Pontificia Universidad Católica del Ecuador

Facultad De Ingeniería

Escuela de Sistemas

TEMA:

Desarrollo de un dashboard para la gestión de minoristas.

AUTOR:

JEAN PIERRE RAMIREZ VALLEJO

TRABAJO PREVIA A LA OBTENCIÓN DEL TÍTULO DE INGENIERO DE SISTEMAS DE LA INFORMACIÓN

QUITO, JUNIO - 2023

DEDICATORIA

El presente trabajo se lo dedico a mis padres por ser los principales pilares para mi formación tanto académica como personal. Este logro se lo dedico a ellos, gracias a su apoyo incondicional, he podido alcanzarlo.

A mi tía, quien me ha brindado una guía constante y me ha impulsado día a día para lograr este objetivo. Ella me ha demostrado que, a pesar de las dificultades en el camino, debo seguir adelante y es gracias a su apoyo y aliento que he llegado a ser la persona que soy hoy en día.

AGRADECIMIENTO

Agradezco profundamente a mis padres y a mi tía por su amor, apoyo y confianza en mí. Han sido mi fuente de inspiración y motivación en cada paso de este camino académico. Sin ellos, no habría sido posible lograr este éxito.

Les dedico este trabajo como una muestra de gratitud y reconocimiento por todo lo que han hecho por mí. Espero poder seguir honrándolos con mis logros futuros y continuar siendo una persona de la que estén orgullosos.

RESUMEN

El siguiente plan de titulación se refiere al desarrollo de un dashboard destinado a la gestión de minoristas. La justificación se basa en la necesidad de mejorar la eficiencia y competitividad de los minoristas mediante el uso de herramientas tecnológicas como los dashboards. Algunos minoristas aún no han adoptado estas soluciones, se argumenta que implementar un dashboard proporcionaría una forma efectiva de analizar y presentar datos para tomar decisiones informadas. El desarrollo del dashboard requerirá comprender las necesidades de los minoristas, seleccionar fuentes de datos apropiadas y utilizar herramientas de análisis de datos. Se menciona la importancia de los indicadores clave de rendimiento (KPIs) para impulsar el éxito de la organización. La gestión de minoristas se considera una habilidad y estrategia crucial para maximizar las ventas y la experiencia del cliente. El problema planteado se centra en la necesidad de visualizar datos precisos de manera dinámica y en tiempo real para facilitar la toma de decisiones diarias. Los objetivos específicos incluyen identificar los KPIs relevantes, analizar las fuentes de datos y desarrollar un dashboard de calidad y usabilidad. El alcance del proyecto se limita a desarrollar un dashboard que automatice la visualización y obtención de datos, proporcionando una interfaz intuitiva y accesible desde cualquier lugar con acceso a Internet. El objetivo final es mejorar la eficiencia de la toma de decisiones mediante el acceso y análisis de datos relevantes para el negocio minorista

ÍNDICE

Contenido

ÍNDICE DE FIGURAS, GRÁFICOS Y TABLAS

CAPÍTULO I: INTRODUCCIÓN

1. MARCO DE REFERENCIA

1.1. JUSTIFICACIÓN

Los minoristas representan un sector esencial dentro de la economía, abarcando una amplia variedad de tipos de negocios y establecimientos. A pesar de la importancia de contar con herramientas que permitan la gestión eficiente de estas empresas, aún existen minoristas que no han adoptado soluciones tecnológicas. Por lo tanto, resulta fundamental llevar a cabo un análisis y desarrollo de un dashboard para la gestión de minoristas en la ciudad, con el objetivo de mejorar su rendimiento y competitividad en el mercado.

La implementación de un dashboard de gestión de minoristas proporcionaría una solución efectiva para que los minoristas analicen y presenten sus datos de manera efectiva, lo que les permitiría tomar decisiones informadas. Sin embargo, el diseño y desarrollo de un dashboard eficiente y efectivo para la gestión de minoristas requiere una comprensión profunda de las necesidades y requisitos de los minoristas, así como la selección adecuada de las fuentes de datos y herramientas de análisis de datos.

Key Performance Indicators (KPIs) son aquellos indicadores que se centran en los aspectos organizativo rendimiento que son los más crítico para la presente y futuro éxito de la organización. La información derivada de los indicadores puede vaciarse en visualizaciones que favorece a las metas empresariales. (David Parmenter, 2013)

La gestión de minoristas es un conjunto de habilidades y estrategias que administra de manera efectiva una tienda o una red de tiendas. Una gestión efectiva puede ayudar a maximizar las ventas y las ganancias, al mismo tiempo que permite que el cliente tenga una experiencia positiva y memorable.

Por lo tanto, el análisis de datos y desarrollo de un dashboard para la gestión de minoristas es una tarea importante y desafiante que tiene como objetivo mejorar la gestión y el rendimiento de los minoristas mediante la automatización y visualización de datos. Este plan justifica la necesidad de esta tarea y proporcionará una solución efectiva para mejorar la gestión y el rendimiento de los minoristas.

1.2. PLANTEAMIENTO DEL PROBLEMA

Según Freddy Rodolfo Lalaleo, “El impacto que ha tenido el Internet en los últimos años ha propiciado que cambie en gran medida el comportamiento del consumidor”, en (Revista de Ciencias de la Administración y Economía Del Ecuador, 2021). Esto ha hecho que el mercado se adapte y mejore a la par con las nuevas tecnologías que surgen.

En los minoristas la precisión de los datos es el atributo más importante para el monitoreo de los visitantes. Es la base para decidir cuál es la mejor estrategia o método para poner en marcha a la tienda, necesitando visualizar datos de manera dinámica, sencilla y tiempo real, para que los comercios minoristas tomen decisiones día a día, no es una tarea sencilla. Por esto, también se toma en cuenta distintos aspectos que arrojan la información y datos en la optimización de los procesos de la tienda. La clave principal es la precisión de los datos, la y la elección de las tecnologías para monitorear a los visitantes.

1.3. OBJETIVO GENERAL

Desarrollar un dashboard para la gestión de minoristas.

1.4. OBJETIVOS ESPECÍFICOS

Identificar los cuatro indicadores clave de rendimiento (KPIs) relevantes para la gestión de minoristas, a través de una revision de los procesos de negocio. Utilizando técnicas de análisis de datos que mejor reflejen el desempeño de la empresa en áreas ventas.

Seleccionar y analizar las fuentes de datos necesarias para el dashboard. Utilizando técnicas de minería de datos y análisis exploratorio de datos para garantizar la calidad y coherencia de los datos obtenidos, así como la integración y transformación de estos en una única fuente para el dashboard.

Desarrollar un dashboard de calidad y usabilidad para la gestión de minoristas, utilizando técnicas de visualización de datos y diseño centrado en el usuario.

1.5. ALCANCE

Se propone desarrollar un dashboard que permita automatizar la visualización y obtención de datos, así como crear una interfaz intuitiva que facilite el análisis de información para una marca de minoristas. El dashboard permitirá una visualización rápida y accesible desde cualquier lugar con acceso a internet, lo que brindará información clara y útil para la gestión de minoristas. El objetivo es mejorar la eficiencia de la toma de decisiones mediante una herramienta que facilite el acceso y análisis de datos relevantes para el negocio.

CAPÍTULO II: FUNDAMENTACIÓN TEÓRICA

2. Marco Teórico

2.1. Inteligencia de Negocios

La implementación de BI en el retail implica la utilización de diversas técnicas y herramientas para la recopilación, análisis y visualización de datos relacionados con el negocio. Algunas de las áreas críticas que pueden ser beneficiadas por la BI incluyen la gestión de inventario, la segmentación de clientes, la gestión de proveedores y la optimización de precios. (UEES, 2022)

2.2. KPIs (Key Performance Indicator)

Las KPI retail aportan datos concretos y tangibles sobre las ventas de un producto, sobre la acogida de dicho producto por parte del público objetivo (que en el caso del retail suele ser siempre el cliente final), así como de cualquier otro tipo de acción llevada a cabo con el fin de incrementar las ventas. (BeeTrack, 2023)

2.3. Análisis de Datos

El análisis de datos en el comercio retail consiste en analizar los datos derivados de la actividad de tu negocio para sacar conclusiones y conocer cuáles son sus puntos fuertes y débiles. Estos datos pueden ser de distinto tipo y su objetivo fundamental es servir para optimizar los recursos. (Status2, 2022)

2.4. Data Mining

Esencialmente, la minería de datos es el proceso de descubrir patrones, modelos y otros tipos interesantes del conocimiento en grandes conjuntos de datos. El término, minería de datos, como una visión vívida de la búsqueda de pepitas de oro a partir de datos, apareció en 1990s. Sin embargo, para referirnos a la extracción de oro de rocas o arena, decimos oro minería en lugar de minería de roca o arena. Análogamente, la minería de datos debería haber sido más apropiadamente llamado "extracción de conocimiento a partir de datos", que lamentablemente

es algo largo. Sin embargo, cuanto más corto término, la extracción de conocimiento puede no reflejar el énfasis en la extracción de grandes cantidades de datos. (Morgan Kaufmann, 2022).

2.5. Análisis de canasta

El análisis de canasta es una técnica de minería de datos que se utiliza comúnmente en el sector retail para identificar patrones en las compras de los clientes. El objetivo principal del análisis de canasta es mejorar la disposición de los productos en la tienda y aumentar las ventas, al identificar las combinaciones de productos que se compran juntos con mayor frecuencia. Esta técnica también puede ayudar a mejorar la satisfacción del cliente al permitir que la empresa proporcione una experiencia de compra personalizada y relevante. (Pateli, A. 2017)

2.6. React js

React es una librería Javascript focalizada en el desarrollo de interfaces de usuario. Así se define la propia librería y evidentemente, esa es su principal área de trabajo. Además, facilita mucho el desarrollo, ya que nos ofrece muchas cosas ya listas, en las que no necesitamos invertir tiempo de trabajo. (Desarrolloweb.com, 2016)

2.7. Dashboard

El uso del dashboard también permite medir el rendimiento de la empresa en función del tiempo, y compararlo con sus objetivos y previsiones. También es posible comparar el rendimiento de varios puntos de venta durante un mismo periodo. (DigDash, 2021)

2.8. Metodologías

2.8.1. SCRUM

La metodología Scrum permite abordar proyectos complejos desarrollados en entornos dinámicos y cambiantes de un modo flexible. Está basada en entregas parciales y regulares del producto final en base al valor que ofrecen a los clientes. (Hurtado, J. S. en IEBS, 2021)

2.9. Herramientas de trabajo

2.9.1. Crisp-DM

“Proporciona una descripción normalizada del ciclo de vida de un proyecto estándar de análisis de datos, de forma análoga a como se hace en la ingeniería del software con los modelos de ciclo de vida de desarrollo de software. El modelo CRISP-DM cubre las fases de un proyecto, sus tareas respectivas, y las relaciones entre estas tareas. En este nivel de descripción no es posible identificar todas las relaciones; las relaciones podrían existir entre cualquier tarea según los objetivos, el contexto, y el interés del usuario sobre los datos”. (Gil, B. 2016).

2.10. Selección de datos

2.10.1. Fuente de datos

“El conjunto de datos disponible contiene información de compras realizadas en 10 centros comerciales diferentes en Estambul entre los años 2021 y 2023. La información recopilada comprende datos de distintos grupos etarios y géneros, con el objetivo de brindar una visión completa sobre los hábitos de compra en la ciudad. Los datos incluyen números de factura, identificaciones de clientes, edades, géneros, métodos de pago, categorías de productos, cantidades, precios, fechas de pedidos y ubicaciones de los centros comerciales. Se espera que este conjunto de datos sea de gran utilidad para investigadores, analistas de datos y entusiastas del aprendizaje automático que deseen obtener información sobre las tendencias y patrones de compra en Estambul.” (Aslan,2023)

2.11. Mockups

“Es un prototipo hecho antes del desarrollo del trabajo. Sirve para transformar ideas en funcionalidades y ayuda al cliente a exteriorizar y comprender lo que necesita.

Normalmente tenemos una idea general de lo que queremos, pero no tenemos tanta seguridad respecto a los detalles de desarrollo, como por ejemplo la ubicación de los botones y del contenido, o cómo las páginas y funcionalidades van a interactuar entre sí”. (Pagés S., 2018)

2.12. Postman

“Es una plataforma que permite y hace más sencilla la creación y el uso de APIs. Esta herramienta es muy útil para programar porque da la posibilidad hacer pruebas y comprobar el correcto funcionamiento de los proyectos que realizan los desarrolladores web” (Assembler Institute, 2022)

2.13. Base de datos

2.13.1. MongoDB

Es una base de datos NoSQL orientada a documentos. Se diferencia de las bases de datos relacionales por su flexibilidad y rendimiento. Descubre todo lo que necesitas saber sobre esta herramienta imprescindible para la ingeniería de datos. (DataScientest, 2022)

2.14. Lucidchart

Permite que los usuarios creen borradores y compartan diagramas de flujo profesionales, proporcionando diseños para todo, desde procesos de lluvia de ideas hasta administración de proyectos. (Todd McKinnon, 2023)

2.15. Diagramas de caso de uso

“Es una forma de diagrama de comportamiento en lenguaje de modelado unificado (UML, del inglés Unified Modelling Language), con la que se representan procesos empresariales, así como sistemas y procesos de programación orientada a objetos. Por lo tanto, UML no es un lenguaje de programación, sino un lenguaje de modelado, es decir, un método estandarizado para representar sistemas planificados o ya existentes. En este diagrama, todos los objetos involucrados se estructuran y se relacionan entre sí.” (IONOS Digital, 2023)

2.16. Worflows

Un workflow, o flujo de trabajo en español, es un conjunto de actividades relacionadas, que son completadas en un determinado orden para alcanzar un objetivo de la organización. Estas actividades, o tareas, son realizadas por los llamados «participantes» del proceso, que pueden ser humanos o no (en ese caso, pueden ser software, máquinas, etc.). (INTEGRADOC BPM, 2020)

2.17. Tasa de error

Es la capacidad del sistema para ofrecer una tasa baja de errores, apoyar a los usuarios a cometer pocos errores durante el uso del sistema, y en caso de que cometan errores ayudarles a recuperarse fácilmente. (Walter Sanchez, 2018).

2.18. Tasa de éxito

Consiste en medir la eficiencia de la tarea, es decir, el tiempo que el usuario debe dedicar para completar y la efectividad de esta, que se refiere a la cantidad de usuarios que finalizan con éxito la tarea. (Antonio Marquez, 2022)

CAPÍTULO III: METODOLOGÍA

3. Metodología

La investigación aplicada es un elemento esencial en el desarrollo de proyectos exitosos. En el caso de un dashboard para la gestión de minoristas, la investigación aplicada puede ayudar a identificar las necesidades y expectativas de los minoristas en términos de herramientas de gestión y análisis de datos.

Al incorporar la investigación aplicada para el proyecto, se puede garantizar que el resultado final sea efectivo y útil para los minoristas. Además, la investigación aplicada puede ayudar a tomar decisiones informadas sobre el diseño, funcionalidad y tecnología necesarios para desarrollar un dashboard de calidad.

En el presente proyecto la investigación aplicada también puede ayudar a identificar los desafíos específicos que enfrentan los minoristas y las mejores prácticas de la industria para abordar estos desafíos. Esto puede incluir la identificación de las herramientas y tecnologías más efectivas para el análisis de datos y la presentación de información relevante para la toma de decisiones.

3.1. Metodología de desarrollo

Las metodologías tradicionales se centran en cumplir con procesos y mantener un control estricto, siguiendo una serie de etapas y procesos claramente definidos. Estos métodos han demostrado ser efectivos y exitosos en proyectos de gran envergadura en términos de tiempo y recursos necesarios. Tradicionalmente, las metodologías de gestión de proyectos como PMBOK y PRINCE2 han tenido una fuerte orientación predictiva. Se basan en una planificación detallada, partiendo del análisis funcional/técnico y los requisitos del producto, definiendo fases y actividades perfectamente planificadas en el tiempo, considerando los recursos disponibles. Durante el transcurso del proyecto, el objetivo es lograr que se cumplan los plazos, costos, calidad y tiempo establecidos inicialmente (Fabiola, 2015).

En contraste, las metodologías ágiles ponen un mayor énfasis en las personas que participan en el proyecto, considerando fundamental la comunicación entre todos sus miembros: cliente, director de proyecto, equipo del proyecto e interesados. Estas metodologías no se enfocan en una entrega final con el producto terminado, sino que buscan un enfoque iterativo e incremental. Su flexibilidad y adaptabilidad hacia los cambios en las características del producto, requisitos o tiempos de desarrollo han demostrado dar buenos resultados. Según un estudio comparativo de metodologías tradicionales y ágiles, los proyectos gestionados con metodologías ágiles se inician sin un detalle cerrado de lo que va a ser construido. Además, a nivel comercial, los proyectos pueden ser vendidos como servicios en lugar de productos.

En este caso al aplicar esta metodología ágil, se considera la adaptabilidad y flexibilidad ante los cambios en las necesidades y preferencias de los minoristas. El enfoque iterativo permite una mayor interacción y retroalimentación con los usuarios finales, lo que facilita la incorporación de mejoras y ajustes según sus necesidades específicas. Al considerar el proyecto como un servicio en lugar de un producto, se promueve una relación cercana entre el equipo de desarrollo y los minoristas, lo que permite una mayor alineación con sus objetivos y requisitos en tiempo real.

Se eligió una metodología ágil como Scrum para el desarrollo del dashboard, prioriza la flexibilidad, adaptabilidad y comunicación continua entre los miembros del equipo, lo que permite una gestión efectiva del proyecto y la entrega constante de resultados satisfactorios para el cliente.

3.1.1. Etapas de Sprint

Para el desarrollo de un dashboard utilizando la metodología Scrum, se propone una división en cuatro sprints:

Sprint 1:

En el primer sprint, se enfoca en el diseño del dashboard, estableciendo la estructura visual y la disposición de los elementos. Bajo la premisa de ofrecer una experiencia de usuario intuitiva y atractiva, se realizan análisis de requisitos y se define la arquitectura del dashboard, teniendo en cuenta los flujos de interacción esperados. Se crean prototipos y se diseñan wireframes para representar la estructura y el flujo de interacción de manera visual. Mediante el uso del framework React, se desarrollan los componentes y estilos necesarios, prestando especial atención a los principios de diseño centrados en el usuario. Se implementan las vistas iniciales del dashboard, incluyendo la disposición de paneles, gráficos y elementos de navegación, priorizando una interfaz de usuario limpia, intuitiva y visualmente agradable.

Sprint 2:

El enfoque se dirige hacia la conexión de la fuente de datos con el dashboard, mientras se mantiene una atención constante en la experiencia de usuario. Se configura y establece la conexión con la API de Flask para obtener los datos necesarios, asegurándose de que el proceso de carga de datos sea eficiente y rápido para mantener una experiencia fluida para el usuario. Se desarrollan componentes y lógica para la gestión de las solicitudes y respuestas de la API, teniendo en cuenta la necesidad de retroalimentación visual durante la carga y actualización de datos. Se implementan mecanismos de autenticación y seguridad para acceder a los datos de manera adecuada, brindando una capa de protección adicional para los usuarios del dashboard.

Sprint 3:

En el tercer sprint, además de optimizar el rendimiento del dashboard, se continúa mejorando la experiencia de usuario. Se llevan a cabo ajustes finos en la interfaz de usuario, como la mejora de estilos, tamaños, colores y elementos visuales para lograr una apariencia y una legibilidad óptimas. Se implementan funcionalidades adicionales requeridas, como filtros avanzados, búsqueda de datos, ordenación y paginación, teniendo en cuenta la usabilidad y la facilidad de uso. Se desarrollan gráficos interactivos y visualizaciones avanzadas utilizando bibliotecas especializadas como Plotly o D3.js, asegurándose de que sean fáciles de entender y manipular para el usuario final. Se realizan pruebas exhaustivas para garantizar la funcionalidad correcta de las optimizaciones y las nuevas funcionalidades implementadas, prestando especial atención a la experiencia de usuario durante el uso normal del dashboard.

Sprint 4:

En el último sprint del desarrollo del dashboard, se lleva a cabo el lanzamiento, teniendo en cuenta los aspectos de UI/UX en todo momento. Se dedica especial atención a la refinación de la interfaz de usuario, realizando ajustes y mejoras en los estilos, tamaños, colores y elementos visuales para garantizar una experiencia atractiva y coherente. Se llevan a cabo pruebas de usuario exhaustivas para asegurar que el dashboard cumple con las necesidades y expectativas de los usuarios que lo utilizarán. Además, se recopilan comentarios y métricas a través de encuestas de utilidad del diseño y experiencia de usuario, lo que proporciona información valiosa para mejorar los análisis y la usabilidad del dashboard.

3.2. Exploración e investigación de KPIs

Los indicadores son medidas utilizadas para evaluar el desempeño de un negocio en relación con sus objetivos, y son herramientas esenciales para la toma de decisiones estratégicas.

Se investigó los KPIs más importantes en el área de minoristas, considerando su relevancia para los objetivos y estrategias de la empresa. La investigación se realizó a expertos de varios retails reconocidos del país, explicando la relevancia de cada KPI en el departamento de ventas. Durante la investigación se exploraron diferentes perspectivas para obtener una visión completa de los indicadores más relevantes para evaluar el rendimiento del negocio desde el área de venta. Para ello, se tomaron en cuenta tres puestos de trabajo importantes en el departamento: la perspectiva del asesor de ventas, del jefe de local y del administrador de sector. Desde cada una de estas perspectivas, se identificaron y analizaron los KPIs más relevantes para cada rol dentro de la organización, con el objetivo de proporcionar una visión completa y detallada sobre la importancia de los indicadores clave de rendimiento en el sector de retail, identificando los siguientes KPIs como los más importantes para un negocio de retail:

a) Ventas

Las ventas son el KPI más importante para cualquier negocio de retail, ya que es el indicador más directo del éxito comercial. Desde la perspectiva de un asesor de ventas, es la medida más clara de su desempeño y su contribución al negocio. El jefe de local identifica la estrategia de marketing y ventas, y un reflejo de la satisfacción del cliente y competitividad del negocio. También para el administrador de zector, las ventas son una métrica esencial para evaluar el rendimiento del sector y tomar decisiones estratégicas en términos de presupuesto, personal y stock.

b) Frecuencia de compras

La frecuencia de compras es una métrica crucial en el contexto empresarial que cuantifica la regularidad con la que los clientes realizan adquisiciones en un establecimiento. Desde la perspectiva del asesor de ventas, la frecuencia de compras se considera un indicador fundamental de la eficiencia en la comercialización de productos y servicios, demostrando la capacidad para generar transacciones repetidas y contribuir al éxito financiero de la empresa. Para el gerente del local, esta métrica adquiere una importancia significativa al evaluar la rentabilidad del negocio, dado que una frecuencia de compras elevada implica una base de clientes leales y satisfechos que retornan de manera regular, generando ingresos constantes a largo plazo. Además, la frecuencia de compras también influye en las decisiones relacionadas con la estrategia de precios, promociones y descuentos, permitiendo maximizar el retorno de inversión y la satisfacción del cliente. Por otro lado, desde la perspectiva del administrador de sector, la frecuencia de compras se considera una métrica esencial para evaluar la eficiencia de la cadena de suministro y la rentabilidad del sector en comparación con otros segmentos. En síntesis, la frecuencia de compras se erige como una métrica estratégica que provee información valiosa sobre el comportamiento de compra de los clientes, la rentabilidad del negocio y la eficiencia de la cadena de suministro.

c) Tasa de conversión

Es un indicador clave de rendimiento que refleja la capacidad de una tienda para transformar visitantes en clientes. Desde la perspectiva del asesor de ventas, esta métrica mide su habilidad para persuadir y cerrar ventas. En tanto, desde la perspectiva del jefe de local, la tasa de conversión es un indicador clave para evaluar la efectividad de las estrategias de marketing y ventas y la satisfacción del cliente. Finalmente, desde la perspectiva del administrador de sector, la tasa de conversión es un factor crítico en la toma de decisiones estratégicas relacionadas con la disposición del local, el personal y los productos.

d) Ticket promedio

Es un KPI crucial en el sector de retail, ya que permite medir el valor promedio de cada venta realizada. Desde la perspectiva del asesor de ventas, este indicador refleja su capacidad para impulsar la venta de productos complementarios y promociones, incrementando el valor de cada venta. En tanto, para el jefe de local, el ticket promedio es fundamental para evaluar la rentabilidad del negocio y tomar decisiones estratégicas respecto a la disposición de los productos y las promociones en el local. Por último, el administrador de sector utiliza este KPI como un indicador para evaluar la eficacia de las estrategias de marketing y ventas implementadas, lo que le permite tomar decisiones estratégicas para el negocio en su conjunto.

CAPÍTULO IV: DESARROLLO DE LA INVESTIGACIÓN

En el contexto del sector retail, es esencial contar con un dashboard que permita analizar y visualizar los datos clave para la toma de decisiones estratégicas. Para desarrollar un dashboard efectivo, es necesario realizar un análisis exhaustivo de los indicadores clave de rendimiento (KPIs) relevantes para el negocio.

El primer objetivo consiste en identificar los KPIs más útiles para el sector retail, tales como las ventas totales, el margen de beneficio, el inventario, la rotación de productos y el tráfico de clientes. Estos indicadores son seleccionados en función de los objetivos y prioridades específicas de cada empresa. Una vez establecidos los KPIs, se procede a recopilar y procesar los datos utilizando la metodología CRISP-DM. Esta metodología proporciona una estructura sistemática que abarca varias etapas, incluyendo la comprensión del negocio, la comprensión de los datos, la preparación de los datos, el modelado, la evaluación y la implementación.

En la etapa de comprensión del negocio, se investiga a fondo el contexto empresarial, los requisitos del dashboard y las necesidades de información. Posteriormente, se realiza la comprensión de los datos disponibles, su calidad, estructura y características. La preparación de los datos implica llevar a cabo procesos de limpieza, transformación y creación de variables relevantes para el análisis. A continuación, se pasa a la etapa de modelado, donde se aplican técnicas estadísticas y de visualización de datos para obtener información significativa.

Finalmente, se procede al desarrollo del dashboard, poniendo un enfoque especial en la experiencia del usuario. Esto implica diseñar una interfaz intuitiva y fácil de usar, que permita a los usuarios interactuar con los datos de manera eficiente. Se deben considerar aspectos como la selección adecuada de gráficos, la organización de la información y la posibilidad de personalizar las visualizaciones según las necesidades individuales.

4.1. Variables y fórmulas de KPIs

Se explica a detalle las variables y fórmulas que se pueden utilizar para calcular cuatro KPIs importantes en el área de minoristas. Para cada uno de los KPIs antes mencionados, se describen las variables necesarias y la fórmula correspondiente que se pueden aplicar al conjunto de datos para obtener la información requerida. Esta información es útil para los profesionales del área de ventas de retail que deseen medir y mejorar el rendimiento de su tienda minorista.

Tasa de conversión: La tasa de conversión mide la proporción de visitantes de la tienda que realizan una compra. Para calcular la tasa de conversión, se utilizarán las siguientes variables y fórmula:

Variables:

ID único de cada cliente

Número de factura de la compra

Fórmula:

$$Tasa\ de\ conversión\ =\ (Número\ de\ facturas\ /\ Número\ de\ clientes)\ x\ 100$$

Ventas totales: Las ventas totales representan el monto total de las ventas realizadas en la tienda durante un período determinado. Para calcular las ventas totales, se utilizarán las siguientes variables y fórmula:

Variables:

Número de factura de la compra

Precio total de la compra

Fórmula:

$$Ventas\ totales\ =\ Suma\ de\ los\ precios\ totales\ de\ todas\ las\ facturas$$

Ticket promedio: El ticket promedio es el promedio de ventas por transacción. Para calcular el ticket promedio, se utilizarán las siguientes variables y fórmula:

Variables:

Número de factura de la compra

Precio total de la compra

Fórmula:

$$Ticket\ promedio\ =\ Ventas\ totales\ /\ Número\ total\ de\ facturas$$

La frecuencia de compra se refiere a la cantidad de veces que un cliente realiza compras en un determinado período de tiempo. Esta métrica proporciona información sobre la lealtad y el compromiso del cliente con el negocio, así como la frecuencia con la que regresan para realizar nuevas transacciones.

Para calcular la frecuencia de compra, se pueden utilizar las siguientes variables y fórmula:

Variables:

Número de clientes únicos

Número total de compras realizadas

Fórmula:

$$Frecuencia\ de\ compra\ =\ Número\ total\ de\ compras\ realizadas\ /\ Número\ de\ clientes\ únicos$$

La identificación adecuada de las variables y fórmulas correctas es un componente crítico para calcular los indicadores clave de rendimiento (KPIs) de manera efectiva en el análisis de datos. Al utilizar el conjunto de datos de compras de clientes proporcionado por Kaggle, se

seleccionaron cuidadosamente las variables y fórmulas apropiadas para garantizar la precisión y validez de los resultados.

Los KPIs obtenidos pueden ayudar a los minoristas a identificar áreas de mejora en sus operaciones y estrategias de ventas para impulsar el crecimiento y el éxito del negocio. Además, la colaboración de los empleados de la tienda, como un asesor de ventas, jefe de local y administrador de sector, es esencial para el análisis de los datos y en la implementación de acciones para mejorar el rendimiento de la tienda. La participación de estos empleados puede ayudar a recopilar información relevante y a realizar un análisis más profundo para obtener resultados más precisos y significativos.

4.2. Comprensión del negocio

El sector minorista es un componente fundamental de la economía, abarcando una amplia variedad de negocios y establecimientos que proporcionan bienes y servicios a los consumidores. Según un artículo publicado en Harvard Business Review, "el comercio minorista es el segundo sector más grande de la economía global, después de la energía" (Feloni, 2018). La importancia de este sector es evidente en la cantidad de empleos que genera, así como en su impacto en la calidad de vida de las personas y en la economía en general.

El análisis de canasta es una técnica de minería de datos, se utiliza para identificar patrones de compra de los clientes. El objetivo de este análisis es mejorar la disposición de los productos en la tienda y aumentar las ventas, al identificar las combinaciones de productos que se compran juntos con mayor frecuencia. Además, el análisis de canasta puede ayudar a mejorar la satisfacción del cliente al permitir que la empresa proporcione una experiencia de compra personalizada y relevante. Para aplicar esta técnica, se puede utilizar la metodología CRISP-DM, comenzando con el entendimiento del negocio y la recopilación de datos relevantes sobre las compras de los clientes.

4.3. Comprensión de los datos

La comprensión de datos desempeña un papel fundamental en los proyectos de minería de datos, ya que sienta las bases para un análisis posterior efectivo. Dentro del marco de la metodología CRISP-DM (Cross-Industry Standard Process for Data Mining), la fase de comprensión de datos ocupa un lugar crucial entre la comprensión del negocio y la preparación de los datos.

Durante esta etapa, se lleva a cabo un examen exhaustivo de los datos disponibles, con el propósito de obtener un conocimiento profundo de su estructura, calidad y potencial. Es en esta fase donde se descubren los aspectos ocultos dentro de los datos y se adquiere una comprensión de las particularidades que influirán en el desarrollo del proyecto.

A lo largo de la comprensión de datos, se recopilan y evalúan los conjuntos de datos disponibles, se describen sus características clave y se identifican cualquier problema o anomalía que pueda impactar en los análisis subsiguientes. Mediante la aplicación de técnicas de visualización, estadística y análisis exploratorio, se busca descubrir patrones, tendencias y relaciones que permitan extraer información valiosa.

invoice_no: Número de factura. Nominal. Una combinación de la letra 'I' y un número entero de 6 dígitos asignado de forma única a cada operación.

customer_id: Número de cliente. Nominal. Una combinación de la letra 'C' y un número entero de 6 dígitos asignado de forma única a cada operación.

gender: Variable de cadena del género del cliente.

age: Variable entera positiva de la edad del cliente.

category: String variable de la categoría del producto adquirido.

quantify: Las cantidades de cada producto (artículo) por transacción. Numérico.

price: Precio unitario. Numérico. Precio del producto por unidad en liras turcas (TL).

payment_method: variable de cadena del método de pago (efectivo, tarjeta de crédito o tarjeta de débito) utilizado para la transacción.

invoice_date: fecha de la factura. El día en que se generó una transacción.

shopping_mall: variable String del nombre del centro comercial donde se realizó la transacción.

Se seleccionará las siguientes para el análisis de canasta en una tienda con el objetivo de obtener información relevante sobre los patrones de compra de los clientes.

invoice_no: Este es el identificador único de cada compra realizada en la tienda. Esta variable se utilizará para agrupar los productos comprados juntos en una sola transacción, es importante para identificar las transacciones de compra individuales que se utilizarán para construir la matriz de transacciones.

category: Este es el identificador único de cada producto vendido en la tienda. Se utilizará esta variable para identificar el tipo de productos que se compran juntos con mayor frecuencia. Esta variable es crucial para el análisis de canasta, permitiendo identificar las combinaciones de productos que se compran juntos con mayor frecuencia.

quantity: Este es el identificador de la cantidad de tipos de productos que un cliente compro en un periodo de tiempo. Se utilizará para identificar los productos que más compran las personas en una factura.

Además de estas dos variables, es posible que se requiera información adicional, como el precio unitario de cada producto, para calcular los ingresos totales generados por cada combinación de productos. La variable "customerID" también puede ser útil para segmentar a los clientes según su comportamiento de compra y crear KPIs para cada segmento.

4.4. Preparación de los datos

La fase de preparación de datos es una de las etapas más importantes de la metodología CRISP-DM. Durante esta fase, se recopilan, exploran, limpian, transforman y preparan los datos para su posterior análisis. En esta fase, el objetivo es garantizar que los datos sean de alta calidad y adecuados para su uso en el análisis.

Utilizaremos Python y Google Colab para realizar la preparación de datos. A lo largo de esta fase, realizaremos una serie de tareas, como la exploración y limpieza de los datos, la eliminación de valores faltantes y la transformación de los datos en formatos adecuados para el análisis. Ver Anexo A.

4.5. Modelado

En esta fase, se utiliza la información obtenida en la fase de preparación de datos para crear modelos predictivos o descriptivos.

El uso de herramientas de minería y análisis de datos en el sector minorista es de gran importancia, permitiendo a las empresas obtener información valiosa sobre sus clientes y su comportamiento de compra. Esta información se puede utilizar para mejorar la experiencia del cliente, tomar decisiones estratégicas y aumentar la rentabilidad. Por otro lado, la minería de datos también puede ayudar a las empresas a segmentar su base de clientes y personalizar su experiencia de compra en función de sus preferencias y necesidades, este enfoque puede ayudar a determinar patrones o directamente grupos en los que un minorista debe enfocarse más en trabajar. Las técnicas antes mencionadas en el sector minorista, según un estudio de McKinsey & Company encontró que el uso de tecnologías avanzadas de análisis de datos y personalización en el sector puede aumentar las ventas en un 10% y reducir los costos de marketing en un 20%. Esto demuestra claramente la importancia y el valor de utilizar técnicas de minería y análisis de datos en el sector minorista.

Se utilizará el algoritmo descriptivo no supervisado de reglas de asociación Apriori para identificar patrones en los datos de cada combinación de productos que se compran juntos con mayor frecuencia. El objetivo final de esta fase es obtener un modelo que pueda utilizarse para identificar las combinaciones de productos que se compran juntos con mayor frecuencia, lo que permitirá a la empresa tomar decisiones informadas sobre cómo optimizar la disposición de los productos en la tienda y mejorar las ventas.

Después de preparar los datos, se puede aplicar el algoritmo de Apriori para generar reglas de asociación. Se estableció una frecuencia mínima de compra del 1% utilizando el parámetro min_support. Luego, se generaron las reglas de asociación utilizando el parámetro metric="lift", que indica que se debe utilizar el coeficiente de lift para medir la fuerza de las reglas. Ver Anexo B.

A continuación, se evaluará el modelo generado para evaluar su desempeño y su capacidad para proporcionar información valiosa a la empresa.

4.6. Evaluación

Existen varias métricas importantes que se utilizan para evaluar la calidad y el rendimiento del modelo. Estas métricas pueden ayudar a entender cómo de bien el modelo está clasificando las observaciones y cómo de fuertes son las relaciones entre las variables.

Entre las métricas más utilizadas se encuentran el soporte, la confianza, el lift, el leverage y la convicción. El soporte mide la frecuencia absoluta de una regla en el conjunto de datos, mientras que la confianza mide la probabilidad condicional de que la consecuencia sea verdadera dada el antecedente. Por otro lado, el lift mide la fuerza de la relación entre el antecedente y la consecuencia, y la convicción mide la dependencia entre el antecedente y la consecuencia.

El leverage, por su parte, mide la diferencia entre la frecuencia observada de una regla y la frecuencia esperada si no hubiera ninguna relación entre el antecedente y la consecuencia.

Todas estas métricas pueden ser útiles en el análisis de modelos de Data Mining para entender las relaciones entre las variables y para identificar patrones y tendencias que puedan ser útiles en la toma de decisiones.

Un ejemplo está, en la primera fila de la tabla. Ver Anexo C. La primera regla indica que el 14.86% de las transacciones incluyen "Food & Beverage" como antecedente y el 19.87% de las transacciones incluyen "1" como consecuente. El soporte de la regla es de 0.030184, lo que significa que el 3.02% de las transacciones incluyen tanto "Food & Beverage" como "1" como antecedente y consecuente, lo que indica una asociación débil.

Las otras reglas siguen una estructura similar, con diferentes productos como antecedentes y consecuentes. Cada regla tiene un soporte de antecedente y un soporte de consecuente, confianza, lift, leverage y convicción.

El soporte de las reglas es relativamente bajo, lo que indica que en los elementos no existe mucha frecuencia en las transacciones. La confianza también es relativamente baja en la mayoría de los casos, lo que significa que la probabilidad de que un elemento aparezca dado otro elemento es bastante baja. El lift y la convicción también son en su mayoría bajos, lo que indica que no hay una fuerte relación de dependencia entre los elementos en las transacciones.

4.7. Desarrollo de dashboard

En el primer sprint se lleva a cabo el diseño del dashboard, centrándose en establecer la estructura visual y la disposición de los elementos de manera efectiva. Para garantizar una experiencia de usuario intuitiva y atractiva, se realizan análisis detallados de los requisitos y se define la arquitectura general del dashboard.

4.7.1. Flujo de trabajo actual

El diagrama de procesos ilustra el flujo de gestión de un retail, desde la extracción de datos hasta la entrega de reportes. El proceso comienza con la extracción de los datos desde la fuente

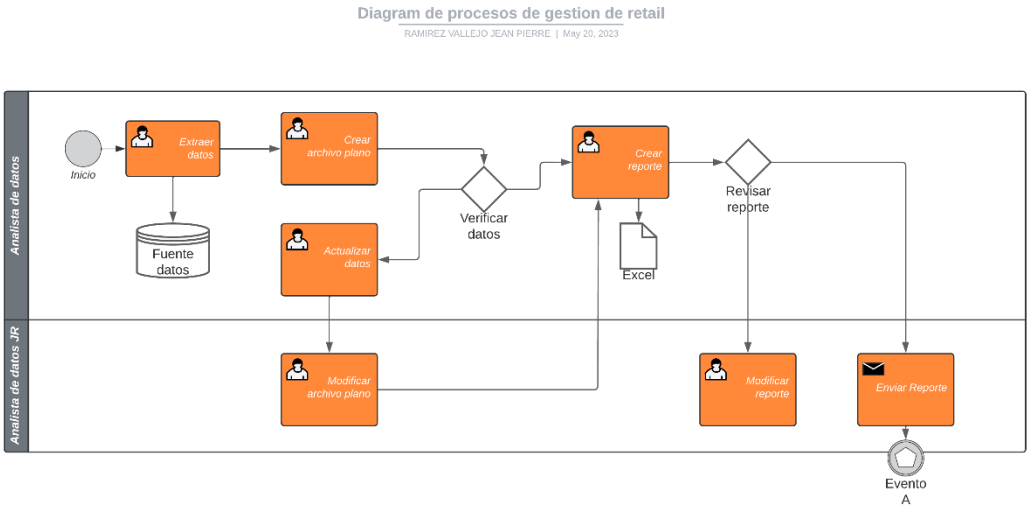
de datos correspondiente. Este paso es realizado por un analista de datos, quien se encarga de recopilar la información necesaria para su posterior procesamiento.

Una vez que los datos han sido extraídos, se crea un archivo plano donde se verifica si los datos requieren ser actualizados. En esta etapa, el analista de datos verifica la integridad y consistencia de los datos recopilados. En caso de detectarse alguna discrepancia o error en los datos, se procede a llamar a un Analista de Datos Junior para que realice las modificaciones necesarias. El Analista de Datos Junior notifica al analista de datos principal sobre la corrección realizada.

Posteriormente, se pasa a la creación del reporte, que se lleva a cabo utilizando un archivo Excel. Una vez creado el reporte, se realiza una revisión exhaustiva para determinar si se requiere alguna corrección adicional. Si se identifican errores o se requiere alguna modificación en el reporte, se notifica nuevamente al analista de datos junior para que realice los ajustes necesarios. Una vez que el reporte ha sido corregido, se informa al analista de datos principal para proceder con el envío del reporte.

En el caso de que no se encuentren inconsistencias en la verificación de los datos y el reporte esté completo y correcto, se procede a enviar directamente el reporte a los remitentes designados. Esto permite que el proceso continúe sin retrasos innecesarios.

Figura 1
Diagrama de procesos de reportes



Nota. El grafico representa un diagrama de procesos de reportes de un retail, en el que se ve el flujo de trabajo para la creación de un reporte. Elaborado en *Lucidchart*

En el segundo diagrama de flujo de procesos, se destaca el proceso de análisis y coaching en el contexto del retail. Este proceso involucra a tres actores principales: el asesor de ventas, quien se encarga de las ventas del local; el jefe de local, responsable de supervisar las operaciones en el establecimiento; y el administrador de sector, encargado de la gestión de los locales sectorizados.

El proceso comienza con la generación de un reporte que proporciona información relevante sobre el desempeño del local. Cada uno de los actores recibe este reporte y realiza un análisis individual para prepararse para una reunión semanal.

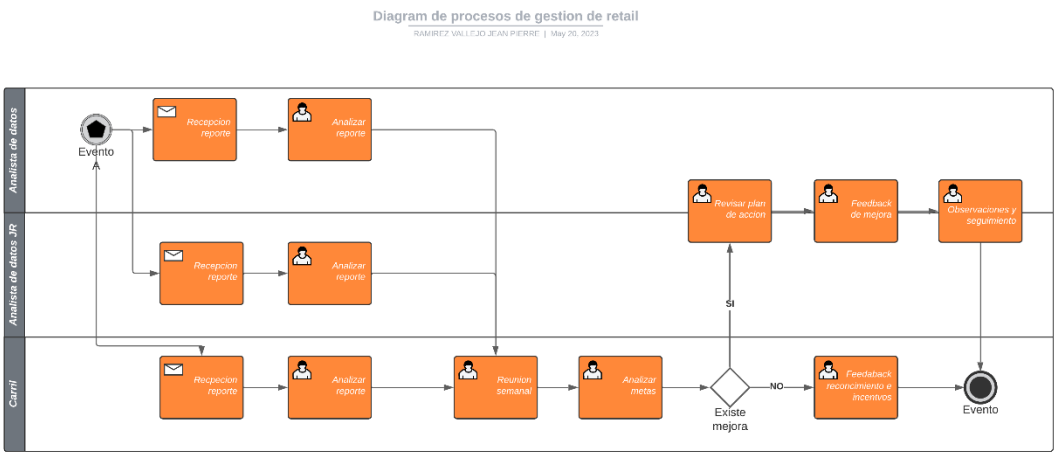
Durante la reunión, el administrador de sector lidera un análisis exhaustivo de las metas establecidas para el local en cuestión. Se evalúa si se ha logrado una mejora en comparación con el período anterior y se verifica si las metas establecidas se han cumplido. En caso de haberse logrado una mejora, se brinda un feedback breve y se reconoce e incentiva a los involucrados en el proceso, ya sea a través de reconocimientos formales o de incentivos.

Sin embargo, si no se ha observado una mejora en el desempeño, se procede a revisar detenidamente el plan de acción establecido previamente. Se analizan las estrategias implementadas y se identifican posibles desafíos o áreas de mejora. A continuación, se proporciona un feedback constructivo con el objetivo de promover la mejora continua. Se realizan recomendaciones y se establecen pautas claras para el cambio y el desarrollo.

Por último, se lleva a cabo un seguimiento y se realizan observaciones adicionales para garantizar que se implementen las acciones correctivas necesarias. Esto implica monitorear de cerca el desempeño del local y brindar apoyo adicional según sea necesario. El objetivo final es optimizar las operaciones y los resultados del local, asegurando un enfoque centrado en el logro de metas y la mejora continua.

Este proceso de análisis y coaching en el retail es fundamental para impulsar el crecimiento y el éxito de los locales. A través de un enfoque sistemático y colaborativo, se busca identificar áreas de mejora, proporcionar retroalimentación efectiva y promover un ambiente de trabajo motivador y orientado a resultados. Con la participación de los actores involucrados y un seguimiento riguroso, se busca garantizar la implementación de estrategias eficaces y el logro de metas establecidas.

Figura 2
Diagrama de procesos de coaching de reporte



Nota: El grafico representa el proceso de coaching dentro de un retail. Elaborado en *Lucidchart*

4.7.2. Análisis de caso de uso

La visualización de datos en tiempo real es uno de los requisitos fundamental para un dashboard en línea utilizado en la gestión de tiendas minoristas. Este aspecto implica que los usuarios puedan acceder a información actualizada sobre las ventas, el inventario, el rendimiento de los productos y otros indicadores clave. La capacidad de visualizar datos en tiempo real permite a los gerentes y ejecutivos obtener una visión precisa y actualizada del rendimiento de las tiendas, lo que les permite tomar decisiones informadas y reaccionar rápidamente ante cualquier cambio o tendencia identificada.

Otro requisito fundamental es la capacidad de generar informes comparativos. Esto significa que el dahsboard debe permitir a los usuarios comparar datos entre diferentes tiendas minoristas, regiones o períodos de tiempo. La capacidad de realizar comparaciones facilita la identificación de patrones, tendencias y diferencias significativas en el rendimiento de las tiendas. Por ejemplo, los usuarios pueden comparar las ventas de diferentes tiendas para identificar aquellas que son más rentables, o pueden comparar el rendimiento actual con el mismo período del año anterior

para evaluar el crecimiento o la disminución en el rendimiento. Los informes comparativos proporcionan una visión más completa y ayudan a los usuarios a tomar decisiones basadas en datos sólidos.

Actores principales: Analista de datos, asesor de ventas, jefe de local, administrador de sector

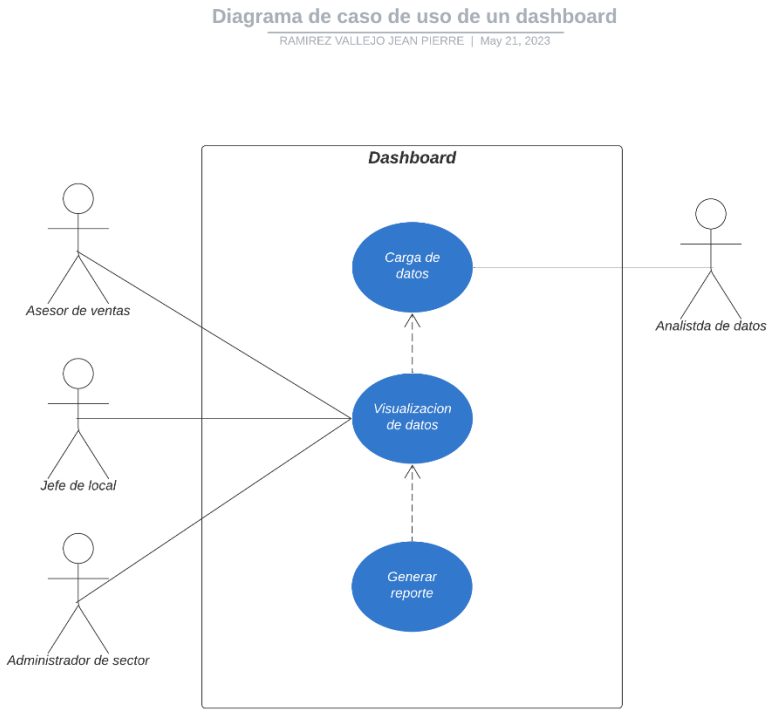
Descripción: Este caso de uso describe cómo los diferentes actores pueden utilizar el dashboard en línea para visualizar información relevante y descargar informes completos en el contexto de la gestión de tiendas minoristas.

Flujo del caso de uso:

1. El analista de datos carga los datos pertinentes al sistema, incluyendo información sobre ventas, inventario, rendimiento de productos y otros indicadores clave para el análisis y la gestión de retails.
2. El asesor de ventas inicia sesión en el sistema y accede al dashboard de gestión de retails.
3. El dashboard presenta una variedad de gráficos, tablas y métricas clave que representan el rendimiento de las tiendas minoristas, incluyendo ventas totales, margen de beneficio, rotación de inventario y eficiencia de las ventas.
4. El asesor de ventas puede personalizar la visualización del dashboard seleccionando los indicadores específicos que desea monitorear y aplicando filtros según su área de responsabilidad, como región geográfica, categoría de productos o línea de negocio.
5. Utilizando el dashboard, el asesor de ventas puede obtener información actualizada y en tiempo real sobre el rendimiento de las tiendas minoristas bajo su supervisión. Esto le permite identificar tendencias, patrones y desviaciones, así como tomar decisiones informadas para mejorar las ventas, optimizar el inventario y aumentar la satisfacción del cliente.

6. El jefe de local inicia sesión en el sistema y accede al mismo dashboard de gestión de retails
7. El dashboard proporcionaría al jefe de local una vista detallada y personalizada del rendimiento de su tienda en particular. Incluye información sobre ventas, margen de beneficio, cumplimiento de objetivos y otros indicadores clave relevantes para la gestión diaria.
8. El jefe de local utilizaría el dashboard para monitorear el rendimiento de su tienda, identificar áreas de mejora y tomar acciones correctivas. Por ejemplo, puede ajustar los horarios del personal, implementar estrategias de marketing específicas o gestionar el inventario en función de la demanda y las ventas.
9. El administrador de sector inicia sesión en el sistema y accede al dashboard de gestión de retails.
10. El dashboard proporcionaría al administrador de sector una vista consolidada y comparativa del rendimiento de múltiples tiendas y regiones. Permite al administrador analizar y comparar el rendimiento de diferentes áreas geográficas, identificar patrones y tendencias a nivel de sector, y tomar decisiones estratégicas para mejorar la eficiencia operativa y el crecimiento del negocio.
11. Una vez finalizadas las interacciones, los actores cierran sesión en el sistema.

Figura 3
Diagrama de caso de uso de un dashboard



Nota. El grafico explica el uso del dahsboard con los actores involucrados en el sistema. Elaborado en *Lucidchart*

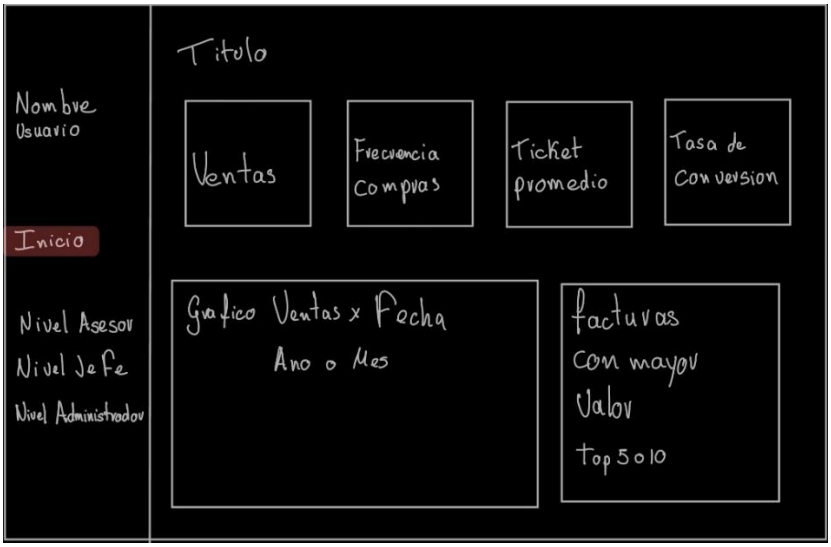
Como se muestra en la figura describe los diferentes actores, incluyendo el analista de datos, el asesor de ventas, el jefe de local y el administrador de sector, el dashboard ayudaría a visualizar información relevante y tomar decisiones basadas en datos en el contexto de la gestión de tiendas minoristas. El dashboard ofrece una amplia gama de gráficos, tablas y métricas que permiten a los usuarios monitorear y analizar el rendimiento de las tiendas en tiempo real, identificar oportunidades de mejora y descargar informes completos para un análisis más detallado. Los detalles específicos y las funcionalidades del dashboard pueden variar según las necesidades y requisitos de la organización.

Mockups

En el caso específico de un dashboard, el uso de un mockup se vuelve especialmente valioso. Permite a los diseñadores, desarrolladores y partes interesadas involucradas en el proyecto visualizar y evaluar de manera efectiva el diseño propuesto antes de embarcarse en la fase de desarrollo completo. Al crear un mockup de un dashboard, se tiene la capacidad de explorar y experimentar con diferentes diseños, esquemas de color, disposición de elementos e interacciones sin incurrir en gastos considerables ni dedicar tiempo a la programación detallada.

El mockup del dashboard sirve como una herramienta primordial para fomentar la colaboración y obtener retroalimentación por parte de los interesados, incluyendo miembros del equipo y clientes. Al proporcionar una representación visual clara de la estructura, el contenido y las interacciones previstas, facilita la comunicación y la alineación de todas las partes involucradas. Además, ayuda a identificar posibles mejoras, ajustes y requisitos adicionales antes de la fase de implementación, lo que resulta en un ahorro de tiempo y recursos.

Figura 4
Mockup de la pantalla de inicio

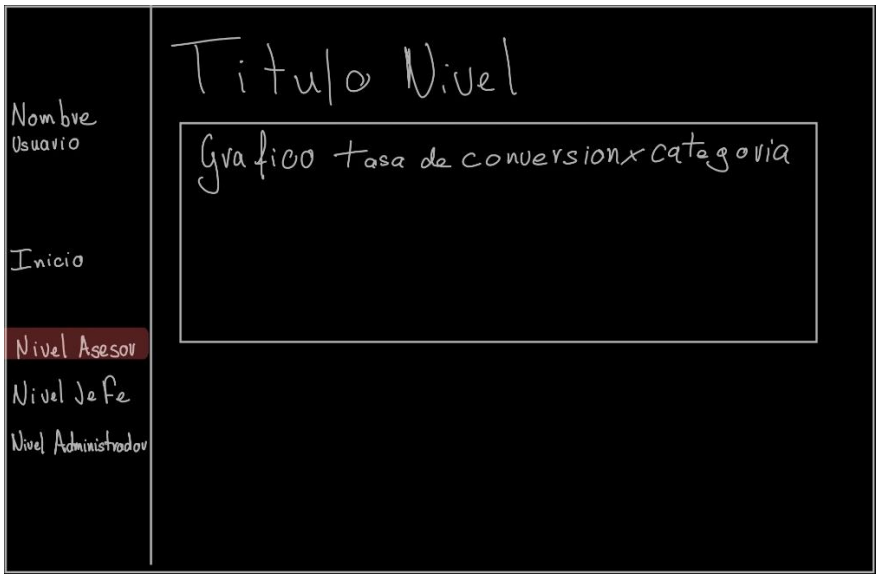


Nota. El grafico representa un borrador sobre la primera pantalla del dashboard.

El primer mockup representa la pantalla inicial del dashboard, diseñada de manera sencilla con el propósito de brindar a los usuarios una rápida visualización de la información. En el lado izquierdo, se encuentra una barra de navegación vertical que muestra el nombre del usuario y permite la navegación entre cuatro pantallas. La pantalla de inicio se despliega de forma predeterminada y muestra los KPIs más influyentes.

En la parte superior de la pantalla, se encuentra el título del dashboard. A continuación, de manera horizontal, se presentan cuadros en los que se exhiben los KPIs más importantes para el sector minorista. Estos indicadores se muestran junto con sus respectivas metas. Además, se incluye un gráfico que permite una visualización más detallada de las ventas por fecha, ya sea en formato anual o mensual. A la izquierda del gráfico, se presenta una tabla que detalla las facturas con mayor valor en orden descendente.

Figura 5
Mockup de la segunda pantalla del dashboard

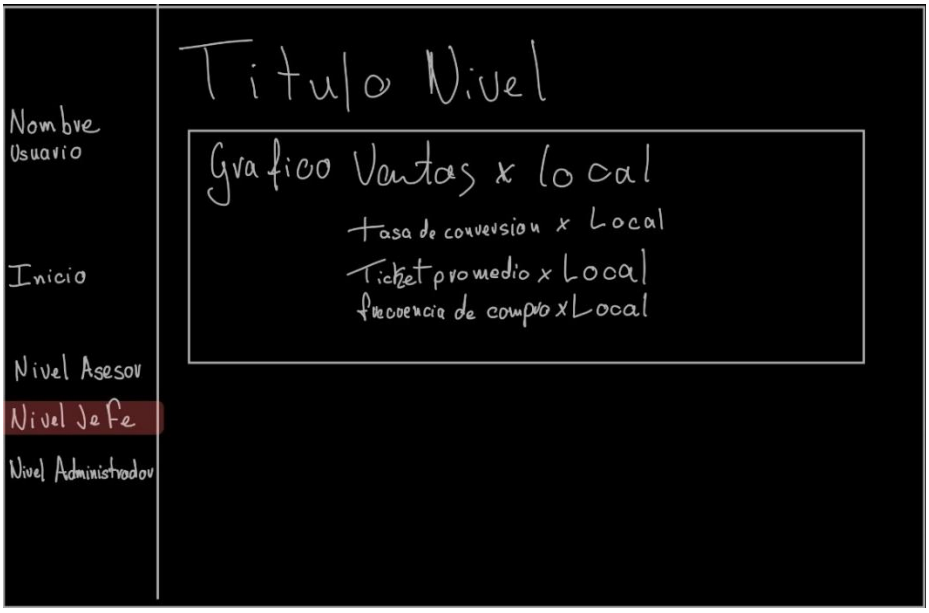


Nota. El grafico representa un borrador sobre la segunda pantalla del dashboard.

En el segundo dashboard, denominado "Nivel Asesor", se representa la tasa de conversión por categoría mediante un gráfico de barras. Esta visualización permite al asesor identificar si

existe una mayor cantidad de personas realizando compras en el retail, brindando información relevante para su desempeño.

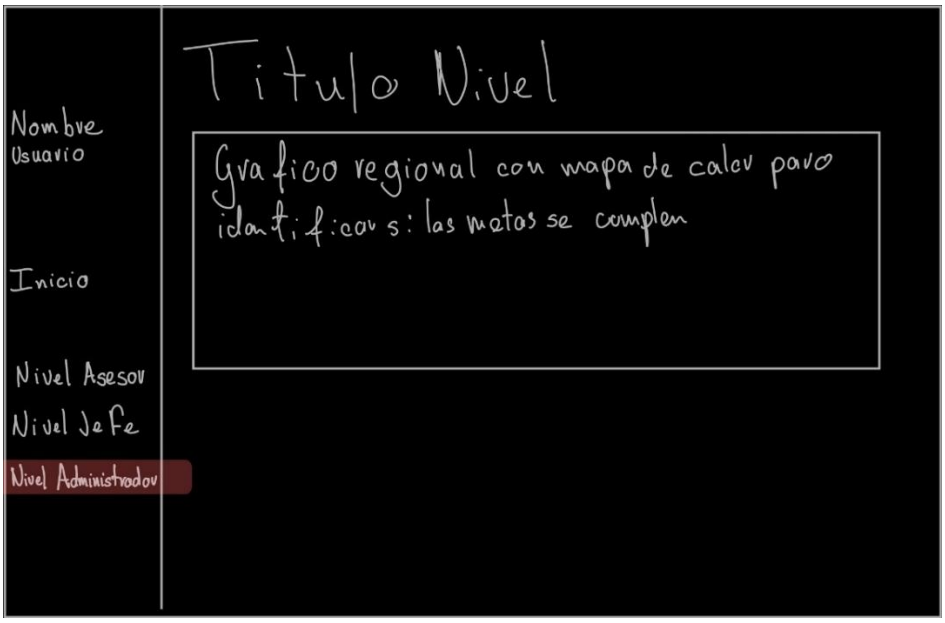
Figura 6
Mockup de la tercera pantalla del dashboard



Nota. El grafico representa un borrador sobre la tercera pantalla del dashboard.

El tercer dashboard, llamado "Nivel Jefe", muestra las ventas por local en un gráfico, lo que permite al Jefe de local tomar decisiones en tiempo real sobre el rendimiento de los diferentes locales. Esto facilita el enfoque en los planes de gestión y proporciona un seguimiento más personalizado.

Figura 7
Mackup de la cuarta pantalla del dashboard



Nota. El grafico representa el borrador de la cuarta pantalla del dashboard.

El último dashboard, denominado "Nivel Administrador", presenta un mapa geográfico que muestra las diferentes ubicaciones de los locales comerciales. Cada ubicación se señala con una barra de color que indica su proximidad a alcanzar la meta establecida. Esta visualización permite al administrador analizar los locales que requieren mayor apoyo para lograr sus objetivos.

Es importante destacar que estos mockups proporcionan una representación estática del diseño final del dashboard. Permiten visualizar la disposición de la información y las interacciones propuestas de manera rápida y sencilla ayudando con las ideas de diseño. Estos mockups se utilizan como una herramienta esencial para obtener retroalimentación y colaboración de los interesados antes de avanzar hacia la etapa de implementación completa.

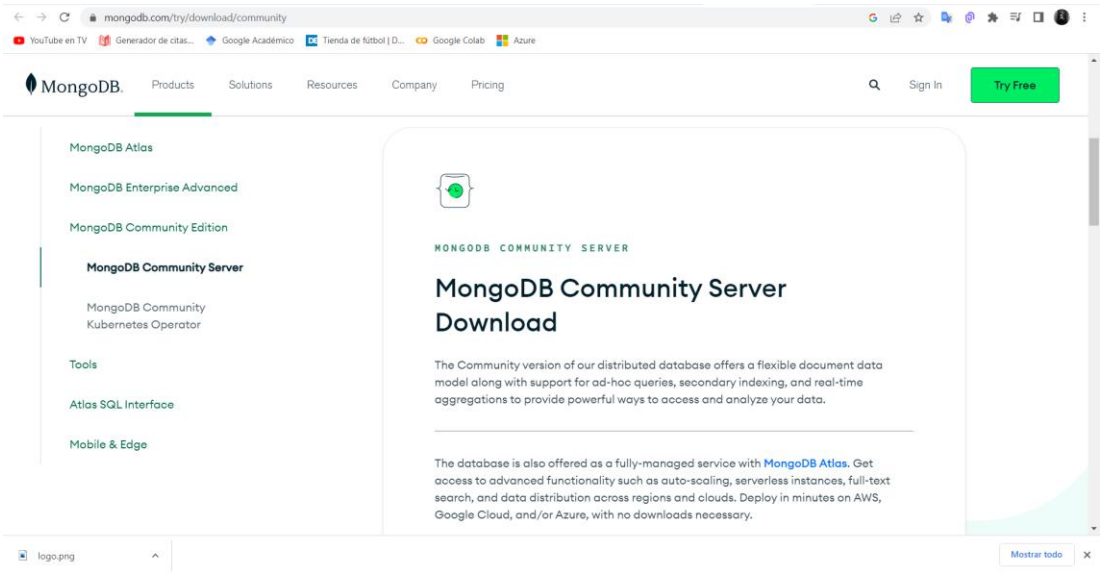
4.8. Backend

En el desarrolló del backend, se decidió utilizar MongoDB como base de datos para almacenar los datos después del preprocesamiento. Se optó por configurar un servidor local de MongoDB para facilitar la carga de datos y el consumo a través del API.

El primer paso para instalar MongoDB es descargar el paquete de instalación correspondiente desde el sitio web oficial de MongoDB, en este caso se utilizará MongoDB Community Server.

En el navegador de preferencia, se busca ‘MongoDB Community Server Download’, ingresamos a la página oficial de MondoDB

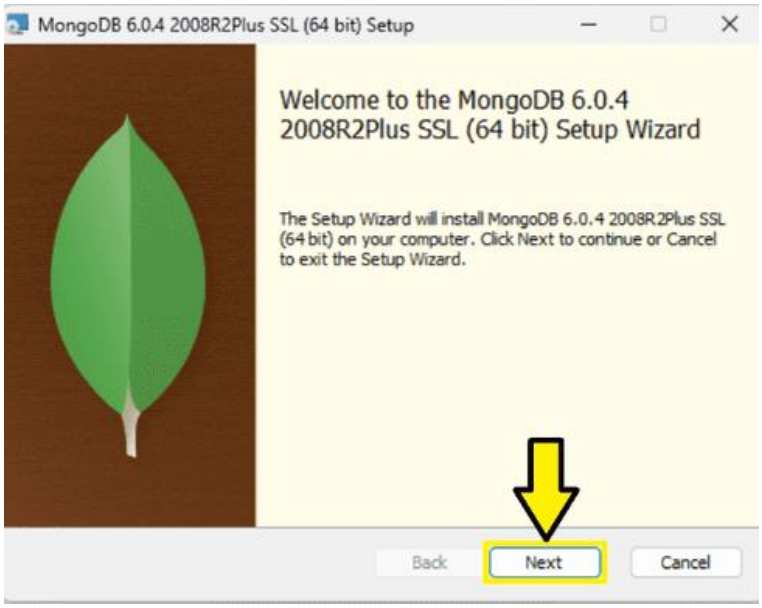
Figura 8
Pantalla de descarga del gestor de base de datos



Nota. La imagen representa la pagina donde se puede descargar MongoDB. Extraída de *MongoDB*

Se debe seleccionar la versión adecuada para el sistema operativo en uso. Una vez descargado el paquete de instalación, se ejecuta el archivo para iniciar el proceso de instalación.

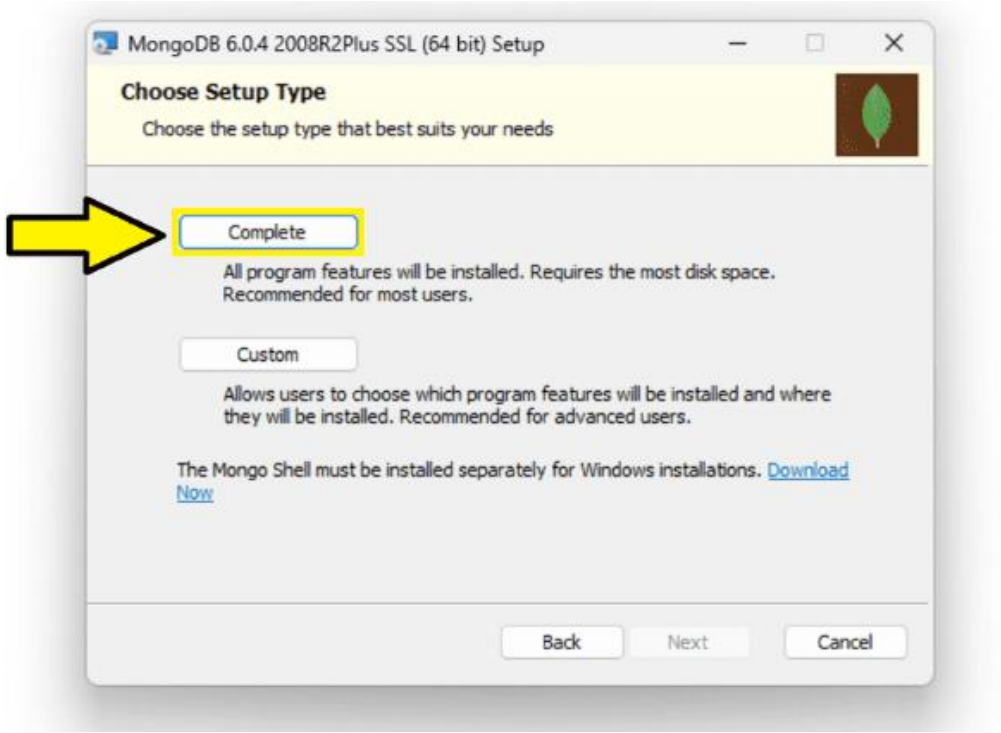
Figura 9
Ventana de instalación de la versión 6.0.4 de MongoDB



Nota. El grafico indica la versión de la instalación de MongoDB. Extraída de *MongoDB*

El asistente de instalación guiará al usuario a través de los pasos necesarios para completar la instalación de MongoDB en el sistema. Durante el proceso de instalación, se solicitará aceptar los términos de licencia. Es importante leer los términos de la licencia cuidadosamente y, si se está de acuerdo con ellos, marcar la casilla correspondiente para aceptarlos.

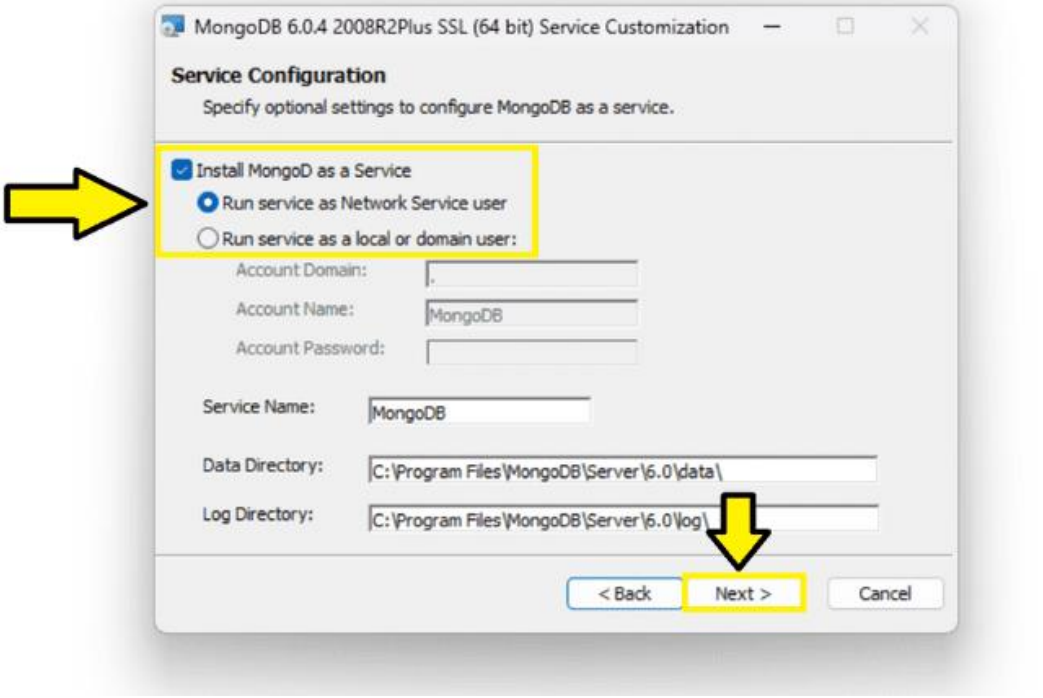
Figura 10
Ventana donde se exige el tipo de instalación



Nota. El grafico indica el tipo de instalación que debe escoger para la instalación. Extraída de *MongoDB*

A continuación, se deberá seleccionar la ruta de instalación para MongoDB. Por defecto, se seleccionará una ruta adecuada en función del sistema operativo, pero el usuario puede optar por cambiarla si lo desea. Luego, se ofrecerá la opción de configurar MongoDB como un servicio en el sistema. Esta opción permite que MongoDB se inicie automáticamente al encender el sistema operativo. Si se desea esta funcionalidad, se deberá marcar la casilla correspondiente durante la instalación.

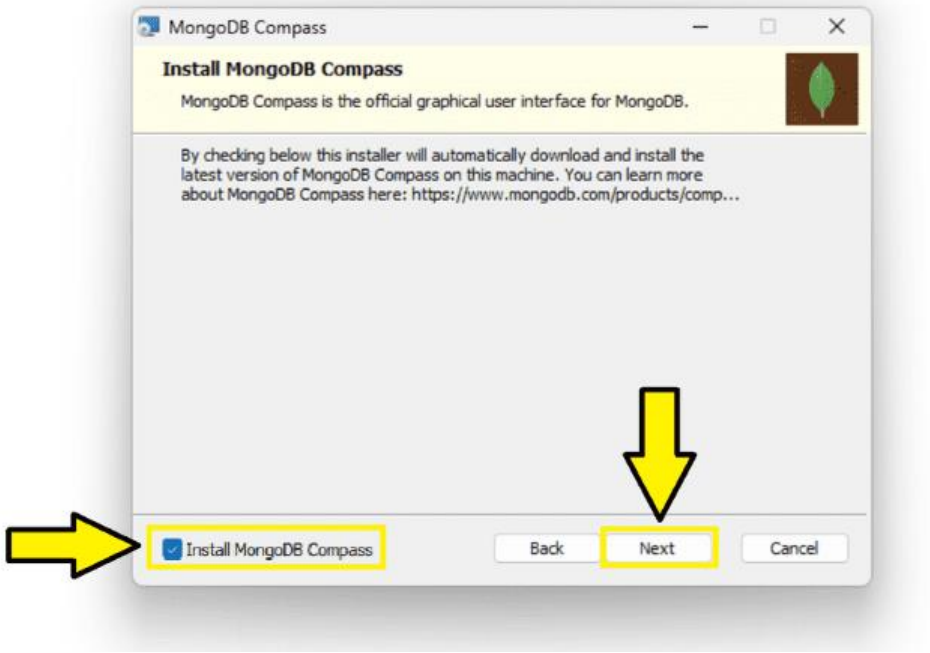
Figura 11
Ventana de selección del servicio del usuario



Nota. El grafico indica la opción para la configuración del servicio del usuario. Extraída de *MongoDB*

Durante el proceso de instalación, también se pedirá configurar el acceso a MongoDB. El usuario podrá elegir si desea habilitar el acceso a través de la red o solo de forma local. En caso de requerir acceso desde otras máquinas, se deberá seleccionar la opción adecuada y especificar las direcciones IP permitidas.

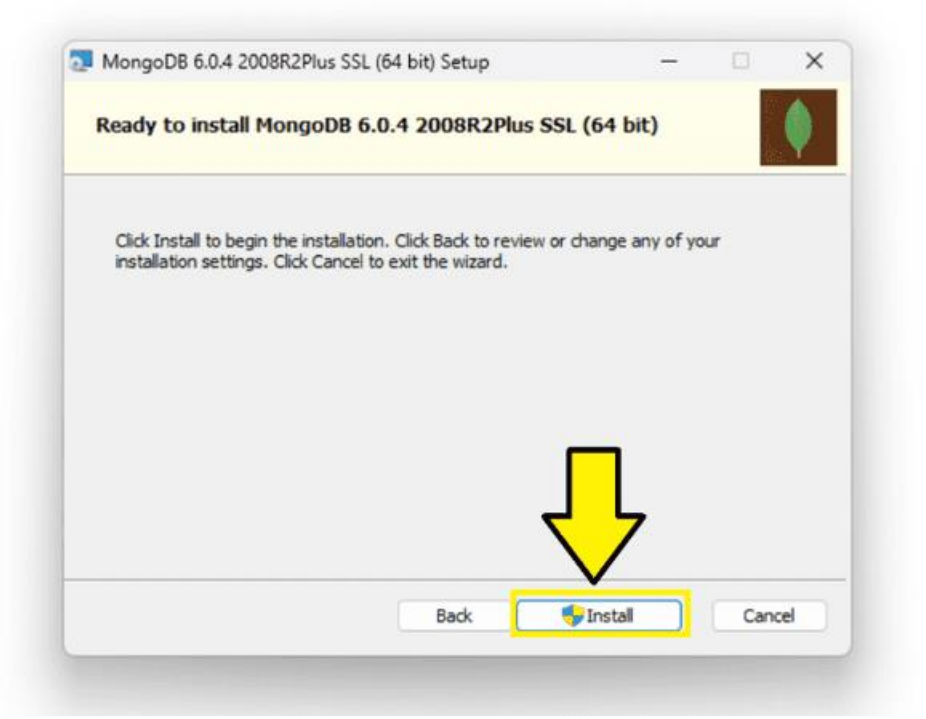
Figura 12
Ventana para instalar MongoDB Compass



Nota. EL grafico representa la opción que se selecciona para la instalación de MongoDB Compass. Extraída de *MongoDB*

Además, se podrá seleccionar características adicionales durante la instalación, como las herramientas de línea de comandos y las herramientas de compatibilidad con BI (Business Intelligence). El usuario elegirá las características que desee instalar según sus necesidades.

Figura 13
Venta de final de instalación



Nota. El grafico indica que esta listo para ser instalado el gestor de base de datos MongoDB. Extraída de *MongoDB*

Una vez revisadas todas las opciones de configuración, se mostrará un resumen de la instalación. El usuario deberá revisar la configuración y, si está satisfecho, podrá iniciar el proceso de instalación haciendo clic en el botón "Instalar". Al completar la instalación, se mostrará una ventana indicando que MongoDB se ha instalado correctamente. Si se desea utilizar MongoDB Compass, una herramienta gráfica de administración de MongoDB, se deberá marcar la casilla "Iniciar MongoDB Compass". Tendremos MongoDB instalado en el sistema y podrá acceder a él a través de la línea de comandos utilizando el comando mongo o utilizando herramientas gráficas como MongoDB Compass.

Carga de datos a la BDD

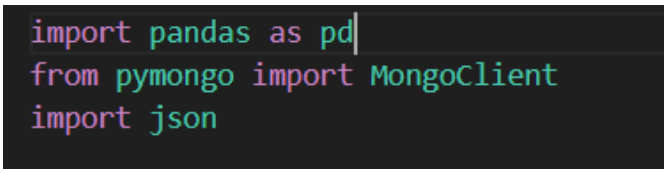
El formato CSV (Valores Separados por Comas) fue elegido debido a su amplia adopción y compatibilidad con diversas herramientas y lenguajes de programación. Al representar los datos

en forma tabular, donde las columnas están separadas por comas y cada fila corresponde a un registro, se facilita la lectura y comprensión de los datos, así como su intercambio entre sistemas y plataformas.

La carga de los datos desde el archivo CSV a un objeto DataFrame se realizó utilizando la biblioteca pandas de Python. Pandas ofrece una amplia gama de funcionalidades para la manipulación y análisis eficiente de datos tabulares. Proporciona métodos y funciones que permiten realizar operaciones como filtrado, agregación, transformación y visualización de datos de manera flexible y sencilla.

Además, la transformación de las columnas del DataFrame asegura que los datos tengan el tipo de dato adecuado para su posterior manipulación. Esto es especialmente relevante en casos como la conversión de fechas a un formato estándar, lo cual posibilita realizar operaciones temporales y análisis de series temporales de manera más precisa.

Figura 14
Bibliotecas para la carga del dataset a la Base de datos



Nota. La imagen representa las bibliotecas de Python que se utilizaran para el proceso de carga del dataset a la base de datos.

En primer lugar, se importan las librerías necesarias, como pandas, numpy, pymongo, TransactionEncoder de mlxtend y json, que serán utilizadas a lo largo del código.

Figura 15
Ruta del archivo csv

Nota: El grafico presenta la ruta que se extraerá el archivo csv para su transformación a csv

Después, se carga el conjunto de datos desde el archivo CSV utilizando la función `pd.read_csv` de la biblioteca pandas. Este conjunto de datos se almacena en un objeto DataFrame denominado df. Se imprimen los tipos de datos de las columnas y se muestran las primeras filas del DataFrame para una inspección inicial.

Figura 16
Conversión de los tipos de datos

```
# Convertir las columnas relevantes al tipo de datos adecuado
df['quantity'] = df['quantity'].astype(str)
df['quantity'] = df['quantity'].astype(str)
df['price'] = df['price'].astype(str)
df['age'] = df['age'].astype(str)
df['gender'] = df['gender'].astype(str)
df['invoice_date'] = df['invoice_date'].astype(str)
df['shopping_mall'] = df['shopping_mall'].astype(str)
df['payment_method'] = df['payment_method'].astype(str)
df['anio'] = pd.to_datetime(df['invoice_date']).dt.year
df['mes'] = pd.to_datetime(df['invoice_date']).dt.month
df['dia'] = pd.to_datetime(df['invoice_date']).dt.day
df['anio'] = df['anio'].astype(str)
df['mes'] = df['mes'].astype(str)
df['dia'] = df['dia'].astype(str)
```

Nota: El grafico presenta los diferentes datos convertidos a tipo objeto.

Se realiza la transformación de las columnas relevantes del DataFrame para asegurar que tengan el tipo de datos adecuado. Para ello, se utiliza el método `astype` de pandas para convertir las columnas a tipos de datos específicos. Además, se utiliza `pd.to_datetime` para convertir las fechas a un formato adecuado. Se imprimen nuevamente los tipos de datos actualizados del DataFrame para verificar los cambios realizados.

Figura 17
Columnas relevantes

```
# Seleccionar las columnas relevantes
df = df[['invoice_no', 'customer_id', 'category',
         'quantity', 'price', 'age', 'gender',
         'invoice_date', 'shopping_mall', 'payment_method',
         'anio', 'mes', 'dia']]
```

Nota: El grafico presenta los datos que se seleccionan para la creación del dataframe

Se seleccionan las columnas relevantes del DataFrame que serán utilizadas en etapas posteriores del análisis. Establece una conexión con una base de datos MongoDB local mediante el uso de MongoClient de la biblioteca pymongo. Se define la base de datos y la colección en las que se almacenarán los documentos.

Figura 18
Conexión a la base de datos para carga de datos

```
# Conexión a la base de datos MongoDB
client = MongoClient('mongodb://localhost:27017')
db = client['dashboard']

# Nombre de la colección en MongoDB
collection = db['ventas']
```

Nota: El grafico presenta la conexión la conexión a la base de datos con el nombre y el nombre de la colección.

El DataFrame se convierte a formato JSON utilizando el método to_json de pandas, y los documentos resultantes se insertan en la colección de MongoDB utilizando insert_many.

Figura 19
Conversión de dataframe a json

```
# Convertir el DataFrame a formato JSON
df_json = df.to_json(orient='records')
documents = json.loads(df_json)

# Insertar los documentos en la colección
collection.insert_many(documents)
```

Nota: El grafico presenta la transformación de dataframe a json, para insertar como documentos a la colección de la base de dato.

ETL

El proceso de ETL se llevó a cabo con el propósito de facilitar la carga de datos al dashboard y permitir el análisis de los diferentes indicadores clave de rendimiento (KPIs). Se extrajeron los datos de la base de datos de origen, en este caso, MongoDB, mediante la conexión establecida. Se realizaron transformaciones en los datos para su adecuación al formato requerido en el dashboard. Esto implicó la conversión de tipos de datos, cálculos de métricas, agrupamientos y otras operaciones necesarias para obtener los KPIs deseados. Finalmente, los datos transformados se cargaron en el sistema del dashboard, donde estarán disponibles para su visualización, análisis y generación de informes. A continuación, se detalla el proceso de desarrolló.

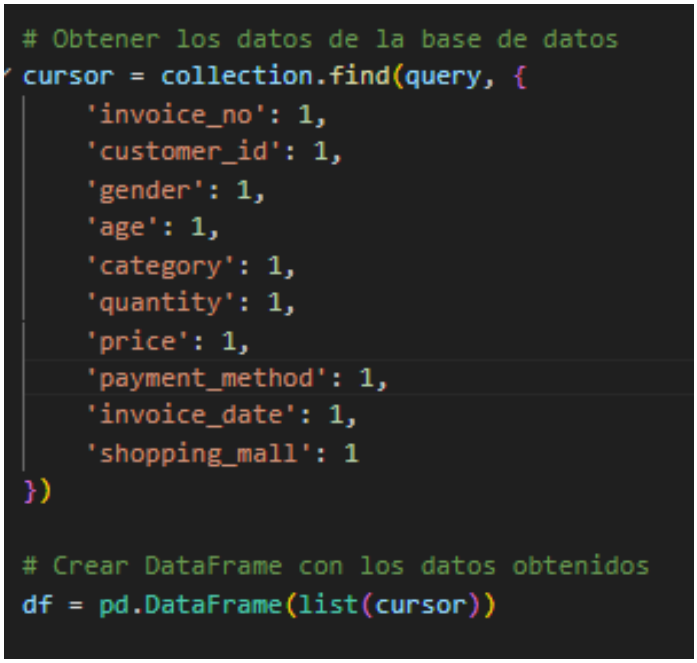
Figura 20
Conexión a la base de datos para ETL

```
# Conexión a la base de datos MongoDB
client = MongoClient('mongodb://localhost:27017')
db = client['dashboard']
collection = db['ventas']
```

Nota: EL grafico presenta la conexión a la base de datos, con el nombre y la colección que se va a utilizar.

En primer lugar, se establece la conexión con la base de datos MongoDB, especificando la dirección del servidor y el puerto correspondiente. A continuación, se selecciona la base de datos "dashboard" y se identifica la colección "ventas" que contiene los datos relevantes.

Figura 21
Estructura de datos para la obtencion de datos



Nota: El grafico presenta los datos que deseamos extraer de la base de datos, y se crea un dataframe

Luego, se realiza la consulta a la base de datos para obtener los datos necesarios. Se utiliza un cursor para iterar sobre los documentos obtenidos y se almacenan en un DataFrame de Pandas.

Figura 22
Proceso de tratamiento de los datos

```
# Convertir la columna de precio a tipo numérico
df['price'] = pd.to_numeric(df['price'], errors='coerce').fillna(0)

# Obtener la tasa de cambio entre TL y USD (ejemplo: 1 TL = 0.12 USD)
tasa_cambio = 0.050

# Convertir el precio de TL a USD
df['price_usd'] = df['price'] * tasa_cambio

# Convertir la columna de fecha a tipo datetime
df['invoice_date'] = pd.to_datetime(df['invoice_date'])

# Calcular los KPIs
df['tasa_conversio'] = (df.groupby('customer_id')
                        ['invoice_no'].transform('nunique') / df['customer_id'].nunique()) * 100
df['ventas_totales'] = df['price_usd'].sum()
df['ticket_promedio'] = df['price_usd'].sum() / df['invoice_no'].nunique()
df['frecuencia'] = df['invoice_no'].nunique() / df['customer_id'].nunique()
```

Nota: El grafico presenta el proceso de tratamiento de los datos y la creación de los KPIs.

Se realizan cálculos para obtener indicadores clave de rendimiento. Se convierte la columna de precios a tipo numérico y se rellenan los valores faltantes con ceros. Se define la tasa de cambio entre la moneda local y el dólar estadounidense. Se calcula el precio en dólares multiplicando el precio original por la tasa de cambio. Además, se convierte la columna de fechas a tipo datetime y se calculan diversos indicadores como la tasa de conversión promedio, las ventas totales, el ticket promedio y la frecuencia de compra.

Figura 23
Código de los dataframes por datos

```
# Crear DataFrame por categoría
df_categoria = df.groupby(['category', pd.Grouper(key='invoice_date', freq='M')]).agg({
    'tasa_conversio': 'mean',
    'ticket_promedio': 'mean',
    'ventas_totales': 'sum',
    'frecuencia': 'sum'
}).reset_index()

# Crear DataFrame por método de pago
df_metodo_pago = df.groupby(['payment_method', pd.Grouper(key='invoice_date', freq='M')]).agg({
    'tasa_conversio': 'mean',
    'ticket_promedio': 'mean',
    'ventas_totales': 'sum',
    'frecuencia': 'sum'
}).reset_index()

# Crear DataFrame por locales
df_locales = df.groupby(['shopping_mall', pd.Grouper(key='invoice_date', freq='M')]).agg({
    'tasa_conversio': 'mean',
    'ticket_promedio': 'mean',
    'ventas_totales': 'sum',
    'frecuencia': 'sum'
}).reset_index()

df_locales_sin_fecha = df.groupby('shopping_mall').agg({
    'tasa_conversio': 'mean',
    'ticket_promedio': 'mean',
    'ventas_totales': 'sum',
    'frecuencia': 'mean',

}).reset_index()
```

Nota: El grafico presenta dataframes por diferentes datos dentro de los KPIs

Posteriormente, se crean DataFrames adicionales mediante el agrupamiento de datos por categoría, método de pago y locales. Se utilizan funciones de agregación para calcular métricas específicas dentro de cada grupo. Esto permite tener información detallada sobre el rendimiento en diferentes categorías, métodos de pago y locales a lo largo del tiempo.

Figura 24
Transformación y carga de los dataframes

```
# Convertir los DataFrames a formato JSON
json_categoria = df_categoria.to_dict(orient='records')
json_metodo_pago = df_metodo_pago.to_dict(orient='records')
json_locales = df_locales.to_dict(orient='records')
json_locales_sin_fecha = df_locales_sin_fecha.to_json(orient='records')
json_metodo_pago_sin_fecha = df_metodo_pago_sin_fecha.to_json(orient='records')
json_categoria_sin_fecha = df_categoria_sin_fecha.to_json(orient='records')

# Insertar los documentos en colecciones nuevas
db['categoria'].insert_many(json_categoria)
db['metodo_pago'].insert_many(json_metodo_pago)
db['locales'].insert_many(json_locales)
db['locales_sin_fecha'].insert_many(json.loads(json_locales_sin_fecha))
db['metodo_pago_sin_fecha'].insert_many(json.loads(json_metodo_pago_sin_fecha))
db['categoria_sin_fecha'].insert_many(json.loads(json_categoria_sin_fecha))
```

Nota: El grafico presenta la conversión de dataframes a json para la carga dentro de la base de datos

Finalmente, se convierten los DataFrames en formato JSON y se cargan en colecciones separadas de la base de datos MongoDB utilizando el método `insert_many()`. Esto facilita el almacenamiento y acceso a los informes generados, que pueden ser utilizados posteriormente para el análisis y visualización de los datos de ventas.

Este proceso proporciona una solución para la extracción, procesamiento y generación de informes a partir de datos de ventas almacenados en una base de datos MongoDB. Proporciona indicadores clave de rendimiento y permite analizar el rendimiento en diferentes categorías, métodos de pago y locales. Además, facilita la visualización de los datos mediante la conversión a formato JSON y su almacenamiento en la base de datos.

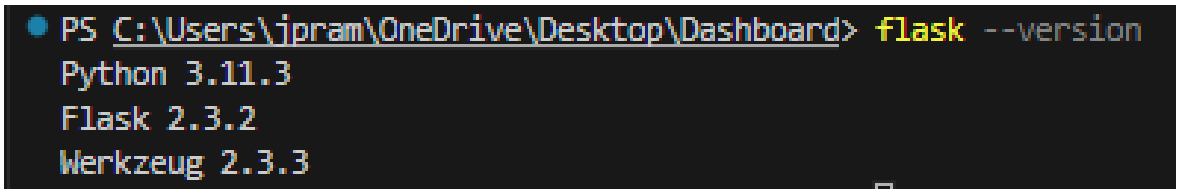
4.8.1. API

La API de Carga Automática de Datos es una interfaz de programación de aplicaciones desarrollada en Flask y conectada con MongoDB. Su propósito principal es automatizar el proceso de carga de datos. Permite a los usuarios enviar datos al sistema de manera eficiente y confiable. Esta API ha sido diseñada para simplificar y agilizar el proceso de carga de datos, brindando una solución eficiente a los usuarios que necesiten automatizar este flujo de trabajo.

Requisitos y dependencias:

Para utilizar la API de Carga Automática de Datos, se requiere tener instalado Python 3.7 o una versión superior, junto con los siguientes paquetes y dependencias: Flask 2.0.1 y Flask-PyMongo 3.0.1. Además, se necesita tener MongoDB 4.4.2 configurado y en funcionamiento en el sistema. Es importante asegurarse de contar con todas estas dependencias correctamente instaladas antes de utilizar la API.

Figura 25
Consola de verificación de versiones



Nota. El grafico representa las diferentes versiones de Python y Flask, para la validación de la instalación.

Descripción de los endpoints:

Figura 26
Endpoint creación de usuario

```python
@app.route('/users',methods=['POST'])
def create_user():
    # Receiving Data
    firstName = request.json['firstName']
    lastName = request.json['lastName']
    username = request.json['username']
    email = request.json['email']
    password = request.json['password']

    if username and email and password:
        hashed_password = generate_password_hash(password)
        #geenera la coleccion
        id = mongo.db.users.insert_one({
            'firstName': firstName,
            'lastName': lastName,
            'username': username,
            'email': email,
            'password': hashed_password}
        )
        response = jsonify({
            '_id': str(id),
            'firstName': firstName,
            'lastName': lastName,
            'username': username,
            'password': hashed_password,
            'email': email
        })
        #response.status_code = 201
        return response
    else:
        return not_found()
```

Nota: El grafico presenta la estructura para la creación del usuario.

La función create_user() parece ser un controlador para la creación de usuarios en una aplicación web. Aquí está la explicación del código:

La función recibe los datos del usuario (nombre, apellido, nombre de usuario, correo electrónico y contraseña) a través de una solicitud JSON.Luego, verifica si se han proporcionado el nombre de usuario, el correo electrónico y la contraseña. Si esos campos no están vacíos, se procede a crear un nuevo usuario.

La función generate_password_hash(password) se utiliza para generar el hash de la contraseña proporcionada. El hash resultante se almacena en la variable hashed_password.

A continuación, se inserta un nuevo documento en la colección "users" de la base de datos. Los datos del usuario, como el nombre, apellido, nombre de usuario, correo electrónico y contraseña hasheada, se pasan como un diccionario al método insert_one() de la biblioteca mongo. El resultado de la operación de inserción se almacena en la variable id, que contiene el ID del nuevo documento creado. Despues, se construye una respuesta JSON que contiene los datos del usuario recién creado, incluyendo el ID, nombre, apellido, nombre de usuario, correo electrónico y contraseña (hasheada). La respuesta se almacena en la variable response.

Finalmente, se devuelve la respuesta JSON como resultado de la función. Si alguno de los campos requeridos (nombre de usuario, correo electrónico o contraseña) está ausente, se llama a la función not_found() (que no se muestra en el código proporcionado) para manejar el caso en el que no se encuentren los campos necesarios.

Figura 27
Función encriptación contraseña

```
def generate_password_hash(password):
    salt = gensalt()
    hashed_password = hashpw(password.encode('utf-8'), salt)
    return hashed_password.decode('utf-8')
```

Nota: El grafico presenta la lógica de encriptamiento

La función generate_password_hash(password) toma una contraseña como entrada, genera una sal aleatoria utilizando la función gensalt(), y luego se hashea la contraseña junto con la sal utilizando la función hashpw() de la biblioteca bcrypt. Finalmente, se devuelve la contraseña hasheada como una cadena de texto.

Figura 28
Obtención de password encriptado

```python
def get_password_hash(username):
    user = mongo.db.users.find_one({'username': username})
    if user:
        return user['password']
    return None
```

Nota: El grafico presenta la obtención del password por usuario creado.

La función get_password_hash(username) recibe un nombre de usuario como entrada y realiza una consulta a la base de datos para obtener el documento de usuario correspondiente al nombre de usuario proporcionado. Si se encuentra el usuario, la función devuelve el valor de la clave 'password' del documento. De lo contrario, devuelve None.

El controlador de la ruta '/login' es la función login(). Esta función maneja las solicitudes POST para iniciar sesión. Recibe los datos del usuario (nombre de usuario y contraseña) desde la solicitud y verifica si el nombre de usuario existe en la base de datos utilizando la función get_password_hash(). Si el nombre de usuario existe y la contraseña coincide con la contraseña almacenada en la base de datos (utilizando la función checkpw() de bcrypt), se devuelve una respuesta JSON con un estado de éxito. De lo contrario, se devuelve una respuesta JSON con un estado de error.

Figura 29
Obtención de credenciales de usuario

```python
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')

    stored_password_hash = get_password_hash(username)

    if stored_password_hash and checkpw(password.encode('utf-8'), stored_password_hash.encode('utf-8')):
        # Ingreso correcto
        return jsonify({'status': 'success'})
    else:
        # Ingreso incorrecto
        return jsonify({'status': 'error'})
```

Nota: El grafico presenta la obtención de credenciales de usuario creadas, para la autenticación de la contraseña por usuario

La función check_user_exists(username) recibe un nombre de usuario como entrada y realiza una consulta a la base de datos para verificar si existe un documento de usuario con ese nombre de usuario. Devuelve un valor booleano indicando si el usuario existe o no.

Figura 30
Verificación de existencia de username

```
# Endpoint para verificar si un usuario ya existe
def check_user_exists(username):
    user = mongo.db.users.find_one({'username': username})
    return user is not None
```

Nota: El grafico presenta la verificación del username, para la creación de uno nuevo.

La función check_password_exists(password) recibe una contraseña como entrada y realiza una consulta a la base de datos para verificar si existe un documento de usuario con esa contraseña. Devuelve el valor de la clave 'password' del documento si se encuentra un usuario con esa contraseña, de lo contrario, devuelve `None`.

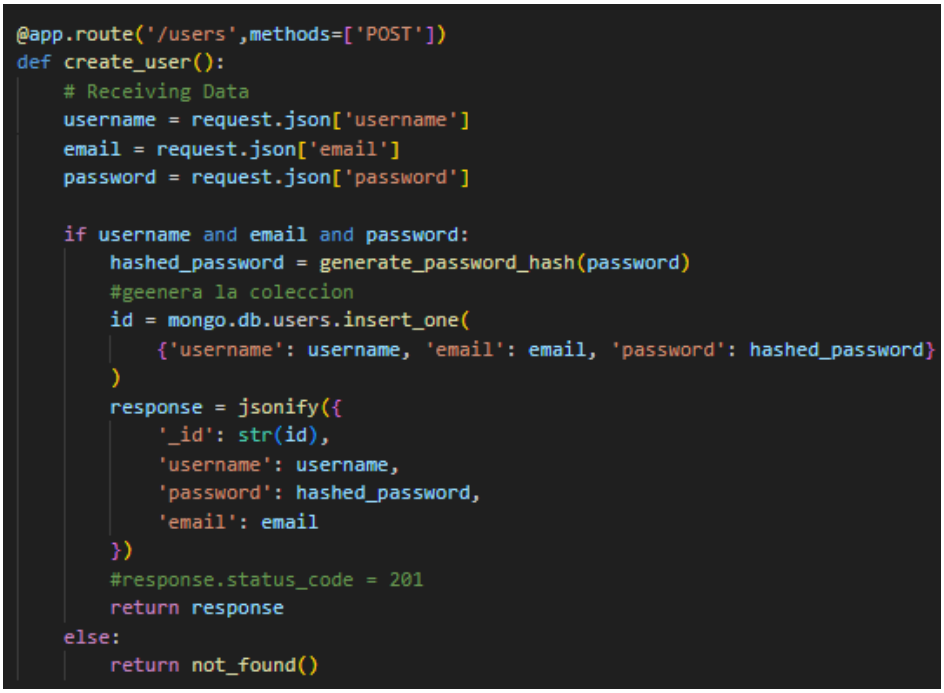
Figura 31
Verificación de existencia de email

```
def check_email_exists(email):
    user = mongo.db.users.find_one({'email': email})
    return user is not None
```

Nota: El grafico presenta la verificación del email, para la creación de uno nuevo

La función check_email_exists(email) recibe un correo electrónico como entrada y realiza una consulta a la base de datos para verificar si existe un documento de usuario con ese correo electrónico. Devuelve un valor booleano indicando si el correo electrónico existe o no.

Figura 32
Obtención de los usuarios creado



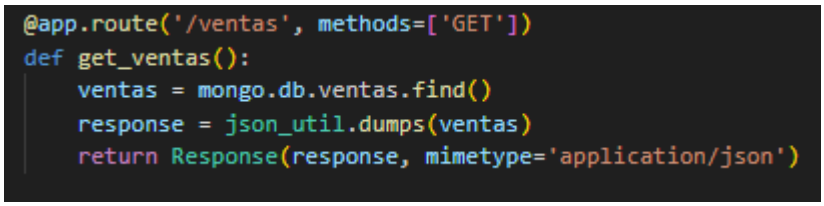
```python
@app.route('/users',methods=['POST'])
def create_user():
    # Receiving Data
    username = request.json['username']
    email = request.json['email']
    password = request.json['password']

    if username and email and password:
        hashed_password = generate_password_hash(password)
        #geenera la coleccion
        id = mongo.db.users.insert_one(
            {'username': username, 'email': email, 'password': hashed_password}
        )
        response = jsonify({
            '_id': str(id),
            'username': username,
            'password': hashed_password,
            'email': email
        })
        #response.status_code = 201
        return response
    else:
        return not_found()
```

Nota: El grafico presenta la obtención de los usuarios con el _id, username, email y password de la colección

GET /users: Este endpoint permite obtener todos los usuarios almacenados en la base de datos. Los usuarios son obtenidos desde la colección "users" en MongoDB. La respuesta se devuelve en formato JSON, proporcionando información como el nombre de usuario, correo electrónico y contraseña de cada usuario.

Figura 33
Obtención de ventas



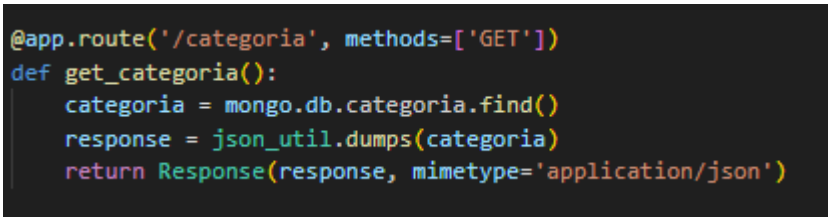
```python
@app.route('/ventas', methods=['GET'])
def get_ventas():
    ventas = mongo.db.ventas.find()
    response = json_util.dumps(ventas)
    return Response(response, mimetype='application/json')
```

Nota: El grafico presenta la estructura con un método GET de la obtención de los datos de ventas

GET /ventas: Aquí, se puede obtener un listado de todas las ventas registradas en la base de datos. Estos datos se recuperan de la colección "ventas" en MongoDB. La respuesta se presenta

en formato JSON, incluyendo información relevante sobre cada venta, como el monto, la fecha y los productos vendidos.

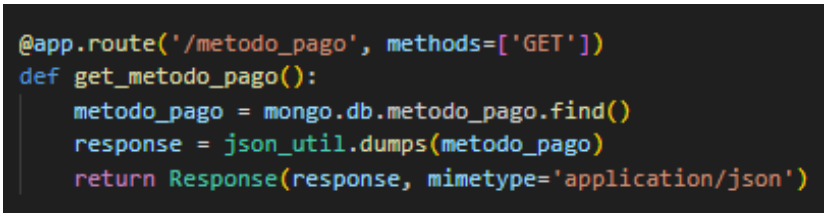
Figura 34
Obtención de categoría



Nota: El grafico presenta la estructura con un método GET de la obtención de los datos de categoría

GET /categoria: Este endpoint permite obtener todas las categorías almacenadas en la base de datos. Las categorías se obtienen de la colección "categoria" en MongoDB. La respuesta se presenta en formato JSON, proporcionando información como el nombre de la categoría y su descripción.

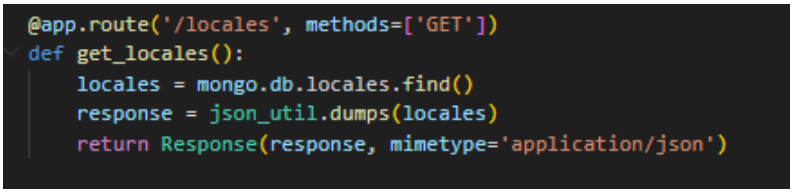
Figura 35
Obtención de método de pago



Nota: El grafico presenta la estructura con un método GET de la obtención de los datos de método de pagos

GET /metodo_pago: Con este endpoint, se puede acceder a todos los métodos de pago registrados en la base de datos. Estos métodos de pago se obtienen de la colección "metodo_pago" en MongoDB. La respuesta se devuelve en formato JSON e incluye detalles sobre cada método de pago, como el nombre y la descripción.

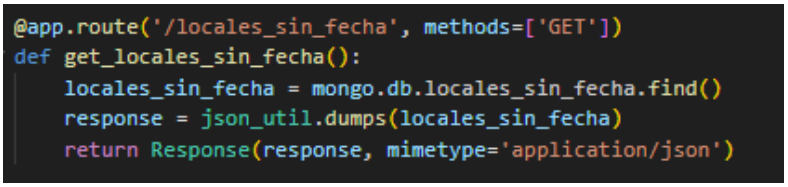
Figura 36
Obtención de locales



Nota: El grafico presenta la estructura con un método GET de la obtención de los datos de locales

GET /locales: Este endpoint permite obtener un listado de todos los locales registrados en la base de datos. Los locales son obtenidos de la colección "locales" en MongoDB. La respuesta se devuelve en formato JSON, proporcionando información como el nombre del local, su dirección y detalles de contacto.

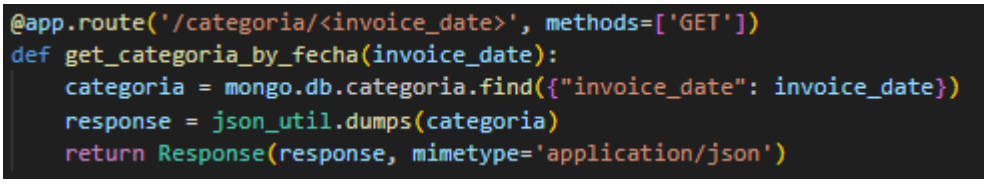
Figura 37
Obtención de locales sin fecha



Nota: El grafico presenta la estructura con un método GET de la obtención de los datos de locales sin fecha

GET /locales_sin_fecha: Aquí, se pueden obtener todos los locales que no tienen una fecha asociada en la base de datos. Estos datos se recuperan de la colección "locales_sin_fecha" en MongoDB. La respuesta se presenta en formato JSON, proporcionando detalles sobre cada local sin fecha, como el nombre y la dirección.

Figura 38
Obtención de categoría según fecha



Nota: El grafico presenta la estructura con un método GET de la obtención de los datos de categoría con fecha

GET /categoria/<invoice_date>: Mediante este endpoint, es posible obtener las categorías según la fecha de la factura (invoice_date) proporcionada. Los datos se obtienen de la colección "categoria" en MongoDB y se filtran de acuerdo con la fecha de la factura especificada. La respuesta se devuelve en formato JSON, incluyendo información detallada sobre cada categoría correspondiente a la fecha de la factura.

Figura 39
Obtención de método de pago por fecha

```python
@app.route('/metodo_pago/<invoice_date>', methods=['GET'])
def get_metodo_pago_by_fecha(invoice_date):
    metodo_pago = mongo.db.metodo_pago.find({"invoice_date": invoice_date})
    response = json_util.dumps(metodo_pago)
    return Response(response, mimetype='application/json')
```

Nota: El grafico presenta la estructura con un método GET de la obtención de los datos de método de pago con fecha

GET /metodo_pago/<invoice_date>: Con este endpoint, se pueden obtener los métodos de pago según la fecha de la factura (invoice_date) proporcionada. Los datos se obtienen de la colección "metodo_pago" en MongoDB y se filtran según la fecha de la factura especificada. La respuesta se presenta en formato JSON e incluye detalles sobre cada método de pago asociado a la fecha de la factura.

Figura 40
Obtención de locales por fecha

```python
@app.route('/locales/<invoice_date>', methods=['GET'])
def get_locales_by_fecha(invoice_date):
    locales = mongo.db.locales.find({"invoice_date": invoice_date})
    response = json_util.dumps(locales)
    return Response(response, mimetype='application/json')
```

Nota: El grafico presenta la estructura con un método GET de la obtención de los datos de locales con fecha

GET /locales/<invoice_date>: Este endpoint permite obtener los locales según la fecha de la factura (invoice_date) proporcionada. Los datos se obtienen de la colección "locales" en MongoDB y se filtran según la fecha de la factura especificada. La respuesta se devuelve en formato JSON, proporcionando información sobre cada local asociado a la fecha de la factura.

Figura 41
Obtención de errores

```python
@app.errorhandler(404)
def not_found(error=None):
    message = {
        'message': 'Resource Not Found ' + request.url,
        'status': 404
    }
    response = jsonify(message)
    response.status_code = 404
    return response
```

Nota: El grafico presenta la estructura para obtención de errores diferentes al 404

Error Handler 404: Esta función se encarga de manejar las solicitudes a recursos no encontrados y devuelve un mensaje de error adecuado. En caso de que se realice una solicitud a un endpoint no definido o a un recurso inexistente, se generará una respuesta JSON con un mensaje indicando que el recurso solicitado no ha sido encontrado.

Postman como herramienta para verificar los datos de la API:

Esta herramienta se destaca como una valiosa herramienta para verificar y evaluar los datos que se intercambian mediante una API. Su interfaz gráfica intuitiva facilita el envío de solicitudes HTTP a los distintos puntos finales de la API y la visualización de las respuestas recibidas. Gracias a Postman, es posible llevar a cabo pruebas exhaustivas que aseguren el correcto funcionamiento de la API y la entrega de los resultados deseados.

A continuación, se presentará el siguiente ejemplo;

Primero levantamos el API desde la consola.

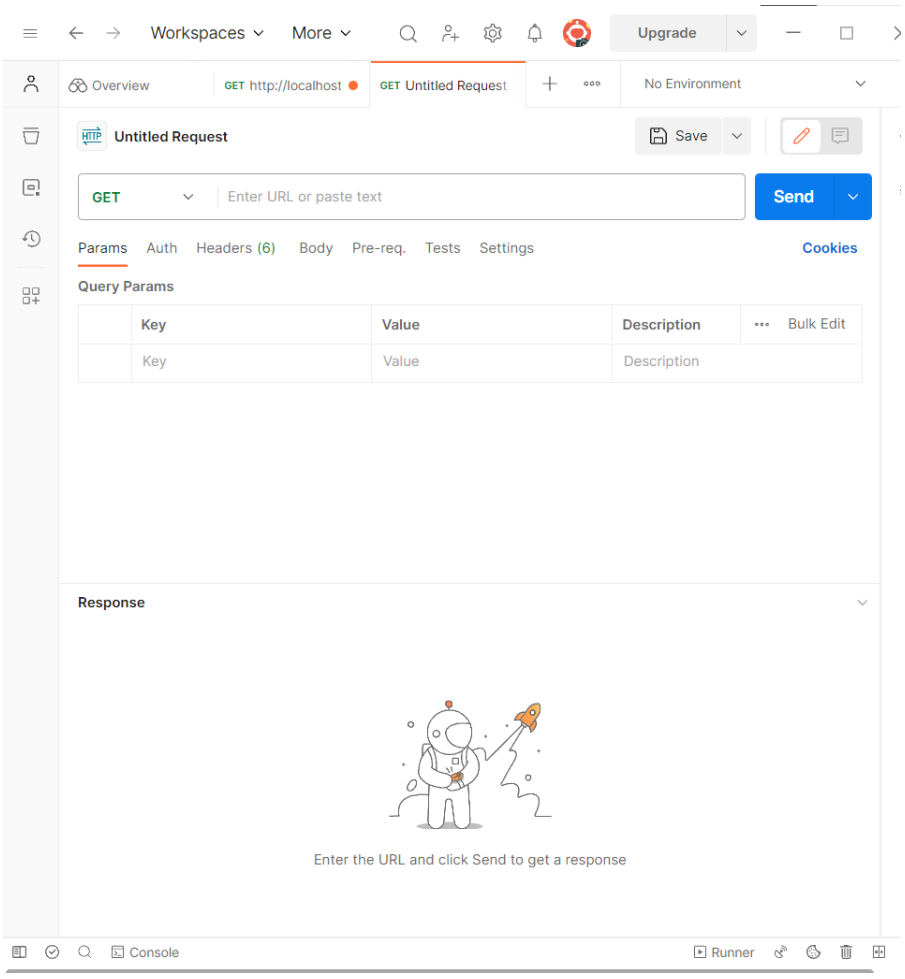
Figura 42
Consola de verificación de ejecución del API

```
(venv) C:\Users\jpram\OneDrive\Desktop\Dashboard\backend>py app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 810-551-314
```

Nota. El grafico indica que el API se levantó correctamente, y da la dirección para su utilización

Después se abre la herramienta Postman.

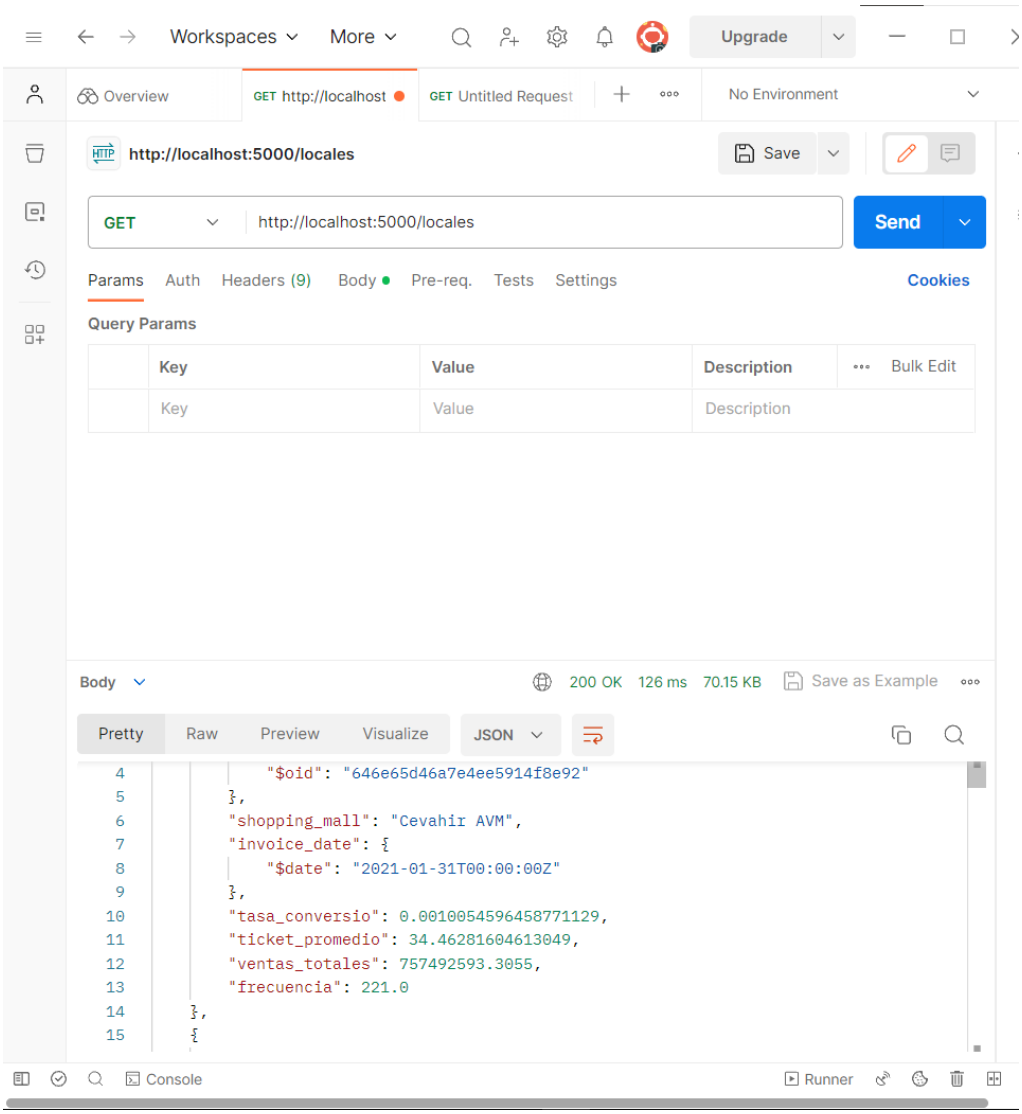
Figura 43
Ventana de la aplicación Postman



Nota. El grafico representa la ventana de inicio de Postman. Extraído desde la herramienta *Postman*

Para la consulta a un endpoint, se tiene los diferentes tipos de peticiones en la parte donde se encuentra el ‘GET’ verde, después ingresamos la dirección proporcionada cuando se inicializa el API seguido de ‘/’, para poder entrar al endpoint que se necesita y se envía la solicitud.

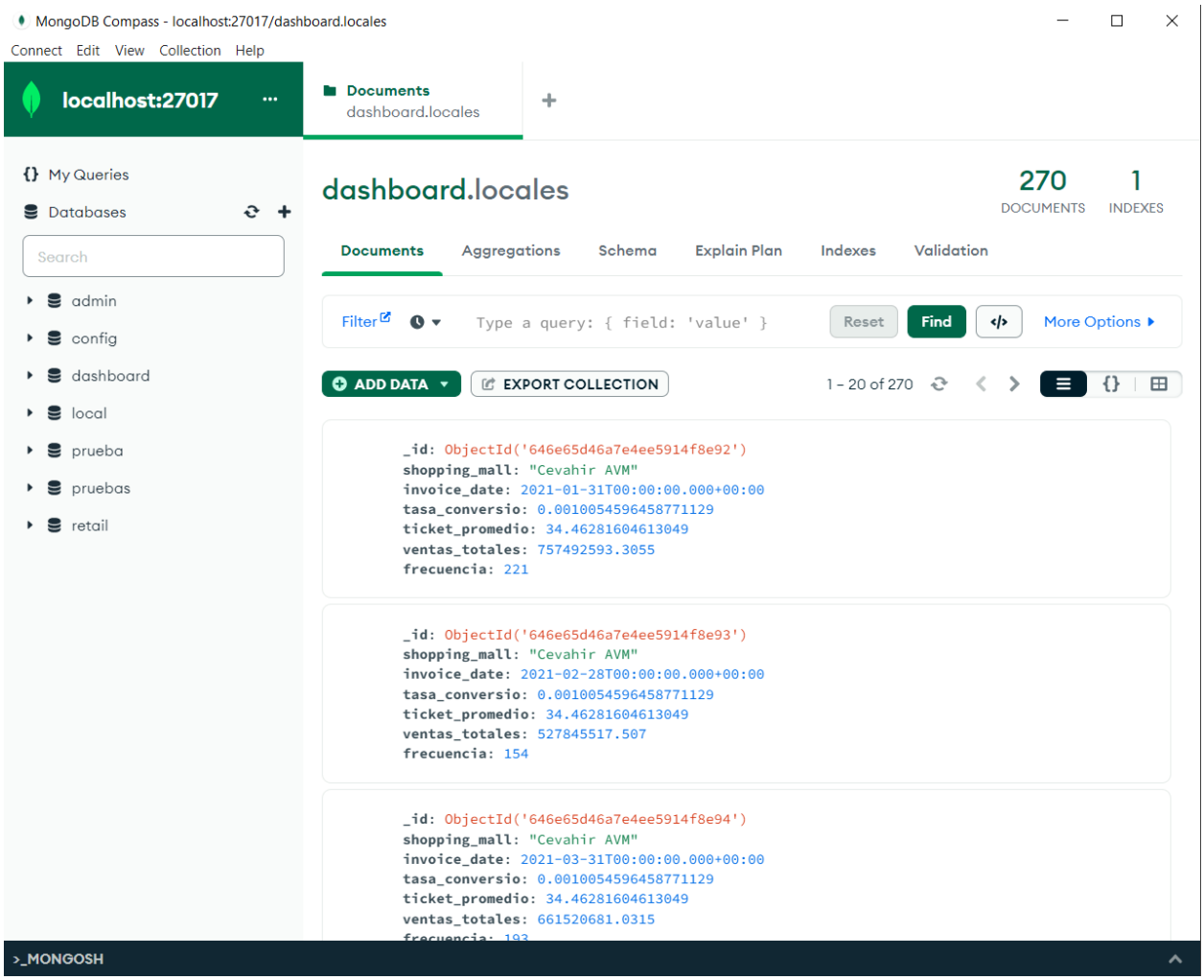
Figura 44
Ventana de ejecucion de prueba para un endpoint



Nota. El grafico representa un ejemplo de consulta a un endpoint en el que devuelve un json de la base de datos. Extraído de la herramienta *Postman*

Observamos que en la parte de body se devuelve una estructura de datos semi estructurada tipo JSON, se puede comprobar la información con la base de datos por consola o con MongoCompass.

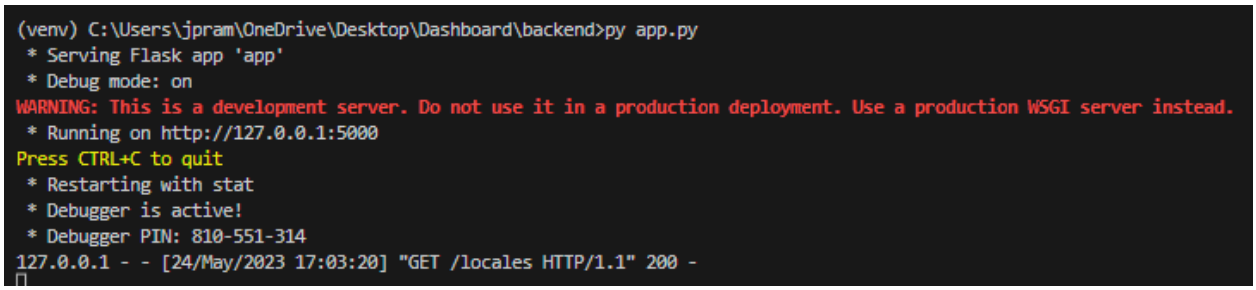
Figura 45
Ventana de MongoDB Compass



Nota. El grafico indica la base de datos dashboard con la colección locales, y los datos que están ya ingresados. Extraído desde *MongoDb Compass*

También, se puede comprobar en la consola donde se ejecuta el API, con una repuesta 200 que nos indica que la conexión fue exitosa.

Figura 46
Consola de estados del API



Nota. El grafico indica que se realizó un GET a un endpoint, y se devolvió un código de respuesta ‘ok’.

Al utilizar Postman, se puede realizar pruebas de integración y verificar que la API responda correctamente a diferentes tipos de solicitudes, como GET, POST, PUT y DELETE. Además, se puede enviar parámetros y datos en el cuerpo de la solicitud, configurar encabezados personalizados, y examinar y validar las respuestas obtenidas. Postman también ofrece funciones adicionales, como la posibilidad de automatizar pruebas y generar documentación de la API. La instalación de React y las diferentes bibliotecas para la creación de un dashboard es un proceso fundamental en el desarrollo de aplicaciones web modernas.

4.9. Frontend

En el tercer sprint, se desarrolló el frontend utilizando React, un framework de JavaScript ampliamente utilizado para construir interfaces de usuario interactivas y escalables. A continuación, se detallará paso a paso cómo llevar a cabo la instalación de React, así como las librerías adicionales mencionadas, como Material-UI, Nivo Charts, Formik, Yup Validation, React Pro Sidebar y Google Fonts.

Pasos para la instalación de React:

1. Verifica la existencia de Node.js en tu sistema asegurándote de que esté instalado. Puedes descargarlo desde el sitio web oficial de Node.js y proceder con la instalación siguiendo las instrucciones específicas para tu sistema operativo.

2. Abre la terminal o línea de comandos de Node.js después de haber completado la instalación y verifica la correcta instalación de Node.js y npm (Node.js Package Manager) ejecutando los comandos que se detallan a continuación:

```
node -v
```

```
npm -v
```

Estos comandos mostrarán las versiones instaladas de Node.js y npm, respectivamente. Si ambos comandos muestran las versiones, significa que la instalación de Node.js se realizó correctamente.

3. A continuación, se instala "create-react-app", una herramienta que simplifica la configuración inicial de un proyecto de React. Ejecuta el siguiente comando en tu terminal:

   ```
   npm install -g create-react-app
   ```

Este comando instalará "create-react-app" globalmente en tu sistema, lo que te permitirá crear proyectos de React en cualquier ubicación.

4. Una vez que se complete la instalación, cree un nuevo proyecto de React ejecutando el siguiente comando:

```
create-react-app frontend
```

Se reemplaza "nombre-del-proyecto" por el nombre de frontend. Esto creará una nueva carpeta con la estructura inicial de un proyecto de React.

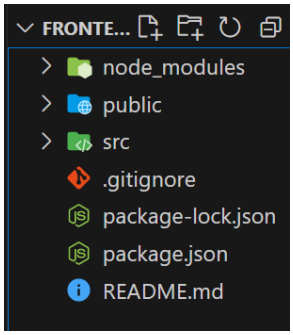
5. Se cambia la dirección del directorio del proyecto recién creado.
6. Ahora, se puede iniciar el servidor de desarrollo de React ejecutando el siguiente comando:

```
npm start
```

Este comando iniciará el servidor de desarrollo y abrirá automáticamente su aplicación de React en tu navegador predeterminado. Cualquier cambio que realices en sus archivos de código se reflejará automáticamente en el navegador.

Explicación de las carpetas generadas por React:

Figura 47
Carpetas del proyecto



Nota. El grafico indica las diferentes carpetas creadas por React para el desarrollo del frontend.

Carpeta "node_modules": En esta carpeta se almacenan todas las dependencias y paquetes de Node.js requeridos por el proyecto. Estas dependencias son instaladas automáticamente al ejecutar el comando "npm install" y son necesarias para el funcionamiento adecuado de la aplicación.

Carpeta "public": En esta carpeta se encuentran los archivos estáticos que se sirven directamente al navegador. Uno de los archivos más destacados es "index.html", que representa la página principal de la aplicación de React. Además, es posible encontrar otros archivos estáticos, como imágenes, iconos o archivos de configuración.

Carpeta "src": Esta carpeta contiene el código fuente de la aplicación de React. Aquí es donde se trabaja principalmente en el proyecto. Incluye varios archivos y carpetas esenciales:

"index.js": Este archivo se utiliza como punto de entrada del proyecto de React. Aquí se importa el componente principal de la aplicación y se renderiza en el elemento del archivo "index.html" ubicado en la carpeta "public".

"App.js": Este archivo alberga el componente principal de la aplicación. Aquí se construye la estructura y lógica de la interfaz de usuario. Es posible modificar este archivo o crear otros componentes y archivos según sea necesario.

"index.css": Este archivo contiene los estilos CSS globales para la aplicación.

Carpeta "assets": En esta carpeta se pueden almacenar archivos estáticos, como imágenes, fuentes u otros recursos utilizados en la aplicación.

Además de las carpetas mencionadas, también se generan otros archivos y carpetas automáticamente, tales como:

"package.json": Este archivo contiene la configuración y metadatos del proyecto, así como la lista de dependencias y scripts de comandos.

"package-lock.json": Este archivo se genera automáticamente para mantener un registro preciso de las versiones de las dependencias instaladas.

"README.md": Este archivo proporciona una descripción básica del proyecto y puede ser editado según sea necesario.

"gitignore": Este archivo especifica qué archivos y carpetas deben ser excluidos al realizar el seguimiento de los cambios en el control de versiones Git. Su objetivo es evitar incluir archivos generados automáticamente o archivos sensibles que no deben ser compartidos públicamente.

También pueden existir otros archivos y carpetas generados, como configuraciones de prueba, configuraciones de linter (herramientas de análisis de código), archivos de configuración de empaquetado, entre otros.

Estas carpetas proporcionan una estructura inicial organizada para desarrollar una aplicación de React utilizando "create-react-app" y las librerías que se instalaron. A medida que se avanza en el proyecto, es posible crear nuevas carpetas y organizar el código según las necesidades y preferencias del desarrollo.

Instalación de librerías adicionales:

Al configurar un proyecto de React, es común la necesidad de agregar funcionalidades adicionales mediante la instalación de librerías específicas. Para facilitar este proceso, se utilizaron diversas librerías populares que ofrecen componentes y funcionalidades útiles para aplicaciones React. Entre estas librerías se encuentran Material-UI, Nivo Charts, Formik, Yup Validation, React Pro Sidebar y Google Fonts.

La instalación de estas librerías adicionales se realiza siguiendo las instrucciones proporcionadas para cada una de ellas. Una vez instaladas, pueden ser importadas en el proyecto de React correspondiente y utilizadas en el código. Para obtener información detallada sobre el uso de estas librerías en la aplicación, se puede hacer referencia al Anexo E, el cual proporciona información adicional las librerías del archivo HTML de la aplicación.

Figura 48
Código del archivo "App.js"

```
function App() {
  const [theme, colorMode] = useMode();
  const [isSidebar, setIsSidebar] = useState(true);
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  const [loginError, setLoginError] = useState(false);

  const handleLogin = (username, password) => {···
  };

  const handleLogout = () => {
    setIsLoggedIn(false);
    window.location.href = '/';
  };

  return (
    <ColorModeContext.Provider value={colorMode}>
      <ThemeProvider theme={theme}>
        <CssBaseline />
        <div className="app">
          <Routes>
            {isLoggedIn ? (
              <>
                <Route path="/" element={<Navigate to="/dashboard" />} />
                <Route path="/registro" element={<SignUp />} />
              </>
            ) : (
              <>
                <Route
                  path="/"
                  element={<Login onLogin={handleLogin} loginError={loginError} />}
                />
                <Route path="/registro" element={<SignUp />} />
              </>
            )}
          </Routes>
          {isLoggedIn && (
            <>
              <Sidebar isSidebar={isSidebar} />
              <div className="content">
                <Topbar setIsSidebar={setIsSidebar} onLogout={handleLogout} />
                <Routes>
                  <Route path="/dashboard" element={<Dashboard />} />
                  <Route path="/team" element={<Team />} />
                  <Route path="/contacts" element={<Contacts />} />
                  <Route path="/invoices" element={<Invoices />} />
                  <Route path="/form" element={<Form />} />
                  <Route path="/bar" element={<Bar />} />
                  <Route path="/pie" element={<Pie />} />
                  <Route path="/line" element={<Line />} />
                  <Route path="/faq" element={<FAQ />} />
                  <Route path="/calendar" element={<Calendar />} />
                  <Route path="/geography" element={<Geography />} />
                </Routes>
              </div>
            </>
          )}
```

Nota: El grafico explica la estructura del archivo App.js

El archivo "App.jsx" contiene el componente principal de la aplicación, denominado "App". Este componente define la estructura general de la aplicación y gestiona el enrutamiento y el estado de autenticación. El componente "App" importa varios componentes y funciones de diferentes archivos. Algunos de estos componentes incluyen "Topbar", "Sidebar", "Dashboard", "Team", "Invoices", "Contacts", "Bar", "Form", "Line", "Pie", "FAQ", "Geography", "Calendar", "Login" y "SignUp". También importa componentes y funciones relacionadas con el tema y el enrutamiento desde las bibliotecas "react-router-dom" y "@mui/material".

Dentro del componente "App", se utiliza el hook "useState" para gestionar el estado de varias variables, como el tema, la visibilidad de la barra lateral, el estado de inicio de sesión y el estado de error de inicio de sesión.

El componente "App" define dos funciones principales: "handleLogin" y "handleLogout". La función "handleLogin" se utiliza para enviar una solicitud de inicio de sesión al servidor cuando se envía el formulario de inicio de sesión. La función "handleLogout" se utiliza para cerrar la sesión del usuario y redirigirlo a la página de inicio.

En el retorno del componente "App", se envuelven los componentes y elementos de la aplicación en los proveedores "ColorModeContext" y "ThemeProvider" para administrar el modo de color y el tema de la aplicación respectivamente. Se utiliza el componente "CssBaseline" para establecer un estilo base consistente en la aplicación.

Dentro del retorno, se definen las rutas de la aplicación utilizando el componente "Routes" de "react-router-dom". Dependiendo del estado de inicio de sesión, se renderizan diferentes rutas y componentes. Si el usuario está autenticado, se muestran rutas protegidas, como el panel de control, el equipo, las facturas, los contactos, etc. Si el usuario no está autenticado, se muestran rutas para el inicio de sesión y el registro de usuario.

Además de las rutas, se renderizan componentes como "Topbar" y "Sidebar" para proporcionar una interfaz de usuario completa. Estos componentes se muestran solo cuando el usuario está autenticado.

Figura 49
Código del archivo "theme.js"

```
import { createContext, useState, useMemo } from "react";
import { createTheme } from "@mui/material/styles";

// color design tokens export
export const tokens = (mode) => ({···
});

// mui theme settings
export const themeSettings = (mode) => {···
};

// context for color mode
export const ColorModeContext = createContext({
  toggleColorMode: () => {},
});

export const useMode = () => {
  const [mode, setMode] = useState("dark");

  const colorMode = useMemo(
    () => ({
      toggleColorMode: () =>
        setMode((prev) => (prev === "light" ? "dark" : "light")),
    }),
    []
  );

  const theme = useMemo(() => createTheme(themeSettings(mode)), [mode]);
  return [theme, colorMode];
};
```

Nota: El grafico explica la estructura de forma general del código del archivo “theme.js”, para su exportación.

El archivo "theme.js" es un archivo JavaScript que contiene la configuración del tema y los colores utilizados en la aplicación de React. Este archivo exporta varias funciones y constantes que se utilizan para definir el tema y los colores del proyecto.

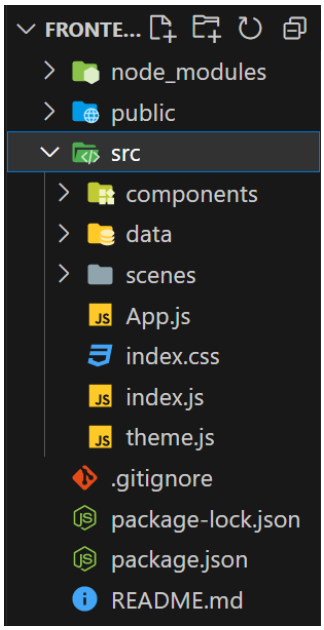
En primer lugar, se define la función "tokens" que recibe como parámetro el modo (ya sea "dark" o "light") y devuelve un objeto con una serie de colores predefinidos. Estos colores incluyen tonos de gris, colores primarios, colores de acento verde, rojo y azul, dependiendo del modo especificado.A continuación, se encuentra la función "themeSettings" que también recibe el modo

como parámetro y utiliza los colores definidos en la función "tokens" para establecer la paleta de colores del tema. Esto incluye los colores principales, secundarios y neutros, así como el color de fondo.

Se define el contexto "ColorModeContext" que se utiliza para proporcionar la capacidad de cambiar el modo de color en la aplicación. Este contexto contiene una función "toggleColorMode" que actualiza el estado del modo de color entre "light" y "dark".

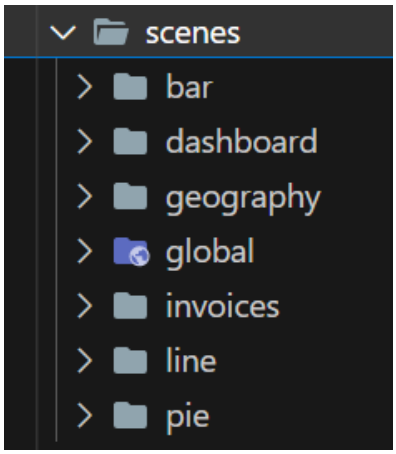
Por último, se encuentra la función "useMode" que utiliza el hook "useState" para gestionar el estado del modo de color y el hook "useMemo" para memorizar el tema basado en el modo de color seleccionado. Esta función devuelve un array que contiene el tema y el contexto de modo de color.

Figura 50
Carpetas y archivos dentro de la carpeta src



Nota. El grafico indica los diferentes archivos y carpetas que se encuentran dentro de la carpeta ‘src’.

Figura 51
Carpetas que se encuentran dentro de la carpeta 'scenes'



Nota. El grafico indica las diferentes carpetas que se encuentran dentro de la carpeta ’sccene’, para la renderización de los objetos

La carpeta "scenes" ha sido creada en paralelo al archivo "App.js" y se utiliza para almacenar los componentes JSX correspondientes a las diferentes secciones del dashboard. Esta estructura organizativa facilita el desarrollo y mantenimiento del proyecto al separar claramente las funcionalidades y componentes relacionados. A continuación, se describen brevemente las subcarpetas presentes dentro de "scenes" y su función principal:

"bar": Esta subcarpeta contiene los archivos JSX relacionados con la sección de gráficos de barras del dashboard. Aquí se encuentran componentes específicos para visualizar datos en forma de gráficos de barras, lo que permite una representación efectiva de información estadística o comparativa.

"dashboard": En la subcarpeta "dashboard" se encuentran los archivos JSX relacionados con la sección principal del dashboard. Aquí se ubican componentes clave para mostrar y organizar información relevante del dashboard, como estadísticas, gráficos y resúmenes de datos.

"geography": En la carpeta "geography" se encuentran los archivos JSX relacionados con la sección de geografía del dashboard. Aquí se encuentran componentes específicos para mostrar y visualizar datos geográficos, como mapas interactivos y gráficos geográficos.

"global": La subcarpeta "global" alberga los archivos JSX relacionados con componentes globales utilizados en todo el dashboard. Aquí se pueden encontrar componentes compartidos, como la barra superior (topbar) y la barra lateral (sidebar), que proporcionan una interfaz coherente y una experiencia de usuario consistente en todas las secciones del dashboard.

"line": La carpeta "line" contiene los archivos JSX relacionados con la sección de gráficos de líneas del dashboard. Aquí se encuentran componentes específicos para visualizar datos en forma de gráficos de líneas y analizar tendencias y cambios en los datos a lo largo del tiempo.

"pie": En la subcarpeta "pie" se encuentran los archivos JSX relacionados con la sección de gráficos circulares (pie charts) del dashboard. Aquí se encuentran componentes específicos para mostrar datos en forma de gráficos circulares, lo cual permite una representación visual clara y concisa de la distribución o proporción de datos en diferentes categorías.

“login”. La subcarpeta contiene el archivo JSX relacionados con la sección del inicio de sesión del usuario para ingresar al dashboard. Aquí se encuentra los componentes para enviar los datos mediante un formulario hacia el “App.js”, y se realice la autenticación de las credenciales del usuario

“signup”. La subcarpeta contiene un archivo JSX con relación a la sección registro de usuario para el ingreso al dashboard. Aquí se encuentra los componentes para enviar mediante un formulario los datos de registro de usuario

Dentro de la carpeta "scenes" se encuentran diferentes subcarpetas que contienen los archivos relacionados con las visualizaciones y escenas específicas que se van a renderizar en

el dashboard. Cada subcarpeta contiene componentes JSX que definen la estructura, el diseño y la lógica de cada una de estas visualizaciones.

El archivo "index.jsx" en la carpeta "dashboard" desempeña un papel fundamental en la representación del panel de control de la aplicación. Contiene una serie de importaciones de componentes y datos necesarios para su funcionamiento.

Figura 52
Código de librerías de index.jsx de la carpeta dashboard

```
import { Box, Button, IconButton, Typography, useTheme } from "@mui/material";
import { tokens } from "../../theme";
import { mockTransactions } from "../../data/mockData";
import DownloadOutlinedIcon from "@mui/icons-material/DownloadOutlined";
import EmailIcon from "@mui/icons-material/Email";
import PointOfSaleIcon from "@mui/icons-material/PointOfSale";
import PersonAddIcon from "@mui/icons-material/PersonAdd";
import TrafficIcon from "@mui/icons-material/Traffic";
import Header from "../../components/Header";
import LineChart from "../../components/LineChart";
import GeographyChart from "../../components/GeographyChart";
import BarChart from "../../components/BarChart";
import StatBox from "../../components/StatBox";
import ProgressCircle from "../../components/ProgressCircle";
```

Nota: El grafico explica las librerías que se utilizan en el archivo index.jsx de la carpeta dashboard.

El componente principal del archivo es "Dashboard". En este componente, se utiliza el hook "useTheme" para obtener el tema actual de la aplicación y generar una paleta de colores correspondiente. Luego, se define la estructura general del panel de control utilizando el componente "Box" como contenedor principal.

Figura 53
Código del archivo index.jsx de la carpeta dashboard

```
const Dashboard = () => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);

  return (
    <Box m="20px">
      {/* HEADER */}
      <Box display="flex" justifyContent="space-between" alignItems="center">
        <Header title="DASHBOARD" subtitle="Welcome to your dashboard" />

        <Box>
          <Button
            sx={{
              backgroundColor: colors.blueAccent[700],
              color: colors.grey[100],
              fontSize: "14px",
              fontWeight: "bold",
              padding: "10px 20px",
            }}
          >
            <DownloadOutlinedIcon sx={{ mr: "10px" }} />
            Download Reports
          </Button>
        </Box>
      </Box>
```

Nota: El grafico explica la estructura del código en archivo index.jsx del dashboard.

Dentro del componente "Dashboard", se crean secciones como el encabezado, los gráficos de líneas, el gráfico de geografía, el gráfico de barras, las cajas de estadísticas y el círculo de progreso. Estas secciones se colocan dentro del contenedor principal utilizando el componente "Box" y se definen sus propiedades, como el estilo, el color de fondo y la disposición en la cuadrícula.

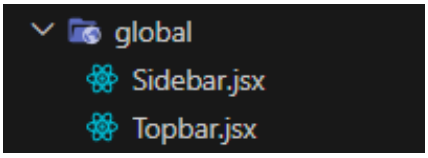
En la sección del encabezado, se muestra el título del panel de control y se agrega un botón para descargar un informe. En la sección de gráficos de líneas, se muestra un gráfico interactivo de las ventas por oportunidades. En la sección de gráfico de geografía, se muestra un mapa que representa el estado de los locales en función de las ventas. En la sección de gráfico de barras, se muestra un gráfico que ilustra las ventas totales por categoría.

Además de los gráficos, se agregan cajas de estadísticas que muestran información relevante, como el número de ventas, el número de facturas, el ticket promedio y la tasa de conversión. Estas cajas incluyen iconos relacionados y una barra de progreso para indicar el crecimiento o disminución de cada métrica.

Por último, se utiliza el componente "mockTransactions" para mostrar transacciones recientes en una sección de lista desplazable.

El componente "Dashboard" se exporta como el valor predeterminado del archivo, lo que permite su uso en otras partes de la aplicación para mostrar el panel de control completo.

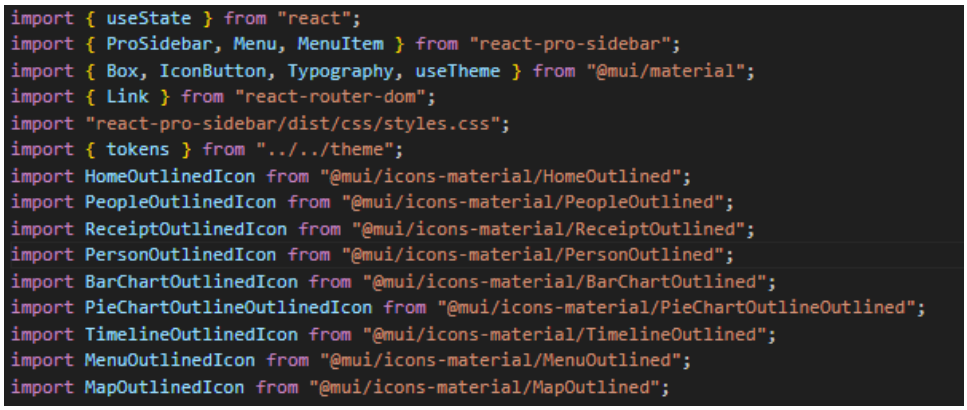
Figura 54
Carpeta global dentro de la carpeta scene



Nota: El grafico explica el contenido de la carpeta global, SideBar.jsx y TopBar.jsx.

El archivo "Sidebar.jsx" ubicado en la carpeta "global" desempeña un papel esencial en la generación y renderizado de la barra lateral en el dashboard. Este archivo contiene una serie de importaciones de componentes y estilos necesarios para el funcionamiento adecuado de la barra lateral.

Figura 55
Código de las librerías del archivo SideBar.jsx

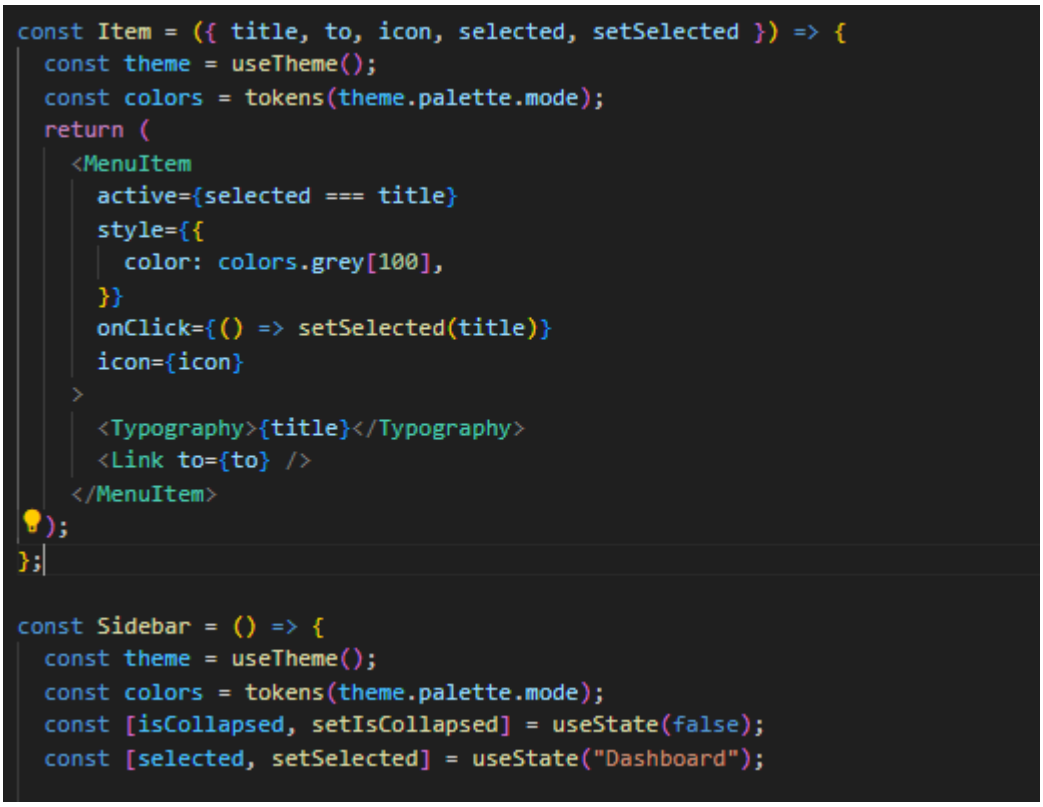


```jsx
import { useState } from "react";
import { ProSidebar, Menu, MenuItem } from "react-pro-sidebar";
import { Box, IconButton, Typography, useTheme } from "@mui/material";
import { Link } from "react-router-dom";
import "react-pro-sidebar/dist/css/styles.css";
import { tokens } from "../../theme";
import HomeOutlinedIcon from "@mui/icons-material/HomeOutlined";
import PeopleOutlinedIcon from "@mui/icons-material/PeopleOutlined";
import ReceiptOutlinedIcon from "@mui/icons-material/ReceiptOutlined";
import PersonOutlinedIcon from "@mui/icons-material/PersonOutlined";
import BarChartOutlinedIcon from "@mui/icons-material/BarChartOutlined";
import PieChartOutlineOutlinedIcon from "@mui/icons-material/PieChartOutlineOutlined";
import TimelineOutlinedIcon from "@mui/icons-material/TimelineOutlined";
import MenuOutlinedIcon from "@mui/icons-material/MenuOutlined";
import MapOutlinedIcon from "@mui/icons-material/MapOutlined";
```

Nota: El grafico presenta las librerrias que se utilizan en el archivo SideBar.jsx

El componente principal del archivo es "Sidebar", el cual representa la barra lateral en sí. Se hace uso del componente "ProSidebar" de la biblioteca "react-pro-sidebar" como contenedor principal. Dentro de este contenedor, se utiliza el componente "Menu" para definir el menú de la barra lateral. Dentro del componente "Sidebar", se definen múltiples componentes "Item", los cuales representan cada elemento del menú de la barra lateral. Estos componentes "Item" se encargan de renderizar los elementos del menú con sus respectivos títulos, íconos y rutas asociadas. Cada elemento del menú se representa mediante un componente "MenuItem" proveniente de la biblioteca "react-pro-sidebar". Además, se importan varios íconos de la biblioteca de íconos de Material-UI, como "HomeOutlinedIcon", "BarChartOutlinedIcon", "PieChartOutlineOutlinedIcon", "TimelineOutlinedIcon" y "MapOutlinedIcon". Estos íconos se asignan a los elementos del menú para proporcionar una representación visual de cada opción.

Figura 56
Código del archivo SideBar.jsx



```
const Item = ({ title, to, icon, selected, setSelected }) => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);
  return (
    <MenuItem
      active={selected === title}
      style={{
        color: colors.grey[100],
      }}
      onClick={() => setSelected(title)}
      icon={icon}
    >
      <Typography>{title}</Typography>
      <Link to={to} />
    </MenuItem>
  );
};

const Sidebar = () => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);
  const [isCollapsed, setIsCollapsed] = useState(false);
  const [selected, setSelected] = useState("Dashboard");
```

Nota: El grafico explica la estructura del código del archivo SideBar.jsx

El archivo también incluye lógica para el manejo del estado de la barra lateral, como el estado de colapso y el elemento actualmente seleccionado. Se utiliza el hook "useState" de React para gestionar estos estados.

Figura 57
Código del archivo "Topbar.jsx"

```jsx
import { Box, IconButton, useTheme } from "@mui/material";
import { useContext } from "react";
import { ColorModeContext, tokens } from "../../theme";
import LightModeOutlinedIcon from "@mui/icons-material/LightModeOutlined";
import DarkModeOutlinedIcon from "@mui/icons-material/DarkModeOutlined";
import ExitToAppOutlinedIcon from "@mui/icons-material/ExitToAppOutlined"; // Nuevo icono de cierre de sesión

const Topbar = ({ onLogout }) => { // Añadimos la prop onLogout para manejar el cierre de sesión
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);
  const colorMode = useContext(ColorModeContext);

  return (
    <Box display="flex" justifyContent="space-between" p={2}>
      <Box>
      </Box>

      {/* ICONS */}
      <Box display="flex">
        <IconButton onClick={colorMode.toggleColorMode}>
          {theme.palette.mode === "dark" ? (
            <DarkModeOutlinedIcon />
          ) : (
            <LightModeOutlinedIcon />
          )}
        </IconButton>
        <IconButton onClick={onLogout}> {/* Añadimos el evento onClick para el cierre de sesión */}
          <ExitToAppOutlinedIcon />
        </IconButton>
      </Box>
    </Box>
  );
};

export default Topbar;
```

Nota: El grafico explica la estructura del código del archivo “Topbar.jsx”

El archivo "Topbar.jsx" contiene el componente "Topbar", que representa la barra superior del dashboard. Esta barra contiene una serie de iconos de acción y se ajusta según el tema y el modo de color de la aplicación.

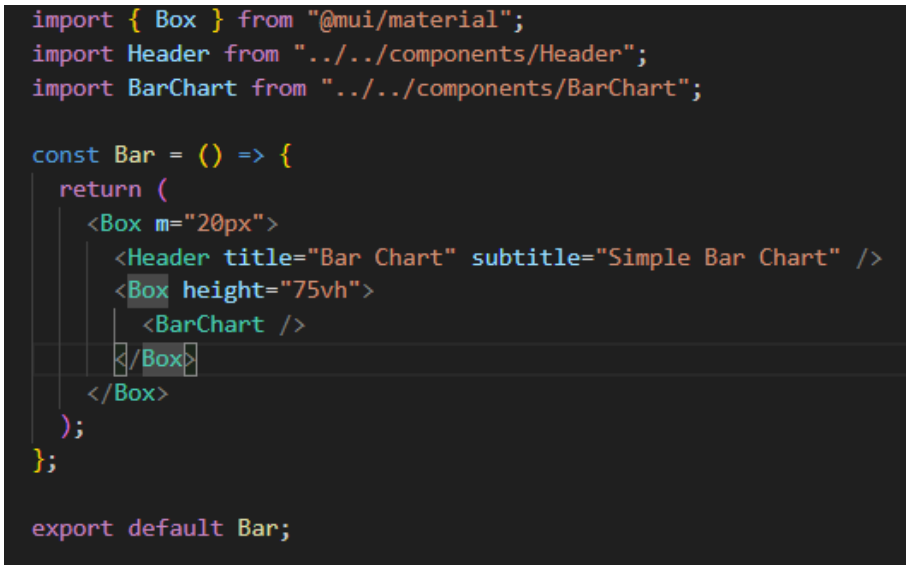
Se utiliza el hook "useTheme" para acceder al tema actual de la aplicación y el hook "useContext" para acceder al contexto de modo de color definido en "ColorModeContext".

Dentro del componente, se utiliza el componente "Box" de Material-UI para crear un contenedor que alinea los elementos de la barra en un diseño de espacio entre ellos. La barra superior consta de dos secciones principales:

En la sección de iconos, se utilizan los componentes "IconButton" de Material-UI para representar cada uno de los íconos de acción. Estos íconos incluyen el ícono de cambio de modo de color (representado por "LightModeOutlinedIcon" y "DarkModeOutlinedIcon"). Al hacer clic en este ícono, se invoca la función "toggleColorMode" del contexto de modo de color para alternar entre los modos de color claro y oscuro. Adicionalmente, se importa el ícono "ExitToAppOutlinedIcon" para el cierre de sesión. Al hacer clic en este ícono, se ejecuta la función "onLogout" pasada como prop para manejar el cierre de sesión.

El archivo "index.jsx" en la carpeta "bar" es esencial para la representación de la sección de gráficos de barras en el dashboard. En este archivo, se importan los componentes necesarios, como "Box", "Header" y "BarChart".

Figura 58
Código del archivo index.jsx de la carpeta bar



```jsx
import { Box } from "@mui/material";
import Header from "../../components/Header";
import BarChart from "../../components/BarChart";

const Bar = () => {
  return (
    <Box m="20px">
      <Header title="Bar Chart" subtitle="Simple Bar Chart" />
      <Box height="75vh">
        <BarChart />
      </Box>
    </Box>
  );
};

export default Bar;
```

Nota: El grafico presenta el código del archivo index.jsx de la carpeta bar

El componente principal del archivo es "Bar", que utiliza el componente "Box" como contenedor principal. Dentro de este contenedor, se muestra el componente "Header" con un título y subtítulo relacionados con los gráficos de barras.

Tambien, se utiliza otro componente "Box" con una altura específica para definir el área de visualización del gráfico de barras. Dentro de este contenedor de altura, se renderiza el componente "BarChart", responsable de generar y mostrar el gráfico de barras en sí.

El componente "Bar" se exporta como el valor predeterminado del archivo, lo que permite su uso en otras partes de la aplicación donde se requiera mostrar la sección de gráficos de barras. Se toma en cuenta la estructura de bar para la demás técnica de visualización, podemos esperar

Figura 59
Código del archivo index.jsx de la carpeta login

```jsx
import React, { useState } from 'react';
import axios from 'axios';
import { Grid, Paper, Avatar, TextField, Button, Typography, Link, FormControlLabel, Checkbox, useTheme } from '@mui/material';
import { LockOutlined } from '@mui/icons-material';
import { tokens } from "../../theme";

const Login = ({ onLogin, loginError }) => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);

  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');

  const handleSignIn = () => {
    onLogin(username, password);
  };

  return (
    <Grid container justifyContent="center" alignItems="center" style={{ minHeight: '100vh' }}>
      <Grid item xs={12} sm={8} md={6} lg={4}>
        <Paper elevation={10} sx={{ p: 4, backgroundColor: colors.blueAccent[700] }}>
          <Grid container alignItems="center" justifyContent="center" spacing={2}>···
          </Grid>
          <TextField
            label="Username"
            placeholder="Ingresa tu username"
            fullWidth
            required
            sx={{ mt: 3 }}
            value={username}
            onChange={event => setUsername(event.target.value)}
          />
          <TextField
            label="Contraseña"
            placeholder="Ingresa tu contraseña"
            type="password"
            fullWidth
            required
            sx={{ mt: 2 }}
            value={password}
            onChange={event => setPassword(event.target.value)}
          />
```

Nota: El grafico presenta la estructura del código del archivo indez.jsx de la carpeta login.

El archivo "index.jsx" contiene el componente "Login", el cual representa el formulario de inicio de sesión en la aplicación. Este componente se encarga de recopilar las credenciales de inicio de sesión del usuario y enviarlas al servidor para su autenticación.

El componente "Login" utiliza el hook "useState" de React para gestionar el estado de los campos de entrada de texto "username" y "password". Estos estados se inicializan con valores vacíos.Dentro del componente se utilizan varios componentes de Material-UI, como "Grid", "Paper", "Avatar", "TextField", "Button", "Typography", "Link" y "FormControlLabel". Además, se importa el ícono "LockOutlined" de Material-UI.

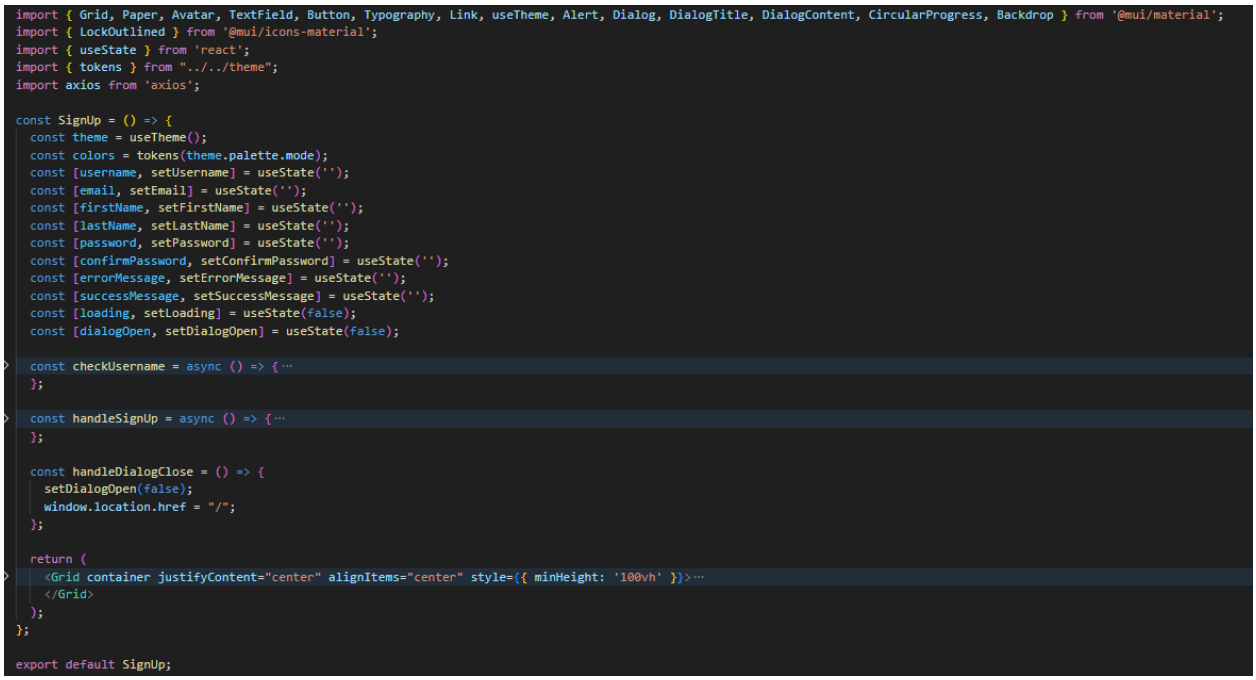
El componente renderiza una cuadrícula "Grid" que centra verticalmente su contenido en la pantalla. Dentro de esta cuadrícula se encuentra un elemento "Paper" que actúa como un contenedor con un sombreado elevado. Este contenedor contiene el formulario de inicio de sesión. El formulario de inicio de sesión consta de un avatar que muestra el ícono de un candado "LockOutlined", un título de encabezado "Typography", dos campos de texto "TextField" para el nombre de usuario y la contraseña, un botón de inicio de sesión "Button", un enlace para registrarse "Link" y un mensaje de error de inicio de sesión "Typography" condicional.

Los campos de texto para el nombre de usuario y la contraseña utilizan los estados "username" y "password", respectivamente, para almacenar los valores ingresados por el usuario. Los eventos "onChange" se utilizan para actualizar estos estados a medida que el usuario escribe en los campos de texto.

Cuando se hace clic en el botón de inicio de sesión, se invoca la función "handleSignIn", la cual a su vez invoca la función "onLogin" pasada como prop. La función "onLogin" se utiliza para enviar las credenciales de inicio de sesión al servidor con el fin de autenticar al usuario.

Si se produce un error durante el inicio de sesión, se muestra un mensaje de error condicional debajo del botón de inicio de sesión.

Figura 60
Código del archivo index.jsx de la carpeta SignUp



```
import { Grid, Paper, Avatar, TextField, Button, Typography, Link, useTheme, Alert, Dialog, DialogTitle, DialogContent, CircularProgress, Backdrop } from '@mui/material';
import { LockOutlined } from '@mui/icons-material';
import { useState } from 'react';
import { tokens } from "../../theme";
import axios from 'axios';

const SignUp = () => {
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);
  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [firstName, setFirstName] = useState('');
  const [lastName, setLastName] = useState('');
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const [errorMessage, setErrorMessage] = useState('');
  const [successMessage, setSuccessMessage] = useState('');
  const [loading, setLoading] = useState(false);
  const [dialogOpen, setDialogOpen] = useState(false);

  const checkUsername = async () => {···
  };

  const handleSignUp = async () => {···
  };

  const handleDialogClose = () => {
    setDialogOpen(false);
    window.location.href = "/";
  };

  return (
    <Grid container justifyContent="center" alignItems="center" style={{ minHeight: '100vh' }}>···
    </Grid>
  );
};

export default SignUp;
```

Nota: El grafico explica una estructura de código resumida del archivo index.jsx de la carpeta SignUp

El archivo "index.jsx" contiene el componente "SignUp", el cual representa el formulario de registro de usuarios en la aplicación. Este componente se encarga de recopilar la información del usuario, como nombre, apellido, nombre de usuario, correo electrónico y contraseña, y enviarla al servidor para crear una nueva cuenta de usuario.

El componente "SignUp" utiliza el hook "useState" de React para gestionar el estado de los campos de entrada de texto, mensajes de error y éxito, así como el estado de carga.

Dentro del componente se utilizan varios componentes de Material-UI, como "Grid", "Paper", "Avatar", "TextField", "Button", "Typography", "Link", "Alert", "Dialog", "DialogTitle", "DialogContent", "CircularProgress" y "Backdrop". Además, se importa el ícono "LockOutlined" de Material-UI. El componente renderiza una cuadrícula ("Grid") que centra verticalmente su contenido en la pantalla. Dentro de esta cuadrícula se encuentra un elemento "Paper" que actúa

como un contenedor con un sombreado elevado. Este contenedor contiene el formulario de registro de usuarios.

El formulario de registro de usuarios incluye campos de texto ("TextField") para el nombre, apellido, nombre de usuario, correo electrónico, contraseña y confirmación de contraseña. Estos campos utilizan los estados correspondientes para almacenar los valores ingresados por el usuario.

Además de los campos de texto, se muestra mensajes de error y éxito condicionales utilizando el componente "Alert" de Material-UI. Si se produce un error durante el registro, se muestra un mensaje de error. Si el registro es exitoso, se muestra un mensaje de éxito.

El botón "Regístrate" trae la función "handleSignUp" cuando se hace clic en él. Esta función realiza varias comprobaciones, como verificar si las contraseñas coinciden y si el nombre de usuario ya existe en el servidor. Si todas las comprobaciones son exitosas, se envía una solicitud al servidor para crear una nueva cuenta de usuario.

Si el registro es exitoso, se muestra un diálogo de confirmación con el mensaje de éxito y un botón "Aceptar". Al hacer clic en el botón "Aceptar", se cierra el diálogo y se redirige al usuario a la página de inicio de sesión. Durante el proceso de registro, se muestra un indicador de carga ("CircularProgress") y un fondo oscuro ("Backdrop") para indicar que el proceso está en curso.

Figura 61
Código del archivo index.jsx de la carpeta line

```jsx
import { Box } from "@mui/material";
import Header from "../../components/Header";
import LineChart from "../../components/LineChart";

const Line = () => {
  return (
    <Box m="20px">
      <Header title="Line Chart" subtitle="Simple Line Chart" />
      <Box height="75vh">
        <LineChart />
      </Box>
    </Box>
  );
};

export default Line;
```

Nota: El grafico presenta la estructura del código del archivo index.jsx de la carpeta linea

El archivo "index.jsx" se encuentra en la carpeta "line" y desempeña un papel importante al representar y mostrar un gráfico de línea simple en el panel de control.

En este archivo, se importa el componente "Box" de Material-UI para crear un contenedor que engloba todo el contenido. También se importa el componente "Header" desde la carpeta "components" para mostrar el título y el subtítulo del gráfico de línea.

La función principal del archivo es "Line", que se encarga de renderizar el contenido del gráfico de línea. Dentro de esta función, se utiliza el componente "Header" para mostrar el título "Line Chart" y el subtítulo "Simple Line Chart". Luego, se crea un contenedor "Box" que envuelve al componente "LineChart", el cual se encarga de representar el gráfico de línea en sí. El archivo "index.jsx" en la carpeta "line" cumple la función de mostrar un gráfico de línea simple en el panel de control. Utiliza los componentes "Header" y "Box" para mostrar el título, el subtítulo y el contenedor del gráfico. El componente "LineChart" es responsable de renderizar el gráfico de línea en el componente principal.

Figura 62
Código del archivo index.jsx de la carpeta pie

```jsx
import { Box } from "@mui/material";
import Header from "../../components/Header";
import PieChart from "../../components/PieChart";

const Pie = () => {
  return (
    <Box m="20px">
      <Header title="Pie Chart" subtitle="Simple Pie Chart" />
      <Box height="75vh">
        <PieChart />
      </Box>
    </Box>
  );
};

export default Pie;
```

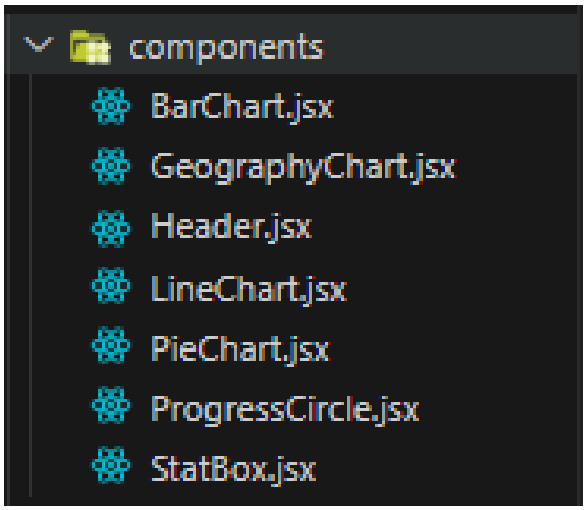
Nota: El grafico presenta la estructura del código del archivo index.jsx de la carpeta pie

El archivo "index.jsx" se encuentra en la carpeta "pie" y desempeña un papel importante al presentar un gráfico de torta simple en el contexto del panel de control.

En dicho archivo, se importa el componente "Box" de la biblioteca Material-UI para crear un contenedor que englobe el contenido relevante. También se importa el componente "Header" desde la carpeta "components" con el propósito de mostrar el título y el subtítulo del gráfico de torta.

La función principal del archivo es denominada "Pie", y su objetivo es renderizar el contenido del gráfico de torta. En su implementación, se utiliza el componente "Header" para mostrar el título "Pie Chart" y el subtítulo "Simple Pie Chart". Además, se crea un contenedor "Box" que envuelve el componente "PieChart", el cual se encarga de representar el gráfico de torta propiamente dicho. Se cumple la función esencial de presentar un gráfico de torta simple en el contexto del panel de control. Se vale de los componentes "Header" y "Box" para mostrar el título, el subtítulo y el contenedor del gráfico, mientras que el componente "PieChart" se encarga de representar visualmente el gráfico de torta dentro de la función principal.

Figura 63
Carpeta con las técnicas de visualización



Nota. El grafico indica las técnicas de visualización desarrolladas para la utilización en las diferentes pantallas del dashboard.

La carpeta "components" contiene una variedad de componentes reutilizables que se utilizan en diferentes partes de la aplicación. Estos componentes están diseñados para mejorar la consistencia visual, promover la reutilización de código y facilitar el desarrollo eficiente.

BarChart.jsx: Este componente proporciona una representación visual de datos en forma de gráfico de barras. Permite mostrar y comparar diferentes categorías o valores de manera clara y concisa.

GeographyChart.jsx: Este componente se encarga de mostrar datos geográficos en forma de mapa interactivo. Permite visualizar información relacionada con ubicaciones geográficas y resaltar diferentes regiones en base a los datos proporcionados.

Header.jsx: Este componente se utiliza para mostrar títulos y subtítulos en diferentes secciones de la aplicación. Proporciona una estructura coherente para la cabecera de la página y ayuda a los usuarios a identificar rápidamente la sección en la que se encuentran.

LineChart.jsx: Este componente permite representar datos en forma de gráfico de línea. Es especialmente útil para visualizar tendencias, cambios a lo largo del tiempo o comparaciones entre diferentes conjuntos de datos.

PieChart.jsx: Este componente ofrece una representación visual de datos en forma de gráfico circular. Permite mostrar proporciones y porcentajes de manera intuitiva, dividiendo el círculo en sectores proporcionales a los valores representados.

ProgressCircle.jsx: Este componente muestra un indicador circular de progreso o carga. Es útil para informar a los usuarios sobre el avance de una tarea o el tiempo restante de una operación en curso.

StatBox.jsx: Este componente se utiliza para mostrar estadísticas o métricas clave de manera visualmente atractiva. Puede incluir números, iconos y otros elementos visuales para resaltar la información más relevante.

Estos componentes están construidos utilizando la biblioteca Material-UI o alguna otra biblioteca de UI, lo que garantiza una apariencia moderna y consistente en toda la aplicación. Al utilizar estos componentes reutilizables, se promueve modularidad, mantenimiento eficiente y la mejora de la experiencia del usuario en la aplicación.

Los archivos en la carpeta "components" siguen una estructura similar en general. Estos archivos contienen componentes reutilizables que se utilizan para representar diferentes elementos en la aplicación.

En cada archivo, se importan las dependencias necesarias, como componentes de bibliotecas o módulos personalizados. También se pueden importar estilos o variables relacionadas con el tema de la aplicación. Además, puede haber importaciones para realizar conexiones con APIs externas u obtener datos necesarios para el componente.

Dentro de cada archivo, se define el componente como una función o clase de React. Dependiendo del caso, se pueden utilizar hooks como useState, useEffect, useContext, entre otros, para gestionar el estado y los efectos dentro del componente. Estos hooks permiten manejar la lógica y los cambios de estado de manera eficiente.

En algunos casos, se pueden realizar solicitudes HTTP a una API externa utilizando librerías como axios para obtener los datos necesarios. Estas solicitudes pueden estar dentro de un efecto useEffect y se utilizan para actualizar el estado del componente con la respuesta de la API.

Además de la lógica específica del componente, se configuran las propiedades y estilos de los componentes importados. Esto implica proporcionar valores para las propiedades requeridas, personalizar estilos utilizando las variables del tema o CSS, y definir la estructura y comportamiento del componente.

Los archivos en la carpeta "components" contienen componentes reutilizables con una estructura similar. Estos archivos importan dependencias, definen componentes de React, pueden realizar conexiones con APIs externas para obtener datos, y configuran propiedades y estilos para personalizar el comportamiento y la apariencia del componente.

La carpeta "public" en un proyecto web desempeña un papel fundamental al proporcionar un directorio especial que contiene los archivos estáticos accesibles públicamente desde el navegador del usuario. Esta carpeta almacena principalmente archivos HTML, CSS, JavaScript, imágenes y otros recursos necesarios para la visualización y funcionamiento adecuado del sitio web.

La inclusión de la carpeta en el proyecto web se basa en la necesidad de separar y organizar de manera clara los archivos que estarán disponibles para el público en general, permitiendo un acceso sencillo y directo a los recursos estáticos requeridos. Estos archivos se envían al cliente sin ningún procesamiento adicional por parte del servidor.

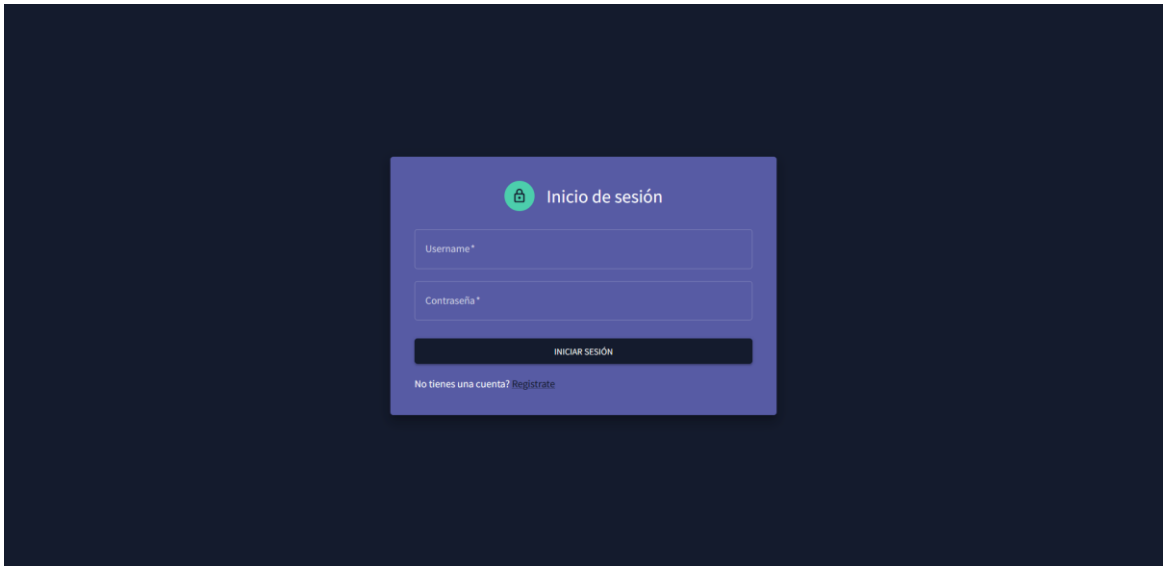
Dentro de la carpeta, se encuentran subdirectorios que agrupan los distintos tipos de archivos estáticos. Por ejemplo, el directorio "css" alberga los archivos de hojas de estilo CSS, el directorio "js" contiene los archivos de JavaScript, y el directorio "images" almacena las imágenes utilizadas en el sitio web.

La presencia de la carpeta "public" resulta fundamental en la arquitectura de un proyecto web, ya que facilita la referencia y carga de los archivos estáticos desde las páginas HTML u otros componentes del proyecto. Esto se logra mediante el uso de rutas relativas para establecer enlaces y referencias a los recursos estáticos. Esta carpeta se utiliza como ubicación predeterminada para los archivos generados durante el proceso de construcción del proyecto, como los archivos optimizados para entornos de producción. Estos archivos se colocan en la carpeta para ser servidos directamente al usuario final sin necesidad de un procesamiento adicional.

4.9.1. Visualización del dashboard

En esta sección se explicará el despliegue de la información con los datos por defecto de la librería de gráficos Nivo Chart. En el siguiente apartado se realizarán las pruebas con los datos conectados a la base de datos del proyecto.

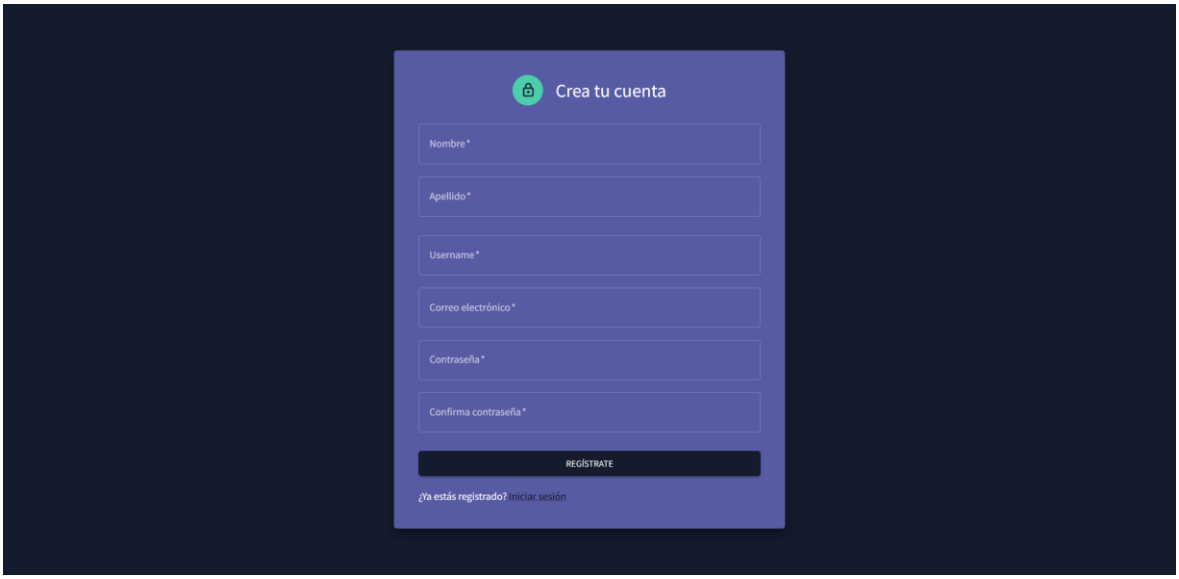
Figura 64
Pantalla de inicio de sesión del dashboard



Nota: El grafico representa el formulario de inicio de sesión para el ingreso al dashboard.

En esta pantalla se presenta un formulario de inicio de sesión en un contenedor, tomando los temas de colores para la visualización. En la parte de abajo se tiene un enlace para ingresar a la pestaña de registro o creación de cuenta.

Figura 65
Pantalla de registro de usuario



Nota: El grafio presenta el formulario de registro de usuario en un contenedor, tomando en los temas de colores para la visualización. En la parte de abajo se tiene un enlace para ir al inicio de sesión.

En la pantalla se presenta el formulario de registro para la creación de una cuenta para la utilización del dashboad, se ingresan los daots de nombre, apellido, username en el que se valida que se único por usuario, el email de igual manera que se ingresa un único correo electrónico por usuario y dos boxes para el ingreso, y validación para el correcto ingreso de la contraseña para los usuarios. Cuando exista un error se presentará una alerta o mensaje en la parte interior del formulario indicando cual es el error para poder ingresar los datos para la creación del usuario.

Figura 66
Pantalla de inicio del dashboard

Nota. El grafico representa la visualización para el usuario de la pantalla inicio del dashboard.

En la pantalla de inicio, como se diseñó anteriormente en los mockups, se dividió con el SideBar una barra lateral para la navegación del usuario dentro del dashboard. En la parte superior izquierda se observa el nombre de la empresa. A continuación, tenemos un botón que permite retraer o expandir la barra de navegación, mostrando iconos más pequeños de las demás secciones del dashboard y desplegando los gráficos un 10% más hacia el lado izquierdo de manera responsiva.

Figura 67
Pantalla de inicio del dashboard contraída la barra de navegacion

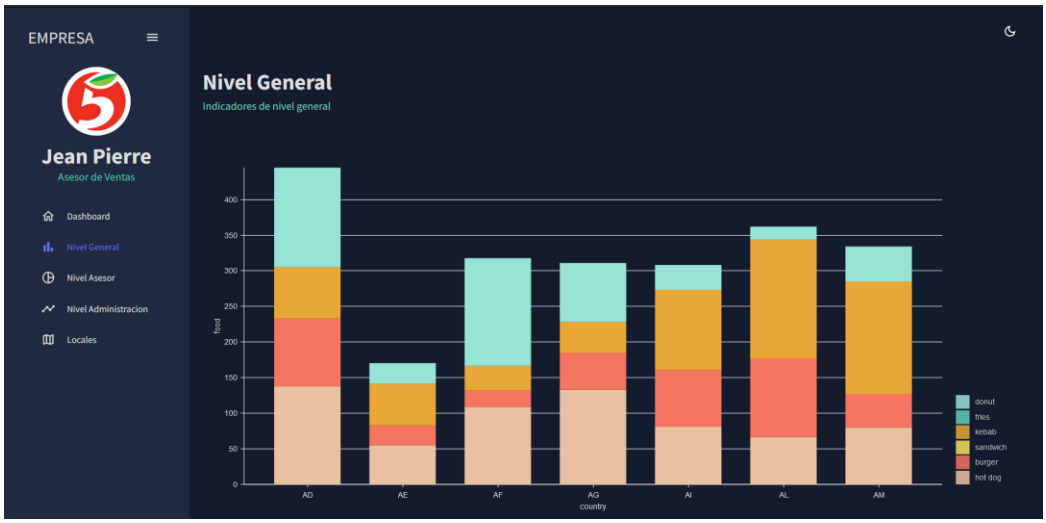
Nota. El grafico representa la parte visual de la pantalla de ‘inicio’ contraída de la barra de navegación.

En la primera figura, se observa el logo de la empresa, seguido del nombre del empleado y de su cargo. Después, se muestra una lista de diferentes secciones clasificadas en niveles. En la primera sección, que corresponde al inicio del dashboard, se presenta la pantalla actual con los niveles "General", "Asesor", "Jefe", "Administración" y "Locales".

En la sección "Inicio", en la esquina superior derecha, se encuentra un icono que nos permite cambiar el tema del dashboard según las necesidades del usuario. Luego se muestra el título de la sección y una pequeña bienvenida. Se tiene dos secciones en las que se dividen los gráficos de los KPIs. En la primera sección, se presentan cuatro cajas con diferentes indicadores del local, mostrando un gráfico del porcentaje de aumento o disminución según las metas de cada KPI. Además, se muestra un icono indicando si está por encima o debajo de la meta establecida.

En la segunda sección, tenemos un gráfico de líneas que indica las ventas totales por año y mes de los locales, proporcionando información sobre los diferentes locales y su estado para el usuario. Por último, se presenta una tabla con las facturas de mayor valor.

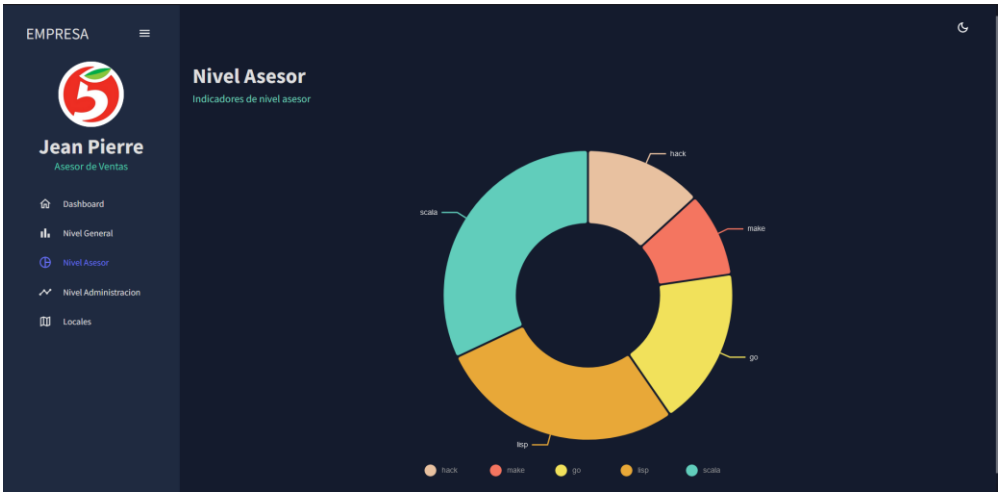
Figura 68
Pantalla del 'Nivel General' del dashboard



Nota. El grafico indica un diagrama de barras donde se observa la tasa de conversión por la categoría de los productos.

En la sección "Nivel General", disponemos de un gráfico de barras que muestra las categorías de productos con mayor venta en diferentes gráficos. Esto permite identificar rápidamente las categorías que necesitan más gestión para aumentar sus ventas.

Figura 69
Pantalla de 'Nivel Asesor' del dashboard'



Nota. El grafico indica un diagrama de pastel donde se observa las ventas por categoría del producto.

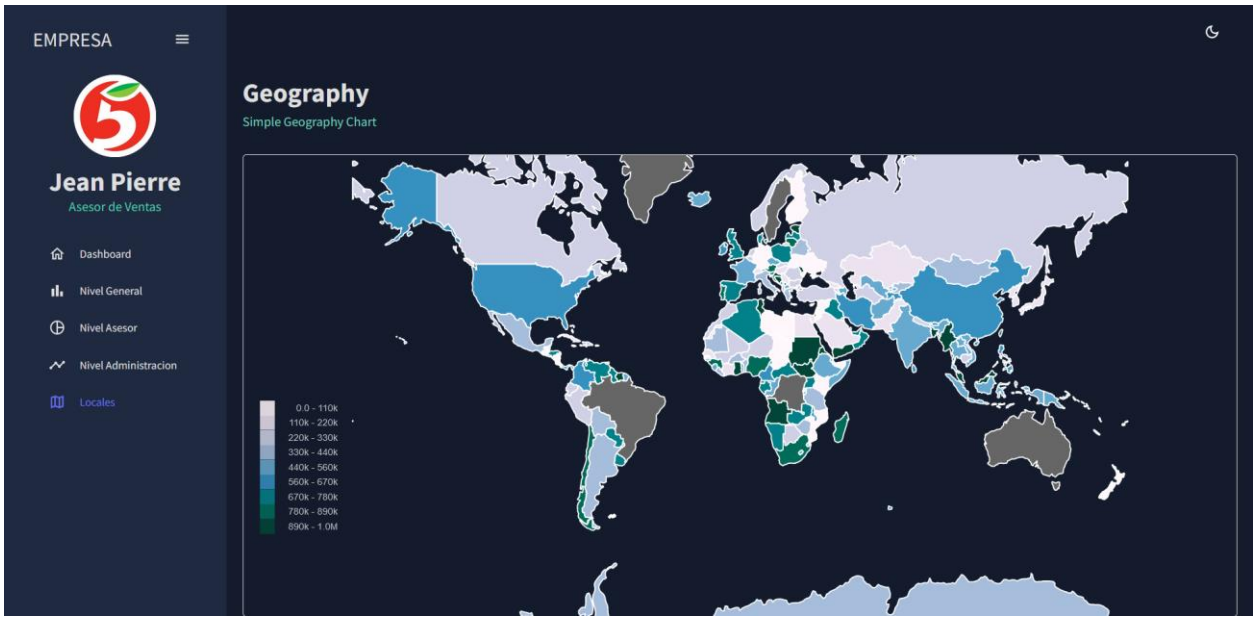
En la sección "Nivel Asesor", se despliega un gráfico de pastel que indica la tasa de conversión de las diferentes categorías de producto, para identificar aquellos productos que no se están transformando en ventas reales.

Figura 70
Pantalla de 'Nivel Administración' del dashboard

Nota. El grafico indica un diagramade líneas en el que se observa la frecuencia de compra por local.

En la sección "Nivel Administración", el gráfico de líneas representa las ventas por locales y categorías, ayudando al usuario en el coaching de los locales con baja venta al comparar rápidamente con otros locales.

Figura 71
Pantalla de 'Locales' del dashboard



Nota. El grafico indica un mapa de calor regional en el que se observa el estado de los locales con una escala que referencia las metas.

La sección "Locales" presenta un gráfico geográfico con mapa de calor para identificar el estado de las ventas en los locales sectorizados. Esto contribuye al análisis de un nivel alto identificando de manera visual y fácil. Además, el coaching se realizaría de acuerdo al estado del sector del local, identificando razones externas que puedan afectar al negocio, como desastres naturales, sociales, etc.

4.10 Pruebas de usuario

En el ámbito del desarrollo de software, las pruebas de usuario desempeñan un papel fundamental para garantizar la calidad y usabilidad de una aplicación. En el caso específico del desarrollo de un dashboard para la gestión de minoristas, se llevaron a cabo pruebas exhaustivas para evaluar su rendimiento y asegurar una experiencia óptima para los usuarios.

Estas pruebas de usuario se enfocaron en dos métricas clave: la tasa de éxito y la tasa de error. Estas métricas proporcionaron una visión cuantitativa sobre el desempeño del dashboard y su capacidad para cumplir con los objetivos establecidos.

Además, es importante destacar que las pruebas de usuario se llevaron a cabo con la participación de personas con experiencia en el sector retail y ventas con su conocimiento y experiencia específica, brindaron una perspectiva valiosa sobre las necesidades y requerimientos particulares de los minoristas en la gestión de sus operaciones.

Al involucrar a profesionales del sector en las pruebas, se pudo evaluar la adecuación del dashboard a los flujos de trabajo y procesos comerciales comunes en la industria minorista. Sus comentarios y sugerencias permitieron ajustar y adaptar el dashboard para que se alineara de manera óptima con las necesidades y expectativas de los usuarios finales.

La participación de personas con experiencia en el sector retail y ventas también ayudó a identificar posibles mejoras y funcionalidades adicionales que podrían beneficiar a los minoristas en su gestión diaria. Sus conocimientos sobre las mejores prácticas y desafíos específicos del sector contribuyeron a enriquecer el diseño y la funcionalidad del dashboard.

La tasa de éxito se definió como la capacidad del dashboard para satisfacer las necesidades y expectativas de los usuarios. Se evaluó la capacidad del dashboard para proporcionar información precisa y relevante, así como su facilidad de uso y navegación intuitiva. Además, se

consideraron los diferentes escenarios de uso y se realizaron pruebas en situaciones simuladas para evaluar la respuesta del sistema.

Por otro lado, la tasa de error se utilizó para medir la frecuencia y gravedad de los errores encontrados durante las pruebas. Se identificaron y registraron los errores y se evaluó su impacto en la experiencia del usuario y en la funcionalidad general del dashboard. Estos errores se clasificaron y priorizaron según su importancia, permitiendo así a los desarrolladores corregirlos de manera oportuna.

El análisis de las métricas de tasa de éxito y tasa de error proporcionó información valiosa para mejorar el dashboard y garantizar que cumpliera con los estándares de calidad esperados. Gracias a estas pruebas de usuario, se logró optimizar la interfaz, corregir errores identificados y proporcionar a los minoristas una herramienta eficiente y efectiva para gestionar sus operaciones diarias.

Se seleccionaron varias tareas para medir de manera adecuada los objetivos del desarrollo del dashboard, ya que cada tarea aborda una funcionalidad específica y esencial del sistema. Al evaluar múltiples aspectos del dashboard, se puede obtener una visión más completa de su rendimiento y usabilidad, así como identificar posibles áreas de mejora.

Las tareas elegidas están diseñadas para abarcar diferentes áreas clave del dashboard, como la creación de cuenta, el inicio de sesión, el análisis de la tasa de conversión por categoría, el análisis del ticket promedio por categoría, la frecuencia de compras por categoría, la identificación del crecimiento de ventas por región y la comparación del rendimiento de ventas entre dos períodos de tiempo. Cada una de estas tareas es fundamental para la gestión de minoristas y ofrece información valiosa para los usuarios.

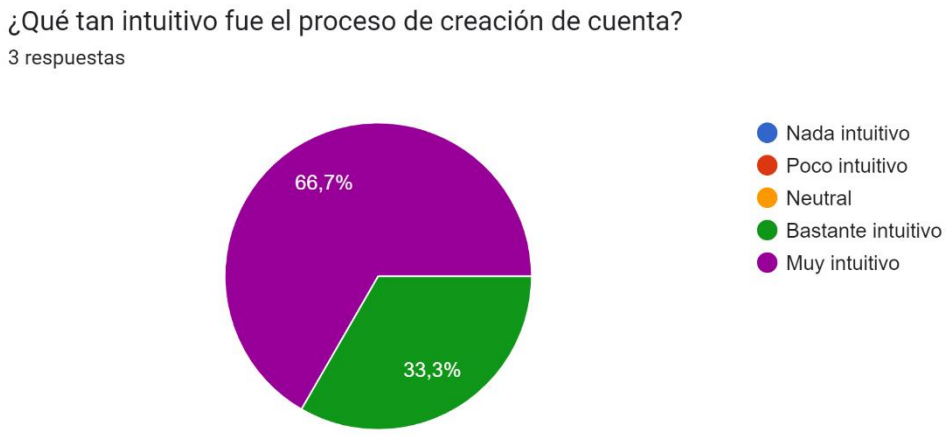
Al evaluar estas tareas, se puede medir la tasa de éxito y la tasa de error en cada una de ellas. La tasa de éxito indica cuántos usuarios lograron completar la tarea de manera exitosa,

mientras que la tasa de error muestra los problemas, dificultades o confusiones encontradas durante el proceso. Estas métricas permiten evaluar la efectividad y la eficiencia del dashboard en relación con las tareas específicas que los usuarios deben realizar.

Al recopilar los resultados de las pruebas de usuario, se obtiene una perspectiva realista de cómo los usuarios interactúan con el dashboard y cómo se cumplen los objetivos del desarrollo. Esta información es invaluable para identificar áreas de mejora, corregir posibles errores y realizar ajustes necesarios para optimizar la experiencia del usuario y garantizar que el dashboard cumpla con las necesidades y expectativas de los usuarios del sector retail y ventas.

El análisis de las pruebas de usuario, en conjunto con las métricas y los comentarios proporcionados por los participantes, arroja una visión exhaustiva sobre la eficacia y la usabilidad del dashboard de gestión de minoristas. Durante la evaluación, se llevaron a cabo diversas tareas fundamentales, como la creación de cuenta, el inicio de sesión y el análisis de distintas métricas de rendimiento, con el objetivo de medir de manera precisa los objetivos establecidos para el desarrollo del dashboard.

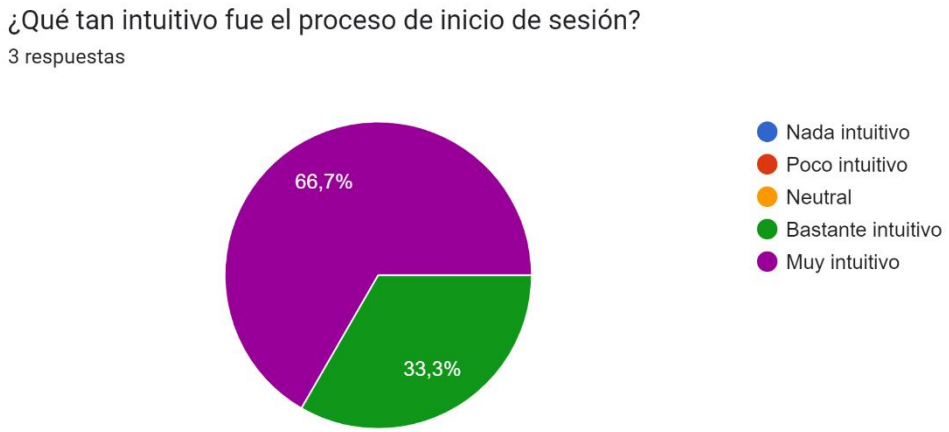
Figura 72
Grafico de tarea de creación de cuenta



Nota. El grafico indica el porcentaje de las respuestas de los usuarios en la tarea creación de cuenta.

La tarea de creación de cuenta se observó que dos de los tres usuarios lograron encontrar fácilmente la opción correspondiente en el dashboard, lo que indica una tasa de éxito del 66.7%. Estos usuarios también manifestaron que el proceso de creación de cuenta fue altamente intuitivo, rápido y eficiente. Estos resultados indican que el diseño e implementación de esta función específica han sido efectivos en términos de facilitar a los usuarios la creación de una cuenta en el dashboard.

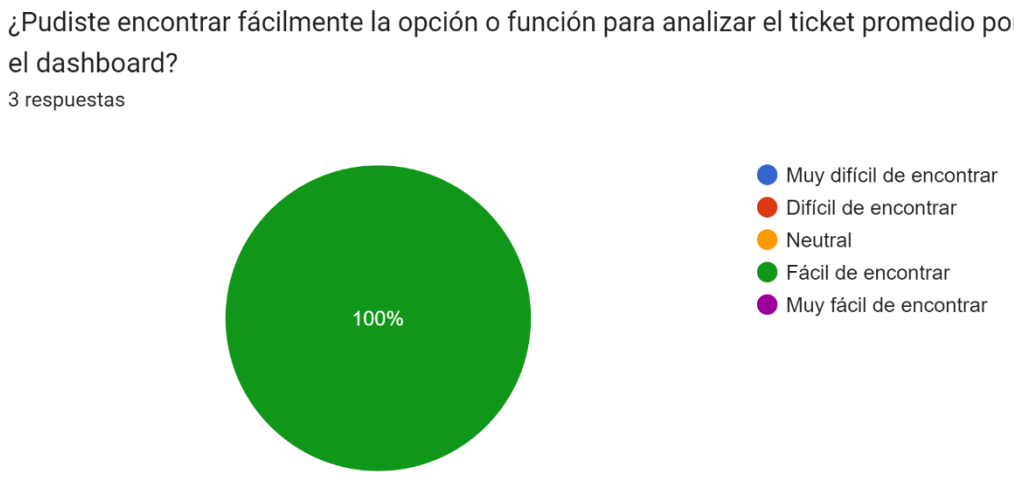
Figura 73
Grafico de tarea de inicio de sesión



Nota. El grafico indica el porcentaje de las respuestas de los usuarios en la tarea inicio de sesión.

El inicio de sesión, todos los usuarios evaluados lograron encontrar sin dificultad la opción para acceder al dashboard, lo que se tradujo en una tasa de éxito del 100%. Además, los participantes coincidieron en que el proceso de inicio de sesión fue altamente intuitivo, rápido y eficiente. Estos resultados resaltan que la interfaz del dashboard brinda una experiencia fluida y accesible a los usuarios al momento de ingresar al sistema.

Figura 74
Grafico de la tarea analizar ticket promedio por categoría



Nota. El grafico indica el porcentaje de las respuestas de los usuarios en tarea analizar ticket promedio por categoría.

El análisis para los KPIs como la tasa de conversión por categoría, el ticket promedio por categoría y la frecuencia de compras por categoría, se obtuvieron resultados positivos. Todos los usuarios lograron encontrar fácilmente las opciones correspondientes y evaluaron de manera favorable la intuición y claridad del proceso de análisis. Esto indica que el dashboard ofrece una funcionalidad adecuada para explorar y comprender estas métricas importantes en el contexto de la gestión de minoristas.

Los participantes también revelaron áreas de mejora clave que deben abordarse para optimizar aún más el dashboard. Uno de los aspectos señalados fue la necesidad de una presentación más detallada de los datos. Los usuarios expresaron su deseo de contar con información más específica, como desgloses adicionales de los datos y la capacidad de aplicar filtros más precisos. Estas sugerencias apuntan a mejorar la capacidad del dashboard para proporcionar una visión más completa y detallada de los aspectos relevantes para la gestión minorista. Destacaron la importancia de una mayor visualización gráfica en el dashboard

sugirieron la incorporación de más gráficos y representaciones visuales para facilitar la interpretación de la información. Una visualización gráfica más efectiva permitirá a los usuarios comprender rápidamente los patrones y las tendencias, lo que a su vez respaldará la toma de decisiones fundamentadas.

Estos comentarios ofrecen una guía concreta para futuras mejoras del desarrollo del dashboard. La inclusión de más detalles, como desgloses adicionales de los datos y opciones de filtrado más precisas, permitirá a los usuarios obtener una visión más completa y detallada de los aspectos relevantes para su gestión minorista. Además, una visualización gráfica mejorada, con la incorporación de más gráficos y representaciones visuales, ayudará a los usuarios a interpretar los datos de manera más efectiva.

CONCLUSIONES Y RECOMENDACIONES

El desarrollo de un dashboard para la gestión de minoristas, junto con los objetivos específicos de identificar los indicadores clave de rendimiento (KPIs) relevantes y utilizar técnicas de análisis de datos, busca mejorar la eficiencia y efectividad de las operaciones comerciales de una empresa minorista. Al alcanzar estos objetivos, se busca obtener una comprensión más profunda y una toma de decisiones informada sobre el rendimiento de la empresa en términos de ventas y otros aspectos fundamentales.

Se identificaron los KPIs relevantes para la gestión de minoristas; que son fundamentales para medir y evaluar el desempeño de la empresa. Al revisar los procesos de negocio y analizar las áreas de ventas, se pueden determinar los indicadores clave que reflejen mejor el éxito y el progreso de la empresa en el área de ventas. Estos KPIs incluyeron métricas como la tasa de conversión, ventas, ticket promedio y la frecuencia de compra. Al identificar y monitorear estos KPIs se pueden tomar decisiones estratégicas basadas en datos con el fin de mejorar la eficiencia y aumentar los resultados comerciales. En cuanto a las técnicas de análisis de datos empleadas, se destaca el análisis descriptivo como la técnica principal. El modelo descriptivo permitió obtener información detallada sobre las ventas y otras variables relevantes a través de la recolección, procesamiento y análisis de grandes volúmenes de datos. Esta técnica ayudaría a descubrir patrones, identificar tendencias y proporcionar una descripción completa del rendimiento de la empresa en relación con las ventas. Además, se utilizaron técnicas adicionales como la minería de datos y la visualización de datos para obtener información significativa y facilitar la comprensión de los resultados.

También, se concluye con la selección y análisis de las fuentes de datos resultan ser etapas cruciales en el desarrollo del dashboard destinado a la gestión de minoristas. Estas actividades permiten identificar y evaluar de manera exhaustiva las fuentes de información relevantes, asegurando su idoneidad para el propósito del proyecto. La aplicación de técnicas de minería de

datos y análisis exploratorio de datos brinda una perspectiva profunda de los datos, revelando patrones, tendencias y relaciones ocultas. Este enfoque resulta de vital importancia para obtener información valiosa que pueda ser empleada en la toma de decisiones y la mejora de la eficiencia en la gestión de minoristas.

Las pruebas de usuario y las métricas utilizadas han permitido evaluar la eficacia del dashboard de gestión de minoristas. Los resultados indican que el dashboard cumple con varios objetivos, como la creación de cuenta, el inicio de sesión y el análisis de métricas clave. Sin embargo, se han identificado áreas de mejora importantes a partir de los comentarios de los usuarios, como la necesidad de una presentación más detallada de los datos y una mayor incorporación de elementos visuales

Además, la integración y transformación de la fuente de datos coherente desempeña un papel esencial en la eficacia del dashboard al permitir un acceso y uso eficientes de los datos. Estos procesos derivaron en la consolidación de los diversos conjuntos de datos y la realización de tareas de limpieza, normalización y agregación, según lo necesario. La integración de los datos en una fuente unificada facilita la visualización y el análisis de la información en el dashboard, proporcionando una visión integral y completa de la gestión de minoristas.

Se aprovechó las capacidades de análisis y visualización de los datos relacionados con la gestión de minoristas. A su vez, se brindó la oportunidad de identificar oportunidades de mejora de toma decisiones fundamentadas y optimizar los procesos en el ámbito minorista.

Finalmente, Mediante la aplicación de buenas prácticas tal como SCRUM de desarrolló de software, se logró garantizar la calidad del dashboard, minimizando errores y fallas. Se llevaron a cabo pruebas para asegurar su correcto funcionamiento y se implementó un enfoque de desarrollo iterativo, permitiendo mejoras continúas basadas en la retroalimentación de los usuarios.

La usabilidad del dashboard fue una preocupación central durante el proceso de desarrollo. Se diseñó una interfaz intuitiva y fácil de usar, teniendo en cuenta las necesidades y habilidades de los expertos del área de ventas de retails. Los principios del diseño centrado en el usuario se aplicaron para asegurar la consistencia, la claridad y la comprensibilidad de la interfaz.

Se aplicaron técnicas efectivas de visualización de datos (grafico de barras, pastel, líneas y geográfico) para presentar la información de manera clara y comprensible. Se seleccionaron gráficos y elementos visuales apropiados para cada tipo de dato, con el fin de facilitar la interpretación y el análisis de la información por parte de los usuarios.

Se recomienda llevar a cabo un análisis de los procesos de negocio de los minoristas con el fin de identificar los indicadores clave de rendimiento relevantes. Esto implica trabajar en colaboración con los equipos de gestión y revisar detalladamente los datos históricos de ventas y el desempeño empresarial.

También, se recomienda en la elaboración de retails robustos utilizar técnicas de análisis de datos enfocadas en datos grandes, el uso de algoritmos de minería de datos y análisis estadístico para descubrir patrones y tendencias en los datos de ventas.

Es recomendable evaluar la calidad y coherencia de las fuentes de datos antes de integrar las fuentes de datos en el dashboard, resulta fundamental llevar a cabo una evaluación de la calidad y coherencia de los datos obtenidos. Es necesario verificar la precisión de estos, identificar posibles inconsistencias y realizar las transformaciones necesarias para garantizar la integridad de los datos en el dashboard.

Se sugiere mejorar el diseño y la funcionalidad del dashboard. Esto incluye proporcionar desgloses adicionales y opciones de filtrado más precisas, así como una mayor incorporación de gráficos y representaciones visuales para facilitar la interpretación de la información.

BIBLIOGRAFÍA

La bibliografía debe ser generada automáticamente en formato APA, solamente deben estar las citas que son utilizadas en la tesis.

Dashboard Ejemplos para el sitio web de comercio electrónico. (2019, enero 10). ClicData. https://www.clicdata.com/es/ejemplos/retail/

Grupo Bit. (s/f). KPIs para retail: ¿qué son, ¿cuáles son y para qué sirven? Grupobit.net. Recuperado el 1 de diciembre de 2022, de https://business-intelligence.grupobit.net/blog/kpi-para-retail-que-son-cuales-para-que-sirven

Qué problemas del sector retail soluciona SAP Business One. (s/f). Exxis-group.com. Recuperado el 1 de diciembre de 2022, de https://exxis-group.com/es-cl/problemas-sector-retail-soluciona-sap-business-one/

Martínez, M. T. (2022, octubre 15). 'Ecuador tiene bastante influencia americana, por eso estas marcas son muy bien recibidas aquí': así Old Navy abrió en Guayaquil, y en noviembre lo hará en Quito. El Universo. https://www.eluniverso.com/noticias/economia/old-navy-por-que-marca-americana-llego-a-ecuador-primera-tienda-guayaquil-nota/

Key Performance Indicators: Developing, Implementing, and Using Winning KPIs 4ta ed. (2013). David Parmenter.

Evolución del comercio electrónico: fases y futuro. (s/f). Beetrack.com. Recuperado el 7 de marzo de 2023, de https://www.beetrack.com/es/blog/evolucion-del-comercio-electronico.

UEES. (2022, marzo 9). La importancia de la inteligencia de negocios. UEES - Universidad Espíritu Santo; Universidad Espíritu Santo - UEES. https://uees.edu.ec/la-importancia-de-la-inteligencia-de-negocios/

Gracia, A. (2020, octubre 9). Analítica Retail: usar los datos para incrementar las ventas. Status2. https://status2.com/analitica-retail/

Élodie. (2021, octubre 26). ¿Cómo construir un dashboard analítico para el comercio de retail? DigDash. https://www.digdash.com/es/news-articles-es/perspectivas/como-construir-un-dashboard-analitico-para-el-comercio-de-retail/

KPI retail: qué son y qué ventajas ofrecen. (s/f). Beetrack.com. Recuperado el 24 de marzo de 2023, de https://www.beetrack.com/es/blog/kpi-retail-que-son-y-ventajas

Qué es React. Por qué usar React. (2016, octubre 4). Desarrolloweb.com. https://desarrolloweb.com/articulos/que-es-react-motivos-uso.html

Hurtado, J. S. (2021). Cómo funciona la Metodología Scrum: Qué es y cómo utilizarla. Thinking for Innovation. https://www.iebschool.com/blog/metodologia-scrum-agile-scrum/

Gil, B. (2016, agosto 2). CRISP-DM: La metodología para poner orden en los proyectos. Sngular. https://www.sngular.com/es/data-science-crisp-dm-metodologia/

Han, J., Pei, J., & Tong, H. (2022). Data mining: Concepts and techniques. Morgan Kaufmann.

Inman, D. (9 Junio, 2021). Retail sales to now exceed $4.44 trillion in 2021, as NRF revises annual forecast. NRF. Recuperado el 25 de abril de 2023, de https://nrf.com/media-center/press-releases/retail-sales-now-exceed-444-trillion-2021-nrf-revises-annual-forecast

Tenorio, A. (2018). Data mining techniques for customer relationship management in retail. In S. Bhatnagar (Ed.), Handbook of research on intelligent techniques and modeling applications in marketing analytics (pp. 163-180). IGI Global. https://doi.org/10.4018/978-1-5225-3245-4.ch008

McKinsey & Company. (2013). Big data: The next frontier for innovation, competition, and productivity. https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/big-data-the-next-frontier-for-innovation

Aslan, M. T. (2023). Customer Shopping Dataset - Retail Sales Data [Data set].

Pagés, S. (2018, marzo 18). *El mock-up en programación: Qué es, importancia y cómo hacer uno - Workana Blog*. Workana Blog - Este es el blog de Workana, la primera y más grande plataforma de freelancing de América Latina; Workana. https://blog.workana.com/uncategorized/importancia-mock-up-proyectos-it/

Juan de Assembler Institute. (2022, octubre 17). ¿Qué es Postman? Características y Ventajas. Assembler Institute. https://assemblerinstitute.com/blog/que-es-postman/

MongoDB: todo sobre la base de datos NoSQL orientada a documentos. (2022, abril 7). Formation Data Science | Datascientest.com. https://datascientest.com/es/mongodb-todo-sobre-la-base-de-datos-nosql-orientada-a-documentos

Comunícate con tu equipo en cualquier momento y en cualquier lugar. ¡Regístrate hoy mismo para obtener una cuenta de prueba gratuita! (s/f). Lucidchart. Recuperado el 29 de mayo de 2023, de https://www.lucidchart.com/pages/es

El diagrama de casos de uso en UML. (s/f). IONOS Digital Guide. Recuperado el 29 de mayo de 2023, de https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/diagrama-de-casos-de-uso/

¿Qué es un workflow y para qué le sirve a tu empresa? (2020, mayo 22). INTEGRADOC BPM; INTEGRADOC. https://www.integradoc.com/que-es-un-workflow/

Reporte de Investigación N°2 Agosto 2018 Ing-novación 7. (s/f). Core.ac.uk. Recuperado el 28 de junio de 2023, de https://core.ac.uk/download/pdf/47264961.pdf

Marquez, A. (2022, febrero 17). Como la metrica tasa de éxito puede mejorar la usabilidad de tus sistemas. Tester moderno. https://www.testermoderno.com/como-la-metrica-tasa-de-exito-puede-mejorar-la-usabilidad-de-tus-sistemas/

ANEXOS

ANEXO A: Tabla de sprints del proyecto

Sprint	Descripción	Duración
Sprint 1	Investigación preliminar	1 semanas
Sprint 2	Recolección de datos	1 semanas
Sprint 3	Análisis de datos	1 semanas
Sprint 4	Desarrollo del modelo de minería de datos	2 semanas

ANEXO B: Codigo para el preprocesamiento del dataset

```
!pip install mlxtend

from google.colab import files

uploaded_file = files.upload()

import pandas as pd

import numpy as np

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent_patterns import apriori, association_rules

import datetime

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read_csv("customer_shopping_data.csv")

print(df.dtypes)

print(df.head())

df['quantity'] = df['quantity'].astype(str)
```

```
df['anio'] = pd.to_datetime(df['invoice_date']).dt.year

df['mes'] = pd.to_datetime(df['invoice_date']).dt.month

df['dia'] = pd.to_datetime(df['invoice_date']).dt.day

df['anio'] = df['anio'].astype(str)

df['mes'] = df['mes'].astype(str)

df['dia'] = df['dia'].astype(str)

print(df.dtypes)

# Seleccionar las columnas relevantes

df = df[['invoice_no', 'customer_id', 'category', 'quantity', 'anio', 'mes', 'dia']]

print(df.head())

print(df.dtypes)
```

ANEXO C: Código para el modelo de reglas de asociación Apriori

El siguiente código muestra la implementación del algoritmo Apriori para generar un modelo de reglas de asociación. Se utilizó la biblioteca "mlxtend" para realizar esta tarea.

```
from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent_patterns import apriori, association_rules

transacciones = df.groupby(['customer_id'])[['category','quantity']].apply(lambda x: [item for sublist in x.values.tolist() for item in sublist]).tolist()

te = TransactionEncoder()

te_transacciones = te.fit_transform(transacciones)

df_transacciones = pd.DataFrame(te_transacciones, columns=te.columns_)

frequent_itemsets = apriori(df_transacciones, min_support=0.001, use_colnames=True)

rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1)
```

```
print(rules)
```

ANEXO D: Tabla de reglas de asociación

N	Antecedentes	Consecuentes	Antecedentes soporte
0	(Food & Beverage)	(1)	0.148567
1	(1)	(Food & Beverage)	0.198749
2	(1)	(Souvenir)	0.198749
3	(Souvenir)	(1)	0.050263
4	(Books)	(2)	0.050082
5	(2)	(Books)	0.199363
6	(Clothing)	(2)	0.346753
7	(2)	(Clothing)	0.199363
8	(Shoes)	(2)	0.100888
9	(2)	(Shoes)	0.199363
10	(Technology)	(2)	0.050233
11	(2)	(Technology)	0.199363
12	(Cosmetics)	(3)	0.151794
13	(3)	(Cosmetics)	0.202590
14	(3)	(Souvenir)	0.202590
15	(Souvenir)	(3)	0.050263
16	(3)	(Toys)	0.202590
17	(Toys)	(3)	0.101421
18	(4)	(Clothing)	0.198307
19	(Clothing)	(4)	0.346753
20	(Cosmetics)	(4)	0.151794
21	(4)	(Cosmetics)	0.198307
22	(Shoes)	(4)	0.100888
23	(4)	(Shoes)	0.198307
24	(Technology)	(4)	0.050233
25	(4)	(Technology)	0.198307
26	(4)	(Toys)	0.198307
27	(Toys)	(4)	0.101421
28	(5)	(Books)	0.200991
29	(Books)	(5)	0.050082
30	(5)	(Clothing)	0.200991
31	(Clothing)	(5)	0.346753
32	(5)	(Cosmetics)	0.200991
33	(Cosmetics)	(5)	0.151794
34	(Food & Beverage)	(5)	0.148567
35	(5)	(Food & Beverage)	0.200991

ANEXO E: Contenido de librerías usadas en el proyecto

Una vez que se tiene el proyecto de React configurado, se puede instalar las librerías adicionales necesarias para la aplicación. A continuación, se mencionan las librerías mencionadas:

- **Material-UI**: Una popular librería de componentes de interfaz de usuario para React. Para instalar Material-UI, ejecuta el siguiente comando en tu terminal:

- **Nivo Charts**: Una librería de gráficos interactivos para React. Para instalar Nivo Charts, ejecuta el siguiente comando:

- **Formik**: Una librería de gestión de formularios para React. Para instalar Formik, ejecuta el siguiente comando:

- **Yup Validation**: Una librería de validación de formularios para React. Para instalar Yup Validation, ejecuta el siguiente comando:

React Pro Sidebar**: Una librería para crear barras laterales (sidebars) en aplicaciones de React. Para instalar React Pro Sidebar, ejecuta el siguiente comando:

- **Google Fonts**: Una librería que permite utilizar fuentes de Google en tu aplicación de React. Para utilizar Google Fonts, agrega el siguiente enlace al archivo HTML de tu aplicación, en la sección `<head>`:

La fuente de Google Fonts que se utilizara de roboto. Puedes encontrar el nombre de la fuente y más opciones en el sitio web de Google Fonts.

Una vez que se haya instalado todas las librerías adicionales, se puede importar en tu proyecto de React y comenzar a utilizarlas en el código.

Cuando se crea un nuevo proyecto de React utilizando la herramienta "create-react-app", se generan automáticamente varias carpetas con una estructura predefinida. Estas carpetas tienen

un propósito específico y contienen los archivos necesarios para desarrollar una aplicación de React.