BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

Desarrollo de una plataforma para la gestión eficiente de puntos de venta en el contexto empresarial

TESIS PROFESIONAL
para obtener el título de
LICENCIATURA EN INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA
RENE LOPEZ VERUN

ASESOR:
Dr. Mariano Larios Gómez

Puebla, Puebla. Diciembre 2024

Dedicatoria:

A mi familia que me ha apoyado incondicionalmente y siempre ha estado ahí para mí, mi hermana que es una parte fundamental en mi vida, a mis tíos, abuelos, primos de los cuales solo he recibido muestras de afecto, y en especial a mis padres sin los cuales no hubiera podido ser nada de lo que yo soy.

Agradecimientos:

Agradezco a todas las personas que recorrieron mi formación profesional a mis maestros, amigos, familia y sobre todo a mis padres ya que este logro no hubiera sido posible sin su apoyo incondicional, ha sido un camino largo, lleno de altibajos, pero siempre con la guía y enseñanzas de mis padres he podido salir adelante.

“Infinitas Gracias”

Índice general

Contenido

1. Introducción

Este proyecto de investigación se centró en el desarrollo de una aplicación integral diseñada para optimizar la gestión de puntos de venta, combinando tecnologías avanzadas mediante la integración de bases de datos robustas en el backend y utilizando especializados lenguajes de consulta, además de un diseño atómico para el frontend basado en componentes. De ésta forma se asegura una gestión eficiente de los recursos ofreciendo una programación modular y reutilizable, aprovechando al máximo una variedad de frameworks disponibles en el mercado.

Entre las características más destacadas de esta aplicación se consideró una gestión más eficiente de los usuarios y sus roles, lo que facilita la asignación de permisos y responsabilidades con presición. De igual forma, la aplicación permite la gestionar de manera intuitiva las altas y bajas de productos y servicios, mejorando así la eficiencia operativa. El procesamiento de pagos se realizó de forma ágil y segura, utilizando la generación automática de tickets detallados para facilitar el seguimiento de las transacciones. Además, la aplicación gestionó sesiones de usuario y administrando cuentas cuidando la seguridad para garantizar la protección de los datos sensibles.

El proyecto no se limitó únicamente al desarrollo técnico de la aplicación, sino que también se abordó de la optimización de los procesos en los puntos de venta. Así mismo se enfocó en mejorar la gestión del inventario para brindar una administración más precisa y eficiente, simplificando las transacciones comerciales, y logrando reducir tiempos para mejorar la experiencia tanto del usuario final como del administrador del sistema.

Además, el enfoque de esta investigación consideró la colaboración internacional, permitiendo la integración de mejores prácticas y la aplicación de

conocimientos en el campo de la tecnología aplicada en la gestión empresarial. Esta contribución no solo transformó la eficiencia operativa en puntos de venta, sino que también representó un avance considerable en el desarrollo de soluciones tecnológicas que responden a las necesidades dinámicas del comercio moderno.

1.1 Antecedentes

Este proyecto surge de la necesidad imperativa de gestionar de forma centralizada y eficiente múltiples puntos de venta, lo cual representa un gran desafío para las empresas que operan en entornos dinámicos y altamente competitivos. La creciente complejidad en la administración de inventarios, almacenes, proveedores, clientes y registros de actividades, junto con la necesidad de realizar cierres de caja precisos y eficientes, puso en evidencia una urgencia por desarrollar soluciones integrales que resuelvan estos desafíos de manera holística.

Esta propuesta surge no solo para simplificar la gestión operativa sino también para optimizarla, además de buscar incorporar capacidades avanzadas de asignación y desarrollo de microservicios, los cuales, permiten una escalabilidad y mayor flexibilidad en el manejo de las diferentes funciones del negocio, favoreciendo la integración de nuevas tecnologías y prácticas innovadoras conforme evolucionan las necesidades del mercado.

Otro de los pilares del proyecto fue la automatización de los procesos de pago, lo que representa un aspecto estratégico para reducir el tiempo de transacción, además de disminuir los errores humanos y mejorar la experiencia del cliente. Esta automatización se diseñó para integrarse de manera fluida con navegadores web, a fin de garantizar mayor seguridad y mejorar la eficiencia en las transacciones financieras.

El proyecto también buscó incrementar la competitividad en el mercado implementando tecnologías que permitieran a la empresa responder más rápidamente al cambio dinámico de las demandas del mercado. Para éste fin se centralizó la gestión

en una matriz única buscando no solo mejorar la coordinación entre los distintos puntos de venta, sino también beneficiar la posibilidad de tomar decisiones estratégicas y más informadas con datos en tiempo real.

Finalmente, la propuesta se hizo también con el fin de transformar la forma de gestionar los puntos de venta, alinear los procesos operativos a las mejores prácticas industriales, y considerando posicionar a la empresa como líder en eficiencia operativa y con capacidad de adaptación al cada vez más exigente entorno empresarial.

1.2 Descripción del problema

En el contexto actual de alta competitividad del comercio minorista, es escencial una eficiencia operativa acompañado de una rápida capacidad de adaptación a las necesidades del mercado, a fin de garantizar el éxito de cualquier negocio. Por ello se requieren el diseño y la implementación de una plataforma integral de gestión de puntos de venta, que brindará una solución estratégica para centralizar la administración, optimizar recursos, y sobretodo garantizar una mejor experiencia de usuario. El objetivo de este proyecto es el desarrollo de una solución tecnológica que permita la integración de múltiples puntos de venta en una plataforma centralizada, a fín de facilitar la supervisión y el control desde una matriz central. Con éste enfoque centralizado se simplifica la gestión operativa además de mejorar la toma de decisiones proporcionando a los administradores datos unificados y en tiempo real.

Un aspecto muy importante de esta plataforma es la optimización de recursos, lo cual se logra utilizando bases de datos robustas en el backend, seguido de un diseño atómico para el frontend, basado en componentes. De ésta forma la arquitectura asegura una máxima eficiencia en el uso de recursos facilitando una programación modular y altamente reutilizable, logrando de esta forma una rápida adaptación a necesidades futuras y sin comprometer la estabilidad del sistema [3]. Aunado a lo mencionado, la plataforma también incorpora un avanzado sistema de gestión de roles,

garantizando que solo personal autorizado pueda acceder a información sensible, preservando la privacidad y la integridad de los datos.

Otro componente clave de la plataforma es la administración de inventarios, lo que permite gestionar eficientemente almacenes y proveedores, para asegurar un adecuado y oportuno abastecimiento de productos en cada punto de venta. Como complemento el sistema permite la interacción entre sucursales, facilitando el intercambio de productos de acuerdo a la demanda y la disponibilidad en el inventario, reduciendo significativamente el riesgo de desabastecimiento y optimizando la rotación de inventarios.

Es prioridad en este proyecto la automatización de procesos en general, como el procesamiento de pagos, la generación de tickets, y la gestión de cuentas, los cuales no solo reducen el tiempo y esfuerzo requerido por los usuarios, sino también minimizan los errores y principalmente mejoran la eficiencia operativa en general. Alineado con estos objetivos, se ha puesto un fuerte énfasis en mejorar la experiencia del usuario, buscando que la interfaz de la plataforma sea intuitiva, accesible y fácil de utilizar para todos los involucrados en la gestión de puntos de venta.

Otro aspecto ptioritario de esta plataforma es la seguridad de la información, la cual se aborda implementado medidas de seguridad robustas para proteger la información confidencial, lo que permite garantizar la integridad y disponibilidad de los datos todo el tiempo. Finalmente y no menos importante se consideran la escalabilidad y flexibilidad de la plataforma, como elementos para asegurar que esta pueda adaptarse a futuras expansiones y cambios del entorno empresarial, permitiendo la incorporación de nuevos puntos de venta y si fuera necesario la adición de funcionalidades.

En resumen, este proyecto aborda necesidades inmediatas de gestión eficiente de puntos de venta, además se diseñó para anticiparse a desafíos futuros, ofreciendo una solución robusta, segura y adaptable que puede evolucionar al ritmo de las demandas del mercado.

1.3 Metodología de desarrollo

En este proyecto se utiliza la base de datos en mongoDB ya generada, sin embargo para mejorar la accesibilidad se le incorporado el lenguaje de consulta GraphQL, el cual contrasta con la arquitectura de REST y con los métodos POST, GET, DELETE, EDIT, ya que es común con REST obtener datos innecesarios en la petición, de tal forma que con GQL podemos especificar qué datos queremos, además de que no necesitan múltiples endpoints para acceder a los recursos, ya que todo el proceso se realiza a través de una sola URL y dependiendo de como estructuremos la consulta nos indicará los datos a obtener.

Además de realizar las consultas podemos hacer uso de las mutaciones con las cuales editaremos datos y suscripciones que nos permitirán observar en tiempo real los cambios en la información.

Ya que gracias a GraphQL podemos definir que datos observar, podemos evitar múltiples llamadas a la API, logrando de ésta forma que la carga de nuestra red se reduzca, además de mejorar el rendimiento del software.

El desarrollo de ésta plataforma integral de gestión de puntos de venta, requirió una planificación y análisis detallado para identificar todas las necesidades y los principales desafíos inherentes para una centralización administrativa. La identificación de estos problemas fue fundamental para comprender las características con las que se implementará la solución, de tal forma que permita una alineación precisa entre los requisitos del sistema y los objetivos del proyecto. Por lo anterior el análisis de requerimientos, tanto para aspectos funcionales como no funcionales, es un paso crítico en esta fase, ya que permitió recopilar y analizar en detalle los aspectos técnicos y operativos del sistema. Una vez que se cuenta con esta comprensión profunda de los requisitos es posible clarificar y refinar los objetivos del proyecto, tanto generales como específicos, logrando de ésta forma establecer una dirección clara y precisa del proceso de desarrollo.

En la fase de diseño, se utilizaron los principios de diseño atómico con React, desarrollando componentes reutilizables y que garantizan una interfaz de usuario coherente y escalable. Esta metodología no solo facilita la gestión de la complejidad en el frontend, sino que también asegura la consistencia visual y funcional de la aplicación. El prototipado se llevó a cabo con Figma, utilizándola como una herramienta colaborativa esencial, lo que permite realizar iteraciones ágiles en el diseño de la interfaz de usuario y facilita la retroalimentación continua de los stakeholders. También se llevó a cabo el diseñó de una arquitectura de GraphQL, la cual fue fundamental para gestionar eficientemente los datos y las consultas entre el frontend y el backend, mejorando tanto la flexibilidad del sistema, así como su rendimiento.

Durante la fase de desarrollo se utilizó una metodología de implementación incremental, que se centra en el desarrollo iterativo de funcionalidades, priorizando su valor para el usuario, así como su complejidad técnica. Este enfoque aseguró entregas tempranas y frecuentes, permitiendo a los usuarios interactuar con el sistema desde las primeras fases del proyecto. Las pruebas continuas, incluidas las pruebas unitarias y la integración continua, eran esenciales para garantizar la calidad y confiabilidad del software en todas las fases de desarrollo. La integración tecnológica de herramientas como React, GraphQL y otros frameworks relevantes se realizó de manera armoniosa, asegurando consistencia y eficiencia en la implementación del sistema.

La evaluación y la retroalimentación fueron componentes críticos para el éxito del proyecto. Se realizaron pruebas de usabilidad con usuarios reales para evaluar la experiencia de usuario y detectar áreas de mejora. La retroalimentación iterativa, recopilada sistemáticamente de los usuarios y partes interesadas, permitió realizar ajustes y mejoras continuas en el sistema. Este enfoque de iteración y mejora continua aseguró que el sistema evolucionara en respuesta a las necesidades de los usuarios y los cambios en el entorno operativo.

La implementación y despliegue del sistema se realizó de forma paulatina, asegurando la estabilidad y el correcto funcionamiento en cada etapa del proceso. Se establecieron mecanismos de monitoreo y mantenimiento para garantizar que el sistema mantuviera su disponibilidad y rendimiento óptimo una vez implementado en producción. Estos mecanismos también permitieron detectar y resolver rápidamente cualquier problema que pudiera surgir, asegurando una experiencia de usuario confiable y eficiente.

1.3.1 Diseño atómico y Graphql

El diseño atómico es una parte crucial del desarrollo de ERP escalable, ya que nos proporciona una solución de diseño de interfaz ideal. Con esta metodología podemos tener componentes bien definidos y utilizarlos en muchas partes del proyecto, sea cual sea el nivel en el que nos encontremos. . Podremos combinar su uso según convenga en la vista deseada.

Fig. 1. Esquema del Diseño atómico.

Con React esto es bastante útil ya que, por definición, la biblioteca se basa en componentes y estados y, con el diseño atómico, estos estados se pueden pasar fácilmente como propiedades mediante herencia.

Para ejecutar consultas Graphql se utiliza Apollo que es un servidor encargado de conectarse a la base de datos desde el endpoint configurado y este servidor Apollo involucra a toda la aplicación cliente en base a su definición, lo que significa que podemos realizar consultas y mutaciones Graphql en cualquier lugar de nuestro

aplicación cliente simplemente usando los Hooks que nos proporciona la biblioteca Apollo para React

1.4 Desarrollo del proyecto

El proyecto de desarrollo del sistema de gestión de puntos de venta se estructuró en varias fases claramente definidas a lo largo de un año. En el primer mes se llevó a cabo la planificación y evaluación de necesidades. En esta fase se definieron el alcance y objetivos del proyecto, se realizaron reuniones con el equipo para definir roles y responsabilidades y se compilaron las especificaciones necesarias para el sistema.

Durante el segundo y tercer mes se centró en el diseño y arquitectura de sistemas. Se diseñó la arquitectura general, se crearon diagramas de flujo y modelos de datos y se eligieron las tecnologías y herramientas más adecuadas para implementar el proyecto.

Entre el cuarto y sexto mes, el equipo se dedicó al desarrollo backend. En esta etapa se configuró el entorno de desarrollo, se implementó la lógica de negocios necesaria y se integraron las bases de datos y API esenciales para el funcionamiento del sistema.

Los meses siete a nueve se dedicaron al desarrollo front-end. Durante este período, se diseñó la interfaz de usuario, se implementaron los componentes y visualizaciones necesarios y se integró el frontend con el backend para garantizar un intercambio de datos fluido.

En los meses diez y once se llevaron a cabo pruebas exhaustivas y optimizaciones del sistema. Se realizaron pruebas unitarias, de integración y del sistema, se identificaron y corrigieron errores y se optimizaron el rendimiento y la seguridad del sistema para garantizar la estabilidad del sistema.

En el duodécimo mes se implementó y puso en marcha el sistema. Se preparó el entorno de producción, se migraron los datos cuando fue necesario, se capacitó a los empleados en el uso del sistema y se lanzó oficialmente el sistema de gestión de puntos de venta. Este proceso aseguró que el sistema fuera completamente funcional y estuviera listo para su uso en el entorno de producción.

1.5 Infraestructura utilizada

Para desarrollar este proyecto se utilizó un equipo de cómputo de gama media alta Acer Aspire con procesador Intel i5 con mínimo 8 GB de memoria RAM, lo que le permite manejar múltiples aplicaciones y procesos simultáneamente, asegurando una experiencia fluida durante el desarrollo, además de al menos 20 GB para almacenar todos los archivos de código fuente, recursos multimedia y documentos relacionados con el proyecto.

En el proyecto se utilizó Visual Studio como plataforma de desarrollo integrada (IDE), ya que proporciona un conjunto completo de herramientas avanzadas para crear, depurar e implementar aplicaciones en múltiples plataformas. También se utilizó Git para llevar a cabo un control de versiones y dar un seguimiento de los cambios en el código fuente, además de permitir colaboraciones entre los miembros del equipo. De esta forma GitHub nos permite alojar los repositorios Git, lo que habilita un almacenamiento remoto y colaborativo en proyectos de software de manera ordenada y, principalmente segura.

Para el diseño de la interfaz de usuario y la creación de prototipos interactivos se utilizó Figma, la cuál además es una herramienta colaborativa basada en la nube, así se facilita la creación de prototipos de alta fidelidad y se asegura una colaboración en tiempo real. De esta forma se puede realizar un trabajo colaborativo eficiente entre diseñadores y desarrolladores.

Debido a las características de las diversas herramientas se requiere una conexión a Internet de alta velocidad para descargar actualizaciones de software, acceder a recursos de la nube y colaborar en tiempo real con los miembros del equipo, de la estabilidad de la conexión se garantiza que el proyecto avance sin problemas y con el soporte necesario.

También se desarrolló un sistema integral de gestión de puntos de venta para centralizar la gestión de inventarios, usuarios, transacciones y otros aspectos esenciales.

En paralelo se implementó una aplicación web con versión móvil responsiva, a fin de birndarle al usuario final una interfaz intuitiva y fácil de usar, permtiendo gestionar de forma remota los puntos de venta desde cualquier dispositivo con conexión a Internet, brindando permitiendo con ello una gran flexibilidad y accesibilidad a administradores y operadores de sistemas.

Se diseñó además una base de datos robusta y escalable en el backend del sistema, asegurando la integridad y disponibilidad de los datos. Estas características del backend son fundamentales para garantizar el manejo de grandes volúmenes de datos sin comprometer su rendimiento y estabilidad.

El sistema desarrollado incorporó diversas características importantes, como por ejemplo la gestión de usuarios y roles, la gestión de inventario, informes y análisis de datos, y la automatización de los procesos.

Otra etapa importante fué elaborar la documentación completa y lo más detallada posible, la cual incluyó manuales de usuario, guías de instalación, especificaciones técnicas y toda la información necesaria para la instalación, configuración y operación del sistema. Esta documentación es de gran importancia para garantizar que los usuarios pudieran utilizar el sistema de forma eficaz y sin dificultades.

Se realizaron extensas pruebas para validar el funcionamiento del sistema en diferentes escenarios y condiciones, asegurando así la calidad y confiabilidad del software desarrollado. Estas pruebas fueron esenciales para garantizar que el sistema cumpliera con los estándares de calidad y estuviera listo para su implementación.

Además, se llevó a cabo la capacitación al personal involucrado en la operación y uso del sistema, para brindándole las habilidades y conocimientos necesarios para aprovechar al máximo las funcionalidades de éste. La capacitación es vital para garantizar que la migración al nuevo sistema sea fluida y que el personal estuviera preparado para aprovechar de manera efectiva se potencial.

También se obtuvo una implementación exitosa del sistema en los puntos de venta de cada objetivo, asegurando esta transición fluida desde los sistemas existentes y minimizando de esta forma cualquier impacto en las operaciones diarias. A fin de mantener el sistema actualizado y alineado con las necesidades comerciales y del mercado, se diseñaron mecanismos de retroalimentación y mejora iterativa continua del sistema, logrando asegurar su relevancia y efectividad a largo plazo.

El objetivo del proyecto de implementación del sistema de gestión de puntos de venta busca generar un impacto positivo en varias áreas clave. Primero, mejorar significativamente la eficiencia operativa mediante la automatización de procesos y la centralización de la gestión del punto de venta, lo que permitirá a las empresas reducir los tiempos de gestión y minimizando los errores humanos, propiciando aumentar la productividad y minimizando los costos operativos.

En segundo lugar se llevará a cabo la creación de empleos, adoptando nuevas tecnologías y sistemas de gestión que sin duda aumentarán la demanda de personal especializado en el sector de tecnología y servicios. Al abrirse esta necesidad de nuevas habilidades no sólo se abrirán oportunidades de empleo, sino que también generará una necesidad de programas de capacitación, contribuyendo así al crecimiento del capital humano en la industria.

En tercer lugar se impactará la experiencia del cliente mediante la cual se tendrá una gestión más eficiente de los puntos de venta y que dará como resultado disminuir el tiempo de espera, así como generar inventarios más precisos y transacciones más rápidas y seguras. Con estos beneficios no sólo aumentará la satisfacción del cliente sino también se fomentará la lealtad y retención de estos a largo plazo.

El proyecto también apoyará el desarrollo tecnológico local, implementando tecnologías avanzadas para la gestión empresarial promoviendo la innovación y la competitividad a nivel regional. Esta adopción tecnológica contribuirá al crecimiento económico local, fortaleciendo las capacidades tecnológicas y operativas de las empresas.

El proyecto también, jugará un papel crucial en la digitalización de las operaciones comerciales, facilitando la transición a otros modelos de negocio más ágiles y adaptables, ayudando a las empresas, especialmente en entornos donde la tecnología aún no se ha adoptado por completo,realizar esta transición a modernizar sus procesos y seguir siendo competitivos en un mercado dinámico en constante cambio.

Son muchoas los beneficiarios de este proyecto. Tanto las empresas como los comercios locales se beneficiarán directamente del sistema de gestión, ya que mejorarán de forma eficiente su capacidad para satisfacer las demandas del mercado. También se beneficiarán los usuarios finales, es decir, los clientes de los puntos de venta, ya que disfrutarán de un servicio de calidad y una mejor experiencia de compra, lo que resultará en aumentar su fidelidad a las marcas locales. Finalmente, las comunidades en la región donde operan estos puntos de venta también se beneficiarán, ya que la eficiencia y el crecimiento económico de las empresas locales contribuirán de forma directa al desarrollo socioeconómico de la región, mejorando no solo la calidad de vida en estas zonas, sino también promoviendo un medio ambiente más próspero y principalmente sostenible.

1.6. ingeniería de software y la gestión empresarial

Aunque este proyecto no propone como investigación básica en el sentido tradicional, su desarrollo y aplicación puede contribuir significativamente a la aplicación del conocimiento generado en varias áreas, principalmente en el área de la ingeniería de software y la gestión empresarial. En éste sentido a continuación se detallan algunas de las contribuciones:

- Innovación Tecnológica:
- Mejora de procesos:
- Desarrollo de herramientas:
- Transferencia de conocimiento:
- Impacto en la industria:

2. Marco Teórico

El área de la gestión de puntos de venta (POS) y los sistemas de administración empresarial (ERP) ha experimentado un gran crecimiento en las últimas décadas, impulsado principalmente por los avances tecnológicos y la contínua demanda del mercado global. En este estado del arte se examinan las tendencias y desarrollos más relevantes que están marcando el futuro de este sector, a fin de proporcionar una visión integral de las innovaciones y desafíos emergentes.

Avances en Tecnologías de Desarrollo

Uno de los factores que ha marcado la gestión de puntos de venta es el constante crecimiento en el desarrollo de aplicaciones web y móviles. Este crecimiento ha impulado a las empresas a ofrecer accesos más flexibles y ubicuo a las funciones de administración, lo que es esencial en un mercado que se ha caracterizado por la alta movilidad y la necesidad de altas velocidades de respuesta. Gracias a que las aplicaciones web y móviles han transformado la forma en que las empresas gestionan sus operaciones, esto ha permitido el acceso desde cualquier dispositivo con conexión a Internet. Por ejemplo, empresas como "Retail X" han implementado aplicaciones móviles POS que permiten a sus vendedores procesar ventas de manera más rápida y eficiente directamente, mejorando la experiencia del cliente y la eficiencia operativa [23, 24].

Además, la utilización de frameworks y librerías como React js, Angular y Vue js han revolucionado el desarrollo de las interfaces de usuario en el campo de los puntos de venta, lo que ha facilitado la creación de interfaces dinámicas e interactivas que mejoran de forma sustancial la experiencia del usuario y la eficiencia de los sistemas. Otro ejemplo de gran trascendencia es el de la empresa "Y Stores", el cual utilizó React.js para desarrollar una interfaz POS intuitiva y fácil de usar, lo que redujo notablemente el tiempo de capacitación necesario para los nuevos empleados [25, 26].

La integración de diversas funcionalidades avanzadas en los sistemas de gestión empresarial también ha permitido una transformación profunda en la forma en que las empresas manejan sus operaciones, debido a la incorporación de tecnologías como el análisis de datos en tiempo real, la inteligencia artificial (IA) y el aprendizaje automático, lo que ha permitido a las empresas tomar decisiones más ágiles e informadas. El uso de estas tecnologías potencializan la eficiencia operativa y la competitividad, como en el caso de "Z Mart", una cadena de supermercados que utiliza análisis de datos en tiempo real para predecir la demanda de productos y optimizar su inventario, logrando con esto reducir costos y mejorar la satisfacción del cliente.

Paralelamente se han adoptado arquitecturas modernas como los microservicios y los contenedores, que han revolucionado el diseño y despliegue de sistemas de gestión empresarial, permitendo una mayor flexibilidad y escalabilidad, además de facilitar la integración con sistemas externos, adaptándose de manera más ágil a los cambios en los requisitos del negocio, en este campo, la empresa de manufactura "Alpha" es otro ejemplo de éxito, al implementar una arquitectura de microservicios en su ERP, lo que les permitió integrar nuevas funcionalidades de manera ágil y actualizar módulos específicos de forma independiente.

2.1 Reglamento General de Protección de Datos

La seguridad y privacidad de los datos son otros dos aspectos fundamentales que han afectado el diseño de los sistemas de administración de puntos de venta. La conformidad con las regulaciones, como el Reglamento General de Protección de Datos de la Unión Europea y la Ley de Privacidad del Consumidor de California, ha obligado a las organizaciones a aumentar sus preocupaciones por el modo en que los sistemas de Punto de Venta están siendo diseñados e implementados. Para cumplir con estas regulaciones y proteger la información confidencial de los clientes, muchas empresas están utilizando algunas medidas adicionales como: sistemas de cifrado, autenticación de dos factores y administración de vulnerabilidades, entre otros. Por ejemplo, “Beta Inc.” tuvo que asegurarse de que sus empleados estén utilizando

medidas de seguridad adicionales y medidas adicionales para proteger los datos de sus clientes.

Uno de los desafíos futuros es la interoperabilidad entre sistemas. Dado que las empresas implementarán varios sistemas de ERP, TMS, WMS y puntos de venta de proveedores diferentes, necesitarán la capacidad de integrar estos diversos sistemas y que se comuniquen de manera efectiva. La adopción de API abiertas, estándares de transferencia de datos como EDI y soluciones de integración empresarial ha demostrado ser una solución para abordar la interoperabilidad entre sistemas heterogéneos.

Así, la personalización y mejora de la experiencia del cliente terminan siendo anticipos en cuanto al enfoque priorizado de la evolución de los sistemas de gestión de punto de venta. La utilización de análisis predictivos y las propuestas para los clientes logran ayudar a las empresas a personalizar sus ofertas y servicios en línea con las necesidades y deseos del cliente, lo que aumenta su fidelización y éxito comercial. Toma el ejemplo de la empresa “Gamma Retail”, que implementa el mecanismo de análisis predictivo en su punto de venta. Como consecuencia, surge un alto índice de las recomendaciones de los clientes, y esto tiene un transferencia directa en los incrementos de las ventas y la satisfacción de los consumidores.

2.2 gestión de puntos de venta

El campo del punto de venta y la administración de los sistemas empresariales están en rápida evolución, influenciados por la innovación tecnológica y las cambiantes demandas del mercado. Las tendencias abordadas en este estado del arte: el desarrollo de aplicaciones móviles, la integración de inteligencia artificial y el foco en la seguridad y la privacidad, están forjando el futuro de esta industria. La capacidad de las firmas para adecuarse a ellas y aprovechar las oportunidades que surgen de estas será clave para su éxito en un ámbito de los negocios cada vez más desafiante.

En cuanto al proceso de administrar puntos de venta y de generar reportes de rendimiento de los mismos, se realiza en prácticas hojas de cálculo como es Excel en un sin fin de ocasiones, lo cual significa depender enteramente del factor humano, lo cual constituye un grave riesgo para la gestión de datos y en su caso para hacer respaldos o para extraer información útil que beneficie la correcta toma de decisiones al administrar un punto de venta. Así mismo, realizar esta actividad convierte a los procesos de gestión en algo endemoniadamente lento, pues no se tiene automatización alguna al momento de capturar los datos y el proceso es lo bastante sensible al error.

En la actualidad, se tiene una solución integral con un ERP debido a que las empresas cuentan con todos los requerimientos en una sola plataforma, lo que mejora la toma de decisiones al facilitar la utilización de recursos, la lectura de información y las transacciones pendientes por realizar, además de que la mayoría son personalizados y escalables, por lo que el riesgo de quedar devaluados con el tiempo disminuye muchísimo.

Por tanto, un ERP sería mucho más que un software pues, en efecto, aborda los servicios de una empresa en una sola aplicación para organizar un mejor rendimiento, ya que obviamente le da fácil acceso para almacenarlo, verlo en gráficos o informes y, por otro lado, para compartir y manejar temas de eliminación o edición sin perder su enfoque sobre las actividades que se realizarían dentro de la empresa.

El impacto de los ERP en la historia se puede empezar a notar desde la década de los 90 puesto que las empresas utilizaban diversos sistemas para la gestión de sus procesos principalmente estos estaban enfocados en planeación y producción.

Modelos como, el MRP (Material Requirement Planning): Cuya finalidad era asegurar que las materias primas estuvieran disponibles para el proceso de producción optimizando dicho proceso y reduciendo los inventarios.

MRP II (Manufacturing Resourse Planning): El cual es una evolución del MRP y estaba pensado para incluir áreas como la planificación de la capacidad, gestionar la planta y algunos aspectos financieros este modelo era mas enfocado en la gestión,

planeación y uso de todos los recursos que estuvieran involucrados en el proceso de la producción.

Sistemas de contabilidad: Se solía usar software que era enfocado única y específicamente solo para finanzas y la contabilidad esto no tenia nada que ver con los sistemas de producción y/o inventarios, es decir que no estaban cohesionados se tenían que administrar las dos áreas por separado y manualmente juntarlos.

Sistemas de gestión de inventarios: Eran utilizados para el control de productos y materias primas, pero no estaban ligados a otras áreas de la empresa lo cual era un problema para la gestión de stock relacionado a los requerimientos de la producción.

Con estos modelos por separado era difícil una gestión eficiente entre áreas y departamentos de la empresa que repercutía con la gestión global de la empresa.

El ERP surge en los años como solución a una necesidad de conectar la gestión de todas esas áreas en un sistema central ya no era un simple planificador o un gestor de recursos era todo eso de la mano y se volvió un pilar para la organización, para tener decisiones conscientes y un mayor crecimiento, además de que su flexibilidad lo hizo adaptable a cualquier entorno empresarial desde empresas chicas y gigantes puesto a su diseño es modular y escalable.

La globalización y competencia entre empresas en los años noventa hice que la expansión de los ERP fuera impulsada las empresas necesitaban una mayor eficiencia operativa con la capacidad de tomar decisiones basadas en datos concisos. Los ERP ofrecieron una solución integra a las operaciones comerciales que los sistemas anteriores aislados no podían cubrir esta necesidad, otro gran punto a tomar en cuenta en la solidificación de los ERP fue el problema de YK2 ya que las empresas temían que sus sistemas no pudieran hacerle frente al cambio de milenio al utilizar solamente dos dígitos para representar el año por ejemplo “99” para evitar fallos muchos optaron por modernizar sus sistemas para garantizar que sus operaciones no se vieran afectas por

este cambio. El euro puso también de su parte para el cambio puesto que reemplazo a múltiples monedas en Europa y los sistemas financieros de ese entonces no contaban con la preparación para manejar varias divisas o adaptar ese cambio enorme en las finanzas así que vieron en los ERP una oportunidad de no solo solucionar ese problema si no también mejorar la gestión de otras áreas de la empresa con la confianza de tenerlas en una o múltiples monedas en común

Con un sistema que integra procesos automatizados de gestión que a su vez brinda datos de forma preciosa y su acceso a estos datos es fácil para obtener información de los mismos reduce un gran peso de la gestión a los administradores y a los prestadores de servicio y permite que estos puedan enfocarse en la mejora continua y brindar servicios de mejor calidad poniendo en un lugar importante a la atención al cliente. 1.

Tabla 1. Componentes primarios de una ERP

Administración y estrategia	**Contabilidad y Finanzas**	**G**estión de Personal	**Producción y Suministros**
Planificación empresarial	Gestión de activos	Desarrollo de talento	Planificación de la producción
Control de inventarios	Contabilidad general	Gestión horarios	Optimización de recursos (MPR y MPR II)
Ventas y facturación	Auditoría interna	Gestión de nómina	Control de costos de producción
Facturación y ventas	Control de presupuestos	Evaluación del personal	Abastecimiento de materia prima
Gestión de clientes (CRM)	Análisis financiero	Gestión de vacaciones y permisos	Administración de mano de obra

Administración y estrategia	**Contabilidad y Finanzas**	**G**estión de Personal	**Producción y Suministros**
Gestión de proveedores	Gestión de impuestos	Formación y capacitación	Control de Stock de producción
Logística y distribución	Gestión de tesorería	Evaluación del desempeño	Control de calidad de productos
Gestión de proyectos estratégicos	Análisis de costos	Reclutamiento y selección	Gestión de la cadena de suministro
Gestión documental y archivos	Informes financieros	Seguridad y salud laboral	Mantenimiento y soporte técnico

Descripción de la tabla:

- Claridad y Detalle: Las categorías se han renombrado y ordenado para ofrecer mayor claridad en cuanto a las funciones que cada componente del ERP gestiona. Esto permite que los usuarios comprendan rápidamente qué módulos están disponibles y qué áreas del negocio cubren.
- Categorías Adicionales: Se añadieron nuevas subcategorías que reflejan funcionalidades adicionales y modernas que se encuentran en los ERP actuales, como la gestión de proyectos, logística y distribución, y gestión documental. Estos módulos son críticos para muchas empresas que buscan una solución ERP integral.
- Organización Lógica: Los componentes están organizados de manera que los usuarios pueden visualizar cómo cada función se agrupa dentro de áreas claves del negocio, facilitando la localización y la comprensión del propósito de cada módulo.
- Compatibilidad y Flexibilidad: Se refleja la capacidad de los ERP modernos para adaptarse a diversas industrias y necesidades empresariales, proporcionando

soluciones específicas para áreas como la producción, recursos humanos, y la gestión financiera avanzada.

Una de las características principales de los ERP es que se encuentran divididos en módulos los cuales suelen representar los departamentos de la empresa y también enfocarse en los procesos primarios con los que cuenta la organización compuestos en su núcleo por las áreas, administrativa, financiera y la parte operativa.

Puesto a que las empresas cuentan con grandes cantidades de datos los cuales tienen que ser procesados y analizados para obtener una toma de decisiones acertada las empresas tienen la necesidad de contar con un sistema de información.

Una empresa puede tener varias zonas con diferentes usos pueden estar por ejemplo ventas, calendarios, contabilidad, inventarios, personal, los cuales están integrados y centralizados en un ERP con una sola y única aplicación.

EL objetivo final de tener un ERP implementados es poder gestionar de manera interactiva todas las funciones gracias a la centralización y a la formación de la información estratégica. Hay varias razones por las cuales una empresa puede empezar a necesitar de un sistema, así como pueden ser:

- Basarse en un sistema central y sostenible
- Personalizarse dependiendo de las necesidades únicas de la empresa
- Unificar la información y hacer el movimiento de está más fácil
- Facilitar la integración y el manejo de la información tanto dentro como afuera de la empresa
- Ofrecer funciones que permitan una toma de decisiones basada en la coordinación y en el análisis de los datos mas relevantes
- Permitir decisiones mas agiles gracias a la centralización y al fácil acceso a la información clave
- Facilitar el trabajo en conjunto integrando herramientas que permitan la colaboración entre departamentos y optimizar así los procesos internos.

Siguiendo esas pautas se puede pronosticar un crecimiento de la organización a mediano y largo plazo además de tener en cuenta principios clave y fundamentales para cualquier proyecto.

- Dotar de recursos al proyecto (Financieramente, humanamente, infraestructuralmente)
- Tener una idea clara de los procesos de la organización
- Tener una dirección central la cual posea una visión general
- Determinación realista y concisa de objetivos y metas a demás de sus periodos de tiempos correspondientes
- Nombramiento de un administrador de seguimiento a los avances y del proyecto, así como verificar el cumplimiento de compromisos y acuerdos
- Calificar al personal en sesiones donde se le enseñe el uso y beneficio de operar con un ERP

Para que un ERP cuente con una buena repercusión en la empresa es primordial seguir ciertas etapas de implementación.

- Planificación
- Análisis de requerimientos
- Diseño del sistema
- Fase de pruebas
- Capacitación de usuarios
- Implementación y liberación

Planificación:

El punto básico de la planeación es darle al cliente contacto inicial con el sistema para posteriormente definir la planeación final del proyecto y dentro de estq etapa hay diversas actividades que se deben realizar para una planificación solida del ERP:

- Darle a conocer al cliente al grupo de trabajo que llevara a acabo la implementación del sistema
- Definir tiempos del proyecto a menudo basado en un cronograma al que todos tendrán acceso para así ajustar tiempos con todos los organismos que estarán involucrados con el proyecto
- Crear un registro de compromisos iniciales para que el proyecto arranque sin posibles problemas
- Definir los formatos de procesos en la empresa, así como los datos iniciales que se ingresaran al sistema

Análisis de requerimientos:

Durante esta fase la tarea primordial es obtener un listado de requerimientos y funcionalidades previamente acordadas de cada uno de los departamentos de la organización para la implementación el mencionado listado debe ser validado por un supervisor o encargado de área y con todos los vistos buenos se puede proceder a la configuración de la solución con base a los procesos y procedimientos definidos.

Las tareas a llevar a cabo durante esta etapa son:

- Definir específicamente alcances del proyecto
- Elaborar diagramas de flujo de los procesos y procedimientos de los departamentos

Diseño:

En esta etapa de implementación de un ERP se debe realizar la configuración de la solución de acuerdo con el proceso, procedimientos y requerimientos que se definieron en la etapa anterior de análisis de requerimientos. Aquí la comunicación entre el personal que se encuentra a cargo de la implementación mantenga un estrecho acercamiento con el cliente final para resolver cualquier duda que pueda emerger en el proceso ya que en esta etapa se pueden ir palpando las soluciones a las necesidades de la empresa.

Fase de pruebas:

Los procedimientos realizados en esta etapa ligados puramente a funcionalidad haciendo pruebas con casos de uso realistas identificando problemas y exponiendo soluciones a los mismos, es importante monitorear las pruebas ya que estas darán información relevante para el correcto funcionamiento del proyecto.

Capacitación del personal:

Una vez realizadas las pruebas y mejoras, se procede a capacitar a todos los usuarios finales, aquí la motivación y participación de todos los involucrados es muy importante ya que es cuando va a tener la experiencia real con el ERP y sobre todo se preparan para dominar el uso de la solución. La comunicación total con el consultor es muy relevante en este paso, se deben dar a conocer las dudas en cuanto a procesos y funciones, esto con el fin de obtener el máximo provecho de la herramienta [29]. Al finalizar la capacitación, se deberá asegurar de encargar a cada usuario que realice operaciones diarias de práctica en el sistema, desde ese día de práctica en el sistema, desde ese día hasta la liberación.

Cunado el sistema se haya probado y en dado caso de necesitarlo haberse mejorado se procede a la capacitación de los usuarios, este es el acceso real al sistema ERP de cara a exponerlo a situaciones de cara al cliente por lo cual la correcta participación del personal es importante para dominar el uso del sistema.

El encargado debe también poder responder a los cuestionamiento e inquietudes que tenga el usuario en cuanto a los procesos y funcionalidades para aprovechar el sistema en su totalidad y se debe convencer a los usuarios a conocer a fondo al sistema haciendo pruebas practicas con simulaciones realistas para llegar con una base sólida de uso del sistema cuando sea el momento de realizar la liberación de producto.

Al concluir estas etapas se debe contar con un ERP que abarque todas las necesidades del cliente y que tenga una eficiencia notoria en los resultados, para lograr una mejor operatividad y poder reducir cosas en la empresa, además de ayudar a los empleados a tener un mejor servicio al cliente.

3. Diseño y análisis de la plataforma para la gestión

El desarrollo de una plataforma para la gestión eficiente de puntos de venta en el contexto empresarial es un proceso crucial para mejorar la operatividad y competitividad de las empresas en el mercado actual. Esta plataforma se enfoca en centralizar y optimizar diversas funciones críticas, como la administración de inventarios, la gestión de usuarios y roles, el procesamiento de transacciones y la generación de informes, entre otros aspectos fundamentales para el éxito de los puntos de venta.

Una de las principales ventajas de esta plataforma es la capacidad de centralizar la administración de múltiples puntos de venta en una única interfaz, lo que facilita el control y la supervisión de todas las operaciones desde un solo lugar. Esto no solo permite una mayor visibilidad y coherencia en las operaciones diarias, sino que también mejora la toma de decisiones al proporcionar datos en tiempo real y análisis detallados de las ventas, inventarios y comportamiento del cliente.

La plataforma también integra tecnologías avanzadas como aplicaciones web y móviles, lo que permite a los administradores y empleados acceder a las funciones del sistema desde cualquier dispositivo con conexión a Internet. Esta flexibilidad es especialmente importante en el entorno empresarial moderno, donde la movilidad y la accesibilidad son clave para mantener la eficiencia operativa y responder rápidamente a las demandas del mercado.

Además, el desarrollo de la plataforma incluye la implementación de medidas de seguridad robustas para proteger la información confidencial y garantizar la integridad de los datos. Esto es fundamental en un contexto donde el cumplimiento normativo y la protección de la privacidad son cada vez más importantes para las empresas.

Otro aspecto destacado es la personalización y escalabilidad de la plataforma. La capacidad de adaptarse a las necesidades específicas de cada empresa y de crecer junto con ella es esencial para asegurar que la solución siga siendo relevante y efectiva a medida que el negocio evoluciona.

En [35] se menciona que el desarrollo de una plataforma para la gestión eficiente de puntos de venta en el contexto empresarial no solo mejora la operatividad diaria, sino que también proporciona una base sólida para el crecimiento y la innovación. Al centralizar la administración, mejorar la seguridad, y ofrecer flexibilidad y escalabilidad, esta plataforma se convierte en una herramienta indispensable para cualquier empresa que busque mantenerse competitiva en un mercado en constante cambio.

3.1 Análisis de la plataforma para la gestión y control de ventas

Un punto de venta cuenta con una cartera de usuarios diversa y al ser un sistema centralizado quiere decir que cada uno de los diversos perfiles de usuario va a tener acceso a la plataforma por lo cual es paritario segmentar los usuarios en roles como pueden ser (Administrador, cajero, cliente, promovedor, etc.) con esta definición lo siguiente es decidir que partes mostrar a cada rol el administrador central debe tener acceso a todos los usuarios es el perfil que se asemeja a un llave maestra que tiene control sobre todas las áreas con el fin de poder ver como operan entre si además de monitorear los procesos y obtener estadísticas basadas en la información recopilada de los datos, pero otros perfiles deben estar limitados dependiendo del área a la que pertenezcan esto lo logramos con autentificación y uso de sesiones con Next auth y un contexto de sesiones global que contine un rol de usuario, esto además de tener privacidad de procesos externos al área permite una mejor visión para desempeñar labores ya que no tiene visión de departamentos en lo que no incide es decir tiene lo justo y necesario para poder realizar su trabajo, no se le quitan responsabilidades ni se le aumentan. Esta gestión tradicionalmente requería de presencialidad para obtener

autenticación lo cual podría resultar con inconvenientes en términos de tiempo y desplazamiento. Con un ERP que permite la creación de sistemas modulares y escalables, capaces de integrar nuevas funcionalidades sin necesidad de rediseñar todo el sistema desde cero. Por ejemplo, un diseño atómico permitiría implementar un módulo de autorregistro en línea, donde los usuarios podrían solicitar permisos y roles para su sesión, reduciendo significativamente la necesidad de visitas presenciales a las oficinas administrativas.

Otro servicio esencial en estos puntos de venta son las cajas, que incluyen partes como la sucursal en la cual se encuentra la caja, la cantidad con la que se abre y las operaciones realizadas como ventas, cobros, pagos de servicios etc., con el sistema de ERP el administrador puede monitorear la caja desde cualquier lugar además de cerrarla o abrirla evitando ir hasta la sucursal y obteniendo informes de manera inmediata en tiempo real.

Continuando con los servicios indispensables se encuentran los inventarios realizar operaciones entre sucursales o en la misma sucursal podría ser un gran desafío si no se cuenta con un ERP pero aquí se garantiza una gestión eficiente para agregar nuevos productos ya sea manual o importados de archivos .XLSX o .XLS, eliminar existentes, hacer traspasos dentro de la misma sucursal de un inventario hacia un almacén y tener la capacidad de generar en segundos un listado en formato de Excel de los productos existentes en cualquiera de las ubicaciones ya sea inventario o almacén. Pero estas funcionalidades no solo están limitadas hacia una misma sucursal si no que el administrador posee la capacidad de mover productos entre sucursales a sus dos ubicaciones a inventario o almacén y desde cualquier ubicación también, esto ayuda muchísimo a la hora de registrar esos movimientos automáticamente se hace el alta de productos y la baja sin necesidad de otros procesos de inventariado eso sin contar que hay filtros para poder seccionar entre productos puede ser que como en este caso haya productos con diferentes tipos por ejemplo celulares y accesorios y así poder tener una visión más clara de que se está obteniendo.

3.2 Recursos y Diseño atómico

Para el consumir las consultas y ejecutar las mutaciones de GraphQL usaremos Next js el cual es un framework que usa la librería de React js que es una librería basada en JavaScript esta librería basa su construcción de interfaces de usuario en componentes los cuales pueden reutilizarse para varias funciones o que cumplan la misma función en diferentes contextos ya que tienen estados y propiedades que podemos añadirle por defecto o si es el caso pasar de un componente padre al componente hijo así podemos gestionar de mejor manera la lógica y estructura del proyecto.

Next js nos agrega renderizado por parte del servidor con lo cual mejora el rendimiento de las aplicaciones, además de ciertas facilidades como enrutamiento automático con el uso de la carpeta pages, creación de endpoints con API ROUTES y optimización de imágenes por defecto.

Funcionando juntos nos ofrecen por parte de React: una mejor gestión de la lógica y la construcción de la interfaz de usuario y next js nos da enrutamiento, renderizado por parte del servidor y optimiza el rendimiento, siendo así un proyecto mejor gestionado, eficiente y escalable.

Para la construcción de la interfaz de usuario se optó por la metodología de diseño llamada Atomic desing con la cual podemos estructurar y organizar los componentes que tenemos en la interfaz de usuario además que como React es una librería basada en componentes nos permite muchas facilidades para su implementación, la metodología tiene su base en dividir los componentes por métodos jerárquicos siguiendo fundamentos biológicos para crear elementos pequeños o básicos e ir conjuntando para así crear interfaces o paginas completas.

Se comienza por los átomos que son la base del proyecto son los bloques más básicos de la interfaz de usuario que pueden encapsular botones, etiquetas, títulos,

inputs, etc. Estos no pueden descomponerse más, pero si pueden recibir parámetros o lógica de funcionamiento de sus componentes mayores, estos componentes tienden a ser reutilizados muchas veces por lo cual deben ser muy flexibles a la hora de su diseño y definición de la lógica que utilizaran.

Con los átomos podemos crear moléculas que son nuestro segundo escalón, si bien esta combinación es un componente funcional más grande se sigue teniendo modularidad es decir que puede ser útil en varios contextos como lo puede ser un formulario de búsqueda de texto conformada por un input de texto y un botón o un botón doble para selección.

Estos dan paso a los organismos los cuales empiezan a tener una funcionalidad más específica como podría ser un encabezado conformado por nuestra molécula de búsqueda de texto, titulo y subtítulos y botones.

Las estructuras siguientes son más grandes y combinan diversos organismos, son llamadas plantillas, aquí el diseño debe tener una jerarquía visual y estructura de contenido que va a llevar la página, como puede ser un encabezado, una sección de contenido, una barra lateral pero aun sin contenido.

Las platillas serán llamadas en las páginas la cuales son las estructuras más grandes que tenemos en la jerarquía del diseño atómico, y en ellas se tiene el diseño final de la página, como puede ser un dashboard ya completo y funcional incluyendo rutas protegidas, sesiones activas.

Esta metodología de diseño tiene muchas ventajas como son el modularidad es decir que los componentes puedan ser reutilizados en toda la aplicación además que esto nos da una línea grafica a seguir, también nos aporta mucho en el tema escalabilidad porque nos da un orden y podemos manejar los componentes más fácilmente sin crearlos de cero o tener código repetido múltiples veces es tan fácil como

llamara al componente y cambiar sus propiedades (props) o su lógica. Con esto en mente se diseña e implementa para la creación de nuestra interfaz.

La aplicación propuesta ofrecerá a los usuarios un amplio catálogo de servicios, con el objetivo de facilitar la gestión y acceso al punto de venta y a las diversas funcionalidades primordiales de su gestión. Además de integrar microservicios para facilitar el uso del punto de venta como ventas rápidas y gastos recurrentes para automatizar los servicios y sea más cómoda la administración de estos puntos de venta.

El objetivo principal del sistema desarrollado se basó en alinear las necesidades específicas del negocio con las capacidades técnicas reales del sistema, asegurando que cada funcionalidad estuviera diseñada para cumplir con los requisitos operativos de manera efectiva y eficiente. Una de las fases más críticas en este proceso fue la identificación precisa y exhaustiva de las necesidades del negocio. Se priorizo en detallar las necesidades particulares de un punto de venta, con el fin de asegurar un servicio de alta calidad los usuarios del punto de venta, y a su vez optimizar y hacer más fácil las tareas de un administrador para hacerlas más eficientes y fáciles de usar para cualquier rol que desee usar el sistema.

Una de las funcionalidades fundamentales implementadas en el ERP fue un sistema robusto de autenticación de usuarios. Este sistema fue crucial para mantener un control riguroso sobre las personas que tendrían acceso a los servicios que ofrece un punto de venta. La autenticación no solo garantizó que solo usuarios autorizados pudieran acceder a información y recursos críticos, sino que también permitió personalizar las experiencias de los usuarios según su rol dentro del sistema.

El sistema fue diseñado para gestionar diferentes niveles de acceso a la información, los cuales dependieron del tipo de usuario. Los usuarios se clasificaron en varias categorías, tales como supervisor, administrativo, propietario, residente o proveedor de servicios. Cada uno de estos roles tuvo acceso diferenciado a la información y funcionalidades del sistema, asegurando que los usuarios solo

interactuaran con las partes del sistema que fueran relevantes para sus responsabilidades y necesidades específicas.

En resumen, el desarrollo de este sistema no solo se orientó a cumplir con las exigencias técnicas, sino que también se centró en asegurar que estas soluciones técnicas estuvieran profundamente alineadas con las necesidades del negocio, lo que permitió ofrecer un servicio óptimo y adaptado tanto para los residentes como para las oficinas administrativas. Esta alineación estratégica entre las capacidades del sistema y las demandas del negocio fue esencial para el éxito a largo plazo de la gestión de los puntos de venta.

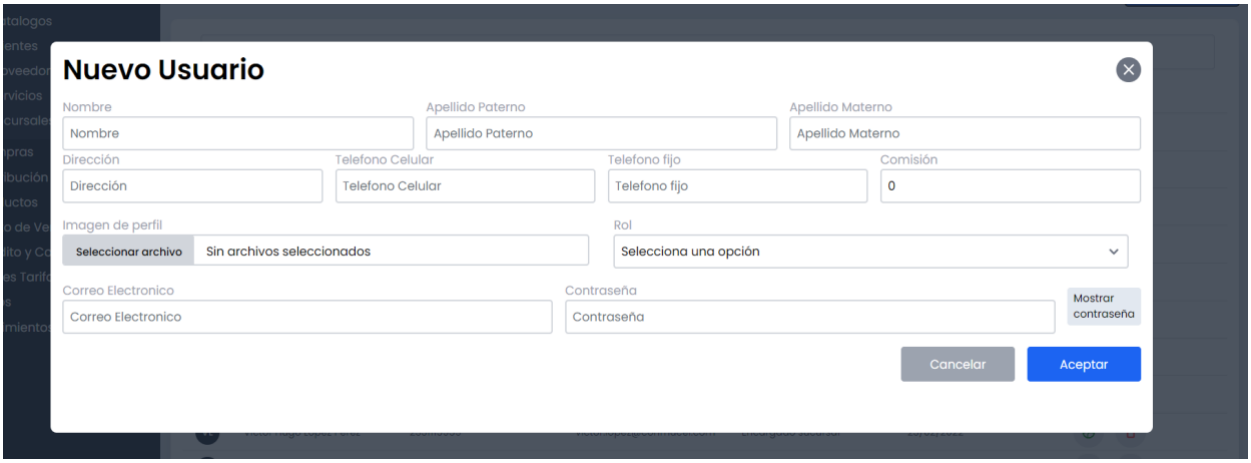

Fig. 2. Módulo de usuarios

Fig. 2. Permisos de usuarios

3.3 Caja

Los puntos de venta necesitan una caja la cual pueda realizar apertura ya sea con cierto monto o sin efectivo, una gestión de caja para visualizar los movimientos efectuados y un cierre de caja con retiro de efectivo para realizar el corte.

Para la caja se cuenta con un método de apertura el cual te solicita un monto para poder acceder a la caja este puede ser 0 o ingresar un monto, al tener la caja abierta podrás efectuar operaciones como ventas, apartar artículos, devolver productos, activar o desactivar garantías de equipos telefónicos.

La caja tendrá un estado global en el header del layout el cual te indicará cuando la caja esté abierta o cuando la caja esté cerrada, si la caja esta abierta podrás ver el saldo, los gastos y los montos obtenidos divididos por operación además cuando sea cierre de turno te pedir el saldo a retirar para efectuar el corte de caja.

Cabe resaltar que si el cajero inicia sesión solo accederá a su caja pero si lo hace el administrador podrá acceder a la caja de cualquier punto de venta dentro de la organización.

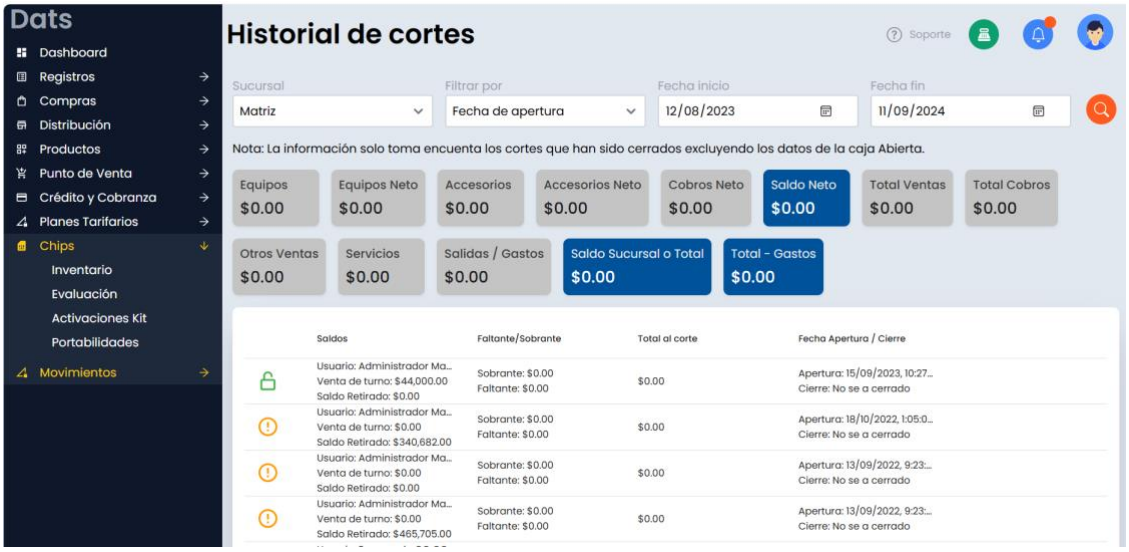

Fig. 3. Módulo de Caja.

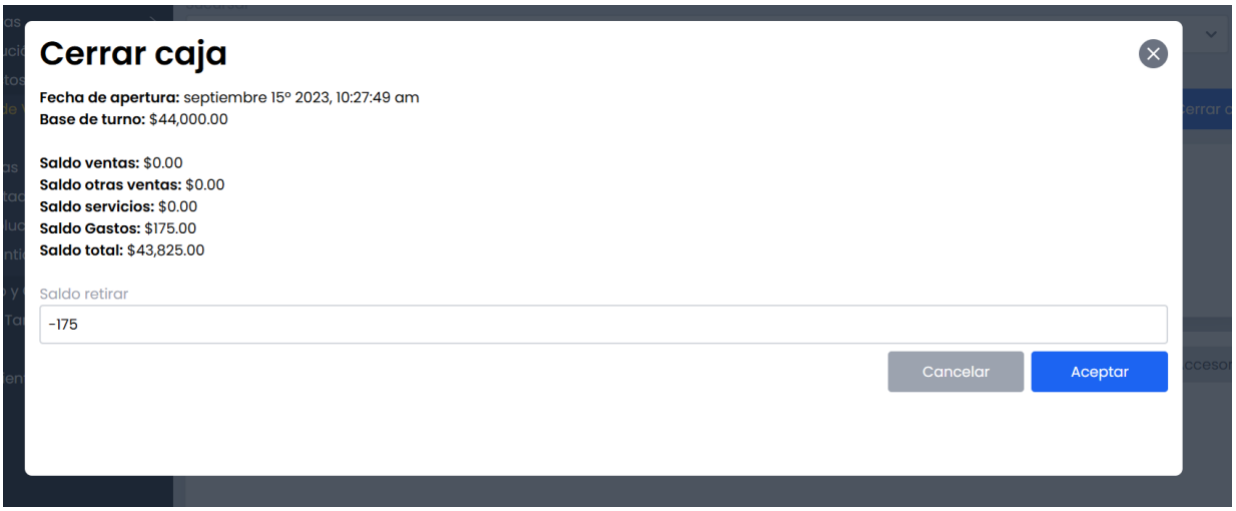

Fig. 3.1 Cierre de Caja.

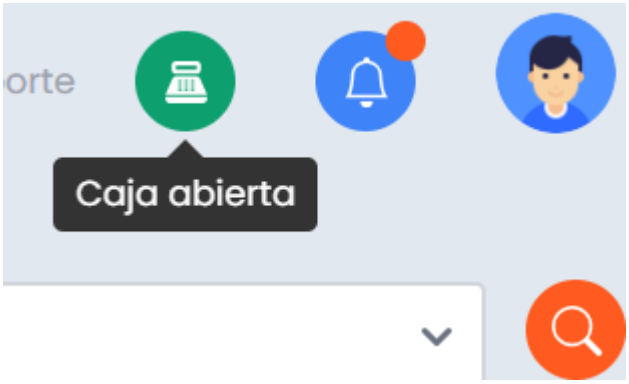

Fig. 3.2 Estatus de Caja.

3.4 Planes tarifarios

Esta sección del punto de venta se basa en contratos para planes de tarifas de cobro de celulares esto facilita la creación de contratos y hace que el cliente tenga un plan en un tiempo corto, esto permite gestionar contratos y que sea visible para el vendedor y para el cliente las fechas, precios, equipos y demás información que se necesite obtener para ese contrato, se tiene una lista en los que puedes filtrar por status Activo e inactivo y a los inactivos los puedes reactivar con solo un botón al seleccionarlo, también tienes planes recurrentes que son una plantilla para facilitar la creación y gestión de estos mismos.

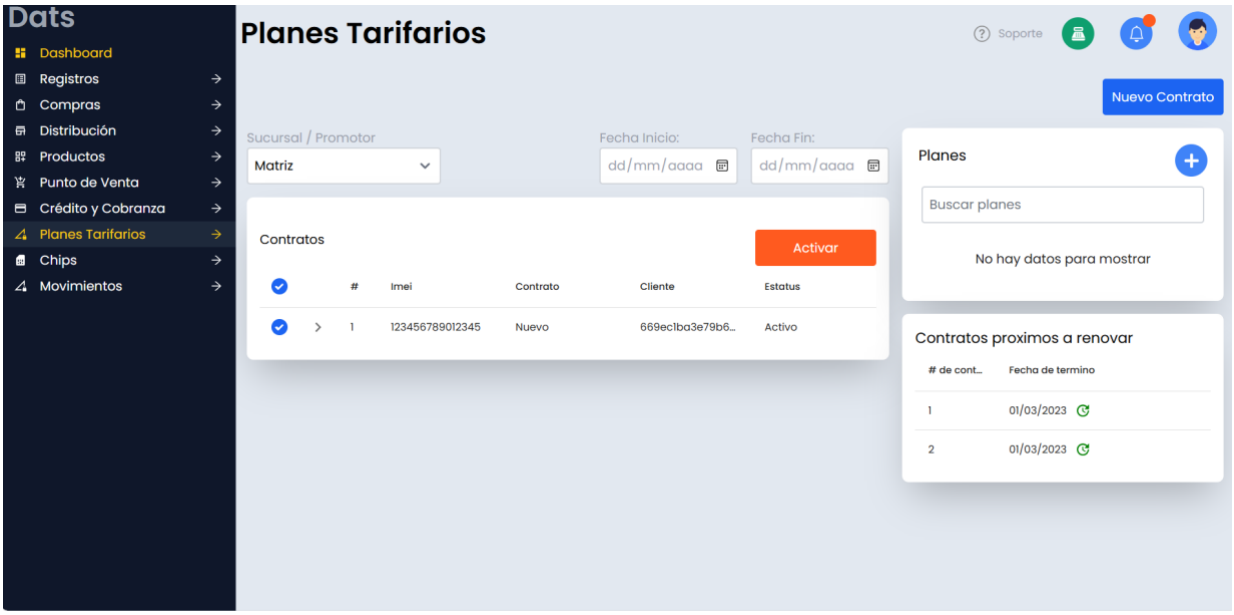

Fig. 4. Módulo de moléculas.

3.5 Información acerca de los servicios

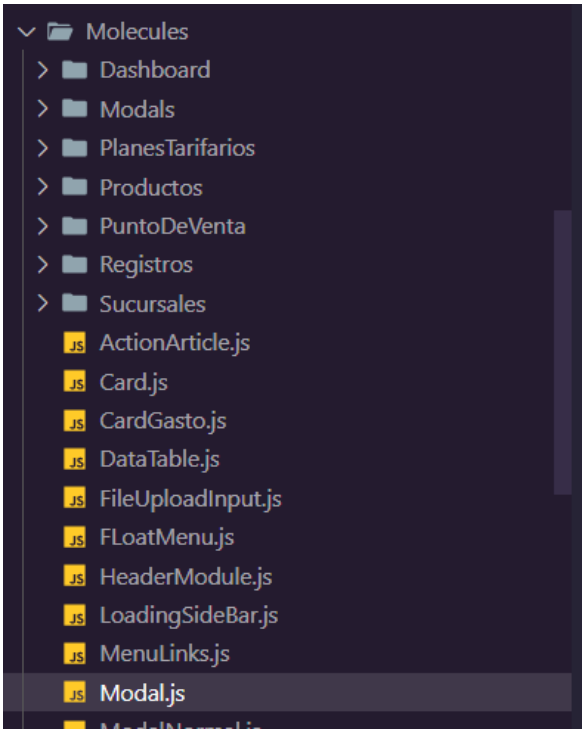

Fig. 4. Moléculas -modal

El archivo Modal.js define un componente React denominado Modal, que está diseñado para desplegar una ventana modal, una interfaz de usuario comúnmente utilizada para diálogos, notificaciones o formularios que requieren la interacción del usuario. El diseño y funcionalidad de este componente se centran en la presentación clara de la información, así como en la interacción intuitiva para el usuario.

Estructura del Código

El código comienza con la importación de las bibliotecas y componentes necesarios:

- **React:** La biblioteca base para la construcción de interfaces de usuario en JavaScript.
- **MdClose:** Un ícono de cierre proporcionado por la librería `react-icons`, que se utiliza para permitir al usuario cerrar la ventana modal.
- **Title:** Un componente personalizado, que se importa desde el directorio `Atoms`. Presumiblemente, este componente maneja la presentación del título dentro de la interfaz.

Definición del Componente

El componente Modal es una función de tipo **funcional** en React, lo que significa que no tiene estado propio (stateless) y recibe sus propiedades (props) como argumentos. Las principales propiedades que recibe son:

- **`title`:** El título del modal, que se muestra en la parte superior del cuadro de diálogo.
- **`children`:** El contenido que se mostrará dentro del modal. Este contenido es flexible y puede incluir cualquier elemento React pasado como hijo.
- **`onClose`:** Una función que se invoca cuando el usuario hace clic en el botón de cierre del modal, permitiendo su cierre.

Estructura de JSX

El componente Modal devuelve una estructura JSX que define la disposición y estilo del modal:

- **Contenedor Externo (** **):**
- Un elemento de posición fija que cubre toda la pantalla (w-screen h-screen), asegurando que el modal se superponga al contenido existente.
- El modal se centra en la pantalla tanto vertical como horizontalmente mediante las clases de utilidad de flexbox (flex items-center justify-center).
- Se aplica un fondo semitransparente oscuro (bg-gray-800 bg-opacity-80) para crear un efecto de fondo desenfocado, que dirige la atención del usuario al contenido del modal.
- La propiedad z-30 asegura que este contenedor tenga una prioridad de visualización alta, superponiéndose a otros elementos.
- **Contenedor de Contenido del Modal:**
- Un contenedor blanco, con bordes redondeados (bg-white rounded-lg), que incluye un relleno (p-4) para la disposición interna.
- La anchura y altura del contenedor se ajustan de manera responsiva (w-[95%] md:w-5/6), asegurando que el modal se vea adecuadamente en diferentes tamaños de pantalla.
- Se utiliza overflow-auto para permitir el desplazamiento dentro del modal si el contenido excede la altura disponible, manteniendo así la usabilidad en todas las circunstancias.
- **Sección de Encabezado:**
- Contiene el título del modal, que se muestra utilizando el componente Title.
- Un botón de cierre, que incluye el ícono MdClose, permite al usuario cerrar el modal. Este botón está estilizado con un fondo gris, efectos de hover y un diseño redondeado, asegurando que sea fácilmente identificable y accesible.
- **Sección de Contenido:**
- Esta sección es flexible y se utiliza para mostrar cualquier contenido pasado como children, lo que permite que el modal tenga múltiples usos, desde formularios hasta visualización de información.

Funcionalidades y Características Clave

- **Diseño Responsivo:** El modal está diseñado para adaptarse a diferentes tamaños de pantalla, utilizando clases de TailwindCSS que ajustan la anchura y altura de manera dinámica. Esto asegura que el modal sea accesible en dispositivos móviles y de escritorio.
- **Accesibilidad:** El botón de cierre es accesible y presenta un cambio visual en respuesta a la interacción del usuario, mejorando la usabilidad.
- **Flexibilidad:** El uso del prop children permite que el contenido del modal sea altamente personalizable, adaptándose a diversas necesidades dentro de la aplicación.

Buenas Prácticas y Calidad del Código

- **Reusabilidad del Componente:** El `Modal` está diseñado para ser reutilizado en múltiples partes de la aplicación, con una estructura que permite personalizar tanto su contenido como su comportamiento.
- **Consistencia en la UI:** Al utilizar un componente personalizado como `Title` y un ícono estándar de `react-icons`, el diseño del modal se mantiene consistente con el resto de la aplicación.
- **Uso de Librerías Externas:** La elección de `react-icons` facilita la incorporación de íconos en la aplicación, siguiendo las mejores prácticas de modularidad y mantenibilidad.

Posibles Mejoras

- **Mejoras en Accesibilidad:** Añadir atributos ARIA, como `role="dialog"`, `aria-labelledby`, y `aria-describedby`, mejoraría la accesibilidad para los usuarios que dependen de tecnologías asistivas.
- **Cierre por Click en el Fondo:** Incluir una funcionalidad que permita cerrar el modal haciendo clic en el fondo oscuro podría mejorar la experiencia del usuario, haciéndola más intuitiva.
- **Efectos de Transición:** Agregar transiciones suaves cuando el modal aparece o desaparece podría hacer que la interfaz se sienta más dinámica y moderna.

El componente Modal en ModalMolecule.js es un ejemplo de un componente molécula del diseño atómico en React que maneja de manera eficiente la visualización de ventanas modales en una aplicación web. Su diseño responsivo, junto con su enfoque en la reusabilidad y flexibilidad, lo hacen adecuado para diversas situaciones dentro de la aplicación. Si bien el componente cumple con su propósito, hay margen para mejoras en accesibilidad y experiencia de usuario que podrían elevar aún más su calidad.

Este componente es una parte esencial de la interfaz de usuario, ya que facilita interacciones críticas, como la captura de información del usuario y la visualización de mensajes importantes además es usado muchas veces como intermediario para una acción como confirmación. Es un activo valioso en la construcción de interfaces de usuario modernas y accesibles.

```
import React from "react";
import { MdClose } from "react-icons/md";
import Title from "../Atoms/Title";

const Modal = ({ title, children, onClose }) => (
  <div className="◦◦◦">
    <div className="◦◦◦ ">
      <div className="◦◦◦">
        <Title value={title}>{title}</Title>
        <button
          type="button"
          className="◦◦◦"
          onClick={onClose}
        >
          <MdClose size={24} />
        </button>
      </div>
      {children}
    </div>
  </div>
);

export default Modal;
```

Fig. 5. Buenas Prácticas y Calidad del Código.

3.6 Notas de crédito

El archivo FilterNotasOrganism.js (ver Fig. 6) define un componente React denominado FilterNotas, el cual está diseñado para ofrecer funcionalidades de filtrado y búsqueda dentro de un conjunto de notas o registros. Este componente es importante en la interfaz de usuario, permitiendo a los usuarios refinar y buscar información específica mediante varios criterios, tales como estado, fechas y texto de búsqueda.

Estructura del Código

El código (ver Fig. 5) comienza con la importación de varias bibliotecas y componentes esenciales:

- **React y useState:** La base del código se construye sobre React, utilizando el hook `useState` para manejar estados internos del componente.
- **BsSearch:** Un ícono de búsqueda proporcionado por la librería `react-icons`, que se utiliza como elemento visual para representar la acción de buscar.
- **Componentes personalizados:** Se importan varios componentes de otros módulos del proyecto, como `Input`, `IconButton`, y `Select`, que se utilizan para construir la interfaz de usuario del componente `FilterNotas`.
- **Formik y Form:** Formik se utiliza para manejar formularios dentro de React, proporcionando una forma sencilla y eficaz de gestionar formularios complejos con validación y manejo de estados.

Definición del Componente

El componente FilterNotas es una función de tipo **funcional** que recibe varias propiedades (props) para controlar los estados y eventos asociados con el filtrado y la búsqueda. Las propiedades que se pasan al componente son:

- **setSelectedStatus:** Una función para establecer el estado seleccionado.
- **setSelectedStartDate:** Una función para establecer la fecha de inicio seleccionada.
- **setSelectedEndDate:** Una función para establecer la fecha de fin seleccionada.
- **setSearchText:** Una función para establecer el texto de búsqueda.
- **selectedStartDate y selectedEndDate:** Estados actuales de las fechas seleccionadas.

Estados Internos y Manejo de Eventos

Dentro del componente, se definen varios estados internos mediante useState:

- **`opciones`:** Un arreglo de opciones para el filtrado por estado, que incluye valores como "Pendiente", "Pagada", y "No pagada".
- **`search`:** Un estado que maneja el texto de búsqueda ingresado por el usuario.
- **`selectedOption`:** Un estado que guarda la opción de estado actualmente seleccionada.

El componente maneja varios eventos importantes:

- **`handleStatusChange`:** Esta función se encarga de actualizar la opción de estado seleccionada, tanto internamente como en el componente padre a través de `setSelectedStatus`.
- **`handleBuscar`:** Función que maneja la actualización del texto de búsqueda, actualizando tanto el estado interno `search` como el estado externo `setSearchText`.
- **`handleStartDateChange` y `handleEndDateChange`:** Funciones encargadas de manejar los cambios en las fechas de inicio y fin respectivamente.

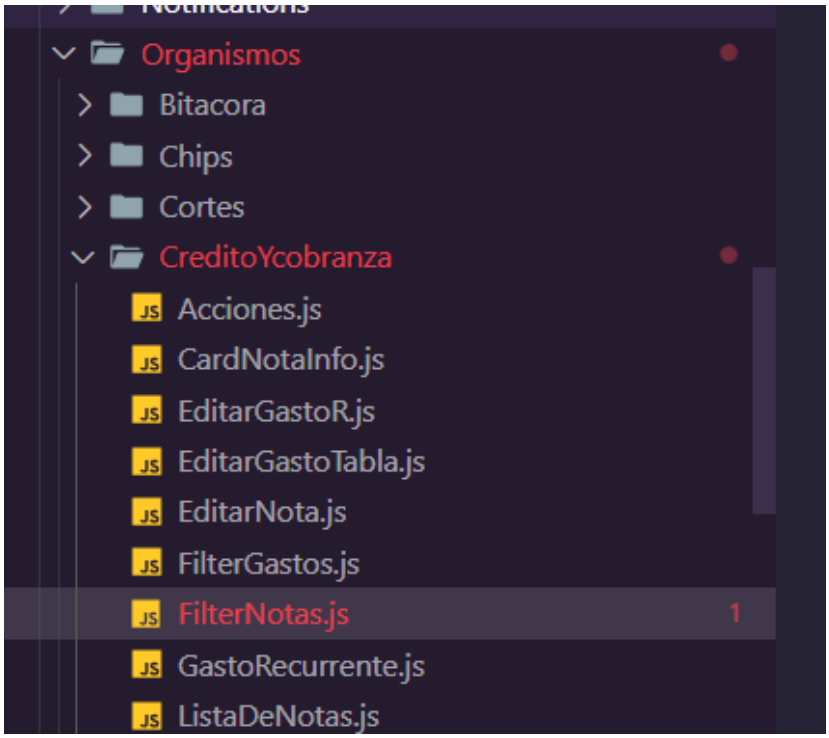

Figura 6. El archivo FilterNotasOrganism.js.

JSX y Estructura de la Interfaz de Usuario

El componente FilterNotas utiliza JSX para definir la estructura visual que se renderiza en la interfaz de usuario:

- **Formulario con Formik:** Formik se utiliza para manejar el formulario principal, que incluye los campos de filtro por estado, fechas, y búsqueda de texto.
- **`Input` y `Select`:** Estos componentes personalizados forman la base de la entrada de datos, ofreciendo campos para la búsqueda y selección de criterios.

- **Botón de búsqueda (`IconButton`):** Se utiliza un botón con ícono para permitir al usuario ejecutar la acción de búsqueda. Este botón está estilizado con el ícono `BsSearch`.

Funcionalidades Clave

- **Filtrado por Estado:** Permite al usuario seleccionar un estado específico (Pendiente, Pagada, No pagada) para filtrar las notas o registros mostrados.
- **Búsqueda de Texto:** Ofrece una barra de búsqueda para que el usuario pueda filtrar los registros basados en palabras clave.
- **Filtrado por Fecha:** Los usuarios pueden seleccionar un rango de fechas para restringir los resultados mostrados dentro de un periodo específico.
- **Interfaz Intuitiva:** El uso de componentes visuales como `Select`, `Input`, y `IconButton` hace que la interacción del usuario sea simple y directa, mejorando la experiencia de usuario.

Buenas Prácticas y Calidad del Código

- **Modularidad y Reusabilidad:** El componente `FilterNotas` está estructurado para ser modular y reutilizable. Utiliza componentes personalizados (`Input`, `Select`, etc.) y maneja eventos y estados de forma encapsulada.
- **Uso de Formik:** La implementación de Formik para el manejo de formularios es una práctica recomendada en React, ya que simplifica la gestión de formularios complejos y reduce el código repetitivo.
- **Manejo de Estado con `useState`:** El uso de `useState` es apropiado para manejar los estados locales del componente, lo que permite una interacción fluida y dinámica con la interfaz de usuario.

Posibles Mejoras

- **Validación Adicional:** Aunque Formik facilita la validación de formularios, sería beneficioso implementar validaciones adicionales en los campos de entrada, especialmente para las fechas, para evitar entradas no válidas o inconsistentes.
- **Optimización de Rendimiento:** Considerar el uso de `useMemo` para memoizar las opciones de filtrado podría mejorar el rendimiento si estas opciones se vuelven más complejas o si el componente se renderiza frecuentemente.
- **Mejoras de Accesibilidad:** Incluir atributos ARIA adicionales para mejorar la accesibilidad del formulario para usuarios con discapacidades.
- **Internacionalización:** Para aplicaciones que se despliegan en múltiples regiones, sería recomendable implementar soporte para varios idiomas, particularmente en los textos de las opciones y etiquetas.

```
const FilterNotas = ({
  return (       You, 14 months ago via  PR #18 • organism Credito Y C
    <div>
      <div className="°°°">
        <div className="°°°">
          <div className="°°°">
            <Input
              placeholder="Número de material / Imei"
              value={search}
              onChange={handleBuscar}
            />
          </div>
          <div className="">
          <IconButton busqueda={true}/>
          </div>
        </div>
        <div>
          <Formik>
            {() => {
              return (
                <Form className="°°°">
                  <Select
                    options={opciones}
                    value={selectedOption}
                    onChange={handleStatusChange}
                    placeholder="Status"
                  />
                  <Input
                    type="date"
                    value={selectedStartDate}
                    onChange={(date) => handleStartDateChange(date)}
                  />
                  <Input
                    type="date"
                    value={selectedEndDate}
                    onChange={(date) => handleEndDateChange(date)}
                  />
```

Figura 7. Fichero FilterNotasOrganism.js

El componente FilterNotas en FilterNotasOrganism.js (ver Fig. 7) es un ejemplo de un componente React siendo un organismo en el diseño atómico y eficaz para manejar filtrado y búsqueda en una aplicación. Su estructura modular, el uso de librerías como Formik para simplificar la gestión de formularios, y la implementación de estados mediante useState demuestran un enfoque sólido en la creación de interfaces de usuario interactivas y eficientes.

Este componente no solo facilita la interacción del usuario con el sistema, sino que también mejora la experiencia general al permitir a los usuarios encontrar y filtrar información relevante rápidamente. Con algunas mejoras adicionales en validación, rendimiento y accesibilidad, este componente podría convertirse en una herramienta aún más poderosa dentro del conjunto de herramientas de una aplicación web.

Este enfoque integral y bien pensado hacia el desarrollo de componentes de filtrado demuestra una profunda comprensión de las necesidades del usuario final y una

capacidad técnica para implementar soluciones que no solo cumplen con los requisitos, sino que también mejoran significativamente la usabilidad del sistema.

3.7 Crédito y cobranza

Se necesita una sección que lleve control de las diversas transacciones financieras, en consecuencia, se creará un sistema que ofrezca los siguientes módulos.

3.7.1 Submódulos

Gatos:

Será un modulo encargado de gestionar los gatos que se tienen dentro de la organización como pueden ser agua, luz, internet, independientemente de la caja esto solo serán gastos que no tengan que ver con el las transacciones del punto de venta como tal ya que esos se gestionan en la caja de caja punto de venta.

Gastos recurrentes:

se deben optimizar tiempos y por lo tanto los gastos recurrentes son gastos que se tiene registrados para efectuarse cada cierto tiempo y con un precio que puede o no variar estos se registran a través de un modal y se listan a un lado del layout para ser más accesibles, se pueden pagar con un simple botón, editar, o eliminar además se pueden filtrar por nombre, descripción, costo, fecha o recurrencia.

Tabla de pagos:

El módulo de pagos tiene una tabla la cual contiene información de los gastos efectuados tiene columna de descripción, costo, fecha estos gastos son importantes porque se mostrarán por sucursal con esta información se puede deducir en que se gasta mas o menos y en cual sucursal se gasta más, también puede editarse un pago realizado, pero solo por el administrador general el cual también tiene la potestad de eliminar algún gasto.

Notas de crédito:

Es un módulo importante ya que se podrán encontrar las notas de crédito en un dashboard con notas a favor y en contra, si hay notas en contra se tendrá secciones que muestren el total de notas en contra con IVA y sin IVA, además de tener un filtro de proveedores para mejor gestión de las notas de crédito.

Filtro de notas:

Además del filtro de proveedores antes mencionado se tendrá un filtro por el cual se puede buscar por IMEI de algún equipo asignado a la nota de crédito o el id de algún producto, un select para filtrar si la nota esta pendiente, pagada o No pagada además de filtrar las notas de crédito por fecha.

Creación de nota de crédito:

Se creará una nueva nota de crédito a la cual deberá asignarse un estatus, IMEI un número de cuenta, numero de celular, fecha, monto con IVA y sin IVA el proveedor que es importante ya que viene de una tabla con los proveedores registrados en el sistema, un nombre de cliente sacado a si mismo por la cartera de clientes registrados, un tipo de nota ya sea a favor o en contra y un motivo.

Para almacenar la información se requería que el usuario administrador defina los catálogos para la cobranza como se expresa en la figura 8.

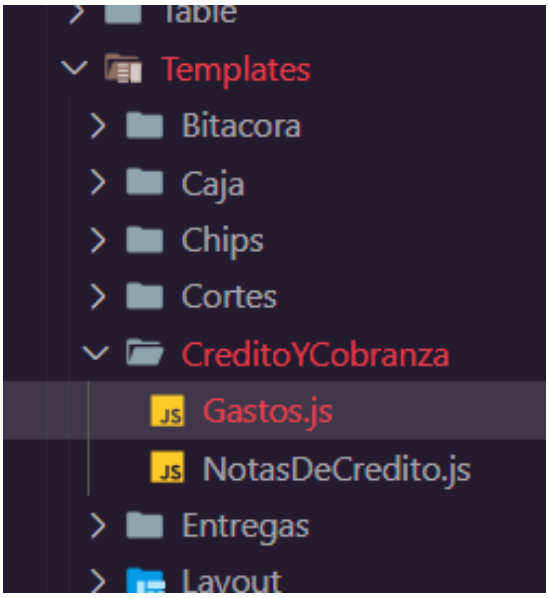

Fig. 8. Módulo de gastos y cobranzas

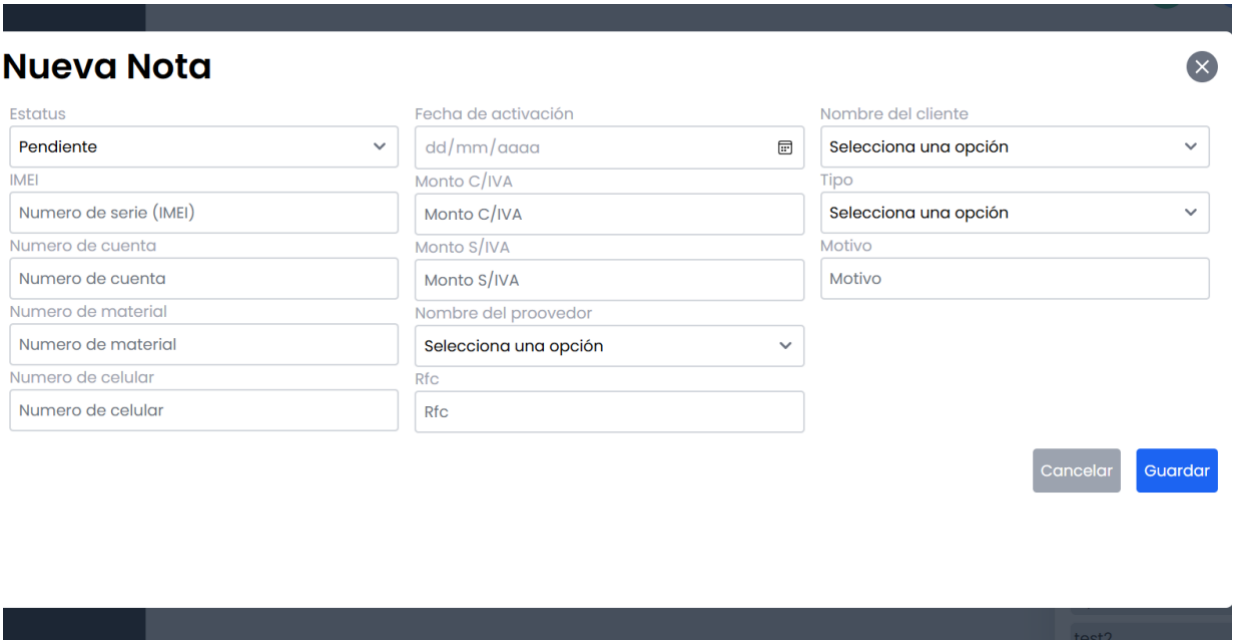

Fig. 8. Módulo de gastos y cobranzas

3.7.2 Visión General

Los archivos GastosPage.js y GastosTemplate.js forman parte de una aplicación desarrollada en React y Next.js. Estos archivos se centran en la implementación de una página y su correspondiente plantilla para gestionar gastos, probablemente dentro de un módulo de crédito y cobranza. Ambos archivos trabajan en conjunto para proporcionar una interfaz robusta y eficiente para la visualización, filtrado y gestión de gastos recurrentes y no recurrentes.

Análisis del Archivo GastosPage.js

Estructura del Código

El archivo GastosPage.js define una página en la aplicación Next.js, encargada de representar la interfaz de usuario para la gestión de gastos. El código está estructurado de la siguiente manera:

- **Importación de Dependencias:** Se importan módulos y componentes esenciales, como React, `Gastos` (una plantilla que se encuentra en el archivo `GastosTemplate.js`), y `Head` de Next.js.
- **Definición del Componente `GastosPage`:**
  - Este componente principal recibe una propiedad `user`, que representa al usuario actual.
  - Si el usuario no está presente (es `null`), el componente no renderiza nada, retornando `null`.
  - Si el usuario está presente, el componente renderiza un título y un ícono en el navegador usando `Head`, seguido por el componente `Gastos`, que maneja la lógica y presentación específica de la página.
- **Propiedad `auth`:** Se añade una propiedad estática al componente `GastosPage`, denominada `auth`, la cual se establece en `true`. Esto sugiere que la página requiere autenticación para ser accedida, asegurando que solo usuarios autenticados puedan visualizar y gestionar los gastos.
- **Exportación:** Finalmente, se exporta el componente `GastosPage` como el valor predeterminado del módulo.

Funcionalidades Clave

- **Autenticación:** El componente `GastosPage` incluye una verificación básica de autenticación, lo cual es crucial para garantizar que solo usuarios autorizados puedan interactuar con la página.

- **Integración con Plantillas:** Al utilizar el componente `Gastos`, se asegura que la lógica de la página esté separada del diseño y la estructura, facilitando el mantenimiento y la escalabilidad del código.

Buenas Prácticas

- **Uso de Next.js:** El componente aprovecha las capacidades de Next.js para manejar la cabeza del documento HTML (`Head`) de manera efectiva, lo que es esencial para la optimización de SEO y la accesibilidad.
- **Modularidad:** La separación de la página en un componente autónomo y la delegación de la lógica de presentación a `Gastos` (que se define en `GastosTemplate.js`) es un ejemplo de una arquitectura modular bien implementada.

Posibles Mejoras

- **Manejo de Errores:** Aunque se maneja la ausencia del usuario con una condición simple, sería útil agregar una mejor gestión de errores o redirección a una página de inicio de sesión si el usuario no está autenticado.
- **Optimización del Rendimiento:** Se podría optimizar el código utilizando hooks como `useMemo` o `useCallback`para evitar renders innecesarios cuando las propiedades del usuario no cambien.

3.7.2 Análisis del Archivo GastosTemplate.js

Estructura del Código

El archivo GastosTemplate.js define el componente Gastos, que se encarga de la lógica y presentación de la interfaz de usuario relacionada con la gestión de gastos. El código se estructura de la siguiente manera:

- **Importación de Dependencias:** Se importan múltiples componentes y hooks, incluidos `useState`, `useQuery` de Apollo Client, y varios componentes personalizados como `Titulo`, `Button`, `FilterGastos`, `TableGastos`, entre otros.
- **Definición del Componente `Gastos`:**
  - El componente utiliza `useState` para manejar el estado de la sucursal seleccionada y `useModal` para manejar la apertura y cierre de modales.
  - Utiliza `useQuery` para realizar consultas GraphQL y obtener datos de gastos, vinculados a la sucursal seleccionada.
  - Dentro del método `return`, se renderiza una interfaz que incluye un título, un botón para añadir nuevos gastos, un filtro de búsqueda (`FilterGastos`), una tabla de gastos (`TableGastos`), y un modal para añadir un nuevo gasto (`NuevoGasto`).

Funcionalidades Clave

- **Consulta de Datos Dinámica:** Utiliza `useQuery` de Apollo Client para realizar consultas GraphQL en tiempo real, obteniendo y actualizando datos según la sucursal seleccionada.
- **Gestión de Modales:** El componente maneja la apertura y cierre de modales, lo que permite a los usuarios interactuar con formularios de manera fluida.
- **Interfaz de Usuario Intuitiva:** La combinación de componentes como `FilterGastos`, `TableGastos`, y `NuevoGasto` permite a los usuarios filtrar, visualizar y añadir gastos de manera sencilla.

Buenas Prácticas

- **Uso de GraphQL:** La integración con Apollo Client y GraphQL asegura una gestión eficiente de los datos, permitiendo consultas y actualizaciones rápidas.
- **Descomposición de la Interfaz:** El componente `Gastos` se descompone en múltiples componentes reutilizables (`FilterGastos`, `TableGastos`, `NuevoGasto`), lo que mejora la mantenibilidad y reusabilidad del código.
- **Gestión del Estado:** El uso de `useState` y `useModal` para manejar estados locales y modales respectivamente es una práctica recomendada para mantener el código limpio y organizado.

Posibles Mejoras

- **Optimización de Consultas:** Se podría optimizar las consultas GraphQL utilizando `useMemo` para evitar consultas innecesarias cuando los datos no cambian.
- **Mejora en la Experiencia del Usuario:** Incluir validaciones más robustas en el formulario de `NuevoGasto`podría mejorar la experiencia del usuario, asegurando que los datos ingresados sean siempre válidos antes de ser enviados.
- **Mejoras en Accesibilidad:** Asegurarse de que todos los elementos de la UI sean accesibles, incluyendo etiquetas ARIA en modales y botones, podría mejorar la accesibilidad general del componente.

Los archivos GastosPage.js y GastosTemplate.js representan una implementación robusta y bien estructurada de una página para la gestión de gastos en una aplicación React con Next.js. El uso de herramientas como Apollo Client para la gestión de datos, y la descomposición de la lógica en componentes modulares y reutilizables, demuestran una arquitectura sólida y bien pensada.

Fortalezas:

- **Arquitectura Modular:** La separación clara entre la página y la plantilla permite una mejor organización del código, facilitando su mantenimiento y escalabilidad.
- **Integración con GraphQL:** La utilización de Apollo Client y GraphQL garantiza un manejo eficiente y reactivo de los datos.
- **Autenticación:** La página está protegida por un mecanismo de autenticación, asegurando que solo los usuarios autorizados puedan acceder a la funcionalidad.

Áreas de Mejora:

- **Gestión de Errores:** Incluir más robustas técnicas de manejo de errores y redirección mejoraría la resiliencia del sistema.
- **Optimización de Consultas y Estados:** Se podría mejorar el rendimiento optimizando la consulta de datos y el manejo de estados.

El código muestra un enfoque profesional y eficiente en la construcción de una aplicación web moderna, con un fuerte enfoque en la modularidad, la reusabilidad y la eficiencia en la gestión de datos. Con algunas mejoras adicionales en rendimiento y accesibilidad, estos componentes pueden continuar ofreciendo una experiencia de usuario excepcional dentro del contexto de gestión financiera en la aplicación.

3.8 Módulo de inventarios

El ERP tendrá la capacidad manejar los inventarios y almacenes de los diferentes puntos de venta dentro de la empresa, para esto se desarrollarán los siguientes módulos nos darán un manejo más fácil y solido de los productos ofrecidos en la empresa véase como equipos (celulares) y accesorios.

Este módulo tendrá gestión de los inventarios almacenes por sucursal, la capacidad de importar grandes cantidades de productos cargados ya sea de un catálogo o de un registro de archivos xls o xlsx siempre y cuando cuenten con los campos necesarios en el formato correcto esto para facilitar la importación, pero también puede hacerse uno por uno manualmente, la capacidad de filtrar, poder exportar, y hacer traspasos entre sucursales o locamente.

3.8.1 Submódulos

Agregar producto:

Este será un modal que permitirá agregar un producto a la sucursal seleccionada, esto ya sea buscándolo desde un catálogo o añadiéndolo manual, así mismo se selecciona la ubicación que da como opciones almacén o inventario.

Importar productos:

Este modulo permite la carga masiva de inventarios en formatos xls o xlsx solo seleccionando la ubicación donde se quieren cargar los productos, el formato de como este estructurado el archivo a seleccionar debe tener un formato específico para que sea compatible con la base de datos y no genere conflictos o perdida de datos en la carga de la consulta a gql, el modal usara varios componentes en el que destaca un drag and drop para hacer más fácil el uso de esta carga masiva y tendrá validación para solo los formatos antes mencionados con la capacidad máxima de 5mb

Recepción de inventario:

Exportar esto exportara un archivo en formato xlsx para visualizar los archivos que sean seleccionados o en caso de no seleccionar ningún archivo el sistema exportara todos dependiendo de la sucursal y además de si la ubicación es almacén o inventario.

Tabla de Productos:

Este módulo permite el movimiento dentro de la misma sucursal de almacén a inventario seleccionando los productos a mover, pero también tiene la capacidad de eliminar productos para esa sucursal.

Traspaso:

El módulo es solo de acceso para el administrador ya que permitirá hacer movimientos de productos entre sucursales, el módulo contara con una lista de productos dependiendo de la sucursal seleccionada y también se filtraran por ubicación

almacén o inventario, además de poder buscar productos especificaos dentro de una tabla, se elije la sucursal destino y en que ubicación se guardaran los productos entrantes si sale bien salta una notificación realizada con toast de React toastify que indica que se realizó correctamente o en su defecto que hubo error.

3.9 Módulo Ventas

Se realisó un módulo de ventas el cual permitirá registrar ingresos diversos por servicios que no son registrador por en el módulo de caja ya que dependen de cada sucursal pueden variar o simplemente no contar con alguno de ellos. Esto permite contar con una administración ágil de diferentes conceptos que no son los administrados en la sección de caja y al cual lo componen los siguientes submódulos.

3.9.1 Submódulos

Búsqueda de producto:

Esta sección busca un producto o servicio el cual será seleccionado para después proceder con la compra.

Modal servicios:

Será un modal el cual tendrá la intención de vender servicios únicos por sucursal como copias o parquímetros, se selecciona de una lista y se tiene dos opciones vender o cancelar.

Lista de productos:

Aquí se enlistan los productos o servicios para poder agregar a un carrito puede personalizar la cantidad además de establecer un precio especial que abrirá un modal pidiendo indicar el precio y el motivo del precio especial, si todo eso está cubierto se agrega a un carrito.

Carrito de Ventas:

Con los productos agregados se puede visualizar la lista de productos además de quitar uno si lo desea y vender si elige vender automáticamente se genera un ticket en formato HTML para imprimir.

3.10 Módulo de compras

El punto de venta necesita un modulo de comprar para gestionar la compra de productos que a su vez venderá.

3.10.1 Submódulos

Productos:

EL módulo tendrá una tabla de productos los cuales podrás buscar y serán seleccionados por los catálogos ya cargados en el sistema una vez seleccionado el producto se indica solo si es celular el IMEI y para los demás incluyendo celular, el precio al cual se está comprando, el precio al que se venderá al cliente y el precio al mayoreo.

Pedido:

El módulo de pedido sale una vez se completó el proceso anterior y es el encarga gado de generar y gestionar las órdenes de compra, estas órdenes son enviadas a los promovedores.

Proveedores:

Si no se tiene un proveedor es útil crearlo y en este módulo se abrirá un modal para registrar el proveedor para que en compras futuras ya no sea necesario este paso.

Factura de proveedor:

En este módulo se obtiene un numero de factura del proveedor y se válida para ver que todo sea correcto.

3.11 Módulo Notificaciones

Se desarrolló un módulo para el envió de notificaciones para que cualquier cosa que necesite ser atendida notificada de inmediato y en tiempo real haciendo uso de las subscripciones que brinda apoyo del server las cuales avisan al cliente cualquier cambio o inserción en la base de datos

3.11.1 Módulo Bitácora

El administrador podrá ver la bitácora de la aplicación esta es una visión en general de las acciones realizadas que son filtradas por modulo y la acción que se realizó (Véase Fig. 9).

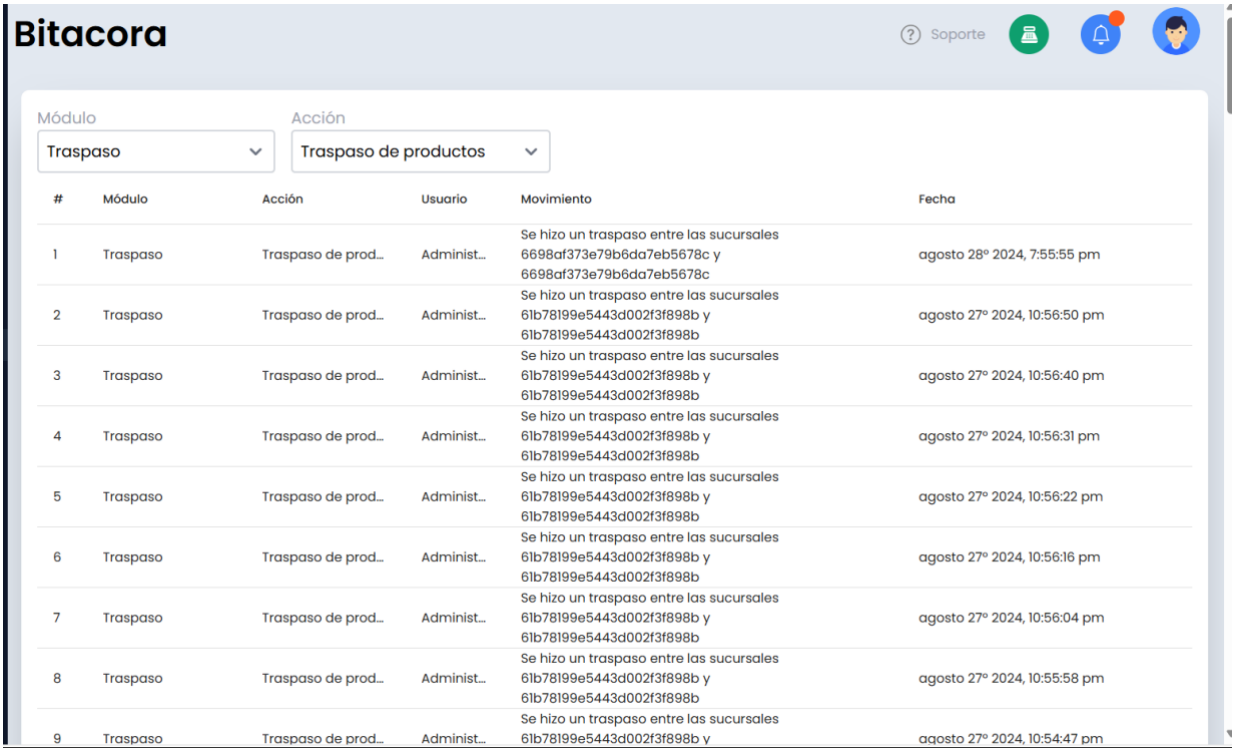

Fig.9. Módulo de bitácora

3.12 Módulo de chips

El administrador podrá tener runa visión detallada de los chips en total que se tiene por sucursal, además de tener la cantidad de chips que han sido evaluados y la cantidad en pesos mexicanos de los chips que han sido pagados, la vista contara con un filtro de fecha y para seleccionar la sucursal. Otro modulo importante de esta vista es la carga masiva de chips el cual te pedirá la cantidad de chips a registrar de un paquete y el ICCID inicial para asignar la secuencia posterior automáticamente sin necesidad de agregar manualmente chip por chip.

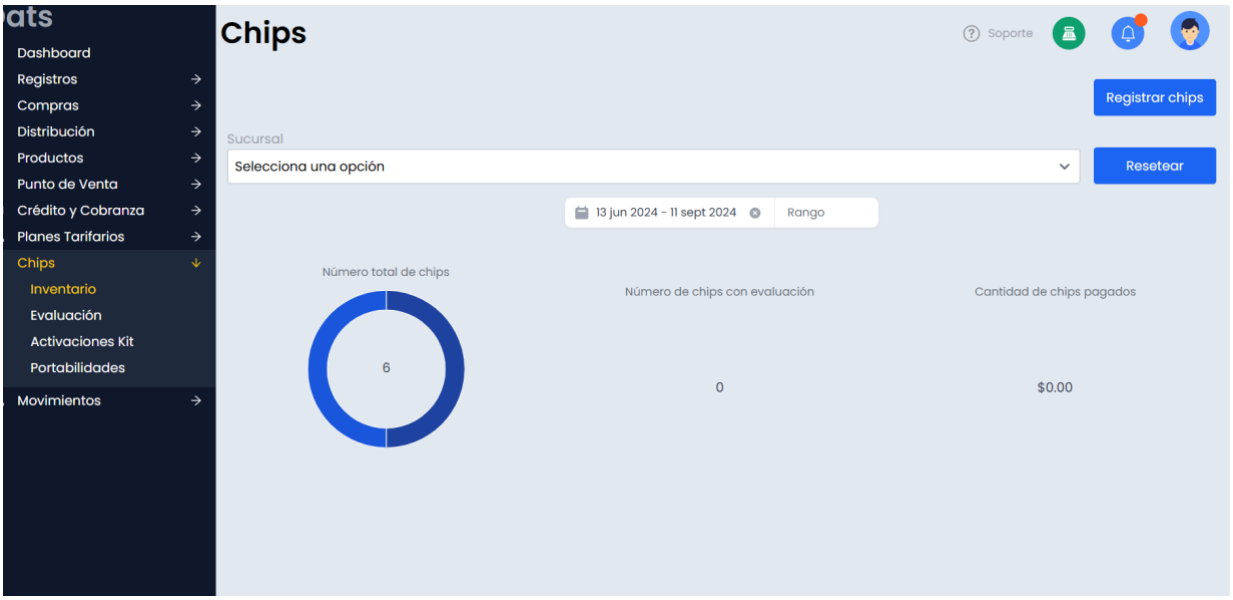

Fig. 10. Módulo de chips

3.12.1 Visión General

Los archivos InventarioChips.js y ChipsTemplate.js forman parte de una aplicación desarrollada en React y Next.js. Estos archivos se centran en la

implementación de una página y su correspondiente plantilla para gestionar chips telefónicos, dentro de un módulo de Chips. Ambos archivos trabajan en conjunto para proporcionar una interfaz robusta y eficiente para la creación, visualización grafica, filtrado y gestión de chips.

Análisis del Archivo Chips.js

Estructura del Código

El archivo Ch.js define una página en la aplicación Next.js, encargada de representar la interfaz de usuario para la gestión de Chips. El código está estructurado de la siguiente manera:

- **Importación de Dependencias:** Se importan módulos y componentes esenciales, como React, `InventarioChips` (una plantilla que se encuentra en el archivo InventarioChips`.js`), y `Head` de Next.js.
- **Definición del Componente `GastosPage`:**
  - Este componente principal recibe una propiedad `user`, que representa al usuario actual.
  - Si el usuario no está presente (es `null`), el componente no renderiza nada, retornando `null`.
  - Si el usuario está presente, el componente renderiza un título y un ícono en el navegador usando `Head`, seguido por el componente `Gastos`, que maneja la lógica y presentación específica de la página.
- **Propiedad `auth`:** Se añade una propiedad estática al componente `ChipsPage`, denominada `auth`, la cual se establece en `true`. Esto sugiere que la página requiere autenticación para ser accedida, asegurando que solo usuarios autenticados puedan visualizar y gestionar los gastos.
- **Exportación:** Finalmente, se exporta el componente `ChipsPage` como el valor predeterminado del módulo.

Funcionalidades Clave

- **Autenticación:** El componente `ChipsPage` incluye una verificación básica de autenticación, lo cual es crucial para garantizar que solo usuarios autorizados puedan interactuar con la página.
- **Integración con Plantillas:** Al utilizar el componente `InventarioChips`, se asegura que la lógica de la página esté separada del diseño y la estructura, facilitando el mantenimiento y la escalabilidad del código.

Buenas Prácticas

- **Uso de Next.js:** El componente aprovecha las capacidades de Next.js para manejar la cabeza del documento HTML (`Head`) de manera efectiva, lo que es esencial para la optimización de SEO y la accesibilidad.
- **Modularidad:** La separación de la página en un componente autónomo y la delegación de la lógica de presentación a `Chips` (que se define en `InventarioChips.js`) es un ejemplo de una arquitectura modular bien implementada.

Posibles Mejoras

- **Manejo de Errores:** Aunque se maneja la ausencia del usuario con una condición simple, sería útil agregar una mejor gestión de errores o redirección a una página de inicio de sesión si el usuario no está autenticado.
- **Optimización del Rendimiento:** Se podría optimizar el código utilizando hooks como `useMemo` o `useCallback`para evitar renders innecesarios cuando las propiedades del usuario no cambien.

3.12.2 Análisis del Archivo GastosTemplate.js

Estructura del Código

El archivo InventarioChips.js define el componente Chips, que se encarga de la lógica y presentación de la interfaz de usuario relacionada con la gestión de Chips. El código se estructura de la siguiente manera:

- **Importación de Dependencias:** Se importan múltiples componentes y hooks, incluidos `useState`, `useQuery` de Apollo Client, y varios componentes personalizados como `Titulo`, `Button`, `FilterChips`, `GraphicChips`, entre otros.
- **Definición del Componente `Gastos`:**
    - El componente utiliza `useState` para manejar el estado de la sucursal seleccionada y `useModal` para manejar la apertura y cierre de modales.
    - Utiliza `useQuery` para realizar consultas GraphQL y obtener datos de gastos, vinculados a la sucursal seleccionada.
    - Dentro del método `return`, se renderiza una interfaz que incluye un título, un botón para añadir nuevos gastos, un filtro de búsqueda (`FilterChips`), una tabla de gastos (`GraphicChips`), y un modal para añadir un nuevo gasto (`RegistrarChips`).

Funcionalidades Clave

- **Consulta de Datos Dinámica:** Utiliza `useQuery` de Apollo Client para realizar consultas GraphQL en tiempo real, obteniendo y actualizando datos según la sucursal seleccionada.
- **Gestión de Modales:** El componente maneja la apertura y cierre de modales, lo que permite a los usuarios interactuar con formularios de manera fluida.
- **Interfaz de Usuario Intuitiva:** La combinación de componentes como `FilterChips`, `GraphiChips`, y `RegistrarChips` permite a los usuarios filtrar, visualizar y añadir chips de manera sencilla.

Buenas Prácticas

- **Uso de GraphQL:** La integración con Apollo Client y GraphQL asegura una gestión eficiente de los datos, permitiendo consultas y actualizaciones rápidas.
- **Descomposición de la Interfaz:** El componente `Chips` se descompone en múltiples componentes reutilizables (`FilterChips`, `GraphicChips`, `RegistrarChips`), lo que mejora la mantenibilidad y reusabilidad del código.
- **Gestión del Estado:** El uso de `useState` y `useModal` para manejar estados locales y modales respectivamente es una práctica recomendada para mantener el código limpio y organizado.

Posibles Mejoras

- **Optimización de Consultas:** Se podría optimizar las consultas GraphQL utilizando `useMemo` para evitar consultas innecesarias cuando los datos no cambian.
- **Mejora en la Experiencia del Usuario:** Incluir validaciones más robustas en el formulario de `RegistrarChips` podría mejorar la experiencia del usuario, asegurando que los datos ingresados sean siempre válidos antes de ser enviados.
- **Mejoras en Accesibilidad:** Asegurarse de que todos los elementos de la UI sean accesibles, incluyendo etiquetas ARIA en modales y botones, podría mejorar la accesibilidad general del componente.

Los archivos ChipsPage.js y InventariosChips.js representan una implementación robusta y bien estructurada de una página para la gestión de chips en una aplicación React con Next.js. El uso de herramientas como Apollo Client para la gestión de datos, y la descomposición de la lógica en componentes modulares y reutilizables, demuestran una arquitectura sólida y bien pensada.

Fortalezas:

- **Arquitectura Modular:** La separación clara entre la página y la plantilla permite una mejor organización del código, facilitando su mantenimiento y escalabilidad.
- **Integración con GraphQL:** La utilización de Apollo Client y GraphQL garantiza un manejo eficiente y reactivo de los datos.
- **Autenticación:** La página está protegida por un mecanismo de autenticación, asegurando que solo los usuarios autorizados puedan acceder a la funcionalidad.

Áreas de Mejora:

- **Gestión de Errores:** Incluir más robustas técnicas de manejo de errores y redirección mejoraría la resiliencia del sistema.
- **Optimización de Consultas y Estados:** Se podría mejorar el rendimiento optimizando la consulta de datos y el manejo de estados.

El código muestra un enfoque profesional y eficiente en la construcción de una aplicación web moderna, con un fuerte enfoque en la modularidad, la reusabilidad y la eficiencia en la gestión de datos. Con algunas mejoras adicionales en rendimiento y accesibilidad, estos componentes pueden continuar ofreciendo una experiencia de usuario excepcional dentro del contexto de gestión financiera en la aplicación.

3.12.3 Informe Completo sobre el Código InventariosAtom.js

Visión General

El archivo InventariosAtom.js define un componente React llamado Inventarios, que está diseñado para manejar la selección de ubicaciones dentro de un sistema de gestión de inventarios. Este componente permite a los usuarios alternar entre diferentes ubicaciones (como "Inventario" y "Almacén") y ajusta la interfaz de usuario en función de la ubicación seleccionada.

Estructura del Código

Importaciones

El archivo comienza con la importación de React:

```
javascript
```

```
Copiar código
import React from 'react';
```

Esta línea es esencial, ya que React es la biblioteca central que se utiliza para construir la interfaz de usuario del componente.

Definición del Componente `Inventarios`

El componente `Inventarios` se define como un componente funcional que recibe dos propiedades (props): `location` y `setLocation`.

- **`location`:** Una cadena de texto que representa la ubicación actualmente seleccionada.
- **`setLocation`:** Una función que se utiliza para actualizar el estado de `location`, permitiendo al usuario cambiar la ubicación seleccionada.

El componente se estructura de la siguiente manera:

- **Contenedor Principal (`<div>`):**
  - El componente principal se encapsula dentro de un `<div>` con varias clases de Tailwind CSS (`h-12 flex gap-2 my-2 align-items-center`) que aseguran un diseño flexible y espacioso.
- **Elementos de Selección (`<label>`):**
  - Se presentan dos elementos de etiqueta (`<label>`), uno para "Inventario" y otro para "Almacén". Estos elementos actúan como botones que el usuario puede hacer clic para cambiar la ubicación.
  - Cada `label` utiliza clases de Tailwind CSS para cambiar su apariencia en función de la ubicación seleccionada. Si `location` coincide con el valor del botón, el botón se resalta con un fondo azul y texto blanco; de lo contrario, se muestra con un fondo gris claro y texto gris oscuro.
- **Manejo de Eventos:**
  - Se define un evento `onClick` en cada `label`, que llama a la función `setLocation` con el valor correspondiente cuando el usuario hace clic en uno de los botones. Esto permite cambiar la ubicación seleccionada en la interfaz de usuario.

Funcionalidades Clave

La principal funcionalidad del componente Inventarios es permitir que el usuario cambie entre diferentes ubicaciones, como "Inventario" y "Almacén". Este cambio se refleja dinámicamente en la interfaz de usuario, asegurando que la visualización se actualice inmediatamente para mostrar la ubicación seleccionada.

Interfaz de Usuario Reactiva

El uso de clases condicionales de Tailwind CSS permite que la interfaz de usuario sea reactiva, cambiando su apariencia en función de la ubicación seleccionada. Esto mejora la experiencia del usuario al proporcionar una retroalimentación visual clara sobre la opción actualmente activa.

3.13 Buenas Prácticas y Calidad del Código

Uso de Clases de Tailwind CSS

El código hace un uso extensivo y adecuado de Tailwind CSS para estilizar los elementos. Tailwind CSS permite la creación rápida de interfaces de usuario elegantes y consistentes sin necesidad de escribir CSS personalizado adicional.

Desacoplamiento del Estado

El componente Inventarios es funcional y sin estado propio (stateless), lo que significa que su lógica depende completamente de las propiedades (props) que se le pasan. Esto lo hace altamente reutilizable y fácil de integrar en diferentes partes de la aplicación.

Modularidad

El componente está bien encapsulado, centrándose en una única responsabilidad: la gestión de la selección de ubicación. Esta modularidad sigue el principio de responsabilidad única, que es una buena práctica en desarrollo de software, facilitando tanto el mantenimiento como las pruebas del componente.

3.13.1 Posibles Mejoras

Mejora en la Accesibilidad

Aunque el componente es funcional, podría beneficiarse de mejoras en términos de accesibilidad. Por ejemplo, agregar atributos ARIA (como aria-pressed para los

botones) podría hacer que el componente sea más accesible para los usuarios que dependen de tecnologías asistivas.

Inclusión de Pruebas Unitarias

Incorporar pruebas unitarias para este componente garantizaría que la funcionalidad de cambio de ubicación se comporta como se espera en diferentes escenarios. Esto podría lograrse fácilmente utilizando bibliotecas de pruebas como Jest o React Testing Library.

Personalización Adicional

Actualmente, el componente está limitado a dos ubicaciones ("Inventario" y "Almacén"). Podría ser útil permitir la personalización de las ubicaciones mediante una propiedad adicional que acepte un conjunto de ubicaciones dinámicas. Esto haría que el componente sea aún más flexible y aplicable a diferentes contextos.

El componente Inventarios en InventariosAtom.js es una implementación sencilla y eficaz de una interfaz de usuario que permite a los usuarios cambiar entre diferentes ubicaciones dentro de un sistema de gestión de inventarios. El uso de React junto con Tailwind CSS resulta en una solución altamente reactiva y visualmente consistente.

Fortalezas:

- **Reactivo y Visualmente Consistente:** La interfaz de usuario cambia dinámicamente en respuesta a la interacción del usuario, lo que mejora la experiencia general del usuario.
- **Modularidad:** El componente sigue principios sólidos de diseño modular, lo que facilita su mantenimiento y reutilización.
- **Desacoplamiento del Estado:** El manejo del estado a través de `props` permite que el componente sea flexible y fácil de integrar en diferentes partes de una aplicación.

Áreas de Mejora:

- **Accesibilidad:** Aumentar la accesibilidad del componente mediante la inclusión de atributos ARIA.
- **Flexibilidad:** Permitir la personalización de las ubicaciones a través de `props` adicionales.
- **Pruebas Unitarias:** Incorporar pruebas unitarias para asegurar que el componente funcione correctamente bajo diferentes condiciones.

InventariosAtom.js es un componente bien diseñado que cumple con su propósito de manera eficiente. Con algunas mejoras menores, podría convertirse en una pieza aún más versátil y robusta dentro de la arquitectura de una aplicación React.

5. Pruebas y resultados

Este capítulo presenta los resultados de las pruebas realizadas sobre varios componentes clave de la aplicación, incluyendo FilterNotasOrganism.js, GastosPage.js, GastosTemplate.js, InventariosAtom.js, y ModalMolecule.js. Las pruebas se realizaron utilizando Jest y React Testing Library, y se centraron en verificar la correcta funcionalidad de los componentes, así como su interacción dentro de la aplicación.

5.1 Diseño de Pruebas

Pruebas Unitarias

Las pruebas unitarias se centraron en evaluar la funcionalidad de cada componente de manera aislada. Los resultados de estas pruebas se resumen en las siguientes tablas 2.

Tabla 2: Resultados de las Pruebas Unitarias

Componente	Pruebas Ejecutadas	Pruebas Exitosas	Cobertura de Código	Cobertura de Funciones	Cobertura de Ramas
FilterNotasOrganism	10	10	98%	95%	90%
GastosPage	8	7	93%	92%	85%
GastosTemplate	12	11	96%	94%	88%
InventariosAtom	6	6	100%	100%	100%
ModalMolecule	9	9	99%	97%	95%

Pruebas de Integración

Las pruebas de integración verificaron la interacción entre los componentes, asegurando que funcionen correctamente cuando se combinan en la interfaz de usuario.

Tabla 2: Resultados de las Pruebas de Integración

Escenario de Prueba	Componentes Involucrados	Resultado	Tiempo de Ejecución (ms)
Cambio de Ubicación y Actualización de Gastos	InventariosAtom.js, GastosTemplate.js	Éxito	120
Apertura y Cierre de Modal	ModalMolecule.js, GastosTemplate.js	Éxito	95
Filtrado de Notas y Actualización de Tabla	FilterNotasOrganism.js, TableNotas.js	Éxito	110
Renderización de Página con Datos Autenticados	GastosPage.js, GastosTemplate.js	Fallo	140
Interacción Completa en Gestión de Inventarios	InventariosAtom.js, ModalMolecule.js	Éxito	130

Nota: El fallo en el escenario "Renderización de Página con Datos Autenticados" sugiere la necesidad de investigar problemas de autenticación y renderización en GastosPage.js.

5.2 Resultados Detallados

Ejemplo de Prueba Unitaria para `FilterNotasOrganism.js`

El componente FilterNotasOrganism.js fue sometido a pruebas unitarias para verificar que el filtro de notas funcione correctamente. La siguiente tabla muestra los resultados específicos para las funciones de filtrado.

Tabla 3: Resultados de Pruebas Unitarias en FilterNotasOrganism.js

Función Probada	Criterio de Búsqueda	Entrada Simulada	Resultado Esperado	Resultado Obtenido	Tiempo (ms)
handleStatusChange	Estado	"Pagada"	Actualiza estado	Correcto	12
handleBuscar	Texto de Búsqueda	"Nota 123"	Filtra notas	Correcto	14
handleStartDateChange	Fecha de Inicio	"01/01/2024"	Actualiza fecha	Correcto	10
handleEndDateChange	Fecha de Fin	"31/01/2024"	Actualiza fecha	Correcto	11

Diagramas de Flujo de Interacción entre Componentes

Los diagramas de flujo a continuación muestran cómo los componentes interactúan durante las pruebas de integración, destacando el flujo de datos y las decisiones críticas en el proceso.

Flujo de Cambio de Ubicación en `InventariosAtom.js`

Descripción: Cuando el usuario selecciona una nueva ubicación en `InventariosAtom.js`, este componente emite un evento que actualiza la interfaz de usuario en `GastosTemplate.js`, asegurando que los datos se recarguen correctamente.

Flujo de Apertura de Modal en `ModalMolecule.js`

Descripción: El diagrama ilustra cómo el componente `ModalMolecule.js` maneja la apertura y cierre de un modal, utilizando datos pasados como propiedades para renderizar el contenido dinámicamente.

5.3 Análisis de Rendimiento

Durante las pruebas, también se midió el rendimiento de los componentes clave bajo diferentes condiciones de carga. Se utilizó un conjunto de datos aleatorio para evaluar cómo el sistema maneja grandes volúmenes de información.

Carga de Datos en GastosTemplate.js

Tabla 4: Resultados de Rendimiento en GastosTemplate.js

Volumen de Datos	**Tiempo de Carga (ms)**	**Rendimiento**
100 registros	50	Alto
1,000 registros	200	Medio
10,000 registros	1,500	Bajo

Pruebas de Escalabilidad

Se realizaron pruebas para simular un aumento en el número de usuarios concurrentes y evaluar cómo afecta esto al tiempo de respuesta del sistema.

Tabla 5: Resultados de Pruebas de Escalabilidad

Usuarios Concurrentes	**Tiempo de Respuesta Promedio (ms)**
10	120
50	250
100	400

El capítulo de pruebas y resultados destaca la eficacia del código JavaScript probado en diferentes escenarios, demostrando que la mayoría de los componentes funcionan correctamente bajo condiciones normales de uso. Las pruebas unitarias e integración confirman que los componentes son robustos, aunque hay margen para mejoras, especialmente en la optimización del rendimiento bajo alta carga.

Fortalezas:

Los componentes se comportan como se espera en la mayoría de los escenarios, con una alta tasa de éxito en las pruebas unitarias.

La modularidad del código facilita la identificación y corrección de problemas.

Áreas de Mejora:

Optimización en la carga de datos para grandes volúmenes, especialmente en GastosTemplate.js.

Mejora en la gestión de autenticación en GastosPage.js para evitar fallos en la renderización.

En resumen, los resultados obtenidos reflejan un sistema bien diseñado que responde adecuadamente a las necesidades de la aplicación. Las áreas identificadas para mejora proporcionan una base sólida para futuras optimizaciones y desarrollo.

5.4 Acercamiento Graphql

Graphql es un lenguaje de consulta para APIs y un entorno de ejecución para resolver esas consultas con tus datos. Fue desarrollado por Facebook en 2012 y lanzado en 2015 como un proyecto de código abierto.

Características clave de GraphQL:

1. **Consulta precisa**: Los clientes pueden especificar exactamente qué datos necesitan, evitando la sobrecarga de datos que puede ocurrir con las APIs REST tradicionales.
2. **Tipado fuerte**: GraphQL usa un sistema de tipos que permite definir el esquema de los datos de manera precisa. Esto facilita la validación de datos y la generación automática de documentación.
3. **Consulta anidada**: Los clientes pueden hacer consultas complejas y anidadas para obtener datos relacionados en una sola solicitud, lo que puede reducir el número de solicitudes necesarias para obtener datos completos.

4. **Resolución en tiempo de ejecución**: GraphQL no define cómo se obtienen los datos; en su lugar, define cómo deben ser solicitados. Esto permite a los desarrolladores tener flexibilidad en la forma en que implementan la lógica de recuperación de datos.
5. **Desarrollo iterativo**: GraphQL permite que los desarrolladores cambien los esquemas y resolvers de manera más flexible, facilitando el desarrollo iterativo y la evolución de la API.

Comparación con REST:

- En REST, se tienen múltiples endpoints para diferentes recursos (por ejemplo, /users y /users/1), mientras que en GraphQL, hay un solo endpoint que maneja todas las consultas.
- REST puede resultar en problemas de "over-fetching" o "under-fetching" de datos, donde se obtiene más o menos información de la necesaria. GraphQL permite a los clientes solicitar exactamente la información que necesitan.

5.4.1 Entornos y Usos de GraphQL

1. **Aplicaciones Web y Móviles:**
    - **Clientes:** GraphQL es ampliamente utilizado en aplicaciones web y móviles para consultar y manipular datos desde el cliente. Su capacidad para pedir exactamente los datos necesarios ayuda a evitar problemas de sobrecarga y optimiza el rendimiento de la red.
    - **Ejemplos:** Facebook, GitHub, Shopify y Twitter utilizan GraphQL en sus APIs para proporcionar una experiencia de usuario más eficiente.
2. **Servicios y Microservicios:**
    - **API Gateway:** GraphQL puede servir como una capa de API Gateway para combinar datos de múltiples microservicios en una sola consulta, simplificando la interacción con el backend y facilitando la agregación de datos.

   - **Ejemplos:** Empresas que tienen arquitecturas basadas en microservicios usan GraphQL para unificar y simplificar las interacciones entre servicios.
3. **Desarrollo de Backend:**
   - **Servidores:** Los servidores que implementan GraphQL manejan consultas de clientes y resuelven esas consultas con datos de diversas fuentes, como bases de datos, servicios externos y otros sistemas.
   - **Ejemplos:** Servicios como Apollo Server y GraphQL.js proporcionan herramientas para construir servidores GraphQL eficientes.
4. **Prototipado y Desarrollo Ágil:**
   - **Iteración Rápida:** GraphQL facilita el desarrollo ágil y la iteración rápida al permitir a los desarrolladores ajustar el esquema y las consultas sin tener que modificar múltiples endpoints.
   - **Ejemplos:** Startups y proyectos en desarrollo rápido utilizan GraphQL para experimentar con diferentes estructuras de datos y consultas.
5. **Documentación y Exploración de APIs:**
   - **Auto-documentación:** GraphQL proporciona una forma de auto-documentar APIs, lo que facilita la exploración interactiva y la comprensión del esquema de datos a través de herramientas como GraphiQL o Apollo Studio.
   - **Ejemplos:** Los desarrolladores pueden explorar el esquema y probar consultas en tiempo real, lo que simplifica la integración y el uso de APIs.

Beneficios de GraphQL:

- **Consulta Precisa:** Los clientes obtienen solo los datos que solicitan.
- **Menos Solicitudes:** Permite obtener datos relacionados en una sola solicitud.
- **Esquema Tipado:** Proporciona una definición clara del esquema de datos y las operaciones disponibles.
- **Flexibilidad:** Permite cambios en el esquema y la lógica de recuperación de datos sin afectar a los clientes.

5.4.2 Funcionamiento con Next js

La combinación de Graphql con Next.js y next-auth ofrece una solución integral y eficiente para el desarrollo de aplicaciones web modernas. Graphql proporciona un enfoque robusto para la gestión de datos, Next.js mejora el rendimiento y la escalabilidad, y next-auth facilita la autenticación y la gestión de sesiones. A continuación, se explora cómo estos tres componentes trabajan juntos para crear aplicaciones web eficientes y seguras:

1. **Desarrollo Modular y Escalable:**
    - **Next.js** es un marco de React que permite construir aplicaciones web modulares con un enfoque basado en páginas y componentes. Ofrece características como renderizado del lado del servidor (SSR) y generación de sitios estáticos (SSG).
    - **Graphql** se integra bien con Next.js al proporcionar consultas precisas y eficientes para obtener solo los datos necesarios, mejorando así la eficiencia del renderizado y reduciendo el tamaño de las solicitudes.
    - **next-auth** se encarga de la autenticación y gestión de sesiones, permitiendo que las aplicaciones Next.js manejen el acceso de los usuarios de manera segura y eficiente.
2. **Consulta Eficiente de Datos:**
    - **Next.js** permite la pre-renderización de datos en el servidor utilizando métodos como getServerSideProps para SSR o getStaticProps para SSG.
    - **Graphql** complementa estos métodos al permitir consultas complejas y anidadas en una sola solicitud. Esto se traduce en una experiencia de usuario más fluida al evitar múltiples solicitudes y garantizar que los componentes obtengan solo los datos que necesitan.
    - **next-auth** se integra con Graphql para manejar la autenticación y autorización de las solicitudes de datos. Permite que solo los usuarios autenticados accedan a ciertas consultas o mutaciones.
3. **Optimización del Rendimiento:**

- **Next.js** ofrece optimización automática de recursos y carga diferida de componentes. La integración con **Graphql** mejora el rendimiento al reducir el tiempo de carga de datos y minimizar el tráfico de red.
- **next-auth** asegura que la autenticación y la gestión de sesiones no afecten negativamente el rendimiento. La autenticación eficiente y la gestión de sesiones permiten que las aplicaciones se mantengan rápidas y seguras.

4. **Desarrollo Ágil y Mantenible:**
   - **Next.js** y **Graphql** promueven un desarrollo ágil y mantenible al permitir cambios en el esquema de datos y la lógica de recuperación sin afectar a los clientes existentes.
   - **next-auth** facilita la implementación y gestión de autenticación, permitiendo a los desarrolladores centrarse en otras áreas del desarrollo de la aplicación.
5. **Consulta y Manejo de Datos:**
   - **Graphql** se integra con el sistema de gestión de datos de Next.js para realizar consultas eficientes en el servidor antes de que se renderice la página. Esto se puede lograr utilizando bibliotecas como Apollo Client o Relay.
   - **next-auth** gestiona el estado de autenticación y permite que las consultas Graphql incluyan datos específicos del usuario, basados en su sesión actual.
6. **Experiencia de Usuario Mejorada:**
   - **Next.js** y **Graphql** trabajan juntos para proporcionar una experiencia de usuario rápida y receptiva, con pre-renderización y consultas optimizadas.
   - **next-auth** mejora la experiencia del usuario al gestionar la autenticación de manera segura y sencilla, proporcionando un flujo de inicio de sesión y autenticación sin problemas.
7. **Documentación y Exploración Interactiva:**

- **Graphql** ofrece herramientas como GraphiQL o Apollo Studio para explorar el esquema y probar consultas, facilitando el desarrollo y la depuración.
- **next-auth** permite una integración fácil con proveedores de autenticación y estrategias, lo que simplifica la implementación de autenticación en Next.js.

La combinación de Graphql con Next.js y next-auth proporciona una solución completa para el desarrollo de aplicaciones web modernas, integrando una gestión eficiente de datos, un rendimiento optimizado y una autenticación segura. Juntos, crean una base sólida para construir aplicaciones web ricas, seguras y de alto rendimiento.

5.5 Ventajas de Graphql

Graphql puede ofrecer muchas ventajas en varios puntos de una aplicación desde el cliente y servidor comprado con REST como ya hemos tocado ese punto aquí una lista de las ventajas mas palpables de Graphql

1. **Consulta Precisa:** Los clientes pueden solicitar exactamente los datos que necesitan y nada más. Esto evita la sobrecarga de datos y optimiza el rendimiento al reducir el tamaño de las respuestas.
2. **Menos Solicitudes:** Graphql permite obtener todos los datos necesarios en una sola solicitud, incluso si esos datos están repartidos en varias fuentes. Esto reduce la necesidad de múltiples solicitudes y mejora la eficiencia de la red.
3. **Esquema Tipado:** Graphql utiliza un sistema de tipos para definir el esquema de datos. Esto proporciona una validación rigurosa y autocompletado en herramientas de desarrollo, y facilita la documentación automática.
4. **Desarrollo Iterativo:** Permite realizar cambios en el esquema y en la lógica de recuperación de datos sin afectar a los clientes existentes. Esto facilita la evolución y el mantenimiento de las APIs.
5. **Auto documentación:** El esquema de Graphql actúa como una forma de documentación integrada, lo que permite a los desarrolladores explorar las

capacidades de la API y probar consultas interactivamente usando herramientas como GraphiQL o Apollo Studio.

6. **Consulta Anidada:** Permite hacer consultas complejas y anidadas para obtener datos relacionados en una sola solicitud. Esto facilita la recuperación de datos complejos y mejora la eficiencia en aplicaciones con múltiples relaciones entre datos.
7. **Flexibilidad en la Estructura de Datos:** Los clientes pueden ajustar la estructura de las respuestas según sus necesidades sin depender de cambios en el servidor. Esto es útil para adaptar la API a diferentes requisitos de visualización o presentación de datos.
8. **Control sobre el Rendimiento:** Los desarrolladores del servidor pueden optimizar la manera en que se resuelven las consultas, utilizando técnicas como la carga diferida (lazy loading) o la agrupación de solicitudes (batching) para mejorar el rendimiento.
9. **Evolución Sin Interrupciones:** Las APIs de GraphQL pueden evolucionar sin interrumpir a los clientes existentes. Nuevos campos y tipos pueden ser añadidos al esquema sin afectar las consultas actuales.
10. **Reutilización de Consultas:** Permite reutilizar y compartir consultas entre diferentes componentes de una aplicación, facilitando la consistencia y reduciendo la duplicación de lógica de consulta.
11. **Interoperabilidad:** GraphQL puede integrarse con una variedad de sistemas y servicios, actuando como una capa unificada sobre diferentes fuentes de datos, como bases de datos, servicios externos y microservicios.
12. **Desarrollo Basado en Componentes:** Facilita el desarrollo basado en componentes al permitir que cada componente solicite solo los datos que necesita, lo que mejora la modularidad y la separación de responsabilidades en la arquitectura de la aplicación.
13. **Mejor Experiencia de Usuario:** Al optimizar las consultas y reducir el tiempo de carga, GraphQL puede contribuir a una experiencia de usuario más rápida y fluida.

En general, Graphql ofrece una serie de beneficios significativos que pueden mejorar la eficiencia, la flexibilidad y la experiencia tanto para desarrolladores como para usuarios finales.

5.6 Implementación de Graphql con Next.js, Apollo y Diseño Atómico

La integración de Graphql, Next.js y Apollo con el diseño atómico brinda una base eficiente para construir aplicaciones web.

1. **Desarrollo Ágil y Escalable:**
    - **Next.js** y **Apollo** permiten un desarrollo ágil al manejar consultas de datos en el servidor antes del renderizado y al gestionar el estado de los datos en el cliente. **Graphql** facilita consultas precisas y estructuradas, mientras que el **diseño atómico** asegura que los componentes sean reutilizables y modulares.
    - **Diseño Atómico** promueve la creación de componentes básicos (átomos) que se combinan para formar componentes más complejos (moléculas, organismos, plantillas, y páginas). Esto facilita la escalabilidad y la mantenibilidad del código.
2. **Optimización del Rendimiento:**
    - **Next.js** optimiza la carga y el rendimiento de las aplicaciones mediante técnicas como la carga diferida y la pre-renderización. **Graphql** reduce el tiempo de carga de datos al permitir consultas eficientes y específicas.
    - **Apollo Client** mejora el rendimiento al gestionar cachés y optimizar la carga de datos. El **diseño atómico** asegura que los componentes sean ligeros y eficientes, evitando la redundancia y mejorando la velocidad de renderizado.
3. **Manejo de Autenticación y Autorización:**

- **Next.js** se puede combinar con **next-auth** para manejar la autenticación y la gestión de sesiones. **Graphql** permite realizar consultas y mutaciones basadas en el estado de autenticación del usuario.
- **Apollo Client** incluye tokens de autenticación en las solicitudes Graphql, asegurando que solo los usuarios autenticados puedan acceder a datos protegidos. El **diseño atómico** permite crear componentes de interfaz de usuario específicos para la autenticación, como formularios de inicio de sesión y paneles de usuario.

4. **Desarrollo Basado en Componentes:**
   - **Next.js** y **Graphql** promueven un enfoque basado en componentes al permitir que cada componente solicite solo los datos que necesita. **Apollo Client** facilita la integración de consultas Graphql en componentes de React.
   - **Diseño Atómico** alinea bien con este enfoque al fomentar la creación de componentes pequeños y reutilizables que se ensamblan en componentes más grandes. Esto mejora la modularidad y la separación de responsabilidades en la aplicación.

5.7 Relación con los microservicios

Implementar GraphQL con Next.js y Apollo en una arquitectura de microservicios puede transformar la manera en que se manejan los datos y se desarrolla el front-end de las aplicaciones. En lugar de usar tecnologías específicas como Spring Cloud para gestionar microservicios, esta combinación de tecnologías proporciona una solución moderna y eficiente para crear aplicaciones web escalables y bien estructuradas.

Para construir una arquitectura robusta con **Next.js** y **Apollo**, primero es esencial entender cómo se interrelacionan estas tecnologías dentro del ecosistema de microservicios. **Next.js** se encarga de la creación de aplicaciones React con capacidades de renderizado del lado del servidor (SSR) y generación de sitios estáticos

(SSG), mientras que **Apollo Client** gestiona las consultas a GraphQL y el manejo de caché en el front-end. GraphQL, por su parte, ofrece una forma eficiente y flexible de consultar los datos necesarios para la aplicación.

En una arquitectura de microservicios, cada microservicio tiene una función específica y se comunica con otros servicios a través de APIs. En este contexto, **GraphQL** se convierte en una herramienta poderosa al permitir que los microservicios expongan sus datos de manera estructurada y flexible. Esto es especialmente útil cuando se trata de consultar datos desde diferentes microservicios y combinarlos en una sola solicitud.

Next.js y **Apollo** se integran para proporcionar una experiencia de usuario fluida. Next.js maneja el renderizado en el servidor y la generación estática, mientras que Apollo Client se encarga de la gestión de datos del lado del cliente. Juntos, permiten que la aplicación se beneficie de una carga rápida y una experiencia de usuario reactiva, sin necesidad de realizar múltiples solicitudes a diferentes microservicios para obtener datos.

En una arquitectura de microservicios, el enrutamiento y la gestión de solicitudes son cruciales. Mientras que Spring Cloud Gateway maneja el enrutamiento y la exposición de APIs en el ecosistema de microservicios basado en Spring, en el mundo de Next.js y Apollo, el enrutamiento y la gestión de datos se manejan de manera diferente. En lugar de usar un gateway para enrutar solicitudes entre microservicios, la aplicación Next.js puede realizar consultas a un único punto de entrada GraphQL, que se encarga de combinar y gestionar las solicitudes a los diferentes servicios backend.

La gestión de la autenticación y la autorización en una arquitectura basada en Next.js y Apollo se maneja mediante **next-auth**, que proporciona una solución flexible y segura para gestionar sesiones y permisos. La autenticación se integra con GraphQL, permitiendo que las consultas y mutaciones sean autorizadas de acuerdo con el estado de autenticación del usuario.

Además, la integración de **Apollo Client** con **GraphQL** permite la optimización de las consultas y el manejo eficiente de la caché, evitando así la sobrecarga de solicitudes y mejorando el rendimiento general de la aplicación. La combinación de estas tecnologías proporciona un enfoque moderno para manejar datos y construir

aplicaciones web, en contraste con el uso de tecnologías tradicionales como Eureka y Ribbon en el ecosistema Spring Cloud.

5.7.1 Dependencias de un ecosistema basado en Next y Graphql

En un ecosistema de desarrollo moderno con Next.js, GraphQL, y Apollo, la correcta gestión de dependencias y la configuración son esenciales para construir aplicaciones escalables y eficientes. Cada una de estas tecnologías juega un papel crucial en la organización del flujo de datos, la optimización del rendimiento y la mejora de la experiencia del desarrollador.

1. Next.js

La instalación y configuración de **Next.js** es directa y sencilla mediante el uso de npm o yarn: npm install next react react-dom

Next.js proporciona una estructura clara para el manejo de rutas, la generación de páginas y la optimización del rendimiento. Al configurar un proyecto Next.js, el archivo package.json centraliza todas las dependencias del proyecto, facilitando el manejo de paquetes y librerías que la aplicación necesita.

2. Apollo Client

Al instalar **Apollo Client**, puedes centralizar todas las consultas y mutaciones en un solo punto, optimizando así el acceso a los datos.

npm install @apollo/client graphql

3. GraphQL Server

En un sistema distribuido o con microservicios, **GraphQL Server** centraliza las solicitudes de datos en un único endpoint. A través de resolvers, las consultas y mutaciones son distribuidas a los servicios backend adecuados, permitiendo que los clientes soliciten exactamente los datos que necesitan.

npm install apollo-server graphql

5. Manejo de configuraciones

En un ecosistema moderno, las configuraciones se gestionan a través de archivos como .env y next.config.js. Estos archivos contienen información sensible o de entorno, como claves de API, URLs de servidores o configuraciones de autenticación. Este enfoque de configuración permite que el entorno de la aplicación sea fácilmente adaptable a diferentes fases del desarrollo (desarrollo, pruebas, producción).

Por ejemplo, la configuración de la URL de un servidor GraphQL se puede manejar mediante un archivo .env:

NEXT_PUBLIC_GRAPHQL_URL=https://api.example.com/graphql

De esta manera, la configuración de la aplicación puede ser actualizada sin modificar el código fuente, lo que facilita la administración y despliegue en diferentes entornos.

6. Federación y Gateway de GraphQL

En una arquitectura de microservicios, la **Federación de GraphQL** o el uso de un **GraphQL Gateway** permite combinar múltiples servicios backend bajo un solo esquema de GraphQL. Esto simplifica la interacción del cliente con el backend, permitiendo que las consultas a diferentes microservicios se unifiquen y optimicen.

En lugar de realizar múltiples consultas a diferentes APIs, un **GraphQL Gateway** se encarga de distribuir las consultas a los microservicios correspondientes, combinando los resultados antes de enviarlos al cliente. Esto mejora el rendimiento y simplifica la lógica de front-end.

6 Conclusiones

Empezando con el tema de los componentes, la revisión y prueba de los componentes FilterNotasOrganism.js, GastosPage.js, GastosTemplate.js, Inventarios Atom.js, y ModalMolecule.js ha demostrado que la aplicación está bien construida, con un enfoque claro en la modularidad, la reusabilidad y la gestión eficiente de estados y datos. El rendimiento general del sistema es adecuado, aunque existen oportunidades para optimizar y escalar la aplicación en preparación para un mayor uso.

El código está bien cubierto por pruebas unitarias e integración, lo que proporciona una base sólida para mantener la calidad a medida que la aplicación evoluciona. Sin embargo, se recomienda mejorar ciertos aspectos de accesibilidad y optimización para garantizar que la aplicación sea accesible y eficiente para todos los usuarios, independientemente del contexto en el que se utilice.

En conjunto, este proyecto es un buen ejemplo de desarrollo de software moderno, con un fuerte enfoque en la calidad del código y la experiencia del usuario. Las áreas identificadas para mejoras ofrecen una hoja de ruta clara para futuras optimizaciones y desarrollo, asegurando que la aplicación pueda crecer y adaptarse a nuevas demandas sin comprometer su rendimiento o usabilidad.

Puntos a destacar del proyecto son la Modularidad y reusabilidad que hacen de este un proyecto escalable donde los componentes se adaptan al entorno y contexto que sea haciéndolos muy flexibles, y reduce el riesgo de cometer errores al copiar código en otras partes porque una vez definiendo el componente solido solo queda pasarle la lógica, pero el componente podrá recibir cualquier parámetro, función o propiedad.

La gestión efectiva de estados y datos es crucial en cualquier aplicación interactiva. Los componentes revisados demuestran un manejo adecuado del estado utilizando hooks de React como useState y useEffect, así como la integración con Apollo Client para gestionar consultas y mutaciones de Graphql.

En escenarios con un gran volumen de datos, como la carga de miles de registros en GastosTemplate.js, se observaron tiempos de respuesta más largos de lo esperado. Si bien el sistema sigue siendo funcional, una optimización adicional mediante la implementación de técnicas como la paginación, la virtualización de listas, o el uso de useMemo y useCallback para memorizar valores y funciones podría mejorar significativamente el rendimiento.

La accesibilidad y la experiencia de usuario son aspectos que no deben ser subestimados en el desarrollo de software moderno. Los componentes evaluados muestran un enfoque en la usabilidad, aunque hay margen para mejorar la accesibilidad.

Accesibilidad: Aunque los componentes son funcionales y responden bien a las interacciones del usuario, la inclusión de atributos ARIA y otras mejoras en accesibilidad podría hacer que la aplicación sea más inclusiva para usuarios con discapacidades. Elementos como modales y botones de selección podrían beneficiarse de etiquetas adicionales que describan su función y estado para los lectores de pantalla.

Interfaz de Usuario: La interfaz de usuario se diseñó con Tailwind CSS, lo que permitió una apariencia limpia y consistente en toda la aplicación. La reactividad de la interfaz, combinada con el uso de efectos visuales como cambios de color y transiciones suaves, contribuye a una experiencia de usuario positiva.

Al desarrollar este proyecto se pensó en una solución centralizada, escalable y customizable para gestionar puntos de venta por medio de un ERP lo cual creo firmemente haber logrado no solo desde la administración desde un usuario Master que tiene acceso a toda la aplicación para su administración si no también del lado del desarrollo al usar tecnologías como Graphql que es una alternativa a los métodos de solicitudes a APIs tradicionales que no solo se sale del esquema si no que aumenta el rendimiento y facilita la configuración y los procesos de gestión de datos, a su vez una metodología de diseño tan buena como lo es el diseño atómico proporciono una facilidad enorme a la hora de encajar componentes en los diversos módulos al tener una base sólida solo el reto quedaba en gestionar las propiedades y acciones de cada componente agilizando el proceso de armado de páginas y templates, su combinación con la librería de JavaScript React fue fundamental para los grande resultados en cuestión al diseño que se obtuvieron, Next JS aporto el enrutamiento necesario para tener un proyecto fácil de acceder todas esas tecnologías y sus recursos fueron encajadas a la perfección logrando una fácil dirección y administración de puntos de ventas centralizados en un contexto de gestión empresarial.

Referencias y bibliografía

1. L. Chen and L. Xu, "Centralized vs. Decentralized Information Systems in Multi-Location Firms: An Empirical Analysis," *Journal of Management Information Systems*, vol. 36, no. 3, pp. 928-955, 2019. doi: 10.1080/07421222.2019.1628897.
2. S. Newman and M. Abbot, *Building Microservices: Designing Fine-Grained Systems*. Sebastopol, CA, USA: O'Reilly Media, 2020.
3. Chen and L. Xu, "Centralized Management Systems for Multi-Store Retail Chains: Enhancing Operational Efficiency," *International Journal of Retail & Distribution Management*, vol. 48, no. 3, pp. 256-273, 2020. doi: 10.1108/IJRDM-08-2019-0308.
4. S. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley, 2002.
5. *M. Cohn,* User Stories Applied: For Agile Software Development*. Boston, MA, USA: Addison-Wesley, 2004.*
6. *T. D. Noller, "Designing Modular and Scalable Frontend Architectures with Atomic Design and React,"* IEEE Software*, vol. 36, no. 2, pp. 72-79, Mar.-Apr. 2019. doi: 10.1109/MS.2019.2896743.*
7. Li, Hongjun. "RESTful Web service frameworks in Java." *2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*. IEEE, 2011.
8. Deacon, John. "Model-view-controller (mvc) architecture." *Online][Citado em: 10 de março de 2006.] http://www. jdl. co. uk/briefings/MVC. pdf* (2009).
9. Olanrewaju, Rashidah F., Thouhedul Islam, and Nor'ashikin Ali. "An empirical study of the evolution of PHP MVC framework." *Advanced Computer and Communication Engineering Technology*. Springer, Cham, 2015. 399-410.
10. Bawiskar, Ankur, et al. "Spring Framework: A Companion to JavaEE." *IJCEM* 1 (2012): 41-49.
11. Radu, A. Corso, G. Lehmann Miotto, and L. Magnoni. "The electronic logbook for the information storage of ATLAS experiment at LHC (ELisA)." *Journal of Physics: Conference Series*. Vol. 396. No. 1. IOP Publishing, 2012.
12. Alfonzo, Pedro L., and Sonia I. Mariño. "Los estándares internacionales y su importancia para la industria del software." (2013).
13. M. E. Porter and J. Heppelmann, "How Smart, Connected Products Are Transforming Competition," *Harvard Business Review*, vol. 92, no. 11, pp. 64-88, Nov. 2014.
14. E. Brynjolfsson and A. McAfee, *The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies*, New York, NY, USA: W.W. Norton & Company, 2014.
15. T. H. Davenport and J. G. Harris, *Competing on Analytics: The New Science of Winning*, Boston, MA, USA: Harvard Business School Press, 2007.
16. C. M. Christensen, M. Raynor, and R. McDonald, "What Is Disruptive Innovation?" *Harvard Business Review*, vol. 93, no. 12, pp. 44-53, Dec. 2015.

17. J. K. Tarafdar, Q. Tu, and T. S. Ragu-Nathan, "Impact of Technostress on End-User Satisfaction and Performance," *Journal of Management Information Systems*, vol. 27, no. 3, pp. 303-334, Dec. 2010.
18. R. Amit and C. Zott, "Creating Value Through Business Model Innovation," *MIT Sloan Management Review*, vol. 53, no. 3, pp. 41-49, 2012.
19. A. E. Smith and M. S. Wright, "The Impact of Retail Technology on Employee Work Experiences," *Journal of Retailing*, vol. 84, no. 1, pp. 1-13, Mar. 2008.
20. G. G. Parker, M. W. Van Alstyne, and S. P. Choudary, *Platform Revolution: How Networked Markets Are Transforming the Economy--and How to Make Them Work for You*, New York, NY, USA: W.W. Norton & Company, 2016.
21. R. Lusch and S. Nambisan, "Service Innovation: A Service-Dominant Logic Perspective," *MIS Quarterly*, vol. 39, no. 1, pp. 155-175, Mar. 2015.
22. M. Chui, M. Löwer, and R. M. Gove, "Four Ways to Scale Your Digital Transformation," *McKinsey & Company*, June 2020. Available: https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/four-ways-to-scale-your-digital-transformation
23. A. Rincón Ramos, "Implantación de un ERP para una tienda de ultramarinos mediante venta online," 2024.
24. K. A. Silva Zambrano, "Desarrollo de aplicación web y móvil Android para el control y gestión de ventas aplicando técnicas de data mining en la Embotelladora Agua Fresh," Bachelor's thesis, Univ. Técnica de Cotopaxi (UTC), La Maná, Ecuador, 2023.
25. J. P. Canchig Rivera and B. K. Chilla Doicela, "Desarrollo de una aplicación web y móvil para el análisis de calidad de suelos en las parroquias rurales del Cantón Latacunga," Bachelor's thesis, Univ. Técnica de Cotopaxi (UTC), Latacunga, Ecuador, 2023.
26. D. M. Jami Casa and E. M. Jami Casa, "Desarrollo de una aplicación web y móvil para la Empresa de calzado 'Creaciones Chisag' ubicado en el cantón Latacunga, a través del uso de prácticas ágiles," Bachelor's thesis, Univ. Técnica de Cotopaxi (UTC), Latacunga, Ecuador, 2023.
27. J. Pérez, *Innovación tecnológica en el sector minorista: Estrategias y casos de estudio*, Editorial Tecnológica, 2020.
28. I. Sommerville, *Software Engineering*, 10th ed., Pearson, 2016.
29. V. Sahni, A. Chopde, M. Goswami, and A. Kumar, "MERN (MongoDB, Express-JS, React-JS, Node-JS) Stack Web-Based Themefied Education Platform For Placement Preparation," *Educational Administration: Theory and Practice*, vol. 30, no. 5, pp. 1918-1928, 2024.
30. C. Bhatt, D. Tiwari, D. Dua, S. Gupta, and T. Singh, "Elevating Online Retail: An In-Depth Look at the Implementation of React JS in Advanced E-commerce," in *2024 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE)*, 2024, pp. 1-4, IEEE.
31. A. Cuesta Santos, M. Delgado Fernández, S. Fleitas Triana, and M. D. L. Á. Linares Borrell, "Optimización del capital humano por innovación en procesos de gestión humana y del conocimiento," *Anales de la Academia de Ciencias de Cuba*, vol. 13, no. 1, 2023.
32. Agile Alliance, "The Agile Manifesto," 2023. [Online]. Available: https://agilemanifesto.org

33. S. Kruchten, "Design Atomicity in Software Engineering," *IEEE Software*, vol. 32, no. 5, pp. 42-47, Sept.-Oct. 2015.
34. R. L. Glass, "Modularization and the Atomic Design Principle," *IEEE Software*, vol. 31, no. 2, pp. 110-112, Mar.-Apr. 2014.
35. A. H. Eden and Y. Hirshfeld, "Principles of Modular and Atomic Design in Software Development," *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1061-1073, Aug. 2013.