UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Implementación de Kohinor ERP en su versión móvil para dispositivos Android

Sebastián Cueva Andrade

Ingeniería en Ciencias de la Computación

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero en Ciencias de la Computación

Quito, 6 de mayo de 2022

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Implementación de Kohinor ERP en su versión móvil para dispositivos Android

Sebastián Cueva Andrade

Nombre del profesor, Título académico **Noel Pérez Pérez, PhD**

Quito, 6 de mayo de 2022

© DERECHOS DE AUTOR

Nombres y apellidos: Sebastián Cueva Andrade

Código: 00202630

Cédula de identidad: 1722168208

Lugar y fecha: Quito, 6 de mayo de 2022

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

Kohinor ERP es un producto de software con más 20 años en el mercado que ha adquirido una robusta estructura de funcionamiento que es capaz de adaptarse a muchos tipos de negocios, para servir como una herramienta de administración y planificación empresarial. Construida sobre un motor de SQL Server como fuente de datos, se distribuye en distintas presentaciones como su versión web construida en ASP.NET y su versión de escritorio en Visual Basic. Se presenta la reingeniería de Kohinor ERP en su versión para teléfonos inteligentes usando Xamarin Forms para dispositivos Android.

Palabras clave: ERP, módulos, Xamarin Forms, aplicación móvil, SOA, C#.

ABSTRACT

Kohinor ERP is a software product with more than 20 years in the market that has acquired a robust operating structure that is able to adapt to many types of businesses, to serve as a management and business planning tool. Built on a SQL Server engine as a data source, it is distributed in different presentations such as its web version built in ASP.NET and its desktop version in Visual Basic. The re-engineering of Kohinor ERP in its smartphone version using Xamarin Forms for Android devices is presented.

Key words: ERP, modules, Xamarin Forms, mobile application, SOA.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

ERP

Un software de tipo Enterprise Resource Planning (o ERP) consiste en una herramienta informática de tipo back office (o uso interno al cual el cliente por lo general no tiene acceso) que sirve para manejar la lógica de un negocio usando módulos de trabajo. Estos módulos pueden ser de ventas, compras, inventarios, manejo de personal, etc. Una de las ventajas de este tipo de programas es que aumentan la organización de la empresa y permite llevar un registro detallado de los movimientos realizados, lo que genera una trazabilidad y entendimiento profundo de cómo se usan los recursos permitiendo a las empresas aumentar su productividad.

Por otra parte, también se hacen cargo de procesos que suelen ser tediosos o largos, y permiten centralizar información en una misma plataforma. De hecho, un estudio realizado por Panorama Consulting afirma que alrededor de 40% de las empresas que adquieren un ERP logran ver aumentos significativos de productividad. (European Knowledge Center for Information Technology, 2015). El caso específico presentado en el trabajo es sobre Kohinor ERP o en su abreviatura para futuras referencias *KERP*.

Contexto general sobre Kohinor ERP

Los mercados y las tendencias empresariales han forzado a KERP a reconstruirse para cubrir un vasto mercado con distintos recursos, ofreciendo la misma solución con distintas facetas, cada cual adaptable a las necesidades de los negocios. Esto permite estar al corriente de nuevas tecnologías, pero al mismo tiempo conserva un software funcional. Actualmente existen dos distribuciones de KERP, la versión de escritorio que suele ser la opción más viable para negocios pequeños y su versión web, una solución más escalable y fácil de distribuir que suele ser la opción para empresas medianas y grandes. Desarrollada

principalmente en una base de datos SQL Server, los distintos paradigmas que ofrecen las herramientas integradas de Microsoft permite implementaciones en herramientas como Excel, Visual BASIC, C#, .NET y garantiza un funcionamiento correcto y escalable. En la actualidad el ERP posee los siguientes módulos:

- Inventarios
- Compras
- Ventas
- Punto de ventas
- Importaciones
- Control de producción
- Contabilidad
- Activos fijos
- Nómina
- Análisis gerencial
- Administración y seguridad

Objetivos generales y específicos

Como KERP posee un funcionamiento ya con experiencia en el mercado que ha demostrado ser altamente eficiente, no hay motivo para cambiar su modo de operación, ya que esto implica arriesgar o incluso desperdiciar una herramienta probada, es por eso que el **objetivo general** es realizar reingeniería para la construcción de su versión móvil que opere de manera familiar a los productos ya existentes, esto quiere decir que existirán los mismos módulos y funcionalidades. Manteniendo la lógica de negocios del ERP que yace en su base de datos es posible que todas las distribuciones de KERP puedan estar sincronizadas entre sí y funcionar en conjunto, lo que permite al usuario trabajar con distintas versiones de la

herramienta sobre el mismo conjunto de datos brindando la posibilidad de un uso inmediato sin tener que realizar modificaciones a los servidores de producción.

Por otra parte, según estadísticas del Banco Mundial, el en año 200 la subscripción a telefonía celular móvil en uso por cada 100 personas pasó de ser de 12.043 en el año 2000 a un 107.512 en el 2020 (Banco Mundial, 2022), esto quiere decir que existen más teléfonos activos funcionando día a día que seres humanos vivos en la actualidad. También se afirma en la misma fuente que en porcentaje de la población que hace uso de internet en su día a día pasó de un 6.34\% al inicio de siglo a un impresionante 56.67% en 2020 (Banco Mundial, 2022) y en Ecuador concretamente, las estadísticas mencionadas anteriormente son del 88% y 54% respectivamente. A pesar de no contar con una estadística definida sobre el uso de teléfonos móviles con acceso a internet destinados exclusivamente a fines de trabajo, la alta presencia de estos recursos ha creado necesidad de tener un enfoque empresarial en herramientas adecuadas específicamente para esas condiciones y necesidades. Por dicha razón, KERP tomará un nuevo enfoque por lo cual los **objetivos específicos** planteados son los siguientes:

- Re implementar KERP desarrollando una aplicación móvil llamada Kohinor Móvil.
- Un ERP basado en KERP que conserve el flujo de trabajo ya conocido sin perder el funcionamiento y efectividad de las otras implementaciones, con la capacidad de operar en bases de datos ya existentes.
- Mantener la compatibilidad y preferencia tecnológica al hacer uso de herramientas propias que brinda el proveedor de servicios de software de KERP bajo un framework asociado con Microsoft capaz de resolver las necesidades de desarrollo.
- Demostrar un caso de uso en el cual la nueva herramienta ponga a prueba su utilidad y su flujo operativo.
- Desarrollar el módulo de ventas y de inventario.

- Basarse en núcleo del ERP para las operaciones.

REQUERIMIENTOS FUNCIONALES

Si bien esto es un acercamiento a un ERP completo, los requerimientos funcionales se ven limitados por cuestiones de tiempo y podrían ser considerados para trabajo futuro. Se realiza una aproximación al problema estableciendo un número limitado de módulos funcionales que son considerados de uso más frecuente que otros. Para tener un contexto más detallado sobre el funcionamiento a continuación se presenta un diagrama general de la estructura que se encuentra actualmente en producción y la misma que será usada como guía para el desarrollo:

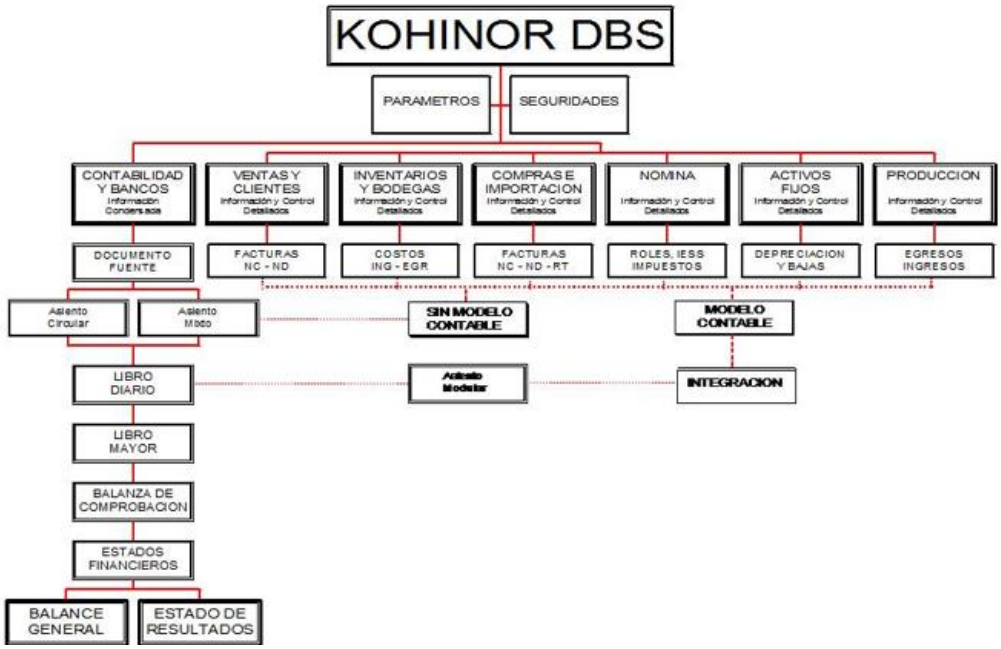

Figura # 1Módulos de Kohinor ERP.

De los módulos existentes el desarrollo girará en torno a dos que han sido seleccionados: ventas e inventario. Ambos módulos son capaces de generar un flujo operativo de ventas completo ya que se pueden consultar elementos disponibles y se los puede vender.

Adicional a esto, se le suma toda la logística que gira en torno al desarrollo de la aplicación móvil, por lo cual los requerimientos funcionales se dividen en dos categorías, siendo la primera la funcionalidad del ERP y los aspectos técnicos del desarrollo de software. Para la primera categoría están los siguientes requerimientos:

- Capacidad de generar nuevos pedidos para despacho de ventas: el usuario accede al módulo de ventas y debe ser capaz de generar un nuevo pedido, lo que incluye agregar elementos seleccionados con su cantidad, descripción y precio, al mismo tiempo que puede ver si el producto que desea aún tiene disponibilidad en stock. También debe ser capaz de seleccionar productos en distintos catálogos de precios y obviamente, registrar el pedido y enviarlo.
- Consultar un historial de pedidos: al existir un historial de ventas y al ser posible acceder al mismo, existe la posibilidad de volver a despachar lo mismo, ya que en ciertas ocasiones existen clientes fijos cuyos pedidos no varían, agilizando procesos de ventas en situaciones conocidas.
- Generar una vista previa de los elementos seleccionados para la venta: para evitar posibles errores durante el proceso es necesaria una revisión previa de tanto los productos involucrados en el flujo de trabajo como la información de los clientes y que ambas categorías puedan ser modificadas en caso de presentarse algún inconveniente antes de realizar una venta.
- Autorización de despacho y/o producción de los elementos en la venta: una de las particularidades de KERP es su proceso de verificación de pedidos, ya que en ocasiones puede ser que no se concrete la venta o que si bien la venta se concreta es necesario un periodo de tiempo de espera que puede significar reabastecimiento de materias primas, reabastecimiento de materia, procesos que deben ser orquestados en

tiempos dados, autorizaciones de despacho, etc., por lo que es necesaria una manera de autorización de pedidos en cola.

- Selección de clientes para el despacho de pedidos: seleccionar un cliente y recuperar su información asociada (nombre, RUC o cédula de identidad) evitando llenar formularios innecesarios.
- Cálculo de costos y selección de catálogo de precios: Al ser un ERP destinado a empresas medianas/grandes, ciertas políticas generales en muchos negocios manejan catálogos de precios. Un ejemplo es el precio al por mayor, cuyos valores varían dependiendo de la cantidad a ser negociada y/o comprada. La selección de estos catálogos permite ventas a distintos tipos de clientes. También y de una manera clara se debe visualizar los cálculos de costos como el precio sin IVA, el valor del IVA en la compra, precio neto y precio con impuestos.
- Selección de productos: por medio de métodos de filtración, el catálogo presentado debe ser capaz de sintetizar los productos de interés y de manera rápida, puesto a que una búsqueda lineal tardaría mucho la cantidad de productos podría ser un problema es por eso la necesidad de una mejor manera de selección desde el inventario.
- Consulta de inventarios: Si no es una venta, por motivos varios el empleador debe ser capaz de consultar inventarios, tanto para resolver dudas de clientes como para funciones logísticas dentro de la empresa.
- Adjuntar observaciones escritas relacionadas con los procesos a realizarse: En ocasiones ciertas ventas requieren de indicadores más específicos que no suelen ser comunes, un cuadro de texto puede ser de utilidad para cuando estos casos ocurran, dando paso a una característica cualitativa que puede ser de utilidad.
- Resolver los permisos y jerarquías de usuarios mediante asignación de roles dentro del ERP: existirán cuentas con ciertos permisos adicionales para acceder a ciertos

módulos. Si bien la construcción modular es clave para un ERP, no todos los miembros deben tener acceso a todas las funcionalidades tanto por motivos de seguridad como para evitar funcionalidades innecesarias.

Los requerimientos funcionales serán parte de los módulos mencionados anteriormente. Por otra parte, la aplicación de manera general debe ser capaz de realizar lo siguiente:

- Inicio de sesión: parte del flujo de trabajo es la identificación apropiada de usuario, que en este caso será por medio de usuario y contraseña como lo solicitan los requerimientos. El inicio de sesión es clave para la resolución y asignación de roles.
- Finalizar sesión: finaliza permisos y posibles acciones por parte del usuario. El requerimiento de seguridad establece no mantener sesiones iniciadas puesto a que ciertos dispositivos serán de uso común entre varios empleadores.
- Asistente de recuperación de contraseña: servicio de recuperación de credenciales, en este caso el conflicto será resuelto por los administradores del sistema directamente.
- Interfaz gráfica acorde a una aplicación móvil: uno de los retos en la construcción de KERP es adaptar la funcionalidad a un teléfono, lo cual requiere un uso de componentes gráficos más acorde al tamaño del dispositivo y a su facilidad de uso en el dispositivo.
- Claridad en la interfaz gráfica para un flujo de trabajo intuitivo: nuevamente el espacio físico de diseño puede limitar el entendimiento del uso del aplicativo, por lo cual se debe generar un flujo de trabajo óptimo e intuitivo.
- Estilos generales acordes a la identidad de la empresa: un uso de colores apropiados y elementos de UI/UX acordes a los estilos de otros productos de la empresa es una manera de crear nuevos productos uniformes a los actualmente en uso.
- Soporte para idioma en español: por el momento el producto está previsto para uso dentro del Ecuador se desea forzar el correcto uso del español evitando hablas

coloquiales o extranjerismos dentro de la construcción de la aplicación. La RAE afirma que los extranjerismos no son, pues, rechazables en sí mismos. Es importante, sin embargo, que su incorporación responda en lo posible a nuevas necesidades expresivas (RAE, 2015). Esto no quiere decir que connotaciones propias de la información a ser usada por las empresas serán traducidas, si existen, por ejemplo, nombres de productos con extranjerismos no serán alterados de alguna manera, tan solo dentro de la construcción de la aplicación y sus componentes escritos propios del proceso de construcción.

- En su mayoría, soporte a varios dispositivos: como el sistema operativo de predominancia en los dispositivos destinados es Android, se intentará alcanzar una gran gama de dispositivos en los cuales KERP pueda funcionar sin problemas o necesidades muy específicas de hardware y/o software en la manera que sea posible.
- Icono de la aplicación: un logo de KohinorERP con los requerimientos acordes del sistema operativo.
- Splashscreen: como una manera de notificar al usuario que la aplicación está arrancando un splashscreen acorde al UI será desplegado al arranque de la aplicación.
- Diálogos de validaciones: con la capacidad para que notifiquen cualquier flujo de trabajo erróneo o cualquier error asociado al funcionamiento evitando errores en envíos de información a la base de datos o un uso incorrecto del aplicativo.

Tecnologías escogidas

Como parte de los requerimientos funcionales mencionados y también la preferencia de apegarse a un proveedor de soluciones informáticas, en este caso Microsoft, los lenguajes núcleos del proyecto serán C# junto con SQL Server, para el desarrollo del frontend y del backend. A parte de los mencionados, otras herramientas de desarrollo varias, entre

frameworks y servicios forman parte del proyecto. Las tecnologías escogidas serán divididas en dos categorías, la primera serán las herramientas y ambientes de desarrollo la segunda los paquetes NuGet que la aplicación usará.

Elemento empleado	Descripción	Versión
C#	Lenguaje de programación	8.5.12
.NET Mono	Framework de gestión de recursos multiplataforma para el desarrollo en varios sistemas operativos	3.x
Xamarin.Forms	Proyecto open source para construcción de aplicaciones multiplataforma basada en .NET	4.11.0
Andorid Emulator	Virtualización de un teléfono con Android OS	Android Oreo 8.0
Microsoft SQL Server Managment Studio	Ambiente integrado para manejo de bases de datos tipo SQL	18.0.5
Visual Studio 2019	IDE de Microsoft recomendado para el desarrollo de Xamarin	17.1.0

Tabla # 1 Frameworks y herramientas usadas en Kohinor Móvil.

Elemento empleado	Descripción	Versión	Autor
Rg.Plugin.Popup	Manejo de diálogos	2.1.0	Kirill Lyubumov
NETStandar.Library	Conjunto de funciones estándar de API's de .NET	2.2.0	Microsoft
Plugin.Toast	Notificaciones estilo toast	4.11.0	Ishrak El Andaloussi

Synfussion.Xamarin.SfDatagrid	Componente de alto rendimiento para visualización de colecciones de datos	18.4.0.46	Syncfusion Inc
System.Data.SqlClient	API escrito en C# orientado a SQL para manipulación de la base de datos desde el cliente por medio de TDS (Tabular Data Stream)	4.8.3	Microsoft
Xamarin.Essentials	API base para el funcionamiento de Xamarin	1.5.3.2	Microsoft
Xamarin.Forms	Controles de UI escritos en C#	4.8.0.1451	Microsoft

Tabla # 2 Paquetes NuGet usados.

Propuesta de solución

Una vez definidas las tecnologías y servicios adicionales para el desarrollo de Kohinor Móvil es posible dar paso a la propuesta de solución detallada. Se ha propuesto previamente una aplicación móvil, pero de manera más específica la aplicación será realizada usando Xamarin.Forms en conjunto con SQL Server bajo las propuestas mencionadas en los cuadros 1 y 2. Las propuestas que serán integradas a la construcción del proyecto por sus utilidades y/o funcionalidades mencionadas. Existiendo varias tecnologías en el mercado se debe justificar de alguna manera las elecciones realizadas siendo la más relevante el uso de Xamarin.Forms por sobre otras opciones.

¿Por qué Xamarin?

Al ser un proyecto que pretende escalar en un mercado creciente en demanda por aplicaciones móviles, centrarse en una solución mono plataforma (como lo puede ser Swift

par iOS o Java para Android) causaría que, en caso de necesitar una distribución alternativa a la escogida, es necesaria una reimplementación por completo para nuevos sistemas operativos. Xamarin permite crear proyectos multiplataforma sobre un mismo código base en C#. Si bien ciertos componentes de interfaz gráfica en ocasiones por su complejidad es necesaria una sobrecarga nativa de código, es mucho más sencillo readaptar componentes ya existentes sobre el flujo de trabajo y arquitecturas ya construidos. La facilidad de crear nuevas soluciones reusando código y apegándose a las tecnologías de Microsoft es uno de los motivos principales del uso de Xamarin. Pero el motivo de mayor peso es crear una solución compatible con soluciones ya en producción de KERP.

Pero ¿qué es realmente la compatibilidad? Una respuesta podría ser tener un código reusable que sea capaz de realizar acciones dentro de otro código. Esta definición tiene un causal que puede ser un problema; si bien el proyecto es distinto, cualquier dependencia del código a reusarse debe constar en su nuevo destino lo que significa que, a nivel de paquetes, deben existir compatibilidades absolutas. Pongamos un ejemplo sencillo, se tiene un framework cuya operabilidad consta en extraer *timestamps* completos. En dos contextos distintos, y por ejemplificar, se podría aproximar una solución con llamados al sistema para extraer la información solicitada pero también es posible realizar un llamado a un API público.

Tanto la primera como la segunda solución realizan su comedido y si dichas soluciones son implementadas en un mismo paradigma (arquitectura o lenguaje de programación) los llamados son válidos puesto a que manejan un lenguaje común pero no son compatibles si es que las condiciones base de su funcionamiento son específicos para el caso de uso. La peculiaridad de Xamarin se basa en que en el framework .NET que es una implementación de herramientas varias independientes de su plataforma de ejecución basada en API's ya estandarizadas para su uso en los frameworks que sean basados en ello. Por ende,

sus SDK más fundamentales pueden correr en variedad de dispositivos. También pose compatibilidad absoluta entre distintas arquitecturas de procesadores como x64, x86 o ARM ampliando en gran medida dicha compatibilidad por medio de paquetes NuGet.

Esto no quiere decir que todas las funcionalidades implementadas bajo un paradigma sean reutilizables en otro, sino que comparten un conjunto de operaciones fundamentales en común logrando mantener sistemas heredados con nuevas capas de presentación, como el objetivo planteado de mantener el motor de base de datos en el núcleo de funcionamiento. La figura siguiente retrata lo explicado:

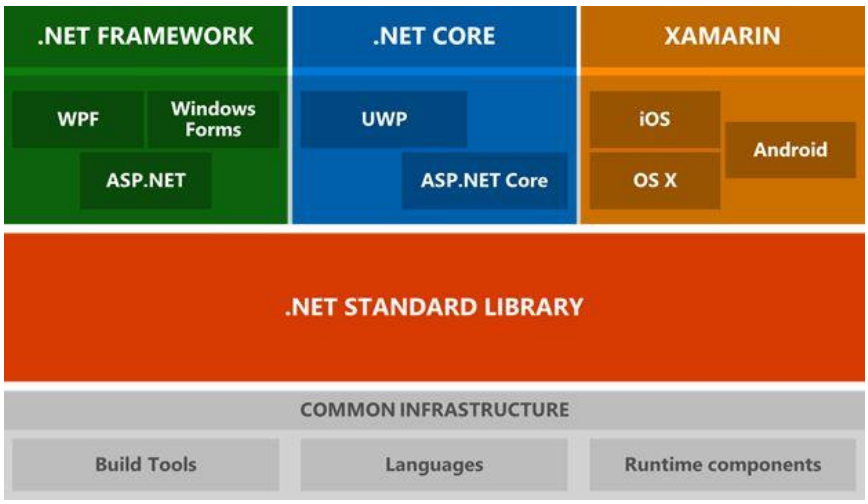

Figura # 2 Relación ente Xamarin y .NET sobre su sistema central y otros frameworks.

Al basarse todo sobre .NET el lenguaje, las herramientas y los componentes de ejecución pasan sobre un punto en común que les otorga sus funcionalidades previo a su implementación específica, creando compatibilidad base absoluta. Gracias a esto, es posible reutilizar partes críticas del código dentro de otras implementaciones sin necesidad de preocuparse por su desempeño, como lo es la interacción con la base de datos, pilar fundamental de la arquitectura SOA. Como consecuencia, operabilidades complejas son readaptables de manera sencilla.

Otro punto fuerte de usar Xamarin es el IDE en el que se desarrolla. Visual Studio es un proyecto activo que es mantenido por Microsoft, consta de un entorno integrado que soporta varios lenguajes de programación y el único para hacer Xamarin. Por esta razón, toda la documentación es entorno a *VS* eliminando cualquier dependencia de funcionalidad asociada al IDE. A parte de sus múltiples ventajas y herramientas de desarrollo y también es altamente personalizable.

Capa intermedia de comunicación

Como propuesta de solución también se encuentra la reutilización de KohinorSB, que es el conjunto operaciones para DML y DDL de la base de datos. Por ende, una capa intermedia propuesta se basa en SOA para la arquitectura de la aplicación móvil. En secciones futuras se entrará más en detalle, pero su capa de comunicación es muy específica y detallada a la hora de realizar operaciones a la base de datos. Por esto es que generalmente se adoptan a los stored proceadures como métodos de manipulación a los datos. Cabe recalcar que el uso de SQL Server proporciona medidas de seguridad como las siguientes:

- Por defecto opera con TDE(Transparent Data Encryption) lo que significa que posee encriptación por defecto de tipo AES de 256 bits.
- Permite asignación de roles para controlar las acciones de los usuarios.
- System.Data.SqlClient posee una función llamada *Always Encripted* que permite encriptar la información y revisa cualquier operación previa a la base de datos para analizar posibles riesgos. Según la documentación oficial hace que controlador determine automáticamente qué parámetros de consulta corresponden a columnas confidenciales de la base de datos (protegidas con Always Encrypted) y cifra los valores de esos parámetros antes de pasar los datos al servidor. De manera similar, el

controlador descifra de manera transparente los datos recuperados de las columnas de la base de datos cifradas en los resultados de la consulta. (Microsoft, 2019)

Support Always Encrypted	Support Always Encrypted with Secure Enclave	Target Framework	Microsoft.Data.SqlClient Version	Operating System
Yes	Yes	.NET Framework 4.6+	1.1.0+	Windows
Yes	Yes	.NET Core 3.0+	2.1.0+[1]	Windows, Linux, macOS
Yes	No	.NET Standard 2.0	2.1.0+	Windows, Linux, macOS
Yes	Yes	.NET Standard 2.1+	2.1.0+	Windows, Linux, macOS

Figura # 3 Always Ecrypted y sus versiones compatibles de .NET.

ARQUITECTURA Y DISEÑO

La arquitectura elegida para el proyecto, como su nombre lo dice, se basa en separar las funciones de un programa a manera de servicios, que si bien están interconectados no dependen el uno del otro para funcionar. Esta idea de los servicios se puede comparar con los módulos ya que cada cual realiza de manera independiente una función el en ERP y estos módulos pueden compartir información y así están relacionados. Cada servicio (o módulo) debe cumplir lo siguiente:

- Poseer un flujo de trabajo completo, lo que quiere decir que si se empieza algún proceso, el proceso debe finalizar en el mismo módulo sin necesidad de requerir a otro para completar la tarea (Ej. una venta debe completarse desde el módulo de ventas).
- Los resultados deben ser inmediatos cuando se finaliza el uso del módulo y esos resultados deben ser específicos (Ej. consultar las ventas de un vendedor particular, realizar una venta a un cliente específico, consultar la cantidad de un producto, etc.)

- Partiendo del punto anterior de forzar completar tareas no deben manejar estados, por lo que procesos inconclusos no son almacenados y tampoco es posible recuperarlos ni tampoco dependen de procesos preexistentes por lo que antes de actuar es necesario recuperar toda la información que sea necesaria.
- La información debe venir de un proveedor de datos, ningún tipo de información es obtenida por procesar la información. La lógica del negocio es la que se encarga del tratamiento de datos con esto se garantiza que las operaciones realizadas sean siempre las mismas.
- Deben ser de acoplamiento débil al sistema, que significa que el la capa de cliente y la capa de negocios mantienen una relación baja de dependencias ente ambos, y solo es necesario que sean capaces de comunicarse entre si.
- Deben ser granulares o atómicos, lo que quiere decir que la funcionalidad del módulo sea muy específica. Para lograr esta atomicidad en el nombre del módulo se debe resumir de cierta manera la funcionalidad. Como por ejemplo el módulo de ventas debe vender productos, contabilidad calcular salarios, inventario debería poder consultar precios, etc.
- No posee una implementación fija en sus operaciones (como las operaciones REST) por lo que no existen dos SOA iguales. El servicio y su construcción puede ser más largo o menor dependiendo de sus funcionalidades.
- Abstracción fuerte, intenta generar la mayor parte de información con la mínima participación del usuario para evitar errores.

Las características mencionadas anteriormente son requisitos que tomar en cuenta para la arquitectura de Kohinor Móvil. Por otra parte, si es que se cumple las características del servicio, este servicio se vuelve reusable y puede ser usado en otro servicio de manera

embebida. Este caso en particular se abordará más a detalle en la sección del módulo de ventas.

Diagrama SOA para KERP y Kohinor Móvil

Para el caso específico de la arquitectura SOA para el presente trabajo se definirán los actores de la siguiente manera:

- Capa de consumidores: implementaciones de los clientes (versión web, de escritorio y móvil).
- Enterprise Service Bus (o Canal de Comunicación Empresarial): a manera de DDL Y DML cualquier instanciación a la fuente de datos.
- Capa de proveedores: los módulos que formar parte del ERP, en estructuras individuales e independientes que comparten la fuente de datos.
- Capa de información: El núcleo KohinorSB.

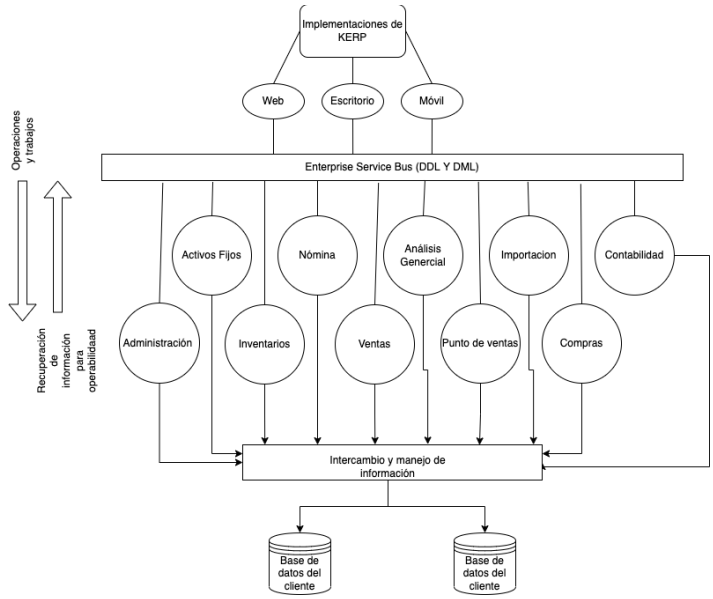

Figura # 4 Arquitectura SOA de KERP.

Para la diagramación de la arquitectura específica que la implementación en la aplicación móvil la figura 4 se toma de base y se aplican los módulos a desarrollarse, pero manteniendo la misma estructura:

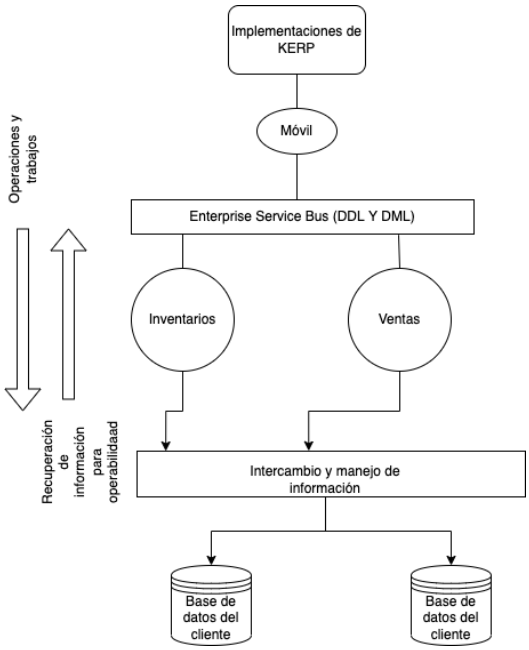

Figura # 5 Arquitectura SOA de Kohinor Móvil.

Procesamiento de credenciales e identificación de usuarios

Resolución de credenciales

El flujo operativo que se sigue dentro del ERP para el procesamiento correcto de las credenciales del usuario para la generación de su rol toma en cuenta a 4 variables que deben ser verificadas. Estas variables se procesan para generar cuatro resultados cuyos valores al mismo tiempo son usados como parámetros para realizar cualquier operación en los módulos. La presencia de tantos datos a tomar en cuenta para la operabilidad del sistema es una de las razones para una capa previa al enlace directo con la base de datos. Las credenciales actúan distinto dentro del proceso de verificación puesto a que aparte de resolver roles actúan como mecanismos de verificación para que el uso del sistema sea exclusivo de back office. A continuación, se presentan las credenciales y los tokens generadas:

Credencial	Utilidad
Usuario	Asignación de rol

Contraseña	Asignación de rol
Código empresarial	Verificación de la empresa
Dirección MAC	Verificación de la estación de trabajo

Tabla # 3 Credenciales para el inicio de sesión en KERP.

Token	Descripción
Código de usuario	Permite acceder a los detalles de quien haya realizado una operación. Récords de ventas, clientes, etc.
Código de vendedor	En ciertos casos existen distintos tipos de vendedor
Secuencia	Identificador alfanumérico con propósitos de análisis gerencia;
Código de almacén	Identificador del lugar donde se realizó la operación
Fecha de expiración	En ocasiones existen tiempos limitados para el uso del ERP

Tabla # 4 Tokens generados si el usuario es válido.

Existen muchos tokens y credenciales que escapan del control del usuario, es por eso que el ERP tiene mucho control desde el lado del administrador que es quien puede variar estos parámetros y por ende generar tokens distintos. Es flujo es el descrito por requerimientos pasados con otros clientes que tenían empleados que variaban su lugar de trabajo por motivos internos de la empresa. Una de las ventajas de esta propuesta de identificación es que permite almacenar todas las ventas de un usuario y estas ventas pueden ser consultadas por almacén y secuencia y empresa. Si bien este es el modelo propuesto, en caso que la empresa solicitante no opere como lo proponen los cuadros 4 y 5 es posible asignar valores por defecto que omiten algunos pasos del proceso. El ERP está en constante evolución y busca cumplir un alto número de requerimientos dependientes de las necesidades del cliente. El caso más complejo es el ejemplificado en los cuadros anteriores. Por otra parte, el token de fecha de expiración cumple dos propósitos los cuales se establecen según los requerimientos solicitados:

- Fecha límite para vender un grupo determinado de productos, por motivos de finales de temporadas de ventas o requerimientos de metas de ventas.
- Agentes externos con la capacidad de interactuar con el sistema por tiempo limitado. Un ejemplo es terceros que están realizando trabajos en conjunto con la empresa pero no pertenecen a ella o trabajadores a tiempo parcial.

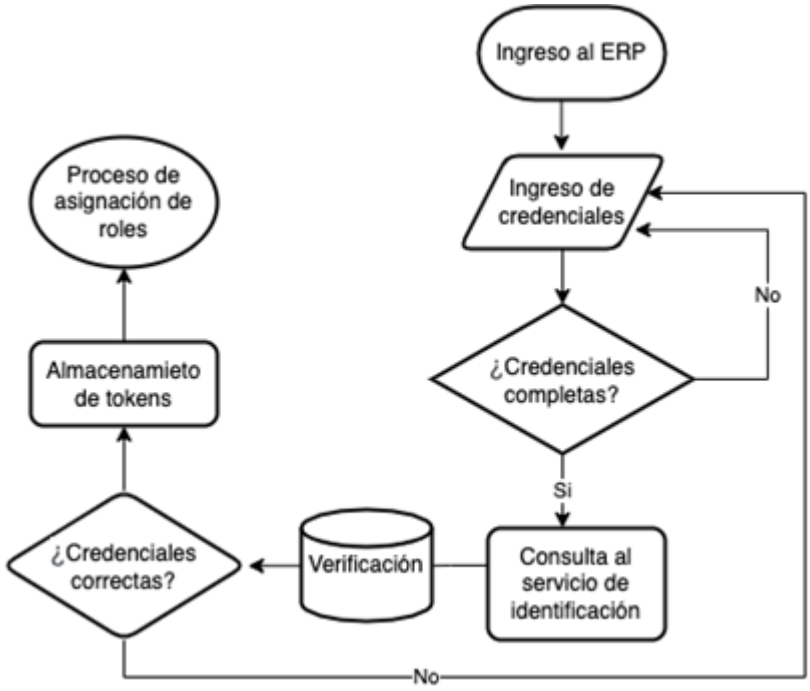

Figura # 6Diagrama de flujo de la resolución de credenciales.

Asignación de roles

A pesar de que los servicios definidos para la aplicación móvil sean menores que los totales que posee ERP se definieron los roles de usuarios que sirven para determinar la accesibilidad a los servicios previstos dependiendo de los privilegios asignados. Se categorizan en 4 distintos identificadores de tipo A, B, C y D. La tabla a continuación explica lo mencionado:

Tipo de usuario	Rol asignado	Módulos accesibles
A	Administrador	Todos los módulos

B	Agente comercial	Ventas e inventarios
C	Asesor de contabilidad	Activos fijos, contabilidad, importación, punto de venta
D	Recursos humanos	Administración, nómina, análisis gerencial

Tabla # 5 Resolución de usuarios en Kohinor Móvil.

El acceso a módulos limitados dependiendo del rol también forma parte de una medida de seguridad ya que dada la arquitectura un usuario no puede hacer uso servicios no asignados. Por cada rol se asume una tarea específica y un número dado de acciones a realizarse por medio del ESB (Enterprise Service Bus). El diagrama a continuación explica el flujo del proceso:

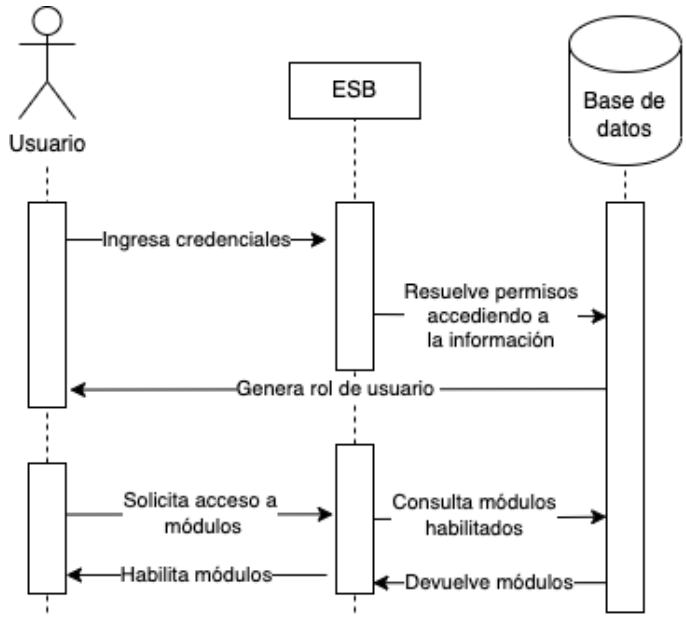

Figura # 7 Diagrama de secuencia de la resolución de credenciales.

Módulo de ventas

La resolución de roles explicada anteriormente termina su participación cuando se devuelven los módulos que el usuario puede solicitar, por ende, según el cuadro 6 los usuarios de tipo A y B (administrador y agente comercial) pueden usarlo. Dentro del módulo

de ventas existen dos posibles casos de uso que emplean el mismo servicio e incluso su punto final es la misma sección de la base de datos. A continuación, se presenta un diagrama:

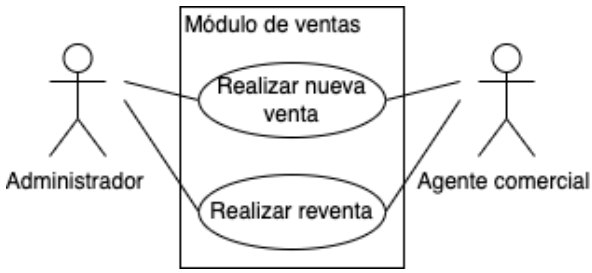

Figura # 8 Casos de uso en el módulo de ventas.

La separación de la figura 8 a pesar de ser el mismo proceso en general (más adelante se detallará) existe más para implementar una funcionalidad para tomar la información de algún pedido realizado anteriormente y basarse en los elementos vendidos para crear uno nuevo actualizando cierta información. Esto se debe a que en ciertas ocasiones los clientes llevan los mismos artículos (ej. proveedores, materias primas). También porque permite agilizar procesos siempre que exista la información almacenada en algún lugar. El módulo de ventas, ya que puede realizar una reventa también es capaz de consultar las ventas del usuario que está usando la aplicación.

Por otra parte, el núcleo de SQL Server del ERP en este módulo se comunica con la base de datos tanto desde la dirección del usuario al ERP y viceversa atomizando toda la información en un objeto llamado Pedido el cual es ensamblado desde el lado del cliente pero se lo reinterpreta al llegar a la base de datos con el fin de que en caso de ser necesario modificar el comportamiento no tenga que refactorizarse todo el canal de comunicación. Por otra parte, para ciertos casos de uso existen campos con información por defecto ya que la lógica del negocio no necesita que por lo general son los varios precios al mismo producto. El objeto Encabezado por otra parte contiene metadatos del proceso y suele ser mutado para la reventa. El objeto Venta contiene a ambos.

Objeto Pedido

En este objeto se modela la lógica de la venta que se realiza ya que su función principal es alimentar el módulo de compras solo con información relacionada a la variación del inventario, información de costes, ítems despachados, etc. Esto con el fin de separar información del núcleo del ERP para llevar una mejor organización de la información y también por funcionalidades externas al módulo que requieren la información que se genera. Por otra parte, este objeto es hijo de la clase DataTable que forma parte de System.Data. Es decir, cualquier pedido se abstrae como un DataTable y dada la compatibilidad de los servicios a usarse es posible enviar mediante el ESB el objeto entero y no datos individuales que consten como argumentos para la interacción con la fuente de información principal y es allí donde otros procesos pueden ejecutarse, estos procesos son ocultos en la capa de presentación y mayoritariamente se centran en actualizar datos como secuencias e identificaciones. A continuación, se presenta el diagrama de clase:

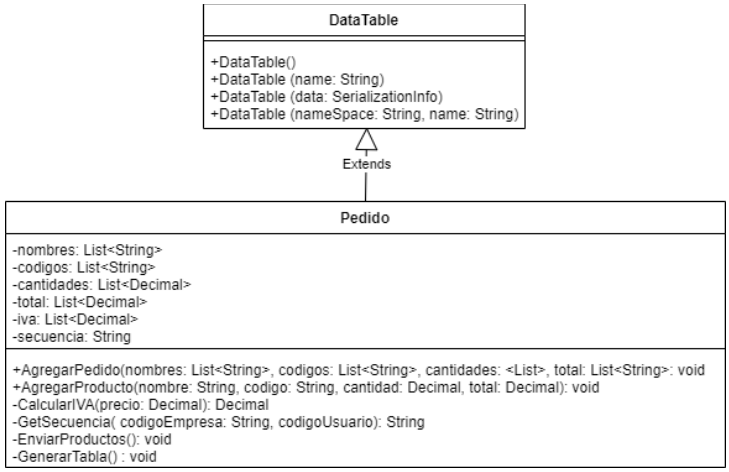

Figura # 9 Diagrama de clase de Pedido.

Es importante mencionar que dentro del objeto de la figura 9 no existe una variable que almacene la tabla, en primer lugar, porque al heredar directamente es una tabla pero también la herencia en esta clase permite crear una arquitectura libre de estados ya que es responsabilidad de la propia instancia enviar los productos a la base de datos. Esto quiere

decir que es posible que exista un pedido que no tiene asignada una venta con el fin de poder crear una preorden también que ya conste como una venta (una venta se considera cuando se le asigna un numero de secuencia único e irrepetible) El motivo de permitir este comportamiento nace a raíz de negocios que suelen manejar varios pedidos por facturas separadas pero al mismo nombre (ej. una empresa realiza varios pedidos para distintas sucursales). Por otra parte, tener separada la información en clases distintas también permite que el pedido pueda actuar como una prefactura con todos los datos calculados. Este comportamiento será analizado más adelante cuando se presente la interfaz de usuario.

La clase pedido modela todo aquello que pueda considerarse mutable ya que puede variar entre objetos. A pesar de este comportamiento la tabla de datos que se envía como respuesta posee muchos otros campos, algunos que cuando se necesita se los deja como valores predeterminados y otros que forman parte de otros identificadores específicamente los que se mencionan en el cuadro 4 sobre los tokens de la resolución de credenciales. La clase, como se dijo anteriormente, sirve para generar la información por lo que su contraparte en la base de datos puede tener otra estructura a conveniencia de la lógica de negocios. Los campos adicionales. Estos campos son los siguientes:

Abreviación	Nombre Completo
codemp	Código de empresa
tiptra	Tipo de transacción
numtra	Número de transacción
codalm	Código de almacén
numren	Número de renglón
numite	Número de item
codart	Código del artículo

nomart	Nombre del artículo
coduni	Código unitario
cantod	Cantidad solicitada
preuni	Precio unitario
codiva	Código de IVA
totrem	Total de remesas
ubifis	Ubicación física

Tabla # 6 Nombres de columnas dentro de la tabla de Pedido.

Existen muchos otros campos que no son mencionados dentro de los atributos de clase, algunos que se consideran variables globales (como el código de vendedor, empresa, almacén, sucursal y secuencia) que son procesadas en otro lugar distinto a los módulos. También existen muchas otras variables que son resultados de procesamiento de los datos dentro del mismo objeto y estos pueden ser resultados de cálculo como totales o valores con impuestos agregados y se opta por una solución de cálculo porque pueden ser datos que cambien.

También otras variables extraen datos de otros objetos para identificar algunos campos como la información del código de artículo y su respectivo nombre. Otros valores son resultados de mapeos como el nombre del cliente a partir de su identificación y del cuadro 7 los campos de codemp, numtra, codalm y sersec son llaves de identificación del pedido. También el tener variables que no sean visibles al usuario permite evitar errores y el ingreso de muchos datos manuales. Por lo general todos estos datos adicionales son definidos específicamente para casos de uso de KERP y pueden llevar valores por defecto si es necesario.

Objeto Encabezado

El objetivo de este objeto es de recopilar otros datos no relacionados a productos sino información metadatos de los actores envueltos en el caso de uso y de identificadores del pedido. El encabezado puede agregarse a un pedido ya existente como también crearse desde cero. Esta característica es la que permite realizar una reventa en el caso del módulo de ventas. Existen otros comportamientos en distintos módulos que aprovechan la utilidad de este objeto, pero aquellas funcionalidades escapan del contexto del trabajo actual. Quienes pueden ser parte de este objeto son un agente de ventas y un cliente, junto a un identificador de cierto pedido. De la misma manera que el anterior objeto, cualquier instancia de Encabezado extiende de un DataTable. Entre ambos objetos se comparte el número de transacción ya que al ser procesados por el núcleo del ERP ciertos campos son enviados a lugares distintos y también para poder realizar otro tipo de consultas en distintos módulos. A continuación, se presenta el diagrama de clase:

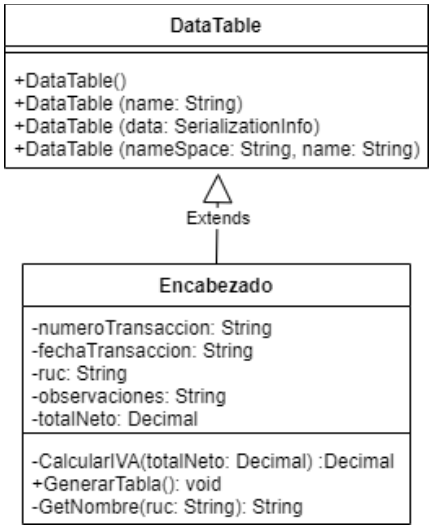

Figura # 10 Diagrama de clase de Encabezado.

En este caso existen datos provenientes de fuentes distintas, por una parte, se identifica a la empresa con el número de transacción y al cliente con el RUC, esto para poder asociar el pedido con dos identificadores. El encabezado engloba información de todos

quienes participan en el pedido y también el resumen de precios. Se realiza esto porque un componente en la interfaz de usuario necesita estos datos porque los usa como criterio de ordenamiento. Aunque sea una clase de menor tamaño con respecto a la anterior su DataTable resultante posee bastantes datos adicionales a las propiedades de la clase:

Abreviación	Nombre completo
codemp	Código de empresa
tiptra	Tipo de transacción
numtra	Número de transacción
codalm	Código de almacén
codsuc	Código de sucursal
sersec	Serie de secuencia
fectra	Fecha de transacción
fecven	Fecha de vencimiento
clixcx	RUC del cliente
nomcxc	Nombre del cliente
listpre	Lista de precios (catálogo del producto)
codcen	Código centralizado
nomcen	Nombre centralizado
codven	Código de vendedor
codeje	Código ejecutivo
nomeje	Nombre ejecutivo
fecent	Fecha de entrega
forpag	Forma de pago
numpro	Numero de proveedor

observ	Observaciones
totnet	Total neto
totiva	Total con IVA

Tabla # 7 Campos de la tabla de Encabezado.

Los campos variados que podría llegar a tener este objeto pueden cubrir una gran cantidad de lógica de negocios y de manera general suele ser en donde más modificaciones se realizan cuando el ERP es solicitado por los clientes. También permite muchas formas de realizar consultas a la base de datos y esto se usa en otros módulos del ERP. Cuando se realiza una reventa los datos que cambian son los siguientes:

- Número de transacción.
- Serie de secuencia.
- Fecha de transacción.

En si todos los datos pueden modificarse en la reventa y hasta puede convertirse en una venta completamente nueva, es por eso que ambos casos de uso desembocan en el mismo procesamiento y también por seguir las reglas de SOA no existe un estado que determine cuando es o no es una venta nueva. Los casos de uso para este objeto se orientan más a una funcionalidad para agilizar el flujo de la aplicación. Los datos de la lista siempre varían al mismo tiempo.

Objeto Venta

Los objetos anteriores por su número de campos pueden ser más complicados de usar cuando un negocio no maneja tantas funciones. Para procesos más sencillos que no pierdan el potencial de consulta de datos del ERP se usa el objeto Venta que incluye a Pedido y Encabezado. Sus atributos de clase son una combinación de datos tanto de Pedido como de Encabezado, así como el número de transacción. A diferencia de los otros casos no extiende

de un DataTable porque es el objeto que se encarga de la transmisión hacia la base de datos. A continuación, su diagrama de clase:

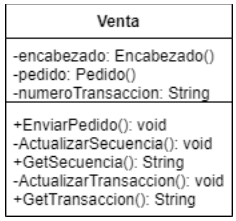

Figura # 11 Diagrama de clase de Venta.

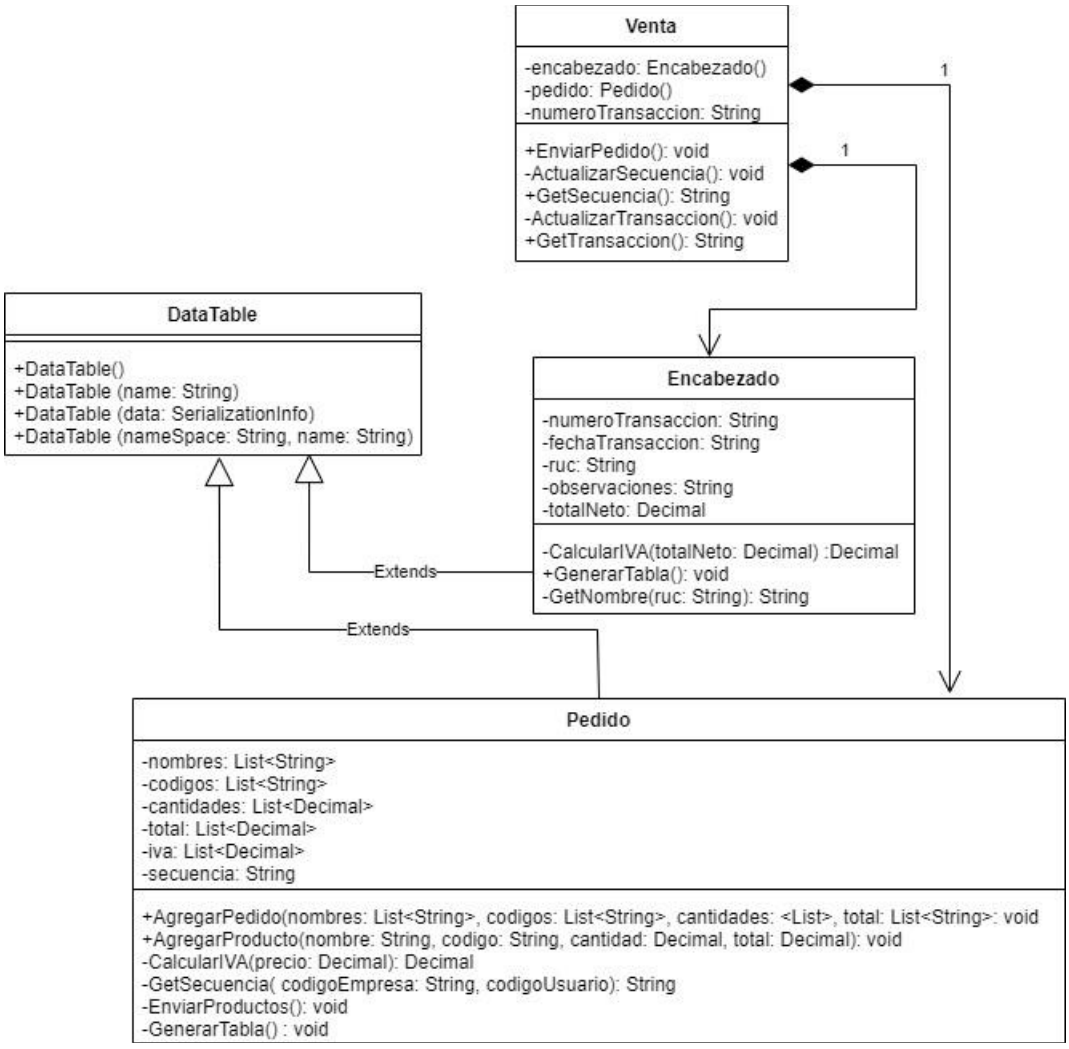

Figura # 12 Diagrama de clases del módulo de ventas y su relación de objetos.

Casos de uso

Como fue mencionado anteriormente ambos casos de uso comparten el mismo proceso, por lo cual diferencian en la toma de decisión del usuario. Aunque sigan esos

comportamientos ambos casos uso crean el objeto Venta de maneras distintas. Si bien las combinaciones de campos que pueden marse siguiendo los cuadros 7, 8 y nueve pueden brindar casos de uso más amplios se presentará como es el flujo de trabajo mínimo que el usuario debe seguir en la aplicación Kohinor Móvil para completar una venta o una reventa. Procesos relacionados con la validación de la información para controlar ingresos de datos erróneos, procesos incompletos o cualquier posible error humano serán abordados más adelante cuando se entre en detalle sobre la capa de presentación. Por el momento y para la diagramación se asume lo siguiente:

- Los casos de uso se completan de inicio a fin.
- Cualquier información necesaria será información válida.

La interacción directa del usuario y los posibles errores dentro del ERP en general son datos incompletos o faltantes. Por otra parte, y como mecanismo de control adicional la interacción con la base de datos también puede devolver errores cuando no se cumplen los requisitos necesarios para poblar las tablas de información. Se presenta a continuación el diagrama de actividades:

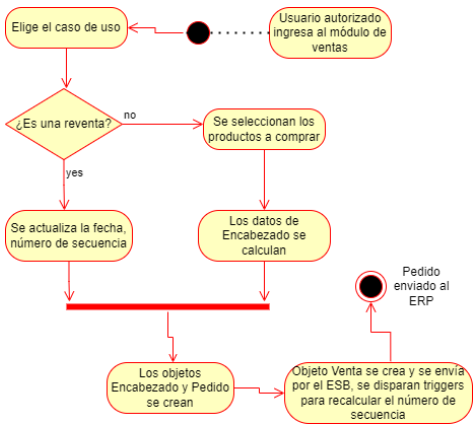

Figura # 13 Diagrama de actividades del módulo de Ventas.

Módulo de Inventario

Para la aplicación móvil se consideró una variación del módulo de inventario que esté acoplado directamente dentro del módulo de ventas. Este acople no significa que la estructura

de cualquier cosa dentro de ventas cambie dentro del núcleo del ERP pero coexisten dentro de la misma sección en la capa de presentación y se comunican entre ellos. La comunicación de ambos módulos también causa que ciertas acciones se ejecuten automáticamente (ej., que se reduzca el stock de un producto) y sucede después que el ESB entregue la información al núcleo del ERP. Sirve para dos propósitos, el primero como una herramienta de consulta de stocks con más detalle que solo informar al usuario cantidades y como proveedor de datos para el objeto Pedido. Como se lo implementa dentro del otro módulo los usuarios que tienen acceso desde la aplicación móvil son usuarios de tipo A (administrador) y B (agente comercial):

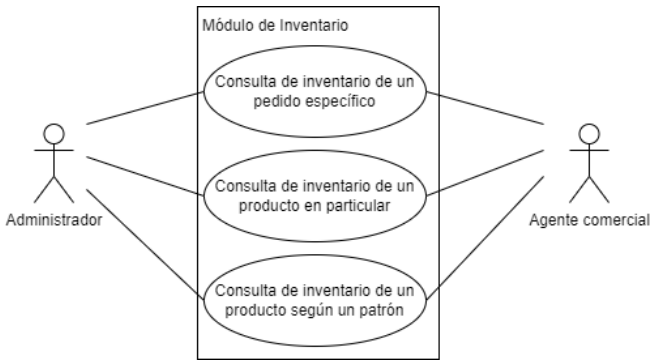

Figura # 14 Diagrama de actividades del módulo de Inventario.

Los casos de uso en la figura 14 también generan el mismo resultado solo que esta vez ese resultado puede devolver solo un valor o múltiples. La consulta realizada puede ser exportada al otro módulo y para Kohinor Móvil la comunicación entre ambos se usa por medio de protocolos de Messaging Center propios de Xamarin. Depende que exista un usuario registrado en la base de datos porque dependiendo del usuario los resultados de búsqueda varían. Esto no quiere decir que por cada usuario el ERP tenga un inventario en específico pero el usuario define la lista de precios de los resultados que desde ahora será llamada catálogo. Por esta razón se lo incorpora dentro de otro módulo que le puede proveer la información que necesita para funcionar.

Identificador de listas de usuario	Catálogo de precios
L-001	1, 2
L-002	2, 4
L-003	1, 3
L-004	2, 3

Tabla # 8 Catálogos de precios según atributos de usuario.

En caso de que el cliente y la lógica de su negocio no maneje distintos catálogos de precio se le asigna primer identificador (L-001) y al segundo catálogo se le asignan valores por defecto. El cuadro 10 permite al cliente poder manejar precios distintos sobre el mismo producto muchas veces porque se suelen vender el mismo producto en distintos estados (nuevo o usado) y también para empresas que manejen precios al por mayor y al por menor. El objeto que maneja este módulo lleva de nombre *Consulta* y su diagrama de clase es el siguiente:

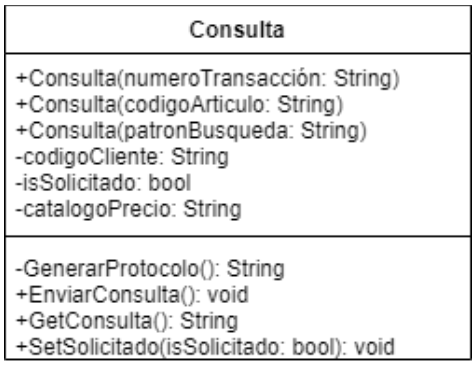

Figura # 15 Diagrama de clase de Consulta.

La clase posee tres constructores para resolver la propuesta de la figura 14 con tres propiedades e identificaciones que sus funciones son las listadas a continuación:

- codigoCliente: el identificador que ayuda a resolver qué catálogo de precio usar.

- isSolicitado: Si es que se pretende alterar el stock del inventario esta variable es verdadera y se usa cuando se genera el protocolo de comunicación con el otro módulo para resolver la interfaz de usuario y se visibilizarían en la aplicación móvil.
- catalogoPrecio: la lista de precios que será mostrada en la consulta y depende del cliente.
- GenerarProtocolo(): para usa el MessagingCenter de Xamarin se establece un protocolo que identifica las propiedades del resultado de la consulta en el cual se incluye: nombre y código del artículo, existencia, si es que su stock será alterado y su precio.
- GetConsulta(): devuelve el protocolo y los resultados SetSolicitado(): identificar al producto para modificarlo.

Caso de uso: inventario de un pedido

Esta funcionalidad se usa en conjunto cuando se realiza una reventa o si se quiere consultar una venta hecha en el pasado, extrae la información usando como llave el número de transacción.

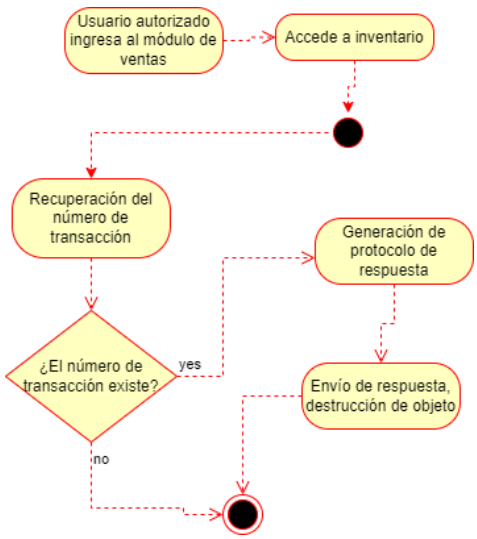

Figura # 16 Diagrama de actividades de consulta de inventario de un pedido.

Caso de uso: inventario de un producto

Con este caso de uso es posible realizar una consulta completa sobre los datos de inventario de un producto, su stock, nombre y sucursal. Viene incorporado en el módulo de ventas y también se usa para crear pedidos.

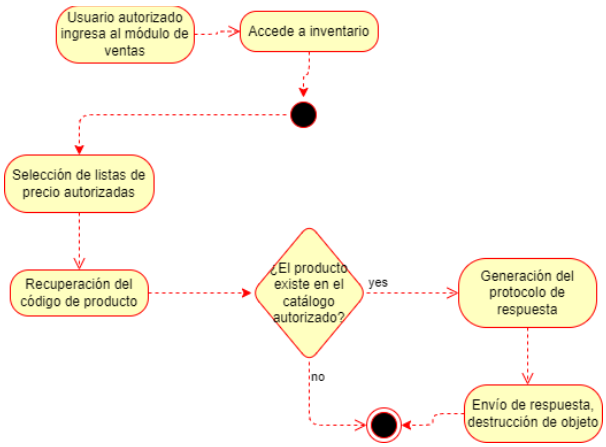

Figura # 17 Diagrama de actividades de consulta de inventario de un producto.

Caso de uso: consulta por patrón

Para evitar que el usuario busque manualmente un producto, sobre todo en inventarios largos se implementa esta funcionalidad para filtrar los resultados y agilizar el proceso de venta. Cuando se envía la respuesta dicha respuesta puede participar en el caso de uso de la sección anterior

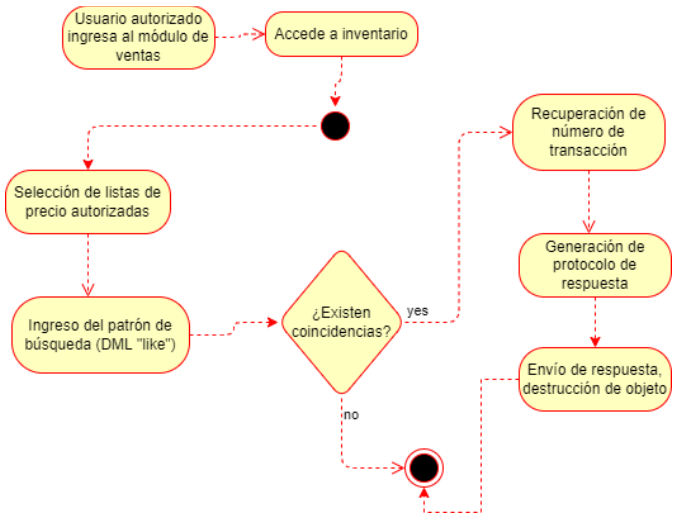

Figura # 18 Diagrama de actividades de consulta por patrón.

IMPLEMENTACIÓN

Como se mencionó anteriormente, la implementación de la arquitectura y diseño se basa en una reingeniería de KERP para dispositivos móviles. Generalmente una arquitectura popular podría ser la siguiente:

- Capa de presentación: lo relacionado al desarrollo de frontend usando Xamarin Forms y su objetivo es de mostrar y/o almacenar datos para el procesamiento.
- Capa de aplicación: se encarga de la verificación y control de los datos generalmente que ingresa o interactúa el usuario. Aquí yace la lógica del procesamiento de la presentación.
- Capa de datos: es donde se almacena y se gestiona la información procesada por los niveles anteriores y sirve como fuente primaria para la aplicación en general.

Kohinor Móvil implementa estas capas con una variación. Por motivos externos al trabajo y por implementaciones anteriores existe un procesamiento de cierta manera distribuida entre los niveles de aplicación y de datos, puesto a que el ERP en su nivel de datos se encarga de algunas cosas que la capa de aplicación debería hacer. Uno de los motivos de este comportamiento es la arquitectura SOA, puesto a que mucha de esta información se replica en varios módulos por lo que si se intentara acceder al nivel de datos de manera eficiente se necesitaría mucho código que en ocasiones no tiene que ver con el flujo de trabajo. Pero existen casos de uso en los que la aplicación realiza procesamientos más detallados de información y después de aquello ingresa la información a la fuente de datos sin pasos intermedios.

También muchas interacciones a la base de datos disparan triggers y varían información sin control

del usuario y sin control de la capa de presentación, todo a criterio del backend por lo cual realmente no existe una capa de datos en la que se manipule con facilidad la información, sino una fuente de datos que acepta llamadas muy específicas. Es por eso por lo que se usan solo dos capas, presentación para el frontend y aplicación, que se comunica con la fuente de datos por stored procedures limitados e independientes que en ciertos casos realizan otras acciones de ser necesario.

Capa de aplicación

Esta capa cumple dos funciones, controla el flujo de trabajo y la información del usuario para un correcto uso de la aplicación y también gestiona información generada por el usuario hacia la base de datos y viceversa. Ya que Xamarin fuerza al usuario a seguir un patrón MVC, es el controlador. El modelo es directamente la fuente de información y las llamadas a los stored procedures existentes. Como fue mencionado en la sección anterior, la capa de aplicación trabaja en conjunto con la base de datos para la verificación de los datos y es porque hay algunas funcionalidades adicionales fuera del módulo planteado que necesitan de este control directamente desde la base de datos. Por ejemplo, existen casos de uso donde un vendedor no puede vender un producto específico. El flujo de trabajo no varía, puede crear una nueva orden e intentar enviarla, pero antes de modificar la base de datos se dispara un trigger que verifica si el vendedor puede realizar la venta del producto y autoriza esa venta, caso contrario se dispara otro trigger para notificar que se intentó una venta fallida, pero autoriza la venta de los productos que estén autorizados.

Por lo general este caso de uso suele ser extendido al módulo de producción y contabilidad además de ser una solución que evita procesamiento desde la aplicación y entrega cualquier procesamiento pesado a la base de datos que puede actuar de manera más rápida. La aplicación en cambio verifica errores tipográficos, control de campos vacíos o

inválidos, notificaciones en casos de errores o usos incorrectos y la creación de formatos correctos para enviar la información.

Estructura del proyecto de Xamarin Forms

Xamarin Forms provee un punto de entrada a la aplicación por medio de un archivo por defecto llamado App. Desde aqui se pueden establecer propiedades globales que sean distribuidas a cualquier parte de la aplicación, puesto a que App contiene a todo. Para este caso, se hereda de la superclase Application por medio de una partial class, propia de C#. Pero Application hereda propiedades de la clase abstracta Element que a la vez hereda de Bindable Object. La clase padre de toda la jerarquía de Xamarin proporciona un mecanismo mediante el cual los desarrolladores de aplicaciones pueden propagar los cambios que se realizan en los datos de un objeto a otro, habilitando la validación, la coerción de tipos y un sistema de eventos y es una propiedad enlazable, es decir si esta se actualiza una vez y el cambio se refleja en todos los estados. Junto a esto, se sobrecargan tres funciones que manejan el estado global de la aplicación:

- onStart: se llama cuando inicia la aplicación desde cero.
- onSleep: cuando se detiene la aplicación, pero sigue en memoria.
- onResume: al reanudar la aplicación.
- onClose: cuando se cierra la aplicación y se elimina de memoria.

Junto a esto se implementa un constructor sin argumentos que también puede ejecutar código al crearse el objeto. Entonces, ¿en qué se diferencia onStart de la llamada del constructor? (el mismo caso que onClose y el destructor) Xamarin genera código nativo para su carácter multiplataforma pero algunos casos requieren procesos específicos para el sistema operativo. Un caso muy común es el render personalizado de componentes gráficos que requieren tratamiento para cada plataforma.

Entonces el arranque o cierre de la aplicación toma en cuenta la acción y la plataforma, por lo que onStart responde de manera general mientras que el constructor es más específico. Incluso cuando el proyecto no se lo crea como multiplataforma, este comportamiento continúa puesto a que el código nativo es una transpilación de código C#. Esta sección en concreto se usa para registrar la licencia de uso de Syncfussion (cuadro 2).

Este punto de entrada en el constructor redirige directamente a otra página que, a diferencia de App, su código XAML (eXtensible Application Markup) permite crear componentes gráficos. Este lenguaje de marcado propio de Xamarin se caracteriza por permitir la construcción de la interfaz a manera de XML, pero al mismo tiempo representa a una clase y permite acceder a métodos y propiedades para enlazarlo con componentes gráficos. Los estilos se aplican directamente como propiedades XAML o creando temas que se los establece en el inicio de la aplicación.

Dentro de la definición de la interfaz se define la clase a la que pertenece y solo puede ser una, entonces para cada vista existe una sola clase que puede acceder a sus componentes gráficos causando que el ciclo de vida de las variables se limite a la existencia de la instancia. Este comportamiento obliga al usuario a seguir un patrón MVC. Para el manejo de vistas, Xamarin se basa en una superclase llamada Page. De esta clase hereda la subclase Navigation Page, que implementa navegación y una herramienta de router para cambiar la vista y finalmente todas las vistas heredan directamente de Content Page. Para todas las vistas de Kohinor Móvil se sigue esta estructura y modelo. También se establece que la compilación en general se base en XamlCompilation, una herramienta que permite capturar errores en compilación para evitar que XAML genere errores en ejecución. Xamarin de manera general hace uso de un método de renderización llamado Initialize Component que resuelve todo lo anteriormente mencionado. A continuación, se presenta el diagrama de clases:

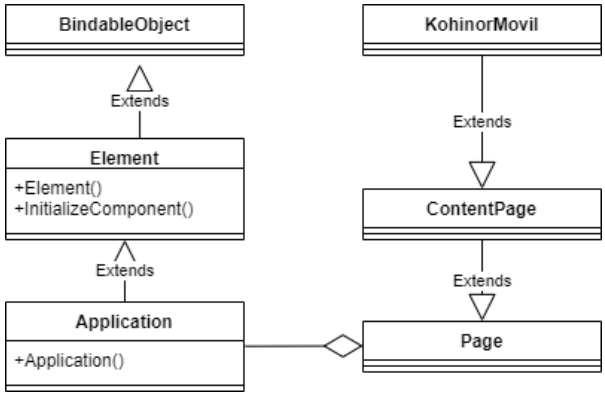

Figura # 19 Diagrama de clase de la capa de aplicación.

Sintetizando todo lo anterior, el modelo de la capa de presentación para cada vista debe cumplir siguiente:

- Debe heredar obligatoriamente a Content Page o a sus clases superiores. Seguir el modelo MVC.
- Debe tener un archivo XAML con su controlador de clase designado. Compartir App y sus configuraciones.
- Implementar XamlCompilation.
- Llamar a InitializeComponent en su constructor.

Gestor de información compartida

En el cuadro 4 se listan unos tokens que dentro del ERP cumplen varias funciones, desde comandar ventas hasta consulta de historiales o productos. Incluso estas llaves se usan para extraer información ajena a los módulos planteados. Al generarse todo eso durante el proceso de resolución de credenciales y asignación de roles se hace uso de un gestor de información que sea persistente durante la sesión, para poder acceder a esos datos sin tener que comunicarse con la fuente de información en múltiples ocasiones. Dentro del diseño de la capa de implementación a parte de las vistas existe una clase que contiene todo lo necesario para recuperar estos tokens ya generados previamente. Esta implementación sigue el patrón

de diseño Singleton. Esta implementación adapta una variante conocida como double check lock que se caracteriza por lo siguiente:

- Posee una verificación adicional por medio de la función propia de C# lock cuyo objetivo es verificar instancias en otros hilos y sincronizar el singleton.
- Es thread-safe pero la manera en la que logra esta característica puede ser costosa en términos de tiempo.
- Es un objeto serializado.

Aunque es costosa en tiempo esto permite manejar de una manera sincronizada los datos que posee. Ya que hay ciertas navegaciones asíncronas para mejorar el desempeño de la aplicación es necesario un acceso controlado al gestor de información. Se almacena lo siguiente:

- Código de usuario.
- Código de vendedor.
- Número de secuencia.
- Código de almacén.
- Instancia de la conexión SQL (creación, recuperación y eliminación).

Comunicación entre vistas

El gestor de información por sus mecanismos de seguridad es más lento, por lo que para crear un canal de comunicación para conectar pantallas distintas o enviar información la aplicación emplea a la clase Messaging Center. Sigue un patrón conocido como editor-subscriptor en donde un editor envía mensajes sin conocer a los receptores quieres pueden elegir suscribirse al canal de comunicación para interceptar los mensajes. Este canal es persistente, por lo que su ciclo de vida es independiente a la vista y siempre se puede recuperar la información. Se crea al iniciar la aplicación y se destruye al finalizarla.

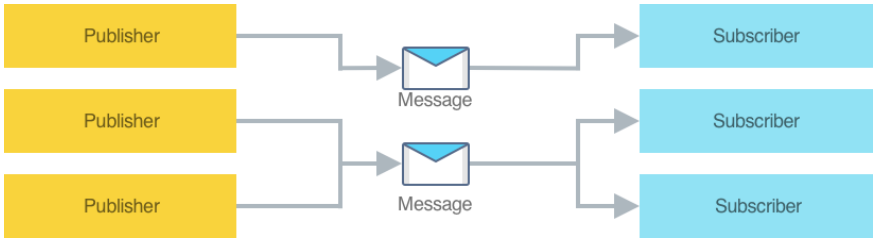

Figura # 20 Arquitectura de la clase Messaging Center.

Kohinor Móvil no establece una regla fija sobre esta comunicación, razón por la cual su uso depende de la necesidad de hacerlo, por lo que en ciertas vistas puede existir esta implementación como otros casos en los que no. Este tipo de mensajería puede ser muy amplio, incluso puede usar objetos, pero para propósitos del proyecto se usan cadenas de texto con la información necesaria. Esta comunicación sigue un protocolo específico cuando ocurre. Los casos son los siguientes:

Editor	Subscriptores	Funcionalidad
NuevosItems	NuevoPedido.cs, Inventario.cs	Enviar protocolo con nuevos elementos
ClienteElegido	NuevoPedido.cs, Clientes.cs	Enviar protocolo con información del cliente
HabilitarInventario	NuevoPedido.cs, Inventario.cs	Habilita los inventarios cuando se elige el catálogo

Tabla # 9 Relación de editores-subscriptores

Editor	Protocolo
NuevosItems	|producto:precio:codigo|
ClienteElegido	cliente:ruc
HabilitarInventario	inventarioA:inventarioB

Tabla # 10 Protocolos de MessaginCenter.

Los protocolos son resueltos por posiciones fijas, con el propósito de forzar el flujo de trabajo y la información generada.

Interacción con la base de datos

Por medio de stored proceadures ya definidos previamente, la aplicación puede comunicarse con la base de datos. Esta interacción en ocasiones dispara triggers por necesidad de otros módulos o para actualizar valores por lo que esta interacción es indirecta. Como mecanismo de seguridad todas las instancias que se abren en producción tienen bloqueadas la posibilidad de ejecutar querys. De esta manera se controla las acciones del usuario, automatiza ciertas acciones y se reduce la cantidad de código desde el lado del cliente. Para el manejo de las credenciales de conexión y su seguridad Xamarin provee una herramienta llamada App Center propia de Visual Studio. Es un gestor de acciones que funciona en la nube y se lo asocia con uno o varios proyectos compatibles de .NET y sirve para automatizar el ciclo de vida de una aplicación. Es una microherramienta para el lado de DevOps. Entre sus varias funciones permite que la instancia de la aplicación haga uso de variables de ambiente. Soporta cifrado para brindar un mecanismo de seguridad adicional.

Variable	Identificador
Usuario de la base de datos	ID_
Contraseña	CONTRASENA_
Dirección del servidor	FUENTE_DATOS_
Catálogo del cliente	CATALOGO_
Query de conexión	CONEXION_

Tabla # 11 Variables de ambiente.

Los stored procedures empleados dentro de la aplicación se encargan tanto de la resolución de credenciales y del funcionamiento del módulo de ventas. La comunicación entre la base de datos con capa del cliente en su mayoría es por medio de la clase DataTable que es soportada tanto en C como una variante en su implementación de SQL Server por lo

que para grandes manipulaciones de datos como son los pedidos, los parámetros son instancias de esta clase.

Capa de presentación

Kohinor Móvil a diferencia de sus otras implementaciones posee un espacio de visualización mucho menor, por lo que la experiencia de usuario debe adaptarse al dispositivo. La capa de presentación aborda las decisiones y el diseño de la nueva presentación del ERP. Esta sección es la parte del proyecto más delicada ya que debe resumir la complejidad de todo el sistema en elementos interactivos para el usuario. Esta sección aborda los siguientes temas:

- Estructura de la solución.
- Recursos generales.
- Vistas.
- Flujo de trabajo.

Estructura de la solución

Una solución es un concepto propio del desarrollo .NET que propone un espacio de trabajo donde se pueden combinar varios proyectos relacionados pero independientes que pueden compartir código en común. En este caso, la solución contiene tres secciones internas:

- Código multiplataforma compartido para iOS y Android (C#).
- Código nativo de iOS (Swift).
- Código nativo de Android (Java).

Toda la lógica de la aplicación se encuentra escrita en C#, pero las secciones de código nativo se usan para resolver problemas muy propios del sistema operativo y para personalizar componentes complejos. También para las configuraciones de los recursos y accesos a los archivos de configuraciones Info.plist y AndroidManifest.xml. Es importante mencionar que

se genera un directorio completo de las solucione nativas. Aunque en concreto Kohinor Móvil está destinado a Android para trabajo futuro se añadió que sea multiplataforma.

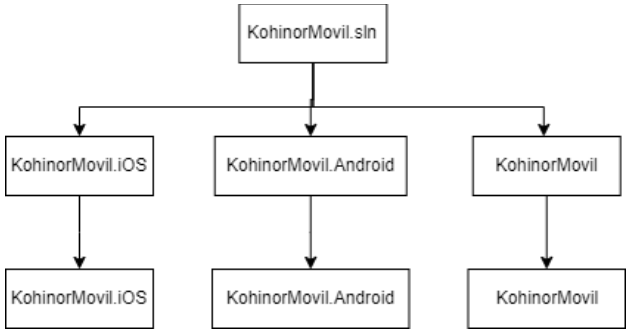

Figura # 21 Distribución de directorios para la solución Kohinor Móvil.

Durante el desarrollo de la aplicación los directorios nativos no se modificaron a excepción de añadir los contenidos multimedia a las carpetas correctas. Dentro de la solución multiplataforma se sigue el modelo de la sección anterior (Estructura del proyecto de Xamarin Forms).

Recursos compartidos

A pesar de que el ERP se distribuye individualmente y con los requerimientos solicitados por los clientes, los componentes gráficos se mantienen con el fin de evitar apropiación de la empresa sobre el producto. No todo es reusable, pero existen componentes ya de uso general como logos, imágenes y colores con el fin de apegarse a un estilo de diseño. Los recursos son presentados a continuación.

Atributo	Valor
Tipo de letra	Regular y negrillas
Fuente	Roboto
Tamaño	12 pts
Colores	#000000, #00445e y #5D5D5D

Tabla # 12 Propiedades de visualización de texto.

Figura # 22 Imágenes de los módulos y logos compartidos.

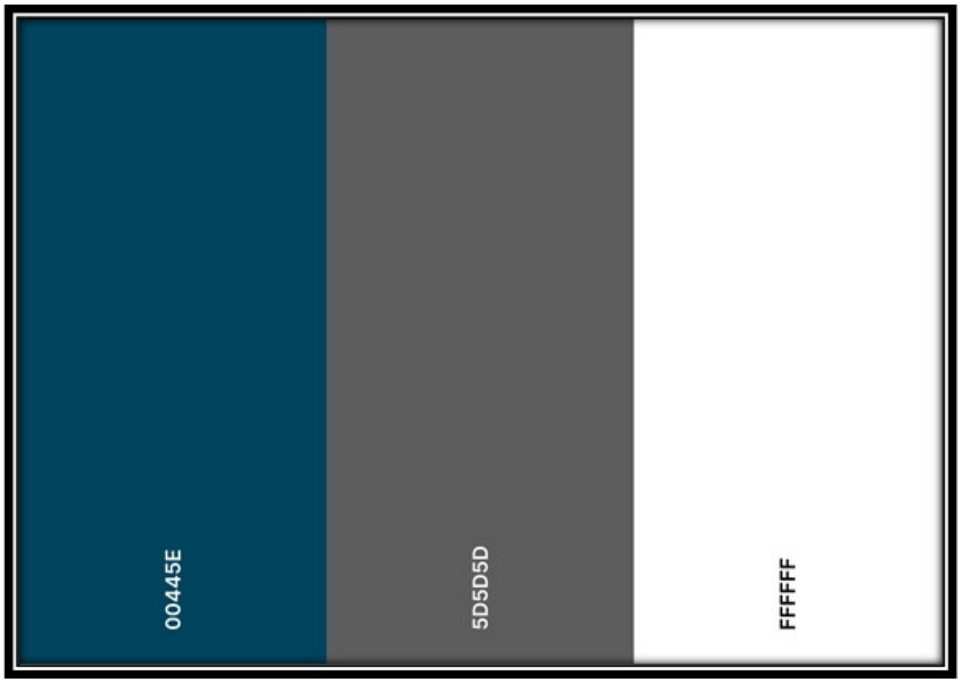

Figura # 23 Paleta de colores.

Vistas

Siguiendo la estructura de la sección *Estructura del proyecto de Xamarin Forms* las vistas son las siguientes:

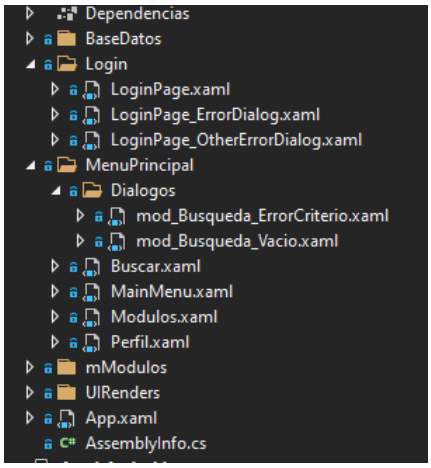

Figura # 24 Vistas generales de Kohinor Móvil.

En la figura 24 se encuentran las vistas de la aplicación excluyendo al módulo de ventas. Para cada archivo XAML existe su correspondiente controlador (o también conocido como Code Behind) de tipo C#.

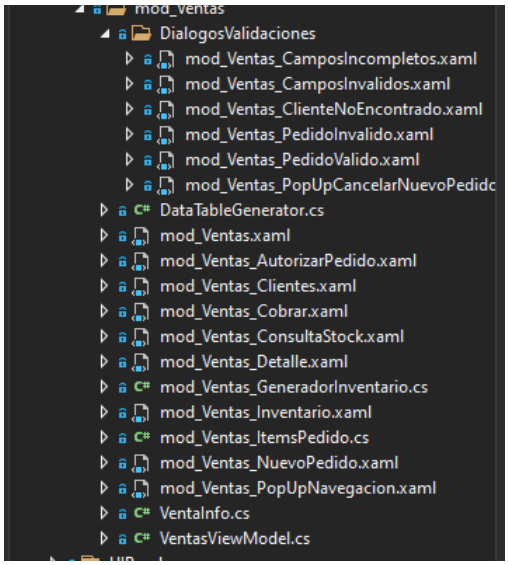

Figura # 25 Vistas generales del módulo de ventas.

La figura 25 es la estructura de las vistas y el directorio del proyecto para el módulo de ventas. Cualquier proceso relacionado a flujo de trabajo se lo mantiene dentro del directorio para conservar la idea de la arquitectura SOA. Xamarin soporta CSS pero es extremadamente limitado por lo que los estilos son configurados dentro de los archivos XAML la mayoría de veces. Tan solo el tipo de letra se configuró globalmente.

Detalle de las vistas

Figura # 26 Splashscreen.

Figura # 27 Inicio de sesión.

Figura # 28 Diálogo de recuperación de credenciales.

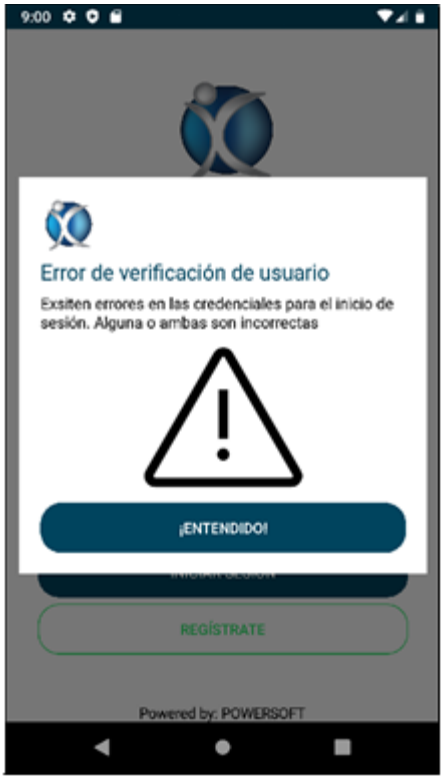

Figura # 29 Diálogo de error de credenciales.

Figura # 30 Diálogo de registro.

Figura # 31 Menú principal, módulos.

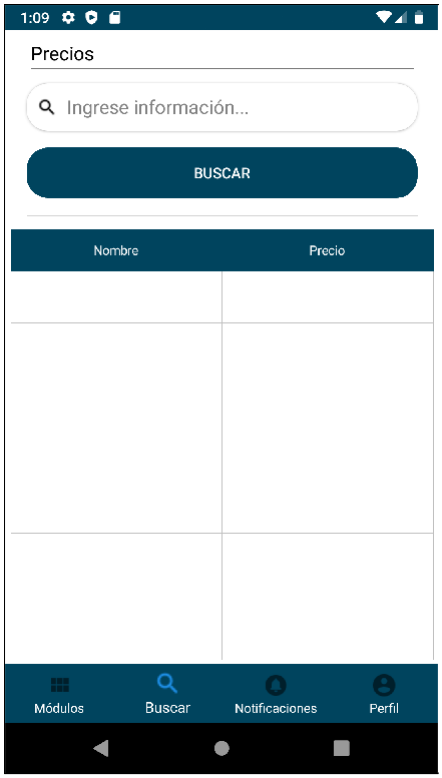

Figura # 32 Menú principal, búsqueda.

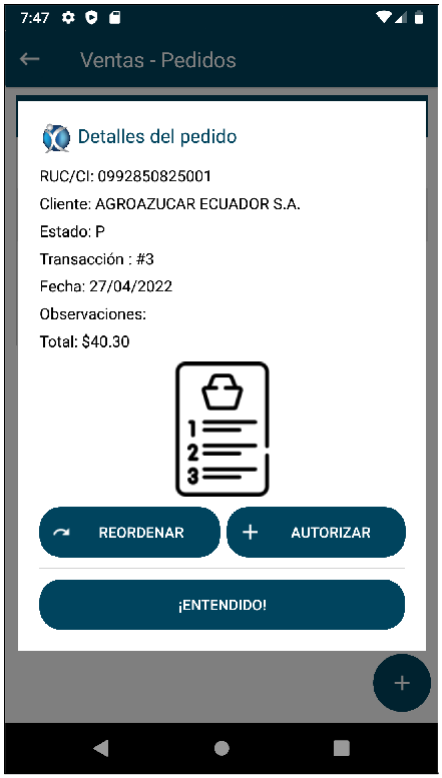

Figura # 33 Reordenar pedido.

Figura # 34 Menú principal, perfil.

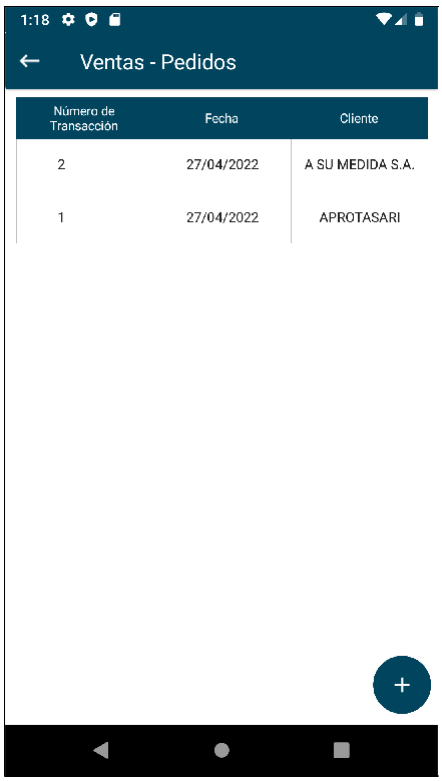

Figura # 35 Módulo de ventas, pedidos.

Figura # 36Autorización de pedido.

Figura # 37 Confirmación de la autorización.

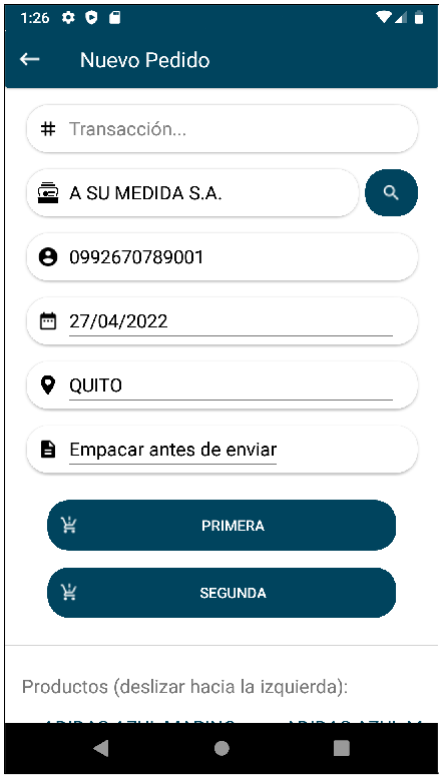

Figura # 38 Reordenar pedido.

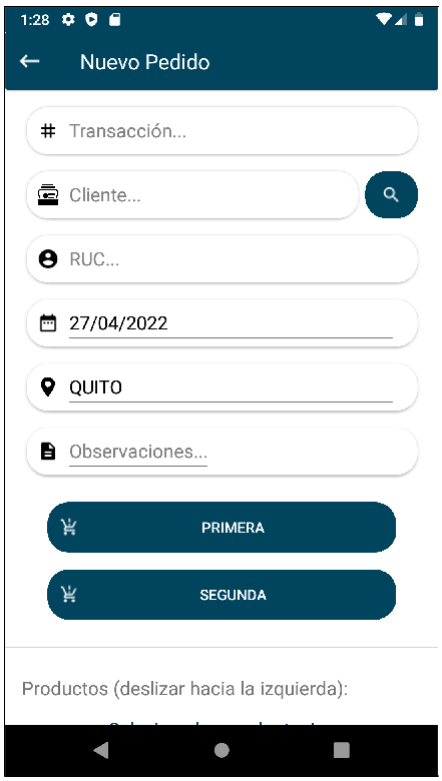

Figura # 39 Nuevo pedido.

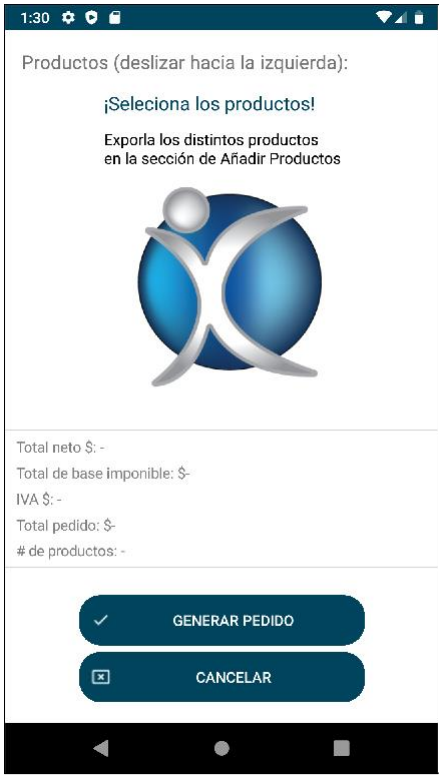

Figura # 40 Preventa y resumen de compra.

Figura # 41 Búsqueda de cliente.

Figura # 42 Campos inválidos.

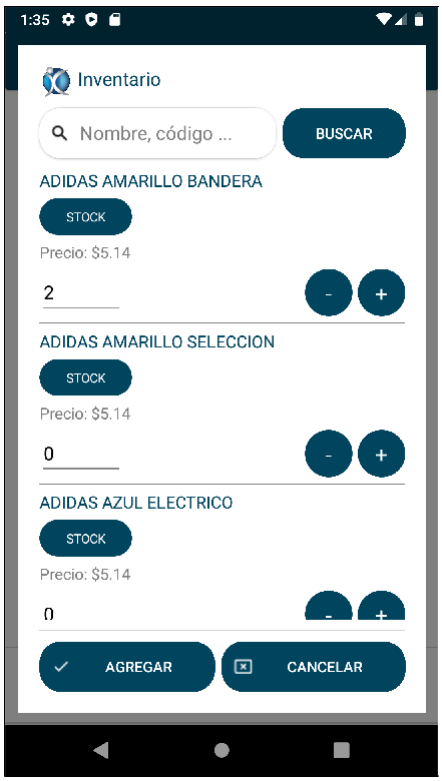

Figura # 43 Módulo de inventario.

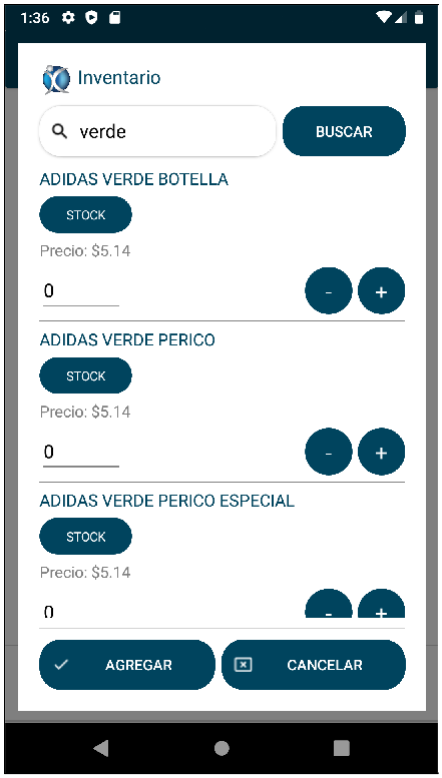

Figura # 44 Consulta por patrón.

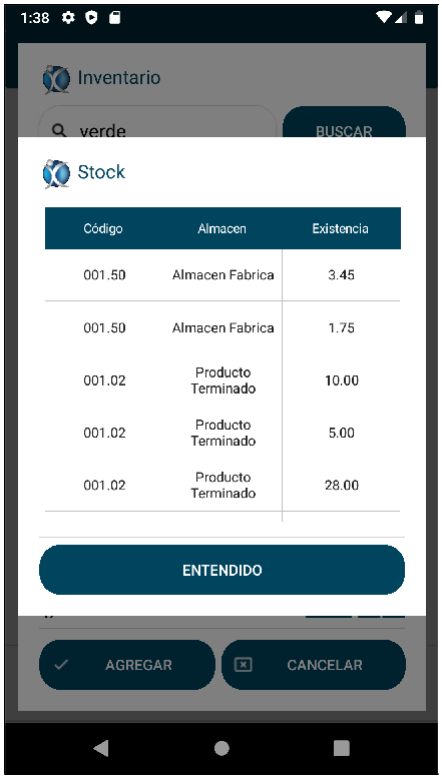

Figura # 45 Resultado de una consulta.

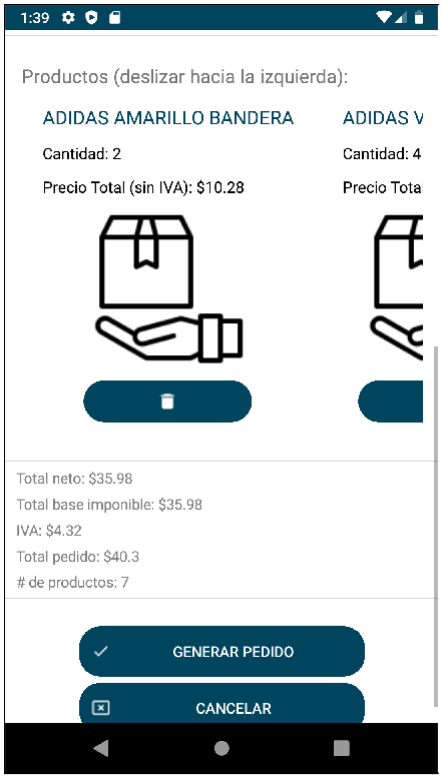

Figura # 46 Cálculos de preventa.

Figura # 47 Pedido correcto.

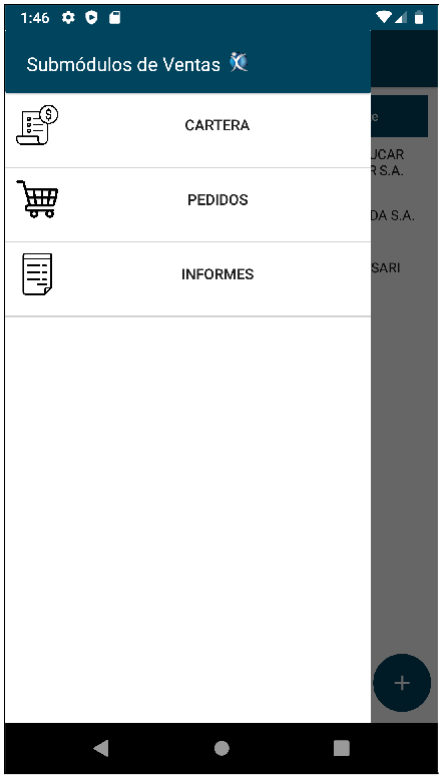

Figura # 48 Patrón maestro-detalle en módulo de ventas.

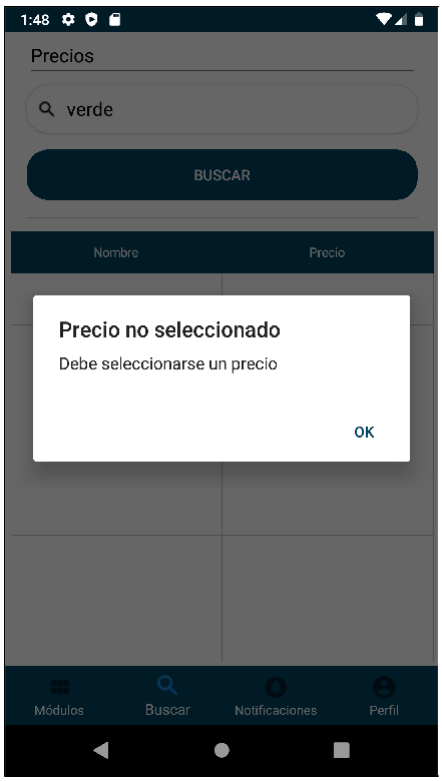

Figura # 49 Precio no seleccionado en búsqueda.

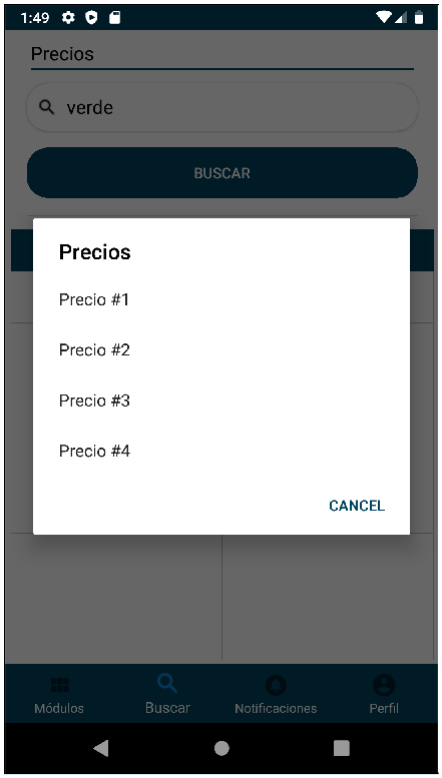

Figura # 50 Selección de precios.

Figura # 51 Resultados módulo inventario en pestaña de búsqueda.

Flujo de trabajo

Las vistas presentadas en la sección anterior son todos los posibles casos de uso a los que el usuario puede llegar. No siempre se van a dar todos los casos de uso, muchos son diálogos de validaciones que cuando se llega a esa vista es porque no hay un flujo incorrecto de trabajo. Cuando ocurre eso, el usuario lee un mensaje del error y automáticamente se reinicia el proceso. Cuando todo está en orden se regresa al punto inicial. En resumen esos son los dos casos de uso más generales de Kohinor Móvil. Para la explicación del flujo de trabajo se asume un flujo de aplicación correcto (que el usuario haya iniciado sesión de manera correcta y tenga conocimiento mínimo de la operabilidad de la aplicación como para poder dirigirse al módulo de ventas). Se usa un mapa de casos de uso que son las vistas de la sección anterior y su número de figura.

Uso correcto

Realizar un pedido consta en abrir el módulo de ventas en la sección de pedidos, cuando la tabla de historial se muestra en la esquina inferior derecha hay un botón con un icono (+) que sirve para desplegar la vista de un nuevo pedido. Dentro de esta nueva vista y por ser un nuevo pedido el campo de información del cliente debe llenarse. Para esto se aplasta en el botón cuyo icono es una lupa para buscar un nuevo cliente y despliega un diálogo en el cual se busca al cliente y se lo selecciona para luego confirmar los cambios y cerrar automáticamente el diálogo. Una vez los campos validados los botones de selección de inventario son habilitados y se elige un inventario. Al aplastar el botón aparece un diálogo que despliega el módulo de inventario para elegir un producto por cantidad o consultar más detalles de inventario. Cuando los productos deseados son encontrados, se confirma y el diálogo se cierra. En la vista de nuevo pedido se visualizan los productos en la preorden, y estos productos si es necesario pueden eliminarse. En caso de verificar que todo esté correcto

se presiona el botón de enviar pedido. Se despliega una confirmación para regresar a la vista de la tabla de historial, en la cual se puede comprobar que el pedido fue generado correctamente. Los pasos a seguir según el mapa de vistas:

$$31 \rightarrow 35 \rightarrow 39 \rightarrow 41 \rightarrow 38 \rightarrow 43 \rightarrow 44 \rightarrow 46 \rightarrow 47 \rightarrow 31 \qquad (1)$$

Como la arquitectura propuesta considera a un pedido nuevo igual que reordenar el flujo de trabajo de la secuencia 1 se mantiene sustituyendo las visas 39 y 41 por la 33. La secuencia 2 consta de los pasos siguientes:

$$31 \rightarrow 35 \rightarrow 33 \rightarrow 38 \rightarrow 43 \rightarrow 44 \rightarrow 46 \rightarrow 47 \rightarrow 31 \qquad (2)$$

Uso incorrecto

Cualquier flujo de trabajo que sea distinto en al menos un paso de las secuencias 1 y 2 se considera incorrecto y no tiene efecto alguno ya que se cancela y desde el paso que existió un error se dan instrucciones para reanudar los pasos correctos o reiniciar todo el proceso nuevamente.

CONCLUSIONES Y TRABAJO FUTURO

Como conclusiones, y haciendo referencia a la sección *Objetivos generales y específicos* se puede concluir lo siguiente:

- La reingeniería de KERP para su versión móvil conserva la operabilidad de sus otros productos y por el diseño de la capa de presentación es similar a sus otras presentaciones.
- Las funcionalidades de la aplicación si bien pueden ser distintos flujos de trabajo por usarse en diferentes tipos de dispositivos siguen usando el mismo núcleo del ERP en SQL Server.
- La información que usa opera en bases de datos ya existentes La solución fue una aplicación móvil.
- Los módulos de ventas e inventarios forman parte de la solución.

Como observaciones generales de las conclusiones todas las soluciones y herramientas son de Microsoft a excepción de los iconos. Office UI Fabric Icons es un repositorio de iconos propuestos para mantener una interfaz similar propuesta por Microsoft, pero los recursos aunque no provengan de una fuente centralizada sirven para conservar el diseño de los otros productos. Como trabajo futuro se propone un estudio de UI/UX más detallado y usar el repositorio mencionado para realmente hacer uso de un stack completo de tecnologías.

También un tema que no se abordó en el trabajo fueron procesos de Development Operations para despliegues de producción más controlados ya que por el momento Kohinir Móvil se distribuye por medios de APK's. Los limitantes de DevOps en este caso fueron porque los potenciales clientes no tienen o no están interesados en departamentos de IT y/o infraestructuras físicas de servidores que soporten un despliegue a producción real. Por otra parte, el trabajo se enfocó en su totalidad para Android y aunque la aplicación fue creada como solución multiplataforma no hubieron pruebas en dispositivos iOS. A la fecha del documento, la aplicación trabaja sobre bases de datos reales por lo que, aunque no forma parte de las conclusiones planteadas originalmente, es un producto que si es una solución de software empresarial.

REFERENCIAS

Banco Mundial, Indicadores de Desarrollo Mundiales. (2022). *Subscripciones a telefonía celular móvil. Recuperado de Banco Mundial.* Recuperado de https://datos.bancomundial.org/indicator/IT.CEL.SETS.P2

Banco Mundial, Indicadores de Desarrollo Mundiales. (2022). *Personas que usan el internet. Recuperado de Banco Mundial.* Recuperado de https://datos.bancomundial.org/indicator/IT.NET.USER.ZS

European Knowledge Center for Information Technology (Ed.). (2015). *¿Qué es un sistema ERP y para qué sirve?* Recuperado de TIC Portal. https://www.ticportal.es/temas/enterprise-resourceplanning/que-es-sistema-erp

Real Academia de la Lengua Española. (2015). *Diccionario panhispánico de tratamiento de dudas*. Recuperado de https://www.rae.es/dpd/ayuda/tratamiento-de-los-extranjerismos

Microsoft. (2019). *Using Always Encrypted with the Microsoft .NET Data Provider for SQL Server*. Recuperado de https://docs.microsoft.com/en-us/sql/connect/ado-net/sql/sqlclient-supportalways-encrypted?view=sql-server-ver15