Informe de Proyecto Final

“Incorporación de factura eléctrónica y desarrollo de aplicación *e-commerce*”

2021

Martín, Rodrigo Alexis

Ingeniería en Sistemas - PLAN 2011

Fecha de aprobación: **Noviembre 2021**

Modalidad: **Práctica en Empresa** - Empresa: **Iduo S.A**

Docente tutor: **Minetti, Gabriela Fabiana (Facultad de Ingeniería)**

Tribunal Evaluador:

- Becker, Pablo Javier (Facultad de Ingeniería)
- Martín, María de los Angeles (Facultad de Ingeniería)
- Papa, María Fernanda (Facultad de Ingeniería)

Agradecimientos

Primeramente, a mis padres, personas llenas de valores que inculcaron en mí lo mejor de ellos, respeto, humildad, perseverancia... Sin ellos quizás hoy no estuviera en este momento. Este trabajo está dedicado a ellos, por quienes nunca bajé los brazos.

A mis amigos de toda la vida, que aún siguen presentes. A los amigos de facultad, los que tuve y los que tendré por siempre. Con quienes compartimos charlas, mates y largas horas de estudio, sobrepasando momentos más que estresantes y que siempre estuvieron para brindar su apoyo, sus palabras y un hombro donde apoyarse.

A la empresa Iduo SA, quien depositó su confianza en mí, dándome la posibilidad de acceder a mi primer trabajo. Permitiendo crecer como profesional y accediendo a que pueda hacer este trabajo de tesis. A mis compañeros y jefe, por toda la ayuda y colaboración.

Finalmente, a mi tutora de tesis, la Dra. Gabriela Minetti, quien con su ayuda me guió a lo largo de este proyecto. Gracias.

Resumen

Iduo S.A es una empresa de software radicada en la ciudad de General Pico, que actualmente cuenta con alrededor de 300 clientes en todo el país. La empresa se encarga del análisis, diseño y desarrollo de soluciones de software para el sector empresarial con el objetivo de mejorar los estándares administrativos. Provee una plataforma que soporta procesos de gestión complejos e integra múltiples áreas operativas, brindando información gerencial para la toma de decisiones; un sistema del tipo ERP – CRM, comercializado bajo modalidad SaaS (Software as a Service). De esta forma, permite a los clientes simplificar la administración de los procesos de negocio mediante la integración de módulos informáticos que constituyen al sistema global, y que son específicos para cada tarea o actividad.

En el año 2017, con el propósito de hacer frente a la gran demanda de sus usuarios de realizar las operaciones en cualquier momento y lugar, la empresa decidió incorporarse al mundo de las aplicaciones móviles, desarrollando la aplicación móvil para Android, denominada Iduo.

La velocidad de cambio en cuanto a requerimientos de los clientes y a los tecnológicos, sumado a la falta de personal en la empresa, provocaron que la aplicación no tenga el mantenimiento adecuado. Es aquí donde surge la oportunidad de actualizar la aplicación móvil con el objetivo de que facilitara la manipulación y el procesamiento de información, brindando nuevas soluciones.

Como resultado de este trabajo se desarrolló una aplicación más completa, con más funciones y una arquitectura nueva que permite facilitar su mantenimiento y escalabilidad. Además, uno de los grandes beneficios obtenidos fue la distribución mediante Google Play, permitiendo llegar a un mayor número de clientes.

Palabras clases: Android, E-commerce, Factura, Comprobante

Abstract

Iduo S.A is a software company based in the city of General Pico, which currently has around 300 clients throughout the country. The company is responsible for the analysis, design and development of software solutions for the business sector with the aim of improving administrative standards. It provides a platform that supports complex management processes and integrates multiple operational areas, providing managerial information for decision-making; a system of the ERP - CRM type, marketed under the SaaS (Software as a Service) modality. In this way, it allows customers to simplify the administration of business processes by integrating computer modules that make up the global system, and that are specific for each task or activity.

In 2017, in order to face the great demand of its users to carry out operations at any time and place, the company decided to join the world of mobile applications, developing the mobile application for Android, called Iduo.

The speed of change in terms of customer and technological requirements, added to the lack of personnel in the company, caused the application not to have adequate maintenance. This is where the opportunity arises to update the mobile application in order to facilitate the manipulation and processing of information, providing new solutions.

As a result of this work, a more complete application was developed, with more functions and a new architecture that facilitates its maintenance and scalability. In addition, one of the great benefits obtained was distribution through Google Play, allowing it to reach a greater number of customers.

Key words: Android, E-commerce, Invoice, Receipt

Índice

Capítulo I Introducción

Actualmente vivimos en una sociedad donde las aplicaciones móviles tienen mucha importancia en el día a día. Cada vez usamos más las nuevas tecnologías y sobre todo la móvil para facilitar nuestra vida cotidiana (mensajería instantánea, GPS, correo electrónico y un largo etcétera). Todos estos servicios han sido una evolución lógica de Internet.

La aparición de dispositivos móviles como Tablets[1], y especialmente con los Smartphone[2], como así también la disminución de costos de fabricación ante la gran demanda de estos dispositivos, ha permitido que estén al alcance de la mayoría de los usuarios. Con ello las aplicaciones móviles también han experimentado un auge en cuanto a sus ofertas hacia sus usuarios, dándoles la libertad de tener una gran variedad de aplicaciones de todo ámbito y permitiendo el acceso a la información en todo momento y desde cualquier lugar del mundo.

Hoy en día un gran número de compañías, sobre todo las del sector de ventas se ven impulsadas a tener o desarrollar su propia aplicación móvil para sacar ventaja ante la competencia y así llegar a muchos otros mercados y agilizar sus procesos. Los usuarios necesitan obtener información de su empresa en tiempo real y en cualquier parte, ya sea para consultar stock y precios de productos, leer y responder un correo electrónico, o conversar directamente con algún cliente o proveedor y no depender de una computadora con acceso a internet para obtener esta información.

Considerando el crecimiento que está teniendo el mercado de aplicaciones móviles, el proyecto se enfoca en desarrollar nuevas funcionalidades para la aplicación Iduo y desarrollar una aplicación móvil e-commerce específicamente para el entorno Android[3]. Sin duda, esto nos obliga, como profesionales del sector tecnológico, a conocer los retos y posibilidades de este entorno, para lo cual se realizará un estudio de las herramientas y tecnologías a utilizar para luego comenzar el desarrollo de la aplicación cumpliendo con cada una de las metas definidas.

---

[1] *Tablet*: Es una computadora portátil de mayor tamaño que un teléfono, integrada en una pantalla táctil con la que se interactúa primariamente con los dedos, sin necesidad de teclado físico ni ratón. Tomado de: Wikipedia https://es.wikipedia.org/wiki/Tableta_(computadora) [25/08/2021].

[2] *Smartphone*: Es un tipo teléfono móvil construido sobre una plataforma informática móvil, con una mayor capacidad de almacenar datos y realizar actividades semejantes a una minicomputadora. Tomado de: Wikipedia https://es.wikipedia.org/wiki/Tel%C3%A9fono_inteligente [25/08/2021].

[3] Android: Es un sistema operativo basado en el kernel o núcleo de Linux diseñado principalmente para dispositivos móviles con pantalla táctil, como smartphones o tablets. Tomado de: Wikipedia https://es.wikipedia.org/wiki/Android [28/08/2021].

Capítulo II Situación actual del proyecto

Iduo S.A es una empresa de software radicada en la ciudad de General Pico, que actualmente cuenta con alrededor de 300 clientes en todo el país. La empresa se encarga del análisis, diseño y desarrollo de soluciones de software para el sector empresarial con el objetivo de mejorar los estándares administrativos. Provee una plataforma que soporta procesos de gestión complejos e integra múltiples áreas operativas, brindando información gerencial para la toma de decisiones; un sistema del tipo ERP[4]– CRM[5], comercializado bajo modalidad SaaS (*Software as a Service*). De esta forma, permite a los clientes simplificar la administración de los procesos de negocio mediante la integración de módulos informáticos que constituyen al sistema global, y que son específicos para cada tarea o actividad.

En el año 2017, con el propósito de hacer frente a la gran demanda de sus usuarios de realizar las operaciones en cualquier momento y lugar, la empresa decidió incorporarse al mundo de las aplicaciones móviles, desarrollando la aplicación móvil para Android, denominada Iduo. El objetivo fue proveer una herramienta que facilitara la manipulación y el procesamiento de información, brindando una solución que permita mejorar la eficiencia de la empresa, como así también ahorrar tiempo y realizar operaciones sin la necesidad de tener acceso a Internet. Así, la aplicación Iduo permite conectarse a la plataforma de software ERP (también provisto por Iduo), con el objetivo de realizar acciones como tomar pedidos, cargar devoluciones, gestionar productos, gestionar clientes. Una de las ventajas principales es que permite realizar dichas operaciones sin la necesidad de tener acceso a internet. De esta forma, una vez que el usuario cuenta con acceso a internet, exporta dichas entidades al sistema y la información se mantiene consistente entre ambos sistemas.

La continua mejora en los procesos de negocio, como así también el crecimiento en la demanda de los clientes de contar con nuevas funcionalidades y unidades de negocio, obligaron a la empresa a dedicar una mayor cantidad de tiempo en la aplicación, con el objetivo de hacer frente a estas nuevas necesidades.

Por lo tanto, tomaron la decisión de actualizar la aplicación de Iduo, comenzando con la refactorización, modificación de la estructura interna y el desarrollo de nuevas funcionalidades de la aplicación móvil, potenciando los servicios y beneficios que la misma ofrece a los usuarios.

Este nuevo proyecto se abordó en el marco de Práctica Profesional Supervisada (PPS en adelante), la cual se trató sobre la refactorización y el desarrollo de nuevas funcionalidades de la aplicación móvil. Dicha práctica se orientó principalmente a la refactorización de la aplicación Iduo, con el objetivo de mejorar los procesos; es decir, la modificación del diseño y la reestructuración de los módulos con los que cuenta la aplicación. Por otro lado, con la meta de lograr una mayor competitividad y brindar una herramienta más completa, se desarrollaron e integraron un conjunto de funcionalidades nuevas a la aplicación. Con estas

[4] El término ERP se refiere a Enterprise Resource Planning, que significa “sistema de planificación de recursos empresariales”. Estos programas se hacen cargo de distintas operaciones internas de una empresa, desde producción a distribución o incluso recursos humanos. Tomado de : HubSpot https://blog.hubspot.es/sales/diferencia-erp-crm [25/08/2021]

[5] Un CRM (Customer Relationship Management) es un software que permite a las empresas rastrear cada interacción con los leads, prospectos y clientes actuales. Tomado de: HubSpot https://blog.hubspot.es/sales/diferencia-erp-crm [25/08/2021]

nuevas funcionalidades se buscó proveer mayores beneficios a los usuarios, como así también mantener una mayor sincronización y consistencia con el sistema central.

De esta forma, con el desarrollo de la práctica se logró cumplir con los objetivos planteados anteriormente, desarrollando y poniendo a disposición de la empresa una aplicación más completa, con más funciones y una mejor calidad interna. Además, uno de los grandes beneficios obtenidos fue la distribución mediante Google Play, permitiendo llegar a un mayor número de clientes.

La aplicación móvil aportó un valor agregado a los servicios que la empresa brinda a sus clientes, abriendo camino al desarrollo de nuevas soluciones que permitan acceder a la información en cualquier momento y desde cualquier lugar. En este contexto, se decidió continuar con el desarrollo e integración de nuevas funcionalidades, dando origen al proyecto final de carrera.

Por lo tanto, el trabajo propuesto en el presente proyecto bajo la modalidad proyecto final de desarrollo trata sobre:

- Desarrollo de servicios que permitan la sincronización automática con el sistema central, manteniendo la consistencia de la información y liberando al usuario de realizar tareas manuales.
- La incorporación de un nuevo comprobante de factura electrónica, junto con el desarrollo de nuevas funcionalidades que permitan facilitar la manipulación y distribución del mismo.
- El desarrollo de una nueva aplicación e-commerce para Capitano Craft.

2.1 Justificación

En las últimas décadas, el continuo desarrollo tecnológico y el fácil acceso a equipos informáticos e internet en la actualidad, ha generado un cambio en la economía y en los procesos administrativos de las diferentes empresas a nivel mundial. Desde la incorporación de software de sistemas integrados de información en las empresas, las actuales formas de operar en los mercados y hasta el surgimiento de la empresa totalmente digital, han dado lugar a diversas tecnologías de información que siendo utilizadas por parte de los individuos y empresas, generan una serie de ventajas en ellos, tales como, la velocidad e irradiación de la información. En cuanto al comercio electrónico, surgieron nuevas regulaciones dispuestas por la Administración Federal de Ingresos Públicos (AFIP), cuyo objetivo es intensificar el uso de herramientas informáticas, destinadas a facilitar a los contribuyentes y/o responsables el cumplimiento de sus obligaciones fiscales, así como a optimizar las funciones de fiscalización y control de los gravámenes a su cargo. Así, las nuevas resoluciones generales tienden a la implementación de la factura electrónica. Dada estas nuevas exigencias, Iduo se vio obligado a incorporar esta herramienta en la aplicación móvil. Los objetivos de esta nueva incorporación son ampliar las herramientas de autorización de comprobantes, emitiendo toda la facturación por medios electrónicos y permitiendo registrar en las bases de datos de AFIP los comprobantes facturados en tiempo real, anulando costos de impresión, distribución, almacenamiento y adquisición de servicios web para el emisor e integrando el modelo con medios de pago electrónico, tal como mercado pago.

Por otro lado, muchos negocios y empresas se están empezando a dar cuenta que la reputación digital y el valor que la marca posee en internet es esencial para prolongar el crecimiento del negocio. Una visión global y a futuro muy positiva teniendo en cuenta el número de usuarios que cada día entran en Google o cualquier página web en busca de productos.

En este contexto, Capitano Craft, empresa que contrata los servicios de Iduo, se encontró con la necesidad de contar con un e-commerce. Las tiendas online son una opción precisa para conseguir más ventas, convirtiéndose en una de las opciones más eficaces y productivas para la venta de productos y/o servicios online.

A continuación se detallan algunas de las ventajas de contar con un e-commerce y cómo esta opción puede beneficiar a la empresa:

- **Compra inmediata**: La sociedad, y más si nos enfocamos en los procesos de compra, lo que están buscando es rapidez. Comprar el producto en el momento. El tiempo es muy importante y las nuevas generaciones están cada vez más acostumbradas a comprar online que a acudir a las tiendas de cercanía.
- **Información**: Para poder aprovechar esta oportunidad es necesario ofrecer la máxima información posible al usuario para que no tenga ninguna duda mientras navega por los productos de la web. En este sentido, ofrecer descripciones detalladas, cuidar el contenido de las categorías y colocar llamadas a la acción son conceptos fundamentales. Además, también se puede hacer uso de promociones y ofertas para llamar la atención y difundirlas por redes o grupos que frecuente el público objetivo.
- **Comodidad**: La comodidad es otro de los factores clave, pensar en la necesidad del usuario para ponerle las cosas fáciles. Acceder a la tienda online desde cualquier dispositivo ya es un deber y una obligación. Y lo es porque es eso lo que el público demanda y utiliza. Las búsquedas por móvil cada vez crecen más.
- **Disponibilidad en todo momento**: Una tienda abierta todos los días del año a todas horas. Esta es una de las grandes ventajas de crear una tienda online.

2.2 Objetivos del proyecto

El objetivo principal del proyecto es desarrollar nuevas funcionalidades para la aplicación Iduo, que permitan cumplir en mayor medida con las nuevas necesidades de los usuarios. Se utilizó como base el desarrollo realizado como Práctica Profesional Supervisada, en la cual se generó una nueva estructura funcional de la aplicación Iduo, que permitió optimizar procesos existentes, mejorar el diseño de la misma e incorporar nuevas funcionalidades que permitieran facilitar las tareas que se llevan a cabo. Es en este contexto que se decidió agregar nuevas herramientas a partir de esta estructura, con el fin de seguir aumentando el número de usuarios que utilizan la aplicación y aportar mayor valor para la empresa.

Por lo tanto, el trabajo que se propone en el presente proyecto se orienta a la incorporación de un nuevo tipo de comprobante (comprobante de factura electrónica) a la aplicación, con el objetivo de lograr una mayor competitividad y brindar una herramienta más completa.

Dicha mejora permite proveer a los usuarios operar con un documento comercial (factura) en formato digital que reemplaza al formato físico (en papel). También se deben desarrollar e integrar un conjunto de funcionalidades nuevas a la aplicación existente. Con estas nuevas funcionalidades se busca proveer de mayores beneficios a los usuarios, como así también mantener una mayor sincronización y consistencia con el sistema central.

Por otro lado, se pretende desarrollar una aplicación móvil de e-commerce para la empresa Capitano Craft, ofreciendo funciones claves como registro de usuarios, inicio de sesión, gestión de pedidos, compras y realización de pagos.

Cada una de las etapas realizadas dentro de la PPS son piezas fundamentales para lograr obtener los resultados esperados y poder cumplir con cada uno de los objetivos planteados.

Con el desarrollo de este proyecto se desea adquirir conocimientos en cuanto al diseño y construcción de una aplicación de android, comprendiendo su ciclo de vida y sus principales componentes. También se busca comprender y utilizar técnicas de procesamiento de fondo (segundo plano) y de almacenamiento persistente (tales como preferencias, archivos, bases de datos y proveedores de contenido), como así también procesar pagos electrónicos, gestionar carritos de compra, gestionar catálogos de productos.

2.2.1 Objetivos generales

Lo que se busca en esta nueva etapa es agregar herramientas en la aplicación que permitan a los usuarios realizar un mayor número de tareas. Con la incorporación del documento de factura electrónica, se pretende que el usuario pueda realizar operaciones como cargar, listar o visualizar dichos documentos de una forma más sencilla, intuitiva y rápida. Los objetivos son ampliar las herramientas de autorización de comprobantes, emitir la facturación por medios electrónicos, registrar en la base de datos de AFIP todos los comprobantes facturados en tiempo real, anular costos de impresión, distribución, almacenamiento e integración de medios de pago electrónico (en este caso, Mercado Pago).
Los beneficios que produce esta aplicación en el proceso de facturación son:

- Agilización de las operaciones.
- Generación del comprobante con la obtención del código de autorización electrónico en línea. La factura se almacena en la base de datos al momento de la venta.
- Integración con medios de pago disponibles en el mercado.
- Generación de PDF con los datos propios del documento (factura electrónica).
- Ejecución de acciones como imprimir o compartir el documento a otras personas.

Por otro lado, con el desarrollo de la aplicación móvil de Capitano Craft se pretende brindar una herramienta que permita a los usuarios realizar compras de forma electrónica. De esta forma, los principales objetivos son:

- Fortalecer la identidad de marca, destacando de la competencia al tener una aplicación móvil mediante la cual los usuarios puedan interactuar con la empresa de una manera que no se puede conseguir mediante otros canales.

- Lograr una mayor visibilidad, al estar disponible para miles de usuarios (distribución mediante Google Play).
- Generar un nuevo canal de venta desde donde los usuarios podrán realizar compras de los productos como lo harían desde una sucursal física o una tienda en línea.
- Aumentar la velocidad al contar con contenido descargado y preparado para su visualización.
- Utilizar las notificaciones.
- Lograr disponibilidad offline, permitiendo que los usuarios puedan navegar por el catálogo de los productos, noticias y promociones en cualquier momento que lo deseen, cuenten o no con conexión a Internet.
- Alcanzar un mayor grado de fidelización de los clientes, fortaleciendo así el vínculo con la empresa y pudiendo utilizarla como canal directo de compra

2.2.2 Objetivos específicos

Para llevar a cabo estas tareas se utiliza como base la estructura organizacional y funcional de la aplicación desarrollada durante la PPS, por lo que se considera que esta etapa inicial es fundamental para poder cumplir con los objetivos que se plantean. Las tareas que se desarrollaron y que fueron llevadas a cabo con éxito fueron las siguientes:

- Refactorización de la aplicación móvil existente; en esta etapa se modificó la organización y ubicación de los distintos elementos con los que contaba la aplicación, logrando una consistencia entre la aplicación y los módulos con los que cuenta el sistema central. Además, se aplicaron conceptos de diseño responsivo y técnicas de ingeniería de software para mejorar la apariencia, comportamiento y la calidad de código, eliminando duplicación de código y mejorando el diseño de la interfaz de usuario.
- Desarrollo de procesos para validar y actualizar de forma automática los permisos de un usuario en el sistema central, logrando un mayor control sobre los accesos a las funcionalidades del sistema.
- Desarrollo de proceso para capturar fotos desde la aplicación, agilizando la carga de nuevos comprobantes.

A continuación se explican las diferentes actividades que se planificaron para alcanzar los objetivos planteados en el capítulo anterior. En los capítulos siguientes se detallan cada uno de los desarrollos correspondientes.

Desarrollo de nuevas funcionalidades

A. **Optimización de las imágenes**
   Desarrollar un proceso que permita optimizar las imágenes que manipula la aplicación, con el objetivo de optimizar la lectura, procesamiento y subida al sistema central.
B. **Importación automática de los datos**

Desarrollar servicios que permitan la importación automática de datos del sistema central, con el objetivo de mantener actualizada la base de datos de la aplicación. De esta forma, se logra cierta integridad entre la base de datos del sistema central y de la aplicación. Por lo tanto, también es necesario desarrollar mecanismos de sincronización que permitan mantener dicha integridad. Este proceso se debe ejecutar cada cierto tiempo, independientemente si la aplicación está o no en ejecución, por lo que debe ser transparente para el usuario.

C. **Exportación automática de los datos**
   Desarrollar servicios que permitan la exportación automática de datos hacia el sistema central. De esta forma, se le quita la responsabilidad al usuario de exportar los datos al sistema central manualmente, simplificando tareas repetitivas.

Gestión de nuevo comprobante electrónico, la factura

A. **Migración de la base de datos**
   Definir un proceso de migración de base de datos que permita agregar las tablas correspondientes en la base de datos, como así también los atributos necesarios.
B. **Diseño e implementación para manipular nuevo comprobante**
   Desarrollar un nuevo módulo de facturación, con el fin que se puedan gestionar facturas electrónicas desde el dispositivo (alta, baja, modificación y consulta) y exportarlos al sistema web.
C. **Generar PDF de Factura Electrónica**
   Desarrollar una nueva funcionalidad para generar un documento PDF del comprobante, utilizando diseños de facturas A, B o C. De esta forma, el usuario puede compartir o enviar este documento a quien corresponda.

Desarrollo de e-commerce

A. **Gestión de pedidos y pagos**
   Diseñar un catálogo de productos que permita a los usuarios conocer los productos que ofrece la empresa, navegar entre categorías o visualizar detalles de cada producto. Además, se debe agregar un carrito de compras, que permita gestionar los pedidos que realizan los usuarios y facilitar la finalización de la compra. Finalmente, para el pago de los pedidos se debe desarrollar un módulo que permita utilizar los servicios de Mercado Pago, con el objetivo de que los usuarios puedan realizar el pago electrónico de pedidos a través de la aplicación.
B. **Gestionar inicio de sesión utilizando diferentes plataformas**
   Desarrollar dos opciones para llevar a cabo el inicio y el registro de usuarios a la aplicación; la creación de una cuenta utilizando el correo electrónico o el inicio de sesión de Facebook. Esto permite gestionar información personalizada para cada individuo, tales como las compras realizadas o el carrito de compras. Se deben tener en cuenta procesos de validación de cuentas al momento de registrar nuevos usuarios, con el fin de validar la existencia del mismo.

C. Gestionar notificaciones para monitorear el estado de un pedido

Cuando un usuario exporta un pedido al sistema, este se encarga de crear un turno y asociarlo a dicho pedido. Esta entidad contiene los datos necesarios para identificar el pedido correspondiente, como así también el estado en el que se encuentra. Por lo tanto, se debe desarrollar un servicio que consulte constantemente al sistema los turnos generados e integrar un sistema de notificaciones que permita mantener informado al usuario sobre los cambios en el estado del mismo.

Capítulo III Marco teórico

En este capítulo se definen los conceptos básicos que se utilizaron en el desarrollo del proyecto con el objetivo de brindar una mayor comprensión del documento, como así también se introducen los principales componentes involucrados en el desarrollo de una aplicación en Android.

3.1 Android

Android es un sistema operativo móvil basado en núcleo Linux, un núcleo de sistema operativo libre, gratuito y multiplataforma. Fue diseñado para dispositivos móviles con pantalla táctil, como *smartphones*, *tablets*, relojes inteligentes (Wear OS), automóviles (Android Auto) y televisores (Android TV) [1].

El sistema permite programar aplicaciones en una variación de Java llamada Dalvik. El sistema operativo proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas o la agenda) de una forma muy sencilla en un lenguaje de programación muy conocido como es Java [2].

Software Libre

Una de las mejores características de este sistema operativo es que es completamente libre. Es decir que, el uso del mismo es totalmente gratuito, ya sea para programar partes en dicho sistema o incluirlo en un teléfono. Y esto lo hace muy popular entre fabricantes y desarrolladores, ya que los costos para lanzar un teléfono o una aplicación son muy bajos.

Cualquier usuario puede bajar el código fuente, inspeccionarlo, compilarlo e incluso cambiarlo. Esto da seguridad a los usuarios, ya que el código abierto permite detectar fallos más rápidamente. Y también a los fabricantes, pues pueden adaptar mejor el sistema operativo a los terminales [3].

Un poco de historia

Android era un sistema operativo para móviles prácticamente desconocido hasta que en 2005 Google lo compró. Hasta noviembre de 2007 sólo hubo rumores, pero en esa fecha se lanzó la Open Handset Alliance, formada por muchos fabricantes de teléfonos móviles, chipsets y Google, que proporcionó la primera versión de Android, junto con el SDK (del inglés *Software Development Kit*) para que los programadores empezaran a crear sus aplicaciones para este sistema.

Aunque los inicios fueron un poco lentos, debido a que se lanzó antes el sistema operativo que el primer móvil, rápidamente se convirtió en el sistema operativo de móviles más vendido del mundo, situación que se alcanzó en el último trimestre de 2010.

En febrero de 2011 se anunció la versión 3.0 de Android, llamada con nombre en clave Honeycomb, que está optimizado para *tablets* en lugar de teléfonos móviles. Por tanto Android ha transcendido los teléfonos móviles para extenderse a dispositivos más grandes [4].

Arquitectura de la plataforma

Android es un sistema con distintas capas de software de código abierto, donde se incluye un sistema operativo, *middleware*[6] y aplicaciones básicas para el usuario [5]. En la Figura 1 se muestran los componentes principales de la plataforma Android.

Figura 1: Pila de software de Android.
Tomado de: Android Developers https://developer.android.com/. [20/05/2021]

[6] Middleware es software que se sitúa entre un sistema operativo y las aplicaciones que se ejecutan en él. Básicamente, funciona como una capa de traducción oculta para permitir la comunicación y la administración de datos en aplicaciones distribuidas. Tomado de: https://azure.microsoft.com/es-es/overview/what-is-middleware/ [21/08/2021]

Kernel de Linux

La base de la plataforma Android es el kernel de Linux. Por ejemplo, el tiempo de ejecución de Android (ART) se basa en el kernel de Linux para funcionalidades subyacentes, como la generación de subprocesos y la administración de memoria de bajo nivel.

El uso del kernel de Linux permite que Android aproveche funciones de seguridad claves y, al mismo tiempo, permite a los fabricantes de dispositivos desarrollar controladores de hardware para un kernel conocido [6].

Capa de abstracción de hardware

La capa de abstracción de hardware (HAL) brinda interfaces estándares que exponen las capacidades de hardware del dispositivo al marco de trabajo de la API de Java de nivel más alto. La HAL consiste en varios módulos de biblioteca y cada uno de estos implementa una interfaz para un tipo específico de componente de hardware, como el módulo de la cámara o de Bluetooth. Cuando el marco de trabajo de una API realiza una llamada para acceder al hardware del dispositivo, el sistema Android carga el módulo de biblioteca para el componente de hardware en cuestión [7].

Tiempo de ejecución de Android

Para los dispositivos con Android 5.0 (nivel de API[7] 21) o versiones posteriores, cada app o *application* (en español aplicación) ejecuta sus propios procesos con sus propias instancias del tiempo de ejecución de Android (ART o *Android Runtime*). El ART está escrito para ejecutar varias máquinas virtuales en dispositivos de memoria baja ejecutando archivos DEX, un formato de código de bytes diseñado especialmente para Android y optimizado para ocupar un espacio de memoria mínimo. Crea cadenas de herramientas, como Jack, y compila fuentes de Java en código de bytes DEX que se pueden ejecutar en la plataforma Android.

Estas son algunas de las funciones principales del ART:

- Compilación *ahead-of-time* (AOT) y *just-in-time* (JIT).
- Recolección optimizada de elementos no utilizados (GC).
- En Android 9 (nivel de API 28) y versiones posteriores, se convierten los archivos de formato ejecutable (DEX) de un paquete de aplicaciones a un código de máquina más compacto.
- Esto mejora la compatibilidad con la depuración, el generador de perfiles de muestras dedicado, las excepciones de diagnóstico detalladas y los informes de fallos, y la capacidad de establecer puntos de control para supervisar campos específicos.

[7] La API o Application Programming Interface (interfaz de programación de aplicaciones) abre la puerta a que dos componentes de software distintos se comuniquen para que ambos obtengan información del contrario. En Android se dispone de numerosas APIs que ofrecen a los desarrolladores la posibilidad de gestionar funcionalidades del dispositivo a través de sus propias aplicaciones. Tomado de https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces [25/08/2021]

Antes de Android 5.0 (nivel de API 21), Dalvik era el entorno de ejecución del sistema operativo. Si una app se ejecuta bien en el ART, también debe funcionar en Dalvik, pero es posible que no suceda lo contrario.

En Android, también se incluye un conjunto de bibliotecas de entorno de ejecución centrales que proporcionan la mayor parte de la funcionalidad del lenguaje de programación Java; se incluyen algunas funciones del lenguaje Java 8, que usa el marco de trabajo de la API de Java [8].

Bibliotecas C/C++ nativas

Muchos componentes y servicios centrales del sistema Android, como el ART y la HAL, se basan en código nativo que requiere bibliotecas nativas escritas en C y C++. La plataforma Android proporciona API del marco de trabajo de Java para exponer la funcionalidad de algunas de estas bibliotecas nativas a las apps. Por ejemplo, se puede acceder a *OpenGL-ES* a través de la API de OpenGL de Java del marco de trabajo de Android para agregar a la aplicación compatibilidad con los dibujos y la manipulación de gráficos 2D y 3D.

Si se desarrolla una aplicación que requiere C o C++, se puede utilizar el NDK de Android para acceder a algunas de estas bibliotecas de plataformas nativas directamente desde código nativo [9].

Marco de trabajo de la API de Java

Todo el conjunto de funciones del SO Android está disponible mediante APIs escritas en el lenguaje Java. Estas APIs son los cimientos que se necesitan para crear aplicaciones de Android simplificando la reutilización de componentes del sistema y servicios centrales y modulares, como los siguientes:

- Un sistema de vista enriquecido y extensible que se puede usar para compilar la IU de una app; se incluyen listas, cuadrículas, cuadros de texto, botones e incluso un navegador web integrable.
- Un administrador de recursos que brinda acceso a recursos sin código, como strings localizadas, gráficos y archivos de diseño.
- Un administrador de notificaciones que permite que todas las apps muestren alertas personalizadas en la barra de estado.
- Un administrador de actividad que administra el ciclo de vida de las apps y proporciona una pila de retroceso de navegación común.
- Proveedores de contenido que permiten que las apps accedan a datos desde otras apps, como la app de Contactos, o compartan sus propios datos.

Los desarrolladores tienen acceso total a las mismas API del marco de trabajo que usan las aplicaciones del sistema Android [10].

Apps del sistema

En Android se incluye un conjunto de aplicaciones centrales para correo electrónico, mensajería SMS, calendarios, navegación en Internet y contactos, entre otros elementos. Las aplicaciones incluidas en la plataforma no tienen un estado especial entre las aplicaciones que el usuario elige instalar; por ello, una aplicación externa se puede convertir en el navegador web, el sistema de mensajería SMS o, incluso, el teclado predeterminado del usuario (existen algunas excepciones, como la app *Settings* del sistema).

Las aplicaciones del sistema funcionan como aplicaciones para los usuarios y brindan capacidades claves a las cuales los desarrolladores pueden acceder desde sus propias aplicaciones. Por ejemplo, si en una aplicación se intenta entregar un mensaje SMS, no es necesario que se desarrolle esa funcionalidad; como alternativa, se puede invocar la aplicación de SMS que ya está instalada para entregar un mensaje al receptor que se especifique.

3.2 Componentes de Android

Existe una serie de elementos claves que resultan imprescindibles para desarrollar aplicaciones en Android. En este apartado se realiza una descripción inicial de algunos de los más importantes, debido a la influencia y el papel que tienen en el trabajo realizado. A lo largo del desarrollo del proyecto, se describen con más detalle las clases Java que implementan algunos de estos componentes [11].

Vista (o *View*)

Las vistas son los elementos que componen la interfaz de usuario de una aplicación, permitiendo que el usuario pueda interactuar con la misma. Dentro de las vistas podemos distinguir varios elementos, siendo los más comunes las etiquetas, los cuadros de texto, las casillas y los botones, entre otros. Todas las vistas van a ser objetos descendientes de la clase *View*, y por tanto, pueden ser definidas utilizando código Java. Sin embargo, lo habitual es definir las vistas utilizando un archivo XML (*Extensible Markup Language*, o en español, Lenguaje de Marcado Extensible) y dejar que el sistema cree los objetos por nosotros a partir de este archivo. Esta forma de trabajar es muy similar a la definición de una página web utilizando código HTML (*HyperText Markup Language*, o en español, Lenguaje de Marcas de Hipertexto).

Las propiedades de las vistas son las características de cada uno de los elementos. Dentro de estas características podemos encontrar algunas que son útiles sólo para algunas vistas y otras características que serán comunes e importantes para otras vistas.

Actividad (o *Activity*)

Las actividades son las pantallas que vemos en un dispositivo móvil, que está conformada por dos archivos, un archivo XML (parte gráfica) y un archivo .Java (la parte lógica) de la ventana.

La clase *Activity* es un componente clave de una aplicación para Android, y la forma en que se inician y se crean las actividades es una parte fundamental del modelo de aplicación de la plataforma. A diferencia de los paradigmas de programación en los que las apps se inician con un método *main*, el sistema Android inicia el código en una instancia de *Activity* invocando métodos de devolución de llamada específicos que corresponden a etapas específicas de su ciclo de vida.

Servicios (o Services)

Un servicio es un componente de una aplicación el cual se ejecuta en segundo plano (*background*), este puede realizar las mismas acciones que un *Activity* con la única diferencia que este no proporciona ningún tipo de interfaz de usuario.

Los servicios son útiles para acciones que se requieran realizar durante un tiempo en background, sin tener en cuenta lo que está en pantalla tales como: Ejecutar transacciones de red, reproducir música, ejecutar archivos, ejecutar procesos, todo desde *background*.

Un servicio es ejecutado desde el hilo principal, este no crea un hilo propio y no corre en un proceso separado, esto significa que si un servicio realiza un trabajo intensivo en el CPU es necesario crear un nuevo Hilo dentro del servicio para poder reducir el riesgo de un ANR (*Application Not Responding*) en el dispositivo.

Proveedor de contenidos (o *Content Provider*)

El Proveedor de contenidos se utiliza para proporcionar datos a la aplicación, esto ayuda a compartir datos sin preocuparse por el mecanismo de almacenamiento, lo que importa es la información que puede proporcionar para una o varias tareas que necesitan los datos en la aplicación.

Intención (o Intent)

Los *Intents* son objetos que sirven para mandar mensajes. Además, se puede pedir ejecutar una acción a otro componente con el mismo intent. Dentro de los *Intents* existen diferentes tipos dependiendo de lo que se necesite.

3.3 Manejo de tareas asíncronas en Android

Cuando una aplicación se inicia, lo que sucede de manera muy técnica, es que se crea un proceso. Entonces todo lo que se ejecuta dentro de esa aplicación es parte del mismo proceso de ejecución. Un hilo es otra forma de dividir el trabajo, un proceso puede tener

varios hilos de ejecución, el hilo lo podemos tomar como una secuencia de control dentro de cualquier proceso y este ejecuta lo que se indique de manera independiente.

Cuando se lanza una nueva aplicación el sistema Android crea un nuevo hilo de ejecución (*thread*) para esta aplicación conocida como hilo principal. Este hilo es muy importante dado que se encarga de atender los eventos de los distintos componentes, por lo que también se lo conoce como hilo de la interfaz de usuario.

El sistema no crea un hilo independiente cada vez que se crea un nuevo componente de la aplicación. Es decir, todas las actividades y servicios de una aplicación son ejecutados por el hilo principal. Por lo tanto, al momento de realizar operaciones complejas como respuesta a una interacción de usuario, se deben tomar ciertas medidas. Por ejemplo, si una aplicación pretende descargar datos de Internet desde el hilo de ejecución principal, éste quedará bloqueado a la espera de que termine la descarga. Esto implica que a partir de ese momento el hilo principal no podrá atender nuevos eventos del usuario, dando la impresión al usuario de que la aplicación ha dejado de funcionar. Además si el hilo principal es bloqueado más de 5 segundos, el sistema mostrará un cuadro de diálogo al usuario "La aplicación no responde" para que decida forzar el cierre de la aplicación o esperar a que termine.

Por otro lado, las herramientas de la interfaz de usuario de Android han sido diseñadas para ser ejecutadas desde un único hilo de ejecución, el hilo principal. Esto implica que no se permite manipular la interfaz de usuario desde otros hilos de ejecución.

Para solucionar estos inconvenientes, Android cuenta con la clase *AsyncTask*, la cual permite realizar tareas en segundo plano de una forma organizada y legible. La forma básica de utilizar la clase *AsyncTask* consiste en crear una nueva clase que extienda de ésta y sobrescribir varios de sus métodos, entre los que se reparte la funcionalidad de tarea a realizar. Estos métodos son los siguientes:

- *onPreExecute()*. Se ejecutará antes del código principal de la tarea a realizar. Se suele utilizar para preparar la ejecución de la tarea, inicializar la interfaz.
- *doInBackground()*. Contendrá el código principal de la tarea a realizar.
- *onProgressUpdate()*. Se ejecutará cada vez que se llame al método *publishProgress*() desde el método *doInBackground*().
- *onPostExecute()*. Se ejecutará cuando finalice la tarea, o dicho de otra forma, tras la finalización del método *doInBackground*().
- *onCancelled()*. Se ejecutará cuando se cancele la ejecución de la tarea antes de su finalización normal.

El método *doInBackground()* se ejecuta en un hilo secundario (por tanto no se puede interactuar con la interfaz). Sin embargo todos los demás se ejecutan en el hilo principal, lo que quiere decir que dentro de estos métodos se puede hacer referencia directa a los componentes del hilo principal y actualizar la interfaz. Por su parte, dentro del método *doInBackground()* se puede invocar periódicamente al método *publishProgress()* para que automáticamente desde el método *onProgressUpdate()* se actualice la interfaz, en caso de que sea necesario [12].

3.4 Bases de datos

La base de datos se define como un conjunto auto descriptivo de registros integrados. Una base de datos es auto descriptiva porque contiene una descripción de su propia estructura, a esto se le llama diccionario de datos. Una base de datos es un conjunto de

registros integrados por que contiene archivos de datos del usuario que se relacionan entre sí, a esta relación se le llama índices.

Una base de datos es un modelo de un modelo porque no es un modelo de la realidad ni parte de ella, sino es un modelo del modelo del usuario, es decir, es el modelo que el usuario percibe de su negocio (clientes, empleados, productos, etc.) y la representación de los hechos que relacionan esas características.

El objetivo de la base de datos es dar seguimiento a datos tales como órdenes, clientes, empleos, empleados, llamadas telefónicas, entre otros aspectos de interés para el negocio.

EL modelado de la base de datos se puede representar mediante el modelo entidad –relación creada por Peter Chen en 1976.

Los elementos del modelo entidad – relación son: entidades, atributos, identificadores y relaciones.

- **Entidad**: es aquello que se quiere dar seguimiento como por ejemplo, empleado, cliente, vendedor, producto.
- **Atributos**: son las propiedades de las entidades que describe sus características. Por ejemplo, nombre del empleado, fecha de contrato.
- **Identificadores**: son atributos que identifican las instancias de una entidad. Por ejemplo, la instancia empleado puede ser identificado por el número de empleado o el nombre del empleado.
- **Relaciones**: las entidades se pueden asociar con otras mediante relaciones. Existes tres tipos de relaciones: 1..1 (uno a uno), 1..* (uno a muchos) y *..* (muchos a muchos).

3.4.1 Bases de datos SQLite en Android

SQLite es un motor de bases de datos de código abierto ligero, que se caracteriza por mantener el almacenamiento de información persistente de forma sencilla. A diferencia de otros sistemas gestores de bases de datos como MySQL, SQL Server y Oracle DB, SQLite tiene las siguientes ventajas:

- No requiere el soporte de un servidor: SQLite no ejecuta un proceso para administrar la información, sino que implementa un conjunto de librerías encargadas de la gestión.
- No necesita configuración: Libera al programador de todo tipo de configuraciones de puertos, tamaños, ubicaciones, etc.
- Usa un archivo para el esquema: Crea un archivo para el esquema completo de una base de datos, lo que permite ahorrarse preocupaciones de seguridad, ya que los datos de las aplicaciones Android no pueden ser accedidos por contextos externos.
- Es de código abierto: Está disponible para el dominio público de los desarrolladores al igual que sus archivos de compilación e instrucciones de escalabilidad.

Es por eso que SQLite es una tecnología muy utilizada para los dispositivos móviles. El principal motivo es que solo requiere un simple archivo para almacenar los datos, ya que la lógica de funcionamiento debe ser implementada por la plataforma que desee interactuar

con los datos. En el caso de la plataforma Android, el kit de desarrollo de software, o SDK incluye soporte completo para SQLite [13].

Object Relational Mapping (ORM)

Un ORM es un modelo de programación que permite mapear las estructuras de una base de datos relacional (algunas son SQL Server, MySQL, SQLite), en adelante RDBMS (Relational Database Management System), sobre una estructura lógica de entidades con el objeto de simplificar y acelerar el desarrollo de nuestras aplicaciones.

Las estructuras de la base de datos relacional quedan vinculadas con las entidades lógicas o base de datos virtual definida en el ORM, de tal modo que las acciones CRUD (Create, Read, Update, Delete) a ejecutar sobre la base de datos física se realizan de forma indirecta por medio del ORM.

La consecuencia más directa que se infiere del párrafo anterior es que, además de "mapear", los ORMs tienden a "liberarnos" de la escritura o generación manual de código SQL (Structured Query Language) necesario para realizar las queries o consultas y gestionar la persistencia de datos en el RDBMS. Así, los objetos o entidades de la base de datos virtual creada en nuestro ORM podrán ser manipulados por medio de algún lenguaje de nuestro interés según el tipo de ORM utilizado, por ejemplo, LINQ sobre Entity Framework de Microsoft. La interacción con el RDBMS quedará delegada en los métodos de actualización correspondientes proporcionados por el ORM. Los ORMs más completos ofrecen servicios para gestionar la persistencia de todos los cambios en los estados de las entidades, previo seguimiento o tracking automático, sin escribir una sola línea de SQL [14].

ORMs en Android

En los primeros desarrollos de software, un programa debía acceder físicamente al disco para poder grabar los datos, y por lo tanto se debía implementar toda la lógica de la base de datos en el programa. Posteriormente, la aparición de las bases de datos relacionales, permitió, a través de consultas *Transact-SQL*, acceder a aquellos datos necesarios para el programa, pero realizando las consultas manualmente dentro del mismo, y por lo tanto, no se aprovechaba precisamente lo que hace fuerte a los lenguajes de programación orientados a objetos, que es el uso de objetos en sus interacciones. Gracias a los ORMs, gran parte de ese trabajo se realiza de una manera más sencilla, ya que una de sus principales funcionalidades es evitar tener que acceder a la capa de datos, y por lo tanto no realizar la conversión de los objetos en tipos de datos primitivos, para mantener la persistencia de objetos.

Específicamente Android, permite olvidarse de la compleja tarea del uso directo de SQLite, en el que se debe tener especial cuidado a la hora de guardar registros, debido a que no tiene tipo de datos (permite guardar una cadena de texto en un campo de tipo numérico, con los posibles problemas que pueden generar ese tipo de acciones). Además, permite la reutilización de código, es decir, poder realizar llamadas a los métodos del objeto de datos desde diferentes partes del proyecto, o en proyectos diferentes, y todo esto manteniendo la lógica de negocio en los propios objetos de datos definidos [15].

Sus principales ventajas son las siguientes:

- Desarrollo fácil y rápido.
- Abstracción de la Base de Datos mapeada.
- Capa de seguridad implementada, que evita los ataques SQL injections (inserción de código malicioso en variables incrustadas en sentencias Transact-SQL).
- Se complementa con el patrón de diseño MVC (Model-View-Controller).

3.5 Diseño responsivo en Android

El diseño responsivo (también diseño adaptativo o adaptable), es una filosofía de diseño y desarrollo cuyo objetivo es adaptar la apariencia de las vistas de una aplicación al dispositivo que se esté utilizando para accederlas. Los dispositivos Android vienen de diversas formas y tamaños, por lo que el diseño de una aplicación debe ser flexible. Es decir que, en lugar de definir un diseño con dimensiones rígidas para un tamaño de pantalla y una relación de aspecto determinados, el diseño debería ajustarse a pantallas de diferentes tamaños y orientaciones. Esta técnica pretende que con un único diseño, todo se vea correctamente en cualquier dispositivo, optimizando la aplicación, mejorando la experiencia del usuario y reduciendo los costos de creación y mantenimiento de múltiples diseños [16].

Material Design

Material Design es un sistema de diseño creado por Google con el fin de ser capaz de adaptarse a múltiples dispositivos y plataformas. El objetivo es incorporar coherencia estética y funcional de forma progresiva a todos sus productos, incluyendo las aplicaciones web y móviles, con la finalidad de crear una experiencia similar en todas sus plataformas. Esta filosofía se enfoca en un diseño más limpio, en el que predominan animaciones y transiciones de respuesta, el relleno y los efectos de profundidad tales como la iluminación y las sombras.

Material Design fue anunciado en 2014, junto con el lanzamiento de la versión 5.0 de Android Lollipop y desde entonces se ha ido implementando en cada una de sus aplicaciones para crear un sistema que ofrezca la misma experiencia de usuario en todas las plataformas y dispositivos. Google ha facilitado el trabajo a desarrolladores y diseñadores externos para que incorporen Material Design a sus aplicaciones a través de multitud de herramientas, recursos y documentación procedentes de la guía de diseño Google Material Theme [17].

En la Figura 2 se muestran las principales herramientas que permiten a través de sus guías y componentes incorporar los conceptos de Material Design al proyecto.

Figura 2: Herramientas de Google Material Design.
Tomado de: Google Material Design. https://material.i [24/08/2021]

3.6 Servicios Web: arquitectura REST

Los servicios web facilitan el intercambio de datos entre aplicaciones implementando un conjunto de protocolos y estándares que normalizan esta comunicación. Este tipo de tecnología permite que software realizado en diferentes lenguajes de programación, y funcionando en distintas plataformas, puedan intercambiar datos y hacer uso de ellos, usando estándares de interoperabilidad. A continuación se describen brevemente los principales estándares para la creación y consumo de servicios web , entrando luego más en profundidad en la arquitectura REST [18].

SOAP

Siglas que corresponden a *Simple Object Access Protocol*. Se define como un protocolo estándar, que posibilita que dos objetos en diferentes procesos pueden comunicarse mediante mensajes codificados en XML y transportados por un protocolo de transporte. No define un medio de transporte de los mensajes, pero puede asociarse a protocolos existentes como HTTP (*Hypertext Transfer Protocol*), FTP (*File Transfer Protocol*), o SMTP (*Simple Mail Transfer Protocol*).

XML

El *eXtensible Markup Language*, es un meta-lenguaje (lenguaje usado para describir a otros lenguajes) que permite definir lenguajes de marcado adaptados o enfocados a usos concretos. Los archivos de este tipo tendrán una extensión .xml y un tipo de MIME (estándar para el intercambio de todo tipo de archivos a través de internet) application/xml, text/xml.

WSDL

Web Services Description Language. Se trata de un formato XML para describir servicios web. Se basa en XML y se centra en describir los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo.

JSON

JavaScript Object Notation, es un formato ligero utilizado para el intercambio de datos, y se trata de un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso del formato XML. Los archivos de este tipo tendrán una extensión .json y un tipo de MIME application/json.

REST

Acrónimo de *REpresentational State Transfer*, es un estilo de arquitectura software que define un conjunto de principios, por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, además de cómo se acceden a dichos recursos y cómo se transfieren por el protocolo de transporte HTTP hacia clientes escritos en diversos lenguajes. En el siguiente apartado, se describen los principios que define la arquitectura REST, y que permiten abstraer los elementos de dicha arquitectura dentro de un sistema *hypermedia* distribuido.

Métodos HTTP de forma explícita

Es una de las principales características de un servicio web REST, ya que proporciona una serie de métodos que permiten a la aplicación cliente realizar operaciones sobre determinados recursos del servidor web [19]:

- GET: obtiene un recurso del servidor.
- PUT: actualiza o cambia el estado de un recurso del servidor.
- POST: creación de un recurso en el servidor.
- DELETE: elimina un recurso del servidor.
- HEAD: comprueba si cambia el contenido en un servidor.

Arquitectura cliente/servidor que no mantiene el estado

Al tratarse de una tecnología en constante crecimiento, se debe controlar la escalabilidad del servidor web para satisfacer el número de peticiones. Debido a esta premisa, el servidor no guarda el estado (*stateless*) o contexto de ninguna petición previa del cliente al servidor, por lo que toda petición para el acceso a los recursos del mismo deberá incluir los parámetros y datos necesarios para procesar y devolver la respuesta.

Uso de URIs a modo de directorios

Permite definir un sistema de sintaxis universal, que posibilite la identificación de cada recurso de manera individual mediante su URI o identificador único (*Uniform Resource Identifier*) asociado. Toda URI deberá regirse por algunas de las siguientes reglas básicas:

- Una URI no debe implicar acciones, por lo tanto deben evitarse el uso de verbos.
- Cada URI debe identificar de manera única a cada recurso.
- Independencia de formato.
- Mantener una jerarquía lógica.

Uso de los formatos XML y JSON para la transferencia de datos

Partiendo de la base de que toda comunicación en internet se mueve a través de recursos, y que estarán representados en un determinado formato, la utilización de los formatos XML y JSON para la transferencia de datos posibilitó el uso del servidor web desde diferentes clientes, escritos en distintos lenguajes de programación, y funcionando en diferentes plataformas.

3.6.1 Retrofit como Cliente REST

Retrofit es un cliente REST para Android y Java, desarollado por Square. Permite hacer peticiones GET, POST, PUT, DELETE y HEAD, gestionar diferentes tipos de parámetros y parsear automáticamente la respuesta a un POJO[8]. Como la mayoría del software de código abierto, Retrofit fue construido encima de algunas otras librerías y herramientas poderosas. Hace uso de *OkHttp* (del mismo desarrollador) para manejar peticiones de red [20]. La Figura 3 muestra el funcionamiento básico de Retrofit.

[8] POJO (acrónimo de *Plain Old Java Object*) es una sigla utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un framework en especial. Tomado de: https://www.it-swarm-es.com/es/java/cual-es-la-diferencia-entre-pojo-plain-old-java-object-y-dto-data-transfer-object/967128342/ [25/08/2021]

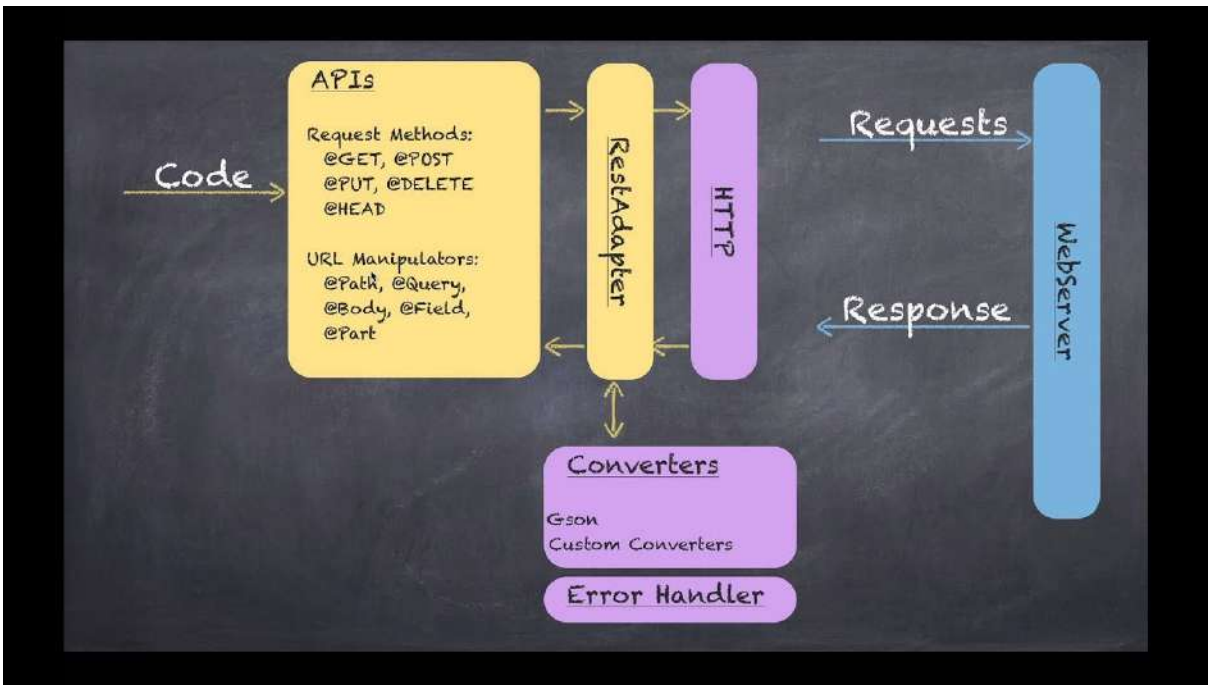

Figura 3: Descripción del funcionamiento de Retrofit.
Fuente: Daniel Álvarez. https://alvarez.tech/tutorials/retrofit/ [24/08/2021]

3.7 Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado (UML) fue creado para forjar un lenguaje de modelado visual común y semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de sistemas de software complejos, tanto en estructura como en comportamiento. UML tiene aplicaciones más allá del desarrollo de software, p. ej., en el flujo de procesos en la fabricación [21].

Es comparable a los planos usados en otros campos y consiste en diferentes tipos de diagramas. En general, los diagramas UML describen los límites, la estructura y el comportamiento del sistema y los objetos que contiene.

UML no es un lenguaje de programación, pero existen herramientas que se pueden usar para generar código en diversos lenguajes usando los diagramas UML. Este lenguaje guarda una relación directa con el análisis y el diseño orientados a objetos.

Hay muchos paradigmas o modelos para la resolución de problemas en la informática, que es el estudio de algoritmos y datos. Hay cuatro categorías de modelos para la resolución de problemas: lenguajes imperativos, funcionales, declarativos y orientados a objetos. En los lenguajes orientados a objetos, los algoritmos se expresan definiendo 'objetos' y haciendo que los objetos interactúen entre sí. Esos objetos son cosas que deben ser manipuladas y existen en el mundo real. Pueden ser edificios, artefactos sobre un escritorio o seres humanos.

Los lenguajes orientados a objetos dominan el mundo de la programación porque modelan los objetos del mundo real. UML es una combinación de varias notaciones orientadas a objetos: diseño orientado a objetos, técnica de modelado de objetos e ingeniería de software orientada a objetos.

Tipos de diagramas UML

UML usa elementos y los asocia de diferentes maneras para generar diagramas que representan aspectos estáticos o estructurales de un sistema, y diagramas de comportamiento, que captan los aspectos dinámicos de un sistema.

3.7.1 Diagramas UML estructurales

- Diagrama de clases: es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos. El diagrama UML más comúnmente usado, y la base principal de toda solución orientada a objetos.
- Diagrama de componentes: muestra la relación estructural de los elementos del sistema de software, muy frecuentemente empleados al trabajar con sistemas complejos con componentes múltiples. Los componentes se comunican por medio de interfaces.
- Diagrama de implementación: ilustra el hardware del sistema y su software. Útil cuando se implementa una solución de software en múltiples máquinas con configuraciones únicas.
- Diagrama de paquetes: hay dos tipos especiales de dependencias que se definen entre paquetes; la importación de paquetes y la fusión de paquetes. Los paquetes pueden representar los diferentes niveles de un sistema para revelar la arquitectura. Se pueden marcar las dependencias de paquetes para mostrar el mecanismo de comunicación entre niveles.

3.7.2 Diagramas UML de comportamiento

- Diagramas de actividades: flujos de trabajo de negocios u operativos representados gráficamente para mostrar la actividad de alguna parte o componente del sistema. Los diagramas de actividades se usan como una alternativa a los diagramas de máquina de estados.
- Diagrama de secuencia: muestra cómo los objetos interactúan entre sí y el orden de la ocurrencia. Representan interacciones para un escenario concreto.
- Diagrama de máquina de estados: similar a los diagramas de actividades, describen el comportamiento de objetos que se comportan de diversas formas en su estado actual.
- Diagrama de caso de uso: representa una funcionalidad particular de un sistema. Se crea para ilustrar cómo se relacionan las funcionalidades con sus controladores (actores) internos/externos.

3.7.3 Casos de uso

Un caso de uso es una interacción típica entre un usuario y un sistema de cómputo. Algunas de las propiedades de los casos de uso son:

- El caso de uso capta alguna función visible para el usuario.
- El caso de uso puede ser pequeño o grande.
- El caso de uso logra un objetivo discreto para el usuario.

El caso de uso se obtiene hablando con los usuarios habituales y analizando con ellos las diferentes cosas que deseen realizar con el sistema.
En 1994 Jacobson, diseño un diagrama para la representación gráfica de los casos de uso.
Los elementos son los siguientes:

- **Actor**: un actor es un rol que desempeñan un usuario con respecto al sistema; además, un actor no necesariamente representa a una persona en particular, sino a la labor que realiza en el sistema.
- **Caso de uso**: es una operación o tarea que se realiza por una orden de algún agente externo, este puede ser dada por un actor o por otro caso de uso.
- **Relaciones**:
    - Asociación: indica la invocación desde un actor o caso de uso a un caso de uso.
    - Generalización: Cumple una doble función dependiendo de su uso: *Uses* (usa) ocurre cuando se tiene una porción del mismo comportamiento en más de un caso de uso y no se desea mantener copiada la descripción de ese comportamiento; *extends* (extiende) ocurre cuando un caso de uso es similar a otro caso de uso pero realiza un poco más.

3.8 Arquitectura MVP para Android

El patrón MVP (*Model View Presenter)* es un derivado del conocido MVC (*Model View Controller*), y uno de los patrones más populares para organizar la capa de presentación en las aplicaciones de Android. Este patrón permite separar la capa de presentación de la lógica para que todo sobre cómo funciona la interfaz de usuario sea independiente de cómo se ve en la pantalla. Idealmente, el patrón MVP lograría que la misma lógica pudiera tener vistas completamente diferentes e intercambiables.

En Android, existe un problema derivado del hecho de que las actividades de Android están estrechamente relacionadas con la UI y los mecanismos de acceso a datos. La necesidad de utilizar este patrón surge debido a lo complicado que puede llegar a ser el mantenimiento y escalamiento de un proyecto que va creciendo a lo largo del tiempo y líneas de código. El uso de esta arquitectura lleva a facilitar las implemnetación de mejoras, actualizaciones, modificaciones o arreglos de cualquier parte de la aplicación.

Si bien el patrón MVP establece una serie de lineamientos, se debe tener en cuenta que cada proyecto puede llegar a implementarlo con algunas variaciones, por lo que no es una solución absoluta. Esto se debe a que solo modela la capa de presentación, pero el resto de capas aún requerirán una buena arquitectura para lograr una aplicación flexible y escalable.

Un ejemplo de una arquitectura completa podría ser *Clean Architecture*, aunque hay muchas otras opciones [22].

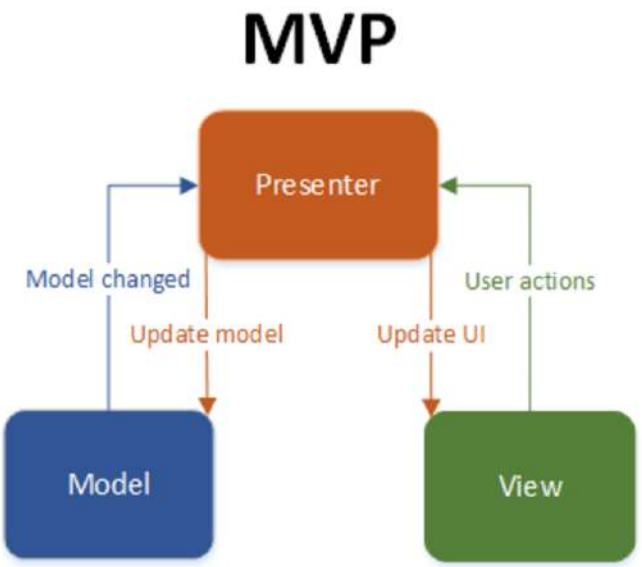

Figura 4: Diagrama MVP.
Fuente: Deveapps. http://www.develapps.com/es/noticias/

El diagrama de la Figura 4 muestra la interacción básica en donde se puede observar cuatro elementos claves, el modelo, la vista, el presentador y las flechas. Algo a notar muy importante es que la vista y el modelo nunca interactúan, toda acción entre ellos dos será a través del presentador:

- **Model:** en esta capa se incluyen las clases relacionadas al acceso de datos, reglas de negocio u obtención de información. Es muy común que al implementar MVP en un proyecto de Android se haga de la mano con la arquitectura *Clean*, que implica el uso de *Interactors*. Tanto un *Model* como un Interactor proporcionan información de datos, donde el Model proporciona funcionalidades generales y el *Interactor* que debe tener funcionalidades específicas.
- **Vista**: en esta capa se incluyen todas las clases relacionadas directamente con la interfaz gráfica, usualmente Activities y Fragments (Las vistas XML son parte de la vista pero de una forma más abstracta debido a que forman parte de los recursos del proyecto, aunque están muy acopladas con estas clases).
- **Presentador**: en esta capa se incluyen todas las clases Presenter que se encargará de comunicar a los Interactores (Modelos) y las vistas.

3.9 Control de versiones con Git

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación.

Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión dando lugar a los llamados sistemas de control de versiones o VCS (del inglés Version Control System). Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (por ejemplo, para algún cliente específico).

El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se permite recuperar versiones específicas más adelante.

Git es el sistema de control de versiones, que permite tener un control de cambios sobre el código fuente de una aplicación. Fue diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

Git es uno de los sistemas de control de versiones más populares entre los desarrolladores. Uno de los principales motivos es que GitHub brinda un excelente servicio de alojamiento de repositorios de software, con este sistema ofrece hoy en día un conjunto de características muy útiles y completas para el trabajo en equipo [23].

Capítulo IV Desarrollo de nuevas funcionalidades

La continua evolución del mundo de las aplicaciones móviles hace indispensable el mantenimiento de las mismas, aportando a los usuarios mayores funcionalidades y una mejor personalización. Esto se debe principalmente a los siguientes factores:

- Las continuas actualizaciones de los sistemas operativos de los dispositivos móviles, hacen necesaria una continua adaptación de las aplicaciones a los mismos.
- La salida al mercado de dispositivos nuevos hace que sin un adecuado mantenimiento la aplicación pueda dejar de ejecutarse en algunos de estos dispositivos.
- El usuario, por otro lado, no valora favorablemente las aplicaciones que no se adecuan a las nuevas versiones de los sistemas operativos.
- Las necesidades de los clientes cambian constantemente.

En las siguientes secciones se describen una serie de funcionalidades que se desarrollaron e integraron a la aplicación con el objetivo de aportar una mayor satisfacción a los clientes, optimizando y agilizando sus operaciones diarias en el uso de la misma. La integración de estas funcionalidades a la aplicación se ilustran en el diagrama de la Figura 5, donde los nuevos paquetes son aquellos representados con color.

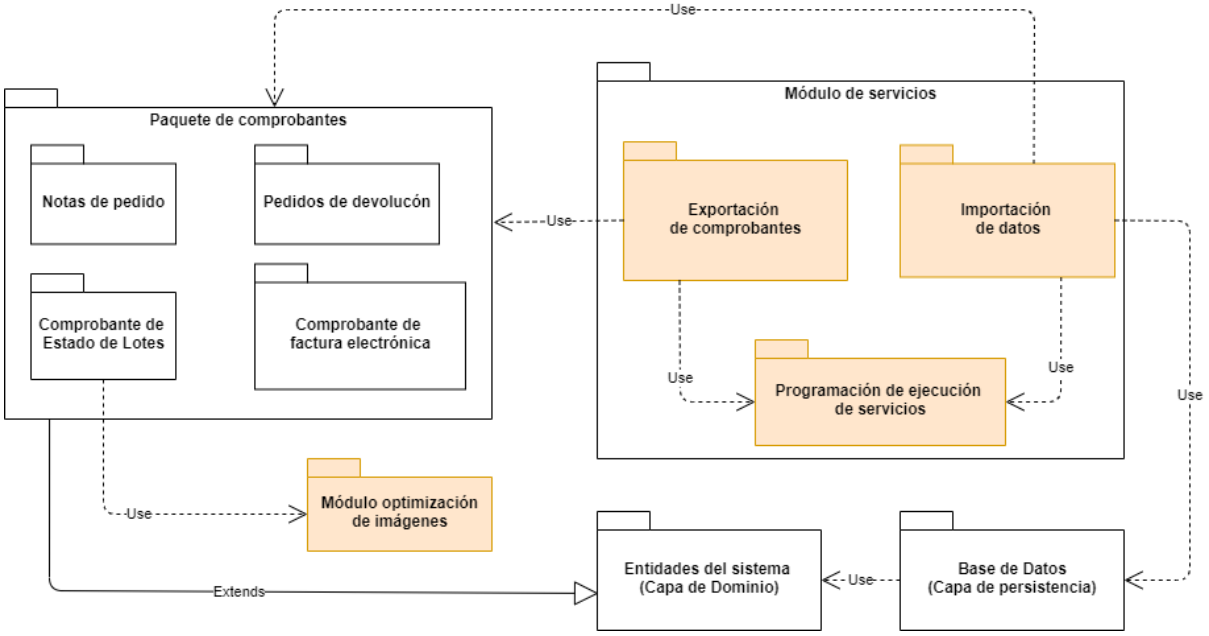

Figura 5: Diagrama de paquetes con las nuevas funcionalidades de la aplicación.

4.1 Optimización de las imágenes

La mayor carga de procesamiento al comunicarse con el servidor central es la carga y descarga de imágenes, las cuales están asociadas a los comprobantes. En consecuencia, cuanto más pequeñas sean las imágenes para descargar, mejor será la experiencia de red que la aplicación ofrece a los usuarios. En esta sección, se describe el proceso que se llevó a cabo para reducir el tamaño de los archivos de imagen y optimizarlos para la red.

Las aplicaciones para Android suelen usar imágenes con uno o más formatos de archivo, tales como PNG, JPG y WebP. Estos formatos de imagen se adecúan de acuerdo al tipo de imagen, por lo que hay que tener en cuenta la complejidad de las imágenes y los resultados que se esperan obtener. La Figura 6 proporciona una visualización simple que permite definir el esquema de compresión a utilizar. Por lo tanto, debido a que las imágenes que deben procesar no cuentan con una gran cantidad de pequeños detalles y la aplicación no soporta WebP, el formato de imágen seleccionado fue JPG.

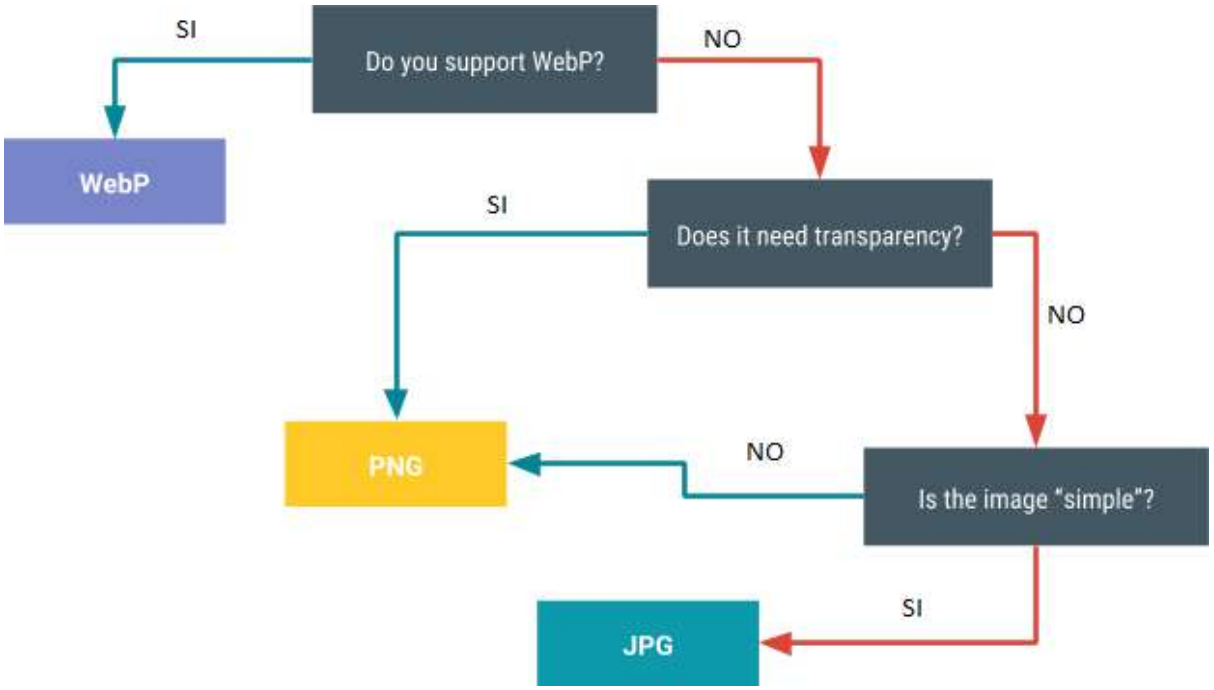

Figura 6: Selección del esquema de compresión a utilizar.
https://developer.android.com/topic/performance/network-xfer?hl=es-419 [28/08/2021]

Si bien existen varias técnicas que se pueden utilizar para lograr el equilibrio correcto entre la compresión y la calidad de la imagen, se utilizó la técnica de valores escalares. Esta técnica permite usar un valor escalar para equilibrar la calidad con el tamaño del archivo. El objetivo es descubrir cuál es el valor de calidad correcto para la imágen. Un nivel de calidad demasiado bajo produce un archivo pequeño, en el que se pierde la calidad de la imagen. Un nivel de calidad demasiado alto aumenta el tamaño del archivo sin proporcionar un beneficio notable al usuario [24].

El Código 1 corresponde a un fragmento del archivo *ImageCompressor.java*, que se encarga de llevar a cabo la reducción del tamaño de la imagen con el objetivo de optimizar el procesamiento de la misma.

```java
public static File getCompressed(Context context, String path,
                                 String width, String height,
                                 String quality) throws IOException
{
    File rootDir = getOrCreateRootDirectory(context);

    // Crear contenedor para el archivo de la imágen comprimida
    File compressed = new File(root, SDF.format(new Date()) + ".jpg");

    boolean reduce = true;
    while (reduce) {
        reduce = false;

        // Decodificar y redimensionar el bitmap original de @param path
        Bitmap bitmap = decodeImageFromFiles(path, width, height);
        bitmap = rotateImageIfRequired(context, bitmap, Uri.fromFile(new File(path)));

        // Convertir el bitmap decodificado a stream
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        bitmap.compress(Bitmap.CompressFormat.JPEG, quality, byteArrayOutputStream);

        FileOutputStream fileOutputStream = new FileOutputStream(compressed);
        fileOutputStream.write(byteArrayOutputStream.toByteArray());
        fileOutputStream.flush();
        fileOutputStream.close();

        if (compressed.length() > 100000 /* Tamaño requerido por el servidor */) {
            reduce = true;
            width = width - ((int) width * 0.1F);
            height = height - ((int) height * 0.1F);
            quality = quality - ((int) quality * 0.1F);
        }
    }
    return compressed;
}
```

Código 1: Método que se encarga de comprimir una imágen.

En el Código 2 se puede visualizar la implementación específica del método *decodeImageFromFiles*, que utiliza la técnica de valores escalares mencionada anteriormente para la compresión de la imágen.

```
public static Bitmap decodeImageFromFiles(String path, int width, int height)
{
    BitmapFactory.Options scaleOptions = new BitmapFactory.Options();
    scaleOptions.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(path, scaleOptions);
    int scale = 1;

    while (scaleOptions.outWidth / scale / 2 >= width
            && scaleOptions.outHeight / scale / 2 >= height) {
        scale *= 2;
    }

    BitmapFactory.Options outOptions = new BitmapFactory.Options();
    outOptions.inSampleSize = scale;
    return BitmapFactory.decodeFile(path, outOptions);
}
```

Código 2: Método que utiliza la técnica de valores escalares para comprimir una imágen.

Esta implementación permitió mejorar la velocidad de carga y descarga de las imágenes de la aplicación, logrando reducir hasta un 80% el tiempo de procesamiento. Esto permió que este proceso sea menos costoso para los usuarios, que tal vez estén usando planes de datos medidos o limitados, o la conexión a internet es lenta. Por otro lado, también hace que la imagen ocupe menos espacio en el dispositivo y en la memoria principal, optimizando el rendimiento del mismo.

4.2 Exportación automática de comprobantes

El objetivo principal de la aplicación es alimentar al sistema central de información. De esta forma, una de las tareas que se llevan a cabo en la aplicación es generar y almacenar comprobantes que luego son exportados al sistema. Anteriormente, un usuario de la aplicación tomaba pedidos durante su jornada laboral y una vez que finalizaba la toma de pedidos exportaba los comprobantes generados al sistema central.

Este proceso de exportación era manual, por lo que el usuario determinaba en qué momento se debía ejecutar y qué comprobantes se iban a enviar. Esto presentaba ciertas desventajas tales como:

- Acumulación de comprobantes, llegando a situaciones en la que la exportación cuenta con una gran cantidad de datos, haciendo que se ralentice y aumentando la vulnerabilidad a fallos. Además, la aplicación queda inutilizable durante el proceso de exportación, obligando al usuario a esperar hasta que el proceso finalice.
- Pérdida de tiempo; es una tarea que el usuario tiene que repetir por cada tipo de comprobante que exista en la aplicación.

- Seguimiento continuo para asegurarse que cada comprobante sea exportado al sistema central. El tiempo dedicado a monitorear esta situación se podría utilizar en actividades más productivas para la empresa.
- Aumento de la posibilidad de errores humanos.

Por lo tanto, con el fin de simplificar al máximo estas tareas rutinarias que se desempeñan con mucha frecuencia, se desarrolló un proceso que permite la automatización de dichas tareas. De esta forma, se reduce la posibilidad de cometer errores y permite aumentar la productividad de la empresa. Además, la automatización de procesos permite programar las tareas, permitiendo la planificación sin supervisión.

Esta funcionalidad pudo implementarse al descomponerla en dos pasos: implementar el servicio con las operaciones que queremos realizar y programar la ejecución periódica del servicio.

4.2.1 Implementación del servicio de exportación

Para llevar a cabo este desarrollo se utilizaron los servicios de Android, que permiten ejecutar operaciones de larga ejecución en segundo plano, sin la necesidad de proporcionar una interfaz de usuario. Esto quiere decir que se puede ejecutar incluso cuando el usuario no está interactuando con la aplicación. Por lo tanto, mientras se hace un uso normal de la aplicación, se puede llevar a cabo la exportación de los comprobantes [25]. El diagrama de la Figura 7 permite representar gráficamente el proceso de automatización de exportación de comprobantes al sistema central utilizando los servicios de Android.

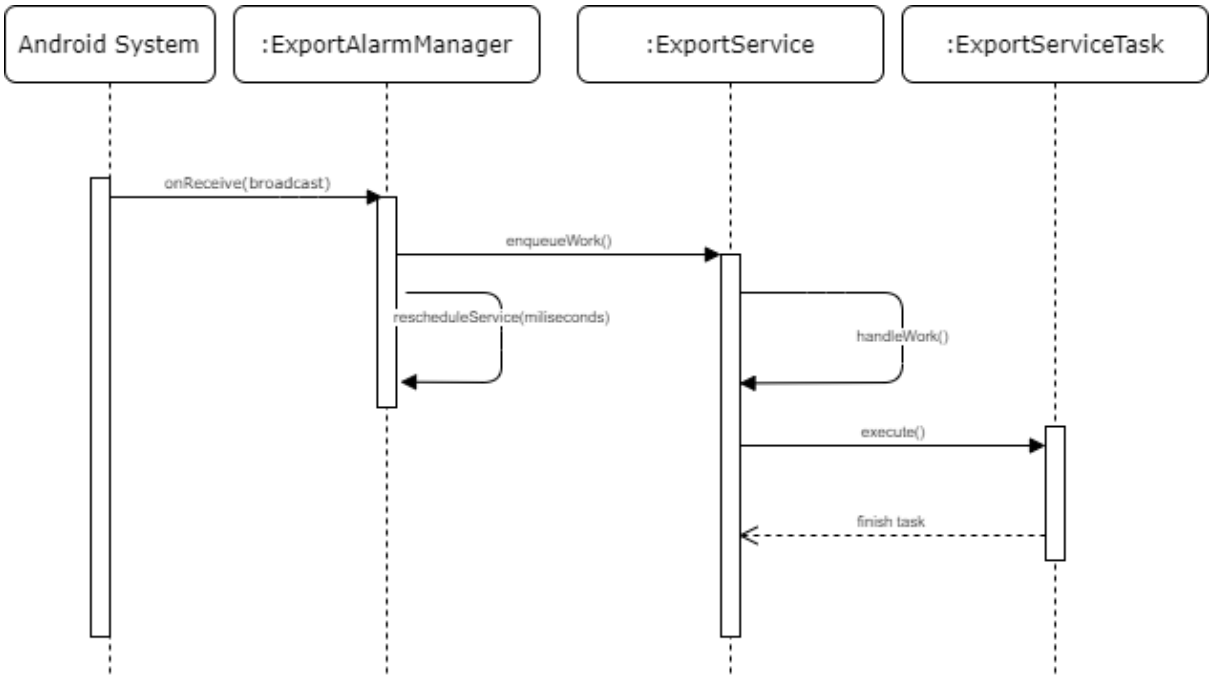

Figura 7: Diagrama de flujo del proceso de exportación automática utilizando servicios.

Este nuevo proceso se encarga de exportar en cada ejecución los comprobantes almacenados, independientemente del tipo que sean y de forma transparente al usuario. De

esta forma, el usuario no debe preocuparse por trasladar esta información diariamente de forma manual al sistema, automatizando todo el proceso.

En la práctica se suelen implementar dos tipos de servicios: el propio *Service* o bien *IntentService*. El primero se ejecuta en segundo plano pero en el hilo principal de la aplicación, mientras que el segundo se ejecuta en un hilo independiente y se detiene automáticamente tras finalizar su ejecución [26]. En este caso se utilizó el servicio *JobIntentService* (subclase de *IntentService*).

Esta clase provista por Android es considerada la forma moderna de usar servicios en *background* para llevar a cabo distintas tareas, ya que funciona de forma correcta en todos los dispositivos y evita los fallos que otros servicios producen en dispositivos Oreo (API 26) o posterior. El Código 3 muestra la definición del servicio.

```
public class ExportService extends JobIntentService {

    private static final int JOB_ID = 1;

    public void enqueueWork(Context context, Intent intent) {
        enqueueWork(context, ExportJobIIService.class, JOB_ID, intent);
    }

    @Override
    protected void onHandleWork(@NonNull Intent intent) {
        new ExportServiceTask().execute();
    }
}
```

Código 3: Definición del servicio ExportService.

- El atributo **JOB_ID** define un identificador único para el servicio. Este identificador es utilizado por el sistema android para la planificación de ejecución de tareas, por lo que se debe mantener el mismo valor para cada trabajo iniciado por la misma clase.
- El método ***enqueueWork*** se utiliza para iniciar directamente el servicio (cuando se ejecuta en plataformas pre-O) o pondrá en cola el proceso para ser ejecutado en segundo plano (cuando se ejecuta en O y posterior)
- El método ***onHandleWork*** se llama en un subproceso en segundo plano, por lo que se utiliza para realizar largas operaciones de bloqueo, es decir, aquí se lleva a cabo la lógica de la operación. En este caso, se ejecuta el método correspondiente de la clase *ExportServiceTask*, encargada de exportar todos los comprobantes al sistema central.

Los servicios se deben declarar en el archivo manifiesto de la aplicación, de igual forma que se hace con las actividades y demás componentes. Esto permite al sistema Android interactuar con los mismos. Para declarar el servicio, se agregó un elemento <service> como campo secundario del elemento de <*application*>. El Código 4 muestra el resultado en el archivo manifiesto.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="igd.iduo">

    ...

    <uses-permission android:name="android.permission.WAKE_LOCK" />

    <application
        android:name=".App">

        ...

        <service
            android:name=".exportAsService.ExportService"
            android:exported="true"
            android:permission="android.permission.BIND_JOB_SERVICE" />

    </application>
</manifest>
```

Código 4: Servicio declarado en el manifest.

4.2.2 Programación de la ejecución del servicio

Una vez configurado el servicio, es necesario ejecutarlo cada cierto período de tiempo, incluso si la aplicación está en *background* (o no está en ejecución). Entre las diversas clases que existen en Android para realizar esto, se utilizaron las clases *AlarmManager* y *BroadcastReceiver* [27]. Básicamente, el proceso consiste en definir un *AlarmManger* para que lance un evento cada X minutos. Luego, se configura el *BroadcastReceiver* para que reciba este evento e inicie la ejecución del *JobIntentService* creado anteriormente para que realice el trabajo correspondiente.

Un *BroadcastReceiver* es el componente que está destinado a recibir y responder ante eventos globales generados por el sistema. Todos los *Receivers* registrados para un evento serán notificados por Android una vez que éstos ocurran, tal como muestra la Figura 8.

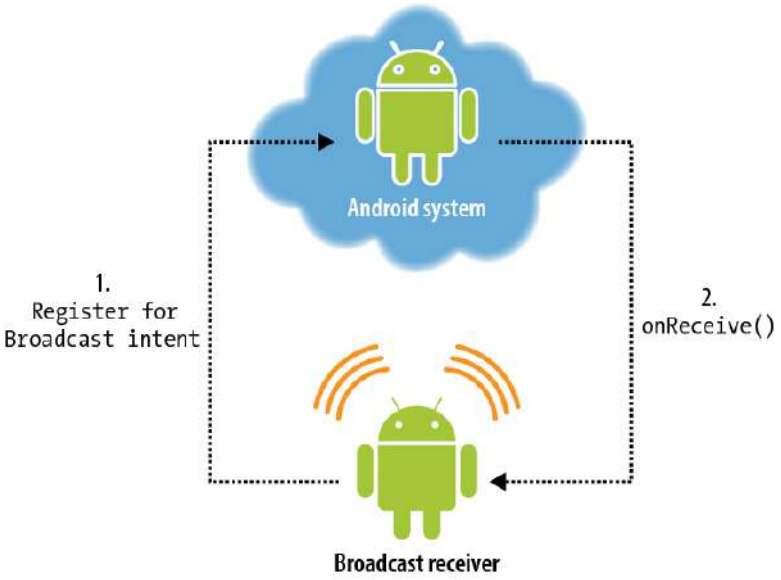

Figura 8: Descripción de un BroadcastReceiver.
Fuente: Lars Vogel. https://www.vogella.com/AndroidBroadcastReceiver/article.html [28/08/2021]

Para implementarlo se creó una clase que herede de *BroadcastReceiver* y se sobrecribió el método *onReceive*, el cual se ejecutará cada vez que se produzca el evento al que esté suscrito el *BroadcastReceiver*, tal como lo muestra el Código 5.

```
public class ExportAlarmReceiver extends BroadcastReceiver {

    public static final String CUSTOM_INTENT = "igd.iduo.intent.action.EXPORT_ALARM";

    @Override
    public void onReceive(Context context, Intent intent) {
        exportService.enqueueWork(context, intent);
        rescheduleService(1000 * 60 * 60 * 4);
    }

   public void rescheduleService(Integer miliseconds) {
        AlarmManager alarm = (AlarmManager) context.getSystemService(Context.
                ALARM_SERVICE);
        long when = System.currentTimeMillis() + miliseconds;
        int SDK_INT = Build.VERSION.SDK_INT;
        if (SDK_INT < Build.VERSION_CODES.KITKAT) {
            alarm.set(AlarmManager.
                    RTC_WAKEUP, when, getPendingIntent(context));
        }else if (SDK_INT < Build.VERSION_CODES.M) {
            alarm.setExact(AlarmManager.
                    RTC_WAKEUP, when, getPendingIntent(context));
        } else{
            alarm.setExactAndAllowWhileIdle(AlarmManager.
                    RTC_WAKEUP, when, getPendingIntent(context));
        }
        Log.d(TAG, "setAlarm: Job service reprogramado");
    }
}
```

Código 5: Definición del BroadcastReceiver.

Es importante tener en cuenta que dentro del método *onReceive* no se deben realizar operaciones que lleven mucho tiempo, ni operaciones de persistencia. Por este motivo se utiliza *JobIntentService*, detallado en el apartado anterior.

Por otro lado, la clase *AlarmManager* provee acceso a los servicios de alarma del sistema, permitiendo programar a la aplicación para que lance el evento correspondiente luego de cierto período de tiempo (en este caso, 4 horas expresada en milisegundos). El Código 6 muestra la declaración del *Receiver* en el archivo AndroidManifest, dentro del elemento *<application>*. El servicio se puede iniciar o detener desde el panel de administración, donde el usuario tiene la posibilidad de activar o desactivar dicha funcionalidad.

```
<receiver
    android:name=".exportAsService.ExportAlarmReceiver"
    android:exported="false">
    <intent-filter>
        <action android:name="igd.iduo.intent.action.EXPORT_ALARM" />
    </intent-filter>
</receiver>
```

Código 6: Definición del receiver en el archivo manifiesto.

4.3 Importación automática de datos

La aplicación se encarga de almacenar información localmente con el fin de permitir a los usuarios continuar con la operatoria diaria sin la necesidad de tener acceso a Internet. Esto implica que debe existir una comunicación constante con el servidor central que permita mantener una consistencia entre los datos almacenados localmente y aquellos que se encuentran de forma remota.

Anteriormente, para llevar a cabo esta sincronización el usuario debía iniciar manualmente la importación de cada una de las entidades con las que contaba la aplicación, tales como usuarios, productos, empresas, sucursales, etc. Éstos eran los encargados de realizar esta tarea diariamente antes de comenzar con su jornada laboral, asegurándose de contar con la última información disponible.

Con el objetivo de facilitar esta tarea, y evitarle a los usuarios la responsabilidad de llevarla a cabo diariamente, se desarrolló un servicio automático programado para ejecutarse cada cierta cantidad de tiempo (actualmente definida cada 4 horas). En el caso de que al momento de inciar el servicio no se cuente con conexión a internet, éste se cancela y se programa nuevamente para ejecutarse en las próximas horas. Para ello se utilizaron los conceptos mencionados en el apartado anterior, como son la definición del servicio utilizando *JobIntentService* y la programación del mismo con la clase *AlarmManager*, ambas provistas por el *SDK* de Android.

Capítulo V Incorporación del comprobante electrónico

El sistema web de Iduo comprende circuitos de facturación fiscal, electrónica, control de stock y cuentas corrientes de clientes y proveedores, administración financiera, reportes y estadísticas, administración de sucursales y tesorería:

- Permite la emisión de Ticket, Ticket Factura A, B y C, Factura A y B, Notas de Crédito A y B, Notas de Débito A y B, y otros Documentos Fiscales Homologados en controladores fiscales. Además de otros documentos definidos por el usuario como notas de pedido, presupuestos o remitos. También permite agregar el módulo de facturación electrónica, con el objetivo de efectuar los documentos mencionados anteriormente en línea contra la autoridad fiscal (AFIP –Argentina).
- Maneja todos los tipos de Códigos de Barra.
- Permite el ingreso de productos mediante código interno, código alfanumérico, código de barras y parte de la descripción.
- Maneja recargos/descuentos por medio de pago, porcentajes de descuento por cliente a nivel de producto y a nivel del total de la venta.
- Maneja múltiples porcentajes de comisión para vendedores.
- Maneja un número ilimitado de listas de precios en todas sus versiones.
- Permite generar etiquetas de códigos de barra en todas sus versiones.
- Permite exportar e importar la información desde planillas MS Excel en todas sus versiones.
- Permite identificar productos pesables, productos compuestos (tales como combos o recetas de elaboración).
- Soporta todos los controladores fiscales (Epson, Hasar, Samsung, NCR).

El comprobante de factura electrónico generado mediante el sistema es digital, funcional y legalmente equivalente a la factura en formato papel, que la reemplaza en la mayoría de las operaciones correspondientes. Permite la gestión, almacenamiento e intercambio de comprobantes por medios electrónicos o digitales sin necesidad de su impresión, pudiéndose gestionar directamente desde el sistema web. De esta forma, el sistema web cumple el rol de intermediario entre el usuario y la AFIP, facilitando la generación de comprobantes.

Teniendo en cuenta esta situación y para ofrecer un nuevo servicio a los usuarios de Iduo, se decidió desarrollar e integrar, parcialmente, el módulo de facturación a la aplicación móvil con el objetivo de que se pueden generar y gestionar facturas electrónicas desde el mismo. Esta nueva herramienta permite:

- Ampliar las herramientas de autorización de comprobantes.
- Emitir toda la facturación por medios electrónicos. Registrar en las bases de AFIP todos los comprobantes facturados en tiempo real.
- Anular costos de impresión, distribución, almacenamiento y adquisición de software para el emisor.
- Integrar el modelo con medios de pago electrónico.

La Figura 9 muestra una serie de capturas de pantalla de la aplicación Iduo, las cuales ilustran el diseño de las pantallas principales por las cuales navega el usuario en la gestión de este nuevo comprobante.

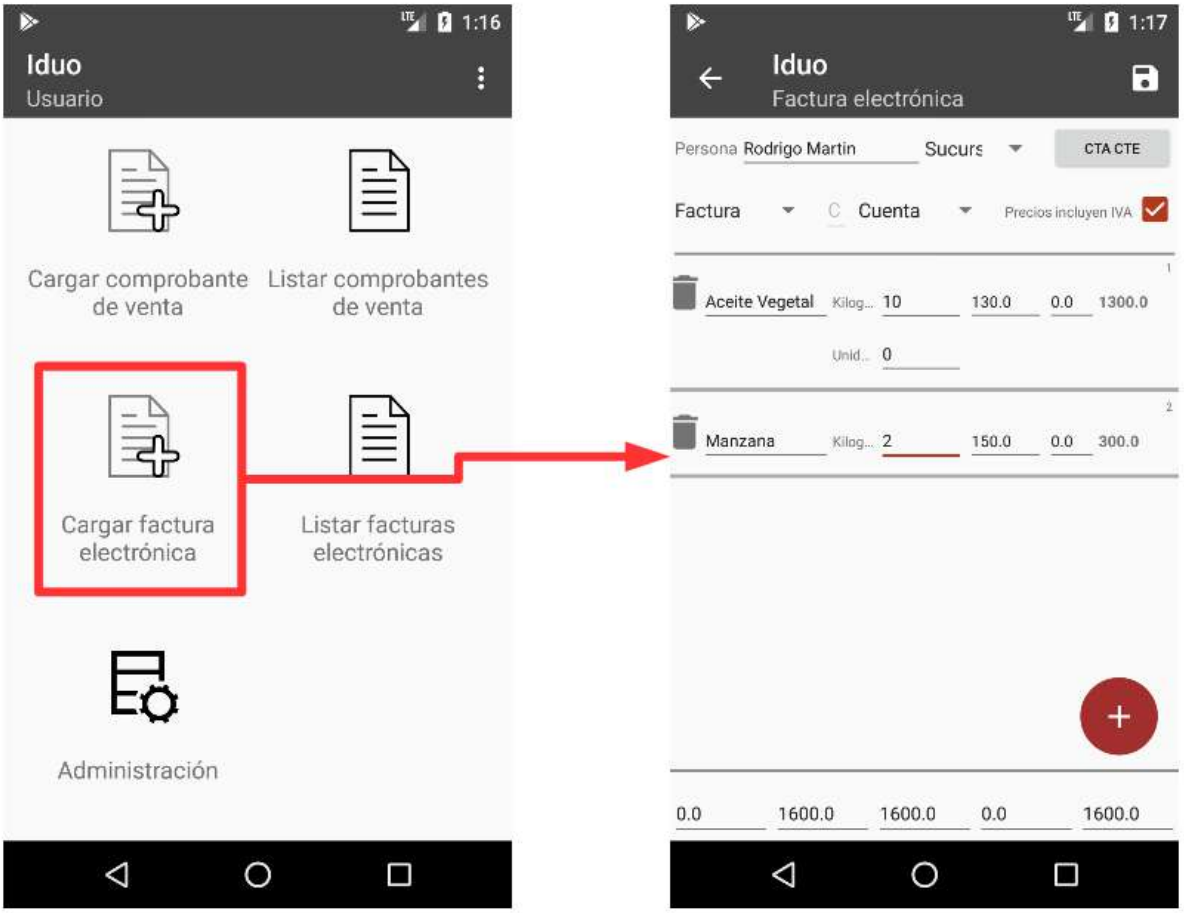

Figura 9: Capturas de la aplicación asociadas a la carga del comprobante.

5.1 Migración del esquema de base de datos de la aplicación

A medida que se agregan y cambian funciones en las aplicaciones, las entidades se deben modificar para reflejar esos cambios. Sin embargo, es importante conservar los datos del usuario que ya existen en la base de datos del dispositivo cuando una actualización de la aplicación cambia el esquema de la base de datos. Para satisfacer esa necesidad es necesario definir un sistema de migraciones incrementales, el cual permite modificar la base de datos para añadir las tablas correspondientes.

Cuando un usuario actualiza a la versión más reciente de la aplicación, si la migración de datos no se realiza correctamente, o existen errores en la ejecución, puede perder todos sus datos y llevar a un bloqueo permanente de la aplicación (provocando que esta quede inutilizable). Para evitar estos inconvenientes se utilizó la clase *SQLiteOpenHelper*, la cual provee soporte para migrar entre diferentes versiones de esquemas de base de datos. Así, permite identificar cuando hay un cambio y llevar a cabo las modificaciones correspondientes, en este caso, la incorporación de nuevas tablas. Para esto se creó la clase *DatabaseUpgradeHelper*, que hereda de *SQLiteOpenHelper* y se encarga, en tiempo de ejecución, de identificar cambios en el esquema de la base de datos (mediante el número de versión) y de ejecutar el método *onUpgrade* para migrar la base de datos a una versión

posterior, utilizando el orden correcto de pasos. La Figura 10 muestra un diagrama de las clases agregadas y sus relaciones.

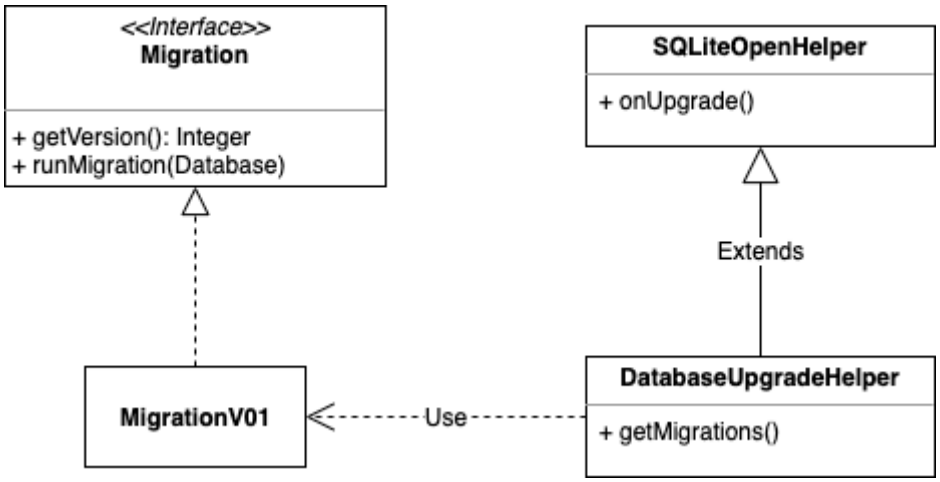

Figura 10: Diagrama de clases con las entidades y relaciones que llevan a cabo la migración del esquema de la base de datos.

El Código 6 muestra la definición de la clase *DatabaseUpgradeHelper*.

```
public class DatabaseUpgradeHelper extends SQLiteOpenHelper {

    public DatabaseUpgradeHelper(Context context, String name) {
        super(context, name);
    }

    @Override
    public void onUpgrade(Database db, int oldVersion, int newVersion) {

        List<Migration> migrations = getMigrations();
        for (Migration migration : migrations) {
            if (oldVersion < migration.getVersion()) {
                migration.runMigration(db);
            }
        }
    }

    private List<Migration> getMigrations() {
        List<Migration> migrations = new ArrayList<>();
        migrations.add(new MigrationV01());
        Comparator<Migration> migrationComparator =
                (m1, m2) -> m1.getVersion().compareTo(m2.getVersion());
        Collections.sort(migrations, migrationComparator);
        return migrations;
    }
}
```

Código 6: Definición de la clase DatabaseUpgradeHelper, la cual lleva a cabo la migración de la base de datos.

A continuación, en el Código 7 se puede observar la interfaz *Migration*, la cual define los métodos necesarios que se deben definir en una migración determinada. Así, *getVersion* permite identificar la versión asociada al esquema de la base de datos y realizar comparaciones entre diferentes versiones. Por otro lado, *runMigration* permite definir la lógica necesaria para llevar a cabo la migración del esquema de base de datos, ya sea la incorporación de nuevas tablas, modificación de atributos o la migración de información. La clase concreta *MigrationV01* se encarga de la creación de las nuevas tablas en la base de datos, las cuales se detallan en el siguiente párrafo.

```
private static class MigrationV01 implements Migration {

    @Override
    public Integer getVersion() {
        return 1;
    }

    @Override
    public void runMigration(Database db) {
        FacturaElectronicaDAO.createTable(db)
        DetalleFacturaElectronicaDAO.createTable(db)
        DetalleUnidadFacturaElectronicaDAO.createTable(db)

    }
}


private interface Migration {
    Integer getVersion();
    void runMigration(Database db);
}
```

Código 7: Código de la clase Migrations.

Para gestionar el nuevo comprobante de factura electrónica se agregaron las tablas correspondientes en la base de datos, como así también los atributos necesarios. La Figura 11 muestra un diagrama de entidad-relación (o también conocido como DER) con la representación gráfica de las entidades que fueron creadas (*FacturaElectronica*, *DetalleFacturaElectronica* y *DetalleUnidadFacturaElectronica*) junto con sus relaciones.

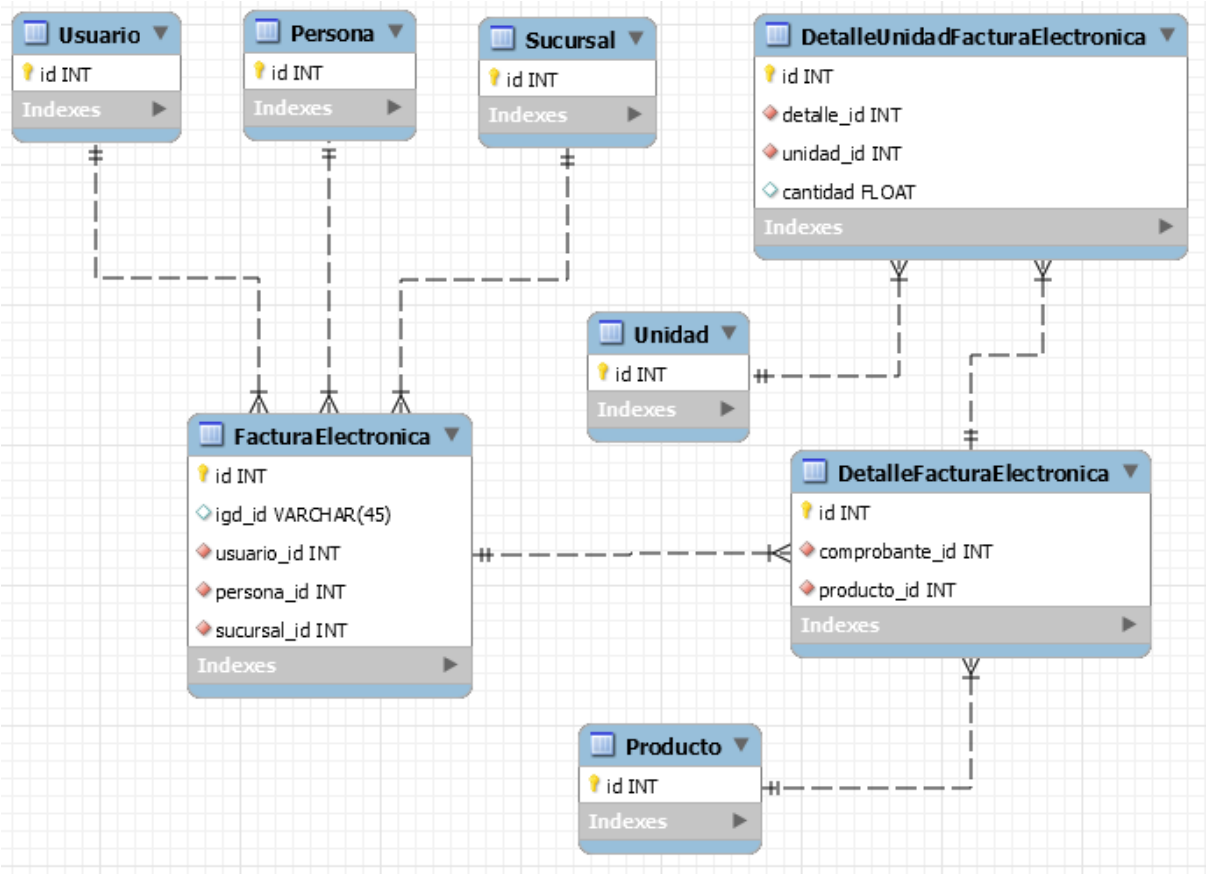

Figura 11: DER de las entidades y relaciones agregadas para gestionar comprobante de factura electrónica.

El Código 8 muestra la definición de la clase *DetalleUnidadFacturaElectronica*, utilizando anotaciones provistas por *greenDao* (*https://greenrobot.org/greendao/*), una solución ORM ligera y rápida que mapea objetos de bases de datos SQLite. Esto permite generar clases y métodos necesarios para poder acceder a la base de datos (conocidos como objetos DAO), los cuales se van a incluir en el proyecto, con la creación del esquema de la base de datos correspondiente [28].

```java
@Entity
public class DetalleUnidadFacturaElectronica {

    @Id(autoincrement = true)
    private Long id;
    @Property(nameInDb = "detalle_id")
    private Long detailId;
    @Property(nameInDb = "unidad_id")
    private Long unitId;
    @Property(nameInDb = "cantidad")
    private Double amount;

    @ToOne(joinProperty = "detailId")
    private DetalleFacturaElectronica detail;

    @ToOne(joinProperty = "unitId")
    private Unidad unit;

    // ....
}
```

Código 8: Declaración de la entidad DetalleUnidadFacturaElectrónica.

- **Entity**: la anotación @Entity convierte la clase Java *DetalleUnidadFacturaElectronica* en una entidad respaldada por una base de datos. Esto también indicará a *greenDao* que genere el código necesario (por ejemplo, DAO).
- **Id**: la anotación @Id selecciona una propiedad long / Long como ID de entidad. En términos de base de datos, es la clave principal. El parámetro autoincrement es una bandera para hacer que el valor de ID sea incremental (sin reutilizar valores antiguos).
- **Property**: @Property permite definir un nombre de columna no predeterminado, al que se asigna la propiedad. Si está ausente, *greenDao* usará el nombre del campo en una forma SQL-ish (mayúsculas, guiones bajos en *camel case*, por ejemplo, *customName* se convertirá en CUSTOM_NAME). *Nota*: actualmente solo puede usar constantes en línea para especificar un nombre de columna.
- **ToOne**: la anotación @ToOne define una relación con otra entidad (un objeto de entidad). Se debe aplicar a la propiedad que contiene el otro objeto de entidad. Internamente, *greenDao* necesita una propiedad adicional que apunte al ID de la entidad de destino, que se especifica mediante el parámetro *joinProperty*. Si este parámetro está ausente, se crea automáticamente una columna adicional para contener la clave.

5.2 Generar PDF de Factura Electrónica

Para aquellos comprobantes almacenados localmente, se agregó la opción que permite generar un documento PDF utilizando diseños de facturas A, B o C. De esta forma, el usuario puede compartir o enviar este documento a sus clientes. Para cumplir con las normativas dispuestas por AFIP y que sea considerado un documento válido de presentación a terceros, se replicó el modelo utilizado por el sistema web, que ya cumple con las normativas vigentes.

Para la creación del documento se utilizó la librería **iText PDF**, por lo que se agregó la dependencia correspondiente en el archivo gradle del proyecto. Esta librería cuenta con un conjunto de clases y métodos que permiten construir el documento con un diseño personalizado. El Código 9 muestra lo primero que se debe hacer para crear un documento nuevo PDF y posteriormente, poder escribir sobre él.

```
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.pdf.PdfWriter;

...

// Se crea el documento
Document document = new Document();

// Se crea el OutputStream para el archivo donde queremos dejar el pdf.
FileOutputStream pdfFile = new FileOutputStream("comprobante.pdf");

// Se asocia el documento al OutputStream y se indica que el espaciado entre
// lineas sera de 10. Esta llamada debe hacerse antes de abrir el documento
PdfWriter.getInstance(document, pdfFile).setInitialLeading(10);

// Se abre el documento.
documento.open();
```

Código 9: Creación del objeto Document para la generación del PDF.

El objeto de tipo *Document* cuenta con métodos *add* para añadir cualquier tipo de clase de *iText* que implemente la interfaz *Element*. También tiene métodos para añadir las propiedades del documento, como *addTitle* o *addAuthor*. Dentro de los *Element* que se pueden añadir, existen muchos y muy variados: frases, párrafos, capítulos, imágenes, tablas, separadores, entre otros. Estos permiten ir añadiendo elementos y lograr el diseño esperado.

Una vez que se creó el archivo y se guardó en el dispositivo, el usuario puede compartirlo utilizando cualquier aplicación instalada que permita realizar esta operación. Android permite enviar datos a otras aplicaciones, utilizando *Intents* y su contenido adicional asociado para compartir información de forma rápida y sencilla. De esta forma, los usuarios pueden compartir contenido en Android utilizando un agente de resolución de *Intent* (*Android Sharesheet*), que se diseñó para enviar contenido fuera de la aplicación o directamente a otro usuario.

Para construir el *Intent*, se debe especificar la acción que se quiere llevar a cabo. Para ello se utilizó la acción **ACTION_SEND** que permite enviar datos de una aplicación a otra, indicando el URI correspondiente del archivo y su tipo. El sistema automáticamente identifica las aplicaciones compatibles que pueden recibir el dato y las muestra al usuario. Si solo una aplicación puede manejar un determinado *Intent*, esa aplicación se inicia de inmediato. Por otro lado, si no existe ninguna que pueda resolver dicha acción, el sistema indicará que no hay una aplicación que pueda manejar la solicitud. El Código 10 muestra como se declara el *Intent* encargado de compartir el documento pasado como parámetro.

```
public void sharePdf(URI pdf){
    Intent target = new Intent(Intent.ACTION_SEND);
    target.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
    target.putExtra(Intent.EXTRA_STREAM, pdf);
    target.setType("application/pdf");
    Intent intent = Intent.createChooser(target, "Compartir archivo PDF");
    startActivity(intent);
}
```

Código 10: fragmento de código que inicia el Intent para compartir archivo PDF.

Con el objetivo de añadir un valor agregado y teniendo en cuenta el desarrollo que se llevó a cabo al integrar Mercado Pago a la aplicación (detallado en el siguiente capítulo), se incorporó el pago de facturas electrónicas mediante Mercado Pago. De esta forma, al momento que el usuario crea un nuevo comprobante, se genera un *link* que permite efectuar el pago del mismo utilizando Mercado Pago. Este link se agrega al PDF generado, de forma que el usuario tenga la posibilidad de acceder y realizar el pago correspondiente en cualquier momento.

Capitulo VI Diseño e implementación de aplicación e-commerce

Hoy en día un gran número de empresas, sobre todo las del sector de ventas se ven impulsadas a obtener o a desarrollar su propia aplicación móvil para sacar ventaja ante la competencia y, así, llegar a muchos otros mercados como también agilizar sus procesos. El rápido crecimiento de los negocios electrónicos en el mercado ha provocado que muchas empresas apuesten por una aplicación móvil e-commerce como una extensión del negocio, la cual permita seguir creciendo y vendiendo. La flexibilidad para comprar en cualquier momento, comparar productos y precios o ahorrar tiempo en la tienda son algunas de las razones por la cual los usuarios demandan este tipo de aplicaciones.

Algunos de los puntos más destacados de la aplicaciones móviles *e-commerce* son:

- **Experiencia de compra premium**: una de las grandes ventajas de las aplicaciones móviles es que permiten acceder a elementos propios de cada sistema operativo. Esta característica hace que la forma en la que los usuarios interactúan con los productos de la empresa sea ampliamente superior a la página web, permitiendo mejorar la experiencia de los usuarios mientras aumentan los indicadores de ventas.
- **Excelente herramienta de fidelización de clientes**: una ventaja muy importante de las aplicaciones móviles es que son ideales para aumentar el porcentaje de retención de los clientes. Aumentar la base de clientes fidelizados es un objetivo de casi todos los negocios online, principalmente porque es sencillamente más económico que un cliente vuelva a comprar a tener que estar buscando nuevos clientes todo el tiempo. Las aplicaciones móviles están alineadas detrás de este objetivo y para eso cuentan con una poderosa herramienta: las notificaciones *push*, o sea, alertas de texto programadas que se envían directamente a los teléfonos de los usuarios. Se podrían enviar notificaciones para comunicar promociones, eventos especiales, descuentos en alguna línea de productos u otro mensaje que sea relevante.
- **Velocidad**: las aplicaciones móviles permiten navegar por la tienda de una forma atractiva y rápida. Esta cualidad es importante porque los usuarios cada vez son más demandantes en cuanto al tiempo de carga de información.
- **Distribución**: las aplicaciones móviles deben ser descargadas desde un mercado de aplicaciones. Tanto la Play Store (Android) como la App Store (Apple) reciben miles de visitas por día de usuarios interesados en descargar nuevas aplicaciones a sus teléfonos. Si se logra posicionar la aplicación dentro de los primeros lugares de estos mercados, se puede generar un gran impacto y atraer nuevos clientes.
- **Posicionamiento de marca:** cuando se instala una aplicación móvil, aparece el ícono en la pantalla central del teléfono de los clientes. Esta característica es muy importante porque prácticamente asegura que el cliente la tiene dentro de su menú de opciones a la hora de elegir una tienda para comprar.

Desde esta perspectiva, se decidió llevar a cabo el desarrollo la aplicación móvil Capitano Craft. En las siguientes secciones, se describe con mayor detalle el análisis del sistema y las implementaciones que se desarrollaron e integraron en la aplicación.

6.1 Análisis y diseño del sistema

En esta sección se muestan los diseños que sirven de referencia para la implementación de la aplicación de e-commerce. Para esto se emplea el modelado UML, ya que se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware, y organizaciones del mundo real. UML ofrece varios diagramas para modelar sistemas, los desarrolladores lo eligen puesto que es un lenguaje que ayuda a discutir problemas y soluciones implicadas en la construcción del sistema y principalmente porque es el lenguaje de modelado de software más conocido y utilizado [29].
Para este proyecto, primero fue necesario realizar el relevamiento y análisis de requerimientos, para luego pasar al modelado del sistema. Para lo cual se optó por la realización de los diagramas más relevantes como son:

- Diagramas de casos de uso
- Diagrama de paquetes
- Diagramas de clases

También se contempló en el diseño un esquema de Navegación, el cual ayuda en la comprensión del orden de presentación de las pantallas de nuestra aplicación con los contenidos y los vínculos que existen en cada una de ellas.

Requerimientos

Se pretende realizar una aplicación móvil *e-commerce* para el comercio de Capitano Craft. En base a ello se definieron las siguientes funcionalidades:

- Sistema de registro y login de usuario: permitir a las personas iniciar sesión en tu aplicación mediante un proceso de creación y validación de cuenta o el inicio de sesión con Facebook (simplificando el proceso de carga y validación de información).
- Catálogo de producto: conjunto de todos los productos, dividiendo los artículos en categorías y subcategorías para mantenerlo organizado.
- Barra de búsqueda: facilitar a los usuarios la búsqueda de productos.
- Carrito de compra: permitir a los usuarios añadir productos a un carrito de compra digital y pagar por todos los artículos en una sola operación. Los productos pueden ser añadidos fácilmente o borrados del carrito de compras.
- Pedidos y pagos en la aplicación. Todo el concepto de *e-commerce* app consiste en la posibilidad de pedir y pagar a través de la aplicación.
- Notificaciones *push*: enviar mensajes personalizados.
- Localización del local: encontrar fácilmente un local cercano.
- Acceso sin conexión a Internet: navegar por la aplicación, visualizar el catálogo y utilizar funcionalidades básicas cuando no se cuenta con acceso a Internet.

6.1.1 Diagrama de casos de uso

El actor que se identificó de la aplicación móvil de Capitano Craft es el Cliente, aquel que se descarga y usa la aplicación para gestionar su compra online en el bar. El diagrama de casos

de usos de la Figura 12 permite identificar cómo éste interactúa en el escenario de la aplicación.

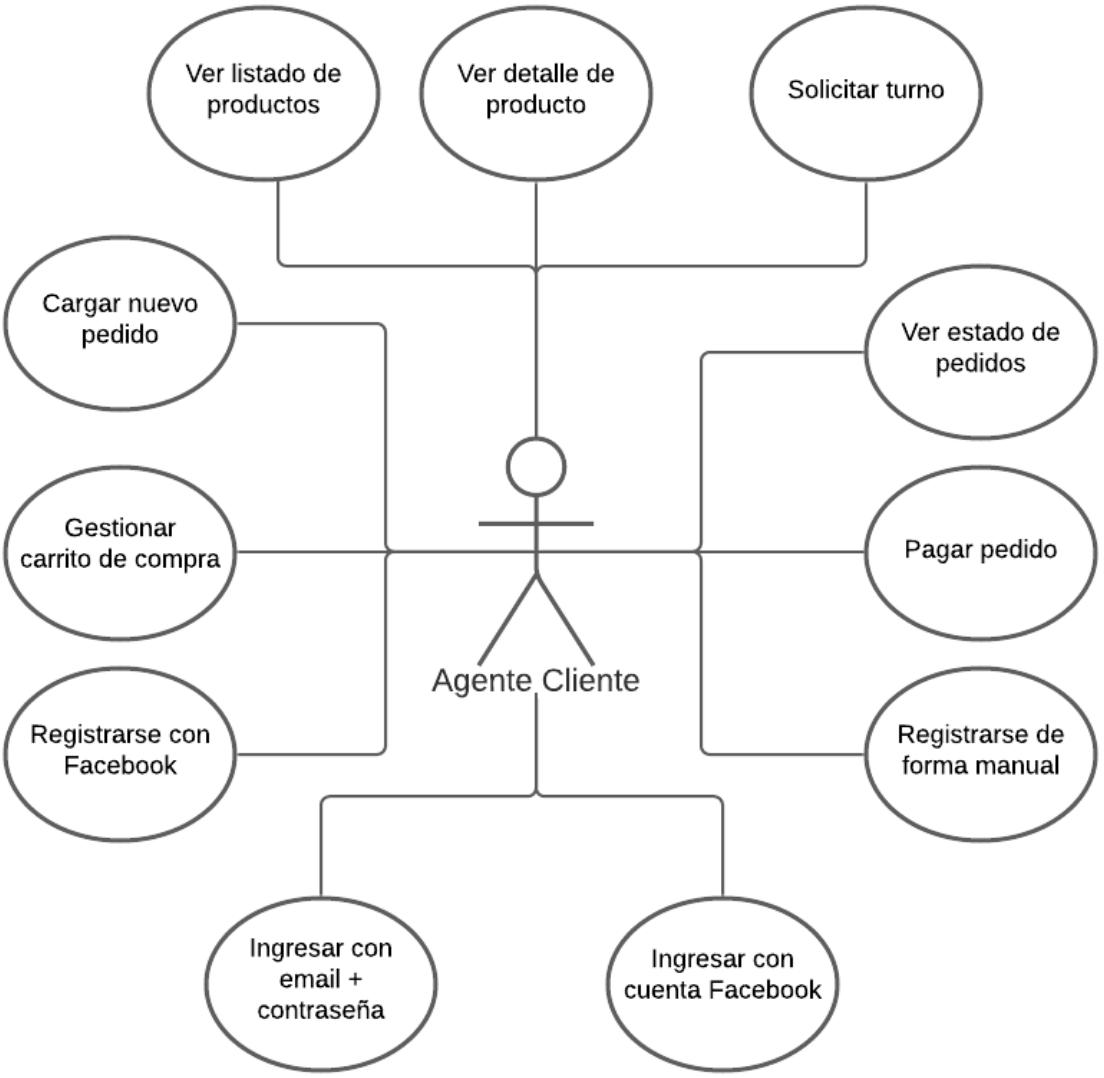

Figura 12: Diagrama de casos de uso para el Agente Cliente.

6.1.2 Diagrama de paquetes

La aplicación de Capitano Craft se construyó utilizando como base una parte del código de la aplicación Iduo. De esta forma se aprovechó el trabajo realizado anteriormente, se economiza el tiempo, y se minimiza la redundancia. En el diagrama de paquetes de la Figura 13 se puede observar aquellos módulos que se reutilizaron y aquellos nuevos que fueron creados para desarrollar las funcionalidades requeridas.

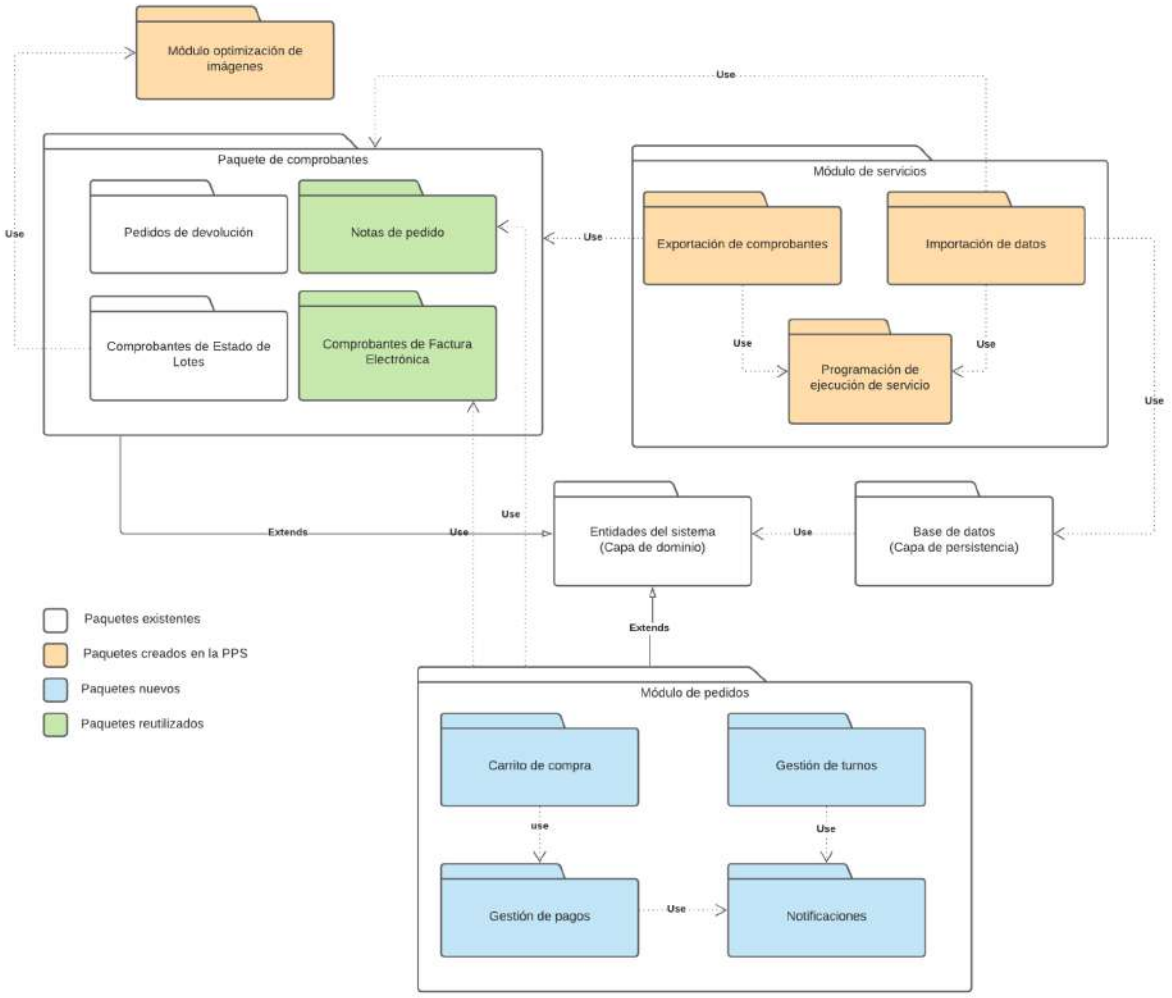

Figura 13: Diagrama de paquetes que se reutilizaron y aquellos que fueron creados para desarrollar las funcionalidades requeridas.

6.1.3 Diagrama de clases

A continuación, se detallan las clases que componen el sistema e-commerce y que sirven de base para el desarrollo de este capítulo. Las clases y sus relaciones son representados en el diagrama de clases de la Figura 14.

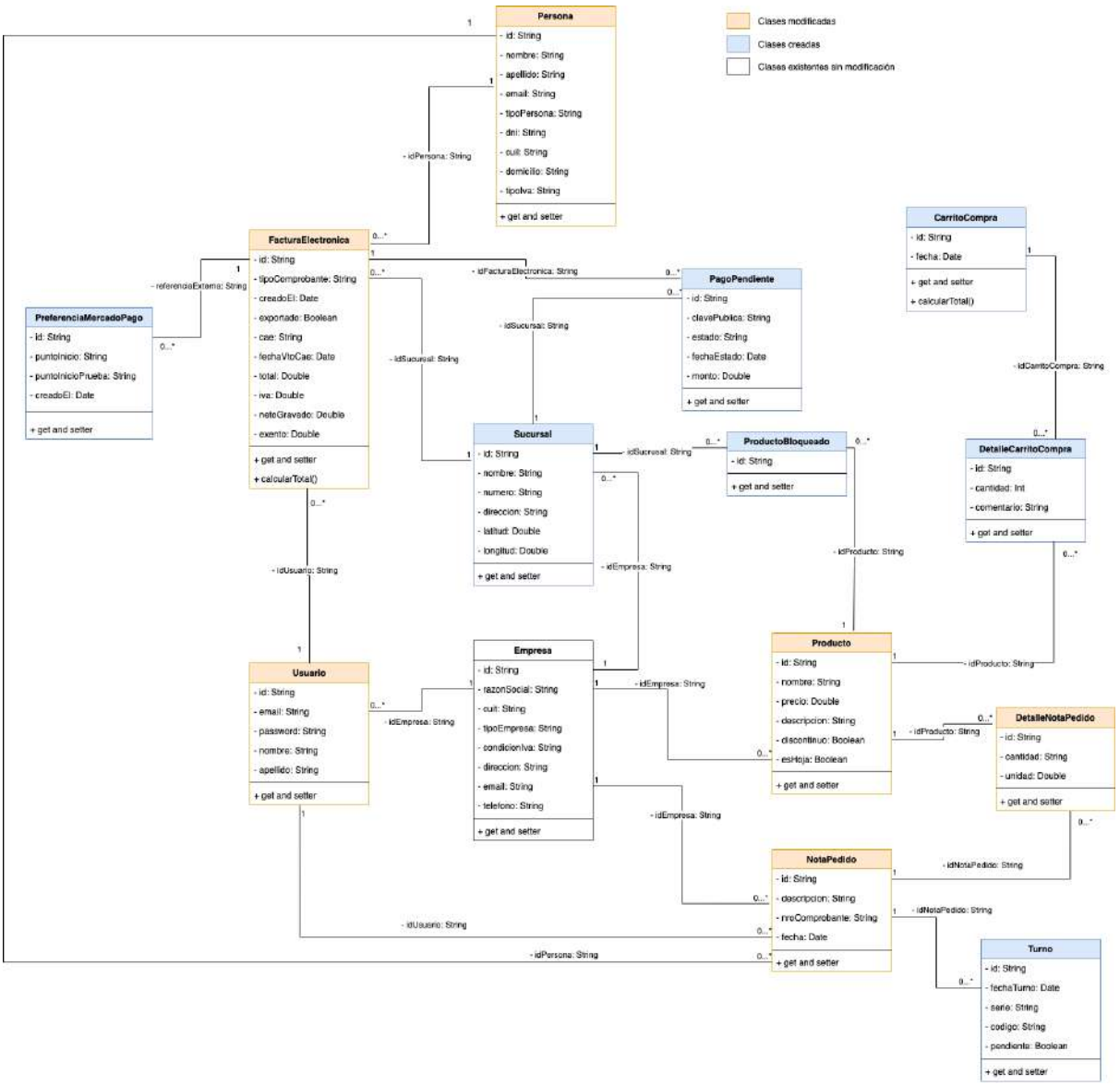

Figura 14: Diagrama de clases clases que componen el sistema e-commerce.

6.1.4 Descripción de la Interfaz de Usuario

En la Figura 15 se describe el esquema de navegabilidad del sistema, cuyo objetivo es describir la navegabilidad y conexiones entre las principales pantallas de la aplicación desarrollada.

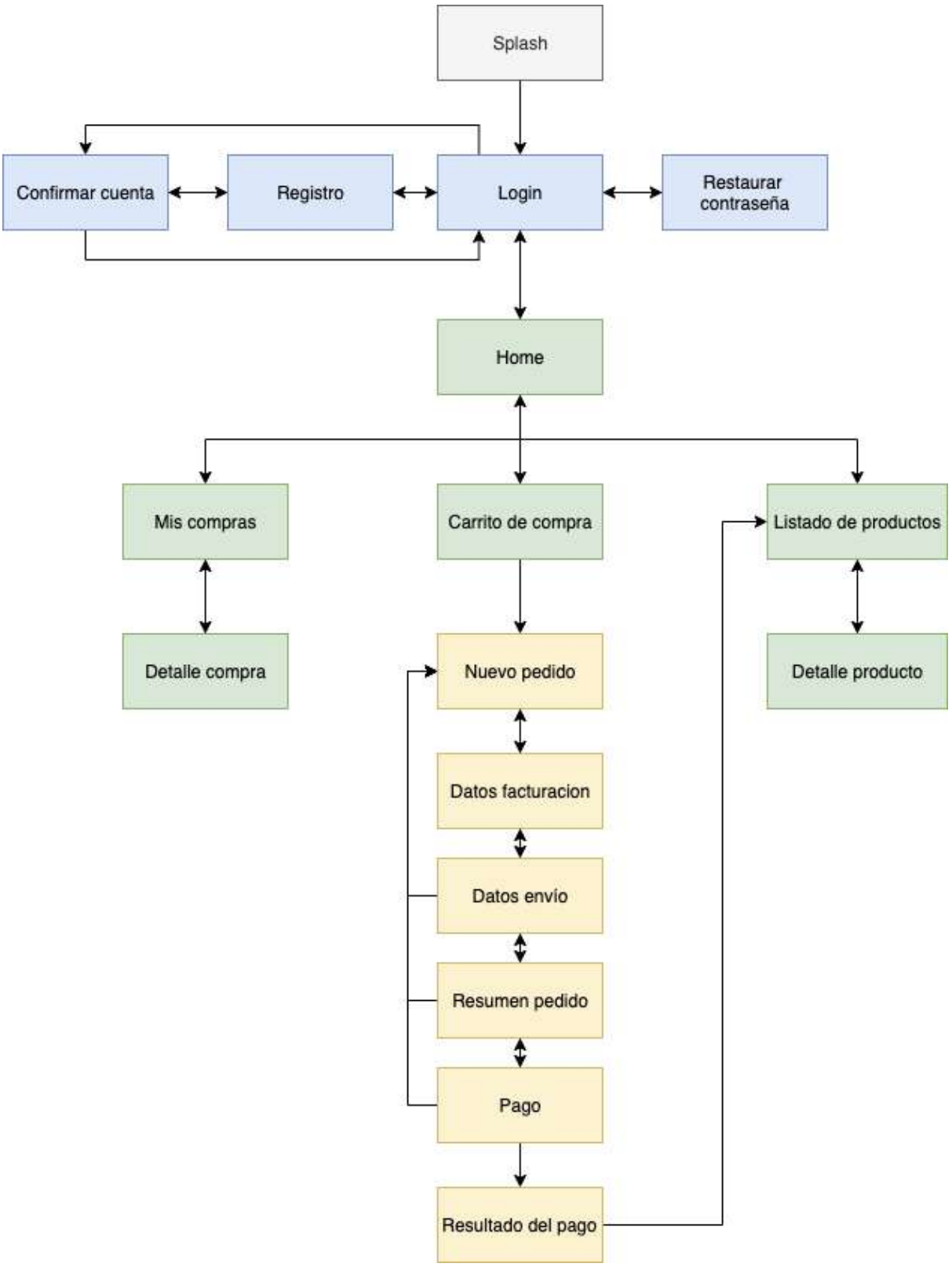

Figura 15: Esquema de navegación de la aplicación Capitano Craft

6.2 Inicio de sesión mediante Facebook

Para poder utilizar la aplicación Iduo y acceder a las funcionalidades básicas, una persona debe estar registrada en el sistema. Para ello se decidió integrar dos opciones de autenticación de un nuevo usuario; la creación de una cuenta utilizando el correo electrónico o el inicio de sesión de Facebook. Esto permite gestionar información personalizada para cada usuario, tales como las compras realizadas o el carrito de compras. La Figura 16 muestra el diseño del inicio de sesión y el registro de usuario, que permiten autenticar e iniciar el proceso de registración de un usuario respectivamente.

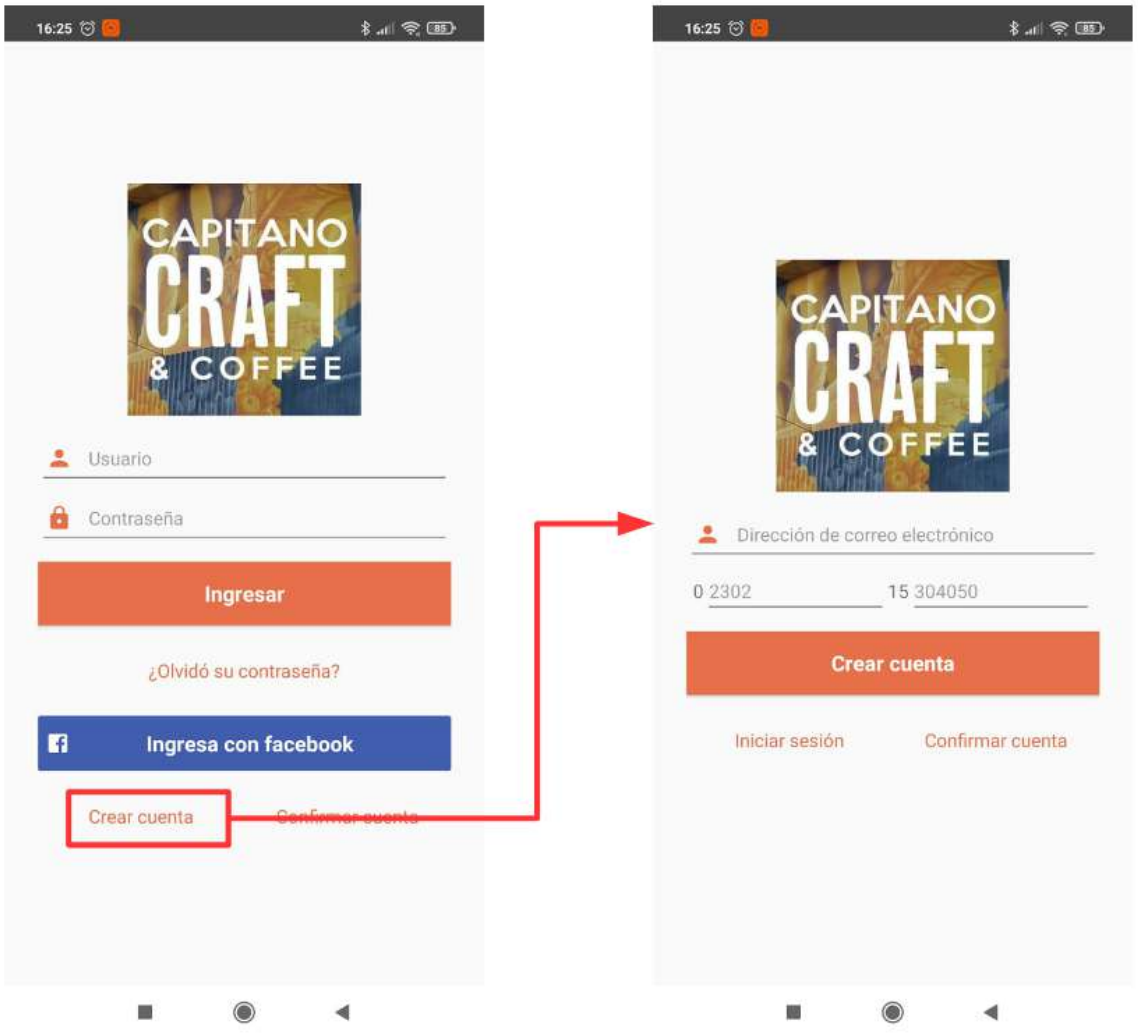

Figura 16: Diseño de las pantallas de Login y Registro

Por lo tanto, el usuario tiene la posibilidad de crear una cuenta de forma manual, utilizando el proceso de creación y validación de cuenta correspondiente, o registrarse utilizando su cuenta existente de Facebook y simplificar el proceso de validación de cuenta. Además, se evita que el usuario tenga que recordar nuevas credenciales de acceso.

Para incorporar esta característica se utilizó el SDK de Facebook el cual permite enlazar el inicio de sesión de esta red social con la aplicación. Al hacerlo, las personas conceden a la aplicación permiso para obtener información o realizar acciones en Facebook

en su nombre, lo cual se utiliza para crear el usuario y registrarlo en la aplicación de forma automática.

En las siguientes secciones se detallan los pasos que se llevaron a cabo para integrar el inicio de sesión mediante Facebook.

6.2.1 Registrar aplicación en la plataforma de desarrolladores de Facebook

Para poder acceder a los servicios de Facebook, se registró la aplicación en el panel de aplicaciones, accediendo desde la plataforma de desarrolladores de Facebook (https://developers.facebook.com/?locale=es_ES). El registro crea un identificador, el cual permite que Facebook pueda identificar la aplicación y distinguirla de otras, proporcionando una forma de solicitar acceso a las APIs de Facebook. También se definió el tipo de plataforma, indicando que los usuarios accederán a la aplicación mediante un dispositivo móvil.

Una vez registrada la aplicación en Facebook, se agregó el identificador generado en los recursos de la aplicación y se agregó un nuevo elemento en el archivo manifiesto. El Código 11 muestra un fragmento del archivo manifiesto donde se agrega el contenedor personalizado sobre el cual se ejecutará el servicio de inicio de sesión de Facebook.

```
<meta-data
    android:name="com.facebook.sdk.ApplicationId"
    android:value="@string/facebook_id" />

<activity
    android:name="com.facebook.FacebookActivity"
    android:configChanges="keyboard|keyboardHidden|screenLayout|screenSize|orientation"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.Translucent.NoTitleBar" />
```

Código 11: fragmento de código del archivo AndroidManifest.xml

6.2.2 Crear el botón de inicio de sesión con Facebook

El botón de inicio de sesión en Facebook es el encargado de controlar el estado de inicio de sesión y de mostrar el texto correcto en función del estado de autenticación. Para crear dicho botón, luego de registrar la aplicación en la plataforma de Facebook, se agregó el elemento visual en el diseño de la vista y se lo enlazó al controlador. Este es un elemento de la interfaz de usuario provisto por el SDK de Facebook, que conforma las funciones disponibles para comenzar el flujo de inicio de sesión. De este modo, al seleccionar el botón se activa el inicio de sesión con los permisos definidos para obtener el correo electrónico y el perfil del usuario. La Figura 17 muestra el código XML correspondiente a la definición del botón y su respectiva vista en el dispositivo real una vez ejecutada la aplicación.

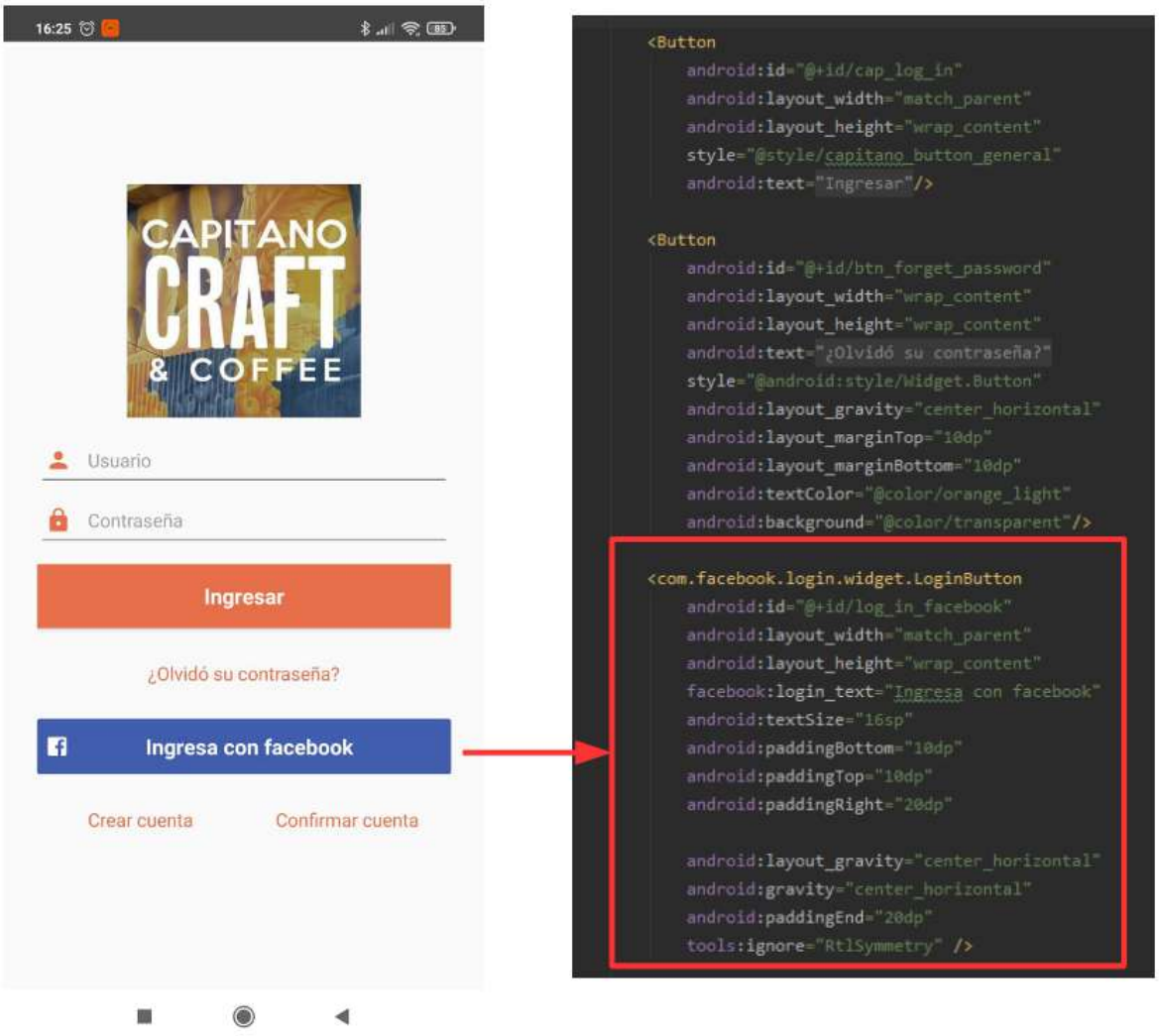

Figura 17: Vista final y código XML del botón de inicio de sesión de Facebook

6.2.3 Verificar el estado de inicio de sesión

La aplicación solo puede iniciar la sesión de un usuario a la vez, estableciendo el *AccessToken* actual para ese usuario. El SDK de Facebook guarda estos datos en las preferencias de uso compartido y los configura al principio de la sesión. Para comprobar si una persona ya tiene sesión iniciada, es decir, realizar el seguimiento de los tokens de acceso, se utiliza la clase *AccessTokenTracker*, la cual permite recibir notificaciones sobre los cambios del token de acceso y de esta forma identificar cuando un usuario se loguea utilizando las credenciales de Facebook o cierra sesión. Por lo tanto, el objeto *AccessToken* contiene los datos sobre el usuario logueado, o null en caso contrario (al momento de cerrar sesión el valor devuelto es null). Para implementarlo se extiende de la clase *AccessTokenTracker* y se sobreescribe el método *onCurrentAccessTokenChanged*. Este método es invocado cada vez que el *AccessToken* cambia de estado.

6.2.4 Validación de cuenta de forma automática utilizando servicios de Google

El código de verificación por SMS es uno de los métodos más seguros para verificar la autentificación de los usuarios, vincular un usuario a su número de teléfono o enviar contraseñas de un solo uso. Las aplicaciones pueden optar por la introducción manual del código de verificación que el usuario recibió por SMS, o por automatizar este proceso, facilitando la tarea al usuario.

En Android, las aplicaciones pueden solicitar al usuario permiso para leer sus SMS. Aunque esto ofrece la ventaja de que los usuarios no tienen que molestarse en leer el código de verificación, también supone un peligro para la privacidad de los usuarios. Es por esto que a principios de 2019, Google restringe las aplicaciones que solicitaban permisos de lectura de SMS por la potencial inseguridad que supone [30]. Como alternativa para reemplazar este tipo de funcionalidad, desarrolló *SMS Retriever API*, la cual permite ejecutar el proceso de verificación por SMS de una forma más segura.

SMS Retriever API es la opción recomendada por Google, ya que ofrece una mejor experiencia a los usuarios, permitiendo completar todo el proceso de verificación en poco tiempo y sin ninguna interacción por parte de los usuarios. De esta forma, se logra ofrecer una experiencia más segura y sin interrupciones. La Figura 18 describe el flujo de dicho proceso.

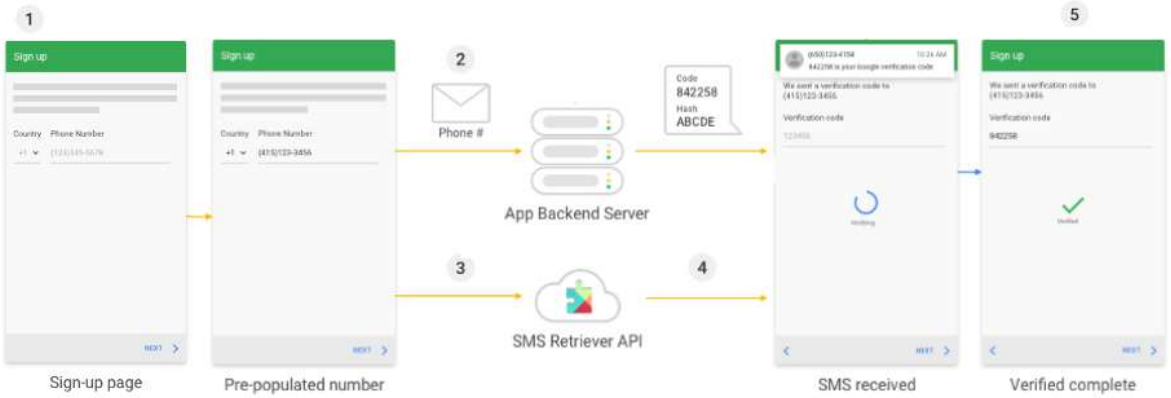

Figura 18: Representación gráfica de SMS Retriever API.

De esta forma, los pasos que se realizan para la validación del número de teléfono son los siguientes:

1. El usuario inicia la verificación por SMS en la aplicación.
2. La aplicación realiza una solicitud al servidor para enviar el SMS al destinatario correspondiente. Se debe incluir en el cuerpo del mensaje un código de 11 dígitos, el cual permite identificar de forma única la aplicación. Se debe incluir también el código de autenticación requerido para la operación que está realizando el usuario.
3. Al mismo tiempo, la aplicación utiliza la API de *SMS Retriever* para comenzar a escuchar una respuesta de SMS.
4. Cuando el dispositivo del usuario recibe el mensaje SMS, los servicios de Google Play utilizan el *hash* de la aplicación para determinar que el mensaje está destinado a

la aplicación y hace que el texto del mensaje esté disponible para la aplicación a través de la API de *SMS Retriever*.

5. La aplicación analiza el contenido del mensaje, válida la procedencia del mensaje y extrae el código de autenticación. Finalmente, el usuario verifica con éxito su cuenta y se registra en la aplicación.

Para integrar este flujo de trabajo en el proyecto, se comenzó agregando las dependencias necesarias de las librerías de *SMS Retriever* en el archivo *gradle*.

Luego, se agregó el método *startSmsListener* en el controlador que lleva adelante el registro del usuario. Este método se encarga de iniciar el cliente de *SMS Retriever* una vez que el usuario haya solicitado al servidor el envío del mensaje de autenticación, tal como muestra el Código 12.

```
private void startSmsRetriever() {

    SmsRetrieverClient client = SmsRetriever.getClient(this /* context */);
    Task<Void> task = client.startSmsRetriever();

    task.addOnSuccessListener(aVoid -> {

        if (activity != null && !activity.isFinishing()){
            Log.d(TAG, "Cliente SMS Retriver iniciado exitosamente");
        }
    });

    task.addOnFailureListener(error -> {
        if (activity != null && !activity.isFinishing()){
            Log.e(TAG, "Error al iniciar cliente SMS Retriever: " + error.getMessage());
        }
    });
}
```

Código 12: Método que inicia el cliente SMS Retriever.

Con el objetivo de escuchar el mensaje que se va a recibir desde la API de *SMS Retriever*, se creó la clase *SmsBroadcastRecevier.* El Código 13 muestra la definición de la clase, la cual hereda de *BroadcastReceiver,* definida anteriormente.

```java
public class SmsBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {

        if (SmsRetriever.SMS_RETRIEVED_ACTION.equals(intent.getAction())) {

            Status status = (Status) intent.getExtras().get(SmsRetriever.EXTRA_STATUS);
            switch(status.getStatusCode()) {
                case CommonStatusCodes.SUCCESS:

                    // Obtengo el contenido del SMS
                    String message = (String) extras.get(SmsRetriever.EXTRA_SMS_MESSAGE);

                    // Extraigo codigo unico para verificar la cuenta
                    String code = fetchVerificationCode(message);

                    Intent confirmAccountActivity = new Intent(context, ConfirmAccountActivity.class);
                    confirmAccountActivity.putExtra(CODE, code);
                    context.startActivity(confirmAccountActivity);
                    break;

                case CommonStatusCodes.TIMEOUT:
                    break;
            }
        }
    }
}
```

Código 13: Definición de la clase SmsBroadcastReceiver.

Además, es necesario registrar el *Receiver* en el archivo manifiesto, de forma que el sistema pueda identificar la clase que se encarga de procesar el mensaje recibido.

Finalmente, una vez que se recibe el mensaje en el *Receiver*, se obtiene el código de verificación y finaliza el proceso de autentificación del usuario. Es necesario tener en cuenta que el mensaje que se envía desde el servidor debe cumplir con el formato de mensaje OTP introducido por Google. La Figura 19 muestra el formato estándar que debe tener el mensaje.

```
<#> SampleApp: Your verification code is 143567
QbwSot12oP
```

Figura 19: Formato estándar de mensaje OTP.

Se puede observar que:

- El mensaje debe comenzar con <#>, lo que permite indicar al sistema que hay un mensaje OTP.
- El mensaje debe terminar con el código *hash* que identifica a la aplicación. El sistema operativo, basándose en este código, pasará el mensaje a la aplicación correspondiente.

6.3 Gestión de pedidos y pagos

Para que los usuarios puedan conocer los productos que ofrece la empresa, navegar entre categorías o visualizar detalles de cada producto, se agregó un catálogo de productos. Esto permite llegar a más clientes, convirtiéndose también en un importante medio publicitario y de promoción. Entre las ventajas que ofrece se encuentran:

- **Mejor organización de la información**: los catálogos permiten clasificar y jerarquizar la información de una manera más efectiva; ya que la navegación guía al usuario a conseguir lo que busca por múltiples maneras de una forma rápida y eficiente.
- **Mayor espacio**: los catálogos digitales permiten incluir gran cantidad de información y ampliarla considerablemente en comparación con el limitado espacio que se tiene en un catálogo impreso. En este sentido, se pueden agregar las características de los productos, su descripción, los consejos o recomendaciones sobre su utilización y mantenimiento.
- **Disponibilidad:** los usuarios pueden consultar el catálogo en todo momento.
- **Realización de búsquedas**: se provee un buscador que permite ubicar, rápida y eficientemente, el producto correspondiente.
- **Actualización rápida**: se pueden hacer las modificaciones pertinentes de un modo rápido y económico; permitiendo así, que la última versión esté disponible de manera instantánea. Además, es posible agregar o quitar productos, cambiar los precios o editar las características.
- **Impacto visual**: un catálogo virtual brinda la oportunidad de incorporar imágenes, que permiten captar con mayor rapidez y efectividad la atención de los clientes.
- **Posicionamiento**: un catálogo genera mayor presencia de la marca y de los productos que ofrece. Al igual que, facilita el proceso de dar a conocer nuevos productos o de mantener en la mente del consumidor los ya existentes. Así como también, al ser una representación gráfica de la empresa y de los productos que vende; se refuerza la imagen que los clientes tienen de la organización.
- **Acceso permanente**: permite que los productos que ofrece la empresa puedan ser consultados en cualquier momento del día.
- **Comercio electrónico**: permite recibir pedidos en línea; habilitando un sistema de pago tradicional (efectivo) o por Internet (mediante Mercado Pago).

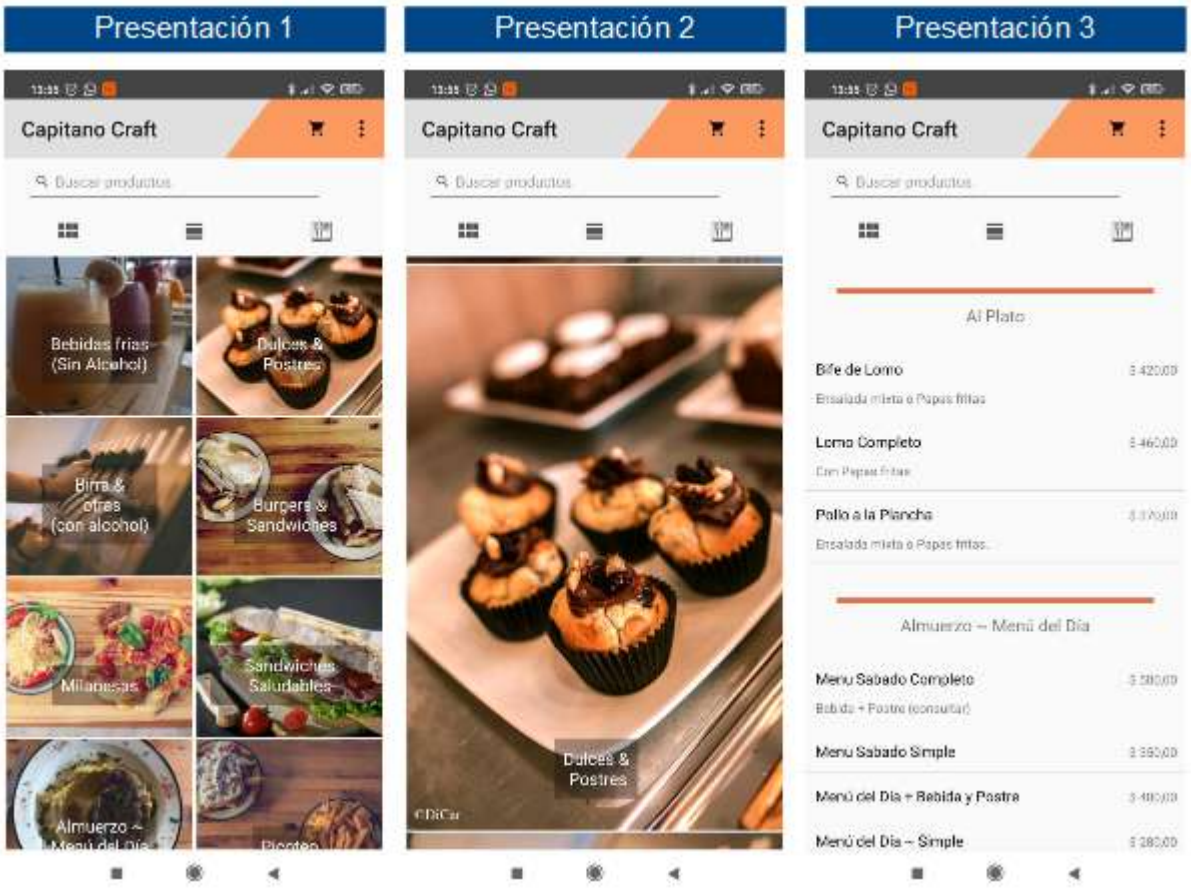

Figura 20: Presentación del catálogo de productos en 3 vistas diferentes.

La Figura 20 muestra el diseño del catálogo principal de la aplicación, donde la información puede ser presentada de tres maneras diferentes:

- **Presentación 1**: se presentan las categorías de productos en forma de cuadrilla. Cuando el usuario selecciona una categoría se despliega una lista con los productos correspondientes a dicha categoría.
- **Presentación 2**: se presentan las categorías de productos en forma de lista. De igual manera, cuando el usuario selecciona una categoría se despliega la lista con los productos correspondientes a dicha categoría.
- **Presentación 3**: se presentan todas las categorías con sus productos en forma de lista. Esto permite que el usuario visualice todos los productos que ofrece la empresa segmentados por categoría.

Cuando el usuario selecciona un producto en particular, navega a una nueva pantalla donde se muestra la ficha de dicho producto. A partir de aquí puede agregar el producto al carrito de compras, definiendo la cantidad requerida.

En el proceso de compra de un *e-commerce* interviene un elemento muy importante como es el carrito de la compra. Éste se utiliza para gestionar los pedidos de los usuarios. En la aplicación se integró un carrito de compras, el cual permite mostrar los productos que el usuario ha añadido y también editar la compra. Esto posibilita mejorar la experiencia del usuario y facilitar la preparación del proceso de compra.

El proceso de compra, por parte del usuario, consiste en una serie de pasos que permiten obtener los datos necesarios para que la empresa realice la venta, como son los datos de la persona que realiza el pedido, forma de entrega del producto y forma de pago. Así, el usuario navega entre distintas pantallas de acuerdo a las opciones seleccionadas. Cuando el usuario confirma el pedido, se crea un nuevo comprobante y se lo exporta al sistema central.

Luego, de acuerdo a la forma de pago seleccionada por el usuario, se distinguen dos posibles casos:

- **Pago en efectivo**: en este caso el proceso de compra finaliza y se indica al usuario sobre el resultado correspondiente.
- **Pago mediante Mercado Pago**: en este caso el proceso debe continuar, utilizando los servicios que ofrece Mercado Pago para completar el pago, y finalmente registrar el resultado del pago en la aplicación y en el sistema web.

La Figura 21 muestra una serie de pantallas que permiten visualizar el proceso de pago completo utilizando Mercado Pago como forma de pago del pedido realizado.

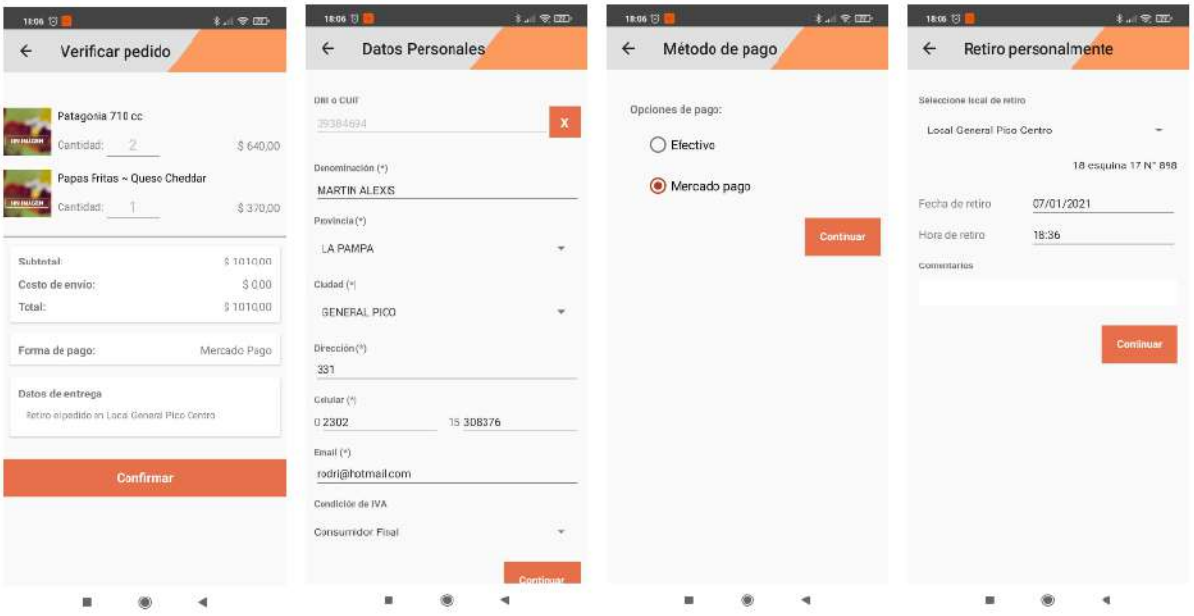

Figura 21: Pantallas correspondientes al proceso de generación de pedidos

En el siguiente apartado se describe cómo se integró Mercado Pago a la aplicación, con el objetivo de poder realizar pagos electrónicos.

6.3.1 Pago de pedidos utilizando servicios de Mercado Pago

Con el objetivo de brindar una experiencia completa en la gestión de pedidos, se integró la posibilidad de realizar pagos dentro de la aplicación. Para esto se utilizó el SDK de Mercado Pago, una librería que ofrece a los usuarios la posibilidad de realizar pagos con tarjeta o transferencia bancaria y financiación (cuotas).

Para comenzar a utilizar la librería se agregaron las dependencias correspondientes en el archivo *gradle*. Luego se integró el componente visual de pago de Mercado Pago, el cual

permite integrar el flujo de pago de Mercado Pago en Android. Este componente se encarga de manejar la selección del medio de pago, la recolección de datos del medio de pago del usuario y la comunicación del resultado de pago. La integración consta de dos etapas:

A. **Integración en el servidor**: en esta etapa se gestiona la información del pago.
B. **Integración en la aplicación**: en esta etapa se configura el componente visual.

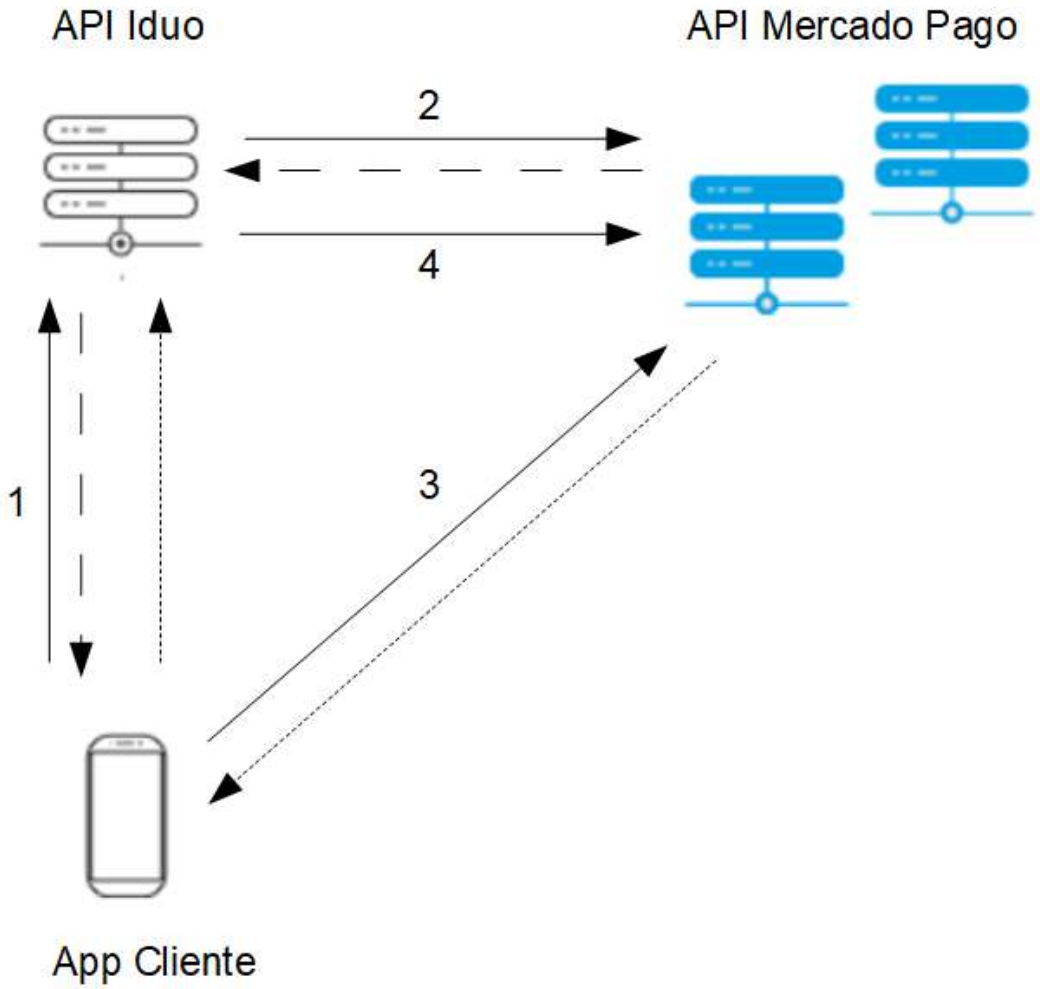

Figura 22: Proceso de pago utilizando Mercado Pago.

La Figura 22 muestra el proceso de pago con los pasos que se llevan a cabo, descritos a continuación:.

1. El proceso inicia cuando la aplicación solicita la creación de la preferencia de pago al servidor, enviando los datos correspondientes de la compra realizada (contenido de la preferencia).
2. El servidor cuenta con la información necesaria para crear la preferencia de pago en los servidores de Mercado Pago, como así también el pago pendiente del comprobante. El pago pendiente generado contiene información sobre el estado del pago del pedido. Los datos de la preferencia creada son procesados por el servidor y se envían a la aplicación.

3. Una vez que la aplicación recibe la respuesta del servidor, con el identificador de la preferencia, se inicia el *Checkout*. El *Checkout* es el encargado de realizar el pago en los servidores de Mercado Pago.
4. Finalmente, desde el servidor se suscribe a las notificaciones de Mercado Pago para enterarse sobre nuevos pagos realizados y las actualizaciones de sus estados.

Por lo tanto, una vez creada la preferencia de pago y definido el evento a partir del cual comenzar el flujo de pago (cuando el usuario da click en el botón de pago), se puede iniciar el *Checkout*, tal como muestra el Código 14.

```
private void startMercadoPagoCheckout(String publicKey, String preferenceId) {
    new MercadoPagoCheckout.Builder(publicKey, preferenceId)
            .build()
            .startPayment(CheckOrderActivity.this, REQUEST_CODE);
}
```

Código 14: Fragmento de código que inicia el flujo de pago.

Hay que tener en cuenta que el SDK de Mercado Pago siempre devuelve un resultado del pago, por lo que si hubo algún error o el usuario abandonó el flujo, devolverá un objeto representando el error. Los atributos más importantes del pago son los siguientes:

- **id**: identificador del pago.
- **status**: estado del pago.
- **payment_method_id**: identificador del medio de pago que eligió el usuario.
- **payment_type_id**: tipo de medio elegido.
- **card**: objeto que identifica la tarjeta del usuario.
- **issuer_id**: identificador del banco de la tarjeta que eligió el usuario.
- **installments**: cantidad de cuotas elegidas.

Finalmente, para obtener la respuesta de pago del *Checkout* se utiliza el código de identificación de solicitud *REQUEST_CODE,* que se envió anteriormente .

En caso de que el proceso de pago haya finalizado con éxito, se obtiene el objeto *Payment* con los datos de la respuesta. Este objeto es procesado en la aplicación para identificar el estado del pago y actualizar el pago pendiente en el sistema web, tal como se muestra en la Código 15.

```
@Override
protected void onActivityResult(final int requestCode, final int resultCode, final Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == REQUEST_CODE) {
        if (resultCode == MercadoPagoCheckout.PAYMENT_RESULT_CODE) {
            final Payment payment = (Payment) data.getSerializableExtra(MercadoPagoCheckout.EXTRA_PAYMENT_RESULT);
            checkOrderViewModel.processCheckout(payment, mpPreference,
                    salesReceipt.getId());
        } else if (resultCode == RESULT_CANCELED) {
            if (data != null && data.getExtras() != null
                    && data.getExtras().containsKey(MercadoPagoCheckout.EXTRA_ERROR)) {
                final MercadoPagoError mercadoPagoError =
                        (MercadoPagoError) data.getSerializableExtra(MercadoPagoCheckout.EXTRA_ERROR);
                showErrorDialog(mercadoPagoError.getMessage());
            }
        }
    }
}
```

Código 15: Método que se encarga de procesar el resultado del pago realizado con Mercado Pago.

6.4 Solicitud y gestión de turnos asociados a un pedido

Cuando un cliente hace un pedido en Capitano Craft, se le otorga un turno. Una vez que el pedido fue elaborado, el cliente se presenta con el turno y retira el pedido. Con el objetivo de facilitar la gestión de los turnos y la operatoria diaria que esto implica, se agregó la entidad *TURNO* en la aplicación. Para ello se incorporó la tabla *TURNO* a la base de datos, junto con sus relaciones correspondientes.

Por lo tanto, cada vez que un usuario realiza un pedido desde la aplicación, se genera automáticamente un turno, el cual es almacenado tanto en el sistema central como en la aplicación. Esto permitió desarrollar un sistema de notificaciones para mantener informado al usuario sobre el estado de su turno y notificar cuando el pedido esté listo.

Para llevar a cabo esta nueva funcionalidad, se desarrolló un nuevo servicio que permite consultar el estado de un turno en el sistema y notificar al usuario en caso de que se produzca un cambio en el estado del mismo. Este proceso se inicia cada vez que se confirma un nuevo pedido y continúa ejecutándose cada 5 minutos hasta que el turno se indique como finalizado o transcurran 7 días desde que se generó el pedido.

Para la implementación del servicio se utilizaron los conceptos mencionados en la sección 4.2, siguiendo la misma metodología para ejecutar automáticamente un proceso. Por este motivo, se consideró omitir el detalle de las clases y métodos creados para llevar a cabo la funcionalidad.

También se desarrolló un panel de administración de pedidos. Este panel posibilita que los usuarios vean información como fecha del pedido, número del turno, estado del turno, estado del pago, comprobante generado.

Una notificación, en su forma más básica y compacta (también conocida como forma contraída), muestra un ícono, un título y una pequeña cantidad de texto contextual. El diseño de una notificación se determina mediante plantillas del sistema, por lo que se definió el contenido de cada parte de la plantilla.

Para configurar el contenido de la notificación, se definió el contenido y el canal de la notificación usando un objeto *NotificationCompat.Builder*. En el Código 16 se muestra cómo se creó la notificación con los siguientes elementos:

- Un pequeño ícono, establecido por *setSmallIcon*, que es el único contenido necesario visible para el usuario.
- Un título, establecido por *setContentTitle*.
- El texto del cuerpo, establecido por *setContentText*.
- La prioridad de la notificación, establecida por *setPriority*. La prioridad determina cuán intrusiva debería ser la notificación en Android 7.1 y versiones anteriores. (Para Android 8.0 y posteriores, debes establecer la importancia del canal, lo que se muestra en esta sección).

```
Notification.Builder builder = new NotificationCompat
    .Builder(@activity:this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle(textTitle)
    .setContentText(textContent)
    .setPriority(NotificationCompat.PRIORITY_HIGH);
```

Código 16: Fragmento que muestra la creación del objeto Notification.

Se proporcionó un ID del canal (indicado como *CHANNEL_ID*), que se requiere por motivos de compatibilidad con Android 8.0 (API nivel 26) y versiones posteriores, pero se ignora en versiones anteriores. Esto se debe a que, a partir de Android 8.0, todas las notificaciones deben asignarse a un canal o, de lo contrario, no aparecerán. Por este motivo, se registró el canal de notificación de la aplicación en el sistema transfiriendo una instancia de *NotificationChannel* en el método *createNotificationChannel*, como se muestra en el Código 17.

```
private void createNotificationChannel(){
    // Crear un canal de notificaciones para API 26+
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O){
        CharSequence name = getString(R.string.channel_name);
        String description = getString(R.string.channel_description);
        int importance = NotificationManager.IMPORTANCE_DEFAULT;
        NotificationChannel channel = new NotificationChannel(
            CHANNEL_ID,
            name,
            importance
        );
        channel.setDescription(description);
        NotificationManager manager = getSystemService(NotificationManager.class);
        manager.createNotificationChannel(channel);
    }
}
```

Código 17: Método que inicia el canal de notificaciones teniendo en cuenta la versión de Android del dispositivo móvil.

Debido a que se debe crear el canal de notificación antes de publicar notificaciones en Android 8.0 y versiones posteriores, este código se ejecuta al iniciarse la aplicación. Se

consideró que realizar llamadas repetidas es seguro porque cuando se intenta crear un canal de notificación con un identificador existente, no se lleva a cabo ninguna operación.

Al categorizar las notificaciones en canales, los usuarios pueden desactivar canales de notificaciones específicos de la aplicación (en lugar de desactivar todas las notificaciones), así como controlar opciones visuales y auditivas para cada canal, todo desde la configuración del sistema Android.

Los usuarios también pueden mantener presionada una notificación para cambiar los comportamientos del canal asociado. El canal es también donde se especifica el nivel de importancia de las notificaciones en Android 8.0 y versiones posteriores. Por lo tanto, todas las notificaciones del mismo canal tienen el mismo comportamiento.

Hay que tener en cuenta que el constructor *NotificationChannel* requiere definir un nivel de importancia y para ello se usa una de las constantes de la clase *NotificationManager*. Este parámetro determina cómo interrumpir al usuario con cualquier notificación que pertenezca a este canal, aunque también se definió la prioridad con *setPriority* para admitir Android 7.1 y versiones anteriores (como se muestra en la Código 18). La importancia de una notificación se utiliza para determinar hasta qué punto puede interrumpir al usuario (de manera visual y audible). Cuanto mayor sea la importancia de una notificación, mayor será la interrupción.

Los posibles niveles de importancia son los siguientes:

- Urgente: emite un sonido y aparece como una notificación emergente.
- Alta: emite un sonido.
- Media: no se emite ningún sonido.
- Baja: no se emite ningún sonido y no aparece en la barra de estado.

Si bien se debe definir la importancia/prioridad de la notificación, el sistema no garantiza el comportamiento de alerta que se obtendrá. En algunos casos, el sistema puede cambiar el nivel de importancia en función de otros factores, y el usuario siempre puede redefinir el nivel de importancia para un canal determinado.

También se definió una acción de toque de la notificación, la cual permite abrir una actividad en la aplicación (que se corresponda con la notificación) y mostrar información ampliada. Para hacerlo, se especificó un *Intent* de contenido definido con un objeto *PendingIntent* y se añadió mediante el método *setContentIntent*. En el Código 18, se muestra cómo se creó un *Intent* para abrir una actividad cuando el usuario presiona la notificación.

```
Intent intent = new Intent(this, NotificationDetail.class);
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);

Notification.Builder builder = new NotificationCompat
    .Builder(@activity:this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle(textTitle)
    .setContentText(textContent)
    .setContentIntent(pendingIntent)
    .setPriority(NotificationCompat.PRIORITY_HIGH);
```

Código 18: Fragmento de código que se encarga de crear el Intent de contenido.

En el código anterior se llama a *setAutoCancel*, el cual quita automáticamente la notificación cuando el usuario la presiona. Por otro lado, el método *setFlags* permite conservar la experiencia de navegación prevista del usuario después de que este abre la aplicación desde la notificación. De esta forma, se define una actividad que existe exclusivamente para respuestas a la notificación, de modo que inicia una nueva tarea, en lugar de agregarse a la pila de actividades existente de la app.

Finalmente, para mostrar la notificación informando el cambio de estado del turno se llama al método *notify* desde el servicio en ejecución, pasándole como parámetro el ID único para la notificación y el objeto *NotificationCompat* construido a partir de la clase *Builder* provista, como lo muestra la Código 19.

```
NotificationManagerCompat manager = NotificationManagerCompat.from(this);

// notificationId debe ser único para cada notificación que se defina
manager.notify(notificationId, builder.build());
```

Código 19: Fragmento de código que se encarga de mostrar una notificación en el dispositivo móvil.

Conclusión

Durante el desarrollo del presente trabajo se destacaron una serie de características y desafíos que hicieron de ésta una experiencia muy completa, permitiendo adquirir nuevas habilidades tanto en los personal como en lo profesional:

- El grupo de desarrollo fue muy atento durante todo el transcurso del proyecto, permitiendo trabajar activamente y avanzar de forma muy rápida en la comprensión de los procesos de negocios actuales y los requerimientos que se debían abordar.
- El ambiente de trabajo fue muy cómodo, donde se dispuso en todo momento de los recursos y herramientas necesarias para el desarrollo de la práctica.
- Se trabajó sobre una aplicación existente, desarrollada en un lenguaje de programación y plataforma definidos. Esto presentó el desafío de modificar y extender una aplicación que se utilizaba hasta ese momento en un ambiente real.
- Optimizar la gestión del tiempo en el desarrollo de las distintas tareas a realizar con el objetivo de lograr la mayor eficiencia posible, teniendo en cuenta todas las restricciones del proyecto y ajustándose a un cronograma y planificación definido.
- Adquirir nuevos conocimientos en cuanto al desarrollo de una aplicación móvil para el sistema operativo Android, mediante la utilización del entorno de desarrollo integrado (Integrated Development Environment, o IDE, en inglés) Android Studio. De esta forma, se logró una mayor comprensión de los componentes y funcionalidades necesarias en la construcción de una aplicación móvil.

El uso de Java, conocido mundialmente, facilitó el desarrollo debido a la gran cantidad de documentación y librerías existentes. Esto permitió optimizar los tiempos de desarrollo y mejorar el desempeño de la aplicación.

Finalmente, con el objetivo de dar un cierre al presente trabajo, se presentan algunos comentarios finales, en cuanto a los resultados obtenidos:

- Se desarrolló y brindó a la empresa una aplicación más completa, con más funciones y una mejor calidad interna. Con la incorporación del documento de factura electrónica, se logró que los usuarios puedan realizar operaciones como cargar, listar o visualizar dichos documentos de una forma más sencilla, intuitiva y rápida.
- Se desarrolló una aplicación desde cero, lo que permitió aplicar los conocimientos y las técnicas adquiridas en la facultad en todo el ciclo de vida del desarrollo de un software.

Referencias

1. Android. [En línea] https://es.wikipedia.org/wiki/Android [Último acceso: 18 Febrero 2021]
2. Qué es Dalvik. [En línea] https://www.xatakandroid.com/sistema-operativo/que-es-android [Último acceso: 18 Febrero 2021]
3. Software Libre. EcuRed [En línea] https://www.ecured.cu/Software_libre#:~:text=Seg%C3%BAn%20la%20Free%20Software%20Foundation,programa%2C%20con%20cualquier%20prop%C3%B3sito%3B%20de [Último acceso: 18 Febrero 2021]
4. Historia de Android. [En línea] https://www.xatakandroid.com/sistema-operativo/historia-y-evolucion-de-android-como-un-sistema-operativo-para-camaras-digitales-acabo-conquistando-los-moviles#:~:text=Android%20Inc.,ubicaci%C3%B3n%20y%20preferencias%20del%20usuario%22. [Último acceso: 18 Febrero 2021]
5. Android Developers. Arquitectura de la plataforma. [En línea] https://developer.android.com/guide/platform?hl=es-419#:~:text=Android%20es%20una%20pila%20de,principales%20de%20la%20plataforma%20Android. [Último acceso: 18 Febrero 2021]
6. Núcleo. Android Open Source. [En línea] https://source.android.com/devices/architecture/kernel [Último acceso: 18 Febrero 2021]
7. Capa de abstracción de hardware. [En línea]. https://es.wikipedia.org/wiki/Capa_de_abstracci%C3%B3n_de_hardware. [Último acceso: 18 Febrero 2021]
8. Android Developers. ART [En línea] https://developer.android.com/guide/practices/verifying-apps-art?hl=es-419 [Último acceso: 18 Febrero 2021]
9. Android Developers. API nativas [En línea] https://developer.android.com/ndk/guides/stable_apis?hl=es-419 [Último acceso: 18 Febrero 2021]
10. Descripción general de las funciones y API [En línea] https://developer.android.com/about/versions/11/features?hl=es [Último acceso: 18 Febrero 2021]
11. Componentes en Android [En línea] https://desarrolloweb.com/articulos/6-componentes-basicos-android.html [Último acceso: 18 Febrero 2021]
12. AsyncTask: Tareas Asíncronas en Android [En línea] https://www.develou.com/android-asynctask-hilos/ [Último acceso: 18 Febrero 2021]
13. SQLite para Android [En línea] https://openwebinars.net/blog/sqlite-para-android-la-herramienta-definitiva/ [Último acceso: 18 Febrero 2021]

14. Asignación objeto-relacional [En línea] https://es.wikipedia.org/wiki/Asignaci%C3%B3n_objeto-relacional [Último acceso: 18 Febrero 2021]
15. Introducción a ORM y GreenDAO [En línea] https://academiaandroid.com/introduccion-a-orm-y-greendao/ [Último acceso: 18 Febrero 2021]
16. ¿Qué es el diseño responsivo? [En línea] https://trazada.com/que-es-el-diseno-responsivo/ [Último acceso: 18 Febrero 2021]
17. Material Design. Android. [En línea] https://material.io/develop/android [Último acceso: 18 Febrero 2021]
18. Arquitectura REST. Concepto y fundamentos [En línea] https://gausswebapp.com/arquitectura-rest.html#:~:text=REST%20es%20una%20arquitectura%20de,utilizada%20en%20cualquier%20cliente%20HTTP.&text=Es%20decir%2C%20toda%20la%20informaci%C3%B3n,y%20almacenamiento%20interno%20del%20servidor. [Último acceso: 18 Febrero 2021]
19. Servicios Web de RESTful [En línea] https://developer.ibm.com/es/technologies/web-development/articles/ws-restful/ [Último acceso: 18 Febrero 2021]
20. Retrofit [En línea] https://programacionymas.com/blog/consumir-una-api-usando-retrofit [Último acceso: 18 Febrero 2021]
21. Lenguaje unificado de modelado (UML) [En línea] https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado [Último acceso: 18 Febrero 2021]
22. Modelo Vista Presentador (MVP). Develapps [En línea] http://www.develapps.com/es/noticias/modelo-vista-presentador-mvp-en-android#:~:text=Los%20patrones%20de%20arquitectura%2C%20nos,la%20l%C3%B3gica%20de%20nuestras%20aplicaciones. [Último acceso: 18 Febrero 2021]
23. Manual de Git [En línea] https://desarrolloweb.com/manuales/manual-de-git.html [Último acceso: 18 Febrero 2021]
24. Técnicas para el procesamiento de imágenes. [En línea] https://www.famaf.unc.edu.ar/~pperez1/manuales/cim/cap2.html [Último acceso: 18 Febrero 2021]
25. Crear un servicio en Android. Código Facilito. [En línea] https://codigofacilito.com/articulos/crear-un-servicio-en-android [Último acceso: 18 Febrero 2021]
26. Servicios en background. Android Developers. [En línea] https://developer.android.com/training/run-background-service/send-request?hl=es [Último acceso: 18 Febrero 2021]
27. AlarmManager and BroadcastReceiver. [En línea] https://codeplayon.medium.com/how-to-use-alarm-manager-and-broadcast-receiver-in-android-aa91415be8e3 [Último acceso: 18 Febrero 2021]
28. GreenDAO: Android ORM. [En línea] https://greenrobot.org/greendao/ [Último acceso: 18 Febrero 2021]

29. El lenguaje unificado de modelado. [En línea] https://ingenieriasoftware2011.files.wordpress.com/2011/07/el-lenguaje-unificado-de-modelado-manual-de-referencia.pdf [Último acceso: 02 Octubre 2021]
30. Android automatic SMS verification - Google's SMS retriever API. [En línea] https://medium.com/android-dev-hacks/autofill-otp-verification-with-latest-sms-retriever-api-73c788636783 [Último acceso: 24 Marzo 2021]