ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO DE SISTEMA DE PAGO AUTOMÁTICO DE COMPRAS MEDIANTE UNA APLICACIÓN MÓVIL PARA UNA CAJA RÁPIDA CON TECNOLOGÍA RFID

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

SAMUEL DANIEL HWANG SARANGO

daniel_92ec@hotmail.com

DANIEL VINICIO TENEMAZA HURTADO

d.v.t.h@hotmail.com

DIRECTOR: M.Sc. FELIPE LEONEL GRIJALVA ARÉVALO

felipe.grijalva@epn.edu.ec

CODIRECTOR: M.Sc. FRANKLIN LEONEL SÁNCHEZ CATOTA

franklin.sanchez@epn.edu.ec

Quito, agosto 2018

AVAL

Certificamos que el presente trabajo fue desarrollado por Samuel Daniel Hwang Sarango y Daniel Vinicio Tenemaza Hurtado, bajo nuestra supervisión.

M.Sc. FELIPE LEONEL GRIJALVA ARÉVALO
DIRECTOR DEL TRABAJO DE TITULACIÓN

M.Sc. FRANKLIN LEONEL SÁNCHEZ CATOTA
CODIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Nosotros, Samuel Daniel Hwang Sarango y Daniel Vinicio Tenemaza Hurtado, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

SAMUEL DANIEL HWANG SARANGO	DANIEL VINICIO TENEMAZA HURTADO

DEDICATORIA

A Dios, por amarme más allá de mi entendimiento “porque de Él, y por Él, y para Él, son todas las cosas. A él sea la gloria por los siglos.”

A Isabel mi ejemplo de vida, con amor eterno.

DANIEL HWANG

DEDICATORIA

A mi madre Sari, quien es una guerrera en todo sentido, quien da tanto amor a sus hijos que parece que sus fuerzas son infinitas, usted que jamás se dejó vencer por una enfermedad y luchó cada día por estar junto a nosotros, esto es por usted mi madrecita amada, nos costó mucho pero llegamos a la meta.

Al ser más puro y especial que mi hermana trajo a este mundo, Alisson tú que eres y serás siempre la luz que ilumina nuestras vidas, sin saberlo nos has unido aún más, con esa hermosa sonrisa que nos das cada día.

A mi torbellino Abigail, recuerda siempre que tú llegarás tan lejos como quieras hacerlo, el cielo es el límite, y por si alguna vez lees estas palabras, tú nunca estarás sola, tu padrino está aquí para ti.

Daniel Tenemaza

AGRADECIMIENTO

A lo más hermoso que tengo en la vida, mis padres Cesar y Sari, quienes son mis ángeles en Tierra, ustedes son mis súper héroes, a quienes les debo todo y jamás me alcanzará la vida para agradecer por tan hermoso e infinito amor, gracias por enseñarme a nunca darme por vencido, aun cuando las cosas no fueron fáciles jamás dejaron de creer en mí, incluso cuando yo mismo deje de hacerlo, les amo infinitamente.

A mis hermanos Johanna, Andrés, Wlady, Ángelo, quienes siempre han estado para mí con una palabra, un abrazo, un gesto de amor y cariño, gracias por estar a mi lado cada instante y demostrarme que lo más especial que existe, es la familia que somos.

A mi querido padrino Ernesto, quien estuvo ahí con sus palabras que llenaban mi corazón, dando ese aliento de lucha y perseverancia, recordándonos los planes maravillosos que siempre tiene Dios para nosotros.

A quienes se convirtieron en parte de mi familia, Alejo, Pablo, Freddy, ustedes permitieron que esta etapa universitaria fuese una aventura completa, tantos recuerdos que quedan en mi memoria son gracias a ustedes y su cariño.

A mi compañero de proyecto de titulación por mantenerse firme, aun cuando parecía que todo esto se venía abajo.

En especial a Dios, ya que, sin Él, nada es posible de realizar, y que ha iluminado mi vida con todos los hermosos seres que tengo junto a mí.

Daniel Tenemaza

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

CAPÍTULO 1

CAPÍTULO 2

CAPÍTULO 3

ÍNDICE DE TABLAS

CAPÍTULO 1

CAPÍTULO 2

CAPÍTULO 3

ÍNDICE DE CÓDIGO

CAPÍTULO 2

RESUMEN

El presente proyecto comprende el desarrollo de un prototipo de sistema de pago automático de compras para las cajas rápidas de un supermercado. Este prototipo utiliza un sistema RFID para la lectura de los productos y una aplicación Android para la realización del pago. Además, incluye el desarrollo de una aplicación web ASP.NET para la administración del supermercado.

El prototipo está compuesto por cuatro componentes esenciales: la base de datos, la caja electrónica, la aplicación Android y la aplicación web.

En primer lugar, la base de datos almacena la información del supermercado, (e.g. los productos). Segundo, la aplicación web permite la administración de las cuentas de los clientes, la gestión de productos y la generación de reportes de venta. Tercero, la caja está compuesta por un computador conectado a un interrogador RFID, el cual permite la lectura de los productos a ser adquiridos. Por último, la aplicación móvil Android funciona como billetera electrónica permitiendo al cliente la visualización de los detalles de su compra y el pago de la misma.

Cabe destacar que los resultados obtenidos en las pruebas de funcionalidad y las estimaciones del tiempo medio de compra verifican la utilidad del prototipo como sistema alternativo y complementario al modelo tradicional de cobro mediante código de barras.

PALABRAS CLAVE: RFID, Android, ASP.NET

ABSTRACT

This final career project develops an automatic payment prototype system for express checkout counters at supermarkets. This prototype uses RFID technology to read the products and an Android application for making the payment. It also includes the development of an ASP.NET web application for supermarket management.

The system consists of four essential components: the database, the express checkout counter, the Android application and the web application.

First, the database stores the information of the supermarket (e.g. the products). Second, the web application allows the management of customer accounts, product management and the generation of sales reports. Third, the checkout counter is composed of a computer connected to an RFID reader to scan the products to be acquired. Finally, the Android mobile application works as an electronic wallet allowing the customers to view the details of their purchases.

It should be noted that functionality tests and average purchase time estimates show the usefulness of this prototype as an alternative and complementary system to the traditional barcode scanning model.

KEYWORDS: RFID, Android, ASP.NET

ACRÓNIMOS Y ABREVIATURAS

API	Application Programming Interface
CDROM	Compact Disc Read Only Memory
DOS	Disk Operating Systems
HF	High Frequency
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JDBC	Java Data Base Connectivity
LAN	Local Area Network
LF	Low Frequency
RF	Radio Frequency
RFID	Radio Frequency IDentification
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
UHF	Ultra High Frequency
VB	Visual Basic
VHF	Very High Frequency
XML	eXtensible Markup Language

1. INTRODUCCIÓN

En la actualidad, las personas visitan un supermercado esperando disponer de una gran variedad de productos, un buen ambiente y una atención rápida. Por consiguiente, los supermercados buscan la manera más efectiva de ofrecer al cliente un servicio y atención adecuados a sus necesidades. Particularmente, para aquellos clientes que desean realizar compras pequeñas de 10 artículos o menos, los establecimientos han puesto a su disposición una o dos cajas rápidas exclusivas. Sin embargo, estas cajas tienen gran afluencia de clientes por lo cual se forman largas filas en las mismas, lo que ha producido que el cliente prefiera realizar sus compras cotidianas fuera del establecimiento.

En Ecuador, el modelo convencional de cobro en los supermercados se realiza mediante la lectura de productos con códigos de barra, acompañado de un pago en efectivo o tarjeta. Por su parte, la naturaleza óptica de los códigos de barra lleva consigo un obstáculo inherente ya que las etiquetas requieren ser vistas por el láser del lector. Normalmente se requiere un escaneo a corta distancia, realizado manualmente por el personal de caja. Por otro lado, la tecnología RFID (*Radio Frequency Identification*)[1] presenta características útiles para la lectura de los elementos. Una de las ofertas más atractivas de RFID es su atributo fundamental de no requerir línea de vista al momento de leer las etiquetas. En consecuencia, los artículos no necesitan una orientación particular para su escaneo.

Por lo mencionado, se plantea una alternativa al modelo tradicional de cobro que puede utilizarse en un supermercado como un sistema complementario que brinde al cliente otros métodos de pago. Asimismo, esta alternativa incluye el desarrollo de un prototipo de sistema de pago automático de compras a través de una aplicación móvil Android para una caja rápida con tecnología RFID. Además, el prototipo permite experimentar los beneficios de los sistemas RFID para la lectura de productos y su aplicación en los sistemas de cobro. Finalmente, el prototipo incluye el desarrollo de una aplicación web ASP.NET para la administración del supermercado.

1.1 Objetivos

El objetivo general de este proyecto de titulación es desarrollar un sistema de pago automático de compras mediante una aplicación móvil para una caja rápida con tecnología RFID.

[1] ***Radio Frequency Identification*** **(RFID):** Tecnología inalámbrica que permite la identificación de objetos mediante etiquetas.

En esta línea, los objetivos específicos de este proyecto de titulación son:

- Analizar la fundamentación teórica necesaria para el desarrollo del proyecto, i.e. la plataforma Android, el *framework* de .NET, las bases de datos, los sistemas de identificación por radiofrecuencia y las metodologías de desarrollo ágiles.
- Diseñar los componentes que forman parte del sistema de pago automático de compras
- Implementar los componentes del sistema de pago automático de compras
- Analizar los resultados de las pruebas ejecutadas en el sistema de pago automático de compras.

1.2 Alcance

Para el prototipo de sistema de pago automático de compras, se utilizarán los elementos que se detallan a continuación: un *access point* inalámbrico, etiquetas RFID, un computador con un lector de etiquetas RFID compatible, un celular con sistema operativo Android, una base de datos con la información de los productos, empleados, clientes, facturas y cajas. El prototipo estará compuesto por cuatro componentes: la base de datos, la caja electrónica, la aplicación Android y la aplicación web. La conexión de dichos elementos se describe en la Figura 1.1.

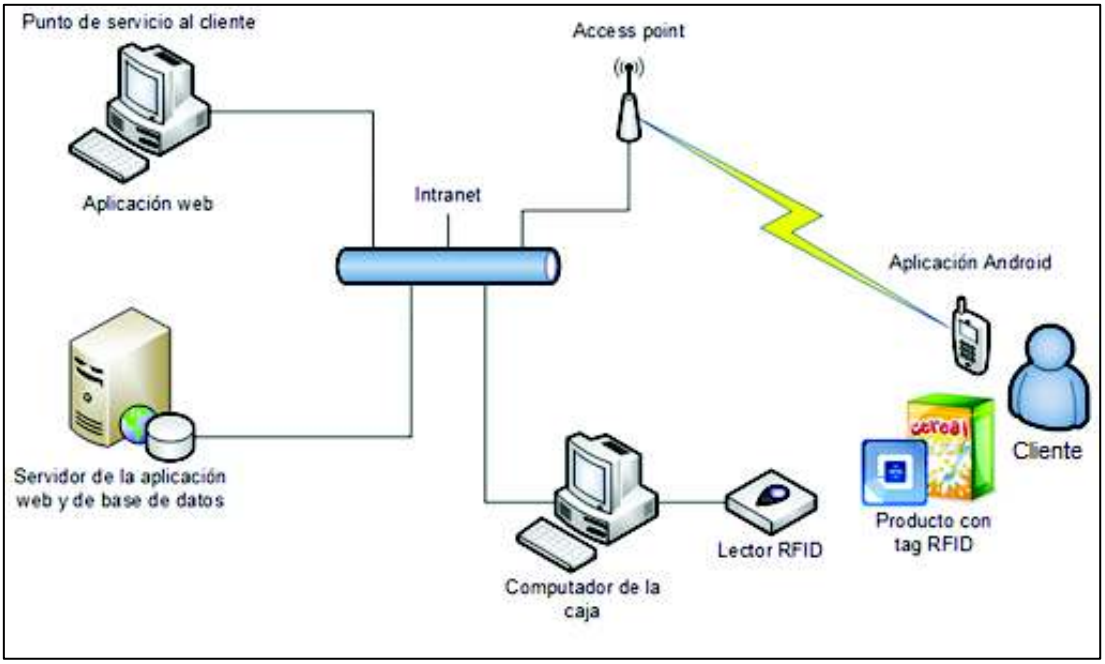


Figura 1.1. Esquema general del prototipo

La caja constará de un computador conectado a un lector RFID, el cual permitirá al cliente la lectura de los productos a ser adquiridos, además, dispondrá del espacio físico suficiente para que el cliente pueda pasar uno a uno sus productos por el lector. Todas las cajas estarán conectadas al *access point* mediante conexión Ethernet, además, cabe mencionar que el prototipo dispondrá de una sola caja.

Todos los productos llevarán adherida su propia etiqueta RFID y se tomarán en cuenta 10 artículos de consumo cotidiano y de primera necesidad debido a que se trata de una caja rápida. Estos productos podrán ser de precio fijo o variable. En el caso de los productos de precio variable como frutas y verduras vendrán previamente empacados en pesos definidos y debidamente etiquetados, (e.g. una libra de uvas). El cliente seleccionará los productos que desee de las perchas y los guardará en una bolsa.

Tanto el servidor de base de datos como las cajas estarán dentro de la intranet del supermercado. La base de datos SQL *Server* almacenará los identificativos únicos de cada producto, la información de los empleados de servicio al cliente, las cuentas de los clientes con sus respectivas credenciales y saldo disponible, el detalle de las compras realizadas por cada cliente y la información de las cajas.

La aplicación web ASP.NET permitirá al personal: la administración de las cuentas de los clientes y los productos del supermercado, así como, la generación de reportes de compras de cada cliente.

Para hacer efectivo el pago, el cliente deberá previamente tener una cuenta registrada en el sistema. La creación de cuentas podrá realizarse en el punto de servicio al cliente dentro del supermercado mediante la aplicación web mencionada. Al crear la cuenta, el cliente hará recargas de dinero para contar con saldo disponible para sus compras. Para la realización del pago se utilizará la aplicación Android que funcionará a manera de una billetera electrónica con el saldo recargado por el cliente.

Una vez que el usuario haya tomado sus productos en una bolsa se acercará al módulo de caja electrónica, donde hará uso de la aplicación móvil Android. En primer lugar, el cliente tendrá que conectarse a la red local del supermercado mediante el *access point*, posteriormente, tendrá que ingresar sus credenciales para la autenticación de su cuenta. Entonces, se visualizará en pantalla las cajas disponibles y se permitirá la elección de una para su utilización. En ese momento, el cliente deberá pasar uno a uno sus productos por el lector RFID de la caja para su lectura y el detalle de los productos será mostrado en la pantalla del teléfono móvil para su confirmación. Finalmente, se procederá a realizar el

pago mediante la aplicación móvil, descontando el valor de la compra del saldo disponible en la cuenta.

El prototipo será probado dentro de un ambiente controlado de pruebas. Cabe mencionar que el sistema no contempla la salida al Internet, de este modo se evitará posibles fallos de seguridad.

1.3 Marco Teórico

En esta sección se presenta un resumen del fundamento teórico necesario para el desarrollo del presente proyecto. En primer lugar, en la Sección 1.3.1 se provee una descripción del sistema operativo Android y sus funcionalidades. Después, en la Sección 1.3.2 se describe el *framework* de .NET y se mencionan los conceptos más relevantes de ASP.NET. Luego, en la Sección 1.3.3 se presenta una descripción de las funcionalidades y herramientas necesarias para el desarrollo de la base de datos. Por otro lado, en la Sección 1.3.4 se exponen los conceptos relacionados a los sistemas de identificación por radiofrecuencia. Por último, en la Sección 1.3.5 se exhiben los fundamentos de la metodología de desarrollo Kanban utilizada para el desarrollo del *software*.

1.3.1 Sistema operativo Android

Android es una plataforma completa de código abierto para dispositivos móviles que integra un sistema operativo, una interfaz de usuario y aplicaciones [1, Sec. 6.3]. Desde su lanzamiento público oficial, Android ha capturado el interés de las empresas, desarrolladores y la audiencia en general. Desde ese momento hasta ahora, esta plataforma de *software* ha mejorado constantemente en términos de características y *hardware* compatible [2].

Recientemente, Android se ha desarrollado más allá de ser una plataforma únicamente de telefonía móvil para proporcionar una plataforma de desarrollo para una gama cada vez más amplia de *hardware*, incluyendo tabletas y televisores; considerándose como un ecosistema compuesto por una combinación de tres componentes [3, Sec. 2]:

- Un sistema operativo gratuito y de código abierto para dispositivos integrados.
- Una plataforma de desarrollo de código abierto para crear aplicaciones.
- Dispositivos, particularmente teléfonos móviles que ejecutan el sistema operativo Android y las aplicaciones creadas para él.

Por último, la fortaleza fundamental de Android es su filosofía abierta que garantiza que se puede corregir cualquier deficiencia en la interfaz de usuario o en el diseño de las aplicaciones nativas[2].

a. Kit de desarrollo de software Android

El SDK[3] de Android incluye todos los elementos necesarios para comenzar a desarrollar, probar y depurar aplicaciones de Android. A continuación, se describen sus elementos [3, Sec. 1]:

- *Las APIs de Android:* el núcleo del SDK son las bibliotecas API[4] de Android que brindan acceso de desarrollador a la pila o *stack* de Android. Estas son las mismas bibliotecas que usa Google para crear aplicaciones nativas de Android.
- *Herramientas de desarrollo:* el SDK incluye varias herramientas de desarrollo que permiten compilar y depurar aplicaciones para convertir el código fuente de Android en aplicaciones ejecutables.
- *El emulador y el administrador de dispositivos virtuales de Android:* la plataforma provee de un emulador de dispositivo móvil totalmente interactivo. El emulador se ejecuta dentro de un dispositivo virtual Android que simula la configuración de *hardware* de un dispositivo. Además, el emulador permite apreciar cómo sería la apariencia y el comportamiento de las aplicaciones en un dispositivo Android real.
- *Documentación completa:* el SDK incluye una amplia información de referencia a nivel de código que detalla exactamente qué se incluye en cada paquete o clase, además de cómo usarlos.
- *Código de muestra:* el SDK de Android incluye una selección de aplicaciones de ejemplo que demuestran algunas de las posibilidades disponibles con Android. Así como, programas simples que destacan cómo usar las funciones de APIs individuales.
- *Soporte en línea:* Android ha generado rápidamente una gran comunidad de desarrolladores.

[2] **Aplicación nativa:** conjunto de aplicaciones preinstaladas en los dispositivos Android.
[3] ***Software Development Kit* (SDK):** conjunto completo de herramientas de desarrollo.
[4] ***Application Programming Interface* (API):** Conjunto de comandos, funciones, métodos y procedimientos que ofrece una librería para interactuar con otro *software*.

a.1. Pila de software de Android

La pila o *stack* de *software* de Android consta de [3, Sec. 1]:

- Un *kernel* de Linux
- Una colección de librerías C/C++
- Un *framework* de aplicación que proporciona servicios
- La administración del *runtime*
- Las aplicaciones

La pila de *software* de Android se compone de los elementos que se muestran en la Figura 1.2.

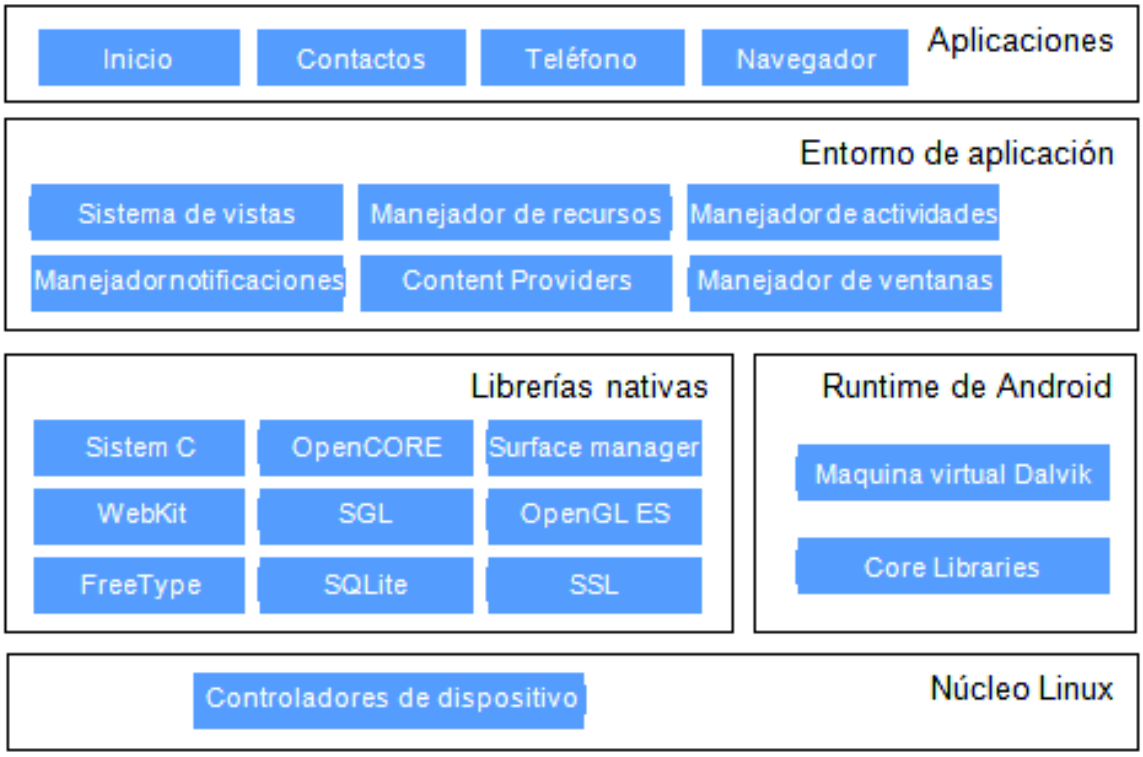


Figura 1.2. Pila de software de Android [4]

a.1.1. Kernel Linux

Los servicios principales (incluidos los controladores del *hardware*, la administración de procesos y memoria, la seguridad, la red y la administración de energía) son manejados por un núcleo Linux. Asimismo, proporciona una capa de abstracción entre el *hardware* y el resto de la pila de *software*.

a.1.2. Librerías

Las librerías se ejecutan sobre el *kernel* Linux. Android incluye varias bibliotecas centrales de C/C++ como: una biblioteca de medios para la reproducción de audio y video, bibliotecas de gráficos, SQLite[5] para soporte de base de datos nativa, SSL[6], Surface Manager[7] y WebKit[8] para la administración de un navegador web integrado y seguridad en Internet.

a.1.3. Android Runtime

El *runtime* de Android incluye las bibliotecas centrales y la máquina virtual Dalvik. Además, el *runtime* es el motor que alimenta las aplicaciones y junto con las bibliotecas centrales forma la base del *framework* de aplicación. Por otra parte, Dalvik es una máquina virtual basada en registros que ha sido optimizada para garantizar que un dispositivo pueda ejecutar múltiples instancias de manera eficiente. Aunque la mayoría del desarrollo de aplicaciones de Android está escrito usando el lenguaje de programación Java[9], Dalvik no es una máquina virtual Java.

Las bibliotecas centrales de Android brindan la mayoría de las funcionalidades disponibles en las bibliotecas centrales de Java, así como en las bibliotecas específicas de Android.

a.1.4. Framework de Aplicación

El *framework* o entorno de aplicación proporciona las clases utilizadas para crear aplicaciones Android. También, proporciona una abstracción genérica para el acceso al *hardware* y administra la interfaz de usuario y los recursos de la aplicación.

a.1.5. Capa de Aplicación

Todas las aplicaciones, tanto nativas como las de terceros, se crean en la capa de aplicación mediante las mismas librerías API. La capa de aplicación se ejecuta dentro del *runtime* de Android, utilizando las clases y servicios disponibles desde el *framework* de aplicación.

---

[5] **SQLite:** Motor de base de datos relacional administrado para almacenar datos de forma segura y eficiente.
[6] ***Secure Sockets Layer* (SSL):** Protocolo de seguridad que establece un canal seguro entre dispositivos conectados al Internet.
[7] ***Surface Manager*:** Maneja los subsistemas de gráficos 2D y 3D.
[8] **WebKit:** Motor de navegador web de código abierto.
[9] **Java:** Lenguaje de programación orientado a objetos de propósito general.

a.2. Componentes de una aplicación Android

Las aplicaciones de Android están conformadas por una serie de componentes acoplados. Todos estos están vinculados por el manifiesto de la aplicación, que describe cada componente y cómo interactúan. Además, el manifiesto se utiliza para especificar los metadatos de la aplicación, los requisitos de plataforma y *hardware*, las bibliotecas externas y los permisos necesarios. Los siguientes componentes comprenden los bloques de construcción para todas las aplicaciones de Android [3, Sec. 3]:

a.2.1. Actividades

Las actividades están representadas en cada una de las interfaces de usuario de la capa de aplicación. Así mismo, usan fragmentos y vistas para el diseño y la visualización de información, y para responder a las acciones del usuario.

a.2.2. Intents

Un *intent* provee un mecanismo de transferencia de datos entre aplicaciones. Se puede utilizar para: iniciar o detener actividades y servicios, transmitir mensajes o solicitar que se realice una acción en un dato en particular.

a.2.3. Servicios

Un servicio podría considerarse como un trabajador invisible de la aplicación que se ejecuta sin una interfaz de usuario, actualizando sus fuentes de datos y actividades, disparando notificaciones y transmitiendo *intents.* Además, se utilizan para realizar tareas de ejecución prolongada o aquellas que no requieren la interacción del usuario.

a.2.4. Proveedores de contenido

Los proveedores de contenido administran los datos de la aplicación y generalmente interactúan con las bases de datos SQL. También, son los medios preferidos para compartir datos a través de las aplicaciones. Por consiguiente, los dispositivos Android incluyen varios proveedores de contenido nativos que exponen bases de datos útiles, como la tienda multimedia y los contactos.

a.2.5. Receptores de difusión

Los receptores de difusión permiten que la aplicación escuche *intents* que coincidan con los criterios especificados por el desarrollador. Además, los receptores de difusión inician la aplicación para reaccionar ante cualquier *intent* recibido, lo que los hace perfectos para crear aplicaciones impulsadas por eventos.

a.2.6. Widgets

Un *widget* es un componente de la aplicación visual que normalmente se agrega a la pantalla de inicio del dispositivo y se puede definir como una variación especial de un receptor de difusión. Asimismo, permiten crear componentes de aplicaciones dinámicos e interactivos para que los usuarios los inserten en sus pantallas de inicio.

a.2.7. Notificaciones

Las notificaciones permiten alertar a los usuarios sobre eventos de aplicaciones sin robar el foco o interrumpir la actividad actual. Así mismo, son la técnica preferida para llamar la atención de un usuario cuando la aplicación no está visible o activa, particularmente desde un servicio o un receptor de difusión. Por ejemplo, cuando un dispositivo recibe un mensaje de texto o un correo electrónico, las aplicaciones de mensajería usan notificaciones para alertarlo mediante luces intermitentes, reproducción de sonidos, visualización de íconos o desplazamiento de un resumen de texto.

a.2.8. Archivo Manifiesto

Cada proyecto de Android incluye un archivo de manifiesto almacenado en la raíz de la jerarquía de proyectos. El manifiesto define la estructura y los metadatos de su aplicación, sus componentes y sus requisitos.

Incluye nodos para cada una de las actividades, servicios, proveedores de contenido y receptores de difusión que conforman la aplicación. Asimismo, utiliza filtros de intención y permisos para determinar cómo interactúan dichos componentes entre sí y con otras aplicaciones. Adicionalmente, el archivo manifiesto permite especificar metadatos de la aplicación (como el ícono, número de versión) y nodos adicionales de nivel superior que pueden especificar los permisos necesarios, y definir los requisitos de *hardware*, pantalla o plataforma.

a.3. Arquitectura de una aplicación Android

La arquitectura de Android fomenta la reutilización de componentes de aplicación, por lo tanto, permite publicar y compartir actividades, servicios y datos con otras aplicaciones. Conviene señalar que, el acceso a los mismos está definido por las restricciones de seguridad que el desarrollador defina. Finalmente, los siguientes servicios de aplicaciones son las piedras angulares de todas las aplicaciones Android [3, Sec. 3]:

- *Administrador de actividades y administrador de fragmentos.* El administrador de actividades y el administrador de fragmentos controlan el ciclo de vida de las actividades y fragmentos respectivamente. Asimismo, el *stack* de actividades.

- *Vistas.* Las vistas son utilizadas para construir las interfaces de usuario para las actividades.

- *Administrador de notificaciones.* El administrador de notificaciones proporciona un mecanismo consistente y no intrusivo para brindar señalización a los usuarios.

- *Proveedor de contenido.* El proveedor de contenido permite a la aplicación compartir datos.

- *Administrador de recursos.* El administrador de recursos permite externalizar recursos sin código, por ejemplo, cadenas y gráficos.

- *Intents.* Un *intent* proporciona un mecanismo para transferir datos entre las aplicaciones y sus componentes.

a.4. Ciclo de vida de una aplicación Android

a.4.1. Estados de una actividad

A medida que las actividades se crean y se destruyen, éstas realizan transiciones a través de cuatro estados posibles dentro de la pila o *stack* de actividades (Figura 1.3). A continuación, se describen los estados de una actividad [3, Sec. 3]:

- *Activa.* Una actividad está activa cuando está en la parte superior de la pila. Entonces, la actividad es visible, está enfocada y en primer plano para recibir la entrada del usuario. Asimismo, Android intentará mantener con vida a esta actividad a toda costa, terminando a las actividades que se encuentran más abajo en la pila, con el objetivo de garantizar la reserva de recursos para la misma. Cuando otra actividad se activa, ésta se pausará.

- *Pausada.* En algunos casos, la actividad será visible pero no tendrá foco; en este punto está en pausa. Este estado se alcanza si una actividad de pantalla transparente o de pantalla incompleta está activa al frente. Además, cuando se pausa una actividad se la trata como si estuviera activa; sin embargo, no recibe eventos de entrada del usuario. Por último, en casos extremos, Android terminará una actividad pausada para recuperar recursos para la actividad activa.

- *Detenida.* Cuando una actividad no está visible, se "detiene". Asimismo, esta actividad permanecerá en la memoria, conservando toda la información de estado; sin embargo, la actividad se convierte en un candidato para la terminación cuando el sistema requiere memoria en otro lugar. Además, cuando una actividad está detenida, es importante guardar los datos y el estado actual de la interfaz de usuario, y detener las operaciones no críticas.

- *Inactiva.* El estado inactivo está presente después de que una actividad ha sido eliminada y antes de su lanzamiento. Asimismo, las actividades inactivas se eliminan de la pila de actividades y deben reiniciarse antes de que puedan mostrarse y utilizarse.

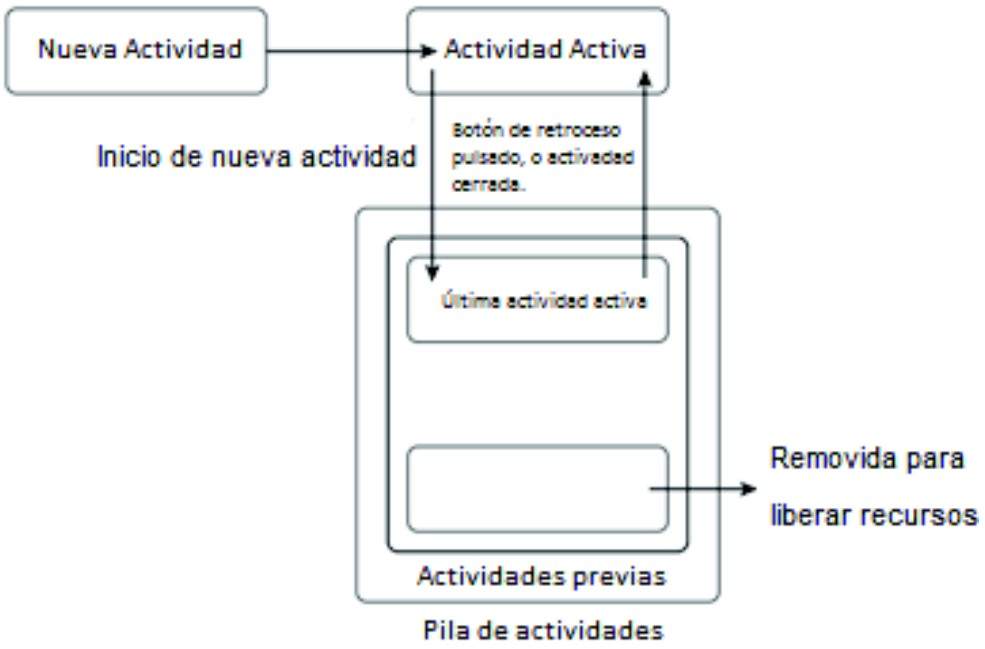


Figura 1.3. Pila de actividades [3]

A diferencia de muchas plataformas de aplicaciones tradicionales, las aplicaciones de Android tienen un control limitado sobre sus propios ciclos de vida. En su lugar, los componentes de la aplicación deben escuchar los cambios en el estado de la aplicación y reaccionar en consecuencia, teniendo cuidado especial en estar preparados para una finalización prematura.

De forma predeterminada, cada aplicación de Android se ejecuta en su propio proceso en una instancia separada de Dalvik. Además, la gestión de memoria y proceso se maneja exclusivamente por el *runtime*.

Adicionalmente, Android maneja agresivamente sus recursos, haciendo lo que sea necesario para garantizar una experiencia de usuario estable y sin inconvenientes. Entonces, en la práctica, esto significa que los procesos y sus aplicaciones alojadas serán eliminados (en algunos casos sin previo aviso), para liberar recursos para las aplicaciones de mayor prioridad.

Sin embargo, el orden en que se terminan los procesos para reclamar recursos está determinado por la prioridad de sus aplicaciones alojadas. La prioridad de una aplicación es igual a la de su componente de mayor prioridad. En consecuencia, si dos aplicaciones tienen la misma prioridad, el proceso que ha estado en esa prioridad por más tiempo será eliminado primero.

La Figura 1.4 muestra el árbol de prioridad utilizado para determinar el orden de finalización de la aplicación.

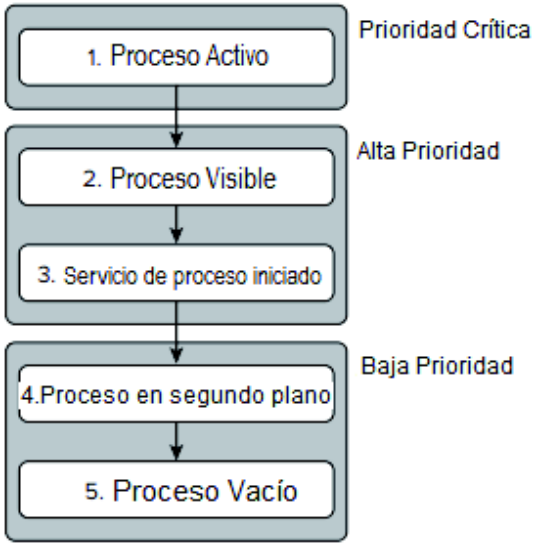


Figura 1.4. Estados de una aplicación Android [3]

a.4.2. Procesos activos

Los procesos activos (en primer plano) tienen componentes de aplicación con los que el usuario está interactuando. Por consiguiente, éstos son procesos de alta prioridad, por lo que Android procurará responder a ellos reclamando recursos de otras aplicaciones. En general, hay muy pocos de estos procesos y se eliminan solo como último recurso.

a.4.3. Procesos visibles

Los procesos visibles pero inactivos son aquellos que albergan actividades "visibles". Como su nombre lo sugiere, las actividades visibles son observables, pero no están en

primer plano ni responden a los eventos del usuario. Así que, esto ocurre cuando una actividad está parcialmente oscurecida, por ejemplo, por una actividad de pantalla transparente. En general, hay muy pocos procesos visibles, y se terminan sólo en circunstancias extremas para permitir que los procesos activos continúen.

a.4.4. Procesos de servicio iniciado

Son procesos de servicios de alojamiento que se han iniciado en la aplicación. Debido a que estos servicios no interactúan directamente con el usuario, reciben una prioridad ligeramente menor que las actividades visibles o los servicios en primer plano. Asimismo, las aplicaciones con servicios en ejecución se consideran procesos en primer plano y no se eliminan a menos que se necesiten recursos para procesos activos o visibles. Finalmente, cuando el sistema termina un servicio en ejecución, intentará reiniciarlo cuando los recursos estén disponibles.

a.4.5. Procesos en segundo plano

Son procesos que alojan actividades que no son visibles y que no tienen ningún servicio en ejecución. En general, habrá una gran cantidad de procesos en segundo plano que Android terminará para obtener recursos para los procesos en primer plano.

a.4.6. Procesos vacíos

Los procesos vacíos tienen como objetivo mejorar el rendimiento general del sistema. Android a menudo conservará una aplicación en la memoria una vez que haya llegado al final de su vida útil. Además, Android mantiene este caché para mejorar el tiempo de inicio de las aplicaciones cuando se vuelven a lanzar. Por consiguiente, estos procesos se eliminan rutinariamente, según sea necesario.

1.3.2 .NET Framework

La plataforma .NET es un *framework* de desarrollo con una interfaz de programación para servicios de Windows y APIs. El *framework* de .NET es realmente un conjunto de algunas tecnologías [5, Sec. 1.1]:

- *Los lenguajes .NET:* éstos incluyen Visual Basic, C#, F# y C++, aunque los desarrolladores de terceros han creado cientos más.
- *Common Language Runtime (CLR):* es el motor que ejecuta todos los programas .NET y proporciona servicios automáticos para estas aplicaciones. Por ejemplo: comprobación de seguridad, gestión de memoria y optimización.

- *La biblioteca de clases de .NET Framework:* la biblioteca de clases recopila miles de piezas de funcionalidad precompiladas que pueden ser útiles a las aplicaciones del desarrollador. Estas características se organizan en conjuntos de tecnología, como ADO.NET (la tecnología para crear aplicaciones de bases de datos) y Windows Presentation Foundation (WPF, la tecnología para crear interfaces de usuario de escritorio).

- *ASP.NET:* éste es el motor que aloja las aplicaciones web creadas con .NET, y es compatible con las funciones de la biblioteca de clases de .NET Framework. Además, incluye un conjunto de servicios específicos de la web. Por ejemplo: autenticación segura y almacenamiento de datos.

- *Visual Studio:* esta herramienta de desarrollo opcional contiene un amplio conjunto de funciones de depuración y productividad. Asimismo, incluye el .NET *Framework* completo.

La plataforma .NET proporciona una base sólida para desarrollar aplicaciones, tales como:

- *Aplicaciones basadas en formularios Windows:* aplicaciones de cliente tradicionales.

- *Aplicaciones de consola:* aplicaciones tradicionales tipo DOS[10], como las secuencias de comandos por lotes.

- *Aplicaciones web ASP.NET:* aplicaciones basadas en navegadores dinámicos.

- *Bibliotecas de componentes:* .NET recopila miles de piezas de funcionalidad precompiladas y las pone a disposición del desarrollador para sus aplicaciones.

- *Controles personalizados de Windows:* .NET permite desarrollar sus propios controles de Windows.

- *Controles personalizados web:* el concepto de controles personalizados se puede extender a las aplicaciones web permitiendo la reutilización y la modularización del código.

[10] ***Disk Operating Systems*** **(DOS):** sistema operativo que se ejecuta desde una unidad de disco duro.

- *Servicios web:* son funcionalidades disponibles a través de estándares de la industria como HTTP[11], XML[12] y SOAP[13].

a. ASP .NET

ASP.NET simplifica el desarrollo de páginas web con programación basada en formularios. Estos se denominan formularios web y son análogos a los formularios VB (Visual Basic)[14], donde el desarrollador simplemente arrastra los controles deseados al formulario y configura las funciones de los mismos. Además, la programación de formularios web se basa en eventos y proporciona a los desarrolladores controles de servidor. Por lo tanto, los formularios web de ASP.NET permiten la separación de la lógica de aplicación y la capa de presentación. En resumen, el modelo de programación ASP.NET presenta los siguientes beneficios [6, Sec. 2]:

- Un gran conjunto de controles de servidor que hacen que HTML sea adecuado para cualquier cliente.
- Gestión del estado de la sesión.
- Un modelo de programación basado en eventos en el lado del servidor, simple e intuitivo.
- Lógica de aplicación que se puede escribir en cualquier lenguaje Microsoft .NET (VB, C #, C ++, etc); el código del lado del servidor de las aplicaciones se compila para un mejor rendimiento.
- Visual Studio.NET como herramienta RAD[15] (*Rapid Application Development*), que simplifica el proceso de desarrollo de formularios web.

ASP.NET está diseñado ante todo como una plataforma de programación del lado del servidor. En consecuencia, todo el código ASP.NET se ejecuta en el servidor web. Cuando el código ASP.NET termina de ejecutarse, el servidor web envía al usuario el resultado final, una página HTML común que se puede ver en cualquier navegador.

---

[11] ***Hypertext Transfer Protocol*** **(HTTP):** protocolo usado para la transferencia de datos por la web.
[12] ***Extensible Markup Language*** **(XML):** se usa para definir documentos con un formato estándar que pueda ser leído por cualquier aplicación compatible.
[13] ***Simple Object Access Protocol*** **(SOAP):** es un método para transferir mensajes o pequeñas cantidades de información a través de Internet.
[14] **Visual Basic (VB):** es un lenguaje de programación y entorno de desarrollo creado por Microsoft.
[15] **RAD:** conjunto de técnicas de metodología de desarrollo de *software* utilizadas para agilizar el desarrollo de aplicaciones de *software*.

La programación del lado del servidor no es la única forma de crear una página web interactiva. Otra opción es la programación del lado del cliente, que le solicita al navegador que descargue el código y lo ejecute localmente en la computadora del cliente. Así como, hay una variedad de plataformas de programación del lado del servidor, también hay varias maneras de realizar la programación del lado del cliente, desde fragmentos de código JavaScript que pueden integrarse directamente dentro del código HTML de una página web, hasta complementos como Adobe Flash[16] y Microsoft Silverlight[17]. Por último, la Figura 1.5 muestra la diferencia entre los modelos del lado del servidor y del lado del cliente.

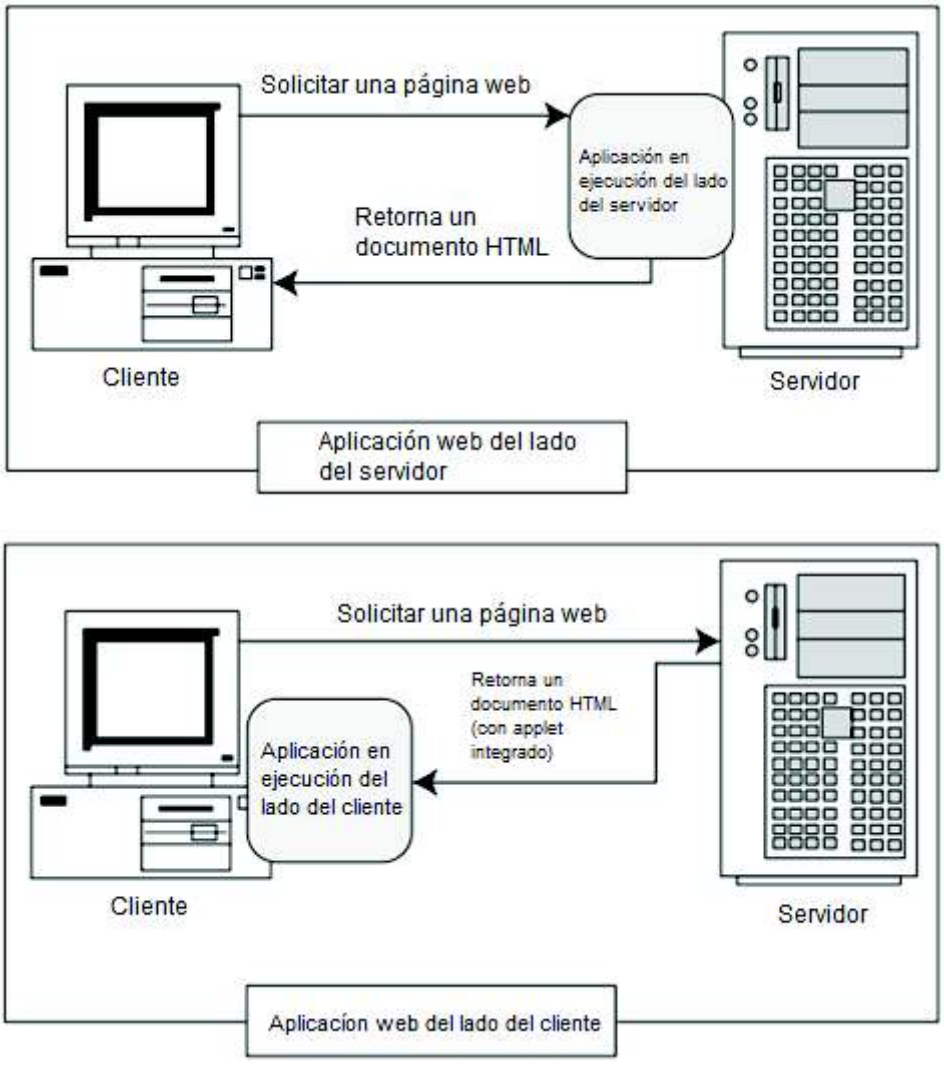


Figura 1.5. Aplicación web del lado del servidor y del lado del cliente **[6]**

[16] **Adobe Flash:** es una plataforma de desarrollo de aplicaciones desarrollada por Adobe Systems. El enfoque principal de la plataforma Flash es la creación de aplicaciones de Internet.
[17] **Microsoft Silverlight:** es un *framework* para el diseño de aplicaciones web enriquecidas con gráficos y video.

Cabe mencionar que ASP.NET utiliza programación del lado del servidor para evitar varios problemas, entre ellos:

- *Aislamiento:* el código del lado del cliente no puede acceder a los recursos del lado del servidor. Por ejemplo, una aplicación cliente no tiene una forma sencilla de leer un archivo o interactuar con una base de datos en el servidor (presentará problemas de seguridad y compatibilidad con el navegador).

- *Seguridad:* los usuarios finales pueden ver el código del lado del cliente, entonces, una vez que un usuario malicioso entiende cómo funciona una aplicación puede manipularla.

- *Clientes ligeros*: hoy en día, tabletas y los teléfonos inteligentes son ampliamente utilizados. Estos dispositivos suelen tener algún tipo de capacidad de navegación web incorporada, pero es posible que no sean compatibles con plataformas de programación del lado del cliente como Adobe Flash o Microsoft Silverlight.

En los últimos años, hubo un renacimiento en la programación del lado del cliente, especialmente con JavaScript[18] [6, Sec. 1]. Por su parte, los desarrolladores crean aplicaciones que se comunican con un servidor web para buscar información y realizar tareas que no serían posibles si las aplicaciones estuvieran limitadas a la computadora local. Afortunadamente, ASP.NET aprovecha este cambio de dos maneras:

- *JavaScript frills:* en algunos casos, ASP.NET permite combinar lo mejor de la programación del lado del cliente y del servidor. Por ejemplo, determinados controles ASP.NET pueden detectar "inteligentemente" las características del navegador del cliente. Si el navegador admite JavaScript, estos controles devolverán una página web que incorpora JavaScript para una interfaz de usuario más interactiva y llamativa al usuario.

- *Características Ajax de ASP.NET:* Ajax es un conjunto de técnicas de JavaScript utilizadas para crear páginas rápidas y receptivas con contenido dinámico.

Sin embargo, es importante entender un hecho fundamental, no importa cuáles sean las capacidades del navegador, el código C # que se escribe siempre se ejecuta en el servidor.

[18] **JavaScript:** es un lenguaje de *scripting*, usado principalmente en la Web. Se utiliza para mejorar las páginas HTML y se encuentra comúnmente incorporado en el código HTML.

1.3.3 Base de datos

Una base de datos es una colección de datos organizados en un formato estructurado definido mediante metadatos. Estos metadatos pueden ser definidos como datos sobre los datos que se almacenan. Además, determinan cómo se almacena la información dentro de la base de datos.

En la actualidad, se han implementado varios modelos de bases de datos para el almacenamiento y la administración de información. Algunos de los modelos más comunes son [7, Sec. 1]:

- *Jerárquico:* este modelo tiene una estructura jerárquica padre-hijo similar a un árbol invertido, sus datos están organizados en nodos que son los equivalentes lógicos de las tablas en una base de datos relacional. Un nodo padre puede tener muchos nodos secundarios, pero un nodo secundario sólo puede tener un nodo principal. A pesar de que, este modelo ha sido ampliamente utilizado, a menudo se considera inadecuado para muchas aplicaciones debido a su estructura inflexible y la falta de soporte para relaciones complejas.

- *Red:* este modelo aborda algunas de las limitaciones del modelo jerárquico. Los datos están organizados en tipos de registro, el equivalente lógico de las tablas en una base de datos relacional. Al igual que el modelo jerárquico, el modelo de red utiliza una estructura de árbol invertido, pero los tipos de registro se organizan en una estructura de conjunto que relaciona los pares de tipos de registro en propietarios y miembros. Al acceder a la base de datos, los usuarios deben estar muy familiarizados con la estructura y mantener un seguimiento cuidadoso de dónde están y cómo llegaron allí. Asimismo, es difícil cambiar la estructura sin afectar las aplicaciones que interactúan con la base de datos.

- *Relacional*: este modelo aborda muchas de las limitaciones tanto de los modelos jerárquicos como de red. En una base de datos jerárquica o de red, la aplicación se basa en una implementación definida de esa base de datos, que luego está codificada en la aplicación. Si se agrega un nuevo atributo (elemento de datos) a la base de datos, se debe modificar la aplicación, incluso si no se utiliza el atributo. Sin embargo, una base de datos relacional es independiente de la aplicación. Entonces, puede realizar modificaciones no destructivas en la estructura sin afectar la aplicación. Además, la estructura de la base de datos relacional se basa en tablas, junto con la capacidad de definir relaciones complejas entre las mismas. Por lo tanto, se puede acceder directamente a cada tabla, sin las limitaciones de un

modelo jerárquico o de red que requiere la navegación de una estructura de datos compleja.

a. Modelo Relacional

El modelo relacional se basa principalmente en los principios matemáticos de la teoría de conjuntos y la lógica de predicados, admite la recuperación fácil de datos, impone la integridad de los datos (precisión y coherencia de la información) y proporciona una estructura de base de datos independiente de las aplicaciones que acceden a los datos almacenados.

En el núcleo del modelo relacional está la relación, definida como un conjunto de columnas y filas recopiladas en una estructura similar a una tabla que representa una sola entidad compuesta de datos relacionados. Por su parte, una entidad es una persona, lugar, cosa, evento o concepto acerca de los datos recopilados como un artista, un libro o una transacción de venta. Cada relación comprende uno o más atributos o columnas, que son hechos unitarios que describen o caracterizan una entidad de alguna manera. Por ejemplo, en la Figura 1.6 la entidad es un disco compacto (CD) con los atributos *CD_NAME* (el título del CD), *ARTIST_NAME* (el nombre del artista), y *COPYRIGHT_YEAR* (el año en que la grabación fue registrada).

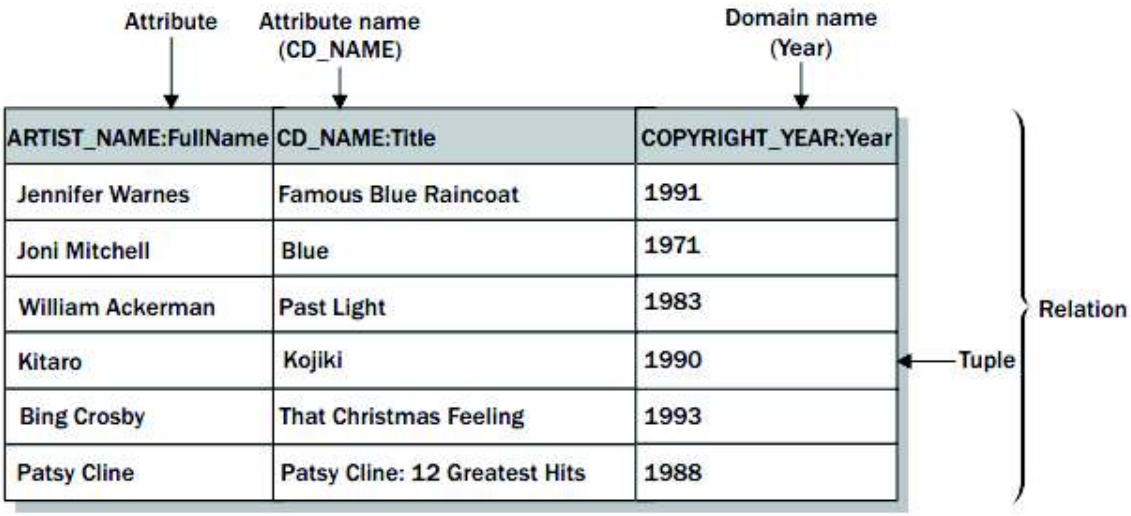


Figura 1.6. Relación con atributos [7]

En una relación los datos se almacenan en tuplas (filas), definidas como un conjunto de datos cuyos valores constituyen una instancia de cada atributo definido para esa relación. Además, cada tupla representa un registro de datos relacionados. En ocasiones, al conjunto de datos se lo conoce como registro. Por ejemplo, en la Figura 1.6 la primera tupla contiene el valor “Jennifer Warnes” para el atributo *ARTIST_NAME*, el valor “Famous Blue

Raincoat" para el atributo *CD_NAME* y el valor "1991" para el atributo *COPYRIGHT_YEAR.* Juntos los tres valores forman una tupla.

b. SQL

Las bases de datos relacionales se han convertido en el mecanismo de almacenamiento de datos más común para las aplicaciones informáticas modernas. Muchas de estas fuentes de datos son administradas por un sistema de administración de bases de datos relacionales (RDBMS: *Relational Database Management System*), como Oracle, Microsoft SQL Server, MySQL. Estos sistemas de administración se basan en el lenguaje de consulta estructurado (SQL: *Structured Query Language*) para crear y modificar objetos de la base de datos [7].

SQL aún carece de muchas de las capacidades básicas de programación de la mayoría de los otros lenguajes de programación. A menudo se hace referencia a SQL como un sublenguaje de datos porque se usa con mayor frecuencia en asociación con lenguajes de programación de aplicaciones como C y Java que no están diseñados para manipular datos almacenados en una base de datos. Como resultado, SQL se usa junto con el lenguaje de la aplicación para proporcionar un medio eficiente para acceder a esos datos.

b.1. Transacciones

Una transacción es una secuencia de una o más declaraciones SQL que juntas forman una unidad lógica de trabajo. Por otro lado, las declaraciones SQL que forman la transacción suelen estar estrechamente relacionadas y realizan acciones interdependientes. Asimismo, cada declaración en la transacción realiza una parte de una tarea, pero se requiere que todos completen la tarea. Por otro lado, agrupar las declaraciones como una sola transacción le dice al sistema de administración de base de datos que la secuencia de instrucciones completa debe ejecutarse atómicamente. Por lo tanto, todas las declaraciones deben completarse para que la base de datos esté en un estado consistente [8, Sec. 1].

c. Procedimientos almacenados

Los procedimientos almacenados proporcionan la capacidad de realizar el procesamiento de aplicaciones relacionadas con la base de datos dentro de la misma base de datos. Por ejemplo, un procedimiento almacenado podría implementar la lógica de la aplicación para aceptar un pedido de un cliente o transferir dinero de una cuenta bancaria a otra.

SQL utiliza una sentencia de creación *CREATE PROCEDURE* para definir inicialmente un procedimiento almacenado y una sentencia *DROP PROCEDURE* para descartar los

procedimientos que ya no son necesarios. La sentencia de creación se define de la siguiente manera [8, Sec. 20]:

- El nombre del procedimiento almacenado
- El número y los tipos de datos de sus parámetros
- Los nombres y tipos de datos de cualquier variable local utilizada por el procedimiento
- La secuencia de instrucciones ejecutadas cuando se llama al procedimiento.

c.1. Ventajas de los procedimientos almacenados

Los procedimientos almacenados ofrecen varias ventajas, tanto para los usuarios como para los administradores de una base de datos, por ejemplo:

- *Rendimiento en tiempo de ejecución:* ejecutar un procedimiento almacenado precompilado puede ser mucho más rápido que ejecutar las sentencias SQL equivalentes a través de un proceso PREPARE / EXECUTE.
- *Reusabilidad:* una vez que se ha definido un procedimiento almacenado para una función específica, se puede llamar a ese procedimiento desde diferentes programas de aplicación que necesitan realizar la función, permitiendo una reutilización muy fácil de la lógica de la aplicación y reduciendo el riesgo de error del programador de la aplicación.
- *Red reducida de tráfico:* en una configuración cliente/servidor, enviar una llamada a un procedimiento almacenado a través de la red y recibir los resultados en un mensaje de respuesta genera mucho menos tráfico de red que el envío bidireccional de cada declaración SQL individual. De manera que, se puede mejorar considerablemente el rendimiento general del sistema en una red con mucho tráfico o con conexiones de menor velocidad.
- *Seguridad:* el procedimiento almacenado se trata como una entidad confiable dentro de la base de datos y se ejecuta con sus propios privilegios. El usuario que ejecuta el procedimiento almacenado necesita tener sólo permiso para ejecutarlo, no permisos en las tablas subyacentes a las que el procedimiento almacenado puede acceder o modificar. Por lo tanto, el procedimiento almacenado permite al administrador de la base de datos mantener una seguridad más estricta en los datos subyacentes, al mismo tiempo que proporciona a los usuarios individuales la

actualización de datos específica o las capacidades de acceso a datos que requieren.

- *Encapsulación:* los procedimientos almacenados son una forma de lograr uno de los objetivos centrales de la programación orientada a objetos, la encapsulación de valores de datos, estructuras y acceso dentro de un conjunto de interfaces externas muy limitadas y bien definidas. En la terminología de objetos, los procedimientos almacenados pueden ser los métodos a través de los cuales los objetos en el RDBMS subyacente son manipulados exclusivamente.

- *Simplicidad de acceso:* en una base de datos de un gran establecimiento, una colección de procedimientos almacenados puede ser la forma principal en que los programas acceden a la base de datos. Los procedimientos almacenados forman un conjunto bien definido de transacciones y consultas que las aplicaciones pueden realizar en la base de datos. Por ejemplo, una llamada a una función simple y predefinida que verifica el saldo de una cuenta, dado un número de cliente, es más fácil de entender que las correspondiente declaraciones SQL.

- *Aplicación de reglas de negocio:* las capacidades de procesamiento condicional de los procedimientos almacenados a menudo se utilizan para colocar reglas comerciales en la base de datos. Por ejemplo, un procedimiento almacenado utilizado para agregar un pedido a la base de datos puede contener la lógica para verificar el saldo del cliente que hace el pedido y verificar si hay suficiente inventario disponible para completar el pedido. Además, se puede rechazar el pedido si estas condiciones no pueden ser cumplidas.

d. Triggers

Un *trigger* o disparador es un conjunto particular de código de procedimiento almacenado cuya activación es causada por modificaciones en el contenido de la base de datos.

A diferencia de los procedimientos almacenados creados con una instrucción *CREATE PROCEDURE*, un disparador no se activa mediante una instrucción *CALL* o *EXECUTE,* sino que está asociado con una tabla de la base de datos.

Por lo tanto, cuando se modifican los datos de la tabla (mediante una instrucción *INSERT*, *DELETE* o *UPDATE*), se desencadena el disparador, lo que significa que el RDBMS ejecuta las instrucciones SQL que componen el cuerpo del disparador [8, Sec. 20].

d.1. Ventajas y desventajas de los triggers

Los *triggers* pueden ser extremadamente útiles como parte integral de una definición de base de datos, y se pueden usar en una variedad de funciones diferentes, que incluyen las siguientes:

- *Cambios de auditoría.* Un *trigger* puede detectar y rechazar actualizaciones específicas y cambios que no deberían permitirse en la base de datos.
- *Operaciones en cascada.* Un *trigger* puede detectar una operación dentro de la base de datos y tomar las acciones pertinentes automáticamente. Por ejemplo, al eliminar el registro de un cliente, se puede realizar el ajuste de los saldos de las cuentas de manera inmediata.
- *Invocación de procedimiento almacenado.* Un *trigger* puede llamar a uno o más procedimientos almacenados o incluso invocar acciones fuera del DBMS a través de llamadas a procedimientos externos en respuesta a las actualizaciones de la base de datos.

En cada uno de estos casos, un *trigger* incorpora un conjunto de reglas que rigen la información en la base de datos y las modificaciones respectivas. Por su parte, dichas reglas están integradas en un solo lugar en la base de datos (i.e. la definición del *trigger*). Como resultado, las reglas se aplican de manera uniforme en todas las aplicaciones que acceden a la base de datos. En consecuencia, cuando las reglas necesitan modificarse, se pueden cambiar una vez con la garantía de que el cambio se aplicará de manera uniforme.

La principal desventaja de los *triggers* es su posible impacto en el rendimiento, si se establece un *trigger* en una tabla particular, entonces cada operación en la base de datos que intente actualizar esa tabla hace que el RDBMS ejecute el procedimiento desencadenante. De modo que, para una base de datos que requiere altas tasas de inserción o actualización de datos, la sobrecarga de este procesamiento puede ser considerable.

1.3.4 Sistemas de identificación por radiofrecuencia RFID

La identificación por radiofrecuencia (RFID) es una tecnología inalámbrica que utiliza los campos electromagnéticos de radiofrecuencia para transferir datos con el objetivo de identificar y seguir automáticamente las etiquetas adjuntas a los activos o productos.

Según [9, Sec. 1], la identificación por radiofrecuencia tiene sus raíces en los descubrimientos hechos por Faraday a mediados del siglo XIX y los descubrimientos

realizados entre 1900 y 1940 en las tecnologías de radio y radar. Faraday descubrió el concepto de inducción mutua, que constituye la base para alimentar las etiquetas pasivas que operan en el campo cercano. Además, los avances tecnológicos que permitieron el desarrollo de las etiquetas de campo lejano se produjeron durante la primera mitad del siglo XX.

Hoy en día, la comprensión dentro de la comunidad empresarial de los beneficios de las metodologías eficientes de manejo de datos, así como de los avances en las técnicas de fabricación han permitido el crecimiento acelerado de los sistemas de identificación por radiofrecuencia. Las aplicaciones habilitadas con RFID han crecido a un ritmo vertiginoso con implementaciones de sistemas en varias industrias, tales como, productos farmacéuticos, cuidado de la salud, transporte, venta minorista, defensa y logística.

Las principales áreas que impulsaron el despliegue comercial de la tecnología RFID son la logística, la gestión de la cadena de suministro, el seguimiento de artículos, los implantes médicos, los peajes, el control de acceso a edificios, la seguridad de la aviación y aplicaciones de seguridad nacional. Estos sistemas se utilizan para una amplia gama de aplicaciones que rastrean, supervisan, informan y administran elementos mientras se mueven entre diferentes ubicaciones físicas.

RFID presenta riesgos de seguridad y privacidad (e.g. la lectura o seguimiento ilícito de etiquetas) que deben ser cuidadosamente mitigados a través de controles administrativos, operacionales y técnicos con el fin de aprovechar los numerosos beneficios que la tecnología tiene para ofrecer. Cada sistema RFID tiene diferentes componentes y personalizaciones para que pueda admitir un proceso comercial particular para una organización; como resultado, los riesgos de seguridad para los sistemas RFID y los controles disponibles para abordarlos son muy variados.

a. Arquitectura de los sistemas RFID

Como se muestra en la Figura 1.7, un sistema RFID posee los siguientes componentes básicos [10, Sec. 1]:

- *Una etiqueta o transpondedor:* que se compone de un chip semiconductor, una antena y en ocasiones, una batería.
- *Un interrogador o lector:* que se encuentra transmitiendo constantemente pulsos de energía mediante ondas de radio. Estos pulsos son recibidos por las etiquetas, las mismas que responden al lector con un pulso que contiene un identificador.

- *Un controlador:* que generalmente adopta la forma de una PC o una estación de trabajo que ejecuta un *software* de control para procesar la información.

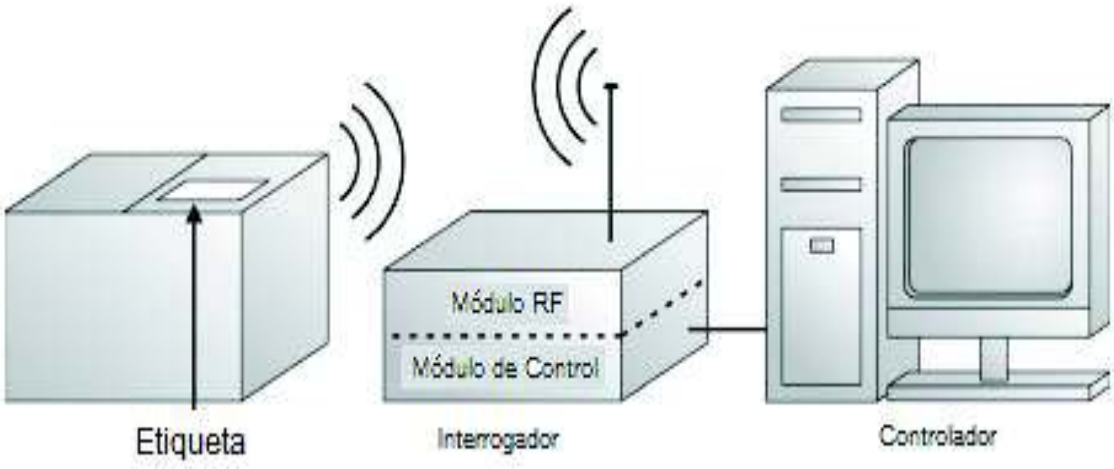


Figura 1.7. Los bloques de construcción básicos de un sistema RFID [10]

La etiqueta y el interrogador comunican información entre ellos a través de ondas de radio. Por lo tanto, cuando un objeto etiquetado entra en la zona de lectura de un interrogador, este envía una señal a la etiqueta para transmitir sus datos almacenados.

Las etiquetas pueden contener muchos tipos de información sobre los objetos a los que están adjuntos, incluidos los números de serie, las instrucciones de configuración y mucho más. Una vez que el interrogador ha recibido los datos de la etiqueta, esa información se transmite al controlador a través de una interfaz de red estándar, como una LAN[19] o incluso Internet.

El controlador puede utilizar esa información para una variedad de propósitos. Por ejemplo, el controlador podría emplear los datos para simplemente inventariar el objeto en una base de datos [10].

b. Etiquetas

La función fundamental de una etiqueta RFID es almacenar datos y transmitir datos al lector. Como se muestra en la Figura 1.8, en su aspecto más básico, una etiqueta consiste en un chip electrónico y una antena. Algunas etiquetas también contienen baterías, esto es lo que diferencia las etiquetas activas de las pasivas.

[19] ***Local Area Network*** **(LAN):** es una red informática dentro de un área geográfica pequeña, como un hogar, una escuela, un edificio de oficinas, etc.

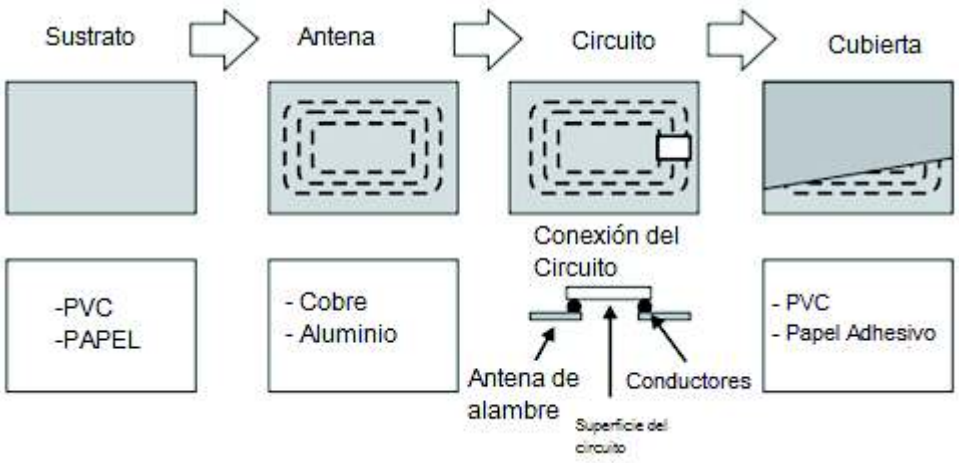


Figura 1.8. Componentes de una etiqueta RFID [10]

b.1. Etiquetas activas y pasivas

Las etiquetas RFID son activas contienen una fuente de alimentación incorporada, como una batería. Cuando la etiqueta necesita transmitir datos al interrogador, utiliza esta fuente para derivar la potencia de la transmisión, del mismo modo que el teléfono celular usa la batería. Debido a esto, las etiquetas activas se pueden comunicar con los interrogadores menos poderosos y pueden transmitir información en rangos mucho mayores, hasta decenas de metros. Además, estos tipos de etiquetas suelen tener memorias más grandes, de hasta 128 KBytes. Sin embargo, son mucho más grandes y más complejas que sus contrapartes pasivas, lo que las hace más costosas de producir.

En contraste, las etiquetas pasivas no tienen fuente de alimentación incorporada. En su lugar, obtienen potencia del interrogador para transmitir datos, aunque mucho menor en comparación a una etiqueta activa. Como resultado de esto, las etiquetas pasivas son típicamente más pequeñas y menos costosas de producir que las etiquetas activas. Sin embargo, el rango efectivo de etiquetas pasivas es mucho más corto que el de las etiquetas activas, a veces por debajo de 50 centímetros. Además, requieren interrogadores más potentes y tienen menos capacidad de memoria, del orden de algunos kilobytes.

Algunas etiquetas pasivas tienen baterías incorporadas, pero no son utilizadas para ayudar en la transmisión de la señal de radio. Estos tipos de etiquetas pasivas se denominan etiquetas asistidas por batería y usan la batería solo para alimentar los componentes electrónicos incorporados.

Por ejemplo, un productor de alimentos puede aplicar etiquetas RFID equipadas con sensores de temperatura a los palés con el fin de controlar la temperatura de su producto durante el envío y el almacenamiento. Por lo tanto, si la temperatura del producto

aumentara por encima de un cierto nivel, el sensor podría arrojar una alarma que alerte a los empleados. Más tarde, en el momento de la entrega o venta, se podría verificar la etiqueta para asegurar un envío a la temperatura adecuada. Las etiquetas pasivas equipadas con este tipo de sensor periférico necesitarían una batería incorporada para operar durante el envío o el almacenamiento.

b.2. Etiquetas de Solo Lectura y Lectura/Escritura

Otro factor de diferenciación entre etiquetas es el tipo de memoria. Fundamentalmente existen dos categorías: de solo lectura “RO” (*Read-Only*) y de lectura/escritura “RW” (*Read/Write*).

La memoria RO únicamente permite la lectura de la información. Las etiquetas RO son similares a los códigos de barras en el sentido de que son programados una vez. Por ejemplo, por un fabricante de productos, y de allí no se pueden alterar, del mismo modo que un CD-ROM[20] no puede ser alterado después de ser quemado en la fábrica. Estos tipos de etiquetas generalmente se programan con una cantidad muy limitada de datos que están destinados a ser estáticos, como los números de serie, y se integran fácilmente en los sistemas de códigos de barras existentes.

Por su parte, las etiquetas RW a menudo se denominadas etiquetas "inteligentes” ofrecen al usuario mucha más flexibilidad que las etiquetas RO ya que permiten almacenar grandes cantidades de datos y tener una memoria direccionable que se puede cambiar fácilmente. Por su parte, los datos en una etiqueta RW pueden borrarse y reescribirse a voluntad.

Existen algunas variaciones en estos dos tipos de memoria que necesitan mencionarse. Primero, la memoria *write-once-read-many* (WORM). Dicha etiqueta es similar a la etiqueta RO ya que debe ser programada con información estática. Además, una etiqueta WORM permite codificar la información en la primera instancia de uso, luego el código se bloquea y no se puede cambiar. Este tipo de memoria se podría utilizar en una línea de ensamblaje para sellar la fecha o ubicación de fabricación en una etiqueta una vez que se complete el proceso de producción.

Por otro lado, algunas etiquetas pueden contener tanto memoria RO como RW al mismo tiempo. Por ejemplo, una etiqueta de RFID conectada a un palé para productos podría marcarse con un número de serie en la sección RO de la memoria, el mismo que permanecería estático durante toda la vida útil del palé. Por otro lado, la sección RW podría

[20] ***Compact Disc-Read-Only Memory*** **(CDROM):** es un dispositivo de almacenamiento que permite lectura, pero no escritura.

usarse para indicar el contenido del *pallet* en cualquier momento dado. Entonces, cuando los productos del palé sean entregados y se vuelva a cargar con mercancía nueva, la sección RW de la memoria podría reescribirse para reflejar el cambio.

b.3. Formas de las etiquetas RFID

Las etiquetas RFID pueden venir en muchas formas y pueden no parecerse a una etiqueta real en absoluto. Debido a que el conjunto de chip/antena en una etiqueta RFID se ha hecho tan pequeño, ahora se pueden incorporar a casi cualquier forma:

- Algunos de los primeros sistemas de RFID se usaron en el manejo del ganado, y las etiquetas eran como pequeñas "balas" de plástico adheridas a las orejas del ganado.
- Las etiquetas RFID utilizadas en los sistemas automáticos de cobro de peajes no son realmente etiquetas, sino tarjetas de plástico o varillas tipo llavero.
- En las aplicaciones de administración de prisiones, las etiquetas RFID se están incorporando a las pulseras usadas por los reclusos y los guardias. Del mismo modo, algunos conductores de FedEx[21] llevan pulseras RFID en lugar de un llavero para acceder a sus furgonetas a través de sistemas de encendido y apagado sin llave.

En resumen, la forma que toma una etiqueta depende en gran medida de la aplicación. Algunas etiquetas deben fabricarse para soportar altas temperaturas, humedad y sustancias químicas, por lo que están recubiertas con materiales protectores. Otras etiquetas están hechas para ser baratas y desechables. La Figura 1.9, ilustra algunos tipos de etiquetas RFID para múltiples aplicaciones.

c. Interrogadores RFID

Un interrogador o lector RFID actúa como un puente entre la etiqueta RFID y el controlador. Sus funciones fundamentales son:

- Leer la información almacenada en la etiqueta RFID
- Escribir datos en la etiqueta (en el caso de las etiquetas inteligentes)
- Retransmitir datos desde y hacia el controlador
- Suministrar energía a la etiqueta (en el caso de las etiquetas pasivas).

[21] **FedEx:** es una empresa de servicios de entrega de mensajería multinacional estadounidense.

Figura 1.9. Etiquetas RFID [11]

Los interrogadores RFID son esencialmente computadoras pequeñas que constan de tres partes básicas: una antena, un módulo de electrónica de RF[22], que es responsable de la comunicación con la etiqueta RFID, y un módulo de electrónica de control, que es responsable de la comunicación con el controlador.

Además de realizar las cuatro funciones básicas anteriores, los interrogadores RFID más complejos pueden realizar funciones más críticas:

- Implementar medidas de anticolisión para asegurar lecturas y escrituras simultánea con varias etiquetas
- Autenticación de etiquetas para evitar el fraude o el acceso no autorizado al sistema
- Cifrado de datos para proteger la integridad de la información.

c.1. Componentes de un interrogador RFID

Los componentes de un lector RFID son:

[22] ***Radio Frequency*** **(RF):** se refiere a la velocidad de oscilación de las ondas de radio electromagnéticas en el rango de 3 kHz a 300 GHz, así como a las corrientes alternas que transportan las señales de radio.

- *Transmisor:* permite el envío de ondas electromagnéticas hacia las etiquetas. Estas ondas producen la activación del circuito integrado que contiene la información que está almacenada en las etiquetas.

- *Receptor:* se encarga de recibir las señales que son enviadas por las etiquetas, en respuesta al estímulo enviado por el lector. Además, realiza la demodulación de las señales de radio que fueron enviadas por cada una de las etiquetas.

- *Circuito integrado:* es el encargado de revisar la integridad de los datos, en caso de que esté permitido el proceso de lectura múltiple. Asimismo, se encargará de la negociación para que este proceso se lleve a cabo con éxito.

- *Memoria:* permite que se almacenen los datos necesarios para el correcto funcionamiento del sistema.

- *Software de procesamiento:* debido a que es necesario que la información recibida por el lector sea procesada, existe el *software* de procesamiento. Este es el responsable de interpretar los datos que se reciben de las etiquetas.

c.2. Lectura y escritura múltiple

Los algoritmos de anticolisión se implementan para permitir que un interrogador se comunique con muchas etiquetas a la vez. Por ejemplo, imagine que un interrogador, al no saber cuántas etiquetas RFID podría haber en su zona de lectura o incluso si hay etiquetas en su zona de lectura, emite un comando general para que las etiquetas transmitan sus datos. Asimismo, imagine que hay unos cientos de etiquetas en la zona de lectura y todos intentan responder a la vez. Ciertamente, se debe hacer un plan para esta contingencia. En RFID se llama anticolisión. Hay tres tipos de técnicas anticolisión: espacial, en frecuencia y dominio de tiempo. Los tres se usan para establecer un orden jerárquico o una medida de aleatoriedad en el sistema, a fin de evitar que ocurra el problema anterior o, al menos, hacer que la ocurrencia sea estadísticamente improbable.

c.3. Autenticación

Los sistemas de alta seguridad también requieren que el interrogador autentique a los usuarios del sistema. Los sistemas de punto de venta, por ejemplo, en los que se intercambia dinero y se carga dinero en cuentas, serían propensos al fraude si no se toman medidas. En este ejemplo de alta seguridad, el procedimiento de autenticación probablemente sería de dos niveles, con parte del proceso en el controlador y parte del proceso en el interrogador.

Existen dos tipos de autenticación, denominados: claves simétricas y derivadas mutuas. En ambos sistemas, una etiqueta RFID proporciona un código al interrogador, que luego se adhiere a un algoritmo, o un "candado", para determinar si la clave se ajusta y si la etiqueta está autorizada para acceder al sistema.

c.4. Cifrado y descifrado de datos

El cifrado de datos es otra medida de seguridad que se debe tomar para evitar ataques externos al sistema. En el ejemplo anterior, imagine que un tercero intercepta la clave de un usuario. Esa información podría usarse para realizar compras fraudulentas, como en una estafa con una tarjeta de crédito. Con el fin de proteger la integridad de los datos transmitidos de forma inalámbrica, y para evitar la intercepción por parte de un tercero, se utiliza el cifrado. Por lo tanto, el interrogador implementa el cifrado y descifrado para realizar esta tarea.

Asimismo, la encriptación es fundamental para contrarrestar el espionaje industrial, el sabotaje industrial y la falsificación.

c.5. Colocación del interrogador y factores de forma

Los sistemas RFID no requieren línea de vista entre etiquetas y lectores, de la misma forma que los sistemas de códigos de barras. Como resultado, los diseñadores de sistemas tienen mucha más libertad al decidir dónde ubicar a los interrogadores. Los interrogadores de posición fija pueden montarse en puertas, a lo largo de cintas transportadoras y en entradas para rastrear el movimiento de objetos a través de cualquier instalación.

Algunas aplicaciones de almacenamiento incluso colocan antenas de interrogador al techo, a lo largo de los pasillos de los estantes, para rastrear el movimiento de las carretillas elevadoras y el inventario.

Los lectores portátiles pueden montarse en carretillas elevadoras, camiones y otros equipos de manipulación de materiales para rastrear palés y otros artículos en tránsito. Incluso hay dispositivos interrogadores portátiles más pequeños que permiten a los usuarios ir a ubicaciones remotas donde no es factible instalar interrogadores de posición fija.

A menudo estos dispositivos portátiles están conectados a una PC o computadora portátil, ya sea de forma inalámbrica o con un cable. Estas PC o computadoras portátiles a su vez se conectan en red al controlador.

d. Controladores RFID

Los controladores RFID son los "cerebros" de cualquier sistema RFID. Se usan para conectar en red múltiples interrogadores de RFID y para procesar centralmente la información.

El controlador en cualquier red suele ser una computadora o estación de trabajo que ejecuta una base de datos o software de aplicación, o una red de estas máquinas. El controlador podría usar la información reunida en el campo por los interrogadores para:

- Mantener el inventario y alertar a los proveedores cuando se necesita un nuevo inventario, como en una aplicación de venta minorista
- Rastrear el movimiento de objetos a lo largo de un sistema
- Verificar la identidad y la autorización de concesión, como en los sistemas de entrada sin llave.

e. Frecuencias

Una consideración clave para RFID es la frecuencia de operación. Así como la televisión se puede transmitir en una banda VHF (*Very High Frequency*) o UHF (*Ultra High Frequency*), también los sistemas RFID pueden usar bandas diferentes para la comunicación, como se muestra en la Figura 1.10.

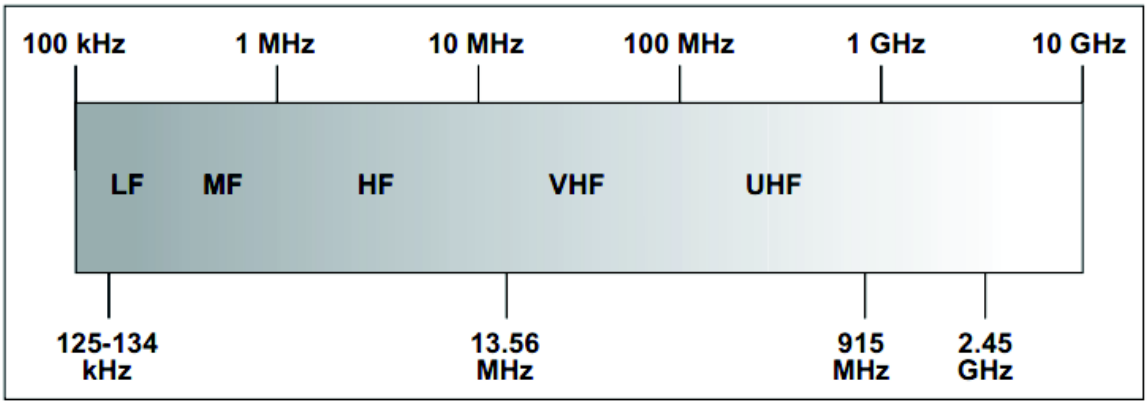


Figura 1.10. Espectro de Radio Frecuencia [10]

En RFID se usan bandas de baja frecuencia y de alta frecuencia de radio, como se muestra en la siguiente lista [10]:

Bandas RFID de baja frecuencia:

- Baja frecuencia LF (*Low frequency*): 125-134 KHz

- Alta frecuencia HF (*High frequency*): 13.56 MHZ

Bandas RFID de alta frecuencia:

- Frecuencia ultra alta UHF (*Ultra-high frequency*): 860-960 MHz
- Microondas (*Microwave*): 2.5 GHz y superior

La elección de la frecuencia afecta varias características de cualquier sistema RFID, como se explica a continuación.

e.1. Rango de lectura

En las bandas de frecuencia más bajas, los rangos de lectura de las etiquetas pasivas no son superiores a una decena de centímetros, debido principalmente a la escasa ganancia de la antena, puesto que la ganancia de la misma es directamente proporcional al tamaño de la antena en relación con la longitud de onda. Por lo tanto, la ganancia de la antena en estas frecuencias es muy baja. A frecuencias más altas, el rango de lectura generalmente aumenta, especialmente cuando se usan etiquetas activas. Sin embargo, se han establecido límites de potencia en los sistemas de UHF y microondas, reduciendo el rango de lectura de estos sistemas de alta frecuencia de 3 a 10 metros en promedio en el caso de etiquetas pasivas.

e.2. Etiquetas pasivas vs. etiquetas activas

Por razones históricas, las etiquetas pasivas se operan normalmente en las bandas de ondas kilométricas y decamétricas, mientras que las etiquetas activas se utilizan normalmente en las bandas de ondas decimétricas y de microondas. Los primeros sistemas RFID usaron la banda HF y LF con etiquetas pasivas. Sin embargo, los avances recientes en la tecnología han hecho posible el uso de etiquetas activas y las bandas de frecuencia más altas, y esta se ha convertido en la tendencia de la industria.

e.3. Interferencia de otros sistemas de radio

Los sistemas RFID son propensos a la interferencia de otros sistemas de radio. Además, los sistemas operan en la banda LF son particularmente vulnerables, debido a que las frecuencias LF no experimentan mucha pérdida de trayectoria, o se atenúan muy poco en distancias cortas, en comparación con las frecuencias más altas. Esto significa que las señales de radio de otros sistemas de comunicación que operan a casi la misma frecuencia LF tendrán altas intensidades de campo en la antena de un interrogador RFID, lo que puede traducirse en interferencia. En el otro extremo del espectro, los sistemas microondas son

los menos susceptibles a la interferencia, ya que la pérdida de trayectoria en la banda de microondas es mucho mayor para las frecuencias más bajas y generalmente se requiere una línea de vista para que los radiadores de microondas interfieran.

e.4. Líquidos y metales

El rendimiento de los sistemas de RFID se verá afectado negativamente por el agua o las superficies mojadas. Por su parte, las señales de HF, debido a sus longitudes de onda relativamente largas, son más capaces de penetrar el agua que las señales de UHF y microondas. Las señales en las bandas de alta frecuencia son más propensas a ser absorbidas en un líquido. Como resultado, las etiquetas de HF son una mejor opción para etiquetar recipientes que contienen líquidos.

Por otro lado, el metal es un reflector electromagnético y las señales de radio no pueden penetrarlo. Como resultado, el metal no solo obstruirá la comunicación si se coloca entre una etiqueta y un interrogador, sino que solo la presencia cercana de metal puede tener efectos adversos en el funcionamiento de un sistema; cuando se coloca metal cerca de cualquier antena, las características de esa antena cambian y se puede producir un efecto perjudicial.

Las bandas de alta frecuencia se ven afectadas por el metal más que las bandas de frecuencia más bajas. Para etiquetar objetos hechos de metal, contenedores que llevan líquido o materiales con alta permitividad dieléctrica, se deben tomar precauciones especiales, lo que en última instancia aumenta los costos.

e.5. Velocidad de datos

Los sistemas RFID que operan en la banda LF tienen velocidades de datos relativamente bajas, del orden de Kbps. Las velocidades de datos aumentan con la frecuencia de operación, alcanzando el rango de Mbps en las frecuencias de microondas. La Tabla 1.1 expone las velocidades promedio de datos.

Tabla 1.1 Velocidad de datos en RFID [10]

Frecuencia	LF	HF	UHF	Microondas
Velocidad	1 a 10 Kbps	1 a 50 Kbps	1 a 160 Kbps	1 a 10 Mbps

1.3.5 Metodologías de desarrollo Ágiles

En el año 2001, se estableció el manifiesto *Agile* que resume el comportamiento de las metodologías ágiles a través de cuatro principios fundamentales [12, Sec. 2]:

- Se valora a los individuos y las interacciones sobre los procesos y las herramientas
- Se valora a las aplicaciones que funcionan sobre la documentación exhaustiva
- Se valora la colaboración del cliente sobre las negociaciones contractuales
- Se valora la respuesta al cambio sobre el seguimiento de un plan

Las metodologías ágiles adoptan un enfoque iterativo para el desarrollo de software. A diferencia de un modelo de cascada lineal tradicional, los proyectos ágiles consisten en una serie de ciclos más pequeños denominados iteraciones o *sprints*. Cada uno de ellos es un proyecto en miniatura y posee sus propias etapas de diseño, implementación y pruebas, dentro del alcance de trabajo predefinido. Además, al priorizar la flexibilidad y el cambio rápido, el enfoque ágil ofrece los siguientes beneficios:

- Capacidad de gestionar las prioridades cambiantes
- Aumento de la productividad del equipo a través de la asignación de tareas diarias
- Mejor visibilidad de las tareas del proyecto debido al sistema de planificación simple.

Por último, *Agile* es un término general para una gran variedad de metodologías y técnicas, que comparten los principios y valores descritos anteriormente. Cada uno de ellos tiene sus propias áreas de uso y características distintivas.

a. Metodología Kanban

Kanban es una metodología ágil muy utilizada en la actualidad, según reportes, se usa de alguna forma en un 43% de las organizaciones ágiles [13, p. 3]. Originario de un sistema visual de tarjetas utilizado en las fábricas de Toyota como método de control de producción, Kanban es un enfoque muy simple pero eficiente para desarrollar productos de software.

Además, Kanban prioriza el trabajo en proceso (TEP), limitando su alcance para que coincida efectivamente con la capacidad del equipo. Tan pronto como se completa una tarea, el equipo puede tomar el siguiente elemento de trabajo. Por lo tanto, el proceso de desarrollo ofrece mayor flexibilidad en la planificación, plazos de entrega más cortos, objetivos claros y transparencia.

Por otra parte, en Kanban no se requieren procedimientos estándar, así como las iteraciones fijas, a diferencia de otras metodologías ágiles (e.g. Scrum). El desarrollo del proyecto se basa en la visualización del flujo de trabajo a través de un tablero Kanban

(Figura 1.11), generalmente representado por notas adhesivas y pizarras blancas o herramientas en línea.

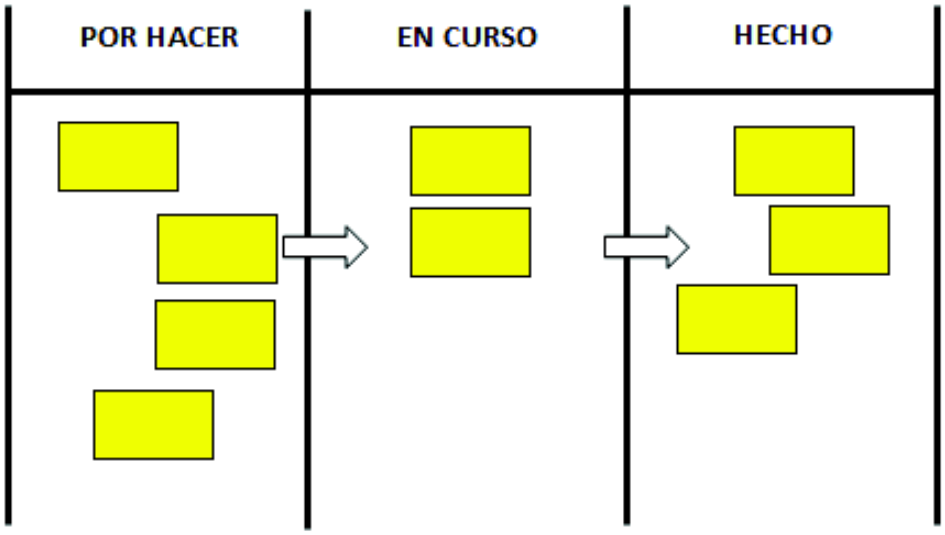


Figura 1.11. Tablero Kanban [14]

Finalmente, el concepto central de Kanban comprende:

- Visualizar el flujo de trabajo: Se divide todo el trabajo en segmentos o piezas. Cada elemento es representado en una tarjeta y colocado en una columna dentro tablero para su identificación.
- Limitar el trabajo en proceso (TEP): se asigna límites explícitos a la cantidad de elementos que pueden estar en progreso en cada estado del flujo de trabajo.
- Medir el tiempo de entrega: se procura optimizar el proceso para hacer que el tiempo de entrega sea lo más pequeño y predecible posible.

2. METODOLOGÍA

En primer lugar, en la Sección 2.1 se presenta el análisis de requerimientos del sistema prototipo para la determinación de las funcionalidades del mismo. Luego, en la Sección 2.2 con base en el análisis se detalla el diseño del sistema. Finalmente, en la Sección 2.3 se describe la implementación del prototipo.

2.1 Análisis de requerimientos

El análisis de requerimientos es la primera etapa en el proceso de desarrollo de *software* y abarca las tareas que determinan las necesidades o condiciones que debe cumplir el sistema prototipo. Por otro lado, los requerimientos deben ser procesables, medibles, comprobables y deben estar relacionados directamente con las necesidades identificadas del usuario. Además, deben estar definidos con un nivel de detalle suficiente para el diseño del sistema.

La obtención de los requerimientos del sistema prototipo, se realizó a partir del análisis de las soluciones comerciales existentes en el mercado (e.g. Regi Robo de Panasonic). Esta solución comercial deja marcado un precedente con respecto a la viabilidad del proyecto y permite establecer los requerimientos base del mismo.

Asimismo, se realizaron encuestas a los usuarios potenciales del sistema, con el objetivo de conocer sus expectativas y determinar sus necesidades.

2.1.1 Descripción de las soluciones comerciales existentes

Lawson, Inc. y Panasonic desarrollaron el sistema robótico de pago automatizado Regi-Robo para compras en un supermercado. Su primera demostración experimental tuvo lugar en Osaka - Japón en febrero del 2017, en una tienda Lawson apoyada por el ministerio de Economía, Comercio e Industria del país [15]. Cada producto dentro de la tienda tiene adherido una etiqueta RFID como se muestra en la Figura 2.1.

Al iniciar su compra, el cliente debe tomar una canasta especialmente diseñada para funcionar con el sistema (Figura 2.2) donde podrá depositar los productos de su compra.

Una vez que el cliente haya seleccionado todos los productos de su compra, debe acercarse al módulo robótico de *checkout* como se ilustra en la Figura 2.3. Dicho módulo posee un compartimiento dedicado para la canasta, un lector o interrogador RFID y una pantalla táctil para interactuar con el usuario.

Figura 2.1. Producto con etiqueta RFID adherida [15]

Figura 2.2. Canasta especial para el sistema Regi-Robo [15]

Figura 2.3. Sistema Robótico de Checkout [15]

Luego, como se muestra en la Figura 2.4, el cliente debe depositar la canasta en un compartimiento del módulo de *checkout*.

Figura 2.4. Compartimiento para la canasta [15]

Posteriormente, el fondo de la canasta se abre dejando caer los productos en una funda plástica (Figura 2.5). Mientras tanto, el sistema robótico realiza la lectura de las etiquetas RFID gracias al interrogador que posee.

Figura 2.5. Mecanismo de empacado del sistema Regi-Robo [15]

En breve, el detalle de los productos se muestra en la pantalla para la verificación y el posterior pago del cliente (Figura 2.6). El sistema permite la realización de pagos mediante efectivo o tarjeta de crédito.

Figura 2.6. Pantalla del sistema Regi-Robo [15]

Finalmente, una vez que el cliente realiza el pago, el sistema devuelve la funda con los productos de la compra (Figura 2.7).

Figura 2.7. Devolución de los productos empacados [15]

Ciertamente, el sistema robótico de pago automatizado Regi-Robo es la solución comercial existente más similar al prototipo propuesto, ya que brinda un modelo alternativo de pago al cliente de un supermercado.

Sin embargo, cabe mencionar que existen otras soluciones comerciales que hacen uso de sistemas RFID para diferentes áreas de aplicación, por ejemplo, para la administración de materiales e inventarios [16], seguimiento vehicular [17], trazabilidad de productos [18], etc.

2.1.2 Encuestas para la determinación de los requerimientos

Para la obtención de los requerimientos del cliente se realizaron 82 encuestas a una muestra de estudiantes universitarios de entre 20 a 25 años de edad, mediante la plataforma Google Forms[23]. A continuación, se detallan las cinco preguntas de la encuesta y se adjunta el gráfico circular con las respuestas correspondientes:

- *Pregunta 1.* Tomando en cuenta la gran afluencia de clientes a las cajas rápidas de los supermercados ¿Considera que sería útil otro sistema de pago para las

[23] **Google Forms:** es un *software* que permite la realización de formularios para encuestas y obtención de estadísticas.

compras en dichas cajas a más de los métodos de pago tradicionales (efectivo, tarjeta de crédito/débito)? Respuestas: Sí, No.

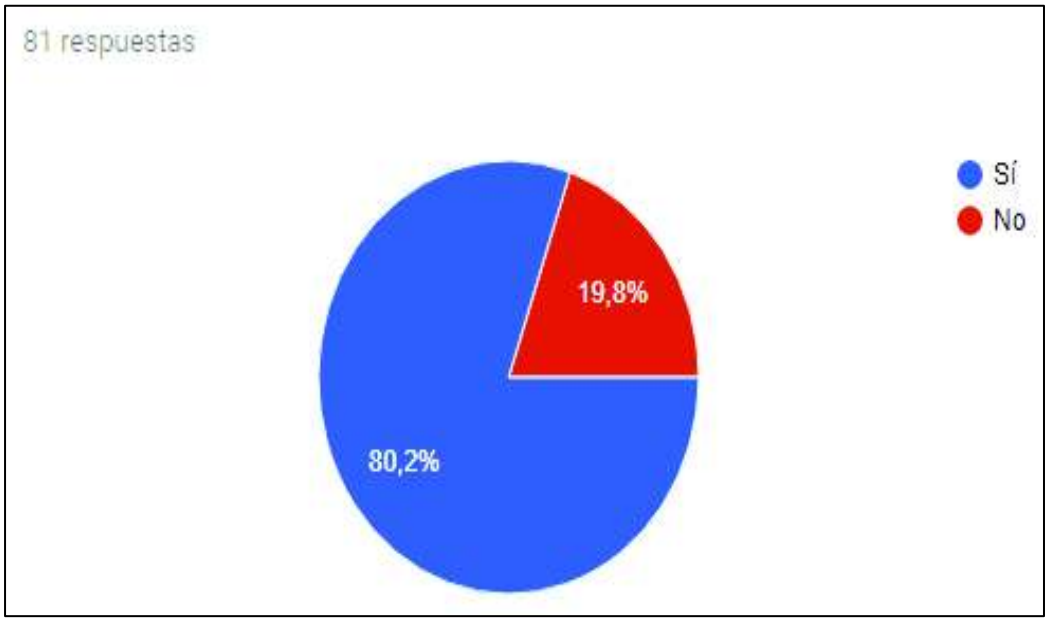


Figura 2.8. Gráfico circular de las respuestas a la pregunta 1

- *Pregunta 2.* ¿Considera que los sistemas de auto-pago, aquellos donde el cliente selecciona sus productos, los empaca y los paga sin la intervención del personal del establecimiento, son una alternativa de pago útil? Respuestas: Sí, No, A veces.

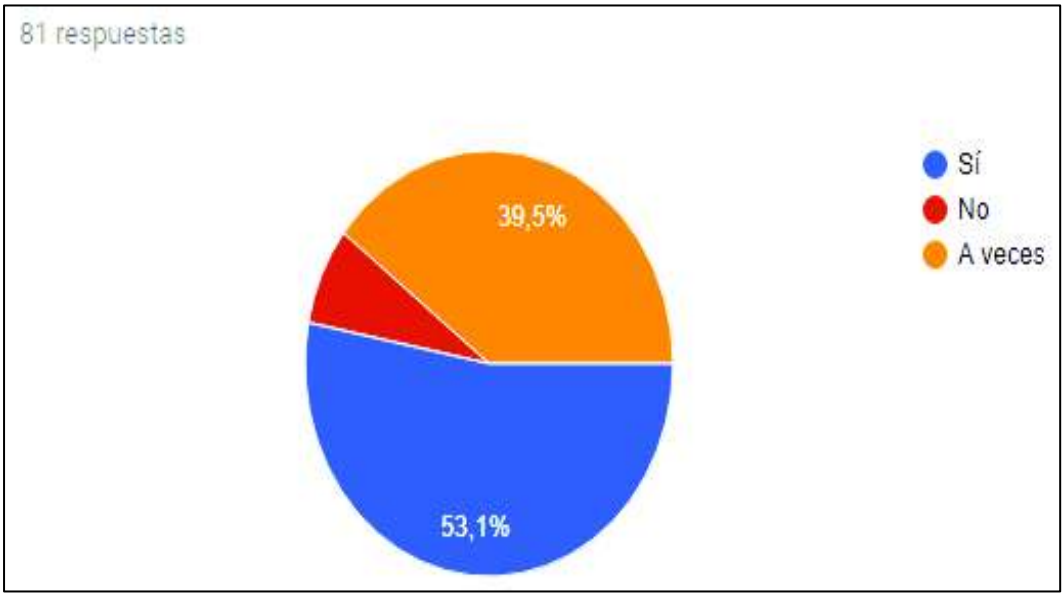


Figura 2.9. Gráfico circular de las respuestas a la pregunta 2

- *Pregunta 3.* ¿Estaría dispuesto a utilizar su teléfono móvil para realizar el pago de sus compras del supermercado? Respuestas: Sí, No.

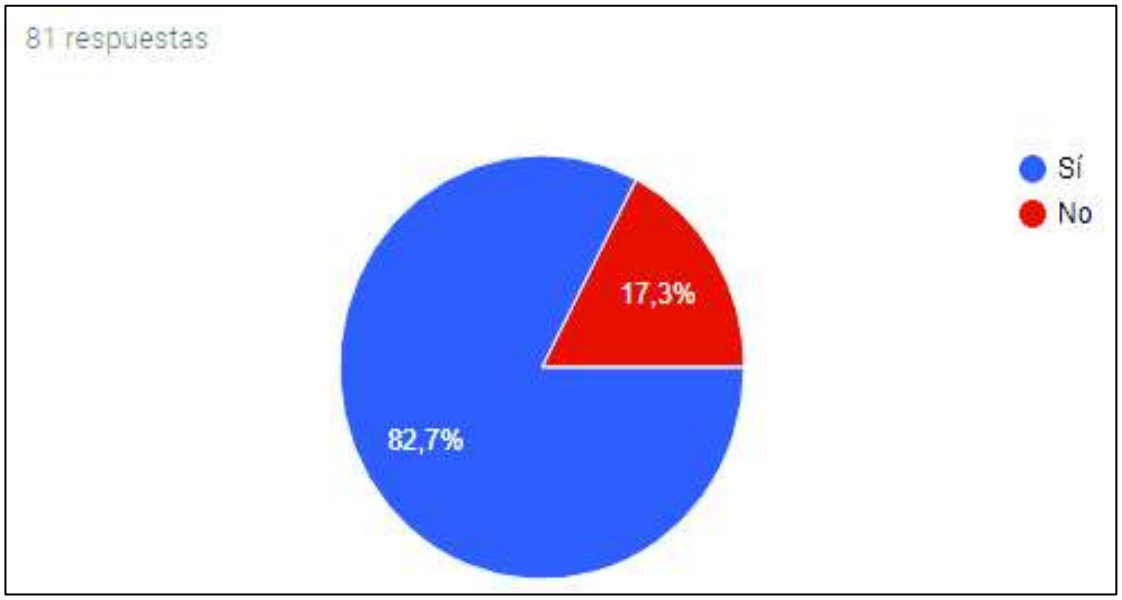


Figura 2.10. Gráfico circular de las respuestas a la pregunta 3

- *Pregunta 4.* Considerando una compra para una caja rápida (máximo 10 productos). ¿Estaría dispuesto a utilizar un sistema de auto-pago que haga uso de una aplicación móvil a manera de billetera electrónica para realizar su compra? Respuestas: Sí, No.

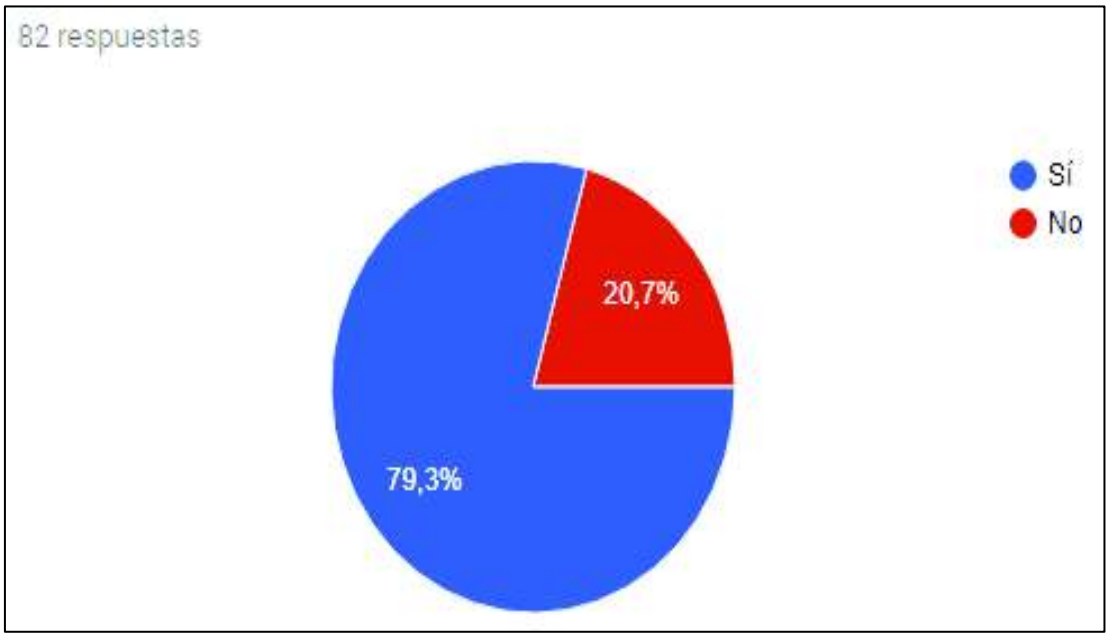


Figura 2.11. Gráfico circular de las respuestas a la pregunta 4

- *Pregunta 5.* ¿Qué tan cómodo se sentiría utilizando el sistema de auto-pago descrito en la pregunta anterior? Considere 5 como muy cómodo y 1 como nada cómodo. Respuestas: 5,4,3,2,1.

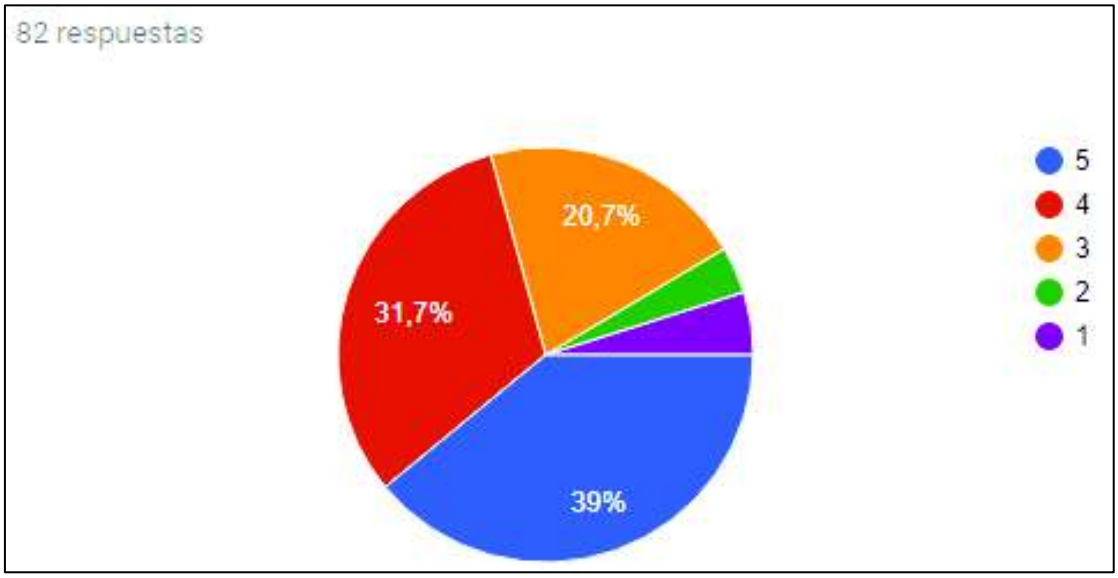


Figura 2.12. Gráfico circular de las respuestas a la pregunta 5

2.1.3 Obtención de requerimientos

Según la Figura 2.8, un total del 80,2% de los encuestados consideran que sería útil un sistema alternativo para el pago de compras en las cajas rápidas de un supermercado. Además, consideran que los sistemas de auto-pago, donde el cliente selecciona sus productos, los empaca y los paga sin la intervención del personal del establecimiento son una alternativa de pago útil (Figura 2.9).

Asimismo, la Figura 2.10 revela que el 82,7% de los usuarios encuestados estaría dispuesto a utilizar su teléfono móvil para realizar dichas compras. De la misma manera, de acuerdo con los resultados mostrados en la Figura 2.11, un total del 79,3% de los encuestados estaría dispuesto a utilizar un sistema de auto-pago para realizar dichas compras.

Por lo tanto, conforme a los resultados arrojados por la encuesta, el usuario final estaría dispuesto a utilizar un sistema de auto-pago que haga uso de una aplicación móvil a manera de billetera electrónica para realizar sus compras. Por último, la Figura 2.12 arroja una idea respecto a la experiencia de utilización del sistema prototipo por parte del usuario, donde el 51,14% de la muestra piensa que se sentirían cómodos utilizando este sistema.

Adicionalmente, las respuestas individuales a cada una de las encuestas realizadas se presentan en el Anexo I.

Por otro lado, conforme a la descripción realizada de la solución comercial existente Regi-Robo, un sistema de auto-pago que haga uso de la tecnología RFID podría ser una alternativa útil y viable para el pago de compras en establecimientos comerciales.

En consecuencia, el sistema prototipo de auto-pago propuesto hará uso de la tecnología RFID para la identificación de los productos empleando una aplicación móvil para efectivizar el pago.

2.2 Diseño del sistema

Dado que, la extracción de requerimientos fue realizada en la sección anterior. A continuación, se elabora el diseño del sistema. Para lo cual, se emplea la metodología de desarrollo ágil Kanban.

Cada producto del supermercado tiene adherido una etiqueta RFID para su identificación. Por su parte, el cliente selecciona los productos que desee de las perchas y los guarda en una bolsa reutilizable. Como se ilustra en la Figura 2.13, el prototipo está compuesto por cuatro componentes esenciales:

- *La base de datos:* almacena la información del supermercado, i.e. los identificativos únicos de cada producto, la información de los empleados, las cuentas de los clientes con sus respectivas credenciales y saldo disponible, el detalle de las compras realizadas por cada cliente y la información de las cajas.
- *La caja electrónica:* está compuesta por un computador conectado a un interrogador RFID, el cual permite la lectura de los productos a ser adquiridos mediante un *software* de control que realiza el procesamiento de la información. La función principal de la caja es permitir al sistema la identificación de los productos mediante las etiquetas que llevan.
- *La aplicación web:* permite al personal del supermercado la administración de las cuentas de los clientes, la gestión de los productos del supermercado y la generación de reportes de compras de cada cliente. Además, la aplicación distingue dos tipos de usuario: el administrador y el personal de servicio al cliente. El usuario administrador goza de control total sin restricciones, mientras que el usuario de servicio al cliente tiene acceso únicamente a opciones relativas a las

cuentas de los clientes y sus facturas. Conviene señalar que el cliente debe crear una cuenta y realizar una recarga de dinero para contar con saldo disponible para sus compras. Como se mencionó, la creación de cuentas se podrá realizar mediante la aplicación web en el punto de servicio al cliente dentro del establecimiento.

- *La aplicación Android:* funciona a manera de una billetera electrónica. En primer lugar, el cliente debe ingresar sus credenciales para la autenticación de su cuenta. Una vez autenticado, podrá visualizar en pantalla las cajas disponibles y tendrá que seleccionar una de ellas para su utilización. Entonces, deberá pasar uno a uno sus productos por el interrogador de la caja para su lectura. Finalmente, podrá verificar el detalle de los productos y proceder a realizar el pago respectivo.

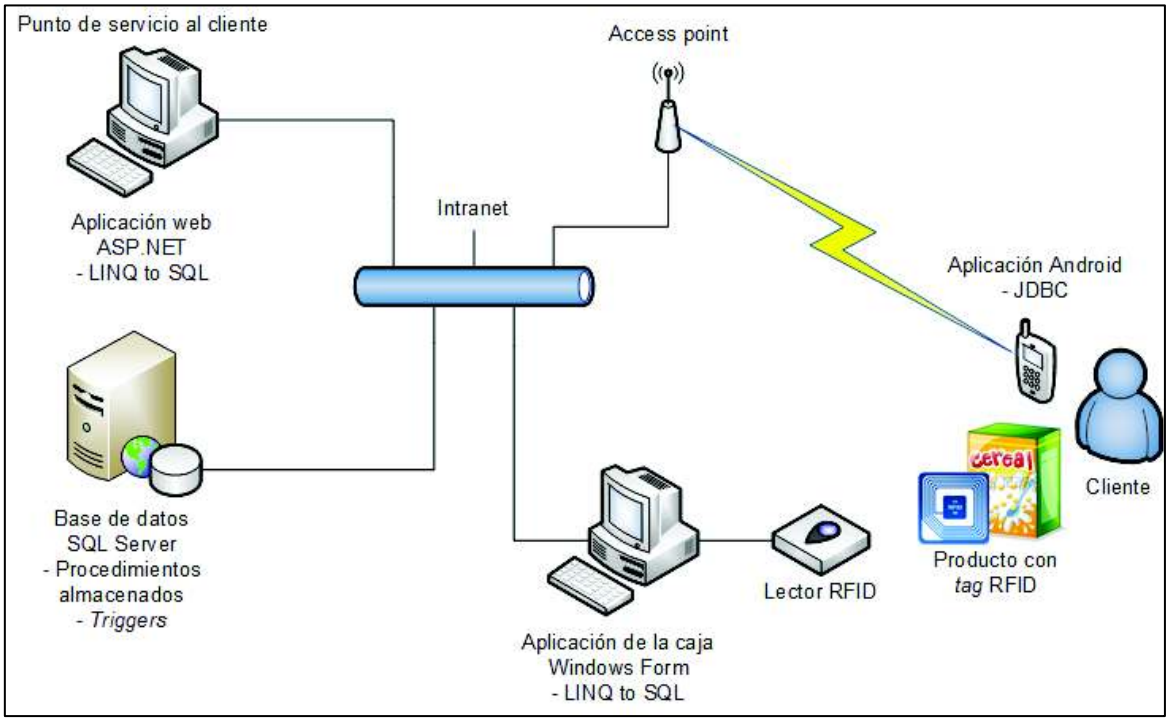


Figura 2.13. Componentes del prototipo

2.2.1 Diagramas de caso de uso

En la presente sección se exhiben los diagramas de casos de uso del prototipo para ilustrar la relación de las funcionalidades con sus actores respectivos. El cliente es el actor fundamental de la aplicación Android, como se ilustra en la Figura 2.14.

La aplicación le permite al cliente autenticarse en el sistema del supermercado. Asimismo, le brinda la posibilidad de visualizar su saldo disponible, realizar la lectura y el pago de sus compras mediante una caja y obtener un listado de sus facturas.

Por otro lado, la aplicación web dispone de dos actores, el usuario del punto de servicio al cliente y el administrador (Figura 2.15). El administrador puede hacer uso de todos los módulos de gestión del establecimiento, entre ellos, la administración de productos, clientes, usuarios y reportes de ventas. Por su parte, el usuario de servicio al cliente tiene acceso a los módulos referentes a las cuentas de los clientes y la generación de sus facturas.

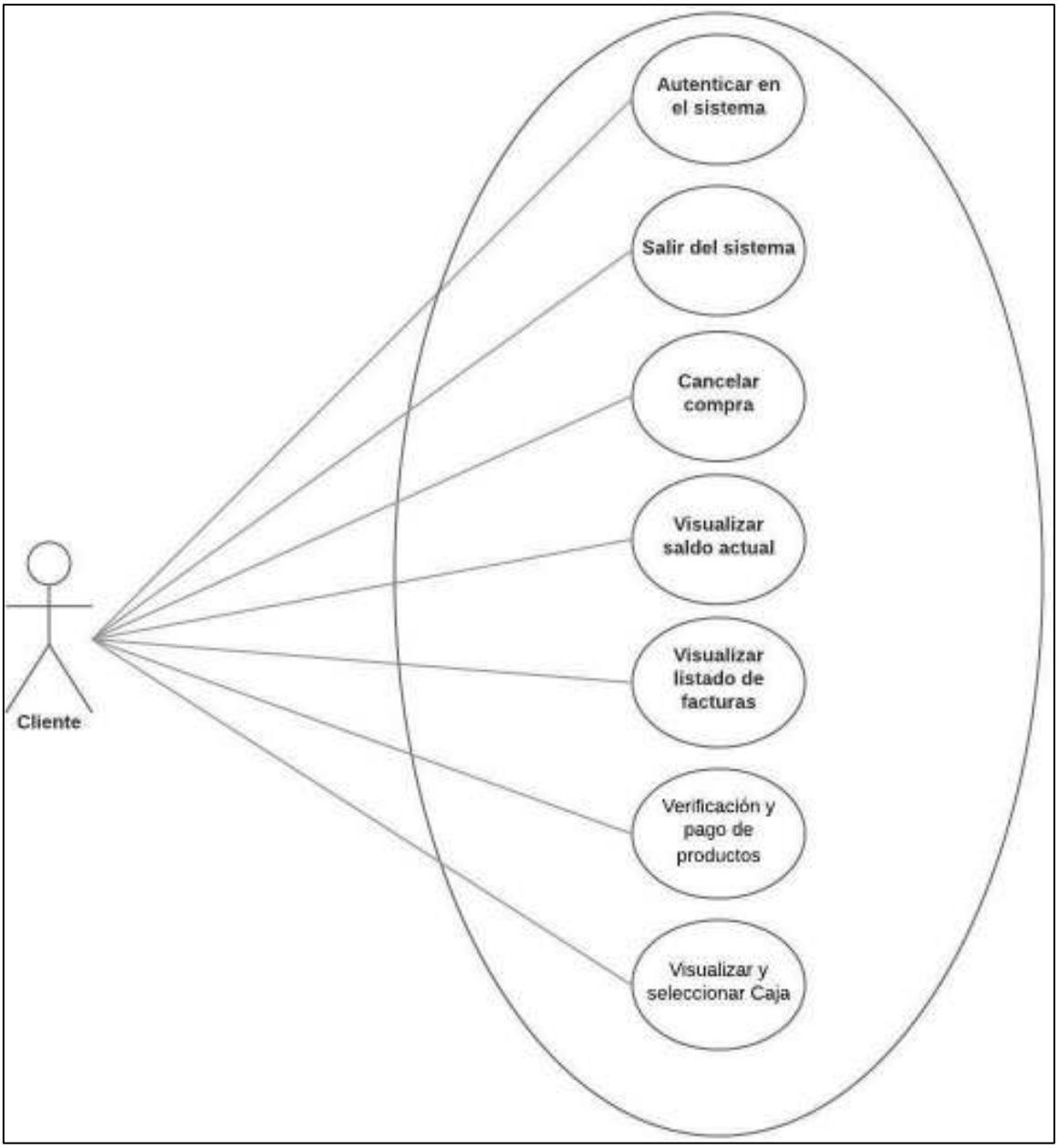


Figura 2.14. Diagrama de casos de uso de la aplicación Android

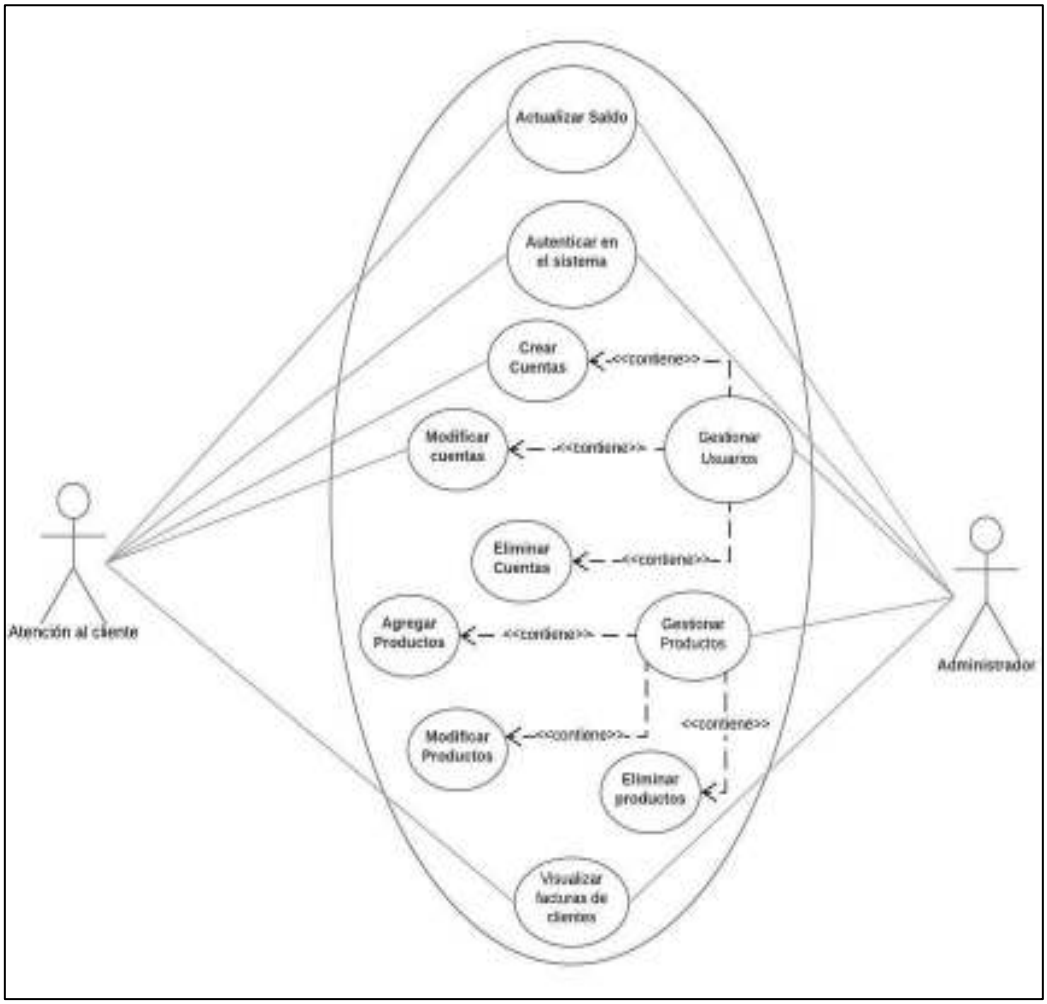


Figura 2.15. Diagrama de casos de uso de la aplicación web

2.2.2 Diagrama de actividades

La Figura 2.16 presenta el diagrama de actividades que ilustra el comportamiento del flujo de trabajo del sistema prototipo. Asimismo, representa la naturaleza dinámica del sistema al modelar el flujo de control de una actividad a otra.

Para empezar, el cliente que haya descargado la aplicación debe asegurarse de tener una cuenta creada en el sistema, para lo cual debe acercarse al punto de servicio al cliente del establecimiento. Cuando tenga su cuenta debe conectarse a la red local del supermercado para autenticarse en el sistema. Entonces, debe seleccionar una caja electrónica para su utilización. Luego, debe acercarse a esta caja para realizar la lectura de los productos de su compra. A continuación, debe verificar la lectura correcta de sus productos y su saldo disponible. Si estos son correctos, entonces puede hacer efectivo el pago. Si no cuenta con el saldo suficiente tendrá que acercarse al punto de servicio al cliente para realizar una recarga. Finalmente, el usuario puede visualizar el listado de sus facturas y finalizar la aplicación móvil.

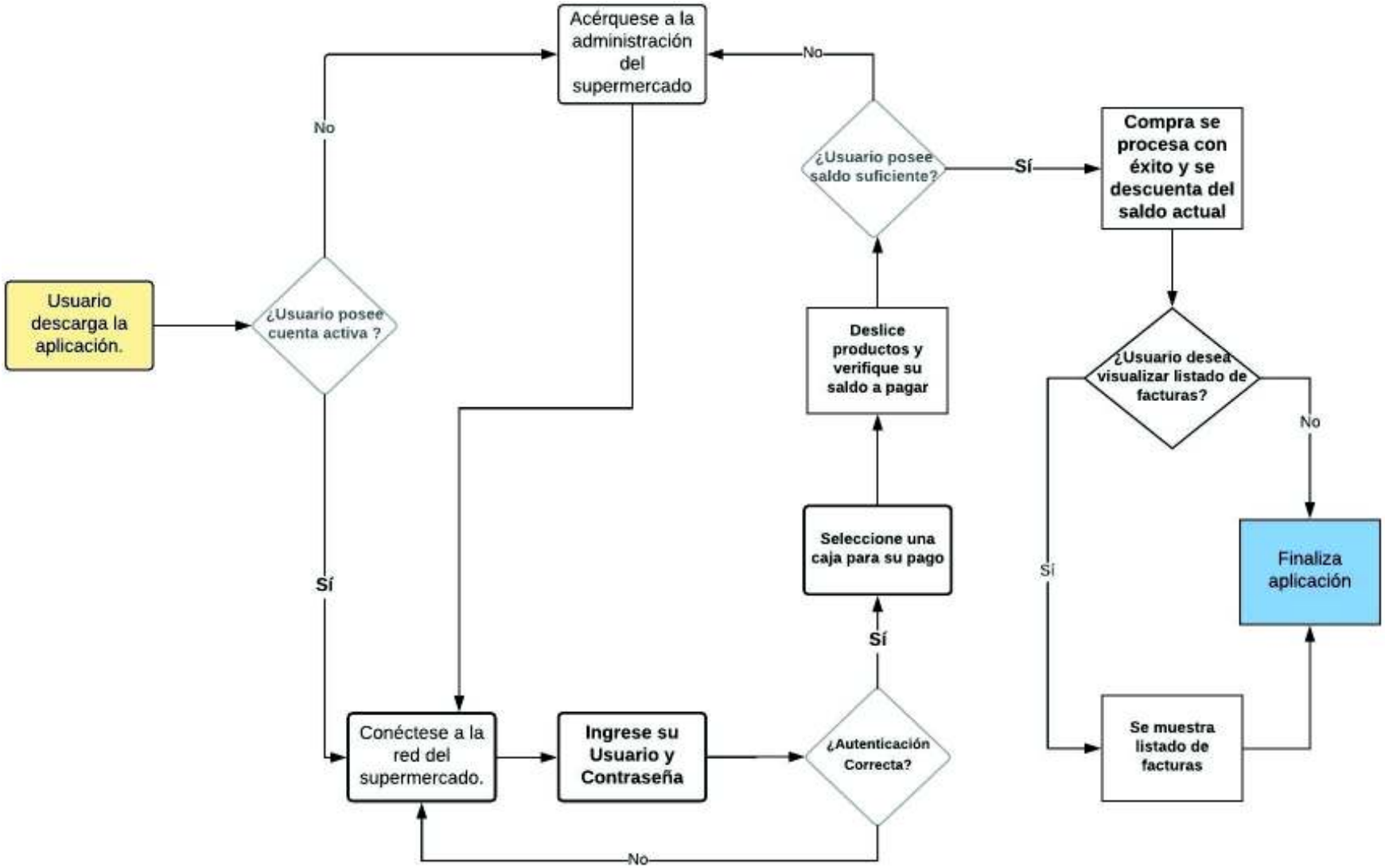


Figura 2.16. Diagrama de actividades del sistema prototipo

2.2.3 Historias de usuario

Las historias de usuario son descripciones breves y simples de un requerimiento del sistema, generalmente contados desde la perspectiva de un cliente o usuario final. Además, deben contener la información suficiente para que los desarrolladores puedan calcular una estimación razonable del esfuerzo para implementar los requerimientos [19].

La Tabla 2.1 presenta el formato utilizado para las historias de usuario del sistema propuesto.

Tabla 2.1. Formato para la escritura de las historias de usuario

<table>
<tr><td>Código: Especifica un identificador para cada historia de usuario.

HUSM#: Historia de usuario de la aplicación móvil.

HUSW#: Historia de usuario de la aplicación web.

HUSB#: Historia de usuario de la base de datos.</td><td>Fecha: Determina cuándo fue escrita o corregida la historia de usuario.</td><td>Autor: Indica la persona que creó la historia de usuario.</td></tr>
<tr><td colspan="2">Programador: Determina el programador asignado para el desarrollo de la historia de usuario.</td><td>Estimación: Delimita el tiempo aproximado (en horas), para el desarrollo en código de la historia de usuario.</td></tr>
<tr><td colspan="3">Prioridad: Especifica el nivel de prioridad para el desarrollo (1 mayor prioridad, 5 menor prioridad).</td></tr>
<tr><td colspan="3">Descripción:

Detalla en lenguaje sencillo el requerimiento solicitado por el usuario final o el cliente.</td></tr>
<tr><td colspan="3">Observación:

Proporciona información adicional relevante o determina condiciones específicas para la historia de usuario.</td></tr>
</table>

a. Base de datos

La primera iteración de historias de usuario abarca los detalles del funcionamiento general de la base de datos del prototipo. La Tabla 2.2 expone la historia de usuario correspondiente al desarrollo de la base de datos del sistema.

Tabla 2.2. Historia de usuario Desarrollo de la base de datos

Código: HUSB01	**Fecha**: 2017-11-06	**Autor:** Daniel Hwang
Programador: Programador 2		**Estimación:** 16 horas
Prioridad: 1		
Descripción: Desarrollo de la base de datos para el prototipo.		
Observación: Se almacenará toda la información del supermercado.		

b. Aplicación móvil

La segunda iteración de historias de usuario abarca los detalles del funcionamiento general de la aplicación Android. Enseguida, se precisan los aspectos más representativos que serán implementados en la misma:

- La autenticación del cliente mediante sus credenciales personales (Tabla 2.3)
- La gestión de las cajas disponibles y la selección de una en particular para su utilización (Tabla 2.4)
- La visualización de los productos escaneados en la caja electrónica (
- Tabla 2.5)
- La visualización del detalle a pagar (Tabla 2.6)
- La posibilidad de anular la compra (Tabla 2.7)
- La realización del pago (Tabla 2.8)
- La visualización del listado de facturas (Tabla 2.9)

Tabla 2.3. Historia de usuario: Autenticación en la aplicación móvil

Código: HUSM01	**Fecha**: 2017-12-12	**Autor:** Daniel Tenemaza
Programador: Programador 1		**Estimación:** 10 horas
Prioridad: 1		

Descripción: Autenticación del cliente en la aplicación móvil mediante un nombre de usuario y una contraseña.

Tabla 2.4. Historia de usuario: Gestión de las cajas disponibles

Código: HUSM02	**Fecha**: 2017-12-14	**Autor:** Daniel Hwang
Programador: Programador 2		**Estimación:** 10 horas
Prioridad: 1		
Descripción: Visualización de las cajas disponibles y selección de una de ellas para su utilización.		

Tabla 2.5. Historia de usuario: Visualización de los productos escaneados

Código: HUSM03	**Fecha**: 2017-12-16	**Autor:** Daniel Tenemaza
Programador: Programador 1		**Estimación:** 15 horas
Prioridad: 1		
Descripción: Visualización del listado de productos escaneados en la caja electrónica.		
Observación: Cuando la lectura de los productos es incorrecta, se debe brindar la posibilidad de volver a realizar la lectura de los mismos.		

Tabla 2.6. Historia de usuario: Visualización del detalle a pagar por la compra

Código: HUSM04	**Fecha**: 2017-12-18	**Autor:** Daniel Hwang
Programador: Programador 2		**Estimación:** 15 horas
Prioridad: 1		
Descripción: Visualización previa del detalle a pagar por la compra.		

Observación: El usuario podrá verificar el saldo disponible en su cuenta.

Tabla 2.7. Historia de usuario: Anulación de la compra

Código: HUSM05	**Fecha**: 2017-12-18	**Autor:** Daniel Hwang
Programador: Programador 2		**Estimación:** 10 horas
Prioridad: 1		
Descripción: Habilitar la posibilidad de anular la compra.		

Tabla 2.8. Historia de usuario: Realización del pago de la compra

Código: HUSM06	**Fecha**: 2017-12-19	**Autor:** Daniel Tenemaza
Programador: Programador 1		**Estimación:** 15 horas
Prioridad: 1		
Descripción: Realización del pago de la compra.		
Observación: Se descontará el valor de la compra del saldo disponible del cliente.		

Tabla 2.9. Historia de usuario: Visualización del listado de facturas

Código: HUSM07	**Fecha**: 2017-12-20	**Autor:** Daniel Tenemaza
Programador: Programador 1		**Estimación:** 15 horas
Prioridad: 3		
Descripción: Visualización del listado de facturas de las compras.		

c. Aplicación web

La tercera iteración de historias de usuario abarca los detalles del funcionamiento general de la aplicación web para la administración del supermercado. A continuación, se precisan los aspectos más representativos:

- La autenticación del personal del supermercado mediante sus credenciales (Tabla 2.10)
- La administración de los perfiles de empleados (Tabla 2.11)
- La gestión de las áreas de trabajo con sus respectivos encargados, por ejemplo, las bodegas, las perchas, el inventario, etc. (Tabla 2.12)
- La administración de las cuentas de los clientes con sus credenciales y saldo correspondientes (Tabla 2.13)
- La administración de los registros de los productos (Tabla 2.14)
- La gestión de las categorías de productos (Tabla 2.15)
- El manejo de un inventario de productos (Tabla 2.16)
- La generación de reportes de ventas del establecimiento (Tabla 2.17)
- La generación de facturas (Tabla 2.18).

Tabla 2.10. Historia de usuario: Autenticación en la aplicación web

Código: HUSW01	**Fecha**: 2018-01-04	**Autor:** Daniel Tenemaza
Programador: Programador 1		**Estimación:** 10 horas
Prioridad: 1		
Descripción: Autenticación del personal del supermercado en la aplicación web mediante un nombre de usuario y una contraseña.		
Observación: La aplicación debe manejar dos perfiles de usuario: 1. Usuario administrador que tiene acceso a todos los módulos. 2. Usuario de atención al cliente que tiene acceso únicamente a opciones relativas a las cuentas de clientes y sus facturas.		

Tabla 2.11. Historia de usuario: Administración de perfiles de empleados

Código: HUSM02	**Fecha**: 2018-01-05	**Autor:** Daniel Hwang
Programador: Programador 2		**Estimación:** 10 horas
Prioridad: 1		
Descripción: Administración de los perfiles de empleados del supermercado.		
Observación: Se debe permitir la lectura, creación, modificación y eliminación de los registros de empleados.		

Tabla 2.12. Historia de usuario: Administración de las áreas de trabajo

Código: HUSM02	**Fecha**: 2018-01-05	**Autor:** Daniel Tenemaza
Programador: Programador 1		**Estimación:** 10 horas
Prioridad: 2		
Descripción: Administración de las áreas de trabajo del supermercado con sus respectivos encargados.		

Tabla 2.13. Historia de usuario: Administración de las cuentas de los clientes

Código: HUSM02	**Fecha**: 2018-01-06	**Autor:** Daniel Tenemaza
Programador: Programador 1		**Estimación:** 10 horas
Prioridad: 1		
Descripción: Administración de las cuentas de los clientes. Se especificará la información personal del cliente (nombres, apellidos, dirección, etc.). Así como, un nombre de usuario, una contraseña y el saldo de la cuenta.		

Tabla 2.14. Historia de usuario: Administración de los registros de productos

Código: HUSM02	**Fecha**: 2018-01-10	**Autor:** Daniel Hwang
Programador: Programador 2		**Estimación:** 10 horas
Prioridad: 1		
Descripción: Administración de los registros de productos.		
Observación: Se debe permitir la lectura, creación, modificación y eliminación de los registros.		

Tabla 2.15. Historia de usuario: Gestión de las categorías de productos

Código: HUSM02	**Fecha**: 2018-01-10	**Autor:** Daniel Hwang
Programador: Programador 2		**Estimación:** 10 horas
Prioridad: 2		
Descripción: Utilización de categorías de productos para su clasificación.		
Observación: Se debe permitir la lectura, creación, modificación y eliminación de los registros.		

Tabla 2.16. Historia de usuario: Gestión del inventario de productos

Código: HUSM02	**Fecha**: 2018-01-11	**Autor:** Daniel Tenemaza
Programador: Programador 1		**Estimación:** 10 horas
Prioridad: 2		
Descripción: Disponer de un inventario de los productos del establecimiento.		

Tabla 2.17. Historia de usuario: Reportes de ventas

Código: HUSM02	**Fecha**: 2018-01-11	**Autor:** Daniel Tenemaza
Programador: Programador 1		**Estimación:** 10 horas
Prioridad: 1		
Descripción: Posibilitar la generación de reportes de ventas del supermercado.		

Tabla 2.18. Historia de usuario: Facturas de los clientes

Código: HUSM02	**Fecha**: 2018-01-12	**Autor:** Daniel Hwang
Programador: Programador 2		**Estimación:** 10 horas
Prioridad: 1		
Descripción: Posibilitar la generación de facturas de los clientes.		

2.2.4 Tablero Kanban

En la sección anterior se obtuvieron las historias de usuario con las descripciones breves y simples de los requerimientos del sistema, mediante el tablero Kanban se representan estas tareas en un flujo de trabajo. Para el desarrollo del proyecto se utilizó un tablero Kanban virtual proporcionado por Kanbantool [20]. Este tablero permite al equipo de desarrolladores visualizar y optimizar el flujo de trabajo, puesto que:

- Proporciona un medio de fácil acceso a todos los involucrados
- Facilita la comunicación entre los miembros del equipo
- Ofrece un medio para visualizar el progreso de las tareas
- Evita los cuellos de botella tan pronto como ocurren

Además, cada tarea dentro del tablero está representada por una tarjeta, de esta manera se garantiza un seguimiento y trabajo adecuados. La Figura 2.17 ilustra el tablero Kanban implementado para el desarrollo del sistema prototipo, donde se pueden identificar tres

etapas en el flujo de trabajo, por hacer (amarillo), en proceso (gris) y hecho (verde). De la misma manera, se distinguen tres categorías de actividades correspondientes a la base de datos, la aplicación Android y la aplicación web respectivamente.

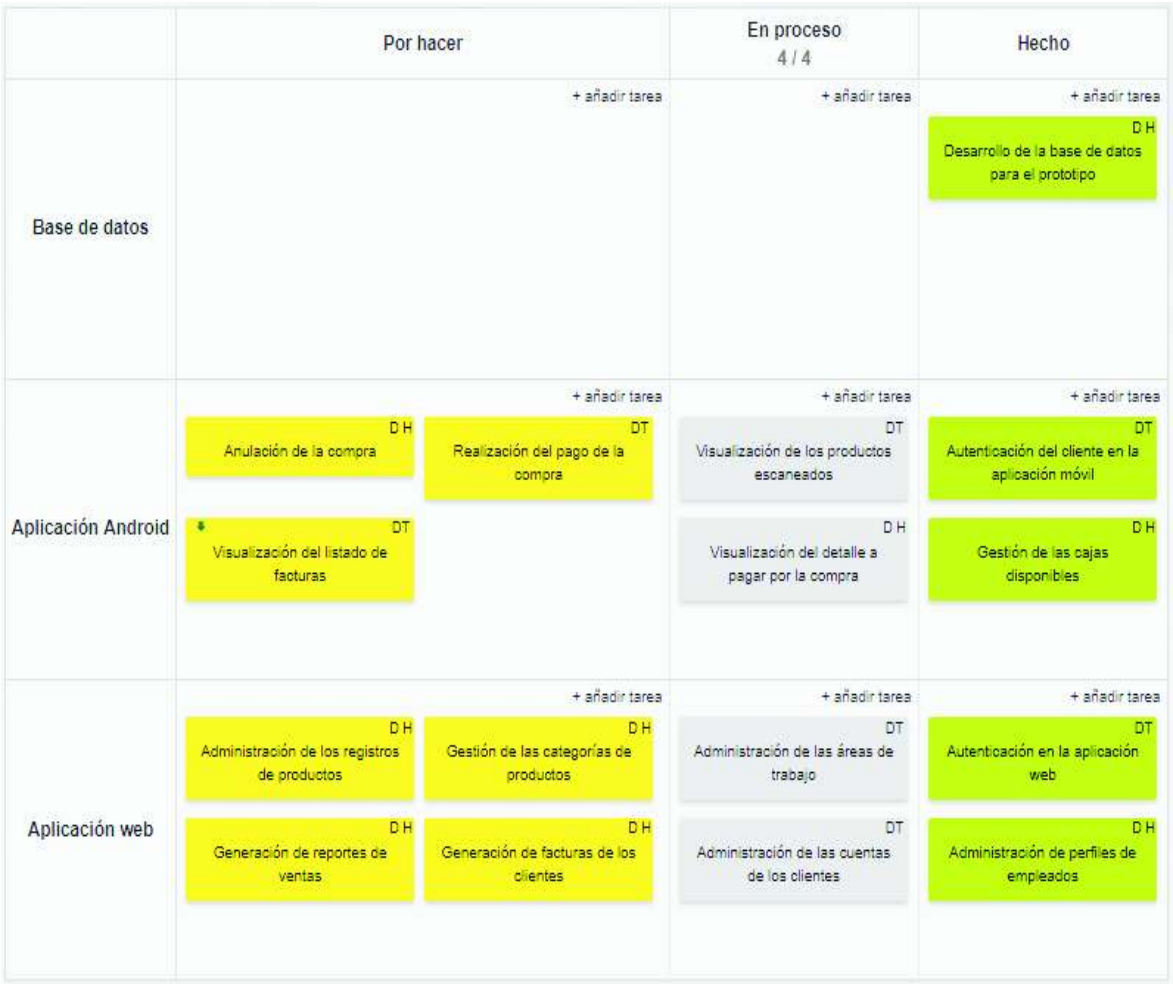


Figura 2.17. Tablero Kanban del sistema prototipo

2.2.5 Diagrama Entidad-Relación

El diagrama entidad-relación (ER) muestra las relaciones del conjunto de entidades almacenadas en la base de datos. En este contexto, una entidad es simplemente un componente de datos. La Figura 2.18 ilustra la estructura lógica de la base de datos.

Para la administración del personal del establecimiento se definieron 5 entidades fundamentales: `Empleado`, `Cargo`, `Encargado`, `Ciudad` y `Usuario`. La entidad `Empleado` se asocia con la entidad `Cargo` de tal manera que un empleado puede tener un único cargo. Además, presenta una relación con las entidades `Encargado`, `Ciudad` y `Usuario`, donde el empleado puede pertenecer a una sola ciudad, estar encargado de un área de trabajo y tener un único usuario para utilizar el sistema de administración web.

A fin de poder gestionar los productos del supermercado se definieron 5 entidades principales: `Producto`, `Categoría`, `Tipo`, `IngresoProducto` e `Inventario`. La entidad `Producto` se relaciona con la entidad `Categoría` de tal manera que un producto puede pertenecer a una sola categoría.

De manera análoga, se presenta su relación para con las entidades `Tipo` e `Inventario`, donde el producto puede tener estar relacionado con un único tipo de artículo y estar asociado a un solo inventario. En lo relativo a la entidad `IngresoProducto` se establece una cardinalidad de uno a muchos, ya que un producto podrá tener múltiples ingresos en diferentes instancias de tiempo.

Con el propósito de gestionar las facturas de compras de los clientes del supermercado se definieron 4 entidades fundamentales: `Factura`, `Cliente`, `Caja` y `ProductosVendidos`. La entidad `Factura` se asocia con la entidad `Cliente` mediante una cardinalidad de muchos a uno, debido a que varias facturas pueden pertenecer a un mismo cliente.

De igual manera, se presenta la relación para con la entidad `Caja`, donde varias facturas pueden ser generadas en una misma caja. Con respecto a la entidad `ProductosVendidos`, la entidad `Factura` establece una cardinalidad de uno a muchos, puesto que una factura puede contener varios productos vendidos.

Por último, la entidad `Empleado` se asocia con la entidad `IngresoProducto` a través de una carnalidad de uno a muchos, dado que un empleado puede realizar múltiples ingresos. Además, la entidad `Producto` se asocia con la entidad `ProductosVendidos` estableciendo una cardinalidad de uno a muchos, puesto que un mismo tipo de producto puede ser vendido a varios clientes.

2.2.6 Diagrama Relacional

Una vez definido el modelo conceptual de Entidad-Relación, es necesario traducirlo a un modelo lógico particular del gestor de base de datos. Para ello, se utilizó el gestor de base de datos Microsoft SQL Server. Entonces, con base en las especificaciones del diagrama ER, se realiza un *script* para la generación de las tablas, los atributos y las relaciones de cada elemento dentro de la base de datos. La Figura 2.19 muestra el diagrama relacional obtenido a través del gestor de base de datos SQL Server.

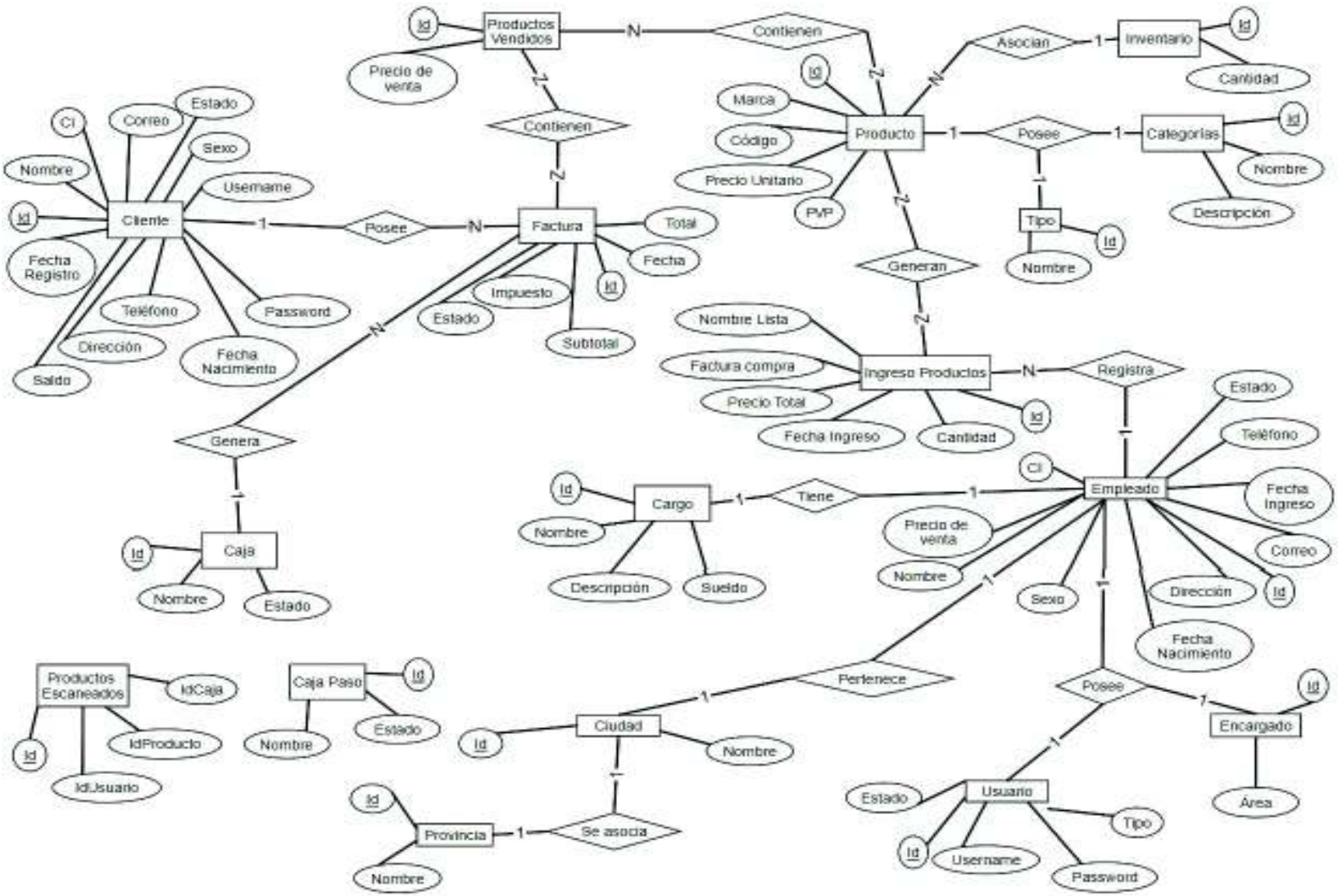


Figura 2.18. Diagrama Entidad-Relación de la base de datos

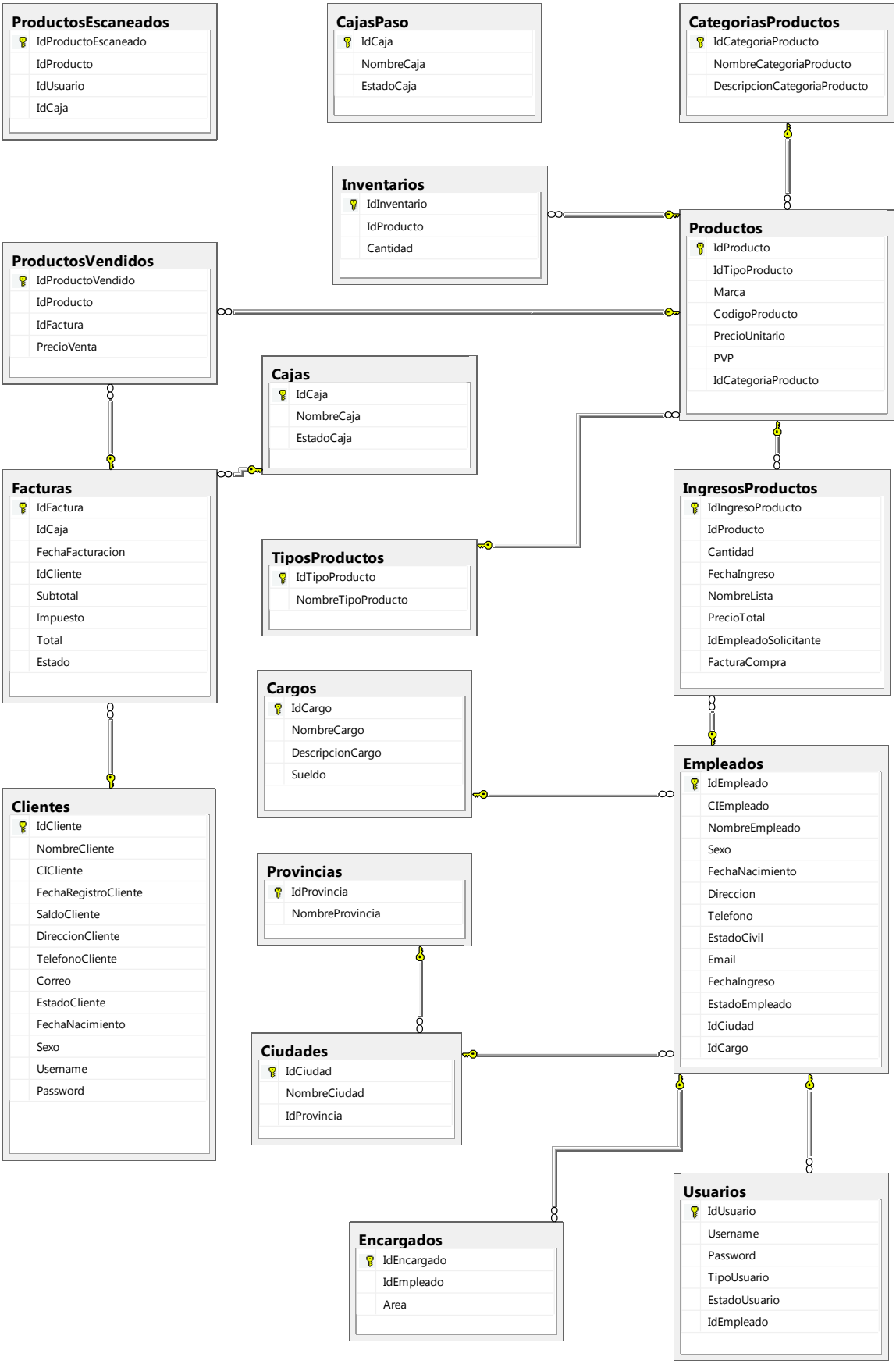


Figura 2.19. Diagrama relacional de la base de datos

Adicionalmente, la Tabla 2.19 presenta el detalle de las tablas definidas para el sistema prototipo.

Tabla 2.19. Tablas utilizadas en la base de datos

Tabla	Descripción
Provincias	Almacena el nombre de las provincias del país.
Ciudades	Almacena el nombre de las ciudades de acuerdo a la provincia a la que pertenecen.
Empleados	Almacena la información personal de los empleados del establecimiento.
Cargos	Almacena el nombre y la descripción de los cargos para los empleados.
Encargados	Almacena el identificador del empleado y el área bajo su supervisión.
Usuarios	Almacena el identificador del empleado y sus credenciales para la utilización del sistema web.
Cajas	Almacena el identificador de la caja con su respectivo estado.
Productos	Almacena la información del producto junto a un identificador único perteneciente a la etiqueta RFID.
CategoriasProductos	Almacena los nombres de las categorías de los productos ofertados.
TiposProductos	Almacena el nombre del tipo de producto ofertado.
IngresosProductos	Almacena la información relativa al ingreso de nueva mercadería al establecimiento.
Inventarios	Almacena el identificador del producto junto a la cantidad disponible correspondiente.
ProductosVendidos	Almacena el identificador del producto junto a su precio de venta.
Facturas	Almacena la información relativa a la facturación como: el identificador del cliente y el total a pagar.
Clientes	Almacena los datos personales del cliente. Así como, sus credenciales para la utilización de la aplicación móvil.
ProductosEscaneados	Almacena el identificador del producto, del usuario y de la caja utilizada por cliente.

Cabe destacar que para identificar a un usuario activo o inactivo dentro del sistema prototipo se utiliza una variable de control. De esta manera, para la gestión de los registros de las tablas `Empleados`, `Clientes` y `Usuarios` se utilizan los campos `EstadoEmpleado`, `EstadoCliente` y `EstadoUsuario`, respectivamente. Posteriormente, la Tabla 2.20 indica los posibles estados para los registros de las tablas nombradas.

Tabla 2.20. Estados de un empleado, cliente o usuario

Estado	Descripción
0	Empleado, cliente o usuario inactivo
1	Empleado, cliente o usuario activo

De manera similar, para denotar si una caja electrónica está disponible u ocupada se utiliza el campo `EstadoCaja` dentro de la tabla `Cajas`. En la Tabla 2.21 se describen sus posibles estados.

Tabla 2.21. Estados de una caja

Estado	Descripción
0	Caja libre o disponible
1	Caja ocupada o en uso

2.2.7 Diagramas de Clases

Un diagrama de clases se define como una notación gráfica utilizada para construir y visualizar sistemas orientados a objetos. Además, es un tipo de diagrama de estructura estática que identifica las clases, los atributos, los métodos y las relaciones entre los objetos.

a. Aplicación móvil

La Figura 2.20 presenta el diagrama de clases correspondiente a la aplicación móvil. Dicho diagrama expone los métodos y las variables utilizados en las diferentes clases. Ciertamente, todas las clases de la aplicación manejan variables y métodos similares.

Por lo tanto, se describe a manera de ejemplo la clase `Productos`. La Figura 2.21 presenta los métodos y variables de la misma.

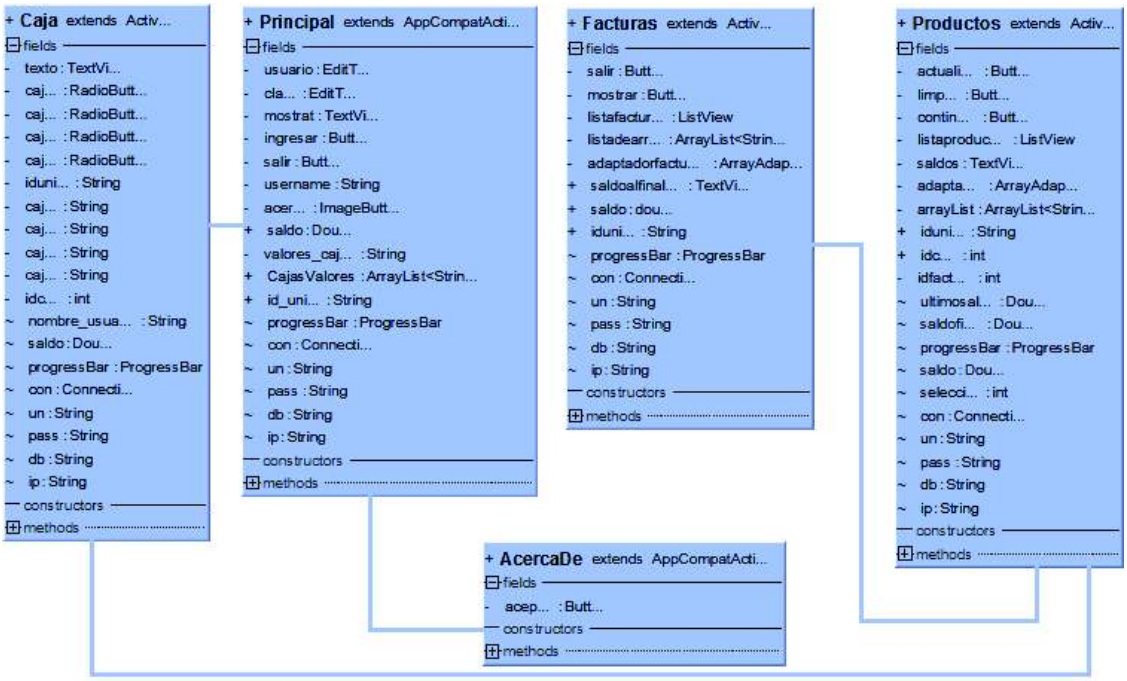


Figura 2.20. Diagrama de clases de la aplicación móvil

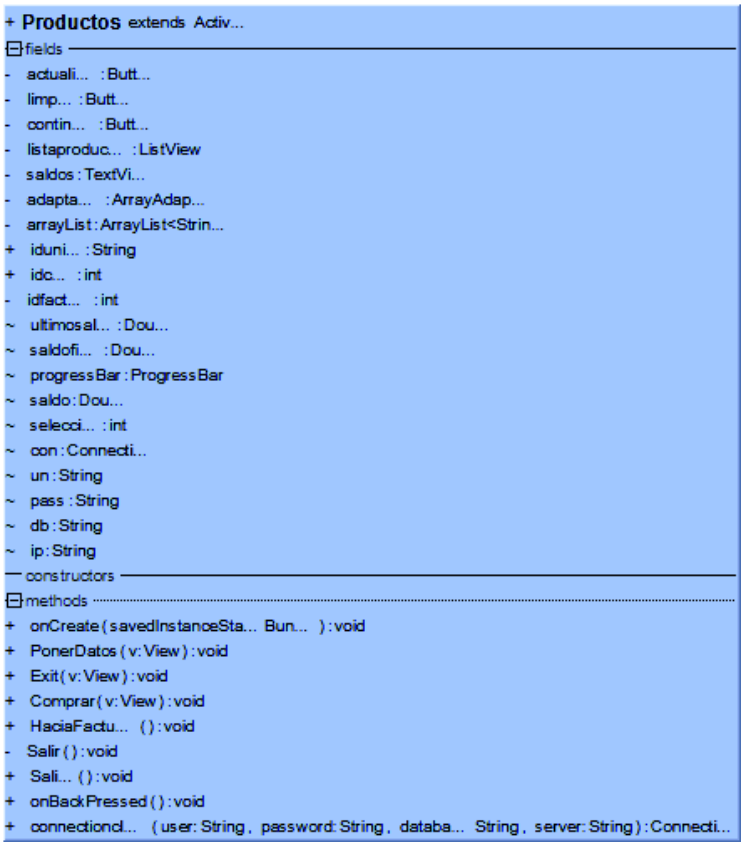


Figura 2.21. Diagrama de la clase `Productos`

Adicionalmente, la Tabla 2.22 presenta una descripción de las variables utilizadas en la clase `Productos`. Asimismo, la Tabla 2.23 ilustra los métodos de la misma.

Tabla 2.22. Variables de la clase `Productos`

Variable	Descripción
`actualizar`	Define el control `Button` que actualiza la lista de los productos que son escaneados
`continuar`	Define el control `Button` que permite efectuar la compra
`limpiar`	Define el control `Button` que permite borrar la lista de productos que han sido escaneados
`listaproducto`	Define el control `ListView` que permite enlistar los productos que serán escaneados
`saldos`	Define el control `TextView` que permite mostrar en pantalla el saldo disponible del cliente
`adaptador`	Define un `ArrayListAdapter`, el cual se activa cuando se detecta un cambio en los datos de un `ArrayList`
`arraylist`	Define una lista de arreglos que permiten almacenar los datos que son extraídos de la consulta a la base de datos
`idunico`	Variable del tipo `String` que permite almacenar el identificador único del usuario
`idcaja`	Variable del tipo `Int` que permite almacenar el identificador de la caja en la cual se está efectuando la compra.
`idfactura`	Variable del tipo `Int` que permite almacenar número de factura una vez que se ha efectuado la compra.
`ultimosaldo`	Variable del tipo `Double` que se utiliza para almacenar el valor del impuesto de la compra
`saldofinal`	Variable del tipo `Double` que se utiliza para almacenar el saldo al finalizar la compra
`progressBar`	Define el control `ProgressBar` que permite determinar la ejecución de una tarea en segundo plano
`saldo`	Variable del tipo `Double` que contiene la información del saldo inicial del cliente
`seleccion`	Variable del tipo `Int` que se utiliza para definir la acción a ejecutarse al pulsar un botón
`con`	Variable utilizada para enviar sentencias SQL a la base de datos
`un`	Variable del tipo `String` que almacena el nombre de usuario de la cuenta
`pass`	Variable de tipo `String` que almacena la clave del usuario
`db`	Variable del tipo `String` que contiene el nombre del servidor de base de datos
`ip`	Variable del tipo `String` que almacena la dirección IP del servidor de base de datos

Tabla 2.23. Métodos de la clase `Productos`

Método	Definición
`onCreate`	Permite la creación de la actividad, asociar el `layout` e inicializar las variables
`PonerDatos`	Permite colocar los datos de los productos que pasan por el lector dentro de la lista que se visualiza en la interfaz del usuario
`Exit`	Permite llamar al método `Salir` para terminar la ejecución de la aplicación
`Comprar`	Permite realizar la compra y registrarla en la base de datos
`HaciaFactura`	Permite inicializar y cargar los datos que serán enviados hacia la nueva actividad llamada `Factura`
`Salir`	Permite terminar los procesos que se encuentran ejecutándose en la aplicación
`onBackPressed`	Permite controlar las acciones que se ejecutarán cuando se presione el botón de retroceso
`connectionClass`	Permite establecer los parámetros necesarios para realizar la conexión con la base de datos a través del uso de la librería JDBC

b. Aplicación web

La Figura 2.22 ilustra el diagrama de clases de la aplicación web de los modelos: `Empleado`, `Cargo`, `Encargado`, `Usuario`, `Ciudad`, `Provincia`, `Producto`, `Tipo Producto`, `Categorías Producto`, `Inventario`, `Factura`, `Cliente`, `Caja` y `ProductosVendidos`. Las propiedades de cada una de estas clases se obtienen a través de un mapeo de los atributos o columnas de las tablas correspondientes en la base de datos. En consecuencia, cada instancia de dichas clases representa una fila o tupla en las tablas.

Adicionalmente, la Figura 2.23 muestra el diagrama de las clases: `Wfrm_CategoriasProductos`, `Wfrm_Clientes`, `Wfrm_Empleados`, `Wfrm_Encargados`, `Wfrm_Facturas`, `Wfrm_Inicio`, `Wfrm_Inventario`, `Wfrm_Login`, `Wfrm_Productos` y `Wfrm_Reporte_Ventas`. Es preciso señalar que las clases nombradas corresponden a los formularios para la gestión de los elementos del supermercado (e.g. productos). Por lo tanto, los campos y métodos utilizados en los mismos son similares, por lo que se describe a manera de ejemplo la clase `Wfrm_Encargados` (Tabla 2.24, Tabla 2.25).

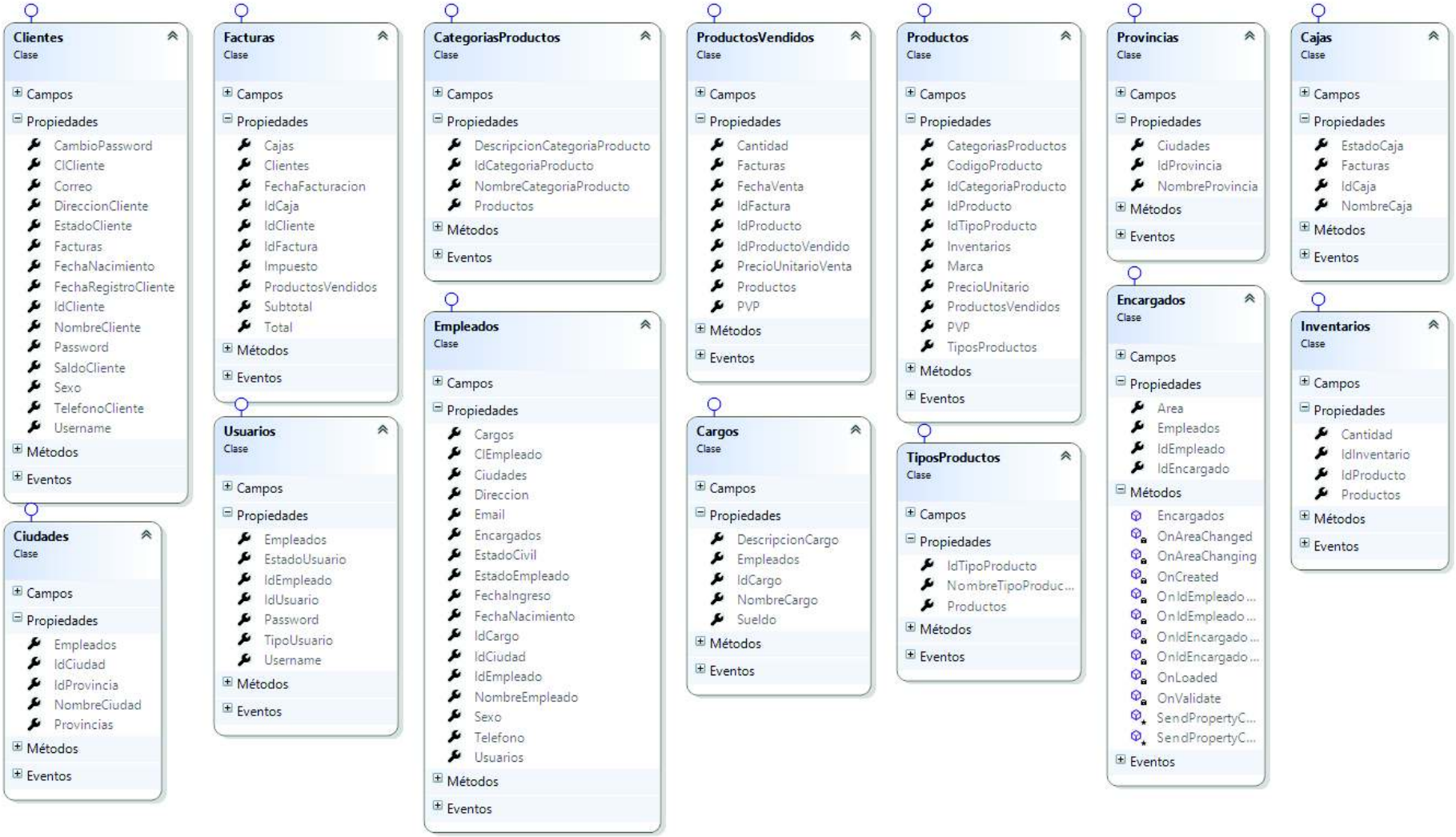


Figura 2.22. Diagrama de clases de la aplicación web

Figura 2.23. Diagrama de clases de los formularios de la aplicación web

Tabla 2.24. Campos de la clase `Wfrm_Encargados`

Campos	Descripción
`btnAgregarEncargado`	Define el control `Button` que permite agregar un perfil de encargado
`btnLimpiarEncargado`	Define el control `Button` que permite limpiar los datos del perfil de encargado seleccionado
`btnNuevoEncargado`	Define el control `Button` que permite abrir el panel de información para el ingreso de un perfil de encargado
`db`	Instancia de clase `Context` para el manejo de la base de datos
`ddlNombreEmpleados`	Define el control `DropDownList` que permite mostrar el listado de empleados existentes
`pnlDatosEncargados`	Define un `Panel` con la información del perfil de encargado
`rptEncargados`	Define el control `Repeater` para cargar información en la tabla de encargados
`txtAreaEncargado`	Define el control `TextBox` que obtiene o establece el nombre del área bajo la supervisión del encargado
`txtIdEncargado`	Define el control `TextBox` que obtiene el identificador único del encargado

2.2.8 Bocetos de las interfaces de usuario de la aplicación móvil

Para el diseño de la aplicación móvil se requiere una aproximación de las interfaces gráficas que dispondrá el sistema. Para ello, se realizaron bocetos de cada uno de los módulos.

a. Módulo de autenticación

Esta vista dispone de un botón que permite realizar el envío de la información del nombre de usuario y contraseña del cliente. Una vez validados los datos de la cuenta, la aplicación móvil debe estar en capacidad de indicar al usuario cuál fue la respuesta por parte del servidor (afirmativa o negativa). Para ello, se agregan ventanas de notificaciones (Figura 2.24). Además, esta vista permite el ingreso de una nueva contraseña cuando el usuario se ha autenticado al sistema por primera vez.

Adicionalmente, esta vista cuenta con un `ImageButton` el cual permite acceder a una ventana en la cual se muestra la información de los desarrolladores.

Tabla 2.25. Métodos de la clase `Wfrm_Encargados`

Método	Definición
`ActualizarEncargado`	Permite la actualización de la información del encargado dado su identificador único
`AgregarEncargado`	Permite agregar un perfil de encargado nuevo
`btnAgregarEncargado_Click`	Permite la llamada a los métodos `AgregarEncargado` y `ActualizarEncargado`
`btnEditarEncargado_Click`	Permite la llamada al método `CargarEncargadoSeleccionado`
`btnEliminarEncargado_Click`	Permite la llamada al método `EliminarEncargado`
`btnNuevoEncargado_Click`	Permite la visualización del panel de datos para el ingreso de un nuevo perfil de encargado
`CargarEmpleados`	Carga los nombres de empleados en el `ddlNombreEmpleados`
`CargarEncargadoSeleccionado`	Permite visualizar la información del encargado seleccionado
`CargarEncargadosTabla`	Carga los datos de los perfiles de encargados existentes
`ddlNombreEmpleados_ SelectedIndexChanged`	Permite guardar el identificador del encargado al cambiar el índice del `ddlNombreEmpleados`
`EliminarEncargado`	Permite eliminar un perfil de encargado
`LimpiarCampos`	Limpia los campos de información del encargado
`ObtenerIdEmpleado`	Obtiene el identificador del empleado seleccionado del `ddlNombreEmpleados`
`Page_Load`	Permite cargar el contenido de la página por primera vez

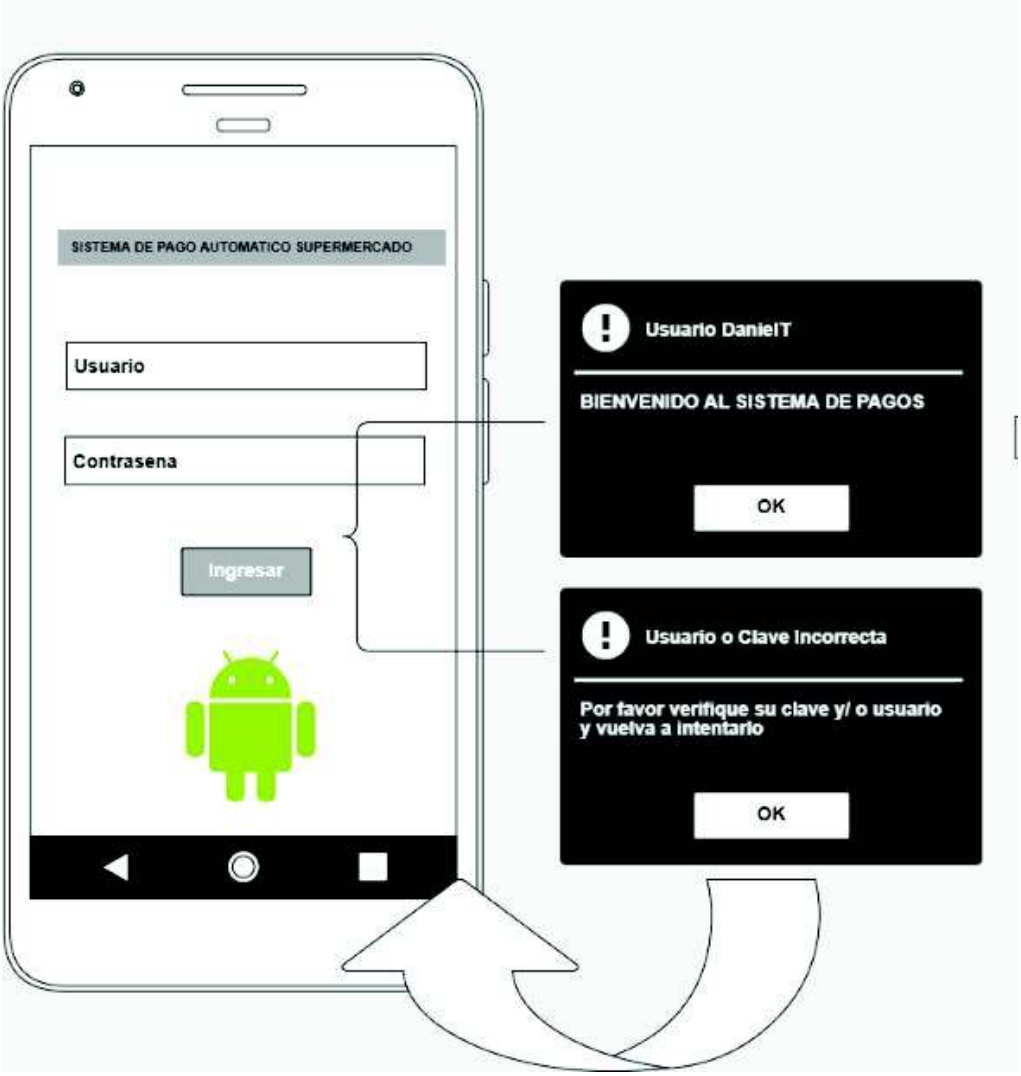


Figura 2.24. Vista del módulo de autenticación de la aplicación móvil

b. Módulo de selección de caja

Esta vista está conformada por dos `RadioButtons`, los cuales permiten realizar la selección de la caja deseada (Figura 2.25). En caso de seleccionar una caja que se encuentra en uso, se notifica mediante un cuadro de diálogo que dicha caja no se encuentra disponible al momento.

c. Módulo de pago

El objetivo principal de esta vista es exponer la información de los productos escaneados en forma de lista, con los detalles de cada producto, tales como: precio, marca, etc. Además, la vista consta de una etiqueta que muestra el saldo dispone del cliente, un botón que permite efectivizar la compra y un botón para actualizar la lista de productos leídos (Figura 2.26).

Figura 2.25. Vista del módulo de selección de caja de la aplicación móvil

Figura 2.26. Vista del módulo de selección de pago de la aplicación móvil

Finalmente, cuando se presionar el botón de compra se muestra una ventana de confirmación. Una vez confirmado el pago, se pueden mostrar dos ventanas de notificación. La primera indicando un pago exitoso y la segunda indicando que el saldo disponible en la cuenta es insuficiente.

2.3 Implementación del sistema

2.3.1 Desarrollo de la base de datos

Para el almacenamiento de la información del prototipo se implementa una base de datos SQL Server mediante el IDE[24] SQL Server Management Studio. Los elementos a desarrollarse son: las tablas, los procedimientos almacenados y los disparadores o *triggers*.

a. Tablas

En un entorno SQL, las tablas son la unidad básica de administración de información. El Código 2.1 ilustra la sintaxis necesaria para la creación de una tabla. En la línea 15, la sentencia `CREATE` seguida del nombre "Provincias" permite la creación de la tabla `Provincias`. Luego, en las líneas 17 y 18 se especifican las columnas de la tabla con sus respectivos tipos de dato. En el Anexo II se adjunta el *script* con la implementación de todas las tablas de la base de datos.

```
14 --Provincias
15 CREATE TABLE Provincias
16 (
17    [IdProvincia] [int] PRIMARY KEY IDENTITY(1,1) NOT NULL,
18    [NombreProvincia] [varchar](50)
19    )
20 GO
```

Código 2.1. Creación de la tabla `Provincias`

b. Procedimientos almacenados

Los procedimientos almacenados proporcionan la capacidad de realizar parte del procesamiento de las aplicaciones del sistema prototipo dentro de la base de datos. A continuación, se presenta a manera de ejemplo el procedimiento almacenado `Sp_Ventas_Cliente_Factura` (Código 2.2). Este procedimiento almacenado realiza la consulta del detalle de una compra específica del cliente dado su identificador y número de

[24] ***Integrated Development Environment*** **(IDE):** es un paquete de *software* que consolida las herramientas básicas que los desarrolladores necesitan para escribir y probar *software*.

factura. Luego, el resultado de la consulta se utiliza para la generación de un reporte de factura.

```
294CREATE PROCEDURE Sp_Ventas_Cliente_Factura
295@Id_Cliente int,
296@Id_Factura int
297AS
298SELECT tp.NombreTipoProducto, p.Marca, p.IdProducto,
pv.PrecioVenta, c.IdCliente, c.NombreCliente, ca.NombreCaja,
f.IdFactura, f.FechaFacturacion, f.Subtotal, f.Impuesto, f.Total FROM
Productos p
299INNER JOIN TiposProductos tp on p.IdTipoProducto=tp.IdTipoProducto
300INNER JOIN ProductosVendidos pv on pv.IdProducto = p.IdProducto
301INNER JOIN Facturas f on f.IdFactura = pv.IdFactura
302INNER JOIN Clientes c on c.IdCliente = f.IdCliente
303INNER JOIN Cajas ca on ca.IdCaja = f.IdCaja
304WHERE f.IdCliente = @Id_Cliente
305AND f.IdFactura =@Id_Factura
306GO
```

Código 2.2. Creación del procedimiento almacenado `Sp_Ventas_Cliente_Factura`

El procedimiento almacenado expuesto en esta sección permite tener una noción global de la manera en que los mismos son implementados a lo largo del desarrollo del sistema. La implementación individual de todos los procedimientos almacenados se adjunta en el Anexo II. Adicionalmente, la Tabla 2.26 presenta la colección de procedimientos almacenados y su funcionalidad en el desarrollo del sistema.

Tabla 2.26. Procedimientos almacenados del sistema

Procedimiento almacenado	Descripción
`Sp_AutenticarUsuario`	Permite la autenticación de los usuarios de la aplicación web mediante la verificación del nombre de usuario y contraseña
`Sp_Ventas_General_Total`	Realiza la consulta del total de ventas del supermercado
`Sp_Ventas_General_Fechas`	Realiza la consulta del total de ventas del supermercado en un intervalo de tiempo definido por una fecha inicial y una final
`Sp_Ventas_Cliente_Total`	Realiza la consulta del total de compras realizadas por un cliente
`Sp_Ventas_Cliente_Fechas`	Realiza la consulta del total de compras realizadas por un cliente en un intervalo de tiempo definido por una fecha inicial y una final

c. Triggers

Un *trigger* o disparador es un conjunto particular de código cuya activación es causada por las modificaciones en el contenido de la base de datos. Para el desarrollo del proyecto se crearon dos *triggers* con los cuales se controla la lectura del interrogador RFID de la caja.

En primer lugar, se define el *trigger* `trgLectorON` (Código 2.3) para ejecutar la aplicación de la caja que realiza la lectura de productos. Este disparador está asociado con la tabla `Cajas` de la base de datos (línea 371).

En el momento en que se modifican los datos de la tabla mencionada mediante una instrucción `UPDATE` (línea 372), se desencadena el disparador, ejecutando las instrucciones SQL que componen el cuerpo del mismo, donde se especifica la ejecución de la aplicación de la caja que realiza la lectura de productos (línea 376).

En segundo lugar, se define el *trigger* `trgLectorOFF`. La definición de este disparador es análoga al descrito anteriormente, sin embargo, las instrucciones SQL de este *trigger* ordenan la terminación del proceso que está ejecutando la aplicación de la caja para detener la lectura de productos.

```
371 CREATE TRIGGER trgLectorON ON dbo.Cajas
372 AFTER UPDATE
373 AS
374 BEGIN
375 DECLARE @CMDSQL VARCHAR(1000);
376 SET @CMDSQL = 'cmd.exe /C
"C:\Users\danie\Desktop\UHFReader18demomain\obj\Debug\UHFReader18demom
ain.exe"'
377 Exec xp_cmdshell @CMDSQL,no_output
378 END
379 GO
```

Código 2.3. Creación del *trigger* `trgLectorON`

El disparador o *trigger* expuesto en esta sección permite tener una noción global de la manera en que los mismos son implementados a lo largo del desarrollo del sistema. Cabe destacar que, la implementación individual de cada uno de los *triggers* se adjunta en el Anexo II.

Finalmente, la Tabla 2.27 presenta la colección de *triggers* utilizados y su funcionalidad en el desarrollo del sistema.

Tabla 2.27. *Triggers* del sistema

Trigger	Descripción
`trgLectorON`	Permite la activación del lector RFID cuando se establece en “1” la variable estado de la caja
`trgLectorOFF`	Permite la desactivación del lector RFID cuando se establece en “0” la variable estado de la caja

2.3.2 Desarrollo de la aplicación para la caja electrónica

La funcionalidad principal de aplicación de la caja electrónica es la lectura de los identificadores de las etiquetas o *tags* RFID y el almacenamiento de los mismos en la base de datos. Para cumplir con ello, se utiliza el código fuente de demostración proporcionado por el fabricante del lector RFID, añadiendo las adaptaciones pertinentes para el funcionamiento adecuado del presente proyecto.

Conviene señalar que, tanto el código de la aplicación del lector como el manual de usuario se adjuntan en el Anexo III.

El código fuente provisto por el fabricante se desarrolló en el lenguaje de programación C#, utilizando formularios Windows para la interfaz gráfica de usuario. A continuación, se describen las adaptaciones puntuales realizadas al *software.*

a. Conexión a la base de datos

Para la comunicación de la aplicación de la caja con la base de datos, se utiliza el componente LINQ to SQL proporcionado por el .NET *Framework.* Este lenguaje de consulta de Microsoft está completamente integrado y ofrece fácil acceso a los datos, i.e. objetos en memoria, bases de datos, documentos XML, etc. Por su parte, LINQ proporciona una infraestructura de tiempo de ejecución para administrar datos relacionales como objetos.

En primer lugar, se requiere añadir la clase LINQ to SQL al proyecto como se muestra en la Figura 2.27, posteriormente, se realiza la conexión a la base de datos y simplemente se arrastran las tablas deseadas (`Productos` y `ProductosEscaneados`) al diseñador relacional de objetos (*object relational designer*). Entonces, automáticamente se crean las clases de datos como se puede apreciar en la Figura 2.28.

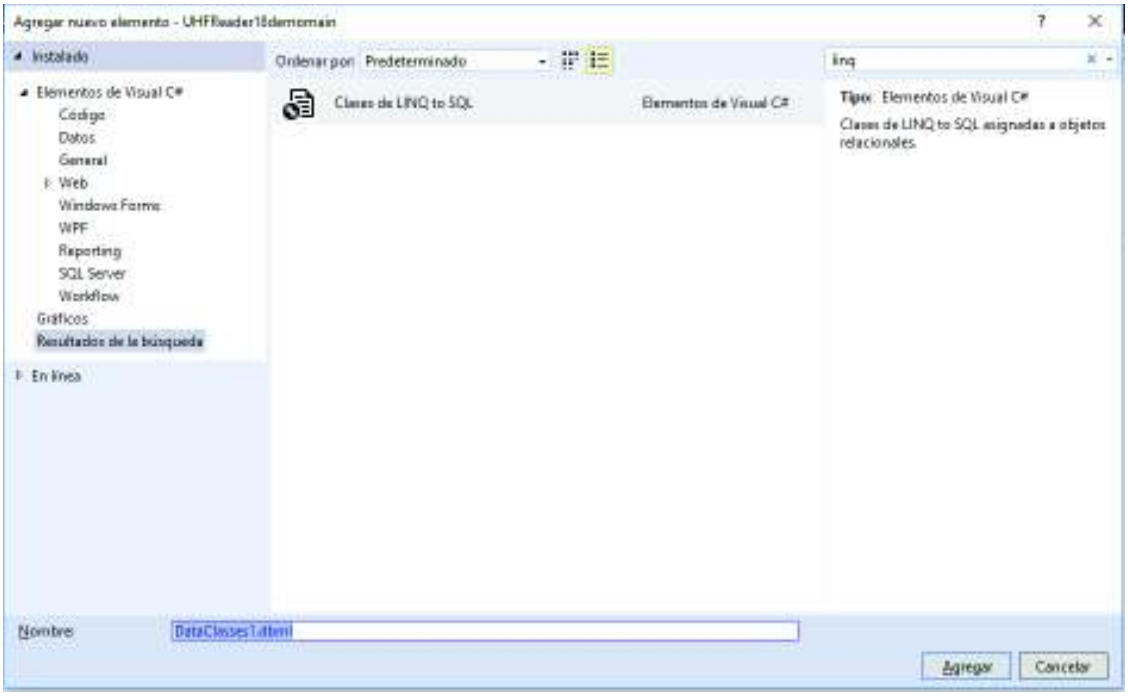


Figura 2.27. Creación de la clase LINQ to SQL

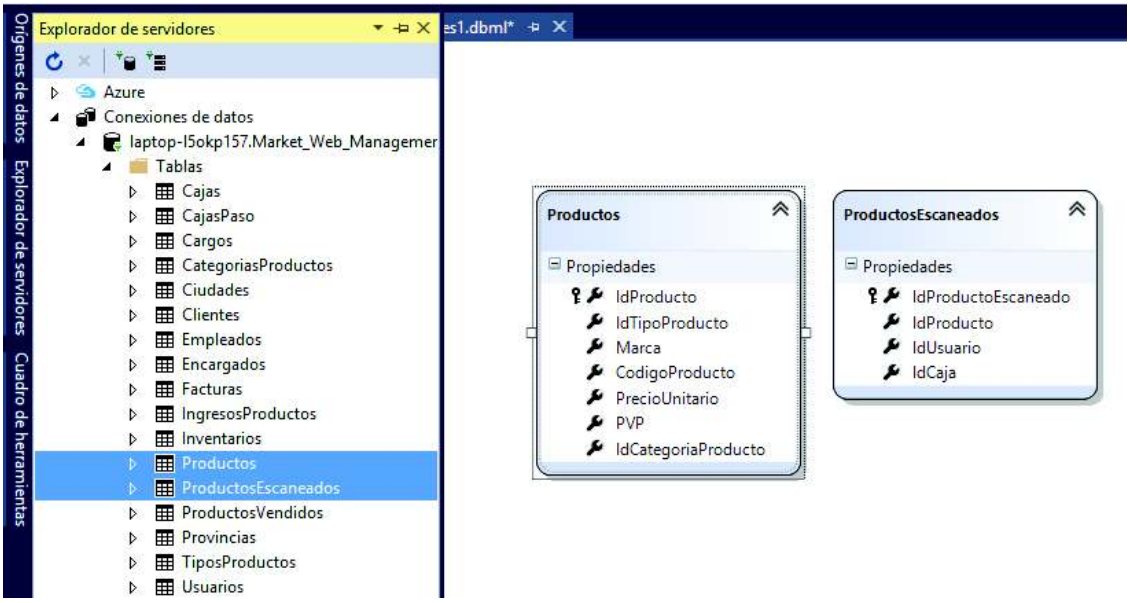


Figura 2.28. Clases de datos `Productos` y `ProductosEscaneados`

b. Almacenamiento de datos

Una vez que la aplicación haya hecho la lectura de las etiquetas RFID es necesario almacenar sus identificadores en la tabla `ProductosEscaneados` de la base de datos. Para ello, se comparan los identificadores únicos de las etiquetas con los identificadores almacenados en la tabla `Productos` para identificar cada artículo. Finalmente, se guardan los identificadores de los artículos en la tabla `ProductosEscaneados`. El Código 2.4 muestra la codificación del proceso mencionado.

```
/*modificación para permitir el envío de datos hacia la base desde el
lector */
//se crea una instancia de la tabla productos escaneados
ProductosEscaneados inserta = new ProductosEscaneados();
//se realiza una selección del Id de los productos en la tabla
productos y almaceno en variable ver
//ID corresponde al código EPC de cada etiqueta
var ver = from a in Db.Productos where a.CodigoProducto == sEPC select
a.IdProducto;
//Se selecciona todos los elementos que existen en la tabla productos
seleccionados para poder obtener el Id de la caja, y el Id de usuario
que esta realizando la compra
var seleccion = from f in Db.ProductosEscaneados select f;
//se eliminan los valores que se encuentren como null en la primera
columna dentro de la tabla productos escaneados
var eliminar = from a in Db.ProductosEscaneados where
a.IdProducto.Equals(0) select a;
//lazo para eliminar todos los elementos que contengan null
foreach (var v in eliminar)
{
Db.ProductosEscaneados.DeleteOnSubmit(v);
}
//se crea un lazo para recorrer toda la información almacenada en la
variable ver la cual contiene la información de la consulta realizada
a la tabla productos
foreach (var b in ver)
{
//se ingresa en Idproducto, todo lo que se encuentra en la variable b
inserta.IdProducto = b;
//lazo para recorrer la variable selección
foreach (var f in seleccion)
{
//se obtien el usuario e id de caja, y se almacena en variable locales
inserta.IdUsuario = f.IdUsuario;
inserta.IdCaja = f.IdCaja;
}
//se imprime en consola la información que se tiene en la variable b
System.Console.WriteLine(b);
}
//se ingresa en productos escaneados, la información de usuario e id
de caja para cada etiqueta que se escanee
Db.ProductosEscaneados.InsertOnSubmit(inserta);
//Se almacenan los cambios en la base de datos
Db.SubmitChanges();
```

Código 2.4. Almacenamiento de los identificadores de los productos

2.3.3 Desarrollo de la aplicación móvil

La aplicación móvil tiene como objetivo permitir la comunicación entre el cliente y la información almacenada en la base de datos, permitiéndole realizar:

- La autenticación al sistema
- La visualización de las cajas disponibles
- La selección de una caja libre para su utilización

- La visualización de los productos escaneados en caja
- Efectivizar el pago.

Para el desarrollo de la aplicación se utilizó el IDE Android Studio. Adicionalmente, en el Anexo IV se encuentra todo el código de la aplicación debidamente comentado para su compresión.

a. Módulos del sistema

a.1. Módulo de autenticación

En primer lugar, el cliente tiene que registrarse en el punto de servicio al cliente donde se le asignará un nombre de usuario y una contraseña provisional. La autenticación se realiza a través de la verificación del nombre de usuario y la contraseña en el servidor de base de datos. Además, cuando el cliente se autentique por primera vez, el sistema le solicitará cambiar su contraseña provisional.

a.2. Módulo de selección de caja

El usuario podrá visualizar las cajas disponibles del supermercado y elegir una para su utilización. Una vez se haya realizado la selección de la caja, la aplicación se comunica con la base de datos para cambiar el estado de dicha caja de disponible a ocupado. De esta manera, se deshabilita la caja para los demás clientes. Entonces, se activa el interrogador RFID para que se realice la lectura de los productos. Adicionalmente, de existir errores en el escaneo se posibilita la realización de una nueva lectura de productos.

a.3. Módulo de pago

En este módulo, el usuario puede verificar el detalle de los productos escaneados: el precio individual de cada producto, la marca y el total a pagar. Entonces, el cliente puede proceder a realizar el pago de compra. Adicionalmente, se permite la anulación de la compra ya sea por saldo insuficiente o por voluntad propia del usuario.

b. Conexión a la base de datos

Para la conexión a la base de datos se utilizó el API JDBC (*Java DataBase Connectivity*) que proporciona acceso a datos universales desde el lenguaje de programación Java. Además, proporciona una base común sobre la cual se pueden construir herramientas e interfaces alternativas.

La funcionalidad de la librería JDBC puede ser definida en tres procesos:

- Establecimiento de la conexión con una base de datos.

- Envío de sentencias en lenguaje SQL a la base de datos.
- Procesamiento de los resultados obtenidos por parte de la base de datos.

Para el desarrollo del sistema prototipo actual se utilizó la librería JDBC versión 1.3.1.

c. Lógica de aplicación

c.1. Manifiesto de Android

El manifiesto Android constituye el fundamento o núcleo de toda aplicación, pues define todas las actividades asociadas al programa, los permisos que se concederán a la aplicación, la versión del API que se utilizará, la codificación, etc.

El Código 2.5 muestra el manifiesto Android del proyecto que determina un nivel de API 21 (i.e. Android 5.0 o Lollipop). Asimismo, se puede evidenciar las actividades que componen la aplicación, `Principal`, `Productos`, `Caja`, `Facturas` y `AcercaDe`. Por último, se detallan los permisos concedidos a la aplicación.

```
1  <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="tesis.tesis">
4
5     <application
6         android:allowBackup="true"
7         android:icon="@mipmap/ic_launcher"
8         android:label="Web Market App"
9         android:roundIcon="@mipmap/ic_launcher_round"
10        android:supportsRtl="true"
11        android:theme="@style/Theme.AppCompat.DayNight.NoActionBar">
12        <activity android:name=".AcercaDe"></activity>
13        <activity android:name=".Facturas"></activity>
14        <activity android:name=".Caja"></activity>
15        <activity android:name=".Productos"></activity>
16        <activity android:name=".Principal">
17            <intent-filter>
18                <action android:name="android.intent.action.MAIN" />
19
20                <category
android:name="android.intent.category.LAUNCHER" />
21            </intent-filter>
22        </activity>
23
24    </application>
25    <uses-permission android:name="android.permission.INTERNET"/>
26    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
27
28 </manifest>
```

Código 2.5. Manifiesto de Android de la aplicación móvil

c.2. Módulo de Autenticación

En primer lugar, para el establecimiento de la comunicación con el servidor de base de datos, se realiza la inicialización de los parámetros de conexión en el método `OnCreate` (Código 2.6).

```
43      @Override
44      //método OnCreate, que permite asociar la clase con el layout
correspondiente
45      protected void onCreate(Bundle savedInstanceState) {
46          super.onCreate(savedInstanceState);
47
48          //Se indica la dirección ip, el nombre de la base de datos
49          // el usuario y clave para conexión hacia la base de datos
remota.
50          //ip del servidor
51           ip = "192.168.1.2";
52           //nonbre de la base de datos
53          db = "/Market_Web_Management";
54           //usuario de la base
55          un = "daniel";
56           //clave de acceso a la base
57          pass = "1234";
```

Código 2.6. Parámetros para la conexión con la base de datos

Una vez que el usuario presione el botón "ingresar", se dispara el evento que efectúa la validación de los datos ingresados en cada uno de los `EditText`, los cuales contienen el identificador de usuario y la clave de acceso (Código 2.7).

```
80 //se define la referencia a la que llamara el button
81ingresar = (Button) findViewById(R.id.btnIngresar);
82{//es un escuchador de evento
83ingresar.setOnClickListener(new View.OnClickListener()
84 public void onClick(View view) {
85 ValidarLogeo validarLogeo = new ValidarLogeo();
86 //creacion de una tarea asincrónica,
87// la cual se ejecutara en segundo plano (background) para reducir
88// el uso de procesamiento en la actividad principal.
89 validarLogeo.execute("");//se ejecuta la tarea en un solo hilo
secundario.
90              }
91          });
92      }
```

Código 2.7. Evento `OnClick` del botón Ingresar

El Código 2.7 ilustra la llamada al método `ValidarLogeo`, el cual se ejecuta como una tarea asincrónica, con la finalidad de realizar el procesamiento en segundo plano. Cabe señalar que es necesario establecer 3 tipos de argumentos dentro de una tarea asincrónica para su correcto funcionamiento (Código 2.8):

- *Parámetros:* permite definir el tipo de datos utilizado para el envío de la información hacia la tarea asincrónica, por ejemplo, *string, int, bollean*, etc.
- *Progreso:* se utiliza para definir qué tipo de datos se esperan recibir por parte de la tarea asincrónica, mientras la tarea en segundo plano está en ejecución.
- *Resultado:* permite definir el tipo de datos que serán devueltos al hilo principal, una vez que se haya terminado la ejecución de la tarea en segundo plano.

```
93      //clase para validar la conexión, la cual se extiende de una
tarea asincrónica,
94      // y se indican el tipo de datos que utilizaran en la tarea
95      public class ValidarLogeo extends
AsyncTask<String,String,String>
96      {
```

Código 2.8. Fragmento de la Clase `ValidarLogeo`

Mientras la tarea en segundo plano se esté ejecutando, se visualiza un `progressBar` definido en el método `onPreExecute`, tal como se muestra en el Código 2.9.

```
99 //definición  de onPreExecute, las tareas que serán realizadas,
antes de iniciar la tarea en segundo plano
100         @Override
101         protected void onPreExecute()
102         {
103             //se inicializa la visualización del progessBar,
104             // mientas se esté ejecutando la tarea asincrónica.
105             progressBar.setVisibility(View.VISIBLE);
```

Código 2.9. Método `onPreExecute`

La validación de las credenciales del usuario, nombre de usuario y contraseña, se efectúan dentro del método `doInBackgroud`. Para lo cual, se definen dos variables: `usernam` y `paswordd`, las mismas que son enviadas como una sentencia SQL para realizar la consulta en la tabla `Clientes` de la base de datos, tal como se puede apreciar en el Código 2.10. Finalmente, el resultado de la consulta es almacenado en la variable `rs`.

De manera similar, para la verificación de las cajas que se encuentran en uso se envía un `string` con la sentencia SQL que realiza la consulta en la tabla `Cajas` de la base de datos y se almacena su resultado en la variable `rs2` (Código 2.11).

```
153 else
154 {
155 //se crea una cadena de string, con el nombre query, la cual
contendrá
156 // la consulta que se realizara a la base de datos,
157 // seleccionando de la tabla Clientes
158 //validando la información de usuario y clave
159 String query = "select * from Clientes where Username= '" +
160 usernam.toString() + "'collate Modern_Spanish_CS_AS " +
161 "and Password= '"+ passwordd.toString()
162 +"' collate Modern_Spanish_CS_AS";
163 //se establece un stament para llamar a conexión
164 Statement stmt = con.createStatement();
165 //se crea una variable rs para almacenar la información,
166 // como resultado de la ejecución del query solicitado.
167 ResultSet rs = stmt.executeQuery(query);
168 //se establece una condición en caso de que
169 // el requerimiento a la base de datos se cumpla
```

Código 2.10. Autenticación del usuario en la base de datos

```
172 //se crea un nueva consulta para la base de datos
173 String query2="select * from CajasPaso";
174 //se inicializa el statement para enviar la cadena de datos
175 Statement stame = con.createStatement();
176 //se crea una variable para almacenar la respuesta a la solicitud
realizada
177 ResultSet rs2 = stame.executeQuery(query2);
178 //se crea un lazo para recorrer todos los valores que se obtengan
como respuesta a la consulta
179  while (rs2.next())
180 {
181 //se agrega al arraylist, los valores que se encuentran en la
182 // tercera columna, de la matriz de datos que nos da como
respuesta la consulta
183      CajasValores.add(rs2.getString(3));
184 }
```

Código 2.11. Obtención del estado de las cajas

Una vez finalizada la ejecución del método `doInBackground`, todos los resultados obtenidos a través de las sentencias SQL son enviados como un parámetro hacia el método `onPostExecute`, el mismo que envía la información hacia el hilo principal de la aplicación.

Finalmente, en el Código 2.12 se muestra el método `onPostExecute` que despliega un mensaje de autenticación exitosa junto al nombre del usuario almacenado en la variable `username`.

```
107//Se define las tareas que se realizaran una vez cumplido, la
actividad en segundo plano, para ello se utiliza el método
onPostExecute.
108 Override
109 protected void onPostExecute(String r)
110 {
111 progressBar.setVisibility(View.GONE);//se establece el progressBar
como no visible una vez cumplida la tarea secundaria.
112  Toast.makeText(Principal.this, r, Toast.LENGTH_SHORT).show();
113 //Se crea un mensaje que aparecerá en la actividad principal,
114 // con la respuesta esperada de la variable Z.
115       if(isSuccess)//valido si el booleano isSuccess
116 // se cambia  a estado true, de ser así se ejecuta una condición
117    {
118 //se ejecuta la condición de mostrar un mensaje,
119 // si la tarea asincrónica se ejecuto correctamente
120       Toast.makeText(Principal.this , "Bienvenido: " +
121       ""+username , Toast.LENGTH_LONG).show();
122    }
123  }
```

Código 2.12. Método `onPostExecute`

c.3. Módulo de selección de caja

Cuando las tareas en segundo plano terminan su ejecución, tiene lugar el método `Logeado`, el mismo que permite el intercambio de la información obtenida mediante las consultas SQL (nombre de usuario, saldo, identificador único, estados de las cajas) y la actividad principal `C` que define la interfaz de usuario (Código 2.13).

```
276  //creación de un método llamado logeado
277    public void Logeado()
278    {
279       //se establece un nuevo intento, para llamar a la
visualización de la actividad Caja
280       Intent nuevo = new Intent(this, Caja.class);
281       //se envían los valores que serán utilizados en el layour
caja.
282       nuevo.putExtra("nombre", username);
283       nuevo.putExtra("saldo",saldo);
284       nuevo.putExtra("idunico",id_unico);
285       nuevo.putExtra("valores",valores_cajas);
286       nuevo.putExtra("caja1",CajasValores.get(0));
287       nuevo.putExtra("caja2",CajasValores.get(1));
288       nuevo.putExtra("caja3",CajasValores.get(2));
289       nuevo.putExtra("caja4",CajasValores.get(3));
290       //se inicia la nueva actividad pasando todos los valores
que se han definido
291       startActivity(nuevo);
292    }
```

Código 2.13. Método `Logeado`

La obtención de la información de la actividad principal se logra mediante el método `OnCreate`, el cual asocia los datos de la actividad a variables designadas (Código 2.14). Entonces, una vez obtenidos los datos referentes al estado de las cajas, se establece una condición que impida la selección de una caja en uso (Código 2.15).

```
43      @Override public void onCreate(Bundle savedInstanceState) {
44          super.onCreate(savedInstanceState);//método OnCreate, que
permite asociar la clase con el layout correspondiente
45          setContentView(R.layout.seleccioncaja);//declaración del
layour que se utilizara
46
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);//de
finicion de la orientación de la pantalla
47          Bundle extraer = getIntent().getExtras();//se extrae la
información enviada por la actividad principal
48          nombre_usuario = extraer.getString("nombre");//almaceno en
nombre_usario, el valor enviado con el nombre
49          saldo=extraer.getDouble("saldo");//se extrae la información
de saldo y se almacena en una variable
50          idunico=extraer.getString("idunico");//se extrae la
información de id y se almacena en una variable
51          caja1=extraer.getString("caja1");//se extrae la información
del estado de idcaja y se almacena en una variable
52          caja2=extraer.getString("caja2");//se extrae la información
del estado de idcaja y se almacena en una variable
53          caja3=extraer.getString("caja3");//se extrae la información
del estado de idcaja y se almacena en una variable
54          caja4=extraer.getString("caja4");//se extrae la información
del estado de idcaja y se almacena en una variable
```

Código 2.14. Método `OnCreate`

Una vez que el usuario selecciona una caja disponible, se da inicio al proceso de lectura de los productos por parte del interrogador RFID. Para ello, se realiza una actualización en el estado del registro de la caja, el cual dispara el *trigger* `trgLectorON` que activa el lector, como se puede apreciar en el Código 2.16.

Por último, se envía la información del identificador del usuario, su saldo y el identificador de la caja hacia la nueva actividad en el módulo de pago mediante el método `CajaSelect` (Código 2.17).

c.4. Módulo de pago

Este módulo se compone de 3 botones. El botón *Actualizar* permite visualizar el listado de productos a ser adquiridos, el botón *Comprar* que permite la realización del pago y el botón *Cancelar* que permite anular la compra y salir del sistema. La Figura 2.29 muestra la vista del módulo de pago.

```
108 //se establece un evento cuando el elemento cambia su estado
109       cajaa.setOnClickListener(new View.OnClickListener() {
110       @Override
111       public void onClick(View view) {
112       idcaja=1;//se establece el id de la caja con un valor de 1
113       if(!(cajaa.isDirty()))//se establece una condición cuando
114 // el radio button se encuentra marcado
115 //mensaje a ser mostrado cuando se cumple la condición anterior
117      Toast.makeText(Caja.this , "Por favor seleccione una caja" +
118      " que no este en uso" , Toast.LENGTH_LONG).show();
119       }
120    else {
121 //se llama a la tarea asincrónica para inicializar un proceso
122 // en segundo plano, y reducir la carga en el proceso de la
aplicacion.
123    ValidaciondeConexion validaciondeConexion = new
ValidaciondeConexion();
124  //se establece que la tarea asincrónica
125 // se inicie en un hilo distinto no secuencial.
126
validaciondeConexion.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR)
;
127                 }
128             }
129         });
```

Código 2.15. Condición para la selección de una caja

```
233    else
234  {
235    Statement stmt = con.createStatement();//se establece un
236 // stament para llamar a conexión
237    stmt.executeUpdate( "insert into ProductosEscaneados" +
238 " values (0," + idunico + "," + idcaja + ")");//se realiza un
insert
239   // en la tabla productos escaneados con los valores deseados
240    CajaSelect();//se llama al método CajaSelect
241   stmt.executeUpdate("update CajasPaso set" +
242 " EstadoCaja=1 where IdCaja='"+idcaja+"'");//se establece el valor
243 // de 1 en la caja seleccionada
244 stmt.executeUpdate("update Cajas set" +
245 " EstadoCaja=1 where IdCaja='"+idcaja+"'");//se establece el valor
246 // de 1 en la caja seleccionada
247  }
```

Código 2.16. Establecimiento del estado de la caja a ocupado

Al presionar el botón de actualización se realiza una consulta a la tabla `ProductosEscaneados` de la base de datos para extraer la información de los productos leídos (Código 2.18). Estos elementos son agregados a un control `listview` para la visualización del cliente. Cabe señalar que, todo este proceso tiene lugar dentro de una tarea asincrónica.

```
187     //se crea un nuevo método
188     private void CajaSelect()
189     {
190         //se declara una nueva intención para llamar a otra vista
191         Intent haciaproductos = new Intent(this,Productos.class);
192         //se establece los datos que se enviaran, con el nombre, y
la variable que contiene los datos
193         haciaproductos.putExtra("saldo",saldo);
194         //se establece los datos que se enviaran, con el nombre, y
la variable que contiene los datos
195         haciaproductos.putExtra("idunico",idunico);
196         //se establece los datos que se enviaran, con el nombre, y
la variable que contiene los datos
197         haciaproductos.putExtra("idcaja",idcaja);
198         startActivity(haciaproductos);//se inicia la nueva
actividad
199
200     }
```

Código 2.17. Método `CajaSelect`

Figura 2.29. Vista del módulo de pago

```
210  case 1://condición a ejecutarse cuando selección =1
211  continuar.setEnabled(true);//se activa el botón continuar
212  //se establece la sentencia a ser enviada a la base de datos
213  String query = "select * from ProductosEscaneados where
IdCaja="+idcaja;
214  //se envía la sentencia a la base de datos
215  Statement stmt = con.createStatement();
216  // variable para almacenar la respuesta a la base de datos
217  ResultSet rs = stmt.executeQuery(query);
218  int ac=1;//se establece variable para numerar listado.
```

Código 2.18. Consulta de los productos escaneados

Además, dentro de la misma tarea asincrónica se realiza el cálculo del valor total a cancelar por los productos, como se puede observar en el Código 2.19.

```
244 //se calcula el saldo con el iva incluido
245     double pagar=(saldofinal*0.12)+saldofinal;
246 //se agrega al arraylist el valor total a pagar
247     arrayList.add("TOTAL A PAGAR ES
DE:$"+Math.round(pagar*100d)/100d);
248 //se notifica al adaptador la información que esta en el arraylist
249     adaptador.notifyDataSetChanged();
```

Código 2.19. Cálculo del valor total a pagar

Por lo tanto, cuando el cliente valide el detalle de los productos, así como, el total a pagar puede proceder a efectivizar el pago presionando el botón *Comprar*. Cabe indicar que únicamente cuando el cliente haya presionado el botón de compra se registra la información en la base de datos (Código 2.20), realizando la validación de saldo disponible correspondiente.

c.5. Módulo de Facturas

Para la visualización de las facturas del cliente se realiza una consulta a la base de datos con el identificador único del usuario (Código 2.22).

Entonces, el resultado es desplegado en un `listview` como se puede apreciar en la Figura 2.30.

2.3.4 Desarrollo de la aplicación web

Para el desarrollo de la aplicación ASP.NET se utiliza el IDE Microsoft Visual Studio Community 2015, a través de *Web Forms* (Formularios Web). Adicionalmente, en el Anexo V se adjunta toda la codificación debidamente comentada para su comprensión. La Tabla 2.28 expone la estructura de la aplicación.

```
case 3://caso a ejecutarse cuando selección =3
//se realiza el calculo de 12% al saldo final
 double impuesto = saldofinal*0.12;
//se suma el valor de compra y el impuesto total
 double total= saldofinal+impuesto;
//se descuenta del saldo del cliente, el valor que se tiene de
compra
 ultimosaldo = Math.round ((saldo-total) * 100d)/100d;
/condicion a cumplirse en caso de que saldo de cliente sea mayor a
cero
if (ultimosaldo>=0) {
//se inicializa la variable para envío de una sentencia a la base
de datos
  Statement stmt2= con.createStatement();
//escribe el saldo final en la cuenta del usuario y permite q se
cierre el lector
int a = stmt2.executeUpdate("update Clientes set SaldoCliente=" +
""+ultimosaldo+"where IdCliente='"+idunico+"'");
//sentencia para crear las facturas en la tabla facturas
int f= stmt2.executeUpdate("insert into Facturas values
("+idcaja+"," +

"GETDATE(),"+idunico+",null,null,null,0)");
//se ingresa el valor de cero para poder filtrar la búsqueda de la
factura
   Statement stmt4= con.createStatement();
//selección el id de factura, donde estado es activo, con el
numero de caja e identificador de cliente
ResultSet resultado = stmt4.executeQuery("select IdFactura from" +
" Facturas where Estado=0 and IdCaja="+idcaja+" and
IdCliente="+idunico);
```

Código 2.20. Fragmento de código para registrar una compra del usuario

```
case 2://condición en caso de que selección contenga valor 2
//se inicializa la variable para el envío de una nueva sentencia
sql
Statement stmt1 = con.createStatement();
//escribe el saldo final en la cuenta del usuario y permite q se
cierre el lector
int nw = stmt1.executeUpdate("update Clientes set
SaldoCliente="+saldo+"where IdCliente='"+idunico+"'");
// sentencia para eliminar solo lo que corresponde a una caja
int r = stmt1.executeUpdate("delete ProductosEscaneados where
IdCaja="+idcaja);
//se establece el estado de caja en 0, en la tabla caja paso
int n = stmt1.executeUpdate("update CajasPaso set EstadoCaja=0
where IdCaja="+idcaja);
//se establece en 0 el valor de estado caja
int k = stmt1.executeUpdate("update Cajas set EstadoCaja=0 where
IdCaja="+idcaja);
Salir();//se llama al método salir
```

Código 2.21. Fragmento de código utilizado para cancelar una compra

```
119 else
120 {
121 //se establece la sentencia a ser enviada a la base de datos
122 String query = "select * from Facturas where
IdCliente='"+idunico+"'";
123 //se envia la sentencia a la base de datos
124 Statement stmt = con.createStatement();
125 // variable para almacenar la respuesta a la base de datos
126 ResultSet rs = stmt.executeQuery(query);
127 /se establece un lazo para recorrer la información almacenada en
variable rs
128   while (rs.next()) {
129
130 //se agrega en la base de datos los valores que se obtienen en las
131 // distintas columnas que se encuentran almacenados en la variable
rs
132 listadearray.add("      "+rs.getString(3)+"       "+"    " +
133 "    "+rs.getString(1)+"         "+"      "+rs.getString(2)+"     " +
134 "     "+"     "+rs.getString(7));
135 }
adaptadorfacturas.notifyDataSetChanged();//se notifica al adaptador
137 // sobre un cambio en el array, para que agregue los elementos al
listview
138 }con.close();//se cierra la conexión con la base de datos
```

Código 2.22. Fragmento de código para la consulta de facturas

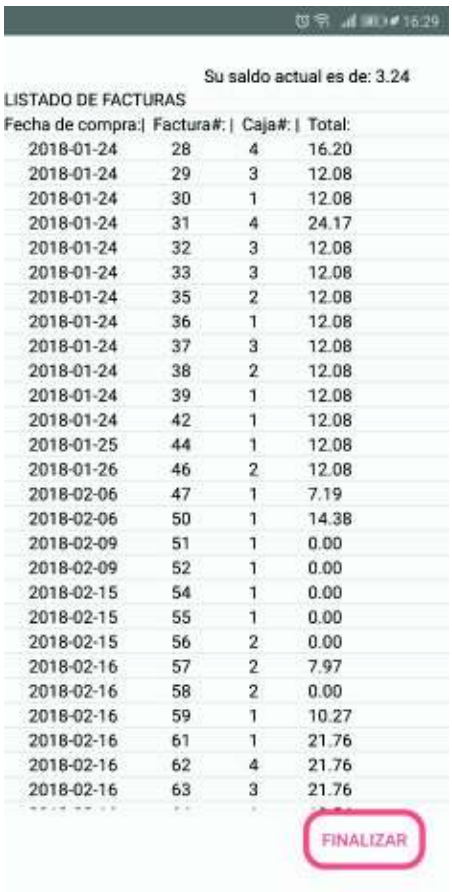


Figura 2.30. Vista del módulo de facturas

Tabla 2.28. Estructura de la aplicación ASP.NET

Nombre del archivo	Descripción
Wfrm_Categorias_Productos.aspx Wfrm_Clientes.aspx Wfrm_Empleados.aspx Wfrm_Encargados.aspx Wfrm_Facturas.aspx Wfrm_Inicio.aspx Wfrm_Inventario.aspx Wfrm_Login.aspx Wfrm_Productos.aspx Wfrm_Reporte_Ventas.aspx	Estos archivos son las páginas web ASP.NET que contienen la interfaz de usuario. Los usuarios solicitan o navegan directamente en una de estas páginas para interactuar con la aplicación.
Wfrm_Categorias_Productos.cs Wfrm_Clientes.cs Wfrm_Empleados.cs Wfrm_Encargados.cs Wfrm_Facturas.cs Wfrm_Inicio.cs Wfrm_Inventario.cs Wfrm_Login.cs Wfrm_Productos.cs Wfrm_Reporte_Ventas.cs	Estos archivos contienen el código C # subyacente (*code-behind*). Además, permiten separar la lógica de la aplicación de la interfaz de usuario.
Web.config	Este es el archivo de configuración de la aplicación ASP.NET.

a. Conexión a la base de datos

De manera análoga a la aplicación para la caja electrónica, la aplicación web utiliza el componente LINQ to SQL proporcionado por el .NET *Framework* para su comunicación con la base de datos.

En primer lugar, se requiere añadir la clase LINQ to SQL al proyecto. Después, se realiza la conexión a la base de datos y simplemente se arrastran las tablas deseadas al diseñador relacional de objetos. Entonces, automáticamente se crean las clases de datos como se puede apreciar en la Figura 2.31.

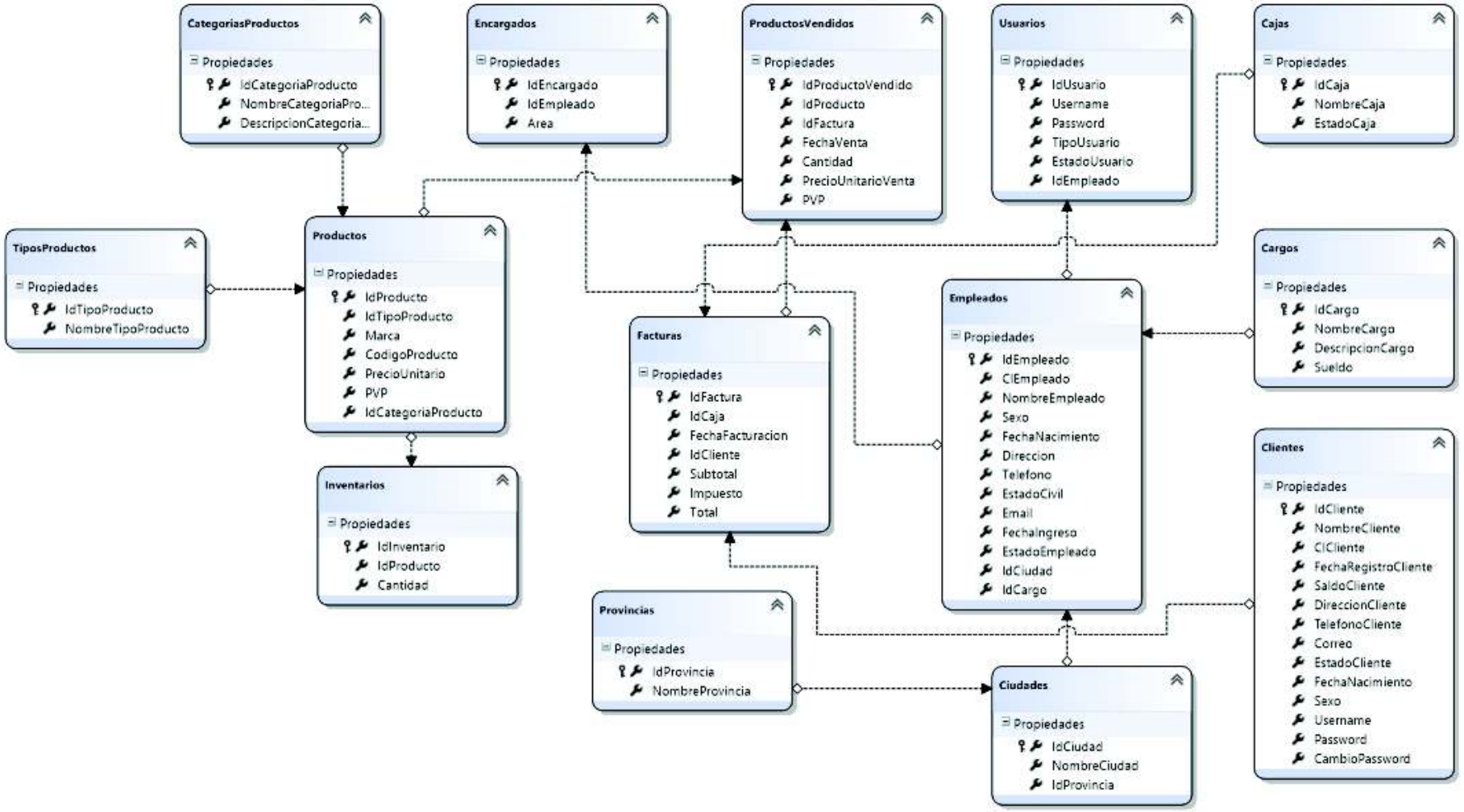


Figura 2.31. Clases de datos de la aplicación web

b. Interfaces de usuario

Para el desarrollo de las interfaces de usuario se utiliza una plantilla elaborada con los lenguajes de marcado y estilo HTML5 y CCS3 respectivamente, bajo una licencia Creative Commons, con el fin de, obtener un sitio web responsivo e interactivo para el usuario final.

Las interfaces que se implementan en la aplicación se detallan en la Tabla 2.29.

Tabla 2.29. Formularios de la aplicación web

Formulario	Descripción
`Wfrm_Categorias_Productos`	Formulario que permite insertar, modificar y eliminar registros en el componente gestión de categorías
`Wfrm_Clientes`	Formulario que permite insertar, modificar y eliminar registros en el componente gestión de clientes
`Wfrm_Empleados`	Formulario que permite insertar, modificar y eliminar registros en el componente gestión de empleados
`Wfrm_Encargados`	Formulario que permite insertar, modificar y eliminar registros en el componente gestión de encargados
`Wfrm_Facturas`	Formulario que permite visualizar las facturas de las compras de los clientes
`Wfrm_Inicio`	Formulario de inicio que brinda información del establecimiento
`Wfrm_Inventario`	Formulario que permite gestionar el inventario de productos del establecimiento
`Wfrm_Login`	Formulario que permite la autenticación del usuario al sistema
`Wfrm_Productos`	Formulario que permite insertar, modificar y eliminar registros en el componente gestión de productos
`Wfrm_Reporte_Ventas`	Formulario que permite visualizar los reportes de ventas del establecimiento

ASP.NET proporciona controles web para la implementación de las interfaces. La Tabla 2.30 enumera los controles y los elementos HTML utilizados en el desarrollo de los formularios de la aplicación web.

Tabla 2.30. Controles web

Controles web	Elemento HTML subyacente
Etiqueta	`<span>`
Botón	`<input type="button">`
Caja de texto	`<input type="text">`
Casillas de verificación	`<input type="checkbox">`
Botón de radio	`<input type="radio">`
Hipervínculo	`<a>`
Botón de imagen	`<input type="image">`
Imagen	`<img>`
Lista desplegable	`<select>`
Panel	`<div>`
Tabla, fila, celda	`<table>, <tr>, <td>`
Formulario	`<form>`

b.1. Página maestra

En primer lugar, se define la página maestra `MasterPage.Master` que establece la estructura general y los elementos comunes a todos los formularios. Entonces, cada página del sitio web utiliza esa página maestra para obtener la misma organización básica: el mismo encabezado, pie de página y menú lateral. Sin embargo, cada página inserta su información específica en esta plantilla. En el Código 2.23, se aprecia el control `ContentPlaceHolder` donde se inserta dicha información específica a cada página. Por lo tanto, toda la información fuera de este control se mantiene igual en todos los formularios que hacen uso de la página maestra.

```
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
```

Código 2.23. Control `ContentPlaceHolder` de la página maestra

Después, al crear una página de contenido, ASP.NET vincula dicha página a la página maestra agregando un atributo a la directiva de la página. Este atributo, denominado `MasterPageFile`, indica la página maestra asociada. En el Código 2.24, se expone la asociación entre la página maestra `MasterPage.Master` y la página de contenido `Wfrm_Encargados.aspx`.

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPage.Master"
AutoEventWireup="true" CodeBehind="Wfrm_Encargados.aspx.cs"
Inherits="Market_Web_Management.Wfrm_Encargados" %>
```

Código 2.24. Atributo `MasterPageFile` de una página de contenido

b.2. Formularios de gestión

Los formularios de gestión de productos, categorías, clientes, empleados y encargados utilizan una tabla definida por la etiqueta `<table>` para la visualización de sus registros existentes (Figura 2.32), mientras que para la inserción y modificación de la información de sus registros utiliza un panel ASP (Figura 2.33) definido por la etiqueta `<ASP:panel>`.

Figura 2.32. Tabla de elementos del formulario `Wfrm_Encargados`

Figura 2.33. Panel de datos del formulario `Wfrm_Encargados`

b.3. Formularios de reportes

Los formularios para la visualización de reportes de ventas y facturas utilizan un panel donde el usuario puede filtrar la información pertinente al reporte deseado, por ejemplo, el intervalo de tiempo, el nombre del cliente, el número de factura, etc. Además, se utiliza el control `ReportViewer` proporcionado por ASP.NET para la generación de reportes. La Figura 2.34 muestra la disposición de los controles del formulario `Wfrm_Reporte_Ventas`, mientras que, la Figura 2.35 presenta el formato del reporte de ventas utilizado en el establecimiento.

Figura 2.34. Vista del formulario `Wfrm_Reporte_Ventas`

Figura 2.35. Vista del reporte de ventas

c. Lógica de aplicación

c.1. Formularios de gestión

Esta sección ilustra la codificación de los formularios de gestión de productos, categorías, clientes, empleados y encargados. A manera de ejemplo se muestra la codificación correspondiente al formulario de encargados `Wfrm_Encargados`.

c.1.1 Lectura de registros

Para la visualización de los registros de encargados se define un repetidor. Este control se enlaza a un origen de datos y genera una lista de elementos individuales. Cuando se ejecuta el formulario, el control repite el diseño para cada uno de los elementos del origen de datos establecido. En primer lugar, para realizar las consultas a las entidades de la base de datos se instancia la clase `DataContext` (Código 2.25).

```
MarketWebDBDataContext db = new MarketWebDBDataContext();
```

Código 2.25. Instancia de la clase `DataContext`

Posteriormente, para la lectura de los registros existentes se define el método `CargarEncargadosTabla` como se ilustra en el Código 2.26, donde a través de una consulta a las tablas `Empleados`, `Cargos` y `Encargados` se obtiene la información deseada y se la almacena en la variable `encargadosActuales`. Entonces, dicha información se establece como el origen de datos y se enlaza al repetidor `rptEncargados` a través del método `DataBind`.

```
private void CargarEncargadosTabla()
        {
            var encargadosActuales = from tbEncargados in db.Encargados
                                     join tbEmpleados in db.Empleados
                                     on tbEncargados.IdEmpleado equals
                                     tbEmpleados.IdEmpleado
                                     join tbCargos in db.Cargos
                                     on tbEmpleados.IdCargo equals
                                     bCargos.IdCargo
                                     select new { tbEncargados, tbEmpleados,
                                     tbCargos };
            rptEncargados.DataSource = encargadosActuales;
            rptEncargados.DataBind();
        }
```

Código 2.26. Método `CargarEncargadosTabla`

c.1.2. Inserción de registros

En primer lugar, se define un método que limpia los campos de información del panel de datos del encargado `pnlDatosEncargado` (Código 2.27).

```
private void LimpiarCampos()
        {
            txtIdEncargado.Text = string.Empty;
            txtAreaEncargado.Text = string.Empty;
        }
```

Código 2.27. Método `LimpiarCampos`

Luego, una vez que el usuario hace *click* en el botón agregar `btnAgregar`, se realiza la llamada al método `AgregarEncargado` para insertar el nuevo registro en la base de datos (Código 2.28).

```
Encargados encargadoNuevo = new Encargados();
                encargadoNuevo.Area = txtAreaEncargado.Text;
                encargadoNuevo.IdEmpleado = (int)Session["IdEmpleado"];
                db.Encargados.InsertOnSubmit(encargadoNuevo);
                db.SubmitChanges();
```

Código 2.28. Extracto del método `AgregarEncargado`

c.1.3. Modificación de registros

Para la modificación de un registro se utiliza el identificador único del encargado. Para ello, dentro del repetidor se define un `ImageButton` denomidado `btnEditarEncargado` que se utiliza para la obtención de este parámetro. En el Código 2.29, se evidencia como la propiedad `CommandArgument` del botón, se enlaza al origen de datos que contiene el identificador único del encargado a través de la expresión: `<%#Eval("tbEncargados.IdEncargado")%>`.

```
<asp:ImageButton ImageUrl="images/icons/btnEdit.png" runat="server"
ID="btnEditarEncargado" ToolTip="Editar" Width="8%"
CommandArgument='<%# Eval("tbEncargados.IdEncargado") %>'
OnClick="btnEditarEncargado_Click" />
```

Código 2.29. Extracto del control `rptEncargados`

Después, una vez que el usuario hace *click* en el botón editar `btnEditar`, el identificador del encargado es enviado al método que carga la información en el panel de datos (Código 2.30).

```
protected void btnEditarEncargado_Click(object sender, EventArgs e)
        {
            ImageButton btnEditar = (ImageButton)sender;
         CargarEncargadoSeleccionado(Convert.ToInt16(btnEditar.CommandArgument));
            pnlDatosEncargados.Visible = true;
            btnAgregarEncargado.Text = "Actualizar";
        }
```

Código 2.30. Método `btnEditarEncargado_Click`

Finalmente, una vez que el usuario haya realizado las modificaciones deseadas, se utiliza el método `ActualizarEncargado` para guardar los cambios en la base de datos (Código 2.31). Puesto que, se emplea el identificador único como parámetro para discriminar los registros se garantiza que no haya alteraciones no deseadas.

```
Encargados encargadoActualizar = db.Encargados.ToList().Find(x =>
x.IdEncargado.Equals(idEncargado));
encargadoActualizar.Area = txtAreaEncargado.Text;
encargadoActualizar.IdEmpleado = (int)Session["IdEmpleado"];
db.SubmitChanges();
```

Código 2.31. Extracto del método `ActualizarEncargado`

c.1.4. Eliminación de registros

De manera análoga a la modificación de registros, para la eliminación de encargados se utiliza el identificador único como parámetro discriminatorio. En el Código 2.32, se muestra como la propiedad `CommandArgument` del botón, se enlaza al origen de datos que contiene el identificador único del encargado.

```
<asp:ImageButton ImageUrl="images/icons/btnDelete.png" runat="server"
ToolTip="Eliminar" Width="8%"
CommandArgument='<%# Eval("tbEncargados.IdEncargado") %>'
CommandName='<%# Eval("tbEncargados.Area") %>'
OnClick="btnEliminarEncargado_Click" />
```

Código 2.32. `Imagebutton` del `rptEncargados`

Luego, una vez que el usuario hace *click* en el botón eliminar `btnEliminar`, el identificador del encargado es enviado al método que se encarga de borrar dicho registro (Código 2.33).

```
var encargadoEliminar = db.Encargados.ToList().Find(x =>
x.IdEncargado.Equals(int.Parse(idEncargado)));
db.Encargados.DeleteOnSubmit(encargadoEliminar);
db.SubmitChanges();
```

Código 2.33. Extracto del método `EliminarEncargado`

c.2. Formularios de reportes

Esta sección ilustra la codificación de los formularios de reportes de ventas y facturas. A manera de ejemplo se muestra la codificación correspondiente al formulario reporte de ventas `Wfrm_Reporte_Ventas`.

En primer lugar, se proporciona un panel de datos para filtrar la información pertinente del reporte de ventas deseado. Por ejemplo, se puede obtener un reporte de ventas total o uno que comprende un intervalo de tiempo definido por fechas.

Se define un solo control `ReportViewer`, el mismo que se actualiza dependiendo del filtro de reporte deseado. El Código 2.34 muestra el método que realiza la actualización del origen de datos del reporte.

```
void ActualizarReporte(ReportDataSource rds)
        {
            //Se permite la visualización del control del reporte
            ReportViewer1.Visible = true;
            //Se quita todos los elementos de la colección de orígenes de datos
del reporte
            ReportViewer1.LocalReport.DataSources.Clear();
            //Se agrega un objeto a la colección de orígenes de datos del reporte
            ReportViewer1.LocalReport.DataSources.Add(rds);
            //Se enlaza los datos al control del reporte
            ReportViewer1.DataBind();
            //Se refrescan los datos del reporte
            ReportViewer1.LocalReport.Refresh();
        }
```

Código 2.34. Método `ActualizarReporte`

Para obtener la información de los orígenes de datos de los reportes se utilizaron los procedimientos almacenados definidos en la base de datos. El Código 2.35 ilustra el método utilizado para generar el reporte de ventas total del establecimiento.

Además, se define una variable denominada `Resultados` que almacena los datos obtenidos a través del procedimiento almacenado `Sp_Ventas_General_Total` y se construye un origen de datos con esta información para enlazarlo al reporte. Por último, se realiza la llamada al método `ActualizarReporte` para actualizar el origen de datos del reporte.

```
private void CargarVentasGeneral()
        {
            //Se almanacena los datos de la factura en una variable
            //Los datos son obtenidos mediante el procedimiento almacenado
            var Resultados = db.Sp_Ventas_General_Total();
            //Se construye un nuevo origen de datos con los resultados obtenidos
previamente
            ReportDataSource rds = new ReportDataSource("DataSet1", Resultados);
            //Se llama al método de actualización de reportes y se envía el
origen de datos creado anteriormente
            ActualizarReporte(rds);
        }
```

Código 2.35. Método `CargarVentasGeneral`

3. RESULTADOS Y DISCUSIÓN

En el presente capítulo se indican los resultados más relevantes obtenidos de la evaluación del sistema prototipo. Se evaluó tanto el sistema colectivo como sus componentes individuales, teniendo en consideración los objetivos planteados. En primer lugar, en la Sección 3.1 se presentan los resultados obtenidos de las pruebas de funcionalidad de cada uno de los módulos de la aplicación web. Después, en la Sección 3.2 se describen los resultados obtenidos al realizar pruebas similares a los módulos de la aplicación móvil. Entonces, una vez evaluadas ambas aplicaciones, en la Sección 3.3 se presentan las pruebas de funcionalidad del sistema prototipo en conjunto, donde se obtuvieron estimaciones de los tiempos de lectura individuales y colectivos de los productos de las compras. Finalmente, en la Sección 3.4 se expone un análisis de costos para la puesta en funcionamiento del sistema prototipo.

3.1 Pruebas de funcionalidad de la aplicación web

La aplicación móvil está conformada por tres tipos de módulos: el formulario de autenticación o *login*, los formularios de gestión (productos, categorías, clientes, empleados y encargados) y los formularios de reportes de ventas. A continuación, se presentan los resultados obtenidos de las pruebas de funcionalidad para cada módulo mencionado. Conviene señalar que las pruebas a los formularios de gestión fueron realizadas sobre el formulario de empleados `Wfrm_Empleados`.

3.1.1 Módulo de autenticación

La autenticación en la aplicación web se realiza mediante un nombre de usuario y contraseña. Para esta prueba se utilizó el perfil del empleado “Daniel Hwang” cuyo nombre de usuario es “dhwang” y su contraseña “corpo123”. Además, se desactivó temporalmente el atributo `TextMode=Password` para poder visualizar los caracteres ingresados en el cuadro de texto de la contraseña.

Primeramente, se evaluó el ingreso mediante las credenciales erróneas, nombre de usuario “daniel” y contraseña “123”. La Figura 3.1 ilustra el resultado de la autenticación fallida.

Posteriormente, se verificó el ingreso a la aplicación web mediante las credenciales correctas (Figura 3.2). Una vez que se validan las credenciales de autenticación, la aplicación se redirige automáticamente al formulario de inicio o *home* `Wfrm_Inicio.aspx` (Figura 3.3).

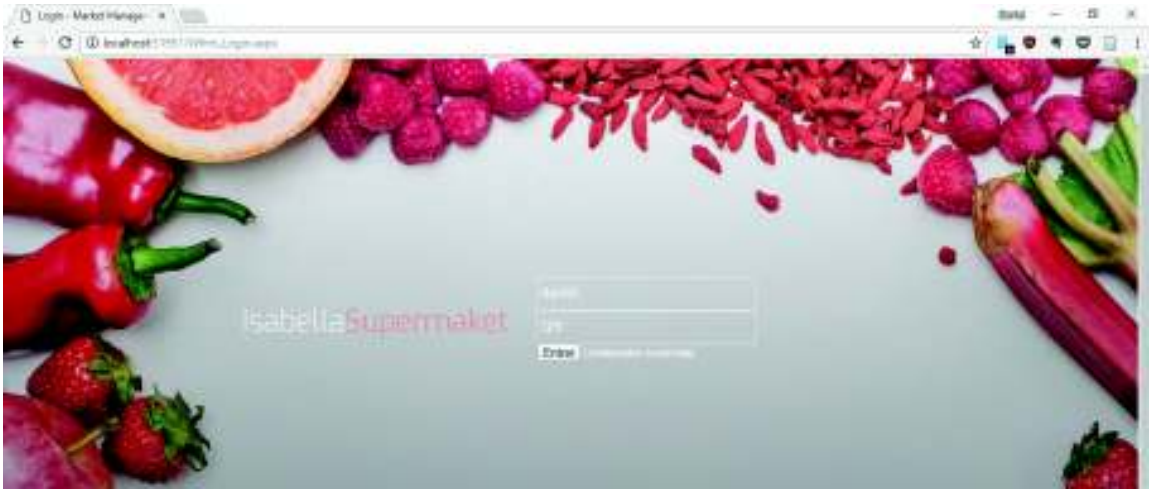


Figura 3.1. Autenticación fallida

Figura 3.2. Autenticación exitosa

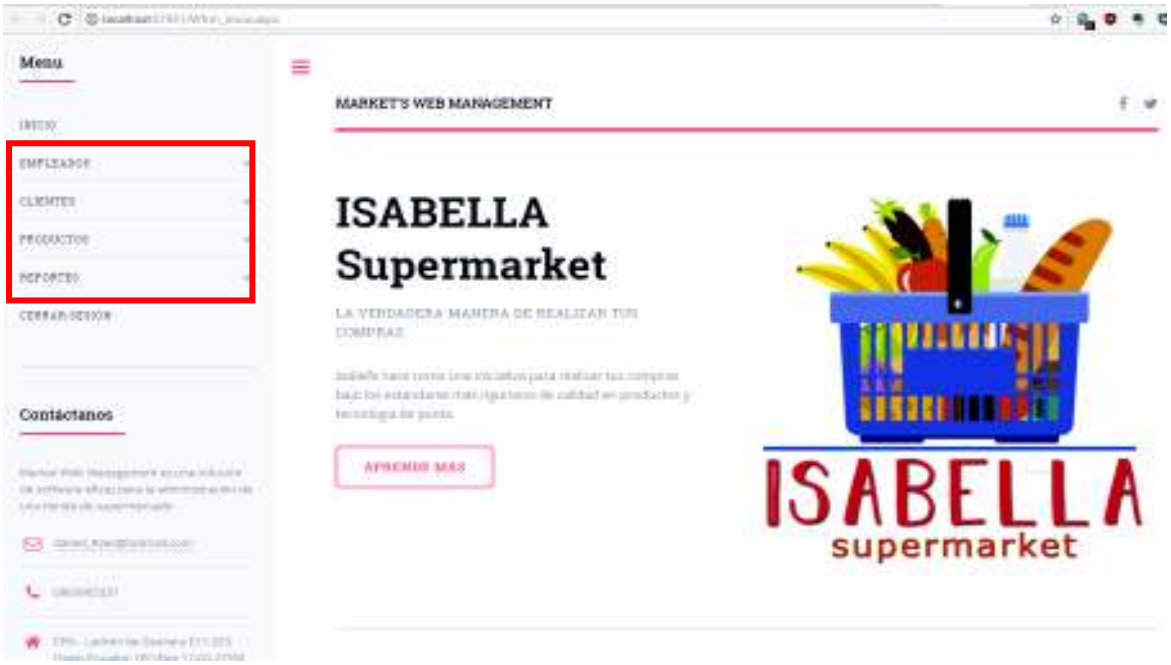


Figura 3.3. Formulario de inicio para un usuario administrador

Conviene señalar que el usuario “Daniel Hwang” es del tipo administrador, por lo tanto, como se muestra en la figura Figura 3.3 posee acceso a todos los módulos de la aplicación web (i.e. empleados, clientes, productos y reportes).

Sin embargo, cuando un usuario de servicio al cliente se autentica es redireccionado a un formulario de inicio con funcionalidades relativas a las cuentas de los clientes y sus facturas (Figura 3.4).

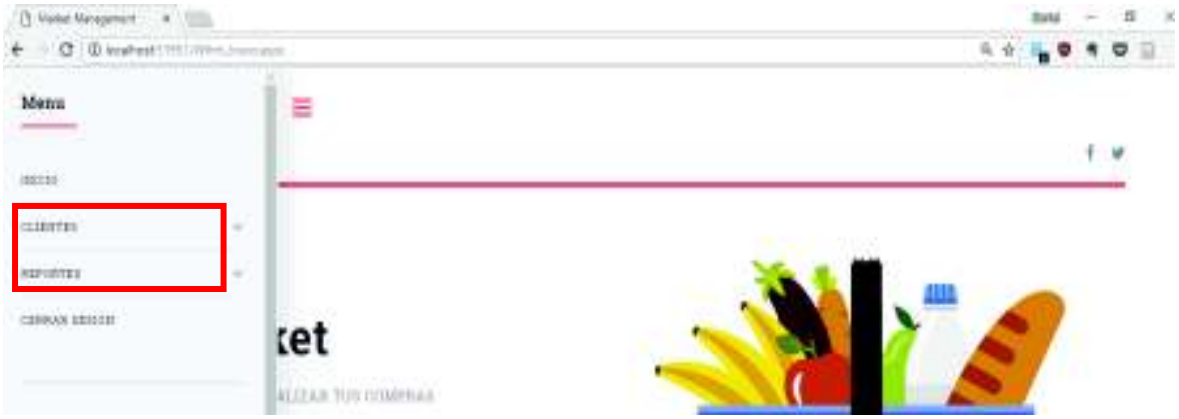


Figura 3.4. Formulario de inicio para usuario de servicio al cliente

3.1.2 Módulos de gestión

a. Lectura de los registros existentes

Se comprobó que los registros existentes en la tabla `Empleados` de la base de datos (Figura 3.5) coincidan con los registros mostrados en la tabla de empleados del formulario `Wfrm_Empleados.aspx` (Figura 3.6).

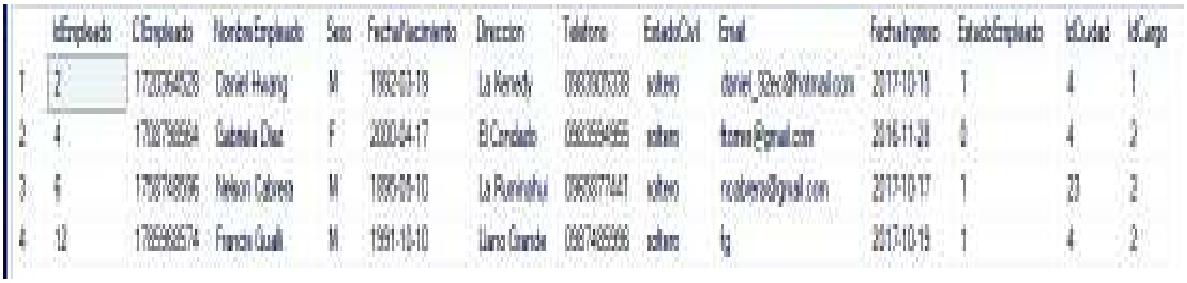


Figura 3.5. Registros de la tabla `Empleados`

b. Creación de un registro

En primer lugar, se verificó el ingreso correcto de un perfil nuevo de empleado mediante la aplicación web. Para ello se presionó el botón “Nuevo Empleado” (Figura 3.7). Luego, la Figura 3.8 ilustra el panel de datos a ingresarse, mientras que la Figura 3.9 muestra el mensaje de confirmación de un ingreso adecuado.

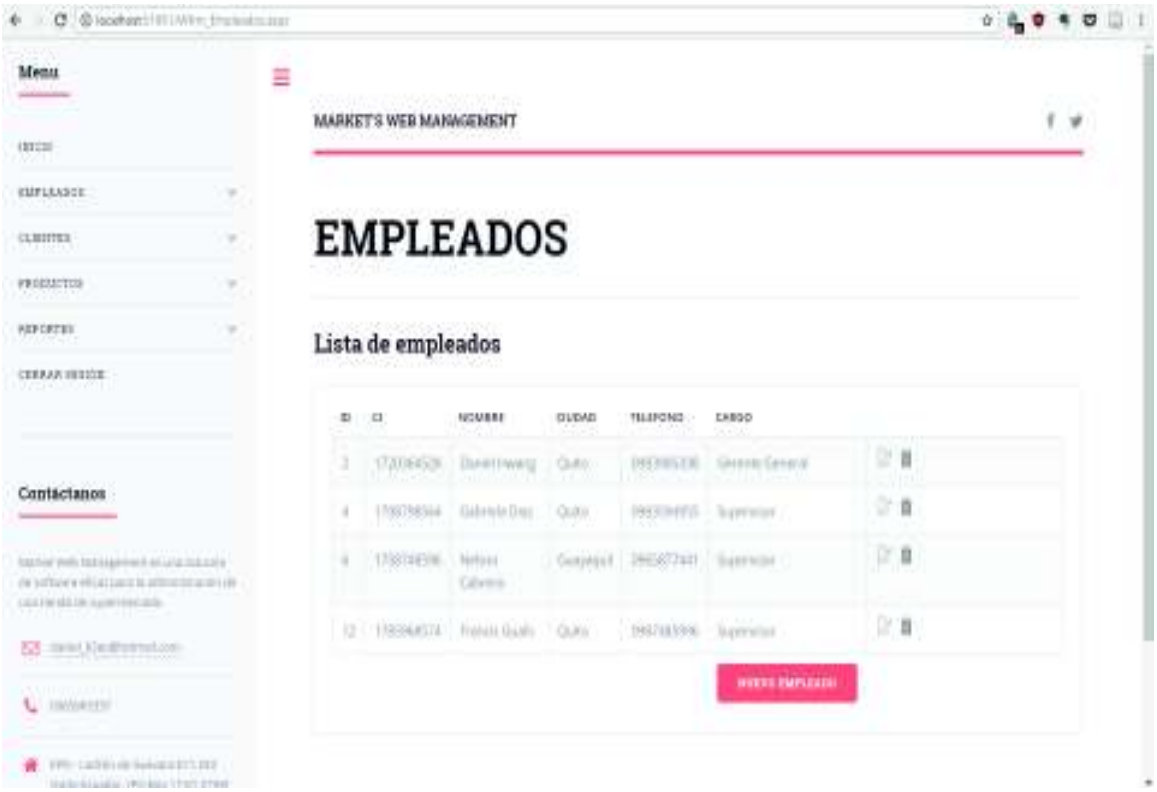


Figura 3.6. Elementos de la tabla de empleados del formulario

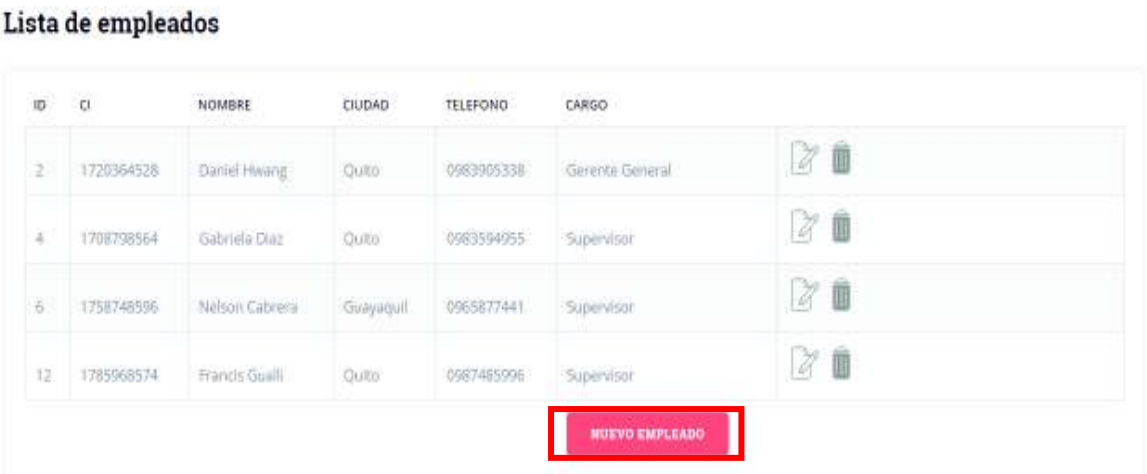


Figura 3.7. Botón para la creación de un perfil de empleado

Después, se verificó que el perfil de empleado nuevo creado anteriormente se muestre en la tabla de empleados existentes del formulario `Wfrm_Empleados.aspx` (Figura 3.10). Finalmente, se comprobó el ingreso de dicho perfil en la tabla `Empleados` de la base de datos (Figura 3.11).

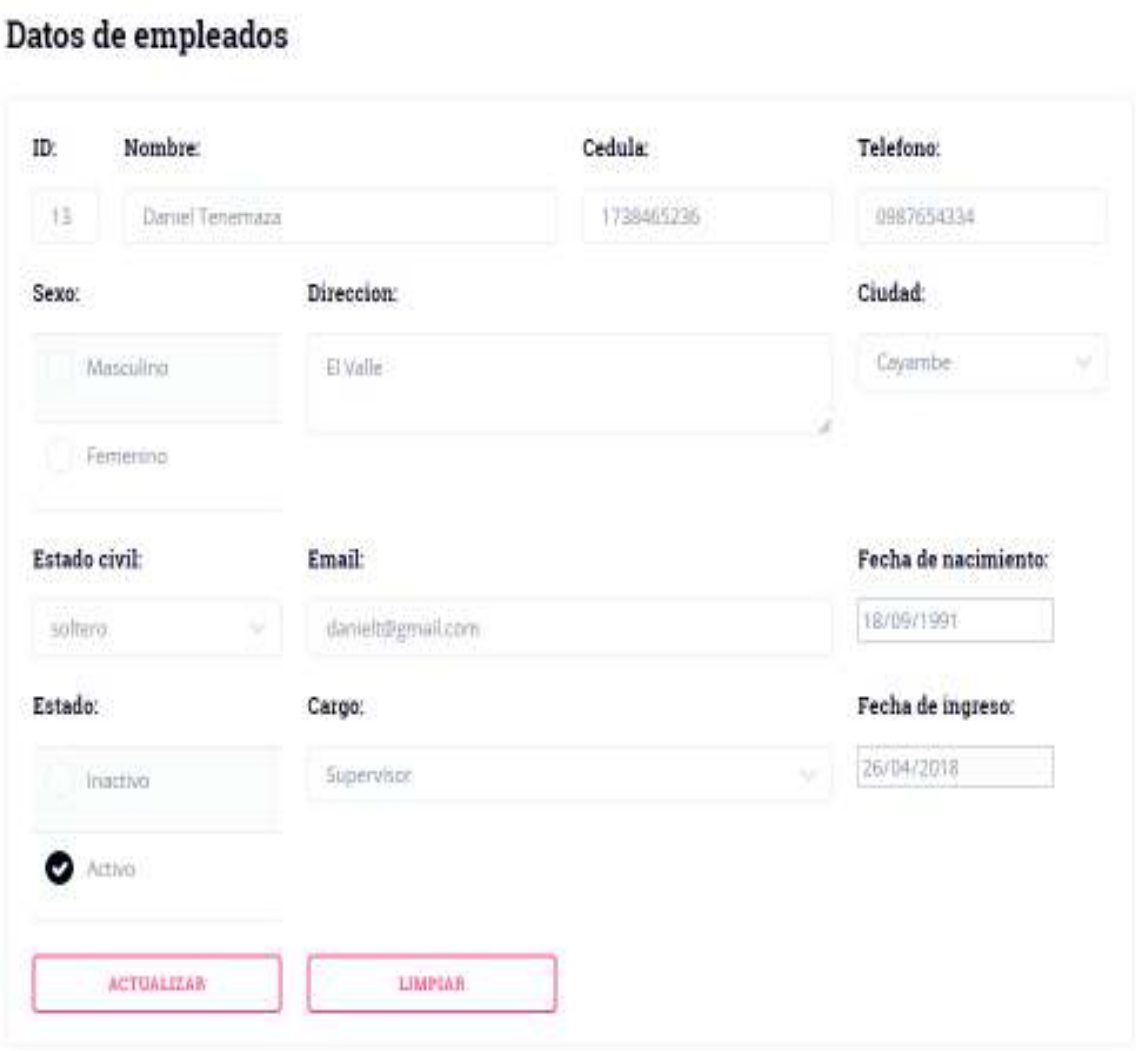


Figura 3.8. Panel de datos para la creación de un nuevo empleado

Figura 3.9. Mensaje de confirmación de creación de un empleado

Figura 3.10. Verificación de la creación del perfil de empleado

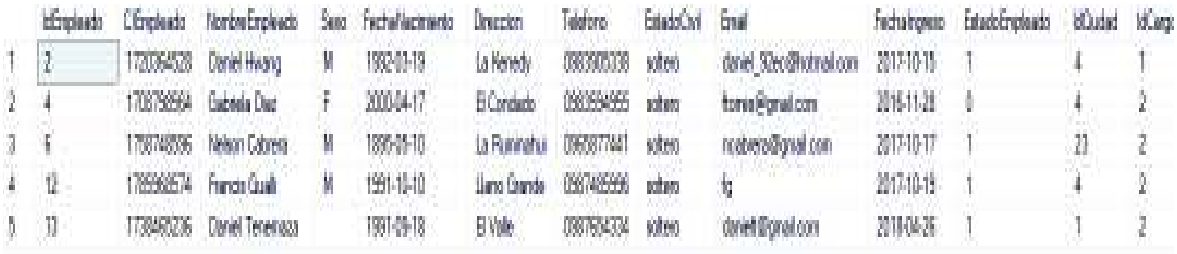


Figura 3.11. Comprobación de la creación de un empleado en la base de datos

c. Modificación de un registro

En primer lugar, se verificó la modificación correcta del perfil de empleado creado en la sección anterior, mediante la aplicación web. Para ello se presionó el botón de edición (Figura 3.12). Luego, la Figura 3.13 ilustra el panel con los datos modificados, mientras que la Figura 3.14 muestra el mensaje de confirmación de modificación del perfil de empleado.

Figura 3.12. Botón para la modificación de un perfil de empleado

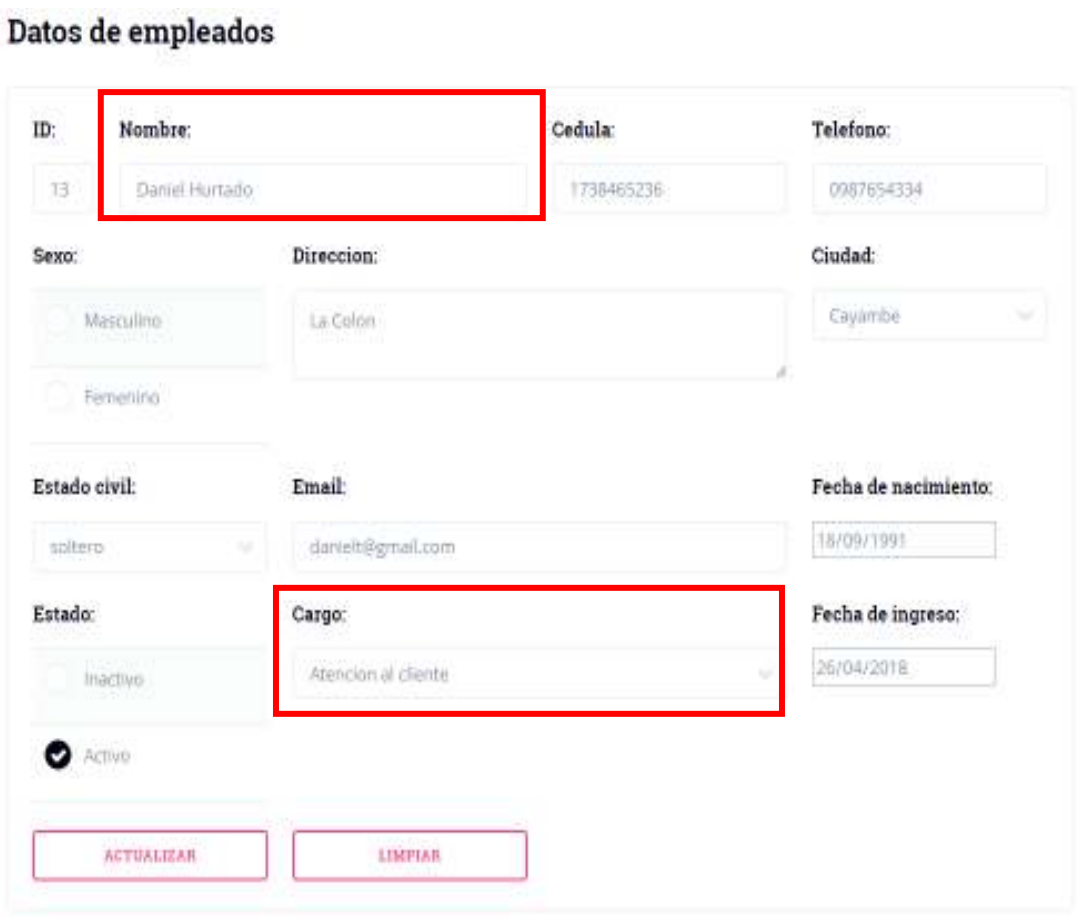


Figura 3.13. Panel de datos del perfil del empleado a modificarse

Figura 3.14. Mensaje de confirmación de la modificación de un empleado

Después, se verificó que el perfil del empleado modificado se actualizó en la tabla de empleados existentes del formulario `Wfrm_Empleados.aspx` (Figura 3.15). Finalmente, se comprobó el ingreso de dicho perfil en la tabla `Empleados` de la base de datos (Figura 3.16).

Figura 3.15. Verificación de la modificación del perfil de empleado

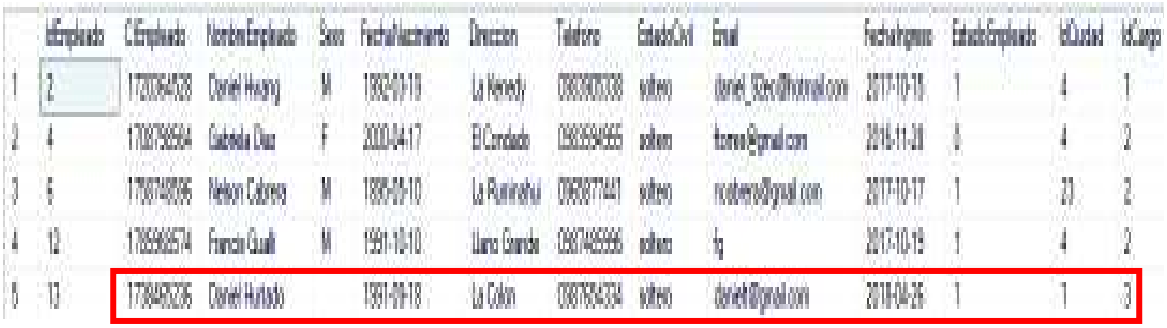


Figura 3.16. Comprobación de la modificación de un empleado en la base de datos

d. Eliminación de un registro

En primer lugar, se verificó la eliminación correcta del perfil de empleado creado en la sección A, mediante la aplicación web. Para ello se presionó el botón para eliminar (Figura 3.17). Luego, la Figura 3.18 muestra el mensaje de confirmación para proceder a la eliminar el registro.

Figura 3.17. Botón para la eliminación de un perfil de empleado

Figura 3.18. Mensaje de confirmación para eliminar un empleado

Después, se verificó que el perfil de empleado eliminado ya no conste más en la tabla de empleados existentes del formulario `Wfrm_Empleados.aspx` (Figura 3.19). Finalmente, se comprobó la eliminación de dicho perfil en la tabla `Empleados` de la base de datos (Figura 3.20).

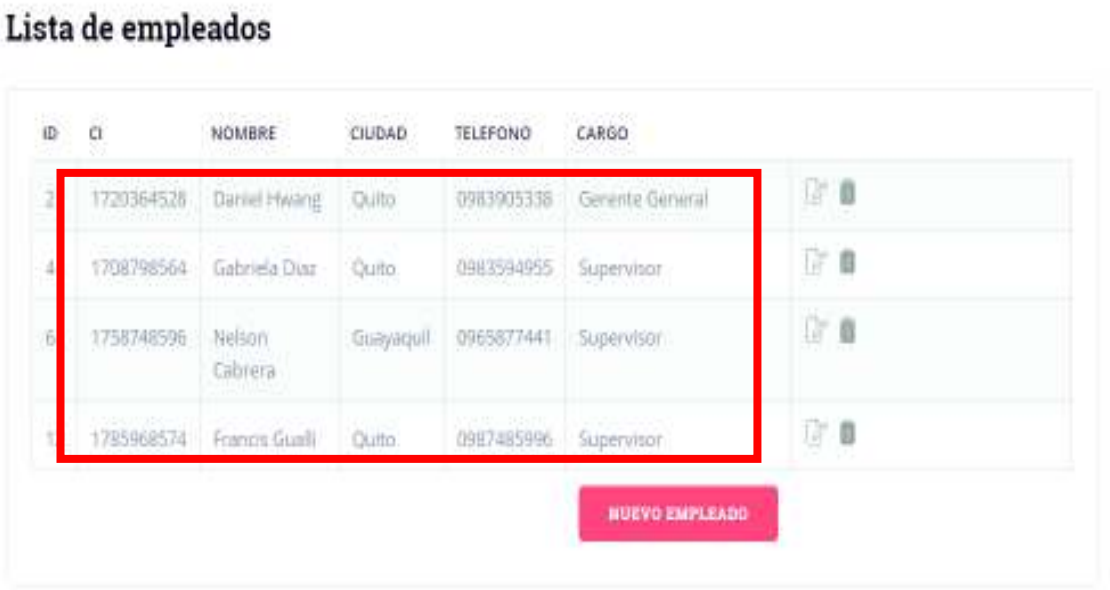


Figura 3.19. Validación de la eliminación de un empleado

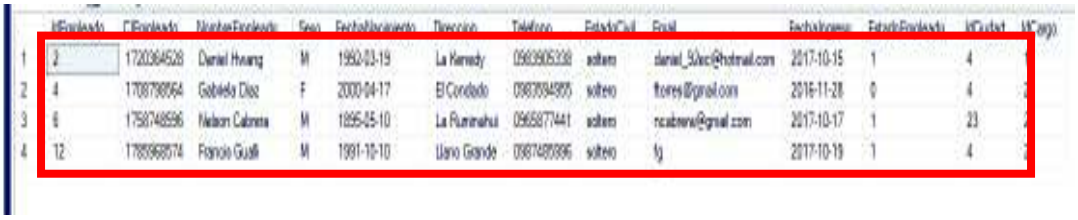


Figura 3.20. Comprobación de la eliminación de un empleado en la base de datos

3.1.3 Módulo de reportes

Se corroboró la generación adecuada de reportes de ventas del establecimiento. Este formulario dispone de un panel con filtros para obtener el reporte con la información pertienente. En este ejemplo, se seleccionó el filtro “General” para obtener la información de todos los clientes y se definió un intervalo de tiempo comprendido entre las fechas 19/02/2018 y 20/02/2018 (Figura 3.21).

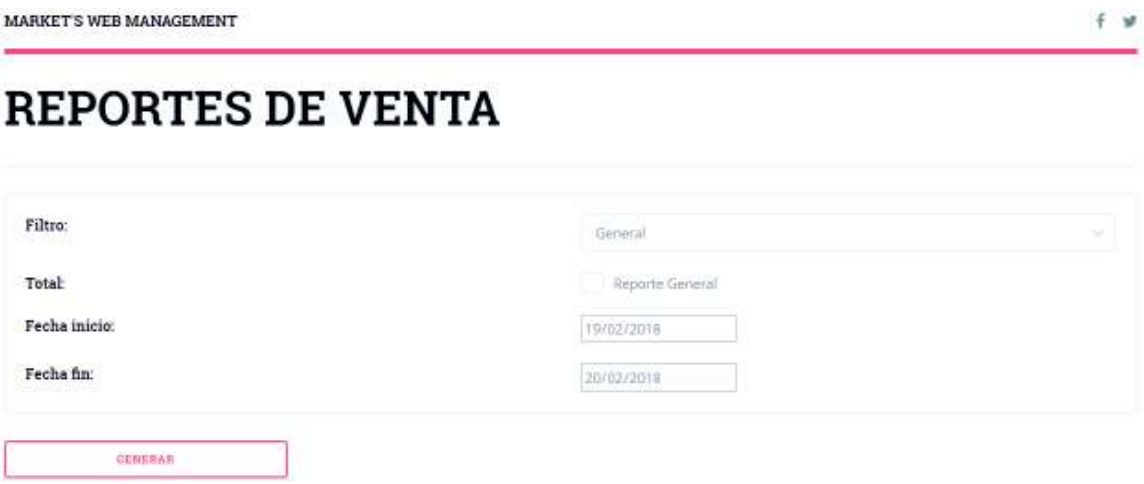


Figura 3.21. Panel de filtros para los reportes

La Figura 3.22 ilustra el resultado del reporte obtenido en el formulario `Wfrm_Reporte_Ventas.aspx`. Además, se verificó la generación de los reportes en formato PDF y Excel (Figura 3.23, Figura 3.24 y Figura 3.25).

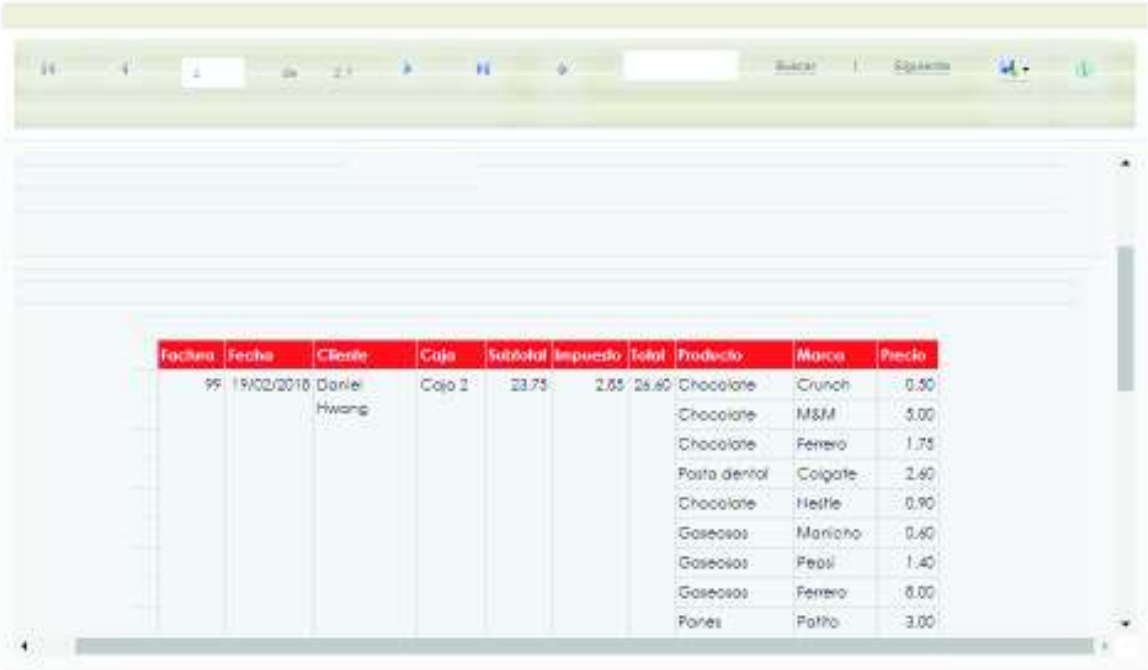
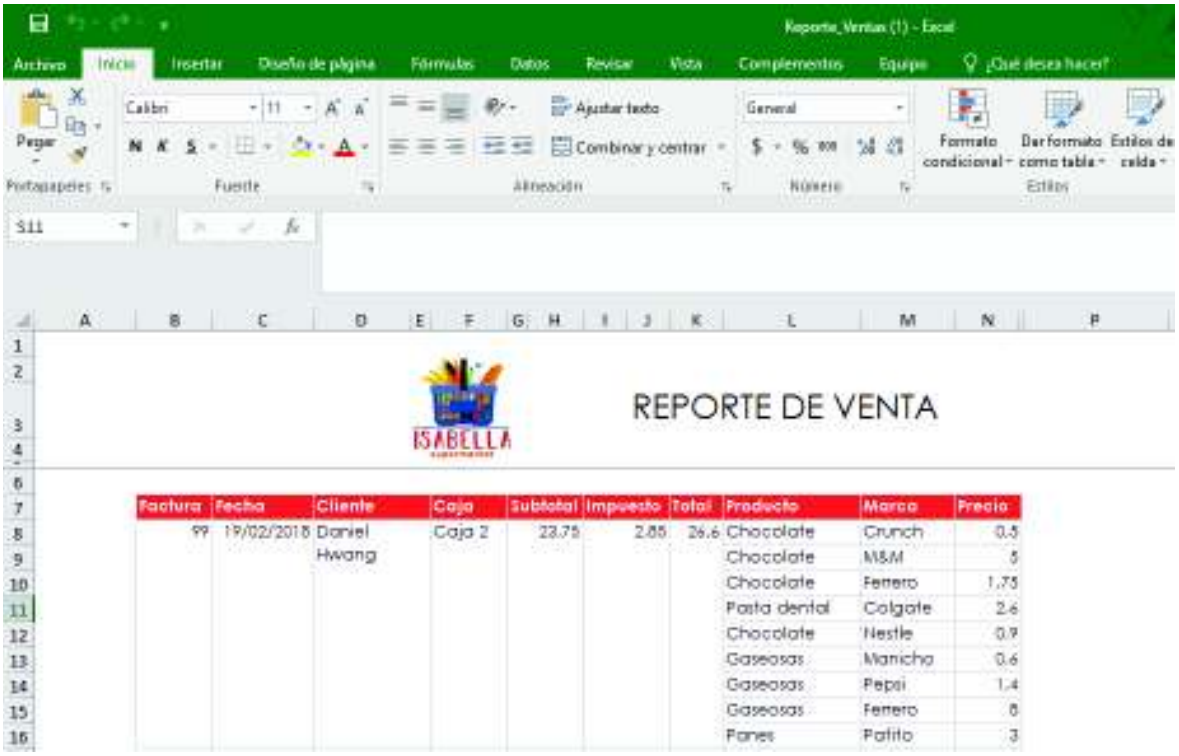


Figura 3.22. Reporte generado en el formulario `Wfrm_Reporte_Ventas.aspx`

Figura 3.23. Botón para la generación de reportes en formato PDF y Excel

Factura	Fecha	Cliente	Caja	Subtotal	Impuesto	Total	Producto	Marca	Precio
99	19/02/2018	Daniel Hwang	Caja 2	23.75	2.85	26.60	Chocolate	Crunch	0.50
							Chocolate	M&M	5.00
							Chocolate	Ferrero	1.75
							Pasta dental	Colgate	2.60
							Chocolate	Nestle	0.90
							Gaseosas	Manicho	0.60
							Gaseosas	Pepsi	1.40
							Gaseosas	Ferrero	8.00
							Panes	Patito	3.00

Figura 3.24. Reporte de ventas en formato PDF



Factura	Fecha	Cliente	Caja	Subtotal	Impuesto	Total	Producto	Marca	Precio
99	19/02/2018	Daniel Hwang	Caja 2	23.75	2.85	26.6	Chocolate	Crunch	0.5
							Chocolate	M&M	5
							Chocolate	Ferrero	1.75
							Pasta dental	Colgate	2.6
							Chocolate	Nestle	0.9
							Gaseosas	Manicho	0.6
							Gaseosas	Pepsi	1.4
							Gaseosas	Ferrero	8
							Panes	Patito	3

Figura 3.25. Reporte de ventas en formato Excel

3.2 Pruebas de funcionalidad de la aplicación móvil

La aplicación móvil fue sometida a pruebas de funcionalidad tanto en el simulador que provee el IDE Android Studio como en un dispositivo real. Para las pruebas dentro del simulador se utilizó un dispositivo virtual con un sistema operativo Android 5.1 o Lollipop, mientras que el dispositivo móvil real que se utilizó para las pruebas fue un Huawei P Smart que posee un sistema operativo Android 8.0 u Oreo.

3.2.1 Módulo de autenticación

La autenticación en la aplicación móvil se realiza mediante la verificación del nombre de usuario y contraseña del cliente en la base de datos, para ello es necesario que el dispositivo móvil esté conectado a la red local del establecimiento.

En la primera prueba se intentó ingresar al sistema sin ingresar las credenciales de autenticación, por lo que la aplicación arrojó un mensaje indicando la ausencia de las mismas (Figura 3.26).

Figura 3.26. Intento de autenticación sin credenciales

Posteriormente, se intentó la autenticación del usuario sin estar conectado previamente a la red local del establecimiento. La Figura 3.27 ilustra el notificación de alerta para la comprobación de la conexión a la red.

Figura 3.27. Notificación de ausencia de conexión a la red local

Luego, se verificó la discriminación de letras mayúsculas y minúsculas en las credenciales de autenticación. Dicha prueba se realizó con el perfil del cliente “Daniel Tenemaza” cuyo nombre de usuario es “danielct” y su contraseña es “daniel12345”. La Figura 3.28 muestra la notificación de error al intentar autenticarse con el nombre de usuario “Danielct” y la contraseña “Daniel12345”.

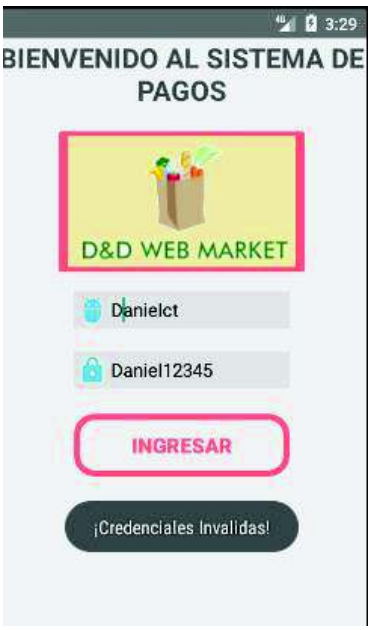


Figura 3.28. Intento de autenticación con credenciales incorrectas

Una vez que se ingresaron las credenciales correctas (i.e. el nombre de usuario y la contraseña provisional) el sistema mostró un mensaje de éxito en la autenticación (Figura 3.29).

Figura 3.29. Autenticación exitosa

Finalmente, se verificó que el sistema solicite al cliente el ingreso de una nueva contraseña (Figura 3.30).

Figura 3.30. Solicitud de cambio de contraseña

3.2.2 Módulo de selección de caja

Se comprobó que el sistema garantice la utilización exclusiva de cajas libres o disponibles. La distinción de cajas libres y ocupadas se realizó mediante un `radiobutton`, cuando este control está marcado la caja está ocupada, caso contrario, está libre.

En esta prueba se intentó seleccionar la caja 2, por lo tanto, la aplicación presentó una notificación de alerta solicitando la elección de una caja libre (Figura 3.31).

Figura 3.31. Intento de selección de una caja en uso

3.2.3 Módulo de pago

En este módulo se verificó la visualización del detalle de los productos escaneados (i.e. el precio individual de cada producto, la marca y el total a pagar de la compra), como se muestra en la Figura 3.32.

Además, se comprobó que la aplicación permita actualizar la lista de productos a ser adquiridos. La Figura 3.33 ilustra la actualización de los productos de la compra anterior.

Figura 3.32. Verificación de productos escaneados

Figura 3.33. Actualización de los productos de la compra

Posteriormente, se verificó que la aplicación permita efectuar el pago de la compra. La Figura 3.34 ilustra el mensaje para la confirmación de la compra.

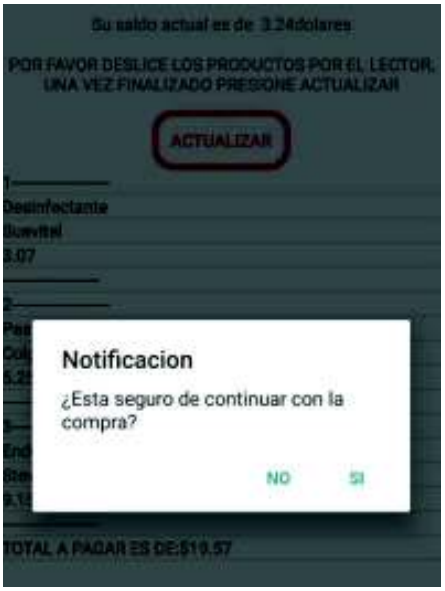


Figura 3.34. Confirmación de la compra

Finalmente, se verificó que el sistema impida la compra cuando el cliente no dispone de saldo disponible. La Figura 3.35 muestra la notificación de saldo insuficiente.

Figura 3.35. Notificación de saldo insuficiente para realizar una compra

3.3 Pruebas del sistema prototipo

Una vez que se ha validado el funcionamiento individual de los componentes del sistema se procede a comprobar el funcionamiento global del prototipo. Para las pruebas se seleccionaron diez productos de primera necesidad y de consumo cotidiano (Figura 3.36 y Tabla 3.1).

Figura 3.36. Productos utilizados para las pruebas de funcionamiento

Tabla 3.1. Productos utilizados para las pruebas de funcionamiento

#	Producto
1	Leche
2	Aromatizante
3	Caja de té
4	Frasco de aceitunas
5	Shampoo
6	Pasta dental
7	Lava vajilla
8	Caja de endulzante
9	Cloro
10	Suavizante

Para empezar, a cada producto se le adhirió una etiqueta o *tag* RFID (Figura 3.37). La Figura 3.38 presenta los identificadores de las etiquetas RFID almacenados en la base de datos.

Figura 3.37. Producto etiquetado

List EPC of Tags

No.	ID	EPCLength	Times
1	E200016871302550860C1EE	12	92
2	E200016871302530860C1ED	12	86
3	E200016871302520860C1F5	12	82
4	E200016871302470860C1FE	12	52
5	E200016871302450860C1FD	12	50
6	E200016871302480860C207	12	31
7	E200016871302510860C200	12	29

Figura 3.38. Verificación del código único de las etiquetas RFID

Posteriormente, se realizó la asignación de los códigos únicos de las etiquetas a los productos (Figura 3.39).

	IdProducto	IdTipoProducto	Marca	CodigoProducto	PrecioUnitario	PVP	IdCategoriaProducto
1	1	11	Suavitel	E200016871302480860C207	3.07	3.07	1
2	2	8	Te Pusuqui	E200016871302530860C1ED	1.29	1.29	3
3	3	11	Clorox Gel	E200016871302450860C1FD	2.09	2.09	1
4	4	2	Lava	E200016871302470860C1FE	2.92	2.92	1
5	5	4	Colgate 3 Pack	E200016871302520860C1F5	5.25	5.25	2
6	6	10	La Lechera	E200016871302540860C1F6	1.49	1.49	1
7	7	12	Dimcofrut	E200016871302400860C217	2.10	2.10	3
8	8	9	Stevia life	E200016871302550860C1EE	9.15	9.15	3
9	9	7	Titanium	E200016871302510860C200	15.00	15.00	2
10	10	5	Arom	E200016871302380860C216	3.00	3.00	1

Figura 3.39. Asignación de los códigos RFID a los productos

Finalmente, se realizó la lectura de los productos con las etiquetas RFID. Entonces, se pudo evidenciar el problema en la lectura de productos con envases metálicos y tetra pack.

Los productos con envases tetra pack contienen una lámina de aluminio. Dado que el metal es un reflector electromagnético, las señales de radio no pueden penetrarlo, por lo tanto, se impide la lectura de las etiquetas RFID adheridas al envase. Entonces, para subsanar

este inconveniente fue necesario utilizar una capa de material aislante entre la superficie de contacto y la etiqueta RFID, en este caso, espuma Flex (Figura 3.40).

Figura 3.40. Productos metálicos con aislantes adheridos

3.3.1 Estimación del tiempo de lectura individual por producto

El intervalo de tiempo de lectura está comprendido desde el momento en el que el cliente toma el producto de la bolsa y lo pasa por el lector RFID hasta obtener su lectura. Para la estimación del tiempo medio de lectura por producto se realizaron 50 lecturas individuales. La Tabla 3.2 presenta los resultados obtenidos. Cabe señalar que el detalle individual de las 50 lecturas de cada producto se adjunta en el Anexo VI.

Tabla 3.2. Tiempo de lectura promedio por producto

Lectura de productos				
#	**Producto**	**Tipo de envase**	**Tiempo promedio (segundos)**	**Desviación estándar**
1	Leche	Tetra pack	3,6	0,9
2	Aromatizante	Metálico	3,6	0,9
3	Caja de té	Cartón	3,5	0,8
4	Frasco de aceitunas	Vidrio	3,6	0,9
5	Shampoo	Plástico	3,6	0,8
6	Pasta dental	Cartón	3,5	0,9
7	Lava vajilla	Plástico	3,4	0,8
8	Caja de endulzante	Cartón	3,6	0,9
9	Cloro	Plástico	3,4	0,9
10	Suavizante	Plástico	3,3	0,8
Tiempo promedio (segundos)			3,5	

3.3.2 Estimación del tiempo de compra promedio a través de la lectura individual de cada producto

Para evaluar el tiempo de compra promedio a través de la lectura individual de los productos se usó una bolsa reutilizable para su almacenamiento. El tiempo de compra está comprendido desde el momento en el que el cliente retira el primer producto de la bolsa y lo lee hasta que se realiza la lectura de los 10 productos uno a uno.

Además, la estimación del tiempo promedio de compra se realizó con base en una muestra de 50 compras. Cabe mencionar que el detalle particular de cada una de las lecturas se adjunta en el Anexo VI. La Tabla 3.3 expone los resultados obtenidos.

Tabla 3.3. Tiempo de compra a través de la lectura individual de cada producto

Compra #	Tiempo (s)	# Compra	Tiempo (s)
Compra 1	34,6	Compra 26	37,7
Compra 2	38,6	Compra 27	38,8
Compra 3	35,1	Compra 28	35,1
Compra 4	33,9	Compra 29	32,8
Compra 5	34,0	Compra 30	28,6
Compra 6	34,8	Compra 31	38,5
Compra 7	32,9	Compra 32	36,5
Compra 8	41,2	Compra 33	33,6
Compra 9	37,4	Compra 34	34,2
Compra 10	33,4	Compra 35	37,1
Compra 11	31,1	Compra 36	33,9
Compra 12	33,8	Compra 37	37,4
Compra 13	32,0	Compra 38	34,2
Compra 14	35,3	Compra 39	35,7
Compra 15	34,0	Compra 40	35,3
Compra 16	34,6	Compra 41	36,7
Compra 17	34,0	Compra 42	33,6
Compra 18	32,7	Compra 43	38,8
Compra 19	31,6	Compra 44	32,4
Compra 20	39,6	Compra 45	38,8
Compra 21	28,6	Compra 46	38,1
Compra 22	34,0	Compra 47	35,4
Compra 23	33,8	Compra 48	37,9
Compra 24	37,5	Compra 49	31,2
Compra 25	33,7	Compra 50	36,4
Promedio		35,0	
Desviación Estándar		2,7	

3.3.3 Estimación del tiempo de compra promedio a través de la lectura simultánea de los productos

Para evaluar el tiempo de compra promedio a través de la lectura simultánea de los productos se usó una bolsa reutilizable, como se evidencia en la Figura 3.41. El tiempo de compra está comprendido desde el momento en el que el cliente toma la bolsa de productos y la pasa por el lector RFID hasta obtener la lectura de los 10 productos dentro de la misma.

Figura 3.41. Lectura simultánea de productos dentro de la bolsa reutilizable

Además, la estimación del tiempo promedio de compra se realizó con base en una muestra de 50 compras. Cabe mencionar que el detalle particular de cada una de las lecturas se adjunta en el Anexo VI. Las Tabla 3.4 y Tabla 3.5 exponen los resultados obtenidos.

Tabla 3.4. Tiempo de compra a través de la lectura simultánea de los productos

#	Tiempo (seg)	#	Tiempo (seg)	#	Tiempo (seg)	#	Tiempo (seg)	#	Tiempo (seg)
1	8,8	11	5,8	21	14,6	31	4,5	41	5,0
2	5,2	12	7,7	22	11,8	32	6,2	42	13,3
3	10,5	13	6,5	23	8,6	33	12,1	43	6,7
4	4,7	14	14,0	24	12,1	34	11,7	44	13,8
5	4,7	15	14,0	25	13,9	35	5,5	45	10,1
6	8,1	16	14,8	26	9,4	36	12,2	46	8,1
7	6,6	17	12,5	27	4,8	37	9,1	47	13,4
8	10,5	18	11,7	28	4,8	38	8,7	48	10,7
9	5,3	19	11,1	29	9,3	39	12,7	49	4,8
10	11,7	20	5,3	30	9,4	40	11,6	50	4,9
Promedio (segundos)					9,3				
Desviación Estándar					3,3				

3.4 Análisis de costos

Al concluir con las pruebas del sistema y corroborar su correcto funcionamiento es necesario el desarrollo de un análisis de costos para la puesta en funcionamiento del sistema prototipo. Dentro del análisis se consideran tanto los componentes de *hardware* como de *software*, que fueron utilizados para el desarrollo del presente proyecto.

3.4.1 Software del prototipo

Para el desarrollo del *software* del sistema prototipo se considera el salario que percibirá cada uno de los programadores en función del mercado laboral actual. Donde la remuneración mensual estipulada para un desarrollador de *software* es de 817 dólares americanos, según la resolución No MRL-2012[25].

En consecuencia, para un tiempo estimado de desarrollo de 6 meses y tomando en cuenta los valores de facturación correspondientes, se obtienen los resultados en la Tabla 3.5.

Tabla 3.5. Costo del *software* del prototipo

Desarrollador	**Salario (USD$)**
Programador 1	5490,24
Programador 2	5490,24
Total	10980,48

3.4.2 Hardware del prototipo

La Tabla 3.6 expone los elementos de *hardware* requeridos para el sistema con sus respectivos valores comerciales.

3.4.3 Costo del prototipo

Con base en los análisis de software y hardware, se presenta un costo referencial de la implementación final del sistema. Dentro del análisis no se contemplan ni el mantenimiento del hardware ni el soporte de actualizaciones de software. La

Tabla 3.7 expone los resultados obtenidos.

[25] Resolución establecida por el Ministerio de Relaciones Laborales del Ecuador para la remuneración de servidores públicos.

Tabla 3.6. Elementos de *hardware* y costo del prototipo

Elemento	Características	Cantidad	Costo ($)
Ordenador de escritorio(Servidor)	Procesador: Core i7 2.4Ghz RAM: 8 GB Disco Duro: 1TB	1	899,00
Lector RFID[26]	Frecuencia de operación:902-928 MHz Interfaz: RS232 Soporte ISO18000-6c Distancia: 10 m	1	209,00
Adaptador USB-DB9	Longitud: 30 Cm	100	20,00
Etiquetas RFID	Banda de Frecuencia: UHF Norma: ISO18000-6c	1	22,00
Subtotal			1150,00
IVA			138,00
Total			1288,00

Tabla 3.7. Costo total de la implementación del sistema

Componente	Costo($)
Hardware	1288,00
Software	10980,48
Total	12268,48

[26] Precio referencial en EEUU.

4. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- El análisis de la fundamentación teórica correspondiente a los sistemas de identificación por radiofrecuencia brindó un panorama esclarecedor relativo a la utilidad de los mismos para las aplicaciones de rastreo y monitoreo de objetos (e.g. suministros de un supermercado). Además, el estudio realizado a la plataforma Android permitió tener una apreciación del gran potencial de esta plataforma para el desarrollo de aplicaciones móviles debido al entorno completo (i.e. *software* y *hardware*) que provee al desarrollador. Por tanto, el sistema prototipo presente fue desarrollado a través de un sistema RFID para la identificación y lectura de productos, mientras que para efectivizar el pago se implementó una aplicación móvil Android.
- Dos aspectos fueron precisados para la obtención de los requerimientos del prototipo. En primer lugar, el análisis de las soluciones comerciales existentes de los supermercados que utilizan sistemas RFID, lo que respaldó la viabilidad del proyecto. Segundo, la definición de las necesidades de los posibles clientes del sistema prototipo obtenidos mediante la realización de encuestas. En consecuencia, la elaboración del proyecto acorde a los requerimientos obtenidos incidió favorablemente en el desarrollo de un modelo alternativo para el pago compras en un supermercado ajustado a las demandas del usuario final.
- La implementación del sistema prototipo incluyó el desarrollo de tres módulos. En primer lugar, la aplicación de la caja permite la lectura de productos mediante el interrogador RFID. Segundo, la aplicación móvil Android hace posible la visualización del detalle de los productos y la realización del pago. Por último, la aplicación web permite al supermercado gestionar la información de sus empleados, productos y clientes. Además, posibilita la automatización del inventario de productos y permite la generación de facturas de compra del cliente y reportes de venta para el supermercado. Por tanto, se determinó que la implementación de los módulos del sistema prototipo proveen de una solución integral para la administración y manejo de un supermercado.
- De acuerdo con los resultados obtenidos en las pruebas realizadas se estimó un tiempo medio de lectura por producto de 3,5 segundos, de manera que se evidenció la utilidad de los sistemas RFID como método alternativo a los códigos de barra para el escaneo de artículos. Sin embargo, es preciso señalar que para la lectura

de productos con envases metálicos fue necesario utilizar una capa de material aislante entre la superficie de contacto y la etiqueta RFID, debido a que el metal es un reflector electromagnético.

- Según los resultados obtenidos en las pruebas propuestas se determinó que el tiempo medio de compra a través del sistema prototipo desarrollado fue de 35 segundos. Sin embargo, se realizaron pruebas fuera del alcance del proyecto donde se determinó el tiempo medio de compra al leer simultáneamente todos los productos de la bolsa, obteniendo un resultado de 9,3 segundos. Por consiguiente, se comprobó la ventaja de emplear un sistema RFID para la lectura simultánea de productos, en lugar de realizar una lectura individual de los mismos. No obstante, es preciso tener en consideración los inconvenientes en la lectura de productos cuando las etiquetas RFID están en contacto unas con otras.

4.2 Recomendaciones

- Se recomienda el uso de servicios web para la implementación de las funcionalidades comunes a la aplicación web ASP.NET y la aplicación móvil Android, con la finalidad de conseguir independencia en cuanto a la plataforma que utilice el cliente.

- La lectura de productos a través de la aplicación de la caja se realizó mediante *triggers* almacenados en el servidor de base de datos. Bajo este enfoque, la limitante que se produjo fue el impedimento del acceso a las tablas que utiliza el *trigger* hasta que su operación no sea finalizada. Por lo tanto, una nueva línea de trabajo podría definir un *socket* para la comunicación entre la aplicación móvil y la aplicación de la caja electrónica.

- El desarrollo de la aplicación móvil para efectuar el pago de las compras se realizó en la plataforma Android. No obstante, futuros trabajos pueden contemplar el desarrollo de una aplicación móvil multiplataforma para tomar en consideración a todo el público posible, i.e. dispositivos iOS, Android o Windows.

- Considerando que las etiquetas RFID pueden ser removidas por el cliente, se recomienda la implementación de un sistema complementario de seguridad (e.g. sensores antirrobo, cámaras IP) para salvaguardar al supermercado de pérdidas potenciales.

- Las interfaces de usuario de la aplicación web fueron implementadas utilizando una plantilla elaborada con los lenguajes de marcado y estilo HTML5 y CCS3

respectivamente, lo que permitió disponer de una aplicación web responsiva e interactiva. A pesar de que el usuario puede utilizar la aplicación web a través de un navegador en su teléfono celular, una futura línea de trabajo puede tomar en consideración la implementación de una aplicación móvil para la administración del supermercado, puesto que permitiría una interacción más amigable con el usuario final.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] B. Skogberg, «Android Application Development», Technology Malmö University, Suecia, 2010.

[2] N. Gandhewar y R. Sheikh, «Google Android: An Emerging Software Platform For Mobile Devices», *IJCSE*, 2010.

[3] R. Meier, *Android 4 Application Development*, 2012.ª ed. Indianapolis, Indiana: John Wiley & Sons, Inc.

[4] «Arquitectura de Android - ApuntesAndroid». [En línea]. Disponible en: https://sites.google.com/site/pruebajoseog/arquitectura-de-android. [Accedido: 29-may-2018].

[5] T. Thuan y L. Hoang, *.NET Framework Essentials*, Segunda. O'Reilly, 2002.

[6] M. MacDonald, *Beginning ASP.NET 4.5 in C#*. Apress.

[7] A. Oppel y R. Sheldon, *SQL A Beginner's Guide*, Tercera. McGraw-Hill, 2009.

[8] J. Groff y P. Weinberg, *SQL: The Complete Reference*, Segunda. McGraw-Hill, 2002.

[9] S. Ahson y M. Ilyas, *RFID Handbook Applications, Technology, Security, and Privacy*. CRC Press, 2008.

[10] D. Hunt, A. Puglia, y M. Puglia, *RFID A Guide To Radio Frequency Identification*. John Wiley & Sons, Inc., 2007.

[11] «RFID Tags». [En línea]. Disponible en: http://www.ravirajtech.com/rfid-tags.html. [Accedido: 24-abr-2018].

[12] J. Joskowicz, «Reglas y Prácticas en eXtreme Programming». 2008.

[13] A. Scrum, «The 2015 State of Scrum Report», Scrum Alliance, 2015.

[14] R. Hernampérez, «Gestiona tus tareas mediante tableros Kanban». [En línea]. Disponible en: https://rafaelhernamperez.wordpress.com/2012/11/16/gestiona-tus-tareas-mediante-tableros-kanban/. [Accedido: 31-may-2018].

[15] «The Industry-first Experimental Demonstration of "Regi-Robo(TM)"», *Panasonic Newsroom Global*. [En línea]. Disponible en: http://news.panasonic.com/global/topics/2017/46190.html. [Accedido: 24-abr-2018].

[16] «Administración de materiales e inventarios Zebra», *Zebra Technologies*. [En línea]. Disponible en: https://www.zebra.com/la/es/solutions/transportation-logistics-solutions/warehouse/inventory-materials-management.html. [Accedido: 31-may-2018].

[17] «RFID Vehicle Tracking Systems». [En línea]. Disponible en: https://www.ramprfid.com/rfid-solutions/rfid-vehicle-tracking/. [Accedido: 31-may-2018].

[18] «ITL Hortofrutícola RFID», *Tag Ingenieros Tecnología RFID*. [En línea]. Disponible en: http://www.tagingenieros.com/RFID-soluciones-RFID/ITL-hortofruticola-RFID. [Accedido: 31-may-2018].

[19] «User Stories: An Agile Introduction», *Agile Modeling*. [En línea]. Disponible en: http://www.agilemodeling.com/artifacts/userStory.htm. [Accedido: 30-abr-2018].

[20] T. Kanban, «Kanban Tool - Kanban Boards for Business», *Increase team performance with a visual project management tool*. [En línea]. Disponible en: https://kanbantool.com/. [Accedido: 01-jun-2018].

6. ANEXOS

Los anexos se incluyen en el CD adjunto al presente documento.

ANEXO I. Respuestas individuales a la encuesta para la obtención de requerimientos.

ANEXO II. *Script* para la creación de la base de datos (i.e. tablas, procedimientos almacenados, *triggers*).

ANEXO III. Código de la aplicación del lector RFID y manual de usuario.

ANEXO IV. Código de la aplicación móvil Android.

ANEXO V. Código de la aplicación web ASP.NET.

ANEXO VI. Pruebas de lectura del prototipo.

ORDEN DE EMPASTADO