el proceso de extracción, primera fase del ETL, hay que seguir los siguientes pasos:

- Extraer los pasos desde los sistemas de origen.
- Analizar los datos extraídos obteniendo un chequeo.
- Interpretar este chequeo para verificar que los datos extraídos cumplen la pauta o estructura que se esperaba. Si no fuese así, los datos deberían ser rechazados.
- Convertir los datos a un formato preparado para iniciar el proceso de transformación.

Además, una de las prevenciones más importantes que se deben tener en cuenta durante el proceso de extracción de datos sería el exigir siempre que esta tarea cause un impacto mínimo en el sistema de origen. Este requisito se basa en la práctica ya que, si los datos a extraer son

muchos, el sistema de origen se podría ralentizar e incluso colapsar, provocando que no pudiera volver a ser utilizado con normalidad para su uso cotidiano.

Fase de Transformación

La fase de transformación de un proceso de ETL aplica una serie de reglas de negocio o funciones, sobre los datos extraídos para convertirlos en datos que serán cargados. Estas directrices pueden ser declarativas, pueden basarse en excepciones o restricciones, pero, para potenciar su pragmatismo y eficacia, hay que asegurarse de que sean:

- Declarativas.
- Independientes.
- Claras.
- Inteligibles.
- Con una finalidad útil para el negocio.

Proceso de Carga

En esta fase, los datos procedentes de la fase anterior (fase de transformación) son cargados en el sistema de destino. Dependiendo de los requerimientos de la organización, este proceso puede abarcar una amplia variedad de acciones diferentes.

Existen dos formas básicas de desarrollar el proceso de carga:

- **Acumulación simple:** esta manera de cargar los datos consiste en realizar un resumen de todas las transacciones comprendidas en el periodo de tiempo seleccionado y transportar el resultado como una única transacción hacia el data warehouse, almacenando un valor calculado que consistirá típicamente en un sumatorio o un promedio de la magnitud considerada. Es la forma más sencilla y común de llevar a cabo el proceso de carga.
- **Rolling:** este proceso sería el más recomendable en los casos en que se busque mantener varios niveles de granularidad. Para ello se almacena información resumida a distintos niveles, correspondientes a distintas agrupaciones de la unidad de tiempo o diferentes niveles jerárquicos en alguna o varias de las dimensiones de la magnitud almacenada (por ejemplo, totales diarios, totales semanales, totales mensuales, etc.).

Sea cual sea la manera elegida de desarrollar este proceso, hay que tener en cuenta que esta fase interactúa directamente con la base de datos de destino y, por eso, al realizar esta operación se aplicarán todas las restricciones que se hayan definido en ésta. Si están bien definidas, la calidad de los datos en el proceso ETL estará garantizadas.

CAPÍTULO 3: INTERFAZ DE APLICACIÓN PARA ENCONTRAR Y RESOLVER INCONSISTENCIAS DE LOS CIERRES CONTABLES AUTOMÁTICAMENTE

3.1 Motor de Integridad de Datos Contables

3.1.1 Objetivo

Crear un sistema que ayude a encontrar y resolver las inconsistencias causadas por los demás sistemas utilizados en la empresa, de igual forma, ayudar a agilizar la contabilidad y poder identificar con mayor rapidez las altas y bajas de ventas de la empresa para enfocar la publicidad y mercadotecnia a dichas áreas, para, de esta forma, poder elevar el trato al cliente e impedir que existan bajas de sus socios.

3.1.2 Antecedentes

El cliente, al dedicarse en la venta de tiempo vacacional en distintos países, adquiere una cartera muy amplia de socios y, para poder llevar un buen control de los pagos de membresías y facturas generadas, desarrolló un sistema en el cual puede alojar toda la información y le permite a todas las áreas trabajar en tiempo real.

Con el paso del tiempo, el cliente observó que su sistema tenía un problema muy grande al manejar todo en dólares, lo cual afectó demasiado ya que debían de trabajar con distintas monedas de acuerdo el país del socio. La solución aparente a la que se vieron forzados realizar, fue crear pequeños sistemas para cada tipo de moneda, país, tipo de venta y forma de pago que fueran necesitando y, estos a su vez, conectados al sistema principal para que todos los empleados siguieran trabajando como un solo sistema y vieran todo como en un inicio.

Aunque para las Áreas de Ventas y Call Center, así como para los socios, no veían cambio alguno en el sistema, las Áreas de Contabilidad y Cartera comenzaron a observar un incremento enorme de errores en los números y en las transacciones realizadas al realizar cada cierre de mes, obligándolos a revisar detenidamente las transacciones, anotar los errores y enviarles éstos al Área de Sistemas para que los corrigieran. Esto a su vez, afectó en los tiempos del cierre de mes, al hacer que se invirtieran 8 días para poder terminarlos.

Los contratiempos antes mencionados, afectaron la entrega de reportes tanto para los jefes, como para poder realizar las planeaciones de ventas del siguiente mes. Además, algunas personas del Área de Sistemas se tuvieron que dedicar tiempo completo a estar llenando o corrigiendo toda la información errónea que se encontraba en la base de datos, creando un atraso para otros proyectos al quedarse con menos personas que pudieran dedicarse a éstos, afectando tanto al Área de Sistemas como a las áreas de Contabilidad y Cartera, en sus jornadas laborales, puesto que, al hablar de errores de un nivel muy alto, no podían dejarlos para otros días, por lo que no se podían retirar hasta haber terminado de corregir todos los errores encontrados.

3.1.3 Necesidades del cliente

Debido a todos los problemas generados por el sistema que manejan, además de que no es posible cambiar dicho sistema, requieren de una herramienta que les ayude a recuperar todo el tiempo perdido, así como facilitar para el Área de Sistemas el corregir todos los fallos de los sub-sistemas que usan para el funcionamiento de la empresa y, de esta forma, poder contar con su equipo para otros trabajos y no para quedarse fijos corrigiendo errores diarios.

El sistema debe de realizar todo esto con la finalidad de que los cierres de mes no cuenten con errores, causando que no se demoren en éste y se pueda realizar en menos tiempo, al igual que con las siguientes entregas que se deben de hacer después del cierre, mostrar toda la información correcta para cada una de las áreas involucradas.

El sistema debe permitir mostrar diariamente las inconsistencias que se encuentran, así como el encargado de resolverlas, esto para que el encargado de cada Área pueda acceder y revisar el Reporte de Inconsistencias y así no esperar a que sea cierre de mes para corregir todo. En caso de que los problemas ya hayan sido corregidos desde raíz, la aplicación debe de ser capaz de corregir las inconsistencias restantes de forma automática.

Del mismo modo, para ayudar al trabajo de las Áreas de Contabilidad y de Cartera, todos los reportes con los que trabajan, deberán ser depurados de tal forma que sólo se queden aquellos que realmente utilizan y, en caso de necesitar alguno nuevo, igual deberá de ser agregado con la información que se solicite; todos localizados en la aplicación, pero impidiendo que puedan ver reportes que no les corresponden.

3.2 Participación Profesional

La Administración del Proyecto se llevó a cabo utilizando una metodología Ágil, para poder aprovechar el tiempo ya que existía una forma de trabajar no adecuada por parte de las Áreas de Contabilidad y Cartera. Por esta misma razón, en el Área de Sistemas decidimos trabajar el proyecto haciendo uso de las tecnologías Microsoft Visual Studio 2012 y SQL Server 2010, junto con un patrón MVC; esto para evitar los problemas existentes en el lugar de trabajo iniciando con la administración de servidores, así como de los proyectos que se manejan ya que al contar con una rotación constante de personal, no se lleva un buen control de todo lo que se trabaja causando una mala optimización de recursos.

El patrón MVC fue elegido para poder tener un orden en lo que se iba desarrollando, así las partes del proyecto que ya funcionaran se podrían mantener en pruebas o producción y no existiría el problema de que dejara de funcionar al agregar alguna sección o módulo nuevo a dicho proyecto, puesto que son códigos completamente independientes entre ellos.

Con estas dos cosas resueltas, se nos permitió trabajar entre las Áreas y comenzar a poner orden cada una en lo que realmente necesitan para trabajar, puesto que todos los reportes con los que contaban Contabilidad y Cartera no les sirvieron realmente en agilizar su trabajo. De esta forma se procedió a dar comienzo a todo el proyecto.

Requerimientos

Fue la etapa que requirió de menos tiempo (Figura 3.1), pero una de las más complicadas ya que se debía de ajustar la idea real pedida por el cliente (Áreas de Contabilidad y Cartera) con el presupuesto asignado y con las herramientas de trabajo que contaba el Área de Sistemas. Desafortunadamente surgieron varios problemas desde esta etapa, puesto que lo solicitado por el cliente era demasiado en comparación a lo que se podía hacer, esto debido a que la idea que solicitaron era un sistema que estuviera funcionando todo el tiempo y que corrigiera las inconsistencias que cayeran en la Base de Datos en el mismo instante que aparecieran y así no existieran más errores en ningún momento.

Lo solicitado en un inicio por el cliente no fue considerado como una opción viable ya que se requerirían muchos recursos desde el ancho de banda de internet que haya, como las características del servidor para pudiera aguantar un tráfico muy grande de datos y que estuviera trabajando todo el tiempo.

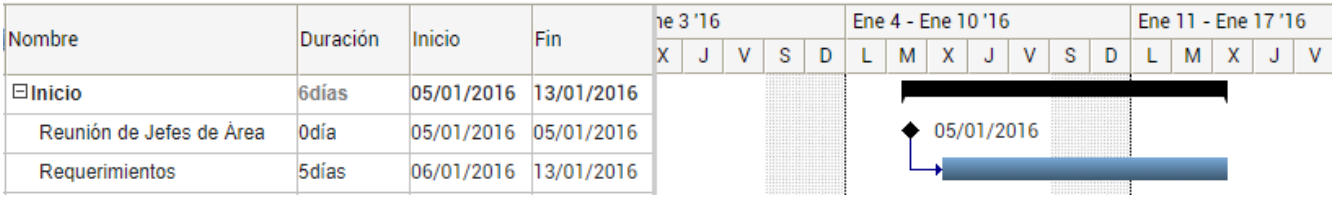


Figura 3.1 Tiempos Requerimientos

Sabiendo esto se tuvo que dedicar muchas horas de junta y revisiones de los requerimientos entre las Áreas para que el proyecto se pudiera ajustar a una opción parecida a lo que solicitaron, pero sin la necesidad de poner en riesgo el rendimiento de los servidores o de otros sistemas (Figura 3.2). De igual forma, con las juntas que se tuvieron se descubrió que las Áreas de Contabilidad y Cartera no tenían una forma de trabajar adecuada ya que contaban con muchas versiones de los reportes que requerían para sus cierres de mes y por esta misma razón cada empleado usaba el reporte que se le acomodara sin saber si la información estaba completa o si era la correcta. Sabiendo esto, se reforzó la idea de que el proyecto que estaban pidiendo era aún menos posible porque se desconocía qué tantos problemas eran del lado de Sistemas y cuáles otros eran de las otras Áreas.

Después de todas las horas de junta invertidos en el levantamiento de requerimientos, se llegó a la siguiente solución la cual no fue igual a lo solicitado en un inicio, pero fue aceptada por lo que se recomendó:

- Aplicación para uso único de las áreas de Cartera y Contabilidad, no puede acceder ningún usuario que no pertenezca a éstas con excepción de las personas del Área de Sistemas encargadas en solucionar problemas referentes de su área.
- La aplicación deberá de contener todos los reportes que se utilizan para los cierres de mes, todos estos siendo única versión para que todos los empleados que lo utilicen trabajen con lo mismo.
- Los reportes de cierre de mes deberán de guardarse en históricos para que puedan ser revisados en cualquier momento sin que la información se altere.
- Deberá de haber una sección en la aplicación donde se podrá revisar las inconsistencias que vayan surgiendo día a día, indicando si la inconsistencia corresponde al mes o si es de la historia, así como indicar el área que le corresponde revisarlo y corregirlo.
- Se podrá exportar en Excel el reporte de inconsistencias.
- Sólo aquellos catalogados como SUPER USUARIO podrán visualizar un botón para poder resolver todas las inconsistencias de forma automática, así como recibir por correo el Script generado para resolver éstas.
- Para cada cierre de mes, ya no debe de existir ninguna inconsistencia, a menos de que sean casos que requieran una mayor revisión o una decisión de alto mando.

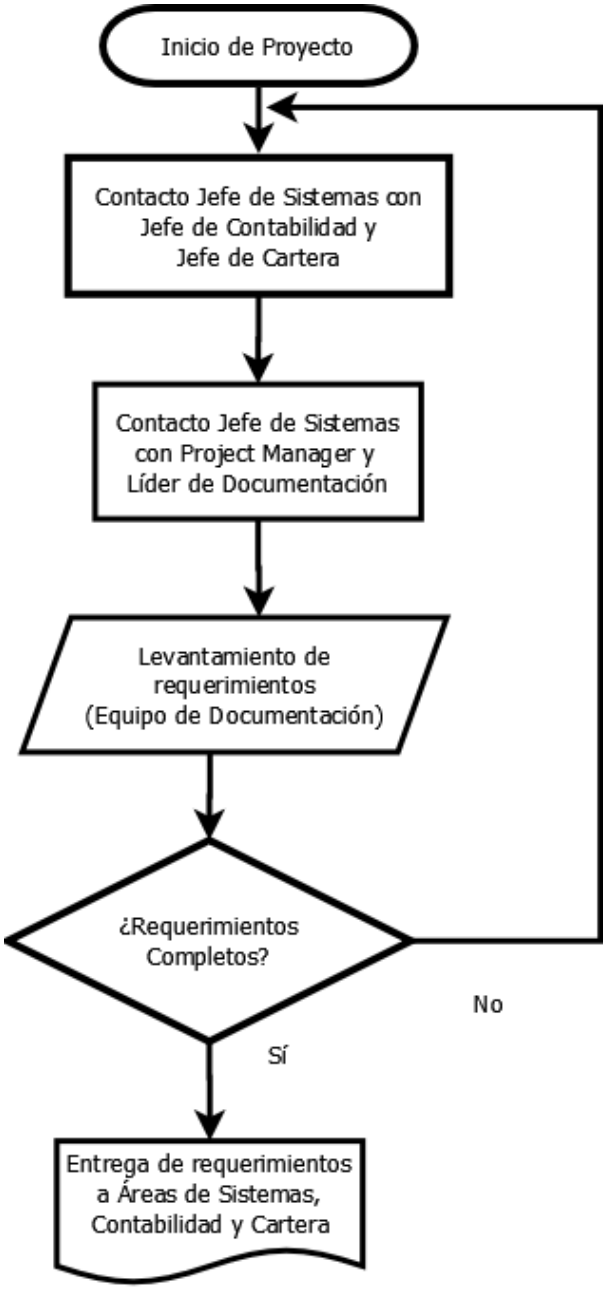


Figura 3.2 Levantamiento de Requerimientos

Planeación y Diseño

Una vez teniendo los requerimientos, se procedió a la planificación del proyecto (Figura 3.3), esto ya nada más trabajando entre el Área de Sistemas, el cual se invirtió una gran cantidad de tiempo puesto que no se sabía de inicio cómo se podría realizar el proyecto sin afectar tanto la Base de Datos principal de la empresa. Fue hasta que se observó que las Áreas de Contabilidad y Cartera no trabajan nada referente al cierre del mes sino hasta el día primero del mes siguiente

(excepto el cierre de mes de diciembre el cual se trabaja el día 2 de enero) que se pudo pensar en una idea que no afectara a la Base de Datos principal de la empresa.

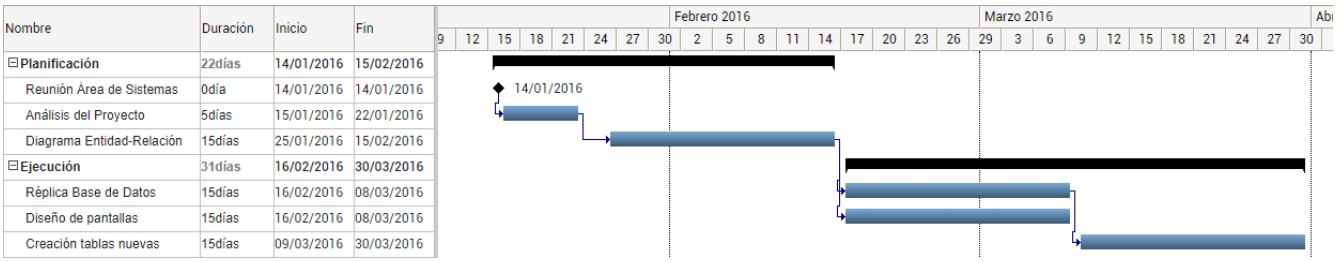


Figura 3.3 Tiempos de la Planificación y el Diseño

A pesar de que los tiempos utilizados para esta parte del proyecto fueron muy extensos, se logró disminuir una parte al dividir el trabajo a los distintos Equipos del Área de Sistemas, ya que el Equipo de Base de Datos y el Equipo de Front-End no requieren de mucha comunicación entre ellos (Figura 3.4), sólo bastó de una buena planificación y de estar a la espera de que el Equipo de Base de Datos creara el Diagrama Entidad-Relación para el proyecto para que estos Equipos se pusieran a trabajar en sus actividades.

Para el Diagrama Entidad-Relación se utilizó el Diagrama existente de las Bases de Datos principal y secundaria para reportes para agilizar el trabajo, puesto que la Base de Datos estaría formada con base en estas otras, sólo se debió revisar con cuidado cuáles eran las tablas que eran necesarias para que, una vez teniendo esto, se pudieran anexar las nuevas tablas que se agregarían para complementar el proyecto.

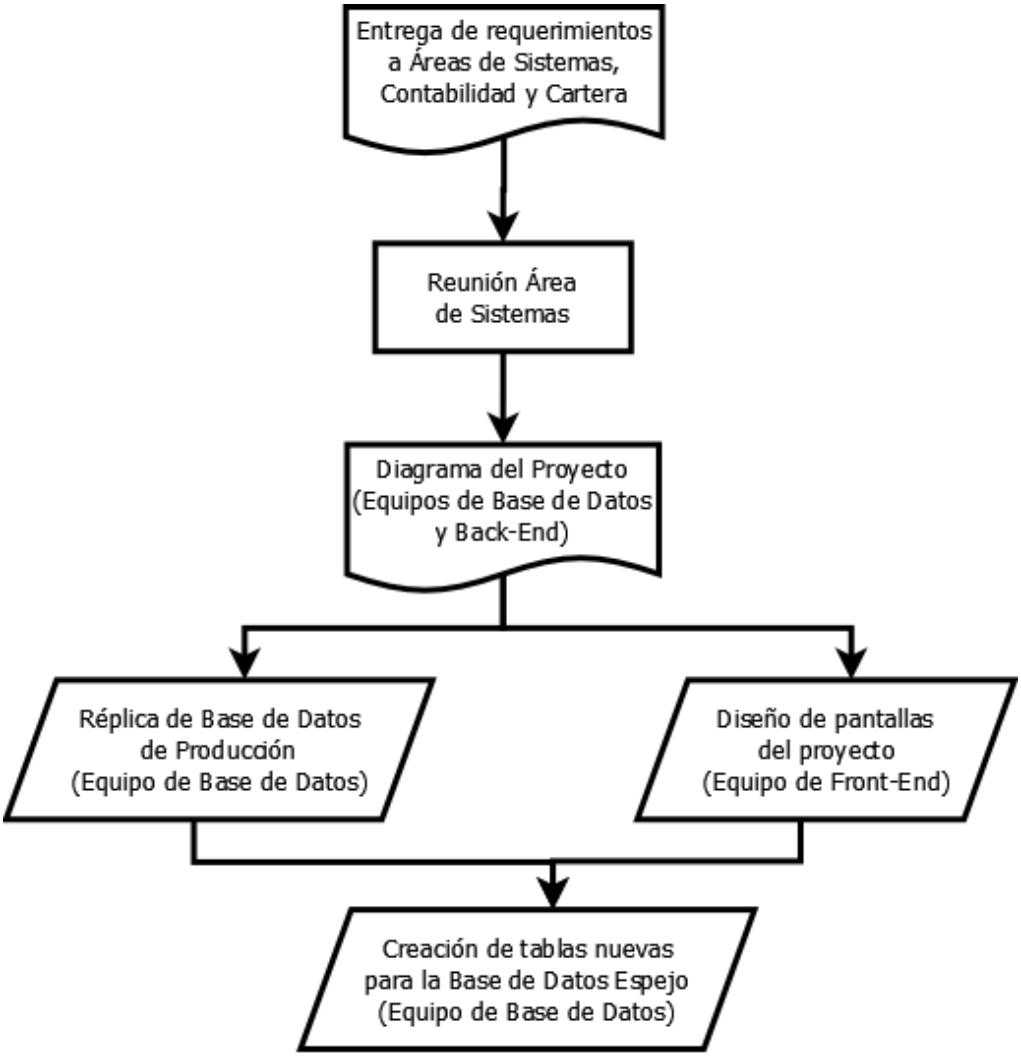


Figura 3.4 Diseño del Proyecto

Sabiendo la fecha en que comienzan a trabajar con los cierres de mes y complementándolo con una regla de negocio de la empresa la cual dice que **no pueden existir movimientos de un mes que ya haya pasado a menos que sean autorizados por los Jefes de Contabilidad o Cartera**, se decidió instalar y configurar un servidor, dicho servidor fue virtual puesto que los recursos a utilizar no serían otros más que los consumidos por la aplicación a crear, por lo que se alojó en éste una nueva base de datos de SQL Server 2010.

La nueva base de datos debería de contener toda la información tanto de la base de datos principal de la empresa, así como de la base de datos secundaria usada en ese momento para los reportes de todas las áreas (Figura 3.5). De esta forma se procedió a que el Equipo de Base de Datos diseñara un proceso ETL que estaría funcionando diario en dos horarios en el que la empresa tuviera poco tráfico de información en sus sistemas, eligiendo como horarios las 01:00 hrs y las 04:30 hrs de cada día.

La razón por la que se manejaron dos horarios para la ejecución del proceso ETL y que estos horarios no estuvieran tan separados uno del otro, fue porque el funcionamiento de dicho ETL debería de hacer un doble pase de información, uno completo y uno diferencial. A las 01:00 hrs de cada día, se ejecuta de forma automática el pase de información, borrando las tablas de la base de datos que existan en alguna de las bases de datos principal o secundarias, las vuelve a crear e inserta nuevamente toda la información que contengan éstas con un límite de fecha hasta el día anterior de la carga de información; esta parte del proceso tiene una duración aproximada de 3 hrs. Por otra parte, a las 04:30 hrs de cada día, se vuelve a ejecutar el proceso ETL, pero en esta ocasión sólo realiza un pase de información diferencial, esto es, revisa que la información insertada a la 01:00 hrs se encuentre completo y, en caso de que exista información diferente a causa de alguna transacción o movimiento en la base de datos principal durante el horario de reposo del proceso ETL, se completa la información, asegurando que se tiene integro los datos de la nueva base de datos la cual llamamos Base de Datos Espejo.

Además del pase de la información, el proceso ETL debe de crear los índices requeridos en la base de datos espejo para hacer que las consultas que se ejecuten lo hagan de forma rápida. Teniendo esto, se ejecutan los Stored Procedure necesarios que hacen que funcione la aplicación.

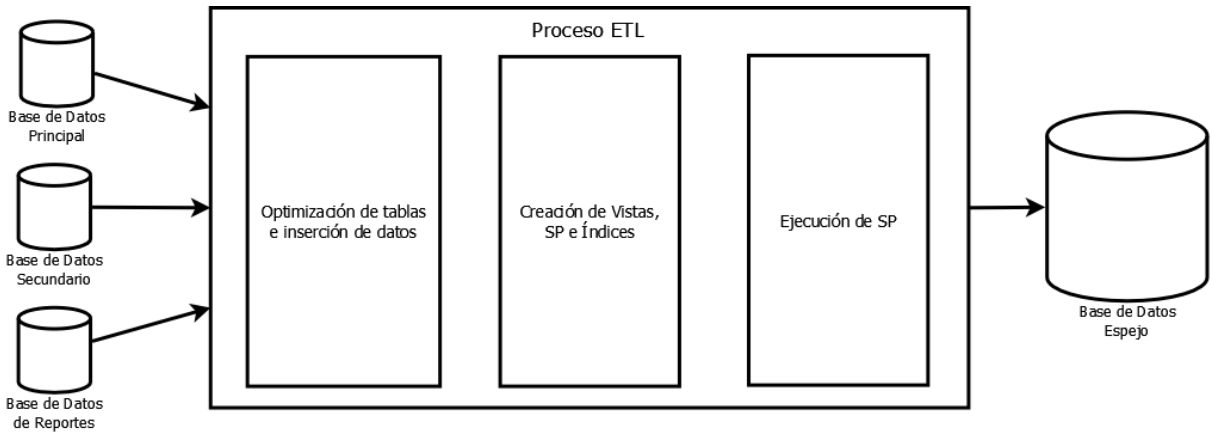


Figura 3.5 Proceso ETL

Por otro lado, participé con el Equipo de Front-End al comenzar a elaborar el diseño de la aplicación (Figura 3.6), así como comenzar a desarrollar el Inicio de Sesión, esto en ASP.NET de Visual Studio 2012. Para el diseño no se utilizó ningún framework que facilitara éste, por reglas de la empresa, se tuvo que programar el diseño desde cero haciendo uso de JavaScript y CSS, aunque no fue tan tardado puesto que requería que se hiciera Responsivo únicamente para equipos de cómputo y no para dispositivos móviles.

El proyecto se realizó con una Master Page para colocar el encabezado de la interfaz, ya que en todas las pantallas tendría que ser igual éste. Dicho encabezado contó con:

- Logo de la empresa
- Nombre de la aplicación
- Botón de Inicio de Sesión
- Botón de Ayuda
- Menú

El botón de Inicio de Sesión dirige a la pantalla de Login y, una vez que el usuario inicie sesión correctamente, este botón cambiará por la información del usuario. En el caso del botón de Ayuda, se muestra información referente a la pantalla en la que se localice para hacer más fácil su uso.

Para el menú, se encuentra separado de acuerdo al área, pero depende del perfil del usuario para poder ver las distintas opciones del menú, los únicos capaces de visualizar el menú completo son aquellos usuarios que cuenten con un perfil SUPER USUARIO.

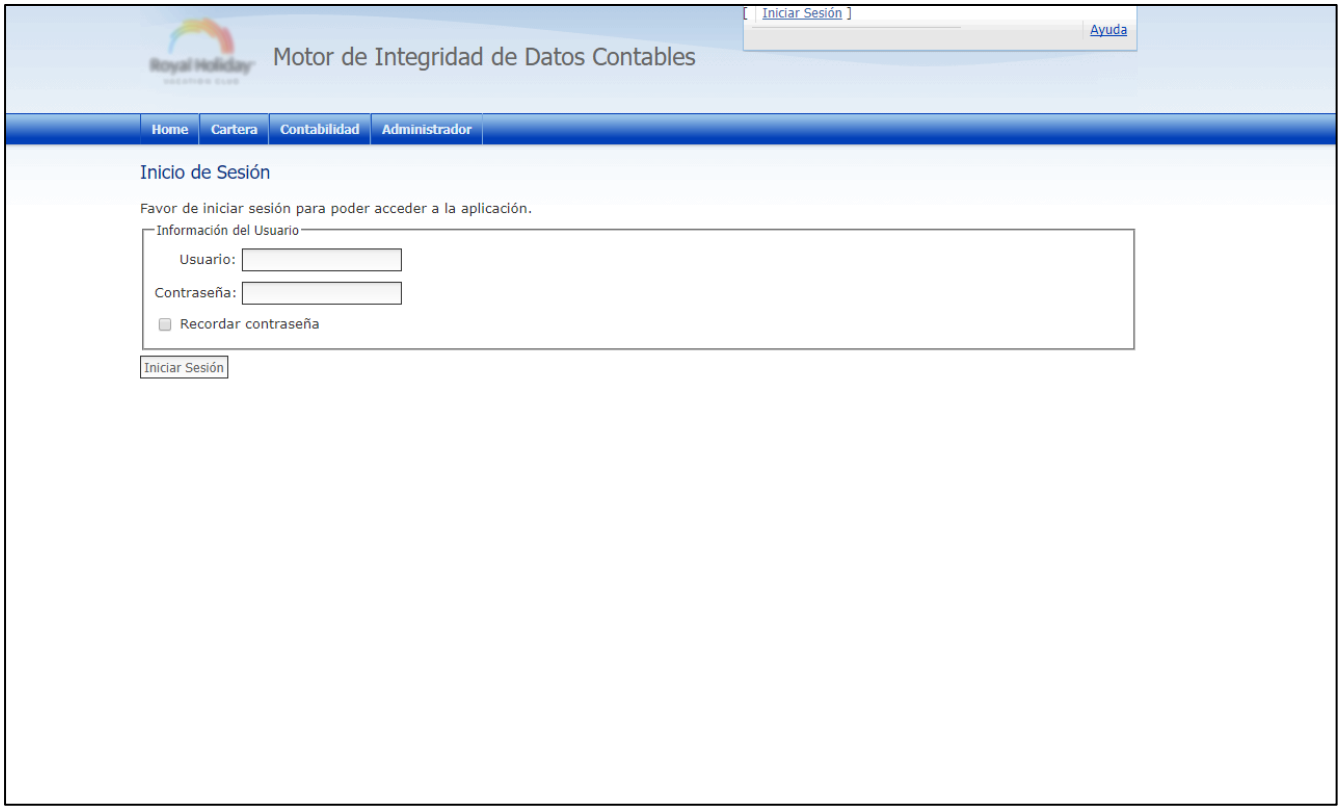


Figura 3.6 Diseño del proyecto

Una vez que el Equipo de Base de Datos terminó de elaborar el proceso ETL y lo puso a funcionar para llenar la Base de Datos Espejo con las tablas y la información provenientes de las otras bases de datos, me tocó continuar creando las tablas de datos nuevas requeridas para el

correcto funcionamiento de la aplicación. Ya que en un inicio se comenzó trabajando con el Inicio de Sesión, las tablas nuevas creadas fueron una para almacenar a los usuarios que podrían hacer uso de la aplicación y uno para almacenar los perfiles.

Teniendo las tablas nuevas, pude comenzar a programar la lógica del Inicio de Sesión por parte del Equipo de Back-End, claro que lo primero fue crear un proyecto vacío en Visual Studio 2012 y, apenas creado el proyecto, realizar la conexión con la Base de Datos Espejo, para esto fue dirigirse al archivo *Web.config* y agregar el nombre de la conexión, dirección IP a dónde conectarse, nombre de usuario y contraseña para acceder a dicha base de datos (Figura 3.7).

```
<connectionStrings>
  <add name="sqlConnStringProduccion" connectionString="Data Source=[redacted];
       User ID= usr_GesCartera; password= [redacted];Initial Catalog=db_CierresCartera;
       MultipleActiveResultSets=true;" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Figura 3.7 Conexión a base de datos

Con esto aseguramos que podemos hacer peticiones a la base de datos y así hacer uso de Stored Procedures según los requiramos. Por lo tanto, seguimos en el proyecto de Visual Studio 2012 creando las carpetas requeridas para trabajar haciendo uso de un patrón MVC, para esto creamos las siguientes carpetas:

- **BusinessEntities:** Contiene los modelos del proyecto.
- **BusinessProcessors:** Almacena los controladores del proyecto.
- **Styles:** Para guardar los archivos css del proyecto.
- **Scripts:** Guarda los archivos js del proyecto.

Para el caso de este proyecto, todas las vistas se dejaron a nivel raíz en lugar de almacenarlos en una carpeta **Views**.

Ya teniendo las carpetas creadas, el primer archivo que agregué en la carpeta de **BusinessEntities** (modelos) fue el llamado *StoredProcedure.cs*, este archivo es el modelo que contiene todos los Procedimientos utilizados para cada una de las vistas del proyecto, dicho modelo se conforma por distintas clases, cada una representando una vista y dentro de estas clases se contienen los contructores de tipo string con los que hacemos el llamado a los distintos Stored Procedure del proyecto. En un inicio sólo se le agregó la clase para la vista de Inicio de Sesión, puesto que fue con lo que se podía trabajar en ese momento, pero más adelante se fueron agregando los Procedimientos para las vistas que se iban creando (Figura 3.8).

```
// Procedimientos para la vista de Log In.
public class LogIn
{
    public const string INICIOSESION_BYPARAMETER = "[dbo].[sp_MC_Sesion]";
}
```

Figura 3.8 Stored Procedure para Inicio de Sesión

El Stored Procedure *[dbo].[sp_MC_Sesion]* sirve para revisar quiénes tienen permiso de los usuarios de la empresa para poder acceder a la aplicación y de igual forma para conocer los permisos que tienen y así sólo mostrarles las opciones que pueden usar para su trabajo. Este Stored Procedure funciona con base en tres tablas (Tabla 1), una de ellas ya existente en las Base de Datos de la empresa y otras dos creadas únicamente en la Base de Datos Espejo, de esta forma revisamos si el usuario puede hacer uso de la aplicación y, en caso de que así sea, registrar su acceso y conocer tanto rol como perfil para mostrarle únicamente las opciones del menú correspondientes.

Tabla 1 Inicio de Sesión

Elemento	Tipo de Elemento	Elementos Usados
[dbo].[sp_MC_Sesion]	Procedimiento	[dbo].[tbl_MC_Usuarios]
		[dbo].[tbl_MC_Perfil]
		[dbo].[t_Personnel]

Para el caso del Inicio de Sesión, además de tener creado el Stored Procedure para acceder a la aplicación, se tuvo que agregar un WebService el cual se utiliza como primer filtro para acceder y que de esta forma no se pueda dar permisos a nadie que no corresponda a la empresa. Este WebService se llama *LoginService* y se agrega a la carpeta **ServiceReference** para poder hacer uso de éste en el código (Figura 3.9).

Figura 3.9 ServiceReference

El WebService recibe dos variables de entrada para que pueda funcionar correctamente, uno para el usuario y otro para la contraseña, devolviendo como resultado una variable tipo string que representa al usuario insertado o, para el caso de que sean incorrectos los datos enviados, devuelve el texto “Invalid Username or Password” (Figura 3.10).

```
public List<LoginBalance> GetByParameter(Constantes.FiltroSesion filtroSesion, ConnectionBD db, DbTransaction dbTrans)
{
    AuthenticateSoapClient authentication = new AuthenticateSoapClient();

    string login = authentication.Login(filtroSesion.Usuario_Sesion, filtroSesion.Psw_Sesion);

    if ((login != "Invalid Username or Password"))
    {
        DbCommand dbCommand = db.GetStoredProcCommand(StoredProcedures.LogIn.INICIOSESION_BYPARAMETER);
        db = SetParameters(db, dbCommand, filtroSesion);
        dbCommand.Transaction = dbTrans;
        using (IDataReader dataReader = db.ExecuteReader(dbCommand))
        {
            List<LoginBalance> perfil = this.GetFromDataReader(dataReader);

            return perfil;
        }
    }
    else
    {
        List<LoginBalance> perfil = null;

        return perfil;
    }
}
```

Figura 3.10 LoginService

Continué trabajando en la lógica para el correcto funcionamiento del Inicio de Sesión, por lo que creé un modelo llamado *LoginBalance.cs*; este archivo, al ser un modelo, está dividido en tres secciones, la primera sección es para hacer llamado a todas las variables que puede hacer uso dicho modelo tomando en cuenta que éstas deben de ser todas de tipo private, la segunda sección son las propiedades (funciones get-set) para poder hacer uso de las variables en otras partes del código, así como agregar el formato o las restricciones que puede contener dicha variable, en nuestro caso en el controlador, y por último la sección de constructores para poder inicializar y crear nuestra instancia especificando el tipo de datos que debe de ser trabajado (ver Anexo 2 para ejemplo del modelo).

Teniendo el modelo completo y ya agregado el WebService, proseguí construyendo el controlador, el cual llevó el nombre de *LoginProcess.cs* y es donde se realiza la conexión a la base de datos, enviando las variables del modelo *LoginBalance.cs* y se le envían al WebService *LoginService* para saber si el usuario y la contraseña ingresados sean correctos y existan dentro de la empresa, en caso de así serlo, se ejecuta el Stored Procedure *[dbo].[sp_MC_Sesion]* para verificar los permisos que tiene el usuario de acuerdo al rol y el perfil y con base en esto poder mostrar en la aplicación las opciones con las que puede trabajar.

El archivo del controlador está formado por varias funciones públicas y privadas, las funciones públicas son aquellas que reciben las variables e inician la conexión a la base de datos y, una vez teniendo esto, se hace el llamado de las funciones privadas con las cuales se ejecutan los Stored Procedure y devuelve las variables de salida (ver Anexo 3 para ejemplo de controlador).

Ya con modelo y controlador terminados, sólo hizo falta hacer uso de éstos, dentro del archivo *Login.aspx.cs* el cual es nuestro archivo que genera la vista de Inicio de Sesión. Este archivo contiene dos funciones, Page_Load y LoginButton_Click, la primer función (Page_Load) se queda vacío puesto que no debe de realizar ninguna operación al acceder a esta vista, por otra parte, para la función LoginButton_Click se agrega toda la lógica que se debe realizar al hacer clic en el botón para Iniciar Sesión, en este caso se envían el usuario y la contraseña por medio del controlador y se espera que devuelva la información del usuario, en caso de que se devuelva un vacío, entonces se despliega un mensaje de que fue incorrecto el inicio de sesión, pero si existe información, entonces se permite acceder a la aplicación mostrando la vista *Home.aspx* y guardando los datos en variables de sesión, de igual forma se puede notar que en la esquina superior derecha aparecen los datos de dicho usuario (Figuras 3.11 y 3.12).

```
// Verifica el usuario para hacer un Login correcto.
protected void LoginButton_Click(object sender, EventArgs e)
{
    string user = LoginUser.UserName;
    string psw = LoginUser.Password;

    Constantes.FiltroSesion objFiltroSesion = new Constantes.FiltroSesion();
    objFiltroSesion.Usuario_Sesion = (user == "") ? null : user;
    objFiltroSesion.Psw_Sesion = (psw == "") ? null : psw;

    try
    {
        LoginProcess objLoginProcess = new LoginProcess();
        List<LoginBalance> listPerfil = objLoginProcess.GetByIdParameter(objFiltroSesion);

        if (listPerfil == null)
        {
            errorMessage.Visible = true;
        }
        else
        {
            errorMessage.Visible = false;
            string perfil = (listPerfil == null) ? "" : (listPerfil.First().NombrePerfil).ToString();
            string fecha = (listPerfil == null) ? "" : (listPerfil.First().FechaModifUsuario).ToString();
            string dept = "";

            if (listPerfil.Count() > 0)
                dept = (listPerfil.First().Department == "") ? "Sistemas" : (listPerfil.First().Department).ToString();

            Session["usuario"] = user.ToUpper();
            Session["perfil"] = perfil.ToUpper();
            Session["fecha"] = fecha;
            Session["department"] = dept.ToUpper();
            Session["regla"] = "";

            Response.Redirect("~/Default.aspx");
        }
    }
    catch (FormatException ex)
    {
        string b = ex.ToString();
    }
    catch (OutOfMemoryException ex)
    {
        string hola = ex.ToString();
    }
}
```

Figura 3.11 LoginButton_Click

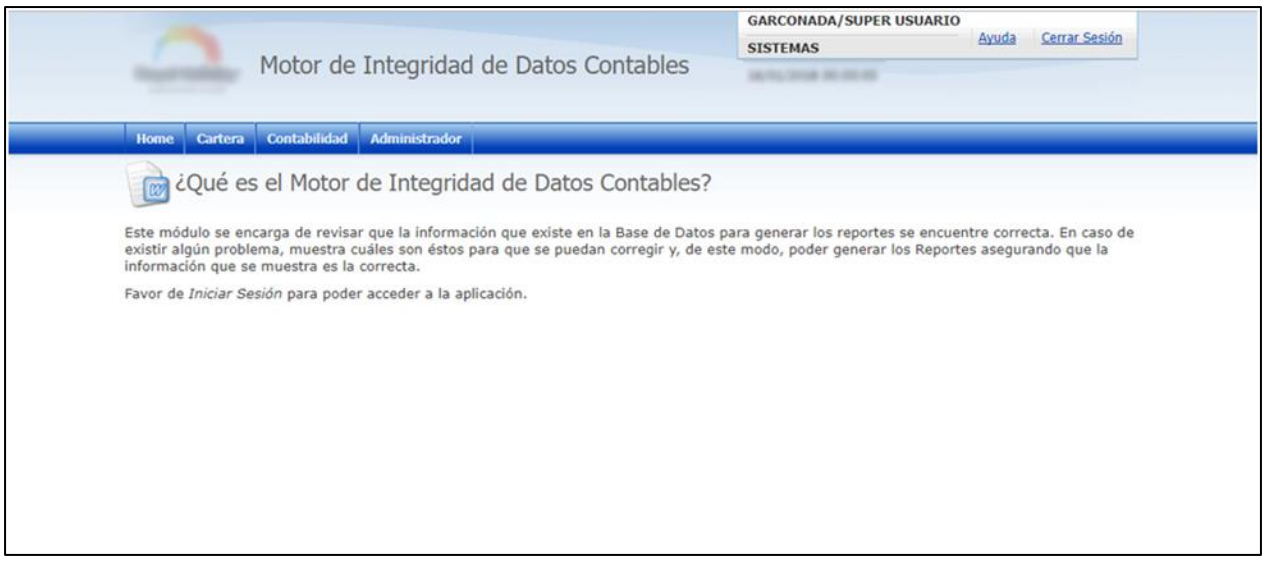


Figura 3.12 Vista Home

Para mi caso, ya que era quien estaba realizando el proyecto y para poder revisar y hacer pruebas correctamente, contaba con un perfil SUPER USUARIO, por lo que podía ver todo el menú y hacer uso de toda la aplicación sin restricción alguna.

Módulo Note Balance

Una vez que terminamos con el inicio del diseño de las vistas, entre ellas la vista de *Inicio de Sesión* y *Home*, así como el Equipo de Base de Datos terminó de hacer el pase de información de las Bases de Datos principal y secundaria a la Base de Datos Espejo por medio del proceso ETL, se realizó toda la parte de Note Balance (Figura 3.13), para eso se requirió un gran tiempo para las reuniones entre todas las Áreas ya que en realidad no se contaba con una versión correcta de reporte para trabajar, por lo que se creó uno desde cero y, una vez teniendo esto, el resto de la implementación en la aplicación así como los manuales ya no requeriría de tanto tiempo en cada actividad.

Nombre	Duración	Inicio	Fin
Ejecución	29días	31/03/2016	11/05/2016
Desarrollo pantallas Note Balance	5días	31/03/2016	07/04/2016
Implementación de Note Balance	8días	29/04/2016	11/05/2016
Seguimiento y Control	36días	31/03/2016	20/05/2016
Reunión para revisar información NB	20días	31/03/2016	28/04/2016
Revisión del módulo NB	6días	12/05/2016	20/05/2016
Cierre	6días	23/05/2016	31/05/2016
Manuales del módulo NB	6días	23/05/2016	31/05/2016

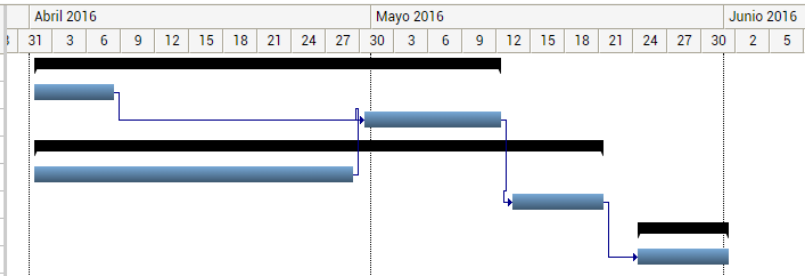


Figura 3.13 Tiempos para Note Balance

Estuve junto con el Equipo de Base de Datos para las reuniones con los Equipos de Contabilidad Note Balance y Cartera Note Balance, desde ese momento se me permitió estar con este Equipo ya que la creación del proceso ETL fue algo en lo que se encargaron aquellos integrantes con mayor experiencia como el Administrador de Base de Datos, y se comenzó a ver la información que requerían para su reporte de Note Balance, esto se realizó viendo cómo trabajaba cada uno de estos Equipos a la hora de hacer un cierre de mes (Figura 3.14).

La parte sencilla que encontramos al ver la forma de trabajo de los Equipos de Contabilidad Note Balance y Cartera Note Balance fue que coincidían los filtros que utilizaban, por lo que creé el Stored Procedure *[Report].[sp_NotesBalance_PrmList]*, igual cuidando no crear demasiados Stored Procedure, todos los catálogos de los filtros estuvieron contenidos en uno mismo y, dependiendo cuál se iba a utilizar, se le pasaba una variable para saber. Dicho Stored Procedure hace uso de varias tablas las cuales son donde se almacenan dichos catálogos (Tabla 2).

Tabla 2 Catálogos de filtros Note Balance

Elemento	Tipo de Elemento	Elementos Usados
[Report].[sp_NotesBalance_PrmList]	Procedimiento	[dbo].[t_Site]
		[dbo].[CLookup]
		[dbo].[Club]
		[dbo].[Currency]
		[dbo].[tbl_MC_GetParametroFiltroNB]

Los filtros que debe de tener el reporte de Note Balance para que puedan hacer uso correcto de este y sin tener filtros de sobra son:

- **Year:** Del periodo que se quiere, así se puede diferenciar si se busca generar algún reporte histórico.
- **Month:** Esto debido a que no se puede generar un reporte que abarque más de un mes.
- **Origen-Site:** Representa la sala de ventas.
- **Lender Pago:** Indica la cartera al que está dirigido el socio.
- **CompaniaContable:** Indica la parte de la empresa a la cual va dirigido los pagos del socio.
- **ClubList:** Representan el tipo de políticas a las que están basados los socios.
- **Currency:** Muestra el tipo de moneda con el que se están realizando los pagos del Mortgage.
- **MortgageStatus:** Muestra si el Mortgage se encuentra activo o en alguna etapa de cancelamiento.

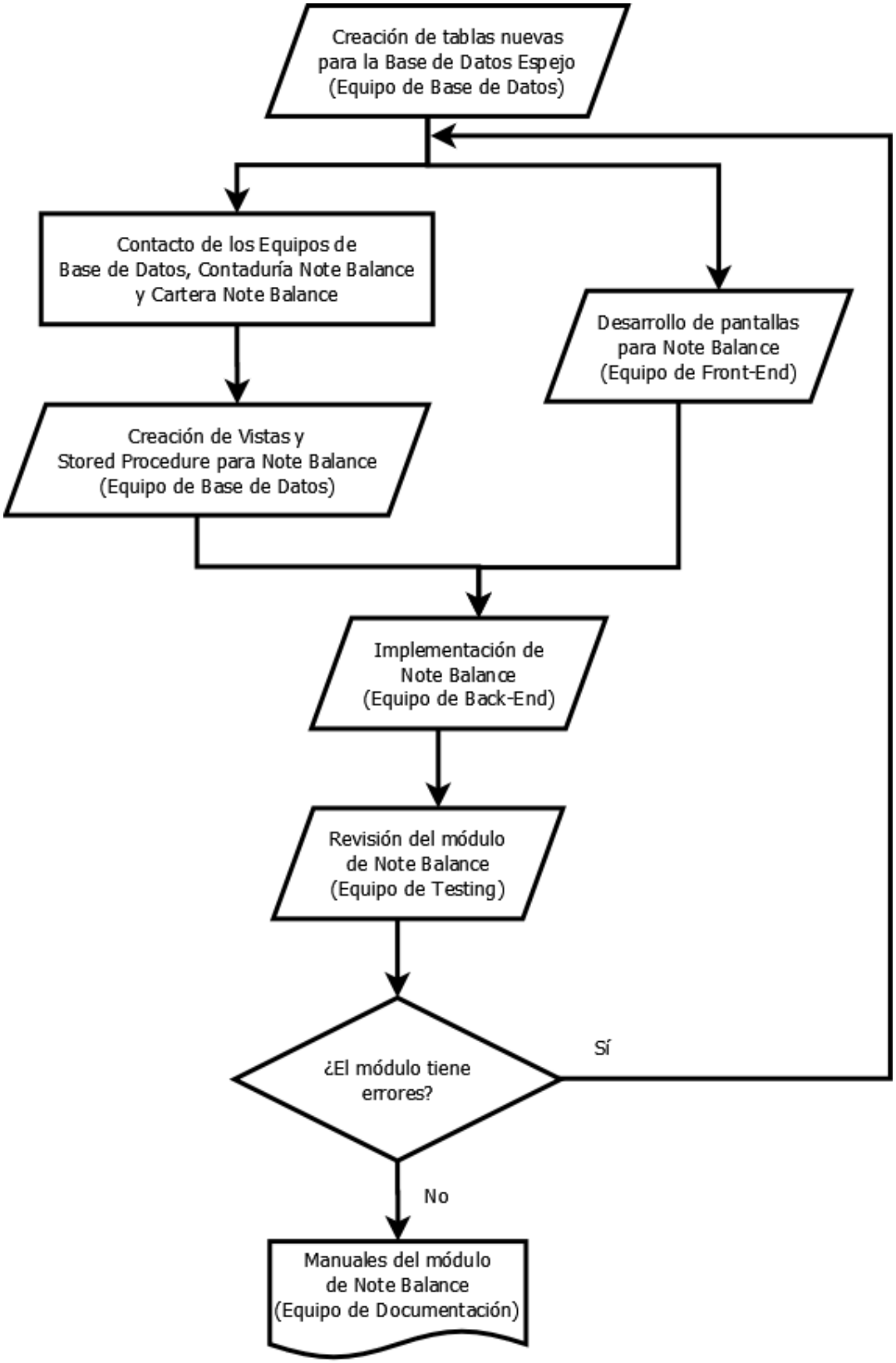


Figura 3.14 Planeación Note Balance

De igual forma, por mi parte les agregué una opción para que pudieran estar guardando los filtros que cada empleado estuviera usando, de esta forma agilizarían su generación de reportes entre un mes y otro, claro que estos filtros no los ve nadie más que la persona que lo creo, no pueden ser compartidos.

Continuando con las reuniones para seguir viendo el contenido del reporte de Note Balance, se logró concluir los campos que requerían para trabajar y, como un apoyo extra de mi parte, al fijarnos que trabajaban los reportes en distintas monedas y al final todo lo pasaban a dólares, se les agregó la opción de que ingresaran el tipo de cambio al que estaban trabajando para que el reporte en automático fuera descargado tanto en la moneda original como en dólares (Figura 3.15). Como resultado, las columnas por las que se conforma el reporte son:

- Mortgage Number
- Year Sold
- Transfer Date
- Lender
- Total Payments
- Next Payment Due Date
- Exchange Rate
- Begin Balance
- Principle Paid
- Principal Paid S IVA
- Principal Paid IVA
- Principle Adjustment
- Principle Adjustment S IVA
- Principle Adjustment IVA
- Principal Transfered
- End Balance
- End Balance S IVA
- End Balance IVA
- IVA
- To Interest
- To Memo
- From Memo
- Late Fee Total Balance
- Late Fee Payment
- Late Fee Adjustment
- Total Paid
- Memo Trx
- To Memo Bal

- From Memo Bal
- Memo Balance

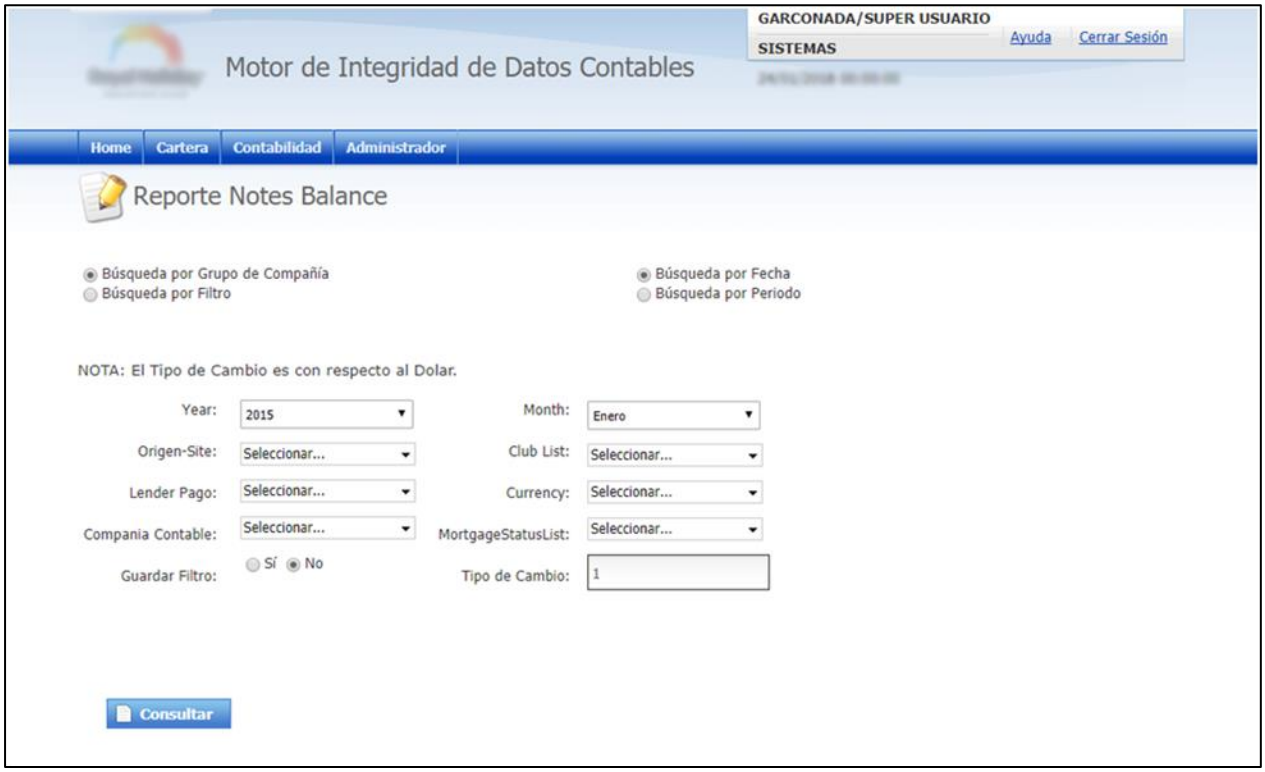


Figura 3.15 Pantalla Note Balance

De esta forma se pudo trabajar en elaborar el Stored Procedure para el reporte de Note Balance, el cual estuvo conformado por un Stored Procedure para crear las tablas donde se almacenaría la información de cada mes, así como el Stored Procedure que llenaría dicha información, este último lo agregué al proceso ETL para que se ejecutara de forma automática diariamente, cuidando las fechas de restricciones para que la información no sea errónea. Primero realicé el Stored Procedure *[Report].[sp_NotesBalance_Prm_IVM]*, el cual lo agregué hasta el final del ETL para que así se ejecutara una vez después de que estuviera completa la Base de Datos Espejo (Ver Anexo 4 para revisar Stored Procedure), cabe mencionar que el Stored Procedure, por lo mismo de que contenía las reglas de negocio para generar todas las columnas, estuvo conformado por un gran número de vistas y tablas (Tabla 3), de igual forma, a pesar de todo los cálculos a realizar, esa fue la razón de que se agregara al ETL y no que se ejecutara directo desde la aplicación, reduciendo el tiempo de ejecución desde la aplicación considerablemente.

Tabla 3 Note Balance de proceso ETL

Elemento	Tipo de Elemento	Elementos Usados
[Report].[sp_NotesBalance_Prm_IVM]	Procedimiento	[dbo].[cat_office_iva_CierresCartera]
		[dbo].[Clookup]
		[dbo].[Club]
		[dbo].[ClubPkgType]
		[dbo].[CreditScore]
		[dbo].[Currency]
		[dbo].[sp_NotesBalance_CierreMes]
		[dbo].[t_Contract]
		[dbo].[t_FinTransCode]
		[dbo].[t_Mortgage]
		[dbo].[t_MortgageMemo]
		[dbo].[t_MortgagePayment]
		[dbo].[t_Owner]
		[dbo].[t_Site]
		[dbo].[t_SoldInventory]
		[Report].[t_MortgageNBReport] (Respectivo año y mes)
		[Report].[vw_MortgagePaymentTransactions_2]
[dbo].[sp_NotesBalance_CierreMes]	Procedimiento	[Report].[t_MortgageNBReport] (Respectivo año y mes)
[Report].[vw_MortgagePaymentTransactions_2]	Vista	[centralcash].[cat_bank]
		[dbo].[clookup]
		[dbo].[club]
		[dbo].[clubpkgtype]
		[dbo].[creditscore]
		[dbo].[currency]
		[dbo].[t_contract]
		[dbo].[t_fintranscode]
		[dbo].[t_mortgage]
		[dbo].[t_mortgagepayment]
		[dbo].[t_owner]
		[dbo].[t_site]
		[dbo].[t_soldinventory]
		[vpos].[banktransaction]
		[vpos].[master_banktransaction]

Continué elaborando el Stored Procedure *[Report].[sp_NotesBalance_Prm_Contrato]*, el cual es sólo una consulta la tabla correspondiente de acuerdo al reporte que necesiten junto con los filtros ingresados, y lo agregué al modelo del proyecto de Visual Studio **StoredProcedure.cs**. Con este Stored Procedure, así como el usado para los catálogos, esta parte del modelo queda completo (Figura 3.16).

```
// Procedimientos para la vista de NotesBalance.
public class ReporteContabilidad
{
    public const string SELECTNOTEBALANCE_BYPARAMETERLIST = "[Report].[sp_NotesBalance_PrmList]";
    public const string SELECTNOTEBALANCE_BYPARAMETER = "[Report].[sp_NotesBalance_Prm_Contrato]";
}
```

Figura 3.16 Stored Procedure para Note Balance

Teniendo los Stored Procedure, me pude poner a trabajar con la lógica para Note Balance, primero creando un modelo llamado **Constantes.cs** donde se haría el llamado de las variables requeridas para los filtros del reporte; a pesar de que la estructura es la misma que la del modelo utilizado para el **LoginBalance.cs**, se le agregaron unas variables para identificar el catálogo a llamar para cada filtro (Figura 3.17).

```
public const string SITELIST_CONTABILIDAD = "1";
public const string LENDERPAGO_CONTABILIDAD = "2";
public const string COMPANIACONTABLE_CONTABILIDAD = "3";
public const string CLUBLIST_CONTABILIDAD = "4";
public const string CURRENCY_CONTABILIDAD = "5";
public const string MORTGAGESTATUSLIST_CONTABILIDAD = "6";
public const string PERIODOFECHA_CONTABILIDAD = "7";
public const string FILTROUSUARIO_CONTABILIDAD = "8";
```

Figura 3.17 Identificador de catálogo Note Balance

De igual forma construí el Controlador para los filtros **FiltroProcess.cs** donde hago el llamado del Stored Procedure *[Report].[sp_NotesBalance_PrmList]* y en el archivo **NoteBalance.aspx.cs** se hace el llamado al controlador enviando el valor del filtro de acuerdo al DropBoxList que se quiere llenar (Figura 3.18), esto en la función *Page_Load* puesto que se requiere que se ejecuten estas acciones al momento de cargar la vista o de lo contrario no se mostraría información.

```
FiltroBalanceProcess filtro = new FiltroBalanceProcess();

List<FiltroBalanceContabilidad> listLlenadoFiltro = filtro.GetByIdParameter(Constantes.SITELIST_CONTABILIDAD, Convert.ToString(Session["usuario"]));
siteList.DataSource = listLlenadoFiltro;
siteList.DataValueField = "varIdentity_Contabilidad";
siteList.DataTextField = "varDescription_Contabilidad";
siteList.DataBind();
```

Figura 3.18 Llenado de DropDownList Note Balance

Con esto la única función a la que le hacía falta ponerle lógica era la del botón *btnBuscar_Click* para que se enviaran los filtros seleccionados por el usuario y se descargara el reporte resultante de Note Balance. Para este caso, la lógica se divide en varias tareas a realizar, el primero es guardar en variables la información de los filtros (Figura 3.19), el año y mes se guarda directo ya que sólo se puede seleccionar un año y un mes, pero para los demás filtros, como se pueden elegir varias opciones, se tiene que transformar la lista de filtros en una sola variable donde se van concatenando los datos y separando por una coma, esto para facilidad al llamar el Stored Procedure.

```
string ye = year.Text;
string mo = month.SelectedValue;

// Se concatenan los valores seleccionados de SiteList
string sl = "";

foreach (System.Web.UI.WebControls.ListItem item in siteList.Items)
{
    if (item.Selected)
    {
        if (item.Value != "0")
        {
            if (sl == "")
                sl = item.Value;
            else
                sl = sl + "," + item.Value;
        }
        else
        {
            sl = "0";
        }
    }
}
```

Figura 3.19 Almacenamiento de filtros Note Balance

Teniendo los filtros guardados en sus respectivas variables, se revisa que el mes y el año sean correctos, puesto que no existiría información de un mes o año superior al actual y, en caso de no ser correctos, se muestra un mensaje indicando cuál de los dos filtros es el erróneo. Si las las fechas son correctas, entonces procede a enviar la información al Stored Procedure, para eso se pasan todas las variables a un objeto que se construyó cuando creamos el archivo **Constantes.cs** (Figura 3.20).

```
Constantes.FiltroReporteContabilidadDTO objFiltroContabilidadDTO = new Constantes.FiltroReporteContabilidadDTO();

objFiltroContabilidadDTO.Month_Contabilidad = (mo == "0" || mo == "") ? null : mo;
objFiltroContabilidadDTO.Year_Contabilidad = (ye == "0" || ye == "") ? null : ye;
objFiltroContabilidadDTO.SiteList_Contabilidad = (sl == "") ? null : sl;
objFiltroContabilidadDTO.LenderPago_Contabilidad = (lp == "") ? null : lp;
objFiltroContabilidadDTO.ConpaniaContable_Contabilidad = (cc == "") ? null : cc;
objFiltroContabilidadDTO.ClubList_Contabilidad = (cl == "") ? null : cl;
objFiltroContabilidadDTO.Currency_Contabilidad = (cu == "") ? null : cu;
objFiltroContabilidadDTO.MortgageStatusList_Contabilidad = (ms == "") ? null : ms;
objFiltroContabilidadDTO.TipoCambio_Contabilidad = (tipoCambio.Text == "" || tipoCambio.Text == "0") ? "1" : tipoCambio.Text;
objFiltroContabilidadDTO.MortgageNumber_Contabilidad = (mn == "") ? null : ("'" + mn + "'");
objFiltroContabilidadDTO.BeginBalance_Contabilidad = bb;
objFiltroContabilidadDTO.PeriodoFecha_Contabilidad = (p == "") ? null : p;
objFiltroContabilidadDTO.FiltroUsuario_Contabilidad = activoFiltro;
objFiltroContabilidadDTO.IdFiltroUsuario_Contabilidad = (activoFiltro == true) ? filtro : null;
objFiltroContabilidadDTO.NombreFiltro_Contabilidad = (nombreFil == "") ? null : nombreFil;
objFiltroContabilidadDTO.User_Contabilidad = (us == "") ? null : us;
```

Figura 3.20 Objeto de filtros para Note Balance

Para poder hacer uso del objeto de los filtros y enviarlo al Stored Procedure correspondiente, tuve que crear el modelo y controlador correspondientes, los cuales llevaron los nombres de **ContabilidadBalance.cs** y **ContabilidadProcess.cs**, respectivamente. El primero es el modelo, el cual almacena los constructores con las variables de los filtros; estos se envían a **ContabilidadProcess.cs**, que es el controlador para generar el reporte, haciendo uso del Stored Procedure *[Report].[sp_NotesBalance_Prm_Contrato]*, el cual ya se había agregado en el modelo **StoredProcedures.cs**. De esta forma, en caso de que la información enviada sea correcta, devuelve la información de la Base de Datos Espejo y le da formato de Excel, descargándolo en la computadora del usuario (Figuras 3.21 y 3.22).

Una vez teniendo todo esto terminado, se procedió a realizar todas las pruebas correspondientes para verificar que el Módulo de Note Balance se encontraba correcto, revisando que la aplicación funcionara correcto y, junto con los Equipos de Contabilidad Note Balance y Cartera Note Balance, se hicieron pruebas para verificar que la información que aparecía en el reporte fuera el correcto, en cuanto a columnas o cálculo de los montos, ya que si una de estas cosas estaba mal, se corregía de inmediato y las pruebas comenzaban desde nuevamente.

Lo último por hacer en cuanto al Módulo de Note Balance fue documentar todo, desde manuales de usuario, como manuales técnicos separados en la parte de Back-End y Base de Datos, este último junto con Diagrama entidad-Relación y Diagrama Secuencial para saber cómo estaba formada la Base de Datos Espejo y de qué forma estaban ligadas las tablas y vistas, así estar conscientes cuánto repercute una modificación o cambio en dado caso que se requiera.

```
try
{
    ContabilidadProcess objContabilidadProcess = new ContabilidadProcess();
    System.Data.DataTable listCOntabilidad = objContabilidadProcess.GetByIdParameter(objFiltroContabilidadDTO);

    string nombreArchivo = (checkFiltro.Checked == true) ? (filtroUser.SelectedItem).ToString() : nombreFiltroTxt.Text;

    if (checkFiltro.Checked == true)
        nombreArchivo = (filtroUser.SelectedItem).ToString();
    else
        nombreArchivo = ((nombreFiltroTxt.Text == null) || (nombreFiltroTxt.Text == "")) ? "NotesBalance" : nombreFiltroTxt.Text;


    nombreArchivo = nombreArchivo + "_" + (DateTime.Today).ToString("yyyyMMdd").ToUpper() + ".xls";

    string attachment = "attachment; filename=" + nombreArchivo;
    Response.ClearContent();
    Response.AddHeader("content-disposition", attachment);
    Response.ContentType = "application/vnd.ms-excel";
    string tab = "";
    foreach (DataColumn dc in listCOntabilidad.Columns)
    {
        Response.Write(tab + dc.ColumnName);
        tab = "\t";
    }
    Response.Write("\n");
    int i;
    foreach (DataRow dr in listCOntabilidad.Rows)
    {
        tab = "";
        for (i = 0; i < listCOntabilidad.Columns.Count; i++)
        {
            if (dr[i].GetType().Name == "Decimal")
            {
                Response.Write(tab + String.Format(CultureInfo.InvariantCulture, "{0:0,0.00}", Convert.ToDouble(dr[i])).Trim());
                tab = "\t";
            }
            else
            {
                Response.Write(tab + dr[i].ToString().Trim());
                tab = "\t";
            }
        }
        Response.Write("\n");
    }
    Response.End();
}
```

Figura 3.21 Generación de Excel Note Balance

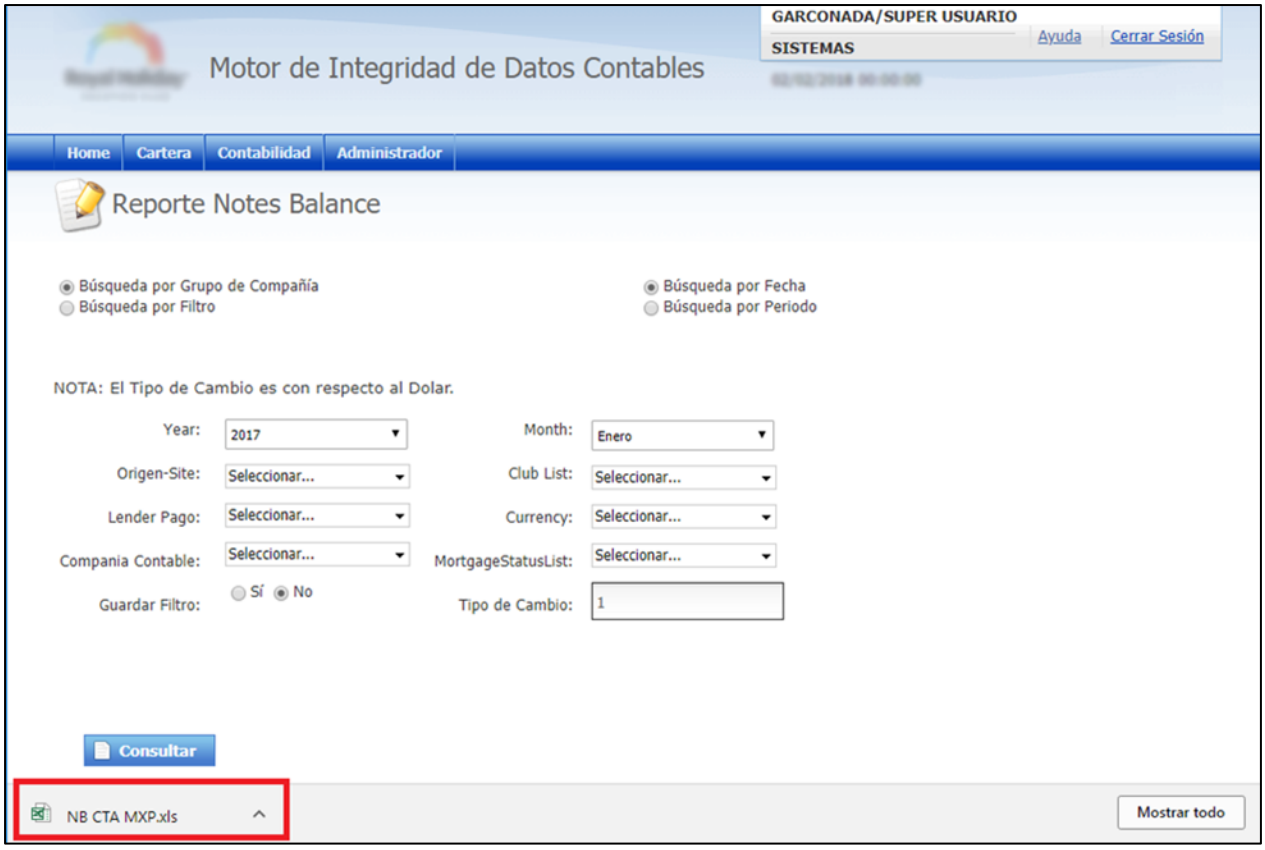


Figura 3.22 Descarga de Reporte Note Balance

Módulo Base de Transacciones

A mí me tocó incorporarme al módulo de Base de Transacciones una vez que se terminó todo el módulo de Note Balance, pero ya para esto llevaban algunas reuniones el Equipo de Base de Datos con los Equipo de Contabilidad Base de Transacciones y Cartera Base de Transacciones. En cuanto al tiempo establecido para este módulo fue muy parecido al del módulo anterior ya que el funcionamiento era el mismo (Figura 3.23), ingresar filtros para descargar un reporte, claro que al existir demasiadas versiones para un Reporte de Base de Transacciones, se mantuvo una constante comunicación y continuas juntas entre equipos para así concretar un reporte definitivo. Para el caso de este módulo, por lo mismo de que la forma de trabajo fue demasiado similar al del módulo de Note Balance, únicamente se tocarán aquellas partes en las cuales existió un cambio (Figura 3.24).

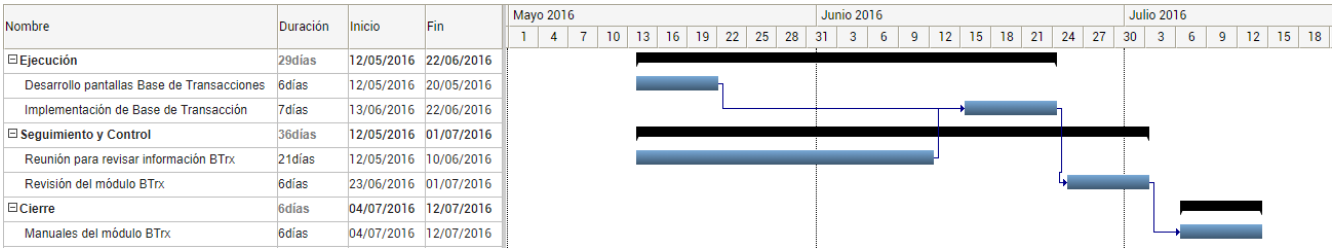


Figura 3.23 Tiempos para Base de Transacciones

En las reuniones con los Equipos de Contabilidad Base de Transacciones y Cartera Base de Transacciones se revisaron catálogos por los cuales podrían filtrar el reporte y las columnas que llevaría éste con la finalidad de tener una versión final, se llegó a un resultado aunque no se pudo optimizar demasiado puesto que cada uno de los integrantes de cada equipo hacía uso de distintas columnas para realizar su trabajo. Así pude crear el Stored Procedure *[Report].[spTrxCartera_PrmList]* para consultar los catálogos utilizados para cada uno de los filtros usados (Tabla 4).

Tabla 4 Catálogos de filtros Base de Transacciones

Elemento	Tipo de Elemento	Elementos Usados
[Report].[sp_TrxCartera_PrmList]	Procedimiento	[dbo].[CLookup]
		[dbo].[Club]
		[dbo].[t_FundingInstitutution]
		[dbo].[t_Owner]
		[dbo].[cCountry]
		[dbo].[t_Site]
		[dbo].[Currency]
		[dbo].[tbl_MC_GetParametroFiltroTrx]
		[dbo].[tbl_MC_Usuarios]

Los filtros para el reporte de Base de Transacciones y su correcto manejo fueron:

- **CompaniaContable:** Indica la parte de la empresa a la cual va dirigido los pagos del socio.
- **ClubList:** Representan el tipo de políticas a las que están basados los socios.
- **CurrentLender:** Indica la cartera al que está dirigido el socio.
- **Cia Cobranza:** Muestra la compañía que le cobra al socio para luego enviarle el dinero a la Compañía Contable correspondiente.
- **Grupo Cobranza:** Grupo de cobro de cual forma parte la Compañía de Cobranza.
- **Lender Pago:** Indica la cartera por la cual hizo el pago el socio.

- **Country:** Es el país al cual pertenece el socio.
- **Site List:** Representa la sala de ventas.
- **Currency:** Muestra el tipo de moneda con el que se están realizando los pagos del Mortgage.
- **Year:** Del periodo que se quiere, así se puede diferenciar si se busca generar algún reporte histórico.
- **Month:** Esto debido a que no se puede generar un reporte que abarque más de un mes.

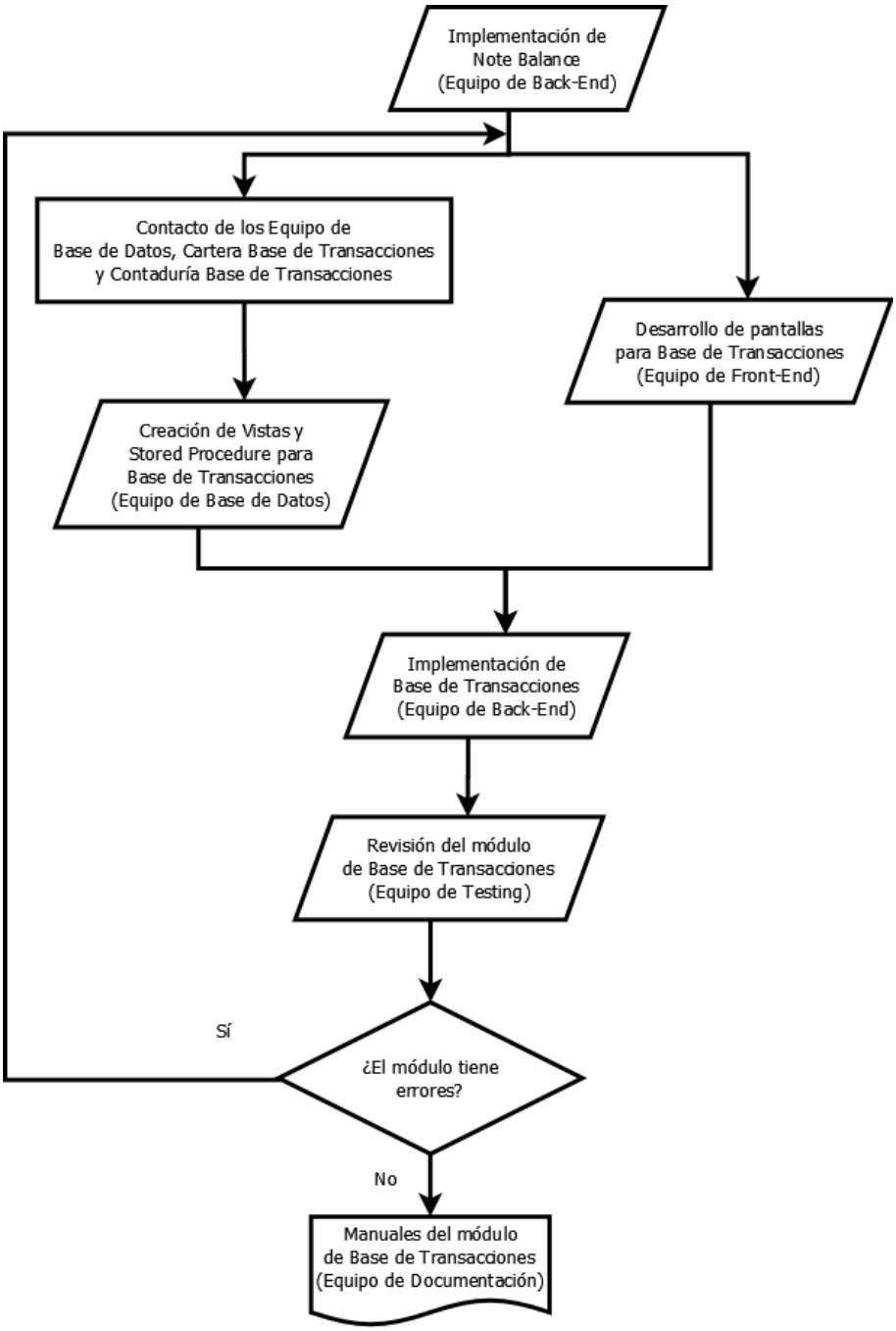


Figura 3.24 Planeación Base de Transacciones

Al igual que el módulo de Note Balance, le agregué la opción para que el usuario pueda guardar los filtros que utiliza para así reutilizarlos cada mes logrando agilizar su trabajo.

Al finalizar las reuniones entre los Equipos de Base de Datos, Contabilidad Base de Transacciones y Cartera Base de Transacciones, se concluyó con las columnas que debería de llevar el reporte de Base de Transacciones, para este caso a pesar de que cada área trabaja con columnas diferentes, no aceptaron tener un reporte adaptado a cada uno ya que al trabajar con un reporte igual, tendrían mayor confianza en que la información que ven todos es lo mismo. Como resultado, las columnas que contiene el reporte de Base de Transacciones son:

- ContractID
- mortgageid
- lastName
- Firstname
- ContractNumber
- Mes
- Anio
- TransactionDate
- postedDate
- SiteName
- SiteId
- MortgageNumber
- SaleDate
- CloseDate
- TipoContrato
- PkgType
- Points
- ContracStatus
- StatusCarteraid
- StatusCartera
- CountryId
- Country
- Nacionalidad
- ContractCurrency
- Compania
- CompaniaDesc
- CodCompaniaContable
- LenderTrxId
- CurrentLenderid
- CurrentLender
- MortgageCurrencyId
- MortgageCurrency
- Numpagoshechos
- PaymentAmount
- APR
- BankID
- ForAmount
- BkDesc
- UserName
- CodTransanccion
- TransactionCode
- MextPaymentDueDate
- Iva
- TransactionCodeId
- Amount
- BaseAMountTrx
- AccAmount
- BaseAmountTrx_S_IVA
- ToCapital
- To_Capital_S_IVA
- ToInterest
- To_Interest_IVA
- To_Interest_S_IVA
- Imp_Transaccion_IVA
- ToCapital_IVA
- ToLateFees
- FromMemo
- ToMemo
- CurrentPrincipalBal
- InstrumentPayID

- IntrumentDescription
- SaleYear
- Reference
- AuthCode
- Morosidad_Tipo
- Morosidad_Status
- Morosidad_Tipo_Inicial
- Morosidad_Inicial
- ClubID
- ClubName
- FundingInstitutionID
- CiaCobranza
- FundingStatusID
- Fundingstatus
- TrxCurrencyID
- TrxCurrency
- CiaContableMortgage
- LoanPortfolioid
- LoanPortolio
- PortfolioComment
- AssignedDate
- PaymentRemaining
- OwnerId
- GrupoCobranza
- user10
- CiaContableRCC
- ReferenceMBT
- TrxExchangeRate
- BillingCodeID
- BillingCode
- PaymentType
- MortgagePaymentID
- Origen
- LenderTrx
- Site_Origen
- CSC_AccNum
- CSCTransType

Contando con toda la información necesaria, al igual como se trabajó para el módulo de Note Balance, construí los dos Stored Procedure que se utilizarían para generar el reporte de Base de Transacciones, el primero fue el Stored Procedure *[Report].[sp_TrxCartera_Prm_IVM]*, el cual es el que se aloja en el proceso ETL para pasar la información de las Bases de Datos Principal y Secundarias a la Base de Datos Espejo (Ver Anexo 5 para revisar Stored Procedure) y, por lo mismo de que cuenta con las reglas de negocio para funcionar, el Stored Procedure trabaja con gran cantidad de tablas (Tabla 5).

Tabla 5 Base de Transacciones de proceso ETL

Elemento	Tipo de Elemento	Elementos Usados
[Report].[sp_TrxCartera_Prm_IVM]	Procedimiento	[dbo].[CLookup]
		[Report].[fnOneColumnint]
		[Report].[t_mortgageNBTrax_V2]
		[Report].[vw_TrxCartera]
		[Report].[vw_trxMortgageMemo]
[Report].[vw_TrxCartera]	Vista	[ACartera].[CarteraHistoricaDia1]
		[CentralCash].[cat_bank]
		[Concord].[CSCPostPayments]

		[dbo].[cat_office_iva_CierresCartera]
		[dbo].[cat_office_tsw]
		[dbo].[CLookup]
		[dbo].[Club]
		[dbo].[ClubPkgType]
		[dbo].[ContractSubStatus]
		[dbo].[CreditScore]
		[dbo].[Currency]
		[dbo].[CurrencyExchangeRate]
		[dbo].[ExchangeRate]
		[dbo].[t_Contract]
		[dbo].[t_FinTransCode]
		[dbo].[t_FundingInstitutution]
		[dbo].[t_Mortgage]
		[dbo].[t_MortgagePayment]
		[dbo].[t_MortgageSalesTran]
		[dbo].[t_Owner]
		[dbo].[t_Site]
		[dbo].[t_SoldInventory]
		[VPOS].[BankTransaction]
		[VPOS].[Master_BankTransaction]
		[VPOS].[VPOS_Merchants_LoanNew]
		[VPOS].[Vpos_TblMerchants]
[Report].[vw_trxMortgageMemo]	Vista	[ACartera].[CarteraHistoricaDia1]
		[CentralCash].[cat_bank]
		[Concord].[CSCPostPayments]
		[dbo].[cat_office_iva_CierresCartera]
		[dbo].[CLookup]
		[dbo].[Club]
		[dbo].[ClubPkgType]
		[dbo].[ContractSubStatus]
		[dbo].[CreditScore]
		[dbo].[Currency]
		[dbo].[CurrencyExchangeRate]
		[dbo].[ExchangeRate]
		[dbo].[t_Contract]
		[dbo].[t_FinTransCode]
		[dbo].[t_FundingInstitutution]
		[dbo].[t_Mortgage]
		[dbo].[t_MortgageMemo]
		[dbo].[t_MortgagePayment]
		[dbo].[t_MortgageSalesTran]

		[dbo].[t_Owner]
		[dbo].[t_Site]
		[dbo].[t_SoldInventory]
		[VPOS].[BankTransaction]
		[VPOS].[Master_BankTransaction]
		[VPOS].[VPOS_Merchants_LoanNew]
		[VPOS].[Vpos_TblMerchants]

El segundo Stored Procedure para generar el reporte de Base de Transacciones fue *[Report].[sp_TrxCartera_Prm_FiltroTrx]* y sólo hace una consulta a la tabla donde se almacena la información de acuerdo a los filtros que se agreguen. Este Stored Procedure lo agregué en el código en el modelo **StoredProcedures.cs** junto con el Stored Procedure que ya había realizado para los catálogos de los filtros (Figura 3.25).

```
// Procedimientos para la vista de Base de Transacciones.
public class ReporteCartera
{
    public const string SELECTTRXCARTERA_BYPARAMETERLIST = "[Report].[sp_TrxCartera_PrmList]";
    public const string SELECTTRXCARTERA_BYPARAMETER = "[Report].[sp_TrxCartera_Prm_FiltroTrx]";
}
```

Figura 3.25 Stored Procedure para Base de Transacciones

Con el Stored Procedure terminado, seguí con la lógica para Base de Transacciones; al igual que en Note Balance, agregué en el modelo **Constantes.cs** las variables requeridas para los filtros y, en el archivo **BaseTransacciones.aspx.cs** hice el llenado de los filtros en la función *Page_Load*, así como la lógica de la función *btnBuscar_Click* haciendo el envío de los filtros a la Base de Datos y devolviendo el reporte de Base de Transacciones en formato Excel (Figuras 3.26 y 3.27).

```
Constantes.FiltroReporteCarteraDTO objFiltroCarteraDTO = new Constantes.FiltroReporteCarteraDTO();

objFiltroCarteraDTO.Month_Cartera = (mo == "0" || mo == "") ? null : mo;
objFiltroCarteraDTO.Year_Cartera = (ye == "0" || ye == "") ? null : ye;
objFiltroCarteraDTO.CompaniaContable_Cartera = (ccontable == "0" || ccontable == "") ? null : ccontable;
objFiltroCarteraDTO.LenderActual_Cartera = (clender == "0" || clender == "") ? null : clender;
objFiltroCarteraDTO.FundingStatusList_Cartera = (gc == "0" || gc == "") ? null : gc;
objFiltroCarteraDTO.Country_Cartera = (co == "0" || co == "") ? null : co;
objFiltroCarteraDTO.ClubList_Cartera = (clist == "0" || clist == "") ? null : clist;
objFiltroCarteraDTO.FundingInstitutionList_Cartera = (ccobranza == "0" || ccobranza == "") ? null : ccobranza;
objFiltroCarteraDTO.LenderPago_Cartera = (lp == "0" || lp == "") ? null : lp;
objFiltroCarteraDTO.SiteList_Cartera = (sl == "0" || sl == "") ? null : sl;
objFiltroCarteraDTO.Currency_Cartera = (cu == "0" || cu == "") ? null : cu;
objFiltroCarteraDTO.FiltroUsuario_Cartera = filtro;
objFiltroCarteraDTO.IdFiltroUsuario_Cartera = (filtro == true) ? idFiltro : null;
objFiltroCarteraDTO.NombreFiltro_Cartera = (nombreFil == "") ? null : nombreFil;
objFiltroCarteraDTO.User_Cartera = (us == "") ? null : us;
objFiltroCarteraDTO.Pagina_Cartera = 500000;
```

Figura 3.26 Objeto de filtros para Base de Transacciones

Figura 3.27 Descarga de Reporte Base de Transacciones

Módulo Inconsistencias

El último módulo a trabajar y el más importante fue el de Inconsistencias, por esa razón el tiempo dedicado fue aproximadamente la mitad del tiempo de todo el proyecto (Figura 3.28), además de que algunos casos de inconsistencias pasaban desapercibidos durante el transcurso de los meses y sólo aparecían a final de mes que es cuando había una cantidad muy grande de transacciones en las Bases de Datos.

Nombre	Duración	Inicio	Fin
Ejecución	81días	**23/06/2016**	**14/10/2016**
Generar Reporte NB	0día	23/06/2016	23/06/2016
Listado inconsistencias NB	20días	24/06/2016	22/07/2016
Generar Reporte BTrx	0día	13/07/2016	13/07/2016
Listado Inconsistencias BTrx	21días	14/07/2016	12/08/2016
Clasificación de inconsistencias	5días	15/08/2016	22/08/2016
Sistematización de inconsistencias	14días	23/08/2016	12/09/2016
Diseño de pantallas para Inconsistencias	5días	13/09/2016	20/09/2016
Implementación de Inconsistencias	17días	21/09/2016	14/10/2016
Seguimiento y Control	30días	**17/10/2016**	**28/11/2016**
Revisión del módulo de Inconsistencias	11días	17/10/2016	01/11/2016
Corrección de inconsistencias	10días	14/11/2016	28/11/2016
Cierre	26días	**02/11/2016**	**08/12/2016**
Manuales del módulo de Inconsistencias	7días	02/11/2016	11/11/2016
Documentación de inconsistencias	5días	29/11/2016	06/12/2016
Recarga de información	1día	07/12/2016	08/12/2016

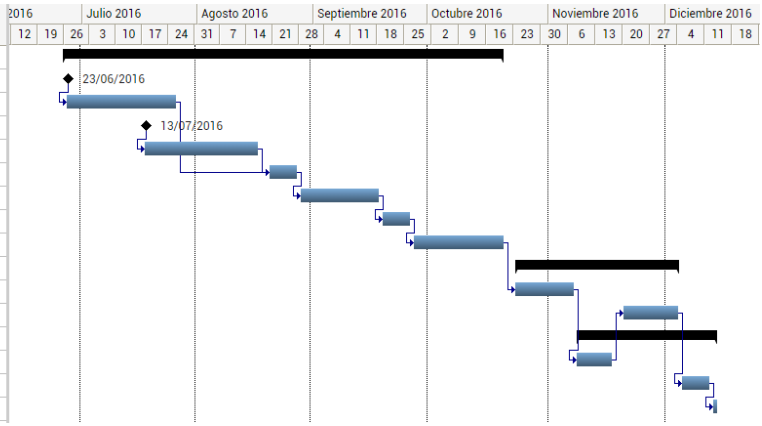


Figura 3.28 Tiempos para Inconsistencias

Para el trabajo de este módulo se tuvo que esperar a que los otros dos módulos estuvieran terminados (Figura 3.29), esto debido a que la forma de trabajar este módulo de acuerdo a lo que se acordó con el Área de Sistemas, sería que se trabajaría a partir del reporte (de atrás para delante) porque no teníamos otra forma de identificar las inconsistencias existentes desde nuestro lado a causa de los constantes cambios que se realizan en los demás proyectos de la empresa, así las Áreas de Contabilidad y Cartera podrían generar los respectivos reportes constantemente y revisarlos a fondo para buscar las inconsistencias y, conforme encontraran éstas, yo me encargaría de programar la regla para detectarlos en automático, así como para arreglarlos mientras que el resto del Área de Sistemas los solucionaba desde la raíz para que no volvieran a aparecer. Por último, cada una de las inconsistencias que se encontraran, se clasificarían para saber el Equipo al que le corresponde solucionarlo.

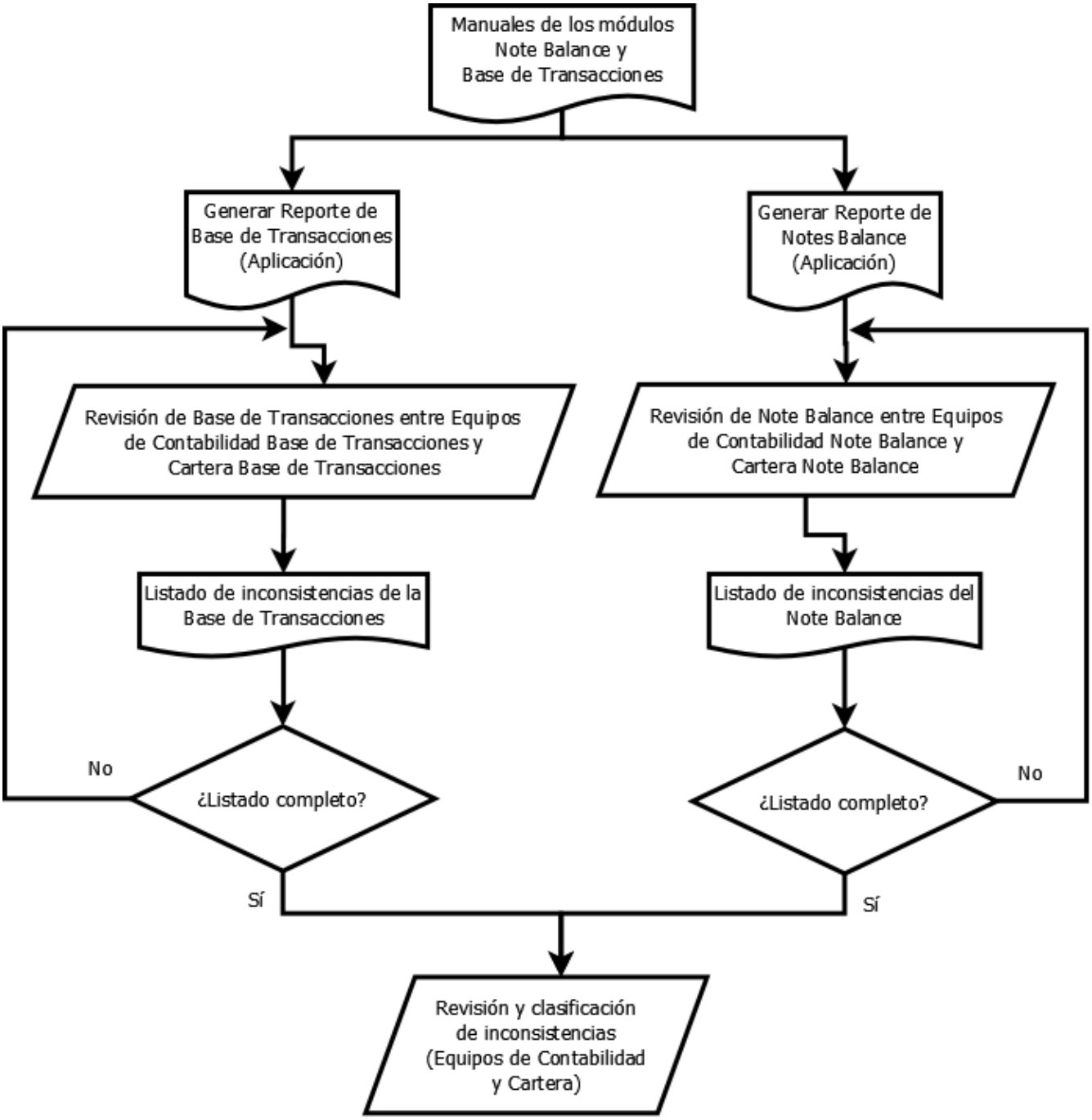


Figura 3.29 Listado de inconsistencias

En la revisión de las inconsistencias, conforme aparecían me encargué de programar la lógica para localizarlas en automático, de aquí surgieron trece reglas las cuáles se anexaron a dicho módulo y que se iban a estar revisando constantemente para solucionarlos desde raíz. Las reglas programadas fueron:

1. Los mortgage que existen en el reporte de BTrx del mes correspondiente, deben de existir en el reporte de NB por falta de SiteList.
2. Los mortgage que existen en el reporte de BTrx del mes correspondiente, deben de existir en el reporte de NB por falta de LenderPago.
3. Los mortgage que existen en el reporte de BTrx del mes correspondiente, deben de existir en el reporte de NB por falta de CompaniaContable.
4. Los mortgage que existen en el reporte de BTrx del mes correspondiente, deben de existir en el reporte de NB por falta de ClubList.
5. Los mortgage que existen en el reporte de BTrx del mes correspondiente, deben de existir en el reporte de NB por falta de Currency.
6. Los mortgage que existen en el reporte de BTrx del mes correspondiente, deben de existir en el reporte de NB por falta de MortgageStatusList.
7. Muestra los mortgage con un '*' que no tienen los balances en 0.
8. Revisa el BeginBalance de NB con el BeginBalance calculado por las transacciones.
9. Revisa el EndBalance de NB con el EndBalance calculado por las transacciones.
10. Verifica que ningún NextPaymentDueDate se encuentre en nulo.
11. Verifica que los TotalPayments no sean montos.
12. Revisa que se recorran los CurrentPrincipalBal de las transacciones.
13. Revisa que no existan pagos sin banco.

Con dichas reglas, continúe trabajando creando el Stored Procedure donde se almacenarían, haciendo que al final devolviera una tabla con el detalle de los *Mortgage Number* que contenían la inconsistencia y mencionando las reglas que se debían checar en cada uno, el nombre del Stored Procedure es *[dbo].[sp_VerificaErrores]* y hace un gran uso de las tablas donde se almacenan la información para Note Balance y Base de Transacciones (Tabla 6). El Stored Procedure fue agregado al proceso ETL para que las inconsistencias aparecieran actualizadas al inicio de cada día sin la necesidad de que un usuario actualizara la información.

Tabla 6 Inconsistencias de Proceso ETL

Elemento	Tipo de Elemento	Elementos Usados
[dbo].[sp_VerificaErrores]	Procedimiento	[dbo].[t_Contract]
		[dbo].[t_Mortgage]
		[dbo].[t_Mortgage]
		[Report].[t_mortgageNBTrax_V2]
		[dbo].[t_MortgagePayment]
		[dbo].[tbl_Inconsistencia]
		[dbo].[tbl_Responsable]
		[dbo].[vw_MortgageNBReport] (Respectivo año y mes)
[dbo].[vw_MortgageNBReport] (Respectivo año y mes)	Vista	[Report].[t_MortgageNBReport] (Respectivo año y mas)

Desde el proyecto de Visual Studio creé la vista **Comparador.aspx**, la cual contendría la muestra de información de las inconsistencias generadas por el Stored Procedure *[dbo].[sp_VerificaErrores]* que se encuentra en el proceso ETL, así como las opciones de filtrar por número de regla, área responsable o si se quiere ver el reporte a detalle o de forma general (Figura 3.30). También la vista cuenta con varios botones, uno para exportar las inconsistencias en formato Excel y otros dos botones que ayudan para corregir las inconsistencias, estos últimos sólo son vistos por los usuarios del Área de Sistemas.

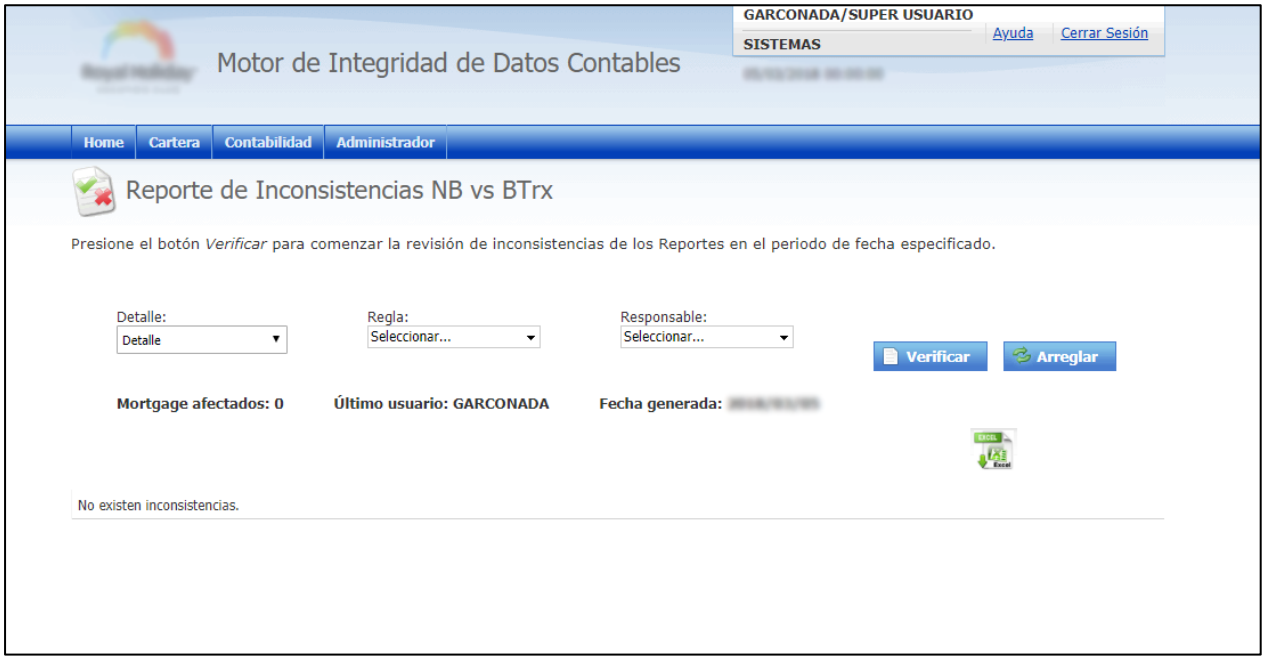


Figura 3.30 Vista de Inconsistencias

Para que la vista funcionara correctamente, al igual que en los módulos anteriores, agregué los Stored Procedures requeridos en el modelo **StoredProcedures.cs** (Figura 3.31), los cuales son el mismo Stored Procedure *[dbo].[sp_VerificaErrores]* que se agregó al proceso ETL, pero además hubieron otros, dos de ellos creados por mí y uno más que ya existía y únicamente lo reutilicé (Tabla 7). Estos Stored Procedure fueron:

- *[dbo].[sp_CorrectorErrores]:* Usado para la corrección de inconsistencias de un Mortgage en específico.
- *[dbo].[sp_CorrectorErrores_Completo]:* Genera el código para corregir todas las inconsistencias detectadas por la aplicación.
- *[dbo].[sp_CorrectorErrores_Completo_Parte2]:* Convierte el resultado de otro Stored Procedure (en este caso para *[dbo].[sp_CorrectorErrores_Completo]*) en un archivo formato TXT.

```
// Procedimientos para la vista de Verificador de Errores.
public class VerificadorErrores
{
    public const string SELECTVERIF_BYPARAMETER = "[dbo].[sp_VerificaErrores]";
    public const string SELECTCORRECTOR_BYPARAMETER = "[dbo].[sp_CorrectorErrores]";
    public const string SELECTCORRECTOR_BYPARAMETERALL = "[dbo].[sp_CorrectorErrores_Completo]";
    public const string SELECTCORRECTOR_BYPARAMETERALL2 = "[dbo].[sp_CorrectorErrores_Completo_Parte2]";
}
```

Figura 3.31 Stored Procedure para Inconsistencias

Tabla 7 Stored Procedures para corrección de inconsistencias

Elemento	Tipo de Elemento	Elementos Usados
[dbo].[sp_CorrectorErrores]	Procedimiento	[VPOS].[BankTransaction]
		[Concord].[CSCPostPayments]
		[dbo].[t_Contract]
		[dbo].[t_Mortgage]
		[Report].[t_mortgageNBTrax]
		[dbo].[t_MortgagePayment]
		[Report].[t_MortgageNBReport] (Respectivo año y mes)
[dbo].[sp_CorrectorErrores_Completo]	Procedimiento	[VPOS].[BankTransaction]
		[CentralCash].[cat_bank]
		[dbo].[CLookup]
		[Concord].[CSCPostPayments]
		[dbo].[t_Contract]
		[dbo].[t_Mortgage]
		[dbo].[t_MortgageMemo]
		[Report].[t_mortgageNBTrax_V2]
		[dbo].[t_MortgagePayment]
		[dbo].[tbl_Inconsistencia]
		[dbo].[vw_MortgageNBReport] (Respectivo año y mes)
[dbo].[sp_CorrectorErrores_Completo_Parte2]	Procedimiento	

En el caso del llenado de los filtros no fue necesario crear un catálogo desde la Base de Datos ya que estos siempre son los mismos, por lo que se pusieron directo en cada uno de los DropDownList y desde Back-End sólo fue necesario crear el controlador para los Stored Procedure ya antes mencionado, junto con la lógica para los respectivos botones.

Lo primero en programar fue la lógica para la función Page_Load, donde revisa el perfil y rol del usuario para saber los botones que permite que utilice el usuario junto con llenado de la tabla donde se muestran las inconsistencias detectadas por la aplicación unido con un resumen de la información para saber el número de Mortgage con inconsistencia, el usuario que ejecutó por última vez la revisión junto con la fecha de dicha ejecución (Figura 3.32).

```
// Se crea el objeto donde se guardarán los valores para el procedimiento.
Constantes.FiltroComparador objFiltroComparador = new Constantes.FiltroComparador();

// Envía los valores filtrados al procedimiento que realiza la búsqueda.
//objFiltroComparador.MortgageStatus_Comparador = (ms == "") ? "'Active'" : ms;
objFiltroComparador.Activo_Comparador = true;
objFiltroComparador.Detalle_Comparador = detalleTabla;
objFiltroComparador.User_Comparador = Session["usuario"].ToString();
objFiltroComparador.Rol_Comparador = Session["perfil"].ToString();
objFiltroComparador.Regla_Comparador = null;

// En caso de no haber ningún fallo, genera el Grid
try
{
    ComparadorProcess objComparadorProcess = new ComparadorProcess();
    System.Data.DataTable listComparador = objComparadorProcess.GetByIdParameterDetalle(objFiltroComparador);

    DataView vista = new DataView(listComparador);

    GridError.DataSource = listComparador;
    GridError.DataBind();
    GridError.Visible = true;
    tablaPrincipal.Visible = true;

    if (detalleTabla == true)
    {
        btnExportExcel.Visible = false;
        GridError.Columns[3].Visible = true;
        GridError.Columns[4].Visible = false;
        GridError.Columns[5].Visible = false;
        GridError.Columns[6].Visible = false;
        GridError.Columns[7].Visible = false;
    }
    else
    {
        btnExportExcel.Visible = true;
        GridError.Columns[3].Visible = false;
        GridError.Columns[4].Visible = true;
        GridError.Columns[5].Visible = false;
        GridError.Columns[6].Visible = true;
        GridError.Columns[7].Visible = true;

        registroMortgage.Text = "Mortgage afectados: " + (vista.ToTable(true, "varCampo").Rows.Count).ToString();
        ultimoUser.Text = "Último usuario: " + (vista.ToTable(true, "varUser").Rows[0][0]).ToString();
        ultimaFecha.Text = "Fecha generada: " + (vista.ToTable(true, "varFecha").Rows[0][0]).ToString();
        reglaLi.Visible = true;
        reglaDrop.Visible = true;
        responsableLi.Visible = true;
        responsableDrop.Visible = true;
    }
```

Figura 3.32 Muestra de tabla con inconsistencias

Después de tener la función Page_Load completa, continué con las funciones btnVerificar_Click, donde la lógica fue muy parecida a la de Page_Load con la diferencia de agregar nueva ejecución del Stored Procedure *[dbo].[sp_VerificaErrores]* y así actualizar la tabla de inconsistencias, y la btnExportExcel_Click con el que la información de la tabla se exporta en Excel.

Por último, la función que completé fue para el funcionamiento del botón Arreglar, en esta función (btnArreglar_Click) se agregan los correos de los integrantes del Área de Sistemas

quienes van a recibir el código que repara todas las inconsistencias para que, después de que lo revisen y validen, lo ejecuten sabiendo que no existirá ningún problema mayor (Figura 3.33).

```
protected void btnArreglar_Click(object sender, EventArgs e)
{
    string correos = "guillermo.arconada@ait.mx; ayudadb@royal-holiday.com; ivelasco@royal-holiday.com; rricardez@royal-holiday.com";

    Constantes.FiltroCorrector objFiltroCorrector = new Constantes.FiltroCorrector();

    // En caso de no haber ningún fallo, genera el Grid
    try
    {
        CorrectorProcess objCorrectorProcess = new CorrectorProcess();

        List<CorrectorBalance> listCorrector = null;
        if (objCorrectorProcess.GetByIdParameterCompleto(objFiltroCorrector).Count() > 0)
        {
            listCorrector = objCorrectorProcess.GetByIdParameterCompleto(objFiltroCorrector);

            listCorrector = objCorrectorProcess.GetByIdParameterParte2(objFiltroCorrector);
        }


        //string query = (listCorrector.Count() > 0) ? (listCorrector.First().Query_Comparador).ToString() : "";

        using (SqlConnection conn = new SqlConnection("Data Source=TSWReport; User ID= usr_CierresCartera_Act_DB; password= M3z0|gs2B16~;Initial Catalog=msdb;"))
        {
            conn.Open();
            SqlCommand cmd = new SqlCommand("[Sistemas].[EnviaMail]", conn);
            cmd.CommandType = CommandType.StoredProcedure;

            cmd.Parameters.Add(new SqlParameter("@body", "El archivo adjunto contiene el query para corregir las inconsistencias."));
            cmd.Parameters.Add(new SqlParameter("@body_format", "TEXT"));
            cmd.Parameters.Add(new SqlParameter("@recipients", correos));
            //cmd.Parameters.Add(new SqlParameter("@recipients", "guillermo.arconada@ait.mx;"));
            cmd.Parameters.Add(new SqlParameter("@subject", "CORRECCIÓN TODAS LAS INCONSISTENCIAS"));
            cmd.Parameters.Add(new SqlParameter("@FileNameAttachment", "Query.txt"));

            SqlDataReader sdr = cmd.ExecuteReader();

            conn.Close();
        }
    }
    catch (FormatException ex)
    {
        string b = ex.ToString();
    }
    catch (OutOfMemoryException ex)
    {
        string hola = ex.ToString();
    }

    textoActivar.Text = "Correo con las correcciones enviado.";
}
```

Figura 3.33 Función para arreglar inconsistencias

Finalizada la lógica del módulo, se procede a hacer pruebas, inicialmente por parte del Equipo de Testing e inmediatamente después por las Áreas de Contabilidad y Cartera para así comenzar a corregir todas las inconsistencias mostradas por la aplicación. De este modo, cada Área diariamente descarga el reporte de Inconsistencias para solucionar todo lo que les corresponda y así llegar a cada cierre de mes sin ningún fallo (Figuras 3.34 y 3.35) mientras que el Equipo de Documentación termina los manuales y documentos necesarios para la entrega y liberación total del proyecto.

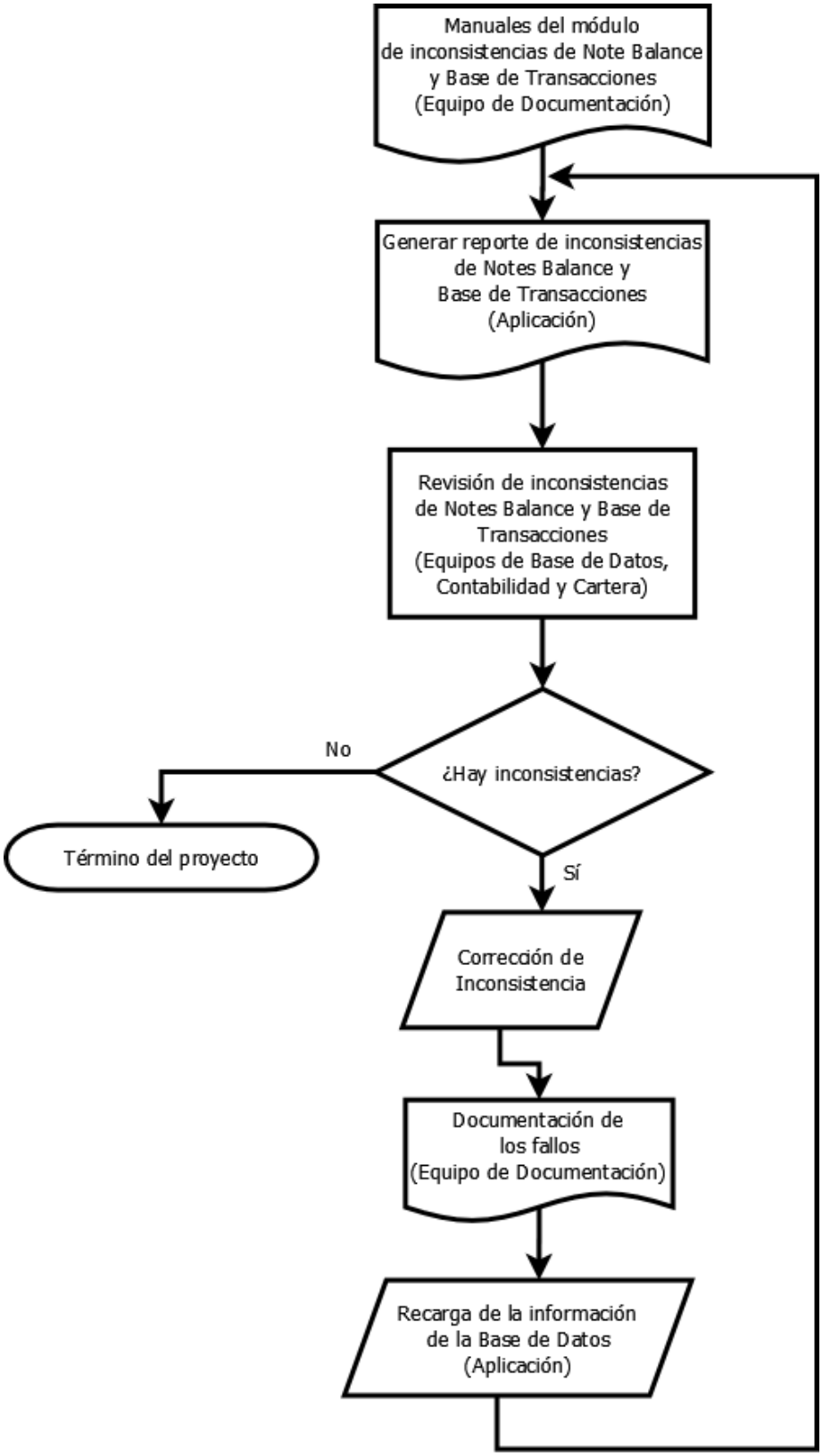


Figura 3.34 Corrección de inconsistencias

M34

	A	B	C	D	E	F	G
1		Regla	Descripción	Mortgage Number			
2	1	8	BeginBalance incorrecto	1-501377-4			
3	2	9	EndBalance incorrecto	1-501377-4			
4	3	12	CurrentPrincipalBal mal recorridos	1-501377-4			
5	4	3	Mortgage no existe en NB por CompaniaContable	10-81346-1			
6	5	8	BeginBalance incorrecto	102-624381-1			
7	6	9	EndBalance incorrecto	102-624381-1			
8	7	12	CurrentPrincipalBal mal recorridos	102-624381-1			
9	8	11	TotalPayment incorrecto	102-815312-1			
10	9	8	BeginBalance incorrecto	150-400901-1			
11	10	9	EndBalance incorrecto	150-400901-1			
12	11	8	BeginBalance incorrecto	150-615192-1			
13	12	9	EndBalance incorrecto	150-615192-1			
14	13	12	CurrentPrincipalBal mal recorridos	150-615192-1			
15	14	12	CurrentPrincipalBal mal recorridos	155-402191-1			
16	15	11	TotalPayment incorrecto	155-403136-1			
17	16	12	CurrentPrincipalBal mal recorridos	155-403136-1			
18	17	13	Banco nulo	155-403136-1			
19	18	10	NextPaymentDueDate nulo	407-633437-1			
20	19	12	CurrentPrincipalBal mal recorridos	5-604029-4			
21	20	11	TotalPayment incorrecto	6-615454-1			
22	21	12	CurrentPrincipalBal mal recorridos	6-615454-1			
23	22	13	Banco nulo	6-615454-1			
24	23	8	BeginBalance incorrecto	9-501282-1			
25	24	9	EndBalance incorrecto	9-501282-1			
26	25	12	CurrentPrincipalBal mal recorridos	9-501282-1			
27	26	12	CurrentPrincipalBal mal recorridos	9-501696-5			
28							
29							
30							
31							
32							
33							
34							
35							
36							

Figura 3.35 Reporte de Inconsistencias

CAPÍTULO 4: RESULTADOS

Al finalizar el proyecto, a pesar de no poder realizar la idea inicial que tenía el cliente por lo mismo de que no se conocía la magnitud del problema, con las adaptaciones y requerimientos a los que se llegaron a un acuerdo se obtuvo como resultado una aplicación completa que no únicamente fue capaz de detectar las inconsistencias de las Bases de Datos, sino que ayudó en varios aspectos más a la empresa:

- Teniendo versiones finales, únicas y completas de los Reportes de Note Balance y Base de Transacciones.
- Reduciendo la carga de trabajo para el Área de Sistemas al tener que corregir errores en tiempos críticos, puesto que ahora tienen todo el transcurso del mes para corregir dichos errores.
- Disminuyendo el tiempo de los cierres de mes, de ocho días que duraban estos, a cuatro horas.
- Encontrando fallos a nivel operativo que causaban grandes pérdidas de dinero a la empresa.

Por otro lado, así como existieron grandes beneficios con la aplicación, igual hubieron contras que iban más allá del beneficio de la empresa, estos problemas fueron, debido a la disminución del tiempo de trabajo tanto para las Áreas de Sistemas, Cartera y Contabilidad, el despido de una parte de los integrantes de dichas Áreas y una resistencia al cambio para usar la aplicación por parte de los integrantes que quedaron por el mismo miedo generado al ver que la función de varios ya no era esencial.

Como beneficio propio en base a los satisfactorios resultados del proyecto, se me permitió continuar con el proyecto con otros módulos de la empresa para así poder acabar con las inconsistencias de toda la base de datos y dedicarse a una depuración completa de todos los sistemas que se tienen. Para esto, se nos asignó a los integrantes que más participamos en el proyecto, pequeños equipos de trabajo para monitorear que corrigieran los sistemas con fallos.

CONCLUSIONES

Ya había participado a otros proyectos como fueron:

- Sistema para Control y Reportes.
- Sistema para Seguimiento de Compra/Venta de Materiales.
- App móvil para Sector Salud.

Gracias a esos proyectos pude llegar con conocimientos previos al actual en desarrollo con patrones MVC y MVVM tanto con ASP.NET como JavaScript y PHP, así como manejo de distintos Motores de Bases de Datos como SQL Server, MySQL, Oracle y DB2. Pero en este proyecto el nivel para uso de MVC y sobretodo SQL Server, ya que la Base de Datos contó con uno de los mayores pesos en el proyecto, fue mucho mayor al que conocía, de tal forma que la experiencia conseguida fue enorme y la prueba fue que en el crecimiento del proyecto mencionado en este reporte, se pudo realizar más actividades en mucho menor tiempo.

En cuanto al proyecto en que logré participar fue una gran experiencia en varios aspectos, porque me sirvió para reafirmar mucho mis conocimientos en distintas áreas de la carrera que estudié, así como aprender a desenvolverme en el mundo profesional, adquiriendo seguridad para hablar con otras personas y dar opiniones en vez de cerrarse a nada más recibir órdenes.

Por otro lado, es interesante pero triste saber que la carrera es muy desvalorada por muchas otras áreas, esto causó un conflicto en un inicio ya que al apenas comenzar a trabajar en dicha empresa, el área de Sistemas había perdido toda credibilidad y, aunque para el final del proyecto se recuperó ésta, nada más fue para las personas que formamos parte en el proyecto, por lo que muchas actividades correspondientes a otros equipos de trabajo querían que nosotros los realizáramos por esa confianza que ganamos.

Todo esto no fue sólo a causa de malos trabajos de un área, sino que es algo en el que todos se van orillando por lo mismo que observan un freno de los superiores para hacer crecer a todos, causando que eviten trabajar y sólo hagan el mínimo esfuerzo, pero cuidando verse siempre como un elemento útil para evitarse cualquier problema serio. De aquí una de las enseñanzas por parte de la Facultad, el cual es ser lo suficientemente éticos y responsables profesionalmente y entender que realmente no hay ningún área ni mucho menos una persona completamente esencial, rompiendo esa lucha de ver nada más por el área y aventarse la bolita y aprender a ver como el equipo que realmente se es, toda la empresa, aprender a buscar soluciones en lugar de buscar al culpable y ayudar a otros ya que así como uno puede brindar su conocimiento, en algún momento se lo pueden brindar a uno, sólo así uno puede demostrar su mejor potencial y continuar creciendo para hacer cosas de mayor nivel.

GLOSARIO

Equipo: Grupo de personas que se organiza para realizar una actividad o trabajo.

Base de Datos: Conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

Back-End: Es aquel que se encuentra del lado del servidor, es decir, se encarga de interactuar con base de datos, verificar manejo de sesiones de Usuario, montar la página en un servidor y desde éste "server" todas las vistas que el Front-End crea.

Front-End: Son todas aquellas tecnologías que corren del lado del cliente, es decir, todas las tecnologías que corren en el navegador web, generalizándose en tres tecnologías, HTML, CSS y JavaScript.

Note Balance: Reporte general de la cantidad de dinero que se mueve cada mes, sirve para verificar que no exista un desface de dinero con los meses anteriores.

Base de Transacciones: Reporte específico de las transacciones y movimientos en el mes de cada uno de los socios de la empresa.

Diagrama Entidad-Relación: Tipo de diagrama de flujo que ilustra cómo las "entidades", como personas, objetos o conceptos, se relacionan entre sí dentro de un sistema. Utilizado a menudo para diseñar o depurar bases de datos relacionales en los campos de ingeniería de software.

Vista: Alternativa para mostrar datos de varias tablas. Es como una tabla virtual que almacena una consulta. Los datos accesibles a través de la vista no están almacenados en la base de datos como un objeto.

Stored Procedure: Por medio de estos, el sistema de la base de datos es capaz de ejecutar un conjunto de instrucciones bien coordinadas entre sí que afectan la información con el fin de lograr un objetivo dentro del sistema.

Pantalla: Conjunto de Base de Datos, Back-End y Front-End el cual da forma y uso completo a una vista para un usuario cliente, visto desde un navegador web.

Socio: Nombre que se le da a los clientes de la empresa que van a estar haciendo uso de las instalaciones para hospedarse.

Membresía: Cobro que se debe de realizar para seguir perteneciendo como socio de la empresa o institución y, de esta forma, continuar recibiendo los beneficios acorde al nivel de ésta.

Patrón de diseño: Solución probada para un problema en un contexto.

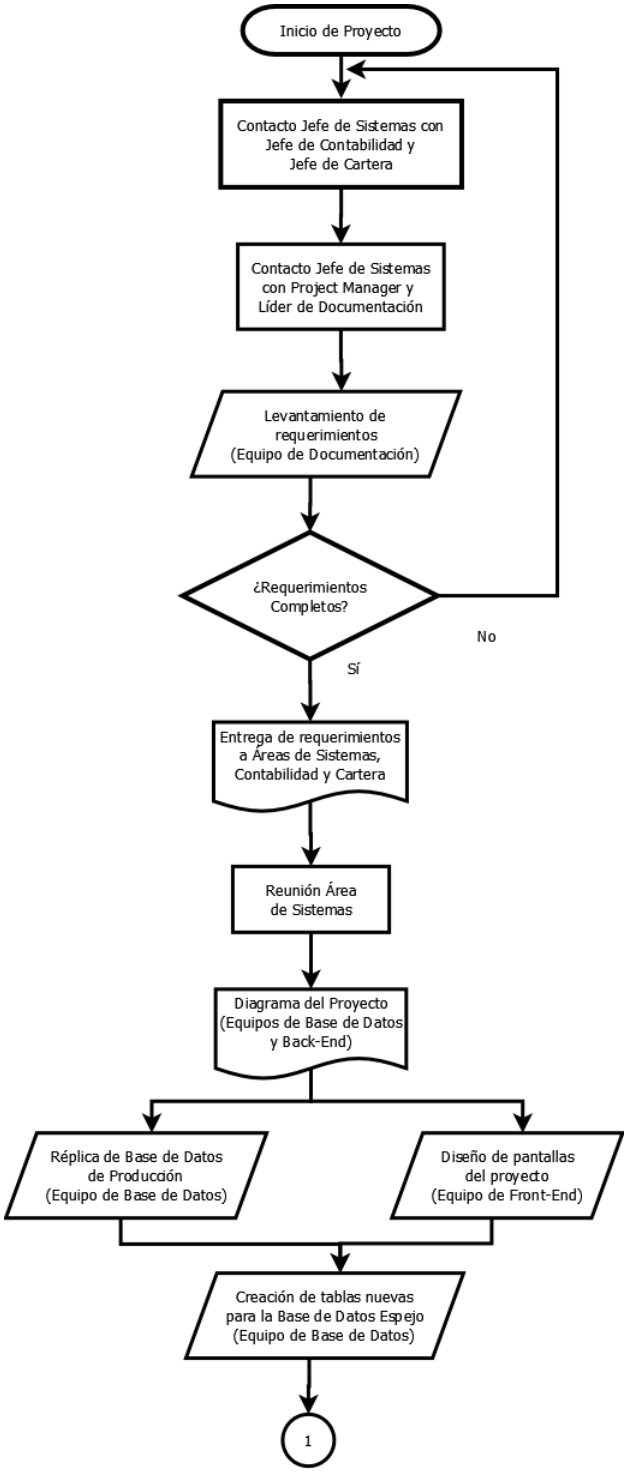
Interfaz: Utiliza un conjunto de imágenes y objetos gráficos. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

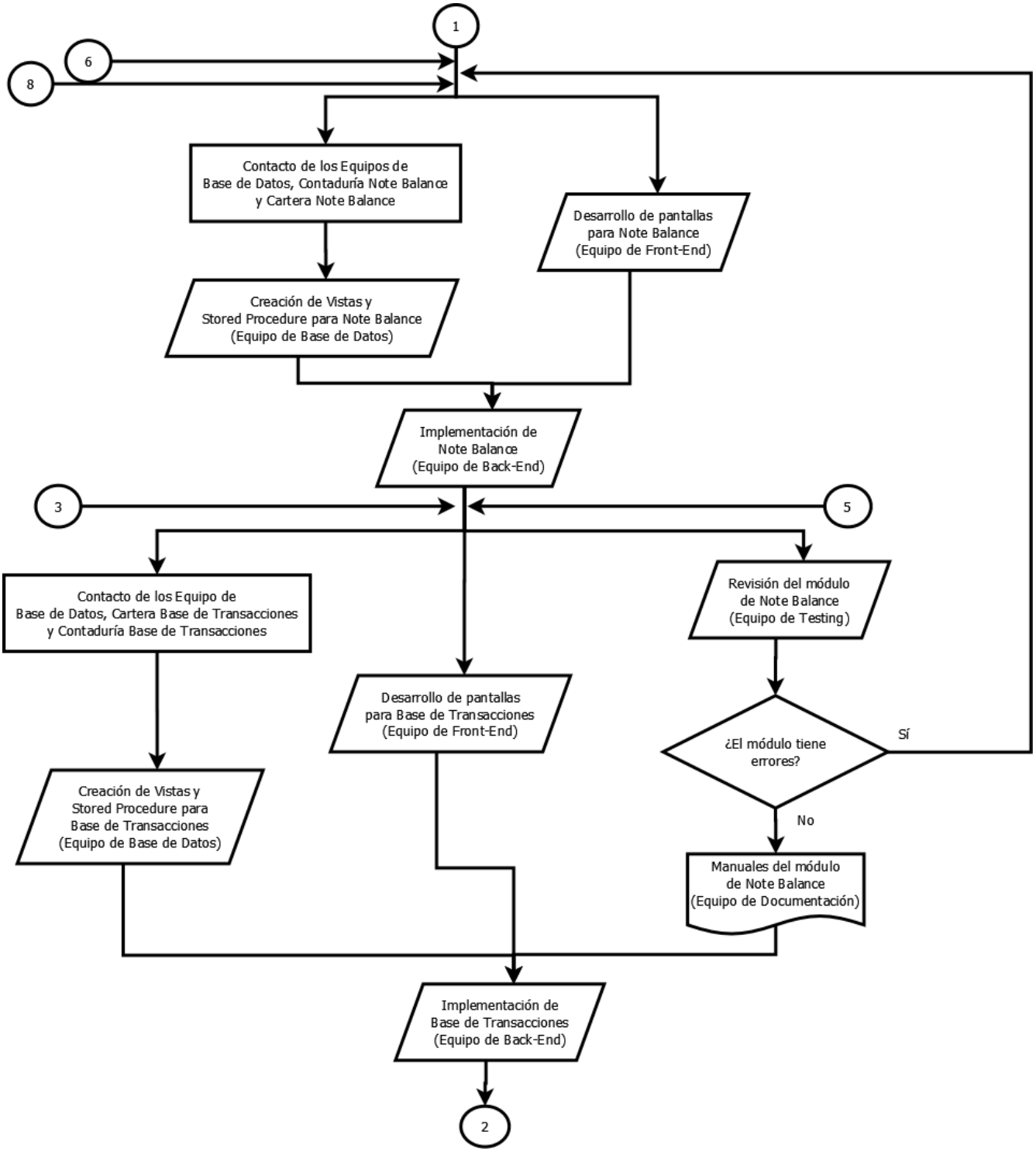
Framework: Conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

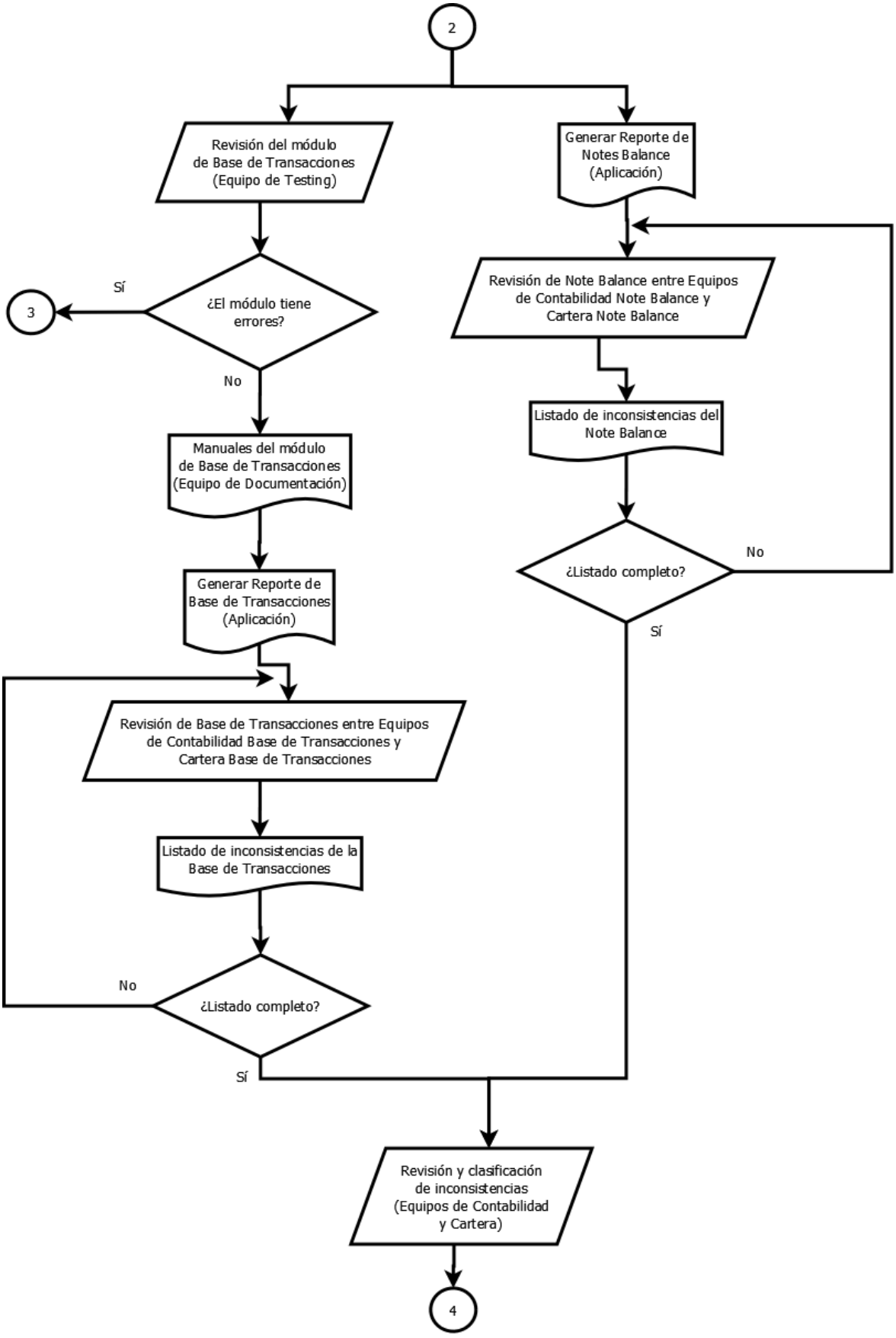
Lógica de negocio: Conjunto de reglas que se siguen en el software para reaccionar ante distintas situaciones. Aparte de marcar un comportamiento cuando ocurren cosas dentro de un software, también tiene normas sobre lo que se puede hacer y lo que no se puede hacer.

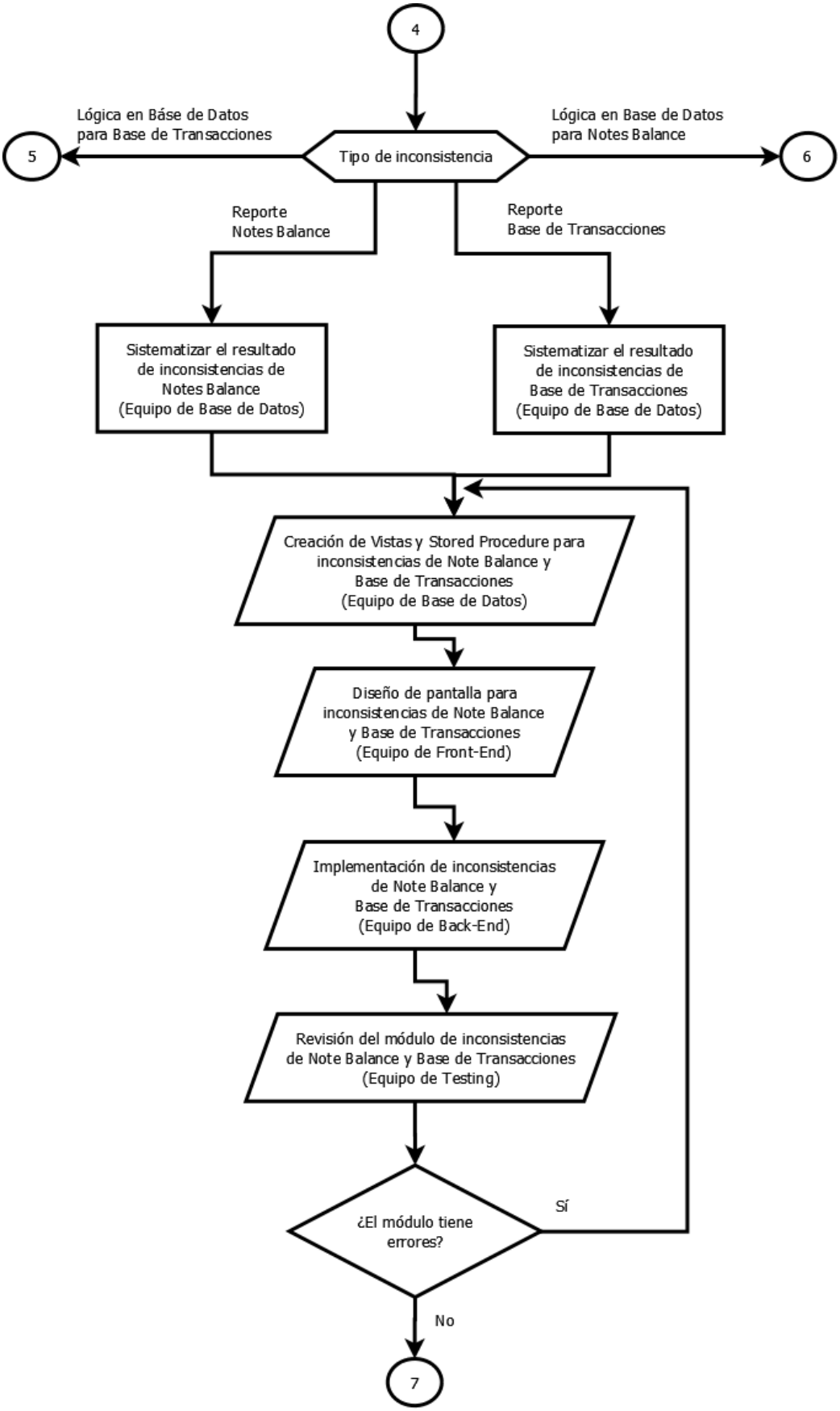
ANEXOS

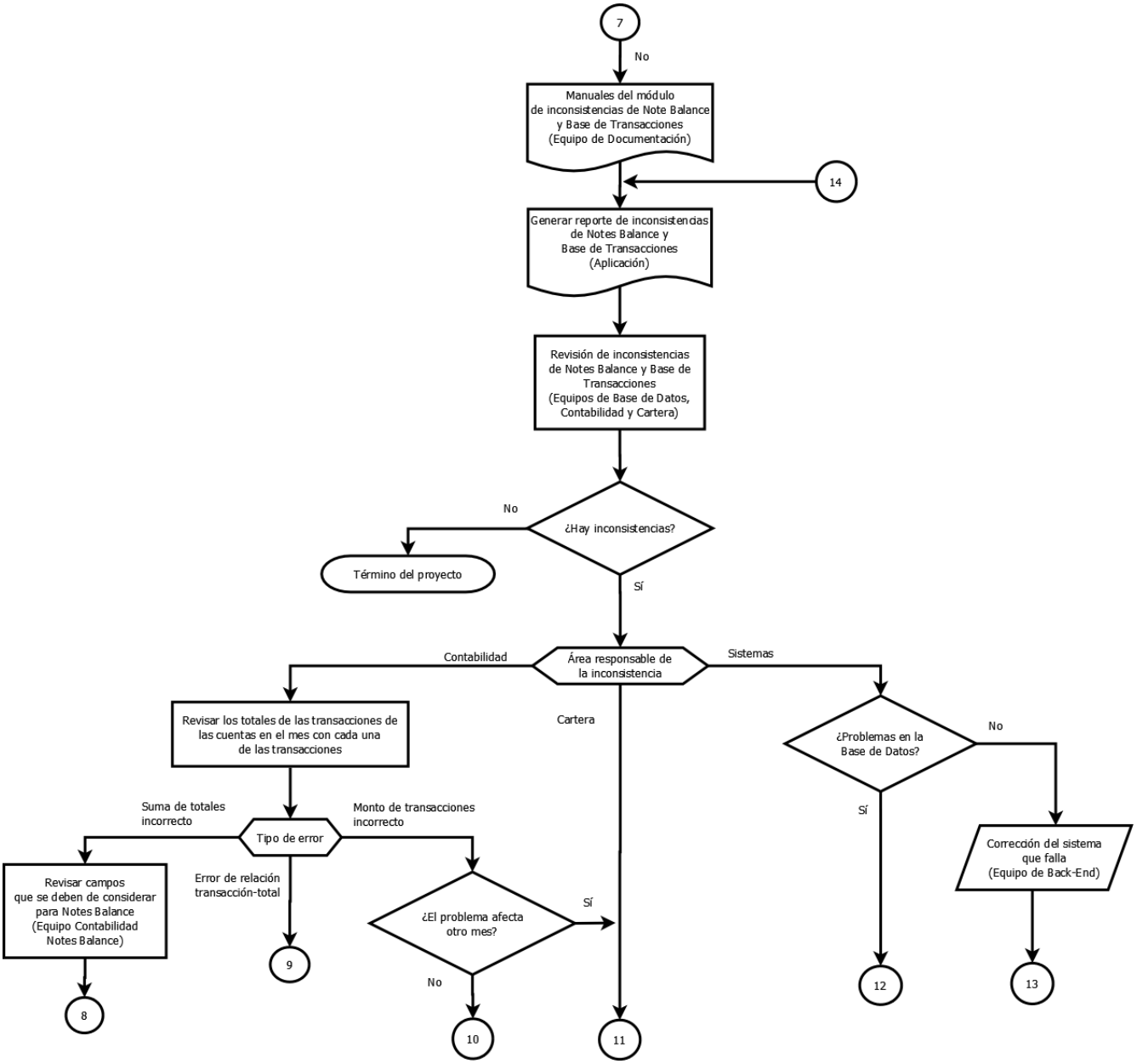
Anexo 1: Diagrama del Proyecto

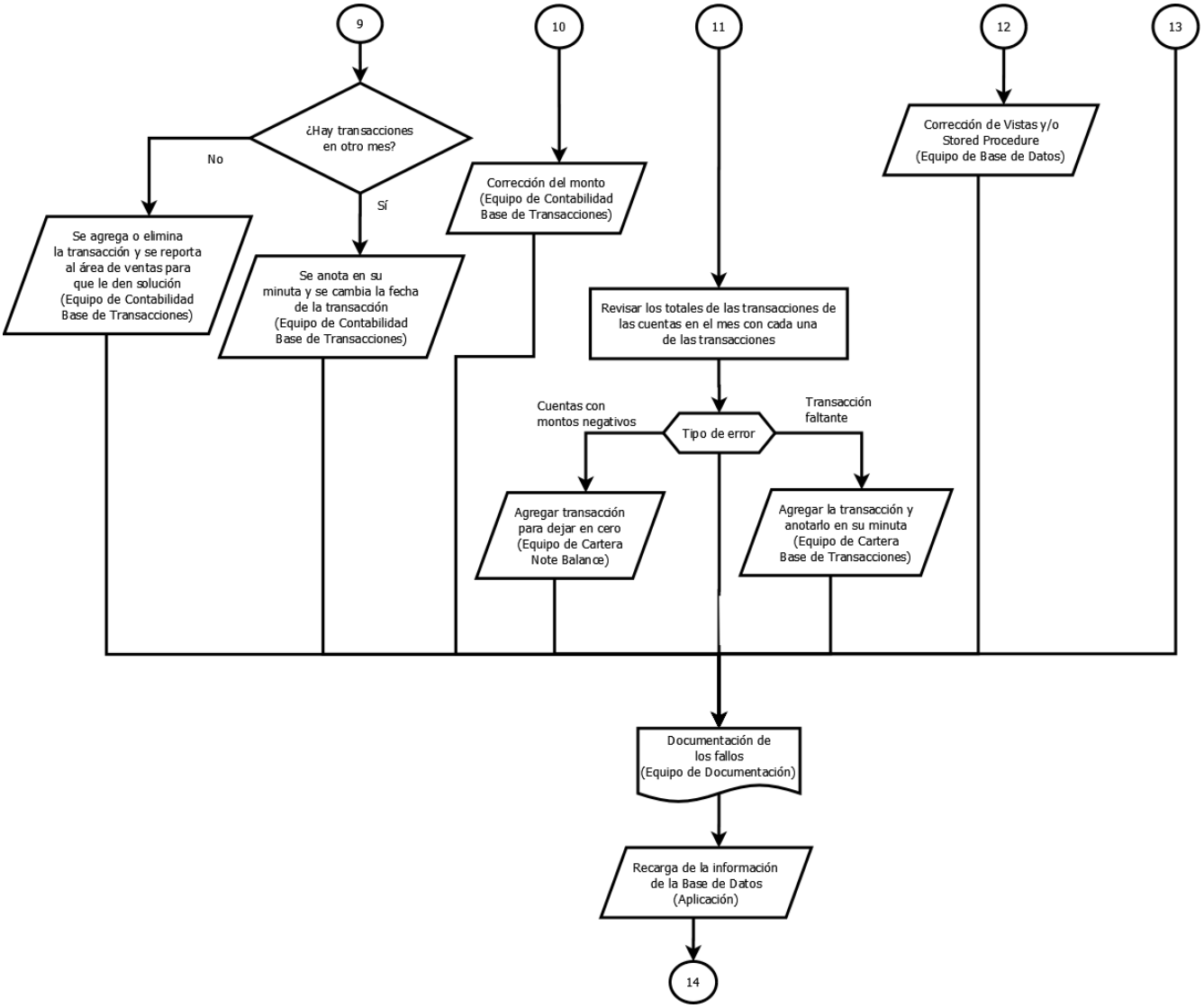



1
6
8
Contacto de los Equipos de Base de Datos, Contaduría Note Balance y Cartera Note Balance
Desarrollo de pantallas para Note Balance (Equipo de Front-End)
Creación de Vistas y Stored Procedure para Note Balance (Equipo de Base de Datos)
Implementación de Note Balance (Equipo de Back-End)
3
5
Contacto de los Equipo de Base de Datos, Cartera Base de Transacciones y Contaduría Base de Transacciones
Desarrollo de pantallas para Base de Transacciones (Equipo de Front-End)
Revisión del módulo de Note Balance (Equipo de Testing)
¿El módulo tiene errores?
Sí
No
Creación de Vistas y Stored Procedure para Base de Transacciones (Equipo de Base de Datos)
Manuales del módulo de Note Balance (Equipo de Documentación)
Implementación de Base de Transacciones (Equipo de Back-End)
2


2
Revisión del módulo de Base de Transacciones (Equipo de Testing)
Generar Reporte de Notes Balance (Aplicación)
Sí
3
¿El módulo tiene errores?
No
Manuales del módulo de Base de Transacciones (Equipo de Documentación)
Generar Reporte de Base de Transacciones (Aplicación)
Revisión de Base de Transacciones entre Equipos de Contabilidad Base de Transacciones y Cartera Base de Transacciones
Listado de inconsistencias de la Base de Transacciones
No
¿Listado completo?
Sí
Revisión de Note Balance entre Equipos de Contabilidad Note Balance y Cartera Note Balance
Listado de inconsistencias del Note Balance
No
¿Listado completo?
Sí
Revisión y clasificación de inconsistencias (Equipos de Contabilidad y Cartera)
4


4
Lógica en Báse de Datos
para Base de Transacciones
5
Tipo de inconsistencia
Lógica en Base de Datos
para Notes Balance
6
Reporte
Notes Balance
Reporte
Base de Transacciones
Sistematizar el resultado
de inconsistencias de
Notes Balance
(Equipo de Base de Datos)
Sistematizar el resultado
de inconsistencias de
Base de Transacciones
(Equipo de Base de Datos)
Creación de Vistas y Stored Procedure para
inconsistencias de Note Balance y
Base de Transacciones
(Equipo de Base de Datos)
Diseño de pantalla para
inconsistencias de Note Balance
y Base de Transacciones
(Equipo de Front-End)
Implementación de inconsistencias
de Note Balance y
Base de Transacciones
(Equipo de Back-End)
Revisión del módulo de inconsistencias
de Note Balance y Base de Transacciones
(Equipo de Testing)
¿El módulo tiene
errores?
Sí
No
7


7
No
Manuales del módulo de inconsistencias de Note Balance y Base de Transacciones (Equipo de Documentación)
14
Generar reporte de inconsistencias de Notes Balance y Base de Transacciones (Aplicación)
Revisión de inconsistencias de Notes Balance y Base de Transacciones (Equipos de Base de Datos, Contabilidad y Cartera)
No
¿Hay inconsistencias?
Término del proyecto
Sí
Contabilidad
Área responsable de la inconsistencia
Sistemas
Cartera
Revisar los totales de las transacciones de las cuentas en el mes con cada una de las transacciones
Suma de totales incorrecto
Tipo de error
Monto de transacciones incorrecto
Revisar campos que se deben de considerar para Notes Balance (Equipo Contabilidad Notes Balance)
Error de relación transacción-total
¿El problema afecta otro mes?
Sí
No
8
9
10
11
¿Problemas en la Base de Datos?
No
Sí
Corrección del sistema que falla (Equipo de Back-End)
12
13


9
10
11
12
13
¿Hay transacciones en otro mes?
No
Sí
Se agrega o elimina la transacción y se reporta al área de ventas para que le den solución (Equipo de Contabilidad Base de Transacciones)
Se anota en su minuta y se cambia la fecha de la transacción (Equipo de Contabilidad Base de Transacciones)
Corrección del monto (Equipo de Contabilidad Base de Transacciones)
Revisar los totales de las transacciones de las cuentas en el mes con cada una de las transacciones
Corrección de Vistas y/o Stored Procedure (Equipo de Base de Datos)
Cuentas con montos negativos
Tipo de error
Transacción faltante
Agregar transacción para dejar en cero (Equipo de Cartera Note Balance)
Agregar la transacción y anotarlo en su minuta (Equipo de Cartera Base de Transacciones)
Documentación de los fallos (Equipo de Documentación)
Recarga de la información de la Base de Datos (Aplicación)
14

Anexo 2: Ejemplo de Modelo

```
namespace GestorCartera.BusinessEntities
{
    public class LoginBalance
    {
        #region Variables

        private String _nombreperfil;
        private DateTime _fechamodifusuario;
        private String _department;

        #endregion

        #region Properties

        public String NombrePerfil
        {
            get { return _nombreperfil; }
            set { _nombreperfil = value; }
        }

        public DateTime FechaModifUsuario
        {
            get { return _fechamodifusuario; }
            set { _fechamodifusuario = value; }
        }

        public String Department
        {
            get { return _department; }
            set { _department = value; }
        }

        #endregion

        #region Constructors

        public LoginBalance()
        {
        }

        public LoginBalance(String nombreperfil, DateTime fechamodifusuario, String department)
        {
            this.NombrePerfil = nombreperfil;
            this.FechaModifUsuario = fechamodifusuario;
            this.Department = department;
        }

        #endregion
    }
}
```

Anexo 3: Ejemplo de Controlador

```
public List<LoginBalance> GetByIdParameter(Constantes.FiltroSesion filtroSesion)
{
    ConnectionBD db = ConnectionBD.Instance;
    List<LoginBalance> perfil = new List<LoginBalance>();
    using (DbConnection dbConn = db.Connect())
    {
        dbConn.Open();
        using (DbTransaction dbTrans = dbConn.BeginTransaction())
        {
            try
            {
                perfil = this.GetByParameter(filtroSesion, db, dbTrans);
                dbTrans.Commit();
                return perfil;
            }
            catch (SqlException ex)
            {
                dbTrans.Rollback();
                throw ex;
            }
            finally
            {
                dbConn.Close();
            }
        }
    }
}

public List<LoginBalance> GetByParameter(Constantes.FiltroSesion filtroSesion, ConnectionBD db, DbTransaction dbTrans)
{
    AuthenticateSoapClient authentication = new AuthenticateSoapClient();

    string login = authentication.Login(filtroSesion.Usuario_Sesion, filtroSesion.Psw_Sesion);

    if ((login != "Invalid Username or Password") || (filtroSesion.Usuario_Sesion == "garconada"))
    {
        DbCommand dbCommand = db.GetStoredProcCommand(StoredProcedures.LogIn.INICIOSESION_BYPARAMETER);
        db = SetParameters(db, dbCommand, filtroSesion);
        dbCommand.Transaction = dbTrans;
        using (IDataReader dataReader = db.ExecuteReader(dbCommand))
        {
            List<LoginBalance> perfil = this.GetFromDataReader(dataReader);

            return perfil;
        }
    }
    else
    {
        List<LoginBalance> perfil = null;

        return perfil;
    }
}

private ConnectionBD SetParameters(ConnectionBD db, DbCommand dbCommand, Constantes.FiltroSesion filtroSesion)
{
    if ((filtroSesion.Usuario_Sesion != null) || (filtroSesion.Usuario_Sesion != ""))
        db.AddInParameter(dbCommand, "@usuario", DbType.String, filtroSesion.Usuario_Sesion);
    else
        db.AddInParameter(dbCommand, "@usuario", DbType.String, DBNull.Value);

    return db;
}

private List<LoginBalance> GetFromDataReader(IDataReader dataReader)
{
    string stringDate = "01-01-2015";
    List<LoginBalance> perfil = new List<LoginBalance>();
    while (dataReader.Read())
    {
        perfil.Add(new LoginBalance(
            dataReader.IsDBNull(0) ? string.Empty : dataReader.GetString(0),
            dataReader.IsDBNull(1) ? Convert.ToDateTime(stringDate) : dataReader.GetDateTime(1),
            dataReader.IsDBNull(2) ? string.Empty : dataReader.GetString(2)
        ));
    }

    dataReader.Close();

    return perfil;
}
```

Anexo 4: [Report].[sp_NotesBalance_Prm_IVM]

```
ALTER  PROCEDURE [Report].[sp_NotesBalance_Prm_IVM]

@FechaInicial DATE = null,
@FechaFinal DATE = null,
@MortgageStatusList VARCHAR (MAX) = null,
@SiteList VARCHAR (MAX)  = null,
@clubList varchar (max) = null,
@CompaniaContable varchar (max) = null,
@LenderPago varchar (max) = null,
@Currency varchar (max) = null

,@pagina integer = null
,@regxpag integer = null

AS

IF NOT OBJECT_ID('tempdb.dbo.##TBLPASOREPORTNB') IS NULL DROP TABLE ##TBLPASOREPORTNB

if @FechaInicial is null SET @FechaInicial = GETDATE() - 1;
IF @FechaFinal IS NULL SET @FechaFinal = GETDATE();

IF @MortgageStatusList IS NULL
SELECT
       @MortgageStatusList =
       STUFF((SELECT
                     ', ' + LTRIM(RTRIM(CONVERT(CHAR(10), p2.CLookupID)))
              FROM dbo.CLookup AS p2 WITH (NOLOCK)
              WHERE p2.CLookupParentID = 41
              FOR
              XML PATH (''))
       , 1, 1, '')

IF @SiteList IS NULL
SELECT
       @SiteList =
              STUFF((SELECT
                     ', ' + LTRIM(RTRIM(CONVERT(CHAR(10), p2.SiteID)))
              FROM dbo.t_Site AS p2 WITH (NOLOCK)
              WHERE p2.SiteTypeID = 720
              FOR
              XML PATH (''))
       , 1, 1, '')

IF @clubList IS NULL
SELECT
       @clubList =
       STUFF((SELECT
                     ', ' + LTRIM(RTRIM(CONVERT(CHAR(10), p2.ClubID)))
              FROM dbo.Club AS p2 WITH (NOLOCK)
              FOR
              XML PATH (''))
       , 1, 1, '')

IF @CompaniaContable IS NULL
SELECT
       @CompaniaContable =
```

```
		STUFF((SELECT
			', ' + LTRIM(RTRIM(CONVERT(CHAR(10), p2.CLookupID)))
		FROM dbo.CLookup AS p2 WITH (NOLOCK)
		WHERE p2.CLookupParentID = 29
		FOR
		XML PATH (''))
	, 1, 1, '')

IF @LenderPago IS NULL
SELECT
	@LenderPago =
	STUFF((SELECT
			', ' + LTRIM(RTRIM(CONVERT(CHAR(10), p2.CLookupID)))
		FROM dbo.CLookup AS p2 WITH (NOLOCK)
		WHERE p2.CLookupParentID = 42
		FOR
		XML PATH (''))
	, 1, 1, '')

IF @Currency IS NULL
SELECT
	@Currency =
	STUFF((SELECT
			', ' + LTRIM(RTRIM(CONVERT(CHAR(10), p2.CurrencyID)))
		FROM dbo.Currency AS p2 WITH (NOLOCK)
		FOR
		XML PATH (''))
	, 1, 1, '')

DECLARE @LOCAL_FechaInicial DATE, @LOCAL_FechaFinal DATE
SELECT
	@LOCAL_FechaInicial = @FechaInicial,
	@LOCAL_FechaFinal = @FechaFinal

IF NOT OBJECT_ID('tempdb.dbo.#ClubList') IS NULL DROP TABLE #ClubList
----Get clubList  List
CREATE TABLE #ClubList(ClubListid INT)
INSERT INTO #clublist (clublistid)
	SELECT
		ilist
	FROM fnonecolumnint(@clubList)

IF NOT OBJECT_ID('tempdb.dbo.#SiteTable') IS NULL DROP TABLE #SiteTable
--Get Site List
CREATE TABLE #SiteTable(SiteId INT)
IF @sitelist = 'all' BEGIN
	INSERT INTO #sitetable (siteid)
	SELECT
		siteid
	FROM [dbo].[t_Site] WITH (NOLOCK)
	WHERE (sitetypeid = 720)
--AND (sitestatusid = 730)
END ELSE IF @SITELIST = 'NOTCANCUN' BEGIN
	INSERT INTO #sitetable (siteid)
	SELECT
		siteid
	FROM [dbo].[t_site] WITH (NOLOCK)
	WHERE (sitetypeid = 720)
```

```
	--AND (sitestatusid = 730)
	AND siteid NOT IN (1, 2, 30)
END ELSE BEGIN
	INSERT INTO #sitetable (siteid)
	SELECT
		ilist
	FROM [fnonecolumnint](@SiteList)
	ORDER BY ilist
END

IF NOT OBJECT_ID('tempdb.dbo.#CompaniaContableTable') IS NULL DROP TABLE
#CompaniaContableTable
--Get CompaniaContableTable  List
CREATE TABLE #CompaniaContableTable(FundingPoolID INT)
INSERT INTO #companiacontabletable (FundingPoolID)
	SELECT
		ilist
	FROM [fnonecolumnint](@CompaniaContable)

INSERT INTO #companiacontabletable (FundingPoolID)
	SELECT
		CL.CLookupID
	FROM dbo.CLookup CL WITH (NOLOCK)
	INNER JOIN (SELECT
			*
		FROM dbo.CLookup c WITH (NOLOCK)
		WHERE c.CLookupID IN (SELECT
				FundingPoolID
			FROM #companiacontabletable)) CL2
		ON CL2.Description = CL.Description
		AND CL2.CLookupParentID <> CL.CLookupParentID
	WHERE CL.CLookupParentID IN (97, 29)

-------------------------------------
IF NOT OBJECT_ID('tempdb.dbo.#LenderPagoTable') IS NULL DROP TABLE #LenderPagoTable
CREATE TABLE #LenderPagoTable([LenderPagoid] INT)
IF @LenderPago = 'ALLLEN' BEGIN
	INSERT INTO #lenderpagotable ([lenderpagoid])
	SELECT
		clookupid
	FROM [dbo].[clookup] WITH (NOLOCK)
	WHERE (clookupparentid = 42)
END ELSE IF @LenderPago = 'NOTLENDER' BEGIN
	INSERT INTO #lenderpagotable ([lenderpagoid])
	SELECT
		clookupid
	FROM [dbo].[clookup] WITH (NOLOCK)
	WHERE (clookupparentid = 42)
	AND clookupid NOT IN (1318, 1321, 1348, 1374, 1379, 1393, 1401)
END ELSE BEGIN
	INSERT INTO #lenderpagotable ([lenderpagoid])
	SELECT
		ilist
	FROM [fnonecolumnint](@LenderPago)
END

INSERT INTO #LenderPagoTable ([lenderpagoid])
	SELECT
```

```
		CL2.CLookupID
	FROM #lenderpagotable L
	INNER JOIN dbo.CLookup CL WITH (NOLOCK)
		ON CL.CLookupID = L.[LenderPagoid]
	INNER JOIN dbo.CLookup CL2 WITH (NOLOCK)
		ON CL.LookupName = CL2.LookupName
		AND CL2.CLookupParentID = 100


----Get Currecy  List
IF NOT OBJECT_ID('tempdb.dbo.#CurrencyTable') IS NULL DROP TABLE #CurrencyTable
CREATE TABLE #CurrencyTable(CurrencyID INT)
INSERT INTO #currencytable (currencyid)
	SELECT
		ilist
	FROM [fnonecolumnint](@Currency)


----Get STATUS  List
IF NOT OBJECT_ID('tempdb.dbo.#StatusTable') IS NULL DROP TABLE #StatusTable
CREATE TABLE #StatusTable(MortgageStatusID INT)
INSERT INTO #statustable (mortgagestatusid)
	SELECT
		ilist
	FROM [fnonecolumnint](@MortgageStatusList)

IF NOT OBJECT_ID('tempdb.dbo.#NotesBalance') IS NULL DROP TABLE #notesbalance
SELECT
	m.mortgageid,
	m.mortgagenumber,
	c.contractdate,
	c.siteid,
	--cl.reportdate AS 'TransferDate',
	m.OriginationDate 'TransferDate',
	ISNULL(m.originationdate, c.ContractDate) AS originationdate,
	m.TitleTypeID AS 'CurrentLenderID',
	m.titletype AS 'CurrentLender',
	ISNULL(clo.CLookupID, m.TitleTypeID) AS 'NotesBalanceLenderID',
	ISNULL(clo.lookupname, m.titletype) AS 'NotesBalanceLender',
	m.mortgagestatus,
	m.totalpayments,
	m.nextpaymentduedate,
	cu.currencyid,
	cu.currencyname,
	m.amountfinanced,
	m.exchangerate,
	CASE
		WHEN li.creditvendorluid IN (SELECT
				[lenderpagoid]
			FROM #lenderpagotable) THEN CASE
			WHEN ISNULL(mt2.totalmp, 0) > 0 THEN (SELECT TOP 1
					mp.currentprincipalbal
				FROM [dbo].[t_MortgagePayment] mp WITH (NOLOCK)
				WHERE mp.mortgageid = m.mortgageid
				AND mp.transactiondate <= @LOCAL_FechaInicial
				AND mp.transactioncodeid NOT IN (653) --Seguro de
Vida,Seguro por Fallecimiento
				ORDER BY mp.mortgagepaymentid DESC)
```

```
                        ELSE CASE
                                    WHEN CAST(c.ContractDate AS DATE) <= '20131130'
THEN m.amountfinanced
                                    ELSE m.amountfinanced + m.UserAmount6
                                END
                    END
                ELSE 0
        END
        AS 'BeginBalance',
        coi.iva,
        SUM(ISNULL(mt.principlepaid, 0)) [principlepaid],
        CONVERT(DECIMAL(10, 2), (SUM(ISNULL(mt.principlepaid, 0))) / (coi.iva)) AS
'PrincipalPaid_S_IVA',
        CONVERT(DECIMAL(10, 2), (SUM(ISNULL(mt.principlepaid, 0))) -
((SUM(ISNULL(mt.principlepaid, 0))) / (coi.iva))) AS 'PrincipalPaid_IVA',
        SUM(ISNULL(mt.principleadjustment4 + mt.principleadjustment6, 0))
[principleadjustment],
        CONVERT(DECIMAL(10, 2), (SUM(ISNULL(mt.principleadjustment4 +
mt.principleadjustment6, 0))) / (coi.iva)) AS 'PrincipleAdjustment_S_IVA',
        CONVERT(DECIMAL(10, 2), (SUM(ISNULL(mt.principleadjustment4 +
mt.principleadjustment6, 0))) - ((SUM(ISNULL(mt.principleadjustment4 +
mt.principleadjustment6, 0))) / (coi.iva))) AS 'PrincipleAdjustment_IVA',

        CASE
                WHEN ISNULL(li.creditvendorluid, 0) NOT IN (SELECT
                                [lenderpagoid]
                        FROM #lenderpagotable) THEN CASE
                        WHEN ISNULL(mt2.totalmp, 0) > 0 THEN (SELECT TOP 1
                                                mp.currentprincipalbal
                                        FROM [dbo].[t_MortgagePayment] mp WITH (NOLOCK)
                                        WHERE mp.mortgageid = m.mortgageid
                                        AND mp.transactiondate <= cl.reportdate
                                        AND mp.transactioncodeid NOT IN (653) --Seguro de
Vida,Seguro por Fallecimiento
                                        ORDER BY mp.mortgagepaymentid DESC)
                        ELSE m.amountfinanced + m.UserAmount6
                    END
                ELSE CASE
                        WHEN CAST(FLOOR(CAST(m.originationdate AS FLOAT)) AS DATETIME)
BETWEEN DATEADD(D, 1, @LOCAL_FechaInicial) AND @LOCAL_FechaFinal THEN CASE
                                    WHEN ISNULL(mt2.totalmp, 0) > 0 THEN (SELECT TOP
1
                                                           mp.currentprincipalbal
                                                    FROM [dbo].[t_MortgagePayment] mp
WITH (NOLOCK)
                                                    WHERE mp.mortgageid = m.mortgageid
                                                    AND mp.transactiondate <=
cl.reportdate
                                                    AND mp.transactioncodeid NOT IN
(653) --Seguro de Vida,Seguro por Fallecimiento
                                                    ORDER BY mp.mortgagepaymentid DESC)
                                    WHEN (SELECT
                                                           COUNT(*) totalmp
                                                    FROM [dbo].[t_MortgagePayment] a
WITH (NOLOCK)
                                                    WHERE a.transactiondate BETWEEN
DATEADD(D, 1, @LOCAL_FechaInicial) AND DATEADD(D, -1, cl.ReportDate)
                                                    AND a.MortgageID = m.MortgageID
```

```
                                                    AND a.transactioncodeid NOT IN
(653) --Seguro de Vida,Seguro por Fallecimiento
                                                )
                                                > 0 THEN (SELECT TOP 1
                                                        mp.currentprincipalbal
                                                    FROM [dbo].[t_MortgagePayment] mp
WITH (NOLOCK)
                                                    WHERE mp.mortgageid = m.mortgageid
                                                    AND mp.transactiondate <=
cl.reportdate
                                                    AND mp.transactioncodeid NOT IN
(653) --Seguro de Vida,Seguro por Fallecimiento
                                                    ORDER BY mp.mortgagepaymentid DESC)
                                        ELSE m.amountfinanced
                                    END
                                ELSE CASE
                                        WHEN cs1.CreditVendorLUID NOT IN (SELECT
                                                [lenderpagoid]
                                            FROM #lenderpagotable) THEN CASE
                                            WHEN ISNULL(mt2.totalmp, 0) > 0
THEN (SELECT TOP 1

        mp.currentprincipalbal * (-1)
                                                                FROM
[dbo].[t_MortgagePayment] mp WITH (NOLOCK)
                                                                WHERE mp.mortgageid =
m.mortgageid
                                                                AND mp.transactiondate
<= cl.reportdate
                                                                AND
mp.transactioncodeid NOT IN (653) --Seguro de Vida,Seguro por Fallecimiento
                                                                ORDER BY
mp.mortgagepaymentid DESC)
                                                    ELSE 0
                                                END
                                        ELSE 0
                                    END
                        END

        END
        AS 'PrincipalTransfered',
        --Se calcula abajo ---m.amountfinanced - ISNULL(tb.endbalance, 0) AS 'EndBalance',
        SUM(ISNULL(mt.interest, 0)) [interest],
        SUM(ISNULL(mt.interestadjustment, 0)) [interestadjustment],
        SUM(ISNULL(mt.latefee, 0)) [latefee],
        SUM(ISNULL(mt2.LateFeesBal, 0)) * (-1) AS [LateFeesBal],
        SUM(ISNULL(mt.latefeeadjustment, 0)) [latefeeadjustment],
        SUM(ISNULL(mt.tomemo, 0)) 'ToMemo',
        SUM(ISNULL(mt.frommemo, 0)) 'FromMemo',
        SUM(ISNULL(mt.partialpayment, 0)) [partialpayment],
        SUM(ISNULL(mt.partialpaymentadjustment, 0)) [partialpaymentadjustment],
        SUM(ISNULL(mt.processingfee, 0)) [processingfee],
        ISNULL(SUM(ISNULL(mt.principlepaid, 0)) + SUM(ISNULL(mt.interest, 0)) +
SUM(ISNULL(mt.interestadjustment, 0)) + SUM(ISNULL(mt.tomemo, 0)) +
SUM(ISNULL(mt.frommemo, 0)) + SUM(ISNULL(mt.latefee, 0)) +
SUM(ISNULL(mt.latefeeadjustment, 0)), 0) [totalpaid],

        clu.clubname,
```

```
	cpt.clubid,
	ISNULL(cs1.creditvendorluid, m.TitleTypeID) AS 'LenderPagoID',
	c.user16luid AS 'CompaniaContableId',
	cc.lookupname AS 'CompaniaContable',
	ISNULL(li.CreditVendorLUID, m.TitleTypeID) AS CreditVendorLUID,
	ISNULL(li.lenderinicial, m.TitleType) AS lenderinicial,
	cs1.CreditVendorLUID 'CiaContableRCC'

	INTO #notesbalance
FROM [dbo].[t_mortgage] AS m WITH (NOLOCK)
INNER JOIN [dbo].[t_Contract] AS c WITH (NOLOCK)
	ON m.contractid = c.contractid
INNER JOIN [dbo].[t_Site] AS s WITH (NOLOCK)
	ON c.siteid = s.siteid
INNER JOIN [dbo].[t_Owner] AS o WITH (NOLOCK)
	ON c.ownerid = o.ownerid
INNER JOIN [dbo].[Currency] AS cu WITH (NOLOCK)
	ON m.currencyid = cu.currencyid
INNER JOIN [dbo].[CLookup] cc WITH (NOLOCK)
	ON cc.clookupid = c.user16luid
LEFT JOIN [dbo].[cat_office_iva] AS coi WITH (NOLOCK)
	ON coi.offnum = s.siteid
	AND coi.anio = YEAR(c.contractdate)
OUTER APPLY (SELECT TOP 1
		si1.ClubPkgTypeID
	FROM t_soldinventory si1 WITH (NOLOCK)
	WHERE si1.contractid = c.contractid) si
LEFT JOIN [dbo].[ClubPkgType] AS cpt WITH (NOLOCK)
	ON cpt.clubpkgtypeid = si.clubpkgtypeid
LEFT JOIN [dbo].[Club] clu WITH (NOLOCK)
	ON clu.clubid = cpt.clubid
-------------------------------------------------------------------------------------------
--------
/*Cambio de la vista vw_mortgagepaymenttransactions_2 para cierre de marzo14*/
LEFT JOIN [vw_mortgagepaymenttransactions_2] mt WITH (NOLOCK)
	ON mt.mortgageid = m.mortgageid
	AND mt.transactiondate BETWEEN DATEADD(D, 1, @LOCAL_FechaInicial) AND
@LOCAL_FechaFinal
	AND mt.lendertrxid IN (SELECT
			[lenderpagoid]
		FROM #lenderpagotable)
	AND mt.transactioncodeid NOT IN (653) --Seguro de Vida,Seguro por Fallecimiento
-------------------------------------------------------------------------------------------
--------
LEFT JOIN (SELECT
		a.mortgageid,
		COUNT(a.MortgagePaymentID) totalmp,
		SUM(ISNULL([a].[ToLateFees], 0)) AS [LateFeesBal]
	FROM [dbo].[t_MortgagePayment] a WITH (NOLOCK)
	WHERE a.transactiondate <= @LOCAL_FechaInicial
	AND a.transactioncodeid NOT IN (653) --Seguro de Vida,Seguro por Fallecimiento
	GROUP BY a.mortgageid) mt2
	ON mt2.mortgageid = m.mortgageid
LEFT JOIN (SELECT
		mortgageid,
		SUM(CASE
			WHEN transactiondate <= @LOCAL_FechaInicial THEN toprinciple
			ELSE 0
```

```
              END) AS beginbalance,
              SUM(CASE
                    WHEN transactiondate <= @LOCAL_FechaFinal THEN toprinciple
                    ELSE 0
              END) AS endbalance
       FROM [dbo].[t_MortgagePayment] WITH (NOLOCK)
       GROUP BY mortgageid) tb
       ON tb.mortgageid = m.mortgageid
OUTER APPLY (SELECT TOP 1
              *
       FROM CreditScore cs WITH (NOLOCK)
       WHERE cs.MortgageID = M.MortgageID
       AND CAST(FLOOR(CAST(cs.ReportDate AS FLOAT)) AS DATETIME) <= @LOCAL_FechaFinal
       ORDER BY cs.CreditScoreID DESC) cl
--Lender Pago
OUTER APPLY (SELECT TOP 1
              *
       FROM CreditScore csa WITH (NOLOCK)
       WHERE csa.MortgageID = M.MortgageID
       AND CAST(FLOOR(CAST(csa.ReportDate AS FLOAT)) AS DATETIME) <= @LOCAL_FechaFinal
       ORDER BY csa.CreditScoreID DESC) cs1
LEFT JOIN [dbo].[CLookup] clo
       ON cs1.creditvendorluid = clo.clookupid

--Lender inicial
OUTER APPLY (SELECT TOP 1
              csb.*,
              c.lookupname 'LenderInicial'
       FROM CreditScore csb WITH (NOLOCK)
       JOIN [dbo].[CLookup] c WITH (NOLOCK)
              ON csb.creditvendorluid = c.clookupid
       WHERE csb.MortgageID = M.MortgageID
       AND CAST(FLOOR(CAST(csb.ReportDate AS FLOAT)) AS DATETIME) <= @LOCAL_FechaInicial
       ORDER BY csb.CreditScoreID DESC) li


WHERE (ISNULL(m.amountfinanced, 0) + ISNULL(m.CurrentPrincipleBal, 0)) > 0
AND CAST(ISNULL(ISNULL(m.OriginationDate, m.DateFunded), c.ContractDate) AS DATE) <=
@LOCAL_FechaFinal --~NALR~ 2015Agp04 Acuerdo verbal ecastrillo - mgarcial
AND c.siteid IN (SELECT
              siteid
       FROM #sitetable)
AND m.currencyid IN (SELECT
              CurrencyID
       FROM #currencytable)
AND ISNULL(clo.CLookupID, m.TitleTypeID) IN (SELECT
              lenderpagoid
       FROM #lenderpagotable)
AND m.mortgagestatusid IN (SELECT
              mortgagestatusid
       FROM #statustable)
AND m.FundingPoolID IN (SELECT
              FundingPoolID
       FROM #CompaniaContableTable)

AND c.ContractStatusID NOT IN (1, 2, 3, 4) ---Pender,Suspense,Kick,Rescission no se
consideran para NB 2015/Jul/14 ~NALR~ ~nalr~ 2015Dic02 Se regresa a asol de ECastrillo
GROUP BY      m.mortgageid,
```

```
                    m.mortgagenumber,
                    c.contractdate,
                    c.siteid,
                    mt.transferdate,
                    m.TitleTypeID,
                    m.titletype,
                    clo.CLookupID,
                    clo.lookupname,
                    cl.reportdate,
                    m.originationdate,
                    m.mortgagestatus,
                    m.totalpayments,
                    m.nextpaymentduedate,
                    tb.beginbalance,
                    tb.endbalance,
                    m.amountfinanced,
                    cu.currencyid,
                    cu.currencyname,
                    clo.lookupname,
                    cpt.clubid,
                    clu.clubname,
                    cc.lookupname,
                    cs1.creditvendorluid,
                    li.creditvendorluid,
                    coi.iva,
                    c.user16luid,
                    li.lenderinicial,
                    mt.contractid,
                    m.currentprinciplebal,
                    m.FundingPoolID,
                    mt2.totalmp,
                    m.ExchangeRate,
                    m.UserAmount6

OPTION (MAXDOP 3);
-------------------------------------------------------------------------------------------

SELECT *,ROW_NUMBER() OVER(ORDER BY mortgageid DESC) as RowNum into #notesbalance1  FROM
#notesbalance n0
where
((n0.lenderpagoid IN (SELECT
             [lenderpagoid]
       FROM #lenderpagotable)
)
OR (n0.creditvendorluid IN (SELECT
             [lenderpagoid]
       FROM #lenderpagotable)))



SELECT
       n.mortgageid,
       n.mortgagenumber,
       n.contractdate,
       n.siteid,
       n.TransferDate,
       n.originationdate,
       n.CurrentLenderId,
```

```
	n.CurrentLender,
	n.notesbalancelender,
	n.mortgagestatus,
	n.totalpayments,
	n.nextpaymentduedate,
	n.currencyid,
	n.currencyname,
	n.amountfinanced,
	n.exchangerate,
	n.BeginBalance,
	n.principlepaid,
	n.PrincipalPaid_S_IVA,
	n.PrincipalPaid_IVA,
	n.principleadjustment,
	n.PrincipleAdjustment_S_IVA,
	PrincipleAdjustment_IVA,
	PrincipalTransfered,
	interest + interestadjustment 'interest',
	0 'interestadjustment',
	latefee 'latefee',
	latefeeadjustment,
	[LateFeesBal],
	ToMemo,
	ISNULL(FromMemo, 0) + ISNULL(mdev.Devolucion, 0) 'FromMemo',
	partialpayment,
	partialpaymentadjustment,
	processingfee,
	ISNULL(totalpaid, 0) + ISNULL(mDev.Devolucion, 0) 'totalpaid',
	clubname,
	clubid,
	LenderPagoID,
	CompaniaContableId,
	CompaniaContable,
	CreditVendorLUID,
	lenderinicial,
	iva,
	CiaContableRCC,
	((beginbalance * iva) /*-m.AmountFinanced*/) AS 'Imp_S_Iva',
	((((beginbalance * iva) - beginbalance) - beginbalance) * -1) AS 'Imp_Del_Iva',
	(((((beginbalance * iva) - beginbalance) - beginbalance) * -1) * 0.1) AS
'CTA_Incobrable',
	--Original beginbalance + principaltransfered + principleadjustment - principlepaid
AS 'EndBalance'

	/* --TotalPaid con el primer cambio del 26-03-2014*/
	beginbalance + principaltransfered - principleadjustment - principlepaid AS
'EndBalance',

	/* --EndBalance Iva con el cambio de minuta 27-03-2014 */
	CONVERT(DECIMAL(10, 2), (ISNULL(beginbalance + principaltransfered -
principleadjustment - principlepaid, 0))) / (n.iva) AS 'EndBalance_S_IVA',
	CONVERT(DECIMAL(10, 2), (ISNULL(beginbalance + principaltransfered -
principleadjustment - principlepaid, 0))) - ((ISNULL(beginbalance + principaltransfered -
principleadjustment - principlepaid, 0)) / (n.iva)) AS 'EndBalance_IVA',
	tmbal.ToMemoBal,
	ISNULL(tmbal.FromMemoBal, 0) + (ISNULL(md.MemoDev, 0) * -1) 'FromMemoBal',
	ISNULL(mbal.MemoBalance, 0) * (-1) 'MemoBalance',
	(ISNULL(ma.MemoAmount, 0) * -1) 'MemoTrx'
```

```
------------------------------
       ,RowNum
INTO ##TBLPASOREPORTNB
FROM #notesbalance1 n
JOIN #clublist cl
      ON cl.clublistid = n.clubid
--Se añaden campos por peticion de contabilidad 12/06/2014
OUTER APPLY (SELECT
             SUM(mm1.amount) 'MemoBalance'
      FROM t_MortgageMemo mm1 WITH (NOLOCK)
      WHERE mm1.mortgageid = n.mortgageid
      AND CAST(FLOOR(CAST(mm1.transactiondate AS FLOAT)) AS DATETIME) <=
@LOCAL_FechaFinal) mbal
OUTER APPLY (SELECT
             SUM(mp1.servicefee) 'ToMemoBal',
             SUM(mp1.frommemo) * (-1) 'FromMemoBal'
      FROM t_MortgagePayment mp1 WITH (NOLOCK)
      WHERE mp1.mortgageid = n.mortgageid
      AND CAST(FLOOR(CAST(mp1.transactiondate AS FLOAT)) AS DATETIME) <=
@LOCAL_FechaFinal
      AND mp1.voideddate IS NULL) tmbal
OUTER APPLY (SELECT
             SUM(ISNULL(mm2.amount, 0)) 'MemoAmount'
      FROM t_MortgageMemo mm2 WITH (NOLOCK)
      WHERE mm2.mortgageid = n.mortgageid
      AND CAST(FLOOR(CAST(mm2.transactiondate AS FLOAT)) AS DATETIME) BETWEEN DATEADD(D,
1, @LOCAL_FechaInicial) AND @LOCAL_FechaFinal
      AND mm2.TransactionCodeID IN (544, 597, 656, 687) --->>Cxl Memo,To Credit,Last
Payment,Trans Partial-Princ)
) ma
OUTER APPLY (SELECT
             SUM(ISNULL(mm3.amount, 0)) 'MemoDev'
      FROM t_MortgageMemo mm3 WITH (NOLOCK)
      WHERE mm3.mortgageid = n.mortgageid
      AND CAST(FLOOR(CAST(mm3.transactiondate AS FLOAT)) AS DATETIME) <=
@LOCAL_FechaFinal
      AND mm3.TransactionCodeID IN (392, 544, 597, 656, 687) --->>Devoluciones,Cxl
Memo,To Credit,Last Payment,Trans Partial-Princ)
) md
OUTER APPLY (SELECT
             SUM(ISNULL(mm4.amount, 0) * (-1)) 'Devolucion'
      FROM t_MortgageMemo mm4 WITH (NOLOCK)
      WHERE mm4.mortgageid = n.mortgageid
      AND CAST(FLOOR(CAST(mm4.transactiondate AS FLOAT)) AS DATETIME) BETWEEN DATEADD(D,
1, @LOCAL_FechaInicial) AND @LOCAL_FechaFinal
      AND mm4.TransactionCodeID IN (392) --->>Devoluciones
) mDev
WHERE
RowNum BETWEEN ((((@pagina-1)*@regxpag)+1) and ((((@pagina)*@regxpag))


ORDER BY N.MortgageNumber

      -- Se declaran variables con el cual se va a llenar de información la tabla.
      DECLARE @query NVARCHAR(MAX),
            @anio NVARCHAR(4),
            @mes NVARCHAR(2)
```

```
	-- Se guardan mes y año para especificar el nombre de la tabla en la que se
llenarán los datos
	SET @anio = CAST(YEAR(@FechaFinal) AS NVARCHAR)

	IF MONTH(@FechaFinal) < 10
	BEGIN
		SET @mes = '0' + CAST(MONTH(@FechaFinal) AS NVARCHAR)
	END
	ELSE
	BEGIN
		SET @mes = CAST(MONTH(@FechaFinal) AS NVARCHAR)
	END

	-- Se llena la tabla con la nueva información.
	SET @query = 'delete from [Report].[t_MortgageNBReport' + @anio + @mes + ']'

	EXEC SP_EXECUTESQL @query

	SET @query = 'insert into [Report].[t_MortgageNBReport' + @anio + @mes + ']
(mortgageNumber,
	Contractdate,
	SiteId,
	TransferDate,
	Originationdate,
	CurrentLenderId,
	Currentlender,
	NotesBalanceLender,
	MortgageStatus,
	Totalpayments,
	Nextpaymentduedate,
	CurrencyId,
	CurrencyName,
	AmountFinanced,
	Exchangerate,
	Beginbalance,
	Principlepaid,
	Principalpaid_S_IVA,
	Principalpaid_IVA,
	Principleadjustment,
	Principleadjustment_S_IVA,
	Principleadjustment_IVA,
	Principaltransfered,
	Interest,
	InterestAdjustment,
	Latefee,
	Latefeeadjustment,
	Latefeesbal,
	Tomemo,
	Frommemo,
	PartialPayment,
	PartialPaymentAdjustment,
	ProcessIngFee,
	Totalpaid,
	ClubName,
	ClubId,
	LenderPagoId,
	CompaniaContableId,
	CompaniaContable,
```

```
CreditVendorLUID,
LenderInicial,
IVA,
CiaContableRCC,
Imp_S_IVA,
Imp_Del_IVA,
CTA_Incobrable,
Endbalance,
Endbalance_S_IVA,
Endbalance_IVA,
Tomemobal,
Frommemobal,
Memobalance,
Memotrx,
RowNum,
InsertDate)
select mortgageNumber,
Contractdate,
SiteId,
TransferDate,
Originationdate,
CurrentLenderId,
Currentlender,
NotesBalanceLender,
MortgageStatus,
Totalpayments,
Nextpaymentduedate,
CurrencyId,
CurrencyName,
AmountFinanced,
Exchangerate,
Beginbalance,
Principlepaid,
Principalpaid_S_IVA,
Principalpaid_IVA,
Principleadjustment,
Principleadjustment_S_IVA,
Principleadjustment_IVA,
Principaltransfered,
Interest,
InterestAdjustment,
Latefee,
Latefeeadjustment,
Latefeesbal,
Tomemo,
Frommemo,
PartialPayment,
PartialPaymentAdjustment,
ProcessIngFee,
Totalpaid,
ClubName,
ClubId,
LenderPagoId,
CompaniaContableId,
CompaniaContable,
CreditVendorLUID,
LenderInicial,
IVA,
```

```
CiaContableRCC,
Imp_S_IVA,
Imp_Del_IVA,
CTA_Incobrable,
Endbalance,
Endbalance_S_IVA,
Endbalance_IVA,
Tomemobal,
Frommemobal,
Memobalance,
Memotrx,
RowNum,
GETDATE()
 from ##TBLPASOREPORTNB'

 EXEC SP_EXECUTESQL @query
```

Anexo 5: [Report].[sp_TrxCartera_Prm_IVM]

```
ALTER PROCEDURE  [Report].[sp_TrxCartera_Prm_IVM]


	@CompaniaContable varchar(max) = null,
	@ClubList varchar(max)=null,
	@LenderActual varchar (max) = null,
	@FundingInstitutionList VARCHAR(MAX) = null,
	@FundingStatusList VARCHAR(MAX)   = null,
	@Country varchar (max) = null,
	@SiteList varchar (max) = null,
	@LenderPago varchar(max) = null,
	@Year int = null,
	@Month int = null

	,@pagina integer = null
	,@regxpag integer = null

AS
BEGIN

DECLARE @LOCAL_Year INT,
		@LOCAL_Month INT

if @pagina is null
	set @pagina = 1
if @regxpag is NULL
	set @regxpag = 100000

if @Year is null
	set @Year = year(getdate());

if @Month is null
	set @Month = month(getdate());


SELECT @LOCAL_Month=@Month ,
		@LOCAL_Year= @Year

IF OBJECT_ID('tempdb..##TBLPASOTRAXREPORT') IS NOT NULL
    DROP TABLE ##TBLPASOTRAXREPORT

--Get CompaniaContable List
DECLARE @CompaniaContabletable TABLE(CodCompaniaContable INT)
--Get LenderActual List
DECLARE @LenderActualtable TABLE([CurrentLenderid] INT)
--Get ClubList List
DECLARE @ClubListTable TABLE(ClubListid INT)
--Get FundingInstitution List
DECLARE @FundingInstitutionTable TABLE(FundingInstitutionID INT)
--Get FundingStatus List
DECLARE @FundingStatusTable TABLE(FundingstatusID INT)
--Get site List
DECLARE @SiteTable TABLE(siteid INT)
----Get Country  List
declare @CountryTable table (Countryid INT)
--Get LenderPago List
```

```
DECLARE @LenderPagotable TABLE([LenderPagoid] INT)


if @CompaniaContable is null
	INSERT INTO @CompaniaContabletable (CodCompaniaContable) SELECT CLookupID FROM
CLookup AS comp WHERE (CLookupParentID = 97)
ELSE
	INSERT INTO @CompaniaContabletable (CodCompaniaContable) SELECT iList FROM
TSW_LIVE_ClientCustom.Report.fnOneColumnint(@CompaniaContable)

if @ClubList is null
	INSERT INTO @ClubListTable (ClubListid) SELECT ClubID FROM Club
else
	INSERT INTO @ClubListTable (ClubListid) SELECT iList FROM
TSW_LIVE_ClientCustom.Report.fnOneColumnint(@ClubList)

IF @LenderActual is null
	INSERT INTO @LenderActualtable ([CurrentLenderid]) SELECT CLookupID FROM CLookup
WHERE (CLookupParentID = 42)
ELSE
	INSERT INTO @LenderActualtable ([CurrentLenderid]) SELECT iList FROM
TSW_LIVE_ClientCustom.Report.fnOneColumnint(@LenderActual)

Insert into @LenderActualtable
	SELECT CL.CLookupID FROM dbo.CLookup CL
		INNER JOIN (SELECT CLookupID, CLookupParentID, LookupName
		FROM dbo.CLookup WHERE CLookupID IN (SELECT [CurrentLenderid] from
@LenderActualtable)) CL1
	ON CL1.LookupName = CL.LookupName AND CL1.CLookupParentID <>cl.CLookupParentID

IF @FundingInstitutionList is null
	INSERT INTO @FundingInstitutionTable (FundingInstitutionID)
	SELECT FundingInstitutionID FROM t_FundingInstitututionGROUP BY FundingInstitutionID, FundingInstitution ORDER BY FundingInstitution
ELSE
	INSERT INTO @FundingInstitutionTable (FundingInstitutionID)
	SELECT iList
	FROM TSW_LIVE_ClientCustom.Report.fnOneColumnint(@FundingInstitutionList)


IF @FundingStatusList is NULL
	INSERT INTO @FundingStatusTable (FundingstatusID) SELECT CLookupID FROM CLookup AS
c WHERE (CLookupParentID = 30)
ELSE
	INSERT INTO @FundingStatusTable (FundingstatusID) SELECT iList FROM
TSW_LIVE_ClientCustom.Report.fnOneColumnint(@FundingStatusList)

IF @Country IS NULL
	INSERT INTO @CountryTable (Countryid)
	SELECT DISTINCT o.CountryID FROM t_Owner AS o INNER JOIN cCountry AS co ON
co.CountryID = o.CountryID

ELSE
	INSERT INTO @CountryTable (Countryid) SELECT ilist FROM
[tsw_live_clientcustom].[report].[fnonecolumnint](@Country)

IF @SiteList IS NULL
	INSERT INTO @SiteTable (siteid)  SELECT SiteID FROM  t_Site
```

```
WHERE        (SiteTypeID in (720,719))
ELSE
       INSERT INTO @SiteTable (siteid) SELECT   iList FROM
TSW_LIVE_ClientCustom.Report.fnOneColumnint(@SiteList)



IF @LenderPago IS NULL
       INSERT INTO @LenderPagotable ([LenderPagoid]) SELECT CLookupID FROM CLookup WHERE
(CLookupParentID = 42)
ELSE
       INSERT INTO @LenderPagotable ([LenderPagoid]) SELECT  iList FROM
TSW_LIVE_ClientCustom.Report.fnOneColumnint(@LenderPago)

Insert into @LenderPagotable
              SELECT CL.CLookupID
              FROM dbo.CLookup CL
              INNER JOIN (SELECT
                                          CLookupID
                                        , CLookupParentID
                                        , LookupName
                                 FROM dbo.CLookup
                                 WHERE CLookupID IN (SELECT [LenderPagoid] from
@LenderPagotable)) CL1
                     ON CL1.LookupName = CL.LookupName AND CL1.CLookupParentID
<>cl.CLookupParentID


IF NOT OBJECT_ID ('tempdb.dbo.#TrxCar') IS NULL
DROP TABLE #TrxCar

SET NOCOUNT ON;

SELECT *
INTO #TrxCar
FROM (
                     SELECT  [ContractID]
                                          ,[mortgageid]
                                          ,[LastName]
                                          ,[Firstname]
                                          ,[ContractNumber]
                                          ,[Mes]
                                          ,[Anio]
                                          ,[TransactionDate]
                                          ,[postedDate]
                                          ,[SiteName]
                                          ,VTC.[SiteId]
                                          ,[MortgageNumber]
                                          ,[SaleDate]
                                          ,[CloseDate]
                                          ,[TipoContrato]
                                          ,[PkgType]
                                          ,[Points]
                                          ,[ContractStatus]
                                          ,[StatusCarteraid]
                                          ,[StatusCartera]
                                          ,VTC.[CountryID]
                                          ,[Country]
```

```
,[Nacionalidad]
,[ContractCurrency]
,[Compania]
,[CompaniaDesc]
,VTC.[CodCompaniaContable]
,[LenderTrxid]
,VTC.[CurrentLenderid]
,[CurrentLender]
,[MortgageCurrencyID]
,[MortgageCurrency]
,[Numpagoshechos]
,[PaymentAmount]
,[APR]
,[BankID]
,[ForAmount]
,[BkDesc]
,[UserName]
,[CodTransaccion]
,[TransactionCode]
,[NextPaymentDueDate]
,[IVA]
,[TransactionCodeID]
,[Amount]
,[BaseAmountTrx]
,[AccAmount]
,[BaseAmountTrx_S_IVA]
,[ToCapital]
,[To_Capital_S_IVA]
,[ToInterest]
,[To_Interest_IVA]
,[To_Interest_S_IVA]
,[Imp_Transaccion_IVA]
,[To_Capital_IVA]
,[ToLateFees]
,[FromMemo]
,[ToMemo]
,[CurrentPrincipalBal]
,[InstrumentPayID]
,[InstrumentDescription]
,[SaleYear]
,[Reference]
,[AuthCode]
,[Morosidad_Tipo]
,[Morosidad_Status]
,[Morosidad_Tipo_Inicial]
,[Morosidad_Inicial]
,[ClubID]
,[ClubName]
,VTC.[FundingInstitutionID]
,[CiaCobranza]
,VTC.[FundingStatusID]
,[fundingstatus]
,[TrxCurrencyID]
,[TrxCurrency]
,[CiaContableMortgage]
,[LoanPortfolioid]
,[LoanPortfolio]
,[PortfolioComment]
```

```
                                        ,[AssignedDate]
                                        ,[PaymentsRemaining]
                                        ,[OwnerID]
                                        ,[GrupoCobranza]
                                        ,[user10]
                                        ,[CiaContableRCC]
                                        ,[ReferenceMBT]
                                        ,VTC.TrxDayExRate as [TrxExchangeRate]
                                        ,[BillingCodeID]
                                        ,[BillingCode]
                                        ,[PaymentType]
                                        ,[MortgagePaymentID]
                                        ,[Origen]
                                        ,[LenderTrx]
                                        , [VTC].[Site_Origen]
                                        , VTC.CSC_AccNum
                                        , VTC.CSCTransType
                        FROM
                                [Report].[vw_TrxCartera] AS VTC
                                inner JOIN @CompaniaContabletable CC
                                        ON CC.CodCompaniaContable =
isnull(VTC.CodCompaniaContable,0)
                                inner JOIN @LenderActualtable LA
                                        ON LA.[CurrentLenderid] =
isnull(VTC.[CurrentLenderid],0)
                                inner JOIN @ClubListTable CL
                                        ON CL.ClubListid = isnull(VTC.ClubID,0)
                                inner JOIN @FundingInstitutionTable FI
                                        ON FI.FundingInstitutionID =
isnull(VTC.FundingInstitutionID,0)
                                inner JOIN @FundingStatusTable FST
                                        ON FST.FundingstatusID = isnull(VTC.FundingstatusID,0)
                                --JOIN @LenderPagotable LP
                                --      ON LP.LenderPagoid = t.LenderTrxid
                                inner JOIN @SiteTable SL
                                        ON SL.Siteid = isnull(VTC.Siteid,0)
                                inner JOIN @CountryTable c
                                        ON c.Countryid = isnull(VTC.countryid,0)
                        WHERE VTC.ANIO = @LOCAL_Year
                        AND VTC.MES = @LOCAL_Month
                        UNION ALL

                        SELECT [ContractID]
                                ,[mortgageid]
                                ,[LastName]
                                ,[Firstname]
                                ,[ContractNumber]
                                ,[Mes]
                                ,[Anio]
                                ,[TransactionDate]
                                ,[postedDate]
                                ,[SiteName]
                                , [Tm].[SiteId]
                                ,[MortgageNumber]
                                ,[SaleDate]
                                ,[CloseDate]
                                ,[TipoContrato]
                                ,[PkgType]
```

```
,[Points]
,[ContractStatus]
,[StatusCarteraid]
,[StatusCartera]
, [Tm].[CountryID]
,[Country]
,[Nacionalidad]
,[ContractCurrency]
,[Compania]
,[CompaniaDesc]
, [Tm].[CodCompaniaContable]
,[LenderTrxid]
, [Tm].[CurrentLenderid]
,[CurrentLender]
,[MortgageCurrencyID]
,[MortgageCurrency]
,[Numpagoshechos]
,[PaymentAmount]
,[APR]
,[BankID]
,[ForAmount]
,[BkDesc]
,[UserName]
,[CodTransaccion]
,[TransactionCode]
,[NextPaymentDueDate]
,[IVA]
,[TransactionCodeID]
,[Amount]
,[BaseAmountTrx]
,[AccAmount]
,[BaseAmountTrx_S_IVA]
,[ToCapital]
,[To_Capital_S_IVA]
,[ToInterest]
,[To_Interest_IVA]
,[To_Interest_S_IVA]
,[Imp_Transaccion_IVA]
,[To_Capital_IVA]
,[ToLateFees]
,[FromMemo]
,[ToMemo]
,[CurrentPrincipalBal]
,[InstrumentPayID]
,[InstrumentDescription]
,[SaleYear]
,[Reference]
,[AuthCode]
,[Morosidad_Tipo]
,[Morosidad_Status]
,[Morosidad_Tipo_Inicial]
,[Morosidad_Inicial]
,[ClubID]
,[ClubName]
, [Tm].[FundingInstitutionID]
,[CiaCobranza]
, [Tm].[FundingStatusID]
,[fundingstatus]
```

```
                              ,[TrxCurrencyID]
                              ,[TrxCurrency]
                              ,[CiaContableMortgage]
                              ,[LoanPortfolioid]
                              ,[LoanPortfolio]
                              ,[PortfolioComment]
                              ,[AssignedDate]
                              ,[PaymentsRemaining]
                              ,[OwnerID]
                              ,[GrupoCobranza]
                              ,[user10]
                              ,[CiaContableRCC]
                              ,[ReferenceMBT]
                              , [Tm].[TrxExchangeRate] as [TrxExchangeRate]
                              ,[BillingCodeID]
                              ,[BillingCode]
                              ,[PaymentType]
                              , [Tm].[MortgageMemoID]
                              ,[Origen]
                              ,[LenderTrx]
                              , [Tm].[Site_Origen]
                              , [Tm].[CSC_AccNum]
                              , Tm.CSCTransType
                       FROM
                              [Report].vw_trxMortgageMemo AS Tm
                              inner JOIN @CompaniaContabletable CC
                                     ON CC.CodCompaniaContable =
isnull(Tm.CodCompaniaContable,0)
                              inner JOIN @LenderActualtable LA
                                     ON LA.[CurrentLenderid] = isnull(Tm.[CurrentLenderid],0)
                              inner JOIN @ClubListTable CL
                                     ON CL.ClubListid = isnull(Tm.ClubID,0)
                              inner JOIN @FundingInstitutionTable FI
                                     ON FI.FundingInstitutionID =
isnull(Tm.FundingInstitutionID,0)
                              inner JOIN @FundingStatusTable FST
                                     ON FST.FundingstatusID = isnull(Tm.FundingstatusID,0)
                              inner JOIN @SiteTable SL
                                     ON SL.Siteid = isnull(Tm.Siteid,0)
                              inner JOIN @CountryTable c
                                     ON c.Countryid = isnull(Tm.countryid,0)
                       WHERE tm.ANIO = @LOCAL_Year
                       AND tm.MES = @LOCAL_Month
) trx

-------------------------------------------------------------
SELECT *,ROW_NUMBER() OVER(ORDER BY mortgageid DESC) as RowNum into #TrxCar1  FROM #TrxCar
n0

-------------------------------------------------------------

       SELECT *
       INTO ##TBLPASOTRAXREPORT
       FROM #TrxCar1 t
                              WHERE
RowNum BETWEEN (((@pagina-1)*@regxpag)+1) and (((@pagina)*@regxpag))
```

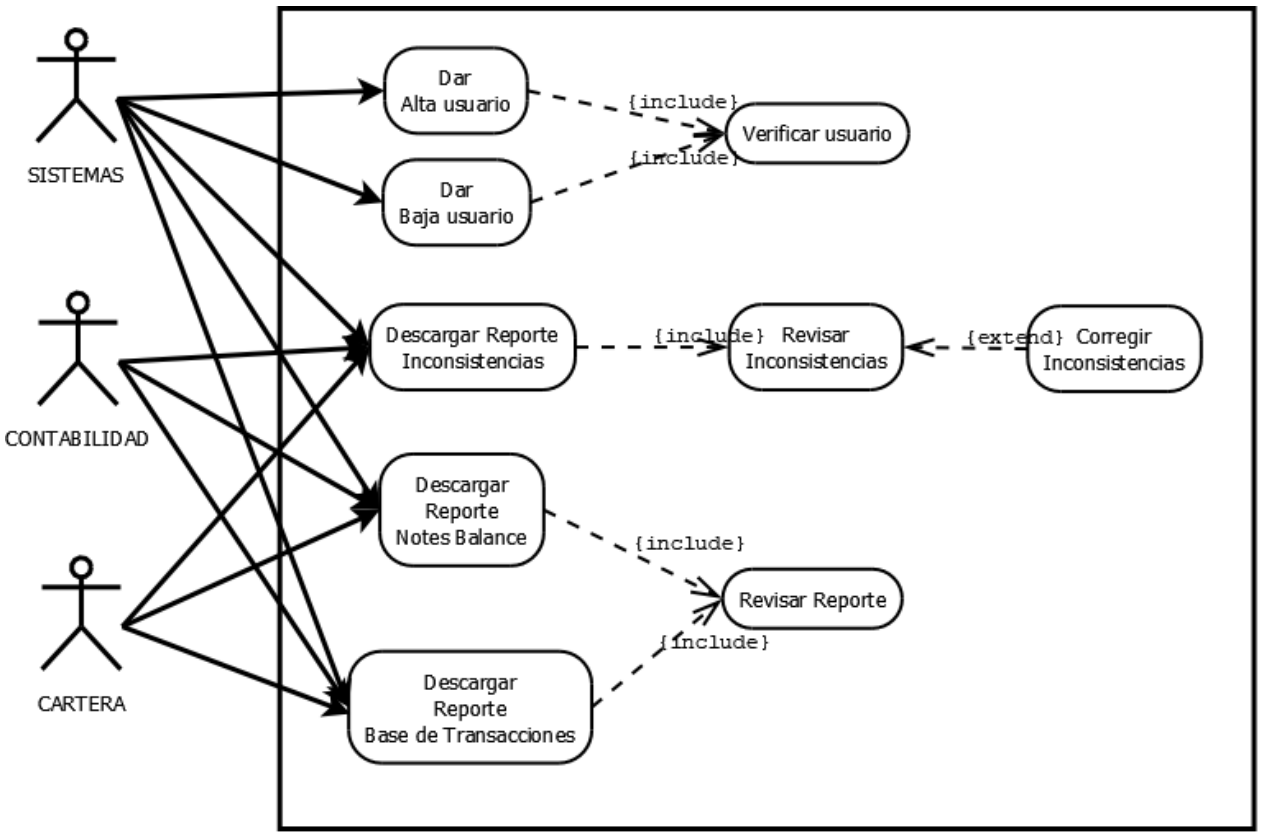
```
OPTION (MAXDOP 2);

DECLARE @Mes VARCHAR(2)

IF @Month < 10
BEGIN
	SET @Mes = '0' + CAST(@Month AS VARCHAR)
END
ELSE
BEGIN
	SET @Mes = CAST(@Month AS VARCHAR)
END

print 'eliminando datos con mes ' + @Mes

DELETE FROM [Report].[t_mortgageNBTrax]
WHERE Anio = @Year
	AND Mes = @Mes

print 'insertando datos nuevos'

INSERT INTO [Report].[t_mortgageNBTrax] (
	[ContractID],
	[mortgageid],
	[lastName],
	[Firstname],
	[ContractNumber],
	[Mes],
	[Anio],
	[TransactionDate],
	[postedDate],
	[SiteName],
	[SiteId],
	[MortgageNumber],
	[SaleDate],
	[CloseDate],
	[TipoContrato],
	[PkgType],
	[Points],
	[ContracStatus],
	[StatusCarteraid],
	[StatusCartera],
	[CountryId],
	[Country],
	[Nacionalidad],
	[ContractCurrency],
	[Compania],
	[CompaniaDesc],
	[CodCompaniaContable],
	[LenderTrxId],
	[CurrentLenderid],
	[CurrentLender],
	[MortgageCurrencyId],
	[MortgageCurrency],
	[Numpagoshechos],
	[PaymentAmount],
	[APR],
```

```
[BankID],
[ForAmount],
[BkDesc],
[UserName],
[CodTransanccion],
[TransactionCode],
[MextPaymentDueDate],
[Iva],
[TransactionCodeId],
[Amount],
[BaseAMountTrx],
[AccAmount],
[BaseAmountTrx_S_IVA],
[ToCapital],
[To_Capital_S_IVA],
[ToInterest],
[To_Interest_IVA],
[To_Interest_S_IVA],
[Imp_Transaccion_IVA],
[ToCapital_IVA],
[ToLateFees],
[FromMemo],
[ToMemo],
[CurrentPrincipalBal],
[InstrumentPayID],
[IntrumentDescription],
[SaleYear],
[Reference],
[AuthCode],
[Morosidad_Tipo],
[Morosidad_Status],
[Morosidad_Tipo_Inicial],
[Morosidad_Inicial],
[ClubID],
[ClubName],
[FundingInstitutionID],
[CiaCobranza],
[FundingStatusID],
[Fundingstatus],
[TrxCurrencyID],
[TrxCurrency],
[CiaContableMortgage],
[LoanPortfolioid],
[LoanPortolio],
[PortfolioComment],
[AssignedDate],
[PaymentRemaining],
[OwnerId],
[GrupoCobranza],
[user10],
[CiaContableRCC],
[ReferenceMBT],
[TrxExchangeRate],
[BillingCodeID],
[BillingCode],
[PaymentType],
[MortgagePaymentID],
[Origen],
```

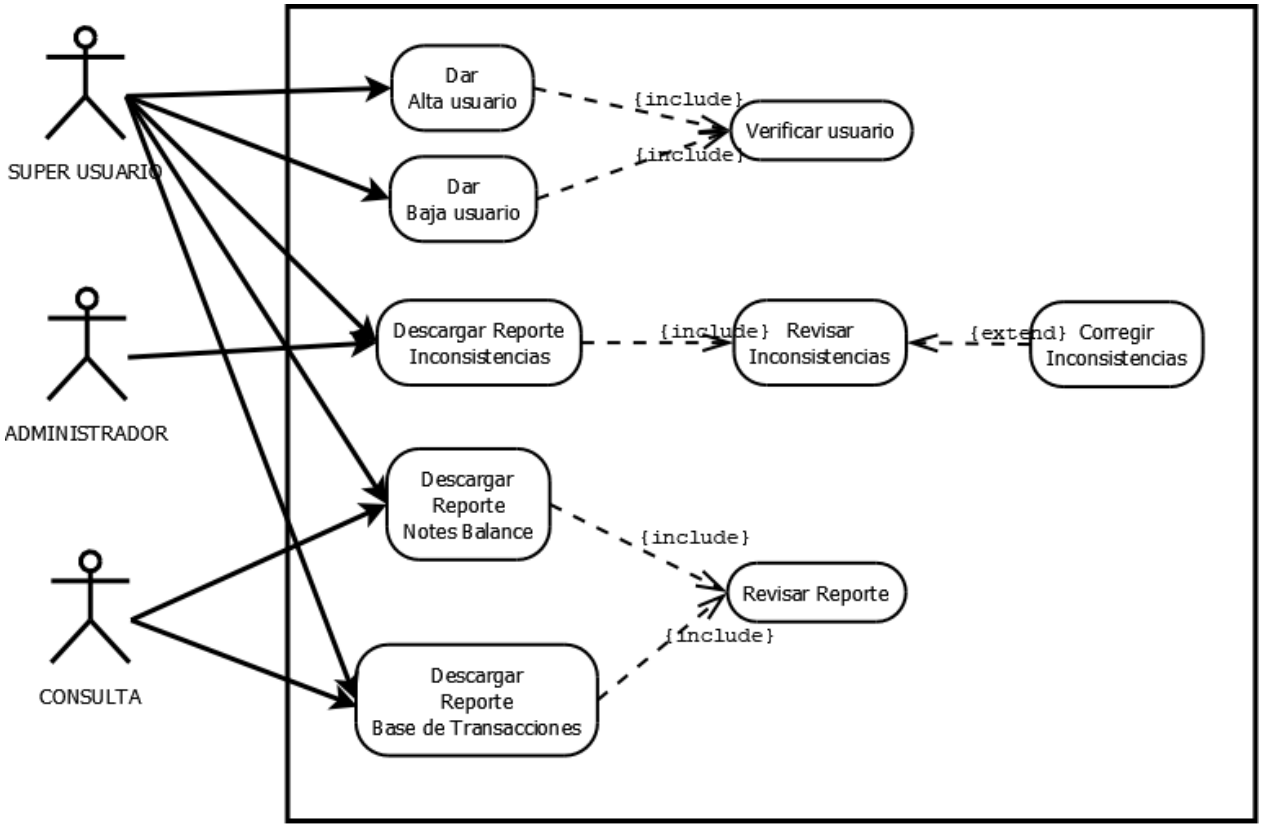
```
        [LenderTrx],
        [Site_Origen],
        [CSC_AccNum],
        [CSCTransType],
        [RowNum],
        [InsertDate])
SELECT [ContractID],
        [mortgageid],
        [lastName],
        [Firstname],
        [ContractNumber],
        [Mes],
        [Anio],
        [TransactionDate],
        [postedDate],
        [SiteName],
        [SiteId],
        [MortgageNumber],
        [SaleDate],
        [CloseDate],
        [TipoContrato],
        [PkgType],
        [Points],
        [ContractStatus],
        [StatusCarteraid],
        [StatusCartera],
        [CountryId],
        [Country],
        [Nacionalidad],
        [ContractCurrency],
        [Compania],
        [CompaniaDesc],
        [CodCompaniaContable],
        [LenderTrxId],
        [CurrentLenderid],
        [CurrentLender],
        [MortgageCurrencyId],
        [MortgageCurrency],
        [Numpagoshechos],
        [PaymentAmount],
        [APR],
        [BankID],
        [ForAmount],
        [BkDesc],
        [UserName],
        [CodTransaccion],
        [TransactionCode],
        [NextPaymentDueDate],
        [Iva],
        [TransactionCodeId],
        [Amount],
        [BaseAMountTrx],
        [AccAmount],
        [BaseAmountTrx_S_IVA],
        [ToCapital],
        [To_Capital_S_IVA],
        [ToInterest],
        [To_Interest_IVA],
```

```
        [To_Interest_S_IVA],
        [Imp_Transaccion_IVA],
        [To_Capital_IVA],
        [ToLateFees],
        [FromMemo],
        [ToMemo],
        [CurrentPrincipalBal],
        [InstrumentPayID],
        [InstrumentDescription],
        [SaleYear],
        [Reference],
        [AuthCode],
        [Morosidad_Tipo],
        [Morosidad_Status],
        [Morosidad_Tipo_Inicial],
        [Morosidad_Inicial],
        [ClubID],
        [ClubName],
        [FundingInstitutionID],
        [CiaCobranza],
        [FundingStatusID],
        [Fundingstatus],
        [TrxCurrencyID],
        [TrxCurrency],
        [CiaContableMortgage],
        [LoanPortfolioid],
        [LoanPortfolio],
        [PortfolioComment],
        [AssignedDate],
        [PaymentsRemaining],
        [OwnerId],
        [GrupoCobranza],
        [user10],
        [CiaContableRCC],
        [ReferenceMBT],
        [TrxExchangeRate],
        [BillingCodeID],
        [BillingCode],
        [PaymentType],
        [MortgagePaymentID],
        [Origen],
        [LenderTrx],
        [Site_Origen],
        [CSC_AccNum],
        [CSCTransType],
        [RowNum],
        GETDATE()
FROM ##TBLPASOTRAXREPORT

print 'datos insertados'

END
```

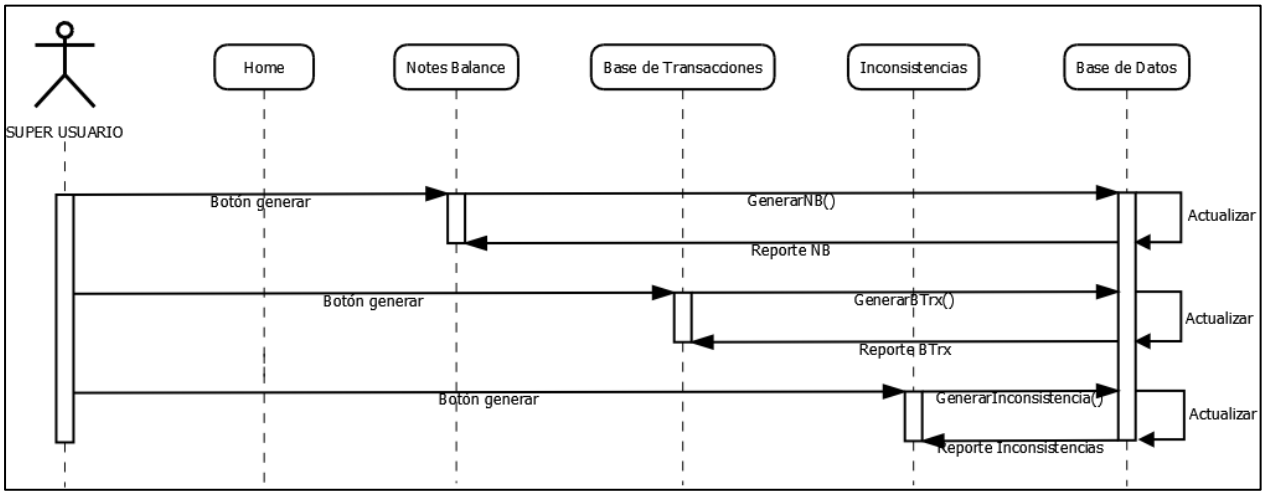
Anexo 6: Caso de Uso de Perfil



Anexo 7: Caso de Uso de Rol



Anexo 8: Diagrama de Secuencia



REFERENCIAS

[1] Sommerville Ian, *Ingeniería de Software 9*, Pearson, 2011.
[2] Pressman Roger, *Ingeniría del Software. Un enfoque práctico / 7 ED.*, Mc Graw Hill, 2010.
[3] Freeman Adam, *Pro ASP.NET MVC 5*, Apress, 2014.
[4] Firtman Maximiliano, Natale Leonardo, *Visual Studio .NET Framework 3.5 para profesionales*, Alfaomega Grupo Editor, 2010.
[5] Rui Machado, *SSIS Succinctly*, Syncfusion, 2014.
[6] http://enc2014.cicese.mx/Memorias/paper_17.pdf
[7] http://www.desarrolloweb.com/articulos/1718.php?manual=54

Otras referencias

- https://www.visualstudio.com/es/
- https://www.microsoft.com/es-es/sql-server/sql-server-downloads
- https://www.javascript.com/
- https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS
- https://www.sublimetext.com/