TRABAJO DE DIPLOMA

Para optar por el título de Ingeniería Informática

Título: Aplicación para dispositivos móviles para la gestión de asignaciones de productos a trabajadores del Centro de Software Libre (CESOL) de la Universidad de las Ciencias Informáticas

Autor: Lian Felipe Hernández Delgado

Tutores: Ing. Ivelisse Montero Jiménez

MSc. Nurileidis Almeida Cintra

Consultor: Ing. Abel Ochoa Izquierdo

LA HABANA, DICIEMBRE DEL 2022

AÑO 63 DE LA REVOLUCIÓN

Declaración de autoría

Declaro ser autora de la presente tesis que tiene por título: “Aplicación para dispositivos móviles para la gestión de asignaciones de productos a trabajadores del Centro de Software Libre (CESOL) de la Universidad de las Ciencias Informáticas”; y se reconoce a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____

Lian Felipe Hernández Delgado

Autor

_____ _____

MSc. Nurileidis Almeida Cintra Ing. Ivelisse Montero Juménez

Tutor Tutor

DEDICATORIA

A mi madre por brindarme el apoyo incondicional durante todo mi paso por la universidad, por dotarme de todos los recursos necesarios. A mi padre por siempre creer en mi talento y enorgullecerse de saber que estudiaba en la mejor universidad del país. A mi novia por conocerme desde cero y tener la paciencia para haber superado muchos momentos difíciles en este camino. Y a todas aquellas personas especiales que adornaron este viaje de alguna forma, mis abuelos, mis amigos (los verdaderos), mis hermanos, mis tutores y consultantes. A todos ellos va dedicado este trabajo de diploma.

Índice de Contenidos

Índice de figuras

Índice de tablas

Resumen

Durante el transcurso de la pandemia provocada por el coronavirus, el mundo cayó en una profunda crisis económica que conllevó a la escasez de muchos recursos, sobre todo de los artículos de primera necesidad. Nuestro país no estuvo exento de dicha crisis, lo cual lo llevó a la regulación de las compras de los diferentes productos en las unidades comercializadoras y posteriormente a idear un sistema de entrega de varios de estos a los trabajadores a modo de intentar cubrir sus necesidades. La Universidad de las Ciencias Informáticas comenzó también a implementar un sistema de distribución de estos productos dentro de sus instalaciones mediante el órgano popular de los trabajadores, el Sindicato. Estos productos son distribuidos mediante el secretario del sindicato de cada centro, y debido a lo engorroso que resulta el proceso de distribución de estos productos, se tuvo en cuenta el desarrollo de una aplicación móvil para mediar en este proceso y agilizarlo. Este trabajo plantea describir el proceso de desarrollo de “Asignador de Productos”, una aplicación para gestionar los trabajadores de cada centro, sus hijos, así como los productos a asignar y sus diferentes asignaciones por cada trabajador.

Palabras clave: Android, Asignador de Productos, gestión, productos, trabajadores.

Abstract

During the course of the pandemic caused by the coronavirus, the world fell into a deep economic crisis that led to the scarcity of many resources, especially basic necessities. Our country was not exempt from this crisis, which led it to regulate the purchases of different products in the marketing units and later to devise a delivery system for several of these to workers in order to try to cover their needs. The University of Informatics Sciences also began to implement a distribution system for these products within its facilities through the workers' popular body, the Union. These products are distributed through the union secretary of each center, and due to the cumbersome process of distributing these products, the development of a mobile application was taken into account to mediate in this process and speed it up. This work proposes to describe the development process of "Product Assigner", an application to manage the workers of each center, their children, as well as the products to be assigned and their different assignments for each worker.

Keywords: *Android, management, products, Product Assigner, worker*

INTRODUCCIÓN

La gestión de la información no es más que el proceso de organizar, evaluar, presentar, comparar los datos en un determinado contexto, controlando su calidad, de manera que esta sea veraz, oportuna, significativa, exacta y útil y que esta información esté disponible en el momento que se le necesite. Ella se encamina al manejo de la información, documentos, metodologías, informes, publicaciones, soportes y flujos en función de los objetivos estratégicos de una organización.

Gestión de la información se percibe como una actividad sin límites definidos, genérica, sin rasgos conceptualmente diferenciados. La Gestión de la información tiene como objetivo optimizar la utilidad y contribución de los recursos de información con el fin de alcanzar los objetivos de la organización. En este sentido, la práctica de la Gestión de la información se traduce en la creación de canales y medios para transmitir y acceder a la información, así como, en añadirle valores a ésta. (Vidal Ledo, y otros, 2012).

En las empresas ha evolucionado radicalmente el tema de gestión de la información, ya que esto no consiste sólo en administrar los flujos de información y el adecuado suministro a los usuarios internos que la necesiten, sino que, es también un marco para establecer líneas de acción y decisiones dentro de las organizaciones. Ha surgido un crecimiento exponencial del valor de la información en las empresas, que ha definido que los usos de los sistemas de gestión de la información pasen de ser una opción empresarial costosa a una necesidad estratégica que determina el nivel de competitividad en el mercado. Asimismo, la importancia que día a día ha adquirido la información contable y financiera dentro de las organizaciones se debe a tres aspectos:

1) La necesidad de las empresas de adaptarse a la realidad cambiante y compleja;

2) La información como un elemento a incluir en las actividades económicas y sociales de la empresa; y

3) La nueva sociedad del conocimiento basada en la revolución de las TIC y su impacto potencial en la eficacia y eficiencia del procesamiento de la información.

Las TIC han constituido el principio fundamental para la mecanización de las tareas. Tanta ha sido su integración, que ya no es posible observar de manera separada estos dos componentes. La utilización de las TIC, ha cambiado la forma en la que se registra y se llevan a cabo las tareas operativas mediante los

instrumentos y programas, caracterizándolos por ser sistemas de información donde la disponibilidad de los datos es oportuna y veraz para la toma de decisiones. (Bermeo-Giraldo, y otros, 2019).

Como parte del desarrollo de las TIC, surgen las aplicaciones móviles, las cuales constituyen también un medio de innovación tecnológica, ya que promueven y facilitan la invención y la producción de nuevos servicios, productos o procesos en el ámbito gerencial. Es importante entender el impacto que estos cambios han generado en la gerencia de nuestras sociedades, comprendiendo que los beneficios asociados a la difusión de una tecnología de uso general van más allá de su aplicación a los procesos de negocio y permiten generar mejoras en la calidad y eficiencia de los procesos gerenciales dentro de las organizaciones. Al igual que con los teléfonos fijos, la difusión de la telefonía móvil y sus nuevas aplicaciones digitales implica cambios en la organización cotidiana de la vida privada, lo que a su vez repercute en las actividades laborales tomando en cuenta que ninguna organización funciona como un ente aislado de su contexto. Ya se trate de empresas grandes o pequeñas, de empresas formales o informales, desde un punto de vista puramente económico, podemos identificar una serie de áreas en las que la presencia de aplicaciones móviles está impulsando cambios.

Según *The Computer Language Company Inc. (2016) PC Magazine*, las aplicaciones móviles pueden definirse como: "…una aplicación de software que se ejecuta en un teléfono inteligente, tableta u otro dispositivo portátil. En contraste con las aplicaciones de escritorio. Se dividen en: aplicación móvil nativa, versión móvil y tienda de aplicaciones en línea". (El Impacto de las Aplicaciones Móviles en la Gestión Empresarial en Latinoamérica, 2017).

En Cuba, se han venido implementando nuevas medidas para el beneficio de sus trabajadores luego del paso de la pandemia, los cuales, a consecuencia de la escasez de diversos elementos en el mercado formal, reciben periódicamente productos mediante el Sindicato, órgano que se dio a la tarea de distribuirlos entre todos de manera equitativa, como sucede en la Universidad de las Ciencias Informáticas y sus diferentes centros de desarrollo. El secretario de la organización en el centro es el encargado de mantener el control de la gestión de las asignaciones de los productos y además se encarga de gestionar un fondo por cada trabajador para tener disponibilidad de efectivo en el momento que estos se reciban, todo este proceso se organiza de forma manual utilizando un fichero Excel. Los productos son escasos, variados y no llegan con regularidad, por lo que, la asignación de productos se realiza mediante una rotación por los trabajadores. El Centro de Software Libre se ha planteado automatizar el proceso de asignar productos a los trabajadores

de manera que le permita a su secretario del sindicato manejar la información al instante y en tiempo real de los productos, trabajadores y los fondos de cada uno de ellos. El contexto antes descrito origina el siguiente **problema a resolver**: ¿Cómo agilizar y automatizar el proceso de la gestión de las asignaciones de productos del sindicato a trabajadores del Centro de Software Libre?

Se propone como **objeto de estudio**: el proceso de gestión de la información.

El **campo de acción** de este trabajo está definido como: aplicaciones móviles para la gestión de información de las asignaciones de productos a los trabajadores de CESOL.

El **objetivo general** es desarrollar una aplicación móvil para la gestión y control de la información correspondiente a la asignación de productos para los trabajadores de CESOL, que facilite la toma de decisiones en tiempo real. Para ello se definieron un conjunto de **tareas de investigación**, las cuales son:

1. Elaboración del marco teórico referencial a partir del estudio de diferentes aplicaciones móviles que existen, así como las tecnologías y herramientas que se emplean en su desarrollo.
2. Definición de las tecnologías, las herramientas y la metodología para la implementación de la aplicación móvil.
3. Generación de los artefactos que corresponden a las etapas ingenieriles definidas por la metodología seleccionada.
4. Implementación de las funcionalidades de la propuesta de solución.
5. Validación de las funcionalidades de la aplicación móvil.

Para cumplimentar estas tareas se han empleado métodos empíricos y teóricos de la investigación científica. Los **métodos empíricos** que se utilizan para poder recopilar información fueron:

- La **observación**, este método permitió observar cómo se realiza la gestión de la información del proceso de asignación de productos a trabajadores del centro; y para profundizar sobre el campo de acción a partir de la investigación realizada sobre las herramientas y tecnologías.
- La **entrevista** permitió recopilar información necesaria para valorar la situación actual del problema lo cual permitió realizar un análisis y determinar los principales requisitos del sistema. (Ver anexos)

Como **métodos teóricos** se emplearon:

Se hace uso del método **analítico-sintético** para el análisis, evaluación y selección de las técnicas a emplear en el desarrollo de la aplicación para la asignación de productos a trabajadores del centro. Se emplea para sintetizar la información que se obtuvo mediante el intercambio con los clientes para que pueda ser utilizada en la construcción del sistema, además, en la identificación de los elementos del marco teórico de la investigación.

El **histórico-lógico**, se emplea para identificar las tendencias actuales de los sistemas de gestión de información para la asignación de productos y posteriormente su evolución de estos dentro de los sistemas de gestión de información, además en la recopilación de información para el estudio de sistemas homólogos.

La información de este trabajo se encuentra estructurada de la siguiente forma:

Capítulo 1: En este capítulo se tratará la fundamentación teórica (del objeto de estudio y el campo de acción), se realizará el análisis de sistemas homólogos, se reflejarán las metodologías, lenguaje y tecnologías usados para el desarrollo de la herramienta.

Capítulo 2: En este capítulo se tratará lo referente a la fase de planificación y diseño de la aplicación móvil en cuestión, a partir de sus requisitos, arquitectura y patrones de diseño.

Capítulo 3: En este capítulo se describirán las fases de implementación y pruebas del sistema. Se proporcionarán detalles de las tareas de ingeniería por historia de usuario a partir de las iteraciones definidas y se dará paso a la fase de pruebas que permitan comprobar el funcionamiento y la aceptación de la herramienta desarrollada.

Para finalizar se presentan las conclusiones, las recomendaciones, las referencias bibliográficas utilizadas y los anexos para un mejor entendimiento de lo expuesto a lo largo de este trabajo.

CAPÍTULO 1: Fundamentación teórica de la aplicación

En este capítulo abarcará en gran medida la información referente a las aplicaciones móviles: tipos, características; y se realiza un análisis de aplicaciones similares a la que se tiene como objetivo y las tecnologías que se utilizan para su desarrollo.

1.1 Tipos de aplicaciones móviles y características

Durante el desarrollo de este epígrafe se plantea lo referente a los diferentes tipos de aplicaciones móviles y sus características, para luego arribar a conclusiones sobre el tipo de aplicación a desarrollar.

1.1.1 Las aplicaciones móviles nativas

Estas aplicaciones son aquellas que se desarrollan para un sistema operativo específico, principalmente Android o iOS ya que son los más conocidos y utilizados en los dispositivos móviles mundialmente. Se llaman aplicaciones nativas debido a que se desarrollan para el sistema operativo nativo de cada dispositivo. Este tipo de aplicaciones móviles son descargadas desde las tiendas de aplicaciones como pueden ser Play Store (Android) y App Store (iOS). Se desarrollan tantas aplicaciones como sistemas operativos sean en los que se van a instalar dichas aplicaciones. Lo más habitual es crear dos aplicaciones, una para Android y otra para iOS pero que a nivel de diseño, funcionalidades y experiencia de usuario sean iguales. De esta forma, se consigue crear aplicaciones nativas totalmente adaptadas a cada sistema operativo y también a los dispositivos, ofreciendo así una experiencia más completa y mejorada a los usuarios. Aunque suelen ser las aplicaciones a las que más presupuesto se tiene que dedicar debido al mayor trabajo que conlleva, las aplicaciones nativas son también las más desarrolladas gracias a su gran rendimiento.

Características principales de las aplicaciones nativas:

- Las aplicaciones nativas no necesitan conexión a internet para que funcionen.
- La descarga e instalación de estas aplicaciones se realiza siempre a través de las tiendas de aplicaciones (app store de los fabricantes).
- Pueden hacer uso de las notificaciones del sistema operativo para mostrar avisos importantes al usuario, aun cuando no se esté usando la aplicación.
- No requieren Internet para funcionar, por lo que ofrecen una experiencia de uso más fluida y están realmente integradas al teléfono, lo cual les permite utilizar todas las características de hardware del

terminal, como la cámara y los sensores (sistema de posicionamiento GPS, acelerómetro, giróscopo, entre otros). (Mendoza, 2015) (Martínez, 2017)

Esta clase de aplicaciones tiene una interfaz basada en las guías de cada sistema operativo, logrando mayor coherencia y consistencia con el resto de aplicaciones y con el propio Sistema Operativo. Esto favorece la usabilidad y beneficia directamente al usuario que encuentra interfaces familiares. (Mendoza, 2015)

1.1.2 Las aplicaciones híbridas.

Una aplicación híbrida se basa en el desarrollo de una página móvil con capacidad para manejar los elementos nativos del dispositivo (cámara y GPS, entre otros). Para esto se utiliza la nueva versión del lenguaje HTML conocida como HTML5, que está siendo utilizada de manera creciente por los equipos de desarrollo (Martínez, 2017).

Como ejemplo de este tipo de aplicaciones se pueden señalar las tan conocidas aplicaciones: Facebook, Evernote, Instagram, Twitter, Uber, entre otras. Podemos ver que estas aplicaciones poseen como ventajas el bajo costo de la inversión en cuanto a tiempo y recursos, debido a que el desarrollo es único y de un solo maquetado para las diferentes plataformas, de manera que se obtendrá mucho antes un producto y solamente habría que mantener una sola fuente de código y son aplicaciones multiplataforma. En cambio, tenemos como desventajas el hecho de lo tedioso que resulta ofrecer una buena experiencia de usuario. También tenemos que la lógica de la aplicación estará hecha en JavaScript, esto hará que tengamos mucho código en este lenguaje si la aplicación es compleja. Con respecto a las aplicaciones nativas, estas son más lentas y las integraciones con funciones del dispositivo son más fluidas en las aplicaciones nativas. La documentación en las aplicaciones híbridas puede ser un poco escasa y desordenada y su diseño visual no siempre está relacionado con el sistema operativo en el que se muestre (Angulo, 2013).

1.2 Análisis de las aplicaciones similares

Como parte de los avances de la tecnología de la información y la necesidad de automatizar los diferentes procesos para el manejo de datos o información, se van creando aplicaciones con el objetivo de satisfacer a esas demandas y poder brindar un servicio de calidad de acuerdo al campo que abarquen las mismas. En este apartado se trata de analizar algunas aplicaciones similares para así poder poner en evidencia los diferentes puntos en común. Se muestra un recorrido por las diferentes aplicaciones que existen tanto en el ámbito nacional como internacional para luego llegar a mencionar algunas de las aplicaciones desarrolladas por cubanos.

En primer lugar, la aplicación **Stock and Inventory Management System** (Sistema de Gestión de Inventario y Existencias). Es una aplicación que administra y rastrea los inventarios de productos y verifica el inventario. Esta aplicación administra el producto agregando detalles como nombre, ID del producto, tasa de compra y descripción del producto, también gestiona transacciones de productos: entrada (importación) / salida (exportación). Muestra productos con un inventario bajo según el límite de producto bajo establecido en la configuración. La lista de productos con existencias limitadas lo ayuda a decidir qué comprar para verificar el inventario. Entre las características de esta aplicación se muestran las siguientes:

- Una aplicación gratuita, simple y compacta que gestiona el stock y el inventario de productos.
- Administrar los detalles del producto agregando, actualizando y eliminando detalles del producto.
- Administrar fácilmente las transacciones de importación y exportación de productos.
- Mostrar un resumen de las importaciones, exportaciones y existencias disponibles de cada producto.
- Mostrar la lista de productos con poco stock en función de establecer un valor de alerta de stock bajo.
- Proporcionar QR y escáner de código de barras para leer el código del producto.
- Mostrar el informe del gráfico circular del producto entrante (importación), el producto saliente (exportación) y el stock disponible que le permite analizar fácilmente el stock.
- Proporcionar funciones de copia de seguridad y restauración de datos de almacén.
- Las utilidades de búsqueda y filtro están disponibles para buscar fácilmente productos y transacciones de filtro.
- Exportar informes sobre detalles de productos y transacciones a Excel o PDF. Estos informes se pueden abrir, compartir y eliminar (Laboratorio Magnético, 2020).

Otra aplicación que posibilita el mantener al día el inventario es **Stock e Inventario Simple**. Donde se podrán gestionar todos los productos y servicios de forma rápida y sencilla, ya sea un pequeño almacén, tienda o negocio comercial. Este sistema permite exportar sus datos a Excel o Google Drive para posterior uso en sistemas de reportes. Las características generales de la aplicación mencionada son:

- Fácil ingreso de datos, sea de forma manual o importando un archivo de Excel.
- Permitir el manejo de imágenes por cada producto.
- Organizar los productos en carpetas y grupos de forma ilimitada.

- Escanear códigos de barra para mayor rapidez a la hora de cargar productos nuevos con la cámara de tu teléfono.
- Hacer seguimiento de clientes y proveedores, así como de sucursales.
- Notificar cantidades mínimas para fácil reposición de mercancía en tiempo real.
- Generar reportes de ingresos, márgenes y marcadores.
- Seguir ventas diarias por producto o por clientes.
- Exporta e importa archivos en Excel.
- Usa Google Drive para intercambiar datos y generar respaldos (Chester SW, 2022).

Otro ejemplo a mostrar es la aplicación **AlierSGA.** El SGA para almacén de Aliernet funciona con la tecnología de Microsoft. Enlaza con fluidez con los más populares ERPs (Enterprise Resource Planning) del mercado y con agencias de transporte. Permite el completo control de stock y la trazabilidad durante toda la cadena de suministro. Es un programa de gestión de almacenes muy completo. Fácil de usar y muy visual, agiliza todas las operaciones del almacén. Entre las funcionalidades destacadas de este sistema están las siguientes:

- Hace posible la gestión multialmacén y multicliente.
- Gestión de stock por clientes, artículos, referencias.
- Dispone de inventarios generados según diferentes filtros.
- Registra artículos y referencias: lotes, colores, modelos, fechas de caducidad, números de serie.
- Habilita la ubicación según reglas estándares LIFO, FIFO Y FEFO.
- Integra radiofrecuencia, y puestos de trabajo en mesa.
- Permite entrada de datos manual o por vía de comunicación electrónica.
- Gestión de picking. Es el proceso por el cuál la mercancía se extrae de las unidades de embalaje y se prepara dando como resultado un pedido individual. En muchas ocasiones estos pedidos están formados por la combinación de distintas mercancías, cada una de las cuales se recoge de una ubicación distinta.
- Organiza los flujos de trabajo.
- Genera inventarios y recuentos cíclicos.
- Automatiza la facturación según acuerdos establecidos.

En nuestro país, se pueden encontrar diversas aplicaciones destinadas a gestionar información y brindarle un servicio de calidad al usuario. Recientemente existen desarrolladores como (Madruga, 2022), que ha puesto a disposición del pueblo cubano aplicaciones como **Habana Servicentro**. Esta es una aplicación colaborativa que muestra la existencia de combustible en tiempo real a partir de la información de los mismos choferes, realiza búsquedas rápidas por tipo de combustible y por el nombre del servicentro. Contiene una calculadora en la nube con los precios predeterminados y actualizados en tiempo real, con la posibilidad de crear un historial de consumo de dinero y combustible por dos meses.

Otra de las aplicaciones desarrolladas en nuestro país es **Almacena tu Negocio**, disponible en la modalidad de pago en la tienda de aplicaciones cubana Apklis (Rivero, 2021). Entre las características de la aplicación se puede resumir que es dedicada a la administración de negocios particulares de venta de productos (dígase ropa, alimentos, electrodomésticos, etc.). En ella se permite llevar a cabo anotaciones de las operaciones realizadas con cada uno de esos productos haciendo más cómodo el proceso de manejar el negocio.

A lo largo de los años se han desarrollado numerosas herramientas de gestión de la información a nivel mundial, y en nuestro país en particular, teniendo en cuenta las diferentes necesidades que pueden presentar sus usuarios en la vida cotidiana, haciendo tan necesarias sus implementaciones en su momento. Luego del estudio realizado se puede concluir que los sistemas presentados anteriormente en su mayoría son o cuentan con funcionalidades de pago, además, no cumplen con los requisitos funcionales de los clientes del centro, debido a que cada una en particular cumple con sus propias necesidades específicas de gestión de la información.

1.3 Análisis de las tecnologías y metodologías para el desarrollo de aplicaciones móviles

La idea fundamental del presente trabajo de diploma, es elaborar un marco teórico-investigativo sobre el desarrollo de una aplicación móvil que permita la gestión de información para automatizar procesos internos dentro del Centro de Software Libre de la Universidad de las Ciencias Informáticas, para ello se utilizan varias tecnologías que ayudan a facilitar el mismo y condicionar el trabajo a realizar. En el siguiente epígrafe se caracterizarán dichas tecnologías para su posterior uso en el desarrollo.

1.3.1 Tecnologías

Android es un sistema operativo móvil basado en el núcleo Linux y otros softwares de código abierto. Fue diseñado para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tabletas, relojes

inteligentes Wear OS, automóviles con otros sistemas a través de Android Auto, al igual los automóviles con el sistema Android Automotive y televisores Android TV. Inicialmente fue desarrollado por Android Inc., que fue adquirido por Google en 2005. Android fue presentado en 2007 junto con la fundación del Open Handset Alliance (un consorcio de compañías de hardware, software y telecomunicaciones) para avanzar en los estándares abiertos de los dispositivos móviles. El código fuente principal de Android se conoce como Android Open Source Project (AOSP por sus siglas en inglés), que se licencia principalmente bajo la Licencia Apache. Android es el sistema operativo móvil más utilizado del mundo, con una cuota de mercado superior al 90 % al año 2018, muy por encima de IOS.

Un entorno de desarrollo integrado (IDE) es un sistema de software para el diseño de aplicaciones que combina herramientas comunes para desarrolladores en una sola interfaz de usuario gráfica (GUI). Generalmente, un IDE cuenta con las siguientes características:

- Editor de código fuente: editor de texto que ayuda a escribir el código de software con funciones como el resaltado de la sintaxis con indicaciones visuales, el relleno automático específico para el lenguaje y la comprobación de errores a medida que se escribe el código.
- Automatización de compilaciones locales: herramientas que automatizan tareas sencillas y repetitivas como parte de la creación de una compilación local del software para su uso por parte del desarrollador, como la compilación del código fuente de la computadora en un código binario, el empaquetado de ese código y la ejecución de pruebas automatizadas.
- Depurador: programa que sirve para probar otros programas y mostrar la ubicación de un error en el código original de forma gráfica. (Aliernet, 2022)

Durante el desarrollo de la aplicación, se utiliza el IDE Android Studio, el cual lleva muchos años siendo pionero en el desarrollo móvil, y brinda facilidades para codificar en los lenguajes Java, C++ y Kotlin, siendo este último el que se utiliza en el desarrollo de la aplicación que corresponde a este trabajo. Kotlin es un lenguaje de programación de código abierto creado por JetBrains que se ha popularizado gracias a que se puede utilizar para programar aplicaciones Android.

Como ocurre con la mayoría de entornos de desarrollo modernos, estos ofrecen las herramientas necesarias en la generación del código, lo que se denomina la lógica de la aplicación. Pero también los mecanismos con los que diseñar la interfaz de usuario que lucirá el desarrollo final. El entorno permite y guía al creador

para que pueda desde realizar sus primeras pruebas de código hasta la publicación del desarrollo final en Play Store, la tienda de aplicaciones de Google, y Apklis, la tienda online de aplicaciones desarrollada por la UCI para dispositivos móviles. Para realizar el proceso no faltan elementos necesarios como el compilador, el analizador de archivos APK o un emulador. Al poder desarrollar aplicaciones para el sistema operativo móvil de Google, por extensión también se confeccionan para ChromeOS, por lo que se tiene en cuenta esta opción. También los diferentes destinos que puede tener, más allá del teléfono: relojes, tablets, coche. Entre las ventajas de este IDE, se puede encontrar un emulador de Android, con la capacidad de probar las aplicaciones en desarrollo en forma de simulador, de modo que parezcan instaladas. Tiene la capacidad de reutilizar código abierto de otros creadores para su uso en las aplicaciones que se estén desarrollando. Por último, cuenta con la capacidad de realizar una depuración USB para la prueba de las aplicaciones en teléfonos reales, para así disfrutar de la mejor experiencia de prueba de una aplicación móvil.

El IDE estará acompañado por otras herramientas, una de ellas es el Gradle que permite la automatización de compilación de código abierto, la cual se encuentra centrada en la flexibilidad y el rendimiento. Los scripts de compilación de Gradle se escriben utilizando Groovy o Kotlin DSL (Domain Specific Language). En el presente trabajo se hace uso de su versión 7.4.2.

También Se utiliza el SDK, es el acrónimo de “Software Development Kit” (Kit de desarrollo de software). El SDK reúne un grupo de herramientas que permiten la programación de aplicaciones móviles. Este conjunto de herramientas se puede dividir en 3 categorías:

- SDK para entornos de programación o sistemas operativos (iOS, Android, etc.)
- SDK para el mantenimiento de aplicaciones
- SDK de marketing y publicidad

1.3.2 Metodología

El desarrollo de aplicaciones móviles tiene las mismas complicaciones que el desarrollo de una página web. Los desarrollos de software sufren también problemas. Una de las cualidades que tiene es que la duración es más corta y hay mucha competencia en el sector. De esta manera, obliga a que esté constantemente innovando e implementando cambios en la plataforma de desarrollo. Todo ello influye a la hora de elegir

una metodología concreta de desarrollo. Existen diferentes tipos de metodologías para el desarrollo de aplicaciones móviles:

a) Metodología ágil: La metodología ágil se basa en tener como principal característica la realización entregas rápidas y continuas de software. Por ejemplo, en el marco de trabajo Scrum, el proyecto se divide en pequeñas partes que tienen que completarse y entregarse en plazos cortos, llamados 'sprints'. Así, en caso de tener que modificar alguna cosa, solo se cambia la parte que queremos cambiar. Esta metodología surge en la industria del desarrollo de software.

b) Modelo Waterfall o en cascada: Es uno de los modelos de metodologías para el desarrollo de aplicaciones móviles clásico. Esta aplicación sólo es adaptable cuando están totalmente cerrados los requisitos y no van a cambiar. No hay retroalimentación entre las fases en que se divide el proyecto. Por lo que cada fase se va cerrando de forma secuencial. Todo el proceso está fijado por fechas límites y presupuestos. Este modelo sólo es aconsejable para proyectos móviles muy controlados y previsibles, no existe incertidumbre por lo que se quiere hacer ni influyen los cambios en la industria.

c) Desarrollo rápido de aplicaciones: Esta metodología da énfasis en la obtención de un prototipo funcional de una aplicación para posteriormente ir mejorándolo. Después va incluyendo más funcionalidades y complejidad. Es recomendable el uso de patrones de diseño bien conocidos para adaptarse a los cambios de requisitos. Dentro de las metodologías para el desarrollo de aplicaciones móviles se suele usar cuando los plazos de entrega son muy cortos. Se precisa tener un entregable de forma inmediata. No se descarta utilizar otras metodologías de forma posterior, debido a que este tipo de desarrollo puede ser usado para mostrar un esbozo de la aplicación a un cliente, generalmente en un par de días.

d) Mobile-D: El objetivo de esta metodología es conseguir ciclos de desarrollo muy rápidos en equipos muy pequeños. Se basa en metodologías para el desarrollo de aplicaciones móviles conocidas pero aplicadas de forma estricta como: Extreme Programming, Crystal Methodologies y Rational Unified Process.

La Universidad de las Ciencias Informáticas (UCI) desarrolló una versión de la metodología de desarrollo de software AUP (Proceso Ágil Unificado), con el fin de crear una metodología que se adapte al ciclo de vida definido por la actividad productiva de la universidad. Esta versión decide mantener para el ciclo de vida de los proyectos la fase de Inicio, pero modificando el objetivo de la misma y se unifican las restantes fases de la metodología de desarrollo de software AUP en una sola, nombrada Ejecución y agregándose también una nueva fase llamada Cierre. A continuación, se muestran las diferentes fases de la metodología y sus objetivos:

a) Fase de inicio: Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
b) Ejecución (Elaboración, construcción, transición): En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.
c) Cierre: En esta fase se analizan tanto los resultados del proyecto como la ejecución y se realizan las actividades formales de cierre del proyecto.

La metodología de software AUP-UCI a partir de que el modelado de negocio propone tres variantes a utilizar en los proyectos, como son: CUN (Casos de uso del negocio), DPN (Descripción de proceso de negocio) o MC (Modelo conceptual) y existen tres formas de encapsular los requisitos los cuales son: CUS (Casos de uso del sistema), HU (Historias de usuario), DRP (Descripción de requisitos por proceso), surgen cuatro escenarios para modelar el sistema en los proyectos, los cuales son:

- Escenario No 1: Proyectos que modelen el negocio con CUN solo pueden modelar el sistema con CUS.
- Escenario No 2: Proyectos que modelen el negocio con MC solo pueden modelar el sistema con CUS.
- Escenario No 3: Proyectos que modelen el negocio con DPN solo pueden modelar el sistema con DRP.
- Escenario No 4: Proyectos que no modelen negocio solo pueden modelar el sistema con HU

A partir del análisis e investigación realizada, se utiliza la variante de Historias de usuario y el escenario #4 de la metodología AUP-UCI, porque dicha metodología es apropiada para proyectos como el que se va a desarrollar durante este trabajo de diploma.

Tras el estudio del capítulo y haber realizado un análisis de los problemas en cuestión, se puede concluir, luego de haber investigado sobre las aplicaciones para la gestión de información, que el Centro de Software Libre necesita de una herramienta que le permita automatizar los procesos manuales que actualmente realizan con el manejo de los diferentes datos sobre la gestión de los productos que le asignan a sus trabajadores, elaborando para ello una aplicación móvil para la gestión de la información que lo permita y posibilite dicho proceso. Además, fueron seleccionadas las herramientas y metodología a utilizar en el proceso de desarrollo para dar comienzo a la realización de dicho sistema.

CAPÍTULO 2: Análisis y diseño de la aplicación móvil

En el actual capítulo se describe el procedimiento y los elementos que se tuvieron en cuenta en el desarrollo de la aplicación para la gestión de productos asignados para los trabajadores del Centro de Software Libre. Se presentan los conceptos asociados al modelo conceptual y la representación formal del mismo. Se identifican los requisitos no funcionales y funcionales. Además, se emplea la técnica de descripción de requisitos, acorde con la metodología empleada.

2.1 Modelado de la aplicación

Como propuesta de aplicación se trazó el desarrollo de una aplicación móvil que brinde opciones al usuario de gestionar mediante varias interfaces el comportamiento de las asignaciones de los productos, así como el control del personal, sus hijos y de los productos asignados; también permite filtrar diferentes búsquedas dentro de las diferentes listas de entidades. El secretario del Sindicato podrá interactuar con la aplicación mediante botones y formularios. A continuación, se muestra el flujo del evento para insertar un nuevo producto.

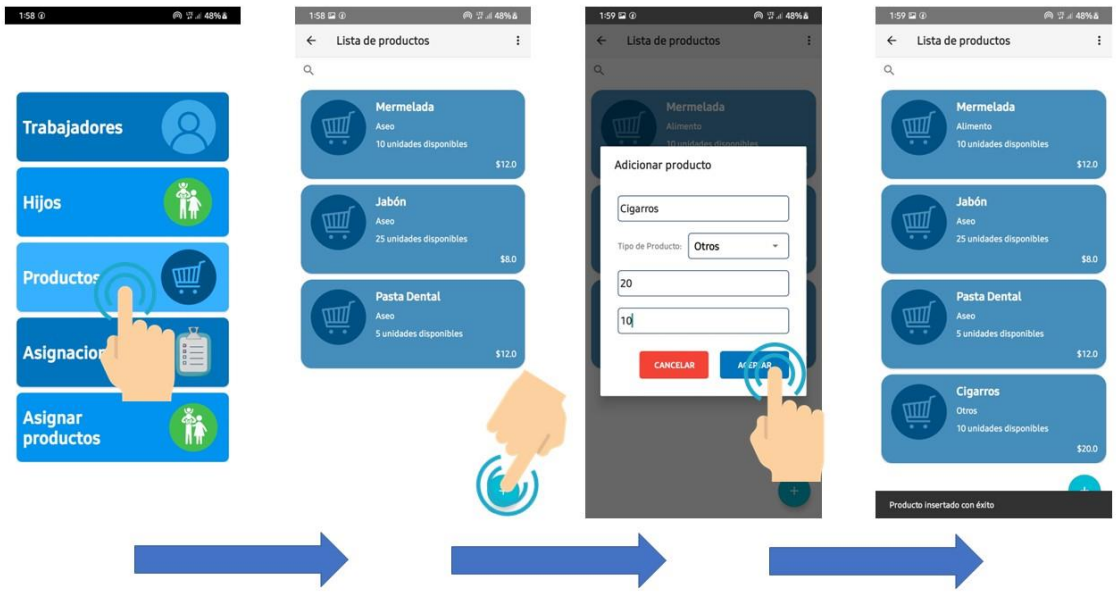


Figura 1: Insertar nuevo producto (Representación). Fuente: Elaboración propia

El proceso de gestión parcial de trabajadores y productos inicia una vez que el secretario del Sindicato del Centro obtiene la información de los nuevos trabajadores o productos que deben ser añadidos, algún grupo debe ser modificado o un determinado trabajador debe ser eliminado. Cuando se añade un nuevo registro de un trabajador se le asignan un conjunto de características que lo diferencian de otros para la asignación de los productos por el secretario.

El proceso de gestión parcial de los hijos parte de los trabajadores que han sido previamente insertados, permitiendo asignar uno o varios hijos a cada uno de los mismos y luego en caso de ser necesario editarlos.

Las asignaciones son tratadas mediante la vista de generación de reportes (AsignarActivity), en la cual se muestra una lista de trabajadores con el total de las asignaciones realizadas a cada uno. El secretario puede modificar esta lista en base a varios criterios en los cuales se puede auxiliar para generar una nueva lista en orden de prioridades y de forma particular teniendo en cuenta el producto a asignar. Luego puede realizar la asignación del producto de acuerdo a su decisión

Tabla 1: Personas que intervienen en el sistema. Fuente: Elaboración propia

Nombre	**Descripción**
Secretario del Sindicato	Encargado de gestionar los trabajadores y productos.

2.2 Requisitos de la aplicación de gestión de trabajadores y productos

Para solucionar la problemática planteada por el Centro de Software Libre para con la entrega de los módulos destinados a los trabajadores del mismo, se ha propuesto realizar una aplicación como herramienta para facilitar el trabajo a la persona encargada de realizar esta tarea. Contando con una base de datos externa en donde se almacenan los datos de todos los trabajadores, productos y asignaciones. Cumpliendo con las exigencias del cliente, se determinan a continuación los requisitos funcionales y no funcionales de la aplicación móvil a desarrollar.

2.2.1 Requisitos funcionales

Los requisitos funcionales de un sistema describen lo que éste debe hacer. Estos requisitos dependen del tipo de software que se desarrolle, de los posibles usuarios del software y del enfoque general tomado por

la organización al redactar requerimientos. Cuando se expresan como requerimientos del usuario, habitualmente se describen de una forma bastante abstracta. Sin embargo, los requisitos funcionales del sistema describen con detalle la función de éste, sus entradas, salidas, excepciones, etc. (Sommerville, 2007)

En el análisis, se identifican los siguientes requisitos funcionales:

Tabla 2: Requisitos funcionales. Fuente: Elaboración propia

Número de requisito	**Descripción**	**Prioridad**
Requisito Funcional # 1	Insertar trabajador.	Alta
Requisito Funcional # 2	Modificar trabajador.	Alta
Requisito Funcional # 3	Listar trabajadores.	Alta
Requisito Funcional # 4	Insertar producto.	Alta
Requisito Funcional # 5	Modificar producto.	Alta
Requisito Funcional # 6	Listar productos.	Alta
Requisito Funcional # 7	Insertar asignación.	Alta
Requisito Funcional # 8	Modificar asignación.	Alta
Requisito Funcional # 9	Listar asignaciones.	Alta
Requisito Funcional # 10	Crear hijo de trabajador.	Alta
Requisito Funcional # 11	Modificar hijo de trabajador.	Alta
Requisito Funcional # 12	Listar hijos de trabajador.	Alta
Requisito Funcional # 13	Crear base de datos.	Alta
Requisito Funcional # 14	Cargar base de datos.	Alta

Requisito Funcional # 15	Insertar varias asignaciones	Alta
Requisito Funcional # 16	De un producto específico, filtrar por mes o rango de fecha y mostrar las personas que no tienen asignado ese producto en dicho rango.	Media
Requisito Funcional # 17	De un trabajador, filtrar por mes o rango de fecha y mostrar el total de productos que tuvo asignado	Media
Requisito Funcional # 18	Filtrar por cantidad de niños y mostrar la cantidad de niños y los trabajadores y mostrar los datos.	Media
Requisito Funcional # 19	Filtrar asignaciones hacia un trabajador.	Media
Requisito Funcional # 20	Filtrar asignaciones en cuanto a un producto.	Media
Requisito Funcional # 21	Filtrar asignaciones dado una fecha o rango de fechas.	Media

Requisito Funcional # 22	Filtrar productos por nombre y mostrar los datos.	Media
Requisito Funcional # 23	Filtrar productos por precio y mostrar los datos.	Media
Requisito Funcional # 24	Filtrar productos por peso y mostrar los datos.	Media
Requisito Funcional # 25	Filtrar productos por tipo y mostrar los datos.	Media
Requisito Funcional # 26	Filtrar trabajadores por nombre y mostrar los datos.	Media
Requisito Funcional # 27	Filtrar trabajadores por cantidad de hijos y mostrar los datos.	Media
Requisito Funcional # 28	Filtrar trabajadores por género y mostrar los datos.	Media
Requisito Funcional # 29	Filtrar trabajadores en cuanto a si es fumador o no y mostrar los datos.	Media
Requisito Funcional # 30	Filtrar trabajadores activos y mostrar los datos.	Media
Requisito Funcional # 31	Mostrar trabajadores inactivos.	Media

Requisito Funcional # 32	Filtrar trabajadores en cuanto a residencia.	Media
Requisito Funcional # 33	Filtrar trabajadores en cuanto a fondo disponible.	Media
Requisito Funcional # 34	Filtrar hijos por nombre.	Media
Requisito Funcional # 35	Filtrar hijos por padre.	Media
Requisito Funcional # 36	Filtrar hijos por edad	Media

2.2.2 Requisitos no funcionales

Los requisitos no funcionales, son aquellos que no se refieren directamente a funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. De forma alternativa, definen las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y las representaciones de datos que se utilizan en las interfaces del sistema (Sommerville, 2007).

Se identifican mediante un análisis realizado, los siguientes requisitos no funcionales:

RNF1: La información brindada por el sistema, así como los mensajes interactivos deben ser lo más informativos posible, en español y no deben revelar información interna (Usabilidad).

RNF2: La base de datos es interna (Usabilidad).

RNF3: El tiempo de respuesta no debe exceder de 3 segundos (Rendimiento).

RNF4: Diseño sencillo, amigable e intuitivo, con color azul predominante y tamaño de letra adecuado (Interfaz).

RNF5: Se debe disponer de un teléfono móvil o tableta (Hardware).

RNF6: El dispositivo móvil debe contar con el sistema operativo Android con una versión de 5.0 Lollipop o superior (Software).

RNF7: Se utilizará el IDE Android Studio, el lenguaje de programación Kotlin y como gestor de base de datos SQLite3 (Software).

2.3 Historias de usuario y plan de iteraciones

2.3.1 Descripción de las historias de usuario

Uno de los artefactos generados por la metodología AUP-UCI son las historias de usuarios, que sustituyen a los documentos de especificación funcional, y a los casos de uso. Estas son escritas por el cliente, en su propio lenguaje, como descripciones cortas de lo que el sistema debe realizar. La diferencia más importante entre estas historias y los tradicionales documentos de especificación funcional se encuentra en el detalle mínimo para que los programadores puedan realizar una estimación poco riesgosa del tiempo que lleva su desarrollo (Joskowicz, 2008).

A continuación, se muestran dos de las historias de usuario identificadas en la aplicación (ver anexo 3):

Tabla 3: Representación de las historias de usuario. Fuente: Elaboración propia

Historia de usuario	
Numero: HU 2	**Nombre Historia de usuario:** Administrar parcialmente trabajador
Usuario: Lian Felipe Hernández Delgado	**Iteración asignada: 1**
Prioridad en negocio: Alta	**Puntos estimados:** 2
Riesgo en desarrollo: Media	**Puntos reales: 1**
Descripción: Permite insertar, modificar y listar los trabajadores	
Observaciones: Se guardan todos los datos de los trabajadores, permite modificarlos y listarlos.	

A continuación, y haciendo referencia a (Joskowicz, 2008), se describen las historias de usuario para una mejor comprensión de los datos brindados en las tablas anteriores.

Las historias de usuario se clasifican en cuanto a:

Prioridad en el negocio:

- **Alta**: se le otorga a las HU que resultan funcionalidades fundamentales en el desarrollo del sistema, a las que el cliente define como principales para el control integral del sistema.
- **Media**: se le otorga a las HU que resultan para el cliente como funcionalidades a tener en cuenta, sin que estas tengan una afectación sobre el sistema que se esté desarrollando.
- **Baja**: se le otorga a las HU que constituyen funcionalidades que sirven de ayuda al control de elementos asociados al equipo de desarrollo, a la estructura y no tienen nada que ver con el sistema en desarrollo.

El riesgo en su desarrollo:

- **Alta**: cuando en la implementación de las HU se considera la posible existencia de errores que conlleven a la inoperatividad del código.
- **Media**: cuando pueden aparecer errores en la implementación de la HU que puedan retrasar la entrega de la versión.
- **Baja**: cuando pueden aparecer errores que serán tratados con relativa facilidad sin que traigan perjuicios para el desarrollo del proyecto.

Son representadas mediante tablas divididas en las siguientes secciones:

- **Número**: esta sección representa el número, incremental en el tiempo, de la HU que se describe.
- **Nombre de HU**: identifica la HU que se describe entre los desarrolladores y el cliente.
- **Modificación de HU número**: sección que representa si la HU se le realizó alguna modificación con respecto al estado anterior.
- **Usuario**: los programadores responsables de la HU.
- **Iteración asignada**: número de la iteración donde va a desarrollarse la HU.
- **Prioridad en negocio**: se le otorga una prioridad (Alta, Media, Baja) a las HU de acuerdo a la necesidad de desarrollo.
- **Riesgo en desarrollo**: se le otorga una medida de (Alto, Medio, Bajo), a la ocurrencia de errores en el proceso de desarrollo de la HU.

- **Puntos estimados**: es el tiempo estimado en semanas que se demorará el desarrollo de la HU.
- **Puntos reales**: representa el tiempo que se demoró en realidad el desarrollo de la HU.
- **Descripción**: breve descripción de la HU.
- **Observaciones**: señalamiento o advertencia del sistema.

2.3.2 Plan de iteraciones

Al estar identificados los requisitos y descritas las historias de usuario, se procede a realizar un plan de estimación del esfuerzo que deberá requerir la implementación de cada una. Dicha planificación se muestra en la tabla a continuación que describe el plan de iteraciones de la siguiente forma:

Tabla 4: Plan de iteraciones. Fuente: Elaboración propia

Iteración	Id.	Historia de Usuario	Puntos de estimación	Duración en semanas
1	HU1	Administrar parcialmente base de datos	1	5
	HU2	Administrar parcialmente trabajadores	2	
	HU3	Administrar parcialmente productos	2	
2	HU4	Administrar parcialmente asignaciones	3	8
	HU5	Administrar parcialmente hijos de los trabajadores	2	
	HU6	Generar reportes	3	
3	HU7	Filtrar trabajadores	1	4

	HU8	Filtrar productos	1	
	HU9	Filtrar asignaciones	1	
	HU10	Filtrar hijos de los trabajadores	1	

2.4 Fase de diseño

Se describen a continuación las fases de diseño propias de la metodología de desarrollo, y se detalla además la arquitectura del sistema, así como los patrones de diseño utilizados en el desarrollo de la aplicación.

2.4.1 Arquitectura de software

El patrón Modelo-Vista-Modelo de Vista o en inglés Model-View-ViewModel (MVVM), está diseñado para plataformas de desarrollo de interfaz de usuario modernas donde la vista es responsabilidad de un diseñador en lugar de un desarrollador. MVVM se basa en un mecanismo general de enlace de datos que facilita el desarrollo de la capa de separación de vista desde el resto del patrón mediante la eliminación de todo el código subyacente de la capa de la vista.

- El modelo, encapsula la lógica de negocio y datos. La lógica empresarial se define como cualquier lógica de aplicación que se ocupa de la recuperación y gestión de datos de la aplicación y de asegurar que las reglas del negocio que garanticen la coherencia de los datos y la validez se cumplan. Para maximizar las oportunidades de reutilización, los modelos no deben contener ningún caso de uso o conducta específica del usuario o lógica de la aplicación.
- La vista, encapsula la interfaz de usuario y la lógica de la interfaz de usuario. Las vistas definen la interfaz de usuario específica para una parte de la aplicación, son normalmente los controles que se han diseñado para funcionar bien cuando se une a un ViewModel.
- El ViewModel, encapsula la lógica de presentación y estado. Lógica de presentación se define como la lógica de la aplicación que se ocupa de casos de uso de la aplicación (o casos de usuario, tareas de usuario, flujo de trabajo, etc.) y define el comportamiento lógico y la estructura de la aplicación. Para maximizar las oportunidades de reutilización, el ViewModel no debe tener ninguna referencia

a las clases específicas de interfaz de usuario, elementos, controles o comportamiento. (Arcos-Medina, y otros, 2018).

A continuación, se muestra una figura donde se representa un esquema con el patrón Model-View-ViewModel que se utiliza en la aplicación a desarrollar y su funcionamiento.

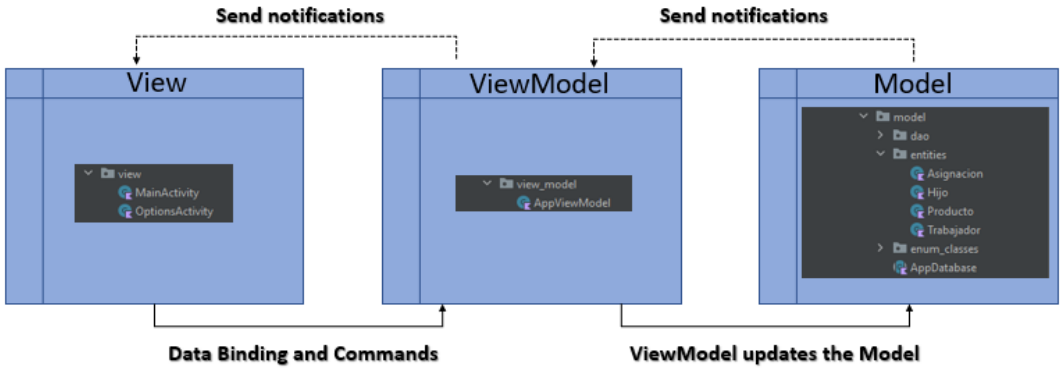


Figura 2: Representación de Model-View-ViewModel en la aplicación. Fuente: Elaboración propia

A continuación, se representa mediante el diagrama de paquetes agrupando elementos relacionados semánticamente y a su vez mostrando las dependencias entre esas agrupaciones la arquitectura de software MVVM:

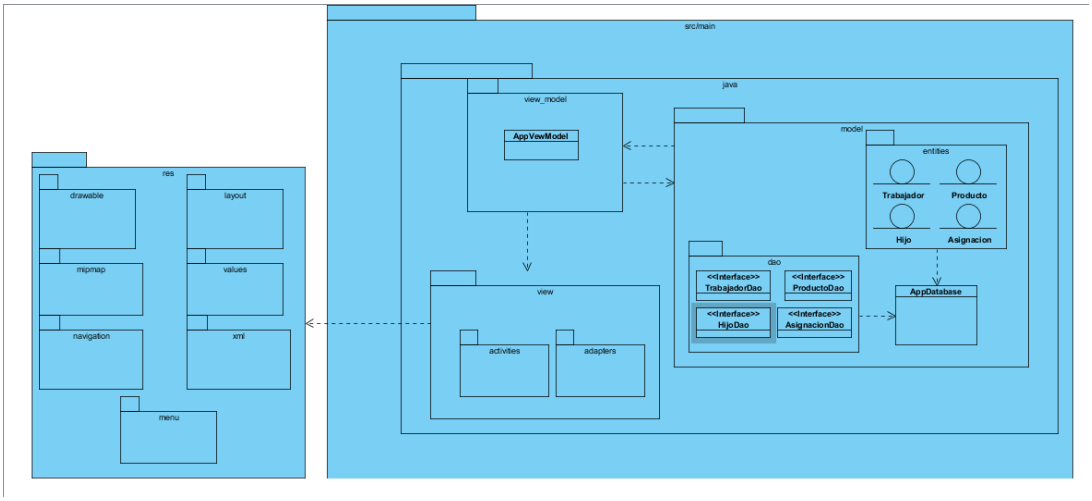


Figura 3: Diagrama de paquetes. Fuente: Elaboración propia

Los proyectos realizados en Android eventualmente van construyendo en los archivos .apk todo el código fuente, recursos y elementos necesarios para el funcionamiento del software. Estos son los ejecutables de instalación de las aplicaciones. Algunos son generados por el desarrollador por defecto, mientras que otros solo en caso de ser necesario.

2.4.2 Tarjetas CRC

El modelado CRC (Clase-Responsabilidad-Colaboración) proporciona una manera sencilla de identificación y organización de las clases que son relevantes para los requerimientos de un sistema o producto (Pressman, 2007).

El modelo CRC hace uso de tarjetas de índices reales o virtuales. El objetivo es desarrollar una representación organizada de las clases. Las responsabilidades son los atributos y operaciones relevantes para la clase. Los colaboradores son aquellas clases que se requieren para dar a una clase la información necesaria a fin de completar una responsabilidad, es decir, una colaboración implica una solicitud de información o de cierta acción (Pressman, 2007).

A continuación, se presenta las tarjetas CRC de la clase AppDatabase asociada al proyecto:

Tabla 5: Tarjeta CRC de la clase AsignacionActivity. Fuente: Elaboración propia

Nombre de la clase: AsignacionActivity	
Responsabilidades	**Clases relacionadas**
onCreate(…)	Asignacion
createDialogSearch()	AsignacionAdapter
onResume()	AppViewModel

2.4.3 Patrones de diseño

Un patrón de diseño se caracteriza como “una regla de tres partes que expresa una relación entre cierto contexto, un problema y una solución”. Para el diseño de software, el contexto permite al lector entender el ambiente en el que reside el problema y qué solución sería apropiada en dicho ambiente. Un conjunto de requerimientos, incluidas limitaciones y restricciones, actúan como sistema de fuerzas que influyen en la manera en la que puede interpretarse el problema en este contexto y en cómo podría aplicarse con eficacia la solución (Pressman, 2007).

Ventajas

- Proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifica y describe formas de solucionar problemas que ocurren de forma frecuente en el desarrollo.
- Están basados en la recopilación del conocimiento de los expertos en desarrollo de software.
- Es una experiencia real, probada y que funciona.
- Facilitan la localización de los objetos que formarán el sistema, la determinación de la granularidad adecuada, el aprendizaje y la comunicación entre programadores.
- Especifican interfaces para las clases e implementaciones al menos parciales.

Son denominados GRASP a los patrones generales de asignación de responsabilidades. Se basan en la determinación de las clases adecuadas y decidir cómo estas clases deben interactuar. Incluso cuando se utilizan metodologías rápidas como XP (eXtreme Programming) y el proceso se centra en el desarrollo continuo, es necesario elegir cuidadosamente las responsabilidades de cada clase desde la primera codificación y, fundamentalmente, en la refactorización del programa. Los patrones GRASP describen los

principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones (Carmona, 2012).

A continuación, se mencionan los patrones utilizados en la implementación de la aplicación:

Alta cohesión y bajo acoplamiento: La cohesión es la medida en la que un componente o clase realiza únicamente la tarea para la cual fue diseñada (Una clase debe hacer lo que respecta a su entidad, y no hacer acciones que involucren a otra clase o entidad). En este contexto, hablamos de alta cohesión cuando la relación es unívoca entre sí. Por el contrario, si hablamos de baja cohesión es cuando existe relación con otros componentes de otro tipo de naturaleza. Proporcionalmente, las buenas prácticas señalan que, a mayor cohesión, menos acoplamiento, esto último ocurre cuando existe una independencia entre los componentes entre sí, por el contrario, un alto acoplamiento es cuando tenemos varias dependencias relacionadas a un solo componente (Carmona, 2012). Este patrón se pone de manifiesto en las clases del sistema, que todas tienen asociadas funciones y componentes específicos, por ejemplo, la clase DatabaseModule, que es la encargada de crear la base de datos de la aplicación en caso de que no exista, además de las vistas, las cuales manejan únicamente sus componentes.

Controlador: Es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es el controlador quien recibe los datos del usuario y quien los envía a las distintas clases según el método llamado, sugiere que la lógica debe estar separada de la capa de presentación, ejemplo de MVC (Carmona, 2012). La clase AppViewModel es un ejemplo de este patrón.

Creador: El patrón creador nos ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. La nueva instancia deberá ser creada por la clase que tiene la información necesaria para realizar la creación del objeto, usa directamente las instancias creadas del objeto, almacena o maneja varias instancias de la clase, o contiene o agrega la clase (Carmona, 2012). Se pone de manifiesto en el hecho de que cada entidad establecida es la responsable de crear sus propios objetos llamados desde cualquier instancia.

Se ponen de manifiesto también en el desarrollo de la aplicación los patrones GoF (Gang of Four), los cuales están divididos en tres categorías: creacionales, estructurales y de comportamiento. A continuación, se presenta la relación de patrones utilizados, y su descripción de acuerdo al libro *Design Patterns. Elements of Reusable Object-Oriented Software*, (Gamma, y otros, 2008).

a) **Creational Design Patterns (Patrones de diseño creacionales)**, los que se ocupan de la creación de un objeto.

- **Singleton:** Este patrón restringe la inicialización de una clase para garantizar que solo se pueda crear una instancia de la clase. Este patrón se pone de manifiesto en la clase DatabaseModule, la cual crea una única instancia de creación y conexión a la base de datos, que le permite ser llamada desde cualquier clase que la necesite.

Figura 4: Singleton Design Pattern en la aplicación. Fuente: Elaboración propia

- **Builder (Constructor):** Crea un objeto paso a paso y un método para finalmente obtener la instancia del objeto. Ese patrón se pone de manifiesto en la construcción de las diferentes vistas en los adaptadores (adapters) y en las mismas vistas.

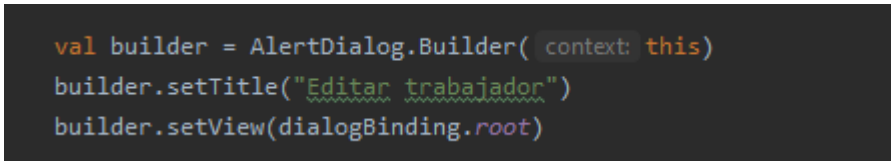
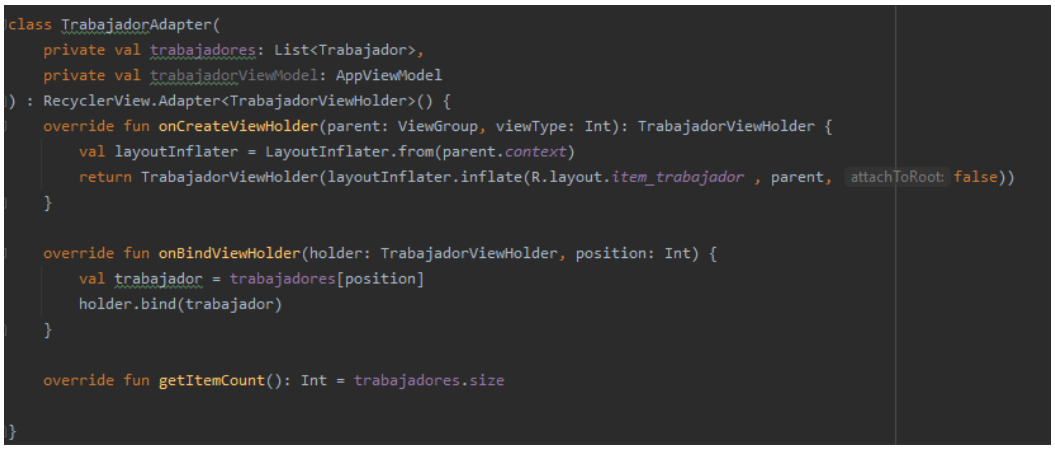


Figura 5: Fragmento de código de la clase DetallesTrabajadorActivity donde se muestra el uso de Builder. Fuente: Elaboración propia

b) **Structural Design Patterns (Patrones de Diseño Estructurales):** Los patrones de diseño en esta categoría se ocupan de la estructura de clases, como herencia y composición.

- **Adapter (Adaptador):** Proporciona una interfaz entre dos entidades no relacionadas para que puedan trabajar juntas. Por ejemplo, las entidades conectan con las vistas mediante estos adaptadores.

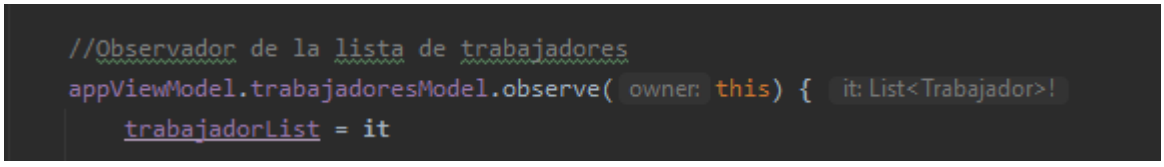


```
class TrabajadorAdapter(
    private val trabajadores: List<Trabajador>,
    private val trabajadorViewModel: AppViewModel
) : RecyclerView.Adapter<TrabajadorViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TrabajadorViewHolder {
        val layoutInflater = LayoutInflater.from(parent.context)
        return TrabajadorViewHolder(layoutInflater.inflate(R.layout.item_trabajador , parent, attachToRoot: false))
    }

    override fun onBindViewHolder(holder: TrabajadorViewHolder, position: Int) {
        val trabajador = trabajadores[position]
        holder.bind(trabajador)
    }

    override fun getItemCount(): Int = trabajadores.size

}
```

Figura 6: Ejemplo de Adaptador (TrabajadorAdapter). Fuente: Elaboración propia

c) **Behavioral Design Patterns (Patrones de Diseño de Comportamiento):** Este tipo de patrones de diseño brindan una solución para una mejor interacción entre los objetos, como proporcionar un acoplamiento perdido y flexibilidad para extenderse fácilmente en el futuro.

- **Observer (Observador):** Este patrón es útil cuando se está interesado en el estado de un objeto y desea recibir una notificación cada vez que haya algún cambio. Por ejemplo, en cada cambio que se realiza en las vistas al agregar o actualizar un nuevo elemento, hay presente un observador para comunicar los cambios y generar una nueva vista con los mismos.



```
//Observador de la lista de trabajadores
appViewModel.trabajadoresModel.observe( owner: this) {  it: List<Trabajador>!
    trabajadorList = it
```

Figura 7: Observador de la Lista de trabajadores en TrabajadorActivity. Fuente: Elaboración propia

También en el desarrollo de esta aplicación se encuentra presente el patrón de diseño orientado a objetos llamado **Inyección de dependencias,** en el que se suministran objetos a una clase en lugar de ser la propia clase la que cree dichos objetos. Esos objetos cumplen contratos que necesitan nuestras clases para poder funcionar (de ahí el concepto de dependencia). Nuestras clases no crean los objetos que necesitan, sino que se los suministra otra clase 'contenedora' que inyectará la implementación deseada a nuestro contrato. Se pone de manifiesto en cada una de las clases que hace referencia a otra en la creación de algún objeto tomando otra clase, como es el ejemplo de la clase AppViewModel y DatabaseModule.

```
@Module
@InstallIn(SingletonComponent::class)
object DatabaseModule {

    @Provides
    fun provideTrabajadorDao(appDatabase: AppDatabase):TrabajadorDao{
        return appDatabase.getTrabajadorDao()
    }
    @Provides
    fun provideHijoDao(appDatabase: AppDatabase):HijoDao{
        return appDatabase.getHijoDao()
    }
    @Provides
    fun provideProductoDao(appDatabase: AppDatabase):ProductoDao{
        return appDatabase.getProductoDao()
    }
    @Provides
    fun provideAsignacionDao(appDatabase: AppDatabase):AsignacionDao{
        return appDatabase.getAsignacionDao()
    }

    @Provides
    @Singleton
    fun provideDatabase(@ApplicationContext appcontext:Context):AppDatabase{
        return Room.databaseBuilder(appcontext,AppDatabase::class.java, name: "database").build()
    }
```

Figura 8: Ejemplo de Inyección de dependencias (clase DatabaseModule). Fuente: Elaboración propia

2.4.4 Modelo de datos

Un modelo de datos permite describir los elementos de la realidad que intervienen en un problema dado y la forma en que se relacionan esos elementos entre sí. Se puede definir como un conjunto de conceptos, reglas y convenciones bien definidos que permiten aplicar una serie de abstracciones a fin de describir y manipular los datos de un cierto mundo real que se desea almacenar en la base de datos. El modelo relacional se caracteriza a muy grandes rasgos por disponer que toda la información debe estar contenida

en tablas, y las relaciones entre datos deben ser representadas explícitamente en esos mismos datos (Sommerville, 2007).

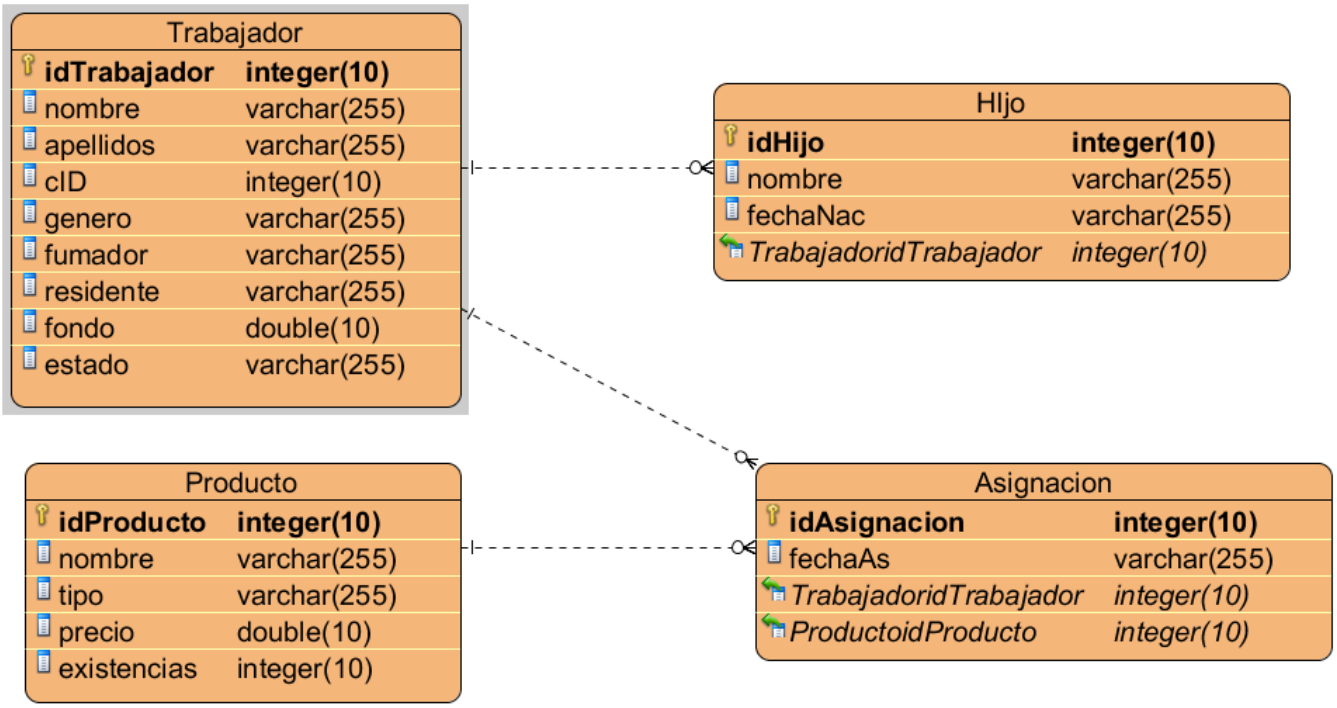


Figura 9: Modelo de datos de las entidades de la aplicación. Fuente: Elaboración propia

En el capítulo quedan plasmados los elementos esenciales para el correcto desarrollo de la solución a la problemática planteada. Se puntualizan los requisitos funcionales y no funcionales del sistema de acuerdo a las exigencias del actor para así realizar la correcta planificación de la implementación. Se selecciona la variante Modelo-Vista-Modelo de Vista (MVVM) como patrón de arquitectura de software. Quedan definidas además las diferentes historias de usuario para describir correctamente los aspectos principales a tener en cuenta a la hora del desarrollo, estableciendo sus prioridades, orden de implementación e iteraciones. Se obtienen las tarjetas CRC de las clases, se establecen los diferentes patrones de diseño a emplear y se modelan los datos de las entidades para con todo lo establecido llegar a desarrollar una herramienta de calidad.

CAPÍTULO 3: Implementación y pruebas de la aplicación

En el presente capítulo se describen las fases de implementación y prueba del sistema. Además, se realiza la descripción de las tareas de ingeniería por cada historia de usuario a partir de las iteraciones planificadas. Al finalizar esta etapa se continúa con la fase de prueba, el cual permite que se realicen las pruebas unitarias y pruebas de aceptación que permitan comprobar el correcto funcionamiento de la aplicación.

3.1 Fase de desarrollo

En la fase de desarrollo se realizarán las tareas relacionadas con la implementación de cada una de las historias de usuario correspondientes a cada iteración. Además, se lleva a cabo una revisión del plan de iteraciones en caso de existir alguna posible modificación. A partir del desglose de cada historia de usuario, se le definen un conjunto de tareas de programación o ingeniería que permiten implementar exitosamente cada una de estas.

Tareas de ingeniería

A continuación, se definen las tareas de ingeniería para las historias de usuario “Administrar parcialmente trabajadores” y “Administrar parcialmente productos”. Estas tareas tienen como objetivo mostrar de manera simple cada una de las actividades necesarias para dar cumplimiento a las historias de usuario.

Tabla 6: Tareas de ingeniería. Fuente: Elaboración propia

Historias de usuario	Tareas de ingeniería
Administrar parcialmente trabajadores	1. Crear trabajador 2. Modificar trabajador 3. Listar trabajador
Administrar parcialmente productos	1. Crear producto 2. Modificar producto 3. Eliminar producto

Tabla: Tarea de ingeniería “Crear trabajador”

Tarea	
Número de tarea: 1	**Número de historia de usuario:** 1
Nombre de la tarea: Crear trabajador	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.8
Fecha de inicio: 10 de septiembre	**Fecha de fin:** 12 de septiembre
Descripción: El usuario puede insertar un nuevo trabajador	

Tabla: Tarea de ingeniería “Modificar trabajador”

Tarea	
Número de tarea: 2	**Número de historia de usuario:** 1
Nombre de la tarea: Modificar trabajador	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.8
Fecha de inicio: 12 de enero	**Fecha de fin:** 14 de septiembre
Descripción: El usuario puede modificar un trabajador en caso de que exista.	

3.2 Pruebas de software

Las pruebas son el proceso que consiste en todas las actividades del ciclo de vida, tanto estáticas como dinámicas relacionadas con la planificación, preparación y evaluación de productos de software y productos relacionados con el trabajo para determinar que cumplen los requisitos especificados, para demostrar que son aptos para el propósito y para detectar defectos. Las pruebas de software son la investigación empírica y técnica realizada para facilitar a los interesados información sobre la calidad del producto o servicio bajo pruebas. El objetivo principal de las pruebas es aportar calidad al producto que se está desarrollando (Sánchez Peño, 2015).

Se decide realizar las pruebas de aceptación a los módulos implementados, debido a que el objetivo de estas es verificar que el sistema cumpla con los requisitos establecidos por el usuario. De esta forma se puede obtener el grado de satisfacción del cliente.

3.2.1 Pruebas unitarias

En programación, una prueba unitaria o test unitario es una forma efectiva de comprobar el correcto funcionamiento de las unidades individuales más pequeñas de los programas informáticos . Por ejemplo, en diseño estructurado o en diseño funcional una función o un procedimiento, en diseño orientado a objetos una clase. Además, garantizan que:

- Se ejecutan al menos una vez todos los caminos independientes de cada interfaz.
- Se utilizan las decisiones en su parte verdadera y en su parte falsa.
- Se ejecuten todos los bucles en sus límites.
- Se utilizan todas las estructuras de datos internas.

Prueba del camino básico

La prueba del camino básico, es una prueba de «caja blanca» que consiste en verificar el código de nuestros sistemas de manera que comprobemos que todo funciona correctamente, es decir, se debe verificar que todas las instrucciones del programa se ejecutan por lo menos una vez.

Los pasos para desarrollar la prueba del camino básico son:

1.- Dibujar el grafo de flujo.

2.- Calcular la complejidad ciclomática.

3.- Determinar el conjunto básico de caminos independientes.

Paso 1: Dibujar el grafo de flujo:

Detectamos los nodos que conformaran el grafo de flujo, así como los caminos que se pueden recorrer durante la ejecución del programa.

Paso 2: Complejidad Ciclomática:

La complejidad ciclomática mide el número de caminos independientes dentro del código que es sometido a prueba. La fórmula para su cálculo es:

$V(G) = a – n +2$

$V(G) = 11-9+2$

donde:

a: Es el número de aristas (lados).
n: Es el número de nodos (vértices).

Paso 3: Caminos independientes:
Se van formando los caminos independientes (4 según la complejidad ciclomática) desde el más largo al más corto observando nuestro grafo de flujo.

En la figura 10 se muestra el ejemplo de la clase TrabajadorActivity, específicamente la función createDialog, la cual presenta una complejidad ciclomática de 4.

Caso de prueba para el camino básico 1.
Descripción: Cierra el diálogo sin insertar ningún trabajador enviando un mensaje de error.
Condición de ejecución: Si existen campos vacíos en el formulario de trabajador.
Procedimiento de prueba: Automatizada.
Datos de entrada: nombre, apellidos, ci, genero, fumador, residente, saldo, estado.
Tipo de dato esperado: Vacío.
Evaluación del caso de prueba: Satisfactoria.

Caso de prueba para el camino básico 2.
Descripción: Cierra el diálogo sin insertar ningún trabajador enviando un mensaje de error.
Condición de ejecución: Si se introduce algún dato erróneo.
Procedimiento de prueba: Automatizada.
Datos de entrada: nombre, apellidos, ci, genero, fumador, residente, saldo, estado.
Tipo de dato esperado: Vacío.
Evaluación del caso de prueba: Satisfactoria.

Caso de prueba para el camino básico 3.

Descripción: Inserta un nuevo trabajador en la lista de trabajadores.

Condición de ejecución: Si todos los datos son introducidos correctamente y no existen campos vacíos en el formulario.

Procedimiento de prueba: Automatizada.

Datos de entrada: nombre, apellidos, ci, genero, fumador, residente, saldo, estado.

Tipo de dato esperado: Trabajador.

Evaluación del caso de prueba: Satisfactoria.

A continuación, se muestra el fragmento del código de la función createDialog().

```
private fun createDialog() {
    val dialogBinding = DialogTrabajadorBinding.inflate(layoutInflater)

    //creación de los adaptadores de los spinners, declaración de variables
    val adapterGenero = ArrayAdapter(
        this,
        R.layout.spinner_items,
        resources.getStringArray(R.array.gender)
    )
    dialogBinding.spGenero.adapter = adapterGenero

    val adapterFumador = ArrayAdapter(
        this,
        R.layout.spinner_items,
        resources.getStringArray(R.array.fumador)
    )
    dialogBinding.spFumador.adapter = adapterFumador

    val adapterResidente = ArrayAdapter(
        this,
        R.layout.spinner_items,
```

```
        resources.getStringArray(R.array.residente)
    )
    dialogBinding.spResidente.adapter = adapterResidente

    val adapterEstado = ArrayAdapter(
        this,
        R.layout.spinner_items,
        resources.getStringArray(R.array.estado)
    )
    dialogBinding.spEstado.adapter = adapterEstado

    //Inicialización del diálogo

    val builder = AlertDialog.Builder(this)
    builder.setTitle("Adicionar trabajador")
    builder.setView(dialogBinding.root)

    val dialog = builder.create()
    dialog.show()

    var selected = false

    dialogBinding.acceptButton.setOnClickListener {

//bloque if
        if (dialogBinding.etNombre.text.isEmpty() || dialogBinding.etApellidos.text.isEmpty()
|| dialogBinding.etCi.text.isEmpty() ||
            dialogBinding.etApellidos.text.isEmpty() || dialogBinding.etSaldo.text.isEmpty()
        ) {
            Toast.makeText(this, "Existen campos vacíos", Toast.LENGTH_SHORT).show()
        } else if (dialogBinding.etCi.text.length < 11) {
            Toast.makeText(
                this,
```

```
                "El Carnet de Identidad debe tener 11 dígitos",
                Toast.LENGTH_LONG
            ).show()
        } else {
            val nombre = dialogBinding.etNombre.text.toString()
            val apellidos = dialogBinding.etApellidos.text.toString()
            val ci = dialogBinding.etCi.text.toString()
            val fumador = dialogBinding.spFumador.selectedItem.toString()
            val residente = dialogBinding.spResidente.selectedItem.toString()
            val genero = dialogBinding.spGenero.selectedItem.toString()
            val estado = dialogBinding.spEstado.selectedItem.toString()
            val saldo = dialogBinding.etSaldo.text.toString().toDouble()
            val trabajador = Trabajador(
                null,
                nombre,
                apellidos,
                ci,
                genero,
                fumador,
                residente,
                saldo,
                estado
            )
            selected = true

            CoroutineScope(Dispatchers.IO).launch {
                appViewModel.insertTrabajador(trabajador)
            }

            if (selected) {
                dialog.dismiss()
            }
        }
```

```
    }
    dialogBinding.buttonCancel.setOnClickListener {
        dialog.dismiss()
    }
}
```

Se logró el cálculo de la complejidad ciclomática para los algoritmos de la aplicación. La misma visualiza la forma que se emplea para obtener dicho cálculo con las estructuras especificadas. Los resultados adquiridos en el cálculo de la complejidad ciclomática definen el número de caminos independientes dentro de un fragmento de código y los casos de pruebas diseñados.

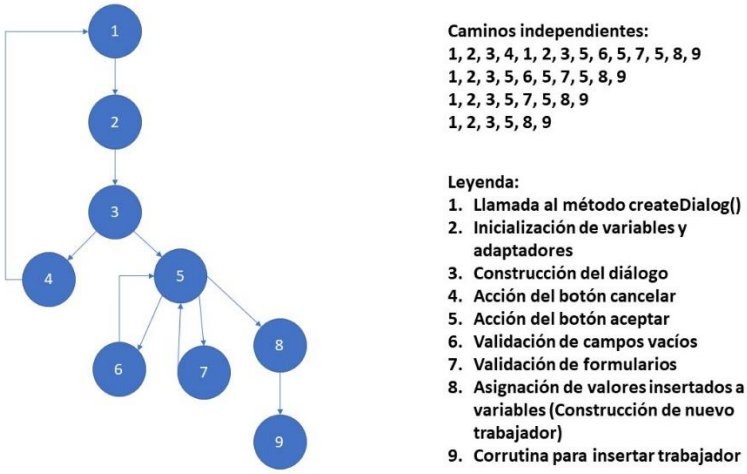


Figura 10: Grafo de la función createDialog. Fuente: Elaboración propia

3.2.2 Pruebas de aceptación

En ingeniería de software y pruebas de software, las pruebas de aceptación (User Acceptance Testing**,** UAT) pertenecen a las últimas etapas previas a la liberación en firme de versiones nuevas a fin de determinar si cumplen con las necesidades y/o requerimientos de las empresas y sus usuarios. Al finalizar las pruebas automatizadas, que garantizan los requisitos tecnológicos del diseño inicial, se pasa a las pruebas manuales.

Dichas pruebas manuales primero son hechas por usuarios internos en entornos intermedios: ambientes típicos de producción donde se verifica que se desempeña del modo necesitado. Luego viene el acceso beta a los clientes que así lo soliciten repitiendo de nuevo el ciclo, pero en esta oportunidad en entornos realistas y muchas veces muy diferentes entre sí en cuanto a otros softwares agregados (Pressman, 2007).

A continuación, se muestran las representaciones de varias pruebas de aceptación realizadas a la aplicación.

Tabla 7: Pruebas de aceptación. Fuente: Elaboración propia

Caso de prueba de aceptación	
Código: HU1_PA1	**Historia de usuario:** 1
Nombre: Crear trabajador.	
Descripción: Prueba para la funcionalidad Crear trabajador.	
Condiciones de ejecución: El usuario debe rellenar el formulario de trabajador correctamente.	
Pasos de ejecución: Se intenta añadir un nuevo trabajador.	
Resultados esperados: El trabajador fue añadido correctamente	

Caso de prueba de aceptación	
Código: HU1_PA2	**Historia de usuario:** 1
Nombre: Editar trabajador	
Descripción: Prueba para la funcionalidad Editar trabajador	
Condiciones de ejecución: El usuario debe haber creado previamente un trabajador y seleccionarlo.	
Pasos de ejecución: Se intenta editar los campos del formulario existente y luego actualizar el trabajador.	

Resultados esperados: El trabajador fue modificado correctamente.

Caso de prueba de aceptación	
Código: HU3_PA1	**Historia de usuario:** 3
Nombre: Crear asignación	
Descripción: Prueba para la funcionalidad Crear asignación	
Condiciones de ejecución: El usuario debe haber creado previamente un trabajador y un producto a asignar.	
Pasos de ejecución: Se intenta crear una asignación.	
Resultados esperados: La asignación fue añadida correctamente.	

En la siguiente figura se muestran los casos de prueba realizados con las inconformidades presentadas, las cuales fueron tomadas en cuenta para su posterior solución.

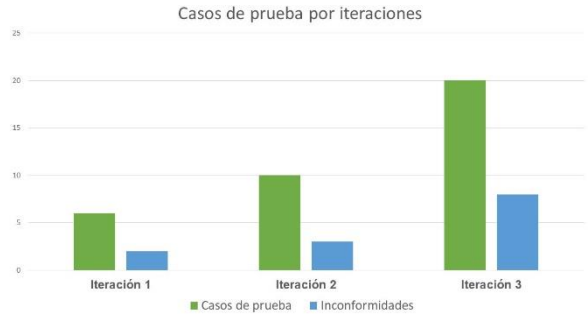


Figura 11: Pruebas de aceptación. Fuente: Elaboración propia

3.3 Satisfacción de potenciales usuarios

Para evaluar los niveles de satisfacción de la aplicación propuesta como solución, se concibió una encuesta a 20 trabajadores del centro de forma anónima. Para la elaboración de esta encuesta fue utilizada una técnica creada en su versión original por V. A. Iadov para evaluar la satisfacción de los usuarios, la misma posee dos preguntas cerradas y dos abiertas. La técnica de Iadov constituye una vía indirecta para el estudio de la satisfacción, ya que los criterios que se utilizan se fundamentan en las relaciones que se establecen entre las preguntas cerradas, que se intercalan dentro de un cuestionario y cuya relación el encuestado desconoce.

Tabla 8: Técnica de V. A. Iadov aplicada en la encuesta

	¿Cree usted que la aplicación Asignador de Productos es complicada de utilizar?								
	No			No sé			Si		
¿Le gusta la aplicación para el proceso de asignación de los productos a los trabajadores del centro?	¿Utilizaría Asignador de Productos para realizar el proceso de asignaciones de productos a los trabajadores?								
	Si	No sé	No	Si	No sé	No	Si	No sé	No
Me gusta mucho	1	2	6	2	2	6	6	6	6
No me gusta tanto	2	2	3	2	3	3	6	3	6
Me da lo mismo	3	3	3	3	3	3	3	3	3
Me disgusta más de lo que me gusta	6	3	6	3	4	4	3	4	4
No me gusta nada	6	6	6	6	4	4	6	4	5
No sé qué decir	2	3	6	3	3	3	6	3	4

El número resultante de la interrelación de las tres preguntas nos indica la posición de cada encuestado en la siguiente escala de satisfacción:

1. Clara satisfacción.
2. Más satisfecho que insatisfecho.
3. No definida.
4. Más insatisfecho que satisfecho.
5. Clara insatisfacción.
6. Contradictoria.

El índice de satisfacción grupal (ISG) se expresa en una escala numérica que va desde +1 (máxima satisfacción), hasta -1 (máxima insatisfacción). Para obtenerlo se trabaja con los diferentes niveles de satisfacción obtenidos para cada encuestado de la forma que se muestra en la fórmula:

$$ISG = \frac{A(+1) + B(+0.5) + C(0) + D(-0.5) + E(-1)}{N}$$

Para ponderar el ISG se establece una escala numérica entre +1 y -1 como se muestra a continuación:

- +1: Máximo de satisfacción
- +0.5: Más satisfecho que insatisfecho
- 0: No definido y contradictorio
- -0.5: Más insatisfecho que satisfecho
- -1: Máxima insatisfacción

En las figuras 12 y 13 se muestran el procedimiento realizado para el cálculo del Índice de Satisfacción Grupal (ISG) y el gráfico de satisfacción respectivamente. Este último en la modalidad de gráfico de pastel.

Clara Satisfacción	Más satisfecho que insatisfecho	No definida	Más insatisfecho que satisfecho	Clara insatisfacción	Contradictoria
9	7	2	1	0	1

$$ISG = \frac{A(+1) + B(+0.5) + C(0) + D(-0.5) + E(-1)}{N}$$

$$ISG = \frac{9(+1) + 7(+0.5) + 3(0) + 1(-0.5) + 0(-1)}{20}$$

$$ISG = \frac{9 + 3.5 + 0 - 0.5 + 0}{20}$$

$$ISG = \frac{12.5 - 0.5}{20}$$

$$ISG = 0.6$$

Figura 12: Cálculo de Índice de Satisfacción Grupal. Fuente: Elaboración propia

Luego de realizado el proceso de validación mediante la técnica de Iadov de la consulta a los usuarios donde se ha implementado el sistema de indicadores propuesto, se confirma su factibilidad de uso, expresado cuantitativamente en el Índice de Satisfacción Grupal (ISG = 0.6) donde se evidencia la satisfacción por el sistema de gestión de asignaciones de productos a los trabajadores del centro, lo que refleja la aceptación de la propuesta y un reconocimiento a su utilidad.

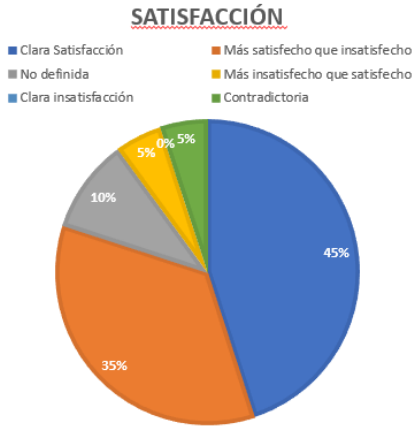


Figura 13: Gráfico de satisfacción. Fuente: Elaboración propia

Conclusiones del capítulo

Durante el desarrollo de este capítulo se aprecia que la especificación de las tareas de programación permitió obtener un desglose por unidades de las funcionalidades encapsuladas mediante las historias de usuario. Los diseños de los casos de prueba de aceptación y unidad sirven de guía para la ejecución de las actividades de verificación y validación para un correcto funcionamiento de la aplicación. Las no conformidades resultantes por iteraciones permiten realizar un análisis incremental sobre el funcionamiento de la aplicación Android para la gestión de las asignaciones de productos a los trabajadores del Centro de Software Libre. Por último, se aplica la técnica de Iadov para determinar el nivel de satisfacción de los usuarios, el cual determinó un índice de satisfacción grupal de 0,6 que demostró su aceptación por parte de los usuarios.

CONCLUSIONES GENERALES

Durante el desarrollo de este trabajo, se obtuvo como resultado una aplicación móvil para la gestión de las asignaciones de los productos a los trabajadores del Centro de Software Libre.

- Se determinó que, luego de la realización de un estudio a profundidad de sistemas homólogos, ninguno de los resultados se adecuaba a la solución de la problemática presentada al inicio, pero fueron tomadas en cuenta las buenas prácticas en el desarrollo de aplicaciones móviles y las diferentes herramientas y tecnologías que se emplearon en ellas.
- Se logró desarrollar las estructuras de datos adecuadas para el correcto funcionamiento del sistema acorde a las necesidades presentadas, permitiendo, además, la correcta toma de los requisitos funcionales de la herramienta creada.
- El uso de la metodología AUP en su modelo UCI, en su cuarta variante, dio paso a la generación de los artefactos de ingeniería de software utilizados: la planificación de las historias de usuario junto a su plan de iteraciones, las tarjetas CRC y modelo de datos, así como el patrón arquitectónico *Model-View-ViewModel*.
- Con la infraestructura tecnológica conformada por las herramientas utilizadas y correctamente actualizadas, fue posible la creación de una herramienta con la función de brindar portabilidad y estructuración de los datos pertinentes en un solo lugar, mediante sencillas interfaces flexibles para su uso.
- La aplicación fue sometida a pruebas unitarias para la comprobación de su funcionamiento, pruebas de aceptación y la aplicación de la técnica de V. A. Iadov para hallar el índice de satifacción, resultando satisfactorio y dando paso al comienzo de la usabilidad, experiencia y satisfacción del cliente con la garantía de haberse cumplido con los objetivos generales y específicos planteados en este trabajo para dar solución a la problemática presentada.

RECOMENDACIONES

Se recomienda para futuras actualizaciones de la aplicación y dar continuidad a la presente investigación:

- Establecer tres colores para el campo “saldo” de los trabajadores, de forma que se pueda intuir cuándo un trabajador presenta saldo negativo (color rojo), cercano a cero (color naranja) o posea el suficiente saldo para realizar la compra (color verde).
- Realizar varias asignaciones de un mismo producto a varios trabajadores y viceversa.
- Aplicar de forma correcta los patrones de usabilidad a nivel de usuario dentro de las diferentes interfaces.
- Incorporar otras funcionalidades para lograr llevar su uso hacia otros niveles, por ejemplo, a nivel de institución.

Referencias bibliográficas

Aliernet. 2022. Aliernet. [En línea] 2022. [Citado el: 30 de 09 de 2022.] www.aliernet.com.

Angulo, Roberto. 2013. Debates IESA. *Debates IESA.* [En línea] 2013. [Citado el: 10 de Junio de 2022.] cmapspublic2.ihmc.us.

Araya, Raúl Alberto Garita. 2013. Dialnet. *Dialnet.* [En línea] 1 de julio de 2013. [Citado el: 13 de junio de 2022.] https://dialnet.unirioja.es/servlet/articulo?codigo=5511036.

Arcos-Medina, Gloria, Menéndez, Jorge y Vallejo, Javier. 2018. KnE Enginnering. [En línea] 2018. [Citado el: 7 de Septiembre de 2022.] https://knepublishing.com/index.php/KnE-Engineering/article/view/1498/3384.

Asencio, Jaime Antonio Rodríguez. 2014. Universidad Autónoma de Ciudad Juárez - Recursos Electrónicos. *Universidad Autónoma de Ciudad Juárez - Recursos Electrónicos.* [En línea] mayo de 2014. [Citado el: 26 de Junio de 2022.]

Bohórquez, Diana Paola y Chaviano, Orlando Gregorio. 2017. Biblioteca Nacional de Cuba José Martí. *Biblioteca Nacional de Cuba José Martí.* [En línea] 2017. [Citado el: 24 de Junio de 2022.] http://revistas.bnjm.cu/index.php/BAI/article/view/147.

Carmona, Juan García. 2012. *SOLID y GRASP. Buenas prácticas hacia el éxito en el desarrollo de software.* 2012.

Chester SW. 2022. Google Play Store. [En línea] 2022. [Citado el: 9 de Julio de 2022.] https://play.google.com/store/apps/details?id=com.stockmanagment.next.app&hl=es&gl=US.

Contributor, TechTarget. 2019. TechTarget. *TechTarget.* [En línea] Agosto de 2019. [Citado el: 10 de Junio de 2022.] https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app.

Escribano, David. 2018. Skyscanner. *Skyscanner.* [En línea] 27 de Noviembre de 2018. [Citado el: 13 de Junio de 2022.] https://www.skyscanner.es/noticias/esta-es-la-historia-de-las-aplicaciones-moviles.

Flores Caiza, Enver Danilo y Acosta Galindo, Freddy Alfredo. 2018. Universidad Politécnica Salesiana de Ecuador. *Universidad Politécnica Salesiana de Ecuador.* [En línea] Agosto de 2018. [Citado el: 25 de Junio de 2022.] https://dspace.ups.edu.ec/handle/123456789/16018.

Gabriel, Lic. Enriquez Juan. 2014. Informes CT. *Informes CT.* [En línea] 11 de junio de 2014. [Citado el: 10 de junio de 2022.] https://publicaciones.unpa.edu.ar/.

Gamma, Erich, y otros. 2008. *Design Patterns. Elements of Reusable Object-Oriented Software.* 2008.

Haines, Steven. 2017. Wayback Machine. [En línea] 12 de Abril de 2017. [Citado el: 9 de Julio de 2022.] https://web.archive.org/web/20170412180225/http://stevenhaines.net/assets/intro-to-product-manager-s-desk-reference.pdf.

Joskowicz, José. 2008. *Reglas y prácticas en eXtreme Programming.* 2008.

Laboratorio Magnético. 2020. Vipapk. [En línea] 2020. [Citado el: 9 de Julio de 2022.] https://vipapk.org/es/stock-and-inventory-management-system-pro-apk-1-2/.

López, Bryan Salazar. 2021. Ingeniería Industrial Online. [En línea] 24 de Julio de 2021. [Citado el: 9 de Julio de 2022.] https://www.ingenieriaindustrialonline.com/gestion-de-almacenes/que-es-la-gestion-de-almacenes/.

Madruga, Dalexi González. 2022. Apklis. *Apklis.* [En línea] 3 de Julio de 2022. [Citado el: 3 de Julio de 2022.] https://www.apklis.cu/application/com.habanaservicentropro.com.

Martínez, Alfonso. 2017. Cuatroochenta. *Cuatroochenta.* [En línea] 06 de Junio de 2017. [Citado el: 13 de Junio de 2022.] https://cuatroochenta.com/app-hibrida-o-app-nativa-segun-para-que/.

Mendoza, María Guadalupe García. 2015. Academia. *Academia.* [En línea] 7 de Julio de 2015. [Citado el: 10 de Junio de 2022.] www.academia.edu.

Pressman, Roger. 2007. *Ingeniería de Software. Un enfoque práctico.* 2007.

Reyes Rodríguez, Eduardo Martín y Arce Huamán, Lenin Froy. 2019. Repositorio Institucional Digital de la Universidad del Callao. *Repositorio Institucional Digital de la Universidad del Callao.* [En línea] 2019. [Citado el: 25 de Junio de 2022.] http://repositorio.unac.edu.pe/handle/20.500.12952/4140.

Rivero, Luis. 2021. Apklis. [En línea] 7 de Agosto de 2021. [Citado el: 9 de Julio de 2022.] https://www.apklis.cu/application/com.luis.almacena.negocio.

Sánchez Peño, José Manuel. 2015. Archivo Digital UPM. [En línea] 16 de Junio de 2015. [Citado el: 22 de Octubre de 2022.] https://oa.upm.es/40012/.

Sommerville, I. 2007. *Software Enginnering.* 2007.

Valencia, Daniela, y otros. 2017. Investigación y Desarrollo en TIC - Revistas Científicas Universidad Simón Bolívar. *Investigación y Desarrollo en TIC - Revistas Científicas Universidad Simón Bolívar.* [En línea] 1 de Marzo de 2017. [Citado el: 25 de Junio de 2022.] https://revistas.unisimon.edu.co/index.php/identic/article/view/2478.

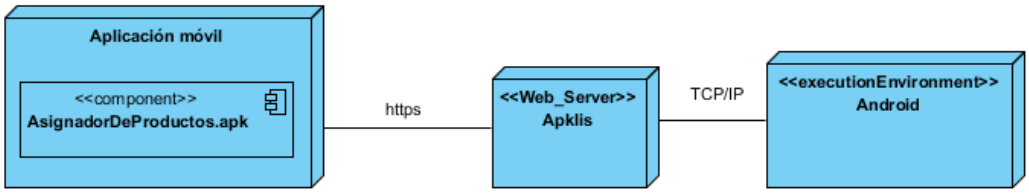
Vidal Ledo, María Josefina y Araña Pérez, Ana Bárbara. 2012. Revista Cubana de Educación Médica Superior. *Revista Cubana de Educación Médica Superior.* [En línea] 2012. [Citado el: 10 de junio de 2022.] https://www.medigraphic.com/.

Villa Jiménez, Oscar Manuel, y otros. 2018. Cuba Salud 2018. [En línea] 2018. [Citado el: 3 de Julio de 2022.] http://www.convencionsalud2018.sld.cu/index.php/connvencionsalud/2018/paper/viewPaper/1754.

Vique, Robert Ramírez. 2013. Exaforo.com. *Exaforo.com.* [En línea] 2013. [Citado el: 13 de Junio de 2022.] https://www.exabyteinformatica.com/uoc/Informatica/Tecnologia_y_desarrollo_en_dispositivos_moviles/Tecnologia_y_desarrollo_en_dispositivos_moviles_(Modulo_4).pdf.

ANEXOS

Anexo 1: Diagrama de despliegue de la aplicación



Anexo 2: Modelo de encuesta realizada a los secretarios del sindicato del Centro de Software Libre

Esta encuesta tiene como objetivo recopilar información acerca de las funcionalidades que desea como cliente para el funcionamiento de la aplicación móvil a realizar, además de evaluar el proceso de asignación de productos actual y conocer su funcionamiento.

¿Cuál es el actual sistema utilizado para realizar el proceso de asignaciones de productos y cómo funciona?

En base a lo respondido anteriormente:

¿Cuáles son las diferentes entidades a gestionar?

¿Se deben gestionar de forma total o parcial?

En caso de ser parcial especificar las funcionalidades.

¿Cuáles son los atributos a tener en cuenta para las diferentes entidades?

En aras de establecer los diferentes requisitos de la aplicación:

¿Cuáles son las principales funcionalidades que desea para esta aplicación?

¿Qué tipos de reporte desea generar?

¿Qué utilidad tendrán estos reportes generados?

¿Cree usted que la aplicación a desarrollar sería de gran utilidad para el proceso de asignaciones de productos?

¿Qué mejoras incorporaría en la aplicación?

Anexo 3: Historias de usuario

Historia de usuario	
Numero: HU 1	**Nombre Historia de usuario:** Administrar parcialmente base de datos
Usuario: : Lian Felipe Hernández Delgado	**Iteración asignada: 1**
Prioridad en negocio: Alta	**Puntos estimados: 1**
Riesgo en desarrollo: Media	**Puntos reales: 1**
Descripción: Permite insertar, modificar y listar los elementos de la base de datos.	
Observaciones: Se guardan todos los datos de los productos, permite modificarlos y listarlos.	

Historia de usuario	
Numero: HU 3	**Nombre Historia de usuario:** Administrar parcialmente productos
Usuario: : Lian Felipe Hernández Delgado	**Iteración asignada: 1**
Prioridad en negocio: Alta	**Puntos estimados: 2**
Riesgo en desarrollo: Media	**Puntos reales: 2**
Descripción: Permite insertar, modificar y listar los productos.	

Observaciones: Se guardan todos los datos de los productos, permite modificarlos y listarlos.

Historia de usuario	
Numero: HU 4	**Nombre Historia de usuario:** Administrar parcialmente asignaciones
Usuario: Lian Felipe Hernández Delgado	**Iteración asignada: 2**
Prioridad en negocio: Alta	**Puntos estimados: 3**
Riesgo en desarrollo: Media	**Puntos reales: 3**
Descripción: Permite insertar, modificar y listar las asignaciones	
Descripción: Se guardan todos los datos de las asignaciones, permite modificarlos y listarlos.	

Historia de usuario	
Numero: HU 5	**Nombre Historia de usuario:** Administrar parcialmente hijos de los trabajadores
Usuario: Lian Felipe Hernández Delgado	**Iteración asignada: 2**
Prioridad en negocio: Alta	**Puntos estimados: 2**
Riesgo en desarrollo: Media	**Puntos reales: 2**
Descripción: Permite insertar, modificar y listar los hijos de los trabajadores.	
Observaciones: Se guardan todos los datos de los hijos de cada trabajador, permite modificarlos y listarlos.	

Historia de usuario	
Numero: HU 6	**Nombre Historia de usuario:** Generar reportes
Usuario: Lian Felipe Hernández Delgado	**Iteración asignada: 2**
Prioridad en negocio: Alta	**Puntos estimados: 3**
Riesgo en desarrollo: Media	**Puntos reales: 3**
Descripción: Permite generar un listado como reporte sobre los trabajadores teniendo en cuenta uno o varios criterios	
Observaciones: Se introducen uno o varios criterios y de acuerdo a esto(s) se genera un reporte en forma de lista.	

Historia de usuario	
Numero: HU 7	**Nombre Historia de usuario:** Filtrar trabajadores
Usuario: Lian Felipe Hernández Delgado	**Iteración asignada: 3**
Prioridad en negocio: Baja	**Puntos estimados: 1**
Riesgo en desarrollo: Bajo	**Puntos reales: 1**
Descripción: Permite filtrar los trabajadores en base a uno o varios de sus atributos.	
Observaciones: Se listan los trabajadores filtrados teniendo en cuenta los atributos seleccionados.	

Historia de usuario

Numero: HU 8	**Nombre Historia de usuario:** Filtrar productos
Usuario: Lian Felipe Hernández Delgado	**Iteración asignada: 3**
Prioridad en negocio: Baja	**Puntos estimados: 1**
Riesgo en desarrollo: Bajo	**Puntos reales: 1**
Descripción: Permite filtrar los productos en base a uno o varios de sus atributos.	
Observaciones: Se listan los productos filtrados teniendo en cuenta los atributos seleccionados.	

Historia de usuario	
Numero: HU 9	**Nombre Historia de usuario:** Filtrar asignaciones
Usuario: Lian Felipe Hernández Delgado	**Iteración asignada: 3**
Prioridad en negocio: Baja	**Puntos estimados: 1**
Riesgo en desarrollo: Bajo	**Puntos reales: 1**
Descripción: Permite filtrar las asignaciones en base a uno o varios de sus atributos.	
Observaciones: Se listan las asignaciones filtradas teniendo en cuenta los atributos seleccionados.	

Historia de usuario	
Numero: HU 10	**Nombre Historia de usuario:** Filtrar hijos de los trabajadores
Usuario: Lian Felipe Hernández Delgado	**Iteración asignada: 3**
Prioridad en negocio: Baja	**Puntos estimados: 1**

Riesgo en desarrollo: Bajo	**Puntos reales: 1**
Descripción: Permite filtrar los hijos de los trabajadores en base a uno o varios de sus atributos.	
Observaciones: Se listan los hijos filtrados teniendo en cuenta los atributos seleccionados.	

Anexo 4: Tarjetas CRC

Nombre de la clase: AsignarActivity	
Responsabilidades	**Clases relacionadas**
onCreate()	Asignacion
futurosDeudoresFilter()	Hijo
deudoresFilter()	Producto
createDialogFilter()	Trabajador
onResume()	TrabajadorAsignacionAdapter
showData()	AppViewModel
loadData()	

Nombre de la clase: DetallesAsignacionActivity	
Responsabilidades	**Clases relacionadas**
onCreate()	Asignacion
dialogEditAsignacion()	Producto
onResume()	Trabajador
showData()	AppViewModel
loadData()	

Nombre de la clase: DetallesHijoActivity	
Responsabilidades	**Clases relacionadas**
onCreate()	Hijo
takeAge()	Trabajador
createEditDialog()	AppViewModel
onResume()	
showData()	
loadData()	

Nombre de la clase: DetallesProductoActivity	
Responsabilidades	**Clases relacionadas**
onCreate()	Producto
showData()	AppViewModel
loadData()	
createDialog()	
onResume()	

Nombre de la clase: DetallesTrabajadorActivity	
Responsabilidades	**Clases relacionadas**
onCreate()	Asignacion
showData()	Hijo

loadData()	Trabajador
addSaldo()	Producto
createHijo()	AppViewModel
createEditDialog()	
onResume()	

Nombre de la clase: HijoActivity	
Responsabilidades	**Clases relacionadas**
onCreate()	Hijo
showData()	Trabajador
loadData()	HijoAdapter
onResume()	TrabajadorAdapter
createDialogSearch()	AppViewModel
takeAge()	

Nombre de la clase: MainActivity	
Responsabilidades	**Clases relacionadas**
onCreate()	AppViewModel

Nombre de la clase: HijoActivity	
Responsabilidades	**Clases relacionadas**
onCreate()	Producto

showData()	ProductoAdapter
loadData()	AppViewModel
onResume()	
createDialogSearch()	
createProductoDialog()	

Nombre de la clase: TrabajadorActivity	
Responsabilidades	**Clases relacionadas**
onCreate()	Trabajador
showData()	TrabajadorAdapter
loadData()	AppViewModel
onResume()	
createDialogSearch()	
createDialog()	

Nombre de la clase: AppViewModel	
Responsabilidades	**Clases relacionadas**
getAllTrabajadores()	Trabajador
getTrabajadorById(id: Int)	Producto
insertTrabajador(trabajador: Trabajador)	Hijo
updateTrabajador(trabajador: Trabajador)	Asignacion
getAllProductos()	TrabajadorDao
	ProductoDao

getProductoById(id: Int)	HijoDao
insertProducto(producto: Producto)	AsignacionDao
updatePrducto(producto: Producto)	
getHijoById(id: Int)	
getAsignacionById(id: Int)	
getAllAsignaciones()	
getAllHijos()	
insertHijo(hijo: Hijo)	
updateHijo(hijo: Hijo)	
insertAsignacion(asignacion: Asignacion)	
insertAllAsignacion(asignaciones: List<Asignacion>)	
updateAsignacion(asignacion: Asignacion)	