UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Aplicaciones móviles para Terminales Punto de Venta en C/C++

INFORME DE ACTIDADES POFESIONALES

Que para obtener el título de

Ingeniero Mecatrónico

P R E S E N T A

Darío Adalid Alvarado Sánchez

ASESOR DE INFORME

M. en I. Francisco Daniel Soria Villegas

Ciudad Universitaria, Cd. Mx., 2019

ÍNDICE

Introducción

En la actualidad resultan cada vez más comunes los pagos electrónicos, ya sean por transferencias, con tarjetas de crédito o débito, e incluso con los teléfonos inteligentes. Es de estas actividades que surge la necesidad de implementar soluciones que ayuden al mejoramiento y seguridad de estas herramientas.

Con respecto a las tarjetas de crédito y débito, su uso se está incrementando de manera acelerada, al grado de que en muchas empresas los sueldos se pagan a través de ellas, y estas mismas tarjetas resultan imprescindibles para adquirir servicios a través de internet.

Nosotros como tarjetahabientes, necesitamos estar seguros con respecto a las transacciones que realizamos, pues estamos expuestos a las operaciones de carácter fraudulento. Por ello requerimos tener en cuenta aspectos básicos para salvaguardar la integridad y seguridad de nuestra información bancaria, esto en conjunto con las medidas adoptadas por los encargados de realizar los cobros.

Por ejemplo, en México existen dispositivos que están certificados por organismos internacionales para manejar de forma segura nuestra información, y con estos podemos realizar transacciones seguras. Por otro lado, en los últimos años el uso de dispositivos no certificados en nuestro país ha aumentado, trayendo como consecuencia el incremento de operaciones fraudulentas y robo de información de los tarjetahabientes. Debido a que la legislación mexicana para regular este tipo de dispositivos aún sigue en discusión, es menester que adoptemos personalmente medidas extraordinarias para proteger nuestra información.

En el desarrollo de este informe, se definen conceptos relacionados con los medios de pago, se explica el proceso de funcionamiento de los mismos medios de pago, se realiza un estudio de caso y al final se adjunta un glosario con acrónimos utilizados a lo largo de este informe.

Para fines prácticos de este trabajo se omitirán nombres reservados de algunas marcas, empresas y partes de código que sean parcialmente mostradas para no atentar contra los derechos de autor.

Los objetivos del trabajo que se presenta a continuación son ofrecerle al cliente una aplicación de propósito general que funcione para su país, ya que en la actualidad no hay una aplicación que funcione de esta manera y soporte cualquier flujo de negocio que los comercios propongan.

La solución realizada es una aplicación de tipo modular que representará segmentos del proceso de una transacción con tarjeta bancaria de chip, para que de esta forma el comercio puede utilizar esta aplicación como base para realizar otras aplicaciones de propósito más específico.

La propuesta del cliente por una aplicación de este tipo nace a partir de que se crean cierto tipo de terminales nuevas que contienen un sistema operativo nuevo basado en Linux cuyos procesos son completamente distintos de los de sistemas operativos antiguos y ninguna otra empresa ofrece una solución de este tipo.

Capítulo 1

Perfil de la empresa

Para el desarrollo de aplicaciones móviles en las terminales punto de venta se requieren ciertos conocimientos, no solo del ámbito de la programación, sino también aquellos referentes a los medios de pago. Paradox, la empresa para la cual laboré por un año y ocho meses, es una empresa que cubre con estos requisitos y colabora en proyectos para diferentes países distribuidos a lo largo del continente americano.

Paradox es una empresa que se dedica principalmente al desarrollo de software para los medios de pago, ya sean estos para clientes directos o a través de un proveedor. La empresa, también da consultoría sobre temas de la misma índole y desarrolla productos propios para brindar servicios de la banca electrónica.

1.1 Medios de pago

Banxico define los medios de pago como cualquier activo que pueda ser usado como dinero; para nuestro propósito, serán aquellas tarjetas emitidas por una institución bancaria, pudiendo ser estas de crédito o débito[1].

1.1.1 Regulación EMV

Las siglas EMV hacen referencia a un estándar de interoperabilidad bancaria de tarjetas con chip integrado ICC y de TPV con soporte y lectura de IC para medios de pago.

[1] Página de internet de Banxico, consultada el 26 de enero de 2019. URL completa en la sección de anexos.

El nombre EMV proviene de tres compañías dedicadas a la interoperabilidad bancaria que desarrollaron, en conjunto, el protocolo: *Europay®, MasterCard® y VISA®*. Dicho estándar pasó a convertirse en el protocolo a nivel mundial para garantizar la interoperabilidad bancaria.

Fig.1.1 Logotipos de las marcas que desarrollaron EMV

La interoperabilidad bancaria es la garantía de que una tarjeta (de crédito o débito) puede ser utilizada como medio de pago alrededor del mundo, sin necesidad de que el banco emisor (el que emite la tarjeta) conozca al resto de los bancos adquirientes (banco que presta sus servicios, ya sean de ATM o cobro en establecimientos) del mundo.

Aunque el protocolo EMV tiene difusión a nivel mundial y requisito para los sistemas TPV, este no funciona para tarjetas de banda magnética; ni para los pagos sin contacto (*Contactless)* que se han venido popularizando en los últimos años.

1.1.2 ISO 8583

El ISO 8583 es el estándar para Transacciones Financieras con Mensajes originados en una tarjeta – Especificaciones de los mensajes de intercambio. Es el estándar de la ISO para sistemas que intercambian transacciones electrónicas realizadas por tarjetahabientes.

Este estándar es el encargado de indicar la forma en la que deben de ser enviados los mensajes, la información que deben de contener en el mismo, así como el procedimiento para envío, recepción y respuesta de los mensajes.

El ISO 8583 trabaja con mapas de bits, donde dichos bits indican campos o subcampos del mensaje y su contenido. A pesar de que los paquetes de información pueden contener información sensible que no es recomendable enviar como texto plano, el ISO 8583 no indica alguna forma de cifrado de la información, delegando esta tarea al banco adquiriente.

1.1.3 ABM

La Asociación de Bancos de México tiene como objetivo representar los intereses generales de la banca mexicana, desempeñándose desde su fundación como el organismo de las instituciones de crédito.

Ha sido moderador principal para el desempeño de medios de pago a nivel nacional y su relación con el estándar EMV e ISO 8583, poniendo de manifiesto las características de seguridad que requiere una aplicación de TPV para cifrar, enviar y recibir llaves; y los tipos de cifrado que se deben de realizar dentro del país.

El capítulo X de la circular única de bancos emitida por la ABM hace referencia al uso del servicio de la banca electrónica. Dicho capítulo es válido únicamente dentro de la república mexicana, por tanto, su implementación radica en las transacciones nacionales.

Un punto importante para la labor que se realiza en los medios de pago, es el **artículo** 316 Bis 6. de la Circular Única de Bancos[2] que escribe.

[2] Circular única de bancos, 2005, p. 237. Ver Anexo A

> “Las instituciones que pongan al alcance de sus Usuarios equipos electrónicos o de telecomunicaciones en sus instalaciones o en áreas de acceso público para el uso del servicio de Banca Electrónica, deberán:
>
> I.-Adoptar medidas que procuren detectar e impedir la instalación en tales equipos, de dispositivos o programas que puedan interferir con el manejo de la información de los Usuarios, o que puedan permitir que dicha información sea leída, copiada, modificada o extraída por terceros [...]”

De conformidad con lo anterior, se debe de mantener siempre un compromiso primordial con la seguridad de la información del tarjetahabiente, ya que, además de las repercusiones que se puedan presentar al mismo, las responsabilidades legales hacia la empresa que desarrolló el software que permitió que dicha información fuera robada por no utilizar los estándares de seguridad necesarios pueden ser de carácter penal.

Aunque, como la mayoría de los países del mundo, México no cumple con todas las reglas estipuladas por EMV, pues ha sufrido una “tropicalización” a causa de algunas circunstancias acontecidas a nivel nacional; el estándar EMV acredita a los bancos pertenecientes a la ABM como instituciones que garantizan la interoperabilidad bancaria.

Fig. 1.2. Logotipo de la ABM

1.2 Descripción de la empresa

La empresa para la que laboré del periodo que comprende del mes de mayo de 2017 al mes de enero de 2019 se llama *Paradox Technologies* y se ubica en la calle de Ladera 7, Hacienda de San Juan, Tlalpan, Ciudad de México CP 14377; la cual cuenta con más de 14 años de experiencia en el sector financiero, teniendo como principales actividades el desarrollo de aplicaciones móviles para TPV, soporte y consultorías relacionadas con el sector de los medios de pago.

Fig. 1.3 Logotipo de Paradox Technologies

1.2.1 Misión

La misión de *Paradox* versa de la manera siguiente: “Brindar soluciones eficientes a empresas del sector financiero (medios de pago) a través del desarrollo de aplicaciones para terminales punto de venta. Contribuyendo en el logro de objetivos de cada uno de nuestros clientes”.

1.2.2 Visión

La visión de *Paradox* versa de la manera siguiente: “Convertirnos en la primera elección de nuestros clientes para dar mantenimiento a los proyectos nuevos o a los ya existentes, siendo líderes en el mercado, convirtiéndonos en la mejor opción de nuestros clientes para lograr posicionarnos como marca a nivel internacional”.

1.2.3 Desarrolladores de aplicaciones

El equipo de programadores (también conocidos como desarrolladores) de la empresa es el pilar principal del desarrollo, mejoras, cambios y asesorías correspondientes al tema de las TPV, esto debido a que son los principales integradores de las diferentes soluciones que ofrece la empresa. Los desarrolladores necesitan tener un perfil ingenieril enfocado a la tecnología, pues necesitan manejar diferentes tipos de equipos.

También se requiere contar con altos conocimientos en programación a bajo nivel (esencialmente C) ya que el manejo de las TPV requiere acceso a hardware a través del software. A su vez los desarrolladores requieren alta flexibilidad y facilidad de aprendizaje puesto que pueden estar envueltos en proyectos que impliquen el uso de diferentes herramientas y lenguajes; mismos cuyo aprendizaje puede ser requerido de manera casi inmediata.

Capítulo 2

Experiencia dentro de la empresa

Mi experiencia dentro de la empresa abarcó el periodo comprendido del mes de mayo de 2017 al mes de enero de 2019 y dentro de este lapso colaboré como desarrollador de aplicaciones móviles en C/C++.

Como parte de mi trabajo participé en distintos proyectos, los cuales en su mayoría fueron desarrollos de aplicaciones móviles y mantenimientos de las mismas aplicaciones para TPV; pero sin limitarme únicamente a estas actividades.

2.1 Estándares de programación orientada a objetos

La POO es un paradigma de programación que tiene como herramienta principal el desarrollo de algunos tipos de datos definidos por el usuario llamados *objetos*. Estos *objetos* contienen ciertas particularidades que nos permiten hacer de la programación una actividad menos genérica y por ende más concreta para los fines que nos exige el mundo real.

Un ejemplo de la manera en cómo podemos crear algunos objetos para que mantengan una relación concreta con el mundo real es la siguiente: el programador (o usuario) puede crear un tipo de *objeto* llamado “casa”, la cual tendrá como “parámetros” algunas de sus características que pueden ser las siguientes:

- Una dirección.
- Número de focos.
- Número de habitantes.
- Metros cuadrados de construcción.

- Metros cuadrados de terreno.
- Si es casa propia o rentada.
- Vecino de la izquierda.
- Vecino de la derecha.

A pesar de que es claro que en ningún lenguaje de programación contamos con un tipo de dato llamado “Dirección”, podemos guardar dicho valor en una variable cuyo tipo de dato sea *string* (esto en C++), cuyo nombre sea “Dirección” y así sucesivamente cada característica del tipo de dato “casa” puede ser almacenado en tipos de datos base, quedando en el lenguaje C++ algo de la siguiente forma:

```
class Casa
{
	std::string stDireccion;
	int inFocos;
	int inHabitantes;
	double dbConstruccion;
	double dbTerreno;
	bool blPropia;
	Casa* VecinoIzq;
	Casa* VecinoDer;
};
```

Y esto es que acabamos de crear es una representación de un ente del mundo exterior como un *objeto* dentro de nuestro programa. Pero además de cada una de las propiedades que contenga el *objeto*, este puede contener algo llamado *métodos* que son básicamente aquellas operaciones que puede realizar el mismo objeto. Por ejemplo, una operación “setHabitantes” que le permita a un objeto de tipo casa cambiar el número de habitantes, siguiendo el ejemplo, nuestro objeto de tipo casa tendría la siguiente forma:

```
class Casa
{
	std::string stDireccion;
	int inFocos;
	int inHabitantes;
	double dbConstruccion;
	double dbTerreno;
```

```
        bool blPropia;
        Casa* VecinoIzq;
        Casa* VecinoDer;

        void setHabitantes(int habitantes)
        {
                this->inHabitantes = habitantes;
        }
};
```

Estas son algunas características básicas que comprende la programación orientada a objetos, pero existen algunas otras fundamentales del paradigma de programación orientada a objetos que se detallaran a continuación utilizando el ejemplo de nuestro objeto tipo casa creado arriba.

Otro concepto importante dentro de la programación orientada a objetos es la herencia de clases. Este concepto se refiere a la cualidad de poder adquirir los mismos atributos y métodos de otra clase llamada *base* hacia alguna otra clase llamada *hija*.

Por ejemplo, tomando otros tipos de datos análogos que quisiéramos crear en nuestro programa, podríamos querer crear un tipo llamado “Mansion” y otro tipo llamado “CasaDeCampo”, donde ambos tipos comparten características fundamentales con la clase base “Casa” y además incluyen algunas otras propiedades o métodos extras que los diferencian entre sí y con respecto a la clase base.

Suponiendo que la clase “Mansion” incluye además el número de espacios para aparcar automóviles y el número de cámaras de seguridad. El ejemplo de este tipo de dato quedaría (en C++) de la siguiente manera:

```
class Mansion:Casa
{
        int inEspaciosAparcar;
        int inCamarasSeguridad;
};
```

De esta forma es como la clase “Mansion” logra heredar los atributos de la clase “Casa”. Lo cual es lo mismo que la clase “Mansion” por ser hija de “Casa” contenga los mismos atributos que esta tiene (número de focos, vecinos, etc.).

Por otro lado, la clase “CasaDeCampo” puede tener parámetros distintos a “Mansion” pero sigue compartiendo las características base de la clase “Casa”. Por ejemplo, puede contener propiedades como el tamaño del tanque de agua, y los litros de agua que quedan en él. Sea así que la clase quedaría de la siguiente forma:

```
class CasaDeCampo :Casa
{
	double dbTamañoTanque;
	double dbLitrosRestantes;
};
```

2.1.1 Conocimientos previos

Cuando ingresé a Paradox, contaba con los conocimientos básicos de programación, esencialmente aquellos adquiridos en mis clases de “Programación para Ingenieros” y “Técnicas de Programación”, siendo de esta última donde aprendí los conceptos fundamentales de la POO detallados arriba.

También contaba con conocimientos adquiridos en diferentes materias de la carrera que no están relacionadas directamente con la programación, por ejemplo, de la materia de “Circuitos Digitales” aprendí la importancia de los registros en la memoria, programar componentes a bajo nivel y a trabajar con datos a nivel de *bits* y *bytes.*

En la materia de “Instrumentación” adquirí conocimientos relacionados al acondicionamiento de señales de entrada, las propiedades de algunos sistemas para el intercambio de información, siendo estos de utilidad al momento de desarrollar las aplicaciones móviles para el envío de paquetes de información y recepción de mensajes; también las estructuras de los circuitos cuyas analogías fueron de utilidad al momento de conocer diferentes componentes de las TPV.

Otra materia cuyos conocimientos me sirvieron como base al entrar a Paradox fue “Sistemas Embebidos” donde aprendí a trabajar con redes de información, enviar

paquetes de datos por medio de diferentes tipos de comunicaciones, y algunos parámetros, y su significado dentro de las telecomunicaciones.

2.1.2 Conocimientos generados

Si bien, la cantidad de conocimientos generados en la empresa, fue amplia y en diferentes ámbitos, tanto de índole personal como profesional, a continuación, procedo a ahondar acerca de aquellos más significativos con referencia al ámbito profesional y a detallar acerca de las herramientas que aprendí a utilizar.

Los conocimientos los enumeraré en 3 categorías que detallaré conforme vayan apareciendo; al final de estos, describiré brevemente las herramientas utilizadas.

La primera categoría de los conocimientos, es el aprender a usar compiladores con licencia. Ya que estos necesitan estar enlazados con una licencia original para su correcto funcionamiento, además del hecho de que me enfrenté con la cantidad limitada de información que contenían los manuales de los mismos.

Con respecto a los manuales, como estudiante estaba acostumbrado a trabajar con programas y herramientas ampliamente conocidos y de los cuales existían manuales y libros de diferentes autores; de esta forma si había alguna fuente de información que no me resultara de utilidad o sin la información que yo requería, podía acceder a una fuente distinta a dónde buscar la información requerida.

Lo anterior, no sucede en el ámbito profesional, pues muchas veces las herramientas son desarrolladas para un tipo de tecnología en particular, como lo son las TPV de cada uno de los fabricantes. Una marca en particular, *Verifone®,* tiene su propio compilador, y con él deben de ser compilados los proyectos para cierto tipo de sus terminales.

De esta forma, la única fuente de información y documentación proporcionada, es la liberada por la misma marca. Por ende, tuve que aprender a leer con mayor precaución y detenimiento los manuales, para realizar las operaciones en la forma en cómo estaban detalladas. Desafortunadamente los manuales están hechos por seres humanos, y todos los seres humanos somos susceptibles de errores.

En una ocasión cuando realicé algunos procedimientos de compilación, tal cual se indicaba en los manuales, me percaté de que esos procedimientos no resultaban en la salida esperada. Lo anterior debido a que había cambios en las versiones del compilador, pero no se habían actualizado los manuales del mismo. Por ende, aprendí a diferenciar entre cuando había un procedimiento mal descrito en el manual, y cuando yo había realizado el procedimiento de manera inadecuada; esto con la finalidad de acudir por ayuda con el equipo de soporte técnico, cuando fuera necesario.

La segunda categoría de conocimientos es el desarrollo apropiado de manuales y documentación para los cambios realizados. Esto con el fin de mantener el orden y funcionalidad de los proyectos. Me sucedió que, al retomar un proyecto de algún otro desarrollador, me encontré con falta de documentación y manuales para entender el funcionamiento del mismo proyecto.

Un proyecto promedio consta alrededor de 50 archivos diferentes, los cuales incluyen, pero no se limitan a: archivos fuentes en diferentes lenguajes de programación, archivos de configuración para la aplicación, *scripts* de compilación, bibliotecas estáticas, bibliotecas dinámicas. Y de los archivos con código, el número de líneas de código varía desde las 2 000 hasta las 15 000. Por estas razones resulta necesario contar con la documentación adecuada del proyecto, para evitar releer hasta 150 000 líneas de código.

Dicho lo anterior, resultan más claras las razones por las cuales dediqué una cantidad considerable de tiempo en redactar manuales y documentación apropiada que sirviera de guía para el desarrollador futuro del proyecto y para los usuarios finales

que iban a ser los encargados de utilizar la aplicación final (cajeros, tenderos, administradores de tienda, etc.).

La tercera categoría de conocimiento, fue el aprender a utilizar metodologías ágiles para el desarrollo de software junto con el uso de herramientas que dicha metodología conlleva. La metodología ágil con la que tuve contacto, es una llamada *Scrum*, cuya idea es dividir el proyecto en pequeñas tareas, e ir realizando dichas tareas una por una de manera secuencial; y con ayuda de un administrador (o software) revisar diario el estado de cada una de ellas; esto es, si ya se encontraban finalizadas, pendientes, en proceso.

Con respecto a las herramientas que me parecieron de mayor utilidad, es el uso de FVCS que en conjunto con un repositorio permiten a diferentes desarrolladores trabajar en un mismo proyecto en tiempo real. De esta forma, en equipo, logramos trabajar con un proyecto en conjunto y evitamos tiempos muertos en compartirnos nuestros avances, y revisar que no hubiésemos modificado las mismas partes de manera distinta. Además de que el uso de estas herramientas (FVCS y repositorios) son conocimientos ampliamente solicitados en la industria del desarrollo de software.

Otra herramienta que también forma parte de las metodologías ágiles, son los gestionadores de tareas que sirven como canal para informar entre las distintas áreas (como pueden ser el área de desarrollo, de control de calidad, administrativa, finanzas, ventas) sobre las etapas del proceso, el reporte de *bugs*, la etapa de las pruebas en las que se encuentra, etcétera.

Finalmente, aprendí a trabajar realmente en equipo, durante mi trayectoria escolar colaboré con distintos tipos de compañeros e hicimos trabajos en conjunto, pero fue hasta que laboré a nivel profesional que entendí lo que realmente significa trabajar en equipo.

Trabajar en equipo no significa que todos hagan cantidades iguales de trabajo, sino que cada uno realice las tareas que le fueron asignadas, pero manteniendo un estándar de calidad común con el resto de los compañeros y sobretodo un buen canal de comunicación para entender en qué afecta el trabajo a otros miembros del equipo. También significa saber colaborar cuando una tarea requiere a varios miembros del equipo, saber comunicar los conocimientos que uno posee, también las dudas que a uno lo aquejan. Tuve que aprender a comunicar mis opiniones a personas que no conocían temas de programación y a atender las dudas y expectativas que pudieron tener con respecto a mi trabajo.

2.1.3 Buenas prácticas en la programación

Como el desarrollo de aplicaciones móviles comenzó cuando aún no eran muy difundidas las buenas prácticas en la programación, estos cuentan con malas prácticas que, aunque no influyen directamente sobre el funcionamiento del negocio en sí, sí afectan el óptimo desempeño a bajo nivel (velocidad de las operaciones) y sobre todo a las dificultades que conllevan al momento de darle mantenimiento al código (incluir nuevas funcionalidades, resolver problemas nuevos, remover funcionalidades obsoletas), por esto, la importancia de mantener buenas prácticas al momento de escribir código nuevo y mantener el código viejo.

Estas malas prácticas de programación dentro de las aplicaciones móviles con los que me encontré durante mi trabajo profesional, representaron dificultades para desempeñar mi trabajo de manera apropiada, ya que, al momento de implementar nuevas soluciones, me encontraba con excepciones dentro del código que tenían como resultado un mayor tiempo para entender dichas excepciones del flujo y acondicionarlas de manera apropiada.

Básicamente las buenas prácticas de código que encontré a lo largo de mi experiencia en Paradox son las siguientes:

- Nombres: Se tiene que tener consistencia al nombrar *variables*, tipos de datos, *objetos*, *funciones* e incluso archivos. Tienen que ser nombrados de forma que no resulten abstractos al programador, se recomienda el uso de sustantivos. A su vez con respecto al estilo se recomienda utilizar convenciones previamente desarrolladas, como lo pueden ser *pascal case, camel case* o *hungarian notation.*
- Uso de funciones y/o métodos: Se recomienda que ninguna función (en medida de lo posible) tenga una longitud mayor a una pantalla de monitor, esto es aproximadamente entre 40 y 80 líneas; en caso de tener una longitud mayor, se recomienda el uso de funciones o métodos de la clase para ejecutar los procedimientos.
- Sangría: Los niveles de sangrado (*identation* en inglés) tienen que estar correctamente definidos, se recomienda utilizar entre 3 y 4 espacios en blanco para cada nivel. Así mismo es recomendable que si una función o método, contiene más de tres niveles de sangrado, es posible recurrir a otra función o método.
- Longitud de líneas: cada línea de código debe contener a lo mucho una sentencia para facilitar su lectura, tampoco ninguna línea debe sobrepasar la longitud de un monitor, esto es aproximadamente 100 caracteres. Si una función o método requiere muchos parámetros, se pueden utilizar objetos como parámetros, en vez de parámetros sueltos.
- Comentarios: Se recomienda que los comentarios no estén en la misma línea que el código, que agreguen información no trivial, que terminen con un punto y que cumplan con el resto de las convenciones.
- Métodos set y get: Se recomienda que las propiedades de un objeto se pongan como privados, esto es que nadie más que él mismo objeto tenga acceso a ellos; y que, en caso de ser requeridos por algún otro objeto, existan métodos *set* y *get* para asignar un valor u obtenerlo según se requiera.

En el desarrollo de proyectos y mantenimiento de los mismos, intenté adaptarme a los distintos estilos de programación que se utilizaban, en un par de casos resultó

sencillo, principalmente en nuevos proyectos, pues las buenas prácticas iniciaban junto con el mismo proyecto, haciendo más fácil detectar las malas prácticas y corregirlas. En cambio, en proyectos antiguos resultaba más complicado implementar las buenas prácticas de programación, y en un caso en particular me resultó imposible implementarlas.

2.2 Iniciación en las aplicaciones

Durante mi segunda semana de haber ingresado a *Paradox* me asignaron como aprendiz con un compañero que se encargó de supervisar mis primeros desarrollos, con el apoyo de él tuve mi primer acercamiento a las terminales.

Aprendí de esta manera a llevar a cabo el proceso de compilación para los proyectos de las TPV, a empaquetarlos para lograr cargarlos a la terminal, a realizar un proceso de firmado del aplicativo (para autenticar la aplicación final con la misma terminal) y a probar el aplicativo ya instalado dentro de la terminal.

El proceso duró 4 semanas, periodo durante el cual mi compañero corrigió algunos errores de programación que yo tenía, también se encargó de mostrarme algunos consejos al momento de implementar las soluciones para las TPV.

2.2.1 Documentación particular de las marcas

En el mercado de las TPV existen diferentes marcas, las dos principales a nivel mundial son *Verifone*® e *Ingenico*®. Cada una de ellas cuenta con modelos propios de sus terminales, sus propios sistemas operativos, sus compiladores, sus modos de autenticación y la forma en la que se controlan los componentes de sus terminales.

Por componentes se deben de entender los diferentes dispositivos que tienen asignados una tarea en particular, pudiendo ser estos componentes hardware que

cumple una función concreta para el sistema completo, o algún software que se encuentre dentro de la misma terminal.

La marca que trabajé en la mayoría de mis proyectos fue *Verifone*®, razón por la cual me familiaricé con sus terminales, documentación, sistemas operativos, procesos y estructuras.

Verifone® presenta unos componentes dentro de sus terminales que se encuentran de alguna manera ligados al SO de cada TPV, esto es que fungen como administradores de algunos procesos internos dentro de la aplicación o dentro de la terminal. Por ejemplo, hay un controlador que se encarga de gestionar las comunicaciones a nivel usuario, dicho de otro modo, existe un asistente que permite al usuario configurar la red a la cual se conectará la terminal; aunque, la terminal esté conectada a una red en específico, aún es necesario que el desarrollador garantice que la aplicación sea capaz de leer dicha configuración para acceder a otro dispositivo a través de esa red.

La misma marca maneja la mayoría de los componentes de sus terminales como archivos, siendo de esta forma relativamente fácil para el desarrollador hacer uso de algunos de ellos como lo son los lectores de tarjetas, el teclado y la pantalla, ya que el proceso resulta, en su mayoría, similar al de escribir o leer de un archivo: crear un *objeto* del tipo del componente, abrir el componente, escribir/leer del componente y cerrar el componente.

Pese a que la documentación de los componentes de las terminales, resulta dispersa y a veces desactualizada, la mayoría de ella la encontré concentrada en carpetas que resultaban relativamente obvias con respecto a la posición de los componentes. Toda su documentación se encuentra en idioma inglés, alguna de esta información resultaba de bastante utilidad si se leía con la precaución suficiente, mientras que otra resultaba con casi nada de información.

2.2.2 Acercamiento con las aplicaciones

Mi primer acercamiento con una aplicación de TPV fue para un banco mexicano, que por razones de confidencialidad nombraré el “Banco A”.

Este Banco tiene una amplia gama de giros de comercio, que van desde hoteles, restaurantes, comercios al por menor, hasta algunos otros que cuenten con su propio sistema de cobros, pero requieran el uso de una TPV certificada para realizar el cobro. Por estas razones el Banco A contaba con un aplicativo que era configurable a través de un archivo y de algunas variables almacenadas dentro de la misma terminal llamadas “variables de entorno”.

Las modalidades en las que podía funcionar este aplicativo son llamadas “*standalone*” y “*pinpad”*, siendo la modalidad “*standalone”* aquella en la que la terminal puede efectuar transacciones completas sin requerir otro dispositivo que funcione como ECR (como en los restaurantes); mientras que en la modalidad “*pinpad”* la TPV requería de otro dispositivo que le mandara las peticiones para realizar el cobro (como en los supermercados).

Además, la modalidad “*standalone”* tenía tres formas de trabajar: “restaurante”, “hotel” y “*retail”*. Para la modalidad restaurante, algunas de sus características son las siguientes: cada venta podía ser cargada con una propina y ser asignada a un mesero; se puede dejar la línea de propina abierta; se pueden realizar devoluciones; se pueden generar reportes por mesero; entre otras.

Para la modalidad “hotel” algunas de sus características son las siguientes: se pueden hacer autorizaciones (*check-in)* a la llegada del cliente al hotel; se pueden agregar cargos a una cuenta; se puede hacer un cierre de autorización (*check-out)*; se pueden generar reportes de ventas y ver las autorizaciones pendientes; entre otros.

La modalidad "*retail"* presenta únicamente ventas, devoluciones, cancelaciones y generación de reportes sencillos.

Durante aproximadamente 10 meses que trabajé con el Banco A me familiaricé con este aplicativo, manejé todo tipo de componentes de la terminal para modificar su funcionamiento a solicitud del mismo Banco, conocí los detalles de configuración del aplicativo, e incluso acudí en un par de ocasiones a instalar el aplicativo directamente en un par de hoteles.

2.2.3 Experiencia con los tipos de comunicación

Las particularidades que me resultó de mayor dificultad en mi trabajo en Paradox fue en manejo de las comunicaciones. Las TPV tienen diferentes tipos de comunicación, estas pueden ser vía wifi, ethernet, *Bluetooth*®, 3G, serial, USB; además de que puede ser la comunicación cifrada vía TLS, SSL o algún protocolo propio, o simplemente en texto plano. Cabe mencionar que esta comunicación es distinta a la encriptación demandada por el protocolo EMV para información sensible.

Por la variedad de tipos de comunicación que pueden existir para los distintos tipos de TPV me resultó complicado configurarlas todas, puesto que el cliente espera que el técnico que instale las TPV en el comercio pueda configurar la red (del tipo que sea) y simplemente iniciar la operación. Esta petición me resultaba en tener que realizar las configuraciones de cada tipo de comunicación con parámetros que no existían al momento de programarla, pero realizando siempre la validación de que tanto el SO como el hardware de la terminal soportaran la comunicación deseada. Esto porque se daban casos de TPV que no soportan ciertos tipos de comunicaciones.

Trabajar con estas complicaciones propició que aprendiera acerca de algunos parámetros de las redes de comunicaciones, los procesos que se llevan a cabo dentro de cada una de ellas y de las maneras que pueden conectarse entre ellas. Por ejemplo, una comunicación vía wifi puede necesitar contraseña o no, y esta contraseña puede ser de distintos tipos (WEP, WPA, WPA2); además de que la

dirección asignada puede ser hecha de manera dinámica o estática, y en caso de ser estática requiere parámetros adicionales (dirección IP, DNS1, DNS2, puerto).

2.2.4 Forma de generar documentación

Como comenté anteriormente, el generar documentación es un proceso importante dentro de la industria de desarrollo de software. Para esto me apoyaba de los documentos que existían con anterioridad (aunque pertenecieran a proyectos distintos) y copiaba el formato, para generar documentación consistente dentro de la empresa.

Si bien la mayoría de la documentación se generaba independiente del código fuente; como buena práctica de programación, yo personalmente incluía una breve descripción de las funciones, métodos e incluso objetos que yo desarrollé. Esta práctica también la lleve a cabo con funciones casi criptográficas que yo no desarrollé pero que por necesidad requerí entender de su funcionamiento, esto con la idea de que en algún momento en el futuro otro desarrollador necesitaría entender dicho funcionamiento.

Como desarrollador generé documentación de los procesos que se llevaban a cabo dentro de la aplicación, el encargado de generar la documentación final era el equipo de QA, quienes eran los encargados de llevar a cabo las pruebas que podrían hacer fallar el aplicativo y reportarlas a los desarrolladores, y de generar la documentación que se le entregaría al cliente final.

2.3 Curva de aprendizaje

Los humanos, al comenzar a desempeñarse en una actividad nueva transitamos por algo llamado curva de aprendizaje. Dicha curva nos explica que cuanto menor conocimiento tengamos acerca de un tema, el tiempo invertido para aprender algo nuevo será relativamente corto en comparación con el conocimiento

adquirido; por el contrario, cuanto más conocimiento tengamos de un tema, más tiempo tardaremos en ahondar los conocimientos y por ende en generar nuevos de estos.

Vale la pena remarcar, que siempre aprendemos cosas nuevas rápidamente, y cuando las estamos aprendiendo, seremos más lentos y cometeremos muchos errores por encontrarnos en nuestro proceso de aprendizaje.

Para mi caso particular, el desarrollo de mi curva de aprendizaje fue como se muestra en la Fig. 2.1, donde tardé 6 meses en alcanzar un nivel de desarrollo de aplicaciones móviles que me permitía trabajar a un buen ritmo, contando con conocimientos y uso de herramientas que me permitían desarrollar mis actividades de forma apropiada.

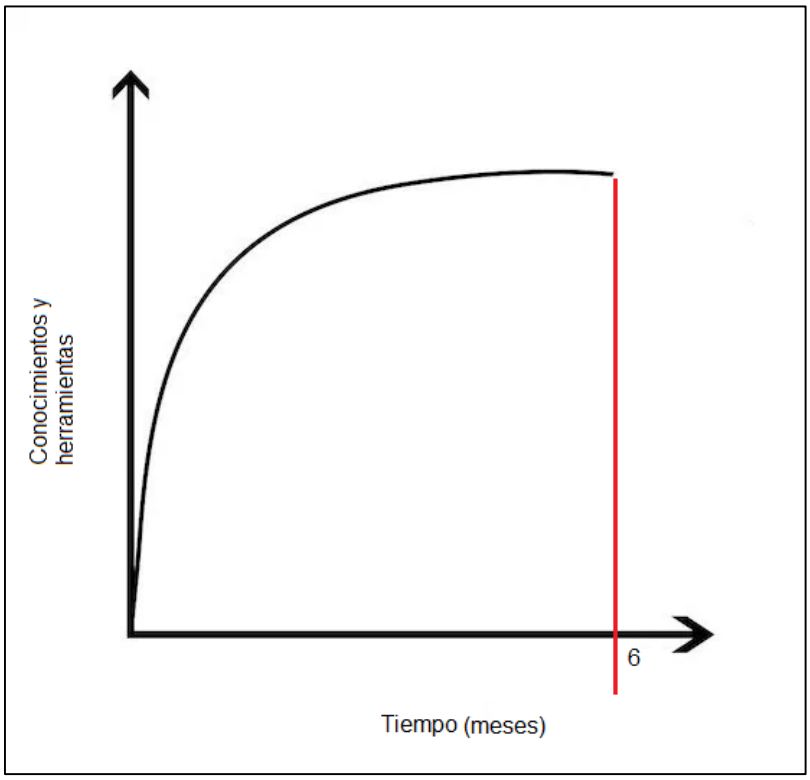

Fig. 2.1 Curva de aprendizaje

Capítulo 3

Estudio de caso de una aplicación móvil

En esta sección trataré lo relacionado a un caso de estudio correspondiente a un cliente en un país al cual me referiré como “País P”. Por razones de propiedad intelectual me referiré a este cliente como “Cliente B” y los códigos mostrados a continuación fueron modificados para fines demostrativos únicamente.

Este proyecto tuvo la particularidad de que el Cliente B no quería una aplicación final desarrollada[3], sino un tipo de interfaz que le facilitara el uso de las TPV ya que en el País P el mismo cliente se encarga de prestar servicios referentes a muchos medios de pago que le dificultan mantener un aplicativo universal. Por lo mismo le es más fácil mantener un tipo de interfaz genérica que le permita manipular de manera directa y efectiva los componentes de las TPV como cada negocio lo requiera, dicho modelo se puede observar en la fig. 3.1.

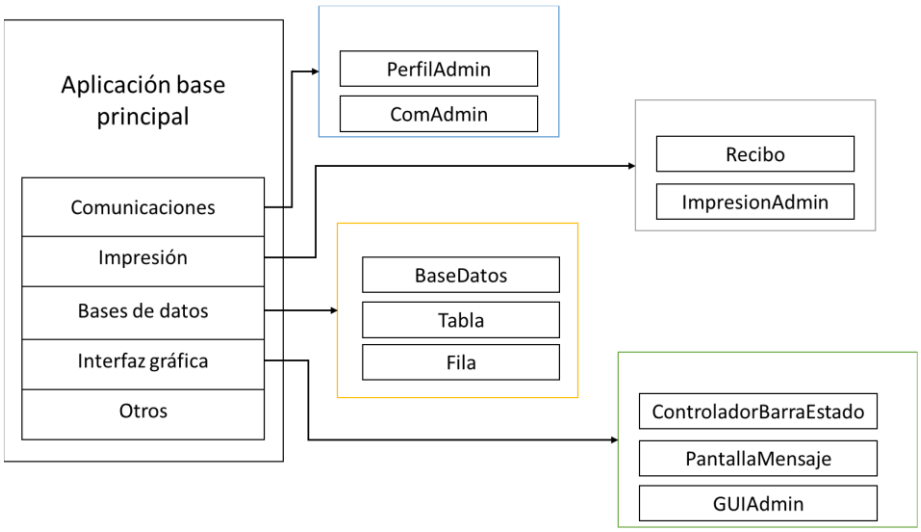

Fig 3.1Diagrama general de la aplicación

[3] Ver Anexo B

3.1 Módulo de comunicaciones

El primer módulo que voy a tratar es el módulo de las comunicaciones, este es el módulo encargado de gestionar lo referente a los medios de comunicación que pueden presentarse en la TPV.

Para hacer el módulo encargado de las comunicaciones se pensó en un par de clases. La primera que fuera la encargada de administrar las comunicaciones, llamada “ComAdmin” y una segunda clase que se encargara de administrar los perfiles de comunicaciones llamada “PerfilAdmin”.

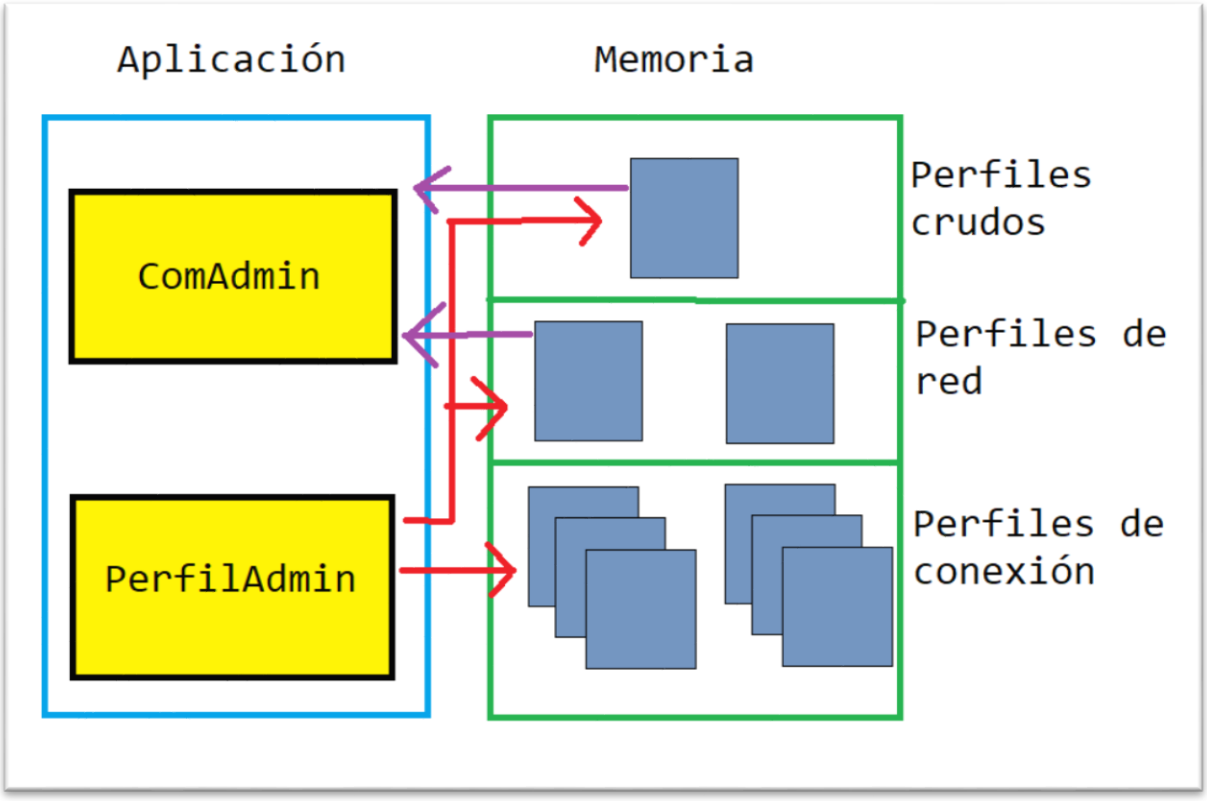

Fig. 3.2 Esquema del módulo de comunicaciones

La diferencia entre el “ComAdmin” y el “PerfilAdmin” radica en que el primero se encarga de tratar a la comunicación (sin importarle el medio por el cual esté conectado) como un archivo, ya que se tiene que inicializar, abrir, conectar, enviar, recibir y cerrar el canal de comunicación. Mientras que la segunda se encarga de crear los perfiles de la red, esto es, de enviarle a la TPV los parámetros necesarios para garantizar que una conexión pueda ser establecida; y este último solo puede ser instanciado (creado/utilizado) por el primero.

La clase “PerfilAdmin”:

```
class PerfilAdmin {
public:
	void SetConnTimeOut(int newConnTimeOut);
	void SetRespTimeOut(int newAnswerTimeOut);

	int CrearPerfilRed(COMM_MEDIA media, std::string name);

	int CrearPerfilRed(std::string media,
		std::string name,
		bool autoStart,
		int dhcpEnabled,
		std::string localIP,
		std::string netmask,
		std::string gateway,
		std::string dns1,
		std::string dns2,
		bool preConfig);

	int CrearPerfilRed(std::string media,
		std::string name,
		bool autoStart,
		std::string gprsAuth,
		std::string gprsAPN,
		std::string gprsUser,
		std::string gprsPWD,
		std::string gprsPIN,
		int SIMSlot,
		bool preConfig);

	int CrearPerfilRed(std::string media,
		std::string name,
		bool autoStart,
		int dhcpEnabled,
		std::string wifissid,
		std::string wifikey_mgmt,
		std::string wifipsk,
		bool preConfig,
		WLAN_Node* stWlanNode);

	int CrearPerfilConexion(std::string connProfName,
```

```
                std::string ip,
                unsigned short port,
                std::string protocol,
                std::string caCertFile,
                std::string protVersion,
                std::string cipherList,
                int policy,
                bool comboWiFiGPRS);

        int CrearDialCruda(std::string connProfName,
                std::string hostDialNum,
                bool sync = 0,
                std::string dialPrefix = "",
                std::string dialType = "TONE",
                bool blindDial = 0,
                bool defaultParams = true);

        void setBaudRate(unsigned int newBaudRate) { usBaudRate = newBaudRate; };
        void setPariedad(std::string newParity) { stParity = newParity; };
        void setDataBits(unsigned int newDataBits) { usDataBits = newDataBits; };
        void setStopBits(unsigned int newStopBits) { usStopBits = newStopBits; };

private:
        PerfilAdmin();

        friend class ComAdmin;

        static COMM_MEDIA currentMedia;
        int inTipoRed;
        unsigned short usBaudRate;
        std::string stParity;
        unsigned short usDataBits;
        unsigned short usStopBits;
};
```

Se puede observar a primera instancia que la clase “PerfilAdmin” contiene varios métodos sobrecargados, esto es que tienen el mismo nombre; en concreto el método llamado “CrearPerfilRed”, pues este método es el encargado de crear un archivo que se guardará dentro de la misma TPV que contenga los parámetros para realizar una conexión. Dependiendo de los datos ingresados será el tipo de archivo para una conexión específica que se guardará.

Por ejemplo, el método siguiente, se encarga de crear un archivo con las configuraciones para una conexión GPRS, los parámetros necesarios para la conexión son los que el método recibe como parámetros de entrada. Cabe resaltar que esta clase únicamente se encarga de generar un archivo con toda la información

necesaria para que sea la clase “ComAdmin” la que se encargue de realizar la conexión:

```
int CrearPerfilRed(std::string media,
	std::string name,
	bool autoStart,
	std::string gprsAuth,
	std::string gprsAPN,
	std::string gprsUser,
	std::string gprsPWD,
	std::string gprsPIN,
	int SIMSlot,
	bool preConfig);
```

Cuando los parámetros para la red se encuentran listos, la misma clase puede ejecutar otro método que se encarga de crear el perfil de conexión:

```
int CrearPerfilConexion(std::string connProfName,
	std::string ip,
	unsigned short port,
	std::string protocol,
	std::string caCertFile,
	std::string protVersion,
	std::string cipherList,
	int policy,
	bool comboWiFiGPRS);
```

Este perfil requiere otros parámetros que serán independientes del tipo de conexión, por ejemplo, la TPV siempre intentará conectarse a la misma dirección IP (que será la dirección del servidor a donde se dirigen los cobros) independientemente de si está conectado vía Wifi, ethernet o GRPS. Los parámetros: “protocol”, “caCertFile”, “protVersion”, “cipherList” y “policy” son únicamente requeridos si la conexión llevará algún tipo de seguridad especial (TLS o SSL).

El tipo de conexión por cable dial es un caso particular, pues no requiere parámetros de configuración para el perfil de red (pues solo existe un cable telefónico al que se puede conectar; dicho de otra manera, solo hay una línea telefónica por cable). Por esta razón si se desea hacer la conexión por este medio, es necesario establecer primero algunos valores para el intercambio de información:

```
void setBaudRate(newBaudRate);
void setPariedad(newParity);
void setDataBits(newDataBits);
void setStopBits(newStopBits);
```

donde cada parámetro estará almacenado como valor privado dentro de la misma clase (para que nadie más pueda modificarlo sin saberlo). Posterior a que los parámetros de intercambio de información están configurados apropiadamente, se puede proceder a crear el perfil de conexión llamado crudo (en inglés *raw*).

```
int CrearDialCruda(std::string connProfName,
	std::string hostDialNum,
	bool sync = 0,
	std::string dialPrefix = "",
	std::string dialType = "TONE",
	bool blindDial = 0,
	bool defaultParams = true);
```

Una vez que los perfiles de conexión han sido creados por el objeto de tipo “PerfilAdmin”, el objeto de tipo “ComAdmin” puede proceder a realizar las acciones que le correspondan para el envío/recepción de mensajes sin importar su índole. El código de la clase a continuación:

```
class ComAdmin {
public:
	ComAdmin();

	int init();
	int abrir();
	int cerrar();

	int conectar(std::string connProfName, bool sync);

	int conectar(std::string connProfName,
		bool sync,
		std::string ip,
		unsigned short port,
		std::string protocol,
		std::string caCertFile,
		std::string protVersion,
		std::string cipherList,
		int policy);
```

```
        int conectar(std::string ip,
                unsigned short port,
                std::string protocol,
                bool sync = false,
                std::string caCertFile = "",
                std::string protVersion = "",
                std::string cipherList = "",
                int policy = -1);

        int desconectar();
        int enviar(const void *data, size_t len);
        int recibir(void *data, size_t maxLen, size_t *bytesRead);
        int reiniciarRed(char *NPFileName);

        void habilitarDatosPreconfig() { usarDatosPreconfig = true; };
        void deshabilitarDatosPreconfig() { usarDatosPreconfig = false; };

        PerfilAdmin managerProfile;

private:

        bool usarDatosPreconfig;
};
```

El primer paso es indicarle a la TPV que se utilizaran los recursos de hardware necesarios por ello se “encienden” con el método `init()` del “ComAdmin”, para posteriormente abrir la conexión con el método `abrir()`.

Posterior a haber inicializado el hardware y abierto la comunicación con los métodos arriba descritos, debe de conectarse con la red, para ello la misma clase cuenta con tres métodos sobrecargados que podrá utilizar dependiendo de los parámetros que se utilicen. Por ejemplo, podrá utilizar el siguiente método, para realizar la conexión con un perfil de conexión ya establecido previamente por el “PerfilAdmin”.

```
        int conectar(std::string connProfName,
                bool sync,
                std::string ip,
                unsigned short port,
                std::string protocol,
                std::string caCertFile,
                std::string protVersion,
                std::string cipherList,
                int policy);
```

Con la conexión levantada, se puede proceder a enviar o recibir mensajes según se requiera con los métodos siguientes:

```
int enviar(const void *data, size_t len);
int recibir(void *data, size_t maxLen, size_t *bytesRead);
```

donde el parámetro *data* hace referencia a una cadena de texto a ser enviada o recibida; "len" y "maxLen" hacen referencia a la longitud del texto, el primero es exacto para enviar el mensaje, mientras que el segundo es el tamaño máximo de respuesta que se espera recibir. Finalmente, el parámetro "bytesRead" es un apuntador que indica el tamaño real del paquete recibido con el fin de garantizar la autenticidad e integridad del mensaje.

El mismo "ComAdmin" es capaz de habilitar o deshabilitar los datos pre configurados con los métodos siguientes:

```
void habilitarDatosPreconfig();
void deshabilitarDatosPreconfig();
```

También es capaz de reiniciar la red en caso de que se haya perdido la conexión:

```
int reiniciarRed(char *NPFileName);
```

Si el usuario ha finalizado el envío y recepción de paquetes puede desconectarse de la red actual con el método:

```
int desconectar();
```

Si el usuario desea cerrar la aplicación, esto es, que ya no se requiera más el uso de envío y/o recepción; o requiera ahorrar batería, puede "apagar" el hardware encargado de las comunicaciones con el método:

```
int cerrar();
```

3.2 Módulo de impresión.

El módulo de impresión es el encargado de, como su nombre lo indica, imprimir los recibos, cuyas partes se muestran en la Fig. 3.3, en donde se observa de color rojo la cabecera, de color verde el cuerpo, y de azul el fondo o pie del recibo. Este módulo está conformado por dos clases básicamente, estás clases son “Recibo” e “ImpresionAdmin”. La primera de ellas es una clase que representa a un objeto de tipo recibo que contiene esencialmente 3 parámetros que son:

- La cabecera del recibo.
- El cuerpo del recibo.
- El pie de página (o fondo) del recibo.

Fig. 3.3. Imagen ejemplo de un recibo.

Cada una de las partes contiene un método *set* y un método *get* que sirven al usuario para instanciar un valor y para obtenerlo de cada uno de los parámetros. A continuación, se muestra la clase “Recibo”:

```
class Recibo {
public:
	void setNombrePlantillaCabecera(std::string nombrePlantillaCabecera);
	void setNombrePlantillaCuerpo(std::string nombrePlantillaCuerpo);
	void setNombrePlantillaFondo(std::string nombrePlantillaFondo);
	void setValorVar(std::string nombreVar, std::string valorVar);

	std::string getNombrePlantillaCabecera();
	std::string getNombrePlantillaCuerpo();
	std::string getNombrePlantillaFondo();
	std::string getValorVar(std::string nombreVar);
	stringmap getValores();
private:
	std::string nombrePlantillaCabecera;
	std::string nombrePlantillaCuerpo;
	std::string nombrePlantillaFondo;
	stringmap valores;
};
```

Se observa que los parámetros “nombrePlantillaCabecera”, “nombrePlantillaCuerpo” y “nombrePlantillaFondo” son atributos de tipo privado, por lo que no pueden ser obtenidos directamente del objeto instanciado.

Por ejemplo, si deseáramos guardar el cuerpo del recibo en un archivo, tendríamos que llamar al método siguiente:

```
	void setNombrePlantillaCuerpo(std::string nombrePlantillaCuerpo);
```

siendo que el parámetro “nombrePlantillaCuerpo” es el nombre del archivo al cual se quiere asignar el valor del atributo de mismo nombre de la clase. Posteriormente cuando quisiéramos obtener el mismo valor utilizamos el método siguiente, de esta forma obtengamos el valor del archivo que asignamos con anterioridad:

```
	std::string getNombrePlantillaCuerpo();
```

La misma clase “Recibo” puede almacenar variables que dependan de otros factores, por ejemplo, la fecha, la hora, el monto a cobrar, la propina, el número de tarjeta; que a pesar de ser valores que cambian de un recibo a otro, dentro de la impresión se mantienen relativamente en su posición uno con respecto a otro, es por

eso que la clase puede almacenar un tipo de dato llamado *stringmap* que es simplemente una tabla que relaciona el nombre de una variable con su valor.

Tabla 1 ejemplo de un stringmap.

Nombre	Valor
Tarjeta	4152 3113 5825 1234
Fecha	31/12/2018
Hora	10:58
Monto	610.20

De esta forma la clase “Recibo” contiene una propiedad llamada *valores* que tiene dos métodos *get, el primero de ellos devuelve únicamente el valor de la variable que le pasemos como parámetro, mientras que el segundo devuelve todos los valores de las variables.*

```
std::string getValorVar(std::string nombreVar);
stringmap getValores();
```

Si quisiéramos añadir o cambiar algún valor contamos con el método *set*, recibe como parámetros el nombre de la variable que deseamos modificar y el valor que deseamos asignarle:

```
void setValorVar(std::string nombreVar, std::string valorVar);
```

La segunda clase que compone el módulo de impresión es la llamada “ImpresionAdmin”, esta clase es la encargada de levantar el hardware que se va a utilizar para imprimir. También puede revisar el estado en el que se encuentra la impresora, esto es, si se encuentra funcionando, si le falta papel, si está ocupada imprimiendo algún ticket, si se atoró el papel, etcétera.

La clase “ImpresionAdmin”, contiene distintos métodos para imprimir, dependiendo de los parámetros que tenga un atributo de tipo “Recibo” que está contenido dentro de la misma clase, o bien, puede imprimir archivos HTML en “crudo” esto es, como si fueran archivos que muestran una página web. También tiene un método para imprimir directo desde una URL.

La clase “ImpresionAdmin” se muestra a continuación:

```
class ImpresionAdmin {
public:
	ImpresionAdmin();

	int init();

	int revisarEstado();

	int imprimir(Recibo& Recibo,
		bool imprimirCabecera = true,
		bool imprimirCuerpo = true,
		bool imprimirFondo = true);

	int imprimir(bool imprimirCabecera = true,
		bool imprimirCuerpo = true,
		bool imprimirFondo = true);

	int imprimir(std::string HTMLCrudo);
	int imprimir(std::string HTMLCrudo, stringmap valores);


	int imprimirCrudoAsync(std::string HTMLCrudo);
	int imprimirCrudoAsync(std::string HTMLCrudo, stringmap valores);

	int imprimirURLAsync(stringmap valores, std::string url);

	Recibo* getRecibo();
};
```

El primer proceso que debe realizar es levantar el hardware de impresión con el método:

```
	int init();
```

posteriormente ya se puede revisar el estado de la impresora para saber si está lista para imprimir con el método:

```
	int revisarEstado();
```

a continuación, si así se desea, se puede asignar un apuntador a un objeto de clase "Recibo", si es que se va a trabajar con él, en caso contrario se deberán usar los métodos cuyo sufijo es *Async* o que contengan como parámetro una variable de nombre "*HTMLCrudo"*. Para imprimir se puede llamar a cualquiera de los métodos del mismo nombre, por ejemplo:

```
int imprimir(Recibo& Recibo,
       bool imprimirCabecera = true,
       bool imprimirCuerpo = true,
       bool imprimirFondo = true);
```

si se quiere utilizar el objeto de tipo "Recibo", en caso de que se quisiera utilizar un archivo HTML se puede utilizar alguno de los siguientes métodos:

```
int imprimir(std::string HTMLCrudo);
int imprimir(std::string HTMLCrudo, stringmap valores);
```

el primero de ellos solo contiene instrucciones HTML planas, mientras que el segundo incluye variables y sus valores están indicados en el parámetro del mismo nombre.

Los métodos de sufijo *Async* indican que son asíncronos, esto es que, el código al llegar a esa línea mandará la instrucción para imprimir y continuará su flujo normal sin interrupciones; mientras que las instrucciones sin ese prefijo se realizan de forma síncrona, esperan a que la impresión termine para continuar con la ejecución del código. Las principales ventajas y desventajas se muestran en la siguiente tabla:

Tabla 2 comparación de impresión síncrona y asíncrona.

	Ventajas	Desventajas
Síncrono	Se puede condicionar el comportamiento dependiendo del estado de la impresión (si se acabó el papel, si se imprimió bien, si se apagó la impresora)	Mayor tiempo de la transacción completa.
Asíncrono	Reducción en el tiempo de las transacciones.	No se puede validar que la impresión se haya realizado de manera apropiada

3.3 Módulo de manejo de bases de datos.

El módulo encargado de la base de datos está conformado por tres clases que están íntimamente relacionadas. La primera y más importante de ellas es la clase “BaseDatos” que es la encargada de realizar las ejecuciones de consultas y tratar, a nivel código, la base de datos como un archivo. La segunda clase es la clase “Tabla” que pueden existir n de ellas dentro de la base de datos. Finalmente, la última clase de este módulo es la clase “Fila” que, dentro de cada tabla, podrán existir n cantidad de objetos de este tipo.

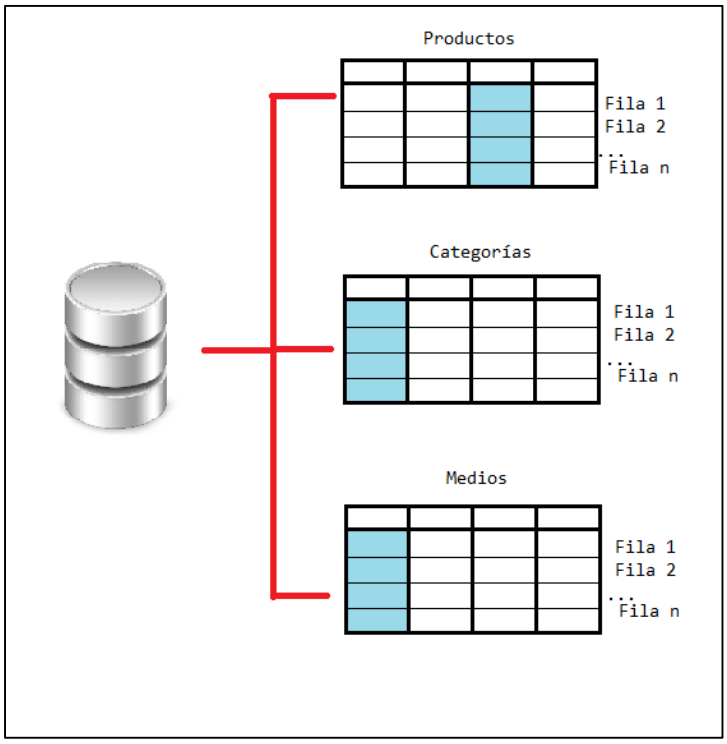

Fig. 3.4 Ejemplo de una base de datos.

La clase “BaseDatos” se muestra a continuación:

```
class BaseDatos {
public:
	BaseDatos();

	int abrirBD(std::string DBName);
	void cerrarBD();

	int ejecutarSQL(std::string query);
```

```
        int ejecutarSQL(std::string query, Tabla **Tabla);

        int vaciar();
};
```

La clase tiene un primer método encargado de abrir la base de datos, al ser tratada como un tipo de archivo, no es necesario que se levante ningún tipo de hardware en particular como en el resto de los módulos.

```
        int abrirBD(std::string DBName);
```

El parámetro “DBName” indica el nombre de la base de datos que será abierta, el nombre del archivo.

Una vez abierta la base de datos deseada, se puede proceder a ejecutar una petición (*query* en inglés), con alguno de los dos métodos siguientes, el primero de ellos es para ejecutar una petición completa, el segundo es por si deseamos obtener una tabla como resultado de nuestra petición:

```
        int ejecutarSQL(std::string query);
        int ejecutarSQL(std::string query, Tabla **Tabla);
```

Si deseamos borrar la base de datos completamente, existe el siguiente método, el cual se tiene que tener extremo cuidado, pues los datos borrados no podrán ser recuperados:

```
        int vaciar();
```

Finalmente, cuando se haya terminado de trabajar, se debe cerrar la base de datos, como si fuera un archivo más, con el método siguiente, y con esto queda explicado el sencillo funcionamiento de la clase “BaseDatos”.

```
        void cerrarBD();
```

La segunda clase involucrada es “Tabla”, que se muestra a continuación:

```
class Tabla {
public:
	Fila* getFilaActual();
	int moverSig();

private:
	friend class BaseDatos;
	Tabla(sqlite3 *DBConn);

	int abrir(std::string query);

	sqlite3 *DBConn;
	Fila filaActual;
};
```

Lo primero que debemos notar es que la clase “Tabla”, contiene su constructor `Tabla(sqlite3 *DBConn)` como un método privado, esto significa que nadie puede crear un objeto de este tipo, por eso es que se encuentra la línea: `friend class BaseDatos;` ya que esto indica que la única clase que puede crear un objeto de tipo “Tabla” será la clase amiga (*friend class*) “BaseDatos”.

Otro de los atributos que se pueden observar como privados son un apuntador de tipo “sqlite3”, que esto es un tipo especial para el manejo de base de datos; y un objeto de tipo “Fila”, que será el objeto con el que se trabajará a bajo nivel.

La clase “Tabla”, cuenta con solo dos métodos que son públicos:

```
	Fila* getFilaActual();
	int moverSig();
```

el primero de ellos (`getFilaActual();`) nos devuelve el objeto de tipo “Fila” en el cual nos encontramos (la fila completa de la tabla, el valor por defecto es la primera fila). Mientras que el segundo método (`moverSig();`) nos devuelve el valor de la siguiente fila (el número de fila siguiente).

Se observa que este objeto solo es de utilidad para desplazarse de una fila a otra dentro de una misma tabla.

La última clase de este módulo es la clase “Fila” cuya estructura es la siguiente:

```
class Fila {
public:
	unsigned int getNumCampos();
	std::string getValorCampo(std::string key);

private:
	friend class Tabla;
	Fila();

	void borrar();
	void setCampo(std::string key, std::string value);

	std::map <std::string, std::string> campos;
};
```

Al igual que con la clase “Tabla”, lo primero que debemos notar es que la clase “Fila” tiene su constructor definido como privado y tiene como clase amiga a la clase “Tabla”. Entonces haciendo una rápida recapitulación, la clase “Fila” solo puede ser creada por la clase “Tabla”, la cual a su vez solo puede ser creada por la clase “BaseDatos”, quien realiza todo a través de sus peticiones.

Un atributo de la clase “Fila”, es el siguiente, indicándonos que cada columna tiene un valor y ambos son de tipo cadena de caracteres.

```
	std::map <std::string, std::string> campos;
```

Dichos campos pueden ser modificados con los siguientes dos métodos privados, el primero de estos se encarga de borrar por complete la fila, mientras que el segundo modifica un valor en particular con los parámetros “key” y “value”, donde “key” es el nombre de la columna a modifica, tal cual está guardada en el atributo “campos”; y “value” es el valor nuevo que se almacenará.

```
	void borrar();
```

```
void setCampo(std::string key, std::string value);
```

Los dos métodos públicos con los que cuenta la clase “Fila” son los siguientes, siendo que el primero de ellos devuelve el número de columnas que contiene la fila, y el segundo de ellos devuelve el valor particular de un dato dando como parámetro al método el nombre de la columna.

```
unsigned int getNumCampos();
std::string getValorCampo(std::string key);
```

3.4 Módulo de interfaz gráfica.

El módulo de interfaz gráfica es aquel encargado de gestionar la pantalla, esto es mensajes, imágenes y partes esenciales del flujo de una transacción.

Un ejemplo de cómo se vería una pantalla inicial de una aplicación se muestra en la Fig. 3.5:

Fig. 3.5 Menú principal de una aplicación en una TPV

Este módulo cuenta con tres clases esenciales para manejar lo que se muestra en pantalla. La primera de estas es la clase llamada “ControladorBarraEstado” que es

la clase encargada de crear, actualizar y detener la barra de estado (barra de color verde de la Fig. 3.5). Esta clase se muestra a continuación:

```
class ControladorBarraEstado {
private:
	std::string htmlFile;
	int refrescarTime;

public:
	ControladorBarraEstado();

	void setArchivoHTML(std::string sHtmlName);
	int inicio(vfisysinfo::sysStatusbarCallback sbCb = 0, void *cbData = 0);

	int refrescar(void);
	void setTiempoRefrescar(int refrescarsec);

	void detener();
};
```

Esta clase cuenta con dos atributos que le sirven, el primero para almacenar la dirección (o nombre) del archivo HTML y el otro es un indicador, en segundos, del tiempo para que se esté actualizando la barra de estado, téngase en cuenta que cuanto menor sea este tiempo, menor será el rendimiento de la aplicación y por ende de la duración de la batería. Los atributos que se mencionan son los siguientes:

```
	std::string htmlFile;
	int refrescarTime;
```

Se puede observar que la clase contiene un par de métodos *set* como se muestra a continuación, y cada uno de estos sirve para actualizar el valor de las dos propiedades privadas de la clase “ControladorBarraEstado”.

```
	void setArchivoHTML(std::string sHtmlName);
	void setTiempoRefrescar(int refrescarsec);
```

Para iniciar a mostrar la barra de estado es necesario únicamente llamar al método siguiente:

```
int inicio(vfisysinfo::sysStatusbarCallback sbCb = 0, void *cbData = 0);
```

este método cuenta con un par de parámetros que pueden ir vacíos y serán tomados como ceros, el primero es un objeto de tipo “sysStatusbarCallback” que es una estructura (fuera del alcance de este trabajo) que contendrá información necesaria para el SO; mientras que el segundo es un apuntador de tipo *void*, esto es que, puede aceptar cualquier tipo de apuntador, donde se almacenarán las variables y sus respectivos valores, según lo requiere el archivo “htmlFile” de esta misma clase.

El tiempo para actualizar la barra de estado puede suceder cada determinado tiempo, el usuario puede realizarla de manera manual a su discreción con el método siguiente:

```
int refrescar(void);
```

Para dejar de mostrar la barra de estado puede llamar simplemente el método siguiente:

```
void detener();
```

La segunda clase relacionada con este módulo de interfaz gráfica es una llamada “PantallaMensaje” que es la encargada de mostrar directamente algún tipo de mensaje en la pantalla, o alguna imagen, su estructura a continuación:

```
class PantallaMensaje {
private:
	int msgType;
	stringmap images;
	int imgCounter;
	std::string message;

public:
	PantallaMensaje();

	int display();

	void agregarImagen(std::string imgFile);
	void setMensaje(std::string msgText);
	void setTipoMensaje(int inType);
};
```

Esta clase cuenta con cuatro propiedades privadas que se muestran a continuación:

```
int msgType;
stringmap images;
int imgCounter;
std::string message;
```

La primera de estas es un identificador que indica que tipo de mensaje es, dependiendo del valor asignado puede mostrarse en diferentes partes de la pantalla o como algún tipo de mensaje en una ventana emergente, de ser el caso. La segunda propiedad es un identificador de imágenes que señala el nombre de la imagen con su respectiva ruta dentro de la memoria de la terminal, siendo la tercera propiedad un contador de imágenes para saber el número de imagen que se requiera mostrar. La última propiedad es el mensaje que se almacenará para mostrarse posteriormente.

La clase “PantallaMensaje” cuenta con dos métodos propiamente *set* que son los siguientes:

```
void setMensaje(std::string msgText);
void setTipoMensaje(int inType);
```

el primero de estos almacena el valor como el mensaje a ser mostrado, mientras que el segundo almacena el tipo de mensaje que será mostrado.

Para agregar una imagen a la estructura, existe un método encargado de esto:

```
void agregarImagen(std::string imgFile);
```

Finalmente, si se desea mostrar en pantalla los valores almacenados en este objeto, basta con llamar al método:

```
int display();
```

La última de estas clases es una llamada “GUIAdmin” que se muestra a continuación:

```
class GUIAdmin {
public:
	GUIAdmin();

	int inicio(void);

	void setArchivoCSS(std::string cssfile);
	void setUsoHTML(bool flag);
	void setDirRelativaImagenes(std::string path);

	int mostrarMensaje(int type,
		std::string title,
		std::string message,
		vfihtml::stringmap map_values,
		bool sync = true);

	int mostrarMensajeHTML(int type,
		std::string htmlPage,
		vfihtml::stringmap map_values,
		bool sync = true);

	int mostrarMenuInicial(void);
	int mostrarMenuLista(std::string title, bool async);
	void prepararEntrada(int inPrecision, int inMinL, int inMaxL);

	int mostrarEntrada(InputType type,
		std::string title,
		std::string message,
		std::vector<std::string> &value,
		bool async = false);

	int mostrarEntradaPIN(std::string title, int min, int max);
	int mostrarEntradaPIN(std::string title);
private:
	ControladorBarraEstado statusBar;
	std::string cssFile;
	bool useHTML;
	std::string imgRelPath;
	vfihtml::JSObject menu;
};
```

Lo primero a notar son cinco atributos privados, uno de ellos es la barra de estado, la cual será modificada con ayuda de sus métodos. Los otros cuatro se muestran a continuación:

```
	std::string cssFile;
	bool useHTML;
	std::string imgRelPath;
```

```
vfihtml::JSObject menu;
```

el primero de estos es la ruta al archivo CSS que sirve para dar estilo a la pantalla que se mostrara, el segundo es una bandera que indica si se usarán archivos HTML para mostrarse en pantalla, el tercero es una ruta donde estarán guardadas las imágenes que se pueden mostrar en pantalla, el último de estos objetos es un archivo de tipo JSON que contiene la información del menú a mostrar.

Los primeros tres atributos contienen su propio método *set* tal y como se muestra a continuación:

```
void setArchivoCSS(std::string cssfile);
void setUsoHTML(bool flag);
void setDirRelativaImagenes(std::string path);
```

Si bien, propiamente la pantalla funciona por sí misma, se requiere de cierto software capaz de mostrar los textos e imágenes de manera apropiada con ciertas configuraciones, es por eso necesario comenzar con el método siguiente, antes de mostrar algo en la pantalla:

```
int inicio(void);
```

Se pueden mostrar mensajes directamente en la pantalla con cualquiera de los dos métodos siguientes:

```
int mostrarMensaje(int type,
       std::string title,
       std::string message,
       vfihtml::stringmap map_values,
       bool sync = true);

int mostrarMensajeHTML(int type,
       std::string htmlPage,
       vfihtml::stringmap map_values,
       bool sync = true);
```

La única diferencia entre ambos métodos es que el primero mostrara el texto con la configuración predeterminada en el archivo CSS, mientras que el segundo lo hará con el formato HTML que contenga el archivo de la misma extensión.

Se pueden mostrar estructuras de menús con cualquiera de los siguientes métodos, el primero de ellos mostrará el menú que es una propiedad privada de la clase “GUIAdmin”, y el segundo mostrará un menú en forma de lista y podrá hacerlo de manera síncrona o asíncrona.

```
int mostrarMenuInicial(void);
int mostrarMenuLista(std::string title, bool async);
```

Si requiere ingresar algún tipo de información para realizar la transacción, como puede ser el caso de la cantidad, propina, número de tarjeta, etcétera. Se deben utilizar los siguientes dos métodos de manera consecutiva:

```
void prepararEntrada(int inPrecision, int inMinL, int inMaxL);

int mostrarEntrada(InputType type,
       std::string title,
       std::string message,
       std::vector<std::string> &value,
       bool async = false);
```

El primero prepara la entrada de información en caso de ser una cantidad numérica la que vaya a ser ingresada, los parámetros de este método son la precisión, es decir el número de cifras después del punto; además de la longitud mínima y la máxima que puede ser ingresada por el teclado.

Existe un caso particular de entrada que requiere encomendar el manejo de la interfaz gráfica a la parte segura del kernel del SO, dicho caso es al solicitar el ingreso del PIN, ya que para salvaguardar la información del tarjetahabiente es necesario hacerlo de esta forma, para esto se deben llamar a cualquiera de los dos métodos siguientes:

```
int mostrarEntradaPIN(std::string title, int min, int max);
int mostrarEntradaPIN(std::string title);
```

estos métodos en lo único que se diferencian, es que en el primero se establece una longitud mínima y una máxima que debe tener el pin, mientras que la segunda acepta un pin de cualquier longitud.

3.5 Otros módulos.

Una particularidad de desarrollar este proyecto como un marco de trabajo, en vez de desarrollarlo como una aplicación terminada, es el uso de módulos independientes que permiten al usuario añadir más de estos a la aplicación final. Siendo esta misma aplicación final, la encargada de gestionar el acceso final de recursos (memoria y hardware).

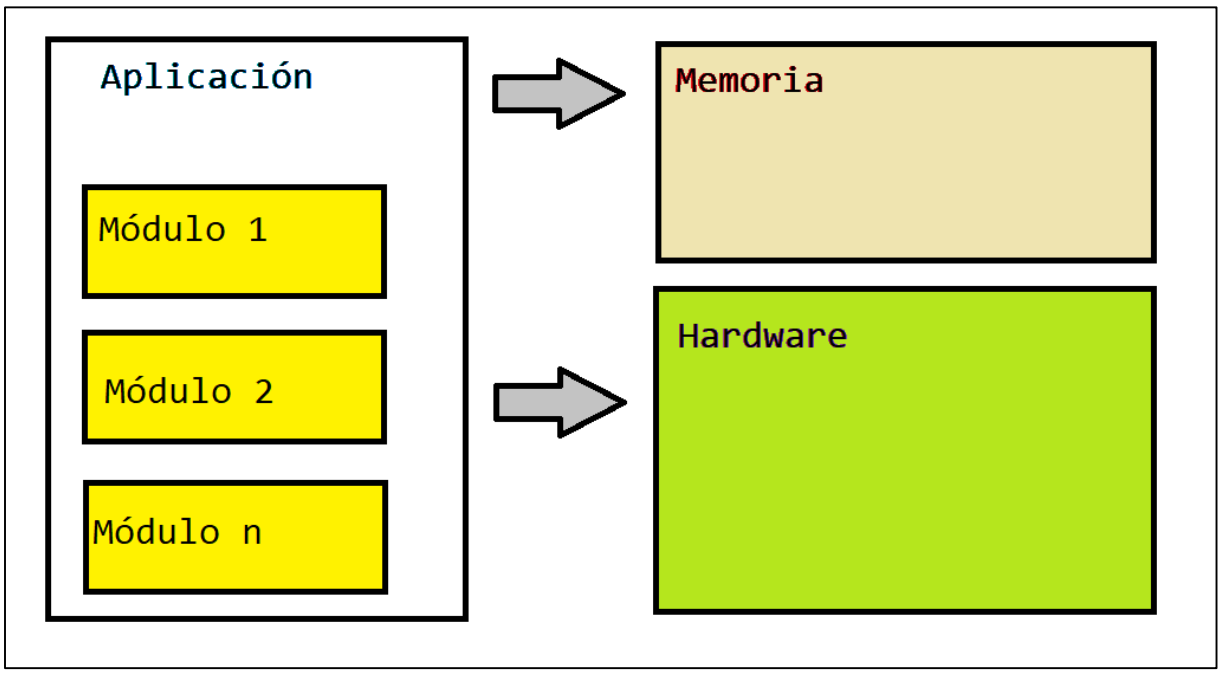

Fig. 3.6. Muestra de la aplicación y sus módulos.

Esta forma de trabajo, además permite al usuario minimizar los recursos según lo requiera la aplicación con la que se vaya a trabajar, no serán los mismos módulos que se necesitarán para una terminal que funcione como PINPAD en comparación con una que funcione como TPV.

La modularidad también hace más sencillo el mantenimiento del código, pues cuando se encuentre algún problema con cierto componente, se puede proceder a analizar y probar el módulo aislado del resto de la aplicación.

Unos módulos incluidos dentro de esta misma aplicación fueron los siguientes:

- Módulo de EMV.
- Módulo lector de tarjeta.
- Módulo de encriptación.
- Módulo de lectura de teclado.
- Módulo de aplicaciones simultaneas.

Estos módulos funcionan independientes unos de otros, aunque en algunos casos algunos requieren los métodos de algunas clases de otros módulos, como lo es en el caso del módulo de EMV que requiere del módulo de lectura de tarjeta leer la misma tarjeta, pero este intercambio no lo realiza ninguno de estos dos módulos, sino que lo realiza la aplicación final.

A modo de aclaración, los módulos de lectura de tarjeta y EMV se encuentran separados, porque algunos comercios pueden tener tarjetas propias, como lo pueden ser tarjetas de lealtad, de puntos, de clientes frecuentes, etc. (llamadas de circuito cerrado) que no siguen necesariamente el protocolo EMV.

Conclusiones

Paradox es una empresa que cuenta con la experiencia y conocimientos necesarios para desarrollar aplicaciones móviles para terminales punto de venta en el ámbito nacional e internacional. Sus proyectos han abarcado trabajos en conjunto con empresas de diferentes países de América: EEUU, Canadá, Nicaragua, El Salvador, República Dominicana, Perú, Argentina, entre otros.

Mi desarrollo profesional en este ámbito se vio impulsado de manera positiva por el respaldo que Paradox me brindó, ya que además de las capacitaciones que tuve dentro de la empresa, la experiencia de haber trabajado en proyectos nacionales e internacionales me llena de satisfacción pues sé que afuera hay un producto que la gente utiliza de manera cotidiana y en el cuál yo participé de manera activa.

El haber tenido contacto con proyectos de diferentes tipos, hizo que mi experiencia fuera bastante amplia, ya que además de los conocimientos técnicos referentes a la aplicación en sí; obtuve aprendizajes que me permiten manejar mi información bancaria con mayor seguridad. Esto debido al conocimiento de los protocolos manejados para la seguridad de dicha información y al conocimiento de la situación actual en la que se encuentra legal y técnicamente este ámbito.

Los conocimientos técnicos que tengo por mi experiencia dentro de Paradox, me permiten desarrollarme de manera profesional en cualquier otra rama referente a la programación. Esto gracias al constante esfuerzo que invertí en conocer lo referente a la misma programación y al gran apoyo que tuve por parte de la empresa para desarrollar mis actividades de la mejor manera posible.

Cada aplicación móvil es distinta una de la otra, siempre hay que tener en cuenta el fin con el cual se va a utilizar. Para el caso de estudio que se desarrolló en el capítulo 3 de este este trabajo, se observa claramente que el desarrollo de esta aplicación como un marco de trabajo dividido en módulos, es más eficiente que el

desarrollo de una aplicación en conjunto que crezca exponencialmente conforme más implementaciones sean requeridas.

Lo anterior gracias a la programación orientada a objetos, que además de permitirnos separar dichos componentes en módulos y mantenerlos unidos por un administrador general de la aplicación; nos permite añadir, remover y/o modificar los mismos módulos sin interferir directamente en las relaciones que mantiene cada uno de estos módulos con el resto de ellos.

Los conocimientos adquiridos durante la carrera son los fundamentos de otros conocimientos que son utilizados en el ramo privado. Estos conocimientos no pueden ser tan profundos para cubrir las necesidades del sector privado, ya que esto extendería ampliamente la duración de los planes de estudio. Pero, con los conocimientos fundamentales de la ingeniería, un recién egresado puede ahondar en un cierto ramo, en un periodo relativamente breve de tiempo.

En cualquier industria siempre existirá conocimiento nuevo que podamos adquirir, a pesar de que se cuente con años de experiencia en el nicho, esto debido a la naturaleza siempre evolutiva de la ciencia y la tecnología. Por ende, siempre debemos permanecer a la vanguardia y abiertos al cambio.

En mi experiencia profesional, me sentí orgulloso de toda la trayectoria recorrida durante mis estudios, ya que, gracias al esfuerzo realizado por cinco años dentro de la Universidad Nacional Autónoma de México, puedo aplicar mis conocimientos en la industria y generar valor agregado a cada uno de los proyectos en los que participo. Además, aprendí como se manejan los proyectos en la industria privada, las metodologías con las que se trabaja, y las habilidades que resultan importantes para el desenvolvimiento en el mundo laboral.

Las personas, cuando comenzamos a desempeñarnos en una actividad nueva, recorremos una curva de aprendizaje. En este recorrido, conforme menos

conocimientos tengamos acerca de un tema, el tiempo invertido para aprender acerca de este tema es relativamente más corto, en comparación con el conocimiento adquirido en momentos en los que tenemos mayor cantidad de conocimientos.

Una implicación de dicha curva de aprendizaje es que la velocidad con la que desarrollaremos una actividad depende del recorrido que hayamos realizado de dicha curva, esto es, que una persona novata en una actividad aprenderá más que alguien experto, pero realizará las tareas de una forma más lenta.

En mi caso particular, el periodo que me llevó alcanzar un punto en el cual la velocidad de desarrollo de mis proyectos fuera aceptable, fue de aproximadamente 6 meses.

Todas las experiencias vividas en estos años de estudiante, junto con el tiempo que laboré en el ámbito profesional, hicieron de mí una persona más íntegra, responsable; y termino mi ciclo de estudiante con grata alegría y entusiasmo por mi porvenir profesional.

Referencias

Bryant, R. (2018). *Computer Systems. A Programmer's Perspective.*
EEUU: Pearson.

Koening, A. (2015). *Accelerated C++. Practical Programming by Example.*
EEUU: Addison Wesley.

Stroustrup, B. (2014). *Programming Principles and Practice Using C++.*
EEUU: Addison Wesley.

Stroustrup, B. (2017). *The C++ Programming Language.*
EEUU: Pearson.

http://www.anterior.banxico.org.mx/divulgacion/sistemas-de-pago/sistemas-pago.html#Mediospago
Banxico, página consultada el 1 de febrero de 2019.

http://www.paradox-tech.com/portalwebparadox/
Paradox, página consultada el 1 de febrero de 2019.

Glosario

ABM	Asociación de Bancos de México.
ATM	*Automated Teller Machine* (Cajero Automático).
CSS	*Cascading Style Sheets.*
ECR	*Electronic Cash Register.*
EMV	Protocolo creado por las marcas *Europay, MasterCard* y *Visa*.
FVCS	*File Version Control System.*
HTML	*Hyper Text Markup Language.*
IC	*Integrate Circuit.*
ICC	*Integrated Chip Card.*
ISO	*International Organization for Standardization.*
JSON	*JavaScript Object Notation.*
PIN	*Personal Identifier Number.*
POO	Programación Orientada a Objetos.
QA	*Quality Assurance.*
SO	Sistema Operativo.
SSL	*Secure Sockets Layer.*
TLS	*Transport Layer Security.*
TPV	Terminales Punto de Venta.
USB	*Universal Serial Bus.*

Anexos

Anexo A

Extracto de la circular única de Bancos

(2) **Sección Tercera**
(46) De la operación del servicio de Banca Electrónica
(46) **Artículo 314.-** Para la celebración de las Operaciones Monetarias previstas en las fracciones I y II del Artículo 313
de las presentes disposiciones, a través de los servicios de Banca Electrónica, las Instituciones deberán asegurarse
de que sus Usuarios registren en el servicio de Banca Electrónica de que se trate, las Cuentas Destino previamente a
su uso, ya sea para ser utilizadas dentro del mismo servicio o, si así lo convienen con sus Usuarios, en otros servicios de Banca Electrónica.
(46) Para el caso de pago de servicios e impuestos se considerará como registro de Cuentas Destino, al registro de
los convenios, referencias para depósitos, contratos o nombres de beneficiarios, mediante los cuales las Instituciones
hacen referencia a un número de cuenta.
(46) En ningún caso se podrán registrar Cuentas Destino a través de Banca Telefónica Voz a Voz.
(46) En el caso de los servicios ofrecidos a Usuarios que sean personas morales o personas físicas con actividad empresarial en términos de la legislación fiscal, las Instituciones podrán permitirles el registro de cuentas por conjuntos de cuentas, considerando el registro de cada conjunto de cuentas como una sola operación.
(46) Las Cuentas Destino deberán quedar habilitadas después de un periodo determinado por la propia Institución, sin
que este sea menor a treinta minutos contados a partir de que se efectúe el registro. Las Instituciones deberán informar al Usuario el plazo en que quedarán habilitadas dichas cuentas. Se exceptúa de este periodo a las Cuentas
Destino que hayan sido registradas a través de Banca Móvil, sin perjuicio de lo dispuesto en el último párrafo de este
artículo, las registradas en Oficinas Bancarias utilizando la firma autógrafa del Usuario, así como aquellas para efectuar pago de impuestos, excluyendo en este último concepto, el pago de tenencias vehiculares a que se refiere la
Ley del Impuesto sobre Tenencia o Uso de Vehículos.
(46) Asimismo, las Instituciones podrán habilitar Cuentas Destino registradas por sus Usuarios sin que les sea aplicable el periodo mínimo de tiempo referido en el párrafo anterior, siempre y cuando sea para la realización de Operaciones Monetarias a través de Banca por Internet cuyo monto agregado diario no exceda al equivalente en moneda nacional a las de Baja Cuantía, o bien, el equivalente en moneda nacional a 1,000 UDIs mensuales y obtengan la previa autorización de la Comisión. Las Instituciones deberán exponer en la solicitud respectiva los controles que les permitirán a los Usuarios realizar operaciones de forma segura. En todo caso, las Instituciones deberán determinar el tiempo para que queden habilitadas las Cuentas Destino, una vez que el Usuario haya realizado el registro previo de las mismas.
(46) Las Instituciones, con base en la información disponible deberán validar al momento del registro, la estructura del
número de la Cuenta Destino, del contrato o de la clave bancaria estandarizada, ya sea que se trate de cuentas para
depósito, pago de servicios, tarjetas bancarias u otros medios de pago.
(46) Para las Operaciones Monetarias que se realicen a través de Banca Host to Host, Terminales Punto de Venta o
Cajeros Automáticos, no se requerirá que los Usuarios registren las Cuentas Destino; tampoco para las que se realicen mediante Pago Móvil y Banca Móvil, siempre que, tratándose de estos dos últimos, el monto de dichas operaciones sea hasta el equivalente a las de Baja Cuantía por cada operación.
(46) **Artículo 315.-** Las Instituciones podrán permitir a sus Usuarios establecer límites de monto para las Operaciones
Monetarias que se realicen a través de los servicios de Banca Electrónica, obteniendo su consentimiento mediante
firma autógrafa en Oficinas Bancarias, previa identificación de estos.
(46) Asimismo, las Instituciones deberán proveer lo necesario para que sus Usuarios establezcan límites de monto para las Operaciones Monetarias previstas en las fracciones I y II del Artículo 313 de las presentes disposiciones,

para los servicios de Banca por Internet, Banca Telefónica Voz a Voz, Banca Telefónica Audio Respuesta y Banca
Móvil.
(46) Las Instituciones deberán permitir a sus Usuarios reducir los límites establecidos previamente en dichos servicios
de Banca Electrónica, utilizando un Factor de Autenticación Categoría 2 a que se refiere el Artículo 310 de las presentes disposiciones. Para el caso del servicio de Banca Telefónica Voz a Voz, las Instituciones podrán emplear
un Factor de Autenticación Categoría 1 a que se refiere el Artículo 310 de las presentes disposiciones.
(46) Tratándose de Cajeros Automáticos, el monto acumulado diario de las Operaciones Monetarias que representen
un cargo a la cuenta del cliente, no podrá exceder del equivalente en moneda nacional a las Operaciones Monetarias
de Mediana Cuantía por cuenta.
(46) En ningún caso el monto acumulado de las Operaciones Monetarias realizadas por un Usuario a través de Pago
Móvil, aún cuando tenga asociadas hasta dos tarjetas o cuentas bancarias, en su caso, podrá exceder del equivalente en moneda nacional a las Operaciones Monetarias de Mediana Cuantía en un día y no deberán superar
el equivalente en moneda nacional a 4,000 UDIs mensuales. Tratándose de Operaciones Monetarias de Micro Pagos, el saldo disponible de la cuenta asociada al Teléfono Móvil no podrá ser mayor al equivalente en moneda nacional a 70 UDIs.
(46) Sin perjuicio de lo dispuesto en el presente artículo, las Instituciones podrán definir límites inferiores específicos
para cada servicio de Banca Electrónica, siempre y cuando no contravengan lo previsto por las presentes disposiciones.
(46) **Artículo 316.-** Las Instituciones deberán solicitar a sus Usuarios que confirmen la celebración de una Operación
Monetaria, previo a que se ejecute, haciendo explícita la información suficiente para darle certeza al Usuario de la
operación que se realiza.
(46) Se exceptúa de lo anterior a los servicios de Banca Electrónica ofrecidos a través de Terminales Punto de Venta.
(47) **Artículo 316 Bis.-** Las Instituciones deberán establecer mecanismos y procedimientos para que los servicios de
Banca Electrónica generen los comprobantes correspondientes respecto de las operaciones y servicios realizados
por sus Usuarios a través de dichos servicios de Banca Electrónica.
(47) **Artículo 316 Bis 1.-** Las Instituciones estarán obligadas a notificar a sus Usuarios a la brevedad posible y a través del medio de comunicación cuyos datos haya proporcionado el Usuario para tal fin, cualquiera de los siguientes eventos realizados a través de los servicios de Banca Electrónica:
(47) I. Transferencias de recursos dinerarios a cuentas de terceros u otras Instituciones, incluyendo el pago de créditos y de bienes o servicios, así como las autorizaciones e instrucciones de domiciliación de pago de bienes o servicios;
(47) II. Pago de impuestos;
(47) III. Modificación de límites de montos de operaciones;
(47) IV. Registro de Cuentas Destino de terceros u otras Instituciones;
(47) V. Alta y modificación del medio de notificación al Usuario, debiendo enviarse tanto al medio de notificación anterior como al nuevo;
(47) VI. Contratación de otro servicio de Banca Electrónica o modificación de las condiciones para el uso del servicio
de Banca Electrónica previamente contratado;
(47) VII. Desbloqueo de Contraseñas o Números de Identificación Personal (NIP), así como para la reactivación del
uso de los servicios de Banca Electrónica;
(47) VIII. Modificación de Contraseñas o Números de Identificación Personal (NIP) por parte del Usuario, y
(47) IX. Retiro de efectivo en Cajeros Automáticos.
(47) Las Instituciones deberán asegurarse de que la información transmitida para notificar al Usuario sobre los eventos
a que se refiere el presente Artículo, no contenga números de cuenta completos, domicilios, ni saldos.
(47) Las notificaciones sobre la realización de las operaciones señaladas en las fracciones I, II y IX del Artículo 313 de

estas disposiciones, efectuadas a través de Pago Móvil, Cajeros Automáticos y Terminales Punto de Venta, deberán
ser enviadas cuando el acumulado diario de dichas operaciones por servicio de Banca Electrónica de que se trate,
sea mayor al equivalente en moneda nacional a 600 UDIs, o bien, cuando las Operaciones Monetarias en lo individual sean mayores al equivalente en moneda nacional a 250 UDIs. En este último caso, siempre y cuando las
Instituciones cuenten con esquemas específicos de prevención de fraudes con el fin de revisar continuamente aquellas operaciones que puedan constituir un uso no autorizado de los servicios de Banca Electrónica.
(47) En ningún caso las Instituciones permitirán la modificación del medio de notificación a través de Cajeros Automáticos y Terminales Punto de Venta. Las Instituciones deberán permitir a sus Usuarios modificar el medio de
notificación de los servicios de Banca Electrónica ofrecidos en Cajeros Automáticos o Terminales Punto de Venta mediante un centro de atención telefónica, utilizando un Factor de Autenticación Categoría 1 a que se refiere el Artículo 310 de las presentes disposiciones.
(47) Se exceptúa de lo señalado en el presente artículo a las operaciones realizadas mediante el servicio de Banca
Host to Host.
(47) **Artículo 316 Bis 2.-** Las Instituciones deberán proveer lo necesario para que una vez autenticado el Usuario en el
servicio de Banca Electrónica de que se trate, la Sesión no pueda ser utilizada por un tercero. Para efectos de lo anterior, las Instituciones deberán establecer, al menos, los mecanismos siguientes:
(47) I. Dar por terminada la Sesión en forma automática, e informar al Usuario del motivo en cualquiera de los casos
siguientes:
(47) a) Cuando exista inactividad por más de veinte minutos.
(47) Tratándose de operaciones realizadas mediante Pago Móvil, Cajeros Automáticos y Terminales
Punto de Venta, el periodo de inactividad no podrá exceder de un minuto.
(47) Para operaciones realizadas mediante Banca Host to Host, las Instituciones podrán definir el periodo
de inactividad, con base en los riesgos asociados al servicio que las propias Instituciones determinen.
(47) b) Cuando en el curso de una Sesión del servicio de Banca por Internet, la Institución identifique cambios relevantes en los parámetros de comunicación del Medio Electrónico, tales como identificación del
Dispositivo de Acceso, rango de direcciones de los protocolos de comunicación, ubicación geográfica,
entre otros.
(47) II. Impedir el acceso en forma simultánea, mediante la utilización de un mismo Identificador de Usuario a más de
una Sesión en el servicio de Banca Electrónica de que se trate e informar al Usuario, cuando el Identificador de Usuario esté siendo utilizado en otra Sesión.
(47) III. En el evento de que las Instituciones ofrezcan servicios de terceros mediante enlaces en el servicio de Banca
Electrónica, deberán comunicar a sus Usuarios que al momento de ingresar a dichos servicios, se cerrará automáticamente la Sesión abierta con la Institución de que se trate y se ingresará a otra cuya seguridad no depende ni es responsabilidad de dicha Institución.
(47) **Artículo 316 Bis 3.-** Las Instituciones deberán establecer procesos y mecanismos automáticos para Bloquear el
uso de Contraseñas y otros Factores de Autenticación para el servicio de Banca Electrónica, cuando menos para los
casos siguientes:
(47) I. Cuando se intente ingresar al servicio de Banca Electrónica utilizando información de Autenticación incorrecta.
En ningún caso los intentos de acceso fallidos podrán exceder de cinco ocasiones consecutivas, situación en la cual se deberá generar el Bloqueo automático.
(47) II. Cuando el Usuario se abstenga de realizar operaciones o acceder a su cuenta, a través del servicio de Banca
Electrónica de que se trate, por un periodo que determine cada Institución en sus políticas de operación y de acuerdo con el Medio Electrónico correspondiente, así como en función de los riesgos inherentes al mismo. En ningún caso, dicho periodo podrá ser mayor a un año. Lo anterior, no será aplicable a los servicios de Banca Electrónica ofrecidos a través de Cajeros Automáticos y Terminales Punto de Venta.
(47) Las Instituciones podrán Desbloquear el uso de Factores de Autenticación que previamente hayan sido Bloqueados en los casos contemplados en las fracciones I y II anteriores, para lo cual podrán utilizar un Factor de

Autenticación Categoría 1 a que se refiere el artículo 310 de las presentes disposiciones, en términos de lo previsto
por la fracción III del Artículo 312 de estas disposiciones, o bien, realizar a sus Usuarios preguntas secretas, cuyas
respuestas deben conservarse almacenadas en forma Cifrada. Para efectos de lo previsto en el presente párrafo, se
entenderá por pregunta secreta al cuestionamiento que define el Usuario o la Institución durante el proceso de contratación del servicio de Banca Electrónica, respecto del cual se genera información como respuesta. Cada pregunta secreta que se defina únicamente podrá ser utilizada en una ocasión.
(47) Con independencia de lo anterior, las Instituciones deberán permitir al Usuario el Restablecimiento de Contraseñas y Números de Identificación Personal (NIP) utilizando el procedimiento de contratación al servicio descrito en el Artículo 307 de las presentes disposiciones.
(47) **Artículo 316 Bis 4.-** Para el manejo de Contraseñas y otros Factores de Autenticación, las Instituciones se sujetarán a lo siguiente:
(47) I. Deberán mantener procedimientos que proporcionen seguridad en la información contenida en los dispositivos
de Autenticación en su custodia, la distribución, así como en la asignación y reposición a sus Usuarios de dichas Contraseñas y Factores de Autenticación.
(47) II. Tendrán prohibido contar con mecanismos, algoritmos o procedimientos que les permitan conocer, recuperar o
descifrar los valores de cualquier información relativa a la Autenticación de sus Usuarios.
(47) III. Tendrán prohibido solicitar a sus Usuarios, a través de sus funcionarios, empleados, representantes o comisionistas, la información parcial o completa, de los Factores de Autenticación de las Categorías 2 ó 3 a que se refiere el Artículo 310 de las presentes disposiciones.
(47) Se exceptúa de lo previsto en esta fracción, a las operaciones realizadas por Banca Telefónica Voz a Voz, siempre y cuando el Usuario haya iniciado la llamada, se requiera información parcial del Factor de Autenticación de
las Categorías 2 ó 3 a que se refiere el Artículo 310 de las presentes disposiciones, y este sea utilizado exclusivamente para este servicio de Banca Electrónica.
(47) **Artículo 316 Bis 5.-** Las Instituciones deberán establecer procedimientos para que sus Usuarios de Pago Móvil y
Banca Móvil puedan, en todo momento, desactivar su uso de forma temporal en caso de requerirlo, así como establecer procedimientos para reactivar el uso cuando el Usuario lo disponga.
(47) La desactivación del uso de manera temporal de los servicios de Banca Electrónica mencionados en el párrafo
anterior, deberá realizarse en todo momento dentro de una Sesión en el mismo servicio, o bien, a través de algún otro servicio de Banca Electrónica que el Usuario tenga contratado, debiendo requerir en ambos casos, un Factor de
Autenticación de cualquiera de las categorías previstas en el Artículo 310 de las presentes disposiciones.
(47) Para la reactivación del uso de los servicios de Banca Electrónica mencionados en el primer párrafo de este artículo, los Usuarios podrán utilizar los mismos mecanismos señalados en el Artículo 307 de estas disposiciones, o
bien, un Factor de Autenticación Categoría 1 a que se refiere el Artículo 310 de las presentes disposiciones. Las Instituciones deberán observar lo señalado en el Artículo 308 de estas disposiciones para poder iniciar una Sesión
una vez que se haya reactivado el servicio.
(47) **Artículo 316 Bis 6.-** Las Instituciones que pongan al alcance de sus Usuarios equipos electrónicos o de telecomunicaciones, en sus instalaciones o en áreas de acceso al público, para el uso del servicio de Banca Electrónica, deberán:
(47) I. Adoptar medidas que procuren detectar e impedir la instalación en tales equipos, de dispositivos o programas
que puedan interferir con el manejo de la información de los Usuarios, o que puedan permitir que dicha información sea leída, copiada, modificada o extraída por terceros. Adicionalmente, deberán informar a sus Usuarios, mediante campañas de difusión, sobre la apariencia y el funcionamiento de los equipos electrónicos o de telecomunicaciones que pongan al alcance de estos, a fin de prevenir actos que deriven o pudieran derivar en operaciones irregulares o ilegales que afecten a los Usuarios o a las propias Instituciones.
(47) II. Contar con procedimientos tanto preventivos como correctivos, que permitan correlacionar la información proveniente de las reclamaciones de los clientes con lo siguiente:
(47) a) El modo de operación del personal interno o externo de la Institución, que opera o administra los equipos electrónicos o de telecomunicaciones;
(47) b) Si los equipos han sido sujetos a alteraciones para robo de información de tarjetas, Números de Identificación Personal (NIP) o Contraseñas, y

(47) c) El resultado de las labores de identificación, monitoreo y análisis de comportamientos fuera de los parámetros establecidos por la Institución.
(47) Para tal fin, la Institución deberá presentar a los Comités de Auditoría y de Riesgos, cada vez que sesionen, un
informe de los resultados de la ejecución de dichos procedimientos.
(47) **Artículo 316 Bis 7.-** Las Instituciones que ofrezcan al público operaciones y servicios a través de centros de atención telefónica, deberán:
(47) I. Mantener controles de seguridad física y lógica en la infraestructura tecnológica de los centros de atención telefónica, incluyendo los dispositivos de grabación de llamadas y los medios de almacenamiento y respaldo de estas, que protejan en todo momento la confidencialidad e integridad de la información proporcionada por sus Usuarios.
(47) II. Delimitar las funciones de los operadores telefónicos a fin de que sean Independientes respecto de otras funciones operativas.
(47) III. Impedir que los operadores telefónicos cuenten con mecanismos que les permitan registrar la información proporcionada por sus Usuarios en medios diferentes a los dispuestos por la propia Institución para efectos de Autenticación. Para ello, las Instituciones deberán cerciorarse que las personas que tengan acceso a los centros de atención telefónica, no utilicen equipos electrónicos u otros dispositivos, servicios de correo electrónico externo, programas de mensajería instantánea, programas de cómputo, o que a través de estos tengan acceso a páginas de Internet no autorizadas, o cualquier otro mecanismo que les permita copiar, enviar o extraer por cualquier medio o tecnología información relacionada con los Usuarios, o con las operaciones y servicios que se realicen a través de los centros de atención telefónica.
(47) **Artículo 316 Bis 8.-** Las Instituciones que ofrezcan servicios de Banca Electrónica a través de Cajeros Automáticos y Terminales Punto de Venta, deberán asegurarse que estos cuenten con lectores que permitan obtener
la información de las Tarjetas Bancarias con Circuito Integrado, en el entendido de que la información deberá ser leída directamente del propio circuito o chip.
(47) **Artículo 316 Bis 9.-** Las Instituciones podrán consultar la Guía para el uso del servicio de Banca Electrónica, contenida en el Anexo 63 de las presentes disposiciones, sin que dicho documento tenga carácter vinculante para las
mismas.

Anexo B
Diagrama general de aplicación de EMV, Libro III de EMV p.157

El diagrama de flujo siguiente es el utilizado para desarrollar las aplicaciones móviles de pago según los estándares establecidos por el protocolo de EMV.

El primer paso es el arranque de la aplicación, posterior a este se procede a leer la información precargada en la terminal, la cual definirá el tipo de aplicación que esta pueda ser, esto es: si va a ser una terminal atendiad por un cajero, si funcionara como un Pinpad o un TPV, si aceptará ciertos tipos de marcas de tarjetas, etcétera.

En este diagrama se realizan de forma paralela el paso denominado “Terminal Risk Management” y el “Data Authentication”. El primero es la lectura de parámetros que se tomarán en cuenta para determinar qué tipo de autenticación se debe solicitar al tarjetahabiente, así como también si la operación se realizará en línea o fuera de línea.

El “Data Authentication” son los pasos para verificar la autenticidad de los datos que presenta la tarjeta, posterior se hará lectura de los “Processing Restrictions” que son las restricciones a las cuales estará sometida la transacción. Como paso siguiente se realiza el “Cardholder Verification” que es la forma en la que el comercio podrá verificar y certificar que el tarjetahabiente es quien está presentando la tarjeta, las tres formas de autenticación del tarjetahabiente son: firma autógrafa, solicitu de NIP y lectura de parámetros biomédicos (huella digital, reconocimiento de iris).

Una vez leída la información de la transacción, la terminal ejecuta el “Terminal Action Analysis”, que es básicamente verificar los procedimientos que tiene que realizar la terminal con respecto a la transacción.

Posteriormente el “Card Action Analysis” realiza una acción similar, pero con respecto a la tarjeta, para ver si esta tiene alguna acción particular que realizar con respecto a la transacción.

A continuación, se decide si la transacción se va en línea o se puede realizar fuera de línea, esto significa que se puede aprobar o rechazar la transacción sin necesidad de contrastar los datos con el banco emisor de la tarjeta. Si la transacción es realizada fuera de línea, ahí finaliza el proceso de EMV, en caso contrario se hace la transacción

en línea y se realiza la autenticación del portador y finalmente se ejecutan algunos scripts que pudieran existir.