ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN SISTEMA PARA GESTIÓN DE ROLES DE PAGO CON ENVÍO DE NOTIFICACIONES EN TIEMPO REAL

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN INGENIERÍA ELECTRÓNICA Y REDES DE INFORMACIÓN

JONNY FABIAN BONILLA AYALA

jonny.bonilla@epn.edu.ec

DIRECTOR: PH.D. ANA MARIA ZAMBRANO VIZUETE

ana.zambrano@epn.edu.ec

Quito, Julio 2019

AVAL

Certifico que el presente trabajo fue desarrollado por Jonny Fabián Bonilla Ayala bajo mi supervisión.

Ph.D. ANA MARÍA ZAMBRANO
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Jonny Fabián Bonilla Ayala, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

JONNY FABIÁN BONILLA AYALA

DEDICATORIA

Dedico este trabajo a mis padres, por trabajar cada día para darme lo mejor y brindarme las oportunidades que ustedes no las tuvieron.

Dedico este trabajo a mis hijos, ya que gracias a su inocencia y alegría me llenan de energía y ganas para seguir adelante y buscar un mejor futuro para ustedes.

Jonny Fabián Bonilla Ayala

AGRADECIMIENTO

Agradezco a Dios por la fuerza que me brinda cada día para seguir adelante, por acompañarme en todos los momentos difíciles y brindarme el aliento para culminar las metas que muchas veces parecían imposibles.

Gracias a mis padres Verónica y José por brindarme la oportunidad de estudiar, ya que sin ellos no hubiese sido posible alcanzar este objetivo.

Gracias a mi compañera y amiga Diana, por aportar con su granito de arena en este proyecto y sobre todo ser mi apoyo en esta etapa tan importante para nuestras vidas.

Agradezco a mi directora de tesis Ph.D. Ana María Zambrano por su tiempo, paciencia y ser mi luz guía para alcanzar esta meta.

Jonny Fabián Bonilla Ayala

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

Una tarea obligatoria que posee toda empresa legalmente constituida en el Ecuador es el registro contable por el concepto de pago de servicios prestados por sus empleados, esto se lo conoce comúnmente como Rol de Pagos. La generación de Roles de Pago es una tarea que se realiza todos los meses del año y dependiendo del número de empleados puede tomar una cantidad considerable de tiempo, por tal motivo es necesario un sistema que permita automatizar dicha tarea. Es importante mencionar que la empresa tiene la obligación de notificar el rol de pagos a sus empleados para que quede constancia del pago realizado y evitar posibles demandas por incumplimiento. Por lo expuesto anteriormente en el presente Proyecto de Titulación se implementará un sistema que ayude a la generación de los roles de pago para una empresa y además enviará notificaciones a sus empleados a una aplicación móvil con sistema operativo *Android* para que puedan verificar la información generada. El sistema también busca facilitar la generación de asientos contables que pueden ser utilizados en los balances de la empresa.

El presente documento se encuentra dividido de la siguiente manera: en el primer capítulo se encuentra el marco teórico con todos los conceptos necesarios. En el segundo capítulo se encuentra de manera detallada el diseño e implementación del sistema. En el tercer capítulo se observan las pruebas donde se verifica el correcto funcionamiento y por último en el cuarto y quinto capítulo se pueden observar las conclusiones, recomendaciones y anexos.

PALABRAS CLAVE: rol de pago, .Net Remoting, Openfire, XMPP, Notificaciones Push, Android, Xamarin.

ABSTRACT

A mandatory task that every company legally constituted in Ecuador has is the accounting record for the concept of payment of services provided by its employees, this is commonly known as the Payment Role. The generation of Payment Roles is a task that takes place every month of the year and depending on the number of employees it can take a considerable amount of time, for this reason a system is necessary to automate this task. It is important to mention that the company has the obligation to notify the role of payments to its employees so that there is proof of the payment made and to avoid possible claims for non-compliance. Due to the foregoing in this Project will implement a system that helps the generation of payment roles for a company and also send your employees a notification to mobile devices with Android operating system so they can verify the information generated . The system also seeks to facilitate the generation of accounting entries that can be used in the balance sheets of the company.

This document is divided as follows: in the first chapter there is the theoretical framework with all the necessary concepts. In the second chapter, the design and implementation of the system is detailed. In the third chapter we observe the tests where the correct functioning is verified and finally in the fourth and fifth chapters we can observe the conclusions, recommendations and annexes.

KEY WORDS: payment role, .Net Remoting, Openfire, XMPP, Push Notifications, Android, Xamarin.

1. INTRODUCCIÓN

1.1. OBJETIVOS

El objetivo general de este Proyecto de Titulación es desarrollar un sistema para la gestión de roles de pago con envío de mensajes de notificación al empleado en tiempo real. Los objetivos específicos del proyecto son:

a) Analizar la situación actual de la generación de roles de pago.

b) Diseñar los componentes del sistema.

c) Implementar las funcionalidades de los componentes del sistema.

d) Analizar los resultados de las pruebas realizadas con los distintos dispositivos que soporten el sistema operativo *Android*.

1.2. ALCANCE

El presente Proyecto de Titulación tiene como primer objetivo exponer en qué consiste la gestión del rol de pagos en Ecuador, para lo cual, se revisará la documentación publicada por el gobierno sobre las obligaciones patronales. Una vez analizada la parte funcional se procederá a revisar la tecnología *.Net Remoting* y el protocolo *XMPP* (*Extensible Messaging and Presence Protocol*) con el propósito de realizar el envío de notificaciones en tiempo real a un empleado con el detalle del último rol de pagos; esto se lo realizará a través del servidor *Openfire*, el cual dispone del protocolo mencionado, estableciéndose los requisitos para incorporarlo en el Proyecto de Titulación.

Durante el desarrollo del Proyecto de Titulación se empleará la metodología *Scrum*; además se establecerán los requerimientos funcionales y no funcionales para la generación del rol de pagos.

El gestor de base de datos que se utilizará será *Sql-Server Express*, por otra parte, los reportes se realizarán utilizando *Crystal Report*.

Además, se desarrollará una aplicación móvil para ser instalado sobre el sistema operativo *Android* utilizando *Xamarin*, componente que pertenece a *Visual Studio*.

En resumen, este Proyecto de Titulación se centra en la gestión de roles de pago, que incluye: registro de información de los empleados, registro de tipo de descuentos que dispone la empresa, generación del rol de pagos individual y provisión, generación de asientos contables del rol de pagos, impresión de reportes y lo más relevante, la incorporación de notificaciones a sus empleados.

1.3. MARCO TEÓRICO

Esta sección tiene como objetivo, primeramente, brindar una descripción del marco teórico referente a los conceptos básicos del rol de pagos, describiendo cuáles son sus elementos y para qué sirve el registro del mismo en una empresa. Por otra parte, se revisará la tecnología *.Net Remoting,* la cual nos permitirá realizar el desarrollo de una aplicación distribuida y emplearla en el sistema de gestiones de roles de pago. Además, se revisará brevemente el protocolo *XMPP (Extensible Messaging and Presence Protocol),* el cual nos permitirá enviar notificaciones en tiempo real a dispositivos móviles; dichos dispositivos utilizarán el sistema operativo *Android*. Dentro de este Apartado también se revisará específicamente el servidor *Openfire*, el cual utiliza el protocolo antes mencionado.

Una vez repasado los temas anteriores se revisará la metodología ágil de desarrollo de software *Scrum,* ya que el Proyecto de Titulación estará basado en las buenas prácticas de dicha metodología para facilitar el desarrollo del sistema de roles de pago.

1.3.1. ROL DE PAGOS

a) Definición

El rol de pagos también conocido como nómina de pago, es un documento que se entrega al empleado en el que se detalla los ingresos que recibe, así como los descuentos que se le va a aplicar según la ley vigente, quedando como resultado el salario neto a percibir [1]. Este proceso se lo puede llevar a cabo de manera quincenal o mensual [1].

El rol de pagos está conformado por:

- **Encabezamiento**: Información general de la empresa [1].
- **Datos generales del empleado:** Información correspondiente al empleado o beneficiario del rol de pagos [1].
- **Ingresos:** Conceptos por los cuales el empleado recibe un pago a su favor, entre los cuales pueden existir los siguientes conceptos: sueldo o salario, horas extras, horas suplementarias, decimotercer sueldo, decimocuarto sueldo, pago de utilidades, fondos de reserva, etc [1].
- **Descuentos:** Conceptos por los cuales se realiza un decremento en los ingresos de un empleado, por ejemplo: Seguridad social, comedor, multas, préstamos, anticipos [1].
- **Total:** Valor resumen de ingresos y egresos [1].

- **Firmas de responsabilidad y recibimiento:** Firmas de aceptación del empleado y empleador de los valores generados en el rol [1].

b) Tipos de roles de pago

El rol de pagos puede ser general o individual [1].

- **Rol de pagos general:** El rol de pagos es el registro que toda empresa debe realizar, ya que mediante este proceso se realiza el control de los pagos y descuentos que se deben realizar a sus empleados cada mes [1]. Mediante un rol general se dispone de información de los ingresos y descuentos en una lista de todos los empleados identificados por departamentos que disponga la empresa [1]. Esta información ayuda a la empresa para el registro contable de los movimientos de dinero por concepto de pago de empleados [1].

En la Figura 1.1 se muestra una plantilla de un rol de pagos general.

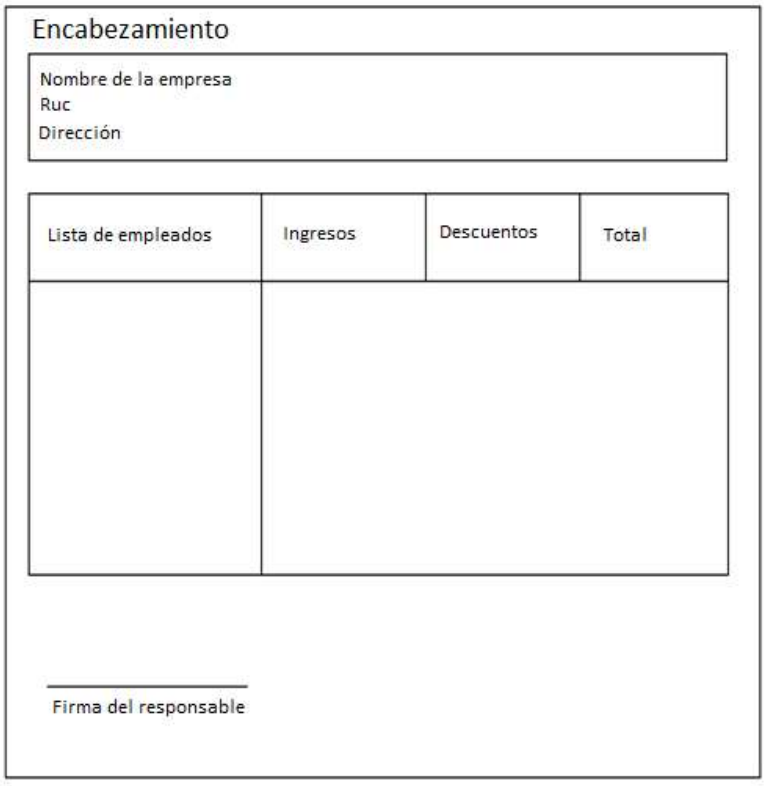

Figura 1.1 Plantilla de rol de pagos general.

- **Rol de pagos individual:** Este tipo de rol recibe de manera individual cada empleado, se imprime por duplicado donde la primera copia le pertenece al empleado y el otro la conserva la empresa debidamente firmado como constancia

de haber realizado el pago [1]. Este documento puede ser utilizado como respaldo en un posible reclamo o demanda [1].

En la Figura 1.2 se muestra una plantilla para un rol de pagos individual.

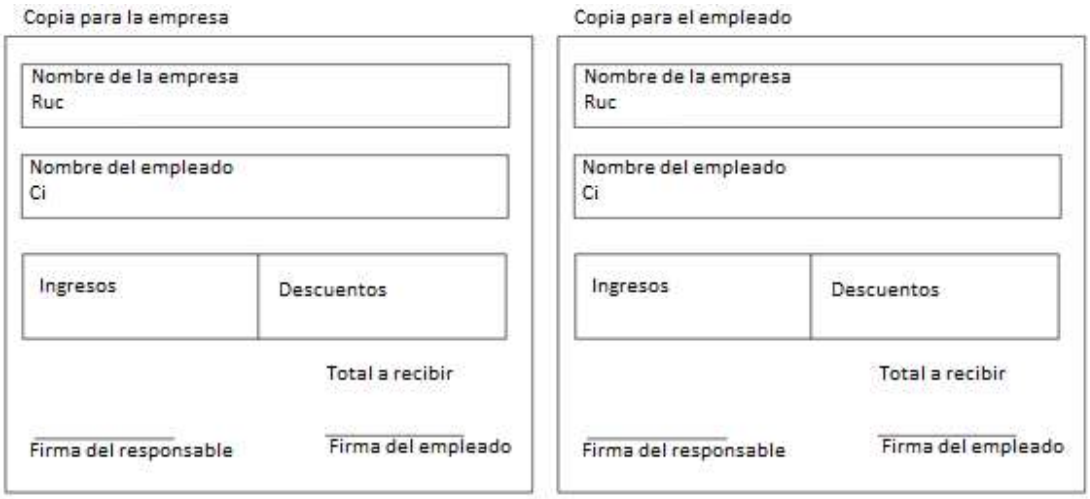

Figura 1.2 Plantilla de rol de pagos individual.

Todo rol de pagos tiene como consecuencia un asiento contable que es registrado para que se refleje en el balance de la empresa. Los asientos contables son registros que se realizan de todas las actividades financieras que se producen en los negocios [2]. Cada vez que se realice un cambio en el patrimonio de la empresa, debe registrarse contablemente sobre este cambio; por ejemplo si una empresa realiza una venta, debe quedar registrado el ingreso de dinero que se produjo por dicha venta [2]. Un asiento contable se compone de al menos dos anotaciones (cuentas contables) una al debe o débito y la otra al haber o crédito [2].

Se denominan cuentas contables al conjunto de registros donde se detallan de forma cronológica todas las transacciones que ocurren en un ente económico (empresa o negocio) [3].

Una cuenta contable está compuesta por:

- **Titular o nombre de la cuenta:** Nombre que se asigna a la cuenta contable para poder identificar el origen de la misma, por ejemplo: venta de mercadería en Quito, Sueldo de empleados del departamento de sistemas [3].
- **Debe o débito:** Valor que se coloca en la parte izquierda de la cuenta y representa todo dinero que entra al negocio o el motivo de por qué salió [3].

- **Haber o crédito:** Es aquel que se ubica en la parte derecha de la cuenta y representa todo dinero que sale de la empresa o el motivo por qué entró [3].
- **Saldo:** El saldo es la diferencia entre el “debe” y el “haber”. Si los débitos de la cuenta son mayores a los créditos, se dice que existe un saldo deudor; si el valor de los créditos es mayor a los débitos de la cuenta se dice que existe un saldo acreedor [3].

En la Figura 1.3 se muestra un diagrama de una cuenta contable:

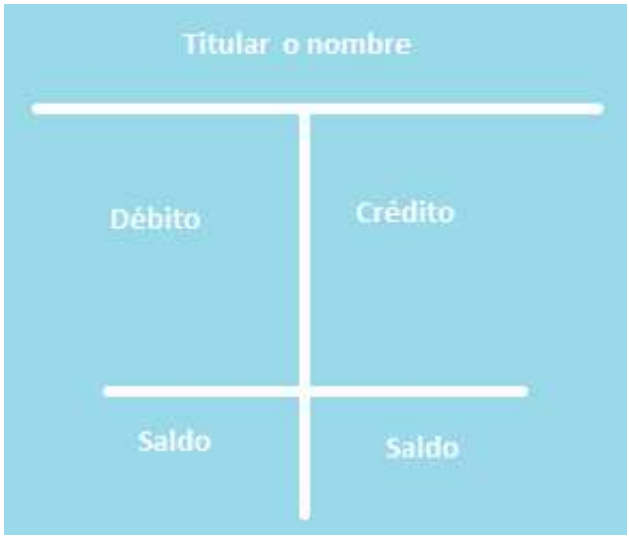

Figura 1.3 Cuenta contable **[3]**.

1.3.2. .NET REMOTING

.Net Remoting es una tecnología propietaria de *Microsoft* que facilita la implementación de aplicaciones distribuidas. Esta tecnología permite que aplicaciones cliente utilicen componentes en equipos remotos y pueda usarlos como si fuesen componentes locales.

Existen dos tipos de interacción remota entre componentes; el primero utiliza objetos serializables los cuales envían una copia del objeto al proceso remoto (*ByValueObjects*) [4], mientras que, el segundo emplea objetos del lado del servidor (remoto) que permiten utilizar sus métodos (*MarshalbyRefObjects*) [5].

MarshalByRefObjects puede ser activado de dos maneras:

- *Server Activated Object* (SAO).
- *Cliente Activated Object* (CAO).

En SAO, la vida de los objetos es controlado por el servidor, los mismos que pueden ser especificados en los modos *Singlecall* y *Singletone* [6].

Para los objetos *Singlecall* el servidor crea un objeto, ejecuta el método solicitado y destruye el objeto [5]. En la Figura 1.4 se muestra un objeto *Singlecall*.

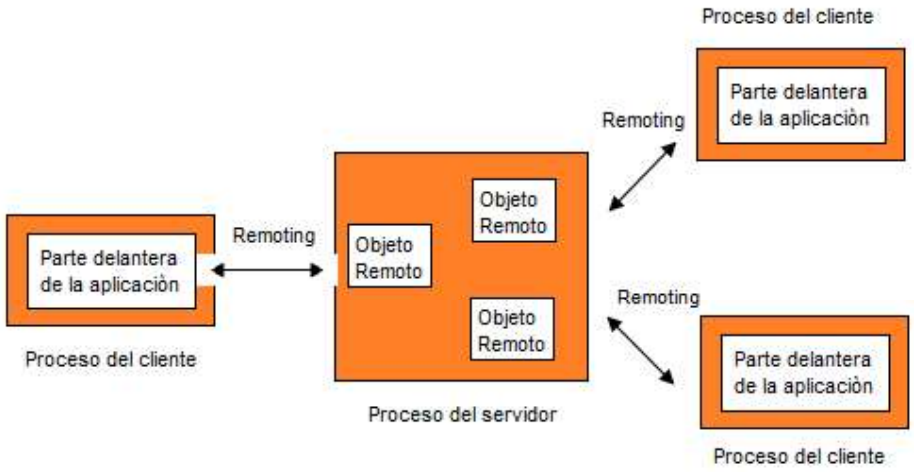

Figura 1.4 Objetos *Singlecall* **[7]**.

Mediante *Singletone* solo puede existir una instancia de un objeto. Cuando el servidor recibe la petición de un cliente, el servidor comprueba sus tablas internas para determinar si ya existe una instancia de esta clase; si no existe, este objeto será creado y almacenado en la tabla. Después de esta comprobación el método será ejecutado. Mediante este proceso el servidor garantiza que habrá exactamente una o ninguna instancia disponible de ese objeto [5]. En la Figura 1.5 se muestra un objeto *Singletone*.

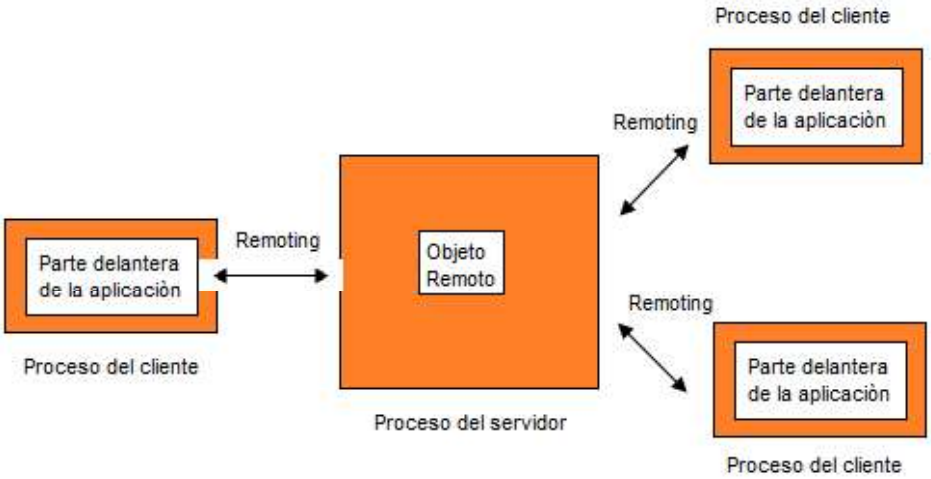

Figura 1.5 Objetos *Singletone* **[7]**.

En CAO el tiempo de vida del objeto del servidor es controlado por los clientes. Este método no es recomendable cuando el servidor hace la mayor parte de la lógica de negocio y ejecución, pero si se lo aplica en el caso que el cliente realiza tareas críticas en las que la perdida de información puede afectar al sistema [5].

1.3.3. PROTOCOLO *XMPP*

XMPP (*Extensible Messaging And Presence Protocol*) es un protocolo de la capa aplicación diseñado originalmente por Jeremie Miller en 1998 para la mensajería instantánea (originalmente llamado *Jabber*) [8]. El protocolo consiste en un sistema de mensajería presencial el cual usa mensajes *XML* para implementar las funcionalidades necesarias en este tipo de mensajería [8]. Es un protocolo libre y abierto creado con la intención de unificar los servicios de mensajería instantánea [8].

a) Arquitectura de *XMPP*

XMPP tiene como una de sus funciones hacer la mensajería instantánea entre los usuarios de manera accesible y menos compleja, es por esto que dispone de una arquitectura cliente-servidor descentralizada, es decir, para la comunicación entre clientes no es obligatorio que pase por un servidor central, sino que cualquier cliente tiene la capacidad de implementar su propio servidor *XMPP* y unirse a la red [8]. Cuando un "cliente *XMPP1*" envía un mensaje *XMPP* a un "cliente *XMPP2*" el cual posee un servidor *XMPP* diferente al primero, el primer cliente se conecta con su servidor *XMPP* y dicho servidor se contacta con el servidor del segundo cliente y posteriormente se entrega el mensaje, esto se muestra en la Figura 1.6.

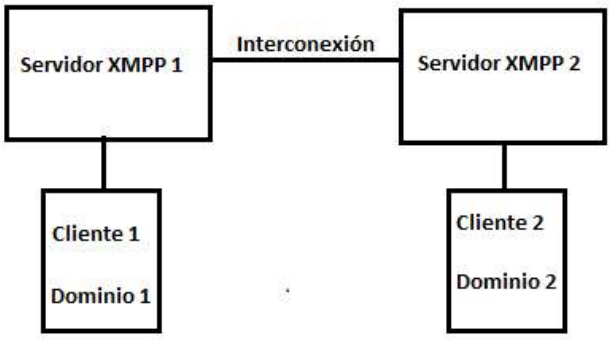

Figura 1.6 Arquitectura XMPP**[8]**.

b) Direcciones *XMPP*

XMPP se basa normalmente en nombres de dominio (*DNS*) para proporcionar la dirección a un usuario, en lugar de usar una dirección IP; además el puerto destino para la recepción

de mensajes es el 5222 [9]. En *XMPP* se establece el concepto *Jabber ID* [10], el cual está compuesto por un nombre de usuario, un dominio y un recurso, estructurado de la siguiente manera:

JID=<usuario@dominio>/recurso

Un ejemplo se puede observar en *Whatsapp,* ya que usa *XMPP* como servicio de notificaciones [11].

JID=<593994462627@*whatsapp.net*>/*Whatsapp*

En *Whatsapp* el **usuario** es el código de país más el número de teléfono, el **dominio** es el servidor en el que reside el usuario, y el **recurso** es una cadena de referencia que identifica al usuario que está utilizando el servicio que por lo general es el nombre del dispositivo, ubicación o software cliente.

c) Comunicación *XMPP*

En *XMPP* la información se envía como texto *XML*. Cuando se inicia una sesión con un servidor *XMPP* se genera una conexión *TCP* persistente ("*Always On*") y luego se negocia un *stream XML* con el servidor.

Una vez establecida la conexión con el servidor, el servidor abre un *stream* de retorno con el cliente, es decir existe un flujo de datos en ambos sentidos. De este modo tanto el cliente como el servidor pueden intercambiar de manera asincrónica y en ambos sentidos tres fragmentos especiales *XML* [8], estos son:

- ***Message* (<*message*/>):** Mecanismo de envío de mensaje de una entidad a otra, existiendo 5 tipos: Normal, *Chat*, *GroupChat*, *HeadLine*, Error [12].
- ***Presence* (<*presence*/>):** Mecanismo utilizado para mostrar la presencia de un usuario a otros usuarios. Este puede mostrar el estado en línea, cambio de estado, *nickname,* etc [12].
- **Iq (<iq/>):** (*Info*/*Query*) es una interacción de solicitud-respuesta con el cual una entidad es capaz de solicitar información a otra entidad, pero no interfiere con la interfaz de comunicación. Existen 4 tipos: *Get*, *Set*, *Result*, Error [12].

En el Código 1.1 se muestra un ejemplo de sesión en *XML* de *XMPP* [13], donde se encuentra resaltado el tipo de mensaje como normal, el tipo de *iq* en *Get.* Además, se puede observar la dirección del usuario origen, la dirección del usuario destino y el cuerpo del mensaje.

```
<iq type='get'
       from='romeo@montague.lit/pda'>
       <query xmlns='jabber:ip:roster'/>
</channel>

<iq type='result'
       to='romeo@montague.lit/pda'>
  <query xmlns='jabber:ip:roster'/>
       <item jid='juliet@capulet.lit'/>
       <item
        jid='mercutio@shakespeare.lit'/>
       <item
       jid='benvolio@shakespeare.lit'/>
  </query>
</ip>
```

Código 1.1 Ejemplo de sesión *XMPP*.

d) Servidor *Openfire*

Openfire es un servidor de software libre para mensajería instantánea que utiliza el protocolo *XMPP,* mediante el cual un desarrollador puede implementar su propio servicio de mensajería instantánea permitiendo administrar usuarios, enviar mensajes simples, de grupo o de *broadcast*; además se puede compartir archivos, implementar mensajes de voz y video llamadas mediante *plugins* adicionales [14]. *Openfire* dispone de una interfaz web para la administración del servidor que ejecuta por defecto en el puerto 9090 (*HTTP*) y 9091 (*HTTPS*) [15]. *Openfire* cuenta con las siguientes características:

- Panel de administración vía web (puerto 9090 *HTTP* y 9091 *HTTPS*)
- Interfaz para agregar *plugins* adicionales
- Compatible con *SSL*/*TLS*.
- Puede interactuar con *MSN*, *Google Talk*, *Yahoo Messenger*, entre otros.
- Puede comunicarse con otros servidores *XMPP*.
- Transferencia de archivos.
- Compresión de datos.
- Mensajes *offline*.
- Autenticación vía certificados, *Kerberos*, *LDAP*, *PAM* y *Radius*.
- Permite el almacenamiento de datos en *SQL Server*, *MySQL*, *Oracle* y *PostgreSQL*.
- Se encuentra disponible para *Mac*, *Windows* y *Linux*.

Nota: Toda la documentación, instaladores, *plugins*, acuerdos de licencia, manuales de instalación se encuentran disponibles en la página de *Ignite Realtime* [15].

1.3.4. METODOLOGÍA DE DESARROLLO DE SOFTWARE *SCRUM*

Una metodología de desarrollo de software es un marco de trabajo que ayuda con el diseño e implementación de un sistema a través de la planificación, estructura y lineamientos con los que cada una cuenta. Para el propósito de este Proyecto de Titulación se ha decidido utilizar *Scrum*, que es parte de las metodologías ágiles de desarrollo.

Scrum se basa en la teoría de control de procesos de manera empírica; es decir, el empirismo asegura que el conocimiento procede de la experiencia y poder tomar decisiones basadas en lo que se conoce [16]. *Scrum* tiene un enfoque iterativo e incremental para optimizar la predictibilidad y el control del riesgo [16]. El enfoque iterativo es debido a que *Scrum* utiliza *Sprints* o iteraciones de tiempo para realizar un incremento o entregable del producto [16].

a) Definición de *Sprint*

El corazón de *Scrum* es el *Sprint,* se define como un periodo de tiempo (*time-box*) en el cual se crea un incremento en el producto "Terminado" utilizable y potencialmente desplegable [16]. Cada nuevo *Sprint* comienza inmediatamente después de la finalización del *Sprint* anterior [16]. Además, *Scrum* gestiona las iteraciones a través de reuniones diarias como se observa en la Figura 1.7, ya que una vez que inicia el *Sprint* se realiza un análisis diario de los avances o dificultades se han presentado durante el desarrollo [16].

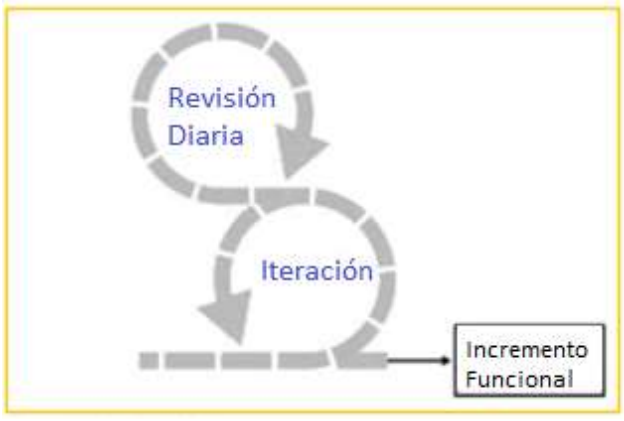

Figura 1.7 Ciclo principal de *Scrum* **[9]**.

Durante un *Sprint* no se pueden realizar cambios que puedan afectar el objetivo también denominado (*Sprint Goal*) [16].

b) Roles de *Scrum*

El equipo *Scrum* está formado por los siguientes roles [17]**:**

- ***Scrum master*****:** persona encargada de comprobar que la metodología y el modelo funciona, además eliminará los inconvenientes que se presenten. También es la persona que interactuará con el cliente y los gestores [17].
- ***Product owner* (PO):** persona que conoce el negocio del cliente y su visión del producto por lo que toma las decisiones. Escribe las ideas del cliente ordenandolas por prioridad [17].
- ***Team*****:** Grupo de profesionales con los conocimientos técnicos necesarios que desarrollan el proyecto de manera conjunta llevando a cabo las historias de usuario a las que se comprometen al inicio de cada *Sprint* [17].

c) Componentes de *Scrum*

Los componentes de esta metodología se describen a continuación [18]:

- **Pila del producto (*Product Backlog*):** La Pila del Producto enumera todas las características, funcionalidades, requisitos, mejoras y correcciones que constituyen cambios a realizarse sobre el producto [18].
- **Incremento:** un avance del producto desarrollado en un *Sprint* con la particularidad de que se le ha realizado pruebas y se lo ha documentado, es decir, está listo para ser ejecutado [18].

d) Ciclo de *Scrum*

El ciclo de desarrollo de *Scrum* consta de 5 fases que definen el desarrollo ágil [19]:

- **Concepto:** es donde se conocen de manera general los requisitos a desarrollar y se asigna el equipo de desarrollo correspondiente [19].
- **Especulación:** se establece los alcances de los requerimientos funcionales y no funcionales, y en base a ello, se marca la complejidad y tiempo necesario para implementarlo [19]. Se realiza el plan de entrega y se empieza con el desarrollo del producto realizando pruebas al mismo para comprobar la funcionalidad que se espera [19].
- **Exploración:** En base a la fase de especulación, y de acuerdo a los resultados obtenidos se realizan al producto incrementos de funcionalidad [19].

- **Revisión:** El equipo realiza una revisión del producto obtenido y se compara con el producto deseado [19].
- **Cierre:** Siguiendo el plan de entrega, en la fecha correspondiente, se da una versión del producto deseado. Al tratarse de una versión, el cierre no significa que se ha dado por terminado el producto, sino que en esta etapa ya es propenso a ser objeto de cambios, denominados "mantenimiento" para así acercarse al producto final deseado [19].

En la Figura 1.8 se observa el ciclo de *Scrum* en el cual como primer paso se realiza el análisis de los requerimientos con el objetivo de definir la pila de producto o *Product Backlog.* Posteriormente, se realiza la planificación del *Sprint* estableciento el tiempo y tareas que se deben realizar. Una vez definido el *Sprint*, se inicia el desarrollo e implementación de las tareas anteriormente planificadas. Una vez terminado el *Sprint,* se obtendrá un incremento funcional para el cliente.

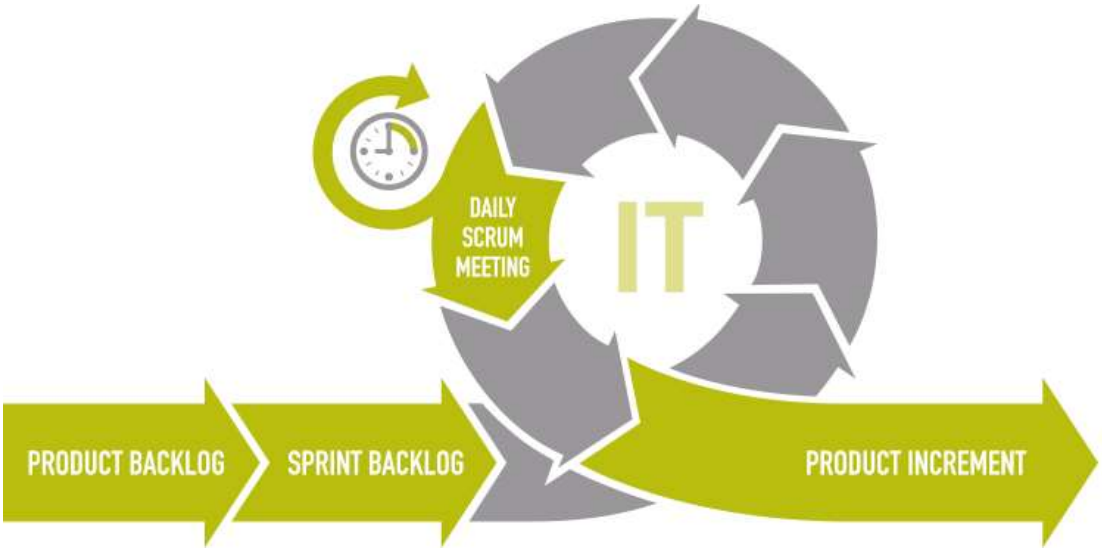

Figura 1.8 Ciclo de *Scrum* **[20]**.

e) Definición de Historia de Usuario

Una vez revisado la metodología *Scrum* se debe definir lo que es una historia de usuario, ya que mediante esta herramienta se obtendrán los requerimientos funcionales del sistema. Una historia de usuario describe una funcionalidad que debe incorporar un sistema software y cuya implementación aporta valor al cliente [21].

Una historia de usuario contiene la siguiente información:

- Nombre

- Descripción de la funcionalidad.
- Personas responsables de la implementación.

El nombre debe dar una pista clara de la funcionalidad que se cumplirá una vez implementada, tratando que sea lo más corto posible. En la descripción se detalla en lenguaje natural lo que el cliente espera que realice el sistema. Por último, se debe incluir el nombre de la persona encargada en el equipo en realizar la implementación del requerimiento. En la Tabla 1.1 se muestra la plantilla de una historia de usuario:

Tabla 1.1 Plantilla para historia de usuario.

Historia de usuario	
Código y número:	**Tipo:**
Nombre de Historia:	
Programador responsable:	
Descripción:	
Observación: (opcional)	

Para detallar las historias de usuario se ha utilizado una plantilla que consta de los siguientes elementos:

- **Código y número:** Cada historia de usuario estará identificada por el código **HU** seguido de un número consecutivo de tres dígitos.
- **Tipo:** Describe si el requerimiento es funcional o no funcional, su código será **RF** seguido de un número consecutivo de tres dígitos.
- **Nombre de historia:** Nombre único que identificará a la historia de usuario, además del código y número.
- **Programador responsable:** Nombre de la persona responsable de la implementación del requerimiento.
- **Descripción:** Detalla de manera breve el requerimiento solicitado por el cliente.
- **Observaciones:** Campo adicional en el cual se puede detallar una observación para que el programador tome en cuenta al momento de implementar el requerimiento.

2. METODOLOGÍA

En esta sección de detalla la manera en que se obtuvo los requerimientos del software, posteriormente se muestra el diseño realizado, la instalación de herramientas necesarias para la codificación y por último se tiene la implementación del sistema.

2.1. DISEÑO

A continuación, se detalla el diseño a realizar para la generación de los roles de pago y el envío de las notificaciones a los dispositivos móviles con sistema operativo *Android* [22]. Este sistema se lo nombrará como **SRSOFT**. Como primer punto se realizará encuestas con el fin de obtener los requerimientos funcionales y no funcionales del sistema y poder realizar un diseño que una vez implementado nos permita generar un rol de pagos de una manera más sencilla para las personas encargadas de esta tarea. Con estos requerimientos se realizará las historias de usuario que detallarán las funcionalidades necesarias para el sistema y además definir el *Product Backlog*. A continuación, se definirá los *Sprints* en los cuales se contemplará las tareas necesarias para cumplir con el análisis de requerimientos, diseño del sistema, implementación de sus componentes y pruebas de verificación.

2.1.1. REQUERIMIENTOS DEL SOFTWARE

Una vez realizadas las encuestas y entrevistas a personas encargadas de generar los roles de pago, se deben elaborar las historias de usuario para los requerimientos funcionales y no funcionales del sistema. Dichas encuestas se las puede visualizar en el **Anexo A**. Mediante las historias de usuario se puede determinar las funcionalidades del software y además nos permite definir el *Product Backlog*. Desde la **Tabla 2.1** hasta la **Tabla 2.26** se describe las historias de usuario con los requerimientos del software a desarrollarse. Es muy importante mencionar que cada historia de usuario tiene un identificativo único.

Tabla 2.1 Historia de Usuario Ficha del empleado

Historia de usuario	
Código y número: HU001	**Tipo:** RF001
Nombre de Historia: Ficha del empleado.	
Programador responsable: Jonny Bonilla.	
Descripción: Ingresar, modificar y eliminar la ficha del empleado.	

Tabla 2.2 Historia de Usuario Gestión de cargos

Historia de usuario	
Código y número: HU002	**Tipo:** RF002
Nombre de Historia: Gestión de cargos.	
Programador responsable: Jonny Bonilla.	
Descripción: El sistema debe permitir la creación, modificación y eliminación de cargos. Para eliminar el registro de un cargo, no debe existir transacciones realizadas.	

Tabla 2.3 Historia de Usuario Gestión de departamentos

Historia de usuario	
Código y número: HU003	**Tipo:** RF003
Nombre de Historia: Gestión de departamentos.	
Programador responsable: Jonny Bonilla.	
Descripción: El sistema debe permitir la creación, modificación y eliminación de departamentos. Para eliminar el registro de un departamento, no debe existir transacciones realizadas.	

Tabla 2.4 Historia de usuario Gestión de cuentas contables

Historia de usuario	
Código y número: HU004	**Tipo:** RF004
Nombre de Historia: Gestión de cuentas contables.	
Programador responsable: Jonny Bonilla.	
Descripción: El sistema debe permitir la creación, modificación y eliminación de cuentas contables. Para eliminar el registro de una cuenta contable, no debe existir transacciones realizadas.	

Tabla 2.5 Historia de Usuario Gestión de Tipos de descuento

Historia de usuario	
Código y número: HU005	**Tipo:** RF005
Nombre de Historia: Gestión de Tipos de descuento.	
Programador responsable: Jonny Bonilla.	
Descripción: El sistema debe permitir la creación, modificación y eliminación de Tipos de descuento. Para eliminar el registro de un Tipo de descuento, no debe existir transacciones realizadas.	

Tabla 2.6 Historia de Usuario Configuración de permisos

Historia de usuario	
Código y número: HU006	**Tipo:** RF006
Nombre de Historia: Gestión de permisos	
Programador responsable: Jonny Bonilla	
Descripción: El administrador del sistema puede crear, modificar y eliminar permisos de acceso al sistema.	

Tabla 2.7 Historia de Usuario Permisos por cargo

Historia de usuario	
Código y número: HU007	**Tipo:** RF007
Nombre de Historia: Gestión de permisos.	
Programador responsable: Jonny Bonilla.	
Descripción: Configurar el acceso a las diferentes opciones del sistema dependiendo del cargo que ejerce. Desplegar una lista con todos los permisos creados y otorgar el acceso a la opción mediante un *check*.	

Tabla 2.8 Historia de Usuario Asignación de Tipos de descuento

Historia de usuario	
Código y número: HU008	**Tipo:** RF008
Nombre de Historia: Asignación de Tipos de descuento.	
Programador responsable: Jonny Bonilla.	
Descripción: Permitir la asignación de tipos de descuento a cada empleado dependiendo del grupo contable al que pertenece. Los descuentos que se realizan a los empleados pueden ser fijos o mensuales.	

Tabla 2.9 Historia de Usuario Gestión de periodos

Historia de usuario	
Código y número: HU009	**Tipo:** RF009
Nombre de Historia: Gestión de periodos.	
Programador responsable: Jonny Bonilla.	
Descripción: Ingresar los periodos contables los cuales deben cumplir con el formato Año-mes. Ejemplo: 201801 corresponde al periodo enero del 2018.	

Tabla 2.10 Historia de Usuario Control de transacciones en un período cerrado

Historia de usuario	
Código y número: HU010	**Tipo:** RF010
Nombre de Historia: Control de transacciones en un período cerrado.	
Programador responsable: Jonny Bonilla.	
Descripción: Si un periodo se encuentra en estado cerrado, no debe permitir la generación de roles de pago.	

Tabla 2.11 Historia de Usuario Asignación de cuentas contables a Tipos de descuento

Historia de usuario	
Código y número: HU011	**Tipo:** RF011
Nombre de Historia: Asignación de cuentas contables a Tipos de descuento.	
Programador responsable: Jonny Bonilla.	
Descripción: Cada tipo de descuento debe estar asociado a una cuenta contable para su registro en los comprobantes. Esta asignación se la debe realizar según su grupo contable.	

Tabla 2.12 Historia de Usuario Gestión de anticipos a un empleado

Historia de usuario	
Código y número: HU012	**Tipo:** RF012
Nombre de Historia: Gestión de anticipos a un empleado.	
Programador responsable: Jonny Bonilla.	
Descripción: Registrar los anticipos solicitados por un empleado. Mostrar la lista de empleados para realizar el registro de anticipos de forma masiva.	

Tabla 2.13 Historia de Usuario Gestión de rol de pagos

Historia de usuario	
Código y número: HU013	**Tipo:** RF013
Nombre de Historia: Gestión de rol de pagos.	
Programador responsable: Jonny Bonilla.	
Descripción: Generar el rol de pagos general e individual para un periodo. Además, se debe visualizar la provisión que debe realizar la empresa para el pago de los décimos.	

Tabla 2.14 Historia de Usuario Gestión de asientos contables

Historia de usuario	
Código y número: HU014	**Tipo:** RF014
Nombre de Historia: Gestión de asientos contables.	
Programador responsable: Jonny Bonilla.	
Descripción: Generar asientos contables de los roles de pago. Los asientos contables pueden ser: • Asiento de provisión. • Asiento correspondiente al rol de pagos general.	

Tabla 2.15 Historia de Usuario Envío de notificaciones

Historia de usuario	
Código y número: HU015	**Tipo:** RF015
Nombre de Historia: Envío de notificaciones.	
Programador responsable: Jonny Bonilla.	
Descripción: Enviar notificaciones al dispositivo móvil con sistema operativo *Android.*	

Tabla 2.16 Historia de Usuario Impresión de Roles de pago individuales

Historia de usuario	
Código y número: HU016	**Tipo:** RF016
Nombre de Historia: Impresión de Roles de pago individuales.	
Programador responsable: Jonny Bonilla.	
Descripción: Imprimir el rol de pagos individual con una copia.	

Tabla 2.17 Historia de Usuario Impresión del Rol General

Historia de usuario	
Código y número: HU017	**Tipo:** RF017
Nombre de Historia: Impresión del Rol General.	
Programador responsable: Jonny Bonilla.	
Descripción: Imprimir el rol de pagos general del periodo seleccionado. Este documento debe tener la siguiente información: • Lista de empleados ordenados de manera alfabética. • Lista de los ingresos. • Lista de egresos. • Total general.	

Tabla 2.18 Historia de Usuario Impresión de ficha del empleado

Historia de usuario	
Código y número: HU018	**Tipo:** RF018
Nombre de Historia: Impresión de ficha del empleado.	
Programador responsable: Jonny Bonilla.	
Descripción: Imprimir la ficha del empleado con toda la información proporcionada en el formulario de registro.	

Tabla 2.19 Historia de Usuario Reporte listado de empleados

Historia de usuario	
Código y número: HU019	**Tipo:** RF019
Nombre de Historia: Reporte listado de empleados.	
Programador responsable: Jonny Bonilla.	
Descripción: Imprimir un listado de empleados de la empresa.	

Tabla 2.20 Historia de Usuario Reporte listado de cargos

Historia de usuario	
Código y número: HU020	**Tipo:** RF020
Nombre de Historia: Reporte listado de cargos.	
Programador responsable: Jonny Bonilla.	
Descripción: Imprimir un listado de cargos de la empresa ordenados de manera alfabética de forma ascendente.	

Tabla 2.21 Historia de Usuario Reporte listado de departamentos

Historia de usuario	
Código y número: HU021	**Tipo:** RF021
Nombre de Historia: Reporte listado de departamentos.	
Programador responsable: Jonny Bonilla.	
Descripción: Imprimir un listado de departamentos de la empresa ordenados de manera alfabética de forma ascendente.	

Tabla 2.22 Historia de Usuario Reporte listado de cuentas contables

Historia de usuario	
Código y número: HU022	**Tipo:** RF022
Nombre de Historia: Reporte listado de cuentas contables.	
Programador responsable: Jonny Bonilla.	
Descripción: Imprimir un listado de cuentas contables de la empresa ordenados de manera ascendente y agrupados según su cuenta padre. Es indispensable distinguir si la cuenta es de movimiento o de orden para que se incluya en el balance de la empresa.	

Tabla 2.23 Historia de Usuario Autenticación de usuarios

Historia de usuario	
Código y número: HU023	**Tipo:** RNF001
Nombre de Historia: Autenticación de usuarios.	
Programador responsable: Jonny Bonilla.	
Descripción: Realizar la autenticación y acceso a la aplicación mediante usuario y contraseña.	
Observaciones: Preferentemente el código de usuario debe ser su número de cédula.	

Tabla 2.24 Historia de Usuario Seguridad en la comunicación

Historia de usuario	
Código y número: HU024	**Tipo:** RNF002
Nombre de Historia: Seguridad en la comunicación.	
Programador responsable: Jonny Bonilla.	
Descripción: La información debe ser transportada de tal manera que no pueda ser visualizada mediante una herramienta que capture y analice paquetes que viajan por la red.	

Tabla 2.25 Historia de Usuario Acceso multiusuario

Historia de usuario	
Código y número: HU025	**Tipo:** RNF003
Nombre de Historia: Acceso multiusuario.	
Programador responsable: Jonny Bonilla.	
Descripción: Permitir el acceso de varios usuarios a la vez.	

Tabla 2.26 Historia de Usuario Interfaz amigable

Historia de usuario	
Código y número: HU026	**Tipo:** RNF004
Nombre de Historia: Interfaz amigable.	
Programador responsable: Jonny Bonilla.	
Descripción: Los usuarios deben contar con una interfaz amigable y fácil de usar.	

En la Tabla 2.27 se muestran los requerimientos no funcionales que se identificaron para el presente Proyecto de Titulación. Los requerimientos no funcionales tienen que ver con la calidad del software a realizar, además establecen restricciones en el diseño y la implementación [23].

Tabla 2.27 Lista de requerimientos no funcionales.

Requerimientos no funcionales			
	Historia de Usuario	**Requerimiento no Funcional**	**Descripción**
Autenticación	HU023	RNF001	Realizar la autenticación y acceso a la aplicación mediante usuario y contraseña.
Seguridad en la comunicación	HU024	RNF002	La información debe ser transportada de tal manera que no pueda ser visualizada mediante una herramienta que capture y analice paquetes que viajan en la red.
Desempeño	HU025	RNF003	Permitir el acceso de varios usuarios a la vez.
Simplicidad	HU026	RNF004	Los usuarios deben contar con una interfaz amigable y fácil de usar.

En la Tabla 2.28 se muestra un resumen de los requerimientos funcionales [24], donde se visualiza los campos número de historia de usuario, numero de requerimiento funcional y la descripción. Los títulos HU y RF corresponden a '*Historias de usuario'* y '*Requerimientos funcionales*' respectivamente.

Tabla 2.28 Lista de requerimientos funcionales.

HU	RF	Descripción
HU001	RF001	Ingresar, modificar y eliminar la ficha del empleado.
HU002	RF002	El sistema debe permitir la creación, modificación y eliminación de cargos. Para eliminar el registro de un cargo, no debe existir transacciones realizadas.
HU003	RF003	El sistema debe permitir la creación, modificación y eliminación de departamentos. Para eliminar el registro de un departamento, no debe existir transacciones realizadas.
HU004	RF004	El sistema debe permitir la creación, modificación y eliminación de cuentas contables. Para eliminar el registro de una cuenta contable, no debe existir transacciones realizadas.
HU005	RF005	El sistema debe permitir la creación, modificación y eliminación de Tipos de descuento. Para eliminar el registro de un Tipo de descuento, no debe existir transacciones realizadas.
HU006	RF006	El administrador del sistema puede crear, modificar y eliminar permisos de acceso al sistema.
HU007	RF007	Configurar el acceso a las diferentes opciones del sistema dependiendo del cargo que ejerce.
HU008	RF008	Permitir la asignación de tipos de descuento a cada empleado dependiendo del grupo contable al que pertenece.
HU009	RF009	Ingresar los periodos contables los cuales deben cumplir con el formato Año-mes.
HU010	RF010	Si un periodo se encuentra en estado cerrado, no debe permitir la generación de roles de pago.
HU011	RF011	Cada tipo de descuento debe estar asociado a una cuenta contable para su registro en los comprobantes. Esta asignación se la debe realizar según su grupo contable.
HU012	RF012	Registrar los anticipos solicitados por un empleado.
HU013	RF013	Generar el rol de pagos general e individual para un periodo.
HU014	RF014	Generar asientos contables de los roles de pago.
HU015	RF015	Enviar notificaciones al dispositivo móvil con sistema operativo *Android.*
HU016	RF016	Imprimir el rol de pagos individual con una copia.
HU017	RF017	Imprimir el rol de pagos general del periodo seleccionado.
HU018	RF018	Imprimir la ficha del empleado con toda la información proporcionada en el formulario de ingreso de información.
HU019	RF019	Imprimir un listado de empleados de la empresa.
HU020	RF020	Imprimir un listado de cargos de la empresa ordenados de manera alfabética de forma ascendente.
HU021	RF021	Imprimir un listado de departamentos de la empresa ordenados de manera alfabética de forma ascendente.
HU022	RF022	Imprimir un listado de cuentas contables de la empresa ordenados de manera ascendente y agrupados según su cuenta padre.

2.1.2. DEFINICIÓN DEL *PRODUCT BACKLOG*

Siguiendo las recomendaciones de la metodología *Scrum*, es necesario definir el documento denominado *Product Backlog,* en el cual constarán todos los requerimientos

que el cliente espera por parte del software a diseñarse. Antes de realizar esta tarea, es necesario definir los roles que cumplirán las personas que intervienen en el sistema, ya que ellos son los que definen y validan los requerimientos que formarán parte del *Product Backlog.*

- **Dueño del producto:** Persona encargada de generar los roles de pago, en este caso son las personas encargadas de generar el rol de pagos y fueron encuestadas.
- **Director de *Scrum*:** Ana María Zambrano PhD.
- **Desarrollador:** Jonny Bonilla

En la Tabla 2.29 se muestra la valoración de los requerimientos que formarán parte del *Product Backlog***,** ya que *Scrum* recomienda establecer la prioridad y complejidad que tiene cada requerimiento**.**

Tabla 2.29 Valoración de los requerimientos.

Valor	Prioridad (P)	Complejidad (C)
1	Baja	Sencilla
2	Media	Media
3	Alta	Compleja
4	Muy Alta	Muy compleja

Una vez definido los roles y establecido las valoraciones de los requerimientos, se procede a definir el *Product Backlog* con los requerimientos especificados en términos generales, los mismos que fueron detallados en las historias de usuario del Apartado 2.1.1. En la Tabla 2.30 se muestra la definición del *Product Backlog* del sistema valorado según su prioridad, complejidad para ser ejecutado y el tiempo estimado en horas que requiere su culminación. En la Tabla 2.30 se definen las siguientes columnas:

- ID: número de identificación único.
- Requerimiento: nombre del requerimiento.
- RF: requerimiento funcional asociado a la tarea.
- P: prioridad definido según la Tabla 2.29.
- C: complejidad definido según la Tabla 2.29.
- T: tiempo en horas necesario para cumplir el requerimiento.

Tabla 2.30 Definición del *Product Backlog*.

ID	Requerimiento	RF	P	C	T (Horas)
01	Realizar el diagrama de componentes del sistema.	RF001- RF 023	4	4	8
02	Diseño de base de datos.	RF 001-RF 023	4	4	20
03	Diseño de lógica del negocio.	RF001- RF016	4	4	20
04	Diseño de la interfaz de Usuario.	RF001- RF016	4	4	20
05	Diseño del aplicativo móvil.	RF017	4	4	20
06	Instalación de las herramientas para el desarrollo.	RF001- RF016	3	3	4
07	Implementación de la base de datos.	RF001- RF016	4	4	8
08	Implementación de la lógica de negocio.	RF001- RF016	4	4	100
09	Implementación de la presentación.	RF001- RF023	4	4	50
10	Instalación y configuración del servidor *Openfire.*	RF017	4	4	2
11	Implementación del aplicativo móvil.	RF017	4	4	50
12	Pruebas de funcionamiento.	RF001- RF023	3	2	100
			Total		**402**

Una vez definido el *Product Backlog* es necesario establecer los *Sprints* en los cuales se realizarán las tareas necesarias para cumplir con los requerimientos del software.

Para definir los *Sprints* se consideró las siguientes etapas:

- Recolección de información.
- Análisis de los requerimientos.
- Diseño del sistema.
- Instalación de Herramientas necesarias.
- Implementación del modelo planteado.
- Pruebas de funcionamiento.

En la Tabla 2.31 se puede visualizar la definición de los *Sprints* necesarios para la implementación del sistema.

Tabla 2.31 Definición de *Sprints* del sistema SRSOFT.

No.	Nombre	Tareas
1	*Sprint* Elaboración del Diagrama de Componentes.	Elaborar el diagrama de componentes.
2	*Sprint* Flujo de Información.	Elaborar el diagrama de secuencia.
3	*Sprint* Diseño de la Capa de Datos.	Elaborar el diagrama entidad-relación.
4	*Sprint* Diseño de la Lógica de Negocio.	Elaborar el diagrama de clases.
5	*Sprint* Diseño de la Capa de Presentación.	Diseñar los formularios Windows para la presentación. Elaborar el diseño de los reportes.
6	*Sprint* Diseño del Aplicativo Móvil.	Elaborar el diseño de los *layout*. Elaborar el diseño de las notificaciones.
7	*Sprint* Instalación de Herramientas.	Instalar los programas: *SQL-Server Express Edition* 2017, *Visual Studio Expres* 2017, *Xamarin*, *Openfire*, *Crystal Report*
8	*Sprint* Implementación de la Capa de Datos.	Codificación de la base de datos.
9	*Sprint* Implementación de la Lógica de Negocio.	Codificación de las clases con sus respectivos atributos, propiedades, constructores y métodos.
10	*Sprint* implementación de la capa presentación.	Codificación de los formularios Windows. Implementación de los reportes.
11	*Sprint* Implementación del Aplicativo Móvil.	Codificación de los layout. Implementación de las notificaciones.
12	*Sprint* Pruebas del sistema.	Pruebas de los requerimientos establecidos.

2.1.3. *SPRINT* ELABORACIÓN DEL DIAGRAMA DE COMPONENTES.

Siguiendo el lineamiento del *Product Backlog* establecido anteriormente, la primera tarea es realizar un diagrama de componentes del sistema con el objetivo de tener una idea general del funcionamiento y como se relacionan los diferentes componentes entre sí.

En la Figura 2.1 se muestra los componentes del sistema SRSOFT detallado desde la interfaz que utilizará la persona encargada de generar los roles de pago, pasando por el

servidor que dispone de la base de datos y la conexión con *Openfire* mediante el cual se enviará las notificaciones al dispositivo móvil.

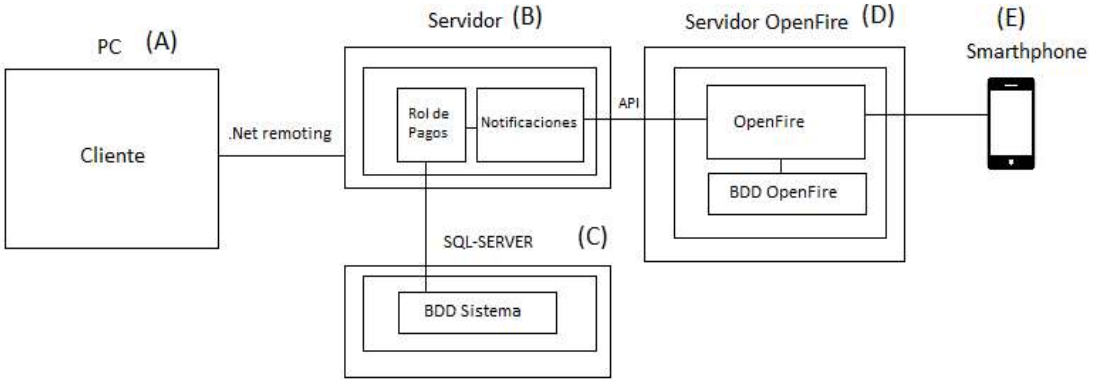

Figura 2.1 Componentes del sistema SRSOFT.

- **Componente (A):** Cliente que utilizarán las personas encargadas de generar los roles de pago y permitirá el ingreso de información, consultas y generación de reportes a través del servidor (componente B). La interacción entre el componente (A) y el componente (B) se lo realizará utilizando .*Net Remoting*.
- **Componente (B):** Servidor que dispondrá de todos los procesos para la generación de roles de pago. Interactuará con el gestor base de datos (componente C), además mediante una *API* podrá enviar las notificaciones a través del servidor *Openfire* hacia los *smarthphone* (componente E).
- **Componente (C):** Gestor de base de datos que almacenará la información.
- **Componente (D):** Servidor *Openfire* que recibirá las notificaciones del servidor y las enviará a los *smarthphone* mediante el protocolo *XMPP*.
- **Componente (E):** Aplicación móvil que correrá sobre un *smarthphone* que permitirá al usuario conocer sobre su rol de pagos. Este aplicativo será desarrollado para sistemas operativos *Android*.

2.1.4. *SPRINT* FLUJO DE INFORMACIÓN

Previo al diseño de las capas que conforman el sistema, se va a realizar el *Sprint* flujo de información en el cual mediante diagramas de secuencia se muestra la forma en que los componentes interactúan entre sí.

a) Diagramas de secuencia

A continuación, se muestra los diagramas de secuencia en los cuales se define la interacción entre los componentes A, B, C, D y E del sistema en función del tiempo [25].

Es importante definir los actores que participan en el sistema, los cuales son:

- Usuario: Persona encargada de generar los roles de pago en la empresa que generalmente es el contador y tendrá acceso al aplicativo cliente o componente A.
- Empleado: Persona que presta sus servicios a la empresa y recibirá el detalle de su rol en el dispositivo móvil o componente E.

En la Figura 2.2 se muestra el diagrama de secuencia para el almacenamiento y consulta de información en la base de datos. Aquí se puede resaltar cómo el aplicativo cliente y servidor interactúan o se comunican utilizando *.Net Remoting.* Este diagrama brinda la solución planteada para los requerimientos RF001, RF002, RF003, RF004, RF005, RF006, RF008, RF009, RF010, RF011 y RF012.

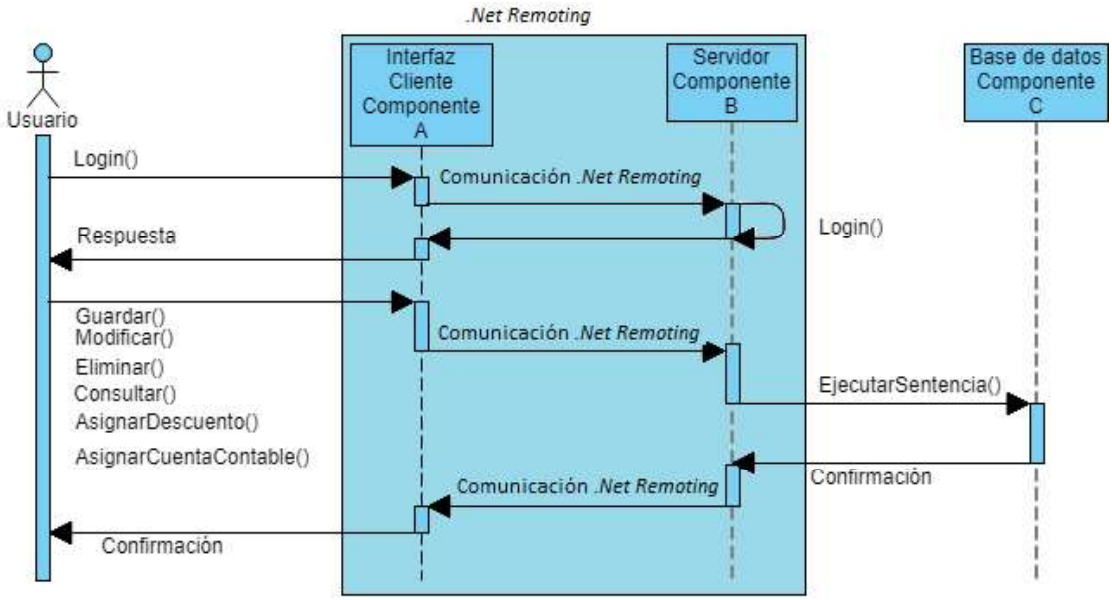

Figura 2.2 Diagrama de secuencia para almacenamiento y consulta de datos.

En la Figura 2.3 y Figura 2.4 se muestran los diagramas de secuencia en los cuales se visualiza la interacción de los diferentes componentes para la generación del rol de pagos y el envío de notificaciones a los dispositivos móviles. Este diagrama muestra la solución para los requerimientos RF013 y RF015.

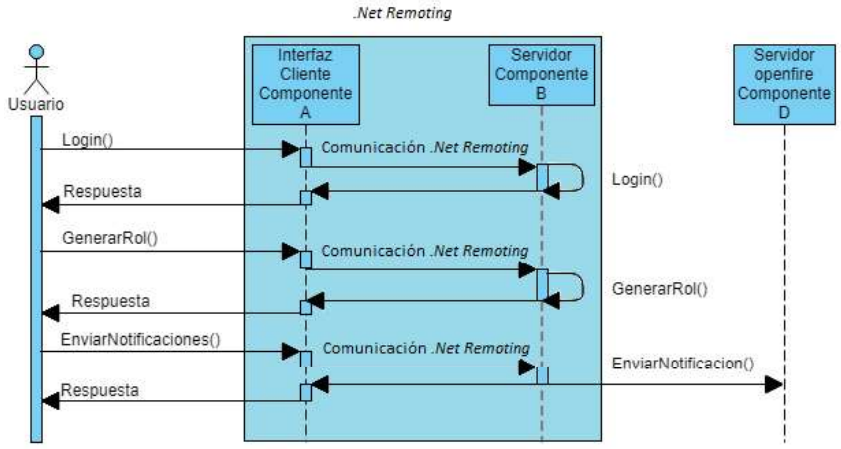

Figura 2.3 Diagrama de secuencia para envío de notificaciones a *Openfire*.

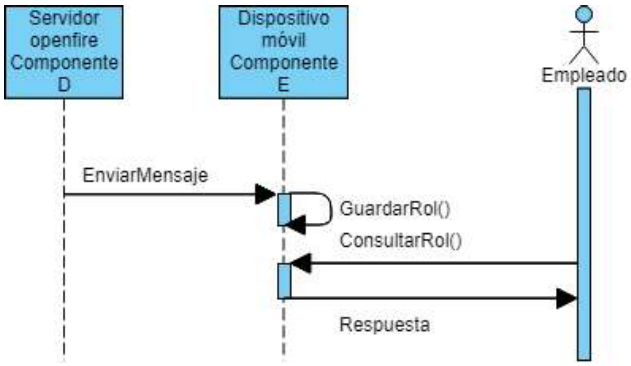

Figura 2.4 Diagrama de secuencia para el envío de mensajes al dispositivo móvil

En la Figura 2.5 se muestra el diagrama de secuencia para la generación de comprobantes contables, el cual es la solución planteada para el requerimiento RF014.

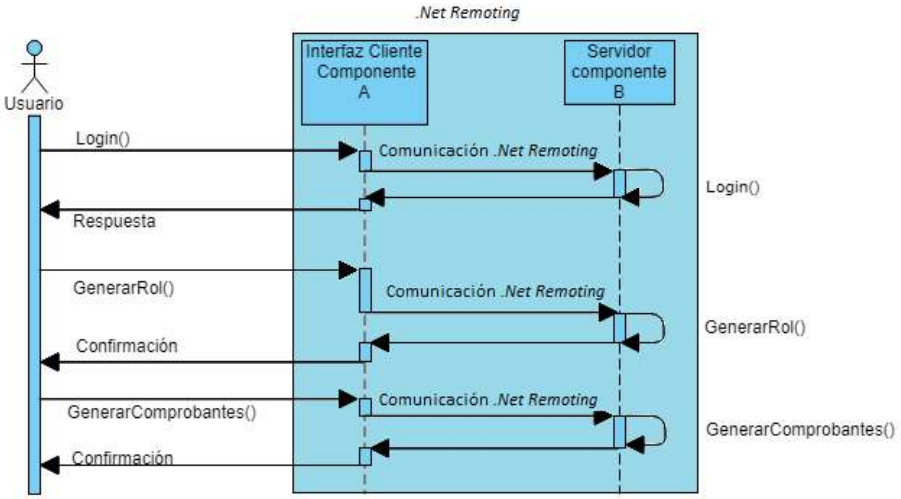

Figura 2.5 Diagrama de secuencia para generar comprobantes contables.

En la Figura 2.6 se muestra el diagrama de secuencia para la impresión de reportes, diseño que brinda solución a los requerimientos RF016, RF017, RF018, RF019, RF020, RF021 y RF022.

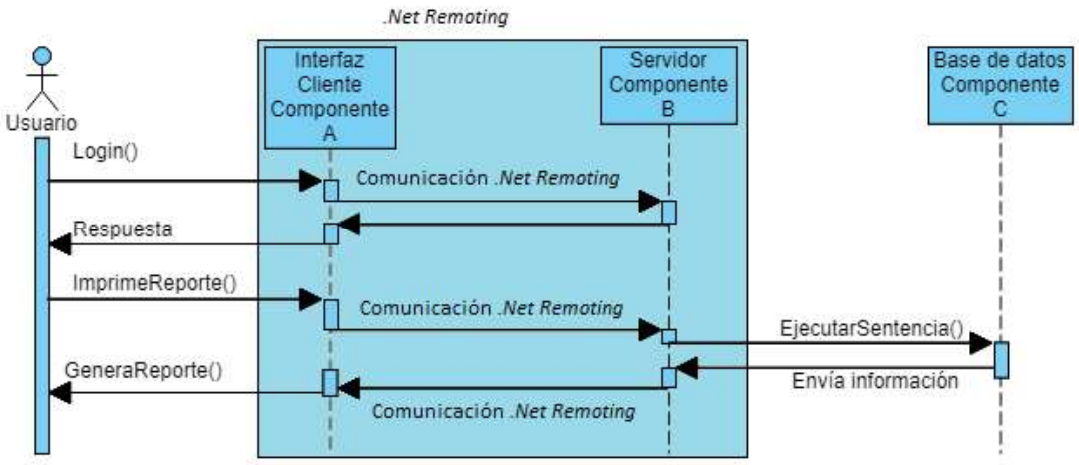

Figura 2.6 Diagrama de secuencia para impresión de reportes.

2.1.5. *SPRINT* DE DISEÑO DE LA CAPA BASE DE DATOS

Para el diseño de la base de datos se realizó un diagrama entidad-relación donde constan las tablas o entidades con sus campos respectivos, tomando en consideración las relaciones que existen entre ellas.

a) Diagrama entidad-relación

En la Figura 2.7 se puede ver el diagrama entidad-relación completo [26]. En el diagrama se puede observar cómo la tabla **Empleado** se relaciona con las tablas **Cargo**, **Departamento**, **Banco**, **NivelEstudio** y **AsientoRol**. Es importante mencionar que un **Empleado** ejerce un solo **Cargo** en un **Departamento**; además se registrará solamente una cuenta bancaria en la cual se le realizará el depósito de su sueldo. Para eso es necesario contar con la información de la entidad bancaria que se encuentra representado por la tabla **Banco**. Por otra parte, dentro de una empresa, el registro contable se lo realiza según su grupo o **Cuenta contable**, es por eso que se realiza la relación con la tabla **AsientoRol** para establecer la cardinalidad entre el **Empleado** y la **Cuenta contable**. Además, a un **Empleado** se le debe asignar los tipos de descuento establecidos por la ley o las políticas de la empresa, los mismos que se encuentran representados por la tabla **TipoDescuento**. Una vez establecido la relación entre estas tablas, se debe incluir la tabla **Periodo** la cual representa el ciclo en el cual se genera los roles de pago. Por último, se tiene a los **Comprobantes contables** los cuales están conformados por un grupo de

Cuentas contables que se asignaron anteriormente a los empleados en sus respectivos grupos. La definición de cada una de las tablas se presenta en el **Anexo C**.

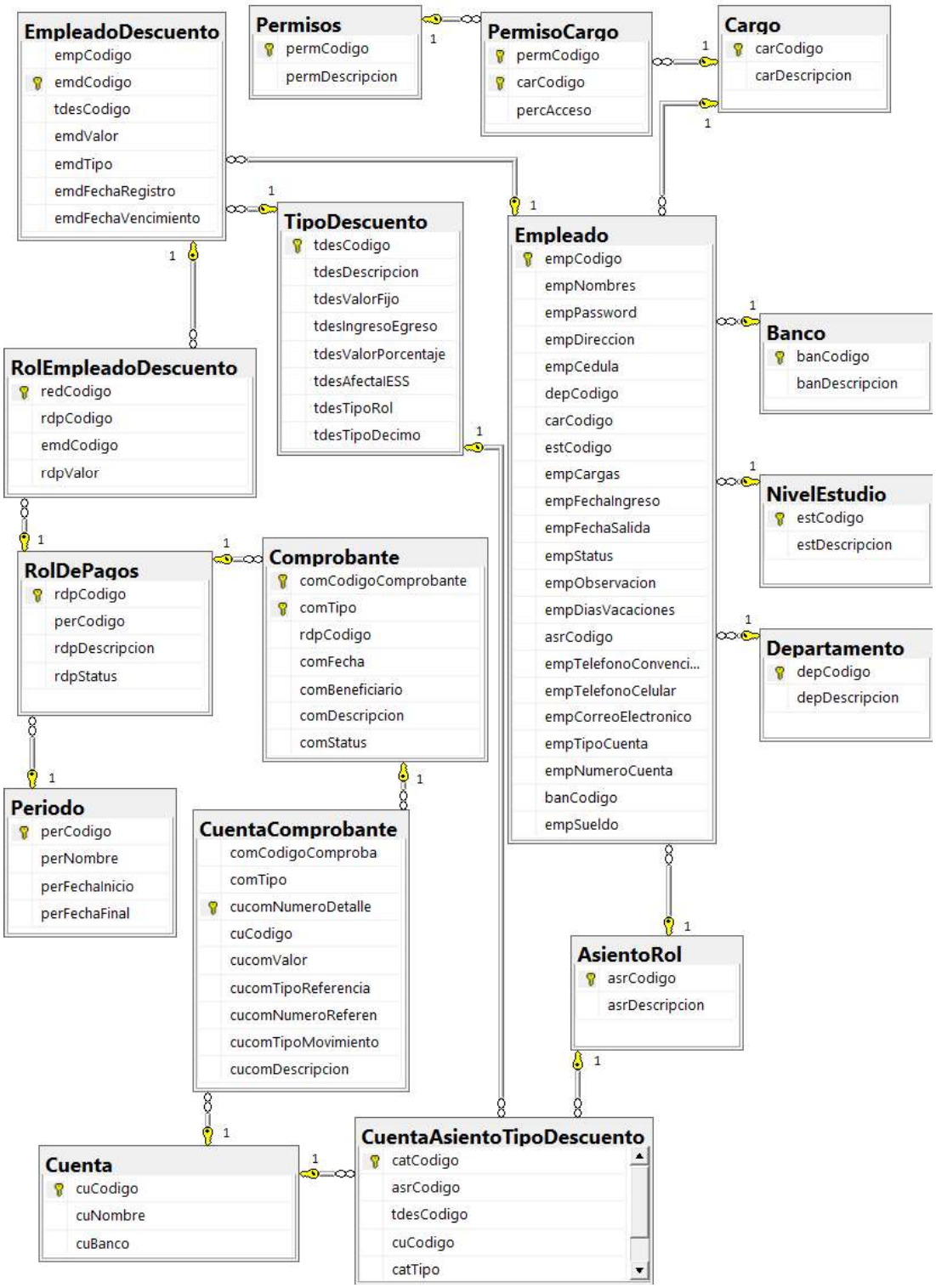

Figura 2.7 Diagrama Entidad-Relación de SRSOFT.

2.1.6. *SPRINT* DE DISEÑO DE LA CAPA DE NEGOCIO

En la Capa de Negocio se debe diseñar y detallar un diagrama de clases en el cual se puede visualizar de mejor manera sus respectivos atributos y métodos [27]. El diagrama de clases presenta la estructura del sistema de manera gráfica basado en el modelo *UML*. Como ejemplo, en la Figura 2.8 muestra el detalle de la clase llamada **BDD**, la cual interactúa directamente con el servidor *SQL Server* mediante sentencias *SQL*. Dicha clase dispone de atributos estáticos necesarios para estructurar la cadena de conexión hacia la base de datos para interactuar con ella. Todo el resto de clases interactuarán con la clase **BDD** para enviar o consultar información, es decir el resto de clases es dependiente de esta clase.

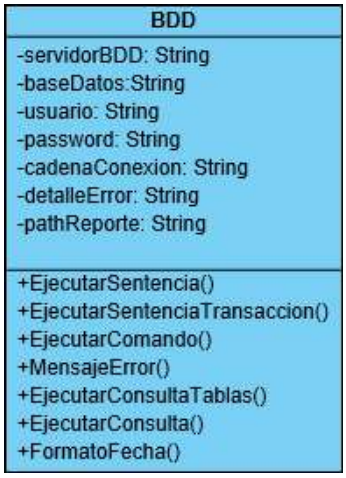

Figura 2.8 Clase BDD para interactuar con la base de datos.

En la Figura 2.9 se muestra el detalle de la Capa de Negocio, en dicho diagrama están todas las clases identificadas para el funcionamiento del sistema. Además, se puede visualizar los atributos con su respectivo tipo de dato y los métodos de cada una de ellas.

Por ejemplo, se puede visualizar que un **Empleado** ejerce un **Cargo** en la empresa; también dicho **Empleado** pertenece a un **Departamento** definido y además se registrará su **Nivel de Estudios**. Adicionalmente, un empleado debe proporcionar la información de su cuenta bancaria para que la empresa pueda realizar el depósito de su sueldo. Se debe resaltar que, para realizar el registro contable de un rol de pagos, el empleado debe pertenecer a un grupo de **Asientos** con el fin de que se refleje estas transacciones en la contabilidad de la empresa. Estos **Asientos Contables** dependen del **Periodo** contable que se encuentre vigente. Dentro de un periodo contable se generará el **Rol de Pagos** y este registrará los **Comprobantes** o **Asientos**, también enviará las notificaciones utilizando el servidor *Openfire* hacia los dispositivos móviles de los empleados.

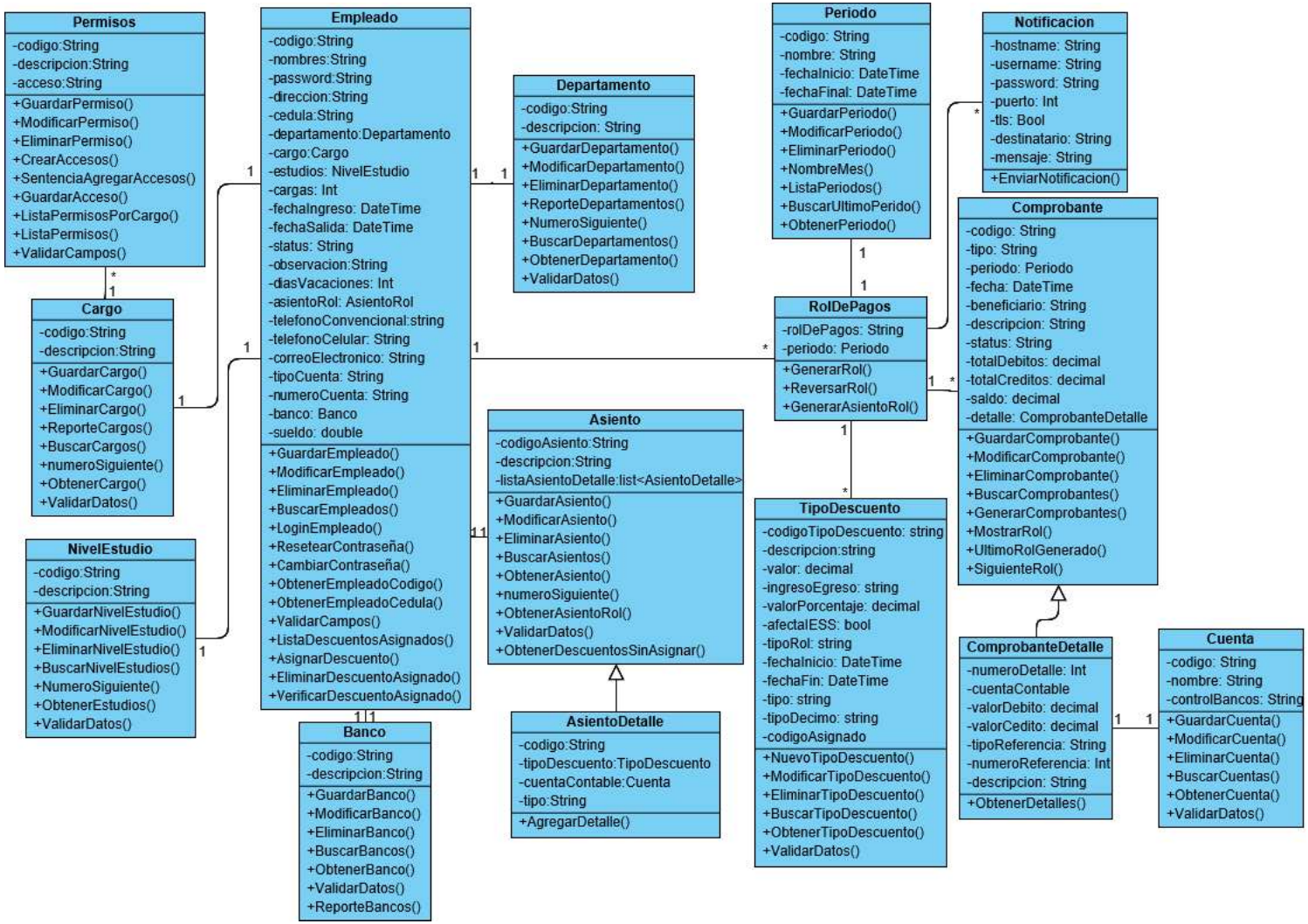

Figura 2.9 Diagrama de clases de la Capa de Negocio.

Respecto al diseño de las notificaciones, se puede destacar que mediante la clase **Notificacion** se enviará un *String* como mensaje a los dispositivos móviles de los empleados a través de *Openfire* con los datos de la cabecera y el detalle del rol de pagos separados por caracteres delimitadores como se puede observar a continuación:

Estructura del diseño:

codigoRol|descripcion|nombreEmpleado|cedula|cargo|departamento;idconcepto1|concepto1| tipo1|valor1$ idconcepto2|concepto2| tipo2|valor2

Ejemplo del mensaje:

201801|Enero 2018|Diana García|1721106894|ESPECIALISTA DE DESARROLLO| SISTEMAS;010-Sueldo|I|800$090-Alimentacion|I|70$110-Transporte|I|20$510-9.45% Aporte Personal|E|72

La barra "|" es el separador de datos de la cabecera, el caracter ";" nos indica el inicio de los detalles y el signo "$" es el separador de datos del detalle. Este mensaje será enviado al servidor *Openfire* con la respectiva información del destinatario para que sea entregado al dispositivo móvil del empleado y en base a los caracteres delimitadores se pueda estructurar la presentación del rol de pagos.

2.1.7. *SPRINT* DE DISEÑO DE LA CAPA DE PRESENTACIÓN

Continuando con el diseño del sistema, se realizará el bosquejo de los formularios que se presentará al usuario para poder ingresar, modificar o eliminar información; además debe disponer de formularios en los cuales pueda generar los roles de pago, imprimir los reportes y enviar las notificaciones. El primer formulario que se le presentará al usuario será el de autenticación (Requerimiento RNF001), en el cual debe ingresar sus credenciales para acceder al sistema. En la Figura 2.10 se muestra la plantilla del formulario de autenticación.

Figura 2.10 Plantilla del formulario de autenticación.

Una vez que el usuario sea autenticado de manera satisfactoria, se le presentará un formulario con las opciones principales del sistema. Este formulario presentará las opciones Rol de pagos, Reportes y Configuración.

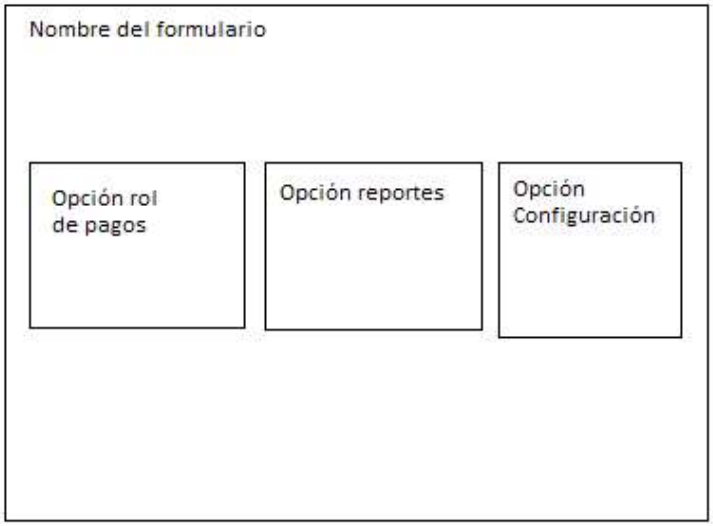

Figura 2.11 Plantilla del formulario de opciones principales.

En la opción de Rol de pago podrá ingresar en la lista de catálogos, asignar los tipos de descuento a los empleados, asignar las cuentas contables a los tipos de descuento, registrar los anticipos, generar y reversar roles de pago, visualizar los roles generados y consultar los comprobantes contables. En la Figura 2.12 se muestra lo expuesto anteriormente.

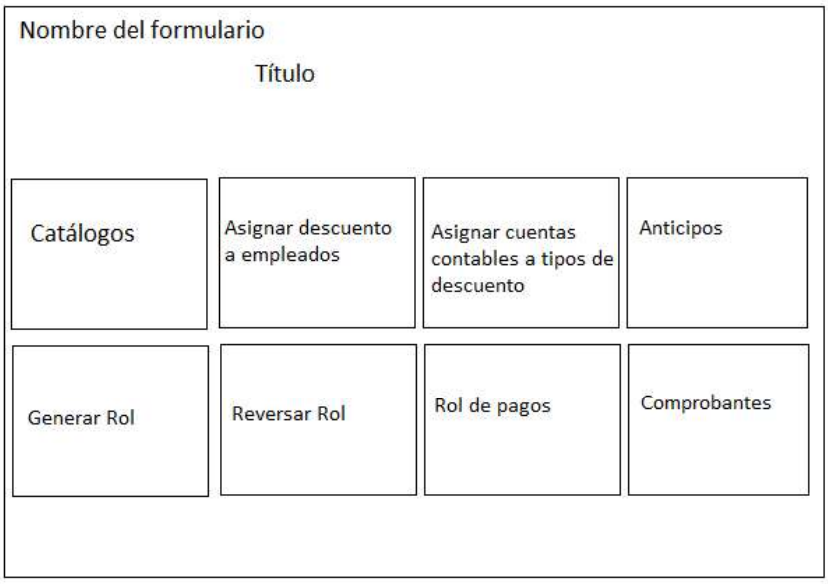

Figura 2.12 Plantilla del formulario de opción de roles de pago.

Al presionar el botón de catálogos el usuario podrá ingresar la información de empleados, cargos, departamentos, asientos contables, periodos, tipos de descuento, cuentas contables, entidades bancarias y grupos de asientos. Los catálogos se basarán en la plantilla que se muestra en la Figura 2.13. Utilizando esta plantilla se brindará solución a los requerimientos funcionales RF001, RF002, RF003, RF004, RF005, RF006, RF007, RF009 y RF012.

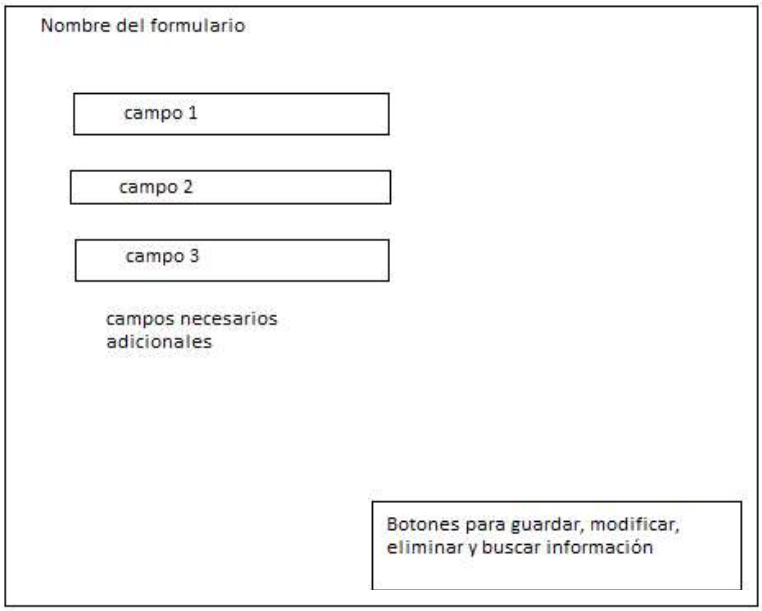

Figura 2.13 Plantilla para los catálogos.

Además, los formularios Generar rol y Reversar rol se basarán en la plantilla mostrada en la Figura 2.14, el cual brinda solución a los requerimientos RF010 y RF013.

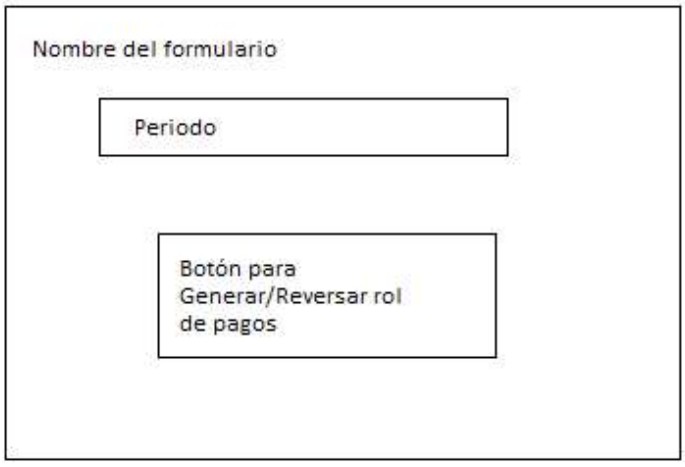

Figura 2.14 Plantilla para generar o reversar rol.

Una vez generado el rol de pagos el usuario dispondrá de la opción Ver rol de pagos en el cual se visualizará el resultado. En la Figura 2.15 se muestra la plantilla descrita anteriormente, la cual fue diseñada para dar solución a los requerimientos RF013, RF014 y RF015.

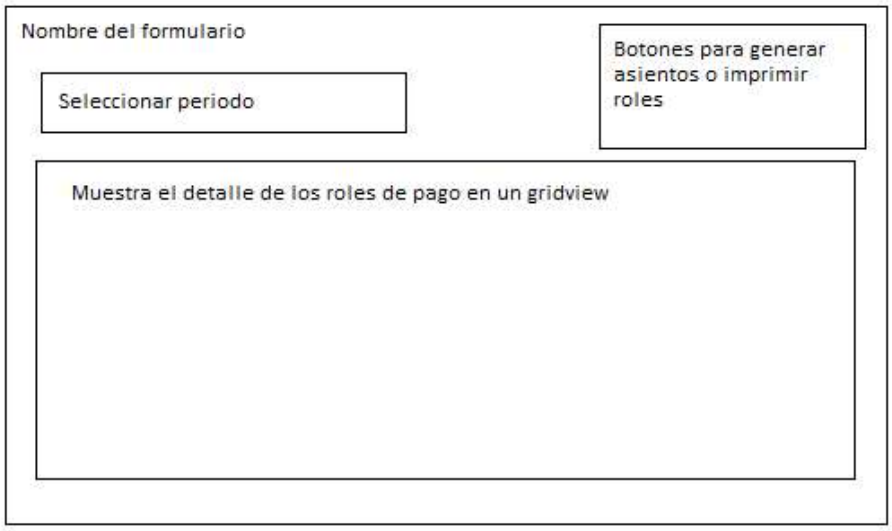

Figura 2.15 Plantilla para visualizar un rol de pagos.

Por último, la opción de Rol de pagos presentará las opciones para asignar tipos de descuento a los empleados y asignar cuentas contables a dichos tipos de descuento. En la Figura 2.16 se encuentra definido la plantilla para la asignación ya sea de tipos de descuento o cuentas contables según corresponda, este diseño brindará solución a los requerimientos RF008 y RF011.

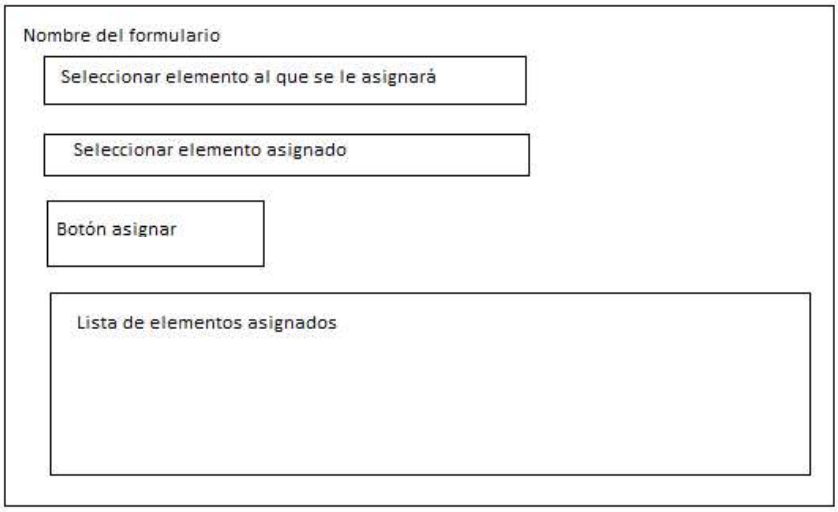

Figura 2.16 Plantilla para asignar elementos.

Anteriormente se mencionó que el menú principal dispone de la opción Reporte; donde se podrá seleccionar: lista de empleados, lista de cargos, lista de departamentos, lista de cuentas contables, lista de periodos contables, lista de tipos de descuento, entidades bancarias y lista de niveles de estudio, los cuales se presentarán en el visor por defecto de *Crystal Report* incluido en *Visual Studio.* En la Figura 2.17 se presenta la plantilla general que se utilizará para un reporte, diseño que brindará solución a los requerimientos RF016, RF017, RF018, RF019, RF020, RF021 y RF022.

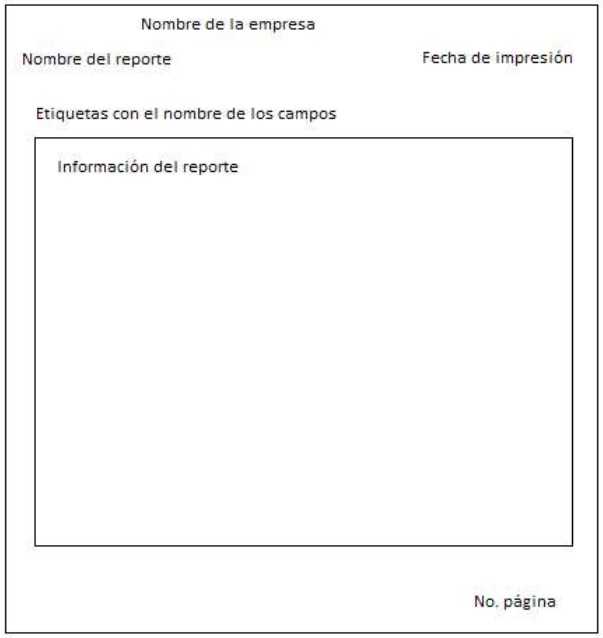

Figura 2.17 Plantilla general para los reportes.

Por último, en la opción de configuración se podrá resetear la contraseña de un usuario, se podrá asignar los permisos según su cargo y se visualizará la lista de opciones que dispone el sistema.

2.1.8. *SPRINT* DE DISEÑO DEL APLICATIVO MÓVIL

Para este *Sprint*, se debe realizar el diseño del aplicativo móvil para cumplir el requerimiento RF015, el cual se encargará de recibir las notificaciones enviadas por el servidor *Openfire* y lo mostrará al empleado. Es importante recordar que *Openfire* se encarga de entregar el mensaje en tiempo real pero no lo guarda, es por eso que se realizará el diseño de una base de datos para el móvil, ya que estas notificaciones deben ser almacenas en el teléfono con el propósito que el usuario pueda revisar sus roles en el momento que desee, adicionalmente se necesita almacenar la configuración para poder

conectarse al servidor *Openfire*. También es importante mencionar que dispondrá de sus propias clases con sus respectivos métodos. Por último, se debe diseñar las vistas que presentarán la información al usuario.

a) Diagrama entidad-relación del aplicativo móvil

El aplicativo móvil en su base de datos dispondrá de tres tablas: una tabla que almacenará la configuración para conectarse al servidor *Openfire,* una tabla para almacenar la cabecera del rol de pagos y una tabla con la información de su detalle.

En la Figura 2.18 se puede visualizar el diagrama entidad-relación de la base de datos del aplicativo móvil.

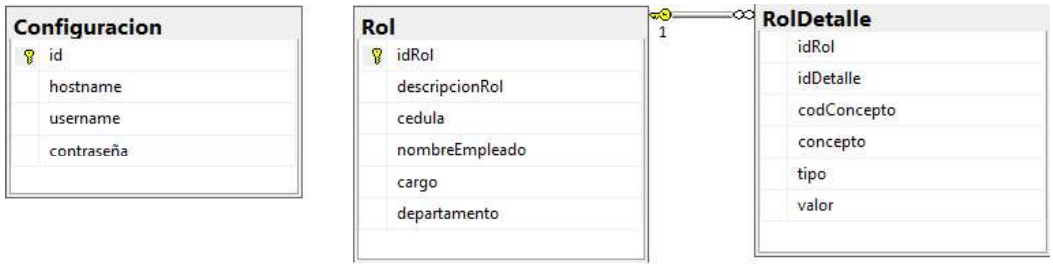

Figura 2.18 Diagrama entidad-relación del aplicativo móvil.

b) Diagrama de clases del aplicativo móvil

En la Figura 2.19 se puede visualizar el diagrama de clases que se utilizará en el aplicativo móvil. Este modelo se lo realiza tomando en consideración que los recursos que dispone el móvil son más limitados que un computador, es por eso que se selecciona cierta información del modelo de clases anterior para ser implementado en el *smarthphone.*

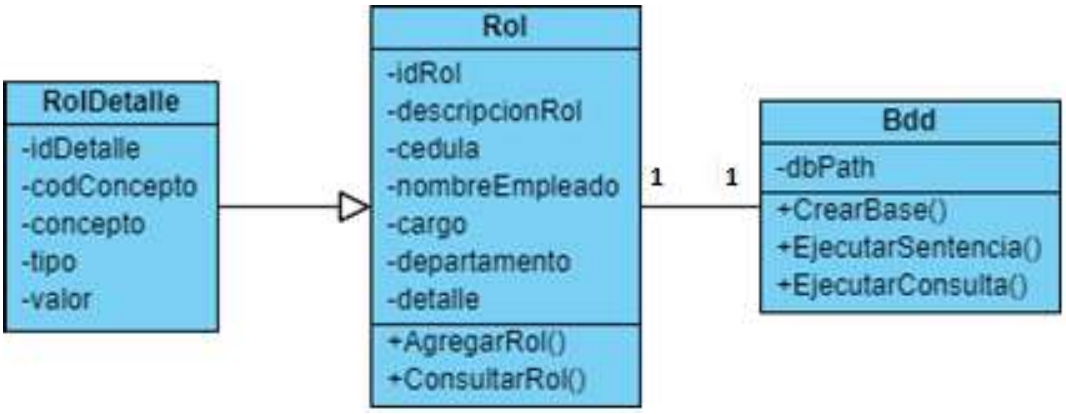

Figura 2.19 Diagrama de clases del aplicativo móvil.

c) Interfaz gráfica del aplicativo móvil

Continuando con el diseño, el aplicativo móvil presentará una pantalla con el menú principal en el cual podrá seleccionar las opciones Configuración y Roles de pago como se puede ver en la Figura 2.20.

Figura 2.20 Pantalla Menú principal del aplicativo móvil.

Si ingresa en la opción de configuración se mostrará una pantalla para ingresar los datos *hostname*, *username* y contraseña, esta información será utilizada para acceder al servidor *Openfire*. En la Figura 2.21 se muestra la plantilla para la pantalla de configuración.

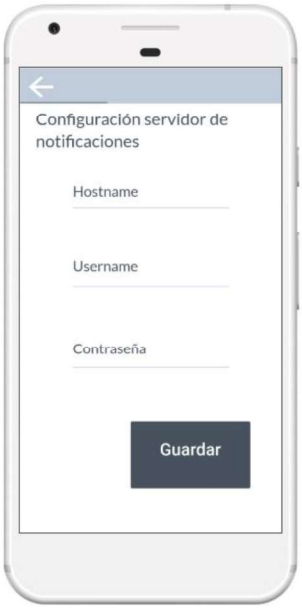

Figura 2.21 Pantalla configuración para conexión a *Openfire.*

Al contrario, si el usuario ingresa en la opción Roles de pago del menú principal, se presentará una pantalla con la lista de roles de pago recibidos. Es importante aclarar que solo se mostrará el nombre del periodo ya que existirá otra pantalla en la cual se presentará de manera detallada los datos del rol de pagos seleccionado. En la Figura 2.22 se muestra la plantilla que se utilizará para mostrar la lista de roles de pago.

Figura 2.22 Pantalla con la lista de roles recibidos.

Por otra parte, en la Figura 2.23 se muestra la plantilla para la pantalla que mostrará el detalle del rol de pagos.

Figura 2.23 Pantalla con el detalle del rol de pagos.

2.2. IMPLEMENTACIÓN

En esta sección se detallará en primera instancia, las herramientas necesarias para la implementación del sistema. Se mostrará los pasos que se deben ejecutar para la instalación y configuración del gestor de base de datos que para este Proyecto de Titulación será *SQL Server 2017 Express Edition*. También se instalará *Visual Studio Comunity* 2017 mediante el cual se realizará la implementación de la capa de negocio, *.Net remoting* y los formularios que se utilizarán en la presentación. Adicionalmente se instalará *Xamarin*, una extensión disponible en *Visual Studio* la cual nos permite implementar aplicativos móviles. A continuación, se instalará y configurará el servidor *Openfire* para incorporarlo en el sistema y enviar las notificaciones a los dispositivos móviles. Por último, se realizará la codificación de cada una de las capas con sus respectivas herramientas.

2.2.1. *SPRINT* DE INSTALACIÓN DE HERRAMIENTAS

Una vez realizado el diseño de cada uno de los componentes del sistema, se procederá a instalar y configurar las herramientas necesarias para la implementación de cada una de las capas que conforman el sistema.

a) Instalación y Configuración de *SQL Server Express Edition* 2017

Para la implementación de la base de datos se requiere instalar y configurar el gestor de base de datos, para este Proyecto de Titulación se seleccionó *Microsoft SQL Server 2017 Express Edition* disponible en [28]. En la Figura 2.24 se muestra una captura de la instalación del gestor de base de datos.

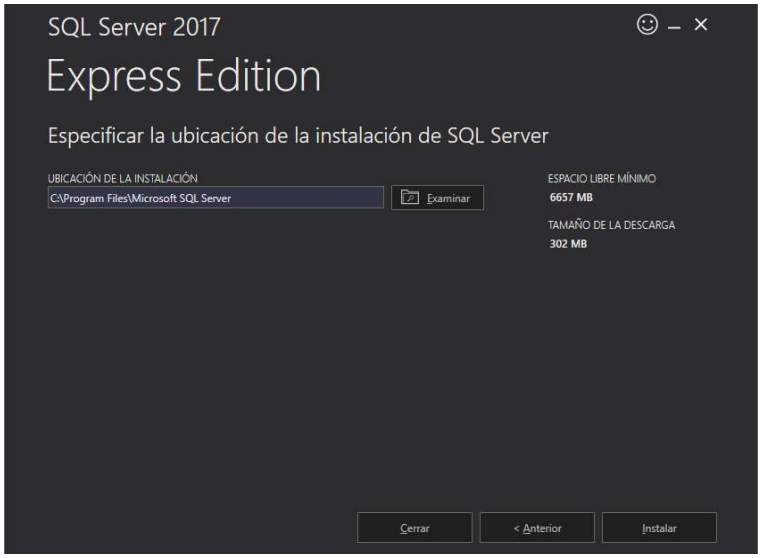

Figura 2.24 Instalación *SQL Server 2017 Express Edition.*

Es importante mencionar que durante la instalación es necesario proporcionar un usuario y contraseña para la administración del gestor. Una vez instalado el gestor de base de datos, es necesario instalar la interfaz gráfica para facilitar su administración. Se procede con la instalación de *Microsoft SQL Server Management Studio* [29].

En la Figura 2.25 se muestra la pantalla de inicio de *SQL Management Studio.*

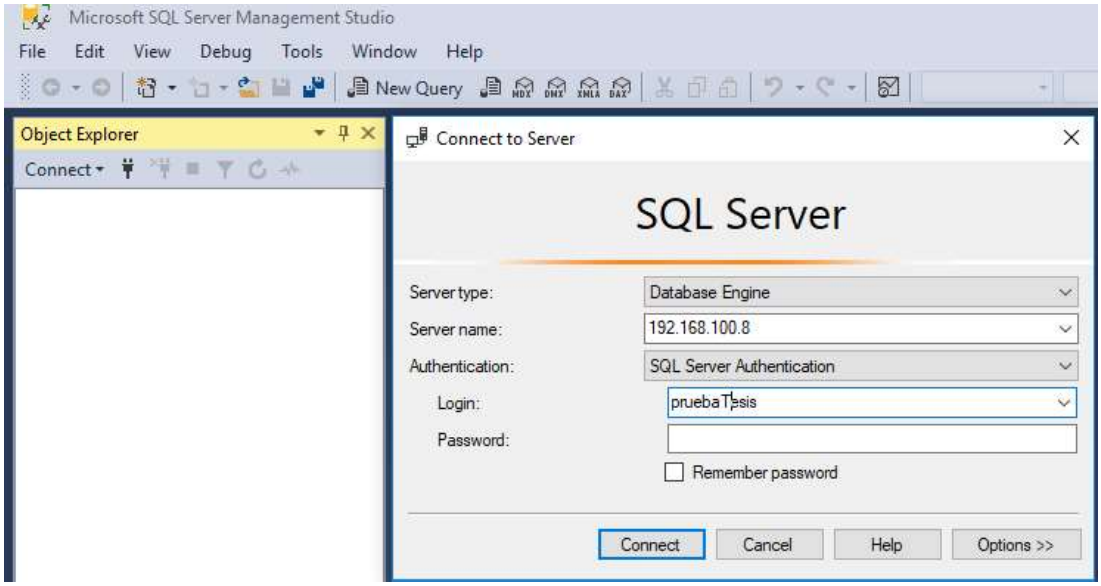

Figura 2.25 Pantalla de inicio de SQL Management Studio.

b) Instalación de *Visual Studio Comunity* 2017

La Capa de Negocio se implementará utilizando *Visual Studio Express* 2017 [30]. Para realizar la instalación es necesario descargar la herramienta *Visual Studio Community* 2017 en el siguiente enlace [30]. Mediante dicha herramienta se puede instalar y configurar *Visual Studio* con sus diferentes lenguajes de programación como por ejemplo *C++*, *C#*, etc. También se debe recordar que para instalar esta herramienta es necesario contar el *Netframework* 4.0. En la Figura 2.26 se puede observar la pantalla de instalación de *Visual Studio Community* 2017 con los componentes que lo conforman.

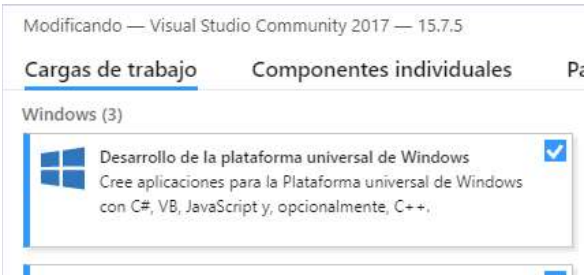

Figura 2.26 Instalación de *Visual Studio Community* 2017.

c) Instalación de *Xamarin*

Xamarin es un SDK de *Andriod* para desarrolladores de *.Net* que permite desarrollar aplicativos nativos de *Android* utilizando *Visual Studio* con el lenguaje *C#* [31].

Para instalar *Xamarin* se debe iniciar *Visual Studio Community* 2017 y dirigirse a la opción "Desarrollo móvil con .NET" y activarla. Automáticamente se seleccionará los kits de desarrollo necesarios [32].

En la Figura 2.27 se muestra lo expuesto anteriormente.

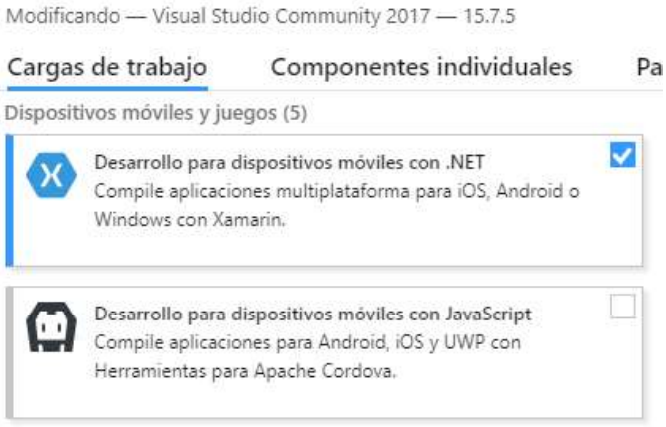

Figura 2.27 Instalación de *Xamarin.*

d) Instalación de *Openfire*

Para instalar el servidor *Openfire* se debe descargar el ejecutable en el enlace [33], este servidor se encuentra disponible para Windows, *Linux* y *Mac.* Una nota muy importante es que dicho servidor puede utilizar una base de datos interna para el almacenamiento de sus tablas, pero por defecto recomienda que esta base se lo configure para utilizar un gestor externo. Para el presente Proyecto de Titulación se utiliza *SQL Server Express*. Primeramente, se creó una base de datos llamada "*Openfire*" para que almacene su información. En la Figura 2.28 y Figura 2.29 se muestra las plataformas disponibles y la instalación de *Openfire.*

Figura 2.28 Ejecutables disponibles de *Openfire.*

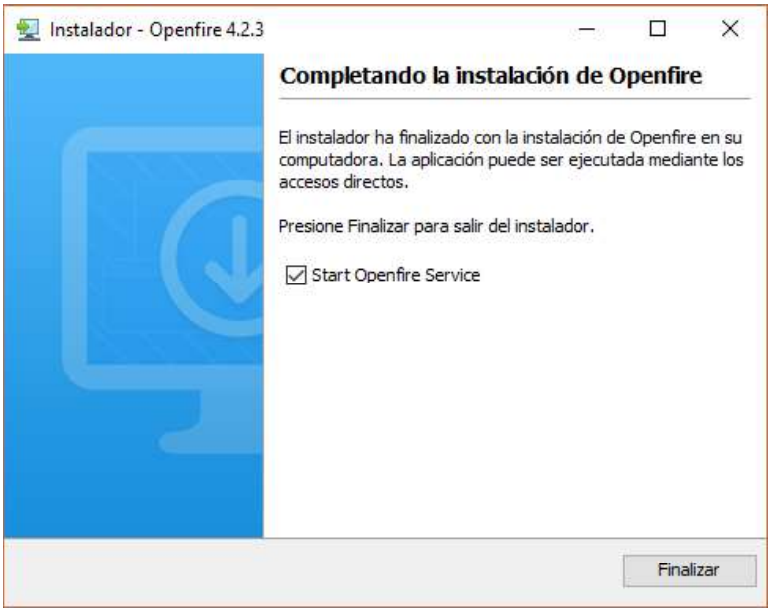

Figura 2.29 Instalación de *Openfire.*

Terminada la instalación se presenta una pantalla con el estado del servicio como se puede ver en la Figura 2.30. Una vez iniciado el servicio, se requiere configurar el servidor *Openfire.* Esto se lo realiza mediante el portal de configuración que nos brinda dicho servidor y se puede acceder mediante la dirección *127.0.0.1:9090/server-db.jsp* (dirección por defecto). En la Figura 2.31 se puede visualizar el portal de configuración mencionado. Adicionalmente, en la Figura 2.32 se muestra la pantalla para la configuración de la base de datos externa que utilizará *Openfire* para almacenar su información.

Figura 2.30 Pantalla de servicio *Openfire.*

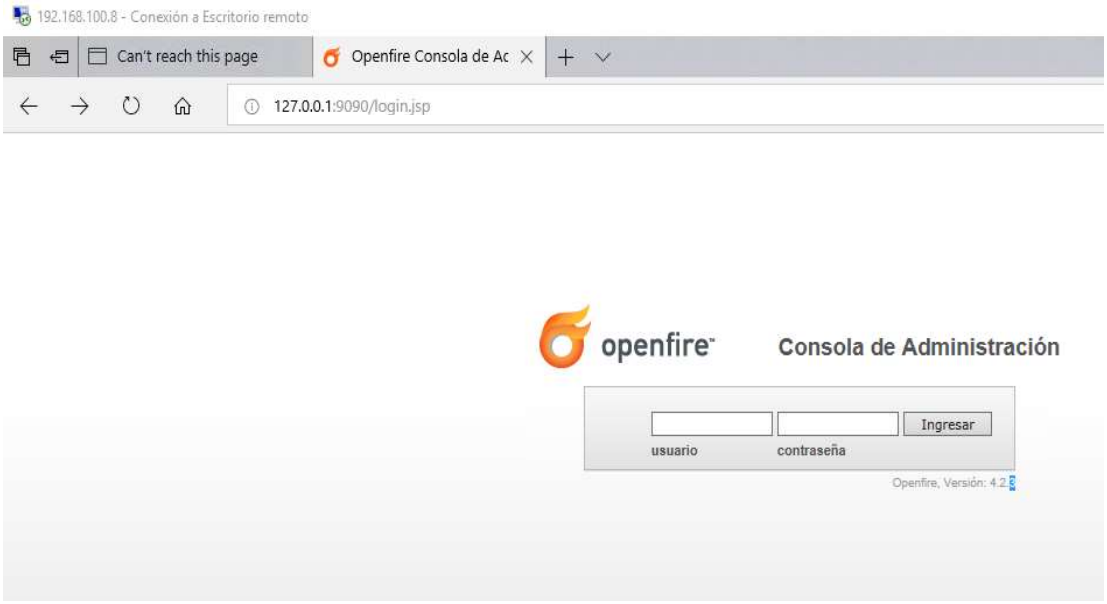

Figura 2.31 Portal de configuración de *Openfire.*

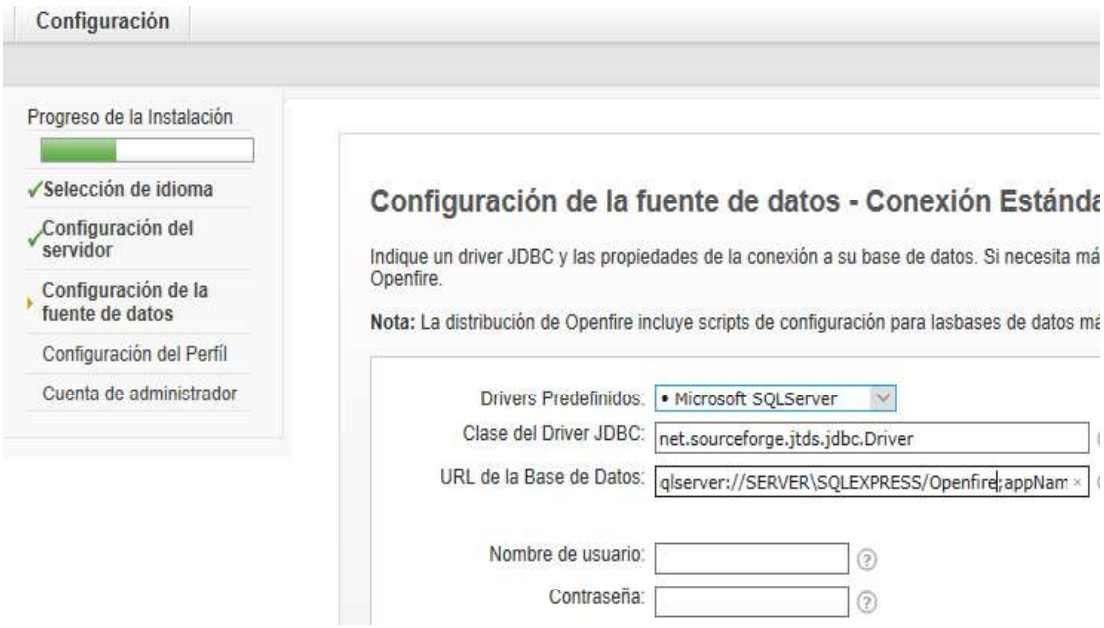

Figura 2.32 Pantalla de configuración de base de datos *Openfire.*

2.2.2. *SPRINT* IMPLEMENTACIÓN DE LA CAPA DE DATOS

Una vez instaladas las herramientas necesarias, se realizará la codificación de la base de datos diseñada en el Apartado 2.1.5. Se utiliza el software *Management SQL-Server* que establece la conexión con el gestor de base de datos utilizando las credenciales de acceso creadas durante la instalación de *SQL-Server Express.*

El Código 2.1 corresponde a una parte de la codificación que se utilizó para la creación de la base de datos siguiendo el modelo relacional presentado en el diseño del Apartado 2.1.5.

```
--Empleado
create table Empleado(
empCodigo varchar(15) not null,
empNombres varchar(100),
empPassword varbinary(255) not null,
empDireccion varchar(80),
empCedula varchar(15) unique,
depCodigo varchar(12) references Departamento(depCodigo),
carCodigo varchar(12) references Cargo(carCodigo),
estCodigo varchar(12) references NivelEstudio(estCodigo),
empCargas int,
empFechaIngreso datetime,
empFechaSalida datetime,
empStatus char(1),
empObservacion varchar(80),
empDiasVacaciones int,
asrCodigo varchar(10) references AsientoRol(asrCodigo),
empTelefonoConvencional varchar(15),
empTelefonoCelular varchar(15),
empCorreoElectronico varchar(200),
empTipoCuenta varchar(10),
empNumeroCuenta varchar(200),
banCodigo varchar(5) references Banco(banCodigo),
empSueldo float,
PRIMARY KEY(empCodigo)
)
```

Código 2.1 Creación de la Tabla Empleado.

Este código permite la creación de la tabla **Empleado,** la misma que contiene los siguientes campos:

- empCodigo: Código principal de la tabla.
- empNombres: Nombre y apellido del usuario que se ha registrado para el uso del aplicativo.
- empPassword: Contraseña utilizada para la autenticación y acceso al aplicativo.
- empDireccion: Dirección del domicilio.
- empCedula: Número de identificación del usuario.
- depCodigo: Código del departamento al que pertenece el empleado.
- carCodigo: Código del cargo que ejerce el empleado.
- estCodigo: Código del nivel de estudio del empelado.

- empCargas: Número de cargas familiares que tiene el empleado.
- empFechaIngreso: Fecha de ingreso a la empresa.
- empFechaSalida: Fecha de salida de la empresa.
- empStatus: Estado del empleado en el sistema, estos pueden ser activos o no activos.
- empObservacion: Campo utilizado para agregar alguna información relevante.
- empDiasVacaciones: Número de días de vacaciones que dispone el empleado.
- asrCodigo: Código del grupo contable al que pertenece el empleado.
- empTelefonoConvencional: Número de teléfono del domicilio del empleado.
- empTelefonoCelular: Número celular del empleado.
- empCorreoElectronico: Correo electrónico registrado en el aplicativo.
- empTipoCuenta: Tipo de cuenta bancaria, puede ser ahorros, corriente, etc.
- empNumeroCuenta: Número de cuenta bancaria.
- banCodigo: Código del banco en el cual el empleado posee una cuenta.
- empSueldo: Valor del sueldo neto que recibe el empleado por sus servicios.

Los campos tienen la propiedad *"NOT NULL"*, como se puede observar en el Código 2.1. Esto indica que no se aceptan valores nulos. Además, se puede resaltar que el campo **empCódigo** se utilizará como clave primaria.

La descripción de las entidades del Proyecto de Titulación se encuentra en el **Anexo B**.

2.2.3. *SPRINT* DE IMPLEMENTACIÓN DE LA CAPA DE NEGOCIO

En esta sección se presenta el detalle de la implementación de la capa de negocio en la cual se codificará todo lo referente al diseño realizado en el Apartado 2.1.6.

Después de instalar *Visual Studio* 2017, se debe generar la solución para la implementación del proyecto. Esta solución constará de los siguientes proyectos:

- **Cliente:** Formularios para la presentación al usuario.
- **Datos:** Biblioteca de clases para interactuar con el gestor de base de datos.

- **Negocio:** Biblioteca de clases con la lógica de negocio.
- **Servidor:** Formulario Windows en el que se mostrará la configuración de servicio *.Net Remoting* y conexión al servidor *Openfire***.**

En la Figura 2.33 se muestra la creación de la solución en *Visual Studio*.

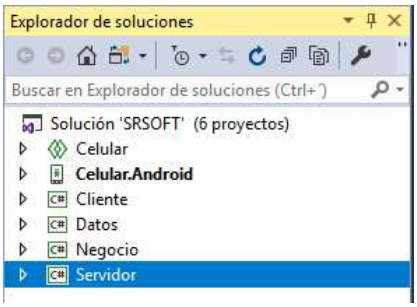

Figura 2.33 Estructura de la solución en *Visual Studio.*

Una vez creada la solución, se procede a la codificación de los proyectos **Datos** y **Negocio** donde se encuentran todas las clases que se diseñaron en el Apartado 2.1.6.

a) Implementación del proyecto Datos

A continuación, se puede observar la codificación de la clase **BDD** que pertenece al proyecto **Datos.** Este proyecto servirá como intermediario para que la capa **Negocio** pueda comunicarse con el gestor de base de datos implementado en el apartado anterior; es decir, el proyecto **Datos** generará una librería que funcionará como una caja negra en la cual la capa **Negocio** proporcionará los parámetros requeridos y la librería ejecutará las funcionalidades solicitadas. El Código 2.2 corresponde a la codificación de los atributos de la clase **BDD.**

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;

namespace Datos
{
	public class BDD
	{
	//Atributos
	private static string servidorBDD;
	private static string baseDatos;
	private static string usuario;
	private static string password;
	private static string cadenaConexion;
	private static string detalleError;
```

Código 2.2 Codificación de los atributos de la clase BDD.

Los atributos de la clase **BDD** se describen a continuación:

- servidorBDD: Dirección Ip o nombre del servidor de la base de datos.
- baseDatos: Nombre de la base de datos a la que se conectará el proyecto Datos.
- usuario: Nombre de usuario utilizado para conectarse al gestor de base de datos.
- password: Contraseña utilizada para conectarse al gestor de base de datos.
- cadenaConexion: *string* de conexión utilizado para interactuar con el gestor de base de datos.
- detalleError: cadena de caracteres en la cual se almacena el mensaje de error que se puede generar al momento de transaccionar con la base de datos. Este mensaje está personalizado utilizando el código presentado por la excepción y consultado en el siguiente enlace [34].

En el Código 2.3 se puede apreciar como ejemplo la codificación de uno de los métodos que dispone la clase BDD.

```
public string ejecutarSentencia(string sentenciaSQL)
{
#region [Ejecutar una sentencia Sql]
SqlConnection conexion = new SqlConnection(cadenaConexion);
SqlCommand comando = conexion.CreateCommand();
try
{
   conexion.Open();
   comando.CommandText = sentenciaSQL;
   comando.ExecuteNonQuery();
   return "Ok";
}
catch (SqlException e)
{
   detalleError = mensajeError(e.Number,e.Message);
   return detalleError;
}
finally
{
   conexion.Close();
}
#endregion
                                    }
```

Código 2.3 Codificación del método ejecutar sentencia de la clase BDD.

El método del Código 2.3 recibe como parámetro de entrada la sentencia *SQL* que se desea enviar al gestor de base de datos, luego genera una respuesta de tipo *String* en el

cual puede informar que la operación se realizó correctamente o se notifica mediante un mensaje personalizado el error que ocurrió.

Toda la codificación del proyecto **Datos** se lo puede visualizar en el **Anexo E**.

b) Implementación del proyecto Negocio

Para la implementación de las clases se seguirá el diseño de clases presentado en el Apartado 2.1.6. En el Código 2.4 se presenta la implementación de los atributos que corresponden a la clase **Empleado,** la cual implementa los mismos campos de la tabla Empleado. Es importante destacar que existen atributos que hacen referencia a otras clases como son Cargo, Departamento, NivelEstudio, Asiento y Banco.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using Datos;

namespace Negocio
{

        public class Empleado : MarshalByRefObject
        {

        #region [Atributos]

        //Atributos
        private string codigo;
        private string nombres;
        private string password;
        private string direccion;
        private string cedula;
        private Departamento departamento;
        private Cargo cargo;
        private NivelEstudio estudios;
        private int cargas;
        private DateTime fechaIngreso;
        private DateTime fechaSalida;
        private string status;
        private string observacion;
        private int diasVacaciones;
        private Asiento asientoRol;
        private string telefonoConvencional;
        private string telefonoCelular;
        private string correoElectronico;
        private string tipoCuenta;
        private string numeroCuenta;
        private Banco banco;
        private decimal sueldo;
        #endregion
```

Código 2.4 Codificación de atributos de la clase Empleado.

Los atributos de la clase **Empleado** son los siguientes:

- codigo: Código principal del empleado.
- nombres: Nombres y apellidos del empleado.
- password: Contraseña utilizada para ingresar al sistema.
- dirección: Dirección del domicilio del empleado.
- cedula: Número de identificación del empleado.
- departamento: Departamento en el que trabaja el empleado.
- cargo: Cargo que ejerce el empleado en la empresa.
- estudios: Nivel de estudio del empleado.
- cargas: Numero de cargas familiares.
- fechaIngreso: Fecha de ingreso del empleado a la empresa.
- fechaSalida: Fecha de salida o liquidación del empleado.
- status: Estado del empleado, el cual puede ser activo o no activo.
- observación: Observación adicional que se puede incluir a la ficha del empleado.
- diasVacaciones: Días de vacaciones que dispone el empleado.
- asientoRol: Grupo de cuentas contables a la que pertenece el empleado.
- telefonoConvencional: Número de teléfono convencional al que se puede comunicar la empresa en caso de emergencia.
- telefonoCelular: Número de teléfono celular del empleado.
- correoElectronico: Dirección de correo electrónica registrado por el empleado.
- tipoCuenta: Tipo de cuenta bancaria que posee el empleado, esta puede ser AHO para ahorros o CTE para cuenta corriente.
- numeroCuenta: Numero de cuenta bancaria.
- banco: Entidad bancaria en la cual el empleado posee una cuenta.
- sueldo: Sueldo que recibe el empleado.

También es importante destacar que en el Código 2.4 la clase **Empleado** tiene una herencia de la clase *MarshalByRefObject.* Mediante esta herencia se puede utilizar la clase empleado de manera remota utilizando *.Net Remoting.*

A continuación, en el Código 2.5 se presenta a manera de ejemplo uno de los métodos de la clase empleado.

```
//Login
   public string LoginEmpleado(Empleado empleado,out string valor)
   {
       #region[login del empleado]

       valor = string.Empty;

      try
      {
      if (empleado.Cedula != string.Empty && empleado.Password != string.Empty)
           {
               valor = string.Empty;
               BDD baseDatos = new BDD();
               return baseDatos.ejecutarComando("if exists (select * from
               Empleado where empCedula='" + empleado.Cedula + "') if exists
               (select * from Empleado where empCedula='" + empleado.Cedula +
               "' and replace(empStatus,' ','')='A') if exists (select
               empCedula from Empleado where pwdcompare('" + empleado.Password
               + "',empPassword)=1 and empCedula='" + empleado.Cedula + "')
               select 'Ok' else select'Contraseña incorrecta' else
               select'Usuario inactivo'else select 'No existe el usuario'",
               out valor);
           }
           else
           {
               return "El usuario y password son obligatorios";
           }
       }
       catch (Exception ex)
       {
           return "Empleado: " + ex.Message;
       }

       #endregion
   }
```

Código 2.5 Codificación del método Login de la clase Empleado

El método del Código 2.5 es utilizado para la autenticación de un usuario en el sistema. Como se explicó anteriormente los métodos responderán una cadena de caracteres para indicar si se ejecutó de manera correcta o si existió algún inconveniente. Todas las clases del proyecto **Negocio** se comunicarán con la base de datos a través de la clase **BDD** perteneciente al proyecto **Datos**, es decir la capa de negocio no podrá enviar comandos directamente al gestor de base de datos.

Toda la codificación del proyecto **Datos** se lo puede visualizar en el **Anexo E**.

2.2.4. *SPRINT* DE IMPLEMENTACIÓN DE LA CAPA DE PRESENTACIÓN

Tomando en consideración el diseño realizado en el Apartado 2.1.7, se presenta la implementación de los formularios Windows tanto para la aplicación Cliente como la aplicación Servidor; estos elementos están nombrados como componente (A) y componente (B) en el Apartado 2.1.1 . También se muestra la configuración necesaria para que el Cliente y Servidor puedan interactuar utilizando la tecnología *.Net Remoting*.

a) Implementación de los formularios del aplicativo cliente

Como siguiente paso se realizó la codificación de los formularios del aplicativo Cliente o **Componente A** la cual utilizará el usuario encargado de generar los roles de pago. A continuación, en la Figura 2.34 se muestra como ejemplo la implementación del formulario **Empleado**, el cual se basó en la plantilla de catálogos descrito en el Apartado 2.1.7.

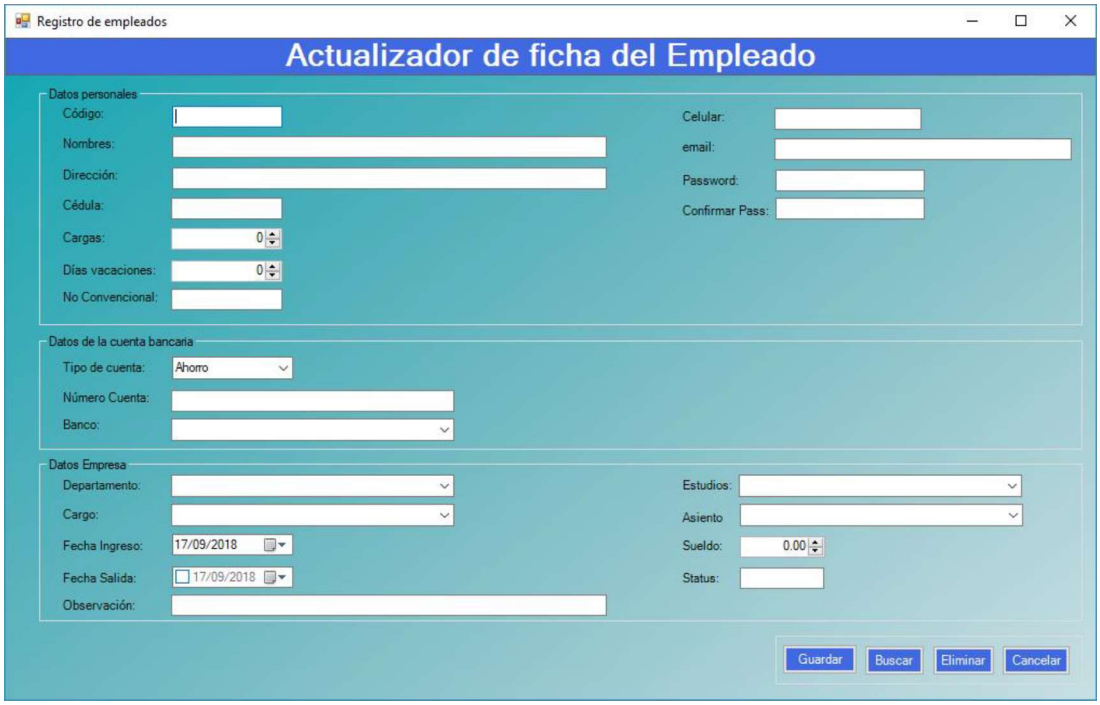

Figura 2.34 Interfaz gráfica del formulario *frmEmpleado.*

La codificación de todos los formularios se encuentra en el **Anexo E**.

b) Implementación del formulario para el aplicativo servidor

Continuando con la implementación, se realizó la implementación del formulario Windows para el componente B o Aplicativo Servidor. Este formulario es netamente informativo ya que se presenta la dirección Ip en la cual se brinda el servicio mediante *.Net Remoting* y

además la dirección Ip del servidor *Openfire* al cual se va conectar. En la Figura 2.35 se muestra lo mencionado.

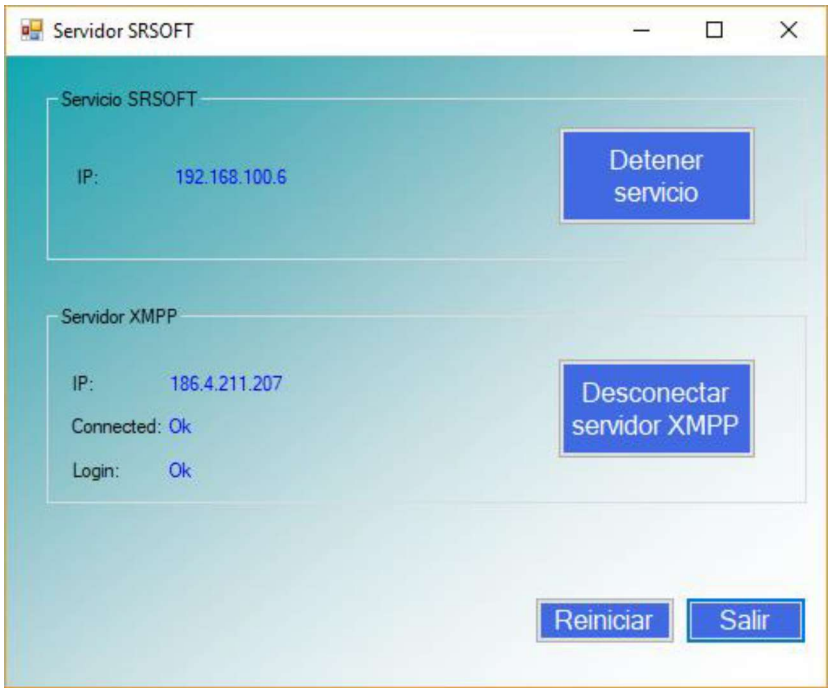

Figura 2.35 Interfaz gráfica formulario Servidor SRSOFT.

c) Implementación de Reportes

Tomando en consideración el diseño de los reportes presentado en el Apartado 2.1.7, se realiza la implementación utilizando *Crystal Reports* [35]. En la Figura 2.36 se muestra la creación de un nuevo reporte utilizando la herramienta mencionada.

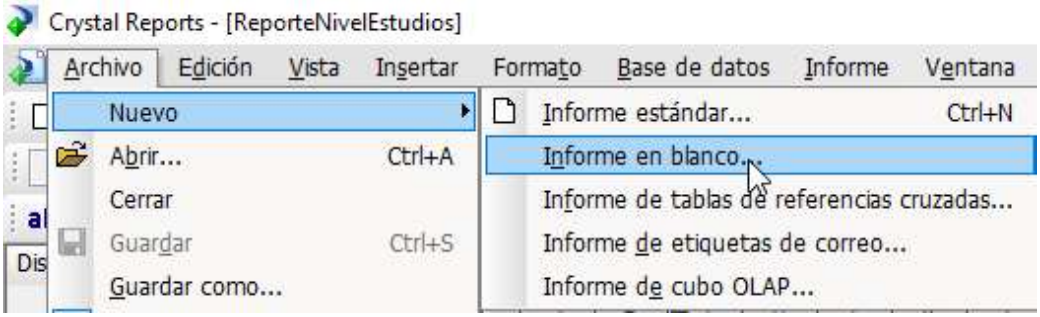

Figura 2.36 Captura de pantalla para la creación de un Nuevo informe.

En la Figura 2.37 se muestra la implementación del reporte que obtiene la lista de empleados registrados en el sistema, en el cual se puede visualizar los campos requeridos por el usuario y además dispone de una cabecera para identificar el nombre del reporte.

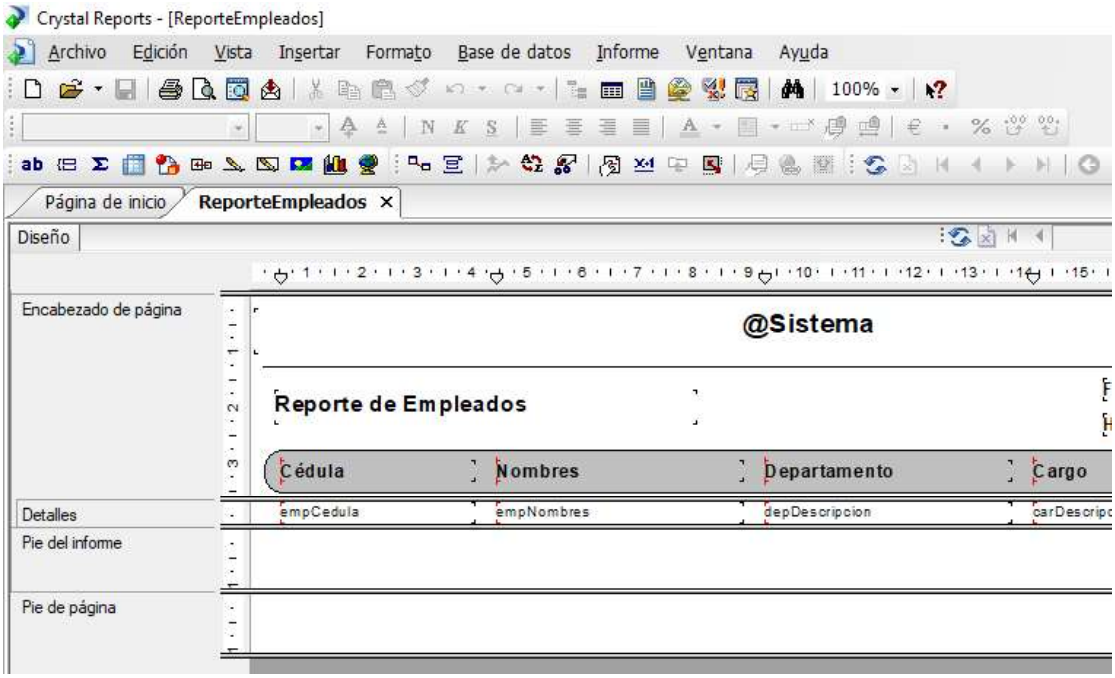

Figura 2.37 Captura de pantalla del Reporte que obtiene la lista de empleados registrados en el sistema.

d) Implementación de *.Net Remoting*

Para la implementación de *.NET Remoting* es necesario agregar como referencia el proyecto Negocio tanto al Cliente o componente A como al Servidor o componente B, ya que ambos extremos deben conocer los objetos y métodos que se ejecutarán de manera remota. En la Figura 2.38 se puede visualizar lo mencionado.

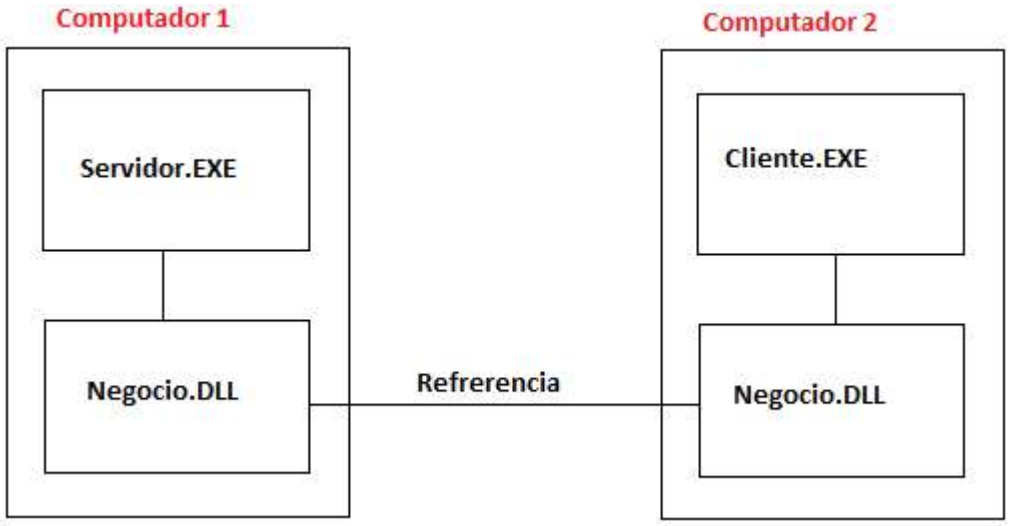

Figura 2.38 Estructura inicial de .Net Remoting.

Una vez agregado en *Visual Studio,* las referencias de los proyectos “Cliente” y “Servidor” quedan como se indica en la Figura 2.39.

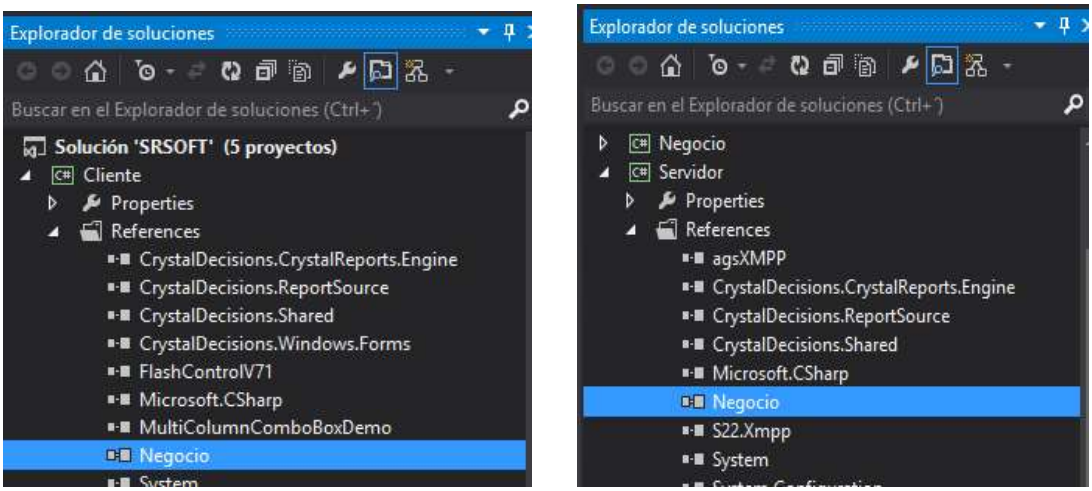

Figura 2.39 Referencia Negocio- Cliente. Negocio - Servidor**.**

Posteriormente es necesario que todas las clases las cuales se ejecutarán de manera remota incluyan la herencia *MarshalByRefObject,* por ejemplo en la Figura 2.40 se muestra el resultado de agregar la referencia para que se pueda utilizar el objeto de manera remota.

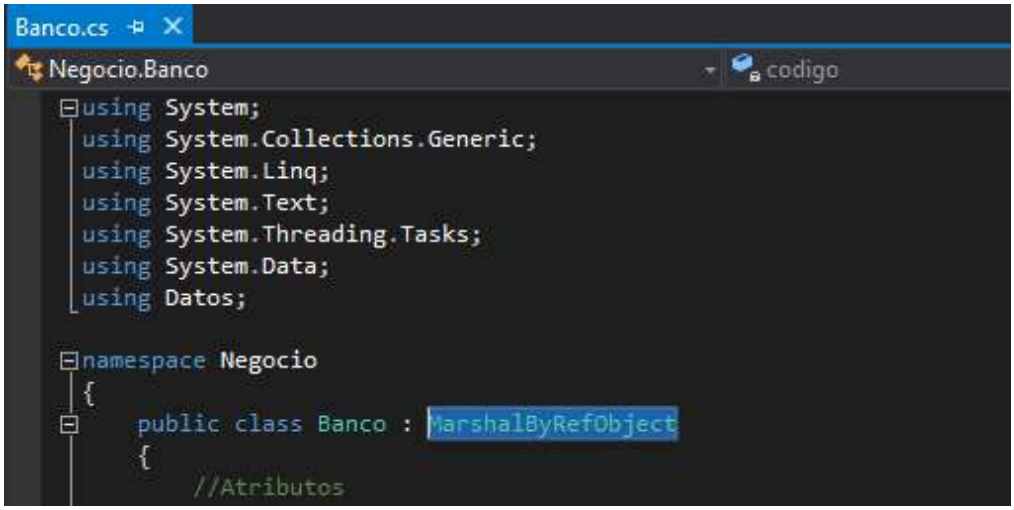

Figura 2.40 Herencia *MarshalByRefObject.*

Una vez implementada esta herencia se debe editar los archivos de configuración tanto en el Cliente como en el Servidor para agregar la información que *.Net Remoting* necesita.

En el Código 2.6 se puede observar la codificación del archivo de configuración del aplicativo cliente o **componente A** para que pueda comunicarse con el servidor a través de *.Net Remoting*.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.runtime.remoting>
    <!—Nombre de la aplicación cliente-->
    <application name="SRSOFT">
      <!—URl del servidor-->
      <client url="tcp://127.0.0.1:65000/Servidor">
        <!—Objetos que brinda servicio la aplicación servidor-->
        <activated type="Negocio.Asiento,Negocio"/>
        <activated type="Negocio.AsientoDetalle,Negocio"/>
        <activated type="Negocio.Banco,Negocio"/>
        <activated type="Negocio.Cargo,Negocio"/>
        <activated type="Negocio.Comprobante,Negocio"/>
        <activated type="Negocio.ComprobanteDetalle,Negocio"/>
        <activated type="Negocio.Consulta,Negocio"/>
        <activated type="Negocio.Cuenta,Negocio"/>
        <activated type="Negocio.Departamento,Negocio"/>
        <activated type="Negocio.Empleado,Negocio"/>
        <activated type="Negocio.NivelEstudio,Negocio"/>
        <activated type="Negocio.Periodo,Negocio"/>
        <activated type="Negocio.Permisos,Negocio"/>
        <activated type="Negocio.RolDePagos,Negocio"/>
        <activated type="Negocio.TipoDescuento,Negocio"/>
        <activated type="Negocio.Notificacion,Negocio"/>
      </client>
      <channels>
          <!—Definición del canal-->
          <channel ref="tcp client" secure="false">
            <serverProviders>
            <!—Formato-->
            <formatter ref="binary" typeFilterLevel="Full"/>
          </serverProviders>
        </channel>
      </channels>
    </application>
  </system.runtime.remoting>
  <appSettings file="">
    <clear/>
    <!--Sistema-->
    <add key="Sistema" value="SRSOFT"/>
       <add key="PathReportes"
       value="C:\Users\Jonny\Desktop\Proyecto\Sistema\SRSOFT\Negocio\bin\Debug\Repor
       tes\"/>
  </appSettings>
  <startup useLegacyV2RuntimeActivationPolicy="true">
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
    </startup>
</configuration>
```

Código 2.6 Archivo de configuración de la aplicación cliente.

El archivo de configuración de la aplicación Cliente debe contener la siguiente información:

- Nombre de la aplicación.
- Url de conexión al servidor.
- Objetos que utilizará del servidor.
- Tipo de canal que utilizará (Tcp,Http).
- Credenciales de red (opcional).

Nota: la url de conexión incluye dirección IP, puerto y nombre de la aplicación del servidor.

En el Código 2.7 se puede observar la codificación del archivo de configuración de la aplicación servidor o **componente B**.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <customErrors mode="off" />
    <!—Nombre de la aplicación-->
    <application name="Servidor">
      <service>
        <!—Objetos que brinda servicio-->
        <activated type="Negocio.Asiento,Negocio"/>
        <activated type="Negocio.AsientoDetalle,Negocio"/>
        <activated type="Negocio.Banco,Negocio"/>
        <activated type="Negocio.Cargo,Negocio"/>
        <activated type="Negocio.Comprobante,Negocio"/>
        <activated type="Negocio.ComprobanteDetalle,Negocio"/>
        <activated type="Negocio.Consulta,Negocio"/>
        <activated type="Negocio.Cuenta,Negocio"/>
        <activated type="Negocio.Departamento,Negocio"/>
        <activated type="Negocio.Empleado,Negocio"/>
        <activated type="Negocio.NivelEstudio,Negocio"/>
        <activated type="Negocio.Periodo,Negocio"/>
        <activated type="Negocio.Permisos,Negocio"/>
        <activated type="Negocio.RolDePagos,Negocio"/>
        <activated type="Negocio.TipoDescuento,Negocio"/>
        <activated type="Negocio.Notificacion,Negocio"/>
        <!—Configuración del tipo de .Net Remoting -->
        <wellknown mode="SingleCall"
                   type="Server.SampleService, Server"
                   objectUri="Server.rem"/>
      </service>
      <channels>
          <!—Configuración del canal-->
          <channel ref="tcp server" port="65000">
            <serverProviders>
              <!--Formato-->
              <formatter ref="binary" typeFilterLevel="Full" />
            </serverProviders>
        </channel>
      </channels>
    </application>
  </system.runtime.remoting>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
                                </configuration>
```

Código 2.7 Archivo de configuración de la aplicación Servidor.

El archivo de configuración de la aplicación Servidor debe contener la siguiente información:

- Nombre de la aplicación.
- Objetos que se ejecutarán de manera remota.

- Modo de creación de objetos (*Singlecall,Singletone o CAO*)
- Tipo de canal que utilizará (*Tcp*,*Http*).
- Número de puerto.
- Formato en el que se transmitirá la información.

2.2.5. *SPRINT* IMPLEMENTACIÓN DEL APLICATIVO MÓVIL

La implementación del aplicativo móvil se lo realizó utilizando *Xamarin*, el cual nos permite implementar aplicativos utilizando *.Net.*

a) Implementación de la base de datos del móvil

La implementación de la base de datos del teléfono se lo realizó utilizando *SQLite* [36], el cual se lo puede incorporar en *Visual Studio* mediante la herramienta *NuGet* [37]. *Nuget* es un administrador de paquetes que permite la incorporación de librerías disponibles en internet. Mediante *clic* derecho sobre el proyecto se puede acceder a esta herramienta. En la Figura 2.41 se puede observar la opción para acceder al *Nuget*.

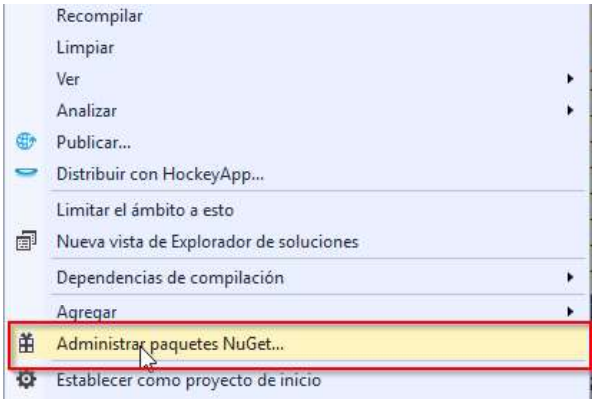

Figura 2.41 Administrador de paquetes *Nuget* en *Visual Studio.*

En la Figura 2.42 se puede observar la incorporación de *SQLite* al proyecto.

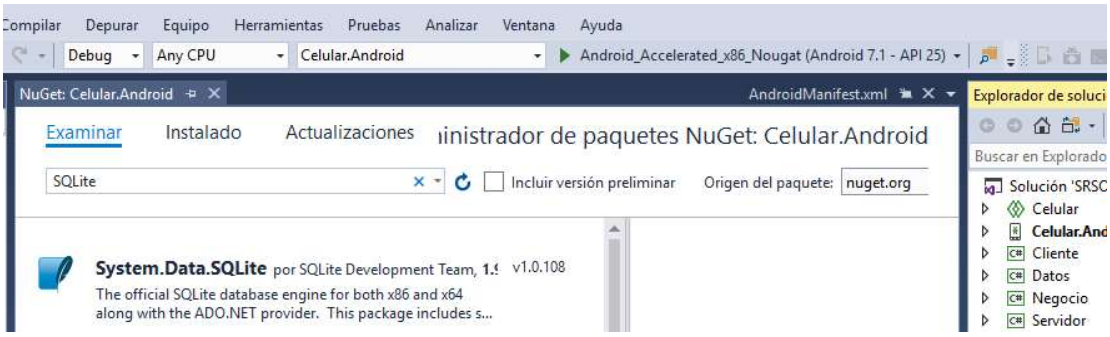

Figura 2.42 Incorporación de *SQLite* en el proyecto.

En el Código 2.8 se puede observar la codificación que permite crear una base de datos en el dispositivo móvil utilizando *SQLite.*

```
public string CrearBase()
{
       SqliteConnection connection = new SqliteConnection();
       try
       {
       bool exists = File.Exists(dbPath);
       if (!exists)
       {
       SqliteConnection.CreateFile(dbPath);
       connection = new SqliteConnection("Data Source=" + dbPath);
       connection.Open();
       var commands = new[] { "CREATE TABLE [Rol] (idRol ntext PRIMARY
       KEY,descripcionRol ntext, cedula ntext,nombreEmpleado ntext,cargo
       ntext,departamento ntext);", "CREATE TABLE [RolDetalle] (idRol
       ntext,idDetalle int,codConcepto ntext, concepto ntext,tipo ntext, valor
       decimal,FOREIGN KEY(idRol) REFERENCES Rol(idRol))", "create table
       Configuracion(Hostname ntext PRIMARY KEY,Username ntext,Password ntext,Puerto
       int)" };
       foreach (var command in commands)
       {
       using (var c = connection.CreateCommand())
       {c.CommandText = command;
       var i = c.ExecuteNonQuery();}
       }
       }
       return "Ok";
       }
       catch (Exception ex)
       {
       return ex.Message;
       }
       finally
       {
       if (connection.State == ConnectionState.Open)
       connection.Close();
       }
```

Código 2.8 Codificación del método para crear la base de datos en el móvil.

b) Implementación de las notificaciones

Una vez implementada la base de datos se incorporarán las notificaciones que serán enviadas utilizando el protocolo *XMPP*. Para cumplir con este objetivo es necesario incorporar dos paquetes adicionales disponibles en el *Nuget* de *Visual Studio,* los cuales son *Sharp.XMPP* [38] y *Xam.Plugins.Notifier* [39]*.* El paquete *Sharp.XMPP* dispone de los métodos necesarios para comunicarse con cualquier servidor que utilice el protocolo *XMPP,* para el presente Proyecto de Titulación este paquete se lo utilizará para enviar el detalle de los roles de pago a través del servidor *Openfire*. Previo al envío de las notificaciones, es importante mencionar que *Openfire* dispone de su propia base de datos en la que se guardará la información de sus clientes; es por esto que cada empleado

dispondrá de sus propias credenciales para acceder a este servicio. En la Figura 2.43 se puede visualizar la creación de las credenciales de un cliente *XMPP* en el servidor *Openfire.*

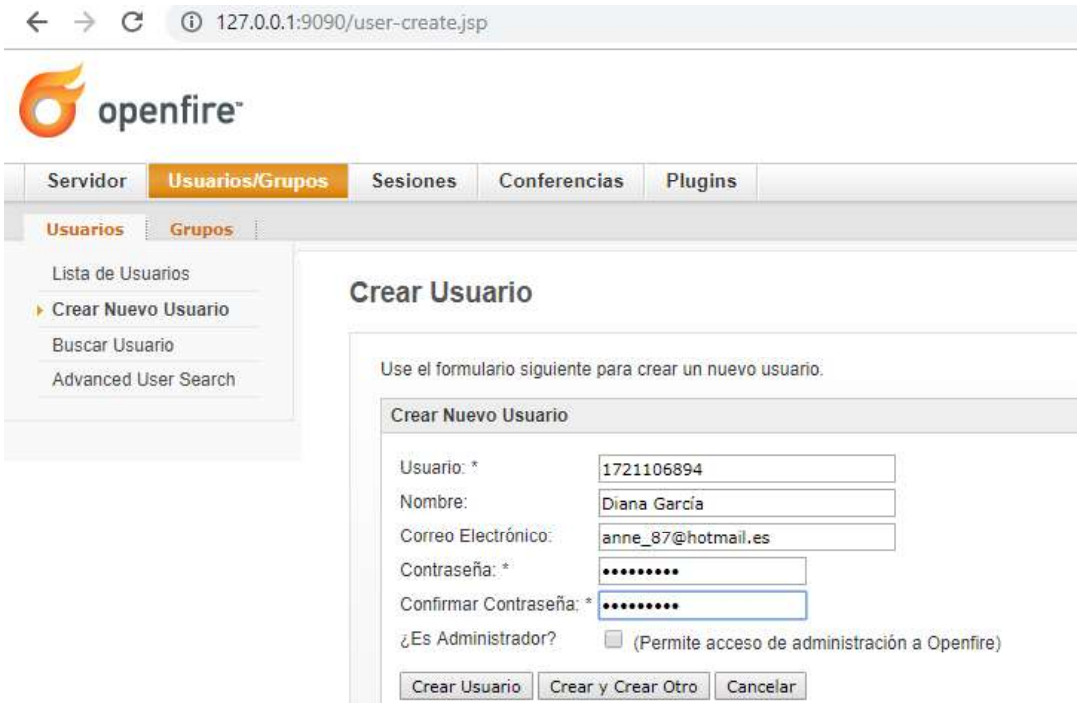

Figura 2.43 Creación de clientes en *Openfire.*

Una vez creadas las credenciales se realiza la codificación del método **EnviarNotificacion** de la clase **Notificacion** que se diseñó en el Apartado 2.1.6, el cual es utilizado para enviar un mensaje de tipo *string* hacia un cliente *XMPP*, ver el Código 2.9.

```
public string EnviarNotificacion(RolDePagos rol)
{
XmppClient cliente = new XmppClient(Hostname, Username, Password, Puerto, Tls);
 Empleado empleado = new Empleado();
 string respuesta = string.Empty;
 try
 {
   cliente.Connect();
   //Genera los string para enviar las notificaciones
   foreach (DataRow dr in rol.RolPagos.Tables[0].Rows)
   {Mensaje=generacionString();
   //Envia los mensajes a través de Openfire
   cliente.SendMessage(destinatario, mensaje)}; cliente.Dispose();
 }
 catch(Exception ex)
 { cliente.Dispose();
   respuesta = respuesta + "\n" + ex.Message;
 }
}
```

Código 2.9 Codificación del método de envío de notificaciones mediante *Openfire.*

Por otra parte, el paquete *Xam.Plugins.Notifier* permite mostrar notificaciones locales en el dispositivo móvil. En la Figura 2.44 se muestra un ejemplo de una notificación local visualizada en un dispositivo móvil con sistema operativo *Android*.

Figura 2.44 Ejemplo de notificación local en el dispositivo móvil.

c) Implementación de la presentación del aplicativo móvil

En este apartado se presenta a manera de ejemplo la implementación de un formulario que dispone el aplicativo móvil. Este formulario se encarga de mostrar la información de los roles de pago generados por la persona encargada en la empresa. Dicha información llega al dispositivo móvil utilizando el protocolo *XMPP*. En la Figura 2.45 se observa el formulario que presenta el detalle del rol de pagos generado para el mes de enero del 2018.

Figura 2.45 Formulario para la revisión de los roles de pago en el aplicativo móvil.

En el Código 2.10 se muestra una parte de la codificación utilizada para la implementación del formulario que presenta el detalle del rol de pagos.

```
using System.Globalization;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Celular
{
       [XamlCompilation(XamlCompilationOptions.Compile)]
       public partial class RolDetalle : ContentPage
       {
        Droid.Rol rol = new Droid.Rol();
        Label lblTitulo = new Label();
        Label lblDescripcionRol = new Label();
        Label lblCedula = new Label();
        Label lblNombreEmpleado = new Label();
        Label lblCargo = new Label();
        Label lblDepartamento = new Label();
        TableView tableView = new TableView();

        public RolDetalle ()
             {
                    InitializeComponent ();
             }
        public RolDetalle(Droid.Rol rolAux)
        {
            InitializeComponent();
            rol = rolAux;
            NumberFormatInfo nfi = new CultureInfo("en-US", false).NumberFormat;
            nfi.NumberDecimalSeparator = ".";
            nfi.PerMilleSymbol = "";

            var titulos = new Style(typeof(Label))
            {
                Setters =
                {
                 new Setter { Property = Label.TextColorProperty , Value =
          Color.SkyBlue },
                 new Setter { Property = Label.FontSizeProperty, Value = 18 },
                 new Setter { Property = Label.HorizontalTextAlignmentProperty,
          Value =       TextAlignment.Center }
                }
         }
```

Código 2.10 Codificación del formulario detalle del Rol de pagos en el móvil

En el Código 2.10 se puede ver claramente la facilidad que *Xamarin* nos brinda para la implementación de formularios en los aplicativos móviles y debido a que utiliza lenguaje *C#* es muy sencillo adaptarse a una persona que conoce dicho lenguaje.

3. RESULTADOS Y DISCUSION

En este capítulo se muestra las pruebas realizadas sobre el sistema implementado las cuales pretenden cumplir con cada uno de los requerimientos funcionales y no funcionales que se definieron en el Apartado 2.1.1. Finalmente se realiza un análisis de los resultados obtenidos.

3.1. *SPRINT* PRUEBAS DE REQUERIMIENTOS FUNCIONALES

A continuación, se valida el cumplimiento de los requerimientos funcionales que se describieron en las Historias de Usuario del Apartado 2.1.1. Primero se detalla los requerimientos funcionales que debe cumplir el sistema para luego proceder con las validaciones de cada uno. En la Tabla 3.1 se observa el resumen de los requerimientos.

Tabla 3.1 Resumen de los requerimientos funcionales.

RF	**Descripción**
RF001	Ingresar, modificar y eliminar la ficha del empleado.
RF002	Creación nuevos cargos, de igual manera se puede modificar y eliminar en el caso que el usuario requiera.
RF003	Creación nuevos departamentos, de igual manera se puede modificar y eliminar en el caso que el usuario requiera.
RF004	Creación nuevas cuentas contables, de igual manera se puede modificar y eliminar en el caso que el usuario requiera.
RF005	Creación nuevos tipos de descuento, de igual manera se puede modificar y eliminar en el caso que el usuario requiera.
RF006	Configurar los permisos de acceso dependiendo del cargo.
RF007	Configurar el acceso a los diferentes módulos del sistema dependiendo del cargo.
RF008	Permitir la asignación de tipos de descuento a cada empleado dependiendo del grupo contable al que pertenece.
RF009	Ingresar los periodos contables los cuales deben cumplir con el formato Año-mes.
RF010	Si un periodo se encuentra en estado cerrado, no debe permitir la generación de roles de pago.
RF011	Cada tipo de descuento debe estar asociado a una cuenta contable.
RF012	Registrar los anticipos solicitados por un empleado.
RF013	Generar el rol de pagos general e individual para un periodo.
RF014	Generar asientos contables de los roles de pago.
RF015	Enviar notificaciones al dispositivo móvil con sistema operativo *Android.*
RF016	Imprimir el rol de pagos individual con una copia.
RF017	Imprimir el rol de pagos general del periodo seleccionado.
RF018	Imprimir la ficha del empleado.
RF019	Imprimir un listado de empleados de la empresa.
RF020	Imprimir un listado de cargos de la empresa.
RF021	Imprimir un listado de departamentos de la empresa.
RF022	Imprimir un listado de cuentas contables de la empresa.

En la Figura 3.1 se puede visualizar el cumplimiento del requerimiento RF001 correspondiente a la gestión de la ficha del empleado.

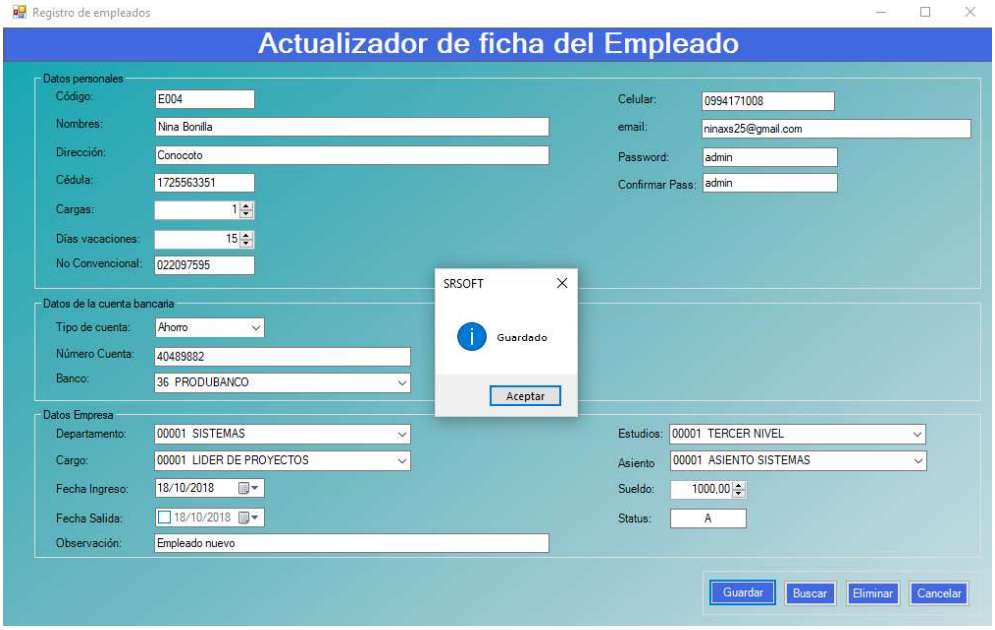

Figura 3.1 Formulario para el registro de la ficha del empleado.

En la figura Figura 3.1 se muestra el mensaje de confirmación cuando el proceso es exitoso, este concepto se lo maneja en todos los formularios. Por otra parte, en las Figura 3.2 y Figura 3.3 se muestra el cumplimiento de los requerimientos RF002 y RF003.

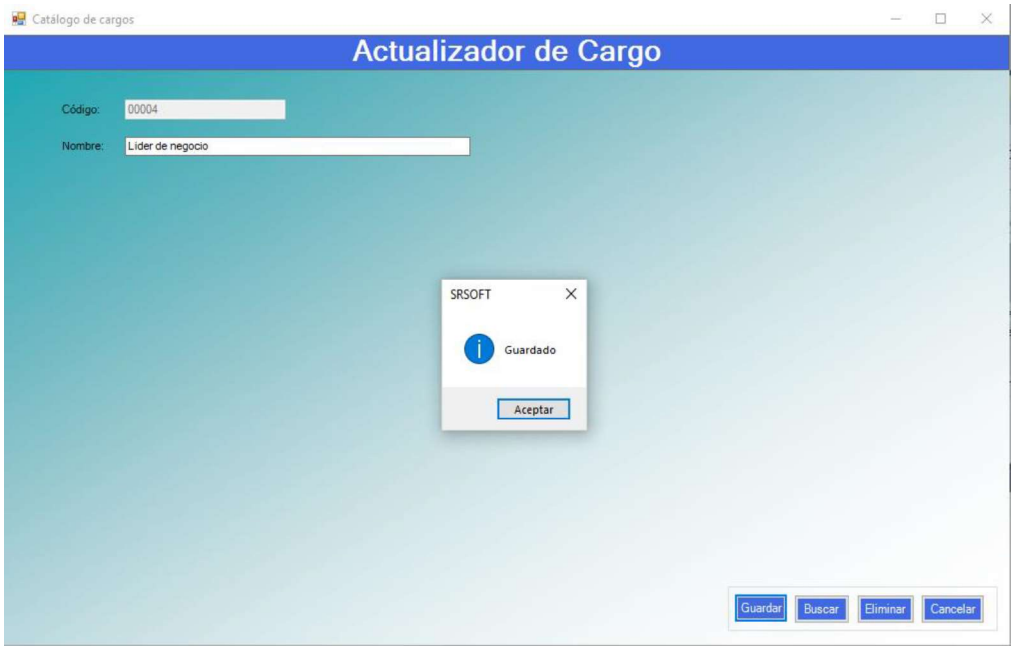

Figura 3.2 Formulario para la gestión de cargos.

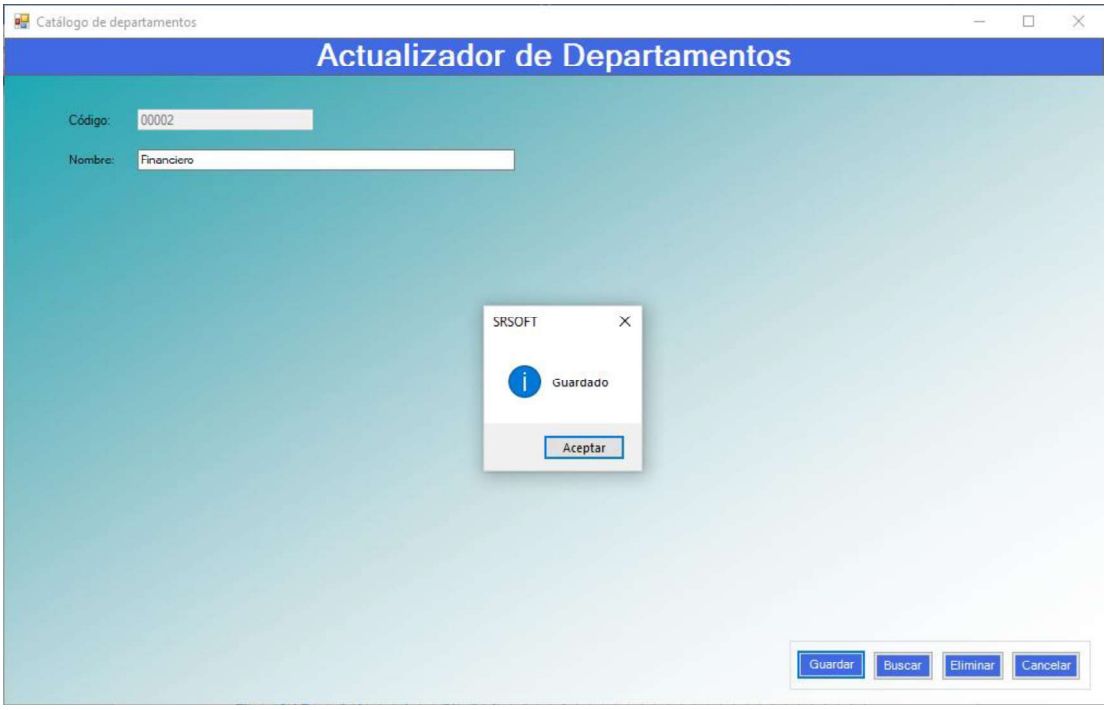

Figura 3.3 Formulario para la gestión de departamentos.

El cumplimiento de los requerimientos RF004 y RF005 se los puede observar en la Figura 3.4 y Figura 3.5.

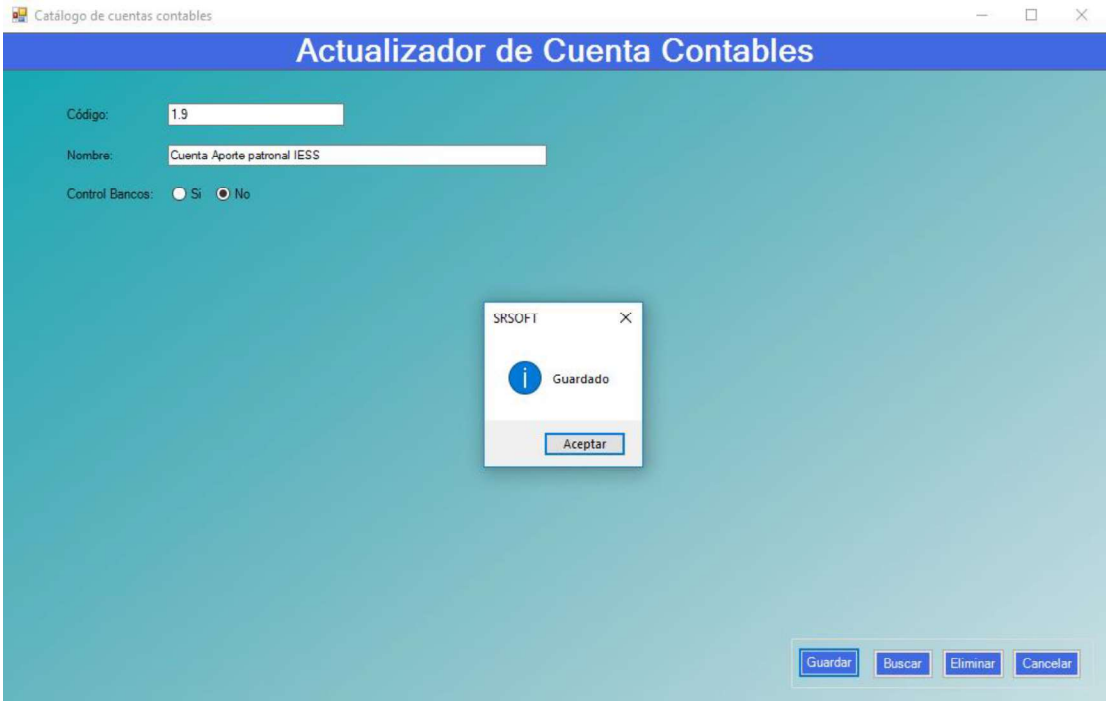

Figura 3.4 Formulario para la gestión de cuentas contables.

Figura 3.5 Formulario para la gestión de tipos de descuento.

El cumplimiento de los requerimientos funcionales RF006 y RF007 correspondientes a la gestión de permisos se los puede observar en la Figura 3.6 y Figura 3.7.

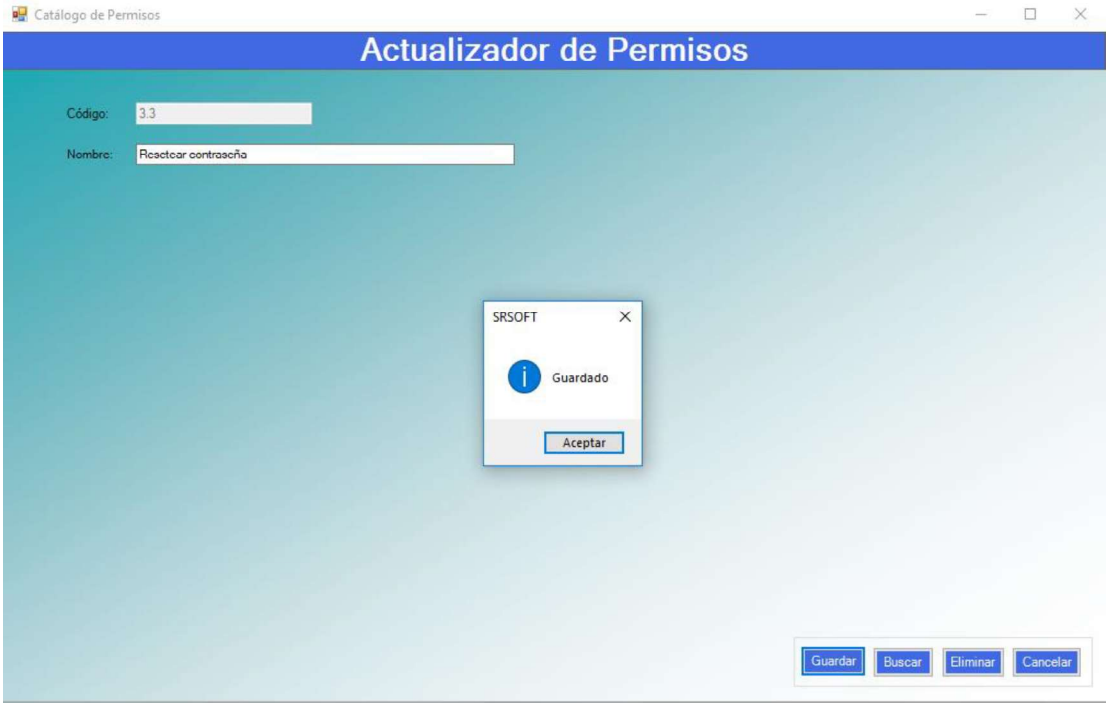

Figura 3.6 Formulario para la gestión de accesos.

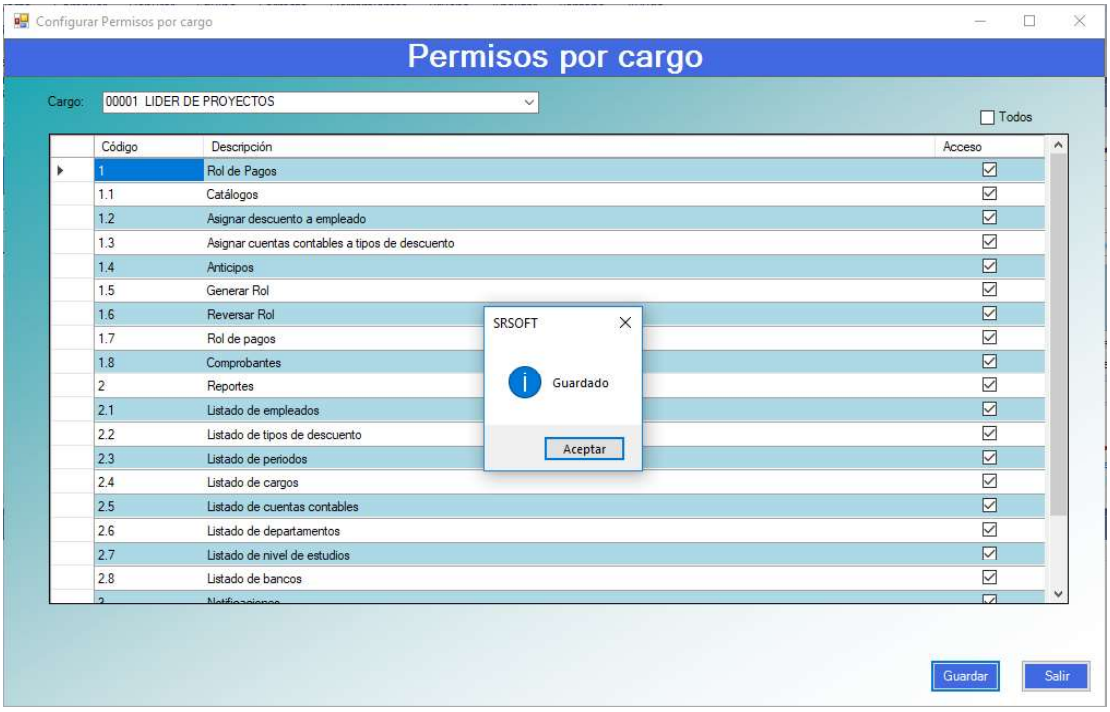

Figura 3.7 Formulario para la configuración de accesos por cargo.

En la Figura 3.8 se muestra el cumplimiento al requerimiento RF008 utilizado para la asignación de tipos de descuento a los empleados.

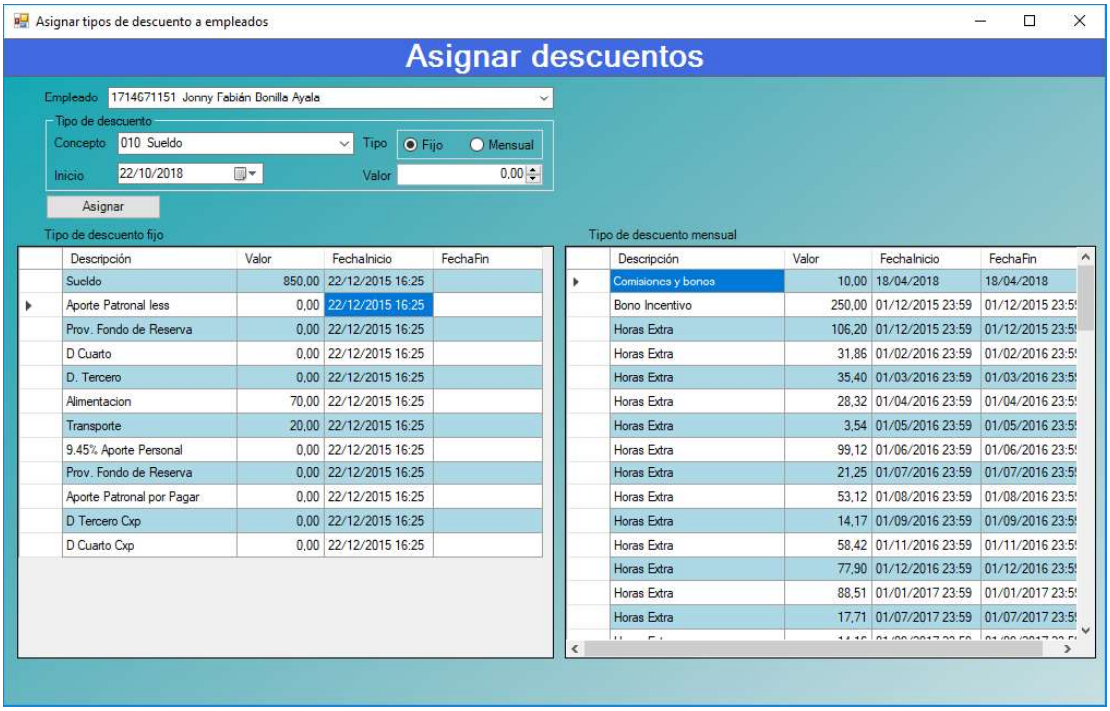

Figura 3.8 Formulario para asignar el tipo de descuento a un empleado.

En la Figura 3.9 se puede observar el cumplimiento que corresponde al requerimiento RF009 para la gestión de periodos contables.

Figura 3.9 Formulario para la gestión de periodos contables.

La prueba que muestra el cumplimiento del requisito funcional RF011 correspondiente a la asignación de cuentas contables a los tipos de descuento se lo muestra en la Figura 3.10.

Figura 3.10 Formulario para la asignación de cuentas a los tipos de descuento.

En la Figura 3.11 se puede observar el cumplimiento del requerimiento funcional RF012 correspondiente al registro de anticipos que solicita el empleado.

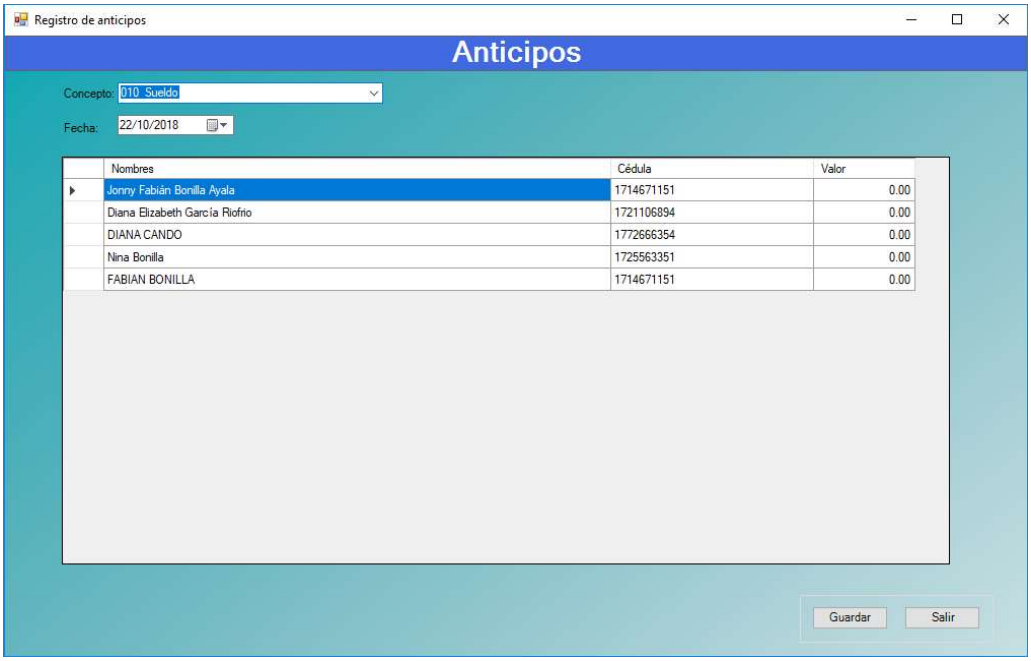

Figura 3.11 Formulario para el registro de anticipos del empleado.

El cumplimiento al requerimiento RF010 y RF013 correspondiente a la generación del rol de pagos se lo puede visualizar en la Figura 3.12. Para el ejemplo se lo realizó para el periodo abril del 2018.

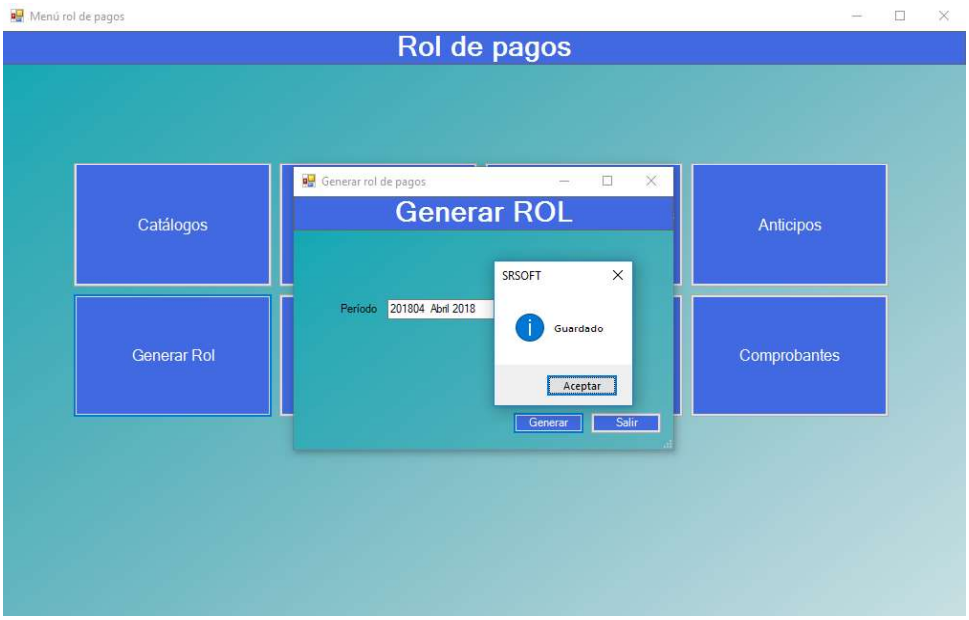

Figura 3.12 Formulario para la generación del rol de pagos de un período.

Además, en la Figura 3.13 y Figura 3.14 se puede observar el resultado del rol general y provisión para el mes de abril del 2018 obtenido mediante el proceso de generación de roles de pago mostrado anteriormente.

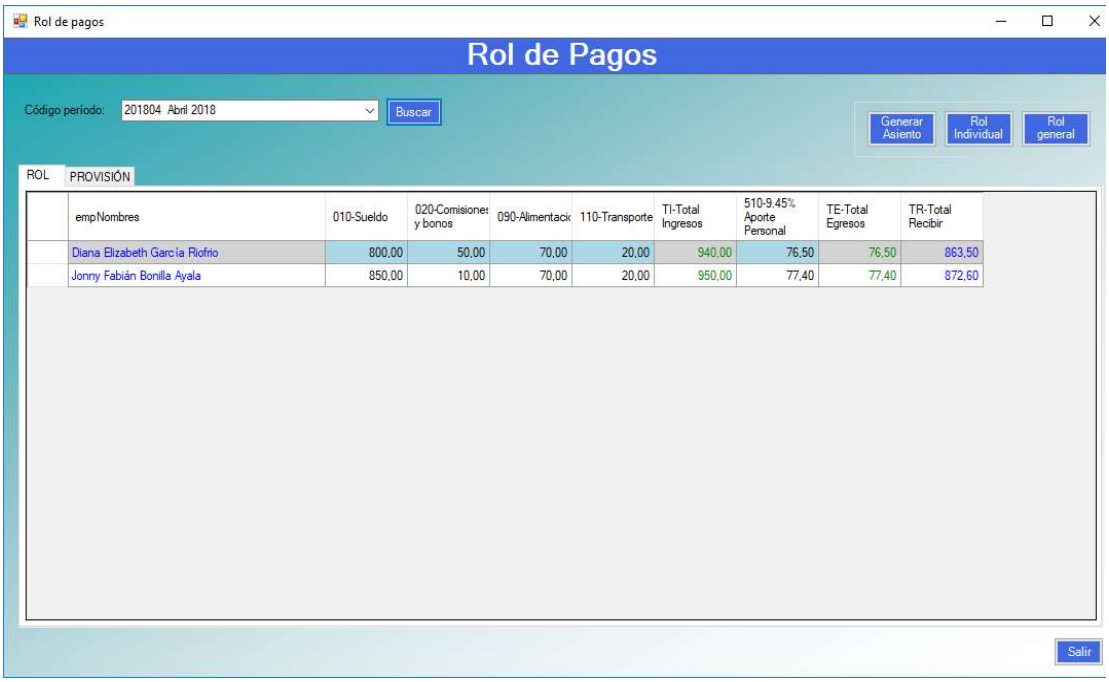

Figura 3.13 Formulario para la visualización de los roles de pago (Rol general).

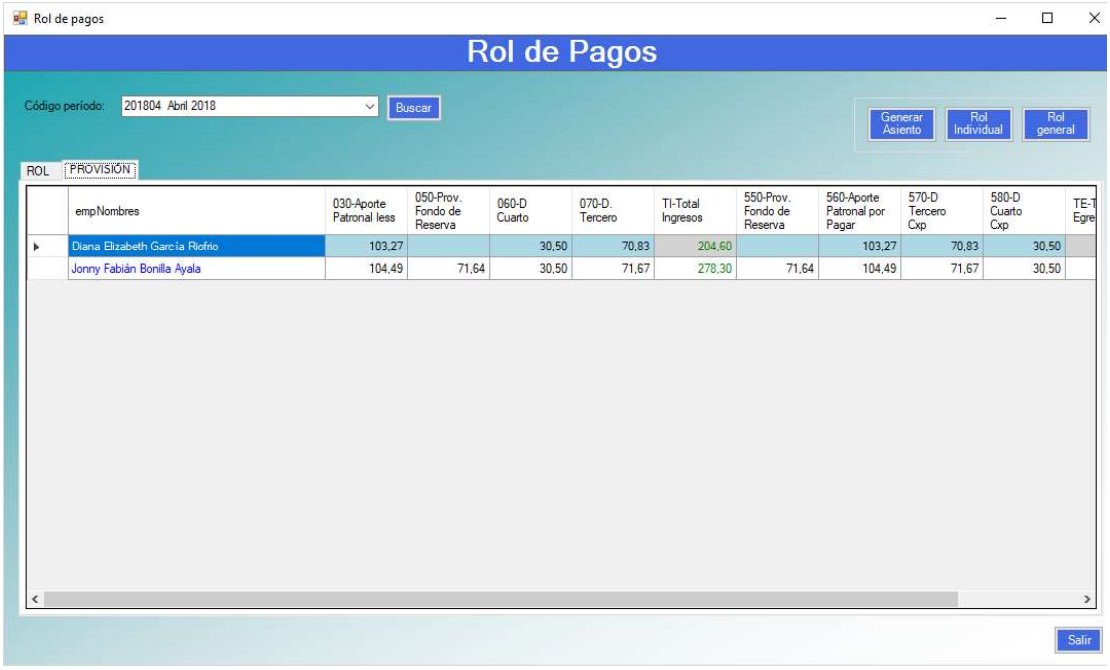

Figura 3.14 Formulario para la visualización de los roles de pago (Rol provisión).

En la Figura 3.15 se muestra el cumplimiento del requerimiento funcional RF014 en el cual se solicita la generación de asientos contables.

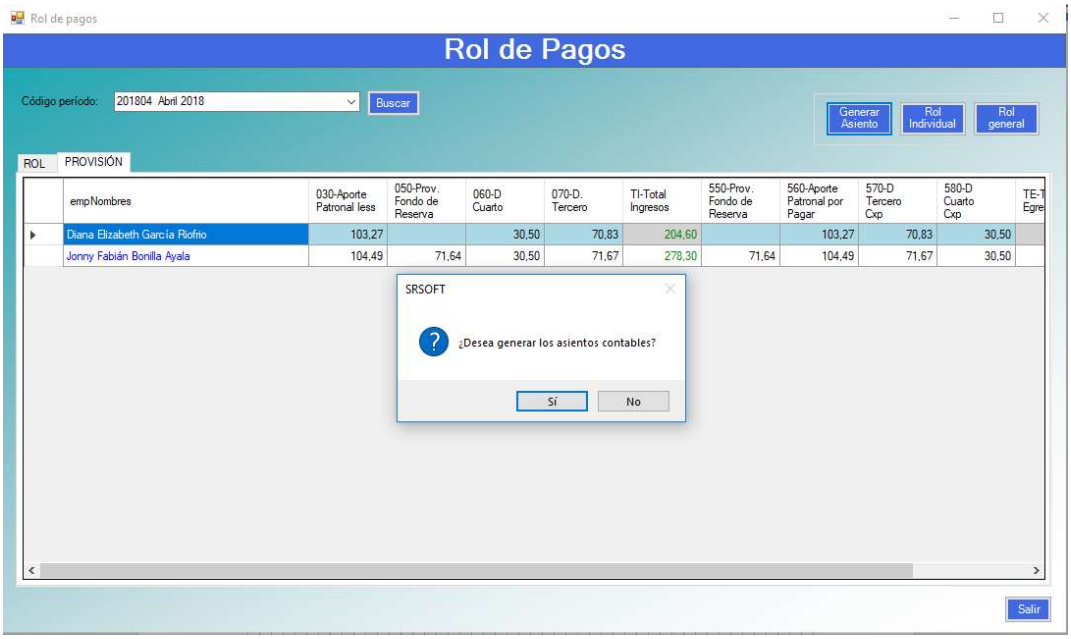

Figura 3.15 Opción para generar los asientos contables de un rol de pagos.

A continuación, en la Figura 3.16 se muestra el formulario en el cual se puede consultar los comprobantes contables generados anteriormente.

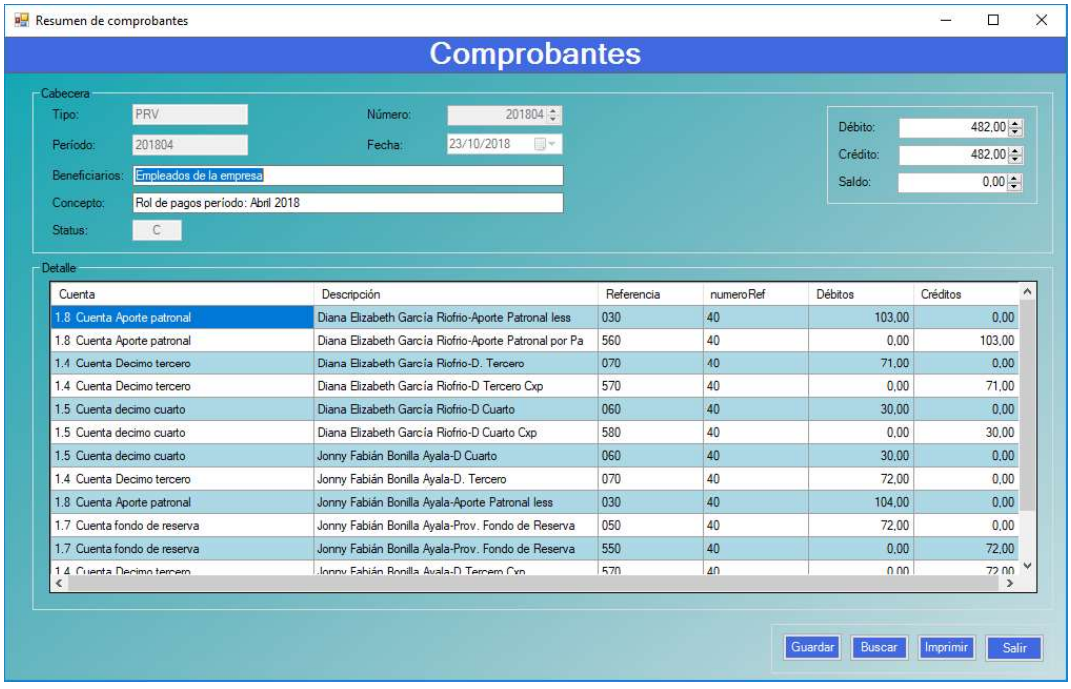

Figura 3.16 Formulario para la consulta de comprobantes.

Por otra parte, en la Figura 3.17 y Figura 3.18 se muestra el cumplimiento del requerimiento RF015 correspondiente a las notificaciones enviadas a los teléfonos móviles.

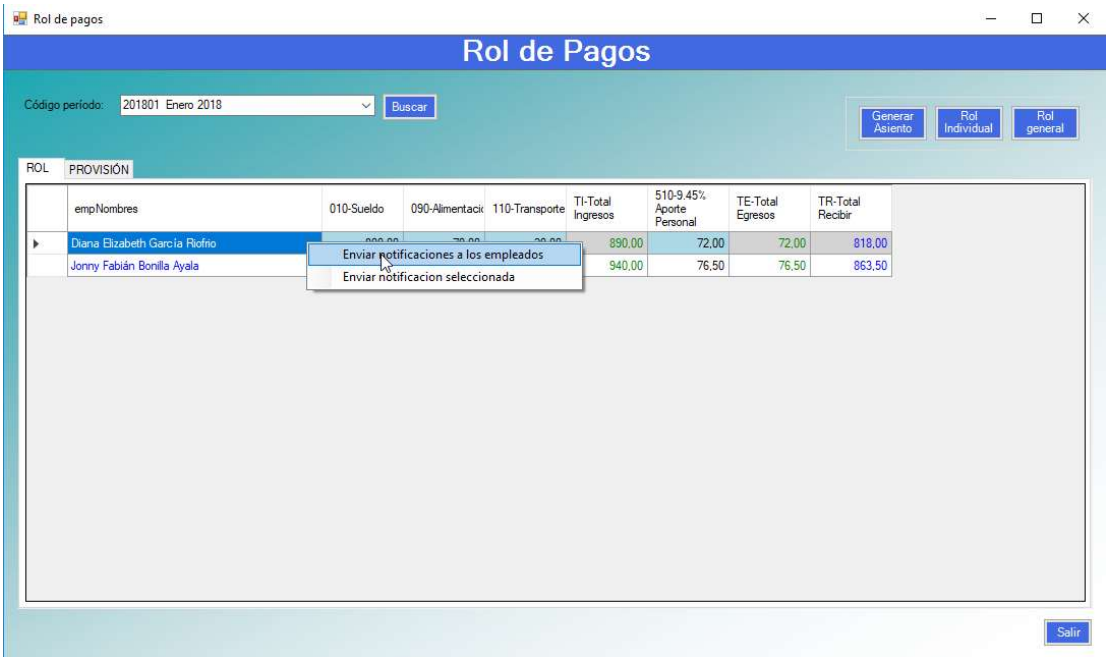

Figura 3.17 Opción para enviar notificaciones a los empleados.

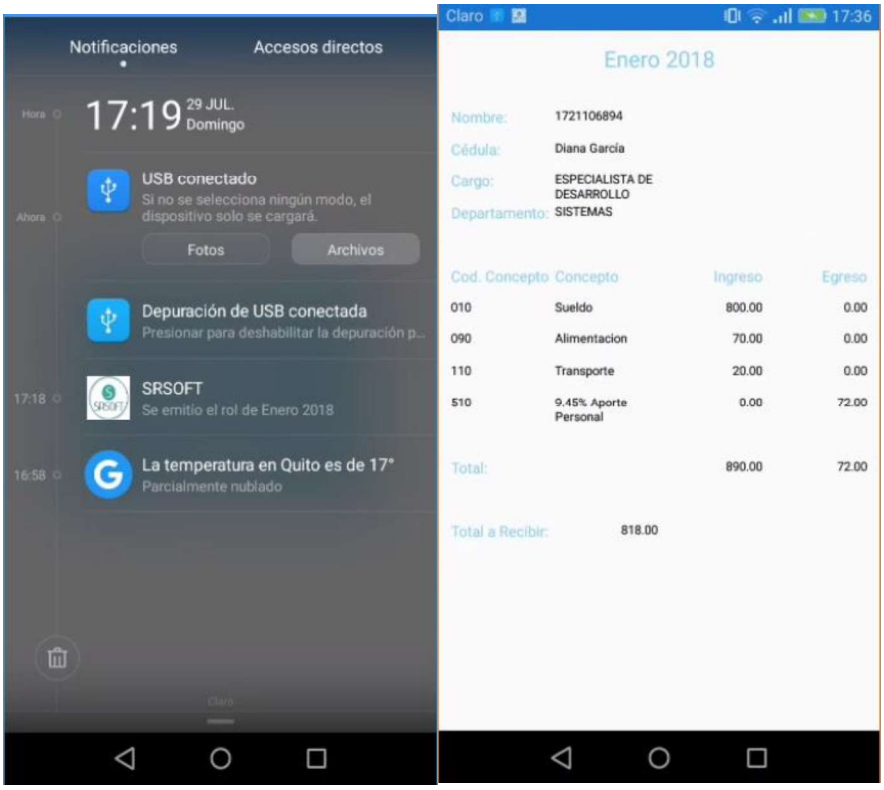

Figura 3.18 Notificación del rol de pagos visualizada en el teléfono móvil.

En la Figura 3.19 se muestra el cumplimiento del requerimiento funcional RF016 correspondiente a la impresión del rol de pagos individual.

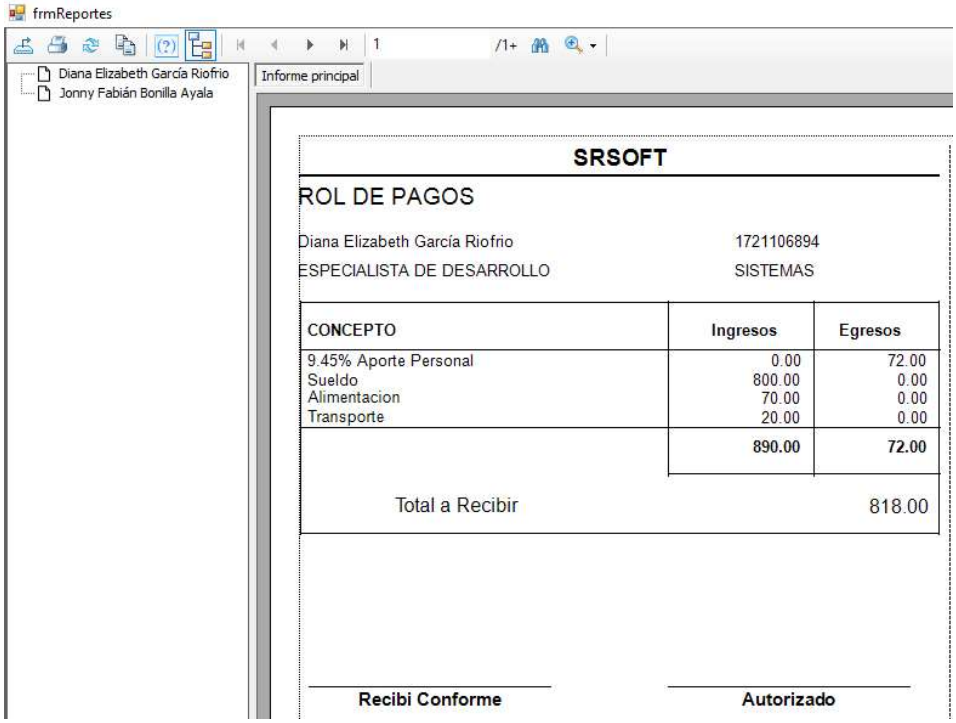

Figura 3.19 Impresión del rol de pagos individual.

En la Figura 3.20 se muestra el cumplimiento del requerimiento funcional RF017 correspondiente a la impresión del rol general.

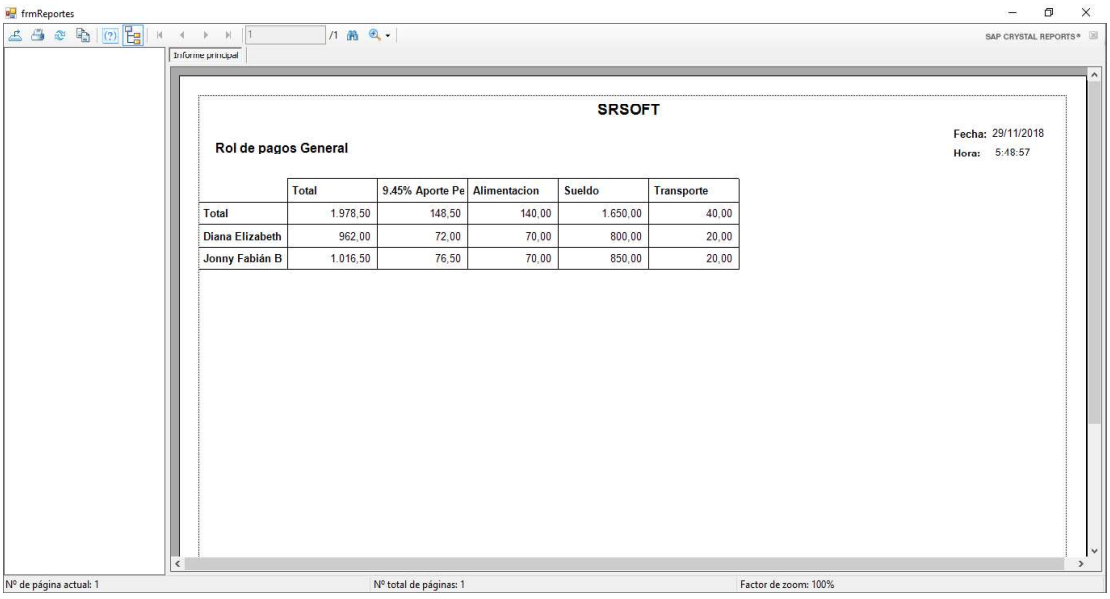

Figura 3.20 Impresión del rol de pagos general para un periodo seleccionado.

En la Figura 3.21 se muestra el cumplimiento del requerimiento funcional RF018 correspondiente a la impresión de la ficha del empleado.

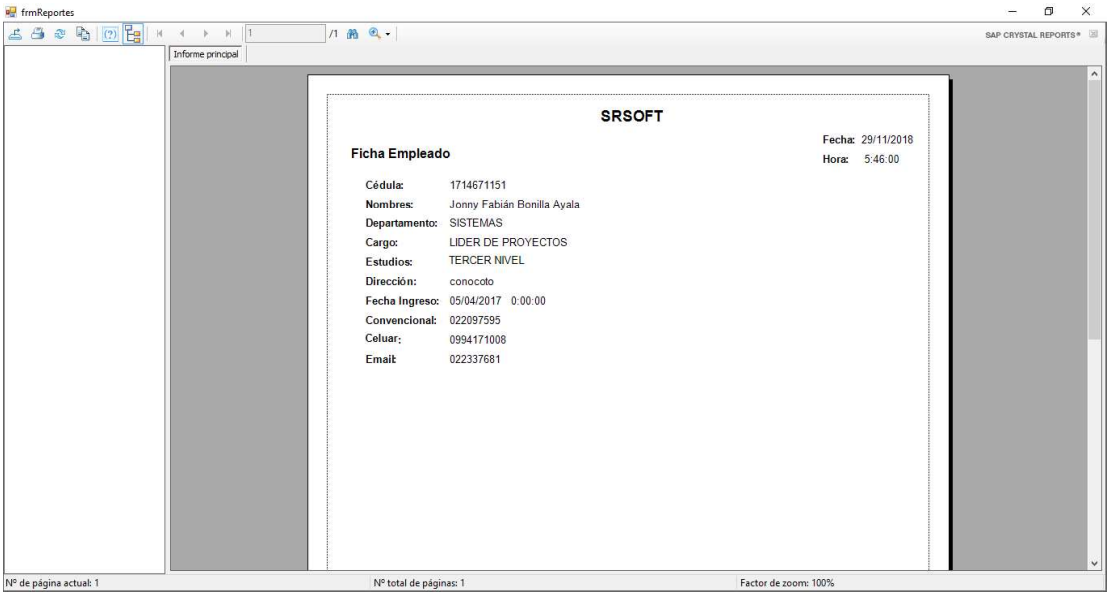

Figura 3.21 Impresión la ficha del empleado.

En la Figura 3.22 se muestra el cumplimiento del requerimiento funcional RF019 en el cual se obtiene un listado con los datos generales de todos los empleados que trabajan en relación de dependencia con la empresa.

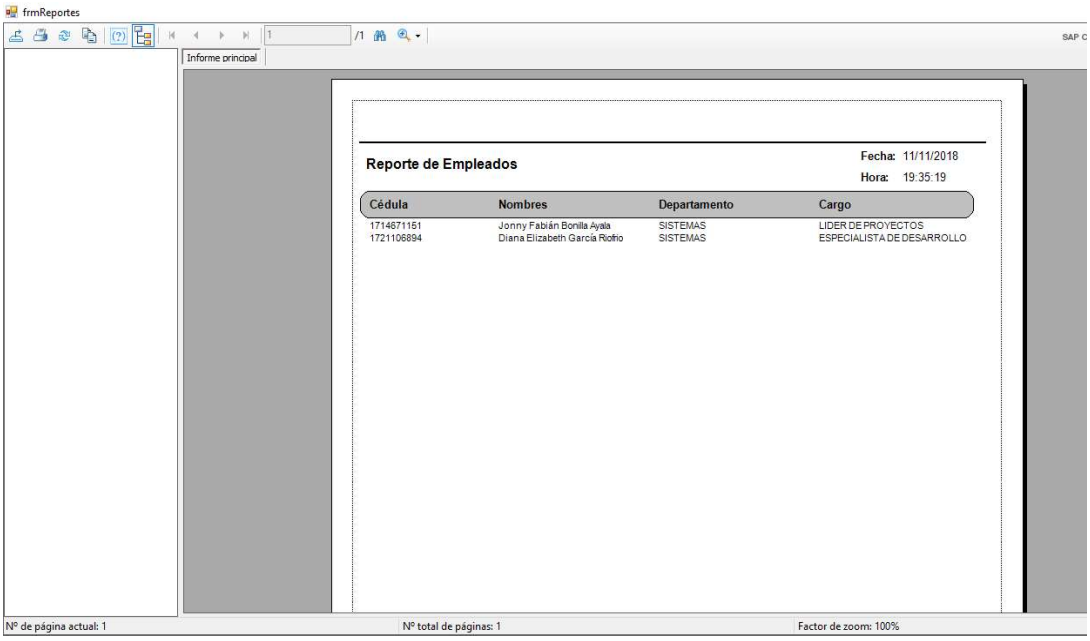

Figura 3.22 Reporte de empleados registrados en la empresa.

En la Figura 3.23 se muestra el cumplimiento del requerimiento funcional RF020 correspondiente a la lista de cargos de la empresa.

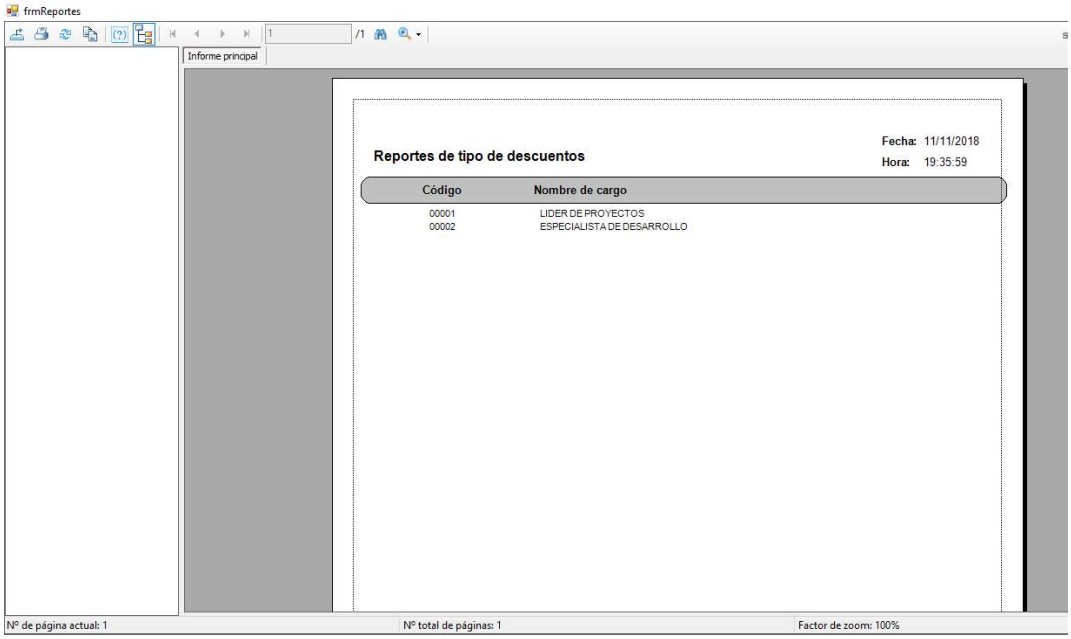

Figura 3.23 Reporte de cargos existentes en la empresa.

En la Figura 3.24 se muestra el cumplimiento del requerimiento funcional RF021 correspondiente a la lista de departamentos que dispone la empresa.

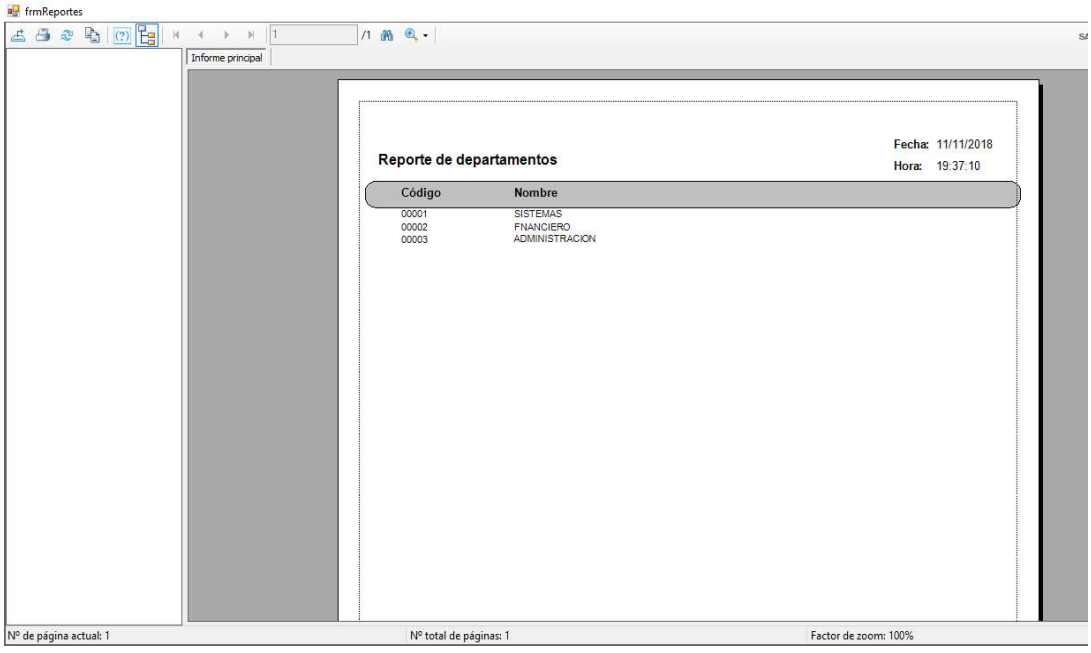

Figura 3.24 Reporte de departamentos existentes en la empresa.

En la Figura 3.25 se muestra el cumplimiento del requerimiento funcional RF022 correspondiente a la lista de cuentas contables de la empresa.

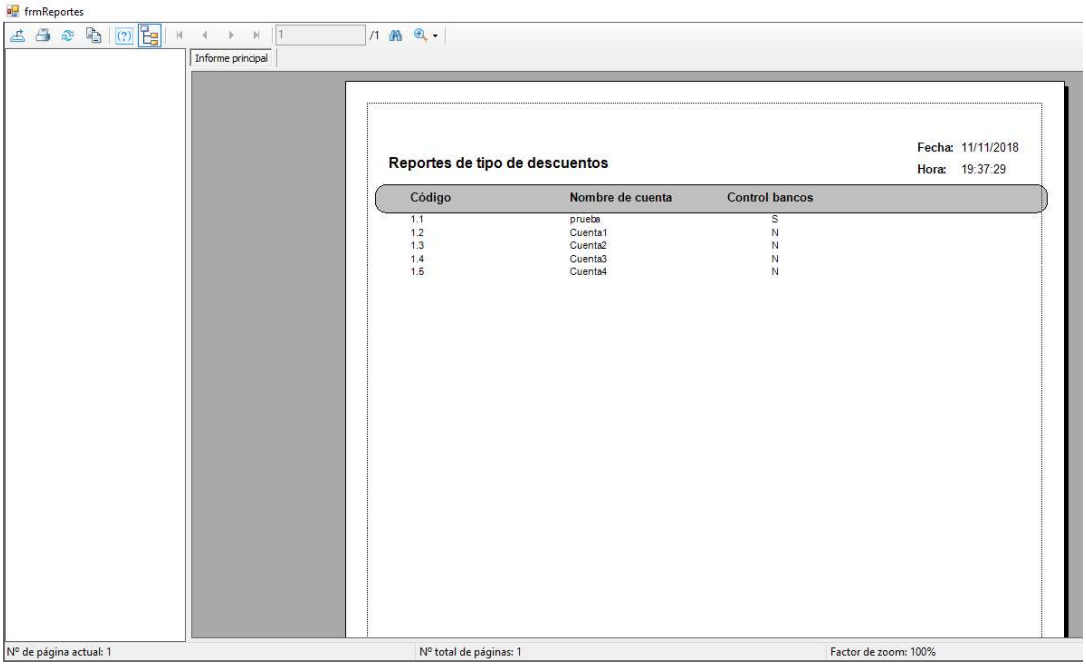

Figura 3.25 Reporte de cuentas contables de la empresa.

En la Figura 3.26 se muestra un reporte adicional muy necesario para la verificación de periodos contables creados.

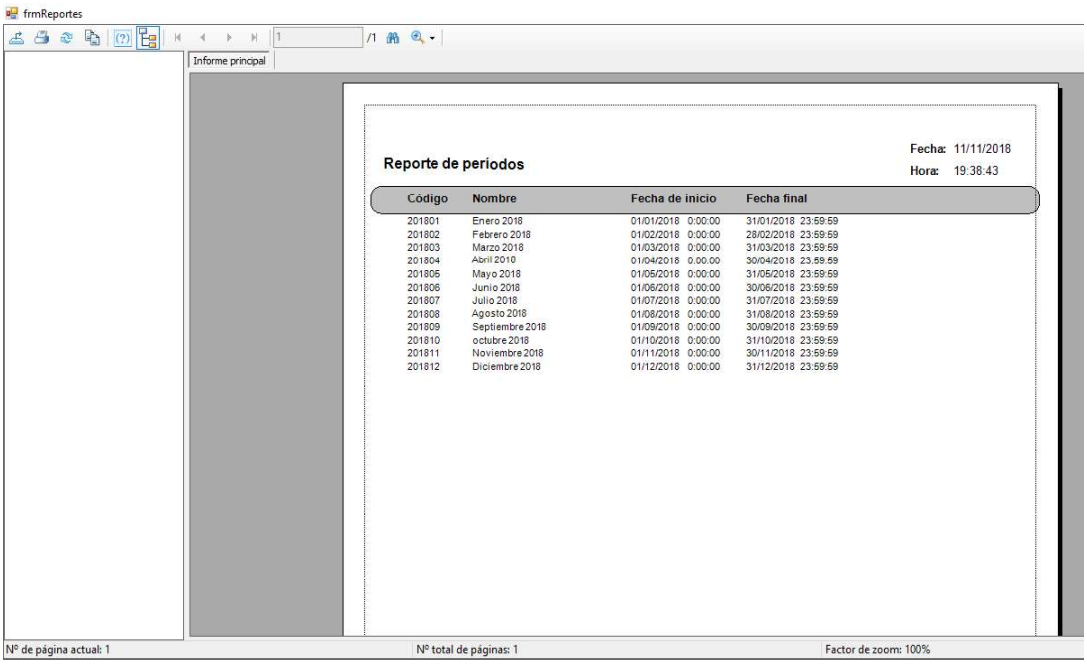

Figura 3.26 Reporte de periodos creados en el sistema.

Finalmente, en la **Tabla 3.2** se muestra la lista de requerimientos funcionales con su respectiva confirmación de cumplimiento y el tiempo en el que se lo implementó.

Tabla 3.2 Lista de cumplimiento de los requerimientos funcionales.

RF	Descripción	Cumple	Tiempo (H)
RF001	Ingresar, modificar y eliminar la ficha del empleado.	Si	8
RF002	Creación nuevos cargos, de igual manera se puede modificar y eliminar en el caso que el usuario requiera.	Si	8
RF003	Creación nuevos departamentos, de igual manera se puede modificar y eliminar en el caso que el usuario requiera.	Si	8
RF004	Creación nuevas cuentas contables, de igual manera se puede modificar y eliminar en el caso que el usuario requiera.	Si	8
RF005	Creación nuevos tipos de descuento, de igual manera se puede modificar y eliminar en el caso que el usuario requiera.	Si	8
RF006	Configurar los permisos de acceso dependiendo del cargo.	Si	10
RF007	Configurar el acceso a los diferentes módulos del sistema dependiendo del cargo.	Si	10
RF008	Permitir la asignación de tipos de descuento a cada empleado dependiendo del grupo contable al que pertenece.	Si	32
RF009	Ingresar los periodos contables los cuales deben cumplir con el formato Año-mes.	Si	12
RF010	Si un periodo se encuentra en estado cerrado, no debe permitir la generación de roles de pago.	Si	8
RF011	Cada tipo de descuento debe estar asociado a una cuenta contable.	Si	24
RF012	Registrar los anticipos solicitados por un empleado.	Si	8
RF013	Generar el rol de pagos general e individual para un periodo.	Si	60
RF014	Generar asientos contables de los roles de pago.	Si	16
RF015	Enviar notificaciones al dispositivo móvil con sistema operativo *Android.*	Si	60
RF016	Imprimir el rol de pagos individual con una copia.	Si	8
RF017	Imprimir el rol de pagos general del periodo seleccionado.	Si	8
RF018	Imprimir la ficha del empleado.	Si	8
RF019	Imprimir un listado de empleados de la empresa.	Si	8
RF020	Imprimir un listado de cargos de la empresa.	Si	8
RF021	Imprimir un listado de departamentos de la empresa.	Si	8
RF022	Imprimir un listado de cuentas contables de la empresa.	Si	8
		Total	**336**

3.2. *SPRINT* PRUEBAS DE REQUERIMIENTOS NO FUNCIONALES

Adicionalmente, se realizó pruebas con la colaboración de personas encargadas de generar los roles de pago, mismos que al final de la prueba se les realizó una encuesta para determinar el nivel de dificultad que tuvieron para realizar ciertas acciones dentro del sistema.

En la Figura 3.27 se puede observar el formato que se utilizó para realizar la evaluación.

DESARROLLO DE UN SISTEMA PARA GESTION DE ROLES DE PAGO CON ENVIO DE NOTIFICACIONES EN TIEMPO REAL

ESCUELA POLITECNICA NACIONAL

Recolección de datos de la prueba de integración

Nombre:

Edad:

Rol:

Escala de dificultad presentada:

0	1	2	3	4	5
Extremadamente fácil	Muy fácil	Fácil	Un poco difícil	Muy difícil	Extremadamente difícil

Conteste:

Descripción	Dificultad
Ingresar, modificar y eliminar la ficha del empleado.	
El sistema debe permitir la creación nuevos cargos, de igual manera se puede modificar y eliminar en el caso que el usuario requiera. Para eliminar un registro no debe existir transacciones realizadas.	
El sistema debe permitir la creación nuevos departamentos, de igual manera se puede modificar y eliminar en el caso que el usuario requiera. Para eliminar un registro no debe existir transacciones realizadas.	
El sistema debe permitir la creación nuevas cuentas contables, de igual manera se puede modificar y eliminar en el caso que el usuario requiera. Para eliminar un registro no debe existir transacciones realizadas.	
El sistema debe permitir la creación nuevos tipos de descuento, de igual manera se puede modificar y eliminar en el caso que el usuario requiera. Para eliminar un registro no debe existir transacciones realizadas.	
El administrador del sistema puede configurar los permisos de acceso dependiendo del cargo que disponga en la empresa.	
Configurar el acceso a los diferentes módulos del sistema dependiendo del cargo que ejerce.	
Permitir la asignación de tipos de descuento a cada empleado dependiendo del grupo contable al que pertenece.	
Ingresar los periodos contables los cuales deben cumplir con el formato Año-mes.	
Si un periodo se encuentra en estado cerrado, no debe permitir la generación de roles de pago.	
Cada tipo de descuento debe estar asociado a una cuenta contable para su registro en los comprobantes. Esta asignación se la debe realizar según su grupo contable.	
Registrar los anticipos solicitados por un empleado.	
Generar el rol de pagos general e individual para un periodo.	
Generar asientos contables de los roles de pago.	
Enviar notificaciones al dispositivo móvil con sistema operativo *Android*.	
Imprimir el rol de pagos individual con una copia.	
Imprimir el rol de pagos general del periodo seleccionado.	
Imprimir la ficha del empleado con toda la información proporcionada en el formulario de ingreso de información.	
Imprimir un listado de empleados de la empresa.	
Imprimir un listado de cargos de la empresa ordenados de manera alfabética de forma ascendente.	
Imprimir un listado de departamentos de la empresa ordenados de manera alfabética de forma ascendente.	

Figura 3.27 Formato para la evaluación a usuarios del sistema.

Basado en las pruebas realizadas anteriormente, se puede realizar la evaluación y cumplimiento de los requerimientos no funcionales de la Tabla 2.27 planteados para el presente Proyecto de Titulación.

3.2.1. AUTENTICACIÓN

Para el uso de la aplicación, cada uno de los usuarios debe proveer sus credenciales de acceso, las cuales serán su número de cedula como **usuario** y una **contraseña** que es creada al momento de registrar la ficha del empleado, ver la Figura 3.1. En la Figura 3.28 y la Figura 3.29 se puede observar el cumplimiento del requerimiento RNF001.

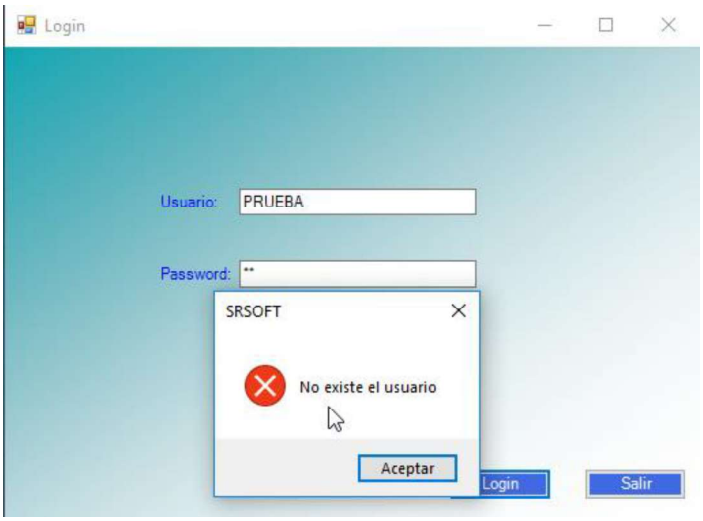

Figura 3.28 Prueba de login fallido.

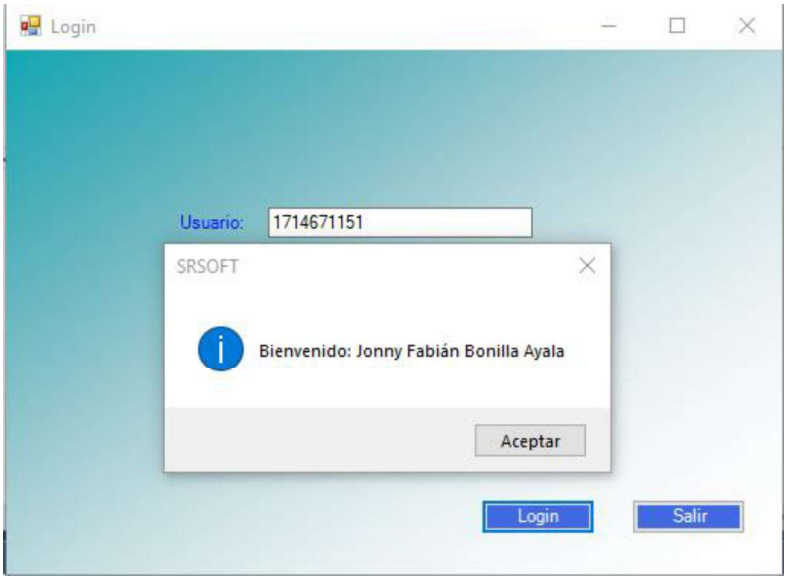

Figura 3.29 Prueba de ingreso de credenciales correctas en el login.

Además, un punto importante que se debe considerar es la manera de custodiar las credenciales de acceso al sistema, por tal motivo en el presente Proyecto de Titulación las contraseñas de los empleados se almacenan utilizando la función *pwdencrypt* de *SQL-Server* [40]. Mediante la función *pwdencrypt* se calcula el *Hash* de la contraseña con el algoritmo Sha1 [41] y se almacena los *Bytes* resultantes. Para realizar la comparación entre el *Hash* almacenado y el *string* que el usuario proporciona se lo realiza con la función *pwdcompare*, dicha función nos retorna el valor de 1 cuando existe la coincidencia, caso contrario nos retornará 0. Desde la Figura 3.30 hasta la Figura 3.32 se muestra el cumplimiento del requerimiento no funcional RNF002 correspondiente a la seguridad.

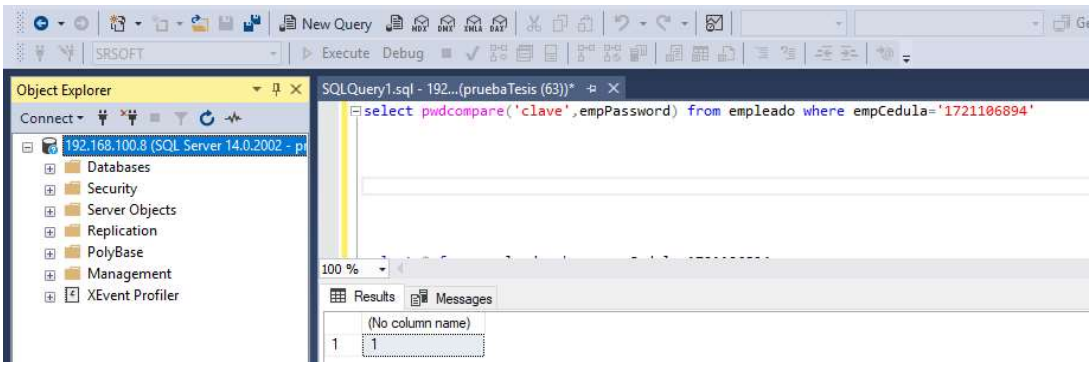

Figura 3.30 Comparación exitosa entre la contraseña y el *Hash* almacenado.

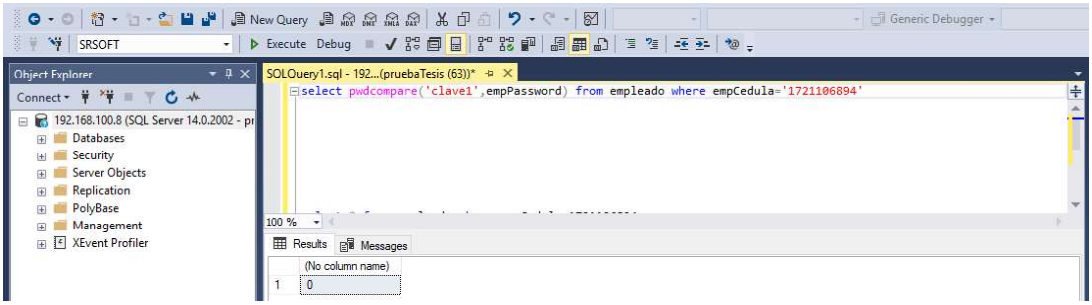

Figura 3.31 Comparación fallida entre la contraseña y el *Hash* almacenado.

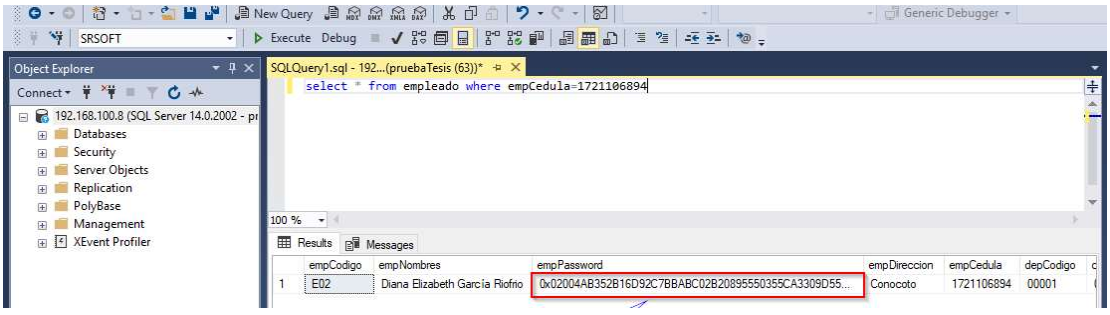

Figura 3.32 Consulta de verificación del Hash almacenado en la base de datos.

3.2.2. SEGURIDAD EN LA COMUNICACIÓN

La información debe ser enviada por un medio de comunicación seguro para garantizar que no sea capturada y visualizada por los programas de análisis de protocolos. Es por esto que, en el presente Proyecto de Titulación se ha implementado el uso de *SSL* para la comunicación entre el Cliente y el Servidor mediante la funcionalidad que ofrece *.Net Remoting*. En la Código 3.1 se muestra la codificación necesaria para habilitar la seguridad.

```
<channels>
  <channel ref="tcp server" port="8001" secure="true" impersonate="true">
      <serverProviders>
        <formatter ref="binary" typeFilterLevel="Full" />
    </serverProviders>
  </channel>
```

Código 3.1 Codificación del *AppConfig* para un canal seguro en *.Net Remoting.*

La Figura 3.33 es un ejemplo de captura de paquetes de datos para el inicio de sesión de un usuario en la aplicación cliente; como se observa en la Figura 3.33 se encuentran cifrados brindado la seguridad necesaria y de esta manera cumple con el requerimiento no funcional RNF002.

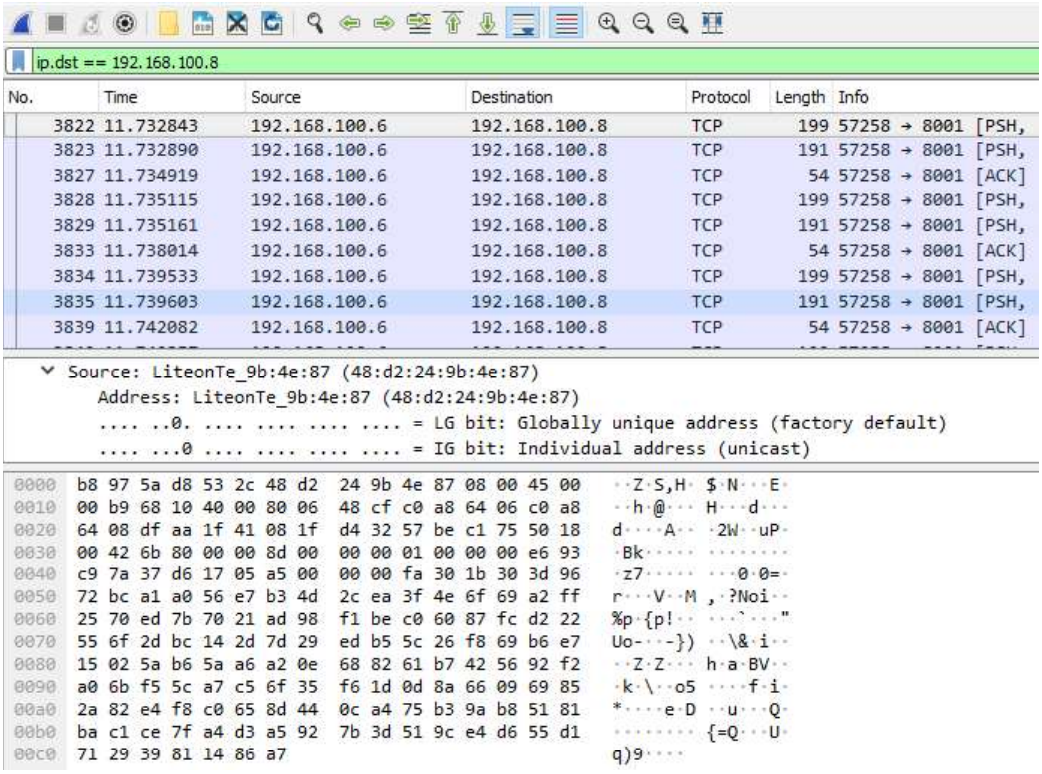

Figura 3.33 Captura de paquetes entre la comunicación de la aplicación cliente y el servidor.

3.2.3. DESEMPEÑO

El presente Proyecto de Titulación utiliza métodos que son ejecutados en la base de datos de manera transaccional, mediante este tipo de ejecuciones se evita que exista cruce de información de los usuarios que deseen guardar en la base de datos de manera concurrente. Desde la Figura 3.34 hasta la Figura 3.36 se puede observar que dos usuarios acceden al mismo tiempo al mismo registro de cargo y lo modifican, pero se registra el cambio de la última persona, dando por cumplido el requerimiento no funcional RNF003.

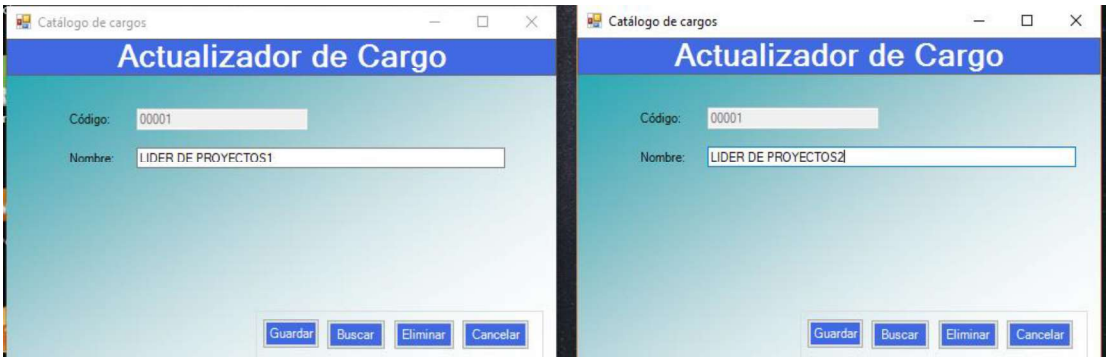

Figura 3.34 Acceso simultaneo al mismo cargo en dos clientes diferentes.

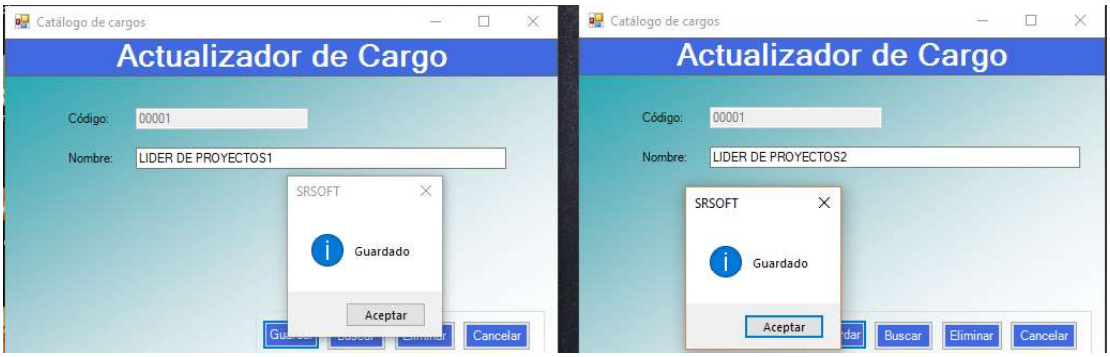

Figura 3.35 Modificación del mismo cargo en la aplicación cliente 1 y 2.

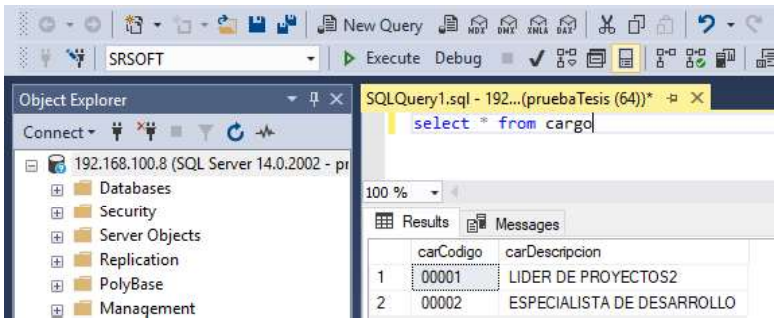

Figura 3.36 Verificación que el registro fue modificado por el aplicativo cliente 2 después de la ejecución simultánea.

3.2.4. SIMPLICIDAD

Una evaluación muy importante que se debe realizar al aplicativo es la facilidad con la cual el usuario puede adaptarse a las funcionalidades que brinda, es por este motivo que se realizó una encuesta a los usuarios que utilizaron el sistema para conocer el nivel de dificultad que tuvo para realizar ciertas operaciones como se puede visualizar en el **ANEXO D**, para esto se basara en la escala entre 0 y 6 donde 0 es considerado como extremadamente fácil, 1 muy fácil, 2 fácil, 3 intermedio, 4 difícil, 5 muy difícil y por ultimo 6 es considerado como extremadamente difícil. En la Tabla 3.3 se observan los promedios de dificultad, los cuales se encuentran alrededor de 2.0 (fácil), por ende, se puede afirmar que está cumplido el requerimiento no funcional correspondiente a la simplicidad.

Tabla 3.3 Promedio de los resultados obtenidos en las pruebas de los usuarios.

Descripción	Dificultad
Ingresar, modificar y eliminar la ficha del empleado.	1,67
Ingresar, modificar y eliminar cargos de la empresa.	1,00
Ingresar, modificar y eliminar la departamentos de la empresa.	1,00
Ingresar, modificar y eliminar cuentas contables.	1,33
Ingresar, modificar y eliminar Tipos de descuento.	1,33
Ingresar, modificar y eliminar los permisos de acceso.	1,67
Configurar el acceso a los diferentes módulos del sistema dependiendo del cargo que ejerce.	1,00
Permitir la asignación de tipos de descuento a cada empleado dependiendo del grupo contable al que pertenece.	1,67
Gestionar los periodos contables.	0,33
Asignar cuentas contables a os tipos de descuento.	0,00
Registrar los anticipos solicitados por un empleado.	1,00
Generar el rol de pagos general e individual para un periodo.	1,00
Generar asientos contables de los roles de pago.	0,00
Enviar notificaciones al dispositivo móvil.	0,33
Imprimir el rol de pagos individual con una copia.	1,00
Imprimir el rol de pagos general.	0,67
Imprimir la ficha del empleado.	1,00
Imprimir un listado de empleados de la empresa.	0,67
Imprimir un listado de cargos de la empresa.	0,67
Imprimir un listado de departamentos de la empresa.	0,67
Imprimir un listado de cuentas contables de la empresa.	0,67

Finalmente, en la Tabla 3.4 se presenta la lista de cumplimiento de los requerimientos no funcionales, también se especifica el tiempo requerido para su implementación y correcto funcionamiento.

Tabla 3.4 Lista de cumplimiento de los requerimientos no funcionales.

Historia de Usuario	Requerimiento no Funcional	Descripción	Cumple	Tiempo (horas)
HU023	RNF001	Realizar la autenticación y acceso a la aplicación mediante usuario y contraseña.	Si	8
HU024	RNF002	La información debe ser transportada de tal manera que no pueda ser visualizada mediante una herramienta que capture y analice paquetes que viajan en la red.	Si	10
HU025	RNF003	Permitir el acceso de varios usuarios a la vez.	Si	10
HU026	RNF004	Los usuarios deben contar con una interfaz amigable y fácil de usar.	Si	10

3.3. ACTUALIZACIÓN DE LOS *SPRINTS*

Una vez terminadas las pruebas, se puede dar por culminado los *Sprints* que se plantearon siguiendo la metodología *Scrum*. En la Tabla 3.5 se puede observar el cumplimiento de cada *Sprint* y además se destaca el tiempo que tomó ejecutar cada uno de ellos.

Tabla 3.5 Lista del cumplimientos de los *Sprints.*

No.	Nombre	Terminado	Tiempo
1	*Sprint* Elaboración del Diagrama de Componentes.	Si	8
2	*Sprint* Flujo de Información.	Si	3
3	*Sprint* Diseño de la Capa de Datos.	Si	15
4	*Sprint* Diseño de la Lógica de Negocio.	Si	22
5	*Sprint* Diseño de la Capa de Presentación.	Si	18
6	*Sprint* Diseño del Aplicativo Móvil.	Si	20
7	*Sprint* Instalación de Herramientas.	Si	8
8	*Sprint* Implementación de la Capa de Datos.	Si	6
9	*Sprint* Implementación de la Lógica de Negocio.	Si	85
10	*Sprint* implementación de la capa de presentación.	Si	45
11	*Sprint* Implementación del Aplicativo Móvil.	Si	60
12	*Sprint* Pruebas del sistema.	Si.	110
		Total	**390**

3.4. ACTUALIZACIÓN DEL *PRODUCT BACKLOG*

En la Tabla 3.6 se puede visualizar el cumplimiento de todos los ítems que conforman el *Product Backlog* del sistema, además se encuentra actualizado con el tiempo real que tomó realizar cada una de sus tareas. Una vez finalizado el *Product Backlog* se puede dar por finalizado de manera satisfactoria la implementación del presente Proyecto de Titulación.

Tabla 3.6 Lista de cumplimiento del *Product Backlog*.

ID	Requerimiento	Cumple	Tiempo
01	Realizar el diagrama de componentes del sistema.	Si	8
02	Diseño de base de datos.	Si	15
03	Diseño de lógica del negocio.	Si	22
04	Diseño de la interfaz de Usuario.	Si	18
05	Diseño del aplicativo móvil	Si	20
06	Instalación de las herramientas para el desarrollo.	Si	8
07	Implementación de la base de datos.	Si	6
08	Implementación de la lógica de negocio.	Si	85
09	Implementación de la presentación	Si	45
10	Instalación y configuración del servidor *Openfire*	Si	3
11	Implementación del aplicativo móvil	Si	60
12	Pruebas de funcionamiento	Si	100
		Total	**390**

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- El objetivo general del presente Proyecto de Titulación fue desarrollar un sistema para la generación de roles de pago con envío de notificaciones en tiempo real al dispositivo móvil, el cual ha sido cumplido de manera satisfactoria mediante el desarrollo de una aplicación que facilita esta tarea y además notifica a los empleados enviándoles un mensaje utilizando el protocolo *XMPP* para que pueda visualizar el detalle de los descuentos realizados en su rol de pagos.

- Uno de los objetivos del presente Proyecto de Titulación fue diseñar un sistema que permita gestionar los roles de pago enviando notificaciones en tiempo real, este objetivo se ha cumplido de manera correcta mediante el uso de diagramas *UML* los cuales nos permitieron representar de manera gráfica la solución al problema planteado inicialmente.

- La implementación de un sistema que permita gestionar los roles de pago y además informar de manera oportuna a los empleados proporciona una solución innovadora en la cual los empleados pueden revisar y corregir su rol de pagos de ser el caso.

- El sistema planteado para el presente Proyecto de Titulación ha sido probado para verificar el cumplimiento de los requerimientos funcionales y no funcionales establecidos en la fase de análisis, y el resultado fue positivo ya que se generaron roles de pago y se enviaron las notificaciones a los dispositivos móviles de manera satisfactoria.

- La tecnología *.Net Remoting* facilita la implementación de aplicaciones distribuidas ya que su configuración es sencilla y además es muy simple poderlo incluir en una aplicación que tenga la necesidad de ejecutar métodos de manera remota.

- El servidor *Openfire* es una herramienta que brinda un gran alcance a las aplicaciones que tienen la necesidad de incluir un protocolo en tiempo real. Además, es una solución recomendable cuando se desea interactuar entre aplicaciones de escritorio, web o móviles debido a que es independiente de la plataforma de desarrollo.

- El sistema desarrollado ayuda a generar roles de pago de manera ágil permitiendo que los contadores de la empresa disminuyan el tiempo empleado para la gestión

de la nómina y además informa de manera oportuna a sus empleados los descuentos que fueron aplicados en el periodo correspondiente.

4.2. RECOMENDACIONES

- Una fase muy importante para realizar un proyecto de cualquier índole es el análisis y planificación, ya que es ahí donde se realiza la investigación de los posibles inconvenientes que puede tener un proyecto para su culminación, ya sea por cuestiones técnicas o de negocio. Por tal motivo es recomendable revisar documentación que cubra dicho tema y seguir los lineamientos que recomiendan algunas metodologías de desarrollo de proyectos.

- Es importante mantener reuniones periódicas con el cliente como lo recomienda la metodología *Scrum*, ya que si el cliente conoce el estado de la implementación del proyecto se evita que existan modificaciones mayores que pueden afectar a los tiempos de desarrollo y entrega del producto.

- Se recomienda utilizar *Visual Studio* en modo de administrador, ya que existen carpetas o librerías del sistema las cuales no podrá acceder si no se inicia de esta manera. Por lo general se accede al proyecto desde el acceso rápido de la solución, y *Visual Studio* ejecuta con los permisos más bajos por defecto.

- En la tecnología *.Net Remoting* existen niveles de seguridad para el acceso a los objetos remotos por parte del cliente, dependiendo de los requerimientos del sistema se puede utilizar *sockets* seguros o no. Además, es importante considerar que el tiempo de respuesta es mayor mientras más seguridad se aplique y por ende pueden existir aplicaciones que no sea recomendable utilizar esta tecnología.

- Es recomendable utilizar protocolos abiertos debido a que es más accesible a la documentación, guías de usuario del mismo y estándares de la arquitectura. *Openfire* es una buena opción para utilizar el protocolo en tiempo real denominado *XMPP* ya que en la comunidad que lo publica existen foros donde se dispone de ayudas para la instalación, facilitando la incorporación de sus funcionalidades en cualquier tipo de proyecto que sea de desarrollo.

- Es importante analizar y establecer la versión mínima de *Android* en la cual va a ejecutarse el aplicativo móvil, esto se lo realiza con el objetivo de no afectar la funcionalidad del aplicativo y además que cubra el mayor número de dispositivos móviles disponibles en el mercado.

- Se recomienda para trabajos futuros realizar un alcance para que el sistema pueda generar archivos *pdf* de los informes disponibles y enviarlos por correo electrónico al empleado. Además, se podría realizar la implementación para dispositivos móviles con sistema operativo *IOS*.
- Otra recomendación para trabajos futuros es realizar la implementación del registro biométrico mediante el sistema, ya que mediante esta información se puede realizar el cálculo directo de las horas extra.
- Por último, se recomienda realizar la implementación del sistema utilizando servicios en la nube con el objetivo de disminuir costos y aumentar la disponibilidad del servicio, para lo cual es muy importante realizar un análisis del ancho de banda que consume la aplicación.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] L. Caguana, " Rol de pagos", *contabilidad*, 31-oct-2012.

[2] "¿Qué son los asientos contables? - El Contador". [En línea]. Disponible en: http://elcontador.net/que-son-los-asientos-contables/. [Accedido: 09-feb-2017].

[3] "Las cuentas contables y su clasificación - El Contador". [En línea]. Disponible en: http://elcontador.net/las-cuentas-contables-clasificacion/. [Accedido: 11-ene-2017].

[4] "*.Net Remoting Framework*". [En línea]. Disponible en: https://www.scss.tcd.ie/~ebarrett/lectures/cs7051/pdfs/dotnet1.pdf. [Accedido: 11-ene-2017].

[5] M. Ingo Rammer, *Advance .NET Remoting, Second Edition*. Apress, 2005.

[6] "An Introduction to Microsoft .NET Remoting Framework". [En línea]. Disponible en: https://msdn.microsoft.com/en-us/library/ms973864.aspx. [Accedido: 20-nov-2018].

[7] Matthew MacDonald, *Distributed applications: Integrating XML Web services and .NET Remoting.* Microsoft Press, 2003.

[8] "Plantilla para la confección de informes de laboratorio - Protocolo_XMPP.pdf". [En línea]. Disponible en: http://profesores.elo.utfsm.cl/~agv/elo322/1s13/project/reports/Protocolo_XMPP.pdf. [Accedido: 09-feb-2017].

[9] "Indicar direcciones Jabber con xmpp: | JabberES". [En línea]. Disponible en: https://www.jabberes.org/node/884/. [Accedido: 20-nov-2018].

[10] C. Kaes, "Definition of Jabber Identifiers (JIDs)". [En línea]. Disponible en: https://xmpp.org/extensions/xep-0029.html. [Accedido: 20-nov-2018].

[11] "XMPP | FAQ". [En línea]. Disponible en: https://xmpp.org/about/faq.html. [Accedido: 20-nov-2018].

[12] "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence". [En línea]. Disponible en: https://xmpp.org/rfcs/rfc3921.html. [Accedido: 20-nov-2018].

[13] P. Saint-Andre, "Streaming XML with Jabber/XMPP", *IEEE Internet Comput.*, vol. 9, n.º 5, pp. 82-89, sep. 2005.

[14] "Ignite Realtime: About". [En línea]. Disponible en: https://www.igniterealtime.org/about/index.jsp. [Accedido: 26-sep-2018].

[15] "Ignite Realtime: Openfire Server". [En línea]. Disponible en: https://www.igniterealtime.org/projects/openfire/. [Accedido: 09-feb-2017].

[16] "Scrum Guide Downloads | Scrum Guides". [En línea]. Disponible en: https://www.scrumguides.org/download.html. [Accedido: 30-nov-2018].

[17] "Proceso y Roles de Scrum", *www.softeng.es*. [En línea]. Disponible en: https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum/proceso-roles-de-scrum.html. [Accedido: 20-nov-2018].

[18] "What is SCRUM? - CodeProject". [En línea]. Disponible en: https://www.codeproject.com/Articles/4798/What-is-SCRUM. [Accedido: 20-nov-2018].

[19] "Metodología Scrum". [En línea]. Disponible en: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf. [Accedido: 20-nov-2018].

[20] "Dark Data | Zabalit. Consultoría, diseño y desarrollo de apps". [En línea]. Disponible en: http://zabalit.com/metodologia-scrum/. [Accedido: 30-may-2019].

[21] "Historia de usuario - Scrum Manager BoK". [En línea]. Disponible en: http://www.scrummanager.net/bok/index.php/Historia_de_usuario. [Accedido: 26-sep-2018].

[22] "Android", *Android*. [En línea]. Disponible en: https://www.android.com/. [Accedido: 20-nov-2018].

[23] "Requerimientos no funcionales y arquitectura de software", J.F. Castillo. [En línea]. Disponible en: https://www.cimat.mx/Eventos/setys2009/jfcastillo.pdf. [Accedido: 20-nov-2018].

[24] J. P. Quiroga, "Requerimientos Funcionales y No Funcionales, p. 27.

[25] "UML: Diagrama de Secuencia – INGENIERÍA DEL SOFTWARE". [En línea]. Disponible en: https://ingsotfwarekarlacevallos.wordpress.com/2015/07/07/uml-diagrama-de-secuencia/. [Accedido: 26-sep-2018].

[26] "Modelo entidad-relación", 17-feb-2017. [En línea]. Disponible en: https://desarrolloweb.com/articulos/modelo-entidad-relacion.html. [Accedido: 17-feb-2017].

[27] "Diagramas de clases de UML: Instrucciones". [En línea]. Disponible en: https://msdn.microsoft.com/es-es/library/dd409416.aspx. [Accedido: 26-sep-2018].

[28] "SQL Server 2017 Express Edition | Microsoft". [En línea]. Disponible en: https://www.microsoft.com/en-us/sql-server/sql-server-editions-express. [Accedido: 26-sep-2018].

[29] "Download SQL Server Management Studio (SSMS)". [En línea]. Disponible en: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms. [Accedido: 26-sep-2018].

[30] "Visual Studio Express | Ahora, Visual Studio Community". [En línea]. Disponible en: https://visualstudio.microsoft.com/es/vs/express/. [Accedido: 26-sep-2018].

[31] "Instalación de Xamarin para Visual Studio - Visual Studio". [En línea]. Disponible en: https://docs.microsoft.com/es-es/visualstudio/cross-platform/setup-and-install. [Accedido: 26-sep-2018].

[32] "Xamarin.Android - Xamarin". [En línea]. Disponible en: https://docs.microsoft.com/en-us/xamarin/android/. [Accedido: 26-sep-2018].

[33] "Ignite Realtime: Openfire Server", 09-feb-2017. [En línea]. Disponible en: https://www.igniterealtime.org/projects/openfire/. [Accedido: 09-feb-2017].

[34] "SQL error codes - database connection error". [En línea]. Disponible en: https://docs.microsoft.com/en-us/azure/sql-database/sql-database-develop-error-messages. [Accedido: 11-nov-2018].

[35] "SAP Crystal Reports, developer version for Microsoft Visual Studio | Crystal Solutions", *SAP*. [En línea]. Disponible en: https://www.sap.com/products/crystal-visual-studio.html. [Accedido: 20-nov-2018].

[36] "SQLite Home Page". [En línea]. Disponible en: https://sqlite.org/index.html. [Accedido: 26-sep-2018].

[37] "NuGet Gallery | Home". [En línea]. Disponible en: https://www.nuget.org/. [Accedido: 20-nov-2018].

[38] "Sharp.Xmpp". [En línea]. Disponible en: https://github.com/pgstath/Sharp.Xmpp. [Accedido: 20-nov-2018].

[39] "NuGet Gallery | Xam.Plugins.Notifier 3.0.1". [En línea]. Disponible en: https://www.nuget.org/packages/Xam.Plugins.Notifier/. [Accedido: 20-nov-2018].

[40] "PWDENCRYPT (Transact-SQL)". [En línea]. Disponible en: https://docs.microsoft.com/en-us/sql/t-sql/functions/pwdencrypt-transact-sql. [Accedido: 11-nov-2018].

[41] "HASHBYTES (Transact-SQL)". [En línea]. Disponible en: https://docs.microsoft.com/en-us/sql/t-sql/functions/hashbytes-transact-sql. [Accedido: 11-nov-2018].

ANEXOS

ANEXO A: Encuestas rol de pagos.

ANEXO B: Descripción de las entidades del sistema.

ANEXO C: Codificación de la base de datos.

ANEXO D: Encuestas pruebas del sistema.

ANEXO E: Código del sistema.

NOTA: *Los Anexos se encuentran adjuntos en el CD.*

ORDEN DE EMPASTADO